Imperial College London

MEng Individual Project

Imperial College London

Department of Computing

Data Mining Calibration Points from Oilfield Documents

using Natural Language Processing and Machine Arguing

Author:
Athithan Dharmaratnam

Supervisor:
Prof. Lucia Specia

Second Marker:
Dr. Pedro Baiz

June 15, 2020

Abstract

Daily Drilling Reports where drilling activities are entered as free text contain a wealth of information which is underutilised. Manual screening of Daily Drilling Reports (DDRs) is a tedious time-consuming exercise undertaken by engineers where only one particular focus is extracted leaving potential data insights undiscovered.

This project investigates one such data insight called calibration points which are extracted from similar wells in a field in order to generate Mechanical Earth Models (MEMs) for downstream field planning. This project demonstrates a complete automated workflow from taking in the raw DDR data sets to producing a calibration point output file using data mining with natural language processing techniques and machine arguing for feature analysis. The application framework forms an ensemble system which has been implemented as three stages: extracting features from free text, mapping normalised features using semantic triples to argumentation cases and formulating an explanation for a data insight using abstract argumentation for case-based reasoning.

The implemented workflow improves upon a prior lower bound method for calibration point extraction and shows results which exceed the manual benchmark for calibration point extraction for some calibration points. This project has novel contributions in demonstrating a practical implementation of machine arguing research in a new domain. The three-stage application framework while applied to calibration points, can generalise to any data insight for textual data given appropriate domain inputs and engineered features for the new problem domain.

Acknowledgements

I would like to thank my project supervisor Professor Lucia Specia for her advice and support throughout the course of this project. Thank you for helping me to widen my horizons to the diverse field of Natural Language Processing.

I would also like to thank Schlumberger and the supporting team who are my manager, Francisco Gomez and my domain expert, Ivan Diaz Granados. It is only due to your continual support and valuable domain knowledge that I was able to complete this project.

Special thanks goes to my friends who kept me sane during the lockdown period. I also greatly appreciate my sister Abina for her moral support and her kind offer to proof read my rather long report. Finally, I would like to thank my mother without whom I would have never made it so far in my education. She has always and will always be my strongest support in all the challenges I face.

Contents

List of Figures

List of Tables

Chapter 1

Introduction

1.1 Motivations

Oil companies have a strong incentive in acquiring different kinds of data necessary to understand their reservoirs and produce hydrocarbons more efficiently [DS19]. A report by Mckinsey has shown that drilling and completion operations constitute 40% to 50% of the expenditure of an average offshore Oil and Gas operator [ABJ15]. Therefore, companies have a financial incentive to exploit the wealth of historical drilling activity data in order to reduce the cost of drilling activities. Improving drilling operation efficiency can be done through better planning to identify potential risks for similar offset wells [ST10]. In addition, historical data in combination with real-time data from the well can be used to identify latent patterns to make predictions for non-productive time drilling events. [SCS15].

This project aims to provide an automated solution to analyse calibration points from daily drilling reports which are a 24-hour record of drilling activities (defined formally in Section 2.1.1) using Natural Language Processing techniques [MM20]. The automated solution will be broadly designed such that it can be generalised to extract other data insights from free text in different oilfield documents. The daily drilling report contains free text comments which may contain calibration points which can be used to calibrate mechanical earth models. A mechanical earth model is used in the planning phase of oil field development and calibration points increase the accuracy of the model. A more accurate model mitigates the risks of non-productive time drilling events and thus increases the efficiency of the drilling operation [Afs+09].

Extracting calibration points from daily drilling reports is traditionally a manual exercise conducted by a domain expert done as part of a review before planning a new oil well using best practices [NS19]. This process is quite time-consuming often taking weeks for a single offset well and is also a tedious process. A complete analysis is often not possible due to time constraints which results in plenty of missed data insights as the analysis focus is normally limited to a subset of drilling events in the case of calibration points extraction. A study by Brule has shown that generally, half an engineer's and geo-scientist time is spent trying to collate unstructured data into a structured format in order to be able to extract useful data insights traditionally done through manual analysis [BI15]. This is further supported by a survey conducted at Schlumberger which shows that 60% to 80% of project time for the extraction of calibration points is utilised for data collection rather than more valuable data analysis [DS19]. Therefore an automated solution is preferred as it will reduce the time taken for offset well analysis and will potentially unearth unexpected data insights which can be reused in similar problems. Therefore the aim of this project is to create an automated workflow that ingests free text data from daily drilling reports to extract calibration points.

1.2 Objectives

The complete workflow for this project is made up of four modules in a linear flow as shown in Figure 1.1. The high-level workflow represents a complete solution starting from the raw data inputs to the presentation of the data insights in a user-friendly form.

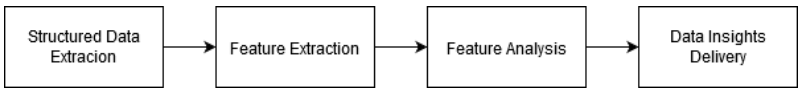


Figure 1.1: High-Level Workflow

However, the core focus of the research in this project is the three-stage implementation workflow in the feature extraction and feature analysis modules shown in Figure 1.2. Structured data extraction and data insights delivery are not fully developed in this project as they fall out of the scope of the research goals of this project but the general workflow of these modules will still be outlined for future research.

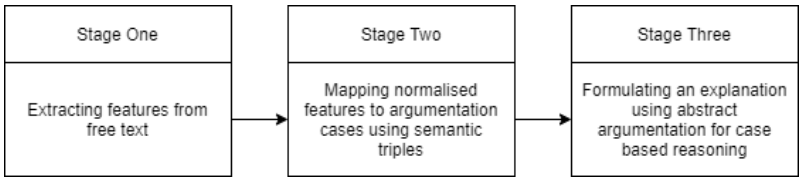


Figure 1.2: Three-Stage Implementation Workflow

1.2.1 Structured Data Extraction

The first step of the workflow is to extract the free-text comments into a structured format. The daily drilling reports come in a variety of formats depending on the company and date of publication as shown in the Appendix A. Formats include PDF, scanned documents, WITSML, CSV or stored in databases within internal company servers. Given the breadth of input formats, structured data extraction poses a significant challenge in handling the various different file formats. Furthermore, the challenge escalates when considering the lack of a standardised industry format of a daily drilling report between companies and drilling periods.

The primary goal of this project is to analyse calibration points from free-text comments. Therefore, this problem will be shelved as the current data sets (described in Section 2.2) already have free-text comments in a structured CSV format.

1.2.2 Feature Extraction

The second step would be to extract useful features from the comment data. The free-text comments in the data sets exist in a structured format but they have to be pre-processed in order to standardise them into a consistent internal format across different input data sets. The data quality challenges (defined in Section 1.3) have to be handled in the pre-processing of the raw comment data. There are many techniques for pre-processing the data (defined in Section 2.3.4) and different feature extraction methods such as noun phrase chunking and measurement parsing will have different degrees of pre-processing conducted on the raw text data.

The first feature extraction method that can be used is deep learning. This approach is a coarse approach as it seeks to directly extract the drilling event from lightly pre-processed text data. This involves using a neural network architecture such as sequence classification with BERT to take the pre-processed text data as an input and then output a binary classification of a possible drilling event. Given the drilling event sparsity problem it is more efficient to start with a single drilling event before moving on to categorical classification of at most fifteen potential drilling events. This method has been experimented with but not integrated into the final application due to problems with the labels in the training data set being mislabelled.

The second feature extraction method that can be used is a pseudo-dimensionality reduction method to reduce the pre-processed text tokens to more relevant tagged entities. This approach is a fine-grained analysis approach on the textual data. This process can involve using domain-specific Named Entity Recognition (NER) model to extract relevant named entities in the pre-processed text data. However, this project does not have access to a domain-specific labelled data set for the training of the NER model. Standard NER libraries have been trained to extract general entities but the focus of this project is heavily reliant on oilfield domain-specific entities therefore the NER approach is not feasible within the scope of this project. Another approach is to use phrase chunking to extract noun phrase features as a substitute to named entities. The factual descriptive nature of the drilling comments means that particular noun phrases, which will be referred to as domain phrases in this project, are likely to co-occur with drilling events.

The extracted features from the aforementioned feature extraction method will be used in combination with measurement entities extracted with a syntactic parser. Measurement data is more standardised than variable textual data so a custom parser will be adequate to extract numerical values and unit values. The unit definition will be defined externally using a standard oilfield unit glossary.

1.2.3 Feature Analysis

Having extracted relevant features for stage one of the three-stage workflow shown in Figure 1.2, the next step is to analyse them to extract calibration points with an explanation of the reasons behind the extraction. The feature analysis module contains two important research areas which are entity linking and abstract argumentation for case-based reasoning.

The entity linking system provides a way of removing some of the data quality challenges such as disparate comment data. It acts as a way to normalise certain features such as the extracted noun phrase so that the features can be comparable to other features. The normalised features can then be mapped into cases for the argumentation framework using semantic triples from the Resource Description Framework [RDF14]. The feature converted cases can be inputted into a pre-defined argumentation framework model which will logically ascertain the presence of a calibration point while providing the reasons why it occurred through the arguments satisfied. The predefined argumentation framework model will rely heavily on domain knowledge provided by a domain knowledge expert. A recent study suggests one of the main points for a successful Big Data project is the collaboration in an interdisciplinary team of computer scientists and petroleum engineers [MT18]. The choice of the argumentation framework model will vary dependent on the features that can be extracted from the textual data but will initially be an abstract argumentation framework.

1.2.4 Data Insights Delivery

A non-core aspect of the workflow is delivering the data insights extracted from the analysed features in a user-friendly interface. The complete workflow would constitute a complete prototype and hence delivering and more importantly explaining the extracted calibration points to a client is a key aspect in an industry project. However, since the main research goals of the project are quite labour intensive this aspect of the high-level workflow will be limited to generating a calibration point file with generated explanations for the scope of this project.

1.3 Challenges

Data quality is often the main issue with machine learning projects especially in the exploration and production industry [CA14]. Furthermore, the data sourcing and preparation often takes up a significant amount of project time as most of the untapped data is unstructured and distributed in different formats and files [JG15]. The main sources of unstructured data in the oil and gas industry are well logs, daily drilling reports and CAD drawings [MT18]. In order to extract useful data insights from this unstructured data, it needs to be streamlined into a consistent structured format for a program to analyse. The only industry-standard format for daily drilling reports is WITSML [Ene17]. However, most companies have PDFs of daily drilling reports with varying structures

which pose a challenge to extract the relevant comment data from in a generalised manner.

The main challenges involved with this project are listed below:

1. **Disparate comment data**
   The writing style of the free-text comments varies between company, era and personal. The quality of the comments ranges from complete English sentences to noisy error-prone sentences as shown in Appendix B.

2. **Disparate metadata**
   The contents of daily drilling reports are varied between companies. Different companies and operators have varying concerns and standards which leads to the daily drilling report having varying metadata bar the core operations table of the drilling operations as shown in Appendix A. The metadata is useful as a form of quality control and additional features for the data analysis but is limited in usefulness given the lack of consistency across different data sets.

3. **Data Sparsity**
   The drilling events are an important target to extract. However, these events are often sparse within a single well history. Calibration points are normally extracted at the occurrences of drilling events which normally represent an unwanted event in the operation of the drilling well. Therefore, these drilling events often do not occur regularly which leads to data sparsity issues when using some methods especially unsupervised machine learning methods such as topic modelling.

4. **Noise**
   Pre-processing is a key aspect of this task as there are artefacts within the provided data sets provided after having been parsed into a structured format in some data sets. After removing the irrelevant artefacts, there still exists valid text which provides semantic noise and thus invalidates certain naive methods from being used such as keyword indexing [CA14] [NS19].

5. **Uncertainty**
   The reports are written by humans and thus are prone to human error which creates uncertainty especially given that some of the measurements taken are subsurface [MT18] [Ene17].

6. **Distributed data**
   Existing daily drilling reports are stored in legacy systems which have to be retrieved [DV14].

7. **Technical Symbols**
   The contents of the comments are filled with useful technical symbols such as measurements. However, for some processes such as noun phrase chunking, these technical symbols are noise and need to be removed.

8. **Acronyms**
   One issue is the prevalence of acronyms used by the operators when writing the daily drilling reports. Acronyms can be expanded in order to increase the accuracy of some processes but the expansion process may be flawed which leads to issues in other processes.

9. **Missing Data**
   Some daily drilling reports do not record key information and there may be gaps in the reports which need to be inferred.

10. **Data Volume**
    The volume of the data in some of the data sets is quite large reaching hundreds of thousands of comments. Intermediate storage of the results from the various stages in the pipeline will be necessary to reduce processing time.

11. **Surface Mention Ambiguity**
    There are a lot of terms within the sentences which have multiple meanings and without contextual information, it is difficult to understand the correct sense of the word [All88].

Chapter 2

Background

2.1 Oilfield Domain Knowledge

This project involves data mining in the oil and gas industry. Therefore, an understanding of the oilfield domain is necessary to develop some of the pipelines such as the abstract argumentation pipeline. The main oilfield domain knowledge necessary is the understanding of the manual process of the extraction of calibration points from daily drilling reports to define the cases for the argumentation framework for case-based reasoning.

2.1.1 Daily Drilling Report

A daily drilling report is an industry-standard report of drilling activities on the drilling rig. It is a 24-hour summary report of the prior day's operations to keep the interested parties aware of the operations and issues on the rig [MM20]. It is used for a variety of needs such as logging drilling data or tracking drilling performance dependent on the operator [IAD18]. The format of the daily drilling report has been paper form for decades resulting in historical daily drilling reports being scanned PDF. Daily drilling reports in recent decades have evolved to be more digital as the industry tries to move towards the goal of smart reporting [Ene17]. The current generation of the daily drilling report is now a combination of automatic inputs from sensor data and manually inputted comments about the drilling activities. Daily drilling reports come in many different formats dependent on operation area, drilling type and company culture.

Appendix A shows daily drilling report samples where Figure A.1 is a sample from the public Australian dataset, Figure A.2 is a sample from a geothermal well and Figure A.3 is a historical scanned sample.

Appendix B shows the daily drilling report comment samples from three of the data sets used. Figures B.1 and B.2 for the Schlumberger Training data set show the noisy nature of the comments. Figures B.3 and B.4 represent a more formal language used when writing operational comments for the Public Australian data set. Figures B.5 and B.6 show the short nature of some comments from the Schlumberger Version Two Data Set.

While the contents vary, there are some common aspects which are relevant to the project listed below [19].

- **Report Date**
  The daily drilling report is generated in a 24-hour period and hence this is useful identifying metadata.
- **Well Name**
  The format of the well name is dependent on the company and can be either an internal name or a legal well name but is nonetheless useful identifying metadata.
- **Header Summaries**
  The header of the report contains other useful summary information such as measured depth

and true vertical depth which can be used as quality control parameters for the extracted measurements in the drilling report summary.

- **Blowout Preventer**
  Some reports have a section to record tests on this safety device which is vital to monitor lead-ups to a catastrophic failure event called a blowout.

- **Mud**
  Mud tables listing various properties of the mud in the drilling operation are useful as mud weight is an important calibration point and can be used to quality control the automated measurement extraction. Furthermore, fluid loss recorded in the mud tables may also be a measurement which is useful to validate drilling event such as losses.

- **Time Breakdown Section**
  This is the key aspect of the daily drilling report for this project. This section constitutes around a quarter of the report but takes half the effort to generate as it is done manually [Ene17]. It is usually in the form of a table with varying columns dependent on the operator. The target column is the operation comments which are free text comments written by the rig supervisor summarising the drilling activities. The operation comments are the primary target to analyse using the implementation workflow to extract calibration points.

2.1.2 Calibration Points

Calibration points are used to calibrate parameters of a mechanical earth model which is "a numerical representation of the state of stress and rock mechanical properties for a specific stratigraphic section in a field or basin" [Knö16]. A mechanical earth model is a useful model generated to better understand the geomechanics in the well construction region to reduce potential drilling risks and thus forms an essential part of well planning [Afs+09]. The calibration parameters for the mechanical earth model can be mined from drilling data including daily drilling reports. The parameters are normally in the form of a drilling event and a corresponding measurement value mined from the operation free-text comments. Drilling events such as a leak off test or formation integrity test in combination with measurements such as mud weights are an example of a calibration point.

Drilling Events

Drilling events are extreme events that may occur during drilling activities. A total of fifteen drilling events have been specified for extraction by the domain expert but within this project, only six of them will be considered. These six drilling events have been chosen as they are the most valuable events for a calibration point and they occur in the public Australian data set and thus can be evaluated. It is worth noting that the majority of the fifteen events do not occur within some data sets making a supervised method to extract these drilling events difficult. Furthermore, the general definition of these drilling events may vary between different operators and engineers.

- **Formation Integrity Test** "is a test of the strength and integrity of a new formation" [AS05]. This measurement is especially useful as it is a field measurement so it can be used for field development planning of offset wells. This test usually results in non-productive time so it does not occur regularly.

- **Kick** "is a well control problem in which pressure found within the drilled rock is higher than the mud hydrostatic pressure action on the borehole" [20b]. Kicks are usually a result of insufficient mud weight and therefore an important calibration point.

- **Cavings** are loose debris that arise from drilling operations but are not removed with the drill bit [20a]. They occur in various shapes and are recorded as drilling events often being an indicator of well bore instability [Osi12].

- **Stuck Pipe** is a drilling event where a "pipe cannot be freed from the hole without damaging the pipe" [OBM99]. This is an especially costly drilling event normally due to related lost circulation events which can be used as early warning indicators.

- **Losses** "is the uncontrolled flow of whole mud into a formation" [20d]. There are varying degrees of lost circulation that can occur during drilling operations and the occurrence of lost circulation can be a good indicator of formation issues for calibration points.
- **Tight Hole** "is a section of a well bore where drilling tools with large diameters face resistance" [20e]. The occurrence of a tight hole is an indicator of well bore stability problems due to formations.
- **Leak Off Test** "is a test carried out to determine the pressure of fracture of an open formation" [20c]. Leak off tests happen very rarely but should occur at least once during the operation of a drilling rig and the measured value is the mud weight for the test. Leak off tests are not labelled in the Public Australian data set and therefore are not evaluated but are still an important drilling event to consider as they are closely related to formation integrity tests.

Measurements

In addition to the drilling events, measurements are also necessary for a calibration point. Measurements can be any numerical data in the free text comment but with regard to calibration points, the key measurements are mud weight, depth and time. In some daily drilling reports, time and depth are directly noted down as a column in the operation table or they may be embedded in the free text. The units used for the measurements vary between regions and also the format of the measurements differ in notation. One example is using commas to denote thousands in one data set which may not be used in other data sets.

2.2 Datasets

This project is done in collaboration with an oilfield service company called Schlumberger who have provided company data sets as well as the help of a domain expert in understanding the domain associated with the problem. The data sets used for this project will be operational comments of the drilling activities with some identifying metadata in a CSV format. While daily drilling reports are normally in a PDF format, this project mainly seeks to research into the feature extraction and analysis aspect of the core workflow listed in Section 1.2. Therefore, the data sets initially used in this project will be structured and already extracted from PDF form. The current list of data sets used in this project is shown below.

- **Schlumberger Training Data Set** This data set contains around 56,000 comments across 146 wells with drilling event labels for six drilling events. This data set will be used as the training data set for the application.
- **Public Australian Data Set** This data set contains around 6,000 comments for 11 wells with drilling events labels for six drilling events. This data set will be used as a test data set for evaluating the accuracy of calibration point extraction.
- **Schlumberger Version One Data Set** This data set contains around 32,000 unlabelled comments from North Sea Wells. This data set will be used as a test data set for evaluating the coverage of calibration point extraction.
- **Schlumberger Version Two Data Set** This data set contains around 300,000 unlabelled comments from North Sea Wells. This data set will also be used as a test data set for evaluating the coverage of calibration point extraction.

2.3 Data Mining

Data Mining is the study of extracting data insights using processes such as data collection, data cleaning and data analysis. The data insights extracted are dependent on the application-specific goals but generally need to be concise and actionable. The process of data mining works in a pipeline where each stage transforms the data into a more useful format. Real-world data is often unstructured and noisy. Therefore, the majority of project time is often devoted to data preprocessing before any analysis work can take place. Given the breadth of the field, data mining

is an umbrella term and can be applied to a variety of domains [Agg15]. In the context of this project, data mining will be utilised in the oil and gas domain.

One also needs to consider the type of data, which in this case is dependency-oriented data as there exists a temporal relationship between the free-text comments. The free-text comments are a discrete time series data type as each comment is an ordered sequence of described operational drilling activities. The format of the text data as an input is a string but some language processing models such as deep learning require the text data to be converted into a vector space representation such as a document term matrix or word vectors. In addition, the data is multidimensional as there are contextual attributes which define the free text comment such as a time range or a spatial value (depth).

Another issue to consider with data mining projects is scalability issues especially with the large volume of data in some of the data sets. The data in this project is static once collected so some of the issues such as concept drift will not be relevant, especially considering that the target data sets are historical daily drilling reports [Agg15].

2.3.1 Pipeline Processes

The general stages of the data mining process are shown below [Agg15].

- **Data Collection** is the process of aggregating the relevant data sources for the project. In the context of this project, this process has mainly been reformatting the aforementioned data sets into the correct input format. The formatted data sets collected can then be stored in a database.
- **Data Pre-Processing** is the process of transforming the collected data into a more suitable application-specific format for feature extraction. Real-world data is often noisy and needs to be reduced into a token format for processing. Data quality challenges such as missing data also have to be tackled in this stage. The output of this process will be a structured data set which should be uniform across all input data sets. Data transformation is also an important consideration as some data needs to be converted to a more standardised form such as acronyms.
- **Feature Extraction** is an application-specific process but involves extracting informative features in the processed data set. The data is normally high dimensional which may not be favourable to some data mining methods and hence feature extraction can act as a dimensionality reduction method. There will be a lot of irrelevant data in the data set which hold no value to the target data insights so filtering them out in this stage reduces processing time downstream and also increases the accuracy of the final extraction.
- **Feature Analysis** is also application-specific and for complex applications can consist of several language processing models in the form of building blocks which contribute to an overall ensemble system. This project has several language processing models for feature extraction which form an ensemble system where the outputs of the feature extraction process are aggregated and analysed with the abstract argumentation framework. The output of this stage is normally the final data insights given to the interested party to evaluate which in this project is the domain expert.
- **Feedback Loop** this is an important stage conducted by the domain expert where the data insights are evaluated and improvements to the previous stages are implemented to generate better quality data insights.

2.3.2 Data Analysis Methods

There are various techniques that data mining encompasses that can be applied to the extracted features in feature analysis listed below [Agg15].

- **Association Pattern Mining** is a form of mining that seeks to identify trends within entities with the most popular version being frequent pattern mining. In the context of this project, once the features have been extracted it would be useful to identify patterns

of extracted features in relations to calibration points using association rule mining across and within comments. This stage can be used as an intermediate step to produce automated cases for other language processing models.

- **Data Classification** is the main focus of the project where the extracted features are mapped to particular drilling events. This is a supervised problem as the mapping of a set of features have to be learned with training data both with machine learning methods or with domain knowledge using the feedback loop.

2.3.3 Exploratory Data Analysis

Exploratory data analysis is an approach to exploring data sets [Fil13]. It involves a variety of techniques in order to gain a better understanding of the data sets to guide the development of the application through the direct insights discovered. Techniques include testing underlying assumptions, uncovering underlying structures and detecting outliers and anomalies. Data surveying is an important part of the project especially in the oil and gas industry where there are a lot of challenges with data quality [ZH03]. It helps survey whether there is sufficient information within the data set to answer the problem statement which in this project is to extract calibration points. There are several techniques that can be used to visualise the data listed below [Li19].

- **Data Set Summary Statistics** provides low-level information regarding the data set which helps estimate processing times as well as get basic statistics of the data set for comparison
- **Uni-variate visualisations** examine a single characteristic and are mainly frequency charts such as histograms based on feature counts. It is helpful to analyse the distribution of extracted statistical features across the different levels of the data set to identify patterns or areas of interest in subsets of the data set.
- **Bi-variate visualisations** examine the correlation between two features on a single plot and identify relationships which can be further explored to generate a better hypothesis. The focus for the feature comparison would be the target drilling event if available, the location and also report date to see if there are patterns across sequential comments using time series plots.

2.3.4 Pre-Processing

There are several pre-processing steps shown below that can be used in order to clean the raw free text comments before either converting them into vector form through word embedding or to process them further into tokens for feature extraction [Gan19].

- **Stop Word Removal** is the removal of frequent words that have no value in relation to the extraction task. The stop words are normally defined in an external lookup list and includes terms such as conjunctions or prepositions.
- **Lower Casing** is a standard pre-processing step where the input textual data is converted into lower case. Some string matching algorithms rely on the comparison being in the same case so this step is essential.
- **Stemming** is a form of normalising words by reducing them to their common base form. Stemming is a heuristic process that removes the end of the word such as by cutting of prefixes. This aggressive method runs the risk of the word losing its original meaning [MRS08].
- **Lemmatisation** is also a form of normalising words by reducing them to their common base form. However, lemmatisation aims to reduce the inflexion endings of words to return the lemma form of a word [MRS08].
- **Punctuation Removal** involves removing any punctuation symbols in the text. This process is useful as some processes such as phrase chunking would yield better results without punctuation noise in the text.
- **Artefact Removal** is the process of removing noise in the text that may come about as a result of converting between different text formats. These artefacts hold no value to the extraction process.

- **Text Augmentation (Part of Speech Tagging)** augments the text with useful features that help other processes such as phrase chunking. Part of speech tags are the grammatical part of speech labels assigned to each word in the input text.

2.4 Named Entity Linking

Named Entity Linking is the process of linking named entity mentions to entities stored in a knowledge base and is normally used as a form of data enrichment. Named Entity Linking is defined in three-stages which are candidate entity generation, candidate entity ranking and unlinkable mention prediction. Each stage is designed to successively filter out candidate entities for a single surface mention in the text until the last stage where the best candidate entity is outputted or NIL is outputted if the candidate entity doesn't have an adequate confidence score. Named entity mentions are usually entities extracted from the free text using Named Entity Recognition. In this project, named entity linking is used with noun phrases and pre-defined domain phrases due to the problems described in Section 1.2.2 and will thus be referred to as Entity Linking.

Wei Shen et al provides a survey of entity linking that can be found in the paper titled "Entity Linking with a Knowledge Base: Issues, Techniques, and Solutions" [SWH15]. This survey defines all the processes of entity linking as well as the algorithms used for each stage in entity linking which are described in more detail in Section 4.8 with a practical example implementation.

2.5 Abstract Argumentation for Case-Based Reasoning

One of the key problems with machine learning problems is that with methods such as deep learning, it is hard to explain the results of the output of the neural network. Abstract argumentation tackles this problem by helping understand why an outcome has been reached for a framework consisting of a set of arguments with relations defined between the arguments. Decision trees also implement a level of explainability through the decision nodes evaluated when traversing the tree but lack the natural point and counterpoint nature of arguments that machine arguing frameworks provide.

Dung defines an argumentation framework in his paper titled "On the acceptability of arguments and its fundamental role in non-monotonic reasoning, logic programming and n-person games" published in 1995 [Dun95]. The basis of abstract argumentation framework used in this project including the definitions of the set semantics such as the grounded extension has been defined in this paper.

Since then many extensions have built upon this original work including the main extension used in this project which has been defined in a paper titled "Abstract Argumentation for Case-Based Reasoning" by Kristijona Cyras et al published in 2016 [CST16]. The definitions of the case-based reasoning framework can be found in this paper including the formal definitions of the rules such as concision used to construct the framework as implemented in the project.

The implementation of the abstract argumentation for case-based reasoning framework described in Section 4.9 provides a practical example of the framework and a more in-depth description of the details defined in these papers and thus provide a better understanding of the concepts behind abstract argumentation.

2.6 Evaluation Metrics

There are several standard evaluation metrics used in order to evaluate the success of the project shown below:

- **True Positive (TP)** is an outcome where the model correctly predicts the positive class [Goo20c].
- **False Positive (FP)** is an outcome where the model incorrectly predicts the positive class [Goo20c].

- **False Negative (FN)** is an outcome where the model incorrectly predicts the negative class [Goo20c].
- **True Negative (TN)** is an outcome where the model correctly predicts the negative class [Goo20c].
- **Confusion Matrix** is a NxN table which shows the correlation between the true label and the model's classification where N represents the number of classes [Goo20d]. One axis of the matrix is the predicted label and the other is the actual label.
- **Precision** is the proportion of positive identifications that were correct as shown in the formula below [Goo20b]:

$$Precision = \frac{TP}{TP + FP}$$

- **Recall** is the proportion of actual positives that were identified correctly as shown in the formula below [Goo20b]:

$$Recall = \frac{TP}{TP + FN}$$

- **F1-Score** is the harmonic mean of the precision and recall score as shown in the formula below [Wik20a]:

$$F_1 = 2 * \left(\frac{Precision * Recall}{Precision + Recall} \right)$$

- **Matthews Correlation Coefficient** is a measure of the quality of binary classifications which is used when classes are of different sizes defined in the formula below [Wik20b]:

$$MCC = \frac{TP * TN - FP * FN}{\sqrt{(TP + FP)(TP + FN)(TN + FP)(TN + FN)}}$$

- **Classification Accuracy** is the fraction of predictions that the model correctly classified as shown in the formula below [Goo20a]:

$$Accuracy = \frac{TP + TN}{TP + TN + FP + FN}$$

Chapter 3

Literature Review

Analysing documents using text mining techniques is a mature field in industries such as the Health Industry where a survey conducted shows increasing publications of text mining research papers in the last decade especially in cancer research [Luq+19] [ARG17]. In the Oil and Gas Industry, text mining is a developing field having garnered more research interest in recent years especially in the analysis of daily drilling reports to extract various entities such as depths, hole sizes and drilling events [NS19] [Hof+18]. An analysis of related research into text mining in the Oil and Gas Industry using different methodologies is summarised in the following section.

3.1 Supervised Learning

This work uses supervised learning methods and is titled "Using Classified Text and Deep Learning Algorithms to identify risk and provide early warning" [Bre16]. This work is not specifically restricted to the oil and gas industry as it aims to create a text classifier to identify risks in internal communications which could lead to potential litigation. The interesting point regarding this work is that it tries to create a deep learning algorithm which learns the latent pattern which leads to litigation with the aim to proactively avoid it through an early warning system.

The author utilises subject matter experts in order to find appropriate labelled data sets to train a deep learning algorithm for a chosen category of risk. The data set was then processed to reduce the amount of unnecessary metadata in the documents to reduce the noise in training data. The authors augment the training data with unrelated negative data (e.g. random text from Wikipedia) which do not overlap with other categories but allows the system to have a more representative accuracy score as the output. The author encodes the words using word vectors and therefore the distance between the positive category events and the negative category events will be a sufficient metric for an accuracy score.

The choice of neural network architecture is a recurrent neural network. The author utilises the Receiver Operating Characteristic graph to calculate the Area Under the Curve score as this score gives a good indication of true positives vs false positives which is an important measure in this work. The author trains several recurrent neural networks for binary classification of the presence of the risks in test data. This methods of having several neural networks may be more beneficial than one neural network with categorical classification. However, the author did not explore this hypothesis. The author also suggests a continual machine learning system such that when risks are flagged, subject matter experts can validate these risks as false or true positives which when sufficient data has been collected can be used to augment the training data with client-specific data.

The author notes that in order for the deep learning algorithm to generalise well, the training samples have to be in the thousands. In experimental results, the author trained the deep learning algorithm with increasing training data sizes and found that the number of false positives decreased proportionally.

Another work explored deep learning methods as well in the context of daily drilling reports and is titled "Sequence Mining and Pattern Analysis in Drilling Reports with Deep Natural Language

Processing" [Hof+18]. This work advances on the previous methods this time using deep learning methods in order to classify sentences in the daily drilling report with three labels (event, symptom and action). The authors suggest that the use of deep learning is beneficial over traditional n-gram statistics for text mining as deep learning provides greater advantages in finding a latent semantic relationship in the corpus for a greater volume of data.

The authors conducted exploratory data analysis on the data set they received in the context of analysing reports with regards to productive time or non-productive time. Non-productive time is a good indicator of the occurrences of drilling events. The authors found that 63% of wells analysed had non-productive time drilling reports. Furthermore, the authors noted that the style of reports varied between productive and non-productive time reports. This work also uses standard pre-processing techniques in addition to custom ones such as regular expressions to handle the noisy symbols in the text corpus.

In order to utilise deep learning methods, the authors manually labelled extracted sentences from the corpus with the three aforementioned labels which resulted in an unbalanced labelled training dataset with a greater weight towards the action label. Furthermore, for the input to the deep learning neural network, the authors utilised word embedding to create word vectors as input features. The authors, however, only utilised the basic Word2Vec algorithm without exploring other word embedding methods such as Glove. After experimenting with different neural network architectures, the authors found the LSTM provided the best classification accuracy with a mean accuracy of 82.7%. LSTM neural networks are optimal for natural language processing tasks as they capture contextual information. Another approach that the authors didn't analyse is a hybrid machine learning methods such as a CNN-LSTM neural network rather than a standalone LSTM and CNN.

The work analysed the whole dataset using the classifier trained for the three labels and found that there were variations in the reporting behaviour based on the operator reporting, the drilling stage and well location. The authors noted that to improve the accuracy of the network, more balanced training data is necessary.

3.2 Unsupervised Learning

This work utilises an unsupervised learning approach and is titled "Augmented Text Mining for Daily Drilling Reports using Topic Modelling and Ontology" [ARG17]. This work focuses on utilising text mining techniques most notably topic modelling using Latent Dirichlet Allocation to extract entities relevant to drilling events. This work also tries to provide insight into drilling risks by creating a domain-based ontology to link symptoms and drilling events.

The sample size of daily drilling reports was relatively small using only 230 daily drilling reports across five offset wells. The authors briefly describe standard pre-processing steps used on the text corpus extracted from the daily drilling reports such as removal of stop words and a stemmer. Though from the generated tri-gram word cloud in the work, acronyms seem to be left compact rather than expanded. It is worth noting that the authors allowed bi-gram phrases as singular tokens.

In the exploratory data analysis stage, the authors generated a Term Document Matrix in order to generate graphical representations of the text corpus based on term frequency using Bag of Words or Histograms. Exploratory data analysis was an important step as it allowed the authors to manually reduce common domain-specific phrases that yielded little value to the extraction task.

In order to account for word order in their language model, the authors used Topic Modelling with Latent Dirichlet Allocation as the Dirichlet distribution is a good measure of word distributions. Latent Dirichlet Allocation is an iterative process that assigns words to topics and generates a mixture of the topics to a document. Given that Latent Dirichlet Allocation is an unsupervised method the algorithm generated topics which were not meaningful in the context of the work goals. Therefore, the authors manually refined their Latent Dirichlet Allocation parameters from an initial 20 topics and 100 terms extracted to nine relevant topics with five terms. A limitation

with the method proposed in this work is that extracted topics may also not directly correspond to desired drilling events, once again due to the unsupervised nature of the method.

A novel concept proposed by the authors was the use of a domain-based ontology to represent the links between extracted topics and identifying metadata for the text corpus such as Well Numbers as well associated unit data such as loss volume. The authors have not mentioned implementing a unit parser to extract unit data from the text corpus so it is assumed that the unit data is extracted from the metadata in the daily drilling report. A limitation to the use of the ontology in the paper is that it was done manually with subject matter experts rather than an automated process from the data though it has value in being a good visual demonstration of drilling risks and causes.

3.3 Feature Engineering

This work utilises feature engineering to handcraft features which are then classified using a supervised learning method. This work is titled "Automated Operations Classification using Text Mining Techniques" [Esm+10]. This work also analyses daily drilling reports with the aim of classifying normal drilling operations rather than drilling events in the text corpus. The authors similar to other works note that the Bag of Words representation is not sufficient to represent the text. However, they argue that the short sentence length in the text corpus will not yield enough co-occurrence information for other methods. Instead, the authors designed a feature vector with three sub-features parsed from the text corpus. The classification method utilises a Support Vector Machine with the extracted feature vector. The authors guided their feature design using exploratory data analysis to identify that measurements entities and operation verbs entities are the most informative in the text. Common key phrases were also found to be a distinctive indicator of an operation.

The authors utilise a three-phase process for the workflow with pre-processing steps dispersed within the whole workflow. The first phase subdivides the document text at a sentence level using punctuation. The second phase extracts the three sub-features and the last phases performs classification on the feature vector. An important aspect of this work is the use of external sources to validate semantics in the text such as a measurement dictionary to normalise units to a base unit when extracting the measurement sub-features using regular expressions. It is important to have a comprehensive unit dictionary, especially when working in a specific domain like the oil and gas industry. The authors used a WordNet dictionary to label stemmed words in the corpus to tag with part of speech which was then filtered to only extract the most frequent verbs for the keywords sub-feature. For phrase sub-feature extraction, the authors created a grammatical syntax tree with the part of speech tagged text keeping noun phrases as the features. A limitation of this work is that the authors did analyse the degree of overlap between extracted phrase sub-features across the drilling operations. This is important as it gives a weighting to the utility of the sub-features.

Activities classification with the composite feature vector utilised a Support Vector Machine where the kernel hyperparameter after some experimental results was found to be best served with the use of a linear kernel rather than an RBF kernel. The authors had an 80% classification accuracy with the linear kernel with a large dataset size of 87470 samples classifying 19 drilling operations. This work considered drilling operations rather than drilling events.

3.4 Ensemble Method

This work utilises an ensemble method for the language processing models and is titled "Natural Language Processing For Extracting Conveyance Graphs" [Bat+15]. This work aims to extract conveyance records from unstructured text documents using Natural Language Processing technique to improve a manual process with the aim of determining mineral rights to a plot of land in the oil and gas industry.

The input of this process involves documents in various formats and the authors try to streamline the extraction of the free-text comments in this work through the use of OCR with additional

metadata in the document. Given the domain of the documents having origins in different states, the authors have suggested a design which tailors the language processing models using term frequencies as indicators of different regions.

The authors suggest different language processing models where the output of the models are the parameters of interests (e.g. oil and gas lease, mineral deed, lot number, etc.). One challenge that is tackled in this work is semantic noise where the authors translate post-processing tokens of similar semantic meaning into canonical form. The authors also suggest a two-fold language processing model where the analysis of the document through the first language model identifies parameters which choose a specialised second language model to extract the final parameters. This is an interesting approach as the system will dynamically choose the best language processing model using an initial pass guided by heuristic methods thus increasing the final extraction accuracy.

The authors suggest several natural language processing models such as named entity recognition to extract relevant entities in the free text which can be further augmented with relationship labels between entities. Word windows are used to help determine the relationships between words with hard coded rules. The authors also suggest for some parameters such as dates, simple regular expressions are sufficient to extract them using decision trees. Machine learning is also suggested with manually labelled documents. The authors suggest extracting rules from labelled documents where they identify n-gram tokens within a certain threshold distance of the manually identified conveyance record parameters. The n-gram tokens would then be weighted with frequency in all labelled documents to create a rule-set for unlabelled documents.

A final interesting point in this work is the ranking of multiple candidate values for a parameter with a likelihood score through a language processing model to be manually reviewed by a human reviewer. The authors find that this partial solution is still better than a fully manual process.

Another work that utilises an ensemble method is titled "A Hybrid multiple classifier system for recognising usual and unusual drilling events" [Esm+12]. This work develops on the previous paper by the same author with an ensemble system approach for classifying drilling events. The authors suggest that a single language processing model would be outperformed by a hybrid machine learning approach while recognising the main fault would be additional computation time. The authors proposed to use three classifiers exploiting two different data sources where the final classification is the maximum product of each classifier class label output.

The authors utilise Support Vector Machines for two base classifiers including the text data classifier for the daily drilling report data source. The text feature design is the same as the previous work by the same author. The data sets analysed only considered two types of unusual drilling events which were stuck pipe and overpull. The experimental results showed that using multiple classifiers increased the classification accuracy from 80% to 87.9%.

Chapter 4

Project Implementation

4.1 Minimum Viable Product (MVP)

The MVP is a thin vertical slice of the core functionality which lays the groundwork for future improvements. The core functionality is a complete workflow from taking in the raw Daily Drilling Report data sets to producing a calibration point output file. The focus of the project is implementing the core application framework shown in Figure 1.2 which is the three high-level stages: extracting features from free text, mapping normalised features using semantic triples to argumentation cases and formulating an explanation for a data insight using abstract argumentation for case-based reasoning. These three-stages are split across the four library folders which represent the four core data mining processes shown in Figure 1.1. Each of the different modules in the folders can have their components replaced or extended in the workflow as the MVP has been designed in a modular fashion. The modules implemented in the MVP are shown in Figure 4.1 with their corresponding section numbers in the report. Table E.1 in Appendix E shows all the external libraries used in the development of the project.

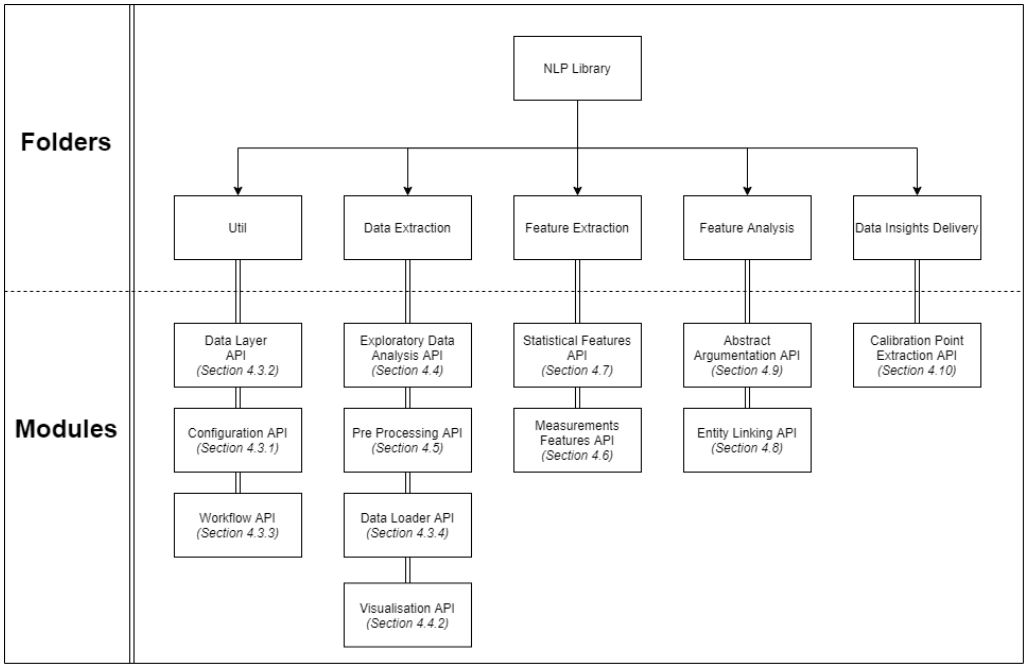


Figure 4.1: MVP modules

The data tables produced in each major pipeline of the application is shown in Figures 4.2 and 4.3.

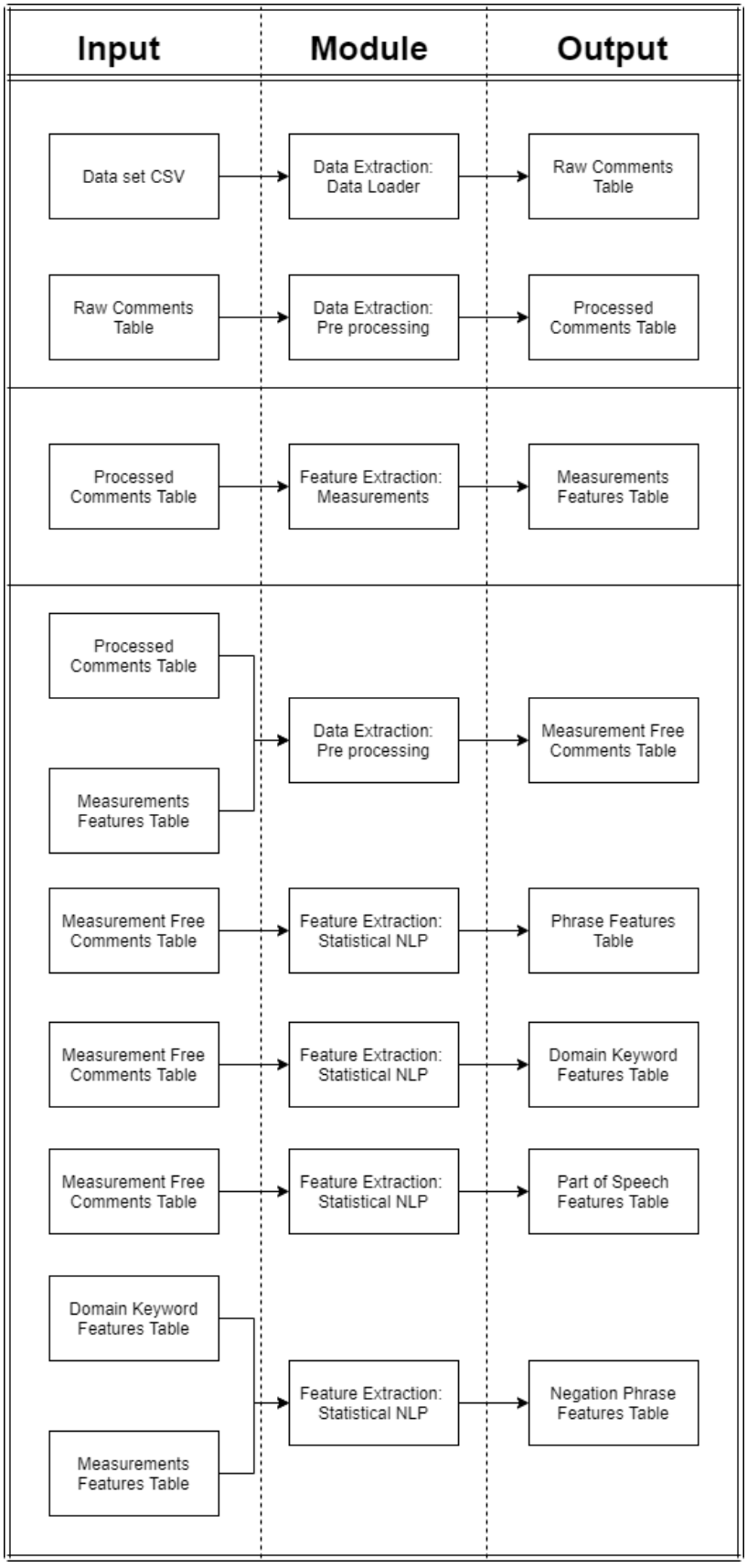


Figure 4.2: Data Tables for Data / Feature Extraction

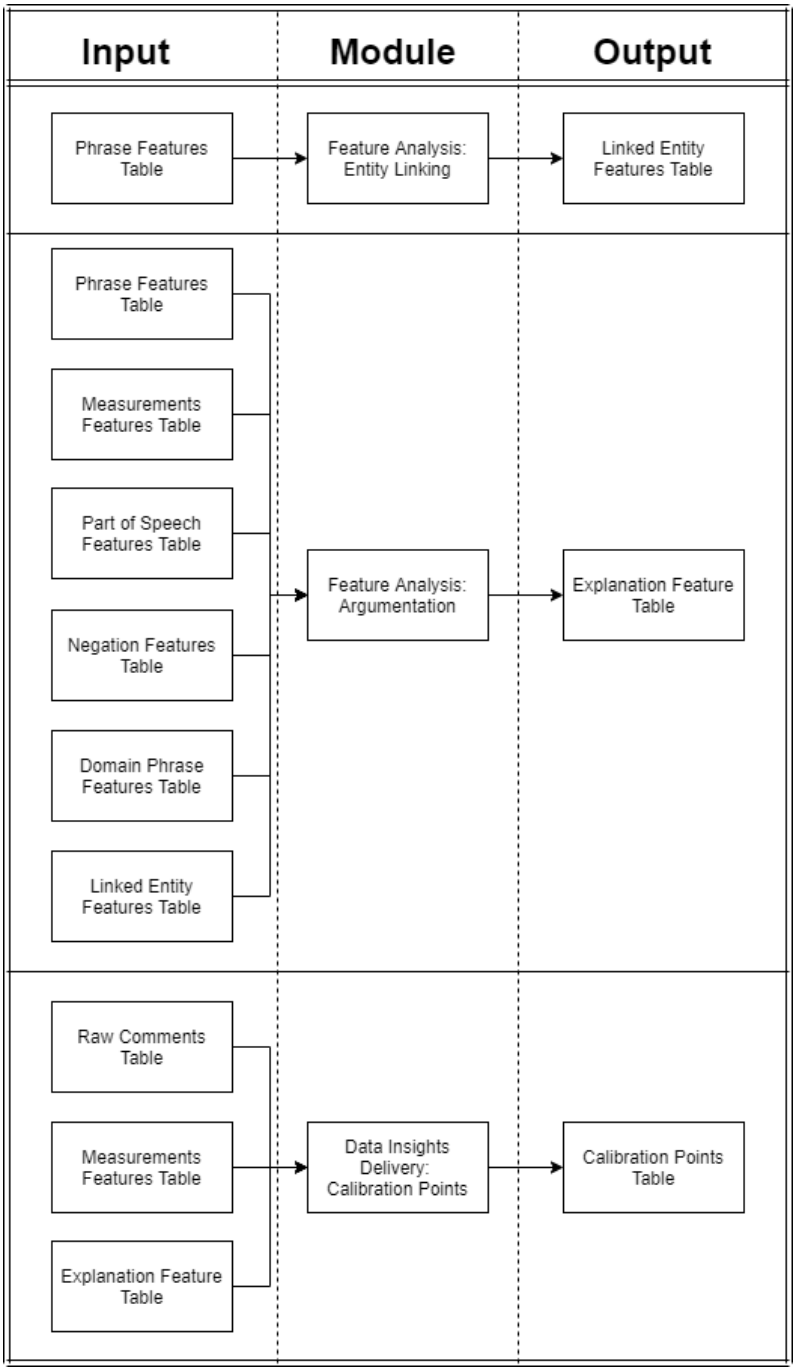


Figure 4.3: Data Tables for Feature Analysis / Data Insights Delivery

4.2 Experimentation Architecture and Workflow

The architecture of the NLP application is separated into two sections as shown in Figure 4.4. The first section is the experimentation testbed which is a Jupyter Notebook which calls all the NLP pipeline functions for an experiment workflow. The second section is local NLP library modules which contain all the NLP pipeline functionality interfaced with the Jupyter notebook via module class APIs. An interface between the Jupyter notebook experiment and module functions is necessary as it promotes a separation of concerns between the experimentation code and the reusable NLP application functionality. The development process for the NLP application was API-driven as it ensured that the NLP application is modular and will be inherently cloud-friendly for future extensions that involve deploying the application as a web app instead of a CLI application.

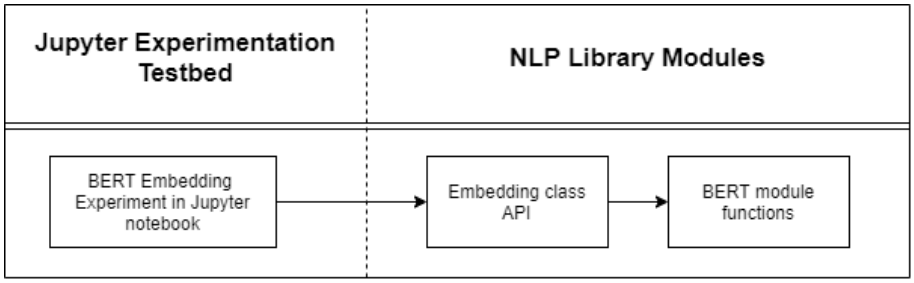


Figure 4.4: High-Level Code Architecture with example BERT experiment

This design choice for the separation between the experiments and NLP functions was also motivated by the need to have some data versioning for the machine learning models and data and also a way to keep track of the history and results of experiments for reproducibility. The workflow for the experimentation is shown in Figure 4.5. An experiment is created using a CLI command which takes in a configuration file for the experiment parameters. A folder titled with the experiment name is generated when running the setup command and stored in the Experiments folder in the project directory and has the following contents:

- Logs
  - Contains the debug logs from running the experiment in the Jupyter notebook
  - This includes error messages, experiment running times and a history of the library API calls
- Results
  - Contains the results saved at the end of the experiment
  - This includes configuration parameters, performance metrics generated by the experiment, models and any intermediate data (CSV) files.
- Configuration file
  - Contains the parameters for the whole experiment workflow
- SQLite3 Database
  - The database stores any intermediate data processed in the pipeline, not including the models
- Experiment Jupyter Notebook
  - Testbed for running the pipelines and storing the outputs

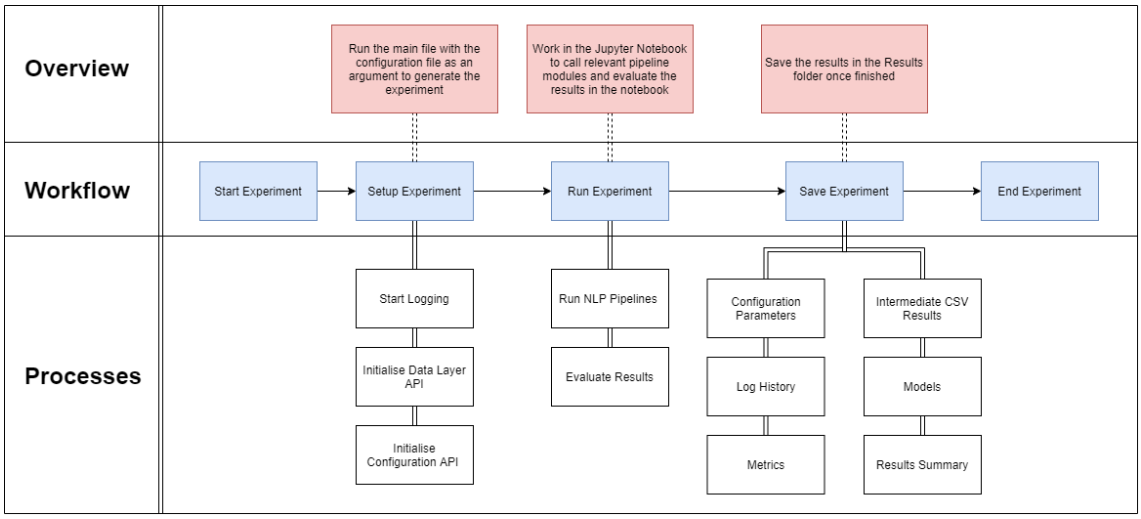


Figure 4.5: Experiment Workflow

A possible alternative was to use a platform called Data Version Control (DVC) which provided an interface similar to Git to handle data and models [Ite20]. However, the additional complexity to the project when integrating and using this platform was not worth the trade-off in the benefits gained. DVC worked best when using cloud storage while in this project everything was done through local files and the project only had one developer which reduced the need for data version control for collaboration during the project development phase. The reproducible experiments mainly mattered for knowledge transfer after the project finished as this is an industry project.

4.3 General API Design

The APIs for each module presents a consistent interface for the experiment notebook to run module functions. The interface design was an abstraction of some elements of web APIs as a possible extension would be to lift the CLI application to a web application. Therefore, the public-facing API interface consists of GET, PUT, POST, DELETE based functions for a specified resource such as the raw data set table, the summary stats of the data set and the measurements in the comments.

The HTTP based functions can also take in parameters which are validated dependent on the module function in order to prevent malformed or malicious inputs. HTTP status codes were also returned dependent on the outcome in the execution of the module function. For example, a HTTP status code of 201 was returned on a successful PUT request while a status code of 500 was returned for an internal application error when running the module functions. The class APIs can communicate with each other by calling the public class API functions. Intra API communication mainly occurs between the Data Layer API and the NLP library APIs as a way to post and fetch data tables stored in the SQLite3 Database. Furthermore, each API function is documented loosely following the OpenAPI specification as shown in Figure 4.6 [Sma20].

```
def {HTTP_METHOD}_{FUNCTION_NAME}(self, resource, parameters, ....):
    """
        Summary:
        Description:
        Resource:
        Parameters:
        Responses:
    """
    self.log.info(f"{HTTP_METHOD} Request: {resource}")

    return status_code, objects
```

Figure 4.6: API Function Code Format

4.3.1 Configuration API

A configuration file was used to handle the settings of the application and the hyperparameters in the experiments. The configuration file is a YAML file that is taken as an input for a command-line argument to the setup function in the main file. An alternate option to a YAML file is to use a JSON file but YAML files are generally better suited to configuration and are designed for editing [20g]. The high-level settings for the configuration file are defined in Table 4.1.

The Configuration API class in the Util folder is a Singleton Class. It handles the loading and validation of the configuration file to ensure that the configuration for the experiment has correct inputs. It exposes parts of the configuration settings in the configuration to other modules and the Jupyter notebook via getter functions in a consistent interface such that any refactoring to the configuration file format will have minimal impact in the rest of the code-base.

Configuration Setting	Description	Appendix Example
General	The experiment folder setup settings	D.1
Data Extraction	The settings for the input data such as domain files and pre-processing settings	D.5
Feature Extraction	The resource names for the features as well some parameters	D.3
Feature Analysis	The entity linking and argumentation parameters and resources	D.4
Data Insights	The calibration point output settings	D.2
Workflow Orchestration	The input data set-specific settings such as labels and file paths	D.6

Table 4.1: A table showing the settings for the configuration file

4.3.2 Data Layer API

The Data Layer API in the Util folder is also a Singleton Class. It handles the creation of the SQLite3 Database for an experiment through the setup function in the main file. It also acts as an interface between the SQLite3 Database and the modules / Jupyter notebook as shown in Figure 4.7.

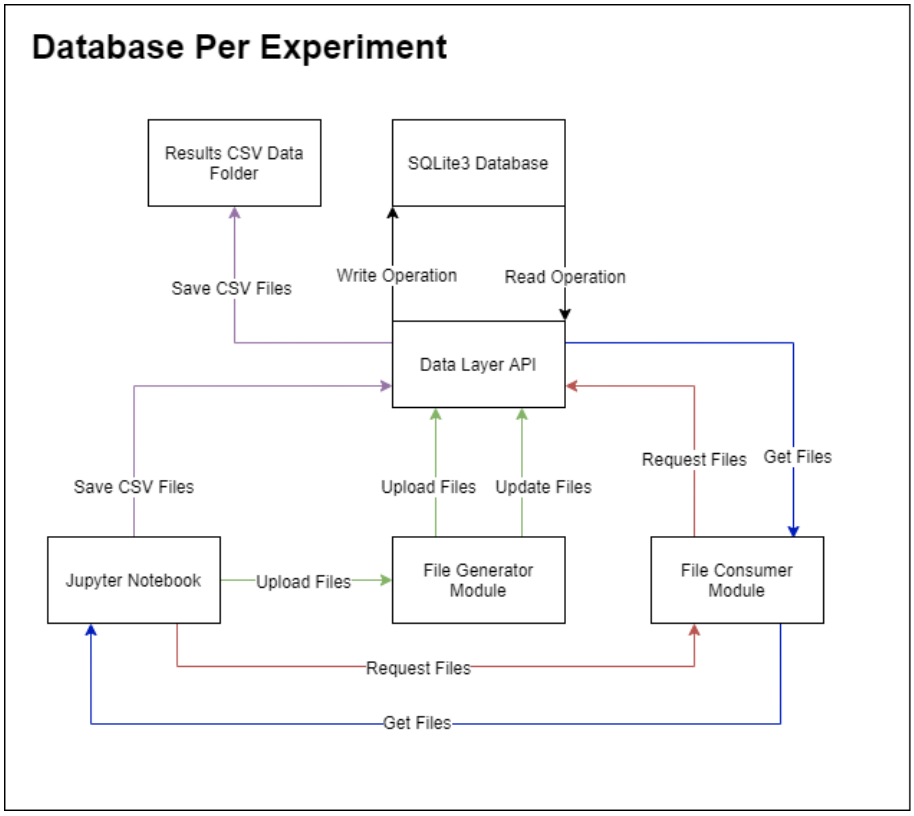


Figure 4.7: Data flows through the Data Layer API

The choice of SQLite for the database was chosen because SQLite has an easy setup and is stored locally as a database file. The usage is fine for development purposes and switching to more feature-rich Relational Database Management System would not pose too much of an issue when all the database functions are encapsulated in the Data Layer API. The design choice of having a database per experiment does yield problems of duplicated storage for common pipelines used in the experiments. A potential solution is to have a single data store with a smarter data layer API to manage SQL queries but this adds coding overhead in the API and the storage problem is not a major issue within this project.

4.3.3 Workflow API

The Workflow API provides a way to chain API calls in one single API call. During the experimentation, it is helpful to individually call and examine each of the pipelines in the Jupyter notebook in order to visualise the intermediate results. However, at the end of the project, a single API call to run the entire workflow or particular sub workflows is necessary. The Workflow API takes the

processes defined in the configuration file as shown in Appendix D.6. The chosen processes are then validated and run in a pre-defined order of operations with the outputs of each operation updating the SQLite3 Database. The parameters for each operation are fetched from the configuration file.

4.3.4 Data Loader API

The Data Loader API handles the initial loading of all the data sets and the domain files such as the measurements definition JSON into the SQLite3 Database. This API is necessary as it abstracts all complexities of loading the different input file formats into one API class away from the experimentation testbed with the Jupyter notebook. Figure 4.8 shows the workflow for the loading of the raw data sets.

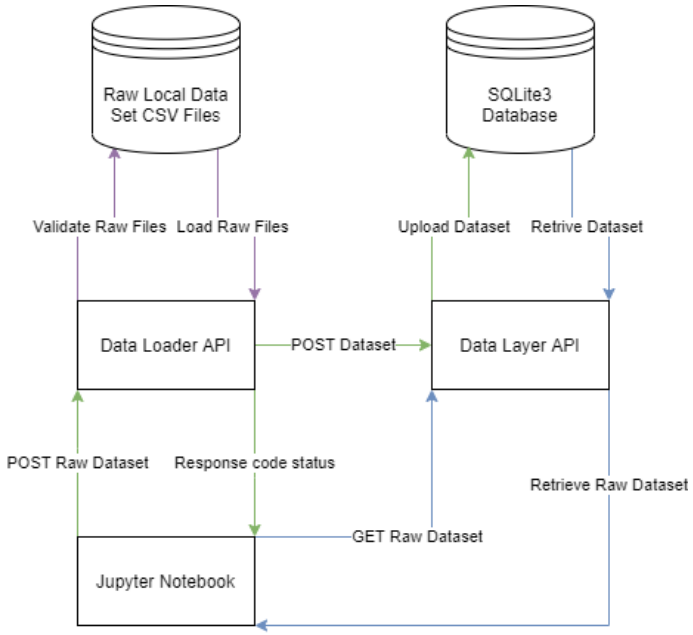


Figure 4.8: Data Set Uploading through the Data Loader API

4.3.5 Logging

Logging was implemented in this project in order to get a history of pipeline operations for the results summary and as a way to debug issues. Rather than print debugging in place within the code, logging error messages assumes defensive coding practises through catching errors in try-except blocks. Performance of the code is also captured in the logs through timing operations. The python standard library has a logging system which was used as it provided adequate granularity for logging messages as well as formatting options for the log messages.

4.4 Exploratory Data Analysis

Exploratory Data Analysis (EDA) aims to better understand the data to get an intuition of which NLP models and methods can best solve the problem. It is necessary to perform a thorough exploratory analysis of the data as it will help reveal features of interest in the data set which are not likely visible through traditional close reading [Sal18]. It can also help assess the data quality by revealing outliers and any missing values as well as highlighting data bias. The process involves an iterative cycle which generates more insightful questions about the data.

EDA was conducted on the training data set on three different levels which are sentence, comment and daily drilling report level. Furthermore, analysis of the training data set also took into account different levels of pre-processing steps such as before and after removing stop words which helped assess the quality of the pre-processing functions.

4.4.1 Data Set Summary Statistics

Data set summary statistics shown below were generated as they are low-level information regarding the training data set which helps estimate processing times as well as get basic statistics of the data set for comparison to other data sets. Figure 4.9 shows an example data set summary for the training data set.

1. Name of the file
2. Vocabulary size
3. Total number of lines
4. Total number of words
5. Total number of characters
6. The number of characters / words in the longest line
7. The number of characters / words in the shortest line
8. The size of the data set

```
{'file_name': 'training',
 'file_size': 13355672,
 'max_line_num_char': 1014,
 'max_line_num_word': 211,
 'min_line_num_char': 1,
 'min_line_num_word': 1,
 'num_characters': 8211584,
 'num_comments': 57264,
 'num_words': 1807948,
 'vocab_size': 122193}
```

Figure 4.9: File Summary Statistics for the Training Data Set

4.4.2 Uni-variate Visualisations

Uni-variate visualisations were generated for the training data set considering the following statistical features shown below:

- Character Text Length
- Letter frequency
- Word count length
- N-gram counts
- Sentiment polarity scores
- Named Entities
- Extracted feature tokes (Decimal Token, Range token etc....)
- Readability metrics (Flesch-Kincaid, Smog Index, etc....)
- Part of speech tags
- Measurements
- Noun phrases
- Verb phrases

The graphical visualisations from the exploratory data analysis results for these statistical features were generated in the Visualisation API which was responsible for plotting generic diagrams such as word clouds for variable input data. Appendix F contains some visualisations produced from the EDA process.

4.4.3 Bi-variate Visualisations

Bi-variate visualisations were examined between the aforementioned uni-variate visualisation and the occurrence of the six drilling events labels in the training data set as well as other meta-data labels such as time and depth of the comment.

The bi-variate visualisation helped with different data exploration areas (shown below) which in turn informed feature engineering and model choices.

- Find characteristic terms and their associations with time segments or locations as a feature to represent those segments.
- Detect outlier comments in the visualisations to account for special / edge cases.
- Word association analysis via word co-occurrence in document term matrices.
- Word clustering analysis plotted with cluster dendrograms

4.5 Pre-Processing

Pre-processing is a vital aspect of the workflow to clean and transform the raw textual data into an appropriate format for the modules downstream such as feature extraction. The pre-processing library implemented allowed a variety of pre-processing functions to be used on different data sets depending on an input configuration. The implementation also allows the same the data set to be processed incrementally with different levels of pre-processing should different processes require different pre-processing functions.

A multi-parse approach to the pre-processing pipeline was implemented where a compulsory custom tokenisation process would first occur on the data set resulting in tokenised comments. The optional pre-processing functions would then be composed into a chain of functions parsing successive processed tokenised comments in the order defined by the configuration. This approach applies a pre-processing function on a whole set of comments rather than the alternative approach of applying the whole set of pre-processing functions on a single comment. The alternative approach is inefficient due to the overhead in creating data frames using Pandas for each comment rather than a single data frame for the whole set of comments in the first approach. An example configuration is shown in Figure 4.10.

```
pre_process_config = {
    "lower_case": {
        "use": True,
        "parameters": None,
    },
    "tokenise_reduce": {
        "use": True,
        "parameters": []
    },
    "remove_numbers": {
        "use": False,
        "parameters": ["Number", "Decimal", "Fraction", "CommaNumber", "Time", "Range"],
    },
    "remove_symbols": {
        "use": False,
        "parameters": ["Symbol"]
    },
}
```

Figure 4.10: Pre-Processing Configuration Example

The pre-processing functions are grouped into four categories which are basic tokenisation, token normalisation, token reduction and token expansion.

4.5.1 Basic Tokenisation

Custom basic tokenisation is a compulsory process that occurs with every pre-processing pipeline which cleans that input text. The tokens are initially separated by white space and then converted into 'pure' tokens (Number, Word, Symbol) consisting of only those character elements. This step is necessary as the initial whitespace separated tokens are sometimes malformed due to data quality issues such as words being concatenated next to artefacts. Therefore a separation process needs to happen to convert the tokens into their most basic instance before they can be processed on.

4.5.2 Token Normalisation

Token normalisation functions are pre-processing functions that apply a one to one transformation of the input tokens in the comment. The normalisation functions currently implemented are shown below:

- **Lowercase** the comment to standardise the text
- **Stemming** to reduce the token 'Word' to their root form
- **Lemmatization** to reduce the token 'Word' to their root form while still being an actual defined word

4.5.3 Token Reduction

Token reduction functions are pre-processing functions that apply a many to one transformation of the input tokens in the comment. The reduction functions currently implemented are shown below:

- **Composite Token Reduction** which reduces defined token patterns to singular tokens using rolling windows across the token stream as shown in Table 4.2.
- **Remove 'Number' tokens** in order to increase the accuracy of processes such as part of speech tagging
- **Remove 'Symbol' tokens** in order to increase the accuracy of processes such as phrase chunking
- **Remove repeating words** to clean the input comment stream
- **Remove repeating pairs of words** to clean the input comment stream

Composite Token	Singular Token
Number '.' Number	Decimal
Number ':' Number	Time
Number '/' Number	Fraction
Number ',' Number	Comma Number
Number '-' Number	Range
Decimal '-' Decimal	Range
Number '-' Decimal	Range
Decimal '-' Number	Range

Table 4.2: A table showing the composite token reduction patterns

4.5.4 Token Expansion

Token expansion functions are pre-processing functions that apply a one to many transformations of the input tokens in the comment. The expansion functions currently implemented are shown below:

- **Acronym expansion** so that shorthand expansions are less ambiguous for some of the feature extraction methods such as phrase extraction

Currently, acronym expansion is limited as it relies on a lookup list in order to define an acronym in the input token stream. This method is flawed as key domain terms such as 'FIT' occur as acronyms for formation integrity test. However, there are a lot of surface mentions of the word 'fit' as a verb. Therefore, more research needs to be conducted in order to implement an algorithm that can reliably identify acronyms for expansion.

4.6 Measurement Syntactic Parser

The measurement syntactic parser extracts the measurements data from the processed comments. Measurement data consists of a unit and its corresponding values. The measurement feature table schema is shown in Table 4.3.

Column	Description
Comment ID	The internal comment identification number from which the measurement has been extracted
Unit Type	The format of the unit of the measurement which can be a Single Unit or Composite Unit
Start Character	The starting character index in the comment of the measurement
End Character	The ending character index in the comment of the measurement
Unit	The measurement unit
Minimum Value	The measurement in case the measurement was recorded as a range
Maximum Value	The measurement in case the measurement was recorded as a range

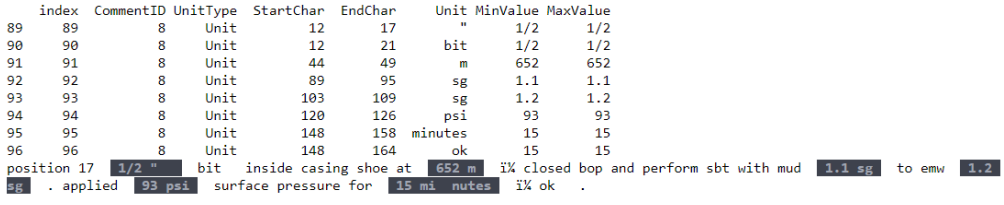
Table 4.3: A table showing the schema for the measurement feature table

The extraction of the measurement data utilises an external dictionary to define units called DELFI measurements which is ingested as a JSON file. The data loader initially loads and extracts the units in the DELFI measurement JSON file. The pre-processing module then processes the extracted units to clean them of redundant artefacts such as leading numbers and reducing the units to singular units such as metres and feet as opposed to composite units such as metres per hour.

The measurement modules utilise the units defined in the measurements dictionary to index the processed comments. If a singular unit has been found then it is validated in a context window around the unit in case of false positives such as 'at' being defined as a unit but also being a preposition. The context window is a left word window which in order to be a valid measurement should contain a numerical value.

Furthermore, a parse of the singular units is conducted to see if they are part of a composite unit using a rolling window and regular expressions of the comment tokens flagged as units, words or symbols. This stage in the extraction tackles the problem of overlapping units, as singular units overlap to form composite units, by choosing the longest composite unit in the overlap in case of a conflict of overlapping measurements. The start and end character location of the measurement data is also captured in the measurement extraction process so that pipelines further downstream can gain contextual information about the measurement.

An example of the extracted measurements for a single comment is shown in Figure 4.11 where the measurements are highlighted in the comment. There are still minor extraction inaccuracies such as '17' not being captured as a value in '17 1/2 "'. This particular example could be fixed by adding a combine continuous numbers stage as a pre-processing step.



	index	CommentID	UnitType	StartChar	EndChar	Unit	MinValue	MaxValue
89	89	8	Unit	12	17	"	1/2	1/2
90	90	8	Unit	12	21	bit	1/2	1/2
91	91	8	Unit	44	49	m	652	652
92	92	8	Unit	89	95	sg	1.1	1.1
93	93	8	Unit	103	109	sg	1.2	1.2
94	94	8	Unit	120	126	psi	93	93
95	95	8	Unit	148	158	minutes	15	15
96	96	8	Unit	148	164	ok	15	15

Figure 4.11: Measurement Extraction Example

The output of the measurement parser is a measurement feature table shown in Figure 4.12.

	CommentID	UnitType	StartChar	EndChar	Unit	MinValue	MaxValue
0	0	Unit	8	12	"	23	23
1	0	Unit	23	28	m	406	406
2	0	Unit	32	37	m	448	448
3	0	Unit	52	65	lpm	3400	3600
4	0	Unit	75	88	psi	1450	1550
5	0	Unit	101	113	mw	1.1	1.1
6	0	Unit	120	126	sg	1.1	1.1
7	0	Unit	135	152	rpm	10	20
8	0	Unit	173	181	ft	5	9
9	0	CompositeUnit	206	215	m/hrs	7	7
10	0	CompositeUnit	235	244	m/hrs	9	9
11	0	Unit	337	340	m	6	6
12	0	Unit	390	406	at	3	3
13	0	Unit	435	438	m	2	2
14	0	Unit	603	607	%	40	40
15	0	Unit	624	628	%	20	20
16	0	Unit	645	649	%	30	30
17	1	Unit	8	12	"	23	23
18	1	Unit	23	28	m	448	448
19	1	Unit	32	37	m	495	495
20	1	Unit	52	65	lpm	3500	3600

Figure 4.12: Measurement Feature Table Example

4.7 Statistical NLP Parser

The statistical NLP parser extracts handcrafted features from the comments and outputs the features as feature tables in the SQLite3 Database. The output feature tables are then used in feature analysis processes such as entity linking and abstract argumentation. The handcrafted features are those chosen in the exploratory data analysis stage to be the most informative in relation to drilling events. The input to the statistical NLP parser is the pre-processed comment data set with two additional processing steps. The first processing step removes all measurements found in the Measurement Syntactic Parser pipeline (Section 4.6) which constitutes a noticeable proportion of the data sets as shown in Table 4.4 where the average proportion of measurement data is 12%. The second processing step utilises the pre-processing pipeline to remove numbers and symbols from the comments to leave only the words in the data set. The two-step post-processing improves the accuracy of feature extraction processes such as part of speech tagging as the measurements are often inserted as statements between descriptive sentences in the data set rather than being part of natural sentence constructions. This style of writing in training data set hinders the part of speech tagging process.

The comment with the highest proportion of measurement data is '1230-1240-1253-1254-1269 - wash t / 1273 m' in the training data set. This comment is an edge case where the writer specifies a series of drilling depths for a particular action which in this case is wash. This comment is also an example of the disparate writing styles that occur in different data sets.

Max	0.78
Mean	0.12
Min	0.00
Standard Deviation	0.11
Variance	0.01

Table 4.4: A table showing the proportion of measurement character statistics in the training data set

4.7.1 Phrase Feature

Phrase features are n-gram verb and noun phrases extracted using the NLTK Chunker and a regular expression grammar as shown in Table 4.6 [BKL09]. The phrase feature consists of the extracted phrase with its identifying character index span in the comment in order to uniquely identify each phrase in the comment and compare the extracted feature with other features such as the part of speech feature. The phrase feature table schema is shown in Table 4.5.

Column	Description
Comment ID	The internal comment identification number from which the phrase has been extracted
Phrase Type	The type of the phrase which can be either a Noun Phrase or Verb Phrase
Phrase	The extracted phrase
Start Character	The starting character index in the comment of the phrase
End Character	The ending character index in the comment of the phrase

Table 4.5: A table showing the schema for the phrase feature table

The meaning of the symbols used in the grammar are defined in Table 4.7 and the part of speech tags are defined in Table 4.8 [Rex20] [20f].

NP:	{<DT>? <JJ>* <NN>*}
PP:	{<IN><NP>}
VP:	{<VB.*><NP | PP | CLAUSE>+}
	{<NP | PP | CLAUSE>+ <VB.*>}
CLAUSE:	{<NP><VP>}

Table 4.6: A table showing the regular expression grammar used for phrase chunking

The first rule parses noun phrases which are defined as a sequence of zero or one determiners followed by a sequence of zero or more adjectives and zero or more nouns. The second rule parses prepositions which are defined as a single preposition part of speech tag followed by a single noun phrase. The third rule parses verb phrases which can be any number of repetitions of verb bases followed by at least one of either a noun phrase, a preposition or a clause. The third rule alternative reverses the order of the previous one. The fourth rule parses a clause which is defined as a single noun phrase followed by a verb phrase.

Symbol	Meaning
{}	Rule body group
<>	Part of speech tag
?	Once or none
*	Zero or more times
.*	Any number of repetitions
|	OR operand
+	One or more

Table 4.7: A table showing the definitions of the regular expression symbols

Part of Speech Tag	Meaning	Example
NP	Noun Phrase	A yellow house
DT	Determiner	A
JJ	Adjective	Yellow
NN	Noun	House
IN	Preposition	In
PP	Preposition Phrase	In the house
VP	Verb Phrase	Drilling a hole
VB	Verb Base	Drilling
CLAUSE	Clause	Oil leaked

Table 4.8: A table showing the definitions of the part of speech tags

The NLTK chunk parser for the user-defined regular expression grammar takes a tokenised comment with part of speech tags as an input and generates a simple tree structure with nodes being the defined grammar and leaves being the comment tokens. A traversal of the subtree chunks of the extracted syntax tree yields the noun and verb phrases. However, the default NLTK regular expression parser does not have the capability to capture the character index span of the comment tokens. Therefore a two parse approach of the tree was implemented with a pre-computed token span dictionary where comment token indexes are keys and character index spans are values for each comment. In the first parse of the tree, the comment token index was embedded in the comment token string of the tree leaves assuming in order traversal of the tree leaves. The second parse traversed the subtrees capturing the verb and noun phrases where the character spans of the phrases were extracted using the embedded character token index in the leaves and the lookup token span dictionary for the comment.

	PhraseType	Phrase	StartChar	EndChar
198490	NounPhrase	rih scraper bha	0	15
198491	NounPhrase	theoretical displacement	27	51
198492	NounPhrase	actual displacement	52	71
198493	NounPhrase	every stands	87	99

rih scraper bha on from to theoretical displacement actual displacement fill up string every stands

Figure 4.13: Noun Phrase Extraction Example

	PhraseType	Phrase	StartChar	EndChar
196364	VerbPhrase	installed kill line coflex line bell nipple	8	51
196365	VerbPhrase	tightened all	70	83
196366	VerbPhrase	performed pressure test for quick connector latch	101	150
196367	VerbPhrase	screw connection with min	151	176
196368	VerbPhrase	witnessed by wss	180	196
196369	VerbPhrase	performed maintenance of tds	229	257

n u bop installed kill line coflex line bell nipple and flow line and tightened all bolts mean while performed pressure test for quick connector latch screw connection with min ok witnessed by wss and z fod wss function bop test performed maintenance of tds and draw works

Figure 4.14: Verb Phrase Extraction Example

An example of the extracted phrases for a single comment is shown in Figure 4.13 where the noun phrases are highlighted in the comments and Figure 4.14 where the verb phrases are highlighted in the comments. Figure 4.13 shows the limitations in the phrase extraction as it treats acronyms as nouns without expansion. For example, 'rih' stands for 'Rotate in Hole' which should be tagged as a verb phrase. This can be fixed with an acronym expansion process in the pre-processing using a pre-defined domain acronym lookup list. A comment is constructed of one or more sentences, but the pre-processing stage of removing symbols removes full stop identifiers for sentence boundaries. Part of speech tagging may yield better tags for the regular expression grammar parser if full stops were exempt from symbol removal as the sentences would be better formed. However, for the initial implementation simplicity in other pipelines such as abstract argumentation, multiple sentence comments were treated as a single sentence. The output of the phrase parser is a phrase feature table shown in Figure 4.15.

	CommentID	PhraseType	Phrase	StartChar	EndChar
145	50	NounPhrase	no downhole loss	164	180
146	50	NounPhrase	npt p	0	5
147	50	VerbPhrase	oil leaked from the chamber lay	113	144
148	51	NounPhrase	actual displacement m	50	71
149	51	NounPhrase	npt cont rih bha	0	16
150	51	NounPhrase	theoretical displacement	25	49
151	52	NounPhrase	last	58	62
152	52	NounPhrase	toc	39	42
153	52	VerbPhrase	tag toc	35	42
154	53	NounPhrase	wob	45	48
155	53	VerbPhrase	trq wob	41	48
156	53	VerbPhrase	flow rate	27	36
157	54	NounPhrase	every connection	114	130
158	54	NounPhrase	middle	151	157
159	54	VerbPhrase	drilled cement formation	4	28
160	55	VerbPhrase	confirm hole inclination	49	73
161	55	VerbPhrase	totco survey	33	45
162	55	NounPhrase	circulation	130	141
163	56	NounPhrase	every connection	85	101
164	56	VerbPhrase	flow rate	37	46
165	56	NounPhrase	dolomite	162	170

Figure 4.15: Phrase Feature Table Example

4.7.2 Domain Phrases

The domain phrases are phrases that commonly occur with drilling events as specified by a domain expert. The domain phrase feature table schema is shown in Table 4.9.

Column	Description
Comment ID	The internal comment identification number from which the domain phrase has been extracted
Domain Event	The drilling event that the keyword is associated with
Label	The level of importance of the keyword to the drilling event which can be either Primary or Secondary
Feature	The domain phrase
Start Character	The starting character index in the comment of the phrase
End Character	The ending character index in the comment of the phrase

Table 4.9: A table showing the schema for the domain feature table

The domain phrases are defined in a CSV file and loaded into the SQLite3 database using the Data Loader API. In order for the keyword search algorithm to work, the input domain phrases require the same pre-processing steps such as lowercasing as the comment data set. Therefore, the raw input domain phrases are also pre-processed with the PreProcessing API. The processed domain phrase schema is shown in Table 4.10 and an example domain phrase is shown in Figure 4.16. The extraction of the domain phrases utilises a regular expression search on the word only data set which can extract multiple occurrences of a domain phrase within a comment while also adding character spans to uniquely identify the domain phrase.

Column	Description
Domain Event	The drilling event that the keyword is associated with
Label	The level of importance of the keyword to the drilling event which can be either Primary or Secondary
Feature	The domain phrase

Table 4.10: A table showing the schema for the processed domain phrase table

	CommentID	StartChar	EndChar	DomainEvent	Label	Feature
867	1192	19	23	StuckPipe	Primary	work
868	1192	50	56	StuckPipe	Primary	unable
869	1192	67	73	StuckPipe	Primary	torque
870	1192	4	9	StuckPipe	Primary	stuck
871	1192	38	49	StuckPipe	Primary	circulation
872	1192	26	32	StuckPipe	Primary	string
873	1192	60	66	StuckPipe	Primary	rotate
874	1192	50	66	StuckPipe	Primary	unable to rotate
875	1192	94	98	StuckPipe	Primary	free

Figure 4.16: Domain Phrase Feature Table Example

The domain phrase table is a useful feature as the nature of drilling event comments is factual and scientific. Therefore if a drilling event occurs there is a high likelihood that the corresponding domain phrase terms will co-occur within the same comment. However, simply using these domain phrase terms without any contextual information would yield false positives as these terms are not unique to the drilling events. For example for stuck pipe, a common domain phrase for stuck pipe is 'string' which may be a shorthand for drill string in the comment. However, the report writer may be referring to other operations with a drill string rather than a stuck pipe event. 'Unable to rotate' is also another false positive as it may refer to drilling equipment such as a Whipstock which is not related to stuck pipe events. While domain phrases will increase the true positive rate it will have also increase the false positives at a much greater rate. Figure 4.17 shows an example extraction and the false positive problem where neither a leak off test or a stuck pipe occur in the comment but the associated domain phrase does.

	DomainEvent	Feature	StartChar	EndChar
50585	StuckPipe	string	6	12
50586	StuckPipe	string	57	63
50584	StuckPipe	bled	91	95
50587	LeakOffTest	pressure	100	108

```
check string weight s and set well commissioner with sow string weight up slack off weight bled off pressure from to by psi ste
ps
```

Figure 4.17: Domain Phrase Extraction Example

4.7.3 Part of Speech Feature

Part of Speech features are the part of speech tags for each comment token with the identifying character index span for each comment. The NLTK part of speech tagger is used to tag comment tokens generated by splitting comments on white space from the word only comment data set. The part of speech feature table schema is shown in Table 4.11 [BKL09].

Column	Description
Comment ID	The internal comment identification number from which the part of speech tag has been extracted
Part of Speech Tag	The part of speech tag from the Universal POS tagset
Token Content	The tagged token
Start Character	The starting character index in the comment of the phrase
End Character	The ending character index in the comment of the phrase

Table 4.11: A table showing the schema for the part of speech feature table

The part of speech tags are useful features as they can help reduce false positives for the extraction of drilling events such as leak off tests (LOT) and formation integrity tests (FIT) as shown in Figure 4.18.

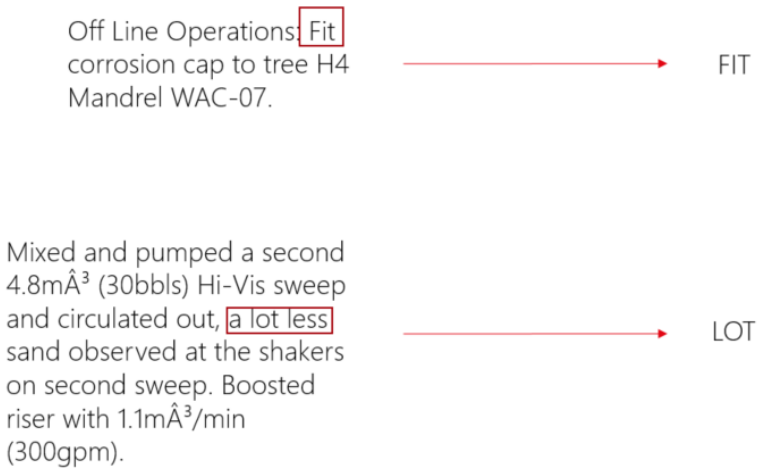


Figure 4.18: False Positive Examples

When a leak off test or formation integrity test occurs it is generally stated formally in the drilling report as it is a planned drilling event rather than an unplanned one such as a stuck pipe. Report authors often state a leak off test has occurred in the report using its abbreviated form, 'LOT' and 'FIT'. With other features such as domain phrase features, a false positive could occur with the domain phrase feature 'lot' where the context could be a leak off test or 'a lot of'. Therefore, having the part of speech feature could reduce this ambiguity as a leak off test 'lot' would be tagged as a noun and the 'lot' in 'a lot of' would be tagged as an adjective. Figure 4.19 shows two comments with 'fit' identified while being tagged with a different part of speech tags to differentiate between a false positive formation integrity test event.

	CommentID	PartOfSpeech	TokenContent	StartChar	EndChar
140978	7091	Verb	fit	95	98

npt attempt to n u new bell nipple observed the bell nipple out let union connection would not fit the flow line l d same to we ld another out let average d hole losses fluid lost this period m

	CommentID	PartOfSpeech	TokenContent	StartChar	EndChar
170692	8372	Noun	fit	62	65

position inside casing shoe at closed annular bop and perform fit with mud to emw pressure dropped from to time emw sg test per formed with mud pump

Figure 4.19: Formation Integrity Test (FIT) False Positive Example

The NLTK part of speech tagger by default uses the Penn Treebank part of speech tags which is a list of 36 tags. However, the simpler Universal tag set was chosen instead which is a list of 12 coarser tags for the NLTK part of speech tagger as the pre-processing steps such as lemmatisation would invalidate the different variations of the base tag in the universal tag set that the Penn Treebank tag set offers [20f]. Furthermore, a more general tag set would enable better feature comparisons would other feature extracted from different natural language libraries such as Spacy. Figure 4.20 shows the part of speech tags extracted for an example comment with the Universal tag set.

	CommentID	PartOfSpeech	TokenContent	StartChar	EndChar
954412	43726	Adverb	npt	0	3
954413	43726	Verb	circulated	4	14
954414	43726	Particle	to	15	17
954415	43726	Verb	evaluate	18	26
954416	43726	Adjective	dynamic	27	34
954417	43726	Noun	loss	35	39
954418	43726	Noun	rate	40	44
954419	43726	Noun	flow	45	49
954420	43726	Noun	rate	50	54
954421	43726	Verb	spp	55	58
954422	43726	Adverb	down	59	63
954423	43726	Adjective	hole	64	68
954424	43726	Noun	losses	69	75
954425	43726	Noun	loss	76	80
954426	43726	Noun	rate	81	85
954427	43726	Noun	hr	86	88

Figure 4.20: Part of Speech Feature Table Example

4.7.4 Negation Phrase Feature

The negation phrase features are the domain phrases that have been flagged in the comment augmented with a negation flag which dictates whether the domain phrase is a negative mention. The negation phrase feature table schema is shown in Table 4.12.

Column	Description
Comment ID	The internal comment identification number from which the negation phrase has been extracted
Domain Event	The drilling event that the keyword is associated with
Negation	The negation keyword associated with the domain phrase
Feature	The domain phrase
Start Character	The starting character index in the comment of the phrase
End Character	The ending character index in the comment of the phrase

Table 4.12: A table showing the schema for the domain feature table

The negation phrases are captured by indexing on the character spans of the extracted domain phrases and searching for a pre-defined negative word set in a left context window around the captured domain phrase. The size of the left context window is a hyperparameter that was tuned to avoid false positives in the negation phrases. Figure 4.21 shows the negation table for some example comments.

	CommentID	StartChar	EndChar	DomainEvent	Negation	Feature
0	0	150	156	Losses	no	losses
1	1	97	103	Losses	no	losses
2	2	97	103	Losses	no	losses
3	23	29	42	Losses	no	static losses
4	23	36	42	Losses	no	losses
5	24	62	73	TightHole	no	obstruction
6	92	68	72	Losses	no	pump
7	93	84	88	Losses	no	pump
8	109	410	415	TightHole	avoid	tight

Figure 4.21: Negation Phrase Feature Table Example

The negation phrases are useful to extract as they represent a simple contextual check that reduces a lot of false positives, especially for the losses drilling event. The frequent phrase 'downhole losses' is captured by the domain phrase feature extraction but leads to false positives when the context of the mention is 'no downhole losses'. Therefore, the negative keyword 'no' needs to captured in the negation feature table to avoid the false positive.

Currently, the implementation of the negation phrase feature looks only at pre-defined negative words which is not scale-able across different data sets. A better implementation would be to capture the sentence in which the domain phrase occurs and conduct sentiment analysis on the sentence to get a sentiment score for the context. If the score is negative then the domain phrase would be flagged as a negative mention.

4.8 Named Entity Linking

Named Entity Linking is a process that assigns a unique identity to entities mentioned in the text and links those entities in the comments with their corresponding articles in a knowledge base. The overall system design of the named entity linking is split into three stages which are candidate entity generation, candidate entity ranking and unlinkable mention prediction. Before these three stages, the entities which need to be mapped i.e. the surface mentions, need to be extracted from the text which is done with a predefined knowledge base and name dictionary. The output of the entity linking pipeline is a linked entity feature table which is a more reliable feature table than the domain phrase feature table which relies on naive string search.

4.8.1 Knowledge Base

The knowledge base is defined as a key-value pairing between an entity and its definitions stored in a CSV file where the columns are defined in Figure 4.13.

Column	Description
Entity	The object of interest that can found in the data set
Filename	The filename of the JSON file containing the definition of the entity

Table 4.13: A table showing the schema for the knowledge base entities

The contents of the definitions in this project is a glossary article of the definition of the entity with a list of possible aliases of the entity. The definition file is stored as a JSON file as it can easily be converted to a python dictionary within the NLP application and is easily extensible to add new properties for the definitions such as hierarchical relationships between entities. An example definition file for a leak off test entity is shown in Figure 4.22.

```
{
    "entity": "leak off test",
    "alias": ["lot", "leak-off", "leakoff test", "leak-off test", "leak-off pressure"],
    "article": "LOT, a drilling test. A LOT is intended to determine the point of leak-off. Compare to FIT."
}
```


Figure 4.22: Leak Off Test Definition Example

The source of the glossary articles was taken from the open-source Schlumberger Glossary [Sch20]. PetroWiki is also another good source as it is a specialised wiki for the oil and gas domain but there may be licensing issues when using this encyclopedia so it was not used in the current implementation [Soc20]. These knowledge sources have been chosen as they are reliable sources closely associated with the oil and gas domain bearing in mind that the data set comments are heavily domain-specific. The current knowledge base consists of hand-picked entities closely associated with the drilling events. The definitions of these entities have been manually extracted from the knowledge sources rather than building an automatic web crawler to extract all the domain entities as the entity linking aspect of this project was implemented as a proof of concept.

4.8.2 Name Dictionary

The name dictionary is an offline dictionary used in the candidate entity generation stage as a way of mapping the possible surface mentions aliases in the comments to the entities in the knowledge base. The name dictionary schema is shown in Table 4.14. The mapping is a one to one / many mapping where the key column is the list of possible name variations and the value is a set of named entities that could match to it.

Column	Description
Variation	The alias of the candidate entity
Candidate Entities	The set of possible entities that could match to the surface mention variation

Table 4.14: A table showing the schema for the name dictionary

The aliases of the entities are shown in Figure 4.22 for a leak off test. These include name variations (leakoff test), abbreviations (lot), confusable names (leak off pressure), spelling variations (leak-off test) and shorthand variations (leak-off). An example name dictionary is shown in Figure 4.23.

	Variation	CandidateEntities
0	lot	leak off test
1	leak-off	leak off test
2	leakoff test	leak off test
3	leak-off test	leak off test
4	leak-off pressure	leak off test
5	leak off test	leak off test
6	stuck	stuck pipe
7	stuck tool	stuck pipe
8	mechanical stuck	stuck pipe
9	differential stuck	stuck pipe
10	differential stick	stuck pipe
11	stuck pipe	stuck pipe
12	fishing tool	jar
13	slickline	jar
14	jar	jar

Figure 4.23: Name Dictionary Example

4.8.3 Surface Mentions

The surface mentions are entities within the text that are chosen to be linked to their corresponding entities in the knowledge base. The surface mentions are extracted from the phrase feature table generated in the feature extraction pipeline. The phrases that are mapped to linked entities are only

the noun phrases rather than the verb phrases as they represent defined entities in the knowledge base. The surface mention schema is shown in Table 4.15.

Column	Description
Comment ID	The internal comment identification number from which the mention has been extracted
Mention	The noun phrases that represent the surface mentions of the entities in the knowledge base
Start Character	The starting character index in the comment of the phrase
End Character	The ending character index in the comment of the phrase

Table 4.15: A table showing the schema for the surface mention table

An example surface mention table is shown in Figure 4.24. The surface mentions extracted relies on the accuracy of entity extraction in the noun phrases chunker. Shorthand phrases and acronyms such as 'ge' in 'the rig floor ge' will add noise to the entity linking process but the various algorithm used in candidate entity generation help disambiguate and the noise, as well as the surface mentions style of writing.

	CommentID	Mention	StartChar	EndChar
12030	1234	pjsm	0	4
12031	1234	start esp installation p	12	36
12032	1234	u esp equipment	37	52
12033	1234	the rig floor ge	56	72
12034	1234	crew	73	77
12035	1234	assemble	78	86
12036	1234	hole unit pump motor	92	112
12037	1234	seal	113	117
12038	1234	pass assembly	121	134
12039	1234	multi sensor	146	158

Figure 4.24: Surface Mention Example

Normally surface mentions are generated from named entities extracted from named entity recognition run on the input text. However, the accuracy of the named entity recognition parser using the off the shelf model in Spacy was poor after experimenting with different pre-processing steps. A more fine-tuned named entity recognition model for the domain-specific text is necessary before the surface mentions can be generated from the named entity recognition process.

4.8.4 Candidate Entity Generation

Candidate entity generation takes the surface mentions extracted from the comments and matches them with the name variations in the name dictionary to generate candidate entities using string-based methods. The process filters out irrelevant entities in the knowledge base and the surface mentions which cannot be matched against the name dictionary. The schema for the candidate entity generation process output is shown in Table 4.16.

Column	Description
Comment ID	The internal comment identification number from which the mention has been extracted
Mention	The noun phrases that represent the surface mentions of the entities in the knowledge base
Start Character	The starting character index in the comment of the mention
End Character	The ending character index in the comment of the mention
Candidate Entities	The set of possible entities that could match to the surface mention variation

Table 4.16: A table showing the schema for the candidate entity generation table

An example candidate entity generation table is shown in Figure 4.25. In comment '19442' there are three surface mentions found which map to the name dictionary variations columns. The surface mentions are then mapped to the entities in the knowledge base which are 'stuck pipe' and 'jar' in a singular set in the candidate entities column.

	CommentID	Mention	StartChar	EndChar	CandidateEntities
299	19442	stuck pipe	25	35	stuck pipe
300	19442	stuck	53	58	stuck pipe
301	19442	jar	208	211	jar

Figure 4.25: Candidate Entity Generation Example

4.8.5 Candidate Entity Ranking Scores

Candidate entity ranking scores is a process which assigns a score to each candidate entity based on various matching algorithms. The ranking methods evaluate the mapping between the surface mention and the candidate entity using different kinds of evidence. The schema for the candidate entity ranking scores process output is shown in Table 4.17.

Column	Description
Comment ID	The internal comment identification number from which the mention has been extracted
Mention	The noun phrases that represent the surface mentions of the entities in the knowledge base
Start Character	The starting character index in the comment of the mention
End Character	The ending character index in the comment of the mention
Candidate Entity	The candidate entity that could match to the surface mention variation
Exact Match Rank	The ranking score for the exact string matching algorithm
Prefix Match Rank	The ranking score for the prefix string matching algorithm

Table 4.17: A table showing the schema for the candidate entity ranking scores table

The candidate entity ranking scores process takes a configuration which defines the ranking methods to use shown in Figure 4.26. The configuration is divided into levels with the first level being the type of evidence being considered which currently is only context-independent. The second level is the matching algorithms implemented based on the type of evidence. The matching algorithms flatten the set of candidate entities in the candidate entities columns into one to one rows of mentions to candidate entities.

```
ranking_scores_config = {
    "context_independent": {
        "use": True,
        "parameters": ["exact_string_match", "prefix_match"],
    },
}
```

Figure 4.26: Candidate Entity Ranking Scores Configuration

The output candidate entity ranking scores table is a variable column length table which generates columns for each matching algorithm run based on the configuration as shown in Figure 4.27.

	CommentID	Mention	StartChar	EndChar	CandidateEntity	ExactMatchRank	PrefixMatchRank
299	19442	stuck pipe	25	35	stuck pipe	1	1
300	19442	stuck	53	58	stuck pipe	0	1
301	19442	jar	208	211	jar	1	1

Figure 4.27: Candidate Entity Ranking Scores Example

Since the named entity linking process was implemented as a proof of concept, the matching algorithms were simple string-based matching algorithms (i.e. context-independent features) as shown in Table 4.18.

Ranking Method	Description
Exact Match Rank	Binary score (0, 1) for the exact string match of the surface mention and candidate entity
Prefix Match Rank	Binary score (0, 1) for the prefix string match of the surface mention and candidate entity

Table 4.18: A table showing the candidate ranking method definitions

Context independent features only take into account the direct relationship between the mention and the candidate entity without looking at the context of the sentence. The one-to-one matching algorithms used also does not take into account coherent entities. Coherent entities are entities with hierarchical relationships such as the entity 'jar' which has a relationship with the event 'stuck pipe'. Due to the limitations of the knowledge base which does not have hierarchical relationships defined, coherent entities were not implemented.

4.8.6 Candidate Entity Composite Ranking

Candidate entity composite ranking aggregates the ranking scores for each extraction algorithm with a weighting based on the importance of the ranking algorithm. The candidate entity with the maximum composite score for each surface mention is chosen to generate the output table with the schema shown in Table 4.19.

Column	Description
Comment ID	The internal comment identification number from which the mention has been extracted
Mention	The noun phrases that represent the surface mentions of the entities in the knowledge base
Start Character	The starting character index in the comment of the mention
End Character	The ending character index in the comment of the mention
Candidate Entity	The candidate entity that has been matched to the surface mention variation
Composite Score	The composite score for the matching algorithms
Duplicate Max	A flag to check if there duplicate maximum composite score for the surface mention

Table 4.19: A table showing the schema for the candidate entity composite ranking scores table

The weightings for each of the string matching algorithms are defined in a configuration dictionary passed into the candidate entity ranking process as shown in Figure 4.28. The weightings are user-defined parameters in the range zero to one where one is the maximum weighting.

```
ranking_config = {
    "norm_range": [0, 1],
    "weights": {
        "exact_string_match": 1,
        "prefix_match": 1,
    }
}
```

Figure 4.28: Candidate Entity Composite Ranking Scores Configuration

The output of the candidate entity ranking process has to be a singular confidence score which is comparable across candidate entities for a singular surface mention. Therefore the scores are normalised within a specified range defined in the configuration before and after they are aggregated. The normalisation formula used is shown below:

$$z_i = \frac{x_i - min(x)}{max(x) - min(x)}$$

$$where\ x = (x_1, ..., x_n)\ is\ the\ column\ ranking\ scores\ and\ z_i\ is\ the\ normalised\ data$$

In the case where there is a duplicate maximum composite score value for the surface mention, the surface mention is chosen at random but a duplicate maximum flag is raised for filtering in processes downstream as shown in the example output in Figure 4.29.

	CommentID	Mention	StartChar	EndChar	CandidateEntity	CompositeScore	DuplicateMax
299	19442	jar	208	211	jar	1.0	False
300	19442	stuck	53	58	stuck pipe	0.5	False
301	19442	stuck pipe	25	35	stuck pipe	1.0	False

Figure 4.29: Candidate Entity Composite Ranking Scores Example

4.8.7 Unlinkable Mention Prediction

Unlinkable mention prediction validates the candidate entity composite ranking score using a threshold value. It is a simple heuristic based filtering process that removes all candidate entities generated in the candidate entity composite ranking process below the user-defined threshold value to generate an output table with the schema shown in Figure 4.21.

Column	Description
Comment ID	The internal comment identification number from which the mention has been extracted
Mention	The noun phrases that represent the surface mentions of the entities in the knowledge base
Start Character	The starting character index in the comment of the mention
End Character	The ending character index in the comment of the mention
Candidate Entity	The candidate entity that has been matched to the surface mention variation
Composite Score	The composite score for the matching algorithms
Duplicate Max	A flag to check if there duplicate maximum composite score for the surface mention

Table 4.20: A table showing the schema for the unlinkable mention prediction table

The output unlinkable mention prediction table is a left join of the initial surface mention table input and the results of the candidate entity composite score process after filtering with the threshold value. The NaN values are then filled with a NIL placeholder for the surface mentions that do not have a verified mapping at the end of the entity linking process as shown in Figure 4.30.

	CommentID	Mention	StartChar	EndChar	CandidateEntity	CompositeScore	DuplicateMax
128065	19442	stuck pipe	25	35	stuck pipe	1.0	False
128066	19442	stuck	53	58	stuck pipe	0.5	False
128067	19442	p slack	64	71	NIL	0.0	False
128068	19442	wt wih	76	82	NIL	0.0	False
128069	19442	no rotation	83	94	NIL	0.0	False
128070	19442	rotation	116	124	NIL	0.0	False
128071	19442	max	130	133	NIL	0.0	False
128072	19442	rh torque	141	150	NIL	0.0	False
128073	19442	different	176	185	NIL	0.0	False
128074	19442	jar	208	211	jar	1.0	False

Figure 4.30: Unlinkable Mention Prediction Example

This is an important stage in the entity linking pipeline as the optimally chosen entity in the candidate entity set may not be correct if the set of candidate entities have a low score generally. This stage provides a filter for accepting only candidate entities with a reasonable confidence score.

4.8.8 Linked Entities Feature

The linked entity features are the non NIL candidate entities found in the unlinkable mention prediction output. The linked entity feature table schema is shown in Table 4.21.

Column	Description
Comment ID	The internal comment identification number from which the mention has been extracted
Mention	The noun phrases that represent the surface mentions of the entities in the knowledge base
Start Character	The starting character index in the comment of the mention
End Character	The ending character index in the comment of the mention
Candidate Entity	The linked entity that has been matched to the surface mention

Table 4.21: A table showing the schema for the linked entities table

The output linked entity table has been generated by filtering out the NIL values in the candidate entity column as shown in Figure 4.31.

	CommentID	Mention	StartChar	EndChar	CandidateEntity
268	19446	stuck pipe	14	24	stuck pipe
269	19446	jar	149	152	jar
270	19446	stuck pipe	294	304	stuck pipe

Figure 4.31: Linked Entity Feature Table Example

The goal of entity linking in this project is to mainly reduce false positives with a sub-goal of enriching the comments with definitions. The algorithms involved with the three main stages can use contextual knowledge to extract the linked entities which reduce ambiguity by mapping the surface mentions to their correct definition. It helps differentiate between ambiguous surface mentions such as for the domain phrase 'jar'. Jar can be a container or a tool and for 'stuck pipe' events, the jar mention is only needed for the instance where it is a tool.

Furthermore, the process of data enrichment can help a domain expert with understanding the entities within a comment. A domain expert will still have limitations in their own knowledge especially when working with new data sets which may have unfamiliar terms. This scenario occurred when discussing the comment shown in Figure 4.32 where the term 'castellation' was unfamiliar to the domain expert. With the entity linking system, the term would be linked to an article definition which would help resolve any confusion about the meaning of the terms for the end-users.

	CommentID	Comment
4206	4206	RIH and land off on castellationâ€™s, setting down 2.3 to 4.6tonne (5 to 10klbs). Unable to rotate string. Picked up weight to aid rotation. Unable to rotate string unless all weight off which in turn prevents string from dropping in to slots. Picked up clear of SSTT latch body and reset string. RIH and land off on castellationâ€™s setting down 2.3tonne (5klbs). Unable to rotate string. Inched up weight to aid rotation.

Figure 4.32: Data Enrichment Example

4.9 Abstract Argumentation for Case-Based Reasoning

Abstract argumentation provides a framework to evaluate arguments in relation to a default outcome label such as a drilling event. The abstract argumentation framework is adapted for case-based reasoning, where a pre-defined set of cases generated from a domain expert and automated methods on a labelled data set is used to generate an argumentation graph. The argumentation graph is then augmented with a new query case from which a set semantic is calculated which assigns the query case with either a support or attack polarity with regards to the default outcome of the argumentation graph. The augmented argumentation graph can then be converted into a subgraph called a dispute tree which contains a set of arguments that provide an explanation for the new case either supporting or attacking the default outcome.

4.9.1 Cases

The basis of case-based reasoning is in the definition of the cases. A generic case in the context of this project is a set of translated features extracted in processes such as phrase extraction and entity linking grouped together with an outcome with respect to an argumentation framework based around a default case. There are also two special types of cases which are the default case and the new case.

Feature to Case Feature Mapping

The raw features extracted in the feature extraction pipeline and entity linking pipeline are too varied in order to fit into a tractable argumentation framework. It is also difficult to define relationships between the features and compare them in their extracted form. Therefore, a mapping between the raw features into a consistent format was implemented in order to remove these limitations.

The chosen format for the case features is a semantic triple based on the Resource Description

Framework (RDF) data model [RDF14]. The RDF model triple consists of a subject, predicate and object which in addition to identifying metadata and a polarity defined with respect to the default outcome forms the case format schema shown in Table 4.22.

Column	Description
Argument	The name of the overall argument that the framework is debating
Case	The identification number of the case
UID	The identification hash of the case feature
Subject	The subject is a feature that has been extracted such as a phrase or character span range.
Predicate	The predicate is the relationship between features such as 'found in' or 'is a'.
Object	The object is another feature such as 'Verb' or a feature table such 'measurement'.
Level	The context level of the case feature which can be sentence or report
Polarity	The polarity of the case feature which is defined with respect to the default outcome

Table 4.22: A table showing the schema for the unlinkable mention prediction table

The case features defined in a single case have the same case identification number and polarity as a single case either supports or attacks the default outcome. The case features also have the same UID hash if they have the same subject, predicate, object and level combination. The level of the case feature was implemented in order to compare case features at different context levels. For example, a stuck pipe may have been found in a particular comment but persists in subsequent comments in the daily drilling report without a direct mention. A stuck pipe drilling event feature case can be created at the report level to account for this case. An example set of cases is shown for the stuck pipe argument in Figure 4.33.

	Argument	Case	UID	Subject	Predicate	Object	Level	Polarity
0	stuck pipe	0	default	stuck pipe	not found in	comment	comment	-
1	stuck pipe	1	A	stuck pipe	found in	linked entity	comment	+
2	stuck pipe	2	B	stuck	found in	domain phrase	comment	+
3	stuck pipe	2	A	stuck pipe	found in	linked entity	comment	+
4	stuck pipe	3	C	psi	found in	measurement	comment	+
5	stuck pipe	3	D	jar	found in	linked entity	comment	+
6	stuck pipe	3	A	stuck pipe	found in	linked entity	comment	+
7	stuck pipe	4	E	jar	found in	domain phrase	comment	+

Figure 4.33: Cases Example

The feature to case feature mapping process iterates through the input features tables and calls a feature table-specific conversion function which uses Numpy Vectorisation to iterate through the feature table. The conversion function is dependent on the feature table. For example, the domain phrase conversion function outputs two case features for each input feature table row which are a 'is a' semantic triple between the character span and the domain phrase as well as a 'found in' semantic triple between the domain phrase and the feature table.

Default Case

The default case is a case that contains only a singular case feature which is the default outcome of the argumentation framework debate. For the purposes of calibration extraction, the argumentation frameworks are defined for drilling events with a fixed default case in the format shown in Table 4.23.

Column	Description	Format
Argument	The drilling event	(Stuck Pipe, Kick, Tight Hole, ...)
Case	The default case number	0
UID	The default case feature UID	default
Subject	The drilling event	Stuck Pipe, Kick, Tight Hole
Predicate	The default case opposing predicate	not found in
Object	The lowest context level	comment
Level	The lowest context level	comment
Polarity	The default case negative polarity	-

Table 4.23: A table showing the default cases format for drilling events

The default case is always identified with a case identification number of zero with a UID identification hash of 'default'. The default case sets up the drilling event argumentation framework with a premise of the drilling event not being in the comment as drilling events are extreme rare events. The subsequent case arguments try to counter and support this default stance which has a negative default polarity since the default case is arguing for a negative event presence. This can easily be interchanged with a positive polarity as long as the subsequent case arguments are consistent.

New Case

A new case is generated using the feature to case feature mapping process from the feature tables defined in a configuration dictionary for a query set of comments. The configuration dictates which feature tables to convert as well as any custom parameters to use to filter the number of case features produced. An example part of a configuration dictionary for the feature mapping process is shown in Figure 4.34.

```
feature_table_config = {
    "phrase_table": {
            "use": True,
            "parameters": {
                "name": phrase_table_welltrack_60K,
            },
        },
    "domain_phrase_table": {
            "use": True,
            "parameters": {
                "name": domain_phrase_table_welltrack_sixty,
                "argument": "StuckPipe"
            },
        },
```

Figure 4.34: Case to Case Feature Mapping Configuration Example

The new case has the same schema as shown in Table 4.22 but with the argument column replaced by the comment identification number. However, similar to the default case, the identification metadata is fixed as shown in Table 4.24.

Column	Description	Format
Comment ID	The comment identification number	Integer number
Case	The new case	New
UID	The unassigned UID	None
Level	The lowest context level	comment
Polarity	The unassigned polarity	?

Table 4.24: A table showing the new cases format for comments

The case number is assigned a special keyword 'New' to differentiate it from the integer cases numbers in the generic cases. Since this is a new case, the UID hash and the polarity are unknown as they are defined relative to the pre-defined argumentation cases and the argumentation framework default outcome respectively. An example new case table is shown in Figure 4.35 with the aforementioned format.

	CommentID	Case	UID	Subject	Predicate	Object	Level	Polarity
29	1192	New	None	torque	is a	Noun	comment	?
59	1192	New	None	stuck	found in	domain phrase	comment	?
20	1192	New	None	at	is a	Adposition	comment	?
56	1192	New	None	work	found in	domain phrase	comment	?
32	1192	New	None	and	is a	Conjunction	comment	?

Figure 4.35: New Case Example

Predefined Argumentation Cases

The argumentation framework is constructed around pre-defined argument cases created manually with the help of a domain expert and stored in CSV files. A table of defined argumentation frameworks is loaded into the SQLite3 Database using the Data Loader API with an example table shown in Figure 4.36. This process enables the application to keep track of the potential arguments that can be reasoned about in the subsequent pipelines.

	Argument	FileName
0	stuck pipe	StuckPipe.csv
1	cavings	Cavings.csv
2	fit	FIT.csv
3	kick	Kick.csv
4	losses	Losses.csv
5	tight hole	TightHole.csv

Figure 4.36: Argumentation framework file definitions

The defined argumentation frameworks are then parsed to generate a case table with the schema defined in Table 4.22. The process of loading the cases in this two-step process ensures that all the pre-defined cases are collected before reassigning the UID alphabetical hash for the case table to avoid a hash conflict in the argumentation CSV files.

4.9.2 Abstract Argumentation Framework

The abstract argumentation framework consists of arguments as nodes and attack relations between the arguments as edges in the directed argumentation graph. The arguments for the framework are the cases extracted from the pre-defined argumentation cases and a new case generated for a query comment. The arguments are represented by their case number for simplicity in processing and visualising in graphical outputs.

The attack relations for the framework are a tuple of a case and its polarity attacking a different tuple of a case and its polarity. For example an attack relation could be for cases 0 and 1 with polarities '-' and '+' respectively: (0, '-') attacks (1, '+'). The attack relations are defined by four conditions based on different case sets of pre-defined cases and new cases which are shown in Table 4.25 [CST16].

Attack Relation	Description	Case Set
Different Outcomes	The polarities of the cases are different	pre-defined cases
Specificity	The attacked case features are a subset of the attacker case features	pre-defined cases
Concision	The attacker case feature set must be the subset minimal potential attacker case feature with the same attacker polarity	pre-defined cases
Irrelevance	The attacked feature set is not a subset of the new case feature set when the new case is the attacker	pre-defined cases and the new case

Table 4.25: A table showing the attack relations conditions for abstract argumentation with case-based reasoning

The argumentation graph is constructed in two stages. The first stage builds the graph with the first three attack relations for pre-defined cases only. The nodes of the graph are taken as the cases defined in the pre-defined cases for the argumentation graph (e.g. stuck pipe). The graph edges are then defined recursively starting from the root of the argumentation graph which is the default case.

The first recursion takes the default case as the current case and the list of all cases as an argument. Each recursion generates an attack edges by successively filtering the given list of cases using the first three attack relation conditions. Different outcomes cases are calculated by removing the cases with the same polarity as the current case from the given case list in the recursion. Specificity cases are calculated by finding all the cases in the different outcome cases with case features that are a subset of the current case in the recursion. Concision cases are generated by iterating over the specificity cases and removing all cases which are not subset minimal with respect to the other specificity cases.

The directed attack edges for the case in the recursion can now be added using the cases left in the concision cases and the recursion case. The recursion continues with the cases in the concision set being the new recursion cases and the case list argument being the same as the input case list bar the current case in the recursion. The recursion terminates when the input case list is empty or there are no concision cases that are generated. Since every recursion reduces the case list, the recursion is guaranteed to terminate.

The constructed argumentation graph defined by the directed edges and cases as nodes can be used to query a comment for the default outcome by adding the new case generated from that comment into the argumentation graph in a second stage with the fourth attack relation condition. Before the second stage, the new cases are labelled with a UID which matches the UID for the pre-defined cases so that the case features are consistently labelled and comparable in the downstream processes. The second stage simply iterates through all the cases in the pre-defined case set and adds attack relation edges based on the irrelevance attack relation condition. The attack edge between the new case and the pre-defined case is added if the pre-defined feature case is not a subset of the new case feature set and the pre-defined case is not the default case. The default case feature set is the empty set in the argumentation framework and since the empty set is always a subset of the new set, there cannot be a direct attack relation between the default case and the new case.

An example argumentation graph for stuck pipe and a query comment is shown in Figure 4.37. The nodes show the case numbers which are defined with the corresponding case features in Figure 4.38. The node colour green represents a positive polarity while the node colour red represents a negative polarity. The new case does not have a colour since its polarity is unassigned. The directed edge shows the attack relation between cases.

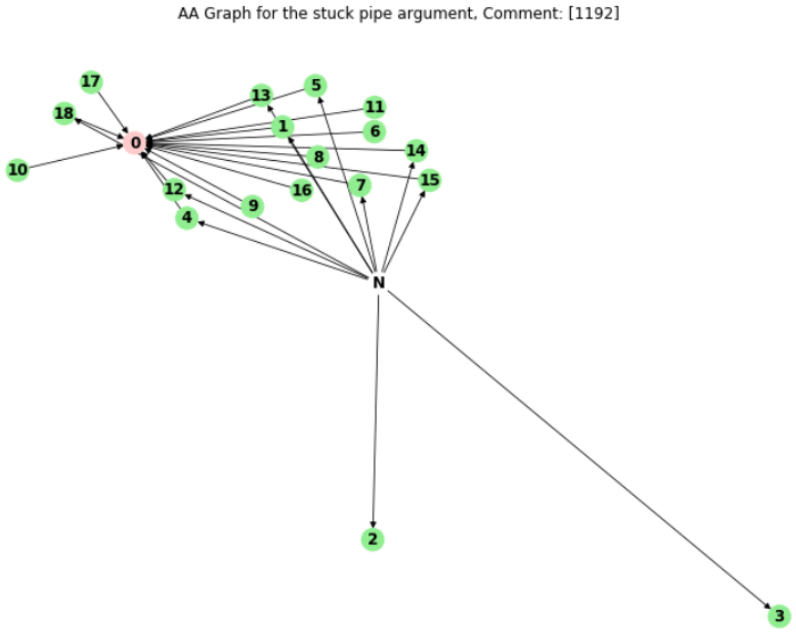


Figure 4.37: Argumentation Graph for Stuck Pipe Example

	Case	Polarity	Features
0	0	-	default
1	1	+	A
2	2	+	B,A
4	3	+	C,D,A
7	4	+	E,B
8	5	+	F,B
9	6	+	G,B
10	7	+	H,B
11	8	+	I,B
12	9	+	J,B
13	10	+	K,B
14	11	+	L,B
15	12	+	M,B
16	13	+	N,B
17	14	+	O,B
18	15	+	P,B
19	16	+	C,B
20	17	+	Q,B
21	18	+	R,B

	Argument	Level	Object	Predicate	Subject	UID
0	stuck pipe	comment	comment	not found in	stuck pipe	default
1	stuck pipe	comment	linked entity	found in	stuck pipe	A
2	stuck pipe	comment	domain phrase	found in	stuck	B
4	stuck pipe	comment	measurement	found in	psi	C
5	stuck pipe	comment	linked entity	found in	jar	D
7	stuck pipe	comment	domain phrase	found in	jar	E
8	stuck pipe	comment	domain phrase	found in	backream	F
9	stuck pipe	comment	domain phrase	found in	torque	G
10	stuck pipe	comment	domain phrase	found in	spike	H
11	stuck pipe	comment	domain phrase	found in	free	I
12	stuck pipe	comment	domain phrase	found in	work	J
13	stuck pipe	comment	domain phrase	found in	string	K
14	stuck pipe	comment	domain phrase	found in	circulation	L
15	stuck pipe	comment	domain phrase	found in	pipe	M
16	stuck pipe	comment	domain phrase	found in	bleed	N
17	stuck pipe	comment	domain phrase	found in	bled	O
18	stuck pipe	comment	measurement	found in	ton	P
20	stuck pipe	comment	measurement	found in	gpm	Q
21	stuck pipe	comment	measurement	found in	lpm	R

Figure 4.38: Argumentation Graph Example Case Table (Left) and Example Feature Table (Right)

4.9.3 Grounded Extension

The grounded extension is the set of unattacked case arguments in the argumentation graph union the set of arguments that the unattacked case arguments set defends [CST16]. The grounded extension is chosen as the set semantic for evaluating the argumentation framework as it is always guaranteed to exists and also provides a unique set of arguments to evaluate the polarity of the new case with respect to the default outcome.

Since the new case argument is unattacked by the definition of irrelevance it is always in the

grounded extension and the grounded extension provides a set of coherent arguments. Therefore, the polarity of the new case is determined by whether it is in the grounded extension with the default case or not. If the new case and default case occur in the grounded extension together than they have the same polarity. For the drilling event argumentation frameworks, this would mean that the drilling event does not occur in the query comment. In the opposite scenario where the new case and default case do not occur in the grounded extension, the polarity of the new case would be opposite to the default case. Therefore, the drilling event would be in the query comment.

An example grounded extension is shown in Figure 4.39 for the argumentation graph in Figure 4.37. In this example, since the default case is not in the grounded extension, the drilling event stuck pipe does occur in the example comment.

```
{'16', 'N', '11', '6', '8', '9', '10', '17'}
```

Figure 4.39: Grounded Extension Example

Two contrasting approaches were used when calculating the grounded extension with the inputs being the nodes and edges of the argumentation graph.

Exhaustive Grounded Labelling Algorithm

The first approach was a top-down approach concerned with exhaustively calculating the grounded extension. The approach involved computing all the successive argumentation set semantics which constitutes the definition of the grounded extension. The arguments were labelled as either in, out or undecided for each of the set semantics required to satisfy the definition of the grounded extension set. This method starts with the computation of all conflict-free sets which are defined as sets of arguments which do not attack themselves. The set of conflict-free sets are reduced to admissible sets which are defined as conflict-free and are sets which attack each attacking argument. The set of the admissible sets would then be reduced to only complete sets which are defined as admissible sets which contain all arguments the set defends. Finally, the complete sets would be reduced to the unique grounded extension which is defined as the minimally complete set with respect to subset inclusion.

Initially, with a small case set, this method was computable but the calculation of the complete sets relied on generating power sets of the admissible sets in order to calculate all possible permutations of complete sets. This approach was not able to scale due to the intractability of calculating the power sets of large case sets. Therefore, this approach was not chosen as the case sets needed to be scale-able in order to handle complex argumentation frameworks.

Fixpoint Grounded Labelling Algorithm

Given the intractability of the first approach when scaling up the number of cases in the argumentation framework, an alternative approach had to be implemented. The fixpoint algorithm takes the approach from the bottom up in the sense it builds up the grounded extension from cases which are guaranteed to be in the grounded extension [Dun95].

Before the fixpoint algorithm starts, the input edges representing the attacks are pre-processed into an attack dictionary which has the attacker nodes as a key and a list of attacked nodes as a value. The set of unattacked arguments is calculated next which is the set difference between the set of all nodes and the set of attacked nodes which can be calculated through the taking all the second values in the attack tuples.

The set of unattacked arguments becomes the in set and an empty set is initialised as the out set. A two-step iterative process is then conducted until there is no change in the grounded extension i.e. a fixpoint is reached. The first step is adding all arguments attacked by the in set arguments to the out set. The second step is adding all arguments attacked by the new out set to the in set. When there is no change in the size of the in and out set, the resultant sets form the grounded labelling of the argumentation graph with the nodes not in the in and out set being

labelled undecided. The construction of the argumentation framework ensures that the grounded extension is unique and exists and hence this iterative process will always terminate. The in set then represents the grounded extension.

4.9.4 Dispute Tree

A dispute tree for the default case is a subtree of the argumentation graph where every case is labelled either a proponent or opponent node [CST16]. The root of the tree which is the default case is always a proponent node. For every proponent node, all attacks against it are added to the dispute tree where the attacking cases are labelled opponent nodes. For every opponent node, there is at most one proponent node attacker.

When the new case has the same polarity as the default case, an admissible dispute tree which has the aforementioned properties can be generated. However, when the new case has a different polarity to the default case, an admissible dispute tree will not be possible. In which case, a maximal dispute tree can be generated instead where all opponent nodes which are leaves in the dispute tree have no attackers as shown in Figure 4.40 generated from the dispute tree in Figure 4.37. The green coloured nodes are proponent nodes while the red coloured nodes are opponent nodes and the new case is left white as a special case.

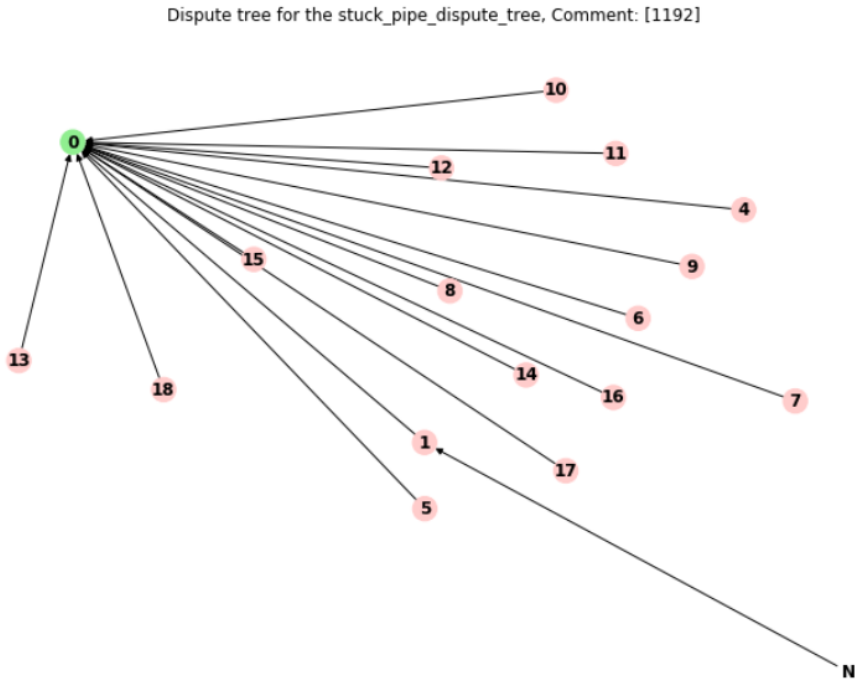


Figure 4.40: Dispute Tree Example

The dispute tree is computed through traversing the argumentation graph recursively starting with the default case. The traversal visits the nodes alternating between labelling adjacent nodes as proponents and opponent nodes. If the current recursion node is an opponent node and there is more than one proponent attacker, only a single proponent attacker can be chosen to satisfy the admissible dispute tree constraints. While any choice of proponent attacker will yield a dispute tree, a depth-first search is conducted on the successor nodes of the current recursion nodes in order to choose the proponent attacker which has the greatest depth. A weighting is added to nodes which contain the new case in the depth-first search as the final explanation will be better in the context of the query comment. This does add to the computation time of the dispute tree but yields a better explanation as the more cases in a path for the dispute tree, the better the generated debate regarding the default outcome will be. In all other cases of recursion node, the successors nodes are recursed with the node identity (proponent or opponent) being switched.

While the grounded extension is enough to calculate the polarity of the new case and hence, the presence of the drilling event in the comment, it does not provide a satisfactory explanation beyond the other nearest cases in the grounded extension. The grounded extension is transparent as it shows supporting and conflicting nearest cases but it does not show the argumentative trace across similar cases which provides a better explanation. Traversing the dispute tree generates a set of cases which argue for and against the default case providing a more informative explanation of the argument premise with a hierarchical relation between cases rather than a flat set of transparent cases [CST16].

4.9.5 Argumentation Explanation

The value in this machine arguing methodology is in the generation of natural explanations to reason about the default case for the argumentation framework. Since the case features are RDF semantic triples, they can be converted very easily into natural language which is one of the main motivations of encoding the extracted features into the RDF semantic triple format. An example explanation generated for the dispute tree in Figure 4.40 is shown in Figure 4.41.

```
stuck pipe not found in comment is False as
stuck found in domain phrase and torque found in domain phrase and free found in domain phrase and work found in domain phrase
and string found in domain phrase and circulation found in domain phrase and psi found in measurement and gpm found in measurem
ent
```

Figure 4.41: Generate Explanation Example

Only a proof of concept implementation was created for this explanation process in the argumentation pipeline. The current implementation of the explanation generator traverses each level of the dispute tree extracting the natural language format of the RDF semantic triple and outputting at different levels connected by conjunctions.

4.9.6 Automated Cases

The focal point of the argumentation pipeline is in the definition of the argumentation framework cases by a domain expert. The case-based reasoning framework is designed so that the domain expert can knowledge transfer their insights when reasoning about the presence of drilling events in a natural way. However, while the domain expert defined cases cover most of the drilling event cases, there may be some underlying case features that correspond to drilling events that are not covered. Therefore, an automated case extraction solution was attempted in order to augment the case set for a drilling event argumentation framework.

Test Train Split

The training data set evaluation CSV file with labels for four drilling events including 'stuck pipe' was loaded into the SQLite3 Database. In order to evaluate the performance of the automated case generation, the training data set was split into a test and train set with 80% of the data set being the train data set.

Comment Case Extraction

A process in the Argumention API computes the cases defined per comment by extracting the case features from the feature tables defined in a configuration shown in Figure 4.34. A label mapping is necessary to translate the training data set event labels to the internal application drilling event labels as the event labels across different data sets may have different styles of writing. Once the case features have been extracted for all comments in the train data set, they are assigned a unique UID hash as well as a new case id which does not conflict with the predefined cases for the query argument. Using the event labels, a binary polarity is assigned to the extracted cases based on the query argument.

Case Scoring

The number of cases generated from the training data set train set was too large to add directly to the argumentation framework as it increased the processing time of the argumentation generation

pipeline and would result in extreme overfitting. Therefore, the number of cases and case features needed to be filtered to only retain the most relevant ones in relation to the argument drilling event. Two case scoring approached were implemented which are term frequency inverse document frequency (TF-IDF) and case feature frequency counts.

The case feature frequency count approach involved counting the cases for positive and negative polarity comments with respect to the presence of the drilling event argument. The thousand most common case features for each polarity were then kept and converted into cases to be inputted into the pre-defined cases for the argument. This method did not yield a significant improvement over the domain defined only cases as it generated too many false positives and simply increased the processing time for the argumentation pipeline significantly. The approach did not take into account the contextual case features of the most common case features which is necessary to form a more representative case argument.

Term frequency inverse document frequency is normally used to evaluate how important a word is to a document in a corpus. In this approach, the TF-IDF score was used to give a weighting to how important case features are relative to cases in the data set. TF-IDF is a good case score as it accounts for varying length cases in terms of the number of case features as well as weighting down frequent case features and scaling up rare case features. The importance of the case feature increases proportionally to the number of times it appears in a case offset by the frequency of the case in the data set.

The formula used to calculate term frequency which is a measure of the frequency of a case feature in a case is shown below [Wik20c]:

$$TF(c) = \frac{Number\ of\ times\ case\ feature\ c\ appears\ in\ a\ case}{Total\ number\ of\ case\ features\ in\ the\ case}$$

The formula used to calculate inverse document frequency which is a measure of the importance of a case feature in the data set is shown below [Wik20c]:

$$IDF(c) = log_e\left(\frac{Total\ number\ of\ cases}{Number\ of\ cases\ with\ case\ feature\ c\ in\ it}\right)$$

The TF IDF case feature score is calculated for two subsets of the training data set which are sets of positive polarity and negative polarity cases. The desired case feature score is a singular case feature score across all input cases rather than a case feature score in relation to each case. Therefore, the case feature score for each case was aggregated using the mean case feature score. Other aggregation functions such as min and max case feature scores were experimented with but yielded worse results. The aggregated case feature score for the negative polarity set is then subtracted from the case feature score for the positive polarity set to create a case feature score for all case features in the training data set. At this point, there are three aggregated case feature score sets: the positive cases only case feature score, the negative cases only case feature score and the all cases weighted case feature score. The cases and case features of the whole data set can then be filtered using the three aggregated case feature scores sets.

Case Filtering

The cases are iterated through three levels of filtering. The first level of filtering looks at a score for a case rather than a case feature to get a measure of the uniqueness of the case weighted towards the positive polarity. The case score is defined as the average case feature score using the all cases weighted case feature score. The average aggregation function yielded better results than the min or max aggregation function in the calculation of the case score. For negative polarity cases, the case score was filtered to be in the range 0 and -0.1. For positive polarity cases, the case score was filtered to be in the range 10 and -10. These values were chosen after experimentation to yield the most common cases while limiting the number of case features generated. The number of positive polarity cases is far fewer than the negative polarity cases, hence why the threshold limit has a greater range. Figure 4.42 shows the average feature case score before using the filtering thresholds while figure 4.43 shows the average feature case score for the selected cases after filtering.

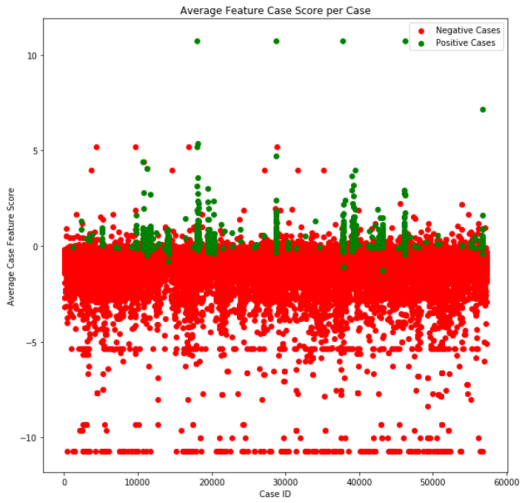


Figure 4.42: Average Feature Case Score per Case

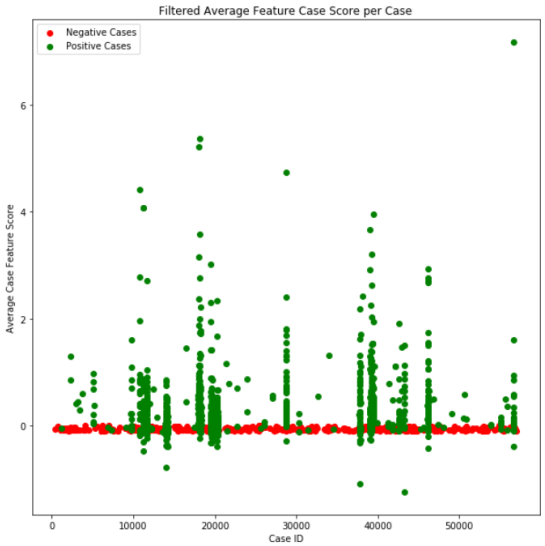


Figure 4.43: Filtered Average Feature Case Score per Case

The second level of filtering looks at the positive cases only case feature score and the negative cases only case feature score to try to choose case features which are the most unique, weighted towards a positive polarity and a negative polarity respectively. For positive cases, a case feature is chosen if it has a positive cases only case feature score of greater than 0. For negative cases, a case feature is chosen if it has a negative cases only case feature score of greater than 3.0. These threshold values were chosen after experimentation to balance the number of case features chosen and the processing time of the argumentation pipeline. Figure 4.44 shows the positive and negative case feature scores of the case features that have been chosen.

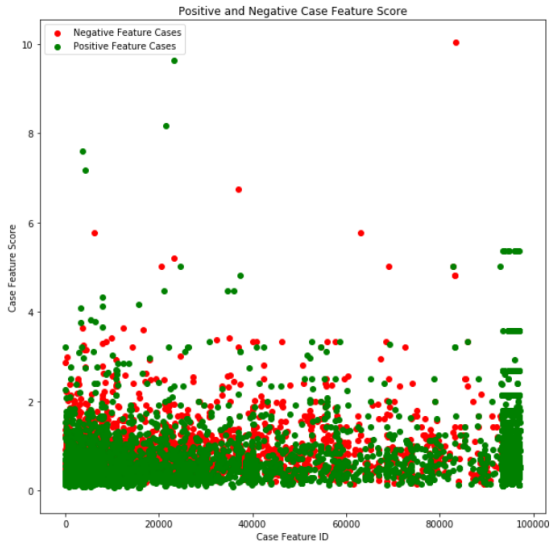


Figure 4.44: Positive and Negative Case Feature Score

The final level of filtering converts then chosen cases and case features into useful case arguments. A case is not useful if it only has a single feature, therefore only cases that have more than one feature are chosen. Furthermore, the number of case features per case is set to a maximum of 6 which has been chosen after experimentation. The case features with the maximum positive case feature score or negative case feature are chosen.

The filtering process reduces the number of cases chosen from 44140 to 2071 and the number of case features from 97098 to 7895 in the training data set. The final filtered cases can then be added to the pre-defined cases for the query argument which in the examples was 'stuck pipe'.

4.9.7 Bulk Processing Comments

The classification of the polarity of the new case generated from a comment as well as the generation of an explanation of the outcome encompasses all the aforementioned argumentation processes. Scaling up the process for all 56,000 comments in the training data set required optimisations in order to finish the computation in a reasonable time. The bulk processing process produces an explanation table which has the schema shown in Table 4.26.

Column	Description
Comment ID	The internal comment identification number from which the argument explanation has been extracted
Argument	The name of the argumentation framework processed
Agree?	Boolean value to state whether the explanation agrees with the default outcome of the argument
Explanation	The natural language explanation generated from the dispute tree
Grounded Extension	The grounded extension of case numbers for the argumentation framework

Table 4.26: A table showing the schema for the argumentation explanation table

In order to avoid out of memory issues for larger data sets, the input query data set of comments is divided into batches of 20,000 comments for processing. Each batch output updates a table generated using a PUT API request to the Data Layer API for the query argument and data set. The case features for the set of query comments are pre-computed and stored in a table dictionary to be indexed when iterating over the query comments to generate explanations for them. The argumentation graph for the predefined cases is also pre-computed and only the list of nodes and edges are kept so that they can be augmented with a new case for each comment iteration. The main explanation loop iterates through the query comments to generate the new case augmented argumentation framework, the grounded extension, the dispute tree and finally the explanation. An example output argumentation explanation table is shown in Figure 4.45.

	CommentID	Argument	Agree?	Explanation	GroundedExtension
1192	1192	stuck pipe	False	stuck pipe not found in comment is False as\nstuck found in domain phrase and torque found in domain phrase and free found in domain phrase and work found in domain phrase and string found in domain phrase and circulation found in domain phrase and psi found in measurement and gpm found in measurement\n	{'16', 'N', '11', '6', '8', '9', '10', '17'}
12312	12312	stuck pipe	True	stuck pipe not found in comment is True as\ntorque found in domain phrase and psi found in measurement and lpm found in measurement\nlpm found in measurement	{'0', 'N'}
12518	12518	stuck pipe	True	stuck pipe not found in comment is True as\nstring found in domain phrase\nstring found in domain phrase	{'0', 'N'}
13065	13065	stuck pipe	False	stuck pipe not found in comment is False as\nstuck found in domain phrase and free found in domain phrase and work found in domain phrase and circulation found in domain phrase and psi found in measurement and lpm found in measurement\n	{'16', '9', 'N', '8', '18', '11'}

Figure 4.45: Argumentation Explanation Table Example

4.10 Calibration Point Generation

The main output of the application is calibration points which are a combination of a drilling event and measurements such as mud weight or depth measurements. The majority of the work has been done in the upstream processes which results in the calibration point pipeline mainly being used to process the previous process outputs into the desired output schema which is defined in Table 4.27. The rows are formatted such that there is a single measurement in each row as there can be many measurements extracted for a single comment. There can also be many explanations for each comment as the comment could be queried for different drilling event which would each have their own explanation. However, for the sake of simplicity a separate calibration point output table will be generated for each drilling event argument.

Column	Description
Comment ID	The internal comment identification number from which the argument explanation has been extracted
Location	The location of the drilling operation which is normally a borehole or a well name
Start Date	The starting date of the drilling operation described in the comment
Start Time	The starting time of the drilling operation described in the comment
End Date	The end date of the drilling operation described in the comment
End Time	The end time of the drilling operation described in the comment
Comment	The raw input comment
Argument	The name of the argumentation framework processed
Agree?	Boolean value to state whether the explanation agrees with the default outcome of the argument
Explanation	The natural language explanation generated from the dispute tree
Grounded Extension	The grounded extension of case numbers for the argumentation framework
Unit Type	The format of the unit of the measurement which can be a Single Unit or Composite Unit
Start Character	The starting character index in the comment of the measurement
End Character	The ending character index in the comment of the measurement
Unit	The measurement unit
Minimum Value	The measurement in case the measurement was recorded as a range
Maximum Value	the measurement in case the measurement was recorded as a range

Table 4.27: A table showing the schema for the calibration point table

The calibration point generation process takes the raw comment, the measurements table and the explanations table for the query data set. It also takes a unit list parameter which is a list of units to retain from the measurements table. For calibrations points, the unit list contains depth units such as 'm', 'ft' as well as mud weight units such as 'mw', 'sg' and 'ppg'. The process then does a left join between the raw comments and explanations table on the comment identification number. Then another left join is done using the output from the first join and the measurements table on the comment identification number again. An example output for the calibration point generation pipeline is shown in Figure 4.46.

	CommentID	Location	StartDate	StartTime	EndDate	EndTime	Comment	Argument
2048	1192	ABJF-256	27/10/2014	00:00:00	27/10/2014	01:00:00	Got stuck at 7090 while reaming up stand. Worked string with circulation (unable to rotateï¼ torque higher them 15 klb.ft) and GOT FREE.- Partial pack-off 350 GPMï¼ 3700 PSIï¼ out of bottom.- Pumped 10 bbl Lo and 10 bbl Hi-Vis	stuck pipe

	Agree?	Explanation	GroundedExtension	UnitType	StartChar	EndChar	Unit	MinValue	MaxValue
2048	False	stuck pipe not found in comment is False as\nstuck found in domain phrase and torque found in domain phrase and free found in domain phrase and work found in domain phrase and string found in domain phrase and circulation found in domain phrase and psi found in measurement and gpm found in measurement\n	{'16', 'N', '11', '6', '8', '9', '10', '17'}	Unit	115.0	126.0	ft	15	15

Figure 4.46: Calibration Point Table Example

The final output of the workflow is the calibration point CSV file which is generated by simply using the Data Layer API save function which fetches a table such as the calibration point table into a Pandas dataframe. The dataframe can then be outputted directly using a Pandas function into a CSV file saved in the Results folder of the current experiment.

Chapter 5

Evaluation

This chapter aims to present the results of the application evaluated with the data sets shown in Section 2.2. The goal of this application is to replace the traditionally manual process of calibration point extraction with an automated solution. Therefore, the main form of evaluation for this project is through comparison between the automated application and the traditional manual analysis for calibration point extraction in three key areas which are accuracy, coverage and performance. Another sub-goal of this project is to improve upon the results achieved by a different methodology implemented for calibration point extraction which is naive keyword indexing. The nature of the comments being scientific statements of fact results in domain keywords being a highly informative feature for drilling event extraction. Naive keyword indexing string searches for these domain keywords in a processed data set and thus provides a good lower bound evaluation method to compare the current implementation framework against. However, naive keyword indexing results in a lot of false positives being generated as contextual information is not taken into account when extracting the domain keywords.

5.1 Accuracy

Calibration points consist of a measurement and a drilling event. The accuracy evaluation for calibration points is examined only for the drilling event aspect of calibration points as calibration points only occur when there is the presence of certain drilling events. The drilling event labelled comments represent on average less than 3% of the total comments in a given data set. Once a comment with a drilling event has been flagged, a domain expert can lookup the comment and extract the correct measurement from the generated measurement table in the application. Therefore evaluating measurement extraction is not necessary and only the results of the drilling event classification will be evaluated.

The classification metrics used to assess the accuracy of extraction are defined in Section 2.6. The classification of drilling events is done as a binary classification for each query drilling event with the extraction result being the drilling event or a no event. The design of the workflow results in a calibration point file generated for each drilling event as a binary classification due to the use of the argumentation framework being a binary classification method in feature analysis. The drilling events that are assessed in the evaluation are stuck pipe, kick, tight hole, losses, cavings and FIT. The definitions of these drilling events can be found in Section 2.1.2. These drilling events are chosen as they are found in the labelled test data set and also represent the most informative drilling events for calibration points.

Generally, the drilling event extraction differentiates between drilling event comments (true positives) and null drilling events comments (true negatives) quite well. Given the heavily unbalanced nature of the data set where drilling events are extreme events in the data set, the general classification accuracy score is not an informative evaluation metric as it is too sensitive to class imbalance with the true negatives far outweighing the true positives.

A better classification metric would be recall and precision as well as F1-score as they are asymmetric to class imbalance due to the fact they ignore true negatives and only consider true positives,

false positives and false negatives. Precision is an important metric for evaluation as a goal of this project is to reduce the false positive rate which is high in the naive keyword extraction method. Recall is a more important evaluation metric as false positives can always be manually refined by a domain expert from the calibration point output of the application. However, false negatives cannot be reduced for unlabelled data sets and given the rarity of drilling event occurrence in the data set, every missed drilling event affects the calibration of the downstream geomechanics model.

Matthews correlation coefficient is an informative evaluation metric as it is a balanced evaluation measure for the unbalanced data set with regards to drilling event occurrence. The correlation coefficient between the true drilling event and the predicted drilling event is captured and the symmetric evaluation metric accounts for all four metrics in the confusion matrix.

The public Australian data set is a data set with 5868 comments which has been manually labelled by a domain expert and has been used as a test data set to evaluate the accuracy of the drilling events extraction for calibration points. The extracted feature tables and domain phrases have been developed to maximise the accuracy for the training data set only to get an unbiased evaluation on the test set. The manually labelled dataset will represent the upper bound (the gold standard) in the accuracy evaluation bearing in mind that there may be human errors in the labelling of the public Australian data set. Furthermore, the lower bound in the accuracy scores will be the naive keyword indexing methods which will also be evaluated against the public Australian manually labelled data set. The goal of the project implementation is to meet the gold standard of extraction while at least exceeding the evaluation metrics of the naive keyword method.

5.1.1 Stuck Pipe

The confusion matrix for Stuck Pipe extraction is shown in Figure 5.1 and the evaluation metrics are shown in Table 5.1 for both the project implementation and the keyword indexing implementation.

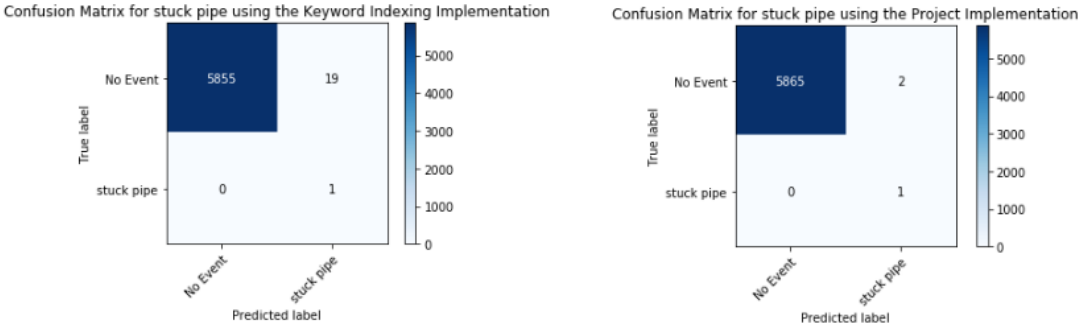


Figure 5.1: Confusion Matrix for Stuck Pipe Evaluation

	Project Implementation	**Keyword Indexing Implementation**
Precision	0.33	0.05
Recall	1.00	1.00
F1-Score	0.50	0.09
Matthews Correlation Coefficient	0.57	0.22
No. of Events	1.00	1.00
Classification Accuracy (all comments)	99.96	99.67

Table 5.1: A table showing the stuck pipe classification results for both extraction methods on the test data set

The public Australian data set for evaluating stuck pipe events is limited as it only has one stuck pipe event. However, the consequence of stuck pipe events is quite severe in terms of financial loss, especially in offshore drilling. Should they occur, they are clearly stated in the drilling report

rather than having an ambiguous surface mention of the key domain phrases so false negatives are unlikely to occur with the current methodology using domain phrases feature. The main goal with stuck pipe extraction is to reduce the number of false positives. This project implementation is successful in achieving this goal compared to the keyword indexing implementation as the precision score is much higher relative for the project implementation. The low precision score for the project implementation of 0.33 is mainly due to there being only one stuck pipe event. The two false positives generated for the project implementation are within the scope of a domain expert to refine.

5.1.2 Kick

The confusion matrix for Kick extraction is shown in Figure 5.2 and the evaluation metrics are shown in Table 5.2 for both the project implementation and the keyword indexing implementation.

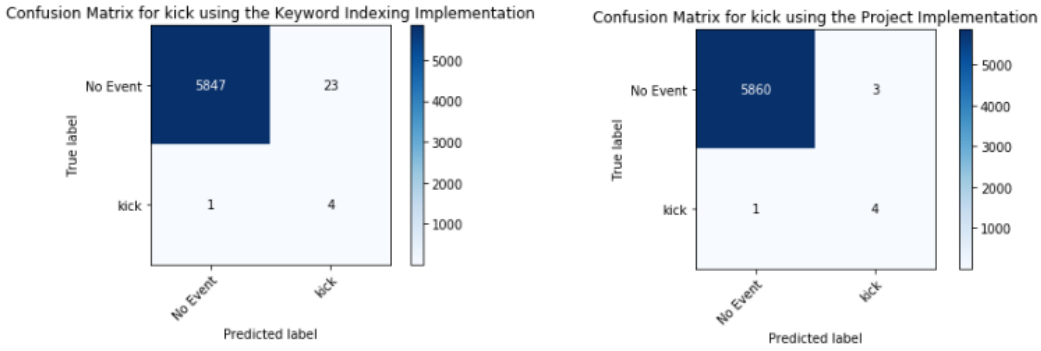


Figure 5.2: Confusion Matrix for Kick Evaluation

	Project Implementation	**Keyword Indexing Implementation**
Precision	0.57	0.14
Recall	0.8	0.8
F1-Score	0.66	0.25
Matthews Correlation Coefficient	0.67	0.34
No. of Events	5	5
Classification Accuracy (all comments)	99.93	99.59

Table 5.2: A table showing the kick classification results for both extraction methods on the test data set

Kick events also show improvement in reducing the false positive rate with the precision for the project implementation being far greater than the keyword indexing implementation. However, there is no improvement in the reduction of the false negative shown in Figure 5.3.

	CommentID	Comment
596	277	Observed an increase of 1.19m3 (12bbls) in active system. P/ U off bottom, shut down mud pumps, spaced out and monitored well on the trip tank. Informed NDSV and Toolpusher. Confirmed all crane operations and mud transferring stopped. Flow checked, well static.

Figure 5.3: False Negative Example Comment for Kick

The false negative is missed due to a surface mention variation ('increase') for one of the domain phrases ('gain') extracted from the comment which was not accounted for in the kick argumentation pre-defined cases. The best solution to avoiding this false negative would be the utilise the entity linking system to map the surface mention variation to a linked entity domain phrase rather than hard coding the surface mention variation in the pre-defined cases. This solution would result in better coverage and avoid overfitting the domain phrases to a particular data set.

The false positives generated were due to an unforeseen case where the possibility of a kick event was tested for with a flow check on the trip tank but resulted in a loss event instead shown in Figure 5.4.

	CommentID	Comment
939	470	Observed gain in active system. Flow checked well on trip tank, 3bbl/hr losses. Confirmed loss rate with GeoService. Offline operations: added defoamer to mud pits.

Figure 5.4: False Positive Example Comment for Kick

The way of reducing this false positive is to add a counter-argument to the flow check case in the pre-defined cases for the kick event to account for the negative result. This highlights in the incompleteness of the current set of domain cases and therefore more knowledge transfer from the domain expert is needed to fully develop the pre-defined domain cases.

5.1.3 Losses

The confusion matrix for Losses extraction is shown in Figure 5.5 and the evaluation metrics are shown in Table 5.3 for both the project implementation and the keyword indexing implementation.

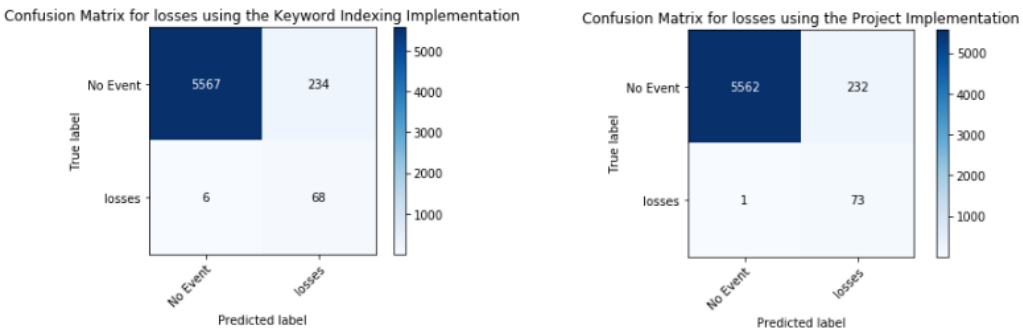


Figure 5.5: Confusion Matrix for Losses Evaluation

	Project Implementation	**Keyword Indexing Implementation**
Precision	0.23	0.22
Recall	0.98	0.91
F1-Score	0.38	0.36
Matthews Correlation Coefficient	0.47	0.44
No. of Events	74	74
Classification Accuracy (all comments)	96.02	95.91

Table 5.3: A table showing the losses classification results for both extraction methods on the test data set

Losses occur far more frequently than the other drilling events which enables a better evaluation for the drilling event extraction process. A significant improvement is in the reduction of the false negative rate in the new implementation method compared to the keyword indexing implementation shown in the increased recall score. The false negatives with the project implementation method are due to the losses event being resolved within the same comment shown in Figure 5.6.

	CommentID	Comment
461	225	M/U TDS and washed to bottom from 1.52MPa (220psi). Observed a loss of returns at 3,057m, picked up string and full returns established. No further losses. S/O weight trending down from: 145.15 MT (320klbs) to 127.0MT (280klbs).

Figure 5.6: False Negative Example Comment for Losses

The argumentation framework does flag the comment as having a loss event but another case argues against the loss event if it has been resolved. The fix for this problem would be having a case which attacks the resolution case if the drilling event occurs in the same comment.

Both implementations flag a lot of false positives as shown by the confusion matrix and low precision scores. On examination of the flagged false positives for losses with the domain expert, the vast majority of them were true positives that were mislabelled. Once a loss event occurs, the loss mentions in the comments generally persist over successive comments. Domain experts are prone to miss these successive mentions which has led to the mislabelling of the loss events as they have to parse through thousands of comments which is a tedious process. Therefore, for losses, the project implementation exceeds the performance of the manual labelling.

Of course, there are examples of verified false positives such as the ambiguous surface mention of loss being confused with a tool rather than a loss event shown for in Figure 5.8. The solution for this case is to use entity linking to correctly map the surface mention to its correct entity based on the contextual information.

	CommentID	Comment
646	310	Confirmed deck count with completion tally: 16 Screens and 18 Blanks remaining, OK. Changed elevators to 127mm (5") drill pipe. P/U and M/ U Lower Completion Assembly B:70-14-17-08: 1 x 140mm (5 1/2") 25.4kg/m (17ppf) JFE Bear pup joint, crossover and 140mm (5 1/2") 25.4kg/m (17ppf) JFE Bear pup joint to 178mm (7") 43.2kg/m (29ppf) JFE Bear, 178mm (7") pup joint, EGF Fluid Loss Control Flapper assembly s/n2622296-08 and 178mm (7") pup joint. M/U connection to 10kNm (7.4kft.lbs). Recorded P/U weight 86.18MT (190klbs) and S/O weight 88.45MT (195klbs). Landed 178mm (7") elevators on C-plate in rotary table. Broke out top half of MUS and handling sub. Rigged up split bushings, MLT C-Plate, TS100 Bowls, 73mm (2 7/8") slips and dog collar. Laid out MUS and handling sub.

Figure 5.7: False Positive Example Comment for Losses

Another example of a verified false positive is due to a loss event occurring but not being a drilling event for a calibration point. The loss event, in this case, occurs in the casing of the well and cannot be used as a calibration point. The solution for this false positive is to expand the pre-defined cases to account for this edge case.

9614	5502	P/U a single of 127mm (5â€) DS50 drill pipe and installed PS16 auto slips. Continued to POOH with Whipstock on 127mm (5â€) DS50 DP at 4min/std to prevent swabbing from 2,450m to 2,062m. Monitored well on trip tank. Average down hole losses: 0.64mÂ³/hr (4bbls/hr). Trip Speed POOH 127mm (5") DS50 (whipstock): 259m/hr.

Figure 5.8: Second False Positive Example Comment for Losses

5.1.4 Cavings

The confusion matrix for Losses extraction is shown in Figure 5.9 and the evaluation metrics are shown in Table 5.4 for both the project implementation and the keyword indexing implementation.

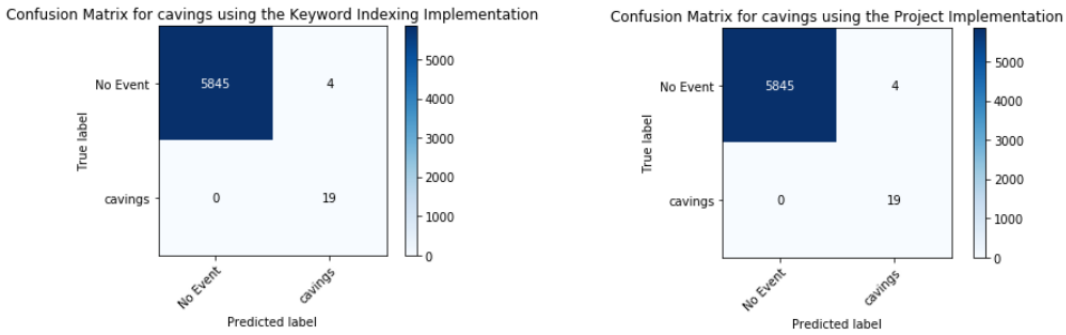


Figure 5.9: Confusion Matrix for Cavings Evaluation

	Project Implementation	Keyword Indexing Implementation
Precision	0.82	0.82
Recall	1.0	1.0
F1-Score	0.90	0.90
Matthews Correlation Coefficient	0.90	0.90
No. of Events	19	19
Classification Accuracy (all comments)	99.93	99.93

Table 5.4: A table showing the cavings classification results for both extraction methods on the test data set

The extraction of cavings was fairly accurate shown by the high F1-score for both the keyword indexing implementation and the project implementation. The project implementation performed identically well to the keyword indexing implementation. However, that was due to both methods performing optimally for caving extraction as on examination with the domain expert for the four false positives generated, they were found to be mislabelled. So in fact both implementations performed better than the manual benchmark and achieved an F1-Score of 1.0 for caving extraction.

5.1.5 Tight Hole

The confusion matrix for Tight Hole extraction is shown in Figure 5.10 and the evaluation metrics are shown in Table 5.5 for both the project implementation and the keyword indexing implementation.

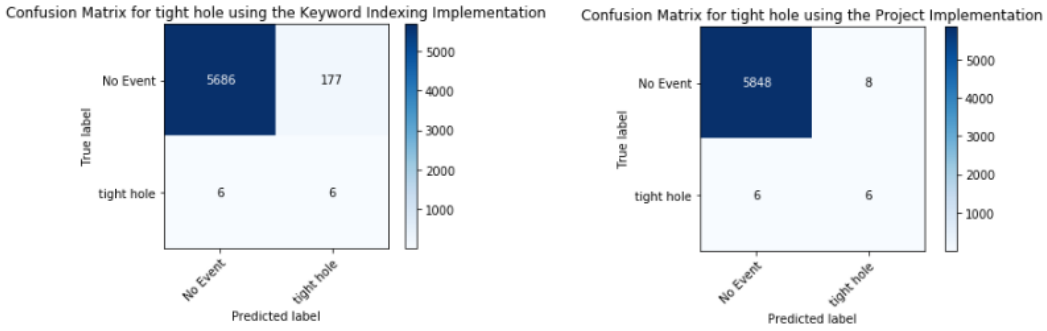


Figure 5.10: Confusion Matrix for Tight Hole Evaluation

	Project Implementation	Keyword Indexing Implementation
Precision	0.42	0.03
Recall	0.5	0.5
F1-Score	0.46	0.06
Matthews Correlation Coefficient	0.46	0.12
No. of Events	12	12
Classification Accuracy (all comments)	99.76	95.88

Table 5.5: A table showing the tight hole classification results for both extraction methods on the test data set

Tight hole is the most difficult drilling event to extract using the current set of features. While the project implementation method greatly reduces the number of false positives, the false negatives occur at a similar rate to the keyword indexing implementation. Upon examination with the domain expert, the false negatives occurred due to a variety of reasons.

The prevailing reason was a missing key case argument for the pre-defined cases when reasoning about the presence of a tight hole shown in Figure 5.11. The missing case was a 'pull out

of hole' operation resulting in 'drag' exceeding a certain threshold being recorded. The solution would be to use features from the linked entities and measurement table to encode this case in the tight hole argumentation framework.

	CommentID	Comment
98	50	POOH racking back 140mm (5 1/2") drill pipe from 1,197m to 989m, drag 40-50klbs from 1,060-1,050m wiped away clean without pumps or rotary.

Figure 5.11: False Negative Example Comment for Tight Hole

The next reason was due to surface mention ambiguity of a domain phrase (Slack Off) that was written as shorthand (S/O) shown in Figure 5.12. The solution for this problem would be to use entity linking to resolve the surface mention to the correct linked entity.

8279	4838	Backreamed out of hole from 2,318m to 1,320m, 10 minutes a stand slip to slip. Observed occasional torque spikes to 27.1kNm (20kft-lbs). Only hole issue observed while backreaming was at 1,700m, unable to S/O, reciprocated stand for 30 minutes until hole good and continued to backream. Recorded pick up and slack off weights every 5 stands. Flow Rate: 4,353Lpm (1,150gpm) Pump Pressure: 25.51MPa (3,700psi) Rotary: 160rpm (High gear) Torque off bottom: 13.56 to 20.34kNm (10 to 15kft-lbs) ECD: 1.28 to 1.29sg

Figure 5.12: Second False Negative Example Comment for Tight Hole

The final reason was due to wrongly assigning the tight hole event to a preceding comment in the labels by the domain expert in the test data set.

5.1.6 FIT

The confusion matrix for FIT extraction is shown in Figure 5.13 and the evaluation metrics are shown in Table 5.6 for both the project implementation and the keyword indexing implementation.

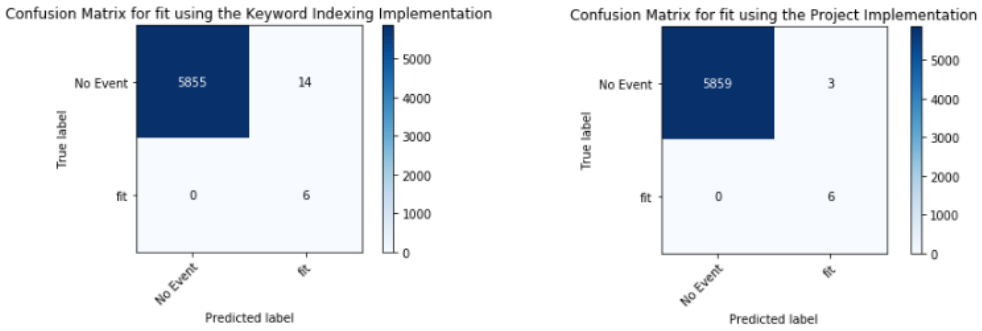


Figure 5.13: Confusion Matrix for FIT Evaluation

	Project Implementation	**Keyword Indexing Implementation**
Precision	0.66	0.3
Recall	1.0	1.0
F1-Score	0.8	0.46
Matthews Correlation Coefficient	0.81	0.54
No. of Events	6	6
Classification Accuracy (all comments)	99.94	99.76

Table 5.6: A table showing the FIT classification results for both extraction methods on the test data set

Formation integrity test was extracted accurately with the project implementation method as shown by the high Matthews Correlation Coefficient score. The project implementation also reduced the false positive rate compared with the keyword indexing implementation as shown by the higher precision score. The false positives that remained with the project implementation are still valid FIT mentions but not calibration points as they do not have associated measurement value

shown in Figure 5.14. This issue can be solved by refining the pre-defined domain cases to look for associated measurements.

	CommentID	Comment
3960	2246	Pulled into shoe 1,181m, circulated and cycled MWD to record FIT data.

Figure 5.14: False Positive Example Comment for FIT

5.1.7 All Drilling Events

Overall the project implementation shows a reduction in false positives in comparison to the keyword indexing implementation especially when looking at tight hole events as shown in Figure 5.15.

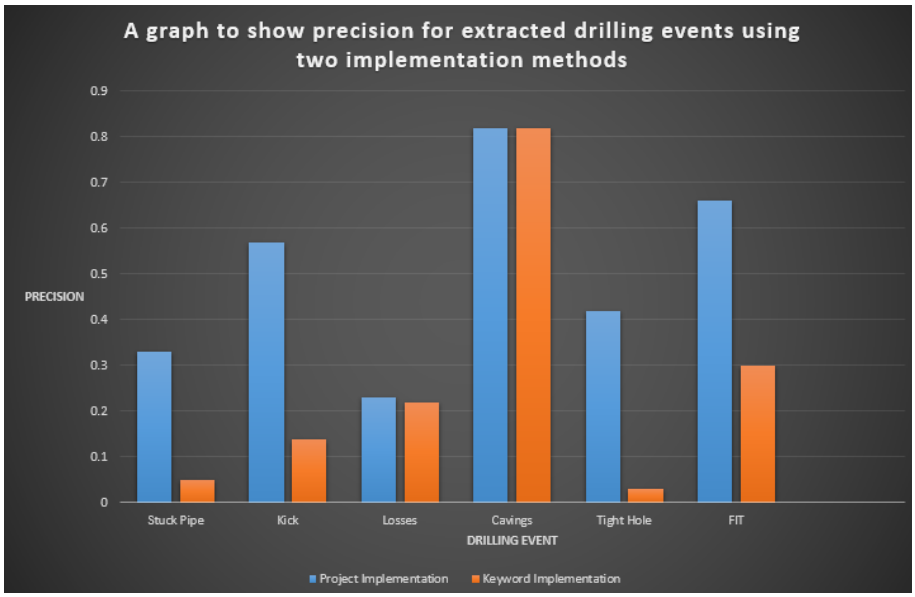


Figure 5.15: A graph to show drilling event precision for the project implementation and keyword index implementation

The overall evaluation accuracy as shown by the F1-score in Figure 5.16 is generally greater than 0.4 for all drilling events bar losses due to the mislabelling of the data set which show that losses extraction exceeds the manual benchmark. Cavings shows the best extraction results while tight hole needs refining in the pre-defined domain cases. Most of the false negatives that occur in the drilling event can be solved with better defined domain cases in combination with a full implementation of entity linking to reduce the surface mention ambiguity.

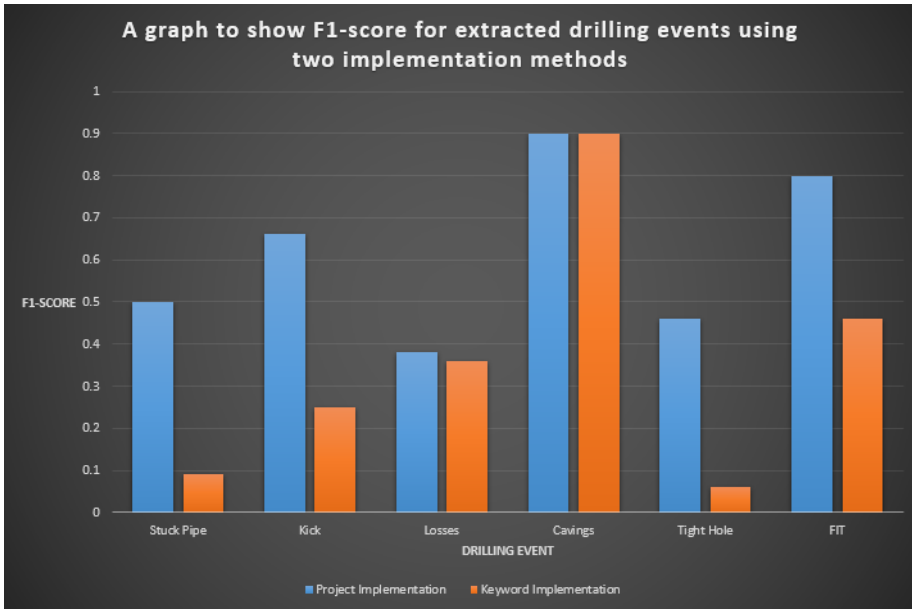


Figure 5.16: A graph to show drilling event F1-Score for the project implementation and keyword index implementation

The results show that the current project implementation is an improvement over the naive keyword indexing method mainly due to the scope to account for the false negatives and false positives in the current project implementation with the argumentation framework and entity linking. Furthermore, the project implementation exceeds the manual extraction benchmark especially in losses and cavings while falling behind in events such as kick. Badly performing events can be improved with better knowledge transfer to the domain inputs but manual labelling will always show mistakes especially with large data sets. Furthermore, manual labelling will never be done on very large data sets for all drilling events whereas the project implementation has shown that comparable results can be achieved to the manual benchmark and therefore, it is reliable to run on the very large data sets.

5.2 Coverage

The coverage of the application is a measure of how well the application generalises across different data sets. The current project implementation involves extracting features from the comments such as domain phrases which may occur in various forms in different data sets due to different writing styles or company standards. The variations leading to surface mention ambiguity can be solved with entity linking replacing the domain phrase features in future iterations. For the current implementation, coverage is important to assess in order to see if the application extraction methods have overfitted on the training data set through the domain defined features or the predefined cases for the argumentation framework. In order to get an estimate of the coverage, the application was run on two unlabelled data sets. The extracted drilling events were then labelled by a domain expert to evaluate the coverage of the application.

Given the heavily unbalanced nature of the data set regarding the proportion of drilling event comments to non-drilling event comments, only the extracted events were labelled by the domain expert. This means that only true positives and false positives and by extension precision could be evaluated. The number of extracted events from one coverage test data set is 92 comment events in a data set size of 32000 comments. Therefore, taking a small random sample of non-drilling event comments to check for false negatives and true negatives would be meaningless as there is a very low likelihood of randomly sampling a drilling event from the data set.

5.2.1 Schlumberger Version One Data Set

The Schlumberger version one data set was used as a test coverage data set. The data set contains around 32,000 comments and 92 extracted drilling events were evaluated by a domain expert. Table 5.7 shows the evaluation metrics for each drilling event.

Drilling Event	No. of Events	True Positive	False Positive	Precision
Cavings	9	9	0	1
FIT	14	13	1	0.92
Kick	20	7	13	0.35
Losses	21	19	2	0.90
Stuck Pipe	13	8	5	0.61
Tight Hole	15	11	4	0.73

Table 5.7: A table showing the coverage classification results for the Schlumberger Version One Data Set

The precision scores for the drilling event extraction were quite high for all events bar kick which yielded a lot of false positives in comparison to true positives. Upon analysis of the kick false positives, the current limitations with the approach were shown mainly in the underdeveloped pre-defined cases and the lack of a mature entity linking system.

The lack of a developed knowledge base for the entity linking system results in false positives due to a surface mention ambiguity for the domain phrases and overfitting to the training data set. One example for kick false positives is a surface mention variation for well static which signifies a negative outcome for a flow check test for kick events in a comment. The convention in the training data set was to specify negative outcomes for a flow check test with 'well static'. The convention in the test coverage data set was to specify 'well static' as 'well stable'.

Another set of false positives were due to underdeveloped pre-defined cases sets for the drilling event argumentation framework. One false positive for kick was when there was a 'gain' which should signify a kick event. However, the gain was due to air in the mud which is not a calibration point as gain is valid for a kick event only if it is from a reservoir fluid. While the instance of this scenario is quite rare, the argumentation case which would solve this doesn't have to be this explicit scenario. It can be a case that attacks the gain false positive argument by restricting it to only reservoir fluids rather than encoding the gain due to air feature case.

5.2.2 Schlumberger Version Two Data Set

The Schlumberger version two data set was also used as a test coverage data set. The data set contains around 290,358 comments and a random sample of 300 extracted drilling events were evaluated by a domain expert. Table 5.8 shows the evaluation metrics for each drilling event.

Drilling Event	No. of Events	True Positive	False Positive	Precision
Cavings	50	46	4	0.92
FIT	50	41	9	0.82
Kick	50	38	12	0.76
Losses	50	43	7	0.86
Stuck Pipe	50	35	15	0.7
Tight Hole	50	48	2	0.96

Table 5.8: A table showing the coverage classification results for the Schlumberger Version Two Data Set

The coverage results from this data set are quite good with all events having a precision greater than 0.7. Since there is a larger sample of drilling events to evaluate, the higher precision scores are more representative of the coverage of the project implementation than the Schlumberger Version One data set. Kick and Stuck pipe are still the worst performing drilling events relative to the other drilling events and thus will need more work put into the pre-defined domain cases.

5.2.3 All Data Sets

Figure 5.17 shows the precision scores per drilling event for the public Australian data set, the Schlumberger Version One data set and the Schlumberger Version Two data set using the current implementation method.

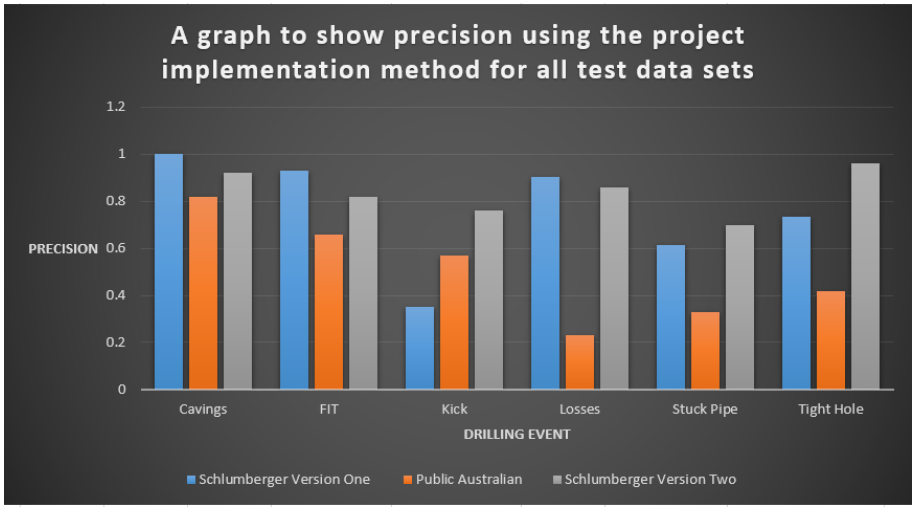


Figure 5.17: A graph to show drilling event precision for two data sets

All drilling events bar Kick perform consistently well between the two coverage data sets. The public Australian data set shows a major difference in losses and tight hole mainly due to some mislabelling as well as the low sample size for events. There is a slight drop in precision for Cavings, FIT and Losses from coverage test set one to two which shows that even for cases that perform well on the test accuracy data set, there will be generalisation issues when examined on larger data sets where there is more variance.

5.3 Performance

The automated solution will always be much faster than a manual approach as it takes roughly 20 mins to manually process a drilling report with 5 comments. However, the automated solution still needs to execute in a reasonable time when scaling up in order to be viable for the iterative process of improving the accuracy of the calibration point extraction process with new features and domain knowledge improvements. Furthermore, the end-user experience will be impacted by a long processing time especially on larger data sets exceeding 50,000 comments. Since both the entity linking and abstract argumentation frameworks were created from scratch, the performance of these two pipelines especially needs to be assessed. However, entity linking without a complete knowledge bank was as implemented as a proof of concept mainly to show how contextual information could be used to reduce surface mention ambiguity. Therefore, the performance of the entity linking pipeline will not be assessed as scalability issues will not occur with the current number of entities in the knowledge bank.

Scaling up the application for the abstract argumentation pipeline involves adding new cases and case features for the abstract argumentation framework to process. The initial implementation of the bulk processing abstract argumentation pipeline was simply chaining the initial proof of concept sub-processes such as argumentation framework generation, dispute tree generation and explanation generation. A single iteration for these unoptimised processes computed within seconds for a single query comment and a small pre-defined case feature set. However, testing on 1000 comments with the same pre-defined case feature set without any optimisations took around one hour which when scaled up to 50,000 comments for the training data set which would not be a reasonable computation time. A target was set to reduce the computation time of a 1000 comments to within one minute. The list below shows the successive optimisations implemented labelled by their optimisation number to achieve this target performance goal with the results of

the optimisation shown in Figure 5.18.

The performance analysis was done through the python standard library profiler, 'cProfile'. The results were analysed using another python standard library called 'pstats'. These two libraries enabled bottleneck analysis through profiling various performance statistics of the application such as cumulative time and the number of calls for functions.

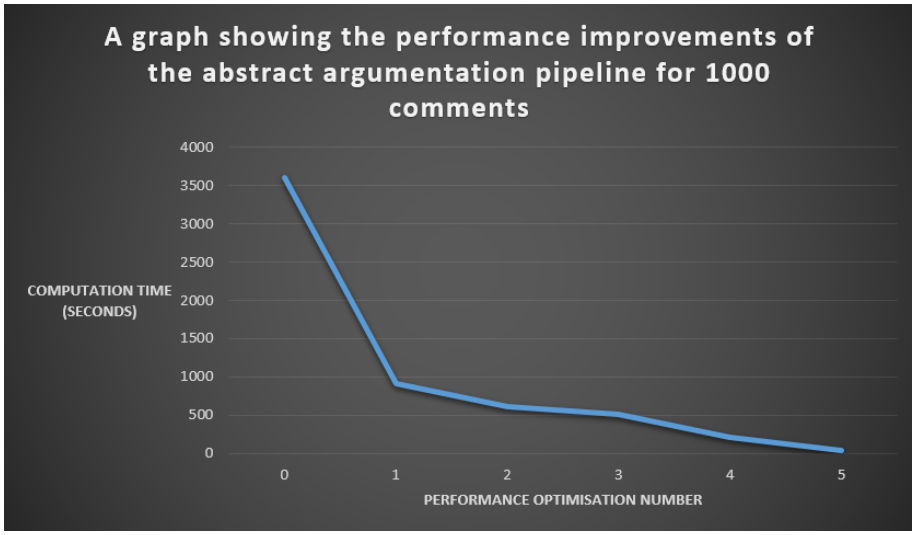


Figure 5.18: Performance Optimisation Graph for 1000 comments

- **Performance Optimisation One**
  The pre-defined argumentation framework for each new comment was computed every iteration unnecessarily as only the addition of the new case had to be computed each iteration so it was moved out of the comment iteration loop. The new comment features were also computed every iteration which created an unnecessary overhead in the new data frame construction when using the Pandas library. Therefore, the comment features for the set of query comments was pre-computed as a lookup table rather than computing the new comment features in the iteration loop. This optimisation reduced the computation time from around an hour to around 15 minutes with the new bottlenecks shown in Appendix C in Figure C.1.

- **Performance Optimisation Two**
  The initial implementation of the argumentation pipeline used a graph library called Networkx which was used to define a directed graph structure for the argumentation framework and dispute tree. However, the overhead in using the library was slowing down the iterations in the creation of the directed graphs in each iteration with the functions 'add nodes from' and 'add edges from' as well as the 'reverse' graph function used in the dispute tree and explanation generation. The library Networkx was useful in some calculations such as the longest child path for choosing a proponent attacker in the dispute tree generation. However, the representation as a Networkx directed graph was unnecessary and only having the nodes and edges passed through as parameters between the argumentation pipeline processes was sufficient and more optimal. Therefore, the Networkx implementation functions such as 'reverse' were removed from the argumentation graph sub-processes resulting in a performance improvement from 15 minutes to around 10 minutes with the new bottlenecks shown in Appendix C in Figure C.2.

- **Performance Optimisation Three**
  The next optimisation was to completely remove the Networkx integration from the dispute tree generation resulting in a performance improvement from around 10 minutes to 8.5 minutes with the new bottlenecks shown in Appendix C in Figure C.3.

- **Performance Optimisation Four**
  The generation of the explanations was relying on iterating through the cases in the different levels of the dispute tree and string concatenating a natural language translation of the

semantic triple from the case features in each node traversal. The string concat function was inefficient to generate the natural language translation as it was done inplace within the iteration for each case feature added to the explanation. Therefore, the explanation generation algorithm was refactored to remove the string concat function and make the traversal more optimal. Furthermore, iterating through a Pandas dataframe with itertuples and iterrows is inefficient. These loops were replaced with vectorised operations in optimised C code by using Numpy vectorisation. These optimisations resulted in a performance improvement from around 8.5 minutes to 3.4 minutes with the new bottlenecks shown in Appendix C in Figure C.4.

- **Performance Optimisation Five**
  At this point having analysed the performance of the application using the profiler, unseen bugs in the argumentation implementation were found such as a bug which resulted in the list of edges linearly increasing with each iteration. Fixing these bugs resulted in a performance improvement from around 3.4 minutes to around 40 seconds which meets the goal of reducing the argumentation processing time to less than a minute for 1000 comments. The final performance analysis for 1000 comments is shown in Appendix C in Figure C.5.

While the performance for a 1000 comments has been optimised to a reasonable time, scaling up to 50000 comments was still an issue with regards to memory problems. The current methodology relies on loading case features for all new comments in the query data set and using the resulting new case feature table as a lookup table. However, this speed optimisation results in increased memory usage. Therefore, batch processing on the query data set was implemented to reduce memory usage while also improving the computation time for larger data set sizes. Table 5.9 shows the results of different experiments run to find the optimal batch size with regards to computation time for different data set sizes.

Experiment	**Data Set Size (No. of Comments)**	**Batch Size (No. of Comments)**	**Computation Time (Seconds)**
1	1000	100	142.883
2	1000	1000	40.640
3	10000	1000	351.536
4	10000	10000	258.393
5	57270	10000	1532.889
6	57270	5000	1672.855
7	57270	57270	1820.388

Table 5.9: A table to show the results of the batch size experiments

From the experiments, it is clear that batch processing only has a positive performance gain in terms of computation time at data set sizes greater than 10,000. The optimal batch size for processing the whole training data set is shown in experiment five with a batch size of 10,000.

Chapter 6

Conclusion

This project demonstrates a complete automated workflow from taking in the raw comment data sets to producing a calibration point output file using traditional data mining and natural language processing techniques.

An extract transform load pipeline has been implemented in the data extraction module to extract the raw comments, pre-process them into a clean data set and load them into the SQLite3 Database. Exploratory data analysis was conducted on the Schlumberger training data set in order to understand which features to extract for calibration point extraction as well as any hidden correlations between drilling events and the explored features.

The feature extraction module contains statistical features that have been hand-engineered based on the exploratory data analysis to increase the accuracy of the extraction of calibration points. Statistical features include part of speech tags, noun phrases and domain keywords. A syntactic parser has also been implemented to extract measurement data from the comments for the measurement aspect of the calibration points.

A proof of concept implementation for entity linking has been developed to provide a way to reduce surface mention ambiguity as a way of reducing false positives. The viability of applying abstract argumentation with case-based reasoning as a feature analysis method to evaluate the drilling events aspect of calibration points has also been demonstrated.

Finally, the calibration points were aggregated from the upstream pipelines to generate the calibration point file in the data insights module. A calibration evaluation pipeline has also been implemented in the data insights module to assess the success of this project in terms of accuracy, coverage and performance which have been largely achieved.

6.1 Key Achievements

The main achievement of this project was to create a framework for data mining which justifies the outputs of the mining process through generated explanations.

The framework has been implemented for the extraction of calibration points through three stages: extracting features from free text, mapping normalised features using semantic triples to cases and formulating an explanation for a data insight using abstract argumentation for case-based reasoning. While the focus of this implementation is calibration points, the framework can be utilised for any data insight in textual data. Furthermore, the framework allows the user to not only extract calibration points from comments more accurately as shown in the evaluation results but also to understand why the calibration points were extracted. Explain-ability is an important factor in presenting the output to the user and this framework provides a basis to formulate thorough explanations.

This project has novel contributions in demonstrating a practical implementation of machine arguing research in a new domain. The problem with abstract argumentation with case-based reasoning

is finding a mapping from extracted variable features in the text to fixed predefined cases to formulate an explanation which is understandable. The aforementioned second stage in the framework provides the missing link between extracted feature tables and the abstract argumentation framework in a way that allows the relations to be clearly reasoned about in the framework. The second stage is the feature analysis module consisting of the entity linking pipeline and the feature to case (semantic triples) translations in the argumentation pipeline. The use of semantic triples in the web resource description framework as the format for the cases allows relations to be easily defined and reasoned with both across and within cases. The semantic triple also translates nicely into natural language for explanation generation. The implementation of entity linking not as a form of data enrichment but as a way to reduce the variability of surface mentions is key to the success of the framework when generalising the domain inputs to the application across different data sets.

In terms of the success of the framework implementation with regard to calibration point extraction, the results discussed in the accuracy evaluation show that across all drilling events there have been improvements in precision, recall and f1-score when compared against the keyword indexing implementation. In terms of meeting the gold standard of the manual benchmark, the number of false positives was reduced to an acceptable level for manual refinement for all events. The project implementation improved upon the results of some of the drilling events such as losses and cavings in comparison to the manual benchmark by capturing missed drilling events.

However, there is still improvement to be made regarding tight hole extraction due to the low precision score. Coverage scores were quite positive in terms of precision for all events bar kick events. The performance of the project implementation for abstract argumentation with case-based reasoning was also considerably improved from hours to less than a minute for a batch of 1000 comments. This performance improvement helped achieve the goal of implementing a practical algorithm for the argumentation pipeline especially when considering that the whole pipeline was implemented from scratch. Running the application on the Schlumberger Version Two Data Set which had around 300 thousand comments was completed within a few hours and generated 18786 drilling events which constitute around 6.46% of the data set. Without this application, there is no possibility of manually parsing all 300 thousand comments for the six drilling events and the coverage results for this data set show that the proportion of actual drilling events is quite high within the subset of comments extracted.

6.2 Limitations and Future Research

The three-stage framework established the groundwork for this project which many improvements can be developed on. The main bottleneck in improving the accuracy and coverage of calibration point extraction was to establish an efficient complete application in order to be able to experiment with different methods and parameters. Now that the application workflow has been completed more experiments can be conducted in each of the pipelines to tackle the limitations of the current project implementation. There are two current limitations that can be solved by developing the three-stage framework which are the incomplete predefined cases and proof of concept entity linking implementation. Furthermore, neglected modules that were not the focus of research can also be developed with the MVP now in place.

6.2.1 Structured Data Extraction

The scope of this project was limited to working with structured CSV file inputs for daily drilling report comments. However, for future research, the structured data extraction modules could be developed to extract tabular data from PDF documents to complete the high-level workflow. Most of the daily drilling reports will be in a PDF format with free text comments in tables so a possible solution would be to use a custom computer vision method to identify tabular structures within the PDF document in combination with an open-source OCR library to extract the free-text comments. There do exist various open-source python libraries that do tabular extraction directly from PDF but these solutions are not robust as they sometimes miss tables in the document and also have errors in extracting complete words over line breaks in one instance.

6.2.2 Domain Knowledge Improvements

The success of the abstract argumentation framework relies on the completeness of the predefined cases for each drilling event. From the coverage results, there are still general cases that can be captured in the predefined domain cases. Improvements to the domain knowledge inputs including the predefined cases, in general, were difficult due to communication issues with the domain expert created by remote working as a result of the Covid-19 pandemic during the course of the project. While the general cases can be tackled through more complete domain cases, there may exist unique edge cases for drilling events per data set that will be difficult to account for with only textual information contained in the comment. To solve this in future iterations, the inputs to the three-stage framework can be augmented with additional data from the drilling operation. This additional support evidence can be analysed to extract features from, in order to support the cases arguing for a drilling event occurrence in future iterations.

6.2.3 Supervised Feature Improvements

Better features can be explored in the feature extraction modules as a way to generate better cases. The general supervised machine learning problem area of this project is extreme event detection. Some supervised features such as a feature table of drilling event predictions from a model trained using sequence classification with BERT were experimented with during the course of the project. However, due to time limitations, they were not fully implemented and during the experimentation, it was found that the training data set was mislabelled which significantly reduced the potential accuracy of a supervised model. In future iterations with a better-labelled data set, the supervised feature tables can be explored. Unsupervised learning methods to extract such as topic modelling were also experimented with but not implemented in the framework due to the extreme event nature of drilling event occurrence in the data sets.

6.2.4 Entity Linking Improvements

The current implementation relies on domain phrases as a key case feature in the predefined cases. However, as the coverage results have shown, surface mention ambiguity for domain phrases leads to false positives and false negatives. Therefore, a significant improvement would be fleshing out the entity linking pipeline especially in the development of the knowledge base. Knowledge bases do exist for general entity linking such as DBpedia but a domain-specific knowledge base is necessary as the comments contain terms which are heavily domain-specific [SWH15]. A web parser which trawls through Petro Wikipedia articles would populate the knowledge base with enough entities to increase the extraction accuracy once the licensing issues are solved [Soc20]. In this improvement, the domain phrase feature would be phased out and replaced by the normalised linked entity equivalent of the domain phrases removing surface mention ambiguity.

The algorithms within entity linking can also be improved especially in candidate entity ranking where a graph-based algorithm which enables collective ranking would result in better candidate entity ranking scores as it would take into account contextual information in the comment and with the other candidate entities.

6.2.5 Machine Arguing Framework Improvements

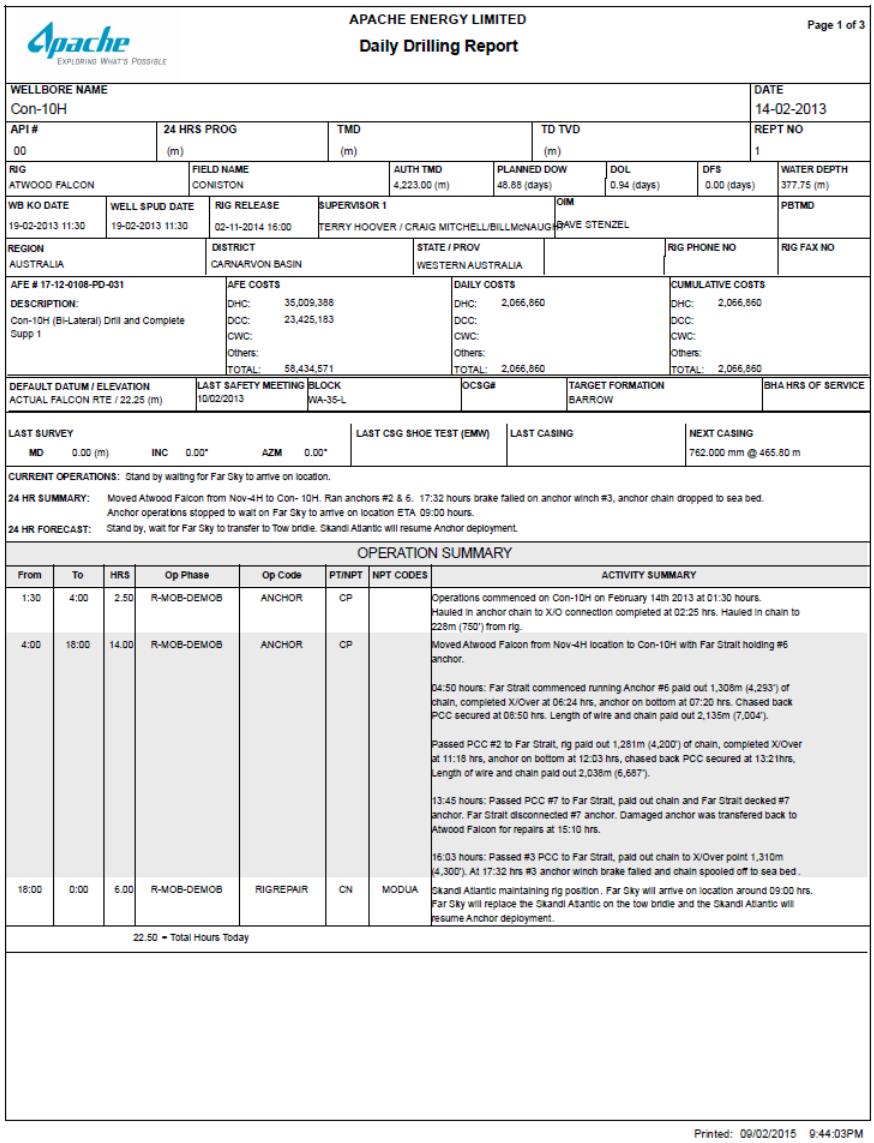
The machine arguing framework is currently limited to an abstract argumentation for the initial implementation. There are extensions that can be added to the abstract argumentation framework such as different definitions of attacks. Currently, an attack is a simple binary relation between two cases but it can be augmented to have weights based on the case feature scores within the two cases. Another extension could be to add preferences to certain sets of cases based on the sources of the case features such as objects in the semantic triple or the contextual level of the case feature. Preferences allow prioritising cases or rejecting an attack if a lower preference level case attacks a higher preference level case.

6.2.6 User Experience Improvements

The final set of improvements is to improve the usability of the application. Currently, the application is implemented as a series of library modules that are exposed via several module class APIs. The APIs are called by a Juptyer notebook with the general experiment parameters such as data set and argument set through a configuration file. The architectural design of the application was done in order to be able to refactor the code with minimal effort into a web application where the python library modules can be put in a Flask back end. A web application can be created which ingests the data set specific aspects of the project such as the raw data set files and data set labels. The value of the user interface would be in creating a page which allows cases to defined for arguments in a more intuitive way than defining case features lines in a CSV file. This web application would also allow the data insights to be visualised in a graphical format. A graph for each well location of depth and mud weight could be plotted with the data points being calibration points.

Appendix A

Daily Drilling Reports



APACHE ENERGY LIMITED

Daily Drilling Report

Page 1 of 3

WELLBORE NAME					DATE
Con-10H					14-02-2013
API #	24 HRS PROG	TMD	TD TVD		REPT NO
00	(m)	(m)	(m)		1

RIG	FIELD NAME	AUTH TMD	PLANNED DOW	DOL	DFS	WATER DEPTH
ATWOOD FALCON	CONISTON	4,223.00 (m)	48.88 (days)	0.94 (days)	0.00 (days)	377.75 (m)

WB KO DATE	WELL SPUD DATE	RIG RELEASE	SUPERVISOR 1	OIM	PBTMD
19-02-2013 11:30	19-02-2013 11:30	02-11-2014 16:00	TERRY HOOVER / CRAIG MITCHELL/BILLMcNAUGH	DAVE STENZEL	

REGION	DISTRICT	STATE / PROV	RIG PHONE NO	RIG FAX NO
AUSTRALIA	CARNARVON BASIN	WESTERN AUSTRALIA		

AFE # 17-12-0108-PD-031	AFE COSTS	DAILY COSTS	CUMULATIVE COSTS
DESCRIPTION:	DHC: 35,009,388	DHC: 2,066,860	DHC: 2,066,860
Con-10H (Bi-Lateral) Drill and Complete Supp 1	DCC: 23,425,183	DCC:	DCC:
	CWC:	CWC:	CWC:
	Others:	Others:	Others:
	TOTAL: 58,434,571	TOTAL: 2,066,860	TOTAL: 2,066,860

DEFAULT DATUM / ELEVATION	LAST SAFETY MEETING	BLOCK	OCSG#	TARGET FORMATION	BHA HRS OF SERVICE
ACTUAL FALCON RTE / 22.25 (m)	10/02/2013	WA-35-L		BARROW	

LAST SURVEY	LAST CSG SHOE TEST (EMW)	LAST CASING	NEXT CASING
MD 0.00 (m) INC 0.00° AZM 0.00°			762.000 mm @ 465.80 m

CURRENT OPERATIONS: Stand by waiting for Far Sky to arrive on location.

24 HR SUMMARY: Moved Atwood Falcon from Nov-4H to Con- 10H. Ran anchors #2 & 6. 17:32 hours brake failed on anchor winch #3, anchor chain dropped to sea bed. Anchor operations stopped to wait on Far Sky to arrive on location ETA 09:00 hours.

24 HR FORECAST: Stand by, wait for Far Sky to transfer to Tow bridle. Skandi Atlantic will resume Anchor deployment.

OPERATION SUMMARY

From	To	HRS	Op Phase	Op Code	PT/NPT	NPT CODES	ACTIVITY SUMMARY
1:30	4:00	2.50	R-MOB-DEMOB	ANCHOR	CP		Operations commenced on Con-10H on February 14th 2013 at 01:30 hours. Hauled in anchor chain to X/O connection completed at 02:25 hrs. Hauled in chain to 228m (750') from rig.
4:00	18:00	14.00	R-MOB-DEMOB	ANCHOR	CP		Moved Atwood Falcon from Nov-4H location to Con-10H with Far Strait holding #6 anchor. 04:50 hours: Far Strait commenced running Anchor #6 paid out 1,308m (4,293') of chain, completed X/Over at 06:24 hrs, anchor on bottom at 07:20 hrs. Chased back PCC secured at 08:50 hrs. Length of wire and chain paid out 2,135m (7,004'). Passed PCC #2 to Far Strait, rig paid out 1,281m (4,200') of chain, completed X/Over at 11:18 hrs, anchor on bottom at 12:03 hrs, chased back PCC secured at 13:21hrs, Length of wire and chain paid out 2,038m (6,687'). 13:45 hours: Passed PCC #7 to Far Strait, paid out chain and Far Strait decked #7 anchor. Far Strait disconnected #7 anchor. Damaged anchor was transfered back to Atwood Falcon for repairs at 15:10 hrs. 16:03 hours: Passed #3 PCC to Far Strait, paid out chain to X/Over point 1,310m (4,300'). At 17:32 hrs #3 anchor winch brake failed and chain spooled off to sea bed.
18:00	0:00	6.00	R-MOB-DEMOB	RIGREPAIR	CN	MODUA	Skandi Atlantic maintaining rig position. Far Sky will arrive on location around 09:00 hrs. Far Sky will replace the Skandi Atlantic on the tow bridle and the Skandi Atlantic will resume Anchor deployment.

22.50 = Total Hours Today

Printed: 09/02/2015 9:44:03PM

Figure A.1: Public Australian dataset sample

Scientific Drilling

ORMAT
Carson Field

Drilling Run Report for Forge 21-31 BHA #5

Drillstring Components (12 1/4 BHA #5)

Item #	Description	Vendor	Serial No.	Outside Diameter (in)	Inside Diameter (in)	Fishing Neck OD (in)	Fishing Neck Length (ft)	Top Connection	Length (ft)	Total Length (ft)
14	3 x 6 1/2 Steel DC's	Paul Graham	N/A	6 1/2	2 3/4	0	0.00	4 1/2 XH	91.34	355.64
15	20 x 4 1/2" HWDP	Paul Graham	N/A	5	2 13/16	0	0.00	4 1/2 XH	619.32	974.96

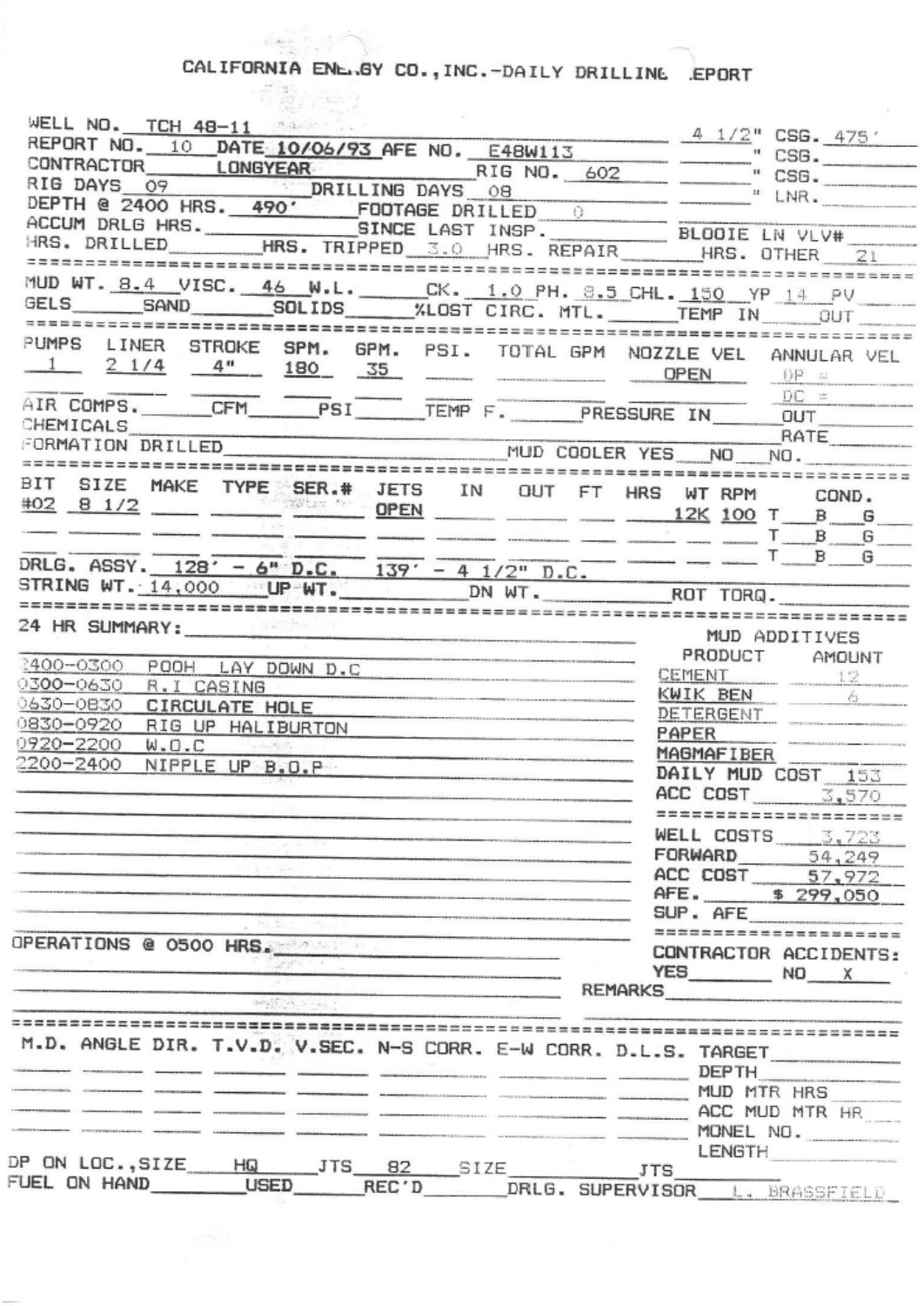
Mud Properties

Date	Time	Bit Depth (ft)	Mud Weight (ppg)	Yield Point (lbf/100ft²)	Plastic Viscosity (cp)	Funnel Viscosity (Secs)	Chlorides (ppm)	pH	Oil (%)	Gas (%)	Sand (%)	Solids (%)	Water Line (ft)	Bot.Hole Temp. (°F)	Flow Temp. (°F)
20 Feb 2018	09:30	4,380.00	9.2	19.0	14.00	45	1000.00	10.30	0.00	0.00	0.25	5.00	0.00	0.0	0.0
21 Feb 2018	00:00	4,480.00	9.1	10.0	14.00	40	1100.00	10.00	0.00	0.00	0.25	4.00	0.00	156.0	148.0

Activity Breakdown

Date	Start Time	End Time	Hours	Start Depth (ft)	End Depth (ft)	Course Length (ft)	Rate of Penetr'n (ft/hr)	Activity	Comment
20 Feb 2018	00:00	02:30	2.50	4,380.00	4,380.00			Change BHA	Pick up and scribe motor, wait on new Bit choice
20 Feb 2018	02:30	04:30	2.00	4,380.00	4,380.00			Rig Repair	Accumulator & BOP stack frozen
20 Feb 2018	04:30	05:00	0.50	4,380.00	4,380.00			Change BHA	Pick up non mag drill collars out of the derrick
20 Feb 2018	05:00	05:45	0.75	4,380.00	4,380.00			Change BHA	Scribe collar, Heat collar and orient MWD tools to scribe
20 Feb 2018	05:45	06:00	0.25	4,380.00	4,380.00			Change BHA	Pick up shock sub
20 Feb 2018	06:00	08:15	2.25	4,380.00	4,380.00			Tripping In	Trip in hole with remaining BHA #5 (DC's, Jars, HWDP) to the shoe
20 Feb 2018	08:15	09:30	1.25	4,380.00	4,380.00			Other	Mud pumps froze up. Thaw mud pumps
20 Feb 2018	09:30	10:00	0.50	4,380.00	4,380.00			Circulate and/or Condition Mud	Fill pipe, circulate and test MWD. No pressure on transducer. Check transducer for obstruction
20 Feb 2018	10:00	10:15	0.25	4,380.00	4,380.00			Circulate and/or Condition Mud	Change transducer, test MWD, Good test
20 Feb 2018	10:15	14:00	3.75	4,380.00	4,380.00			Circulate and/or Condition Mud	Circulate with pump #1 while working on pump #2
20 Feb 2018	14:00	15:05	1.08	4,380.00	4,380.00			Tripping In	Trip in hole to 3000` MD. Fill pipe.
20 Feb 2018	15:05	17:00	1.92	4,380.00	4,380.00			Circulate and/or Condition Mud	Circulate with 2 pumps, test MWD, check temp. MWD temp = 111° F
20 Feb 2018	17:00	17:30	0.50	4,380.00	4,380.00			Tripping In	Trip in hole to 4320` MD
20 Feb 2018	17:30	18:00	0.50	4,380.00	4,380.00			Reaming	Ream last two joints from 4320` to 4380`
20 Feb 2018	18:00	18:18	0.30	4,380.00	4,387.00	7.00	23.33	Rotating	
20 Feb 2018	18:18	18:28	0.17	4,387.00	4,387.00			Survey & Connection	MWD temp = 136° F
20 Feb 2018	18:28	00:00	5.53	4,387.00	4,460.00	73.00	13.19	Rotating	
21 Feb 2018	00:00	01:10	1.17	4,460.00	4,480.00	20.00	17.14	Rotating	MWD temp = 156° F
21 Feb 2018	01:10	01:20	0.17	4,480.00	4,480.00			Survey & Connection	Fix mud cooler
21 Feb 2018	01:20	04:00	2.67	4,480.00	4,540.00	60.00	22.50	Rotating	MWD temp = 138° F

Figure A.2: Public Forge dataset sample



CALIFORNIA ENERGY CO.,INC.-DAILY DRILLING REPORT

WELL NO. TCH 48-11 — 4 1/2" CSG. 475'
REPORT NO. 10 DATE 10/06/93 AFE NO. E48W113 — " CSG.
CONTRACTOR LONGYEAR RIG NO. 602 — " CSG.
RIG DAYS 09 DRILLING DAYS 08 — " LNR.
DEPTH @ 2400 HRS. 490' FOOTAGE DRILLED 0
ACCUM DRLG HRS. SINCE LAST INSP. BLOOIE LN VLV#
HRS. DRILLED HRS. TRIPPED 3.0 HRS. REPAIR HRS. OTHER 21

MUD WT. 8.4 VISC. 46 W.L. CK. 1.0 PH. 8.5 CHL. 150 YP 14 PV
GELS SAND SOLIDS %LOST CIRC. MTL. TEMP IN OUT

PUMPS	LINER	STROKE	SPM.	GPM.	PSI.	TOTAL GPM	NOZZLE VEL	ANNULAR VEL
1	2 1/4	4"	180	35			OPEN	DP =
								DC =

AIR COMPS. CFM PSI TEMP F. PRESSURE IN OUT
CHEMICALS RATE
FORMATION DRILLED MUD COOLER YES NO NO.

BIT	SIZE	MAKE	TYPE	SER.#	JETS	IN	OUT	FT	HRS	WT	RPM	COND.
#02	8 1/2				OPEN					12K	100	T B G
												T B G
												T B G

DRLG. ASSY. 128' - 6" D.C. 139' - 4 1/2" D.C.
STRING WT. 14,000 UP WT. DN WT. ROT TORQ.

24 HR SUMMARY:

2400-0300 POOH LAY DOWN D.C
0300-0630 R.I CASING
0630-0830 CIRCULATE HOLE
0830-0920 RIG UP HALIBURTON
0920-2200 W.O.C
2200-2400 NIPPLE UP B.O.P

OPERATIONS @ 0500 HRS.

MUD ADDITIVES

PRODUCT	AMOUNT
CEMENT	12
KWIK BEN	6
DETERGENT	
PAPER	
MAGMAFIBER	

DAILY MUD COST 153
ACC COST 3,570

WELL COSTS 3,723
FORWARD 54,249
ACC COST 57,972
AFE. $ 299,050
SUP. AFE

CONTRACTOR ACCIDENTS:
YES NO X
REMARKS

M.D.	ANGLE	DIR.	T.V.D.	V.SEC.	N-S CORR.	E-W CORR.	D.L.S.

TARGET
DEPTH
MUD MTR HRS
ACC MUD MTR HR
MONEL NO.
LENGTH

DP ON LOC.,SIZE HQ JTS 82 SIZE JTS
FUEL ON HAND USED REC'D DRLG. SUPERVISOR L. BRASSFIELD

Figure A.3: Public Historical dataset sample

Appendix B

Daily Drilling Reports Comments

NPT: REAMED DOWN FROM 2405 M TO 2491 M .FLOW RATE: 1500 LPM ï¼ SPP : 690 PSI ï¼ 120 RPMRECORDED SCR @ 2491 M FOR PUMP 2 & 3 :SPM SPP(PSI)----- ----------30 24040 30050 380

Figure B.1: Training Data Set Comment Sample One

Drilled 12-1/4¡± hole from 3940 ft to 4817 ft Sliding Mode:- Flow Rate = 800 GPM *SPP(On/Off) = 2500/2000 psi. - WOB = 20-35 klbs. Rotary Mode:- Flow Rate = 800 GPM* SPP = 2700psi. - WOB = 30-45 klbs. - RPM = 40+203 (Sur+Motor);TQ= 4.5-10 Klb-ft* Avg on bottom Sliding ROP = 21.7 ft/hr. * Avg on bottom Rotating ROP = 54.3 ft/hr. * Avg overall ROP = 36.6ft/hr- MWin = 68PCF* MWout = 68PCFSliding interval:4280'-4295'ï¼ 4363'-4390'ï¼ 4431'-4441'ï¼ 4460'-4480'ï¼ 4520'-4535'ï¼ 4747'-4764'.Sliding percentage: 12%.Av. ROP (From 3144)= 33.62ft/hr. - Last survey @ 4666ft: inc=40.7degï¼ Azi=69.07degï¼ TVD=4509.86'- Actual well path: 2.4FT above & 12.6FT right of plan.- Hole static- Top Hith formation @ 4162'TVD-SCR took @ 4538'ï¼ MW: 68pcfP1 : 30spm 320psi / 50spm 420psiP3 : 30spm 300psi / 50spm 400psConducted fire drill and fire pump test @ 15:00. Good response.

Figure B.2: Training Data Set Comment Sample Two

With ROV observing bullseye, slacked -off conductor string weight in 13.6MT (30klbs) increments. No slump or bullseye movement observed. Backed out CART and recovered CART complete with inner string to moonpool (flushed through string with wiper ball with stinger above wellhead). GE hand removed the 4 x lock down bolts from CART. Recovered CART through rotary table. Inspected seals - all good. Racked back CART assembly and inner string in the derrick . Neptune completed final survey & issued final location report (summarized below): Easting 196,445.84m Northing 7,636,346.83m 1.37m on a bearing of 32.9° True from target location. PGB orientation: 321.9° Grid LP housing elevation above virgin seabed = 2.28m Commenced kedging rig at 10:30hrs. Operations transferred to Coniston-12H.

Figure B.3: Public Australian Data Set Comment Sample One

POOH with 216mm (8 1/2") BHA on 127mm (5â€Vam EIS drill pipe from 4,059m to 3,776m at 4.0 min/std. Average P/Up weight 113.4MT (250klbs), max overpull observed 4.5-9.1MT (10-20klbs). Hole condition good.

Figure B.4: Public Australian Data Set Comment Sample Two

Test motor. Motor passed the test, tested at 300 GPM showing 350 PSI.

Figure B.5: Schlumberger Version One Data Set Comment Sample One

Drilled 12-1/4¡± hole from 3940 ft to 4817 ft Sliding Mode:- Flow Rate = 800 GPM *SPP(On/Off) = 2500/2000 psi. - WOB = 20-35 klbs. Rotary Mode:- Flow Rate = 800 GPM* SPP = 2700psi. - WOB = 30-45 klbs. - RPM = 40+203 (Sur+Motor);TQ= 4.5-10 Klb-ft* Avg on bottom Sliding ROP = 21.7 ft/hr. * Avg on bottom Rotating ROP = 54.3 ft/hr. * Avg overall ROP = 36.6ft/hr- MWin = 68PCF* MWout = 68PCFSliding interval:4280'-4295'ï¼ 4363'-4390'ï¼ 4431'-4441'ï¼ 4460'-4480'ï¼ 4520'-4535'ï¼ 4747'-4764'.Sliding percentage: 12%.Av. ROP (From 3144)= 33.62ft/hr. - Last survey @ 4666ft: inc=40.7degï¼ Azi=69.07degï¼ TVD=4509.86'- Actual well path: 2.4FT above & 12.6FT right of plan.- Hole static- Top Hith formation @ 4162'TVD-SCR took @ 4538'ï¼ MW: 68pcfP1 : 30spm 320psi / 50spm 420psiP3 : 30spm 300psi / 50spm 400psConducted fire drill and fire pump test @ 15:00. Good response.

Figure B.6: Schlumberger Version One Data Set Comment Sample Two

Appendix C

Performance Optimisations

```
Mon May 25 10:19:08 2020    pstats

         1339567890 function calls (1305284510 primitive calls) in 910.206 seconds

   Ordered by: cumulative time, function name
   List reduced from 1477 to 20 due to restriction <20>

   ncalls  tottime  percall  cumtime  percall filename:lineno(function)
  87003/1   42.174    0.000  912.654  912.654 {built-in method builtins.exec}
        1    0.171    0.171  912.654  912.654 argumentationAPI.py:1348(post_argumention_explanations)
        1    0.790    0.790  912.435  912.435 argumentationAPI.py:894(__generate_dataframe_explanations)
     1000    4.615    0.005  331.426    0.331 argumentationAPI.py:811(__generate_explanation)
     1000    0.011    0.000  280.765    0.281 argumentationAPI.py:736(__compute_dispute_tree)
88000/1000    3.011    0.000  280.584    0.281 argumentationAPI.py:755(recurse_child)
    94988    0.917    0.000  277.170    0.003 digraph.py:1181(reverse)
    87002    1.625    0.000  195.638    0.002 frame.py:849(itertuples)
     1000    3.668    0.004  190.007    0.190 argumentationAPI.py:281(__add_new_cases_arg_graph)
1031166/1027889    3.140    0.000  177.568    0.000 indexing.py:1485(__getitem__)
55792156/31745204   65.809    0.000  150.904    0.000 copy.py:132(deepcopy)
   549668    3.934    0.000  147.772    0.000 frame.py:2893(__getitem__)
    96988   34.454    0.000  141.693    0.001 digraph.py:643(add_edges_from)
    95989   28.274    0.000  133.175    0.001 digraph.py:428(add_nodes_from)
   214213    1.243    0.000  127.089    0.001 frame.py:2952(_getitem_bool_array)
  1024565    2.793    0.000  119.097    0.000 indexing.py:2205(_getitem_axis)
   217582    2.284    0.000  110.905    0.001 generic.py:3323(_take)
1383213/1382213    7.266    0.000  110.610    0.000 series.py:152(__init__)
   110005    0.299    0.000  107.790    0.001 {method 'extend' of 'list' objects}
   869016    1.637    0.000  107.491    0.000 frame.py:919(<genexpr>)
```

Figure C.1: Performance Optimisation pStats output for Optimisation One with 1000 comments

```
Tue May 26 11:07:12 2020    pstats

         835548057 function calls (828337574 primitive calls) in 603.707 seconds

   Ordered by: cumulative time, function name
   List reduced from 1433 to 20 due to restriction <20>

   ncalls  tottime  percall  cumtime  percall filename:lineno(function)
  87003/1   38.154    0.000  605.880  605.880 {built-in method builtins.exec}
        1    0.179    0.179  605.879  605.879 argumentationAPI.py:1239(post_argumention_explanations)
        1    6.040    6.040  605.649  605.649 argumentationAPI.py:785(__generate_dataframe_explanations)
     1000    4.142    0.004  302.480    0.302 argumentationAPI.py:693(__generate_explanation)
    87002    1.539    0.000  179.854    0.002 frame.py:849(itertuples)
906166/902889    2.606    0.000  138.710    0.000 indexing.py:1485(__getitem__)
     1000    0.908    0.001  124.163    0.124 argumentationAPI.py:618(__compute_dispute_tree)
     2000   84.650    0.042  108.666    0.054 digraph.py:643(add_edges_from)
111735/111729    0.284    0.000  100.461    0.001 {method 'extend' of 'list' objects}
   869016    1.546    0.000  100.177    0.000 frame.py:919(<genexpr>)
   782018    0.804    0.000   91.292    0.000 indexing.py:2141(_getitem_tuple)
   899565    2.184    0.000   84.395    0.000 indexing.py:2205(_getitem_axis)
   301668    2.111    0.000   70.366    0.000 frame.py:2893(__getitem__)
   782018    5.790    0.000   69.949    0.000 indexing.py:960(_getitem_lowerdim)
        1    3.329    3.329   68.643   68.643 argumentationAPI.py:969(__generate_argument_cases)
     1000   51.219    0.051   68.198    0.068 argumentationAPI.py:545(__compute_grounded_extension)
    89213    0.519    0.000   56.446    0.001 frame.py:2952(_getitem_bool_array)
    92582    0.921    0.000   51.525    0.001 generic.py:3323(_take)
    87002    3.054    0.000   50.933    0.001 __init__.py:357(namedtuple)
 78010507   21.579    0.000   49.634    0.000 {built-in method builtins.isinstance}
```

Figure C.2: Performance Optimisation pStats output for Optimisation Two with 1000 comments

```
Tue May 26 12:24:29 2020    pstats

         732506331 function calls (723794863 primitive calls) in 511.610 seconds

   Ordered by: cumulative time, function name
   List reduced from 1417 to 20 due to restriction <20>

   ncalls  tottime  percall  cumtime  percall filename:lineno(function)
  87003/1   35.189    0.000  513.650  513.650 {built-in method builtins.exec}
        1    0.172    0.172  513.650  513.650 argumentationAPI.py:1257(post_argumention_explanations)
        1    5.132    5.132  513.428  513.428 argumentationAPI.py:803(__generate_dataframe_explanations)
     1000    3.854    0.004  286.845    0.287 argumentationAPI.py:711(__generate_explanation)
    87002    1.557    0.000  171.049    0.002 frame.py:849(itertuples)
906166/902889    2.458    0.000  132.632    0.000 indexing.py:1485(__getitem__)
   111453    0.260    0.000   96.201    0.001 {method 'extend' of 'list' objects}
   869016    1.425    0.000   95.942    0.000 frame.py:919(<genexpr>)
   782018    0.767    0.000   87.535    0.000 indexing.py:2141(_getitem_tuple)
   899565    2.094    0.000   80.684    0.000 indexing.py:2205(_getitem_axis)
   782018    5.577    0.000   67.164    0.000 indexing.py:960(_getitem_lowerdim)
   301668    1.910    0.000   66.189    0.000 frame.py:2893(__getitem__)
     1000   48.809    0.049   65.460    0.065 argumentationAPI.py:545(__compute_grounded_extension)
        1    3.181    3.181   65.281   65.281 argumentationAPI.py:987(__generate_argument_cases)
     1000   17.504    0.018   55.911    0.056 argumentationAPI.py:618(__compute_dispute_tree)
    89213    0.465    0.000   53.075    0.001 frame.py:2952(_getitem_bool_array)
    92582    0.847    0.000   48.602    0.001 generic.py:3323(_take)
 78001389   21.001    0.000   47.961    0.000 {built-in method builtins.isinstance}
    87002    2.843    0.000   47.063    0.001 __init__.py:357(namedtuple)
    92582    0.963    0.000   42.783    0.000 managers.py:1329(take)
```

Figure C.3: Performance Optimisation pStats output for Optimisation Three with 1000 comments

```
Tue May 26 15:03:58 2020    pstats

         390630201 function calls (388042977 primitive calls) in 205.919 seconds

   Ordered by: cumulative time, function name
   List reduced from 1391 to 20 due to restriction <20>

   ncalls  tottime  percall  cumtime  percall filename:lineno(function)
   2003/1    0.804    0.000  206.228  206.228 {built-in method builtins.exec}
        1    0.136    0.136  206.228  206.228 argumentationAPI.py:1252(post_argumention_explanations)
        1    3.839    3.839  206.047  206.047 argumentationAPI.py:798(__generate_dataframe_explanations)
        1    3.128    3.128   64.564   64.564 argumentationAPI.py:982(__generate_argument_cases)
     1000   12.881    0.013   47.223    0.047 argumentationAPI.py:619(__compute_dispute_tree)
     1000   33.728    0.034   45.577    0.046 argumentationAPI.py:546(__compute_grounded_extension)
226850/222231    0.479    0.000   42.481    0.000 indexing.py:1485(__getitem__)
   888636    0.270    0.000   36.864    0.000 notebook.py:216(__iter__)
   888636    1.513    0.000   36.594    0.000 std.py:1096(__iter__)
   131565    0.188    0.000   33.811    0.000 indexing.py:2205(_getitem_axis)
   114271    0.232    0.000   32.251    0.000 ops.py:134(get_iterator)
   114271    0.287    0.000   32.018    0.000 ops.py:811(__iter__)
   114268    0.314    0.000   31.463    0.000 ops.py:862(_chop)
   114269    0.250    0.000   30.683    0.000 indexing.py:2170(_get_slice_axis)
   114269    0.108    0.000   28.878    0.000 indexing.py:148(_slice)
   114269    0.667    0.000   28.770    0.000 generic.py:3155(_slice)
   114269    0.781    0.000   26.374    0.000 managers.py:684(get_slice)
88000/1000   14.543    0.000   22.099    0.022 argumentationAPI.py:665(recurse_child)
253898/250634    3.115    0.000   19.216    0.000 base.py:253(__new__)
   244222    0.395    0.000   19.043    0.000 numeric.py:67(_shallow_copy)
```

Figure C.4: Performance Optimisation pStats output for Optimisation Four with 1000 comments

```
Thu May 28 14:24:27 2020    pstats_mod

         37285592 function calls (36663583 primitive calls) in 40.640 seconds

   Ordered by: internal time, function name
   List reduced from 1275 to 20 due to restriction <20>

   ncalls  tottime  percall  cumtime  percall filename:lineno(function)
       19    4.358    0.229    4.358    0.229 {method 'fetchall' of 'sqlite3.Cursor' objects}
    38413    2.105    0.000    2.105    0.000 {pandas._libs.lib.maybe_convert_objects}
  5890394    1.947    0.000    4.527    0.000 {built-in method builtins.isinstance}
3859617/3855941    1.637    0.000    1.682    0.000 {built-in method builtins.getattr}
   2004/1    0.955    0.000   40.687   40.687 {built-in method builtins.exec}
  2674862    0.938    0.000    2.132    0.000 generic.py:7(_check)
    55270    0.682    0.000    1.006    0.000 {pandas._libs.lib.infer_dtype}
344934/316509    0.546    0.000    1.192    0.000 {built-in method numpy.array}
   412320    0.521    0.000    1.118    0.000 common.py:1845(_is_dtype_type)
   897009    0.515    0.000    0.522    0.000 {built-in method builtins.hasattr}
1928851/1647473    0.449    0.000    0.772    0.000 {built-in method builtins.len}
24315/21048    0.423    0.000    2.742    0.000 base.py:253(__new__)
  1768321    0.401    0.000    0.401    0.000 {built-in method builtins.issubclass}
   498876    0.400    0.000    2.288    0.000 base.py:75(is_dtype)
   172703    0.356    0.000    0.977    0.000 _dtype.py:319(_name_get)
168815/1734    0.341    0.000   20.625    0.012 function_base.py:2082(func)
   268030    0.340    0.000    0.662    0.000 dtypes.py:68(find)
   322828    0.336    0.000    0.587    0.000 abc.py:180(__instancecheck__)
    87289    0.324    0.000    0.675    0.000 indexing.py:1801(_is_scalar_access)
    16616    0.313    0.000    0.746    0.000 managers.py:186(_rebuild_blknos_and_blklocs)
```

Figure C.5: Performance Optimisation pStats output for Optimisation Five with 1000 comments

Appendix D

Configuration

```yaml
title: "AccuracyImprovement"

description: "An configuration to improve the accuracy of extraction"

general_settings:
  database_name: "welltrack_database"
  config_name: "config_workflow"
```

Figure D.1: Configuration File for the general settings

```yaml
data_insights_delivery_setting:
  res_calibration_points: "calibration_points"
  keep_units: ['m', 'sg', 'ppg', 'ft', 'mw']
  res_calibration_point_eval: "calibration_points_eval"
```

Figure D.2: Configuration File for the data insights delivery settings

```yaml
feature_extraction_setting:
  res_measure_feature: "measurements_feature"
  res_phrase_feature: "phrases_feature"
  res_negation_feature: "negation_feature"
  res_pos_feature: "pos_feature"
  res_domain_phrase_feature: "dp_feature"
  train_test_split:
    res_train_test_dataset: "train_test_dataset"
    train_split: 0.8
    random_seed: 2
  res_bert_embeddings: "bert_embeddings"
  res_decision_tree: "decision_tree"
```

Figure D.3: Configuration File for the feature extraction settings

```yaml
feature_analysis_setting:
  entity_linking:
    res_name_dictionary: "name_dictionary"
    ranking_scores_config:
      context_independent:
        use: True
        parameters: ["exact_string_match", "prefix_match"]
    ranking_config:
      norm_range: [0, 1]
      weights:
        exact_string_match: 1
        prefix_match: 1
    nil_threshold: 0.4
    res_linked_entities_feature: "linked_entities_feature"
  argumentation:
    res_default_cases: "argumentation_cases"
    res_comment_case_features: "comment_case_features"
    feature_table_config:
      phrase_table:
        use: True
        parameters:
          name: "phrases_feature"
      measurement_table:
        use: True
        parameters:
          name: "measurements_feature"
      part_of_speech_table:
        use: True
        parameters:
          name: "pos_feature"
      linked_entity_table:
        use: True
        parameters:
          name: "linked_entities_feature"
      feature_table:
        use: False
        parameters:
          name: "statistical_feature"
      domain_phrase_table:
        use: True
        parameters:
          name: "dp_feature"
      negation_table:
        use: True
        parameters:
          name: "negation_feature"
    res_automated_case: "automated_cases"
    res_arg_comment_expl: "explanations"
```

Figure D.4: Configuration File for the feature analysis settings

Figure D.5: Configuration File for the data extraction settings

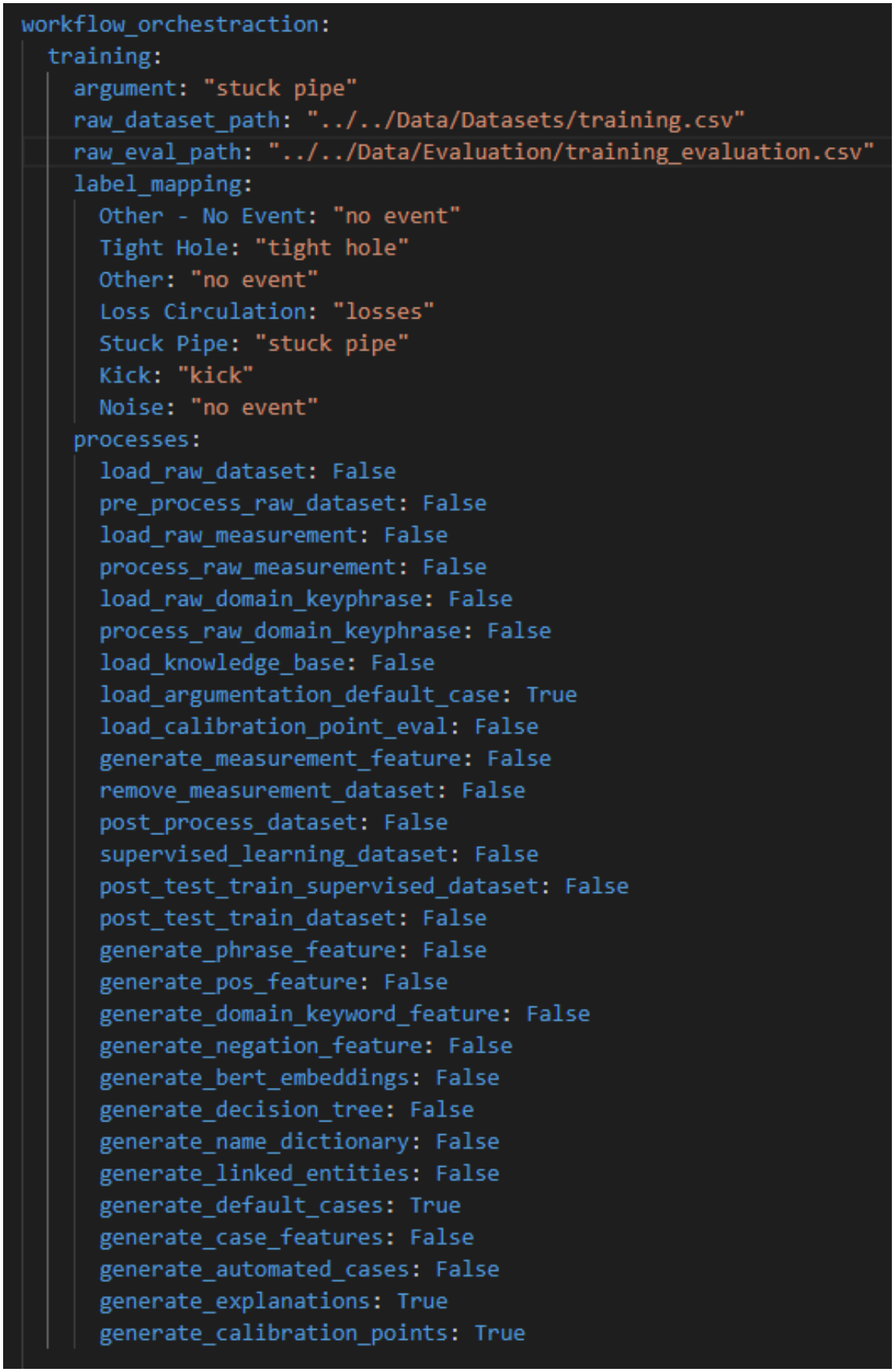


Figure D.6: Configuration File for the workflow orchestration settings

Appendix E

External Libraries

Library	Description	Reference
numpy (1.16.3)	Used for optimised scientific calculations in most pipelines	[CVW11]
pandas (0.24.2)	Used as the main data structure in all pipelines	[McK10]
PyYAML (5.1)	Used to parse the configuration file	[Sim20]
nltk (3.4.5)	Used in the statistical features pipeline	[BL02]
textstat (0.6.0)	Used to generate readability metrics for Exploratory Data Analysis	[BA20]
spacy (3.1.0)	Used to experiment with Named Entity Recognition	[HI17]
transformers (2.11.0)	Used to experiment with sequence classification with BERT	[Wol+19]
tensorflow (2.2.0)	Used to experiment with supervised machine learning neural networks	[Aba+15]
torch (1.5.0+cu101)	Used to experiment with supervised machine learning neural networks	[Pas+19]
scikit-learn (0.23.1)	Used to generate evaluation metrics	[Ped+11]
networkx (2.4)	Used to visualise argumentation graphs	[HSS08]
tqdm (4.31.1)	Used to track progress of loops	[Cos19]
notebook (6.0.3)	Used as the experimentation test bed	[Klu+16]
nbformat (5.04)	Used to auto generate Jupyter Notebooks in experiment setup	[Klu+16]
ipywidgets (7.5.1)	Used for tqdm to work with Jupyter Notebooks	[Klu+16]
wordcloud (1.6.0)	Used to visualise word frequency counts	[Mue20]
matplotlib (3.0.3)	Used to visualise graphical plots	[Hun07]

Table E.1: A table showing the external libraries used in the development of the project

Appendix F

Exploratory Data Analysis Visualisations

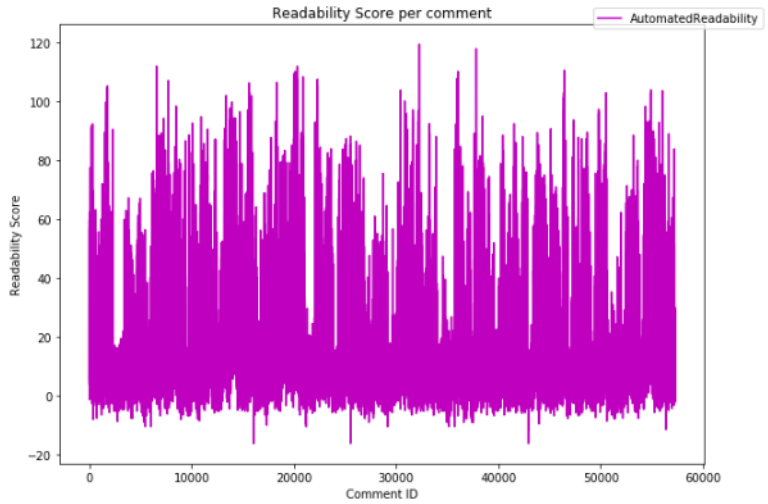


Figure F.1: Automated Readability Graph for Schlumberger Training Data Set

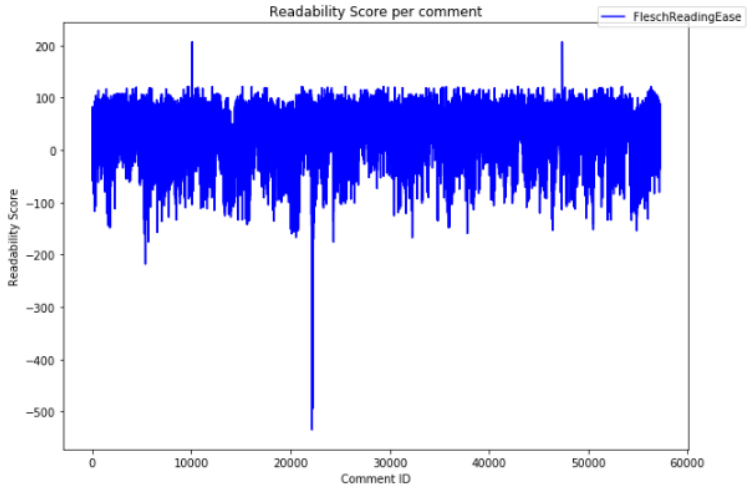


Figure F.2: Flesch Reading Ease Graph for Schlumberger Training Data Set

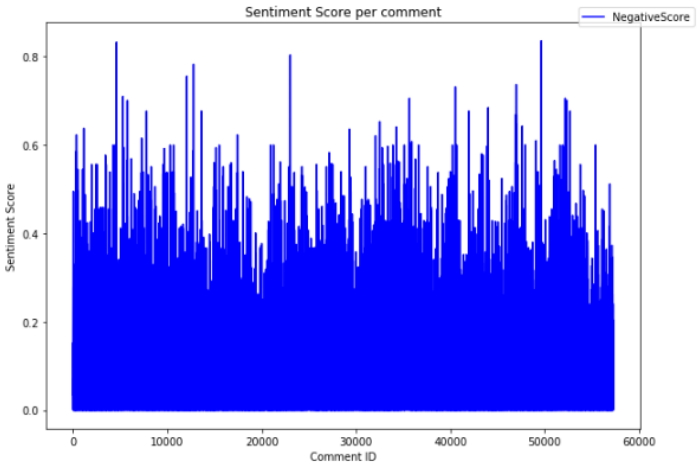


Figure F.3: Negative Sentiment Analysis Score for Schlumberger Training Data Set

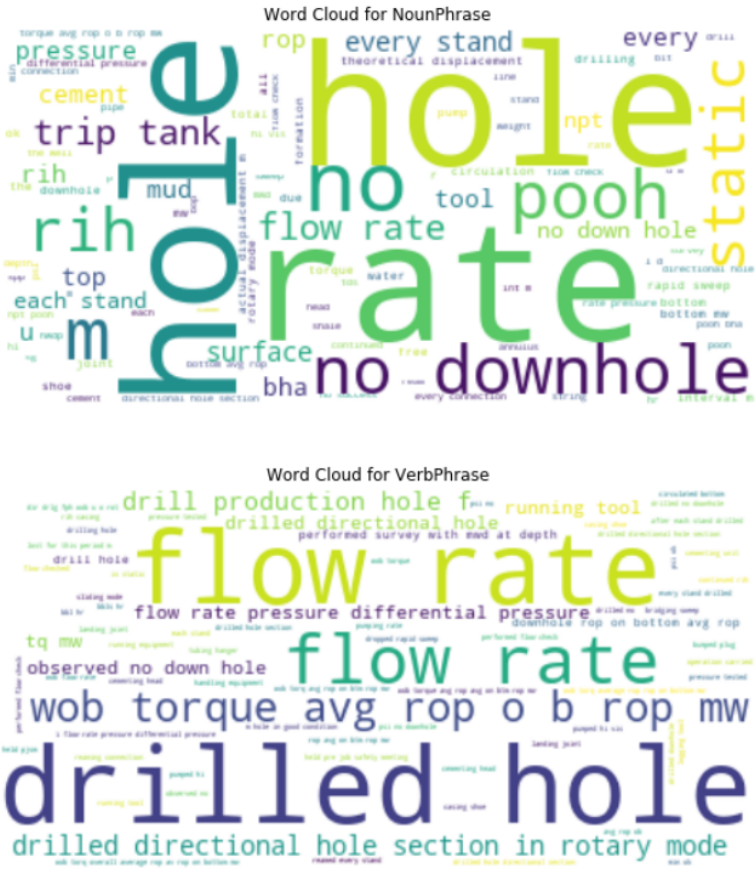


Figure F.4: Phrase Chunker Word Clouds for Schlumberger Training Data Set

NounPhrase

```
{' bha': 1416,
 ' every stand': 1711,
 ' flow rate': 1835,
 ' hole': 4356,
 ' m': 2923,
 ' mud': 1438,
 ' no': 3703,
 ' no downhole': 2370,
 ' pressure': 1407,
 ' rate': 4246,
 ' rih': 1383,
 ' rop': 1403,
 ' static': 1914,
 ' surface': 1628,
 ' top': 1282,
 ' trip tank': 1803,
 ' u': 1293,
 'hole': 3747,
 'pooh': 2322,
 'rih': 1988}
```

VerbPhrase

```
{'': 9877,
 ' downhole rop on bottom avg rop': 245,
 ' every stand drilled': 208,
 ' flow rate': 2599,
 ' flow rate pressure differential pressure': 318,
 ' lost for this period m': 224,
 ' observed no down hole': 261,
 ' running tool': 293,
 ' tq mw': 274,
 ' wob torque avg rop o b rop mw': 640,
 ' ï flow rate pressure differential pressure': 242,
 'drill hole': 243,
 'drill production hole f': 367,
 'drilled directional hole': 260,
 'drilled directional hole section in rotary mode': 444,
 'drilled hole': 2929,
 'drilling hole': 226,
 'flow rate': 982,
 'performed flow check': 219,
 'performed survey with mwd at depth': 258}
```

Figure F.5: Most Common Noun and Verb Phrases for Schlumberger Training Data Set

CommentCount	2639959
WordCount	1507526
NumberCount	260947
SymbolCount	739391
DecimalCount	61732
FractionCount	28435
TimeCount	1755
CommaNumberCount	0
RangeCount	40173

Figure F.6: Token Frequency Counts for Schlumberger Training Data Set

Bibliography

[All88] James Allen. *Natural Language Understanding*. 2nd. USA: Benjamin-Cummings Publishing Co., Inc., 1988. ISBN: 0805303308.

[Dun95] Phan Minh Dung. "On the acceptability of arguments and its fundamental role in nonmonotonic reasoning, logic programming and n-person games". In: *Artificial Intelligence* (1995). ISSN: 00043702. DOI: 10.1016/0004-3702(94)00041-X.

[OBM99] S. Ottesen, S. Benaissa, and J. Marti. "Down-Hole Simulation Cell for Measurement of Lubricity and Differential Pressure Sticking". In: *SPE/IADC Drilling Conference*. Amsterdam: Society of Petroleum Engineers, Apr. 1999. DOI: https://doi.org/10.2118/52816-MS. URL: http://www.onepetro.org/doi/10.2118/52816-MS.

[BL02] Steven Bird and Edward Loper. *NLTK: the Natural Language Toolkit*. 2002. DOI: https://doi.org/10.3115/1118108.1118117. URL: https://dl.acm.org/doi/10.3115/1118108.1118117.

[ZH03] Georg Zangl and Josef Hannerer. *Data Mining Applications in the Petroleum Industry*. Round Oak Pub, 2003. ISBN: 0967724813. URL: https://searchworks.stanford.edu/view/5806917.

[AS05] Jay Akers and Jay Sellers. "Use of Pressure-While-Drilling Tools to Improve Formation Integrity Test Interpretation". In: *SPE/IADC Drilling Conference*. Amsterdam: Society of Petroleum Engineers, Apr. 2005. DOI: https://doi.org/10.2118/91852-MS. URL: http://www.onepetro.org/doi/10.2118/91852-MS.

[Hun07] John D Hunter. *Matplotlib: A 2D Graphics Environment*. 2007. DOI: DOI:10.1109/MCSE.2007.55. URL: https://matplotlib.org/.

[HSS08] Aric A Hagberg, Daniel A Schult, and Pieter J Swart. *Exploring Network Structure, Dynamics, and Function using NetworkX*. 2008. URL: http://conference.scipy.org/proceedings/SciPy2008/paper_2/.

[MRS08] Christopher D Manning, Prabhakar Raghavan, and Hinrich Schütze. *Introduction to Information Retrieval*. 1st. New York: Cambridge University Press, 2008. ISBN: 0521865719. URL: https://dl.acm.org/doi/book/10.5555/1394399.

[Afs+09] M Afsari et al. "Mechanical Earth Model (MEM): An Effective Tool for Borehole Stability Analysis and Managed Pressure Drilling (Case Study)". In: *SPE Middle East Oil and Gas Show and Conference*. Manama: Society of Petroleum Engineers, 2009, pp. 15–18. DOI: https://doi.org/10.2118/118780-MS. URL: https://www.onepetro.org/conference-paper/SPE-118780-MS.

[BKL09] Steven Bird, Ewan Klein, and Edward Loper. *Natural Language Processing with Python*. First. Sebastopol: O'Reilly, 2009, pp. 261–286. URL: http://www.datascienceassn.org/sites/default/files/Natural%20Language%20Processing%20with%20Python.pdf.

[Esm+10] Bilal Esmael et al. "Automated Operations Classification using Text Mining". In: *The 3rd International Conference on Computational Intelligence and Industrial Application*. Huazhong: Research Gate, 2010. URL: https://www.researchgate.net/publication/233382409.

[McK10] Wes McKinney. *Data Structures for Statistical Computing in Python*. 2010. URL: https://pandas.pydata.org/pandas-docs/stable/index.html#.

[ST10] Catheryn Staveley and Paul Thow. "Increasing Drilling Efficiencies Through Improved Collaboration and Analysis of Real-Time and Historical Drilling Data". In: *SPE Intelligent Energy Conference and Exhibition*. Utrecht: Society of Petroleum Engineers, 2010, pp. 23–25. DOI: https://doi.org/10.2118/128722-MS.

[CVW11] S. Chris Colbert, Gaël Varoquaux, and Stéfan van der Walt. *The NumPy Array: A Structure for Efficient Numerical Computation*. 2011. DOI: DOI:10.1109/MCSE.2011.37. URL: https://numpy.org/.

[Ped+11] Fabian Pedregosa et al. *Scikit-learn: Machine Learning in Python*. 2011. URL: http://jmlr.org/papers/v12/pedregosa11a.html.

[Esm+12] Bilal Esmael et al. "A hybrid multiple classifier system for recognizing usual and unusual drilling events". In: *2012 IEEE International Instrumentation and Measurement Technology Conference Proceedings*. IEEE, May 2012, pp. 1754–1758. ISBN: 978-1-4577-1772-7. DOI: 10.1109/I2MTC.2012.6229541. URL: http://ieeexplore.ieee.org/document/6229541/.

[Osi12] Samuel O Osisanya. *Practical Approach to Solving Wellbore Instability Problems*. Tech. rep. Norman: The University of Oklahoma, 2012. URL: https://www.spe.org/dl/docs/2012/osisanya.pdf.

[Fil13] James J Filliben. *NIST/SEMATECH e-Handbook of Statistical Methods*. Ed. by Carroll Croarkin and Paul Tobias. NIST/SEMATECH, 2013, p. 1. URL: http://www.itl.nist.gov/div898/handbook/.

[CA14] David Cameron and Steria As. "Big Data in Exploration and Production: Silicon Snake-Oil, Magic Bullet, or Useful Tool?" In: *SPE Intelligent Energy Conference & Exhibition*. Utrecht: Society of Petroleum Engineers, 2014, pp. 1–3. DOI: https://doi.org/10.2118/167837-MS. URL: https://www.onepetro.org/conference-paper/SPE-167837-MS.

[DV14] Colin Dawson and Harry Verkuil. "SPE-167846-MS From a Daily Drilling Report to a Data and Performance Management Tool". In: *From a Daily Drilling Report to a Data and Performance Management Tool*. Vol. 2. Society of Petroleum Engineers, 2014, p. 167846. DOI: https://doi.org/10.2118/167846-MS.

[RDF14] RDF Working Group. *Resource Description Framework (RDF)*. 2014. URL: https://www.w3.org/RDF/.

[Aba+15] Martín Abadi et al. *TensorFlow: Large-scale machine learning on heterogeneous systems*. 2015. URL: tensorflow.org.

[ABJ15] Guus Aerts, Anders Brun, and Marte Jerkø. *How to achieve 50 percent reduction in offshore drilling costs*. 2015. URL: https://www.mckinsey.com/industries/oil-and-gas/our-insights/how-to-achieve-50-percent-reduction-in-offshore-drilling-costs#.

[Agg15] Charu C Aggarwal. *Data Mining The Textbook*. eng. 1st ed. 2015. Springer International Publishing, 2015. ISBN: 3-319-14142-2. DOI: 10.1007/978-3-319-14142-8. URL: https://www.springer.com/gp/book/9783319141411.

[Bat+15] David T Bateman et al. *Natural Language Processing For Extracting Conveyance Graphs*. 2015. URL: https://patents.google.com/patent/US9251139B2/en.

[BI15] M R Brulé and IBM Software Group. "The Data Reservoir: How Big Data Technologies Advance Data Management and Analytics in E&P Introduction-General Data Reservoir Concepts". In: *SPE Digital Energy Conference and Exhibition*. Woodlands: Society of Petroleum Engineers, 2015, pp. 3–5. DOI: https://doi.org/10.2118/173445-MS. URL: https://www.onepetro.org/conference-paper/SPE-173445-MS.

[JG15] J Johnston and A Guichard. "New Findings in Drilling and Wells using Big Data Analytics". In: *Offshore Technology Conference*. Houston: Offshore Technology Conference, 2015, pp. 4–7. DOI: https://doi.org/10.4043/26021-MS. URL: https://www.onepetro.org/conference-paper/OTC-26021-MS.

[SWH15] Wei Shen, Jianyong Wang, and Jiawei Han. "Entity linking with a knowledge base: Issues, techniques, and solutions". In: *IEEE Transactions on Knowledge and Data Engineering* 27.2 (Feb. 2015), pp. 443–460. DOI: 10.1109/TKDE.2014.2327028. URL: https://ieeexplore.ieee.org/document/6823700.

[SCS15] Mohamed Sidahmed, Christopher J Coley, and Shawn Shirzadi. "Augmenting Operations Monitoring by Mining Unstructured Drilling Reports". In: *SPE Digital Energy Conference and Exhibition*. BP. Woodlands: Society of Petroleum Engineers, 2015, pp. 3–5. DOI: https://doi.org/10.2118/173429-MS. URL: https://www.onepetro.org/conference-paper/SPE-173429-MS.

[Bre16] Nelson E Brestoff. *Using Classified Text and Deep Learning Algorithms to identify risk and provide early warning*. 2016. URL: https://patents.google.com/patent/US9552548B1/en.

[CST16] Kristijonaš Cyras, Ken Satoh, and Francesca Toni. "Abstract Argumentation for Case-Based Reasoning". In: *Proceedings of the Fifteenth International Conference on Principles of Knowledge Representation and Reasoning*. Cape Town: AAAI Press, 2016, pp. 549–552. DOI: 10.5555/3032027.3032100. URL: https://dl.acm.org/doi/10.5555/3032027.3032100.

[Klu+16] Thomas Kluyver et al. *Jupyter Notebooks – a publishing format for reproducible computational workflows*. 2016. DOI: 10.3233/978-1-61499-649-1-87.

[Knö16] Leonard Knöll. "The Process of Building a Mechanical Earth Model Using Well Data". Leoben, 2016. URL: https://pure.unileoben.ac.at/portal/files/1868591/AC13436690n01vt.pdf.

[ARG17] Sethupathi Arumugam, Shebi Rajan, and Sanjay Gupta. "Augmented Text Mining for Daily Drilling Reports using Topic Modeling and Ontology". In: *SPE Western Regional Meeting*. Bakersfield: Society of Petroleum Engineers, 2017. DOI: https://doi.org/10.2118/185711-MS. URL: https://www.onepetro.org/conference-paper/SPE-185711-MS.

[Ene17] Energistics. *The Solution Lean Automated Reporting Powered by WITSML Data*. 2017. URL: https://www.energistics.org/wp-content/uploads/2018/01/2017ids-case-study.pdf.

[HI17] Matthew Honnibal and Montani Ines. *spaCy 2: Natural language understanding with Bloom embeddings, convolutional neural networks and incremental parsing*. 2017. URL: https://github.com/explosion/spaCy.

[Hof+18] Julio Hoffimann et al. "Sequence Mining and Pattern Analysis in Drilling Reports with Deep Natural Language Processing". In: *Sequence Mining and Pattern Analysis in Drilling Reports with Deep Natural Language Processing*. Dallas: Society of Petroleum Engineers, 2018, pp. 24–26. DOI: 10.2118/191505-MS. URL: https://www.onepetro.org/conference-paper/SPE-191505-MS.

[IAD18] IADC. *Daily Drilling Report based on Sensor Data*. Tech. rep. International Association of Drilling Contractors, 2018. URL: https://www.iadc.org/wp-content/uploads/2018/05/IADC-Daily-Drilling-Report_RevA.pdf.

[MT18] Mehdi Mohammadpoor and Farshid Torabi. "Big Data analytics in oil and gas industry: An emerging trend". In: *Petroleum* (2018). DOI: https://doi.org/10.1016/j.petlm.2018.11.001. URL: http://www.sciencedirect.com/science/article/pii/S2405656118301421.

[Sal18] Zoë Wilkinson Saldaña. *Sentiment Analysis for Exploratory Data Analysis*. 2018. URL: https://programminghistorian.org/en/lessons/sentiment-analysis.

[Cos19] Casper O da Costa-Luis. *tqdm: A Fast, Extensible Progress Meter for Python and CLI*. 2019. DOI: https://doi.org/10.21105/joss.01277. URL: https://joss.theoj.org/papers/10.21105/joss.01277.

[DS19] Athithan Dharmaratnam and Schlumberger. *Information Extraction from Daily Drilling Reports Using Machine Learning*. 2019.

[Gan19] Kavita Ganesan. *All you need to know about text preprocessing for NLP and Machine Learning*. 2019. URL: https://towardsdatascience.com/all-you-need-to-know-about-text-preprocessing-for-nlp-and-machine-learning-bc1c5765ff67.

[Li19] Susan Li. *A Complete Exploratory Data Analysis and Visualization for Text Data*. 2019. URL: https://towardsdatascience.com/a-complete-exploratory-data-analysis-and-visualization-for-text-data-29fb1b96fb6a.

[Luq+19] Carmen Luque et al. *An advanced review on text mining in medicine*. May 2019. DOI: 10.1002/widm.1302. URL: https://onlinelibrary.wiley.com/doi/abs/10.1002/widm.1302.

[19] *National Data Repository*. 2019. URL: https://en.wikipedia.org/wiki/National_Data_Repository.

[NS19] Christine Noshi and Jerome Schubert. "A Brief Survey of Text Mining Applications for the Oil and Gas Industry". In: *International Petroleum Technology Conference*. Beijing: International Petroleum Technology Conference, 2019. DOI: https://doi.org/10.2523/19382-MS. URL: https://www.onepetro.org/conference-paper/IPTC-19382-MS.

[Pas+19] Adam Paszke et al. *PyTorch: An Imperative Style, High-Performance Deep Learning Library*. 2019. URL: http://papers.neurips.cc/paper/9015-pytorch-an-imperative-style-high-performance-deep-learning-library.pdf.

[Wol+19] Thomas Wolf et al. *HuggingFace's Transformers: State-of-the-art Natural Language Processing*. 2019. URL: https://arxiv.org/abs/1910.03771.

[BA20] Shivam Bansal and Chaitanya Aggarwal. *textstat*. 2020. URL: https://pypi.org/project/textstat/.

[20a] *Cavings*. 2020. URL: https://www.glossary.oilfield.slb.com/en/Terms/c/cavings.aspx.

[Goo20a] Google Developers. *Classification: Accuracy*. 2020. URL: https://developers.google.com/machine-learning/crash-course/classification/accuracy.

[Goo20b] Google Developers. *Classification: Precision and Recall*. 2020. URL: https://developers.google.com/machine-learning/crash-course/classification/precision-and-recall.

[Goo20c] Google Developers. *Classification: True vs. False and Positive vs. Negative*. 2020. URL: https://developers.google.com/machine-learning/crash-course/classification/true-false-positive-negative.

[Goo20d] Google Developers. *Confusion Matrix*. 2020. URL: https://developers.google.com/machine-learning/glossary#confusion_matrix.

[Ite20] DVC Iterative.ai. *Versioning Data and Model Files*. 2020. URL: https://dvc.org/doc/use-cases/versioning-data-and-model-files.

[20b] *Kick*. 2020. URL: https://petrowiki.org/index.php?title=Kicks&oldid=48073.

[20c] *Leakoff Test*. 2020. URL: https://www.petropedia.com/definition/2223/leakoff-test.

[20d] *Lost circulation*. 2020. URL: https://petrowiki.org/index.php?title=Lost_circulation&oldid=48286.

[MM20] Bob Malone and Malone Petroleum Consulting. *Daily Drilling Report*. 2020. URL: https://www.malonepetroleumconsulting.com/articles/daily_drilling_report.

[Mue20] Andreas Mueller. *WordCloud*. 2020. URL: https://amueller.github.io/word_cloud/.

[Rex20] RexEgg. *Regex Cheat Sheet*. 2020. URL: https://www.rexegg.com/regex-quickstart.html.

[Sch20] Schlumberger. *Schlumberger Oilfield Glossary*. 2020. URL: https://www.glossary.oilfield.slb.com/maincredits.aspx.

[Sim20] Kirill Simonov. *PyYAML - The next generation YAML parser and emitter for Python*. 2020. URL: https://github.com/yaml/pyyaml.

[Sma20] SmartBear. *OpenAPI Specification*. 2020. URL: https://swagger.io/docs/specification/basic-structure/.

[Soc20] Society of Petroleum Engineers. *PetroWiki Permissions*. 2020. URL: https://petrowiki.org/index.php?title=PetroWiki:Permissions&oldid=49608.

[20e] *Tight Hole*. 2020. URL: https://www.petropedia.com/definition/3948/tight-hole.

[20f] *Universal POS tags*. 2020. URL: https://universaldependencies.org/u/pos/all.html.

[Wik20a] Wikipedia contributors. *F1 score*. 2020. URL: https://en.wikipedia.org/w/index.php?title=F1_score&oldid=961198411.

[Wik20b] Wikipedia contributors. *Matthews correlation coefficient*. 2020. URL: https://en.wikipedia.org/w/index.php?title=Matthews_correlation_coefficient&oldid=958781090.

[Wik20c] Wikipedia contributors. *Tf–idf*. 2020. URL: https://en.wikipedia.org/w/index.php?title=Tf%E2%80%93idf&oldid=962443067.

[20g] *YAML vs JSON*. 2020. URL: https://www.json2yaml.com/yaml-vs-json.