




University
of Stavanger

FACULTY OF SCIENCE AND TECHNOLOGY

MASTER'S THESIS

Study programme/specialization:	Spring semester, 2022
MSc. Petroleum Engineering	Open
Author: Alina Shashel	 Alina Shashel
Programme coordinator: Øystein Arild Supervisors: Postdoktor Jie Cao Prof. Dan Sui	
Title of master's thesis: Uncertainty analysis of Supervised machine learning predictions applied to Lithology classification	
Credits: 30	
Keywords: Supervised Machine Learning, Uncertainty Quantification, Model evaluaton, Real-time drilling, Volve, Lithology classification.	Number of pages: 86 + supplemental material/other: 13 Stavanger, 15th July 2022

Alina Shashel

Uncertainty analysis of Supervised machine learning predictions applied to Lithology classification

Master Thesis Project for the degree of
MSc in Petroleum Engineering

Stavanger, July 2022

University of Stavanger
Faculty of Science and Technology
Department of Energy and Petroleum Engineering



Abstract

Geosteering is the technique of guiding directional drilling to remain within the pay zone. This process demands a thorough survey of the lithological properties of the surrounding geological strata. Since logging while drilling (LWD) tools are positioned a few meters above the bit, it generates depth lag and, thus, a time delay between what the LWD sensors report to the surface and the performance of the bit. Drill bit and drill string performance factors are the earliest markers to determine formations' characteristics without the temporal delay.

Implementing automated lithology identification would enhance the quality of the geosteering operation. This thesis investigated the extent to which various supervised machine learning (ML) classification algorithms may be utilized to recognize the lithological features of drilled formations.

ML models were trained using preprocessed real-time drilling data from the Volve field. The data included nine wells with a total of 198 928 tagged observations and the accompanying measured parameters at various depths within the wells. The ML algorithms were tested on the selected well with a minority of samples presented in the dataset.

The progress in ML algorithms application provides an incentive for more study on model trustworthiness, including uncertainty analysis, to improve classification algorithms used in lithology identification. Most ML algorithms may be thought of as "black box" models, meaning that the process by which variables are integrated to form predictions cannot be seen or transparently understood. Hence, it is required to quantify and limit the uncertainties in models' performance to apply ML to real-life classification problems successfully.

Within the scope of this research, Feature Sensitivity and Vulnerability Analysis, as well as Dataset shift Measurement, were applied to investigate the reliability of ML models. A novel Black Box Metamodel approach and Bayesian Neural Networks were employed to compute aleatoric and epistemic uncertainties.

After testing seven ML classification algorithms, the Random Forest and Adaptive Boosting ones demonstrated the most accurate results and were chosen for comparative reliability analysis.

In classification tasks, it is more crucial to estimate the probability that an observation belongs to a specific class than the prediction results. Consequently, the Probability Calibration techniques improved the quality of the quantified uncertainties. It was proven that the Adaptive Boosting algorithm with the better scoring results is less confident and ambiguous regarding epistemic uncertainty than the Random Forest one after calculating and comparing the difference between the confidence and accuracy results obtained after the Probability Calibration.

Keywords – Supervised Machine Learning, Uncertainty Quantification, Model evaluation, Real-time drilling, Volve, Lithology classification

Acknowledgments

I would like to express my gratitude towards my supervisors Postdoctoral researcher Jie Cao and Professor Dan Sui at UiS. Their guidance and valuable advice carried me through all the stages of writing my thesis.

Many thanks to my parents Vadim and Elena and little sister Uliana for having a space for me in their minds and hearts regardless of the distance. And to my incredible grandmothers for their understanding and limitless support. Special thanks to my partner Miguel who went through this challenging journey with me.

Per Aspera Ad Astra!

Alina Shashel

Table of Contents

Abstract	ii
Acknowledgments	iii
List of Figures	x
List of Tables	xi
1 Introduction	1
1.1 Background and Motivation	1
1.2 Thesis Structure	2
2 Disclosed Volve dataset	4
2.1 The Volve Dataset	4
2.2 General Information	4
2.3 Geology	5
2.3.1 Lithostratigraphy Description	5
2.4 Wells	8
2.5 Drilling data	9
3 Methodology	12
3.1 Supervised Machine Learning Classification	12
3.1.1 K-Nearest Neighbor	12
3.1.2 Decision Trees	14
3.1.3 Logistic Regression	15
3.1.4 Ensemble Learning	16

3.1.5	Gradient Boosting	19
3.1.6	Naïve Bayes	21
3.2	Classification on Imbalanced Data	22
3.2.1	Oversampling and Undersampling	22
3.2.2	Class weights	25
3.3	Model Analysis	26
3.3.1	Model Evaluation Metrics	26
3.3.2	ROC curve	28
3.3.3	Precision-Recall (PR) curve	29
3.3.4	Dataset Shift	29
3.3.5	Model Interpretability	31
3.4	Uncertainty Quantification	34
3.4.1	Intrinsic and Extrinsic UQ Algorithms	34
3.4.2	Calibration	35
3.4.3	Metamodel	38
3.4.4	Bayesian Neural Network	39
4	Data Analytics	42
4.1	Data Labeling	42
4.2	Lithology Classes Distribution	42
4.3	Feature creation	45
4.4	Data Quality	46
4.5	Feature Selection	48
4.5.1	Wrapper Methods	49
4.5.2	Filter Methods	49
4.5.3	Embedded methods	50
5	Classification Results	52
5.1	Train/Test Split	52
5.2	Categorical Encoding	54

5.3	Experimental Setting 1 - Gradient Boosting	55
5.4	Experimental Setting 2 - Decision Tree	57
5.5	Experimental Setting 2 - Random Forest	60
5.6	Experimental Setting 4 - Adaptive Boosting	62
6	ML Model Analysis - Case Study	65
6.1	Model Evaluation	65
6.1.1	ROC and Precision-Recall Curves	65
6.2	Feature Sensitivity and Model Vulnerability	66
6.3	Covariate Shift Measurement	68
6.4	Uncertainty Quantification	70
6.4.1	Blackbox Metamodel Classification	71
6.4.2	Bayesian Neural Network	73
6.4.3	Probability Calibration	75
7	Conclusions and Future Work	81
	References	85
	Appendices	86
	Appendix A Python Code	88
A.1	Installed Packages	88
A.2	Decision Tree	88
A.3	Random Forest	89
A.4	Gradient Boosting	89
A.5	Adaptive Boosting	90
A.6	Model Analysis	90
A.7	Blackbox Metamodel	92
A.8	Isotonic Regression Recalibration	93
A.9	BNN	93

Appendix B Lithology columns for the tested wells F-14, F-15, F-15S	96
--	-----------

List of Figures

1.1	Thesis workflow	3
2.1	Location of Volve field in the North Sea. Source: [1]	5
2.2	Regional stratigraphic sequence of the study area, Volve field. Source: [1]	6
3.1	A simple example of 3-Nearest Neighbour Classification. Source: [2]	13
3.2	Result of the Decision Tree for Classification	15
3.3	Sigmoid Function	16
3.4	Example of a random forest. Source: [3]	17
3.5	An illustration of synthetic data points in the SMOTE algorithm. Source: [4]	24
3.6	Confusion matrix for binary classification	26
3.7	Examples of a PR curve and a ROC curve. The red plot shows ideal PR and ROC curves. The blue plot shows some PR and ROC curves, as they typically arise in experiments. Arrows indicate the increase of the threshold level value. Source: [5]	29
3.8	Illustrative explanation of Covariate Shift. Source: [6]	30
3.9	Toy example to present intuition for LIME. Source: [7]	33
3.10	Taxonomy of the uncertainty estimation algorithms included in the UQ360 toolkit. Source: [8]	34
3.11	Example of Confidence Histogram and Reliability Plot.	37
3.12	The concept of meta modeling. Source: [9]	39
3.13	Workflow to design (a), train (b) and use a BNN for predictions (c). Source: [10]	40
3.14	Difference between Standard NN Bayesian NN. Source: [11]	40

4.1	Distribution of the formations in the data set. The figure illustrates the skewness in the percentage distribution of the formations.	43
4.2	Lithology columns for wells F-5, F-14, F-15, F-15S.	44
4.3	Lithology columns for wells F-7, F-9, F-9a.	45
4.4	Distribution of the formations in the data set after elimination of wells F-9 and F-9a to balance the class distribution.	46
4.5	Lithology column for the well F-14 before and after implementing SMOTE technique.	47
4.6	Outlier detection with boxplots.	48
4.7	Descriptive statistics of the dataset.	48
4.8	Cross correlation plot.	51
4.9	Feature Information Gain.	51
5.1	Train/test split process.	52
5.2	Lithology columns	53
5.3	The samples percentage for each well in the dataset.	54
5.4	Histogram of the class distribution the training and testing sets.	54
5.5	Lithology column for tested well F-5 and predicted lithology classes - Gradient Boosting algorithm.	56
5.6	Confusion matrix for Gradient Boosting algorithm.	57
5.7	Lithology column for tested well F-5 and predicted lithology classes - Decision Tree algorithm.	59
5.8	Confusion matrix for Decision Tree algorithm.	59
5.9	Lithology column for tested well F-5 and predicted lithology classes - Random Forest algorithm.	61
5.10	Confusion matrix for Random Forest algorithm.	61
5.11	Lithology column for tested well F-5 and predicted lithology classes - Adaptive Boosting algorithm.	63
5.12	Confusion matrix for Adaptive Boosting algorithm.	63

6.1 ROC- and PR-curves for Random Forest algorithm. 66

6.2 ROC- and PR-curves for Adaptive Boosting algorithm. 66

6.3 Feature Sensitivity 67

6.4 Models Vulnerability 67

6.5 Histogram of the shifted features. 69

6.6 Distribution of the shifted features. 69

6.7 Guidance on choosing UQ algorithms. Source: [12] 70

6.8 The synthetic example of Risk vs Rejection and Risk vs Selection Threshold plots for well performed model. 72

6.9 Metamodel results for Random Forest algorithm. 73

6.10 Metamodel results for Adaptive Boosting algorithm. 73

6.11 BNN Uncertainties based on Random Forest’s probabilities score for different classes 74

6.12 Perfectly Calibrated Model. 75

6.13 Calibration plot for the Random Forest algorithm. 75

6.14 Calibration plot for the Adaptive Boosting algorithm. 76

6.15 Recalibration plot for the Random Forest algorithm. 76

6.16 Confidence Histogram and Reliability plot for each class (Random Forest algorithm). 77

6.17 Confidence Histogram and Reliability plot - Average (Random Forest algorithm). 77

6.18 Recalibration plot for the Adaptive Boosting algorithm. 78

6.19 Confidence Histogram and Reliability plot for each class (Adaptive Boosting algorithm). 78

6.20 Confidence Histogram and Reliability plot - Average (Adaptive Boosting algorithm). 79

B.1 Lithology columns of tested well F15 and predicted outcomes 97

B.2 Lithology columns of tested well F15S and predicted outcomes 98

B.3 Lithology columns of tested well F14 and predicted outcomes 99

List of Tables

4.1	Class weights.	46
4.2	Number or percentage of outliers for the features.	49
5.1	Encoded labels for the lithology classes.	55
5.2	Scoring metrics for the Gradient Boosting algorithm.	57
5.3	Scoring metrics for the Decision Tree algorithm.	58
5.4	Scoring metrics for the Random Forest algorithm.	60
5.5	Scoring metrics for the Adaptive Boosting algorithm.	64
6.1	Aleatoric and epistemic uncertainties for the Random Forest and Adaptive Boosting algorithms.	74
6.2	The results after calculation the Accuracy - Confidence gap for Random Forest and Adaptive Boosting algorithms.	79
6.3	Accuracy scores for the wells: F-15, F-15S, F-14	80
A.1	Required packages.	89

Chapter 1

Introduction

1.1 Background and Motivation

Geosteering is the process of directional steering the drilling according to the geology. It requires understanding the lithology properties of the formation surrounding it. Despite obtaining this information through logging while drilling (LWD), a cost-efficient and almost real-time solution is lacking. In general, there is a depth lag, and therefore, a temporal delay, between what the LWD tool transmits to the surface and the bit location at the current time (Gupta et al., 2020) [13].

Moreover, the incorporation of real-time drilling data enables asset monitoring teams to simultaneously update the static and dynamic reservoir model with the data collected from newly drilled wells.

Throughout this study, drill-bit, and drillstring-performance data is used in a machine learning (ML) pipeline to predict the lithology classes in the drilled wells.

Even though ML holds much promise, its results are not entirely unreliable because of the challenges brought about by uncertainties. ML models generate optimal predictions based on the training data. When uncertainties in data and algorithms are not considered, these optimal predictions will likely fail in real-life deployment.

Most studies on ML application in petroleum engineering aim to train the model and obtain satisfactory prediction results. Contrary to that, the model quality evaluation and uncertainty quantification (UQ) receive too little attention.

ML model analysis and UQ methods employed in this study are essential in detecting weaknesses in models' performance and reducing the impact of uncertainties during decision-making processes.

While training the model for lithology classification, it is crucial to have an insight into the prediction boundaries of the model. This will help to determine the probability of each class being correctly identified. Since, in real life, the trained model would be tested on unseen data acquired from the newly drilled wells. Therefore, it will be reliable only if the uncertainties are eliminated, and the user is sure about the model's prediction efficiency.

1.2 Thesis Structure

This thesis is divided into seven chapters. Chapter 2 provides a description of the formations' lithostratigraphy and essential background information on drilling parameters and wells included in the dataset. Chapter 2 covers the methodological framework of the ML classification algorithms applied to lithology identification in addition to model analysis and uncertainty quantification approaches. Chapter 4 addresses the data preprocessing before its introduction to the models. The findings of four modeling experiments are presented in Chapter 5. Chapter 6 provides the comparative analysis of model assessment and uncertainty quantification. The validity of the model's performance is discussed in this chapter, along with ideas for its further enhancement. Chapter 7 concludes the finding of the thesis and proposes further work on the research.

The reserach workflow is presented on the Diagram 1.1.

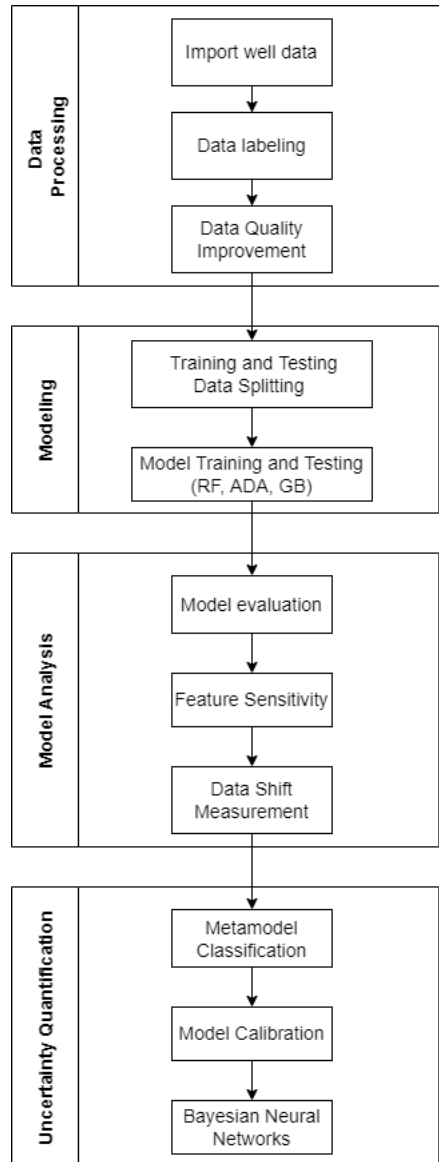


Figure 1.1: Thesis workflow

Chapter 2

Disclosed Volve dataset

2.1 The Volve Dataset

Equinor released the data for subsurface and production in June 2018. This dataset has approximately 40 000 files, which represent all phases of the field development. The most important folders are those containing well data, real-time drilling data, daily reports, and final reports.

Initially, real-time drilling data is stored in WITSML format, a standard one for transmitting technical data between organizations in the petroleum industry. The parsed and pre-processed dataset in CSV format developed by Andrzej Tunkiel, Tomasz Wiktorski, and Dan Sui (Tunkiel et al., 2020) [14] is used for lithology classification.

2.2 General Information

Located 200 kilometers west of Stavanger at the southern end of the Norwegian sector, Volve is a shallow-water oil field discovered in 1993 in the central part of the North Sea (Figure 2.1). A jack-up drilling and processing facility was installed at the field. With a life expectancy of about 3-5 years, the field started well drilling in 2007 and started producing after pressure support from water injection in February 2008. During the peak of production, Volve produced about 56,000 barrels of oil per day, delivering 63 million barrels more than anticipated. With a recovery rate of 54% of reserve estimates, it was shut down in September 2016 after operating for over 8 years.

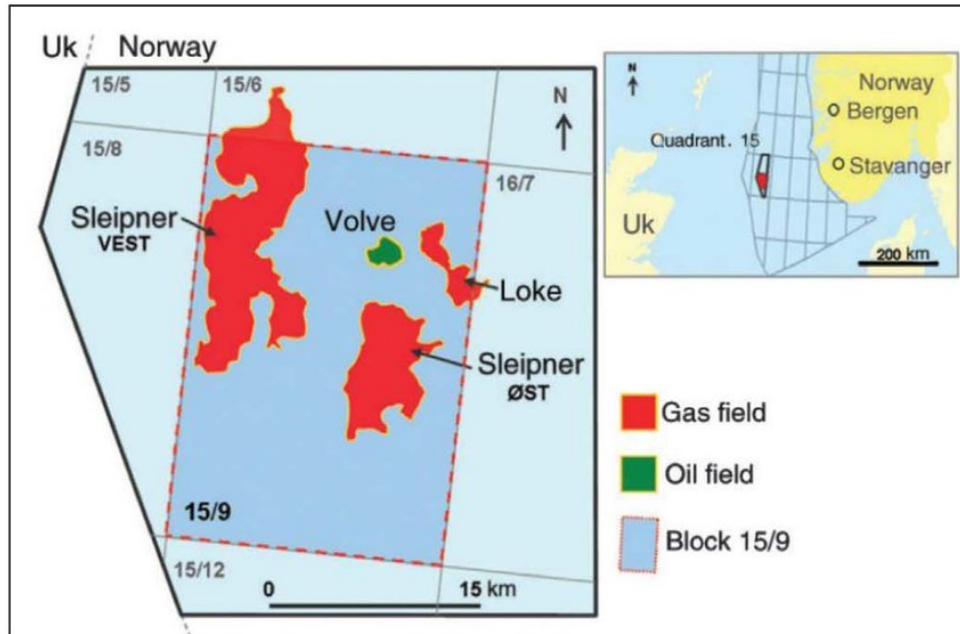


Figure 2.1: Location of Volve field in the North Sea. Source: [1]

2.3 Geology

Volve extracted oil from middle Jurassic sandstones of the Hugin formation. It is believed that the dome-shaped reservoir was formed by the downfall of contiguous salt ridges during the Jurassic period. This field is characterized by salt tectonic faults. Western parts of the structure are heavily faulted, with most of the faults being influenced by regional extension. Communication across the faults is uncertain. This reservoir is located at a depth of 2750-3120 m TVDSS, and is about 20 m thick at the crest, reaching up to 100 m thick on either side of the structure (Ganguli et al., 2019) [1].

Generalized regional stratigraphy has been presented in Table 2.2.

2.3.1 Lithostratigraphy Description

The formation subdivision is based on wireline logs, lithostratigraphy, biostratigraphy of well 15/9-19 SR, and correlation to neighbouring wells (Discovery Evaluation Report Well 15/9-19 SR, Statoil) [?].

Table 1—Regional stratigraphic sequence of the study area, Volve field, as encountered in the studied wells.

System	Group	Formation	Lithology
Quaternary	Nordland	Recent sediments	Sandstone, Claystone
		Upper Utsira Formation	Sandstone
Tertiary	Hordaland	Lower Utsira Formation	Claystone
		Skade Formation	Sandstone, Claystone
		Grid Formation	Sandstone, Claystone
	Rogaland	Balder Formation	Claystone, Anhydrite
		Sele Formation	Claystone
		Lista Formation	Claystone
		Ty Formation	Sandstone, Claystone, minor Limestone
Cretaceous	Shetland	Ekofisk Formation	Limestone
		Tor Formation	Limestone
		Hod Formation	Limestone
		Blodoeks Formation	Limestone
		Hidra Formation	Limestone
	Cromer Knoll	Roedby Formation	Marl
		Asgard Formation	Marl, Limestone
Jurassic	Viking	Draupne Formation	Claystone, minor Limestone
		Heather Formation	Claystone
	Vestland	Hugin Formation	Sandstone, minor Claystone and Limestone
		Sleipner Formation	Sandstone-Claystone intercalation, minor Coal

Figure 2.2: Regional stratigraphic sequence of the study area, Volve field. Source: [1]

The Nordland Group (84.0 - 1034.3 m TVD)

It is composed of silty claystone with some sand stringers, down to the Utsira Formation's top. The Utsira Formation (817.5 - 1034.3 m TVD) consists of mainly sand with some claystone stringers.

The Hordaland Group (1034.3 - 2206.8 m TVD)

Claystone is found mostly in the upper part, along with some stringers of limestone. Two sand sequences were encountered, the first being the Skade Formation (1156.3 -1230.3 m TVD). The Grid Formation is the second one (2040.2 - 2169.6 m TVD), consisting of sand. The lower Hordaland Group is composed of claystone below the Grid Formation.

The Rogaland Group (2206.8 - 2527.6 m TVD)

Balder Formation (2206.8 - 2268.0 m TVD) consists primarily of gray to dark gray or red-brown claystone with occasional limestone stringers.

The Sele Formation (2268.0 - 2318.0 m TVD) is composed of dark gray to brown-gray claystone, with thin limestone and dolomitic stringers.

The Lista Formation claystone is medium to dark in color (2318.0-2405.0 m TVD). Dolomitic limestone and limestone stringers occur frequently.

The Shetland Group (2527.6 - 2758.6 m TVD)

The Ekofisk Formation (2527.3-2542.5 m TVD) is characterized by white to occasionally light gray limestone with minor claystone.

There is predominantly white limestone in the Tor Formation (2542.5 - 2669.6 m TVD).

There is no apparent lithological change in the Hod Formation (2669.6 - 2740.4 m TVD).

Blodøks Formation (2740.4 - 2752.9 m TVD) is composed of gray marl. At the base of the formation, the marl grades to calcareous claystone.

The Hydra Formation (2752.9 - 2758.6 m TVD) is lithologically the same as the Blodøks Formation.

The Shetland Group (2527.6 - 2758.6 m TVD)

Rødby Formation (2758.6 - 2767.3 m TVD) is composed of marl and claystone as described for the Blodøks Formation.

A major part of the Åsgard Formation (2777.2 - 2854.4 m TVD) consists of interbedded limestone and marl with some minor claystone.

The Viking Group (2854.4 - 2864.0 m TVD)

The group has two formations: Draupne (2854.4-2858.6 m TVD) and Heather (2858.6-28864.0 m TVD). Between the formations, there are no obvious lithological differences. Draupne and Heather formations are composed of very dark brownish gray claystone.

The Vestland Group (2864.0 - 2882.0 m TVD)

It is represented by a minor part of the Hugin Formation. The formation consists of olive grey sandstone.

The Trlasslc (2882.0 -3110.3 m TVD))

A part of the Skagerrak Formation represents the Triassic interval. It consists primarily of sandstone, interbedded with some silty sections.

2.4 Wells

Well 15/9-F5

15/9-F-05 is planned as a water injection well to support production from the 15/9-F-14 Hugin producer. Sulfate-free water for injection will be produced from the Utsira water-producing wells.

Well 15/9-F14

15/9-F-14 is the second producer in the Volve development drilling program. This production well is located in a structurally high position on the crest of the structure 800 m up the flank of the 15/9-19 (discovery well).

Well 15/9-F15

15/9-F-15 is the third oil producer in the Volve development drilling program. The well will be drilled in the south/southwestern part of the structure. Volve wellhead module slot 15 is used to drill the well.

Well 15/9-F15S

15/9-F15S is drilled as an exploration well on slot no. 15.

Well 15/9-F7

15/9-F7 is planned as a water producer producing low-sulfate water from the Utsira Fm. The well is drilled through the Utsira Fm and completed with screens in the reservoir section.

2.5 Drilling data

The attributes recorded while drilling:

Measured Depth m

Wellbore length, as if measured with a measuring stick. Except in vertical wells, this measurement differs from the true vertical depth.

Hole Depth (TVD) m

A vertical distance is measured from the bottom of a well (the current or ultimate depth) to the surface, often the height of the rotary kelly bushing (RKB).

Weight on Bit, kkgf

A combination of downward force exerted by thick-walled tubular pieces in the drilling assembly, referred to as drill collars, on the drill bit and gravity's downward pressure on these steel tubes provide the force necessary to break rock effectively.

Average Standpipe Pressure, kPa

Pressure drops occur when drilling fluid is circulated due to friction between the fluid and the surface in contact. The drilling fluid is forced to circulate through the hydraulic system through the mud pump. The mud pump pressure is partly used to overcome the friction between the fluid and the hole, casing, and surface equipment. The remaining pump pressure is lost to bit nozzle pressure, where the high nozzle speed assists in removing cuttings from the bit and its surroundings. The standpipe pressure is the total pressure drop due to fluid friction.

Average Surface Torque, kN.m

The surface torque is the moment required to rotate the drill string and the bit on the bottom of the hole. It is used to overcome rotational friction against the wellbore, the viscous force between the drilling fluid and pipe string, as well as bit torque.

Rate of Penetration, m/h

In drilling, a measure of the speed at which a bit drills into formations is usually expressed in feet (meters) per hour or minutes per foot (meter).

Average Rotary Speed, rpm

The rotary speed is measured by how many revolutions the rotary table makes in one minute (rpm). The rotary table on a drilling platform provides clockwise rotation to the drill string in order to facilitate the drilling process.

Mud Flow In L/min

A drilling fluid, also known as mud flow, is circulated within a construction. The drilling fluid is used to transport drill cuttings, lubricate and cool equipment, and apply pressure to the borehole. Mud flow rate is one of the monitored measurements to ensure that drilling is done correctly.

Mud Density In g/cm³

In a wellbore, mud weight controls hydrostatic pressure and prevents unwanted flow. Additionally, the mud prevents casings and openholes from collapsing. As a result of the weight of the mud, fractures in the rock can propagate and be filled, leading to lost circulation. The API has standardized and published mud weight test procedures using a mud balance.

Diameter, mm

Diameter refers to the nominal wellbore diameter. Caliper logs record the diameter of the wellbore measured with spring-loaded caliper arms. Caliper logs are usually run simultaneously with an acoustic log or a neutron log. The average diameter of the wellbore is usually measured and recorded.

Average Hookload, kkgf

The total force pulling the hook down. The total force comprises the weight of the drill string in air, the drill collars, and any ancillary equipment, reduced by any force that tends to reduce that weight. Friction along the wellbore wall (especially in deviated wells) and buoyant forces caused by the drill string immersed in drilling fluid may reduce the weight.

USROP Gamma, gAPI

Gamma rays are naturally emitted by a formation, which can be measured in an inexpensive and common way. A gamma ray log is particularly useful since sandstones and shales typically have different gamma ray signatures that can be correlated. The Gamma Ray sensor is located

right before the drill bit, so the data is not significantly delayed even though it is referred to as the logging-while-drilling (LWD) measurements. As there was no unified gamma reading for all of the wells in the original Volve dataset, a new attribute, USROP Gamma, was introduced by Andrej Tunkiel (Tunkiel et al., 2020) [14]. Data was recorded under different names and with different equipment, sometimes even within the same well.

Chapter 3

Methodology

3.1 Supervised Machine Learning Classification

The supervised learning process involves training the machines with well-labeled training data and then predicting the output using the loss function, adjusting until the error is minimized. Tagged data is input data that is already associated with the correct output.

During supervised learning, the algorithm learns how to correctly predict the outcome based on the training data provided. The concept is the same as a student learning under the supervision of a teacher.

There are two types of supervised learning problems - regression and classification:

- The objective of regression is to determine the connection between dependent and independent variables. It is utilized to predict continuous variables.
- When the output variable is categorical, classification methods are utilized. Based on the observed data, algorithms attempt to generate some conclusions on how to label or define these entities.

3.1.1 K-Nearest Neighbor

A K-Nearest Neighbor (K-NN) classifier is used for supervised ML, and it should not be confused with a K-mean classifier used for unsupervised learning. Cover and Hart introduced the

method in 1967 /citeCover1967, which is widely used in various ML applications. Classification is done based on K-Nearest Neighbors. Training examples need to be in memory at run-time, so this method is known as Memory-Based Classification.

Figure 3.1 shows the 3-Nearest Neighbour Classifier applied to a two-class problem in a two-dimensional feature space. Point q_1 is classified as class 0 based on all three of its nearest neighbors being of class 0. There are two neighbors in class 2 and one in class 0 for point q_2 , so the situation is a bit more complicated. It can be resolved by simple majority voting or by distance weighted voting, where greater weight is assigned to the closest neighbors in determining the class of the point (Cunnigham et al., 2007) [2].

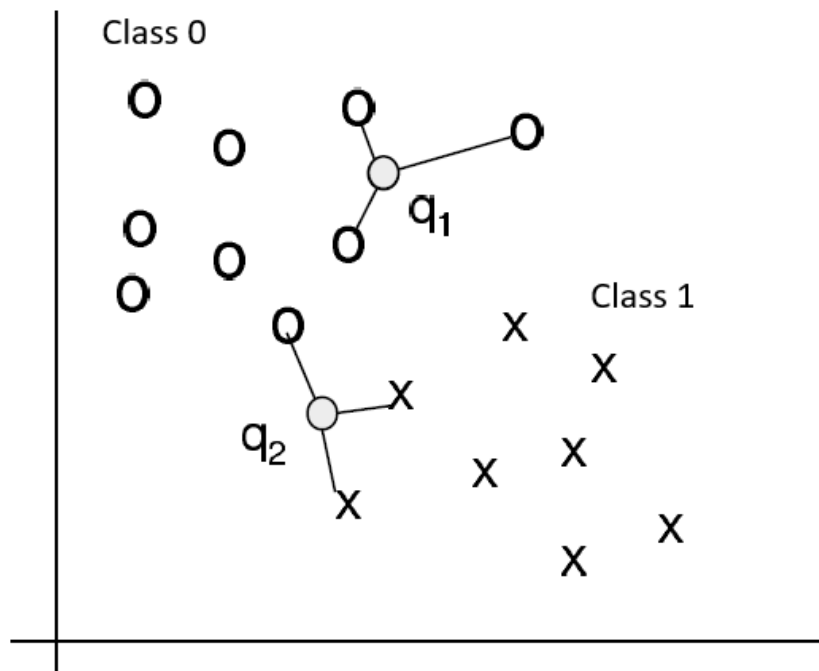


Figure 3.1: A simple example of 3-Nearest Neighbour Classification. Source: [2]

The k-NN classification consists of two steps: the first step is to determine the nearest neighbors, and the second is to determine the class based on those neighbors.

A more mathematical approach to the problem can be found by studying the observation point x_0 and its neighbors. The K-neighboring points to x_0 are defined as N_0 in the K-NN classifier. As a result of the classifier's calculations, on the formula below, the probability for class j is calculated as a fraction of the points in N_0 whose response values equal j .

$$\Pr(Y = j | X = x_0) = \frac{1}{K} \sum_{i \in N_0} I(y_i = j) \quad (3.1)$$

By using the Bayes rule to identify the neighbor with the highest probability, x_0 is then given a new class (Hastie et al., 2014) [15].

3.1.2 Decision Trees

The root node of a decision tree (DT) is comprised of all the instances given in a dataset. In the attribute node, the instances are divided into several subsets starting from the first instance. A decision tree may contain an attribute more than once but not in the same path. A path will eventually lead to a leaf node.

Typically, attribute selection criteria include a measure of the purity of a node, that is, the degree to which the node consists only of a single class examples. In terms of impurity measures, the information-theoretic entropy and the Gini index are defined as:

$$GiniIndex = 1 - \sum_j p_j^2 \quad (3.2)$$

$$Entropy = - \sum_j p_j \cdot \log_2 p_j \quad (3.3)$$

Where p_j is the probability of class j .

Gini index represents the likelihood that any element in a dataset will be mislabeled when it is randomly labeled.

The Gini index has a minimum value of zero. It occurs when a node is pure, meaning all the elements contained are of one unique class. This node cannot be split again. Therefore, the features with the lowest Gini Index determine the best split.

Entropy measures the degree of disorder in the relationship between the features and the target. As with the Gini Index, the feature with less entropy is chosen as the optimum split. When the probabilities of the two classes are equal, a node is pure, and its entropy is minimum.

As much as possible, a good attribute should divide the dataset into subsets that are as pure as possible, ideally into sets that contain only examples from the same class. A desirable

attribute, the so-called gain, would be the one that results in the most significant decrease in average impurity:

$$\text{Gain}(S, A) = \text{Impurity}(S) - \sum_t \frac{|S_t|}{|S|} \cdot \text{Impurity}(S_t) \quad (3.4)$$

where t is a test for attribute A which partitions the set S into non-overlapping disjoint subsets S_t , and Impurity is any measure of impurity. Considering that the first term, $\text{Impurity}(S)$, is constant for all attributes, it is possible to omit it and directly minimize the average impurity. This is typically done when Gini is used to measure impurity.

For large datasets, the decision tree can become very complex. As the amount of data is increased, more subsets will be generated, causing the variations between the outcomes to become smaller. Additionally, excessively complex decision trees can lead to overfitting, which is a common consequence of the algorithm being trained on too much data (Fürnkranz et al., 2010) [16].

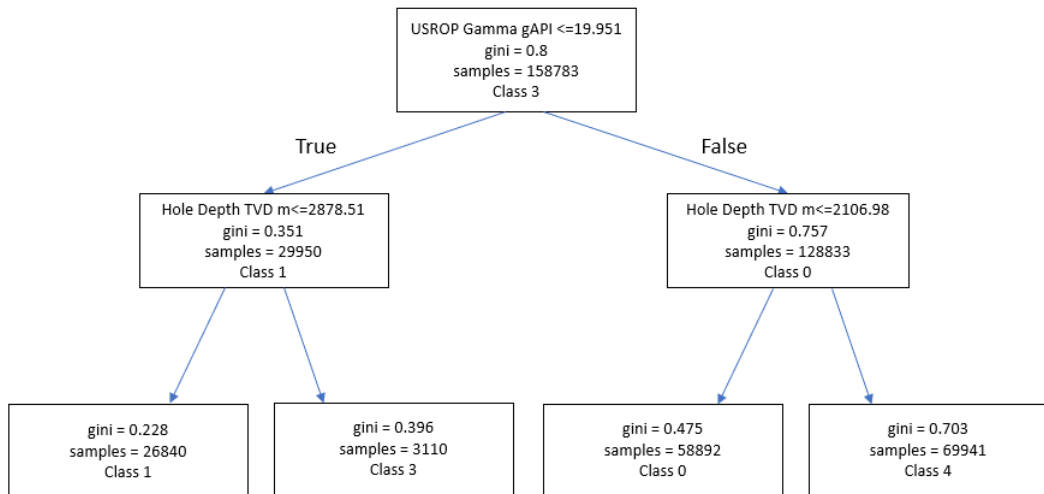


Figure 3.2: Result of the Decision Tree for Classification

3.1.3 Logistic Regression

The most common application of logistic regression is binary classification, which enables observations to be assigned to discrete classes. Logistic Regression is basically a Linear Regression algorithm, but it utilizes a more complex cost function referred to as the Sigmoid Function,

which is illustrated in the Figure 3.3.

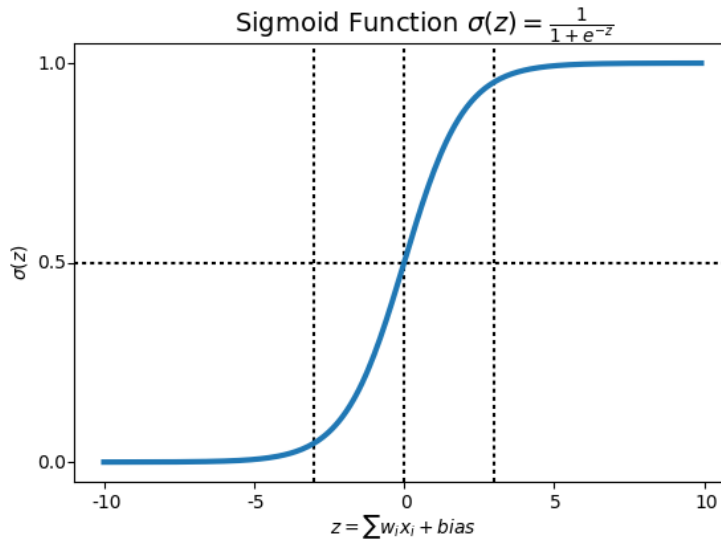


Figure 3.3: Sigmoid Function

In logistic regression, the cost function tends to be in the range of 0 and 1.

$$h\theta(X) = \frac{1}{1 + e^{-(\beta_0 + \beta_1 X)}} \quad (3.5)$$

X is a vector of data points, and $h(x)$ represents the probability $P(x)$ belongs to a particular class. β_1 is the slope, and β_0 is the y-intercept of the Sigmoid Function.

3.1.4 Ensemble Learning

An ensemble is a combination of learning machines that improve the overall system's performance. Over the past decade, one of the leading research areas in ML has been the development of ensembles of learning machines.

The empirical evidence shows that ensembles of classification or regression problem learners are often more accurate than the individual base learners that make them up, and several theoretical explanations have recently been proposed in order to justify the effectiveness of some of the commonly used ensemble methods (Valentini et al., 2002) [17].

Random Forests

Random forests are a type of classifier that combines decision tree-based classifiers $\{h(x, \theta_k), k = 1, \dots\}$ where the θ_k are independent identically distributed random vectors, and each tree decides the most popular class based on input x (Bernard, 2014) [18].

Bagging, also known as Bootstrap Aggregation, is a technique used by the random forest algorithm. It involves choosing a random sample from the data set. Thus, each model is generated from the samples in the dataset using row sampling. The method of row sampling with replacement is called bootstrapping. With each model trained separately, results can be generated. Combining the results of all models, the final output is based on majority voting. This process is known as aggregation.

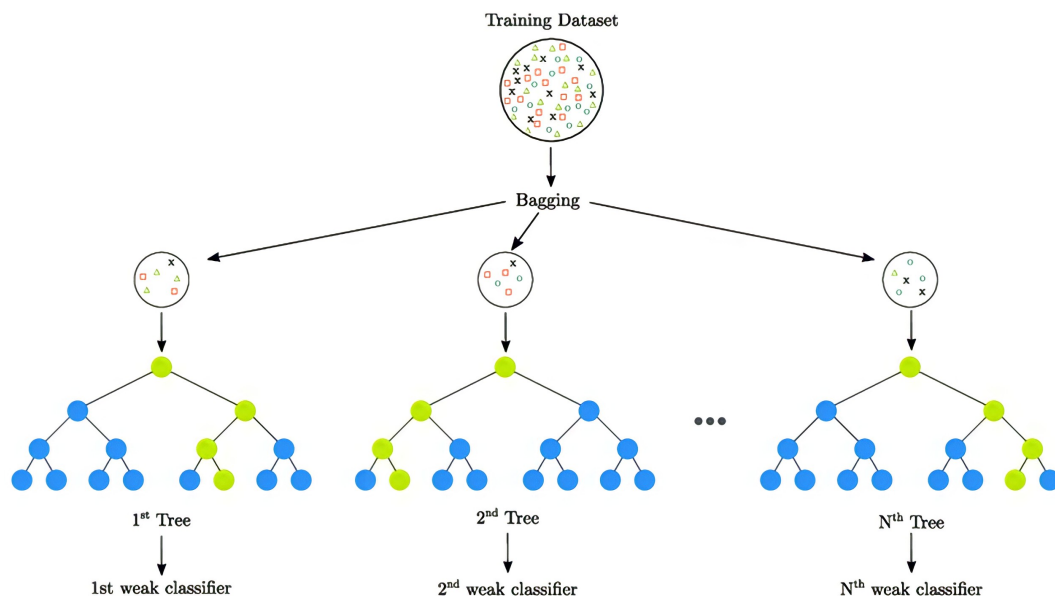


Figure 3.4: Example of a random forest. Source: [3]

Key Benefits

A reduced risk of overfitting: Decision trees tend to tightly fit all samples in training data, which increases the chance of overfitting. When there are a large number of decision trees in a random forest, the classifier will not overfit the model since the averaging of uncorrelated trees will reduce the variance and prediction error.

Feature importance: Random forest provides a simple way to assess the influence variables have on the model. There are a few ways to evaluate the importance of features. Gini importance and Mean Decrease in Impurity (MDI) are usually used to assess how much the model's

accuracy decreases when certain variables are excluded. Another critical metric is Permutation importance, or Mean Decrease Accuracy (MDA), which identifies the average decrease in accuracy by randomly permuting the feature values.

Key Challenges

Time-consuming process: Random forest algorithms are one of the most accurate prediction methods. However, they take a long time to process since they compute data for each decision tree separately.

Demanding more resources: As random forests are employed to evaluate larger data sets, more storage capacity will be required.

Adaptive Boosting

Most ensemble techniques rely on simple averages of models in the ensemble. Boosting methods use different, constructive strategies for forming ensembles. The idea behind adaptive boosting is to add new models to the ensemble sequentially. In each iteration, a weak, base-learner model is learned based on the error of the whole ensemble.

- AdaBoost's weak learners are decision trees with one split, called decision stumps.
- AdaBoost prioritizes difficult-to-classify instances over those that are already effectively handled.

AdaBoost was the first practical boosting algorithm developed by Freund and Schapire [19], and it remains one of the most studied and widely used algorithms, with applications in many different fields.

AdaBoost algorithm steps [19]:

Given:

$$(x_1, y_1), \dots, (x_m, y_m)$$

where $x_i \in X, y_i \in \{-1, +1\}$.

Initialize:

$$D_1(i) = 1/m \text{ for } i = 1, \dots, m. \tag{3.6}$$

For $t = 1, \dots, T$:

- Train weak learner using distribution D_t .
- Get weak hypothesis $h_t : X \rightarrow \{-1, +1\}$.
- Aim: select h_t with low weighted error:

$$\varepsilon_t = \Pr_{i \sim D_t} [h_t(x_i) \neq y_i] \quad (3.7)$$

- Choose:

$$\alpha_t = \frac{1}{2} \ln \left(\frac{1 - \varepsilon_t}{\varepsilon_t} \right). \quad (3.8)$$

- Update, for $i = 1, \dots, m$:

$$D_{t+1}(i) = \frac{D_t(i) \exp(-\alpha_t y_i h_t(x_i))}{Z_t} \quad (3.9)$$

where Z_t is a normalization factor (chosen so that D_{t+1} will be a distribution). Output the final hypothesis:

$$H(x) = \text{sign} \left(\sum_{t=1}^T \alpha_t h_t(x) \right) \quad (3.10)$$

$$h\theta(X) = \frac{1}{1 + e^{-(\beta_0 + \beta_1 X)}} \quad (3.11)$$

3.1.5 Gradient Boosting

A gradient boosting machine, or GBM, successfully fits new models to provide a more accurate estimate of the response variable. With this algorithm, the new base-learners are constructed to be maximally correlated with the negative gradient of the loss function associated with the entire ensemble. If the loss function is the classic squared-error loss, the learning procedure will result in sequential error-fitting. Gradient boosting is a method that excels at predicting large, complex datasets with speed and accuracy (Natekin and Knoll, 2013) [20].

Gradient Boosting Algorithm steps [21]:

1. Initialize model with a constant value:

$$F_0(x) = \operatorname{argmin}_{\gamma} \sum_{i=1}^n L(y_i, \gamma) \quad (3.12)$$

2. for $m = 1$ to M :

- Compute residuals:

$$r_{im} = - \left[\frac{\partial L(y_i, F(x_i))}{\partial F(x_i)} \right]_{F(x)=F_{m-1}(x)} \quad (3.13)$$

for $i = 1, \dots, n$

- Train regression tree with features x against r and create terminal node regions R_{jm} for $j = 1, \dots, J_m$

- Compute:

$$\gamma_{jm} = \operatorname{argmin}_{\gamma} \sum_{x_i \in R_{jm}} L(y_i, F_{m-1}(x_i) + \gamma) \quad (3.14)$$

for $j = 1, \dots, J_m$

- Update the model:

$$F_m(x) = F_{m-1}(x) + v \sum_{j=1}^{J_m} \gamma_{jm} 1(x \in R_{jm}) \quad (3.15)$$

AdaBoost requires users to specify a set of weak learners (alternatively, it will randomly generate weak learners before the actual learning process begins). The weights of the incorrectly predicted instances are increased, while those of the correctly predicted ones are decreased. Weak learners thus focus on difficult instances. Weak learners are added to strong learners based on the strength of their performance (alpha weight). The greater its performance, the more it contributes to a strong learner.

In contrast, gradient boosting does not change the sample distribution. The weak learner trains on the residual errors of the strong learner instead of on a newly sampled distribution.

This is another way to give more weight to difficult instances. A weak learner is fitted to the pseudo-residuals at the end of each iteration. After that, the contribution of the weak learner to the strong learner is not computed based on its performance on a new distributed sample but on a gradient descent optimization process. The contribution computed by the strong learner is the one that minimizes the overall error.

3.1.6 Naïve Bayes

The Naïve Bayes algorithm uses the Bayes rule in conjunction with the assumption that the attributes are conditionally independent, given the class. Despite often violating this independence assumption in practice, Naïve Bayes offers competitive classification accuracy.

Bayes Rule:

$$P(y | \mathbf{x}) = \frac{P(y)P(\mathbf{x} | y)}{P(\mathbf{x})}, \quad (3.16)$$

where y is a class, and x represents the feature vector. Based on the class, it is assumed that the features are conditionally independent.

For feature-value data, this assumption entitles:

$$P(\mathbf{x} | y) = \prod_{i=1}^n P(x_i | y), \quad (3.17)$$

x_i is the value of the i th feature in \mathbf{x} , and n is the number of them.

$$P(\mathbf{x}) = \prod_{i=1}^k P(c_i) P(\mathbf{x} | c_i), \quad (3.18)$$

k is the number of classes, and c_i is the i th class. Hence, Bayes Rule can be calculated by normalizing the numerators on the right-hand side of the equation.

Numeric attributes are either discretized or probability density estimates are employed (Webb, 2011) [22].

3.2 Classification on Imbalanced Data

The imbalanced classification problem occurs when there is an imbalance in the distribution of examples across the known classes. There can be a slight bias or a severe imbalance where there is one example in a minority class for multiple examples in a majority class or classes.

As most ML algorithms used for classification assume an equal number of examples for each class, imbalanced classification presents a challenge for predictive modeling. This causes models to have poor predictive performance, especially for the minority class. Usually, the minority class is more important than the majority, so the problem is more sensitive to classification errors for the minority class.

Several solutions to the class-imbalance problem have been proposed at the data and algorithmic levels. These methods include oversampling with replacement, random undersampling, directed oversampling (in which no new samples are created, but the samples to replace are informed rather than random), directed undersampling (in which the examples to eliminate are informed), oversampling with the informed generation of new samples, and combinations of these methods. At the algorithmic level, we can adjust the costs of various classes to counterbalance class imbalance, adjust the probabilistic estimate at the leaf of the tree (when using decision trees) and adjust the decision threshold (Kotsiantis et al., 2005) [23].

3.2.1 Oversampling and Undersampling

Sampling is one of the major methods for addressing the problem of imbalanced learning. Different procedures are used to modify a set of imbalanced data to provide a more balanced or adequate data distribution for the learning tasks.

There are two types of sampling approaches: undersampling, which reduces the data by eliminating examples belonging to the majority class, to equalize the number of examples of each group; and oversampling, which aims to generate new positive examples to gain importance (Fernández et al, 2018) [4].

Oversampling Techniques

Random oversampling

Oversampling involves randomly replicating data from minority classes and adding them to the training dataset.

In the training dataset, samples are selected randomly with replacement. This allows examples from the minority class to be added more than once to the new "balanced" training dataset since they are selected from the original dataset, added to the new dataset, and then returned to the original dataset so they can be selected again.

Using this technique can be effective for ML algorithms affected by skewed distributions and where multiple duplicates for a given class can impact the model's fit.

Overfitting is more likely to occur due to random oversampling, which copies the exact minority class examples. Thus, a symbolic classifier, for example, may construct rules that appear to be accurate but are based on one example (Fernández et al., 2018) [4].

Synthetic Minority Oversampling Technique (SMOTE)

In order to rebalance the original training set, the SMOTE algorithm employs oversampling. Rather than merely replicating minority class instances, SMOTE introduces synthetic examples. These new examples are created by interpolating between several positive instances that lie together. The procedure is thus said to be centered around the "feature space" and not the "data space".

A simple example of this oversampling process is illustrated in Figure 3.5. A new synthetic data point is created by selecting an X_i instance as a basis. From the training set, a distance metric is used to select several samples of the same class (points x_{i1} to x_{i4}). Lastly, a randomized interpolation is applied to obtain new instances r_1 to r_4 (Fernández et al., 2018) [4].

Adaptive Synthetic (ADASYN)

ADASYN generates minority examples adaptively, based on the distribution of minority classes: more synthetic data is generated for minority classes that are more difficult to learn in comparison with minority classes that are easier to learn. In addition to reducing the learning

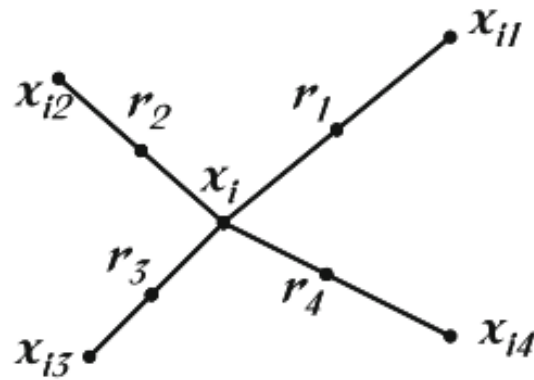


Figure 3.5: An illustration of synthetic data points in the SMOTE algorithm. Source: [4]

bias introduced by the original unbalanced distribution of data, the ADASYN method is also able to adapt the decision boundary to focus on the hard-to-learn samples (Fernández et al., 2018) [4].

Undersampling Techniques

Random undersampling

During random undersampling, examples from the majority class are randomly selected to be deleted from the training set.

In the transformed version of the training dataset, fewer examples appear in the majority class. This process can be repeated until the desired class distribution is achieved, such as an equal number of examples per class.

It may be more appropriate to use this approach for datasets with an imbalance between classes, but there are enough examples in the minority class.

A major disadvantage of random undersampling is that this method can omit potentially valuable data that could be vital to the learning process. Undersampling is a critical decision, and many heuristics are used to overcome the limitations of nonheuristics when deciding how to remove data (Fernández et al., 2018) [4].

NearMiss

The family of four methods uses informed heuristics. In the first "NearMiss-1" method, we select samples from the majority class similar to some of the minority class samples. This

method selects samples of the majority class when the average distance between them and the three samples of the closest minority class is the smallest.

In the second “NearMiss-2” method, the majority samples are selected when their average distance to the three largest minority samples is the smallest.

A third method, called "NearMiss-3" determines how many majority class samples are closest to each minority class sample (Fernández et al., 2018) [4].

Tomek's Links

Tomek's Links can be defined as follows: given two examples $E_i = (x_i, y_i)$ and $E_j = (x_j, y_j)$ where $y_i \neq y_j$ and $d(E_i, E_j)$ being the distance between E_i and E_j . A pair (E_i, E_j) is called Tomek's links if there is not an example E_l , such that $d(E_i, E_l) < d(E_i, E_j)$ or $d(E_j, E_l) < d(E_i, E_j)$.

Tomek's links can be used as an undersampling method by removing only examples belonging to the majority class in each Tomek link (Fernández et al, 2018) [4].

3.2.2 Class weights

To modify the current training algorithm to take into account the skewed distribution of the classes, different weights are given to both the majority and minority classes. During the training phase, the weight difference will impact classification. To penalize the minority class for misclassification, the weights of each class are set higher while weights for the majority class are reduced.

The weight of the minority class is higher than that of the majority class. ML models pay more attention to observations with a higher weight as they are being learned.

The weights of the classes are automatically assigned inversely proportional to their frequencies when the *class_weights = 'balanced'* option is specified as the hyperparameter in the classification algorithm in the Scikit-learn library.

Within the framework, the balanced class weights are calculated:

$$w_j = \frac{n_{\text{samples}}}{n_{\text{classes}} * n_{\text{samples}_j}} \quad (3.19)$$

- w_j is the weight assigned to each class (where j is the class)
- n_{samples} is the total number of samples or rows in the dataset
- n_{classes} is the total number of distinct classes in the target
- n_{samples_j} is the total number of rows of the respective class

3.3 Model Analysis

3.3.1 Model Evaluation Metrics

Confusion Matrix

In classification problems, the confusion matrix is a prevalent measure. It can be applied to both binary and multiclass classification. The confusion matrix, displayed in the Figure 3.6 represents counts from predicted and actual values.

		True Class	
		Positive	Negative
Predicted Class	Positive	TP	FP
	Negative	FN	TN

Figure 3.6: Confusion matrix for binary classification

For the binary classification, True Negatives (TN), False Positives (FP), False Negatives (FN), and True Positives (TP) are intuitively clear, where negative classes are referred to as class X and positive ones as class Y.

- The true negatives (TN): class X which is correctly predicted as class X
- The false positives (FP): class X which is wrongly predicted as class Y
- The false negatives (FN): class Y which is wrongly predicted as class X
- The true positives (TP): class Y which is correctly predicted as class Y

Positives and negatives cannot be directly defined in the multiclassification problem. The classes that were predicted correctly/wrongly in relation to the rest:

- The true negatives (TN): X's which is correctly predicted as X's
- The false positives (FP): X's which are wrongly predicted as non-X's
- The false negatives (FN): non-X's which are wrongly predicted as non-X's
- The true positives (TP): non-X's which are correctly predicted as non-X's

Therefore, True Positive and False Positive rates are computed individually for each class. True positive rate (TPR) gives the proportion of correct predictions in predictions of positive class:

$$TPR = \frac{TP}{TP + FN} \quad (3.20)$$

False positive rate (FPR) gives the proportion of incorrect predictions in positive class:

$$FPR = \frac{FP}{FP + TN} \quad (3.21)$$

Accuracy

$$Accuracy = \frac{TP + TN}{TP + TN + FP + FN} \quad (3.22)$$

Accuracy is a metric used to describe how the model performs across all classes. It works only if the number of samples in each class is equal.

For example, if 10 samples out of a dataset with 100 data points belong to class Limestone and if the model predicts every data as non-Limestone, it has the accuracy of 90%. However,

in this case, the model did nothing to indicate the right class. The accuracy of classifications typically does not allow for differentiating errors in predictions.

Precision

$$Precision = \frac{TP}{TP + FP} \quad (3.23)$$

Provides a measure of the class prediction correctness. This evaluation is based on positive predictions. For example, if precision equals 0.25, then the model will correctly predict a class 25% of all time.

Recall

$$Recall = \frac{TP}{TP + FN} \quad (3.24)$$

Recall measures of how accurately the model can define the relevant class. It helps to measure how many ML models correctly classified positive samples.

While evaluating the ML model, there is a trade-off between precision and recall metrics. In lithology classification, it is critical that each class is classified correctly rather than the number of correctly predicted data points related to the class. Therefore, one should aim for higher precision, not recall.

F1-score

$$F_1 - score = 2 \times \frac{Precision \times Recall}{Precision + Recall} = \frac{2TP}{2TP + FP + FN} \quad (3.25)$$

F1-score combines precision and recall into a single measure. Mathematically it's the harmonic mean of precision and recall.

3.3.2 ROC curve

Receiver operating characteristic (ROC) curves plot the true positive rate against the false positive rate at all possible thresholds and thus represent their trade-off as shown in the Figure 3.7a. The decision threshold δ is the borderline between predicting one or another class. In the multi-classification problem, it predicts all instances with $p > \delta$ as X and all others as $non - X$. The

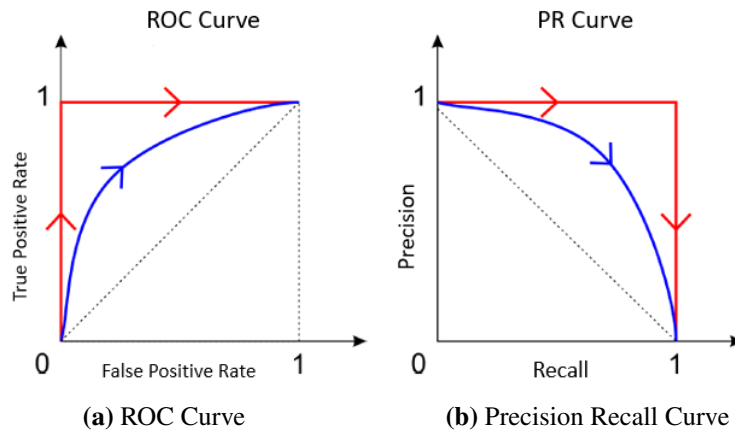


Figure 3.7: Examples of a PR curve and a ROC curve. The red plot shows ideal PR and ROC curves. The blue plot shows some PR and ROC curves, as they typically arise in experiments. Arrows indicate the increase of the threshold level value. Source: [5]

true positive rate increases with a lower threshold δ , but the false positive rate also tends to rise. A ROC curve should be as close as possible to the top-left corner, and a diagonal line represents random guessing. In addition, the area under the ROC curve (AUC or AUROC) summarizes this curve in one single number. The larger the AUC, the better. With an AUC value of 0.7, the classifier is expected to correctly rank two randomly chosen test points 70% of the time.

3.3.3 Precision-Recall (PR) curve

Precision-recall (PR) curves plot precision vs. recall (=true positive rates) for all possible decision thresholds (3.7b). An overall goal is to have both a high recall and a high precision. Precision and recall are linked in an inverse relationship: the lower the threshold, the better the recall, but the worse the precision. This curve can be summarized in one number through the area under the precision-recall curve (AUPRC or average precision). The higher the AUPRC, the better. In contrast to the AUC, the AUPRC does not have an intuitive interpretation.

3.3.4 Dataset Shift

Shifting datasets occurs predominantly in supervised ML and hybrid models of semi-supervised learning.

Dataset shift can be caused by several factors, including the type of features utilized, how training and test sets are selected, data sparsity, and shifts in the data distribution caused by

non-stationary environments. Of all the notations of dataset shift, the simplest to understand is covariate shift.

Covariate shift appears only in $X \rightarrow Y$ problems, i.e probabilistic classification including conditional probabilities, where $P_{tra}(y|x) = P_{tst}(y|x)$ and $P_{tra}(x) \neq P_{tst}(x)$. It occurs in supervised learning when the training and prediction distributions differ from each other, but the concept being learned remains stationary (Herrera, 2011) [6].

Causes of Dataset Shift

- *Sample selection bias.* The distribution discrepancy is because the training examples have been obtained through a biased method and thus do not represent the operating environment where the classifier is to be deployed reliably.
- *Non-stationary stationary environments.* It appears when the training environment is different from the test one, whether it is due to a temporal or a spatial change.

Covariate Shift

Training and test input follow different distributions, but functional relation remains unchanged.

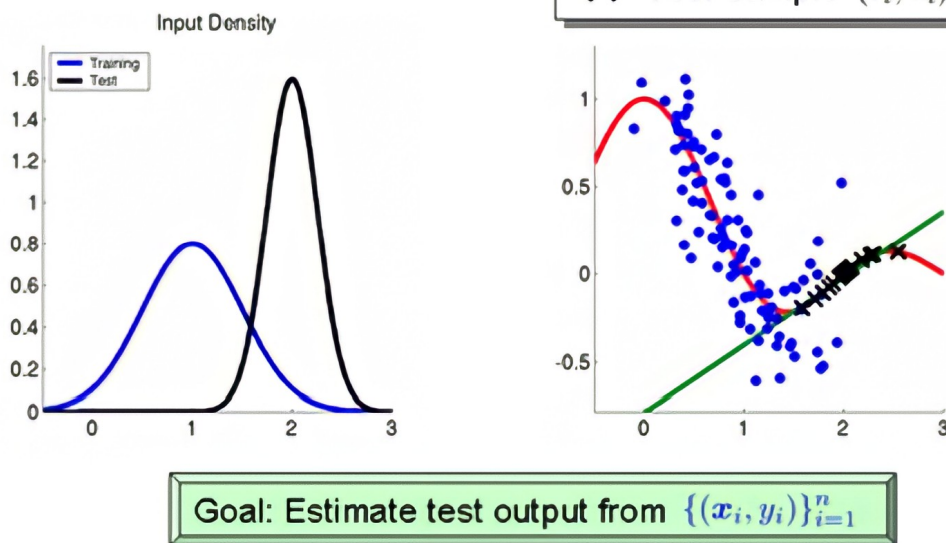


Figure 3.8: Illustrative explanation of Covariate Shift. Source: [6]

Handling Dataset Shift

If possible, the model should be retrained. There may be situations, however, where this is impossible, for example, if retraining is delayed. In such cases, several techniques are available for correcting dataset shift.

1. **Feature Removal** There is a trade-off between removing features that contribute to the covariate shift and adding features and tolerating some covariate shift. When a feature is very different between training and testing and doesn't provide much predictive power, it should always be dropped.
2. **Importance Reweighting** Refers to upweighting training instances that are very similar to test instances. This is accomplished by altering the training data set so that it appears that it is drawn from the test data set. The relative probability of the training and test sets should be reweighted for each of the training examples. Various methods such as density estimation, kernel mean matching, or discriminative reweighting can be used.
3. **Adversarial Search** In the adversarial search method, the ML models are tricked by the learning algorithm by providing false input. In this way, it includes both generating and detecting adversarial examples, which are inputs that are intended to deceive classifiers. This problem is defined as finding the optimal minimax strategy for an adversary that deletes features, and bundle optimization is shown to be effective in finding the optimal strategy.

3.3.5 Model Interpretability

The model's interpretability can be divided into two categories: global interpretability, which explains the model's behavior over the entire population, and local interpretability, which explains a specific prediction. Feature sensitivity relates to the global model interpretability of black-box models. LIME explanation provides local model interpretability. It modifies a single data sample by tweaking the feature values and observing the output's resulting impact. LIME offers a list of explanations reflecting the contribution of each feature to the prediction of a data sample.

Feature Sensitivity

The easiest and most effective way to understand the ML model is through sensitivity analysis, which examines how each feature impacts the model's predictions. As a result of changing the feature value or ignoring it somehow while keeping all the other features constant, one calculates feature sensitivity. Changes in feature value drastically alter the model's outcome, so this feature significantly impacts the prediction.

Feature Sensitivity report provided by Pytolemaic toolbox [24] gives the insight about:

- Feature Importance score is calculated for each input feature in a given model - the score represents the "importance" of each feature. A higher score indicates a greater impact of a particular feature on predicting a certain class.
- Imputation vulnerability provides insight into how the model deals with missing values. An increase in these values indicates that the imputation mechanism is prone to mistakes, so the model should be improved. Imputation techniques are used to fill in the missing data to create a complete data matrix.
- Data Leakage - Term used in ML to identify data that contains unexpected extra information about the subject being estimated. Learning leaks occur when information about a target label or number is introduced but not lawfully available during training. Such cases, however rare, cause models to have an excellent score but be entirely useless.
- Data Leakage refers to the error when information about the target variable from the testing dataset is leaking into the model's input during the model's training. The model will perform well on the specific training set but poorly in production.
- Analysis on too many features indicates the risk of the model overfitting and wasted effort in the data engineering phase.

many features mean two things: a) higher risk of overfitting and b) wasted effort in the data engineering phase. If the sensitivity to too many features is high, feature selection or regularization should be considered. The regularization process is used to optimize ML models, minimizing the adjusted loss function and preventing overfitting or underfitting.

Lime Explanation

Local Interpretable Model-Agnostic Explanations (LIME) is an explanation technique that explains classifier predictions in a faithful manner by learning an interpretable model locally around the predictions (Ribeiro et al., 2016) [7].

LIME Algorithm [25]:

- By sampling X values from a Normal distribution inferred from the training set, generate points all around the \mathbb{R}^p space
- Utilize the ML model to predict the Y coordinate of each sampled point
- Assign weights based on the distance from the chosen point
- Use the generated weighted dataset to train the linear ridge regression: $E(Y) = \beta_0 + \sum \beta_j X_j$. The β coefficients are regarded as the LIME explanation.

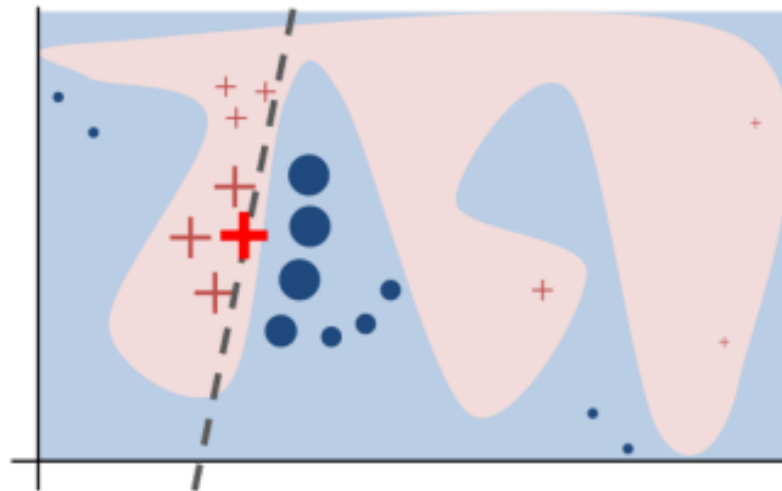


Figure 3.9: Toy example to present intuition for LIME. Source: [7]

A black-box model's complex decision function f (unknown to LIME) is illustrated by the blue/pink background in Figure 3.9, which a linear model cannot adequately approximate. The bold red cross represents the instance being explained. With LIME, instances are sampled, f function predictions are derived, and they are weighted based on their proximity to the instance being explained (represented by the size of the crosses and circles). The dashed line is the learned explanation that is locally faithful but not globally (Ribeiro et al., 2016) [7].

3.4 Uncertainty Quantification

ML has become a preferred methodology for data analysis and prediction in the modern world due to the availability of data and computational technologies. Despite ML's great promise, the results from such models are not entirely reliable due to the challenges presented by uncertainty. The ML model generates the optimal solution based on the training data. However, if the uncertainty in the data and the model parameters are not taken into account, such optimal solutions may fail in the real world (Ghosh et al., 2021) [8].

Uncertainty Quantification 360 (UQ360) is an open-source toolkit with a Python package developed by IBM for data science practitioners and researchers to calculate, evaluate, improve, and communicate the uncertainty of ML models as common practice for AI transparency.

3.4.1 Intrinsic and Extrinsic UQ Algorithms

There are two types of uncertainty quantification algorithms, intrinsic and extrinsic, depending on how the uncertainties are derived from the AI models. The following diagram shows the taxonomy of the UQ algorithms. Most of them are included in the UQ360 toolkit.

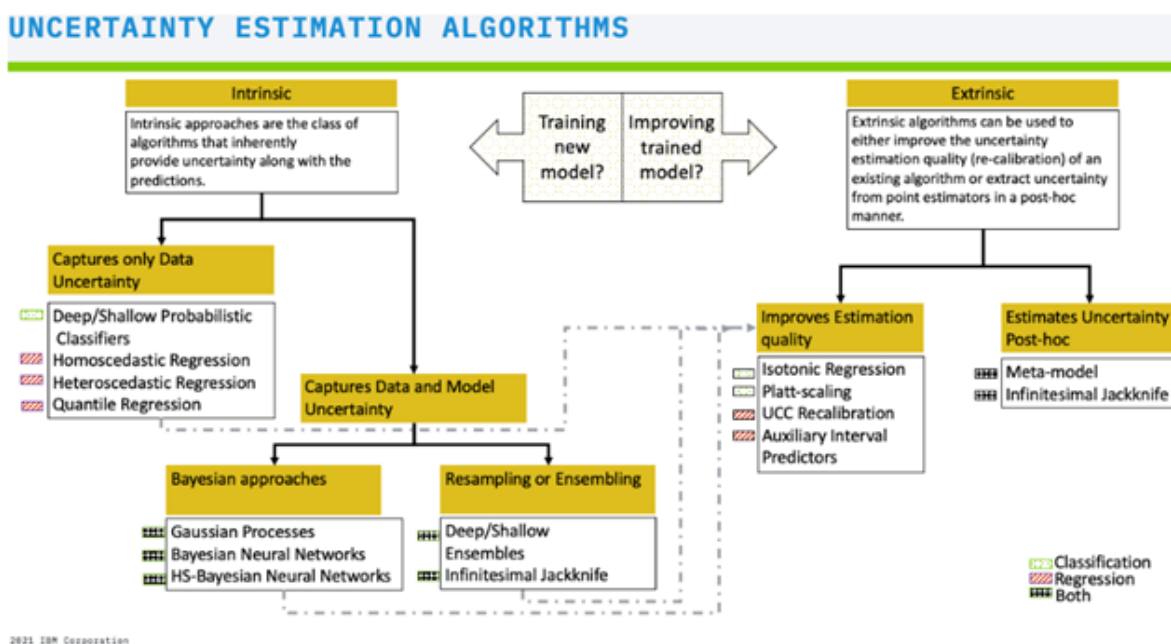


Figure 3.10: Taxonomy of the uncertainty estimation algorithms included in the UQ360 toolkit. Source: [8]

Intrinsic UQ Algorithm

Algorithms produce uncertainty estimates along with predictions. Bayesian and ensemble/re-sampling approaches are employed in the UQ360 toolkit to train models that capture data and model uncertainty. A Bayesian approach has a general computational cost advantage but is based on a robust theory. In this study, the Bayesian Neural Network algorithm is tested on the predicted class probabilities score obtained from the trained model.

Extrinsic UQ Algorithm

Algorithms to calculate the post-hoc uncertainty of trained models. Although neural networks can capture data and model uncertainties, commonly used supervised and unsupervised algorithms cannot. Therefore, methods such as Infinitesimal Jackknife or Meta-Models could be applied.

3.4.2 Calibration

Before elaborating on the topic of calibration, let us define the notations of probability and conditional probability in notations of ML:

- Probability or Confidence refers to how likely is the event to occur. In the case of ML, class probabilities are any real number between 0 and 1. The model objective is to match predicted probabilities with class labels, i.e. to maximize the likelihood by observing class labels given the predicted probabilities.

$$\mathbb{P}(\vec{x}, \vec{y}) = \prod_{i=1}^N \hat{p}(x_i) \cdot y_i \quad (3.26)$$

Likelihood for class labels y and predicted probabilities based on features x .

- Conditional probability is the likelihood of one condition being true if another state is known to be true. In ML notation, the conditional probability distribution of Y given X is the probability distribution of Y if X is known to be a particular feature of another parameter.

$$P(Y_i | X_i) = \frac{P(Y_i \cap X_i)}{P(X_i)} \quad (3.27)$$

Conditional probabilities for class labels y based on their known predicted probabilities on features x .

In classification tasks, it is often essential to estimate the probability that an observation belongs to a particular class - the conditional class probability.

Calibrating a model involves applying a post-processing operation to an already trained model to improve its probability estimation. This is required when the likelihood of a prediction is more important than classification results.

A well-calibrated model should have data points related to a class in the testing set at a frequency of 60% if it correctly predicts that class using 60% probability. Accordingly, "relative frequency" can also be referred to as "conditional probability", which is the probability of a specific class prediction, i.e., a positive outcome conditioned upon a predicted likelihood of 60%.

A calibration model is perfect if a class prediction with confidence p is correct $100 * p$ percent of the time for any probability value p .

Brier Score

Brier Score is in the nature of a cost function. A low value indicates accurate predictions and vice versa. The objective of dealing with this concept is to decrease it. According to the type of predicted variable, the Brier Score for class labels y and predicted probabilities based on features x is formed as follows:

$$\text{Brier Score}(\vec{x}, \vec{y}) = \frac{1}{N} \sum_{i=1}^N (\hat{p}(x_i) - y_i)^2 \quad (3.28)$$

Expected Calibration Error:

An indicator of the calibration of uncertainty produced by a classifier. This is defined as the difference between the classifier's accuracy and confidence.

Confidence histogram and Reliability plot:

The confidence histogram shows each bin's sampling fraction and the class prediction's

confidence. In the reliability diagram, the X-axis shows the average confidence for each bin, and the Y-axis shows the prediction accuracy for each bin. The accuracy and confidence should be equal. The confidence score could be interpreted as a probability if the model is calibrated.

The diagonal represents the ideal level of accuracy for each confidence level. If the reliability curve is below the diagonal, the model has too much confidence in its predictions. Whenever the reliability curve crosses the diagonal, the accuracy is greater than the confidence, and the model is not confident enough.

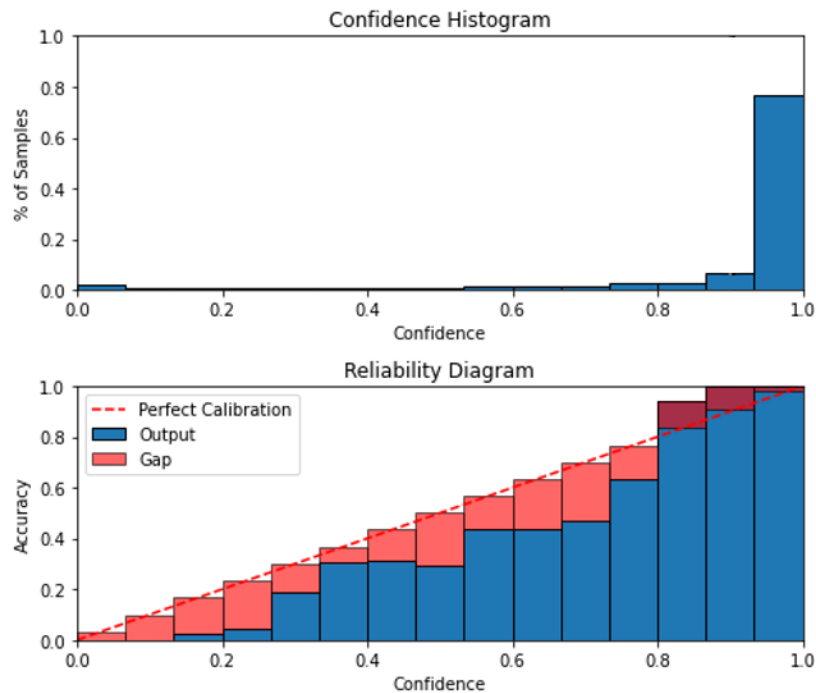


Figure 3.11: Example of Confidence Histogram and Reliability Plot.

Recalibration Techniques

Platt scaling:

A method for transforming model's outputs from $[-1;+1]$ to posterior probabilities proposed by Platt in 1999 [26]. Mostly efficient for SVM algorithm. $F(x_i)$ is the probability assigned to the record by the classifier for x_i . Two classes -1 and +1 are assumed to be arbitrarily labeled, and the classifier assigns those records a class of sign ($f(x_i)$). Given that x_i is observed, $P(y = 1)$ is the probability that x_i belongs to the class $y = 1$. Based on the logistic regression equation, the calibrated probabilities are as follows:

$$P(y = 1 | x_i) = \frac{1}{1 + \exp(Af(x_i) + B)} \quad (3.29)$$

A and B are scaling parameters that control how scaling is applied in a fitting process (using a maximum likelihood estimation algorithm). They are calculated per neighborhood or bin by applying a maximum likelihood estimation algorithm, which seeks to find the slightest difference between the mean and true probability.

Isotonic Regression:

A method developed by Zadrozny and Elkan in 2002 [27] for calibrating predictions from Naive Bayes, SVM, and Decision Tree models. In statistics, isotonic or monotonic regression is applied when fitting a line to a sequence of observations under certain conditions: the fitted line must be non-decreasing (or non-increasing) everywhere, and it must be as near to the observations as possible. Since isotonic regression is unconstrained by any functional form, such as linear regression's linearity, so long as the function is monotonically increasing, it has the advantage of being unbiased.

$$y_i = m(f_i) + \epsilon_i \quad (3.30)$$

$$\hat{m} = \operatorname{argmin}_z \sum (y_i - z(f_i))^2 \quad (3.31)$$

3.4.3 Metamodel

Meta Modeling (MM), illustrated in the Figure, combines two models, a base model that performs the primary task (e.g., regression) and a meta model that predicts the error behavior of the base model. The amount of information shared between these two is what separates the different settings, namely (1) base and meta components are trained jointly (joint models), (2) base and meta components are trained separately (black-box), and (3) base and meta components are trained independently (white-box).

With white-box and joint models, the meta model has access to rich information to capture salient patterns, which lets it generate accurate predictions. On the other hand, the black-box

setting often occurs in practice (Navratil et al., 2020) [9].

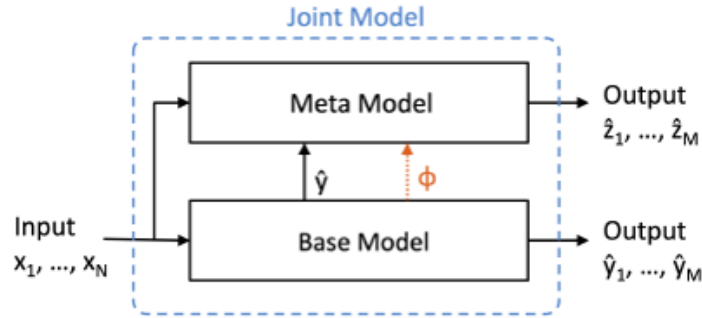


Figure 3.12: The concept of meta modeling. Source: [9]

3.4.4 Bayesian Neural Network

The literature defines BNNs slightly differently, but a widely accepted definition is that a BNN is a stochastic artificial neural network trained with Bayesian inference.

The goal of standard neural networks (SNNs) is to represent an arbitrary function $y = \Phi(x)$. SNNs are built using one input layer l_0 , a succession of hidden layers l_i , $i = 1, \dots, n - 1$, and one output layer l_n . Here, $n + 1$ is the total number of layers (Jospin et al., 2020) [10].

A BNN is designed by choosing a deep neural network architecture, i.e., a functional model. To determine the probability distribution $p(\theta)$ and the confidence in the predictive power of a model $p(y|x, \theta)$, one has to choose a stochastic model (Figure 3.13a). The hypothesis H is the model parameterization, and the data D is the training data. A BNN's stochastic model is analogous to the choice of a loss function when training a point estimate neural network. Model parameters are defined by θ , the training set by D , the training inputs by D_x , and the training labels by D_y . The Bayesian posterior can be expressed as follows by applying Bayes' theorem and enforcing independence of parameters and input:

$$p(\boldsymbol{\theta} | D) = \frac{p(D_y | D_x, \boldsymbol{\theta}) p(\boldsymbol{\theta})}{\int_{\boldsymbol{\theta}'} p(D_y | D_x, \boldsymbol{\theta}') p(\boldsymbol{\theta}') d\boldsymbol{\theta}'} \propto p(D_y | D_x, \boldsymbol{\theta}) p(\boldsymbol{\theta}) \quad (3.32)$$

When using a BNN for prediction, the probability distribution $p(y|x, D)$, called the marginal and quantifies the model's uncertainties on its prediction, is of particular interest.

$$p(\mathbf{y} \mid \mathbf{x}, D) = \int_{\theta} p(\mathbf{y} \mid \mathbf{x}, \theta') p(\theta' \mid D) d\theta' \tag{3.33}$$

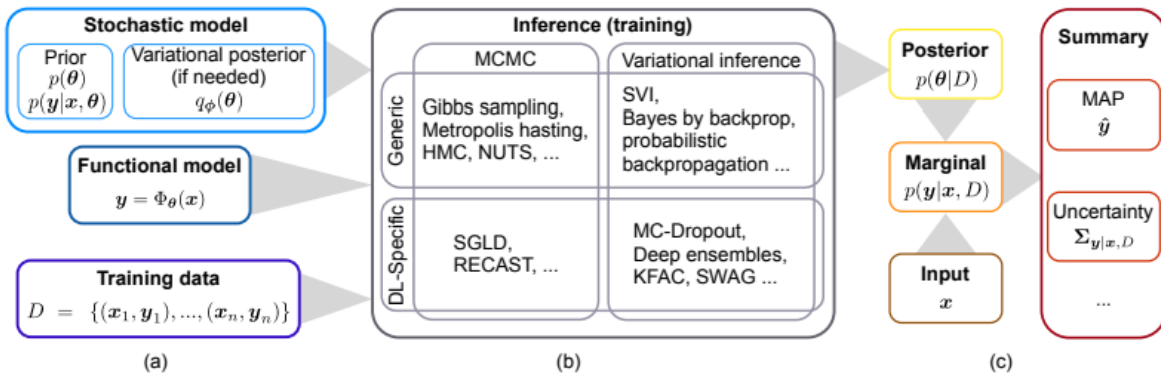


Figure 3.13: Workflow to design (a), train (b) and use a BNN for predictions (c). Source: [10]

The main difference between the SNNs and BNN is that SNN focuses on optimization while BNN focuses on marginalization. Optimization would find one optimal point to represent a weight, while marginalization would treat each weight as a variable and find its distribution (Jospin et al., 2020) [10].

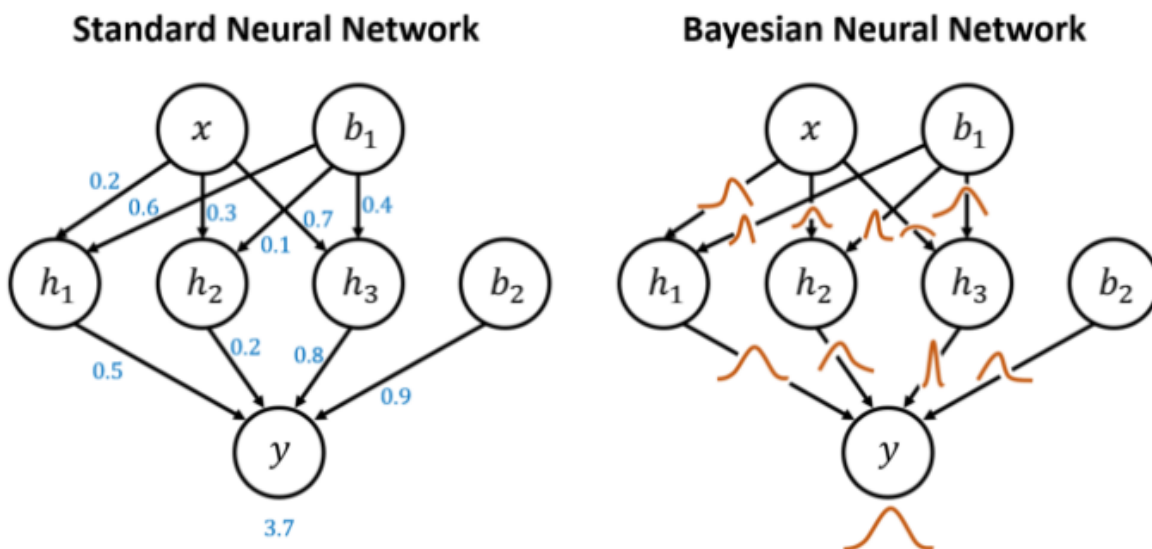


Figure 3.14: Difference between Standard NN Bayesian NN. Source: [11]

Uncertainties in statistics can be divided into two types which are epistemic and aleatory uncertainties:

- Aleatoric Uncertainty

Statistical uncertainty is also known as aleatoric or data uncertainty. A random variable is representative of different unknowns each time we run the same experiment (train the ML model). This refers to the degree of uncertainty associated with the model outputs in ML. It can be regarded as the confidence level of the prediction. Aleatoric uncertainty refers to the irreducible part of the uncertainty.

- Epistemic Uncertainty

A systematic uncertainty is also known as epistemic or algorithmic uncertainty. In deep learning, epistemic uncertainty primarily refers to the uncertainty of the model weights. In this case, a tiny number of training data points in some regions causes uncertainty, and one cannot be sure how the model should behave. The weights may change slightly each time the model is trained. These variations are the result of epistemic uncertainty. As opposed to aleatoric uncertainty, epistemic uncertainty can be reduced with additional information.

In general, epistemic uncertainty focuses on the possibility of a single event (or a single statement that may be true). In contrast, aleatoric uncertainty focuses on the possible results of repeated experiments.

Chapter 4

Data Analytics

4.1 Data Labeling

In ML, labeled data consists of data marked up or annotated to show the target, which is the outcome to be predicted by the model. The term "data labeling" encompasses many tasks that include data tagging, annotations, moderation, transcription, and processing.

The first step of the process has been the data extraction from the Volve field dataset and labeling of the penetrated depth intervals. There are 9 wells in the preprocessed Volve dataset: F-5, F-7, F-9, F-9a, F-14, F-15, F-15S, F-9 and F-9a. According to the Mud Sampling reports and Well Log Interpretation results, each depth interval is labeled with a certain formation and its lithology class.

4.2 Lithology Classes Distribution

An important discovery was that target classes are imbalanced, with Claystone samples reaching up to 40% of the data as shown on the histogram 4.1. Analyzing the lithology columns in Figures 4.2 and 4.3, the wells F-9 and F-9a penetrate the formation belonging to Nordland GP, which predominantly consists of the Claystone class. These wells were eliminated from the dataset to reduce the skewness in lithology class distribution.

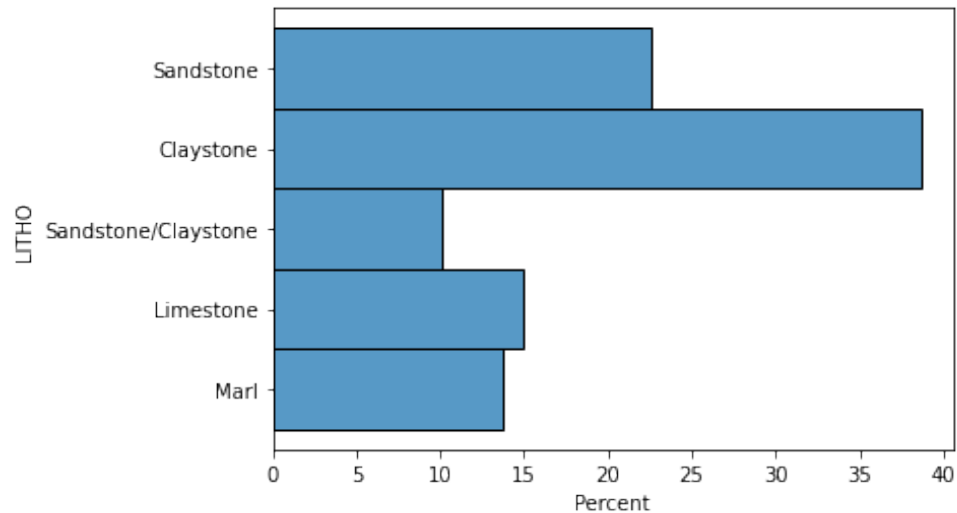


Figure 4.1: Distribution of the formations in the data set. The figure illustrates the skewness in the percentage distribution of the formations.

Handling the Imbalanced Data

Because of the imbalanced distribution of the target classes, classification ML algorithms tend to get biased towards the majority values and do not perform well on the minority values.

There are several drawbacks to the resampling techniques described in section 3.2.1. These include the loss of a lot of majority class data points using undersampling or creating multiple samples within a minority class. This may result in overfitting the model in the case of oversampling.

Considering the nature of the dataset, resampling techniques applied to the training dataset could lead to depth points overlapping. Hence, the trained model would provide more inaccurate predictions.

After testing the SMOTE techniques on the well F-14, the synthetic samples filled in the depth interval where no original data or information about the lithology was available.

To handle the imbalanced data, Scikit-learn comes with the *class weight* parameters for all Decision Trees, Random Forests, Logistic Regression, and Support Vector Machine algorithms.

Regardless of how many samples we have of each class in the training data, class weights give all classes equal weight in gradient updates. It prevents models from predicting the more frequent class.

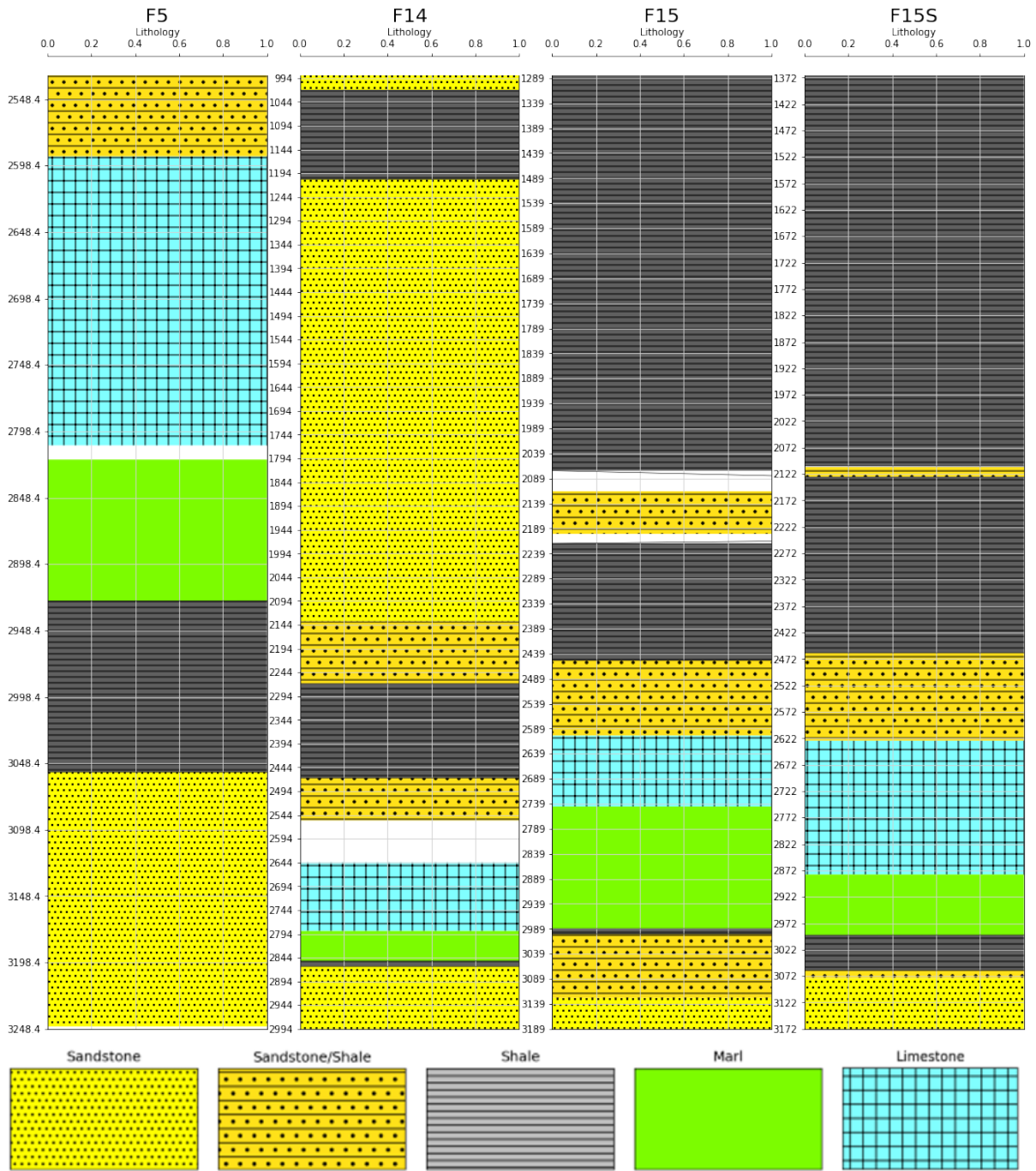


Figure 4.2: Lithology columns for wells F-5, F-14, F-15, F-15S.

The formula for calculating the weights used in this study:

$$W_i = \frac{\sum \text{Training set [Class } i]}{\sum_{i=1}^n \text{Training set}} \quad (4.1)$$

where the number of rows for each class is divided on the the total number of rows in the training dataset.

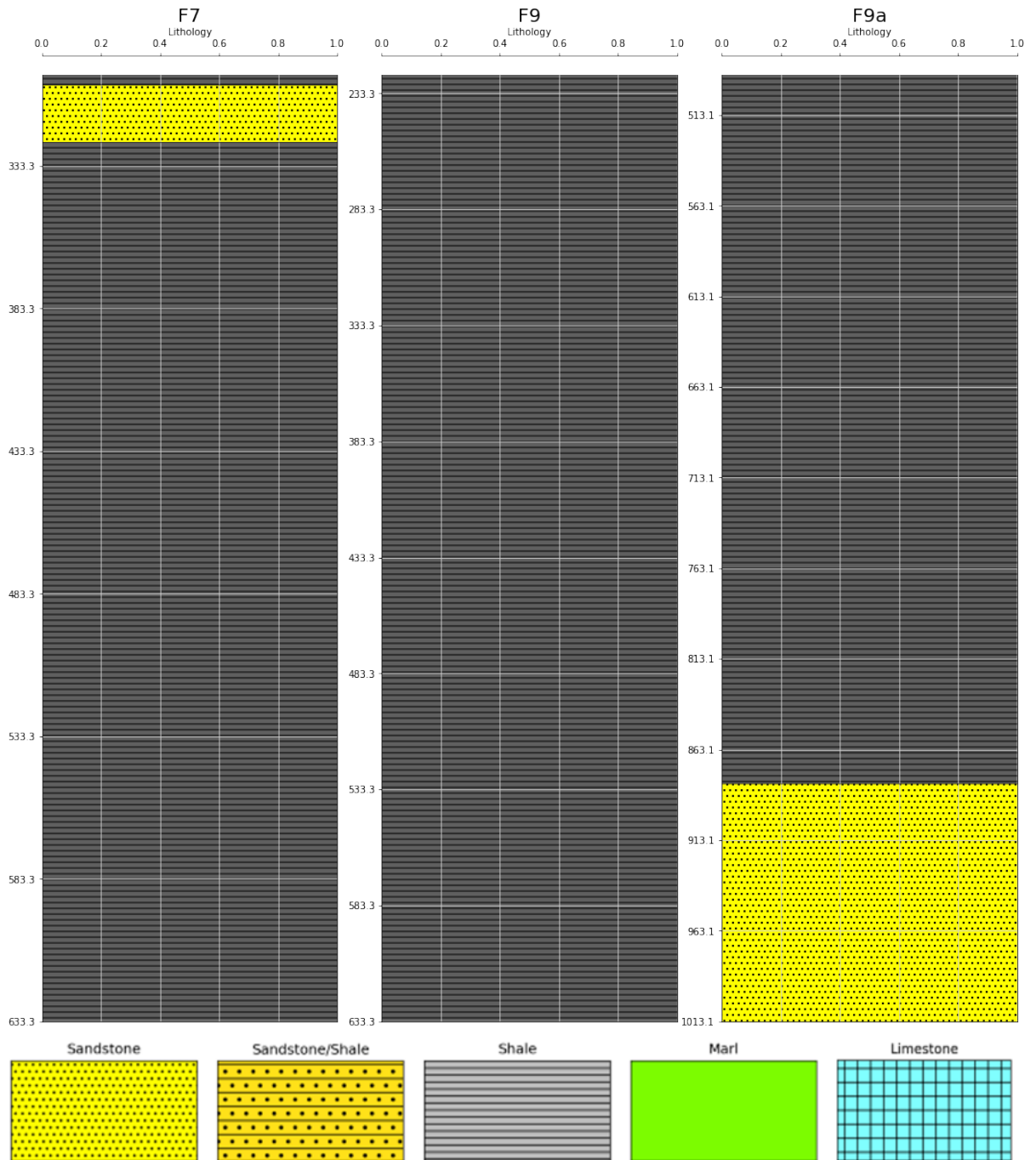


Figure 4.3: Lithology columns for wells F-7, F-9, F-9a.

4.3 Feature creation

Feature creation involves deriving new features from existing ones.

Mechanic specific energy (MSE) was calculated based on the given features in.

$$MSE = \frac{WOB}{A_b} + \frac{120\pi \cdot RPM \cdot T}{A_b \cdot ROP} [MPa] \quad (4.2)$$

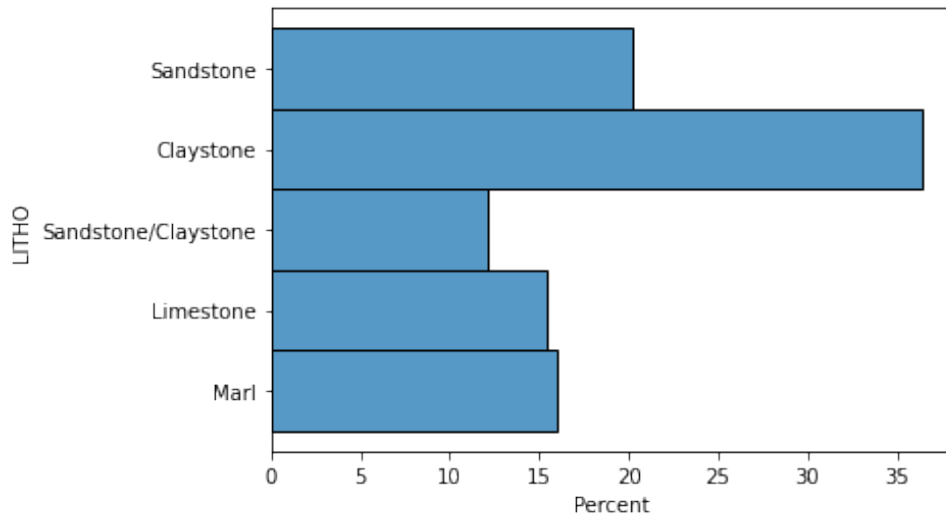


Figure 4.4: Distribution of the formations in the data set after elimination of wells F-9 and F-9a to balance the class distribution.

Class Weights	
Lithology Class	Weight
Claystone	0.36
Limestone	0.20
Marl	0.16
Sandstone	0.15
Sandstone/Claystone	0.12

Table 4.1: Class weights.

where WOB - Weight on Bit, kkgf

TOB - Average Surface Torque, kN*m

A_b - Cross-sectional area of bit, m²

RPM - Revolutions Per Minute, min⁻¹

ROP - Rate of Penetration, [m/h]

4.4 Data Quality

Outlier Detection

To identify the number of outliers in the data, the IQR score method was applied.

The interquartile range – IQR – is a widely used statistical method to identify outliers. A range between the first and third quartiles is known as the interquartile range. An outlier is a data point that is located outside either the first quartile or the third quartile by 1,5 times the IQR

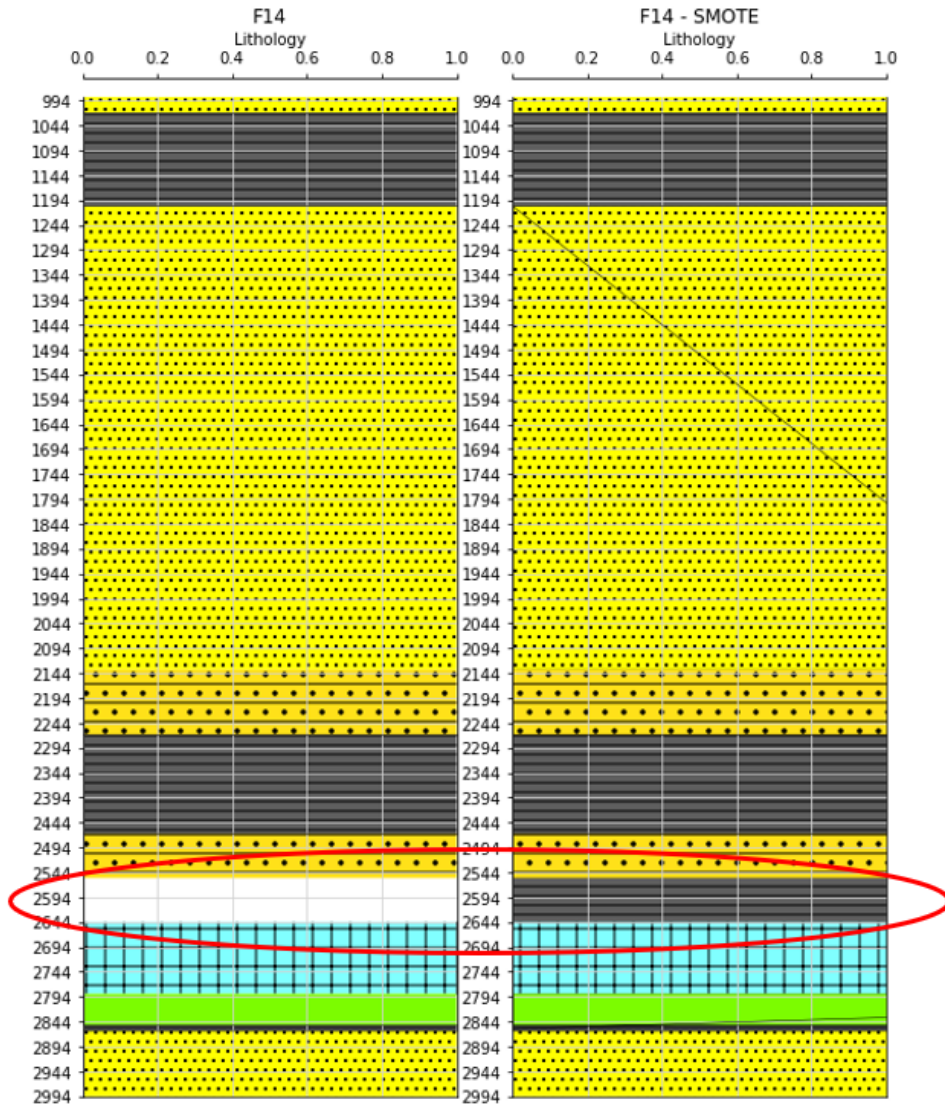


Figure 4.5: Lithology column for the well F-14 before and after implementing SMOTE technique.

[28]. The graphical representation of the IQR method in Python is a boxplot from the Seaborn library, with outliers displayed as black dots (Figure 4.6).

According to the descriptive statistics on the Figure 4.7, the total dataset has no missing values. When comparing the mean and median for all attributes, it is evident that outliers have little impact upon the distribution of the data, however IQR scores were used to identify the number of outliers.

Based on Table 4.2, the highest percentage of outliers is for Average Hookload kkgf, 14.83%, and Average Rotary Speed rpm, 7.02%. These outliers were removed from the total dataset.

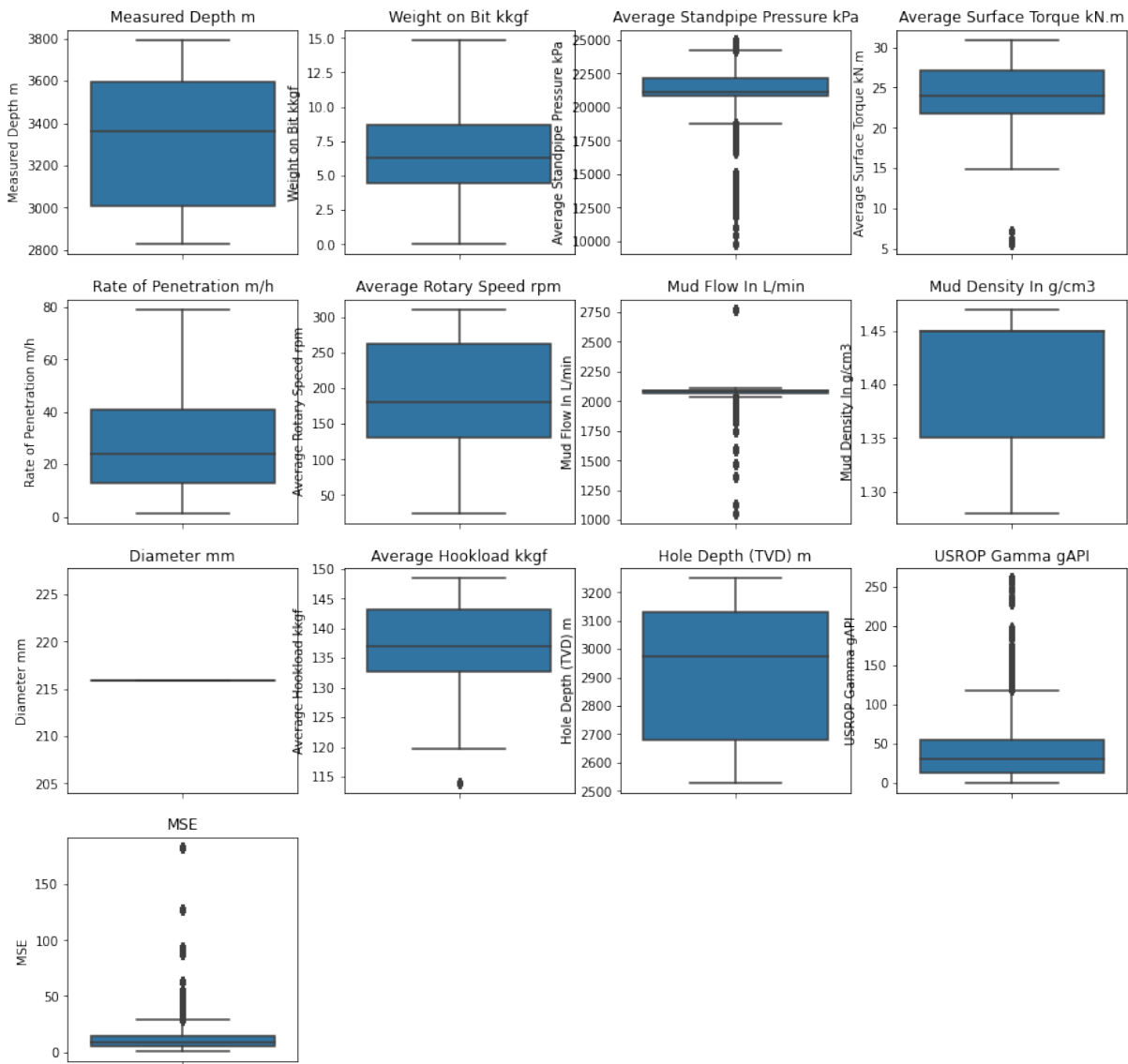


Figure 4.6: Outlier detection with boxplots.

	Measured Depth m	Weight on Bit kkgf	Average Standpipe Pressure kPa	Average Surface Torque kN.m	Rate of Penetration m/h	Average Rotary Speed rpm	Mud Flow In L/min	Mud Density In g/cm3	Diameter mm	Average Hookload kkgf	Hole Depth (TVD) m	USROP Gamma gAPI	MSE
count	198928.000000	198928.000000	198928.000000	198928.000000	198928.000000	198928.000000	198928.000000	198928.000000	198928.000000	198928.000000	198928.000000	198928.000000	198928.000000
mean	2411.781370	6.087997	17451.302922	14.430343	24.978426	152.368019	2873.576072	4.008286	297.470693	127.503708	2153.624785	67.197939	6.176068
std	1066.443448	3.957126	4236.531326	7.239629	15.086731	49.998842	1055.406219	4.496288	100.344171	15.551890	846.519288	50.519888	8.887657
min	225.171000	0.001814	1432.661618	0.008135	0.330000	0.000000	185.420836	1.020000	215.900000	84.047945	225.162770	0.000000	0.025985
25%	1548.686750	3.283486	14655.999757	9.639866	12.990000	129.250000	1993.927917	1.300000	215.900000	124.925435	1516.593360	25.380000	0.894521
50%	2697.182500	5.190356	16683.999633	12.798921	21.090000	139.736000	2121.382739	1.350000	215.900000	130.905851	2528.949222	54.490000	3.601412
75%	3288.946500	8.127138	21153.529061	17.815448	32.470001	179.283000	3987.855834	10.849026	444.500000	138.719433	2865.091577	98.140000	8.642591
max	4090.001000	31.411272	24998.459413	36.489128	99.206304	311.230011	4538.450195	12.017384	444.500000	152.926842	3248.389893	260.899000	182.781976

Figure 4.7: Descriptive statistics of the dataset.

4.5 Feature Selection

In terms of dimensionality reduction, feature selection aims to select a small number of relevant features from the original features by eliminating irrelevant, redundant or noise features. Feature selection usually leads to better learning performance, i.e., better learning accuracy,

Feature	Number of outliers	Percentage of outliers
Weight on Bit kkgf'	4859	2.42%
Average Standpipe Pressure kPa	237	0.11%
Average Rotary Speed rpm	13975	7.02%
Rate of Penetration m/h	4939	2.48%
Mud Flow In L/min	0	0%
Mud Density In g/cm ³	0	0%
Average Hookload kkgf	29519	14.83%
Hole Depth (TVD) m	0	0%
USROP Gamma gAPI	2073	1.04%
MSE	7695	3.86%

Table 4.2: Number or percentage of outliers for the features.

lower computational costs, and better interpretation of the model. There are also three types of feature selection based on the different strategies for searching, namely filter methods, wrapper methods, and embedded methods (Miao and Niu, 2016) [29].

4.5.1 Wrapper Methods

A wrapper method uses an inductive ML algorithm to estimate a subset's or an attribute's merit. The process of inductive learning involves creating a generalized rule for all data given to the algorithm. It is widely accepted as a superior method in supervised learning problems because, by engaging the inductive algorithm to evaluate options, they take into account the biases of the algorithm. Even for algorithms with moderate complexity, the number of executions required during feature search may result in high computational cost, particularly as we move towards more exhaustive search strategies (Nnamoko et al., 2014) [30].

4.5.2 Filter Methods

Filter methods use evaluation functions based solely on data properties, thus are independent of any particular ML algorithm. A common method would be to score each feature based on some criterion and provide a ranking. Several feature subsets can be selected from the ranking list either manually or by setting a threshold. As a single step approach without any search, this filter may be less optimal, but can still be extremely efficient. However, the efficiency depends on the computational complexity of the ranking procedure (Nnamoko et al., 2014) [30].

4.5.3 Embedded methods

Contrary to filter and wrapper approaches, embedded methods do not separate the learning of a class of functions from the selection of its features - it is the structure of the class of functions under consideration that is crucial. Regularization methods are among the most common embedded methods. The methods require considerable computational resources.

In order to find the most suitable features, two filter techniques Information Gain along with Correlation Coefficient were tested.

Correlation measures a linear relationship between the variables. It gives us the ability to predict one variable based on another. The logic behind using correlation for feature selection is that the variables are not correlated with one another.

Information gain is a measure of entropy reduction resulting from a transformation. This can be used for feature selection by assessing each variable's information gain in relation to the target variable.

According to the Figure 4.8, Diameter mm highly correlates with Mud Flow in L/min, 0.91. Measured depth m is well correlated with the Hole Depth (TVD) m, 0.98. From the Figure 4.9 it is seen that Diameter mm, Mud Density in g/cm³ and Measured Depth m reach lower gains than other attributes. However, Mud Density has a low correlation with other parameters. Therefore, it was made a decision to exclude Diameter mm and Measured Depth m from the dataset.

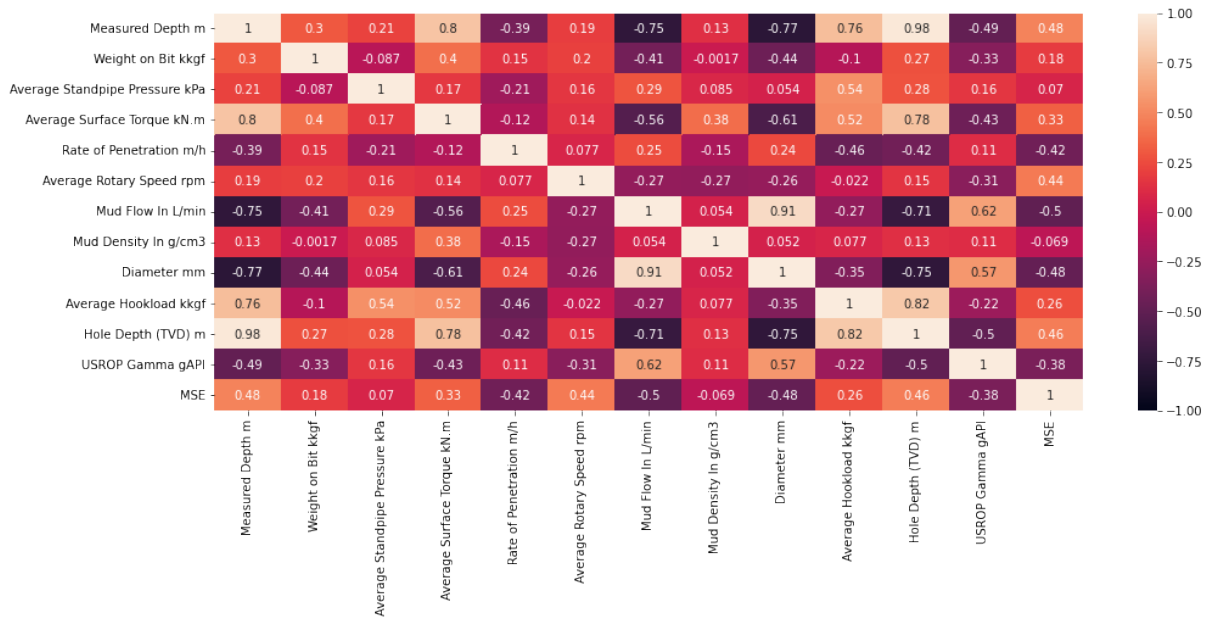


Figure 4.8: Cross correlation plot.

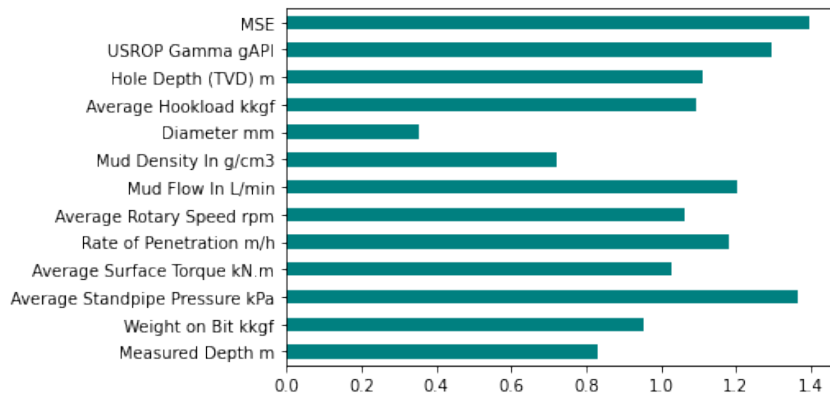


Figure 4.9: Feature Information Gain.

Chapter 5

Classification Results

Seven ML models were trained, including k-Nearest Neighbors, Logistic Regression, Naive Bayes, Decision Trees, Random Forests, Gradient Boosting, and Adaptive Boosting algorithms. The last four algorithms demonstrated the best prediction results and are presented in this chapter.

5.1 Train/Test Split

Train/test splits serve to simulate how a model will perform on unseen data by dividing the dataset into training and testing sets.

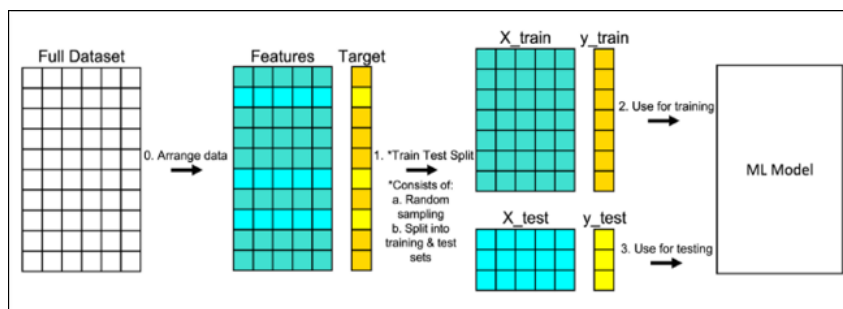


Figure 5.1: Train/test split process.

There are relatively few data samples. It was decided to test the model for the well with a minimal percentage of the total dataset. The model cannot be tested for the well F-7 since it includes only two types of lithology classes (Figure 5.2a). Therefore, F-5 was chosen to test.

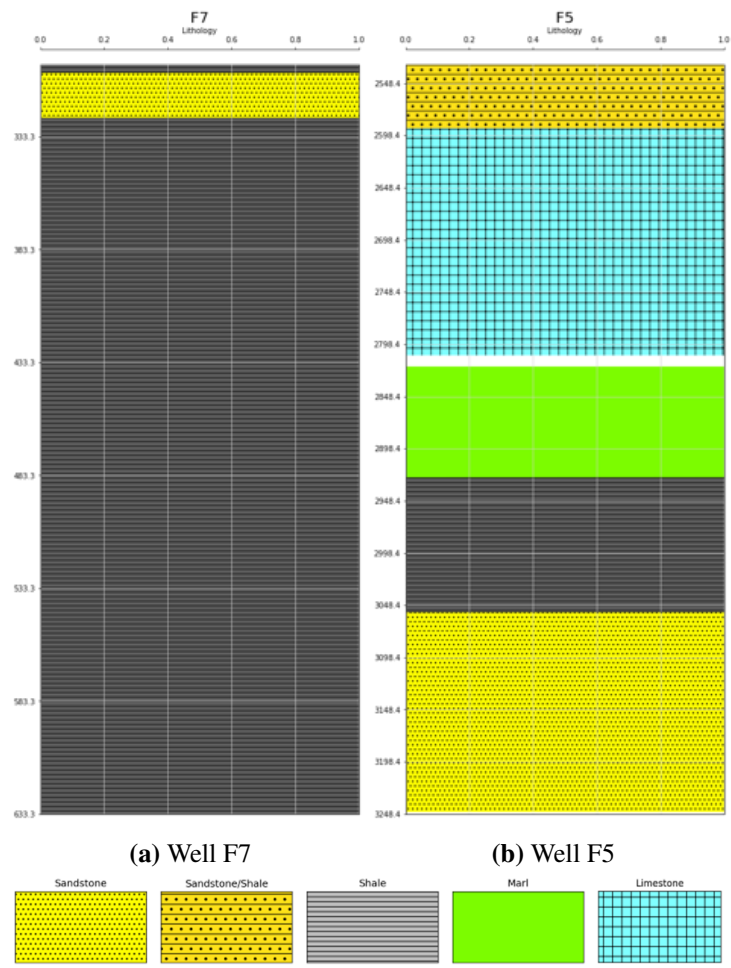


Figure 5.2: Lithology columns

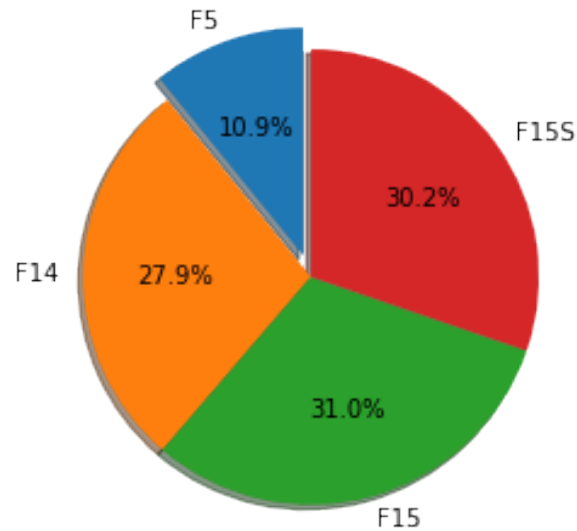


Figure 5.3: The samples percentage for each well in the dataset.

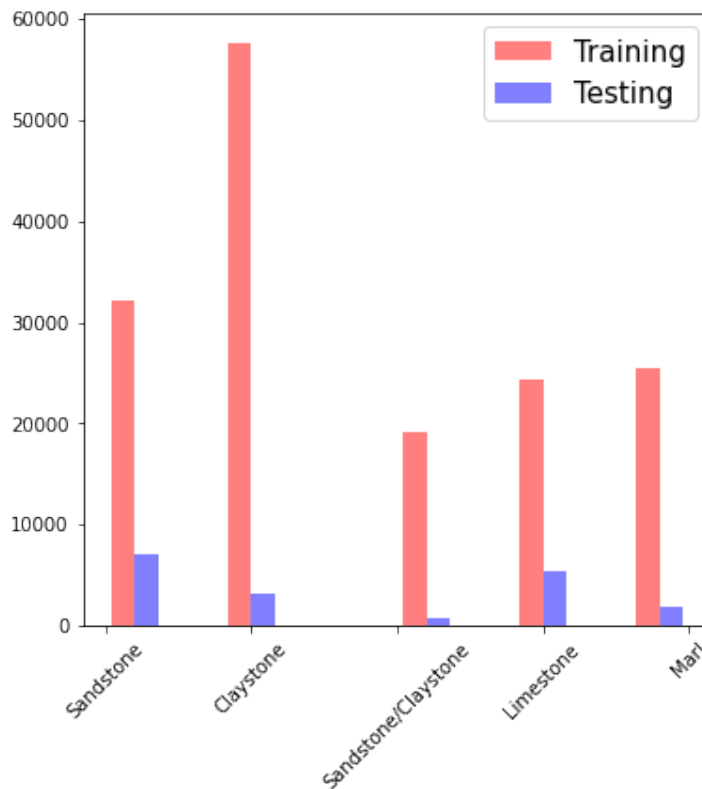


Figure 5.4: Histogram of the class distribution the training and testing sets.

5.2 Categorical Encoding

ML and Data Science activities often involve a data set containing text or categorical values (basically non-numerical values).

Label Encoding A popular method of encoding categorical variables is label encoding. The

Label Encoding	
Lithology Class	Label
Claystone	0
Limestone	1
Marl	2
Sandstone	3
Sandstone/Claystone	4

Table 5.1: Encoded labels for the lithology classes.

technique uses alphabetical ordering to assign a unique integer to each label.

One-Hot Encoding Another popular technique for handling categorical variables is One-Hot Encoding. Additional features are created based on the number of unique values in the categorical feature. For every unique value, a feature is created.

The One-Hot Encoding method is used to create dummy variables.

In this study, Label Encoding was used. In label encoding, the class names are ranked based on the alphabets, and since there are different numbers in the same column, the model interprets the data as being in some kind of order, $0 < 1 < 2$.

5.3 Experimental Setting 1 - Gradient Boosting

A model hyperparameter is a characteristic of a model that is external to the model and whose value cannot be estimated from data. The value of the hyperparameter has to be set before the learning process begins.

Grid-search is used to determine the hyperparameters of a model that produce the most 'accurate' predictions. With the *param_grid* setting, it exhaustively generates candidates based on a grid of parameter values. Cross-validation splitting is determined by *cv* and *n_jobs* is the number of parallel jobs to run, where -1 means that all the processors are being used.

Some of the hyperparameters that influence the performance of the Gradient Boosting algorithm are:

- *n_estimators* is the number of boosting stages to be performed
- *max_depth* is the maximum depth of individual regression estimators
- *learning_rate* reduces the contribution of each tree

```

1 from sklearn.tree import GradientBoostingClassifier
2 from sklearn.model_selection import GridSearchCV
3 gbc = GradientBoostingClassifier()
4 # Create the parameter grid based on the results of random search
5 params = {"n_estimators": [5, 50, 250, 500],
6           "max_depth": [1, 3, 5, 7, 9],
7           "learning_rate": [0.01, 0.1, 1, 10, 100]}
8 grid_search = GridSearchCV(estimator = gbc, param_grid = params, cv =
9                             3, n_jobs = -1, verbose = 1, scoring = "accuracy")
9 # Fit the grid search to the data
10 grid_search.fit(X_train, y_train)
11 print('Best Parameters: ', grid_search.best_params_, '\n')
12 Best Parameters:  {'n_estimators': 50, 'max_depth': 7, '
                    learning_rate': 0.1}

```

Listing 5.1: Code for Hyperparameter Tuning (Decision Tree)

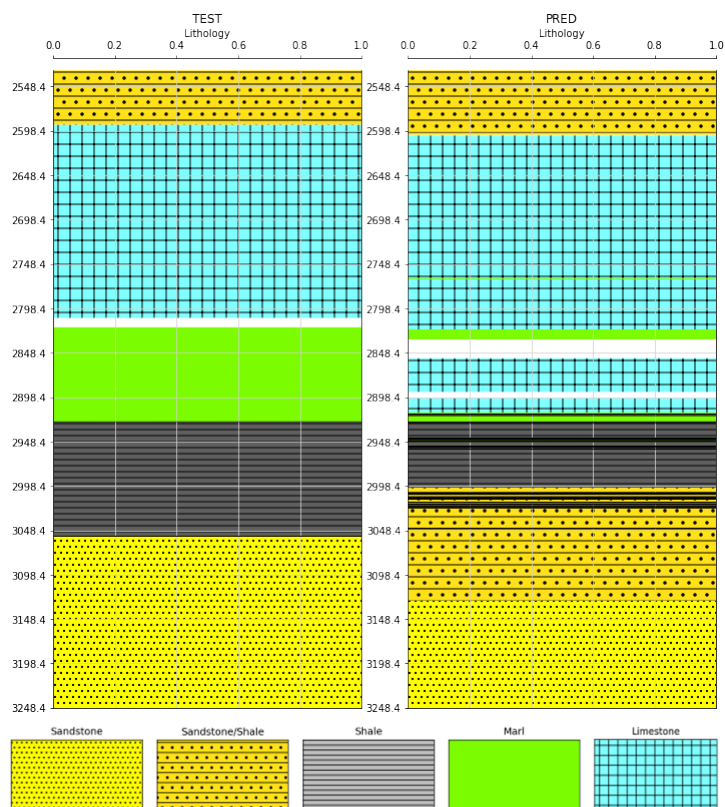


Figure 5.5: Lithology column for tested well F-5 and predicted lithology classes - Gradient Boosting algorithm.

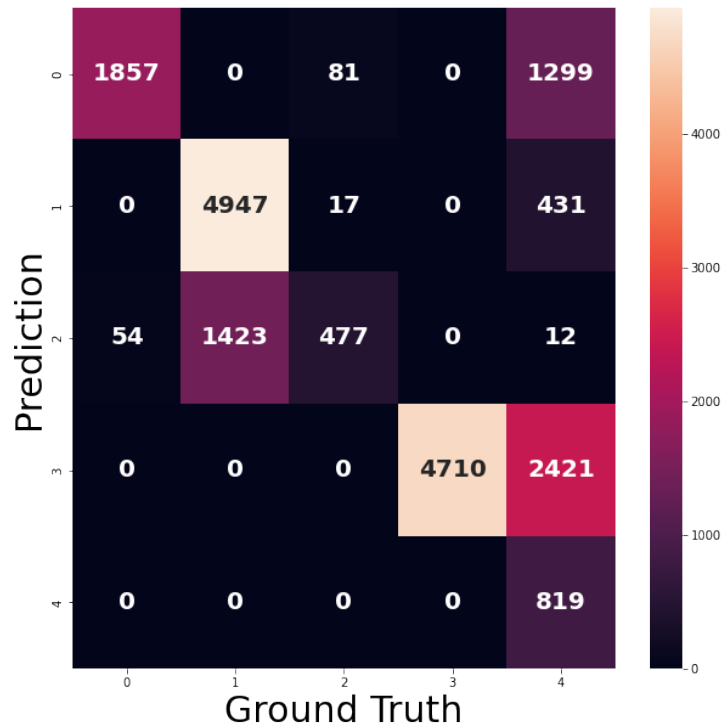


Figure 5.6: Confusion matrix for Gradient Boosting algorithm.

Classification Report for Gradient Boosting			
	Precision	Recall	F1-score
Claystone	0.97	0.57	0.72
Limestone	0.78	0.92	0.84
Marl	0.85	0.24	0.38
Sandstone	1.00	0.66	0.80
Sandstone/Claystone	0.16	1.00	0.28
Weighted Average	0.75	0.68	0.60
Accuracy	0.69		

Table 5.2: Scoring metrics for the Gradient Boosting algorithm.

The scoring results are presented in Table 5.2. From the confusion matrix (Figure 5.5), it is evident that the algorithm has poorly predicted class 2 (Marl) and class 3 (Sandstone) but provided the absolutely accurate prediction of class 4 (Sandstone/Claystone).

5.4 Experimental Setting 2 - Decision Tree

The following hyperparameters among others control the Decision Tree's algorithm:

- *max_depth* identifies the maximum number of nodes from the root to the farthest leaf
- *criterion* measures the split quality

Classification Report for Decision Tree			
	Precision	Recall	F1-score
Claystone	0.62	0.22	0.33
Limestone	0.92	0.86	0.89
Marl	0.48	0.76	0.59
Sandstone	0.96	0.88	0.92
Sandstone/Claystone	0.19	0.62	0.29
Weighted Average	0.63	0.67	0.60
Accuracy	0.70		

Table 5.3: Scoring metrics for the Decision Tree algorithm.

- `min_samples_leaf` is the minimum number of samples must be present at each leaf node

```

1 from sklearn.tree import DecisionTreeClassifier
2 from sklearn.model_selection import GridSearchCV
3 dt = DecisionTreeClassifier(class_weight=class_weights)
4 # Create the parameter grid based on the results of random search
5 params = {'max_depth': [2, 3, 5, 10, 20],
6           'min_samples_leaf': [5, 10, 20, 50, 100],
7           'criterion': ["gini", "entropy"]}
8 grid_search = GridSearchCV(estimator = dt, param_grid = params, cv =
9                             3, n_jobs = -1, verbose = 1, scoring = "accuracy")
10 # Fit the grid search to the data
11 grid_search.fit(X_train, y_train)
12 print('Best Parameters: ', grid_search.best_params_, '\n')
Best Parameters: {'max_depth': 10, 'min_samples_leaf': 20, '
criterion': "gini"}

```

Listing 5.2: Code for Hyperparameter Tuning (Decision Tree)

The performance results for the Decision Tree algorithm are shown in Table 5.2. Based on the confusion matrix 5.8, none of the classes were accurately predicted. However, compared to Gradient Boosting, class 2 (Marl) and class 2 (Sandstone) were predicted more accurately.

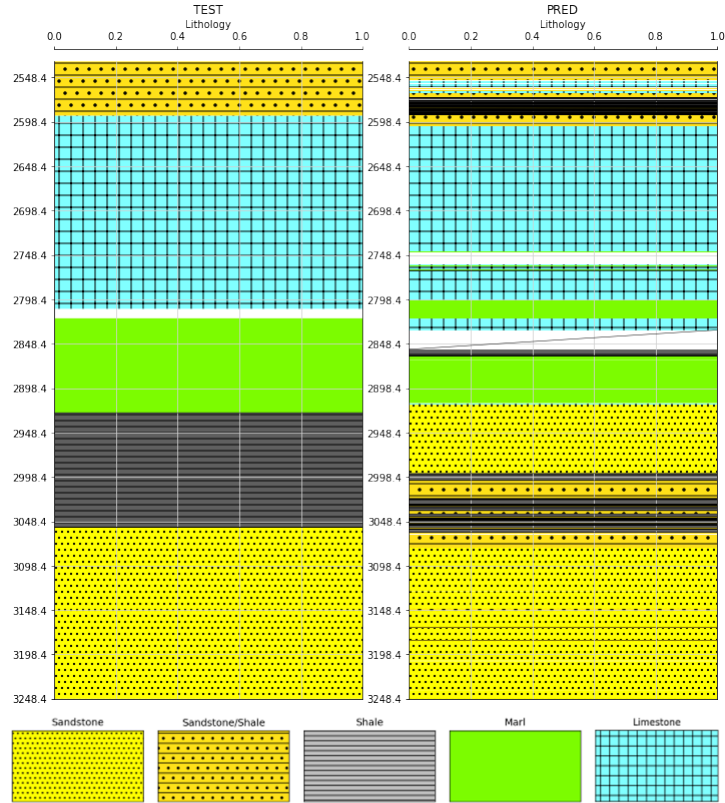


Figure 5.7: Lithology column for tested well F-5 and predicted lithology classes - Decision Tree algorithm.

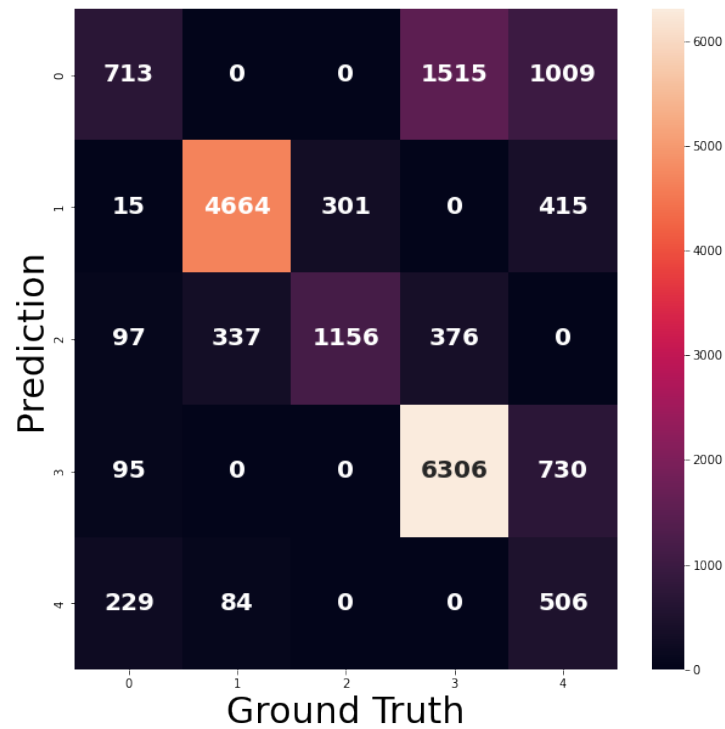


Figure 5.8: Confusion matrix for Decision Tree algorithm.

Classification Report for Random Forest			
	Precision	Recall	F1-score
Claystone	0.99	0.08	0.15
Limestone	0.84	1.00	0.92
Marl	0.85	0.53	0.65
Sandstone	0.97	0.83	0.89
Sandstone/Claystone	0.16	0.88	0.27
Weighted Average	0.89	0.72	0.72
Accuracy	0.72		

Table 5.4: Scoring metrics for the Random Forest algorithm.

5.5 Experimental Setting 2 - Random Forest

Hyperparameters which are controlling the Random Forest algorithm:

- *n_estimators* is the number of trees in the forest
- *max_depth* is the maximum depth of the tree
- *min_samples_leaf* is the minimum number of samples needed at a leaf node

```

1 from sklearn.tree import RandomForestClassifier
2 from sklearn.model_selection import GridSearchCV
3 rf = RandomForestClassifier(class_weight=class_weights)
4 # Create the parameter grid based on the results of random search
5 params = {'max_depth': [80, 90, 100, 110],
6           'min_samples_leaf': [3, 4, 5],
7           'n_estimators': [100, 200, 300, 1000]}
8 grid_search = GridSearchCV(estimator = rf, param_grid = params, cv =
9                             3, n_jobs = -1, verbose = 1, scoring = "accuracy")
9 # Fit the grid search to the data
10 grid_search.fit(X_train, y_train)
11 print('Best Parameters: ', grid_search.best_params_, '\n')
12 Best Parameters: {'max_depth': 100, 'min_samples_leaf': 5, '
n_estimators': 100}

```

Listing 5.3: Code for Hyperparameter Tuning (Random Forest)

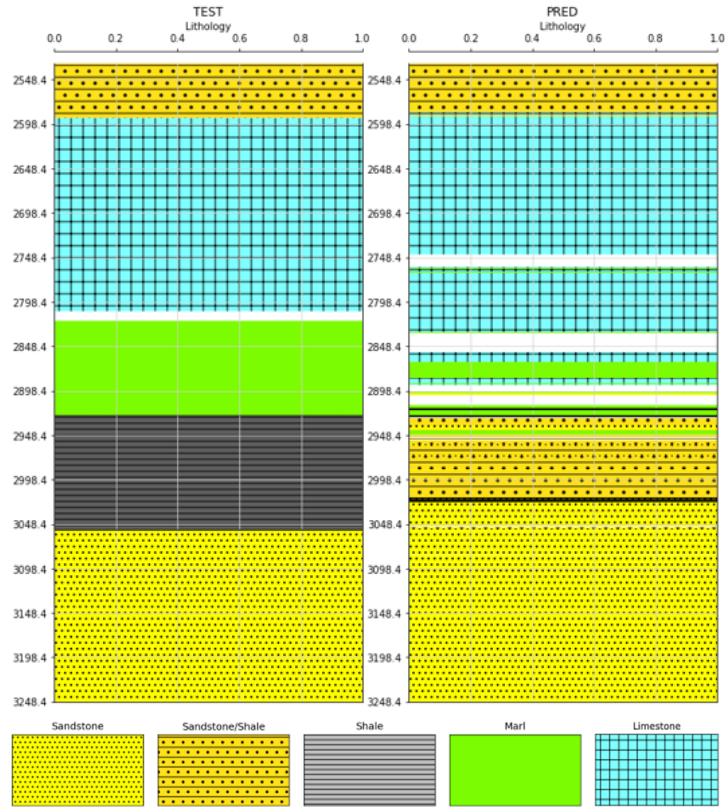


Figure 5.9: Lithology column for tested well F-5 and predicted lithology classes - Random Forest algorithm.

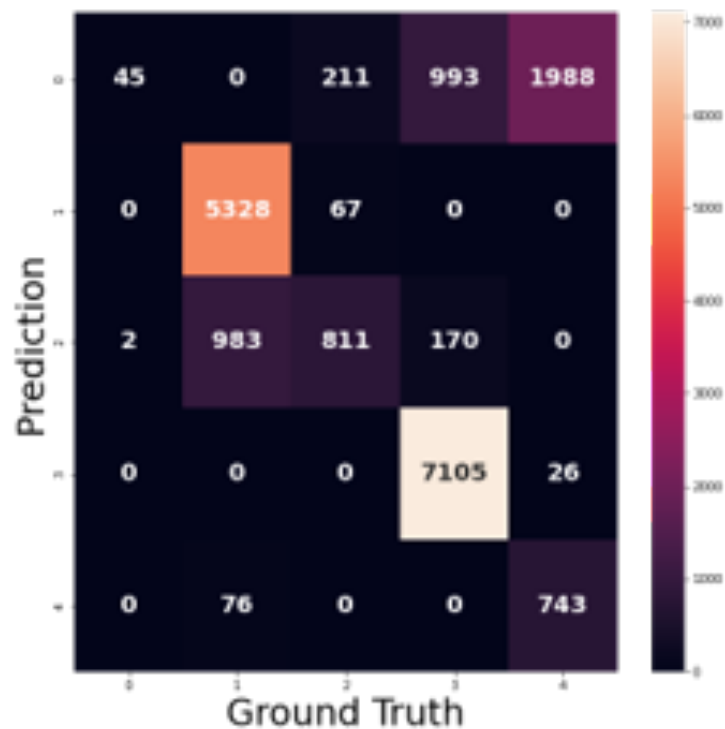


Figure 5.10: Confusion matrix for Random Forest algorithm.

The Random Forest algorithm does not distinguish the Claystone from the Sandstone/Claystone class (Figure 5.9). Marl class was also poorly differentiated from the Limestone class.

However, other classes were predicted quite precisely. According to the Confusion matrix 5.10, the algorithm performed better regarding class error differentiation than the Gradient Boosting and Decision Tree algorithms.

5.6 Experimental Setting 4 - Adaptive Boosting

The main hyperparameters which are impacting the Adaptive Boosting algorithm are:

- *n_estimators* is the maximum number of estimators at which boosting is terminated.
- *learning_rate* refers to the weight applied to each classifier at each boosting iteration.
- *algorithm* is the option that includes the choice between a real boosting algorithm (SAMME.R) and a discrete boosting algorithm (SAMME). The SAMME.R algorithm typically achieves a lower test error with fewer boosting iterations than SAMME.

```

1 from sklearn.ensemble import AdaBoostClassifier
2 from sklearn.model_selection import GridSearchCV
3 ab = AdaBoostClassifier()
4 # Create the parameter grid based on the results of random search
5 params = {'n_estimators': [50, 53, 55, 57, 60, 62],
6           'learning_rate': [(0.97 + x / 100) for x in range(0, 20, 2)],
7           'algorithm': ['SAMME', 'SAMME.R']}
8 grid_search = GridSearchCV(estimator = ab, param_grid = params, cv =
9                             3, n_jobs = -1, verbose = 1, scoring = "accuracy")
9 # Fit the grid search to the data
10 grid_search.fit(X_train, y_train)
11 print('Best Parameters: ', grid_search.best_params_, '\n')
12 Best Parameters: {'algorithm': 'SAMME', 'learning_rate': 1.089, '
n_estimators': 60}

```

Listing 5.4: Code for Hyperparameter Tuning (Adaptive Boosting)

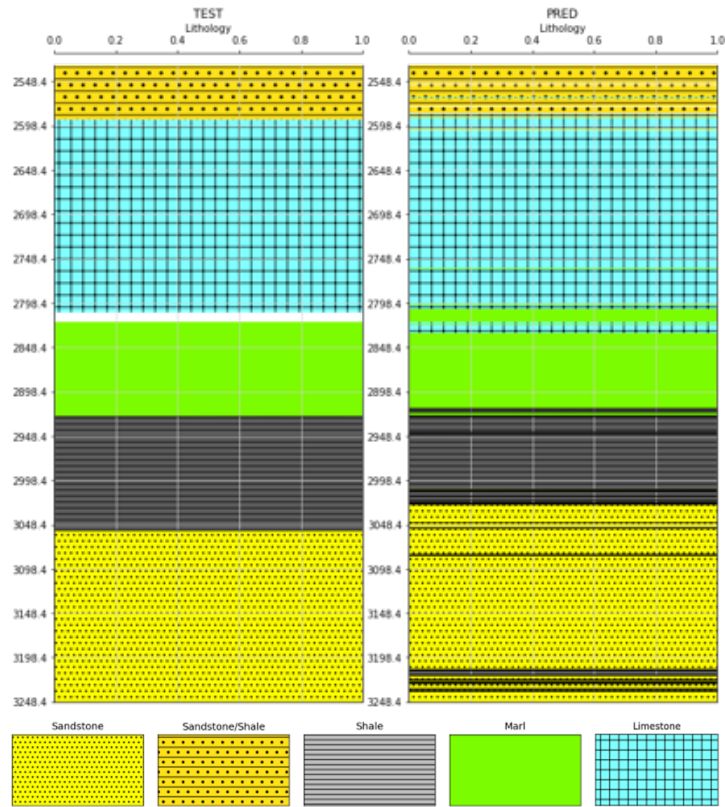


Figure 5.11: Lithology column for tested well F-5 and predicted lithology classes - Adaptive Boosting algorithm.

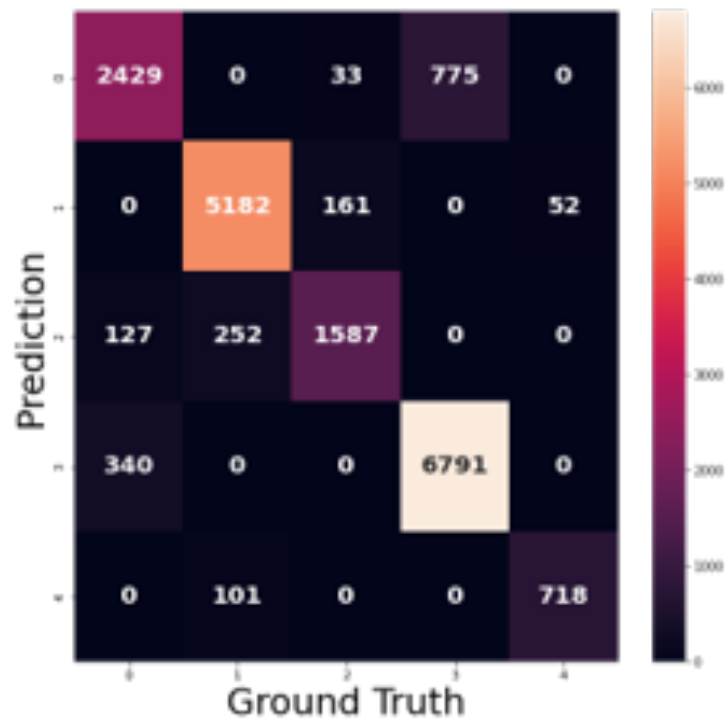


Figure 5.12: Confusion matrix for Adaptive Boosting algorithm.

Among other algorithms, adaptive boosting one demonstrated the best predictions results (Figure 5.11), although none of the classes were predicted totally accurately.

Classification Report for Adaptive Boosting			
	Precision	Recall	F1-score
Claystone	0.84	0.75	0.79
Limestone	0.94	0.96	0.95
Marl	0.89	0.81	0.85
Sandstone	0.90	0.95	0.92
Sandstone/Claystone	0.93	0.88	0.90
Weighted Average	0.90	0.90	0.90
Accuracy	0.90		

Table 5.5: Scoring metrics for the Adaptive Boosting algorithm.

After analyzing the classification results, the Random Forest and Adaptive Boosting algorithms were chosen to imply the comprehensive model evaluation analysis and uncertainty quantification.

Chapter 6

ML Model Analysis - Case Study

6.1 Model Evaluation

For the case study, it was chosen to perform a comparative analysis on the two ML models with the most accurate prediction results: Random Forest and Adaptive Boosting.

Model evaluation is the process of using different evaluation metrics to understand a ML model's performance and its strengths and weaknesses. The general metrics that evaluate the classification metrics are Accuracy, Recall, Precision, F1-score, and Confusion Matrix.

Pytolemaic package for Python, developed by Orion Talmi [31], was used in this study to analyze the classification model and dataset and measure their quality.

It is evident that the scoring metrics (Tables 5.4 and 5.5) are better for Adaptive Boosting algorithms.

By analyzing the confusion matrixes (Figures 5.10 and 5.12), the Adaptive Boosting algorithm has higher accuracy than Random Forest because of the difference in Claystone class prediction. The rest of the classes' Adaptive Boosting Classifier predicted slightly worse.

6.1.1 ROC and Precision-Recall Curves

ROC curves and Precision-Recall curves are diagnostic tools that can help interpret probabilistic forecasts for multiclassification predictive modeling problems.

a AUC: higher for Random Forest algorithm for all five classes, although its confidence in-

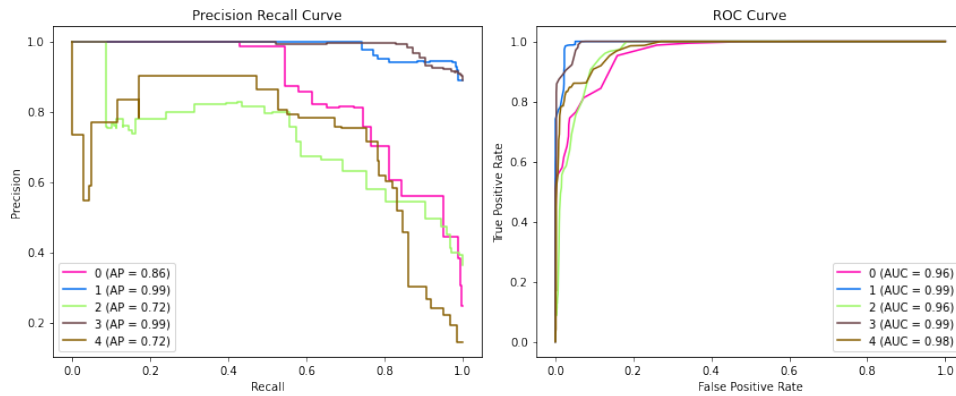


Figure 6.1: ROC- and PR-curves for Random Forest algorithm.

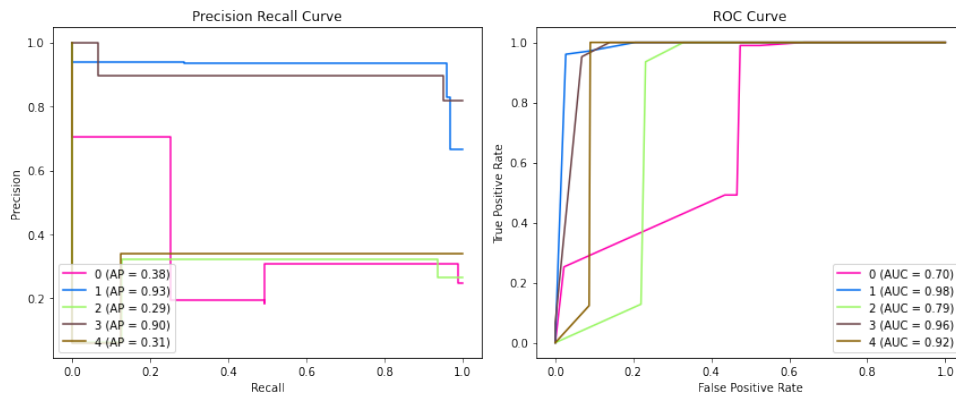


Figure 6.2: ROC- and PR-curves for Adaptive Boosting algorithm.

interval is lower and uncertainty in score calibration is higher in comparison with Adaptive Boosting algorithm. Hence the ROC curve in the case of Random Forest is not the most reliable technique to check the model's quality. Keeping in mind that the accuracy score for Adaptive Boosting reaches 90%, the AUC resembles it with a low uncertainty score.

b AUPRC: it is challenging to compare models using this metric. Considering the trade-off between recall and precision, the Random Forest is better balanced than Adaptive Boosting. The recall score is lower for Random Forest.

6.2 Feature Sensitivity and Model Vulnerability

The feature sensitivity was determined by altering or ignoring the feature value while all other features remained constant and observing the model's output. As the model's outcome changes dramatically after changing the feature value, this implies that this feature has a significant impact on the prediction.

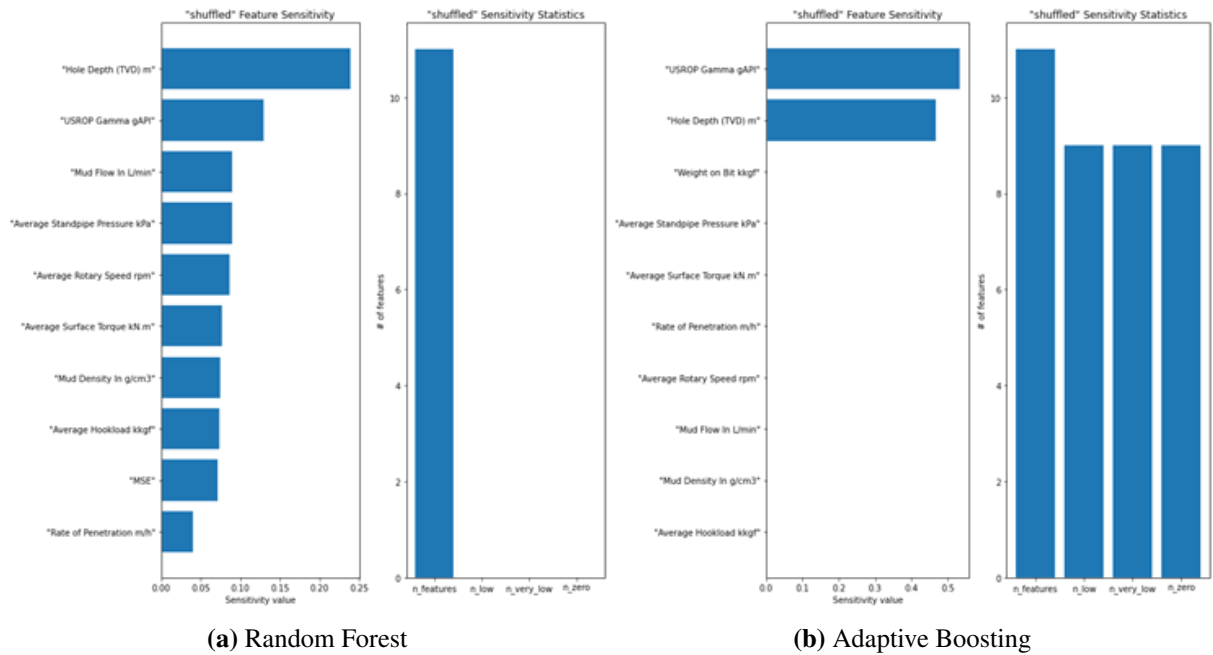


Figure 6.3: Feature Sensitivity

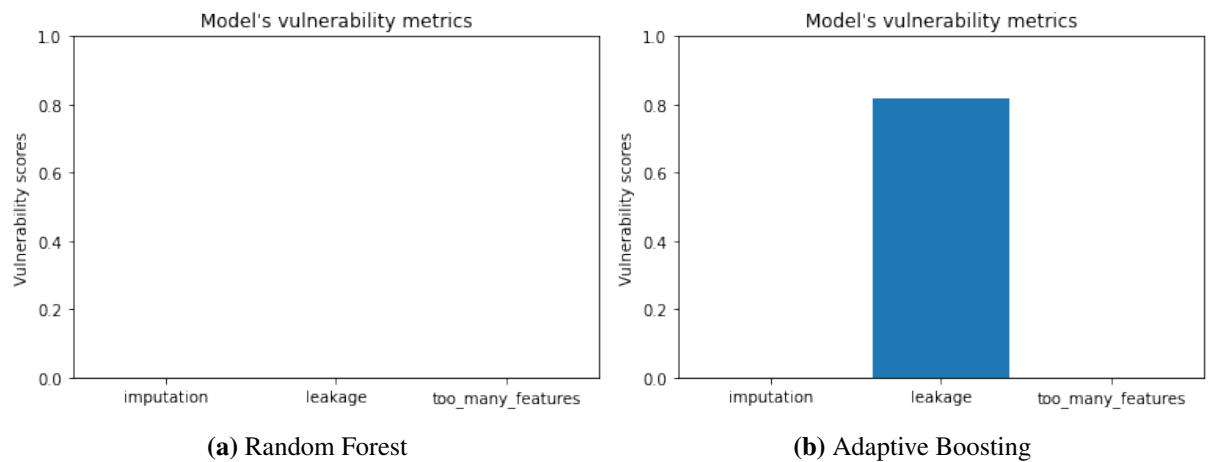


Figure 6.4: Models Vulnerability

Model vulnerability is part of feature sensitivity report. It includes three quality measurements answering the questions of how the model handles the missing values in the dataset – imputation vulnerability, were samples from the testing set used to train the model – data leakage, and is there a risk of model overfitting – too many features.

The feature sensitivity was in detail described in the section 3.3.1.

Random Forest is sensitive to the different degrees of all of the features in the training dataset. Adaptive Boosting is sensitive to USROP Gamma and Hole Depth attributes (Figures 6.3).

Adaptive Boosting is vulnerable to data leakage to a great extent, but Random Forest has

no vulnerability (Figure 6.4). The reason could be that the Random Forest algorithm allows the introduction of class weights to the model.

What could be done to improve the Adaptive Boosting algorithm?

- Minimize data leakage:
 - a By evaluating simple rule-based models that identify leaky variables, then remove them. A rule-based models produces pre-defined outcomes that are based on a set of certain rules coded by the user.
 - b In input data add the random noise for the purpose to smooth out the effects of possibly leaking variables.
 - c By using pipelines architectures, it is easy to do a different sequence of steps for data preparation which is used to be performed in cross-validation. Cross-validation is a technique for validating the model efficiency by training it on the subset of input data and testing on previously unseen subset of the input data. For this purpose, a Python Scikit-learn library could be used.
- Train the model with the sensitive features only, i.e., USROP Gamma and Hole Depth.

6.3 Covariate Shift Measurement

Covariate shift refers to the change in the distribution of the input variables present in the training and the test data.

Since the covariate shift refers only to the difference between training and testing datasets distributions, it is model agnostic. As observed, Average Surface Torque, Average Standpipe Pressure cause the covariate shift with high sensitivity, and Mud Flow cause it to a lower extent.

What could be done to improve the models?

- Adaptive Boosting is not sensitive to mentioned above attributes. Hence dataset shift does not affect its prediction quality. In Random Forest, the reweighting of the training points, which are similar to the testing, could be implemented.

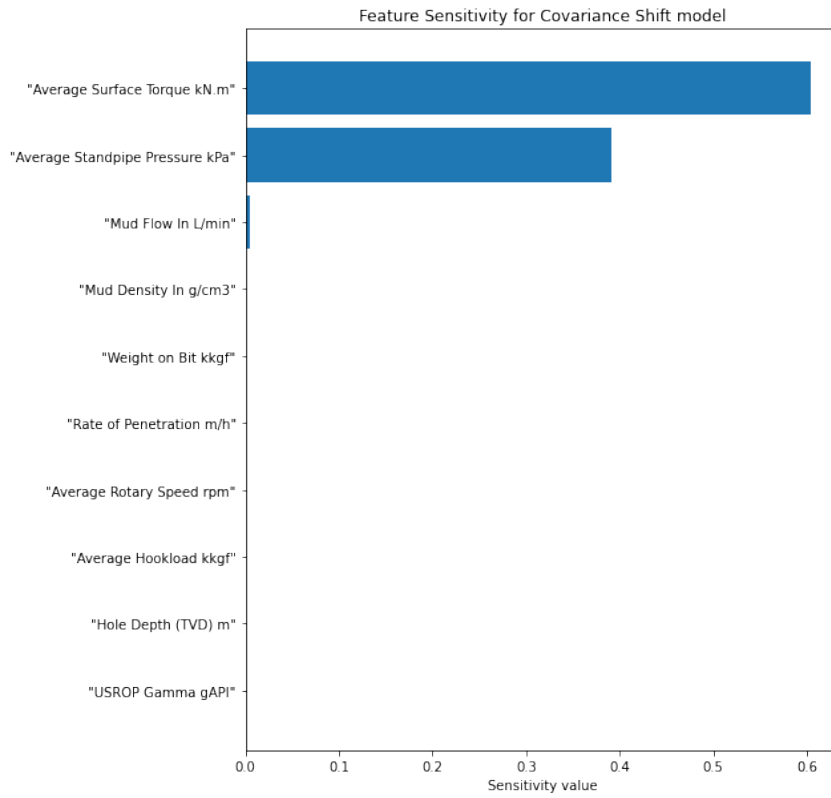


Figure 6.5: Histogram of the shifted features.

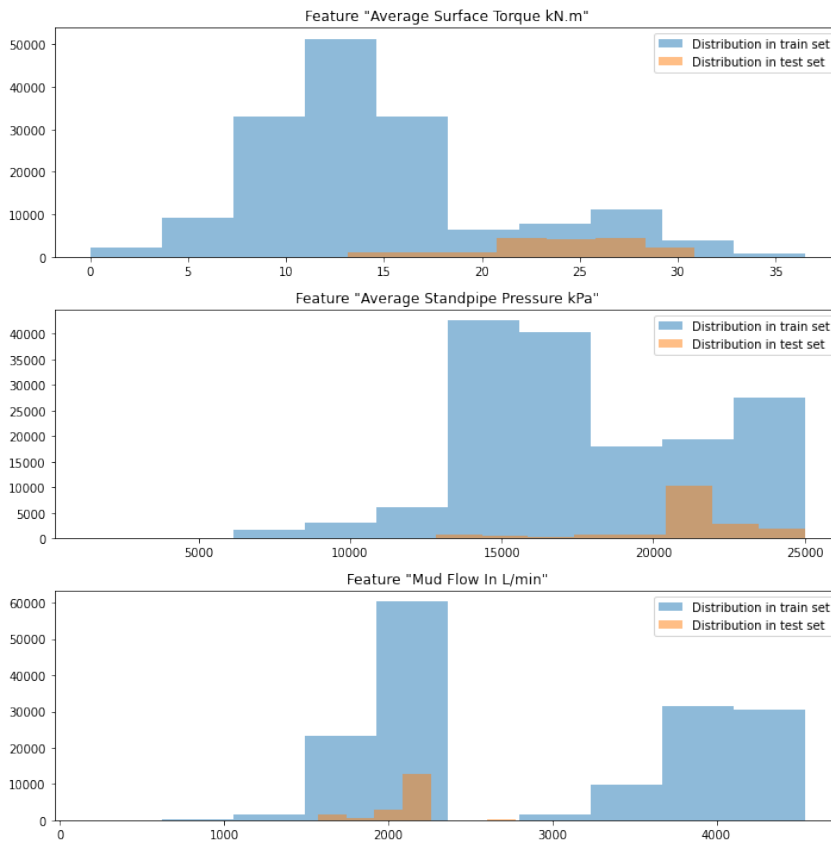


Figure 6.6: Distribution of the shifted features.

- Adversarial Search method, described in section 3.3.4, could be applied to handle the shift in the current datashift both for Random Forest and Adaptive Boosting algorithms.

6.4 Uncertainty Quantification

Model evaluation metrics and model analysis provide information about the predictive model's performance. However, it does not reveal the model's efficacy when classifying a particular sample. To estimate whether the model's predictions are reliable, the probabilities of each predicted class should be calibrated, and the model's aleatoric/epistemic uncertainties should be calculated.

The choice of the UQ algorithm depends on whether you intend to train a new model or already have a trained model. In the former case, intrinsic methods can be used to train a model that provides uncertainty estimates. If the model is already trained, extrinsic methods can be used to either improve the quality of the model's existing uncertainty estimates or generate post-hoc uncertainty estimates if your model does not provide them.

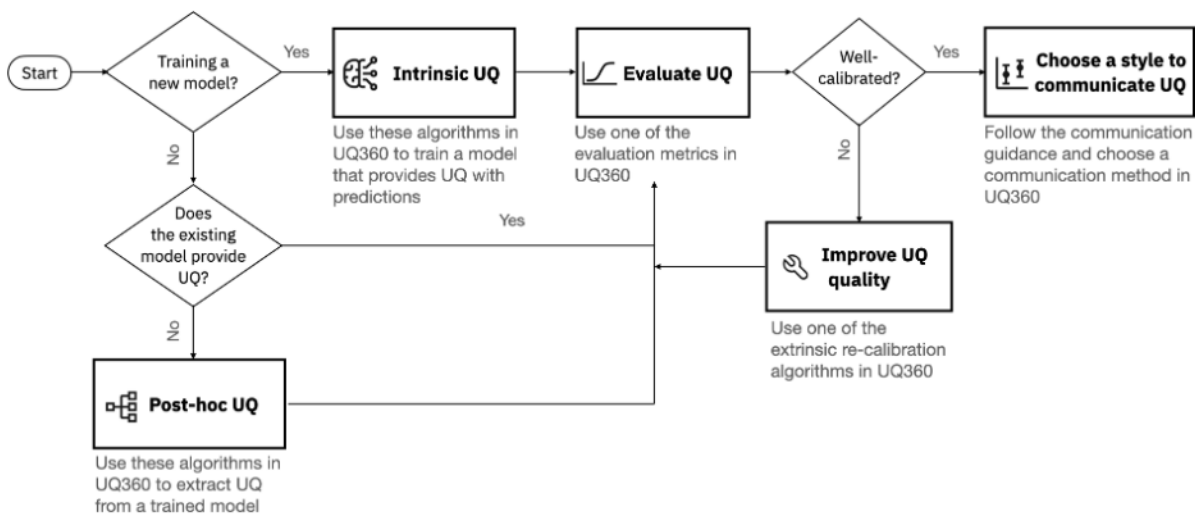


Figure 6.7: Guidance on choosing UQ algorithms. Source: [12]

In the current case, the analyzed models were already trained. Neither Random Forest nor Adaptive Boosting does not capture the data or model uncertainties. Therefore the Metamodel Classification approach will be applied to measure the post-hoc uncertainties.

Afterward, the evaluation metric for classification, the behavior of Risk vs Rejection Rate/Risk

vs Selection Threshold plots, provided in UQ360, is to be examined. The next step is to implement probability calibration measurement, and if the model is not well calibrated, the recalibration techniques are applied to improve UQ quality. After analyzing the confidence histogram and reliability plot, it would be possible to conclude the models' performance regarding the probabilities of each predicted class.

The intrinsic algorithm, Bayesian Neural Networks (BNN), allows obtaining information about the aleatoric and epistemic uncertainties of the pre-trained model. Instead of training the BNN algorithm on the testing dataset, it is trained on calibrated class probabilities matrix.

6.4.1 Blackbox Metamodel Classification

A base model performs a task, whether classification or regression, on one side, while a secondary model acts as an observer on the other. Using input and output data from the base model and the ground truth data, it is trained to capture the base model's aleatoric uncertainty.

There are several types of meta-modeling. As a more straightforward variant, the observer only sees the inputs and outputs of the base model, also known as the BlackBox method, which was used in this study.

In this study the Gradient Boosting algorithm was chosen to behave as an observer.

Risk vs Rejection Rate and Risk vs Selection Threshold plots

While analyzing the model performance, one of the condition was that the classifiers gave a prediction for all the samples.

However, in the Metamodel approach, the prediction are made only when they are sufficiently reliable, with no prediction made by the classifier in other cases. This adds the 'rejection' option to the classifier.

The classifier rejects an observation if the category to which it should be classified cannot be predicted reliably, in which case no class labels are assigned. Chow (1970) introduced the reject option, whereby decisions are not taken for samples with low confidence to reduce the possibility of error. A classifier's performance with reject options depends on accuracy and rejection rate (Nadeem et al., 2010) [32].

The risk function is the expected value of an inverted loss function. In ML, the loss function is used to measure how well your algorithm models your featured data set. In other words, loss functions measure how good your model is in predicting the expected outcome, which is, in the current case, the accuracy score.

The selection threshold refers to the model's rate of failure based on the aleatoric uncertainty score.

If both Risk vs Rejection Rate and Risk vs Selection Threshold plots are decreasing and tend to be diagonal. In that case base model performs well, and the blackbox uncertainty score is possible y_{prob} as the confidence to filter individual instances of the test set. The post-hoc UQ quality should be improved by checking the probabilities calibration.

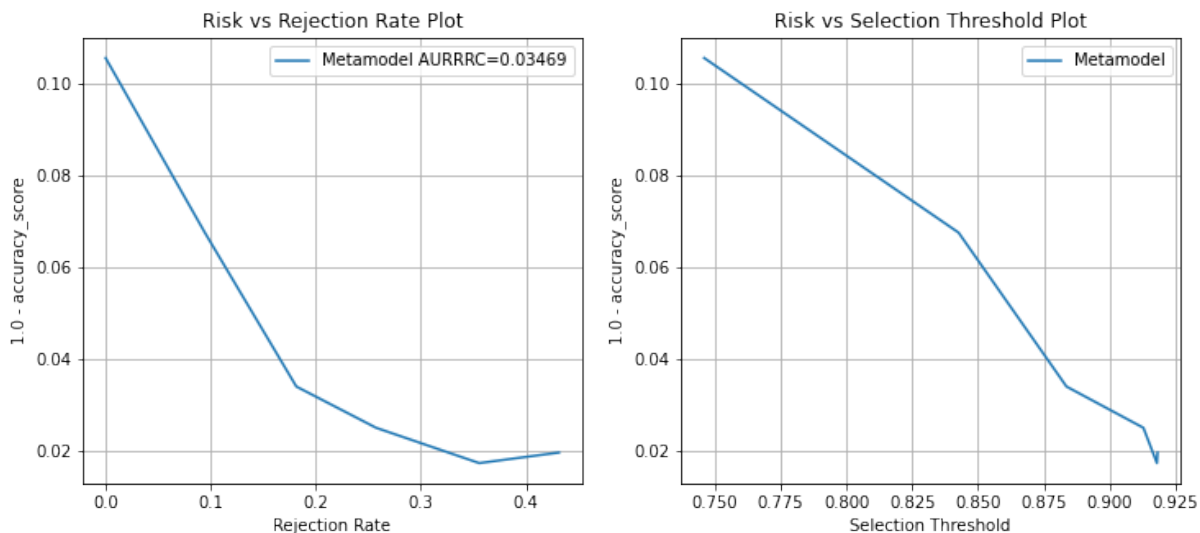


Figure 6.8: The synthetic example of Risk vs Rejection and Risk vs Selection Threshold plots for well performed model.

The Metamodel results for Random Forest demonstrate that with the decreasing accuracy, the rejection and selection threshold rates, and therefore aleatoric uncertainty, increase. Thus, the Random Forest probability score is uncertain. On the other hand, the results for Adaptive Boosting illustrate that even though the rejection and selection threshold rates decrease, the model probability score is not perfectly correlated but more confident than in the case of Random Forest.

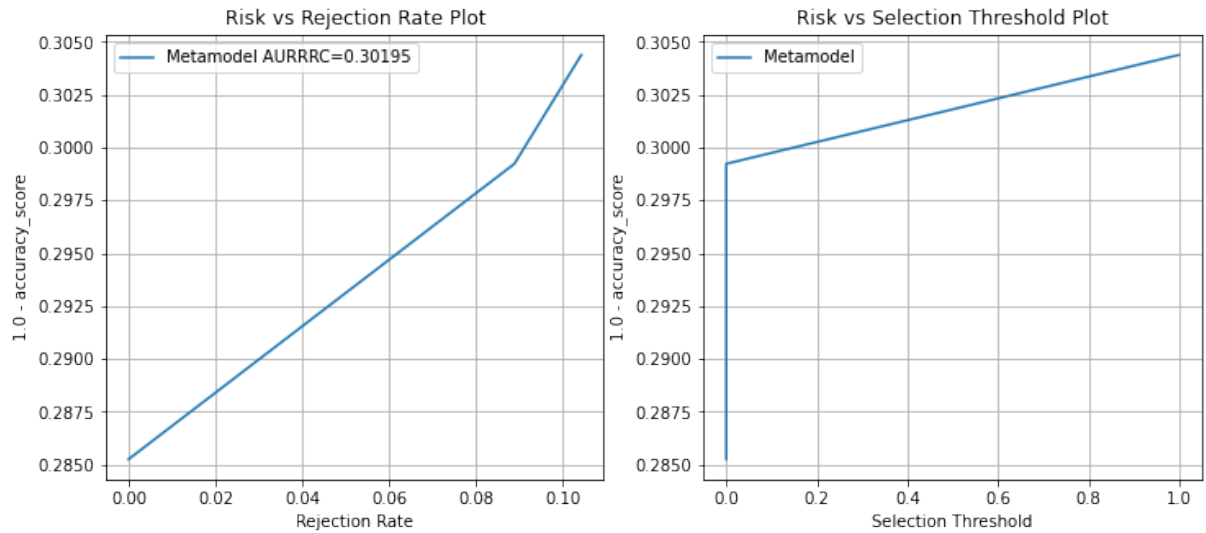


Figure 6.9: Metamodel results for Random Forest algorithm.

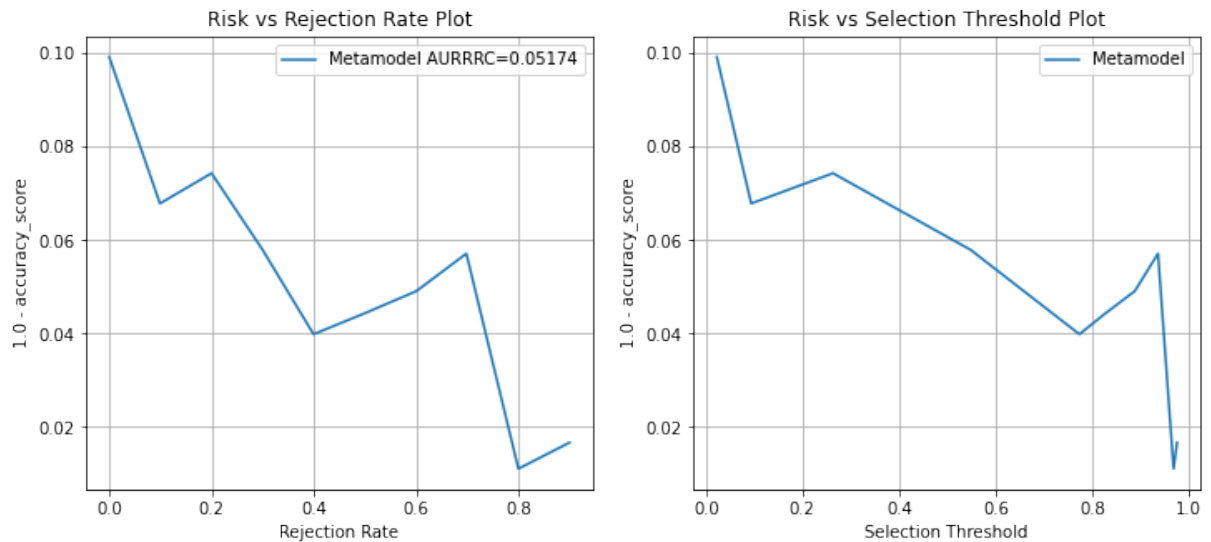


Figure 6.10: Metamodel results for Adaptive Boosting algorithm.

6.4.2 Bayesian Neural Network

The Bayesian neural network (BNN) combines neural networks with Bayesian inference. The BNN uses weights and outputs as variables to find the marginal distributions that best fit the data. Ultimately, a BNN aims to explain a prediction's reliability by quantifying the uncertainty introduced by a model's outputs and weights.

The BNN algorithm was trained on calibrated prediction probability scores from Random Forest and Adaptive Boosting algorithms.

Aleatoric or statistical uncertainty refers to the notion of randomness, that is, the variability in an experiment's outcome caused by inherent randomness.

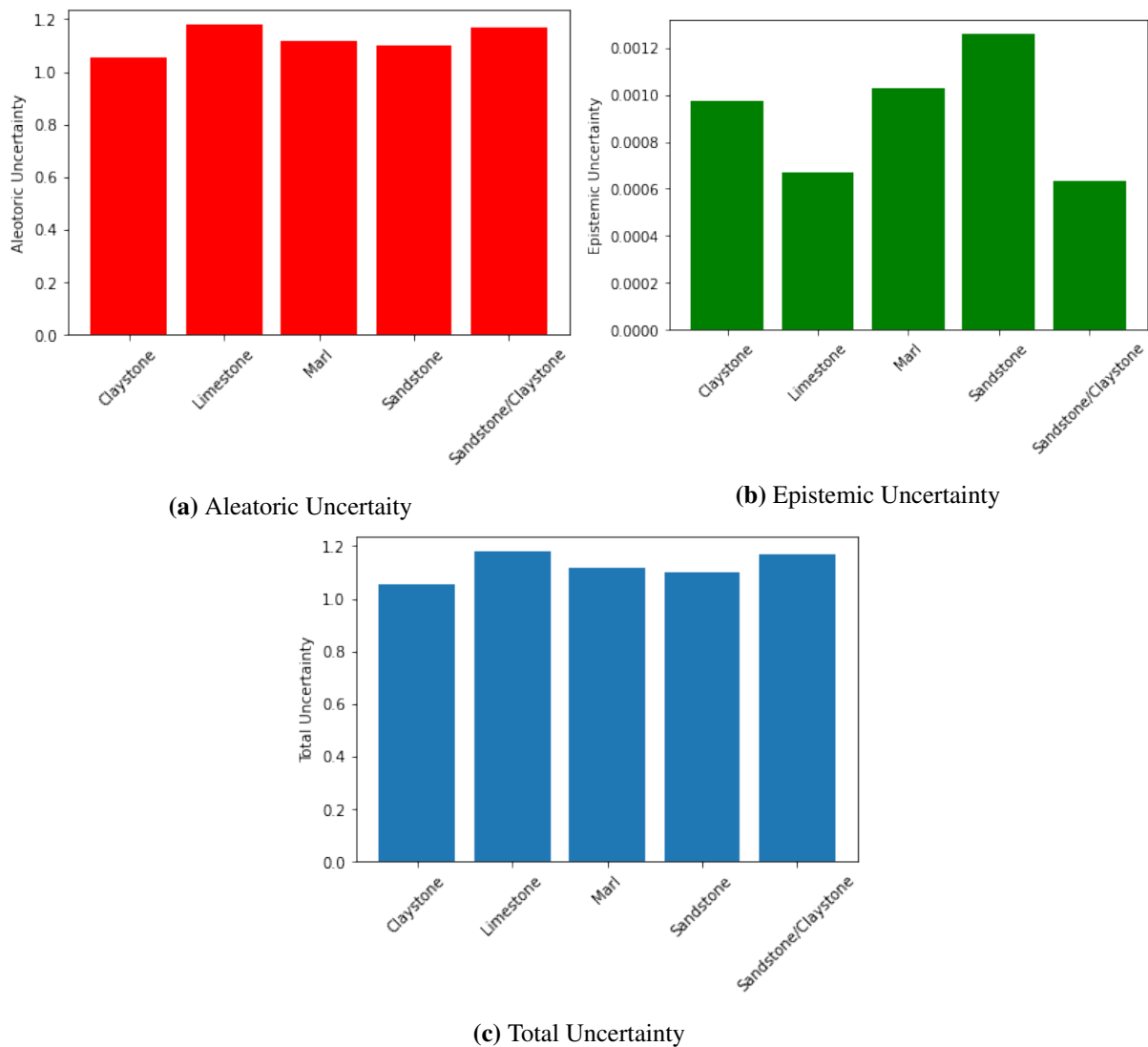


Figure 6.11: BNN Uncertainties based on Random Forest's probabilities score for different classes

Epistemic uncertainty, also called systematic uncertainty, refers to uncertainty arising from a lack of knowledge, that is, the agent's epistemic state.

In contrast to aleatoric uncertainty, epistemic uncertainty can, in principle, be reduced with additional information.

With respect to epistemic uncertainty, the Random Forest algorithm is seen to be more reliable but produces less trustworthy predictions since aleatoric uncertainty is higher.

Mean Uncertainty	Random Forest	Adaptive Boosting
Aleatoric	1.12	0.72
Epistemic	0.0008	0.001

Table 6.1: Aleatoric and epistemic uncertainties for the Random Forest and Adaptive Boosting algorithms.

6.4.3 Probability Calibration

Probability calibration is the process of calibrating an ML model to return the true likelihood of an event. This is necessary when one needs the probability of the event in question rather than its classification, which is extremely important while conducting the model uncertainties evaluation. The theory behind the calibration method is comprehensively described in section 3.4.2.

Ideally, the calibration plot should represent the diagonal line, reflecting that each fraction of positive predictions is perfectly predicted by the model as shown on the Figure 6.12.

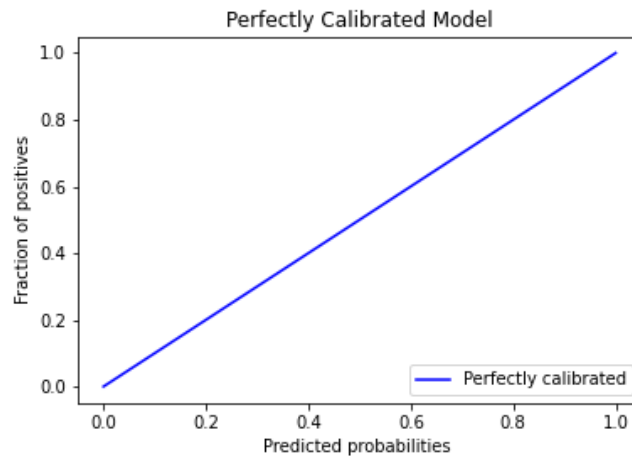


Figure 6.12: Perfectly Calibrated Model.

The Pytolemaic package provides the algorithm for model calibration.

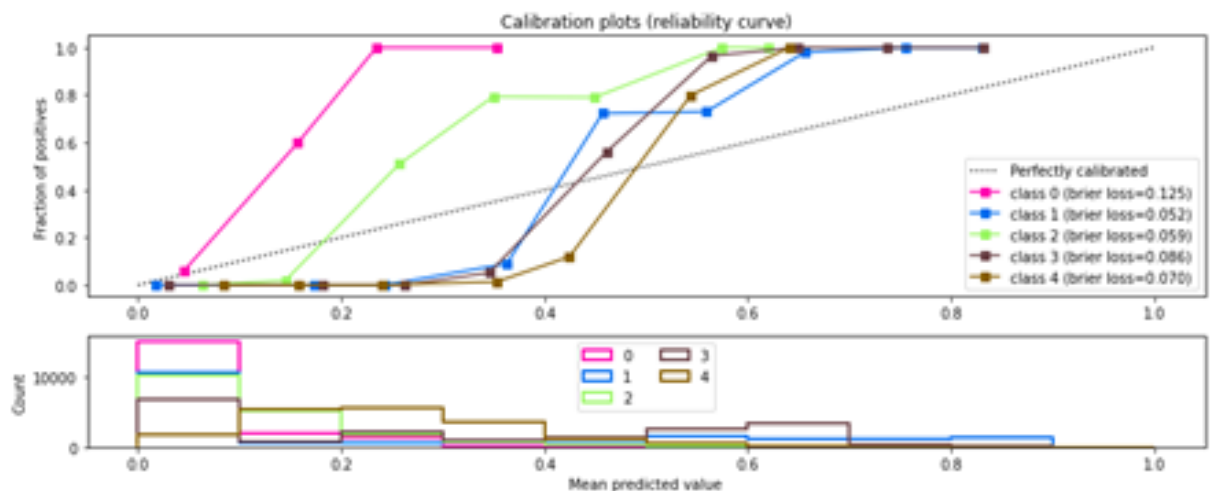


Figure 6.13: Calibration plot for the Random Forest algorithm.

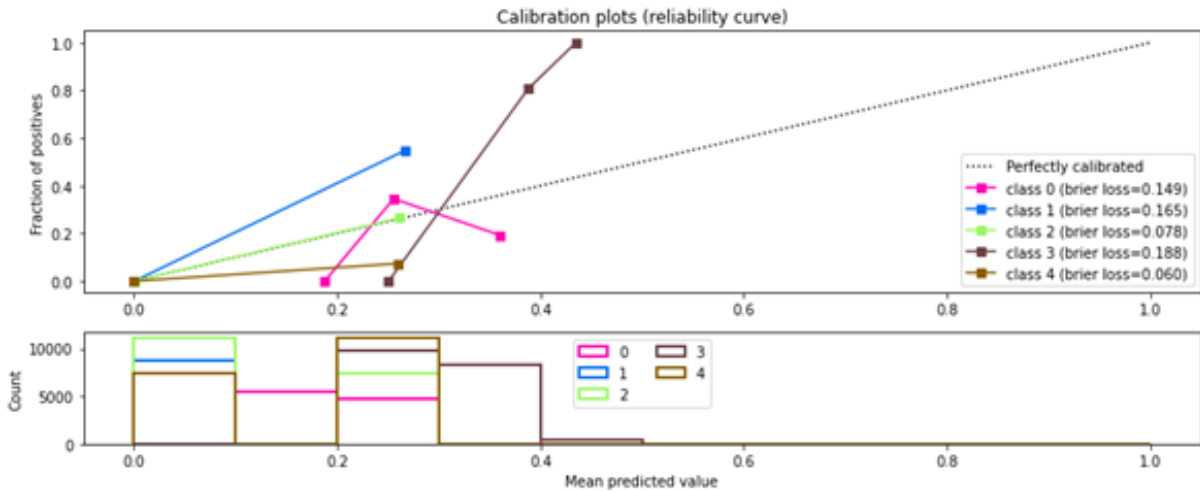


Figure 6.14: Calibration plot for the Adaptive Boosting algorithm.

Models are not well calibrated. Recalibration methods, such as Isotonic regression and Platt-scaling methods, could be applied to achieve better calibration. These methods are described in section 3.4.2. After running several tests, the Isotonic method provided better results for recalibration. ClassificationCalibration in the UQ360 framework was used to run the recalibration process.

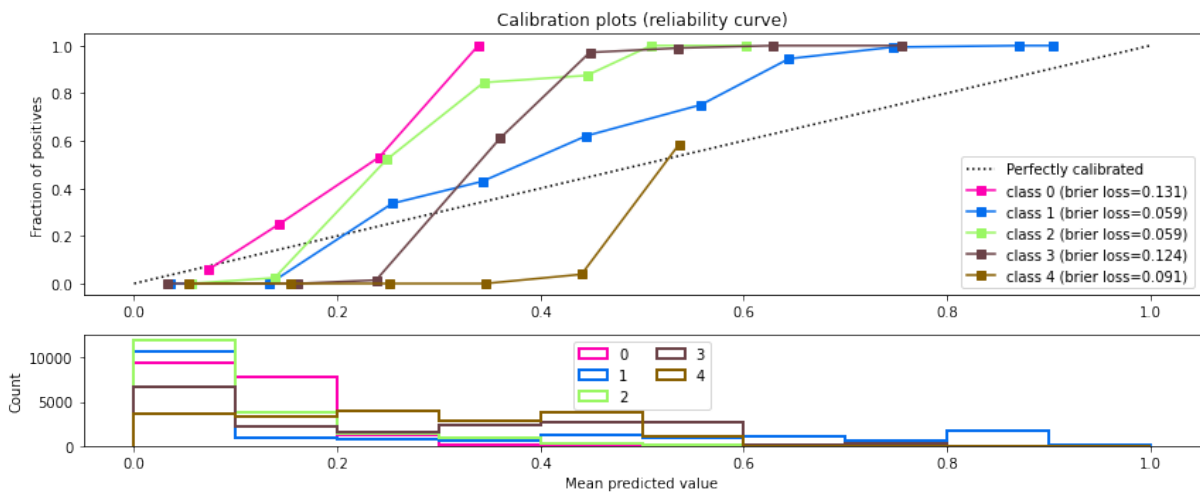


Figure 6.15: Recalibration plot for the Random Forest algorithm.

Usually, the average confidence for a bin lies on or close to the diagonal. The calibration analysis shows how confident the model's predictions are. The model is overconfident if confidence is higher than accuracy (below the diagonal line). Otherwise, the model is underconfident.

It is difficult to conclude the prediction confidence visually in a multiclassification problem. The gap between accuracy and confidence is calculated using the formula below. The formula

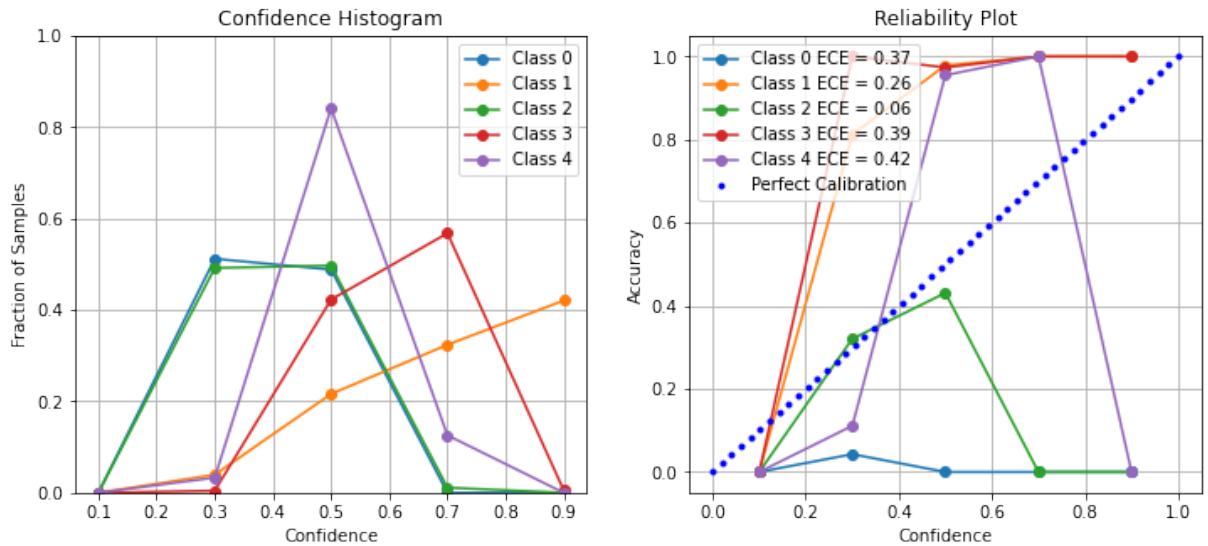


Figure 6.16: Confidence Histogram and Reliability plot for each class (Random Forest algorithm).

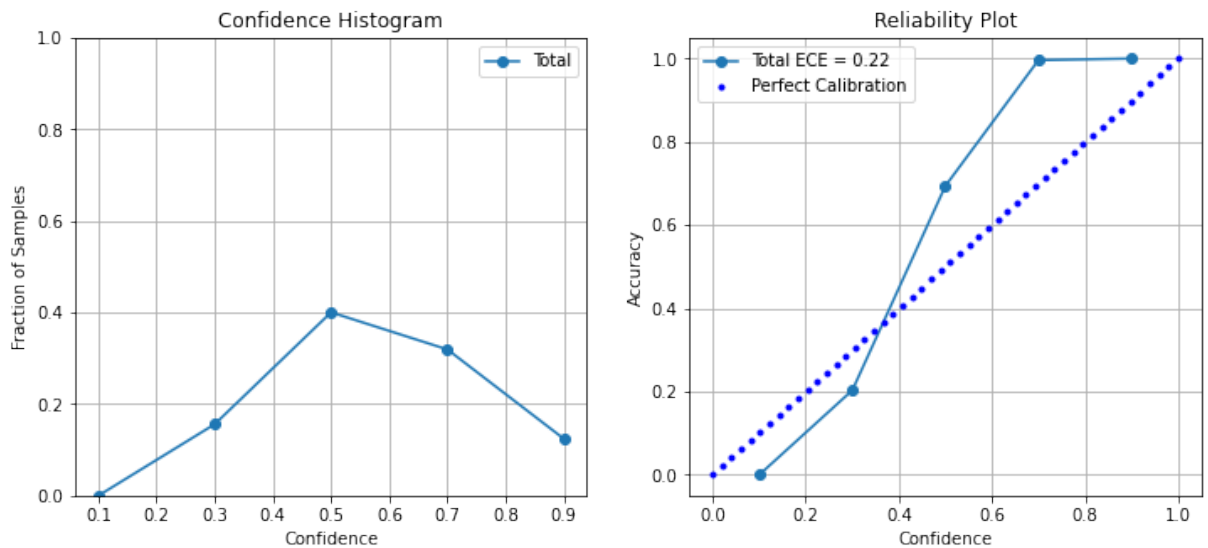


Figure 6.17: Confidence Histogram and Reliability plot - Average (Random Forest algorithm).

is derived based on results from the Reliability plot and Confidence Histogram.

$$Accuracy-ConfidenceGap [ibins] = \sum (Accuracy [i] - Confidence [i]) * Fraction of samples [i] \quad (6.1)$$

The chosen Accuracy - Confidence Gap threshold to prove the model confidence is in the range $[-0.25, 0.25]$. Lower than -0.25 : the model is overconfident. Higher than 0.25 : underconfident.

What do over-confidence and under-confidence mean?

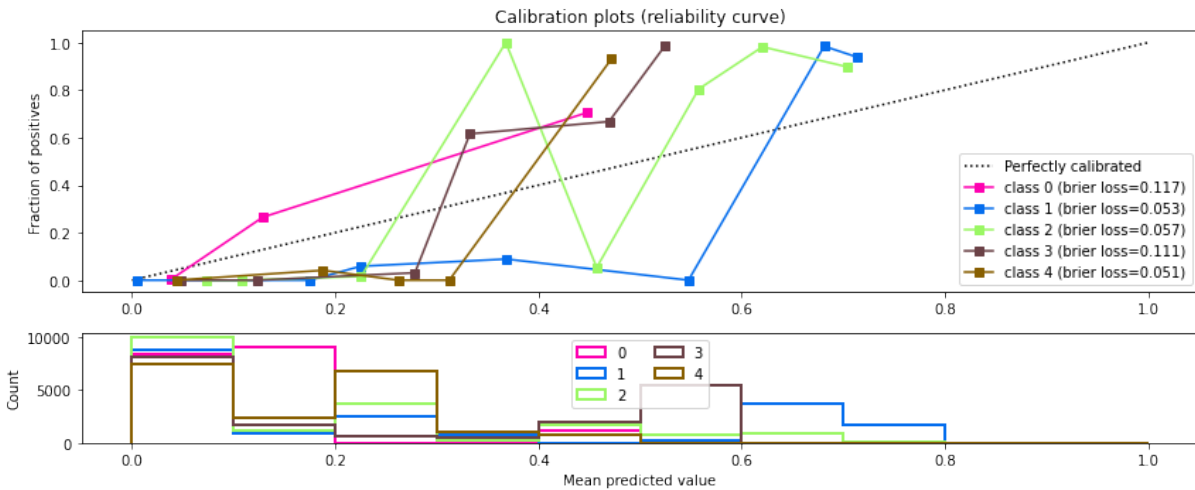


Figure 6.18: Recalibration plot for the Adaptive Boosting algorithm.

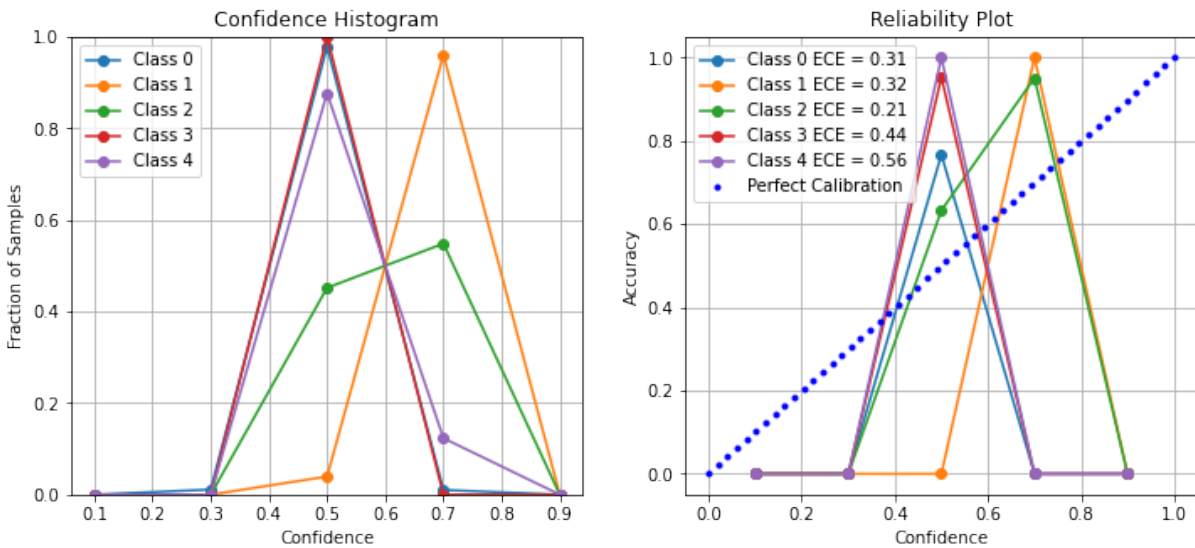


Figure 6.19: Confidence Histogram and Reliability plot for each class (Adaptive Boosting algorithm).

Over-confidence (confident > accuracy): model gives more false positives, it provides observations that are not actually true. If the threshold for distinguishing between classes is too low, this could occur.

Under-confidence (accuracy > confidence): model gives more false negatives. The decision threshold is high. There should be a small gap between accuracy and confidence as long as the aim is to interpret predictions as probabilities. In our case, the Adaptive Boosting algorithm could have been improved by decreasing the decision threshold.

Table 6.2 shows that the Random Forest is more trustful than Adaptive Boosting in terms

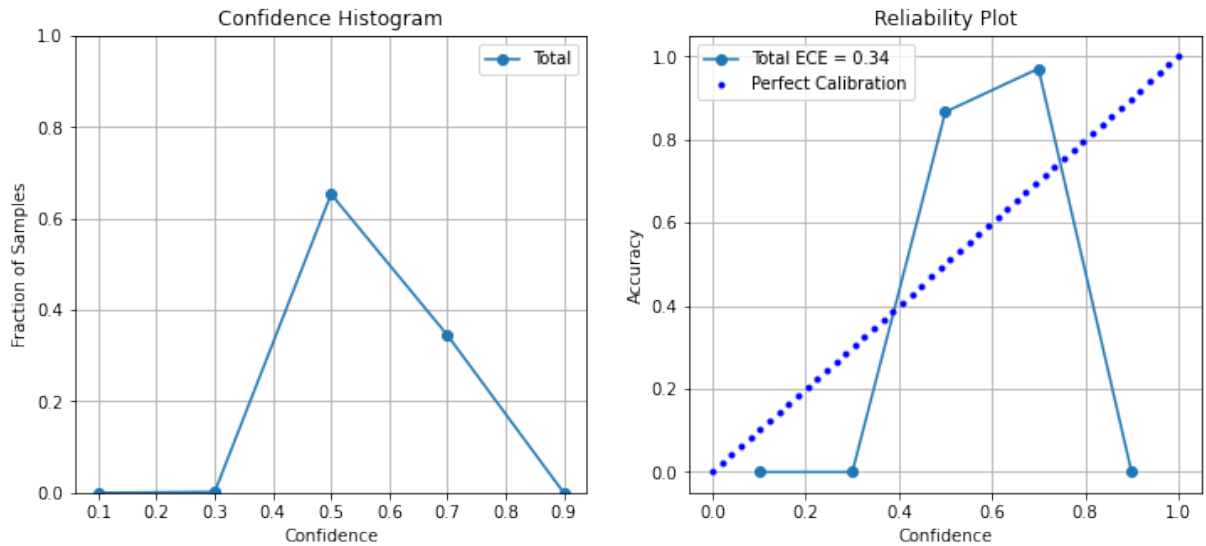


Figure 6.20: Confidence Histogram and Reliability plot - Average (Adaptive Boosting algorithm).

	Random Forest			Adaptive Boosting		
	Average confidence	Confidence vs Accuracy	Result	Average confidence	Confidence vs Accuracy	Result
Claystone	0.39	-0.1	Confident	0.5	0.28	Underconfident
Limestone	0.72	0.27	Underconfident	0.68	0.27	Underconfident
Mark	0.4	-0.04	Confident	0.62	0.22	Confident
Sandstone	0.62	0.38	Underconfident	0.5	0.48	Underconfident
Sandstone/Claystone	0.55	0.42	Underconfident	0.53	0.42	Underconfident
Overall (Average)	0.57	0.16	Confident	0.58	0.32	Underconfident

Table 6.2: The results after calculation the Accuracy - Confidence gap for Random Forest and Adaptive Boosting algorithms.

of confidence notation. This is one of the most discoveries that even though Adaptive Boosting produced a better result in terms of accuracy for the tested well, the Random Forest is proven to be more reliable in a case if the trained model would be tested on other wells. The Random Forest and Adaptive Boosting algorithms were tested on other wells from the Volve dataset to prove this statement.

Before running the models, the tested well were removed from the training dataset. To keep the training dataset almost unchanged, the tested well F-5, on which the model and uncertainty analysis was based, was not added to the dataset.

The results of the testing experiments are presented in Appendix B. The experiment has proven that the Random Forest algorithm performs better for other testing datasets (Table 6.3). Therefore, even though the Adaptive Boosting demonstrated the high accuracy for one tested

	Accuracy	
Tested well	Random Forest	Adaptive Boosting
F-15	0.85	0.75
F-15S	0.64	0.38
F-14	0.33	0.23

Table 6.3: Accuracy scores for the wells: F-15, F-15S, F-14

well, if the trained model is tested on the new well, it is more likely to fail. In conclusion, the discovery demonstrated that it is crucial to conduct the model performance and uncertainty quantification before deploying the model in real life applications.

Chapter 7

Conclusions and Future Work

Conclusion

This thesis aimed to investigate to what extent ML techniques can use the preprocessed real-time drilling data from the Volve field as input to label the lithological properties of the formations. To achieve this, seven different supervised algorithms were trained on one selected well with the minority of the samples presented in the dataset.

One of the conclusions that can be drawn from the thesis is that supervised ML algorithms can classify lithology classes with relatively high accuracy.

The two best performing models were the Random Forest algorithm with an accuracy of 72% and the Adaptive Boosting one, reaching an accuracy of 90%. Hyperparameter tuning was applied to improve the algorithms' predictive power. Sequentially, these two algorithms were evaluated, and uncertainties were quantified. From the Model Vulnerability analysis, it is observed that the Adaptive Boosting algorithm is prone to the Data Leakage, meaning that some of the attributes from the testing dataset were implemented in the model training phase. Another drawback of the Adaptive Boosting algorithm is that it is not sensitive to most features.

Both algorithms were affected by the Covariate Data Shift, which referred to the difference between training and testing datasets distributions resulting in the less predictive power of the algorithms. The data shift issue should be eliminated in the data preprocessing phase.

The uncertainty quantification (UQ) consisted of the Blackbox Metamodel approach, Bayesian Neural Networks (BNNs), and Probability Calibration. Metamodeling measured the aleatoric

or data uncertainty, demonstrating that the Random Forest algorithm is more uncertain than the Adaptive Boosting one. The problem arises from the nature of testing data nature.

BNNs were utilized to quantify the aleatoric and epistemic uncertainties. After running the algorithm based on the predicted class probability score, the Random Forest algorithm is more reliable regarding the model uncertainty than the other algorithm.

To enhance the quality of UQ, Probability Calibration returns the predicted class's actual likelihood. This technique is necessary when the probability of the prediction is more crucial to consider than the classification results.

After comparing the calibration results and computing the Accuracy - Confidence based on Confidence Histogram and Reliability Plot, the most important discovery was that the less accurate algorithm, according to the scoring classification report, is more confident than the best-performed algorithm.

When training the model on the selected well, there is no evident risk that the trained model would not perform satisfactorily on the other testing wells. In a real-life application, the before-hand trained model would be tested on the obtained data from the newly drilled wells. After running the trained algorithms on the other available wells in the Volve dataset, it was proven that the Random Forest algorithm is more reliable due to its higher confidence in the predictions.

Future work

The thesis opens up several areas for further research. The supervised classification algorithms applied to the Volve field dataset considered in Model Analysis Case Study could be improved by the proposed methods, including eliminating the Data Leakage and Covariate Data Shift.

A more advanced intrinsic UQ technique, the Gaussian Process Regression (GPR) algorithm, which is similar to BNN, is suggested to quantify the uncertainties. GPR is an exceptionally effective ML algorithm that, in contrast to many of today's UQ techniques, estimates uncertainties with few parameters required. The algorithm is incorporated in the UQ360 toolkit, and as stated in the documentation, it is better to handle relatively small datasets.

The extrinsic method, Infinitesimal Jackknife, is recommended to improve UQ quality. A closed-form Gaussian distribution can characterize this standard statistical technique for con-

structuring a pseudo-ensemble without retraining. The approach approximates the impact of changes to training data on the model's predictions (Lu et al., 2020) [33].

The next step in the current research is to train and analyze the sequential ML approach, called Recurrent Neural Networks (RNNs). A recurrent neural network (RNN) is an artificial neural network that employs sequential or time-series data. The notion of 'memory' enables RNNs to retain the states or data of prior inputs to construct the subsequent output in a series. This algorithm would allow predicting lithology classes based on previously recorded parameters in a drilled wellbore.

References

- [1] Shib Ganguli and Souvik Sen. Estimation of pore pressure and fracture gradient in volve field, norwegian north sea. *SPE Journal*, 04 2019.
- [2] Pdraig Cunningham and Sarah Delany. k-nearest neighbour classifiers. *Mult Classif Syst*, 54, 04 2007.
- [3] Nikolaos Sapountzoglou, Jesus Lago, and Bertrand Raison. Fault diagnosis in low voltage smart distribution grids using gradient boosting trees. *Electric Power Systems Research*, 182:106254, 05 2020.
- [4] Alberto Fernández, Salvador García, Mikel Galar, Ronaldo Prati, Bartosz Krawczyk, and Francisco Herrera. *Learning from Imbalanced Data Sets*. 01 2018.
- [5] Ines Dedovic. *Efficient probability distribution function estimation for energy based image segmentation methods*. PhD thesis, 02 2017.
- [6] Francisco Herrera. Dataset shift in classification: Approaches and problems. <https://www.inf.ufpr.br/lesoliveira/sticamsud/fr14-simon.pdf>, 2011. Accessed: 2022-05-11.
- [7] Marco Túlio Ribeiro, Sameer Singh, and Carlos Guestrin. "why should I trust you?": Explaining the predictions of any classifier. *CoRR*, abs/1602.04938, 2016.
- [8] Soumya Shubhra Ghosh, Qingzi Vera Liao, Karthikeyan Natesan Ramamurthy, Jirí Navrátil, Prasanna Sattigeri, Kush R. Varshney, and Yunfeng Zhang. Uncertainty quantification 360: A holistic toolkit for quantifying and communicating the uncertainty of ai. *ArXiv*, abs/2106.01410, 2021.
- [9] Jiri Navratil, Matthew Arnold, and Benjamin Elder. Uncertainty prediction for deep sequential regression using meta models, 07 2020.
- [10] Laurent Valentin Jospin, Wray L. Buntine, Farid Boussaïd, Hamid Laga, and Mohammed Bennamoun. Hands-on bayesian neural networks - a tutorial for deep learning users. *CoRR*, abs/2007.06823, 2020.
- [11] Why you should use bayesian neural network. <https://towardsdatascience.com/why-you-should-use-bayesian-neural-network-aaf76732c150>. Accessed: 2022-04-24.
- [12] Uncertainty quantification 360. <https://uq360.mybluemix.net/>. Accessed: 2022-05-05.
- [13] Ngoc Tran Deepak Devegowda Vikram Jayaram Chandra Rai Carl Sondergeld Gupta, Ishank and Hamidreza Karami. Looking ahead of the bit using surface drilling and petrophysical data: Machine-learning-based real-time geosteering in volve field.
- [14] Andrzej Tunkiel, Tomasz Wiktorski, and Dan Sui. Drilling dataset exploration, processing and interpretation using volve field data. 08 2020.
- [15] Trevor Hastie Robert Tibshirani Gareth James, Daniela Witten. *An Introduction to Statistical Learning with applications in R*. Springer, 2013.
- [16] Johannes Fürnkranz. *Decision Tree*, pages 263–267. Springer US, Boston, MA, 2010.
- [17] Giorgio Valentini and Francesco Masulli. Ensembles of learning machines. volume 2486, pages 3–22, 05 2002.
- [18] Simon Bernard. Random forests - parametrization and dynamic induction. <https://www.inf.ufpr.br/lesoliveira/sticamsud/fr14-simon.pdf>, 2014. Accessed: 2022-05-11.
- [19] Yoav Freund and Robert E. Schapire. A desicion-theoretic generalization of on-line learning and an application to boosting. In Paul Vitányi, editor, *Computational Learning Theory*, pages 23–37, Berlin, Heidelberg, 1995. Springer Berlin Heidelberg.

- [20] Alexey Natekin and Alois Knoll. Gradient boosting machines, a tutorial. *Frontiers in Neurorobotics*, 7, 2013.
- [21] Understanding gradient boosting from scratch with a small dataset. <https://towardsdatascience.com/understanding-gradient-boosting-from-scratch-with-small-dataset-587592cc871f>. Accessed: 2022-04-20.
- [22] Geoffrey I. Webb. *Naïve Bayes*, pages 713–714. Springer US, Boston, MA, 2010.
- [23] Sotiris Kotsiantis, D. Kanellopoulos, and P. Pintelas. Handling imbalanced datasets: A review. *GESTS International Transactions on Computer Science and Engineering*, 30:25–36, 11 2005.
- [24] Pytolemaic — a toolbox for model quality. <https://towardsdatascience.com/pytolemaic-package-for-model-quality-analysis-2b7bea751cfd>. Accessed: 2022-03-04.
- [25] Lime: explain machine learning predictions. <https://towardsdatascience.com/lime-explain-machine-learning-predictions-af8f18189bfe>. Accessed: 2022-04-04.
- [26] John Platt. 1999, month = 06, pages = , title = Probabilistic Outputs for Support Vector Machines and Comparisons to Regularized Likelihood Methods, volume = 10, journal = Adv. Large Margin Classif.
- [27] Bianca Zadrozny and Charles Elkan. Obtaining calibrated probability estimates from decision trees and naive bayesian classifiers. *ICML*, 1, 05 2001.
- [28] Interquartile range, statistic how to. <https://www.statisticshowto.datasciencecentral.com/probability-and-statistics/interquartile-range/>, note = Accessed: 2022-04-20.
- [29] Jianyu Miao and Lingfeng Niu. A survey on feature selection. *Procedia Computer Science*, 91:919–926, 12 2016.
- [30] Nonso Nnamoko, Farath Arshad, David England, Jiten Vora, and James Norman. Evaluation of filter and wrapper methods for feature selection in supervised machine learning. 06 2014.
- [31] Pytolemaic Github Repository, <https://github.com/broundal/pytolemaic>.
- [32] Malik Nadeem, Jean-daniel Zucker, and Blaise Hanczar. Accuracy-rejection curves (arcs) for comparing classification methods with a reject option. *Journal of Machine Learning Research - Proceedings Track*, 8:65–81, 01 2010.
- [33] Zhiyun Lu, Eugene Ie, and Fei Sha. Uncertainty estimation with infinitesimal jackknife, its distribution and mean-field approximation. *CoRR*, abs/2006.07584, 2020.
- [34] SLEIPNER PETEK. Discovery Evaluation Report Well 15/9-19 SR Theta Vest Structure, Dec 1993.
- [35] Ajay Kulkarni, Deri Chong, and Feras A. Batarseh. 5 - foundations of data imbalance and solutions for a data democracy. In Feras A. Batarseh and Ruixin Yang, editors, *Data Democracy*, pages 83–106. Academic Press, 2020.
- [36] Thomas M. Cover and Peter E. Hart. Nearest neighbor pattern classification. *IEEE Trans. Inf. Theory*, 13:21–27, 1967.

Appendices

Appendix A

Python Code

A.1 Installed Packages

The packages required to run the codes and visualize results (Table A.1).

A.2 Decision Tree

```
1 from sklearn.metrics import classification_report
2 from sklearn.tree import DecisionTreeClassifier
3
4 # Create Decision Tree classifier object
5 # clf = DecisionTreeClassifier(max_depth=10, min_samples_leaf=20,
6 # criterion = "gini")
7 clf = DecisionTreeClassifier(class_weight=class_weights)
8
9 # Train Decision Tree Classifier
10 clf = clf.fit(X_train, y_train)
11
12 #Predict the response for test dataset
13 y_pred_dt = clf.predict(X_test)
14
15 print('Decision Tree Report \n')
16 print(classification_report(y_test, y_pred_dt))
```

Listing A.1: Decision Tree algorithm

Package	Version
aif360	0.4.0
imbalanced-learn	0.8.1
imblearn	0
matplotlib	3.2.2
matplotlib-inline	0.1.3
numpy	1.21.6
pandas	1.3.5
plotly	5.5.0
pytolemaic	0.14.1
scikit-learn	1.0.2
scipy	1.4.1
seaborn	0.11.2
sklearn	0.0
tensorflow	2.8.2
uq360	0.2

Table A.1: Required packages.

A.3 Random Forest

```

1
2 from sklearn.metrics import classification_report
3 from sklearn.tree import Random ForestClassifier
4
5 # Create Random Forest classifier object
6 clf=RandomForestClassifier(class_weight=class_weights , max_depth =
7     100, min_samples_leaf = 5, n_estimators = 100)
8
9 # Train Random Forest Classifier y_pred=clf.predict(X_test)
10 clf.fit(X_train , y_train)
11
12 #Predict the response for test dataset
13 y_pred_rf=clf.predict(X_test)
14
15 print('Random Forest Report \n')
16 print(classification_report(y_test , y_pred_rf))

```

Listing A.2: Random Forest algorithm

A.4 Gradient Boosting

```

1 from sklearn.metrics import classification_report
2 from sklearn.tree import GradientBoostingClassifier

```

```
3
4 # Create Gradient Boosting classifier object
5 grad = GradientBoostingClassifier(n_estimators = 50, max_depth = 7,
   learning_rate = 0.1)
6
7 # Train Gradient Boosting Classifier
8 model = grad.fit(X_train, y_train)
9
10 #Predict the response for test dataset
11 y_pred_grad = model.predict(X_test)
12
13 print('Gradient Boost Report \n')
14 print(classification_report(y_test, y_pred_grad))
15
```

Listing A.3: Gradient Boosting algorithm

A.5 Adaptive Boosting

```
1 from sklearn.metrics import classification_report
2 from sklearn.tree import AdaBoostClassifier
3
4 # Create adaboost classifier object
5 abc = AdaBoostClassifier(algorithm = 'SAMME', learning_rate = 1.089,
   n_estimators = 60)
6
7 # Train Adaboost Classifier
8 model = abc.fit(X_train, y_train)
9
10 #Predict the response for test dataset
11 y_pred_ad = model.predict(X_test)
12
13 print('AdaBoost Classifier Report \n')
14 print(classification_report(y_test, y_pred_ad))
15
```

Listing A.4: Adaptive Boosting algorithm

A.6 Model Analysis

```
1
2 from pytolemaic import Metrics
3 from pytolemaic import PyTrust
4 from pytolemaic.utils.general import GeneralUtils
5
6 def run():
```

```

7
8     # Train estimator
9     estimator = RandomForestClassifier(class_weight=class_weights ,
10    max_depth = 100, min_samples_leaf = 5, n_estimators = 100)
11    estimator.fit(X_train , y_train)
12
13    # Initiating PyTrust
14    pytrust = PyTrust(
15        model=estimator ,
16        xtrain=X_train , ytrain=y_train ,
17        xtest=X_test , ytest=y_test)
18
19    # Initiating PyTrust with more information
20    pytrust = PyTrust(
21        model=estimator ,
22        xtrain=X_train , ytrain=y_train ,
23        xtest=X_test , ytest=y_test ,
24        feature_names=feature_names ,
25        target_labels=labels)
26
27    pytrust.scoring_report.plot()
28
29    pytrust.sensitivity_report.plot()
30
31    pytrust.dataset_analysis_report.plot()
32
33    pytrust.quality_report.plot()
34
35    if __name__ == '__main__':
36        run()
37        plt.show()

```

Listing A.5: Pytolemaic Code to run Model Evaluation for the Random Forest algorithm

```

1
2    from pytolemaic import Metrics
3    from pytolemaic import PyTrust
4    from pytolemaic.utils.general import GeneralUtils
5
6    def run():
7
8        # Train estimator
9        estimator = AdaBoostClassifier(algorithm = 'SAMME', learning_rate
10    = 1.089, n_estimators = 60)
11    estimator.fit(X_train , y_train)
12
13    # Initiating PyTrust
14    pytrust = PyTrust(
15        model=estimator ,
16        xtrain=X_train , ytrain=y_train ,
17        xtest=X_test , ytest=y_test)

```

```

17
18 # Initiating PyTrust with more information
19 pytrust = PyTrust(
20     model=estimator ,
21     xtrain=X_train , ytrain=y_train ,
22     xtest=X_test , ytest=y_test ,
23     feature_names=feature_names ,
24     target_labels=labels)
25
26 pytrust.scoring_report.plot()
27
28 pytrust.sensitivity_report.plot()
29
30 pytrust.dataset_analysis_report.plot()
31
32 pytrust.quality_report.plot()
33
34 if __name__ == '__main__':
35     run()
36     plt.show()
37

```

Listing A.6: Pytolemaic Code to run Model Evaluation for the Adaptive Boosting algorithm

A.7 Blackbox Metamodel

```

1
2 from uq360.algorithms.blackbox_metamodel import
   MetamodelClassification
3
4 # split the training partition to provide base and meta training sets
5 X_train_base , X_train_meta , y_train_base , y_train_meta =
   train_test_split(X_train , y_train , test_size=0.4 , random_state=42)
6
7 # Simulate a pre-existing , pre-trained base model
8 clf_md1 = RandomForestClassifier() #or AdaBoostClassifier()
9
10 base_config = {} #according to the hyperparameters of the algorithm
11
12 meta_config = {}
13
14 meta_md1 = GradientBoostingClassifier()
15 uq_model = MetamodelClassification(base_model=clf_md1 , meta_model=
   meta_md1 , base_config=base_config , meta_config=None)
16
17 # now fit the meta model only
18 _ = uq_model.fit(X=None , y=None , base_is_prefitted=True ,
   meta_train_data=(X_train_meta , y_train_meta))
19

```

```

20 y_test_pred , y_test_score = uq_model.predict(X_test)
21

```

Listing A.7: BlackBox Metamodel code

A.8 Isotonic Regression Recalibration

```

1
2 from uq360.algorithms.classification_calibration import
   ClassificationCalibration
3
4 #Initialize the calibration function
5 calib = ClassificationCalibration(num_classes = 5, fit_mode='probs',
   method='isotonic', base_model_prediction_func=None)
6
7 #Class probability score from the pretrained model clf
8 score=calibrated_clf.predict_proba(X_test)
9
10 #Score output after recalibration
11 _ = calib.fit(score , y_test)
12 pred , score_iso = _.predict(score)
13

```

Listing A.8: Isotonic Regression Recalibration code

A.9 BNN

```

1
2 from uq360.algorithms.variational_bayesian_neural_networks.bnn import
   BnnClassification
3
4 import torch
5 import torch.nn as nn
6 import torch.nn.functional as F
7 import torch.optim as optim
8 from torch.utils.data import DataLoader
9 import torch.utils.data as data_utils
10
11 #Fit test data to the Tensor array
12 test = data_utils.TensorDataset(torch.Tensor(y_test_score), torch.
   Tensor(y_test.values))
13 test_loader = data_utils.DataLoader(test , batch_size=1, shuffle=True)
14
15 #Class probability score from the pretrained model clf_md1
16 y_test_score = clf_md1.predict_proba(X_test)
17

```



```

18 #BNN congifuration settings
19 config = {"ip_dim": y_test_score.shape[1], "op_dim": 5, "num_nodes":
128, "num_layers": 1,
20           "num_epochs": 5, "step_size": 0.001}
21 config['hshoe_scale'] = 1e-1
22 config['use_reg_hshoe']=True
23 device = torch.device("cuda:0" if torch.cuda.is_available() else "cpu
")
24
25 #Train BNN
26 bnn = BnnClassification(config=config, device=device, prior="RegHshoe
")
27
28 #Fit BNN algorithm with probability class score and tested targets
29 bnn = bnn.fit(X=y_test_score, y=y_test.values)
30
31 #BNN outputs
32 def get_test_results(test_loader, bnn):
33     correct = 0
34     total = 0
35
36     all_mean = []
37     all_var = []
38     all_true_labels = []
39     all_pred_labels = []
40
41     all_total = []
42     all_epistemic = []
43     all_aleotoric = []
44
45     for test_batch_x, test_batch_y in test_loader:
46
47         with torch.no_grad():
48             predicted, pred_mean, pred_var, y_prob_samples = bnn.
predict(test_batch_x)
49             pred_total_uq, pred_aleo, pred_epi =
entropy_based_uncertainty_decomposition(y_prob_samples)
50
51             all_mean.append(pred_mean)
52             all_var.append(pred_var)
53             all_true_labels.append(test_batch_y)
54             all_pred_labels.append(predicted)
55
56             all_total.append(pred_total_uq)
57             all_epistemic.append(pred_epi)
58             all_aleotoric.append(pred_aleo)
59
60             total += test_batch_y.size(0)
61             correct += (predicted == test_batch_y.numpy()).sum()
62     print("accuracy: %d %%" % (100 * correct / total))
63

```

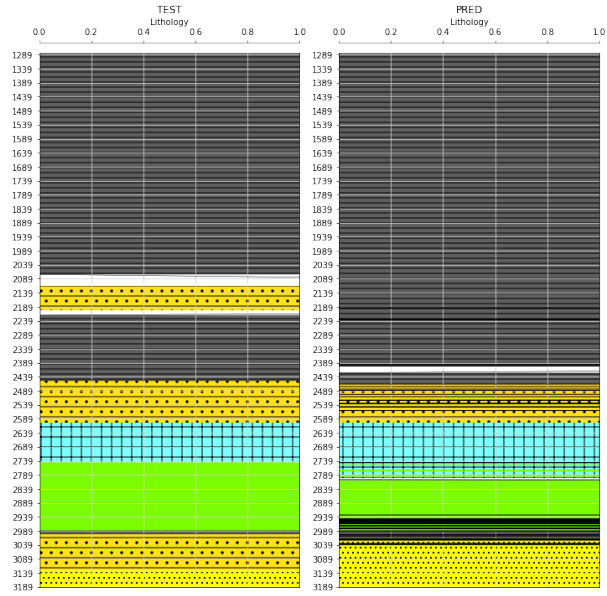
```
64 all_mean_mat = np.concatenate(all_mean, axis=0)
65 all_true_labels_mat = np.concatenate(all_true_labels, axis=0)
66 all_pred_labels_mat = np.concatenate(all_pred_labels, axis=0)
67
68 all_total_mat = np.concatenate(all_total, axis=0)
69 all_epi_mat = np.concatenate(all_epistemic, axis=0)
70 all_aleo_mat = np.concatenate(all_aleotoric, axis=0)
71
72 return all_mean_mat, all_true_labels_mat, all_pred_labels_mat,
73 all_total_mat, all_epi_mat, all_aleo_mat,
74
```

Listing A.9: BNN code

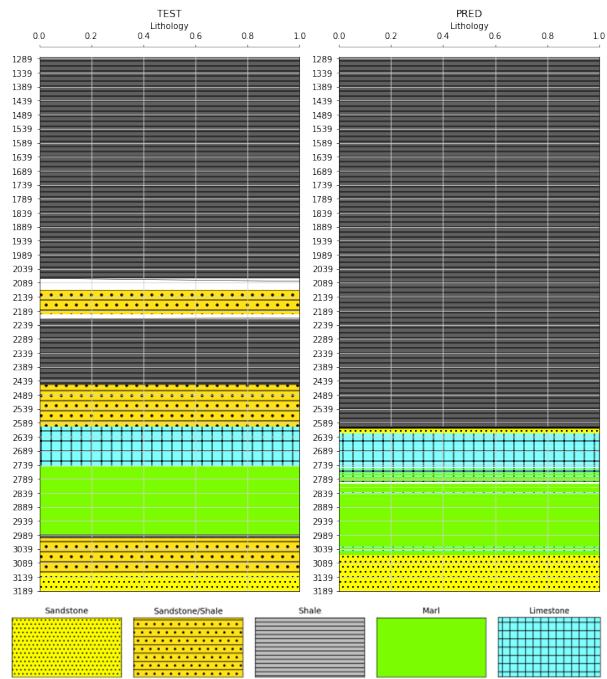
Appendix B

Lithology columns for the tested wells

F-14, F-15, F-15S

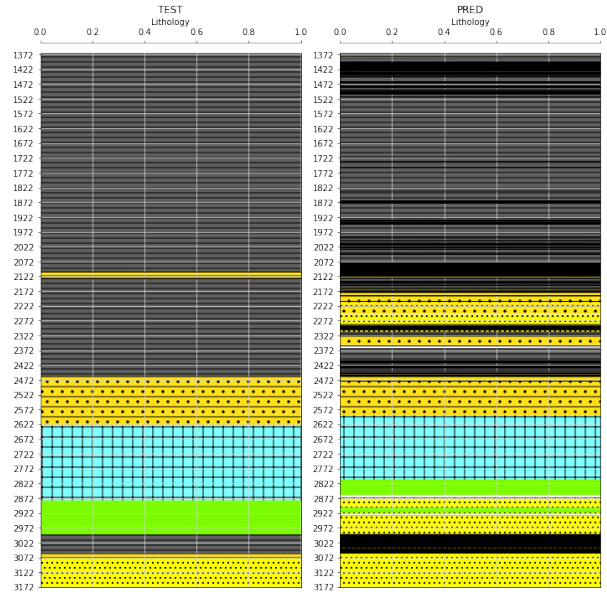


(a) Random Forest Classifier

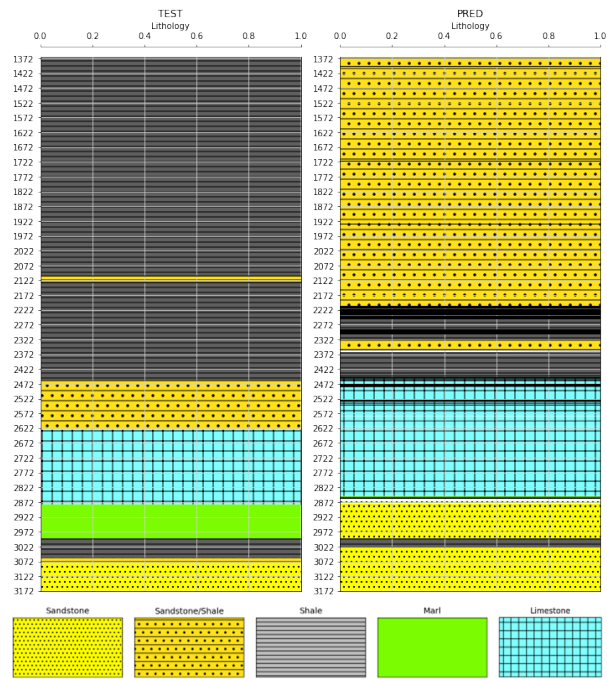


(b) Adaptive Boosting Classifier

Figure B.1: Lithology columns of tested well F15 and predicted outcomes

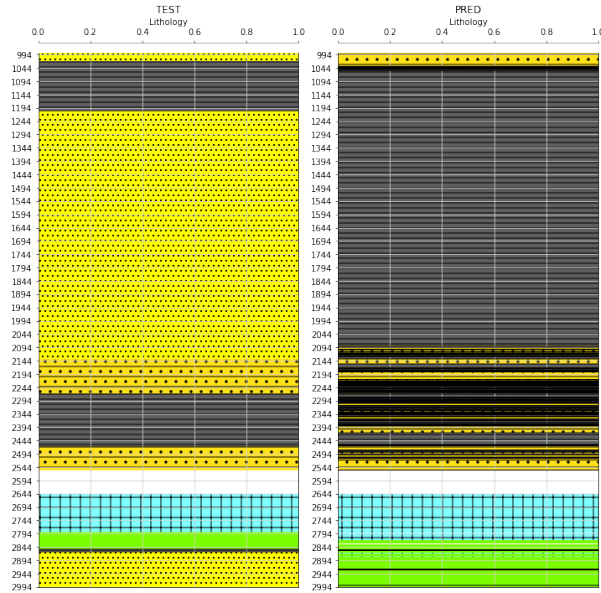


(a) Random Forest Classifier

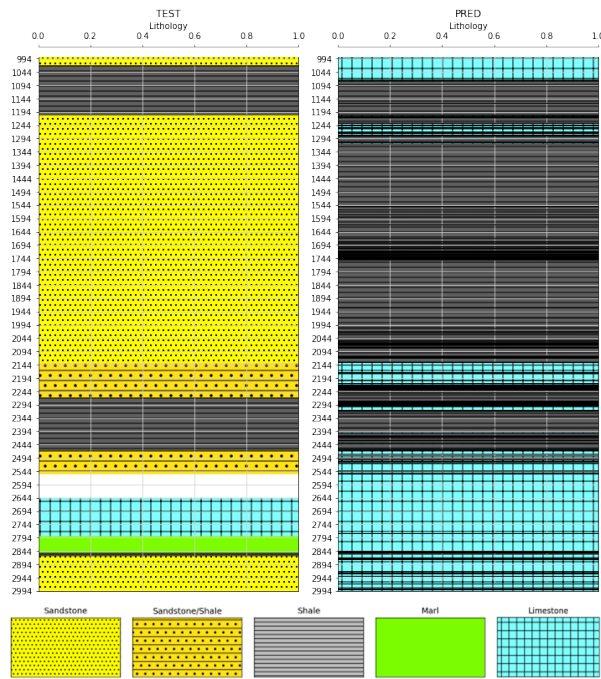


(b) Adaptive Boosting Classifier

Figure B.2: Lithology columns of tested well F15S and predicted outcomes



(a) Random Forest Classifier



(b) Adaptive Boosting Classifier

Figure B.3: Lithology columns of tested well F14 and predicted outcomes