Sensitivity of Machine Learning Algorithms to Dataset Drift for the Natural Language Processing Application of Spam Filters

by

Tonya Fields

Submitted in partial fulfillment
of the requirements for the degree of
Doctor of Professional Studies
in Computing

at

Seidenberg School of Computer Science and Information Systems

Pace University

October 2021

We hereby certify that this dissertation, submitted by Tonya L. Fields, satisfies the dissertation requirements for the degree of *Doctor of Professional Studies in Computing* and has been approved.

Lixin Tao

Dr. Lixin Tao
Advisor of Dissertation Committee

Date: October 26, 2021

Susan Feather-Gannon
Susan Feather-Gannon (Nov 8, 2021 11:52 EST)

Dr. Susan Feather-Gannon
Dissertation Committee Member

Date: October 26, 2021

Istvan Barabasi
Istvan Barabasi (Nov 11, 2021 10:20 EST)

Dr. Istvan Barabasi
Dissertation Committee Member

Date: October 26, 2021

Seidenberg School of Computer Science and Information Systems
Pace University

Abstract

Sensitivity of Machine Learning Algorithms to Dataset Drift for the Natural Language Processing Application of Spam Filters

by
Tonya Fields

Submitted in partial fulfillment
of the requirements for the degree of
Doctor of Professional Studies
in Computing

October 2021

The datum used for many machine learning projects drifts over time. The problem of dataset drift causes the performance of the models to deteriorate. This can have devastating consequences for critical real-world applications that rely on the accuracy and robustness of the model, such as autonomous vehicles, fraud detection, medical diagnosis, etc. The resilience to dataset drift will impact the degree and ways the model suffers in production.

The impact of data drift depends on the selected algorithms, as well as the application domains. Application domains involving graphics and pattern recognitions, such as traffic-toll systems and image classification are less likely to suffer from data drift because the data used for training is static and is expected to have the same distribution that will be seen in production. On the other hand, the output of daily business processes that rely on natural language understanding for text classification involving user behavior are subject to change over time.

We research the stability of four popular high-performing machine learning algorithms for the text-classification task of email spam classification. In this work, we compare the resilience to dataset drift of the Random Forest (RF) algorithm, Convolutional Neural Network (CNN), Long Short-Term Memory (LSTM) Network, as well as the pre-trained transformers architecture of the Bidirectional Encoder Representation from Transformer (BERT) developed by Google. To simulate various amounts of data drift, we create four hybrid datasets using the *SpamAssassain* benchmark dataset, combined with various percentages of emails from the *Enron* benchmark dataset.

Our study found that for the specific Natural Language Understanding tasks of text classification for use in spam filters, the RF, CNN, and LSTM were less likely to suffer from data drift. These three models suffered only a 1% loss compared to an 11% loss for BERT when using a dataset representing 60% data change. When using a dataset representative of 90% data change, the BERT model suffered a 16% loss compared to the 7% loss of the RF and LSTM, and the 6% loss of the CNN. Overall, we found the RF, CNN, and LSTM algorithms were less likely to be impacted by dataset drift when used for spam filter applications.

Acknowledgements

My first expression of gratitude is to God Almighty. Completing this work is a testament of His Power, Glory, Grace, and Mercy. Thank you, Bishop T. D. Jakes for your sermons of Faith and Perseverance.

Thank you to Dean Dr. Jonathan Hill who championed me into the program. I have the distinction of being the very last person accepted into the DPS in Computing at Pace University. Thank you to Dr. Lixin Tao for his guidance. He has served as an invaluable mentor for this research, providing insight and direction. I thank my committee members: Dr. Susan Feather-Gannon, Dr. Istvan Barabasi, Dr. Charles Tappert, and the late Dr. Ronald Frank, who made the time in their jammed-packed schedules to help me. Screaming THANK YOU to my "Team Two" colleagues: Lisa Ellrodt, Dr. Suzanna Schmeelk, Dr. Ashley Haigler-Findlay, and Ion Freeman. I thank God that I was placed on your team!

Special thanks to my Norfolk State University work family and mentors, beginning with the late Dr. Sandra J. DeLoatch whose guidance and influence continue to be a source of inspiration for so many of us—she will always be "Still Here." Thank you to Dr. George Hsieh for it was his Center of Excellence in Cybersecurity funded by the DoD Air Force Research Lab that provided the initial project motivation, as well as equipment and laboratory support to launch this research. Thank you for bringing me onboard your project. I thank Dr. Aurelia Williams, a Pace alumna for paving the way and being such an inspirational mentor. Thank you, Dr. Jonathan Graham, for your kind and steadfast encouragement and support for so many hundreds of students—we remain forever in your debt. Thank you Dr. Chenou for your support, you helped my academic professional maturity in many ways.

I am so very thankful for the love and support of my family. Thank you to my loving husband Joel Fields (Foe & Five), mother Patricia "Patty Cakes," grandparents Wilma & Joe, Mama Blanche & Richard. I thank my beloved Geneva Moore and my father Johnny Brown, both of whom passed away when I was a young girl—proof that it is the quality and depth of love that endures through the ages. I thank a host of uncles and aunts, most especially my Auntie Rita, Auntie Bonnie Blue, and Auntie Cecilia. I thank my brothers, Johnny and Brett, my big sister, Patrice Jackson, and my little sister, Vicki Jackson, for loving the real me and for always having my back! I extend a special thank you to our children from our blended family—Larry, Dondria, Shy, Gia, and Matthew, who have enriched my life on so many levels. To Quinton I say, "I am so glad you chose me when you were in Heaven—you inspire my very best effort every time. Being your mother has been the highest privilege of my life and I am so proud of the man that you have become."

I thank all of those who encouraged and prayed for me, especially my church family: Apostle Joan and Pastor Edward Henderson. Finally, I thank my dearest friends: Missy Duvea, Crystal Naylor, Keisha Lucky, Tracy Finley, Cynthia Duhe-Harris, Melody & John Matthews, Kim Tynes Hibbler, Candy King, and Tanya Weaver. Thanks to the Power and Grace of God Almighty, I can now do as suggested by Henry David Thoreau and "Go confidently in the direction of your dreams! Live the life you've imagined."

Table of Contents

Abstract iii

List of Tables vii

List of Figures ix

Chapter 1 Introduction 1

1.1 The Problem of Data Drift 1

1.2 Algorithms and Applications Differ in Data Drift Behavior 2

1.3 Problem Statement 4

1.4 Solution Methods 4

1.5 Contributions 5

1.6 Dissertation Roadmap 5

Chapter 2 Review of the Literature and Background 7

2.1 Related Works in the Literature 7

2.2 Background on Natural Language Preprocessing Techniques 9

2.3 Background on Text Feature Extraction Methods 13

2.4 Algorithms in Our Study 17

Chapter 3 Dataset Drift Sensitivity Research Methodology and Design 23

3.1 Dataset Shift: Formal Definitions 23

3.2 Examples of Data Drift in Real World Application 26

3.3 Existing Measures to Mitigate Data Drift 27

3.4 Addressing Gaps in Data Drift Research 28

3.5 Research Question 28

3.6 Solution Methodology 29

3.7 Evaluation Metrics 34

3.8 Expected Achievement 35

3.9 Assumptions 35

3.10 Experimental Solution Design and Workflow 35
Chapter 4 Experiment Design and Findings 36
4.1 Data Preprocessing Implementation 36
4.2 Model Hyperparameter Selections and Implementations 41
4.3 Experimental Result and Data Analysis 56
4.4 Experiment Highlights: The Importance of Stopwords 67
4.5 Environment (Hardware, Software, Tools) 73
Chapter 5 Conclusion 76
5.1 Conclusion 76
5.2 Contributions 77
5.3 Future Work 79
References 81

List of Tables

Table 1 NLP Preprocessing: Tokenization, Lemmatization, Parts of Speech, and Stopwords 11
Table 2 Email Composition of Benchmark Datasets 30
Table 3 Email Composition of Hybrid Datasets 30
Table 4 Confusion Matrix General Structure 34
Table 5 Email Preprocessing 37
Table 6 Random Forest Hyperparameter Tuning 42
Table 7 Random Forest Pseudo Code Source: Adapted from [86] 43
Table 8 Random Forest Model Implementation 43
Table 9 CNN Hyperparameter Tuning 44
Table 10 CNN Model Implementation 45
Table 11 LSTM Hyperparameter Tuning 46
Table 12 LSTM Experiment Model Design Implementation 47
Table 13 BERT Hyperparameter Tuning 49
Table 14 BERT Model Design Implementation Source: Adapted from [92], [93] 50
Table 15 Baseline Performance Comparison in Terms of Overall Accuracy 57
Table 16 $BERT_{BASE}$ Results from Table 1 in [28] 58
Table 17 Random Forest Drift Analysis 59
Table 18 CNN Dataset Drift Performance Across All Metrics 61
Table 19 LSTM Dataset Drift Performance across all Metrics 62
Table 20 BERT Dataset Drift Performance Across All Metrics 64
Table 21 Dataset Drift Performance All Models Baseline vs. Drift Scores 66
Table 22 $BERT_{BASE}$ Baseline Performance Across Various Parameters 71
Table 23 Bert $_{LARGE}$ Baseline Performance Across Various Parameters 72
Table 24 List of Software and Tools 74

Table 25 Training Tutorials, Videos, and Guides..... 75

List of Figures

Figure 1 SpaCy Results for Named Entity Recognition on Sample Pace Corpus 12
Figure 2 SpaCy Results—Dependency Parsing..... 12
Figure 3 Example Word2Vec and GloVe Word Embedding. Image Source [35]..... 15
Figure 4 Example Random Forest - 3 Tress Vote. Source [39]..... 18
Figure 5 Simplified CNN Architecture. Source [46]..... 19
Figure 6 Raw Email Structure before Preprocessing. Source of Email: [68] 32
Figure 7 Email after Header Removed 33
Figure 8 Simplified Research Design. Source: Adapted from [123] 36
Figure 9 Most Frequent Non-Spam Words—Stopwords Included..... 38
Figure 10 Most Frequent Non-Spam Words—Stopwords Removed 39
Figure 11 Most Frequent Spam Words—Stopwords Removed 40
Figure 12 Emails before Cleaning 40
Figure 13 Emails after Cleaning 41
Figure 14 Baseline Performance in Terms of Accuracy: All Models..... 57
Figure 15 Random Forest Dataset Drift Performance: All Metrics..... 60
Figure 16 Random Forest Baseline Vs. Dataset Drift Metric: Accuracy 60
Figure 17 CNN Dataset Drift Performance Across All Metrics..... 61
Figure 18 CNN Dataset Drift Performance Metric: Accuracy 62
Figure 19 LSTM Dataset Drift Performance across all Metrics..... 63
Figure 20 LSTM Dataset Performance Metric: Accuracy..... 63
Figure 21 BERT Dataset Performance across all Metrics 64
Figure 22 BERT Dataset Drift Performance Metric: Accuracy..... 65
Figure 23 Drift Loss across all Models in Terms of Accuracy..... 67
Figure 24 Random Forest Baseline Stopwords Removed from Training Dataset..... 68

Figure 25 Random Forest Baseline Stopwords Included in Training Dataset 68
Figure 26 Random Forest - Performance with and Without Stopwords 68
Figure 27 BERT- With Stopwords in Training Dataset 70
Figure 28 BERT-Stopwords Removed 70
Figure 29 BERT- With and Without Stopwords 70

Chapter 1

Introduction

1.1 The Problem of Data Drift

When training supervised machine learning models, a single large historical dataset is split into a train file and test file, therefore the train and test files are said to have been generated from the same distribution. In many real-world applications the data in production does not necessarily come from the same distribution as the training data, resulting in a change in the distribution, which is also known as data drift, concept drift, covariate shift, etc. Data drift reduces the machine learning solution quality, which is especially important in critical real-world applications that rely on the accuracy and resiliency of the models [1], [2], [3], [4].

Data drift may occur gradually over time in use cases, such as banking security systems for credit-card fraud where the model of users' purchase behavior may change over time compared to the behavior the model was initially trained on. Data drift may also occur seasonally or suddenly. For example, many recommendation systems trained to make predictions prior to the COVID-19 global pandemic would abruptly fail after the onset of the virus due to the sudden changes in consumer behavior. [5], [6], [7].

Machine learning algorithms behave differently in terms of their sensitivity to data drift. The impact of data drift depends on the selected algorithms, as well as the application

domains. Application domains involving graphics and pattern recognitions, such as traffic toll systems, and image classification are less likely to suffer from data drift because the data used for training is static and expected to have the same distribution that will be seen in production. On the other hand, production data distribution can be very different from the training data, particularly the output of daily business processes that rely on natural language understanding for text classification involving user behavior.

1.2 Algorithms and Applications Differ in Data Drift Behavior

1.2.1 *Data Drift in Air Pollution Models*

In an initial study of air pollution detection models [8], we compared the sensitivity and robustness of data drift for the deep learning algorithms: Multiplayer Perceptron (MLP), Convolutional Neural Network (CNN), Gated Recurring Unit (GRU), and the Long Short-Term Memory (LSTM) network. The dataset was provided by [9], [10] and consisted of 43,824 air pollution samples with data features that measured the concentration of particle matter, air pressure, air temperature, and dew point. The base models for the MLP, CNN, GRU, and LSTM had near identical performance accuracy of more than 99% in predicting air pressure. When applying increasing levels of simulated drift (using colored noise) to the base models, the solution quality of the model deteriorated as expected, however, the deterioration was much less for the LSTM model. The algorithms based on RNN's (LSTM and GRU) turned out to be more robust to the dataset drift. The LSTM and GRU algorithms suffered a 2% and 4% loss respectively in accuracy as compared to the MLP, which performed the worst with a drop of 44%.

1.2.2 *Data Drift in Spam Filter Models*

As explained in [11] the term "spam" as it relates to email and text messages refers to unsolicited, often unwanted, email messages [11]. Spam costs businesses more than $20.5 billion every year [12] and is of particular concern to cybersecurity professionals since 94% of malware, fraud, and scams are often initiated through spam email and text messages [13]. Data drift is a challenge for spam filters because spammers continually add new data manipulation strategies to try to outwit spam filters, which results in changes in data distribution and population changes and a decline in the filter's ability to classify spam correctly.

The 2019 work of Dada et al. [14] provided a review of the application of popular machine learning approaches to email spam of the leading internet service providers, such as Gmail, Yahoo!, and Outlook. Their work focused on machine learning techniques used for email spam filtering of twenty published papers during the period of 2000–2018. For each paper, they provided the dataset description, machine learning techniques, algorithms used, performance metrics, and the limitations of each study. Of the twenty papers, the most popular algorithms studied included support vector machines (SVM), K-Nearest Neighbor (K-NN), Naïve Bayes (NB), and ensemble approaches that combined several algorithms. We find it interesting that only one paper used the Random Forest algorithm and only one paper employed the use of deep learning algorithms. In our work, we focus on the RF and deep learning algorithms.

1.3 Problem Statement

The datum used for many machine learning projects drifts over time. The resilience to dataset drift will affect the degree and ways the model suffers in production. An important aspect in the selection of the machine learning algorithm for a given real-world application is evaluating the inherent sensitivity and robustness to data drift, which is an underexplored area of research.

In this work, we provide a comparison of the performance change due to data drift for the machine learning subset of natural language understanding (NLU) used for text classification tasks, specifically spam filters. We compare a traditional Random Forest (RF) algorithm with the more advanced deep learning Convolutional Neural Network (CNN), and the Long Short-Term Memory Network (LSTM), as well as the cutting-edge pre-trained transformers architecture of the Bidirectional Encoder Representation from Transformer (BERT) developed by Google [15].

In our study, we seek to answer the question: “Which machine learning algorithms are more resilient to dataset drift for the application of text classification in spam filters?” We compare state of the art BERT pre-trained Transformer models with LSTM, CNN, and RF.

1.4 Solution Methods

The methodology for our study includes seven major steps:

1) Collect benchmark email datasets
2) Clean and preprocess data
3) Create hybrid datasets to train baseline models

4) Feature extraction (TF-IDF, Word Vector Representation, Transformers)
5) Create baseline models
6) Evaluate models using second datasets to simulate dataset drift
7) Compare and analyze results.

1.4.1 *Solution Validation.*

The solution is validated in terms of the calculated accuracy, recall, and precision scores. We compare performance of the baseline models using data from the same distribution with the performance of the models when evaluated with simulated data drift. When applying increasing levels of simulated drift to the base models, we expect deterioration across all models, leading us to find which models are more robust in the face of drift.

1.5 Contributions

We contribute to an underexplored area of research by providing performance analysis of the inherent sensitivity and, conversely, the robustness to data drift of four high-performing machine learning, deep learning, and transformer algorithms. We expect this work will assist with algorithm selection for natural language understanding applications that are subject to data drift.

1.6 Dissertation Roadmap

This dissertation is organized as follows: Chapter 2 provides a survey of relevant research and background on the well-documented data preprocessing techniques, machine learning algorithms, and deep learning algorithms compared in this study. Chapter 3 formally

describes our research and proposed methods. Chapter 4 discusses our experimental results and data collection. Chapter 5 provides our conclusion and our plans for future work.

Chapter 2

Review of the Literature and Background

2.1 Related Works in the Literature

Several works in the literature proposed machine learning algorithms for use in text classification for spam filtering [14], [16]. The 2019 work of Dada et al. [14] provided a review of the application of popular machine learning approaches to email spam by the leading internet service providers like Gmail, Yahoo!, and Outlook. As mentioned in our introductory chapter, their work focused on machine learning techniques used for email spam filtering of twenty published papers during the period of 2000–2018. Of the twenty papers, the most popular algorithms studied included SVM, K-NN, NB, and ensemble approaches that combined several algorithms. We find it interesting that only one paper used the RF algorithm, and only one paper employed the use of deep learning algorithms.

In [17], the authors provided a study of email classification research trends and open issues based on the review of ninety-eight articles that were published from 2006–2016. Among these ninety-eight articles, forty-nine were related to spam classification, and the remaining were dedicated to other email classifications, such as phishing, multi-folder categorization, and combinations of phishing and spam. Of the forty-nine spam classifications summarized by their work, SVM were the most frequently used machine learning algorithms, followed by Decision Trees, NB, and K-NN. We note of the forty-nine techniques identified, the RF was only explored in one of the studies.

Traditional machine learning classification methods have been extensively evaluated for use in text classification application [16], [18], [19], [20], [21]. However, missing from the twenty publications identified in [14] and the forty-nine publications dealing with spam classification identified in [17] is an exploration of how the models perform when faced with dataset drift (real concept drift, population drift, virtual drift, etc.).

The authors of related works [17], [14] concluded that more research is needed to study the application of deep learning for spam classification due to the automation of feature engineering that is inherent in deep learning layers. Both works also identified the need to address the problem of concept drift (one form of dataset drift) when building email spam filters. Our work directly addresses the open research problem for the need to employ deep learning for text classification in the application of spam filters.

The 2021 work of Minaee et. al. [22] provided a comprehensive review of more than 150 deep learning based text classification models. Their review provided a quantitative analysis of the performance of different deep learning models on popular benchmark datasets. We were encouraged to see the results we achieved in our work for the BERT-base model were competitive with the results of their BERT-base models in terms of accuracy. They provided results on NB, however, they did not compare the performance of BERT to that of the RF as we do in our work. Furthermore, they conducted performance analysis on classification tasks, such as sentiment analysis, news categorization, questions and answers, and natural language inference, however, they did not perform analysis of models for the application of spam filters as we do.

The authors of [23] performed a detailed analysis on the impact of different types of data drift available in the literature (sudden, reoccurring, gradual, and incremental). They designed a procedure to analyze and detect different forms of concept drift in the email domain. They developed an email concept drift analyzer tool. Their study concluded that concept drift should be considered when building any spam detection solution to ensure a lifelong spam email classification system. The focus of [23] was on the concept drift detection **s**trategies rather than on the inherent sensitivity and robustness of machine learning algorithms when no concept drift detection strategies have been applied.

2.2 Background on Natural Language Preprocessing Techniques

The author of [24] makes a clear distinction between data mining and text data mining, or text analytics, explaining that in the case of text data mining, patterns are extracted from unstructured natural language text, and in the case of data mining, data is extracted from structured databases containing facts. Natural language can be expressions in both written and spoken forms and requires various tools for the transformation, interpretation, and analysis of words, sentences, phrases, statements, advertisements, claims, etc.

As a prerequisite for the computer to extract intelligence from the words, it is necessary to first clean and preprocess the text, and transform unstructured natural language into a form that is more useful for the creation of features that will be used for training machine learning and deep learning models [25].

Preprocessing steps for text data mining involves procedures to help normalize the text and is dependent on the use case of the application. Normalizing the text helps to reduce noise

and define a “standard” of near-identical words such as “stopwords,” “stop-words,” and “stop words” to just “stopwords.” Python libraries such as SpaCy [26] and Natural Language Tool Kit (NLTK) [27] provide excellent NLP libraries to easily implement the text mining preprocessing tasks. We briefly explain well-documented NLP preprocessing techniques used in our study.

- **Tokenization**—Considered the first stage of the preprocessing pipeline, tokenization is a necessary step for all other stages. Tokenization is the process where each sentence in the text document is separated by breaking it into words or small fragments called tokens based on a pre-built model.
- **Sentence Segmentation**—Depending on the application, the text may also undergo sentence segmentation, which is a technique to split the input document into sentences that will be important when parsing multiple sentences to define parts of speech and dependent relations (noun, verb)
- **Stopword Removal**—This technique removes “non-relevant” words, with the goal of reducing the dimensionality down to the words that are supposed to add value to the model.
- **Lemmatizing**—This reduces a word to its lemma, or core representative word. To work properly, the lemmatizer requires the identification of the part of speech of the word.
- **Parts of Speech Tagging**—This tags words in the text with their part of speech, including verb, adjective, noun, etc.

Table 1 provides an example of tokenization, lemmatization, parts of speech tagging, and the identification of stopwords applied to the following sample corpus.

> *"This research was inspired by emerging technologies introduced during the DPS program at the Pace University. Pace's Pleasantville, NY campus is located near IBM. I am thankful for the outstanding professors who led our classes."*

Table 1 NLP Preprocessing: Tokenization, Lemmatization, Parts of Speech, and Stopwords

Tokenization	Lemmatization	Parts of Speech	Stopword
This	*this*	*DET*	*True*
research	*research*	*NOUN*	*False*
was	*be*	*AUX*	*True*
inspired	*inspire*	*VERB*	*False*
by	*by*	*ADP*	*True*
emerging	*emerge*	*VERB*	*False*
technologies	*technology*	*NOUN*	*False*
introduced	*introduce*	*VERB*	*False*
during	*during*	*ADP*	*True*
the	*the*	*DET*	*True*
DPS	*DPS*	PROPN	False
Program	*program*	NOUN	False
at	*at*	ADP	True
Pace	*Pace*	PROPN	False
University	*University*	PROPN	False
.	*.*	PUNCT	False
Pace	*Pace*	PROPN	False
's	*'s*	PART	True
Pleasantville	*Pleasantville*	PROPN	False
,	*,*	PUNCT	False
NY	*NY*	PROPN	False
campus	*campus*	NOUN	False
is	*be*	AUX	True
located	*locate*	VERB	False
near	*near*	SCONJ	False
IBM	*IBM*	PROPN	False
.	*.*	PUNCT	False

I	*-PRON-*	PRON	True
am	*be*	AUX	True
thankful	*thankful*	ADJ	False
for	*for*	ADP	True
the	*the*	DET	True
outstanding	*outstanding*	ADJ	False
professors	*professor*	NOUN	False
who	*who*	PRON	True
led	*lead*	VERB	False
our	*-PRON*	DET	True
classes	*class*	NOUN	False
.	.	PUNCT	False

Figure1 is an example of entity recognition from the sample corpus and Figure 2 is an example of dependency parsing.

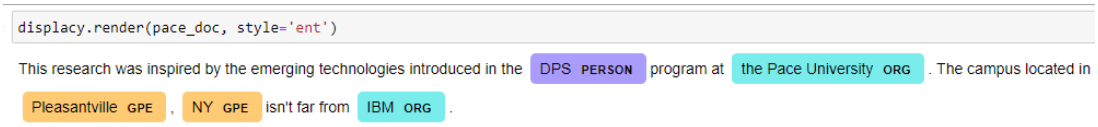

Figure 1 SpaCy Results for Named Entity Recognition on Sample Pace Corpus

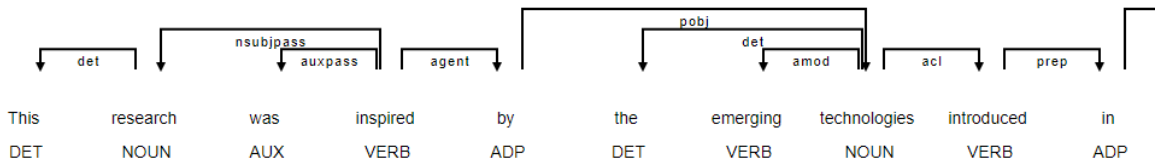

Figure 2 SpaCy Results—Dependency Parsing

2.3 Background on Text Feature Extraction Methods

Machine learning algorithms cannot process straight text as input, so it is necessary to represent the text numerically. Feature extraction is the process of numerical representation of the text. Common feature extraction techniques in the literature that we use in our study include bag of words (BOW), term frequency–inverse document frequency (TF-IDF), word embeddings, and the BERT language model.

2.3.1 *BOW and TFIDF*

BOWs and TF-IDF are simple techniques to numerically represent text documents [28] [29]. A BOW or multiset of words represents text in a way that describes the occurrence of words within a document based on the multiplicity of each word. Important preprocessing aimed at reducing the dimensionality of the corpus is normally conducted prior to the BOW processing. TF-IDF creates a vector representation of the document based on the one-hot-encoding schema, where words are given weight that measures relevance, not frequency. Word counts are replaced with TF-IDF scores across the whole dataset. TF-IDF combines two concepts, Term Frequency and Document Frequency. TF-IDF assumes there are several words that appear more often compared to others, and assesses the importance of words in each email, as well as the importance of each word in the document database. Inverse Document Frequency is the weight of a term; it looks to reduce the weight of a term if the term occurred throughout all the documents. TF-IDF is

a multiplication of the term's frequency matrix with its IDF and is calculated as shown in Equation 1.

BOW and TF-IDF discards word order, and therefore the context, meaning, and the relationship between words are lost, which is a major disadvantage of these methods.

$$w_{i,j} = tf_{i,j} \times \log(\frac{N}{df_i})$$

$tf_{i,j}$ = *number of occurrences of i* in *j*
df_i = $number\ of\ documents\ containing\ i$
N = Total number of documents

Equation 1 TF-IDF Formulation

Additionally, BOW and TF-IDF produces sparse representation vectors, resulting in large number of weights. Approaches to overcoming these challenges include pretrained word embedding, such as Word2Vec [30], Global Vectors for Word Representation (GloVe) [31], and Transformers (BERT) [15].

2.3.2 *Word Embeddings*

Word2Vec is an algorithm invented by Google researchers Mikolov, Sutskever, Chen, Corrado, and Dean [30] that uses a neural network model to learn word associations and create numerical representations known as word embeddings; these pretrained word vectors are based on millions of words. The numerical vector representation of the Word2Vec models captures the semantic similarity and analogous relationship between different words. The authors of [30] provided an example of vector space representation as shown in Figure 3, that depicts computation of $(king - man) + woman = queen$.

Figure 3 also demonstrates the vector space representation for verb tense (middle Figure) as well as geographical localities (Figure on right).

As explained in [32] [33] "Word embeddings are often used as the first data processing layer in a deep learning model". Machine learned embedding feature extraction techniques allow models to perform well without any external hand-designed resources or time-intensive feature engineering since no hand-crafted features are needed beforehand. Extensions of Google's Word2Vec methods are the GloVe developed by Stanford [31] as well as FastTex developed by Facebook [34].

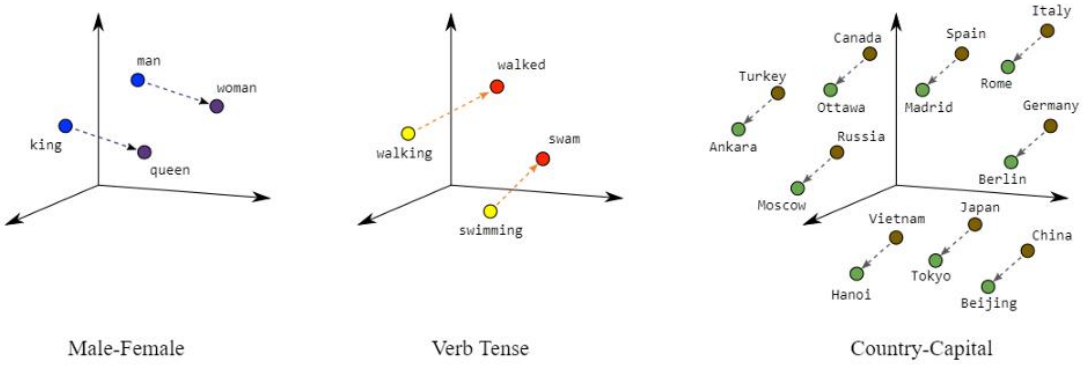

Figure 3 Example Word2Vec and GloVe Word Embedding. Image Source [35]

2.3.3 *BERT*

BERT [15], introduced in 2018 by researchers at Google, is a language model that builds on the foundational concepts of the Transformer architecture proposed in [36]. The BERT framework uses deep neural networks based on an encoder architecture with an attention mechanism designed to deal with the long-range dependency challenge suffered by CNN and RNN. As stated in [36], "the Transformer is the first transfer learning model relying

entirely on self-attention to compute representation of its input and output *without* using sequence-aligned RNNs or CNN."

BERT builds on the success of self-attention used in previous applications, such as reading comprehension, abstractive summarization, and learning task-independent sentence representations [36], [37], [38]. BERT provided a better understanding of the context of words within sentences. The "attention" mechanism determines the importance of words in relation to other words in a sentence. The authors in [39] explain, "Not all words contribute equally to the representation of the sentence meaning. Hence, we introduce attention mechanism to extract such words that are important to the meaning of the sentence and aggregate the representation of those informative words to form a sentence vector."

The original Transformer architecture detailed in [36] is both an encoder as well as a decoder. BERT, however, is only an encoder. For the attention mechanism, BERT uses three embeddings to represent the input representations: token, segment, and positional. BERT is trained using the Masked Language Model (Masked LM) and the Next Sentence Prediction. Directional models read the text input sequentially (left-to-right or right to left), however the BERT transformer encoder reads the entire input sequence of words at once using a bidirectional method. BERT is pretrained on an enormous corpus of unlabeled text, including the entire Wikipedia (more than 2,500 million words) and Book Corpus (more than 800 million words). We refer the interested reader to [15] for an extensive background on BERT technology.

2.4 Algorithms in Our Study

2.4.1 *Classical Machine Learning—Random Forest*

Classical machine learning models require hand-crafted features to construct a dataset to first train a model, then, as a second step, the model can make predictions based on the data it was trained on. Machine learning based methods automatically classify text based on observation of the data. Rule-based methods classify text into categories using a set of predefined rules and require extensive knowledge of the domain. In contrast, machine learning methods learn associations between text and their labels. The classical machine learning algorithm explored in our study is the RF.

One reason we selected the RF algorithm in this study is because it is an underrepresented algorithm in the surveys conducted in the related works discussed in Section 2.1 above.

The RF is a supervised meta learning algorithm based on decision tree algorithms. It was developed by Leo Breiman [41], a statistician at the University of California at Berkley. The RF algorithm grows many decisions trees, and each tree “votes” for one overall classification for the set of data; the RF then selects the classification having the most votes over all the trees in the forest (as shown in Figure). The RFs usually require shorter training time compared to that of SVMs and neural networks. RF is competitive with the existing machine learning algorithms in terms of accuracy. According to Breiman [41], the RF is unparalleled in accuracy among current algorithms, runs efficiently on large databases, and can handle thousands of input variables without variable deletion. The RF also gives estimates on which variables are important for the classification [3], [4].

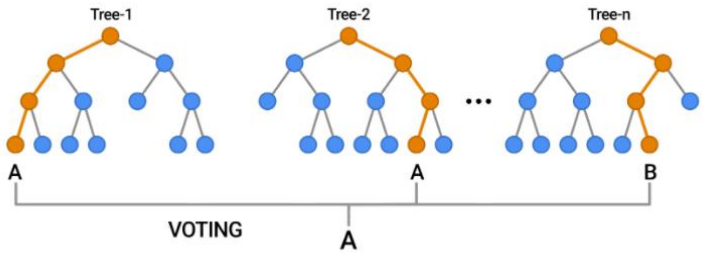

Figure 4 Example Random Forest - 3 Tress Vote. Source [39]

2.4.2 *Deep Learning—CNN, LSTM, BERT*

The ability of deep learning to process and learn from enormous quantities of unlabeled data gives it an advantage over classical/traditional machine learning algorithms. To be considered deep learning, the neural network must have a least three hidden layers. Modern neural networks have hundreds of hidden layers [42]. Feature extraction is a time-consuming task that can take teams of data scientists years to accomplish; deep learning performs automatic feature extraction without human intervention. During the feature learning stages, important features are extracted. Each layer of the model trains on a distinct set of features based on the output of the previous layers. The further you advance into the neural network, the nodes begin to accurately recognize more complex features, as they aggregate and recombine features from the previous layer. Neural networks are capable of handling very large, high-dimensional datasets with billions of parameters that pass through nonlinear (sigmoid or tahn) functions.

2.4.2.1 Convolutional Neural Networks (CNN)

CNNs were initially developed in the neural network image processing community where they achieved near humanlike results in recognizing objects from predefined categories [42], [43]. The first layer of a CNN is always the convolutional layer that uses filters (also

called neurons or kernels) to move (or convolve) through the input, and begins to identify the most significant features. CNNs do not learn from a single filter; they learn multiple filters in parallel for a given input. As explained in [44], It is common for a convolutional layer to learn from 32 to 512 filters in parallel for a given input, which gives the model 32, or even 512, different ways of learning and extracting features from the input data. Since the convolutional layers increase the dimensions of the input, the second layer, known as the pooling layer, reduces this dimensionality.

The architecture of CNNs, explained in [43], includes methods for dimensionality reduction through filters, stride, pooling, flattening, and the output (classification) as depicted in Figure 5. We refer the interested reader to [45], [42], [43] for expanded explanations of CNN architectures.

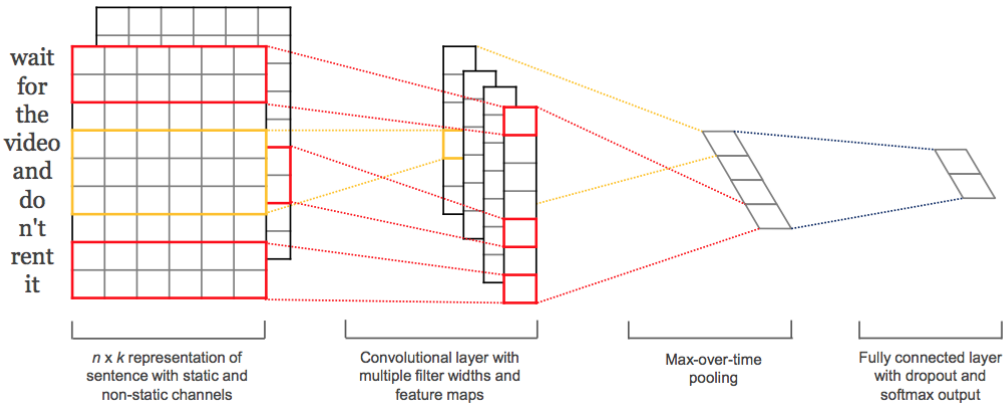

Figure 5 Simplified CNN Architecture. Source [46]

Applications where CNNs excel include image recognition, video recognition, image analysis, recommendation systems, natural language processing for small corpus data, such

as Twitter data, chat boxes, and sentiment analysis for business reviews. CNNs are designed to capture the most important information in a sentence.

Shortcomings of CNNs identified in the work of Kalchbrenner et al. [47] reported CNNs inability to (1) model long-distance contextual information and (2) preserve sequential order in their representations.

2.4.2.2 Recurrent Neural Network (RNN)

Recall that a feedforward neural network only considers the current example it is exposed to and does not consider order in time. As explained in [44], the architecture of RNN, on the other hand, "takes as their input not just the current input example they see, but also what they have perceived previously in time". A loop in the hidden layer allows data to be sent to the next layer [37], [42], [45]. In this way, an RNN is said to have "memory" of previous computations and uses the information to perform tasks that feedforward networks are not designed for. After producing the output, it is copied and sent back into the recurrent network using backpropagation through time of error and gradient descent to find correlations between events separated by many moments, which is helpful to classify sequential input. One shortfall of the basic RNN is that it suffers in maintaining the context of words when the length of the input becomes longer. This problem is helped by the use of LSTM networks, which are a special type of RNN.

2.4.2.3 Long Short-Term Memory (LSTM) Network

LSTMs can retain long-term dependencies and connect information from the past to the present. LSTM gates learn which information is important to keep and which is not. The

LSTM helps to overcome the vanishing gradient descent problems that are suffered by RNNs [42], [37]. LSTMs help preserve the error that can be back propagated through time, and allow recurrent nets to continue to learn over many time steps (more than 1,000), thereby opening a channel to link causes and effects remotely. LSTM networks are superior to the basic RNNs in applications based on time series, handwriting recognitions, semantic parsing, and robot control, among many others.

The authors of [48] provided interesting insights on the comparative performance between RNNs and CNNs. After testing on multiple NLP tasks that included sentiment classification, questions and answers, and parts of speech tagging, they concluded that there is no clear winner: the performance of each network depends on the global semantics required by the task itself.

2.4.2.4 BERT

BERT pretrained word embedding can be used for specific NLP application through a transfer learning process known as fine-tuning. The pretrained BERT model can be fine-tuned with just one additional output layer to create state of the art models for a wide range of NLP tasks. The output layer is created from models for specific tasks (classification, entity recognition, question answers, etc.) with application specific data to produce state of the art predictions [15]. The Transformer models have been shown to be superior on machine translation tasks and were more parallelizable, which requires significantly less time to train. The work of [36] proposed a simple network architecture, known as the Transformer, based solely on attention mechanisms that completely eliminated RNNs and CNNs. Self-attention allows models to look at other words in the input sequence to obtain

a better understanding of certain words in the sequence. Transformers were introduced to deal with the long-range dependency challenge and take advantage of massive amounts of training data.

2.4.2.5 BERT Use Cases

The October 2019 blog post by Google Fellow and Vice President of Search [40] introduced the news that BERT was being used to enhance Google's search engine optimizations (SEO). Google also uses BERT in its Google Translate applications that supports over 100 languages. Other use cases for BERT include sentiment analysis for reviews such as movie reviews as well as business reviews provided by services such as Yelp. BERT may also be used in question-answering (QA) applications such as Siri, OK Google, and chat boxes [15]

Chapter 3

Dataset Drift Sensitivity Research Methodology and Design

3.1 Dataset Shift: Formal Definitions

When training supervised machine learning models, a single large historical dataset is split into a train file and a test file, therefore the train and test files are said to have been generated from the same distribution. In real-world application, the data that the model is tasked with classifying rarely come from the same distribution it was trained on, resulting in a change in the distribution, which is also known as dataset drift. The problem of dataset drift causes the solution quality of the model to decline, making the model less accurate and, therefore, less reliable.

There are many names and definitions in the literature for dataset drift because the field lacks of standard terminology. As explained in the 2012 work of [49], “The literature on the topic is mostly scattered, and different authors used different names to refer to the same concepts or use the same name for different concepts.” This sentiment was shared by the authors of [50].

The most representative terms include: covariate shift [51] , prior probability shift [3], dataset drift [2], [52], and concept drift [50], [3], among many other names [50], [3].

The authors of [53] offer, “Dataset shift is a challenging situation where the joint distribution of inputs and outputs differs between the training and test stages. Dataset shift is when the training and test distributions are different, which potentially introduces unseen patterns and variation in the data.”

The authors of [2] defined dataset shift as "cases where the joint distribution of inputs and outputs differs between training and test stage." They provided background and definitions on various types of dataset shift as follows.

Definition 1: ***Dataset shift*** appears when training and test joint distributions are different. That is, when $P_{tra}(y, x) \neq P_{tst}(y, x)$

Definition 2: ***Covariate shift*** appears only in $X \longrightarrow Y$ problems, and is defined as the case where $P_{tra}(y|x) = P_{tst}(y|x)$ and $P_{tra}(x) \neq P_{tst}(x)$

Covariate shift is when only the distributions of covariates (shift in the input or features) x change and everything else is the same. Covariate shift means that only the input distribution changes, whereas the conditional distribution of the outputs given the inputs p(y|x) remains unchanged.

Definition 3: ***Prior probability shift*** appears only in $Y \longrightarrow X$ problems, and is defined as the case where $P_{tra}(x|y) = P_{tst}(x|y)$ and $P_{tra}(y) \neq P_{tst}(y)$

Prior probability shift is when only the distribution over y changes and everything else stays the same.

Definition 4: ***Concept shift*** is defined as

- $P_{tra}(y|x) \neq P_{tst}(y|x)$ and $P_{tra}(x) = P_{tst}(x)$ in $X \longrightarrow Y$ problems
- $P_{tra}(x|y) \neq P_{tst}(x|y)$ and $P_{tra}(y) = P_{tst}(y)$ in $Y \longrightarrow X$ problems

Concept shift, also called concept drift, is not related to the data distribution or the class distribution, but instead is related to the relationship between the two variables. With

concept drift, the interpretation of data changes over time even though the distribution of the data does not.

Definition 5: ***Sample selection bias***, in general, causes the data in the training set to follow $P_{tra} = P(s = 1|x, y)$, while the data in the test set follows $P_{tst} = P(y, x)$. Depending on the type of problem, we have:

$P_{tra} = P(s = 1|x, y)P(y|x)P(x)$ and $P_{tst} = P(y|x)P(x)$ in $X \longrightarrow Y$ problems.

$P_{tra} = P(s = 1|x, y)P(x|y)P(y)$ and $P_{tst} = P(x|y)P(y)$ in $Y \longrightarrow X$ problems.

where s is a binary selection variable that decides whether a datum is included in the training sample process $(s = 1)$ or rejected from it $(s = 0)$.

Sample selection bias is when the distributions differ because of an unknown sample rejection process.

In our study, our focus in on dataset shift, and we use data drift and data shift interchangeably. In previous works, much of the focus has been on detection and adaptation strategies for dataset drift [3], [50], [52], [54], however, the algorithms inherent response to dataset drift (before incorporating detection or mitigation methods) has received less attention and is the focus of our research.

The problem of dataset shift can be caused from the way input features are constructed, the sparsity of data, shifts in the data distribution due to non-stationary environments, and from changes in the activation patterns within layers of deep neural networks [2]. One problem with dataset shift research is the lack of real-life datasets that contain dataset shift. This problem is expressed by the authors of [55] who point out, “Even though it is natural to

observe shift in several real-life datasets, unfortunately, existence of clear dataset shift is rarely found in the publicly available real-life datasets.

In our work, we combine two real-world datasets to create different distribution to simulate dataset shift for experimentation with the sensitivity and robustness of several machine learning algorithms.

3.2 Examples of Data Drift in Real World Application

Popular real-life use-cases where data changes over time are prevalent in industries such as finance and banking, retail and ecommerce, and healthcare, among others [50]. Specific applications may involve monitoring and control, information management, analytics, and diagnostics. Examples may appear in models trained on consumer habits in ecommerce, predictions for the stock exchange, advertising targets, and medical diagnosis, among many others.

Data drift may occur gradually over time in use cases such as banking security systems for credit card fraud where the model of a user's purchase behavior may change compared to the behavior on which the model was initially trained. Data drift may also occur seasonally and/or suddenly. For example, many recommendation systems trained to make predictions prior to the COVID-19 global pandemic would abruptly fail after the onset of the virus due to the sudden changes in consumer behavior [5], [6], [7].

Some types of dataset drift are related to the deliberate and clandestine adversarial attempts to circumvent systems as in the case of fraud or other undesirable activities, such as

malware and spam. As such, spam filters are a real-world application that may suffer from dataset drift and the use case application focus of our study.

3.3 Existing Measures to Mitigate Data Drift

Existing measures for drift mitigation include drift detection and adaptation measures. The works of Lu et al. [56] reviewed 130 publications in concept drift related research and surveys of the most popular concept drift detection and adaptation methods and algorithms. They concluded that more research on effectively integrating concept drift handling techniques with machine learning methodologies is highly desired.

In [57], the authors provided a comprehensive survey of the most representative detection learning measures that they categorize into three groups.

1. *Sequential Analysis* based methods include the Cumulative Sum (CUSUM) and its variant Page Hinkley (PH) [58].

2. *Statistical based* approaches include the Drift Detection Method (DDM) [59], Early Drift Detection Method (EDDM) [60], Exponentially Weighted Moving Average (EWMA) [61], and Reactive Drift Detection Method (RDDM) [62], among others.

3. *Windows* based methods include the Adaptive Windowing (ADWIN) [63], [64], the SeqDrift detector [65], the Drift Detection Methods based on Hoeffding's Bound (HDDMA-test and HDDMW-test) [66], and Fast Hoeffding Drift Detection Method (FHDDM) [67], among others.

3.4 Addressing Gaps in Data Drift Research

Our research helps fill the gap suggested in [56] to integrate concept drift handling techniques with machine learning methodologies. The authors of [14] concluded that further research is needed to tackle the issue of email spam filtering as a concept drift problem, and listed the "inability of current spam filtering techniques to effectively deal with the concept drift phenomenon" as open research problem. Our work helps to address this research gap by using spam filters as our application of study for robustness of data drift.

The existing measures to mitigate dataset drift explored various drift detection and adaptation methods as mentioned in section 3.3. To our knowledge, there is no study that compares the inherent robustness of algorithms before any detection or mitigating methods have been applied. Our study will help fill the research gap by comparing the inherent robustness of the underexplored RF, as well as the high-performing CNN, LSTM, and BERT models.

3.5 Research Question

In our study, we seek to answer the question, "Which machine learning algorithms are more resilient to dataset drift for the application of text classification in spam filters?" We provide a performance comparison of the state of the art BERT pretrained Transformer models with the popular and high performing LSTM, CNN, and RF algorithms.

3.6 Solution Methodology

The methodology for our study of the robustness and sensitivity of the inherent nature of selected algorithms to dataset drift for the application of spam filters was briefly introduced in section 1.6. We provide details of each step in in sections 3.6.1 through 3.6.7, as well as an overall graphical research design in Figure 5. The methodology includes seven major steps.

1) Collect benchmark email datasets
2) Create hybrid datasets to train baseline models
3) Clean and preprocess emails
4) Extract features (TF-IDF, Word Embeddings, Transformers)
5) Create baseline models
6) Evaluate models using second datasets to simulate dataset drift

3.6.1 *Collection of Benchmark Email Datasets*

The benchmark email datasets used in our study were obtained from two sources. This allowed us to train from one distribution and test from a different distribution to simulate dataset distribution change. One benchmark dataset was obtained from the SpamAssassin public mail corpus [68] . The second dataset was from the Enron emails introduced in the work of Metsis, Androutsopoulos, and Paliouras [18] and obtained from [69].

3.6.2 *Create Hybrid Datasets*

To establish baseline performance with no drift present, we created models using datasets from the same distributions. We created four hybrid datasets; each contains all 3,051 emails

from the SpamAssassin's dataset. We then added specific amounts of emails from the Enron dataset to the SpamAssassain to create the hybrid datasets. The amounts added consisted of 40%, 30%, 20%, and 10% of the Enron dataset. Table 2 provides the number of emails in each of the original datasets. Table 3 provides the details of the hybrid datasets. Models trained with the hybrid datasets will be trained from the same distributions in varying amounts.

Table 2 Email Composition of Benchmark Datasets

Source	Total	Legitimate (Ham) Emails	Unwanted Spam Emails
Spam Assassin (SA)	3042	2545	497
Enron Total (E)	5171	3672	1499

Table 3 Email Composition of Hybrid Datasets

Spam Assassin Combined with Percentage of Enron	No. of Emails Added	Ham Emails Added	Spam Emails Added	Total Hybrid Ham	Total Hybrid Spam	Total Hybrid Data set
10%	**517**	259	258	2804	755	**3559**
20%	**1034**	517	517	3062	1014	**4076**
30%	**1551**	775	776	3320	1273	**4593**
40%	**2068**	1034	1034	3579	1531	**5110**

3.6.3 *Clean and Preprocess Emails*

Data preprocessing, also referred to as data wrangling, is a crucial step in the machine learning process, as it helps to transforms the data into a format that is understandable by the machine learning models. Many reports estimate that up to 60–80% of time and effort

is invested in the data preprocessing tasks of collecting, loading, cleaning, and preparing the data [70], [45] [71]. Section 2.2. provided a brief review of common NLP preprocessing techniques. In this section, we expand on the methods and tools that were used to prepare the data for our study.

Many of the NLP tasks were completed using the open-source library Natural Language Toolkit (NLTK). Python's email package provides a library for managing email messages [72] that uses Multipurpose Internet Mail Extensions (MIME) formatting. This library was used to help load and parse emails as a first step in preparing the dataset. Other NLP packages and tools used to prepare the text include Beautiful Soup [73], Regex [74], [75], UrlExtract [76], and Spacy [77]. The tutorial to implement the data cleaning was provided by [78]. Beautiful Soup is a Python library for pulling data out of HTML and XML files. We used Beautiful Soup to parse and clean the emails by removing the HTML tags, email headers, and non-Latin symbols from the emails. The Regex (Regular Expression) Python package was used to remove punctuation and the Python class UrlExtract were used to extract urls from the emails.

The general structure of an email is composed of two mail elements: the body and the header. The Simple Mail Transfer Protocol (SMTP) defines the email header section, which contains fields in the header that identify the subject, senders name, email ID, sending data, routing information, timestamp, recipient information, success of delivery, etc. To create the datasets, none of the header information is needed and therefore must be parsed, identified, and removed. Figure 6 is an example of a "raw" email before cleaning.

```
From exmh-workers-admin@redhat.com  Thu Aug 22 12:36:23 2002
Return-Path: <exmh-workers-admin@example.com>
Delivered-To: zzzz@localhost.netnoteinc.com
Received: from localhost (localhost [127.0.0.1])
          by phobos.labs.netnoteinc.com (Postfix) with ESMTP id D03E543C36
          for <zzzz@localhost>; Thu, 22 Aug 2002 07:36:16 -0400 (EDT)
Received: from phobos [127.0.0.1]
          by localhost with IMAP (fetchmail-5.9.0)
          for zzzz@localhost (single-drop); Thu, 22 Aug 2002 12:36:16 +0100 (IST)
Received: from listman.example.com (localhost.localdomain [127.0.0.1]) by
   listman.redhat.com (Postfix) with ESMTP id 8386540858; Thu, 22 Aug 2002
   07:35:02 -0400 (EDT)
Delivered-To: exmh-workers@listman.example.com
Received: from munnari.OZ.AU (localhost [127.0.0.1]) by delta.cs.mu.OZ.AU
   (8.11.6/8.11.6) with ESMTP id g7MBQPW13260; Thu, 22 Aug 2002 18:26:25
   +0700 (ICT)
From: Robert Elz <kre@munnari.OZ.AU>
To: Chris Garrigues <cwg-dated-1030377287.06fa6d@DeepEddy.Com>
Cc: exmh-workers@example.com
Subject: Re: New Sequences Window
In-Reply-To: <1029945287.4797.TMDA@deepeddy.vircio.com>
References: <1029945287.4797.TMDA@deepeddy.vircio.com>
   <1029882468.3116.TMDA@deepeddy.vircio.com> <9627.1029933001@munnari.OZ.AU>
   <1029943066.26919.TMDA@deepeddy.vircio.com>
   <1029944441.398.TMDA@deepeddy.vircio.com>
MIME-Version: 1.0
Content-Type: text/plain; charset=us-ascii
Message-Id: <13258.1030015585@munnari.OZ.AU>
X-Loop: exmh-workers@example.com
Sender: exmh-workers-admin@example.com
Errors-To: exmh-workers-admin@example.com
X-Beenthere: exmh-workers@example.com
X-Mailman-Version: 2.0.1
Precedence: bulk
List-Help: <mailto:exmh-workers-request@example.com?subject=help>
List-Post: <mailto:exmh-workers@example.com>
List-Subscribe: <https://listman.example.com/mailman/listinfo/exmh-workers>,
   <mailto:exmh-workers-request@redhat.com?subject=subscribe>
List-Id: Discussion list for EXMH developers <exmh-workers.example.com>
List-Unsubscribe: <https://listman.example.com/mailman/listinfo/exmh-workers>,
   <mailto:exmh-workers-request@redhat.com?subject=unsubscribe>
List-Archive: <https://listman.example.com/mailman/private/exmh-workers/>
Date: Thu, 22 Aug 2002 18:26:25 +0700
   Date:       Wed, 21 Aug 2002 10:54:46 -0500
   From:       Chris Garrigues <cwg-dated-1030377287.06fa6d@DeepEddy.Com>
   Message-ID:  <1029945287.4797.TMDA@deepeddy.vircio.com>
 | I can't reproduce this error. For me it is very repeatable
```

Figure 6 Raw Email Structure before Preprocessing. Source of Email: [68]

Figure 7 shows the email after the header information has been parsed and removed using Regex, Beautiful Soup, and SpaCy.

Date: Wed 21 Aug 2002 10:54:46 0500

From: Chris Garrigues wgdated1030377287.06fa6d@DeepEddy.Com

MessageID: 1029945287.4797.TMDA@deepeddy.vircio.com

| I can't reproduce this error. For me it is very repeatable like every time without fail. ….

Figure 7 Email after Header Removed

One important data cleaning task is the removal of stopwords. Stopwords are a set of commonly used words in a language. Stopwords in machine learning are typically considered to be unimportant in the modeling of features for the dataset; some examples include “a,” “the,” “is,” “are,” “to,” “in,” etc. We created hybrid datasets with and without stopwords for performance comparison of our models.

We show our preprocessing implementation details in Chapter 4.

3.6.4 *Feature Extraction*

For our traditional RF model, we extract the data features using the TF-IDF method discussed in section 2.3.1. For the CNN and LSTM, we use the Tokenizer method in the Keras library to preprocess and extract features for use by the embedding layers of the deep learning networks. For the BERT models, we use the specialized BERT Tokenizer and BERT embedding to extract features.

3.6.5 *Creating Baseline Models*

We use the four datasets as detailed in section 3.6.2 to establish the baseline performance using data from the same distribution. We build baseline models based on the RF, CNN, LSTM, and BERT algorithms.

3.6.6 *Evaluate Model Performance in the Face of Data Drift*

We evaluate the performance of all baseline models using datasets from different distributions. Our study will show which algorithms are more robust to the data drift.

3.7 Evaluation Metrics

We refer to the confusion matrix (also known as error matrix) to describe the performance of our classification models [45], [79]. The most common metrics calculations include accuracy, recall, precision, and F1, among others [45], [79].

Table 4 Confusion Matrix General Structure

		Model Prediction	
		Negative	**Positive**
Actual Ground Truth	**Negative**	True Negative	False Positive
	Positive	False Negative	True Positive

- Accuracy is the percentage of correctly classified instances out of all instances.

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

- The precision metrics calculates the accuracy for the minority class. It is calculated as the ratio of correctly positive examples divided by the total number of positive instances.

$$Precision = \frac{\sum True\ Positive}{\sum True\ Positive \quad + \quad \sum False\ Positive}$$

- The recall metric is the ratio tp / (tp + fn) where tp is the number of true positives and fn the number of false negatives. The recall is intuitively the ability of the classifier to find all the positive samples. Recall is defined as follows.

$$Recall = \frac{\sum True\ Positive}{\sum True\ Positive \quad + \quad \sum False\ Negative}$$

3.8 Expected Achievement

The goal of this exploratory study is to determine which machine learning and deep learning models are more sensitive to data that changes over time. The traditional machine algorithm represented in the study is the RF, the deep learning algorithms include CNN, LSTM, and BERT. We expect the accuracy from the dataset drift evaluation will be higher on the dataset constructed from 40% of the Enron Dataset, and will decline in performance from the 10% Enron augmented dataset. This is expected because the training and testing ratio of 100:40 will be more stable than the training and testing ration of 100:10.

3.9 Assumptions

We assume the emails from these two diverse populations are from different distributions, as they were collected at different times from different sources. We expect diverse content within each dataset.

3.10 Experimental Solution Design and Workflow

The experimental solution design workflow is shown in Fig 8.

Chapter 4

Experiment Design and Findings

In this chapter we implement the experiments to research the stability of four popular high-performing machine learning algorithms for the text-classification task of email spam classification. We compared the resilience to dataset drift of RF, CNN, Long LSTM, as well as BERT. The experimental solution design workflow is shown in Fig 8.

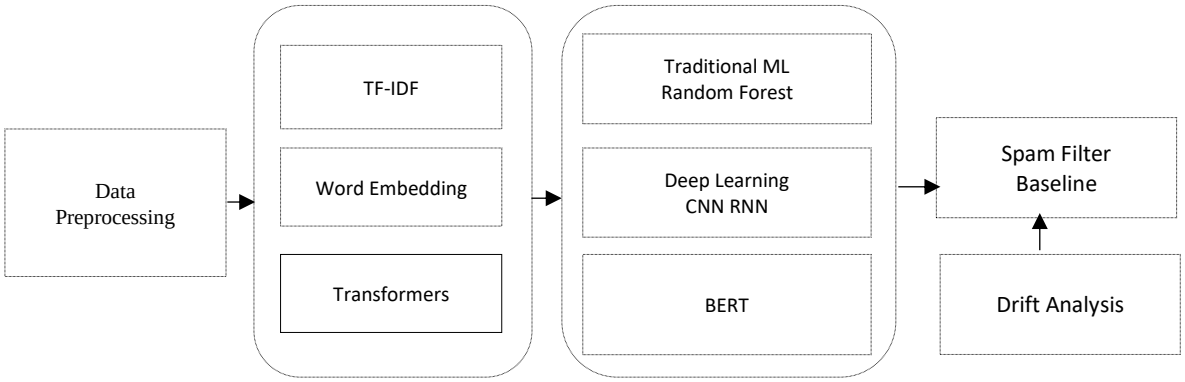

Figure 8 Simplified Research Design. Source: Adapted from [123]

4.1 Data Preprocessing Implementation

To create the datasets, the email header information is not needed and therefore must be parsed, identified, and removed. We implemented the NLP tasks using the open-source library Natural Language Toolkit (NLTK), Python's email package, *email* [72], Beautiful Soup [73], Regex [74], [75], UrlExtract [76] and SpaCy [77]. Beautiful Soup is a Python library for pulling data out of HTML and XML file. We used Beautiful Soup to parse and clean the emails by removing the HTML tags, email headers, and non-Latin symbols from the emails. The Regex (Regular Expression) Python package was used to remove

punctuation and the Python class UrlExtract was used to extract urls from the emails. Table 5 provides a pseudo code of the steps taken to parse the email headers and to clean and prepare the emails for feature extraction.

Table 5 Email Preprocessing

Pseudocode of email preprocessing
Input:
Bulk Emails
Output
Cleaned and Parsed Emails Prepared for Feature Extraction
BEGIN
#Useing Python *email* package
Load & parse emails in the given path
Create lists of all parsed HAM & SPAM emails
#Useing Numpy
Create array of all the email objects
Create an array containing binary labels in the same order as x (0 for ham emails & 1 for spam emails)
Create a test set & training set using scikit-learn
#Using Beautiful Soup
Create a BS object containing email object contents
Return extracted plain text from the HTML soup - remove newlines & spacing,
Convert HTML to plaintext
#Using urlextract
strip headers
Convert all URLs present in the plain text to the word ' URL '
Use Regex
remove punctuation
Create an array with the cleaned email
Lower case all words
#Using Spacy
Transform Text
tokenize
Lemmatize
Remove stopwords
End

4.1.1 *Stopwords Analysis*

Our initial analysis of the training dataset captures the top twenty words observed before stopwords are removed, as shown in Figure 13. We can see that because of tokenizing and stemming, the apostrophe (’) occurred nearly 80,000 times, “the” and “to” each occurred more than 10,000 times. The text analysis captured in Figure 13, offers limited insights of the words that matter in the non-spam (ham) emails because of the domination of the stopwords. On the other hand, Figure 14 and Figure 15 provide more insight into the ham and spam words respectively after the stopwords are removed from the dataset. Figure 16 is an example of emails with stopwords and punctuation included, and Figure 17 is an example of the same emails after the removal of stopwords and punctuation

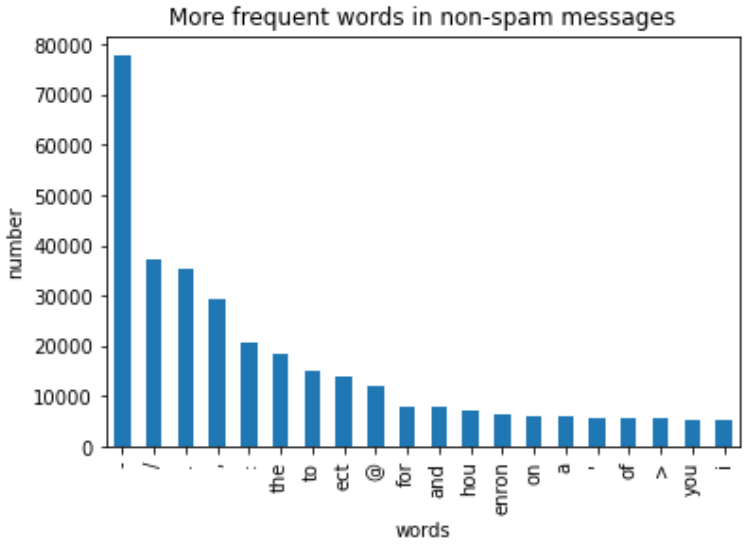

Figure 9 Most Frequent Non-Spam Words—Stopwords Included

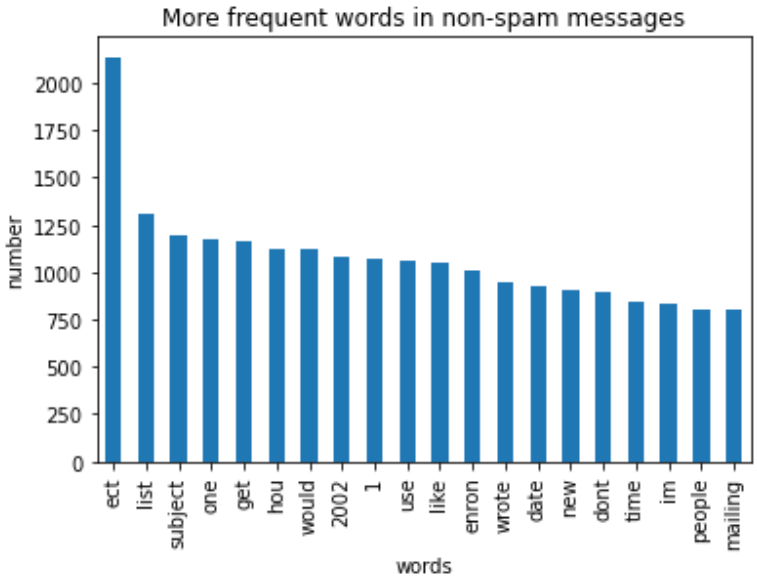

Figure 10 Most Frequent Non-Spam Words—Stopwords Removed

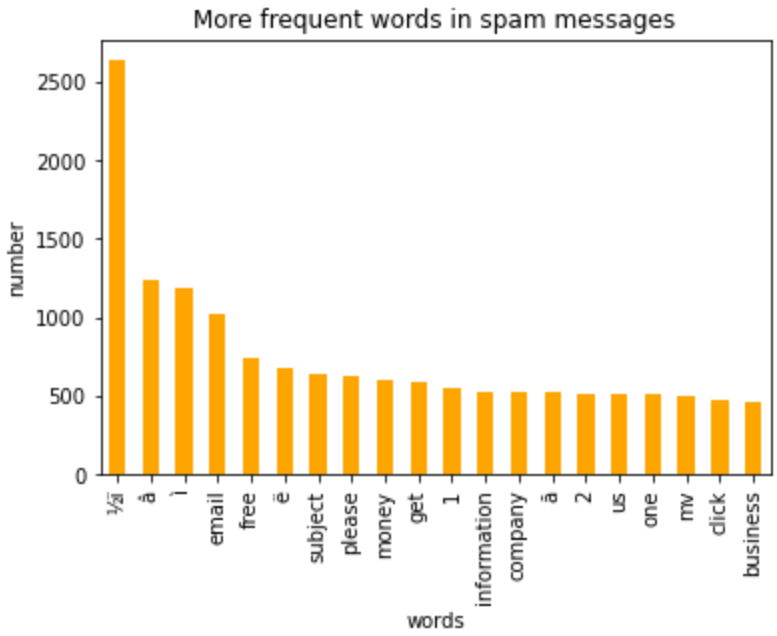

Figure 11 Most Frequent Spam Words—Stopwords Removed

```
<bound method NDFrame.head of      label
0       ham  If you haven't already you should enable the d...
1       ham  On Mon 23 Sep 2002 Russell Turpin wrote:  Here...
2       ham  use Perl Daily Headline Mailer  New Perl Monge...
3       ham  Hello Chris  Oh I don't know Timelag synchroci...
4       ham  use Perl Daily Headline Mailer  Announcing Sou...
...     ...                                                ...
4037   spam  Subject: pictures\r\nstreamlined denizen ajar ...
4038   spam  Subject: penny stocks are about timing\r\nnoma...
4039   spam  Subject: anomaly boys from 3881\r\nuosda apapr...
4040   spam  Subject: slutty milf wants to meet you\r\ntake...
4041   spam  Subject: important online banking alert\r\ndea...
```

Figure 12 Emails before Cleaning

```
<bound method NDFrame.head of      label
0       ham   havent  already  enable  debug  log  hacking ...
1       ham   mon 23 sep 2002 russell turpin wrote heres be...
2       ham   use perl daily headline mailer new perl monge...
3       ham   hello chris oh dont know timelag synchrocity ...
4       ham   use perl daily headline mailer announcing sou...
...     ...                                                ...
4037  spam   subject pictures streamlined denizen ajar cha...
4038  spam   subject penny stocks timing nomad internation...
4039  spam   subject anomaly boys 3881 uosda apaproved mle...
4040  spam   subject slutty milf wants meet take ilaa liqa...
4041  spam   subject important online banking alert dear v...

[4042 rows x 2 columns]>
```

Figure 13 Emails after Cleaning

4.2 Model Hyperparameter Selections and Implementations

When creating machine learning models, the optimal parameters that define the model's architecture may not be apparent, and thus requires exploration of a range of possibilities. Exploring the set of optimal parameters is known as hyperparameter tuning. The impact of algorithm parameter tuning has been studied in a number of works [80], [81], [82], [83]. The authors of [83] concluded: ". . .tuning hyperparameters is often more important than the choice of the machine learning algorithm".

In section 4.2.1 we provide the Random Forest hyperparameter tuning results and implementation, section 4.2.2 provides the hyperparameter tuning and implementation for the CNN, section 4.2.3 details the hyperparameter tuning and model implementation for the LSTM and section 4.2.4 provides the hyperparameter tuning and implementation for the BERT models.

4.2.1 *Random Forest Hyperparameter Tuning and Model Experiment Design*

The authors in [81] provided a literature review and experimentation of the influence of various parameters on the prediction performance of the RF. They concluded that the number of trees (mtry in the R language and n_estimators in scikit implemented in Python) is the most influential, and although sample size and node size have a minor influence on performance, they should still be tuned for maximum performance. Using the Grid Search with Cross Validation (*GridSearchCV)* from the scikit-learn library, the optimal hyperparameter values were found and used to create the RF baseline models. We found the optimal parameters for the RF models for our dataset to be: 100 trees, max features value of 50, max depth of 1, the gini index selected, and the bootstrap set to false. Hyperparameter tuning range tested and optimal values selected are presented in Table 6.

Table 6 Random Forest Hyperparameter Tuning

Hyperparameter	Description	Range Tested	Optimal Value
n_estimators	Number of trees in random forest	[75, 100, 200, 200]	100
max_features	max number of features considered for splitting a node	[25, 50, 100]	50
max_depth	max number of levels in each decision tree	1 for Classification 5 for Regression	1
criterion	optimum split of the features.	gini, or entropy	gini
bootstrap	if *True*- controlled with the max_samples parameter bootstrap if *False* the whole dataset is used to build each tree.	True or False	False

We created the RF based on the basic pseudo code presented in the works of [84], [85], [14]. The basic pseudo code is presented in Table 7. Table 8 details the implementation of

the code using a Jupyter notebook running on a commodity laptop with 8 GB RAM, 294 GB hard drive, using the Windows 10 OS.

Table 7 Random Forest Pseudo Code Source: Adapted from [86]

```
TF-IDF
Precondition: A training set S = ⟨x_1, y_1⟩, ⋯, ⟨x_n, y_n⟩, features F,
and number of trees in the forest B.
function RandomForest(K, L)
        H ← ∅
        For i ∈ {1, ⋯, B} do
            K^(i) ← A bootstrap sample from S
            h_i ← RandomizedTreeLearn(K^(i), L)
            H ← H ∪ {h_i}
        end for
        return H
end function
function RandomizedTreeLearn(K, L)
        At each node:
         f ←very small subset of L
         Split on the best feature in f
         Return the learned tree
end function
```

Table 8 Random Forest Model Implementation

Random Forest Model Implementation
Import required libraries and packages
from keras.models import Sequential
From sklearn.feature_extraction.text import TfidfVectorizer from sklearn.ensemble import RandomForestClassifier from sklearn.metrics import accuracy_score, classification_report, confusion_matrix from sklearn.pipeline import Pipeline **Vectorize the Text** tfidf = TfidfVectorizer() X = tfidf.fit_transform(data['text']) **Define Model Pararmeter** classifier = Pipeline([('tfidf', TfidfVectorizer()), ('classifier',RandomForestClassifier(n_estimators=100, criterion='gini',max_depth=None, min_samples_split=2, min_samples_leaf=1, min_weight_fraction_leaf=0.0,

```
max_features='auto', max_leaf_nodes=None,
min_impurity_decrease=0.0, min_impurity_split=None,
bootstrap=False, oob_score=False, n_jobs=None,
random_state=None, verbose=0, warm_start=False,
class_weight=None, ccp_alpha=0.0, max_samples=None))])
classifier.fit(X_train, y_train)

Predict the Results
y_pred = classifier.predict(X_test)
confusion_matrix(y_test, y_pred)
print(classification_report(y_test, y_pred))
```

4.2.2 *CNN Hyperparameter Tuning and Model Experiment Design*

The authors in [87] and [80] provided the hyperparameters used to build their CNN models for classification on NLP tasks. They provided the values tuned for the learning rate, hidden layers, batch size, sentence length, and filter size. We included the hyperparameters from both works for tuning our models using the grid search with cross validation (*GridSearchCV)* from the scikit-learn library. We present the ranges tested and the optimal values used for our CNN model in Table 9.

Table 9 CNN Hyperparameter Tuning

Hyperparameter	Description	Range Tested	Optimal Value Used
number of filters		[32, 64, 128]	32
batch size	Number of samples taken for each epoch	[10, 20, 32, 64, 128	64
epochs	Number of times the entire training set passed through the network	10, 20, 30	10
kernel size	Dimension of the sliding window over the input	[3, 5, 7]	7
learning rate	Controls how much to update the weight in the optimization algorithms	[0.001, 0.01, 0.1, 0.2]	0.01

optimizer	Controls how well the hidden layers learn the dataset	[sdg, adam, rmsprop]	adam

The CNN models contain 5 layers: the embedding layer, the convolution layer, the pooling layer, and two dense layers. Table 10 provides details of the CNN model construction. The CNN models were implemented using a Jupyter notebook provided by the cloud-based Google Colab using 16 GB GPUs provided with the Colab Pro subscription.

Table 10 CNN Model Implementation

CNN Neural Network Basic Algorithm

Import required libraries and packages

```
from keras.models import Sequential
from keras import layers
from sklearn.model_selection import train_test_split
import pandas as pd
from keras.preprocessing.text import Tokenizer
from keras.preprocessing.sequence import pad_sequences
import numpy as np
from keras.wrappers.scikit_learn import KerasClassifier
```

Preprocess Ham and Spam Hybrid Email Datasets (Table 3)

-Number of Samples as shown in Table 3 70% for Training and 30% for Test (Baseline Models)

Load Cleaned Email Hybrid Datasets and Create Training and Test Sets:

```
X_train, y_train X, text, Y_test
df = pd.read_csv
(r'C:..//PaceResearch//Dissertation//Spam//Experiments//Datasets//St
opwords_Removed_spam_assassainEmails_cleaned_3042_augmented_1034_20_
percent_enron.csv',encoding='ISO-8859-1')
df = df.dropna()
df['label'] = df['label'].apply(lambda x: 1 if x == 'ham' else 0)
x_train,  x_test,  y_train,  y_test  =  train_test_split(df['text'],
df['label'], test_size = 0.3, random_state = 0)
```

Convert sentences to sequences

```
maxlen = 100
embedding_dim = 50
tokenizer = Tokenizer(num_words=5000)
tokenizer.fit_on_texts(x_train)
X_train = tokenizer.texts_to_sequences(x_train)
X_test = tokenizer.texts_to_sequences(x_test)
vocab_size = len(tokenizer.word_index) + 1
```

```
X_train = pad_sequences(X_train, padding='post', maxlen=maxlen)
X_test = pad_sequences(X_test, padding='post', maxlen=maxlen)
```

CNN Architecture

```
vocab_size=44642
model = Sequential()
model.add(layers.Embedding(vocab_size, embedding_dim,
input_length=maxlen))
model.add(layers.Conv1D(128, 5, activation='relu'))
model.add(layers.GlobalMaxPooling1D())
model.add(layers.Dense(10, activation='relu'))
model.add(layers.Dense(1, activation='sigmoid'))
model.compile(optimizer='adam',
              loss='binary_crossentropy',
              metrics=METRICS)
model.summary()
history = model.fit(X_train, y_train,
                    epochs=10,
                    verbose=True,
                    validation_data=(X_test, y_test),
                    batch_size=64)
loss, tp,fp,tn,fn,accuracy, precision,recall, auc =
model.evaluate(X_train, y_train, verbose=True)
```

4.2.3 *LSTM Hyperparameter Tuning and Model Experiment Design*

The authors in [87], [80], [88] provided the learning rate, hidden layers, batch size, sentence length, and the filter size hyperparameters used to build their LSTM models for classification on NLP tasks. We included the hyperparameters from their works within our grid search, which helped to provide the optimal hyperparameters for our datasets. We present the ranges tested and the optimal values used for our tasks in Table 11.

Table 11 LSTM Hyperparameter Tuning

Hyperparameter	Description	Range Tested	Optimal Value
batch size	Number of samples taken for each epoch	[8, 16, 32, 64, 128]	64

epochs	Number of times the entire training set passed through the network	5, 10, 20, 30	30
learning rate	Controls how much to update the weight in the optimization algorithms	[0.001, 0.01, 0.1, 0.2]	0.02
dropout rate	Helps to avoid overfitting	[0.2, 0.25, 0.3, 0.4]	0.4
Optimizer	Controls how well the hidden layers learn the dataset	[sdg, adam, msprop]	rmsprop

The LSTM models contain four layers: the embedding layer, the LSTM layer, the pooling layer, and the dense layer. Table 12 provides details of the LSTM model implementation. The LSTMs were implemented using a Jupyter notebook provided by the cloud-based Google Colab using 16GB GPUs provided with the Colab Pro subscription.

Table 12 LSTM Experiment Model Design Implementation

LSTM Model Implementation
Import Libraries
import tensorflow as tf
import keras
from tensorflow.keras.preprocessing.text import Tokenizer
from tensorflow.keras.preprocessing.sequence import pad_sequences
from tensorflow.keras.layers import Dense, Input, GlobalMaxPooling1D
from tensorflow.keras.layers import Conv1D, MaxPooling1D, Embedding
from keras.layers import Bidirectional, GlobalMaxPool1D
from keras.models import Model
from keras import initializers, regularizers, constraints, optimizers, layers
Preprocess ham and spam hybrid email datasets (See Table 3)
#Load Emails in a data frame, the number of samples as shown in Table 3, 70% for training and 30% for test (for each baseline model)
df = pd.read_csv (r'C:..//PaceResearch//Dissertation//Spam//Experiments//Datasets//Stopwords_Removed_spam_assassainEmails_cleaned_3042_augmented_1034_20_percent_enron.csv',encoding='ISO-8859-1')
Eliminate the null values and create training and test sets
df = df.dropna()
X_train, X_test, y_train, y_test = train_test_split(df['text'], y, test_size=0.33, random_state = 0)
Convert sentences to sequences

```
max_vocab_size = 20000
tokenizer = Tokenizer(num_words=max_vocab_size)
tokenizer.fit_on_texts(X_train)
sequences_train = tokenizer.texts_to_sequences(X_train)
sequences_test = tokenizer.texts_to_sequences(X_test)
Pad Sequence
data_train = pad_sequences(sequences_train)
Get Sequence Length
T = data_train.shape[1]
Pad Test Set
data_test = pad_sequences(sequences_test, maxlen=T)
LSTM Architecture
Select an embedding dimensionality D
D = 50
Hidden state vector size
M = 30
i = Input(shape=(T,))
Embedding Layer
x = Embedding(V + 1, D)(i)
LSTM layer
x = LSTM(M, return_sequences=True, dropout=0.4)(x)
x = GlobalMaxPooling1D()(x)
Dense Layer
x = Dense(1, activation='sigmoid')(x)

model = Model(i, x)
print(model.summary())
Compile the LSTM Model
model.compile(optimizer='RMSprop', loss='binary_crossentropy',
metrics= METRICS)
Train the Model
r = model.fit(x=data_train, y=y_train, epochs=30, batch_size=64,
validation_data=(data_test, y_test))
```

4.2.4 *BERT Hyperparameter Tuning and Model Experiment Design*

The fine-tuning of the BERT hyperparameters is still in its infancy, with only a few published works produced on the subject. As pointed out in [89], ". . . despite significant success, fine-tuning remains unstable, especially when using the large variant of BERT ($BERT_{LARGE}$) on small datasets, where pre-training stand to provide the most significant benefit." In our study, we experimented with $BERT_{BASE}$ as well as $BERT_{LARGE,}$ however, the results of $BERT_{LARGE}$ were significantly lower in performance and not competitive with

other models in the study. Due to low performance, we removed $BERT_{LARGE}$ from the study.

To build the $BERT_{BASE}$ model, we noted the hyperparameters explored in the works of [89], [90], [91] . We incorporate their values when testing for the optimal hyperparameters for our models. After experimenting with a series of parameter tunings, we found the optimal parameters for the BERT model to be: batch size of 32, number of epochs 10, learning rate of 1e-3m, sequence length of 100, and the best optimizer to be adamW. We present the ranges tested and the optimal values used for our tasks in Table 13.

Table 13 BERT Hyperparameter Tuning

Hyperparameter	Description	Range Tested	Optimal Value
batch size	Number of samples taken for each epoch	[10, 15, 32, 64]	32
epochs	Number of times the entire training set passed through the network	[3, 4, 10, 15, 32, 64]	10
learning rate	Controls how much to update the weight in the optimization algorithms	[1e-3m, 1e-4, 3e-5, 5e-5]	1e-3m
sequence length		[50, 75, 100, 150, 175, 200, 300]	100
optimizer	Controls how well the hidden layers learn the dataset	[sdg, adamw, rmsprop]	adamw

The $BERT_{BASE}$ architecture is based on Transformers and uses multilayer bidirectional Transformer encoders for language representations. The developers of BERT [15] explained the $BERT_{SMALL}$ model uses 12 layers of transformer blocks with a hidden size of 768 and 12 self-attention heads. The BERT models were implemented using Jupyter

notebooks provided by the cloud-based Google Colab using 16GB GPUs provided with the Colab Pro subscription. We use the instructions provided by [92] and [93] to fine-tune the BERT model for our tasks. Table 14 provides details of the BERT model construction.

Table 14 BERT Model Design Implementation Source: Adapted from [92], [93]

```
BERT Model Construction
Install Transformers Library and other required packages
!pip install transformers=3.0.0
import transformers
import torch
import torch.nn as nn
import numpy as np
import pandas as pd
from sklearn.model_selection import train_test_split
from sklearn.metrics import classification_report
from transformers import AutoModel, BertTokenizerFast

specify GPU
device = torch.device("cuda")

Preprocess ham and spam hybrid email datasets  (Table 3)
#Load Emails in a data frame, the number of samples as shown in Table
3, 70% for training and 30% for test for each baseline model
df = pd.read_csv
(r'C:..//PaceResearch//Dissertation//Spam//Experiments//Datasets//S
topwords_Removed_spam_assassainEmails_cleaned_3042_augmented_1034_2
0_percent_enron.csv',encoding='ISO-8859-1')
Create Train and Test datasets
train_text, temp_text, train_labels, temp_labels =
train_test_split(df['text'], df['label'],
random_state=2018,
test_size=0.3,
stratify=df['label'])
Create Validation Datasets
Val_text, test_text, val_labels, test_labels =
train_test_split(temp_text, temp_labels,
random_state=2018,
test_size=0.5,
stratify=temp_labels)
Import BERT-base pretrained model
bert = AutoModel.from_pretrained('bert-base-uncased')
Load the BERT tokenizer
tokenizer = BertTokenizerFast.from_pretrained('bert-base-uncased')
```

Tokenize and encode sequences in the training set

```
tokens_train = tokenizer.batch_encode_plus(
    train_text.tolist(),
    max_length = max_seq_len,
    pad_to_max_length=True,
    truncation=True,
    return_token_type_ids=False
)
```

Tokenize and encode sequences in the validation set

```
tokens_val = tokenizer.batch_encode_plus(
    val_text.tolist(),
    max_length = max_seq_len,
    pad_to_max_length=True,
    truncation=True,
    return_token_type_ids=False
)
```

Tokenize and encode sequences in the test set

```
tokens_test = tokenizer.batch_encode_plus(
    test_text.tolist(),
    max_length = max_seq_len,
    pad_to_max_length=True,
    truncation=True,
    return_token_type_ids=False
)
```

Convert Integer Sequences to Tensors

```
# for train set
train_seq = torch.tensor(tokens_train['input_ids'])
train_mask = torch.tensor(tokens_train['attention_mask'])
train_y = torch.tensor(train_labels.tolist())

# for validation set
val_seq = torch.tensor(tokens_val['input_ids'])
val_mask = torch.tensor(tokens_val['attention_mask'])
val_y = torch.tensor(val_labels.tolist())

# for test set
test_seq = torch.tensor(tokens_test['input_ids'])
test_mask = torch.tensor(tokens_test['attention_mask'])
test_y = torch.tensor(test_labels.tolist())
```

Create Data Loaders and define batch size

```
from torch.utils.data import TensorDataset, DataLoader,
RandomSampler, SequentialSampler

#Define a batch size
batch_size = 32

# wrap tensors
train_data = TensorDataset(train_seq, train_mask, train_y)

# sampler for sampling the data during training
train_sampler = RandomSampler(train_data)
```

```
# dataLoader for train set
train_dataloader = DataLoader(train_data, sampler=train_sampler,
batch_size=batch_size)

# wrap tensors
val_data = TensorDataset(val_seq, val_mask, val_y)

# sampler for sampling the data during training
val_sampler = SequentialSampler(val_data)

# dataLoader for validation set
val_dataloader = DataLoader(val_data, sampler = val_sampler,
batch_size=batch_size)
```

Freeze BERT Parameters

```
for param in bert.parameters():
    param.requires_grad = False
```

Define Model Architecture

```
   def __init__(self, bert):

      super(BERT_Arch, self).__init__()

      self.bert = bert

      # dropout layer
      self.dropout = nn.Dropout(0.1)

      # relu activation function
      self.relu =  nn.ReLU()

      # dense layer 1
      self.fc1 = nn.Linear(768,512)

      # dense layer 2 (Output layer)
      self.fc2 = nn.Linear(512,2)

      #softmax activation function
      self.softmax = nn.LogSoftmax(dim=1)

     #define the forward pass
      def forward(self, sent_id, mask):

      #pass the inputs to the model
      _, cls_hs = self.bert(sent_id, attention_mask=mask)

      x = self.fc1(cls_hs)

      x = self.relu(x)

      x = self.dropout(x)

      # output layer
      x = self.fc2(x)
```

```
      # apply softmax activation
      x = self.softmax(x)

      return x
Pass the pretrained BERT to our define architecture
model = BERT_Arch(bert)

Push the model to GPU
model = model.to(device)

Optimizer from hugging face transformers
from transformers import AdamW

# define the optimizer
optimizer = AdamW(model.parameters(), lr = 1e-3)

Convert class weights to tensor
weights= torch.tensor(class_wts,dtype=torch.float)
weights = weights.to(device)
Set Loss Function
cross_entropy  = nn.NLLLoss(weight=weights)
Set number of training epochs
epochs = 10

Fine-Tune BERT
# function to train the model
def train():

  model.train()

  total_loss, total_accuracy = 0, 0

  # empty list to save model predictions
  total_preds=[]

  # iterate over batches
  for step,batch in enumerate(train_dataloader):

    # progress update after every 50 batches.
    if step % 50 == 0 and not step == 0:
      print('Batch {:>5,} of {:>5,}.'.format(step,
len(train_dataloader)))

    # push the batch to gpu
    batch = [r.to(device) for r in batch]

    sent_id, mask, labels = batch

    # clear previously calculated gradients
    model.zero_grad()

    # get model predictions for the current batch
    preds = model(sent_id, mask)

    # compute the loss between actual and predicted values
```

```
    loss = cross_entropy(preds, labels)

    # add on to the total loss
    total_loss = total_loss + loss.item()

    # backward pass to calculate the gradients
    loss.backward()

    # clip the the gradients to 1.0. It helps in preventing the
exploding gradient problem
    torch.nn.utils.clip_grad_norm_(model.parameters(), 1.0)

    # update parameters
    optimizer.step()

    # model predictions are stored on GPU. So, push it to CPU
    preds=preds.detach().cpu().numpy()

    # append the model predictions
    total_preds.append(preds)

  # compute the training loss of the epoch
  avg_loss = total_loss / len(train_dataloader)

  # predictions are in the form of (no. of batches, size of batch,
no. of classes).
  # reshape the predictions in form of (number of samples, no. of
classes)
  total_preds  = np.concatenate(total_preds, axis=0)

  #returns the loss and predictions
  return avg_loss, total_preds

Function for evaluating the model
def evaluate():

  print("\nEvaluating...")

  # deactivate dropout layers
  model.eval()

  total_loss, total_accuracy = 0, 0

  # empty list to save the model predictions
  total_preds = []

  # iterate over batches
  for step,batch in enumerate(val_dataloader):

    # Progress update every 50 batches.
    if step % 50 == 0 and not step == 0:

      # Calculate elapsed time in minutes.
      elapsed = format_time(time.time() - t0)
```

```
      # Report progress.
      print('  Batch {:>5,}  of  {:>5,}.'.format(step,
len(val_dataloader)))

    # push the batch to gpu
    batch = [t.to(device) for t in batch]

    sent_id, mask, labels = batch

    # deactivate autograd
    with torch.no_grad():

      # model predictions
      preds = model(sent_id, mask)

      # compute the validation loss between actual and predicted
values
      loss = cross_entropy(preds,labels)

      total_loss = total_loss + loss.item()

      preds = preds.detach().cpu().numpy()

      total_preds.append(preds)

  # compute the validation loss of the epoch
  avg_loss = total_loss / len(val_dataloader)

  # reshape the predictions in form of (number of samples, no. of
classes)
  total_preds = np.concatenate(total_preds, axis=0)

  return avg_loss, total_preds
```

Start Model Training

```
# set initial loss to infinite
best_valid_loss = float('inf')

# empty lists to store training and validation loss of each epoch
train_losses=[]
valid_losses=[]

#for each epoch
for epoch in range(epochs):

    print('\n Epoch {:} / {:}'.format(epoch + 1, epochs))

    #train model
    train_loss, _ = train()

    #evaluate model
    valid_loss, _ = evaluate()

    #save the best model
    if valid_loss < best_valid_loss:
```

```
            best_valid_loss = valid_loss
            torch.save(model.state_dict(), 'saved_weights.pt')

        # append training and validation loss
        train_losses.append(train_loss)
        valid_losses.append(valid_loss)

        print(f'\nTraining Loss: {train_loss:.3f}')
        print(f'Validation Loss: {valid_loss:.3f}')

Load Weights  of the Best Saved Model
path = 'saved_weights.pt'
model.load_state_dict(torch.load(path))

Get Predictions for Test Data
with torch.no_grad():
  preds = model(test_seq.to(device), test_mask.to(device))
  preds = preds.detach().cpu().numpy()

Model's performance
preds = np.argmax(preds, axis = 1)
print(classification_report(test_y, preds))
```

4.3 Experimental Result and Data Analysis

4.3.1 *Baseline Performance of All Models*

In our first experiment, we build models that are trained and tested on datasets discussed in section 3.6.2 and detailed in Table 3. For each algorithm in the study, we create four models, one based on the hybrid datasets consisting of the 40%, 30%, 20%, and 10% respectively. These datasets are hybrids of two benchmark datasets to create models from the same distribution.

The impressive performance of the RF, which is an older, traditional algorithm was nearly equal in accuracy as the more advanced deep learning algorithms of the CNN and LSTM. Recall from section 2.1 that the RF is an underexplored algorithm for use in spam filters, which makes its impressive results especially encouraging.

We note the competitive performance in terms of accuracy of the RF, CNN, and LSTM models, and the lower accuracy score of the BERT model. The RF, CNN, and LSTM on average perform at about 97% on across all datasets. The BERT model performs at 94% on three of the datasets. In our study, the BERT models did not perform as well as the other models. The baseline performance of all models is presented in Table 15 and Figure 18. We note the results we achieved in our BERT models are competitive with the results of the work of [22] in terms of baseline accuracy on NLP classification tasks.

Table 15 Baseline Performance Comparison in Terms of Overall Accuracy

		RF	CNN	LSTM	BERT
Hybrid Datasets	40%	97	96.95	96.21	94
	30%	96	97.1	96.51	94
	20%	98	96.89	97.17	89
	10%	98	97.47	94.54	94
	0%	98	98.25	98.11	94

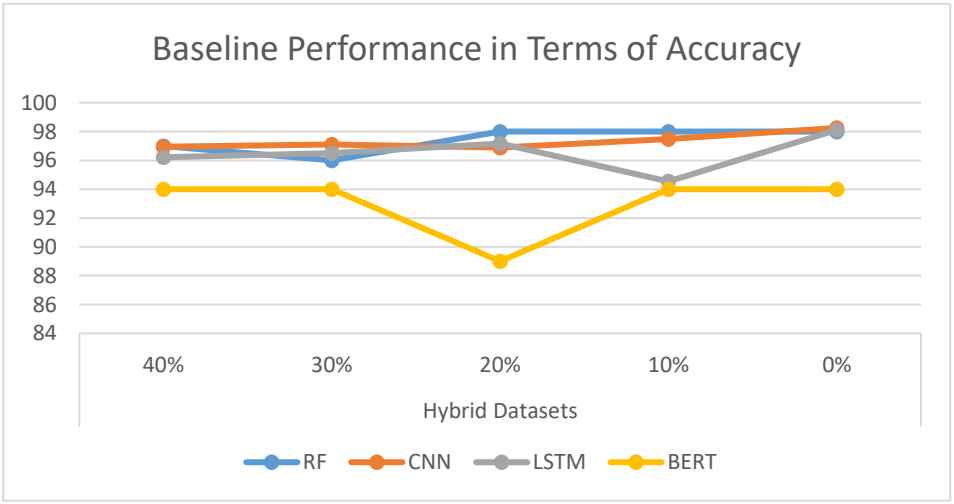

Figure 14 Baseline Performance in Terms of Accuracy: All Models

In [22], the authors analyzed 150 deep learning models for text classification on sentiment analysis using six popular benchmark datasets, including: IMDB-Large Movie Review Dataset [94], SST-2–The Stanford Sentiment Treebank [95], Yelp Reviews [96], and Amazon Reviews [97]. We add their results in Figure 23 for ease of illustrating their BERT results compared to our results. As shown in Figure 23, their $Bert_{BASE}$ models achieve scores of: IMDB: 95.63, SST-2: 93.50, Amazon-2: 96.04, Amazon-5: 61.60, Yelp-2: 98.08, and Yelp-5: 70.58, with an average performance of 85.9%. Our score of 94% accuracy is competitive with five of the six models, except for their Yelp-2 model, which scored 98% in accuracy. We note the use of the Naïve Bayes classical machine learning in their work

rather the RF compared in our work. We also note the text classification task of sentiment analysis for opinion mining in their work, rather than the spam classification in our work.

Table 16 $BERT_{BASE}$ Results from Table 1 in [28]

Method	IMDB	SST-2	Amazon-2	Amazon-5	Yelp-2	Yelp-5
Naive Bayes [43]	-	81.80	-	-	-	-
LDA [214]	67.40	-	-	-	-	-
BoW+SVM [31]	87.80	-	-	-	-	-
tf.Δ idf [215]	88.10	-	-	-	-	-
Char-level CNN [50]	-	-	94.49	59.46	95.12	62.05
Deep Pyramid CNN [49]	-	84.46	96.68	65.82	97.36	69.40
ULMFiT [216]	95.40	-	-	-	97.84	70.02
BLSTM-2DCNN [40]	-	89.50	-	-	-	-
Neural Semantic Encoder [95]	-	89.70	-	-	-	-
BCN+Char+CoVe [217]	91.80	90.30	-	-	-	-
GLUE ELMo baseline [22]	-	90.40	-	-	-	-
BERT ELMo baseline [7]	-	90.40	-	-	-	-
CCCapsNet [76]	-	-	94.96	60.95	96.48	65.85
Virtual adversarial training [173]	94.10	-	-	-	-	-
Block-sparse LSTM [218]	94.99	93.20	-	-	96.73	
BERT-base [7, 154]	95.63	93.50	96.04	61.60	98.08	70.58
BERT-large [7, 154]	95.79	94.9	96.07	62.20	98.19	71.38
ALBERT [147]	-	95.20	-	-	-	-
Multi-Task DNN [23]	83.20	95.60	-	-	-	-
Snorkel MeTaL [219]	-	96.20	-	-	-	-
BERT Finetune + UDA [220]	95.80		96.50	62.88	97.95	62.92
RoBERTa (+additional data) [146]	-	96.40	-	-	-	-
XLNet-Large (ensemble) [156]	96.21	96.80	97.60	67.74	98.45	72.20

This section provides the results of the baseline performance on datasets that were trained and tested with datasets from the same distribution. In the next section, we present the results of these models when tested with datasets from a different distribution to simulate dataset drift.

4.3.2 *Model Performance in the Face of Dataset Drift*

We present the overall performance to simulated dataset drift of the RF across all metrics in Table 17 and Figures 19 and 20. We note only a small loss in accuracy with the datasets constructed with 40% of the Enron dataset as compared to a larger decline when constructed with only 10%. This performance is repeated by the CNN as shown in Table 18 and Figures 21 and 22, as well as the LSTM as shown in Table 19 and Figures 23 and 24. The BERT model was more sensitive to dataset drift as shown by the decline in performance as shown in Table 20 and Figures 25 and 26.

Table 17 Random Forest Drift Analysis

Random Forest Drift Analysis	Accuracy	Recall Ham	Recall Spam	Precision Ham	Precision Spam	F1 Ham	F1 Spam
Drift 40%	97	96	99	99	91	98	95
Drift 30%	94	94	95	98	87	96	91
Drift 20%	95	94	95	98	87	96	91
Drift 10%	91	93	86	94	84	94	85
0% - OOD	75	98	17	74	78	85	28

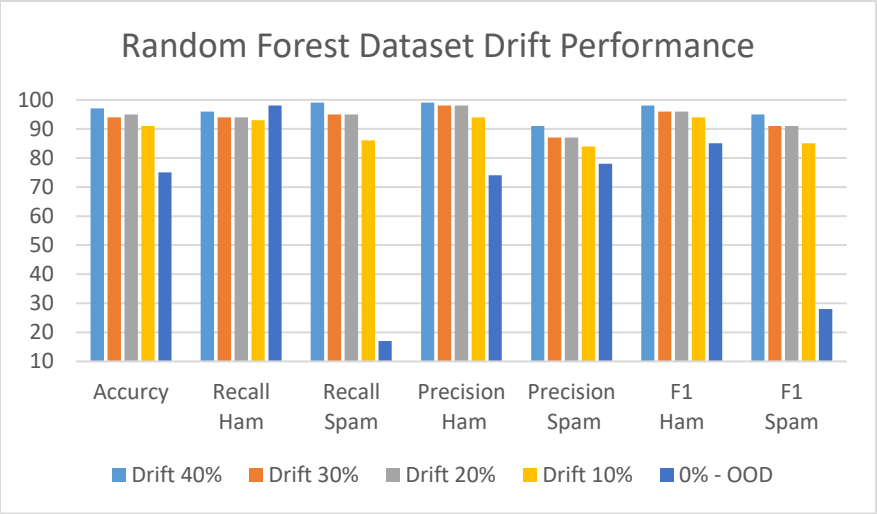

Figure 15 Random Forest Dataset Drift Performance: All Metrics

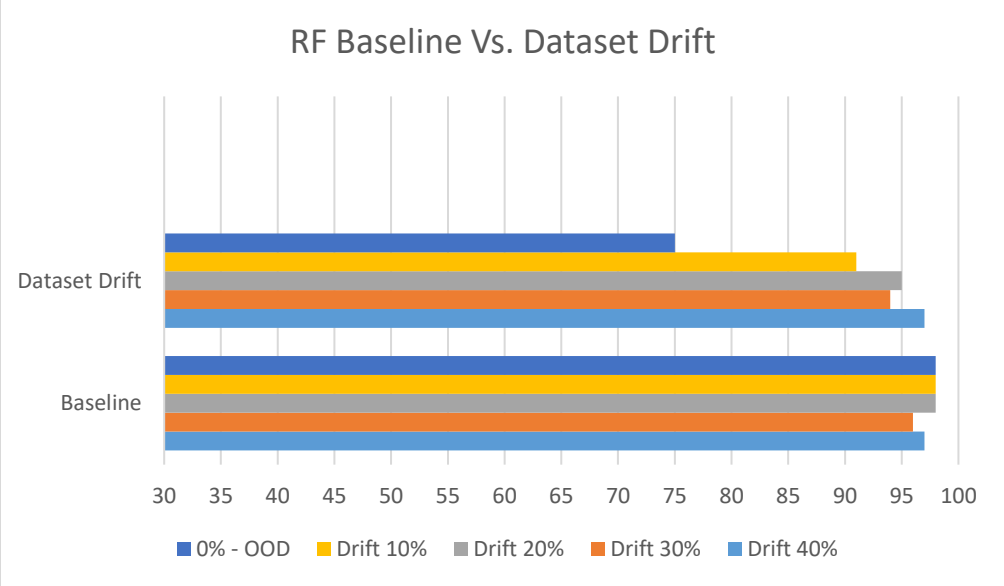

Figure 16 Random Forest Baseline Vs. Dataset Drift Metric: Accuracy

Table 18 CNN Dataset Drift Performance Across All Metrics

	Precision Wgt. Avg	Recall Wgt. Avg	Accuracy	Precision ham	Precision Spam	Recall Ham	Recall Spam	F1 ham	F1- 1 Spam
40%	98.5	96.08	96.2	91.09	96.1	96.49	96.07	93.71	97.27
30%	98.5	96.17	96.26	91.28	98.5	96.49	96.16	93.81	97.32
20%	96.92	94.71	94.14	97.94	96.9	92.76	94.7	90.28	95.8
10%	96.15	91.15	91.17	81.09	96.2	91.28	91.14	85.58	93.58
OOD	80.55	50.64	56.51	37.3	80.6	70.61	50.63	48.82	62.18

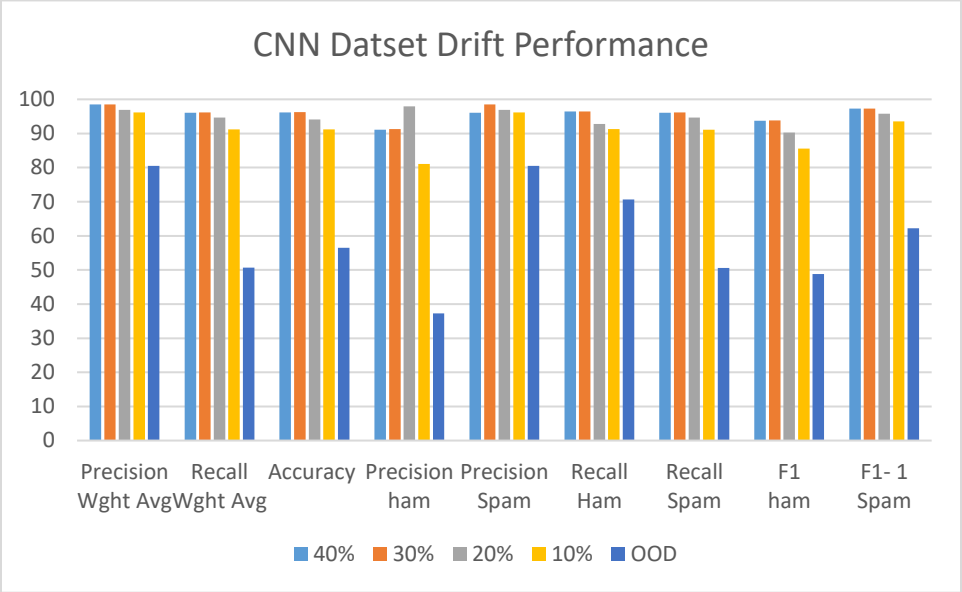

Figure 17 CNN Dataset Drift Performance Across All Metrics

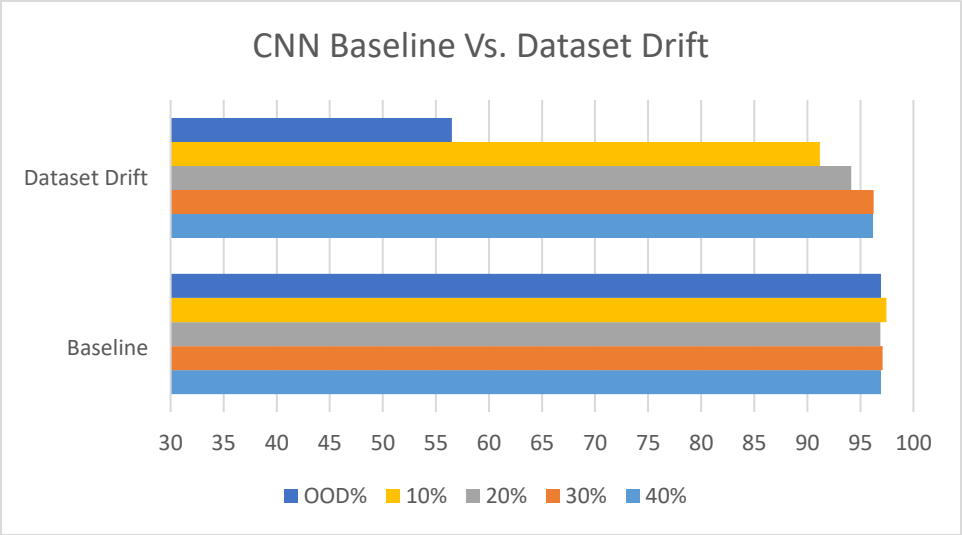

Figure 18 CNN Dataset Drift Performance Metric: Accuracy

Table 19 LSTM Dataset Drift Performance across all Metrics

	Precision Wgt Avg	Recall Wgt Avg	Accuracy	Precision ham	Precision Spam	Recall Ham	Recall 1Spam	F1- Ham	F1- 1 Spam
40%	92.65	96.73	96.83	98.64	92.65	96.8	96.73	97.74	94.64
30%	93.72	92.6	96.05	96.99	93.72	97.5	92.59	97.22	93.15
20%	90.11	94.26	94.34	97.61	90.11	95.8	94.26	96.68	92.14
10%	76.62	84.19	87.97	93.27	76.62	89.5	84.18	91.35	80.22
OOD	41.08	69.65	62.75	82.86	41.51	59.9	69.64	69.56	52.01

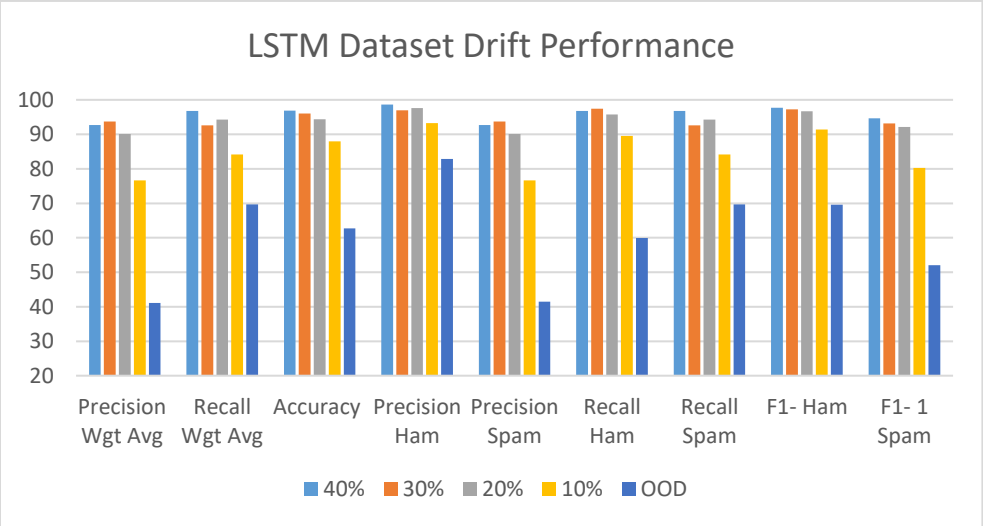

Figure 19 LSTM Dataset Drift Performance across all Metrics

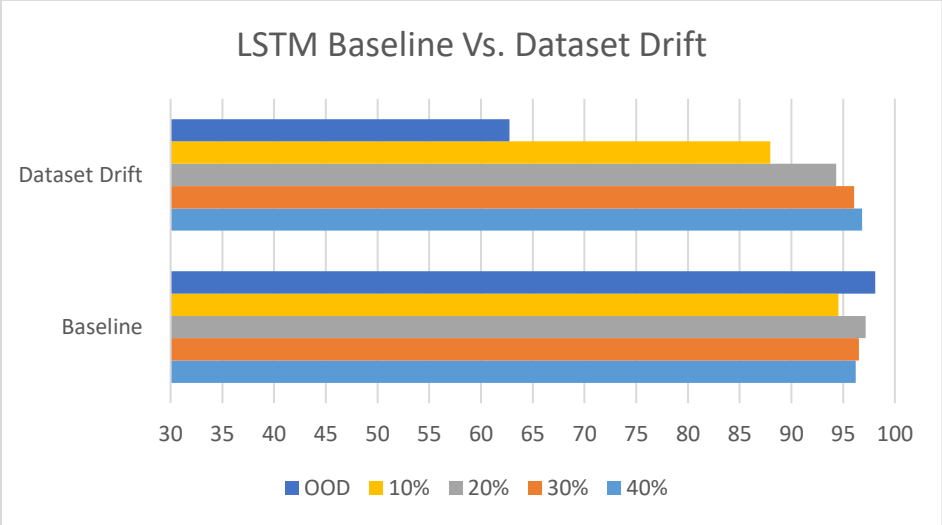

Figure 20 LSTM Dataset Performance Metric: Accuracy

Table 20 BERT Dataset Drift Performance Across All Metrics

	Precision Wt. Avg	Recall Wt. Avg	Accuracy	Precision ham	Precision Spam	Recall Ham	Recall Spam	F1 Ham	F1- 1 Spam
40%	85	84	84	83	89	97	52	90	66
30%	86	86	86	89	80	93	71	91	75
20%	80	79	79	78	85	98	33	87	47
10%	73	75	75	76	66	95	25	84	37
OOD	67	70	70	76	47	85	39	80	39

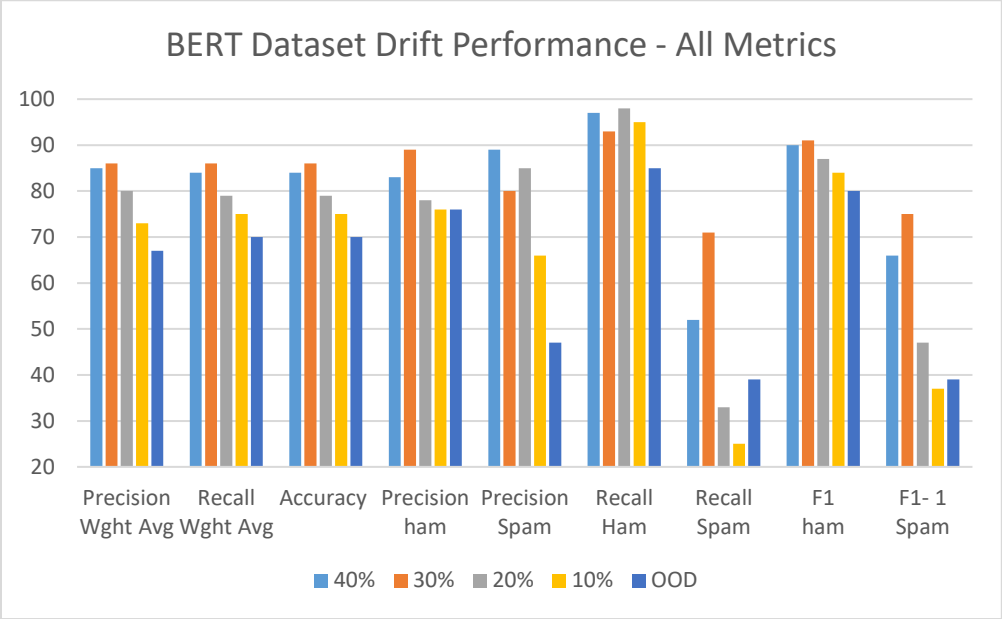

Figure 21 BERT Dataset Performance across all Metrics

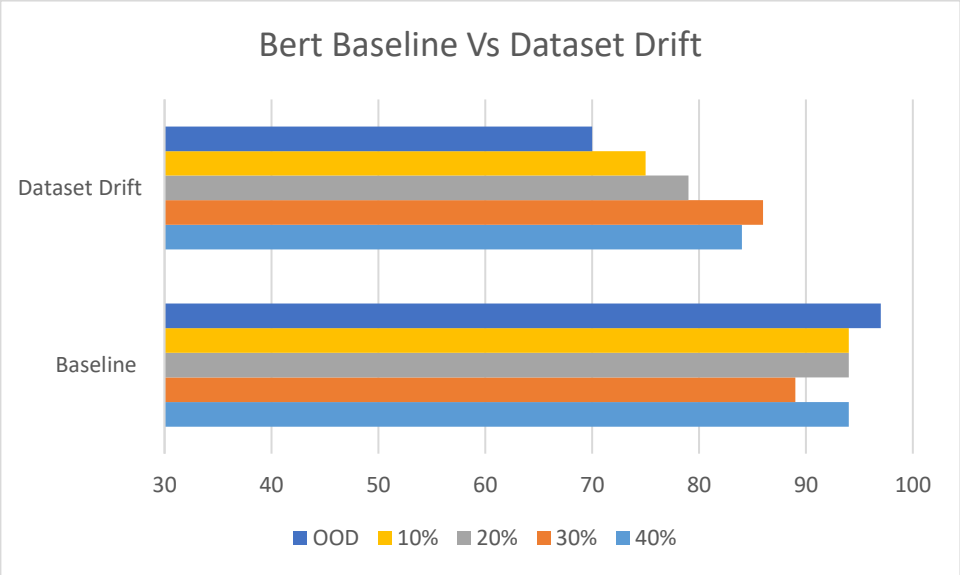

Figure 22 BERT Dataset Drift Performance Metric: Accuracy

On comparing the results of all the models in Table 21, we see the baseline performance was competitive across all modes with an average of 97% for the RF, 96% for the CNN and LSTM, and 92% for BERT. However, in the face of drift, it is clear the BERT model suffers the most loss, as can be seen from results of the 10% hybrid dataset. We see a basic downward trend from the 40% hybrid dataset to the 10% hybrid dataset. The results of loss using the 40% hybrid dataset was a loss of only 1% for RF, CNN, and LSTM, while BERT has a loss of 11%. It is interesting to note the loss of BERT using the 20% hybrid dataset was only 3%, which was consistent with the other models. When using the 10% hybrid dataset, which represents the most dataset drift, BERT suffered a 16% loss compared to the 7% of the RF and LSTM and 6% of the CNN.

It is important to note that regarding the BERT model, we experienced vastly different results depending on the random seeds selected for the data, as well as the random seeds

selected for the model. The dramatic effect of the random seed on the model performance was reported in the work of [98], where they indicated:,

> "Despite the strong empirical performance of fine-tuned models, fine-tuning BERT is an unstable process: training the same model with multiple random seeds can result in a large variance of the task performance. . . . [F]ine-tuning a model multiple times on the same dataset, varying only the random seed, led to a large standard deviation of the fine-tuning accuracy" [98].

We also note the importance of stopwords remaining in the BERT model for better overall results in terms of accuracy.

Table 21 Dataset Drift Performance All Models Baseline vs. Drift Scores

		RF			CNN			LSTM			BERT		
		Baseline	Drift	Loss	Baseline	Drift	Loss	Baseline	Drift	Loss	Baseline	Drift	Loss
Hybrid Dataset	40%	**97**	**96**	**-1%**	**96.95**	**96.2**	**-1%**	**96.21**	**96.83**	**1%**	**94**	**84**	**-11%**
	30%	96	95	-1%	97.1	96.26	-1%	96.51	96.05	0%	94	84	-11%
	20%	98	94	-4%	96.89	94.14	-3%	97.17	95.34	-2%	89	86	-3%
	10%	**98**	**91**	**-7%**	**97.47**	**91.17**	**-6%**	**94.54**	**87.97**	**-7%**	**94**	**79**	**-16%**
	0%	98	75	-23%	98.25	56.51	-42%	98.11	62.75	-36%	94	75	-20%

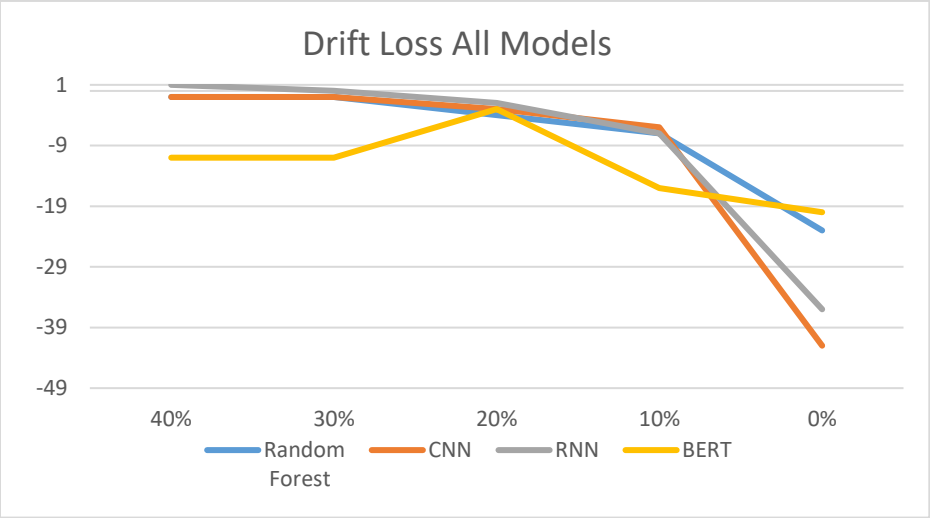

Figure 23 Drift Loss across all Models in Terms of Accuracy

We suspect that BERT was more brittle to dataset drift because of how it is designed to contextualize word embeddings at a nuanced level. As a result, the slightest change in the data can drastically affect the context, and hence the performance of BERT.

4.4 Experiment Highlights: The Importance of Stopwords

4.4.1 *Feature Extraction: Stopwords for Random Forest as Compared to BERT*

4.4.1.1 Random Forest and Stopwords

We found that removing the stopwords does indeed increase the performance of the RF model. This is in part due to the TF-IDF vectorization used for feature extraction that obtains no value from the stopwords. As we show in Figure 28, the accuracy performance of 97% for the RF model without stopwords consistently outperformed the model with the stopwords included, resulting in 94% accuracy as shown in Figures 28 and 29. We add Figure 29 for a side-by-side comparison of the RF with and without stopwords.

	precision	recall	f1-score	support
ham	0.97	0.99	0.98	898
spam	0.96	0.91	0.94	315
accuracy			0.97	1213
macro avg	0.97	0.95	0.96	1213
weighted avg	0.97	0.97	0.97	1213

Figure 24 Random Forest Baseline Stopwords Removed from Training Dataset

	precision	recall	f1-score	support
ham	0.93	1.00	0.96	894
spam	0.98	0.80	0.88	319
accuracy			0.94	1213
macro avg	0.96	0.90	0.92	1213
weighted avg	0.95	0.94	0.94	1213

Figure 25 Random Forest Baseline Stopwords Included in Training Dataset

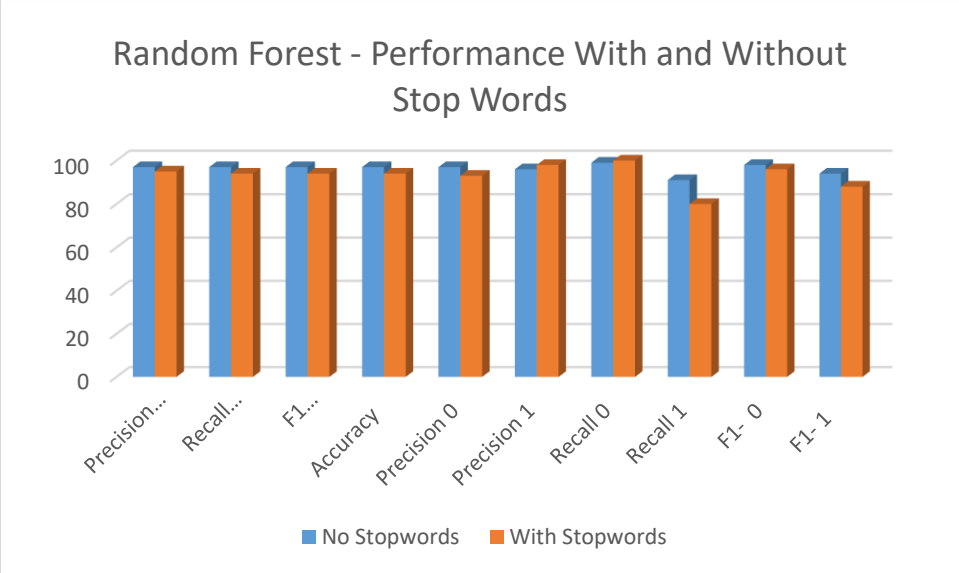

Figure 26 Random Forest - Performance with and Without Stopwords

4.4.1.2 $BERT_{BASE}$ and Stopwords

While the stopwords only added "noise" to the RF algorithm, and reduced its performance, we found the opposite to be true for the Transformer model of BERT. Recall from section 2.4.2.4 that BERT uses Transformer technology to create intuitive word embedding to represent features from the text. Prepositions like "for" and "to" may matter to the meaning of a given word.

In our study, we found that removing the stopwords had a negative impact on the performance of the BERT models since BERT relies on some stopwords to provide context to help capture meaning. As explained by Pandu Nayak, Vice President of Search at Google,

> Here's a search for "2019 brazil traveler to usa need a visa." The word "to" and its relationship to the other words in the query are particularly important to understanding the meaning. It's about a Brazilian traveling to the U.S., and not the other way around. Previously, our algorithms wouldn't understand the importance of this connection, and we returned results about U.S. citizens traveling to Brazil. With BERT, Search is able to grasp this nuance and know that the very common word "to" actually matters a lot here, and we can provide a much more relevant result for this query. [40]

The performance of the models with stopwords consistently performed better than the models without the stopwords included. When stopwords were removed, there was a decline in the performance of the BERT model, as shown in Figure 31 compared to Figure 32. We add Figure 33 for a side-by-side comparison of BERT with and without stopwords.

	precision	recall	f1-score	support
0	0.97	0.95	0.96	457
1	0.84	0.91	0.87	150
accuracy			0.94	607
macro avg	0.91	0.93	0.92	607
weighted avg	0.94	0.94	0.94	607

Figure 27 BERT- With Stopwords in Training Dataset

	precision	recall	f1-score	support
0	0.96	0.89	0.93	457
1	0.79	0.89	0.80	150
accuracy			0.89	607
macro avg	0.85	0.89	0.87	607
weighted avg	0.90	0.89	0.90	607

Figure 28 BERT-Stopwords Removed

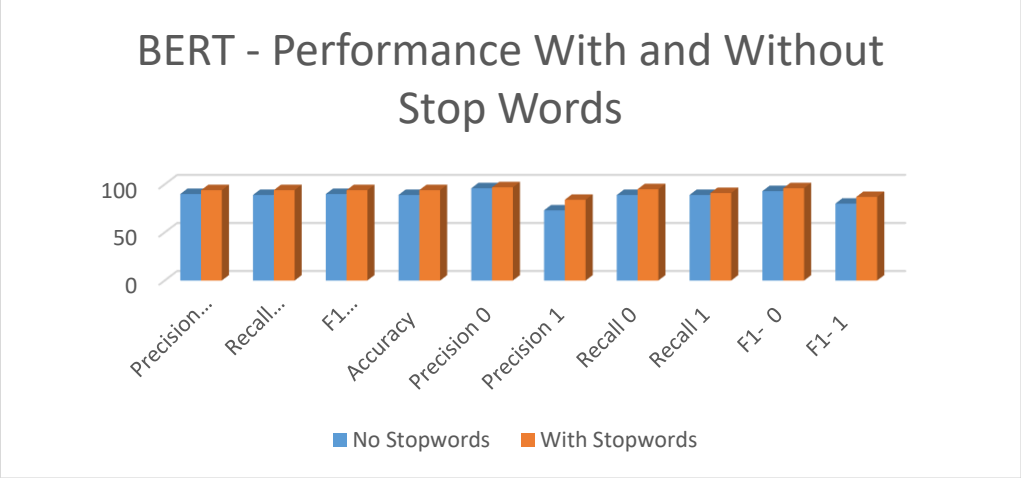

Figure 29 BERT- With and Without Stopwords

4.4.2 *Model Size: Bigger More Advanced Models Are Not Necessarily Better*

Our experiments show that larger more advanced and complex models are not always better when faced with dataset drift. In our study, the $BERT_{BASE}$ did not perform as well as the RF. $BERT_{BASE}$ achieved a baseline accuracy score 94%, which was not as impressive as the 97% accuracy score of the RF. The results of robustness to dataset drift show the RF, which is based on simple decision trees, was more stable, suffering only a 1% to 7% loss, whereas the $BERT_{BASE}$ had a loss of 11% to 16%.

Our research supports the findings of the authors of [99] who concluded that while pretrained Transformer models are moderately robust to out-of-distribution data, there remains room for future research on robustness. Our research also shows that $BERT_{Base}$ outperformed the larger more complex $BERT_{Large}$, as shown in Tables 22 and 23.

Table 22 $BERT_{BASE}$ Baseline Performance Across Various Parameters

$BERT_{BASE}$ Baseline														
Max Length	Batch Size	Epochs	Train loss	Validation loss	Precision Weighted Avg.	Recall Weighted Avg.	F1 Score	**Accuracy**	Precision 0	Precision 1	Recall 0	Recall 1	F1- 0	F1- 1
250	32	10	241	213	94	94	94	**94**	97	84	95	91	96	87
250	32	15	257	233	94	93	93	**93**	98	81	93	93	95	87
250	32	20	317	219	93	93	93	**93**	96	83	94	89	95	86
200	32	10	301	249	92	92	92	**92**	96	81	93	88	95	88
200	32	15	272	282	92	91	91	**91**	97	78	92	90	94	84
200	32	20	279	314	92	92	92	**92**	96	79	92	89	94	84
150	32	10	288	353	92	91	91	**91**	97	77	91	90	94	83
150	32	15	248	181	93	92	92	**92**	97	80	93	91	95	85
150	32	20	255	261	93	93	93	**93**	97	82	94	91	95	86

100	32	10	285	390	92	92	92	**92**	95	85	95	84	95	84
100	32	15	290	355	94	94	94	**94**	97	85	95	91	96	88
100	32	20	300	858	94	94	94	**94**	96	86	95	85	96	88
75	32	10	257	209	94	94	94	**94**	96	86	95	89	96	88
75	32	15	219	188	94	94	94	**94**	97	87	96	90	96	89
75	32	20	217	496	94	94	94	**94**	97	86	95	91	96	88
75	16	20	255	203	94	94	94	**94**	98	85	95	93	96	89
75	16	20	264	192	94	94	94	**94**	97	83	94	93	96	88
75	64	20	231	217	93	92	93	**92**	97	90	93	92	95	86
50	32	10	288	243	90	89	90	**89**	96	73	89	89	93	80
50	32	15	237	201	91	90	90	**90**	97	75	90	91	93	82
50	32	20	237	180	93	93	93	**93**	96	84	94	89	95	86
25	32	10	409	452	86	86	86	**85**	91	71	90	74	91	73
25	32	15	317	276	88	87	86	**86**	94	69	88	83	91	75
25	32	20	260	268	90	89	90	**89**	94	77	92	83	93	79

Table 23 Bert LARGE Baseline Performance Across Various Parameters

Bert Large Baseline Hyper-Parameter Tuning														
Max Length	**Batch Size**	**Epochs**	**Train loss**	**Validation loss**	**Precision Weighted Average**	**Recall Weighted Average**	**F1 Score**	**Accuracy**	**Precision 0**	**Precision 1**	**Recall 0**	**Recall 1**	**F1-Score 0**	**F1-Score 1**
250	32	10	503	476	83	72	74	**72**	94	47	67	88	78	61
250	32	15	579	504	82	70	73	**70**	94	44	63	88	76	59
250	32	20	557	489	83	82	83	**82**	90	62	86	71	88	66
200	32	10	653	616	76	71	71	**73**	87	45	73	66	79	53
200	32	15	516	536	81	73	73	**75**	92	48	71	81	80	60
200	32	20	525	476	82	68	70	**68**	95	43	61	89	74	58
150	32	10	436	363	88	86	87	**86**	94	69	88	82	91	75
150	32	15	450	375	87	86	86	**86**	92	69	89	78	91	73
150	32	20	535	469	85	79	80	**79**	96	54	75	89	85	68
100	32	10	427	435	88	86	87	**86**	95	68	86	87	91	76
100	32	15	491	418	88	84	85	**84**	96	62	82	90	89	74
100	32	20	447	650	86	80	81	**80**	96	56	76	91	85	69

50	32	10	461	415	86	83	84	**83**	84	62	82	85	88	72
50	32	15	485	494	86	86	86	**86**	91	70	89	74	90	72
50	32	20	513	451	85	79	80	**79**	96	55	76	89	84	68
25	32	10	444	403	86	83	84	**83**	94	62	82	85	88	78
25	32	15	493	494	85	83	84	**83**	93	62	84	81	88	70
25	32	20	462	563	87	79	80	**79**	97	54	74	94	84	68

4.5 Environment (Hardware, Software, Tools)

4.5.1 *Hardware*

The experiments in our study were run on CPU's (laptop and desktop) for all models, except the BERT models. For the BERT models, we used the GPU's available on Google's Colaboratory (Colab) [100] [101] environment. Google's Colab provided 8GB of GPU, which worked well for the $BERT_{Base}$ proof of concept, however, was not sufficient when experimenting with various hyperparameter tuning, which made it necessary to upgrade to a Colab Pro subscription, providing a Tesla 0199 16GB GPU. Additionally, we were able to expand BERT experiments using 32GB GPUs made available by Norfolk State University using their high-performance computing (HPC) clusters located in the Center of Excellence in the Cyber Security Research Laboratory [102]. The RF models were implemented using the ensemble method in the scikit learn library version 0.22.2. post 1 on commodity laptop and desktop hardware

4.5.2 *Software*

All experiments were implemented using the Python scripting programming language, including both Anaconda [103] and Jupyter notebooks [104] platforms. Jupyter notebooks are applications used for data analysis that allows you to develop and run codes in blocks using a web browser or using the JupyterLab [105]. Anaconda is a distribution of packages built for data science that comes with Python, and over 150 scientific packages that are

most commonly used in the building of data science applications, including the Spyder IDE. Spyder [106] is used to write and compile Python code in the Anaconda ecosystem. Spyder is a very powerful text editor that provides all the necessary features, starting from code framing to its deployment.

We used many open source natural language and machine learning tools managed by Anaconda ("conda"), including scikit-learn [107] Keras [108], and TensorFlow [109]. Scikit-learn has many ready to use machine learning algorithms. TensorFlow is a Google implementation of tensor calculus and has become one of the primary frameworks used to implement deep learning neural networks on GPUs and TPUs. The Keras framework makes TensorFlow more user-friendly to enable fast experimentation with deep neural networks. We perform our BERT experiments using the Hugging Face Transformer library [110] implemented on the Google Colaboratory platform.

We highlight many of the various frameworks, software tools, and libraries used in our study in Table 24.

Table 24 List of Software and Tools

Use	Frameworks, Software, and Libraries	Ref
Data Science Programming Environments Tools	Anaconda	[103]
	Google's Colabatory (Colab)	[100]
	Jupyter Notebooks	[104]
Data Preparation	NLTK	[27]
	Spacy	[26]
	Beautiful Soup	[73]
	pandas	[111]
	UrlExtract	[76]
	Regex	[75]

Plotting	matplotlib.pyplot seaborn	[112]
Non- contextual Feature Representation	TF-IDF / BoW	[29]
Contextual Feature Representation (Word Embeddings)	BERT	[15]
	Word2VEC	[30]
Machine Learning	Sklearn/SciKit Lern	[107]
Deep Learning	Keras, Tensorflow, Pytorch, Sklearn	[108] [109] [107] [113, 87]
BERT language Model	Transformers (Huggingface)	[110]
Google Colaboratory	Cloud GPU and Extended RAM	[100]

In Table 25, we acknowledge and highlight various training tutorials, videos, and learning guides that were used to help understand the emerging technologies presented in our study.

Table 25 Training Tutorials, Videos, and Guides

Training Source	Reference
Hands on Machine Learning with Scikit-Learn, Keras, and Tensor Flow	[114]
Udemy Course: Natural Language Processing (NLP) in Python with 8 Projects Learn	[115]
Udemy Course: BERT—Most Powerful NLP Algorithm by Google	[93]
Machine-Learning SPAM Classifier	[78]

Chapter 5

Conclusion

5.1 Conclusion

We presented an experimental performance comparison of the sensitivity, and conversely the robustness, for data drift of the classical RF machine learning algorithm, as well as deep learning algorithms of CNN, LSTM, and Google's state of the art BERT models. Overall, we found the RF, CNN, and LSTM algorithms were less likely to be impacted by dataset drift when used for spam filter applications. On comparing the results of all the models, we see the baseline performance was competitive across all modes with an average of 97% for the RF, 96% for the CNN and LSTM, and 92% for BERT. The results of loss in the face of data drift when using the 40% hybrid dataset resulted in a loss of only 1% for RF, CNN, and LSTM, while BERT had a loss of 11%. The 40% dataset represents the dataset with the least amount of dataset drift. When using the 10% hybrid dataset, which represents the most dataset drift, BERT suffered a 16% loss compared to the 7% of the RF and LSTM and 6% of the CNN.

It is important to note that regarding the BERT model, we experienced vastly different results depending on the random seeds selected for the data, as well as the random seeds selected for the model. The dramatic effect of the random seed on the model performance was reported in the work of [98], where they indicated,

> Despite the strong empirical performance of fine-tuned models, fine-tuning BERT is an unstable process: training the same model with multiple random seeds can result in a large variance of the task performance. . . . [F]ine-tuning a model

multiple times on the same dataset, varying only the random seed, led to a large standard deviation of the fine-tuning accuracy" [98].

We also note the importance of stopwords remaining in the BERT model for better overall results in terms of accuracy. We detail our contributions in section 5.2 and our plans for future work in section 5.3.

5.2 Contributions

5.2.1 *Contribution 1: Comparative Analysis of Sensitivity to Dataset Drift*

To our knowledge, we are among the first to perform a comparative analysis for the inherent robustness for dataset drift of selected traditional machine learning algorithms, deep learning algorithms, and the cutting-edge Transformer based BERT. This research may be helpful to future researchers in the model selection for classification tasks, especially if the application is subject to dataset drift or concept drift, such as monitoring and control tasks, information management, analytics, and diagnostics, among many others.

5.2.2 *Contribution 2: The Significant Role of Stopwords in RF Vs. BERT Models*

We contribute to the study and discussion of the significance of stopwords in feature selection for machine learning models. The results of our work should serve as a warning for researchers using BERT to proceed with caution before automatically removing stopwords. In many NLP tutorials and training guides, the removal of stopwords is strongly encouraged as they are portrayed as not being informative to the meaning of the sentence and are said to add unnecessary "noise" that slows the training time. We invite interested readers to the work of [116], who found significant variations and disparities in the

stopword list used by popular open-source software packages such as NLTK, Scikit-learn, and Spacy. The researchers in [117] found there are benefits in model quality when stopwords are removed, however, beyond high probability terms, the effects of stop list on training are limited. We point out that neither of these works evaluate the use of stopwords on BERT models.

In our experimentation, we performed an analysis of stopword removal on a traditional algorithm using TF-IDF vectors, as compared to the word embedding framework of BERT. We found that while traditional learners such as the RF perform better with stopwords removed, Transformer BERT performs better with the stopwords included as it uses stopwords to understand other words in the context of prepositional phrases.

5.2.3 *Contribution 3: Size Matters—Bigger Is Not Necessarily Better*

When faced with dataset drift, the RF declined by a 7% loss compared to baseline, while the performance of the $BERT_{BASE}$ declined by 16%. As discussed in the previous chapter, the $BERT_{LARGE}$ did not perform as well as the $BERT_{BASE}$ when evaluating for robustness to dataset drift and was not competitive with the RF in accuracy for even the baseline model. Our research supports the findings of the authors of [99] who concluded that while pretrained Transformers are moderately robust to out-of-distribution data, there remains room for future research on robustness.

5.2.4 *Contribution 4: Analysis of Hyperparameter Tuning on $BERT_{BASE}$*

As discussed earlier, the fine tuning of the BERT hyperparameters is still in its infancy, with only a few published works produced on the subject. As pointed out in [89], "[D]espite significant success, fine-tuning remains unstable."

Our work contributes to this area of research as we explored the results of the hyperparameter tuning: (1) max sequence lengths, (2) number of epochs, (3) batch size, and (4) learning rate. As shown in Tables 8 and 11. We found the best hyperparmeters for our particular dataset was a max sequence length = 75, number of epochs = 20, batch size = 16, and the learning rate = 1e-5. The full results of hyperparmeter tuning exploration for $BERT_{BASE}$ was provided in Table 8.

5.3 Future Work

5.3.1 *Create a Specialized Pretrained BERT Language Model for Spam and Phishing*

We support the future work recommendations found in [56], which concluded "Research on effectively integrating concept drift handling techniques with machine learning methodologies for data-driven applications is highly desired." To this end, we look at the existence of several specialized pre-trained BERT models publicly available. As future work, we would look to build a "SpamBERT" that would join the current domain-specific pretrained BERT models, including the following.

SciBERT (biomedical and computer science literature corpus)

FinBERT (financial services corpus)

BioBERT (biomedical literature corpus)

ClinicalBERT (clinical notes corpus)

patentBERT (patent corpus)

5.3.2 *Explore Broader Hyperparameter Tuning for the BERT Model*

As discussed earlier, the fine tuning of the BERT hyperparameter is still in its infancy, with only a few published works produced on the subject. Our plans for future works include exploring fine-tuning hyperparameter more broadly.

References

[1] Ovadia, Y. et al., "Can You Trust Your Model's Uncertainty? Evaluating Predictive Uncertainty Under Dataset Shift," 2019, arXiv:1906.02530.

[2] J. Quiñonero-Candela, M. Sugiyama, N. D. Lawrence and A. Schwaighofer, Dataset shift in machine learning, MIT Press, 2009.

[3] J. Gama, I. Žliobaitė, A. Bifet, M. Pechenizkiy and H. Bouchachia, "A Survey on Concept Drift Adaptation," *ACM Computing Surveys (CSUR)*, vol. 46, no. 4, pp. 1-37, 2014.

[4] D. Amodei, C. Olah, J. Steinhardt, P. Christiano, J. Schulman and D. Mané, Concrete problems in AI safety, arXiv:1606.06565, 2016.

[5] P. Gulati, A. Kumar and R. Bhardwaj, "Impact of Covid19 on electricity load in Haryana (India)," *International Journal of Energy Research*, vol. 45, no. 2, pp. 3397-3409, 2020.

[6] S. Nižetić , "Impact of coronavirus (COVID-19) pandemic on air transport mobility, energy, and environment: A case study," *International Journal of Energy Research*, vol. 44, no. 13, pp. 10953-10961, 2020.

[7] C. Isaksson, "The Impact of Coronavirus on Machine Learning Models," phData, 3 June 2020. [Online]. Available: https://www.phdata.io/blog/the-impact-of-covid-19-on-machine-learning-models/. [Accessed 30 Aug 2020].

[8] T. Fields, G. Hsieh and J. Chenou, "Increasing the Robustness of Deep Leaning with Coloured Noise Augmentation," in *The 2019 International Conference on Security and Management (SAM'19)*, Las Vegas, 2019.

[9] D. Dua and C. Graff, "UCI Machine Learning Repository," [Online]. Available: http://archive.ics.uci.edu/ml. [Accessed 12 Mar. 2019].

[10] X. Liang, T. Zou, B. Guo, S. Li, H. Zhang, S. Zhang and H. Huang, "Assessing Beijing's PM2.5 pollution: severity, weather impact, APEC and winter heating," in *Proceedings of the Royal Society A: Mathematical, Physical and Engineering Sciences* , Royal Society of London, 2015.

[11] "The United States Department of Justice: Spam," 24 Jan. 2020. [Online]. Available: https://www.justice.gov/doj/spam. [Accessed 28 Apr. 2020].

[12] "Spam Laws: Pulling the Plug on Internet Imposters," [Online]. Available: https://www.spamlaws.com/spam-stats.html. [Accessed 22 Jun. 2020].

[13] phishingbox, "The Phishing Box News Blog: Verizon Data Breach Investigation (DBIR) - 2019," 28 Aug 2019. [Online]. Available: https://www.phishingbox.com/assets/files/images/Verizon-Data-Breach-Investigations-Report-DBIR-2019.pdf. [Accessed 22 Jun. 2020].

[14] E. Dada, J. Bassi, H. Chiroma, S. M. Abdulhamid, A. Adetunmbi and O. Ajibuwa, "Machine learning for email spam filtering: review, approaches and open research problems," *Heliyon,* vol. 5, no. 6, 2019.

[15] J. Devlin, M.-W. Chang, K. Lee and K. Toutanova, "BERT: Pre-training of Deep Bidirectional Transformers for Language Understanding," arXiv:1810.04805, 2019.

[16] T. S. Guzella, Walmir and . M. Caminhas, "A review of machine learning approaches to Spam filtering," *Expert Systems with Applications,* vol. 36, no. 7, 2009.

[17] G. G. Mujtaba, L. Shuib, R. G. Raj, N. Majeed and M. A. Al-Garadi, "Email Classification Research Trends: Review and Open Issues," *IEEE Access,* vol. 5, pp. 9044-9064, 2017.

[18] V. Metsis, I. Androutsopoulos and G. Paliouras, "Spam Filtering with Naive Bayes - Which Naive Bayes?," in *Proceedings of the 3rd Conference on Email and Anti-Spam (CEAS 2006),* 2006..

[19] J. Wang, "An adaptive nearest neighbor algorithm for classification," *International Conference on Machine Learning and Cybernetics,* vol. 5, pp. 3069-3074, 2005.

[20] N. K. Sourati and A. S. Aski, "Proposed efficient algorithm to filter spam using machine learning techniques," *Pacific Science Review A: Natural Science and Engineering,* vol. 18, no. 2, pp. 145-149, 2016.

[21] A. Khan, B. Baharudin, L. H. Lee and K. Khan, "A review of machine learning algorithms for identification and classification of non-functional requirements," *Journal of Advances in Information Technology,* vol. 1, 2010.

[22] S. Minaee , N. Kalchbrenner , E. Cambria , N. Nikzad , M. Chenaghlu and . J. Gao, "Deep Learning--based Text Classification: A Comprehensive Review.," *CM Computing Surveys (CSUR),* vol. 54, no. 3, pp. 1-40., 2021.

[23] J. R. Méndez, T. R. Cotos-Yañez and D. Ruano-Ordás, "A new semantic-based feature selection method for spam filtering," *Applied Soft Computing,* vol. 76, pp. 89-104, 2019.

[24] M. Hearst, "Untangling text data mining," in *Proceedings of the 37th annual meeting of the Association for Computational Linguistics on Computational Linguistics*, 1999, doi: 10.3115/1034678.1034679.

[25] D. Jedamski, "NLP with Python for Machine Learning Essential Training," Linkedin, [Online]. Available: https://linkedin.com/learning/nlp-with-python-for-machine-learning-essential-training. [Accessed 21 Oct. 2019].

[26] M. Honnibal, I. Montani, S. Van Landeghem and A. Boyd, "SpaCy : Natural language understanding with Bloom embeddings, convolutional neural networks and incremental parsing," 2017.

[27] S. Bird, E. Loper and E. Klein , Natural Language Processing with Python, O'Reilly Media Inc., 2009.

[28] T. Walkowiak, S. Datko and H. Maciejewski, "Bag-of-Words, Bag-of-Topics and Word-to-Vec Based Subject Classification of Text Documents in Polish - A Comparative Study," in *International Conference on Dependability and Complex Systems*, 2018.

[29] Sammut, C; Webb, G. I., Encyclopedia of Machine Learning, Springer Science & Business Media, 2011.

[30] T. Mikolov, I. Sutskever, K. Chen, G. Corrado and J. Dean, "Distributed Representations of Words and Phrases and their Compositionality," in *Advances in neural information processing systems*, 2013.

[31] J. Pennington, R. Socher and C. Manning, "Global Vectors for Word Representation," 2014. [Online]. Available: https://nlp.stanford.edu/projects/glove/. [Accessed 15 Aug. 2019].

[32] E. Saravia and S. Poria, "Modern Deep Learning Techniques Applied to Natural Language Processing," [Online]. Available: https://nlpoverview.com/index.html#1. [Accessed 21 Feb. 2020].

[33] T. Young, D. Hazarika, S. Poria and E. Cambria, "Recent Trends in Deep Learning Based Natural Language Processing," 2018, arXiv:1708.02709v8.

[34] E. Grave, P. Bojanowski, P. Gupta, A. Joulin and T. Mikolov, "Learning Word Vectors for 157 Languages," in *Proceedings of the International Conference on Language Resources and Evaluation (LREC 2018)*, 2018.

[35] Google, "Machine Learning Crash Course," [Online]. Available: https://developers.google.com/machine-learning/crash-

course/embeddings/translating-to-a-lower-dimensional-space. [Accessed 17 Jun. 2020].

[36] A. Vaswani , N. Shazeer, N. Parmar, J. Uszkoreit, L. Jones, A. N. Gomez, L. Kaiser and I. Polosukhin, "Attention is All you Need," in *Advances in Neural Information Processing Systems*, 2017.

[37] J. Cheng, L. Dong and M. Lapata, "Long Short-Term Memory-Networks for Machine Reading," in *Conference on Empirical Methods in Natural Language Processing*, 2016, arXiv:1601.06733v7.

[38] A. Parikh, O. Täckström, D. Das and J. Uszkoreit, "A Decomposable Attention Model for Natural Language Inference," in *Proceedings of the 2016 Conference on Empirical Methods in Natural Language Processing*, 2016, arXiv:1606.01933v2.

[39] Z. Yang, D. Yang, C. Dyer, X. He, A. Smola and E. Hovy, "Hierarchical Attention Networks for Document Classification," in *Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies*, San Diego, 2016.

[40] P. Nayak, "Google Blog: Understanding searches better than ever before," Google, [Online]. Available: https://blog.google/products/search/search-language-understanding-bert/. [Accessed 14 Jul. 2020].

[41] L. Breiman, "Random forests," *Machine Learning*, vol. 45, no. 1, pp. 5-32, 2001.

[42] J. Brownlee, Long short-term memory networks with python: develop sequence prediction models with deep learning, Machine Learning Mastery, 2017.

[43] K. O'Shea and R. Nash. , "An introduction to convolutional neural networks," 2016, arXiv:1511.08458.

[44] pathmind, "A.I. Wiki A Beginner's Guide to Neural Networks and Deep Learning," [Online]. Available: https://wiki.pathmind.com/neural-network. [Accessed 15 Feb 2020].

[45] E. Alpaydin, Introduction to Machine Learning, 4th ed., MIT Press, 2020.

[46] A. Rakhlin, "Convolutional Neural Networks for Sentence Classification in Keras," 2016. [Online]. Available: https://github.com/alexander-rakhlin/CNN-for-Sentence-Classification-in-Keras. [Accessed 23 Jun. 2020].

[47] N. Kalchbrenner, E. Grefenstette and P. Blunso, "A Convolutional Neural Network for Modelling Sentences," in *Proceedings of the 52nd Annual Meeting of the Association for Computational Linguistics*, 2014, arXiv:1404.2188.

[48] Y. Wenpeng, K. Kann, M. Yu and S. Hinrich, "Comparative study of CNN and RNN for natural language processing," 2016, arXiv:1702.01923.

[49] J. G. Moreno-Torres, T. Raeder, R. Alaiz-Rodríguez, N. V. Chawla and F. Herrera, "A unifying view on dataset shift in classification," *Pattern recognition,* vol. 45, no. 1, pp. 521-530, 2012.

[50] I. Žliobaitė, M. Pechenizkiy and J. Gama, "An Overview of Concept Drift Applications," *Big Data Analysis: New Algorithms for a New Society. SBD,* vol. 16, pp. 91-114, 2016.

[51] H. Shimodaira, "Improving predictive inference under covariate shift by weighting the log-likelihood function," *Journal of statistical planning and inference,* vol. 90, no. 2, pp. 227-244, 2000.

[52] R. Elwelln and R. Polikar, "Incremental learning of concept drift in nonstationary environments," *IEEE Transactions on Neural Networks,* vol. 22, no. 10, pp. 1517-1531, 2011.

[53] M. Sugiyama and M. Kawanabe, Machine Learning in Non-Stationary Environments: Introduction to Covariate Shift Adaptation, Cambridge: MIT Press, 2012.

[54] H. Wang and Z. Abraham, "Concept drift detection for streaming data," in *2015 International Joint Conference on Neural Networks (IJCNN),* arXiv:1504.01044.

[55] C. Ahmed, N. Lachiche, C. Charnay and A. Braud, "Dataset Shift in a Real-Life Dataset," in *ECML-PKDD Workshop (LMCE),* 2014.

[56] J. Lu, A. Liu, F. Dong, F. Gu, J. Gama and G. Zhang, "Learning under Concept Drift: A Review," *IEEE Transactions on Knowledge and Data Engineering,* vol. 31, no. 12, pp. 2346-2363, 2018, doi: 10.1109/TKDE.2018.2876857.

[57] A. Pesaranghade, H. L. Viktor and E. Paquet, "McDiarmid Drift Detection Methods for Evolving Data Streams," *International Joint Conference on Neural Networks (IJCNN),* pp. 1-9, 2018.

[58] E. Page, "Continous inspection schemes," *Biometrika,* vol. 41, no. 1/2, pp. 100-115, 1954.

[59] J. Gama, P. Medas, G. Castillo and P. Rodrigues, *Learning with Drift Detection,* Springer, 2003, pp. 286-295.

[60] M. Beana-Garcia, J. D. Campo-Avila, R. Filalgo, A. Bifet, R. Gavalda and R. Morales-Bueno, "Early Drift Detection Method," *4th International workshop on knowledge discovery from data streams,* vol. 6, pp. 77-86, 2006.

[61] G. Ross, N. M. Adams, D. K. Tasoulis and D. J. Hand, "Exponentially weighted moving average charts for detecting concept drift," *Pattern Recognition Letters,* vol. 33, no. 2, pp. 191-198, 2012.

[62] R. Barros, D. R. Cabral, P. M. Gonçalves and S. G. Santos, "RDDM: Reactive drift detection method," *Expert Systems with Applications,* vol. 90, pp. 344-355, 2017.

[63] A. Bifet and R. Gavaldà, "Learning from Time-Changing Data with Adaptive Windowing," in *Proceedings of the Seventh SIAM International Conference on Data Mining*, Minneapolis, 2007.

[64] I. Žliobaitė, M. Pechenizkiy and J. Gama, "An Overview of Concept Drift Applications," *ACM Computing Surveys (CSUR),* vol. 46, 2014.

[65] R. Pears, S. Sakthithasan and Y. Koh, "Detecting concept change in dynamic data streams," *Machine Learning,* vol. 97, no. 3, p. 259–293, 2014.

[66] I. Frías-Blanco, J. d. Campo-Ávila, G. Ramos-Jiménez, R. Morales-Bueno, A. Ortiz-Díaz and Y. Caballero-Mota, "Online and Non-Parametric Drift Detection Methods Based on Hoeffding's Bounds," *IEEE Transactions on Knowledge and Data Engineering,* vol. 27, no. 3, pp. 810-823, 2015.

[67] A. Pesaranghader and H. L. Viktor, "Fast Hoeffding Drift Detection Method for Evolving Data Streams," *Machine Learning and Knowledge Discovery in Databases,* vol. 9852, pp. 96-111, 2016.

[68] Spam Assassin Project (2015) Spam Assassin Public, 2015. [Online]. Available: https://spamassassin.apache.org/publiccorpus/. [Accessed 18 Mar. 2019].

[69] V. Metsis, I. Androutsopoulos and G. Paliouras, "The Enron-Spam datasets," 19 Jun. 2006. [Online]. Available: http://www2.aueb.gr/users/ion/data/enron-spam/. [Accessed 12 Mar. 2019].

[70] EMC Services, Data Science and Big Data Analytics: Discovering, Analyzing, Visualizing, Presenting Data, Indianapolis, IN: John Wiley & Sons, Inc., 2015.

[71] O. Maimon and L. Rokach, Data Mining and Knowledge Discovery Handbook, Heidelberg Berlin: Springer-Verlag Publishing Company, Incorporated, 2010.

[72] "Email — An email and MIME handling package," [Online]. Available: https://docs.python.org/3/library/email.html. [Accessed 23 Feb. 2019].

[73] L. Richardson, "Beautifulsoup4 4.9.3," [Online]. Available: https://pypi.org/project/beautifulsoup4/. [Accessed 02 Dec. 2018].

[74] A. V. Aho, Algorithms for finding patterns in strings, Handbook of theoretical computer science, Cambridge, MA: MIT Press, 1991.

[75] G. Skinner, "RegExr: Learn, Build, & Test RegEx," [Online]. Available: https://regexr.com/. [Accessed 20 Mar. 2020].

[76] J. Lipovský, "urlextract 1.2.0," [Online]. Available: https://pypi.org/project/urlextract/. [Accessed 23 Feb. 2019].

[77] M. Honnibal and I. Montani, *Natural language understanding with Bloom embeddings, convolutional neural networks and incremental parsing*, 2017.

[78] "Machine-Learning SPAM Classifier Using SpaCy NLP Library & Scikit-Learn.," [Online]. Available: https://github.com/just-a-stream/ml-spam-filter/blob/master/THESPAMFILTER.ipynb. [Accessed 29 Aug. 20].

[79] I. Witten, E. Frank and M. Hall, Data Mining: Practical Machine Learning Tools and Techniques, 4th ed., San Francisco: Morgan Kaufmann Publishers Inc., 2011.

[80] A. Aghaebrahimian and M. Cieliebak, "Hyperparameter tuning for deep learning in natural language processing," in *4th Swiss Text Analytics Conference (SwissText 2019)*, 2019.

[81] P. Probst, M. N. Wright and A. Boulesteix, "Hyperparameters and tuning strategies for random forest," *Wiley Interdisciplinary Reviews: Data Mining and Knowledge Discovery*, vol. 9, no. 3, p. 1301, 2019.

[82] R. G. Mantovani, A. L. Rossi, J. Vanschoren, B. Bischl and A. C. Carvalho, "To tune or not to tune: recommending when to adjust SVM hyper-parameters via meta-learning," in *International Joint Conference on Neural Networks (IJCNN)*, 2015.

[83] N. Lavesson and P. Davidsson, "Quantifying the Impact of Learning," in *Proceedings of the 21st National Conference on Artificial Intelligence - Volume 1 (AAAI'06)*, 2006, doi: https://doi.org/10.1609/aaai.v33i01.33018223.

[84] N. N. Sjarif, "SMS spam message detection using term frequency-inverse document frequency and random forest algorithm," *Procedia Computer Science*, vol. 161, pp. 509-15, 2019.

[85] Z. Chen, C. Li and R. V. Sánchez, "Multi-layer neural network with deep belief network for gearbox fault diagnosis," *Journal of Vibroengineering*, vol. 17, no. 5, pp. 2379-2392, 2015.

[86] J. Ranganathan, N. Hedge, A. S. Irudayaraj and A. A. Tzacheva , "Automatic detection of emotions in Twitter data: a scalable decision tree classification method," in *Proceedings of the Workshop on Opinion Mining, Summarization and Diversification*, 2018, doi: 10.1145/3301020.3303751.

[87] W. Yin, K. Kann, M. Yu and H. Schütze, "Comparative study of CNN and RNN for natural language processing," 2017, arXiv:1702.01923 .

[88] P. Poomka, . P. Wattana and K. Nittaya , "SMS spam detection based on long short-term memory and gated recurrent unit," *International Journal of Future Computer and Communication*, vol. 8, no. 1, 2019.

[89] T. Zhang , T. Wu , F. Katiyar, K. Weinberger and Y. Artzi, "Revisiting Few-Sample BERT Fine-Tuning," in *ICLR*, 2021, arXiv:2006.05987.

[90] J. Dodge, G. Ilharco, R. Schwartz, A. Farhadi, H. Hajishirzi and N. Smith, "Fine-tuning pretrained language models: Weight initializations, data orders, and early stopping," 2020, arXiv:2002.06305.

[91] Q. Yaseen, "Spam Email Detection Using Deep Learning Techniques," *Procedia Computer Science 184*, pp. 853-858, 2021.

[92] J. Prateek, "Fine Tuning BERT forSpam Classification," [Online]. Available: https://github.com/prateekjoshi565/Fine-Tuning-BERT/blob/master/Fine_Tuning_BERT_for_Spam_Classification.ipynb. [Accessed 07 Aug. 2020].

[93] M. Jocquevial, "Udemy Course: Learn BERT - most powerful NLP algorithm by Google," [Online]. Available: https://www.udemy.com/course/bert-nlp-algorithm/learn/lecture/17347020?start=0#overview. [Accessed 20 Jul. 2020].

[94] N. Lakshmipathi, "IMDB Dataset of 50K Movie Reviews," [Online]. Available: https://www.kaggle.com/lakshmi25npathi/imdb-dataset-of-50k-movie-reviews. [Accessed 23 Jul. 2020].

[95] R. Socher, A. Perelygin, J. Wu, J. Chuang, C. D. Manning, A. Y. Ng and C. Potts, "Stanford Sentiment Treebank v2 (SST2)," in *Proceedings of the 2013 conference on empirical methods in natural language processing*, 2013.

[96] "Yelp Dataset A trove of reviews, businesses, users, tips, and check-in data!," Yelp Inc., [Online]. Available: https://www.kaggle.com/yelp-dataset/yelp-dataset. [Accessed 23 Jun. 2020].

[97] Consumer Reviews of Amazon Products, [Online]. Available: https://www.kaggle.com/datafiniti/consumer-reviews-of-amazon-products. [Accessed 23 Jun. 2020].

[98] . M. Mosbach, M. Andriushchenko and D. Klakow, "On the stability of fine-tuning bert: Misconceptions, explanations, and strong baselines," 2020, arXiv:2006.04884.

[99] D. Hendrycks, X. Liu, W. E. Dziedzic, R. Krishnan and D. Song, "Pretrained transformers improve out-of-distribution robustness," 2020, arXiv:2004.06100.

[100] "Welcome to Colaboratory," [Online]. Available: https://colab.research.google.com/notebooks/intro.ipynb. [Accessed 12 Jun. 2019].

[101] T. Carneiro, R. V. Medeiros Da NóBrega and T. Nepomuc, "Performance Analysis of Google Colaboratory as a Tool for Accelerating Deep Learning Applications," *IEEE Access,* vol. 6, pp. 61677-61685, 2018.

[102] G. Hsieh, T. L. Field, B. Kc and J. Yurko-Galvin, "Building a Cybersecurity Research and Experimentation Testbed," in *CSCI'18,* Las Vegas, 2018.

[103] "Anaconda Software Distribution Vers. 2-2.4.0," Anaconda Inc., 2020. [Online]. Available: https://docs.anaconda.com/. [Accessed 22 Feb. 2018].

[104] T. Kluyver, B. Ragan-Kelley, F. Pérez, B. Granger, M. Bussonnier, J. Frederic and K. Kelley, "Jupyter Notebooks – a publishing format for reproducible computational workflows," *F. Loizides & B. Schmidt, eds. Positioning and Power in Academic Publishing: Players, Agents and Agendas,* pp. 87-90, 2016.

[105] "Jupyter," [Online]. Available: https://jupyter.org/. [Accessed 12 Jun. 2019].

[106] "Spyder-documentation," [Online]. Available: https://docs.spyder-ide.org/current/index.html. [Accessed 12 Jun. 2018].

[107] L. Buitinck , G. Louppe, M. Blondel, F. Pedregosa, A. Mueller, O. Grisel and V. Niculae , "API design for machine learning software: experiences from the scikit-learn:experiences from the scikit-learn project," 2013, arXiv:1309.0238.

[108] Chollet, Francois; and others, "Keras," 2015. [Online]. Available: https://github.com/fchollet/keras. [Accessed 23 Feb. 2019].

[109] M. Abadi , A. Agarwal, P. Barham, E. Brevdo, Z. Chen, C. Greg, J. Dean, M. Devin and S. Ghemawat, "TensorFlow: Large-Scale Machine Learning on Heterogeneous Systems," 2016, arXiv:1603.04467.

[110] T. Wolf, L. Debut , V. Sanh, J. Chaumond , C. Delangue, A. Moi, P. Cistac and T. Rault, "HuggingFace's Transformers: State-of-the-art Natural Language Processing," 2020, arXiv:1910.03771.

[111] W. McKinney and Others, "Data structures for statistical computing in python," in *Proceedings of the 9th Python in Science Conference*, 2010.

[112] J. D. Hunter, "Matplotlib: A 2D graphics environment," *Computing in Science & Engineering*, vol. 9, no. 3, pp. 90-95, 2007.

[113] A. Paszke, S. Gross, F. Massa, A. Lerer, J. Bradbury, G. Chanan and T. Killeen, "PyTorch: An Imperative Style, High-Performance Deep Learning Library," *Advances in neural information processing systems 32 (2019): 8026-8037.*, p. 8024–8035, 2019.

[114] A. Géron, "Hands-On Machine Learning with Scikit-Learn, Keras, and TensorFlow, 2nd Edition," O'Reilly, 2020.

[115] A. Mistry and V. Gadhave, "Udemy Course: Learn Natural Language Processing (NLP) in Python with 8 Projects," [Online]. Available: https://www.udemy.com/course/complete-natural-language-processing-nlp-with-spacy-nltk/learn/lecture/20072196/start. [Accessed 17 Jun. 2020].

[116] J. Nothman, H. Qin and R. Yurchak, "Stop word lists in free open-source software packages," in *Proceedings of Workshop for NLP Open Source Software (NLP-OSS)*, 2018.

[117] A. Schofield, M. Magnusson and D. Mimno, "Pulling out the stops: Rethinking stopword removal for topic models," in *Proceedings of the 15th Conference of the European Chapter of the Association for Computational Linguistics*, 2017.

[118] G. Webb, R. Hyde, C. Hong, H. Nguyen and F. Petitjean, "Characterizing concept drift," *Data Mining and Knowledge Discovery*, vol. 30, pp. 964-994, 2016.

[119] C. Cortes and V. Vapnik, "Support-vector networks," *Machine learning*, no. 3, pp. 273-297, 1995.

[120] P. Azunre, Transfer Learning for Natural Language Processing, Shelter Island: Manning Publications, 2021.

[121] R. N. Gemaque, R. Noronha, A. F. Josuá Cost, R. Giusti and . E. . M. Dos Santos, "An overview of unsupervised drift detection methods.," *Wiley Interdisciplinary Reviews: Data Mining and Knowledge Discovery* , vol. 10, no. 6, p. e1381, 2020.

[122] N. Zumel, J. Mount, J. Howard and R. Thomas, Evaluating a Classificaiton Model with a Spam Filter, Shelter Island, NY: Manning Publications Co., 2020.

[123] A. Ghourabi, M. Mahmood and Q. Alzubi, "A Hybrid CNN-LSTM Model for SMS Spam Detection in Arabic and English Messages," *Future Internet,* vol. 12, no. 9, 2020.

[124] N. Sutta, Z. Liu and X. Zhang, "A Study of Machine Learning Algorithms on Email Spam Classification," in *CATA 2020*, 2020.

[125] R. Barros and S. Santos, "A large-scale comparison of concept drift detectors," *Information Sciences,* vol. 451, pp. 348-370 , 2018.

[126] R. Gopalakrishnan and A. Venkateswarlu, Machine Learning for Mobile: Practical guide to building intelligent mobile applications powered by machine learning, Packt Publishing Ltd, 2018.

[127] E. D. Liddy, "Natural Language Processing. In Encyclopedia of Library and Information Science, 2nd Ed," Marcel Decker, Inc., NY, 2001.

[128] S. Thrun and L. Pratt, "Learning to learn: Introduction and overview," in *Learning to learn*, Boston, 1998.

[129] McKinsey Global Institue (2018) Notes from the AI Frontier, "Insights from hundreds of use Cases," April 2018. [Online]. Available: https://www.mckinsey.com/featured-insights/artificial-intelligence/notes-from-the-ai-frontier-applications-and-value-of-deep-learning. [Accessed 23 Feb. 2019].

[130] M. E. Peters , M. Neumann , M. Iyyer , M. Gardner , C. Clark, K. Lee and L. Zettlemoyer, "Deep contextualized word representations," 2018, arXiv:1802.05365 .

[131] N. K. Kasabov, Time-Space, Spiking Neural Networks and Brain-Inspired Artificial Intelligence, Springer Series on Bio- and Neurosystems, 2019.

[132] A. Ezen-Can, "A Comparison of LSTM and BERT for Small Corpus," 2020, arXiv:2009.05451.

[133] M. Guillame-Bert, S. Bruch, J. Gordon and J. Pfeifer, "Introducing TensorFlow Decision Forest," 27 May 2021. [Online]. Available: https://blog.tensorflow.org/2021/05/introducing-tensorflow-decision-forests.html. [Accessed 18 Jul. 2021].