University of Stavanger

Faculty of Science and Technology
Department of Electrical Engineering and Computer Science

Use of machine learning algorithms to predict well barrier elements and envelopes in drilling operations

Master's Thesis in Applied Data Science
by

Daiana dos Anjos

Internal Supervisors

Antorweep Chakravorty

August 17, 2021

"I am not young enough to know everything"

Oscar Wilde

Abstract

Digitalization is a concept that have been widely approached in the oil and gas industry in the recent years. The amount of information generated in this industry is vast and the operation requirements are strict. However, the access and management of the information is still a challenge. Many tasks when planning and executing a drilling and well operation are still performed manually by the drilling and well team responsible for the field in development.

The idea proposed in this thesis is the automation of a time consuming task required on the well construction process: define the well barrier element (WBE) and well barrier schematics (WBS) for drilling operations. The idea is to explore the use of text based machine learning classification techniques applied to the text information obtained from previously available well barrier schematics from drilling operations.

Although there are software available on the market that are capable of auto-generate well barrier schematics, this application of machine learning algorithms is believed to be the first attempt. No previous studies have been identified with a similar approach. Consequently, an important contribution of this work is the database, most likely the first open database in a format that facilitate machine learning application in this area.

The experiments performed in this work started with the database creation by reading and extracting the text information from existing WBS in different file formats. Many challenges were identified during this step due to the variation in text and file formatting. After the information was extracted from the original files, text pre-process techniques were applied in the final database, resulting in a total of 1373 drilling operations WBS and over 32000 well barrier elements samples.

After the database is created, eight supervised classifiers are chosen to be evaluated on the experiment: Naive Bayes, Support Vector Machine (SVM), K-Nearest Neighbours (KNN), Decision Tree, Random Forest, Multi Layer Perceptron (MLP), a simple deep learning classifier and finally a Convolutional Neural Network (CNN) classifier.

On the experiments the samples are split in training and test based on the drilling operation WBS. A five fold cross-validation is applied for each model. With exception of the CNN classifier, a feature reduction technique is also evaluated. In the end of the experiment, five different CNN models and ten models of each of the remaining classifiers are trained and the average and best performance is compared during the evaluation.

The models are evaluated based on the correct classification of well barrier, barrier envelope and drilling operation WBS. The results show that machine learning classification can be applied to predict drilling operation WBS. Random Forest is the classifier that performed the best, achieving a maximum accuracy up to 96% for the classification of a well barrier element and 67.4% for the classification of an entire drilling operation WBS.

Acknowledgements

This work has come with a set of personal completely unforeseen challenges, in which life decided that the best timing to throw lemons was in the middle of this master thesis study. But if you are reading this, it means I made it, maybe not fully in the way it was planned, but in the way I could make lemonade out of it.

I am specially grateful to professor Antorweep Chakravorty for the opportunity to pursue this idea and the understanding shown during the development of the work and challenges presented in the way.

I would like to thank my colleagues in FLX for the support I have received along the way, specially Atle Sivertsen and Tomas Fjelde for the flexibility given to me when needed for fulfilling the studies.

To my family and friends for the nice words of encouragement.

And most importantly, to my partner in life, that has been by my side lifting me up even when our ground has been taken away from us.

Contents

Abbreviations

AI	Artificial Intelligence
ANN	Artificial Neural Network
BOP	Blow-out Preventer
CIV	Chemical Injection Valve
CNN	Convolutional Neural Network
CSV	Comma Separated Values
DHPG	Downhole Pressure Gauge
DHSV	Downhole Safety Valve
GLV	Gas Lift Valve
KNN	K-Nearest Neighbours
MD	Measured Depth
MLP	Multi Layer Perceptron
NLP	Natural Language Processing
NCS	Norwegian Continental Shelf
SVM	Support Vector Machine
TOC	Top Of Cement
TVD	True Vertical Depth
XMT	Christmas Tree
WBE	Well Barrier Element
WBS	Well Barrier Schematics

Chapter 1

Introduction

1.1 Motivation

The motivation for this work is somewhat personal. Since October 2009, I have been working in the oil and gas industry as an engineer in drilling and well operations. Being part of the industry has given me an inside view of the potential opportunities for automation and digitalization of activities and processes that are required to ensure the safe planning, operation and production of oil and gas.

Despite the technological advance in the oil and gas industry in the recent years, many of the tasks performed on the preparation for developing a well are performed manually[1] by the personnel responsible for planning and executing the drilling and well activities. These activities include for example creating documents such as calculations and drawings in which the well specific input needs to be entered manually by the user.

One very important task usually assigned to the engineers during the planning of drilling and well operations is the definition of the well barriers and creation of the well barrier schematics (WBS) for those operations. From my own experience, this task is as important as it is tedious to be performed without the correct tools.

The task hold such importance because it determines how safe the drilling and well operation is planned to be. Before drilling any hole in the ground, the planning team needs to ensure that all the operations will be performed with the correct barriers in place at all stages of the well life cycle. In other words, the team needs to think through the entire operation, step by step, and make a visual representation of the equipment installed and identify which of the well components are acting as a barrier to stop the

[1] by manually it can be understood as using digital solutions that are not fit for purpose, requiring manual input from the user.

formation fluids from blowing out uncontrollably to the surface, where people are working in the rig floor.

Without the correct tools, the time spent drawing this well barrier schematics can be quite long. It is a requirement to have a well barrier schematic for all the operations and activities during the well life cycle. In practice, it means roughly twenty to thirty different drawings to be prepared for each new wellbore being constructed. The time used to manually draw these different drawings is quite considerable. However, only a small fraction of the time utilized in this task is to actually plan the well barriers, while a big portion of the time is spent is updating inputs and adjusting color and curves in the drawing itself.

While creating a couple of hundreds of those drawings, I was motivated to look for solutions that could automate the drawing of these well barrier schematics and spare the use of engineering time to quality check the drawings to ensure they are according to the industry requirements.

This work is an attempt on the automation of the before mentioned task, to define the well barrier schematics for different drilling operations by predicting the well barrier elements and envelopes using machine learning classification. The idea is to use available well barrier schematics from previous drilling activities to train machine learning models and further predict the well barrier elements and envelopes for a given drilling operation.

1.2 Related Work

As far as this point, no direct related work has been found in the literature that uses machine learning on the proposed application of predicting well barrier elements and barrier envelopes. However, artificial intelligence (AI) is not new in the oil and gas industry and some inspiring work can be related to the idea proposed.

The oil and gas industry has invested large amounts in innovative technology along the years to make the operations safer and more lucrative. Large part of the investment has been in equipment, such as downhole tools and rig technology. However, the workflow on planning, developing and abandoning a wellbore has remained very similar to the start of the industry and its strength in the 1970's.

Even though computers have facilitated the well planning and operations in the last decades, digitalization is still a recent idea in the oil and gas industry.

The motivation of this work is shared with the recent study presented by B. Brechan [1] in his P.h.D thesis: "Framework for automated well planning and Digital Well Management".

Brechan proposed a theoretical method to achieve automation on well planning, well intervention and well integrity during a well lyfe cycle. According to Brechan, "The techniques to recover oil and gas onshore and offshore are essentially the same. These techniques have been refined and improved over the years, but the workflows and processes from planning through construction to final plugging of wells is still human-oriented. Project teams scoped to plan construction or maintenance of wells often read and produce texts which then is shared with other disciplines vital to achieve the project objective" [1].

The work proposed here could fit in Brechan's Digital Well Management framework as part of the well integrity module, where instead of using engineering hours to create the well barrier drawings, the system proposed here would do the prediction and could be extended further to draw the barrier in an automated way.

In 2011 Tollow Oil presented the importance of a Well Integrity Management System (WIMS) and the implementation of a software solution [2]. These type of systems already exist in the industry, however, most of them remain like a hub of information, gathering the different documents and reports in one place, still depending on the human factor to update and maintain the documentation.

AI and machine learning are slowly entering the oil and gas industry, many ideas have been proposed and some are already implemented. Some examples of these proposed ideas using machine learning in drilling and well operations are presented in [3] and [4].

An important work considered is presented in [5], studying the data management efficiency on oil and gas data; and in [6] using machine learning and Natural Language Processing (NLP) to analyse drilling and completion data. On both works text mining and text processing techniques are applied to extract important information from the heavy text based data available in the industry. Similar techniques will be applied on the development of this work.

1.3 Challenges

The first challenge presented on the development of this study is the lack of previous work in the specific area. Many machine learning applications have been developed in the oil and gas industry, however, no similar application has been found in the same thematic as well integrity and well barrier classification. The lack of references and previous databases adds an extra effort required on the data preparation and opens the possibilities when selecting the classifiers to be used. On the other hand, the challenge presented here can also be seen as an opportunity to start in an area of study that has

not been previously explored and consequently the contributions of this work could be valuable to continue further development.

The multi-disciplinary characteristic of this work is also a challenge. Two different audience might be interested on the solution proposed in this document: the readers with background in the petroleum technology field and the readers from the data science field. In order to reach both audiences, it is important to give the readers enough information to provide the understanding of the problem and the solution proposed. With the intention to make this work accessible to any interested reader, Chapter 2 will approach the necessary background information on drilling and well and machine learning.

A third challenge worth mention is the lack of a fit for purpose database. The quality of the data used in a machine learning algorithm has direct consequences on the response achieved. As mentioned, no previous work has been found in this area, therefore the database created in this work will be crucial to verify if the solution proposed is applicable.

The raw data (samples) utilized in the database creation on this work have different file formats and different user formatting. Each sample raw file is unique, created by a different person. The characteristics of the samples and lack of standardization of the textual content leads to challenges with regards to text preprocessing and text extraction of the data. More details about data extraction and pre-processing will be discussed in Chapter 3.

1.4 Contributions

The database creation is an important contribution of this work. There are many databases containing WBS from the many wellbores constructed in the NCS and world-wide, but this may be the first database for well barrier elements to be used in machine learning.

The main idea proposed is also a valuable contribution of this work. Since there is no published work within this application, the automation on defining well barrier elements by itself is an important contribution.

Chapter 2

Background

The main focus of this work is the application of machine learning on an industry problem, however, some concepts from oil and gas industry and drilling activities are necessary for a better understanding of the application.

This section approaches some fundamentals on machine learning principles and methods used, as well as some simplified explanation on the well construction and components, and the use of these elements as well barriers.

2.1 Machine Learning

2.1.1 Natural Language Processing and text processing

Most of the information available on the samples in this work is text based. Even though text is one of the most common forms of communication for humans, computers require some extra processing when dealing with text information. Machine learning algorithms usually do not accept text as input, instead, the text need to be transformed in a vector of numbers. The processes of transforming text in a vector is known as "vectorization". Prior vectorizing a text, the text information is divided in smaller units called "tokens", that can be comprised of characters, words or combinations of N words (N-grams). The process of breaking down text into tokens is called "tokenization". After the tokenization of text, each token is then assigned to a number during the vectorization process, creating the final vector of numbers to be used in the machine learning algorithm.

The text information in the raw data can appear in different formatting and containing characters that do not necessarily aggregate information. An important task when using text as the input on a machine learning algorithm is the text preprocessing, in which

the text is "cleaned-up" prior the tokenization. The tasks performed during the text preprocessing are varied and depend on the type of text used and its formatting. For example, it is typical during the preprocessing to set the text to lower-case, remove some special characters and punctuations, remove too common words, remove sufixes, etc. The preprocessing if done correctly can reduce significantly the amount of information transferred to the machine learning algorithm without compromising the quality of the results.

Text processing and the use of text in machine learning includes several concepts and techniques that are important to mention:

Term frequency (tf) is associated to the number of times one term appears in the document. The tf of a term in a document is calculated by the following equation:

$$tf(t,d) = \frac{f_{t,d}}{\sum_{t' \epsilon d} f_{t',d}}, \tag{2.1}$$

where $f_{t,d}$ is the term count of term t in the document d and $\sum_{t' \epsilon d} f_{t',d}$ is the count of all terms in the document.

Inverse document frequency (idf) accounts for the importance of a term by measuring the occurrence of the term in the set of all documents. The idf of a term in a document set is calculated by the following equation:

$$idf(t,D) = \log \frac{N}{n_t}, \tag{2.2}$$

where N is the total number of documents in the dataset and n_t is the number of documents in which the term t is present.

$tf - idf$ is the product of term frequency and inverse document frequency, and it gives a measure of importance of a term in the document for a corpus of documents. $tf - idf$ is calculated as follows:

$$tfidf(t,d,D) = tf(t,d) * idf(t,D) \tag{2.3}$$

Natural Language Processing can be understood as techniques that attempt to teach the computer to understand the content in natural language text. Natural language text is

the text that originates from a language, such as english, and contains a set of rules and patterns that makes the sentence to have a meaning. While for a human it can be easy to identify the content and meaning of a sentence, for the computer it can be a challenge. NLP comes into attempt to teach the computer some of the language structure, such as lexical analysis (meaning of each word), syntactic analysis (relationship between words), semantic analysis (meaning of a sentence), etc. [7].

Bag-of-words (BOW) is a technique in NLP to perform the vectorization of text by assigning an integer number (term frequency) to each unique term in the vocabulary. A document is represented by a vector of the vocabulary dimension, containing values of zeros and integers, where integers are only present for the terms present in that document and zeros are assigned to the remaining terms not in the document.

Text Embedding is a NLP technique used to represent a word by a vector in a high dimensional space, where similar words are represented by similar vectors. Text embedding can be used in deep learning as input to the deep learning algorithm. Word2Vec [8] is one of the known algorithms used to perform text embedding by transforming text into the vector representation.

2.1.2 Machine Learning Algorithms

Machine learning algorithms can be understood as algorithms that allows the computer to learn and adapt by using the patterns from the data feed to it. These algorithms can be divided in three main groups:

- Supervised learning - where the machine is given both inputs and outputs (labels) and the algorithm try to identify the relation between input and output.
- Unsupervised learning - the algorithm receives only the inputs and find the similarities on the data samples.
- Reinforcement learning - the machine interacts in an environment to achieve a specific goal and learns through a feedback with reward mechanism.

Supervised learning uses the training data to learn a function that describes the relation between inputs and outputs. After learning, the algorithm is capable of making predictions of the outputs given the inputs. Under supervised learning, three main tasks are worth mentioning:

- Classification - given a set of predefined categories, the algorithm tries to identify which category the inputs belongs to. Ex: given a set of parameters, predict if a well component is a barrier or not.
- Regression - given a range of values, the algorithm tries to predict the output value given the inputs. Ex: predict the price of a house in the market given the size, location, etc.
- Similarity - a combination of classification and regression that tries to predict the similarity given a set of inputs. Ex: ranking of documents in a search engine.

The work presented in this document is a supervised learning classification problem, in which a set of parameters for each well component is given and the algorithm will attempt to classify the well component as a barrier, and to which barrier envelope (primary and/or secondary) the well component belong.

2.2 Drilling and Well

2.2.1 Well Construction

When planning a new wellbore, the design of the well follow the well construction process, that for this application can be defined as the planning and execution workflow to achieve all the preparation necessary in order to plan and develop the well in a safe and efficient way.

Well integrity and well control are two fundamental principles applied in the well construction process, in which the main objective is to reduce the risk that uncontrolled formation fluids are released to the surface. NORSOK D-010 [9] defines the well integrity requirements and guidelines for the Norwegian Continental Shelf and shall be followed in all drilling and well activities in the NCS.

Well integrity can be understood as the use of technical, operational and organizational barriers to prevent uncontrolled well flow. The well integrity must be maintained in all stages of a well life cycle and operations being executed. Well control is strongly linked to well integrity and can be understood as the measures to ensure that for each well operation the control over the formation flow is maintained. If one well barrier fails, a well control incident has happened and the well integrity is compromised. A well control incident can have dramatic consequences to human life and environment, as seen in the Macondo incident in April 2010 in the Gulf of Mexico [10], one of the most recent and catastrophic examples of the consequences when the well control is lost.

As stated in NORSOK D-010 chapter 4, sections 4.2.1 "The well barriers shall be defined prior to commencement of an activity or operation by identifying the required well barrier elements to be in place, their specific acceptance criteria and monitoring method." and 4.2.2, "Well barrier schematics shall be prepared for each well activity and operation" [9].

A well barrier element can be understood as one element that can stop the flow of the well. However, one well barrier element alone might not be enough to stop the flow. A well barrier envelope is comprised by a set of well barrier elements that together can enclose the flow. The NCS and several other locations worldwide follow the two barriers principle, which determines that two barriers shall always be in place in a well during the entire well life cycle. The first barrier that fluids from the formation meet is known as primary barrier. The secondary barrier is the second barrier met by the flow in case the primary barrier envelope fails. A WBS is a visual representation of the well components and barrier envelopes in place for a specific activity or operation in the well.

2.2.2 Well Design

Explaining in a simplified way, a well is constructed by drilling a set of holes in the ground in steps. In order to guarantee the physical structure of the holes drilled, a casing (large diameter pipe) is set inside each hole section and cemented by placing cement on part of the annular volume between the hole wall and the casing outside wall. After the cement has hardened, a seal is created against the formation fluids and pressures, and the mechanical forces from the formations behind the casing and cement. When the sealing is achieved, a smaller new hole can be drilled. This process repeats until the final target depth is reached.

The well total depth depends on the wellpath and location of the target. The target is a location underground in which the wellpath must reach before or at the final depth drilled. For producer wells, the target can be a source of hydrocarbons that will be drained. In the case of an injector well, the target could be a strategical location where water or gas can be injected in order to improve the production of neighbour producer wells.

A well typically contains standard sections and components, such as wellhead, casings, cement, tubing, liners, valves, Christmas tree (XMT), etc. Below is some basic explanation of some of these well components:

- Wellhead - is the well component at the top of the well where the casings are hung, and the Blow-out Preventer (BOP) and/or XMT are installed. For subsea wells

the wellhead is located at seabed, while for dry wellheads they are located at the surface.

- Casing is a large diameter pipe that is placed and cemented into a new drilled section hole with the main purpose of sustain and protect the opening from underground to surface. Typical casings types installed in a well are [11]:
  - Conductor casing - the first and largest casing. Its main function is to give stability of the hole for further drilling and future placement of the wellhead by protecting the hole against hole collapse from the surface (or seabed in case of offshore wells). Common diameter of a conductor casing in the NCS is 30", drilled in a 36" hole section.
  - Surface casing - The second casing installed in a well. The wellhead may be installed on top of the surface casing. Typical sizes in NCS are 18 5/8" and 20" casings, drilled in sections ranging from 20" to 26" holes.
  - Intermediate casing - Usually set prior to a production zone, to isolate the low pressure zones and unstable formations that can cause hole collapse. A well can have more than one intermediate casing. Common sizes of intermediate casings in NCS can range from 11 3/4" to 14" casings. Typical hole sections vary from 13 1/2" to 17 1/2" hole diameter.
  - Production casing - Used to isolate production zones that contain formation pressures in the event of a tubing leak [11]. Typical sizes in NCS range from 7" to 9 5/8" and hole diameter from 8 1/2" to 12 3/4".
  - Liner - is a casing that doesn't go all the way to the wellhead, instead, it stops in the previous casing string installed by the use of a liner hanger. Common sizes in the NCS are from 4" to 7", but can be also found in larger diameters.
  - Tie-back string - is a casing string connected from the liner hanger top to the wellhead, usually not cemented.
  - Tubing - is the last tubular installed in a well and it is responsible for transportation of the hydrocarbons from the reservoir to the surface. Additional components may be installed in the tubing string, such as Downhole Safety Valve (DHSV), Chemical Injection Valve (CIV), Gas Lift Valve (GLV) and Downhole Pressure Gauge (DHPG).

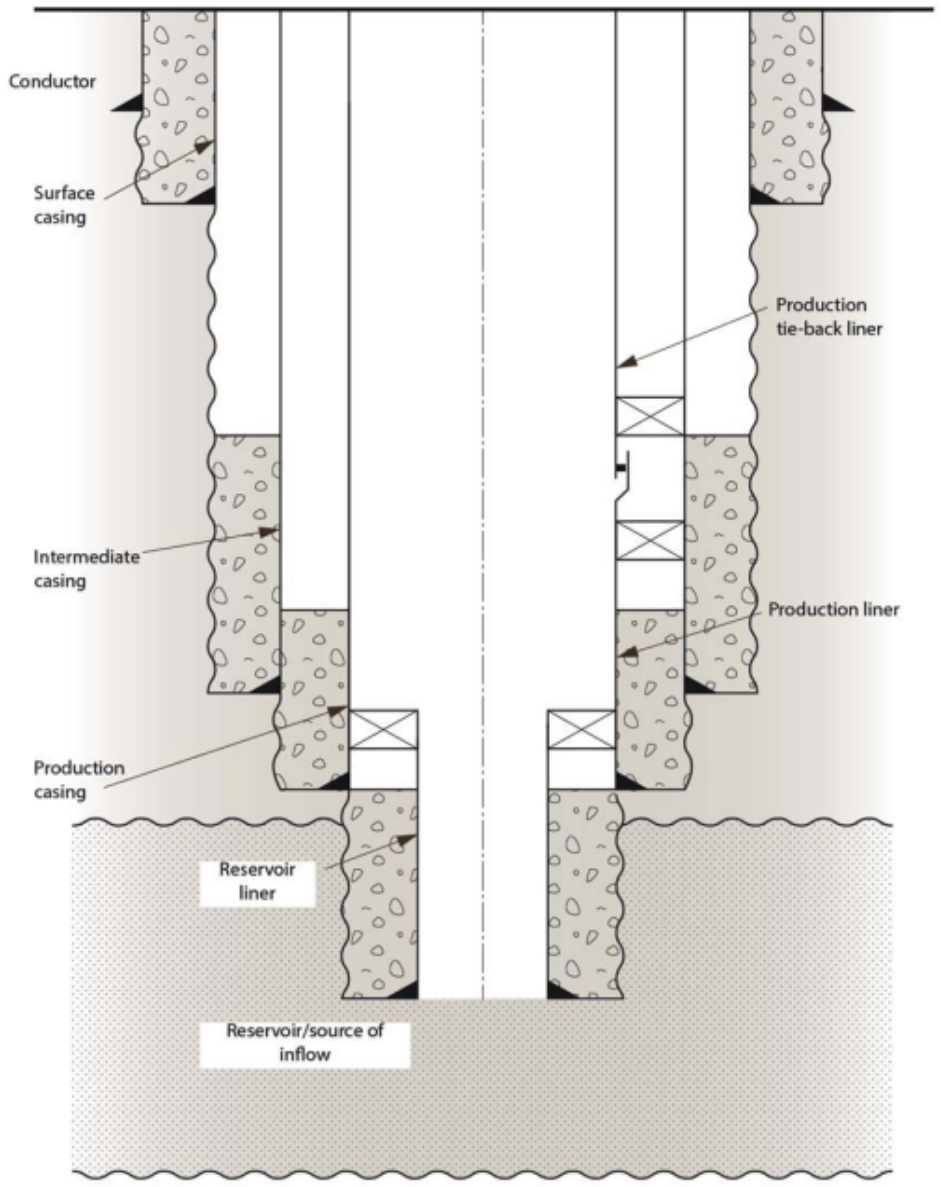


Figure 2.1: Casing Illustration [12].

- Casing hanger - the last piece of a casing installed in the wellhead to hang/connect the casing to the wellhead and provide sealing between the formations behind the casing and the outside environment.
- Liner hanger - it is the last piece of a liner installed in the previous casing wall by anchoring the liner to the casing and providing a seal between the formations behind the liner and the previous casing.
- Casing cement - Cement is applied in most casings and liners installed in a well. Cement must provide the physical structure to hold the casing in place and the hole from collapse directly into the casing wall. In addition, cement may provide

zonal isolation and seal the formations above the cement column from getting into contact with the formation fluids and pressures from the formations drilled deeper.

- Cement plug - a cement plug is placed in a well when the intention is to permanent plug the hole. The cement plug is placed inside the casing or hole section to be plugged.
- Christmas tree - is a set of valves used to control and monitor the flow during production or injection on development wells.

During drilling, completions and well intervention activities, additional equipment is installed in the well in order to prevent a well blowout. Some of the main components are:

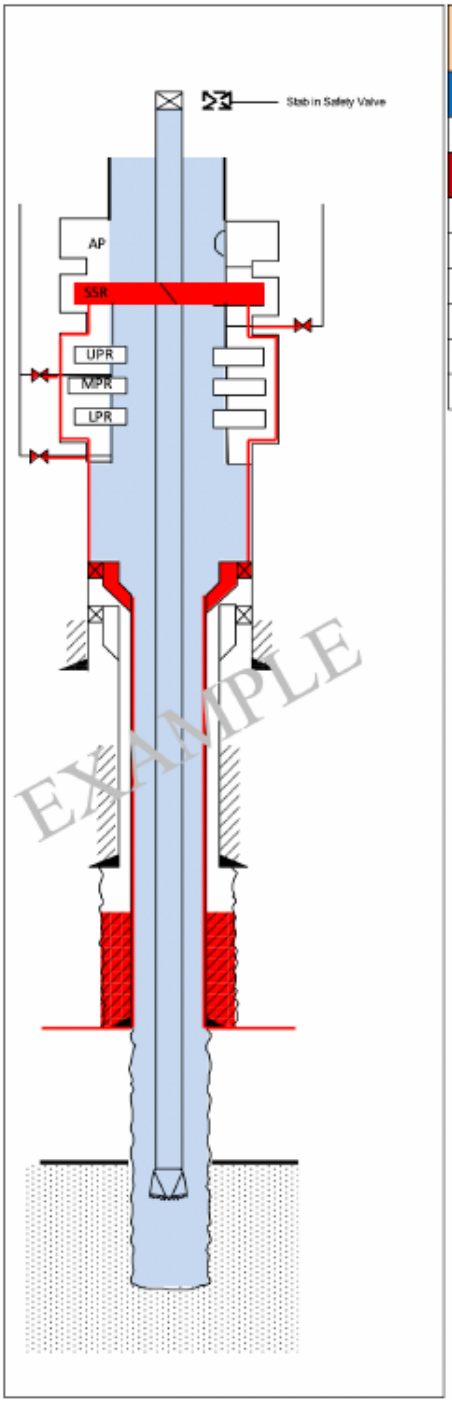
- Blowout Preventer - is the equipment used during drilling, completion and well intervention activities to prevent an uncontrolled flow from the well. A BOP is usually a set of stacked valves with capability to seal the wellbore against the formation flow towards the surface.
- Riser - Riser is a large diameter pipe that connects the BOP to the wellhead or the BOP to the rig surface equipment.
- Fluid column - it is the fluid used to stabilize the well pressure by forcing a hydrostatic pressure from the fluid weight against the formation pressure. In drilling activities, the fluid column is typically provided by the drilling mud. The drilling mud weight and properties are designed and adjusted depending on the formation type and pressure behavior. In conventional drilling it is desired always to keep the formation pressure in overbalance, meaning that the fluid column weight exerts a hydrostatic pressure higher than the pressure exerted by the formation being drilled.

2.2.3 Well Barrier Schematics (WBS)

Norsok Standard D-010 [9] defines the requirements for each well component to be accepted as a barrier element. In Norsok D-010 documentation all acceptable barrier elements are defined and ordered in a table format, in which the barrier elements are identified by an identification number. In the table, the requirements for testing and monitoring the elements are explained. The element identification information used in this work follows the Norsok D-10 Revision 4 [12] due to the time period of the well operations in the samples available.

The requirements for what the well barrier schematics should contain are also presented in Norsok D-010, chapter 5.2.2 [9]. Amongst those requirements, some are replicated below:

- A drawing illustrating the well barriers, with the primary well barrier shown with blue colour and secondary well barrier shown with red colour.
- The formation integrity when the formation is part of a well barrier.
- Reservoirs/potential sources of inflow.
- Tabulated listing of WBEs with initial verification and monitoring requirements.
- All casings and cement. Casing and cement (including TOC) defined as WBEs should be labelled with its size and depth (TVD and MD).
- Well information: field/installation, well name, well type, well status, well/section design pressure, revision number and date, “Prepared by”, “Verified/Approved by”.
- Clear labelling of actual well barrier status – planned or as built.
- Any failed or impaired WBE to be clearly stated.
- A note field for important well integrity information (anomalies, exemptions, etc.).



WBE	EAC table	Verification/monitoring
Primary well barrier		
Fluid column	1	
Secondary well barrier		
In-situ formation	51	
Annulus cement	22	
Casing	2	
Wellhead	5	
High pressure riser	26	
Drilling BOP	4	

Nomenclature

Overbalanced fluid column	
Underbalanced fluid column	
Cement as primary barrier element	
Gas lift gas	
Cement as secondary barrier element	
Cement as primary and secondary barrier element	

Figure 2.2: WBS illustration example from a typical drilling operation, Norsok D-010 [9].

As part of the well integrity requirements, each operation that changes the well barrier envelope needs to have a WBS describing the barriers active for that operation. That results in many WBS drawings for each well being constructed and operated during the well life cycle.

2.3 Norsok D-010 Definitions

1. Well construction process - A subset of activities from the planning to execution of the operations required to construct a well or wellbore.
2. Well life - The period in which a well or wellbore is planned to be active.
3. Primary well barrier - first well barrier that prevents flow from a potential source of inflow [12].
4. Secondary well barrier - second well barrier that prevents flow from a potential source of inflow [12].
5. Well barrier - envelope of one or several well barrier elements preventing fluids from flowing unintentionally from the formation into the wellbore, into another formation or to the external environment [12].
6. Well barrier element - a physical element which in itself does not prevent flow but in combination with other WBE's forms a well barrier [12].
7. Well control - collective expression for all measures that can be applied to prevent uncontrolled release of wellbore fluids to the external environment or uncontrolled underground flow[12].
8. Well integrity - application of technical, operational and organizational solutions to reduce risk of uncontrolled release of formation fluids and well fluids throughout the life cycle of a well [12].

Chapter 3

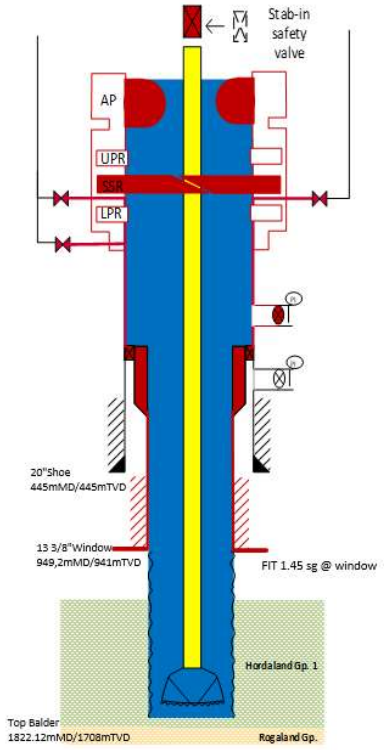
Data Extraction and preparation

3.1 Raw Data

The raw data used in this work is comprised of a collection of well barrier schematics in various file formats, such as Excel with embedded Visio drawing, PDF and Microsoft Visio files. The collection includes drilling operations on wellbores constructed in the Norwegian Continental Shelf, with operation dates ranging from 2014 to 2020.

Each WBS contains information about the well itself, such as components installed, the status of the well in the well life cycle, the operation that the WBS is applicable for, and the well barrier elements that comprise the primary, secondary and if applicable tertiary envelopes. The Figure 3.1 illustrates one WBS drawing from the raw dataset utilized in this work.

WBS applicable for:
Drill 12 1/4" section



Well Data		
Installation		
Well ID		
Well Type		
Revision No.		
Date		
Prepared		
QA/QC		
Approved		
Reference Documentation: Norsok D-010		
Primary well barrier		
Well Barrier elements	Ref. WBEAC table	Verification of barrier elements
Fluid - 1.40 sg Versatec OBM	1	Continuous monitoring
Secondary well barrier		
Well Barrier elements	Ref. WBEAC tables	Verification of barrier elements
Formation (at 13 3/8" casing window)	51	Shmin 1.43 sg, FG 1.53 sg. Note 2. Plan - FIT to min. 1.45 sg
13 3/8" Intermediate casing	2	275 bar with 1.25 sg (DBR 05.06.1995) Plan - logged and tested to 226 bar
13 3/8" Intermediate casing cement	22	32 m3 of lead slurry 1.56 sg and 22 m3 of 1.9 sg of tail. 96% pump efficiency. No losses. (05.06.1995)
13 3/8" Intermediate casing hanger with seal assembly	5	275 bar with 1.25 sg (DBR 05.06.1995) Plan - logged and tested to 226 bar
WH/Annulus valve	12	Pressure tested
Riser	26	Pressure tested
Drilling BOP	4	Pressure tested
Stab-in safety valve	40	
Note: All depth are based on prognosis 1. BTC threads in 13 3/8" casing. Not gas tight. 2. Method: Wellbore Stability (OAF)		
Disp. no.	Comment	

Figure 3.1: WBS drawing from the sample dataset raw data.

In total, 57 wellbores are used as samples. From all the drilling operations in those wellbores, 1373 WBS are analysed and the main relevant information is extracted to be used in the machine learning algorithm.

3.2 Database Creation

The database in a machine learning algorithm can directly define the quality of the results. If bad or wrong data is passed to the model, even the best models will not perform well.

In order to create the database for the machine learning algorithms and due to the different file formatting, the information on each WBS is extracted and pre-processed differently, according to the original file format. The work process for the different file formats is described in the following section.

3.2.1 Information Extraction

The first step on the database creation was to extract the main information from the raw files. This process might sound simple, however, the diversity in file formats and non-standard wording makes the task challenging.

Not all the information available in the WBS is utilized in the final database. From each WBS, the information extracted was the following:

- Operation that the WBS is applicable for (text).
- Well type: text information.
- Primary well barrier elements: Description (text).
- Primary well barrier elements: Norsok table number (int).
- Primary well barrier elements: Verification (text).
- Secondary well barrier elements: Description (text).
- Secondary well barrier elements: Norsok table number (int).
- Secondary well barrier elements: Verification (text).

Excel

For the excel files, Pandas library [13] was used to read the documents and extract the text information into a Pandas Dataframe. The original files have the main information placed on a three column table, with the placement of the table varying slightly from file to file. One excel file may contain several sheets on the working document, one sheet for each drilling operation WBS. The first step in the pre-processing of the Excel files was to divide the main original file into its different sheets, saving each sheet as a CSV (Comma separated Values) file to facilitate further text handling.

Visio

Microsoft Visio files have a different structure than a text file or Excel worksheet document. The latest Microsoft Visio file formats ".vsdx" have the structure similar to a Zipfile, like a container, containing multiple files inside. When unzipped, the Visio file can contain multiple files and information, such as: information in "xml" files, the relationship

between the files in the container (also structured as a "xml") and additional information in other file formats such as ".doc", ".docx", ".xlsx", images, etc.

The first stage on the data extraction of the original Visio files was to ensure that all the files where converted to the ".vsdx" extension. This process was done manually in the data gathering step.

After, each Visio file was unzipped and the information on the many files inside the container was extracted. As for the Excel files, one Visio document may contain several tabs, one for each operation step, resulting in a zipfile containing many "pages" and many "embedding" documents. When analysing the original Visio files, it was observed that the main text information was contained in the equivalent "pageXX.xml" files, where "XX" is the integer ID given by Visio, equivalent to a tab. The table with the barrier elements and barrier envelopes was contained in a "embedding" file, either in Microsoft Word or Excel document format. However, the page ID and embedding document ID are not necessarily the same. Visio creates relationship tags to link the different items. The page relationship to the embedding document was found on the "pages.xml.rels" file, that also uses the "xml" structure to describe the relationships.

In order to facilitate the further text processing, the information of interest from the Visio files was extracted and stored in a "json" format. The information extracted was the page versus embedding document relationship, the text extracted from the "page.xml" file and the corresponding text information from the tables on the embedding files.

PDF

The PDF files presented additional challenges. Even though it is possible to extract the text content from some of the original files, the result was in most cases not the expected. For example, some files returned the text without spacing between the words, making it difficult to extract the relevant information automatically.

Most of the documents available are originated from scanned documents, in those cases the automatic text extraction was not possible.

Due to the challenges mentioned, the information from the PDF files was extracted manually.

3.2.2 Text preprocessing

After extracting the information from the original WBS files, text preprocessing techniques were used to create a standard format. The main text preprocessing techniques applied at this stage were the following:

- The text was set to lower case.
- Multiple spaces were standardized to single space.
- Symbols and punctuation were removed.
- Stop words were removed.

3.2.3 Adding Well Components

In the way that WBS are constructed, only the well components that are active as barrier elements are represented in the WBS table. Consequently, using only the text information from the WBS as input for the machine learning algorithm would lead to a model completely biased simply by the reason that only samples with labels ẗruëẅould be present in the database.

In order to improve the databse in the attempt to reduce the bias and add the elements that are not barriers (label f̈alsë), all the well components that could be present in a wellbore are added to the WBS table for each drilling operation. These additional well components are well elements that by definition could be barrier elements under other circumstances, but are not active for that specific operation. Note that These additional components may include elements that are not installed in the well at that stage on the well construction phase, however, considering the characteristics of machine learning algorithms, the "non-existent" components are assumed to be sorted out by the algorithm during the learning process.

Principles of the well construction design, as described in Section 2.2, are applied in the identification of the components installed in the well and those added to the database.

3.2.4 Final Database

The final database was stored in a CSV file containing in total 1373 drilling operation WBS that can be used as samples. Each operation WBS contains in average 23 well components, resulting in a database with a total of 32014 well barrier elements to be

used during training and evaluation. Each well barrier element input has six parameters and three boolean labels, as following:

- id - integer - Well component identification number.
- ops - string - description of the operation being performed and that the WBS is applicable for.
- name - string - well component name.
- number - integer - corresponding NORSOK table number [12].
- norsok - string - corresponding NORSOK description/name for the well component [12].
- ops_id - integer - operation identification number, unique for each drilling operation WBS.
- barrier - label boolean - True indicates that the well component is acting as a barrier on that specific WBS.
- primary - label boolean - True indicates that the element is part of the primary well barrier envelope for that WBS.
- secondary - label boolean - True indicates that the element is part of the secondary well barrier envelope for that WBS.

Table 3.1 contains a sample of the database, corresponding to a full WBS for the operation id number 10.

Note that a well component can be part of both the primary and secondary well envelopes simultaneously, however, this scenario is not common for drilling operations, consequently no samples are available in the database in which an element belongs to both the primary and secondary envelopes.

Table 3.1: Sample from database.

id	ops	name	number	norsok	barrier	primary	secondary	ops_id
225	cement 1338 intermediate casing	fluid column	1	fluid column	True	True	False	10
226	cement 1338 intermediate casing	formation integrity 20 surface shoe	51	insitu formation	True	False	True	10
227	cement 1338 intermediate casing	20 surface surface casing cement	22	casing cement	True	False	True	10
228	cement 1338 intermediate casing	20 surface surface casing	2	casing	True	False	True	10
229	cement 1338 intermediate casing	wellhead annulus valve	12	wellhead annulus access valve	True	False	True	10
230	cement 1338 intermediate casing	riser	26	surface high pressure riser	True	False	True	10
231	cement 1338 intermediate casing	bop	4	drill bop	True	False	True	10
232	cement 1338 intermediate casing	1338 intermediate casing	2	casing	True	False	True	10
233	cement 1338 intermediate casing	1338 intermediate casing float valve	41	casing float valve	True	False	True	10
234	cement 1338 intermediate casing	conductor casing	2	casing	False	False	False	10
235	cement 1338 intermediate casing	conductor casing cement	22	casing cement	False	False	False	10
236	cement 1338 intermediate casing	insitu formation conductor casing shoe	51	insitu formation	False	False	False	10
237	cement 1338 intermediate casing	production casing	2	casing	False	False	False	10
238	cement 1338 intermediate casing	production casing cement	22	casing cement	False	False	False	10
239	cement 1338 intermediate casing	insitu formation production casing shoe	51	insitu formation	False	False	False	10
240	cement 1338 intermediate casing	wellhead	5	wellhead	False	False	False	10
241	cement 1338 intermediate casing	mechanical plug	28	mechanical tubular plug	False	False	False	10
242	cement 1338 intermediate casing	collapse formation	52	creep formation	False	False	False	10
243	cement 1338 intermediate casing	stabin safety valve	40	stabin safety valve	False	False	False	10
244	cement 1338 intermediate casing	liner hanger packer	43	liner top packer tieback packer	False	False	False	10
245	cement 1338 intermediate casing	drill string	3	drill string	False	False	False	10

Chapter 4

Main Approach

The main objective of this study is to evaluate the use of machine learning techniques on the classification of a well component as a barrier element, and in which barrier envelope the barrier element belongs: primary, secondary or both. The main goal is to evaluate the full set of well components for a well operation, giving as final result the WBS for a drilling operation.

As described in details on Chapter 3, most of the data extracted from the well barrier schematics is text in natural language. Due to the characteristics of the data, the work presented here will focus on the evaluation of text based classification techniques.

In order to evaluate the best results, different approaches on features and model selection are explored as described in the following sections.

4.1 Features and Feature Selection

The features in a text classification task are mainly the text itself and the intrinsic information contained in the text. Since machine learning algorithms are unable to take text as input directly, the text needs to be transformed to numeric information prior running the training and prediction, as explained in more details in Section 2.1.

The first approach in this work was to test the classification using text vectorization based on unigrams (words) and digrams (word pairs) as the features on the machine learning models. Scikit-lean library [14] and the function Vectorize() was applied on the text information to obtain the vectorized model input data.

The second approach applied is based on Natural Language Processing and uses the $tf - idf$ information from the text on the database. For the $tf - idf$, the frequencies of both unigrams and digrams are used as features.

A common challenge in text classification is the high dimensionality of the features. The dimension is proportional to the size of the vocabulary for all the documents evaluated and the use of both unigrams and digrams can increase considerably the number of features, therefore the dimensionality. In order to reduce the number of features, a feature selection technique is also applied and evaluated.

In this work, the γ^2 statistic is the feature selection technique chosen to be evaluated due to its application in text based features, as in [15].

The third and last feature explored in this work are word embeddings, also a NLP technique in which the words or combinations of words are transformed in vectors in a vectors space, where similar words are believed to be close to each other in the vector space.

The Word2Vec [8] algorithm is chosen here to perform the vector space transformation. The algorithm uses neural networks to train and then transform the words into vectors in the defined vector space. Even though oil and gas related pre-trained word2vec models exist in the literature [16], the coverage of words on the pre-trained models didn't seem to be applicable for the specific terms used on the WBS. For this reason, a word2vec model was trained specifically for this work using the words available in the database, limiting the vocabulary to the text contained in the samples available. The word2vec model is trained using unigrams and digrams, through 500 epochs, transforming each word into a vector of dimension 300.

4.2 Model Selection

Along the years, several classification techniques have been researched and applied on text classification tasks. These techniques vary largely in complexity, and their application are strongly correlated to the specific task.

As mentioned previously, up to the time of this study there were no references found on the application of AI on the classification of well barrier elements. This fact opened the research to a broad range of possibilities of machine learning methods that could be explored. Keeping in focus the classification models used in text classification tasks, eight different models are chosen and will be tested and evaluated in this work:

- Multinomial Naive Bayes Classifier

- Support Vector Machine (SVM) Classifier
- K-Nearest Neighbours (KNN) Classifier
- Decision Tree Classifier
- Random Forest Classifier
- Multi-layer Perceptron / Artificial Neural Network (ANN)
- Deep learning ANN - Two dense layers with 64 nodes each
- Convolutional Neural Network (CNN)

The methods chosen to be evaluated in this work vary in complexity as well. Some classical approaches are chosen, such as Naive Bayes, SVM and KNN classifiers, very popular amongst text classification studies.

Decision tree and random forest are selected here based on the technical knowledge of the way the WBS is built and the requirements that the barrier selection needs to follow. As described in Section 2.2.3, there are certain patterns when choosing the barrier elements in a WBS. As a consequence some well components tend to fall in one barrier envelope more often than others. Due to this intrinsic pattern and the characteristics of tree based techniques, tree based classifiers are considered in this work as promising and worth to be evaluated.

Artificial Neural Networks and deep learning ANNs have shown impressive results in the text classification literature. For this reason these methods should not be left unexplored for the problem in this work.

The CNN model tested is inspired by the work presented in [17] and [18], in which convolutional neural networks are used to classify sentences using word embeddings. The idea is implemented here with the thought that each barrier element text input could be understood as a sentence to be classified, therefore in this case the CNN proposed could be a good candidate as a classifier to the problem.

In order to apply the CNN, the word embeddings are implemented using word2vec, as previously mentioned. Word embeddings is only applied to the CNN classifier, while the remaining classifiers evaluated use exactly the same features and feature reduction techniques.

Chapter 5

Experimental Evaluation

5.1 Experimental Set-up

The experiments are built upon testing the performance of the different machine learning classification models presented in Chapter 4 and evaluation of their performance on the well barrier schematics (WBS) prediction given a drilling operation.

All the experiments presented are performed using Python programming language and dedicated python libraries. The main libraries used for building the machine learning models were the following:

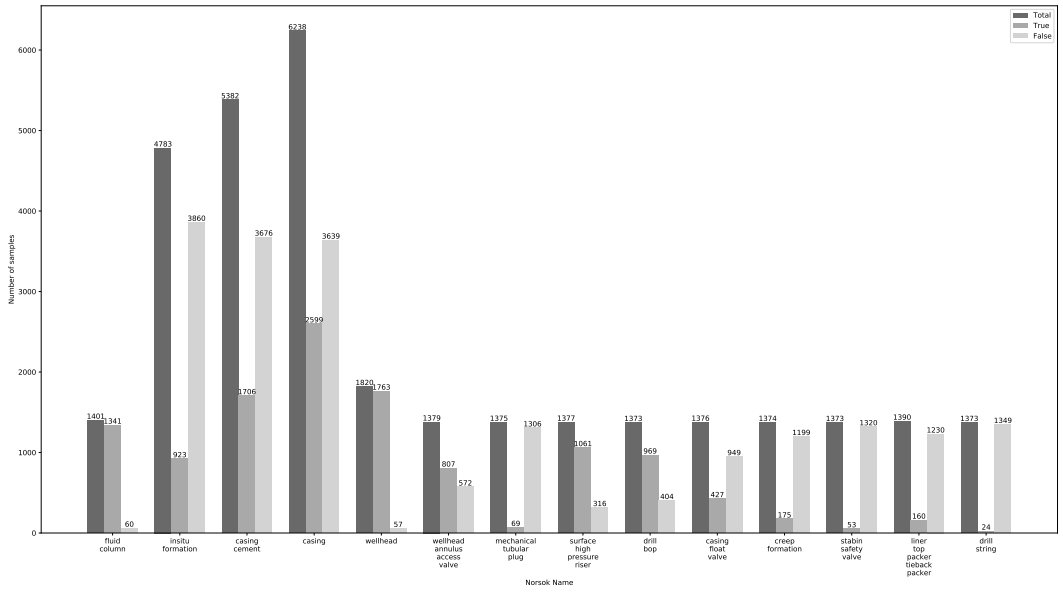
- Scikit-learn [14] - A Python integrated module for machine learning. It contains many machine learning models and preprocessing tools.
- Tensorflow [19] - open source, end-to-end, machine learning platform. Used to build and train deep learning models.
- Keras [20] - A deep learning API build on Tensorflow 2.0, used to facilitate the interface between humans and the deep learning projects. Keras API comes integrated with Tensorflow 2.0.

The input preparation was performed by first splitting the samples into test and training. The items on the dataset were grouped by drilling operation WBS and each operation received an unique identification number (id). To ensure that the final product is a model that can predict a full WBS, the test and training samples were selected based on the operation ids. 20% of the operations are randomly selected as test samples and the 80% remaining operations are used for training.

Each WBS contains on average 23 well components, labelled as "barrier", "primary" and "secondary", indicating if the well component is active as a barrier element for that operation and in which barrier envelope it is active: primary, secondary or both. The final sample split used in the experiments is comprised of 274/1099 test/train operations, and approximately 25650/6350 train/test well components to be trained and evaluated.

The samples are not evenly distributed. The Figure 5.1 shows the number of samples per type of well element according to NORSOK name.

Figure 5.1: Database samples distribution with labels.



Each model is trained and evaluated in a 5-fold cross validation. The test samples for each split are selected by first shuffling the order of the operation ids, and then splitting the total samples in 5 parts.

A word2vec model was trained using the text information from all the samples available on the database. For each test and training split, the samples are transformed using the pre-trained word2vec model and the resulting word embeddings are used as the input in the CNN model.

For the remaining models, text vectorization and $tf - id$ transformation are applied and the combination of the resulting features is used as input for the models. For these models, a γ^2 feature reduction is also applied, in which the features with γ^2 values below 2 are removed and a second training and evaluation is performed.

5.1.1 Evaluation measurements

The evaluation of each model is performed by measuring the accuracy of the classification in five different measurements, in three levels:

- Individual level, in which the individual accuracy for the three different labels is calculated - labels "barrier", "primary" and "secondary";

$$Accuracy_{barrier} = \frac{\text{correct classified elements on label "barrier"}}{\text{total number of elements}} \tag{5.1}$$

$$Accuracy_{primary} = \frac{\text{correct classified elements on label "primary"}}{\text{total number of elements}} \tag{5.2}$$

$$Accuracy_{secondary} = \frac{\text{correct classified elements on label "secondary"}}{\text{total number of elements}} \tag{5.3}$$

- well element level, where a correct prediction is assumed if all the 3 labels for the given well component are correct classified;

$$Accuracy_{element} = \frac{\text{correct classified well components}}{\text{total number of elements}} \tag{5.4}$$

- WBS level, in which a prediction is considered correct if all the well components within the same operation id are correct classified.

$$Accuracy_{operation} = \frac{\text{correct classified operations}}{\text{total number of operations}} \tag{5.5}$$

The best performing classifier is also evaluated per element type according to the NORSOK table[12]. In this step the precision, recall, specificity and F1-score for all the labels is calculated.

$$Precision = \frac{\text{TP}}{\text{TP+FP}} \tag{5.6}$$

$$Recall = \frac{\text{TP}}{\text{TP+FN}} \tag{5.7}$$

$$Specificity = \frac{\text{TN}}{\text{TN+FP}} \tag{5.8}$$

$$F1 - score = 2 * \frac{Precision * Recall}{Precision + Recall} \tag{5.9}$$

Where TP (True Positive) are the *true* samples predicted as *true*, FP (False Positive) are the *false* samples predicted as *true*, TN (True Negative) are the *false* samples predicted as *false* and FN (False Negative) are the *true* samples predicted as *false*.

The total time used by the algorithm to train the model and perform the prediction is also measured and evaluated. The results for the 5-fold cross validation performed for each model is displayed in the Section 5.2.

5.1.2 Model parameters set-up

The models implemented have the following parameters:

- Naive Bayes - used default settings from scikit-lean method MultinomialNB().
- SVM - Default settings from scikit-lean method SGDClassifier().
- KNN - Default settings from scikit-lean method KNeighborsClassifier().
- Decision tree - Default settings from scikit-lean method DecisionTreeClassifier().
- Random Forest - Default settings from scikit-lean method RandomForestClassifier().
- MLP - Default settings from scikit-lean method MLPClassifier(random_state = 1)).
- Deep Learning classifier - Different layer set-ups were experimented by try and error. The final set-up uses an initial Keras sequential layer, followed by two dense layers with 64 nodes each both with reLU activation function; and a dense output layer with 3 nodes and Sigmoid activation function. The model is trained in 10 epochs with a batch size of 5, binary cross-entropy as loss function, Adam optimizer and accuracy as the main metric.
- CNN - Using the idea presented on [17], the final CNN model applied to this problem is build as the following: one static embedding layer initialized with the pre-trained word2vec model weights; two 1D-CNN layers with kernel size 2 and max

pooling 2, activation function reLU; one 1D-CNN layer with kernel size 3, activation function reLU; one flatten layer; one dense layer with 300 nodes, activation function reLU; one drop-out layer with drop-out rate 0.5; and finally, an output dense layer with 3 nodes and Sigmoid activation function. Padding is applied on the CNN layers. During training a Binary cross-entropy is used as loss function, optimizer is Adam and the metric is accuracy.

5.2 Experimental Results

The experiments were run in the linux environment, on a stationary computer with the following configuration:

- Intel(R) Core(TM) i9-10850K CPU @ 3.60GHz x 20
- 64 Gb Memory
- GPU GeForce RTX 3090, Nvidia, Cuda version 11.1
- Linux operational system - Ubuntu 20.04.2 LTS

Tables 5.1 and 5.2 contain the average accuracy for the machine learning classification methods evaluated without and with feature reduction techniques applied, respectively.

Table 5.1: 5-fold cross validation average classification results without feature reduction.

Model	Total time [s]	Barrier [%]	Primary [%]	Secondary [%]	Element [%]	Operation [%]
Naive Bayes	0.48	89.1	94.05	88.3	83.6	1.9
SVM	31.7	94.97	98.07	94.53	92.38	44.76
KNN	65.04	95.75	98.1	94.96	94.32	55.53
Decision Tree	32.82	96.39	98.28	95.65	95	**61.10**
Random Forest	20.25	97.07	98.29	96.18	95.61	**65.35**
MLP	124.28	96.56	98.45	96.01	95.04	**62.20**
DL	39.63	96.55	98.34	95.85	94.94	**60.73**
CNN	163.46	96.85	98.34	96.06	95.23	**60.81**

Table 5.2: 5-fold cross validation average classification results with feature reduction.

Model	Total time [s]	Barrier %	Primary %	Secondary %	Element %	Operation %
Naive Bayes	0.31	89.42	95.6	88.7	84.69	1.9
SVM	19.11	95.19	97.7	94.06	92.09	41.25
KNN	36.46	96.19	98.07	95.36	94.73	57.36
Decision Tree	5.32	96.44	98.27	95.61	94.99	**61.39**
Random Forest	12.59	97.11	98.31	96.19	95.67	**64.76**
MLP	146.81	96.63	98.39	95.91	94.95	**62.34**
DL	39.43	96.56	98.33	95.77	94.91	59.78

From these results, the Randon Forest method is the classifier with the best performance on predicting a full WBS, achieving 65.35 % average accuracy without feature reduction. By applying the γ^2 feature reduction suggested, the Random Forest classifier presented a decrease in the average accuracy of 0.59% and a considerable reduction in the total time for training and prediction, using less than 2/3 of the time used with all the features.

The best results from the 5-fold cross-validation are presented in the Tables 5.3 and 5.4.

Table 5.3: 5-fold cross validation maximum accuracy classification results without feature reduction.

Model	Total time [s]	Barrier %	Primary %	Secondary %	Element %	Operation %
Naive Bayes	0.47	89.67	94.84	88.71	84.46	2.93
SVM	28.55	95.61	98.58	95.09	92.99	49.08
KNN	62.10	95.98	98.4	95.34	94.74	58.61
Decision Tree	19.22	96.8	98.5	95.97	95.45	**63.74**
Random Forest	19.65	97.21	98.54	96.42	95.89	**67.03**
MLP	96.95	96.81	98.72	96.26	95.36	**64.84**
DL	39.28	96.85	98.55	96.21	95.54	**64.1**
CNN	45.81	97.41	98.54	96.73	95.73	**63.74**

Table 5.4: 5-fold cross validation maximum accuracy classification results with feature reduction.

Model	Total time [s]	Barrier %	Primary %	Secondary %	Element %	Operation %
Naive Bayes	0.30	89.91	95.98	89.3	85.5	2.56
SVM	17.35	95.62	98.03	95.31	92.98	47.99
KNN	35.75	96.28	98.3	95.58	94.98	**60.44**
Decision Tree	3.76	96.77	98.48	95.9	95.32	**64.47**
Random Forest	12.22	97.27	98.54	96.5	96.01	**67.40**
MLP	93.58	96.78	98.55	96.29	95.38	**66.30**
DL	38.98	96.79	98.52	96.2	95.42	**63.37**

Again, the results show that the Random Forest classifier is the best performer for the problem proposed, achieving a maximum accuracy on the WBS level of 67.03% without feature reduction and 67.4% when using the feature reduction technique suggested. Note that the maximum accuracy has been improved using feature reduction.

The CNN, Decision Tree, ANN-MLP and Deep Learning classifiers have also performed well, achieving average accuracies above 60%. It is important to note that parameter tuning was not performed during the experiments, leading to a potential of improvement on the results if a more detailed selection would to be performed.

From the classifiers evaluated, considering the computational efficiency and the results achieved, Random Forest classifier was clearly the best performer. The CNN presented a very slow first epoch, something also experienced by other users in tensorflow forums.

The first epoch lasted close to 10 minutes, driving the average training time of the CNN model to a much higher value than the following runs in which it was required approximately 42 seconds for a full training and prediction.

An additional evaluation was performed with the objective of verifying the performance of the classifier for each well barrier element type available in this database. The table 5.5 display the classification results for the Random Forest classifier on the label "barrier".

Table 5.5: Random Forest well barrier classification average results without feature reduction per NORSOK name.

NORSOK name	% True samples	% False samples	Accuracy	Precision	Recall	Specificity	F1-score
fluid column	95.72	4.28	0.96	0.96	1.0	0.13	0.98
insitu formation	19.3	80.7	1	1	1	1	1
casing cement	31.7	68.3	0.96	0.95	0.9	0.98	0.93
casing	41.64	58.36	0.96	0.97	0.93	0.98	0.95
wellhead	96.88	3.12	0.97	0.97	0.99	0.07	0.98
wellhead annulus access valve	58.5	41.5	1	1	1	1	1
surface high pressure riser	77.12	22.88	0.93	0.94	0.97	0.8	0.96
drill bop	70.68	29.32	0.92	0.93	0.95	0.84	0.94
casing float valve	31.1	68.9	0.99	1	0.98	1	0.99
mechanical tubular plug	5.02	94.98	0.98	0.98	0.62	1	0.75
creep formation	12.7	87.3	1	1	1	1	1
stabin safety valve	3.86	96.14	0.98	0.92	0.68	1	0.77
liner top packer tieback packer	11.44	88.56	0.99	0.98	0.89	1	0.93
drill string	1.76	98.24	0.99	0.57	0.33	1	0.38

From the results in Table 5.5 it can be observed that the type of element versus barrier label is unbalanced in the test dataset. For example, there are more "not barrier" elements (label = False) for the well components "drill string", "mechanical tubular plug" and "stab-in safety valve" in the test dataset. This could be explained by the type of operations available in the dataset, in this case only drilling operations and due to the characteristics of drilling operations these elements are more often not active as barriers. In the other hand the elements "fluid column" and "wellhead" are very often a barrier during drilling operations and consequently the dataset is unbalance towards the True values.

The recall for the positive labels is high and the specificity for the negative labels is low for the elements "fluid column" and "wellhead", indicating that the classifier tend to predict the elements as barrier (True). The opposite is observed for the elements "drill string", "mechanical tubular plug" and "stab-in safety valve".

The drilling operations characteristics can be seen as an advantage when considering that the unbalanced characteristic of the dataset will reflect the barrier conditions for the operation and the classifier will tend to predict the barrier correctly. However, it

will be a disadvantage for the operations that are slightly different from the standard operations, since those will have more well elements predicted wrong.

Chapter 6

Conclusion and Future Directions

This study aimed to explore the use of machine learning on the prediction of well barrier elements on a well barrier schematics for drilling operations. The experiments were performed using previous WBS information from drilling operations on a set of wells in the Norwegian continental shelf and known machine learning algorithms previously used for text classification.

The main conclusions with the study are the following:

- Results show that it is possible to use artificial intelligence techniques to achieve the prediction of well barrier elements on the proposed problem, and that the automation for predicting and drawing well barrier schematics is achievable.
- Data preparation is an important, but time consuming step. The different file formats resulted on additional manual work and tailor made automation, making it difficult to apply the same algorithm on a different sample set. However, the result of this effort is an unique database that could be further explored on future studies.

Some possible future directions are suggested:

- Parameter tuning for the best models could potentially improve the results achieved.
- Implementation of a drawing feature using the outputs of the prediction algorithm would complement this study and give the visual feature of the well barrier schematics.
- The time frame for this study was limited and consequently the scope was reduced to the prediction of WBS for drilling operations. However, the idea could be extended to WBS for operations in the entire well life cycle.

- Storing the WBS in a standard format and possibly integrated to a database system would facilitate the automation. If such a standardization is applied in the correct way, the database would increase automatically as new WBS are created, eliminating the manual work of data preprocessing.
- This study focused on the text information contained in the WBS. The oil and gas industry produce a vast amount of information that was not taken into consideration in the prediction methods implemented here, but this information could potentially be valuable if added as input parameters in the algorithms. For example, information such as formation depths, casing shoe and cement depths could possibly improve the prediction of the well barrier elements.

List of Figures

List of Tables

Appendix A

Main Code

The code implemented in this work uses Python language, version 3.8. The final database, trained word2vec model and code will be also available in the authors github repository [21] upon evaluation and approval of this document.

A.1 preprocess.py - Preprocess text from database

```
import pandas as pd
import numpy as np
import os
import re
import json
from random import randint
import multiprocessing
import nltk
nltk.download('stopwords')
from nltk.corpus import stopwords
stop_words = list(stopwords.words('english')) + ['in','inches']

sizes = ['13 5/8', '13 3/8', '11 3/4', '10 3/4','9 5/8', '5 1/2','17 1/2'
    ,'12 1/4', '8 1/2','7 5/8']

casing_sizes = ['20', '13 5/8', '13 3/8', '11 3/4', '10 3/4','9 5/8', '7'
    ,'5 1/2']
std_casings = {
    'conductor':['30', '36'],

    'surface': ['20', '26','18 5/8','18.625'],

    'intermediate': ['13 3/8', '13.375', '13 5/8','13.625', '14', '11 3/4
    ','11.75'],
```

```
    'production': ['10 3/4" x 9 5/8','9 5/8', '9.625', '7 5/8', '7.625','7', '5']
    }

#read the database and return the main information to be used in the ML algorithm as a dataframe
def read_data(file,type_data='processed'):
    if type_data == 'processed':
        df = pd.read_csv(file,usecols = ['id','ops','ops_id','name','number','norsok','barrier','primary','secondary'], index_col='id')
    else:
        df = pd.read_csv(file,usecols = ['id','ops','name','number','norsok','barrier','primary','secondary'], index_col='id')
    return df

# find multiple files in a directory with given extension
#returns a list of full files path + names
def walk_files(initial_path,extension = ['xlsx']):
    all_files = []
    for root, dirs, files in os.walk(initial_path):
        for name in files:
            f, ext = os.path.splitext(name)
            if ext in extension:
                all_files.append(os.path.join(root,name))
    return all_files

# Check if word is stopwords using nktl stopwords corpus
# return NaN if word is stopword
def rmstopwords(word):
    if word not in stop_words:
            return word
    else:
        return 'NaN'

# Remove some sufixes and single letters
def sufix(word):
    if len(word) >= 1:
        # Remove s from plural words
        if word[-1] == 's' and word[-2:] != 'ss' and word[-2:] != 'us':
            return word[:-1]
        #remove punctuation in the end of the word
        elif word[-1] == '.':
            return word[:-1]
        # remove 'ing' from words except casing and tubing
        elif 'string' not in word and 'casing' not in word and 'tubing' not in word and word[-3:] == 'ing':
            return word[:-3]
        # remove 'ed' and 'ly' from word
```

```
65          elif word[-2:] == 'ed':# or word[-2:] == 'ly':
66              return word[:-2]
67          # remove sufixes 'ness' and 'less'
68          elif word[-4:] == 'ness' or (word[-4:] == 'less' and len(word) !=
        4):
69              return word[:-3]
70          elif word[-2:] == 'nn':
71              word = word[:-1]
72          return word
73      else:
74          return 'NaN'
75
76  # Check the type of casing and return the type (surface, conductor, etc)
        or empty string
77  def find_casing_type(size):
78      for key in std_casings.keys():
79          if size in std_casings[key]:
80              return key
81      else:
82          return ''
83
84  # Append the type of casing to the string (surface, conductor, etc)
85  def append_casing_type(text):
86      for size in casing_sizes:
87          if size in text:
88              casing_type = find_casing_type(size)
89              index = text.find(size)+len(size)+1
90              if casing_type != '':
91                  text = text[:index] + ' ' + casing_type + text[index:]
92                  break
93      return text
94
95
96  # split the text by spaces and new line, excluding the sizes (casing and
        hole sizes)
97  def split_text(text):
98      global sizes
99      text = text.replace('Â ','1/4').replace('Â¡','1/2').replace('âĚİ','
        5/8').replace('Â¿','3/4')
100     for size in sizes:
101         if text.find(size) != -1:
102             text = text.replace('csg', 'casing').replace('casg', 'casing'
        ).replace('cais', 'casing')
103             text = text[:text.find(size)] + size.replace(' ','') + text[
        text.find(size)+len(size):]
104
105     return re.split(r'[\s*\n]',text)
106
```

```
107
108
109 def preprocess(doc):
110     """Preprocesses text.
111
112     Args:
113         doc: String comprising the unprocessed contents.
114
115     Returns:
116         String comprising the corresponding preprocessed text.
117     """
118     # Set all text to lower text
119     doc = doc.lower()
120     # Punctuations to be removed
121     punctuation = '!#$%@&\'âÃŹ"âÃİ()*+,-:;<=>?[\\/]^_‘{|}~' #
122
123     #append type of casing on string
124     doc = append_casing_type(doc)
125     # Split by white spaces (take into accont sizes like 11 3/4, converst
        to 113/4)
126     textlist = split_text(doc)
127     # Remove punctuation, stop words and sufix of each word, make it to
       pandas series to easily clean up the empty elements
128     words = pd.Series([sufix(rmstopwords(text.translate(str.maketrans('',
        '', punctuation)))) for text in textlist])
129     words = words[words != 'NaN']
130     text = ' '.join([word for word in words.to_list()])
131     return text
132
133
134 def preprocess_multiple(docs):
135     """Preprocesses multiple texts.
136
137     Args:
138         docs: List of strings.
139
140     Returns:
141         List of strings, each comprising the corresponding preprocessed
       text.
142     """
143     alldocs = []
144     # Process each string for a document using preprocess() and append in
        alldocs
145     for doc in docs:
146         alldocs.append(preprocess(doc)+' ')
147
148     # Returns the list of lists for all documents
149     return alldocs
```

```
150
151
152 def preprocess_dataframe(df):
153     for col in ['ops','name','norsok']:
154         df[col] = preprocess_multiple(df[col].values)
155
156     return df
157
158 def find_ops_ids(df):
159     previous = ''
160     ids = []
161     list_ids = []
162     count = -1
163     for i,ops in enumerate(df['ops'].values):
164         if previous != ops:
165             ids.append(i)
166             previous = ops
167             count += 1
168         list_ids.append(count)
169     return ids,list_ids
```

A.2 wbs_features.py - Features related functions

```
1  import pandas as pd
2  import numpy as np
3  import preprocess as pp
4  from random import sample
5  from time import time
6  import scipy.sparse as sparse
7  import os
8  import pickle
9  import json
10 from datetime import datetime
11 from nltk.util import ngrams
12
13 #import gensim
14 from gensim.models import Word2Vec
15 from tensorflow.keras.preprocessing.sequence import pad_sequences
16
17 # Feature vectorizers
18 from sklearn.feature_extraction.text import CountVectorizer,
       TfidfVectorizer
19 from sklearn.metrics import accuracy_score,make_scorer
20 # Feature selection
21 from sklearn.feature_selection import RFECV
22 from sklearn.feature_selection import chi2, SelectKBest
```

```
23
24
25
26 def vectorize_X(data_X):
27     ''' Prapare dataset using unigrams and digrams. Uses the text from
       the columns
28     'ops', 'name' and 'norsok' concatenated as one string of text as
       input for
29     CountVectorizer.
30     data_X - features in pandas Dataframe
31     Returns a sparse matrix '''
32     vectorizer = CountVectorizer(analyzer='word', ngram_range=(1, 2))
33     X = vectorizer.fit_transform(data_X['ops'] + data_X['name'] + data_X[
       'norsok'])
34     return X
35
36
37 def tfidf_X(data_X):
38     ''' Prapare dataset using TF_IDF for unigrams and digrams. Uses the
       text from the
39     columns 'ops', 'name' and 'norsok' concatenated as one string of text
        as input for
40     CountVectorizer.
41     data_X - features in pandas Dataframe
42     Returns a sparse matrix '''
43
44     vectorizer = TfidfVectorizer(analyzer='word', ngram_range=(1, 2),
       use_idf=True,
45     smooth_idf=True,sublinear_tf=True)
46     X = vectorizer.fit_transform(data_X['ops'] + data_X['name'] + data_X[
       'norsok'])
47     return X
48
49 def append_sparse(A, B):
50     return sparse.hstack((A,B))
51
52 def load_database():
53     ''' Loading database - if preprocessed database not created, do the
       text
54     preprocessing and save database. '''
55     try:
56         data = pp.read_data('preprocessed_database.csv')
57
58     except:
59         data = pp.read_data('database_all_without_pna.csv','main')
60         data = pp.preprocess_dataframe(data)
61         ids,ops_ids = pp.find_ops_ids(data)
```

```
62          # Adding operation id to dataframe columns - input will be
    evaluated per operation
63          data['ops_id'] = ops_ids
64          data.to_csv('preprocessed_database.csv')
65
66      # Save train X inputs as one dataframe - uses columns 'ops','name','
    norsok','number'
67      data_X = data[['ops','name','norsok','number','ops_id']].copy()
68      # Save train y labels (3 labels) in one dataframe - multiply by 1 to
    get 0/1
69      y = data[['barrier','primary','secondary']]*1
70      return data_X, y
71
72  def prepare_features(data_X, features = 'all'):
73      if features == 'all':
74          # Preparing data with TF-IDF for unigrams and digrams
75          X= tfidf_X(data_X)
76          # Preparing data with for unigrams and digrams
77          a = vectorize_X(data_X)
78          # Append sparse vectors with two vectorizers
79          X = append_sparse(X,a)
80
81      elif features == 'tfidf':
82          # Preparing data with TF-IDF for unigrams and digrams
83          X = tfidf_X(data_X)
84
85      else:
86          # Preparing data with for unigrams and digrams
87          X = vectorize_X(data_X)
88
89      # Adding norsok codes (integers) to the X sparse matrix
90      X = sparse.hstack((X,np.array(data_X['number'].values)[:,None])).A
91
92      return X
93
94  # Calculate chi2 statistic of features
95  # Return chi2 and p-values
96  def chi_squared(X, y):
97      chi_scores = chi2(X,y)
98      return chi_scores
99
100 # Select features with chi2 statistic above 2 and return train and test
    datasets reduced
101 def select_best_features(X,y,test_X):
102     k = sum(chi_squared(X,y)[0] > 2)
103     f_select = SelectKBest(chi2, k = k)
104     new_X = f_select.fit_transform(X,y)
105     test_new_X = f_select.transform(test_X)
```

```
106     return new_X, test_new_X
107
108
109 # read pickle file
110 def read_pickle(filename):
111     with open(filename, 'rb') as file:
112         model = pickle.load(file)
113     return model
114
115 # Save data into a json file
116 def save_data_to_json(data,filename):
117     jsonObject = json.dumps(data)
118     with open (filename +'.json', 'w',encoding='utf-8') as f:
119         json.dump(jsonObject,f,ensure_ascii=False)
120     return
121
122 #read a json file and return a dict
123 def read_json(filename):
124     try:
125         with open (filename, 'r', encoding='utf-8') as f:
126             data = json.load(f)
127         return json.loads(data)
128     except:
129         return False
130
131 def load_data(features = 'all'):
132     data_X, y = load_database()
133     X = prepare_features(data_X,features)
134     return X, y, data_X
135
136 def prepare_embeddings(test_index = [], previous=False, resultsfile='',
137                        model_file='word2vec_bigrams.model'):
138     data, y = load_database()
139     data_text,model,max_len = load_word2vec(model_file)
140     # Load results from previous run, load train and test samples ready
    arrays
141     if previous:
142         with open(resultsfile, 'rb') as file:
143             results = pickle.load(file)
144
145         train_X = results['train_X']
146         train_y = results['train_y']
147         test_X = results['test_X']
148         test_y = results['test_y']
149         test_index = results['test_index']
150         print('Loaded test and train...')
151
```

```
152     # Prepare train and test arrays text embedings using word2vec and
    unigram and bigrams
153     else:
154 
155         # Use test index if provided
156         if len(test_index) == 0:
157             # splitting train and test
158             ids = data.ops_id.unique()
159             np.random.shuffle(ids)
160             n_test = len(ids)//5
161             i=0
162             # finding operation ids for the test set
163             test_ids = ids[i*n_test:(i+1)*n_test]
164             # finding indexes for test set
165             test_index = data.loc[data.ops_id.isin(test_ids)].index.
    values
166 
167         # finding indexes for training set
168         train_index = data.loc[~data.index.isin(test_index)].index.values
169 
170         #preparing/spliting train and test tokens
171         train_X = tokenize(data_text.iloc[train_index].values,2)
172         test_X = tokenize(data_text.iloc[test_index].values,2)
173         # preparing labels
174         train_y = y.iloc[train_index]
175         test_y = y.iloc[test_index]
176         # preparing sequences
177         train_X = tokens_to_sequences(train_X,model.wv)
178         test_X = tokens_to_sequences(test_X,model.wv)
179 
180         # padding embeddings
181         train_X = pad_sequences(train_X,maxlen=max_len)
182         test_X = pad_sequences(test_X,maxlen=max_len)
183         print('Test shape/train shape:')
184         print(test_X.shape,train_X.shape)
185 
186     return train_X,train_y,test_X,test_y,test_index,model,max_len
187 
188 # tokens to sequences
189 def tokens_to_sequences(tokens,w2v):
190     ''' Transform tokens in a sequence of numbers using word2vec keys.'''
191     seq_tokens = []
192     for line in tokens:
193         seq = []
194         for word in line:
195             seq.append(w2v.key_to_index[word])
196         seq_tokens.append(seq)
197     return seq_tokens
```

```
198
199 def tokenize(data_text, n_grams=1):
200     ''' Tokenize string to unigrams or digrams'''
201     if n_grams == 1:
202         unigrams = [x.split() for x in data_text]
203         return unigrams
204     if n_grams == 2:
205         # adding bigrams
206         unigrams = [x.split() for x in data_text]
207         bigrams = [[' '.join(l) for l in list(ngrams(x,2))] for x in
    unigrams]
208         tokens = [unigrams[i] + bigrams[i] for i in range(len(unigrams))]
209         return tokens
210     else:
211         return []
212
213
214 def load_word2vec(modelname='word2vec_bigrams.model'):
215     ''' Load word2vec model or train model with text from the samples.
216         returns data_tex - text information from samples; model -
    word2vec model;
217         max_len - max sentence length from the samples analysed'''
218
219     # reading data
220     data, y = load_database()
221     # concatenating the text info from the data. Adds the NORSOK codes as
     text
222     data_text = data['ops'] + data['name'] + data['norsok'] + data['
    number'].astype(str)
223     # Creating tokens
224     tokens = tokenize(data_text,2)
225
226     len_tokens = [len(x) for x in tokens]
227     # max_len is twice the length of the longest sentence, taken as
    number of words
228     max_len = max(len_tokens)*2
229
230     # Loading/Creating Word2Vec model and training using tokens
231     try:
232         model = Word2Vec.load(modelname)
233         print('Loaded Word2Vec')
234     except:
235         model = Word2Vec(vector_size=300,min_count=1)
236         model.build_vocab(tokens)
237         total_examples = model.corpus_count +1
238         model.train(tokens, epochs=500, total_examples=total_examples)
239         model.save(modelname)
240
```

```
241     return data_text, model, max_len
```

A.3 wbs_classifiers.py - Classifiers

```
1  import numpy as np
2  from random import sample
3  from time import time
4  import scipy.sparse as sparse
5  import os
6  from datetime import datetime
7  import pickle
8
9  # Classifiers
10 from sklearn.linear_model import SGDClassifier
11 from sklearn.naive_bayes import MultinomialNB
12 from sklearn.tree import DecisionTreeClassifier
13 from sklearn.ensemble import RandomForestClassifier
14 from sklearn.neural_network import MLPClassifier
15 from sklearn.neighbors import KNeighborsClassifier
16
17 import tensorflow as tf
18 physical_devices = tf.config.list_physical_devices('GPU')
19 tf.config.experimental.set_memory_growth(physical_devices[0], True)
20 from tensorflow import keras
21 from tensorflow.keras.models import Sequential
22 from tensorflow.keras import layers
23 from tensorflow.keras.backend import clear_session
24 from tensorflow.keras.preprocessing.sequence import pad_sequences
25 from tensorflow.keras.callbacks import TensorBoard
26
27
28 # Naive Bayes classifier
29 def nb_classifier(train_X, train_y, test_X,fold='',dirname='.'):
30     ''' Naive Bayes classifier. Uses MultinomialNB() from sklearn.
31     Perform the training and prediction for each label and returns array
       with predictions.
32     train_X - training features
33     train_y - training labels
34     test_X - test features
35     fold - cross-validation actual fold number
36     dirname - folder to save results, model trained and features
37     returns - time used in training and prediction (list) and predictions
        for all labels.
38     '''
39     times = []
40     preds = []
```

```
41      classifier = MultinomialNB()
42      for label in train_y.columns:
43          # Train classifier using fit()
44          t0 = time()
45          classifier.fit(train_X, train_y[label])
46          # record training time
47          times.append(time()-t0)
48          t1 = time()
49          pred = classifier.predict(test_X)
50          #record prediction time
51          times.append(time()-t1)
52          preds.append(pred)
53          #save model
54          filename = os.path.abspath(dirname+'/models/'+'nb_model_'+str(
    fold)+'_'+ label +'.pkl')
55          save_model(classifier,filename)
56      #record total time
57      times.append(sum(times))
58
59      #Format predictions
60      preds = np.array(preds).T
61      return preds, times
62
63  # SVM classifier
64  def svm_classifier(train_X, train_y, test_X,fold='',dirname='.'):
65      ''' Support Vector Machine (SVM) classifier. Uses SGDClassifier()
    from sklearn.
66      Perform the training and prediction for each label and returns array
    with predictions.
67      train_X - training features
68      train_y - training labels
69      test_X - test features
70      fold - cross-validation actual fold number
71      dirname - folder to save results, model trained and features
72      returns - time used in training and prediction (list) and predictions
     for all labels.
73      '''
74      times = []
75      preds = []
76      classifier = SGDClassifier()
77      for label in train_y.columns:
78          # Train classifier using fit()
79          t0 = time()
80          classifier.fit(train_X, train_y[label])
81          # record training time
82          times.append(time()-t0)
83          t1 = time()
84          pred = classifier.predict(test_X)
```

```
85          #record prediction time
86          times.append(time()-t1)
87          preds.append(pred)
88          #save model
89          filename = os.path.abspath(dirname+'/models/'+'svm_model_'+str(
        fold)+'_'+ label +'.pkl')
90          save_model(classifier,filename)
91      #record total time
92      times.append(sum(times))
93      #Format predictions
94      preds = np.array(preds).T
95      return preds, times
96
97  # KN classifier
98  def kn_classifier(train_X, train_y, test_X,fold='',dirname='.'):
99      ''' KN classifier. Uses KNeighborsClassifier() from sklearn.
100     Perform the training and prediction for all labels in one run.
101     train_X - training features
102     train_y - training labels
103     test_X - test features
104     fold - cross-validation actual fold number
105     dirname - folder to save results, model trained and features
106     returns - time used in training and prediction (list) and predictions
         for all labels.
107     '''
108
109     times = []
110     classifier = KNeighborsClassifier()
111     # Train classifier using fit()
112     t0 = time()
113     classifier.fit(train_X, train_y)
114     # record training time
115     times.append(time()-t0)
116     t1 = time()
117     preds = classifier.predict(test_X)
118     #record prediction time
119     times.append(time()-t1)
120     #record total time
121     times.append(sum(times))
122     #save model
123     filename = os.path.abspath(dirname+'/models/'+'kn_model_'+str(fold)+'
        .pkl')
124     save_model(classifier,filename)
125     return preds, times
126
127 # Decision Tree classifier
128 def dt_classifier(train_X, train_y, test_X,fold='',dirname='.'):
```

```
129     ''' Decision Tree classifier. Uses DecisionTreeClassifier() from
        sklearn.
130     Perform the training and prediction for all labels in one run.
131     train_X - training features
132     train_y - training labels
133     test_X - test features
134     fold - cross-validation actual fold number
135     dirname - folder to save results, model trained and features
136     returns - time used in training and prediction (list) and predictions
         for all labels.
137     '''
138
139     times = []
140     classifier = DecisionTreeClassifier()
141     # Train classifier using fit()
142     t0 = time()
143     classifier.fit(train_X, train_y)
144     # record training time
145     times.append(time()-t0)
146     t1 = time()
147     preds = classifier.predict(test_X)
148     #record prediction time
149     times.append(time()-t1)
150     #record total time
151     times.append(sum(times))
152     #save model
153     filename = os.path.abspath(dirname+'/models/'+'dt_model_'+str(fold)+'
        .pkl')
154     save_model(classifier,filename)
155     return preds, times
156
157 # Random Forest classifier
158 def rf_classifier(train_X, train_y, test_X,fold='',dirname='.'):
159     ''' Random Forest classifier. Uses RandomForestClassifier() from
        sklearn.
160     Perform the training and prediction for all labels in one run.
161     train_X - training features
162     train_y - training labels
163     test_X - test features
164     fold - cross-validation actual fold number
165     dirname - folder to save results, model trained and features
166     returns - time used in training and prediction (list) and predictions
         for all labels.
167     '''
168
169     times = []
170     classifier = RandomForestClassifier()
171     # Train classifier using fit()
```

```
172     t0 = time()
173     classifier.fit(train_X, train_y)
174     # record training time
175     times.append(time()-t0)
176     t1 = time()
177     preds = classifier.predict(test_X)
178     #record prediction time
179     times.append(time()-t1)
180     #record total time
181     times.append(sum(times))
182     #save model
183     filename = os.path.abspath(dirname+'/models/'+'rf_model_'+str(fold)+'
        .pkl')
184     save_model(classifier,filename)
185     return preds, times
186 
187 # MLP classifier
188 def mlp_classifier(train_X, train_y, test_X,fold='',dirname='.'):
189     ''' Multilayer perceptron classifier. Uses MLPClassifier() from
        sklearn.
190     Perform the training and prediction for all labels in one run.
191     train_X - training features
192     train_y - training labels
193     test_X - test features
194     fold - cross-validation actual fold number
195     dirname - folder to save results, model trained and features
196     returns - time used in training and prediction (list) and predictions
         for all labels.
197     '''
198     times = []
199     classifier = MLPClassifier(random_state = 1)
200     # Train classifier using fit()
201     t0 = time()
202     classifier.fit(train_X, train_y)
203     # record training time
204     times.append(time()-t0)
205     t1 = time()
206     preds = classifier.predict(test_X)
207     #record prediction time
208     times.append(time()-t1)
209     #record total time
210     times.append(sum(times))
211 
212     #save model
213     filename = os.path.abspath(dirname+'/models/'+'mpl_model_'+str(fold)+
        '.pkl')
214     save_model(classifier,filename)
215     return preds, times
```

```
216
217
218 def dl_classifier(train_X, train_y, test_X, epoc = 100, batch = 10,fold='
    ',dirname='.'):
219     ''' "Deep learning" classifier. Builds a Neural Network with hidden
    layers using
220     Keras and Tensorflow. Perform the training and prediction for all
    labels in one run.
221     train_X - training features
222     train_y - training labels
223     test_X - test features
224     fold - cross-validation actual fold number
225     dirname - folder to save results, model trained and features
226     returns - time used in training and prediction (list) and predictions
     for all labels.
227     '''
228
229     clear_session()
230     times = []
231     dimension = train_X.shape[1]
232     X_val = train_X[-2400:,:]
233     y_val = train_y.iloc[-2400:].copy()
234     new_train_X = train_X[:-2400,:]
235     new_train_y = train_y.iloc[:-2400].copy()
236     t0 = time()
237     model = Sequential()
238     model.add(layers.Dense(64, input_dim=dimension, activation='relu'))
239     model.add(layers.Dense(64, activation='relu'))
240     model.add(layers.Dense(3, activation='sigmoid'))
241     model.compile(loss="binary_crossentropy",optimizer='adam',  metrics="
    accuracy")
242
243     history = model.fit(new_train_X, new_train_y, epochs=epoc, verbose=
    True,
244                 validation_data=(X_val, y_val), batch_size=batch)
245     times.append(time()-t0)
246
247     # saving model
248     model_file = os.path.abspath(dirname+'/models/dl_model_'+ str(fold))
249     model.save(model_file)
250
251     # Predicting using trained model
252     t1 = time()
253     preds = model.predict(test_X, verbose=False)
254     times.append(time()-t1)
255     times.append(sum(times))
256     # Transform results in 0 and 1 using 0.5 as cut
257     preds = (preds >= 0.5)*1
```

```
258     return preds, times
259
260
261 def cnn_classifier(train_X,train_y,test_X,test_y, model, max_len, fold=0,
    dirname='.',
262                     epochs=5,activation='sigmoid'):
263     '''
264     Convolutional Neural Network Classifier. Builds a CNN with embeddings
     as input,
265     using Keras and Tensorflow.
266     Perform the training and prediction for all labels in one run.
267     train_X, train_y, test_X, test_y - input data transformed by word2vec
268     model - word2vec model
269     max_len - max length for embeddings
270     fold - actual cross-validation fold number
271     dirname - folder to save results, models, logs and predictions
272     epochs - number of epochs to be used for training the model
273     activation - activation function on output layer
274
275     returns - the predictions and times used to train the model and
    predict the results'''
276
277     times = []
278     embedding_layer = layers.Embedding(len(model.wv),model.wv.vectors.
    shape[1],
279                 weights = [model.wv.vectors], input_length=max_len,
    trainable = False)
280     input_model = layers.Input(shape=(max_len,),dtype='int32')
281     embedded_input = embedding_layer(input_model)
282     main_model = layers.Conv1D(model.wv.vectors.shape[1], kernel_size
    =(2,),
283                         activation='relu', padding='causal')(
    embedded_input)
284     main_model = layers.MaxPooling1D(2)(main_model)
285     main_model = layers.Conv1D(model.wv.vectors.shape[1], kernel_size
    =(3,),
286                         activation='relu', padding='causal')(main_model)
287     main_model = layers.MaxPooling1D(2)(main_model)
288     main_model = layers.Flatten()(main_model)
289     main_model = layers.Dense(model.wv.vectors.shape[1], activation='relu
    ')(main_model)
290     main_model = layers.Dropout(0.5)(main_model)
291     main_model = layers.Dense(3, activation=activation)(main_model)
292
293     CNN = keras.Model(input_model,main_model)
294     CNN.compile(loss='binary_crossentropy', optimizer='adam',metrics='
    accuracy')
295     print(CNN.summary())
```

```
296
297     # saving logs on tensorboard
298     logdir = dirname + "/logs"
299     if not os.path.exists(logdir):
300         os.mkdir(logdir)
301     logs = logdir + '/'+datetime.now().strftime("%Y%m%d-%H%M%S")
302     tensorboard_callback = TensorBoard(log_dir=logs, histogram_freq=1)
303
304     # Training CNN model
305     t0 = time()
306     history = CNN.fit(train_X,train_y,epochs=20,validation_data=(test_X,
        test_y),verbose=1,
307                 callbacks=[tensorboard_callback])
308
309     times.append(time() - t0)
310
311     # saving trained model
312     model_file = os.path.abspath(dirname+'/models/cnn_model_'+ str(fold)+
        '.model')
313     CNN.save(model_file)
314
315     # Predicting using CNN model trained
316     t1 = time()
317     preds = CNN.predict(test_X)
318     times.append(time()-t1)
319     times.append(sum(times))
320     # Transform results in 0 and 1 using 0.5 as cut
321     preds = (preds >= 0.5)*1
322
323     return preds, times
324
325 #save model as pickle file
326 def save_model(model,filename):
327     with open(filename, 'wb') as file:
328         pickle.dump(model, file)
329     return
```

A.4 wbs_predict.py - Running experiments with cross-validation

```
1 from wbs_features import *
2 from wbs_classifiers import *
3 import os
4 import pickle
5 from datetime import datetime
6
7 def save_results(results, filename):
```

```
    '''
    Save results in pickle format.
    results - dictionary. Times and predictions.
    filename - filename of file to be saved.
    '''
    # saving results in file
    with open(filename, 'wb') as file:
        pickle.dump(results, file)
    return dirname

def save_features(features, filename = './wbs_prediction/features/features_'):
    '''
    Save features as pickle file.
    features - dictionary format, train + test features and labels.
    filename - filename, string.
    return - None.
    '''
    # saving features in file, add date/time and pickle extension.

    with open(filename, 'wb') as file:
        pickle.dump(features, file)
    return

def create_storage_folders(dirname = './wbs_prediction'):
    '''
    Create folders to save results. If folder already exists, creates a new folder.
    dirname - directory to save results.    return - new dirname, string.
    '''
    if not os.path.exists(dirname):
        os.mkdir(dirname)
    else:
        dirname = dirname+datetime.now().strftime('%d%m%Y_%H_%M')
        os.mkdir(dirname)
    os.mkdir(dirname+'/models')
    os.mkdir(dirname+'/features')
    os.mkdir(dirname+'/results')
    return dirname


def split_train_test(data,n_test = 0.2):
    '''
    data - database in a pandas DataFrame format
    n_test - percentage of test samples to be used
    returns - three lists with test operation ids, test and train indexes on the DataFrame '''
    ids = data.ops_id.unique()
```

```
53      np.random.shuffle(ids)
54      n_test = int(n_test*len(ids))
55      # finding operation ids for the test set
56      test_ids = ids[:n_test]
57      # finding indexes for training set
58      train_index = data.loc[~data.ops_id.isin(test_ids)].index.values
59      # finding indexes for test set
60      test_index = data.loc[data.ops_id.isin(test_ids)].index.values
61
62      return test_ids, test_index, train_index
63
64  def split_train_test_cv(data,cv=5):
65      '''
66      Split train and test data by operation id on a cross-validation split
        .
67      data - database in a pandas DataFrame format
68      cv - cross-validation fold
69      returns - dictionary with test operation ids, test and train indexes
        on the dataframe
70      for all the cv folds '''
71      cv_split = {}
72      ids = data.ops_id.unique()
73      np.random.shuffle(ids)
74      n_test = len(ids)//cv
75      for i in range(cv):
76          # finding operation ids for the test set
77          test_ids = ids[i*n_test:(i+1)*n_test]
78          # finding indexes for training set
79          train_index = data.loc[~data.ops_id.isin(test_ids)].index.values
80          # finding indexes for test set
81          test_index = data.loc[data.ops_id.isin(test_ids)].index.values
82          cv_split[i] = {}
83          cv_split[i]['test_ids'] = test_ids
84          cv_split[i]['test_index'] = test_index
85          cv_split[i]['train_index'] = train_index
86
87      return cv_split
88
89  def split_data(X,y,test_index, train_index):
90      '''
91      Split features from database into train and test samples
92      X - features, numpy matrix.
93      y - labels, pandas DataFrame.
94      test_index - index for test samples in DataFrame.
95      train_index - index for train samples in DataFrame.
96      returns - test and train samples features and labels
97      '''
98      train_X = X[train_index]
```

```
99      test_X = X[test_index]
100     train_y = y.iloc[train_index]
101     test_y = y.iloc[test_index]
102     return train_X, train_y, test_X, test_y
103
104 def cross_validate_operations(X, y, data,cv=5,dirname='./wbs_prediction',
105                                feature_reduction = False, previous = False):
106     '''
107     Perform cross-validation on samples for all classifiers.
108     X - features, numpy matrix.
109     y - labels, pandas DataFrame.
110     data - database in DaraFrame format.
111     cv - cross-validation split (folds).
112     dirname - directory to save results.
113     feature_reduction - True/False. Perform feature reduction on samples
        in addition.
114     returns - None. Save times, predictions and models.
115     '''
116     previous = True
117     if previous:
118         old_results = read_pickle('../main/results_cv.pkl')
119         cv_split = {}
120         for i in old_results.keys():
121             cv_split[i] = {}
122             cv_split[i]['test_ids'] = old_results[i]['test_ids']
123             cv_split[i]['test_index'] = old_results[i]['test_index']
124             cv_split[i]['train_index'] = old_results[i]['train_index']
125         del old_results
126     else:
127         cv_split = split_train_test_cv(data,cv)
128     for fold in range(cv):
129         print('\nCross validation fold: ', fold)
130         results = {}
131         # finding operation ids for the test set
132         test_ids = cv_split[fold]['test_ids']
133         # finding indexes for training set
134         train_index = cv_split[fold]['train_index']
135         # finding indexes for test set
136         test_index = cv_split[fold]['test_index']
137         results['test_ids'] = test_ids
138         results['test_index'] = test_index
139         results['train_index'] = train_index
140
141         # Training and predicting all the models
142         pred = predict_all(X,y,test_index,train_index,test_ids,fold,
        dirname,False)
143         results['times'] = pred['times']
144         results['preds'] = pred['preds']
```

```
145         # Scoring models
146         print('Scoring models...')
147         test_y = y.iloc[test_index]
148         results['scores'] = score_all_classifiers(test_y,results['preds'
    ], data, test_ids)
149 
150         # saving results in file
151         filename = os.path.abspath(dirname+'/results/results_'+str(fold)+
    '.pkl')
152         save_results(results,filename)
153         print(results['scores'])
154 
155         # Repeat CV for feature reduction
156         # if feature reduction, uses chi2 with chi2 value 2 to select
    best features
157         if feature_reduction:
158             if not os.path.exists(dirname +'_reduced_'):
159                 dirname_red = create_storage_folders(dirname +'_reduced_'
    )
160             print('Applying feature selection')
161             # selecting best features
162             pred = predict_all(X,y,test_index,train_index,test_ids, fold,
    dirname_red,True)
163             results['times'] = pred['times']
164             results['preds'] = pred['preds']
165             filename = os.path.abspath(dirname_red+'/results/
    results_reduced_'+str(fold)+'.pkl')
166             results['scores'] = score_all_classifiers(test_y,results['
    preds'], data, test_ids)
167             save_results(results,filename)
168             print(results['scores'])
169 
170     return
171 
172 def score_all_classifiers(true_y, predictions, data, test_ids):
173 
174     # Dictionary of results
175     scores = {}
176     # Run classification for classifier in the set "classifiers"
177     for name, pred in predictions.items():
178         score = []
179         if '_' not in name:
180             aux = true_y == pred
181 
182             score.append(np.mean(aux.barrier))
183             score.append(np.mean(aux.primary))
184             score.append(np.mean(aux.secondary))
185             score.append(score_elements(true_y.values, pred))
```

```
186             score.append(score_operations(true_y,pred,data.loc[data.
    ops_id.isin(test_ids)]))
187             score.append(np.mean(aux.values))
188             scores[name] = score
189             #print('{} score: {}'.format(name, score))
190 
191     return scores
192 
193 # Score correct if all labels are classified correct: [1,0,1] == [1,0,1]
    -> True
194 def score_elements(true_y, pred):
195     return accuracy_score(true_y, pred)
196 
197 def score_operations(true_y, pred,test_data_X=[]):
198     ops = test_data_X.iloc[0]['ops_id']
199     n_ele = 0
200     scores = []
201     score = 0
202     for i,line in enumerate(pred):
203 
204         ops_now = test_data_X.iloc[i]['ops_id']
205         if ops != ops_now:
206             scores.append([score/n_ele, ops])
207             ops = test_data_X.iloc[i]['ops_id']
208             score = 0
209             n_ele = 0
210         if np.sum(pred[i] == true_y.iloc[i].values) == 3:
211             score += 1
212         n_ele += 1
213     scores = np.array(scores)
214     return np.sum(scores[:,0] == 1)/len(scores)
215 
216 # run classifiers on the set
217 # Load data if not passed as input
218 def predict_all(X, y,test_index,train_index,test_ids, fold='',dirname='./
    wbs_prediction',
219                 freduction=False):
220     '''
221     Perform train and prediction for all classifiers. Saves models for
    each classifier in
222     'models' folder.
223     X - features, numpy matrix.
224     y - labels, pandas DataFrame.
225     test_index - index for test samples in DataFrame.
226     train_index - index for train samples in DataFrame.
227     fold - actual cross-validation split.
228     dirname - directory to save results.
```

```
    feature_reduction - True/False. Perform feature reduction on samples if True.
    returns - Predictions and times for training and predictions. Saves models and
    features in folder.
    '''
    # Dictionary of results
    pred = {}
    pred['times'] = {}
    pred['preds'] = {}

    #split data
    train_X, train_y, test_X, test_y = split_data(X,y,test_index, train_index)
    #save features
    save_features({'train_X': train_X,'train_y':train_y,'test_X':test_X,'test_y':test_y,
                   'test_ids':test_ids,'test_index':test_index,'train_index':train_index},
                   dirname+'/features/features_'+str(fold)+'_')

    if freduction:
        # selecting best features
        train_X, test_X = select_best_features(train_X,train_y,test_X)
        classifiers = (('Naive Bayes', nb_classifier(train_X, train_y, test_X,fold,dirname)),
                ('SVM' ,svm_classifier(train_X, train_y, test_X,fold, dirname)),
                ('KNN',kn_classifier(train_X, train_y, test_X,fold, dirname)),
                ('Decision Tree', dt_classifier(train_X, train_y, test_X, fold,dirname)),
                ('Random Forest' , rf_classifier(train_X, train_y, test_X ,fold,dirname)),
                ('MLP',mlp_classifier(train_X, train_y, test_X,fold, dirname)),
                ('DL', dl_classifier(train_X, train_y, test_X,10,5,fold, dirname)))
    else:
        # create and split data using embeddings
        cnn_train_X,cnn_train_y,cnn_test_X,cnn_test_y, test_index,model, max_len = prepare_embeddings(

       test_index=test_index)
        save_features({'train_X': cnn_train_X,'train_y':cnn_train_y,'test_X':cnn_test_X,
                'test_y':cnn_test_y,'test_ids':test_ids,'test_index': test_index,
```

```
261                 'train_index':train_index},dirname+'/features/
    cnn_features_'+str(fold)+'_')
262         classifiers = (('Naive Bayes', nb_classifier(train_X, train_y,
    test_X,fold,dirname)),
263                 ('SVM' ,svm_classifier(train_X, train_y, test_X,fold,
    dirname)),
264                 ('KNN',kn_classifier(train_X, train_y, test_X,fold,
    dirname)),
265                 ('Decision Tree', dt_classifier(train_X, train_y, test_X,
    fold,dirname)),
266                 ('Random Forest' , rf_classifier(train_X, train_y, test_X
    ,fold,dirname)),
267                 ('MLP',mlp_classifier(train_X, train_y, test_X,fold,
    dirname)),
268                 ('DL', dl_classifier(train_X, train_y, test_X,10,5,fold,
    dirname)),
269                 ('CNN',cnn_classifier(cnn_train_X,cnn_train_y,cnn_test_X,
    cnn_test_y, model,
270                                       max_len,fold=fold, dirname=dirname,
    epochs = 129)))
271
272
273
274     # Run classification for classifier in the set "classifiers"
275     for name, classifier in classifiers:
276         print('\nClassifier: ', name)
277         preds, times = classifier
278         pred['times'][name] = times
279         pred['preds'][name] = preds
280
281     return pred
282
283 if __name__ == '__main__':
284     '''
285     Load database. Load features. Run 5-fold cross-validation for all
    classifiers and
286     classification using feature reduction.
287     '''
288     dirname = create_storage_folders(dirname = './wbs_prediction_original
    ')
289     data,y = load_database()
290     X = prepare_features(data)
291     cross_validate_operations(X,y,data,5,dirname='./
    wbs_prediction_original',
292                               feature_reduction = True)
```

Bibliography

[1] BjÃÿrn A. Brechan. *Framework for automated well planning and Digital Well Management*. PhD thesis, NTNU - Norwegian University of Science and Technology, 2020.

[2] S. J. Sparke, R. Conway, and S. Copping. The seven pillars of well integrity management: The design and implementation of a well integrity management system. Society of Petroleum Engineers, 2011. ISBN SPE 142449.

[3] Ying Zhao, Ting Sun, Jin Yang, Qishuai Yin, Hongshu Wei, Zhengli Liu, Zhong Li, and Yi Huang. Combining drilling big data and machine learning method to improve the timeliness of drilling. Society of Petroleum Engineers, 2019. ISBN SPE 194111-MS.

[4] Anisa Zhurda. Automated well monitoring: Machine learning and web application. Master's thesis, University of Stavanger, 2020.

[5] Hasan Asfoor and Walid Kaskas. Harnessing the power of natural language processing and fuzzy theory to improve oil and gas data management efficiency. Society of Petroleum Engineers, 2019. ISBN SPE 196259-MS.

[6] David CastiÃśeira, Robert Toronyi, Nansen Saleri, and Quantum Reservoir Impact. Machine learning and natural language processing for automated analysis of drilling and completion data. Society of Petroleum Engineers, 2018. ISBN SPE 192280-MS.

[7] ChengXiang Zhai and Sean Massung. *Chapter 2 - Text Data Understanding*. Association for Computing Machinery and Morgan and Claypool Publishers, first edition, 2016. ISBN 978-1-97000-116-7, paperback.

[8] Tomas Mikolov, Kai Chen, Gregory S. Corrado, and Jeffrey A. Dean. Computing numeric representations of words in a high-dimensional space, 2013. URL https://patents.google.com/patent/US9037464B1/en. US patent number US9037464B1.

[9] NORSOK. Norsok Standard D-010, Revision 5, January 2021. www.standard.no/petroleum, 2021.

[10] CSB Chemical Safety and Hazard Investigation Board. Macondo blowout and explosion. https://www.csb.gov/macondo-blowout-and-explosion/, 2016.

[11] Larry W. Lake and Robert F. Mitchell. *Petroleum engineering handbook : Volume II, : Drilling engineering*. Society of Petroleum Engineers, Richardson, TX, USA, first edition, 2006. ISBN 1-55563-332-3, 978-1-55563-332-5.

[12] NORSOK. Norsok standard d-010, revision 4, june 2013. www.standard.no/petroleum, 2013.

[13] Pandas. Pandas documentation. https://pandas.pydata.org/pandas-docs/stable/index.html, 2021.

[14] F. Pedregosa, G. Varoquaux, A. Gramfort, V. Michel, B. Thirion, O. Grisel, M. Blondel, P. Prettenhofer, R. Weiss, V. Dubourg, J. Vanderplas, A. Passos, D. Cournapeau, M. Brucher, M. Perrot, and E. Duchesnay. Scikit-learn: Machine learning in python. *Journal of Machine Learning Research*, 12:2825–2830, 2011.

[15] Yujia Zhai, Wei Song, Xianjun Liu, Lizhen Liu, and Xinlei Zhao. A chi-square statistics based feature selection method in text classification. In *2018 IEEE 9th International Conference on Software Engineering and Service Science (ICSESS)*, pages 160–163, 2018. doi: 10.1109/ICSESS.2018.8663882.

[16] Language Technology Group at the University of Oslo. Nlpl word embeddings repository, 2021. URL http://vectors.nlpl.eu/repository/.

[17] Yoon Kim. Convolutional neural networks for sentence classification. 2014. ISBN 1408.5882v2.

[18] Ye Zhang and Byron C. Wallace. A sensitivity analysis of (and practitionersâĂŹ guide to) convolutional neural networks for sentence classification. 2016. ISBN 1510.03820v4.

[19] Martín Abadi, Ashish Agarwal, Paul Barham, Eugene Brevdo, Zhifeng Chen, Craig Citro, Greg S. Corrado, Andy Davis, Jeffrey Dean, Matthieu Devin, Sanjay Ghemawat, Ian Goodfellow, Andrew Harp, Geoffrey Irving, Michael Isard, Yangqing Jia, Rafal Jozefowicz, Lukasz Kaiser, Manjunath Kudlur, Josh Levenberg, Dandelion Mané, Rajat Monga, Sherry Moore, Derek Murray, Chris Olah, Mike Schuster, Jonathon Shlens, Benoit Steiner, Ilya Sutskever, Kunal Talwar, Paul Tucker, Vincent Vanhoucke, Vijay Vasudevan, Fernanda Viégas, Oriol Vinyals, Pete Warden, Martin Wattenberg, Martin Wicke, Yuan Yu, and Xiaoqiang Zheng. Tensorflow: Large-scale machine learning on heterogeneous systems, 2015. URL https://www.tensorflow.org/. Software available from tensorflow.org.

[20] François Chollet et al. Keras. https://keras.io, 2015.

[21] Github repository., 2021. URL https://github.com/daianjos/wbs.