AN ABSTRACT OF THE THESIS OF

Michael A. Castillo for the degree of Master of Science in Nuclear Engineering presented on December 6, 2023.

Title: Classification Modeling of Nuclear Power Plant Outage Severities with Complement Naïve Bayes and Bidirectional Encoder Representations from Transformers.

Abstract approved:

_____

Andrew C. Klein

Low-level nuclear power plant outages in the United States can lead to unanticipated costs, potentially compromising the expected operation lifetime of the plant. Nuclear power plants are complex systems of interfacing components and highly regulated processes. This inherent complexity makes predicting outages from system dependencies very challenging. When outages do occur, natural language is used to explain the cause, effect, and solution of the outage.

The purpose of this study is to construct a classification model capable of accurately predicting the severity of an unseen nuclear power plant outage using historical natural language and machine learning algorithms. The construction of a classification model leveraging historical light water reactor text data can provide experts better insight into outages of varying severity. The performance of a Naïve Bayes algorithm is compared to that of the state-of-the-art Bidirectional Encoding Representations from Transformers (BERT) model. To improve the performance of the BERT model, external fast reactor text data is applied to the training methodology.

Classification Modeling of Nuclear Power Plant Outage Severities with Complement Naïve Bayes and Bidirectional Encoder Representations from Transformers

by
Michael A. Castillo

A THESIS

submitted to

Oregon State University

in partial fulfillment of
the requirements for the
degree of

Master of Science

Presented December 6, 2023
Commencement June 2024

Master of Science thesis of Michael A. Castillo presented on December 6, 2023

APPROVED:

Major Professor, representing Nuclear Engineering

Head of the School of Nuclear Science and Engineering

Dean of the Graduate School

I understand that my thesis will become part of the permanent collection of Oregon State University libraries. My signature below authorizes release of my thesis to any reader upon request.

Michael A. Castillo, Author

ACKNOWLEDGEMENTS

I would first like to express my deepest appreciation to my advisor, Dr. Andrew Klein, for the guidance these past few years. Never did I imagine that the opportunity you have given me would teach me so much more than just science. I would also like to thank Dr. Curtis Smith at Idaho National Laboratory for providing access to the MORP data and his early guidance in this effort.

To my parents Stephanie and George, and my brother, Kuya CJ. Challenges are easier to overcome when the support given is as selfless, and continuous as what you have given me. I love you all and thank you for everything.

Lastly, to my wife, Fatima. There are not enough words to express my gratitude for the wisdom, patience, and understanding you provided me since the very beginning. Every obstacle I've faced you've resolutely treated as your own, and I could not have done it without you.

TABLE OF CONTENTS

Page

1 Introduction 1

1.1 Data Source 2

1.2 Objective 3

1.3 Structure of the Document 5

2 Background 6

2.1 Introduction to Artificial Intelligence 6

2.2 Fundamental approaches to Learning 6

2.2.1 Supervised Learning 6

2.2.2 Unsupervised Learning 7

2.3 Artificial Intelligence in Natural Language Processing 7

3 Approaches to Natural Language Processing 9

3.1 Rule Based Systems 9

3.2 Machine Learning 11

3.3 Deep Learning with Bidirectional Encoder Representations from Transformers 14

4 Natural Language Processing in Nuclear Power 17

5 Monthly Operating Report Database 19

5.1 Data Exploration 19

5.2 Dataset Attributes 20

5.2.1 Summary Column 20

5.2.2 Outage Hours Data 21

TABLE OF CONTENTS (Continued)

Page

5.3 Limitations and Modeling Approach 21

6 Theory 22

6.1 Vector Representation: Complement Naïve Bayes 22

6.2 Complement Naïve Bayes 24

6.3 Bidirectional Encoder Representations from Transformers 26

6.4 Encoders 28

6.4.1 Embedding Layer 29

6.4.2 Transformer Encoder 30

6.4.3 Self-Attention 31

6.4.2 Feed Forward Neural Network 32

6.5 Training Objectives 32

6.6 Transfer Learning 33

7 Methodology 34

7.1 Data Preparation 35

7.1.1 Cleaning 35

7.1.2 Labeling 38

7.2 BERT Models 42

7.3 Further Training BERT Models 42

7.4 BERT Training 43

7.4.1 Cleaning Vs. Uncleaned 44

TABLE OF CONTENTS (Continued)

7.4.2 Base Models Vs. Further Trained 44

7.4.3 Batch Size and Max Sequence Length 44

7.4.4 Epochs 45

7.4.5 Weighting 46

7.4.6 Learning Rate Decay 46

7.5 Naïve Bayes Data Preparation 47

7.6 Naïve Bayes Models 47

7.7 Metrics 47

7.8 Training Computational Environment 49

8 Results 50

8.1 Top Precision Scores 50

8.2 Top Recall Scores 58

8.3 F1 Scores by Hour 66

9 Discussion 74

10 Conclusion 76

11 Future Work 78

Bibliography 80

Appendices 85

LIST OF FIGURES

Figure Page

Figure 5-1: Outage Hour Distribution..... 19

Figure 5-2: Outage Hours by Docket..... 20

Figure 6-1: Transformer Architecture (Ashish Vaswani, 2017)..... 27

Figure 6-2: Encoder and Decoder Components of Transformer 28

Figure 6-3: Encoder Component (Ashish Vaswani, 2017)..... 29

Figure 6-4: Input Representation (Jacob Devlin, 2018)..... 30

Figure 6-5: Encoder Stack 31

Figure 6-6: Encoder Sublayers..... 32

Figure 7-1: Outage Occurrences 34

Figure 7-2: Relevant Outage Reason Occurrences 35

Figure 7-3: Class Distributions for Defined Severity Limits..... 40

Figure 7-4: Stratification of 744 Hour Limit Dataset 41

Figure 7-5: Stratification of 250 Hour Limit Dataset 41

Figure 7-6: Example of MLM (Reimers, 2022)..... 43

Figure 8-1: 744+ Hour Limit Precision Scores, All Models..... 51

Figure 8-2: 500+ Hour Limit Precision Scores, All Models..... 52

Figure 8-3: 400+ Hour Limit Precision Scores, All Models..... 53

Figure 8-4: 350+ Hour Limit Precision Scores, All Models..... 54

Figure 8-5: 300+ Hour Limit Precision Scores, All Models..... 55

Figure 8-6: 250+ Hour Limit Precision Scores, All Models..... 56

Figure 8-7: 744 Hour Limit Recall Scores, All Models..... 59

Figure 8-8: 500+ Hour Limit Recall Scores, All Models 60

Figure 8-9: 400 Hour Limit Recall Scores, All Models..... 61

LIST OF FIGURES (Continued)

Figure Page

Figure 8-10: 350 Hour Limit Recall Scores, All Models..... 62

Figure 8-11: 300 Hour Limit Recall Scores, All Models..... 63

Figure 8-12: 250 Hour Limit Recall Scores, All Models..... 64

Figure 8-13: 744 Hour Limit F1 Scores, All Models 67

Figure 8-14: 500 Hour Limit F1 Scores, All Models 68

Figure 8-15: 400 Hour Limit F1 Scores, All Models 69

Figure 8-16: 350 Hour Limit F1 Scores, All Models 70

Figure 8-17: 300 Hour Limit F1 Scores, All Models 71

Figure 8-18: 250 Hour Limit F1 Scores, All Models 72

LIST OF TABLES

Table Page

Table 1-1: Description of MORP Data Fields 4

Table 6-1: Input Sentences..... 22

Table 6-2: Resulting Vector Representations 23

Table 6-3: TFIDF Vector Representations 24

Table 7-1: Categorical Definitions for Outage Method..... 39

Table 7-2: Summary of Dataset Class Distributions..... 41

Table 7-3: BERT Large Configurations 45

Table 7-4: DistilBert Base Configurations 45

Table 7-5: Early Stopping Parameters 46

Table 7-6: Learning Rate Scheduler Parameters 47

Table 7-7: Confusion Matrix 48

Table 8-1: Top Precision Score by Hour 57

Table 8-2: Confusion Matrix of 500+ Hour Limit, Max Precision Model 58

Table 8-3: Top Recall Score by Hour..... 65

Table 8-4: Confusion Matrix of 300 Hour Limit, Max Recall Model 66

Table 8-5: Top F1 Score by Hour..... 73

LIST OF APPENDICES

Appendix		Page
A.	Python Code	85
B.	masterExpansions.txt	153
C.	hyphensInVocab.txt	160
D.	unknown.txt	168
E.	reformat.txt	169
F.	unknownWordNumbers.txt	177

1 Introduction

As the United States nuclear power industry works toward deploying advanced reactor designs, existing light water reactors face financial strains as they struggle to maintain themselves as economic options in the energy market. With the rise of cheap natural gas, the aging fleet of nuclear reactors must reduce costs to compete as an energy option. Some of the largest expenses that are difficult to mitigate are low-level accidents that lead to extended unplanned shutdowns and events that are extremely challenging to identify and predict. Most of these accidents are extremely small compared to the large scope of overall operations, but they can develop into costly consequences that may lead to premature retirement of the plants.

As an example, on June 7th, 2011, an electrical switchgear that distributes power to vital systems and components needed for the safe shutdown of the Fort Calhoun Station (FCS) caught fire (Special NRC Oversight at Fort Calhoun Station, 2022). According to the reports, the fire started in a replacement circuit breaker that was modified to fit inside the existing electrical switchgear. Poor alignment between the electrical components and lack of cleaning of the connections increased the electrical resistance at the junction causing a fire and complete loss of spent fuel pool cooling for 90 minutes. Upon inspection, there were also several other degradations found which forced the Nuclear Regulatory Commission (NRC) to place Fort Calhoun Station under its Multiple/Repetitive Degraded Cornerstone categorization, which requires the fixing of multiple adverse conditions within the power plant. The outage cost the district $341 M with an additional $195 M for startup (Murphy P. V., 2015), (Robertson, 2016), costing a total of $536 M. In 2016, after extensive financial analysis, Omaha Public Power District chose to shut down the reactor due to an estimated loss of approximately $900 M over the next 20 years (World Nuclear News, 2016). These low-level accidents caused an unexpected loss of $536 M, which in hindsight may have provided Fort Calhoun nuclear station more financial options to offset losses from future economic challenges.

Of course, in hindsight, there are many simple things that could have been done to stop the fire from occurring altogether. The simplest task would be executing routine cleaning of the conductors to reduce electrical resistance. Nonetheless, this was a task that was not seen as a priority and simply flew under the radar for concern. In fact, the replacement breaker was operating for nearly 18 months before the fire occurred without any indications of concern. This low-level, but severe, accident is tremendously difficult to identify, let alone predict. With most U.S. Nuclear Power Plants (NPP) nearing 50 years of operation, more work is needed to mitigate and reduce these types of low-level, “needle in a haystack”, severe outages that lead to premature shutdowns.

To the researcher's knowledge, there is no existing body of literature for predicting these types of accidents. The complex nature of the problem requires the ability to predict with such extreme initial conditions and unknowns, any solution may be too restrictive to be generalized across all power plant systems. Therefore, instead of devising a solution to predict an outage occurrence, this research aims to develop a tool that can provide an estimated duration of an outage post hoc.

In the last decade, developments in the field of predictive maintenance have begun to shape the methodology for understanding problems surrounding systems and processes. Advancements in Machine Learning (ML), Deep Learning (DL) and statistical analysis have been the cornerstone in these developments due to the ability to derive knowledge from incredibly complex and large amounts of data. Using ML and modern sensors, real time data from system components can be analyzed to provide useful insight for lifetime longevity and operational decisions. However, with an aging fleet of nuclear reactors, retrofitting a solution for assessing system wide relationships is expensive and time consuming. Furthermore, unless an entire digital twin model of an existing Nuclear Power Plant (NPP) is developed, predictive maintenance of only specific components of a NPP may not be sufficient in describing systemic relationship. To minimize severe cost, NPP licensees require a tool that can quickly assess the severity of present and future outages that account for system wide dependencies. Unlike much of the current work in predictive maintenance where numerical data is utilized, this research focuses on leveraging historically recorded natural language and Natural Language Processing (NLP) techniques to develop such a tool.

NLP is a field within Artificial Intelligence (AI) that focuses on getting computers to understand human language by using text data. NLP has been successfully deployed across many industries particularly in topic modeling, language translations, chat bots and many human-computer interfaces. NLP methods contain ML and DL techniques that derive patterns from large amounts of text data. The authoritative nature of the Nuclear Regulatory Commission (NRC) has led to a rich recorded history of NPP operations and experience, filled with extensively formatted documentation, records, and reports. The text within these sources is of great value for developing NLP solutions, yet there has been minimal work conducted in this area. Part of this research aims to expand on this technology and identify the scope in which NLP is applicable to the nuclear industry.

1.1 Data Source

The nuclear power industry has an unprecedented reputation of being a heavily regulated entity that cannot afford mistakes. By bestowing regulative and authoritative guidance, the NRC helps ensure that nuclear power remains a safe energy producer. One way the NRC can guarantee safety and successful operation is by monitoring and documenting the performance of NPP's through standardized reporting procedures for licensees that manage commercial NPP's in the U.S. Over the long history of the

nuclear power industry, these documented reports have led to the development of numerous structured and unstructured databases that encompass a vast number of measurable quantities to describe the operation of NPP's.

The database used in this research contains documented monthly operating reports for all NPP's in the U.S. Together, the NRC and Idaho National Laboratory worked to develop this structured database containing all monthly outages between January 1997 and March 2016. The Monthly Operating Report Database (MORP) was designed to collect data that reflects operating statistics and shutdown experience for assisting NRC management in identifying poor and/or declining safety performance, as well as good and/or improving performance (Marcel R. Harper, 1997). Within MORP is a section dedicated to unit shutdowns, where details for describing every shutdown a NPP unit has experienced is documented. Highly structured and organized, the information in the unit shutdowns section is provided in a spreadsheet of data columns that reflect the name of the NPP unit, start date of each shutdown, type of shutdown, duration, reason for each shutdown, the method of shutdown and a written summary of the shutdown. To instill standardization, some data fields are only allowed specific categorical parameters to be entered. For example, "type of shutdown" only allows for letter entries of "F" (forced) or "S" (scheduled). "Reason for each shutdown" accepts categorical entries represented as a range of letters, where each letting indicates a general reason, for instance, "A" represents equipment failure, "B" means maintenance or testing and so forth. Each of these columns provides a general expression for the outage experienced by the individual NPP. However, for a more explicit description of why the NPP was shutdown, one would refer to the "summaries" (SUM) column. In the SUM column, the operator provides textual information specific to the observed event during the outage. Based on the reporting requirements set by the NRC, each summary consists of written free text that explains the reasons for each shutdown, and if available, provides the corrective action taken. This text data is rich with information that cannot be expressed with traditional integers or numerical values. The text provides context, cause and the effects, system components, system relationships and physical phenomenon responsible for the outage. Unlike Licensee Event Reports (LER), this combination of rich text data and metadata existing as a structured dataset makes MORP unique in NLP tasks.

1.2 Objectives

The MORP database is made of two datasets containing power plant generation data and power plant outage data. This research focuses on the outage data, which contain data fields requested by the NRC to document events where the generator is offline regardless of the reactor status. A description of each data field is provided in Table 1-1.

Table 1-1: Description of MORP Data Fields

Field	Description
DOCKET	Last three digits of plant docket number
RPT_PERIOD	Applicable report period (YYYYMM)
DESCRIP	Outage description
OUTG_DATE	Start date of outage
OUTG_HRS	Outage duration (hours)
OUTG_LER	Licensee Event Report number as applicable
OUTG_METH	Method of shutting down the reactor: 1 - Manual (normal reactor shutdown or generator offline with reactor critical) 2 - Manual Scram 3 - Auto Scram 4 - Continued (from previous month) 5 - Reduced Load (only captured through August 1997) 9 - Other (outages that transition within the month to another outage)
OUTG_REASN	Outage reason: A - Equipment Failure B - Maintenance or Test C - Refueling D - Regulatory Restriction E - Operator Training and License Examination F - Administrative G - Operational Error H - Other
OUTG_SEQ	Sequential number assigned to each outage by the licensee
OUTG_TYPE	Outage type: forced (F) or scheduled (S). A forced outage is one required to be initiated no later than the weekend following discovery of an off normal condition). All other outages are scheduled outages.
OUTG_COMP	Component codes*

OUTG_SYSTM	System codes*
Refuel Cycle	The number of the refuel cycle
Outage_Date_time	Start date/time of outage

* Fields no longer requested/discontinued by NRC Generic Letter 97-02

MORP offers 20 years of data that captures various perspectives of a power plant outage, including documented summaries within the 'DESCRIP' column of the database. Using both text data from the reported summaries and the outage duration of the generator is offline for provided in 'OUTG_HRS', this research aims to train a classification model using machine learning (ML) techniques to predict the severity of future unseen NPP outages. Text features provided in outage summaries contain essential information, outlining interfaces within a complex system. A classification model can provide an approximated off-line time based on historical recorded data to assist Light Water Reactor (LWR) operations and economic forecasting. Researchers can also utilize the model to better characterize outages and further understand the complex relationships between systems at a higher level. It is hypothesized that MORPs inherent structured nature and relevant data pertaining to NPP outages, a text classification model can be fine-tuned with the state-of-the-art language framework, Bidirectional Encoder Representations from Transformers (BERT). BERT has been shown to perform exceptionally well across various NLP tasks including text classification (Jacob Devlin, 2018). Most NLP tasks solved with BERT are not applied with text data relevant to the power industry, thus there exists very little peer reviewed literature on the fine-tuning performance of base BERT for the nuclear power domain. This research aims to close this gap by assessing its classification performance on relevant NPP outage data. This objective is accomplished by performing the following:

- Obtain relevant information regarding previous work in NLP,
- Identify successful text classification methods using MORP data,
- Perform text classification,
- Compare results of different methods.

1.3 Structure of the Document

This thesis is outlined in the following manner. Section 2.0 covers an introduction to the field of ML, its relation to NLP, current trends and a review of literature addressing past research inside and outside the nuclear power industry. Section 3.0 will discuss the theory that drives state-of-the-art tools in NLP. Section 4.0 contains the methodology, and the last section covers the results, conclusion, and future work of this study

2 Background

As computational hardware and refined software methods improve, so does the field of NLP. This section will introduce ML, DL, and their relation to NLP, and will extend into current trends and a review of literature that covers the use of NLP inside and out of the nuclear industry.

2.1 Introduction to Artificial Intelligence

In effort to reduce ambiguity, it is necessary to define the clear distinction between AI, NLP and their constituent responsibilities. Both ML and DL have distinct meanings and exist as sub-fields within the domain of AI. The main objective for both is to leverage existing data that characterizes the real word to enable problem solving in machines. In general, ML – or "Classical" ML – uses algorithms and real-world features chosen by a human to make decisions that appear subjective. The algorithms require a human to define the correct input features tailored to a specific task, such as fitting of data, pattern detection or classification.

Similarly, DL is a subset of ML that utilizes artificial neural networks (ANN) and characteristically larger datasets to automate the extraction of features within data to make conclusive decisions without the need of human intervention. At its core, the ANN's within DL models emulate the human brain through a set of algorithms. At a high level, ANN's "learn" by self-adjusting weights and biases using mathematical techniques that quantify errors to improve future performance on tasks relating to classification, object recognition and object descriptions. A successful outcome from both ML and DL is to construct a general function representative to real world phenomenon in which the function can make predictions on unseen data with a certain accuracy. The primary difference between both methods resides in how each learns, and the amount of data required to establish a general function.

2.2 Fundamental Approaches to Learning

There is a constant growth of different algorithms being created, researched, and applied throughout ML and DL literature. However, majority of all these algorithms are built upon on two learning approaches: Unsupervised and Supervised learning. Reinforcement Learning would be considered another learning approach but will not be discussed here. Generally, the learning approach to be implemented is selected based on the type of problem being solved.

2.2.1 Supervised Learning

The objective of Supervised Learning is to learn a mapping from inputs to outputs, given a set of input-output pairs (Murphy K. P., 2012). From a high level, the machine "learns" by providing it features,

and “Supervising” the machine on what is correct or incorrect. A feature could be something as simple as a height or weight of a person, or as complex as an image or in the case of NLP, a sentence or number of words. In general, Supervised Learning approaches are used for classification or regression tasks. One of the biggest challenges and most important aspects to successfully train a model with Supervised Learning is obtaining enough data in a structured format for training the machine. Gathering quality data with relevant features applicable to the task then processing it in a structured format is time consuming and costly.

2.2.2 Unsupervised Learning

Unlike the necessity for structured data, Unsupervised Learning leverages unstructured – or “Unlabeled” – data for learning. Unsupervised Learning algorithms are used for discovering patterns, relationships, or groupings in data without human intervention. Due to the nature of this approach, Unsupervised Learning in many clustering tasks. Although the need for human intervention and structured data is appealing, the complexity associated with high volumes of unstructured training data can lead to extensive computational costs.

2.3 Artificial Intelligence in Natural Language Processing

Fundamentally, the basis of NLP is to digitally analyze text using computers and theories pertaining to both fields of computer science and linguistics. Many people in the NLP field agree that the Weaver memorandum (Weaver, 1949) was the initial spark to coupling computers and text. The memorandum outlined the initial methods for a developing a solution for translating between languages before the capabilities of a computer were known. This task (commonly known as machine translation) structured the desire and importance of utilizing computers for linguistic tasks, even when computational resources were not readily available. In 1954, a joint project between IBM and Georgetown University demonstrated the successful translation of 60 Russian sentences into English using a total of 250 words and six ‘grammar’ rules (Hutchins, 2004). This work inspired optimistic promise, resulting in a great push in NLP research between the late 1940’s and mid 1960’s. During this time many challenges were identified, particularly when dealing with the syntactic and semantic attributes associated with language. Due to the lack of computational resources, most NLP projects at that time came to a stop. However, between the late 1960’s and early 1990’s there was a significant boom in the accessibility and advancements in computers which advanced the framework and for how NLP tasks are approached today. In addition to better and more accessible technology, development of theoretical work allowed for an improved understanding in the types of challenges associate with language understanding using computers. For example, the theory of Case Grammar showed that the syntactic structure can be predicted by semantic entities (Fillmore, 1968), (Ye, 2015) and the Conceptual Dependency Theory worked to take a step back from highly specific applications of NLP (i.e., machine

translation) and instead worked to construct a general theory for human natural language understanding (Schank, 1972). As the theory of NLP began to expand, so did the solutions and strategies for interfacing human language with computers. Conceptual Ontologies worked to structure real-world information into a framework a computer could understand (Cullingford, 1977) and symbolic approaches built the foundation for fundamentals still applied today (e.g., tokenization) (Jonathan J. Webster, 1992). One of the more revolutionary aspects of NLP to come out of advancement in computational power in the 1980s-1990s was the investigation of statistical models (Liberman, 1991). Up until this point many of the systems to solve NLP tasks required extensive handwritten, laborious rules that attempted to cover many scenarios of human language. Instead, solutions began shifting to statistical models and ML approaches to leverage mathematical probability for making informed decisions on NLP tasks (Tanaka, 1996). Less than ten years later, Yoshua Bengio would introduce an approach that expressed the joint probability function of word sequences in terms of "feature vectors" to alleviate complications from high-dimensional spaces commonly found when modeling words in a sentence (Yoshua Bengio, 2003). "Feature vectors" -- or more commonly referred to as "word embeddings" -- are numerical vector representations of a word and are developed as a method to extract features out of text so computers and ML algorithms can work with them. By representing individual words in a sentence as vectors, they can be the subject of mathematical operations and lend themselves useful to ML strategies (Almeida, 2019). These word embeddings are central to many present applications of NLP. Much of the research in the last decade was on developing sophisticated techniques that can develop word embeddings to better capture context and general understanding of language. Models such as Word2Vec (Mikolov, 2013) and GloVe (Global Vectors for Word Representation) (Manning J. P., 2014) were at the forefront of NLP technology, until 2017 when the concept of attention-based Transformer methods (Ashish Vaswani, 2017) ushered in the framework for recent models such as BERT (Jacob Devlin, 2018) and GPT-3 (Brown, 2020).

3 Approaches to Natural Language Processing

The field of NLP is a continually evolving and heavily researched area, making identifying clear methodologies challenging for nuclear power domain experts without a background in NLP. This is mainly due to the many approaches one can take to successfully perform NLP tasks. It is essential that a well-defined hypothesis or tightly bounded scope, is constructed to help guide the research. Fundamentally, the most important elements to solving an NLP task is the available text data, the pre-processing methodology and the analytical tools chosen for the task. For example, when using a ML models, the quality and quantity of the data is important to have the model learn effectively (ML_Classifier_data_rews). Additionally, the pre-processing methods for converting text to machine-readable vectors can vary depending on the specified goals. Or if data is lacking, a rule-based system may be more beneficial for extracting information. Overall, even if a well-defined hypothesis is established, NLP tasks are all unique and require iterations and tuning to improve performance (Carola A. Gregorich, 2020). Thankfully, there are many Python libraries developed to automate many phases of a NLP pipeline. These libraries leverage both traditional and ML algorithms, including DL transformer-based models. It is important to note that many of these ML and DL models are trained using generic datasets, therefore their performance can be limited to the energy industry. Furthermore, much of the NLP literature pertaining to the nuclear power domain is very limited, especially with the use of state-of-the-art language models like BERT for classification. Therefore, a comprehensive literature review is provided below to address the current state and gaps of NLP in the nuclear industry.

3.1 Rules Based Systems

Before sophisticated language models and ML algorithms, many NLP applications were driven by rule-based systems (RBS). RBS utilize rules, facts, and manual labeling to derive knowledge from text data (Goldberg, 2017). This made NLP tasks challenging because many applications required domain experts that understood the content of the text, while also needing a linguist to derive rules from semantics and sentence dependencies. Typical RBS are algorithms based on “if-then” conditions that use the defined rules to drive the outcome of the model. Without modern tools, constructing rules required significant time, labor, and intellectual resources. Now, Python libraries such as SpaCy (spaCy, n.d.) and NLTK (nltk) make understanding unstructured text data much easier, allowing for automated parsing, identifying, and labeling of linguistic components.

Although open-sourced software libraries such as SpaCy and NLTK have indeed lowered the requirement for an expert linguist, they still require knowledge or referenceable data to be useful. More explicitly, these tools only have accessibility to generic, referenceable data. This can make development of NLP applications for specific domains challenging as the Python libraries may not be able to recognize words unique to that subject. Many industries have worked to bridge this gap by constructing comprehensive datasets comprised of unique vocabulary, phrases and even parts-of-

speech (POS). These datasets provide the knowledge to construct useful NLP applications, serving as the foundation of knowledge to reference. For example, researchers Valenzuela and Escarcega built a RBS to perform event extraction tasks for use in Bioinformatics (Valenzuela-Escárcega, 2015). Part of their methods included the ability to automatically identify specific keywords typically only found in bioinformatics, such as “TopBP1”, “cyclin-D1” and “ATR”. This process was possible because they referenced data from an extensive corpus constructed by Tomoko and Ohta (Tomoko Ohta, 2013). Tomoko and Ohta used large scale repositories and documents from the biomedical domain to construct a referenceable database to build from. Many industries have followed suit by constructing their own general datasets, but the nuclear power domain is lagging in structured referenceable material.

Without commonly known vernacular in the nuclear power industry to reference, modern tools like SpaCy will incorrectly process common keywords. For example, components like “steam generator” will be broken up into two words “steam” and “generator”, losing all relevance to a complex system. Because of this, many researchers using NLP in the nuclear power industry find themselves constructing their own datasets tailored to their needs. For example, Carola Gregorich and researchers at the Electrical Power Research Institute (EPRI) used internal and publicly available documents from the NRC to build a comprehensive dataset for supporting NLP applications in the nuclear power industry. Methods were not discussed, but many of these documents originated in various forms, requiring a sophisticated approach for pre-processing. Documents such as incident reports, operating experience, corrective action reports and evaluation reports were examined to construct a raw corpus of 1.3 million elements, including words, numbers, and punctuation. Using a subset of this data, researchers could use NLP tools to extract knowledge from existing documentation regarding leaks at NPPs. Driven by questions such as, “What is the concentration of radionuclides when there is a spill or leak?”, “What systems, structures, and components contribute to spills and leaks?” and “What work practices might be associated with spills and leaks?”, the researchers use the text data, unstructured documentation, and NLP techniques to gather knowledge about leaks. The result of this knowledge led to identifying total counts of systems, structures and components contributing to leaks. It was discovered that several cases of spills and leaks shared common causes associated with vent structures which was not a recognized common occurrence. This newfound knowledge helped assist and guide proactive measures in power plant systems.

When referenceable datasets don’t exist, other more linguistically driven methods may be applied to investigate unstructured data. Instead of relying on a referenceable dataset, researchers Yunfei Zhao et al. constructed a RBS by identifying keywords that are used when describing cause and effect relationships. A total of 11 keywords such as “caused” and “due to” were used for identifying causal events in LER’s. When a keyword was identified, the POS for each word were tagged using the

Stanford CoreNLP software package (C. Manning, 2014). The Stanford CoreNLP tools were then used to analyze the dependencies between the keywords and surrounding context in the sentence. Using the identified dependency and tagged POS, 184 rules were constructed to automatically extract the causal and consequent event documented in LER's. To further increase the performance of this tool, researchers express the necessity of developing a full complete set of rules. Without a full complete set of rules, the current state of the tool is limited in its capacity to examine certain sentences. It was found that some sentences did not contain matches for the combinations of part of speech and dependencies, requiring manual examination or other methods for rule development.

The clever approach to extracting causal relationships was driven by the ability to evaluate the relationships shared between a set of keywords. However, it is unclear if the researchers ran into domain related complications using the Stanford CoreNLP API. This toolset hosts deep learning and rule-based NLP tools with limited access to the data it is trained on. In other words, the performance on POS tagging may suffer if the tools implemented lack knowledge on nuclear power domain language.

The success of RBS systems are reliant on many factors. First, a tightly defined research goal is required to guide data collection, pre-processing techniques, and rule development. Secondly, existing NLP tools may not have the capability to comprehend nuclear domain language. Lastly, depending on the method, referenceable text data may be challenging to gather and construct. For the nuclear power domain, these challenges are not trivial as there is no common consolidated database of structured referenceable text, and useful documentation may not be available to the public. Furthermore, the construction of RBS are tedious and require a bit of explicit instruction, even with modern tools.

3.2 Machine Learning

NLP applications requiring rules require tightly defined research questions and available data. Although modern NLP tools have reduced the burden of linguistical expertise, due to knowledge restriction of nuclear domain data, there is still a massive reliance subject matter expert, and manual labor to characterize the problem being solved. Within the last 10 years, however, recent advancements in ML have lowered the requirements for NLP applications by adopting statistical and probabilistic models that depend on less stringent instructions. In some aspect, dependency on subject matter expertise has softened in replacement for data, and many of the complex linguistics have been supplanted with statistical modeling. This is accomplished by using ML techniques that allow for computers to make decisions from the input it has been given. Depending on the chosen learning algorithm, the computer will use a collection of statistics and probability to learn from the data, such that it can make informed decisions on future unseen data.

Using ML for NLP, begins by representing human language as numerical vectors. There are few ways to do this, all with varying levels of complexity and sophistication. The simplest way is by one-hot encoding the categorical text data into numerical form, constructing what is called an embedding. One-hot encoding is one of the simplest approaches for representing text in numerical form. However, a level of consideration is warranted as the chosen method to convert text data to numerical vectors can lead to extremely large, sparse matrices unsuitable for training. Also known as the "curse of dimensionality", too many training features lead to large number dimensions, causing noise and reduction in performance when training the model. Therefore, prior to any training, much attention is placed on feature extraction, where the objective is to extract the most useful features from the text. Useful features are those that capture important components of the text data features and are of enough quantity that the model can learn from.

There are many methods to feature extraction in NLP. Arguably the simplest, is the bag-of-words (BOW) representation. BOW is a vector containing counts of every word in a sentence. This type of approach is extremely simple in that important features are considered to have high frequencies. In practice however, this may lead to obscure features since stop words like "the", "it", "to", etc. can overpopulate the vector space. More common approaches are Term Frequency (TF) and Term Frequency-Inverse Document Frequency (TF-IDF). These approaches also calculate frequencies but undergo weighting schemes for better representation. TF calculates the frequency of the word in the document, then divides by the total number of words in said document. TF-IDF takes the logarithm of the ratio between total number of documents and the number of documents that contain the word. The latter has shown to be most popular for classification tasks since it quantifies the importance of a word in the entire collection of documents, thus extracting most relevant features from the text for training.

With extracted features, a machine learning algorithm is chosen for training the classifier. There are many models that can be used for text classification. Arguably the simplest and most common is the family of Naïve Bayes (NB) algorithms. NB is a common class of ML algorithms that excels on text classification tasks by implementing Bayes Theorem. It calculates the conditional probabilities of two events based on the probabilities of occurrence of each individual event. The NB algorithm is considered "naïve" because it operates under the assumptions that all features are unrelated, and each feature contributes equally to the target outcome. This assumption makes NB a favored text classification algorithm as it can operate quickly on both large and small datasets and its performance is mainly dependent on the features extracted from the unstructured text. However, a caveat to the simplicity and speed is loss of all context and semantics of the text input. Even with this, NB is regarded as a basic but powerful classifier for text data and according to Kamran Kowsari et al., serves as the baseline of many papers regarding classification (Kamran Kowsari, 2019).

There are many variations of NB classifiers, each utilized for different types of features. For example, Gaussian NB is used when features are discreet, Bernoulli NB uses features that are of Boolean type, Multinomial NB is popular for features that follow a multinomial distribution and Complement NB is popular with imbalanced datasets. Multinomial and Bernoulli variations are frequently used in many text classification tasks but significantly differ in their approaches. Gurinder Singh showed that Multinomial variations are highly dependent on the frequencies of a term in a document, where Bernoulli models rely on knowing if a term is present in the considered document or not (Gurinder Singh, 2019). Moreover, Charu Aggarwal explains that Bernoulli performs well when working with short documents and non-sparse representations with respect to a small lexicon (Aggarwal, 2018). In instances where target classes are imbalanced, that is, there is an uneven distribution of outcomes, Jason Rennie demonstrated that Complement NB models outperform Multinomial models (Rennie, 2003).

Due to the speed and versatility of NB, their utility have been used extensively across various industries and applications for classification tasks. For instance, Lizhong Xiao utilized the TF-IDF extraction method and a NB classifier to classify patent texts into 4 security domain related categories (L. Xiao, 2018). Using 12,000 security patent documents for training and testing, the final classification model evaluated accuracy, recall, and F1-Score at 93.9%, 93.6%, and 93.7%, respectively. Priyanka Harjule et al. showed that multiple classifiers including NB performed well when classifying highly informal Twitter data (P. Harjule, 2020). Trained on over 1.6 million tweets, the performance of NB was like those observed in other classification models like support vector machines and recurrent neural networks. Furthermore, it is known that tweets are highly unstructured, as many tweets do not follow grammar rules and can have misspellings. In RBS, many rules would need to be applied to process this type of data, where the NB classifier bypasses the need for explicit instruction.

Recognized for its speed and simplicity and regarded as the baseline model, NB serves as an exceptional candidate for classifying MORP text data. The probabilistic nature of NB, along with feature extraction methods like TF-IDF make it a great machine learning tool for classifying NPP outages reported in MORP. Like tweets, the reported outages in MORP are short and occasionally contain misspellings. TF-IDF feature extraction can provide a focus on important systems and components while lowering the priority for irrelevant terms. Furthermore, MORP is a heavily imbalanced dataset, with more low-level outages documented than severe, the Complement NB classifier is best suited for classifying the outages appropriately.

As common as NB is for text classification, there are some caveats to its implementation, particularly in the feature extraction process. Popular Python ML libraries like Scikit-learn provide built-in TF-IDF extractors called tokenizers. These tokenizers are trained with generic text data, so they can

automatically recognize and break up input text sentences into small components for training a classifier. Like the problems recognized in RBS, the generic tokenizer may not have the capability to recognize NPP components like “steam generator”. Instead, it will be broken up into “steam” and “generator”, slightly altering the accounting of keyword frequencies. For example, “turbine generator” and “steam generator” will yield two counts for “generator”. Addressing this issue is outside the scope of this research as it will require training a custom TF-IDF tokenizer. Lastly, NB classifiers do not retain semantics or context of text inputs. TF-IDF embeddings lose all sense of context due to the statistics of a text input.

3.3 Deep Learning with Bidirectional Encoder Representations from Transformers

With sufficient data, machine learning and deep learning architectures have bypassed the requirements for explicit rules when constructing NLP applications. However, as discussed in Section 0, traditional ML classifiers like NB are highly dependent on quality word embeddings for good performance on classification tasks. Even if the features are of high quality, semantics of the text content are completely lost in the process. Jacob Devlin and researchers at Google were able to close this gap and significantly advance the field of NLP by designing the DL transformer-based model, BERT (Jacob Devlin, 2018).

BERT is the encoder stack of a transformer model. It is constructed from various neural network architectures, all working to develop sophisticated, advanced contextual embeddings. Unlike previous NLP models and methods, BERT leverages an attention mechanism to process text sequence bilaterally. By processing information bilaterally, BERT learns context across the entire sequence, allowing it to store positions of the input while utilizing parallelization. In contrast to word embeddings, BERT closes the gap in semantics by generating multiple vector representations for the same word based on the context surrounding the word. By doing so, BERT generates embeddings that capture the words across varying contexts, retaining the knowledge of the entire sentence. These advanced embeddings are called Contextual Embeddings, which contain sequence-level semantics allowing for encoded knowledge to capture polysemous ambiguation. For example, the use of the word “Bank” can be used in context of a financial institution or to describe the side of a river. This is challenging to address in traditional ML methods.

Most real-world applications of NLP utilize a state-of-the-art Transformer based language model and Transfer Learning for NLP tasks. Language models like BERT require an immense amount of data to train. BERT itself was pretrained on a 3.3 billion word text corpus consisting of data from BookCorpus, text corpus and Wikipedia (Jacob Devlin, 2018), requiring a 4 day training process with

an energy consumption of 1,500 kWh (Strubell, 2019). Obviously, it is highly impractical to continue retraining these language models for different applications. To address this crucial issue, Transfer Learning is a method applied to use what has been learned in one setting as an exploit to improve generalization in another setting (Ian Goodfellow, 2016). This method allows BERT to be continually reused as a "starting point" for different models on various tasks.

Explicitly, the "starting point" is an abstract term commonly used for referring to a pre-trained language model that can be deployed for downstream NLP tasks. At a high level, BERT viewed as an off-the-shelf tool for performing NLP tasks. Pre-training is the act of training a transformer-based model with specific tasks for it to learn optimal weights throughout the deep layered architecture. Depending on the tasks for training, the end of the training phase results in pre-trained weights that serve as the basis of learned knowledge. For example, BERT was pre-trained by assigning it to perform two unsupervised learning tasks with unlabeled data: Masked Language Modeling (MLM) and Next Sentence Prediction (NSP) (Anna Rogers, 2020; Jacob Devlin, 2018). With these tasks, in addition to the large amount of quality training data provided in the 3.3 billion word text corpus, BERT can be trained so its weights can be adjusted to incredibly vast scenarios and examples in which the English language was used. In summary, when enough pre-training has been performed, the resulting weights of the model will have extensive experiences of the English language, thus reflecting syntactic, semantic and world knowledge.

The weights representing off-the-shelf BERT are representing learned patterns from the corpus it was trained on. Weights from pre-trained BERT can be utilized immediately for many NLP tasks, including text classification, but like other NLP approaches, may be limited for specific domains However, unlike ML and RBS methods, BERT can be "fine-tuned" with specific domain content, allowing BERT to gain insight into the domain being worked with. Fine-tuning is a required step in the training phase of BERT to adapt the parameters in BERT to the specified domain for the classification task. More explicitly, a single additional layer is added on the final layer of pre-trained BERT neural network. When BERT has been fine-tuned, a final layer is added to the encoder model which is responsible for adapting parameters for classification. Researcher Anna Rogers explains that final layers added to BERT are mostly task-specific, where the bulk of learning general linguistic patterns occurs in early layers (Anna Rogers, 2020). BERT has truly been revolutionary in NLP applications as it solved many issues capturing semantics by learning from contextual representations. Contextual embeddings have greatly reduced the dependency on investing significant time and tedious methods used in traditional pre-processing practices.

Transformer models like BERT and other large language models are considered the current standard for NLP applications, but it does come with a new set of complications. Just like any DL approach,

BERT is a complex tool with simultaneous calculations occurring at once, making traceability of the numerous neural networks in operation difficult. Secondly, the high dependency of pre-processing and feature extraction is traded for the need of domain-specific text data and computational hardware. This work will limit this scope of work by utilizing documented outages in MORP for fine-tuning, as opposed to using external sources.

4 Natural Language Processing in Nuclear Power

Past research in NLP and text mining within the nuclear power domain has been relatively bare compared to the long operational history of the NPP industry. However, in the last two decades there have been significant strides in the use of natural language for extracting insight and gaining deeper knowledge. For example, Yanhua Zou used event reports for identifying causal factors of human errors for a correlation analysis that included clustering and association rule mining (Zou, 2018). Jooyoung Park identified and extracted the relative importance of performance shaping factors from investigation reports of NPPs (Park, 2017). In 2015, Justin Pence and researchers at University of Illinois Urbana-Champaign, built a big data theoretic approach for the quantification of organizational failure mechanisms (Pence, 2015). Written documents and text served as the sources of data providing a more realistic and plant-specific estimation of human error. These research efforts made advancements to gain insight in various reported events, however they do not explicitly define a framework that utilizes past outage data for characterizing future outages.

In more recent years, researchers have begun leveraging the abundance of relevant text data with newer NLP techniques. For example, Yongqing Guan (Yongqing, 2016) identified that certain nuclear quality assurance management activities could be constructed into NLP tasks. Quality assurance activities involving event investigations, knowledge management and workflow control, require work that that can be automated by NLP tasks. It is important to note that the methods employed in this research applied traditional ML practices and did not rely on a large language model such as BERT. Nevertheless, the researchers developed two models to perform an event classification task. One model, known as Label-Latent Dirichlet Allocation to perform supervised learning tasks and the second model is a common ML algorithm used for classification know as Support Vector Machine (SVM). The origination of the input text data came from State Nuclear Power Engineering Co. (SNPEC). It was not discussed in detail, but the SNPEC organized and maintains a database responsible for tracking equipment and their related issues for quality assurance purposes. The models were given documented events such as “The threaded pipe end of injection hole of motor trailing edge was broken” and would classify the event into one of twelve categories. Each category was a “topic” the document was associated with. Both models are trained on manually labeled datasets and tested on a “held out” test dataset. This Is a very common procedure known throughout the industry as “cross-validation”. Both models demonstrated the applicability to generalize with reported precisions above 75%, where the SVM reports a 87% precision. Unlike a fine-tuned BERT model, both SVM and Label-LDA do not take semantics of the input into consideration. In other words, these traditional models relied strictly on statistical techniques to demonstrate the development of a model generalized

for quality assurance tasks in the nuclear domain. This implies that in the case of text data within the nuclear domain, semantics of a sentence and word use may not be a required feature for text classification. It can be hypothesized that the highly technical nature and safety standards implemented by the nuclear industry results in unambiguous text data and communication. Another important element of this research is the extensive rigor required to develop their structured dataset. Researchers reported manually labeling every document instance of both the training and test set, resulting in over 1000 labeled datapoints.

In more recent research, there has been a steady increase in identifying ways to incorporate modern NLP techniques in the nuclear industry. Within the past decade, the opportunities with both abundance of digital text data and modern NLP methodologies have opened doors to new solutions and improvements that vary across different sectors. For example, Electric Power Research Institute (EPRI) explored a variety of NLP based projects (Mirzazad, 2019). These projects ranged from assisting customers by managing their demand during periods with high Time-of-use rates (Clarin, 2019) to better classifying jargon filled text reports from field workers into correct IEEE 1782 outage codes (IEEE, 2014) in efforts to reduce labor hours (Lewis, 2019). Until recently, many national laboratories have been making efforts to capitalize on text data in the nuclear domain. Researcher Sai Zhang at INL provided a presentation demonstrating the preliminary progress on a large framework that analyzes free-text reports from NPP operating experience data for estimating risk model parameters. Technical documentation discussing the methodologies of this research could not be found, possibly due to the early stages of its progress. However, at a high level, the presented research aims to develop a “causal network” which represents event initiation and propagation. This network will be driven by free-text event reports (i.e., Licensee Event Reports) and leveraging NLP techniques that can automatically identify causal relationships. This initial research appears to be a tool or precursor to the overall vision of the end project, where the end project looks to be a massive collection of truly representative data, encompassing all past NPP operation and maintenance activities. The clear use of traditional NLP methods is demonstrated in the causal relationship identification task, where keywords are identified, and relationships are extracted. One interesting component of this presentation is the use of synthetic data as a contributor to the massive, envisioned database. Synthetic data is a technique typically deployed for training a ML model to learn rare scenarios or to adapt to specific domains (Nikolenko, 2021). It is hypothesized that the use of synthetic data in this research may imply a low abundance of data that reflect very rare events. Synthetic data shown to be representative of real plausible scenarios can supplement this missing knowledge.

5 Monthly Operating Report Database

Fundamentally, text classification is a supervised learning task that involves assigning predefined categories or labels to a piece of text, based on its content. Assignment of text inputs is managed with a chosen algorithm, defining how the model will learn and make predictions on new unseen text data. The algorithm for text classification is chosen based on the available data, attributes, and its limitations. This section will introduce the limitations and challenges associated with the MORP database and will illustrate the methodology for selecting training features and choosing an appropriate training model.

5.1 Data Exploration

Between 1996-2016, the nuclear industry experienced a total of 2093242.92 outage hours, not including the hours spent in refueling. Of these outages, roughly 317240 hours resulted from forced outages caused by Equipment Failure and Maintenance or Testing. The distribution of non-refueling outages recorded in MORP are shown in Figure 5-1 and the outages per docket is provided in Figure 5-2.

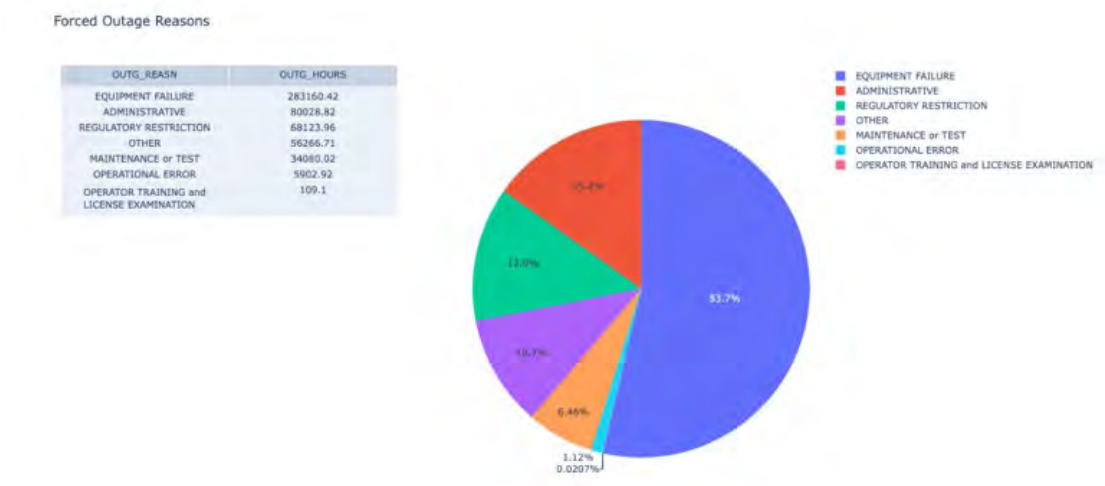

Figure 5-1: Outage Hour Distribution

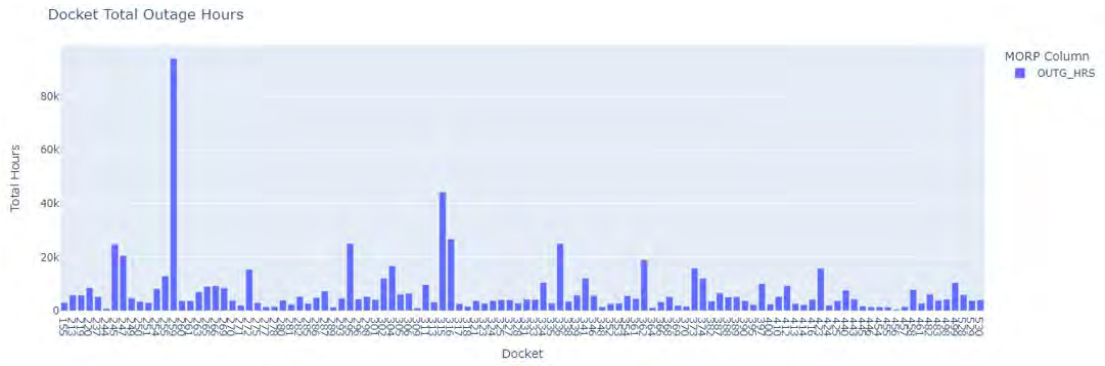

Figure 5-2: Outage Hours by Docket

5.2 Dataset Attributes

The following subsections outline the attributes of the MORP dataset and the origin of the features used for classification. These attributes generally describe the nature of the data and reflect nuclear power domain characteristics.

5.2.1 Summary Column

MORP contains raw text data of summarized outage reports located in the *DESCRIP* column. These reports are very short and only provide high level information about related components, systems and if applicable, reason and solution for the outage. This can be beneficial for training speed and simplicity, but because of such small reports, it is important to identify the possible ways context can be expanded on and noise can be reduced.

5.2.1.1 Acronym and Operator Codes

Majority of the reports contain acronyms that are not explicitly defined. However, in many of the reports with ill-defined acronyms, there is sufficient detail where one can manually deduce the expanded form. Similarly, many of the systems, components, and operations are referred to as codes that are not easily identifiable unless significant research is done through parsing of LER's or other publicly available documentation. Due to the short reports throughout MORP, these are items that represent valuable context that cannot be neglected.

5.2.1.2 Duplicated Reports

Some outages exceeded the maximum time allowed of 745 hours, for the *OUTG_HOURS* data field. When this occurs, the same text input was repeated in a new entry, but the reported *OUTG_HOURS*

may differ from the original outage. This could lead to poor classification performance since the model would learn from text inputs associated with serious outages but classified into less severe class labels.

5.2.1.3 Refueling Outages

Reported refueling outages makeup 61.7% of the MORP dataset. These outages do not contribute any important knowledge to the training process and must be removed from the training data.

5.2.1.4 Vernacular and Taxonomy

The nuclear industry domain has many unique names for system components that should be retained throughout the text data. For example, “Steam Generator” must be retained as a single representative token, instead of being mistakenly recognized as “Steam” and “Generator”.

5.2.2 Outage Hours Data

Unlike the text data, the class labels that the model will be supervised with, is the reported outage hours shown in *OUTG_HRS* column. Because the values of this column span anywhere between 1 and 744 hours, classifying documents into these exact values would lead to poor learning since the model would not have enough training data to learn all the class labels. Therefore, the *OUTG_HRS* column requires binning into appropriate levels of severity.

5.3 Limitations and Modeling Approach

To our knowledge, no other classification on NPP outage severity has been found in current existing literature. Due to the vast challenges and obstacles, one may investigate, this NLP task is subjected to significant scope creep, therefore this research will adhere to fundamentals and data applicability. More concretely, the ML models chosen to perform the classification task will be based on the limitations of the MORP dataset.

Since MORP is a small dataset consisting of roughly 4000 relevant short documents, this research will aim to develop a baseline comparison of two models known for successful performance with such limitations: Naïve Bayes (NB) and BERT.

NB has been identified as a basic, initial learning algorithm for text class classification, and shown to thrive with limited data. Although it does not retain semantics, there are different variations of NB learning algorithms, with even more pre-processing techniques and feature representations that can be

leveraged for improved accuracy. The classifier variation to be explored is the Complement NB algorithm, as it leverages a weighting calculation beneficial to imbalanced datasets. The Complement NB will use traditional one-hot encoding methods TF-IDF and BOW

Since the NB method cannot retain semantics or word positions, fine-tuning BERT on MORPs raw and partially cleaned data will also be explored.

6 Theory

Naïve Bayes differs significantly from transformer-based models like BERT. Fundamentally, Naïve Bayes operates on probabilities without any dependence on surrounding context, making it easy to implement. Contrary to Naïve Bayes, BERT is capable of capturing context but requires many complex subprocesses in its architecture. The performance of Naïve Bayes models is highly dependent on the quality of input features, where BERT models are limited by the quality and availability of data it was trained on for a specific domain and task. This section documents the theory behind text classification using both Naïve Bayes and BERT.

6.1 Vector Representation: Complement Naïve Bayes

Prior to diving into the theory of the Complement Naïve Bayes algorithm, it is important to address how natural language is inputted into a computer. As discussed in Section 0, traditional ML methods achieved more versatility than RBS methods by representing text data as a numerical representation.

There are many practices and methods used for representing text data as numerical vectors, where some work to capture more meaning of the natural language than others. In this research, both BOW and TF-IDF are explored for training the Complement Naïve Bayes classifier.

BOW is a commonly used method to represent text input as a vector containing a binary value of words existing in a document. This is accomplished by first identifying all important words (feature) in all documents. Then stepping through each document and constructing a matrix of whether an important word is present in a document. This results in a large table of 1's and 0's, which will be used to represent the text input. For example, given input sentences like those in Table 6-1, a matrix of training vectors can be built. An example of the resulting matrix is shown in Table 6-2.

Table 6-1: Input Sentences

Doc ID	Input Sentence

Doc1	Small shrimp
Doc2	Weak shrimp
Doc3	You are a bay shrimp
Doc4	I am a tiger prawn
Doc5	Water sucks, Gatorade is better
Doc6	Gatorade sucks, water is better

Table 6-2: Resulting Vector Representations

	bay	better	gatorade	prawn	shrimp	small	sucks	tiger	water	weak
Doc1	0	0	0	0	1	1	0	0	0	0
Doc2	0	0	0	0	1	0	0	0	0	1
Doc3	1	0	0	0	1	0	0	0	0	0
Doc4	0	0	0	1	0	0	0	1	0	0
Doc5	0	1	1	0	0	0	1	0	1	0
Doc6	0	1	1	0	0	0	1	0	1	0

Since BOW vectors are represented in binary, there exists no way to discern the importance of the word. To get around this, a common method to apply is the TF-IDF approach, which evaluates how relevant a word is to a document in a collection of documents. Python library `Sklearn` calculates this weighting strategy with the following presented below.

$$TFIDF = TF * IDF$$

Where,

$$TF(t) = Number\ of\ times\ term\ t\ exists\ in\ the\ document$$

$$IDF = \ln\left(\frac{n}{df(t)}\right) + 1$$

Where,

$$n = Total\ Number\ of\ Documents$$

$$df(t) = Number\ of\ Documents\ in\ which\ term\ t\ appears$$

For example, the TFIDF value for “prawn” in Doc4 would be calculated by the following,

$$TF = 1$$
$$\text{IDF} = \ln(6) + 1$$
$$TFIDF = TF * IDF = 2.79$$

Applying this for every term in each document results in a matrix of representative vectors used for training. Table 6-3 shows the TFIDF representations calculated from the initial example. By applying TFIDF, the vectors are transformed to have a weighting strategy, assigning more importance to different features within the input text. Although the TFIDF highlights important words, it still vastly limited in the ability to retain context. For example, Doc5 and Doc6 are clear opinions about Gatorade and water. Since semantics are lost, it is not clear which document has a preference on Gatorade or water. Thus, these documents will appear similar when trained with Naïve Bayes models.

Table 6-3: TFIDF Vector Representations

	bay	better	gatorade	prawn	shrimp	small	sucks	tiger	water	weak
Doc1	0.00	0.00	0.00	0.00	1.69	2.79	0.00	0.00	0.00	0.00
Doc2	0.00	0.00	0.00	0.00	1.69	0.00	0.00	0.00	0.00	2.79
Doc3	2.79	0.00	0.00	0.00	1.69	0.00	0.00	0.00	0.00	0.00
Doc4	0.00	0.00	0.00	2.79	0.00	0.00	0.00	2.79	0.00	0.00
Doc5	0.00	2.10	2.10	0.00	0.00	0.00	2.10	0.00	2.10	0.00
Doc6	0.00	2.10	2.10	0.00	0.00	0.00	2.10	0.00	2.10	0.00

6.2 Complement Naïve Bayes

The Complement NB classifier is based on Bayes Theorem, and the assumption that the presence of a particular feature (i.e., words) in a class is unrelated to the presence of any other feature. It is one variation of the NB classifiers, and an adaption of the common Multinomial NB algorithm. However, unlike the Multinomial NB classifier, the Complement NB classifier is better suited for imbalanced datasets as it uses statistics from the complement of each class to calculate the models weights (Jason Rennie, 2003).

The Complement NB classifier computes a probability that a document belongs to a specific target class using Bayes rule shown below.

$$\Pr(c|t_i) = \frac{\Pr(c) * \Pr(t_i|c)}{\Pr(t_i)}, c \in C$$

Where the classifier will classify test document t_i to class c based on the highest computed probability $\Pr(c|t_i)$. The class prior $\Pr(c)$, is estimated by dividing the number of documents that belong to the class, by the total number of documents.

The normalization factor $\Pr(t_i)$, is calculated as,

$$\Pr(t_i) = \sum_{k=1}^{|C|} \Pr(k) \Pr(t_i|k)$$

Lastly, $\Pr(t_i|c)$ is the probability of obtaining a test document t_i in class c and is shown to be calculated as,

$$Pr(t_i|c) = \alpha \prod_n \Pr(w_n|c)^{f_{ni}} = \alpha \prod_n \widehat{\theta_{ci}}^{f_{ni}}$$

Where the count of the word n in the test document is given as f_{ni}. Where Complement NB differs from that of other variants, is how the parameters θ_{ci} or, probability of the word given the class $\Pr(w_n|c)$ is approximated using the training data. Complement NB calculates parameters based on the following,

$$\widehat{Pr}(w_n|c) = \widehat{\theta_{ci}} = \frac{\alpha_n + \sum_{j:y_j \neq c} d_{nj}}{\alpha + \sum_{j:y_j \neq c} \sum_k d_{kj}}$$

Where the summations are over all documents j not in class c, d_{nj} is the count frequency or TF-IDF value of word n in document j and α is the smoothing hyperparameter used to avoid the zero-frequency problem (Andrew McCallum, 1998) and calculated as $\alpha = \sum_n \alpha_n$.
The weights for the decision boundary are then calculated by,

$$w_{cn} = log\ \widehat{\theta_{ci}}$$

$$\widehat{w_{cn}} = \frac{w_{cn}}{\sum_j |w_{cj}|}$$

Then used in the classification rule by assigning the document to the class with the lower complement match with the following,

$$\hat{c} = \arg\min_{c} \sum_{n} f_n w_{cn}$$

Where f_n is the count of word n.

Assuming conditional independence allows for this model to be fast and work well with small datasets. However, assuming conditional independence is what makes NB *naïve* as it comes at the cost of losing contextual relationships. For example, given the following input, "`manual trip of reactor and turbine due to trip of "b" circulating water pump.`"

All word order is lost, and capturing the relationship shared between the `reactor/turbine` with `circulating water pump` is lost.

6.3 Bidirectional Encoder Representations from Transformers

Bidirectional Encoder Representations from Transformers (BERT) is a state-of-the-art NLP model introduced by Google in 2018. Transformer models have addressed many recurring challenges faced in NLP. In (Ashish Vaswani, 2017), the transformer architecture has been shown to solve issues such as capturing long-range dependency problems, efficiency through parallelism, transfer learning, and contextual understanding. At the root of most of these issues is the implementation of an attention mechanism, which is a component in BERT that permits the model to capture contextual dependencies between words in a sentence. This mechanism, along with the bidirectional processing, allows sequences to be examined in both directions, while providing quantified attention scores to individual words and their relationship to surrounding words in a sequence.

At a high level, a transformer is a complex type of DL architecture containing many layers responsible for processing and transforming inputs provided by a previous layer. The primary objective of the transformer is to effectively model and understand sequential data. A detailed image of the full transformer architecture is provided in Figure 6-1.

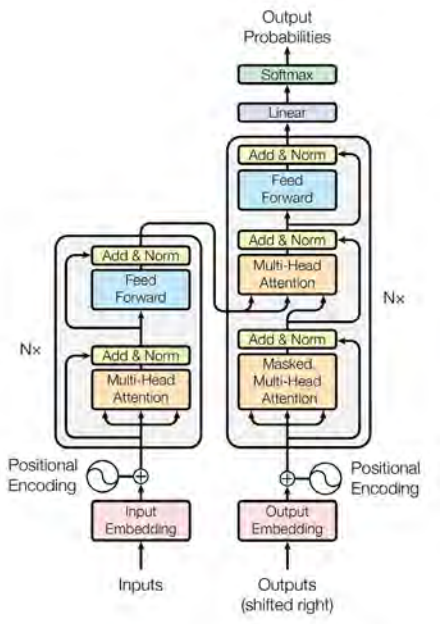

Figure 6-1: Transformer Architecture (Ashish Vaswani, 2017)

It is often easier to view the architecture as two main components: Encoder and Decoder. Figure 6-2 provides a high-level flow chart of the main components that make up the transformer architecture.

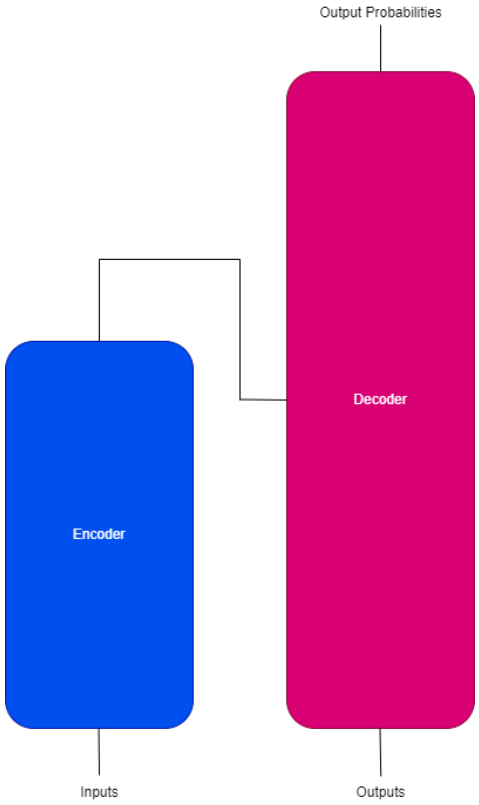

Figure 6-2: Encoder and Decoder Components of Transformer

Depending on the application, the entire architecture is not always required. For example, decoder-only models use the decoder component of the transformer architecture for tasks such as text generation, or summarization, where encoder-decoder models are typically used for language translation tasks. Encoder models can be used by themselves for natural language understanding, information extracting and sequence classification tasks. BERT is constructed from an encoder, which have demonstrated great utility at extracting vectors containing useful information about an inputted sequence. These context-rich vectors can be used "downstream" by adding additional task specific-layers (or neurons) to compute them for a desired outcome. In the context of BERT, (Jacob Devlin, 2018) describes how an initial multi-layer bidirectional Transformer encoder is trained on two unsupervised learning tasks using over 3,000 million words from Wikipedia and BooksCorpus. Since BERT is the model intended for this research, the theory will be reduced to only the encoder of the transformer model.

6.4 Encoders

The purpose of BERTs encoder is to create meaningful contextualized embeddings of an input sequence. Figure 6-3 breaks up the encoder component into two main subcomponents, where the

Transformer encoder lies in the orange box and the required Embedding Layer is shown in the green. This section describes the theory behind the encoder and the architecture for pre-training BERT.

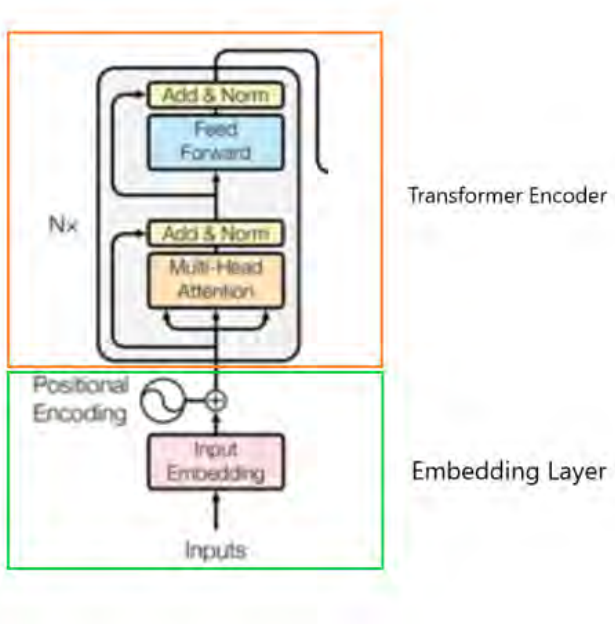

Figure 6-3: Encoder Component (Ashish Vaswani, 2017)

6.4.1 Embedding Layer

Three inputs are required for the Transformer encoder: tokens, position embeddings and segment embeddings. These inputs are created in the Embedding Layer box highlighted in Figure 6-3. Tokens are generated using the WordPiece tokenizer algorithm which uses methods outlined in (Yonghui Wu, 2016) to tokenize words in a sentence. The maximum allowed input length is 512 tokens. If a word exists in BERT's vocabulary, it will be tokenized as a complete word. If a word does not exist in BERTs vocabulary, it will be broken up into subtokens represented as a root word and the residual subwords. The objective of WordPiece is to improve model understanding by reducing the vocabulary size. It can achieve this by leveraging the meaning between previously seen root tokens and subtokens. BERT leverages these words and subwords to easily identify related words that share similar input tokens. This strategy also allows BERT to gain some understanding of unknown words, by using known and previously seen subwords to construct it. Generally, if a word does not exist in BERTs vocabulary, WordPiece will break up the word into subwords prefixed with '##' symbols. If a word does exist in BERTs vocabulary it will not be divided and will be represented as a single token.

In addition to token embeddings, position and segment embeddings are also constructed. Position embeddings are vectors of integers representing the position a token exists in a sequence. This is

important because it will provide BERT positional context to a token such that repeated words are distinguishable. For example, positional information on the nominative singular pronoun – "I" in "I eat chicken therefore I am a carnivore" would be retained. The last requirement created in the embedding layer is the segment embeddings. This vector contains values of 0 and 1, indicating the sentence the token exists in when given two pairs of sentences. This is useful when performing certain training tasks. After these three vectors are constructed, they are summed elementwise to a single input embedding matrix with dimensions $(1, N_{dim}, 768)$ for every sequence, which is fed into the initial encoder layer. In other words, each token is a is represented as a vector with a length of 768. An example of the input tokens, segment embeddings and position embedding are show Figure 6-4.

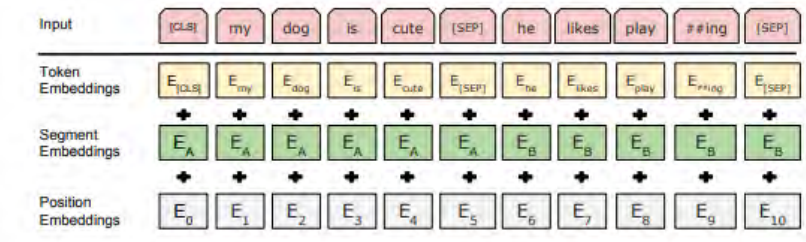

Figure 6-4: Input Representation (Jacob Devlin, 2018)

6.4.2 Transformer Encoder

The Transformer Encoder (or Transformer Block) is a stack made up of N identical encoding layers. Each layer has two sub-layers. The first is a multi-head self-attention layer containing a self-attention mechanism, and the second is a position-wise fully connected feed-forward network (Ashish Vaswani, 2017). Each layer undergoes a summation and normalization used for treating the vanishing gradient problem (Hochreiter, 1998). Figure 6-5 shows the encoder stack with 6 encoding layers, where the bottom encoding layer feeds its output to the next encoding layers.

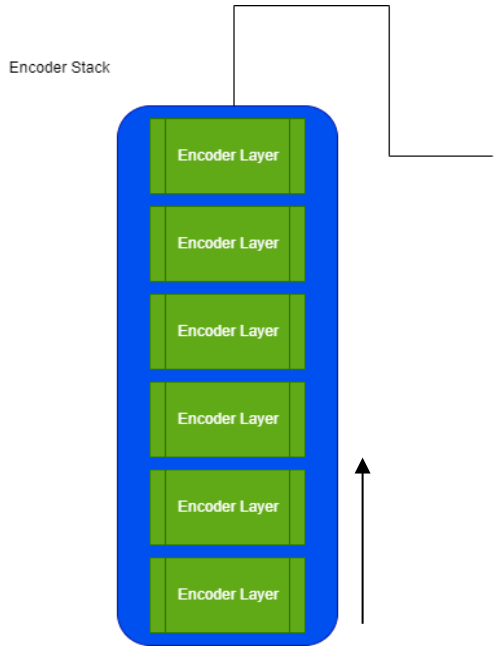

Figure 6-5: Encoder Stack

Each input embedding is sent into the first encoder layer located at the bottom of Figure 6-5. The input embedding is processed through two sublayers, Multi-Head Attention and Feedforward Neural Networks.

6.4.3 Self-Attention

Prior to diving into the Multi-Head Attention, it is useful to discuss the self-attention mechanism that has given Transformers the ability to achieve state of the art performance. Self-attention is a mechanism addressed in (Ashish Vaswani, 2017). The primary objective of self-attention is to understand contextual relationships between words in a sentence. This is achieved by creating a vector with an attention-based score that can be used to quantify how relevant each word is for a given input sentence with respect to itself and other words. Word embeddings are improved by performing an attention calculation with surrounding words in a sentence. The attention function is described as mapping a query and a set of key-value pairs to an output, where the query, keys, values, and output are all vectors (Ashish Vaswani, 2017). The attention function used in training BERT, is referred to as Scaled Dot-Product Attention function and is applied on a set of queries simultaneously packed in to a matrix Q, using matrices keys K and values V and the dimension of the key vectors $\sqrt{d_k}$. The calculated attention score is applied for each input word x_i and used to compute a weighted sum of the tokens indicated as vector J_i shown in Figure 6-6.

$$Attention(Q, K, V) = softmax\left(\frac{QK^T}{\sqrt{d_k}}\right) * V$$

The Multi-Head Attention is a layer that runs 8 self-attention calculations in parallel is shown in Figure 6-6.

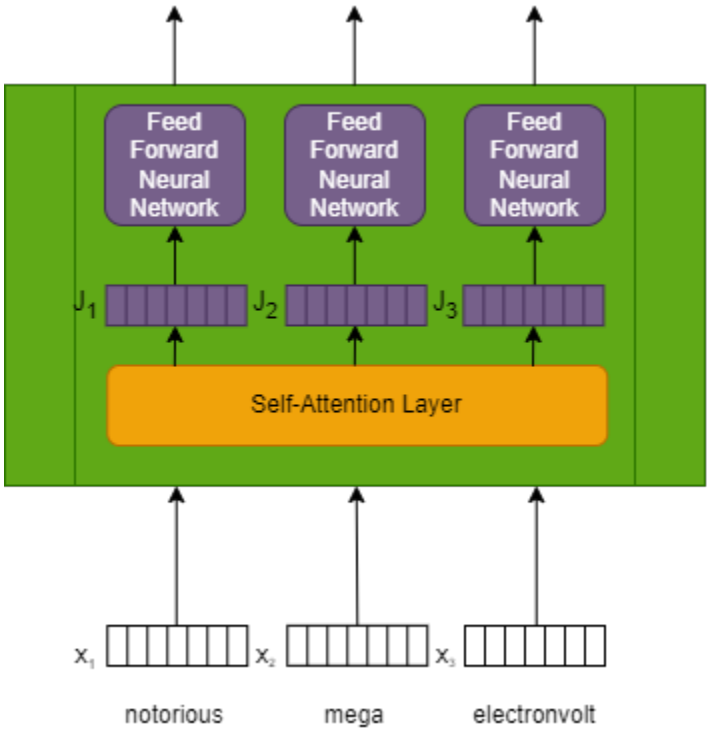

Figure 6-6: Encoder Sublayers

6.4.4 Feed Forward Neural Network

The Feed Forward Neural Network (FFNN) layer takes the self-attention vectors as input and is propagated forward through the network. At each hidden layer, the weighted sum of the inputs is calculated and passed through an activation function, which introduces non-linearity into the model.

6.5 Training Objectives

Using the encoder architecture, BERT was constructed by being trained on two unsupervised learning tasks: Masked Language Modeling (MLM) and Next Sentence Prediction (NSP). In MLM, BERT randomly masks 15% of the tokens in the input sequence and attempts to predict the original masked tokens. The model receives the masked input sequence and generates representations for each token. The goal of this task is to correctly predict the original tokens based on the context of the surrounding tokens, to gain better insight into the relationships between each word. The objective in NSP, is to have

BERT take pairs of sentences as input and learns to predict whether the second sentence follows the first sentence in the original text. Training requires BERT to randomly select sentence pairs from a large corpus and inserts them into training examples. Some of these pairs are consecutive sentences from the original text collection, while others are randomly paired sentences. The model is then tasked to classify whether the second sentence is a genuine consecutive sentence or a random one. Unlike MLM where the goal is understanding word relationships, NSP aims to provide insight into sentence relationships as a broader focus. It was shown in (Jacob Devlin, 2018) that by training on both tasks, BERT showed an overall higher improvement in performance as to only training with MLM. Once a prediction is made for each task, the error is calculated. The error is called a “loss function” which is a difference between the predicted output and the actual output. This error is then propagated back through the network, and the weights used to within the neural networks are adjusted to minimize the error for the next iteration. This process of adjusting weights is called “back propagation”, which is the main concept used for learning. Back propagation is typically done using a gradient descent optimization algorithm.

6.6 Transfer Learning

After sufficient training, the current state of BERT contains knowledge from its previous tasks to be used for many NLP techniques. This state of BERT is referred to as a pre-trained language model where its weights represent knowledge obtained from the dataset used for MLM and NSP tasks. Using transfer learning, these weights can be loaded into any BERT model and used as an “off-the-shelf” language model for performing many tasks like text classification and sentiment analysis.

7 Methodology

Many outages within MORP are irrelevant to the scope of this research problem, therefore many are removed before any text pre-processing. All outage reasons and number of their occurrences are reported in MORP are shown in Figure 7-1.

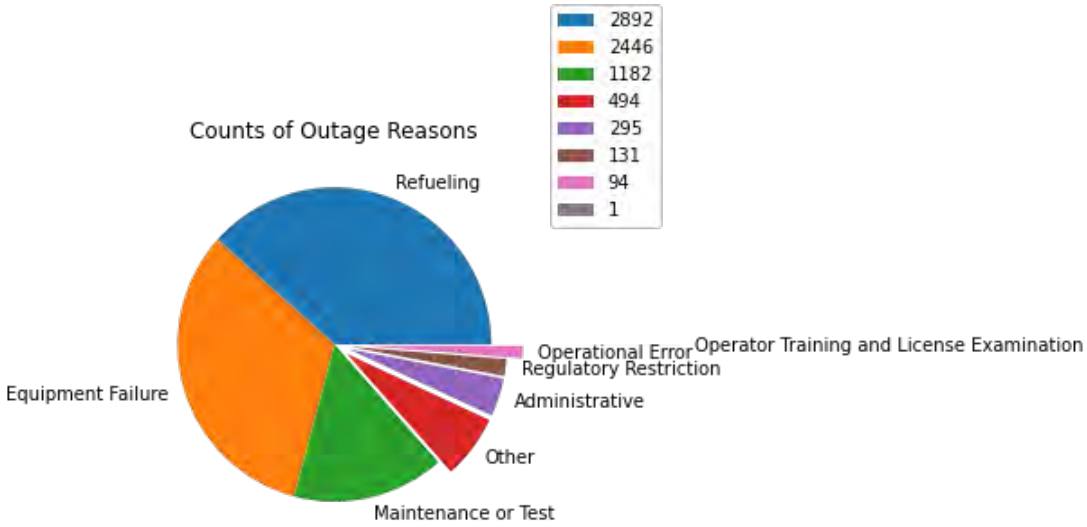

Figure 7-1: Outage Occurrences

Outage reasons that do not provide relevant information about power plant systems or components are removed and not include in the training process. For example, refueling outages, regulatory restrictions, operator training and license examination and administrative outages were removed from the dataset. Therefore, the final training set only contains the following outage reason occurrences shown in Figure 7-2.

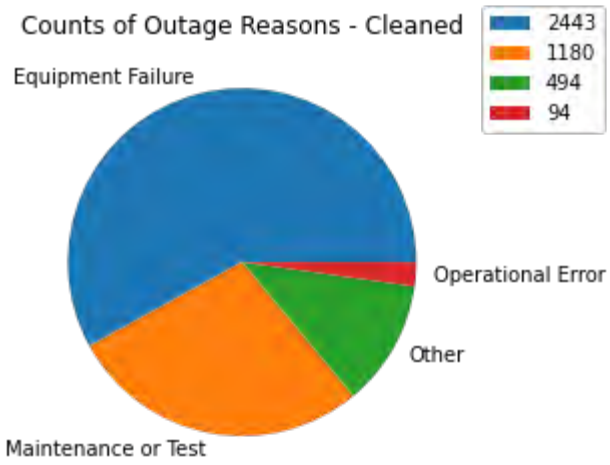

Figure 7-2: Relevant Outage Reason Occurrences

After removing the irrelevant report summaries, the final dataset contains a total of 4,211 summaries which will be split for training and testing.

7.1 Data Preparation

Training NB and BERT for text classification requires two separate approaches for data preparation. Since NB is reliant on purely statistical methods, preserving excessive irrelevant features results in high-dimensionality, overfitting, unwanted noise, and poor performance. Therefore, NB requires training data to contain quality features and the removal of irrelevant ones. In contrast, BERT performs better by utilizing the context and semantic relationships, therefore much less pre-processing of text is required.

All data preparation for both methods is completed using a combination of both manual cleanup and Python modules and functions. Python modules for data preparation can be found in in Appendix A.

7.1.1 Cleaning

MORP naturally consists of raw text data detailing the outage of a NPP. The current state of the raw data itself poses many challenges for training any text classifier. Data preparation for both NB and BERT will require the need to preserve features relevant to a NPP outage, relevant features would be those that best describe the outage. This includes abbreviations, acronyms, plant specific labels, procedural codes, and specific equipment identification numbers. These features communicate system components and equipment beneficial for the algorithm to learn and generalize future outages on.

Preserving and properly representing these features is not a trivial task and requires manual labor to represent them correctly in the training data. For example, representing the contraction “SG” as “Steam Generator” during the training process was preferred, since “Steam Generator” could be tokenized better than “SG”. The following subsections provides and overview of the existing challenges in data preparation.

Since NB do not retain semantics or context, punctuation and stop words such as “the”, “to”, etc. are completely removed from the text input, but retained in the BERT methodologies.

7.1.1.1 Acronym Expansions

Acronyms are heavily used throughout the documented outage reports in MORP. Majority of these acronyms describe system components and power plant equipment that serve as valuable, domain specific features for training the classifier. To extract and expand the acronyms, an iterative approach requiring both automated and manual verification was implemented. This methodology operates on the assumption that acronyms without being explicitly defined are commonly used throughout the industry. For example, “SG” is an acronym not explicitly defined in most document outages but is commonly known to represent “Steam Generator”.

When an acronym was correctly identified, it was appended to the text file “masterExpansions.txt” where the contracted acronym and its expanded form are stored and used in the pre-processing phase to clean MORP. The text file “masterExpansions.txt” can be found in Appendix B.

Some acronyms were explicitly written in their full form prior to contraction. In most cases, the contractions were initially presented within parenthesis, then used freely throughout the remainder of the outage document. For example,“`minor maintenance and surveillance testing. reactor shutdown automatically due to low electrohydraulic control (ehc) pressure while paralleling ehc pumps.`”

These acronyms were first extracted using Regular Expressions (Regex) for entities enclosed in parenthesis then manually verified in the document of origin. In many instances, some acronyms were not readily known and not explicitly defined in the report summary. For example,“`shutdown to repair a crdm canopy weld leak, which was the root cause of a dropped control rod.`

Using Python code, these acronyms were first cross-referenced with defined acronyms listed in (NRC, Nuclear Regulatory Commission, NUREG-0544, Rev. 4, "NRC: Collection of Abbreviations", 1998). However, in some cases (NRC, Nuclear Regulatory Commission, NUREG-0544, Rev. 4, "NRC:

Collection of Abbreviations", 1998) would list more than one definition for an acronym, requiring manual assessment of the acronym used in context of the summary and engineering judgment to choose the best expansion.

If an acronym could not be verified in (NRC, Nuclear Regulatory Commission, NUREG-0544, Rev. 4, "NRC: Collection of Abbreviations", 1998), they were identified in publicly available NRC documentation by manually searching through event reports, emails, or other technical documentation. Once majority of the acronyms were identified, Python code was used to replace the acronyms with their expansions and Regex was used to find and extract all existing 2-4 letter words from each report summary. The same approach was conducted multiple iterations until majority of the 2-4 letter acronyms were identified. The acronyms that could not be identified were recorded in the text file "unknown.txt" shown in Appendix D.

7.1.1.2 Hyphenated Plant Specific Entities and Procedures

Report summaries frequently contained hyphenated plant specific labels and codes, existing as combinations of both acronyms and numerical entities. For example, `"manually shutdown the unit due to a socket weld leak on the rcs pressure boundary upstream of valve sia-v056."`

These objects were extracted using Regex rules that matched any objects with hyphens and alphanumeric combinations. Hyphenated instances like `sia-v056` in the above example are searched across publicly available information. If there was a match, the hyphenated object, expansion, and reference was recorded in text file "reformat.txt", provided in Appendix E. Additionally, the text component of the hyphenated entity was recorded in "hyphensInVocab.txt" located in Appendix C for downstream utility. If there was not a successful match in publicly available documentation, the hyphenated object and its context was manually examined in the MORP report summary and compared against pre-existing acronyms in the "masterExpansions.txt". An engineering judgment was then made to expand the hyphenation into the best representation. One example of this is in the following report summary, `"planned outage for 2rcs-p-1b seal replacement."`

In this example, `"2rcs-p-1b"` is represented as `"reactor coolant system pump"`, since `"rcs"` was previously seen and recorded in "masterExpansions.txt", furthermore, it was common to for seal replacements to exist in context with pump in previous report summaries.

Finally, there were some instances in which hyphenated objects represent compound words or expressions. For example, "`unit 1 due to a failed no-load disconnect switch on the main generator.`"

These were matched based on Regex rules for only containing hyphens and non-numeric characters. The treatment for these types of hyphens was switching them for the non-hyphenated representation, such as "`no load`".

7.1.1.3 Dates

Dates and datetime objects do not contribute any useful information for this application of text classification. Therefore, Regex patterns were used to remove them from all report summaries.

7.1.1.4 Backslashes

All instances of backslashes are removed while splitting the left and right components. For example, in "`edg d5 and d6 not operable per tech specs, due to lube oil/fuel oil problem.`" "`oil/fuel`" is split to "`oil fuel`".

7.1.1.5 Unknown Combinations

There were many instances in which the combinations of letters, numbers, and/or plant entities were not captured with Regex patterns or too challenging to manually identify. These entities were recorded in "unknownWordNumbers.txt" and removed from the corpus.

7.1.2 Labeling

To perform classification, outage hours in MORP need to be mapped from continuous values to binned, discretized representations. This section outlines the challenges and strategies associated with performing this mapping.

7.1.2.1 Class Imbalance

Training the classifier requires the input data to be labeled for its target class, in this case, the target class being outage hours (OUTG_HRS). In its raw form, the reported outage hours in MORP exist as a continuous range of values between 0 and 744 hours. These hours are binned into two groups which correspond to "mild" and "spicy" outages. By splitting the target class into two bins, the classification problem is reduced to a binary classification task, which is suitable for the small dataset with class imbalances.

As discussed in previous sections, class imbalances can result in learning only the majority class, and not capable accurately identifying the minority class. Previous data preparation steps result in pruning over half of MORP to a total amount of 4,211 relevant summaries. Depending on the defined outage hour limit, the resulting dataset could be significantly impacted by class imbalance. For example, defining the hour limit to be 744 hours, results in a dataset where only 8.9% of reported summaries are 744 or more hours.

To obtain the best model, multiple datasets are constructed from a range of hour limits to investigate model performance. The hour limits investigated are $[744, 500, 400, 350, 300, 250]$.

7.1.2.2 Binning Scheme

The maximum outage reported per report summary is 744 hours. However, this does not indicate whether it was a single outage or a continuation from a previous outage. To improve the binning of outage hours, Python code was used to group outages by docket numbers (DOCKET) and sort the outages based on the report period (RPT_PERIOD). After grouping and sorting, outage hours are grouped on whether the outage was a continued outage or not. After grouping accordingly, the total sum was calculated and if the sum exceeded the hour limit, it was given a label of "1" to indicate a "spicy" outage, else it was given a value of "0" for "mild". Information regarding the outage method is provided in the outage method column (OUTG_METH). Definitions for the categorical values reported in the outage method column are taken from (MORP2 Definitions, 2016) and shown in

Table 7-1: Categorical Definitions for Outage Method

Code	Description
1	Manual (normal reactor shutdown or generator offline with reactor critical)
2	Manual Scram
3	Auto Scram
4	Continued (from previous month)
5	Reduced Load (only captured through August 1997)
9	Other (outages that transition within the month to another outage)

Finally, after cleaning and binning outages based on the defined hour limits, datasets are constructed. Figure 7-3 shows the variation of class distributions across changes in severity limits.

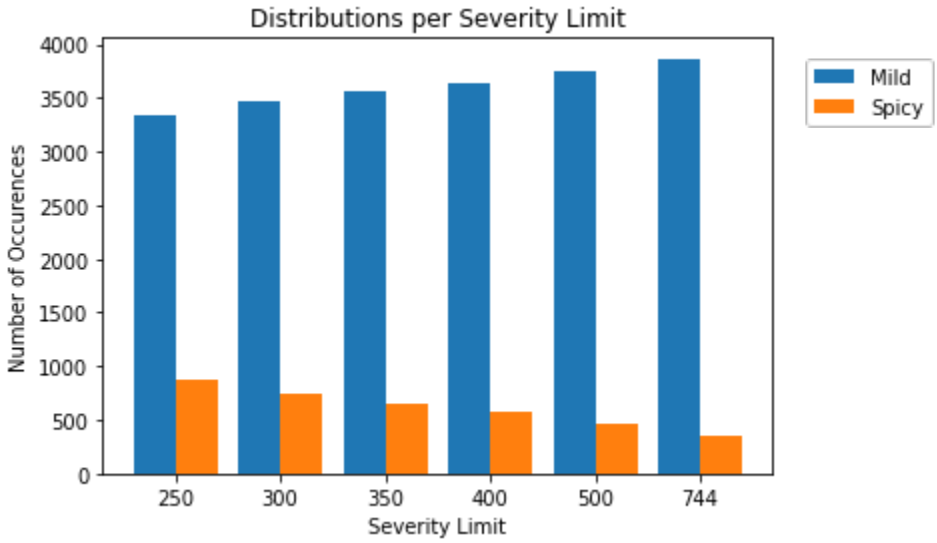

Figure 7-3: Class Distributions for Defined Severity Limits

7.1.2.3 Splitting Scheme

The holdout method is used to evaluate the performance of the trained classifier. This technique involves splitting the cleaned, labeled dataset into three separate datasets which are responsible for training, validating, and testing. 15% of the dataset is "held out" of the training process and used to test the classifier. To ensure unbiased validation of the trained model, the validation dataset was designed to have no overlap or shared report summaries in the training dataset. Similarly, the test dataset was assembled without any overlapping data from both validation and training datasets.

Constructing the test and validation datasets required temporary removal of all duplicate text inputs from the initial dataset, resulting in a dataset with original report summaries. 15% of this dataset is randomly sampled while retaining constant class proportions for both validation and testing datasets. This method of equally partitioning the class distributions is known as "stratifying" and is used to reduce bias in the validation and testing of the trained model. Figure 7-4 shows the resulting class distribution when stratified across training, validation, and testing datasets for defined '744' hour severities. Similarly, Figure 7-5 demonstrates the class distributions after stratifying the '250' hour limit dataset. A summary of class distributions across all datasets is provided in Table 7-2.
Note, during the pre-processing stages, a datapoint from two large BERT datasets were mistakenly not removed and included in the test set.

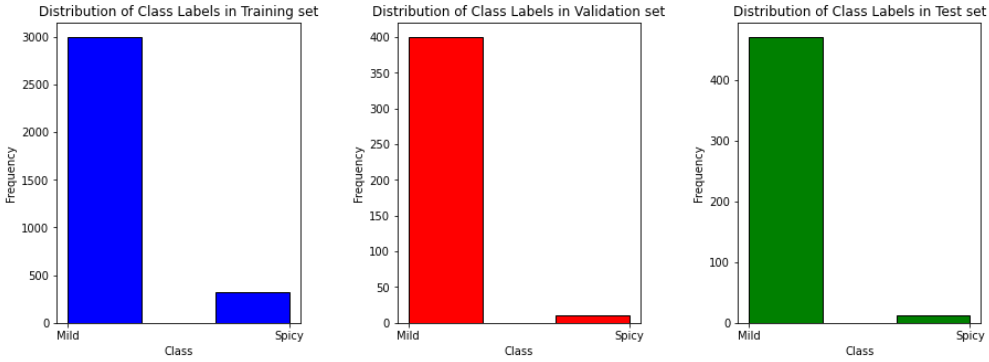

Figure 7-4: Stratification of 744 Hour Limit Dataset

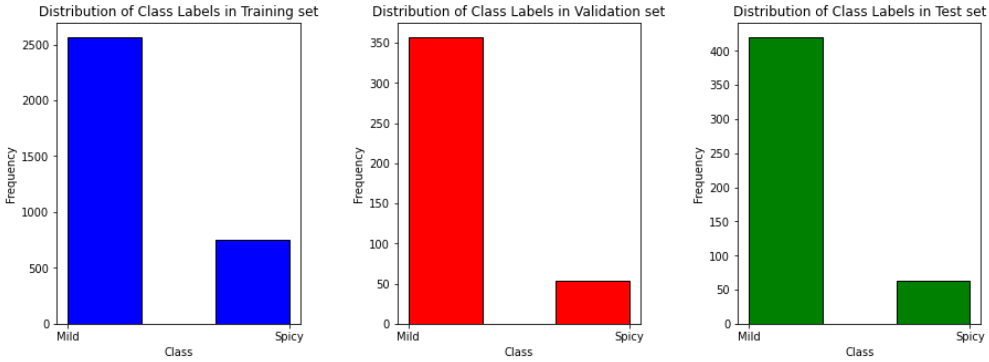

Figure 7-5: Stratification of 250 Hour Limit Dataset

Table 7-2: Summary of Dataset Class Distributions

Hour Limit	Class Counts					
	Training		Testing		Validation	
	Mild	Spicy	Mild	Spicy	Mild	Spicy
250	2566	752	420	63	357	53
300	2673	645	434	49	369	41
350	2742	576	443	40*	376	34
400	2809	509	452	31	383	27
500	2897	421	462	21*	392	18
744	2997	321	471	12	400	10

** +1 Additional Spicy Outage for the BERT classification tasks – Additional data point from non-removal of Unicode string.*

7.2 BERT Models

Two pre-trained BERT models investigated for fine-tuning MORP are BERT large and DistilBERT base. Each pre-trained model was fine-tuned using different batch sizes and max length sequences. The models are fine-tuned separately on the cleaned datasets outlined in Section 0 and unclean datasets, where methods from Section 0 are omitted.

BERT large consists of 24-layers of the transformer architecture, 1024 hidden dimensions, 16 self-attention heads per transformer block and 336M parameters. It was trained on masked language modeling and next sentence prediction tasks on BookCorpus which is a dataset consisting of 11,038 unpublished books and English Wikipedia.

With 336M parameters, BERT large can capture complex patterns, allowing better performance across large diverse datasets. However, the large number of parameters may lead to overfitting when fine-tuning with small imbalanced datasets. The second pre-trained model DistilBERT is used to assess the challenges of small imbalanced datasets. DistilBERT is a smaller, faster version of BERT, with 40% fewer parameters while still retaining 97% of language understanding capabilities by utilizing knowledge distillation in the pre-training phase (Victor Sanh, 2020). Like BERT large, it was also trained on masked language modeling and next sentence prediction using the same datasets.

7.3 Further Training BERT Models

BookCorpus and Wikipedia provide both pre-trained models with data on language used across broad categories. This allows the pre-trained models to have a general understanding of natural language and knowledge used in lots of examples. However, BookCorpus and Wikipedia may not have enough data about the nuclear power domain to effectively learn the language and knowledge associated with nuclear power plants. This domain-specific problem can typically be overcome with fine-tuning when enough data exists in the training dataset. However, after cleaning and splitting the MORP dataset, the resulting 3318 short report summaries may not be sufficient for the pre-trained model to learn and classify on

To address the lack of nuclear power domain data, two additional models are created by further training BERT large and DistilBERT on custom dataset Ntext. Ntext is the “Nuclear Textual” dataset containing textual data related to nuclear domain constructed by (Ayush Jain, 2020). The Ntext dataset was constructed from 7000 internal reports, thesis and research papers in PDF format from the Indira Gandhi Centre for Atomic Research (IGCAR). The sizes of the reports ranged from a couple of pages to a few thousand pages. Much of the unlabeled text data consisted of very old reports, some of which were stored as scanned copies. The reports primarily dealt with the nuclear domain, many of them

explicitly dealing with Fast Breeder Reactors (FBR). Ntext was designed to provide further context to numerous language modelling tasks in the nuclear power domain.

The models were further trained on the original MLM and NSP tasks. MLM involves randomly masking a percentage of input tokens and training the model to predict the original masked tokens. The purpose of this objective is for BERT to learn the contextual representations of words within Ntext.

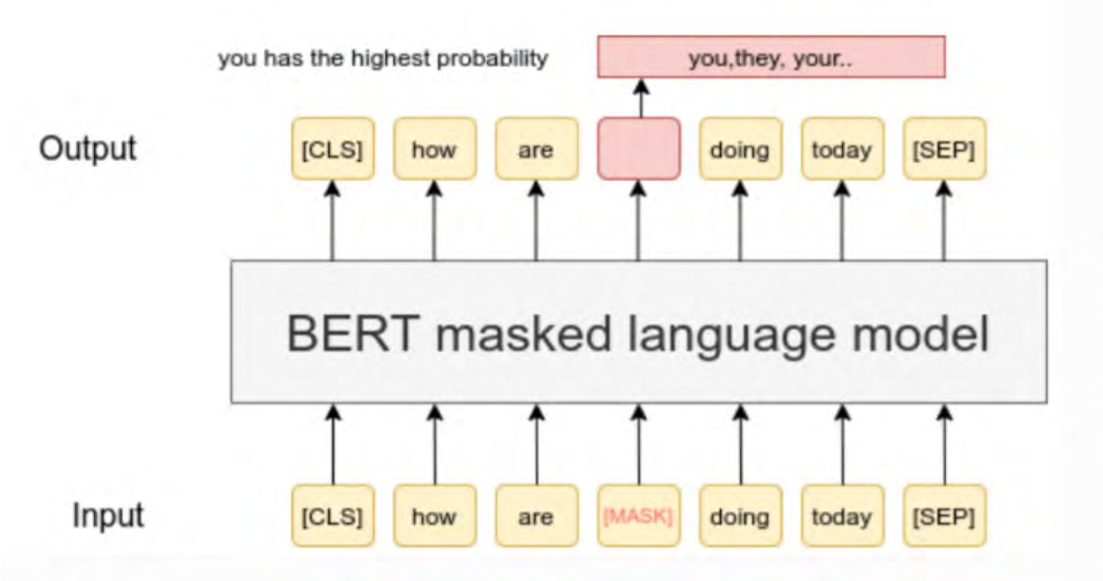

Figure 7-6: Example of MLM (Reimers, 2022)

Additionally, NSP involves training the models to predict whether two input sentences are consecutive or not, allowing for the model to learn relationships between sentences in Ntext.

By further training, these models have exposure to both BookCorpus and Wikipedia, and now scientific literature on fast breeder reactors.

7.4 BERT Training

16 models were trained to examine performance of cleaning MORP and continued training of BERT large and DistilBERT. Hyperparameter tuning is the method of adjusting parameters until the best model performance is achieved. However, the purpose of this research is to establish a base classifier for MORP, thus hyperparameter is outside the scope of this research. The following subsections provides the overview for the configuration of the fine-tuned models.

7.4.1 Cleaned Vs. Uncleaned

To examine the impact of text cleaning, each configured model was fine-tuned on the MORP datasets with and without cleaned text inputs. This was to assess the performance on raw text inputs as opposed to any required cleaning.

7.4.2 Base Models Vs. Further Trained

To evaluate the effects of further training using Ntext for MLM and NSP, a base model and its further trained counterpart were fine-tuned and compared.

7.4.3 Batch Size and Max Sequence Length

The batch size and max sequence length are the only two hyperparameters that are varied across models. The choice of these values are dependent on the available hardware and memory for training in the computational environment, therefore a balance between max sequence length and batch size is necessary to ensure there are no issues with memory.

Batch size refers to the number of training examples that are processed during each iteration of the training phase. More explicitly, it is the number of samples that are simultaneously fed into the model for computation. Each batch is used to calculate the gradients and update model parameters. Large batch sizes are preferred since it reduces the noise in gradient estimations and allows for faster convergence. However, BERT large requires more memory to store the model parameters, therefore a large batch size might not be possible.

Max sequence length represents the maximum number of tokens allowed in a sequence. Both BERT large and DistilBERT base can take a maximum amount of 512 tokens which is approximately 400 words. However, like batch size, larger sequence lengths require more memory. If an input sequence exceeds the max sequence length, the input sequence is truncated.

Many combinations of batch sizes and max sequence lengths were investigated for the less parameter dense model, DistilBERT base. The average length of text inputs in the cleaned MORP dataset is 108 words, with a maximum of 383, therefore max sequence lengths of 128 and 512 were used. Due to the memory requirements, BERT large did not vary in batch and max sequence. A summary of each trained model, including the batch size and max sequence lengths for BERT large and DistilBERT is shown in Table 7-3 and Table 7-4, respectively.

Table 7-3: BERT Large Configurations

Model Name	Model	Batch Size	Max Sequence Length
bert-large-uncased	BERT large	32	128
morpFinalTrain	Further trained BERT Large	32	128

Table 7-4: DistilBERT Base Configurations

Model Name	Model	Batch Size	Max Sequence Length
DistilBERT-base-uncased	DistilBERT base	32	512
		32	128
		16	128
DistilBERT-morp	Further trained DistilBERT base	32	512
		32	128
		16	128

7.4.4 Epochs

An epoch is a hyperparameter that determines the number of times the model will see the training dataset. During the training phase, an epoch refers to a complete pass through the entire training dataset. During the epoch, the model iterates over all the training dataset, calculates loss, and updates the parameters based on the gradients from the backpropagation algorithm. An epoch consists of a number of steps the model takes until the epoch is completed. The number of iterations is defined below.

$$\frac{Number\ of\ Training\ Samples}{Batch\ Size} = Number\ of\ Iterations\ to\ Complete\ One\ Epoch$$

BERT has been shown to fine-tune well with 2-4 epochs and is typically common practice. However, (Tianyi Zhang, 2021) shows that increasing the number of epochs could help stabilize the fine-tuning process when using small datasets. Therefore, it was chosen to use an early stopping technique to prevent overfitting and improve model performance.

Early stopping involves monitoring the model's performance on the validation dataset and stopping the training early if the performance degrades. During model training the model will typically reach a point where it begins to overfit the training data and fails to generalize unseen examples. This is measured by seeing a drop in performance on validation loss after a given number of epochs has been trained. In other words, the goal of training is to minimize validation loss, after a specified number of epochs, if the validation loss has not improved, the weights from when the validation loss was the lowest is returned.

The early stopping parameters used in this research are shown below in Table 7-5,

Table 7-5: Early Stopping Parameters

Parameter	Value	Description
Monitor	Validation Loss	The monitored value that determines early stopping
Patience	10	The number of epochs

7.4.5 Weighting

When considering class imbalance, raw accuracy of a classifier could be misleading because it is possible the classifier performed well at predicting only the majority class. Since all datasets used in this class are imbalanced, the classes are weighted heavily to the minority class, in this case the "spicy" outages. Weights for each class are calculated using the following equations,

$$\frac{1}{Number\ of\ Mild\ Outages} * \frac{Total\ Outages}{2} = Weights\ for\ Mild\ outages$$

$$\frac{1}{Number\ of\ Spicy\ Outages} * \frac{Total\ Outages}{2} = Weights\ for\ Spicy\ outages$$

7.4.6 Learning Rate Decay

The learning rate decay is a method used during training to gradually reduce the learning rate over time. The learning rate controls the speed at which the model learns. Gradually reducing the learning rate provides smaller updates to the model's parameters, helping the model converge effectively and improving performance.

All models utilize a learning rate scheduler which adjusts the learning rate based on the number of training steps. The training steps vary for each model, and are calculated as the following,

$$\left(\frac{Number\ of\ Training\ Samples}{Batch\ Size}\right) * Number\ of\ Epochs = Number\ of\ Iterations$$

The parameters for the learning rate scheduler are shown below in Table 7-6,

Table 7-6: Learning Rate Scheduler Parameters

Parameter	Value
Initial Learning Rate	5e-05
End Learning Rate	1e-05
Power	1.0

7.5 Naïve Bayes Data Preparation

When training a Naïve Bayes classifier, context is not important since there is no way to retain or understand semantic relationships. By pruning as many unnecessary features as possible from the training and testing text inputs, noise from irrelevant features is reduced leading to improved performance. Therefore, all NB models utilize the cleaned datasets develop in Section 0 but with additional cleaning steps. These steps include the removal of all special characters, punctuation and stop words.

7.6 Naïve Bayes Models

Two Complement NB models using tokenizers TF-IDF and BOW were trained with the further cleaned MORP data. The models were trained across a sweep of smoothing parameters α to assess model performance. This sweep contains values of 1.0, 1.2, 1.4, 1.6, 1.8 and 2.0.

7.7 Metrics

Since MORP is heavily imbalanced across all dataset creations, the choice of metrics to evaluate the trained model by are the F1 score, Recall and Precision scores that are calculated from the trained model's performance on the test dataset. The model performance is evaluated by a confusion matrix, which is a table summarizing the prediction results. For the binary classification task, the confusion matrix is constructed by comparing how many actual results match the predicted results. Table 7-7

provides an example of the confusion matrix used to calculate the performance metrics evaluated in this section.

Table 7-7: Confusion Matrix

		Mild	Spicy
True Labels	Mild	True Negative	False Positive
	Spicy	False Negative	True Positive

Predicted Labels

These metrics describe the performance on the ability to identify a truly severe outage. For example, Precision is calculated as,

$$Precision = \frac{Number\ True\ Positives}{Number\ of\ True\ Positives + Number\ of\ False\ Positives}$$

Precision can be interpreted as, out of everything that has been predicted as 'spicy', precision counts the percentage that is correct. This means that a model with high precision may not find all the 'spicy' outages, but the ones that were classified as 'spicy' are most likely to be correct.

Similarly, recall is interpreted as, every outage that is truly considered 'spicy', recall describes how many the model successfully found. A model with high recall demonstrates that is can successfully identify all the 'spicy' cases, but in the process, misclassify 'mild' cases as 'spicy'. Recall is calculated as the following,

$$Recall = \frac{Number\ True\ Positives}{Number\ of\ True\ Positives + Number\ of\ False\ Negatives}$$

Since there is a trade-off between precision and recall, the F1 score combines the precision and recall into a single metric represented as the harmonic mean and commonly used as the standard metric using imbalanced datasets. F1 is calculated by the following,

$$F1 = 2 * \frac{Precision * Recall}{Precision + Recall}$$

A high F1 score reflects high scores in both recall and precision. In other words, a high F1 score reflects the ability to capture majority of 'spicy' cases with a high likelihood that it is a truly 'spicy' outage.

7.8 Training Computational Environment

All models were trained using the A1000 GPU on Google Colab. The Python programming language was used throughout the entire analysis and pre-trained models; BERT large uncased, DistilBERT base uncased are obtained, further trained, and fine-tuned using the HuggingFace transformers library (HuggingFace, n.d.).

8 Results

Top performing classifiers on Precision, Recall and F1 for each outage hour data split are provided in the following subsections. Due to the imbalanced dataset splits, there are fewer number of available testing points as the hour limit binning increases. For example, Table 7-2 shows that models binned at the 744+ hour limit only have 12 "Spicy" outages to test on. Likewise, the 500+ hour binned outages only have 21 data points to test against. It should be noted that a small test sample size does not fully represent all real-world scenarios, especially in complex systems within nuclear power plants.

To quantify the uncertainty in metrics, the Wilson Score Interval is used to construct confidence intervals with 95% confidence. The Wilson Score Interval is used to approximate the confidence interval for a sample proportion in a binomial distribution. Moreover, it provides a range of values with 95% confidence that it will likely contain the true metric. As a rule of thumb, the Wilson Score Interval becomes less reliable when proportions are less than 30 samples. Therefore, metrics from outage hour splits of 400+ should be regarded as preliminary as further treatment in their evaluation is required. The equation for calculating the Wilson Score Interval is shown below.

$$\frac{\hat{p} + \frac{z^2}{2n} \pm z\sqrt{\hat{p}(1-\hat{p}) + \frac{z^2}{4n^2}}}{1 + \frac{z^2}{n}} = Upper\ Bound, Lower\ Bound$$

where,

$\hat{p}$ is the observed metric

n is the test sample size

z is the Z-score corresponding to the desired confidence level

In addition to the Wilson Score Interval, the width and mean of the interval is reported, along with the total average and standard deviations of all model performances. These figures of merit are considered when recommending the best model for specific metrics.

8.1 Top Precision Scores

Precision scores for every model in each outage severity split are provided in Figure 8-1 through Figure 8-6. The top Precision metrics are observed in transformer-based models for each hour limit split. As the outage severity limit is softened, NB with TF-IDF feature extraction tends to get more competitive, outperforming some transformer-based models and BOW feature extraction techniques.

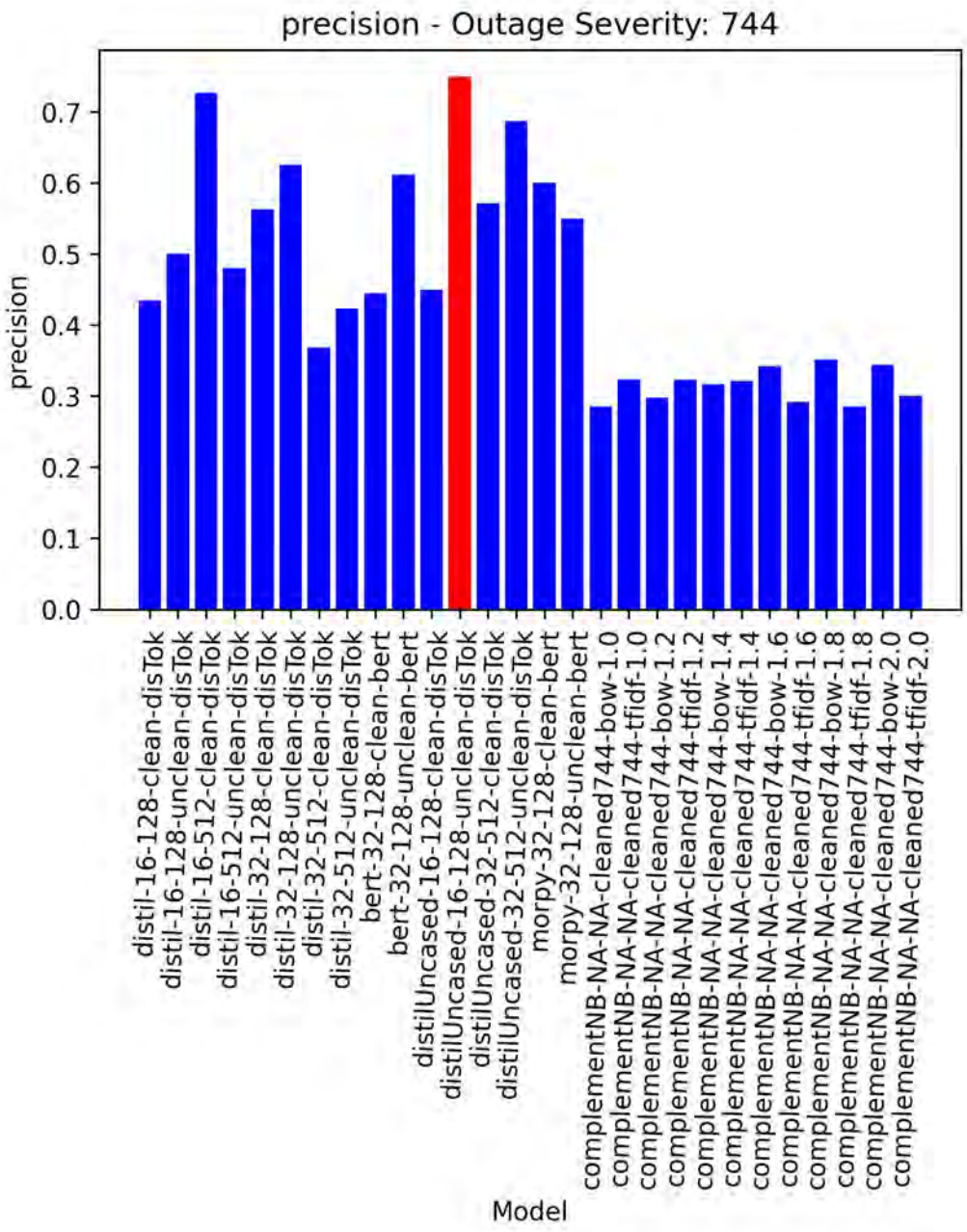

Figure 8-1: 744+ Hour Limit Precision Scores, All Models

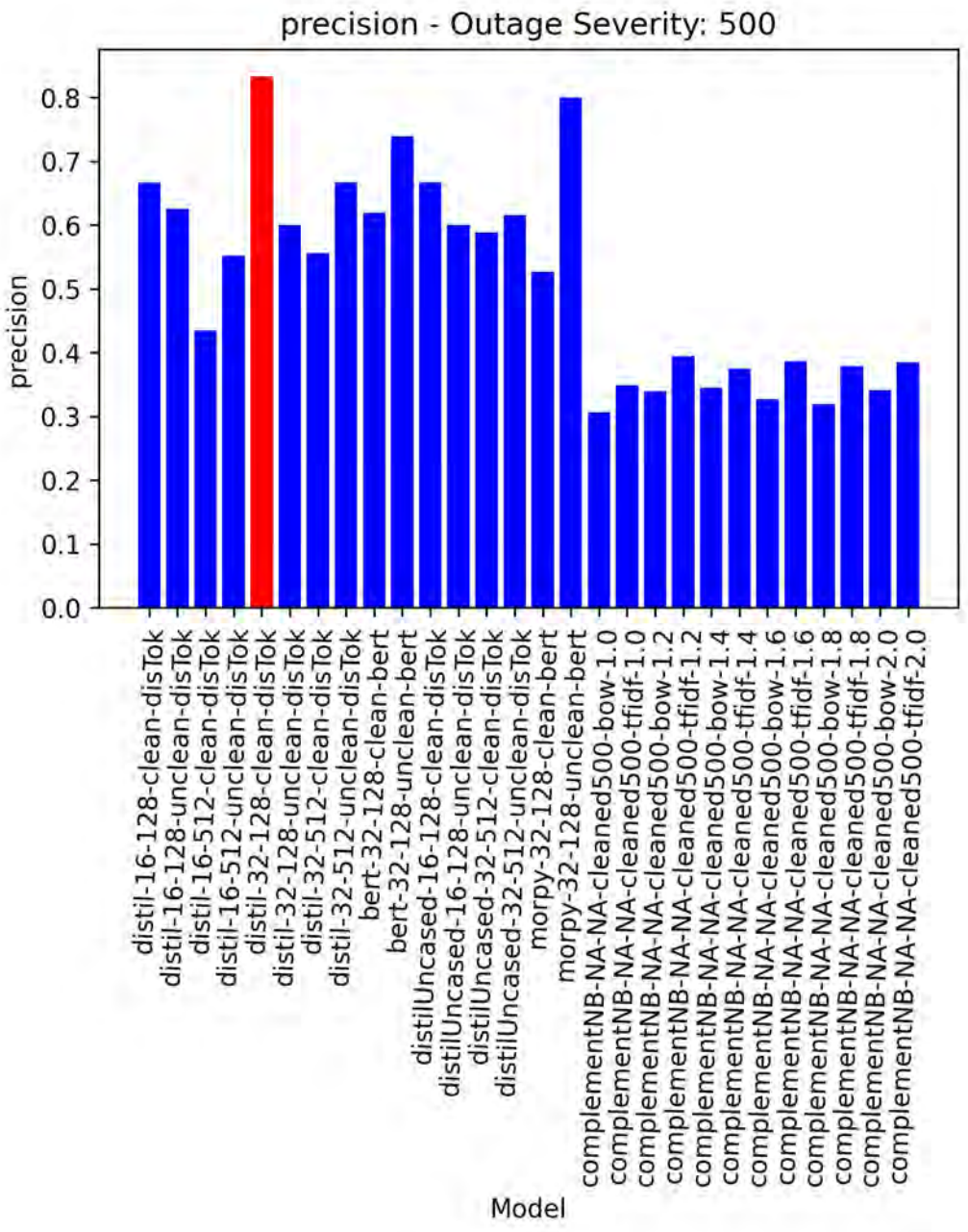

Figure 8-2: 500+ Hour Limit Precision Scores, All Models

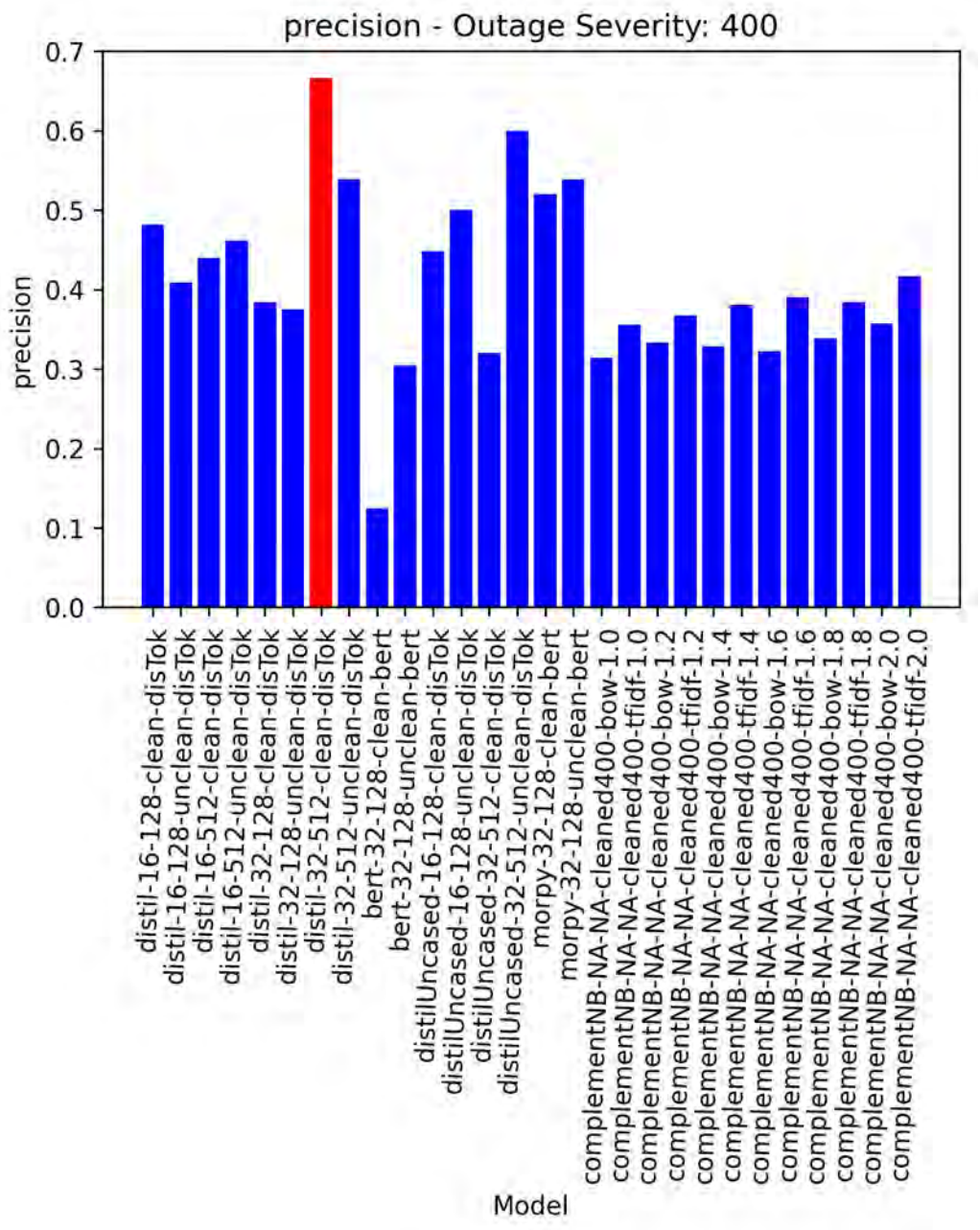

Figure 8-3: 400+ Hour Limit Precision Scores, All Models

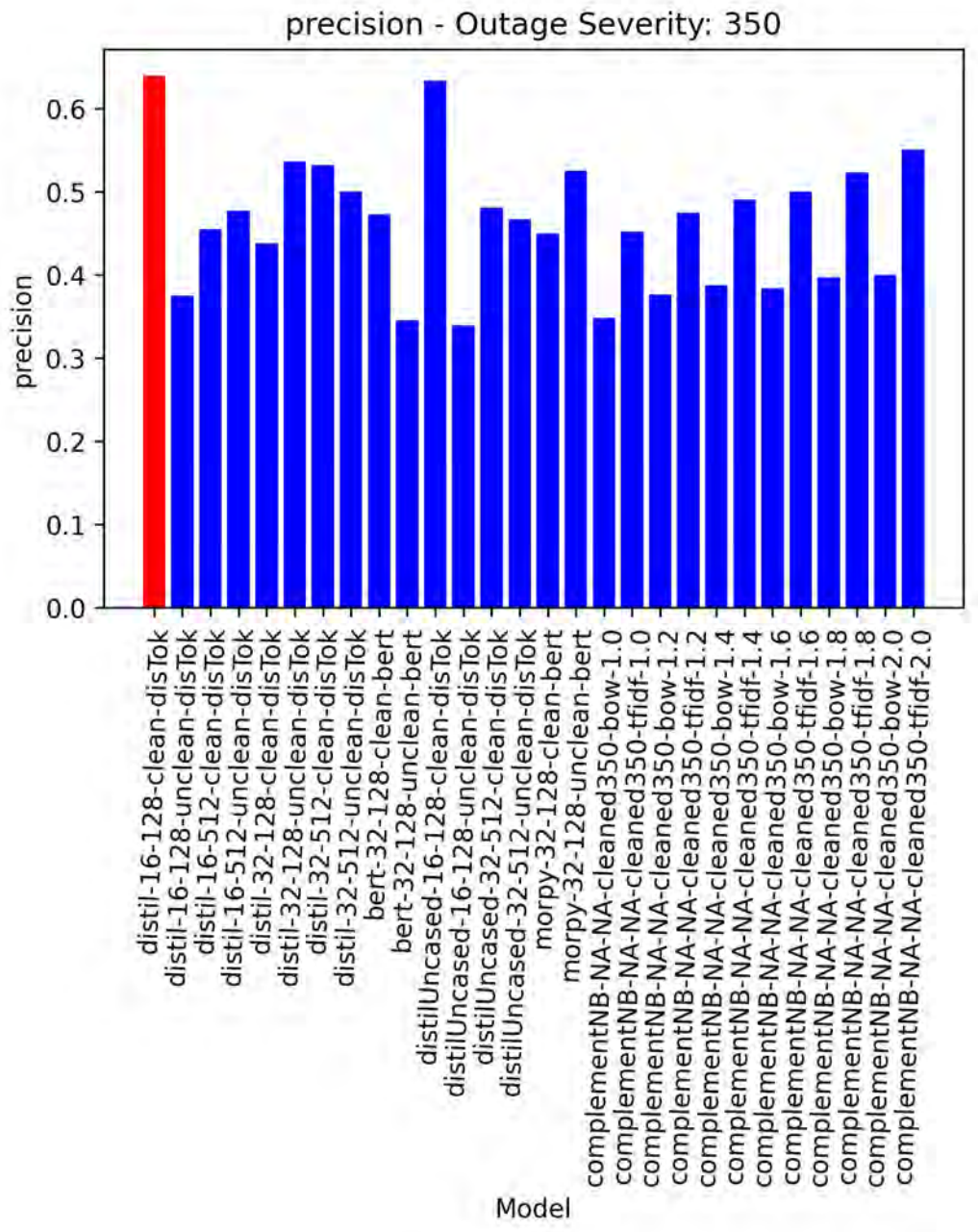

Figure 8-4: 350+ Hour Limit Precision Scores, All Models

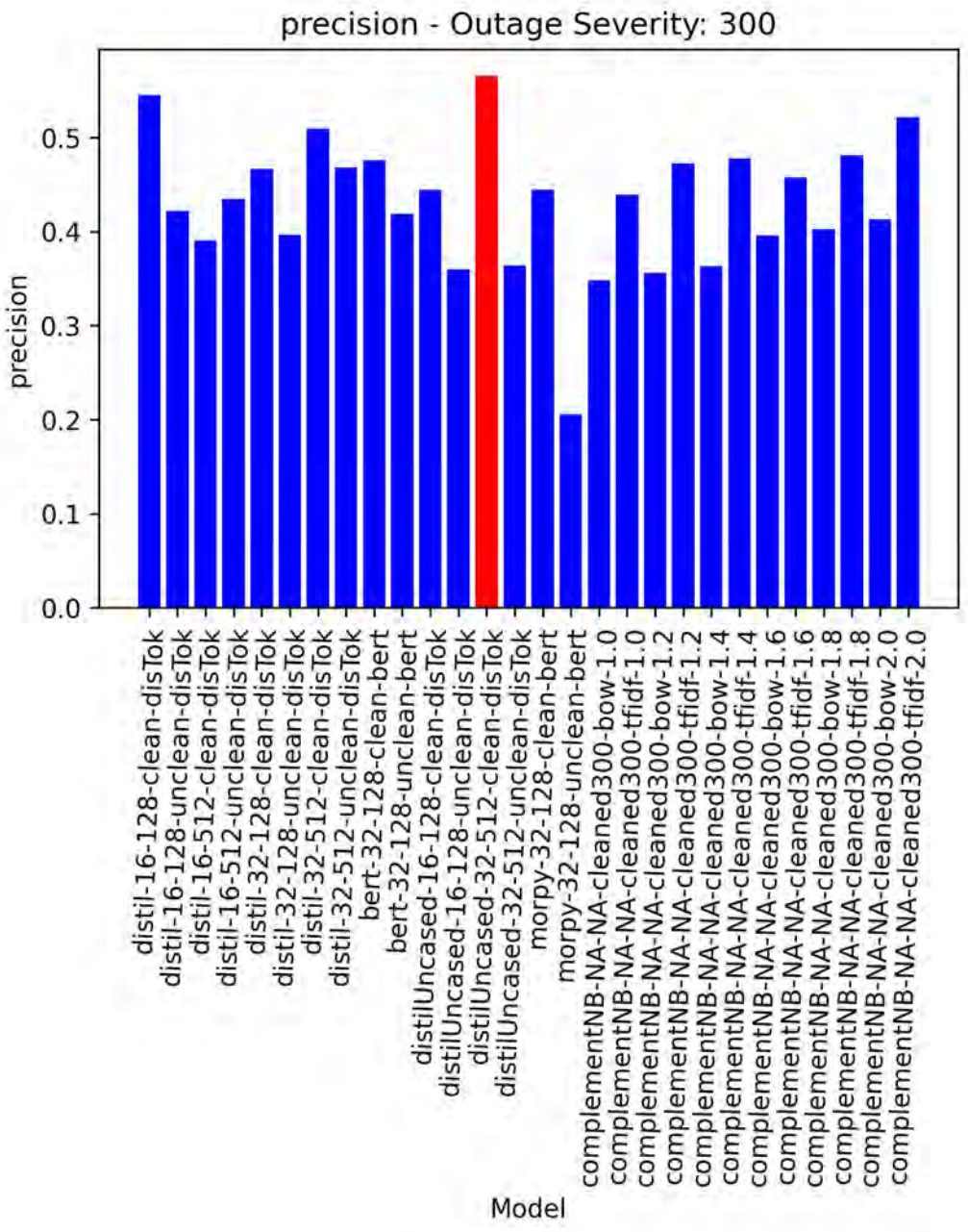

Figure 8-5: 300+ Hour Limit Precision Scores, All Models

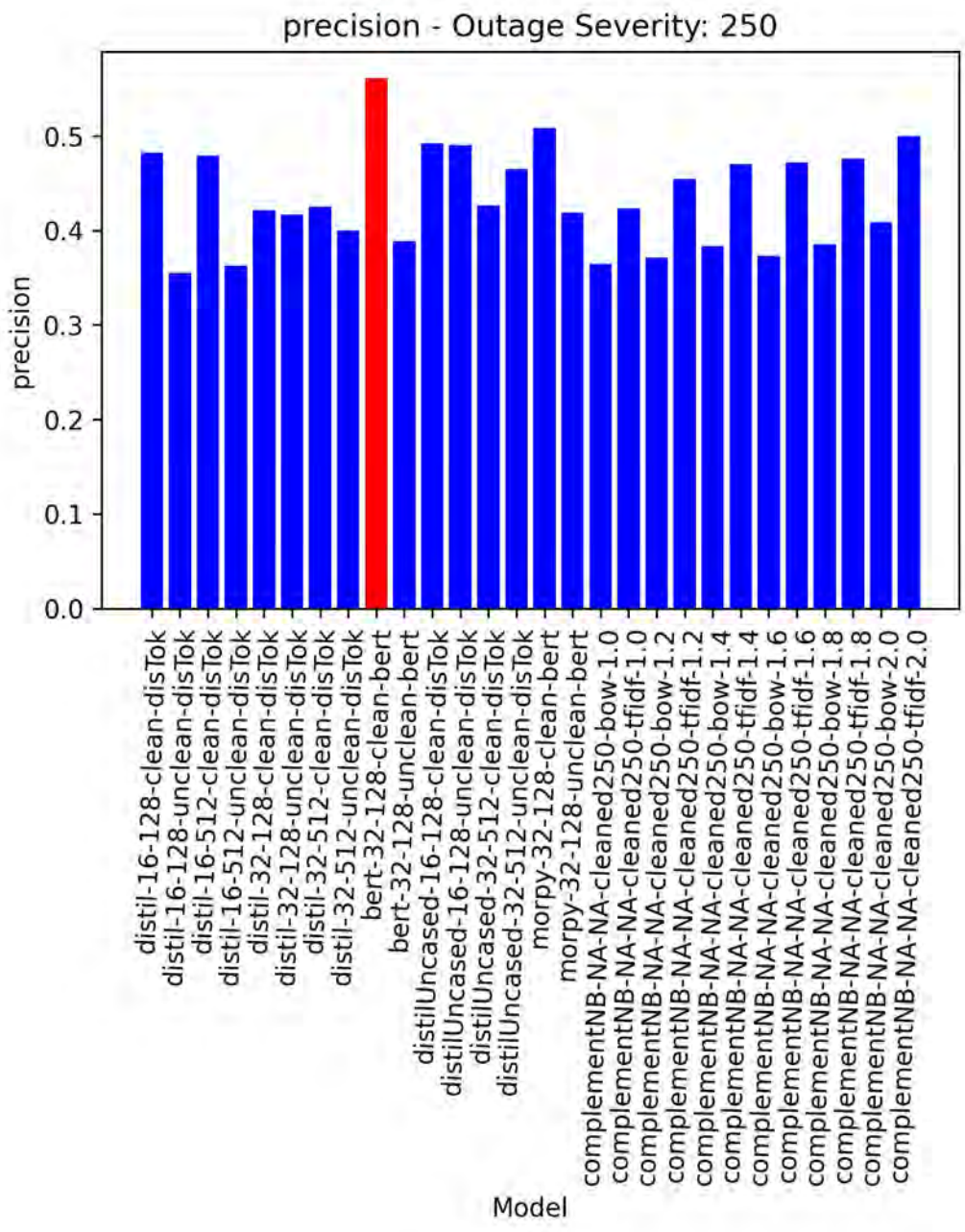

Figure 8-6: 250+ Hour Limit Precision Scores, All Models

Table 8-1 shows the results for models that reported the best Precision score per hour outage split. The precision score demonstrates the model's ability to determine whether a classified severe outage was an actual severe outage. This metric is used when the cost of a false positive is high. In other words, a model chosen for precision will be used when the user cannot afford to mistakenly classify a "Mild" outage as "Severe".

Table 8-1: Top Precision Score by Hour

Hour	Model	Precision	95% Confidence Interval	Width	Midpoint
250	bert-large-uncased-BertTokenizer-32-128-cleanDatasets	0.5614	43 – 68%	24	0.53
300	DistilBERT-base-uncased-DistilBERTTokenizer-32-512-cleanDatasets	0.5652	43 – 70%	27	0.54
350	batch16_maxLength128_distillBert-DistilBERTTokenizer-16-128-cleanDatasets	0.6388	48 – 77%	29	0.60
400	batch32_maxLength512_distillBert-DistilBERTTokenizer-32-512-cleanDatasets	0.6667	50 – 81%	32	0.49
500	batch32_maxLength128_distillBert-DistilBERTTokenizer-32-128-cleanDatasets	0.8333	63 – 94%	31	0.72
744	DistilBERT-base-uncased-DistilBERTTokenizer-16-128-uncleanDatasets	0.75	47 – 91%	44	0.72
Total Average and Standard Deviation		0.6692 ± 0.11		31.20 ± 6.91	0.6 ± 0.01

The DistilBERT models made up majority of the reported Precision results, where the highest Precision score observed was 83.33% from model `batch32_maxLength128_distillBert-DistilBERTTokenizer-32-128-cleanDatasets` of hour outage limit 500+. The average confidence interval width of this model is below the total average, indicating better generalization on the data split than other models. However, further evaluation of the model performance is required. The confusion matrix for this model is provided in Table 8-2.

Table 8-2: Confusion Matrix of 500+ Hour Limit, Max Precision Model

batch32_maxLength128_distillBert-DistilBERTTokenizer-32-128-cleanDatasets	Predicted Negative (Mild)	Predicted Positive (Spicy)
Actual Negative (Mild)	460	2
Actual Positive (Spicy)	11	10

Surprisingly, all but one model in the top Precision metric used the clean datasets. Model `DistilBERT-base-uncased-DistilBERTTokenizer-16-128-uncleanDatasets` for the 744+ hour limit was the only model reporting the use of raw datasets. Although the Precision for this model is above the total average, this does not necessarily mean the model had good generalization across the dataset. With the largest confidence interval width of 44, the above average Precision score is likely due to the lack of severe outage data points in the test sample pool.

The use of cleaned datasets across all other models indicates that cleaning and expanding the unique features in a text document led to better performance in the Precision metric when using the further trained and base DistilBERT models. However, as the outage severity limit is dropped to 350+ and lower, the Complement NB models with the TF-IDF feature extraction method becomes more competitive. This is surprising, since the transformer-based models are expected to improve in performance since there are more severe outages seen in the training process.

In summary, rigorous investigation of the transformer-based models with MORP is required to assess the utility in Precision. Due to the small sample pool of severe outages, synthetic data verified by a subject matter expert could improve the ability to test the robustness of the model. Interestingly, all but one of the transformer-based models reported using the cleaned datasets. This suggests that when solely using MORP data, the Precision metric is influenced by the expansion and labeling of acronyms and specific power plant components.

8.2 Top Recall Scores

Recall scores for every model in each outage severity split are provided in Figure 8-7 through Figure 8-12. The top Recall metrics are observed in transformer-based models for each hour limit split. Contrary to the Precision metric, Complement NB classifiers using the BOW feature extraction method is surprisingly competitive across all data splits.

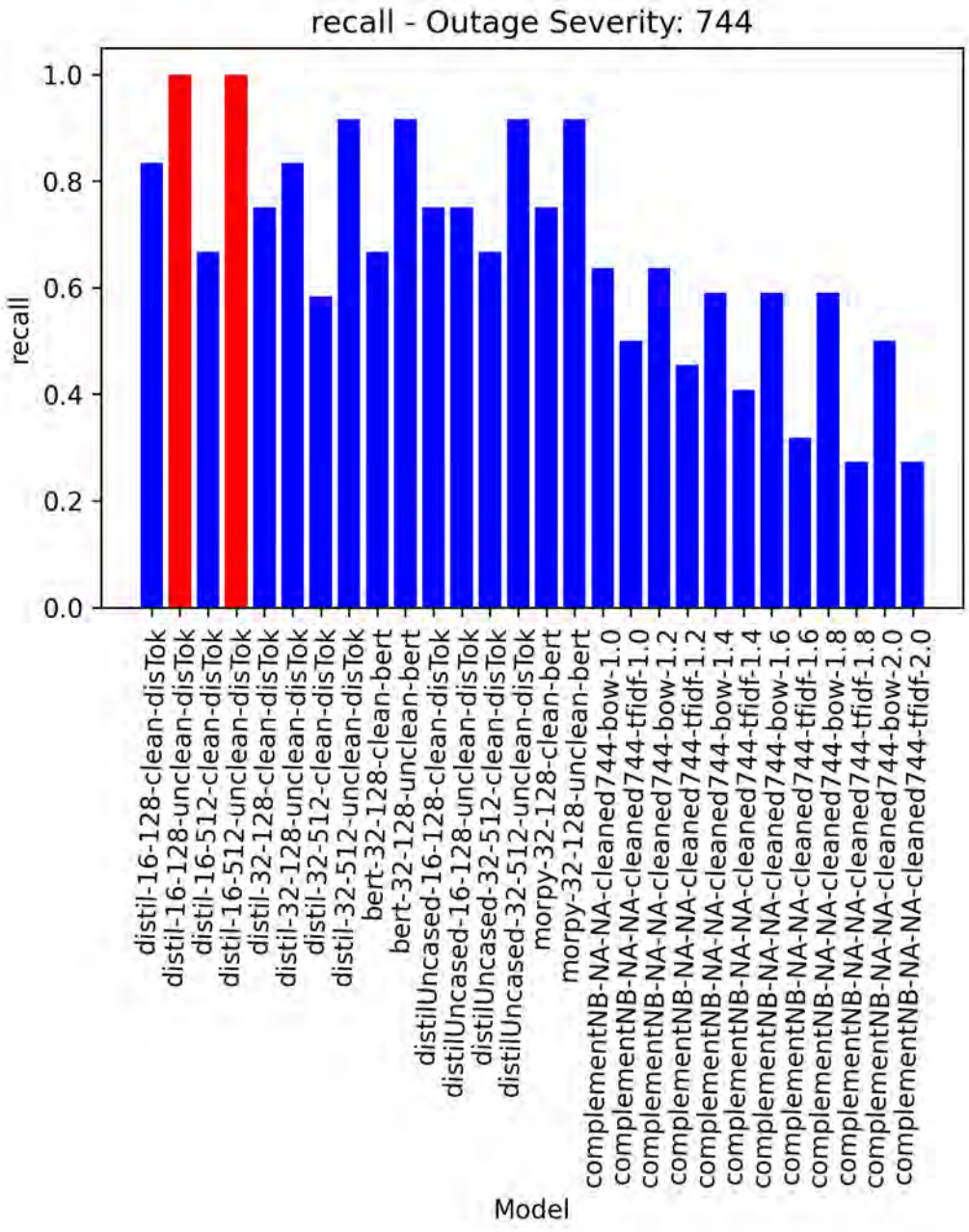

Figure 8-7: 744 Hour Limit Recall Scores, All Models

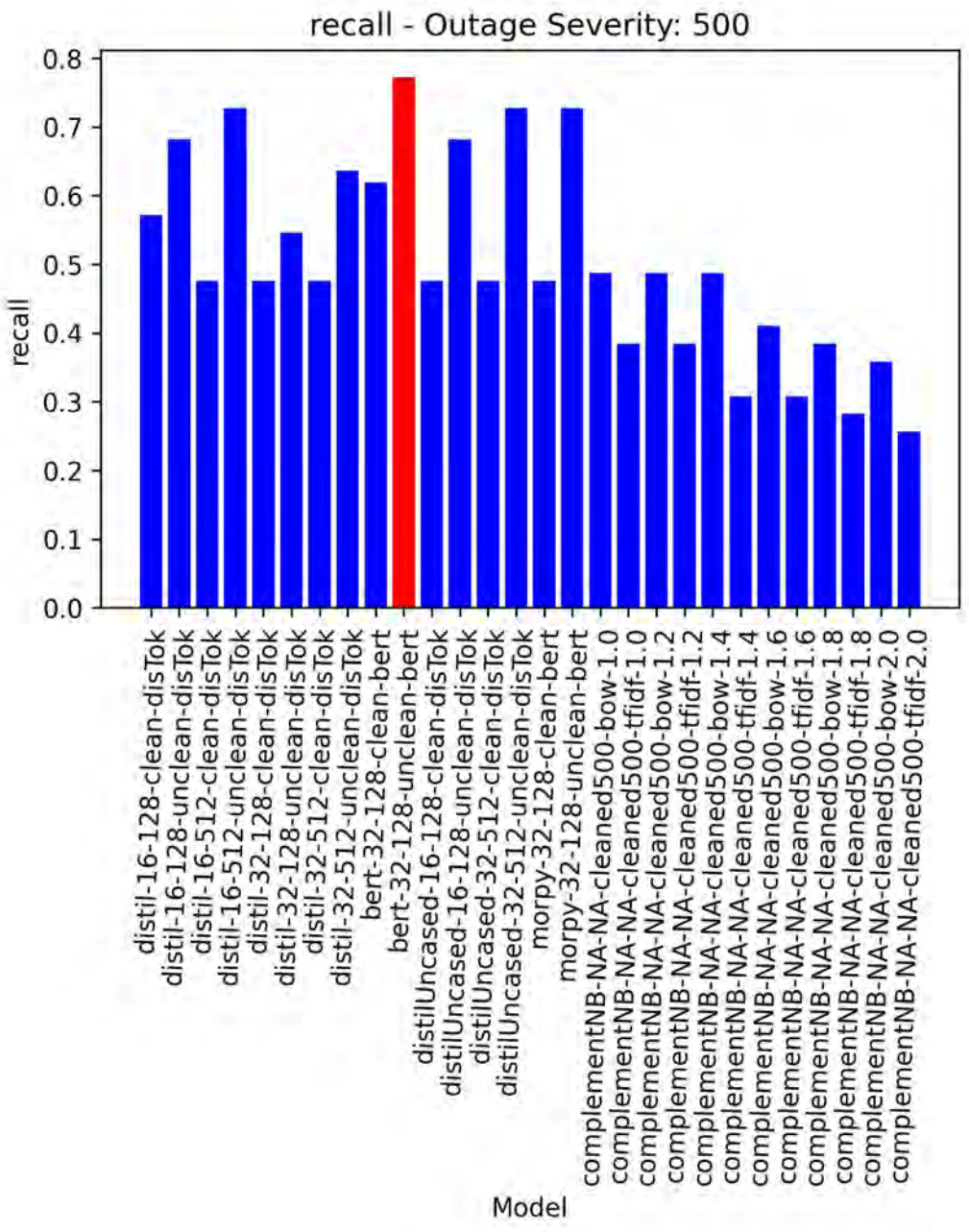

Figure 8-8: 500+ Hour Limit Recall Scores, All Models

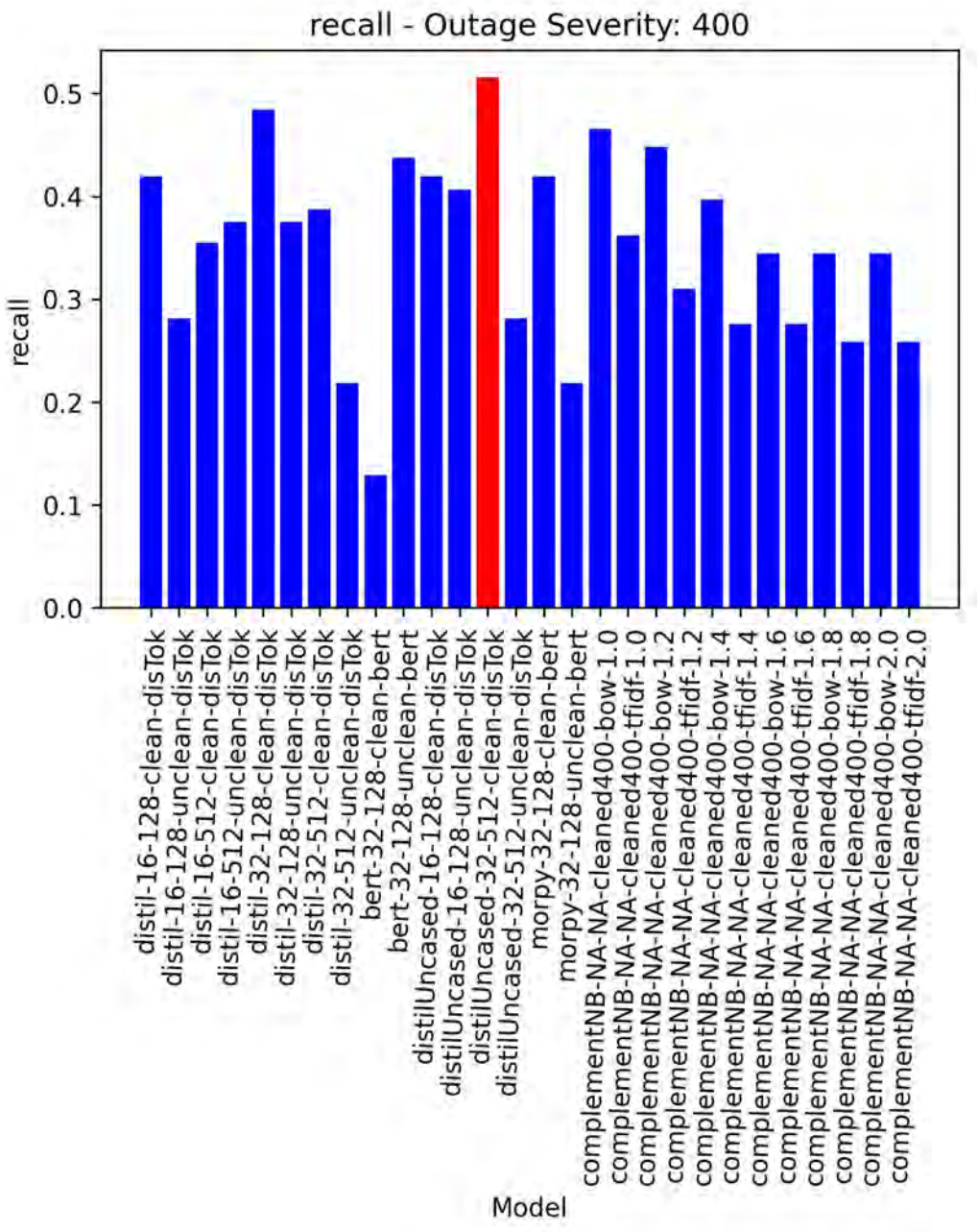

Figure 8-9: 400 Hour Limit Recall Scores, All Models

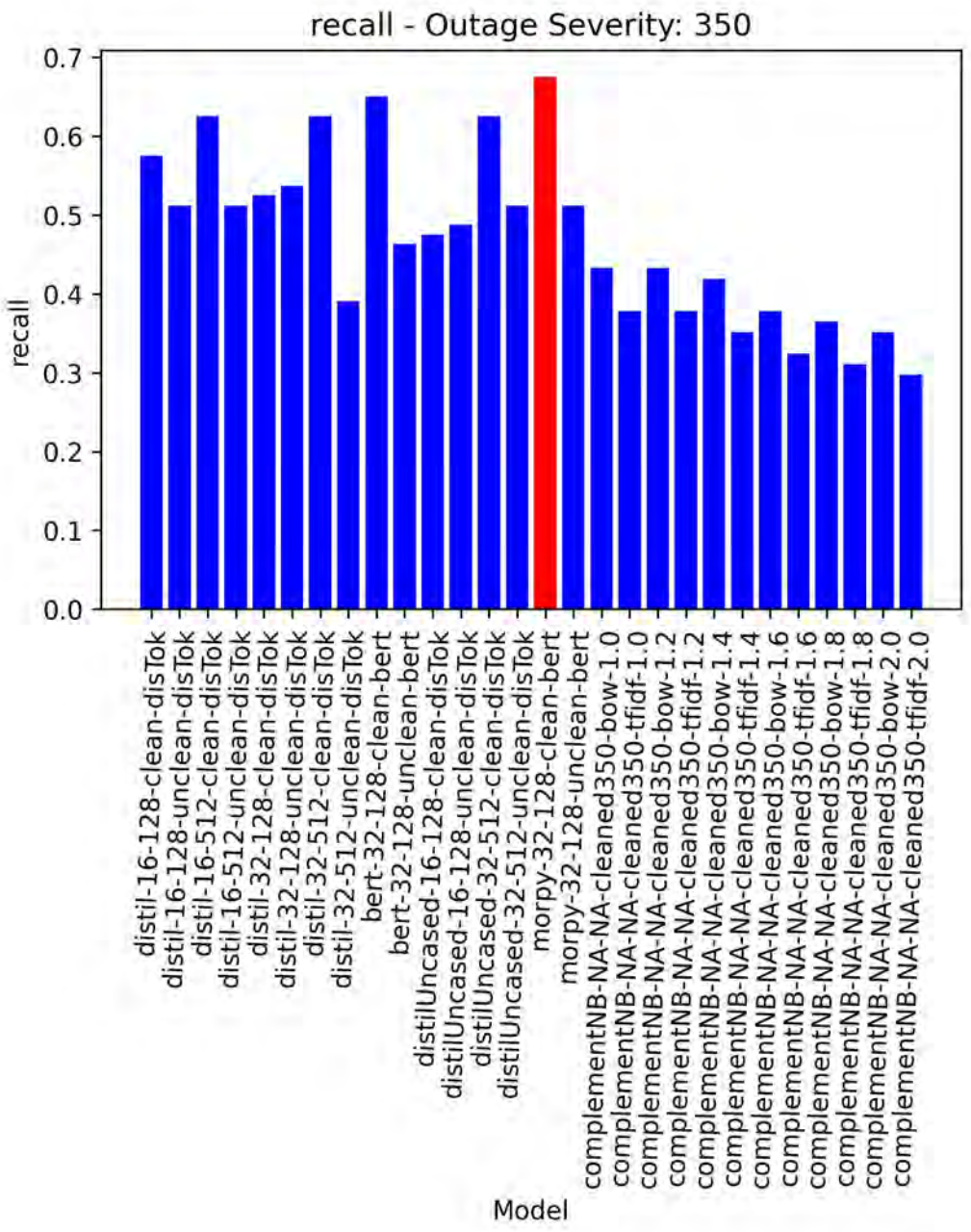

Figure 8-10: 350 Hour Limit Recall Scores, All Models

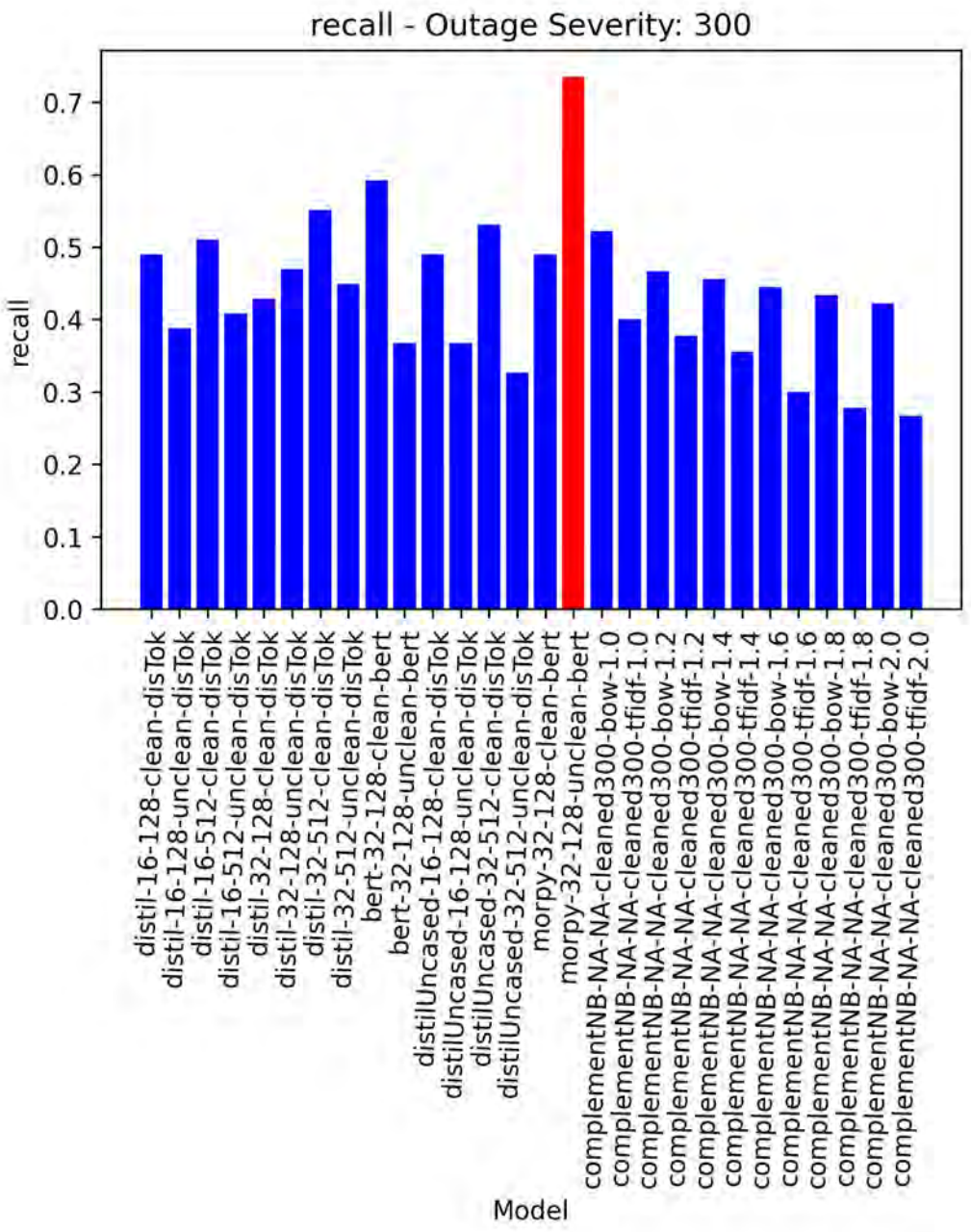

Figure 8-11: 300 Hour Limit Recall Scores, All Models

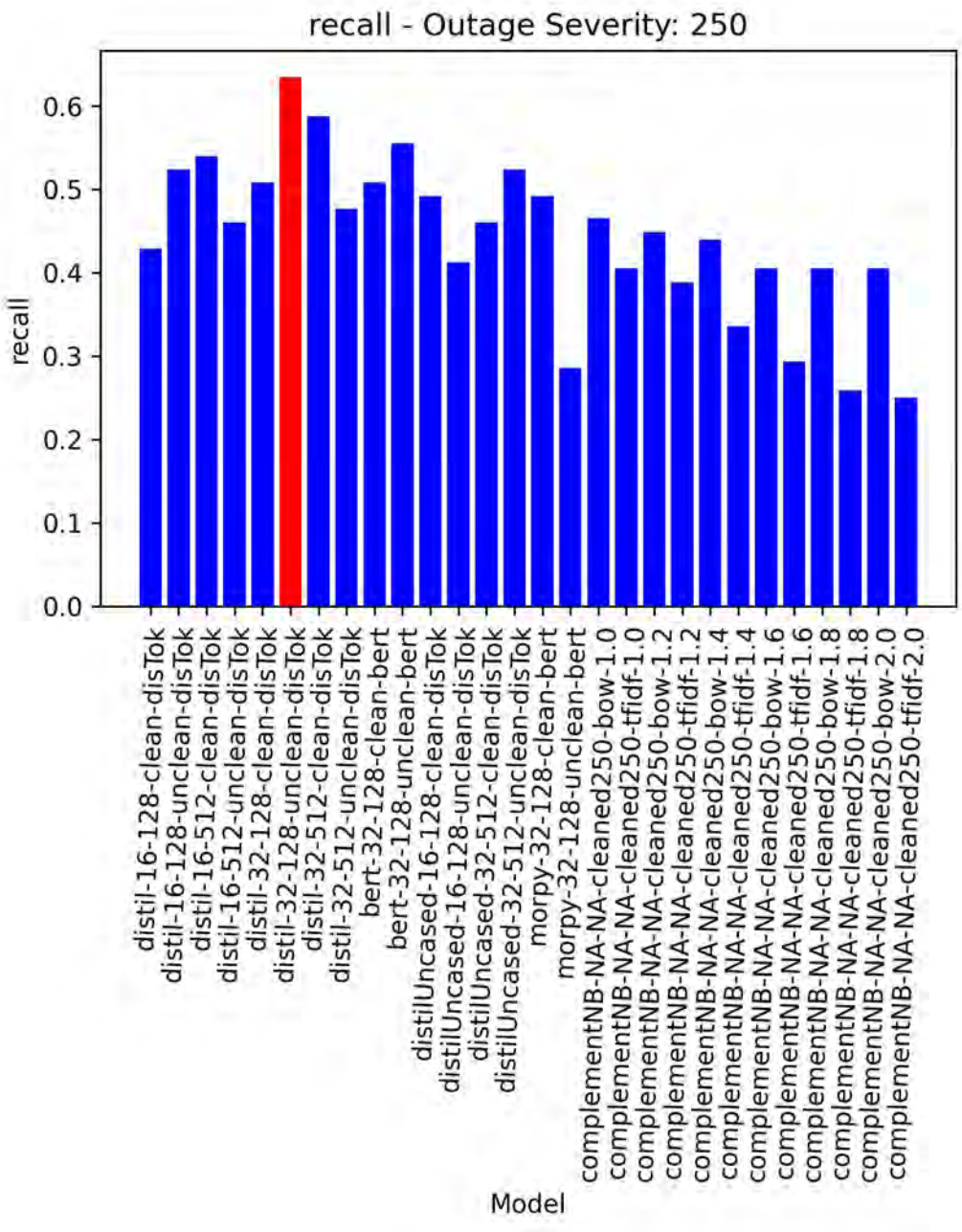

Figure 8-12: 250 Hour Limit Recall Scores, All Models

Table 8-3 provides the top Recall scores from all models per hour outage binning. The Recall score measures how many of the actual severe outages were correctly predicted as “Spicy”. This is important when the cost of a false negative is high. A user will choose a model for Recall when they want to improve the ability to capture all potentially “Spicy” outages, even if that includes misclassifying “Mild” outages.

Table 8-3: Top Recall Score by Hour

Hour	Model	Recall	95% Confidence Interval	Width	Midpoint
250	batch32_maxLength128_distillBert-DistilBERTTokenizer-32-128-uncleanDatasets	0.6349	51 – 74%	23	0.63
300	morpy-32-128-unclean-bert	0.7347	60 – 84%	24	0.72
350	morpy-32-128-clean-bert	0.675	52 – 80%	28	0.66
400	DistilBERT-base-uncased-DistilBERTTokenizer-32-512-cleanDatasets	0.5161	35 – 68%	33	0.51
500	bert-large-uncased-BertTokenizer-32-128-uncleanDatasets	0.7727	56 – 90%	34	0.73
744	batch16_maxLength512_distillBert-DistilBERTTokenizer-16-512-uncleanDatasets	1.0	75 – 100%	24	0.88
Total Average and Standard Deviation		0.7222 ± 0.16		27.67 ± 4.84	0.68 ± 0.12

The models with the highest Recall of 1.0 are reported from`batch16_maxLength512_distillBert-DistilBERTTokenizer-16-512-uncleanDatasets` and `batch16_maxLength128_distillBert-DistilBERTTokenizer-16-128-uncleanDatasets`. However, a perfect Recall of 1.0 is speculative and representative of the lack of severe outages in the sample pool. Similarly, model `bert-large-uncased-BertTokenizer-32-128-uncleanDatasets` for outage hour limit 500+ evaluated a Recall score of 77.27%. This model has a confidence interval higher than the total average, indicating a large uncertainty associated with this model.

Surprisingly, model `morpy-32-128-unclean-bert` for outage hours 300+ performed slightly better than the total average with a Recall score of 73.47%. It was also subjected to the second largest test sampling pool of all the dataset splits, containing 49 severe test samples. Figure 8-11 shows Recall scores for every model trained to classify 300+ hour limits. All other models from the 300+ hour limit range reported significantly lower recall scores, indicating good generalization over other models. The

confusion matrix provided in Table 8-4, shows `morpy-32-128-unclean-bert` incorrectly classified many “Mild” cases as “Spicy”, but captured majority of all “Spicy” cases.

Table 8-4: Confusion Matrix of 300 Hour Limit, Max Recall Model

morpy-32-128-unclean-bert	Predicted Negative (Mild)	Predicted Positive (Spicy)
Actual Negative (Mild)	296	139
Actual Positive (Spicy)	13	36

In summary, the top model demonstrating good generalization of the data and performance on Recall is the further trained BERT model `morpy-32-128-unclean-bert` for outage hour limit 300+. This is most like due to the larger number of parameters associated with the BERT model compared to the distilled version. Another component to the performance of this model is the data split for the 300+ hour limit. Relaxing the outage limit to 300+ allows for a better distribution of “Mild” and “Spicy” outages. Additionally, the metric for Recall is less stringent on false positives, permitting a greater holistic search for all potentially severe outages. A model with good recall performance will require more manual work downstream, but less time is spent in pre-processing techniques associated with RBS systems or traditional ML methods.

8.3 Top F1 Scores

F1 scores for every model in each outage severity split are provided in Figure 8-13 through Figure 8-18. The F1 score is the harmonic mean of precision and recall, evaluating the overall performance of a model. Transformer-based models performed best in all data splits.

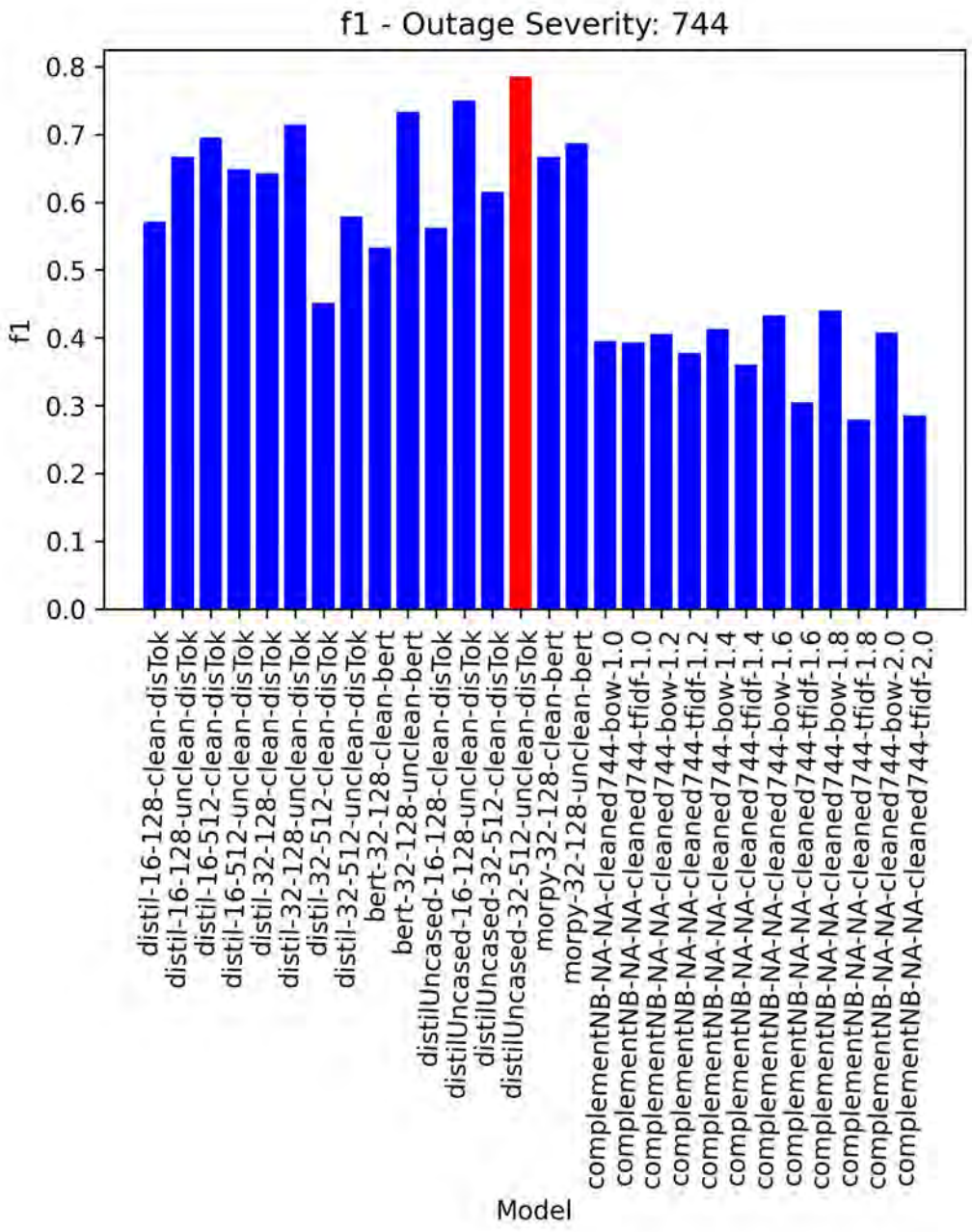

Figure 8-13: 744 Hour Limit F1 Scores, All Models

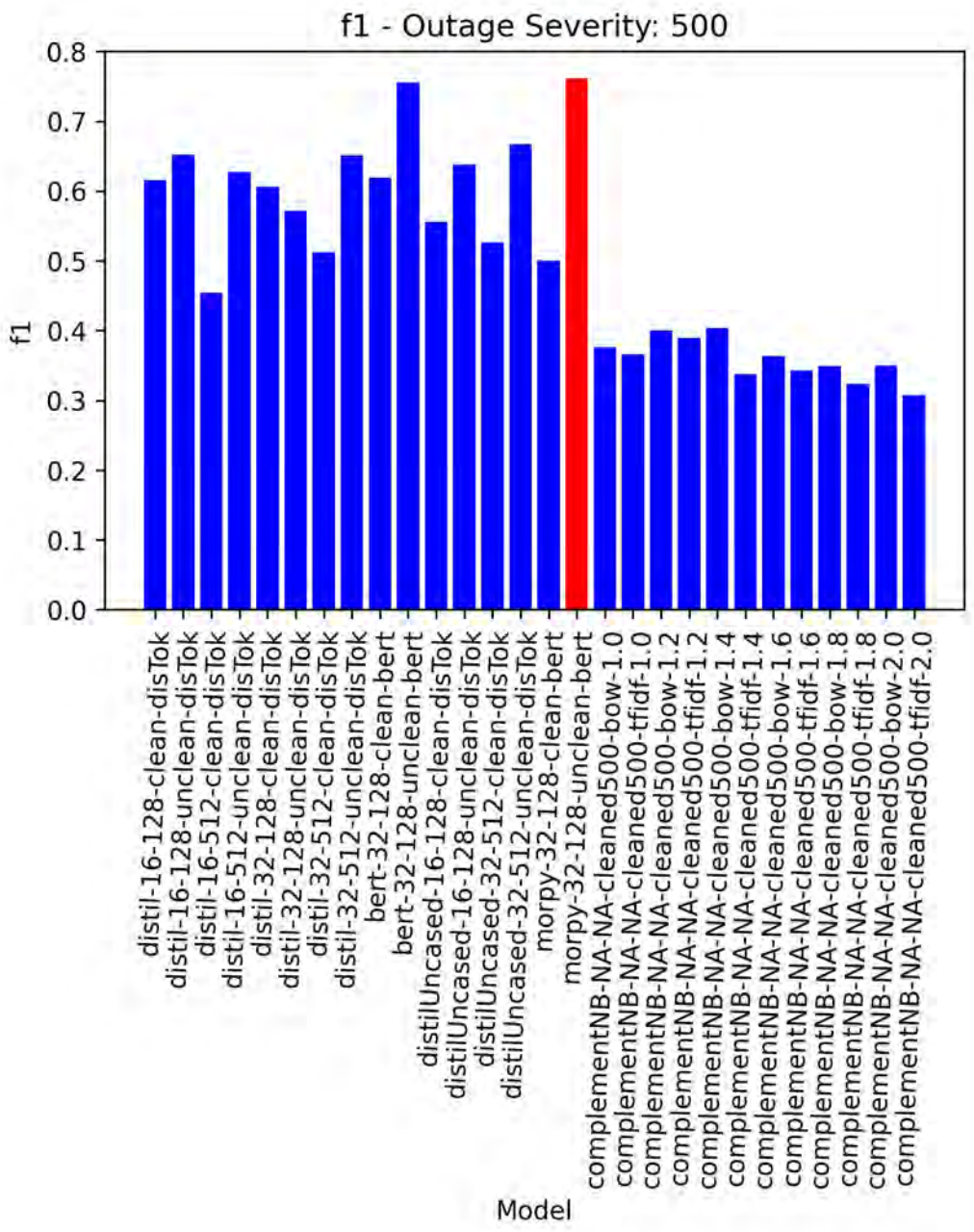

Figure 8-14: 500 Hour Limit F1 Scores, All Models

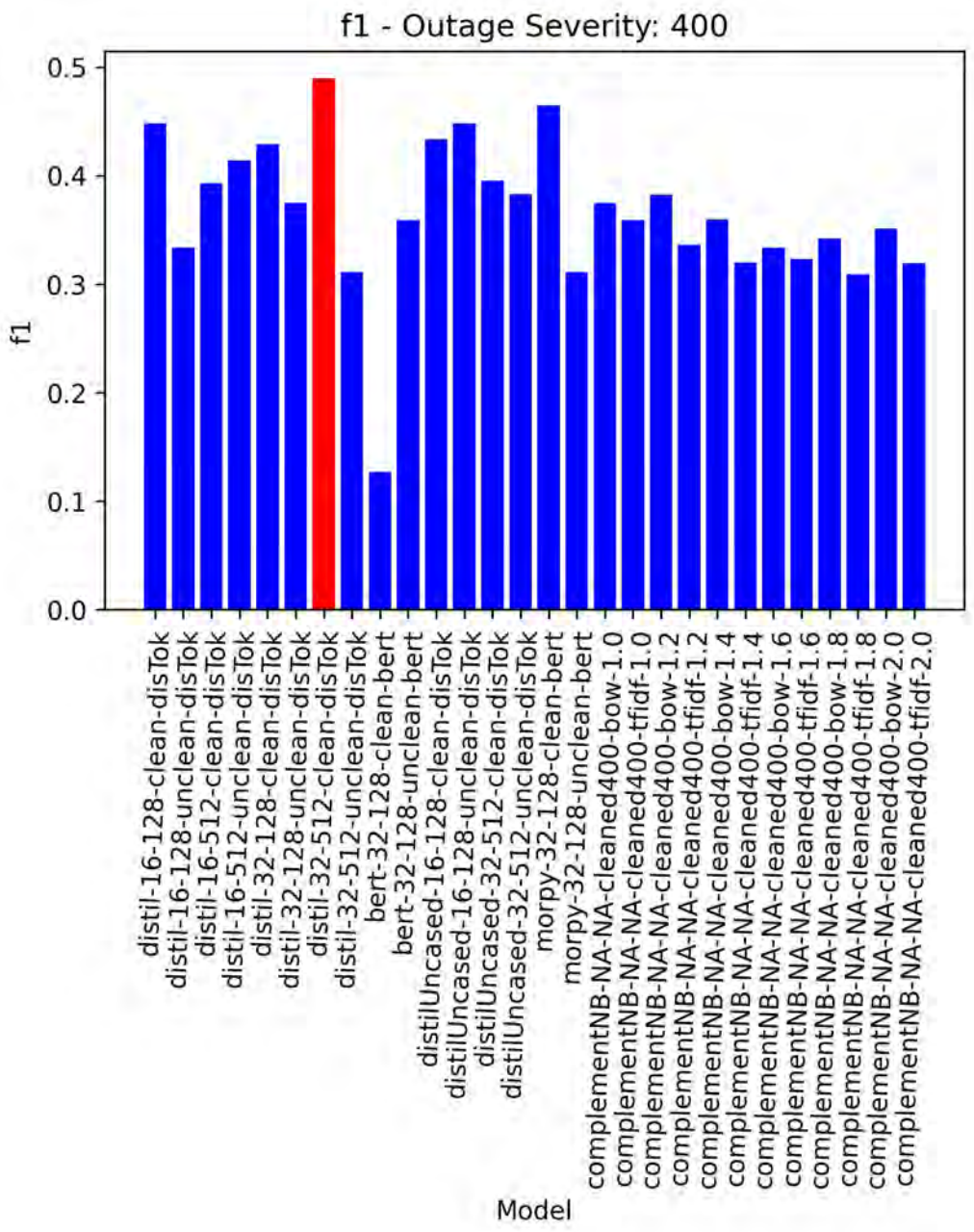

Figure 8-15: 400 Hour Limit F1 Scores, All Models

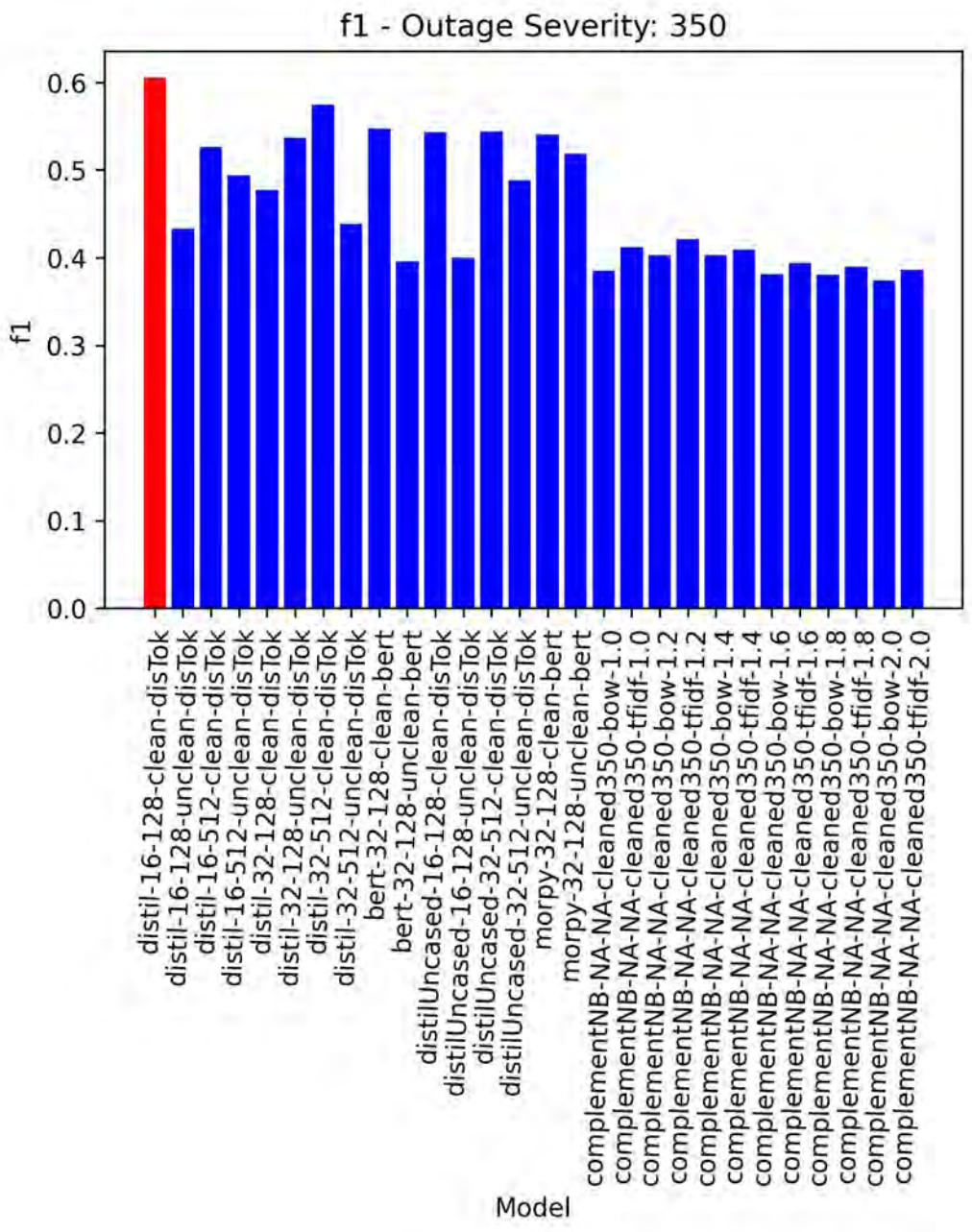

Figure 8-16: 350 Hour Limit F1 Scores, All Models

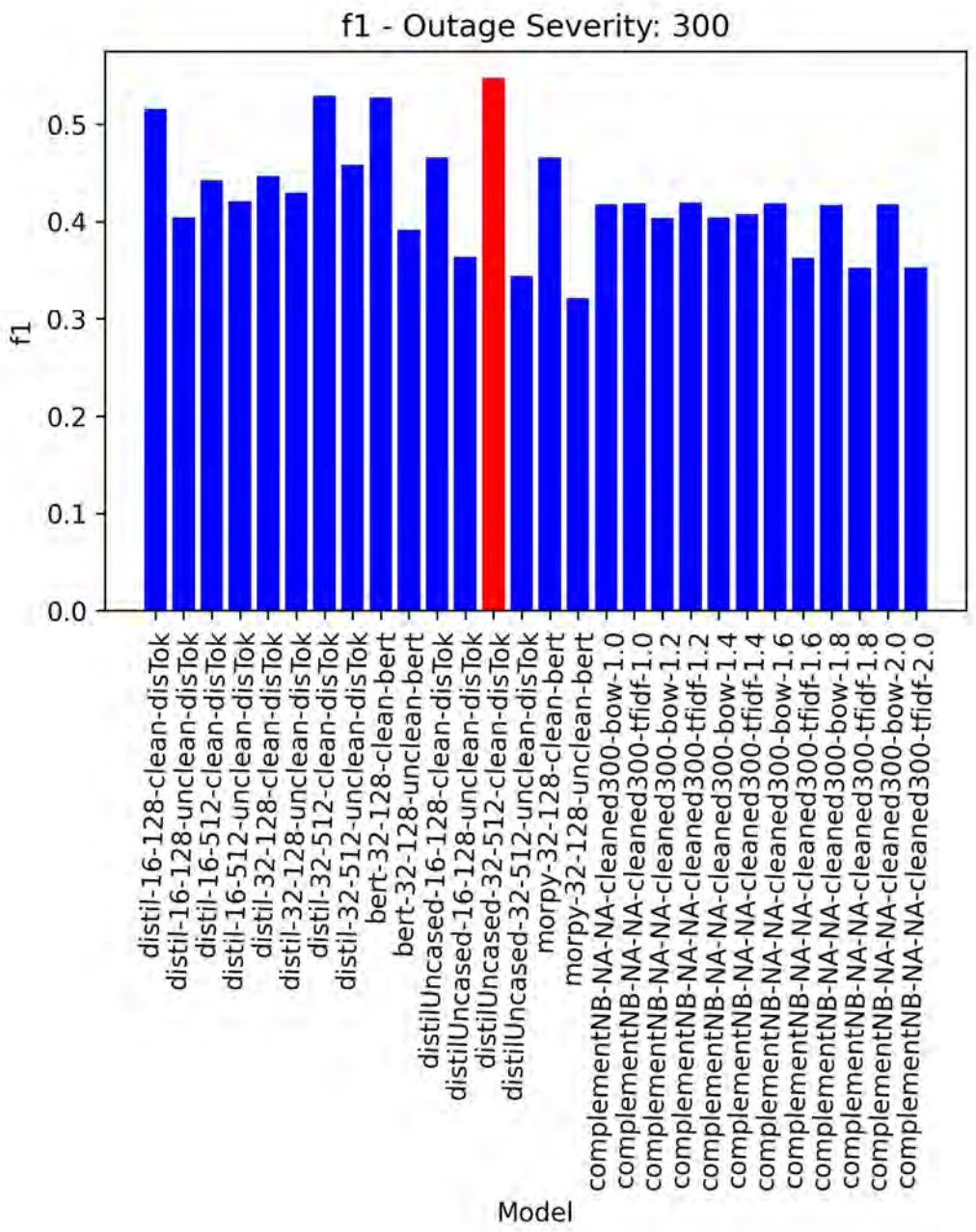

Figure 8-17: 300 Hour Limit F1 Scores, All Models

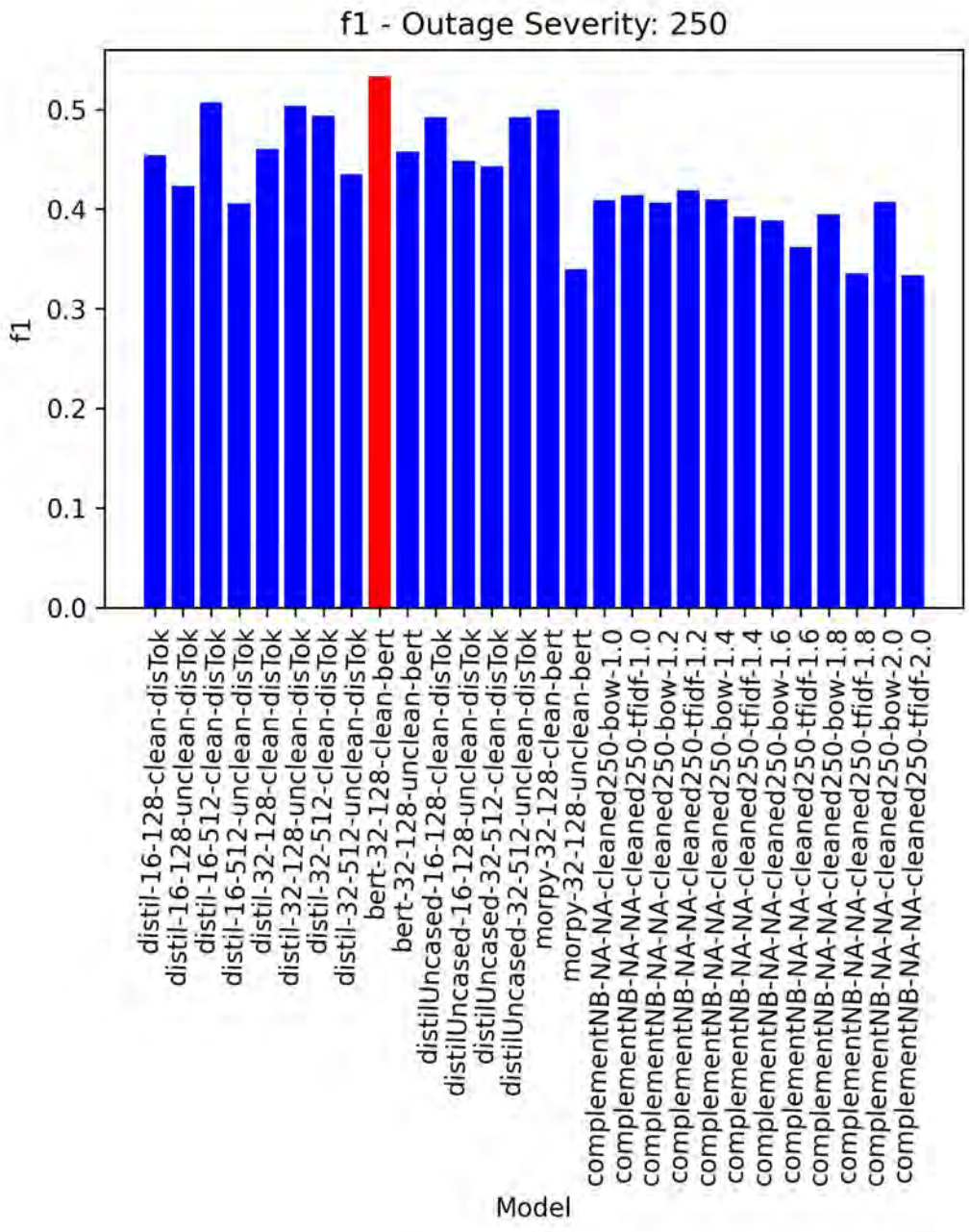

Figure 8-18: 250 Hour Limit F1 Scores, All Models

Table 8-5 shows the results for models that reported the best F1 score per hour outage. Models `morpy-32-128-unclean-bert` and DistilBERT-base-uncased-DistilBERTTokenizer-32-512-uncleanDatasets` were the only two models above the average performance in the top F1 scores. As seen from previous results, this is due to the challenges most models had with the Precision metric.

Table 8-5: Top F1 Score by Hour

Hour	Model	F1	95% Confidence Interval	Width	Midpoint
250	bert-large-uncased-BertTokenizer-32-128-cleanDatasets	0.5333	41 – 65%	24	0.56
300	DistilBERT-base-uncased-DistilBERTTokenizer-32-512-cleanDatasets	0.5474	41 – 68%	27	0.56
350	batch16_maxLength128_distillBert-DistilBERTTokenizer-16-128-cleanDatasets	0.6043	45 – 75%	29	0.63
400	batch32_maxLength512_distillBert-DistilBERTTokenizer-32-512-cleanDatasets-runs	0.4898	32 – 66%	33	0.65
500	morpy-32-128-unclean-bert	0.7620	55 – 89%	34	0.78
744	DistilBERT-base-uncased-DistilBERTTokenizer-32-512-uncleanDatasets	0.7857	50 – 93%	43	0.69
Total Average and Standard Deviation		0.6204 ± 0.125		31.67 ± 6.68	0.65 ± 0.08

There is a clear discrepancy in F1 performance as data splits transitions to the 400+ hour limit. Models at the 400+ and below report scores under the total average. This demonstrates a clear challenge transformer-based models have in performing well in both Precision and Recall. From the figures, Complement NB results tend to get closer with transformer models below the 400+ data splits, demonstrating only a slightly better trade off in classifiers.

9 Discussion

Due to the small test sample pool, metrics for models using dataset splits of 500+ or more are questionable since the Wilson Score Interval degrades as the sample size gets incredibly small. Even though the metrics appear to be great in these models, further assessment in their performance should be considered. Improving the test sample pool requires more reports documenting outages over 500+ hours. Although it is great that the nuclear power plant industry is limited in this data, it removes the capability to train and evaluate the robustness of text classifiers. Because of this limitation, metrics for models trained and evaluated on data splits 400+ and below should be considered as the baseline model from this research. As discussed in previous sections, transformer models utilize data to learn from, alleviating the dependency on explicit rules and extensive pre-processing. Therefore, more external data is required to improve all models. Considering the lack of severe outages, methods in data augmentation can be used to construct synthetic data representative of "Spicy" outages.

Of the models within the 250+ and 400+ range, performance in metrics F1, Recall and Precision were only considered. These models showed difficulty generalizing the MORP dataset for Precision, showing an inability to correctly classify truly severe outages. This in turn lead to overall poor metrics in the F1 score. Interestingly, all models showed the use of cleaned datasets, implying that the Precision metric benefited from the expansion of acronyms and treatment of component names, dates, etc. The top models in this data split range were the DistilBERT models further trained on nuclear domain text data. Complement NB models using the TF-IDF feature extraction method became more competitive for data splits at the 350+ hour limit and below.

Better performance in Recall was observed for all transformer-based models within the 250+ and 400+ range. Complement NB results tended to stay around the 40% range, but less deviation between the TF-IDF and BOW feature extraction approaches. However, unlike NB models observed in Precision, the best NB models reported using the BOW methodology. The best model was `morpy-32-128-unclean-bert` which was trained on the 300+ hour data split and fine-tuned on the nuclear domain data. Clearly observed in Figure 8-11, this model shows the best generalization on the MORP data. Unlike the reliance on cleaned datasets seen in Precision, `morpy-32-128-unclean-bert` was trained and evaluated on the raw text data. Since Recall sacrifices Precision to identify all potentially severe outages, it is speculated that the expansion of acronym and labeling of plant specific components is less important than the undisturbed semantics of the text.

The poor performance in Precision results in low F1 scores for models within the 250+ and 400+ range. Of these models, the best overall model is `batch16_maxLength128_distillBert-DistilBERTTokenizer-16-128-cleanDatasets` trained and evaluated on the 350+ data split. This model demonstrated the best balance between both Precision and Recall with an F1 score of 60.43%. Clearly,

more work is required to improve the Precision metric, however, `morpy-32-128-unclean-bert` showed promising results in Recall and the ability to flag potentially concerning outages.

10 Conclusion

There is no doubt that transformer models are revolutionizing how people interface with large amounts of data. Since the start of this research, many larger sophisticated transformer models have demonstrated incredible abilities. Large language modes like ChatGPT (OpenAI, ChatGPT, 2023), are opening doors to vast opportunities for technical advancement. These large models require an insane amount of data and require expensive computational hardware to train. For example, ChatGPT has an estimated 1.5 billion parameters, which is a staggering difference compared to BERTs 110 million. However, even with these parameters, ChatGPT can still be limited by the type of text data it was trained on. Like BERT, if it is not trained on data from the nuclear power domain, it will suffer in its performance on NLP tasks related to nuclear power.

Often, disciplines involving nuclear technology have been considered niche industries that require substantial regulation. Regulated niche environments like the nuclear power industry are required to follow strict protocols and follow quality standards on documentation for safety and transparency. These regulations and strict standards have arguably made the nuclear industry one the best areas for natural language processing research as it is filled with an incredibly long track record of technical, unambiguous language dating back to the early 1970's.

As seen from this research, all models with the best metrics in the 250+ and 400+ range were transformer-based models further trained on fast reactor text data. As shown from (Ayush Jain, 2020), there is a clear gain in performance when pre-training BERT from scratch. Replacing the fast reactor text data with LWR text data for pre-training and fine-tuning BERT with a larger database on LWR content could greatly improve the performance of all models.

In addition to pre-training with U.S. LWR text content, further investigation into expanding the context of MORP reports for fine-tuning may also improve overall performance. The average lengths of the MORP inputs are short and may not capture enough complexity of components and related sub-systems. Expanding the context could provide a language model more robust semantic knowledge about an outage, potentially improving the capability to classify higher outage severities.

More specific nuclear domain training and context expansion may significantly improve the performance of BERT, yet more research into the hyperparameter performance is required. This research applied no hyperparameter investigation to the transformer models. Further work into hyperparameter tuning would maximize their capabilities when performing classification tasks.

In conclusion, this research demonstrated the ability of transformer models to reasonably predict NPP outages of 300+ hours, using raw text data. Fine-tuning on partially relevant fast reactor text data, the encoder model, BERT could understand the semantics of short outage documents from MORP. With pre-training on specific LWR data, expanded context for fine-tuning and hyperparameter optimization language models like BERT would grant the nuclear power industry sophisticated AI tools to identify and understand complex relationships in power plant systems. As seen from this research, a large limitation for exploring language model utility is the lack of consolidated, organized, verified, and validated text content on LWRs. Without data to train these models, the nuclear power plant industry cannot take full advantage of these tools and will lack insight into economic challenges.

11 Future Work

Since the release of BERT, there have been many more large language models created and pre-trained with incredible amounts of data, far exceeding the capabilities of BERT. However, to the researcher's knowledge, there are only two models that have specifically tailored knowledge on the nuclear power domain. Researcher Ayush Jain pre-trained BERT from scratch, and the training of NukeLM from Lee Burke and researchers at Pacific Northwest National Lab (Pacific Northwest National Laboratory, 2021), both of which are not publicly available. To this extent, using large language models and leveraging their incredible ability to fine-tune for the nuclear power industry is challenging since there is no known database of labeled, consolidated data of all LWR U.S. reactors. Using subject matter expert knowledge, the construction, verification, and validation of datasets for NLP tasks tailored to the nuclear power domain would open doors to various applications.

With new, better models being trained and deployed for the public, the only limiting factor for applying language models in the nuclear power domain is consolidated data for both pre-training and fine-tuning. As seen in this research, the task of classification on outage severity provides a new perspective on NPP outages but are limited to data tailored for LWRs. Improving the quality and quantity of LWR data to pre-train BERT from scratch would lead to a baseline language model to be used for NLP tasks involving the U.S. LWR fleet. Data such as journal articles, textbooks, reports, etc. can be used to construct a LWR focused text set for pre-training, like Ntext from (Ayush Jain, 2020).

Using LER's and plant specific documentation, MORP reports can be expanded with detailed outage context. This would require cross-referencing existing MORP outages with any documents accessible in publicly available NRC databases like ADAMS (NRC, ADAMS, 2012). Longer context with more features can better capture system complexities and relationships about an outage.

Additionally, synthetically increasing the small sample pool of 'severe' outages can be performed to improve the evaluation of language model performance. This may be readily achieved by combining nuclear power domain expertise and existing generated pre-trained models like ChatGPT to generate synthetic text data that accurately represents expected outage lengths. This may be labor intensive, but increasing the severe outage sample pool would balance the dataset and increase the fidelity of outage predictions in the 500 and 744 range.

Due to the lack of available text data, this research combined language for both BWR and PWR systems. One area of future work could look at constructing two datasets with BWR and PWR content, then training two separate models. This would likely lead to better performance since each language model will be constrained to only one system.

This research did not explore any optimization techniques for hyperparameters existing in large language models. Language models are constructed with many layers all with their own complexity. Future work with a specific model should undergo a thorough analysis for hyperparameter optimization ensuring the model is performing its best.

With established datasets, nuclear research and industry can begin making use of qualitative data for quantitative applications. With nuclear domain trained language models, applications may potentially expand outside of reactor operations to specific use cases in licensing, requirement tracing or document classification tasks.

By leveraging a long track record on quality documentation, the nuclear industry may arguably be harboring one of the best sources of text data for natural language tasks. As models improve and become more accessible, the only limitation for sophisticated language modeling is lacking consolidated data availability. introducing a new path of AI in nuclear.

Bibliography

(n.d.). Retrieved from spacy: https://spacy.io/

Aggarwal, C. C. (2018). *Machine Learning for Text.* Yorktown Heights, NY: Springer.

Almeida, F. &. (2019, January 25). *Word embeddings: A survey*. Retrieved from arXiv.org: https://arxiv.org/abs/1901.09069

Anandarajan, M. H. (2019). Text Preprocessing. In: Practical Text Analytics. In *Advances in Analytics and Data Science, vol 2.* Springer. doi:https://doi.org/10.1007/978-3-319-95663-3_4

Andrew McCallum, K. N. (1998). A Comparison of Event Models for Naive Bayes Text Classification. *AAAI Conference on Artificial Intelligence.*

Anna Rogers, O. K. (2020). A Primer in BERTology: What We Know About How BERT Works. *Transactions of the Association for Computational Linguistics, 8*, 842–866. doi:https://doi.org/10.1162/tacl_a_00349

Ashish Vaswani, e. a. (2017). Attention Is All You Need. arXiv. doi:10.48550/ARXIV.1706.03762

Ayush Jain, e. a. (2020). NukeBERT: A Pre-trained language model for Low Resource Nuclear Domain. *arxiv*. Retrieved from https://arxiv.org/pdf/2003.13821.pdf

Brown, T. B. (2020). Language Models are Few-Shot Learners. doi:10.48550/ARXIV.2005.14165

C. Manning, M. S. (2014). The Stanford CoreNLP Natrual Language Processing Toolkit, in: Association for Computational Linguistics., (pp. 55-60). doi:10.3115/v1/P14-5010

Carola A. Gregorich, P. (2020, September). Power Industry Dictionary for Text-Mining and Natural Language Processing Application. (S. Ramirez, Ed.) *EPRI: Electric Power Research Institute: Quick Insight*.

Charu C. Aggarwal, C. Z. (n.d.). *Mining Text Data.* Springer. doi:DOI 10.1007/978-1-4614-3223-4

Clarin, B. (2019). *Customer Engagement with Voice Assistants.* (EPRI, Ed.) Retrieved from EPRI: Electric Power Research Institute: https://download.epri.com/DownloadService/Attachment.svc/AttachmentId=59647

Collobert, R. W. (2011). Natural Language Processing (Almost) from Scratch., (pp. 2493-2537).

Cristianini, N. &.-T. (2000). *An introduction to support vector machines and other kernel-based learning methods.* Cambridge university press.

Cullingford, R. (1977, Feb). SAM: A Program That Uses World Knowledge to Understand. *SIGART Bulletin*, 53-54. doi:10.1145/1045283.1045324

Deepak R. Chandran, V. G. (2022). A Short Review of the Literature on Automatic. *Journal of Computer and Communications*, 55-73.

Fillmore, C. J. (1968). The Cases for Case. In *Form and Meaning of Language.* Chicago: Chicago The University of Chicago Press.

Goldberg, Y. (2017). *Neural Network Methods for Natural Language Processing.* Morgan & Claypool.

González-Carvajal, S. a.-M. (2020). Comparing BERT against traditional machine learning text classification. arXiv. doi:10.48550/ARXIV.2005.13012

Gurinder Singh, B. K. (2019). Comparison between Multinomial and Bernoulli Naïve Bayes for Text Classification. *2019 International Conference on Automation, Computational and Technology Management (ICACTM).* doi:10.1109/ICACTM.2019.8776800

H. Drucker, C. C. (n.d.). Boosting and Other Ensemble Methods. *Neural Computation, 6*, 1289-1301. doi: 10.1162/neco.1994.6.6.1289

Hochreiter, S. (1998). The Vanishing Gradient Problem During Learning Recurrent Neural Nets and Problem Solutions. *International Journal of Uncertainty, Fuzziness and Knowledge-Based Systems*, 107-116. doi:https://doi.org/10.1142/S0218488598000094

HuggingFace. (n.d.). Retrieved from https://huggingface.co/

Hutchins, W. (2004). Machine Translation: From Real Users to Research: The Georgetown-IBM Experiment Demonstrated in January 1954. *6th Conference of the Association for Machine Translation in the Americas* (pp. 102-114). Washington, DC: Springer, Berlin, Heidelberg.

Ian Goodfellow, Y. B. (2016). *Deep Learning.* Massachusetts Institute of Technology.

IEEE. (2014). IEEE Guide for Collecting, Categorizing, and Utilizing Information Related to Electric Power Distribution Interruption Events. Piscataway, New Jersey: IEEE. Retrieved from https://standards.ieee.org/standard/1782-2014.html

Jacob Devlin, M.-W. C. (2018). BERT: Pre-training of Deep Bidirectional Transformers for Language Understanding. arXiv. doi:10.48550/ARXIV.1810.04805

Jason Rennie, e. a. (2003). Tackling the Poor Assumptions of Naive Bayes Text Classifiers. *Twentieth International Conference on Machine Learning.* Washington DC: Artificial Intelligence Laboratory; Massachusetts Institute of Technology. Retrieved from https://people.csail.mit.edu/jrennie/papers/icml03-nb.pdf

Jonathan J. Webster, C. K. (1992). Tokenization as the Initial Phase in NLP. *COLING 1992 Volume 4: The 14th International Conference on Computational Linguistics.* Hong Kong.

Jonathan Webster, C. K. (1992). Tokenization as the Initial Phase in NLP. *Proceedings of the 14th conference on Computational linguistics*, *4*, pp. 1106-1110. doi:DOI: 10.3115/992424.992434

Jones, K. S. (1994). Natural Language Processing: A Historical Review*. In N. C. Antonio Zampolli, *Current issues in computational linguistics.* Springer Dordrecht. Retrieved from https://citeseerx.ist.psu.edu/viewdoc/download?doi=10.1.1.517.5263&rep=rep1&type=pdf: https://citeseerx.ist.psu.edu/viewdoc/download?doi=10.1.1.517.5263&rep=rep1&type=pdf

K. Omura, M. K. (2012). Weighted naïve Bayes classifier on categorical features. *12th International Conference on Intelligent Systems Design and Applications (ISDA)*, (pp. 865-870). doi:10.1109/ISDA.2012.6416651.

Kacker, P. a. (2022). ABB-BERT: A BERT model for disambiguating abbreviations and contractions. *Proceedings of the 18th International Conference on Natural Language Processing* (pp. 289-297). arXiv. doi:10.48550/ARXIV.2207.04008

Kamran Kowsari, e. a. (2019). Text Classification Algorithms: A Survey. *Information*.

Kelly, K. (2016). *The Inevitable.* Viking Press.

L. Xiao, G. W. (2018). Patent Text Classification Based on Naive Bayesian Method. *2018 11th International Symposium on Computational Intelligence and Design*, (pp. 57-60). doi:10.1109/ISCID.2018.00020

Lewis, D. (2019). *Distribution Reliability Analytics.* Retrieved from EPRI: Electric Power Research Institue: https://download.epri.com/DownloadService/Attachment.svc/AttachmentId=61548?errorpage=http://memberc

Liberman, M. Y. (1991). The trend towards statistical models in natural language processing. In *Natural Lanuage and Speech* (pp. 1-7). Berlin: Springer, Berlin, Heidelberg.

Manning, C. S. (2014). The Stanford CoreNLP natural language processing toolkit. *Proceedings of 52nd Annual Meeting of the Association for Computational Linguistics: System Demonstrations.* doi: https://doi.org/10.3115/v1/p14-5010

Manning, J. P. (2014). GloVe: Global Vectors for Word Representation. In *Empirical Methods in Natural Language Processing (EMNLP)* (pp. 1532-1543). Retrieved from http://www.aclweb.org/anthology/D14-1162

Marcel R. Harper, J. W. (1997, May 15). Revised Contents of the Monthly Operating Report (Generic Letter 97-02). Washington, DC. Retrieved from https://www.nrc.gov/reading-rm/doc-collections/gen-comm/gen-letters/1997/gl97002.html.

Martin, D. J. (2009). *Speech and Language Processing: An Introduction to Natural Language Processing.* Upper Saddle Rive, NJ: 2d Ed. Pearson Prentice Hall.

Mikolov, T. a. (2013). Efficient Estimation of Word Representations in Vector Space. doi:10.48550/ARXIV.1301.3781

Miltiadis Alamaniotis, L. H. (2015). Preidcitve Based Monitoring of Nuclear Component Degradation Using Support Vector Regression Approach. *9th International Conference on Nuclear Power Plant Instrumentation, Control & Human-Machine Interface Technologies* (pp. INL/CON-14-32980). Idaho National Lab.

Mirzazad, S. (2019, 9 27). *Technology Innovation.* Retrieved from EPRI: https://www.epri.com/research/products/000000003002017321

(2016). *MORP2 Definitions.* Idaho Falls: Idaho National Lab.

Murphy, K. P. (2012). *Machine Learning: A Probabilistic Perspective.* Massachusetts Institute of Technology.

Murphy, P. V. (2015). *Omaha Public Power District.* New York: Standard & Poor's Rating Services - McGraw Hill Financial.

Nikolenko, S. I. (2021). *Synthetic Data for Deep Learning.* (S. N. 2021, Ed.) Springer Cham. doi:https://doi.org/10.1007/978-3-030-75178-4

Ning, B. J. (2019). Spam message classification based on the Naïve Bayes classification algorithm. *IAENG International Journal of Computer Science 46*, 46-53.

NRC. (1998). *Nuclear Regulatory Commission, NUREG-0544, Rev. 4, "NRC: Collection of Abbreviations".*

NRC. (2012). ADAMS. NRC. Retrieved from https://www.nrc.gov/reading-rm/adams.html

OpenAI, ChatGPT. (2023). Retrieved from https://openai.com/

P. Harjule, A. G. (2020). Text Classification on Twitter Data. *2020 3rd International Conference on Emerging Technologies in Computer Engineering: Machine Learning and Internet of Things*, (pp. 160-164). doi:10.1109/ICETCE48199.2020.9091774

Pacific Northwest National Laboratory. (2021). *NukeLM: Pre-Trained and Fine-Tuned Language Models for the Nuclear and Energy Domains.*

Park, J. K. (2017). Use of a big data mining technique to extract relative importance of performance shaping factors from event investigation reports. *Advances in Human Error, Reliability, Resilience, and Performance*, 230-238. doi:https://doi.org/10.1007/978-3-319-60645-3_23

Pence, J. (2015). Quantifying Organization Factors in Human Reliability Analysis Using the Big Data-Theoretic Algorithm. *International Topical Meeting on Probabilistic Safety Assessment and Analysis PSA*.

Ramage, D. &. (2009). Labeled LDA: A supervised topic model for credit attribution in multi-labeled corpora. *roc. of the Conf. on Empirical Methods in Natural Language Processing: Volume 1*, (pp. 248-256).

Reimers, N. (2022). *Sentence Transformer Documentation: MLM*. Retrieved from SBERT.net: https://www.sbert.net/examples/unsupervised_learning/MLM/README.html

Rennie, J. D. (2003). Tackling the Poor Assumptions of Naive Bayes Text Classifiers. *Proceedings of the Twentieth International Conference on Machine Learning (ICML-2003)* (pp. 616-623). Washington DC: Artificial Intelligence Laboratory.

Robertson, R. (2016, May 16). *Omaha's Public Radio Newsroom*. Retrieved from KVNO News: https://www.kvnonews.com/2016/05/oppd-will-absorb-many-possible-fort-calhoun-station-closes/

Rush, A. (2018). The Annotated Transformer. *Proceedings of Workshop for NLP Open Source Software (NLP-OSS)*. doi:10.18653/v1/W18-2509

Sai Zhang, F. X. (2022). *Natural Language Processing-Enhanced Nuclear.* Idaho National Laboratory, U.S. Department of Energy. Idaho Falls: Idaho National Laboratory. Retrieved from https://inldigitallibrary.inl.gov/sites/sti/sti/Sort_63110.pdf

Schank, R. C. (1972). Conceptual Dependency: A Theory of Natural Language Understanding. In R. C. Schank, *Cognitive Psychology 3* (pp. 552-631). Academic Press, Inc.

Sebastian Schelter, D. L. (2018, August). Automating large-scale data quality verification. *Proceedings of the VLDB Endowment*, 1781-1794. doi:10.14778/3229863.3229867

Siddharth Suman, S. S. (2021). Artificial intelligence in nuclear industry: Chimera or solution? *Journal of cleaner production*, 278.

Šimić, Z. Z. (2015). Development and first application of an operating EVENTS ranking tool. *Nuclear Engineering and Design*, 36–43. doi:https://doi.org/10.1016/j.nucengdes.2014.11.035

spaCy. (n.d.). Retrieved from TRAINED PIPELINES: https://spacy.io/models/en

Special NRC Oversight at Fort Calhoun Station. (2022, February 23). Retrieved from U.S. NRC: https://www.nrc.gov/info-finder/reactors/fcs/special-oversight.html#event

Strubell, E. a. (2019). Energy and Policy Considerations for Deep Learning in NLP. arXiv. doi:10.48550/ARXIV.1906.02243

Sutskever, I. V. (2014). Sequence to Sequence Learning with Neural Networks. *NIPS.*

Tanaka, H. (1996). Decision tree learning algorithm with structured attributes: Application to verbal case frame acquistion. . *COLING 1996 Volume 2: The 16th International Conference on Computational Linguistics.*

Tianyi Zhang, e. a. (2021). REVISITING FEW-SAMPLE BERT FINE-TUNING. *arxiv*. Retrieved from https://arxiv.org/pdf/2006.05987.pdf

Tomoko Ohta, S. P.-W.-J.-P. (2013). Overview of the Pathway Curation (PC) task of BioNLP Shared Task. *In Proceedings of the BioNLP Shared Task 2013 Workshop* (pp. 67-75). Association for Computational Linguistics.

V. Gudivada, A. A. (2017). Data Quality Considerations for Big Data and Machine Learning: Going Beyond Data Cleaning and Transformations. *International Journal on Advances of Software*, 1-20.

Valenzuela-Escárcega, M. A.-P. (2015). A domain-independent rule-based framework for event extraction. *In Proceedings of ACL-IJCNLP 2015 System Demonstrations*, (pp. 127-132).

Victor Sanh, e. a. (2020). DistilBERT, a distilled version of BERT: smaller, faster, cheaper and lighter. *arXiv*.

Weaver, W. (1949). Weaver Memorandum. New York, New York, United States: The Rockefeller Foundation.

World Nuclear News. (2016, October 25). *Final Shutdown for Fort Calhoun*. Retrieved from World Nuclear News: https://www.world-nuclear-news.org/C-Final-shut-down-for-Fort-Calhoun-2510164.html

Xiong, M. W. (2021). Digital twin–driven aero-engine intelligent predictive maintenance. *The International Journal of Advanced Manufacturing Technology*, 114, 3751–3761.

Ye Wang, e. a. (2017). Comparisons and Selections of Features and. *IOP Conference Series: Materials Science and Engineering.*

Ye, B.-b. (2015). Case Grammar and its Application in English Vocabulary Teaching. *International Conference on Applied Social Science Research (ICASSR 2015).*

Yonghui Wu, e. (2016). Google's Neural Machine Translation System: Bridging the Gap between Human and Machine Translation. *ArXiv*. Retrieved from https://arxiv.org/abs/1609.08144v2

Yongqing, G. e. (2016). Natural Language Process: A New Kind of Nuclear Quality Assurance Management Tool. *Energy Procedia 127*, 201-219.

Yoshua Bengio, e. a. (2003). A Neural Probabilistic Language Model. *Journal of Machine Learning Research 3*, 1137-1155.

Yunfei Zhao, X. D. (2019). Automated Identification of Causal Relationships in Nuclear Power Plant Event Reports. *Nuclear Technology, 205:8*, 1021-1034. doi:10.1080/00295450.2019.1580967

ZHAO, X. D. (2018). Preliminary Study of Automated Analysis of Nuclear Power Plant Event Reports Based on Natural Language Processing Techniques. *Probabilistic Safety Assessment and Management PSAM.* Los Angeles.

Zou, Y. X. (2018). A data mining framework within the CHINESE NPPS operating experience feedback system for IDENTIFYING INTRINSIC correlations among human factors. Annals of Nuclear Energy. *Annals of Nuclear Energy*, 163-170. doi:https://doi.org/10.1016/j.anucene.2018.02.038

Appendices

A. Python Code

Definitions.py

```
"""
Maintains variables for directories
"""

import os

ROOT_DIR = os.path.dirname(os.path.abspath(__file__))
LABELED_NO_REFUEL_MORP =
os.path.join(ROOT_DIR,'resources','MORP_2_Raw_labels.csv')
RAW_MORP = os.path.join(ROOT_DIR,'resources','MORP_2.csv')

PROFILE_PATH = os.path.join(ROOT_DIR,'outputs','profileReports')
PLOT_PATH = os.path.join(ROOT_DIR,'plots')
DATA_PATH = os.path.join(ROOT_DIR,'resources')
outputPaths = [
    PROFILE_PATH,
    PLOT_PATH,
]
HOUR_LIMIT = 350
def makeDirectories(outputPaths):
    for outputPath in outputPaths:
        if not os.path.exists(outputPath):
            os.makedirs(outputPath)
        else:
            pass

makeDirectories(outputPaths)
```

IO.py

```
"""
This module reads in the raw or labeled MORP datasets
```

```
"""

import os
import pandas as pd
from definitions import
LABELED_NO_REFUEL_MORP,RAW_MORP,PROFILE_PATH,PLOT_PATH


def getMorp(csvFile):
    """
    Gets morp dataset from ``resources`` directory
    """
    morpVersions = {
        'raw':RAW_MORP,
        'labeled':LABELED_NO_REFUEL_MORP,
    }
    assert csvFile in morpVersions.keys(), f'{csvFile} is not an
expected morp file. Acceptable labels and morp datasets {morpVersions}'
    df = pd.read_csv(morpVersions[csvFile],encoding='ISO-8859-1')
    return df

def write(df,path):
    df.to_csv(path)
    return f'Wrote DataFrame to {path}'

def profiler(df,title,path=PROFILE_PATH):
    profilePath = os.path.join(path,f'{title}.html')
    if os.path.exists(profilePath):
        raise FileExistsError('This file exists. Provide new title.')
    else:
        profile = ProfileReport(df, title="MORP Profile Report")
        profile.to_file(output_file=profilePath)
        return f'{title} written to {path}'
```

```
def savePlot(title):
    pngPath = os.path.join(PLOT_PATH,title)
    try:
        if not os.path.exists(pngPath):
            return True
    except FileExistsError:
        print(f'{pngPath} exists. Please provide different title name
or delete old plot')
        return False
```

docket.py

```
"""
data class used for managing Docket objects
"""
from definitions import HOUR_LIMIT
from dataclasses import dataclass, field
from datetime import datetime

@dataclass
class Docket:
    docket:int
    date: datetime
    hoursList: list
    label: int = field(init=False)

    def __post_init__(self):
        self.getLabel(self.hoursList)

    def __str__(self):
        return f'{self.docket}'

    def getLabel(self, hoursList):
        if sum(hoursList) >= HOUR_LIMIT:
            label = 1
```

```
        else:
            label = 0
        self.label = label
        return self.label
    def unpack(self):
        docket = self.docket
        date = self.date
        hoursList = self.hoursList
        label = self.label
        return docket, date, hoursList, label
```

cleanup.py

```
"""
This module is used for cleaning and editing the MORP pandas dataframe.
"""
import pandas as pd
import numpy as np
from datetime import datetime
from modules.processing.docket import Docket




def setDateTime(df):
    df['OUTG_DATE'] = df['OUTG_DATE'].apply(lambda x:
datetime.strptime(x, '%m/%d/%y  %H:%M').date())
    return df

def outageReasons(df,reason="C"):
    r"""
    Remove all outages from OUTG_REASN of a specific type.

    reasons: list of reasons recognized as categorical value (see MORP2
Definitions)
    """
```

```
    print(df.columns)
    allOutageReasons = {
        "A": "Equipment Failure",
        "B": "Maintenance or Test",
        "C": "Refueling",
        "D": "Regulatory Restriction",
        "E": "Operator Training and License Examination",
        "F": "Administrative",
        "G": "Operational Error",
        "H": "Other",
        }

    assert reason in allOutageReasons.keys(), f"Reason {reason} is not
recognized in MORP {allOutageReasons}"
    df.query(f'OUTG_REASN != "%s"' % reason,inplace=True)
    return df



def lambdaSetLabel(dfCol,docketOutageHoursList):
    """
    Lambda function for applying severity labels to a new column
typically called ``labels``.

    Note: this function is conditionally dependent on columns,
therefore ``.apply`` should have `axis=1`
    Example: df['labels'] = df.apply(lambda x:
setLabel(x,docketOutageHoursList),axis=1)
    """
    for docketObject in docketOutageHoursList:
        docket, date,hoursList,label = docketObject.unpack()

        if (dfCol.DOCKET == docket) & (dfCol.OUTG_DATE == date):
            return label
```

```
def docketOutageHoursList(df):
    dockets = sorted(list(set(df['DOCKET'].tolist())))
    dfGroupBy = df.groupby(['DOCKET'])
    assert 'OUTG_DATE' in df.columns, 'OUTG_DATE not in DataFrame
Column'
    docketOutageHoursList = []
    for docket in dockets:
        OUTG_HOURS_SERIES =
dfGroupBy.get_group(docket).set_index('OUTG_DATE').groupby('OUTG_DATE')
['OUTG_HRS'].apply(list)
        for date,hoursList in OUTG_HOURS_SERIES.items():
            docketOutageHoursList.append(Docket(docket=docket,
date=date,hoursList=hoursList))
    return docketOutageHoursList

def dropNan(df):
    df = df[df['OUTG_HRS'].notna()]
    df = df[df['DESCRIP'].notna()]
    return df

def basicCleanUp(df):
    df = setDateTime(df)
    df = outageReasons(df)
    docketList = docketOutageHoursList(df)
    df['labels'] = df.apply(lambda col:
lambdaSetLabel(col,docketList),axis=1)
    df['DESCRIP'] = df['DESCRIP'].str.lower()
    df = df[df['OUTG_HRS'].notna()]
    return df
```

labeling.py

```
"""
This module was originally a jupyter workbook. It is used for labeling
and creating the MORP datasets by hour limits and
```

```
writing them as preLabels. These still need to undergo text processing
preparation.
"""
# %%
import pandas as pd
import math
import numpy as np
from modules.processing import IO,cleanUp
import matplotlib.pyplot as plt
import seaborn as sns
from collections import defaultdict,OrderedDict
from dataclasses import dataclass, field
import warnings
from datetime import datetime
warnings.simplefilter(action='ignore', category=FutureWarning)

# %%
def periodHours(outageHours):
    totalHours = sum([hrs for prd, hrs in outageHours])
    initialPeriod = [prd for prd,hrs in outageHours][0]
    return str(initialPeriod),totalHours

def checkNextSequence(docket_df_shift,idx,shift=True):
    if shift:
        seqShift = docket_df_shift['outageSequence'].iloc[idx]
    else:
        # shift = False if idx is in the second to last position of
docket_df
            # Last two reports
                 # docket_df    docket_df_shift
            # 0    True          False
            # 1    False         NaN
        # for example, this will return False
        seqShift = docket_df_shift['outageSequence'].iloc[idx-1]
    return seqShift
```

```
def reinit(outageRecord,outageHours,morpIndexRecord,seq):
    initialPeriod,totalHours = periodHours(outageHours=outageHours)
    if (initialPeriod in outageRecord.keys()) and (seq==False):
        initialPeriod = initialPeriod+f'-s{morpIndexRecord[0]}'
    outageRecord[initialPeriod] = (totalHours,morpIndexRecord)

    return outageRecord

# Remove all outages except Equipment failure, Maintenance or test,
Opertional Error, Other
def removeOutages(dfRaw):
    C = cleanUp.outageReasons(dfRaw)
    DC = cleanUp.outageReasons(C,reason='D')
    EDC = cleanUp.outageReasons(DC,reason='E')
    FEDC = cleanUp.outageReasons(EDC,reason='F')
    df = FEDC
    df = cleanUp.dropNan(df)
    df['DESCRIP'] = df['DESCRIP'].str.lower()
    return df

def labelMultiOutage(outgMeth):
    # Pr
    if outgMeth == 4:
        return True
    else:
        return False

def update_df_labeled(df_labeled,outageRecord):
    for k,v in outageRecord.items():
        period = k

        totalHours, morpIndexList = v
        for morpIdx in morpIndexList:
            df_labeled.at[morpIdx,'combinedOutageHours'] = totalHours
```

```
            df_labeled.at[morpIdx,'RPT_PERIOD'] = period
    return df_labeled

def grouping(docket_df,docket_df_shift,outageRecord):

    morpIndexRecord = []
    outageHours = []
    docket_df_length = range(0,len(docket_df))
    firstIdx = docket_df_length[0]
    lastIdx = docket_df_length[-1]
    secondToLastIdx = docket_df_length[-2]


    for idx, row in docket_df.iterrows():
        hours = row['OUTG_HRS']
        morpIndex = docket_df['morpIndex'].iloc[idx]
        seq = row['outageSequence']
        period = row['RPT_PERIOD']

        morpIndexRecord.append(morpIndex)
        outageHours.append((period,hours))
        if (idx == lastIdx):
            outageRecord =
reinit(outageRecord,outageHours,morpIndexRecord,seq)
            continue

        if (idx == secondToLastIdx):
            # Last two reports
            if checkNextSequence(docket_df_shift,lastIdx,shift=False)
== False:
            # Update outage record, reinit lists
                outageRecord =
reinit(outageRecord,outageHours,morpIndexRecord,seq)
                outageHours = []
                morpIndexRecord = []
```

```
                continue

        if checkNextSequence(docket_df_shift,idx) == False:
            # Update outage record, reinit lists

            outageRecord =
reinit(outageRecord,outageHours,morpIndexRecord,seq)
            outageHours = []
            morpIndexRecord = []
            continue
        else:
            continue
    return outageRecord

def applyLabel(row,hourLimit):
    if row >= hourLimit:
        return 1
    else:
        return 0



# %%
hourLimits = [744, 650,550,500,450,425,400,375,350,325,300, 275, 250]

# %%
dfRaw = IO.getMorp('raw')
dfRaw['OUTG_REASN'].value_counts()

df = removeOutages(dfRaw)
#df.to_csv('resources/finalMorpRaw.csv') #This is the final meta
cleaned MORP. It should start as the starting point for any
manipulations

# %% [markdown]
```

```
# Outage Stats

# %% [markdown]
# Aggregate SUM and MEAN

# %%
# Get aggregate mean and sum of final dataset `OUTG_HRS``
data = df.set_index(['DOCKET','OUTG_HRS'])
docketMean =
df.groupby(['DOCKET'])['OUTG_HRS'].agg(np.mean).rename('Average Hour
per Outage')
docketSum =
df.groupby(['DOCKET'])['OUTG_HRS'].agg(np.sum).rename('Total Outage
Hours')
df_agg = pd.concat([docketSum,docketMean],axis=1)
df_agg.to_csv("resources/statistics/aggregate_OUTG_HRS.csv")

# %% [markdown]
# Outage Date Sequence Processing
#
# This dumps out multiple data sets that are grouped and labeled by
outage severity. Where the outage severity limit is defined by
hourLimits = [744, 650,550,500,450,425,400,375,350,325,300, 275, 250]

# %%
df_docket_period =
df.sort_values(['DOCKET','RPT_PERIOD'])#.reset_index(drop=True)
df_docket_period['outageSequence'] =
df_docket_period['OUTG_METH'].apply(lambda row: labelMultiOutage(row))
df_docket_period =
df_docket_period.sort_values(['DOCKET','RPT_PERIOD'])
df_docket_period['combinedOutage'] = ''
df_docket_period['combinedHours'] = np.nan
df_docket_period['labels'] = 0
```

```
# %%
dfGroupBy = df_docket_period.groupby(['DOCKET'])

# %%
df_labeled = df
df_labeled['combinedOutageHours'] = 0
df_labeled['labels'] = 0
df_labeled = df_labeled.sort_values(['DOCKET','RPT_PERIOD'])

# %%
dockets = sorted(list(set(df_docket_period['DOCKET'].tolist())))

for hourLimit in hourLimits:
    df_labeled = df
    df_labeled['combinedOutageHours'] = 0
    df_labeled['labels'] = 0
    df_labeled = df_labeled.sort_values(['DOCKET','RPT_PERIOD'])

    outageRecord = defaultdict()

    for docket in dockets:
        df_rptPeriod =
dfGroupBy.get_group(docket).sort_values(['RPT_PERIOD'])#.set_index('RPT
_PERIOD')
        docket_df =
df_rptPeriod[['RPT_PERIOD','outageSequence','OUTG_HRS','labels']].reset
_index()
        docket = str(docket)
        docket_df.rename(columns={'index':'morpIndex'},inplace=True)
        docket_df['OUTG_HRS'] =
pd.to_numeric(docket_df['OUTG_HRS'],downcast='float')
        docket_df_shift = docket_df.shift(-1,axis=0)


        outageRecord = grouping(docket_df,docket_df_shift,outageRecord)
```

```
        df_labeled = update_df_labeled(df_labeled,outageRecord)


    df_labeled['labels'] =
df_labeled['combinedOutageHours'].apply(lambda row:
applyLabel(row,hourLimit))
    df_labeled.to_csv(f'resources/preLabelsByOutageHours_{hourLimit}.cs
v')




# %%
df_labeled['labels'].value_counts()

# %%
with pd.option_context('display.max_rows', None, 'display.max_columns',
None):  # more options can be specified also
    print(docket_df)


# df_labeled.to_csv(r'resources/preLabelsByOutageHours.csv')
```

textClean.py

```
"""
This module is used to clean and process the text data in MORP using
the reformatted features identified in the text corpus.
"""
```

```
import pandas as pd
import math
import numpy as np
import os
from definitions import DATA_PATH
import re

supplementalDataPath = os.path.join(DATA_PATH,r'supplements')
REFORMAT = os.path.join(supplementalDataPath,r'reformat.txt')
EXPANSIONS = os.path.join(supplementalDataPath,r'masterExpansions.txt')
UNKNOWN_WORD_NUMBERS =
os.path.join(supplementalDataPath,r'unknownWordNumbers.txt')
TRAINING_DATA = os.path.join(DATA_PATH,'trainingDatasets')

def getMasterReformat():
    masterReformat = {}
    with open(REFORMAT,"r") as f:

        for idx, line in enumerate(f):
            if idx>=2 and '-----' not in line:
                if '#' in line:
                    break
                k,v = line.split(':')[0],
line.split(':')[1].split('*')[0].strip()
                masterReformat[k] = v.upper()
    return masterReformat

def getMasterExpansions():
    masterExpansions = {}
    with open(EXPANSIONS,"r") as f:
        for idx, line in enumerate(f):
            if idx>=4 and '-' not in line:
                if '#' in line:
                    break
```

```
                k,v = line.split(':')[0],
line.split(':')[1].split('*')[0].strip()
                masterExpansions[k] = v.upper()

    return masterExpansions

def getUnknownWordNumbers():
    masterUnknownWordNumbers = []
    with open(UNKNOWN_WORD_NUMBERS,"r") as f:
        for wordNumber in f:
            masterUnknownWordNumbers.append(wordNumber.strip())

    masterUnknownWordNumbers =
sorted(masterUnknownWordNumbers,reverse=True)
    return masterUnknownWordNumbers

def getHyphensInVocab():
    hyphenDict = {}
    with open(r'…\hyphensInVocab.txt','r') as f:
        for line in f:
            items = line.split(':::')[0]
            key = items.split(':')[0]
            value = items.split(':')[1]
            if key not in hyphenDict.keys():
                hyphenDict[key] = value.upper()
    return hyphenDict

expansions = getMasterExpansions()
reformat = getMasterReformat()
unknownNumbers = getUnknownWordNumbers()
hyphensInVocab = getHyphensInVocab()

def applyReformat(pandasString):
    rowsChanged = 0
    pandasString = pandasString.upper()
```

```
    masterReformat = getMasterReformat()
    for key, value in masterReformat.items():
        if key in pandasString:
            pandasString = pandasString.replace(key, value)
            rowsChanged += 1

    return pandasString.lower()

def applyRemoveDates(pandasString):
    pattern =
r'\b\d{1,2}/\d{1,2}/\d{2,4}\b|\b\d{1,2}/\d{2}/\d{2,4}\b|\b\d{1,2}/\d{1,
2}/\d{4}\b'
    return re.sub(pattern, '', pandasString)

def applySplitWords(pandasString):
    return ' '.join(pandasString.split('/'))

def applyRemoveParenthesis(pandasString):
    return re.sub(r'\([^)]*\)', '',pandasString)

def applySplitHyphens(pandasString):
    return ' '.join(pandasString.split('-'))

def applyExpandAll(pandasString):
    for key,value in expansions.items():
        if key in pandasString.upper():
            pattern = r'\b({})\b'.format(re.escape(key))
            pandasString = re.sub(pattern, value, pandasString.upper())
    for key,value in hyphensInVocab.items():
        if key in pandasString.upper():
            pattern = r'\b({})\b'.format(re.escape(key))
            pandasString = re.sub(pattern, value, pandasString.upper())
    return pandasString.lower()

hourLimits = [744, 650,550,500,450,425,400,375,350,325,300, 275, 250]
```

```
finalPath = os.path.join(TRAINING_DATA,'finalTrainingData')
for hourLimit in hourLimits:
    dataset = f'preLabelsByOutageHours_{hourLimit}.csv'
    data = os.path.join(TRAINING_DATA,dataset)
    df = pd.read_csv(data)
    df['cleaned'] = df['DESCRIP'].apply(lambda text:
applyReformat(text))
    df['cleaned'] = df['cleaned'].apply(lambda text:
applyRemoveDates(text))
    df['cleaned'] = df['cleaned'].apply(lambda text:
applySplitWords(text))
    df['cleaned'] = df['cleaned'].apply(lambda text:
applyRemoveParenthesis(text))
    df['cleaned'] = df['cleaned'].apply(lambda text:
applySplitHyphens(text))
    df['cleaned'] = df['cleaned'].apply(lambda text:
applyExpandAll(text))
    finalDataSet = f'preLabelsByOutageHours_{hourLimit}_cleaned.csv'
    finalPath = os.path.join(r'…\finalTrainingData',finalDataSet)
    df.to_csv(finalPath)
```

morpFurtherTraining.py

```
# -*- coding: utf-8 -*-
"""
This module was originally a workbook on google collab.
It is used for mlm and nsp training of MORP with Ntext
“””

!pip install transformers
!pip install torch
!pip install torchsummary

from transformers import BertTokenizer, BertForPreTraining
```

```
import os
# BertForPreTraining has both MLM head and NSP head
import torch
from google.colab import drive
drive.mount('/content/gdrive')

model_save_name = 'NuBert.pth'
finalPath = F"/content/gdrive/MyDrive/{model_save_name}"

tokenizer = BertTokenizer.from_pretrained('NukeTokenizer')
model = BertForPreTraining.from_pretrained('bert-large-uncased')

with open("Ntext.txt","r") as f:
  paragraphs = []
  currentParagraph = ''
  for line in f:
    if line.strip() == '':
      if currentParagraph:
        paragraphs.append(currentParagraph)
        currentParagraph = ''
    else:
      currentParagraph += line.strip('\n')
  if currentParagraph:
    paragraphs.append(currentParagraph)

with open("Ntext.txt","r") as f:
  text = f.readlines()

bag = [sentence for para in paragraphs for sentence in para.split('.')
if sentence!='']
bag_size = len(bag)

import random

bag[4]
```

```
paragraphs[:3]

sentence_a = []
sentence_b = []
label = []

for paragraph in paragraphs:
  # Getting sentences in a paragraph
  sentences = [
      sentence for sentence in paragraph.split('.') if sentence != ''
  ]
  num_sentences = len(sentences)
  # This condition checks if the number of sentences in one document
(paragraph) is greater than 1.
  # The reason being, is if we are going to perform next sentence
prediction, we need to concatenate 2 sentences for the
  # training data. Therefore pulling one sentence would not have the
"next sentence" for the model to train with
  if num_sentences > 1:

    # This is where we extract sentence A

    # The next line of code ensures that not matter what random
sentence we pick, sentence B will always come after
    start = random.randint(0,num_sentences-2)
    sentence_a.append(sentences[start])

    if random.random() > 0.5:
      sentence_b.append(sentences[start+1])
      label.append(0)
    else:
      sentence_b.append(bag[random.randint(0,bag_size-1)])
      label.append(1)
```

```
tokenizer = BertTokenizer.from_pretrained('NukeTokenizer')

inputs =
tokenizer(sentence_a,sentence_b,return_tensors='pt',max_length=128,trun
cation=True,padding='max_length')

inputs.keys()

inputs['next_sentence_label'] = torch.LongTensor([label]).T

inputs['next_sentence_label'][:10
]

inputs['labels'] = inputs.input_ids.detach().clone()

inputs.keys()

rand = torch.rand(inputs.input_ids.shape)

mask_arr = (rand < 0.15) * (inputs.input_ids != 101) *
(inputs.input_ids != 102) * (inputs.input_ids != 0)

for I in range(inputs.input_ids.shape[0]):
  selection = torch.flatten(mask_arr[i].nonzero()).tolist()
  inputs.input_ids[I,selection] = 103

class NukeDataset(torch.utils.data.Dataset):
  def __init__(self,encodings):
    self.encodings = encodings
  def __getitem__(self,idx):
    return {key: torch.tensor(val[idx]) for key, val in
self.encodings.items()}
  def __len__(self):
    return len(self.encodings.input_ids)
```

```
dataset = NukeDataset(inputs)

loader = torch.utils.data.DataLoader(dataset,
batch_size=16,shuffle=True)

device = torch.device('cuda') if torch.cuda.is_available() else
torch.device('cpu')

model.to(device)

model.train()

from transformers import AdamW
optim = AdamW(model.parameters(),lr=5e-5)

model.config.vocab_size
model.resize_token_embeddings(len(tokenizer))

from tqdm import tqdm
for epoch in range(2):
  loop = tqdm(loader, leave=True)
  for batch in loop:
    optim.zero_grad()
    input_ids = batch['input_ids'].to(device)
    token_type_ids = batch['token_type_ids'].to(device)
    attention_mask = batch['attention_mask'].to(device)
    next_sentence_label = batch['next_sentence_label'].to(device)
    labels = batch['labels'].to(device)
    outputs = model(input_ids,
                    token_type_ids=token_type_ids,
                    attention_mask=attention_mask,
                    next_sentence_label=next_sentence_label,
                    labels=labels)
    loss = outputs.loss
    loss.backward()
```

```
    optim.step()

    loop.set_description(f'Epoch {epoch}')
    loop.set_postfix(loss=loss.item())



# torch.save(model.state_dict(),"checkpoint.pth")
```

finalFineTune.py

```
# -*- coding: utf-8 -*-
"""
This module was originally a workbook on google collab.
It is used for fine-tuning transformer based Bert models for binary
classification and writing results.
"""

!pip install tensorflow
!pip install transformers
!pip install -U imbalanced-learn

import tensorflow as tf
import numpy as np
from transformers import TFBertForSequenceClassification,
BertTokenizer,AdamW,
AutoTokenizer,AutoModelForSequenceClassification,TFAutoModelForSequence
Classification,
DistilBertTokenizerFast,TFDistilBertForSequenceClassification
from tqdm import tqdm
import os
import pandas as pd
from sklearn.model_selection import train_test_split
import matplotlib.pyplot as plt
```

```
from sklearn.metrics import classification_report, accuracy_score,
confusion_matrix,roc_curve,auc,recall_score, precision_score,
f1_score,cohen_kappa_score,matthews_corrcoef,log_loss
from scipy.special import softmax
from google.colab import drive
import random
from tensorflow.keras.callbacks import EarlyStopping, ModelCheckpoint
from tensorflow.keras.optimizers.schedules import PolynomialDecay
from tensorflow.keras.optimizers import Adam
import json
from google.colab import files
# Set the random seed for Hugging Face Transformers
random.seed(50)
np.random.seed(50)
tf.random.set_seed(50)

print("Num GPUs Available: ",
len(tf.config.list_physical_devices('GPU')))
print(tf.__version__)
print(tf.config.list_physical_devices('GPU'))

"""*Inputs*"""

drive.mount('/content/gdrive')
gDrivePath = r'/content/gdrive/MyDrive/'

checkpointName = r'checkpointMorpy.pth'
nukeTokenizer = r'nukeTokenizer'
morpyFinalTrain = r'morpyFinalTrain'
batch16_maxLength512 = r'batch16_maxLength512'
batch96_maxLength128 = r'batch96_maxLength128'
batch32_maxLength512_bertBase = f'batch32_maxLength512_bertBase'

batch96_maxLength128_distillBert = 'batch96_maxLength128_distillBert'
batch32_maxLength512_distillBert = 'batch32_maxLength512_distillBert'
```

```
batch16_maxLength512_distillBert = 'batch16_maxLength512_distillBert'
batch32_maxLength128_distillBert = 'batch32_maxLength128_distillBert'
batch16_maxLength128_distillBert = 'batch16_maxLength128_distillBert'

nukeTokenizerPath = os.path.join(gDrivePath,nukeTokenizer)
# Morp Data
clean = os.path.join(gDrivePath,r'datasets',r'cleanDatasets')
unclean = os.path.join(gDrivePath,r'datasets',r'uncleanDatasets')

dataPaths = {
    'clean':clean,
    'unclean':unclean,
  }

# Tokenizer
bertTokenizer = BertTokenizer.from_pretrained('bert-large-uncased')
nukeTokenizer = AutoTokenizer.from_pretrained(nukeTokenizerPath)
distilBertTokenizer =
DistilBertTokenizerFast.from_pretrained('distilbert-base-uncased')

# Models
morpyFinalTrainPath = os.path.join(gDrivePath,morpyFinalTrain)
batch16_maxLength512Path =
os.path.join(gDrivePath,batch16_maxLength512)
batch32_maxLength512_bertBasePath =
os.path.join(gDrivePath,batch32_maxLength512_bertBase)
batch96_maxLength128Path =
os.path.join(gDrivePath,batch96_maxLength128)
batch96_maxLength128_distillBertPath =
os.path.join(gDrivePath,batch96_maxLength128_distillBert)

batch32_maxLength512_distillBertPath =
os.path.join(gDrivePath,batch32_maxLength512_distillBert)
batch16_maxLength512_distillBertPath =
os.path.join(gDrivePath,batch16_maxLength512_distillBert)
```

```
batch32_maxLength128_distillBertPath =
os.path.join(gDrivePath,batch32_maxLength128_distillBert)
batch16_maxLength128_distillBertPath =
os.path.join(gDrivePath,batch16_maxLength128_distillBert)
bertVanilla = 'bert-large-uncased'
distilBertVanilla = 'distilbert-base-uncased'


Models = {
    'batch16_maxLength512': batch16_maxLength512Path,
    'batch32_maxLength512_bertBase': batch32_maxLength512_bertBasePath,
    'batch96_maxLength128': batch96_maxLength128Path,
    'batch96_maxLength128_distillBert':
batch96_maxLength128_distillBertPath,
    'bertVanilla': bertVanilla,
    'batch32_maxLength512_distillBert_nuke':
batch32_maxLength512_distillBertPath,
    'batch16_maxLength512_distillBert_nuke':
batch16_maxLength512_distillBertPath,
    'batch32_maxLength128_distillBert_nuke':
batch32_maxLength128_distillBertPath,
    'batch16_maxLength128_distillBert_nuke':
batch16_maxLength128_distillBertPath,
    'distilBertVanilla': distilBertVanilla,
    }
tokenizerChoices = {
    'bert':bertTokenizer,
    'nuke':nukeTokenizer,
    'distil':distilBertTokenizer
  }

# Data, Tokenizer & Model Choice
dataPath = dataPaths['clean']
modelWeights = Models['batch16_maxLength128_distillBert_nuke']
tokenizerChoice = tokenizerChoices['distil']
```

```
if dataPath == clean:
  morpLabels744 = r'preLabelsByOutageHours_744_cleaned.csv'
  morpLabels650 = r'preLabelsByOutageHours_650_cleaned.csv'
  morpLabels550 = r'preLabelsByOutageHours_550_cleaned.csv'
  morpLabels500 = r'preLabelsByOutageHours_500_cleaned.csv'
  morpLabels450 = r'preLabelsByOutageHours_450_cleaned.csv'
  morpLabels400 = r'preLabelsByOutageHours_400_cleaned.csv'
  morpLabels375 = r'preLabelsByOutageHours_375_cleaned.csv'
  morpLabels350 = r'preLabelsByOutageHours_350_cleaned.csv'
  morpLabels325 = r'preLabelsByOutageHours_325_cleaned.csv'
  morpLabels300 = r'preLabelsByOutageHours_300_cleaned.csv'
  morpLabels275 = r'preLabelsByOutageHours_275_cleaned.csv'
  morpLabels250 = r'preLabelsByOutageHours_250_cleaned.csv'
else:
  morpLabels744 = r'preLabelsByOutageHours_744.csv'
  morpLabels650 = r'preLabelsByOutageHours_650.csv'
  morpLabels550 = r'preLabelsByOutageHours_550.csv'
  morpLabels500 = r'preLabelsByOutageHours_500.csv'
  morpLabels450 = r'preLabelsByOutageHours_450.csv'
  morpLabels400 = r'preLabelsByOutageHours_400.csv'
  morpLabels375 = r'preLabelsByOutageHours_375.csv'
  morpLabels350 = r'preLabelsByOutageHours_350.csv'
  morpLabels325 = r'preLabelsByOutageHours_325.csv'
  morpLabels300 = r'preLabelsByOutageHours_300.csv'
  morpLabels275 = r'preLabelsByOutageHours_275.csv'
  morpLabels250 = r'preLabelsByOutageHours_250.csv'

morpLabels = {
    '744': morpLabels744,
              # morpLabels650,
              # morpLabels550,
    '500': morpLabels500,
              # morpLabels450,
    '400': morpLabels400,
```

```
                # morpLabels375,
    '350': morpLabels350,
                # morpLabels325,
    '300': morpLabels300,
                # morpLabels275,
    '250': morpLabels250,
    }
dataHourLimit = morpLabels['250'] # 744, 500, 400, 350, 300, 250

dataClasses = 'binary'
num_epochs = 20
num_labels = 2
batch_size = 16 # [16, 32, 48, 64] Control [16]
max_length = 128 # Control [128]
padding = True
truncation = True
return_tensors = 'tf'
epsilon = 1e-08

if tokenizerChoice == nukeTokenizer:
  tokenizerName = 'NukeTokenizer'
if tokenizerChoice == distilBertTokenizer:
  tokenizerName = 'DistilBertTokenizer'
else:
  tokenizerName = 'BertTokenizer'

tokenizer = tokenizerChoice

def underSampleMild(df,N):
    """
    Removes N random samples from data pool for mild cases.

    Pass in DataFrame from getDataWithoutRefueling()
    """
    underSampledDf = df.drop(df[df['labels'].eq(0)].sample(N).index)
```

```
    print('Initial dataframe target
labels:\n{}'.format(df['labels'].value_counts()))
    print('\n')
    print('Undersampled dataframe target
labels:\n{}'.format(underSampledDf['labels'].value_counts()))

    print('Average amount of words of "text input" in this under
sampled dataset is
{0:.0f}.'.format(np.mean(underSampledDf['DESCRIP'].apply(lambda x:
len(x.split())))))
    print('Max amount of words in "text input" in this under sampled
dataset is
{0:.0f}.'.format(np.max(underSampledDf['DESCRIP'].apply(lambda x:
len(x.split())))))
    print('Max character length of "text input" in this under sampled
dataset is
{0:.0f}.'.format(np.max(underSampledDf['DESCRIP'].apply(lambda x:
len(x)))))
    print('Average character length of "text input" in this under
sampled dataset is
{0:.0f}.'.format(np.mean(underSampledDf['DESCRIP'].apply(lambda x:
len(x)))))
    print('\n')

    underSampledDf.plot(kind='bar')

    return underSampledDf

morpData = os.path.join(gDrivePath,dataPath,dataHourLimit)
print(morpData)
df = pd.read_csv(morpData,encoding="ISO-8859-1")
if dataPath == unclean:
  df['DESCRIP'] = df['DESCRIP'].str.lower().astype(str)
else:
```

```
  df['cleaned'] = df['cleaned'].str.lower().astype(str)

rootFolder = os.path.join(f'{modelWeights}-{tokenizerName}-
{batch_size}-{max_length}-{dataPath.split("/")[-1].split(".")[0]}-
runs')
if not os.path.exists(rootFolder):
  os.mkdir(rootFolder)

rootFolder

if dataClasses == 'binary':
  loss = tf.keras.losses.BinaryCrossentropy(from_logits=True)
else:
  loss =
tf.keras.losses.SparseCategoricalCrossentropy(from_logits=True)

def classWeights(colLabels):
  mild, spicy = np.bincount(colLabels)#bincount
  total = mild + spicy
  print('Examples:\n  Total: {}\n  Spicy: {} ({:.2f}% of
total)\n'.format(total, spicy, 100*spicy/total))
  weight0 = (1/mild)*(total/2.0)
  weight1 = (1/spicy)*(total/2.0)
  class_weight = {0:(weight0), 1:weight1} # + (weight1*.05)
  print('Weight for Mild: {:.2f}'.format(weight0))
  print('Weight for Spicy: {:.2f}'.format(weight1))
  return class_weight

def trainTestSplit(df):
    if dataPath == unclean:
      col = 'DESCRIP'
    else:
      col = 'cleaned'
```

```
    duplicated_rows = df[df.duplicated(subset=[f'{col}'],keep=False)] #
duplicates
    df_unique = df.drop_duplicates(subset=[f'{col}'], keep=False) #
originals

    train_df, test_df =
train_test_split(df_unique,test_size=0.15,stratify=df_unique['labels'],
random_state=42) # split into temporary train and test
    train_df, val_df = train_test_split(train_df,test_size=0.15,
stratify=train_df['labels'],random_state=42) # split into train and val

    print(f'Lengths')
    print(f'train_df {len(train_df)}')
    print(f'test_df {len(test_df)}')
    print(f'val_df {len(val_df)}')

    combined_train_df = pd.concat([train_df,duplicated_rows],
ignore_index=True) # combine duplicated instances with train set

    shuffled_combined_train_df =
combined_train_df.sample(frac=1,random_state=42) # Shuffle again
    shuffled_combined_train_df.reset_index(drop=True,inplace=True)

    train_texts = shuffled_combined_train_df[f'{col}']
    train_labels = shuffled_combined_train_df['labels']

    test_texts = test_df[f'{col}']
    test_labels = test_df['labels']

    val_texts = val_df[f'{col}']
    val_labels = val_df['labels']

    return train_texts, test_texts, val_texts, train_labels,
test_labels, val_labels
```

```
class_weight = classWeights(df['labels'])
train_texts, test_texts, val_texts, train_labels, test_labels,
val_labels = trainTestSplit(df)

train_text = train_texts.tolist()
train_inputs = tokenizer(train_text, padding=padding,
truncation=truncation, max_length=max_length,
return_tensors=return_tensors)
if dataClasses == 'binary':
  train_labels = np.array(train_labels,dtype=np.int64)
else:
  train_labels = np.array(train_labels,dtype=np.int64)


test_text = test_texts.tolist()
test_inputs = tokenizer(test_text, padding=padding,
truncation=truncation, max_length=max_length,
return_tensors=return_tensors)
if dataClasses == 'binary':
  test_labels = np.array(test_labels,dtype=np.int64)
else:
  test_labels = np.array(test_labels,dtype=np.int64)

val_text = val_texts.tolist()
val_inputs = tokenizer(val_text, padding=True, truncation=True,
max_length=128, return_tensors='tf')
if dataClasses == 'binary':
  val_labels = np.array(val_labels,dtype=np.int64)
else:
  val_labels = np.array(val_labels,dtype=np.int64)

train_inputs.keys()
```

```
#print(f'{train_text[190]}\n{train_inputs.input_ids[190]}\n{train_input
s.token_type_ids[190]}\n{train_inputs.attention_mask[190]}\n{train_labe
ls[190]}')

tokenizer.decode(train_inputs.input_ids[190])

#tokenizer.decode(train_inputs.token_type_ids[190])

tokenizer.decode(train_inputs.attention_mask[190])

# steps_per_epoch = len(train_inputs['input_ids'])/batch_size
# num_train_steps = steps_per_epoch * num_epochs
# num_warmup_steps = int(0.1*num_train_steps)

# optimizer = optimization.create_optimizer(init_lr=learning_rate,
#                                           num_train_steps=num_train_steps,
#                                           num_warmup_steps=num_warmup_step
s,
#                                           optimizer_type='adamw')
es = EarlyStopping(monitor='val_loss',
                   verbose=1, # Prints outputs
                   patience=10, # Waits 6 epochs for an improvement
                   restore_best_weights=True) # Restores model back to
best epoch

num_train_steps = (train_texts.shape[0] // batch_size) * num_epochs
print(num_train_steps)

# This is actually a linear decay from 5x10^-5 to 1x10^-5, not actually
polynomial!
if modelWeights == bertVanilla:
  initial_learning_rate = 3e-5
else:
  initial_learning_rate =5e-5
lr_scheduler = PolynomialDecay(
```

```
    initial_learning_rate=initial_learning_rate, # Start point
    end_learning_rate=1e-5, # End point
    decay_steps=num_train_steps
    )
new_opt = Adam(learning_rate=lr_scheduler)
# Load the BERT tokenizer and model
if tokenizerChoice == distilBertTokenizer:
  model =
TFDistilBertForSequenceClassification.from_pretrained(modelWeights,num_
labels=num_labels,from_pt=True)
else:
  model =
TFAutoModelForSequenceClassification.from_pretrained(modelWeights,num_l
abels=num_labels,from_pt=True)

# optimizer = tf.keras.optimizers.Adam(learning_rate=learning_rate,
#                                      epsilon=epsilon)

# metric = tf.keras.metrics.BinaryCrossentropy('accuracy')
metrics = tf.metrics.BinaryAccuracy()
model.compile(optimizer=new_opt,
metrics=[metrics],weighted_metrics=['accuracy'])
model.summary()

history = model.fit(dict(train_inputs), train_labels,
epochs=num_epochs,
batch_size=batch_size,validation_data=(dict(val_inputs),val_labels),cla
ss_weight=class_weight,callbacks=[es])

# Define the class names
if num_labels == 2:
  class_names = ['Class 0', 'Class 1']
  target_names = ['class 0', 'class 1']
  ticks = [0,1]
```

```
else:
  class_names = ['Class 0', 'Class 1', 'Class 2']
  target_names = ['class 0', 'class 1', 'class 2']
  ticks = [0, 1, 2]


# Plot the histograms
fig, axs = plt.subplots(1, 3, figsize=(15, 5))
fig.subplots_adjust(hspace=0.5, wspace=0.4)
colors = ['b','r','g']
axs = axs.ravel()
for i, labels in enumerate([train_labels, val_labels, test_labels]):
#val_labels,
    axs[i].hist(labels, bins=3, edgecolor='black',
align='mid',color=colors[i])
    axs[i].set_xticks(ticks)
    axs[i].set_xticklabels(class_names)
    axs[i].set_xlabel('Class')
    axs[i].set_ylabel('Frequency')
    axs[i].set_title('Distribution of Class Labels in ' +
['Training','Validation', 'Test'][i] + ' set')# 'Validation'

# Add a legend
handles, labels = axs[0].get_legend_handles_labels()
fig.legend(handles, labels, loc='upper right')

# Evaluate the model on the test set
test_scores = model.evaluate(dict(test_inputs), test_labels, verbose=0)
test_loss = test_scores[0]
test_accuracy = test_scores[1]

# Print the evaluation metrics
print(f'Test loss: {test_loss:.4f}')
print(f'Test accuracy: {test_accuracy:.4f}')
```

```
# %%
#test_pred_labels = np.argmax(model.predict(dict(test_inputs)), axis=1)
test_pred_probs = model.predict(dict(test_inputs))
logits = test_pred_probs.logits

# %%
test_pred_labels = np.argmax(logits, axis=1)
print(classification_report(test_labels, test_pred_labels,
target_names=target_names))

# %%
conf_mat = confusion_matrix(test_labels, test_pred_labels)

# Plot the confusion matrix
plt.figure(figsize=(8,8))
plt.imshow(conf_mat, cmap=plt.cm.Blues)
plt.title('Confusion Matrix')
plt.colorbar()
plt.xlabel('Predicted Labels')
plt.ylabel('True Labels')
plt.xticks(ticks, target_names)
plt.yticks(ticks, target_names)
plt.show()

tn, fp, fn, tp = confusion_matrix(test_labels,
test_pred_labels).ravel()

"""SUMMARY"""

results = {
    "False Positive Rate: Probability that a false alarm will be
raised: that a positive result will be given when the true value is
negative":fp/(fp+tn),
    "False Negative Rate: Probability that a true positive will be
missed by the test":fn/(tp+fn),
```

```
    "True Positive Rate | Recall | Sensitivity: Probability that an
actual positive will test positive":
recall_score(test_labels,test_pred_labels),
    "True Negative Rate | Specificity: Probability that an actual
negative will test negative": tn/(tn+fp),
    "Negative Predictive Value: Likelihood that a mild outage is truly
a mild outage": tn/(tn+fn),
    "Positive Predictive Value: Likelihood that a spicy outage is truly
a spicy outage": precision_score(test_labels,test_pred_labels),
    "F1 Score: Harmonic mean between precision and recall":
f1_score(test_labels,test_pred_labels),
    "Cohen Kappa Score: How much better is your model over the random
classifier that predicts based on class frequencies":
cohen_kappa_score(test_labels,test_pred_labels),
    "Matthews Correlation Coefficient: Correlation between predicted
classes and ground
truth":matthews_corrcoef(test_labels,test_pred_labels),
    "Log Loss: The difference between ground truth and predicted score
for every observation and average those errors over all observations":
log_loss(test_labels, test_pred_labels),
}

def
summary(test_labels,test_pred_labels,morpData,modelWeights,tokenizerNam
e,save=True):
  morpData = os.path.basename(morpData).split('.')[0]
  modelWeights = os.path.basename(modelWeights).split('.')[0]
  reportDetails = '{model} {data}
{tokenizer}'.format(model=modelWeights,data=morpData,tokenizer=tokenize
rName)
  print('{:-^150}'.format('SUMMARY'))
  print('\n')
  print(f'Morp Data: {os.path.basename(morpData)}\n')
  print(f'Confusion Matrix: \n
{confusion_matrix(test_labels,test_pred_labels)}')
```

```
  print('\n')
  for test,score in results.items():
    string_format = '{}: \n{}'.format(test,score)
    print(string_format)
    print('\n')
    print('*'*45)
  print('{:-^145}'.format('END SUMMARY'))
  if save:
    if num_labels == 2:
      class_names = ['Class 0', 'Class 1']
      target_names = ['class 0', 'class 1']
      ticks = [0,1]
    #test_pred_labels = np.argmax(model.predict(dict(test_inputs)),
axis=1)
    test_pred_probs = model.predict(dict(test_inputs))
    logits = test_pred_probs.logits
    test_pred_labels = np.argmax(logits, axis=1)
    conf_mat = confusion_matrix(test_labels, test_pred_labels)
    plt.figure(figsize=(8,8))
    plt.imshow(conf_mat, cmap=plt.cm.Blues)
    plt.title('Confusion Matrix')
    plt.colorbar()
    plt.xlabel('Predicted Labels')
    plt.ylabel('True Labels')
    plt.xticks(ticks, target_names)
    plt.yticks(ticks, target_names)
    plt.savefig(os.path.join(rootFolder,f'Confusion Matrix:
{reportDetails}.png'))
    with open(os.path.join(rootFolder,f'Summary Report:
{reportDetails}.txt'),'w') as sum:
      sum.write('{:-^150}'.format('SUMMARY'))
      sum.write('\n')
      sum.write(f'Morp Data: {os.path.basename(morpData)}\n')
      sum.write(f'Tokenizer: {tokenizerName}\n')
      sum.write(f'Pre-Trained Weights: {modelWeights}\n')
```

```
      sum.write(f'Confusion Matrix: \n
{confusion_matrix(test_labels,test_pred_labels)}')
      sum.write('\n')
      for test,score in results.items():
        string_format = '{}: \n{}'.format(test,score)
        sum.write(string_format)
        sum.write('\n')
        sum.write('*'*10)
        sum.write('\n')
      sum.write('\n')
      sum.write('{:-^145}'.format('END SUMMARY'))

summary(test_labels,test_pred_labels,morpData,modelWeights,tokenizerNam
e)

history.history.keys()

history_dict = history.history

print(history_dict.keys())
reportDetails = '{model} {data}
{tokenizer}'.format(model=modelWeights,data=morpData,tokenizer=tokenize
rName)
acc = history_dict['binary_accuracy']
val_acc = history_dict['val_binary_accuracy']
loss = history_dict['loss']
val_loss = history_dict['val_loss']

epochs = range(1, len(acc) + 1)
fig = plt.figure(figsize=(10, 6))
fig.tight_layout()

plt.subplot(2, 1, 1)
# r is for "solid red line"
plt.plot(epochs, loss, 'r', label='Training loss')
```

```
# b is for "solid blue line"
plt.plot(epochs, val_loss, 'b', label='Validation loss')
plt.title('Training and validation loss')
# plt.xlabel('Epochs')
plt.ylabel('Loss')
plt.legend()

plt.subplot(2, 1, 2)
plt.plot(epochs, acc, 'r', label='Training acc')
plt.plot(epochs, val_acc, 'b', label='Validation acc')
plt.title('Training and validation accuracy')
plt.xlabel('Epochs')
plt.ylabel('Accuracy')
plt.legend(loc='lower right')
fileName = f'loss-curves-{tokenizerName}-{dataPath.split("/")[-
1].split(".")[0]}-{dataHourLimit.split(".")[0]}.png'
plt.savefig(os.path.join(rootFolder,fileName))

# Get the dictionary containing each metric and the loss for each epoch
history_dict = history.history
# Save it under the form of a json file
jsonPath = os.path.join(rootFolder,f'{modelWeights}-{tokenizerName}-
{dataPath.split("/")[-1].split(".")[0]}-
{dataHourLimit.split(".")[0]}.history')
json.dump(history_dict,open(jsonPath , 'w'))

import shutil

if dataHourLimit == morpLabels['250']:
  # !zip -r /content/{rootFolder} . -i /content/{rootFolder}.zip
  shutil.make_archive(rootFolder, 'zip', rootFolder)
```

finalNB.py

```
"""
```

```
This module was originally a jupyter notebook.
It is used for Naive bayes classification.
"""

import pandas as pd
import nltk
import re
root = r'…\resources\trainingDataSets\finalTrainingData\clean\data'
from nltk.corpus import stopwords
nltk.download('stopwords')
from nltk.tokenize import word_tokenize
from collections import Counter

def trainTestSplit(df,nb=False):
    if nb:
      col = 'cleaned'
    # if dataPath == unclean:
    #   col = 'DESCRIP'
    # else:
    #   col = 'cleaned'

    duplicated_rows = df[df.duplicated(subset=[f'{col}'],keep=False)] #
duplicates
    df_unique = df.drop_duplicates(subset=[f'{col}'], keep=False) #
originals

    train_df, test_df =
train_test_split(df_unique,test_size=0.15,stratify=df_unique['labels'],
random_state=42) # split into temporary train and test
    train_df, val_df = train_test_split(train_df,test_size=0.15,
stratify=train_df['labels'],random_state=42) # split into train and val

    print(f'Lengths')
    print(f'train_df {len(train_df)}')
    print(f'test_df {len(test_df)}')
```

```
    print(f'val_df {len(val_df)}')

    combined_train_df = pd.concat([train_df,duplicated_rows],
ignore_index=True) # combine duplicated instances with train set

    shuffled_combined_train_df =
combined_train_df.sample(frac=1,random_state=42) # Shuffle again
    shuffled_combined_train_df.reset_index(drop=True,inplace=True)

    train_texts = shuffled_combined_train_df[f'{col}']
    train_labels = shuffled_combined_train_df['labels']

    test_texts = test_df[f'{col}']
    test_labels = test_df['labels']

    val_texts = val_df[f'{col}']
    val_labels = val_df['labels']

    if nb:
      stop_words = stopwords.words('english')
      stopwordsDict = Counter(stop_words)
      final_train_texts = []
      final_test_texts = []

      combined_test_df = pd.concat([test_df,val_df], ignore_index=True)
# combine duplicated instances with train set
      shuffled_combined_test_df =
combined_test_df.sample(frac=1,random_state=42) # Shuffle again
      shuffled_combined_test_df.reset_index(drop=True,inplace=True)
      test_texts = shuffled_combined_test_df[f'{col}']
      test_labels = shuffled_combined_test_df['labels']

      print(f'Initial Lengths of train_texts {len(train_texts)}')
      print(f'Initial Lengths of test_texts {len(test_texts)}')
```

```
      for text in train_texts:
        texts = []
        for word in text.split():
              if word not in stopwordsDict:
                 texts.append(word)
        texts = ' '.join(texts)
        texts = re.sub(" \d+",'',texts)
        texts = re.sub(r"[^a-zA-Z0-9]+",' ',texts)
        texts = re.sub(r"[^\w\s]",' ',texts)
        texts = re.sub(r"[1-9]",' ',texts)
        final_train_texts.append(texts)

      for text in test_texts:
        texts = []
        for word in text.split():
              if word not in stopwordsDict:
                 texts.append(word)
        texts = ' '.join(texts)
        texts = re.sub(" \d+",'',texts)
        texts = re.sub(r"[^a-zA-Z0-9]+",' ',texts)
        texts = re.sub(r"[^\w\s]",' ',texts)
        texts = re.sub(r"[1-9]",' ',texts)
        final_test_texts.append(texts)
      print(f'Final Lengths of train_texts {len(final_train_texts)}')
      print(f'Final Lengths of test_texts {len(final_test_texts)}')



      return final_train_texts, final_test_texts,
train_labels,test_labels

    else:
      return train_texts, test_texts, val_texts, train_labels,
test_labels, val_labels
```

```
# %%
from sklearn.model_selection import train_test_split
from sklearn.feature_extraction.text import TfidfVectorizer,
CountVectorizer
import pandas as pd
import os

from sklearn.metrics import classification_report, accuracy_score,
confusion_matrix,roc_curve,auc,recall_score, precision_score,
f1_score,cohen_kappa_score,matthews_corrcoef,log_loss
from sklearn.naive_bayes import BernoulliNB, MultinomialNB,
ComplementNB
from sklearn.metrics import confusion_matrix,
ConfusionMatrixDisplay,accuracy_score, classification_report,
balanced_accuracy_score

from sklearn.model_selection import GridSearchCV
from sklearn.model_selection import cross_val_score
import numpy as np
# Build basic model
nbResults = r'…\resources\results\naiveBayes'
morpLabels744 = r'preLabelsByOutageHours_744_cleaned.csv'
morpLabels650 = r'preLabelsByOutageHours_650_cleaned.csv'
morpLabels550 = r'preLabelsByOutageHours_550_cleaned.csv'
morpLabels500 = r'preLabelsByOutageHours_500_cleaned.csv'
morpLabels450 = r'preLabelsByOutageHours_450_cleaned.csv'
morpLabels400 = r'preLabelsByOutageHours_400_cleaned.csv'
morpLabels375 = r'preLabelsByOutageHours_375_cleaned.csv'
morpLabels350 = r'preLabelsByOutageHours_350_cleaned.csv'
morpLabels325 = r'preLabelsByOutageHours_325_cleaned.csv'
morpLabels300 = r'preLabelsByOutageHours_300_cleaned.csv'
morpLabels275 = r'preLabelsByOutageHours_275_cleaned.csv'
```

```
morpLabels250 = r'preLabelsByOutageHours_250_cleaned.csv'

morpLabels = {
    '744': morpLabels744,
              # morpLabels650,
              # morpLabels550,
    '500': morpLabels500,
              # morpLabels450,
    '400': morpLabels400,
              # morpLabels375,
    '350': morpLabels350,
              # morpLabels325,
    '300': morpLabels300,
              # morpLabels275,
    '250': morpLabels250,
    }
dataHourLimit = morpLabels['744'] # 744, 500, 400, 350, 300, 250

def
buildModel(X_train,y_train,X_test,y_test,alpha=1.0,model=None,hourLimit
=False,vect=False):
    if model == 'bernoulli':
        vect = 'countVectorizer'
        if vect == 'tfidfVectorizer':
            bow = TfidfVectorizer()
        if vect == 'countVectorizer':
            bow = CountVectorizer(binary=True)

        X_train = bow.fit_transform(X_train) # 90% slit
        X_test = bow.transform(X_test)  # 10% split
        bnb = BernoulliNB(alpha = 1.4295)
        scores = cross_val_score(bnb,X_train,y_train)
        print(f"Scores from Cross-Validation {scores}. Average scores
is {np.mean(scores)}")
```

```
        BNB = BernoulliNB(alpha = 1.4295)
        BNB.fit(X_train,y_train)
        y_pred = BNB.predict(X_test)
        y_pred_train = BNB.predict(X_train)

        print("\n")
        print('Model accuracy score: {0:0.4f}'.
format(accuracy_score(y_test, y_pred)))
        print('Training-set accuracy score: {0:0.4f}'.
format(accuracy_score(y_train, y_pred_train)))
        print("\n")
        print('Training set score: {:.4f}'.format(BNB.score(X_train,
y_train)))
        print('Test set score: {:.4f}'.format(BNB.score(X_test,
y_test)))
        print('Balance Accuracy:
{:.4f}'.format(balanced_accuracy_score(y_test, y_pred)))

        resultPath = os.path.join(nbResults+'-bernoulli')
        if not os.path.exists(resultPath):
            os.makedirs(resultPath)
        report = classification_report(y_test, y_pred)
        with open(os.path.join(resultPath,f'classificationReport-
{model}-{hourLimit}-{vect}.txt'),'w') as f:
            for line in report:
                f.write(f'{line}')
        reportd = classification_report(y_test,
y_pred,output_dict=True)
        print(report)
        report = pd.DataFrame(reportd).transpose()
        report.to_csv(os.path.join(resultPath,f'{model}-
{hourLimit}.csv'))
        tn, fp, fn, tp = confusion_matrix(y_test, y_pred).ravel()
        results = {
```

```
    "False Positive Rate: Probability that a false alarm will be
raised: that a positive result will be given when the true value is
negative":fp/(fp+tn),
    "False Negative Rate: Probability that a true positive will be
missed by the test":fn/(tp+fn),
    "True Positive Rate | Recall | Sensitivity: Probability that an
actual positive will test positive": recall_score(y_test,y_pred),
    "True Negative Rate | Specificity: Probability that an actual
negative will test negative": tn/(tn+fp),
    "Negative Predictive Value: Likelihood that a mild outage is truly
a mild outage": tn/(tn+fn),
    "Positive Predictive Value: Likelihood that a spicy outage is truly
a spicy outage": precision_score(y_test,y_pred),
    "F1 Score: Harmonic mean between precision and recall":
f1_score(y_test,y_pred),
    "Cohen Kappa Score: How much better is your model over the random
classifier that predicts based on class frequencies":
cohen_kappa_score(y_test,y_pred),
    "Matthews Correlation Coefficient: Correlation between predicted
classes and ground truth":matthews_corrcoef(y_test,y_pred),
    "Log Loss: The difference between ground truth and predicted score
for every observation and average those errors over all observations":
log_loss(y_test,y_pred),
    "Balanced Accuracy":balanced_accuracy_score(y_test, y_pred),

}
        with open(os.path.join(resultPath,f'metrics-{model}-
{hourLimit}-CountVectorizer.txt'),'w') as sum:
            sum.write('{:-^150}'.format('SUMMARY'))
            sum.write(f'Confusion Matrix: \n
{confusion_matrix(y_test,y_pred)}')
            sum.write('\n')
            for test,score in results.items():
                string_format = '{}: \n{}'.format(test,score)
                sum.write(string_format)
```

```
                sum.write('\n')
                sum.write(f'Cross-validation scores {scores}. Average
Score is {np.mean(scores)}')
                sum.write('\n')
                sum.write('*'*10)
                sum.write('\n')
            sum.write('\n')
            sum.write('{:-^145}'.format('END SUMMARY'))
        for k,v in results.items():print(k,v)
        reportDepth = pd.DataFrame.from_dict(results, orient='index')
        reportDepth.to_csv(os.path.join(resultPath,f'{model}-
{hourLimit}-reportDepth.csv'))
        target_names = ['Mild', 'Spicy']
        ticks = [0,1]
        # test_pred_labels = np.argmax(MNM.predict(dict(X_test)),
axis=1)
        # test_pred_probs = MNM.predict(dict(X_test))
        # logits = test_pred_probs.logits
        # test_pred_labels = np.argmax(logits, axis=1)
        conf_mat = confusion_matrix(y_test, y_pred)
        plt.figure(figsize=(8,8))
        plt.imshow(conf_mat, cmap=plt.cm.Blues)
        plt.title(f'Confusion Matrix: {model} {hourLimit} Hours')
        plt.colorbar()
        plt.xlabel('Predicted Labels')
        plt.ylabel('True Labels')
        plt.xticks(ticks, target_names)
        plt.yticks(ticks, target_names)
        plt.savefig(os.path.join(resultPath,f'Confusion
Matrix_{model}_{hourLimit}_Hours.png'))
        plt.show()
        return BNB
    if model == 'multinomial':
        v = TfidfVectorizer()
        # v = CountVectorizer()
```

```
        X_train = v.fit_transform(X_train) # 90% slit
        X_test = v.transform(X_test)
        mnm = MultinomialNB(alpha = 1.4295)
        scores = cross_val_score(mnm,X_train,y_train)
        print(f"Scores from Cross-Validation {scores}. Average scores
is {np.mean(scores)}")
        MNM = MultinomialNB(alpha = 1.4295)
        MNM.fit(X_train,y_train)
        y_pred = MNM.predict(X_test)
        y_pred_train = MNM.predict(X_train)
        print("\n")
        print('Model accuracy score: {0:0.4f}'.
format(accuracy_score(y_test, y_pred)))
        print('Training-set accuracy score: {0:0.4f}'.
format(accuracy_score(y_train, y_pred_train)))
        print("\n")
        print('Training set score: {:.4f}'.format(MNM.score(X_train,
y_train)))
        print('Test set score: {:.4f}'.format(MNM.score(X_test,
y_test)))
        print('Balance Accuracy:
{:.4f}'.format(balanced_accuracy_score(y_test, y_pred)))
        resultPath = os.path.join(nbResults+'-multinomial-
tfidfVectorizer')
        if not os.path.exists(resultPath):
            os.makedirs(resultPath)
        report = classification_report(y_test, y_pred)
        with open(os.path.join(resultPath,f'classificationReport-
{model}-{hourLimit}-tfidfVectorizer.txt'),'w') as f:
            for line in report:
                f.write(f'{line}')
        tn, fp, fn, tp = confusion_matrix(y_test, y_pred).ravel()
        results = {
```

```
    "False Positive Rate: Probability that a false alarm will be
raised: that a positive result will be given when the true value is
negative":fp/(fp+tn),
    "False Negative Rate: Probability that a true positive will be
missed by the test":fn/(tp+fn),
    "True Positive Rate | Recall | Sensitivity: Probability that an
actual positive will test positive": recall_score(y_test,y_pred),
    "True Negative Rate | Specificity: Probability that an actual
negative will test negative": tn/(tn+fp),
    "Negative Predictive Value: Likelihood that a mild outage is truly
a mild outage": tn/(tn+fn),
    "Positive Predictive Value: Likelihood that a spicy outage is truly
a spicy outage": precision_score(y_test,y_pred),
    "F1 Score: Harmonic mean between precision and recall":
f1_score(y_test,y_pred),
    "Cohen Kappa Score: How much better is your model over the random
classifier that predicts based on class frequencies":
cohen_kappa_score(y_test,y_pred),
    "Matthews Correlation Coefficient: Correlation between predicted
classes and ground truth":matthews_corrcoef(y_test,y_pred),
    "Log Loss: The difference between ground truth and predicted score
for every observation and average those errors over all observations":
log_loss(y_test,y_pred),
    "Balanced Accuracy":balanced_accuracy_score(y_test, y_pred),
}
        with open(os.path.join(resultPath,f'metrics-{model}-
{hourLimit}-tfidfVectorizer.txt'),'w') as sum:
            sum.write('{:-^150}'.format('SUMMARY'))
            sum.write(f'Confusion Matrix: \n
{confusion_matrix(y_test,y_pred)}')
            sum.write('\n')
            for test,score in results.items():
                string_format = '{}: \n{}'.format(test,score)
                sum.write(string_format)
                sum.write('\n')
```

```
                sum.write(f'Cross-validation scores {scores}. Average
Score is {np.mean(scores)}')
                sum.write('\n')
                sum.write('*'*10)
                sum.write('\n')
            sum.write('\n')
            sum.write('{:-^145}'.format('END SUMMARY'))
        reportd = classification_report(y_test,
y_pred,output_dict=True)
        print(report)
        report = pd.DataFrame(reportd).transpose()
        report.to_csv(os.path.join(resultPath,f'{model}-
{hourLimit}.csv'))

        target_names = ['Mild', 'Spicy']
        ticks = [0,1]
        # test_pred_labels = np.argmax(MNM.predict(dict(X_test)),
axis=1)
        # test_pred_probs = MNM.predict(dict(X_test))
        # logits = test_pred_probs.logits
        # test_pred_labels = np.argmax(logits, axis=1)
        conf_mat = confusion_matrix(y_test, y_pred)
        plt.figure(figsize=(8,8))
        plt.imshow(conf_mat, cmap=plt.cm.Blues)
        plt.title(f'Confusion Matrix: {model} {hourLimit} Hours')
        plt.colorbar()
        plt.xlabel('Predicted Labels')
        plt.ylabel('True Labels')
        plt.xticks(ticks, target_names)
        plt.yticks(ticks, target_names)
        plt.savefig(os.path.join(resultPath,f'Confusion
Matrix_{model}_{hourLimit}_Hours.png'))
        plt.show()
        return MNM
    if model == 'complement':
```

```
            alpha = alpha
            vect = vect
            if vect == 'bow':
                v = CountVectorizer()
            if vect == 'tfidf':
                v = TfidfVectorizer()

            X_train = v.fit_transform(X_train) # 90% split
            X_test = v.transform(X_test)
            cnb = ComplementNB(alpha = alpha)
            scores = cross_val_score(cnb,X_train,y_train)
            print(f"Scores from Cross-Validation {scores}. Average
scores is {np.mean(scores)}")
            CNB = ComplementNB(alpha = alpha)
            CNB.fit(X_train,y_train)
            y_pred = CNB.predict(X_test)
            y_pred_train = CNB.predict(X_train)
            print("\n")
            print('Model accuracy score: {0:0.4f}'.
format(accuracy_score(y_test, y_pred)))
            print('Training-set accuracy score: {0:0.4f}'.
format(accuracy_score(y_train, y_pred_train)))
            print("\n")
            print('Training set score:
{:.4f}'.format(CNB.score(X_train, y_train)))
            print('Test set score: {:.4f}'.format(CNB.score(X_test,
y_test)))
            print('Balance Accuracy:
{:.4f}'.format(balanced_accuracy_score(y_test, y_pred)))
            resultPath = os.path.join(nbResults+f'-complement-{alpha}-
{hourLimit}-{vect}')
            if not os.path.exists(resultPath):
                os.makedirs(resultPath)
            report = classification_report(y_test, y_pred)
```

```
            with open(os.path.join(resultPath,f'classificationReport-
{model}-{alpha}-{hourLimit}-{vect}.txt'),'w') as f:
                for line in report:
                    f.write(f'{line}')
            tn, fp, fn, tp = confusion_matrix(y_test, y_pred).ravel()
            results = {
        "False Positive Rate: Probability that a false alarm will be
raised: that a positive result will be given when the true value is
negative":fp/(fp+tn),
        "False Negative Rate: Probability that a true positive will be
missed by the test":fn/(tp+fn),
        "True Positive Rate | Recall | Sensitivity: Probability that an
actual positive will test positive": recall_score(y_test,y_pred),
        "True Negative Rate | Specificity: Probability that an actual
negative will test negative": tn/(tn+fp),
        "Negative Predictive Value: Likelihood that a mild outage is
truly a mild outage": tn/(tn+fn),
        "Positive Predictive Value: Likelihood that a spicy outage is
truly a spicy outage": precision_score(y_test,y_pred),
        "F1 Score: Harmonic mean between precision and recall":
f1_score(y_test,y_pred),
        "Cohen Kappa Score: How much better is your model over the
random classifier that predicts based on class frequencies":
cohen_kappa_score(y_test,y_pred),
        "Matthews Correlation Coefficient: Correlation between
predicted classes and ground truth":matthews_corrcoef(y_test,y_pred),
        "Log Loss: The difference between ground truth and predicted
score for every observation and average those errors over all
observations": log_loss(y_test,y_pred),
        "Balanced Accuracy":balanced_accuracy_score(y_test, y_pred),
    }
            with open(os.path.join(resultPath,f'SummaryReport-{model}-
{alpha}-{hourLimit}-{vect}.txt'),'w') as sum:
                sum.write('{:-^150}\n'.format('SUMMARY'))
```

```
                sum.write(f'Confusion Matrix: \n
{confusion_matrix(y_test,y_pred)}')
                sum.write('\n')
                for test,score in results.items():
                    string_format = '{}: \n{}'.format(test,score)
                    sum.write(string_format)
                    sum.write('\n')
                    sum.write(f'Cross-validation scores {scores}.
Average Score is {np.mean(scores)}')
                    sum.write('\n')
                    sum.write('*'*10)
                    sum.write('\n')
                sum.write('\n')
                sum.write('{:-^145}'.format('END SUMMARY'))
            reportd = classification_report(y_test,
y_pred,output_dict=True)
            print(report)
            report = pd.DataFrame(reportd).transpose()
            report.to_csv(os.path.join(resultPath,f'{model}-
{hourLimit}.csv'))

            target_names = ['Mild', 'Spicy']
            ticks = [0,1]
            # test_pred_labels = np.argmax(MNM.predict(dict(X_test)),
axis=1)
            # test_pred_probs = MNM.predict(dict(X_test))
            # logits = test_pred_probs.logits
            # test_pred_labels = np.argmax(logits, axis=1)
            conf_mat = confusion_matrix(y_test, y_pred)
            plt.figure(figsize=(8,8))
            plt.imshow(conf_mat, cmap=plt.cm.Blues)
            plt.title(f'Confusion Matrix: {model} {hourLimit} Hours')
            plt.colorbar()
            plt.xlabel('Predicted Labels')
            plt.ylabel('True Labels')
```

```
            plt.xticks(ticks, target_names)
            plt.yticks(ticks, target_names)
            plt.savefig(os.path.join(resultPath,f'Confusion
Matrix_{model}_{alpha}_{hourLimit}_Hours.png'))
            plt.show()
            return CNB


# %%
hourLimits = ['744','500','400','350','300','250']
alphaValues = [1.0,1.2,1.4,1.6,1.8,2.0]
for num in hourLimits:
    dfPath =
os.path.join(root,f'preLabelsByOutageHours_{num}_cleaned.csv')
    df = pd.read_csv(dfPath)
    train_texts, test_texts, train_labels, test_labels =
trainTestSplit(df,nb=True)
    for _alpha in alphaValues:
        buildModel(train_texts,train_labels,test_texts,test_labels,mode
l='complement',alpha=_alpha,vect='bow',hourLimit=num)
        buildModel(train_texts,train_labels,test_texts,test_labels,mode
l='complement',alpha=_alpha,vect='tfidf',hourLimit=num)
        #
buildModel(train_texts,train_labels,test_texts,test_labels,model='berno
ulli',hourLimit=num)
```

postProcessing.py

```
"""
This module was originally a google collab workbook.
It was used for post processing both naive bayes and BERT results
"""


import matplotlib.pyplot as plt
```

```
import os
from definitions import DATA_PATH, PLOT_PATH
from collections import defaultdict


# %%
def getModelName(modelPath):
    elements = modelPath.split("\\")[-1]
    elements = elements.replace('-','_').split('_')

    if 'naiveBayes' == elements[0]:

        model = elements[1] + 'NB'
        tokenizer = elements[-1]
        alpha = elements[-3]
        data = 'cleaned'+ elements[-2]
        name = f'{model}-NA-NA-{data}-{tokenizer}-{alpha}'
        return name



    if 'batch' in elements[0]:
        batch = elements[0][-2:]
        maxLength = elements[1][-3:]
        model = elements[2]
        tokenizer = elements[3]
        data = elements[6].split('Datasets')[0]
    else:
        batch = 32
        maxLength = 128
        model = elements[0]
        tokenizer = 'bert'
        if model == 'bert':
            data = elements[-1].split('Datasets')[0]
        else:
            data = elements[-2].split('Datasets')[0]
```

```
    if model == 'distillBert':
        model = 'distil'
        tokenizer = 'disTok'
    if model == 'morpyFinalTrain':
        model = 'morpy'
    if elements[0] == 'distilbert':
        batch = elements[4]
        maxLength = elements[5]
        data = elements[6].split('Datasets')[0]
        tokenizer = 'disTok'
        model = 'distilUncased'


    name = f'{model}-{batch}-{maxLength}-{data}-{tokenizer}'
    return name

# %%
def f1(report):
    rpt_id = report[0]
    score = report[1]._f1Score
    return rpt_id,score
def precision(report):
    rpt_id = report[0]
    score = report[1]._positivePredictiveValue
    return rpt_id,score
def recall(report):
    rpt_id = report[0]
    score = report[1]._truePositiveRate
    return rpt_id,score
def specificity(report):
    rpt_id = report[0]
    score = report[1]._trueNegativeRate
    return rpt_id,score
def maxValue(dictionary):
    if not dictionary:
```

```
        return None
    maxValueKey = max(dictionary, key=dictionary.get)
    maxValue = max(dictionary.values())
    return maxValueKey,maxValue


# %%
class SummariesOutput:
    def __init__(self, outputFilePath):
        self.outputFilePath = outputFilePath
        for attr in [
            "_morpData",
            "_tokenizer",
            "_preTrainedWeights",
            "_confusionMatrix",
            "_falsePositiveRate",
            "_falseNegativeRate",
            "_truePositiveRate",
            "_trueNegativeRate",
            "_negativePredictiveValue",
            "_positivePredictiveValue",
            "_f1Score",
            "_cohenKappaScore",
            "_matthewsCorrelationCoeff",
            "_logLoss",
            "_averageCrossValidationScore",
            "_balancedAccuracy",
        ]:
            setattr(self, attr, None)
        self._populateEverything()
        self.id = f'{self._morpData}-{self._tokenizer}-
{self._preTrainedWeights}'
    def _populateEverything(self):
        with open(self.outputFilePath, "r") as f:
            l = f.readlines()
```

```
            for i in range(0, len(l)):

                line = l[i]
                if "Morp Data" in line:
                    self._morpData = line.split(':')[1].split()[0]
                else:
                    self._morpData = self.outputFilePath.split('-')[-2]

                if "Tokenizer" in line:
                    self._tokenizer = line.split(':')[1].split()[0]
                else:
                    self._tokenizer = self.outputFilePath.split('-')[-
1]
                if "Pre-Trained Weights" in line:
                    self._preTrainedWeights =
line.split(':')[1].split()[0]
                if "Confusion Matrix" in line:
                    tn = l[i+1].split('[')[2].split(']')[0].split('
')[0]
                    fp = l[i+1].split('[')[2].split(']')[0].split('
')[-1]
                    fn = l[i+2].split('[')[-
1].split(']')[0].split('  ')[-2]
                    tp = l[i+2].split('[')[1].split(']')[0].split('
')[-1]
                    self._confusionMatrix = [tn,fp,fn,tp]
                if 'Cross-validation scores' in line:
                    self._averageCrossValidationScore = line.split('].
')[-1]
                else:
                    self._averageCrossValidationScore = None
                if "False Positive Rate" in line:
                    self._falsePositiveRate = l[i+1].split()[0]
                if "False Negative Rate" in line:
                    self._falseNegativeRate = l[i+1].split()[0]
```

```
                if "True Positive Rate" in line:
                    self._truePositiveRate = l[i+1].split()[0]
                if "True Negative Rate" in line:
                    self._trueNegativeRate = l[i+1].split()[0]
                if "Negative Predictive Value" in line:
                    self._negativePredictiveValue = l[i+1].split()[0]
                if "Positive Predictive Value" in line:
                    self._positivePredictiveValue = l[i+1].split()[0]
                if "F1 Score" in line:
                    self._f1Score = l[i+1].split()[0]
                if "Cohen Kappa Score" in line:
                    self._cohenKappaScore = l[i+1].split()[0]
                if "Matthews Correlation Coefficient" in line:
                    self._matthewsCorrelationCoeff = l[i+1].split()[0]
                if "Log Loss" in line:
                    self._logLoss = l[i+1].split()[0]
                if 'Balanced Accuracy' in line:
                    self._balancedAccuracy = l[i+1].split()[0]

# %%
results = os.path.join('resources','results')
modelPaths = []
summariesByModel = defaultdict()
textFiles = []

for _, dirs, files in os.walk(results):
    for dir in dirs:
        root = os.path.join('resources','results',dir)
        modelPaths.append(root)


for modelPath in modelPaths:
    summaries = []
    modelName = getModelName(modelPath)
    for file in os.listdir(modelPath):
```

```
        if 'metrics' in file:
            textFiles.append(os.path.join(modelPath,file))
            summaries.append(os.path.join(modelPath,file))
        if ('metrics' not in file) and ('classificationReport' not in
file) and file.endswith('.txt'):
            textFiles.append(os.path.join(modelPath,file))
            summaries.append(os.path.join(modelPath,file))
    summariesByModel[modelName] = summaries



# %%
f1Scores = defaultdict()
precisionScores = defaultdict()
recallScores = defaultdict()
specificityScores = defaultdict()
confusionMatrix = defaultdict()

reportSummary = defaultdict()
reports = []

for report in textFiles:
    rpt = SummariesOutput(report)
    rpt_id = rpt.outputFilePath
    reports.append((rpt_id, rpt))

for report in reports:
    rpt_id = report[0]
    score = report[1]

    f1Scores[rpt_id] = score._f1Score
    precisionScores[rpt_id] = score._positivePredictiveValue
    recallScores[rpt_id] = score._truePositiveRate
    specificityScores[rpt_id] = score._trueNegativeRate
    confusionMatrix[rpt_id] = score._confusionMatrix
```

```
# %%



# %%
topScores = {
    'f1':maxValue(f1Scores),
    'precision': maxValue(precisionScores),
    'recall':maxValue(recallScores),
    'specificity':maxValue(specificityScores)
}

# %%
hourLimits = ['744','500','400','350','300','250']
limits = defaultdict(list)
for model,summaries in summariesByModel.items():
    for summary in summaries:
        for limit in hourLimits:
            if limit in summary:
                limits[limit].append((model,summary))

# %%

def getPlotHours(limitsDictionary,scoreType=False):
    hourDict = defaultdict(list)
    for limit, summaries in limitsDictionary.items():
        modelScores = []
        for model,summary in summaries:
            report = SummariesOutput(summary)
            if scoreType == 'f1':
                score = report._f1Score
            if scoreType == 'precision':
                score = report._positivePredictiveValue
```

```
            if scoreType == 'recall':
                score = report._truePositiveRate
            if scoreType == 'specificity':
                score = report._trueNegativeRate
            if scoreType == 'cohen':
                score = report._cohenKappaScore

            score = float(score)
            modelScore = (model,score)
            modelScores.append(modelScore)
        hourDict[limit] = modelScores
    return hourDict

def getPlotHoursByModelType(limitsDictionary,scoreType=False):
    NBHourDict = defaultdict(list)
    BERTHourDict = defaultdict(list)
    for limit, summaries in limitsDictionary.items():
        NBmodelScores = []
        bertmodelScores = []

        for model,summary in summaries:
            report = SummariesOutput(summary)
            if scoreType == 'f1':
                score = report._f1Score
            if scoreType == 'precision':
                score = report._positivePredictiveValue
            if scoreType == 'recall':
                score = report._truePositiveRate
            if scoreType == 'specificity':
                score = report._trueNegativeRate
            if scoreType == 'cohen':
                score = report._cohenKappaScore

            score = float(score)
            modelScore = (model,score)
```

```
            if 'complementNB-NA-NA' in model:
                NBmodelScores.append(modelScore)
            if 'complementNB-NA-NA' not in model:
                bertmodelScores.append(modelScore)

        NBHourDict[limit] = NBmodelScores
        BERTHourDict[limit] = bertmodelScores
    return NBHourDict,BERTHourDict




# %%
def plotHourLimit(hoursDict,scoreType,limit):
    colors = []
    xValues, yValues = zip(*hoursDict[f'{limit}'])
    maxMetric = max(yValues)
    for value in yValues:
        if value == maxMetric:
            colors.append('r')
        else:
            colors.append('b')


    plt.bar(xValues,yValues,color=colors)
    plt.xticks(rotation=88)
    plt.xlabel('Model')
    plt.ylabel(f'{scoreType}')
    plt.title(f'{scoreType} - Outage Severity: {limit}')
    plotPath = os.path.join(PLOT_PATH,f'{limit}',f'{scoreType}')
    if not os.path.exists(plotPath):
        os.makedirs(plotPath)
    plt.savefig(os.path.join(plotPath,f'{scoreType}-
{limit}.jpeg'),dpi=300, bbox_inches = "tight")
    plt.show()
```

```
# %%
def
plotHourLimitByModelType(dictByModelType,scoreType,limit,modelType):
    colors = []
    xValues, yValues = zip(*dictByModelType[f'{limit}'])
    maxMetric = max(yValues)
    for value in yValues:
        if value == maxMetric:
            colors.append('r')
        else:
            colors.append('b')


    plt.bar(xValues,yValues,color=colors)
    plt.xticks(rotation=88)
    plt.xlabel('Model')
    plt.ylabel(f'{scoreType}')
    plt.title(f'{scoreType} - Outage Severity: {limit}')
    plotPath = os.path.join(PLOT_PATH,f'{limit}',f'{scoreType}')
    if not os.path.exists(plotPath):
        os.makedirs(plotPath)
    plt.savefig(os.path.join(plotPath,f'{modelType}-{scoreType}-
{limit}.jpeg'),dpi=300, bbox_inches = "tight")
    plt.show()

def createCM(confusionMatrixValues):
    print(confusionMatrixValues)
    if len(confusionMatrixValues) != 4:
        raise ValueError("Values must contain 4 elements")
    tn, fp, fn, tp = confusionMatrixValues
    table = f"""\
| {'':<25}| {'Predicted Negative (Mild)':<23} | {'Predicted Positive
(Spicy)':<23} |
| {'-'*25}|{'-'* 27}|{'-'* 28}|
| {'Actual Negative (Mild)':^24} | {tn:^17}         | {fp:^20}       |
```

```
| {'Actual Negative (Spicy':^24} | {fn:^17}         | {tp:^20}       |
"""
    return table

# %%
outageHourDict = defaultdict(list)
for report in textFiles:
    rpt = SummariesOutput(report)
    name = getModelName(report)

    if 'Report-bert-large-uncased' in name:
        hour = name.split('_')[-1].split('BertTokenizer.txt')[0]
    if 'morpyFinalTrain' in name:
        hour = name.split('_')[-1]
    if 'preLabelsByOutageHours-bert' in name:
        fileName = rpt.outputFilePath.split("\\")[-1]
        hour = fileName.replace(' ','')
        hour = hour.split('_')[-1][:3]
    else:
        hour = name.split('-')[-2]
    outageHourDict[hour].append(rpt)

f1ScoreByHour = defaultdict(list)
precisionByHour = defaultdict(list)
recallByHour = defaultdict(list)
CMByHour = defaultdict(list)

f1ScoreByHourFinal = defaultdict()
precisionByHourFinal = defaultdict()
recallByHourFinal = defaultdict()

for hour in outageHourDict.keys():
    f1Max = max([report._f1Score for report in outageHourDict[hour]])
    precisionMax = max([report._positivePredictiveValue for report in
outageHourDict[hour]])
```

```
    recallMax = max([report._truePositiveRate for report in
outageHourDict[hour]])
    f1ScoreByHour[hour] = f1Max
    precisionByHour[hour] = precisionMax
    recallByHour[hour] = recallMax
    #CMByHour[hour] = report._confusionMatrix

for hour in outageHourDict.keys():
    reports  = outageHourDict[hour]
    f1Max = f1ScoreByHour[hour]
    precisionMax = precisionByHour[hour]
    recallMax = recallByHour[hour]
    for report in reports:
        if report._f1Score == f1Max:
            rptScore =
(report.outputFilePath,report._f1Score,createCM(report._confusionMatrix
))
            f1ScoreByHourFinal[hour] = rptScore
        if report._positivePredictiveValue == precisionMax:
            rptScore =
(report.outputFilePath,report._positivePredictiveValue,createCM(report.
_confusionMatrix))
            precisionByHourFinal[hour] = rptScore
        if report._truePositiveRate == recallMax:
            rptScore =
(report.outputFilePath,report._truePositiveRate,createCM(report._confus
ionMatrix))
            recallByHourFinal[hour] = rptScore

# %%
with open(os.path.join(PLOT_PATH,'recallByHour.txt'),'w') as f:
    for model, score, cm in recallByHourFinal.values():
        f.write(f'|{model}|')
        f.write('\n')
        f.write(f'|{score}|')
```

```
        f.write('\n')
        f.write(cm)
        f.write('\n')

# %%
metrics = ['f1','precision','recall','specificity']
f1ModelPath = topScores['f1'][0]
precisionModelPath = topScores['precision'][0]
recallModelPath = topScores['recall'][0]
specificityModelPath = topScores['specificity'][0]

with open(os.path.join(PLOT_PATH,'Max_Scores.txt'),'w') as f:

    for metric in metrics:
        modelPath = topScores[metric][0]
        CM_values= confusionMatrix[modelPath]
        score = topScores[metric][1]
        CM = createCM(CM_values)
        header = f'Max {metric} Score: {score} - Model {modelPath}'

        f.write(f'|{modelPath}|')
        f.write('\n')
        f.write('\n')
        f.write(f'|{header}|')
        f.write('\n')
        f.write(CM)
        f.write('\n')
        f.write('\n')

# %%
topScores

# %%
```

```
# %%
recallByHourFinal

# %%
metrics = ['f1','precision','recall','specificity']

for hourLimit, summaries in limits.items():
    with open(os.path.join(PLOT_PATH,hourLimit,f'CM-
{hourLimit}.txt'),'w') as f:
        for summary in summaries:
            header = '\n'
            name = summary[0]
            report = SummariesOutput(summary[1])

            for metric in metrics:
                id = topScores[metric][0]
                score = topScores[metric][1]
                if id == report.outputFilePath:
                    header = f'Max {metric}: {score} - Confusion Matrix
for {name}\n'
            print(hourLimit)
            print(name)
            table = createCM(report._confusionMatrix)

            f.write(name)
            f.write('\n')
            f.write(header)
            f.write(table)
            f.write('\n')



# %%
metrics = ['precision','f1','recall','specificity','cohen']
for limit in hourLimits:
```

```
    for metric in metrics:
        hourDict = getPlotHours(limits,scoreType=metric)
        NBHourDict,BERTHourDict =
getPlotHoursByModelType(limits,scoreType=metric)
        print(NBHourDict)
        plotHourLimit(hourDict,scoreType=metric,limit=limit)
        plotHourLimitByModelType(NBHourDict,scoreType=metric,limit=limi
t,modelType='CNB')
        plotHourLimitByModelType(BERTHourDict,scoreType=metric,limit=li
mit,modelType='BERT')
```

B. masterExpansions.txt

HRS:HOURS

RAD:RADIATION

CCW:COMPONENT COOLING WATER

SIRW:Safety Injection Refueling Water

MSDT:MOISTURE SEPARATOR DRAIN TANK

PLCEA:PART LENGTH CONTROL ELEMENT ASSEMBLY

CEA:CONTROL ELEMENT ASSEMBLY

SAT:SYSTEM AUXILIARY TRANSFORMER

RHR:RESIDUAL HEAT REMOVAL

OOS:out of sequence

MSTV:main steam trip valve

HCU:hydraulic control unit

PMG:PERMANENT MAGNET GENERATOR

FCV:flow control valve

RCP:reactor coolant pump

SJAE:STEAM JET AIR EJECTOR

CWIP:CIRCULATING WATER INTAKE PUMP

TSV:turbine stop valve

RCPS:reactor coolant pumps

ETSV:ELECTRICAL TRIP SOLENOID VALVE

RVDT:RECIRCULATION FLOW CONTROL VALVE POSITIONER

SIRW:safety injection and refueling water

EHC:electrohydraulic control

CST:condensate storage tank

OTDT:Over temperature Delta Temperature

MSIV:Main steam isolation valve

MFPT:MAIN FEEDWATER PUMP TURBINE

HPCI:High pressure coolant injection

CEA:control element assembly calculator

SSPS:SOLID STATE PROTECTION SYSTEM

AE:Architect engineer

PORV:power operated relief valves

AVR:AUTOMATIC VOLTAGE REGULATOR

MSR:material status report

RPIS:ROD POSITION INDICATORS

AR:action request

MOD:motor operated disconnect

TS:technical specification

SRVS:safety relief valves

SRV:safety relief valve

RHR:Reactor heat removal

APRM:Average Power Range Monitor

DEH:digital electrohydraulic

CVCS:CHEMICAL AND VOLUME CONTROL SYSTEM

RBCCW:reactor building closed cooling water

RRP:reactor recirculation pump

EDG:Emergency Diesel generators

MPR:MECHANICAL PRESSURE REGULATOR

FCU:FAN COOLER UNIT

CFCU:CONTAINMENT FAN COIL UNIT

SST:STATION SERVICE TRANSFORMER

SWC:STATOR WATER COOLING

TV:THROTTLE VALVE

PCB:PRINTED CIRCUIT BOARD

F:FAHRENHEIT

TSE:TURBINE STRESS EVALUATOR

CBA:CONTROL BUILDING AIR

MPT:MAIN POWER TRANSFORMER

CPC:CORE PROTECTION CALCULATOR

CCP:CAPACITANCE COUPLED VOLTAGE

AE:ARCHITECT AND ENGINEERING
LCO:LIMITING CONDITION FOR OPERATION
RCC:REACTOR CLOSED COOLING
TT:TURBINE TRIP
MG:REACTOR RECIRCULATION GENERATOR
ERV:ELECTROMATIC RELIEF VALVE
RPS:REACTOR PROTECTION SYSTEM
TCV:TURBINE CONTROL VALVES
MFP:MAIN FEEDWATER PUMP
LS:LEVEL SWITCH
CIV:COMBINED INTERCEPT VALVE
S/G:STEAM GENERATOR
RX:REACTOR
ICS:INTEGRATED CONTROL SYSTEM
RCS:REACTOR COOLANT SYSTEM
FW:FEEDWATER
RR:REACTOR RECIRCULATING PUMP
MO:MOTOR OPERATED
PZR:PRESSURIZER
LP:LOW PRESSURE
LHSI:LOW HEAD SAFETY INJECTION
MVAR:MEGA VOLT AMP REACTIVE
RPV:REACTOR PRESSURE VESSEL
ATWOS:ANTICIPATED TRANSIENT WITHOUT SCRAM
MWTH:MEGAWATT THERMAL
MWT:MEGAWATT THERMAL
CRD:CONTROL ROD DRIVE
CW:COOLING WATER PUMP
CWS:COOLING WATER SEAL
DP:DIFFERENTIAL PRESSURE
HC:HYDRAULIC CONTROL SYSTEM
EHCS:ELECTROHYDRAULIC CONTROL SYSTEM
MSU:MAIN STEP UP TRANSFORMER
BPV:BYPASS VALVE
RFP:REACTOR FEEDWATER PUMP
RRCS:REDUNDANT REACTIVITY CONTROL SYSTEM
FWRV:FEEDWATER REGULATING VALVE

TDRFP:TURBINE DRIVEN REACTOR FEEDWATER PUMP

RFPT:REACTOR FEEDWATER PUMP TURBINE

CT:CURRENT TRANSFORMER

AMSAC:ANTICIPATED TRANSIENTS WITHOUT SCRAM MITIGATING SYSTEM ACTUATION CIRCUITRY

MS:MAIN STEAM

AFW:AUXILLARY FEEDWATER

GEN:GENERATOR

MW:MEGAWATT

U1:UNIT

BYV: BYPASS VALVE

GGN:GRAND GULF NUCLEAR

BFD:BOILER FEEDWATER DISCHARGE VALVE

*https://www.nrc.gov/docs/ML1613/ML16133A035.pdf

MV:MAIN FEEDWATER PUMP DISCHARGE VALVE

*https://www.nrc.gov/docs/ML0037/ML003775084.pdf

RTP:RATED THERMAL POWER

OPRM:OSCILLATION POWER RANGE MONITOR

EPRM:ELECTROMATIC RELIEF VALVE

UAT:UNIT AUXILLARY TRANSFORMER

EOP:EMERGENCY OPERATING PROCEDURE

OCB:OIL CIRCUIT BREAKER

RFW:REACTOR FEEDWATER

FWH:FEEDWATER HEATER

FWP:FEEDWATER PUMP

FWCV:FEEDWATER CONTROL VALVE

FWIV:FEEDWATER ISOLATION VALVE

ATWS:ANTICIPATED TRANSIENT WITHOUT SCRAM

FWR:FEEDWATER INJECTION

EH:ELECTROHYDRAULIC

AOV:AIR OPERATED VALVE

MBFP:MAIN BOILER FEEDWATER PUMP

DNBR:DEPARTURE FROM NUCLEATE BOILING RATIO

ASD:ADJUSTABLE SPEED DRIVE

*https://www.nrc.gov/docs/ML2022/ML20223A258.pdf

EPR:ELECTRONIC PRESSURE REGULATOR

*https://www.nrc.gov/docs/ML1705/ML17056B865.pdf

MPR:MECHANICAL PRESSURE REGULATOR

*https://www.nrc.gov/docs/ML1705/ML17056B865.pdf

VAC:VACUUM

SRMS:SOURCE RANGE MONITORS

ACB:AIR CIRCUIT BREAKER

RMS:RADIATION MONITORING SYSTEM

*https://www.nrc.gov/docs/ML1809/ML18095A744.pdf

DC:DIRECT CURRENT

LOCA:LOSS OF COOLANT ACCIDENT

CRDM:CONTROL ROD DRIVE MECHANISM

XFMR:AUXILIARY TRANSFORMER

CTP:CORE THERMAL POWER

EPU:EXTENDED POWER UPRATE

CEDM:CONTROL ELEMENT DRIVE MECHANISM

*https://www.nrc.gov/docs/ML2128/ML21285A326.pdf

HDDT:HEATER DRAINS DEAERATOR TANK

*https://www.nrc.gov/docs/ML1025/ML102560189.pdf

AOT:ALLOWED OUTAGE TIME

PORC:PLANT OPERATIONS REVIEW COMMITTEE

GV:GOVERNOR VALVE

RWCU:REACTOR WATER CLEANUP SYSTEM

*https://www.nrc.gov/docs/ML1125/ML11258A313.pdf

TACH:TACHOMETER GENERATOR

PWR:PRESSURE WATER REACTOR

MSL:MAIN STEAM LINES

HPSI:HIGH PRESSURE SAFETY INJECTION

LCS:LEAKAGE CONTROL SYSTEM *https://www.nrc.gov/reading-rm/doc-collections/gen-comm/gen-letters/1986/gl86017.html

FME:FOREIGN MATERIAL EXCLUSION

*https://www.nrc.gov/docs/ML0519/ML051920220.pdf

VDC:VOLTAGE DIRECT CURRENT

RCFC:REACTOR CONTAINMENT FAN COOLER

*https://www.nrc.gov/docs/ML1821/ML18212A092.pdf

MFIV:MAIN FEEDWATER ISOLATION VALVE

*https://www.nrc.gov/docs/ML0515/ML051540080.pdf

MFRV:MAIN FEEDWATER REGULATING VALVE

*https://www.nrc.gov/docs/ML0515/ML051540080.pdf

MFRVBV:MAIN FEEDWATER REGULATING BYPASS VALVE

*https://www.nrc.gov/docs/ML0515/ML051540080.pdf

SDV:SCRAM DISCHARGE VALVE

*https://www.nrc.gov/docs/ML0417/ML041760484.pdf

MSLI:MAIN STEAM LINE ISOLATION

*https://www.nrc.gov/docs/ML0311/ML031190607.pdf

LPCI:LOW PRESSURE COOLANT INJECTION

*https://www.nrc.gov/docs/ML1414/ML14140A178.pdf

RCDT:REACTOR COOLANT DRAIN TANK

*https://www.nrc.gov/docs/ML1122/ML11223A213.pdf

PRT:PRESSURIZER RELIEF TANK

*https://adamswebsearch2.nrc.gov/webSearch2/view?AccessionNumber=ML15127A218

FIV:FLOW INDUCED VIBRATION

HEPA:HIGH EFFICIENCY PARTICULATE AIR FILTER

RRC:REACTOR RECIRCULATION SYSTEM

*https://www.nrc.gov/docs/ML0506/ML050610021.pdf

GPM:GALLONS PER MINUTE

DFG:DIODE FUNCTION GENERATOR

SG:STEAM GENERATOR

PCIG:PRIMARY CONTAINMENT INSTRUMENT GAS

*https://www.nrc.gov/docs/ML0407/ML040790363.pdf

PCS:PASSIVE CONTAINMENT COOLING SYSTEM

*https://adamswebsearch2.nrc.gov/webSearch2/view?AccessionNumber=ML20350B435

DG:DIESEL GENERATOR

RN:NUCLEAR SERVICE WATER *https://www.nrc.gov/docs/ML1110/ML111020305.pdf

MTG:MAIN TURBINE GENERATOR

LVDT:LINEAR VOLTAGE DIFFERENTIAL TRANSFORMER

*https://www.nrc.gov/docs/ML2007/ML20078B942.pdf

ESW:ESSENTIAL SERVICE WATER

*https://www.nrc.gov/sr0933/Section%203.%20New%20Generic%20Issues/153r2.html

CV:CONTROL VALVE

EDDY:EDDY CURRENT

ECCS:EMERGENCY CORE COOLING SYSTEM

NRC:NUCLEAR REGULATORY COMMISSION

INLEAKAGE:INLEAKAGE

LOAD LETDOWN:LOAD LETDOWN

LOAD ACTUAL:LOAD ACTUAL

AC:ALTERNATING CURRENT
AF:AUXILIARY FEEDWATER *https://www.nrc.gov/docs/ML1917/ML19171A178.pdf
TG:TURBINE GENERATOR
AUXILIARY SPRAY VALVE:AUXILIARY SPRAY VALVE
AUXILIARY FEEDWATER SYSTEM:AUXILIARY FEEDWATER SYSTEM
AUXILIARY FEEDWATER STEAM SUPPLY VALVE:AUXILIARY FEEDWATER STEAM SUPPLY VALVE
GDC:GENERAL DESIGN CRITERIA
SIT:SAFETY INJECTION TANK *https://www.nrc.gov/docs/ML0037/ML003756995.pdf
PCIS:PRIMARY CONTAINMENT ISOLATION SIGNAL *https://www.nrc.gov/reading-rm/doc-collections/event-status/event/2022/20220518en.html
SW:SERVICE WATER
IRM:INTERMEDIATE RANGE MONITOR
PRC:PROGRAMMABLE LOGIC CONTROLLER
HDT:HEATER DRAIN TANK
HDT:HEATER DRAIN TANK *https://www.nrc.gov/docs/ML2003/ML20038A107.pdf
HP:HIGH PRESSURE
RWS:RAW WATER SYSTEM *https://www.nrc.gov/docs/ML0935/ML093560442.pdf
SI:SAFETY INJECTION
RTD:RESISTANCE TEMPERATURE DETECTOR
*https://www.nrc.gov/docs/ML2005/ML20052H284.pdf
SUMP:SUMP
MCB:MAIN CONTROL BOARD *https://www.nrc.gov/docs/ML1416/ML14167A296.pdf
CTMT:CONTAINMENT
SUT:STARTUP TRANSFORMER *https://www.nrc.gov/docs/ML0733/ML073320172.pdf
59GG:GENERATOR GROUND RELAY
CCP:CENTRIFUGAL CHARGING PUMP
*https://www.nrc.gov/docs/ML1122/ML11223A220.pdf
ORING:ORING
DEMINERALIZER:DEMINERLIZER
DRYWELL:DRYWELL
RECOMBINER:RECOMBINER
RAMPDOWN:RAMPDOWN
OVERVOLTAGE:OVERVOLTAGE
MIDCYCLE:MIDCYCLE
DOWNPOWER:DOWNPOWER
POSITIONER:POSITIONER

DEMINERALIZER:DEMINERALIZER
ISOPHASE:ISOPHASE
NONCONDENSIBLE:NONCONDENSIBLE
NONISOLABLE:NONISOLABLE
SETPOINT:SETPOINT
FOREBAY:FOREBAY
LER:LER
UNDERVOLTAGE:UNDERVOLTAGE
HOTWELL:HOTWELL
WATERBOX:WATERBOX
BASKWASHING:BASKWASHING
ANTIMOTORING:ANTIMOTORING
BACKPRESSURE:BACKPRESSURE
INTERCONDENSER:INTERCONDENSER
REBOILER:REBOILER
WEEPAGE:WEEPAGE
DOWNPOWERED:DOWNPOWERED
UNISOLATE:UNISOLATE
SWITCHYARD:SWITCHYARD
WALKDOWN:WALKDOWN
ANTIREVERSE:ANTIREVERSE
HANDSWITCH:HANDSWITCH
CLAMICIDE:CLAMICIDE
MONOBLOCK:MONOBLOCK
SUBLOOP:SUBLOOP
UNDERFREQUENCY:UNDERFREQUENCY
OVERFREQUENCY:OVERFREQUENCY
DEFUELED:DEFUELED
UNLANDED:UNLANDED
EXCERCISER:EXCERCISER

C. hyphensInVocab.txt

LTD:letdown::: Turbine trip/reactor scram from 100% at 14:32 on 7/30/12. The scram occurred shortly after the Turbine Stress Evaluator (TSE) was turned on. LOAD-LTD and LOAD-ACTUAL dropped, and the turbine tripped.
BYV:Bypass Valve::: UNIT WAS MANUALLY SHUTDOWN TO REPLACE A PRESSURIZER SPRAY BYPASS VALVE (1-BYV-68-555).

MO:motor operator::: REMOVED FROM SERVICE TO INVESTIGATE AND REPAIR LEAKAGE FROM MO-7071 AND/OR VPI-303 ON THE POST INCIDENT SYSTEM. AFTER THE UNIT WAS REMOVED FROM SERVICE, THE REACTOR WAS MANUALLY SCRAMMED TO FULLY INSERT THE REMAINING WITHDRAWN CONTROL ROD DRIVES.

EDG:Emergency Diesel generators::: TROUBLESHOOTING AND REPAIR EFFORTS FOR THE EDG-2 VOLTAGE REGULATION SYSTEM FAILED TO RESTORE EDG-2 TO OPERABLE CONDITION AND IN ACCORDANCE WITH TECHNICAL SPECIFICATION 3.8.1.F THE PLANT WAS SHUTDOWN (LER98-14).

MD:management directive::: POWER REDUCTION FOR ROD IMPROVEMENT AND LEAK REPAIR TO 2-MD-LV-2SRDCV-H-1.

LV:leaky valve::: POWER REDUCTION FOR ROD IMPROVEMENT AND LEAK REPAIR TO 2-MD-LV-2SRDCV-H-1.

MS:main steam::: UNIT WAS SHUTDOWN TO ENTER CONTAINMENT AND REPAIR LEAKING SG SECONDARY SIDE TUBE SHEET DRAIN VALVE LEAK, 1MS-0664.

NON:notice of nonconformance::: AN EQUIPMENT FAULT CAUSED THE LOSS OF ALL NON-1E 13.8 SWITHCHGEAR DUE TO INCORRECT TAP SETTINGS FOR THE INSTANEOUS OVERCURRENT RELAY FOR BREAKER 2NAB03. THIS CAUSED A TURBINE TRIP/REACTOR TRIP WHEN THE MAIN GENERATOR OUTPUT BREAKERS OPENED.

MS:main steam::: UNIT TAKEN OFFLINE TO REPAIR VALVE 1 MS-0063.

COND:Condensate Pump::: AUTOMATIC SCRAM DUE TO LOSS OF FEED. WHILE SHIFTING OIL FILTERS ON COND-P-2B, A MOTOR TRIP OF CONP-P-2B WAS RECEIVED, FOLLOWED BY A TRIP OF BOTH FEED TURBINES.

MO:motor operator::: MO-10

SS:stainless steel::: REACTOR SHUTDOWN TO REPAIR PIN HOLE LEAK AND SMALL CRACK UPSTREAM OF 1-SS-217 "C" STEAM GENERATOR SURFACE SAMPLE LINE MANUAL ISOLATION VALVE.

SIA:Safety injection::: RX WAS MANUALLY TRIPPED TO REPAIR VIBRATIONS ON SIA-UV-651.

UV:Reactor Coolant Valves::: RX WAS MANUALLY TRIPPED TO REPAIR VIBRATIONS ON SIA-UV-651.

TM:technical manual::: POWER REDUCTION FOR 2-OSP-TM-001.

FW:feedwater::: UNIT 1 WAS REMOVED FROM SERVICE TO REPAIR 1-FW-E-4B.

MD:management directive::: Generator removed from service due to failure of valve 2-MD-V14 resulting in internal flooding.

FW:feedwater::: REPAIRED LEAKING FEEDWATER VENT VALVE 2-FW-261B ON THE MAIN FEEDWATER LINE TO #2 STEAM GENERATOR.
FW:feedwater::: POWER REDUCTION DUE TO CRACKED WELD ON MAIN FEEDWATER PUMP 2B CASING VENT VALVE 2FW-0011.
RC:reactor cavity::: UNIT SHUTDOWN TO REPLACE RCP MOTOR (1-RC-P-1B) THAT HAD HIGH VIBRATIONS.
TR: TRANSFORMER LINKS::: REDUCED RX POWER TO 15%. REMOVED MAIN TURBINE AND GENERATOR FROM SERVICE TO FIX E-TR-M1 TRANSFORMER LINKS(i.e. THERMOGRAPHY IDENTIFIED HOT SPOTS IN THE LINK BOLTED CONNECTIONS).
RE:radiation equipment::: THE LOOP 4 MAIN STEAM ISOLATION VALVE DRIFTED CLOSED AND COULD NOT BE RE-OPENED. THE REACTOR OPERATOR MANUALLY TRIPPED THE REACTOR. THE VALVE WENT CLOSED DUE TO A BLOWN FUSE.
RHV:Residual Heat Removal System Valve::: THE PLANT SHUT DOWN TO REPAIR AN EHC FLUID LEAK ON RHV-5.
MS:main steam::: TURBINE SHUTDOWN TO REPAIR A LEAKING WELD AT MS-1607. (CR-IP2-2004-06527).
RRC:radiation recorder controller::: POWER REDUCTION, RRC-ASD-1A2 TRIPPED. PLANT ENTERED SINGLE LOOP OPERATION.
ASD:adjustable speed drive::: POWER REDUCTION, RRC-ASD-1A2 TRIPPED. PLANT ENTERED SINGLE LOOP OPERATION.
BVPS:Beaver Valley Power Station::: BVPS-2 was shutdown on 2/3/16 at 1626 hours for repair of high end turn vibrations on the Main Unit Generator. Upon completion of repairs, the Unit was synchronized to the electrical grid at 0506 hours on 2/12/16 and returned to 100% power.
UV:undervoltage::: POWER REDUCTION FOR HEAT TREAT, AND PERFORMED MAINTENANCE ON HP GOVERNOR VALVE 3UV-2200G.
AOV:air operated valve::: FEEDWATER REGULATOR VALVE ACTUATOR REPAIR AOV-4269.
FW:feedwater::: Manual Reactor Trip due to FW Transient (spurious closure of 2-FW-MOV-250C)
MOV:metal oxide varister::: Manual Reactor Trip due to FW Transient (spurious closure of 2-FW-MOV-250C)
QF:quality factor::: Q2F60-OCCURRED DURING TURBINE THRUST BEARING WEAR DETECTOR SURVEILLANCE.

UNIT:United Illuminating Co::: DUAL-UNIT SHUTDOWN TO REPLACE 87 DP RELAYS ON ALL EDGS.
FW:feedwater::: POWER REDUCTION TO PERFORM MAINTENANCE ON 2-FW-P-1A AND 2-FW-P-1B.
CF:column feed::: POWER REDUCTION TO EVALUATE PACKING LEAK ON FEEDWATER VALVE 2CF-28.
LER:licensee event report::: LER-1999-004, MANUAL SCRAM
MS:main steam::: THE UNIT HAD TO BE SHUT DOWN TO CORRECT A PROBLEM WITH 1-MS-BPV-3.
WMO:World Meteorological Organization::: UNIT 2 WAS SHUTDOWN AFTER THE #23 CW PUMP DISCHARGE VALVE 2-WMO-23 FAILED CLOSED.
FW:feedwater::: FAILURE OF 2-FW-FCY-2498 (FUSE) DRIVER CARD FOR 2-FW-FCV-2498.
FCY:Fuse driver card::: FAILURE OF 2-FW-FCY-2498 (FUSE) DRIVER CARD FOR 2-FW-FCV-2498.
FCV:flow control valve::: MANUAL TURBINE TRIP INITIATED DUE TO WELD FAILURE ON 1-FCV-1-104.
LOP:loss of offsite power::: OVERSPEED TRIP TESTING IAW LOP-TG-02
RCP:reactor coolant pump::: DEGRADED REACTOR COOLANT PUMP (RCP) SEAL ON RCP-3A. REPLACED SEAL.
FCV:flow control valve::: UNIT TAKEN OFFLINE TO REPAIR A PACKING LEAK ON THE 24 SG MAIN FWRV (FCV-447). REACTOR REMAINED CRITICAL.
PT:penetrant test::: REACTOR TRIP DURING THE PERFORMANCE OF SURVEILLANCE TEST 3PT-Q94M PRESSURIZER LEVEL ANALOG FUNCTIONAL, DUE TO A DEGRADED RELAY IN THE REACTOR PROTECTION LOGIC MATRIX. RELAY WAS REPLACED.
CF:column feed::: POWER REDUCTION DUE TO FEEDWATER VALVE 2CF-30 FAILED TO CLOSE IN THE REQUIRED TIME WHILE PERF VALVE STROKE TIMING TEST.
RC:reactor cavity::: UNIT SHUTDOWN TO REPAIR 2-RC-HSS-116
BVPS:Beaver Valley Power Station::: BVPS-1 manually tripped the reactor following a turbine trip while at approximately 46% power during startup from the 1R22 refueling outage due to a cable failue on 11/05/2013 at 17:48. Power generation resumed on 11/08/2013 at 18:07.
FW:feedwater::: UNIT MANUALLY SHUT DOWN DUE TO A STEAM LEAK FOUND ON FEEDWATER CHECK VALVE 2-FW-118-2.

PT:penetrant test::: MANUALLY SECURED THE TURBINE TO FACILITATE THE PERFORMANCE OF SURVEILLANCE TEST 3PT-V21, TURBINE GENERATOR OVERSPEED TRIP TEST.

MFP:main feed power::: AN AUTOMATIC TRIP RESULTED FROM A LOSS OF SUCTION ON THE A-MFP, TURBINE RUNBACK, AND SUBSEQUENT LOW STEAM GENERATOR LEVEL DURING THE RESTORATION OF A CONDENSATE PUMP FROM MAINTENANCE.

LER:licensee event report::: 2M29 SAFETY RELIEF VALVE MAINTENANCE OUTAGE RESULTED IN FORCED AND SCHEDULED LOSSES BECAUSE SRV INADVERTENTLY LIFTED WHILE POWERING DOWN. REQUIRED RESPONSE WAS REACTOR MANUAL SCRAM. REF LER-2-01-001.

MO:motor operator::: A PLANNED MANUAL SCRAM WAS INSERTED DUE TO A RISING TREND IN UNIDENTIFIED DRYWELL LEAKAGE. VALVE PACKING ON RWCU INLET VALVE MO-1201-85 WAS LEAKING. PACKING REPAIRED.

MO:motor operator::: REACTOR SHUTDOWN FOR MO-09.

RRC:radiation recorder controller::: PLANT DOWN TO REPAIR A SEAL ON RRC-P-1A.

PT:penetrant test::: AUTOMATIC REACTOR SCRAM WHILE PERFORMING SURVEILLANCE TEST 3PT-Q95,PRESSURIZER PRESSURE ANALOG FUNCTIONAL TEST.

SOV:solenoid operated valve::: REACTOR TRIP ON MSIV CLOSURE. REWORKED GRAY BOOT CONNECTORS FOR SOV-01-03D AND SOV-01-04D. REPLACED RLY-12K74.

CA:Charge amplifier::: REPAIR FEEDWATER VALVE (2CA-42).

RC:reactor cavity::: AUTOMATIC REACTOR TRIP DUE TO LOSS OF COOLANT FLOW >30% POWER FOLLOWING LOSS OF 2-RC-P-1B MOTOR.

HCV:hand control valve ::: CYCLE 17 REFUELING OUTAGE. Early shutdown of BFN2 on 3/14/13 due to RCIC turbine exhaust hand control valve (2-HCV-71-14). Entered U2R17 RFO 03/16/2013 at 12:00AM.

CW:case work::: Manual reactor trip 0900 on 1/9/14 due to TS 2.0.1(1) entry. All Raw Water pumps were declared inoperable at 0315 on 1/9/14 due to CW-14C, Traveling Screen Sluice Gate, being unable to close due to ice build up and stem damage.

UNIT:United Illuminating Co::: UNIT-1 AUTOMATIC REACTOR TRIP DUE TO A FAILURE IN THE TURBINE DIGITAL ELECTRO-HYDRAULIC CONTROL SYSTEM.

PT:penetrant test::: MANUAL REACTOR SCRAM DUE TO FAILURE OF PT-408B POWER SUPPLY, MAIN BOILER FEED PUMP SUCTION PRESSURE TRANSMITTER. (CR-IP2-2007-1046)

NON:notice of nonconformance::: Replaced a non-safety related 120 VAC regulating transformer which services the digital feedwater control logic. The transformer was showing signs of degradation beginning on August 30, 2013.
RE:radiation equipment::: MAIN GENERATOR WAS REMOVED FROM THE GRID TO REPAIR A LEAK ON A FILTER IN THE STATOR COOLING WATER SYSTEM. THE LEAK WAS REPAIRED AND THE MAIN GENERATOR WAS RE-TIED TO THE GRID.
HV:hand valve::: THE UNIT WAS SHUTDOWN TO: 1) REPAIR A HYDRAULIC LEAK IN THE ACTUATIOR FOR FEEDWATER BLOCK VALVE 3HV-4501 AND 2) REPLACE SECTION OF THE STEAM BYPASS LINE PIPING. NEITHER CONDITION PREVENTED CONTINUED PLANT OPERATION
FCV:flow control valve::: MANUAL REACTOR TRIP DUE TO 22 FEEDWATER (FW) FLOW OSCILLATIONS ATTRIBUTED TO FW CONTROL VALVE FCV-427 (LER-2004-001).
UV:undervoltage::: UNIT SHUTDOWN BY PROCEDURE DUE TO RETESTS FOR AUX FEEDWATER STEAM SUPPLY VALVE SGA-UV-138A.
HCV:hand control valve ::: PSL 2 experienced a manual reactor/turbine trip from full power on 11/12/2014 due to a malfunction with Main Feedwater Isolation Valve, HCV-09-2B.
FCV:flow control valve::: IT WAS DETERMINED THAT THE DIAPHRAGM HAD FAILED ON CONTROL VALVE 2-FCV-62-69. THE VALVE WAS REPAIRED AND THE SYSTEM WAS RETURNED TO SERVICE.
FT:fault tree::: On 09/28/15 at 20:46 the Hope Creek reactor scrammed. During performance of HC.IC-FT.SA-0003 (RRCS-Div 1 Channel B ATWS Recirc Pump Trip), RRCS automatically actuated on a high reactor pressure (>1071 PSIG) on both A and B Channel logic.
BD:blowdown::: LOSS OF STEAM GENERATOR BLOWDOWN FLOW FROM B SG CAUSED BY FAILURE OF 1BD-20. REPAIRS MADE.
PT:penetrant test::: UNIT SHUTDOWN DUE TO INADEQUATE TECHNICAL SPECIFICATION REQUIRED LEAK RATE TESTING OF CONTAINMENT ISOLATION VALVES (CIV). INADEQUATE TESTING OF CIVs WAS DUE TO INADEQUATE REFUELING TEST PROCEDURES (3PT-R25,3PT-R35).
RC:reactor cavity::: 2-RC-MOV-2591 DISC SEPARATED FROM STEM.
MOV:metal oxide varister::: 2-RC-MOV-2591 DISC SEPARATED FROM STEM.
FCV:flow control valve::: U2C14 MAINTENANCE OUTAGE TO REPAIR LEAKING MSRV'S AND THE 2-FCV-003-0077 VALVE.
RC:reactor cavity::: RC-3A Reactor Coolant Pump Seal Leakage
MFP:main feed power::: TRIP DUE TO MFP-B SHAFT FAILURE IN CONJUNCTION WITH STANDBY MFP OUT OF SERVICE FOR RECIRCULATION LINE REPAIRS

AND FOREIGN MATERIAL PARTIAL OBSTRUCTION AT THE NUMBER 4 STEAM GENERATOR FEEDWATER INLET.
FCV:flow control valve::: MANUAL REACTOR TRIP DUE TO DECREASING 23 STEAM GENERATOR LEVEL, ATTRIBUTED TO THE FAILURE OF FCV-437-SOV-E.
SOV:solenoid operated valve::: MANUAL REACTOR TRIP DUE TO DECREASING 23 STEAM GENERATOR LEVEL, ATTRIBUTED TO THE FAILURE OF FCV-437-SOV-E.
MSIV:Main steam isolation valve::: SCRAM AND SAFETY INJECTION SIGNAL DUE TO MSIV FAILURE. MSIV-3516 SPONTANEOUSLY CLOSED. INSTALLATION OF A NON-VENTED PIPE PLUG IN THE VALVE ACTUATOR CAUSED THE FAILURE. REPLACEMENT ACTUATOR WITH PROPER VENTING PATH WAS INSTALLED ON MSVIV-3516.
CF:column feed::: INVESTIGATE/INSPECTOR/REPAIR STEAM GENERATOR "A" FEEDWATER REGULATOR VALVE 2CF-32.
RC:reactor cavity::: 2RC-1 REPAIR
LER:licensee event report::: SCRAM DUE TO TURBINE CONTROL VALVE FAILURE. TURBINE CONTROL SYSTEM CIRCUIT CARD CONNECTION PIN PROBLEM WHICH CAUSED THE CLOSURE OF THE TURBINE CONTROL VALVES WAS REPAIRED. LER-2007-001 ISSUED MARCH 23, 2007, DOCUMENTS THE EVENT.
LER:licensee event report::: AUTOMATIC TRIP OCCURRED DUE TO A FAILURE OF 11 CEDM MOTOR GENERATOR LOCAL VOLTAGE ADJUST HANDSWITCH.
MO:motor operator::: MO-11 WAS TAKEN ON MAY 25, 2001 TO REPAIR STEAM BYPASS AND PRESSURE REGULATION CIRCUITRY. THE PLANT RETURNED TO 100% ON 5/30/01 AT 0316.
SRV:safety relief valve::: 1/20/13 at 21:37 shutdown initiated due to leaking SRV. Offline 1/21/13 at 05:45. All rods in at 09:01. SRV-203-3B repaired. Rx S/U commenced 1/22/13 at 10:21. Rx critical at 15:28. Synched to grid 1/23/14 at 11:21. Full power on 1/24/13 at 03:12.
RE:radiation equipment::: MANAGEMENT RE-VIEWED THE CIRCUMSTANCES FOR THE SHUTDOWN AND DETERMINED THAT THE OUTAGE WOULD BE CLASSIFIED AS FORCED DUE TO REGULATORY CONCERNS.
RE:radiation equipment::: MANAGEMENT RE-REVIEWED THE CIRCUMSTANCES FOR THE SHUTDOWN AND DETERMINED THAT THE OUTAGE WOULD BE RECLASSIFIED AS FORCED DUE TO REGULATORY CONCERNS.
NON:notice of nonconformance::: FAILURE OF THE UNIT 2 SAT NON-SEGREGATED BUS RESULTED IN THE LOSS OF OFFSITE POWER AND A UNIT 2 SHUTDOWN.
PT:penetrant test::: REACTOR TRIP DURING THE PERFORMANCE OF SURVEILLANCE TEST 3PT-Q94M PRESSURIZER LEVEL ANALOG FUNCTIONAL,

DUE TO A DEGRADED RELAY IN THE REACTOR PROTECTION LOGIC MATRIX. RELAY WAS REPLACED.

MS:main steam::: Unit shutdown to repair air leak on 1-MS-TV-101B

RC:reactor cavity::: Shutdown and cooldown to Mode 5 to replace seals on 2-RC-P-1A and 2-RC-P-1C

LO:lock open::: Reactor trip on SG lo-lo level. Caused by 24 SG FWR valve not responding to demand signal. Suspected cause is dirt/debris in the valve positioner. Root cause evaluation in progress.

LO:lock open::: Reactor trip on SG lo-lo level. Caused by 24 SG FWR valve not responding to demand signal. Suspected cause is dirt/debris in the valve positioner. Root cause evaluation in progress.

LT:leak testing::: OUTAGE DELAY DUE TO 1LT-5 "A" & "B" REACTOR VESSEL LEVEL INSTRUMENTATION.

RC:reactor cavity::: PLANT SHUTDOWN DUE TO THROUGH WALL WEEPAGE IN A SPOOL PIECE CONNECTING RELIEF VALVE RC-V89 TO A 12 INCH SUCTION LINE FOR TRAIN B RESIDUAL HEAT REMOVAL PUMP. REMAINED SHUTDOWN DUE TO BOTH TRAINS OF CONTROL BUILDING AIR(CBA) INOPERATIVE.

NON:notice of nonconformance::: The plant shutdown to repair a non-isolable steam leak upstream of a drain valve for an Atmospheric Steam Dump Valve. This is ASME Class II high energy piping that is required to be Operable per technical specifications.

EH:electrohydraulic::: UNIT WAS RAMPED DOWN TO APPROX. 8% POWER AND MAIN GENERATOR WAS REMOVED FROM SERVICE TO REPAIR 1-EH-TV-100 (AUTO STOP OIL INTERFACE VALVE). REACTOR REMAINED CRITICAL.

FW:feedwater::: POWER REDUCTION TO PERFORM MAINTENANCE ON 2-FW-P-1A AND 2-FW-P-1B.

MOD:motor operated disconnect::: TRANSFORMER T-MOD INSTALLATION. REACTOR NOT SHUT DOWN.

EHC:electrohydraulic control::: AUTO TRIP-EHC MALFUNCTION RESULTING IN TURBINE THROTTLE CONTROL VALVES DRIFTING CLOSED.

BI:background information::: THE UNIT EXPERIENCED AN AUTOMATIC REACTOR TRIP FOLLOWING AN INADVERTENT TURBINE TRIP FROM 100% OUTPUT DURING A SOLID STATE PROTECTION SYSTEM TRAIN B BI-MONTHLY TEST.

RE:radiation equipment::: LOSS OF LOAD DUE TO FAILURE OF A STATIC LINE BETWEEN THE PLANT AND THE SWITCHYARD. MADE NECESSARY REPAIRS AND WILL RE-EVALUATE PALISADES RESPONSE TO SOER 99-1, LOSS OF GRID, RECOMMENDATION 3.

PT:penetrant test::: PLANT SHUTDOWN FOR AN INOPERABLE 480 VOLT BUS 6A DUE TO A FAILURE OF THE 32 RESIDUAL HEAT REMOVAL (RHR) PUMP CIRCUIT BREAKER TO OPEN FOLLOWING PERFORMANCE OF SURVEILLANCE TEST 3PT-M18,"RHR PUMP FUNCTIONAL TEST." 480 BUS6A REMAINED ENERGIZED BUT OPERABLE.

NON:notice of nonconformance::: MAIN STEAM FLOW MEASURED WAS NON-CONSERVATELY HIGH DURING POWER ASCENSION WITH RELATION TO SAFETY SYSTEM ACTION PARAMETERS. UNIT TAKEN OFFLINE.

LV:leaky valve::: POWER REDUCTION TO REPLACE POSITIONER ON FEEDWATER HEATER NORMAL DRAIN LEVEL CONTROL VALVE 2-LV-2509.

MS:main steam::: SHUTDOWN IN ACCORDANCE WITH TECH SPEC 4.15.C.1 TO REPAIR A PINHOLE LEAK ON TWO INCH MAIN STEAM PIPING. THE LINE GOES TO 1-MS-TD-4 FROM THE "B" MAIN STEAM LINE.

TD:theoretical density ::: SHUTDOWN IN ACCORDANCE WITH TECH SPEC 4.15.C.1 TO REPAIR A PINHOLE LEAK ON TWO INCH MAIN STEAM PIPING. THE LINE GOES TO 1-MS-TD-4 FROM THE "B" MAIN STEAM LINE.

AO:abnormal occurrence::: Reactor scrammed 8/22/15 at 16:27 due to the unplanned closure of MSIV AO-203-1C. Reactor startup commenced 8/24/15 at 21:47. Reactor critical 8/25/15 at 00:47. Generator synched 8/25/15 at 17:54. Reached 100% power 8/25/15 at 06:37.

FW:feedwater::: U1 TAKEN OFFLINE FOR 1-FW-E-6B, FEEDWATER HREATER TUBE REPAIR.

FW:feedwater::: REACTOR TRIP OCCURRED DUE TO FAILURE OF FW-7B, MAIN FEEDWATER FLOW CONTROL VALVE.

VDC:ventilation duct chase::: FAILURE OF UNIT 2-15VDC TRAIN "A" POWER SUPPLY IN MSIV/FWIV CONTROL CABINET RESULTED IN CLOSURE OF ALL MSIVS, RX TRIPPED ON HIGH PRESSURIZER PRESSURE, AND SUBSEQUENT LOSS OF HEAT SINK. UNIT RETURNED TO SERVICE ON 08/28/00 .

D. unknown.txt

TADOT

MILS

HU

RMSC

AB

STGE

OE

LUG

INPO

REC
ISM
PBTP
JIB
SP
EX
UIT
EDH
ESFSAS
BOP
EENT
OCBS
DFS
IRMS
AMAG
TSEO
GC
ICES
ODS
TBED
INOP
JUNO
OTBD
HR
SWAGR
PLP
MGMT

E. reformat.txt

ON-LINE:ONLINE
OFF-LINE:OFFLINE
MAIN-TURBINE:MAIN TURBINE
OFF-SITE:OFFSITE
MOTOR-GENERATOR:MOTOR GENERATOR
SHORT-CIRCUIT:SHORT CIRCUIT
IN-PROGRESS:IN PROGRESS
AIR-EJECTOR:AIR EJECTOR

TROUBLE-SHOOT:TROUBLESHOOT
CARRY-OVER:CARRYOVER
ELECTRO-HYDRAULIC:ELECTROHYDRAULIC
TURBINE-DRIVEN:TURBINE DRIVEN
IN-SERVICE:IN SERVICE
IN-LEAKAGE:INLEAKAGE
AIR-LEAKAGE:AIR LEAKAGE
MOTOR-DRIVEN:MOTOR DRIVEN
TURBINE-DRIVEN:TURBINE DRIVEN
TURBINE-GENERATOR:TURBINE GENERATOR
OUT-OF-SERVICE:OUT OF SERVICE
TURBINE-DRIVE:TURBINE DRIVE
POWER-LOAD-UNBALANCE:POWER LOAD UNBALANCE
SHUT-DOWN:SHUTDOWN
NON-SAFETY:NON SAFETY
SHORT-CIRCUIT:SHORT CIRCUIT
LEAK-OFF:LEAK OFF
LOW-VOLTAGE:LOW VOLTAGE
MAIN-TURBINE-:MAIN TURBINE
THERMALLY-INDUCED:THERMALLY INDUCED
NO-LOAD:NO LOAD
OFF-GAS:OFF GAS
PRE-PLANNED:PREPLANNED
NON-ISOLABLE:NONISOLABLE
ANTI-MOTORING:ANTIMOTORING
CUT-BACK:CUTBACK
TACHOMETER-GENERATOR:TACHOMETER GENERATOR
SEAL-IN:SEAL IN
POST-REFUEL:POST REFUEL
UNDER-FREQUENCY:UNDERFREQUENCY
TRIP-HIGH:TRIP HIGH
POWER-UP:POWERUP
DOWN-POWERING:DOWNPOWERING
RANGE-HIGH:RANGE HIGH
DUAL-UNIT:DUAL UNIT
CHAMBER-TO-DRYWELL:CHAMBER TO DRYWELL
LEAK-RATE:LEAK RATE

START-UP:START UP
OVER-SPEED:OVERSPEED
HIGH-FLUX:HIGH FLUX
FIFTH-POINT:FIFTH POINT
SUB-CRITICAL:SUBCRITICAL
MOTOR-OPERATED:MOTOR OPERATED
OFF-GRID:OFF GRID
ELECTRO-HYDRUALIC:ELECTROHYDRUALIC
HI-HI:HIGH HIGH
BI-MONTHLY:BIMONTHLY
SYSTEM-ELECTRICAL:SYSTEM ELECTRICAL
RUN-BACK:RUNBACK
BLIZZARD-INDUCED:BLIZZARD INDUCED
STEP-UP:STEPUP
TURBINE-PUMP:TURBINE PUMP
END-BELL:END BELL
IN-PLANT:IN PLANT
PHASE-TO-GROUND:PHASE TO GROUND
LOAD-LTD:LOAD LETDOWN
VOLTAGE-TO-GROUND:VOLTAGE TO GROUND
RE-REVIEWED:REVIEWED
AUTO-STOP:AUTO STOP
HOT-STANDBY:HOT STANDBY
LO-LO:LOW LOW
DE-ENERGIZED:DEENERGIZED
LOW-LOW:LOW LOW
INTER-SYSTEM:INTERSYSTEM
BREAK-IN:BREAK IN
ELCTRO-HYDRAULIC:ELECTROHYDRAULIC
RE-EVALUATE:REEVALUATE
BODY-TO-BONNET:BODY TO BONNET
NON-VENTED:NON VENTED
NON-SEGREGATED:NON SEGREGATED
LIKE-FOR-LIKE:LIKE FOR LIKE
RE-TIED:RETIED
LOCK-OUT:LOCKOUT
MID-POSITION:MIDDLE POSITION

WEATHER-RELATED:WEATHER RELATED
HIGH-HIGH:HIGH HIGH
RAMP-UP:RAMPUP
CONTROL-DUE:CONTROL DUE
QUAD-VOTER:QUAD VOTER
LOAD-ACTUAL:LOAD ACTUAL
UN-COUPLED:DECOUPLED
ISO-PHASE:ISOPHASE
ANTI-REVERSE:ANTIREVERSE
CROSS-TIED:CROSS TIED
UN-ISOLATABLE:UNISOLATABLE
OVER-FREQUENCY:OVER FREQUENCY
OFF-NORMAL:OFF NORMAL
IN-LEAKAGE-RX:IN LEAKAGE REACTOR
PRE-EVENT:PRE EVENT
RE-SYNCHED:RESYNCHED
MID-CYCLE:MIDCYCLE
SHELL-SIDE:SHELL SIDE
OVER-CURRENT:OVER CURRENT
BORG-WARNER:BORG WARNER
CHANGE-OUT:CHANGEOUT
SIX-INCH:SIX INCH
PART-LENGTH:PART LENGTH
END-SHIELD:END SHIELD
HOLD-DOWN:HOLD DOWN
CLEAN-UP:CLEANUP
SMALL-BORE:SMALL BORE
NON-ROUTINE:NON ROUTINE
FIRE-DAMAGED:FIRE DAMAGED
IS-OPERATED:IS OPERATED
SCRAM-CODE:SCRAM CODE
THIRD-PARTY:THIRD PARTY
MINI-FLOW:MINI FLOW
END-OF-CYCLE:END OF CYCLE
MINI-OUTAGE:MINI OUTAGE
SERVO-STRAINERS:SERVO STRAINERS
DOWN-POWER:DOWN POWER

AIR-BOUND:AIR BOUND

TORUS-TO-DRYWELL:TORUS TO DRYWELL

50-DH-350:AIR CIRCUIT BREAKER

RRCS-DIV:REDUNDANT REACTIVITY CONTROL SYSTEM

V-28-21:VENTILATION SYSTEM ISOLATION VALVE

*https://www.nrc.gov/docs/ML0037/ML003751413.pdf

LVC-1127B:HEATER DRAIN TANK LEVEL CONTROL

*https://adamswebsearch2.nrc.gov/webSearch2/main.jsp?AccessionNumber=ML13126A379

SIA-V056:SAFETY INJECTION DRAIN VALVE

*https://www.nrc.gov/docs/ML0410/ML041040027.pdf

1-CH-TV-1204B:OUTSIDE CONTAINMENT ISOLATION VALVE

*https://www.nrc.gov/docs/ML0935/ML093560851.pdf

2-AOP-:ABNORMAL OPERATING PROCEDURE CODE

CR-XXX-:CONDITION REPORT CODE

1-CS-8364B:SEAL INJECTION DRAIN VALVE

*https://www.nrc.gov/docs/ML1331/ML13310C184.pdf

U-25000-11:TRANSFORMER *https://www.nrc.gov/docs/ML9932/ML993240318.pdf

GB-1-02:GENERATOR BREAKER *https://www.nrc.gov/docs/ML0101/ML010120458.pdf

RV-200:PRESSURIZER SAFETY RELIEF VALVE

*https://www.nrc.gov/docs/ML0703/ML070330649.pdf

U-25000-12:TRANSFORMER *https://www.nrc.gov/docs/ML2021/ML20212A349.pdf

MCC-3A1:MOTOR CONTROL CENTER *https://www.nrc.gov/reading-rm/doc-collections/gen-comm/circulars/1977/cr77003.html

DB-50:REACTOR TRIP BREAKERS *https://www.nrc.gov/reading-rm/doc-collections/gen-comm/bulletins/1983/bl83001.html

U1-OPS:UNIT OPERATIONS

HT-ACE:HIGH TIER APPARENT CAUSE EVALUATION

*https://www.nrc.gov/docs/ML1513/ML15133A264.pdf

RHR-1-RV-8708:RESIDUAL HEAT REMOVAL PUMP DISCHARGE HEADER TO RELIEF VALVE *https://www.nrc.gov/docs/ML1616/ML16165A280.pdf

3-ISV-069-0500:ISOLATION VALVE

1P-029-T:TURBINE DRIVEN AUXILLARY FEEDWATER PUMP

*https://www.nrc.gov/docs/ML0037/ML003714618.pdf

CV-3-200B:CONTAINMENT ISOLATION VALVE *https://www.nrc.gov/reading-rm/doc-collections/event-status/event/2003/20030429en.html

NAN-S01:BUSES *https://www.nrc.gov/docs/ML1928/ML19284D677.pdf

CV-31385:VALVE *https://www.nrc.gov/docs/ML0200/ML020090090.pdf

RV-4-551A:PRESSURIZER SAFETY VALVE

*https://www.nrc.gov/docs/ML1332/ML13329A125.pdf

PUMP-1A:PUMP MOTOR OIL

RV-2A: MOISTURE SEPARATOR REHEAT PILOT RELIEF VALVE

2RCS-P-1B:REACTOR COOLANT SYSTEM PUMP

VPI-303:CORE SPRAY SYSTEM *https://www.nrc.gov/docs/ML2024/ML20248B409.pdf

CV-1057:PRESSURIZER SPRAY CONTROL VALVE

*https://www.nrc.gov/docs/ML1604/ML16047A125.pdf

DHV-03:DECAY HEAT REMOVAL SYSTEM ISOLATION VALVE

*https://www.nrc.gov/docs/ML0124/ML012420074.pdf

NI-42:POWER RANGE NUCLEAR INSTRUMENT CHANNEL

*https://www.nrc.gov/docs/ML2021/ML20214F805.pdf

V-28-22:VENTILATION SYSTEM ISOLATION VALVE

*https://www.nrc.gov/docs/ML0037/ML003751413.pdf

2-MRV-220:STEAM STOP VALVE

*https://www.nrc.gov/reactors/operating/oversight/reports/cook_2002003.pdf

CK-ES-3332:SWING CHECK VALVE *https://www.nrc.gov/reading-rm/doc-collections/gen-comm/info-notices/2000/in00021.html

IV-38-01:SHUTDOWN COOLING ISOLATION VALVE

*https://www.nrc.gov/docs/ML1212/ML12125A058.pdf

RV-19:RELIEF VALVE

*https://www.nrc.gov/reactors/operating/oversight/reports/wnp_2002006.pdf

CONP-P-2B:CONDENSATE PUMP

2VBB-UPS3B:UNINTERRUPTIBLE POWER SUPPLY

*https://www.nrc.gov/docs/ML1413/ML14135A187.pdf

2CA-42:ISOLATION VALVE *https://www.nrc.gov/docs/ML2005/ML20056C428.pdf

CR-IP3-2007-1834:CONDITON REPORT

CR-IP3-2007-2130:CONDITON REPORT

CR-1P2-2006-6658:CONDITON REPORT

CR-GGN-2013-00319:CONDITON REPORT

CR-IP2-2007-2208:CONDITON REPORT

CR-IP2-2006-1011:CONDITON REPORT

CR-IP3-2007-1775:CONDITON REPORT

CR-WF3-1999-1207:CONDITON REPORT

CR-1P3-2005-3054:CONDITON REPORT

CR-GGN-2014-3131:CONDITON REPORT

CR-PLP-2012-0078:CONDITON REPORT

CR-GGN-2013-00083:CONDITON REPORT

CR-JAF-2013-00864:CONDITON REPORT

CR-IP3-2006-2071:CONDITON REPORT

CR-IP3-2006-2255:CONDITON REPORT

MO-7071:MOTOR OPERATED VALVE

*https://www.nrc.gov/docs/ML0729/ML072970515.pdf

2-0305-101-18-27:SCRAM INSERT ISOLATION VALVE

*https://adamswebsearch2.nrc.gov/webSearch2/view?AccessionNumber=ML14225A200

2E-7B:FEEDWATER HEATER *https://www.nrc.gov/docs/ML0912/ML091260792.pdf

1H13P637-PS21:POWER SUPPLY *https://www.nrc.gov/docs/ML1412/ML14122A458.pdf

1X-04:TRANSFORMER

VPI-3-3:CHECK VALVE *https://www.nrc.gov/docs/ML2019/ML20196A452.pdf

MS-0063:STEAM GENERATOR ATMOSPHERIC RELIEF VALVE BLOCK

*https://www.nrc.gov/reactors/operating/oversight/reports/cp_2002005.pdf

71T-1B:TRANSFORMER

1-1301-17:REACTOR CORE ISOLATION COOLING VALVE

2P-15B:SAFETY INJECTION PUMP

11-BUS-007:LOAD CENTER

1-0303-3B:ELECTROMATIC RELIEF VALVE

2-CK-075:INJECTION CHECK VALVE

FCV-427:STEAM GENERATOR MAIN FEED REGULATING VALVE

*https://www.nrc.gov/docs/ML0430/ML043080311.pdf

CH-240:BACKPRESSURE CONTROL VALVE

MV-09-01:MAIN FEEDWATER PUMP ISOLATION VALVE

*https://www.nrc.gov/docs/ML1722/ML17229A585.pdf

MO-10:MOTOR OPERATED VALVE

MO-09:MOTOR OPERATED VALVE

PCB-341:SWITCHYARD BREAKER *https://www.nrc.gov/reading-rm/doc-collections/event-status/part21/1997/1997333.html

2-FCV-075-0025:FLOW CONTROL VALVE

*https://www.nrc.gov/docs/ML0712/ML071210012.pdf

1-XA-55-22:REACTOR LOW WATER LEVEL

1XA-55-1-29:NEUTRON MONITOR SYSTEM

CRD-24:CONTROL ROD DRIVE

1P-1A&B:REACTOR COOLANT PUMPS

1&2P-2P-28A&B:MAIN FEEDWATER PUMPS

160-12-21:PRESSURE SWITCH

2-FR-240:FEEDWATER RELIEF VALVE POSITIONER

1HP-27:HIGH PRESSURE INJECTION VALVE

06-01:FUEL LEAK

252-2104:DISLODGED BREAKER

10-11:OIL CIRCUIT BRAKER

1-11:OIL CIRCUIT BREAKER

1F42-11:DISCONNECT SWITCH

C11-N654B:INSTRUMENT TUBING

2E11-F050B:INJECTION LINE CHECK VALVE

01-03:MAIN STEAM ISOLATION VALVE

1P-11A:COMPONENT COOLING WATER PUMP

P-32A:SERVICE PUMP

1X-01C:TRANSFORMER

O6-01:FAILED FUEL

2S32-R017:RECORDER

A0-203-1C:MAIN STEAM ISOLATION VALVES

2-0203-3D:ELECTROMATIC RELIEF VALVE

3HD-149:STEAM LEAK

26-27:DIRECTIONAL CONTROL VALVE

1P-1B:REACTOR COOLANT PUMP SEAL

A0-203-1B:MAIN STEAM ISOLATION VALVES

1-0303-3C:ELECTROMATIC RELIEF VALVE

1-0303-3E:ELECTROMATIC RELIEF VALVE

2B21-F013L:SAFETY RELIEF VALVES

30-48:LEVEL TRANSMITTER

552-1105:BREAKER

1P-25B:CONDENSATE PUMP

68-03:VACUUM BREAKER

2B21-F016:STEAM ISOLATION VALVE FOR VALVE PACKING LEAK

1HD-26:HEATER DRAIN VALVE

BFD-64-10:ISOLATION VALVE

BFD-64-10:ISOLATION VALVE

*https://www.nrc.gov/cdn/legacy/reactors/operating/oversight/2017q1/ip3_pi.pdf

1-LS-006-0206:MAIN FEEDWATER PUMP TURBINE CONDENSER DRAIN TANK LEVEL SWITCH

2-CK-075:INJECTION CHECK VALVE

VPI-303:CHECK VALVE

BFD-1:MAIN BOILER FEED PUMP DISCHARGE CHECK VALVE

K-7B:TURBINE DRIVER

P-29C:MAIN SHAFT DRIVE LUBE OIL PUMP

P-50D:PRIMARY COOLANT PUMP

P-50C:PRIMARY COOLANT PUMP SEAL

2RC-1:PRIMARY COOLANT SYSTEM LOOP

*https://citeseerx.ist.psu.edu/document?repid=rep1&type=pdf&doi=f43fb54e7bccc3c500fb1912d686374763e4fb16

#####

NS04A:MAIN STEAM ISOLATION VALVE

1A2:REACTOR COOLANT PUMP HIGH SEAL FLOW

1D1:HEATER DRAIN PUMP

1D2:HEATER DRAIN PUMP

1B:MAIN FEEDWATER PUMP MECHANICAL SPEED CONTROL

1A1:REACTOR COOLANT PUMP

1A:MAIN FEEDWATER PUMP

F. unknownWordNumbers.txt

116T

2B1

CR-JAF-2013-00864

2M32

A1

(CR-GGN-2013-00083)

2A

106T

CAP060656

3R14

1A2

1B

L2R10

L2R11

"1B'

CR-IP3-2006-2071

2R14

A 1

1E

1A
SP1036
120V
M1B
SGK05A
D50
GV-1
2'S
LI2F48
10CRF50 54(F)
G1
2P99
O1C25
1M36
K1
LI1F40
B33
U1C24
Q2P03
Q1F47
13BF19
1ES01OB
1-5A
SGK05A
SG#2
2ND
OCT 28 2003
1754H
2HV4052
4160V
2NAB03
32A
3B
32L
1ST
22BF19
1B2

C14
CR-1P2-2006-6658
6A
2HV6500
2M23
"1B"
ISM-7
AT17:07
59GG
1R22
3C
1F
CR-1P3-2005-3054
3A
T0
"1C"
CR-IP3-2007-2130
MODE2
D6
2FW009D
3F2
1S18819C
12BF19
86P
22A
2O
BOTHU1
1R15
2PS3
"3B"
LI1F54
B119M3
10CFR50 54(F)VALVES
1NCRD5850
B119R1
Q1P02
DG4

IP2
2F40
500KV
SF6
3RD
LINE-985
1R24
2EOC17
CR-IP3-2007-1775
OPENED:10/20/12
1D2
ISM-7
G9
G-1
Z1M06
CR-PLP-2012-0078
2SJ10
CRITICAL 3/24/14
2B53
3 5 2A
W97
1M27
1H
12-INCH
31/32B
"2D"
H-8
15V
7B
1'S
STATION-2
144D
2D
U2R11
2C1
1M16
4B

EN#50649
4-INCH
10CFR50 82(A)(1)(II)
"1A1"
Q2P01
1D
L1P02
2D3
Q1P01
C1M21
04:02AM
U2C14
5A
Y94
2H
U1C21
U2C17
50D
H-8
2M29
U-2
124A
W93
H2
AFFV58C
4 15 C 1
2RS
1D2
D-11-2
C1R09
AND12
15B
10CFR50 829(A)(1)(I)
3 8 1 F
1F12
ON10/05/00
10CFR50 54(F)

2 1O
CR-WF3-1999-1207
PB03
11A
Q2P01
B2F26
OR15
D2
2S
21ST
Q2P03
16-DAY
3E
CR-GGN-2014-3131
1M
U114
133U1
O-29-2
2M52
3DAY
ST11
SYSTEM 11/09/12
U3
XK1
1F24
1C
M1A
1F2903CS
2F
87DP
W93
K620
Q1F43
IR#1101855
71D
200MWE GENERATOR
2P98

NO 3
"3D1"
345KV
Q1M13
STATION-1
N42
B2F18
1AND3
"2A"
2A
Q2M17
XK3
3D
1F2902CS
IR#1101858
1J
L1P03
K-8
1POAC5
1A1
NO 5
4A
12B
1F23
LI2M51
2R
1C3
2B
D2M12
1-6A
1G06
UF113
D5
2C2
1D3
N-11
U-3

1F38
EL66'
T1R15
LI1F50
3RC4
CR01828253
1-II
2C
B2M05
1MT1
2CC
11Â
2-OUT-OF-3
MU021
H-12
O3VOC24
L2M17
"3D2"
U2
JULY4
480V
L2P01
B120F1
1F25
U1
2HV8701B
D2M11
"1C2"
ISM-1
RATE(3)REPAIR
L1R11
N-44
1SM7
1W
2F31
S01
(CR-GGN-2013-00319)

Q1R21
12W
1EOC16
C1
87T-2
10CFR50 54(F)
CRITICAL:10/20/12
12OV
2C
L2001
1RF09
SG#1
10CFR50 54(F)NRC
1B
UNIT-1
1-I
2KXB
D1
NO 7
CRITICAL 3/25/14
CRITICAL:11/06/12
CR-IP2-2006-1011
1NI1
2CS
F020
"3B'
G-6
T-1B
Q2P02
P1A
31C
CR-IP3-2006-2255
11RFO
C1R15
2B2
LI1F54
B217M1

CR-IP3-2007-1834
3006B
2-N51
130B
BREAKER 3/20/14
U2-
CR-IP2-2007-2208
2A2
G3300F120
30-INCH
Q1P01
1HP5
D3M19
12O
2B
R49
1B2
HOURS/43
3A2
A2F37
"2B"
1PS1
1M30
N43
15VDC
2M23
B220M2
U1-OPS
SP1003
133V
SA036D
2R23
125V
1650HR
1A2
1C
N42

3 8 1F

T S 3 2 2 ACTION

5C

1257P

Q1R19

(2)EVALUATION

E089

B220R1

74/HR

"3B2"