University of Technology, Sydney

Faculty of Engineering and Information Technology

NATURAL LANGUAGE PROCESSING FOR DOCUMENT ANALYSIS AND OUTCOME PREDICTION

by

Michael Garbuz

Student Number: 11995178

Project Number: S19-028

Major: Software Engineering

Supervisor: Massimo Piccardi

A 6 Credit Point Project submitted in partial fulfilment of the requirement for the Degree of Bachelor of Engineering

22 June 2020

NATURAL LANGUAGE PROCESSING FOR DOCUMENT ANALYSIS AND OUTCOME PREDICTION - (6CP)
Michael Garbuz – S19-028

Supervisor: Mr. Massimo Piccardi
Major: Software Engineering

To combine topic modelling with text classification, and build a tool that feeds the discovered topics into the classifier and assess the accuracy of classification.

Natural language processing has been evolving rapidly over the last 20 years, with developments around topic modelling and text classification, however attempts to bridge the two into an effective package or tool have been scarce. Independently, topic modelling and classification make up a majority of the current natural language processing world and are used in revolutionary tools such as Apple’s Siri, Google’s Assistant and Samsung’s Bixby to name a few. To effectively utilise these two aspects of natural language processing requires a deep understanding in both areas, delaying the time taken to produce results and see the models in action.

Two common algorithms used to create a pipeline between topic modelling and classification are Latent Dirichlet Allocation and XGBoost respectively. Currently, there are few clean and efficient implementations that make use of these two algorithms and that provide an understanding of how a change in parameters effects the accuracy of each model.

The intention of this study is to create a more accurate version of the current pipeline that aims to bridge topic modelling with classification through the use of document analysis and labelling. Through the comparison of a variety of topic modelling techniques, this study aims to identify the most impactful technique for topic modelling for the given data set, and use that technique as the input for a text classifier with the most adequate parameters to provide the highest degree of accuracy, and ease of implementation.

Acknowledgements

Firstly, I’d like to acknowledge the University of Technology Sydney (UTS) for providing access to a plethora of scholarly articles and allowing me to use them in the context of this study.

I’d also like to thank my UTS supervisor, Massimo Piccardi, for his continuous help and support throughout this study and for always challenging me to be the best that I can be. He was always there to ensure that my work was of the highest calibre and was continuously pushing me to succeed.

To my friends and family, thank you for dealing with me and my stress throughout the duration of my writing, as well as being the supportive group that you have been for the past 6.5 years of my university degree, I really appreciate it.

Finally, to all those reading this paper, thank you for taking your time to either discover or further your knowledge of Natural Language Processing, and I hope that you can take this knowledge and share it with others in the future.

Introduction

“The use of Machine Learning (ML), and in particular, artificial neural networks (ANN), in engineering applications has increased dramatically over the last years” (Reich and Barai, p. 257). Reich and Barai made this statement in 1998, 21 years ago, and yet the sentiment remains the same. Machine Learning, Deep Learning (DL), Artificial Intelligence (AI) and its branches, are responsible for drastic developments in the social world, from virtual personal assistants like Siri and Bixby, to predictive suggestions on social media platforms.

Natural Language Processing (NLP) “is the subfield of computer science concerned with using computational techniques to learn, understand, and produce human language content.” (Hirschberg et al. 2015 para. 2)

NLP, also known as Computational Linguistics, has grown tremendously over the past 20 years, and can now be seen in the world’s most common products such as Apple’s Siri, Amazon’s Alexa, and Google’s Assistant. The applications for NLP span across Named-Entity Recognition and Sentiment Analysis, all the way to Summarisation, Machine Translation, Topic Modelling, Dialogue Systems and many more. Though there are many broad applications of NLP, this study will look at the Topic Modelling and Text Classification implementations of NLP.

This study makes use of the Latent Dirichlet Allocation (LDA) algorithm, one of many topic modelling algorithms out there. LDA is built on the foundations of Probabilistic Graphical Models, which I will explore further in this study. The data that is made available for the purpose of this study has been provided by Amazon Web Services (AWS) and has been pre-classified by experts in the field that the documents pertain to, allowing a curated data-set that can be run through a Topic Modelling algorithm, and then further through a Classification Model.

Topic Modelling itself aims to identify the main topics of a given collection of documents, or corpus, which provides an overview of its composition and purpose. This is done by taking a single document at a time, and converting it to a bag-of-words, which resembles the entire document as a bunch of words disregarding punctuation, grammar and the order of the words in the document. Simultaneously, Topic Modelling identifies the various proportions of these topics in each document in the corpus. These proportions, also known as Topic Vectors, can be used to categorise, classify, or cluster, the individual documents and organise the corpus. Approaching a topic modelling problem using an algorithm like LDA or using a Term Frequency - Inverse Document Frequency (tf-idf) approach, will provide the aforementioned topic vector or bag of words frequency respectively, but will stop there, and will need to be inferred by the human eye to understand what to classify the documents as.

There is currently no algorithm or single process which can accurately identify the content of a corpus, and further classify each document within the corpus, making it difficult to efficiently and effectively group and label documents based on their content. Topic modelling is a good start; however, it needs to be combined with another NLP approach, and that is Text Classification.

As stated by Shanglun (2018), "There are several NLP classification algorithms that have been applied to various problems in NLP. For example, naive Bayes have been used in various spam detection algorithms, and support vector machines (SVM) have been used to classify texts such as progress notes at healthcare institutions." The classification algorithm that is used in this study is the Extreme Gradient Boosting (XGBoost) algorithm. There are a number of ways to utilise both LDA and XGBoost, some of which will be later identified in this study, however the overall approach is to use the output of the topic modelling, whether it be topic vectors or tf-idf vectors, and feed that output into the text classifier so that a label can be applied to the various documents in the corpus.

Given the lack of a unified tool, the development undertaken in this study will fill a gap in the NLP tools currently out there, by providing a tool that can extract the topics of a corpus, and with some customisation, label that data based on the topics. The tool that is referenced throughout this paper can easily by customised by changing the data input in the form of a comma-separated value (CSV) file.

This study investigates whether or not a trained, algorithm-based tool can mimic, or even improve the decisions that a human panel would make. This new tool will enable a team to free-up their time in assessing applicants and put more time and effort into training successful applicants, as well as determining the common pitfalls for those who are not successful through data provided by the tool.

The general use case for this report is to extract topics from a set of documents, and then classify those documents into user specified groups. For the purpose of this report and to utilise real-world data, AWS has specified a use case of *'enabling a team to free-up their time in assessing applicants and put more time and effort into training successful applicants, as well as determining the common pitfalls for those who are not successful through data provided by the tool'*, meaning the classifications for this study are 'successful' and 'unsuccessful'.

The tool aims to automate the nomination, vetting and graduation of Bar Raisers in Training at AWS. A Bar Raiser is an unbiased member of the hiring team who is responsible for making sure that any biases that the hiring manager or other members of the team have during the hiring process, are mitigated. Bar Raisers are of crucial importance for AWS and Amazon as a whole, as they have the power to veto a hiring decision. This process ensures that AWS continues to 'raise the bar' with every hire, making sure that the individual being hired is better than at least 50% of other employees in the role that is being hired for.

This thesis is divided into 5 chapters: The first chapter, Literature Review, will discuss in more detail the various concepts that are used in NLP and delve into some of the mathematics behind the algorithms, as well as various training techniques. Chapter two, Methodology, outlines the development work carried out in this project and describes the problems encountered, as well as the design choices made. The third chapter, Results, illustrates the results for each development approach, whereas the fourth chapter, Discussions, provides meaning to all the previously stated results, interpretations of the data, and speaks on the limitations of the research. Finally, chapter five, Conclusions, will sum up the entirety of this report and highlight the significance of the work undertaken, as well as what can be done next and where the research can be taken.

Literature Review

Natural Language Processing

As noted by Gunning and Ghosh (2019, Chapter 1), "NLP is an area that overlaps with others. It has emerged from fields such as artificial intelligence, linguistics, formal languages, and compilers. With the advancement of computing technologies and the increased availability of data, the way natural language is being processed has changed. Previously, a traditional rule-based system was used for computations. Today, computations on natural language are being done using machine learning and deep learning techniques."

Gunning and Ghosh (2019, Chapter 1) then discuss NLP through a series of exercises, but crucially discuss the differences between NLP and other analysis types, i.e., text analytics. Taking the comparison of text analytics and NLP into consideration, we can identify text analytics as "The art of extracting useful insights from any given text data" (Natural Language Processing Fundamentals, 2019), whereas NLP is not restricted to text data, as voice recognition and analysis are under the umbrella of NLP.

NLP tends to be composed of two different types, Natural Language Generation (NLG) and Natural Language Understanding (NLU). NLP works on a combination of the two types, as an NLU approach is initially taken so that an inanimate object with computing power is aware of human language. This understanding is then combined with an NLG approach so that the inanimate object may generate its thoughts in a language that humans can understand.

NLU will be used to determine the historic '*successful*' and *'unsuccessful'* decisions that humans make and why, and then NLG will be used to verbalise the predicted decisions, after which a comparison will be made between the NLG output and a human panel's decisions.

Now that the concept of NLP has been established, it's important to understand the process and methodology of developing an ML/NLP project. Gunning and Ghosh (2019, Chapter 1) very simply put the different steps of undertaking an NLP project in a diagram pictured below:

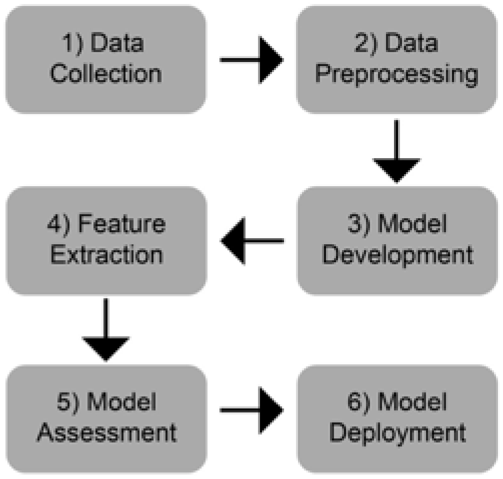


Figure 1: Gunning and Ghosh (2019, Chapter 1)

The image above is self-explanatory, however there are some parts missing. As noted by Andrew Ferlitsch of Google, planning and quality assurance are 2 steps of the machine learning development cycle that should be present throughout the entire cycle. His explanation of the machine learning lifecycle can be seen below and should be compared to the methodology provided by Gunning and Ghosh (2019, Chapter 1).

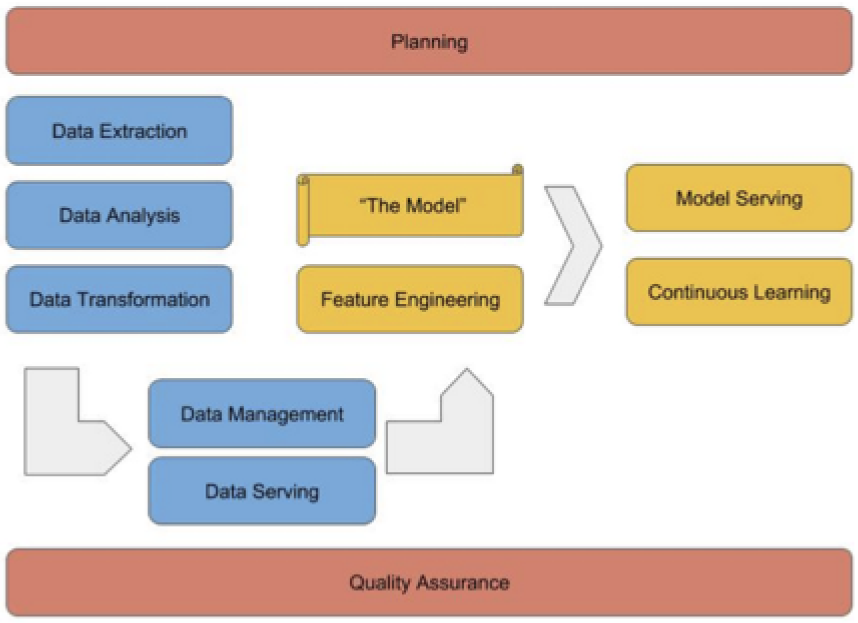


Figure 2: Ferlitsch (2019)

Although there are steps that are similar between the two different approaches, it's important to note that the crux of both approaches is the same. This includes:

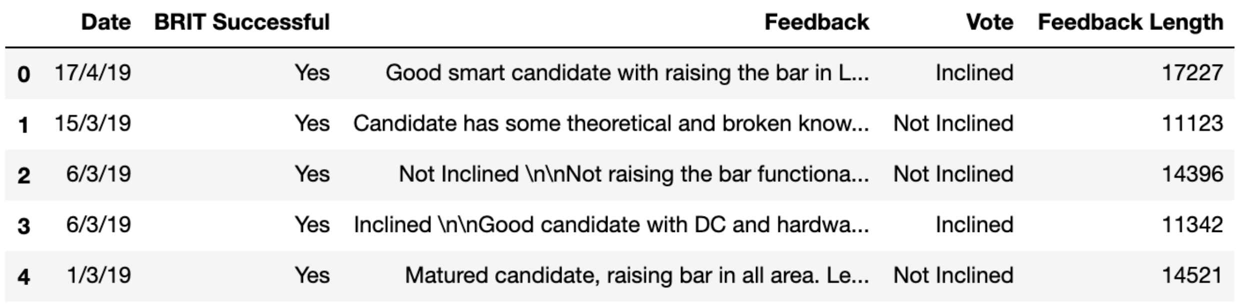
- Data Extraction → Data Collection
- Data Analysis/Transformation/Management/Serving → Data Pre-processing
- The Model → Model Development
- Feature Engineering → Feature extraction

The steps after feature engineering vary in different models, as there may be parts of continuous learning implementation and repetition, or it may be an initial model assessment followed by a model deploy, as seen in the first image.

Topic Modelling

As previously mentioned, topic modelling is a strand of natural language processing, and "in essence, … is an unsupervised model that learns the set of underlying topics (in terms of word distributions) for a set of documents and each document's affinities to these topics" (Nikolenko et al. 2017 para. 2). Over the years, there have been many developments in the field of topic modelling, all trying to answer the problem of “modelling text corpora and other collections of discrete data” (David et al. 2003 p. 993).

There are a number of concepts to discuss to assist in the understanding of topic modelling. Starting with the corpus, which is a collection of documents. Documents can consist of any body of text, ranging from emails, resumes, newspapers etc. For example, a corpus can look like the following, where each row is considered a new document, and the overall table, or dataframe, is the corpus used for topic modelling.



	Date	BRIT Successful	Feedback	Vote	Feedback Length
0	17/4/19	Yes	Good smart candidate with raising the bar in L...	Inclined	17227
1	15/3/19	Yes	Candidate has some theoretical and broken know...	Not Inclined	11123
2	6/3/19	Yes	Not Inclined \n\nNot raising the bar functiona...	Not Inclined	14396
3	6/3/19	Yes	Inclined \n\nGood candidate with DC and hardwa...	Inclined	11342
4	1/3/19	Yes	Matured candidate, raising bar in all area. Le...	Not Inclined	14521

Figure 3: Example Corpus as a Data Frame

For topic modelling to be effective, each document needs to be broken up so that each word can be examined independently, therefore, we can say that a document is a Bag of Words (BoW) which then is fed as the input for topic modelling. For example, if we focus on the Feedback column in the above corpus, and take the third row from the top, excluding new line characters, we have:

‘Not inclined Not raising the bar functionally’

The BoW representation, in JSON format, of the above document would be:

BoW = {“Not”: 2, “inclined”:1, “raising”: 1, “the”: 1, “bar”:1, “functionally”:1}

As can be seen, the BoW representation of the document represents words that occur in the document as a key-value pair, with the key being the word, and the value being the number of times the word occurs within the document. Finally, all the words present in the BoW constitutes the vocabulary of the corpus, as the number of words within the corpus is fixed.

Liu et al. (2016) identify that the BoW approach may have a large dimensionality of word space as it only reflects the words of the original texts. To be able to make some sort of inference from the corpus, common themes that are prevalent throughout the documents are important to discover. These themes are also referred to as *topics*, and as previously stated, the overall aim of topic modelling is to discover topics that are present throughout the corpus by doing some analysis on the BoW. Liu et al. (2016) then continue to define "the key idea behind topic modeling is that documents show multiple topics, and therefore the key question of topic modeling is how to discover a topic distribution over each document and a word distribution over each topic…" (Lie et al. 2016, para. 15).

To understand where topic modelling has arrived to today, it's important to look at the progress that has been made over the past 20 years, starting with the popular tf-idf approach, leading up to the implementation of Latent Semantic Indexing (LSI), all the way to the algorithm used in this study, Latent Dirichlet Allocation (LDA).

Term Frequency and Inverse Document Frequency (tf-idf)

One of the most implemented manipulations that can be done on the BoW is the tf-idf approach. This approach looks at scoring the relevance of words throughout a corpus, and it is broken down into 2 components, term frequency and inverse document frequency.

Term frequency can be thought of as the number of times a word appears in the document, as a fraction of the total words in the document. Each document within a corpus will have its own term frequency. Maklin (2019) identifies the mathematical formula behind term frequency as:

$$tf_{i,j} = \frac{n_{i,j}}{\Sigma_k n_{k,j}}$$

Equation 1: Term Frequency Equation

Where $tf_{i,j}$ is the term frequency of the i-th word in document j, and $n_{i,j}$ represents its count. Maklin (2019) then goes on to provide a Python example as to how this would be developed per document, which can be seen below:

```
def computeTF(wordDict, bagOfWords):
    tfDict = {}
    bagOfWordsCount = len(bagOfWords)
    for word, count in wordDict.items():
        tfDict[word] = count / float(bagOfWordsCount)
    return tfDict
```

Where *wordDict* is a dictionary of words and their corresponding number of occurrences within each document in the corpus, and *bagOfWords* is a list of each word in the document. The term frequency should be computed for each document in the corpus.

Once the term frequency of each document has been calculated, the inverse document frequency for the entire corpus should be calculated. Malkin (2019) continues to explain that the inverse data frequency can be calculated by taking the "log of the number of documents divided by the number of documents that contain the word *w*. [Inverse data frequency] determines the weight of rare words across all documents in the corpus" (Malkin, 2019). This can be represented by the following equation:

$$idf(w) = \log log\left(\frac{N}{df_i}\right)$$

Equation 2: Inverse Document Frequency Equation

The inverse document frequency can be implemented in Python as follows:

```
def computeIDF(documents):
    import math
    N = len(documents)

    idfDict = dict.fromkeys(documents[0].keys(), 0)
    for document in documents:
        for word, val in document.items():
            if val > 0:
                idfDict[word] += 1

    for word, val in idfDict.items():
        idfDict[word] = math.log(N / float(val))
    return idfDict
```

With the term frequency and the inverse document frequency now calculated, the final step is to multiply them together which looks like this:

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

Equation 3: Term Frequency - Inverse Data Frequency Equation

Where the tf-idf value of the *i*-th word in document *j* is denoted as $w_{i,j}$, and in Python can be implemented with the following function:

```
def computeTFIDF(tfBagOfWords, idfs):
    tfidf = {}
    for word, val in tfBagOfWords.items():
        tfidf[word] = val * idfs[word]
    return tfidf
```

This function is to be run on each document in the corpus, to retrieve the specific tf-idf values of each word in its corresponding document.

Although tf-idf can be implemented from scratch using the aforementioned functions, there are a number of Python libraries that come with their own tf-idf functionality. One such library, which was used for this study, is known as *scikit-learn*, or *sklearn* for short. Within *sklearn*, there is a function called *TfidfVectorizer()*[1] which takes away the hassle of custom function definitions, however will provide different results compared to the manual approach, due to sklearn using a smoothed version of inverse document frequency and various other optimisations.

[1] https://scikit-learn.org/stable/modules/generated/sklearn.feature_extraction.text.TfidfVectorizer.html

Although tf-idf is a more intuitive way of determining the composition of documents, as it is essentially a reweighted count of the unique words in a document, a drawback of this approach is that the tf-idf matrices are very high dimensional, sparse and noisy. One way of correcting this is to use a technique called singular value decomposition (SVD), which is the foundation of Latent Semantic Analysis (LSA) or Latent Semantic Indexing (LSI).

Latent Semantic Analysis / Indexing (LSA / LSI)

LSA was introduced in the 1980s to answer questions around inferring the meaning of texts without having a direct mention to the meaning within the text. This approach was then later introduced into the world of NLP by Jerome Bellegarda in 2005. "In LSA, term frequency matrix X is first subjected to a matrix operation called singular value decomposition (SVD). SVD decomposes X into term eigenvectors U, document eigenvectors V, and singular values Σ : " (Evangelopoulos 2013, para. 6)

$$X = U\Sigma V^T$$

Equation 4: Decomposed Term Frequency Matrix using Singular Value Decomposition

Since the objective of LSA is to reduce the dimension for classification, allowing a more defined and coherent matrix to run inference on, SVD is applied on the tf-idf matrix to achieve this. The concept behind implementing SVD on a tf-idf matrix, is to "find the most valuable information and use a lower dimension T to represent the same matrix" (Ma, 2018). The result of conducting SVD on a term frequency matrix X, reproduces X using a space of latent semantic dimensions.

Evangelopoulos (2013) then states that "The relative importance of these dimensions in terms of being able to explain variability in term‐document occurrences is quantified in the r elements of the diagonal matrix Σ, $r \leq min(t,\ d)$, called singular values, which are the square roots of common eigenvalues in the simultaneous principal component analysis of terms as variables (with documents as observations) and documents as variables (with terms as observations). " (Evangelopoulos 2013, para. 7)

For more detailed information about the algorithms behind LSA and the use cases, the full paper published by Evangelopoulos should be referred to, which can be found in the references section of this study.

Latent Dirichlet Allocation (LDA)

As noted by Ma (2018), LSA and LDA are quite similar in the respect that both accept BoW as the input to the models, however, LSA is more focused on reducing the dimension of the matrix through SVD as previously mentioned, meanwhile LDA is focused on solving topic modelling problems. For the purposes of this study, LDA has been chosen to identify the composition of documents and to discover whether or not there are common topics present in documents within a corpus.

LDA is a classic generative probabilistic model of a corpus with an underlying Dirichlet distribution. It can be thought of in a similar way to LSA from a matrix standpoint, in the sense that there is matrix multiplication, however LDA does not make use of the third diagonal matrix present in LSA. Matrix 1 of LDA multiplication consists of the number of unique words and the number of topics as the two axes, whereas matrix 2 consists of the number of documents and the number of topics as the two axes. This type of probabilistic model works on two layers, the document level, and the corpus level. Blei et al. (2003) provide an overview of the model in a graphical representation as can be seen below:

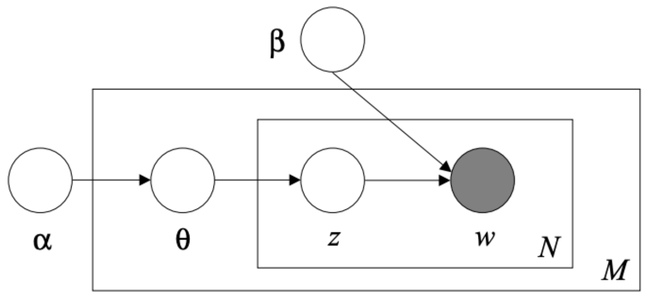


Figure 4: Graphical Representation of LDA

The outer box of the above figure represents *M* documents while the inner box represents a recursive examination of words *w,* and topics *z*.

Blei et al. (2003) then further elaborate on the above probabilistic representation by explaining the three-layered approach that occurs. “The parameters α and β are corpus level parameters, assumed to be sampled once in the process of generating a corpus. The variables θ_d are document-level variables, sampled once per document. Finally, the variables z_{dn} and w_{dn} are word-level variables and are sampled once for each word in each document.” (Blei et al. 2003, pg. 997)

Most commonly, parameters α and β are the parameters that have the most tuning required, as the desirable value of these two parameters is to ensure the maximum log-likelihood of the data, specifically in a corpus of documents represented by $D = \{w_1, w_2, \ldots, w_M\}$. The aim to discover the most optimal hyperparameters α and β can be represented by:

$$l(\alpha, \beta) = \sum_{d=1}^{M} \log p(w_d | \alpha, \beta)$$

Equation 5: Maximisation of Hyperparameters for Latent Dirichlet Allocation

For further information regarding the background of the LDA model and the probabilistic nature, as well as deeper knowledge into the various parameter estimation models[2] and a discussion around the concept of LDA being a conjugate prior of a multinomial distribution[3], see the LDA publication by Blei et al. (2003).

As this study is completed in the Python programming language, there are a number of libraries in Python that have an LDA function, and for the purposes of this study, the *LatentDirichletAllocation* function in the *sklearn.decomposition* package will be used.

Text Classification

Along with topic modelling being one of the most common use cases in NLP, text classification is just as common if not more common. The concept behind text classification is to assign tags,

[2] http://acsweb.ucsd.edu/~yuw176/report/lda.pdf
[3] http://www.jmlr.org/papers/volume3/blei03a/blei03a.pdf

labels, or categories to text. This can be achieved by having two sets of data, one set that can be used for training the underlying model, and a second set that can be used as the testing data for the classification model.

There are a number of use cases for text classification such as spam detection, sentiment analysis and categorisation. For the case of spam detection, the classifier would look over the unseen, new data, and based on the training set of data that was used on the model, would return either a *1,* representing spam, or a *0,* representing no spam. This concept can be applied on a broader level, and in the case of this study, will be used to identify whether the feedback provided by interviewer's is classed as *successful,* represented by a *1,* or *unsuccessful,* represented by a *0*.

It's important to note that a key distinguishing feature between topic modelling and text classification is the type of supervision of the learning. Topic modelling is inherently related to unsupervised learning, as the goal of topic modelling is to infer topics from a corpus, rather than have the topics known already and access the accuracy of them. Text classification on the other hand, is a type of supervised learning, where all the training corpus has to have been labelled prior to ingestion into the algorithm, and then based on this supervised approach, will determine the outcome for documents not seen prior.

There are a multitude of algorithms that can be used for the purposes of text classification such as; logistic regression, random forest, and gradient tree boosting to name a few. Throughout this study, a version of gradient tree boosting was used, called Extreme Gradient Boosting, otherwise known as the XGBoost[4] algorithm.

Extreme Gradient Boosting (XGBoost)

As noted on Wikipedia, "Gradient boosting is a machine learning technique for regression and classification problems, which produces a prediction model in the form of an ensemble of weak prediction models, typically decision trees. It builds the model in a stage-wise fashion like other

[4] http://dmlc.cs.washington.edu/data/pdf/XGBoostArxiv.pdf

boosting methods do, and it generalizes them by allowing optimization of an arbitrary differentiable loss function." (Wikipedia, 2020). Gradient boosting is part of the supervised machine learning group, but differs in the way it operates. As noted by Boehmke et al. (2020), gradient boosting falls into a similar category of supervised learning as random forests and bagging, in the sense that it is not based on a single predictive model, such as linear regression, but gradient boosting works to combine multiple models together.

The main concept behind boosting is to continuously improve and add new models to the ever-expanding ensemble, where an ensemble is a collection of models. These models are added in a sequential order to create the *boosting* effect of the algorithm. Boosting begins with a weak model that contains a number of errors, as expected, and can be thought of as a simple decision tree with minimal decisions. Each new model that is added to the ensemble creates a new tree, where "each new tree in the sequence will focus on the training rows where the previous tree had the largest prediction errors" (Boehmke et al. 2020, pg. 222). Boehmke et al. (2020) outline this approach graphically as seen below:

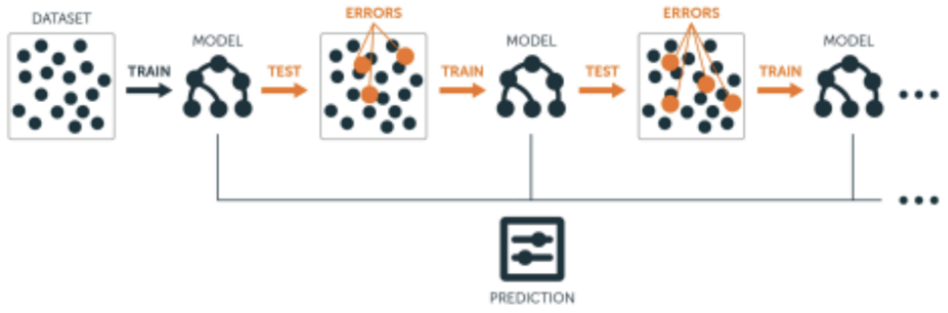


Figure 5: Boosting Tree Graphical Representation

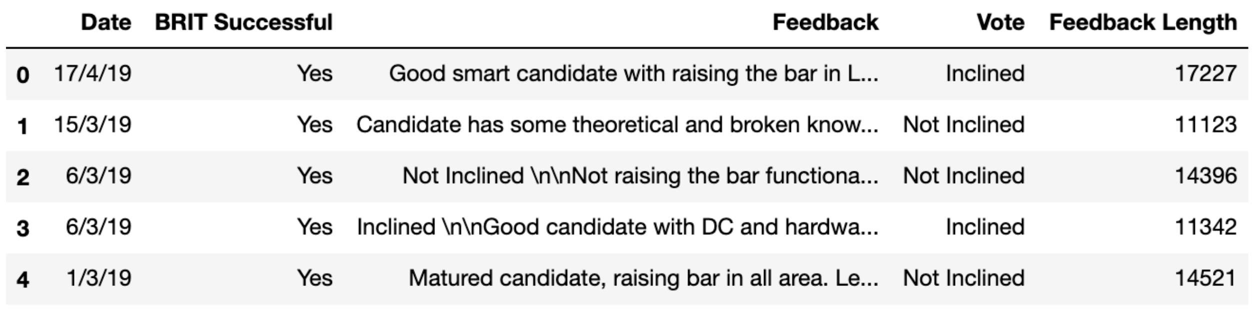
Methodology

Pre-Processing

As previously mentioned, the major objective of this study is to determine the effects of utilising various topic modelling approaches, as well as determining an appropriate label for each document based on text classification. In order to achieve this, the sampled data provided by AWS was put through a variety of pre-processing and algorithm analysis, to identify originally if there are visibility distinguishing features between ‘successful’ and ‘unsuccessful’ candidates for the BRIT program.

To start, a sample of 600 documents, all converted into text, were placed into a CSV file. To be able to individually search the words, punctuation was removed, along with brackets, special characters and new lines to ensure that there was only text left.

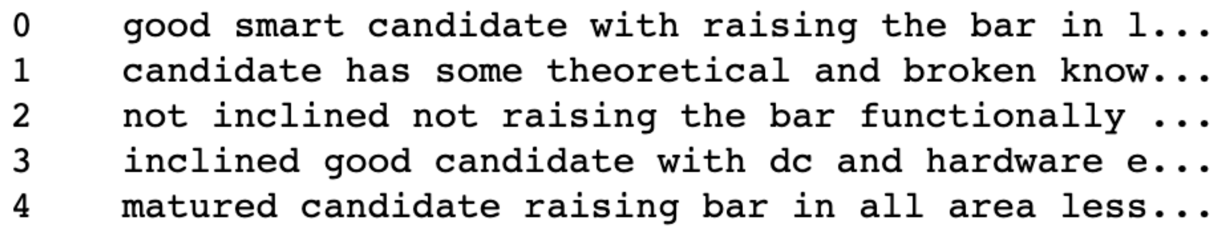
This means the data went from this data frame:



	Date	BRIT Successful	Feedback	Vote	Feedback Length
0	17/4/19	Yes	Good smart candidate with raising the bar in L...	Inclined	17227
1	15/3/19	Yes	Candidate has some theoretical and broken know...	Not Inclined	11123
2	6/3/19	Yes	Not Inclined \n\nNot raising the bar functiona...	Not Inclined	14396
3	6/3/19	Yes	Inclined \n\nGood candidate with DC and hardwa...	Inclined	11342
4	1/3/19	Yes	Matured candidate, raising bar in all area. Le...	Not Inclined	14521

Figure 6: Original Corpus without Processing

To this:



```
0    good smart candidate with raising the bar in l...
1    candidate has some theoretical and broken know...
2    not inclined not raising the bar functionally ...
3    inclined good candidate with dc and hardware e...
4    matured candidate raising bar in all area less...
```

Figure 7: Original Corpus Post Processing

As can be seen in the above image, the only data left after the initial step of processing is the relevant feedback data which will be used in the topic modelling and classification.

To quickly understand the composition of all the documents, a word cloud was used to visualise the most frequently occurring words, as pictured below:

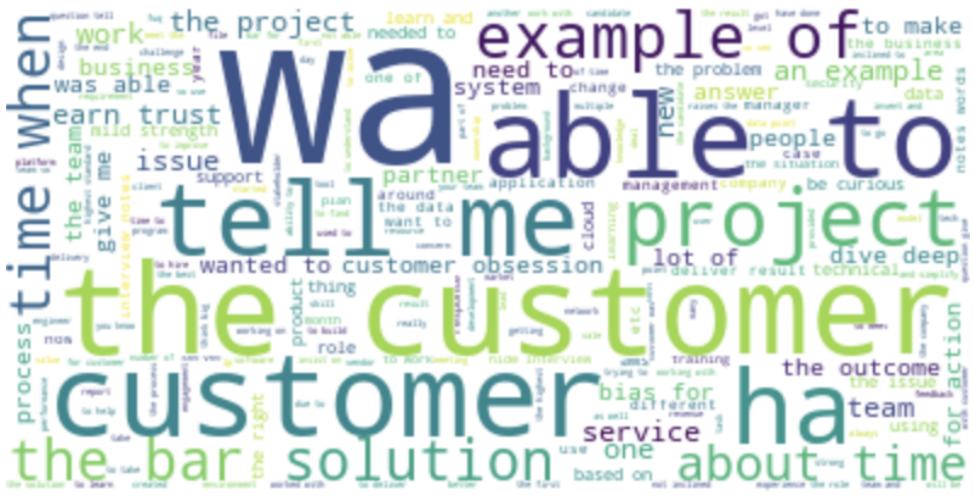


Figure 8: World Cloud of Original Corpus

Examining the above word cloud, there are immediately some signs that there is a need for further processing. One such sign comes from the duplicate of the word *customer*, and the abbreviations *ha* and *wa.* Another identifier that there needs to be another layer of processing on data, is when words such as *the* and *an* are present in the most common words as seen above.

For some further analysis into the most frequent words, the top 50 most frequent words, with their count, on the unprocessed data showed the following:

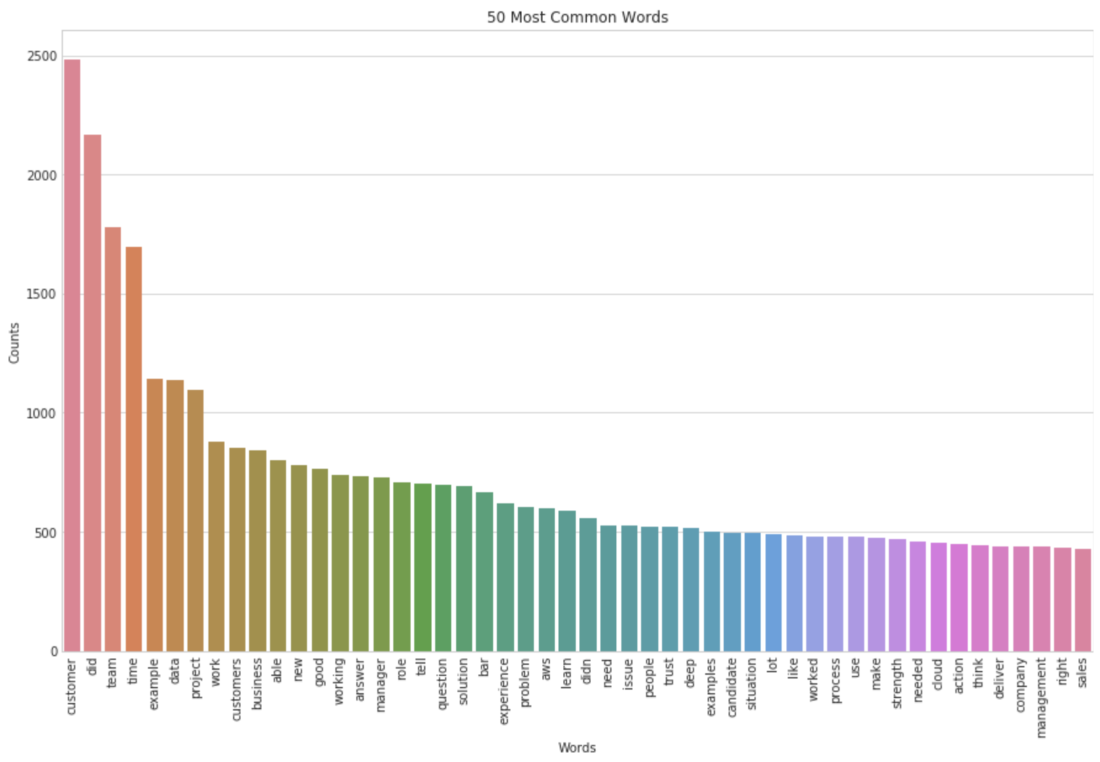


Figure 9: 50 Most Common Words

Deeper examination of the top 50 terms shows that words known as *stop-words* such as *did* are in the top 2 for number of occurrences in the corpus. In addition, the word *customer* and *customers* appear as two distinct words, which in the context of topic modelling, essentially represents a duplicate word. As well as this, when going through a number of documents, it was evident that text documents may contain abbreviations to certain terms, and these abbreviations may not be consistent across the corpus. One such example of this in the text corpus provided in this study is the use of the term *follow up questions*, being abbreviated to any of the following; *fuq, followup, fu/q, foq* and a multitude of others.

An important step of pre-processing of the corpus for this study is to identify whether or not there are certain features that are expected in *'successful'* documents. The AWS team provided a list of features that would assist in labelling whether a document is *successful*, and one feature included *evidence of peeling the onion through asking repeated follow up questions*. Due to the importance of this feature, all occurrences of any representation of asking follow up questions, would have to be aggregated, meaning there needs to be a way to standardise this.

There are a number of techniques that can be applied to the corpus to increase the accuracy, as well as address the above issues. Such techniques include the removal of stop words, stemming and lemmatization, and word replacement to name a few.

Stop Words

As previously mentioned, words such as *'the'* and *'did'* were amongst the most common words, however this provides no insight into the text and allows no inference to be made on the corpus as a whole. These words are known as stop words, in other terms, words that do not provide much meaning to the overall sentence.

A list of English stop words can be seen below:

['i', 'me', 'my', 'myself', 'we', 'our', 'ours', 'ourselves', 'you', "you're", "you've", "you'll", "you'd", 'your', 'yours', 'yourself', 'yourselves', 'he', 'him', 'his', 'himself', 'she', "she's", 'her', 'hers', 'herself', 'it', "it's", 'its', 'itself', 'they', 'them', 'their', 'theirs', 'themselves', 'what', 'which', 'who', 'whom', 'this', 'that', "that'll", 'these', 'those', 'am', 'is', 'are', 'was', 'were', 'be', 'been', 'being', 'have', 'has', 'had', 'having', 'do', 'does', 'did', 'doing', 'a', 'an', 'the', 'and', 'but', 'if', 'or', 'because', 'as', 'until', 'while', 'of', 'at', 'by', 'for', 'with', 'about', 'against', 'between', 'into', 'through', 'during', 'before', 'after', 'above', 'below', 'to', 'from', 'up', 'down', 'in', 'out', 'on', 'off', 'over', 'under', 'again', 'further', 'then', 'once', 'here', 'there', 'when', 'where', 'why', 'how', 'all', 'any', 'both', 'each', 'few', 'more', 'most', 'other', 'some', 'such', 'no', 'nor', 'not', 'only', 'own', 'same', 'so', 'than', 'too', 'very', 's', 't', 'can', 'will', 'just', 'don', "don't", 'should', "should've", 'now', 'd', 'll', 'm', 'o', 're', 've', 'y', 'ain', 'aren', "aren't", 'couldn', ***"couldn't", 'didn', "didn't", 'doesn', "doesn't", 'hadn', "hadn't", 'hasn', "hasn't", 'haven', "haven't", 'isn', "isn't", 'ma', 'mightn', "mightn't", 'mustn', "mustn't", 'needn', "needn't", 'shan', "shan't", 'shouldn', "shouldn't", 'wasn', "wasn't", 'weren', "weren't", 'won', "won't", 'wouldn', "wouldn't"]***

By examining the above stop words, it may seem that the removal of all these words throughout the corpus will improve the results as it will remove the words previously mentioned as problematic words, however, it is important to address a possible problem[5] with the words in bold above, depending on the use case.

In the case of using the removal of stop words for more complex NLP tasks such as sentiment analysis or if one is working with long short-term memory (LSTM), then the removal of the bold words above introduces a problem when the meaning of sentences is desired. Let's take a few sentences as examples:

1. *This product is good* (Positive sentiment)
2. *I wouldn't buy this product* (Negative sentiment)
3. *The product could be better* (Negative sentiment)
4. *I don't like the product* (Negative sentiment)

Now, if all the stop words above were removed, including the bolded words, the processed data would look like:

1. *Product good* (Remains positive sentiment)
2. ***Buy product* (Negative sentiment becomes a positive sentiment)**
3. *Product could better* (Remains negative sentiment)
4. ***Like product* (Negative sentiment becomes a positive sentiment)**

The removal of all the stop words from the first group of statements, results in a change of sentiment, ultimately leading to incorrect statements. With NLP tasks such as BoW, tf-idf and topic modelling, the sentiment of the sentences is not relevant, and each word is examined in isolation. This means that the removal of stop words will in fact refine the corpus and ensure more accurate results.

[5] https://towardsdatascience.com/why-you-should-avoid-removing-stopwords-aa7a353d2a52

Taking stop word removal into account for this study, a new word cloud can be produced by converting all the rows of feedback in the data frame to a long string, tokenizing the string, and then only adding the words that are not found in the stop words to a new list of words, using the following Python code:

```
tokens = word_tokenize(original_string_of_words)
tokens_without_stopwords = [word for word in tokens if not word
in english_stopwords]
```

After running the above commands, the word cloud generated post removal of stop words looks like:

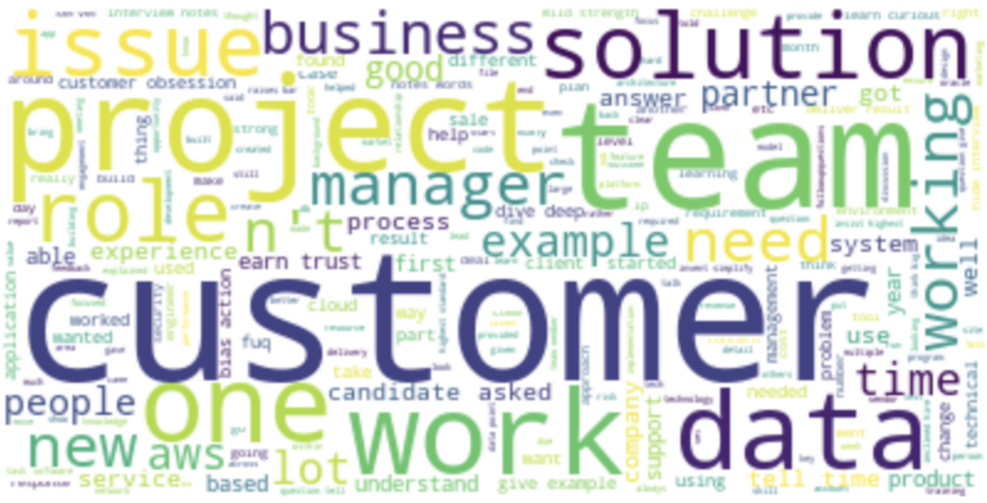
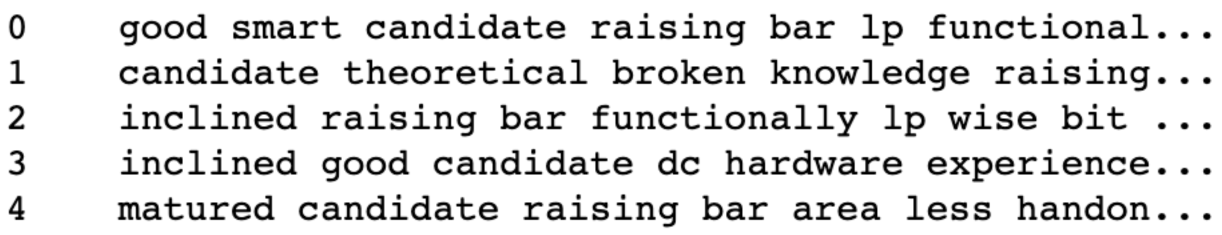


Figure 10: World Cloud Post Stop Word Removal

Immediately it can be seen that the word *'the'* is no longer amongst the words in the cloud, and there starts to be more information regarding the *'successful'* corpus.

Observing the first five documents in the data frame, it is now visibly noticeable that the stop words have been removed, as the first document no longer includes the word *'with'*:



```
0     good smart candidate raising bar lp functional...
1     candidate theoretical broken knowledge raising...
2     inclined raising bar functionally lp wise bit ...
3     inclined good candidate dc hardware experience...
4     matured candidate raising bar area less handon...
```

Figure 11: Original Corpus Example Post Stop Word Removal

Further inspection of the top 50 most common words after removing stop words, shows that the word '*did*' is no longer the second most common word:

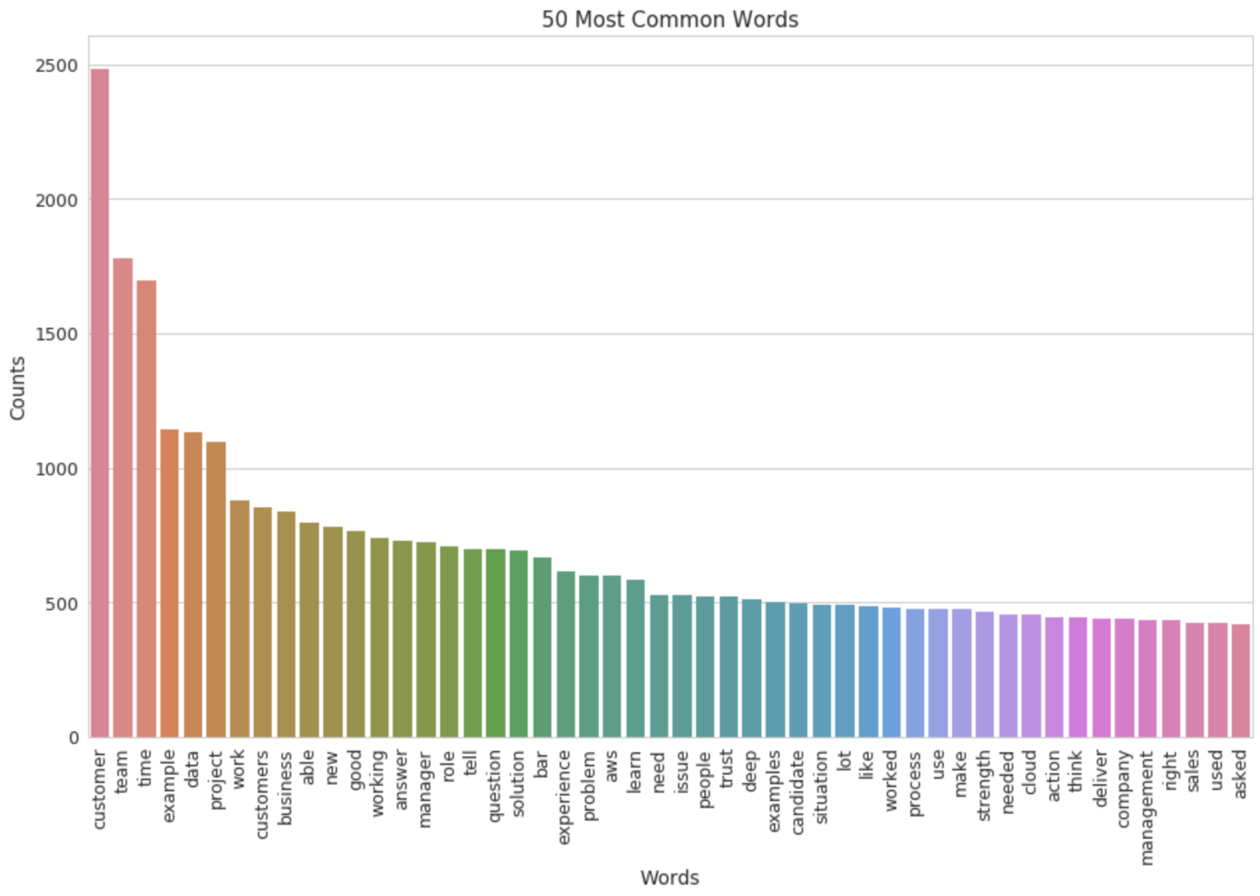


Figure 12: 50 Most Common Words Post Stop Word Removal

Once the corpus has been refined through the removal of stop words, the next pre-processing step can commence. Examining the above bar graph of most common words, the most common word is *'customer'* and the 8th most common word is *'customers'*, plural. To address this, a technique called Stemming and Lemmatization can be used.

Stemming

In 1979, the original stemming algorithm was developed and distributed in Cambridge and discussed in *New models in probabilistic information retrieval* written by *C.J. van Rijsbergen, S.E. Robertson and M.F. Porter*.

"The Porter stemming algorithm (or 'Porter stemmer') is a process for removing the commoner morphological and inflexional endings from words in English. Its main use is as part of a term normalisation process that is usually done when setting up Information Retrieval systems" (Porter 2016, para. 2)

For the purposes of demonstrating what stemming achieves, the Porter stemmer will be used, however in practice, the new Snowball stemmer is the recommended approach.

Let's take the example of the following words:

- *Computer*
- *Computed*
- *Computes*
- *Computing*

Applying the Porter stemmer algorithm to the above words, they would all return:

- *Comput*
- *Comput*
- *Comput*
- *Comput*

The concept behind Porter's stemming algorithm (Porter, 2006) is to identify the composition of words by looking at the number of consonants and vowels that comprise the word. For the purposes of Porter's study, "A list ccc... of length greater than 0 will be denoted by C, and a list vvv... of length greater than 0 will be denoted by V. Any word, or part of a word, therefore … may all be represented by the single form:

[C]VCVC ... [V]

where the square brackets denote arbitrary presence of their contents." (Porter 2006, para. 15-16)

Porter then goes on to introduce another variable, *m,* also known as the measure, where *m=0,* represents the null or base version of the word. The best way to explain this is to visualise it, similar to what Porter did in his original paper:

$$SSES \rightarrow SS\ therefore\quad caresses \rightarrow caressIES \rightarrow I\quad therefore\quad ponies\ \rightarrow poniSS \rightarrow SS\quad therefore\quad c$$

The left column above represents the reduction of the word, meaning that SSES will be replaced with SS, IES with I, SS will be replaced with itself, and S will be replaced with null. The first value in the left column represents S1 and the second element is S2. Porter then narrowed this down to a generalist equation of:

$$(condition)\ S1 \rightarrow S2,$$

Equation 6: Condition for Stemming Replacement

where the condition would usually be in terms of m, i.e., *(m > 1)*, and "this means that if a word ends with the suffix S1, and the stem before S1 satisfies the given condition, S1 is replaced by S2" (Porter 2006, para. 18-19)

The issue with the stemming approach is that the stems that are produced after iterative suffix replacement, are not always guaranteed to be words, as is seen in the initial example with the *compute* words. To overcome this, lemmatization can be used.

Lemmatization

Stemming and Lemmatization achieve similar outcomes, where they both reduce a word to its root. However, the stem of the word, as previously shown, may not produce an actual word, whereas lemmatization will return the lemma of the word, otherwise known as the dictionary form of a word, and aims to remove inflectional endings only. Lemmatization also changes its output, the lemma, based on whether the word is a noun or verb.

Referencing the *compute* words above, lemmatization of the words will return:

- *Computer*
- *Compute*
- *Computes*
- *Compute*

As can be seen, all the words returned are dictionary words and vary depending on the way the word is used, i.e., noun or verb.

Bringing lemmatization into the context of this study, as is seen in *Figure 11*, the words *customer* and *customers* are in the top 10 most common words. Applying lemmatization, the word *customers* would be replaced with *customer*, aggregating the frequency of the two words into one, rather than having distinct words, increasing the accuracy of the topic modelling.

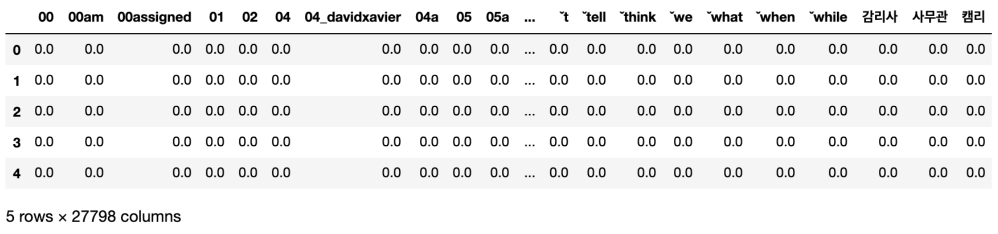
Results

There were a number of tests run for the classification and labelling of the corpus. To begin with, the XGBoost algorithm was run on the initial corpus of documents, which included 600 samples of both *successful* and *unsuccessful* classified data, represented by *1* and *0* respectively. The first classification test was on the unprocessed data, with no maximum document frequency (max_df) or minimum document frequency (min_df) taken into account.

The max_df characteristic sets a hard threshold on the maximum document frequency, i.e., the percentage of documents that contain that specified word. Usually these values are between 0 and 1 inclusive. As previously identified with the tf-idf approach, that max_df characteristic is in line with the concept of tf-idf, as the purpose of this characteristic is to discard words that appear in too many documents, similar to stop words.

On the other hand, the min_df characteristic is the opposite. It sets a hard threshold on the minimum document frequency. Although this approach is in contradiction of the nature of tf-idf, it was chosen for the purposes of this study as the size of the tf-idf matrices computed were not substantial enough to warrant an approach such as Principal Component Analysis (PCA) or Singular Value Decomposition (SVD).

The first test conducted was on the corpus as it is, with minimal pre-processing, and default min/max_df values, ensuring all data is present. Conducting tf-idf on the original corpus with the above conditions presents a tf-idf matrix of over 27,000 columns (representing words) over the course of the 600 documents, seen below:



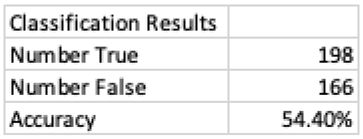
	00	00am	00assigned	01	02	04	04_davidxavier	04a	05	05a	...	ˇt	ˇtell	ˇthink	ˇwe	ˇwhat	ˇwhen	ˇwhile	감리사	사무관	캠리
0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	...	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0
1	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	...	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0
2	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	...	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0
3	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	...	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0
4	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	...	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0

5 rows × 27798 columns

Figure 13: TF-IDF Matrix on Original Corpus with Default Constraints

Using a Python approach found on Kaggle[6], and modified to suit this study's use case, the tf-idf matrix was then broken up into training features and testing features, on the word and character level, to be used as input to train the XGBoost model. The outcome of this first test produced the following results:

[6] https://www.kaggle.com/nicapotato/tf-idf-xgboost

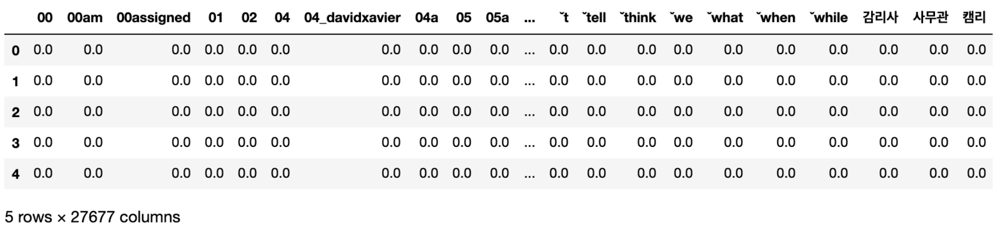


Classification Results	
Number True	198
Number False	166
Accuracy	54.40%

Figure 14: Classification Results on Original Corpus with Default Constraints

The *Number True* field is the number of documents that were classified by XGBoost that match the *actual* documents label, which was decided prior to training. This *actual* label was emitted for training purposes, and only used for comparison post training. The *Number False* field is the number of documents that are not correctly classified in comparison to the actual label. It is important to note that the terms for *correct classification* is if the XGBoost output value is *greater than or equal to 0.5*.

The next test run was still on the original corpus, but using a max_df value of *0.5*, and a default min_df value. Utilising the same code as mentioned above, the tf-idf matrix achieved was as follows:

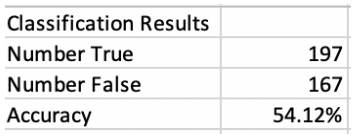


	00	00am	00assigned	01	02	04	04_davidxavier	04a	05	05a	...	ˇt	ˇtell	ˇthink	ˇwe	ˇwhat	ˇwhen	ˇwhile	감리사	사무관	캠리
0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	...	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0
1	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	...	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0
2	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	...	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0
3	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	...	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0
4	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	...	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0

5 rows × 27677 columns

Figure 15: TF-IDF Matrix on Original Corpus with Maximum Document Frequency Constraint

As can be seen, the number of columns was reduced from *27798* to *27677*, a reduction of *121* words. The max_df value of *0.5* puts a restraint on the tf-idf operation to only include terms that are present in at most 50% of the documents in the corpus, i..e, in at most *300* documents.
The purpose of this is to exclude terms that are so frequent that they provide no extra information to either topic modelling or classification. The outcome of the text classification for this test can be seen as:



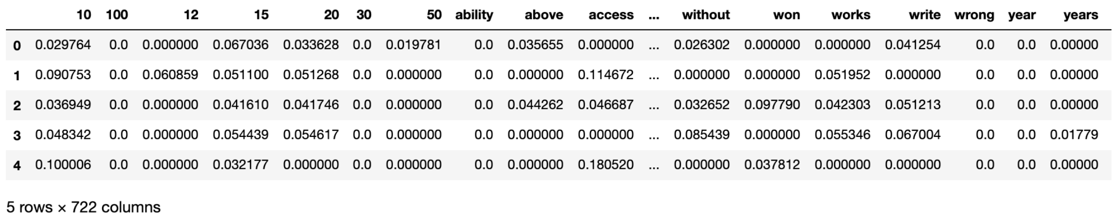
Classification Results	
Number True	197
Number False	167
Accuracy	54.12%

Figure 16: Classification Results on Original Corpus with Maximum Document Frequency Constraint

The accuracy of this test shows a correct labelling of 197 documents compared to the previous 198 documents, decreasing the accuracy by *0.28%,* a very marginal amount.

To further restrain the tf-idf matrix, the next test utilised a min_df value of *0.1* as well as keeping the max_df value of *0.5*. Applying this minimum document frequency value restrains the tf-idf matrix to only include words that appear in at least 10% of the documents in the corpus. The range of document frequency within this test can be understood as: $0.1 \leq df \leq 0.5$, where *df* is the document frequency of a given a word.

The tf-idf matrix generated from the above constraint can be seen as:

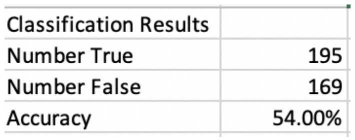


	10	100	12	15	20	30	50	ability	above	access	...	without	won	works	write	wrong	year	years
0	0.029764	0.0	0.000000	0.067036	0.033628	0.0	0.019781	0.0	0.035655	0.000000	...	0.026302	0.000000	0.000000	0.041254	0.0	0.0	0.00000
1	0.090753	0.0	0.060859	0.051100	0.051268	0.0	0.000000	0.0	0.000000	0.114672	...	0.000000	0.000000	0.051952	0.000000	0.0	0.0	0.00000
2	0.036949	0.0	0.000000	0.041610	0.041746	0.0	0.000000	0.0	0.044262	0.046687	...	0.032652	0.097790	0.042303	0.051213	0.0	0.0	0.00000
3	0.048342	0.0	0.000000	0.054439	0.054617	0.0	0.000000	0.0	0.000000	0.000000	...	0.085439	0.000000	0.055346	0.067004	0.0	0.0	0.01779
4	0.100006	0.0	0.000000	0.032177	0.000000	0.0	0.000000	0.0	0.000000	0.180520	...	0.000000	0.037812	0.000000	0.000000	0.0	0.0	0.00000

5 rows × 722 columns

Figure 17: TF-IDF Matrix on Original Corpus with Maximum and Minimum Document Frequency Constraint

It's immediately evident that the size of the tf-idf matrix has reduced significantly, from *27677* in the previous test, to *722* with both a minimum and maximum constraint applied. This signifies that majority of the words that make up the initial tf-idf, fall into the range of: $df < 0.1$, as this constraint caused the largest reduction in the matrix. The accuracy of the above classification was calculated as:

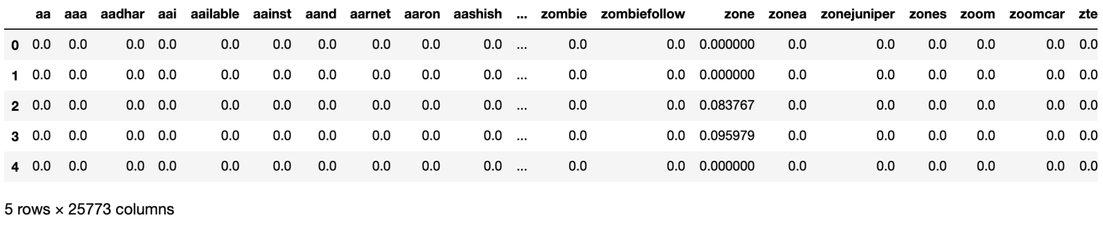


Classification Results	
Number True	195
Number False	169
Accuracy	54.00%

Figure 18: Classification Results on Original Corpus with Maximum and Minimum Document Frequency Constraint

This concludes that between the three tests examined above, the case of no limitations on the document frequency provided the best result of *54.40%*, however the changes in percentage over the three tests is miniscule. The minimum and maximum document frequency constraints can be played around with almost any value between 0 and 1, and therefore would take an enormous amount of time, for potentially a minute percentage delta, therefore only the values of *0.1* and *0.5* were used for this study.

The next set of cases were conducted on the processed corpus, with the removal of stop words and sentiment words, as well as numbers and unhelpful terms. Beginning with the base test case of no maximum or minimum constraints, the tf-idf matrix calculation resulted in:

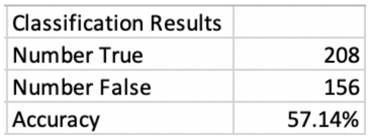


	aa	aaa	aadhar	aai	aailable	aainst	aand	aarnet	aaron	aashish	...	zombie	zombiefollow	zone	zonea	zonejuniper	zones	zoom	zoomcar	zte
0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	...	0.0	0.0	0.000000	0.0	0.0	0.0	0.0	0.0	0.0
1	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	...	0.0	0.0	0.000000	0.0	0.0	0.0	0.0	0.0	0.0
2	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	...	0.0	0.0	0.083767	0.0	0.0	0.0	0.0	0.0	0.0
3	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	...	0.0	0.0	0.095979	0.0	0.0	0.0	0.0	0.0	0.0
4	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	...	0.0	0.0	0.000000	0.0	0.0	0.0	0.0	0.0	0.0

5 rows × 25773 columns

Figure 19: TF-IDF Matrix on Processed Corpus with Default Constraints

The extra steps of processing the data prior to running tf-idf on the corpus has reduced the size of the tf-idf from *27798* to *25773,* and the expectation is that the accuracy will be increased to, which was the case as seen below:

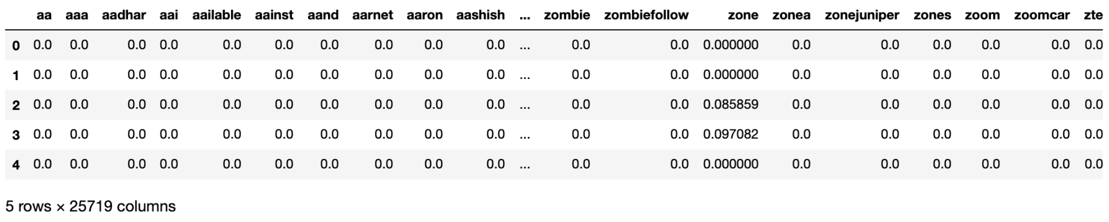


Classification Results	
Number True	208
Number False	156
Accuracy	57.14%

Figure 20: Classification Results on Processed Corpus with Default Constraints

The accuracy of text classification on the processed corpus compared to the unprocessed corpus increased by over 3%, which although is a small increase, represents the need for processing of data prior to analysis and classification.

Similar to the previous test cases on the unprocessed corpus, tests were conducted on the processed corpus with various minimum and maximum document frequency constraints. Keeping the constraints consistent with previous tests, the next test run made use of a max_df constraint of *0.5*, achieving the following tf-idf output:

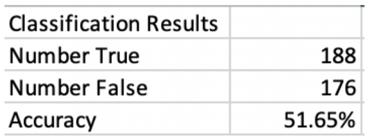


	aa	aaa	aadhar	aai	aailable	aainst	aand	aarnet	aaron	aashish	...	zombie	zombiefollow	zone	zonea	zonejuniper	zones	zoom	zoomcar	zte
0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	...	0.0	0.0	0.000000	0.0	0.0	0.0	0.0	0.0	0.0
1	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	...	0.0	0.0	0.000000	0.0	0.0	0.0	0.0	0.0	0.0
2	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	...	0.0	0.0	0.085859	0.0	0.0	0.0	0.0	0.0	0.0
3	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	...	0.0	0.0	0.097082	0.0	0.0	0.0	0.0	0.0	0.0
4	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	...	0.0	0.0	0.000000	0.0	0.0	0.0	0.0	0.0	0.0

5 rows × 25719 columns

Figure 21: TF-IDF Matrix on Original Corpus with Maximum Document Frequency Constraint

In accordance with the previous test case on the unprocessed data, modifying the max_df constraint did not reduce the matrix by a substantial amount as it was reduced by *54*. In terms of the accuracy of classification, this was also reduced as shown below:

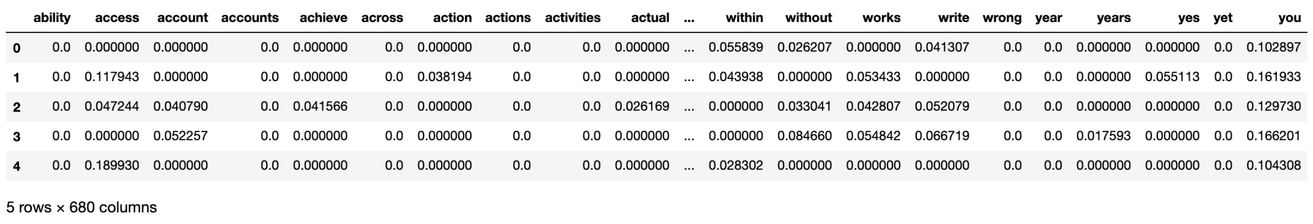


Classification Results	
Number True	188
Number False	176
Accuracy	51.65%

Figure 22: Classification Results on Processed Corpus with Maximum Document Frequency Constraint

The accuracy of the classification on the processed data with a maximum document frequency constraint drops drastically by almost *6%.* An explanation as to why this happened is due to the nature of the max_df attribute as previously mentioned. Since stop words had been removed from the corpus, the words that appear in more than *50% (max_df > 0.5)* of the documents, are the most helpful words to classify the documents as they are so prevalent across the corpus.

Finally, adding a min_df constraint of *0.1* on to the max_df constraint, produces the following matrix:

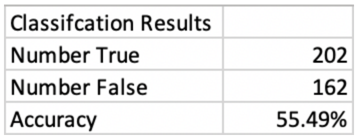


	ability	access	account	accounts	achieve	across	action	actions	activities	actual	...	within	without	works	write	wrong	year	years	yes	yet	you
0	0.0	0.000000	0.000000	0.0	0.000000	0.0	0.000000	0.0	0.0	0.000000	...	0.055839	0.026207	0.000000	0.041307	0.0	0.0	0.000000	0.000000	0.0	0.102897
1	0.0	0.117943	0.000000	0.0	0.000000	0.0	0.038194	0.0	0.0	0.000000	...	0.043938	0.000000	0.053433	0.000000	0.0	0.0	0.000000	0.055113	0.0	0.161933
2	0.0	0.047244	0.040790	0.0	0.041566	0.0	0.000000	0.0	0.0	0.026169	...	0.000000	0.033041	0.042807	0.052079	0.0	0.0	0.000000	0.000000	0.0	0.129730
3	0.0	0.000000	0.052257	0.0	0.000000	0.0	0.000000	0.0	0.0	0.000000	...	0.000000	0.084660	0.054842	0.066719	0.0	0.0	0.017593	0.000000	0.0	0.166201
4	0.0	0.189930	0.000000	0.0	0.000000	0.0	0.000000	0.0	0.0	0.000000	...	0.028302	0.000000	0.000000	0.000000	0.0	0.0	0.000000	0.000000	0.0	0.104308

5 rows × 680 columns

Figure 23: TF-IDF Matrix on Processed Corpus with Maximum and Minimum Document Frequency Constraint

Compared to the unprocessed classification case with the same constraints, there are *42* less terms, and the accuracy of this test was identified as:



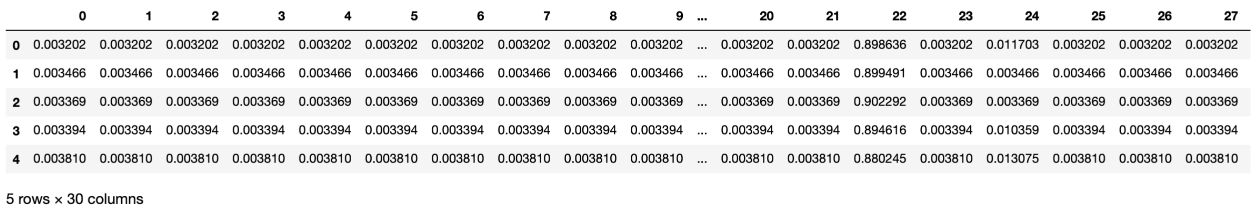
Classifcation Results	
Number True	202
Number False	162
Accuracy	55.49%

Figure 24: Classification Results on Processed Corpus with Maximum and Minimum Document Frequency Constraint

This is an increase from the previous test where there was no min_df value. Applying a min_df value on the processed corpus will most likely remove names and other terms that rarely appear over the course of the corpus, allowing for a more accurate classification. In comparison to the unprocessed test case with minimum and maximum constraints, this test improves the accuracy by just over 1%, again, quite miniscule, but supports the theory of removing words such as names.

For the last classification test in this study, a combination of a tf-idf matrix and LDA topic vectors was used. The tf-idf matrix chosen was that which achieved the highest accuracy on its own, which was the processed corpus tf-idf with no maximum or minimum document frequency constraints. The LDA topic vectors used were that of 30 topics, with 10 words per topic.

The tf-idf matrix for the most accurate model so far was already mentioned in *Figure 18*, whereas the LDA topic vector looks like this:



	0	1	2	3	4	5	6	7	8	9	...	20	21	22	23	24	25	26	27
0	0.003202	0.003202	0.003202	0.003202	0.003202	0.003202	0.003202	0.003202	0.003202	0.003202	...	0.003202	0.003202	0.898636	0.003202	0.011703	0.003202	0.003202	0.003202
1	0.003466	0.003466	0.003466	0.003466	0.003466	0.003466	0.003466	0.003466	0.003466	0.003466	...	0.003466	0.003466	0.899491	0.003466	0.003466	0.003466	0.003466	0.003466
2	0.003369	0.003369	0.003369	0.003369	0.003369	0.003369	0.003369	0.003369	0.003369	0.003369	...	0.003369	0.003369	0.902292	0.003369	0.003369	0.003369	0.003369	0.003369
3	0.003394	0.003394	0.003394	0.003394	0.003394	0.003394	0.003394	0.003394	0.003394	0.003394	...	0.003394	0.003394	0.894616	0.003394	0.010359	0.003394	0.003394	0.003394
4	0.003810	0.003810	0.003810	0.003810	0.003810	0.003810	0.003810	0.003810	0.003810	0.003810	...	0.003810	0.003810	0.880245	0.003810	0.013075	0.003810	0.003810	0.003810

5 rows × 30 columns

Figure 25: LDA Topic Matrix Output for 30 Topics

It currently shows 5 rows as it is a preview, similar to the above data frame displays, however it contains the total number of documents in this frame.

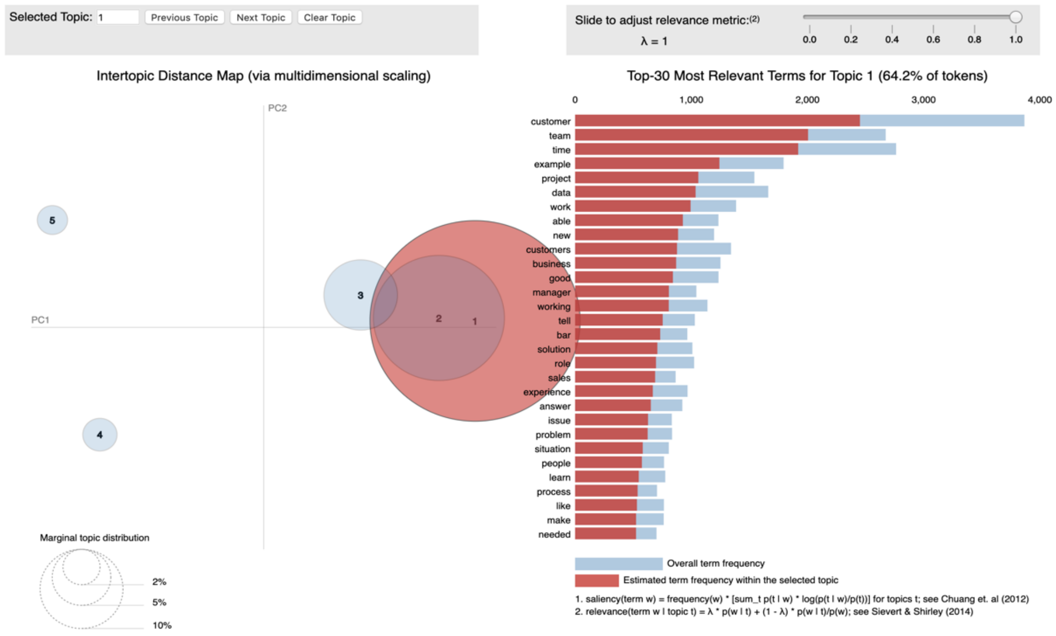
These two data frames were combined and put through the XGBoost algorithm, however the result returned was *57%* as can be seen in the output below in the *f1-score* column, and the *accuracy* row:

```
training score: 0.8893496216651205
testing score: 0.5578388666661509
done in 42.048s.
             precision    recall  f1-score   support

          0       0.47      0.50      0.49       149
          1       0.64      0.61      0.63       215

   accuracy                           0.57       364
  macro avg       0.56      0.56      0.56       364
weighted avg      0.57      0.57      0.57       364
```

Figure 26: Accuracy of Classification Using TF-IDF Concatenated with LDA

Although these topics had no direct impact on the classification accuracy (which is a possibility due to the limited size of the corpus and the type of text that is being used in the corpus), there may be a more desired outcome by tuning the hyperparameters of the LDA algorithm. In the case above, a document topic prior, β*,* of *0.1* was used, and a document word prior*,* α, of *0.01* was used. Overall, including LDA did not reduce the accuracy of the classification

In terms of the topic modelling approach to infer the topics that are present in the documents, LDA was run on both the unprocessed and processed corpuses, with topic numbers *5, 10* and *30.* The Python package pyLDAvis was used to visualise the top 30 common words within the topic, as well as the relationship between the various topics. An example of this can be seen below:



1. saliency(term w) = frequency(w) * [sum_t p(t | w) * log(p(t | w)/p(t))] for topics t; see Chuang et. al (2012)
2. relevance(term w | topic t) = λ * p(w | t) + (1 - λ) * p(w | t)/p(w); see Sievert & Shirley (2014)

Figure 27: LDA Topic Visualisation

The above image was run on the processed corpus with a topic number of *5,* and does very well in demonstrating the inter topic distance between the topics. From the outset, it is evident that topics 1 and 2 are the most related, whereas topics 4 and 5 are large outliers.

Conclusions and Recommendations

Regarding the above results, it is clear to see that an analysis on a corpus post processing, including the removal of stop words through stemming and lemmatization, as well as utilising the concept of term frequency and inverse document frequency to eliminate outliers and unhelpful data, is more accurate, albeit not by a lot in this study due to the corpus size, than running an analysis on the base corpus without processing.

Although LDA and XGBoost were used in this study, there are a plethora of other algorithms that can be used for topic modelling and classification. For example, from a classification perspective, a study[7] can be conducted to determine which classification algorithm works best for the corpus at hand, comparing algorithms such as linear regression and XGBoost to see which produces better results. From a topic modelling perspective, LSA or Probabilistic Latent Semantic Analysis (pLSA) can be compared to LDA, there is no algorithm that is better than all the others, but there might be better algorithms for certain use cases.

In terms of next steps and recommendations to improve the approach taken in this study, there are a few trials and tribulations to be undertaken. Firstly, a choice of which stemmer algorithm between Porter's Stemmer or the Snowball Stemmer can be made. They are both easy to implement and the Snowball Stemmer is the more preferred stemmer within the NLP industry. For lemmatization, the spaCy[8] Python library comes with a *Lemma_* attribute and can convert strings into a spaCy string, to then find the lemma of each word as previously identified in this study. Another area for improvement is to finetune the hyperparameters of the LDA model, which currently consist of a document topic prior, β, of *0.1* and a document word prior, α, of *0.01.* Along the lines of tuning hyperparameters, the remaining parameters of the LDA algorithm such as the number of topics, and the number of words per topic, can also be adjusted to reach a more optimal result.

The most important recommendation to be made for this study is to use a large corpus for both the training and the testing. For this study, a training corpus of 600 documents was used, and a test corpus of 347 documents was used. In reality, the results for both LDA and the classification would be drastically improved on corpus sizes of over 50,000 documents. There are a number of

[7] https://github.com/elleros/text-classification-challenge
[8] https://stackabuse.com/python-for-nlp-tokenization-stemming-and-lemmatization-with-spacy-library/

public sets of available data, such as the *Iris Flow Data Set*[9], that can be used to verify all the models spoken about.

In conclusion, this study has demonstrated that a very diverse document set, with minimal guidelines and consistency, can be modelled and interpreted through the use of NLP methods, and can more often than not, label unseen documents with an acceptable modicum of accuracy, through simple fine tuning of common NLP algorithms. This study has also attempted to fill an evident gap in the NLP world of an end to end tool that does not require much setup or development, to identify common features in a document and then use those features to train a classifier.

[9] https://en.wikipedia.org/wiki/Iris_flower_data_set

References

Blei, D.M., Ng, A.Y. & Jordan, M.I. 2013, 'Latent Dirichlet Allocation', *Journal of Machine Learning Research,* vol. 3, pp. 993-1022.

Boehmke, B. & Greenwell, B.M. 2020, *Hands-on machine learning with R,* CRC Press, Boca Raton, FL.

Chen, S. & Wang, Y. *Latent Dirichlet Allocation,* .

Evangelopoulos, N.E. 2013, 'Latent semantic analysis', *Wiley Interdisciplinary Reviews: Cognitive Science,* vol. 4, no. 6, pp. 683-92.

Ferlitsch, A. 2019, *Making the machine: the machine learning lifecycle,* viewed 13/6/ 2020, <https://cloud.google.com/blog/products/ai-machine-learning/making-the-machine-the-machine-learning-lifecycle>.

Ghosh, S. & Gunning, D. 2019, *Natural Language Processing Fundamentals,* Packt Publishing, .

Gradient boosting, 2020, viewed 21/06/ 2020, <https://en.wikipedia.org/wiki/Gradient_boosting#Gradient_tree_boosting>.

Hirschberg, J., Manning, C.D. & Hirschberg, J. 2015, 'Advances in natural language processing', *Science (New York, N.Y.),* vol. 349, no. 6245, pp. 261-6.

Kondylidis, N., Tzelepi, M. & Tefas, A. 2018, 'Exploiting tf-idf in deep Convolutional Neural Networks for Content Based Image Retrieval', *Multimedia Tools and Applications; An International Journal,* vol. 77, no. 23, pp. 30729-48.

Liu, L., Tang, L., Dong, W., Yao, S. & Zhou, W. 2016, 'An overview of topic modelling and its current applications in bioinformatics', *SpringerPlus,* vol. 5, no. 1, pp. 1-22.

Ma, E. 2018, *2 latent methods for dimension reduction and topic modelling,* viewed 18/06/ 2020, <https://towardsdatascience.com/2-latent-methods-for-dimension-reduction-and-topic-modeling-20ff6d7d547#:~:text=Both%20LSA%20and%20LDA%20have,LDA%20solves%20top

ic%20modeling%20problems.&text=Latent%20Semantic%20Analysis%20(LSA,Latent%20Dirichlet%20Allocation%20(LDA)>.

Maklin, C. 2019, 'TF IDF | TFIDF Python Example', *Towards Data Science,* .

Manning, C.D., Raghavan, P. & Schütze, H. 2008, *Introduction to information retrieval,* Cambridge University Press, New York.

Nikolenko, S.I., Koltcov, S. & Koltsova, O. 2017, 'Topic modelling for qualitative studies', *Journal of Information Science,* vol. 43, no. 1, pp. 88-102.

Porter, M.F. 2006, 'An algorithm for suffix stripping', *Program,* vol. 40, no. 3, pp. 211-8.

Wang, S. 2018, *Build a Text Classification Program: An NLP Tutorial,* viewed 13/06/ 2020, <https://www.toptal.com/machine-learning/nlp-tutorial-text-classification>.