





University of
Stavanger

FACULTY OF SCIENCE AND TECHNOLOGY

MASTER'S THESIS

Study programme/specialisation: Petroleum Engineering	Spring / Autumn semester, 2019 Open / Restricted access
Author: Erik Andreas Løken Jens Løkkevik	  (signature of author)
Programme coordinator: Karina Sanni Supervisor(s): Dan Sui	
Title of master's thesis: Optimization of an Intelligent Autonomous Drilling Rig: Testing and Implementation of Machine Learning and Control Algorithms for Formation Classification, Downhole Vibrations Management and Directional Drilling	
Credits (ECTS): 30	
Keywords: Drilling Automation Drill String Dynamics Vibrations Machine Learning ROP Optimization Fault Detection Drillbotics®	Number of pages: 291 + supplemental material/other: 24 Stavanger, 14. Juni 2019

Abstract

In recent years, considerable resources have been invested to explore applications for- and to exploit the vast amount of data that gets collected during exploration, drilling and production of oil and gas. Such data will potentially become a game changer for the industry in terms of reduced costs through improved operational efficiency and fewer accidents, improved HSE through strengthened situational awareness, ensured optimal placement of wells, less wear on equipment and so on. While machine learning algorithms have been around for decades, it is only in the last five to ten years that increased computational power along with heavily digitalized control- and monitoring systems have been made available. Considering the state of art technology that exists today and the significant resources that are being invested into the technology of tomorrow, the idea of intelligent and fully automated machinery on the drill floor that is capable of consistently selecting the best decisions or predictions based on the information available and providing the driller and operator with such recommendations, becomes closer to a reality every day.

This thesis is the result of research carried out on the topic of drilling automation. Its basis has been improvements and upgrades conducted on a laboratory-scale drilling rig developed at the University of Stavanger, as part of the multi-disciplinary project; UiS Drillbotics. Main contribution of the thesis is a study on how machine learning can be used to develop models that are capable of accurately predicting what rock formation is being drilled using an autonomous control system, along with detecting some common drilling incidents in real-time on the laboratory rig. Methodology is also applied to field data from the Volve field. Furthermore, research and implementation of search algorithms to ensure optimal drilling speed (ROP), safety to personnel and environment (HSE), and efficiency along with a digitalized drilling program for directional drilling, gets presented. Finally, rig upgrades for directional drilling and research into downhole sensors that get used in a closed-loop steering model is elaborated on.

Acknowledgement

Taking on this topic and project has been incredibly rewarding. The challenge of working with machine learning and autonomous systems has encouraged us to explore new areas of research, not covered in our study. The project has also given us the opportunity to get hands on experience with machinery, component design, development and manufacturing as well as rather complex programming. Of equal importance, being able to collaborate with skilled students and industrial partners in a multidisciplinary environment has been particularly rewarding. The project was initiated by Professor Dan Sui at the University of Stavanger in 2016, and for three years, students from several departments at the University have designed, developed and further improved an autonomous rig that gets used for research as well as participation in the international Drillbotics[®] competition. In addition to this Master's thesis, three students have written Bachelor's theses.

We would like to express our gratitude towards our supervisor Professor Dan Sui at UiS. She has been of great guidance throughout the process of our work giving valuable feedback to our research and thesis. We would also like to thank Suranga Chaminda Hemba Geekiyanage and Ekaterina Wiktorski for several interesting and encouraging discussions.

Finally, we would like to thank the UiS Drillbotics team for a great collaboration and the Department of Energy and Petroleum Engineering (IEP) at the University of Stavanger for both technical and financial support.

Table of Contents

1	Introduction	1
1.1	Research Problem	1
1.2	Background of Study	3
1.3	Control System Architecture	4
2	Experimental Setup	6
2.1	Drilling Rig System	6
2.1.1	Rotation System	6
2.1.2	Hoisting System	7
2.1.3	Circulation System	9
2.1.4	Drill String Assembly	9
2.2	Additions to facilitate for directional drilling	11
2.2.1	Downhole motor (pneumatic)	12
2.2.2	2-axis Actuator System	14
2.2.3	BHA components	17
2.3	Rig Sensors	19
2.4	Downhole Sensor Sub	19
2.4.1	Mechanical design	19
2.4.2	Mechanical design - Stress simulation	21
2.4.3	Sensor Package	21
2.4.4	Final Design	23
2.5	Calibration of systems	24
2.5.1	Pneumatic Motor	24
2.5.2	Top Drive	27
2.5.3	WOB Control	28
2.6	Downhole measurements	31
2.6.1	Inclination and azimuth calculations	31
2.6.2	Calibration of sensors	33
2.7	Downhole Position Tracking	34
2.8	Software Architecture	35

2.8.1	gRPC	36
2.8.2	API - OPC UA	37
2.9	Graphical User Interfaces (GUIs)	38
3	Theory	40
3.1	Data Management	40
3.1.1	Data Mining	40
3.1.2	Data Quality	41
3.1.3	Some challenges concerning data aggregating	44
3.1.4	Importance of utilizing a database for data storage	45
3.1.5	Downsampling the data	47
3.1.6	Describing the dataset	48
3.1.7	Noise Reduction Methods	50
3.1.8	Normalization and standardization of the data	54
3.2	Machine Learning Theory	57
3.2.1	A Short Introduction to Supervised Learning	57
3.2.2	A Short Introduction to Unsupervised Learning	58
3.2.3	Training and Cross Validation	60
3.3	Supervised Machine Learning Models	61
3.3.1	Selecting the most optimal model	61
3.3.2	Multilayer Feed Neural Network and Back-propagation	66
3.3.3	Support Vector Machine	69
3.3.4	Decision Tree	71
3.3.5	Gradient Boosting	73
3.3.6	Random Forest	75
3.3.7	K-Nearest Neighbor	76
3.3.8	Bayesian Classification	77
3.3.9	TPOT Algorithm	78
3.4	Unsupervised Machine Learning Models	80
3.4.1	K-Means Clustering	80
3.4.2	Density Based Spatial Clustering of Applications with Noise	82
3.5	Drilling Theory	83
3.5.1	Drill String Vibrations	83
3.5.2	Directional Drilling	87
3.5.3	Buckling Models	89
4	Data Preparation	91
4.1	Laboratory data preparation for rock / formation classification	91
4.1.1	Data source	91

4.1.2	Data concatenation	93
4.1.3	Data labeling	94
4.1.4	Describing the raw data	94
4.2	Data collection for classification of laboratory rig operations	97
4.2.1	Data source	97
4.2.2	Data concatenation	97
4.2.3	Data labeling	97
4.2.4	Describing the data	97
4.3	Surface Data collection for drilling incident classification	99
4.3.1	Surface data for vibration classification	99
4.3.2	Normal pressure vs leak and overpressure	101
4.3.3	Rotating pipe vs stuck pipe	103
4.3.4	Normal drilling vs drill string twist off	105
4.4	Downhole data for vibration classification	108
4.5	Possible errors present in laboratory data	110
4.6	Volve data	111
4.6.1	Volve data aggregated to classify rock formations	111
4.6.2	Volve data aggregated to classify rig operations	113
4.7	Field data challenges	116
5	Data Quality Improvement	118
5.1	Down sampling experimental data	118
5.2	Removing duplicates	119
5.3	Removing Missing data	119
5.4	Normalizing the data	119
5.5	Outlier removal using IQR method	121
5.6	Four-plots for WOB and Torque	123
6	Feature Engineering and Optimization	126
6.1	Drilling Feature Engineering	126
6.1.1	Natural features	126
6.1.2	Artificial Feature Engineering	128
6.1.3	Drilling Features constructed after processing	129
6.2	Feature Selection	137
6.2.1	With outliers kept in the dataset	137
6.2.2	With no outliers in the dataset	141
6.3	Feature Extraction	144

7	Model Development - Sensitivity Study	146
7.1	Sensitivity study objective	146
7.2	Study cases	150
7.2.1	Laboratory formation classification	150
7.2.2	Volve formation classification	168
7.2.3	Laboratory rig operations	186
7.2.4	Volve rig operations	191
7.2.5	Laboratory pressure cases	196
7.2.6	Laboratory vibration cases (surface data)	200
7.2.7	Laboratory vibration cases (downhole data) with downhole motor	205
7.2.8	Laboratory stuck-pipe cases	211
7.2.9	Laboratory twist off cases	215
7.3	Summary of preliminary classifications results	219
8	Validation	221
8.1	Rock Classification	221
8.1.1	Task a. Laboratory Experimental data	221
8.1.2	Laboratory rig voting system	232
8.1.3	Task b. Volve field data	234
8.1.4	Formation classification - recommendations	240
8.2	Drilling Rig Operations Classification	242
8.2.1	Task c. Laboratory Experimental data	242
8.2.2	Task d. Volve field data	243
8.2.3	Rig operations - recommendations	244
8.3	Drilling Incident Classification	245
8.3.1	Task e. Pressure Incident Detection	245
8.3.2	Task f. Surface Drilling Vibrations Detection	246
8.3.3	Task g. Downhole Vibrations	247
8.3.4	Task h. Stuck Pipe Incident Detection	250
8.3.5	Task i. Twist off Incident Detection	252
8.3.6	Incident classification - recommendations	253
8.4	Limitations using machine learning	254
9	Autonomous Drilling	256
9.1	Search algorithms	257
9.1.1	ROP Optimization Background	257
9.1.2	Various search algorithms	259
9.2	Implemented algorithms	263

9.2.1	Column-Row search (Implemented in 2018)	263
9.2.2	Gradient Descent (Implemented in 2019)	264
9.2.3	Triggers that can reinitiate search algorithm	268
9.3	Digital Detailed Operating Procedure	268
9.4	Downhole closed-loop steering	271
9.5	Drilling Incident Detection	273
9.6	Rig performance	275
9.6.1	Experiment 1: Inclined well section	275
9.6.2	Experiment 2: Vertical well section	277
9.6.3	Experiment 3: Deviation well with WOB 5 to 20 kg	278
9.6.4	Experiment 4: Pilot hole section 0 - 166 mm MD. Inclina- tion well from 109 mm to 600 mm MD.	279
9.6.5	Experiment 5: Increasing cross-over OD to 30.75 mm (up from 20)	280
10	Conclusion and Future Research	282
10.1	Discussion of results and end state achieved	282
10.1.1	Machine Learning	282
10.1.2	Control System and Control Algorithms	283
10.1.3	Mechanical	284
10.2	Future recommendations	285
10.2.1	Machine Learning	285
10.2.2	Control System and Control Algorithms	286
10.2.3	Mechanical work	287
A	Sensor Sub Stress Simulation	292
B	Manual Downhole Position Tracking Concept	297
C	Autonomous Search Algorithm Code	299
D	Active Steering Code	301
E	Stuck Pipe Code	304
F	Autonomous Drilling Program Code	305
G	Downhole Vibration Classification Code	309
H	Downhole Position Tracking Code	311

List of Figures

1.1	3D model of Volve field	3
1.2	Laboratory drilling system at the University of Stavanger.	4
2.1	Rotational system	7
2.2	Hoisting system	8
2.3	Circulation System	9
2.4	BHA components: pipe, sensor sub, knuckle joint, downhole motor and bit	10
2.5	Laboratory drilling rig schematic to enable for directional drilling	11
2.6	170 watt drilling motor specifications	12
2.7	Pneumatic output characteristic	13
2.8	Pneumatic output characteristic	14
2.9	Actuator system concept	15
2.10	Assembled actuator system for directional drilling	16
2.11	Bit entering the whipstock	16
2.12	Illustration and design sketch showing the whipstock design	17
2.13	Knuckle joint design	18
2.14	Rig sensors	19
2.15	Dimensions of the sensor sub	20
2.16	Sensor sub CAD drawing	20
2.17	Cross-sectional view of sensor house	21
2.18	FLORA 9-DOF Accelerometer/Gyroscope/Magnetometer	22
2.19	Adafruit Trinket M0	22
2.20	Drilled through thread tap and refurbished threads	23
2.21	Sensor sub end result - after sand blasting and assembling	24
2.22	High frequency camera used to measure bit revolutions	25
2.23	Vertical displacement of a tracked pixel	25
2.24	Frequency of which pixel occurs	26
2.25	Bit RPMs corrected for torque	27
2.26	PID Controller block diagram	28

2.27	Proof of concept, using simple tracker	35
2.28	Concept illustration of the control system	36
2.29	Implemented control system illustrating layers and dataflow	37
2.30	API	38
2.31	Rig performance graphical user interface	38
2.32	Downhole position tracking graphical user interface	39
3.1	Four Plot	43
3.2	User panel for the developed database at UiS	47
3.3	Python function to describe the dataset	48
3.4	MATLAB pairplot of data from five variables (features)	49
3.5	Box plots and heat maps for visualization of the data	50
3.6	Data that is treated as invalid data gets transformed to NaN	51
3.7	Rows of data that contain NaN is removed	51
3.8	Rows containing maximum and minimum values get added	56
3.9	Data is normalized using LFS	56
3.10	Work flow illustration to develop a machine learning model	57
3.11	Characteristics of different machine learning algorithms	62
3.12	Scikit learn and SAS cheat sheets to select an appropriate algorithm	63
3.13	Grid Search CV function from the Scikit Learn software	64
3.14	Classification report providing feedback on the model performance	65
3.15	Feed forward neural network illustration	67
3.16	Example of an overfitted model	69
3.17	Polynomial kernel	70
3.18	Decision Tree concept illustration	72
3.19	K-Nearest Neighbor	76
3.20	TPOT machine learning pipeline	78
3.21	TPOT optimization progress	79
3.22	K-Means Clustering: raw data before clustering	81
3.23	K-Means Clustering - after clustering	81
3.24	Density Based Spatial Clustering of Applications with Noise	83
3.25	Drill String Vibrations	84
3.26	Drill String System	86
3.27	Dogleg angle	88
3.28	Buckling modes	89
3.29	Non-rotating buckling	90
4.1	Rock samples drilled for collecting experimental drilling data	92

4.2	Process of labeling rock formation data	94
4.3	Description of the original data collected for rock classification models	95
4.4	Plot representing the dataset containing natural features only . .	95
4.5	Seaborn pairplot representing raw data - rock formation classification	96
4.6	Description of dataset containing raw observations for rig operations	98
4.7	Plot representing the raw data from laboratory rig operations . .	98
4.8	Seaborn pairplot representing rig operation raw data	99
4.9	Description of the raw data for vibrations using surface sensors . .	100
4.10	Plot of the raw data for vibrations using surface sensors	100
4.11	Seaborn pairplot - normal vs moderately high vibrations case . . .	101
4.12	Pressure loss and overpressure cases for the mud system	102
4.13	Description of raw data for pressure cases	102
4.14	Pairplot of pressure and WOB features for pressure cases	103
4.15	Normal drilling vs stuck pipe cases for stuck pipe classification . .	104
4.16	Description of raw data to later develop a stuck pipe model	104
4.17	Pair plot of raw data gathered in order to develop a stuck pipe model	105
4.18	Transition from normal drilling to a twist off. Cement drilled at high RPM and WOB	106
4.19	Description of natural features in twist off experiment	107
4.20	Pairplot of raw data collected for twist off model	107
4.21	Description of raw data used in downhole vibration study	109
4.22	Pairplot of raw data collected for downhole vibration level model .	109
4.23	Description of the Volve data for formation classification	112
4.24	Plot of features from Volve field data	112
4.25	Pairplot of features from Volve field data	113
4.26	Description of the Volve data for rig operation classification . . .	114
4.27	Plot of features from Volve field data	115
4.28	Pairplot of features from Volve field data	116
5.1	Normalizing data using MinMaxScaler	120
5.2	Renaming the features in Jupyter Notebook	120
5.3	Changing normalized labels back to their original values	120
5.4	Feature sorting – outlier removal	121
5.5	Quartile identification for IQR method	121
5.6	Illustration of how IQR gets calculated	121
5.7	IQR – upper and lower limit	122
5.8	IQR - upper and lower limits for rock formation classification . . .	122
5.9	Setting outlying data to zero	123

5.10	Number of identified outliers that have been converted to NaN . . .	123
5.11	Four-plot for WOB before the data has been pre-processed	124
5.12	Four-plot – WOB, processed data	124
5.13	Four-plot – Torque, raw data	125
5.14	Four-plot – Torque, processed data	125
6.1	Algorithm used to calculate mean-depth	128
6.2	Algorithm to calculate ROP	129
6.3	MSE different between soft and hard formation	131
6.4	Algorithm to calculate RPM*WOB feature	133
6.5	Algorithm to calculate RPM*WOB/ROP feature	134
6.6	Algorithm to calculate RPM ² feature	134
6.7	Algorithm to calculate WOB ² feature	134
6.8	Algorithm to calculate WOB/RPM feature	134
6.9	Algorithm to calculate standard deviation	135
6.10	Algorithm to calculate mean value	135
6.11	Algorithm to calculate median value	135
6.12	Algorithm to calculate maximum value	135
6.13	Algorithm to calculate maximum value	136
6.14	Algorithm to calculate average divided by standard deviation . . .	136
6.15	Algorithm to calculate median divided by standard deviation . . .	136
6.16	Algorithm to calculate maximum value	136
6.17	Result from feature evaluation before outliers are removed	138
6.18	Feature dimensionality reduction using PCA	139
6.19	Mean and variance ratio for original features	139
6.20	Feature dimensionality reduction using PCA when z-scaled	140
6.21	Importance of principal components created	141
6.22	Importance of principal components, z-scaled	141
6.23	ExtraTreesClassifier results after performing IQR	142
6.24	PCA- feature dimensionality reduction and feature importance after using IQR	143
6.25	Real-time flow chart as integrated in Python	144
7.1	Area Under Curve	147
7.42	Volve data rock classification feature importance	168
7.83	Laboratory operation classification feature importance	186
7.89	Volve rig operation classification feature importance	191
7.95	Pressure cases feature importance	196

7.101	Vibration classification feature importance	200
7.106	Unsupervised K-Means Clustering	203
7.108	Downhole vibration classification feature importance	206
7.115	Stuck pipe classification feature importance	211
7.121	Twist off classification feature importance	215
8.1	Laboratory formation classification with all features, models trained for six formations	222
8.2	Median-filtered laboratory formation classification with all features, models trained for six formations	223
8.3	Laboratory formation classification with only six features, models trained for six formations	224
8.4	Median-filtered laboratory formation classification with only six features, models trained for six formations	225
8.5	Laboratory formation classification with all features, models trained for 3 formations	226
8.6	Median-filtered laboratory formation classification with all features, models trained for 3 formations	227
8.7	Laboratory formation classification with only six features, models trained for 3 formations	228
8.8	Median-filtered laboratory formation classification with only six features, models trained for 3 formations	229
8.9	Raw and median-filtered laboratory formation classification using a granite test set	230
8.10	Raw and median-filtered laboratory formation classification using a granite test set	231
8.12	Volve formation classification without removing outliers and using all features	235
8.13	Median-filtered Volve formation classification without removing outliers with all features	236
8.14	Volve formation classification removing outliers using IQR with all features	237
8.15	Median-filtered Volve formation classification removing outliers using IQR with all features	238
8.16	Volve formation classification removing outliers using IQR with only six features	239
8.17	Median-filtered Volve formation classification removing outliers using IQR with only six features	240

8.18	Laboratory rig operations classification with raw- and median-filtered prediction	242
8.19	Volve rig operations classification	244
8.20	Leak and overpressure classification from original dataset	246
8.21	Vibration classification using surface data	247
8.22	Classifying vibration levels in the original dataset	248
8.24	Downhole Vibration Classification as implemented in the drilling system.	249
8.23	Classifying vibration levels in a freshly acquired dataset	249
8.25	Classifying stuck pipe in the original dataset	251
8.26	Classifying stuck pipe in the original dataset without removing invalid data	252
8.27	Twist off detection from original dataset	253
9.1	Simplified logic of the control system illustrating the processes . .	256
9.2	Illustration of ROP response from varying drilling parameters . .	258
9.3	Shortest path is the gradient directly towards the goal state of the machine	259
9.4	Gradient Descent	260
9.5	Unimodal function	261
9.6	Illustration of hill climb search	263
9.7	Snapshot of the code implemented in 2018	264
9.8	State space constraints and ROP contours from Dunlop et al. . .	266
9.9	Gradient descent illustration	267
9.10	DDOP Phases for Autonomous Drilling	269
9.12	Illustration of how the machine uses a look up table to confirm or deny whether sufficient build is achieved	272
9.13	Experiment 1: Well log, downhole motor drilling of 70 mmTVD .	275
9.14	Well profile using 7 degrees whipstock in homogeneous cement . .	276
9.15	Experiment 2: Well log, downhole motor drilling of 170 mmTVD .	277
9.16	Experiment 3: Well log, downhole motor drilling of 600 mmTVD .	278
9.17	Experiment 4: Well log, downhole motor drilling of 166 mmTVD pilot hole	279
9.18	Experiment 4: Well log, downhole motor drilling of 600 mmTVD .	280
A.1	Momentum load case simulation from Fusion 360	292
A.2	Results from momentum load case simulation in Fusion 360	293
A.3	WOB load case simulation from Fusion 360	293

A.4	Results from WOB load case simulation in Fusion 360	294
A.5	Overpull load case simulation from Fusion 360	294
A.6	Results from overpull load case simulation in Fusion 360	295
A.7	Pressure load case simulation from Fusion 360	295
A.8	Results from pressure load case simulation in Fusion 360	296
A.9	Buckling load case simulation from Fusion 360	296
B.1	Tracking concept for downhole position, part 1	297
B.2	Tracking concept for downhole position, part 2	298

List of symbols

δ	the tolerance specified for when gradient search shall terminate
η_k	step size used for steepest descent
\mathbf{g}_k	gradient used in steepest descent
\mathbf{x}_{k+1}	minimal point used in steepest descent
\mathbf{x}_k	current position used in steepest descent
f	frequency
F_N	integer in fibonacci sequence
fr	framerate
g	gravity
J	sum of squared errors
K_d	controller gain for derivative controller
K_i	controller gain for integral controller
K_p	controller gain for proportional controller
$u(t)$	error
u_0	bias
α	positivity constraint
\bar{x}	mean value
β	buoyancy factor
χ	output value of neuron

η	learning rate
Γ	mapping function
λ	learning rate for back-propagation
μ	coefficient of sliding friction
μ	true mean value
ω_{ij}	weight coefficient
σ	standard deviation
τ	torque
ε	neighbourhood radius in DBSCAN
ϑ	threshold coefficient (referred to as bias)
ξ	neuron potential
A_{bit}	area of bit
D_{bit}	drill bit diameter
E	objective function to determine the sum of squared error between predicted and known output in a supervised neural network
E_s	specific energy
L	maximization equation
m_i, c_i, k_i	mass-, damping- and spring constants for a spring-mass system
G	gini index
Q	flow rate

Common Abbreviations

A - *Azimuth*

AI - *Artificial Intelligence*

ANN - *Artificial Neural Network*

API - *Application Programming Interface*

ARI - *Adjusted Rand Index*

AUC - *Area Under Curve*

BA - *Bit Aggressiveness*

BHA - *Bottom Hole Assembly*

BHP - *Bottom Hole Pressure*

CPU - *Central Processing Unit*

CSS - *Cascading Style Sheets*

CSV - *Comma Separated Values*

CV - *Cross-Validation*

DAQ - *Data Acquisition*

DB - *Database*

DDS - *Data Distribution Service*

DT - *Decision Tree*

DBSCAN - *Density-based spatial clustering of applications with noise*

DDOP - *Digital Detailed Operating Procedure*

DL - *Dogleg angle, or Deep Learning*

DLS - *Dogleg Severity*

DM - *Data Mining*

DOC - *Depth of Cut*

DPS - *Degrees Per Second*

DT - *Decision Tree*

ECD - *Equivalent Drilling Density*

FSM - *Finite State Machine*

BG - *Gradient Boosting*
gRPC - *google Remote Procedure Calls*
GUI - *Graphical User Interface*
HSE - *Health, Safety and Environment*
HTML - *HyperText Markup Language*
I - *Inclination*
ID - *Inner Diameter*
IDE - *Integrated Development Environment*
IEP - *Department of Energy and Petroleum Engineering*
IQR - *InterQuartile Range*
K-NN - *K-Nearest Neighbor*
KOP - *Kick-Off Point*
LC - *Load Cell*
LFS - *Linear Feature Scaling*
MD - *Mean Depth*
MLP - *Multi Layer Perceptron*
MSE - *Mechanical Specific Energy*
MW - *Mud Weight*
NaN - *Not a Number*
NPP - *Normal Pore Pressure gradient*
NPT - *Non Productive Time*
OD - *Outer Diameter*
OPC UA - *Open Platform Communications Unified Architecture*
PCA - *Principal Component Analysis*
PDC - *Polycrystalline Diamond Compact*
PID - *Proportional Integral Derivative controller*
PLC - *Programmable Logic Controller*
POOH - *Pull Out Of the Hole*
PWM - *Pulse Width Modulation*
Q-Q plot - *Quantile-Quantile plot*

RBF - *Radial Basis Function*
RDBMS - *Relational Database Management System*
RF - *Random Forest*
RIH - *Run In Hole*
RL - *Reinforced Learning*
RMSE - *Root Mean Squared Error*
ROnB - *Rotating On Bottom*
ROP - *Rate Of Penetration*
RPM - *Revolutions Per Minute*
SD - *Secure Digital*
SPP - *Stand Pipe Pressure*
SQL - *Structured Query Language*
SVD - *Singular Value Decomposition*
SVM - *Support Vector Machine*
TF - *Toolface*
TOB - *Torque On Bit*
TVD - *True Vertical Depth*
UCS - *Uniaxial Compressive Strength*
UI - *User Interface*
UIS - *University of Stavanger*
WOB - *Weight On Bit*
XML - *Extensible Markup Language*

Chapter 1

Introduction

1.1 Research Problem

“My contention is that machines can be constructed which will simulate the behavior of the human mind very closely. They will make mistakes at times, and at times they may make new and very interesting statements, and on the whole the output of them will be worth attention to the same sort of extent as the output of a human mind. The content of this statement lies in the greater frequency expected for the true statements, and it cannot, I think, be given an exact statement. It would not, for instance, be sufficient to say simply that the machine will make any true statement sooner or later, for an example of such a machine would be one which makes all possible statements sooner or later. We know how to construct these, and as they would (probably) produce true and false statements about equally frequently their verdicts would be quite worthless. It would be the actual reaction of the machine to circumstances that would prove my contention, if indeed it can be proved at all.” (Turing, 1951) [1].

In recent years, the concept of drilling automation has advanced from primarily being automation of rig floor equipment to novel solutions that rapidly can be deployed to the rig environment and assist the driller in a variety of operations. Aside from providing an early warning to the driller, and if necessary, for instance perform a controlled shut-in procedure should a kick that is migrating towards surface get detected, such intelligent systems could improve efficiency and reduce financial costs through continuous monitoring and interaction with the driller. Smart drilling systems could also be used to suggest operating parameters to the driller through correlating real-time drilling data with vast amounts of historic data stored in a virtual environment, popularly referred to as a digital twin, or

even exert full control of all rig equipment if permissible (top drive, draw works, mud pumps, elevator, rough neck and so on) leaving only major decision points to be determined by the driller. The latter automation level described for the drilling scene is likely still several years away from being deployable to the field. A timeline that highlights artificial intelligence (AI) applications in drilling practices is given in Application Of Artificial Intelligence Methods In Drilling System Design And Operations: A Review Of The State Of The Art (Bello et al., 2015) [2].

In our opinion, short term advances in drilling automation lies in developing simple, yet robust tools for the driller to strengthen the understanding of the operations during critical phases, and if possible, automate some of the routine tasks that easily can be controlled by a machine. In the same way that a Tesla can self-operate in known terrain, so should the drilling rig, enabling the driller to remain one step ahead, maintaining an overview of all on-going events.

According to Jordan and Mitchell, “The past decade has seen rapid growth in the ability of networked and mobile computing systems to gather and transport vast amounts of data, a phenomenon often referred to as “Big Data.” The scientists and engineers who collect such data have often turned to machine learning for solutions to the problem of obtaining useful insights, predictions, and decisions from such data sets.” (Jordan & Mitchell, 2015) [3]. Given the big amount of wells that get drilled every year, and the amount of data that gets collected from operations, an approach to predict and make decisions based on the data that already gets collected through the use of machine learning models is an important area of research.

In this thesis, an approach to develop data-driven models to classify different rock formations is presented. The models have been developed using so-called *supervised* machine learning, and get trained and validated using both time-based experimental data that have been collected in the laboratory environment on a test bench (see section 1.2) and field data from the Volve field released by Equinor and its partners [4]. Furthermore, unsupervised machine learning models have been developed to classify some drilling incidents such as stuck pipe and plugged drill bit nozzles, in addition to common rig operations such as tripping (POOH – pull out of the hole and RIH – run in hole) and rotating on bottom (ROnB), meaning that the pipe is rotated on bottom of the hole without being raised or lowered in the axial direction.

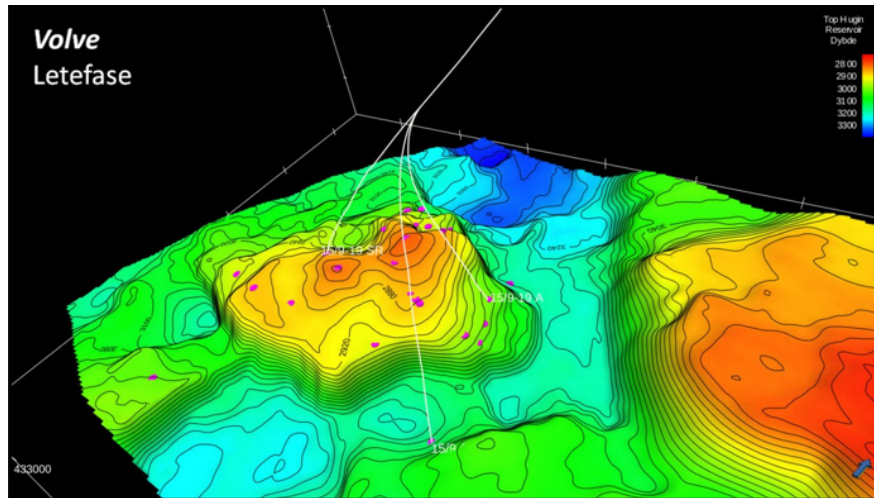
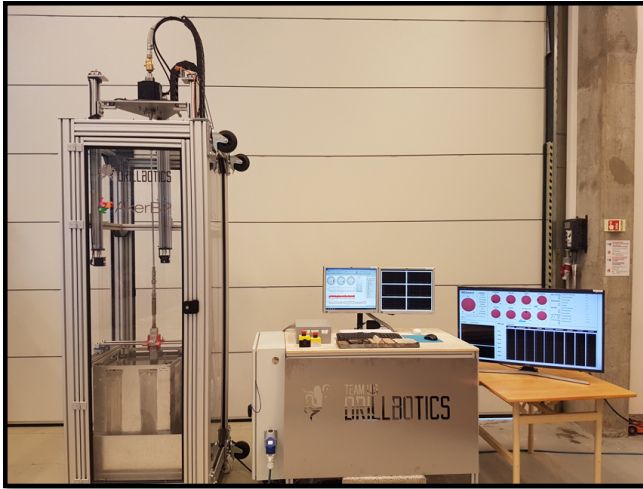


Figure 1.1: *3D model of the Volve field (created in the exploration phase), where all subsurface and production data has been made publicly available [4].*

Since experimental test data has been gathered using an autonomous laboratory-scale drilling rig that has been developed as part of the UiS Drillbotics project at the University of Stavanger, the thesis will also cover some of the parallel activities to the data-driven model development, such as developing a bottom hole assembly (BHA) with an integrated sensor package that is capable of measuring forces and bit-rock interaction in real-time during drilling, autonomous search algorithms allowing the system to maximize the rate of penetration (ROP) while minimizing the risk of running into drilling incidents and drilling with sub-optimal parameter setpoints, and finally a digital algorithm that has been implemented to fully autonomously drill a deviation well using a closed-loop steering system.

1.2 Background of Study

Since 2016, students at UiS have designed, constructed and conducted experiments in order to optimize an autonomous drilling rig of approximately $3 \text{ m} \times 1.5 \text{ m} \times 1.5 \text{ m}$ size. While the system initially got developed with the aim to participate in the annual Drillbotics[®] competition (Loeken and Trulsen, 2017) [5] (Hjelm and Nilsen, 2018) [6], hosted by the Society of Petroleum Engineers (SPE) [7], the machine is now being used as a test bench to investigate the potential in various drilling automation applications and data analytics. A picture of the rig as it was in the end of 2018 is shown in Figure 1.2a. Equipping the rig with all upgrades, the improved system is shown in Figure 1.2b.



(a) *Autonomous Laboratory-scale drilling rig developed at the University of Stavanger.*



(b) *Upgraded rig in 2019.*

Figure 1.2: *Laboratory drilling system at the University of Stavanger.*

Several articles have been published on the research that has been conducted with the drilling system, see (Loeken et al., 2017) [8], (Geekiyanage et al., 2018) [9] (Loeken et al., 2018)[10] . An important objective in 2019 has been to evaluate whether or not methodology that has been developed to create drilling-related models on the laboratory scale can be directly associated with field data, thus strengthening the research value of the project. More information regarding the experiment setup (rig) to develop the models is given in chapter 2.

1.3 Control System Architecture

The drilling system is a collection of several systems including mechanical, hydraulic, electrical, hardware, software, and data systems. The hardware architecture is comprised of three levels; programmable logic controllers (PLCs) on level 1, computers on level 2 and cloud-based storage of time-based data (optionally also depth-based data) on level 3. On level 1, the control aspect (PLCs) for

the five main systems on the rig are located (rotational system, hoisting system, pneumatic system, riser and whipstock positioning system and finally hydraulic mud system). Each system gets controlled through its own microcontroller / PLC, that both executes commands to the controllable systems on the rig (actuators, motors, pumps and so on) and gathers information from the equipped sensors. The PC located on level 2 handles the decision-making logic. This logic involves carrying out the digital detailed operating procedure (DDOP), ROP search algorithms, steering algorithm, stuck pipe model, machine learning models, detection of drilling incidents and carrying out remedial actions, as well as pushing the data to a configured database for post-analysis and data extraction (level 3). A key factor has been to ensure that all components that make up the drilling system are compatible for real-time and autonomous control.

In 2018, the software architecture was configured to operate as a finite state machine (FSM), meaning that the system at any time would operate in a particular state (from a pre-defined number of finite states such as calibration state, normal drilling state, remedial action state and so on). The Arduino Due microcontrollers / PLCs were programmed using the Arduino integrated development environment (IDE), while the control system was configured using Python and the Visual Studio Code IDE. In 2019, the software architecture has seen a major upgrade, and is now configured as a multithreading system, meaning that several client/server modules, referred to as threads that run on multiple cores on the central processing units (CPUs), run in parallel using a gRPC API to handle communication between each module / thread. In addition, an API based on OPC UA has been developed to support remote connectivity and remote event handling, allowing external partners to execute commands remotely to the control system, for instance through their own API, and receive data from the rig in real-time upon request. The control system is described further in section 2.8. Additional details are found in (Sand, 2018) [11] and (Guggedal and Steinstø, 2019) [12].

Chapter 2

Experimental Setup

2.1 Drilling Rig System

The drilling rig consists of in total six hardware systems, in addition to the control system. These are; *rotation, hoisting, circulation, pneumatic, whipstock positioning* and *power* systems. In the following subsections, all key systems get described shortly.

2.1.1 Rotation System

There exists two rotational systems that can be used: a conventional top drive, used for vertical drilling and a downhole motor used for directional drilling. While the two rotational systems in theory can operate simultaneously, with the current downhole motor specifications, the torque is less than in the top drive which could cause severe motor damage if for instance the bit gets stuck.

For the conventional system, a hollow-shaft brushless motor is used to rotate the assembly. The top drive transfers torque directly to the drill string and provides a rated torque of 2.86 Nm and a maximum instantaneous torque of 8.59 Nm. Because the motor is hollow shafted, the mud injection hose that runs along the derrick can be connected on top of the motor using a swivel (rotary union) (See Figure 2.1). This set-up allows us to circulate drilling fluid (either water, water-based mud - WBM or oil-based mud - OBM) all the way from top of derrick to the drill bit nozzles. Velocity margins and pressure loss across the system can be calculated as shown in (Akisanmi, 2016) [13].

The top drive provides rotational speeds up to 3500 RPM but is limited to 1500 RPM due to the rotary union currently in place.



Figure 2.1: *Picture showing the connection between the hose, swivel and motor [6].*

By varying two analog voltage signals one can control the RPM- and torque-output from the top drive. The signals are transmitted from programmable logic controllers (PLCs) to a driver for the top drive. The motor driver also has a dynamic braking function that can be programmed using the autonomous control algorithm, or the driller can define the absolute motor torque limitations. This brake acts as a safety measure for the system so that if the system were to exceed the given torque limitations, it will stop, temporarily, in order to prevent incidents such as buckling or twist-off. Instantaneous twist-off due to fatigue can however still occur. For the case of this particular rig this is very useful because the drill string is made out of aluminum which has a low mechanical strength and buckling limit, as has been calculated by students on the team in 2018. The braking function can be controlled in the same way that RPM and torque operating setpoints get controlled.

2.1.2 Hoisting System

In order to simulate drilling operations, the system is equipped with a hoisting system that consists of three actuators, each equipped with its designated stepper motor and brake of type normally-closed (in order to reduce the holding torque on the stepper motors when the system is not running). The motors raise/lower the top plate that resembles the drill floor where top drive and other components are mounted. The top plate is situated on three brackets, each equipped with a tri-axial load cell that measures the free hanging weight and hook load of ap-

proximately one third of the combined drill floor weight. The decision to use three actuators was made for several reasons where the most important were; to ensure that a sufficient lifting force can be provided, the system will be more rigid when mounted between three actuators to prevent vibrations on the drill floor and finally it allows for smaller incremental changes to actuator distance movement which in turn is essential to regulate the WOB. Various safety algorithms are in place to reduce for instance the risk of a brake accidentally closing when either of the three actuators are moving, which can possibly damage parts of the system (Hagen et al., 2018) [14].

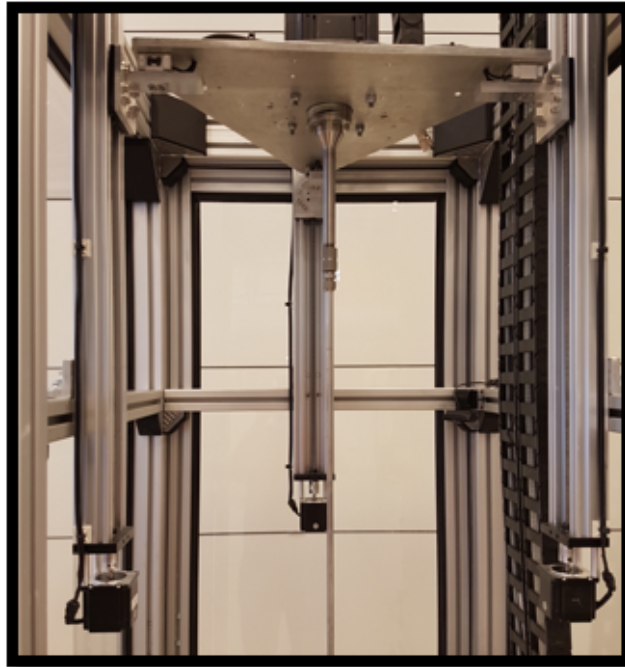


Figure 2.2: *Illustration of hoisting system, consisting of three actuators operating in synchrony.*

The three actuators are controlled by its own stepper motor with a step-angle of 1.8 degrees, where for each 1.8 degrees of stepping 10 micro-steps get transmitted (resulting in a total of 2000 steps/rev). The stepper motor is connected to the lead screw, where each revolution of the screw results in 8 mm travel length of the top plate which gives an elevation resolution of 4 μm . High accuracy for the actuators is as mentioned above key in order to ensure the required WOB control. Therefore, powerful stepper motors that combined can provide an approximate maximum WOB up to 500 N is used. To address the challenge of vibrations in the system, a rigid coupling is used between stepper motor and actuator to reduce the

overall vibration making the system more sturdy compared to the use of spring loaded coupling.

More information and details are found in thesis from 2017 and 2018 [5] [14] [6].

2.1.3 Circulation System

The circulation system consists of two pumps alternating for 30 seconds each. This prevents the pumps from over-heating and should one pump fail, the other will kick in to ensure the required velocity margin for adequate hole cleaning and cooling of the bit. The pumps have a maximum flowrate of 19 L/min at a working pressure of 3.1 bar (Hagen et al., 2018) [14].

Figure 2.3 shows the circulation system mounted inside the rig structure. This gives a mobile set-up where no dismantling is needed before transportation of the rig, while simultaneously reduces the chance of a leak damaging the electrical components on the rig.

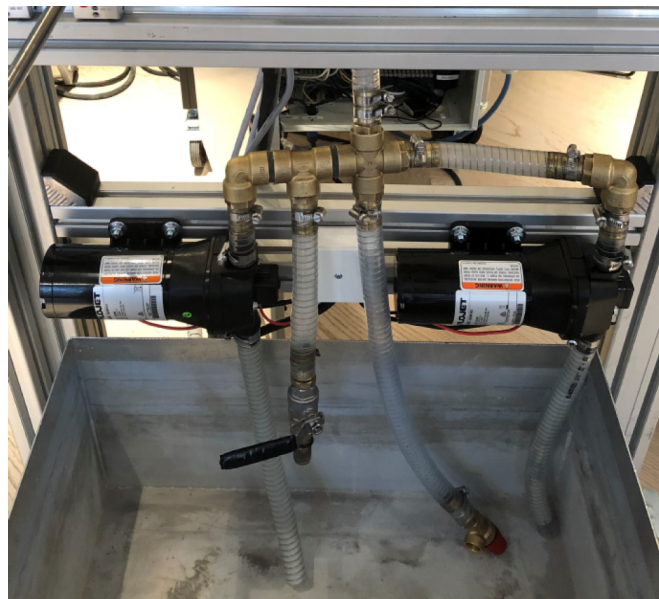


Figure 2.3: *Picture showing the circulation system mounted inside the drill rig.*

More information and details are found in [14] [6].

2.1.4 Drill String Assembly

The drill string assembly developed in 2019 consists of:

- Aluminum drill pipe

- Downhole sensor sub
- Knuckle joint
- Pneumatic downhole motor
- Drill bit

The concept of the drill string assembly developed with the aim to drill directionally is shown in Figure 2.4. While the knuckle joint can be used to control the dogleg by varying the WOB setpoint thresholds either so that a spring gets compressed (bend) or left uncompressed (no bend), the mechanical angle of the top drive can be varied through pulsing to change the azimuth, if an offset is registered. Finally, the bit RPM is controlled by throttling the air flow coming from a compressor to the downhole motor.

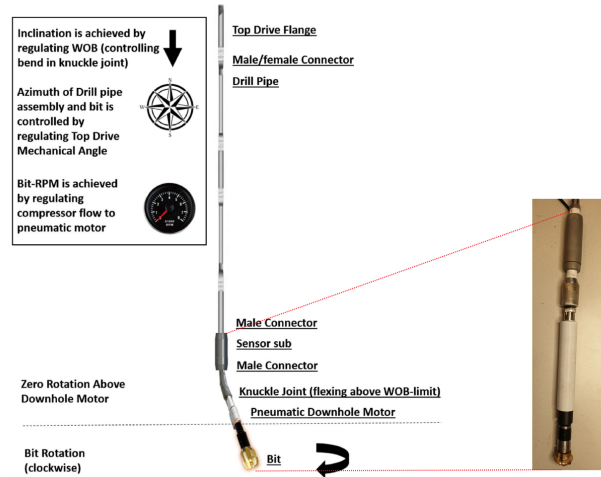


Figure 2.4: *BHA consisting of pipe, sensor sub, knuckle joint, downhole motor and bit.*

Aluminum Drill Pipe

The aluminum drill pipe used for drilling has the following dimension:

Parameter	Description and Unit
Material	Aluminum, 6061-T6 alloy
Pipe length	914.4 mm
Pipe OD	9.53 mm
Pipe ID	7.75 mm
Wall thickness	1.2446 mm

See [14] for pipe limit calculations and destructive testing of the drill pipe. In short, calculations and destructive tests performed have shown that:

- the critical slenderness ratio of the pipe is 70.2, and the slenderness ratio of the drill pipe is approximately 280.2,
- provided the same pipe dimensions and material above, the approximate buckling limit is 280.5 N,
- the maximum torque before the pipe yields is approximately 19.0 Nm, while the maximum torque before the pipe shears is approximately 24.7 Nm.

Destructive testing carried out in both 2017 and 2018 has shown that the pipe can sustain a maximum tensile load of approximately 10.331 kN and a maximum compressive load 985.72 N. [14].

2.2 Additions to facilitate for directional drilling

The rig concept to facilitate for directional drilling consists of several additions, in which the most important is presented below. A schematic of the complete system is presented in Figure 2.5:

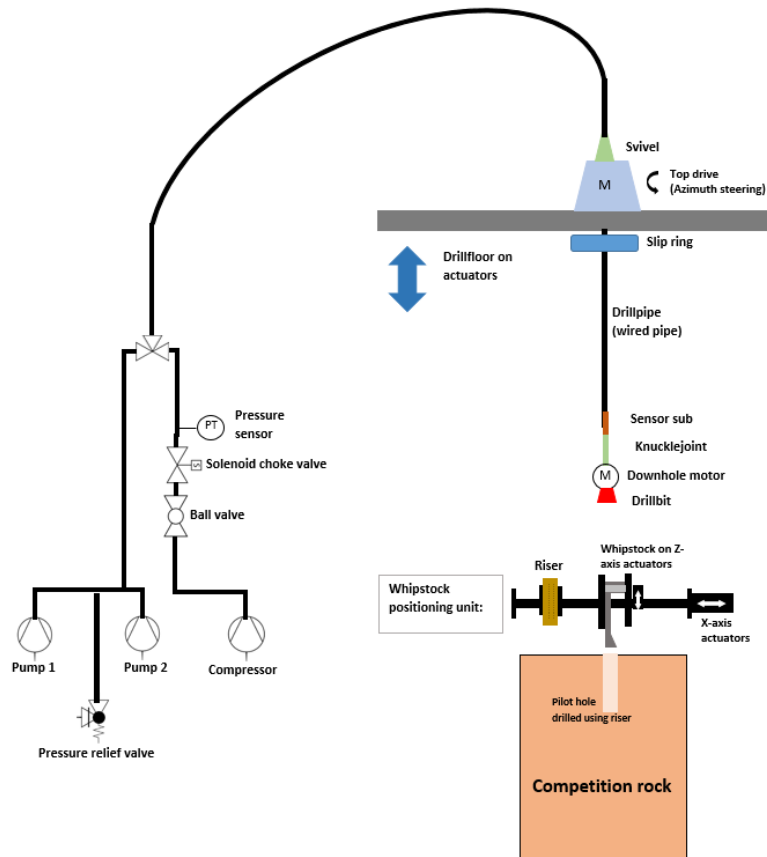


Figure 2.5: Schematic of the drilling rig design that allows for for directional drilling through downhole motor, benable BHA, downhole position tracking and whipstock to kick off from vertical below KOP.

2.2.1 Downhole motor (pneumatic)

Two pneumatic motors have been purchased to allow for both vertical and horizontal drilling by rotating only the bit (and crossover that connects the bit to the pneumatic motor). The pneumatic motors of type 302916D and 302916F have nominal torques of 2.1 and 4.9 Nm and nominal speeds of 750 and 330 RPM respectively, producing a nominal power of 170 W (= 0.23 HP) [15].

Motor size 2			
Series 104- with collet	Type	104-320-1	104-520-1
	Part no.	302916D	302916F
Series 104- for drill chuck	Type	104-320	104-520
	Part no.	302912D	302912F
Nominal power	W / HP	170 / 0.23	170 / 0.23
Nominal speed	rpm	750	330
Speed (idling)	rpm	1500	660
Nominal torque	Nm / in.lbs	2.1 / 18	4.9 / 43
Start torque min.	Nm / in.lbs	3.2 / 28	5 / 44*)
Air consumption	m ³ /min / cfm	0.3 / 11	0.3 / 11
Weight	kg / lbs	0.43 / 0.95	0.43 / 0.95
Hose I.D.	mm / in.	6 / 1/4	6 / 1/4
for drilling steel up to	Ø mm / in.	4 / 5/32	4 / 5/32
for drilling aluminium up to	Ø mm / in.	6 / 15/64	6 / 15/64

Performance data relate to an air pressure of 6 bar (85 PSI) and lubricated air. With oil-free operation the output will be reduced by 10 - 20%.
*) max admissible torque
External parts regular steel.

Figure 2.6: Both pneumatic motors deliver 170 W of nominal power [15].

According to the manufacturer datasheet, the power output in kW can be calculated from the following equation [16]:

$$P[kW] = \frac{M[Nm] \times n[RPM]}{9550} \tag{2.1}$$

The speed and power output from the pneumatic motor can be regulated by either regulating the air volume or pressure of the system. In terms of regulating the air volume, speed can be varied by throttling the exhaust air, while power and torque can be varied through throttling of the air inflow to the motor. Speed, torque, power and air flow can be regulated by varying the operating pressure of the system. In terms of maximum allowable shaft load (assuming the bit being 25 mm away from the collet end) both motors are rated to sustain 380 N axial force and 50 N of radial force.

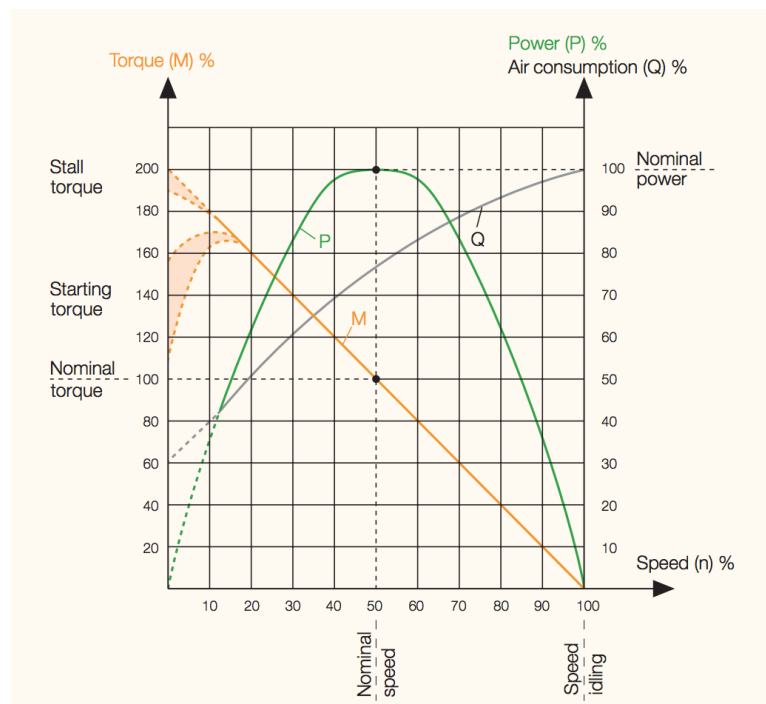


Figure 2.7: *The characteristic of the pneumatic motors [15].*

The pneumatic system on the rig is designed with a compressor that delivers 345 LPM, a hydraulic maintenance unit (to separate out water from the compressed air, add oil to the air flow for lubrication and choke the inlet pressure to the pneumatic motor), a pressure relief valve, a solenoid valve (to throttle the inlet pressure from 0-100 %), a pressure transmitter, a rotary union (to connect the air hose to the rotating shaft of the top drive) and three manual ball valves to 1) relief the pressure between the hydraulic maintenance unit and solenoid valve, 2) relief the pressure between between the solenoid valve and pneumatic motor and 3) manually choke the air flow after the solenoid valve.

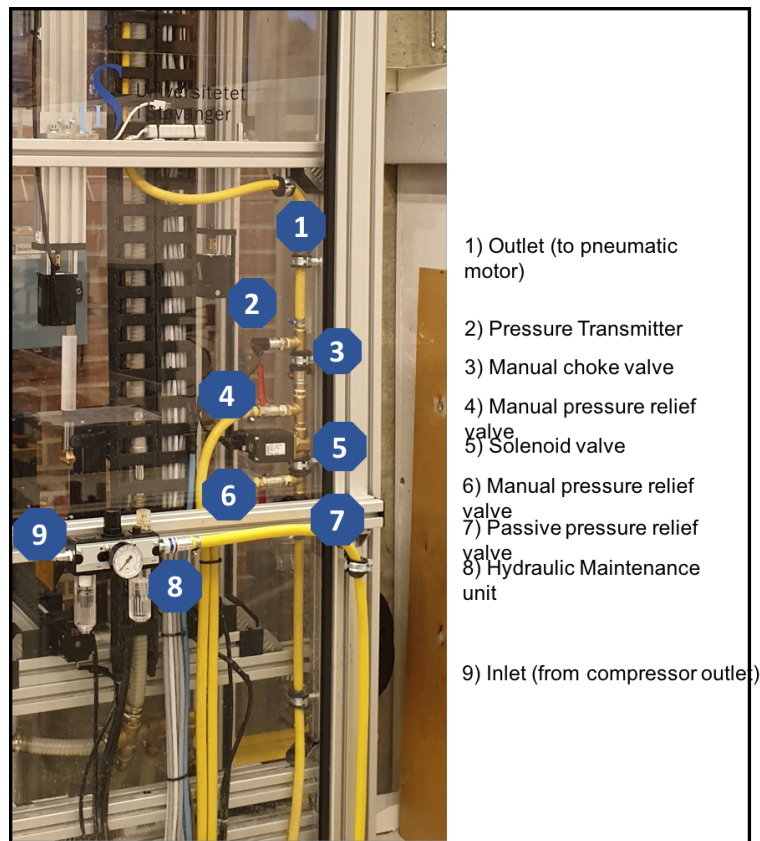


Figure 2.8: *Pneumatic system comprised of various components to regulate the air inflow, monitor pressure and lubricate the system.*

For experiments, the 1st ball valve (between hydraulic maintenance unit and solenoid valve) can be opened during drilling to simulate a leak in the system (which is monitored by the pressure transmitter) or losses into the formation. The 3rd ball valve (manual choke) can be closed to simulate loss of hydraulics, or to tune the solenoid valve opening positions. In addition, a blind plug can be installed at the air hose outlet to simulate plugged nozzles (also monitored by the pressure transmitter). The pressure relief valve ensures that the operating pressure in the pneumatic motor at no time can exceed the maximum pressure rating of 6 bars, if for instance a different compressor is connected to the system, or if the hydraulic maintenance unit is bypassed for various reasons. For more details, see [17].

2.2.2 2-axis Actuator System

To facilitate for the Drillbotics[®] 2019 competition to drill a deviation well, a 2-axis actuator system has been developed to provide the following functionality:

- Shift between drilling with riser, whipstock or no lateral constraints above

the wellbore,

- Lower a whipstock into a pilot hole wellbore section,
- In the future, place a bearing around the drill pipe (which is mounted between actuators on the y -axis) and perform a comparative study on vibrations and the response to drilling with and without a constrained drill pipe.

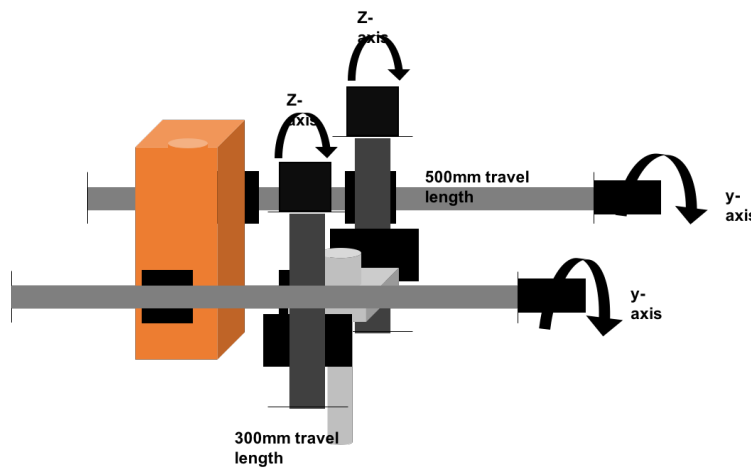


Figure 2.9: *Concept of mounting a riser and a whipstock on the 2-axis actuator system, allowing the system to drill a pilot hole through the riser, re-position and lower the whipstock into the pilot hole, and use the whipstock to kick off from vertical.*

The actuators are operated by in total $4 \times$ Nema23 stepper motors with $4 \times$ GeckoDrive G250X drivers, controlled by an individual Arduino Due microcontroller. The two stepper motors used for lowering and raising the whipstock on the z -axis have a combined holding torque of 25.2 kg-cm while the two motors moving the whipstock and riser in the y -direction produce a holding torque of 49 kg-cm. This is to ensure that two z -axis actuators and a whipstock can be operated on the y -axis without operating close to the maximum torque of the stepper motors. The finalized design is shown in Figure 2.10 below:

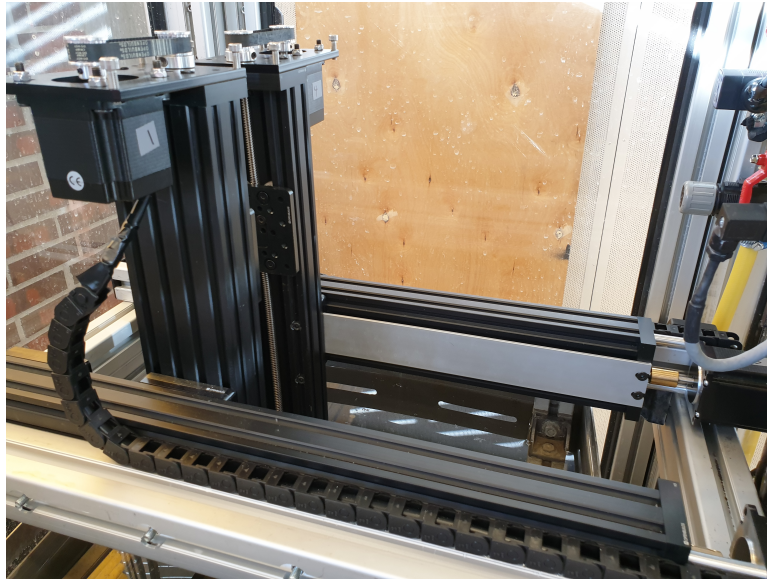


Figure 2.10: *2-axis actuator system as it was installed on the rig.*

An illustration of the bit entering a whipstock (that has been positioned inside of a 165 mm deep pilot hole section) is given in Figure 2.11:

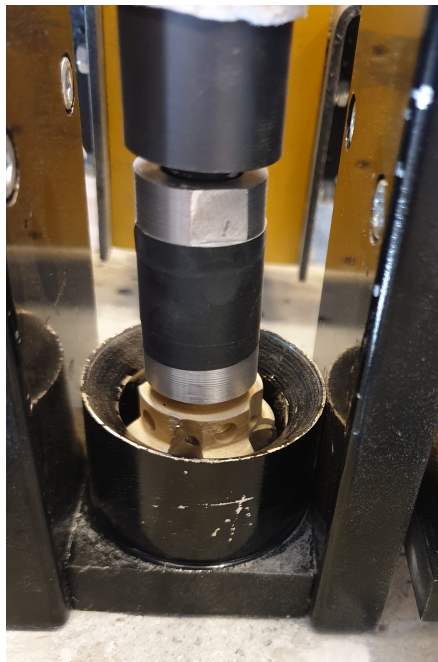


Figure 2.11: *Bit enters the whipstock that has been lowered approximately 165 mm into a pre-drilled pilot hole.*

For more details, see [17].

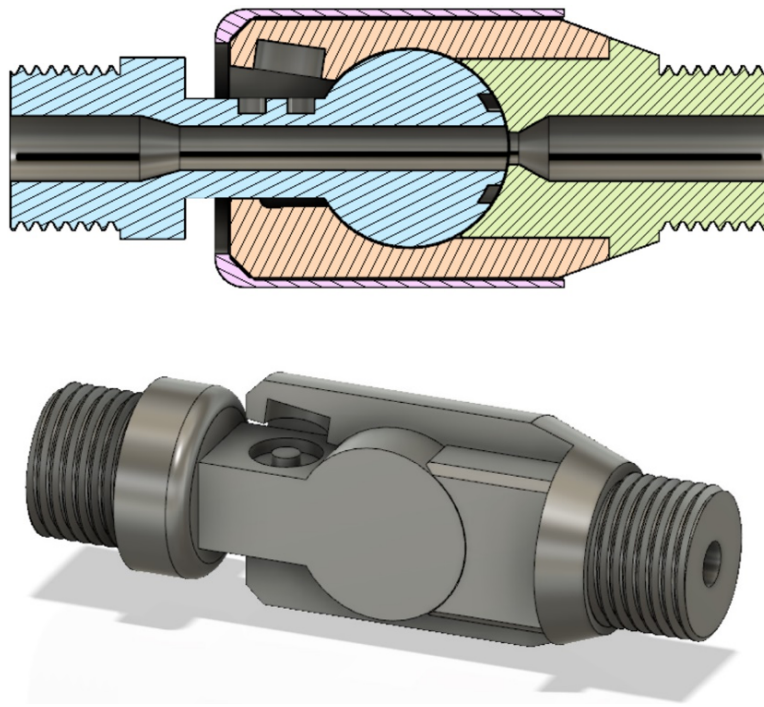


Figure 2.13: *Knuckle joint design, developed by Sander Skjørestad [17].*

Drill bits

For details regarding the Baker Hughes drill bit provided by the Drillbotics[®] competition committee and the 2-cone and 3-cone bits acquired, see [17].

In short, the Baker Hughes poly diamond crystalline (PDC) bit provided has an OD of 1.25 in, a bit length of 1.25 in a back rake angle (BRA) of 25 degrees for the two cone cutters and 20 degrees for both shoulder cutters. Clearance at gauge pads is configurable from 0 to 0.15 in in such way that tungsten carbide avoid elements can get inserted to control either clearance or depth of cut (DOC).

2.3 Rig Sensors

The sensors that are installed on the rig are:

Sensor:	Description:	Range:	Units:
W_1, W_2, W_3	Surface WOB load-cells – <i>Forsentek FNZ 100N</i>	-300 – 300	N
T_1	Surface torque – <i>APM-HE09ACH inbuilt encoder</i>	0 – 8.59	Nm
T_2	Drill pipe torque – <i>TAM Active-3 Advanced</i>	10^{-4} – 10^3	MPa
H	Vertical depth – <i>Sick DT35-B15251</i>	0 – 100	cm
ω	Surface RPM – <i>APM-HE09ACH inbuilt encoder</i>	0 – 3500	RPM
P_{mud}	Mud Pump pressure – <i>Gems 10 bar, IP65, G1/4</i>	0 – 10	bar
$P_{pneumatic}$	Pneumatic pressure – <i>Gems 16 bar, IP65, G1/4</i>	0 – 16	bar
Acc_x, Acc_y, Acc_z	Accelerometer (downhole) – <i>LSM9DS0</i>	2 – 16	G
Gyr_x, Gyr_y, Gyr_z	Gyroscope (downhole) – <i>LSM9DS0</i>	245 – 2000	deg/s
Mag_x, Mag_y, Mag_z	Magnetometer (downhole) – <i>LSM9DS0</i>	2 – 12	gauss
T	Temperature (downhole) – <i>LSM9DS0</i>	-40 – 85	C
L	Virtual step counting – <i>Arduino Due</i>	0 – 900	mm
A	Acoustic sensor* for communication and bit-rock interaction wasted energy dissipation estimation	126 146	dB
V	High-speed camera as a vibration sensor*	0 – 530	frames/sec

(*Non-invasive, portable and underdevelopment sensor).

Figure 2.14: *Rig sensors, including high-speed camera and acoustic sensor for research.*

Rig sensors, are connected either to a high frequency data acquisition (DAQ) module (surface sensors; analog voltage and current signal), or directly to the PC (downhole sensors; digital I²C to microcontroller that transmits usb signal to PC). Each sensor is regularly calibrated according to calibration procedure developed as part of the operating procedure for the drilling rig (provided at the rig).

2.4 Downhole Sensor Sub

2.4.1 Mechanical design

The sensor sub is a essential for several objectives during drilling, ranging from providing the driller with a visualization of the downhole conditions and wellpath to vibration classification, in real-time, while drilling. To do so, a versatile sensor sub design, that is both easy to access, maintain and independent of the other parts of the BHA, has been developed.

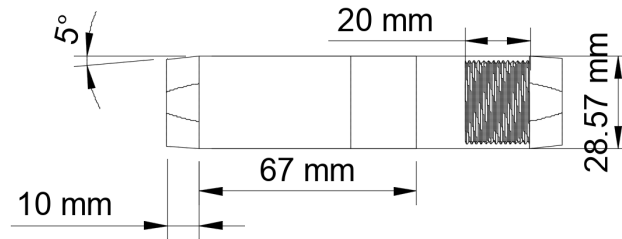


Figure 2.15: *Technical drawings of the sensor sub.*

The sensor sub is designed for easy access, making it possible to open and close the sub in order to replace electronics in case of hardware-failure or damage to internal components. This can be done without interacting with other downhole components which ensures quick replacements or repairs can be carried out if required. Two compartments inside of the house allows for either placing a sensor and a microcontroller (current configuration enabling real-time dataflow to surface), multiple sensors and a tiny controller with internal storage so that data can be accessed post drilling, or the positioning of strain gauges to measure the downhole torque in experiments. Figure 2.16 shows the sensor house when opened. From the housing there is a wire-channel which leads to surface. The OD of the sensor sub is lower than that of the drill bit (under gauge BHA) which ensures that the sub does not scrape along the walls of the wellbore (disturbing measurements if the sub is partially confined in the wellbore) as well as reduces the risk of the non-rotating BHA getting stuck, in particular when passing KOP.

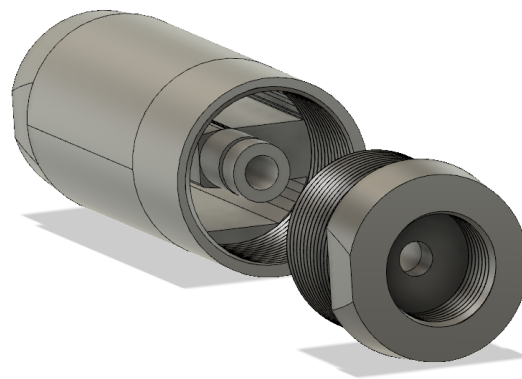


Figure 2.16: *CAD drawing of the sensor sub showing inside compartments.*

The purpose of the sensor sub is to provide the bottom hole assembly with a sensor package that consists of accelerometers, gyroscopes and magnetometers, allowing the rig to calculate inclination build (dogleg), azimuth (heading), and if necessary the orientation of the BHA (in case of over-torque in connections). Hence, through

simple sensor equipment, the well trajectory can get calculated in real-time. In Figure 2.17, a cross-sectional view is provided to illustrate the compartments inside of the sub, as well as the thread pitch alignment of the sensor house and locking mechanism.

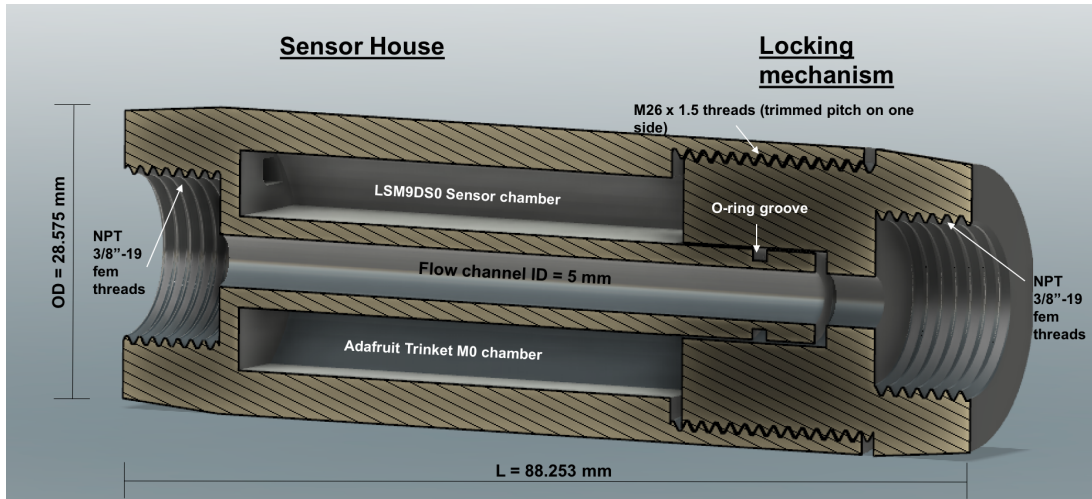


Figure 2.17: *Cross-section analysis showing the dimensions of the sensor house design. The section analysis is used to ensure that thread pitch and depth of threads will match up once the sensor house is 3D printed.*

2.4.2 Mechanical design - Stress simulation

In 2018, a novel BHA design was developed [6] for the Drillbotics[®] competition. While the stabilizers and drill collars were developed to endure severe axial and lateral vibrations (> 300 N force), the flow channel that connected the sensor package, located in the near-bit stabilizer broke shortly after drilling with the BHA commenced (due to fatigue). For this purpose, a short study on five different load cases (static momentum stress, static stress for high WOB, static stress for overpull, structural buckling and burst pressure) for the sensor house was conducted, considering the threaded sleeve that is regarded as the weak point in the sensor house design.

The results from performing the stress simulations are found in Appendix A.

2.4.3 Sensor Package

The inside of the sensor house is fitted with two circuit boards. The sensor circuit board is a 9-DOF accelerometer, gyroscope and magnetometer (see Figure 2.18).

The sensor allows the system to calculate the azimuth (magnetometer), inclination and roll (gyro) and vibrations in the x, y, z -directions (accelerometer).

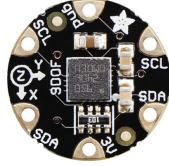


Figure 2.18: *Picture showing the combined FLORA 9-DOF Accelerometer/Gyroscope/Magnetometer sensor [18].*

The sensor circuit board is connected to the Adafruit Trinket M0 (see Figure 2.19) using 3V, GND, SCL and SDA connectors with a I²C communication protocol. From the Adafruit Trinket M0, a micro-USB cable is run to surface. There is already installed a slipring underneath the top drive. This allows for transmission of data from downhole even though the entire system is rotating. For directional drilling using a downhole motor, no slip-ring is required, as all parts other than the bit are stationary permitting a USB cable from running all the way to the PC. This is desirable since when the cable can be shielded in the high-noise area (drill floor surroundings), the risk of lost connection due to interference is greatly minimized. An oscilloscope has been used to measure the interference on and around the drill floor.

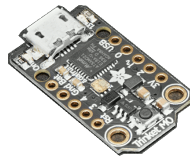


Figure 2.19: *Picture showing the Adafruit Trinket M0 [19].*

The data stream from the downhole sensor, to surface, is controlled by having a pulse generator in the control system transmit a pulse to request data actively; simultaneously a pulse gets sent to the high frequency DAQ to request data from the surface sensors. This alleviates the problem of having multiple data sources that can lead to:

- The internal clocks in the Trinket microcontroller and high frequency DAQ are out of synch, causing data that is sampled at different times to be merged in the control system,
- An overflow of data is piled up in the buffer in the PC, so that the system either crashes or data from the sources is received in the control system with a delay.

There is however one challenge in using such pulse to synchronize sensor measurements taken from multiple sources. If it was not for the pulse, a watchdog timer could easily get implemented to detect anomalies in software, downhole microcontroller, PLCs or even in the control system, and reset the CPU if an incident occurs that causes temporary disconnection between PC and sensor (loss in communication). One could then re-initiate communication with the sensor, if for instance a brownout (a drop in voltage for parts of the system) causes a temporary disconnection from the control system. If a common pulse transmitted to all parts of the control system gets used however, a temporarily disconnection and reconnection could possibly cause instability in the dataflow, which is why such solution has not yet been tested or implemented.

2.4.4 Final Design

Upon receiving the 3D print, supports had to be removed, and all threads needed to get refurbished so that the components in the compartments within the sensor sub are properly sealed off from dust or mud. The inner threads of the sensor house was refurbished by drilling through a thread tap (to allow the flow channel to be bypassed), and fine threading the coarse threads in the sleeve.

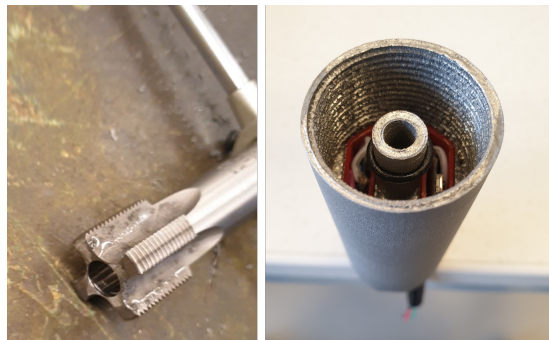


Figure 2.20: *Drilled through thread tap (left) and final results of refurbished threads (right). The tight fit of components inside is shown in the background.*

After sand blasting each component while protecting the threads, and installing the sensors inside of the house, the end results is shown in Figure 2.21.

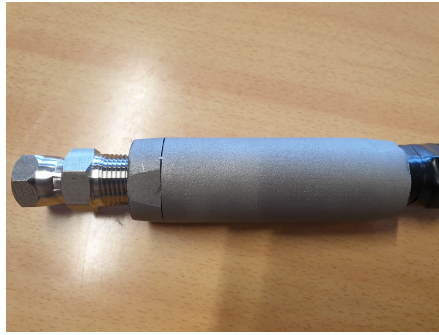


Figure 2.21: *End result of sensor sub in 316L stainless steel, equipped with a sensor board containing ten sensors and a small microcontroller within. A swivel connection (right) allows the sub to be directly connected to the pneumatic motor or to a knuckle joint when drilling deviated well sections.*

2.5 Calibration of systems

2.5.1 Pneumatic Motor

Even if the pneumatic motor is not equipped with an encoder, bit RPM can get calculated in real-time by measuring the torque, pressure, and the solenoid valve opening position (controlled in %). While the top drive brake has been configured to prevent the motor shaft and the drill string assembly from rotating counter-clockwise, the inbuilt encoder of the top drive still measures the torque which is assumed equal in the BHA.

The speed of the downhole motor can be controlled by throttling either the input air flow, the exhaust air flow, or the pressure in the system, as is described in subsection 2.2.1. From Figure 2.7, it can be shown that at nominal torque, the nominal speed is approximately 50 % of the idling speed. This suggests that for motor 302916D, at nominal torque (= 2.1 Nm), the nominal speed is only 750 RPM (as is also confirmed by supplier of motor).

Since the input air flow and pressure are both regulated by the solenoid valve opening position, the first step to calculate the downhole RPM is to use a high frequency camera to measure the number of revolutions per second of the bit at different valve opening positions. A framerate of approximately 150 Hz was selected, since at 1500 RPM only a framerate higher than 25 Hz is required to track 25 revolutions of a pixel per second. The experimental setup is shown in Figure 2.22.

Ekaterina Wiktorski at the University of Stavanger has developed an approach to use pixel tracking to measure the vertical- and horizontal displacement of the drill string. This allows vibrations in the drill string to get measured without attaching sensor equipment to the pipe, thus affecting the natural frequency of the system [20]. Using the same algorithm, the vertical displacement of an isolated pixel is tracked using an algorithm written in MATLAB, and the frequency that the pixel occurs with is then used to determine the number of revolutions per second.

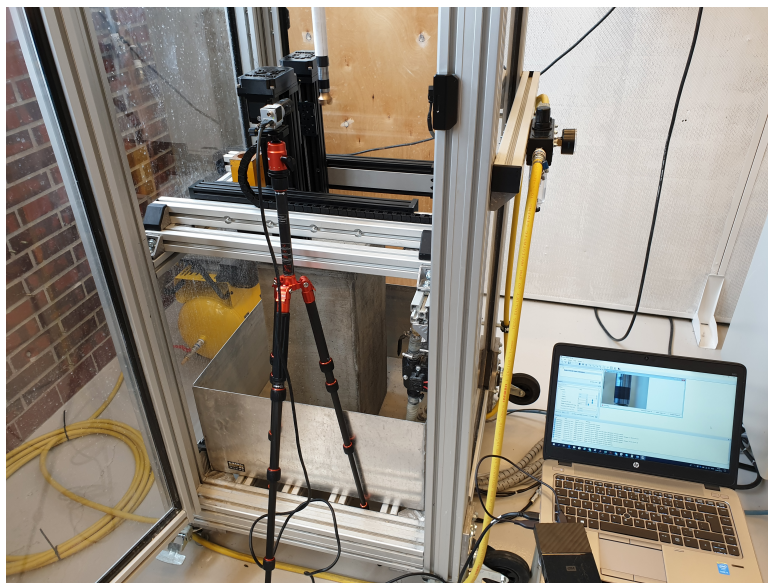


Figure 2.22: *High frequency camera is used to measure the idle speed of the pneumatic motor at different solenoid valve positions (60 to 100 %). Pylon Viewer is used to configure the camera capture settings.*

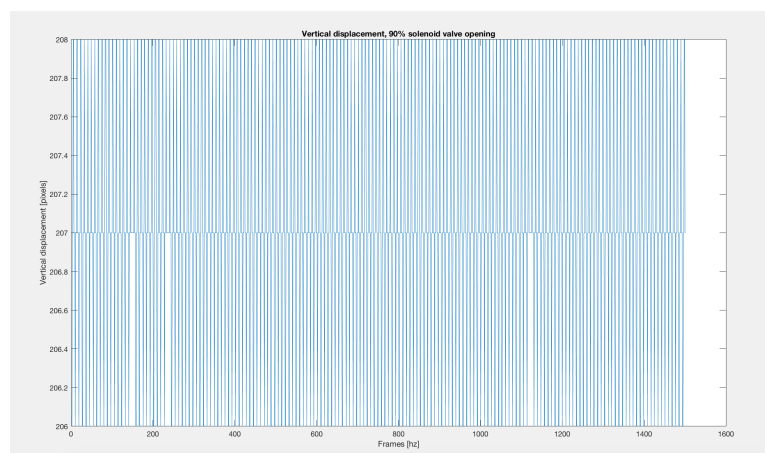


Figure 2.23: *Vertical displacement of a pixel at 90 % solenoid valve opening.*

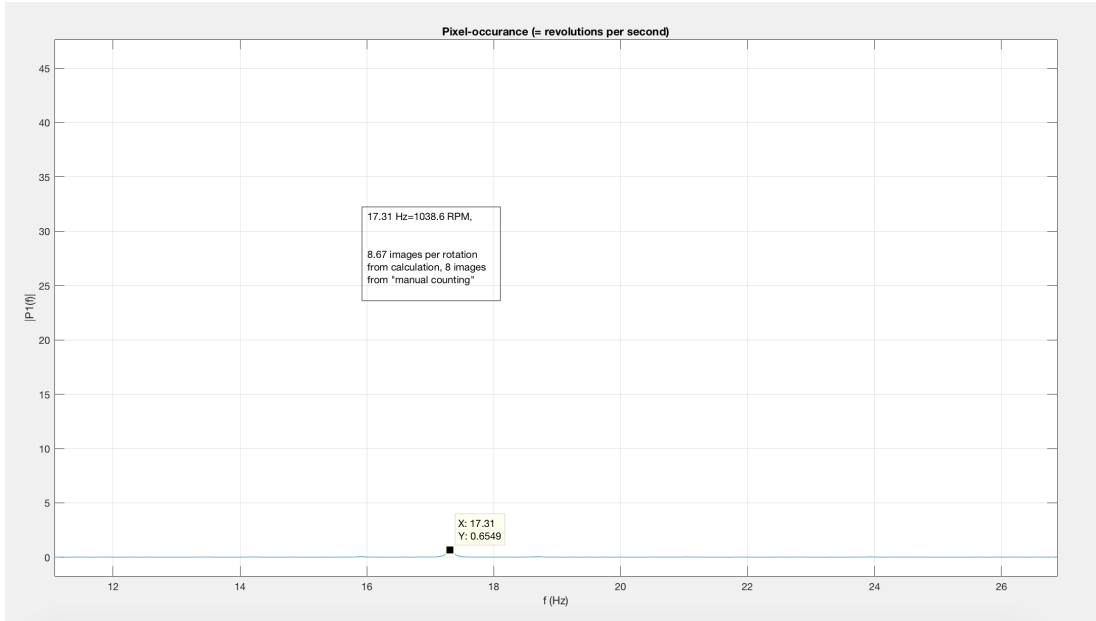


Figure 2.24: Frequency of pixel occurring in the same position, so that bit RPM can be calculated.

The RPM can then be calculated by the following equations:

$$RPM = -0.4509 \times (Valveopening)^2 + 88.289 \times (Valveopening) - 3249.4 \quad (2.2)$$

$$RPM_{idle} = \frac{f \times 60}{fr} \quad (2.3)$$

where f (frequency of occurring pixel) is given in Hz, fr (framerate of the camera) in Hz and the constant 60 refers to the number of seconds in a minute. For example, this suggests that for a 90 % solenoid valve opening, the idle bit RPM is 1038.6, since the pixel occurred every 17.31 Hz. This can be verified by manually counting how many images correspond to one full bit revolution, which in this case corresponded to between eight and nine ($\frac{150Hz \times framerate}{8images} = 18.75$ Hz).

Knowing the idle RPM, the actual bit RPM can now be calculated in the system using linear regression.

$$RPM_{actual,corrected} = RPM_{idle} - \frac{RPM_{idle}}{2.1} \times torque_{topdrive} \quad (2.4)$$

This suggests that if the bit at 90 % solenoid valve opening is rotating at 1038.6 RPM and a torque of 1.2 Nm is registered, the actual downhole RPM and power produced is equal to:

$$RPM(\text{rev}/\text{min}) = 1038.6 - \frac{519,3}{2.1} \times 1.2 = 610RPM \quad (2.5)$$

The power output can be calculated to be:

$$Power = \frac{1.2 \times 610}{9550} = 0.0766kW = 76.6W \quad (2.6)$$

For the pneumatic motor that provides 2.1 Nm nominal torque and 1500 RPM idle speed, a characteristic using this approach can be created so that:

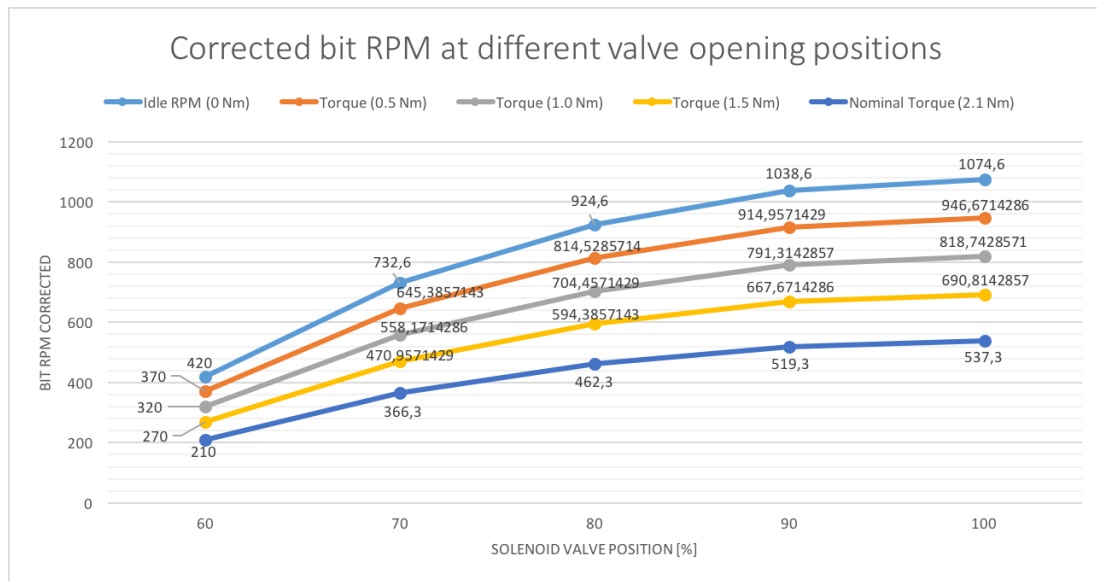


Figure 2.25: Bit RPMs for solenoid valve opening ranging from 60 to 100 %, corrected for 0 to 2.1 Nm torque.

2.5.2 Top Drive

For several experiments, the top drive is still used to provide the torque on bit. Previously, the top drive has been controlled using Pulse Width Modulation (PWM). Due to the inconsistent PWM output from the logic controller, the top drive has been re-tuned using the analog output pins from the Arduino, that support an operating range of 0.55 V to 2.75 V.

Through calibration, the top drive can now be controlled from 0 to 1500 RPM with an offset of only 1-2 RPM, which is a significant improvement compared to previous years where the offset has been as high as 50 RPM at certain rotational speeds. For more details regarding how the top drive gets tuned, see [5] and [6].

Command	Value	Unit
Analog velocity command scale	682	rpm/V
Analog velocity command offset	450	mV
Analog velocity command clamp level	10	rpm (or mm/s)

2.5.3 WOB Control

A brief introduction to the Proportional Integral Derivative Controller

The PID controller is the most used controller for closed-loop systems in the industry and in equation 2.7 below the equation for the controller is shown (Sui, 2019) [21]:

$$u(t) = u_0 + K_p e(t) + K_i \int_0^t e(\tau) d\tau + K_d \frac{de(t)}{dt} \quad (2.7)$$

In the following three subsections, each of the parameters that combined make up the PID controller gets described.

The PID controller block diagram is shown in Figure 2.26:

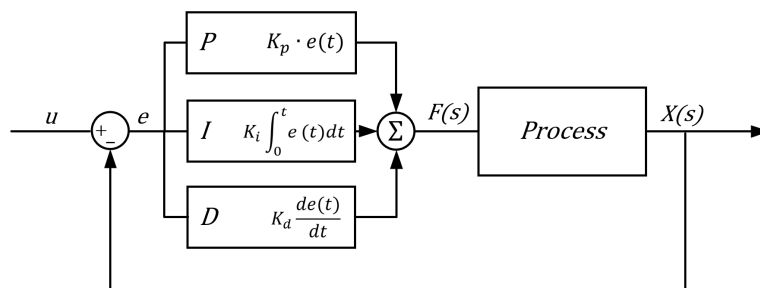


Figure 2.26: PID Controller block diagram illustrating the concept from setpoint (u), controller (P I D), process, output $X(s)$ and error (e) [22].

Proportional controller (P controller)

The P controller is the first and simplest part of the PID controller. The output of this controller is the output of the proportional control mode with a bias added. One needs to add the bias in order to maintain a given output while there is no

error. Equation 2.8 below shows how the output is calculated. The bias (u_0) can be thought of as the “start value” of the loop, while the controller gain (k_p) can be adjusted in order to change the output if necessary. The bias is what controls the error, $u(t)$, and causes it to become 0 (Sui, 2019) [21].

$$u(t) = u_0 + K_p e(t) \quad (2.8)$$

By increasing K_p , one will reduce the offset between setpoint and output and speed up the process response. On the other hand the system may oscillate or even become unstable. It’s important that the system has a realistic proportional gain. A high proportional gain will give the system big responses for the changes in the error but can also make the system unstable. On the other hand, a small change can make the system unreactive, thus not being able to handle larger changes in the system error (Sui, 2019) [21].

Proportional Integral controller (PI controller) In the PI controller the output signal is controlled by a sum of the proportional- and integral control. Using equation 2.9 below, one can alter the control variable (Sui, 2019) [21].

$$u(t) = u_0 + K_p e(t) + K_i \int_0^t e(\tau) d\tau \quad (2.9)$$

The integral part of the controller is added in order to tune the proportional controller in a direction that eliminates the steady state error. There are though negative sides to the adding the integral component such as a reduced response time and decreased stability of the system (Sui, 2019) [21].

Derivative controller (D controller)

The last part of the PID controller is the derivative controller. The D term, illustrated in equation 2.10 below is implemented with the purpose of predicting the future change and increase the stability of the model by considering the rate of change (Sui, 2019) [21].

$$u(t) = u_0 + K_d \frac{de(t)}{dt} \quad (2.10)$$

Implemented Controller

When drilling on the laboratory rig, the hoisting system gets used to control the WOB used to drill the formation. It is important that the selected setpoint is stable during drilling in order to keep a steady drill rate, and the use of a PID controller ensures that the error between the WOB setpoint and measured hook load is minimized, seeing that the WOB is calculated as the difference between free hanging weight and hook load.

In the past, the PID controller that is used for WOB-control on the laboratory-scale drilling rig has been running on an Arduino Due microcontroller to calculate the error between WOB setpoint and actual WOB applied to the formation. With the new control architecture, a PID controller is run as part of the hoisting program in Python (that handles all hoisting-related events) at 60 Hz frequency. A short illustration of the controller implementation using the *simple_pid* library is shown below, where the k_P , k_I and k_D gains can get defined (current gain values of 1,1,1 given for illustration), and upper and lower thresholds allow the operator to adjust how prone the controller should be to axial vibrations caused from vibrations. The rig operator can also select different distances and speeds for the actuators to move either to add or remove WOB.

The PID controller will move down if the WOB setpoint is > 0.5 kg higher than the actual WOB, and likewise the controller will only lift up if the overshoot is more than 1.0 kg greater than the setpoint. If the error is greater than the permissible overshoot, the control system sends an output to the microcontrollers (PLCs) to move 0.5 mm in the appropriate direction. If a new command is received, and an old command is still getting executed in the PLC, the old command gets overwritten by the new. The WOB data that is sampled in real-time is filtered using a moving median filter with a window-size of 5 samples, which combined with 60 Hz frequency has proven to be reliable for WOB control. Since the maximum allowable axial load of the pneumatic downhole motor shaft is 380 N, the maximum permitted WOB with such motor should never exceed 190 N accounting for axial vibrations. There is however a great challenge to tune the PID controller gains when the BHA consists of a bendable knuckle joint. For this reason, test drilling using only a P-controller with $k_P = 1$ has proven adequate.

An algorithm to tune the controller setpoints by measuring the setpoint overshoot, controller response and time to stabilize on setpoint can possibly be created in

the future by performing a grid search of for instance 10^3 combinations of kP, kI and kD.

2.6 Downhole measurements

The downhole sensor (LSM9DS0) is a 9 degree of freedom (DOF) sensor, with an acceleration range of 2 to 16 g, a magnetometer sensitivity of 2 to 12 Gauss and a gyroscope scale between 245 and 2000 degrees per second (DPS). The following sections will describe how inclination, azimuth, and calibration of the sensors in the sensorsub is performed for the sensor sub.

2.6.1 Inclination and azimuth calculations

The downhole sensors are configured with sampling rate of 60 Hz using the Adafruit Trinket M0 and the pulse-algorithm implemented to request and receive measurements synched across all sensor sources (downhole sensor package and surface sensors). The inclination is calculated using the downhole accelerometers:

$$Inclination = \frac{180.0 * \arccos\left(\frac{accelerometer_z}{\sqrt{acc_x*acc_x+acc_y*acc_y+acc_z*acc_z}}\right)}{\pi} \quad (2.11)$$

The azimuth could normally be calculated using the relationship between the magnetometer measurements on the y - and x -axis:

$$Azimuth = \frac{180 * \arctan\left(\frac{magnetometer_y}{magnetometer_x}\right)}{\pi} \quad (2.12)$$

The fluctuation between 360 and 0 degrees can be calculated by a simple if statement so that if $azimuth < 0$, $azimuth = azimuth + 360degrees$.

While the sensor is 90 degrees inclined however, there is a need to calculate a tilt-compensated azimuth, which can be performed according to the example provided in [23]. First, the magnetometer values get calibrated according to an algorithm that measures the minimum and maximum magnetometer values while the sensor is rotated in any direction for 30 seconds. The raw magnetometer measurements

get corrected by the following equation:

$$\begin{aligned} & magnetometer_{x/y/z} \\ &= \frac{(magnetometer_{x/y/z-min} - magnetometer_{x/y/z-max})}{2} \end{aligned} \quad (2.13)$$

Note however that the currently used technique only ensures that the constant offset gets removed. The accelerometer measurements can then be converted to degrees by the equations:

$$Accelerometer_{x-angle} = (\arctan(\frac{accelerometer_y}{accelerometer_z})) * 57.296 \quad (2.14)$$

$$Accelerometer_{y-angle} = (\arctan(\frac{accelerometer_z}{accelerometer_x}) + \pi) * 57.296 \quad (2.15)$$

The constant 57.296 refers to the conversion factor from radians to degrees. Further, the accelerometer values get converted to values between -180 and 180 degrees, and the azimuth can be calculated as shown in equation 2.12. One then needs to normalize raw values from the accelerometer measurements. This can be achieved by equations:

$$\begin{aligned} & Accelerometer_{x-normalization} \\ &= \frac{accelerometer_x}{\sqrt{accelerometer_x^2 + accelerometer_y^2 + accelerometer_z^2}} \end{aligned} \quad (2.16)$$

$$\begin{aligned} & Accelerometer_{y-normalization} \\ &= \frac{accelerometer_y}{\sqrt{accelerometer_x^2 + accelerometer_y^2 + accelerometer_z^2}} \end{aligned} \quad (2.17)$$

The pitch (commonly referred to as inclination), and roll is then calculated from the equations:

$$Pitch = \arcsin(Accelerometer_{ynormalization}) \quad (2.18)$$

$$Roll = -\arcsin \frac{(Accelerometer\,normalization)}{\cos(pitch)} \quad (2.19)$$

Finally, we can calculate the compensated values for the magnetometer measurements:

$$\begin{aligned} magnetometer_{x-compensated} \\ = magnetometer_x \times \cos(pitch) + magnetometer_z \times \sin(pitch) \end{aligned} \quad (2.20)$$

$$\begin{aligned} magnetometer_{y-compensated} \\ = magnetometer_x \times \sin(roll) * \sin(pitch) + magnetometer_y \times \cos(roll) \\ - magnetometer_y \cos(roll) - magnetometer_z \times \sin(roll) \times \cos(pitch) \end{aligned} \quad (2.21)$$

The tilt compensated azimuth can then be calculated from the relationship between the x - and y -compensated magnetometer values:

$$tiltcompAzimuth = \frac{180 \times \arctan\left(\frac{magnetometer_{y-compensated}}{magnetometer_{x-compensated}}\right)}{\pi} \quad (2.22)$$

2.6.2 Calibration of sensors

Magnetometer Calibration

As mentioned, the magnetometer is calibrated by rotating the sensor sub in any direction for 30 seconds, while constantly measuring and saving the magnetometer measurements in the x -, y - and z -axes. After the experiment, the floor and ceiling values (Python command to determine the minimum and maximum value in a series of data) get identified. Equation 2.13 shows how the magnetometer measurements are then corrected for the difference between the maximum and minimum measurement.

Accelerometer Calibration

The accelerometer is similarly calibrated, however while the magnetometer was only rotated for 30 seconds, the following procedure is carried out:

- 1) Rotate the accelerometer for 30 seconds, saving all data and then calculate the maximum and minimum value at normal rotation
- 2) Apply a median filter, for instance with a window size of 60 samples (1 second at 60 Hz), and define the median value as the 0 point for $x/y/z$ axis.
- 3) Based on drilling, perform steps 1 and 2 for the following cases: no movement, normal drilling, aggressive drilling and severe drilling state, so that a classification chart to classify the severity of the vibrations can be used to guide the machine, based on measurements.

An example of how the accelerometer scale can be visualized is:

Vibration level (severity)	-4	-3	-2	-1	0	1	2	3	4
Value	-6 g	-4 g	-2 g	-1 g	0 g	1 g	2 g	4 g	6 g

The final implemented scale and visualization is shown in Figure 2.32.

Gyroscope Calibration

The gyroscope is not being used due to time-constraints caused by delays in the 3D-printing of sensor house, which left only a month to experiment with downhole sensor sub and trajectory calculations.

2.7 Downhole Position Tracking

The downhole position can get tracked as shown in the table presented in Appendix B. An illustration is given in Figure B.1, that illustrate how, by measuring the change in drill bit elevation every five seconds, and then using the median value taken for a five second window to calculate the inclination and azimuth, the mean depth (MD), true vertical depth (TVD), horizontal build (x-displacement), azimuth offset (y-displacement) and estimated inclination and azimuth measuring error can be calculated. By calculating each of the above over a five second window, referred to as a *section* in the figure, the sum of the sections can be added to track the position of the bit.

Based on Appendix B, Figure 2.27 can be generated, which illustrates three 2-D plots of the well trajectory; the rock is seen from either the side (well profile visualization), the top (bird's eye view to visualize the azimuth offset per horizontal build), and the rear (azimuth offset along the TVD). In the figure, the

red line marks the well trajectory calculated, while the dashed pink lines mark the maximum and minimum cumulative offset, based on an error estimate for inclination of 10 degrees and 20 degrees for the magnetometer. A pre-programmed well trajectory, as marked with black, can be calculated for each *section* of five seconds, which can then be used to steer the inclination by either increasing or decreasing the weight on bit on the laboratory-scale drilling rig (see chapter 9). In the example shown in Figure B.1 and Figure 2.27, it can be observed how the well trajectory can get tracked from arbitrarily generated sensor measurements if KOP is assumed to be at 165.15 mm measured depth (MD). A selection of the program written in Python to achieve same functionality in real-time using live measurements is provided in Appendix H.

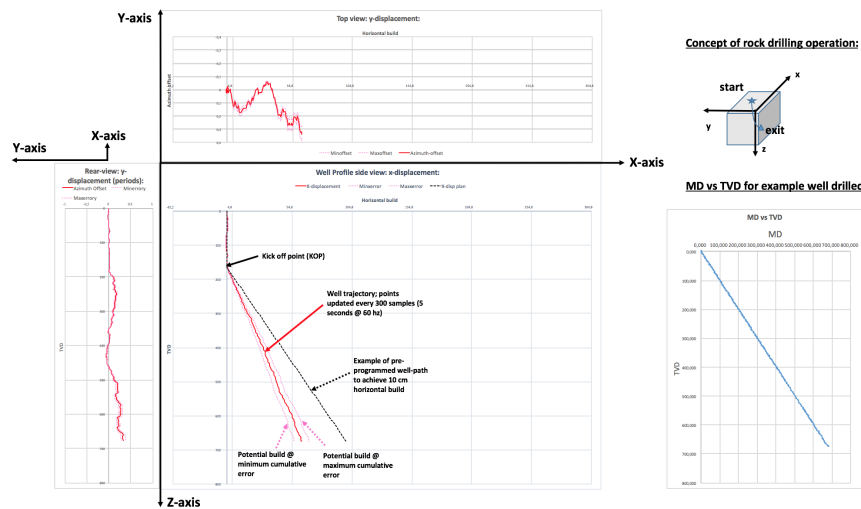


Figure 2.27: Proof of concept; downhole position tracking.

For details regarding autonomous steering of the inclination, by varying the WOB, see section 9.4 as well as a selection of the code presented in Appendix D.

2.8 Software Architecture

The control system is based on multithreading where the programs or modules each communicate through the gRPC-API. In short, the control system short consists of a series of microservices running as parallel processes to distribute the work load among multiple threads on the 12-core CPU in place. With the exception of some variables that are made available to all microservices that subscribe to them, the dataflow in the control system is unidirectional, i.e. from the first layer (layer 1 - left), to the last (layer 7 - right). A prototype of the dataflow is illustrated in Figure 2.28:

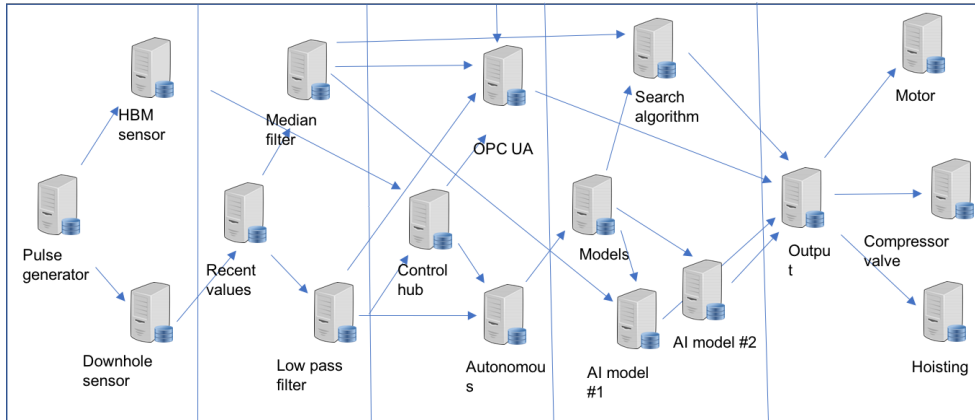


Figure 2.28: *Concept illustration of the uniflow gRPC modules that combined make up the control system on the rig.*

2.8.1 gRPC

gRPC is an open source Remote Procedure Call (RPC) developed by Google. RPC is a communication platform where the program (identified as the caller) performs remote operations on a server. As mentioned above, gRPC was initially launched by Google. The framework allows for communication between a variety of platforms and programming languages, and while traditional APIs share data through formats readable by humans, the gRPC uses protocol buffers for transparent communication between client and server applications. Protocol buffers are communication platforms that are not readable by humans. The user creates the server client implementation. This kind of structure is beneficial for many reasons, but the main one is that both size of buffer files are smaller and communication times are much higher than traditional communication platforms (Guggedal & Steinstø, 2019) [12].

After the final changes were made in accordance with the directional drilling strategy, the following figure to visualize the dataflow in the control system was made:

Uni-directional Control System Dataflow:

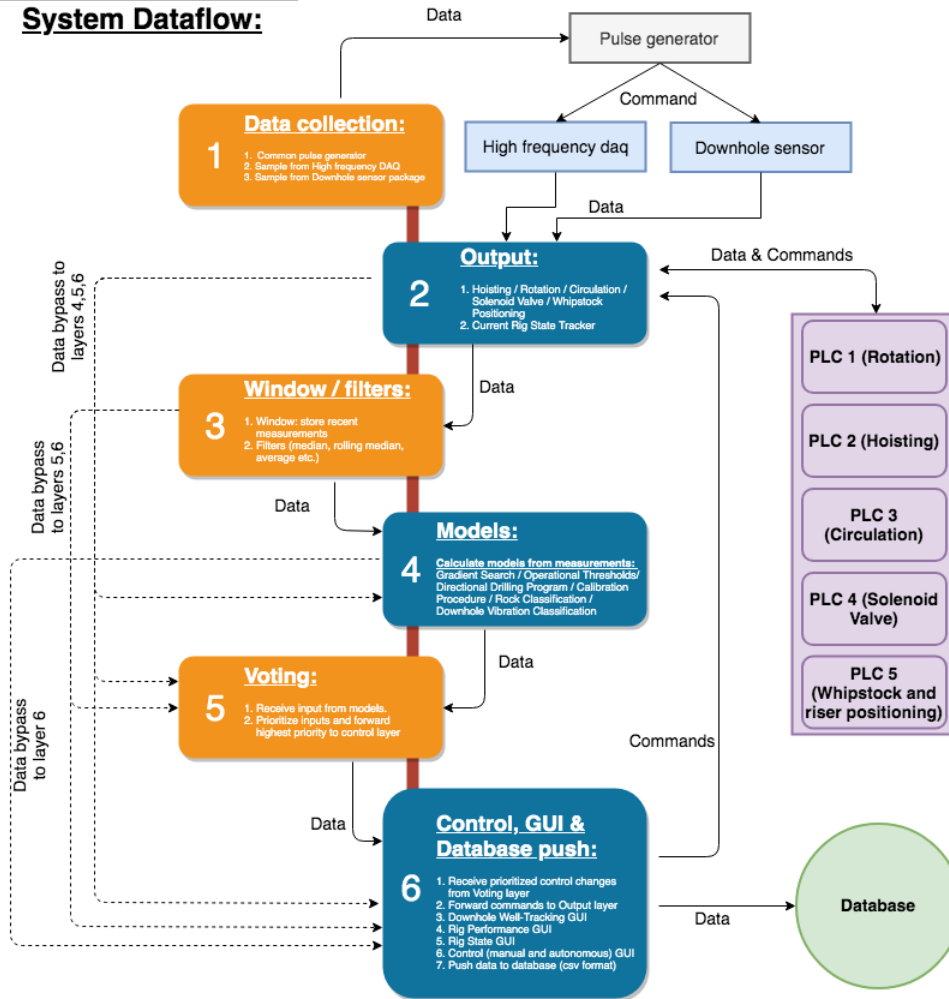


Figure 2.29: Implemented control system illustrating layers and dataflow internally and externally with PLCs and sensor equipment.

2.8.2 API - OPC UA

An Open Platform Communications Unified Architecture (OPC UA) was used for API implementation on the rig in order to integrate a plug-and-play (PnP) concept on the drilling rig. Using OPC UA and gRPC, a client can connect to the rig using the API as illustrated in Figure 2.30. This enables the option for the client to remotely execute two different types of interactions with the rig: request information such as drilling data or exert commands to control the rig (Steinstø & Guggedal, 2019) [12].

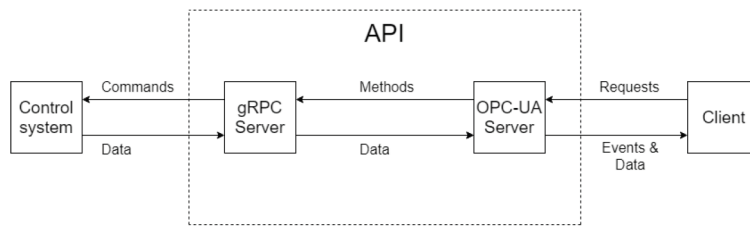


Figure 2.30: Illustration of the API (Steinstø and Guggedal, 2019) [12].

2.9 Graphical User Interfaces (GUIs)

Two GUIs have been developed in 2018 and 2019 for the laboratory scale drilling rig. These are a visualization GUI to track the progress of the autonomous control system and drilling performance Figure 2.31 and a downhole well trajectory environment, also showing the real-time position, inclination and orientation of the bit Figure 2.32.

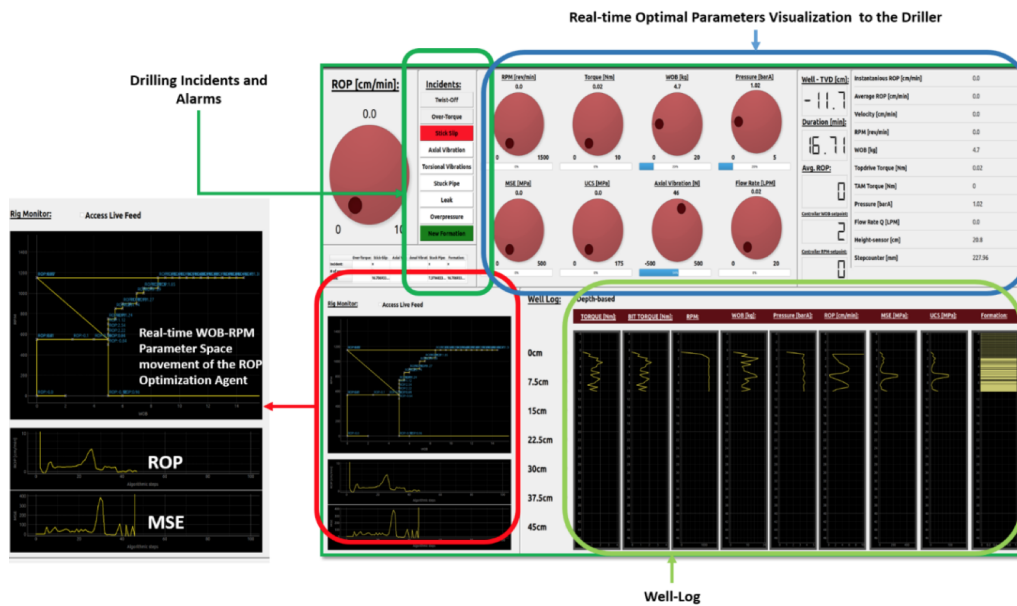


Figure 2.31: GUI is equipped with gauges and parameters showing the control system setpoints and sensor feedback as well as building a well log in real-time for interpretation of all stages of the drilling operation.

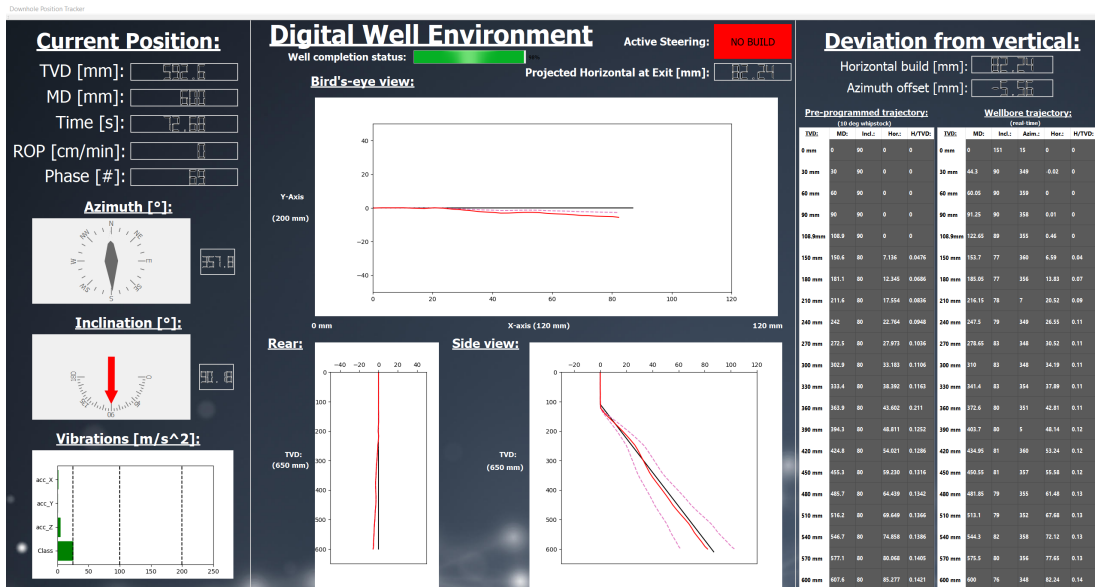


Figure 2.32: GUI consists of both real-time feedback from downhole sensor and downhole vibration model, as well as real-time tracking of the well trajectory position with respect to the pre-programmed trajectory to steer towards.

The GUIs have been created using PyQt and QtEditor and are located in their own gRPC module in the control system.

Chapter 3

Theory

3.1 Data Management

3.1.1 Data Mining

Data mining (DM), or extraction of new and useful knowledge from existing data, can be handled in several different ways and is an important step in the process of understanding a problem. It will depend on both the system and the complexity of it, and can be executed using several methods ranging from those being fully automatic to others requiring manual collection of for instance laboratory test results, that be either sensor information or observations. The data collected to develop models in this thesis was collected both automatically and manually using the test rig at the University while running experiments carefully put together. From the collected data, one can further process, analyse and identify hidden trends or patterns using different statistical models and algorithms.

An important aspect of data mining is the quality of the data gathered. Noisy data can be particularly challenging to work with because a model will only perform at the quality level of the data available to it. When collecting data, it is important that the data gathered can be used along with the existing pool of data from previous runs. If no data has previously been collected, it is important to establish good routines and to be consistent in all phases such as calibration, sensor ranges, operational thresholds and so on. This will ensure that the results are repeatable and comparable, and that datasets collected from different experiments can be used to cross-validate results obtained from previous experiments (Hastie et al., 2016) [24].

Another aspect to consider is the size of the dataset. With a large dataset, the computational time will increase making the prediction process a lot more time consuming. Selection of appropriate algorithms to complement the data available is therefore an essential part of the DM process. Figure 3.11 shows how different supervised learning methods work for different sets of data (Hastie et al., 2016) [24].

Usually only a small portion of the data gathered gets used for prediction. Even if many models can perform well with only a few features, because of domain knowledge one rarely can select the most optimal features or create new artificial ones just considering the data. Feeding the model with too much data could cause overfitting, which will make the model perform poorly when given new input data due to the model becoming overly complex, for instance if too many parameters get used. Another challenge is that the model should not only be able to predict, but also identify correlations between the different features of the data given (Hastie et al., 2016) [24].

The most popular models used for DM are the ones based on tree structures. These structures are the easiest models to build and usually do not require a lot of data preparation to work. They can mix numeric, categorical predictor values and missing data. The many advantages of prediction trees almost make them perfect models to use, but there is particularly in one area they all have a lack which is inaccuracy. Inaccuracy causes them to rarely provide a good enough prediction when compared to best achievable results based on the data available (Hastie et al., 2016) [24]. Tree structures and other models will get further discussed later in this chapter.

3.1.2 Data Quality

In order to develop accurate models that have applications for drilling and well engineering, major importance lies in ensuring that the data is of high quality. According to Good and Hardin, the following are common errors to statistical procedures (Good and Hardin, 2006) [25]:

- the same dataset is used for training (hypothesis) and validation (testing),
- data is collected that either describes the wrong phenomena or are neither random nor representative,

- the wrong variables get measured,
- either inappropriate or inefficient statistical methods get used,
- statistical software with inappropriate defaults gets used,
- the findings in the data are inadequately communicated,
- models get extrapolated outside of the range of observations, and
- failing to validate models.

Good and Hardin recommend that the following steps are carried out (Good and Hardin, 2006) [25]:

- review quality assurance reports,
- describe the dataset, i.e. calculate the maximum, minimum, standard deviation for all variables and compare these against predefined ranges, as well as produce box and whisker plots of the variables,
- remove duplicate values in the dataset,
- verify the physical units of the measured data,
- characterize the extent of, and remove missing data, as long as all populations still are encompassed in the samples,
- remove outliers only if the outliers are signs of poor data that are extreme values and well separated from the main set of observations,
- confirm that for each variable, a serial correlation is computed and a four-plot (containing time plot, lag plot, histogram and Normal Q-Q plot) is generated.

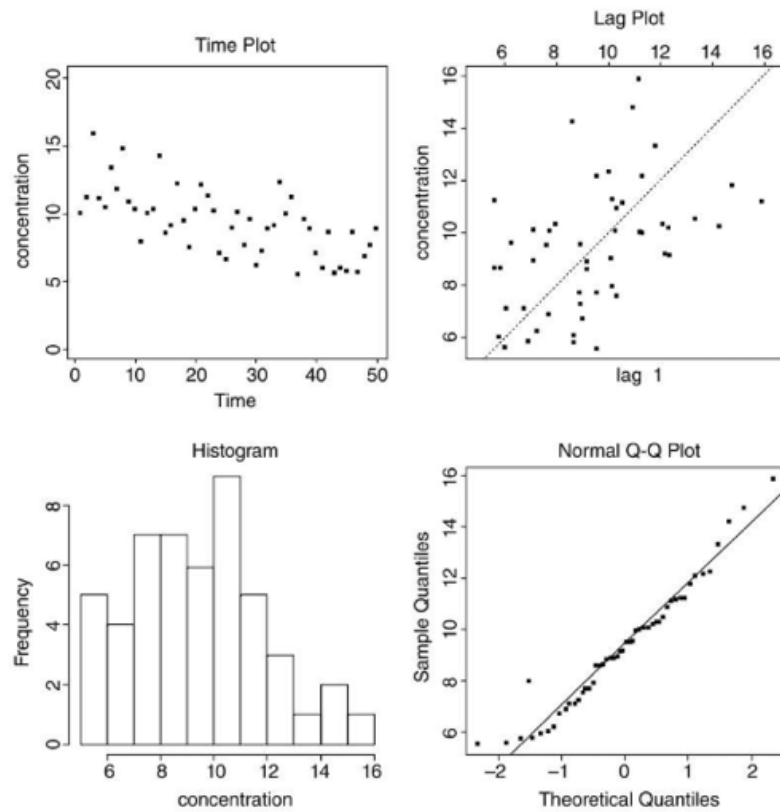


Figure 3.1: *Depiction of a typical Four-plot including a time plot, lag plot, histogram and normal Q-Q plot (Good and Hardin, 2006) [25].*

In the next sections, the theory behind the following data processing methods gets described:

- Data aggregation and collation,
- describing the data,
- noise reduction methods,
- data normalization and/or standardization,
- evaluating the feature importance,
- down sampling of data.

The methodology that has been used to process and thus ensure sufficiently high quality in our datasets that get used to train and validate the models is given in chapter 5.

3.1.3 Some challenges concerning data aggregating

Data is collected from multiple sources in real-time which causes some common challenges to merge the data. Since data could originate from either surface sensors, downhole sensors, control system outputs or manual inputs that describe the operation, all data should be synchronized with a common time reference. As an example, the clock time in every microcontroller or PC, varies slightly.

If the sampling frequency is low, for instance 10 Hz (10 samples per second), there is typically only a need to calibrate each system ahead of the operation and in regular intervals to prevent that the clock times get out of synch. If however one is working with data sampled at hundreds if not thousands of Hz (number of samples per second), only a small offset in synchronization could cause the data to become highly inaccurate once the data gets merged with data from other sources. One solution to this problem is to transmit a common pulse to all sensors or microcontrollers, requesting measurements from all sources simultaneously. Through measuring the time difference between each system's response to sending a package of data back to the control system, the delay in each system can be accounted for. This ensures that for instance a measurement that represents the bottom hole pressure (BHP), i.e. the pressure at the bottom of the well, at a given time can be compared to the stand pipe pressure (SPP) on surface.

Another challenge when aggregating data from several sources is calibration. Before data logging begins, all systems should get calibrated to ensure that data from each operation has the same base value, unit, and threshold. One example could be if the mud weight is given in kilograms per cubic meter (kg/m^3) for one experiment, while in the other the same variable is provided in pounds per gallon (ppg). Another example could be if the hook load gets measured and calibrated for one bottom hole assembly (BHA) configuration, and the hook load is not updated for another configuration for a later experiment or operation. In both cases, the data can be merged if the user is aware of the variations in units or BHA weight. There is however no way that the computer can automatically work with such differences in the data, unless the variations in the data are inserted as metadata that the computer can access and use to correct the data with, which should not be assumed.

Also, there might be a difference in terms of which data and when the data is collected for each operation. If for instance a sensor has failed between two logging

operations, the data in one of the datasets might be either missing, corrupt, or wrong. If for instance data is missing, the user can use techniques such as interpolation, or keep the last datapoint that was measured until a new measurement is received. If the data is corrupt, there might be a need to discard all data taken in a certain time interval. If the data is suspected to be wrong, one should consider to evaluate the standard deviation, maximum and minimum values, changes in the operation or drilling conditions that can explain the change or possibly discard the data due to the uncertainty. In the event of merging datasets that have been sampled at different frequencies, there might be a need to downsample the data to a common sampling frequency, or use a method such as interpolation to increase the number of samples for those sensors that output a lower sampling frequency than the others.

3.1.4 Importance of utilizing a database for data storage

During every drilling experiment, the sensors installed on the rig gather a large amount of timestamped data. These timestamped data are commonly known as time series data. There have been many drilling experiments conducted over the years and a lot of data has been collected, either in real-time using a Python script that saves real-time data received directly from Arduino Due microcontrollers or for post analysis using a high-frequency data acquisition (DAQ) module. In the first years, the data was stored in flat text files and there was a lack of organization. A large amount of the data lacked proper labeling or sorting. In 2017, the team that worked with the laboratory-scale drilling rig received help to implement a database in PostgreSQL.

The database and web application that in the past has served as a user interface for the database and hosted both the application and the database, was hosted externally. However, this database was implemented in a conventional relational database management system (RDBMS). Storing time series data in a RDBMS is not very efficient in the long term. There are more efficient database technologies out there that are specialized for this kind of task. Furthermore, the hosting infrastructure in use was not very reliable.

After a team member properly researched available solutions for time series data, the range of alternatives was narrowed down to two possible candidates; InfluxDB and TimescaleDB. Both are considered state of the art time series DB systems, but there were uncertainties as to which system would best fit the rig and data.

In order to compare the technologies, the two DB systems got installed in identical environments with identical datasets. Then, several queries were run against both databases and their execution time was measured. Based on the results, the team concluded that TimescaleDB is a faster and more reliable solution on the drilling rig. TimescaleDB was created by heavily modifying the architecture of PostgreSQL, a well-known RDBMS. It is offered as an extension of PostgreSQL and enhances its functionality and performance. TimescaleDB still preserves all the features and capabilities of a conventional RDBMS. Data is organized in tables with columns, rows, indexes, primary keys, foreign keys, constraints and all the desirable features usually reserved for relational databases. TimescaleDB satisfied both of our requirements where both leverages the features of relational database systems, and also offers great query optimization for time series data. This is convenient in order to not have to implement two different databases; one to store sensor data and another to store rig state data.

In order to easily manage the data that gets stored in the DB, an application that serves as the interface between the users and the DB is required. A separate small application is also required; responsible for pushing the data to the database. The sensors' signals are sampled at a specific frequency and these data are stored in text files on the computer. A short Python script gets executed on this computer, which reads the text files and pushes the data to the database. The script can either be executed manually by the conductor of the experiment, or it can be scheduled to be executed at a specific time of the day. For simplicity, the scheduling can be achieved by using Task Scheduler, which is a built-in job scheduler program of the Windows operating system.

To make the data in the DB easily accessible, a user-friendly platform to access the data was created in the team. This is solved by writing a web application, where team members can log in and manage the stored data. The user can access and download the data gathered from either specific experiments, or a particular subset of those data. The user can then choose to visualize the data by generating graphs, query experiments using different parameters, manage which variables that the database shall contain, register the drilled rocks (label the rock types, experiments, add comments etc.), manage access to the database, and so on.



Figure 3.2: *Illustration of how data is visualized after the user has selected which sensors and sample resolution to capture. The data can then be exported as a CSV-file.*

The back end of the DB was developed using the Python programming language. Django was used for the web framework. In the frontend, JavaScript, hypertext markup language (HTML) and cascading style sheets (CSS) got used. The frontend libraries and frameworks used are Bootstrap, jQuery, jQuery user interface (UI), Popper.js and Chart.js. The database is hosted at the University of Stavanger, while data can be pushed from any location by running a script that can also be downloaded from the web interface where the user accesses the DB. As is illustrated above in Figure 3.2, the user can select the experiment that was run from a list containing all uploaded experiments, and quickly visualize data from the sensors that shall get exported. In addition, the data can easily be downsampled to for instance 10% of the raw data. When working with datasets of millions of rows (measurements) and between five and thirty columns (sensors or features), this immediately allows the user to select which data-frequency to work with, rather than having to import a large dataset, carry out either a linear downsampling process and then save the data as a new file.

3.1.5 Downsampling the data

For post analysis, the time to work through prediction algorithms and other processes can be tremendous, and in some cases even cause computer crashes when working with large sets of data. In order to cope with this challenge, an important part of data processing is to downsample the data. Other reasons to

downsample the data with respect to our work, include:

- When working with the data, equally high results get obtained even when data is significantly downsampled from 9600 Hz to less than 100 Hz. Downsampling the data allows us to work with several large datasets combined rather than having to develop separate models, or only very specific objectives.
- It suits the control system architecture of the rig better because frequency of the control system is about 60-100 Hz, suggesting that models that get implemented have been developed at approximately the same frequency.

When down-sampling data several considerations must be taken into account. It is particularly important that necessary data is not lost when downsampling and therefore, a combination of both linear and random downsampling has been tested and evaluated. A discovered challenge with using linear downsampling in the model creation phase was that critical information such as for instance a vibration frequency could get removed. There is of course no guarantee that the same information can not get lost when using random downsampling by extracting a certain percentage of random samples from the dataset.

3.1.6 Describing the dataset

Before a dataset can be used to train a model, the user should aim to describe the data in the dataset by calculating the count from each data origin (to identify potential missing data), the mean, standard deviation, minimum value, maximum value, P25, P50, P75 and so on. This can be done by either using a function in for instance Python such as `dataset.describe()`:

	time	z1	z2	z3	rpm	m_torque	depth	wob	displacement
count	815269.000000	815269.000000	815269.000000	815269.000000	815269.000000	815269.000000	815269.000000	815269.000000	8.152690e+05
mean	42.461884	-1097.520157	-76.772746	32.243243	315.845504	0.478590	-1.225244	-1.142050	-5.238670e-07
std	24.515422	354.080421	375.071085	344.629256	6.671424	0.526790	4.146277	0.274559	1.259425e-07
min	0.000000	-2157.500000	-1227.600000	-1073.200000	296.050000	-0.390330	-11.827000	-2.279000	-1.045400e-06
25%	21.231000	-1379.200000	-375.000000	-248.540000	310.440000	-0.078799	-4.706100	-1.341700	-6.154400e-07
50%	42.462000	-1107.200000	-77.445000	29.065000	316.630000	0.540170	-1.126700	-1.175000	-5.389900e-07
75%	63.693000	-807.000000	215.050000	300.030000	320.890000	0.882180	2.317000	-0.953900	-4.375600e-07

Figure 3.3: *Various information about the data present in the dataset can be obtained by using the `.describe()` function in Python.*

Another way to interpret the data is to plot each of the data columns, or features, against each other, to visually inspect the distribution and ranges in the dataset.

Figure 3.4 below is obtained by plotting five different features against each other in a pairplot using MATLAB, where the three different colors represent three class labels, or in this case rock formations, that the observations belong to:

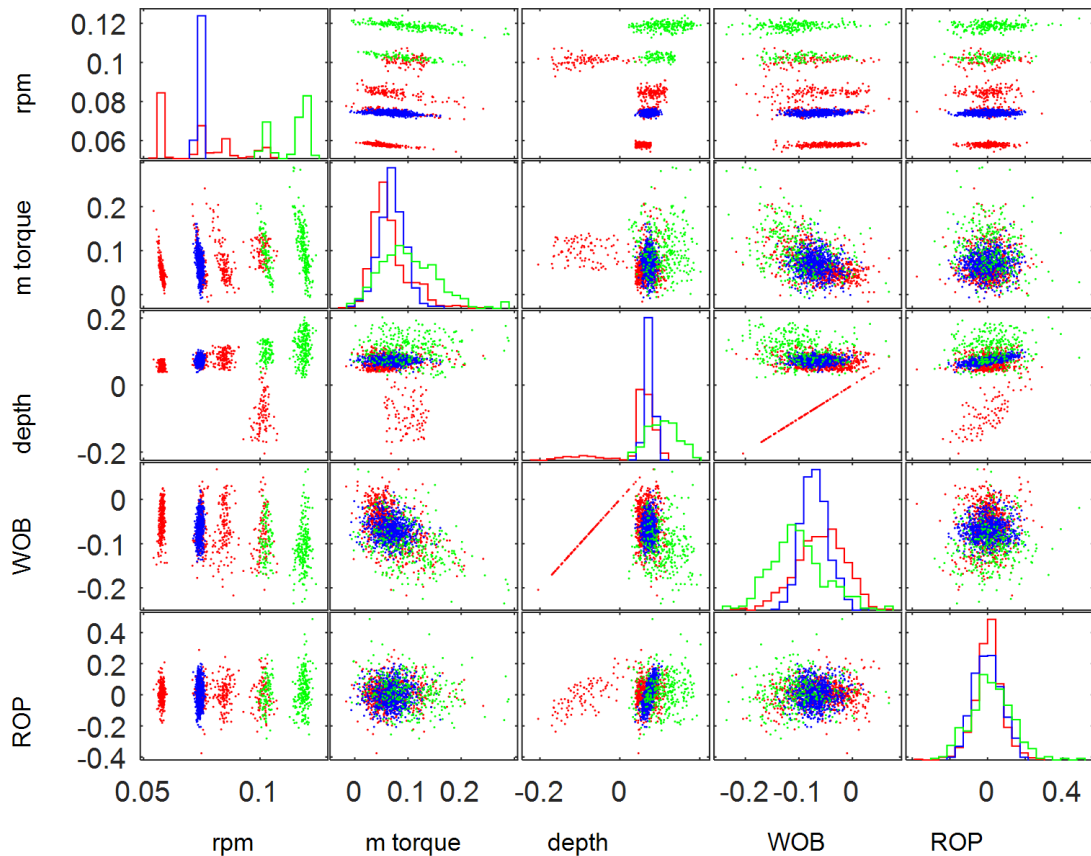


Figure 3.4: *MATLAB pairplot of data from five drilling variables (features). A histogram for each sensor is shown along the diagonal.*

As can be observed from the diagonal, the data distribution for each feature is given. If one wishes to inspect the distribution of a single feature or statistical properties of for instance different rock samples, a box plot or heatmap, as shown below, can provide valuable information that might not be easily interpretable by only considering the raw data:

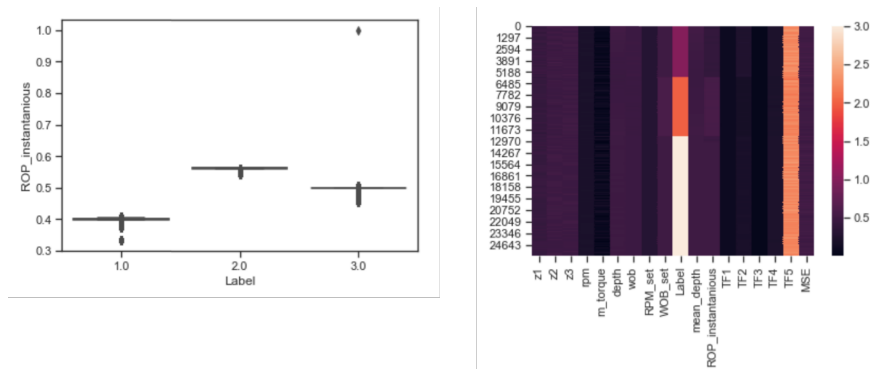


Figure 3.5: As illustrated, a box plot (left) shows the normalized values for rate of penetration (ROP) for three different formations, while a heatmap (right) shows distribution of sensor data for the same three formations.

In the left part of Figure 3.5 (box plot), the x -axis represents the class labels for the three different rock formations used in the pairplot created in MATLAB above. As can be observed in the box plot, an outlier separated from the rest of the observations, exists for the third label class.

3.1.7 Noise Reduction Methods

Before a model can be developed using machine learning, pre-processing and cleaning the dataset is essential. The results will only be as good as the data fed through the algorithm which means that one should be meticulous when preparing the data for analysis. Cleaning the data can include several steps including, but not limited to, outlier removal, removing invalid data, removing missing data, duplicates, and so on.

Invalid Data

Invalid data can cause issues when developing algorithms. For the drilling data captured using the laboratory drilling system, invalid data would typically be data measured outside of the specific sensors measurement range. For example, if torque in the top drive is measured in the 0 to 8.59 Nm range, an invalid measurement would be 11 Nm. For the case of missing data, using a programmed script all measurements outside of the allowed range gets overwritten with Not a Number (NaN). Then, the amount of NaN in the dataset got counted before a decision was made as to whether the experiment would have to be repeated or whether the entire row of data from all variables would get dropped (in order to have the same amount of data in each column of measurements).

```

1 Actualdataframe['z1'].where((Actualdataframe['z1'] >= -10000), inplace=True)
2 Actualdataframe['z1'].where((Actualdataframe['z1'] <= 10000), inplace=True)
3 Actualdataframe['z2'].where((Actualdataframe['z2'] >= -10000), inplace=True)
4 Actualdataframe['z2'].where((Actualdataframe['z2'] <= 10000), inplace=True)
5 Actualdataframe['z3'].where((Actualdataframe['z3'] >= -10000), inplace=True)
6 Actualdataframe['z3'].where((Actualdataframe['z3'] <= 10000), inplace=True)
7 Actualdataframe['rpm'].where((Actualdataframe['rpm'] >= 0), inplace=True)
8 Actualdataframe['rpm'].where((Actualdataframe['rpm'] <= 1500), inplace=True)
9 Actualdataframe['m_torque'].where((Actualdataframe['m_torque'] >= 0), inplace=True)
10 Actualdataframe['m_torque'].where((Actualdataframe['m_torque'] <= 8.59), inplace=True)
11 Actualdataframe['wob'].where((Actualdataframe['wob'] >= -30), inplace=True)
12 Actualdataframe['wob'].where((Actualdataframe['wob'] <= 30), inplace=True)
13 Actualdataframe['ROP_instantaneous'].where((Actualdataframe['ROP_instantaneous'] >= -50), inplace=True)
14 Actualdataframe['ROP_instantaneous'].where((Actualdataframe['ROP_instantaneous'] <= 50), inplace=True)

```

```

1 Actualdataframe.isnull().sum()

```

z1	0
z2	0
z3	0
rpm	0
m_torque	9340
depth	0
wob	0
RPM_set	0
WOB_set	0
Label	0
mean_depth	0
ROP_instantaneous	288
dtype:	int64

Figure 3.6: First, all values that fall outside of the allowed range get treated as false data and is replaced by NaN. The amount of rows with invalid data is then counted.

```

1 Actualdataframe.head()

```

	z1	z2	z3	rpm	m_torque	depth	wob	RPM_set	WOB_set	Label	mean_depth	ROP_instantaneous
0	-1960.6	306.63	531.93	315.60	1.148300	5.9121	-1.12210	450	0	1	5.765513	NaN
1	-1667.7	174.53	254.23	323.22	0.538340	5.9655	-1.23900	450	0	1	5.765513	NaN
2	-1615.6	155.26	153.97	304.94	NaN	6.2187	-1.30640	450	0	1	5.765513	NaN
3	-1787.7	342.73	-256.62	317.86	0.985950	5.2278	-1.70160	450	0	1	5.765513	NaN
4	-1218.5	399.87	-129.15	323.12	0.016962	6.0884	-0.94777	450	0	1	5.765513	NaN

```

1 Actualdataframe.dropna(axis=0, inplace=True)

```

```

1 Actualdataframe.head(10)

```

	z1	z2	z3	rpm	m_torque	depth	wob	RPM_set	WOB_set	Label	mean_depth	ROP_instantaneous
96	-723.32	-674.390	88.57	321.44	0.64285	5.5500	-1.30910	450	0	1	5.583878	-10.898066
98	-1201.30	-699.990	399.66	303.10	0.92139	5.4997	-1.50160	450	0	1	5.583878	-10.898066
99	-1236.10	-638.520	664.90	316.79	0.72538	5.4603	-1.20980	450	0	1	5.583878	-10.898066
101	-1524.40	-142.120	790.12	307.89	1.50870	5.3185	-0.87644	450	0	1	5.583878	-10.898066
102	-1757.20	-126.380	630.82	322.64	0.55122	5.6286	-1.25280	450	0	1	5.583878	-10.898066
104	-1822.50	73.649	283.28	312.18	1.14920	5.7985	-1.46550	450	0	1	5.583878	-10.898066
105	-1802.80	70.340	348.80	319.70	0.53130	5.8317	-1.38360	450	0	1	5.583878	-10.898066
107	-1644.50	489.300	-133.50	315.69	0.83384	5.5062	-1.28870	450	0	1	5.583878	-10.898066
108	-1364.20	359.060	-173.60	320.60	0.24383	5.7238	-1.17870	450	0	1	5.583878	-10.898066
109	-1129.10	225.790	-370.54	305.13	0.20701	5.1211	-1.27380	450	0	1	5.583878	-10.898066

Figure 3.7: Since only a small portion of the dataset in the figure above contains invalid measurements, all rows that contain NaN gets removed.

If a significant part of the dataset falls outside of the validity range, an approach can be to rather than replacing the values with NaN and later remove the complete row of observations, write the value to be either a very high or very low number. One could then keep measurements from the other variables (sensors) in the dataset, but simply discard the measurements in the single variable where invalid data is present.

Other measurements that get removed are NaN values that occur for different reasons. From the data extracted from drilling with the laboratory drilling rig,

less than 5 % of the dataset will typically contain invalid values after the outside of range- and NaN-check is performed.

Missing Data

There can be a number of reasons for why data is missing in a dataset. One common cause is when different sensors get sampled with varying sampling frequencies, for instance 10 Hz for one sensor and 20 Hz for another. Another common cause could be hardware (electrical) failure, where the signal is lost for a short duration of time. A third cause could be that the data is held up in the buffer where the PC stores the data short-term before it gets used.

To handle missing data, two common techniques can be used. These are interpolation or filling the data with the last known value. Three common interpolation techniques that can be used are linear, quadratic or cubic (Bakri et al., 2014) [26].

For linear interpolation, a straight line is drawn between two points so that (Bakri et al., 2014) [26]:

$$F_1(x) = b_0 + b_1(x - x_0) \quad (3.1)$$

If more than two points exist, higher polynomial orders such as quadratic, cubic etc. can be used:

$$F_2(x) = b_0 + b_1(x - x_0) + b_2(x - x_0)(x - x_1), \dots, b_n(x - x_0)(x - x_1)(x - x_n) \quad (3.2)$$

b_0, \dots, b_n can be found from equations;

$$b_0 = f(x_0) \quad (3.3)$$

$$b_1 = \frac{f(x_1) - f(x_0)}{x_1 - x_0} \quad (3.4)$$

$$b_n = \frac{\frac{f(x_n)-f(x_{n-1})}{x_n-x_{n-1}} - \dots - \frac{f(x_1)-f(x_0)}{x_1-x_0}}{x_n - x_0} \quad (3.5)$$

An approach to evaluate which interpolation technique that is most optimal to fill the missing data could be to fit the data to a particular distribution. One can then use various performance indicators to evaluate the error between the interpolation-filled value and the actual data. One performance indicator could be to calculate the Root Mean Squared Error (RMSE) (Bakri et al., 2014) [26]:

$$RMSE = \left(\frac{1}{N} \times \sum_{i=1}^n [P_i - O_i]^2\right)^{\frac{1}{2}} \quad (3.6)$$

Where; N denotes the number of imputations (substitutions of missing data), O_i the actual measurement and P_i the interpolated data point.

Outlier Removal - Interquartile Range (IQR)

Outliers are data that are situated away from the main observation window. An important factor to consider before removing outliers is to find out whether they consist of relevant information or are the result of noise, i.e. can you remove the outliers without deleting valuable information from the data? In some datasets, for example when dealing with kick detection or stuck pipe detection, the important information could be apparent in the outlying points. In our research several techniques have been evaluated for optimal outlier removal. The Interquartile range method (IQR-method) has been identified as most optimal when dealing with outliers in the drilling data. Using this technique, the Q1 and Q3 quartiles must first be defined. The IQR range, which gives the middle 50 % of the data, is then defined as follows (Holdaway, 2014) [27]:

$$IQR = Q3 - Q1 \quad (3.7)$$

Then, using the following two relations, one can find the lower and upper windows that contain outliers according to the method (Holdaway, 2014) [27]:

$$\text{Lower Range} = Q1 - 1,5 * IQR \quad (3.8)$$

and

$$\text{Upper Range} = Q1 - 1,5 * IQR. \quad (3.9)$$

3.1.8 Normalization and standardization of the data

Linear Feature Scaling (LFS)

When working with drilling data where the variables or features originate from different sources or sensors, an important task is to scale all data to a common unit range, ideally so that data that is normal distributed gets represented as values from 0 to 1. This can be achieved through performing a Linear Feature Scaling (LFS), by considering the minimum and maximum value of each variable (James, 2016) [28].

For a set of values $x_j = [x_{1j}, x_{2j}, \dots, x_{nj}]$ where $a = \min(x_j)$ and $b = \max(x_j) - \min(x_j)$:

$$f(x) = \frac{x-a}{b} = \frac{x-\min(x_j)}{\max(x_j)-\min(x_j)} \quad (3.10)$$

While LFS provides a sensible method to scale data that have no predefined range, this technique could still cause a challenge if a significant outlier is present in the set of values. The outlier, which could be either very large or very small, would then cause the rest of our values to be skewed either towards 0 or 1, even if the original values had a common distribution (James, 2016) [28].

Another commonly used technique is standardization, which refers to the process of subtracting the mean value of the set of values for a variable from each measurement and dividing by the standard deviation of the set of values (James, 2016) [28].

$$f(x) = \frac{x-\bar{x}}{\sigma(x)} = \frac{x - \frac{\sum_{i=1}^n x_{ij}}{n}}{\sqrt{\sum_{i=1}^n \frac{(x_{ij}-\mu)^2}{(n-1)}}} \quad (3.11)$$

Where σ represents the standard deviation, \bar{x} the mean value and μ the true mean value for the set.

Rank Scaling

If one only needs to know the relative size of the data, for instance if a value is relatively high or relatively low, rank scaling can be a useful technique. If one considers a set of values where $O_j(x_{ij}) = 1$ represents the highest value, $O_j(x_{ij}) = 2$ represents the second highest value and so on, every value in each variable can be transformed into a score from 0 to 1 by setting (James, 2016) [28]:

$$f(x) = \frac{m - O_j(x)}{m - 1} \quad (3.12)$$

If for instance the transformed measurement would have a score of 0.5, this would suggest that the measurement is higher than 50 % of the measurements in the set of values that has been considered (James, 2016) [28].

Normalizing drilling data in practice

For our case, we are only interested in representing the measurements for each variable relative to the threshold that each system can measure. If we first consider the case of weight on bit, or hook load, we know that even if the system is capable of providing a hook load of approximately -400 N (tension) to 400 N (compression), the load cells are configured to measure -300 (compression) to 300N (tension) of force. Therefore, the first step of processing the data would be to remove all measurements where the data is invalid, leaving only those measurements where the hook load is within the (-300 N, 300 N) range. In terms of normalization however, there is no guarantee that a measurement representing the absolute maximum and minimum values in the sensor threshold occurs, and in real-time, there would be no way to transform the data. This is handled by adding a hardcoded maximum and minimum value for each variable.

As is illustrated in the Figure 3.8 below, two rows of hardcoded maximum and minimum values get added at the end of the dataset. These are user defined boundary conditions for each variable, in order to ensure that a measurement of for instance 750 revolutions per minute in the top drive corresponds to 0.5 (if 0.0 corresponds to 0 RPM and 1.0 to 1500 RPM).

```

1 Actualdataframe.describe()

```

	z1	z2	z3	rpm	m_torque	depth	wob	RPM_set	WOB_set	Label	mean_depth
count	25927.000000	25927.000000	25927.000000	25927.000000	25927.000000	25927.000000	25927.000000	25927.0	25927.000000	25927.000000	25927.000000
mean	-1207.651913	-116.189194	-40.002206	315.627460	0.818051	-1.579851	-1.363843	450.0	1.294789	2.297026	-1.417203
std	352.978169	363.704976	329.585275	7.262265	0.412164	2.672576	0.302466	0.0	2.190354	0.807987	2.571152
min	-2400.700000	-1370.300000	-1132.700000	295.310000	0.000016	-11.241000	-2.889300	450.0	0.000000	1.000000	-8.381292
25%	-1480.000000	-419.365000	-327.745000	309.485000	0.542700	-2.366900	-1.460650	450.0	0.000000	2.000000	-1.878074
50%	-1230.700000	-124.540000	-36.716000	317.990000	0.796720	-0.987970	-1.337600	450.0	0.000000	3.000000	-0.836003
75%	-900.905000	192.415000	228.495000	321.160000	1.114250	-0.423415	-1.224800	450.0	5.000000	3.000000	-0.766559
max	-64.974000	1023.000000	1090.600000	336.910000	2.001500	6.030000	-0.145180	450.0	5.000000	3.000000	5.583878

```

1 lengthofdf = len(Actualdataframe['z1'])
2 print(lengthofdf)
25927

1 # adding a row with hardcoded sensor maximum values:
2 # z1, z2, z3, RPM, Torque_surface, depth, WOB, RPM_setpoint, WOB_setpoint, Label, mean_depth, ROP
3 Actualdataframe.loc[lengthofdf + 1] = [10000, 10000, 10000, 1500, 8.59, 100, 30, 1500, 30, 3, 100, 50]
4 # adding a row with hardcoded sensor minimum values:
5 Actualdataframe.loc[lengthofdf + 2] = [-10000, -10000, -10000, 0, 0, -100, -30, 0, -30, 3, -100, -50]

```

Figure 3.8: Two rows of maximum and minimum values are added as additional rows at the end of the dataset.

After a minimum maximum scaling operation has been run in order to normalize the data, we can check and verify that the last two rows of data (where hardcoded boundary conditions for each variable was added) is now 1.0 (corresponding to the maximum boundary) for the first row and 0.0 (minimum boundary) for the second row.

```

1 Actualdataframe.tail()

```

	z1	z2	z3	rpm	m_torque	depth	wob	RPM_set	WOB_set	Label	mean_depth	ROP_instantaneous
25924	-1344.8	-377.170	372.12	325.15	0.94580	-1.32310	-1.3499	450.0	0.0	3.0	-0.741451	0.021876
25925	-1429.9	-191.930	168.44	308.39	1.40230	-0.39146	-1.4534	450.0	0.0	3.0	-0.741451	0.021876
25926	-1562.3	72.318	152.27	318.93	0.60947	-0.81501	-1.3377	450.0	0.0	3.0	-0.741451	0.021876
25928	10000.0	10000.000	10000.00	1500.00	8.59000	100.000000	30.0000	1500.0	30.0	3.0	100.000000	50.000000
25929	-10000.0	-10000.000	-10000.00	0.00	0.000000	-100.000000	-30.0000	0.0	-30.0	3.0	-100.000000	-50.000000

```

1 from sklearn import preprocessing

1 x = Actualdataframe.values #returns a numpy array
2 min_max_scaler = preprocessing.MinMaxScaler()
3 x_scaled = min_max_scaler.fit_transform(x)
4 Actualdataframe1 = pd.DataFrame(x_scaled)
5 Actualdataframe1.head()
6 Actualdataframe1.tail()

```

	0	1	2	3	4	5	6	7	8	9	10	11
25924	0.432760	0.481141	0.518606	0.216767	0.110105	0.493385	0.477502	0.3	0.5	1.0	0.496293	0.500219
25925	0.428505	0.490403	0.508422	0.205593	0.163248	0.498043	0.475777	0.3	0.5	1.0	0.496293	0.500219
25926	0.421885	0.503616	0.507614	0.212620	0.070951	0.495925	0.477705	0.3	0.5	1.0	0.496293	0.500219
25927	1.000000	1.000000	1.000000	1.000000	1.000000	1.000000	1.000000	1.0	1.0	1.0	1.000000	1.000000
25928	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	0.0	0.0	1.0	0.0	0.000000	0.000000

From the above, it is evident that the two last rows we inserted as max/min-thresholds have become 1.00 and 0.00.

Figure 3.9: The dataset is normalized by running the `MinMaxScaler` function from the `Scikit Learn` library in Python [29]. The red boxes indicate the rows that were added in Figure 3.8.

This technique is especially helpful when trying to correlate drilling data that is captured with one configuration, or one set of sensors with data captured using a different set of sensors, by using the same threshold for all cases.

3.2 Machine Learning Theory

The methodology used to develop and evaluate the different machine learning models is illustrated in Figure 3.10:

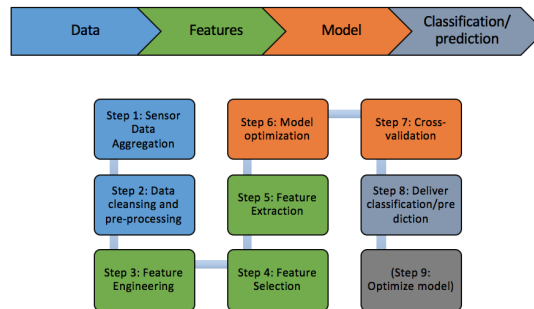


Figure 3.10: *Illustration of the seven steps that go into the development of machine learning models used for various classification objectives.*

3.2.1 A Short Introduction to Supervised Learning

When developing a method for supervised learning we use a model-based approach. In supervised learning there is an observation, x_i , followed by a response, y_i . The goal is to train a given algorithm into predicting a response when given the observation, x_i . Other uses could be to simply have better understanding of the link between the observation and response (Hastie et al., 2014) [30]. Supervised learning is commonly used for several different tasks ranging from engineering applications to cultural behavior.

When build a model, an important part of making the algorithm accurate is to train it for the same conditions or environment that you want the model prediction to work in. For example, if you want to predict at which dates a given plant will start to flower during summer, you could have a dataset with historical information about when the plant will blossom. If you then want your prediction model to give realistic estimates where you live, it's important to train the model using data that is representative for the same location, or at least approximately the same conditions. The same will apply for formation classification and challenges considered in this study. If the data is based on laboratory measurements taken under laboratory conditions, the algorithm will likely score the highest on laboratory data taken in the same conditions, unless the physics and conditions of the real drilling scene can be replicated for the experiment.

Supervised learning is divided into two groups; classification and regression. Which group a classification objective falls under depends on whether the data is categorical or numerical. As an example, we will demonstrate a numerical case, as this is of bigger relevance to the scope of classification in this thesis.

Linear regression is a commonly used method to identify the correlation between numerical data. For such a case, the model is given the response value y and the predictor variables x_1, x_2, \dots, x_n . Using these values, one can create a linear model, \hat{y} given by equation 3.13 (van der Aalst, 2016) [31]:

$$\hat{y} = f(x_1, x_2, \dots, x_n) = a_0 + \sum_{i=1}^n a_i x_i \quad (3.13)$$

For each value, a given error $|y_j - \hat{y}_j|$ can be calculated. Furthermore, the sum of squared errors, J , can be calculated using equation 3.14 for m instances:

$$J = \sum_{j=1}^m (y_j - \hat{y}_j)^2 \quad (3.14)$$

This technique is just one of several approaches that can be used to solve regression problems using supervised learning (van der Aalst, 2016) [31].

3.2.2 A Short Introduction to Unsupervised Learning

As described previously, supervised learning works by analysing a dataset consisting of observations, x_i , that all have a response, or class, y_i . What differs when it comes to unsupervised learning is that it works only by analysing the list of observations, x_i , and areas of application can include finding hidden layers, visualizing the data better or identifying hidden structures or patterns in the observations (Hastie et al., 2014) [30].

Furthermore, unsupervised learning is much more challenging than supervised learning because there is no real answer to the problem since there is no response, y_i , to the observations, x_i (Hastie et al., 2014) [30].

Unsupervised learning is of growing interest since the methods are applicable in most settings where statistics are important. Where a model in only a short

duration of time can be used to organize vast sets of data, consisting of millions if not billions of observations, it would take a human a lifetime to analyse and organize it, if even possible. An examples of an application for unsupervised learning is a cellphone application that can suggest when you should go grocery shopping based on when the activity level or queue in your local store is low. Such suggestion can be created by analysing and organizing data for when other customers typically visit the same store, and the level of complexity behind such analysis is only limited by data available. Another example of an application can be to identify genetic correlations in a population of people that all have a specific kind of cancer, and to finetune their treatment using a specifically tailored recipe for medication (Hastie et al., 2014) [30].

For drilling, an example of an application using unsupervised learning is to identify the most optimal WOB for a well section to ensure a high drilling efficiency, for instance by evaluating the WOB setpoints where the mechanical specific energy (MSE) is low.

There are several different ways of identifying subgroups, for instance visualizing or simplifying the data. Common techniques include Principal Component Analysis (PCA) and clustering. PCA is used to find low-dimensional representations of the data that explain the variance of the data properly, while clustering will locate homogenous subgroups in the dataset (Hastie et al., 2014) [30].

Principal Component Analysis

PCA is a method of analysing small or large datasets. Whether only a few hundred or even millions of observations exists, PCA method extracts the numerical values from the variables and calculates a set of new orthogonal variables called principal components, hence the name Principal Component Analysis. The benefit of using this method is to extract only the required information to explain the variance in the data and thus reduce the size of the dataset by keeping only the valuable information required for prediction and classification. After creating the principal components, the quality of the model can be evaluated by cross validation (Hervé and Williams, 2010) [32].

The principal components are found using Singular Value Decomposition (SVD) and depend on the eigen-decomposition of semidefinite matrices. This is done by finding the components by SVD of the data table \mathbf{X} . The table that is to be analysed by PCA consists of I observations that belong to J variables. The

matrix \mathbf{X} is described as $I \times J$. The matrix \mathbf{X} has a rank L where $L \leq \min\{I, J\}$. The matrix, \mathbf{X} , then has the following SVD (Hervé and Williams, 2010) [32]

$$\mathbf{X} = \mathbf{P}\Delta\mathbf{Q}^T \quad (3.15)$$

where \mathbf{P} is the $I \times L$ matrix and \mathbf{Q} is the $J \times L$ matrix of the right singular vectors, while Δ gives the diagonal matrix of the singular values. Then, using the components obtained from the SVD of the data in table \mathbf{X} , where \mathbf{X} is given by Equation 3.15 the relation $I \times L$, given by \mathbf{F} , is obtained by the following equation (Hervé and Williams, 2010) [32]:

$$\mathbf{F} = \mathbf{P}\Delta \quad (3.16)$$

Furthermore, \mathbf{Q} gives the coefficients of linear combinations needed to calculate the coefficient scores. It can also be used to find the projections from the observations on the principal components. This is done by multiplying \mathbf{X} and \mathbf{Q} , i.e. combining equation 3.15 and 3.16 (Hervé and Williams, 2010) [32], as is shown in 3.17 below:

$$\mathbf{F} = \mathbf{P}\Delta = \mathbf{P}\Delta\mathbf{Q}^T\mathbf{Q} = \mathbf{X}\mathbf{Q} \quad (3.17)$$

Clustering

Clustering is a general term and there exists various clustering methods. The concept behind clustering is to group the data into so-called clusters so that a given observation (point) within a cluster has the same properties as the surrounding points. When dealing with clustering, one will typically have n different samples that are each represented by a number of p observations. One can expect some sort of heterogeneity in the samples which means that there also could be unknown subgroups within the data. Two different methods of clustering, k-means and density-based spatial clustering of applications with noise (DBSCAN) are presented in section 3.4.

3.2.3 Training and Cross Validation

A common way to develop a supervised machine learning model is to randomly divide a set of observations into two pieces. One piece (for instance 80 % of the

original data) can be used to train the model, while the other (in this case the remaining 20 % of the original data) can be used to validate the model and its accuracy by carrying out predictions on the validation set of observations and compare them with the class label that the observations truthfully belong to (van der Aalst, 2016) [31] (DeepAI, viewed 25.03.2019) [33].

A common mistake when splitting the dataset into a train set and a test set is to not pick random samples. By not doing so, a trend which might exist in the testing set, but not in the training set can be erroneously predicted (since the model is not trained for such), which in turn can lead to poor performance when evaluating the model accuracy. One can also risk that the entirety of the population is not represented in the training data, causing the model to be inadequately trained, since a phenomena that has been captured in the observations and original dataset does not make it to the training set.

3.3 Supervised Machine Learning Models

3.3.1 Selecting the most optimal model

Selecting an algorithm

The process of selecting an algorithm, optimizing the input parameters for the model, training the model, evaluating the model performance (score) and finally cross validating the model on a fresh dataset may seem like a tedious and complicated process. Fortunately, if selecting the correct programming language and libraries, the process can become both quick and rather simple. Python is by many viewed as the go-to programming language for machine learning, due to its richness in available libraries for algorithms and scripts. Python will therefore be used from chapter 4 and forward as an alternative to research that in the past has been performed using MATLAB.

According to (Hastie, Tibshirani and Friedman, 2016) [24], the following table illustrates characteristics of some different learning methods, to give an indication on when the different models are applicable.

Key: ▲ = good, ◆ = fair, and ▼ = poor.

Characteristic	Neural Nets	SVM	Trees	MARS	k-NN, Kernels
Natural handling of data of “mixed” type	▼	▼	▲	▲	▼
Handling of missing values	▼	▼	▲	▲	▲
Robustness to outliers in input space	▼	▼	▲	▼	▲
Insensitive to monotone transformations of inputs	▼	▼	▲	▼	▼
Computational scalability (large N)	▼	▼	▲	▲	▼
Ability to deal with irrelevant inputs	▼	▼	▲	▲	▼
Ability to extract linear combinations of features	▲	▲	▼	▼	◆
Interpretability	▼	▼	◆	▲	▼
Predictive power	▲	▲	▼	◆	▲

Figure 3.11: *Characteristics of different machine learning algorithms [24].*

As has been mentioned in the previous sections, different use cases require different approaches. Before one can simply select an algorithm, several important questions should be addressed, that combined can help determine which approach to take. First and foremost, the user should determine what type of (drilling) problem he or she is addressing. If the objective is to predict values such as well pressure or to forecast the production from a well, a regression model that estimates the relationship between different variables is likely the correct approach. If the aim of the model is to classify different rock formations that have been drilled, or the size of cuttings that is returned to surface, a multiclass classification model would allow the drilling engineer to predict which category the data or sample falls within. If on the other hand, the objective is to identify an anomaly that differs from the normal drilling state or to analyse the structure in the data, a clustering or anomaly detection method can be adopted.

Secondly, the user should consider the size, distribution, quality and type of data that is available. What sampling frequency is typical, and is the same sampling frequency available for all data? Can downhole measurements be accessed such as gamma ray logging that measures the emitted gamma rays, or acoustic logs that measure the interval transit time through the formation or is only surface data

such as the free hanging weight and surface RPM available? Is the data that will be used to train and develop the models accurately describing each of the phenomena that the model should predict in the future? The size of the dataset will decide both the requirements to computational power and the likelihood of overfitting (producing a too specific model that only describes the particular dataset) or underfitting (even failing to produce a model that can accurately predict on the original training data, thus also making it impossible to produce estimations or predictions for new sets of data). Finally, one must weigh the need for speed versus the complexity and amount of data against each other; classification or forecasting. This applies to both training the model and using the already trained model to produce a prediction. Some models such as multiclass neural networks can be very accurate, however require long training times. Others, such as decision tree models could retain a high accuracy yet have much faster training times.

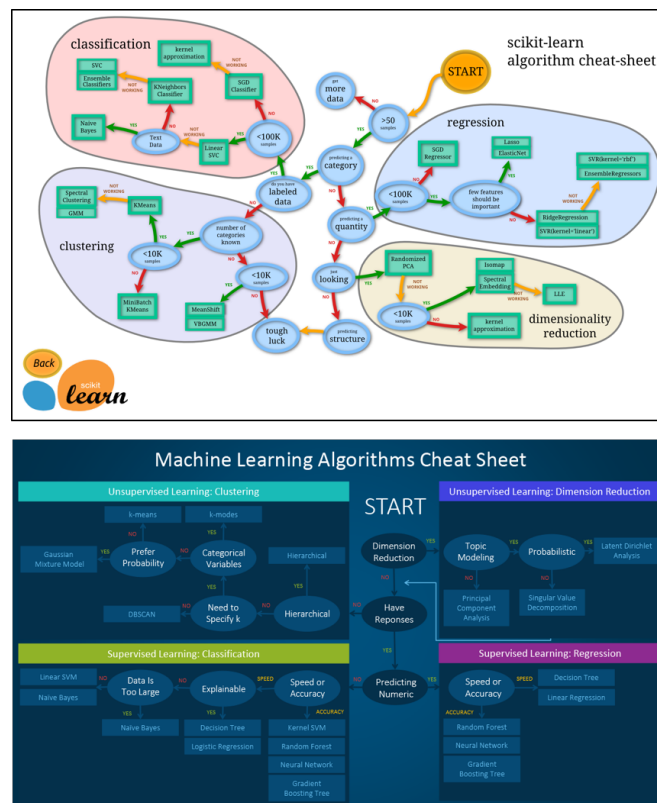


Figure 3.12: *Scikit learn (upper figure) [34] and SAS (bottom figure) [35] machine learning cheat sheets to select the most optimal algorithm.*

One approach to determine which model to select once the aforementioned topics have been addressed is to use a so-called cheat sheet. Cheat sheets allow the user to very quickly determine what model is possibly appropriate to solve a particular

problem, rather than having to thoroughly read up on the theory behind each model, its pro's and con's and field of application. Please note that some cheat sheets often vary slightly from each other, and should only be used as an initial consideration.

Determining the model input parameters

Once the most optimal algorithm has been selected, one needs to define which parameters to use to tune the model. This can be done either theoretically or through a data-driven approach where multiple combinations of different parameter setpoints get tested and evaluated.

In the scikit learn software, two data-driven approaches are provided to tune the parameters of a classifier algorithm, either randomly from the parameter space by specifying the distribution, or exhaustively by defining which parameters and ranges to evaluate. These are Exhaustive Grid Search (GridSearchCV) and Randomized Parameter Optimization (RandomizedSearchCV). Those parameters that do not get defined will be left as the default setting in the model. [36].

```

1 from sklearn.model_selection import GridSearchCV
2 from sklearn.metrics import classification_report
3
4 params_to_test = {
5     'hidden_layer_sizes': [(50,50,50), (100,100,100), (200,)],
6     'activation': ['tanh', 'relu'],
7     'solver': ['sgd', 'adam'],
8     'alpha': [0.0001, 0.1],
9     'learning_rate': ['constant', 'adaptive'],
10 }
11
12 #here you can put any parameter you want at every run, like random_state or verbosity
13 MLP_model = MLPClassifier(random_state=None)
14
15 ## comment: "If int, random_state is the seed used by the random number generator; If RandomState instance,
16 ## random_state is the random number generator; If None, the random number generator is the RandomState
17 ## instance used by np.random."
18
19 #here you specify the CV parameters, number of folds, number of cores to use...
20 grid_search = GridSearchCV(MLP_model, param_grid=params_to_test, cv=10, scoring='f1_macro', n_jobs=4)
21
22 grid_search.fit(x_train, y_train)
23
24 best_params = grid_search.best_params_
25
26 #best_params is a dict you can pass directly to train a model with optimal settings
27 best_model = grid_search.best_estimator_
28 print(best_model)
29
30 # self.max_iter, ConvergenceWarning)
31 /anaconda3/lib/python3.6/site-packages/sklearn/neural_network/multilayer_perceptron.py:564: ConvergenceWarning: Stoch
32 astic Optimizer: Maximum iterations (200) reached and the optimization hasn't converged yet.
33 # self.max_iter, ConvergenceWarning)
34 /anaconda3/lib/python3.6/site-packages/sklearn/neural_network/multilayer_perceptron.py:564: ConvergenceWarning: Stoch
35 astic Optimizer: Maximum iterations (200) reached and the optimization hasn't converged yet.
36 # self.max_iter, ConvergenceWarning)
37 /anaconda3/lib/python3.6/site-packages/sklearn/neural_network/multilayer_perceptron.py:564: ConvergenceWarning: Stoch
38 astic Optimizer: Maximum iterations (200) reached and the optimization hasn't converged yet.
39 # self.max_iter, ConvergenceWarning)
40 /anaconda3/lib/python3.6/site-packages/sklearn/neural_network/multilayer_perceptron.py:564: ConvergenceWarning: Stoch
41 astic Optimizer: Maximum iterations (200) reached and the optimization hasn't converged yet.
42 # self.max_iter, ConvergenceWarning)
43
44 MLPClassifier(activation='relu', alpha=0.0001, batch_size='auto', beta_1=0.9,
45              beta_2=0.999, early_stopping=False, epsilon=1e-08,
46              hidden_layer_sizes=(50, 50, 50), learning_rate='adaptive',
47              learning_rate_init=0.001, max_iter=200, momentum=0.9,
48              nesterovs_momentum=True, power_t=0.5, random_state=None,
49              shuffle=True, solver='adam', tol=0.0001, validation_fraction=0.1,
50              verbose=False, warm_start=False)

```

Figure 3.13: Example of how the GridSearchCV function can be used to identify the most optimal parameter values.

In Figure 3.13 , an example is given using the Exhaustive Grid Search function from Scikit Learn on a Multilayer Perceptron (MLP) machine learning model. The user imports the GridSearchCV function from the Sklearn library and selects which

inputs (could be for instance integer ranges, estimator objects, cross-validation splitting category and so on) to test for five different input parameters to the MLP model in lines 4-10. Since the MLP model has a long training time, please note that by selecting $n_{\text{jobs}} = 4$ as an argument in line 20, we can make use of Python's multiprocessing module to test different parameters by building several models in parallel. This reduces the computational time significantly when evaluating a large number of model input parameter combinations or working with large datasets. As is illustrated in the bottom of the figure, the best parameters for an estimator (built using the most optimal parameters) for the specific training dataset is printed out.

Evaluating the performance of the model

Once the user has selected the most appropriate model, tuned the model parameters and trained a model, it is time to print out a preliminary report that provides feedback on the correlation between the model's predictions and the actual class label that the observation(s) belong to. Continuing to use the Scikit Learn software, the classification report function can be used to print out the precision, recall, $f1$ -score and support for each of the classes and the average score for each metric :

```

1 from sklearn.metrics import classification_report, confusion_matrix
2 print(confusion_matrix(y_test,y_pred))
3 print(classification_report(y_test,y_pred))

[[28647   96    29]
 [ 152 16080  293]
 [    0    59 11832]]
      precision    recall  f1-score   support

     3.0    0.99    1.00    1.00    28772
     4.0    0.99    0.97    0.98    16525
     5.0    0.97    1.00    0.98    11891

 avg / total    0.99    0.99    0.99    57188

```

Figure 3.14: *Classification report evaluates the precision, recall and $f1$ score by comparing the model output (predictions) with the class-label, for instance rock formation label.*

The precision value denotes the ratio between the positive correlation between a model prediction and the class label the observation belongs to (*the truth*), and the total number of predictions [37]:

$$Precision = \frac{truepositivecorrelation}{truepositivecorrelation + falsepositivecorrelation} \quad (3.18)$$

The recall value denotes all positive correlations by considering the number of

true positive and false negative correlations [37]:

$$Recall = \frac{truepositivecorrelation}{truepositivecorrelation + falsenegativecorrelation} \quad (3.19)$$

To explain the concept of true positive, false positive, true negative and false negative correlation, an example can be made for an algorithm that is developed to predict when a kick is about to occur; i.e. when an underbalance in the well pressure allows formation fluids to enter the wellbore.

- True Positive: the algorithm detects a kick, when a kick is indeed occurring,
- False Positive: the algorithm detects a kick, even though a kick actually does not occur,
- False Negative: the algorithm does not detect a kick, when a kick is indeed occurring,
- True Negative: the algorithm detects that a kick is not occurring, when there is no kick either.

The $f1$ -score can be considered a weighted mean value between the precision and recall scores, and support denotes how many times each class or label occurs in the y_{test} . If a satisfactory result is achieved, the user can move on to cross-validate the model on a dataset and later deploy the algorithm. If the performance is insufficient, the user should evaluate the adequacy of the features that have been used, and the estimator parameters.

3.3.2 Multilayer Feed Neural Network and Back-propagation

Multilayer Feed Neural Network

Multilayer feed-forward neural networks, also known as artificial neural networks (ANN) is a network of neurons, or processing elements, that each receives an input signal, processes it and feeds it forward to the next neuron. The neurons are organized in an input layer, either one or several hidden layers, and an output layer. The feed-forward term implies that dataflow is unidirectional from a layer to the consecutive layer. Each neuron is connected with at least one other neuron, and the importance of each connection between the two is represented by a weight coefficient.

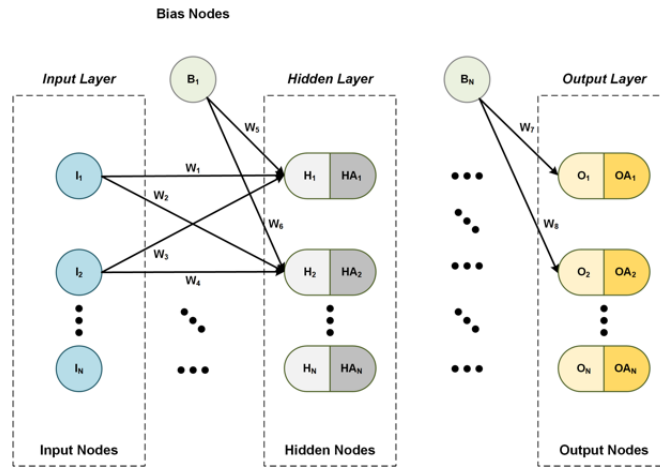


Figure 3.15: *Illustration of a feedforward neural network consisting of an input layer, a hidden layer and an output layer, with weights and biases describing the importance of the neuron-connection in the network [38].*

As is illustrated in Figure 3.15, each neuron in a layer is connected to all neurons in subsequent layer, illustrating how even only a few layers can become a complex network of connections. The number of hidden layers is arbitrary and can be selected depending on the accuracy and complexity required versus the computational power available.

If for each neuron we use a mapping function Γ that assigns a subset $\Gamma(i) \subseteq V$ for all ancestors for neuron i . The subset $\Gamma^{-1}(i) \subseteq V$ consists of all predecessors for neuron i . The weight coefficient ω_{ij} and the threshold coefficient ϑ_i (referred to as bias) describes the connection between neuron i and the next neuron j in the feedforward network, and the processed output value χ from the i neuron is given by [39]:

$$\chi_i = f(\xi_i) \quad (3.20)$$

$$\xi_i = \vartheta_i + \sum_{j \in \Gamma_i^{-1}} \omega_{ij} \chi_j \quad (3.21)$$

ξ_i denotes the i th neuron potential and the function $f(\xi_i)$ is the transfer function

that can be shown as [39]:

$$f(\xi) = \frac{1}{1 + \exp(-\xi)} \quad (3.22)$$

For the case of supervised neural networks, the neural network knows the correct output, and can use this information to correct the weight coefficients until the error between the predicted output and the known output becomes as small as defined. This is achieved by varying the weighting coefficients ω_{ij} and biases ϑ_i to minimize the sum of the squared error between the predicted output and the known output. Minimization of the objective function E is shown below, where x_o and \hat{x}_o are vectors constituting the error. The summation is applied for all output neurons o [39].

$$E = \sum_o \frac{1}{2} (x_o - \hat{x}_o)^2 \quad (3.23)$$

Back-propagation algorithm

A back-propagation algorithm is run iteratively, where for each iteration, the weight coefficients and biases get varied (steepest descent minimization) [39]:

$$\omega_{ij}^{(k+1)} = \omega_{ij}^{(k)} - \lambda \left(\frac{\partial E}{\partial \omega_{ij}} \right)^{(k)} \quad (3.24)$$

$$\vartheta_i^{(k+1)} = \vartheta_i^{(k)} - \lambda \left(\frac{\partial E}{\partial \vartheta_i} \right)^{(k)} \quad (3.25)$$

The term λ denotes the learning rate and is always a positive number. In order to derive the two derivative expressions in equations 3.24 and 3.25, see p. 45 in *Introduction to multi-layer feed-forward neural networks* [39].

To build the feed-forward neural network model, a sequence of training values, or training tuples, are sent to the input layer of the ANN. Here, the data is passed forward into the hidden layers where arbitrary numbers are applied in terms of weight and bias for the connections. Once the training values have passed through the entire network of neurons and all connections have been established, the back propagation algorithm is used to reduce the error. The iterations, where one

complete iteration over the training set is referred to as an epoch, continue until the weights and biases converge below a tolerance that gets specified by the user.

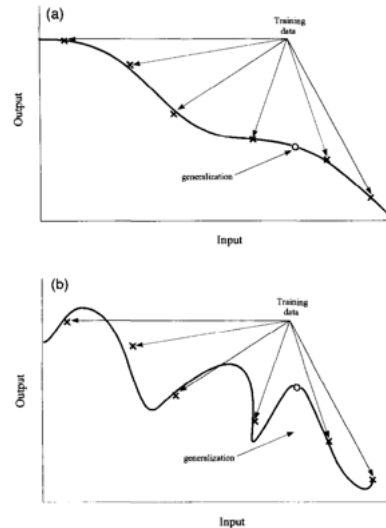


Figure 3.16: *Figure a) illustrates a model built on properly fitted data, where as seen in figure b), the data is overfitted thus less likely to deliver accurate outputs for new input data (Svozil et al., 1997) [39].*

If the number of epochs, or iterations, is set too high, as illustrated in chapter 7, there is a risk of the network memorizing the training data, leading to so-called overfitting. If the model is overfitted, and thus less generalized, there is a high chance that the model will not be capable of delivering accurate outputs for new inputs that the model has not been trained on. This is illustrated in Figure 3.16 [39]:

3.3.3 Support Vector Machine

Support vector machine (SVM) was developed by Vapnik and others and is a machine learning method for training linear learning machines in the kernel-induced feature spaces. These systems offer very efficient algorithms due to the Karush-Kuhn-Tucker conditions. These conditions have a crucial part in the implementation and analysis of these machines. What separates the support vector machine from for example pattern recognition algorithms such as neural networks is that they are convex and have no local minima. Furthermore it has a reduced number of non-zero parameters (Christianini and Shawe-Taylor, 2000) [40].

There is one particular reason for why the support vector machines have become

such powerful classifiers; the kernel trick (illustrated by the polynomial kernel in Figure 3.17). The kernel trick is a mathematical transformation that remaps the data points, making them easier to separate (Sejnowski, 2018) [41].

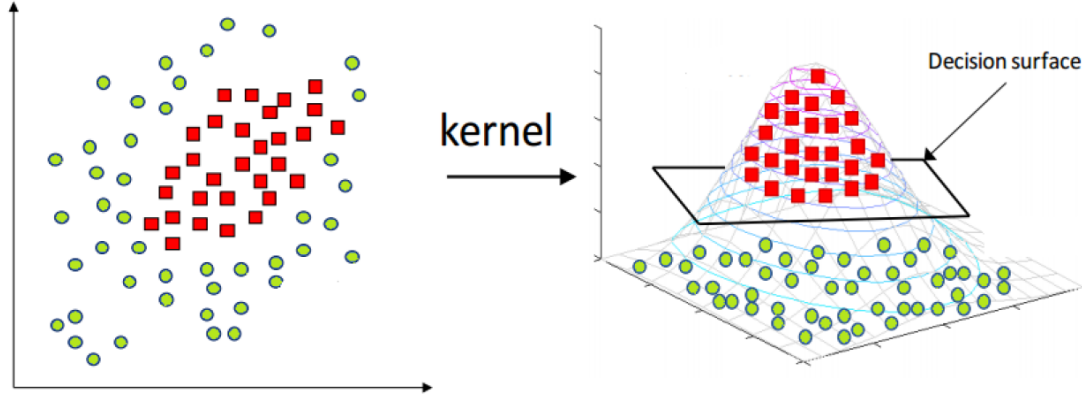


Figure 3.17: *Illustration of the polynomial kernel [6].*

If we use a function that maps our data into higher dimensional space, the maximization (equation 3.26) and decision rules (equation 3.27) will depend on the dot product of the support vectors (Bhattacharyya, 2018) [42]:

$$L = \sum_i \alpha_i - \frac{1}{2} \sum_i \sum_j \alpha_i \alpha_j y_i y_j \phi(\vec{x}_{SV_i}) \cdot \phi(\vec{x}_{SV_j}) \quad (3.26)$$

Where L refers to the Langrange equation for maximization where the dot products of support vectors is calculated. α refers to a positivity constraint. The decision rule, where the dot product of a support vector and a new observation gets calculated is given by:

$$\sum_i \alpha_i y_i \phi(\vec{x}_{SV_i}) \cdot \phi(\vec{u}) + b \geq 0 \quad (3.27)$$

The decision rule gets used to calculate the kernel function K . If K is known, we do not need to know the mapping function. If K is unknown, we need to use the equation for the kernel function, as shown below (Bhattacharyya, 2018) [42]:

$$K(x_i, x_j) = \phi(x_i) \cdot \phi(x_j) \quad (3.28)$$

Several kernel functions exist, and they define the inner product of the trans-

formed space. The most commonly used kernel functions are the polynomial kernel (equation 3.29), the Gaussian kernel (equation 3.30), the Radial Basis Function kernel (RBF) (equation 3.31) and the Sigmoid kernel (Equation 3.32) (Bhattacharyya, 2018) [42].

$$K(x_i, x_j) = (x_i \cdot x_j + 1)^p \quad (3.29)$$

$$K(x_i, x_j) = e^{-\frac{1}{2\sigma^2}(x_i - x_j)^2} \quad (3.30)$$

$$K(x_i, x_j) = e^{-\gamma(x_i - x_j)^2} \quad (3.31)$$

$$K(x_i, x_j) = \tanh(\eta x_i \cdot x_j + v) \quad (3.32)$$

For more details, see (Hastie et al., 2016) [24].

For a function to be defined as a kernel, it must fulfill several requirements. These are defined in Mercer's theorem where the first requirement states that the kernel function needs to be symmetric. In addition, the function needs to be semi-definite. As was mentioned earlier, the kernel function is a matrix element, and as the matrix merges all of the information, the data points and mapping function get fused into the dot product.

3.3.4 Decision Tree

In decision trees (DT), one starts with a root node that consists of all instances given in a dataset. From the first instance, the attribute node splits the instances into several subsets. An attribute can appear several times in a decision tree, but not in the same path. A path will at some point end at a leaf node. Worth noting with regards to the decision tree is that it can become very complex for large datasets. With an increase in the amount of data there will be generated more subsets causing the variations between each of the outcomes to become smaller. Overly complex decision trees can also result in overfitting, which as mentioned

earlier is a common result of the algorithm getting trained on too much data (van der Aalst, 2016) [31].

An important part when designing a decision tree is to know how many nodes to include. Methods like Entropy or the Gini index of diversity can be used to measure the diversity of a leaf node and to select attributes. This is of particular importance, seeing as the tree would become too complex if each attribute had every possible child node attached to it (van der Aalst, 2016) [31].

When an entropy greater than zero gets used (entropy controls the information gain), one can easily observe that when the tree is split into subsets, the resulting outcome of each decision will decrease in diversity as more options get added. If a low entropy value is calculated, there are only small variations between each of the elements in the tree. If the entropy on the other hand is high, then the choices (variations) in the tree are more unique. To calculate the entropy, the following formula gets used (van der Aalst, 2016) [31].

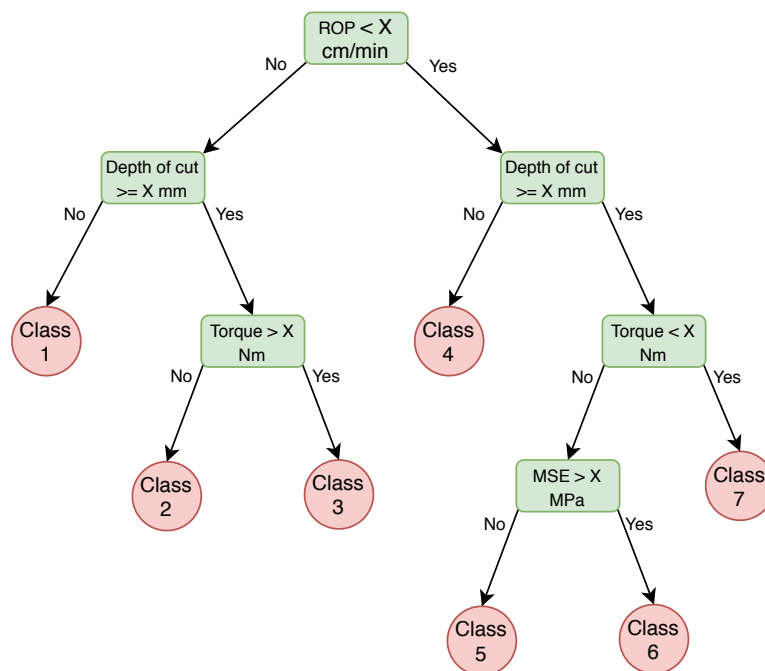


Figure 3.18: *Illustration of how a decision tree could evaluate the features to determine which class an observation or a set of observations belong to.*

$$E = - \sum_{i=1}^k p_i \log_2 p_i \quad (3.33)$$

Where p can be thought of as the proportion or probability and k refers to the number of different elements in X , so that if $k = 1$, all elements are identical, and if $k = n$, all elements in X are different.

When the entropy is zero, the tree is pretty much useless since there no longer is any variation between the elements. It's important to realize when to stop adding options. Another approach to ensure that that an optimal tree model has been developed is to select the attributes that result in the highest G -score from the Gini index of diversity. The Gini index is a similar approach to the concept of evaluating the uniqueness of elements by calculating the entropy, and using the equation below (equation 3.34) one can determine the so-called "impurity" in the decision tree:

$$G = 1 - \sum_{i=1}^k (p_i)^2 \quad (3.34)$$

When $G = 0$, this means that all the classifications of the algorithm are the same. If $G = 1$, then there is great diversity in the classification (van der Aalst, 2016) [31].

3.3.5 Gradient Boosting

In boosting trees, the values of all common predictor variables are partitioned into disjoint regions $R_{j,j} = 1, 2, \dots, J$. These variables can be represented visually as nodes in a tree structure as seen above in Figure 3.18. For each tree structure region, a constant γ_j is given. For this constant, the predictive rule is given by (Hastie et al., 2016) [24]:

$$x \in R_j \Rightarrow f(x) = \gamma_j, \quad (3.35)$$

The tree can then be expressed as (Hastie et al., 2016) [24]:

$$T(x; \Theta) = \sum_{j=1}^J \gamma_j I(x \in R_j), \quad (3.36)$$

with the parameters $\Theta = \{R_j, \gamma_j\}_1^J$ where J is a meta-parameter. Through minimization of the empirical risk, as shown in 3.37, the parameters get found (Hastie et al., 2016) [24]:

$$\hat{\Theta} = \arg \min_{\Theta} \sum_{j=1}^J \gamma_j \sum_{x_i \in R_j} L(y_i, \gamma_j). \quad (3.37)$$

In the equations above, L refers to the loss function $L(y_i, \gamma_j)$ and I is an indicator variable that takes the value of 0 if x is outside of region R_j , and 1 if x belongs to R_j .

This is an optimization problem that usually is set for approximate subplot solutions. In order to solve such optimization problem, it is beneficial to divide the problem into two parts (Hastie et al., 2016) [24]:

1. Finding γ_j given R_j
 - When R_j is given, finding γ_j usually is a trivial task. Often $\hat{\gamma}_j = \bar{y}_j$, where \bar{y}_j is the mean value. \bar{y}_j often falls within the region R_j .
 - If there is a misclassification loss, $\hat{\gamma}_j$ is the modal class that falls within the region of R_j .
2. Finding R_j
 - One needs to find approximate solutions in order to determine R_j . To determine R_j , one also needs to estimate $\hat{\gamma}_j$. To solve this particular problem, one needs to use a top-down recursive partitioning algorithm.

$$\hat{\mathbf{f}} = \arg \min_f L(\mathbf{f}) \quad (3.38)$$

where $\mathbf{f} \in \mathbf{R}^N$ are the approximating function values $f(x_i)$ for each of the N observations x_i (Hastie et al., 2016) [24]:

$$\mathbf{f} = \{f(x_1), f(x_2), \dots, f(x_N)\}. \quad (3.39)$$

The procedures for numerical optimization solve equation 3.38 as a sum of component vectors (Hastie et al., 2016) [24].

$$\mathbf{f}_M = \sum_{m=0}^M \mathbf{h}_m, \quad \mathbf{h}_m \in \mathbf{R}^N, \quad (3.40)$$

where $\mathbf{f}_0 = \mathbf{h}_0$ is the initial guess. Each \mathbf{f}_m in the sequence can then be found based on the value of \mathbf{f}_{m-1} . \mathbf{f}_{m-1} is the sum of the previously induced updates.

The methods for numerical optimization vary in how the increment vector \mathbf{h}_m gets computed (Hastie et al., 2016) [24].

3.3.6 Random Forest

Random forest (RF) is based on the tree structure, and typically gets used for simpler types of prediction problems. It makes use of the same principle as the decision tree model, by selecting parts of the tree structure that further get used to create more powerful prediction models. RF prediction is similar to Bagging, another common ensemble technique, but is considered an improvement due to small tweaks that decorrelate the trees.

In both RF and Bagging, a number of trees get built and connected to a forest-like structure. For each decision where you get a new split, a random sample of m predictors are chosen for the split taken from a set of p predictors. For each split in the prediction model, a new predictor subset-size m is chosen. The subset size m varies in the dataset. For example, when having a large dataset with highly correlated predictors one will typically choose a small subset size m , but usually the size will be $m \approx \sqrt{p}$. There is a good reason for choosing this approach. In Bagging the strongest predictors are considered early in the tree structure. RF on the other hand forces the predictors to get “mixed” for the whole structure of the tree. The lack of “mixing” in the entire tree structure makes the Bagging models highly correlated (as the structures are too similar), while for RF this

issue gets eliminated because at each split there will be a subset of predictors; resulting in that $(p-m)/p$ of the splits will not only consider the strong predictors. This ensures that also some of the weaker predictors get used to build the model (Hastie et al., 2014) [30].

3.3.7 K-Nearest Neighbor

K nearest neighbor (K-NN) is a supervised learning classifier that must not be confused with K-means which is used for unsupervised learning. The method was first introduced by Cover and Hart in 1967 [43] and is widely used in various machine learning applications. Simply explained, the K-NN algorithm tries to classify the class that a sample, or observation, belongs to based on its K surrounding points, as can be observed in Figure 3.19. The classifier has its highest accuracy when surrounded by samples of the same class. If two or more classes are surrounding the samples, the model give an erroneous prediction (Mucherino et al., 2009) [44] and (Xu et al., 2013) [45].

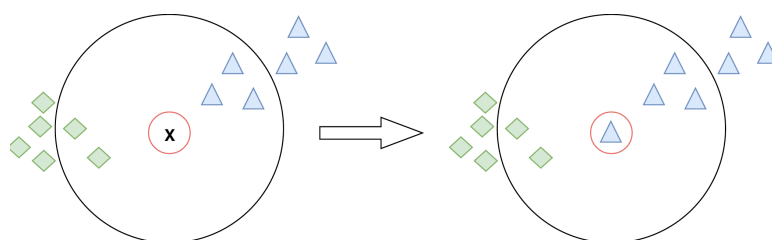


Figure 3.19: *In this example $K=3$, which results in the observation x becoming classified as a blue triangle, since three observations that belong to the blue triangle class are located the closer to the new observation x . than three from the other class (represented as green diamonds).*

For the model to work properly, a correct K -value must be selected. In general, a large K value is preferred, seeing as the model becomes less prone to noisy data. If however K is set too large, a large accumulation of observations that belong to a different class (that outnumbers the correct class in vicinity of the observation point) can cause an erroneous prediction. If on the other hand K is set too small, the decision boundary is too flexible which can lead to patterns being identified that does not exist. Furthermore, a too low K value can take away the advantage of having a large dataset in the first place (Mucherino et al., 2009) [44].

For a more mathematical approach to the problem one can look at the observation point x_0 and the K neighboring points. Using the K-NN classifier, the K neighboring points to x_0 then can be represented as N_0 . Furthermore, using

equation 3.41 the classifier will make calculations, estimating the probability for the class j as a fraction of the identified points in N_0 whose response values are equal to j (Hastie et al., 2014) [30].

$$Pr(Y = j|X = x_0) = \frac{1}{K} \sum_{i \in N_0} I(y_i = j) \quad (3.41)$$

After identifying the neighbor with the highest probability using Bayes rule, x_0 is then given its new class (Hastie et al., 2014) [30].

3.3.8 Bayesian Classification

Bayesian classification method is based on Thomas Bayes theorem. It is a statistical classifier that can predict if a given tuple (finite sequence of elements) belongs to a specific class. Central to the method is that it assumes that a given feature of the dataset is independent of the next. In this classifier, two types of probabilities are used; Posterior probability, $P(c|E)$ and Prior probability, $P(c)$. E is the given tuple while c is the hypothesis.

Knowing this, we can determine whether example E belongs to class c (Zhang, 2004) [46]:

$$P(c|E) = \frac{P(E|c)P(c)}{P(E)}. \quad (3.42)$$

E is classified as the class $C = +$ if the Bayesian Classifier (denoted by f_b) satisfies the condition (Zhang, 2004) [46]:

$$f_b = \frac{p(C = +|E)}{p(C = -|E)} \geq 1, \quad (3.43)$$

To use the naive Bayesian classifier, all attributes need to be independent:

$$p(E|c) = p(x_1, x_2, \dots, x_n|c) = \prod_{i=1}^n p(x_i|c) \quad (3.44)$$

The Bayesian classifier, $f_{nb}(E)$, can then be written as (Zhang, 2004) [46]:

$$f_{nb}(E) = \frac{p(C = +)}{p(C = -)} \prod_{i=1}^n \frac{p(x_i|C = +)}{p(x_i|C = -)} \quad (3.45)$$

3.3.9 TPOT Algorithm

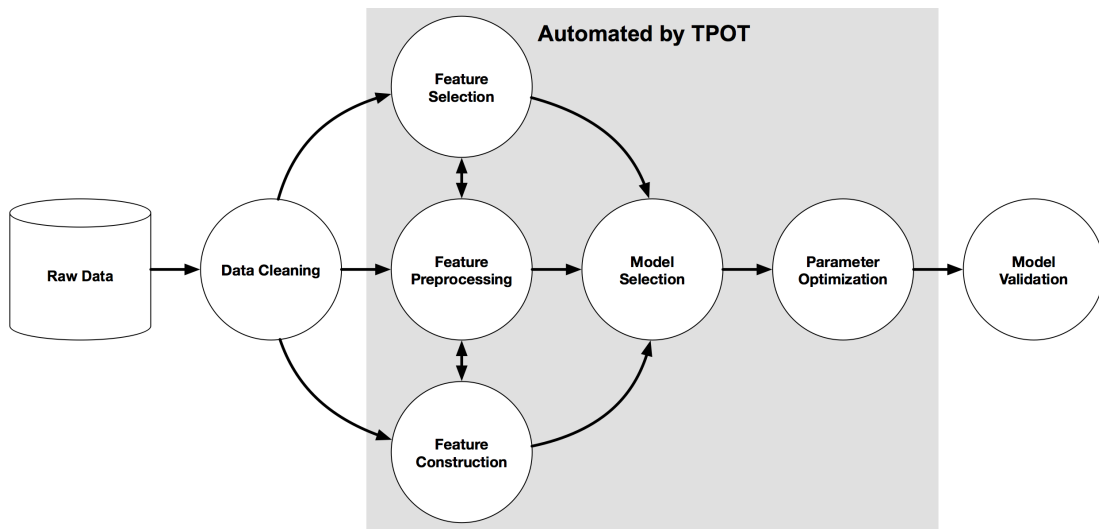


Figure 3.20: *Example of the TPOT algorithm steps that fully automate the process of processing, developing and validating a model [47]*

TPOT is an algorithm that can automate all steps in the methodology to develop and validate a machine learning model, such as data cleaning, feature preprocessing/creation/selection, model selection, parameter tuning and model validation [47]. As is shown in Figure 3.21, the algorithm is based on genetic programming, which is a heuristic search/optimization technique based on the natural selection and genetics principles. The principle of natural selection suggests that a population can only flourish if the genetics constantly improve.

```

from tpot import TPOTClassifier
from sklearn.datasets import load_digits
from sklearn.model_selection import train_test_split

digits = load_digits()
X_train, X_test, y_train, y_test = train_test_split(digits.data, digits.target,
                                                    train_size=0.75, test_size=0.25)

tpot = TPOTClassifier(generations=5, population_size=50, verbosity=2, n_jobs=-1)
tpot.fit(X_train, y_train)

Optimization Progress: 33% | 100/300 [01:02<09:07, 2.74s/pipeline]
Generation 1 - Current best internal CV score: 0.9644750872792087
Optimization Progress: 50% | 150/300 [01:35<05:41, 2.27s/pipeline]
Generation 2 - Current best internal CV score: 0.9681323584103183
Optimization Progress: 67% | 200/300 [01:59<01:59, 1.19s/pipeline]
Generation 3 - Current best internal CV score: 0.9718282518620386
Optimization Progress: 83% | 250/300 [02:23<00:41, 1.21ps]
Generation 4 - Current best internal CV score: 0.9756538552070356

Generation 5 - Current best internal CV score: 0.9756538552070356
Best pipeline: KNeighborsClassifier(input_matrix, KNeighborsClassifier__n_neighbors=10, KNeighborsClassifier__p=DEFAULT, KNeighborsClassifier__weights=distance)

print(tpot.score(X_test, y_test))
0.995555555556

```

Figure 3.21: *The TPOT optimization process is iteratively run, depending on how many generations the algorithm should produce to select the most optimal model to build [47].*

A genetic algorithm works by generating an initial population using randomly generated rules from the data. This initial population is tested against an objective function, and for each iteration a full population of new points is generated. According to Kamruzzaman et al. "The process of generating new populations based on prior populations of rules continues until a population P "evolves" where each rule in P satisfies a pre-specified fitness threshold." [48].

Three main rules apply to the process of producing a new population. These are selection rules, crossover rules and mutation rules. Selection rules can be thought of as selecting the parent data that perform best against the objective function. Crossover rules is the process of selecting parents that have similarities. Finally, the mutation rules are used to apply random genes to the parent data, effectively preventing a premature convergence; the algorithm falling into a local minima prematurely. In short, the genetic algorithms are considered highly applicable to solve optimization problems with multiple possible solutions, illustrating how the TPOT algorithm is capable of selecting the best model, features and tune the input parameters. TPOT is built on the Scikit learn library where all models used in this research are taken from [?].

Since the TPOT algorithm allegedly should be capable of identifying the most optimal pre-processing techniques, features and model pipeline, the algorithm has been used in chapter 7 and chapter 8 for Cases 1-10 in addition to the other models. The TPOT score is then used to evaluate whether our approach to pre-process the raw data, engineer and select our own features and identify the optimal model

parameters, is optimal or not; for instance if one of the other models only achieve an accuracy of 50 %, but the TPOT algorithm scores significantly better.

While the TPOT algorithm can be helpful to get started using machine learning, it is still our recommendation to carry out the steps (as illustrated in Figure 6.25) manually when working with drilling data; particularly with regards to understanding which pre-processing techniques and features that have been selected.

3.4 Unsupervised Machine Learning Models

3.4.1 K-Means Clustering

Clustering, as mentioned earlier, is used to group unlabeled data into so-called clusters, in which a cluster can represent for instance a class, a trend and so on. Since the data is unlabeled, an unsupervised method like K-Means Clustering can be used. This is a distance-based method, and each cluster is marked with a centroid that denotes the center of the cluster. An important factor when using this approach is to choose the correct number of clusters. A simple approach to get started with K means clustering is to start with a small number k , referring to the number of clusters that are to get identified, and then gradually increase the amount if improvements are still observed. If k becomes too large, one might worst case end up with a cluster for each point and a worthless model (van der Aalst, 2016) [31].

Clustering can range from simple diagrams, that are easily interpretable for the human eye, to more complex models that are nearly impossible to interpret for a human. An example of a clustering task that can be difficult for the human mind to interpret is shown below in Figure 3.22. In the example, observations from two variables measured during drilling of three different rock samples are plotted.

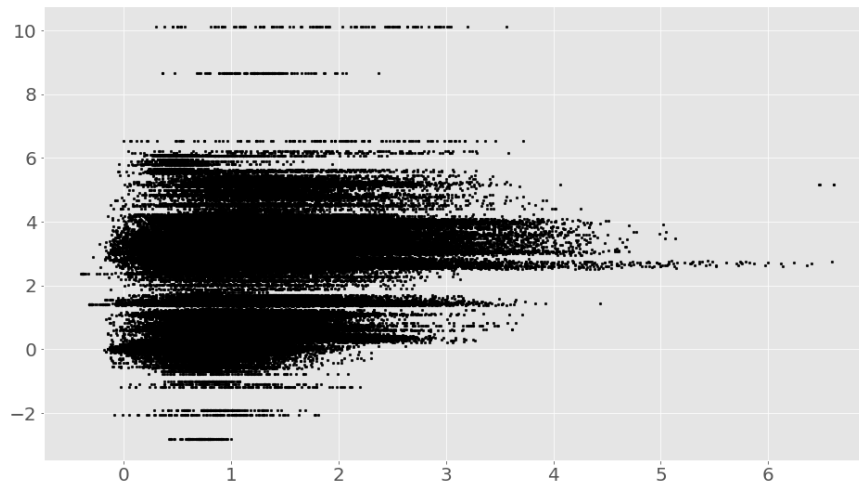


Figure 3.22: *K-Means clustering plot - before clustering.*

From running the K-means clustering algorithm, the user first selects the number of centroids for the machine to randomly generate. The machine then positions the centroids at the center of the three most densely populated areas. Once the centroids have been determined, each observation (point) can be assigned to a cluster, depending on its closeness to the centroids. This can be observed in Figure 3.23.

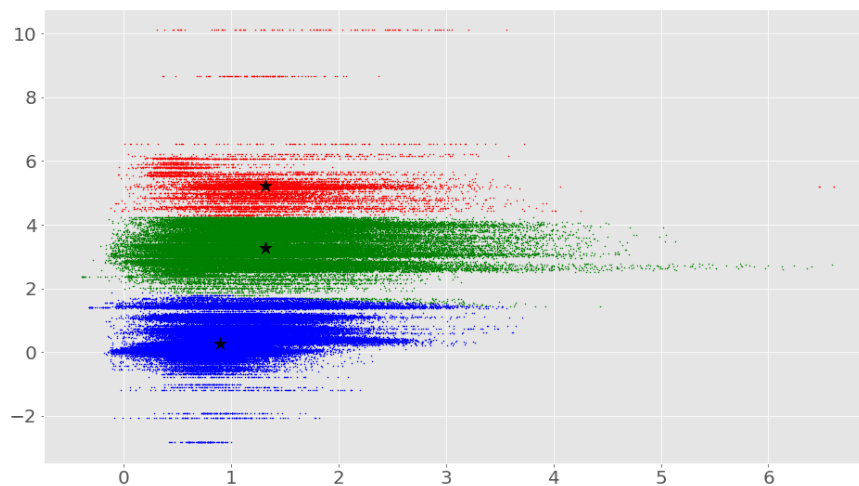


Figure 3.23: *K-Means clustering plot - after clustering.*

There is however no guarantee that the identified clusters are indeed representative for the three different rock formations that the data originate from. One can therefore only use the K means model to organize the data, but not to perform classification and prediction operations.

3.4.2 Density Based Spatial Clustering of Applications with Noise

Continuing on the topic of clustering, Density Based Spatial Clustering of Applications with Noise (DBSCAN) is considered one of the most effective density-based techniques, for reasons such as (He et al., 2011) [49]:

- the technique can identify clusters that are surrounded by other clusters,
- observations can be grouped in clusters without predetermining the number of clusters that should be generated,
- no order is required for the given data points.

There are however some challenges using DBSCAN. For one, datasets are constantly growing in size, which can result in storage challenges. Another aspect to consider is that DBSCAN requires much higher computational capacity, compared to techniques such as K-means clustering (He et al., 2011) [49].

In order to use DBSCAN, one first needs to define when a point becomes a cluster core point. To determine this, we need to define a ε -neighborhood. Then, the number of core points that belong to the given ε -neighborhood can be identified, and when all the core points have been defined, the points that belong to the initial core points can be found. These are the ones with at least a minimum number of samples (defined by the user) (Celebi, 2015) [50].

It is important to understand that a point might not be considered a core point, even though it lies within a ε -neighborhood. It might lie on the edge of a cluster which means that it does not fulfill the requirements for it to be regarded as a core point. For a point to be directly density-reachable to x (a given core point), the following condition needs to be satisfied: $p \in N_\varepsilon(x)$. Furthermore, we need to define whether two points are density-connected. Take for instance the points p and q . They are density-connected if they are density reachable from the point o (Celebi, 2015) [50].

With the different densities defined the clusters can now be found using the DBSCAN technique. A set $C \subseteq D$ is a cluster if the points within C are density connected and there are no subsets within C that also forms clusters. Furthermore, if there are points outside the clusters defined by the DBSCAN algorithm that are not density connected to any point, these values are defined as noise.

Figure 3.24 gives an example of the elements defined above (Celebi, 2015) [50].

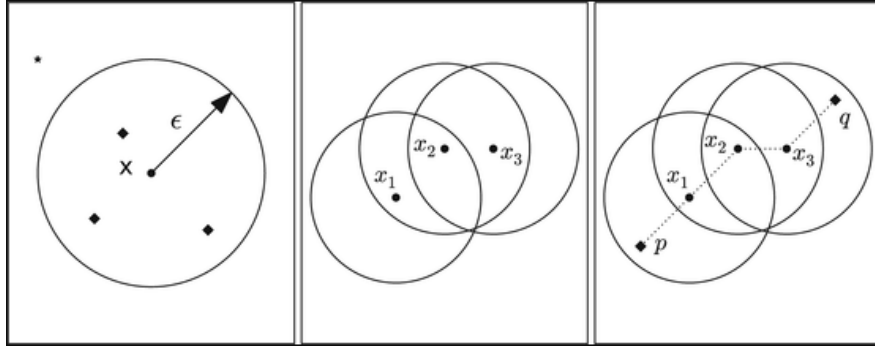


Figure 3.24: *Left figure:* Shows the neighborhood with the core points, labeled by diamonds. The star point (upper left) represents noise.
Middle figure: Shows how x_1 and x_3 are density reachable from x_2 which in turn makes x_3 density reachable from x_1 .
Right figure: Shows how the additional points, p and q , are density connected to the points x_1 , x_2 and x_3 [50].

3.5 Drilling Theory

3.5.1 Drill String Vibrations

Drill string vibrations can be divided into three main categories; torsional, lateral and axial vibrations. The three vibrations types are visualized in Figure 3.25. In drilling vibrations can not be eliminated, considering bit-rock interaction, but is something that can be controlled in order to improve the drilling performance and to reduce the likelihood of damaging equipment such as the drill bit and BHA or the wellbore. The main source of vibrations is the rotation of the drill string which is induced by the top drive or downhole motor. Other causes of vibrations are factors such as torque on bit, hole size, well geometry and so on. In order to measure vibrations, the best practice is to use downhole sensors, but surface measurements are also useful (Sui, 2019) [21].

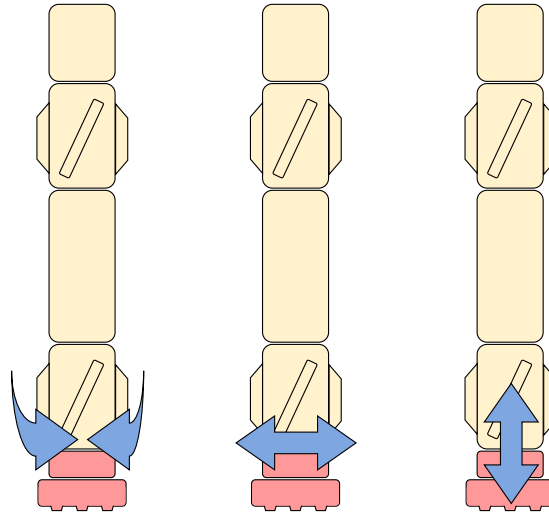


Figure 3.25: *Illustration of torsional-, lateral- and axial vibrations, where the BHA is represented in yellow and the drill bit is represented in red.*

Torsional Vibrations

Considering torsional vibrations, one typically observes oscillations in the drill string without any change in volume. Because of this, the effect of friction from torsional vibrations is often lower than the effect from axial- and lateral vibrations. The torsional oscillations will easily propagate upwards along the drill string until it reaches the top drive where the pipe is fixed. Because the top drive is fixed, it allows the oscillated waves to reflect which in turn will lead to a self-reinforcing transfer of energy. This can be avoided if drilling parameters are changed because this will break the pattern of the oscillations (Sui, 2019) [21].

When talking about torsional vibrations, one typically refers to stick-slip. Stick-slip occurs for two main reasons; the bit digging too aggressively into the formation causing it to get stuck before the built up energy releases the bit or by the interaction between the drill string and the wellbore. The probability of a stick-slip is higher in deviated well sections. This can result in the ROP being decreased by 30-40 % in severe cases, and when the string or bit releases from the stuck situation the RPM downhole can typically become several times the surface RPM. The conditions for a stick-slip to occur will be different depending on the environment downhole, but with increasing WOB, stick-slip can occur more frequently since a higher WOB can increase the friction between the bit and formation. Therefore a strategy to mitigate the probability of stick-slipping occurring is to lower the WOB and rather increase the RPM. This lowers the overall probability of either the bit or string getting temporarily stuck downhole, but can in turn increase the

chance of bit whirl (Sui, 2019) [21] and (Larsen, 2014) [51].

Lateral Vibrations

Lateral vibrations include both bending (transverse shear stress) and whirl. This type of vibration does not travel upwards along the string like axial and torsional vibrations do, making it difficult to detect on surface and to prevent. There are different types of lateral vibrations, but the most complicated is the centrifugal bowing of the BHA between two stabilizers, and in extreme situations, the RPM can coincide with the natural beam-bending mode.

Another type of lateral vibrations that can cause significant damage to both the BHA, bit and wellbore is whirl. Whirl is when the bit starts rotating off-center in an enlarged hole, and the pipe and BHA can get exposed to both forward- and backward whirl, where backward whirl is considered to be the most dangerous as the vibration moves in a counter-clockwise direction opposite to the normal pipe rotation (clockwise). An effect of whirl can be that the BHA starts to climb on the walls wellbore due to friction. When the BHA slips and falls to bottom; acceleration can in extreme cases reach 100 g.

When lateral vibrations and whirl are initiated, they tend to be quite stable which obviously not is desirable. A remedial action to lateral vibrations is to allow the energy to dissipate through stopping the rotation in the top drive and pulling off bottom. Drilling parameters where lateral vibrations occur should then be avoided when drilling continues. In addition to causing high wear to the drill string, BHA and bit, lateral vibrations can also decrease directional control. The hole will also likely become overgauge, enabling even more lateral movement downhole (Sui, 2019) [21] and (Larsen, 2014) [51].

Axial Vibrations

To understand axial vibrations in the drill string, one can think of it as a mass spring system as illustrated in Figure 3.26 (Sui, 2019) [21]:

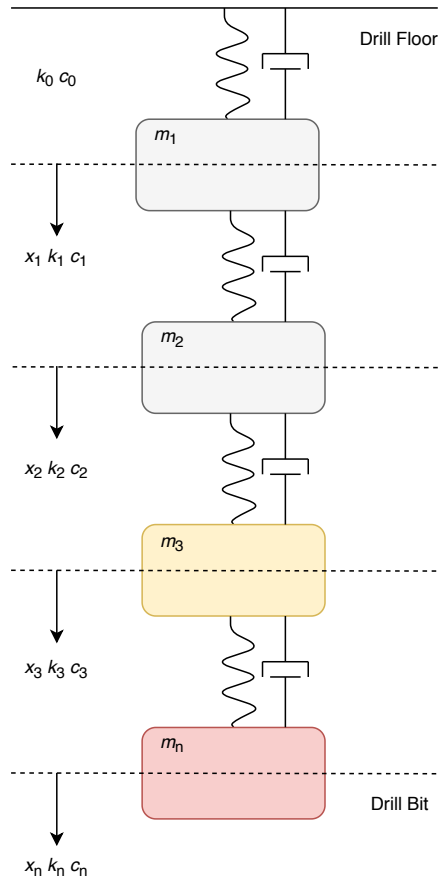


Figure 3.26: *Drill string system - illustrated by a mass spring system. Drill pipe represented as the grey part, BHA as the yellow part and drill bit as the red part.*

To determine the vibrations, one needs to apply Newton's second law to all elements of the system (Sui, 2019) [21].

$$m_1 \ddot{x}_1 + c_0 \dot{x}_1 - c_1 (\dot{x}_2 - \dot{x}_1) - k_1 (x_2 - x_1) + k_0 x_1 = m_1 g \beta, \quad (3.46)$$

$$\begin{aligned} m_i \ddot{x}_i - c_i (\dot{x}_{i+1} - \dot{x}_i) + c_{i-1} (\dot{x}_i - \dot{x}_{i-1}) - k_i (x_{i+1} - x_i) + k_{i-1} (x_i - x_{i-1}) \\ = m_i g \beta, \end{aligned} \quad (3.47)$$

$\forall i = 2, \dots, n-1,$

$$m_n \ddot{x}_n + c_n (\dot{x}_n - \dot{x}_{n-1}) + k_{n-1} (x_n - x_{n-1}) = m_n g \beta - F(t), \quad (3.48)$$

where m_i denotes the mass of the element i and k_i is the spring constant of the

spring connected above the mass i . c denotes the damping coefficient of the spring-mass system. One also needs to account for the buoyancy factor, β , of the fluid that the system is submerged in (for instance the drilling mud). To account for the bit-rock interaction, $F(t)$ is introduced. As can be observed from equation 3.49, $F(t)$ depends on several drilling parameters (Sui, 2019) [21]:

$$F(t) = f(RPM, WOB, ROP, \alpha) \quad (3.49)$$

where RPM is the rotational speed, WOB is the weight on bit, ROP is the rate of penetration and α are other factors that one needs to consider (Sui, 2019) [21].

The system, given by the equations 3.46 to 3.48 is a n -coupled second order ordinary differential equation, meaning that we can write it on matrix form as shown in equation 3.50 (Sui, 2019) [21]:

$$M\ddot{x} + C\dot{x} + Kx = \mathbf{F} \quad (3.50)$$

3.5.2 Directional Drilling

Directional drilling is one of the biggest contributors to the oil and gas industry when it comes to increased oil recovery (IOR). The concept of drilling non-vertical well sections made it possible for operators to not only aim their wells in the exact direction of the reservoir, but also to avoid areas like salt domes and crossing faults in more optimal angles as well as to position the wells at a point in the reservoir where maximum depletion can be achieved. Other advantages with directional drilling include being able to drill several wells from the same platform, the drilling of multilateral wells, performing sidetracking, drilling of relief wells and so on (Fjelde, 2017) [52].

In order to understand the concept of directional drilling one needs to know four concepts (Fjelde, 2017) [52]:

1. True vertical depth (TVD):
 - The vertical distance to the bottom of the well
2. Measured depth (MD):

- The tangential space of the well path
3. Inclination:
- Angle between the tangent to the wellbore and the vertical
4. Azimuth:
- The horizontal plane measured clockwise from north in the range 0-360 degrees

Another important concept, which is the combined change in inclination and azimuth is the dogleg severity (DLS). In the industry it is usually measured in degrees per 30 m, or in other words how many degrees you build per stand of drill pipe. To calculate the DLS one first needs to calculate the dogleg angle (DL), which is the angle between two points along a curve and is given by equation 3.51 below:

$$DL = \cos^{-1}(\cos I_1 \cos I_2 + \sin I_1 \sin I_2 \cos(A_2 - A_1)) \quad (3.51)$$

The variables I_1 , I_2 , A_1 and A_2 refers to the inclination and azimuth at two points (I_1, A_1) and (I_2, A_2) as illustrated in Figure 3.27. The dogleg severity (DLS) can further be found from:

$$DLS = \frac{DL}{L} \quad (3.52)$$

where L is the wellpath or curve connecting the two points marked by red dots in Figure 3.27

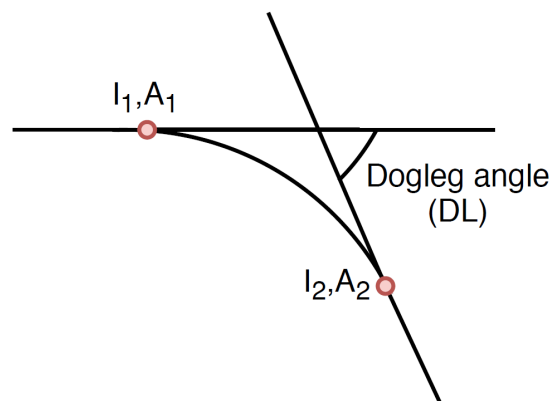


Figure 3.27: Illustration of the dogleg angle between the two points I_1, A_1 and I_2, A_2 in a wellpath.

To calculate the change in azimuth for a new well position, you can use equation 3.53:

$$\Delta A = \tan^{-1}\left(\frac{\tan DL \cdot \sin TF}{\sin I_1 + \tan DL \cdot \cos I_1 \cdot \cos TF}\right) \quad (3.53)$$

To find the new inclination, I_2 , one can use equation 3.54:

$$I_2 = \cos^{-1}(\cos I_1 \cdot \cos DL - \sin I_1 \cdot \sin DL \cdot \cos TF) \quad (3.54)$$

The toolface, TF , is the point set 90 degrees either to the left or right of the borehole. It can be calculated using equation 3.55 given that the dogleg and expected change in inclination is known.

$$TF = \cos^{-1}\left(\frac{\cos I_1 \cdot \cos DL - \cos I_2}{\sin I_1 \cdot \sin DL}\right). \quad (3.55)$$

3.5.3 Buckling Models

In oil wells, the tubing will be constrained by the circular path made up by the well itself. The well typically contains four different sections: vertical, curved, inclined and horizontal. For the different sections we have both sinusoidal and helical buckling (see Figure 3.28 for illustrations) (Belayneh, 2018) [53].

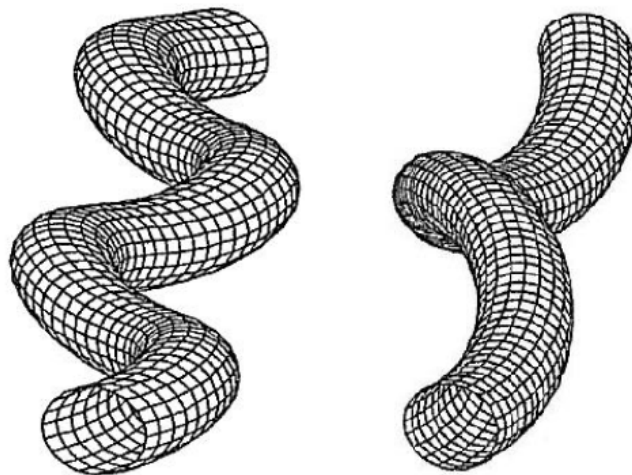


Figure 3.28: *Buckling modes. Left side: Sinusoidal. Right side: Helical [53].*

These are the two modes of buckling that can appear in circularly constrained

wells. **Sinusoidal buckling** is the first phase of buckling. It happens when the pipe encounters lateral constraints along the length of the tube which in turn causes a sinusoidal shape of the pipe, as can be observed in Figure 3.28 above. Sinusoidal buckling is often referred to as the critical buckling load. **Helical buckling** is the second phase of buckling referred to as the critical tube buckling. If one continues to increase the load in the sinusoidal phase of buckling, one will at some point reach the helical buckling load. At this point the tubing will be of helical shape inside the casing. If additional force is applied after sinusoidal and helical buckling modes one could also experience the phenomenon **lock-up**. A lock-up happens because the helix-shaped tubing gets in contact with the casing wall. This results in an outward force that creates friction between the casing and tubing. If the friction becomes large enough it will be impossible to push more tubing into the hole (Belayneh, 2018) [53].

Section	Non-rotating buckling	
	Sinusoidal	Helical
Vertical	Wu et al. (1993): $F_{sin} = 2.55(EIw^2)^{1/3}$ $I = \frac{\pi}{64}(OD^4 - ID^4)$ <ul style="list-style-type: none"> • E = Youngs Modulus 	Wu et al. (1993): $F_{hel} = 5.55(EIw^2)^{1/3}$
Curved	Mitchell (1999): $F_{sin} = \frac{2EI k}{r} \left(1 + \sqrt{1 + \frac{w \sin \alpha r}{EI k^2}} \right)$ <ul style="list-style-type: none"> • $k = \frac{1}{R}$ (build or drop) • $r = \frac{1}{2}(ID_{well/casing} - OD_{tubing})$ (radial clearance) 	Mitchell (1999): $F_{hel} = 2.83F_{sin}$
Inclined	Dawson and Paslay (1984): $F_{sin} = 2 \left(\frac{EIw \sin \alpha}{r} \right)^{0.5}$ <ul style="list-style-type: none"> • r = radial clearance • α = inclination 	Aasen and Aadnøy (2002): $F_{hel} = 3.75(EI)^{0.5}w^{0.5} \left(\frac{\sin \alpha}{r} \right)^{0.5}$ $1.875 \times F_{Dawson Pasay sinusoidal}$

Figure 3.29: A selection of non-rotating buckling models [53].

Chapter 4

Data Preparation

In this chapter, the techniques to collect, concatenate, label and describe each dataset for different classification problems is discussed. Due to its available machine learning libraries, the Python programming language has been used together with the Jupyter Notebook integrated developer environment (IDE) to develop the methodology and results that are discussed in chapter 4 to chapter 8.

4.1 Laboratory data preparation for rock / formation classification

4.1.1 Data source

Six different rocks were drilled at different drilling parameter combinations such as:

- Uniform cement:
 1. 400 RPM with 3 & 6 kg WOB
 2. 600 RPM with 3, 6 & 10 kg WOB
 3. 800 RPM with 10 kg WOB
- Uniform salt:
 1. 400 RPM with 3 & 6 kg WOB
 2. 600 RPM with 3, 6 & 10 kg WOB
 3. 800 RPM with 8 & 10 kg WOB
- Homogeneous sandstone (Drillbotics[®] competition rock 2018):

1. 400 RPM with 3 & 6 kg WOB
 2. 600 RPM with 3, 6 & 10 kg WOB
 3. 800 RPM with 3, 6 & 10 kg WOB
 4. 1000 RPM with 10 kg WOB
- Chalk:
 1. 400 RPM with 5 kg WOB
 2. 600 RPM with 5 kg WOB
 3. 800 RPM with 10 kg WOB
 4. 1000 RPM with 10 kg WOB
 - Shale:
 1. 300 RPM with 3 & 6 kg WOB
 2. 600 RPM with 3, 6 & 10 kg WOB
 3. 800 RPM with 3, 6 & 10 kg WOB
 4. 1000 RPM with 10 kg WOB
 - Granite:
 1. 400 RPM with 3 & 6 kg WOB
 2. 600 RPM with 3, 6, & 10 kg WOB
 3. 800 RPM with 3, 6 & 10 kg WOB
 4. 1000 RPM with 10 kg WOB



Figure 4.1: *Collection of different rock specimen drilled to gather experimental drilling data.*

Before each rock sample was drilled, the same operational procedure was followed including; zeroing all sensors, configuring the sampling rate, and drilling a pilot

hole for each well to prevent that a big portion of the dataset would contain pilot hole drilling (here referred to as the first inch of the well). The sampling frequency was configured to 9600 Hz to gather as much data as possible. The data is then downsampled to approximately 1 % of the original data size; 96 Hz.

4.1.2 Data concatenation

Before data can be concatenated, metadata such as WOB setpoint for the PID controller and RPM setpoint for the top drive was input as separate columns. Drilling features such as instantaneous ROP that need to be calculated before concatenating the data were also calculated and implemented ahead of merging multiple datasets to ensure that transitions between datasets do not get mixed up for these features (see chapter 6). In addition, observations from the computational channels on the high frequency DAQ such as bit torque (calculated) and MSE (calculated) were removed due to uncertainty as to whether these features correctly describe the rock formations and thus were evaluated to be too uncertain to introduce when developing the models. Since there are no perfect drilling operations, data that represent moderate vibrations or slight deviation drilling is intentionally kept in the dataset in an attempt to make the data best represent the actual conditions that occur at high drilling speeds with the laboratory rig. The process of concatenating all experiments and labeling them is repeated for rock formations 1 through 6 so that the pool of data to develop models on consist of the following number of observations for each rock formation specimen:

Rock specimen	Label	Number of samples (rows)
Cement	1	8.263×10^6
Chalk	2	1.864×10^6
Granite	3	1.473×10^7
Sandstone	4	1.020×10^7
Salt	5	7.149×10^6
Shale	6	1.375×10^7

The difference in number of observations per rock specimen is based on the availability of different rock specimen to drill, as well as the drilling speed (a 150 mm thick chalk specimen is drilled in less than a minute for reference, however a well drilled in granite rock would require several hours to drill). At 9600 Hz sampling rate, 10^6 samples represent approximately 104 seconds of drilling for reference.

4.1.3 Data labeling

Once the experimental data from all experiments containing observations from the same rock specimen have been imported, the dataframe is labeled by class-labels 1 to 6 (each rock formation has its own class-label). This ensures that supervised models can be trained (since these require a class-label to train), that the accuracy of supervised models can be evaluated and finally that each rock formation if necessary can get separated by the class-label, for instance if a class is to be dropped from the dataframe.

```

1 combined_df_cement.head()

```

	time	z1	z2	z3	rpm	m_torque	depth	wob	RPM_set	WOB_set	mean_depth	ROP_instantaneous
0	0.000104	-2206.5	-1820.1	-1827.0	268.80	0.52069	1.4512	-5.8536	400	3	1.390149	NaN
1	0.000208	-2323.5	-1769.9	-1931.9	268.41	0.51412	1.4824	-6.0253	400	3	1.390149	NaN
2	0.000313	-2432.1	-1741.4	-1990.2	268.02	0.50641	1.3705	-6.1638	400	3	1.390149	NaN
3	0.000417	-2469.8	-1740.1	-2045.9	268.33	0.47528	1.2064	-6.2558	400	3	1.390149	NaN
4	0.000521	-2458.3	-1742.4	-2081.8	268.61	0.45768	1.3005	-6.2825	400	3	1.390149	NaN

```

1 #NB This is Label 1
2 combined_df_cement['Label'] = '1'
3 combined_df_cement.tail(2)

```

	time	z1	z2	z3	rpm	m_torque	depth	wob	RPM_set	WOB_set	mean_depth	ROP_instantaneous	Label
253559	26.412	-7835.1	-7438.1	-10894.0	758.54	7.7141	-26.167	-0.000012	800	10	-16.031214	-12.644352	1
253560	26.413	-7773.5	-7437.7	-10825.0	758.74	7.7052	-26.036	-0.000012	800	10	-16.031214	-12.644352	1

Figure 4.2: Data is labeled before all dataframes get combined.

4.1.4 Describing the raw data

There are several ways to describe a dataset or dataframe. For rock classification, three initial techniques get used to describe the data that is collected from experiments. These are pandas `describe()` function, matplotlibs plotting function and the `pairplot` function from the Seaborn library.

Describing the raw data

When describing the raw data, a series of statistical properties such as the count, minimum and maximum value, the 25, 50 and 75 percentile, standard deviation and mean value is calculated. By running the functions `Dataframe.describe()` and `Dataframe.shape`, the following table is printed out representing our merged dataset for all rock formations.

	Unnamed: 0	time	z1	z2	z3	rpm	m_torque	depth	wob	RPM_set	WOB_set	mean_depth	ROP_instantaneous	Label
count	5.595299e+07	5.595299e+07	5.595299e+07	5.595299e+07	5.595299e+07	5.595299e+07	5.595299e+07	5.595299e+07	5.595299e+07	5.595299e+07	5.595299e+07	5.595299e+07	5.595299e+07	5.595299e+07
mean	7.781085e+05	8.106470e+01	-2.759743e+03	-2.623857e+03	-2.335816e+03	5.254562e+02	1.148161e+00	2.216989e+00	-7.446882e+00	8.071482e+02	5.720446e+00	2.216986e+00	1.098972e+00	3.846262e+00
std	5.417258e+05	5.641651e+01	2.320807e+03	2.195666e+03	2.161453e+03	1.973906e+02	8.144123e-01	3.354048e+00	5.809061e+00	1.638781e+02	2.690044e+00	3.168358e+00	5.104438e+00	1.677086e+00
min	0.000000e+00	0.000000e+00	-2.186600e+04	-1.719000e+04	-2.044300e+04	-1.085100e+01	-4.186000e-01	-4.661500e+01	-4.139900e+01	4.000000e+02	3.000000e+00	-1.878285e+01	-2.350244e+02	1.000000e+00
25%	3.266710e+05	3.405000e+01	-4.091500e+03	-3.917100e+03	-3.577000e+03	2.936100e+02	7.239900e-01	8.526100e-01	-1.072300e+01	4.000000e+02	3.000000e+00	8.585532e-01	3.545596e-02	3.000000e+00
50%	6.869400e+05	7.156500e+01	-2.530500e+03	-2.368600e+03	-2.149700e+03	5.132900e+02	9.888200e-01	1.536700e+00	-6.722400e+00	8.000000e+02	6.000000e+00	1.512210e+00	2.487134e-01	4.000000e+00
75%	1.175655e+06	1.224600e+02	-1.208700e+03	-1.124800e+03	-8.785100e+02	7.462800e+02	1.329200e+00	3.665000e+00	-3.362200e+00	8.000000e+02	6.000000e+00	3.654129e+00	1.314037e+00	5.000000e+00
max	2.882784e+06	3.002900e+02	1.466300e+04	1.333800e+04	1.015300e+04	1.074900e+03	8.442800e+00	1.858600e+01	2.540300e+01	1.000000e+03	1.000000e+01	1.147545e+01	2.840156e+02	6.000000e+00

Figure 4.3: Description of original data (before pre-processing) and their statistical properties such as mean value, percentiles and maximum value for rock classification.

The number of (rows, columns) is equal to (55952987, 14).

Plotting the raw data

Using the matplotlib plotting function, each natural feature (column) in the dataframe can be plotted against the sample index:

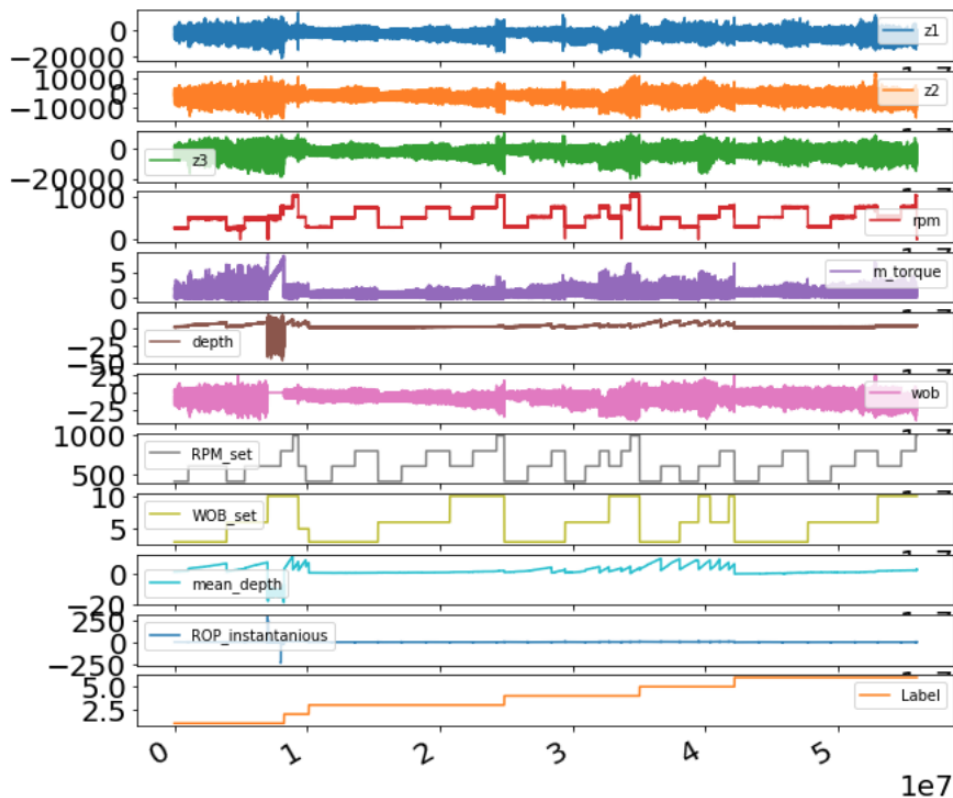


Figure 4.4: Plot representing the dataset containing natural features only (sensor measurements) from experiments where six different rock formations have been drilled.

Generating a raw data pairplot

Using the seaborn pairplot function, a pairplot can be generated to plot every feature (data column) against each other. Note that observations in the data that are referred to as Not a Number (NaN) need to get removed ahead of running this function. Please note that the pairplot below has been generated by randomly downsampling the dataset to 1 % of the original dataset in order to reduce the computational power required to plot the data.

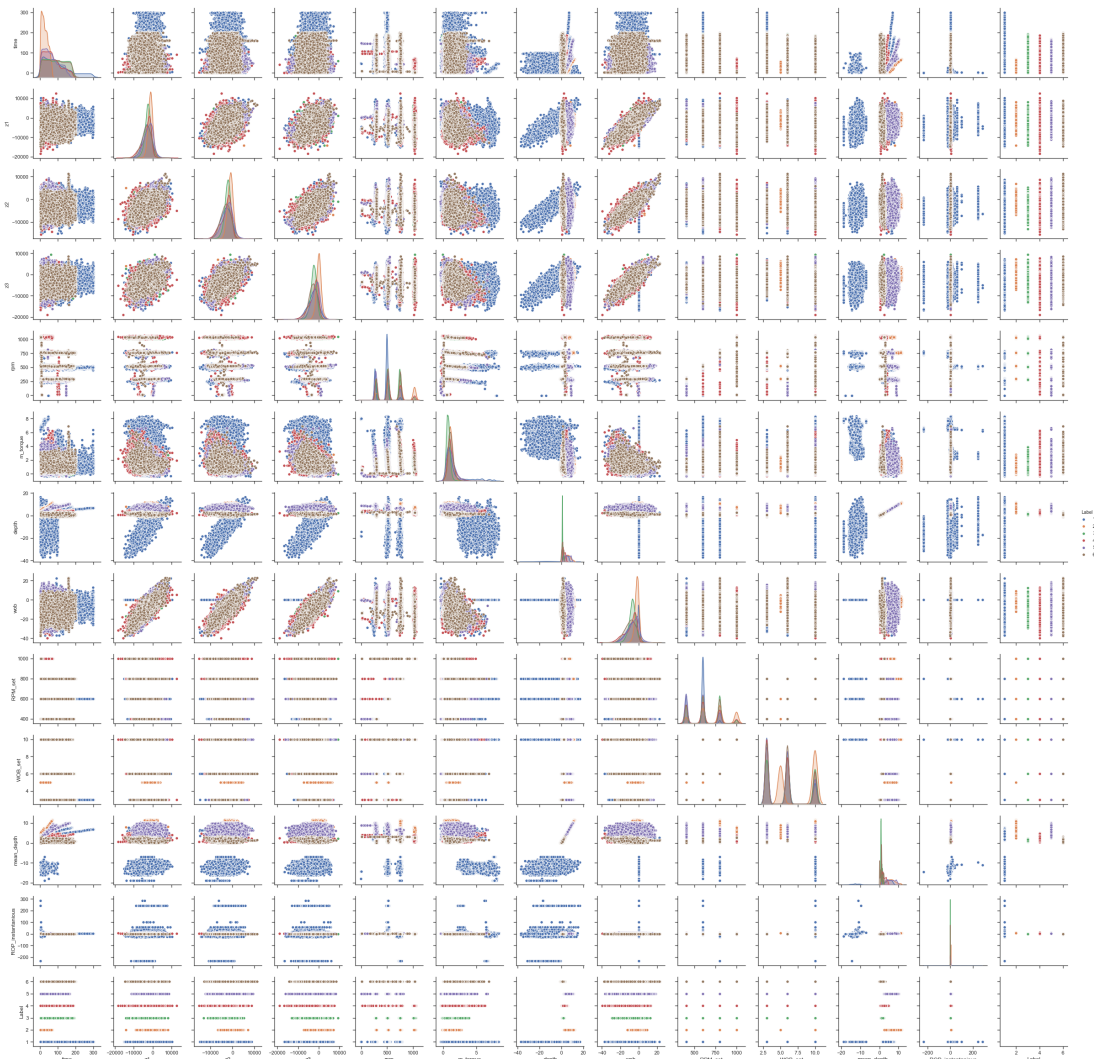


Figure 4.5: *Seaborn pairplot representing raw data from experimental drilling that has been used for rock formation classification.*

4.2 Data collection for classification of laboratory rig operations

4.2.1 Data source

Similar to the approach above, a total of nine experiments were conducted to collect data on three rig operations in an attempt to develop models on the laboratory-scale drilling rig that can distinguish between drilling and NPT activities such as tripping. These three operations are tripping up (POOH), tripping down (RIH), and rotating on bottom (ROnB).

The experiments contain data for each operation, either with or without bit rotation, circulation, or a combination of both.

4.2.2 Data concatenation

The same approach as described in subsection 4.1.2 is used to concatenate the data, after metadata and features such as instantaneous ROP have been calculated and implemented in the dataset.

4.2.3 Data labeling

The data is labeled so that each operation is represented by:

Rig operation	Label	Number of samples (rows)
POOH	1	8.153×10^5
RIH	2	9.633×10^5
ROnB (R)	3	1.77×10^6

4.2.4 Describing the data

Describing the raw data

Figure 4.6 shows the distribution of data and some statistical properties for all three rig operations.

	time	z1	z2	z3	rpm	m_torque	depth	wob	RPM_set	WOB_set	Label	mean_depth	ROP_instantaneous
count	3.548191e+06	3.548191e+06	3.548191e+06	3.548191e+06	3.548191e+06	3.548191e+06	3.548191e+06	3.548191e+06	3548191.0	3.548191e+06	3.548191e+06	3.548191e+06	3.519391e+06
mean	6.934461e+01	-1.187386e+03	-1.433638e+02	-4.972998e+01	3.150741e+02	5.572790e-01	-1.427101e+00	-1.380479e+00	450.0	1.357496e+00	2.268960e+00	-1.427100e+00	-6.772814e-01
std	4.799663e+01	3.585297e+02	3.879194e+02	3.330866e+02	6.895126e+00	5.631268e-01	2.730808e+00	3.085728e-01	0.0	2.223665e+00	8.100379e-01	2.651067e+00	5.780697e+00
min	0.000000e+00	-2.522600e+03	-1.548200e+03	-1.161300e+03	2.943700e+02	-3.903300e-01	-1.182700e+01	-3.186900e+00	450.0	0.000000e+00	1.000000e+00	-8.381292e+00	-1.673011e+01
25%	3.080000e+01	-1.464000e+03	-4.427800e+02	-3.371000e+02	3.095100e+02	-3.123100e-02	-2.349100e+00	-1.484400e+00	450.0	0.000000e+00	2.000000e+00	-2.058214e+00	-1.239529e-01
50%	6.160100e+01	-1.210800e+03	-1.493300e+02	-4.622800e+01	3.156800e+02	6.217100e-01	-8.890900e-01	-1.357600e+00	450.0	0.000000e+00	2.000000e+00	-8.365801e-01	-2.296581e-03
75%	9.613900e+01	-8.792400e+02	1.638800e+02	2.190800e+02	3.206900e+02	9.808300e-01	-1.796900e-01	-1.235000e+00	450.0	5.000000e+00	3.000000e+00	-7.621476e-01	5.679631e+00
max	1.843300e+02	1.462200e+02	1.281700e+03	1.116000e+03	3.491900e+02	2.162800e+00	6.686100e+00	6.936700e-02	450.0	5.000000e+00	3.000000e+00	5.765513e+00	6.463580e+00

Figure 4.6: Description of datasets statistical properties such as mean value, percentiles and maximum value for rig operations.

Plotting the raw data

The plot below has been generated by sorting the dataset by the class-label for each operation:

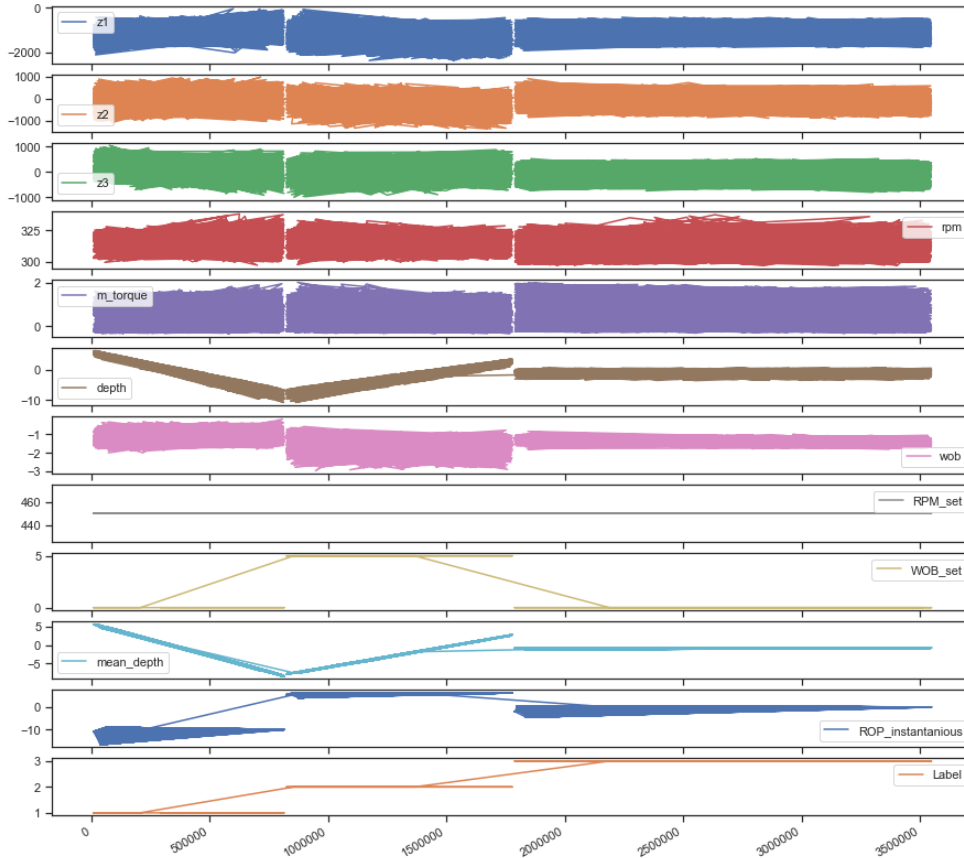


Figure 4.7: Plot representing the raw data for laboratory rig operations.

Pairplot of the raw data

Please note that the pairplot below has been generated by randomly downsampling the dataset to 1 % of the original dataset to reduce to computational power required to plot the data:

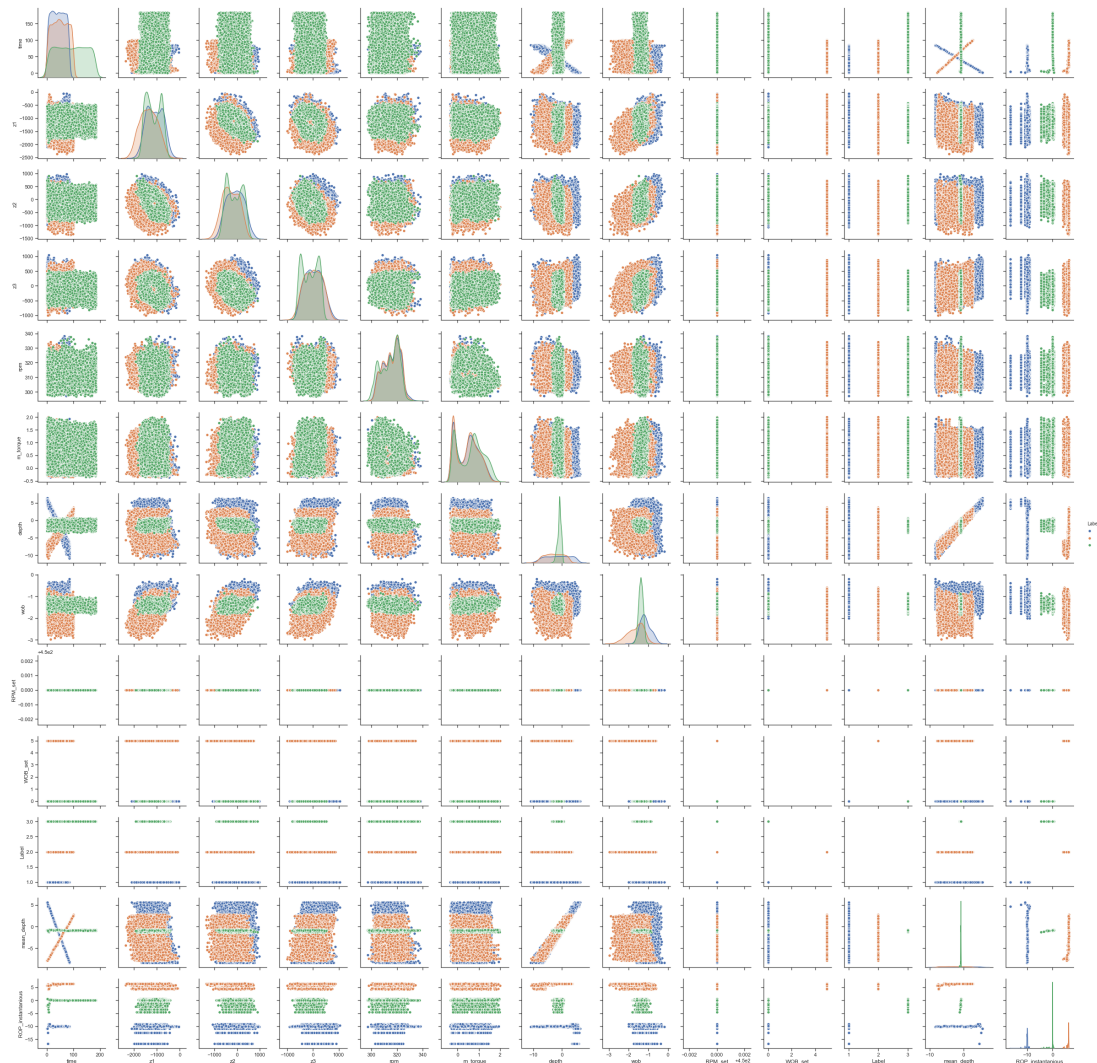


Figure 4.8: Seaborn pairplot representing rig operation raw data.

4.3 Surface Data collection for drilling incident classification

4.3.1 Surface data for vibration classification

Data source

Two experiments were conducted to gather data on low vibrations and moderately high lateral vibrations (whirl-tendency) by drilling at 120 and 510 RPM respectively with the same WOB setpoint. To ensure that vibrations get measured without capturing the impact between BHA stabilizers and the riser, the riser was removed ahead of the operation.

Data labeling

The data is labeled so that normal vibrations (low) is denoted by label 1, and moderately high vibrations is denoted by label 2.

Describing the raw data

The data can be generally described as seen in Figure 4.9 below:

	time	z1	z2	z3	RPM	m_torque	WOB	RPM_set	WOB_set	Label
count	582094.000000	582094.000000	582094.000000	582094.000000	582093.000000	582093.000000	582093.000000	582094.000000	582094.0	582094.000000
mean	15.377994	-2017.591358	-1551.966599	-1031.493717	343.599344	0.704763	-4.601055	338.456521	5.0	1.560145
std	9.121256	2201.105963	2019.195466	2011.685048	186.288488	0.311620	5.450993	193.584235	0.0	0.496370
min	0.000000	-9696.600000	-9008.700000	-8924.600000	120.470000	-0.107210	-21.426000	120.000000	5.0	1.000000
25%	7.579325	-3170.500000	-2522.200000	-2010.100000	133.890000	0.472690	-7.167000	120.000000	5.0	1.000000
50%	15.159000	-2124.500000	-1433.300000	-1144.900000	503.600000	0.643790	-4.679500	510.000000	5.0	2.000000
75%	22.738000	-892.310000	-564.392500	-141.095000	509.300000	0.900440	-2.091100	510.000000	5.0	2.000000
max	33.964000	5873.800000	6099.300000	7322.700000	521.040000	2.101200	13.844000	510.000000	5.0	2.000000

Figure 4.9: Description of the raw data using surface sensors only.

Plotting the raw data

As can be seen from Figure 4.10, the case of classifying vibration levels could rather easily be solved analytically by comparing for example the change in amplitude for load cell measurements (hook load and WOB). The analytical approach was used on the Drillbotics project in 2017 and 2018. The plot below showcases the change in amplitude for load cell and torque measurements by varying the RPM when the BHA is unconfined (no riser in place).

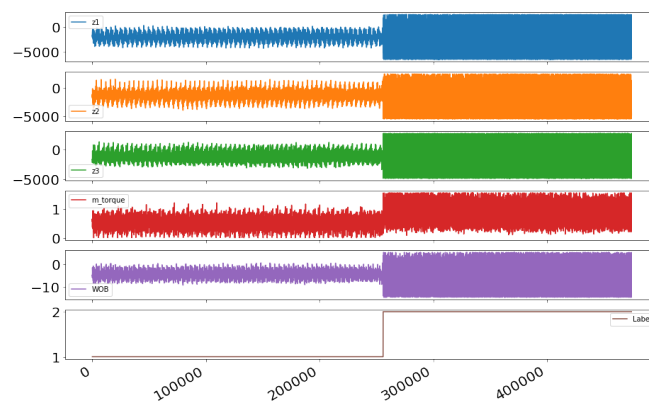


Figure 4.10: Plot of the raw data using surface sensors only.

Pairplot of the raw data

Finally, a pairplot for the surface vibration data can be generated:

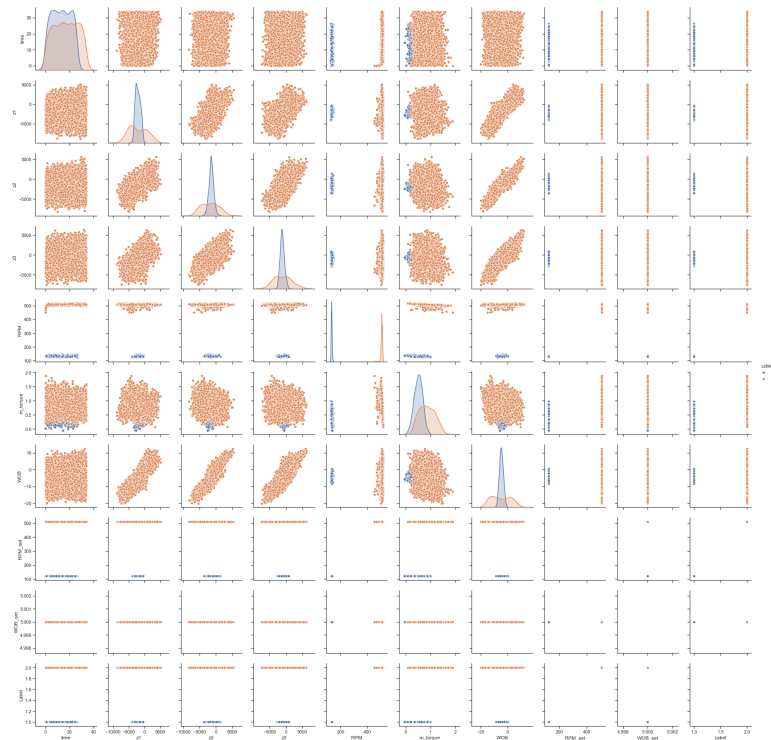


Figure 4.11: *Seaborn pairplot representing the case of normal vs moderately high vibrations.*

4.3.2 Normal pressure vs leak and overpressure

Data source

Similar to the vibration case above, pressure loss and overpressure has been solved analytically in the past by defining safe operating ranges for pressure so that the system could detect a drilling incident such as leak or overpressure if the thresholds got violated. A challenge, however using hardcoded pressure boundaries to determine when the system would go into a leak or an overpressure state, was that these events could be triggered if outliers or noise occurred in the measurements for a short duration of time.

For this purpose, two experiments were conducted in which two ball valves (that are installed on the rig) would get opened or closed fast (to artificially simulate either a leak in the mud system or a plugged nozzle).

Data concatenation

As is shown in Figure 4.12, data from the experiments (top) can be extracted to represent three pressure cases on the rig. These are (starting from left bottom)

leak, normal drilling and overpressure. Only the pressure and WOB data is kept for the pressure dataset.

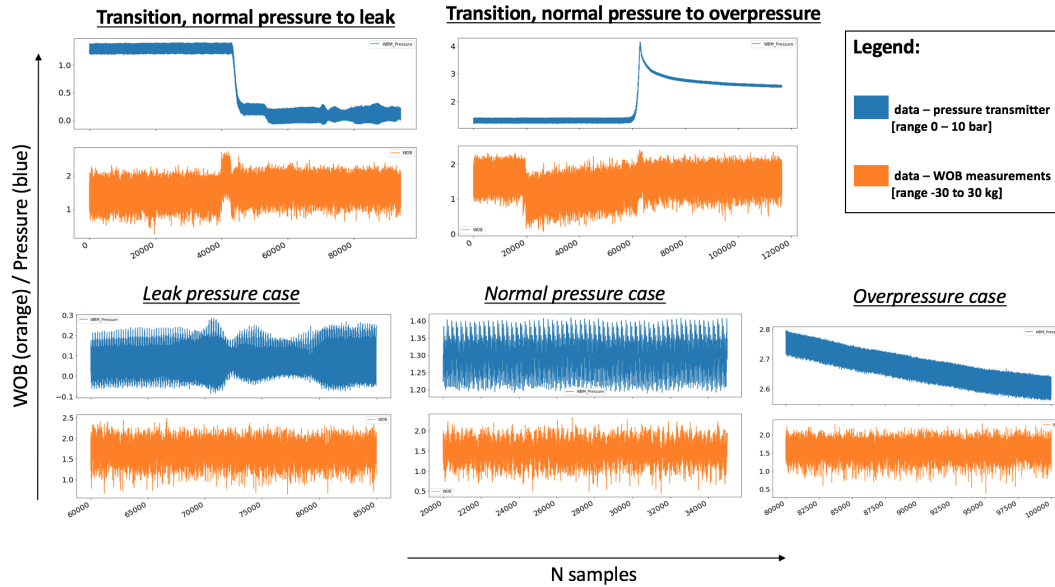


Figure 4.12: Pressure loss and overpressure cases for the mud system. The blue plots are measurements from the pressure transmitter integrated in the mud system and the orange plots represent WOB.

Data labeling

The data is labeled so that normal pressure is denoted by label 2, and leak and overpressure is labeled as 1 and 3, respectively. The transient phase from a normal case to an incident is currently not considered or labeled as separate classes (for instance an aim in the future is to predict that an incident is about to occur).

Describing the raw data

	WBM_Pressure	WOB	Label
count	60000.000000	60000.000000	60000.000000
mean	1.264475	1.638365	2.083333
std	1.118296	0.232395	0.759209
min	-0.087959	0.365880	1.000000
25%	0.109490	1.495900	1.750000
50%	1.292100	1.657200	2.000000
75%	2.651000	1.798600	3.000000
max	2.795900	2.480600	3.000000

Figure 4.13: Description of raw data extracted from experiments containing observations of normal pressure, leak and overpressure.

Pairplot of the raw pressure data

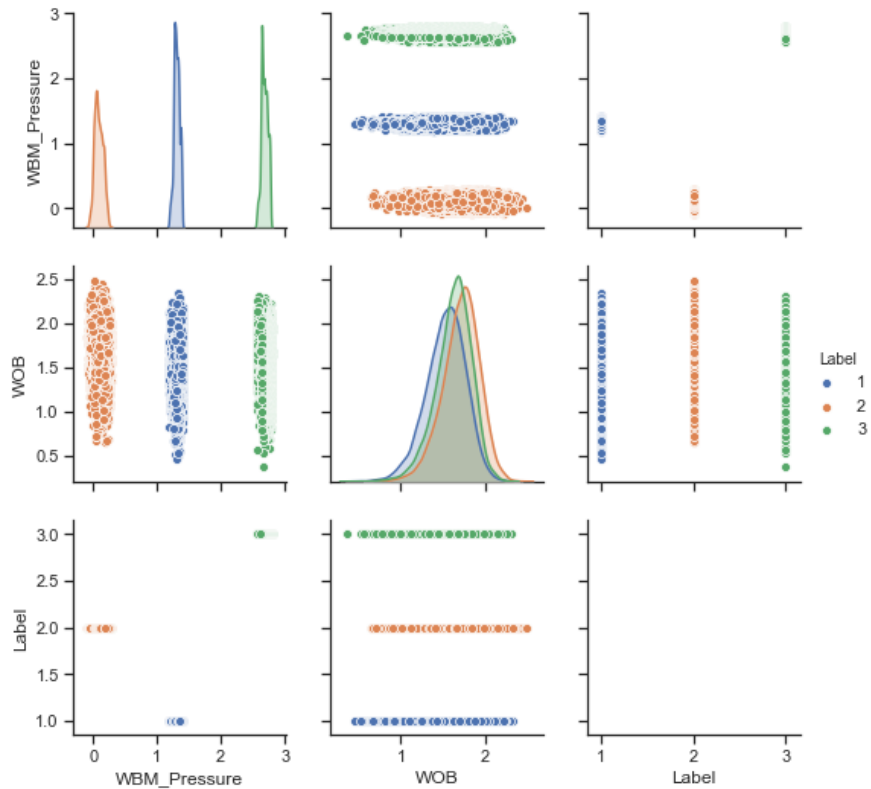


Figure 4.14: *Pairplot of pressure and WOB features for pressure cases.*

4.3.3 Rotating pipe vs stuck pipe

Data source

Same as with pressure, a critical drilling incident can occur if the pipe repeatedly gets stuck in the well, for instance due to stick-slip. Therefore, an experiment was conducted in which a wrench was used to repeatedly clamp the BHA until the bit would come to a complete stop during drilling to measure the impact on RPM feedback and torque. Once the bit came to a complete stop, the BHA was only kept at the stall-torque (where the brake in the top drive would kick in for a momentum above 6.5 Nm) for a very short duration of time before the clamp got released.

Data concatenation

To investigate whether a single occurrence of stuck pipe can be used to classify all incidents, only one stuck pipe occurrence is used to represent a stuck pipe. The same dataset can then be used to test the developed model (see chapter 7). In the Figure 4.15 below, RPM and torque is plotted to illustrate the two cases.

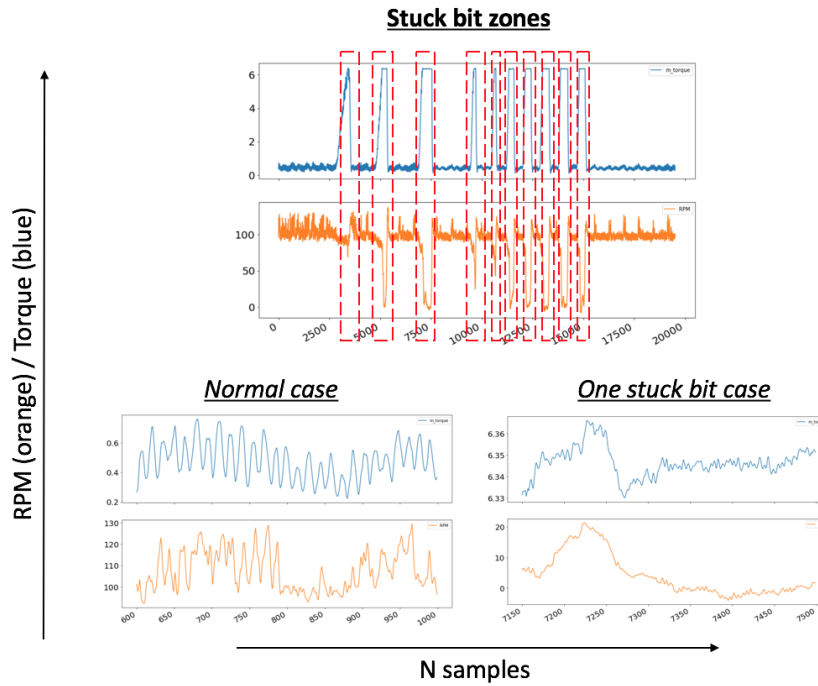


Figure 4.15: Normal drilling vs stuck pipe cases for stuck pipe classification. The blue data represents torque measurements and the orange RPM feedback in the top drive.

Data labeling

In terms of normal drilling, a sufficiently long interval to represent the base case is used. Normal drilling is labeled 1 and stuck pipe is labeled 2.

Describing the raw data

The standard description of the raw data is given in the Figure 4.16 below.

	time	z1	z2	z3	RPM	m_torque	WOB	Label
count	750.000000	750.000000	750.000000	750.000000	750.000000	750.000000	750.000000	750.000000
mean	3.203750	78.034107	-887.130923	-811.887739	59.346282	3.216892	-1.620984	1.466667
std	2.716032	281.190104	1189.734318	422.474338	52.181406	2.930695	1.269991	0.499221
min	0.500000	-514.120000	-2364.600000	-1348.100000	-4.047000	0.223510	-3.564000	1.000000
25%	0.656040	-192.470000	-2159.500000	-1109.375000	2.817425	0.462103	-2.970350	1.000000
50%	0.812085	147.260000	-397.200000	-1011.050000	96.089500	0.670640	-0.596125	1.000000
75%	6.093100	278.385000	289.680000	-535.507500	106.705000	6.345500	-0.427060	2.000000
max	6.249200	637.050000	633.520000	122.960000	129.480000	6.366100	-0.164430	2.000000

Figure 4.16: Description of raw data to later develop a stuck pipe model.

Pairplot of the stuck pipe raw data

A pairplot of all raw data for the two scenarios normal drilling and stuck pipe is shown in Figure 4.17 below.

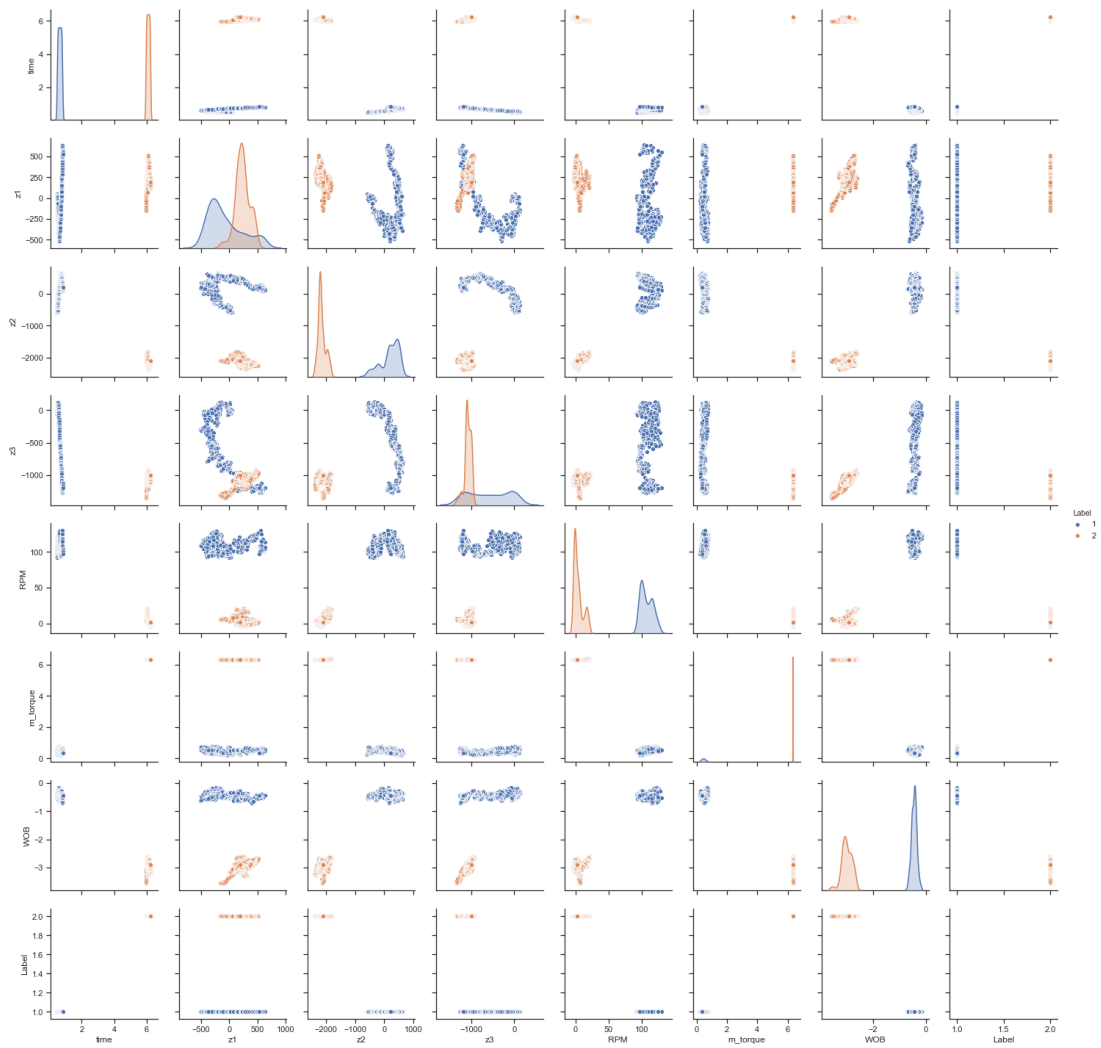


Figure 4.17: Pair plot of raw data gathered that represents either normal drilling or stuck pipe.

4.3.4 Normal drilling vs drill string twist off

Data source

An important incident to detect and confirm is if a twist off has occurred. While the process of predicting an *incoming* twist off incident is difficult using the laboratory drilling system, since the rotational speed is often configured to be in the 1000 to 1500 RPM range, a model that can be used to detect and confirm a twist off has been requested in the team. For this reason, an experiment was conducted in which the rig was allowed to operate at high WOB and RPM during cement drilling until a twist off of the aluminum drill pipe occurred.

Data concatenation

Figure 4.18 below represents a case where a twist off in the pipe connection occurs from fatigue. An illustration of a twist off in pipe connection is given on page 138 in (Hjelm & Nilsen, 2018) [6]

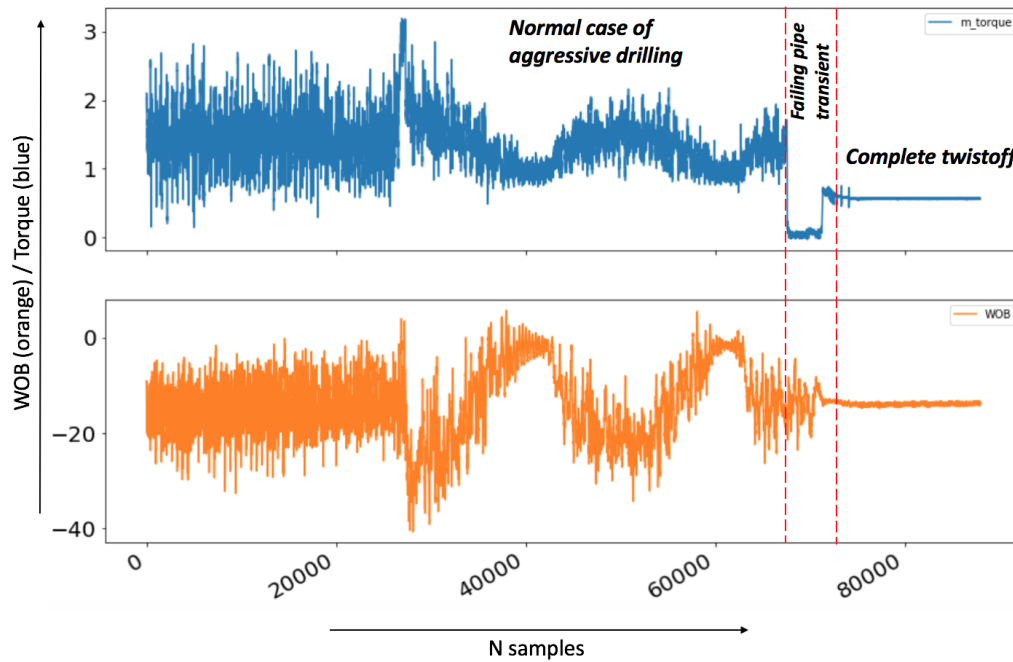


Figure 4.18: Data representing the transition going from normal drilling into a twist off of the aluminum drill pipe when drilling for a long duration of time with high RPM and WOB setpoints. The blue line represents torque and the orange WOB.

Data labeling

As with the other drilling incidents, normal drilling is labeled as 1 and twisted off as 2, by adding a Label column for each case before merging the normal drilling data and twist off datasets together.

Describing the raw data

The statistical properties of the raw dataset can be shown as:

	time	z1	z2	z3	RPM	t_bit	m_torque	depth	WOB	mse	
count	70000.000000	70000.000000	70000.000000	70000.000000	70000.000000	70000.0	70000.000000	70000.000000	70000.000000	70000.000000	70000.0
mean	3.868997	-5526.148009	-4788.633547	-4018.356815	887.042452	-1000000.0	1.288741	2.614516	-14.333137	181481.389367	1.0
std	2.474839	2756.706156	2501.353027	2559.636124	359.285059	0.0	0.476454	0.287978	7.009048	101193.390891	0.0
min	0.000000	-14967.000000	-16629.000000	-15124.000000	5.039200	-1000000.0	0.144730	1.524000	-40.656000	-22554.000000	1.0
25%	1.822875	-7420.750000	-6353.125000	-5668.750000	1029.100000	-1000000.0	0.967287	2.436100	-18.807000	152280.000000	1.0
50%	3.645750	-5486.600000	-4884.000000	-3676.500000	1033.400000	-1000000.0	1.296000	2.622800	-14.090000	196645.000000	1.0
75%	5.468650	-3616.375000	-3252.475000	-2532.600000	1036.100000	-1000000.0	1.589200	2.802625	-10.220000	240402.500000	1.0
max	8.854100	4845.500000	5077.000000	4238.000000	1058.300000	-1000000.0	3.189500	3.295100	5.686700	525480.000000	2.0

Figure 4.19: Description of natural features' statistical properties for twist off experiment conducted when drilling a homogeneous cement sample with high operating setpoints.

Pairplot of the twist off raw data

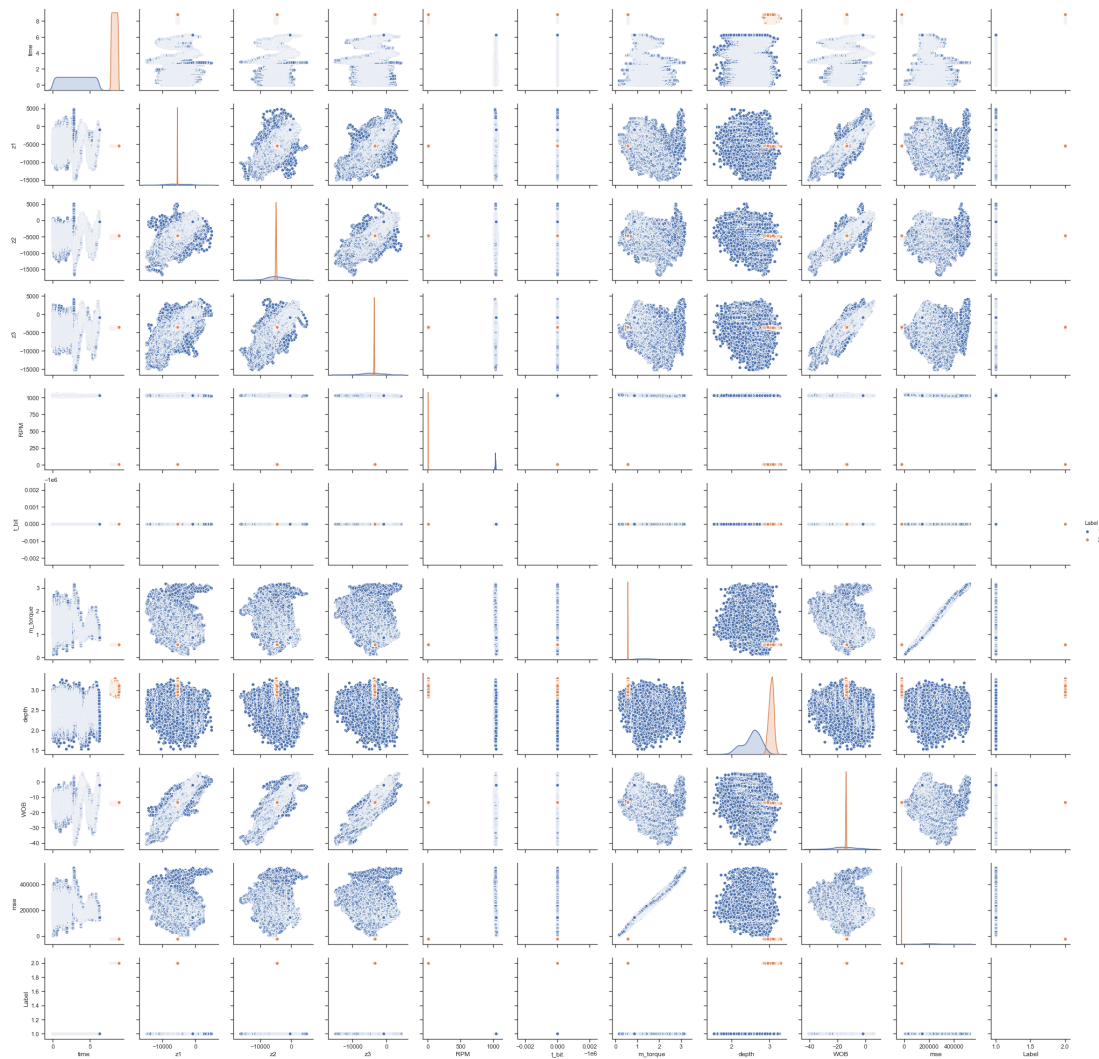


Figure 4.20: Pairplot of raw data that has been used to develop a twist off classification model.

4.4 Downhole data for vibration classification

Data source

When drilling with the downhole motor, there is no need to use a data swivel (slip ring) to transmit data from the non-rotating sensor sub and BHA. If however the top drive is used, a slip ring which is rated to 10 000 RPM [6] can be installed and used to receive data from the sensor sub. In this experiment, drilling the results from three operations where drilling has been performed with what is considered low, moderate and high vibration levels in the BHA.

To capture low vibrations (zero bit-rock interaction), the bit is lowered into a pre-drilled pilot hole section, approximately 5 mm below top of the rock. At this point, there is no contact between the bit and the bottom of the well. The valve opening is set to 80 % (BIT_{RPM} is approximately 925 RPM) and data is captured.

To capture moderate and high vibrations, the bit is lowered into a 355 mm deep slightly inclined wellbore, to resemble the conditions in a deviation well. The WOB-setpoint is 5 kg, and the rig drills at 60 ($BIT_{RPM} = 420$), 70 ($BIT_{RPM} = 733$), 80 ($BIT_{RPM} = 925$), 90 ($BIT_{RPM} = 1039$) and 100 % ($BIT_{RPM} = 1075$) valve opening. From data analysis, observations then get extracted that represent either moderate or high vibrations.

Data labeling

The data is labeled in such a way that low vibrations (idling RPM without rock contact) are denoted by label 1, moderate vibrations are denoted by label 2 and high vibrations (yet not severe) by label 3.

Describing the raw data

The statistical properties of the merged dataset can be shown as:

	acc_x	acc_y	acc_z	mag_x	mag_y	mag_z	gyr_x	gyr_y	gyr_z	temp	z1	z2	z3	opening	height	Label	BIT_RPM	WOB
count	35600.000000	35600.000000	35600.000000	35600.000000	35600.000000	35600.000000	35600.000000	35600.000000	35600.000000	35600.000000	35600.000000	35600.000000	35600.000000	35600.000000	35600.000000	35600.000000	35600.000000	35600.000000
mean	1.237890	-1.163181	3.116982	0.243756	0.941443	0.344647	0.020998	7.185840	-7.253299	20.906993	2992.786136	-2641.050510	-4913.678026	88.598034	-381.315871	2.601124	1009.998585	-4.561842
std	15.137782	14.841486	15.103534	14.282048	14.436311	15.167899	57.309898	55.870724	65.264722	16.668287	1454.226258	1982.537555	2124.086898	7.210674	91.789854	0.598132	59.673636	3.384308
min	-185.912506	-169.513748	-175.848755	-171.438751	-187.145004	-187.529999	-176.085007	-230.544998	-188.177505	-163.907501	-4121.263930	-10677.412109	-12202.413086	80.000000	-438.000000	1.000000	927.960000	-22.173408
25%	-1.802685	-3.467200	-0.261800	-0.117200	0.791840	-0.020200	-35.008750	-22.298437	-51.019063	24.250000	2025.441772	-4035.487610	-6541.868896	80.000000	-420.000000	2.000000	927.960000	-6.783785
50%	1.178485	-1.229912	2.817250	-0.105920	0.812560	-0.047440	0.026250	2.812252	-3.263481	24.500000	3025.713989	-2648.172485	-4936.717529	90.000000	-405.000000	3.000000	1044.320000	-4.572859
75%	4.075575	0.688884	6.281159	-0.081120	0.825980	-0.041840	32.689999	41.188437	31.771250	24.750000	4076.437073	-1149.989410	-3182.130737	90.000000	-379.000000	3.000000	1044.320000	-2.314788
max	182.717499	162.828751	183.478745	197.496246	173.250000	181.615005	203.076751	203.668756	192.456263	203.963751	9098.043945	6751.333496	2420.309176	100.000000	-24.333334	3.000000	1070.500000	9.613270

Figure 4.21: Description of raw data statistical properties for downhole vibration study conducted using a downhole motor and drilling a cement rock sample.

Pairplot of downhole data

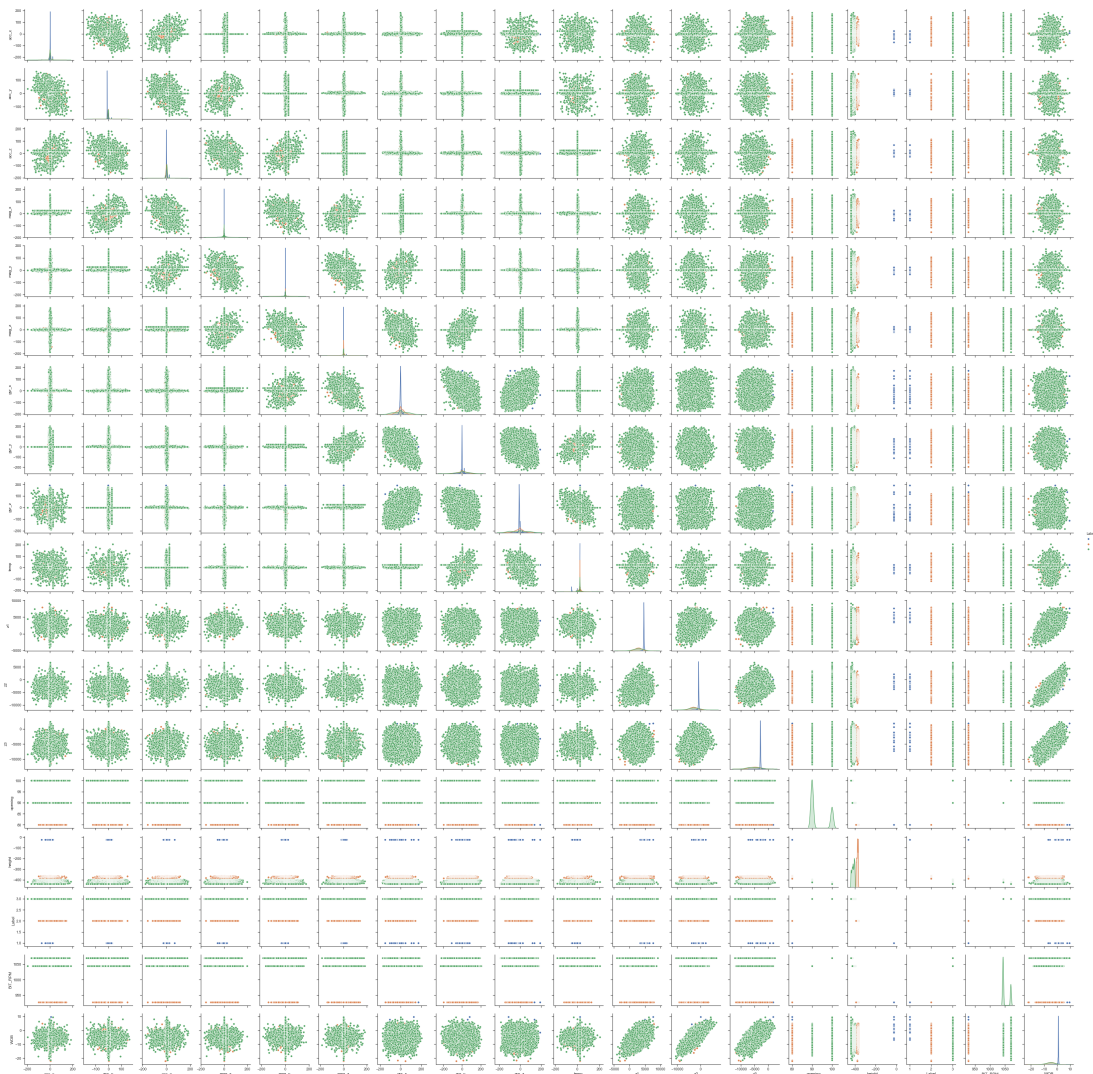


Figure 4.22: Pairplot of raw data that has been used to develop a downhole vibration level classification model.

4.5 Possible errors present in laboratory data

When collecting data, the following experimental errors were noted:

- the actual RPM is deviating from the RPM setpoint, due to the use of PWM in the Arduino Due at the time of the experiment. At 400 RPM setpoint, the actual RPM was approximately 280. At 600 RPM setpoint, the actual RPM was 510. At 800 RPM setpoint, the actual RPM was 760, and at 1000 RPM setpoint, the actual RPM was 1030. Even if the top drive has been re-tuned, this suggests that if one is interested to replicate the data collection, the actual RPM values should be used on the new system,
- the top drive and compressor both run on the 3-phase power grid at the University of Stavanger. Heavy machinery that also runs on the same 3-phase circuit was in different experiments at the same time as data for rock classification got collected. It is possible that from operating these machines in vicinity of the laboratory drilling rig signal noise has been picked up on the analog sensor measurements,
- for some of the experiments such as when drilling the salt-formation, the resulting wells turned out slightly deviated (possibly caused by a high ROP and vibrations) which might have had an effect on the data,
- the PID controller was previously running on the Arduino Due microcontrollers, and not in Python which might have an effect on the response of the hoisting system,
- The experiments were conducted using a PDC bit with two cutters and a BHA with three stabilizers. If changing parts of the drill string assembly, it is important to keep in mind that this has an effect on properties such as the natural frequency, length of assembly and stiffness, as well as the ROP, bit aggressiveness, wellbore friction and so on. The same applies for downhole vibration classification where data gets collected using a PDC bit with four cutters from Baker Hughes and a BHA consisting of a sensor sub, knuckle joint, and downhole motor.

Possibly having noisy data highlights the importance of adequate data cleaning before the data can be used to train models. While the training data should resemble actual drilling data as best as possible, such errors might cause invalid data or outliers to be present in the dataset. For this reason, the pre-processing techniques that have been used to clean the dataset is presented in chapter 5.

4.6 Volve data

4.6.1 Volve data aggregated to classify rock formations

Data source

A student at the University of Stavanger mined data collected from the Volve field, as introduced in chapter 1. Through a developed script that would extract data from XML files and manually extracting data from PDF files and daily drilling reports, a CSV file was generated for each well and well section. For Volve data rock formation classification, data has been extracted from wells 15-9-F(1/4/5/10/11/14/15).

Data concatenation

The data selected has been extracted from the 8-1/2 in sections of the wells mentioned above. From well 15-9-F1, data exists from four lateral sections. From wells 15-9-F4 and 15-9-F5, data exists for two lateral sections in each well. From wells 15-9-F10 and 15-9-F14, data exists from one lateral section. From wells 15-9-F11, data exists from three lateral sections. From well 15-9-F15, data exists from five lateral sections.

Data labeling

In the Volve formation data, seven different formations are present. In the 8-1/2" sections, only six formation types are present. Each formation type in the data is already labeled so that:

Rock specimen	Label	Number of samples (rows)
Claystone	1	598
Sandstone	2	1266
Siltstone	3	168
Tuff	4	0
Marl	5	470
Limestone	6	1099
Coal	7	9

Describing the raw data The raw data from the Volve wells that have been used to classify the six different geological formations encountered when drilling the 8-1/2 in sections that data has been collected can be described as:

	ROP_1	MSE	WOB_1	TORQUE_1	RPM	GEOLOGY NO	FLOW	MWIN	DEPTH OF CUT	BA	MUD TYPE NO
count	3610.000000	3.609000e+03	3610.000000	3610.000000	3610.000000	3610.000000	3610.000000	3610.000000	3610.000000	3610.000000	3610.000000
mean	23.492221	8.697353e+04	6.738609	18.978246	162.454931	3.501662	2258.185405	1.317269	2.528242	6.612185	1.396399
std	9.457476	7.650637e+04	3.413751	3.896662	33.347119	2.015766	259.169653	0.039908	1.406873	52.988287	0.739150
min	0.500000	2.133906e+02	-1.359386	0.000000	1.000000	1.000000	1.000000	1.280000	0.205870	-104.632709	1.000000
25%	17.374039	5.116995e+04	4.457521	16.800000	149.359659	2.000000	2126.400000	1.280000	1.686515	3.073965	1.000000
50%	23.832300	6.670954e+04	6.510133	18.579424	160.000439	2.000000	2227.000000	1.310000	2.488939	4.214830	1.000000
75%	29.896689	9.721137e+04	8.798507	20.828832	179.913198	6.000000	2389.279980	1.340000	3.130725	5.873004	1.000000
max	177.671236	1.102816e+06	86.470380	35.450000	311.000000	7.000000	3566.000000	1.470000	43.654276	3123.783529	3.000000

Figure 4.23: Certain statistical properties of field data from the Volve field that has been used for formation classification of different geological formations.

Plotting the raw data

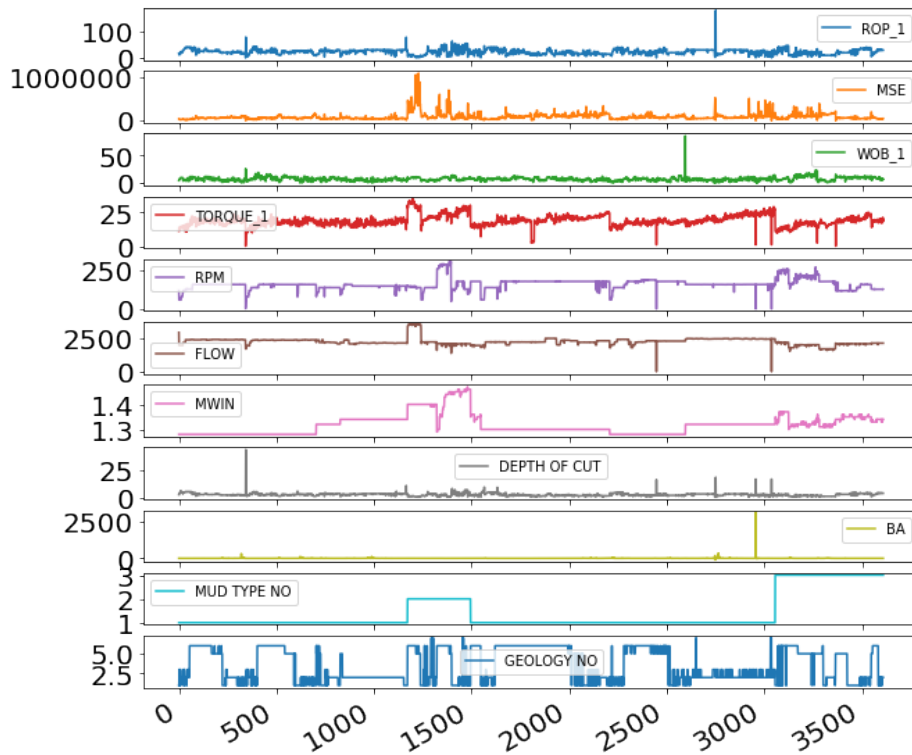


Figure 4.24: Plot of features from the Volve datasets that has been used for formation classification.

Pairplot of the raw data features

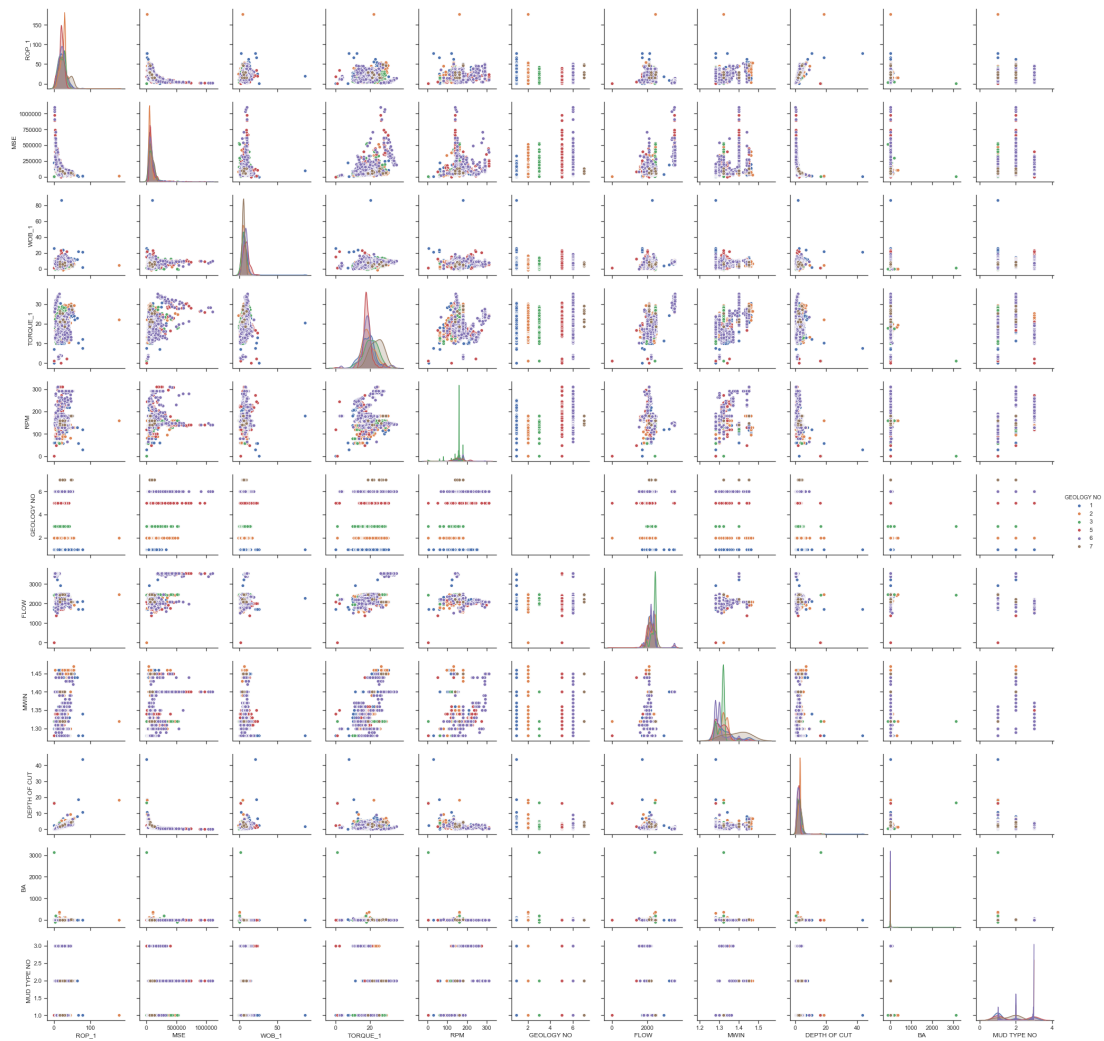


Figure 4.25: *Pairplot of all original features from the Volve datasets.*

4.6.2 Volve data aggregated to classify rig operations

Data source

The second set of field data from the Volve field is taken from F1C WB2/537RGUI. The data that has been extracted from this set represents three rig operations, i.e. Tripping Up, Tripping Down and Drilling. In chapter 8, a section describes an attempt to use the field data on a model that has been developed using laboratory data, to classify the rig operations. The data is sampled at 1 sample every 2.8 seconds, i.e. with a sampling frequency of 0.357 Hz.

Data concatenation

The most important data to extract from the datasets is the bit depth, crown block position, as well as drilling data such as torque, RPM and WOB. By analysis,

only a portion of the dataset is extracted.

Data labeling

The data is labeled so that label 1 corresponds to drilling, label 2 to Tripping Up (POOH) and label 3 to Tripping DOWN (RIH).

Rig operation	Label	Number of samples (rows)
Drilling	1	7500
RIH	2	1405
POOH	3	150

Describing the raw data

Describing the raw data The raw data from the Volve wells is used to classify the three rig operations can be described by:

	Sample	BIT_RPM_AVG	SURF_RPM	Depth	ROP_AVG	BLOCKCOMP	TORQUE_AVG	WOB_AVG	Label	RoBlock
count	9055.000000	9.050000e+03	9050.000000	9055.000000	9050.000000	9050.000000	9050.000000	9.050000e+03	9055.000000	9055.000000
mean	69559.685699	2.952420e+00	1.729102	2809.117527	0.004603	28.066800	20012.227998	1.097623e+05	1.188294	-10.727670
std	15491.154828	7.323734e-01	0.576470	116.897873	0.002880	11.758364	6540.453702	9.052030e+04	0.431266	15.702422
min	59455.000000	-1.079405e-28	-0.015444	2698.986000	0.000000	9.003955	4.132556	-1.021541e+06	1.000000	-36.725576
25%	61809.500000	3.218113e+00	1.949463	2731.736500	0.002302	18.125168	16178.386750	7.083941e+04	1.000000	-23.665976
50%	64601.000000	3.254758e+00	1.987829	2767.929000	0.005290	26.624986	20692.353000	1.441446e+05	1.000000	-12.818757
75%	67345.500000	3.269834e+00	2.001265	2810.253000	0.006824	37.515572	24645.647500	1.574129e+05	1.000000	0.000000
max	161593.000000	3.332110e+00	2.048009	3056.181000	0.009230	49.914883	33483.862000	1.985673e+05	3.000000	37.624763

Figure 4.26: *Certain properties of raw field data from the Volve field that is used to develop models for rig operation classification.*

Plotting the raw data

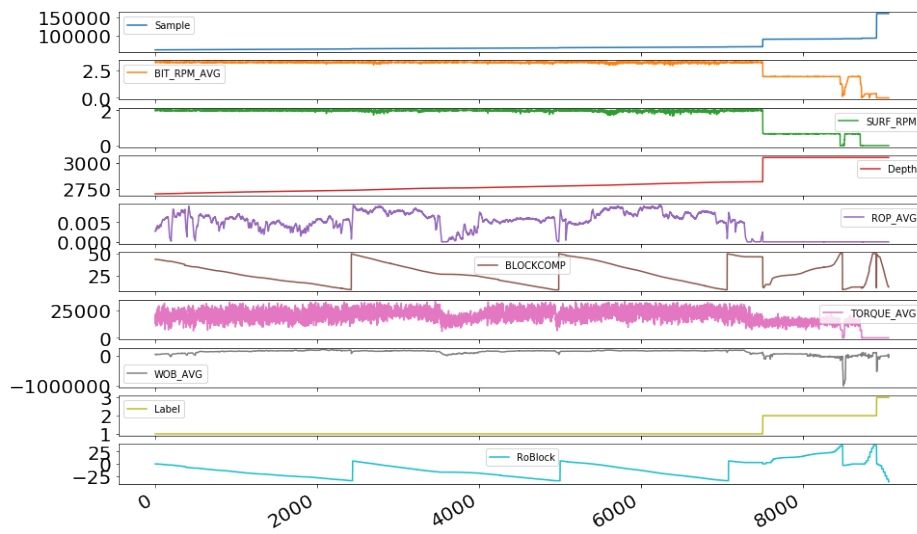


Figure 4.27: Plot of features from the Volve datasets that gets used for rig operation classification.

Pairplot of the raw data features

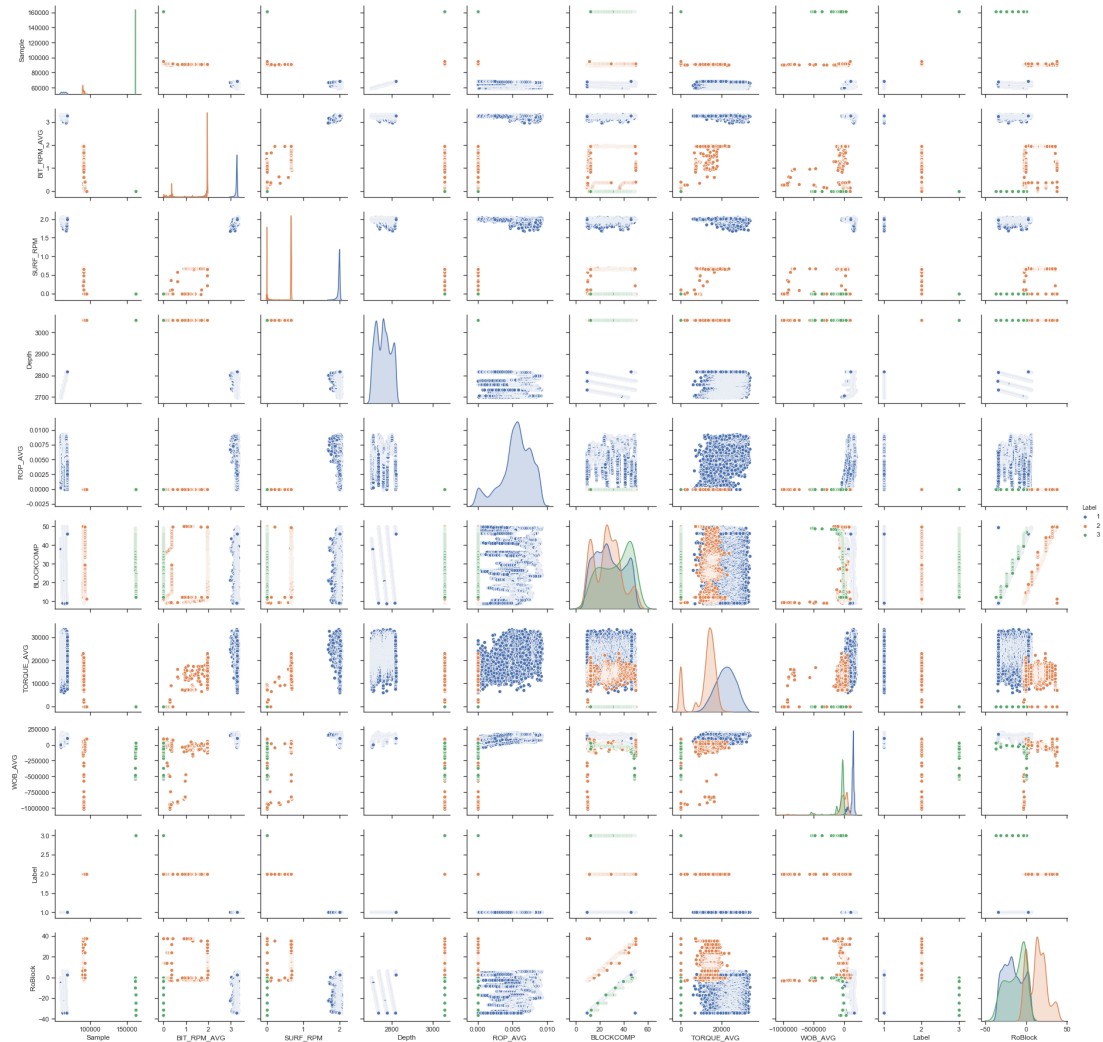


Figure 4.28: *Pairplot of all features from the Volve datasets that has been used for rig operation classification.*

4.7 Field data challenges

The greatest challenge when working with unprocessed field data is to our knowledge the absence of an all inclusive database, where all data is correlated and converted to one format. An example of this is how some data can be found in XML and PDFs files, and other data can only be accessed in the daily drilling reports. Another challenge is the quality of data, and the continuity. When data is required to either be manually inserted into reports, or the data is correlated manually by for instance comparing a daily drilling report with field data, both accuracy and quality assurance get diminished. The same applies when infor-

mation regarding the configuration and accuracy of logging tools and sensors is not provided as metadata, which means that one may be unfortunate and merge data from two completely different data sources that are not directly comparable. It is therefore our recommendation that in order to fully capitalize on available data, algorithms and technology a need exists to standardize drilling data flow schemes and integrate data-driven approaches (with physics-based models) to address uncertainties in drilling.

Chapter 5

Data Quality Improvement

In this chapter, the process that has been used to pre-process the raw data before features get created and selected is presented. The process includes downsampling the data to a frequency that represents the data, yet reduces the computational requirement, noise reduction and normalizing the data.

5.1 Down sampling experimental data

In the experiments conducted using the laboratory drilling rig, sensor data has been sampled with a frequency of 9600 Hz. This results in large amounts of data being generated, particularly when compared to well operations on the full scale, where sampling rate is usually around 10 Hz. Downsampling has therefore been important, and two approaches to downsample the data have been tested and evaluated. These are linear- and random downsampling. Linear downsampling of the dataset is performed through only a single line of code, as illustrated below (code to keep every 96 samples in the original dataset):

```
1 Dataframe_downsampled = ...  
   Dataframe_original.iloc [Dataframe_original.index % 96 == 0]
```

Downsampling can also be handled by randomly selecting a fraction of the original samples. Below, the code to randomly downsample a dataset to 1 % of the original data is shown:

```
1 Dataframe_downsampled = Dataframe_original.sample (frac=0.01)
```

Linear downsampling has been used for all laboratory datasets as part of the pre-processing procedure, but for some models, such as support vector machine, random downsampling has been used to extract a smaller fraction of the data.

5.2 Removing duplicates

While our datasets so far have not contained any duplicates, there are available functions to easily remove duplicates. When processing the data, the Pandas-library has been used. The function shown below has been taken from the Pandas library:

```
1 Dataframe.drop_duplicates()
```

5.3 Removing Missing data

A cell containing missing data is described in the data as a NaN (Not a Number) value. These missing numbers get identified during processing, and the preferred method to handle missing values has been to remove the respective row. Because our dataset initially has been sampled with a high sampling frequency, removing the whole row when a NaN value is identified has not been a problem for further use of the data. The NaN-values are identified and the row is removed using the command `Dataframe.dropna(axis=0, inplace=True)`, as shown below:

```
1 Dataframe.dropna(axis=0, inplace=True)
```

If smaller datasets get used, one could argue that removing the entire row of a missing data might not be beneficial or in some cases ruin the value of the dataset, and different methods can be applied to not have to remove the complete set of observations where one of the variables in the set contains an invalid or missing number.

5.4 Normalizing the data

Before the data gets normalized, hardcoded minimum and maximum boundary conditions for all variables get added for reasons mentioned in section 3.1.8. The steps that have been follow are as following:

Once the boundary conditions have been added, the *MinMaxScaler* function from the preprocessing library in Scikit Learn is used:

```
In [478]: x = Actualdataframe.values #returns a numpy array
min_max_scaler = preprocessing.MinMaxScaler()
x_scaled = min_max_scaler.fit_transform(x)
Actualdataframe1 = pd.DataFrame(x_scaled)
Actualdataframe1.head()
Actualdataframe1.tail()
```

```
Out[478]:
```

	0	1	2	3	4	5	6	7	8	9	10	11
315022	0.191490	0.253135	0.417865	0.499100	0.264680	0.533052	0.287500	0.533333	0.666667	0.533211	0.506167	1.0
315023	0.728255	0.495980	0.628250	0.511933	0.073751	0.533049	0.617495	0.533333	0.666667	0.533211	0.506167	1.0
315024	0.524186	0.593510	0.599655	0.514780	0.071098	0.532524	0.572452	0.533333	0.666667	0.533211	0.506167	1.0
315025	1.000000	1.000000	1.000000	1.000000	1.000000	1.000000	1.000000	1.000000	1.000000	1.000000	1.000000	1.0
315026	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	1.0

Figure 5.1: Normalizing data using *MinMaxScaler*.

After the normalization operation has been performed the columns that represent the different variables (features) must be renamed:

```
In [479]: Actualdataframe1.columns = ['z1', 'z2', 'z3', 'rpm', 'm_torque', 'depth', 'wob', 'RPM_set', 'WOB_set', 'mean_depth', 'ROP_instantaneous']
Actualdataframe1.head()
```

```
Out[479]:
```

	z1	z2	z3	rpm	m_torque	depth	wob	RPM_set	WOB_set	mean_depth	ROP_instantaneous	Label
0	0.384760	0.430665	0.484867	0.189860	0.058698	0.504306	0.433432	0.266667	0.55	0.503787	0.108638	0.0
1	0.437615	0.423505	0.347970	0.187920	0.065620	0.504298	0.403028	0.266667	0.55	0.503787	0.108638	0.0
2	0.353780	0.326060	0.333465	0.188633	0.062437	0.504776	0.337767	0.266667	0.55	0.503787	0.108638	0.0
3	0.414320	0.443105	0.428260	0.186467	0.084269	0.503684	0.428562	0.266667	0.55	0.503787	0.108638	0.0
4	0.463507	0.391265	0.451726	0.182893	0.101321	0.503246	0.435500	0.266667	0.55	0.503787	0.108638	0.0

Figure 5.2: Renaming the features.

The class labels that represent the different classes have also been normalized which means that these also must be renamed to their original label:

```
In [482]: for i in range(len(Actualdataframe['Label'])):
if Actualdataframe['Label'][i] == 1.0:
Actualdataframe['Label'][i] = 5
elif Actualdataframe['Label'][i] == 0.0:
Actualdataframe['Label'][i] = 3
elif Actualdataframe['Label'][i] == 0.5:
Actualdataframe['Label'][i] = 4
```

```
In [483]: Actualdataframe.tail()
```

```
Out[483]:
```

	z1	z2	z3	rpm	m_torque	depth	wob	RPM_set	WOB_set	mean_depth	ROP_instantaneous	Label
315022	0.191490	0.253135	0.417865	0.499100	0.264680	0.533052	0.287500	0.533333	0.666667	0.533211	0.506167	5.0
315023	0.728255	0.495980	0.628250	0.511933	0.073751	0.533049	0.617495	0.533333	0.666667	0.533211	0.506167	5.0
315024	0.524186	0.593510	0.599655	0.514780	0.071098	0.532523	0.572452	0.533333	0.666667	0.533211	0.506167	5.0
315025	1.000000	1.000000	1.000000	1.000000	1.000000	1.000000	1.000000	1.000000	1.000000	1.000000	1.000000	5.0
315026	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	5.0

Figure 5.3: Changing normalized class labels back to their original label.

As can be observed in row 315025 and 315026, the hardcoded maximum and minimum value have been normalized to 1.0 and 0.0, respectively.

5.5 Outlier removal using IQR method

The data has been evaluated both when outliers have been removed from the data, and when these outliers have been kept. For outlier removal, the interquartile range (IQR) method has been used.

First, each column (feature) of observation has been sorted:

```
In [507]: sortz1 = sorted(Actualdataframe['z1'])
sortz2 = sorted(Actualdataframe['z2'])
sortz3 = sorted(Actualdataframe['z3'])
sortrpm = sorted(Actualdataframe['rpm'])
sortm_torque = sorted(Actualdataframe['m_torque'])
sortdepth = sorted(Actualdataframe['depth'])
sortwob = sorted(Actualdataframe['wob'])
sortwobset = sorted(Actualdataframe['WOB_set'])
sortrpmset = sorted(Actualdataframe['RPM_set'])
sortmeandepth = sorted(Actualdataframe['mean_depth'])
sortrop = sorted(Actualdataframe['ROP_instantaneous'])
```

Figure 5.4: *Sorting each feature in the dataset.*

After the features have been sorted separately, the upper and lower quartiles of each feature get identified by the following method:

```
In [508]: q1z1, q3z1 = np.percentile(sortz1, [25,75])
q1z2, q3z2 = np.percentile(sortz2, [25,75])
q1z3, q3z3 = np.percentile(sortz3, [25,75])
q1rpm, q3rpm = np.percentile(sortrpm, [25,75])
q1m_torque, q3m_torque = np.percentile(sortm_torque, [25,75])
q1depth, q3depth = np.percentile(sortdepth, [25,75])
q1wob, q3wob = np.percentile(sortwob, [25,75])
q1wobset, q3wobset = np.percentile(sortwobset, [25,75])
q1rpmset, q3rpmset = np.percentile(sortrpmset, [25,75])
q1meandepth, q3meandepth = np.percentile(sortmeandepth, [25,75])
q1rop, q3rop = np.percentile(sortrop, [25,75])
```

Figure 5.5: *Identifying upper and lower quartiles.*

Once the upper and lower quartiles have been identified, the inter quartile range (IQR) gets calculated using equation 3.7 in the theory chapter:

```
In [509]: iqrz1 = q3z1 - q1z1
iqrz2 = q3z2 - q1z2
iqrz3 = q3z3 - q1z3
iqrrpm = q3rpm - q1rpm
iqr_m_torque = q3m_torque - q1m_torque
iqrdepth = q3depth - q1depth
iqrwob = q3wob - q1wob
iqrwobset = q3wobset - q1wobset
iqrrpmset = q3rpmset - q1rpmset
iqrmeandepth = q3meandepth - q1meandepth
iqrrop = q3rop - q1rop
```

Figure 5.6: *Illustration of how IQR gets calculated.*

When the IQR has been calculated for each feature, one must use equations 3.8 and 3.9 to determine the lower and upper limits. These limits determine the threshold in which all observations that fall outside of the threshold are treated as outliers and removed.

```
In [510]: lower_boundz1 = q1z1 - (1.5 * iqrz1)
upper_boundz1 = q3z1 + (1.5 * iqrz1)
lower_boundz2 = q1z2 - (1.5 * iqrz2)
upper_boundz2 = q3z2 + (1.5 * iqrz2)
lower_boundz3 = q1z3 - (1.5 * iqrz3)
upper_boundz3 = q3z3 + (1.5 * iqrz3)
lower_boundrpm = q1rpm - (1.5 * iqrrpm)
upper_boundrpm = q3rpm + (1.5 * iqrrpm)
lower_boundm_torque = q1m_torque - (1.5 * iqrmtorque)
upper_boundm_torque = q3m_torque + (1.5 * iqrmtorque)
lower_bounddepth = q1depth - (1.5 * iqrdepth)
upper_bounddepth = q3depth + (1.5 * iqrdepth)
lower_boundwob = q1wob - (1.5 * iqrwob)
upper_boundwob = q3wob + (1.5 * iqrwob)
lower_boundwobset = q1wobset - (1.5 * iqrwobset)
upper_boundwobset = q3wobset + (1.5 * iqrwobset)
lower_boundrpmset = q1rpmset - (1.5 * iqrrpmset)
upper_boundrpmset = q3rpmset + (1.5 * iqrrpmset)
lower_boundmeandepth = q1meandepth - (1.5 * iqrmeandepth)
upper_boundmeandepth = q3meandepth + (1.5 * iqrmeandepth)
lower_boundrop = q1rop - (1.5 * iqrop)
upper_boundrop = q3rop + (1.5 * iqrop)
```

Figure 5.7: *Upper and lower limit of the IQR range.*

The lower and upper boundaries that have been identified in the dataset that gets used to train the rock formation classification models in chapter 7 are shown in the figure below:

```
In [511]: print([lower_boundz1, upper_boundz1])
print([lower_boundz2, upper_boundz2])
print([lower_boundz3, upper_boundz3])
print([lower_boundrpm, upper_boundrpm])
print([lower_boundm_torque, upper_boundm_torque])
print([lower_bounddepth, upper_bounddepth])
print([lower_boundwob, upper_boundwob])
print([lower_boundwobset, upper_boundwobset])
print([lower_boundrpmset, upper_boundrpmset])
print([lower_boundmeandepth, upper_boundmeandepth])
print([lower_boundrop, upper_boundrop])

[0.12575750000000005, 0.62949749999999999]
[0.15274250000000001, 0.6218425]
[0.15895556249999993, 0.6323040625]
[-0.25825333333333333, 0.94964]
[-0.020069121071012802, 0.23956184516880094]
[0.48399, 0.53854999999999999]
[0.18423229166666666, 0.591670625]
[0.47500000000000014, 0.67499999999999998]
[-0.13333333333333336, 0.93333333333333333]
[0.48384185725075507, 0.5386009272788834]
[0.04728267409439854, 0.5921502050665313]
```

Figure 5.8: *Upper and lower boundaries for the dataset used to develop rock formation classification models. Please note that the IQR method is conducted after the data has been normalized, which is why all boundaries are values between 0 and 1.*

The results from the IQR method are based on historical data. When implementing the supervised learning models in the real-time control system, these determined upper and lower boundaries can be used to discard observations that fall outside of the permitted range. After the threshold to keep has been determined, the following code (Figure 5.9) is run in order to convert observations that lie outside of the permitted range to NaN.

```
In [512]: Actualdataframe['z1'].where((Actualdataframe['z1'] >= lower_boundz1), inplace=True)
Actualdataframe['z1'].where((Actualdataframe['z1'] <= upper_boundz1), inplace=True)
Actualdataframe['z2'].where((Actualdataframe['z2'] >= lower_boundz2), inplace=True)
Actualdataframe['z2'].where((Actualdataframe['z2'] <= upper_boundz2), inplace=True)
Actualdataframe['z3'].where((Actualdataframe['z3'] >= lower_boundz3), inplace=True)
Actualdataframe['z3'].where((Actualdataframe['z3'] <= upper_boundz3), inplace=True)
Actualdataframe['rpm'].where((Actualdataframe['rpm'] >= lower_boundrpm), inplace=True)
Actualdataframe['rpm'].where((Actualdataframe['rpm'] <= upper_boundrpm), inplace=True)
Actualdataframe['m_torque'].where((Actualdataframe['m_torque'] >= lower_boundm_torque), inplace=True)
Actualdataframe['m_torque'].where((Actualdataframe['m_torque'] <= upper_boundm_torque), inplace=True)
Actualdataframe['wob'].where((Actualdataframe['wob'] >= lower_boundwob), inplace=True)
Actualdataframe['wob'].where((Actualdataframe['wob'] <= upper_boundwob), inplace=True)
```

Figure 5.9: *Setting outlying data to zero, keeping in range values.*

After the outliers have all been converted to NaN, a count of the number of outliers in each variable can be printed, in order to evaluate the number of observations from each feature that are regarded as outliers by the method:

```
In [513]: Actualdataframe.isnull().sum()
Out[513]: z1          9763
z2          10478
z3          8403
rpm           1
m_torque     14923
depth         0
wob         12048
RPM_set       0
WOB_set       0
mean_depth    0
ROP_instantaneous 0
Label         0
TF1           0
TF2           0
TF3           0
TF4           0
TF5           0
MSE           0
dtype: int64
```

Figure 5.10: *Counting number of identified outliers that have been converted to NaN.*

As described earlier, the NaN values can be dropped from the dataset using the `.dropna()` function.

The accuracy and performance of supervised learning models developed on training data where outliers have either been kept or removed are presented in chapter 7 and chapter 8.

5.6 Four-plots for WOB and Torque

In chapter 3, the concept of using a four-plot is discussed. The plots included in a typical four-plot are so-called *time or scatter plot*, *lag plot*, *histogram plot* and *Q-Q plot*. To generate the plots, functions from the Pandas and SciPy (Scientific Python) libraries have been used.

In the figures below, four-plots have been generated both before and after the observations for torque and WOB have been pre-processed. In the plot below (Figure 5.11), the un-processed data for WOB is shown:

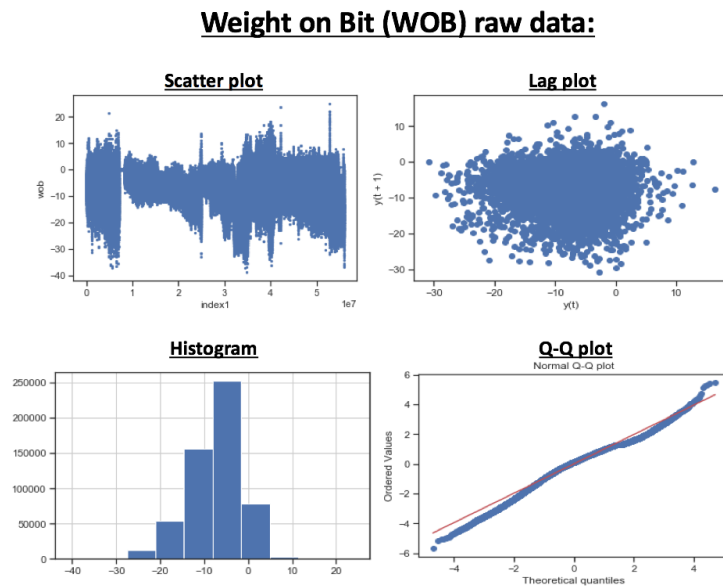


Figure 5.11: *Four-plot showing the raw data for WOB (before the data has been pre-processed by methods discussed above).*

After missing data, invalid data, duplicates and outliers have been removed, and after the dataset has been normalized, we plot the data once again. As seen in Figure 5.12, the normal distribution has improved, as can be observed from the histogram plot:

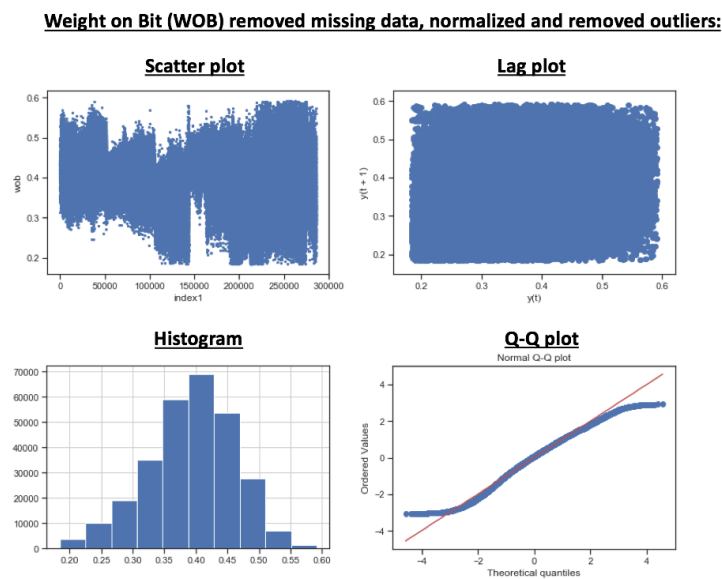


Figure 5.12: *Four-plot showing the plotted processed data of WOB.*

Repeating the same process for the torque measurements, it can once again be

observed that the most optimal normal distribution is achieved once the data has been pre-processed.

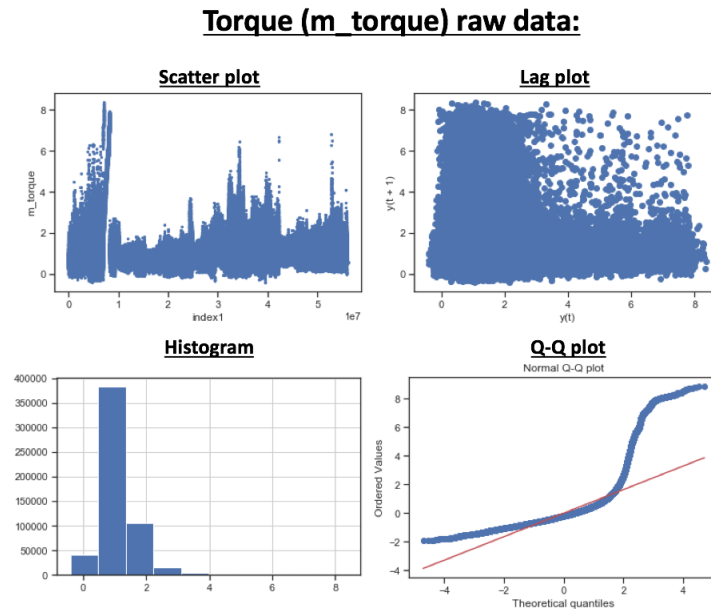


Figure 5.13: Four-plot showing the plotted raw data of torque.

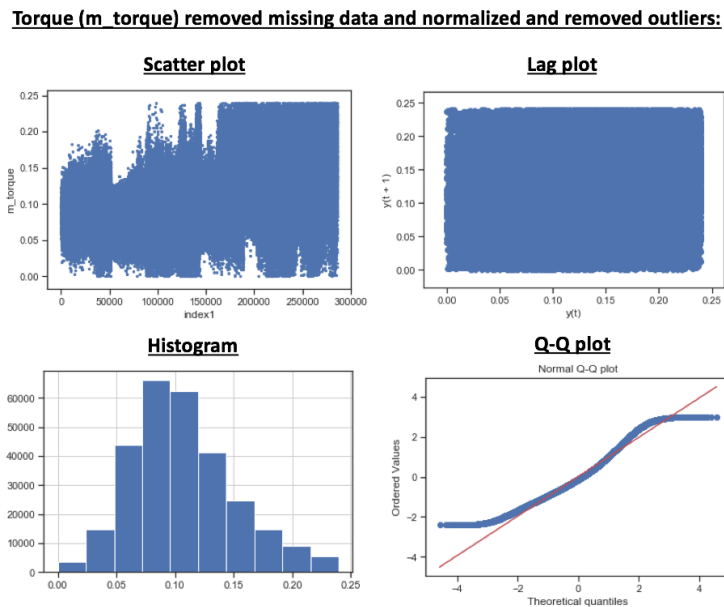


Figure 5.14: Four-plot showing the plotted processed data of torque.

Chapter 6

Feature Engineering and Optimization

In this chapter, the process of developing, evaluating and selecting which features that are most valuable to develop the different models gets presented. Because the objective is to classify different rock formations and drilling activities or incidents in real-time, several artificial features can be engineered to distinguish between classes. In this chapter, we commonly refer to natural features as observations that originally were collected from sensors. Engineered features on the other hand is used to describe features that have been created using the natural features (raw data observations). This helps us to separate the original variables from those that have been created after the data has been pre-processed.

6.1 Drilling Feature Engineering

6.1.1 Natural features

The natural features when classifying rock formations, rig operations and drilling incidents on the laboratory-scale drilling rig are:

Feature	Unit	Min Range	Max Range
LC1	[g]	-10000	10000
LC2	[g]	-10000	10000
LC3	[g]	-10000	10000
RPM	[rev/min]	0	1500
Torque	[Nm]	0	8.59
Depth	[mm]	0	1000
Wob	[kg]	-30	30
Pressure	[bar]	0	10
Rock formation	[#]	1	6

For the Volve field data used to classify different formations, the natural features that are extracted from all available data are:

Feature	Unit	Min Range	Max Range
ROP	[m/hr]	0.50	177.67
MSE	[Pa]	2.13×10^2	1.10×10^6
WOB	[kN]	-1.36	86.47
Torque	[kNm]	0	35.45
RPM	[rev/min]	0	311
Q	[LPM]	0	3566
MW_{in}	[s.g.]	1.28	1.47
DOC	[mm]	0.2059	43.65
BA	[-]	-104.63	3123.78
Mudtype	[#]	1	3
Geological formation class	[#]	1	7

Lastly, the Volve field data used to classify rig operations are:

Feature	Unit	Min Range	Max Range
Bit RPM	[rev/min]	0	3.33
Surf RPM	[rev/min]	0	2.05
Depth	[m]	2698.99	3056.18
ROP	[m/hr]	0	0.0092
Block pos.	[m]	9.00	49.91
Torque	[kNm]	0	33.48
WOB	[kN]	0	199
Operation class	[#]	1	3

In the tables above, LC(1/2/3) denote the hook load strain gauge measurements from load cells, Q denotes flow rate, MW denotes mud weight, DOC denotes depth of cut, BA denotes the bit aggressiveness, block position is a measurement of the crown block position (in the derrick). There exists uncertainty as to whether the Volve rig operation that is not tripping is Rotating On Bottom (ROnB) or indeed drilling. Based on the RPMs, it is unlikely that the data results from drilling, given the low ROPs as well (but instead ROnB). The uncertainty still does not have an effect on the further process of developing models to separate the three activities from each other.

6.1.2 Artificial Feature Engineering

Several drilling-related features can be created from the above natural features. The following features have been created and implemented in either one or several machine learning models:

Features constructed before datasets are merged

Mean depth is a feature that is created for the laboratory-scale drilling rig to output the average height sensor measurement of the drill floor (which travels along with the drill string assembly).

The algorithm used to calculate the mean depth [mm] value for each 9600 samples (i.e. one mean constant mean depth for all observations taken in a single second) can be shown in Figure 6.1:

```

1
2 a= math.ceil(len(DataSetX['time'])/9600) # Calculate the number of iterations to perform
3 rest = len(DataSetX['depth']) # Used to calculate how long dataset is
4 #print(rest)
5
6 for i in range(a):
7
8     remaining = min(rest, 9599) # Used to calculate how many remaining values are left.
9     datamean = DataSetX['depth'][i*9600 : i*9600 + remaining].mean() # Calculates the mean value of 9600 samples
10
11     for j in range(remaining + 1):
12         DataSetX['mean_depth'][i*9600 + j] = float(datamean) # Stores mean value in pre-created column
13         # (for all samples in a second (9600))
14
15     rest -= 9600
16

```

Figure 6.1: *The algorithm shown is used to calculate the mean depth measurement of 9600 observations corresponding to one mean depth value every second. The calculated mean-value is stored for all 9600 observations in a new feature column called mean_depth.*

Rate of Penetration (ROP) denotes the drilling speed; i.e. the change of bit depth per time unit (usually given in m/hour or ft/hour). On the laboratory drilling rig, the ROP can be calculated by a step counter (in real-time), which keeps track of how many pulses that get executed on the Arduino Due microcontrollers.

When collecting data from the high frequency DAQ with however, this information is not available, and we therefore use the mean depth value found above to calculate the ROP as a function of rate of change of depth. The algorithm calculates the difference in mean depth between t and $t-1$ (between every 9600 samples), as is shown in Figure 6.3:

```

1 aal = math.ceil(len(DataSet1['time'])/9600) # Calculate number of iterations to run
2 rest1 = len(DataSet1['mean_depth']) # Calculate length of dataset
3 initial1 = 0 # Defines a counter
4
5 #print(rest1)
6
7 for i in range(aal): # Runs iteration for n/9600 (seconds as 1 sec = 9600 samples) times
8     #print(rest1)
9     remaining1 = min(rest1, 9599) # Calculate how many remaining rows before each iteration is run
10    deltadepth1 = DataSet1['mean_depth'][i*9600] - DataSet1['mean_depth'][i*9600 - initial1]
11    deltatime1 = DataSet1['time'][i*9600] - DataSet1['time'][i*9600 - initial1]
12    rop1 = deltadepth1 / deltatime1 # ROP gets computed as change of depth per time (1s)
13
14    initial1 = initial1 + 9600
15
16    #print(rop1)
17
18    for j in range(remaining1 + 1): # Fills all cells with ROP calculation (same ROP per 9600 samples)
19        DataSet1['ROP_instantaneous'][i*9600 + j] = float(rop1*60)
20
21    rest1 -= 9600
22 DataSet1

```

Figure 6.2: Algorithm performs $n/9600$ iterations, that is one iteration to calculate the ROP which is the change in the mean depth value every 9600 samples, based on the mean-value calculations found above. The same ROP value is written to the $ROP_{instantaneous}$ column for all rows that the ROP value got calculated from.

6.1.3 Drilling Features Constructed after processing

The following features have been constructed after data has been processed to reduce the complexity in the real-time system of first calculating some features, then processing the raw data, and again calculating additional features. Furthermore, the process of normalizing the data require us to hardcode thresholds so that independent of what the maximum and minimum measurements are, the values in the training data correspond to values from future drilling, seeing as hardcoding for instance a maximum and minimum MSE or standard deviation based on only historical data can provide errors at the end.

Mechanical Specific Energy (MSE) is according to (Hamrick, 2011) [54] a measure of the mechanical energy required to remove a unit volume of rock, and is a concept that was introduced by Teale in 1965. The lower the MSE, the less energy is used to remove the same rock volume, which can be explained by that if a formation is broken into smaller fragments than required (to lift the cuttings out of the well) or too large fragments, which means that additional energy must be used downhole to reduce the size of the cuttings, energy is wasted. As technology advancements were made in the early 2000s in terms of accurate downhole tools

and state-of-art sensors, Dupriest, Koederlitz and Weis began computing the MSE during drilling on full scale rigs, and ever since, MSE has been a standard for surveillance of rig data.

Teale derived the following equation for MSE (Fear & Pessier, 1992) [55]:

$$E_s = \frac{WOB}{A_{bit}} + \frac{120 \times \pi \times RPM \times \tau}{A_{bit} \times ROP} \quad (6.1)$$

where: E_s = Specific energy, WOB = Weight on bit [lbs], A_{bit} = Area of bit [in²], RPM = Revolutions per minute [rev/min], τ = Torque [lb_f], ROP = Rate of penetration [ft/hr].

Teale also derived an equation for minimum specific energy, suggesting that the maximum efficiency of drilling is reached when the minimum specific energy is approximately equal to the compressive rock strength (Fear & Pessier, 1992) [55]:

$$\frac{E_{smin}}{E_s} = 1 \quad (6.2)$$

The equation that is used to calculate the MSE [Pa] on our system is (Hjelm & Nilsen, 2018) [6]:

$$MSE = \frac{WOB \times g}{\frac{\pi}{4} \times D_{bit}^2} + \frac{2 \times \pi \times RPM \times \tau}{\frac{\pi}{4} \times D_{bit}^2 \times ROP} \quad (6.3)$$

where: MSE = Mechanical specific energy [Pa], WOB = Weight on bit [kg], D_{bit} = Drill bit diameter [m], RPM = Surface RPM [rev/min], τ = Torque [Nm], ROP = Rate of penetration [m/hr].

As is shown in (Fear & Pessier, 1992) [55], $ROP = f(WOB, \mu, RPM, MSE, \text{bit})$

diameter):

$$ROP = \frac{13.33 \times \mu \times N}{D_B \left(\frac{E_s}{WOB} - \frac{1}{A_B} \right)} \quad (6.4)$$

$$\mu = 36 \times \frac{\tau}{D_B \times WOB} \quad (6.5)$$

where: ROP = Rate of penetration [ft/hr], μ = Coefficient of sliding friction, D_{bit} = Drill bit diameter [in], τ = Torque [lb_f], RPM = Revolutions per minute [rev/min], A_B = Area of bit [in²].

MSE can be used for classifying different rock formations, as is illustrated in Figure 6.3, where the MSE is calculated for a soft and hard formation using the laboratory setup at the University of Stavanger. See [56] for how specific energy (MSE) is used to characterize geotechnical sites, indicating the optimality of using this concept for rock classification.

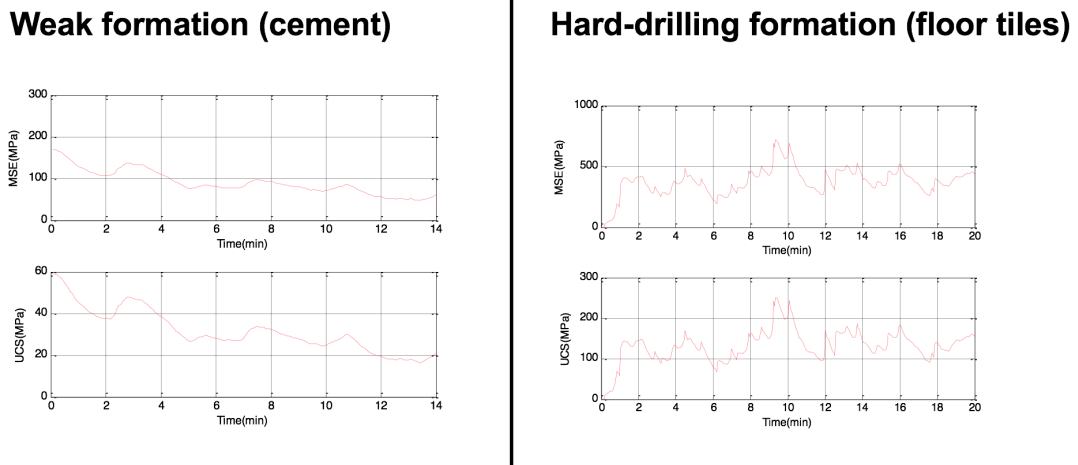


Figure 6.3: *Difference in MSE (and UCS) when drilling a soft rock (cement) and hard rock (floor tiles) with the laboratory-scale drilling rig.*

Depth of cut (DOC) can be described as the axial distance that the drill bit

cuts into the formation per revolution, which is dependent on the ROP and RPM. While the ROP and RPM feedback is already measured, the calculation of DOC gives the machine an additional parameter to distinguish between the drilled formations. The formula to calculate the DOC is [57]:

$$DOC = \frac{ROP}{5 \times RPM} \quad (6.6)$$

where: DOC = Depth of cut [in/rev] ROP = Rate of Penetration [ft/hr] RPM = Revolutions per minute [rev/min]

On the system, the DOC is calculated in mm/rev, hence:

$$DOC = \frac{ROP[m/hr] \times 16.40}{RPM[rev/min]} \quad (6.7)$$

Bit aggressiveness (BA), according to (Karadzhova, 2014), can be determined by how high the DOC of a drill bit is, and depend on the bits backrake angle (cutter angle) and the exposure of cutters. It is common knowledge that since PDC bits shear the formation, they are more aggressive than roller cone bits, and since the PDC bits produce a higher torque from the interaction with the formation, PDC bits are used on the laboratory-scale drilling rig at UiS. In 1992, Pessier and Fear derived the following equation for bit aggressiveness [58]:

$$\mu = \frac{36 \times \tau}{WOB \times D_{bit}} \quad (6.8)$$

where: τ = Torque [lb_f], WOB = Weight on bit [lbs], D_{bit} = Bit diameter [in].

d-exponent describes the so-called *drillability* of different formations. The d-

exponent, that was derived by Bingham, Jordan and Shirley in the late 1960s, is typically used to correct the drilling rate (Akisanmi, 2016) [13]:

$$d = \frac{\log\left(\frac{ROP}{60 \times RPM}\right)}{\log\left(\frac{12 \times WOB}{10^3 \times D_{bit}}\right)} \quad (6.9)$$

where: ROP = Rate of penetration [ft/hr], WOB = Weight on bit [lbf], D_{bit} = Bit diameter [in].

If one wants to correct the d-exponent to also take into account differences in the pore-pressure and equivalent circulating density, the following equation can be applied (Akisanmi, 2016) [13]:

$$d_{corrected} = \frac{NPP}{ECD} \times d \quad (6.10)$$

where: NPP = Normal pore pressure gradient [psi/ft], ECD = Equivalent circulating density [ppg],

When calculating the d-exponent during drilling with the lab-scale rig, the uncorrected equation is utilized.

Interactions of natural features

Similar to the DOC and BA, the following features are created to add additional interactions of the drilling parameters WOB, RPM and ROP to use for classification.

RPM*WOB is referred to as TF1:

```

1 for i in range(length):
2     TF1 = (Actualdataframe['wob'][i] * Actualdataframe['rpm'][i])
3     Actualdataframe['TF1'][i] = TF1

```

Figure 6.4: Algorithm to calculate the RPM*WOB feature.

RPM*WOB/ROP is referred to as TF2:

```
1 for i in range(length):
2     TF2 = (Actualdataframe['wob'][i] * Actualdataframe['rpm'][i]) / Actualdataframe['ROP_instantaneous'][i]
3     Actualdataframe['TF2'][i] = TF2
```

Figure 6.5: Algorithm to calculate the RPM*WOB/ROP feature.

RPM² is referred to as TF3:

```
1 for i in range(length):
2     TF3 = (Actualdataframe['rpm'][i] * Actualdataframe['rpm'][i])
3     Actualdataframe['TF3'][i] = TF3
```

Figure 6.6: Algorithm to calculate the RPM² feature.

WOB² is referred to as TF4:

```
1 for i in range(length):
2     TF4 = (Actualdataframe['wob'][i] * Actualdataframe['wob'][i])
3     Actualdataframe['TF4'][i] = TF4
```

Figure 6.7: Algorithm to calculate the WOB² feature.

WOB/RPM is referred to as TF5:

```
1 for i in range(length):
2     TF5 = (Actualdataframe['wob'][i] / Actualdataframe['rpm'][i])
3     Actualdataframe['TF5'][i] = TF5
```

Figure 6.8: Algorithm to calculate the WOB/RPM feature.

The basis for calculating the interactions of natural features was a data analysis experiment conducted using MATLAB to investigate whether the feature importance of these natural interactions were higher than just using the natural features. The resulting feature importance is shown in section 6.2.1 where an ExtraTreesClassifier algorithm is run to evaluate the feature importance of all features.

Statistical features for drilling incident classification

Several statistical features have also been created for the cases of rock classification, vibration classification, pressure classification, stuck pipe and twist off classification. These describe the average value, standard deviation, median, maximum, minimum, P25, P50 and P75 value for each of the natural features pressure, weight on bit, or torque. Please note that these features however, need to get calculated before conducting the data concatenation, as each feature should be distinct for the class that they represent only. In the following paragraphs, examples are given for how these statistical features have been calculated to classify

either a leak (losses), normal operating pressure or overpressure (plugged nozzles) in Python. Each feature is calculated over an interval of 960 samples; i.e. 10 calculations every second at a sampling rate of 9600 Hz.

Standard deviation is calculated running the algorithm shown:

```

1 a1 = math.ceil(len(Testing['Label'])/960)
2 rest1 = len(Testing['Label'])
3
4 for i in range(a1):
5     remaining1 = min(rest1, 959)
6     datastd = Testing['WBM_Pressure'][i*960 : i*960 + remaining1].std()
7
8     for j in range(remaining1 + 1):
9         Testing['Pressure_std'][i*960 + j] = float(datastd)
10
11     rest1 -= 960

```

Figure 6.9: Algorithm to calculate standard deviation for pressure measurements to classify different pressure cases.

Mean value is calculated running the algorithm shown:

```

1 a2 = math.ceil(len(Testing['Label'])/960)
2 rest2 = len(Testing['Label'])
3
4 for i in range(a2):
5     remaining2 = min(rest2, 959)
6     datamean = Testing['WBM_Pressure'][i*960 : i*960 + remaining2].mean()
7
8     for j in range(remaining2 + 1):
9         Testing['Pressure_avg'][i*960 + j] = float(datamean)
10
11     rest2 -= 960

```

Figure 6.10: Algorithm to calculate mean value for pressure measurements to classify different pressure cases.

Median value is calculated running the algorithm shown:

```

1 a3 = math.ceil(len(Testing['Label'])/960)
2 rest3 = len(Testing['Label'])
3
4 for i in range(a3):
5     remaining3 = min(rest3, 959)
6     datamedian = Testing['WBM_Pressure'][i*960 : i*960 + remaining3].median()
7
8     for j in range(remaining3 + 1):
9         Testing['Pressure_med'][i*960 + j] = float(datamedian)
10
11     rest3 -= 960

```

Figure 6.11: Algorithm to calculate median value for pressure measurements to classify different pressure cases.

Peak maximum value is calculated running the algorithm shown:

```

1 a5 = math.ceil(len(Testing['Label'])/960)
2 rest5 = len(Testing['Label'])
3
4 for i in range(a5):
5     remaining5 = min(rest5, 959)
6     datamax = Testing['WBM_Pressure'][i*960 : i*960 + remaining5].max()
7
8     for j in range(remaining5 + 1):
9         Testing['Pressure_min'][i*960 + j] = float(datamax)
10
11     rest5 -= 960

```

Figure 6.12: Algorithm to calculate maximum value for pressure measurements to classify different pressure cases.

Peak minimum value is calculated running the algorithm shown:

```

1 a5 = math.ceil(len(Testing['Label'])/960)
2 rest5 = len(Testing['Label'])
3
4 for i in range(a5):
5     remaining5 = min(rest5, 959)
6     datamax = Testing['WBM_Pressure'][i*960 : i*960 + remaining5].max()
7
8     for j in range(remaining5 + 1):
9         Testing['Pressure_min'][i*960 + j] = float(datamax)
10
11 rest5 -= 960

```

Figure 6.13: Algorithm to calculate the minimum value for pressure measurements to classify different pressure cases.

P25, P30, P50, P60 and P75 can all be calculated using the Pandas library for data analysis in Python: `Dataframe.quantile(q=0.25/0.30/0.50/0.60/0.75)` function.

Interactions of statistical features for drilling incident classification

Based on the statistical features, the following interactions have been calculated for the mentioned intervals for torque, WOB and pressure (where applicable).

average value / standard deviation

```

1 Testing['avg_std'] = Testing['Pressure_avg'] / Testing['Pressure_std']

```

Figure 6.14: Algorithm to calculate the average divided by the standard deviation within an interval to classify different pressure cases.

median value / standard deviation

```

1 Testing['med_std'] = Testing['Pressure_med'] / Testing['Pressure_std']

```

Figure 6.15: Algorithm to calculate the median divided by the standard deviation within an interval to classify different pressure cases.

Peak to Peak

```

1 Testing['max-min'] = Testing['Pressure_max'] - Testing['Pressure_min']

```

Figure 6.16: Algorithm to calculate the difference between the peak maximum value and the peak minimum value within an interval to classify different pressure cases.

In the training data, the ROP, mean depth and statistical features are computed for each individual dataset before the datasets get concatenated and pre-processed.

6.2 Feature Selection

Before each feature were created, the original dataset was downsampled to 1 % of the original dataset; i.e. a reduction from 9600 Hz to 96 Hz. This has been necessary for two reasons:

- reduce the computational power and time required to develop the models,
- facilitate for classification in real-time on the control system that is only capable of a sampling frequency of up to 100 Hz, even with a 12-core CPU that allows us to execute 24 parallel processes (threads).

6.2.1 With outliers kept in the dataset

Before we remove outliers by the IQR method, we want to evaluate which features that are most important using the `ExtraTreesClassifier` and PCA approaches. This is necessary to see whether some features lose their importance when outliers have been removed.

ExtraTreesClassifier to evaluate feature importance

A tree algorithm has been used to evaluate the feature importance in the datasets. The algorithm is an `ExtraTreesClassification` which is part of the Scikit Learn library [59], and the output of the algorithm is the score rank, feature number and feature score (percentage). Due to this being an `ExtraTreesClassifier`, there is no guarantee that these features are the most optimal for all algorithms (such as neural networks, support vector machine, and so on). The algorithm does however give a quick evaluation of the features, and can be very useful to evaluate especially whether the engineered features are considered important, or whether additional features should get engineered.

From running the `ExtraTreesClassifier` algorithm, the feature importance is shown in Figure 6.17:

```

Feature ranking:
1. feature 9 (0.188506)
2. feature 10 (0.151852)
3. feature 5 (0.121461)
4. feature 13 (0.096320)
5. feature 3 (0.094159)
6. feature 17 (0.089876)
7. feature 8 (0.056198)
8. feature 4 (0.034060)
9. feature 18 (0.022724)
10. feature 7 (0.021249)
11. feature 15 (0.017485)
12. feature 12 (0.016630)
13. feature 11 (0.015851)
14. feature 6 (0.014375)
15. feature 14 (0.013123)
16. feature 19 (0.012327)
17. feature 16 (0.012047)
18. feature 1 (0.007994)
19. feature 2 (0.007327)
20. feature 0 (0.006438)

```

SCORE: (1) Mean depth
(2) ROP_instantaneous
(3) depth
(4) TF3
(5) rpm
(6) DOC - Depth of Cut
(7) WOB_setpoint
(8) m_torque
(9) BA - Bit aggressiveness
(10) RPM_setpoint
(11) TFS
(12) TF2
(13) TF1
(14) wob - Weight on bit
(15) TF4
(16) D-exponent
(17) MSE
(18) Load cell z2
(19) Load cell z3
(20) Load cell z1

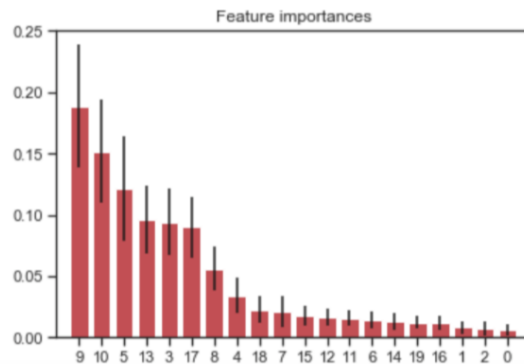


Figure 6.17: Results from running the *ExtraTreesClassifier* algorithm before removing outliers in the data. The rank, feature number, score (percentage) and feature name is given at the top.

By looking at the feature results, three out of the four highest scoring features heavily depend on the depth measurement. Since depth, mean depth, $WOB_{setpoint}$ and $RPM_{setpoint}$ in our opinion should not get used to develop the models (since these are not representative of the interaction with the formation), the six highest scoring features that will get used in chapter 7 are:

Feature:	ROP	TF3	rpm	DOC	Torque	BA
Rank:	1	2	3	4	5	6

PCA to reduce feature dimensionality

As explained in section 3.2.2, PCA restructures the dataset; reducing the amount of variables by creating new ones that are independent of each other. A principle component represents a linear relation, where the variables are all original input parameters. Figures 6.18, 6.19, 6.20, 6.21 and 6.22 have been created from the laboratory drilling rig data using the algorithm examples provided in (Otterbach, 2016) [60].

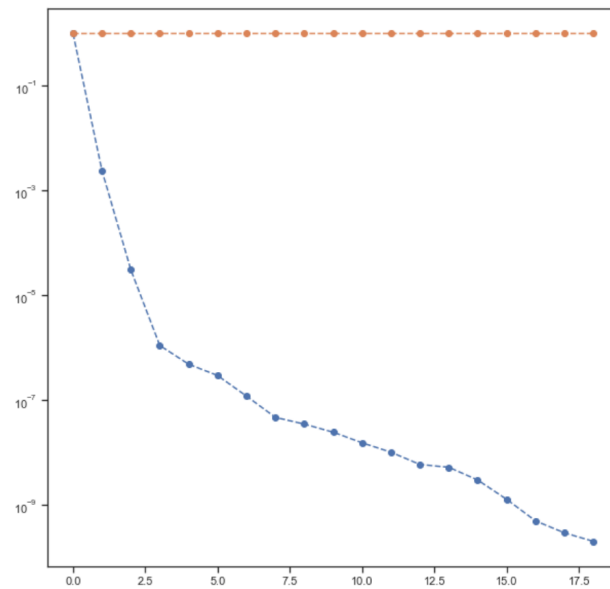


Figure 6.18: *Feature dimensionality reduction using PCA. Yellow line represents the cumulative explained variance ratio and the blue line represents the explained variance ratio.*

As can be observed in Figure 6.18, the principal component is shown on the x -axis while the explained variance is shown on the y -axis. By plotting the mean- and variance ratios, we can investigate how each of the original features have an effect on the variance:



Figure 6.19: *Mean (orange) and variance (blue) ratios for original features are shown from PCA analysis.*

Furthermore, we can then normalize the data using z-scaling, to prevent some components from dominating the PCA:

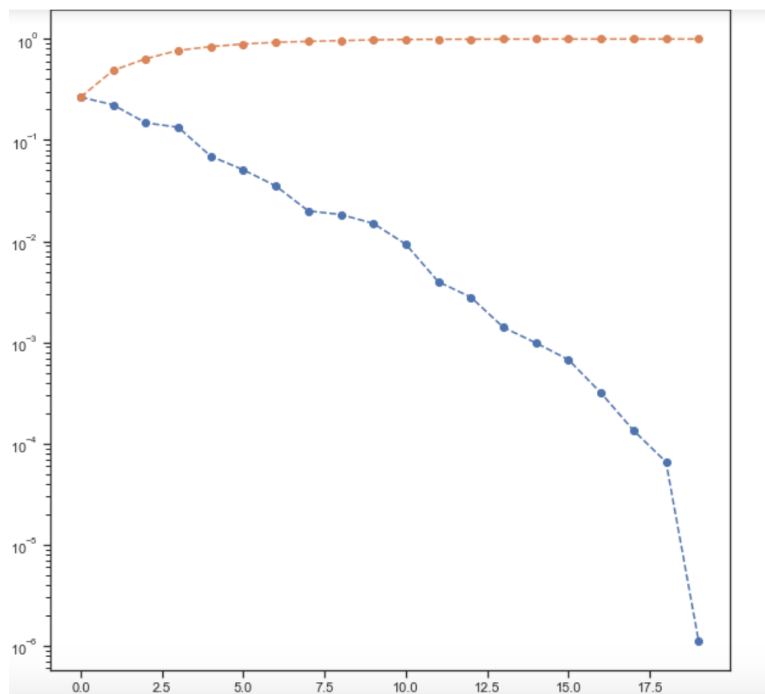


Figure 6.20: *Feature dimensionality reduction using PCA when z-scaled. Yellow line represents the cumulative explained variance ratio and the blue line represents the explained variance ratio.*

From Figure 6.20, we can see that when applying the z-scaling, the slope for variance is less steep, and fewer principal components appear to be necessary to explain the variance in the slope. From the observations, it appears that only five principal components are necessary to explain almost all of the variance in the dataset.

PCA to evaluate feature importance

PCA can also be used to evaluate the importance that each original feature has on the principal components:

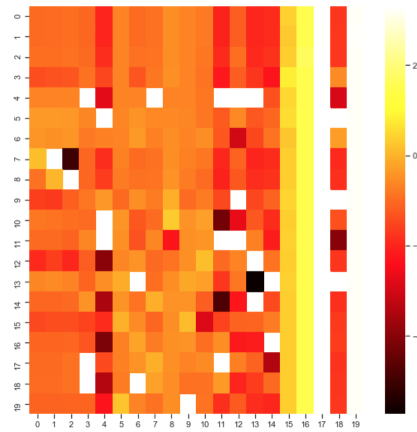


Figure 6.21: Heatmap displaying the feature importance to create the Principal Components that are created using PCA. *x-axis: original feature index. y-axis: principal component.*

With z-scaling applied, the feature importance is shown in Figure 6.22:

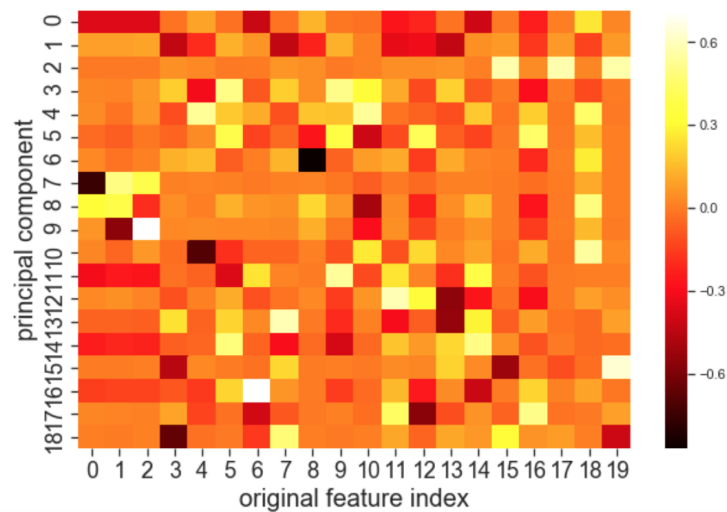


Figure 6.22: Heatmap displaying the feature importance to create the Principal Components first applying a z-scaling normalization and then conducting PCA. *x-axis: original feature index. y-axis: principal component.*

6.2.2 With no outliers in the dataset

In chapter 3 and chapter 5, the concept of using IQR method to remove outliers in the dataset is explained. The feature importance and PCA shown in the next paragraphs are performed when removing outliers in the natural features (measurements) that are outside of the minimum and maximum thresholds given in the table below. IQR has not been used to remove outliers from the engineered

features, since if an outlier is detected in the natural features; the entire data row will get dropped (including the artificially engineered features for those data).

Natural Feature	Minimum threshold	Maximum threshold
Z1	0.08355937499999999	0.654294375
Z2	0.09999812499999997	0.650713125
Z3	0.09999812499999997	0.650713125
rpm	-0.25634	0.94878
m_torque	-0.01944994179278227	0.2572392316647264
wob	0.14139416666666667	0.6255875

ExtraTreesClassifier to evaluate feature importance

```

Feature ranking:
1. feature 9 (0.187500)
2. feature 10 (0.144635)
3. feature 5 (0.127844)
4. feature 13 (0.100985)
5. feature 3 (0.097963)
6. feature 17 (0.088801)
7. feature 8 (0.057773)
8. feature 4 (0.032257)
9. feature 18 (0.024003)
10. feature 7 (0.021808)
11. feature 11 (0.017565)
12. feature 12 (0.016394)
13. feature 15 (0.015823)
14. feature 19 (0.013978)
15. feature 6 (0.011910)
16. feature 14 (0.011550)
17. feature 16 (0.010272)
18. feature 2 (0.007333)
19. feature 1 (0.006761)
20. feature 0 (0.004845)

```

```

Score:
(1) mean_depth
(2) ROP_instantaneous
(3) depth
(4) TF3
(5) rpm
(6) DOC - Depth of cut
(7) WOB_set
(8) m_torque
(9) BA - Bit aggressiveness
(10) RPM_set
(11) TF1
(12) TF2
(13) TF5
(14) D-exponent
(15) wob
(16) TF4
(17) MSE
(18) Load cell z3
(19) Load cell z2
(20) Load cell z1

```

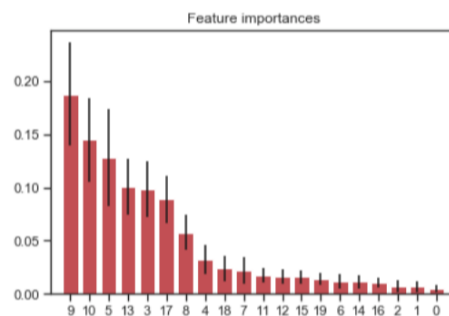


Figure 6.23: Results from *ExtraTreesClassifier* algorithm to evaluate the feature importance after *IQR* has been used to remove outliers.

As can be observed in Figure 6.23, outlier removal only results in minor changes to the feature importances, which possibly suggests that it might not be necessary to remove outliers to retain the feature importance in the dataset for rock classification. Except for the re-arranged order of TF1, TF2 and TF5, the only notable change is the D-exponent, which has climbed from rank 16 to rank 14,

importance of features.

6.3 Feature Extraction

The following workflow is integrated in the real-time system to extract the features that have provided the highest score in the feature importance evaluation.

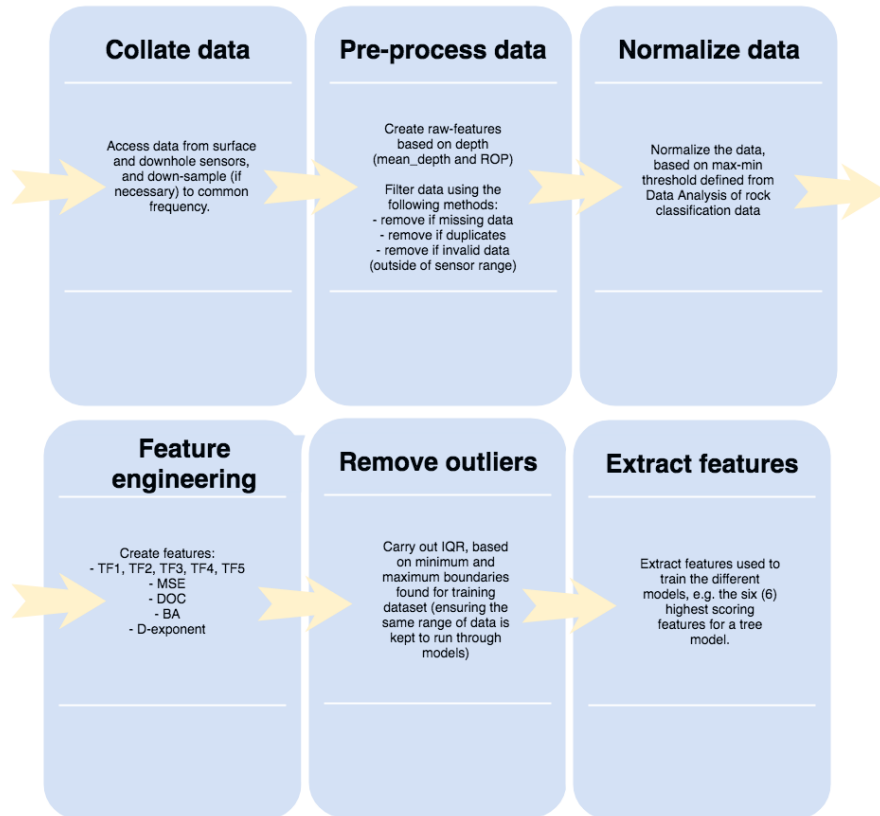


Figure 6.25: Flowchart of data flow and processes performed for real-time rock classification.

Even if the results in this chapter are taken from the rock classification workflow, the same flow chart is applied when developing the models for drilling incidents and vibration classification using downhole measurements. With the currently installed equipment, the data gets sampled at 60 Hz, limited by the sampling frequency of the microcontroller that gets used to transmit downhole measurements to surface. Raw-features such as $\text{mean}_{\text{depth}}$ and ROP get calculated both by the height sensor and step-counting (as mentioned in chapter 2), and the data gets normalized before additional features get engineered. The drilling features are then calculated, before outliers get removed. At last, the features that have been used to train the different models get extracted and are input into the model to

deliver a classification.

Even if methods such as the `ExtraTreesClassifier` can be used to evaluate the importance of features and PCA can be used in order to create new principal components that can explain the variance in the dataset (and thus be used to classify which class an observation belongs to), such scores are not guaranteed to be correct. Methods such as the `ExtraTreesClassifier` algorithm should in our opinion only get used to give an indication of the importance that an engineered feature holds. Particularly when working with drilling data, manual selection of features should be performed. Those features that are considered important to describe a particular phenomena (such as for instance bit-rock interaction for rock formation classification) should get selected rather than blindly trusting the score from an algorithm. A high accuracy score does not guarantee that the model can correctly classify the observations in a new dataset; if the selected features are not directly applicable. One example of this can be if a certain flowrate gets selected for one drilling operation, while another flowrate is used for another formation. While such differences in the training data will result in a high feature importance for flowrate, the setpoint can be independent of the formation type and thus can **not** be used to confirm that a particular formation is being drilled. The same applies to for instance RPM and WOB setpoints.

Chapter 7

Model Development - Sensitivity Study

7.1 Sensitivity study objective

In this chapter, a sensitivity study is conducted to evaluate which preprocessing techniques (missing or invalid data, and IQR) and features that result in the most optimal models for the drilling cases given below. In the pre-processing column, *MD* denotes missing (and invalid) data, *Norm* denotes data normalization and *IQR* denotes outlier removal. For all cases, the most optimal model parameters are found using the GridSearchCV function, as is presented in chapter 3. The input data, model parameters, and a classification report is presented for all supervised cases, where as for unsupervised cases the results are presented graphically. Since the most optimal features have only been presented for rock formation classification using laboratory measurements in chapter 6 each classification task will also be presented with the recommended feature priority based on an ExtraTreesClassifier algorithm. Regardless of the feature priority from the algorithm, manual selection is performed to ensure that only those features that are regarded as applicable gets used. In chapter 8.

- a. Laboratory rock formation classification - 5 cases
- b. Volve rock formation classification - 5 cases
- c. Laboratory rig operation classification - 3 cases
- d. Volve rig operation classification - 3 cases
- e. Laboratory Normal pressure, leak and overpressure classification - 3 cases
- f. Laboratory vibration classification (surface data) - 3 cases

- g. Laboratory vibration classification (downhole data) - 4 cases
- h. Laboratory stuck-pipe classification - 3 cases
- i. Laboratory twist off classification - 3 cases

In terms of classification reports, each case is presented with the fitted model's accuracy and Area Under Curve (AUC). The AUC denotes the separability of the data, i.e. to what extent the model is capable of distinguishing between a true positive and a false positive as illustrated in subsection 3.3.1:

- a true positive can be thought of as the algorithm detecting a class (formation type, or incident type) that is indeed observed, e.g. the algorithm detecting a sandstone when a sandstone indeed is being drilled,
- a false positive indicates when the model detects a class, even if the observation belongs to a different class, e.g. the algorithm detecting a sandstone while a granite rock is encountered.

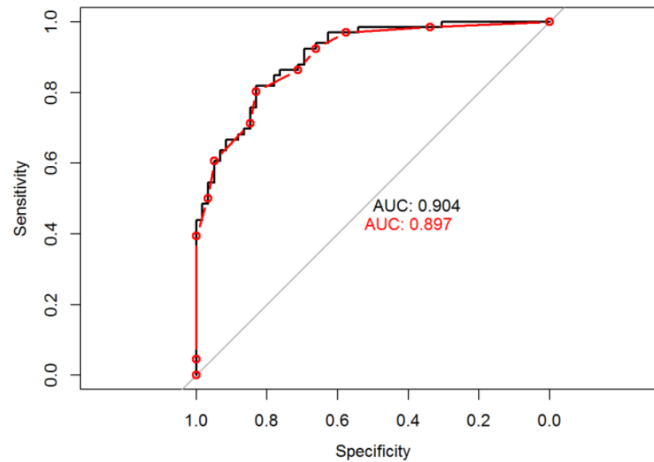


Figure 7.1: Area under curve concept [61].

As can be observed in the figure above, the AUC returns a score between 0 and 1, where the higher the AUC score, the better the model performance is.

For rock classification, the combination of accuracy and AUC is used to evaluate which combination of features and data preparation methods produce the best model. For rig operations and drilling incidents (and vibration levels), unclassified learning has been chosen to organize and evaluate the data. Unclassified learning can not be used to classify or predict different rock formations, rig operations

or drilling incidents. They do however give valuable information in terms of organizing the data and evaluating whether features can be used to separate the different classes from the observations. For classification tasks *c. - i.*, each task is first presented with a case using supervised K-Nearest Neighbors to develop a model that can be used on the drilling rig. Then, unsupervised learning is used to generate clusters; first considering two natural features and then two engineered features. For each unsupervised learning case, an Adjusted Rand Index (ARI) score is given. The ARI denotes a similarity measure between clusters (Rand Index) which is also adjusted for chance. The output range from the adjusted rand index is -1.0 to 1.0, where 1.0 suggests that the clusters are identical.

The different classifiers that have been used to develop the models in the cases below have all been taken from the Scikit Learn library [59]. These are: MLPClassifier, DecisionTreeClassifier, SVC (C-Support Vector Classifier), RandomForestClassifier, GradientBoostingClassifier, KNeighborsClassifier, KMeans, DBSCAN and TPOTClassifier.

Case studies				
Case	Data	Pre-processing	Model(s)	Feature(s)
1.	Laboratory rock data: 6 formations	MD, Norm	Supervised: MLP, DT, SVM, GB, RF, K-NN, TPOT	16 features
2.	Laboratory rock data: 6 formations	MD, Norm, IQR	Supervised: MLP, DT, SVM, GB, RF, K-NN, TPOT	16 features
3.	Laboratory rock data: 6 formations	MD, Norm, IQR	Supervised: MLP, DT, SVM, GB, RF, K-NN, TPOT	6 highest features
4.	Laboratory rock: 3 formations	MD, Norm, IQR	Supervised: MLP, DT, SVM, GB, RF, K-NN, TPOT	16 features
5.	Laboratory rock: 3 formations	MD, Norm, IQR	Supervised: MLP, DT, SVM, GB, RF, K-NN, TPOT	6 highest features
6.	Volve field data	MD, Norm	Supervised: MLP, DT, SVM, GB, RF, K-NN, TPOT	12 features
7.	Volve field data	MD, Norm, IQR	Supervised: MLP, DT, SVM, GB, RF, K-NN, TPOT	12 features

8.	Volve field data	MD, Norm, IQR	Supervised: MLP, DT, SVM, GB, RF, K-NN, TPOT	6 highest features
9.	Volve field data: 3 formations	MD, Norm, IQR	Supervised: MLP, DT, SVM, GB, RF, K-NN, TPOT	12 features
10.	Volve field data: 3 formations	MD, Norm, IQR	Supervised: MLP, DT, SVM, GB, RF, K-NN, TPOT	6 highest features
11.	Laboratory rig operations	MD, Norm, IQR	Supervised: K-NN	21 features
12.	Laboratory rig operations	MD, Norm, IQR	Unsupervised: K means, DBSCAN	2 natural features
13.	Laboratory rig operations	MD, Norm, IQR	Unsupervised: K means, DBSCAN	2 engineered features
14.	Volve rig operations	MD Norm	Supervised: K-NN	15 features
15.	Volve rig operations	MD, Norm	Unsupervised: K means, DBSCAN	2 natural features
16.	Volve rig operations	MD, Norm	Unsupervised: K means, DBSCAN	2 engineered features
17.	Laboratory pressure case	MD, Norm	Supervised: K-NN	10 features
18.	Laboratory pressure case	MD, Norm	Unsupervised: K means, DBSCAN	2 natural features
19.	Laboratory pressure case	MD, Norm	Unsupervised: K means, DBSCAN	2 engineered features
20.	Laboratory vibration case (surface data)	MD, Norm	Supervised: K-NN	13 features
21.	Laboratory vibration case (surface data)	MD, Norm	Unsupervised: K means, DBSCAN	2 natural features
22.	Laboratory vibration case (surface data)	MD, Norm	Unsupervised: K means, DBSCAN	2 engineered features
23.	Laboratory vibration case (downhole data)	Norm	Supervised: K-NN	34 features
24.	Laboratory vibration case (downhole data)	Norm	Supervised: K-NN	6 highest features
25.	Laboratory vibration case (downhole data)	Norm	Unsupervised: K means, DBSCAN	2 natural features
26.	Laboratory vibration case (downhole data)	Norm	Unsupervised: K means, DBSCAN	2 engineered features
27.	Laboratory stuck-pipe case	MD, Norm	Supervised: K-NN	14 features
28.	Laboratory stuck-pipe case	MD, Norm	Unsupervised: K means, DBSCAN	2 natural features

29.	Laboratory stuck-pipe case	MD, Norm	Unsupervised: K means, DBSCAN	2 engineered features
30.	Laboratory twist off case	MD, Norm	Supervised: K-NN	14 features
31.	Laboratory twist off case	MD, Norm	Unsupervised: K means, DBSCAN	2 natural features
32.	Laboratory twist off case	MD, Norm	Unsupervised: K means, DBSCAN	2 engineered features
33.*	Volve field data	MD, Norm, IQR	Supervised: MLP, DT, SVM, GB, RF, K-NN	14 features, incl. flow rate and mud weight into well

* denotes a theoretical test of which results are obtained also using two features for flow rate and mud weight going into the well to classify the formations at the Volve field, as is described in subsection 7.2.2, Volve formation classification. The results from Case 33 is only presented in the summary table at the end of the chapter. For a summary of the results, please see section 7.3.

7.2 Study cases

In each subsection, the GridSearchCV algorithm gets used to identify the most optimal model parameters to develop the models with, except for the unsupervised models. Then the different cases (as shown in the beginning of chapter 7) are developed using the identified model parameters.

7.2.1 Laboratory formation classification

The processed dataset consists of 506147 rows \times 16 feature columns (+ 1 label column), containing a total of six rock types, as mentioned in chapter 6. The dataset has been sampled at 9600 Hz, and downsampled to 96 Hz. The data has further been processed for missing or invalid data, normalized and then outliers have been removed. The features available are: load cells (1/2/3), rpm feedback, motor torque feedback, weight on bit, rate of penetration, TF1, TF2, TF3, TF4, TF5, mechanical specific energy, depth of cut, bit aggressiveness and d-exponent (see chapter 6). Operating setpoints such as WOB setpoint and RPM setpoint have been removed, along with depth and mean depth measurements, since these are either controlled by the driller (or the machine), and do not describe the formation (bit interaction). For case 1 through 5, the following labels are used for

naming the different formations: 1: Cement, 2: Chalk, 3: Granite, 4: Sandstone, 5: Salt and 6: Shale.

Model parameter tuning

a. Multi Layer Perceptron (MLP): optimal parameter search is performed evaluating the following parameters on 10 % of the dataset and all 16 features.

```
1 params_to_test = {'hidden_layer_sizes': [(10,10,10), (50,50,50), ...
      (100,100,100)], 'activation': ['tanh', 'relu'], 'solver': ['sgd', ...
      'adam'], 'alpha': [0.0001, 0.1], 'learning_rate': ['adaptive', ...
      'invscaling'],}
```

The random state, grid-search cross-validation generator (cv), and scoring values are:

```
1 MLP_model = MLPClassifier(random_state=None)
2 grid_search = GridSearchCV(MLP_model, param_grid=params_to_test, cv=3, ...
      scoring='f1_macro', n_jobs=4)
```

The best model parameters are:

```
1 MLPClassifier(activation='tanh', alpha=0.0001, batch_size='auto', ...
      beta_1=0.9, beta_2=0.999, early_stopping=False, epsilon=1e-08, ...
      hidden_layer_sizes=(100, 100, 100), learning_rate='invscaling', ...
      learning_rate_init=0.001, max_iter=200, momentum=0.9, ...
      nesterovs_momentum=True, power_t=0.5, random_state=None, ...
      shuffle=True, solver='adam', tol=0.0001, validation_fraction=0.1, ...
      verbose=False, warm_start=False)
```

b. Decision Tree (DT): optimal parameter search is performed evaluating the following parameters on 100 % of the dataset and all 16 features.

```
1 params_to_test = {'criterion':['gini','entropy'],'max_depth': ...
      [4,5,6,7,8,9,10,11,12,15,20,30,40,50,70,90,120,150]}
```

The random state, grid-search cross-validation generator (cv), and scoring is shown below:

```
1 DT_model = DecisionTreeClassifier(random_state=None)
2 grid_search = GridSearchCV(DT_model, param_grid=params_to_test, cv=3, ...
      scoring='f1_macro', n_jobs=4)
```

The best model parameters are:

```

1 DecisionTreeClassifier(class_weight=None, criterion='entropy', ...
    max_depth=70, max_features=None, max_leaf_nodes=None, ...
    min_impurity_decrease=0.0, min_impurity_split=None, ...
    min_samples_leaf=1, min_samples_split=2, ...
    min_weight_fraction_leaf=0.0, presort=False, random_state=None, ...
    splitter='best')

```

c. Support Vector Machine (SVM): optimal parameter search is performed evaluating the following parameters on 5 % of the dataset and all 16 features.

```

1 params_to_test = {'kernel': ['rbf', 'linear'], 'gamma': [1e-3, 1e-4], ...
    'C': [1, 10, 100, 1000]}

```

The random state, grid-search cross-validation generator (*cv*), and scoring is shown below:

```

1 SV_model = SVC(random_state=None)
2 grid_search = GridSearchCV(SV_model, param_grid=params_to_test, cv=3, ...
    scoring='f1_macro', n_jobs=4)

```

The best model parameters are:

```

1 SVC(C=1000, cache_size=200, class_weight=None, coef0=0.0, ...
    decision_function_shape='ovr', degree=3, gamma=0.001, kernel='rbf', ...
    max_iter=-1, probability=False, random_state=None, shrinking=True, ...
    tol=0.001, verbose=False)

```

d. Random Forest (RF): optimal parameter search is performed evaluating the following parameters on 100 % of the dataset and all 16 features.

```

1 params_to_test = {'n_estimators':[1,50,10], 'max_depth':[5,10,1]}

```

The random state, grid-search cross-validation generator (*cv*), and scoring is shown below:

```

1 RF_model = RandomForestClassifier(random_state=42)
2 grid_search = GridSearchCV(RF_model, param_grid=params_to_test, cv=10, ...
    scoring='f1_macro', n_jobs=4)

```

The best model parameters are:

```

1 RandomForestClassifier(bootstrap=True, class_weight=None, ...
    criterion='gini', max_depth=10, max_features='auto', ...
    max_leaf_nodes=None, min_impurity_decrease=0.0, ...

```

```
min_impurity_split=None, min_samples_leaf=1, min_samples_split=2, ...
min_weight_fraction_leaf=0.0, n_estimators=50, n_jobs=1, ...
oob_score=False, random_state=42, verbose=0, warm_start=False)
```

e. Gradient Boosting (GB): optimal parameter search is performed evaluating the following parameters on 100 % of the dataset and all 16 features.

```
1 params_to_test = {'learning_rate': [0.1, 0.05, 0.02, 0.01], 'max_depth': ...
                   [4, 6, 8], 'min_samples_leaf': [20, 50, 100, 150]}
2 # 'max_features': [1.0, 0.3, 0.1]}
```

The random state, grid-search cross-validation generator (cv), and scoring is shown below:

```
1 GB_model = GradientBoostingClassifier(random_state=None)
2
3 grid_search = GridSearchCV(GB_model, param_grid=params_to_test, cv=3, ...
                             scoring='f1_macro', n_jobs=4)
```

The best model parameters are:

```
1 GradientBoostingClassifier(criterion='friedman_mse', init=None, ...
                             learning_rate=0.1, loss='deviance', max_depth=8, max_features=None, ...
                             max_leaf_nodes=None, min_impurity_decrease=0.0, ...
                             min_impurity_split=None, min_samples_leaf=20, min_samples_split=2, ...
                             min_weight_fraction_leaf=0.0, n_estimators=100, presort='auto', ...
                             random_state=None, subsample=1.0, verbose=0, warm_start=False)
```

f. K-Nearest Neighbor (K-NN): optimal parameter search is performed evaluating the following parameters on 100 % of the dataset and all 16 features.

```
1 params_to_test = {'n_neighbors': [3, 5, 11], 'weights': ['distance', 'uniform']}
```

The random state, grid-search cross-validation generator (cv), and scoring is shown below:

```
1 KNN_model = KNeighborsClassifier()
2 grid_search = GridSearchCV(KNN_model, param_grid=params_to_test, cv=3, ...
                             scoring='f1_macro', n_jobs=4)
```

The best model parameters are:

```
1 KNeighborsClassifier(algorithm='auto', leaf_size=30, metric='minkowski', ...
                       metric_params=None, n_jobs=1, n_neighbors=5, p=2, weights='distance')
```

g. TPOT Algorithm: optimal pipeline search is performed using the following parameters on 1 % of the dataset and all 16 features.

```
1 tpot = TPOTClassifier(generations=10, population_size=50, verbosity=2)
```

With regards to the best pipeline for TPOT, this changes for every case. Due to this, the recommended pipeline from the software is presented at the end of each case (for cases 1 - 10).

Case 1

<i>Model variable input(s):</i>	z1, z2, z3, RPM, m_torque, WOB, ROP_instantaneous, TF1, TF2, TF3, TF4, TF5, MSE, DOC, BA, D-exponent
<i>Sampling rate:</i>	96 Hz
<i>Labels:</i>	1: Cement, 2: Chalk, 3: Granite, 4: Sandstone, 5: Salt, 6: Shale

a. MLP - Accuracy: 94.31 %, Area under curve: 0.1413

```

                precision    recall  f1-score   support

   1.0         1.00         0.92         0.96         1598
   2.0         0.97         1.00         0.99          366
   3.0         0.96         0.96         0.96         2850
   4.0         0.95         0.87         0.90         2005
   5.0         0.98         0.96         0.97         1439
   6.0         0.88         0.98         0.92         2729

 avg / total         0.95         0.94         0.94        10987
0.9431145899699646
confusion matrix [[1474    0   86    6   12   20]
 [   0  365    0    0    1    0]
 [   4    0 2737    4    1  104]
 [   0    5    6 1736    9  249]
 [   3    5    0  44 1386    1]
 [   0    0   22  43    0 2664]]
auc 0.14129883934446807

```

Figure 7.2: Classification report for MLP.

b. DT - Accuracy: 99.67 %, Area under curve: 0.2444

```

                precision    recall  f1-score   support

   0         1.00         1.00         1.00        15909
   1         1.00         1.00         1.00        3671
   2         1.00         1.00         1.00        29104
   3         1.00         1.00         1.00        19792
   4         1.00         1.00         1.00        14130
   5         1.00         1.00         1.00        27257

 avg / total         1.00         1.00         1.00        109863
0.9967413961024185
confusion matrix [[15865    0   11   19    9    5]
 [   1  3668    0    0    2    0]
 [  11    0 29010    3    0   80]
 [  23    0    1 19717   21   30]
 [   1    0    0   16 14112    1]
 [   4    0   79   39    2 27133]]
auc 0.244367413829721

```

Figure 7.3: Classification report for DT.

For decision tree, the class labels have been changed to the order of 0 to 5. 0: Cement, 1: Chalk, 2: Granite, 3: Sandstone, 4: Salt and 5: Shale. In chapter 8, this has been accounted for.

c. SVM - Accuracy: 89.88 %. Area under curve: 0.1524

```

                precision    recall  f1-score   support

     1.0         0.98         0.97         0.98         1585
     2.0         0.96         0.99         0.98         375
     3.0         0.82         0.89         0.85         2909
     4.0         0.95         0.95         0.95         2058
     5.0         0.98         0.99         0.98         1397
     6.0         0.85         0.77         0.81         2663

 avg / total         0.90         0.90         0.90         10987
0.8987894784745608
confusion matrix [[1545  11  7  10  12  0]
 [ 1 372  0  0  2  0]
 [ 9  0 2585  3  0 312]
 [20  2  17 1948 19  52]
 [ 3  3  0  9 1382  0]
 [ 3  0 532  84  1 2043]]
auc 0.1524116094986807

```

Figure 7.4: Classification report for SVM.

d. RF - Accuracy: 98.52 %. Area under curve: 0.1537

```

                precision    recall  f1-score   support

     1.0         0.99         0.99         0.99         1604
     2.0         1.00         0.99         1.00         376
     3.0         1.00         0.99         1.00         2897
     4.0         0.96         0.98         0.97         1948
     5.0         0.98         1.00         0.99         1397
     6.0         0.99         0.97         0.98         2765

 avg / total         0.99         0.99         0.99         10987
0.9851642850641668
confusion matrix [[1589  0  2  4  8  1]
 [ 0 374  0  0  2  0]
 [ 9  0 2875  2  0 11]
 [ 3  1  0 1902 22 20]
 [ 0  0  0  6 1391  0]
 [ 0  0  3  69  0 2693]]
auc 0.15372245181134792

```

Figure 7.5: Classification report for RF.

e. GB - Accuracy: 99.23 %. Area under curve: 0.1488

```

                precision    recall  f1-score   support

     1.0         1.00         1.00         1.00         1580
     2.0         1.00         1.00         1.00         367
     3.0         1.00         0.99         1.00         2921
     4.0         0.99         0.99         0.99         1995
     5.0         0.99         1.00         0.99         1390
     6.0         0.99         0.99         0.99         2734

 avg / total         0.99         0.99         0.99         10987
0.9922635842359152
confusion matrix [[1573  0  1  1  4  1]
 [ 0 367  0  0  0  0]
 [ 3  0 2900  2  0 16]
 [ 4  1  0 1973  8  9]
 [ 0  0  0  2 1388  0]
 [ 0  0  7  25  1 2701]]
auc 0.14882297551789075

```

Figure 7.6: Classification report for GB.

f. K-NN - Accuracy: 85.62 %. Area under curve: 0.1866

```

                precision    recall  f1-score   support

         1.0         0.87         0.92         0.89         1575
         2.0         0.95         0.94         0.95          361
         3.0         0.86         0.92         0.89         2935
         4.0         0.77         0.67         0.72         2012
         5.0         0.89         0.83         0.86         1374
         6.0         0.87         0.89         0.88         2730

 avg / total         0.85         0.86         0.85        10987
0.8561936834440703
confusion matrix [[1447    0   16   15   95    2]
 [   0  341    0    0   20    0]
 [   8    0 2713  137    2   75]
 [  35    0   328 1344   18  287]
 [ 174   19    21   20 1137    3]
 [   3    0    69  231    2 2425]]
auc 0.1866082149413475
    
```

Figure 7.7: Classification report for K-NN.

g. TPOT - GradientBoostingClassifier - Accuracy: 98.09 %. Area under curve: 0.1519

```

                precision    recall  f1-score   support

         1.0         0.98         1.00         0.99          157
         2.0         1.00         1.00         1.00           46
         3.0         0.99         0.99         0.99          290
         4.0         0.97         0.95         0.96          197
         5.0         0.97         0.99         0.98          151
         6.0         0.98         0.97         0.97          258

 avg / total         0.98         0.98         0.98        1099
0.9808917197452229
confusion matrix [[157    0    0    0    0    0]
 [   0   46    0    0    0    0]
 [   2    0  287    0    0    1]
 [   1    0    0  187    4    5]
 [   0    0    0    1  150    0]
 [   0    0    2    5    0  251]]
auc 0.15194681861348525
    
```

Figure 7.8: Classification report for TPOT using an GradientBoostingClassifier.

Best pipeline: GradientBoostingClassifier(ExtraTreesClassifier(input_matrix, bootstrap=False, criterion=entropy, max_features=0.7500000000000001, min_samples_leaf=1, min_samples_split=3, n_estimators=100), learning_rate=0.1, max_depth=6, max_features=0.3, min_samples_leaf=10, min_samples_split=5, n_estimators=100, subsample=0.5)
 Accuracy is 98.08917197452229%

Figure 7.9: Recommended pipeline for Case 1 is a GradientBoostingClassifier.

Case 2

Model variable input(s):	z1, z2, z3, RPM, m_torque, WOB, ROP_instantaneous, TF1, TF2, TF3, TF4, TF5, MSE, DOC, BA, D-exponent
Sampling rate:	96 Hz
Labels:	1: Cement, 2: Chalk, 3: Granite, 4: Sandstone, 5: Salt, 6: Shale

a. MLP - Accuracy: 95.53 %, Area under curve: 0.1338

```

                precision    recall  f1-score   support

     1.0         0.99         0.98         0.99         12824
     2.0         0.99         0.99         0.99          3600
     3.0         0.98         1.00         0.99         28800
     4.0         0.97         0.80         0.88         17506
     5.0         0.98         0.99         0.99         12561
     6.0         0.89         0.98         0.93         25939

 avg / total         0.96         0.96         0.95         101230
0.9553195692976391
confusion matrix [[12625    0  149    3    43    4]
 [    0  3575    0    1   24    0]
 [   71    0 28664    3    0   62]
 [    4    8   199 14051   145 3099]
 [    2   11    1    75 12472    0]
 [    4    0   222  389    4 25320]]
auc 0.1337665022135728

```

Figure 7.10: Classification report for MLP.

b. DT - Accuracy: 99.66 %, Area under curve: 0.2265

```

                precision    recall  f1-score   support

     0         1.00         1.00         1.00         12631
     1         1.00         1.00         1.00          3561
     2         1.00         1.00         1.00         28949
     3         1.00         1.00         1.00         17606
     4         1.00         1.00         1.00         12701
     5         0.99         1.00         0.99         25782

 avg / total         1.00         1.00         1.00         101230
0.9966116763805196
confusion matrix [[12603    0   12   13    2    1]
 [    0  3561    0    0    0    0]
 [    5    0 28835    7    0  102]
 [   11    0    1 17539   15   40]
 [    0    2    0    7 12692    0]
 [    1    0    74   49    1 25657]]
auc 0.22653500300033308

```

Figure 7.11: Classification report for DT.

For decision tree, the class labels have been changed to the order of 0 to 5. 0: Cement, 1: Chalk, 2: Granite, 3: Sandstone, 4: Salt and 5: Shale. In chapter 8, this has been accounted for.

c. SVM - Accuracy: 89.66 %. Area under curve: 0.1344

```

                precision    recall  f1-score   support

     1.0         0.98         0.98         0.98         1263
     2.0         0.98         0.99         0.99          368
     3.0         0.81         0.88         0.84         2878
     4.0         0.95         0.94         0.94         1698
     5.0         0.98         0.99         0.98         1246
     6.0         0.84         0.76         0.80         2670

 avg / total         0.89         0.89         0.89         10123
0.888570581843327
confusion matrix [[1243    0    5    8    7    0]
 [    0  365    0    0    3    0]
 [   13    0 2532    2    2  329]
 [    2    5   14 1600   16   61]
 [    0    2    0    8 1236    0]
 [    5    0  572   74    0 2019]]
auc 0.1343845408151895

```

Figure 7.12: Classification report for SVM.

d. RF - Accuracy: 98.50 %. Area under curve: 0.1245

```

                precision    recall  f1-score   support

     1.0         1.00         0.99         0.99         1218
     2.0         1.00         1.00         1.00          394
     3.0         1.00         0.99         0.99         2971
     4.0         0.96         0.98         0.97         1739
     5.0         0.98         0.99         0.99         1241
     6.0         0.98         0.97         0.98         2560

 avg / total         0.99         0.98         0.99         10123
0.984984688333498
confusion matrix [[1206  0  1  3  7  1]
 [ 0 394  0  0  0  0]
 [ 4  0 2947  1  0 19]
 [ 1  1  0 1702 12 23]
 [ 0  0  0  7 1234  0]
 [ 0  0  8  64  0 2488]]
auc 0.12452461712406204

```

Figure 7.13: Classification report for RF.

e. GB - Accuracy: 99.26 %. Area under curve: 0.1337

```

                precision    recall  f1-score   support

     1.0         1.00         0.99         1.00         1307
     2.0         0.99         1.00         1.00          394
     3.0         1.00         1.00         1.00         2902
     4.0         0.98         0.99         0.99         1747
     5.0         0.99         1.00         1.00         1237
     6.0         0.99         0.99         0.99         2536

 avg / total         0.99         0.99         0.99         10123
0.9925911291119234
confusion matrix [[1297  0  2  4  3  1]
 [ 0 394  0  0  0  0]
 [ 3  0 2893  1  0  5]
 [ 0  0  0 1727  5 15]
 [ 0  2  0  1 1234  0]
 [ 0  0  7  26  0 2503]]
auc 0.1337239181827526

```

Figure 7.14: Classification report for GB.

f. K-NN - Accuracy: 85.37 %. Area under curve: 0.1527

```

                precision    recall  f1-score   support

     1.0         0.88         0.89         0.88         1264
     2.0         0.94         0.96         0.95          340
     3.0         0.85         0.91         0.88         2883
     4.0         0.75         0.65         0.70         1738
     5.0         0.88         0.87         0.88         1284
     6.0         0.89         0.88         0.88         2614

 avg / total         0.85         0.85         0.85         10123
0.8536994961967797
confusion matrix [[1120  0 16 12 116  0]
 [ 0 327  0  0 12  1]
 [ 7  0 2636 159  2  79]
 [ 27  0 337 1134 21 219]
 [118 16 23 10 1117  0]
 [ 4  5  98 199  0 2308]]
auc 0.15265180896032135

```

Figure 7.15: Classification report for K-NN.

g. TPOT - ExtraTreesClassifier - Accuracy: 98.29 %. Area under curve: 0.1395

```

                precision    recall  f1-score   support

     1.0         1.00         0.99         0.99         1310
     2.0         0.98         0.99         0.98          348
     3.0         0.99         0.99         0.99         2937
     4.0         0.97         0.96         0.97         1728
     5.0         0.96         0.99         0.98         1240
     6.0         0.98         0.98         0.98         2560

 avg / total         0.98         0.98         0.98         10123
0.9829102044848365
confusion matrix [[1294  0  4  3  9  0]
 [  0 344  0  0  4  0]
 [  4  0 2913  0  4 16]
 [  1  4  0 1664 28 31]
 [  0  4  0  3 1232  1]
 [  0  0 16  40  1 2503]]
auc 0.13951259664285506

```

Figure 7.16: Classification report for TPOT using an ExtraTreesClassifier.

```

Best pipeline: ExtraTreesClassifier(RFE(input_matrix, criterion=gini, max_features=0.3, n_estimators=100, step=0.3500
000000000003), bootstrap=False, criterion=entropy, max_features=0.7500000000000001, min_samples_leaf=3, min_samples_
split=3, n_estimators=100)
Accuracy is 98.5192497532083%

```

Figure 7.17: Recommended pipeline for Case 2 is an ExtraTreesClassifier

Case 3

Model variable input(s):	RPM, m_torque, ROP_instantaneous, TF3, DOC, BA
Sampling rate:	96 Hz
Labels:	1: Cement, 2: Chalk, 3: Granite, 4: Sandstone, 5: Salt, 6: Shale

a. MLP - Accuracy: 93.10 %, Area under curve: 0.1342

```

                precision    recall  f1-score   support

     1.0         0.99         0.96         0.97         12824
     2.0         1.00         0.98         0.99         3600
     3.0         0.97         0.99         0.98         28800
     4.0         0.96         0.69         0.80         17506
     5.0         0.99         0.98         0.98         12561
     6.0         0.83         0.98         0.90         25939

 avg / total         0.94         0.93         0.93         101230
0.9310481082682999
confusion matrix [[12299  0 508  5 11  1]
 [  0 3545  0 22 33  0]
 [ 105  0 28622  1  0 72]
 [  3 13 245 12067 129 5049]
 [ 23  0  5 206 12310 17]
 [  4  0 197 327  4 25407]]
auc 0.13421719046968716

```

Figure 7.18: Classification report for MLP.

b. DT - Accuracy: 99.71 %, Area under curve: 0.2261

```

                precision    recall  f1-score   support

     0             1.00         1.00         1.00     12631
     1             1.00         1.00         1.00     3561
     2             1.00         1.00         1.00     28949
     3             1.00         1.00         1.00     17606
     4             1.00         1.00         1.00     12701
     5             1.00         1.00         1.00     25782

 avg / total             1.00         1.00         1.00     101230
0.997095722611874
confusion matrix [[12605    0    8   16    1    1]
 [    0 3561    0    0    0    0]
 [    6    0 28854    3    0   86]
 [    4    0    1 17555   13   33]
 [    0    1    0   10 12690    0]
 [    2    0    81   27    1 25671]]
auc 0.22613452751639032

```

Figure 7.19: *Classification report for DT.*

For decision tree, the class labels have been changed to the order of 0 to 5. 0: Cement, 1: Chalk, 2: Granite, 3: Sandstone, 4: Salt and 5: Shale. In chapter 8, this has been accounted for.

c. SVM - Accuracy: 85.12 %. Area under curve: 0.1327

```

                precision    recall  f1-score   support

     1.0             0.98         0.99         0.98     1267
     2.0             0.98         0.99         0.99         380
     3.0             0.76         0.89         0.82     2834
     4.0             0.93         0.80         0.86     1799
     5.0             0.97         0.99         0.98     1258
     6.0             0.77         0.69         0.73     2585

 avg / total             0.85         0.85         0.85     10123
0.8512298725674208
confusion matrix [[1248    0   11    3    5    0]
 [    1 378    0    0    1    0]
 [   12    0 2511    9    0 302]
 [    9    5   77 1448   26 234]
 [    2    4    3    5 1242    2]
 [    5    0 692   97    1 1790]]
auc 0.13273902450882416

```

Figure 7.20: *Classification report for SVM.*

d. RF - Accuracy: 98.68 %. Area under curve: 0.1314

```

                precision    recall  f1-score   support

     1.0             1.00         0.99         0.99     1289
     2.0             0.99         1.00         0.99         374
     3.0             1.00         1.00         1.00     2869
     4.0             0.96         0.97         0.97     1729
     5.0             0.98         1.00         0.99     1267
     6.0             0.99         0.98         0.98     2595

 avg / total             0.99         0.99         0.99     10123
0.9867628173466364
confusion matrix [[1276    0    3    5    5    0]
 [    0 374    0    0    0    0]
 [    2    0 2862    1    1    3]
 [    1    3    0 1681   19   25]
 [    0    2    0    4 1261    0]
 [    0    0    3   57    0 2535]]
auc 0.13144937942353063

```

Figure 7.21: *Classification report for RF.*

e. GB - Accuracy: 99.18 %. Area under curve: 0.1333

```

                precision    recall  f1-score   support

     1.0         1.00         1.00         1.00        1304
     2.0         1.00         1.00         1.00         356
     3.0         1.00         0.99         1.00        2850
     4.0         0.98         0.99         0.98        1717
     5.0         0.99         1.00         1.00        1283
     6.0         0.99         0.98         0.99        2613

 avg / total         0.99         0.99         0.99        10123
0.9918008495505285
confusion matrix [[1302  0  1  0  1  0]
 [ 0 356  0  0  0  0]
 [ 0  0 2832  2  0 16]
 [ 0  0  0 1700  8  9]
 [ 0  0  0  0 1283  0]
 [ 0  0  6  40  0 2567]]
auc 0.13330603051090406

```

Figure 7.22: Classification report for GB.

f. K-NN - Accuracy: 92.34 %. Area under curve: 0.1423

```

                precision    recall  f1-score   support

     1.0         0.95         0.95         0.95        1270
     2.0         0.98         0.98         0.98         381
     3.0         0.96         0.96         0.96        2903
     4.0         0.85         0.83         0.84        1726
     5.0         0.94         0.95         0.95        1203
     6.0         0.90         0.91         0.90        2640

 avg / total         0.92         0.92         0.92        10123
0.9234416674898746
confusion matrix [[1209  0 13  9 39  0]
 [ 0 373  0  0  8  0]
 [ 10  0 2786 39  1 67]
 [ 20  1  52 1432 18 203]
 [ 31  6  7  12 1147  0]
 [ 4  0 39 195  1 2401]]
auc 0.14232055806204272

```

Figure 7.23: Classification report for K-NN.

g. TPOT - GradientBoostingClassifier - Accuracy: 98.02 %. Area under curve: 0.1400

```

                precision    recall  f1-score   support

     1.0         0.98         0.99         0.99         258
     2.0         1.00         0.99         0.99          76
     3.0         0.99         0.98         0.98         557
     4.0         0.96         0.98         0.97         361
     5.0         0.97         1.00         0.98         251
     6.0         0.98         0.97         0.98         522

 avg / total         0.98         0.98         0.98        2025
0.980246913580247
confusion matrix [[255  0  2  1  0  0]
 [ 0 75  0  0  1  0]
 [ 3  0 547  3  0  4]
 [ 1  0  0 352  4  4]
 [ 0  0  0  1 250  0]
 [ 0  0  6  8  2 506]]
auc 0.13996381410169856

```

Figure 7.24: Classification report for TPOT using an GradientBoostingClassifier.

```

Best pipeline: GradientBoostingClassifier(CombinedDFS(input_matrix, input_matrix), learning_rate=0.1, max_depth=9, max
features=0.5, min_samples_leaf=1, min_samples_split=19, n_estimators=100, subsample=0.5)
Accuracy is 98.0246913580247%

```

Figure 7.25: Recommended pipeline for Case 3 is a GradientBoostingClassifier.

Case 4

<i>Model variable input(s):</i>	z1, z2, z3, RPM, m_torque, WOB, ROP_instantaneous, TF1, TF2, TF3, TF4, TF5, MSE, DOC, BA, D-exponent
<i>Sampling rate:</i>	96 Hz
<i>Labels:</i>	1: Cement, 2: Chalk, 3: Granite, 4: Sandstone, 5: Salt, 6: Shale

a. MLP - Accuracy: 99.36 %, Area under curve: nan

```

                precision    recall  f1-score   support

     3.0         1.00         1.00         1.00         29188
     4.0         0.99         0.99         0.99         17282
     5.0         0.99         0.99         0.99         12732

 avg / total         0.99         0.99         0.99         59202
0.9936150805716023
confusion matrix [[29111  59  18]
 [ 104 17052  126]
 [   0   71 12661]]
auc nan

```

Figure 7.26: *Classification report for MLP.*

b. DT - Accuracy: 99.93 %, Area under curve: 0.9995

```

                precision    recall  f1-score   support

     0          1.00         1.00         1.00         29095
     1          1.00         1.00         1.00         17479
     2          1.00         1.00         1.00         12628

 avg / total         1.00         1.00         1.00         59202
0.9993412384716732
confusion matrix [[29094  1  0]
 [  1 17460  18]
 [  0  19 12609]]
auc 0.9995244258546849

```

Figure 7.27: *Classification report for DT.*

For decision tree, the class labels have been changed to the order of 0 to 5. 0: Cement, 1: Chalk, 2: Granite, 3: Sandstone, 4: Salt and 5: Shale. In chapter 8, this has been accounted for.

c. SVM - Accuracy: 98.83 %. Area under curve: nan

```

                precision    recall  f1-score   support

     3.0         0.99         1.00         0.99         2936
     4.0         0.99         0.97         0.98         1786
     5.0         0.98         0.99         0.99         1199

 avg / total         0.99         0.99         0.99         5921
0.9883465630805607
confusion matrix [[2925  8  3]
 [ 25 1741  20]
 [  0  13 1186]]
auc nan

```

Figure 7.28: *Classification report for SVM.*

d. RF - Accuracy: 99.53 %. Area under curve: nan

```

                precision    recall  f1-score   support

     3.0         1.00         1.00         1.00     2951
     4.0         1.00         0.99         0.99     1711
     5.0         0.99         1.00         0.99     1259

 avg / total         1.00         1.00         1.00     5921
0.9952710690761696
confusion matrix [[2948   3   0]
 [   6 1691  14]
 [   0   5 1254]]
auc nan

```

Figure 7.29: Classification report for RF.

e. GB - Accuracy: 99.85 %. Area under curve: nan

```

                precision    recall  f1-score   support

     3.0         1.00         1.00         1.00     2862
     4.0         1.00         1.00         1.00     1812
     5.0         1.00         1.00         1.00     1247

 avg / total         1.00         1.00         1.00     5921
0.9984799864887688
confusion matrix [[2862   0   0]
 [   0 1807   5]
 [   0   4 1243]]
auc nan

```

Figure 7.30: Classification report for GB.

f. K-NN - Accuracy: 89.92 %. Area under curve: nan

```

                precision    recall  f1-score   support

     3.0         0.88         0.94         0.91     2937
     4.0         0.88         0.78         0.83     1734
     5.0         0.97         0.97         0.97     1250

 avg / total         0.90         0.90         0.90     5921
0.8991724370883297
confusion matrix [[2756  177   4]
 [ 350 1353  31]
 [   23  12 1215]]
auc nan

```

Figure 7.31: Classification report for K-NN.

g. TPOT - ExtraTreesClassifier - Accuracy: 99.32 %. Area under curve: nan

```

                precision    recall  f1-score   support

     3.0         1.00         1.00         1.00     544
     4.0         0.99         0.99         0.99     386
     5.0         0.98         1.00         0.99     254

 avg / total         0.99         0.99         0.99     1184
0.9932432432432432
confusion matrix [[542   2   0]
 [   0 381   5]
 [   0   1 253]]
auc nan

```

Figure 7.32: Classification report for TPOT using an ExtraTreesClassifier.


```
Best pipeline: ExtraTreesClassifier(MaxAbsScaler(CombinedDFs(input_matrix, input_matrix)), bootstrap=False, criterion=entropy, max_features=0.45, min_samples_leaf=2, min_samples_split=6, n_estimators=100)
Accuracy is 99.32432432432432%
```

Figure 7.33: Recommended pipeline for Case 4 is a *ExtraTreesClassifier*.

Case 5

<i>Model variable input(s):</i>	RPM, m_torque, ROP_instantaneous, TF3, DOC, BA
<i>Sampling rate:</i>	96 Hz
<i>Labels:</i>	1: Cement, 2: Chalk, 3: Granite, 4: Sandstone, 5: Salt, 6: Shale

a. MLP - Accuracy: 98.60 %, Area under curve: nan

```

                precision    recall  f1-score   support

     3.0         0.99         0.99         0.99         29188
     4.0         0.98         0.97         0.98         17282
     5.0         0.98         0.99         0.98         12732

 avg / total         0.99         0.99         0.99         59202
0.9859802033715077
confusion matrix [[28971  187   30]
 [ 233 16805  244]
 [   1  135 12596]]
auc nan
```

Figure 7.34: Classification report for MLP.

b. DT - Accuracy: 99.94 %, Area under curve: 0.9996

```

                precision    recall  f1-score   support

     0         1.00         1.00         1.00         29095
     1         1.00         1.00         1.00         17479
     2         1.00         1.00         1.00         12628

 avg / total         1.00         1.00         1.00         59202
0.9993919124353907
confusion matrix [[29095   0   0]
 [  1 17459  19]
 [  0   16 12612]]
auc 0.9995582827925693
```

Figure 7.35: Classification report for DT.

For decision tree, the class labels have been changed to the order of 0 to 5. 0: Cement, 1: Chalk, 2: Granite, 3: Sandstone, 4: Salt and 5: Shale. In chapter 8, this has been accounted for.

c. SVM - Accuracy: 97.50 %. Area under curve: nan

```

                precision    recall  f1-score   support

     3.0         0.97         0.99         0.98         2856
     4.0         0.98         0.94         0.96         1758
     5.0         0.97         0.99         0.98         1307

 avg / total         0.98         0.98         0.97         5921
0.9750042222597535
confusion matrix [[2833  15   8]
 [ 77 1646  35]
 [  1  12 1294]]
auc nan

```

Figure 7.36: *Classification report for SVM.*

d. RF - Accuracy: 99.66 %. Area under curve: nan

```

                precision    recall  f1-score   support

     3.0         1.00         1.00         1.00         2864
     4.0         0.99         0.99         0.99         1790
     5.0         0.99         1.00         0.99         1267

 avg / total         1.00         1.00         1.00         5921
0.996622192197264
confusion matrix [[2860   4   0]
 [  0 1779  11]
 [  0   5 1262]]
auc nan

```

Figure 7.37: *Classification report for RF.*

e. GB - Accuracy: 99.93 %. Area under curve: nan

```

                precision    recall  f1-score   support

     3.0         1.00         1.00         1.00         2899
     4.0         1.00         1.00         1.00         1768
     5.0         1.00         1.00         1.00         1254

 avg / total         1.00         1.00         1.00         5921
0.9993244384394528
confusion matrix [[2897   2   0]
 [  0 1766   2]
 [  0   0 1254]]
auc nan

```

Figure 7.38: *Classification report for GB.*

f. K-NN - Accuracy: 97.74 %. Area under curve: nan

```

                precision    recall  f1-score   support

     3.0         0.98         0.98         0.98         2927
     4.0         0.97         0.96         0.97         1743
     5.0         0.98         0.98         0.98         1251

 avg / total         0.98         0.98         0.98         5921
0.9773686877216686
confusion matrix [[2882  40   5]
 [ 50 1675  18]
 [  8  13 1230]]
auc nan

```

Figure 7.39: *Classification report for K-NN.*

g. TPOT - GradientBoostingClassifier - Accuracy: 99.83 %. Area under curve: nan

```

                precision    recall  f1-score   support

         3.0         1.00         1.00         1.00         524
         4.0         1.00         0.99         1.00         395
         5.0         1.00         1.00         1.00         265

 avg / total         1.00         1.00         1.00         1184
0.9983108108108109
confusion matrix [[524  0  0]
 [ 1 393  1]
 [ 0  0 265]]
auc nan

```

Figure 7.40: *Classification report for TPOT using an GradientBoostingClassifier.*

```

Best pipeline: GradientBoostingClassifier(input_matrix, learning_rate=0.5, max_depth=10, max_features=0.45, min_samples_leaf=3, min_samples_split=19, n_estimators=100, subsample=0.9000000000000001)
Accuracy is 99.83108108108108%

```

Figure 7.41: *Recommended pipeline for Case 5 is a GradientBoostingClassifier.*

Evaluation of best models from Cases 1-5:

With regards to Figure 3.11, it can be observed that the tree models (DT, RF, GB) score equally high in cases 1 and 2 (with and without outliers present in the data), corresponding with that these models are considered robust to outliers in the input space. These models also perform very well when considering both the accuracy and area under curve when the number of rock samples that the models get trained on reduce from six to three. The TPOT algorithm also recommends to use either a GradientBoostingClassifier (3 of the cases) or an ExtraTreesClassifier (2 of the cases).

For the support vector machine, Figure 3.11 suggests that the ability to extract linear combinations of features is high, but that the model is both weak with regards to computational scalability and natural handling of mixed-type data. As can be observed in cases 1 to 3, both of these statements appear correct. For cases 4 and 5 however, when the number of rock samples have been reduced to three, an increase by approximately 10 % can be noted. The same applies to the multilayer perceptron model, which appears to perform much better when the number of samples have been reduced to three.

With regards to K-NN, the model appears to score better when the number of features are low. As shown in Figure 3.11, the K-NN models ability to extract linear combinations of features is fairly good.

Considering all models, it is our recommendation to use tree classifiers for rock formation classification on the laboratory drilling rig. The hypothesis from chapter 6 suggesting that outlier removal might not be necessary with the laboratory

rig appears to hold. Furthermore, it can be observed that when using the algorithm to identify the highest scoring features, and manually selecting those that are considered the most applicable, the number of features to train and classify formations can be reduced from 16 to 6.

7.2.2 Volve formation classification

The processed dataset consists of 2335 rows \times 14 feature columns (+ 1 label column), containing data for six rocks as mentioned in chapter 6. The sampling frequency of the dataset is unknown, thus no features get engineered. The data has been processed for missing or invalid data, normalized and then outliers have been removed. Features available in the Volve formation data are: rate of penetration, mechanical specific energy, weight on bit, bit torque, rpm, flow rate, mud weight_{in}, depth of cut, bit aggressiveness, TF1, TF2, TF3, TF4 and TF5 (see chapter 6).

From running the ExtraTreesClassifier feature importance algorithm, the features in the dataset scores as following:

Feature ranking:	Score:
1. feature 6 (0.112874)	(1) Mud weight in
2. feature 12 (0.102129)	(2) TF3
3. feature 5 (0.097017)	(3) Flow
4. feature 4 (0.091674)	(4) RPM
5. feature 3 (0.072398)	(5) Torque
6. feature 7 (0.067173)	(6) Depth of cut
7. feature 1 (0.064503)	(7) MSE
8. feature 10 (0.064132)	(8) TF1
9. feature 0 (0.062740)	(9) Rate of penetration
10. feature 8 (0.058711)	(10) Bit aggressiveness
11. feature 11 (0.057039)	(11) TF2
12. feature 2 (0.051201)	(12) Weight on bit
13. feature 13 (0.050650)	(13) TF4
14. feature 14 (0.047759)	(14) TF5
15. feature 9 (0.000000)	(15) Mud type (invalid)

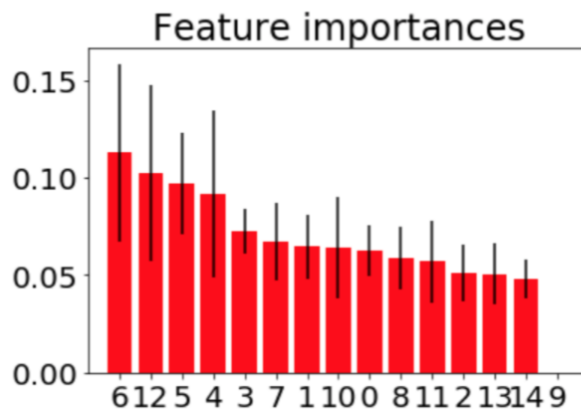


Figure 7.42: Volve data feature importance for rock classification.

The flow rate and mud weight going into the well can not be used to classify the different formations (as these are dependent on the operation). The six highest scoring features then become:

Score:	1	2	3	4	5	6
Feature	TF3	RPM	Torque	DOC	MSE	TF1

Another challenge with the processed Vove data is the number of occurrences for formation type 7 (coal). In the original dataset coal had nine observations, and after outlier removal, only three observations are left. For this reason, the coal class has been removed from the dataset before all models get tuned and built. The six highest scoring features when coal observations have been removed from the dataset are the same as before. For case 6 through 10, the following labels have been used: 1: Claystone, 2: Sandstone, 3: Siltstone, 4: Tuff, 5: Marl, 6: Limestone and 7: Coal.

Model Parameter Tuning

a. Multi Layer Perceptron (MLP): optimal parameter search is performed evaluating the following parameters on 100 % of the dataset and all features except flow rate, mud weight in and mud type.

```
1 params_to_test = {'hidden_layer_sizes': [(10,10,10), (50,50,50), ...
    (100,100,100)], 'activation': ['tanh', 'relu'], 'solver': ['sgd', ...
    'adam'], 'alpha': [0.0001, 0.1], 'learning_rate': ['adaptive', ...
    'invscaling'],}
```

The random state, grid-search cross-validation generator (cv), and scoring values are:

```
1 MLP_model = MLPClassifier(random_state=None)
2 grid_search = GridSearchCV(MLP_model, param_grid=params_to_test, cv=3, ...
    scoring='f1_macro', n_jobs=4)
```

The best model parameters are:

```
1 MLPClassifier(activation='relu', alpha=0.0001, batch_size='auto', ...
    beta_1=0.9, beta_2=0.999, early_stopping=False, epsilon=1e-08, ...
    hidden_layer_sizes=(100, 100, 100), learning_rate='adaptive', ...
    learning_rate_init=0.001, max_iter=200, momentum=0.9, ...
    nesterovs_momentum=True, power_t=0.5, random_state=None, ...
    shuffle=True, solver='adam', tol=0.0001, validation_fraction=0.1, ...
    verbose=False, warm_start=False)
```

b. Decision Tree (DT): optimal parameter search is performed evaluating the following parameters on 100 % of the dataset and all features except flow rate, mud weight in and mud type.

```
1 params_to_test = {'criterion': ['gini', 'entropy'], 'max_depth': ...
    [4, 5, 6, 7, 8, 9, 10, 11, 12, 15, 20, 30, 40, 50, 70, 90, 120, 150]}
```

The random state, grid-search cross-validation generator (cv), and scoring is shown below:

```
1 DT_model = DecisionTreeClassifier(random_state=None)
2 grid_search = GridSearchCV(DT_model, param_grid=params_to_test, cv=3, ...
    scoring='f1_macro', n_jobs=4)
```

The best model parameters are:

```
1 DecisionTreeClassifier(class_weight=None, criterion='gini', max_depth=10, ...
    max_features=None, max_leaf_nodes=None, min_impurity_decrease=0.0, ...
    min_impurity_split=None, min_samples_leaf=1, min_samples_split=2, ...
    min_weight_fraction_leaf=0.0, presort=False, random_state=None, ...
    splitter='best')
```

c. Support Vector Machine (SVM): optimal parameter search is performed evaluating the following parameters on 100 % of the dataset and all features except flow rate, mud weight in and mud type.

```
1 params_to_test = {'kernel': ['rbf', 'linear'], 'gamma': [1e-3, 1e-4], ...
    'C': [1, 10, 100, 1000]}
```

The random state, grid-search cross-validation generator (cv), and scoring is shown below:

```
1 SV_model = SVC(random_state=None)
2 grid_search = GridSearchCV(SV_model, param_grid=params_to_test, cv=3, ...
    scoring='f1_macro', n_jobs=4)
```

The best model parameters are:

```
1 SVC(C=10, cache_size=200, class_weight=None, coef0=0.0, ...
    decision_function_shape='ovr', degree=3, gamma=0.001, ...
    kernel='linear', max_iter=-1, probability=False, random_state=None, ...
    shrinking=True, tol=0.001, verbose=False)
```

d. Random Forest (RF): optimal parameter search is performed evaluating the following parameters on 100 % of the dataset and all features except flow rate, mud weight in and mud type.

```
1 params_to_test = {'n_estimators':[1,50,10], 'max_depth':[5,10,1]}
```

The random state, grid-search cross-validation generator (cv), and scoring is shown below:

```

1 RF_model = RandomForestClassifier(random_state=42)
2 grid_search = GridSearchCV(RF_model, param_grid=params_to_test, cv=10, ...
    scoring='f1_macro', n_jobs=4)

```

The best model parameters are:

```

1 RandomForestClassifier(bootstrap=True, class_weight=None, ...
    criterion='gini', max_depth=10, max_features='auto', max_leaf_nodes=None,
2 min_impurity_decrease=0.0, min_impurity_split=None, min_samples_leaf=1, ...
    min_samples_split=2, min_weight_fraction_leaf=0.0, n_estimators=50, ...
    n_jobs=1, oob_score=False, random_state=42, verbose=0, warm_start=False)

```

e. Gradient Boosting (GB): optimal parameter search is performed evaluating the following parameters on 100 % of the dataset and all features except flow rate, mud weight in and mud type.

```

1 params_to_test = {'learning_rate': [0.1, 0.05, 0.02, 0.01], 'max_depth': ...
    [4, 6, 8], 'min_samples_leaf': [20, 50, 100, 150]
2 #'max_features': [1.0, 0.3, 0.1]}

```

The random state, grid-search cross-validation generator (cv), and scoring is shown below:

```

1 GB_model = GradientBoostingClassifier(random_state=None)
2
3 grid_search = GridSearchCV(GB_model, param_grid=params_to_test, cv=3, ...
    scoring='f1_macro', n_jobs=4)

```

The best model parameters are:

```

1 GradientBoostingClassifier(criterion='friedman_mse', init=None, ...
    learning_rate=0.05, loss='deviance', max_depth=8, max_features=None, ...
    max_leaf_nodes=None, min_impurity_decrease=0.0, ...
    min_impurity_split=None, min_samples_leaf=20, min_samples_split=2, ...
    min_weight_fraction_leaf=0.0, n_estimators=100, presort='auto', ...
    random_state=None, subsample=1.0, verbose=0, warm_start=False)

```

f. K-Nearest Neighbor (K-NN): optimal parameter search is performed evaluating the following parameters on 100 % of the dataset and all features except flow rate, mud weight in and mud type.

```

1 params_to_test = {'n_neighbors':[3,5,11], 'weights':['distance', 'uniform']}

```

The random state, grid-search cross-validation generator (cv), and scoring is shown below:


```

1 KNN_model = KNeighborsClassifier()
2 grid_search = GridSearchCV(KNN_model, param_grid=params_to_test, cv=3, ...
    scoring='f1_macro', n_jobs=4)

```

The best model parameters are:

```

1 KNeighborsClassifier(algorithm='auto', leaf_size=30, metric='minkowski', ...
    metric_params=None, n_jobs=1, n_neighbors=10, p=2, weights='distance')

```

g. TPOT Algorithm: optimal pipeline search is performed using the following parameters on 1 % of the dataset and all features except flow rate, mud weight in and mud type.

```

1 tpot = TPOTClassifier(generations=10, population_size=50, verbosity=2)

```

With regards to the best pipeline for TPOT, this changes for every case. Due to this, the recommended pipeline from the software is presented at the end of each case (for cases 1 - 10).

Case 6

<i>Model variable input(s):</i>	ROP, MSE, WOB, Torque, RPM, DOC, BA, TF1, TF2, TF3, TF4, TF5
<i>Sampling rate:</i>	Unknown
<i>Labels:</i>	1: Claystone, 2: Sandstone, 3: Siltstone, 5: Marl, 6: Limestone

a. MLP - Accuracy: 59.86 %, Area under curve: 0.2655

```

              precision    recall  f1-score   support

     1.0         0.57         0.26         0.36         130
     2.0         0.63         0.79         0.70         234
     3.0         0.00         0.00         0.00          35
     5.0         0.64         0.25         0.36         106
     6.0         0.57         0.86         0.69         215

 avg / total         0.57         0.60         0.55         720
0.5986111111111111
confusion matrix [[ 34  52  0  5  39]
 [ 14 186  0  3  31]
 [  4  25  0  0  6]
 [  3  14  0 27  62]
 [  5  19  0  7 184]]
auc 0.26546727164011114

```

Figure 7.43: Classification report for MLP.

b. DT - Accuracy: 57.36 %, Area under curve: 0.3262

```

                precision    recall  f1-score   support

     0         0.43         0.33         0.37         135
     1         0.66         0.71         0.68         259
     2         0.24         0.16         0.19          32
     3         0.47         0.43         0.45          96
     4         0.60         0.71         0.65         198

 avg / total         0.56         0.57         0.56         720
0.5736111111111111
confusion matrix [[ 44  43   3  10  35]
 [ 26 183   9  11  30]
 [   6  19   5   2   0]
 [   9  15   2  41  29]
 [  17  16   2  23 140]]
auc 0.32619458575581395

```

Figure 7.44: *Classification report for DT.*

For decision tree, the class labels have been changed to the order of 0 to 4. 0: Claystone, 1: Sandstone, 2: Siltstone, 3: Marl and 4: Limestone. In chapter 8, this has been accounted for.

c. SVM - Accuracy: 54.03 %. Area under curve: 0.2795

```

                precision    recall  f1-score   support

     1.0         0.56         0.07         0.12         135
     2.0         0.55         0.84         0.67         259
     3.0         0.00         0.00         0.00          32
     5.0         0.00         0.00         0.00          96
     6.0         0.52         0.82         0.64         198

 avg / total         0.45         0.54         0.44         720
0.5402777777777777
confusion matrix [[   9  84   0   0  42]
 [  5 217   0   0  37]
 [   1  24   0   0   7]
 [   0  33   0   0  63]
 [   1  34   0   0 163]]
auc 0.2794956406670073

```

Figure 7.45: *Classification report for SVM.*

d. RF - Accuracy: 71.39 %. Area under curve: 0.2468

```

                precision    recall  f1-score   support

     1.0         0.61         0.31         0.41         112
     2.0         0.71         0.90         0.79         278
     3.0         1.00         0.09         0.16          35
     5.0         0.83         0.53         0.65          94
     6.0         0.72         0.88         0.79         201

 avg / total         0.73         0.71         0.68         720
0.7138888888888889
confusion matrix [[ 35  49   0   5  23]
 [ 11 250   0   0  17]
 [   4  26   3   1   1]
 [   4  11   0  50  29]
 [   3  18   0   4 176]]
auc 0.2467568931280315

```

Figure 7.46: *Classification report for RF.*

e. GB - Accuracy: 68.89 %. Area under curve: 0.2751

```

                precision    recall  f1-score   support

     1.0         0.65         0.43         0.52         128
     2.0         0.71         0.86         0.78         261
     3.0         0.00         0.00         0.00          20
     5.0         0.66         0.50         0.57         100
     6.0         0.70         0.79         0.74         211

 avg / total         0.67         0.69         0.67         720
0.6888888888888889
confusion matrix [[ 55  47   0   5  21]
 [ 14 224   4   5  14]
 [   3  13   0   1   3]
 [   5  10   0  50  35]
 [   7  22   0  15 167]]
auc 0.27507324768988056

```

Figure 7.47: *Classification report for GB.*

f. K-NN - Accuracy: 67.22 %. Area under curve: 0.3013

```

                precision    recall  f1-score   support

     1.0         0.57         0.47         0.52         127
     2.0         0.71         0.79         0.75         266
     3.0         0.25         0.11         0.15          28
     5.0         0.57         0.56         0.57          85
     6.0         0.73         0.76         0.74         214

 avg / total         0.66         0.67         0.66         720
0.6722222222222223
confusion matrix [[ 60  42   2   6  17]
 [ 24 210   6   5  21]
 [   5  17   3   1   2]
 [   4  11   1  48  21]
 [  12  15   0  24 163]]
auc 0.3012569971183465

```

Figure 7.48: *Classification report for K-NN.*

g. TPOT - GradientBoostingClassifier - Accuracy: 69.86 %. Area under curve: 0.2598

```

                precision    recall  f1-score   support

     1.0         0.51         0.44         0.47         123
     2.0         0.70         0.83         0.76         237
     3.0         0.43         0.22         0.29          27
     5.0         0.77         0.46         0.58         110
     6.0         0.77         0.87         0.82         223

 avg / total         0.69         0.70         0.68         720
0.6986111111111111
confusion matrix [[ 54  51   4   5   9]
 [ 25 197   4   1  10]
 [   3  14   6   0   4]
 [  13  12   0  51  34]
 [  10   9   0   9 195]]
auc 0.25983436853002073

```

Figure 7.49: *Classification report for TPOT using an GradientBoostingClassifier.*

```

Best pipeline: GradientBoostingClassifier(MinMaxScaler(SelectFwe(input_matrix, alpha=0.033)), learning_rate=0.1, max_depth=7, max_features=0.5, min_samples_leaf=1, min_samples_split=9, n_estimators=100, subsample=0.8500000000000001)
Accuracy is 69.86111111111111%

```

Figure 7.50: *Recommended pipeline for Case 6 is a GradientBoostingClassifier.*

Case 7

<i>Model variable input(s):</i>	ROP, MSE, WOB, Torque, RPM, DOC, BA, TF1, TF2, TF3, TF4, TF5
<i>Sampling rate:</i>	Unknown
<i>Labels:</i>	1: Claystone, 2: Sandstone, 3: Siltstone, 5: Marl, 6: Limestone

a. MLP - Accuracy: 62.96 %, Area under curve: 0.2333

```

                precision    recall  f1-score   support

     1.0         0.50         0.04         0.08         49
     2.0         0.61         0.92         0.73        180
     3.0         0.00         0.00         0.00         28
     5.0         0.00         0.00         0.00         51
     6.0         0.67         0.79         0.72        159

 avg / total         0.51         0.63         0.54        467
0.6295503211991434
confusion matrix [[ 2  37  0  0  10]
 [ 1 166  0  0  13]
 [ 0  24  0  0  4]
 [ 1  14  0  0  36]
 [ 0  33  0  0 126]]
auc 0.2332655826558266

```

Figure 7.51: *Classification report for MLP.***b. DT - Accuracy: 62.31 %, Area under curve: 0.3430**

```

                precision    recall  f1-score   support

     0         0.33         0.26         0.29         57
     1         0.69         0.84         0.75        178
     2         0.06         0.05         0.05         21
     3         0.36         0.31         0.33         55
     4         0.77         0.70         0.73        156

 avg / total         0.61         0.62         0.61        467
0.6231263383297645
confusion matrix [[ 15  31  2  4  5]
 [ 8 149 10  5  6]
 [ 3  15  1  0  2]
 [ 7  10  2 17 19]
 [12  12  2 21 109]]
auc 0.3430493273542601

```

Figure 7.52: *Classification report for DT.*

For decision tree, the class labels have been changed to the order of 0 to 4. 0: Claystone, 1: Sandstone, 2: Siltstone, 3: Marl and 4: Limestone. In chapter 8, this has been accounted for.

c. SVM - Accuracy: 62.74 %. Area under curve: 0.2186

```

                precision    recall  f1-score   support

     1.0         0.00         0.00         0.00         57
     2.0         0.62         0.92         0.74        178
     3.0         0.00         0.00         0.00         21
     5.0         0.00         0.00         0.00         55
     6.0         0.64         0.83         0.73        156

 avg / total         0.45         0.63         0.52        467
0.6274089935760171
confusion matrix [[ 0  41  0  0  16]
 [ 0 163  0  0  15]
 [ 0  18  0  0  3]
 [ 0  17  0  0  38]
 [ 0  26  0  0 130]]
auc 0.21860541969596828

```

Figure 7.53: *Classification report for SVM.*

d. RF - Accuracy: **72.38 %**. Area under curve: **0.2166**

```

                precision    recall  f1-score   support

     1.0         0.50         0.30         0.37         44
     2.0         0.74         0.89         0.81        189
     3.0         0.00         0.00         0.00         25
     5.0         0.64         0.16         0.26         43
     6.0         0.74         0.90         0.81        166

 avg / total         0.67         0.72         0.67        467
0.7237687366167024
confusion matrix [[ 13  18  0  1  12]
 [ 5 168  0  1  15]
 [ 1  22  0  1  1]
 [ 3  8  0  7  25]
 [ 4  11  0  1 150]]
auc 0.21661718244452055

```

Figure 7.54: *Classification report for RF.*

e. GB - Accuracy: **69.81 %**. Area under curve: **0.2292**

```

                precision    recall  f1-score   support

     1.0         0.36         0.34         0.35         44
     2.0         0.73         0.84         0.78        189
     3.0         0.20         0.04         0.06         26
     5.0         0.38         0.27         0.32         37
     6.0         0.81         0.82         0.82        171

 avg / total         0.66         0.70         0.68        467
0.6980728051391863
confusion matrix [[ 15  20  1  1  7]
 [ 14 159  3  2  11]
 [ 2  23  1  0  0]
 [ 7  4  0  10  16]
 [ 4  13  0  13 141]]
auc 0.2292451752883408

```

Figure 7.55: *Classification report for GB.*

f. K-NN - Accuracy: **68.95 %**. Area under curve: **0.2532**

```

                precision    recall  f1-score   support

     1.0         0.50         0.32         0.39         59
     2.0         0.70         0.87         0.78        173
     3.0         0.22         0.12         0.16         16
     5.0         0.56         0.29         0.38         51
     6.0         0.76         0.81         0.78        168

 avg / total         0.66         0.69         0.67        467
0.6895074946466809
confusion matrix [[ 19  26   1   0  13]
 [  5 150   5   1  12]
 [  3  10   2   0   1]
 [  2  15   1  15  18]
 [  9  12   0  11 136]]
auc 0.25324407219535217

```

Figure 7.56: Classification report for K-NN.

g. TPOT - GradientBoostingClassifier - Accuracy: 71.09 %. Area under curve: 0.2353

```

                precision    recall  f1-score   support

     1.0         0.46         0.36         0.40         47
     2.0         0.73         0.88         0.80        199
     3.0         0.00         0.00         0.00         27
     5.0         0.48         0.38         0.42         37
     6.0         0.81         0.80         0.80        157

 avg / total         0.67         0.71         0.68        467
0.7109207708779444
confusion matrix [[ 17  18   0   6   6]
 [ 10 176   4   2   7]
 [  3  21   0   0   3]
 [  0   9   0  14  14]
 [  7  17   1   7 125]]
auc 0.23528088202205055

```

Figure 7.57: Classification report for TPOT using an GradientBoostingClassifier.

```

Best pipeline: GradientBoostingClassifier(input_matrix, learning_rate=0.1, max_depth=9, max_features=0.5, min_samples
_leaf=9, min_samples_split=20, n_estimators=100, subsample=0.45)
Accuracy is 71.09207708779444%

```

Figure 7.58: Recommended pipeline for Case 7 is a GradientBoostingClassifier.

Case 8

Model variable input(s):	MSE, Torque, RPM, DOC, TF1, TF3
Sampling rate:	Unknown
Labels:	1: Claystone, 2: Sandstone, 3: Siltstone, 5: Marl, 6: Limestone

a. MLP - Accuracy: 61.67 %, Area under curve: 0.2329

```

                precision    recall  f1-score   support

     1.0         0.00         0.00         0.00         49
     2.0         0.61         0.90         0.72        180
     3.0         0.00         0.00         0.00         28
     5.0         0.00         0.00         0.00         51
     6.0         0.63         0.79         0.70        159

 avg / total         0.45         0.62         0.52        467
0.6167023554603854
confusion matrix [[ 0 36  0  0 13]
 [ 0 162  0  0 18]
 [ 0 24  0  0  4]
 [ 0 12  0  0 39]
 [ 0 33  0  0 126]]
auc 0.2329268292682927

```

Figure 7.59: *Classification report for MLP.*

b. DT - Accuracy: 59.96 %, Area under curve: 0.3835

```

                precision    recall  f1-score   support

     0         0.25         0.25         0.25         57
     1         0.68         0.80         0.74        178
     2         0.10         0.10         0.10         21
     3         0.36         0.29         0.32         55
     4         0.77         0.67         0.72        156

 avg / total         0.59         0.60         0.59        467
0.5995717344753747
confusion matrix [[ 14 30  2  4  7]
 [ 16 143 12  2  5]
 [  2  15  2  0  2]
 [ 10 12  0 16 17]
 [ 13 11  4 23 105]]
auc 0.38351484091394406

```

Figure 7.60: *Classification report for DT.*

For decision tree, the class labels have been changed to the order of 0 to 4. 0: Claystone, 1: Sandstone, 2: Siltstone, 3: Marl and 4: Limestone. In chapter 8, this has been accounted for.

c. SVM - Accuracy: 62.10 %. Area under curve: 0.2212

```

                precision    recall  f1-score   support

     1.0         0.00         0.00         0.00         57
     2.0         0.60         0.94         0.73        178
     3.0         0.00         0.00         0.00         21
     5.0         0.00         0.00         0.00         55
     6.0         0.65         0.79         0.71        156

 avg / total         0.45         0.62         0.52        467
0.6209850107066381
confusion matrix [[ 0 41  0  0 16]
 [ 0 167  0  0 11]
 [ 0 18  0  0  3]
 [ 0 18  0  0 37]
 [ 0 33  0  0 123]]
auc 0.22121029508961546

```

Figure 7.61: *Classification report for SVM.*

d. RF - Accuracy: 71.95 %. Area under curve: 0.1781

```

                precision    recall  f1-score   support

     1.0         0.60      0.20      0.30         45
     2.0         0.75      0.95      0.84        196
     3.0         0.33      0.04      0.07         27
     5.0         0.55      0.22      0.31         51
     6.0         0.71      0.87      0.78        148

 avg / total         0.68      0.72      0.67        467
0.7194860813704497
confusion matrix [[ 9  21  0  2 13]
 [ 2 186  1  1  6]
 [ 0  20  1  0  6]
 [ 3  10  0 11 27]
 [ 1  11  1  6 129]]
auc 0.17811017395888246

```

Figure 7.62: *Classification report for RF.*

e. GB - Accuracy: 70.02 %. Area under curve: 0.2062

```

                precision    recall  f1-score   support

     1.0         0.43      0.24      0.31         55
     2.0         0.72      0.90      0.80        191
     3.0         0.00      0.00      0.00         20
     5.0         0.48      0.33      0.39         43
     6.0         0.78      0.82      0.80        158

 avg / total         0.65      0.70      0.67        467
0.7002141327623126
confusion matrix [[ 13  29  2  2  9]
 [ 9 171  4  1  6]
 [ 1  18  0  0  1]
 [ 3  5  0 14 21]
 [ 4  13  0 12 129]]
auc 0.20618028681994083

```

Figure 7.63: *Classification report for GB.*

f. K-NN - Accuracy: 66.38 %. Area under curve: 0.2407

```

                precision    recall  f1-score   support

     1.0         0.49      0.29      0.36         59
     2.0         0.67      0.86      0.75        173
     3.0         0.10      0.06      0.08         16
     5.0         0.45      0.29      0.36         51
     6.0         0.77      0.77      0.77        168

 avg / total         0.64      0.66      0.64        467
0.6638115631691649
confusion matrix [[ 17  30  2  2  8]
 [ 9 148  5  0 11]
 [ 1  13  1  0  1]
 [ 1  15  1 15 19]
 [ 7  15  1 16 129]]
auc 0.2407494789823444

```

Figure 7.64: *Classification report for K-NN.*

g. TPOT - KNeighborsClassifier - Accuracy: 71.73 %. Area under curve: 0.2291


```

                precision    recall  f1-score   support

     1.0         0.58         0.39         0.46         49
     2.0         0.72         0.90         0.80        184
     3.0         0.33         0.04         0.06         28
     5.0         0.43         0.32         0.37         41
     6.0         0.80         0.82         0.81        165

 avg / total         0.68         0.72         0.69        467
0.7173447537473233
confusion matrix [[ 19  21   2   2   5]
 [ 7 166   0   2   9]
 [ 2  24   1   0   1]
 [ 4   6   0  13  18]
 [ 1  15   0  13 136]]
auc 0.22910585343370715

```

Figure 7.65: Classification report for TPOT using a KNeighborsClassifier.

```

Best pipeline: KNeighborsClassifier(zeroCountExtraTreesClassifier(input_matrix, bootstrap=True, criterion=gini, max_
feature=0.9000000000000001, min_samples_leaf=1, min_samples_split=3, n_estimators=100), n_neighbors=98, p=2, weight_
distance)
Accuracy is 0.7173447537473233

```

Figure 7.66: Recommended pipeline for Case 8 is a KNeighborsClassifier.

Case 9

<i>Model variable input(s):</i>	ROP, MSE, WOB, Torque, RPM, DOC, BA, TF1, TF2, TF3, TF4, TF5
<i>Sampling rate:</i>	Unknown
<i>Labels:</i>	1: Claystone, 2: Sandstone, 6: Limestone

a. MLP - Accuracy: 76.94 %, Area under curve: 0.1878

```

                precision    recall  f1-score   support

     1.0         0.00         0.00         0.00         49
     2.0         0.76         0.87         0.81        191
     6.0         0.78         0.88         0.83        159

 avg / total         0.67         0.77         0.72        399
0.7694235588972431
confusion matrix [[  0  33  16]
 [ 0 167  24]
 [ 0  19 140]]
auc 0.18782722513089004

```

Figure 7.67: Classification report for MLP.

b. DT - Accuracy: 78.20 %, Area under curve: 0.8791

```

                precision    recall  f1-score   support

     0         0.41         0.34         0.37         50
     1         0.79         0.84         0.81        188
     2         0.86         0.86         0.86        161

 avg / total         0.77         0.78         0.78        399
0.7819548872180451
confusion matrix [[ 17  23  10]
 [ 19 157  12]
 [  5  18 138]]
auc 0.8790646693459994

```

Figure 7.68: Classification report for DT.

For decision tree, the class labels have been changed to the order of 0 to 2. 0: Claystone, 1: Sandstone and 2: Limestone. In chapter 8, this has been accounted for.

c. SVM - Accuracy: 75.94 %. Area under curve: 0.1883

```

                precision    recall  f1-score   support

     1.0         0.00         0.00         0.00         50
     2.0         0.75         0.89         0.81        188
     6.0         0.78         0.84         0.81        161

 avg / total         0.66         0.76         0.71        399
0.7593984962406015
confusion matrix [[ 0 31 19]
 [ 0 168 20]
 [ 0 26 135]]
auc 0.1882625794090955

```

Figure 7.69: *Classification report for SVM.*

d. RF - Accuracy: 83.46 %. Area under curve: 0.1686

```

                precision    recall  f1-score   support

     1.0         0.73         0.16         0.26         50
     2.0         0.84         0.92         0.88        206
     6.0         0.83         0.94         0.88        143

 avg / total         0.82         0.83         0.80        399
0.8345864661654135
confusion matrix [[ 8 28 14]
 [ 2 190 14]
 [ 1 7 135]]
auc 0.16859499974847833

```

Figure 7.70: *Classification report for RF.*

e. GB - Accuracy: 79.70 %. Area under curve: 0.2158

```

                precision    recall  f1-score   support

     1.0         0.43         0.21         0.28         48
     2.0         0.78         0.89         0.83        196
     6.0         0.88         0.86         0.87        155

 avg / total         0.78         0.80         0.78        399
0.7969924812030075
confusion matrix [[ 10 33 5]
 [ 9 174 13]
 [ 4 17 134]]
auc 0.21575600683623203

```

Figure 7.71: *Classification report for GB.*

f. K-NN - Accuracy: 78.95 %. Area under curve: 0.2379

```

                precision    recall  f1-score   support

     1.0         0.52         0.31         0.39         51
     2.0         0.77         0.88         0.82        184
     6.0         0.86         0.84         0.85        164

 avg / total         0.78         0.79         0.78        399
0.7894736842105263
confusion matrix [[ 16  27   8]
 [  9 161  14]
 [  6  20 138]]
auc 0.23792972699696668

```

Figure 7.72: Classification report for K-NN.

g. TPOT - RandomForestClassifier - Accuracy: 84.21 %. Area under curve: 0.1696

```

                precision    recall  f1-score   support

     1.0         0.49         0.39         0.43         44
     2.0         0.88         0.87         0.87        187
     6.0         0.88         0.93         0.90        168

 avg / total         0.83         0.84         0.84        399
0.8421052631578947
confusion matrix [[ 17  15  12]
 [ 14 163  10]
 [  4   8 156]]
auc 0.16962213701947332

```

Figure 7.73: Classification report for TPOT using an RandomForestClassifier.

Best pipeline: RandomForestClassifier(MinMaxScaler(PolynomialFeatures(input_matrix, degree=2, include_bias=False, interaction_only=False)), bootstrap=False, criterion=gini, max_features=0.3, min_samples_leaf=3, min_samples_split=13, n_estimators=100)
 Accuracy is 84.21052631578947%

Figure 7.74: Recommended pipeline for Case 9 is RandomForestClassifier.

Case 10

Model variable input(s):	MSE, Torque, RPM, DOC, TF1, TF3
Sampling rate:	Unknown
Labels:	1: Claystone, 2: Sandstone, 6: Limestone

a. MLP - Accuracy: 76.94 %, Area under curve: 0.1892

```

                precision    recall  f1-score   support

     1.0         0.00         0.00         0.00         49
     2.0         0.75         0.90         0.82        191
     6.0         0.80         0.85         0.82        159

 avg / total         0.68         0.77         0.72        399
0.7694235588972431
confusion matrix [[  0  34  15]
 [  0 172  19]
 [  0  24 135]]
auc 0.18916129681836488

```

Figure 7.75: Classification report for MLP.

b. DT - Accuracy: 78.85 %, Area under curve: 0.8609

```

                precision    recall  f1-score   support

     0           0.43         0.32         0.37         50
     1           0.80         0.85         0.83        188
     2           0.84         0.85         0.85        161

 avg / total         0.77         0.78         0.78        399
0.7844611528822055
confusion matrix [[ 16  22  12]
 [ 14 160  14]
 [  7  17 137]]
auc 0.8608747846964873

```

Figure 7.76: *Classification report for DT.*

For decision tree, the class labels have been changed to the order of 0 to 2. 0: Claystone, 1: Sandstone and 2: Limestone. In chapter 8, this has been accounted for.

c. SVM - Accuracy: 74.94 %. Area under curve: 0.1969

```

                precision    recall  f1-score   support

     1.0           0.00         0.00         0.00         50
     2.0           0.73         0.91         0.81        188
     6.0           0.78         0.80         0.79        161

 avg / total         0.66         0.75         0.70        399
0.7493734335839599
confusion matrix [[  0  31  19]
 [  0 171  17]
 [  0  33 128]]
auc 0.19687153372995866

```

Figure 7.77: *Classification report for SVM.***d. RF - Accuracy: 86.72 %. Area under curve: 0.1578**

```

                precision    recall  f1-score   support

     1.0           0.75         0.27         0.40         44
     2.0           0.85         0.95         0.90        191
     6.0           0.89         0.93         0.91        164

 avg / total         0.86         0.87         0.85        399
0.8671679197994987
confusion matrix [[ 12  21  11]
 [  2 182  7]
 [  2  10 152]]
auc 0.15778544099879177

```

Figure 7.78: *Classification report for RF.***e. GB - Accuracy: 82.46 %. Area under curve: 0.1950**

```

                precision    recall  f1-score   support

     1.0           0.52         0.29         0.37         48
     2.0           0.80         0.90         0.85        184
     6.0           0.90         0.90         0.90        167

 avg / total         0.81         0.82         0.81        399
0.8245614035087719
confusion matrix [[ 14  27  7]
 [  9 165  10]
 [  4  13 150]]
auc 0.19504550050556113

```

Figure 7.79: *Classification report for GB.*

f. K-NN - Accuracy: 79.20 %. Area under curve: 0.2355

```

                precision    recall  f1-score   support

     1.0         0.54         0.27         0.36         51
     2.0         0.76         0.89         0.82        184
     6.0         0.87         0.85         0.86        164

 avg / total         0.78         0.79         0.78        399
0.7919799498746867
confusion matrix [[ 14  31  6]
 [ 7 163 14]
 [ 5  20 139]]
auc 0.2354651162790698

```

Figure 7.80: *Classification report for K-NN.***g. TPOT - RandomForestClassifier - Accuracy: 85.21 %. Area under curve: 0.1819**

```

                precision    recall  f1-score   support

     1.0         0.70         0.32         0.44         50
     2.0         0.84         0.93         0.88        192
     6.0         0.90         0.93         0.91        157

 avg / total         0.84         0.85         0.84        399
0.8521303258145363
confusion matrix [[ 16  25  9]
 [ 6 178  8]
 [ 1  10 146]]
auc 0.18186392914653787

```

Figure 7.81: *Classification report for TPOT using an RandomForestClassifier.*

```

Best pipeline: RandomForestClassifier(input_matrix, bootstrap=True, criterion=gini, max_features=0.35000000000000003,
min_samples_leaf=1, min_samples_split=7, n_estimators=100)
Accuracy is 85.21303258145363%

```

Figure 7.82: *Recommended pipeline for Case 10 is a RandomForestClassifier.***Evaluation of best models from Cases 6-10:**

Considering the results from the models that have been developed for classification of six different geological formations at the Volve field, it can immediately be observed that the accuracy is significantly lower than in the models developed using laboratory data. A possible explanation to this is that the data from Volve has been taken from different wells, and at different depths. It is furthermore probable that different equipment has been used, which combined with the other factors mentioned likely has an effect on the results of the models developed. A 10 - 13 % increase in accuracy for all models can be observed when the number of geological formations (that the models have been trained on) is reduced from six to three. If we first consider the difference between training models with and without outliers present in the training data, all models improve slightly (a few percentage higher) when IQR method has been used.

Considering the tree classifiers, random forest and gradient boosting continue to perform the best (out of the different models), while decision tree scores considerably worse than expected. An explanation to why tree algorithms appear to be the best performing are the characteristics shown in Figure 3.11.

Considering the support vector machine model, poor performance is observed for six formations, which further strenghtens the characteristic that this model is less robust to outliers and noisy data. Even if an increase in performance can be noted when the number of geological formations get reduced, SVM continues to score the lowest out of the seven models considered. Again, the same appears to apply for the MLP model.

Considering the K-NN model, performance for six geological formations is moderately high when compared to the highest scoring tree models and the lowest scoring SVM and MLP models. When the number of formations have been reduced to three, it can be observed that almost 80 % accuracy is achieved when using only the six highest scoring features.

Considering all results, it continues to be our recommendation to use tree classifiers for rock formation classification. When outliers get removed, the accuracy increases slightly, suggesting that for field data it is a good approach to make use of the IQR (or similar methods) to remove outliers. The results could possibly have improved even more, if outliers had been removed manually. Furthermore, it can be observed that when using the algorithm to identify the highest scoring features, and manually selecting those that are considered the most applicable, the number of features to train and classify formations can be reduced from to six without noting a drop in the performance.

7.2.3 Laboratory rig operations

The processed dataset contains 22762 rows \times 25 feature columns (+ 1 label column), and the data represents three rig operations, i.e. RIH, POOH and ROnB. The data has been sampled at 9600 Hz and downsampled to 96 Hz. The data has further been processed for missing or invalid data, normalized and then outliers have been removed. The features available in the dataset are: load cells (1/2/3), rpm feedback, motor torque feedback, weight on bit, rate of penetration (ROP), TF1, TF2, TF3, TF4, TF5, mechanical specific energy. Additional features that have been engineered over a 96 sample window (= 1 second at 96 Hz sampling rate when downsampled) are $ROP_{standarddeviation}$, $ROP_{average}$, ROP_{median} , $ROP_{maximum}$, $ROP_{minimum}$, $ROP_{average}$ divided by $ROP_{standarddeviation}$, ROP_{median} divided by $ROP_{standarddeviation}$ and finally the peak-to-peak $ROP_{maximum} - ROP_{minimum}$ (see chapter 6). For case 11 through 13, the following labels are used for naming the samples: 1: POOH, 2: RIH and 3: ROnB.

From running the ExtraTreesClassifier feature importance algorithm, the following result is obtained:

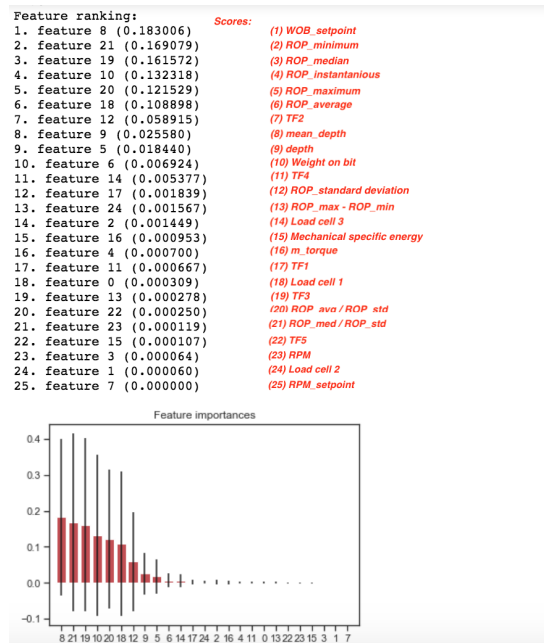


Figure 7.83: Laboratory operation classification feature importance using an ExtraTreesClassifier algorithm.

Operating setpoints for WOB and RPM should get removed together with depth and mean-depth measurements (same as for Laboratory formation classification).

The highest scoring features then become:

Score:	1	2	3	4	5	6
Feature	ROP _{min}	ROP _{med}	ROP _{instant}	ROP _{max}	ROP _{avg}	TF2

From looking at Figure 7.83, a remaining features that can be used in Case 12 for unsupervised classification using natural features is weight on bit (together with the ROP).

Model Parameter Tuning

K-Nearest Neighbor (K-NN): optimal parameter search is performed evaluating the following parameters on 100 % of the dataset and all 21 features.

```
1 params_to_test = {'n_neighbors':[3,5,7,9,11,13,15], ...
                   'weights':['distance', 'uniform']}
```

The random state, grid-search cross-validation generator (cv), and scoring is shown below:

```
1 KNN_model = KNeighborsClassifier()
2 grid_search = GridSearchCV(KNN_model, param_grid=params_to_test, cv=3, ...
                             scoring='f1_macro', n_jobs=4)
```

The best model parameters are:

```
1 KNeighborsClassifier(algorithm='auto', leaf_size=30, metric='minkowski', ...
                       metric_params=None, n_jobs=1, n_neighbors=3, p=2, weights='distance')
```

Case 11

<i>Model variable input(s):</i>	z1, z2, z3, RPM, m_torque, WOB, ROP, TF1, TF2, TF3, TF4, TF5, MSE, ROP _{std} , ROP _{avg} , ROP _{med} , ROP _{max} , ROP _{min} , ROP _{avg/std.} , ROP _{med/std.} , ROP _{max-min}
<i>Sampling rate:</i>	96 Hz
<i>Labels:</i>	1: POOH, 2: RIH, 3: ROnB

K-NN - Accuracy: 100 %. Area under curve: 0.2613


```

                precision    recall  f1-score   support

     1.0         1.00      1.00      1.00         943
     2.0         1.00      1.00      1.00         944
     3.0         1.00      1.00      1.00        2666

 avg / total         1.00      1.00      1.00        4553
1.0
confusion matrix [[ 943    0    0]
 [   0  944    0]
 [   0   0 2666]]
auc 0.2612912164034359

```

Figure 7.84: Classification report for K-NN.

Case 12: ROP and WOB

<i>Model variable</i> <i>input(s):</i>	ROP, WOB
<i>Sampling rate:</i>	96 Hz
<i>Labels:</i>	1: POOH, 2: RIH, 3: ROnB

K-Means - Adjusted Rand Index Score: 1.0000

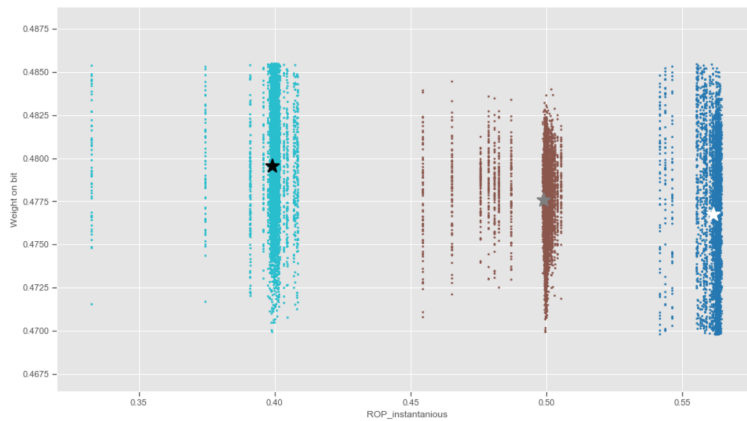


Figure 7.85: Clusters generated by K-Means algorithm for $ROP_{instantaneous}$ versus WOB when $n_{clusters} = 3$.

DBSCAN - Adjusted Rand Index Score: 0.9980

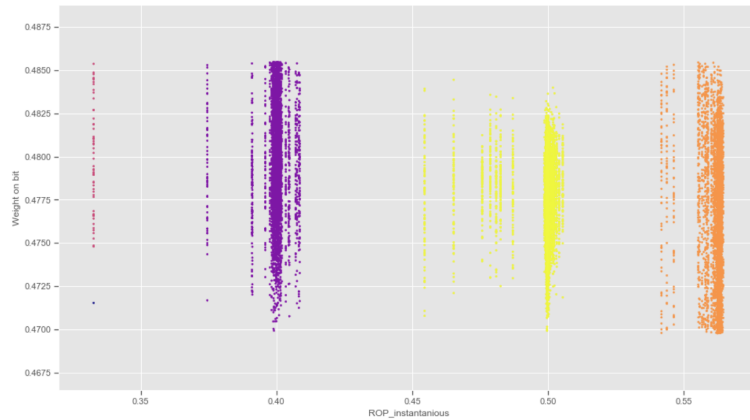


Figure 7.86: Clusters generated by DBSCAN algorithm for $ROP_{instantaneous}$ versus WOB when $\epsilon=0.600$ and $min_{samples} = 2$.

Case 13: ROP_{min} and ROP_{med}

<i>Model variable input(s):</i>	ROP_{min}, ROP_{med}
<i>Sampling rate:</i>	96 Hz
<i>Labels:</i>	1: POOH, 2: RIH, 3: ROnB

K-Means - Adjusted Rand Index Score: 0.9970

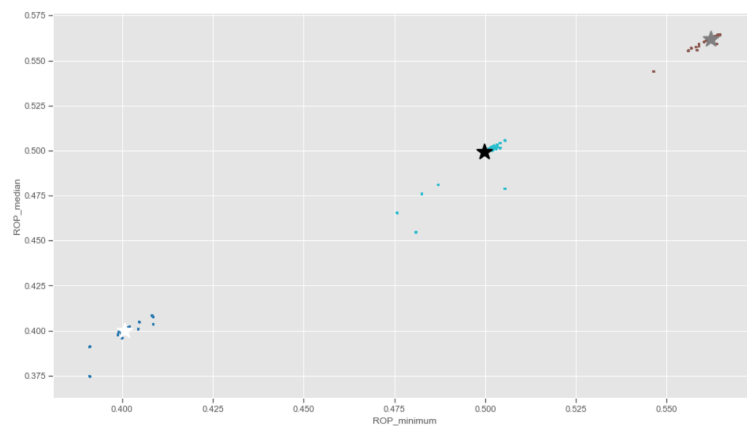


Figure 7.87: Clusters generated by K-Means algorithm for $ROP_{instantaneous}$ versus WOB when $n_{clusters} = 3$.

DBSCAN - Adjusted Rand Index Score: 0.9970

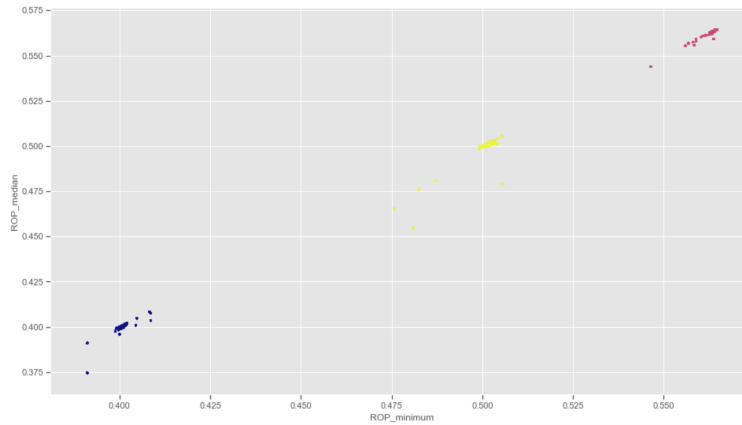


Figure 7.88: Clusters generated by DBSCAN algorithm for $ROP_{instantaneous}$ versus WOB when $\epsilon=0.600$ and $min_{samples} = 2$.

Evaluation of laboratory rig operations results:

Considering the results from cases 11 to 13, it appears that high accuracy can be achieved using only a few select features being either natural or engineered. When the results from cases 12 and 13 get considered, the models appear to more easily be capable of separating the engineered features (representing the three rig activities) from each other.

7.2.4 Volve rig operations

The processed dataset contains 9050 rows \times 17 feature columns (+ 1 label column), and the data represents three rig operations, i.e. RIH, POOH and drilling. The data has been sampled at 0.357 Hz (1 sample every 2.8 seconds), and has further been processed for missing or invalid data and then normalized. Outlier removal by IQR method can not be performed directly on the Volve rig operations dataset, as the IQR method would remove all observations from RIH and POOH operations, leaving only drilling data. The features that are available in the dataset are: bit- and surface RPM, depth of bit, rate of penetration, crown block position, torque, weight on bit, rate of crown block movement (over a 20 Hz sample window corresponding to 56 seconds at approximately 0.357 Hz sampling rate), TF1, TF3 and TF4. Additional features that have been engineered over a 10 sample window (= 28 seconds at 0.357 Hz sampling rate) are crown-block-position_{standarddeviation}, crown-block-position_{average}, crown-block-position_{median}, crown-block-position_{maximum}, crown-block-position_{minimum} and finally peak-to-peak (crown-block-position_{maximum} - crown-block-position_{minimum}). For the cases 14 through 16, the following labels have been used: 1: Drilling, 2: RIH and 3: POOH.

From running the ExtraTreesClassifier feature importance algorithm, the following feature score is obtained:

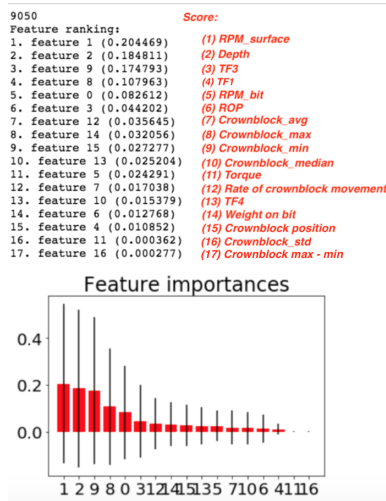


Figure 7.89: Volve rig operation classification feature importance using an ExtraTreesClassifier algorithm.

The depth (bit depth) feature column gets discarded for the same reason as stated

in earlier classification tasks. The highest scoring features are:

Score:	1	2	3	4	5	6
Feature	RPM _{surface}	TF3	TF2	RPM _{bit}	ROP	crownblock _{average}

Model Parameter Tuning

K-Nearest Neighbor (K-NN): optimal parameter search is performed evaluating the following parameters on 100 % of the dataset and all 16 features (except depth).

```
1 params_to_test = {'n_neighbors':[3,5,7,9,11,13,15], ...
                   'weights':['distance', 'uniform']}
```

The random state, grid-search cross-validation generator (cv), and scoring is shown below:

```
1 KNN_model = KNeighborsClassifier()
2 grid_search = GridSearchCV(KNN_model, param_grid=params_to_test, cv=3, ...
                             scoring='f1_macro', n_jobs=4)
```

The best model parameters are:

```
1 KNeighborsClassifier(algorithm='auto', leaf_size=30, metric='minkowski', ...
                       metric_params=None, n_jobs=1, n_neighbors=3, p=2, weights='distance')
```

Case 14

<i>Model variable input(s):</i>	RPM _{bit} , RPM _{surface} , ROP, Blockcomp-position, Torque, WOB, Rate-of-crown-block-movement, TF1, TF3, TF4, Rate-of-block _{std} , Rate-of-block _{avg} , Rate-of-block _{med} , Rate-of-block _{max} , Rate-of-block _{max-min}
<i>Sampling rate:</i>	0.357 Hz
<i>Labels:</i>	1: Drilling, 2: RIH, 3: POOH

K-NN - Accuracy: 100 %. Area under curve: 0.9832

```

                precision    recall  f1-score   support

     1.0         1.00         1.00         1.00        1526
     2.0         1.00         1.00         1.00         258
     3.0         1.00         1.00         1.00          26

 avg / total         1.00         1.00         1.00        1810
 1.0
 confusion matrix [[1526    0    0]
 [    0   258    0]
 [    0    0   26]]
 auc 0.9832474226804123

```

Figure 7.90: Classification report for K-NN.

Case 15: $RPM_{surface}$ and RPM_{bit}

<i>Model variable</i>	$RPM_{surface}, RPM_{bit}$
<i>input(s):</i>	
<i>Sampling rate:</i>	0.357 Hz
<i>Labels:</i>	1: Drilling, 2: RIH, 3: POOH

K-Means - Adjusted Rand Index Score: 0.9805

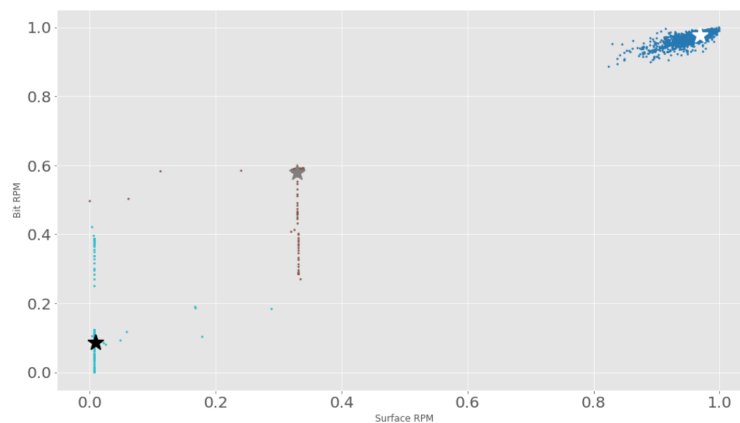


Figure 7.91: Clusters generated by K-Means algorithm for RPM_{bit} versus $RPM_{surface}$ when $n_{clusters} = 3$.

DBSCAN - Adjusted Rand Index Score: 0.9800

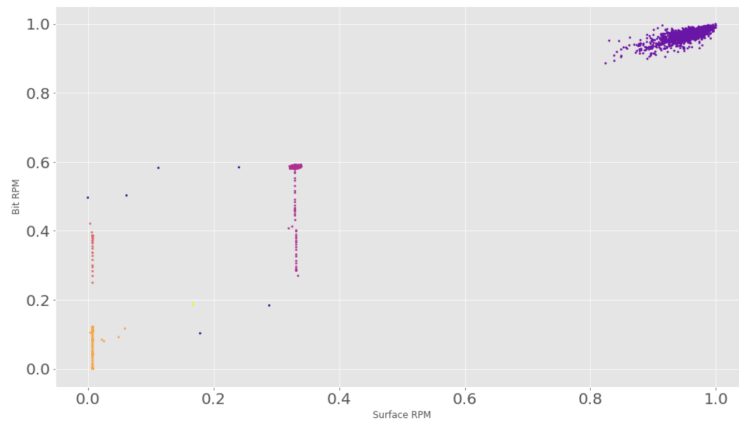


Figure 7.92: Clusters generated by DBSCAN algorithm for RPM_{bit} versus $RPM_{surface}$ when $\epsilon=0.200$ and $min_{samples} = 2$.

Case 16: TF1 and TF3

<i>Model variable</i>	TF1, TF3
<i>input(s):</i>	
<i>Sampling rate:</i>	0.357 Hz
<i>Labels:</i>	1: Drilling, 2: RIH, 3: POOH

K-Means - Adjusted Rand Index Score: 0.9810

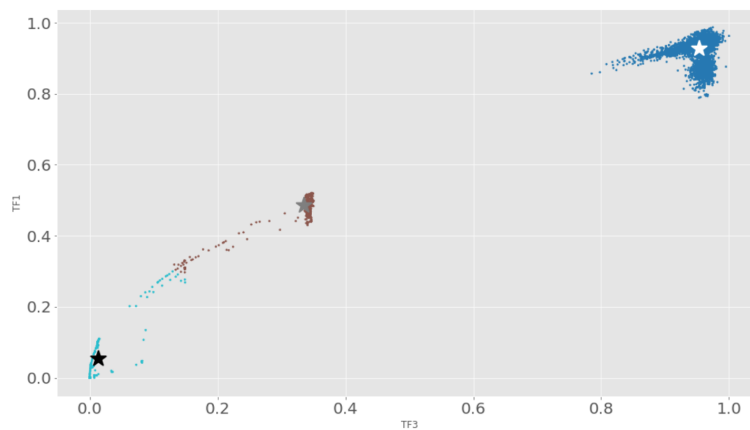


Figure 7.93: Clusters generated by K-Means algorithm for $TF3$ vs $TF1$ when $n_{clusters} = 3$.

DBSCAN - Adjusted Rand Index Score: 0.9825

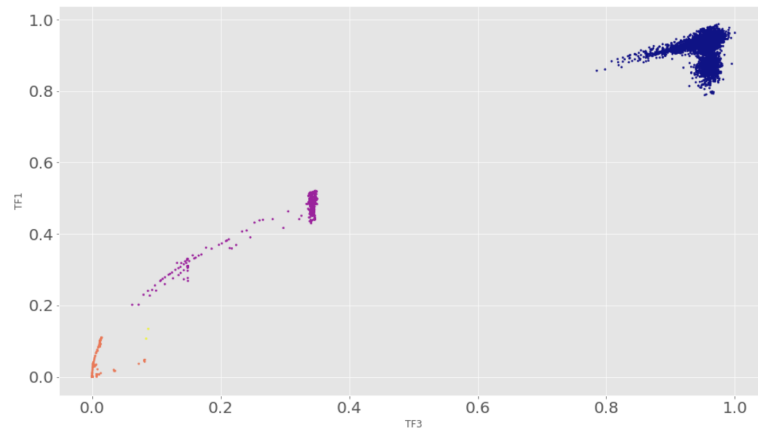


Figure 7.94: Clusters generated by DBSCAN algorithm for $TF3$ versus $TF1$ when $\epsilon=0.200$ and $\min_{samples} = 2$.

Evaluation of laboratory rig operations results:

Considering the results from cases 14 to 16, it appears that high accuracy can also be achieved using only a few select features from the Volve field data. When the results from cases 15 and 16 get considered, it appears that the natural features for surface and bit RPM perform better in terms of separating the three rig activities apart, however one should be careful to rely on only the RPM feature since this only depends on what setpoint was selected at the time the data was collected.

7.2.5 Laboratory pressure cases

The processed dataset contains 592 rows \times 9 feature columns (+ 1 label column), and the data represents three pressure cases, i.e. Normal pressure, Leak and Overpressure. The data has been sampled at 9600 Hz and downsampled to 96 Hz. Furthermore, the data has been processed for missing or invalid data, and then normalized. Outlier removal is not performed in order to keep observations containing peak values (leak and overpressure). The features available in the dataset are: mud system pressure and weight on bit. Additional features that have been engineered over a 10 sample window (= 0.1 second at 96 Hz sampling rate when downsampled from 9600 Hz to 96 Hz) are $Pressure_{standarddeviation}$, $Pressure_{average}$, $Pressure_{median}$, $Pressure_{maximum}$, $Pressure_{minimum}$, $Pressure_{average}$ divided by $Pressure_{standarddeviation}$, $Pressure_{median}$ divided by $Pressure_{standarddeviation}$ and peak-to-peak $Pressure_{maximum} - Pressure_{minimum}$. In case 17 through 19, the following labels are used: 1: Normal Pressure, 2: Leak (losses) and 3: Overpressure (plugged nozzles)

From running the ExtraTreesClassifier algorithm, the following feature score is obtained:

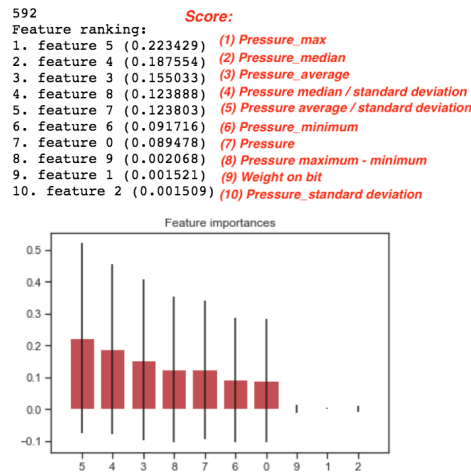


Figure 7.95: Pressure cases feature importance using an ExtraTreesClassifier algorithm.

Score:	1	2	3	4	5	6
Feature	$P_{maximum}$	P_{median}	$P_{average}$	$P_{med/std.}$	$P_{avg/std.}$	$P_{minimum}$

Model Parameter Tuning

K-Nearest Neighbor (K-NN): optimal parameter search is performed evaluating the following parameters on 100 % of the dataset and all 16 features.

```
1 params_to_test = {'n_neighbors':[3,5,7,9,11,13,15], 'algorithm': ['auto', ...
    'ball_tree', 'kd_tree', 'brute'], 'weights':['distance', 'uniform']}
```

The random state, grid-search cross-validation generator (cv), and scoring is shown below:

```
1 KNN_model = KNeighborsClassifier()
2 grid_search = GridSearchCV(KNN_model, param_grid=params_to_test, cv=3, ...
    scoring='f1_macro', n_jobs=4)
```

The best model parameters are:

```
1 KNeighborsClassifier(algorithm='auto', leaf_size=30, metric='minkowski', ...
    metric_params=None, n_jobs=1, n_neighbors=3, p=2, weights='distance')
```

Case 17:

<i>Model variable input(s):</i>	WBM_Pressure, WOB, Pressure _{std} , Pressure _{avg} , Pressure _{med} , Pressure _{max} , Pressure _{min} , Pressure _{avg/std.} , Pressure _{med/std.} , Pressure _{max-min}
<i>Sampling rate:</i>	96 Hz
<i>Labels:</i>	1: Normal Pressure, 2: Leak (losses), 3: Overpressure (plugged nozzles)

K-NN - Accuracy: 99.16 %. Area under curve: 0.4254

```

          precision    recall  f1-score   support

     1.0         1.00      1.00      1.00         28
     2.0         0.98      1.00      0.99         52
     3.0         1.00      0.97      0.99         39

 avg / total         0.99      0.99      0.99        119
0.9915966386554622
confusion matrix [[28  0  0]
 [ 0 52  0]
 [ 0  1 38]]
auc 0.4253731343283582
```

Figure 7.96: Classification report for K-NN.

Case 18: Pressure and WOB

<i>Model variable input(s):</i>	WBM_Pressure, WOB
<i>Sampling rate:</i>	96 Hz
<i>Labels:</i>	1: Normal Pressure, 2: Leak (losses), 3: Overpressure (plugged nozzles)

K-Means - Adjusted Rand Index Score: 0.9951

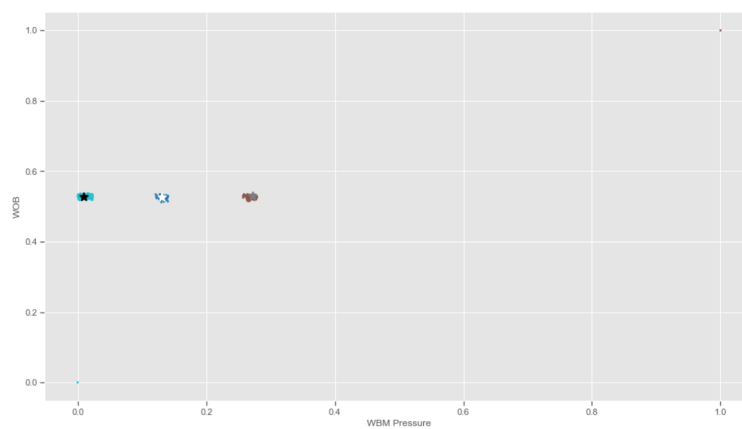


Figure 7.97: Clusters generated by K-Means algorithm for Pressure vs WOB when $n_{clusters} = 3$.

DBSCAN - Adjusted Rand Index Score: 0.9955

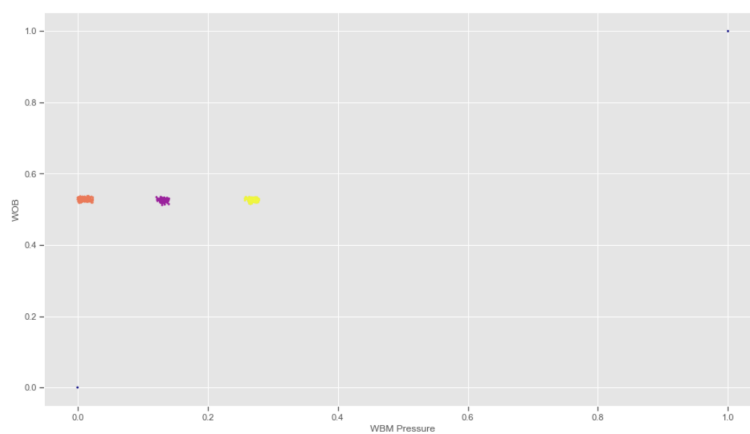


Figure 7.98: Clusters generated by DBSCAN algorithm for Pressure versus WOB when $\epsilon=0.200$ and $\min_{samples} = 2$.

Case 19: P_{max} and P_{med}

<i>Model variable input(s):</i>	P_{max}, P_{med}
<i>Sampling rate:</i>	96 Hz
<i>Labels:</i>	1: Normal Pressure, 2: Leak (losses), 3: Overpressure (plugged nozzles)

K-Means - Adjusted Rand Index Score: 0.9956

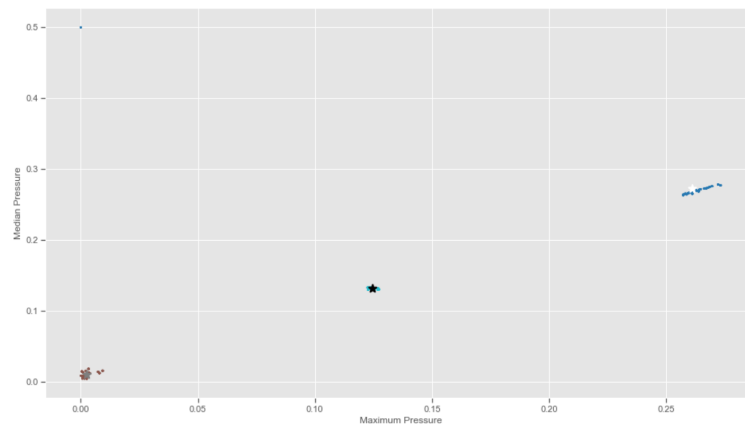


Figure 7.99: Clusters generated by K-Means algorithm for $Pressure_{maximum}$ vs $Pressure_{median}$ when $n_{clusters} = 3$.

DBSCAN - Adjusted Rand Index Score: 0.9955

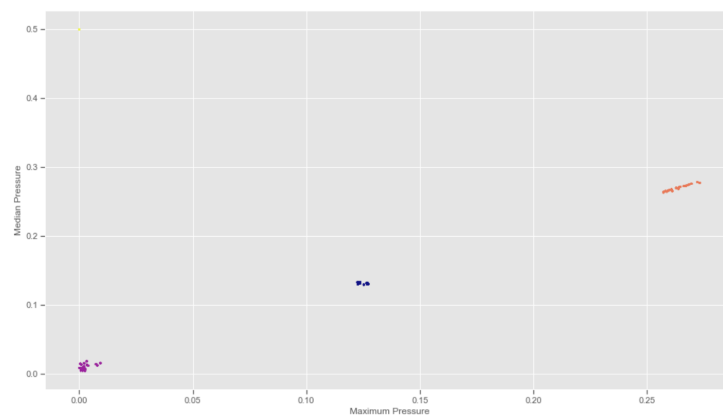


Figure 7.100: Clusters generated by DBSCAN algorithm for $Pressure_{maximum}$ vs $Pressure_{median}$ when $\epsilon=0.200$ and $\min_{samples} = 2$.

Evaluation of pressure cases:

From the results obtained in cases 17 to 19, it can be observed that the models can easily distinguish between a normal pressure case, a leak and overpressure.

7.2.6 Laboratory vibration cases (surface data)

The processed dataset contains 5822 rows \times 16 feature columns (+ 1 label column), and the data represents two vibration cases using surface sensors, i.e. normal vibrations (gathered at a low rotational speed with the top drive) and heavy vibrations (gathered at high RPM with top drive). The data was sampled at 9600 Hz and then downsampled to 96 Hz. The data has further been processed for missing or invalid data, and then normalized. IQR is not performed. The features available in the dataset are: load cells (1/2/3), weight on bit, rpm feedback and motor torque feedback. Features that have been engineered over a 10 sample window (= 0.1 second at 96 Hz sampling rate when downsampled) are $WOB_{standarddeviation}$, $WOB_{average}$, WOB_{median} , $WOB_{maximum}$, $WOB_{minimum}$, $WOB_{average}$ divided by $WOB_{standarddeviation}$, WOB_{median} divided by standard deviation and peak-to-peak $WOB_{maximum} - WOB_{minimum}$. In case 20 through 22, the following labels are used: 1: Normal Vibrations and 2: Heavy Vibrations.

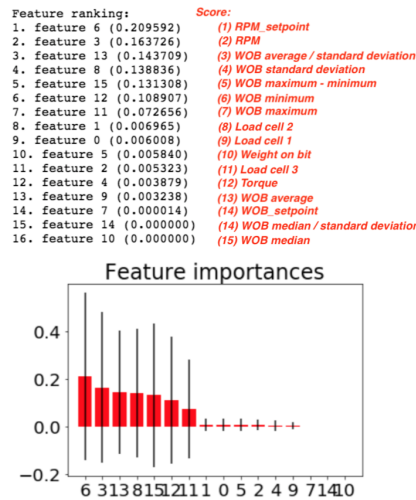


Figure 7.101: *Vibration classification feature importance using an Extra-TreesClassifier algorithm.*

Operating setpoints for WOB and RPM get removed together with RPM feedback, since the RPM is varied to collect data. The highest scoring features become:

Score:	1	2	3	4	5	6
Feature	$WOB_{avg/std.}$	$WOB_{std.}$	$WOB_{max-min}$	WOB_{min}	WOB_{max}	LC 2

Model Parameter Tuning

K-Nearest Neighbor (K-NN): optimal parameter search is performed evaluating the following parameters on 100 % of the dataset and all 13 features.

```
1 params_to_test = {'n_neighbors':[3,5,7,9,11,13,15], 'algorithm': ['auto', ...
    'ball_tree', 'kd_tree', 'brute'], 'weights':['distance', 'uniform']}
```

The random state, grid-search cross-validation generator (cv), and scoring is shown below:

```
1 KNN_model = KNeighborsClassifier()
2 grid_search = GridSearchCV(KNN_model, param_grid=params_to_test, cv=3, ...
    scoring='f1_macro', n_jobs=4)
```

The best model parameters are:

```
1 KNeighborsClassifier(algorithm='auto', leaf_size=30, metric='minkowski', ...
    metric_params=None, n_jobs=1, n_neighbors=3, p=2, weights='distance')
```

Case 20:

<i>Model variable input(s):</i>	z1, z2, z3, m_torque, WOB, WOB _{std} , WOB _{avg} , WOB _{med} , WOB _{max} , WOB _{min} , WOB _{avg/std.} , WOB _{med/std.} , WOB _{max-min}
<i>Sampling rate:</i>	96 Hz
<i>Labels:</i>	1: Normal Vibrations, 2: Heavy Vibrations

K-NN - Accuracy: 100 %. Area under curve: 1.0000

```

                precision    recall  f1-score   support

     1.0         1.00         1.00         1.00         522
     2.0         1.00         1.00         1.00         643

 avg / total         1.00         1.00         1.00         1165

1.0
confusion matrix [[522  0]
 [ 0 643]]
auc 1.0
```

Figure 7.102: Classification report for K-NN.

Case 21: WOB and torque

<i>Model variable</i>	m_torque, WOB
<i>input(s):</i>	
<i>Sampling rate:</i>	96 Hz
<i>Labels:</i>	1: Normal Vibrations, 2: Heavy Vibrations

First, the natural features weight on bit and torque can be plotted to see whether the vibrations are separated or overlapping:

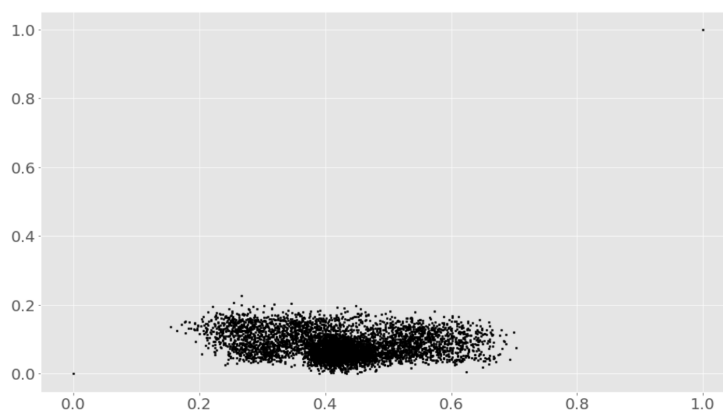


Figure 7.103: *Plot of WOB vs torque to see whether vibrations are separated or overlap.*

This suggests that by using natural features, normal and heavy vibrations can not be separated.

K-Means - Adjusted Rand Index Score: 0.1059

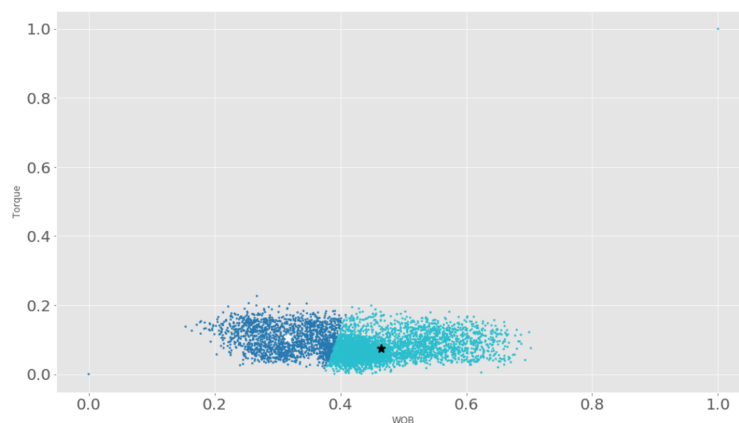


Figure 7.104: *Clusters generated by K-Means algorithm for WOB vs torque when $n_{clusters} = 2$.*

DBSCAN - Adjusted Rand Index Score: 0.0000

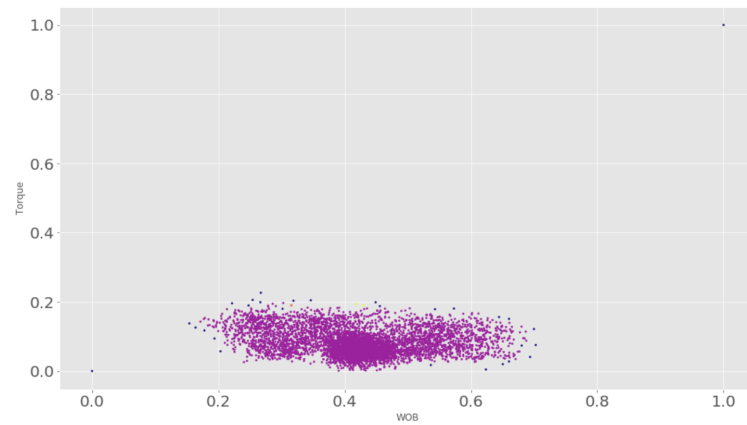


Figure 7.105: Clusters generated by DBSCAN algorithm for WOB and torque when $\epsilon=0.200$ and $\min_{samples} = 2$.

Case 22: WOB_{max} and WOB_{avg/std}.

<i>Model variable input(s):</i>	WOB _{max} , WOB _{avg/std}
<i>Sampling rate:</i>	96 Hz
<i>Labels:</i>	1: Normal Vibrations, 2: Heavy Vibrations

K-Means - Adjusted Rand Index Score: 0.9931

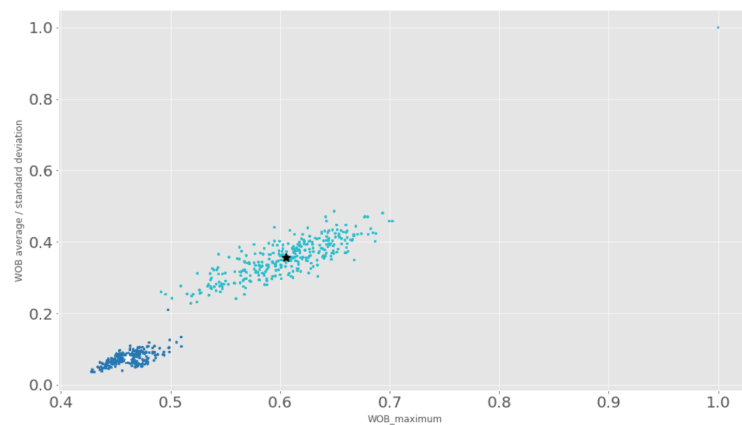


Figure 7.106: Clusters generated by K-Means algorithm for WOB_{maximum} vs WOB_{avg/std}. when $n_{clusters} = 3$.

DBSCAN - Adjusted Rand Index Score: 0.9954

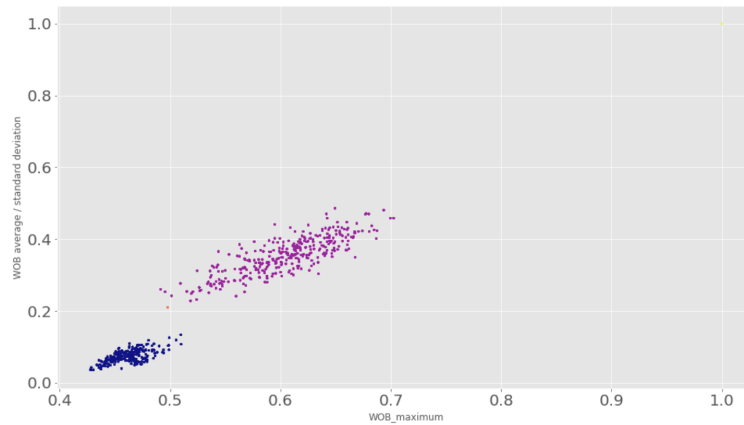


Figure 7.107: Clusters generated by DBSCAN algorithm for $WOB_{maximum}$ vs $WOB_{avg/std.}$ when $\epsilon=0.200$ and $min_{samples} = 2$.

Evaluation of vibrations using surface sensors:

Compared to the results from the pressure cases, it can be observed for cases 20 to 22 that engineered features must be created to distinguish low and moderately high vibrations apart. Despite no time to comprehensively evaluate which features are most applicable to classify vibrations, the method of developing features that consider the maximum and minimum value, the average observation and the standard deviation over a number of samples appear to give good results.

7.2.7 Laboratory vibration cases (downhole data) with downhole motor

The processed dataset contains 35602 rows \times 35 feature columns (+ 1 label column), and the data represents three vibration cases using a combination of downhole and surface sensors, i.e. idle rotation corresponding to low vibrations, moderate vibrations (gathered at 60 to 80 % solenoid valve opening) and high vibrations (gathered at 90 to 100 % valve opening). The data has been sampled at 60 Hz. The data has been processed for missing or invalid data, and normalized. IQR is not carried out. The features available are: load cells (1/2/3), weight on bit (surface sensors) and accelerometer (x/y/z), gyroscope (x/y/z), magnetometer (x/y/z) and temperature (downhole sensors). Engineered features, that have been created over a 60 sample window (= 1 second at 60 Hz sampling rate) are *accelerometer_{x/y/zmaximum}*, *accelerometer_{x/y/zminimum}*, *peak-to-peak accelerometer_{x/y/zmaximum-minimum}*, *gyroscope_{x/y/zmaximum}*, *gyroscope_{x/y/zminimum}*, *peak-to-peak gyroscope_{x/y/zmaximum-minimum}* and finally *gyroscope_{x/y/zstandarddeviation}*. In case 23 through 26, the following labels are used: 1: Low Vibrations, 2: Moderate Vibrations and 3: Heavy Vibrations.

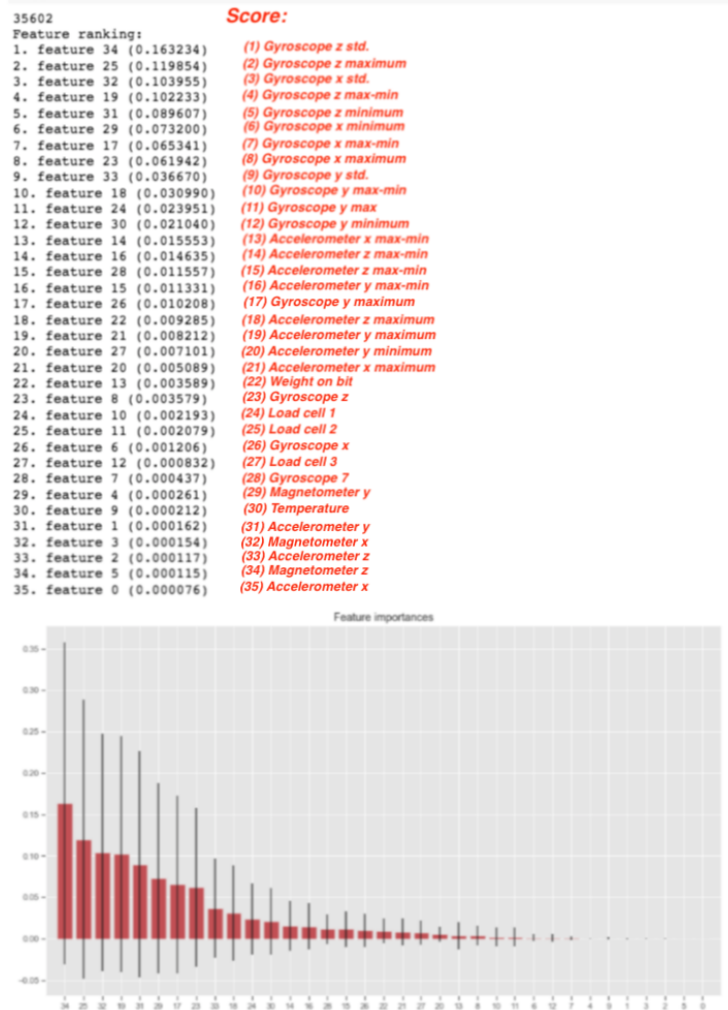


Figure 7.108: Downhole vibration classification feature importance using an ExtraTreesClassifier algorithm.

Operating setpoints for valve opening position setpoint (and the corresponding calculated bit RPM) and height is removed, for the same reasons as state for aforementioned classification tasks. The highest scoring features become:

Score:	1	2	3	4	5	6
Feature	Gyr _{zstd.}	Gyr _{zmax}	Gyr _{xstd.}	Gyr _{zmax-min}	Gyr _{zmin}	Gyr _{xmin}

Model Parameter Tuning

K-Nearest Neighbor (K-NN): optimal parameter search is performed evaluating the following parameters on 100 % of the dataset and all 34 features (except temperature, solenoid valve opening position, bit rpm and height).

```
1 params_to_test = {'n_neighbors':[3,5,7,9,11,13,15], 'algorithm': ['auto', ...
    'ball_tree', 'kd_tree', 'brute'], 'weights':['distance', 'uniform']}
```

The random state, grid-search cross-validation generator (cv), and scoring is shown below:

```
1 KNN_model = KNeighborsClassifier()
2 grid_search = GridSearchCV(KNN_model, param_grid=params_to_test, cv=3, ...
    scoring='f1_macro', n_jobs=4)
```

The best model parameters are:

```
1 KNeighborsClassifier(algorithm='auto', leaf_size=30, metric='minkowski', ...
    metric_params=None, n_jobs=1, n_neighbors=7, p=2, weights='distance')
```

Case 23:

<i>Model variable input(s):</i>	acc _x , acc _y , acc _z , mag _x , mag _y , mag _z , gyr _x , gyr _y , gyr _z , z1, z2, z3, WOB, acc _{xmaxmin} , acc _{ymaxmin} , acc _{zmaxmin} , gyr _{xmaxmin} , gyr _{ymaxmin} , gyr _{zmaxmin} , acc _{xmax} , acc _{ymax} , acc _{zmax} , gyr _{xmax} , gyr _{ymax} , gyr _{zmax} , acc _{xmin} , acc _{ymin} , acc _{zmin} , gyr _{xmin} , gyr _{ymin} , gyr _{zmin} , gyr _{Xstd} , gyr _{Ystd} , gyr _{Zstd}
<i>Sampling rate:</i>	60 Hz
<i>Labels:</i>	1: Low Vibrations, 2: Moderate Vibrations, 3: Heavy Vibrations

K-NN - Accuracy: 99.93 %. Area under curve: 0.0839

```

                precision    recall  f1-score   support

     1.0         1.00         1.00         1.00         424
     2.0         1.00         1.00         1.00        2019
     3.0         1.00         1.00         1.00        4678

 avg / total         1.00         1.00         1.00        7121
0.9992978514253616
confusion matrix [[ 424    0    0]
 [   0 2017    2]
 [   0    3 4675]]
auc 0.08385280058961622
```

Figure 7.109: Classification report for K-NN.

Case 24:

<i>Model variable input(s):</i>	gyr _{zmaxmin} , gyr _{zmax} , gyr _{xmin} , gyr _{zmin} , gyr _{xstd} , gyr _{zstd}
<i>Sampling rate:</i>	60 Hz
<i>Labels:</i>	1: Low Vibrations, 2: Moderate Vibrations, 3: Heavy Vibrations

K-NN - Accuracy: 99.94 %. Area under curve: 0.0835

```

                precision    recall  f1-score   support

     1.0         1.00         1.00         1.00         424
     2.0         1.00         1.00         1.00        2019
     3.0         1.00         1.00         1.00        4678

 avg / total         1.00         1.00         1.00        7121
0.9994382811402893
confusion matrix [[ 424    0    0]
 [    0 2019    0]
 [    0    4 4674]]
auc 0.08349666797334382

```

Figure 7.110: Classification report for K-NN.

Case 25:

<i>Model variable input(s):</i>	gyr _x , gyr _z
<i>Sampling rate:</i>	60 Hz
<i>Labels:</i>	1: Low Vibrations, 2: Moderate Vibrations, 3: Heavy Vibrations

K-Means - Adjusted Rand Index Score: 0.0037

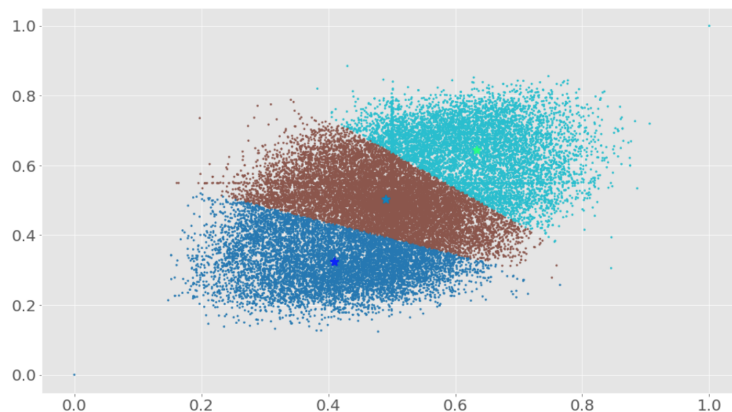


Figure 7.111: Clusters generated by K-Means algorithm for WOB and torque when $n_{clusters} = 2$.

DBSCAN - Adjusted Rand Index Score: -0.0002

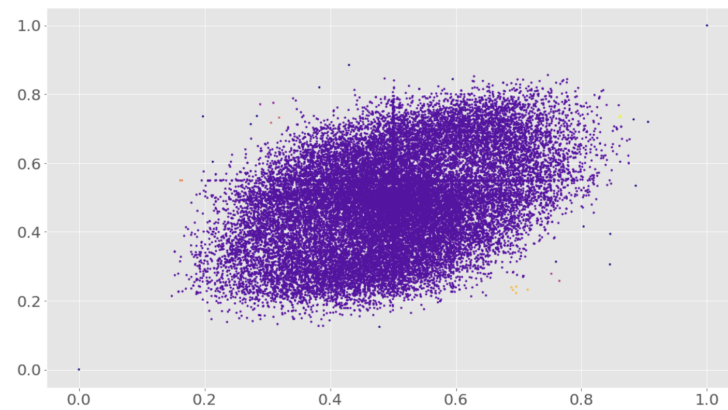


Figure 7.112: Clusters generated by DBSCAN algorithm for WOB and torque when $\epsilon=0.200$ and $\min_{\text{samples}} = 2$.

Case 26:

<i>Model variable input(s):</i>	gyfz _{std.} , gyfx _{std.}
<i>Sampling rate:</i>	60 Hz
<i>Labels:</i>	1: Low Vibrations, 2: Moderate Vibrations, 3: Heavy Vibrations

K-Means - Adjusted Rand Index Score: 0.4869

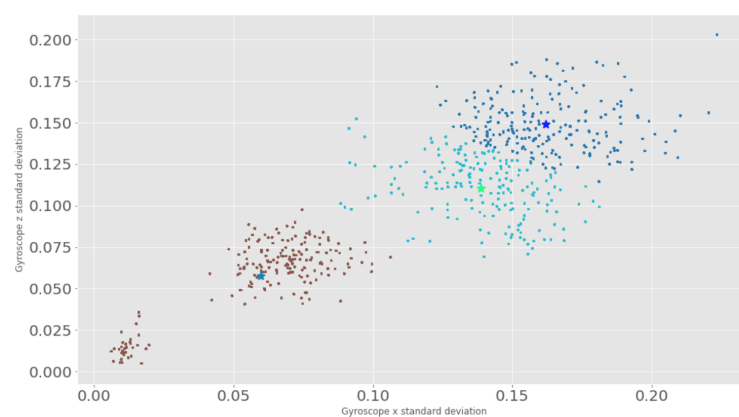


Figure 7.113: Clusters generated by K-Means algorithm for standard deviation in measurements of gyroscope x-axis and z-axis when $n_{\text{clusters}} = 2$.

DBSCAN - Adjusted Rand Index Score: 0.9502

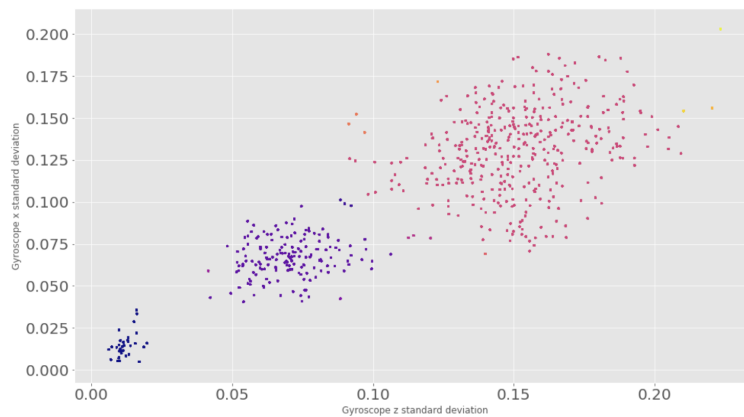


Figure 7.114: Clusters generated by DBSCAN algorithm for measurements for standard deviation in gyroscope x -axis and z -axis when $\epsilon=0.200$ and $\min_{\text{samples}} = 2$.

Evaluation of vibrations using downhole sensors:

It can be observed from the results in cases 23 and 24 that the same accuracy and AUC score is achieved when the number of features is reduced from 34 (containing both natural and engineered features from surface and downhole sensors) to six features that are all from downhole sensors. When downhole sensor measurements get used, it appears to be necessary to engineer features such as those considering the standard deviation and maximum and minimum amplitude over a number of samples, as got concluded when developing the models to distinguish between two vibration levels using only surface sensors.

7.2.8 Laboratory stuck-pipe cases

The processed dataset contains 117 rows \times 14 feature columns (+ 1 label column), and the data represents normal drilling and stuck pipe incident. Data has been downsampled from a sampling frequency of 1200 Hz to 96 Hz. The data has been processed for missing or invalid data, and then normalized. IQR is not performed. The features available in the dataset are: load cells (1/2/3), RPM, top drive torque and weight on bit. Additional features that have been engineered over a 5 sample window (= 0.052 seconds at 96 Hz sampling rate) are $\text{torque}_{\text{standarddeviation}}$, $\text{torque}_{\text{average}}$, $\text{torque}_{\text{median}}$, $\text{torque}_{\text{maximum}}$, $\text{torque}_{\text{minimum}}$, $\text{torque}_{\text{average}}$ divided by $\text{torque}_{\text{standarddeviation}}$, $\text{torque}_{\text{median}}$ divided by standard deviation and peak-to-peak torque $\text{torque}_{\text{maximum}} - \text{torque}_{\text{minimum}}$. The following labels were used for case 27 through 29: 1: Normal drilling and 2: Stuck pipe.

The highest scoring features from the ExtraTreesClassifier algorithm are:

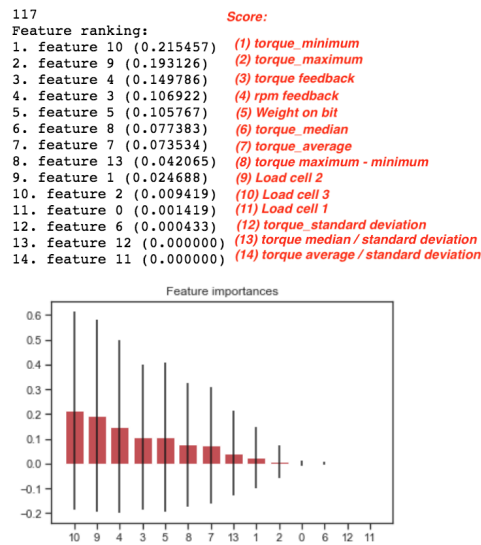


Figure 7.115: Stuck pipe classification feature importance using an ExtraTreesClassifier algorithm.

Score:	1	2	3	4	5	6
Feature	$\text{torque}_{\text{min}}$	$\text{torque}_{\text{max}}$	top drive torque	$\text{rpm}_{\text{feedback}}$	WOB	$\text{torque}_{\text{med}}$

Model Parameter Tuning

K-Nearest Neighbor (K-NN): optimal parameter search is performed evaluating the following parameters on 100 % of the dataset and all 14 features.


```
1 params_to_test = {'n_neighbors':[3,5,7,9,11,13,15], 'algorithm': ['auto', ...
    'ball_tree', 'kd_tree', 'brute'], 'weights':['distance', 'uniform']}
```

The random state, grid-search cross-validation generator (cv), and scoring is shown below:

```
1 KNN_model = KNeighborsClassifier()
2 grid_search = GridSearchCV(KNN_model, param_grid=params_to_test, cv=3, ...
    scoring='f1_macro', n_jobs=4)
```

The best model parameters are:

```
1 KNeighborsClassifier(algorithm='auto', leaf_size=30, metric='minkowski', ...
    metric_params=None, n_jobs=1, n_neighbors=7, p=2, weights='distance')
```

Case 27:

<i>Model variable input(s):</i>	z1, z2, z3, RPM, m_torque, WOB, torque _{std} , torque _{avg} , torque _{std} , torque _{max} , torque _{min} , torque _{avg/std} , torque _{med/std} , torque _{max-min}
<i>Sampling rate:</i>	96 Hz
<i>Labels:</i>	1: Normal drilling, 2: Stuck pipe

K-NN - Accuracy: 100 %. Area under curve: 1.0000

```

                precision    recall  f1-score   support

             1.0         1.00      1.00      1.00         22
             2.0         1.00      1.00      1.00          2

 avg / total         1.00      1.00      1.00         24

1.0
confusion matrix [[22  0]
 [ 0  2]]
auc 1.0
```

Figure 7.116: Classification report for K-NN.

Case 28: WOB and torque

<i>Model variable input(s):</i>	WOB, m_torque
<i>Sampling rate:</i>	96 Hz
<i>Labels:</i>	1: Normal drilling, 2: Stuck pipe

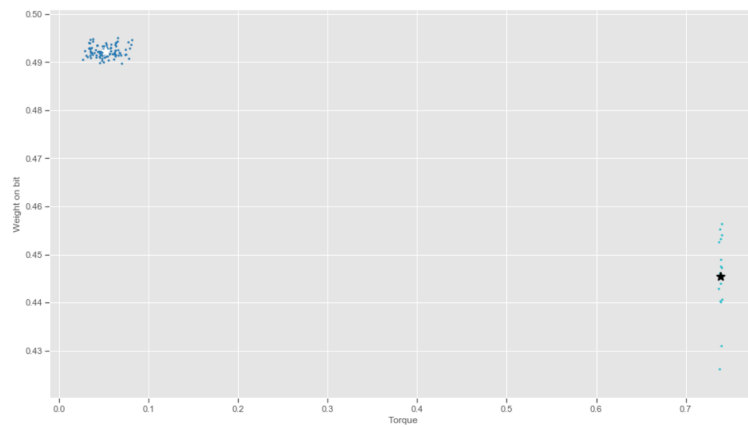
K-Means - Adjusted Rand Index Score: 1.0000

Figure 7.117: Clusters generated by K-Means algorithm for WOB and torque when $n_{clusters} = 2$.

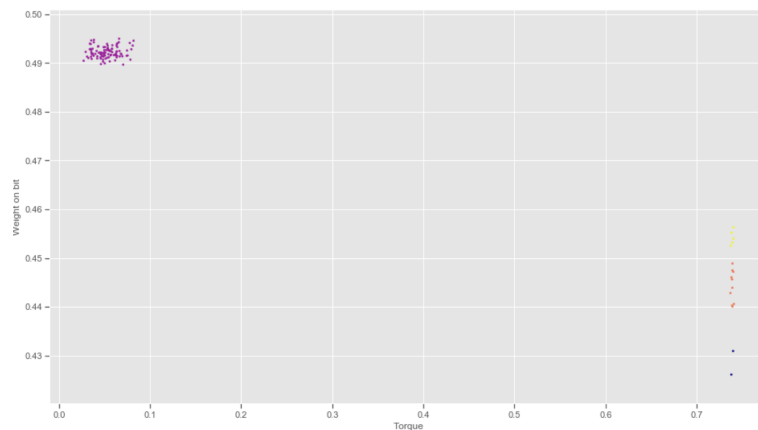
DBSCAN - Adjusted Rand Index Score: 0.9691

Figure 7.118: Clusters generated by DBSCAN algorithm for WOB and torque when $\epsilon=0.200$ and $\min_{samples} = 2$.

Case 29: torque_{maximum} and torque_{minimum}

<i>Model variable</i>	torque _{max} , torque _{avg/std} .
<i>input(s):</i>	
<i>Sampling rate:</i>	96 Hz
<i>Labels:</i>	1: Normal drilling, 2: Stuck pipe

K-Means - Adjusted Rand Index Score: 1.0000

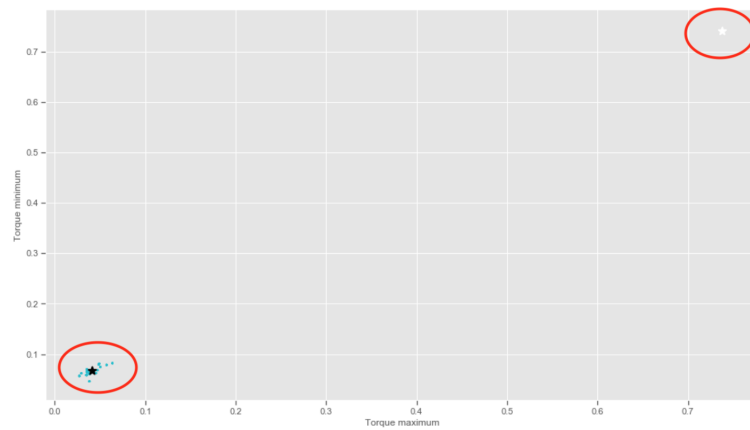


Figure 7.119: Clusters generated by K-Means algorithm for $torque_{maximum}$ and $torque_{minimum}$ when $n_{clusters} = 2$.

DBSCAN - Adjusted Rand Index Score: 1.0000

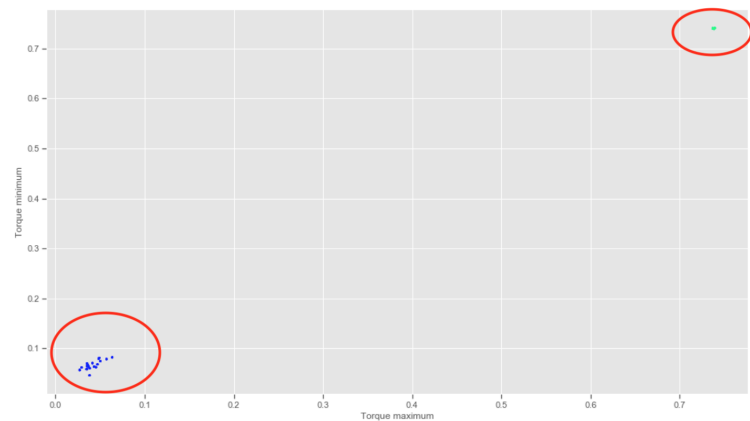


Figure 7.120: Clusters generated by DBSCAN algorithm for $torque_{maximum}$ and $torque_{minimum}$ when $\epsilon=0.200$ and $min_{samples} = 2$.

Evaluation of stuck pipe cases:

Considering the stuck pipe cases 27 to 29, both natural and engineered features appear to be useful to distinguish between a rotating and stuck pipe. It can however be noted that torque and RPM feedback have been considered the most important features, and that these features alone should be more than capable detecting a stuck pipe.

7.2.9 Laboratory twist off cases

The processed dataset contains 659 rows \times 14 feature columns (+ 1 label column), and the data represents normal drilling and twist off incident. Data has been downsampled from a sampling frequency of 9600 Hz to 96 Hz. The data has furthermore been processed for missing or invalid data, and is then normalized. IQR is not performed. The features available in the dataset are: load cells (1/2/3), RPM, motor torque and weight on bit. Additional features that have been engineered over a 5 sample window (=0.052 seconds at 96 Hz sampling rate) are $\text{torque}_{\text{standarddeviation}}$, $\text{torque}_{\text{average}}$, $\text{torque}_{\text{median}}$, $\text{torque}_{\text{maximum}}$, $\text{torque}_{\text{minimum}}$, $\text{torque}_{\text{average}}$ divided by $\text{torque}_{\text{standarddeviation}}$, $\text{torque}_{\text{median}}$ divided by standard deviation and peak-to-peak torque $\text{torque}_{\text{maximum}} - \text{torque}_{\text{minimum}}$. For case 30 through 32, the following labels are used: 1: No Twist Off and 2: Twist Off.

The highest scoring features from the ExtraTreesClassifier algorithm are:

```

659
Feature ranking:
1. feature 3 (0.323606) (1) RPM
2. feature 10 (0.125028) (2) Torque_mininum
3. feature 7 (0.108947) (3) Torque_average
4. feature 8 (0.087595) (4) Torque_median
5. feature 11 (0.083217) (5) Torque average / standard deviation
6. feature 12 (0.079446) (6) Torque median / standard deviation
7. feature 6 (0.057685) (7) Torque_standard deviation
8. feature 13 (0.046904) (8) Torque_maximum - minimum
9. feature 9 (0.040193) (9) Torque_maximum
10. feature 4 (0.024590) (10) Torque
11. feature 0 (0.007363) (11) Load cell 1
12. feature 1 (0.007276) (12) Load cell 2
13. feature 5 (0.005011) (13) WOB
14. feature 2 (0.003139) (14) Load cell 3

```

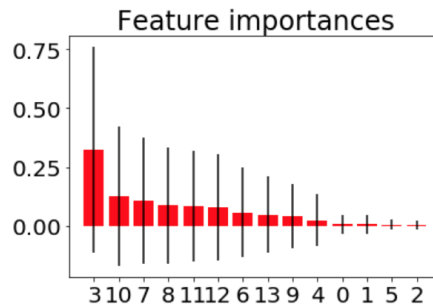


Figure 7.121: Twist off of drill pipe classification feature importance using an ExtraTreesClassifier algorithm.

Score:	1	2	3	4	5	6
Feature	RPM	$\text{torque}_{\text{min}}$	$\text{torque}_{\text{avg}}$	$\text{torque}_{\text{med}}$	$\text{torque}_{\text{avg/std.}}$	$\text{torque}_{\text{med/std.}}$

Model Parameter Tuning

K-Nearest Neighbor (K-NN): optimal parameter search is performed evaluating the following parameters on 100 % of the dataset and all 14 features.

```
1 params_to_test = {'n_neighbors':[3,5,7,9,11,13,15], 'algorithm': ['auto', ...
    'ball_tree', 'kd_tree', 'brute'], 'weights':['distance', 'uniform']}
```

The random state, grid-search cross-validation generator (cv), and scoring is shown below:

```
1 KNN_model = KNeighborsClassifier()
2 grid_search = GridSearchCV(KNN_model, param_grid=params_to_test, cv=3, ...
    scoring='f1_macro', n_jobs=4)
```

The best model parameters are:

```
1 KNeighborsClassifier(algorithm='auto', leaf_size=30, metric='minkowski', ...
    metric_params=None, n_jobs=1, n_neighbors=3, p=2, weights='distance')
```

Case 30

<i>Model variable input(s):</i>	z1, z2, z3, RPM, m_torque, WOB, torque _{std} , torque _{avg} , torque _{std} , torque _{max} , torque _{min} , torque _{avg/std.} , torque _{med/std.} , torque _{max-min}
<i>Sampling rate:</i>	96 Hz
<i>Labels:</i>	1: No Twist Off, 2: Twist Off

K-NN - Accuracy: 100 %. Area under curve: 1.0000

```

                precision    recall  f1-score   support

     1.0         1.00         1.00         1.00         117
     2.0         1.00         1.00         1.00          15

 avg / total         1.00         1.00         1.00         132
1.0
confusion matrix [[117  0]
 [ 0  15]]
auc 1.0
```

Figure 7.122: Classification report for K-NN.

Case 31: RPM and torque

<i>Model variable input(s):</i>	RPM, m_torque
<i>Sampling rate:</i>	96 Hz
<i>Labels:</i>	1: No Twist Off, 2: Twist Off

K-Means - Adjusted Rand Index Score: 1.0000

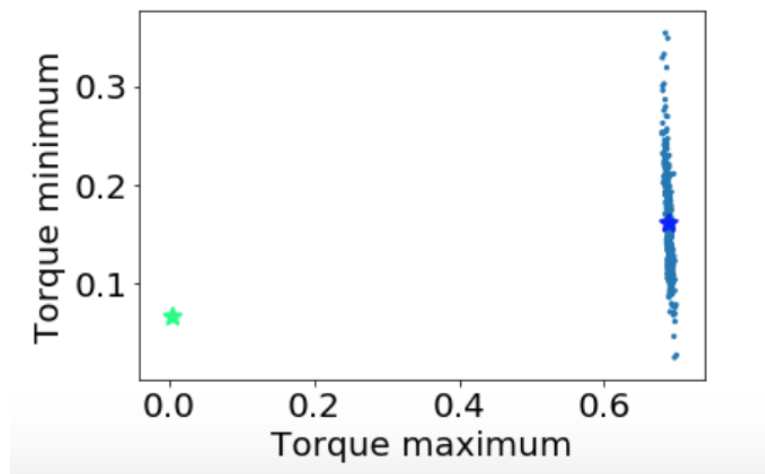


Figure 7.123: Clusters generated by K-Means algorithm for RPM and torque when $n_{clusters} = 2$.

DBSCAN - Adjusted Rand Index Score: 0.9476

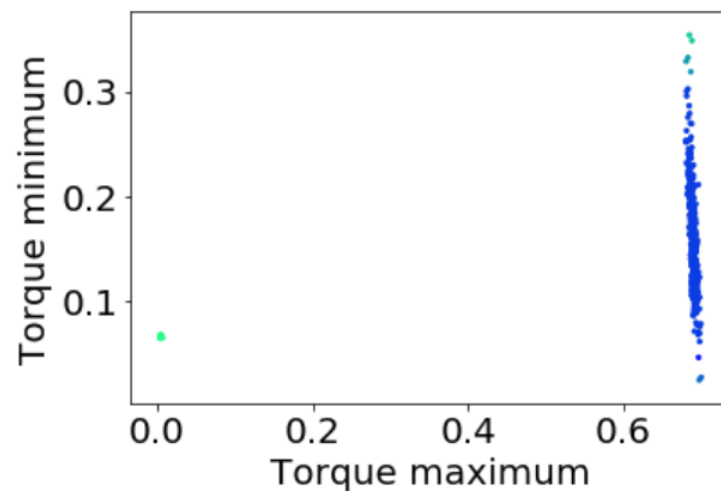


Figure 7.124: Clusters generated by DBSCAN algorithm for RPM and torque when $\epsilon=0.200$ and $\min_{samples} = 2$.

Case 32: torque_{average} and torque_{avg/std.}

<i>Model variable</i>	torque _{average} , torque _{avg/std.}
<i>input(s):</i>	
<i>Sampling rate:</i>	96 Hz
<i>Labels:</i>	1: No Twist Off, 2: Twist Off

K-Means - Adjusted Rand Index Score: 0.9191

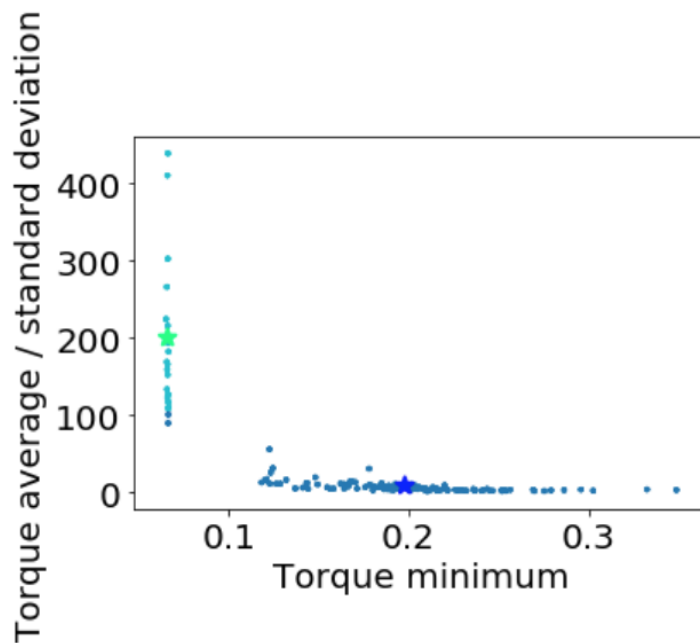


Figure 7.125: Clusters generated by K-Means algorithm for torque_{average} and torque_{avg/std.} when $n_{clusters} = 2$.

DBSCAN - Adjusted Rand Index Score: 0.9893

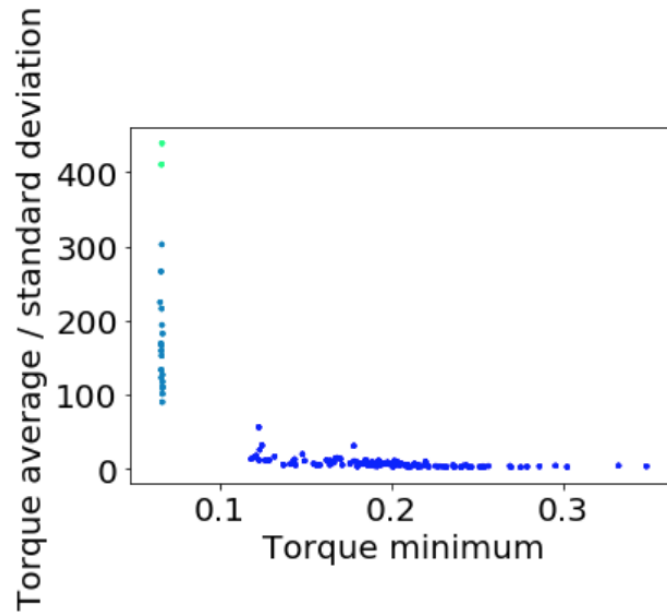


Figure 7.126: Clusters generated by DBSCAN algorithm for $torque_{average}$ and $torque_{avg/std}$. when $\epsilon=0.900$ and $min_{samples} = 2$.

Evaluation of twist off cases:

Considering the twist off cases 30 to 32, both natural and engineered features appear to be capable of detecting a twist off. As was the case for stuck pipe detection, measurements for RPM feedback, torque and WOB should be enough to detect if a twist off has occurred.

7.3 Summary of preliminary clasifications results

Summary of preliminary clasifications results									
Case	MLP	DT	SVM	RF	GB	K-NN	TPOT	K-Means	DBSCAN
1.	94.31%	99.67%	89.88%	98.52%	99.23%	85.62%	98.09%	-	-
2.	95.53%	99.66%	89.66%	98.50%	99.26%	85.37%	98.29%	-	-
3.	93.10%	99.71%	85.12%	98.68%	99.18%	92.34%	98.02%	-	-
4.	99.36%	99.93%	98.83%	99.53%	99.85%	89.92%	99.32%	-	-
5.	98.60%	99.94%	97.50%	99.66%	99.93%	97.74%	99.83%	-	-
6.	59.86%	57.36%	54.03%	71.39%	68.89%	67.22%	69.86%	-	-
7.	62.96%	62.31%	62.74%	72.38%	69.81%	68.95%	71.09%	-	-
8.	61.67%	59.96%	62.10%	71.95%	70.02%	66.38%	71.73%	-	-
9.	76.94%	78.20%	75.94%	83.46%	79.70%	78.95%	84.21%	-	-
10.	76.94%	78.85%	74.94%	86.72%	82.46%	79.20%	85.21%	-	-
11.	-	-	-	-	-	100%	-	-	-

12.	-	-	-	-	-	-	-	ARI: 1.0000	ARI: 0.9989
13.	-	-	-	-	-	-	-	ARI: 0.9970	ARI: 0.9970
14.	-	-	-	-	-	100%	-	-	-
15.	-	-	-	-	-	-	-	ARI: 0.9805	ARI: 0.9800
16.	-	-	-	-	-	-	-	ARI: 0.9810	ARI: 0.9825
17.	-	-	-	-	-	99.16%	-	-	-
18.	-	-	-	-	-	-	-	ARI: 0.9951	ARI: 0.9955
19.	-	-	-	-	-	-	-	ARI: 0.9956	ARI: 0.9955
20.	-	-	-	-	-	100%	-	-	-
21.	-	-	-	-	-	-	-	ARI: 0.1059	ARI: 0.0000
22.	-	-	-	-	-	-	-	ARI: 0.9931	ARI: 0.9954
23.	-	-	-	-	-	99.93%	-	-	-
24.	-	-	-	-	-	99.94%	-	-	-
25.	-	-	-	-	-	-	-	ARI: 0.0037	ARI: -0.0002
26.	-	-	-	-	-	-	-	ARI: 0.4869	ARI: 0.9502
27.	-	-	-	-	-	100%	-	-	-
28.	-	-	-	-	-	-	-	ARI: 1.0000	ARI: 0.9691
29.	-	-	-	-	-	-	-	ARI: 1.0000	ARI: 1.0000
30.	-	-	-	-	-	100%	-	-	-
31.	-	-	-	-	-	-	-	ARI: 1.0000	ARI: 0.9476
32.	-	-	-	-	-	-	-	ARI: 0.9191	ARI: 0.9893
33*.	67.67%	70.88%	64.03%	73.66%	74.73%	76.23%	75.59%	-	-

Chapter 8

Validation

In this chapter, the results obtained from the models developed in chapter 7 are shown. Also, an evaluation is made as to which model(s) is most optimal to classify the different rock formations, rig operations and drilling incidents. For rock formation classification, the most optimal models get validated on a test set of observations that has been prepared using a portion of the original data gathered from either test drilling or the original data from Volve field. For rock classification using the drilling rig, a voting system has been designed that is also presented. For each classification task, a recommendation is given with regards to which features that are recommended.

8.1 Rock Classification

Due to the vast amount of data on the x -axis of the plots included in Task a. and b., only a few erroneous predictions will lead to polluted plots. When the results are inspected closely however, it can be observed that most predictions are accurate. In order to use the data in real-time on the laboratory drilling rig, a median filter gets used to output one filtered prediction each second.

8.1.1 Task a. Laboratory Experimental data

The labels for the Laboratory Experimental data are: 1: Cement, 2: Chalk, 3: Granite, 4: Sandstone, 5: Salt, 6: Shale.

Laboratory formation classification of six formations using models from Case 1

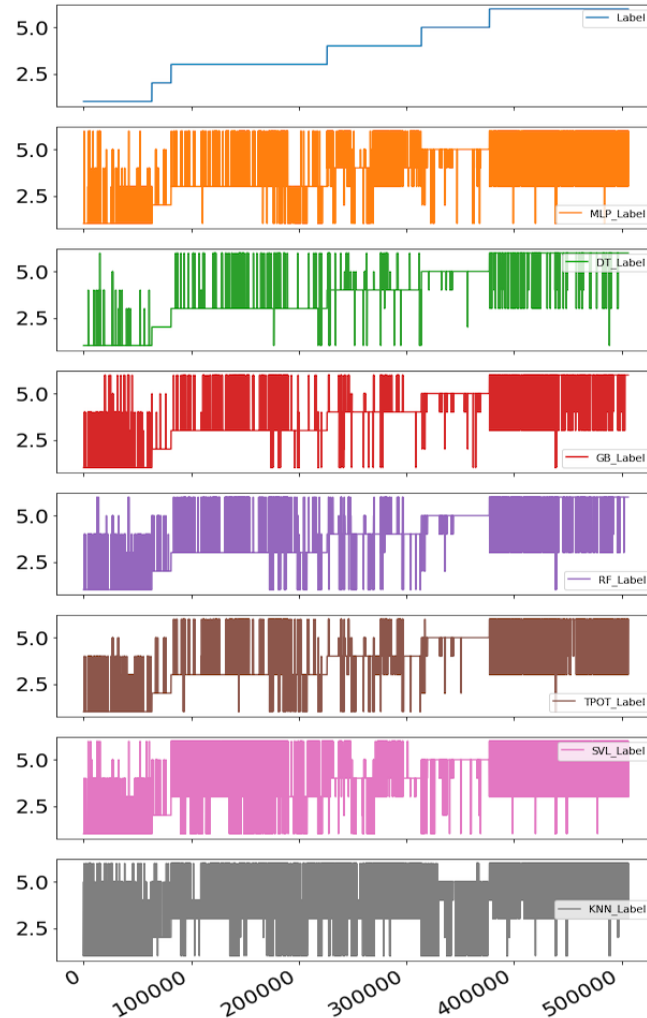


Figure 8.1: *Laboratory formation classification with all features, models trained for six formations.*

In Figure 8.1, the predictions from seven models on a dataset that consists of six different formations (cement, chalk, granite, sandstone, salt and shale) is shown. In the top of the figure, the labels that have been added to distinguish the different datasets from each other are shown.

From the table at the beginning of chapter 7, the data has been pre-processed by removing missing- or invalid data and duplicates, normalized, and then all rows containing observations with outliers in the natural features have been removed. While the results are not shown here, it was determined to not use any of the models that have been trained on the laboratory formation dataset without re-

moving outliers, since this would cause more erroneous predictions compared to when IQR method has been used ahead of training the models.

Laboratory formation classification of six formations, median-filtered predictions

From evaluating the prediction outputs it was determined that the control system only needs to receive one package of predictions outputs (from the seven models) every second. Figure 8.2 shows the output from applying a median filter over a 96 Hz sample window (corresponding to 1 second of samples), and writing the median prediction value from those 96 samples to all rows with observations in that time interval to not have to downsample the dataset to 1 Hz.

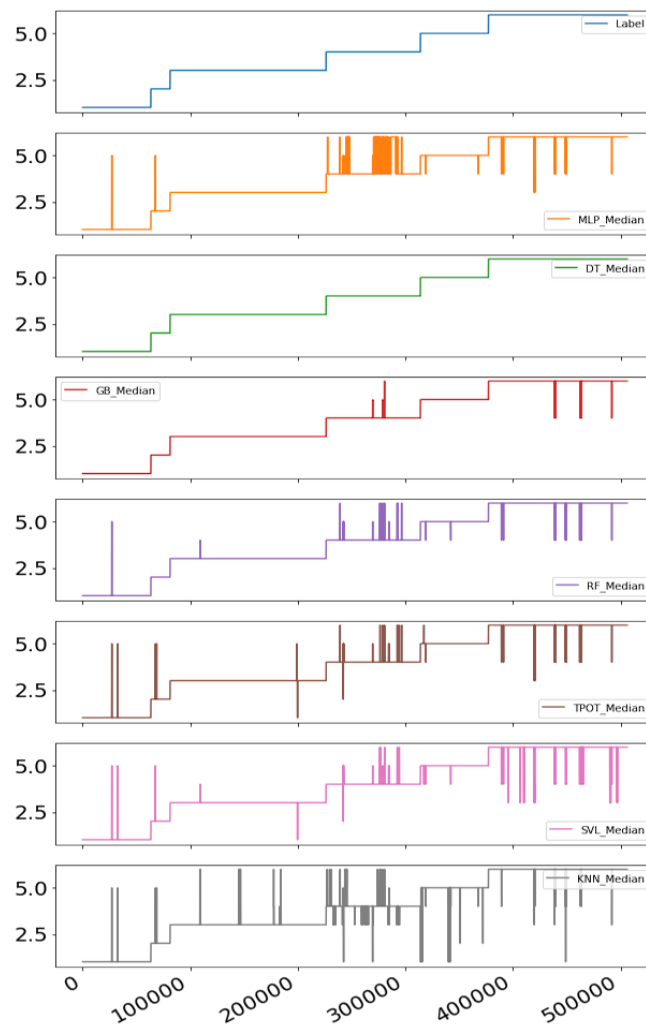


Figure 8.2: *Median-filtered laboratory formation classification with all features, models trained for six formations.*

From analysis of the filtered prediction outputs, the best predictions are achieved from Case 1 with the decision tree, gradient boosting and random forest models.

Laboratory formation classification of six formations using models from Case 3

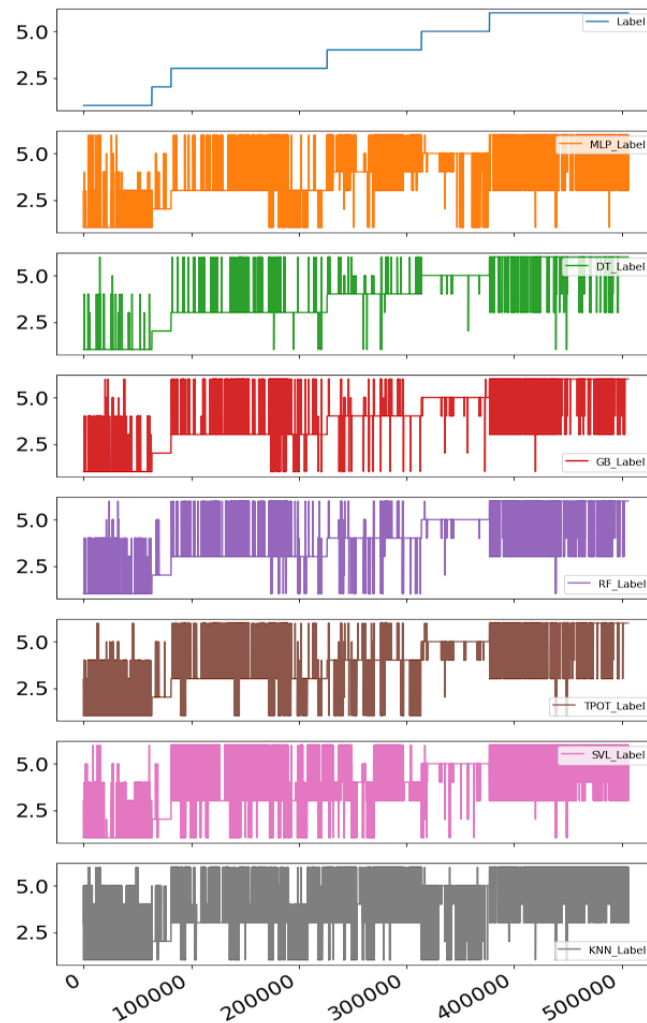


Figure 8.3: *Laboratory formation classification with only six features, models trained for six formations.*

In Figure 8.3 the same dataset (containing six formation types) is run through the models from Case 2, with the only exception being that the models have been trained using only the six highest scoring features obtained by using the ExtraTreesClassifier algorithm. From the results it is seen that by reducing the low-scoring features, the predictions from all models are less noisy, suggesting that fewer of the predictions are erroneous.

Laboratory formation classification, six formations, median-filtered predictions

For the same reasons as given above, a median filter is used to determine the median of the prediction classes over a 96 Hz sample window. While the decision tree, gradient boosting and random forest models continue to deliver the best predictions, all models except the multilayer perceptron and support vector machine now deliver almost identical predictions as the Label (shown in the top of the figure), considering that approximately 500000 observations are shown on the x -axis.

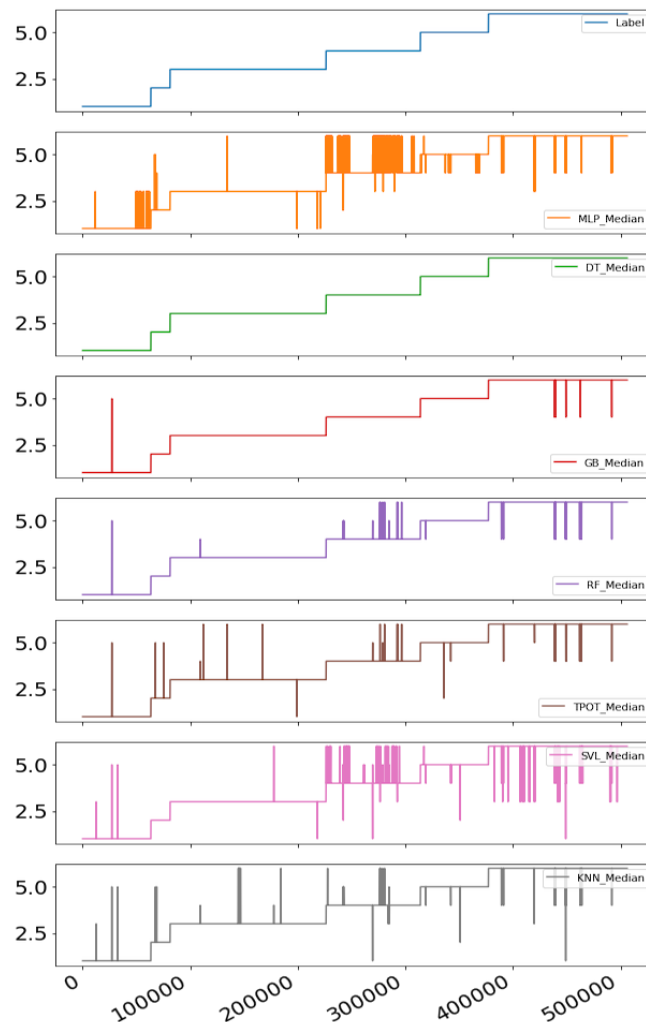


Figure 8.4: *Median-filtered laboratory formation classification with only six features, models trained for six formations.*

Laboratory formation classification of three formations using models from Case 4

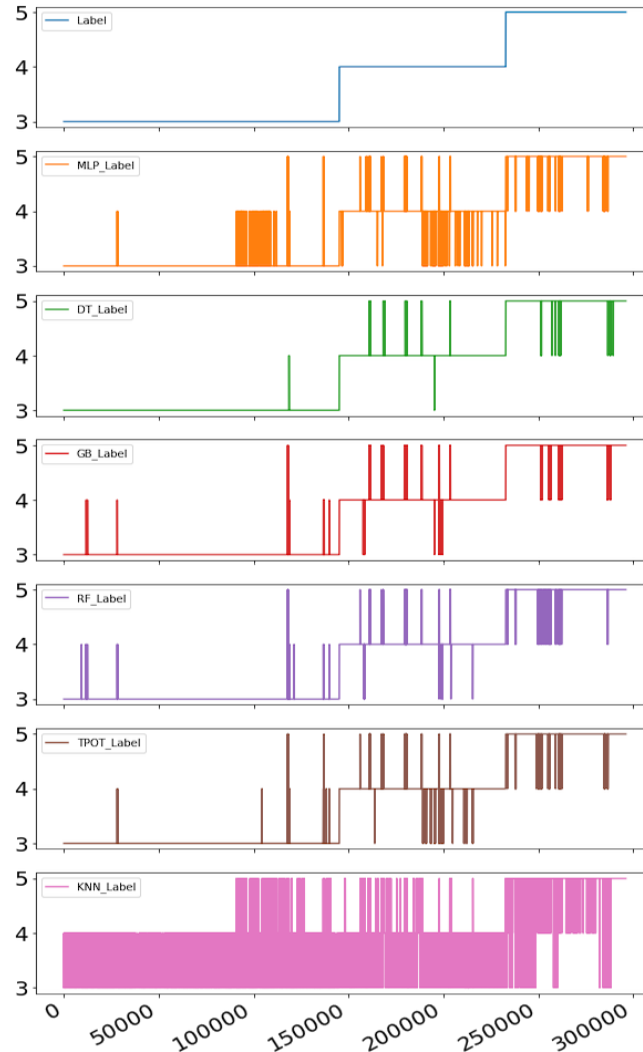


Figure 8.5: *Laboratory formation classification with all features, models trained for 3 formations.*

While the above experiments were conducted for six different formations, several of the formations are similar in drillability such as sandstone and cement. For this reason, the models from Case 4 that have been trained on classes 3: granite, 4: sandstone and 5: salt, which respectively represent a hard-drilling formation, a medium to hard-drilling formation and a soft formation, were used on a dataset containing only those three formations. From the results, it is seen that except from K-nearest neighbor model, all five models multilayer perceptron, decision tree, gradient boosting, random forest and TPOT perform well. Support vector machine was not used for the three different formations due to an error in the

configuration of the model when predicting on the datasets for three formations.

Laboratory formation classification of three formations, median-filtered predictions

The same method of median-filtering the prediction outputs from the six models is used (1 output from each model every second). By inspecting the results in Figure 8.6, decision tree and gradient boosting deliver the best results, but for three formations either model should be sufficiently robust to accurately detect the drilled formation.

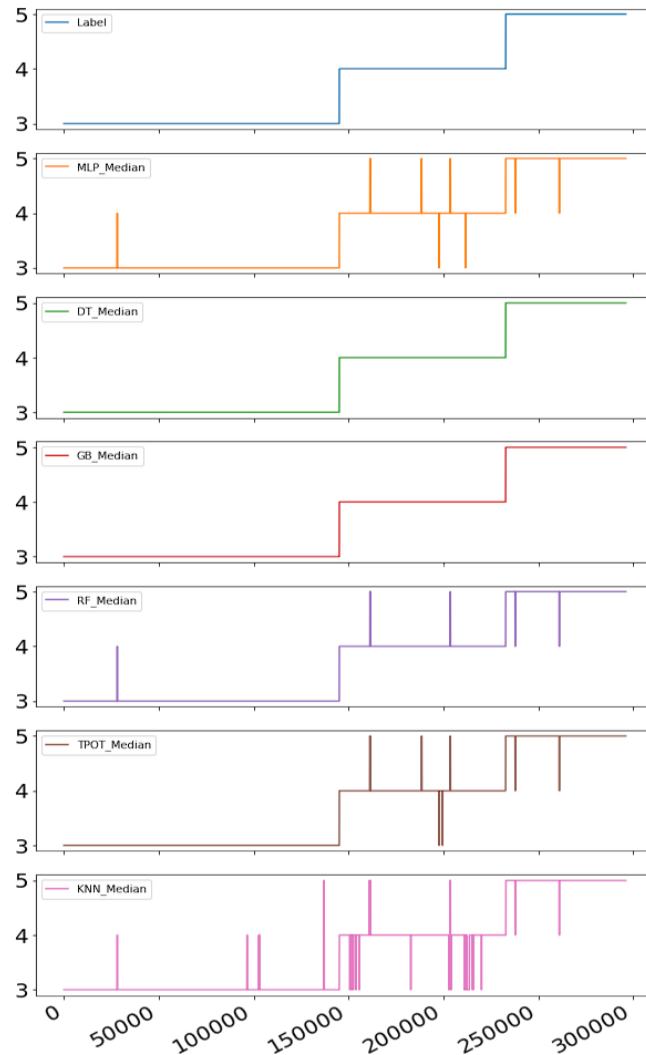


Figure 8.6: Median-filtered laboratory formation classification with all features, models trained for 3 formations.

Laboratory formation classification of three formations using models from Case 5

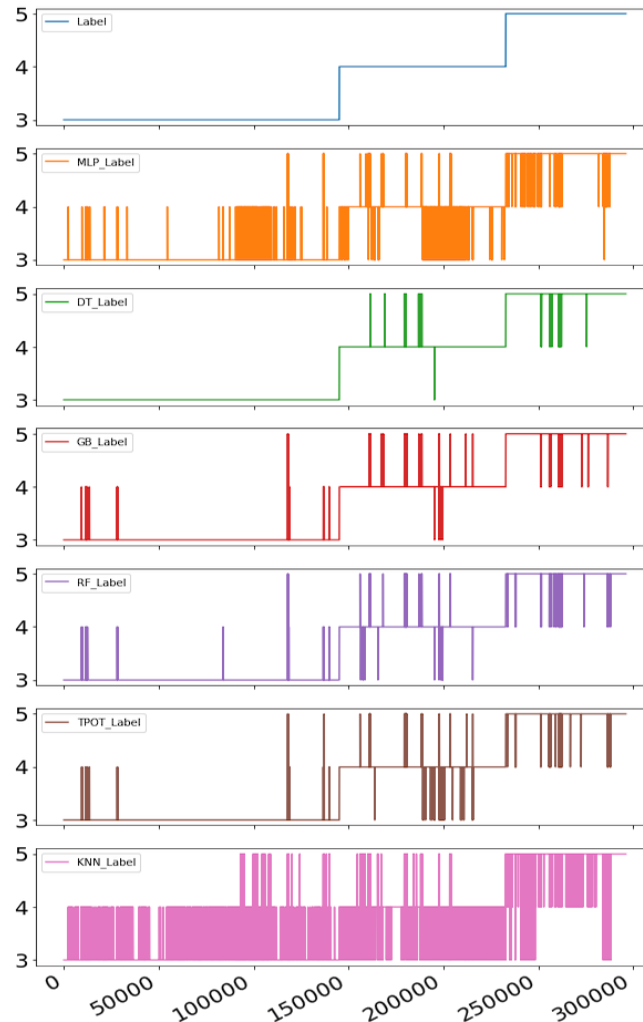


Figure 8.7: *Laboratory formation classification with only six features, models trained for 3 formations.*

Finally, the same dataset is run through the models in Case 5 that have been developed for three formations with the six highest-scoring features from the ExtraTreesClassifier algorithm. From the results of raw model-outputs, all models except for multilayer perceptron and K-nearest neighbor perform well, suggesting that compared to the last case, multilayer perceptron model perform better with more features than the six identified.

Laboratory formation classification of three formations, median-filtered predictions

From median-filtering the prediction outputs to one prediction every second, all models again perform very well. While K-nearest neighbor continue to score slightly below the other models, the accuracy when correlating the prediction results to the Label plot in the top of each figure are high for all models.

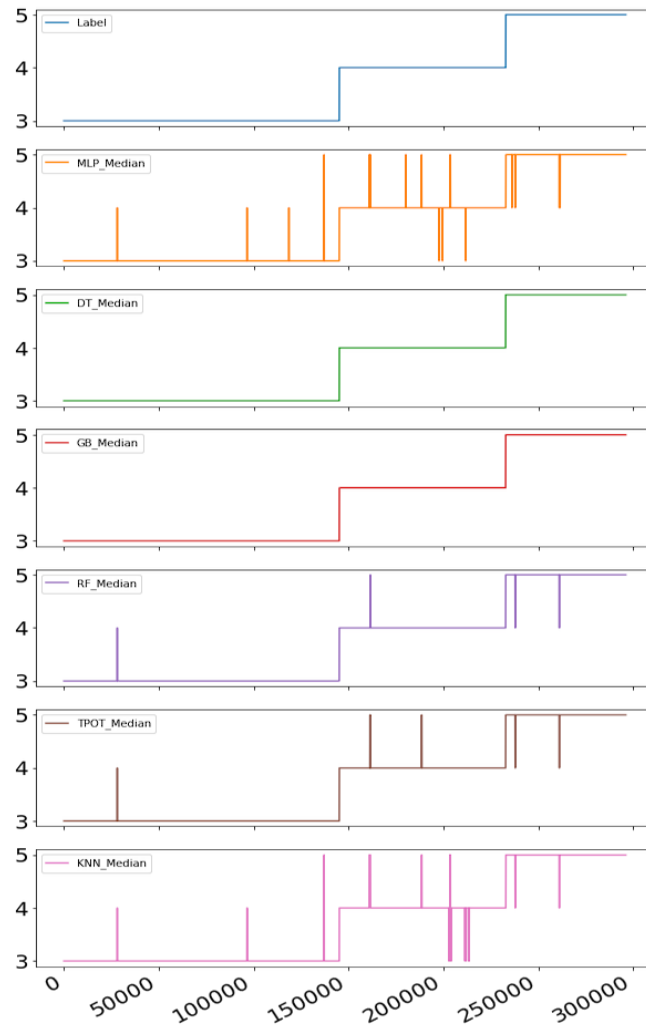


Figure 8.8: *Median-filtered laboratory formation classification with only six features, models trained for 3 formations.*

Granite Test set for validation

Since the validation results presented above are obtained from a test set built using the same operating setpoints for the different formations that the models have been trained on, this makes it difficult to evaluate whether the models are overfitted or if they can be used to predict the rock formation when different

operating setpoints get selected. For this reason, an experiment was run where a granite rock was drilled using different operating setpoints for RPM (300, 500 & 700) and WOB (5 & 7 kg) than was used to train the models. The test set that contains granite drilling data has been downsampled to 96 Hz (from 9600 Hz originally) before it has been pre-processed by removing missing data and outliers, and then normalized. Engineered features have been calculated in accordance with chapter 6.

In the first test, the models developed in Case 3 get used. These are trained on data representing six different rock formations, and it can be re-called that only the six highest scoring features have been used in Case 3. The raw prediction outputs from the models then get median filtered with a window size of 96, in order to obtain one prediction from each model every second. The results are shown in Figure 8.9 below:

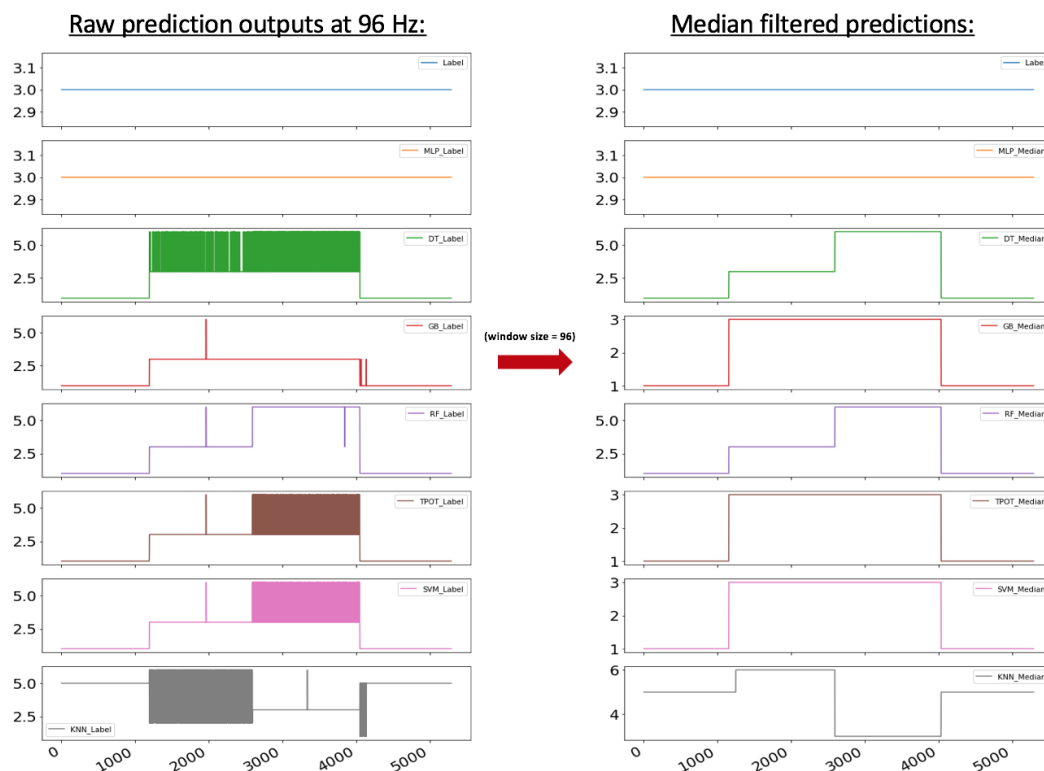


Figure 8.9: Raw (left) and median-filtered (right) laboratory formation classification using a granite test set on models developed in Case 3.

In the second test, the models developed in Case 5 get used, that have been trained on data representing three different rock formations also here using only the six highest scoring features. The results are shown in Figure 8.10

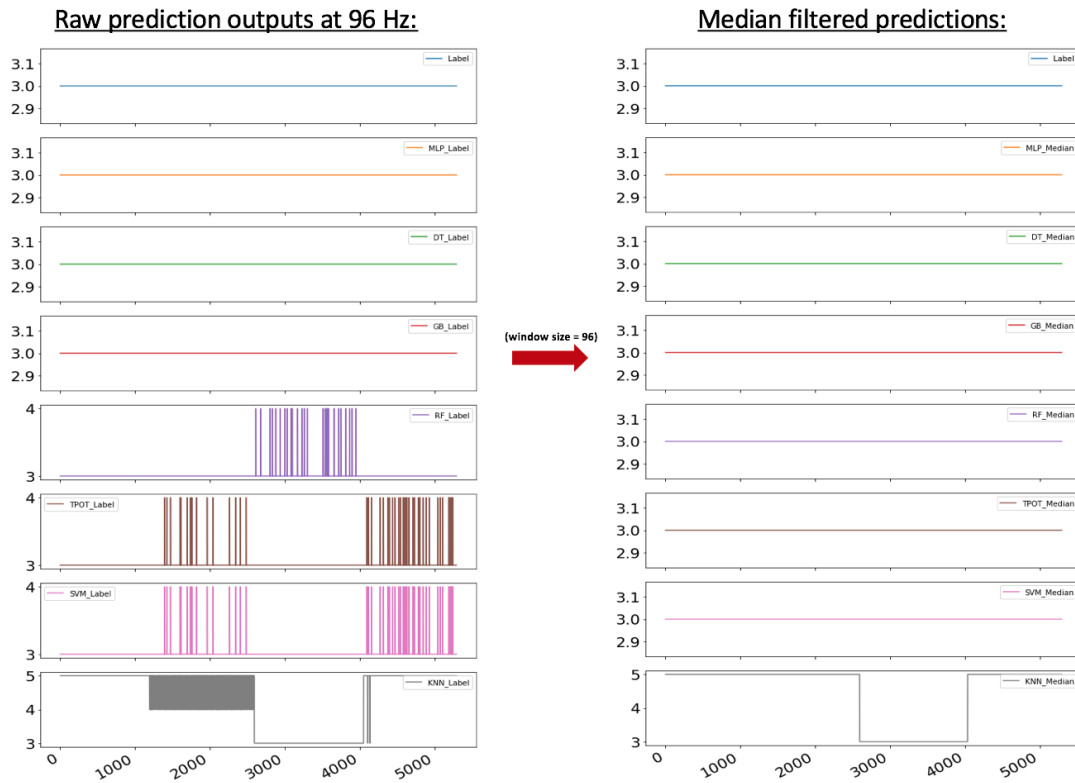


Figure 8.10: Raw (left) and median-filtered (right) laboratory formation classification using a granite test set on models developed in Case 5.

Comparing the results above, it can be observed that all models developed in Case 3 (observations from six different rock formations), except for the artificial neural network model (MLP), are inaccurate when different operating parameters get used for drilling. The erroneous predictions can possibly be explained by that for the six rock formations represented in the data, several are similar in drillability. The similarity could potentially be a challenge seeing as all cases for rock formation classification using the laboratory drilling rig have been developed using surface sensors only. Another explanation can be that the models have been overfitted, i.e. they are too specifically fitted on the training data, and less applicable when exposed to completely different drilling parameters.

Considering the results from the models developed in Case 5 however (observations from three different rock formations), where observations are taken from drilling of a soft (salt), a moderately hard (sandstone) and a hard formation (granite), it can be observed that all models, except for the K-nearest neighbor model, perform very well; in particular when median-filtered to output one prediction every second. Future recommendations to solve the challenge of possibly overfitted models

seen in Case 3, could be to further optimize the models, for instance through reinforcement learning (RL), for cases where several different rock formations get classified. Another future recommendation is to develop the models using downhole data and create additional features from these.

8.1.2 Laboratory rig voting system

Step 1: Model prediction voting

A voting system has been developed to combine the predictions from the seven models into one formation class prediction with a confidence level score. The voting system is further used to signal that a new formation is possibly detected, as well as to confirm that a new formation indeed has been encountered. From analysing the performance of the models (results above) and checking the model performance on a test set separately, the following weights are given to the different models:

- MLP - 1 point
- DT - 3 points
- SVM - 1 point
- RF - 2 points
- GB - 2 points
- TPOT - 1 point
- K-NN - 1 points

The control system is configured to operate at 60 Hz; i.e. 60 predictions per second per model. To ensure as high stability in the predictions as possible, these are median-filtered over a 60 sample window (= 1 prediction per second per model). The prediction value is then used to determine how much weight that the model prediction should be given.

The voting system can be illustrated by considering a case where the median-filtered outputs from the seven models and weights given. By the model weight, each weight is counted as a separate problem, so that if for instance a model is given weight 2, the prediction from that model is equal to the prediction of two models that each have weight 1.

Model	Formation prediction	Model weight	Output
MLP	3	1	3
DT	3	3	3,3,3
SVM	4	1	4
RF	3	2	3,3
GB	5	2	5,5
TPOT	4	1	4
K-NN	3	1	3

Then, a count is performed of the six classes that represent six different rock formations, and a percentage score is calculated that represents the number of times that the class is predicted divided by 11 (total amount of all predictions multiplied by the weight):

Class (rock type)	Occurances	Confidence level
1 (cement)	0	0 %
2 (chalk)	0	0 %
3 (granite)	7	63.64 %
4 (sandstone)	2	18.18 %
5 (salt)	2	18.18 %
6 (shale)	0	0 %

In this example, this suggests that the machine should recognize a granite is being drilled with a 63.64 % confidence, a 18.18 % chance that the formation is a sandstone and a 18.18 % chance that salt is being drilled. The prediction and confidence level is performed once every second (based on the median-filtered raw outputs from the models).

Step 2: New formation encountered

New Formation Detection is handled in the control system by evaluating whether a class (formation type) gets predicted with a higher confidence level than 60 % (from the model voting system) that is different from the previously confirmed formation class; for instance if a sandstone is predicted with 65 % even if the machine previously has confirmed that a granite rock is being drilled.

New Formation Confirmed is handled in the control system by considering the predictions over the last 10 seconds. For example, if 70 % of the predictions in the last 10 seconds are of the same class (all with a higher confidence level than 60 %) the machine could now replace granite with sandstone as the formation being drilled:

Time interval:	Highest Class:	New formation detected?	Old formation:	New formation confirmed?:
t:[0, 9]	Sandstone (SS)	Yes	Shale	NO
t:[1, 10]	Sandstone	Yes	Shale	NO
t:[2, 11]	Shale	No (same as prev. conf.)	Shale	NO
t:[3, 12]	Sandstone	Yes	Shale	NO
t:[4, 13]	Sandstone	Yes	Shale	NO
t:[5, 14]	Sandstone	Yes	Shale	NO
t:[6, 15]	Sandstone	Yes	Shale	NO
t:[7, 16]	Sandstone	Yes	Shale	NO
t:[8, 17]	Salt	Yes (different from SS)	Shale	NO (need >70% of <u>same</u> class)
t:[9, 18]	Sandstone	Yes	Sandstone	YES

Figure 8.11: *A new formation is not yet confirmed in the second last row, since even though a new formation gets detected, this formation class has not occurred in 70 % of the last 10 seconds worth of predictions. The highest class is only filled into the array if the confidence score from the voting output is > 60 %. If less, the cell value is left empty.*

8.1.3 Task b. Volve field data

The labels for Volve field data are: 1: Claystone, 2: Sandstone, 3: Siltstone, 4: Tuff, 5: Marl, 6: Limestone and 7: Coal.

Volve formation classification of six formations using models from Case 6

In Figure 8.12, the predictions from seven models on a dataset that consists of six different formations from the Volve field (same as above except from Tuff) is presented. Same as for the laboratory formation classification, the uppermost plot in the figure represent the geology number, or label, that signals which formation that the observations originally represent. Since the Volve data is collected from multiple sources, the first test is to run the dataset through models that have been trained on data where outliers have not been removed using IQR. All features, as explained in chapter 7 are present in the dataset. From the results, it is almost impossible to distinguish the true and false predictions from each other, as significant noise is present in the dataset, which corresponds with the model accuracy and area under curve results that are presented in the end of chapter 7.

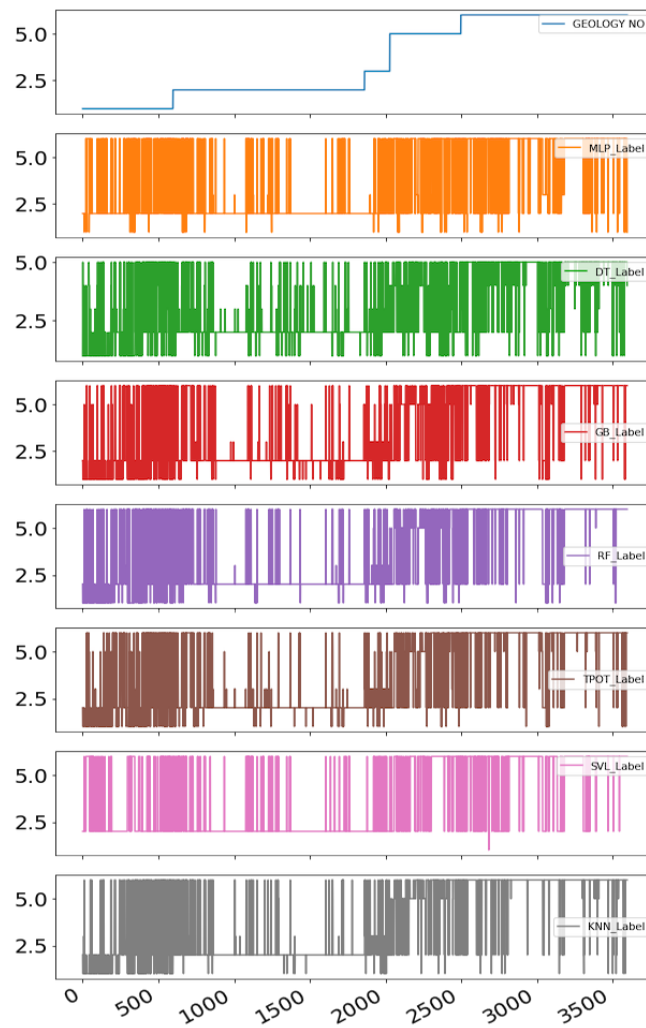


Figure 8.12: *Volve formation classification without removing outliers using IQR with all features.*

Volve formation classification, median-filtered model predictions

While the sampling rate was known in the laboratory formation classification datasets, there exists uncertainty as to exactly what sampling rate the data from the Volve field is captured with. For this reason, an assumption is made that the sampling rate was approximately 10 Hz, and a median filter has been used with sample window of 10 samples. The results are shown in Figure 8.13

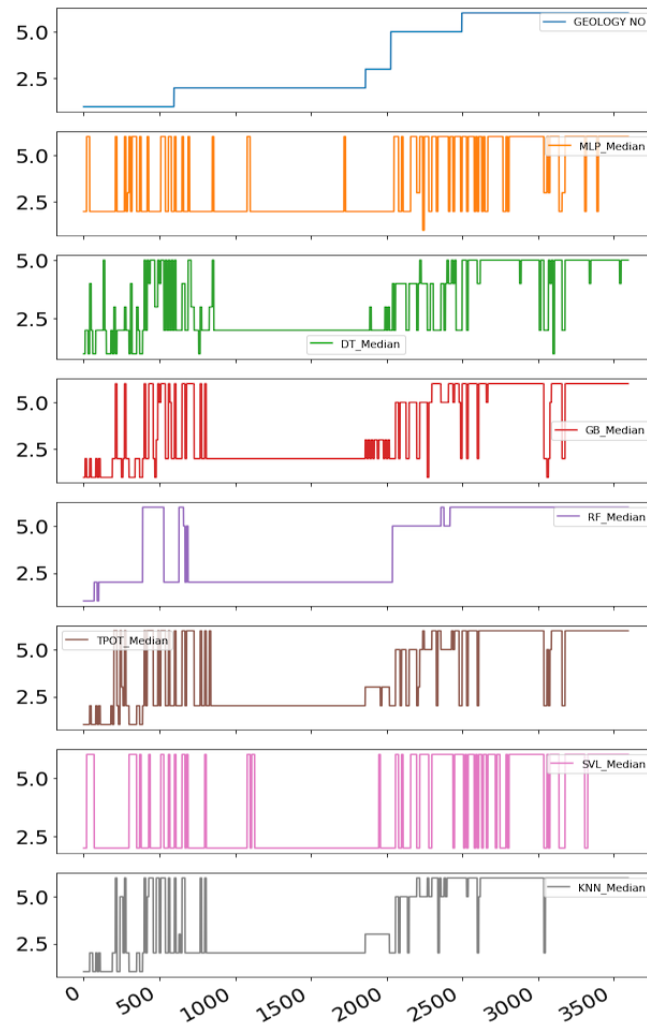


Figure 8.13: Median-filtered Volve formation classification without removing outliers using IQR with all features.

From inspecting the data, only the random forest model is able to accurately predict most of the formations except for classes 1: Claystone and 3: Siltstone. While the decision tree, gradient boosting, TPOT and K-nearest neighbor models are somewhat delivering semi-accurate predictions, still a significant portion of the predictions are erroneous.

Volve formation classification of six formations using models from Case 7

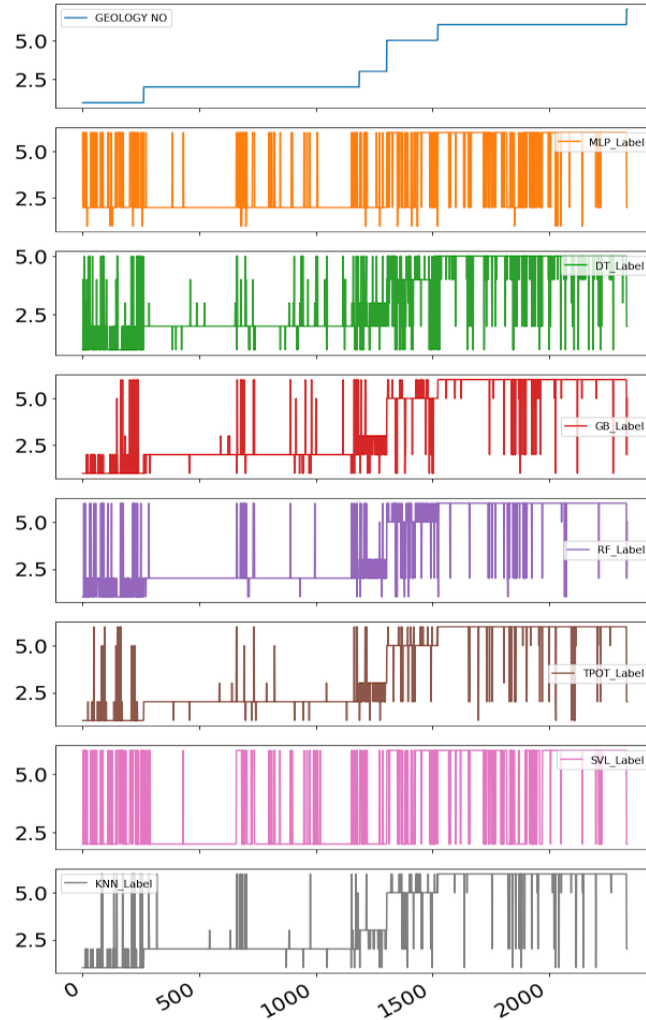


Figure 8.14: *Volve formation classification removing outliers using IQR with all features.*

In Figure 8.14, the same dataset that contains six formations from the Volve field is run through the models from case 7 in which the models have been trained on data where IQR method has been used to remove outliers. From analysis of the prediction outputs, all models except from multilayer perceptron and support vector machine now appear capable of to a certain extent predict the correct geology number (as seen in the top plot of the figure). This suggests that for the Volve field data, it is important to remove outliers in the dataset that could be caused by different drilling phenomena, drilling at different depths and temperatures (which is not accounted for when the datasets were merged) or sampling the data using different equipment such as BHA sensors or other tools.

Volve formation classification, median-filtered model predictions

Once again, a median filter can be used with a window size of 10 samples, assuming that the data has been sampled at 10 Hz.

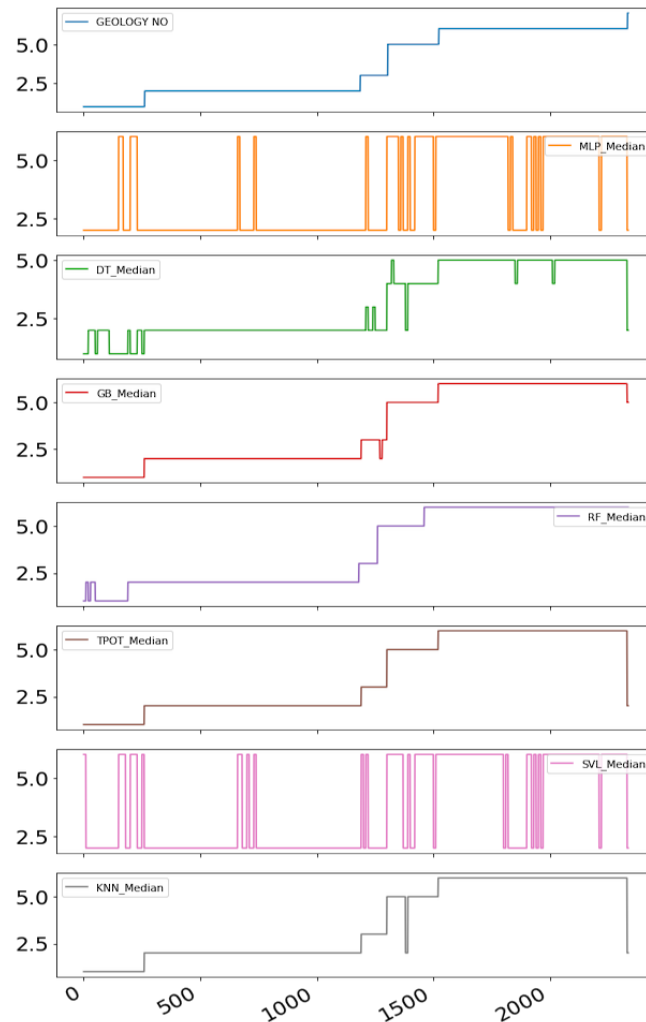


Figure 8.15: Median-filtered Volve formation classification removing outliers using IQR with all features.

By analysing the prediction results, it can be seen that the model developed using the TPOT algorithm is capable of correctly predicting the formation type. The results from the gradient boosting, random forest and k-nearest neighbors models are also very close to the class label (shown as geology no in the uppermost plot). The two models multilayer perceptron and support vector machine still deliver very poor prediction results, suggesting that these models are not applicable for formation classification for field data.

Volve formation classification of six formations using models from Case 8

In the last experiment, the same dataset is run through the seven models from Case 8 that have been trained on a dataset where outliers have been removed using IQR and only the six highest scoring features are used to separate the data from each other. From analysing the raw model predictions, very few changes from the results achieved using Case 7 models are observed. Most significantly, it appears that a smaller amount of erroneous predictions occur, which can possibly be explained by the models not using features that are inseparable for the different formations.

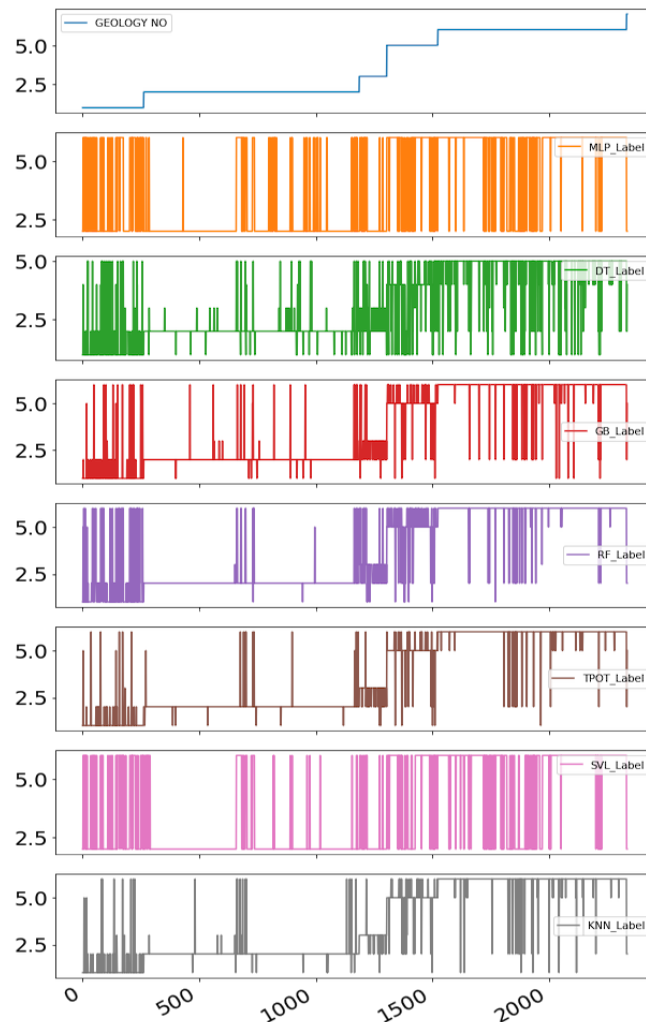


Figure 8.16: *Volve formation classification removing outliers using IQR with only six features.*

Volve formation classification, median-filtered model predictions

When a median filter (window size = 10 samples) gets used however, it can be observed that the decision tree, gradient boosting, random forest, TPOT and K-nearest neighbor models however appear to be less accurate than when all features were used. Results from support vector machine and multilayer perceptron models continue to deliver completely inaccurate predictions, supporting the statement above that these models are not applicable for formation classification using field data collected from drilling.

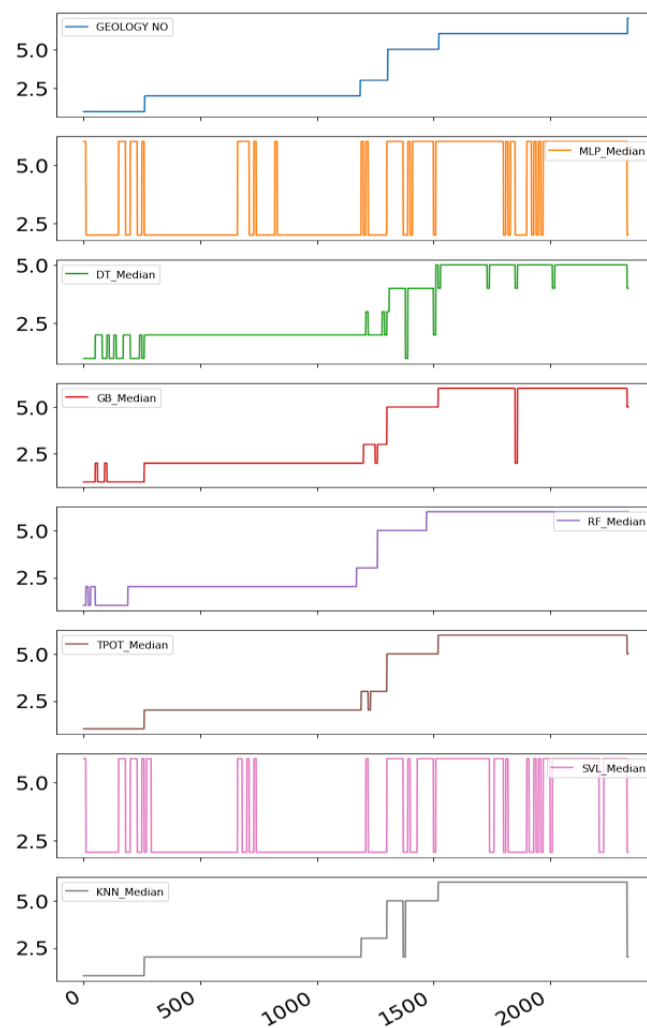


Figure 8.17: Median-filtered Volve formation classification removing outliers using IQR with only six features.

8.1.4 Formation classification - recommendations

On the laboratory drilling rig, the best performing features for rock classification, both based on results when developing the different models, and evaluating rig

performance (from several simulations run) are related to rate of change in depth (i.e. ROP), torque, RPM and WOB setpoints and the artificial features TF3 ($\frac{RPM \times WOB}{ROP}$), depth of cut and bit aggressiveness. For the Volve field data, the best performing features are mud weight going into the well, flow rate setpoint, RPM, torque, depth of cut, MSE and the engineered feature TF3 (same as above).

While high accuracy is achieved using setpoints for RPM, WOB, flow rate and mud weight, such features could be selected depending on which formation that is expected to be encountered, and are therefore not representative for the interaction between bit and the formation. It is therefore our opinion that valuable features are ROP performance, torque, depth of cut and bit aggressiveness. Features recommended to be further developed are interactions between ROP, torque, and the operational setpoints such as the relationship between torque and depth of cut and resulting ROP as a result of torque and WOB. Furthermore, if available, one should aim at developing the models using downhole measurements such as gamma ray and sonic logs, which should greatly increase the performance and accuracy of models.

In terms of pre-processing, best results get achieved when invalid data and outliers have been removed in the training datasets, particularly for the Volve field. Of high importance, when developing models using data from drilling of different wells and at different depths, the data should both get normalized with fixed boundary conditions (such as 0-200 RPM), and correlated for varying depth, pressure and temperature. While not critical on the laboratory scale, Volve formation classification results would likely improve if a correction had been applied.

Again it should be emphasized that methods such as selecting features blindly based on the feature importance scores from for instance an ExtraTreesClassifier is **not** recommended. Instead, a critical assessment should be made with regards to whether a feature describes the phenomena that one wishes to classify or predict, or whether the feature is completely unrelated. While there has been no time in this thesis work, a future recommendation is to develop features that represent changes in the frequency domain, and to develop models using the principal components found from PCA method. While in our opinion machine learning can be an extremely valuable tool to increase the situational awareness and provide early warning, it is still our opinion that these models do not replace the physical models, but instead can be used to compliment them.

8.2 Drilling Rig Operations Classification

8.2.1 Task c. Laboratory Experimental data

The labels used for the Laboratory Experimental data are: 1: POOH, 2: RIH and 3: ROnB.

Laboratory rig operations on original dataset

The three rig operations POOH, RIH and ROnB can similarly be predicted, as is shown in Figure 8.18. The dataset is run on the K-nearest neighbor model in Case 11, that is built using a combination of natural features and engineered features. In the last plot in the figure, the filtered predictions are shown in which the median is calculated over a 96 sample window size.

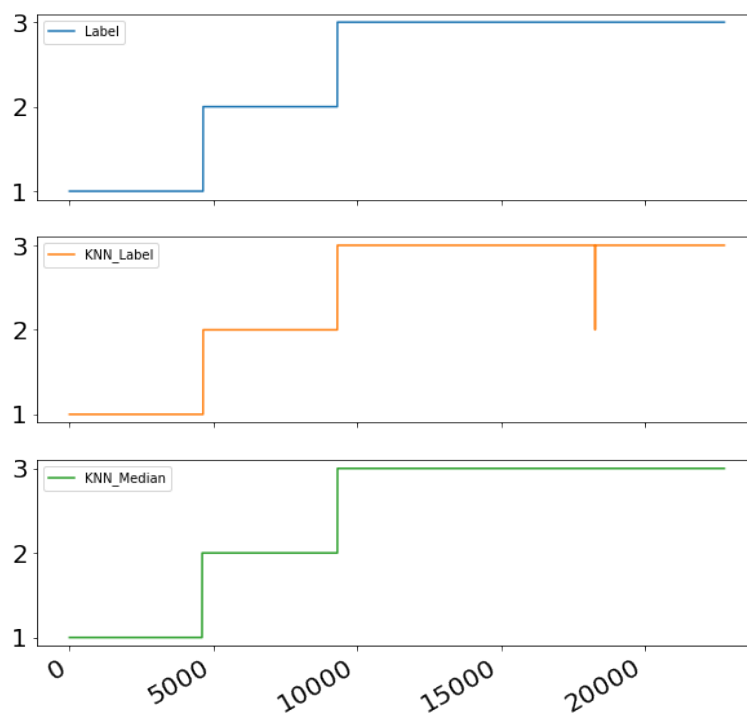


Figure 8.18: *Laboratory rig operations classification with raw- and median-filtered prediction.*

From Case 12, it can be seen that the unsupervised K-means model is capable of identifying the three different rig operations using the parameters WOB and ROP, since for this model the number of clusters can be specified. The unsupervised DBSCAN model however, interprets that four clusters are present, suggesting that only natural features is not robust enough. If one considers Case 13, the two engineered features ROP median (y -axis, window size = 96 samples) and

ROP maximum - ROP minimum (window size = 96 samples), both K-means and DBSCAN models are capable of organizing the different rig operations by their correct classes, where the centermost cluster represents ROnB, the left cluster represents POOH and the right cluster RIH. Considering the results from cases 11 through 13, there is no challenge in classifying the rig operations using the K-nearest neighbor model developed.

8.2.2 Task d. Volve field data

The labels used for the Volve field data are: 1: Drilling, 2: RIH and 3: POOH.

Volve rig operations on original dataset

As with the laboratory rig operations, the prediction of the different rig activities from the K-nearest neighbor model built in Case 14 being drilling, POOH and RIH are shown in Figure 8.19. Immediately, it can be seen that the model is capable of accurately describing the different operations. In Case 15, K-means and DBSCAN clustering models got used to evaluate the three operations when considering the bit RPM and the surface RPM. Other natural features such as the weight on bit or torque were incapable of being organized properly. While K-means models is capable of identifying the three operations to a certain level, DBSCAN continues to predict that there is four classes in the data. In Case 16, the two features TF1 and TF3 are evaluated, which depend on the RPM and WOB changes. Also here, two of the classes are difficult to separate, but judging from the figure, the K-nearest neighbor model performs adequately.

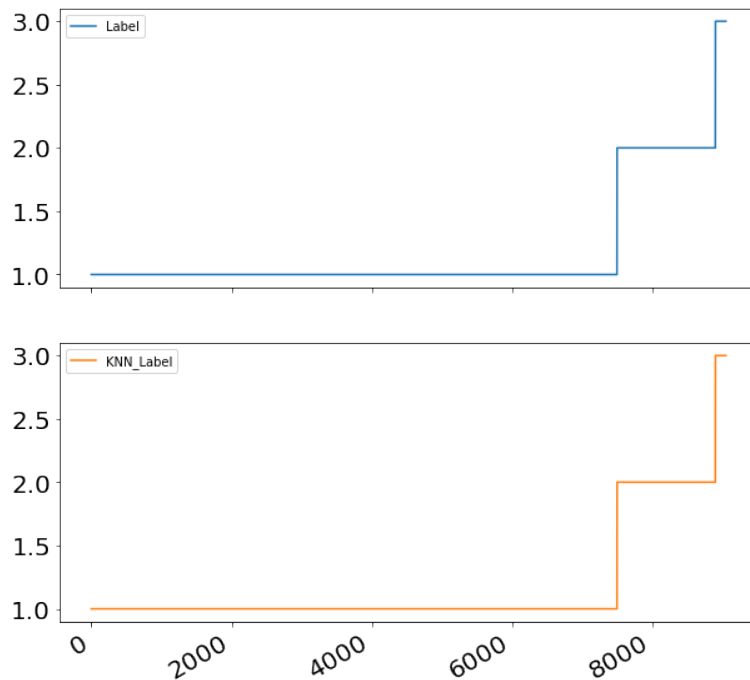


Figure 8.19: *Volve rig operations classification.*

An attempt was made to assimilate the WOB, RPM, torque and ROP measurements from field data with laboratory rig data. When trying to develop the model for the laboratory case and perform predictions on the field data, a poor performance is noted. A future recommendation is to try to correlate the drilling parameters and measurements between the laboratory rig and the full scale, which if performed successfully can contribute to develop the best models using the laboratory rig.

8.2.3 Rig operations - recommendations

For rig operations, features related to rate of change in depth (laboratory drilling rig) or RPM, ROP (differentiate between drilling and tripping), $TF3 \left(\frac{RPM \times WOB}{ROP} \right)$ and crown block position (Volve field data) perform the highest. Judging from the results, classification of rig operations is easy, yet it is very difficult to correlate the data between the drilling rig and the field data.

While it likely is not possible to develop accurate models on the laboratory scale to classify formations on a full-scale drilling rig, it is our recommendation to first and foremost attempt to correlate the data between the laboratory drilling rig and the Volve field data for rig operations. If successful, simple algorithms (either developed using machine learning or for instance the developed API for remote

rig interaction, as is mentioned in subsection 2.8.2) can get used to verify that rig operations get executed in accordance with commands sent forward by a fully digital control system executing digitalized drilling procedures, as has become a hot topic among several operators in recent years. Such functionality of correlating the data would also open up for research and simulation of functionality using the drilling rig, before technology can get deployed for testing on the full scale.

8.3 Drilling Incident Classification

8.3.1 Task e. Pressure Incident Detection

The labels used for Pressure Incident Detection are: 1: Normal Pressure, 2: Leak (losses) and 3: Overpressure (plugged nozzles).

Losses and plugged nozzles detection (classification) of original dataset

Three different pressure cases get predicted in Figure 8.20 using the K-nearest neighbor model developed in Case 17. Comparing the predicted classes with the pressure ranges, the K-nearest neighbor model performs well, which is also supported by both K-means and DBSCAN models developed using natural features in Case 18 and engineered features in Case 19. From evaluating the performance of the pressure model on the rig, the model quickly detects when a drop in pressure (possible leak case) or an increase in pressure (such as plugged nozzles or plugged exhaust on the pneumatic motor) occurs, suggesting that the model can be used when circulating mud (using the conventional setup configured for vertical drilling operations).

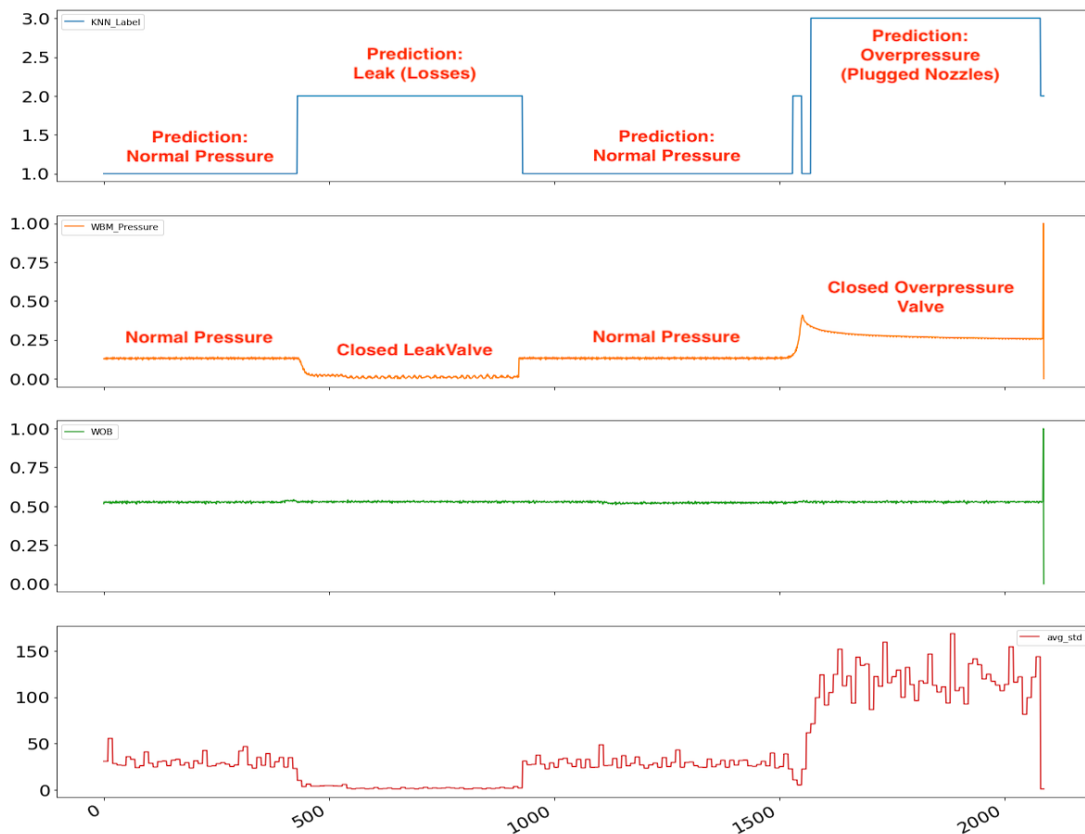


Figure 8.20: Leak and overpressure classified using the K-NN model developed in *Case 17* with data acquired from experiment drilling a rock sample while closing or opening two valves to simulate leak (losses) or overpressure (plugged nozzles). Data has been treated for invalid data and normalized.

For directional drilling, solenoid valve regulation is an important control parameter. During the different phases such as POOH and RIH through a whipstock, a near-closed solenoid valve can trigger a leak (losses). Similarly, overpressure should not occur in the system since both a manual choke and a pressure relief valve is installed. A passive overpressure indicator is therefore more suitable than the model when drilling with the pneumatic motor.

8.3.2 Task f. Surface Drilling Vibrations Detection

The labels used for the Surface Drilling Vibrations Detection are: 1: Normal Vibrations and 2: Heavy Vibrations.

Axial vibrations have in the past been detected by considering the oscillations in the load cells that get used to measure the hook load changes (WOB), mounted between the top plate that resembles the drilling floor (where the top drive is

mounted) and the actuators that acts as draw works. In Figure 8.21, the K-nearest neighbor model from Case 20 is used to predict whether normal or heavy vibrations occur on a portion of the dataset that was collected to train the model, in which the top drive RPM is varied when the BHA is unconfined inside a vertical pilot hole section. Considering the K-means and DBSCAN results in Case 21, it is evident that vibrations can not be separated using only the natural features WOB and torque, since the heavy vibration cycles overlap the natural (low) vibrations. By introducing engineered features however such as the relationship between the average measurements and the standard deviation, and the maximum amplitude (Case 22), it seems that heavy vibrations can possibly get detected with surface sensors equipment.

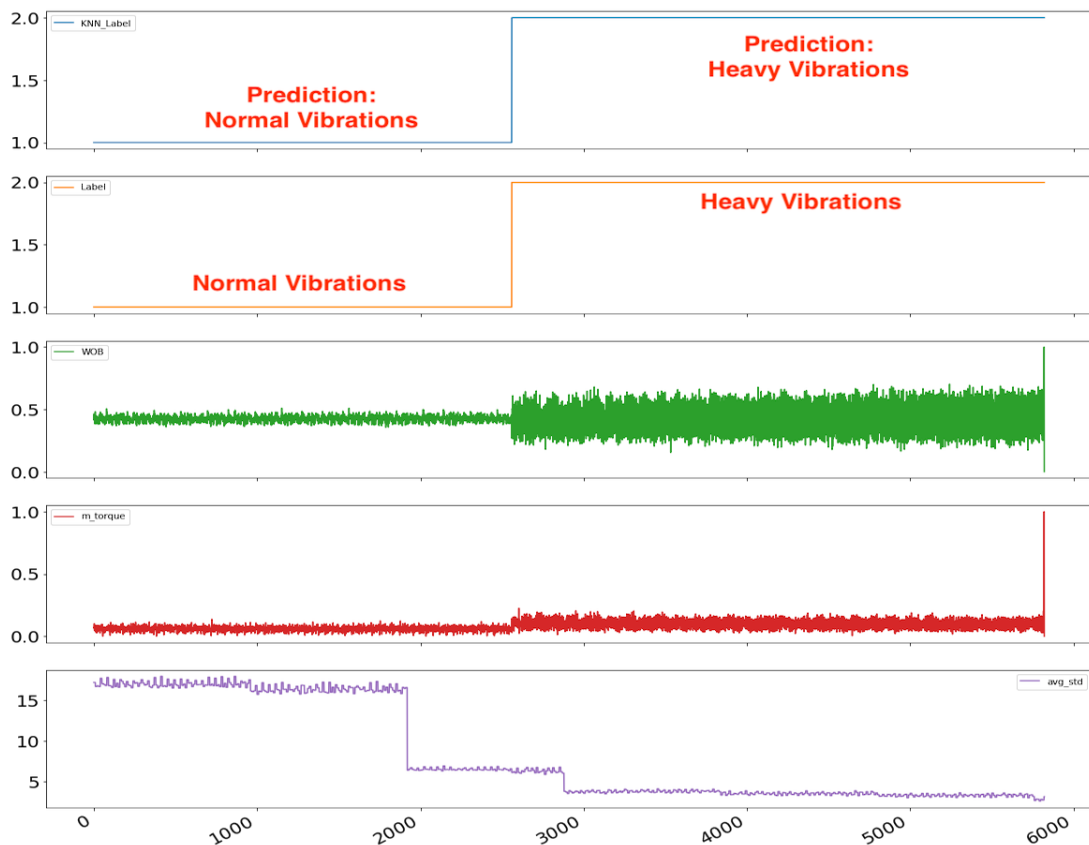


Figure 8.21: *Vibration classification using surface sensors.*

8.3.3 Task g. Downhole Vibrations

The labels used for Downhole Vibrations are: 1: Low Vibrations, 2: Moderate Vibrations and 3: Heavy Vibrations.

Vibration classification using only the highest scoring features from downhole sensors

As was observed using natural features in the surface vibrations model, several of these overlap, which affects the performance of models. For this reason the downhole vibration classification is performed using only the six highest scoring features (Case 24 rather than 23), based on the ExtraTreesClassifier algorithm. In Figure 8.22, the K-nearest neighbor model developed in Case 24 is used on the same dataset that the model was trained on to predict three vibration levels: low, moderate and heavy. In Figure 8.23, the model is used on a fresh dataset when drilling a vertical section for the three vibration classes. As can be seen from the two figures, the model scores well using engineered features only.

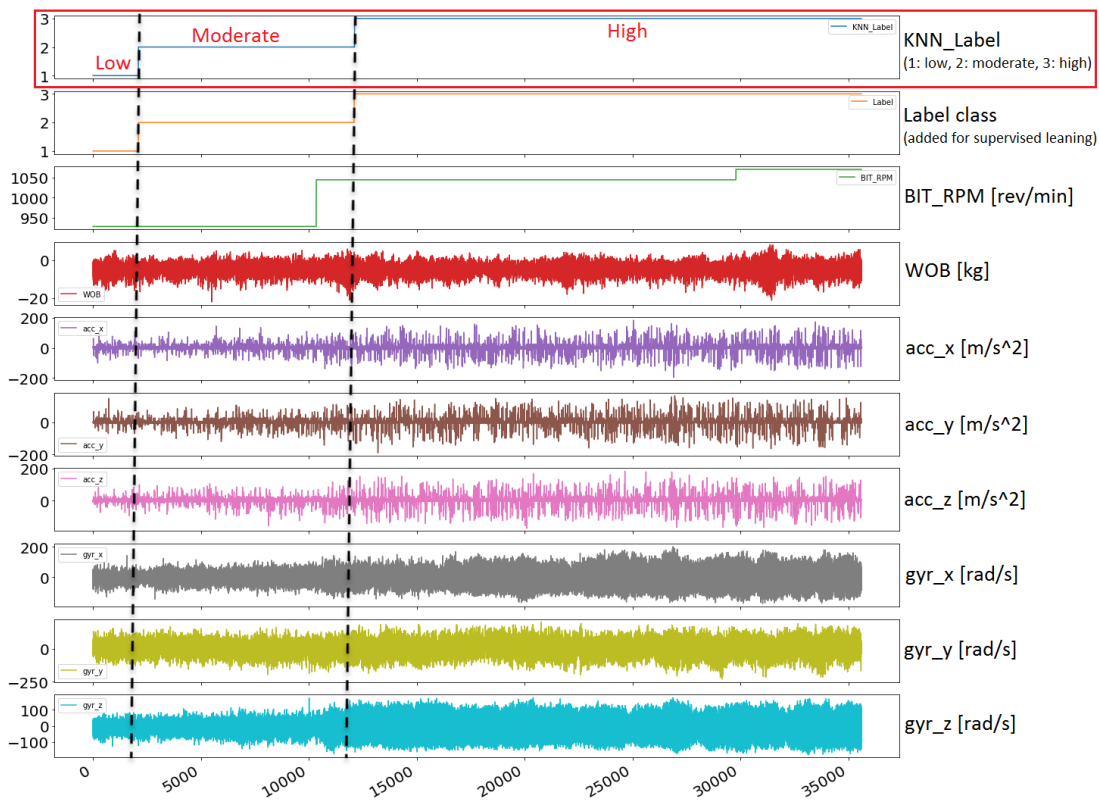
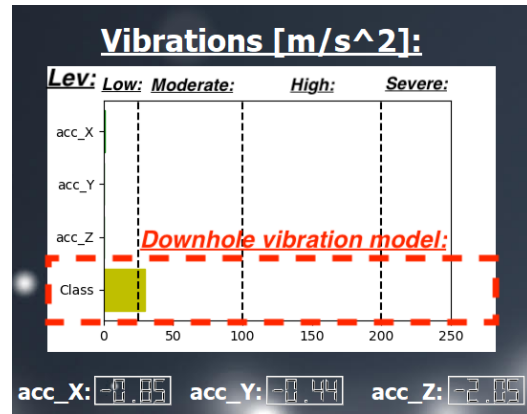


Figure 8.22: Three vibration levels (1: low, 2: moderate, 3: high) classified using KNN-model developed in **Case 24** with data acquired from drilling a short inclined section using a 7 deg whipstock. Accelerometer and gyroscope data in bottom show correspondance between vibrations and classification results.

```

C:\Windows\system32\cmd.exe - python a4e_downhole_vibrations.py
Downhole listener initialized
Downhole vibrations servicer initialized
C:\Users\UISDrillbotics\Anaconda3\lib\site-packages\sklearn\base
rom version 0.19.1 when using version 0.20.3. This might lead to
UserWarning)
Vibration level is: Low
Vibration level is: Low
Vibration level is: Low
Vibration level is: Low
Vibration level is: Low
Vibration level is: Low
Vibration level is: Low
Vibration level is: Low
Vibration level is: Moderate
Vibration level is: Moderate
Vibration level is: Moderate
Vibration level is: Moderate
Vibration level is: High
Vibration level is: High
Vibration level is: Moderate
Vibration level is: Low
Vibration level is: Low
Vibration level is: Moderate
Vibration level is: Moderate
    
```

(a) Python print function of downhole vibration levels.



(b) Visualization of downhole vibration level in GUI.

Figure 8.24: Downhole Vibration Classification as implemented in the drilling system.

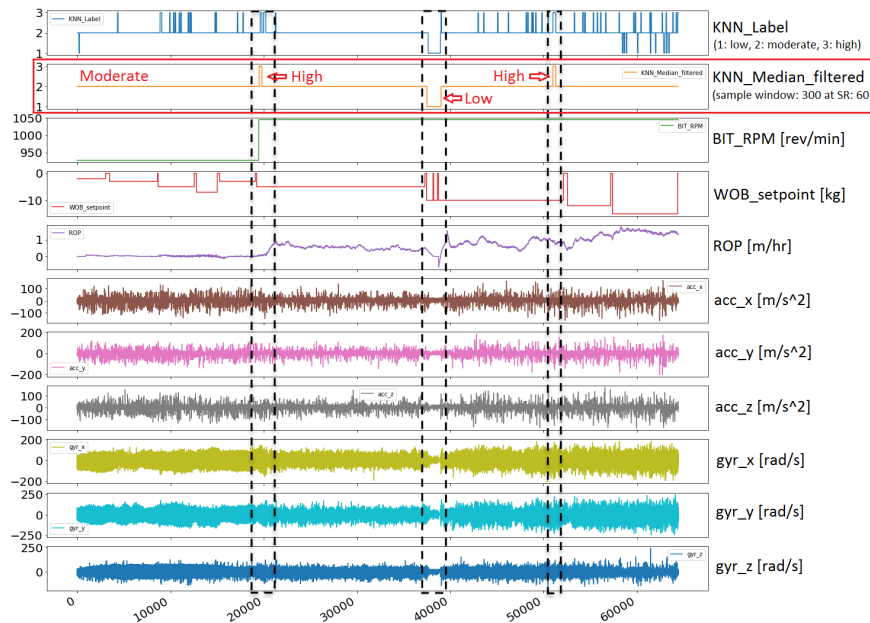


Figure 8.23: Three vibration levels (1: low, 2: moderate, 3: high) classified using the K-NN model developed in **Case 24** on data acquired from drilling a vertical pilot hole section using a riser. Median filter is applied for 2nd figure at top over a 300 sample window (= 5 seconds). Accelerometer and gyroscope data in bottom show correspondance between vibrations and classification results.

In Figure 8.24, the performance of the model is shown as it is implemented on the rig.

Based on several experiments conducted using the implemented models on the rig,

performance is consistent and accurate (based on analysis of the data and visual observations of vibration levels). The downhole sensor package should however be positioned closer to the bit, which was particularly difficult this year given the length of the pneumatic motor and the challenge with running a wired pipe back to surface when drilling with the knuckle joint. Even if the sensor package can be used to classify the string vibrations, there setup is assessed to currently not be viable to classify for instance bit whirl, which could cause major damage to both the BHA and bit.

8.3.4 Task h. Stuck Pipe Incident Detection

The labels used for the Stuck Pipe Incident Detection are: 1: Normal drilling and 2: Stuck pipe.

Stuck pipe classification

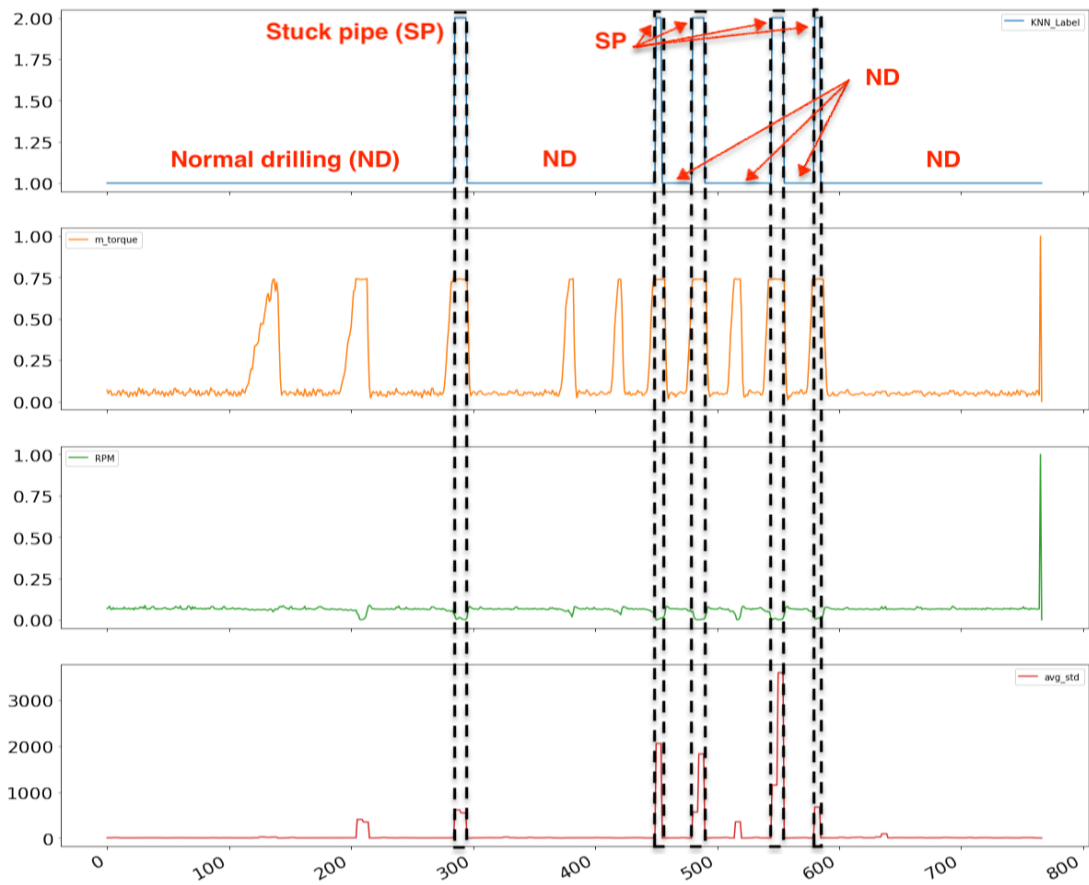


Figure 8.25: *Stuck pipe classified using the K-NN model developed in Case 27 with data acquired experiment where the pipe is clamped repeatedly, simulating a stuck pipe incident. Data has been treated for invalid data and normalized.*

In the past, stuck pipe has been detected when the torque is above a pre-determined threshold, and the RPM drops to zero. In Figure 8.25, the K-nearest neighbor model developed in Case 27, and that has only been trained on one instance of stuck pipe is used on a dataset that contains ten stuck pipe incidents. Data that is considered to be invalid (outside of the sensor range) has been removed before the dataset was normalized. Judging by the performance, the model is capable of detecting five out of the ten stuck pipe incidents. This is possibly caused by the remaining incidents being too short to get detected accurately. In Figure 8.26, the same model (Case 27) is used on the same dataset, except that here, the invalid data is kept in the data. This causes six out of ten stuck pipe incidents to get detected. Judging from the results for Case 28 and 29, where K-means and DBSCAN get used to organize the data for natural features and

engineered features respectively, stuck pipe detection should be straight forward and can possibly be greatly improved by building the model on a larger training set.

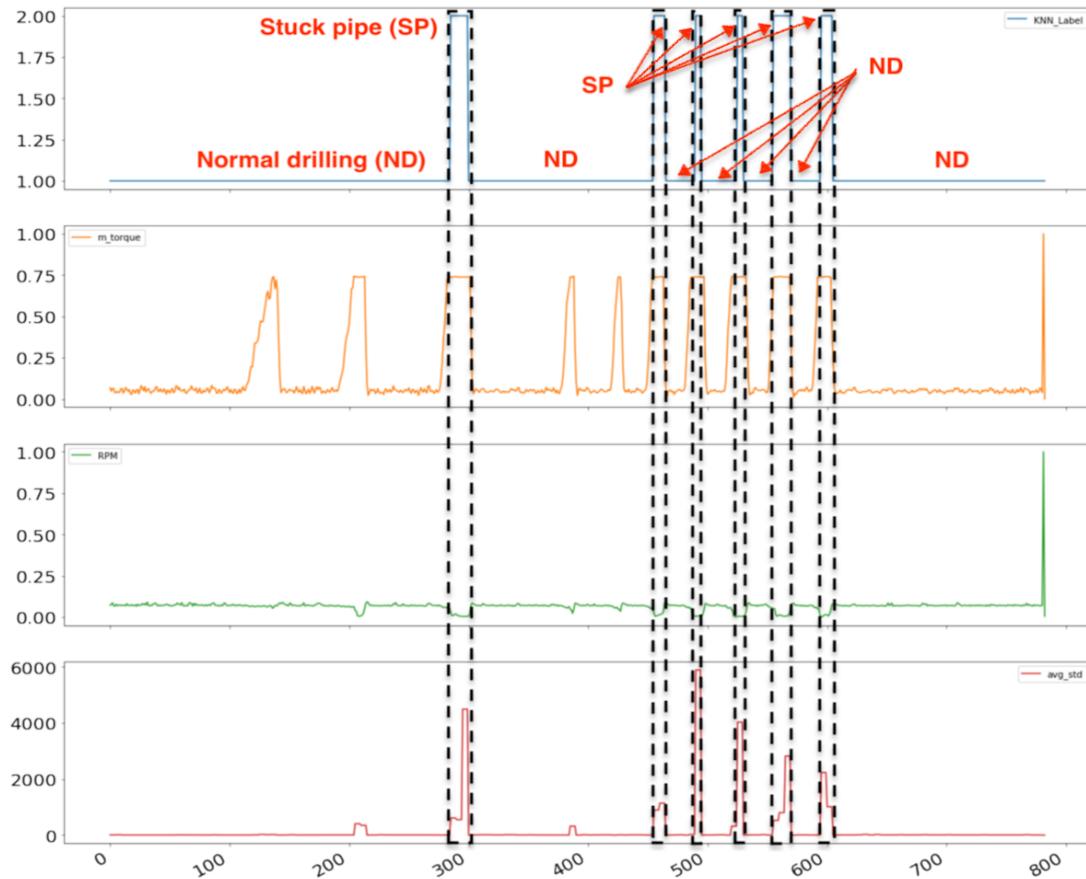


Figure 8.26: *Stuck pipe classified using the K-NN model developed in Case 27 with data acquired experiment where the pipe is clamped repeatedly, simulating a stuck pipe incident. Data has only been normalized.*

8.3.5 Task i. Twist off Incident Detection

The labels used for the Twist off Incident Detection are: 1: No Twist Off and 2: Twist Off.

Twist off detection (classification) of original dataset

As with the stuck pipe, a twist off can get detected considering a sudden change in the torque measurement in the system. In Figure 8.27, the K-nearest neighbor model developed in Case 30 is used on the dataset that contains a twist off incident. The model predicts the twist off with a short time delay, but can be used on the drilling rig to confirm a twist off and thus shut down the system.

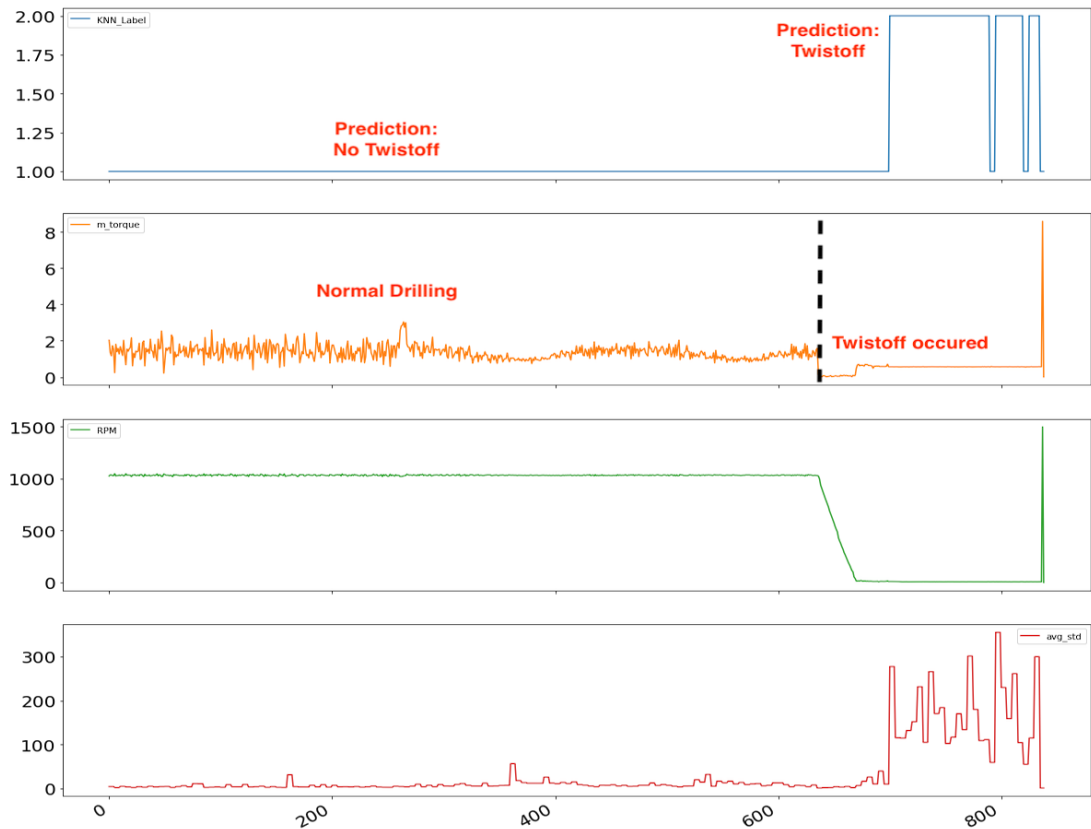


Figure 8.27: *Twist off detected using the K-NN model developed in Case 30 with data acquired from experiment drilling a homogeneous cement rock sample at high operating setpoints where the pipe twists off. Data has been treated for invalid data and normalized.*

Experiments were conducted to develop a model that is capable of twist off detection before the incident occurs. While at more than 1000 RPM it is impossible for the control loop to stop rotation quick enough to prevent a twist off if the pipe gets stuck, a future recommendation is to drill at close to maximum operating conditions, and develop a model that can detect drilling conditions that result in twist off from fatigue in the pipe where the pipe connections are situated.

8.3.6 Incident classification - recommendations

For drilling incidents, it can be observed that natural features such as pressure, torque, WOB and RPM in most cases can get used to distinguish between for instance a normal pressure case, overpressure and leak. It can however be observed that natural features can not be used to distinguish between different vibration levels, suggesting that instead engineered features such as evaluating the maximum, minimum, peak-to-peak and standard deviation over an interval of samples should get used when developing the models. Such features are also incredibly

easy to implement in a real-time system without requiring significant time delay in predictions.

It is our recommendation for incident detection, that outliers do not get removed by IQR method, but instead manually by for instance plotting the data and evaluating each measurement, seeing as IQR method would typically reduce the accuracy of models with 10-15 %. Similarly to the formation classification, the data range should get normalized for all cases to ensure that drilling incidents and trends in data are equally represented in both training data and data used to predict on. Furthermore, it should be a future focus to identify which drilling incidents that can get predicted ahead of the incident occurring. While stuck pipe and twist off occur in a matter of milliseconds on the laboratory drilling rig when drilling at more than 1000 RPM, fatigue in connections and the pipe, as well as pressure build up, are likely incidents that can get predicted and responded to, ahead of system failure.

8.4 Limitations using machine learning

While results from this chapter and chapter 7 show that machine learning can be used to detect different rock formations and drilling incidents, there are however several limitations and challenges using machine learning that have been identified:

- First and foremost, it should be emphasized that the model accuracy heavily depends on the quality of the data used to train the models. This means that while a good model can be created for one objective, there is no guarantee that a good model can be developed for another, unless data of high quality that accurately describes the phenomena exists.
- Secondly, the models depend heavily on the environment that they have been trained for and the data that it gets exposed to. An example of this is a model that has been trained on data acquired in the laboratory environment, but when used in the field is not able to make the correct prediction, even if the trend might be the same.
- Thirdly, another limitation is failing to understand which features that must be selected in order to correctly detect the phenomena that the model gets developed for, and to blindly trust different importance evaluation techniques. An example of this is, as has been mentioned earlier, if the

setpoint for flow rate of mud going into the well gets selected to classify different geological formations.

- When compared to physical models, it is our perception that it can both be difficult to detect and correct if the machine makes a mistake. This is related to the complexity of fully understanding the processes that go into each decision that the machine makes when the model gets built.
- Finally, a major limitation lies in computational power available to train a model on large sets of data. If for instance a deep learning model gets developed from an immense number of observations, the required hardware to train such model can be both expensive and inaccessible. There has however been a big shift in recent years towards cloud-computing, where one can upload the data and use the computational power of a data center to build the model. This also applies to the time that it takes to train a model. If either the time available to train the model or to make a prediction is limited, it is absolutely necessary to understand which models that are computationally expensive to build, and which that are not.

Chapter 9

Autonomous Drilling

The experimental drilling rig that has been developed at the University of Stavanger can also be used to develop, implement and research autonomous drilling algorithms and digital detailed operating procedures (DDOP). As part of the implementation of an autonomous control system to serve in the Drillbotics[®] competition, several search algorithms have been evaluated for ROP optimization..

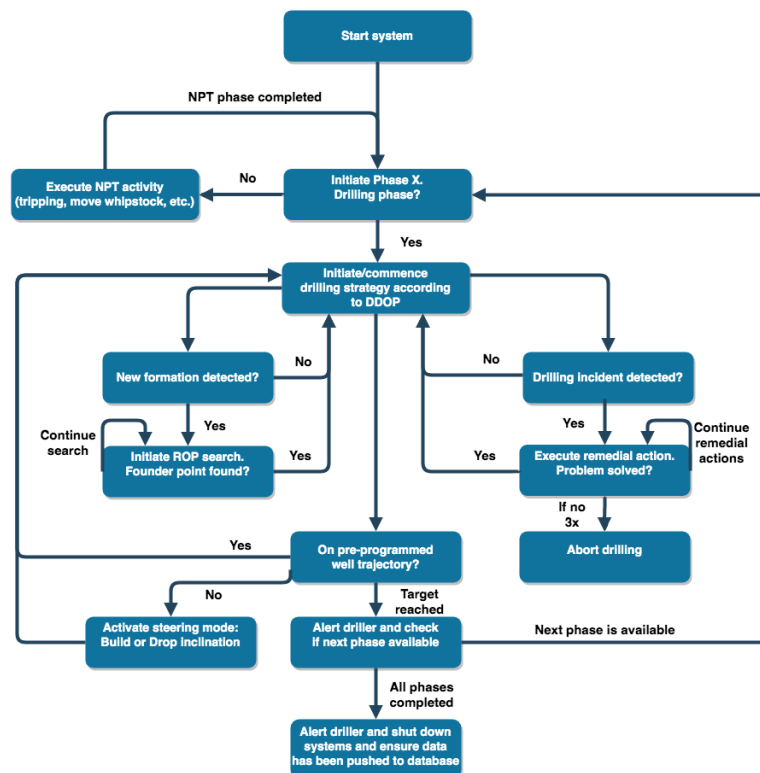


Figure 9.1: *Simplified illustration of the logic behind the autonomous drilling system.*

The concept is a closed-loop autonomous control system that combines several

models with a digital detailed operational drilling procedure developed for directional drilling. A simplified flow chart illustrating the system is shown in Figure 9.1. In addition, a novel voting system has been designed to prioritize which commands that should receive priority if an event, or incident gets detected or if conflicting commands from models get received at the same time. The voting system has been configured so that critical incidents that can risk damage to either equipment or the well receive the highest priority. Because an important objective has been directional drilling, inclination steering (and azimuth steering when enabled) receives the second highest priority. Maximizing the ROP and ensuring that downhole vibrations are low are located at the lowest priority since these tasks are not critical to handle immediately to meet the objective.

9.1 Search algorithms

9.1.1 ROP Optimization Background

Rate of penetration (ROP) is an important performance index on every drilling rig, and in every drilling operation. Ensuring that the ROP is as high, without compromising on HSE, rig equipment safety and well integrity, is not only important for cost efficiency but it also reduces the time that the openhole wellbore is left exposed before running casing, liner (or other equipment into the well) which in turn minimizes the risk of well collapse or time-consuming back-reaming operations.

To optimize the ROP, one first needs to consider what ROP really is. ROP is a multi-variable vector function that depends on various parameters such as WOB, torque on bit (TOB), RPM, flow rate (Q), rock formation strength (UCS), drill bit properties and so on. The machine exerts control over the bit side properties. The formation strength and drilling environment on the other hand is heterogeneous, and changes in the drilling environment must be responded to in real-time. On the laboratory drilling rig, the two most important control variables are WOB and RPM. By varying either of the control variables (that are coupled, meaning that if one is varied so becomes the other) a difference in ROP, drilling efficiency and system response can be expected.

The operational state space can be thought treated as a topographic environment, where two of the control variables constitute the x - and y -axis, and the performance indicator (for instance ROP or mechanical specific energy - MSE) is found

on the z -axis. Since ROP is always non-negative, the state space and resulting performance can be depicted as illustrated in Figure 9.2.

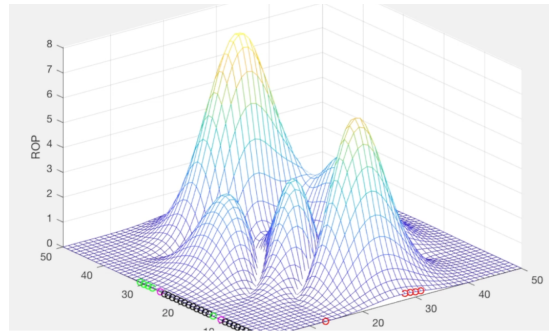


Figure 9.2: *In the topographic environment for ROP, several local peaks exist where the highest ROP is found at the global peak. If a different response variable gets selected, for instance MSE that measures the energy required to remove a unit volume of a formation, the best performance is found at the global minimum point.*

Take the example of a heuristic search algorithm, as is used in Google Maps, where the origin and destination is fed into the algorithm and the algorithm uses a road network map to select the most optimal route based on available infrastructure in the state space. In this algorithm, a cost function and heuristic function can calculate the shortest direction to reach the destination, in the least amount of time depending on speed limits and where the algorithm detects road blocks or even live traffic jams. Treating the resulting performance of varying two control variables as a topographic environment, where no road infrastructure dictates the available routes, the ROP search algorithm should select the shortest path to reach the goal state from its initial starting point, as is illustrated in Figure 9.3 below.

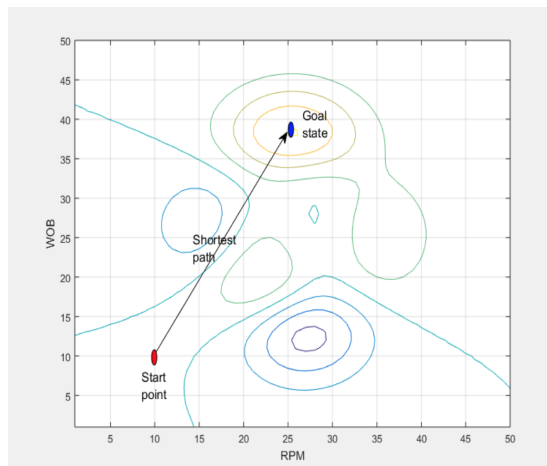


Figure 9.3: *Shortest path in the operational state space of the drilling rig to reach a theoretical goal state of the machine.*

There are however several challenges to perform an optimal ROP search. In the drilling scene, there is lacking information with regards to the destination that the machine needs to reach for maximum ROP and the most optimal route to get there. As is illustrated in Figure 9.2, with several local maximums in the state space, the machine needs to not only locate a peak, but the global peak to ensure that the highest ROP is achieved. Another challenge occurs when considering the route that the machine follows in the search. Perhaps would the shortest path in an operation cause the machine to drill with highly suboptimal controller set points; for instance a very high RPM combined with an unnaturally low WOB which can result in severe lateral vibrations and possibly also damage either the drill string, BHA or bit. A third challenge is the step size that the algorithm selects. By keeping the step size very small, the time to move from the start point to the goal state increases as the machine would need to not only vary the controller setpoint but also drill at those setpoints in order to measure the response. If the step size is too large however, this reduces the time required to reach the destination but increases the likelihood of possibly bypassing the destination (overshooting). In addition, several challenges exist in terms of selecting appropriate termination criteria and deciding which criteria should be used to initiate a new search.

9.1.2 Various search algorithms

Several search algorithms have been considered implemented on the rig. Two of the initial algorithms were stochastic and brute force search algorithms, in which

the rig can be configured to either drill a series of random (stochastic) parameter combinations or drill every combination of RPM and WOB (brute force) in the state-space, depending on the step size and operating range defined by the system boundaries. The rig could then evaluate which operational parameter combination from either of the search algorithms that resulted in the highest ROP performance, the lowest vibrations, and the lowest MSE. While a stochastic search can be quick to execute if only a few parameter combinations get evaluated, there is no guarantee that the stochastic operating parameter-combinations will locate the optimal rig performance. A brute force search can however certainly locate the best drilling parameters to use, but at the cost of significantly increased time required to run the algorithm. Given the limitations of stochastic and brute force algorithms, three other search algorithms got considered; Gradient descent, Fibonacci search and Golden section search.

Method of Steepest Descent

The Method of Steepest Descent, also known as Gradient search is one of the older methods that is still in use. It was developed in the early 1900s, but the principles of the technique dates back to Cauchy (Petrova and Solov'ev, 1997) [62].

This search method works by selecting an initial guess for a local minima. After this initial guess, one moves towards the gradient in order to locate the minimum point, as is illustrated in Figure 9.4.

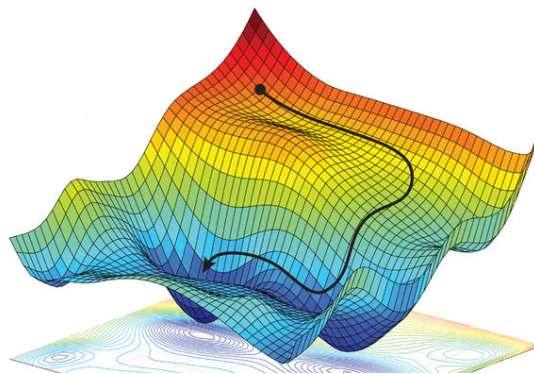


Figure 9.4: *Example illustration of the Gradient Descent method [63].*

The method is defined by the iterative equation (shown in equation 9.1), where in order to locate the minimum point, the product of the gradient multiplied by

the step size gets subtracted from the initial (last) position for each iteration.

$$\mathbf{x}_{k+1} = \mathbf{x}_k - \eta_k \mathbf{g}_k \quad (9.1)$$

\mathbf{x}_{k+1} denotes the minimum point to go towards and \mathbf{x}_k the initial position. The gradient is expressed by \mathbf{g}_k and η_k denotes the step size to move for each iteration (Luenberg and Ye, 2008) [64].

Fibonacci search

The Fibonacci search is a line search method where the objective is to locate a particular point on a non-linear line, for instance the highest or lowest point. This can be achieved either analytically or by performing searches along the line of investigation. The line search is designed for one dimensional objects, but can also be used for searching in higher dimensions. For higher dimensions, n line searches get conducted depending on n dimensions (Luenberg and Ye, 2008) [64].

When using the Fibonacci search, one must have a predefined interval in which the point to locate needs to be approximately determined from a unimodal function. The method only allows searching towards one goal point, such as for instance a maximum Figure 9.5 (Luenberg and Ye, 2008) [64].

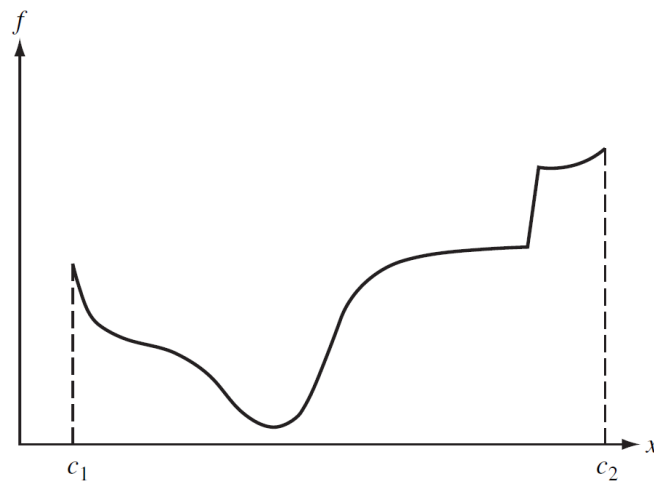


Figure 9.5: *Unimodal function needed for the Fibonacci search [64].*

From Figure 9.5, it becomes clear that the desired point, x_k , must lie on $c_1 \leq x_1 < x_2 \dots < x_{N-1} < x_N \leq c_2$, for N number of samples. Using this row of samples one can determine the uncertainty of the interval $[x_{k-1}, x_{k+1}]$. The width

of uncertainty, d_k , can then be found from equation 9.2:

$$d_k = \left(\frac{F_{N-k+1}}{F_N}\right)d_1 \quad (9.2)$$

where d_1 refers to the initial width of uncertainty ($c_2 - c_1$), F_k refers to the integers that are members of the Fibonacci sequence and can be found from 9.3 below, and F_N is the integer of the Fibonacci sequence (Luenberg and Ye, 2008) [64]:

$$F_N = F_{N-1} + F_{N-2} \quad (9.3)$$

Golden Section search

The Golden Section search is really just a modified version of the previous mentioned Fibonacci search, where rather than selecting a definite number of N sets, N is infinite. Using the correlation for the Fibonacci sequence relation (given by equation 9.3) one can find the solution to the Golden Section search (Luenberg and Ye, 2008) [64] from:

$$F_N = A\tau_1^N + B\tau_2^N \quad (9.4)$$

where A and B can be found using the initial conditions. τ_1 and τ_2 can be found from:

$$\tau_1 = \frac{1 + \sqrt{5}}{2} \text{ and } \tau_2 = \frac{1 - \sqrt{5}}{2}$$

When N becomes large the result on the right hand side of equation 9.4 becomes the dominant part, thus resulting in the following:

$$\lim_{N \rightarrow \infty} \frac{F_{N-1}}{F_N} = \frac{1}{\tau_1} \simeq 0.618$$

Transformation of equation 9.2 leads the following uncertainty:

$$d_k = \left(\frac{1}{\tau_1}\right)^{k-1}d_1 \quad (9.5)$$

From this equation, the following equation can be deduced:

$$\frac{d_{k+1}}{d_k} = \frac{1}{\tau_1} = 0.618 \quad (9.6)$$

and from this, we can conclude that the Golden Section search converges linearly with a convergence ratio of 0.618 (Luenberg and Ye, 2008) [64].

9.2 Implemented algorithms

9.2.1 Column-Row search (Implemented in 2018)

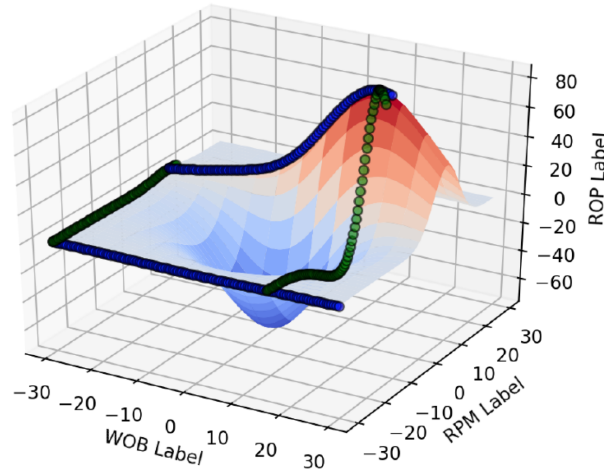


Figure 9.6: *Univariate hill-climb algorithm implemented in 2018 to search for the maximum ROP [11].*

In 2017 and 2018, multiple search algorithms were evaluated including brute force search algorithms, hill climb and so on. The selected algorithm was a univariate column-row search algorithm, that would vary one of the control parameters RPM or WOB up to five times subsequently, and after each iteration step the ROP response would get measured by determining the depth gradient (ROP) over a 15 second interval using a moving average filter. Then the algorithm would move over to vary the other control parameter up to five subsequent times, before the process got repeated. If the ROP-response from varying a control parameter would become lower than the response had been using the previous operating setpoint, the algorithm would revert to the previous setpoint and begin to vary the other drilling parameter right away. When the algorithm would no longer

The search algorithm will attempt to find a set of coordinates $(x_1^*, x_2^*, \dots, x_n^*)$, so that the controller setpoints become:

$$f(X^*) = \text{maximum} \quad (9.9)$$

Step 1: Determine the ROP gradient, i.e. the direction to go towards

The operational state space of the drilling machine constitutes the search space of the algorithm, satisfying the condition $X_{i,min} < X < X_{i,max}$, for instance $0 < RPM < 2000$ and $0 < WOB < 50$.

The depth gradient, referred to as instantaneous ROP (over a five second interval), is calculated using linear regression as can be shown in equation 9.10, where N represents the number of samples considered to obtain an instantaneous gradient and n is always ≥ 2 .

$$ROP_{instantaneous} = \nabla f(X_k) = \frac{N[\sum_{t=1}^n X_{k,t} \times f(X_{k,t})] - \sum_{t=1}^n X_{k,t} \times \sum_{t=1}^n f(X_{k,t})}{N \sum_{t=1}^n (X_{k,t})^2 - [\sum_{t=1}^n X_{k,t}]^2} \quad (9.10)$$

An objective function J can be defined for ROP using Euclidean norm, so that:

$$\min_X J(X) = \|f(X) - ROP_{setpoint}\|^2 \quad (9.11)$$

The ROP gradient vector can now get calculated from:

$$g(\vec{X}_0) = \nabla f(X_0) = \left[\frac{\partial f(X)}{\partial x_1}, \frac{\partial f(X)}{\partial x_2}, \dots, \frac{\partial f(X)}{\partial x_n} \right]_{X=X_0}^T \quad (9.12)$$

At any given combination of the control parameters, the ROP gradient always points to the maximal increase of the ROP function, and the gradient is always perpendicular to the ROP hyper-surface contour $f(X) = c$, where the constant c is an arbitrary real number.

From experiments conducted by Dunlop et al., field data from drilling operations

using poly-diamond crystalline (PDC) bits has been analysed to identify at which combination of RPM and WOB different types of vibrations occur. The constraints in state space that the drilling machine should operate in according to Dunlop et al. can be shown in Figure 9.9:

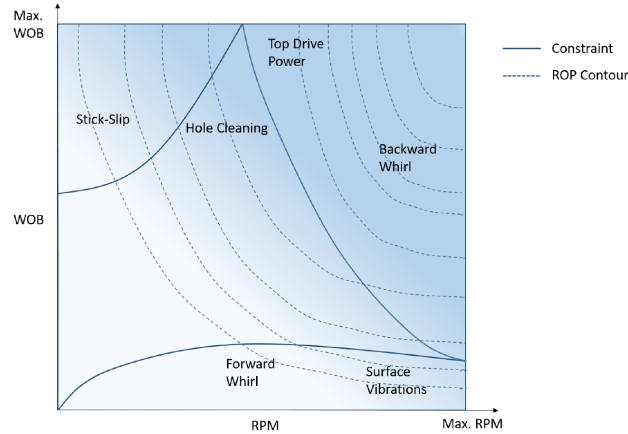


Figure 9.8: Constraints and ROP contours in the state space of the drilling machine [66].

Dunlop et al. furthermore suggest that the effect of flow rate on ROP is minimal, unless a mud motor is used downhole [66]

Step 2: Determine the learning rate (step size) in each iteration

From step 1, the algorithm is now capable of determining the search direction from a previous operating point. The optimization now is reduced to a univariate line search, moving along the local gradients. The gradient descent method can be shown as:

$$\vec{X}_{k+1} = \vec{X}_k - \eta g_k \vec{X}_k \quad (9.13)$$

η represents the learning rate; i.e. the step size to move in each iteration, and k denotes the iteration number. Ideally, the learning rate should be varied for each consecutive iteration, so that by selecting the most optimal η value, the ROP sequence $J(X^0) > J(X^1) > \dots > J(X^*)$ will converge to a minimum. The step size can be selected by the equation below, considering that we define

$\phi(\eta_k) = f(X_k + \eta g_k)$ and $0 < \epsilon < \frac{1}{2}$, which satisfies the Wolfe condition:

$$\phi(\eta_k) \geq (1 - \epsilon)\phi'(0) \quad (9.14)$$

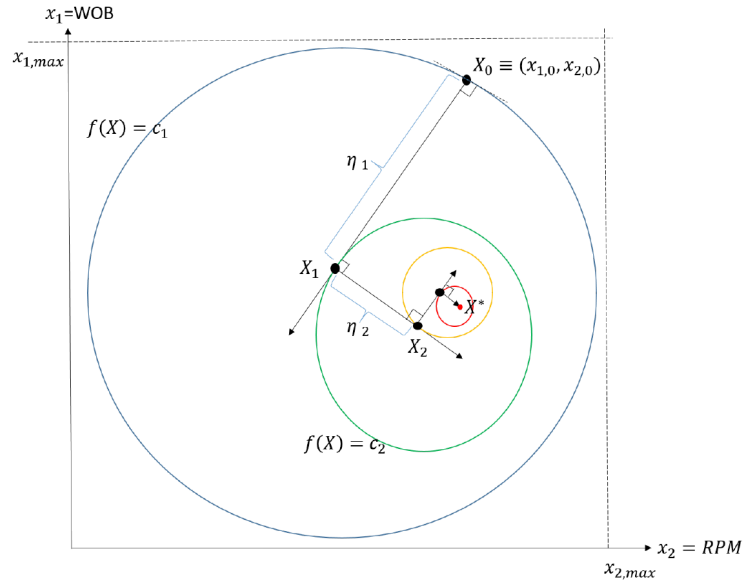


Figure 9.9: *Theoretical illustration of how the gradient descent algorithm should work to search for the maximum ROP in the state space of drilling parameters WOB and RPM.*

Step 3: Update the gradient descent algorithm

Then, we can update the equation for gradient descent:

$$\vec{X}_{k+1} = \vec{X}_k - \eta g_k(\vec{X}_k) \quad (9.15)$$

Where k corresponds to the iteration number.

Step 4: Define the termination criterion

The termination criterion to abort the search if the ROP increase at $k + 1$ is less than k can be defined by:

$$\|f(X_{k+1}) - f(X_k)\| \leq \delta \quad (9.16)$$

9.2.3 Triggers that can reinitiate search algorithm

In the past, a significant change in the ROP and MSE models have suggested that either; a new rock formation is encountered, or a considerable change in the drilling environment has occurred. For the gradient descent algorithm, triggers that can reset the algorithm and force a new search can be:

- the rock classification voting system suggests that a formation change has occurred in 70 % of the classifications over the last 10 seconds,
- either raw accelerometer data or downhole vibration model suggests that severe vibrations occur repeatedly in the system,
- for an inclined well, a section of more than 100 mm TVD has been drilled since the last search was terminated (if wellbore friction has changed the operating environment),
- the ROP gradient, which is continuously calculated even if the search algorithm is not pushing controller setpoints for RPM and WOB, increases so that $\nabla ROP_{k+1} \geq \nabla ROP_k * 1.5$.

For more information on the rock classification voting system and new formation detection see chapter 8

9.3 Digital Detailed Operating Procedure

The following digital detailed operating procedure (DDOP) has been developed to autonomously execute the drilling of a directional well. Please note that the complete code for the DDOP can be found in Appendix F.

The DDOP is arranged in eight phases, as visualized in Figure 9.10. These are:

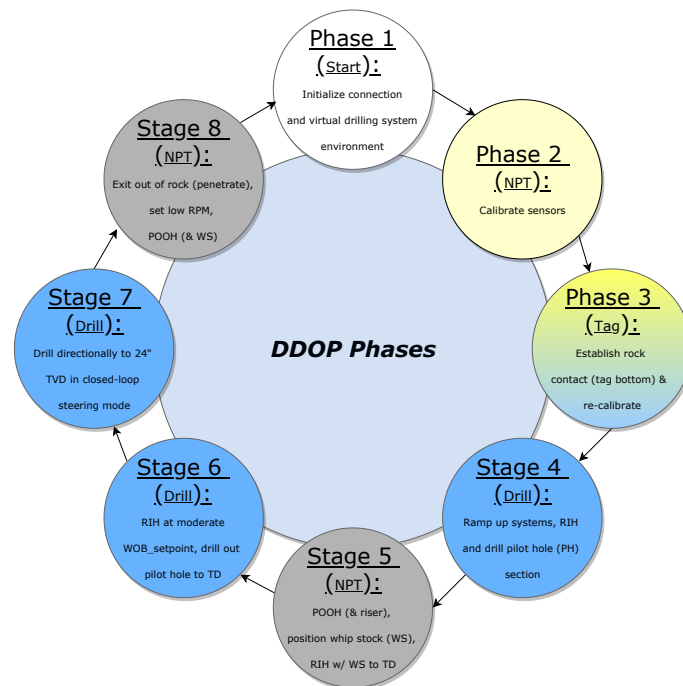
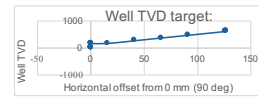


Figure 9.10: *Eight phases constitute the digital directional drilling operation.*

On the next page, a sequential draft is shown for the implemented DDOP to drill a directional well with the laboratory drilling rig.

Digital Detailed Operating Procedure (DDOP), UiS Drillbotics 2019
Autonomous Directional Drilling, 2.1 Nm drilling motor, 7 deg whipstock



#:	Phase	Label:	Description:	Systems Involved:	Sensor Reliance:	Bit target [mm]:	Inclination target [deg]:	Horizontal build	Well TVD target:
0	0	Initialize System.	Start start_system.bat file, open GUI's.	-	-	-	90	0	0
1	1	Start Autonomous Mode.	Establish contact with all PLC's and sensors (surface + downhole).	Hoisting, Pneumatic, Rotation	Surface (8x ch.), Downhole (10x ch.)	-	90	0	0
2	1	Initialize Virtual Environment.	Zero position tracker for downhole virtual environment, establish rig actuator control.	Hoisting, WPU	Downhole (10x ch.), Virtual rig	-	90	0	0
3	2	Calibrate Surface Sensors.	Zero all eight (8x) channels that log surface sensors.	-	Surface (8x ch.)	-	90	0	0
4	2	Calibrate Downhole Sensors.	Set Inclination = 90 deg, Azimuth = 0 deg, Accelerometers = 0 (Temperature is left unchanged).	-	Downhole (10x ch.)	-	90	0	0
5	2	Calibrate early Hook Load.	Define Hook Load = WOB_current so that Hook load (HL) $WOB = Z1 + Z2 + Z3$	-	Surface (3x ch. (z1, z2, z3))	-	90	0	0
6	3	Tag bottom.	Set WOB_setpoint = 3 kg. Activate PID controller until WOB >= 3 kg for 3x consecutive measurements. Set TVD = 0 mm	Hoisting	Surface (3x ch. (z1, z2, z3)), Virtual rig	0	90	0	0
7	3	Pull off bottom, reset Hook Load.	Lift off bottom to TVD = -5.00 mm, define Hook Load = WOB_current to that WOB = $Z1 + Z2 + Z3$ - HL	Hoisting	Surface (3x ch. (z1, z2, z3)), Virtual rig	-5	90	0	0
8	4	Ramp up Drilling Motor.	Set Solenoid Valve Opening so that bit RPM = 1070 RPM free rotating	Pneumatic, Rotation	Surface (1x ch. (P_pneum)), Virtual rig	-5	90	0	0
9	4	Tag bottom.	Set WOB_setpoint = 10-12 kg. Activate PID controller. Use Gradient descent method to toggle between 10 and 12 kg in increments of 0.5 kg to find highest ROP without risking damage to pilot hole.	Pneumatic	Surface (8x ch.), Downhole (10x ch.), Virtual rig	0	90	0	0
10	4	Drill pilot hole w/ ROP-agent.	Drill pilot hole to 175 mm MD + 5mm (clearance) ensuring high well integrity. WOB_max (gradient search) = 1 kg < knuckle joint bend force required to build inclination.	Hoisting, Pneumatic, Rotation	Surface (8x ch.), Downhole (10x ch.), Virtual rig	169.6	90	0	169.6
11	5	POOH to behind riser.	POOH to -243mm (423 mm above MD) with moderately high speed (5 m/min) with Solenoid Valve Opening so that bit RPM = 800 RPM to prevent overpull and stuck pipe.	Hoisting, Pneumatic, Rotation	Surface (8x ch.), Downhole (10x ch.), Virtual rig	-219.6	90	0	169.6
12	5	Close solenoid valve.	Set Solenoid Valve Opening = 0 % (0 RPM).	Pneumatic	Surface (1x ch. (P_pneum)), Virtual rig	-219.6	90	0	169.6
13	5	Position WS above hole.	Move WPU in X-direction away from rig for 128mm at 10 mm/s speed.	WPU	Virtual rig	-219.6	90	0	169.6
14	5	RIH with WS.	Lower WS into pilot hole for 175.5 mm (from original position 10mm above hole) at 3 mm/s speed and land WS 5 mm above TD.	WPU	Virtual rig	164.6 (WS)	90	0	169.6
15	5	RIH with bit (pre-configure)	Simultaneously to when whipstock gets moved, activate PID with WOB_setpoint = 4 kg and follow whipstock down until bit is 5 mm above rock surface (TVD = -5mm)	Hoisting	Surface (8x ch.), Downhole (10x ch.), Virtual rig	-5	90	0	169.6
16	6	Ramp up Drilling Motor.	Set Solenoid Valve Opening = X % (400 RPM free rotating, 200 RPM at maximum Torque).	Pneumatic, Rotation	Surface (8x ch.), Downhole (10x ch.), Virtual rig	-5	90	0	169.6
17	6	Expand pilot hole w/ ROP-agent.	Expand pilot hole to 135 mm depth (middle of bend section for whipstock) ensuring high well integrity. WOB_max (gradient search) = 5 kg and bit RPM = 1070 to prevent stuck pipe.	Hoisting, Pneumatic, Rotation	Surface (8x ch.), Downhole (10x ch.), Virtual rig	106	90	0	169.6
18	6	Drill to TD w/ ROP-agent.	Drill to below whipstock shoe (175 mm until in open hole with WOB_max = 3 kg to prevent getting stuck with BHA due to insufficient hole cleaning and reaming at ROP.	Hoisting, Pneumatic, Rotation	Surface (8x ch.), Downhole (10x ch.), Virtual rig	169.6	83	15.870752	169.6
19	7	Activate ROP-agent + closed-loop steering.	required bend to reach digital well trajectory. E.g. If > trajectory (overbuild), reduce WOB slightly. If < trajectory (underbuild), increase WOB slightly. Maximum WOB for system is 18 kg when operating with the 2.1 Nm downhole motor.	Hoisting, Pneumatic, Rotation	Surface (8x ch.), Downhole (10x ch.), Virtual rig	269.6	83	40.824763	269.6
20	7	Reset ROP-agent.	Reset ROP-agent to moderate parameters in center of state space (resonance, if agent has not been reinitiated by formation / environment change in past 100 mm drilled. Currently, none of the previously described criteria for termination have been implemented).	-	Surface (8x ch.), Downhole (10x ch.), Virtual rig	269.6	83	40.824763	269.6
21	7	Activate ROP-agent + closed-loop steering.	required bend to reach digital well trajectory. E.g. If > trajectory (overbuild), reduce WOB slightly. If < trajectory (underbuild), increase WOB slightly. Maximum WOB for system is 18 kg when operating with the 2.1 Nm downhole motor.	Hoisting, Pneumatic, Rotation	Surface (8x ch.), Downhole (10x ch.), Virtual rig	369.6	83	65.778775	369.6
22	7	Reset ROP-agent.	Reset ROP-agent to moderate parameters in center of state space (resonance, if agent has not been reinitiated by formation / environment change in past 100 mm drilled. Currently, none of the previously described criteria for termination have been implemented).	-	Surface (8x ch.), Downhole (10x ch.), Virtual rig	369.6	83	65.778775	369.6
23	7	Activate ROP-agent + closed-loop steering.	required bend to reach digital well trajectory. E.g. If > trajectory (overbuild), reduce WOB slightly. If < trajectory (underbuild), increase WOB slightly. Maximum WOB for system is 18 kg when operating with the 2.1 Nm downhole motor.	Hoisting, Pneumatic, Rotation	Surface (8x ch.), Downhole (10x ch.), Virtual rig	469.6	83	90.732787	469.6
24	7	Reset ROP-agent.	agent has not been reinitiated by formation / environment change in past 100 mm drilled. Currently, none of the previously described criteria for termination have been implemented.	-	Surface (8x ch.), Downhole (10x ch.), Virtual rig	469.6	83	90.732787	469.6
25	7	Activate ROP-agent + closed-loop steering.	Set ROP-agent to free operation, with WOB setpoint adjusted accordingly to required bend to reach digital well trajectory. E.g. If > trajectory (overbuild), reduce WOB slightly. If < trajectory (underbuild), increase WOB slightly. Maximum WOB for system is 18 kg when operating with the 2.1 Nm downhole motor.	Hoisting, Pneumatic, Rotation	Surface (8x ch.), Downhole (10x ch.), Virtual rig	609.6	83	125.6684	609.6
26	8	Penetrate rock.	Drill through rock by 5 mm to 614.6 mm TVD to ensure successful penetration. Successfully penetrated rock is detected by separately implemented algorithm.	Hoisting, Pneumatic, Rotation	Surface (8x ch.), Downhole (10x ch.), Virtual rig	614.6	83	126.9161	614.6
27	8	Deactivate ROP-agent.	Set ROP-agent = False so that PID control is disabled and RPM is kept until next stop.	Hoisting	Surface (8x ch.), Downhole (10x ch.), Virtual rig	614.6	83	126.9161	614.6
28		Set low RPM.	Set Solenoid Valve Opening = 80 %.	Pneumatic	Surface (8x ch.), Downhole (10x ch.), Virtual rig	614.6	83	126.9161	614.6
29		POOH to behind whipstock.	POOH to 50 mm MD (with moderate high speed: 5 m/min).	Pneumatic, Hoisting, Rotation	Surface (8x ch.), Downhole (10x ch.), Virtual rig	-60	90	0	614.6
30	8	Stop all systems controlled.	Stop all active systems, export drilling data to local folder (User then pushes data to database w/ script).	-	-	-	90	0	614.6

Figure 9.11: Digital Detailed Operating Procedure for Directional Drilling, developed in the design phase in early 2019.

9.4 Downhole closed-loop steering

The concept of closed-loop steering in the control system is currently limited to inclination steering through active management of knuckle joint bend in the BHA (through WOB setpoint manipulation). For every 1 mm MD that gets drilled below the whipstock KOP (at 108.908 mm for the 10 degree whipstock), the relationship between horizontal build and TVD gets calculated so that:

$$\frac{H}{TVD}(MD > 108.908mm) = \frac{horizontalbuild[mm]}{TVD[mm]} \quad (9.17)$$

A look up table has been created which contains the $H/TVD_{planned}$ values for either a 5, 7 or 10 degree whipstock. This is illustrated in Figure 9.12 which shows a section of the downhole environment GUI with actual values for the well trajectory and pre-programmed values for the planned build. The look up table values are then called every mm, and the relationship between the actual build and planned build is checked by the equation:

$$Status_{build} = \frac{H/TVD_{actual}}{H/TVD_{planned}} \quad (9.18)$$

If the build status is ≥ 1 , the system will be in a *drop angle* mode, and if the build status is ≤ 1 , the machine will be in a *build angle* mode and needs to increase the build-rate by increasing the WOB.

Pre-programmed trajectory: (10 deg whipstock)					Wellbore trajectory: (real-time)					Build/Drop	
IVD:	MD:	Incl.:	Hor.:	H/TVD:	IVD:	MD:	Incl.:	Azim.:	Hor.:		H/TVD:
0 mm	0	90	0	0	0 mm	0	151	15	0	0	- (< 108.908mm)
30 mm	30	90	0	0	30 mm	44.3	90	349	-0.02	0	- (< 108.908mm)
60 mm	60	90	0	0	60 mm	60.05	90	359	0	0	- (< 108.908mm)
90 mm	90	90	0	0	90 mm	91.25	90	358	0.01	0	- (< 108.908mm)
108.9mm	108.9	90	0	0	108.9mm	122.65	89	355	0.46	0	- (< 108.908mm)
150 mm	150.6	80	7.136	0.0476	150 mm	153.7	77	360	6.59	0.04	Drop
180 mm	181.1	80	12.345	0.0686	180 mm	185.05	77	356	13.83	0.07	Build
210 mm	211.6	80	17.554	0.0836	210 mm	216.15	78	7	20.52	0.09	Drop
240 mm	242	80	22.764	0.0948	240 mm	247.5	79	349	26.55	0.11	Drop
270 mm	272.5	80	27.973	0.1036	270 mm	278.65	83	348	30.52	0.11	Drop
300 mm	302.9	80	33.183	0.1106	300 mm	310	83	348	34.19	0.11	Build
330 mm	333.4	80	38.392	0.1163	330 mm	341.4	83	354	37.89	0.11	Build
360 mm	363.9	80	43.602	0.211	360 mm	372.6	80	351	42.81	0.11	Build
390 mm	394.3	80	48.811	0.1252	390 mm	403.7	80	5	48.14	0.12	Build

Figure 9.12: Illustration of how the machine uses a look up table to confirm whether or not sufficient build is achieved at any given depth.

Based on analysis of experiments that have been conducted regarding build rate by adjusting the WOB, the following ranges for weight on bit have been identified: $WOB_{dropInclination} = (5, 12)$ [kg] and $WOB_{buildInclination} = (12, 18)$ [kg]. While downhole RPM only depends on the gradient search algorithm and the different phases of the operation, the voting system in the control system will continuously evaluate whether or not the system is in a build or drop mode. If for example the ROP gradient search proposes 14 kg WOB, but the system should be in a drop mode, the $WOB_{setpoint}$ will be overwritten to 12 kg, which is the highest permissible WOB in the drop mode. While a higher WOB from experiments have yielded a higher ROP, another control algorithm exists that evaluates whether the recommended controller setpoint (within the range identified for build or drop mode) gets forwarded to the PID controller, or whether an even lower setpoint should be used in the event of severe downhole vibrations or if the system is in a critical phase (such as passing KOP with the components in the BHA that have the highest OD).

With regards to azimuth control, the top drive can be configured to move a certain number of steps through pulsing. While implementing active azimuth-steering can be highly beneficial to the current laboratory setup, a challenge exists with regards to the accuracy of the magnetometer sensor that is currently installed in the sensor sub. Such challenge is likely the result of the sensor house being made from a ferromagnetic material, and a possible solution can be to use a non-ferrous

metal in the future.

9.5 Drilling Incident Detection

Stuck pipe detection

The gradient descent algorithm has been programmed to only select WOB setpoints that are positive (to prevent the algorithm from calculating a new gradient when POOH). The *Build_{angle}* and *Drop_{angle}* modes have lower limitations of 12 and 5 kg, respectively. Since upwards movement is only allowed if the PID controller calculates an overshoot of (WOB setpoint + 1 kg), there exists a risk that the PID controller stops to move if either bit or BHA gets stuck in the well and the overshoot was less than the defined maximum allowable.

For this reason, a stuck pipe detection model, along with a remedial action plan has been implemented. The model is only activated if the BHA gets stuck inside of the inclined well section, and evaluates whether the actuators that control drill floor (and bit) elevation have moved less than 0.2 mm in either direction over the last 15 seconds. Upon stuck pipe detection, the WOB setpoint is set to -5 kg (5 kg overpull allowed) until the assembly has been pulled 1 mm above the stuck point, in which the gradient search algorithm and build vs drop modes will continue to operate (normal drilling state commences).

Downhole vibrations detection

Downhole vibrations get detected by a machine learning model that has been implemented as its own program in the control system layers. The model has been developed as described in chapter 7.

Leak detection

A leak gets detected if the system detects three consecutive measurements of pressure less than 3 bar (in the pneumatic line), and an active drilling phase is on-going. Hence, leak does not get detected if for instance the whipstock is being lowered into the well, and the solenoid valve that regulates air flow to the downhole motor is closed which results in a 0 bar pressure (NPT phase). Such incident can not get handled by the system, and thus require a controlled shut-down of the system.

Overpressure detection

Similarly to leak detection, overpressure in the system is detected if three consecutive measurements of pressure in the pneumatic line exceeds 6.5 bar (0.5 bar safety margin above the pressure that the downhole motor can sustain due to pressure drop). Such incident can also not get handled by the system; hence a controlled shut-down procedure is implemented if overpressure is detected.

Twist off detection

Twist off detection is handled if three criteria are all met concurrently; a drilling phase is on-going, a leak incident is triggered and WOB drops to a significantly lower weight (varied between operations) than that being the current setpoint. Twist off additionally must get confirmed by all three criteria for three consecutive measurements to shut down the system.

Axial vibrations

Axial vibrations get detected if load cell measurements of the hook load (WOB measurements) are $1.5 \times$ higher than the $WOB_{setpoint}$. The concept of detecting axial vibrations is particularly difficult for directional drilling, since the bendable knuckle joint is present in the BHA, and the downhole motor gets used. Ideally, it would not be necessary to monitor axial vibrations on the rig, but when a downhole motor that can only sustain a maximum force of 380 N (axially on the shaft) gets used, it is important to detect such vibration levels. Since the maximum permissible load is 380 N, the maximum WOB of the system is set to 18 kg (177 N), which means that all forces that exceed 266 N (well below the limit of the motor) get detected.

Although not implemented, a counter can easily be implemented in the control system to measure the number of times that a force above 265 N have occurred, and stop the system if such forces occur a certain amount of times or within a set time interval.

9.6 Rig performance

In this section, five well logs of a short inclined section, a vertical pilot hole, a complete inclined wellbore section and two complete wells (separated into pilot hole and inclined section plots) are presented. The plotted figures have been downsampled to 1 Hz (from a 60 Hz sampling frequency). To smoothen the plots for inclination, azimuth and weight on bit, a Kalman filter has been used.

9.6.1 Experiment 1: Inclined well section

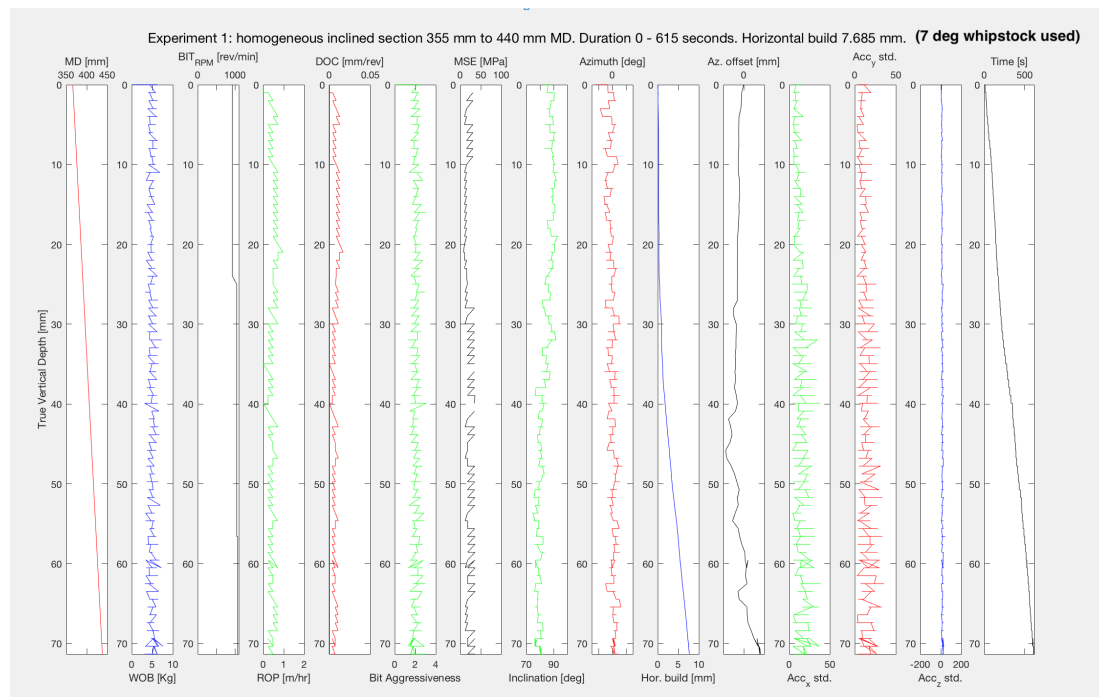


Figure 9.13: *Experiment 1: Well log representing rig performance when drilling with downhole motor in inclined section using a 7 degrees whipstock.*

The first well section with logging tools in the BHA was drilled from 355 mm MD to 440 mm MD in homogeneous cement, using a 7 degree whipstock (Figure 9.13). The WOB setpoint is kept constant at 5 kg, while the RPM setpoint is varied between 925 and 1075 revolutions per minute. The resulting ROP is approximately 0.4 to 0.5 m/hr, which is considered a good result given the increased wellbore friction and a low WOB setpoint. WOB setpoint has from experiments conducted in the past been assessed to be the most important drilling parameter affecting the ROP. The MSE varies between 10 and 40 MPa, and was calculated by hardcoding a constant torque value equal to 1 Nm (since the top drive is turned off during drilling with the downhole motor for less noisy data; resulting in a zero torque

measurement in the motor encoder). From the figure, it can be observed that even with high oscillations in the downhole sensor measurements, approximately 7.5 mm horizontal build is achieved, with an offset on the azimuth of approximately 1 mm. In Figure 9.14 below, a cross-section of the first deviation well that was drilled successfully is shown after a waterjet cutting machine was used to split the rock sample.

Deviation well 1, homogeneous cement, 7 degrees whipstock

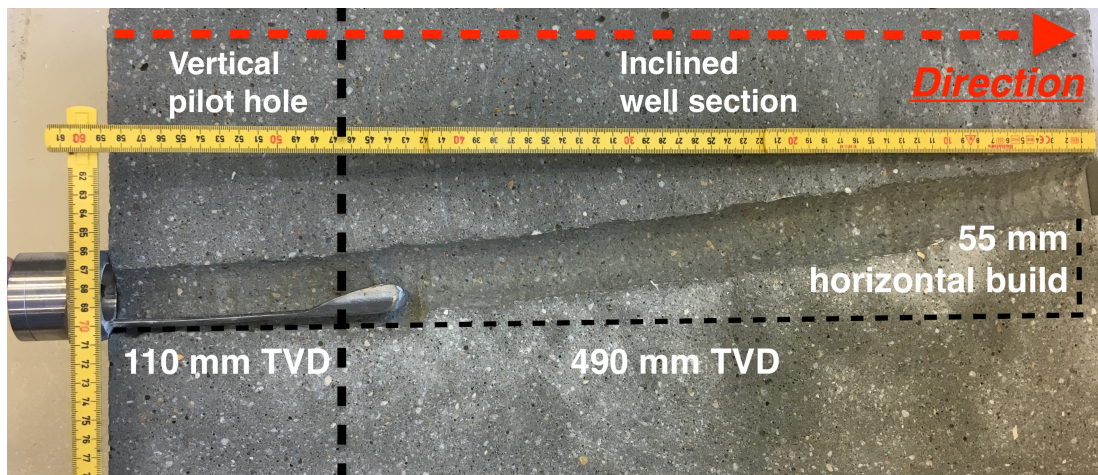


Figure 9.14: *Well profile using 7 degrees whipstock in homogeneous cement.*

From inspection of the rock, approximately 55 mm horizontal build was achieved over 490 mm TVD of drilling below KOP. In the last 150 mm of drilling, the knuckle joint turned slightly due to a twist in the connection between the knuckle joint and sensor sub, causing an azimuth offset of approximately 10 mm. From evaluating the well integrity, it is probable that bit whirl has occurred, possibly caused by the undergauge BHA. It is however difficult to monitor and confirm bit whirl with the current configuration.

9.6.2 Experiment 2: Vertical well section

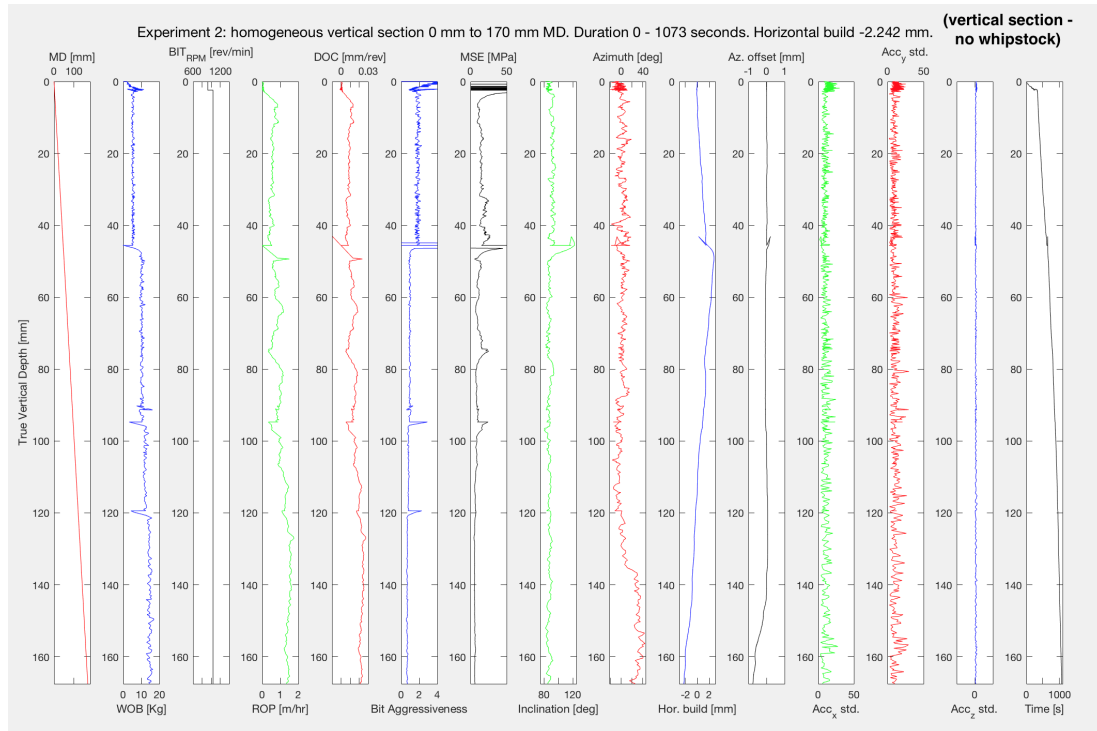


Figure 9.15: *Experiment 2: Well log representing rig performance when drilling with downhole motor in vertical pilot hole section for 170 mm TVD to later RIH with whipstock. Riser is used to ensure verticality.*

The second well log, as shown in Figure 9.15, showcases a pilot hole section that was drilled to evaluate the performance that can be achieved with high drilling parameter setpoints in the first vertical section of the competition well. RPM is kept nearly constant at 1040, and the WOB is increased from first 5, to 10 and later 15 kg. As the WOB gets increased, the DOC increases from approximately 0.01 (5 kg WOB) to 0.015 (10 kg WOB) to 0.025 (15 kg WOB) mm/rev, which results in an ROP increase from approximately 0.5 m/hr to 1.5 m/hr at most. While the increased WOB lead to an improvement in ROP, a slight build can be observed (according to measurements and visual inspection), which is possibly caused by the knuckle joint starting to bend at high WOB. As can be recalled from section 9.4, the range for $WOB_{BuildInclination}$ starts at approximately 12 kg (up to maximum 18 kg). While the MSE also for this well log is calculated with 1 Nm torque constantly, it can be observed that the MSE is reduced as the WOB increases, suggesting less energy is wasted, and a higher percentage of the combined energy usage goes into bit-rock interaction.

9.6.3 Experiment 3: Deviation well with WOB 5 to 20 kg

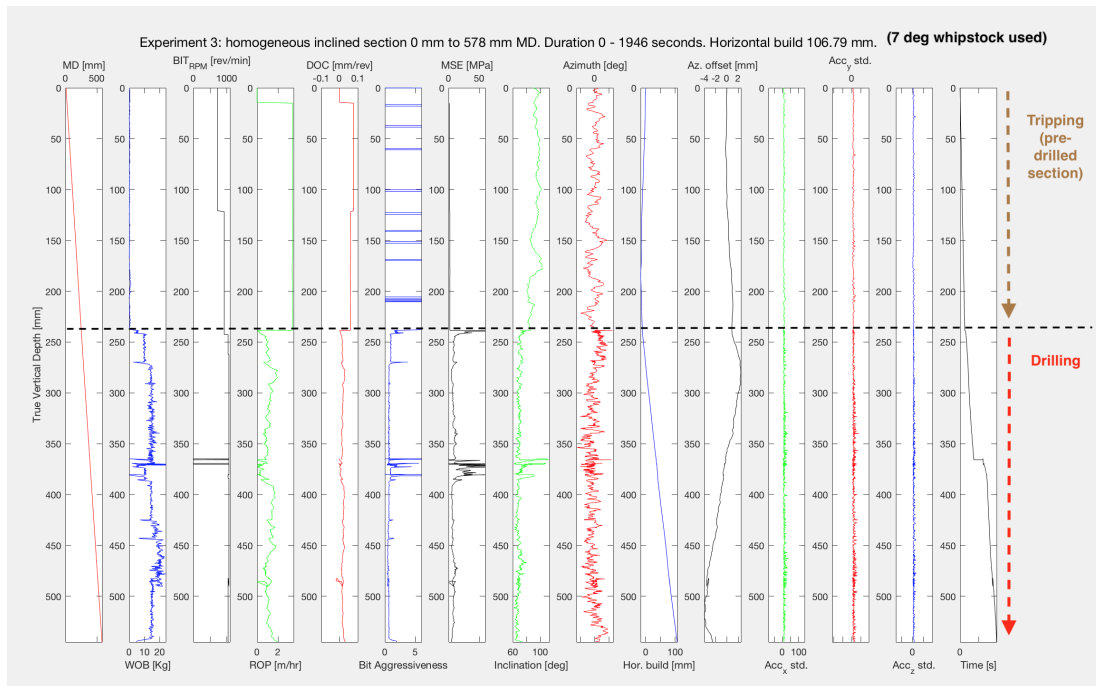


Figure 9.16: *Experiment 3: Well log representing rig performance when RIH to 240 mm TVD and then drilling with downhole motor to 600 mm TVD. A 7 degrees whipstock is used to kick off.*

Continuing to vary the WOB, the same approach as before is used in an inclined section in Figure 9.16. For the first 235 mm, the bit is RIH, With a 5 kg WOB setpoint and an RPM of between 730 and 1040 to pass the KOP where a 10 degree whipstock has been inserted. The previously drilled well section that the whipstock had been landed in got drilled as shown in **Experiment 2** above. Contrary to the last experiment, it is observable that while the ROP continues to increase with an increase in WOB setpoint, the MSE does not decrease. A possible explanation to this is that increased friction between the non-rotating BHA and the wellbore in the inclined section leads to a lower ROP (approximately 1 m/hr) than expected. Horizontal build is calculated from measurements to approximately 100mm, with an offset on the azimuth of approximately 3 to 4 mm (occurs at the end of the inclined section). Upon inspection of the well, approximately 45 mm horizontal build was achieved. Possible explanations of both the MSE, over-estimated horizontal build and only a moderately high ROP can be that; either the increased wellbore friction reduces the overall efficiency of the drilling operation and that constant vibrations against the wellbore influence the measurements for inclination calculation. Another possible explanation can

be that only half of the BHA is submerged inside of the well, leaving the BHA even more unconfined, which in turn affects the vibration cycle of the entire assembly.

9.6.4 Experiment 4: Pilot hole section 0 - 166 mm MD. Inclusion well from 109 mm to 600 mm MD.

In Figure 9.17 and Figure 9.18, a complete well was drilled, first drilling to 166 mm MD using the riser for vertical pilot hole, and then lowering a 10 degree whipstock into the well before the bit kicks off. For the pilot hole, the first 65 mm are drilled with a WOB setpoint of 10 kg, before 15 kg is used for the remaining parts of the well-section. As has been observed from previously conducted experiments, the ROP increases from approximately 0.5 m/hr to 1.5 m/hr with an increase in WOB, and the MSE decreases as a result. In this pilot hole, the horizontal build and azimuth offset plots are lacking, with constant values showing (due to an error in calculations since the downhole sensor was calibrated erroneously).

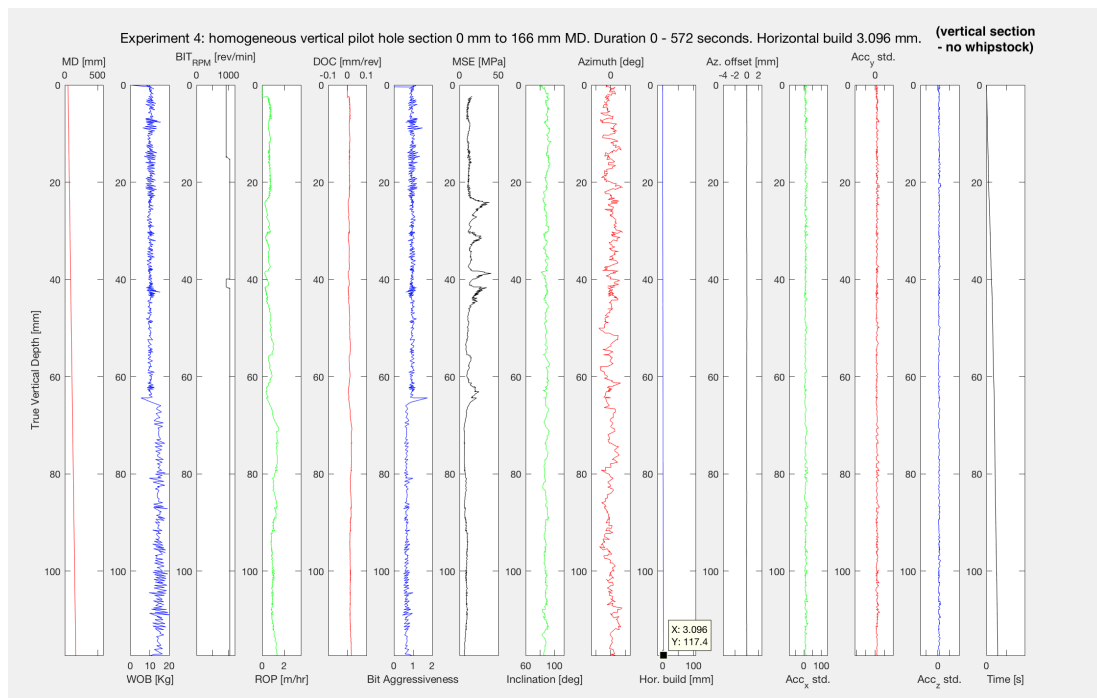


Figure 9.17: Experiment 4: Well log, downhole motor drilling of 166 mm TVD pilot hole.

When the 10 degree whipstock is used, an ROP is achieved of approximately 1 to 1.25 m/hr in the inclined section when WOB is kept at either 10, 15 or 18 kg WOB. In terms of the calculated horizontal build from 167 mm to 600 mm MD, approximately 58 mm is measured with the downhole sensor, while the true build

is approximately 42 mm. A theory, that can possibly explain the low horizontal build is that the small cross-over between the bit and pneumatic motor allows the downhole motor to return to a near-vertical trajectory just below the whipstock, resulting in a rathole and a much lower build at KOP than 10 degrees. Once the cross-over has passed the end of the whipstock, and the downhole motor with a larger OD passes the whipstock, the bit will get forced towards a steeper dogleg than was initially achieved, but still lower than the whipstock dogleg at KOP. It is possible that a higher horizontal build got achieved with the 7 degree whipstock, since such an effect of the bit returning to near-vertical once the cross-over is rotating inside of the whipstock would get limited. Another explanation is that one of the cutters on the PDC bit received from Baker Hughes was observed to be chipped after the operation, which is considered the most likely cause of the reduction in ROP.

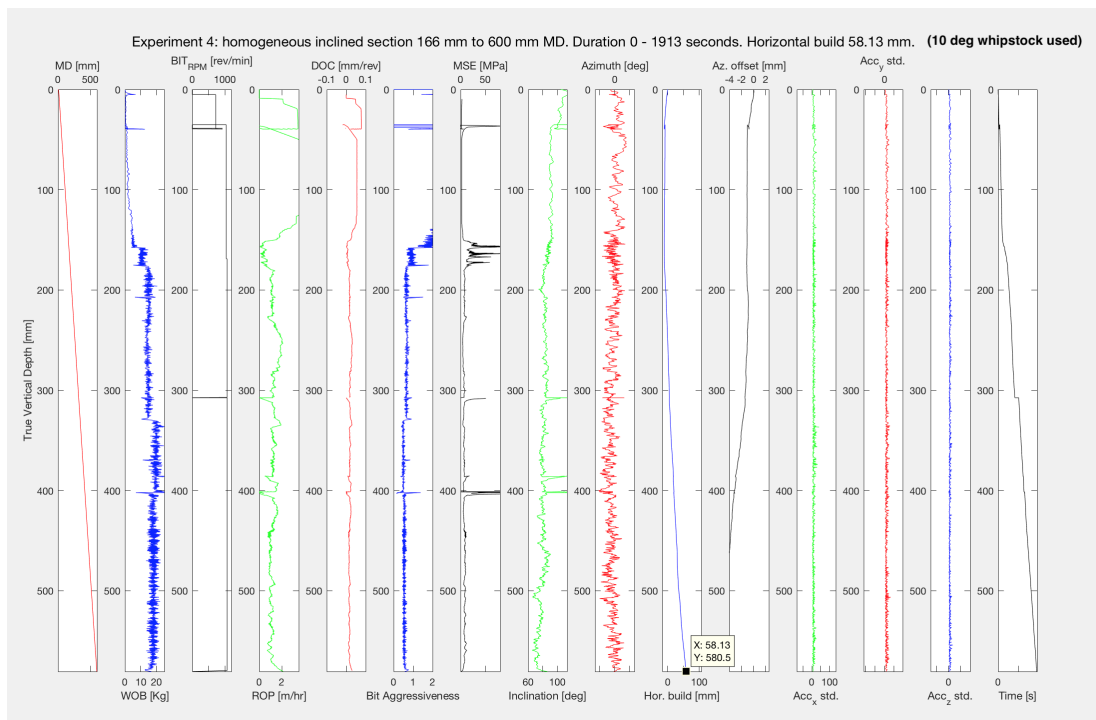


Figure 9.18: *Experiment 4: Well log representing rig performance when RIH to 109 mmTVD (whipstock KOP) and then drilling with downhole motor to 600 mmTVD. A 10 degrees whipstock is used to kick off.*

9.6.5 Experiment 5: Increasing cross-over OD to 30.75 mm (up from 20)

From analysis of the data in Experiment 4, it was evident that the horizontal build using the 10 degree whipstock (measured to 42 mm) was lower than in previous

runs where 55 mm horizontal build was achieved using a 7 degree whipstock. A possible explanation was that by increasing the bend at KOP, the bit would, as the PDC cutters on the bit passed the edge of the whipstock, produce a rat hole below the whipstock and go as far back to vertical as possible since the cross-over OD was only 20 mm (down from the bit OD at 31.75 mm). For this reason, a new cross-over was machined, which had an OD of 26 mm (approximately equal to the downhole motor OD of 25 mm when the 1 mm OD sleeve is attached). The hypothesis with the new cross-over was that as soon as the bit would pass the whipstock, the cross-over would keep the bit in place and act as a stabilizer for the first 3 cm that the bit drilled below whipstock-shoe. Furthermore, once the cross-over would pass the bit, and the OD of the BHA would reduce from 30.75 mm (cross-over OD) to 20 mm (pneumatic motor OD), the cross-over would ensure that the bit could not begin to drill back to vertical (90 degrees inclination), since the cross-over would now continue to act as a stabilizer in the open hole section below whipstock.

Unfortunately, two of the cutters in one of the two available PDC bits broke during test drilling, forcing the team to test the larger cross-over using first the chipped bit, and then a 3-cone PDC bit. From the performance, approximately 5 cm of build was achieved with the chipped bit, up from 4 cm using the reduced-size crossover. Due to the bit and BHA (crossover, downhole motor, knuckle joint and sensor sub) now being on gauge, a slight drop in ROP was noted, compared to previous runs where the ROP has stabilized at around 1.5 m/hr. It is expected that a new drill bit, combined with the larger cross-over, will provide a consistently high ROP in the inclined section, and a horizontal build of at least 6.5 cm when using a 7 degree whipstock, and approximately 7 to 8 cm build when using a 10 degree whipstock.

Chapter 10

Conclusion and Future Research

10.1 Discussion of results and end state achieved

10.1.1 Machine Learning

An end-state has been achieved where in total six different rock formations can successfully get classified on the laboratory drilling rig; using machine learning models developed with data that got collected in early 2019. The predictions from the seven machine learning models that have been used for formation classification can be combined through the use of a voting system in order to present the output prediction along with a confidence level. Furthermore, a sufficiently high confidence level can be used to detect that a new formation has possibly been encountered, and if a new formation has been detected sucesfully over a number of consecutive iterations, the new formation can be confirmed, allowing the autonomous control system to initiate either a new search for an optimal ROP or to use pre-determined drilling parameters for WOB and rotational speed, based on analysis of previous runs.

Similarly, three rig operations and several drilling incidents such as stuck pipe, pressure outside of the normal operating range and heavy vibrations (either measured with downhole or surface equipment) can be classified on the laboratory drilling rig. While preliminary results have shown that the models that have been developed using field data from Volve can also be used for formation classification and rig operation classification, it is difficult to assimilate the models developed for the laboratory drilling rig and the full scale, likely due to the physics of the two different systems being different from each other.

While the models up to this point have only been developed using supervised machine learning (and unsupervised to organize and evaluate the effectiveness of individual features), reinforcement learning is likely the next step to make the predictions even more accurate. Except for the downhole vibrations model, all models have been built using surface sensor measurements, due to the late completion of the downhole sensor sub and integration in the control system.

10.1.2 Control System and Control Algorithms

The control system on the drilling rig has been developed makes use of multithreading, in which multiple threads, or processes, get executed simultaneously with a unidirectional data flow. The developed control system architecture currently runs on a CPU with twelve cores, allowing for up to 24 different threads to run in parallel to each other. The threads, or modules, that all consist of a client and a server in each thread, communicate through the gRPC API / framework, and is currently organized in six layers, in which each layer consists of a core thread and several child threads. The use of multithreading for the laboratory drilling rig has increased the robustness of the drilling rig heavily. Furthermore, the use of gRPC allows different modules, or threads, that have been developed using different programming languages to get combined. Proof of concept for this has been obtained by connecting the high frequency DAQ that runs in C Sharp on the .NET framework to the control system that has been developed in Python.

The resulting control system is capable of executing a series of control algorithms and models in parallel, such as a directional drilling program, ROP search (by gradient descent method), inclination steering, downhole vibrations, incident and stuck pipe detection (and handling), and so on. This is possible through the use of another voting system that prioritizes pending commands from the control algorithms depending on which action is most critical to execute. While tuning of PID controller and gradient descent method is difficult with a bendable BHA developed to be used in inclined well sections, proof of concept has been obtained by the algorithms selecting optimal drilling setpoints based on measured rig performance and response to earlier changes executed.

The operating frequency of the control system is currently configured to be 60 Hz, due to the sampling rate from the microcontroller in the sensor sub being the bottleneck for higher sampling frequencies. While a bottleneck in developing the control system has been the Arduino Due and Mega microcontrollers that execute

commands to the actuators, top drive, pumps, valves and so on, new PLCs have been purchased for future upgrades.

10.1.3 Mechanical

During this semester, a series of mechanical upgrades have been implemented on the rig, to enable directional drilling with closed-loop steering; using both downhole and surface sensors. The most important upgrades are;

- installation of a complete pneumatic system with compressor, several manual and automatic valves and chokes to throttle the air flow to the downhole motor that has been selected according to several critical specifications to function on the laboratory drilling rig,
- designing, prototyping, 3D printing and optimizing a fully functional sensor sub equipped with downhole sensors to measure acceleration forces, inclination and heading of the BHA (and bit),
- development of a knuckle joint, capable of bending to 10 degrees in the BHA to relief the drill pipe of bend stress,
- designing, developing and testing different whipstock solutions on the small-scale, before the most optimal whipstock design has been integrated on a 2-axis positioning system,
- tuning of the top drive,
- and finally, calibrate all parts of the system to allow for complete autonomous operation based on a developed digital detailed operating procedure presented in chapter 9.

The sensor sub, whipstock and knuckle joint were developed in Fusion 360 (CAD) and outsourced to a local manufacturer for 3D printing in 316L stainless steel.

A pneumatic downhole motor was selected for three main reasons; to reduce the friction between the BHA and wellbore in inclined well sections (reducing the likelihood of stuck pipe), to keep the pipe stationary with no rotation so that the knuckle joint can provide the desired bend during directional drilling, and finally to allow for consistently high quality in downhole measurements. Rotating the pipe and sensors would both distort the signals and introduce centrifugal forces that needs to be accounted for.

Results from drilling with the downhole motor have shown that the vibrations in the entire system are significantly reduced when compared to the old system where a conventional top drive would rotate the complete string assembly. The reduction in vibrations furthermore indicates that instead of the energy being lost to reaming the wellbore, most of the energy goes into bit / rock interaction. One can also observe from the well logs presented in section 9.6 that the MSE is decreasing when the WOB is increasing when less wellbore friction is encountered. More importantly, there have been no twist offs in the pipe connection attaching the pipe to the top drive.

The well integrity has improved significantly after the conventional top drive was replaced with a downhole motor. Because the only rotating part of the system now is the bit, there is less reaming by the BHA during drilling and tripping. It can still be observed that bit whirl continues to be a challenge. The bit whirl is likely caused by the combination of high rotational speeds and no stabilizers in the BHA; allowing the bit to move laterally.

Due to less reaming of the wellbore, the chance of getting stuck in a tight open hole section is higher, especially at KOP where the whipstock deflects the bit away from vertical. There have also been instances when stuck pipe has occurred during drilling. Due to this, selection of components that go into the well is extremely important; such as cross over between the bit and motor, running into hole with or without stabilizers and so on. A large cross over might lead to a stuck pipe when high WOB is applied, since high WOB increases the build-rate in the inclined section. RIH with a undergauged BHA can also be problematic, due to the risk of creating a significant rathole below the whipstock where the bit or BHA can get stuck when POOH.

10.2 Future recommendations

10.2.1 Machine Learning

The developed approach of pre-processing the data, selecting the most optimal features and developing multiple models along with a voting system has resulted in reliable results. Future recommendations are:

- Integration of reinforcement learning on the rig, in which the models con-

stantly get improved by *correction of* the prediction outputs from models,

- developing a larger database containing both different rock formations drilled while varying drilling parameters and a collection of drilling incidents,
- making use of feature dimensionality reduction methods such as PCA, as shown in chapter 6 in order to extract the principal components that explain the variance in the dataset and in order to develop the models,
- develop models and perform PCA based on downhole measurements or surface measurements that accurately describe the bit interaction with the formations.

10.2.2 Control System and Control Algorithms

For the control system, several future improvements have been identified:

- make use of so-called IP Multicast to send a data package from a server to multiple clients without sending the data each time that a client subscribes to it (Multicast) and possibly also Data Distributing Service (DDS),
- since the threads communicate through the gRPC API, use the Go programming language to write some of the modules,
- ensure that the libraries that get used in for instance filters (median, average and so on) are the least demanding in terms of computational loads, and
- minimize the use of loops (for and while loops particularly) to reduce the computational load.

In terms of control algorithms, the suggested practice for a complete closed-loop system is that a database gets developed that contains *best practice* drilling parameters for the different formation types, so that when a new formation type gets confirmed, the most optimal drilling parameters can immediately get selected. The rig can then make use of various search-algorithms in the reduced state-space (due to previously established most optimal parameters) to identify the point at which the Founder point of the system is found. This point describes the optimal performance point when drilling with optimal conditions related to WOB and RPM, and the point is a linear function between these two parameters so that when one of the two increases or decreases, equipment might get damaged [67]. Once the Founder point has been located, for instance by gradient descent method, a series of drilling incident models can be used to not only classify the occurrence of a drilling incident, but also if possible, predict that an incident is imminent

(unless measures get taken). Gradient descent method will likely perform significantly better once proper real-time learning rate determination is implemented.

While all drilling operations are unlikely to get automated in the near future, in our opinion the first step on the path is to provide the driller with powerful tools to increase safety, operational efficiency and reduce costs. It is therefore our recommendation to build a control system that promotes human-machine integration, where the machine for instance can suggest optimal strategies going forward, and the driller can select one of these or simply override the suggestion. Furthermore, ensuring high quality of data in all steps of the decision making process will increase reliability and ensure consistently high performance.

10.2.3 Mechanical work

In terms of mechanical aspects with the rig, a recommendation is to rebuild the laboratory rig. As the rig sits today, the experiments performed are not scalable to the full-scale drilling rig, making it difficult to assimilate physics and drilling phenomena on the two rig types. An example of this is that the drill string is kept in compression during drilling, to allow enough WOB to get applied. This is a result of the drill pipe neither being long nor heavy enough to provide the required slenderness and hook load, thus placing the neutral point right beneath the top-drive which in the long run causes high fatigue in the top of the string. This challenge can for instance get solved by developing a rig that drills horizontally with a 10-15m long drill pipe.

Other improvements should include the downhole sensor sub. In the current set-up the sub is placed at the rear-end of the BHA. This places the sub at a great distance from the drill bit when comparing it to the total length of the pipe, which is only about 1.4 m. In order to get near-bit measurements, the sub should for optimal results be placed immediately after the bit, which can possibly be solved by either storing data for post-analysis on an SD card, or installing wireless telemetry solutions, capable of wireless transmission of data in real-time.

References

- [1] A. M. Turing. Intelligent Machinery, A Heretical Theory*. *Philosophia Mathematica*, 4(3):256–260, 09 1996. ISSN 1744-6406. doi: 10.1093/philmat/4.3.256. URL <https://doi.org/10.1093/philmat/4.3.256>.
- [2] Bello et al. Application Of Artificial Intelligence Methods In Drilling System Design And Operations: A Review Of The State Of The Art. *Journal of Artificial Intelligence and Soft Computing Research*, 5(2), pages 121–139, 2015. URL <https://doi.org/10.1515/jaiscr-2015-0024>.
- [3] M. I. Jordan and T. M. Mitchell. Machine Learning: Trends, Perspectives, and Prospects. *Science Volume 349, Issue 6245*, pages 255–260, July 2015. doi: 10.1126/science.aaa8415.
- [4] Equinor. Disclosing all Volve data, viewed 15.02.2019. URL <https://www.equinor.com/en/news/14jun2018-disclosing-volve-data.html>.
- [5] E. A. Løken and A. Trulsen. Construction, Design and Optimization of an Autonomous Laboratory-Scale Drilling Rig. Bachelor thesis, University of Stavanger, 2017.
- [6] O. A. Hjelm and S. J. Nilsen. Further Development and Testing of an Autonomous Drilling Rig and Control Algorithms for Improved Drilling Performance. Master’s thesis, University of Stavanger, 2018.
- [7] DSATS. About Drillbotics, viewed 15.02.2019. URL <https://drillbotics.com/about-drillbotics/>.
- [8] S. C. H. Geekiyanage and E. A. Løken. Autonomous Laboratory-Scale Drilling Rig for Testing and Control of Drilling Systems. *Oil Gas European Magazine*, March 2018.
- [9] S. C. H. Geekiyanage and E. A. Løken. Design of a Finite State Machine; Case study on Algorithm Design for a Smart Drilling Rig. *Oil Gas European Magazine*, March 2019.
- [10] Loeken et al. Design Principles Behind the Construction of an Autonomous Laboratory-Scale Drilling Rig. *IFAC-PapersOnLine Volume 51, Issue 8*, pages 62–69, 2018.
- [11] E. L. Sand. Design and implementation of a control system for a fully automated drilling rig. Bachelor thesis, University of Stavanger, 2018.
- [12] C. Guggedal and M. Steinstø. Control System Architecture and API Integration. Bachelor thesis, University of Stavanger, 2019.
- [13] O. A. Akisanmi. Automatic Management of Rate of Penetration in Heterogeneous Formation Rocks. Master’s thesis, University of Stavanger, 2016.
- [14] Jakobsen A. Hagen, H. and M. Khadisov. Laboratory Drilling Rig Construction, Testing and Modeling for Optimization and Problem Management. Bachelor thesis, University of Stavanger, 2018.

- [15] Deprag. Air Motors - Customized drive solutions, viewed 15.04.2019. URL https://www.deprag.com/fileadmin/bilder_content/emedi/broschueren_pics/emedi_druckluftmotoren/D6000/D6000en.pdf.
- [16] Air Vane Motors for Special Applications. Air Motors - Customized drive solutions, viewed 15.04.2019. URL <http://www.depragusa.com/files/catalogs/D6800en.pdf>.
- [17] S. Skjørestad. Directional Drilling Capabilities of a Rebuilt and Optimized Autonomous Laboratory Scale Drilling Rig. Bachelor thesis, University of Stavanger, 2019.
- [18] Adafruit. FLORA 9-DOF Accelerometer/Gyroscope/Magnetometer - LSM9DS0 - v1.0, viewed 27.04.2019. URL <https://www.adafruit.com/product/2020>.
- [19] Adafruit. Adafruit Trinket M0, viewed 27.04.2019. URL <https://learn.adafruit.com/adafruit-trinket-m0-circuitpython-arduino/overview>.
- [20] E. Wiktorski. Comparative Study of Surface and Downhole Dynamics on a Laboratory-Scale Drilling Rig. SPE ID: SJ-0419-0021. *Society of Petroleum Engineers*, 2019.
- [21] D. Sui. *Drilling Automation and Modeling (PET575)*. University of Stavanger, 2019.
- [22] Dewesoft. PID Control, viewed 08.05.2019. URL <https://dewesoft.pro/online/course/pid-control>.
- [23] Ozzmaker github user: mwilliams03. python-BerryIMU-gryo-accel-compass/berryIMU-simple.py, viewed 30.03.2019. URL <https://github.com/ozzmaker/BerryIMU/blob/master/python-BerryIMU-gryo-accel-compass/berryIMU-simple.py>.
- [24] Tibshirani R. Hastie, T. and J. Friedman. *The Elements of Statistical Learning: Data Mining, Inference, and Prediction, Second Edition (Springer Series in Statistics)*. Springer, 2nd edition, 2016. ISBN 0387848576.
- [25] P. I. Good and J. W. Hardin. *Common errors in statistics (and how to avoid them)*. Wiley, 2006.
- [26] Mustafa Al Bakri et al. *Filling missing data using interpolation methods: Study on the effect of fitting distribution.*, pages 889–895. Trans Tech Publications, 2014. doi: 10.4028. URL <https://www.scientific.net/KEM.594-595.889>.
- [27] K. Holdaway. *Harness Oil and Gas Big Data with Analytics: Optimize Exploration and Production with Data Driven Models*. Wiley Publishing, 1st edition, 2014. ISBN 1118779312, 9781118779316.
- [28] S. James. *An Introduction to Data Analysis using Aggregation Functions in R*. Springer, 2016. URL <https://doi.org/10.1007/978-3-319-46762-7>.
- [29] Scikit Learn. Preprocessing Data, viewed 20.04.2019. URL <https://scikit-learn.org/stable/modules/preprocessing.html>.
- [30] G. James, D. Witten, T. Hastie, and R. Tibshirani. *An Introduction to Statistical Learning: With Applications in R*. Springer, 2014.
- [31] W. van der Aalst. *Process Mining: Data Science in Action - Second Edition*. Springer, 2016.
- [32] A. Hervé and L. J. Williams. Principal component analysis. *Wiley Interdisciplinary Reviews: Computational Statistics*, 2(4):433–459, 2010. doi: 10.1002/wics.101. URL <https://onlinelibrary.wiley.com/doi/abs/10.1002/wics.101>.

- [33] DeepAI. Unsupervised Learning, viewed 25.03.2019. URL <https://deepai.org/machine-learning-glossary-and-terms/unsupervised-learning>.
- [34] Scikit Learn. Choosing the right estimator, viewed 20.02.2019. URL https://scikit-learn.org/stable/tutorial/machine_learning_map/index.html.
- [35] H. Li. Which machine learning algorithm should I use?, viewed 30.03.2019. URL <https://blogs.sas.com/content/subconsciousmusings/2017/04/12/machine-learning-algorithm-use/>.
- [36] Scikit Learn. Tuning the Hyper-Parameters of an Estimator, viewed 15.03.2019. URL https://scikit-learn.org/stable/modules/grid_search.html#grid-search.
- [37] Scikit Learn. Precision Recall F-score and Support function, viewed 20.03.2019. URL https://scikit-learn.org/stable/modules/generated/sklearn.metrics.precision_recall_fscore_support.html#sklearn.metrics.precision_recall_fscore_support.
- [38] E. Beqari. A Very Basic Introduction to Feed-Forward Neural Networks, viewed 10.03.2019. URL <https://dzone.com/articles/the-very-basic-introduction-to-feed-forward-neural>.
- [39] Svozil et al. *Introduction to Multi-Layer Feed-Forward Neural Networks*, pages 44–58. Elsevier, 1997. doi: 10.1016/S0169-7439(97)00061-0. URL [https://doi.org/10.1016/S0169-7439\(97\)00061-0](https://doi.org/10.1016/S0169-7439(97)00061-0).
- [40] N. Christiani and J. Shawe-Taylor. *Support vector machines: Data Analysis, Machine Learning and Applications*. Cambridge University Press, 2000. ISBN ISBN : 0521780195.
- [41] T. J. Sejnowski. *The Deep Learning Revolution*. The MIT Press, 2018. ISBN 978-0-262-03803-4.
- [42] S. Bhattacharyya. Support Vector Machine: Kernel Trick; Mercer’s Theorem, viewed 02.05.2019. URL <https://towardsdatascience.com/understanding-support-vector-machine-part-2-kernel-trick-mercers-theorem-e1e6848c6>.
- [43] T. Cover and P. Hart. *Nearest Neighbor Pattern Classification*. IEEE Transactions on Information Theory 13.1 (1967): 21-27, 1967.
- [44] Papajorgji P. J. Mucherino, A. and P. M. Pardalos. *k-Nearest Neighbor Classification*, pages 83–84. Springer, New York, NY, 2009. ISBN 978-0-387-88615-2. doi: 10.1007/978-0-387-88615-2_4. URL https://doi.org/10.1007/978-0-387-88615-2_4.
- [45] Y. et al. Xu. Coarse to fine k nearest neighbor classifier. *Pattern Recogn. Lett.*, 34 (9):980–986, July 2013. ISSN 0167-8655. doi: 10.1016/j.patrec.2013.01.028. URL <http://dx.doi.org/10.1016/j.patrec.2013.01.028>.
- [46] H. Zhang. The optimality of naive bayes. volume 2, 01 2004.
- [47] Randal S. Olson. TPOT, viewed 12.03.2019. URL <https://epistasislab.github.io/tpot/>.
- [48] S. M. Kamruzzaman, F. Haider, and Hasan A. R. Text classification using association rule with a hybrid concept of naive bayes classifier and genetic algorithm. *CoRR*, abs/1009.4976, 2010. URL <http://arxiv.org/abs/1009.4976>.
- [49] Fan et al. Mr-dbscan: An efficient parallel density-based clustering algorithm using mapreduce. In *2011 IEEE 17th International Conference on Parallel and Distributed Systems*, pages 473–480, Dec 2011. doi: 10.1109/ICPADS.2011.83.

- [50] M. E. Celebi. *Partitional Clustering Algorithms*. Springer, Cham, 2015. ISBN 3-319-09259-6.
- [51] L. K. Larsen. Tools and Techniques to Minimize Shock and Vibration to the Bottom Hole Assembly. Master's thesis, University of Stavanger, 2014.
- [52] K. K. Fjelde. Directional Drilling and Well Flow Engineering (PET505), 2017.
- [53] M. A. Belayneh. *Advanced Well and Drilling Engineering (PET580)*. University of Stavanger, 2019.
- [54] T. R. Hamrick. Optimization of Operating Parameters for Minimum Mechanical Specific Energy in Drilling. Master's thesis, West Virginia University, 2011.
- [55] M. J. Fear and R. C. Pessier. Quantifying Common Drilling Problems With Mechanical Specific Energy and a Bit-Specific Coefficient of Sliding Friction. *SPE-24584*, 1992.
- [56] Celada et al. The Use of the Specific Drilling Energy for Rock Mass Characterisation and TBM Driving During Tunnel Construction. 2009.
- [57] E. Kenneth and S. C. Russel. Innovative Ability to Change Drilling Responses of a PDC Bit at the Rigsite Using Interchangeable Depth-of-Cut Control Features. SPE-178808-MS. *Society of Petroleum Engineers*, 2016.
- [58] G. N. Karadzhova. Drilling Efficiency and Stability Comparison Between Tricone, PDC and Kymera Drill Bits. Master's thesis, University of Stavanger, 2014.
- [59] Scikit Learn. scikit-learn Machine Learning in Python, viewed 05.02.2019. URL <https://scikit-learn.org/stable/>.
- [60] Johannes Otterbach. Principal Component Analysis (PCA) for Feature Selection and some of its Pitfalls, viewed 20.04.2019. URL https://jotterbach.github.io/2016/03/24/Principal_Component_Analysis/.
- [61] B. Horton. Calculating AUC: The Area Under a ROC Curve, viewed 29.05.2019. URL <https://blog.revolutionanalytics.com/2016/11/calculating-auc.html>.
- [62] S. S. Petrova and A. D. Solov'ev. The origin of the method of steepest descent. *Historia Mathematica*, 24(4):361 – 375, 1997. ISSN 0315-0860. doi: <https://doi.org/10.1006/hmat.1996.2146>. URL <http://www.sciencedirect.com/science/article/pii/S0315086096921461>.
- [63] M. Hutson. Ai researchers allege that machine learning is alchemy. *Science Mag*, 2018. URL <https://www.sciencemag.org/news/2018/05/ai-researchers-allege-machine-learning-alchemy>.
- [64] D. G. Luenberger and Y. Ye. *Linear and Nonlinear Programming*. Springer, 3rd edition, 2008. ISBN 978-0-387-74502-2.
- [65] S. C. H Geekiyanage, E. A. Loeken, and D. Sui. Draft: Architectures and Algorithms of an Autonomous Small-Scale Drilling Agent. *Submitted for review, Elsevier*, 2019.
- [66] Dunlop et al. Increased Rate of Penetration Through Automation. *SPE/IADC Drilling Conference and Exhibition*, January 2011. doi: <https://doi.org/10.2118/139897-MS>.
- [67] Sui D. Akisanmi O. Cayeux, E. and O. Alani. Challenges in the Automation of a Laboratory-Scale Drilling Rig and Comparison with the Requirements for Full Scale Drilling Automation. April 2017. doi: 10.2118/185898-MS.

Appendix A

Sensor Sub Stress Simulation

Provided in the software, the material properties considered for all load cases are as illustrated in the table below:

Property	Value	Unit
Density	7.99×10^{-6}	kg / mm ³
Young's modulus	193000	MPa
Poisson's ratio	0.25	-
Yield strength	170	MPa
Ultimate tensile strength	485	MPa
Thermal conductivity	0.0163	W / (mm C)
Specific heat	500	J / (kg C)

Static Stress - Momentum Load Case

In this simulation, a momentum load case is run for 10 Nm torque applied to the threaded sleeve. The top drive mounted on the system can provide a nominal torque of 2.86 Nm and a maximum instantaneous torque of 8.59 Nm, and the pneumatic motors can provide nominal torque 2.1 and 4.9 Nm torque respectively, suggesting the sensor sub should never be subjected to more than 10 Nm of torque.

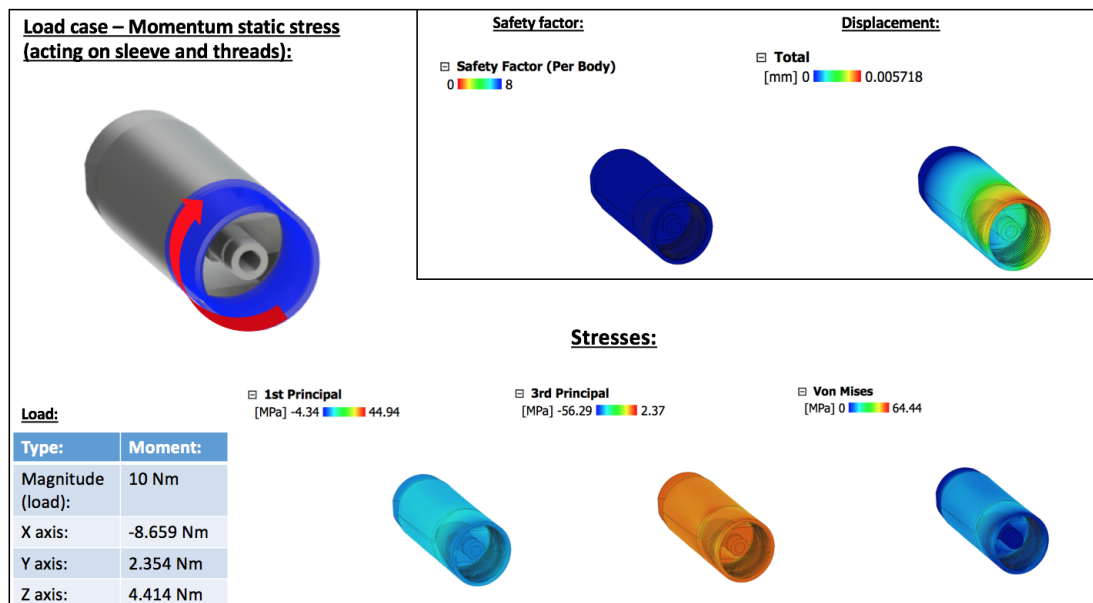


Figure A.1: Safety factor, displacement and stresses acting on the sensor house when 10 Nm force is applied to the threaded sleeve.

Name	Minimum	Maximum
Safety Factor		
Safety Factor (Per Body)	2.638	15
Stress		
Von Mises	1.146E-04 MPa	64.44 MPa
1st Principal	-4.339 MPa	44.94 MPa
3rd Principal	-56.29 MPa	2.368 MPa
Normal XX	-43.44 MPa	24.35 MPa
Normal YY	-14.23 MPa	21.01 MPa
Normal ZZ	-49.73 MPa	14.59 MPa
Shear XY	-12.45 MPa	19.71 MPa
Shear YZ	-14.93 MPa	11.61 MPa
Shear ZX	-23.51 MPa	23.16 MPa
Displacement		
Total	0 mm	0.005718 mm
X	-0.001138 mm	0.001293 mm
Y	-9.111E-04 mm	0.005477 mm
Z	-0.004275 mm	0.001476 mm
Reaction Force		
Total	0 N	37.12 N
X	-23.42 N	29.35 N
Y	-13.36 N	18.96 N
Z	-24.6 N	21.93 N
Strain		
Equivalent	9.818E-10	3.996E-04
1st Principal	-2.877E-09	4.099E-04
3rd Principal	-4.168E-04	-7.487E-11
Normal XX	-1.919E-04	1.026E-04
Normal YY	-5.364E-05	1.199E-04
Normal ZZ	-2.757E-04	6.212E-05
Shear XY	-1.613E-04	2.553E-04
Shear YZ	-1.934E-04	1.503E-04
Shear ZX	-3.045E-04	3E-04

Figure A.2: Stress, strain, reaction force, displacement and safety factor when sensor house is subjected to 10 Nm force.

Static Stress - WOB Force

In this simulation, a force load case is run for 500 N force applied to the surface of the threaded sleeve. The WOB used to drill different formations on the autonomous drilling rig is limited to 30 kg (= 294.2 N), but due to the axial vibrations that occur an additional safety margin of approximately 200 N is added.

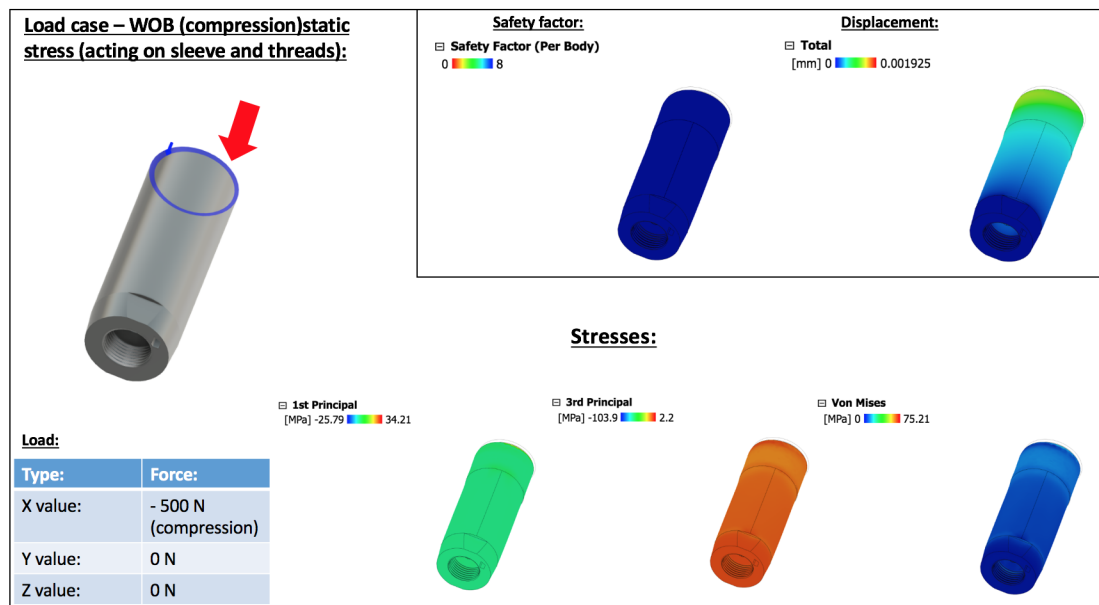


Figure A.3: Safety factor, displacement and stresses acting on the sensor house when 500 N force is applied to the surface of the threaded sleeve.

Name	Minimum	Maximum
Safety Factor		
Safety Factor (Per Body)	2.26	15
Stress		
Von Mises	1.437E-04 MPa	75.21 MPa
1st Principal	-25.79 MPa	34.21 MPa
3rd Principal	-103.9 MPa	2.23 MPa
Normal XX	-60.88 MPa	9.903 MPa
Normal YY	-49.79 MPa	29.27 MPa
Normal ZZ	-51.04 MPa	12.75 MPa
Shear XY	-9.836 MPa	28.75 MPa
Shear YZ	-19.22 MPa	10.23 MPa
Shear ZX	-8.464 MPa	25.55 MPa
Displacement		
Total	0 mm	0.001925 mm
X	-0.001921 mm	1.793E-06 mm
Y	-6.464E-05 mm	5.645E-04 mm
Z	-3.955E-04 mm	2.413E-05 mm
Reaction Force		
Total	0 N	15.03 N
X	-0.2016 N	14.98 N
Y	-2.208 N	2.177 N
Z	-1.359 N	2.953 N
Strain		
Equivalent	1.195E-09	6.743E-04
1st Principal	-5.192E-06	2.519E-04
3rd Principal	-7.812E-04	7.116E-08
Normal XX	-1.924E-04	5.81E-05
Normal YY	-1.219E-04	1.445E-04
Normal ZZ	-1.211E-04	7.001E-05
Shear XY	-1.274E-04	3.724E-04
Shear YZ	-2.489E-04	1.326E-04
Shear ZX	-1.096E-04	3.31E-04

Figure A.4: Stress, strain, reaction force, displacement and safety factor when sensor house is subjected to 500 N force.

Static Stress - Overpull Force

In this simulation, a force load case is run for 500 N force of pull applied to the surface of the threaded sleeve. The actuators that operate the hoisting system on the autonomous rig is similarly to the WOB case limited to maximum 30 kg force before a safety algorithm limits further movement. If however safety algorithms malfunction (or get lifted), estimated pull force of stepper motors controlling actuators is at least 500 N, making an accidental overpull to such force probable.

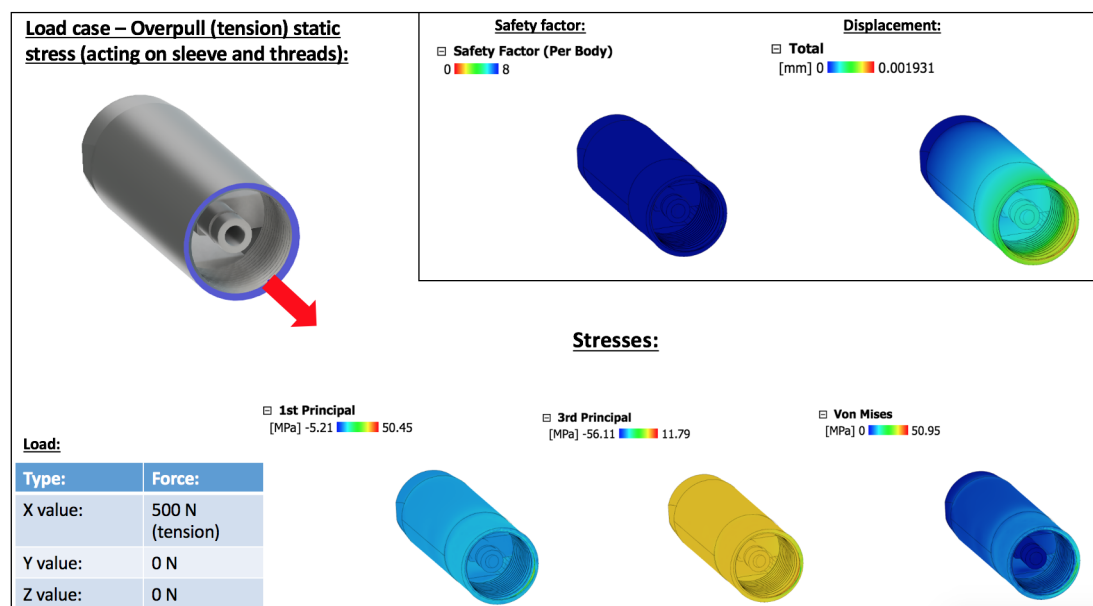


Figure A.5: Safety factor, displacement and stresses acting on the sensor house when 500 N force is applied to surface of the threaded sleeve.

Name	Minimum	Maximum
Safety Factor		
Safety Factor (Per Body)	3.337	15
Stress		
Von Mises	4.384E-12 MPa	50.95 MPa
1st Principal	-5.209 MPa	50.45 MPa
3rd Principal	-56.11 MPa	11.79 MPa
Normal XX	-15.57 MPa	42.14 MPa
Normal YY	-29.27 MPa	30.08 MPa
Normal ZZ	-41.8 MPa	17.05 MPa
Shear XY	-19.79 MPa	16.13 MPa
Shear YZ	-10.23 MPa	12.73 MPa
Shear ZX	-22.47 MPa	8.463 MPa
Displacement		
Total	0 mm	0.001931 mm
X	-1.258E-07 mm	0.001928 mm
Y	-5.745E-04 mm	5.054E-05 mm
Z	-2.77E-05 mm	3.873E-04 mm
Reaction Force		
Total	0 N	16.92 N
X	-16.9 N	0.2563 N
Y	-1.878 N	2.357 N
Z	-2.238 N	1.555 N
Strain		
Equivalent	0	3.96E-04
1st Principal	0	3.668E-04
3rd Principal	-4.236E-04	5.196E-06
Normal XX	-5.811E-05	1.92E-04
Normal YY	-1.445E-04	1.219E-04
Normal ZZ	-1.806E-04	5.111E-05
Shear XY	-2.563E-04	2.09E-04
Shear YZ	-1.325E-04	1.649E-04
Shear ZX	-2.91E-04	1.096E-04

Figure A.6: Stress, strain, reaction force, displacement and safety factor when sensor house is subjected to 500 N force.

Static Stress - Burst Pressure

In this simulation, a pressure load case is run for 10 bar pressure applied to the inside of the small flow channel that the plug screws around, and where an O-ring is used to isolate the sensor chamber inside the sensor sub away from the flow channel. The maximum pressure that the pneumatic motor can sustain is 6 bar, but since the compressor can provide 10 bar pressure at 345 LPM, a 4 bar margin is added to the maximum pressure that the pneumatic motor will get subjected to.

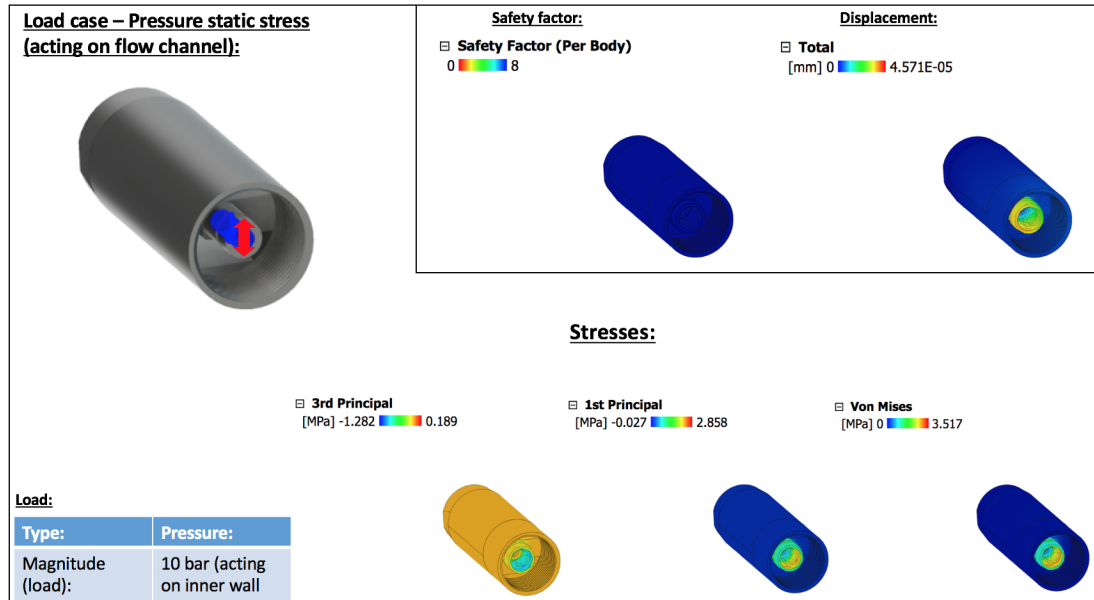


Figure A.7: Safety factor, displacement and stresses acting on the sensor house when 10 bar force is applied to flow channel that extends out of the sensor house and into the plug.

Name	Minimum	Maximum
Safety Factor		
Safety Factor (Per Body)	15	15
Stress		
Von Mises	1.216E-15 MPa	3.517 MPa
1st Principal	-0.02729 MPa	2.858 MPa
3rd Principal	-1.282 MPa	0.1888 MPa
Normal XX	-0.571 MPa	0.4727 MPa
Normal YY	-1.234 MPa	2.765 MPa
Normal ZZ	-1.152 MPa	2.334 MPa
Shear XY	-0.1844 MPa	0.223 MPa
Shear YZ	-1.861 MPa	1.822 MPa
Shear ZX	-0.1964 MPa	0.18 MPa
Displacement		
Total	0 mm	4.571E-05 mm
X	-2.365E-05 mm	6.441E-06 mm
Y	-4.332E-05 mm	2.55E-05 mm
Z	-3.509E-05 mm	3.18E-05 mm
Reaction Force		
Total	0 N	0.01383 N
X	-0.003791 N	0.01369 N
Y	-0.001879 N	0.001303 N
Z	-4.144E-04 N	0.00244 N
Strain		
Equivalent	0	2.718E-05
1st Principal	-1.349E-10	2.649E-05
3rd Principal	-2.061E-05	2.175E-11
Normal XX	-4.253E-06	1.717E-06
Normal YY	-8.909E-06	1.47E-05
Normal ZZ	-9.096E-06	1.317E-05
Shear XY	-2.283E-06	2.761E-06
Shear YZ	-2.305E-05	2.256E-05
Shear ZX	-2.431E-06	2.228E-06

Figure A.8: Stress, strain, reaction force, displacement and safety factor when sensor house is subjected to 10 bar pressure.

Structural Buckling - WOB Force

In this simulation, a buckling load case is run for 500 N force applied to the threaded sleeve, for the same reasons as stated above for the *static stress - WOB force*.

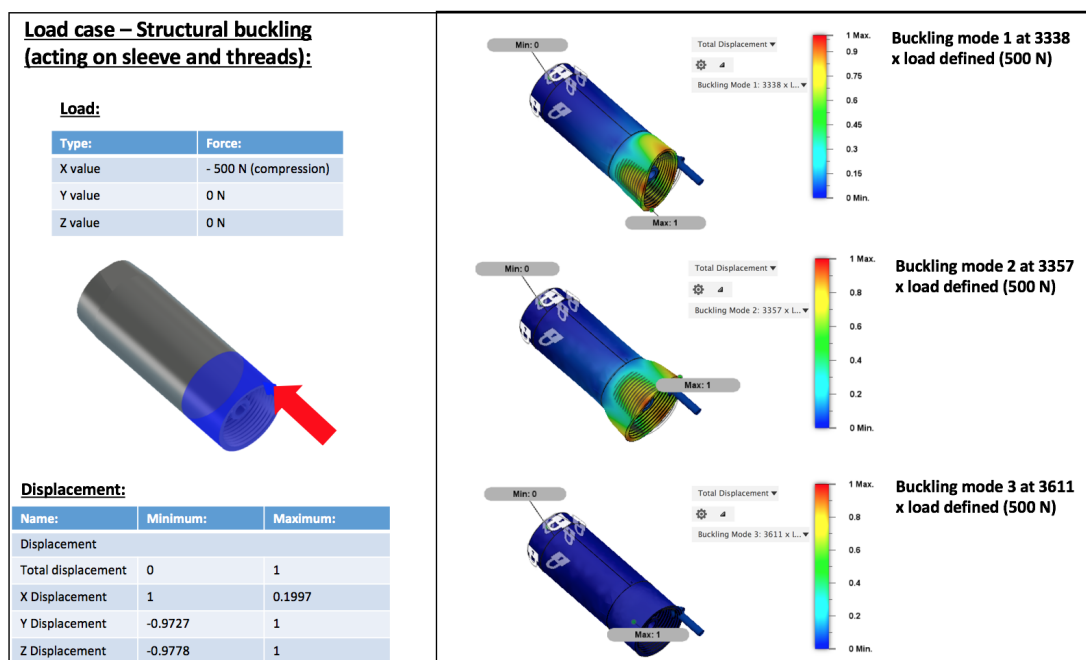


Figure A.9: Displacement and buckling modes acting on the sensor house when 500 N force is applied to the threaded sleeve in compression. Buckling modes (right) illustrate three different buckling cases at 3338, 3357 and 3611 times the 500 N load case simulated.

Appendix B

Manual Downhole Position Tracking Concept

Measurements:										Sections						Well depth		
Equation:	$\Delta t - t-1$	deg	deg	deg	deg	length	length* $\sin(\text{incl})$	cum. TVDs	length*cos(incl)	X disp * $\sin(\text{az})$	cum. TVD	SUM(x disp)	SUM(y disp)	SUM(z disp if > 0)				
Time/Sec/deg	inclin	inclin	azimuth	Error/azim	Error/azim	ft	ft	ft	X displacement	Y displacement	Z displacement	X displacement	Y displacement	Z displacement				
1	1.600	88	2	-2.5	0.5	1.790	1.59925323	1.78876493	0.055839195	0.00194876	1.59925323	0.065783026	0.0026918	1.788764935				
2	2.000	92	1	2	1	4.690	2.896233398	4.68699833	-0.10120854	-0.00176633	2.896233398	-0.035425514	0.000702847	4.686998333				
3	4.000	87	-3	-1	1.5	8.690	3.994518139	8.68151647	0.20934825	-0.010956209	3.994518139	0.173918311	-0.010253362	8.681516472				
4	3.600	87	-1	-0.5	1.5	12.290	3.590666225	12.276828	0.18849942	-0.00228198	3.590666225	0.242227753	-0.15154166	12.276828				
5	3.100	92	-1	0	1.5	15.390	3.098111564	15.3746944	-0.10818844	-0.001888149	3.098111564	0.254139314	-0.011653142	15.37469436				
6	4.800	86	-2	-1.5	0.5	20.190	4.788307441	20.1630018	0.334831074	-0.011685436	4.788307441	0.588970388	-0.02338848	20.1630018				
7	2.900	91	-1	0	23.090	2.89558316	23.0626001	-0.05061979	0.00088301	2.89558316	0.538358409	-0.22455547	23.06260012					
8	4.100	88	0	1	1.5	27.190	4.097502391	27.1600625	0.14308706	0	4.097502391	0.611466345	-0.22455547	27.16006251				
9	4.100	94	-1	2	0.5	31.090	3.890499796	31.0505623	-0.272050248	0.004747931	3.890499796	0.409360698	-0.011707615	31.05056231				
10	3.900	94	-1	2	0.5	35.090	3.984778792	35.0353411	0.348622971	0.00608431	3.984778792	0.788019069	-0.011623006	35.0353411				
11	4.000	85	1	-1.5	1.5	39.190	4.084398262	39.1197394	0.357338445	0.006236418	4.084398262	1.115357614	-0.005868888	39.11973936				
12	4.100	90	1	-2.5	0.5	42.890	4.599299398	42.8097394	2.26652516	3.95636518	1.115357614	1.115357614	-0.005868888	42.80973936				
13	3.700	87	2	-0.5	1.5	43.090	0.189739612	43.090479	0.00948352	0.000478355	0.189739612	1.123014446	-0.005868888	43.09047987				
14	6.300	92	-2	0	0	45.680	2.59841615	45.678951	-0.090738691	0.003166735	2.59841615	1.034562754	-0.001873119	45.6789512				
15	2.600	92	-2	0	0	48.580	2.88964624	48.4968597	-0.252751654	0	2.88964624	0.7818111	-0.001873119	48.49685975				
16	4.900	92	0	0.5	1.5	52.580	3.997563308	52.4944231	-0.139397987	0	3.997563308	0.64211314	-0.001873119	52.49442305				
17	4.600	91	-1	-1.5	0.5	57.180	4.599299398	57.0937225	-0.08028107	0.001401098	4.599299398	0.541932044	-0.00472021	57.09372245				
18	4.100	95	-2	0.5	0	62.280	5.0895296	62.1743154	-0.44449288	0.01512627	5.0895296	0.117437556	0.015046066	62.17431541				
19	5.200	94	-2	-1	1	66.480	4.189769011	66.3640844	-0.29297719	0.010224756	4.189769011	-0.17539434	0.025265363	66.36408442				
20	2.100	93	3	1	0	72.480	5.991777209	72.3586616	-0.314015737	-0.016434314	5.991777209	-0.48955171	0.00881049	72.35866163				
21	6.100	91	1	2	0.5	72.670	0.189971062	72.548327	-0.003105957	-5.78714E-05	0.189971062	-0.492811228	0.00873777	72.5483269				
22	3.600	92	-1	1	0	74.270	1.593911517	74.197442	-0.139449188	-0.007298207	1.593911517	-0.632520317	0.001474971	74.1974421				
23	1.600	95	-1	2.5	1.5	77.170	2.892935746	77.03268	0.202293774	-0.003350513	2.892935746	-0.00205542	77.03267996					
24	2.900	89	-3	-1.5	1.5	81.170	3.999390781	81.0320707	0.06989626	-0.003635354	3.999390781	-0.360216917	-0.005709096	81.03207074				
25	3.600	92	-2	-1	1.5	84.770	3.597866977	84.6296777	-0.12458188	0.00438471	3.597866977	-0.485551616	-0.01124386	84.62967771				
26	3.100	93	0	0.5	0	87.870	3.095751558	87.726293	-0.162241464	0	3.095751558	-0.48098657	-0.01124386	87.72629297				
27	3.100	87	3	0	1	92.670	4.793421767	92.519051	0.25121259	0.013147451	4.793421767	-0.39688598	0.01823065	92.51905104				
28	4.800	94	-2	1.5	1.5	95.570	2.892935746	95.419868	-0.202293774	0.007059951	2.892935746	-0.599177554	0.018883016	95.41986816				
29	4.100	85	-1	-2	0	99.670	4.084398262	99.496385	0.357338445	-0.006236418	4.084398262	-0.241892028	0.012646598	99.49638505				
30	3.900	87	2	0	0.5	103.570	3.89465186	103.39104	-0.204110229	-0.010622304	3.89465186	-0.45949438	0.01964294	103.3910402				
31	4.000	87	-3	0	1.5	107.570	3.994518139	107.385558	0.20934825	-0.010956209	3.994518139	-0.236665613	-0.008991915	107.3855584				
32	4.100	88	2	-2.5	0.5	111.670	4.097502391	111.483061	0.14308706	0.004993697	4.097502391	-0.093517676	0.003982818	111.4830608				
33	3.700	92	-2	-0.5	1	115.370	3.69774606	115.180807	-0.12912818	0.006758045	3.69774606	-0.22265814	0.002759826	115.1808088				
34	6.100	91	-2	1.5	0.5	115.560	0.189971062	115.370778	-0.003105957	0.000115725	0.189971062	-0.22901771	0.002875552	115.3707779				
35	2.600	85	2	-0.5	1	118.160	2.590166215	117.960884	0.226604931	0.007908398	2.590166215	0.00064316	0.01078395	117.9608841				
36	2.100	90	1	-1.5	0	121.060	2.9	120.860884	1.78475E-16	3.10036E-18	2.9	0.00064316	0.01078395	120.8608841				
37	2.900	88	1	-2	1	125.060	3.997563308	124.858447	0.139397987	0.002436321	3.997563308	0.140241347	0.01322027	124.8584474				
38	4.600	85	-1	0	1.5	129.660	4.589395611	129.449493	0.409761719	-0.00096966	4.589395611	0.541157563	0.006223314	129.4494933				
39	4.100	92	-2	1.5	0.5	134.760	3.096893218	134.57836	-0.177987433	0.006211762	3.096893218	0.36317013	0.012434986	134.5783632				
40	5.200	95	-1	-2	1.5	138.960	4.184017732	138.721854	-0.36605412	0.006388225	4.184017732	-0.00283989	0.018823511	138.721854				
41	4.100	90	3	1	0.5	144.960	6	144.721854	3.67545E-16	1.92358E-17	6	-0.00283989	0.018823511	144.721854				
42	6.000	88	-2	-1.5	1	145.160	0.189884257	144.917378	0.006369094	-0.000347035	0.189884257	0.010476477	0.018471282	144.9173782				
43	1.100	87	-2	-2	1.5	146.760	1.597807246	146.590445	0.08373753	-0.002922908	1.597807246	0.087444445	0.015554079	146.5904455				
44	1.600	87	-2	0	0.5	149.650	2.896025651	149.495571	0.151774273	0	2.896025651	0.239258718	0.015554079	149.4955711				
45	2.900	86	-1	2.5	0.5	153.650	3.990256201	153.398827	0.279028995	0.0048869673	3.990256201	0.518284613	0.010684406	153.3988273				
46	4.000	87	0	0	1	157.250	3.69989827	156.999827	2.20527E-16	0	3.69989827	0.010684406	0.010684406	156.9998273				
47	3.600	90	0	0.5	1	161.350	3.095751558	160.991579	-0.162241464	-0.002831604	3.095751558	0.356603169	0.007852902	160.9915789				
48	3.100	88	3	1.5	1.5	165.450	4.797597	164.888655	0.167517584	0.008767193	4.797597	0.523560733	0.016620995	164.8886549				
49	4.800	85	-1	-2	0	168.050	2.88964624	167.776719	0.252751654	-0.004411125	2.88964624	0.776312387	0.01220897	167.7767195				
50	2.100	82	1	1.5	1	172.150	4.06099082	171.837719	0.570609714	0.009938513	4.06099082	1.346922101	0.022167483	171.8377186				
51	3.900	85	-2	-1.5	0.5	176.050	3.885159323	175.722878	0.33997997	-0.011862997	3.885159323	1.608680497	0.010304885	175.7228779				
52	3.100	88	-1	0.5	1.5	180.050	3.997563308	179.720441	0.139397987	-0.002436321	3.997563308	1.820427484	0.00786565	179.7204412				
53	4.100	79	-1	0	1	184.150	4.024671452	183.745113	0.782316881	-0.013653312	4.024671452	2.608744365	-0.005784747	183.7451126				
54	5.700	84	3	-1.5	1	187.850	3.679731013	187.424844	0.386755314	0.020241209	3.679731013	2.995499679	0.014546462	187.4248437				
55	6.100	86	-2	-2	0	188.440	0.189371077	188.141881	0.01252373	-0.000662499	0.189371077	3.008532409	0.013999131	188.1418808				
56	1.100	76	2	0	1.5	190.640	2.543183762	190.157565	0.546570396	0.01866535	2.543183762	3.549323896	0.032859548	190.1575646				
57	2.600	81	2	0	0.5	193.540	2.896233398	193.057578	0.10120854	0.005296846	2.896233398	3.605032346	0.038156394	193.057578				
58	4.900	79	3	-1.5	0	197.540	3.926508734	196.982307	0.763235982	0.039944685	3.926508734	4.413768327	0.078101079	196.9823067				
59	4.600	83	3	-1	1	202.140	4.565712298	201.548019	0.56039988	0.029339484	4.565712298	4.974367307	0.107440562	201.548019				
60	3.100	86	2	2.5	0.5	207.240	5.08756656	206.635966	0.355780016	0.02145776	5.08756656	5.330125223	0.13885338	206.6359657				
61	5.100	84	1	-1.5	1.5	211.440	4.176991961	210.812588	0.439019546	0.007661948	4.176991961	5.769144869	0.127518286	210.8125876				
62	4.000	80	-2	0	1.5	217.440	5.908846518	216.721434	-0.036361404	0.008846518	5.908846518	6.811033935	0.091158882	216.7214342				
63	6.100	78	-3	-1.5	1.5	217.630	0.18848044	216.907282	0.039503221	-0.002067439	0.18848044	6.800537156	0.090899443	216.9072822				
64	1.600	82	3	-2.5	0.5	219.230	1.58442891	218.991711	0.22276962	0.01164012	1.58442891	7.073214118	0.1073454	218.9917111				
65	2.900	86	1	1.5	0	222.130	2.892935746	221.846447	0.202293774	0.003350513	2.892935746	7.275507892	0.104273968	221.8464469				

Calculations:									
Error Well position:									
Well Plan									
xdisp*0.9	xdisp*1.1	xdisp*0.8	vdia*1.2	SUM(xmin error)	SUM(xmax error)	SUM(ymin error)	SUM(ymax error)	875 mm TVD	10 cm hor. Build
Min error	Max Error	Min error	Max Error	Min error	Max Error	Min error	Max Error	TVD Plus	K-cm/deg
0.0089445	0.01093821	0.00844236	0.00598483	0.00894449	0.01093821	0.008442357	0.00598483	0.190	0
0.0525238	0.06142314	0.00165645	0.002241074	0.059204724	0.072361329	0.002098803	0.002839557	1.789	0
-0.09108769	-0.11132939	-0.00150138	-0.00203128	-0.031882963	-0.038968065	0.00059742	0.00808274	4.687	0
0.1884044	0.230278207	-0.00931278	-0.01259964	0.15652648	0.191310142	-0.008715358	-0.011791366	8.682	0
0.1695685	0.207250387	-0.00279497	-0.00378143	0.326094978	0.398650529	-0.011510326	-0.011572794	12.277	0
-0.097696	-0.11909728	0.00160493	0.002171371	0.228725362	0.279553245	-0.00909054	-0.013601423	15.375	0
0.30134797	0.368314181	-0.00993262	-0.01343825	0.530073349	0.647867626	-0.01983802	-0.026839675	20.163	0
-0.04555078	-0.05567318	0.00075081	0.001015796	0.484522568	0.59219425	-0.019087215	-0.025823879	23.063	0
0.12877914	0.15739673	0	0	0.613301711	0.74959098	-0.019087215	-0.025823879	27.160	0
-0.24484522	-0.29925527	0.00403574	0.005460121	0.368454688	0.450335708	-0.015051473	-0.020363758	31.051	0
0.31376067	0.383485268	0.00517166	0.006996956	0.682217162	0.83820976	-0.00987981	-0.013366801	35.035	0
0.32160469	0.3930724	0.00530095	0.00717188	1.003821853	1.226893375	-0.004578855	-0.006194921	39.120	0
2.0309516	2.49318216	3.2623218	4.54098218	1.003821853	1.226893375	-0.004578855	-0.006194921	42.820	0
0.0089445	0.010938215	0.00029498	0.00039099	1.012771301	1.23783159	-0.004283875	-0.005795831	43.009	0
-0.08166482	-0.09981256	0.00269172	0.003641745	0.931106479	1.13801903	-0.001592151	-0.002154086	45.608	0
-0.2274649	-0.27802682	0	0	0.70362999	0.85999221	-0.001592151	-0.002154086	48.497	0
-0.12563819	-0.15355779	0	0	0.577991802	0.706434425	-0.001592151	-0.002154086	52.494	0
-0.07225296	-0.08830918	0.00119093	0.001611263	0.50573884	0.618125248	-0.000401218	-0.000542824	57.094	0
-0.4000446	-0.48894372	0.01318573	0.01783921	0.10360398	0.129181532	0.012784515	0.017266697	62.174	0
-0.26367947	-0.32221491	0.00869104	0.01175847	-0.15795847	-0.193903375	0.021473558	0.029053167	66.364	0
-0.28261416	-0.34541731	-0.01396917	-0.01889946	-0.440259654	-0.538510688	0.007506391	0.010155706	72.56	0
-0.00298436	-0.00364755	-0.19191E-05	-6.6552E-05	-0.443584016	-0.542158241	0.007457201	0.010089154	72.546	0
-0.1250447	-0.15339411	-0.00620348	-0.00839294	-0.569088285	-0.695525349	0.001253725	0.01606216	74.140	0
0.1820644	0.222523151	-0.00300094	-0.00406009	-0.387023889	-0.473029197	-0.001747211	-0.002263874	77.033	0
0.0628266	0.076790588	-0.00310552	-0.00420139	-0.324195226	-0.396238609	-0.004852732	-0.00636546	81.032	0
-0.11807457	-0.13826201	0.003727	0.00502416	-0.457269955	-0.52440616	-0.001123728	-0.001526044	84.630	0
-0.14601732	-0.17845601	0	0	-0.458328613	-0.712902227	-0.001123728	-0.001526044	87.226	0
0.22609133	0.27633849	0.01117533	0.015119569	-0.357195582	-0.436572378	0.010046065	0.013596524	92.519	0
-0.00298436	-0.00364755	0.00600096	0.008118944	-0.539259978	-0.659059529	0.010650563	0.021715468	95.412	0
0.32160469	0.3930724	-0.00530095	-0.00717188	-0.217655288	-0.266023129	0.010749608	0.014543588	99.496	0
-0.1836921	-0.22452125	-0.00907996	-0.01228465	-0.401354494	-0.490544381	0.00166965	0.00228938	103.391	0
0.1884044	0.230278207	-0.00931278	-0.01259964	-0.212945051	-0.262066174	-0.007643128	-0.010340703	107.386	0
-0.12877914	-0.15739673	0.00160493	0.002171371	-0.457269955	-0.52440616	-0.001123728	-0.001526044	111.483	0
-0.11623152	-0.14204095	0.00574434	0.00771751	-0.200381233	-0.244910395	-0.002345852	0.00317338	115.181	0
-0.00298436	-0.00364755	9.8366E-05	0.000133084	-0.203365594	-0.248557948	0.002442419	0.003006884	115.371	0
0.2039444	0.24926424	0.00672214	0.00994658	0.000578844	0.00070476	0.009166357	0.012401542	117.961	0
1.598618	1.95411E-16	2.6335E-18	3.56541E-18	0.000578844	0.00070476	0.009166357	0.012401542	120.861	0
0.12563819	0.153557785	0.00207087	0.002801769	0.126217032	0.154262561	0.01123723	0.015203311	124.858	0
0.36082477	0.441008058	-0.00594741	-0.0080465	0.487041807	0.59237332	0.005289817	0.007156811	129.441	0
-0.16018869	-0.19378618	0.00527992	0.00714323	-0.081615009	-0.102489444	-0.003309486	-0.004097951	131.483	0
-0.32944871	-0.40265953	0.00543025	0.007346004	-0.00259559	-0.003172388	0.015999985	0.021647038	138.722	0
3.3079E-16	4.04299E-16	1.635E-17	2.1212E-17	-0.00259559	-0.003172388	0.015999985	0.021647038	144.722	0
0.00896781	0.007293995	-0.00029498	-0.00039099	0.00337223	0.004121606	0.015705005	0.021247948	144.912	0
0.07536478	0.092111283	-0.00248404	-0.00336076	0.078736	0.096232889	0.013220967	0.017887191	146.510	0
0.13659685	0.1609517	0	0	0.215332846	0.26318459	0.013220967	0.017887191	149.406	0
0.25112531	0.300928484	-0.00413922	-0.00560012	0.466456152	0.570113074	0.009081745	0.012287066	153.396	0
1.9847E-16	2.42579E-16	0	0	0.466456152	0.570113074	0.009081745	0.012287066	156.996	0
-0.14601732	-0.17845601	-0.00240678	-0.00325623	0.320438834	0.391647464	0.006674966	0.009030837	160.992	0
0.15076583	0.184269343	0.00745211	0.010082272	0.47120466	0.57916806	0.01412708	0.019113109	164.889	Hor. Build, Counter:
0.2274649	0.278026819	-0.00374946	-0.00507279	0.698681148	0.853943625	0.014040315	0.014040315	168.785	0.763
0.51354874	0.627670685	0.00846474	0.01145229	1.212229891	1.481614311	0.01884236	0.025492605	172.681	1.527
0.3091466	0.37898136	-0.01008321	-0.01364199	1.518146548	1.855512447	0.008759153	0.011850618	176.577	2.290
0.12563819	0.153557785	-0.00207087	-0.00280177	1.643784736	2.009070233	0.00668828	0.009488449	180.473	3.053
0.70408519	0.860548569	-0.01160532	-0.01570131	2.347869929	2.869618802	-0.004917033	-0.00665246	184.369	3.817
0.34807978	0.425430845	0.0120503	0.023277391	2.695049711	3.295046447	0.01287992	0.016624931	188.265	4.580
0.01192386	0.014579103	-0.00039317	-0.00053193	2.707878068	3.30962875	0.011894286	0.016093	192.161	5.344
0.48651336	0.594627436	0.01603579	0.02169548	3.194391425	3.90425616	0.027930616	0.03778848	196.057	6.107
0.09108769	0.11320394	0.00450232	0.006091373	3.285479111	4.015885881	0.032423935	0.043878853	199.953	6.870
0.68091238	0.839595958	0.03395298	0.045936388	3.972391495	4.85514516	0.006389917	0.00981624	203.849	7.634
0.50453008	0.616658878	0.02403856	0.033740406	4.471093056	5.471084038	0.091324478	0.123556647	207.745	8.397
0.32018221	0.391333818	0.01055341	0.014278142	4.797112791	5.863137856	0.101877887	0.137834789	211.641	9.160
0.39511759	0.4820215	0.00651266	0.00881124	5.192230382	6.346059356	0.108390543	0.146646028	215.537	9.924
0.93700016	1.146077973	-0.03090719	-0.04181561	6.129930541	7.492137328	0.077483349	0.104384314	219.433	10.687
0.0355529	0.043453543	-0.00175732	-0.00237755	6.165483441	7.53590872	0.075726026	0.102452859	223.329	11.454
0.20040927	0.244944658	0.00909591	0.013402113	6.365892706	7.78053553	0.08631936	0.115854973	227.225	12.214
0.1820644	0.222523151	0.00300094	0.00406009	6.547957102	8.000508681	0.08862872	0.11913063	231.121	12.977
0.68091238	0.839595958	0.02264107	0.030652054	7.234869486	8.84261426	0.111273941	0.150547097	235.017	13.740
0.28236461	0.345136741	0	0	7.517254092	9.18755002	0.111273941	0.150547097	238.913	14.504
0.29163441	0.35644206	0.00961246	0.013005087	7.808888505	9.544197061	0.120886397	0.163552184	242.809	15.267
0.6012278	0.734833973	-0.01981685	-0.02681103	8.410116301	10.27903105	0.101069547	0.136741151	246.705	16.031
0.49081148	0.608680695	0.02461591	0.033303881	8.908127779	10.88771173	0.125685459	0.170045032	250.601	16.794
0.38571003	0.471423369	0.01271325	0.017200276	9.293837808	11.3591351	0.138398706	0.187245309	254.497	17.557
0.4277614	0.522819483	0.00705072	0.009593206	9.721599204	11.88195458	0.145449424	0.196784515	258.393	18.321
0.56316407	0.688311646	0	0	10.28476328	12.57026223	0.145449424	0.196784515	262.289	19.084
0.70408519	0.860548569	-0.01160532	-0.01570131	10.82262602	13.32110461	0.138398706	0.187245309	266.185	19.847

Figure B.2: Tracking concept for downhole position, part 2.

Appendix C

Autonomous Search Algorithm Code

In this appendix, a short section of the implemented code for optimal ROP search is included.

```
1 ...
2
3 def on_hoisting_hbm_sync(pulse, timestamp, data):
4     global rop_list, rpm_list, wob_list
5     global LEARNING_RATE, GRADIENT_INTERVAL, rpm_setpoint, wob_setpoint, rop_prev_depth
6     global rop_enabled
7     wob = 0
8     z1 = 0
9     z2 = 0
10    z3 = 0
11    rpm = 0
12    opening = 0
13    rop_inst = 0
14
15    current_wob_setpoint = 0.0
16
17    for package in data:
18
19        if package[0] == module_pb2.HOISTING_UNIT:
20            hoisting_data = package[1]
21
22            for value in hoisting_data:
23                if value.source == module_pb2.HEIGHT:
24                    #print(" height_data(0): ", value.data[0])
25
26                    #print(" height_data(last): ", value.data[-1])
27                    #rop_inst = value.data[0] - value.data[-1]
28                    #print(" ROP_inst is: ", rop_inst)
29
30                    Δ_depth = rop_prev_depth - value.data[0]
31                    depth = value.data[0]*(-1)
32                    rop_prev_depth = value.data[0]
33
34                    rop_inst = Δ_depth
35
36                elif value.source == module_pb2.WOB:
37                    #print("Current WOB setpoint: ", (value.data[-1] / 1000))
38                    current_wob_setpoint = value.data[-1] / 1000
39
40            elif package[0] == module_pb2.HBM:
41                downhole_data = package[1]
42
43                for value in downhole_data:
44                    if value.source == module_pb2.Z1:
45                        z1 = value.data[0]
46                    elif value.source == module_pb2.Z2:
47                        z2 = value.data[0]
48                    elif value.source == module_pb2.Z3:
49                        z3 = value.data[0]
50
51            elif package[0] == module_pb2.VALVE_UNIT:
52                valveopening_data = package[1]
53
54                for value in valveopening_data:
55                    if value.source == module_pb2.OPENING:
56                        opening = value.data[0]
57                        #print(" Valve opening is: ", opening)
58
59
60    wob = (z1+z2+z3) / 1000
61    rpm = valve_opening_to_rpm(opening)
62
63    if rpm == 0 or current_wob_setpoint ≥ 0:
64        rop_enabled = False
65        rpm_list.clear()
66        wob_list.clear()
67        rop_list.clear()
68        return
69    else:
70        if not rop_enabled:
71            # A new search has started. Base start on current setpoint
72            wob_setpoint = current_wob_setpoint
73            rpm_setpoint = rpm
```



```

74         rop_enabled = True
75
76     # Calculate ROP gradient
77     rpm_list.append(rpm)
78     wob_list.append(wob)
79     rop_list.append(rop_inst)
80
81
82     if len(rpm_list) > GRADIENT_INTERVAL:
83         # Calculate and pass on model suggestions
84         rpm_grad = ((GRADIENT_INTERVAL * sum(np.multiply(rpm_list, rop_list))) - . . .
85                 ↪ (sum(rpm_list) * sum(rop_list)) / ((GRADIENT_INTERVAL * . . .
86                 ↪ sum(np.multiply(rpm_list, rpm_list)) - (math.pow(sum(rop_list), 2)))
87         wob_grad = ((GRADIENT_INTERVAL * sum(np.multiply(wob_list, rop_list))) - (sum(wob_list) . . .
88                 ↪ * sum(rop_list)) / ((GRADIENT_INTERVAL * sum(np.multiply(wob_list, wob_list))) - . . .
89                 ↪ (math.pow(sum(rop_list), 2)))
90
91         new_rpm_setpoint = rpm_setpoint + (LEARNING_RATE * rpm_grad)
92
93         # Setpoint can not be less than 0 or greater than 1070 RPM
94         new_rpm_setpoint = min(max(new_rpm_setpoint, 0), 1070)
95
96         print(" New RPM_setpoint: ", str(new_rpm_setpoint), " Old RPM was: ", str(rpm_setpoint))
97
98         new_wob_setpoint = wob_setpoint + (LEARNING_RATE * wob_grad)
99
100        # Setpoint has to be between -5 and -18 kg WOB
101        new_wob_setpoint = max(min(new_wob_setpoint, -5), -18)
102
103        if new_wob_setpoint < wob_setpoint and new_wob_setpoint ≥ wob_setpoint - 0.5:
104            new_wob_setpoint = wob_setpoint - 0.5
105
106        if new_wob_setpoint == wob_setpoint:
107            new_wob_setpoint -= 0.25
108
109        print(" New WOB_setpoint: ", str(new_wob_setpoint), " Old WOB was: ", str(wob_setpoint))
110
111        rpm_setpoint = new_rpm_setpoint
112        wob_setpoint = new_wob_setpoint
113
114        rop_list = []
115        wob_list = []
116        rpm_list = []
117
118        values = []
119        values.append(module_pb2.Values(data=[rpm_setpoint], source=module_pb2.RPM_SETPOINT))
120        values.append(module_pb2.Values(data=[wob_setpoint], source=module_pb2.WOB_SETPOINT))
121
122        suggestion = module_pb2.StreamData(pulse=pulse, timestamp=timestamp, . . .
123            ↪ origin=module_pb2.GRADIENT_SEARCH, values=values)
124
125        with suggestion_cv:
126            suggestion_items.append(suggestion)
127            suggestion_cv.notify()
128
129    def valve_opening_to_rpm(opening):
130        rpm = -0.4509 * math.pow(opening, 2) + 88.289 * opening - 3249.4
131        #print("BIT RPM: ", rpm)
132        return rpm
133
134    ...

```

Appendix D

Active Steering Code

In this appendix, a short section of the implemented code for inclination steering is included.

```
1 ...
2
3 HMD_target = [0,0,0,.....,0.14298618] # Horizontal build per MD distance, updated every mm.
4
5 MD_target = [0,1,2,.....,617.3] # MD reference, updated every mm.
6
7 ...
8
9 def on_hoisting_downhole_mode_sync(pulse, timestamp, data):
10     # Sync data received. Calculate values dependent of both hoisting and downhole
11     global mag_x_min, mag_y_min, mag_z_min, mag_x_max, mag_y_max, mag_z_max, ...
12     ↪ downhole initialized
13     global AZIMUTH_ERROR, INCLINATION_ERROR, MD_sum, x_disp_sum, y_disp_sum, ...
14     ↪ z_disp_sum, TVD_sum, TVD_sum_max
15     ↪ max_error_x_sum, max_error_y_sum, max_error_z_sum
16     ↪ x_disp_sum_max, y_disp_sum_max, z_disp_sum_max
17     ↪ min_error_x_sum_max, min_error_y_sum_max, max_error_x_sum_max, ...
18     ↪ max_error_y_sum_max
19     global acc_measurements, depth_counter
20     global topside_plot_counter, topside_graph_data_x, topside_graph_data_y, topside_graph_max_x
21     ↪ topside_graph_max_y, topside_graph_min_x, topside_graph_min_y
22     ↪ other_plot_counter, rear_graph_data_x, rear_graph_data_y, rear_graph_max_x
23     ↪ rear_graph_max_y, rear_graph_min_x, rear_graph_min_y
24     ↪ side_graph_data_x, side_graph_data_y, side_graph_max_x
25     ↪ side_graph_max_y, side_graph_min_x, side_graph_min_y
26     global tag_bottom, prev_depth, rop_prev_depth
27     ↪ x_disp_last, error_x, maximum_depth
28     ↪ last_mm_step, last_build
29     global acc_x, acc_y, acc_z, gyr_x, gyr_y, gyr_z, mag_x, mag_y, mag_z, depth
30     #print("Data sync")
31     acc_x = 0.0
32     acc_y = 0.0
33     acc_z = 0.0
34     mag_x = 0.0
35     mag_y = 0.0
36     mag_z = 0.0
37
38     autonomous_phase = 0
39
40     horizontal_by_TVD = 0.0
41
42     depth = 0
43     Δ_depth = 0.0
44
45     for package in data:
46         print(package[0])
47
48         if package[0] == module_pb2.HOISTING_UNIT:
49             hoisting_data = package[1]
50
51             for value in hoisting_data:
52                 if value.source == module_pb2.HEIGHT:
53                     #Height is positive upwards and negative downwards, we want ...
54                     ↪ depth to be positive
55                     Δ_depth = (value.data[-1] - value.data[0])*(-1)
56                     Δ_depth = rop_prev_depth - value.data[0]
57                     print("Delta Height: ", Δ_depth)
58                     print("Height: ", depth)
59                     rop_prev_depth = depth*(-1)
60
61             elif package[0] == module_pb2.DOWNHOLE:
62                 downhole_data = package[1]
63                 for value in downhole_data:
64                     if value.source == module_pb2.ACC_X:
65                         acc_x = value.data[0]
66                     elif value.source == module_pb2.ACC_Y:
67                         acc_y = value.data[0]
68                     elif value.source == module_pb2.ACC_Z:
69                         acc_z = value.data[0]
70
71                     elif value.source == module_pb2.GYR_X:
72                         gyr_x = value.data[0]
```

```

72         elif value.source == module_pb2.GYR_Y:
73             gyr_y = value.data[0]
74         elif value.source == module_pb2.GYR_Z:
75             gyr_z = value.data[0]
76
77         elif value.source == module_pb2.MAG_X:
78             mag_x = value.data[0]
79
80         elif value.source == module_pb2.MAG_Y:
81             mag_y = value.data[0]
82
83         elif value.source == module_pb2.MAG_Z:
84             mag_z = value.data[0]
85
86
87     #Check if the drillpipe is moving downwards or upwards and check if bottom has been tagged
88     if not tag_bottom:
89         if prev_depth > 0:
90             if prev_depth > depth * 20:
91                 print("Bottom has been tagged, can begin plotting!")
92                 tag_bottom = True
93
94         prev_depth = depth
95
96     # Calculate inclination
97     inclination = 180.0*math.acos(acc_y/math.sqrt(acc_x*acc_x+acc_y*acc_y+acc_z*acc_z))/math.pi
98     ###was acc_z / ...
99
100
101     if not downhole_initialized:
102         INCLINATION_ERROR = inclination - 90
103         downhole_initialized = True
104     else:
105         inclination -= INCLINATION_ERROR
106
107     # MD (Measured depth) denotes the change in actuator position over the 300(?) sample window.
108     MD = Δ_depth
109     MD2 = depth
110
111     #print("MD", str(MD))
112
113     # MD_sum denotes the sum of all previous TVD values, and is used to track the bit elevation in . . .
114     ↪ real-time
115     # NB! SHOULD BE A GLOBAL VARIABLE
116     MD_sum += MD
117     MD_sum_max = max(MD_sum, MD_sum_max)
118     print("MD TOTAL: ", str(MD_sum), " mm")
119
120
121     # TVD (True vertical depth) denotes the vertical change in actuator position over the 300 sample . . .
122     ↪ window (given that
123         # even if the pipe moves down, it is not guaranteed that the movement is . . .
124         ↪ perfectly 90
125         # degree downwards).
126     TVD = MD * math.sin(inclination * (math.pi / 180))
127     #print("TVD", str(TVD))
128
129     TVD_sum += TVD
130
131     TVD_sum_max = max(TVD_sum, TVD_sum_max)
132     print("TVD TOTAL: ", str(TVD_sum_max), " mm")
133
134     # x_disp (horizontal build) denotes the horizontal movement that is built in each well section . . .
135     ↪ depending on inclination
136     x_disp = (MD) * math.cos(inclination * (math.pi / 180))
137
138     print(x_disp)
139
140     # x_disp_sum denotes the sum of all x_disp in the past so that
141     # NB! SHOULD BE A GLOBAL VARIABLE
142     if inclination ≤90:
143
144         x_disp_sum += x_disp
145
146         x_disp_sum_max = max(x_disp_sum, x_disp_sum_max)
147
148     elif inclination > 90:
149
150         x_disp_sum += x_disp
151
152         x_disp_sum_max = min(x_disp_sum, x_disp_sum_max)
153
154     print("HORIZONTAL BUILD: ", str(x_disp_sum_max), " mm")
155
156     if TVD_sum_max ≤0:
157         horizontal_by_TVD = 0
158     else:
159         horizontal_by_TVD = x_disp_sum_max / TVD_sum_max
160
161     print("H/TVD: ", horizontal_by_TVD)
162
163     # Check H/TVD relative to target path per mm
164     current_mm_step = max(0, min(len(HMD_target), math.floor(depth)))
165
166     print("Target H/TVD: ", HMD_target[current_mm_step])
167
168
169     if current_mm_step > last_mm_step:
170         if constant_build:
171             # A sensor is lost, send constant build messages
172             print("Constant build")
173             build_angle = 1.0
174         elif HMD_target[current_mm_step] ≥horizontal_by_TVD:
175             build_angle = 1.0
176             print("Build angle")
177         else:
178             build_angle = 0.0

```

```
179         print("No build")
180         last_mm_step = current_mm_step
181         last_build = build_angle
182         """else:
183             print()
184             build_angle = 0.0"""
185     else:
186         build_angle = last_build
187         print("No new mm")
188
189     values = []
190     values.append((module_pb2.Values(data=[build_angle], source=module_pb2.BUILD_ANGLE)))
191
192     suggestion = module_pb2.StreamData(pulse=pulse, timestamp=timestamp, . . .
193         ↪ origin=module_pb2.ACTIVE_STEERING, values=values)
194
195     for _, receiver in receivers.items():
196         with receiver.cv:
197             receiver.items.append(suggestion)
198             receiver.cv.notify()
199     ...
```

Appendix E

Stuck Pipe Code

In this appendix, a short section of the implemented code for stuck pipe detection is included.

```
1 ...
2
3 def on_hoisting_downhole_sync(pulse, timestamp, data):
4     # Sync data received. Calculate values dependent of both hoisting and downhole
5     global MD_current, MD_old, md_counter, phase
6     #print("Data sync")
7
8     stuck = False
9
10    for package in data:
11        #print(package[0])
12
13        if package[0] == module_pb2.HOISTING_UNIT:
14            hoisting_data = package[1]
15
16            for value in hoisting_data:
17                if value.source == module_pb2.HEIGHT:
18                    #Δ_height = value.data[-1] - value.data[0]
19                    #print("Delta Height")
20                    #print(Δ_height)
21
22                    MD_current = value.data[0]*(-1)
23
24
25        elif package[0] == module_pb2.MODE_UNIT:
26            mode_data = package[1]
27
28            for value in mode_data:
29                if value.source == module_pb2.AUTONOMOUS_PHASE:
30                    phase = value.data[0]
31
32
33    # every 30 sec, the MD position is stored in phases 6,7,8.
34    # If, by performing a check every 30 sec when a new MD_old position is updated the
35    # bit has moved less than 0.2 mm in either direction; trigger stuck pipe.
36    # Stuck pipe remedial action: set WOB = 5 kg (upwards direction) for 1 second with 1 mm/s speed.
37    # Then, continue drilling.
38
39
40    if phase >= 6 or phase == 4:
41        MD_old = 0
42        if MD_old == 0:
43            MD_old = MD_current
44            md_counter -= 1
45
46        md_counter += 1
47        if md_counter >= 15:
48            if MD_current < MD_old + 0.15 and MD_current > MD_old - 0.15:
49                stuck = True
50                MD_old = MD_current
51                md_counter = 0
52
53        values = []
54        if stuck:
55            print("STUCK PIPE")
56            values.append(module_pb2.Values(data=[1.0], source=module_pb2.PIPE_STUCK))
57            suggestion = module_pb2.StreamData(pulse=pulse, timestamp=timestamp, ...
58            ↪ origin=module_pb2.STUCK_PIPE, values=values)
59            stuck = False
60            with suggestion_cv:
61                suggestion_items.append(suggestion)
62                suggestion_cv.notify()
63
64        else:
65            print("NO STUCK PIPE")
66            #values.append(module_pb2.Values(data=[0.0], source=module_pb2.STUCK))
67
68    ...
```

Appendix F

Autonomous Drilling Program Code

In this appendix, a short section of the implemented code for different phases in the implemented directional drilling program is included.

```

1  ...
2
3
4  def autonomous_operation(start_phase, stop_phase):
5      global autonomous_enabled
6      print("Autonomous operation initialized")
7      if start_phase == 1:
8          autonomous_phase(1)
9          # Control the rig fully autonomously. Operation is done in phases
10         # -----Phase 2: initialize and calibrate-----
11         if start_phase <= 2 and stop_phase >= 2:
12             print("-----Autonomous phase 2-----")
13             autonomous_phase(2)
14             # Zeroize hoisting position
15             print("Zeroizing hoisting position..")
16             zeroize_position(module_pb2.AUTONOMOUS)
17             # Zeroize whipstock position
18             print("Zeroizing whipstock position..")
19             zeroize_x_position(module_pb2.AUTONOMOUS)
20             zeroize_z_position(module_pb2.AUTONOMOUS)
21
22         # -----Phase 3: tag bottom-----
23         if start_phase <= 3 and stop_phase >= 3:
24             print("-----Autonomous phase 3-----")
25             autonomous_phase(3)
26             # Start WOB controller with -3 kg as setpoint
27             print("Setting WOB setpoint to -3.0 kg..")
28             wob_control_on(-3.0, 1.0, module_pb2.AUTONOMOUS)
29             # Wait until WOB is greater than 3 kg for 3 samples in a row
30             print("Waiting until WOB has reached -3.0 kg..")
31             autonomous_condition_wob(-3.0)
32             if not autonomous_enabled:
33                 print("Autonomous operation returned")
34                 stop_all()
35                 return
36
37             print("Stopping WOB agent..")
38             wob_control_off(module_pb2.AUTONOMOUS)
39
40             # Small wait before next operation
41             print("Waiting 3 second..")
42             autonomous_condition_wait(3)
43
44             # Zeroize hoisting position
45             print("Zeroizing hoisting position..")
46             zeroize_position(module_pb2.AUTONOMOUS)
47
48             # Move up from the bottom
49             print("Moving up from bottom..")
50             start_hoisting(5.0, 2.5, RAISE, module_pb2.AUTONOMOUS)
51
52             # Wait until operation is done
53             print("Waiting for operation to finish")
54             autonomous_condition_wait(5.0/2.5)
55             if not autonomous_enabled:
56                 print("Autonomous operation returned")
57                 stop_all()
58                 return
59
60         # -----Phase 4: drill pilot hole-----
61         if start_phase <= 4 and stop_phase >= 4:
62             print("-----Autonomous phase 4-----")
63             autonomous_phase(4)
64             # Set pneumatic motor setpoint to 500 RPM
65             print("Setting RPM setpoint to 500 RPM..")
66             rpm_setpoint_on(950, module_pb2.AUTONOMOUS)
67             # Start WOB controller with 2 kg as setpoint
68             print("Setting WOB setpoint to -2.0 kg..")
69             wob_control_on(-5.0, module_pb2.AUTONOMOUS)
70
71             # Start ROP agent to drill pilot hole
72             print("Starting ROP search..")
73             # TODO: find exact values
74             start_rop_search(rpm_min=1070, rpm_max=1070, wob_min=-12.0, wob_max=-10.0, . . .
75             ↪ controller=module_pb2.AUTONOMOUS)
76             # Wait until depth is greater than 185.5 mm

```

```

76     autonomous_condition_MD(-180.0)
77     if not autonomous_enabled:
78         print("Autonomous operation returned")
79         stop_all()
80         return
81
82     print("Stopping ROP agent..")
83     stop_rop_search()
84
85     # -----Phase 5: insert whipstock-----
86     if start_phase ≤ 5 and stop_phase ≥ 5:
87         print("-----Autonomous phase 5-----")
88         autonomous_phase(5)
89         # Set pneumatic motor setpoint to 800 RPM
90         print("Setting RPM setpoint to 800 RPM..")
91         rpm_setpoint_on(800, module_pb2.AUTONOMOUS)
92         # Reverse WOB controller while POOH
93         print("POOH with inversed WOB controller..")
94         wob_control_on(5.0, 4.0, module_pb2.AUTONOMOUS)
95
96         # Wait until above hole
97         print("Waiting until 5.0 mm out of hole..")
98         autonomous_condition_MD(35.0)
99         if not autonomous_enabled:
100             print("Autonomous operation returned")
101             stop_all()
102             return
103
104         # Setting WOB to POOH to 243 mm above rock to facilitate for whipstock positioning (x-axis lateral movement)
105         print("Setting WOB control with speed 10.0 mm/s")
106         wob_control_on(5.0, 10.0, module_pb2.AUTONOMOUS)
107         # Turning bit of
108         print("Closing valve..")
109         close_valve()
110
111         # Wait until bit is 10 mm above the riser
112         print("Pulling all the way out of the riser..")
113         # TODO: find exact value
114         autonomous_condition_MD(240.0)
115         if not autonomous_enabled:
116             print("Autonomous operation returned")
117             stop_all()
118             return
119
120         # Stop WOB controller
121         print("Stopping WOB controller..")
122         wob_control_off(module_pb2.AUTONOMOUS)
123         # Move whipstock in X direction with the whipstock
124         # TODO: find exact value
125         print("Moving whipstock X position..")
126         whipstock_x_command = "WhipstockX"
127         start_whipstock_x_positioning(128.0, 10.0, RAISE, module_pb2.AUTONOMOUS, whipstock_x_command)
128
129         # Wait until operation is done
130         print("Waiting until operation is done..")
131         autonomous_condition_status(module_pb2.WHIPSTOCK_UNIT, whipstock_x_command)
132         if not autonomous_enabled:
133             print("Autonomous operation returned")
134             stop_all()
135             return
136
137         # Move whipstock in Z direction into the hole
138         print("Inserting whipstock into hole..")
139         whipstock_z_command = "WhipstockZ"
140         # TODO: find exact value
141         start_whipstock_z_positioning(175.5, 3.0, LOWER, module_pb2.AUTONOMOUS, whipstock_z_command)
142         # Follow the whipstock system at the same speed going down
143         print("Lowering BHA with the whipstock..")
144         wob_control_on(-4.0, 3.0, module_pb2.AUTONOMOUS)
145
146         #autonomous_condition_wait((100.0 / 10) + (165 / 5.0))
147         print("Waiting until operation is done..")
148         autonomous_condition_status(module_pb2.WHIPSTOCK_UNIT, whipstock_z_command)
149         if not autonomous_enabled:
150             print("Autonomous operation returned")
151             stop_all()
152             return
153
154         ""# Start WOB controller with -4.0 kg as setpoint
155         print("Setting WOB setpoint to -4.0 kg..")
156         wob_control_on(-4.0, 10.0, module_pb2.AUTONOMOUS)""
157
158         # Wait until bit is 5 mm above the rock surface
159         print("Waiting until 50.0 mm above the rock..")
160         autonomous_condition_MD(50.0)
161         if not autonomous_enabled:
162             print("Autonomous operation returned")
163             stop_all()
164             return
165
166         # Start WOB controller with -4.0 kg as setpoint
167         print("Setting WOB setpoint to -4.0 kg..")
168         wob_control_on(-4.0, 3.0, module_pb2.AUTONOMOUS)
169         # Wait until bit is 0.0 mm above the rock surface
170         print("Waiting until 0.0 mm above the rock..")
171         autonomous_condition_MD(0.0)
172         if not autonomous_enabled:
173             print("Autonomous operation returned")
174             stop_all()
175             return
176
177         # -----Phase 6: expand pilot hole-----
178         if start_phase ≤ 6 and stop_phase ≥ 6:
179             print("-----Autonomous phase 6-----")
180             autonomous_phase(6)
181             # Start ROP search to expand pilot hole
182             print("Starting ROP search")
183             start_rop_search(rpm_min=1040, rpm_max=1040, wob_min=-5.0, wob_max=-5.0, . . .
184                             ↪ controller=module_pb2.AUTONOMOUS)
185             print("Waiting until 135.0 mm into the rock")

```

```

185     autonomous_condition_MD(-135.0)
186     if not autonomous_enabled:
187         print("Autonomous operation returned")
188         stop_all()
189         return
190     stop_rop_search(controller=module_pb2.AUTONOMOUS)
191
192     print("Starting ROP search..")
193     start_rop_search(rpm_min=1070, rpm_max=1070, wob_min=-3.0, wob_max=-3.0, . . .
194                    ↪ controller=module_pb2.AUTONOMOUS)
195     print("Waiting until 175 mm into the rock..")
196     autonomous_condition_MD(-175)
197     if not autonomous_enabled:
198         print("Autonomous operation returned")
199         stop_all()
200         return
201     stop_rop_search(module_pb2.AUTONOMOUS)
202
203     # -----Phase 7: drill deviated well path-----
204     if start_phase ≤ 7 and stop_phase ≥ 7:
205         print("-----Autonomous phase 7-----")
206         autonomous_phase(7)
207         # Start ROP search with closed loop steering
208         print("Starting ROP search..")
209         start_rop_search(rpm_min=1070, rpm_max=1070, wob_min=-18.0, wob_max=-15.0, . . .
210                        ↪ controller=module_pb2.AUTONOMOUS)
211
212     # -----Phase 8: penetrate rock-----
213     if start_phase ≤ 8 and stop_phase ≥ 8:
214         print("-----Autonomous phase 8-----")
215         autonomous_phase(8)
216         # Waiting until 5.0 mm through the rock
217         print("Waiting until through rock..")
218         autonomous_condition_MD(-600)
219         if not autonomous_enabled:
220             print("Autonomous operation returned")
221             stop_all()
222             return
223
224         # Stopping ROP agent
225         print("Stopping ROP search..")
226         stop_rop_search(module_pb2.AUTONOMOUS)
227
228         # Limit WOB setpoint to not break bottom plate
229         # Set pneumatic motor setpoint to 1070 RPM
230         print("Setting RPM setpoint to 1070 RPM..")
231         rpm_setpoint_on(1070, module_pb2.AUTONOMOUS)
232         print("Setting WOB to 8.0kg..")
233         wob_control_on(-8.0, 0.75, module_pb2.AUTONOMOUS)
234
235         print("Waiting to hit bottom plate..")
236         autonomous_condition_end()
237
238         # Mark autonomous mode as finished
239         stop_autonomous()
240         ""
241         # Set pneumatic motor setpoint to 200 RPM
242         print("Setting RPM setpoint to 200 RPM..")
243         rpm_setpoint_on(200, module_pb2.AUTONOMOUS)
244         # Reverse WOB controller while POOH
245         wob_control_on(10.0, module_pb2.AUTONOMOUS)
246         # Waiting until pulled out of whipstock
247         print("Waiting until above whipstock..")
248         # TODO: find exact value
249         autonomous_condition_MD(100.0)
250         if not autonomous_enabled:
251             print("Autonomous operation returned")
252             stop_all()
253             return""
254
255     # Stop all systems
256     stop_all(module_pb2.AUTONOMOUS)
257     print("Autonomous operation returned successfully")
258     autonomous_enabled = False
259
260 def rpm_voting_client(rpm_min, rpm_max):
261     global rpm_enabled, rpm_counter
262     print("RPM voting listener")
263     channel = grpc.insecure_channel(RPM_VOTING_LOCAL_PORT)
264     try:
265         grpc.channel_ready_future(channel).result(timeout=10)
266     except grpc.FutureTimeoutError:
267         print("RPM voting listener timeout")
268     else:
269         stub = module_pb2_grpc.ModuleStub(channel)
270
271         recommendations = stub.DataStream(module_pb2.Request(min=rpm_min, max=rpm_max))
272
273         client_id = 0
274         with rpm_lock:
275             client_id = rpm_counter
276
277         for recommendation in recommendations:
278             # Check if stream should still be active
279             ""if not rpm_enabled:
280                 # Break stream
281                 print("Breaking RPM stream..")
282                 return""
283
284             with rpm_lock:
285                 if client_id != wob_counter:
286                     print("Breaking WOB stream..")
287                     return
288
289             # Transform recommendation into output command
290             rpm = 0
291
292         for value in recommendation.values:

```



```

293         rpm = value.data[0]
294
295         command = module_pb2.Command(output=module_pb2.VALVE, controller=module_pb2.AUTONOMOUS,
296                                     operation=module_pb2.RPM_SET, amount=rpm)
297
298         with model_commands_cv:
299             model_commands_items.append(command)
300             model_commands_cv.notify()
301
302
303
304 def wob_voting_client(wob_min, wob_max):
305     print("WOB voting listener")
306     channel = grpc.insecure_channel(WOB_VOTING_LOCAL_PORT)
307     try:
308         grpc.channel_ready_future(channel).result(timeout=10)
309     except grpc.FutureTimeoutError:
310         print("RPM voting listener timeout")
311     else:
312         stub = module_pb2_grpc.ModuleStub(channel)
313
314         recommendations = stub.DataStream(module_pb2.Request(min=wob_min, max=wob_max))
315
316         client_id = 0
317         with wob_lock:
318             client_id = wob_counter
319
320         for recommendation in recommendations:
321             """if not wob_enabled:
322                 # Break stream
323                 print("Breaking WOB stream..")
324                 return"""
325
326         with wob_lock:
327             if client_id != wob_counter:
328                 print("Breaking WOB stream..")
329                 return
330
331         # Transform recommendation into output command
332         wob = 0
333
334         for value in recommendation.values:
335             wob = value.data[0]
336
337         command = module_pb2.Command(output=module_pb2.HOISTING, . . .
338                                     ↪ controller=module_pb2.AUTONOMOUS,
339                                     operation=module_pb2.WOB_SET, amount=wob, velocity=1.0, enable=True)
340
341         with model_commands_cv:
342             model_commands_items.append(command)
343             model_commands_cv.notify()
344     ...

```

Appendix G

Downhole Vibration Classification Code

In this appendix, a short section of the implemented code for downhole vibration classification using either a K-nearest neighbor or support vector machine model is included.

```
1 ...
2
3 def on_hoisting_downhole_sync(pulse, timestamp, data):
4     # Sync data received. Calculate values dependent of both hoisting and downhole
5     global mag_x_min, mag_y_min, mag_z_min, mag_x_max, mag_y_max, mag_z_max, downhole_initialized
6     global AZIMUTH_ERROR, INCLINATION_ERROR, MD_sum, x_disp_sum, y_disp_sum, ...
7     global MD_sum_max, min_error_x_sum, min_error_y_sum, min_error_z_sum, max_error_x_sum, ...
8     global x_disp_sum_max, y_disp_sum_max, z_disp_sum_max
9     global min_error_x_sum_max, min_error_y_sum_max, max_error_x_sum_max, max_error_y_sum_max
10    #print("Data sync")
11    acc_x = 0.0
12    acc_y = 0.0
13    acc_z = 0.0
14    mag_x = 0.0
15    mag_y = 0.0
16    mag_z = 0.0
17
18    Δ_height = 0.0
19
20    for package in data:
21        #print(package[0])
22
23        if package[0] == module_pb2.HOISTING_UNIT:
24            hoisting_data = package[1]
25
26            for value in hoisting_data:
27                if value.source == module_pb2.HEIGHT:
28                    Δ_height = value.data[-1] - value.data[0]
29                    #print("Delta Height")
30                    #print(Δ_height)
31
32            elif package[0] == module_pb2.DOWNHOLE:
33                downhole_data = package[1]
34                for value in downhole_data:
35                    if value.source == module_pb2.ACC_X:
36                        acc_x = value.data[0]
37                    elif value.source == module_pb2.ACC_Y:
38                        acc_y = value.data[0]
39                    elif value.source == module_pb2.ACC_Z:
40                        acc_z = value.data[0]
41                    elif value.source == module_pb2.MAG_X:
42                        mag_x = value.data[0]
43                    elif value.source == module_pb2.MAG_Y:
44                        mag_y = value.data[0]
45                    elif value.source == module_pb2.MAG_Z:
46                        mag_z = value.data[0]
47
48
49
50
51
52 def downhole_listener():
53     print("Downhole listener initialized")
54
55     loaded_model = pickle.load(open('TrainedSVC.sav', 'rb'))
56
57     #loaded_model = joblib.load('KNN_Model.sav')
58
59     """try:
60         with open("pickle/K_model.pkl", "rb") as f:
61             data = pickle.loads(f)
62     except pickle.UnpicklingError as e:
63         # normal, somewhat expected
64         print("Unpickling error")
65     except (AttributeError, EOFError, ImportError, IndexError) as e:
66         # secondary errors
67         print(traceback.format_exc(e))
68
69     except Exception as e:
```

```

70     # everything else, possibly fatal
71     print(traceback.format_exc(e))
72     return ""
73
74     sources = [module_pb2.GYR_X, module_pb2.GYR_Z]
75     stub = module_pb2_grpc.ModuleStub(grpc.insecure_channel(WINDOW_LOCAL_PORT))
76     status_data = stub.DataStream(module_pb2.Request(component=module_pb2.DOWNHOLE, window_size=60,
77     interval=60, sources=sources))
78
79     MAX = 250
80     MIN = -250
81
82     Vibrationlevel = 0.0
83
84     pulse = 0
85
86     time = 0
87
88
89
90     for data in status_data:
91         normalized_gyr_x = []
92         normalized_gyr_z = []
93
94         min_x = sys.float_info.max
95         max_x = sys.float_info.min
96
97         min_z = sys.float_info.max
98         max_z = sys.float_info.min
99
100        pulse = data.pulse
101        time = data.timestamp
102
103        for value in data.values:
104            if value.source == module_pb2.GYR_X:
105                for data in value.data:
106                    normalized = (data - MIN) / (MAX - MIN)
107                    normalized_gyr_x.append(normalized)
108                    min_x = min(normalized, min_x)
109                    max_x = max(normalized, max_x)
110
111            elif value.source == module_pb2.GYR_Z:
112                for data in value.data:
113                    normalized = (data - MIN) / (MAX - MIN)
114                    normalized_gyr_z.append(normalized)
115                    min_z = min(normalized, min_z)
116                    max_z = max(normalized, max_z)
117
118            #on_downhole_data(data)
119
120        min_max_diff_x = max_x - min_x
121        min_max_diff_z = max_z - min_z
122
123        std_x = np.std(normalized_gyr_x)
124        std_z = np.std(normalized_gyr_z)
125
126        data = [min_max_diff_z, max_z, min_x, min_z, std_x, std_z]
127
128        # Create the pandas DataFrame
129        Observations = pd.DataFrame([data], columns = ['min_max_diff_z', 'max_z', 'min_x', 'min_z', ...
130        ↪ 'std_x', 'std_z'])
131
132        # print dataframe.
133        #print(Observations)
134
135        Prediction = loaded_model.predict(Observations)
136
137        vib_data = 0
138
139        if Prediction == 1:
140            Vibrationlevel = 'Low'
141            vib_data = 25
142
143        elif Prediction == 2:
144            Vibrationlevel = 'Moderate'
145            vib_data = 100
146
147        elif Prediction == 3:
148            Vibrationlevel = 'High'
149            vib_data = 200
150
151        print("Vibration level is: ", Vibrationlevel)
152
153        values = []
154
155        values.append(module_pb2.Values(data=[vib_data], ...
156        ↪ source=module_pb2.DOWNHOLE_VIBRATIONS))
157
158        suggestion = module_pb2.StreamData(pulse=pulse, timestamp=time, ...
159        ↪ origin=module_pb2.DOWNHOLE_VIBRATIONS, values=values)
160
161        with suggestion_cv:
162            suggestion_items.append(suggestion)
163            suggestion_cv.notify()
164
165        ...

```

Appendix H

Downhole Position Tracking Code

In this appendix, a short section of the implemented code for downhole position tracking is included.

```
1 ...
2
3 def on_hoisting_downhole_sync(self, data):
4     # Sync data received. Calculate values dependent of both hoisting and downhole
5     global mag_x_min, mag_y_min, mag_z_min, mag_x_max, mag_y_max, mag_z_max, ...
6     ↪ downhole initialized
7     global AZIMUTH_ERROR, INCLINATION_ERROR, MD_sum, x_disp_sum, y_disp_sum, ...
8     ↪ z_disp_sum, TVD_sum, TVD_sum_max
9     global MD_sum_max, min_error_x_sum, min_error_y_sum, min_error_z_sum, ...
10    ↪ max_error_x_sum, max_error_y_sum, max_error_z_sum
11    global x_disp_sum_max, y_disp_sum_max, z_disp_sum_max
12    global min_error_x_sum_max, min_error_y_sum_max, max_error_x_sum_max, ...
13    ↪ max_error_y_sum_max
14    global acc_measurements, depth_counter
15    global topside_plot_counter, topside_graph_data_x, topside_graph_data_y, topside_graph_max_x
16    global topside_graph_max_y, topside_graph_min_x, topside_graph_min_y
17    global other_plot_counter, rear_graph_data_x, rear_graph_data_y, rear_graph_max_x
18    global rear_graph_max_y, rear_graph_min_x, rear_graph_min_y
19    global side_graph_data_x, side_graph_data_y, side_graph_max_x
20    global side_graph_max_y, side_graph_min_x, side_graph_min_y
21    global tag_bottom, prev_depth, rop_prev_depth
22    global x_disp_last, error_x, maximum_depth
23    global rop_list, rop_inst_counter, reset_counters
24    global acc_x, acc_y, acc_z, mag_x, mag_y, mag_z, gyr_x, gyr_y, gyr_z, vib_class
25    global mode, depth, Δ_depth, time, time_started
26    global side_graph_pilot_x, side_graph_pilot_y, rear_graph_pilot_x, rear_graph_pilot_y
27    #print("Data sync")
28    """
29    acc_x = 0.0
30    acc_y = 0.0
31    acc_z = 0.0
32    mag_x = 0.0
33    mag_y = 0.0
34    mag_z = 0.0
35    """
36
37    #build_angle = 0.0
38
39    for package in data:
40        print(package[0])
41
42        time = int(package[2])
43
44        if package[0] == module_pb2.HOISTING_UNIT:
45            hoisting_data = package[1]
46
47            for value in hoisting_data:
48                if value.source == module_pb2.HEIGHT:
49                    #Height is positive upwards and negative downwards, we want ...
50                    ↪ depth to be positive
51                    #Δ_depth = (value.data[-1] - value.data[0])*(-1)
52                    Δ_depth = rop_prev_depth - value.data[0]
53                    depth = value.data[0]*(-1)
54                    rop_prev_depth = value.data[0]
55                    print("Delta Depth: ", Δ_depth)
56                    print("Depth: ", depth)
57
58            elif package[0] == module_pb2.DOWNHOLE:
59                downhole_data = package[1]
60                for value in downhole_data:
61                    if value.source == module_pb2.ACC_X:
62                        acc_x = value.data[0]
63                    elif value.source == module_pb2.ACC_Y:
64                        acc_y = value.data[0]
65                    elif value.source == module_pb2.ACC_Z:
66                        acc_z = value.data[0]
67
68                    elif value.source == module_pb2.GYR_X:
69                        gyr_x = value.data[0]
70                    elif value.source == module_pb2.GYR_Y:
71                        gyr_y = value.data[0]
72                    elif value.source == module_pb2.GYR_Z:
73                        gyr_z = value.data[0]
74
75                    elif value.source == module_pb2.MAG_X:
```

```

72         mag_x = value.data[0]
73         #mag_x_min = min(mag_x_min, value.data[0])
74         #mag_x_max = max(mag_x_max, value.data[0])
75     elif value.source == module_pb2.MAG_Y:
76         mag_y = value.data[0]
77         #mag_y_min = min(mag_y_min, value.data[0])
78         #mag_y_max = max(mag_y_max, value.data[0])
79     elif value.source == module_pb2.MAG_Z:
80         mag_z = value.data[0]
81         #mag_z_min = min(mag_z_min, value.data[0])
82         #mag_z_max = max(mag_z_max, value.data[0])
83
84     elif package[0] == module_pb2.DOWNHOLE_VIBRATIONS:
85         vib_data = package[1]
86         print("Vibration data: ", vib_data)
87         for value in vib_data:
88             if value.source == module_pb2.DOWNHOLE_VIBRATIONS:
89                 vib_class = value.data[0]
90                 '''
91                 if vib_data == 'Low':
92                     vib_class = 25
93                 elif vib_data == 'Moderate':
94                     vib_class = 100
95                 elif vib_data == 'High':
96                     vib_class = 200
97                 else:
98                     print("Vib_data ERROR")
99                     vib_data = 0
100                 '''
101
102     elif package[0] == module_pb2.MODE_UNIT:
103         mode_data = package[1]
104         print("MODE: ", mode_data)
105         for value in mode_data:
106             if value.source == module_pb2.AUTONOMOUS_PHASE:
107                 mode = value.data[0]
108                 print("Autonomous mode: ", mode)
109
110         '''
111     elif package[0] == module_pb2.ACTIVE_STEERING:
112         steering_data = package[1]
113         print("Active steering data: ", steering_data)
114         for value in steering_data:
115             if value.source == module_pb2.BUILD_ANGLE:
116                 build_angle = value.data[0]'''
117
118
119     print("MODE RECIEVED: ", mode)
120
121     if len(rop_list) < rop_inst_counter:
122         rop_list.append(depth)
123     elif len(rop_list) == rop_inst_counter:
124         tmp2_rop = rop_list[I:]
125         tmp2_rop.append(depth)
126         rop_list = tmp2_rop
127
128     else:
129         print("Skipped rop value calculation for GUI, CHECK!")
130
131     if mode > 0:
132         if time_started == 0:
133             time_started = time
134
135     #Check if the drillpipe is moving downwards or upwards and check if bottom has been tagged
136     '''
137     if not tag_bottom:
138         if prev_depth > 0:
139             if prev_depth > depth*20:
140                 print("Bottom has been tagged, can begin plotting!")
141                 tag_bottom = True
142
143     prev_depth = depth
144     '''
145
146     # Calculate inclination
147     if acc_x == 0 and acc_y == 0 and acc_z == 0:
148         inclination = 90
149     else:
150         inclination = . . .
151         #<math>180.0 * \text{math.acos}(\text{acc}_y / \text{math.sqrt}(\text{acc}_x * \text{acc}_x + \text{acc}_y * \text{acc}_y + \text{acc}_z * \text{acc}_z)) / \text{math.pi}</math>
152         ###was acc_z / ...
153
154     # Calculate azimuth
155     mag_x -= (mag_x_min + mag_x_max) / 2
156     mag_y -= (mag_y_min + mag_y_max) / 2
157     mag_z -= (mag_z_min + mag_z_max) / 2
158
159     azimuth = 180 * math.atan2(mag_z, mag_x) / math.pi
160     '''azimuth = 180 * math.atan2(mag_y, mag_x) / math.pi'''
161
162     if azimuth < 0:
163         azimuth += 360
164
165     #Setting global acc variables for gui to get
166     with acc_cv:
167         acc_measurements[3] = acc_x
168         acc_measurements[2] = acc_y
169         acc_measurements[1] = acc_z
170         acc_measurements[0] = vib_class
171
172     if acc_x == 0 and acc_y == 0 and acc_z == 0:
173         accXnorm = 0
174         accYnorm = 0
175     else:
176         accXnorm = acc_x / math.sqrt(acc_x * acc_x + acc_y * acc_y + acc_z * acc_z)
177         accYnorm = acc_z / math.sqrt(acc_x * acc_x + acc_y * acc_y + acc_z * acc_z)
178
179     #Calculate pitch and roll: <<<<-----
180     pitch = math.asin(accXnorm)

```

```

181 roll = -math.asin(max(-1, min(abs(accYnorm/math.cos(pitch)), 1)))
182 #max(-1, min(pitch, 1))
183
184 #Calculate the new tilt compensated values:
185 mag_x_comp = mag_x * math.cos(pitch) + mag_y * math.sin(pitch)
186 mag_y_comp = mag_x * math.sin(roll) * math.sin(pitch) + mag_z * math.cos(roll) - mag_y * ...
    ↪ math.sin(roll) * math.cos(pitch)
187
188 #tiltCompensatedAzimuth = math.pi * math.atan2(mag_y_comp, mag_x_comp) / 180
189
190 tiltCompensatedAzimuth = 180 * math.atan2(mag_y_comp, mag_x_comp) / math.pi
191
192 #self.ui.actuatorZ3Lcd.display(float("{0:.2f}".format(value.data[0])))
193 print("MAG X: ", mag_x)
194 print("MAG Y: ", mag_y)
195 print("MAG Z: ", mag_z)
196
197 if tiltCompensatedAzimuth < 0:
198     tiltCompensatedAzimuth += 360
199
200
201 print("Azimuth raw data: ", tiltCompensatedAzimuth)
202 if not downhole_initialized:
203     INCLINATION_ERROR = inclination - 90
204     AZIMUTH_ERROR = tiltCompensatedAzimuth - 360
205     downhole_initialized = True
206 else:
207     tiltCompensatedAzimuth -= AZIMUTH_ERROR
208     inclination -= INCLINATION_ERROR
209
210     if tiltCompensatedAzimuth > 360:
211         tiltCompensatedAzimuth -= 360
212
213         ""tiltCompensatedAzimuth -= AZIMUTH_ERROR
214         inclination -= INCLINATION_ERROR
215
216     if tiltCompensatedAzimuth > 360:
217         tiltCompensatedAzimuth -= 360""
218
219 if 15 < tiltCompensatedAzimuth < 180:
220     tiltCompensatedAzimuth = 15
221 elif 180 < tiltCompensatedAzimuth < 345:
222     tiltCompensatedAzimuth = 345
223
224 if inclination > 92 and mode ≤ 5:
225     inclination = 92
226 elif inclination > 95 and (mode ≥ 6 or mode == 0):
227     inclination = 95
228 elif inclination < 88 and mode ≤ 5:
229     inclination = 88
230 elif inclination < 75 and (mode ≥ 6 or mode == 0):
231     inclination = 75
232
233 print("AZIMUTH: ", str(tiltCompensatedAzimuth), " degrees")
234 print("INCLINATION: ", str(inclination), " degrees")
235
236
237 #if 180 > tiltCompensatedAzimuth > 15:
238 # tiltCompensatedAzimuth = 15
239 #elif 180 ≤ tiltCompensatedAzimuth < 345.:
240 # tiltCompensatedAzimuth = 345
241
242
243 # MD (Measured depth) denotes the change in actuator position over the 300(?) sample window.
244 MD = Δ_depth
245 MD2 = ddepth
246
247 print("MD2:", MD2)
248
249 #print("MD", str(MD))
250
251 # MD_sum denotes the sum of all previous TVD values, and is used to track the bit elevation in ...
252 ↪ real-time
253 # NB! SHOULD BE A GLOBAL VARIABLE
254
255 #MD_sum += MD
256 MD_sum = depth
257 MD_sum_max = max(MD_sum, MD_sum_max)
258
259 print("MD TOTAL: ", str(MD_sum), " mm")
260
261 # TVD (True vertical depth) denotes the vertical change in actuator position over the 300 sample ...
262 ↪ window (given that
263     # even if the pipe moves down, it is not guaranteed that the movement is ...
264     ↪ perfectly 90
265     # degree downwards).
266 TVD = MD * math.sin(inclination * (math.pi / 180))
267
268 #print("TVD", str(TVD))
269
270 TVD_sum += TVD
271
272 TVD_sum_max = max(TVD_sum, TVD_sum_max)
273
274 print("TVD TOTAL: ", str(TVD_sum_max), " mm")
275
276 # x_disp (horizontal build) denotes the horizontal movement that is built in each well section ...
277 ↪ depending on inclination
278 x_disp = (MD) * math.cos(inclination * (math.pi / 180))
279
280 print(x_disp)
281 #print("x_disp", str(x_disp))
282
283 # y_disp (azimuth offset) denotes the azimuthal displacement that is built in each well section ...
284 ↪ depending on x_disp and azimuth
285
286 # CHANGED y_disp = x_disp * ... to y_disp = MD * ...
287
288 y_disp = x_disp * math.sin(tiltCompensatedAzimuth * (math.pi / 180))

```

```

287
288
289
290
291
292
293
294
295
296
297
298
299
300
301
302
303
304
305
306
307
308
309
310
311
312
313
314
315
316
317
318
319
320
321
322
323
324
325
326
327
328
329
330
331
332
333
334
335
336
337
338
339
340
341
342
343
344
345
346
347
348
349
350
351
352
353
354
355
356
357
358
359
360
361
362
363
364
365
366
367
368
369
370
371
372
373
374
375
376
377
378
379
380
381
382
383
384
385
386
387
388
389
390
391
392
393
394
395

# print("y_disp", str(y_disp))
# z_disp is equal to the vertical displacement, that is the True Vertical Depth (TVD)
z_disp = TVD
# print("z_disp", str(z_disp))
# x_disp_sum denotes the sum of all x_disp in the past so that
# NB! SHOULD BE A GLOBAL VARIABLE
if inclination <= 90:
    x_disp_sum += x_disp
    x_disp_sum_max = max(x_disp_sum, x_disp_sum_max)
elif inclination > 90:
    x_disp_sum += x_disp
    x_disp_sum_max = min(x_disp_sum, x_disp_sum_max)
print("HORIZONTAL BUILD: ", str(x_disp_sum_max), " mm")
# y_disp_sum denotes the sum of all y_disp in the past so that
# NB! SHOULD BE A GLOBAL VARIABLE
if 0 <= tiltCompensatedAzimuth <= 15:
    y_disp_sum += y_disp
    y_disp_sum_max = max(y_disp_sum, y_disp_sum_max)
elif 345 <= tiltCompensatedAzimuth < 360:
    y_disp_sum += y_disp
    y_disp_sum_max = min(y_disp_sum, y_disp_sum_max)
print("AZIMUTH OFFSET: ", str(y_disp_sum_max), " mm")
# z_disp_sum denotes the sum of all z_disp when length change is positive
# NB! SHOULD BE A GLOBAL VARIABLE
z_disp_sum += z_disp
z_disp_sum_max = min(z_disp_sum, z_disp_sum_max)
# print("z_disp_sum_max", str(z_disp_sum_max))
# min_error_x takes into account that the x_disp could be 10% smaller due to inclination . . .
# measurement errors
min_error_x = (x_disp) * 0.75
# max_error_x takes into account that the x_disp could be 10% greater due to inclination . . .
# measurement errors
max_error_x = (x_disp) * 1.25
# min_error_y takes into account that the y_disp could be 15% smaller due to azimuth . . .
# measurement errors
min_error_y = (y_disp) * 0.50
# max_error_y takes into account that the y_disp could be 15% greater due to azimuth . . .
# measurement errors
max_error_y = (y_disp) * 1.50
# min_error_x_sum is the sum of all errors cumulative
# min_error_x_sum_max += min_error_x
error_delta_x = (x_disp_last - x_disp) * 0.75
x_disp_last = x_disp
if MD > maximum_depth:
    error_x += abs(error_delta_x)
    maximum_depth = MD
if inclination <= 90:
    min_error_x_sum += min_error_x
    min_error_x_sum_max = max(min_error_x_sum, min_error_x_sum_max)
elif inclination > 90:
    min_error_x_sum -= abs(min_error_x + x_disp)
    min_error_x_sum_max = min(min_error_x_sum, min_error_x_sum_max)
print("MIN X:", str(min_error_x_sum_max), " mm")
# min_error_y_sum is the sum of all errors cumulative
if 0 <= tiltCompensatedAzimuth <= 15:
    min_error_y_sum += min_error_y
    min_error_y_sum_max = max(min_error_y_sum, min_error_y_sum_max)
elif 345 <= tiltCompensatedAzimuth < 360:
    min_error_y_sum += min_error_y
    min_error_y_sum_max = min(min_error_y_sum, min_error_y_sum_max)
print("MIN Y:", str(min_error_y_sum_max), " mm")
# max_error_x_sum is the sum of all errors cumulative
# max_error_x_sum_max += max_error_x

```

```
396         if inclination <= 90:
397             max_error_x_sum += max_error_x
398             max_error_x_sum_max = max(max_error_x_sum, max_error_x_sum_max)
399
400         elif inclination > 90:
401             max_error_x_sum -= abs(max_error_x - x_disp)
402             max_error_x_sum_max = min(max_error_x_sum, max_error_x_sum_max)
403
404         print("MAX X:", str(max_error_x_sum_max), " mm")
405
406     # max_error_Y_sum is the sum of all errors cumulative
407
408     if 0 <= tiltCompensatedAzimuth <= 15:
409         max_error_y_sum += max_error_y
410         max_error_y_sum_max = max(max_error_y_sum, max_error_y_sum_max)
411
412     elif 345 <= tiltCompensatedAzimuth < 360:
413         max_error_y_sum += max_error_y
414         max_error_y_sum_max = min(max_error_y_sum, max_error_y_sum_max)
415
416     print("MAX Y:", str(max_error_y_sum_max), " mm")
417
418     ...
419
420
421
422
423
424
425
426
427
428
429
```