The BRITISH UNIVERSITY IN EGYPT

Faculty of Informatics and Computer Science

Artificial Intelligence

Sentiment Analysis and Classification of Yelp Reviews

By: Daniel Youssef

206150

Supervised By

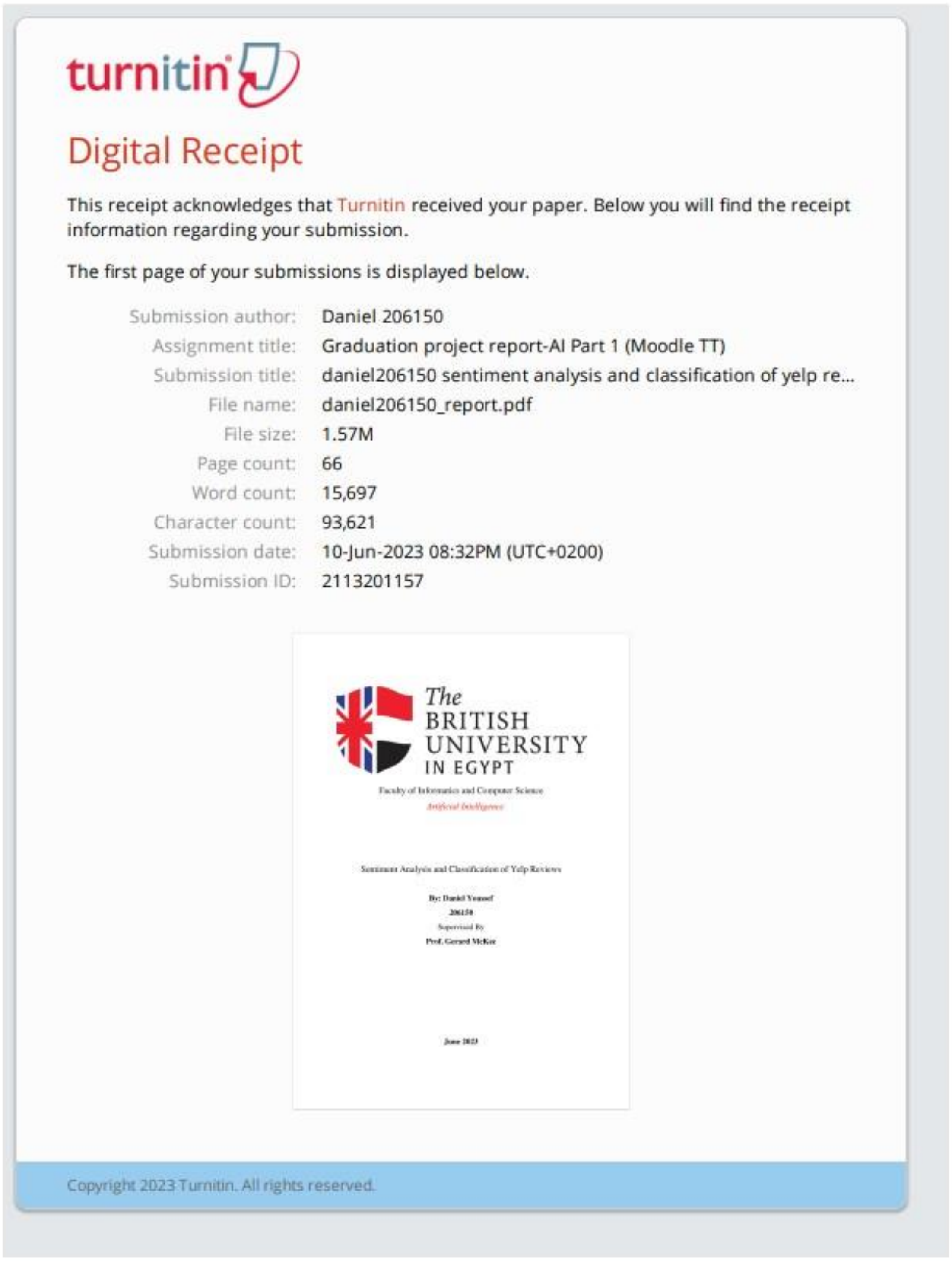
Prof. Gerard McKee

June 2023

Abstract

In the field of Natural Language Processing, one of the problems that has varying degrees of effective solutions is the ability to perform Sentiment Analysis on textual data, and to retrieve an AI-based solution that is able to predict what the textual data conveys. In order to attempt to reach a reliable solution to this problem, a large body of research, as well as related work was analysed, and summarized. As a result of this, a proposed solution was implemented, utilizing a dataset containing 6990280 reviews, and a novel technique for resampling data to further improve upon prediction scores for underrepresented classes. The proposed models utilized an XGBoost architecture, and an LSTM architecture, to achieve accuracy scores of 85%, and 77% respectively.

Turnitin Report



turnitin

Digital Receipt

This receipt acknowledges that Turnitin received your paper. Below you will find the receipt information regarding your submission.

The first page of your submissions is displayed below.

Submission author:	Daniel 206150
Assignment title:	Graduation project report-AI Part 1 (Moodle TT)
Submission title:	daniel206150 sentiment analysis and classification of yelp re...
File name:	daniel206150_report.pdf
File size:	1.57M
Page count:	66
Word count:	15,697
Character count:	93,621
Submission date:	10-Jun-2023 08:32PM (UTC+0200)
Submission ID:	2113201157

The BRITISH UNIVERSITY IN EGYPT

Faculty of Informatics and Computer Science

Artificial Intelligence

Sentiment Analysis and Classification of Yelp Reviews

By: Daniel Youssef

206150

Supervised By

Prof. Gerard McKee

June 2023

Acknowledge

First, I would like to thank Dr. Gerard McKee - my supervisor - for assisting me in learning more about this topic, and for providing me with further guidance, and support while I was working on this project.

Also, I would like to thank Dr. Nahla Barakat, for preparing me with the necessary education in order to pursue this topic and its underlying research.

In addition, I would like to extend my gratitude to Ashraf Adel – my class's student representative – for all the hard work, and effort he put into allowing this project to come into fruition without any setbacks from other modules.

Lastly, I would like to give a big thank you to all my friends, and family who supported me throughout the project timeline. Without them, this project would not have come into fruition.

Table of Contents

Table of Contents

List of Figures

List of Tables

1 Introduction

Since the beginning of time, data has been an integral part in solving varying degrees of problems of everyday life, from predicting the water levels of the nile river in ancient Egyptian times [1], to predicting crime before it occurs in New York City back in 1994 [2]. Today, data is an invaluable tool, that is being compared to valuable natural resources such as oil [3]. However, data comes in all shapes and sizes, meaning it is not just presented in one format. Data comes in the form of numbers, dates, text, Boolean values, and symbols. For this report, a deeper investigation will be made on textual data, and an AI-based solution will be built on this underlying data.

The study of how to process data using a linguistic framework is called "Natural Language Processing", and it is a group of computational steps that breakdown and represent human languages using numbers, such that a computer can work with them more easily [4]. A branch of NLP that is being used to extract emotions, or opinions from textual data is called Sentiment Analysis, and it works by analyzing the hidden sentiment behind the text that humans share with each other over popular social media platforms, and other websites. As such, the technique has also earned the name "Opinion Mining" [5]. In order to perform Sentiment Analysis, a machine learning model will be trained for this particular task on large amount of data, and this process will be explained thoroughly throughout the report.

1.1 Overview

The steps taken in the project will be in a "pipeline" format, meaning the data will be retrieved from the source, and then it will go through several steps to ensure the highest possible quality data is being used for the task. These steps are customary for most textual data-based projects, as textual data cannot be presented in its raw format entirely, if effective results are desired. Any steps that may seem unorthodox or uncommon for text analysis projects, will be explained, and the reason for their existence in this project will be justified. Each step in the project will be done using the Python Programming Language, as well as its accompanying libraries that are available on the Internet. Some of the libraries that will be used include NumPy, Pandas, NLTK, TensorFlow, and more. Below, the project steps are shown, and the explanation for each step, and how it will be taken is added.

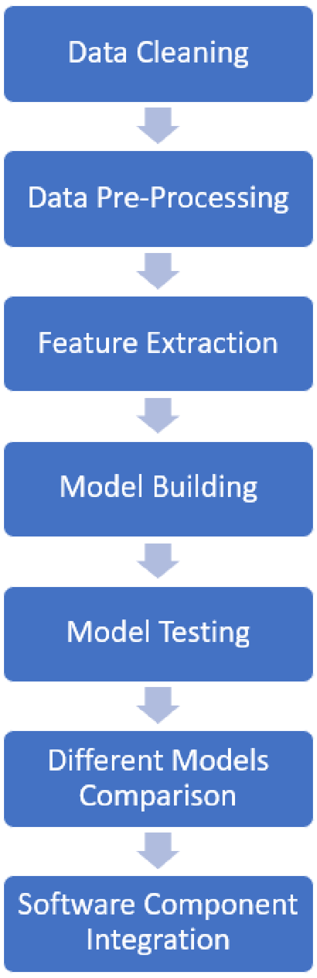


Figure 1. Project Steps

As shown in Figure 1., The data will go through multiple steps before it is used in modelling. All the steps before the "Model Building" step are vital to ensuring the model will be able to train and be tested correctly.

First, before discussing any steps that were taken in the project, the datasets must be discussed first. There were two datasets, both containing yelp reviews about businesses.

First, the smaller dataset was retrieved from a popular Data Science community website called Kaggle. The dataset consisted of 7617 rows, and the columns that are present in the main file (ratings_and_sentiments.csv) consisted of the review data, some basic sentiment data, as well as the rating that each review received. Only the text reviews and the star ratings were used for the project, and the dataset was primarily focused on coffee shops, rather than generic businesses. This dataset was used for small-scale experiments that do not require a large amount of data. However, the steps taken in this dataset were vital, as the results that are generated from these experiments were used to justify some of the decisions that were taken later on in the project.

Second, the larger dataset was retrieved from Yelp's main website. The dataset consisted of 6990280 rows of data as of the time this project was completed, with the columns comprising of some details about the reviewed business, the user who left the review, the text review itself, and the rating that the review received. Once again, only the text reviews, and the star ratings were used for this project, and the dataset did not focus on any type of business. Rather, it consisted of a wide swath of businesses that were reviewed on Yelp. This dataset was used primarily for training the models, as well as for visualizations to better understand the reviews themselves.

Lastly, the links for downloading these datasets will be provided in Appendix I.

1.1.1 Data Cleaning

Since this work involves textual data, the data cleaning process will be restricted to mainly duplicate record removal, and null value record removal. Any further data manipulation will be done in the "Data Pre-Processing" step as it is more relevant to a project of this scope.

1.1.2 Data Pre-Processing

Before beginning to work on any project involving text analysis, a few data pre-processing steps must be taken, and these will be explained firstly, then the steps of the whole project will be explained further.

There are many kinds of ways sentiment analysis can be used to extract sentiments from text, for example: there is a "word level" analysis, a "phrase level" analysis, a "sentence level" analysis, and a "document level" analysis [6]. Since the datasets that will be used for this project comprise of consumer reviews on products, and businesses, the steps for a "document level" type of analysis will be taken here.

In addition, the text data will be split from a massive paragraph to the individual words that make the paragraph. This step is called "Tokenization", and it is the process of creating "Tokens" which are the words that make up a document or paragraph. This step is a fundamental step of the data pre-processing process, as most textual data cleaning is done on individual words [7].

Furthermore, the next data pre-processing step that will be taken will be to remove stop words. Stop words are words that take up spots in a sentence, but do not add any value to an opinion or sentiment. It was also referred to as "useless data", moreso in the field of sentiment analysis [8].

Moving on, each token will be transformed into lowercase, and all alphanumeric characters, such as numbers, and symbols, will be removed from the text data, as it does not impact the sentiment of the text.

Lastly, each token will be tagged using Part Of Speech (POS) tagging, and lemmatized to bring tokens back to their original format, as seen in dictionaries. This step is necessary to providing an accurate sentiment score for each review, as prefixes, and suffixes do not add useful information to each tokenized word.

1.1.3 Feature Extraction

After the dataset was pre-processed, the features of these sentences were extracted. There are many feature extraction methods that can be used, they all, however, fall under the same title of "word embeddings". These word embeddings extract each word in a review and give it a certain numerical score to represent it. In addition, word embeddings use extracted words as features that are ready to be used to perform sentiment analysis by either calculating the sentiment polarity of a review, or to be input into a machine learning model for training.

It is important to note, while there are other techniques to represent words numerically than word embeddings, these techniques are not effective for text analysis, as they do not represent the semantic meaning behind the words, and these techniques cannot capture context, which is crucial when performing a task that requires as much useful information provided sequentially as possible like Sentiment Analysis.

Some examples of the three most popular word embeddings would be Word2Vec, TF-IDF, and GloVe. While many embeddings are viable, and have been used in many papers, the GloVe embedding is superior to the previous ones as it finds the global statistics, also referred to as "word co-occurrences", which determine the value of the words in a document, rather than the small sentences that make up the document [9].

The Word2Vec embedding works by grouping similar words in a vector format. Word2Vec has two different models it can work with, first is the continuous bag-of-words (CBOW) model and the continuous Skip-gram model. The Skip-gram model was found to be much more efficient in working with sparse words in data than the CBOW model [9]. The term frequency–inverse document frequency (TF-IDF) embedding works by giving a weight to a word or phrase, and each word or phrase is given a weight based on the frequency of its appearance in the document. It follows four steps that involve calculating the number of times each word appears in each document (TF), then calculating the number of documents containing a specific word (DF), afterwards it calculates the inverse of DF (IDF), lastly it calculates TF-IDF by multiplying TF by IDF.

In this report, Word2Vec and TF-IDF will be compared with each other, and the better performing one will be compared with GloVe to determine which embedding will be used during model optimization.

1.1.4 Model Building

After the data has been pre-processed, and the relevant features have been extracted, it is time to begin analysing the sentiments behind the data. To do this, a model must be constructed, and trained on the data to make its own predictions. For this type of project, the models will be built to attempt to predict the score rating each review received.

For this report, a selection of models utilizing both machine learning, and deep learning algorithms will be built. An example of the models that were built are using algorithms such as Random Forest, Extreme Gradient Boosting, Recurrent Neural Networks, and Long-Short Term Memory Networks, to name a few. In addition, the models and their underlying algorithms will be discussed in full detail in Section 3, where the modelling aspect of the project will be explored thoroughly.

1.1.5 Model Testing

In order to understand which metrics would be suitable to measure the performance of the trained models, the datasets will have to be analysed to determine these metrics.

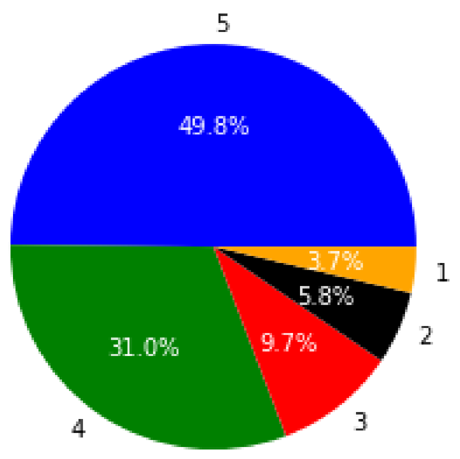


Figure 2 Smaller Dataset Labels Distribution

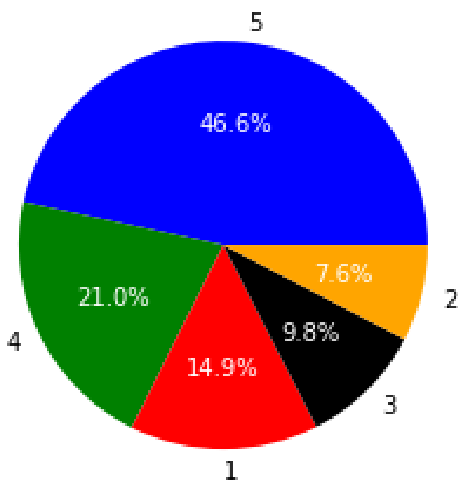


Figure 3 Larger Dataset Labels Distribution

First, looking at the visualizations of the datasets in the previous two figures 2, and 3, we can see that both datasets were not balanced, meaning that there are reviews that have received a certain rating more than another reviews. This is an issue, as using accuracy as a metric in this case would be a bad decision, since not all ratings have the same number of corresponding reviews [11]. Therefore, it is better to use the F1-Score metric, since it captures the number of reviews classified correctly for each class, rather than the overall correct classification count. In addition, the Macro F1-Score metric will be used instead of the regular F1-Score Metric as the Macro variant will calculate the mean F1-Score of all classes, allowing for a more informative metric to be used. In addition, it is important to mention that the accuracy metric will be used whenever a dataset is balanced through resampling, and also later in the project, since the larger dataset will be modified in a certain way to be balanced for all classes, however the exact details of how this process was done will be covered later on in the report.

1.1.6 Comparison of Different Models

After the models have been trained and their evaluation is complete, the models will be compared, to find the best performing one according to the previously mentioned metrics. There will be two types of models that will be compared: the 5-label models, and the 3-label models. As the name suggests, the 5-label models were models trained to predict the typical 5-star ratings format in the dataset. These types of models will be optimized, and the best model will be implemented in the software application. However, the 3-label models were trained to predict only the three basic sentiments that can be inferred from the ratings: Positive, Negative, and Neutral. These types of models were designed to be able to be compared and evaluated by the recent research papers done on sentiment analysis. These models are way more optimized for prediction, and the best model will be placed in a direct comparison with the latest state-of-the-art model in this case. Also, the advantage of using three labels not only improves model performance, but also allows this model to be compared with the value produced by the sentiment polarity, which acts as the baseline for this report.

1.1.7 Software Component Integration

Lastly, as an effort to allow sentiment analysis to be more accessible to the public rather than being predominantly a research-focused field, a small-scale web application will be developed after the research is complete. This application will use the highest performing 5-label model, and it will allow users to input the review one may have for a certain business, and the model will attempt to predict the given rating for such a review. The web application code, and instructions on how to run will be provided in the "web app" directory of the project.

1.2 Problem Statement

A problem that plagues many sentiments analysis-based research projects or hosted applications for review websites or other types of media, is that it can be difficult for the underlying model to accurately predict the rating, given by a certain user, as it is very difficult to understand the sentiment a person exhibits purely from text. What this report hopes to accomplish, is to provide a deep learning approach to solving this problem, even if this solution is not perfect, it will attempt to accurately predict the ratings of the inputted review without any bias towards any single label.

1.3 Scope and Objectives

The scope of this project extends to the research of different NLP techniques for feature extraction, sentiment analysis, and classification of yelp reviews according to their rating from 1 to 5. In addition, a small-scale web application will be developed to allow users to write reviews of their own and have the best performing model predict the rating the user is willing to give based on their sentiment. There will be two datasets used for this project.

1.4 Report Organization (Structure)

The report contains four further sections. Section 2 will contain some research done on the current State-of-The-Art technology being used in the field, as well as a comparative analysis of the related work to this project. Section 3 will explain in more detail the solution, as well as a detailed explanation of each step of the project, and how this step affected the results, Lastly, Section 4 will conclude with the most significant findings that were covered in the report, as well as some final notes.

1.5 Work Methodology

This report attempts to accomplish the ability of accurately predicting ratings, or sentiments behind written text using a NLP, and Deep Learning combined approach by training models on large amounts of review data provided by Yelp. The report will follow the seven steps written in Figure 1. In addition, the research files, and the software application will be released, containing all the code, and visualizations generated by this project.

1.6 Work Plan (Gantt chart)

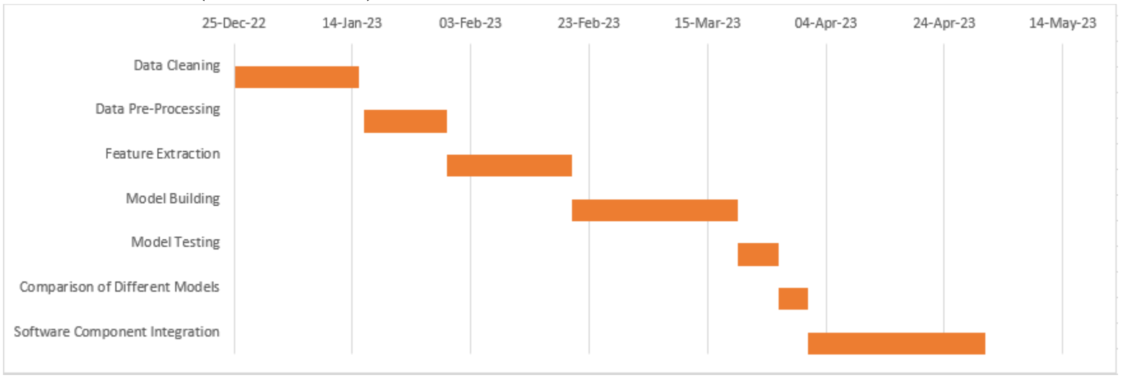


Figure 4 Gantt Chart

2 Related Work (State-of-The-Art)

2.1 Background

The papers that were researched an analysed are papers that were attempting to apply either machine learning, or deep learning techniques to the field of opinion mining. The datasets for each paper are expected to be varied, and the techniques used are expected to overlap across several papers.

2.2 Literature Survey

Looking at [12], Haque, Saber, and Shah looked at how reviews on products sold on the popular e-commerce website Amazon can have their ratings predicted. The researchers used a new technique in the field called "Pool Based Active Learning" which suggests that models can achieve better performance with less training data if the model is allowed to choose the data in which it trains on. In addition, the researchers used three data pre-processing techniques: Tokenization, Removing Stop Words, and POS (Parts of Speech) Tagging. Next, TF-IDF, and Chi Square were the feature extraction tool used by the researchers. Then, the models were trained, and then evaluated using Precision, Recall, F-Measure, and Accuracy metrics. The researchers found that the Linear Support Vector Machine performed the best out of the six algorithms that were used in the paper with an accuracy score of 94.02%.

Then, at [13], the authors of the paper decided to look at online feedback reviews of hotels, and attempt to determine whether customers visiting hotels in Seoul would return later to the same hotel, or would choose another hotel without using an AI-based approach. The researchers used linguistic inquiry and word count (LIWC), which is a text mining software that counts the words in the dataset that are proven to be more cognitively, psychologically more effective, and calculates a sentiment score for them. Based on the respondents' data, 43.2% would return to the same hotel during their second visit of Seoul. The researchers were able to conclude that those who would re-visit the same hotel were less analytical than the re-visitors, and the re-visitors were more likely to use more words per sentences. Also, the re-visitors were more likely to use more positive words than one-time visitors.

Moving forward to [14], the authors decided to take a deep-learning approach for the process of sentiment analysis for movie reviews. The researchers used two different feature extraction methods: Bag of Words, and Word2Vec, which are covered in this paper, as well. Then, the feature vectors were defined by two methods: averaging, where the average of word vectors for all words is calculated, and clustering, where the words are transformed to vectors, and similarities are detected among different words using the K-means clustering algorithm. After the two methods were completed, the following algorithms were applied to the data in order to train three models: Random Forest, Support Vector Machine, Logistic Regression. The three models used a Word2Vec feature extraction technique with average feature vectors, and achieved accuracies of 84.0%, 85.8%, 86.6% respectively. When clustering feature vectors were attempted with the Random Forest algorithm, it obtained an accuracy of 83.5%, which was lower, and therefore less effective than the averaging feature vectors technique. Lastly, the researchers trained a model using a deep learning approach, which used a Recurrent Neural Tensor Network, with a Stochastic Gradient Descent algorithm. This approach was able to improve upon the previous models with a 1.5% increase in accuracy score.

2.3 Analysis of the Related Work

In this portion of Section 2, a few papers that were very closely related to the planned approach this paper will take were analysed and summarized.

Starting first with [15], Liu wanted to test the ability of several machine learning models to classify reviews based on their ratings from 1 to 5 using a yelp reviews dataset. First, an Ablation study is done, which consists of training a model using different variations of the dataset in order to find the best possible parameters. The algorithm used was the Multinomial Naïve Bayes algorithm. Based on the results from the ablation study, the data was imbalanced, and balancing the data yielded better results. Next, the researcher evaluated three different feature extraction methods, CountVectorizer (also known as Bag of Words), TF-IDF, and CountVectorizer with binary indicators, which measures word occurrences over word frequencies. The CountVectorizer with binary indicators variation was found to be the most effective when viewing the results of the model. Then, N-grams were used to capture modified verbs and nouns, which improves model performance. The best result was found when unigrams, and bi-grams were used together. Afterwards, stop words were removed, and the feature space was normalized. Lastly, six different AI-based models were made using algorithms, and neural networks from MNB, LR, SVM, Gradient Boosting, Long Short-Term Memory, and Bidirectional Encoder Representations from Transformers. The best results were from the simpler models such as Logistic Regression and Support Vector Machines, however the researcher notes that there is no best algorithm when it comes to sentiment analysis, and that differences in techniques used, and the dataset used are more likely to skew results than what algorithms, or neural networks were used.

Moving on to [16], The authors are pursuing the same goal of predicting user review ratings from yelp reviews using several machine learning, and deep learning techniques. However, this paper is different, in that the rating prediction is only the first half of the paper. The second half of the paper is dedicated to attempting to predict the user's rating for a business based on previous ratings that same user gave other businesses. It is important to note, that only the first part of the paper will be covered, as the second part is out of this paper's scope. First, the paper starts with cleaning the data, by removing punctuation, and white spaces. Word stemming was used, which removes word suffixes to retrieve the root of the word. Next, stop words were removed, then the feature extraction process began. For feature extraction, these are the exact steps the researchers took: "we need to count the frequency of every word that appears in the users review pool, remove the ones with frequency lower than a certain bound (we chose 1e-4), so as to reduce the sparsity of our training matrix, and also The frequencies of the rest 1,252 words compose our list of training features. Finally, we extracted a sparse training matrix, where the i-th row represents the i-th review, and the j-th column represents the j-th token. The (i, j)-entry of this matrix represents the number of occurrences of the j-th token in the i-th review" [16]. Next, three models were made to predict both sentiment polarity scores and for 5-star classification each. For the models that predicted the 5-star classification ratings, the following models were made: Multinomial Logistic Regression, Naïve Bayes, Support Vector Regression, Support Vector Machine. Based on the results, SVR, and SVM were the best performing algorithms for predicting the 5-star ratings.

Lastly, an in-depth look was taken at [17]. The authors set a goal of predicting review ratings from Yelp based on the reviews themselves alone. First, some data pre-processing is done, in which all punctuation, and white spaces are removed from the text. Then, feature extraction was done using Bing Liu Opinion Lexicon, which summarizes customer reviews according to the adjectives. While the benefit of using lexicons is that the dataset does not have to be looped over to extract its relevant features, lexicons have an issue where some irrelevant features are selected, while other relevant features are not selected, since the features are not extracted from the dataset itself. Also, the researchers added their own variations to the lexicon by appending "not" to every word between negation and the following punctuation, removing stop words, and stemming. Next, the following mix of neural networks, and machine learning algorithms were used to make models: a multi-layer perceptron with stochastic gradient decent algorithm, Binarized Naïve Bayes, which focuses more on word occurrence rather than word frequency, Multi-class SVM, and Nearest Centroid Algorithm. In the end, Binarized Naïve Bayes was able to achieve an accuracy of around 30% and was touted as the best model out of the trained ones.

Lastly, for Sentiment Analysis, the authors at [18], proposed calculating the mode of several machine learning algorithms' predictions. This paper utilized 16000 rows of yelp review data and was able to achieve a solution with an accuracy score of 78%.

For Sarcasm Detection, the authors of [19] propose a machine learning solution to the sarcasm detection problem on the reddit, and twitter datasets. In addition, it is mentioned that the size of the reddit dataset was 4400 rows. Also, a proposed model consisting of an adaboost classifier, with a decision tree as the base estimator achieved an F1-Score of 67% on the reddit dataset. Because of this, machine learning approaches will have to be proposed in order to verify these results.

Another paper on Sarcasm Detection, from the authors of [19] the authors proposed using an RNNs, and LSTMs architectures to detect sarcasm in text. The English dataset that was used in this paper contained 6554 rows. For the results, an LSTM model was used, and it achieved an accuracy of 79%

on the English dataset, and because of this, an LSTM approach will be proposed in this project in order to verify these results.

3 Proposed solution

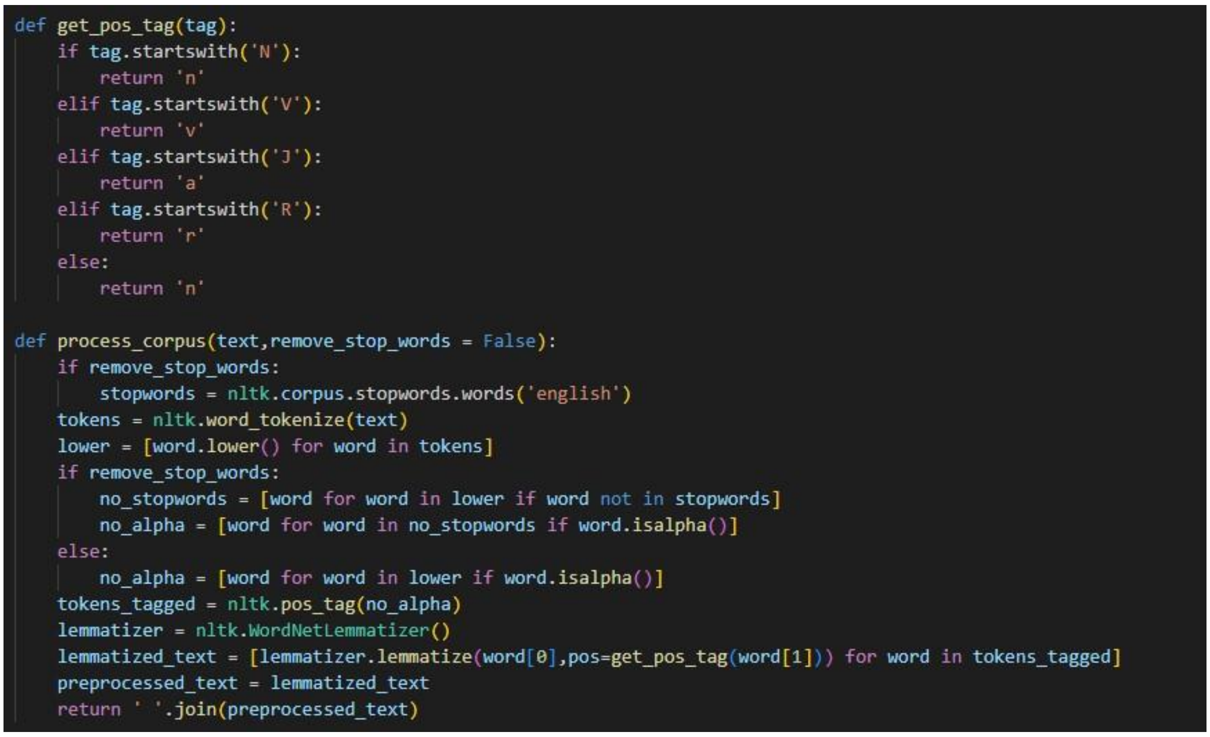
3.1 Solution Methodology

3.1.1 Smaller Dataset

As mentioned previously, the smaller dataset was used for small-scale experiments, to determine the validity of using certain techniques. The experiments that were done on the smaller dataset are:

1. Data Cleaning and Pre-Processing
2. Word2Vec vs TF-IDF
3. Data resampling techniques (random over sampling, random under sampling, SMOTE)
4. Sentiment Polarity
5. Individual Class Predictions
6. Machine Learning Classifiers

3.1.1.1 Data Cleaning and Pre-Processing



```python
def get_pos_tag(tag):
    if tag.startswith('N'):
        return 'n'
    elif tag.startswith('V'):
        return 'v'
    elif tag.startswith('J'):
        return 'a'
    elif tag.startswith('R'):
        return 'r'
    else:
        return 'n'

def process_corpus(text,remove_stop_words = False):
    if remove_stop_words:
        stopwords = nltk.corpus.stopwords.words('english')
    tokens = nltk.word_tokenize(text)
    lower = [word.lower() for word in tokens]
    if remove_stop_words:
        no_stopwords = [word for word in lower if word not in stopwords]
        no_alpha = [word for word in no_stopwords if word.isalpha()]
    else:
        no_alpha = [word for word in lower if word.isalpha()]
    tokens_tagged = nltk.pos_tag(no_alpha)
    lemmatizer = nltk.WordNetLemmatizer()
    lemmatized_text = [lemmatizer.lemmatize(word[0],pos=get_pos_tag(word[1])) for word in tokens_tagged]
    preprocessed_text = lemmatized_text
    return ' '.join(preprocessed_text)
```

Figure 5 Data Preparation Steps

The process of cleaning, and pre-processing the data starts with the previously mentioned simple cleaning process of removing null values and duplicates, then by following a standard data pre-processing pipeline. As shown in **Figure 5**, the pre-processing pipeline starts by tokenizing the dataset. Then, the tokens are converted to lowercase. Afterwards, the tokens have their stop words removed if the flag is set to true, which was the case for both the smaller, and the larger datasets. Moving forward, the tokens get their alphanumeric values removed, and they are POS tagged. Lastly, the tokens are passed through a WordNetLemmatizer provided by the NLTK library, and then the tokens are added together, and returned as a fully pre-processed review. It is important to note, the get_pos_tag, and process_corpus functions were the result of many experiments done across both the smaller, and larger datasets, and after the final versions were formulated, the functions were passed across all datasets in order to evaluate their performance. This means that, the current versions of the functions were not the first, and these functions in particular provide the highest performance, and the most through cleaning, as to be seen in the generated data visualizations.

3.1.1.2 Word2Vec vs TF-IDF

Several comparisons were done between Word2Vec and TF-IDF. First, an unoptimized version of both embeddings were used to train two Multinomial Naïve Bayes models with the same parameters. This particular classifier was chosen as it was found to be the most popular model from the analysed research from the previous section. In addition, the multinomial variant of this classifier was chosen as the dataset has non-binary label counts. The performances of both models were analysed, and the best performing model was recorded. Afterwards, another experiment was done with the same embeddings, and classifier, except this time the embeddings were optimized, to make sure that the best embedding was chosen based on its optimal performance rather than luck, or because of a certain bias of an embedding towards this particular classifier. Once again, both models' performances were analysed, and recorded.

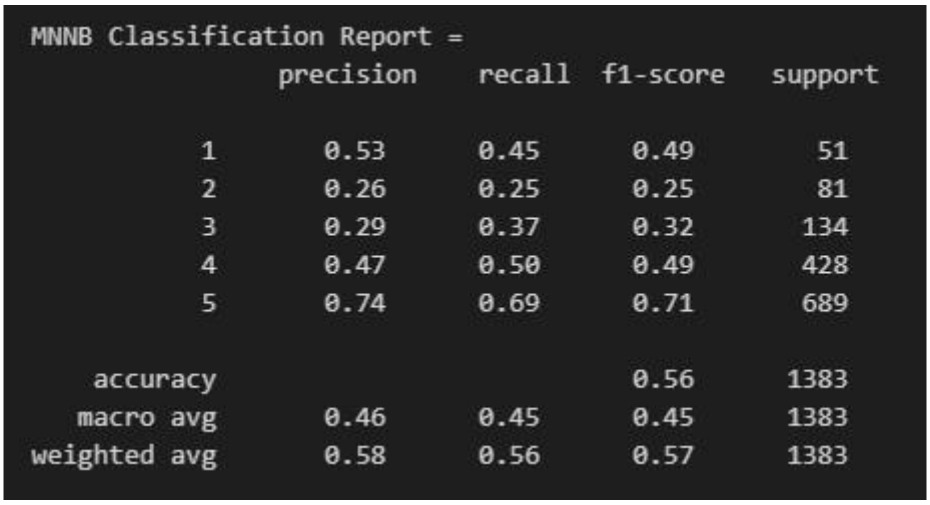
3.1.1.3 Data Resampling Techniques (random over sampling, random under sampling, SMOTE)

After the best embedding is found, it is used with the previously mentioned MNNB classifier to test and find out which one out of the three previously mentioned data resampling techniques is the most optimal. This step was crucial, as the rest of the modelling process is built upon the assumption that the data was resampled with the best resampling technique to allow for the best possible performance. Later in the report, a new resampling technique will be discussed, and how it improves the performance of a minority class.

3.1.1.4 Sentiment Polarity

One of the biggest issues that was faced during this project was the ability to predict several classes to an acceptable level. When using an unbalanced dataset, or a dataset balanced using certain techniques, it has been proven that the model achieves high performance metrics but fails to achieve a good performance on some class(es). Furthermore, the sentiment polarity score was calculated using the most optimal resampling technique, and the resulting score of this technique was used as the baseline for all future models to attempt to surpass.

3.1.1.5 Individual Class Predictions



```
MNNB Classification Report =
               precision    recall  f1-score   support

           1       0.53      0.45      0.49        51
           2       0.26      0.25      0.25        81
           3       0.29      0.37      0.32       134
           4       0.47      0.50      0.49       428
           5       0.74      0.69      0.71       689

    accuracy                           0.56      1383
   macro avg       0.46      0.45      0.45      1383
weighted avg       0.58      0.56      0.57      1383
```

Figure 6 MNNB Classification Report

As shown in **Figure 6**, the accuracy of this particular Naïve Bayes Model was shown to be 56%. However, upon inspecting the classification report of this model, it was revealed that the model was classifying one class better than the rest. Not only that, but two of the five classes were barely classified correctly. This is an important issue to solve, as classifying individual classes is more valuable for accurate predictions than to simply inflate the performance metrics of a model with one class having a high f1-score. This topic will be visited in more detail later, but it is important to mention that this dataset was balanced using the "imblearn" library, which has shown that it was not very effective in allowing the model to accurately predict some of the classes, both in 5-labeled models, and 3-labeled models. Also, 5-labelled, and 3-labelled models played a massive role in determining the best performance of a model, which will be covered in the "larger dataset" subsection.

3.1.1.6 Machine Learning Classifiers

Here are the Machine Learning Classifiers that were used on the smaller dataset:

1. Random Forest
2. Support Vector Machine
3. Bagging Ensemble K Nearest Neighbours
4. Bagging Ensemble Multinomial Naïve Bayes

5. Adaptive Gradient Booster Multinomial Naïve Bayes
6. Extreme Gradient Booster

3.1.1.6.1 Random Forest

Random Forest is an ensemble model that uses multiple decision trees that use their predictions to vote on a class. Each tree splits at a certain node amongst the best trees, rather than the traditional bagging technique of splitting each node among all variables [21]. It was expected that this classifier would achieve a higher score than most classifiers in this project.

3.1.1.6.2 Support Vector Machine

Support Vector Machine is a classifier that maps each data point using a kernel function to a vector space in order to attempt to find a vector space where the data points are linearly separable [22][21]. SVM was a flexible, and strong classifier, which is exactly what is required for working with large word embeddings.

3.1.1.6.3 Bagging Ensemble K Nearest Neighbours

An ensemble model is a model that utilizes multiple classifiers in order to improve its overall predictions [23]. This technique is further combined with a simple classifier known simply as KNN, which works by getting the majority vote of some nearest neighbours, determined by some value K [23][24]. Together, this ensemble model would allow this classifier to achieve a better performance than the KNN classifier on its own.

3.1.1.6.4 Bagging Ensemble Multinomial Naïve Bayes

Naïve Bayes is one of the most popular classifiers for text-based machine learning problems. It works by calculating the probability distribution of each review for each class, and the probability of a word appearing again increase the more occurrences it has [24]. While this approach by itself seems too simple, it was utilized with a bagging ensemble model in order to leverage its full potential.

3.1.1.6.5 Adaptive Gradient Booster Multinomial Naïve Bayes

Another approach that was taken to boost the Multinomial Naïve Bayes classifier, was to use gradient boosters like Adaptive Gradient Boosting. The way gradient boosters work is by combining multiple weak learners, or classifiers, into one powerful classifier . This particular gradient booster utilized Multinomial Naïve Bayes as a weak classifier. However, the next gradient booster will work slightly differently than this one.

3.1.1.6.6 Extreme Gradient Booster

Extreme Gradient Boosting is a gradient boosting model that also utilizes multiple weak learners and combines their performance into one model. However, XGBoost is different in that the model assesses each classifier's misclassified classes and adapts its next classifier to better fix this mistake. XGBoost was chosen for this project as it has been shown to be very effective at handling large amounts of data and performing well when classifying any minority class(es) [26].

3.1.2 Larger Dataset

The larger dataset details were mentioned previously in the report, and more details can be found in Appendix I. However, some important information regarding the dataset will be covered in the "Exploratory Data Analysis, and Variables" subsection.

This dataset infers the results from the previous experiments done on the smaller datasets, and based on these results, some actions were taken. The full details regarding the experiments will be covered in the "Implementation" Section. For now, these following topics were the main focus of the larger dataset's experiments:

3.1.2.1 GloVe

As mentioned previously, GloVe is a word embedding that uses word co-occurrences in a document to determine the value that is given to a particular word, or token. In addition to this, GloVe as an embedding is usually found in a pre-trained format, with varying dimensions depending on the problem at hand [27]. For this project, the 100 dimensions variant of the pre-trained embedding was used, as the dataset dimensions, and shape were assumed to be too large for the smaller 50 dimensions variant. However, it is important to note that this claim could be proven to be false with further experimentation, but these experiments were not covered in this project.

3.1.2.2 Recurrent Neural Network

RNNs are a type of neural network that are suitable for sentiment analysis problems due to their ability to handle inputs of varying lengths. In addition, this type of network has memory cells to allow it to remember recent words, meaning it can keep track of information as it flows through the network. However, this is also a disadvantage for the RNN as it only captures recent words, and can forget earlier words, meaning it is subject to the vanishing gradient problem, which inhibits its ability to learn texts that are of large lengths [28]. RNN had several variants trained, and optimized as it produced results that were higher than most other architectures utilized for this project.

3.1.2.3 Long-Short Term Memory Network

LSTM Networks are a variant of the Recurrent Neural Network that uses gates to regulate the amount of information that is forgotten or remembered. By doing so, the network is able to remember vital information from earlier on in the text, while learning from the current ingested text. This variant was proposed as a solution to the RNN's vanishing gradient problem [29], and this network is the main network that was optimized and proposed as the solution for the larger dataset's portion of the project.

3.1.2.4 Multilayer Perceptron

MLP is a basic feed-forward neural network architecture consisting mainly of fully connected layers of neurons, that learns through backpropagation [30]. This type of network was trained during a certain period during the project when different architectures were tested for overcoming a certain plateau during training. Only one MLP model was trained as it did not produce results that are better than the current proposed architectures for this project such as RNN and LSTM.

3.1.2.5 BERT

The Bidirectional Encoder Representations from Transformers is a type of network model called a transformer, which is a type of model that utilizes the attention mechanism. Attention is a technique in Deep Learning that allows networks to learn the context behind text in NLP problems. BERT works by reading in input using its encoder and producing a prediction using its decoder [31]. In addition, BERT is a pre-trained model that uses its own word embedding called "ElMo". Much like MLP, BERT was introduced to try to overcome a plateau during the modelling process. However, it was not the best performing model, as such, only one model was trained for it.

3.1.3 Sarcasm Detection

A small part at the end of this project was concerned with attempting to detect sarcasm in text as a way to further analyse the opinions, and sentiments behind text. While several models were trained for this problem, the performance of these models were not satisfactory, as this type of problem was not meant to be originally covered in the report, but now it is a small component that will be covered as part of the wider sentiment analysis umbrella. For this problem, several datasets were collected, and combined in order to generate a large dataset that is suitable for training. Some of the machine learning, and deep learning algorithms that were covered in this report already have been trained on these datasets, and the results will be explained in the next section. For more details regarding the datasets that were used for this problem, Appendix I has more information, and sources for each dataset.

3.2 Exploratory Data Analysis, and Variables

3.2.1 Larger Dataset

In order to improve the performance of the models trained on the larger dataset, the dataset had to be cleaned, and visualized in order to better understand it, and to be able to squeeze more performance out of the models using these insights. Next, a deeper look into these insights will be taken.

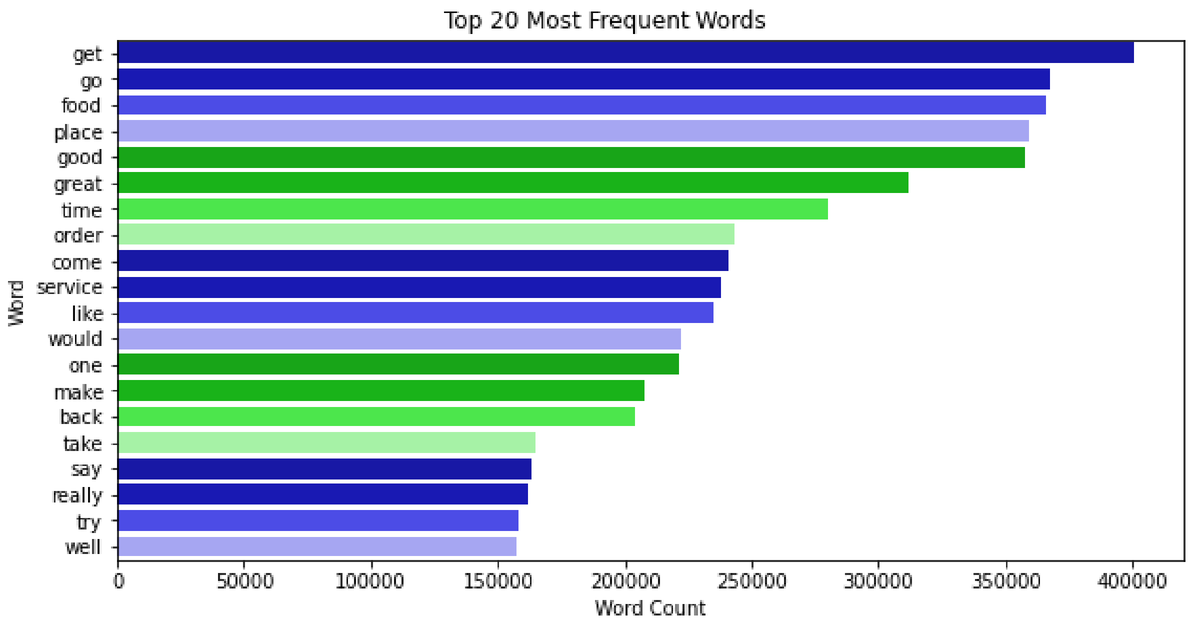


Figure 7 Larger Dataset Most Frequent Words

As shown in **Figure 7**, We can deduce that the top four most frequent words are neutral words, which allows us to understand that most words in a review are going to be neutral by nature. As a result, the data pre-processing pipeline is going to be more effective by removing stop words, and lemmatizing tokens. In addition, after the neutral words, the second common type of sentiment based on this figure is the positive sentiment, which perfectly compliments **Figure 3**, as the most common label for a review is the positive label.

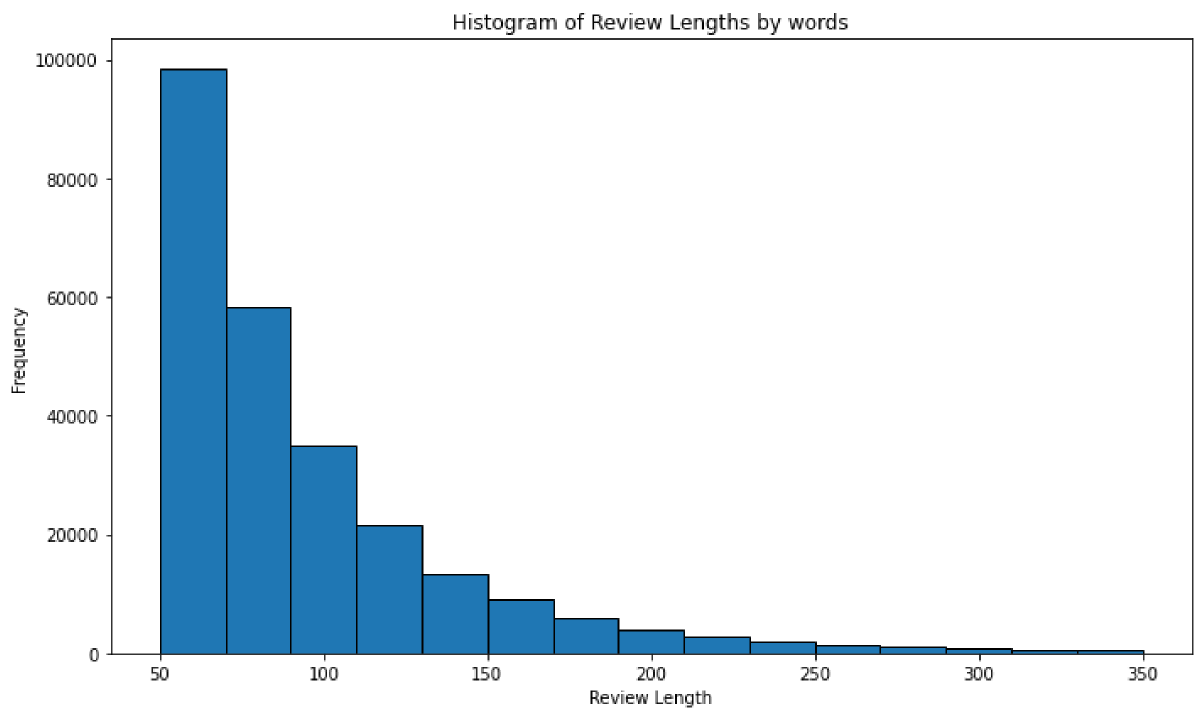


Figure 8 Larger Dataset Review Lengths by Words

As shown in **Figure 8**, Most reviews consist of about 50 words, and the number of words per review steadily decreases beyond that. This indicates that a simple model will be able to achieve a good performance, as being able to classify reviews with around 50 words is more than enough to achieve a high accuracy. In addition, if a model were to be formulated to completely maximize performance, having the model train on reviews with more than 50 words would not increase performance in a significant way.

3.2.2 Smaller Dataset

The same steps taken for the larger dataset were also taken for the smaller dataset to optimize its performance as well.

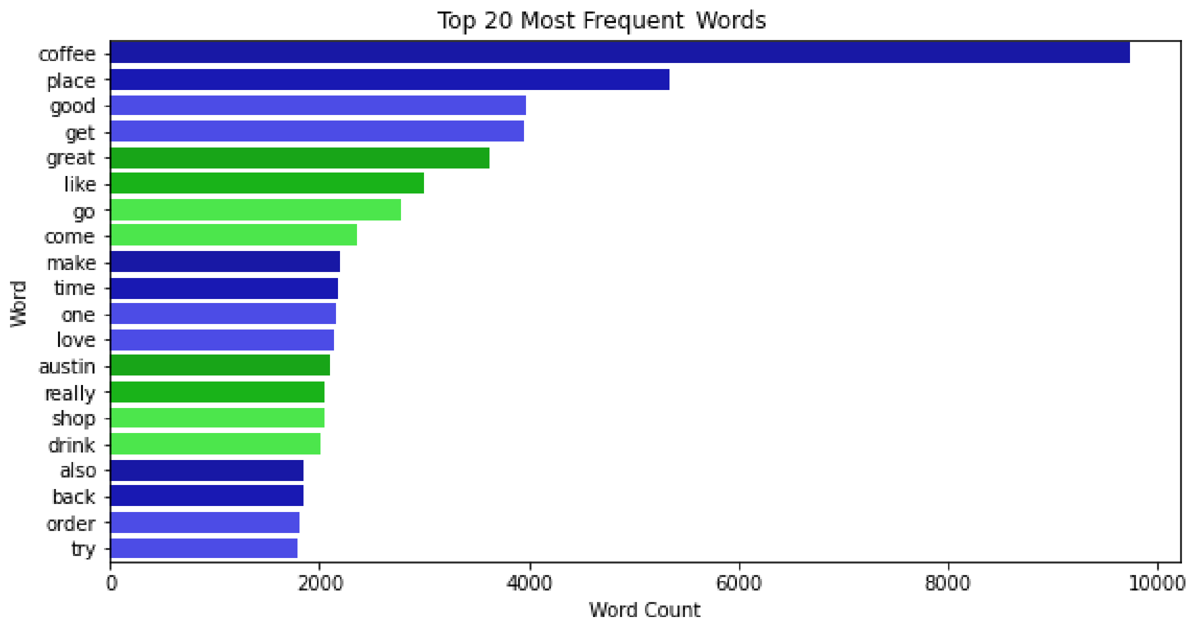


Figure 9 Smaller Dataset Most Frequent Words

As shown in **Figure 9**, the two most common words in the dataset are also neutral words, much like the larger dataset. However, this time the positive label is more showing in this dataset as the third, and fifth most common words in the dataset are positive words. Once again, this insight complements **Figure 2** perfectly, as a large share of the reviews of this dataset are skewed towards the positive label.

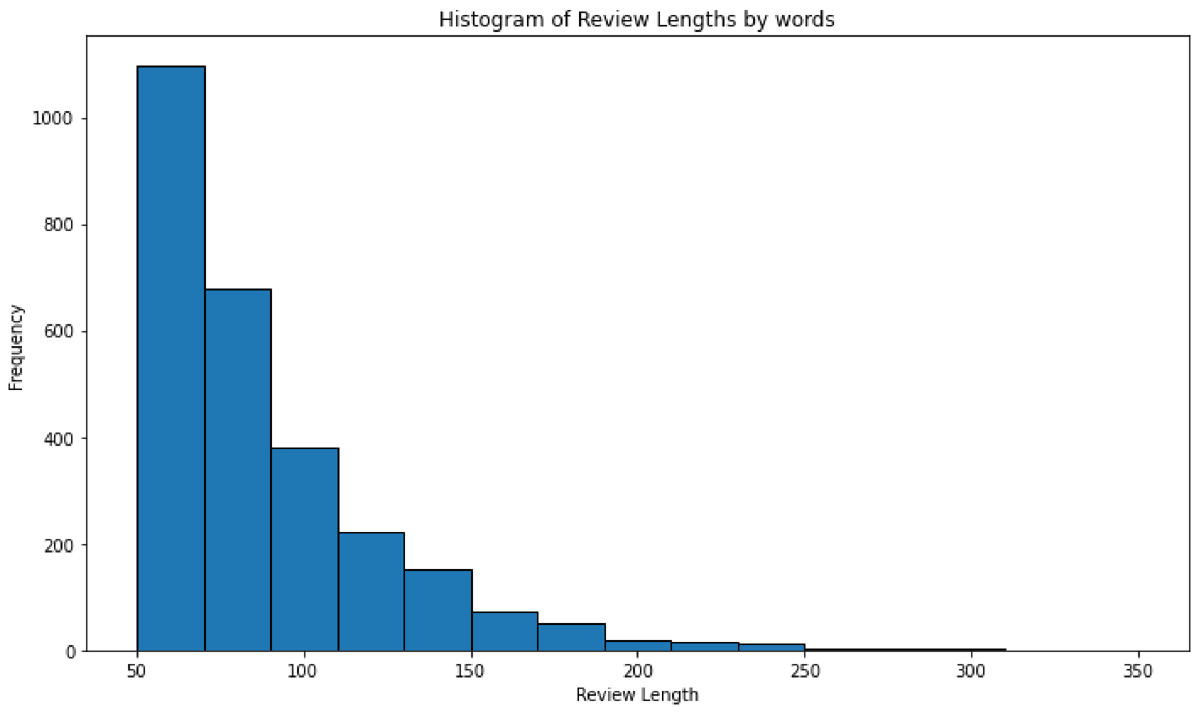


Figure 10 Smaller Dataset Review Lengths by Words

Much like the larger dataset, this dataset also has the most common reviews at around 50 words. However, unlike the larger dataset, this review levels out around at around 200 words per review, meaning that a model attempting to maximize its performance does not have to make an effort to train on longer reviews to improve.

Before moving on to the next section, it is important to note that these findings do not impact the smaller dataset in any way. Furthermore, the maximization of correctly classified reviews, and importance of data labels' distributions are only important, and explored with great detail in the larger dataset, as it has more potential for growth in terms of model performance.

3.3 Environments

These are the following environments that were utilized for the project:

3.3.1 Backblaze

Backblaze is a cloud storage platform for big data professionals. This platform was utilized in this project to source the data to environments that cannot access the data locally such as Vast.ai and Google Colab. In addition, Backblaze was favoured over other popular storage platforms such as Google Drive as it has no limitations on how many requests for sourcing the data can be made for a small fee.

3.3.2 Vast.ai

Vast.ai is a low-cost cloud GPU market for Machine Learning Engineers and Data Professionals. All Deep Learning-based models were trained on Vast.ai as the local machine did not have hardware that can train many models in a reasonable period of time. Also, Vast.ai was favoured over Google Collab Pro as it was more cost effective and provided more resources for the same prices, while providing the Jupyter Notebook environment that was favoured for this project.

3.3.3 Local Machine

The local machine was used to run the notebooks containing code for generating the visualizations for this report, and for training machine learning models that took a long time to train. While the hardware was not ideal, it was a perfect environment for experimentation as the data was sourced directly from the project directory rather than from an online storage platform, which means less time spent waiting for the data to be downloaded, and prepared, and more time spent training models.

3.3.4 Google Collab

Google Collaboratory is a popular online python environment for Data Scientists who wish to collaborate on a specific project. It was utilized in this project for training machine learning models on smaller subsets of the data to experiment with different classifiers, and parameters. While it was proven to be effective for its given tasks, Vast.ai and the local machine were far more utilized and favoured for this project due to the reasons mentioned above.

3.3.5 Jupyter Notebook

The python environment that was used on Vast.ai was using the Jupyter Notebooks environment, which allowed for easier access of documentation, and more productivity by using shortcuts. In addition, Jupyter Notebooks were also used on the local machine in IDEs like Visual Studio Code in order to boost productivity with even more extensions and shortcuts.

3.3.6 Libraries

Python was the main programming language utilized for this project. In addition, many different libraries were used in Python for this project, the most important ones are:

- Sentiment VADER for Sentiment polarity score calculations.
- Sci-kit Learn for training machine learning classifiers.
- TensorFlow, and its high-level API Keras for training deep learning models.
- NLTK for data cleaning and pre-processing of the reviews.
- Matplotlib for generating the visualizations of the dataset.
- NumPy and Pandas for data manipulation and reshaping of the data to prepare it for training.

Note: more libraries than the ones mentioned above were used for the project. A full list of all the libraries utilized for this project will be provided in Appendix II of this report.

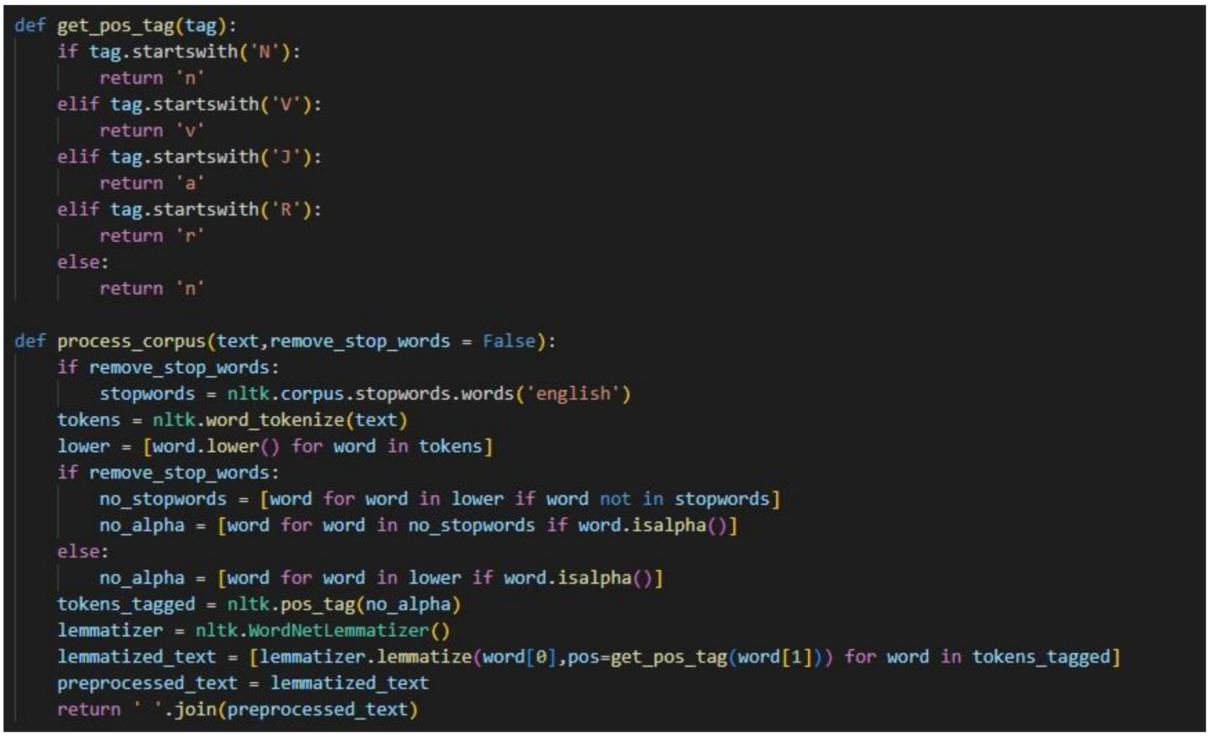
4 Implementation

4.1 Smaller Dataset Experiments and Results

For the smaller dataset, the experiments were conducted by following these steps:

4.1.1 Find a good pre-processing function.

To ensure that the models would produce good results, the data that would go into the models had to be checked for quality and prepared accordingly.



```python
def get_pos_tag(tag):
    if tag.startswith('N'):
        return 'n'
    elif tag.startswith('V'):
        return 'v'
    elif tag.startswith('J'):
        return 'a'
    elif tag.startswith('R'):
        return 'r'
    else:
        return 'n'

def process_corpus(text,remove_stop_words = False):
    if remove_stop_words:
        stopwords = nltk.corpus.stopwords.words('english')
    tokens = nltk.word_tokenize(text)
    lower = [word.lower() for word in tokens]
    if remove_stop_words:
        no_stopwords = [word for word in lower if word not in stopwords]
        no_alpha = [word for word in no_stopwords if word.isalpha()]
    else:
        no_alpha = [word for word in lower if word.isalpha()]
    tokens_tagged = nltk.pos_tag(no_alpha)
    lemmatizer = nltk.WordNetLemmatizer()
    lemmatized_text = [lemmatizer.lemmatize(word[0],pos=get_pos_tag(word[1])) for word in tokens_tagged]
    preprocessed_text = lemmatized_text
    return ' '.join(preprocessed_text)
```

Figure 11 The Pre-processing Functions

After some basic data cleaning is done, the text data is pre-processed before it is transformed into a word embedding format. Furthermore, as shown in **Figure 11**, the pre-processing function used to prepare the data was iterated upon several times before this current optimal one was formulated. It is important to note, that only the sarcasm detection datasets had their stop words kept, and for all sentiment analysis datasets, the stop words were removed, as these words did not improve the model performance, and in some cases like in simple models like machine learning classifiers, were hindering the models' performances.

4.1.2 Find the better base embedding.

First, as previously mentioned, both Word2Vec and TF-IDF were compared using the Multinomial Naïve Bayes Classifier. When put to the test, Word2vec outperformed TF-IDF without any optimizations.

4.1.3 Find the better optimized embedding.

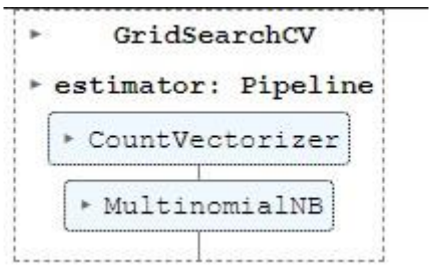
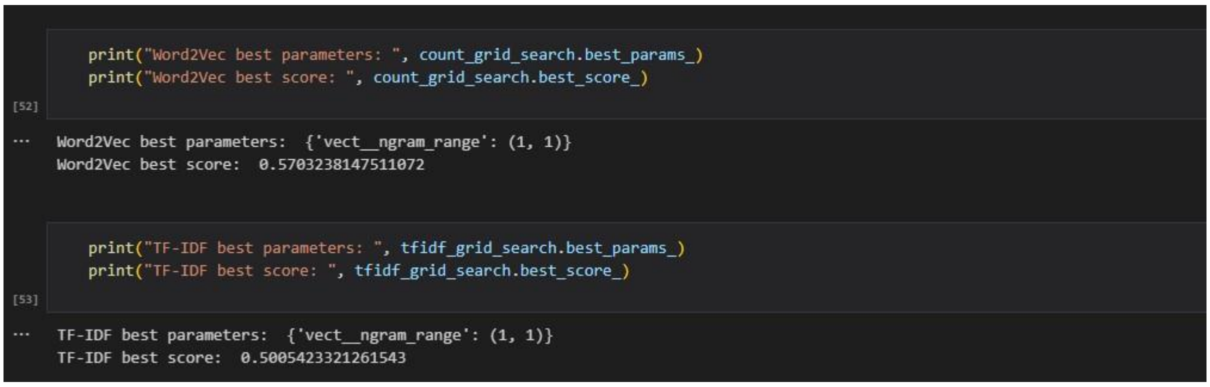


Figure 12 A grid search pipeline

After Word2Vec surpassed TF-IDF in the first test, a pipeline was formed using GridSearchCV in order to tune both word embeddings' parameters. As shown in **Figure 12**, the pipeline consists of each embedding, a separate Naïve Bayes classifier for each, and the "N-gram" parameter that was to be tuned for both embeddings. After tuning, both models were compared, and once again, Word2Vec was still outperforming TF-IDF by a large margin.



```
[52]
print("Word2Vec best parameters: ", count_grid_search.best_params_)
print("Word2Vec best score: ", count_grid_search.best_score_)

Word2Vec best parameters:  {'vect__ngram_range': (1, 1)}
Word2Vec best score:  0.5703238147511072

[53]
print("TF-IDF best parameters: ", tfidf_grid_search.best_params_)
print("TF-IDF best score: ", tfidf_grid_search.best_score_)

TF-IDF best parameters:  {'vect__ngram_range': (1, 1)}
TF-IDF best score:  0.5005423321261543
```

Figure 13 Word2Vec outperforming TF-IDF

As seen in **Figure 13**, Word2Vec achieved an accuracy score of 57%, while TF-IDF only achieved an accuracy score of 50%. This concluded the first word embedding comparison with Word2Vec in the lead.

4.1.4 Find the best data resampling technique.

Resampling Technique	Accuracy Scores
Random Oversampler	0.56
Random Undersampler	0.42
SMOTE	0.57

Table 1 Resampling Techniques' Results for Word2Vec

As seen in **Table 1**, the best resampling technique for Word2Vec was SMOTE. Not only that, but SMOTE is also very suitable for the smaller dataset as there are not that many rows, so oversampling techniques will not take too long to run on this dataset.

4.1.5 Find the best machine learning classifier.

Classifier	Accuracy Score
Multinomial Naïve Bayes (Optimized)	0.59
Random Forest	0.71
Support Vector Machine	0.62
Bagging KNN	0.54
Bagging MNNB	0.58
Adaboost MNNB	0.52
XGBoost	0.85

Table 2 Smaller Dataset Machine Learning Classifiers Results

As shown in **Table 2**, XGBoost was the greatest performing model by a large margin. Furthermore, the second best model was Random Forest, and it was behind by a 14% accuracy score gap.

4.1.6 Compare the best machine learning classifier with the sentiment polarity score.

The library used for generating a sentiment polarity score is called "SentimentIntensityAnalyzer", and it was provided by NLTK. The sentiment polarity score came out to be 82%, which is a massive victory for XGBoost, not only because its accuracy score is better, but also because XGBoost was trained on 5 labels, whereas the SentimentIntensityAnalyzer was trained on only 3 labels, as it can only predict positive, negative, and neutral classes. In order to make a fair comparison, XGBoost was trained on only three labels. The labels in the dataset were converted from a 5-label format to a 3-label format. An important thing to note here, is that this technique will come in handy when the largest dataset section is analysed.

Classifier	Accuracy Score
XGBoost (3-labels)	0.85
SentimentIntensityAnalyzer	0.82

Table 3 XGBoost vs SentimentIntensityAnalyzer

According to **Table 3**, even when XGBoost's labels are downgraded, it received no improvement in its accuracy scores. However, it was still superior to the SentimentIntensityAnalyzer, which is building a case that a more advanced approach than a sentiment polarity calculation is necessary to achieve better predictions of sentiment.

4.1.7 Smaller Dataset Conclusion

it is important to take note of the following when interpreting these results:

1. This dataset is too small to draw a definitive conclusion on which resampling technique is best. We can only say that the experiments suggest the produced results, not definitively proves them to be true.
2. The imbalance in the dataset is very severe, which works better in favour of SMOTE, as it is better at replicating synthetic labels for minority classes. The other sampling techniques might be just as effective on the larger dataset, but oversampling techniques cannot be tested on the largest dataset as it is already too big.
3. Random Oversampling outperformed SMOTE on other metrics such as F1-Score and Precision, showing a trend that oversampling techniques were the superior option for this type of dataset.
4. XGBoost was heavily favoured from the start, as it is famous for its ability to navigate imbalanced, high dimensional datasets without adhering to the effects of noise.

4.2 Larger Datasets Model Progression

Next, for the larger dataset, the following experiments were conducted by taking these steps:

4.2.1 Data Preparation

For the larger dataset, most of the model variants were trained on a subset of the data for optimization, and the best performing variant would be trained on the whole dataset. The subset of the dataset was chosen to be about 700000 rows of data, which was very fitting for deep learning models as they require large amounts of data for training. In terms of the imbalanced data distribution, a solution was proposed to give the classes varying weights according to their distribution. This means that, the more prevalent a class was in the dataset, the lower it would receive when accurately predicted. Furthermore, the metrics used for evaluating the models would be F1-Score, as accuracy would falsely represent, and inflate the actual performance measure of the model.

4.2.2 Word2Vec vs GloVe

After fully preparing the larger dataset, Word2Vec was put in a direct comparison with GloVe to determine which embedding would used for the rest of the project. In order to determine the better embedding, two RNN models were trained on a subset of the larger dataset, and each one used an embedding layer made from the embedding matrices of each embedding. However, both models would end up achieving the exact same F1-Score of 73%. In the end, GloVe was chosen as the embedding of choice for the majority of the future models, however Word2Vec would be revisited one more time during the LSTM experiments in order to examine its performance on this dataset.

4.2.3 Machine Learning Classifiers and Custom Under Sampling Technique (CUST)

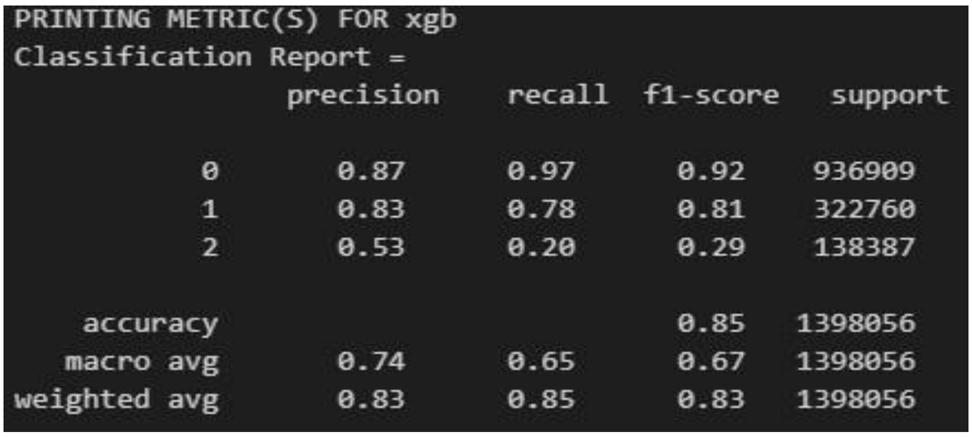
Classifier	Accuracy Score
Random Forest	0.51
Logistic Regression	0.69
Multinomial Naïve Bayes	0.62
XGBoost	0.70

Table 4 Machine Learning Classifiers Results on the CUSF Balanced Dataset

Classifier	Accuracy Score
Logistic Regression	0.87
Multinomial Naïve Bayes	0.79
XGBoost	0.84

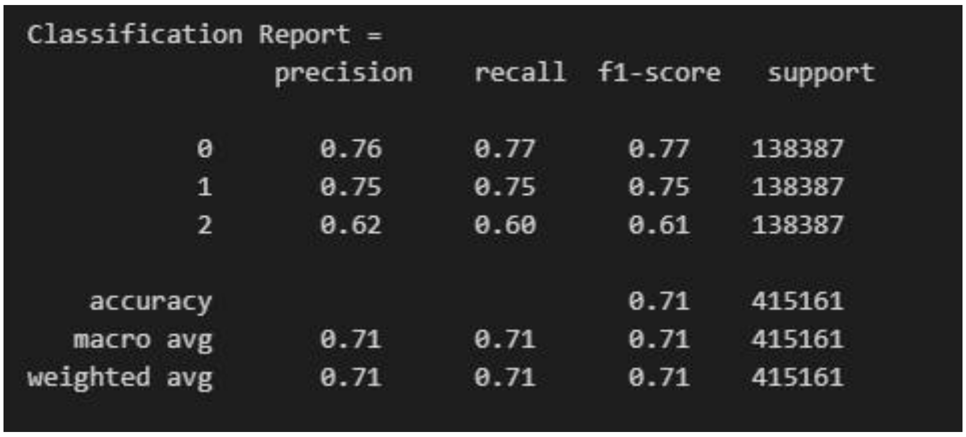
Table 5 Machine Learning Classifiers Results on the SMOTE balanced Dataset

Custom Under Sampling Technique (CUST) is a function in the codebase of this project that was formulated during the process of optimizing the hyperparameters of an LSTM variant. CUST was created fix the biggest issue with the datasets balanced using the “imblearn” library (library that contains random over sampler, random under sampler, and SMOTE), which was that although the data was balanced, the models were biased predicting certain classes over others. It is very important to note, that while the larger dataset contained 6990280 rows, after using CUST, the dataset was brought down to 2075683 rows to balance it out.



```
PRINTING METRIC(S) FOR xgb
Classification Report =
               precision    recall  f1-score   support

           0       0.87      0.97      0.92    936909
           1       0.83      0.78      0.81    322760
           2       0.53      0.20      0.29    138387

    accuracy                           0.85   1398056
   macro avg       0.74      0.65      0.67   1398056
weighted avg       0.83      0.85      0.83   1398056
```

Figure 14 XGBoost Classification Report after SMOTE Balancing



```
Classification Report =
               precision    recall  f1-score   support

           0       0.76      0.77      0.77    138387
           1       0.75      0.75      0.75    138387
           2       0.62      0.60      0.61    138387

    accuracy                           0.71    415161
   macro avg       0.71      0.71      0.71    415161
weighted avg       0.71      0.71      0.71    415161
```

Figure 15 XGB Classification Report after CUST Balancing

As shown in **Table 4**, and **Table 5**, SMOTE balanced datasets were achieving much higher accuracy scores than the CUST balanced datasets, especially with XGBoost. However, as shown in **Figure 14**, and **Figure 15**, The accuracy might be higher for SMOTE balanced datasets, but the F1-Score for the neutral sentiment class in the CUST balanced XGBoost model is more than twice that of the F1-Score for the neutral sentiment class in the SMOTE balanced XGBoost model. This means that although CUST does not achieve a higher accuracy score, it ensures that the lower accuracy score model is not biased towards any certain class. It is vital to mention that CUST will be mentioned again in the trained LSTM variants, however it is imperative that the CUST is used in place of traditional resampling techniques, or other unbalanced classes solutions such as class weights, as these techniques were shown to ignore certain classes in a dataset.

4.2.4 Deep Learning Architectures

4.2.4.1 RNN

RNN Variant Number	Labels Count	F1-Score
Base Model	3	0.74
Variant 1	3	0.62
Variant 2	5	0.59
Variant 3	3	0.73

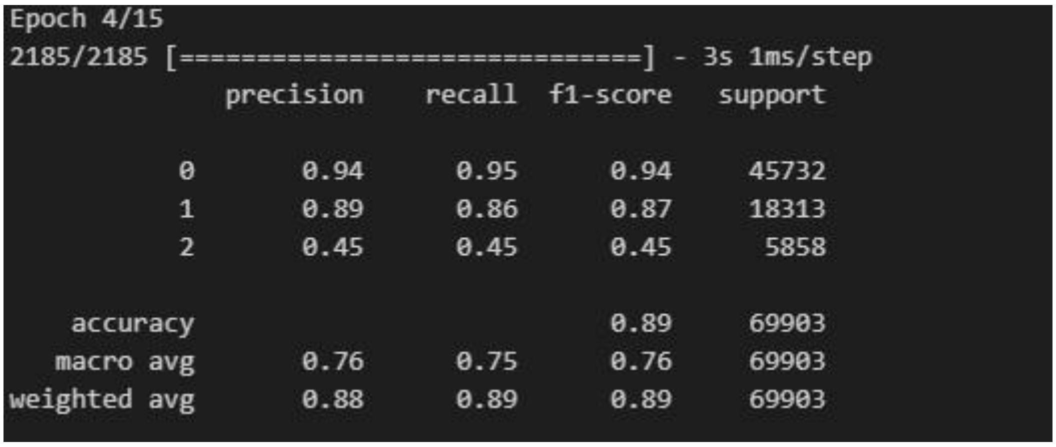
Table 6 RNN Unbalanced Dataset Variants Results

RNN Variant Number	Labels Count	Accuracy
Variant 4	3	0.72
Variant 5	3	0.73
Variant 6	3	0.76

Table 7 RNN Balanced Dataset Variants Results

As shown in **Table 6**, and **Table 7**, The RNN architecture had to go through several iterations before improvements massive improvements were made.

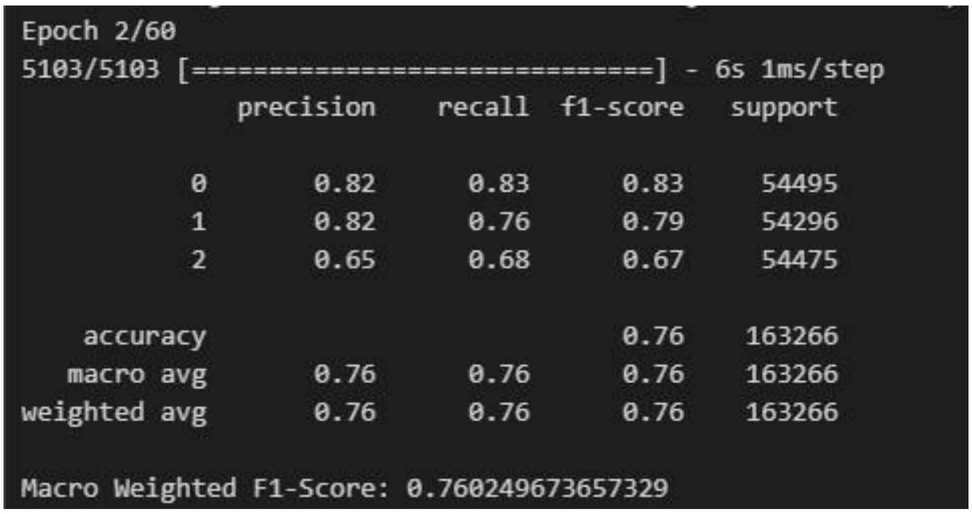
First, the base model, variants one, two, and three were all done on the previously mentioned subset of the larger dataset, and used the SimpleRNN architecture from TensorFlow. Although these models were reaching very high accuracies, and Macro-weighted F1-scores of about 88%, the Macro-F1 was showing that the model was very good at predicting the positive sentiment, and the negative sentiment, but the same model was also very bad at predicting the neutral sentiment.



```
Epoch 4/15
2185/2185 [==============================] - 3s 1ms/step
              precision    recall  f1-score   support

           0       0.94      0.95      0.94     45732
           1       0.89      0.86      0.87     18313
           2       0.45      0.45      0.45      5858

    accuracy                           0.89     69903
   macro avg       0.76      0.75      0.76     69903
weighted avg       0.88      0.89      0.89     69903
```

Figure 16 Example of Neutral Sentiment Scores on Unbalanced Datasets

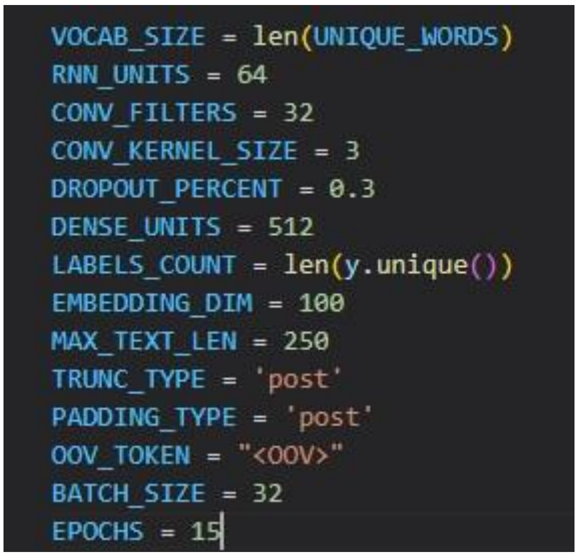
As shown in **Figure 16**, there exists a massive gap between the predicted scores of the first two labels, and the third label. Although by using class weights, the model was able to predict the neutral sentiment with an F1-Score of 45%, that is still a massive gap that proves that the model will not be able to predict the neutral class as accurately as it does for the other classes.



```
Epoch 2/60
5103/5103 [==============================] - 6s 1ms/step
              precision    recall  f1-score   support

           0       0.82      0.83      0.83     54495
           1       0.82      0.76      0.79     54296
           2       0.65      0.68      0.67     54475

    accuracy                           0.76    163266
   macro avg       0.76      0.76      0.76    163266
weighted avg       0.76      0.76      0.76    163266

Macro Weighted F1-Score: 0.760249673657329
```

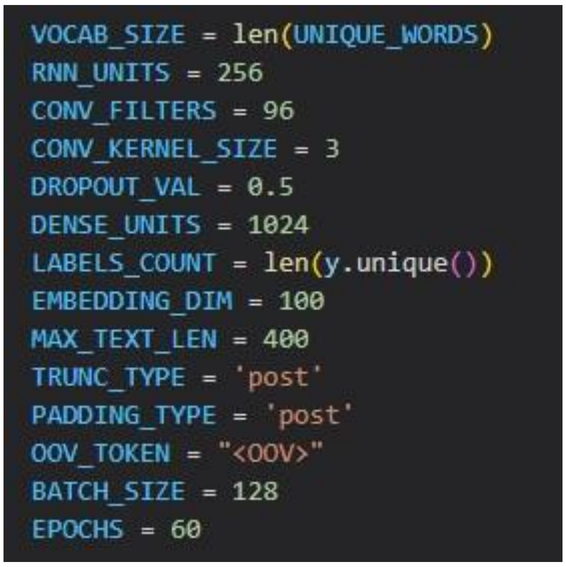
Figure 17 RNN Variant 6 With CUST Balancing

In **Table 7**, the models use accuracy as a metric as the data was balanced using CUST. In **Figure 17**, the accuracy, and macro-weighted F1-score of the model, and the first two classes went down, but the F1-score of the neutral class went up by over 20%. While this means that the model will have to be optimized differently after being CUST balanced, the model is also getting better at detecting different classes. Overall, this is seen as a massive improvement as this minority class's performance has increased significantly when compared to previously, when it was not even at an acceptable level.



```
VOCAB_SIZE = len(UNIQUE_WORDS)
RNN_UNITS = 64
CONV_FILTERS = 32
CONV_KERNEL_SIZE = 3
DROPOUT_PERCENT = 0.3
DENSE_UNITS = 512
LABELS_COUNT = len(y.unique())
EMBEDDING_DIM = 100
MAX_TEXT_LEN = 250
TRUNC_TYPE = 'post'
PADDING_TYPE = 'post'
OOV_TOKEN = "<OOV>"
BATCH_SIZE = 32
EPOCHS = 15
```

Figure 18 RNN Base Model Hyperparameters



```
VOCAB_SIZE = len(UNIQUE_WORDS)
RNN_UNITS = 256
CONV_FILTERS = 96
CONV_KERNEL_SIZE = 3
DROPOUT_VAL = 0.5
DENSE_UNITS = 1024
LABELS_COUNT = len(y.unique())
EMBEDDING_DIM = 100
MAX_TEXT_LEN = 400
TRUNC_TYPE = 'post'
PADDING_TYPE = 'post'
OOV_TOKEN = "<OOV>"
BATCH_SIZE = 128
EPOCHS = 60
```

Figure 19 RNN Variant 6 Model Hyperparameters

The hyperparameters for the base model, and variant 6 are provided in Figure 18, and Figure 19. Although every hyperparameter in both figures changed except the ones concerned with preparing the data for the model, the most significant change was the increase in the "MAX_TEXT_LEN" hyperparameter from 250 to 400. This was done to make sure that the reviews were padded to the length of the longest review as shown in Figure 8. Afterwards, the accuracy rose from the previous variant's 73%, to 76%. As a result, future models had this hyperparameter tested several times, and the results were not unanimous, but for the most trained architectures of RNN and LSTM, it was shown that having this particular hyperparameter at 400 was effective in raising the model's performance.

4.2.4.2 MLP and BERT

Model	Accuracy Score
Multilayer Perceptron	0.73
Bidirectional Encoder Representations from Transformer	0.75

Table 8 MLP and BERT Results

As shown in **Table 8**, BERT outperformed MLP by only 2% in terms of accuracy scores. This is because the BERT model that was used for this project is the older version that was proposed back in 2017, and the model is completely unoptimized, whereas MLP was somewhat optimized during the LSTM variants experiments.

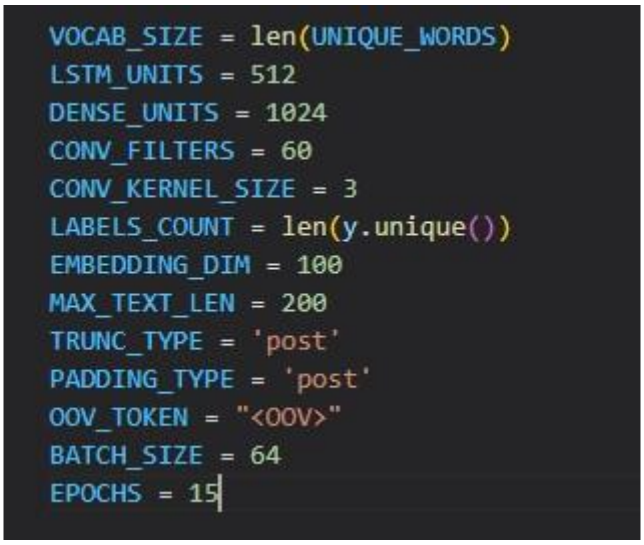
4.2.4.3 LSTM Variants Progression

It would be clear by now that the model architecture that received the most attention was LSTM. This is because the base model for LSTM was effective in reaching a high base model performance than the other architectures. Because of this, many LSTM model variants were trained, and many different experiments and changes were done to achieve the highest possible performance.

A few things are important note here:

1. Most LSTM variants use a 1D Convolutional layer in order to reduce noise in the text data, and to extract relevant features from it.
2. A dropout layer was included in every LSTM variant to prevent overfitting.
3. Fully Connected Layers were included in every LSTM variant in order to learn the nuanced relationships in the data.
4. A Batch Normalization layer was added to later LSTM variants to allow the best performing optimizer (Adam) to use high learning rates, and a Normalization layer was added to fit the data into a smaller range, and prevent a complex word embedding like GloVe from adding large values to certain words in text that can confuse the model.

4.2.4.3.1 LSTM Base, and Variants 1-13



```
VOCAB_SIZE = len(UNIQUE_WORDS)
LSTM_UNITS = 512
DENSE_UNITS = 1024
CONV_FILTERS = 60
CONV_KERNEL_SIZE = 3
LABELS_COUNT = len(y.unique())
EMBEDDING_DIM = 100
MAX_TEXT_LEN = 200
TRUNC_TYPE = 'post'
PADDING_TYPE = 'post'
OOV_TOKEN = "<OOV>"
BATCH_SIZE = 64
EPOCHS = 15
```

Figure 20 LSTM Base Model Hyperparameters

LSTM Variant Number	Labels Count	Macro F1-Score
Base	3	0.70
Variant 1	5	0.61
Variant 2	5	0.51
Variant 3	3	0.74
Variant 4	3	0.69
Variant 5	3	0.67
Variant 6	3	0.70
Variant 7	3	0.75
Variant 8	3	0.67
Variant 9	3	0.75
Variant 10	3	0.74
Variant 11	3	0.71
Variant 12	3	0.71
Variant 13	3	0.74

Table 9 LSTM Models Results Base + Variants 1-13

For LSTM, some of the experiments' results of the RNN architectures were used to influence the early variants of the LSTM models. However, after some models were trained, the results from the RNN models were not used very often, as the LSTM architectures required more empirical results, and experimentation in order to optimize them.

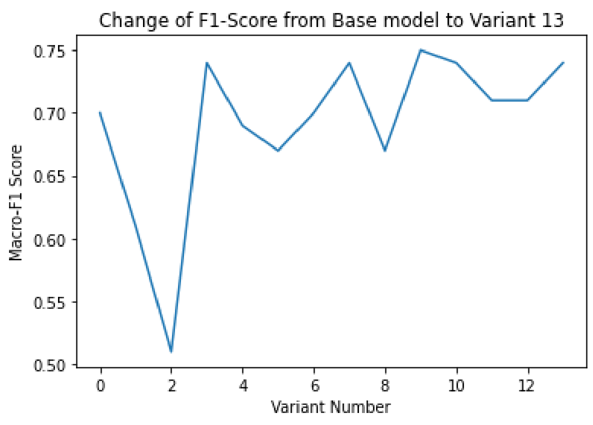


Figure 21 LSTM Base Model + Variants 1-13 Results Visualized

By looking at **Table 9**, and Figure 21, The changes that occurred by altering hyperparameters were very volatile, and spanned a wide range of values. To expand on this further, the difference between the 3-label models spanning from the highest performing model (variant 7), and the lowest performing model (variant 5) is 8%. This means that, for the next variants, any hyperparameter that changes the results for better or for worse, would have been either reverted back to its original value, or kept constant in order to avoid it interfering with the results of changing other hyperparameters. In addition, from Figure 21, a deduction could be made that only 5 out of the 13 variants was able to reach an F1-Score that is above the base model. A conclusion can be drawn from this that this approach of tuning hyperparameters to achieve a higher performance is not effective, and novel approaches will have to be taken in order to surpass the highest performing model from that category.

4.2.4.3.2 LSTM Variants 14-27

LSTM Variant Number	Labels Count	Accuracy
Variant 14	5	0.73
Variant 15	5	0.72
Variant 16	3	0.70
Variant 17	3	0.75
Variant 18	3	0.75
Variant 19	3	0.76
Variant 20	3	0.77
Variant 21	3	0.76
Variant 22	3	0.74
Variant 23	3	0.77
Variant 24	3	0.77
Variant 25	3	0.73
Variant 26	3	0.73
Variant 27	3	0.77

Table 10 LSTM Models Results Variants 14-27

In variants 14, and onward, there were two different approaches taken:

5. A larger subset of the data was used in order to prevent overfitting.
6. CUST was applied on the datasets before training.

7. Different Types of LSTMs were experimented with (Unidirectional LSTM, Bidirectional LSTM, CuDNN LSTM)

These changes allowed the model to achieve a higher performance than what was previously documented, and the models' F1-scores for the neutral class rapidly improved.

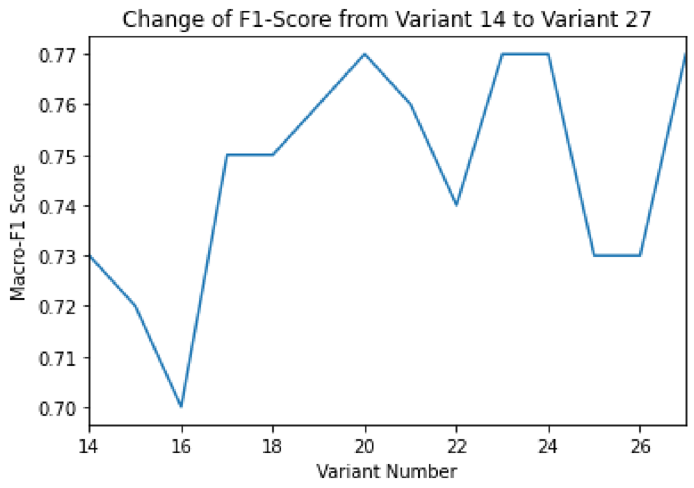


Figure 22 LSTM Variants 14-27 Results Visualized

Looking at **Table 10**, and **Figure 22**, it is clear that CUST for data balancing was very effective at improving the models' performances. Not only did the best performing model surpass the previous category of variants (variant 27), but 9 out of the 13 models were higher than variant 14, which was considered as the base model for models with CUST done on their datasets.

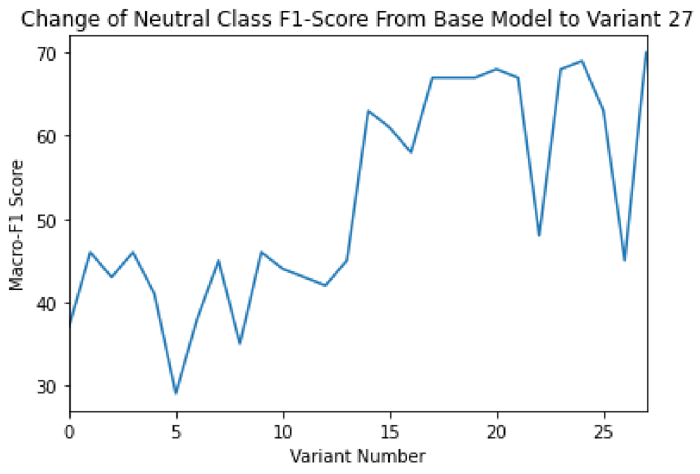


Figure 23 F1-Score of Neutral Class for Each Variant Visualized

Furthermore, the neutral class's F1-Score increased significantly as shown in Figure 23, which made the models more capable of distinguishing between the three classes. In fact, in the previous figure, the exact variant in which CUST was used can be seen without any indication from the plot, as at that point, the F1-score of the neutral class rises to a completely different level.

4.2.5 Macro F1-Score vs Macro-Weighted F1-Score

Before moving on to Sarcasm Detection, it is important to mention why the metrics used in the previous section were important.

First, F1-Score must be explained. F1-Score is calculated by calculating the harmonic mean between the Recall, and the Precision [34]. Whereas Macro F1-Score is the mean of all F1-Scores calculated for each class without taking class imbalance into consideration. However, Macro Weighted F1-Score is the Macro F1-Score but with consideration for class imbalances.

Macro F1-Score simply calculates the unweighted mean for all classes, but Macro Weighted F1-Score will assign a weight to each F1-Score of each class based on the number of correct classifications it received. This means that the positive, and negative classes of the dataset would always inflate the metric's scores, as the dataset would be skewed towards them in terms of label counts. This explains why before using CUST, the accuracy, and Macro Weighted F1-Scores of the models were high, but the Macro F1-Score of the models were low, as there were less instances of the neutral class, and even though class weights were assigned, the class was still too underrepresented in the dataset to improve beyond a certain point.

Therefore, it is determined that Macro Weighted F1-Score is unsuitable for this type of project, as the class imbalances should not affect the F1-Score. In fact, Macro F1-Score is way more suitable as it allows for the model's bias towards certain classes to show, and affect the overall metric's scores.

4.2.6 Why CUST was better than traditional resampling techniques

The reason why CUST was able to improve beyond imblearn's random under sampler is because of the nature of the larger dataset itself.

The random under sampler provided by imblearn samples the data down to the minority class, which is the class with the least amount of samples. However, for the larger dataset, the neutral class is not the minority class in many cases.

In the larger dataset, as one may notice from Figure 3, the minority class was actually the 2-star rating. This means that, although the models were having trouble predicting the minority class, the minority class was still being under sampled down further, instead of under sampling the data down to its samples. This results in fewer overall samples for the neutral class, and the class remains underrepresented, even if the data is balanced on paper.

To solve this problem, CUST was developed in order to check for the class that is the most underrepresented, and to sample the data down to that class, instead of the minority class. While this means that the class which was under sampled down to will be more represented, this also has the downside of misrepresenting other classes that have less samples than the under sampled class. However, for this case, there was no issue detecting the other underrepresented classes compared to the neutral class, as the other underrepresented classes were either the 2-star rating, which was easy to detect as reviews that are labeled two star are more likely to be similar to the 1-star reviews, or the 1-star reviews, which were easy to detect as reviews are more likely to contain negative words, than positive words, leading to the review to be classified as negative without any confusion from the model. In addition, it was unlikely that the model would overfit and be biased towards the majority class, as the model architecture was built from the ground up to contain safeguards for this type of situation, such as dropout layers, convolutional layers, and a high value of LSTM units.

In addition, through several experiments done on multiple variants, random under sampler was not able to remedy the issue of the model's bias towards the first 2 classes, because detecting if a review was neutral or not was already a difficult task for the model. Therefore, by using CUST, even if the model starts to biased towards predicting the neutral class over other classes, the model would still predict the other classes reasonably well as for this particular problem, the differences between the positive labels, and the negative labels is already wide enough for the model to detect, and this can be shown by examining certain variants that used either random over sampling, or random under sampling like variant ten, and variant twelve as examples.

4.2.7 Sentiment Analysis Implementation Wrap up

Overall, the process of detecting sentiment in text was shown to be much better when using deep learning techniques such as RNNs and LSTMs. In addition, when combining these architectures with techniques such as CUST, the performance of the models to detect underrepresented classes increases greatly. While these results have been great, further experimentation is still required in order to achieve excellent results using these techniques.

4.3 Sarcasm Detection

Sarcasm Detection is a binary classification problem that works by attempting to break down text data and detect if the person who wrote it was ironic, and was trying to convey the opposite, or a different meaning to what is actually written. This problem was not solved with full focus for this project; however, it is important to mention that several experiments were done to achieve a high performance for the models trained to solve this problem.

4.3.1 Dataset

When detecting a suitable dataset for this problem, a suggested approach was to gather multiple datasets about this problem and combine them together to form a dataset with many different topics but are all under the same problem umbrella. Furthermore, the datasets that were used were the

MUStARD Dataset (text only), News Headlines Dataset, and the Reddit Dataset. Finally, the sources for each dataset will be provided in Appendix I, and more details can be found there as well. For data cleaning and pre-processing, all datasets were combined, and cleaned to provide a whole dataset consisting of 997438 rows of data, that is labelled 0 and 1, corresponding to non-sarcastic, and sarcastic classes respectively.

4.3.2 Machine Learning Approach

For machine learning, Gaussian Naïve Bayes (MNNB but for binary classification), Logistic Regression, Random Forest, and XGBoost were all used exclusively for this problem.

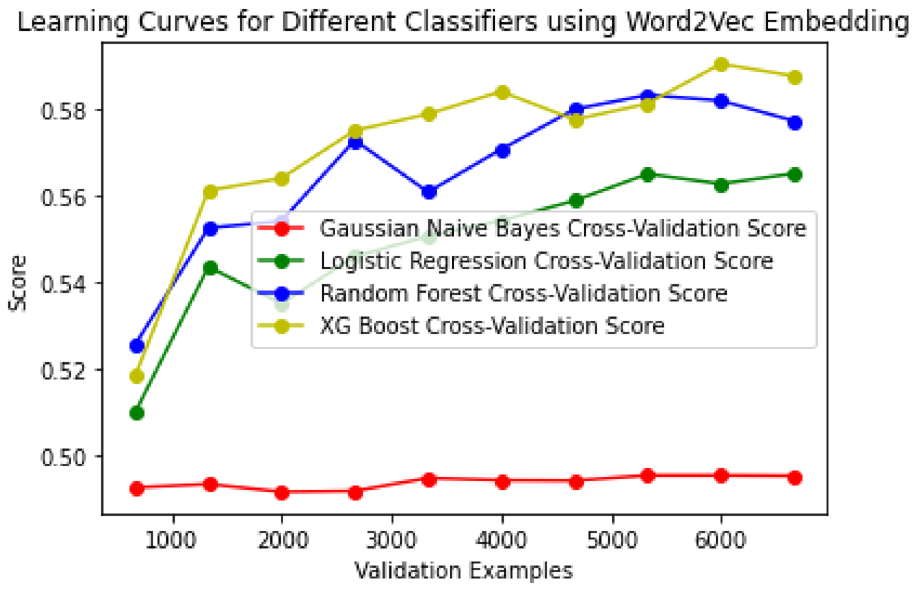


Figure 24 Word2Vec Embedding Used by All Classifiers Results

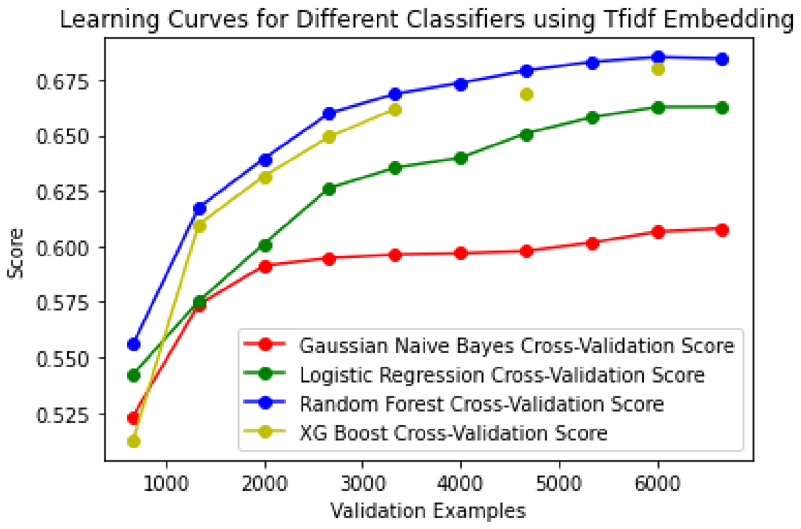


Figure 25 TF-IDF Embedding Used by All Classifiers Results

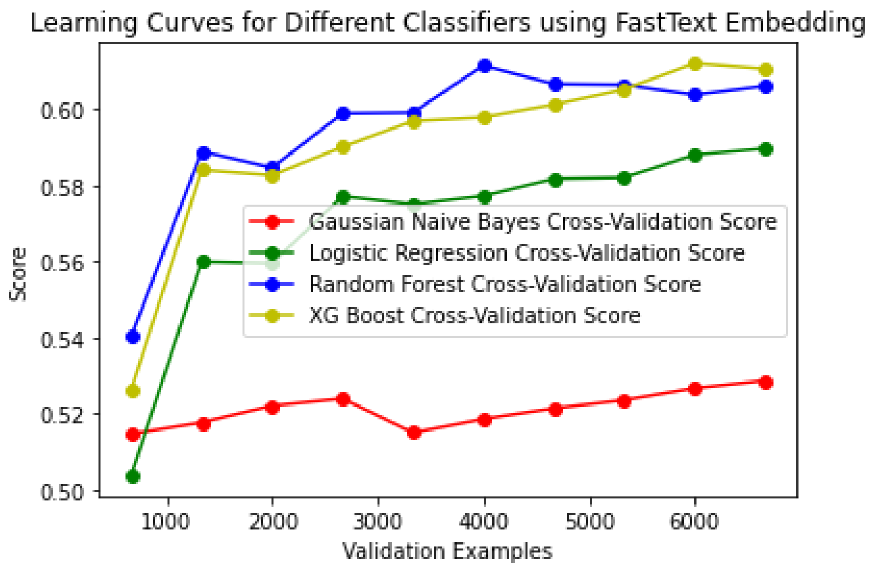


Figure 26 FastText Embedding Used by All Classifiers Results

Based on **Figure 24**, **Figure 25**, and **Figure 26**, it is clear that TF-IDF is the best performing word embedding for sarcasm detection.

Classifier	Word2Vec	TF-IDF	FastText
Gaussian Naïve Bayes	0.50	0.60	0.51
Logistic Regression	0.58	0.68	0.60
Random Forest	0.60	0.71	0.60
XGBoost	0.59	0.71	0.59

Table 11 Machine Learning Classifier Results for Sarcasm Detection

Deducing from **Table 11**, XGBoost using TF-IDF was the best performing model for Machine Learning.

4.3.3 Deep Learning Approach

For deep learning, RNN, and LSTM were used as these architectures have proven to be successful on the previous problem, and therefore an assumption was made that these architectures would be effective for this problem. However, further experimentation is required in order to find the most optimal architecture.

Deep Learning Architecture	Accuracy Score
RNN Base	0.66
LSTM Base	0.67
LSTM Variant 1	0.71
LSTM Variant 2	0.70

Table 12 Deep Learning Classifier Results for Sarcasm Detection

Based on the results from **Table 12**, the best performing deep learning model, and the best model overall for sarcasm detection, was LSTM Variant 1.

5 Results and Evaluation

5.1 Results

For this section, the models from the previous section were compared, and the best performing models are presented here.

5.1.1 Sentiment Analysis

Sentiment Analyser	Score
Larger Dataset Full (SMOTE)	0.75
Larger Dataset Full (CUST)	0.44

Table 13 Sentiment Polarity Scores

Model Name	ML/DL	Accuracy	Macro F1-Score	Label Count	Dataset
XGBoost (SMOTE)	ML	0.85	0.67	5	Larger All
Bagging Multinomial Naïve Bayes (SMOTE)	ML	0.80	0.68	3	Larger All
Logistic Regression (SMOTE)	ML	0.87	0.71	3	Larger All
LSTM Variant 7 (CLASS WEIGHTS)	DL	0.89	0.75	3	Larger Subset
LSTM Variant 27 (CUST)	DL	0.77	0.77	3	Larger All

Table 14 Best Models for Sentiment Analysis

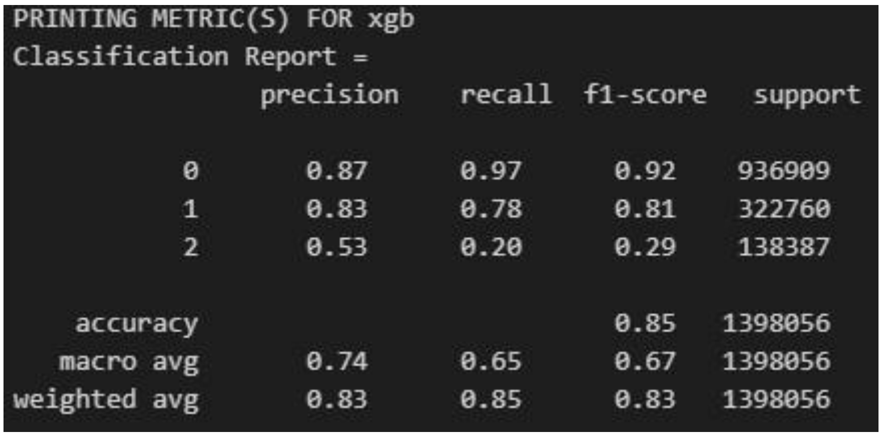
5.1.2 Sarcasm Detection

Model Name	ML/DL	Accuracy	Label Count	Dataset
XGBoost	ML	0.71	3	Sarcasm Subset
LSTM Variant 1	DL	0.71	3	Sarcasm All

Table 15 Best Models for Sarcasm Detection

5.1.3 Summary of Best Models for Sentiment Analysis

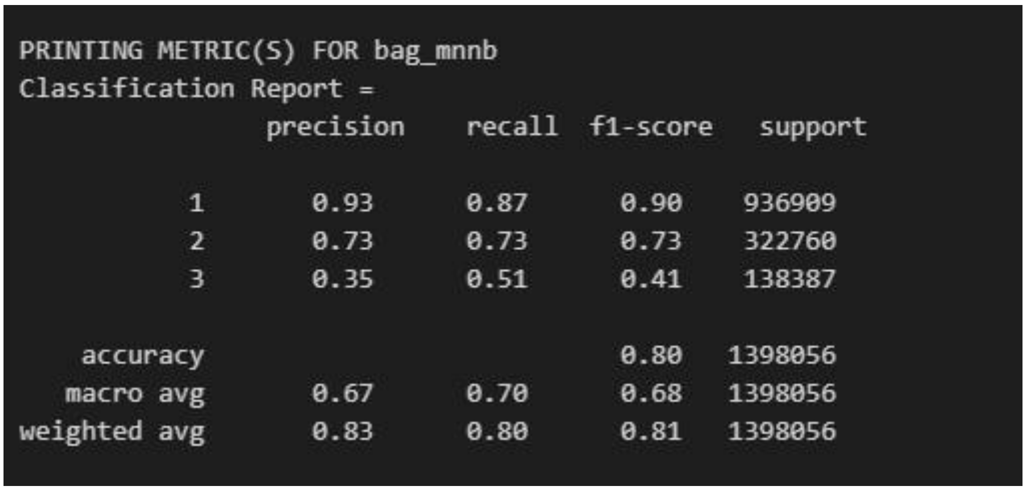
5.1.3.1 XGBoost (SMOTE)



```
PRINTING METRIC(S) FOR xgb
Classification Report =
               precision    recall  f1-score   support

           0       0.87      0.97      0.92    936909
           1       0.83      0.78      0.81    322760
           2       0.53      0.20      0.29    138387

    accuracy                           0.85   1398056
   macro avg       0.74      0.65      0.67   1398056
weighted avg       0.83      0.85      0.83   1398056
```

Figure 27 XGBoost with SMOTE Classification Report

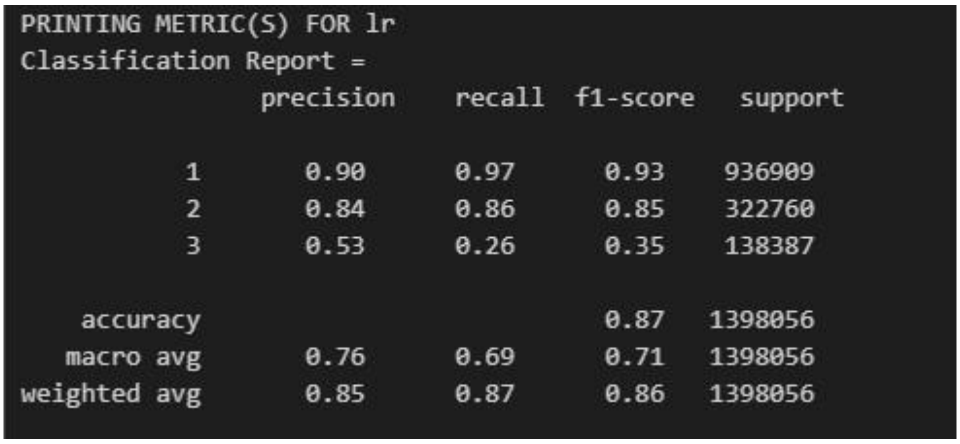
5.1.3.2 Bagging Multinomial Naïve Bayes (SMOTE)



```
PRINTING METRIC(S) FOR bag_mnnb
Classification Report =
               precision    recall  f1-score   support

           1       0.93      0.87      0.90    936909
           2       0.73      0.73      0.73    322760
           3       0.35      0.51      0.41    138387

    accuracy                           0.80   1398056
   macro avg       0.67      0.70      0.68   1398056
weighted avg       0.83      0.80      0.81   1398056
```

Figure 28 Bagging Multinomial Naive Bayes (SMOTE) Classification Report

5.1.3.3 Logistic Regression (SMOTE)



```
PRINTING METRIC(S) FOR lr
Classification Report =
               precision    recall  f1-score   support

           1       0.90      0.97      0.93    936909
           2       0.84      0.86      0.85    322760
           3       0.53      0.26      0.35    138387

    accuracy                           0.87   1398056
   macro avg       0.76      0.69      0.71   1398056
weighted avg       0.85      0.87      0.86   1398056
```

Figure 29 Logistic Regression (SMOTE) Classification Report

5.1.3.4 LSTM Variant 7 (CLASS WEIGHTS)

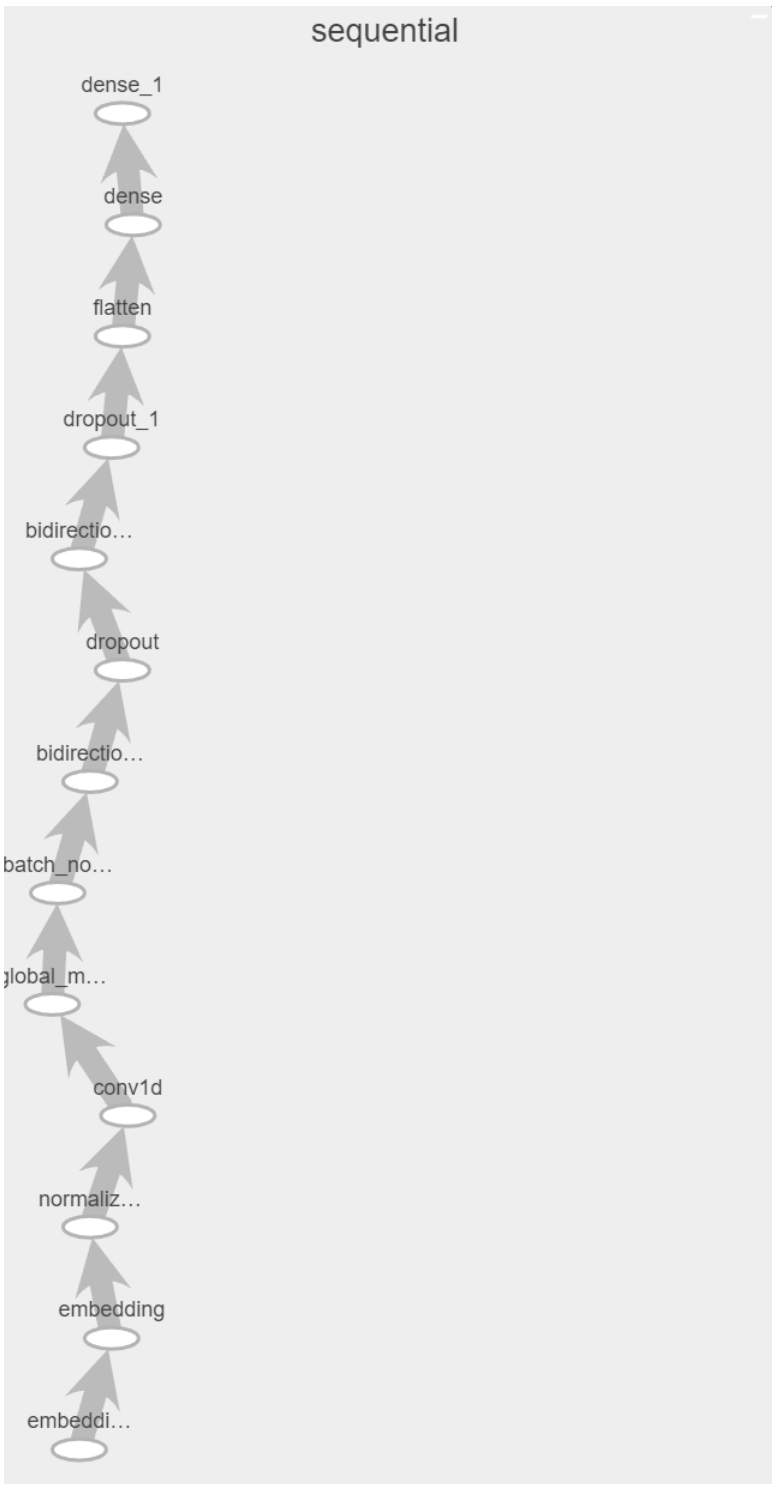
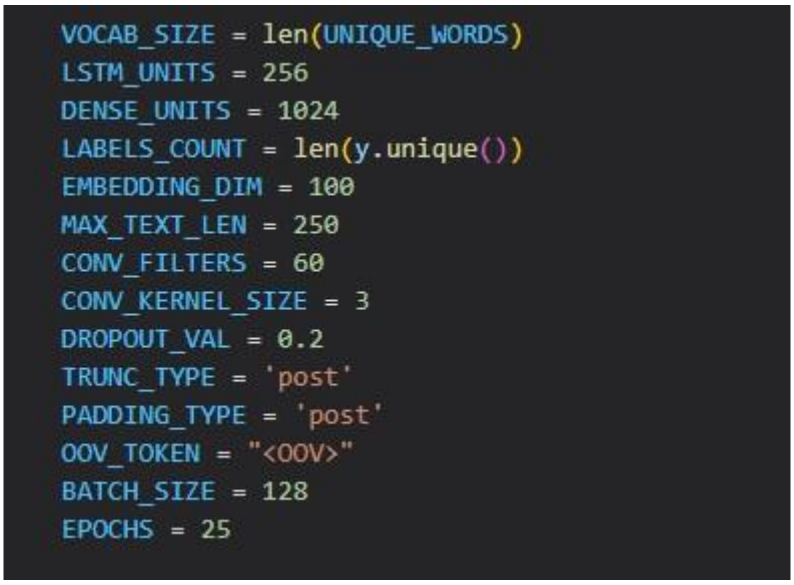


Figure 30 LSTM Variant 7 (Class Weights) Architecture



```
VOCAB_SIZE = len(UNIQUE_WORDS)
LSTM_UNITS = 256
DENSE_UNITS = 1024
LABELS_COUNT = len(y.unique())
EMBEDDING_DIM = 100
MAX_TEXT_LEN = 250
CONV_FILTERS = 60
CONV_KERNEL_SIZE = 3
DROPOUT_VAL = 0.2
TRUNC_TYPE = 'post'
PADDING_TYPE = 'post'
OOV_TOKEN = "<OOV>"
BATCH_SIZE = 128
EPOCHS = 25
```

Figure 31 LSTM Variant 7 Hyperparameters

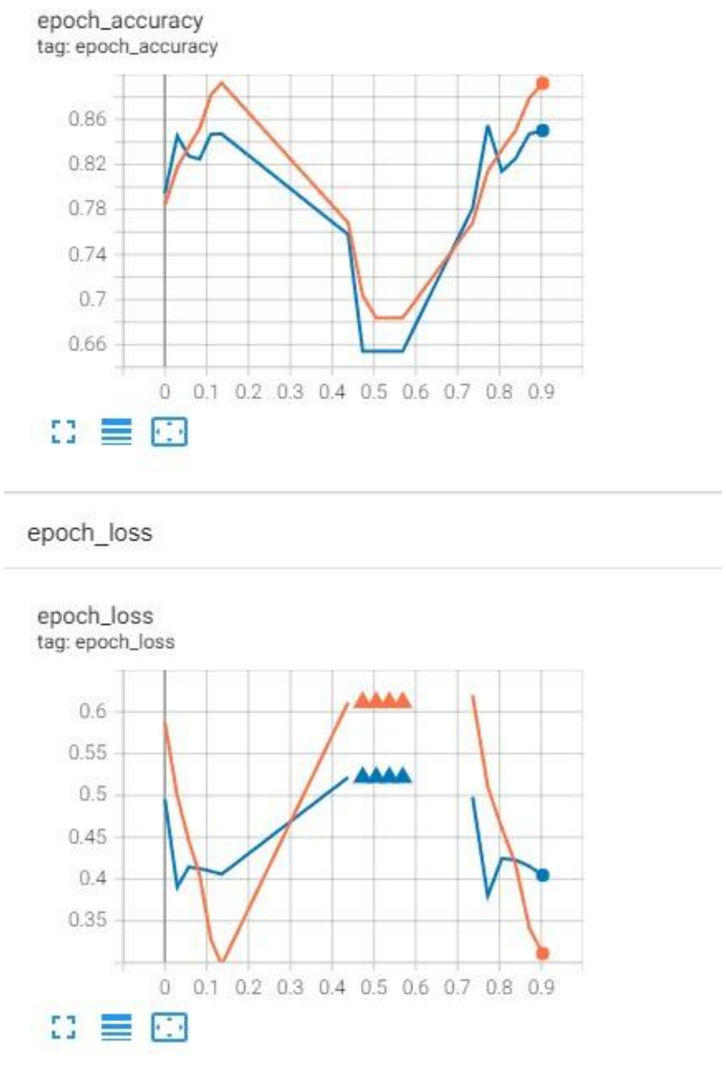


Figure 32 LSTM Variant 7 Learning Curve

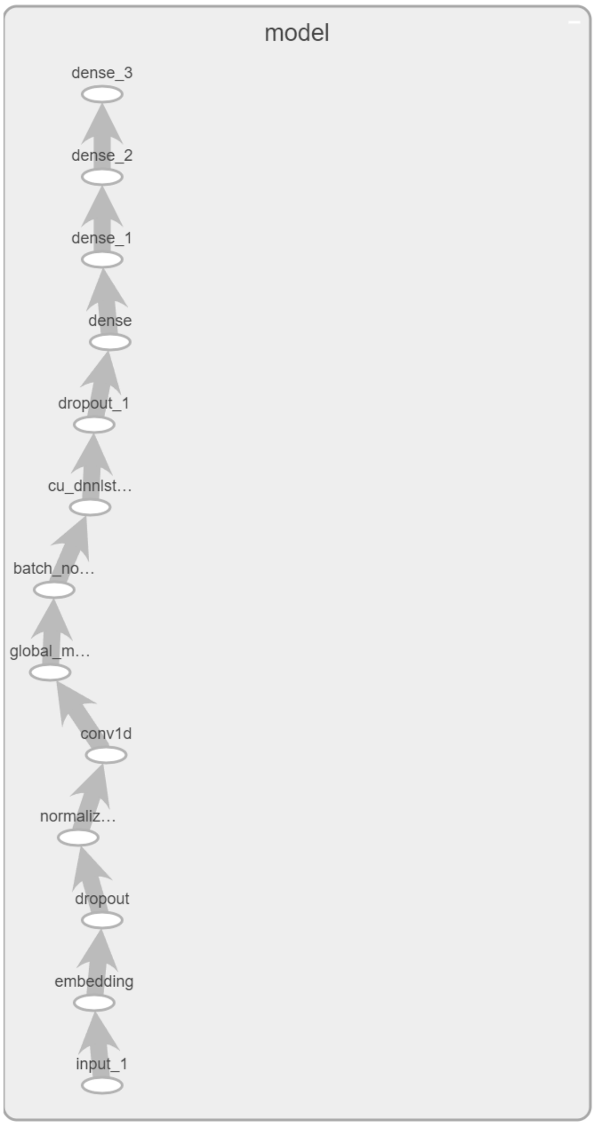
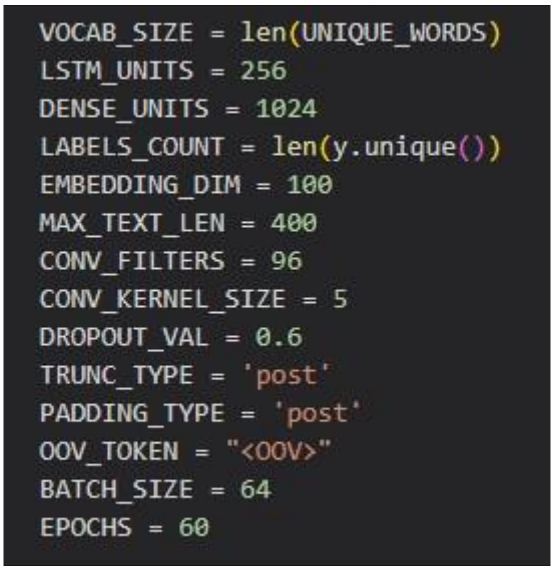


Figure 33 A Tensorboard Visualization of LSTM Variant 27's Architecture



```
VOCAB_SIZE = len(UNIQUE_WORDS)
LSTM_UNITS = 256
DENSE_UNITS = 1024
LABELS_COUNT = len(y.unique())
EMBEDDING_DIM = 100
MAX_TEXT_LEN = 400
CONV_FILTERS = 96
CONV_KERNEL_SIZE = 5
DROPOUT_VAL = 0.6
TRUNC_TYPE = 'post'
PADDING_TYPE = 'post'
OOV_TOKEN = "<OOV>"
BATCH_SIZE = 64
EPOCHS = 60
```

Figure 34 LSTM Variant 27's Hyperparameters

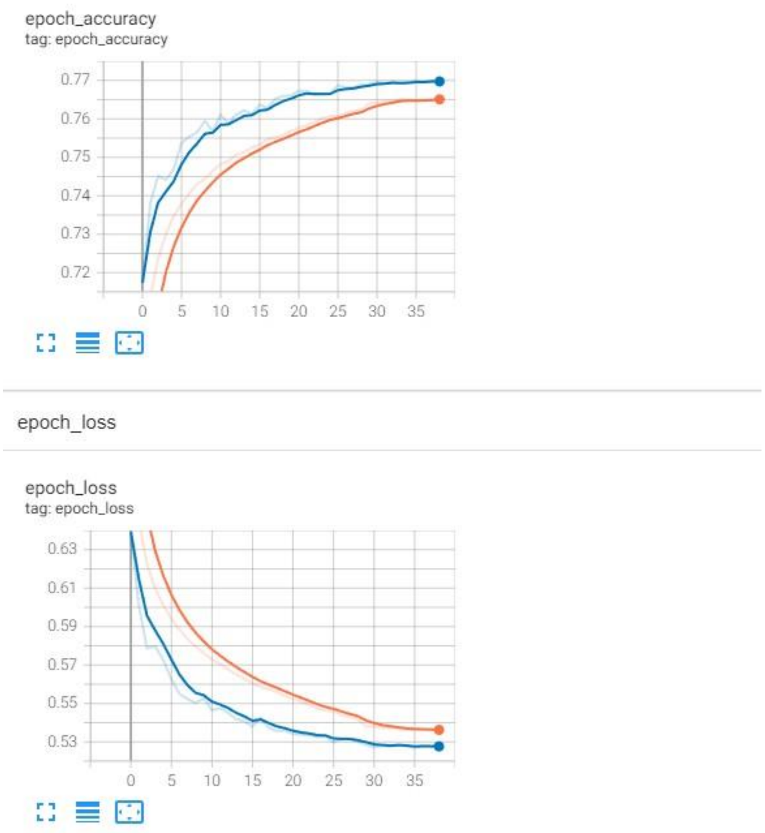
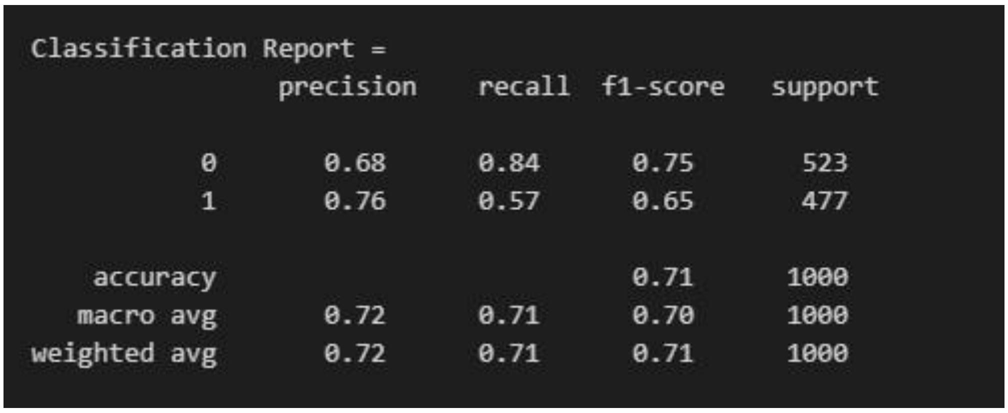


Figure 35 LSTM Variant 27's Learning Curve

To summarize the previous sections, after experimenting with many different architectures, LSTM variants were able to reach the highest levels of performance out of any architecture. Not only that, but when some experimentation is done on the architecture, and CUST is used to balance the data, the most optimal model of the project is achieved, which is variant 27.

5.1.4 Summary of Best Models for Sarcasm Detection



```
Classification Report =
               precision    recall  f1-score   support

           0       0.68      0.84      0.75       523
           1       0.76      0.57      0.65       477

    accuracy                           0.71      1000
   macro avg       0.72      0.71      0.70      1000
weighted avg       0.72      0.71      0.71      1000
```

Figure 36 XGBoost on Subset of Data Classification Report

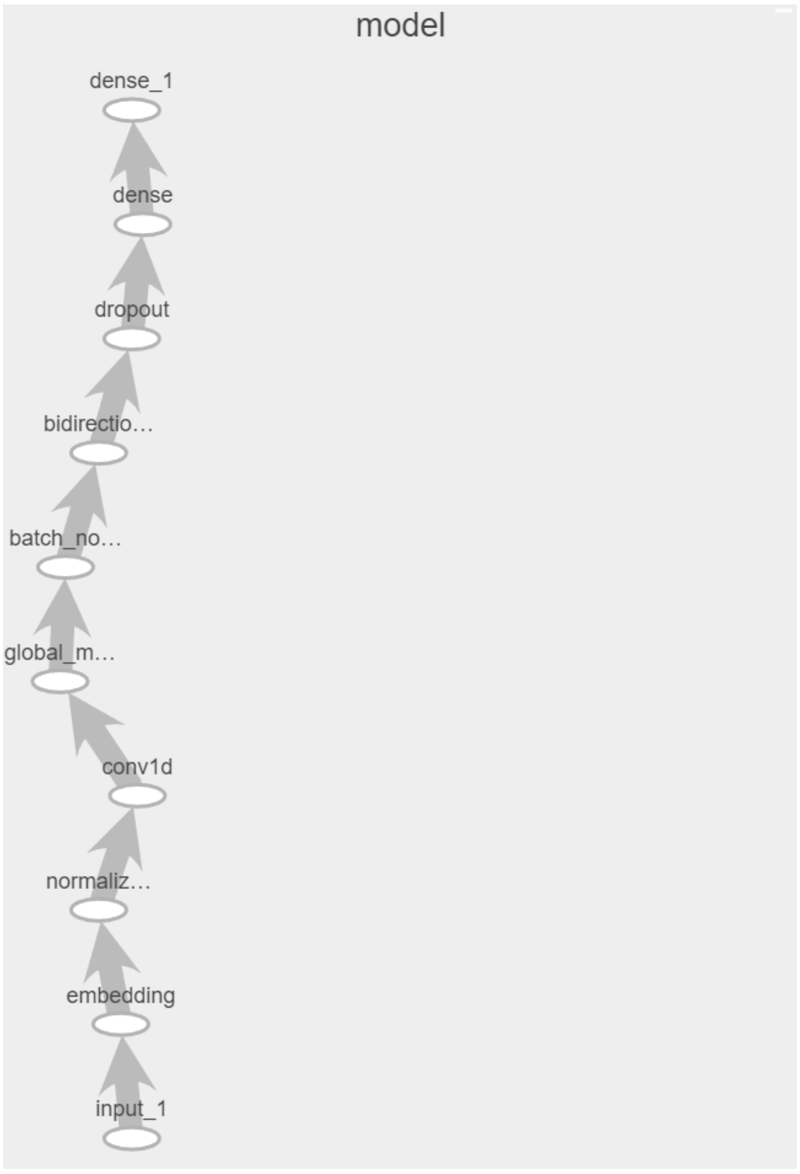
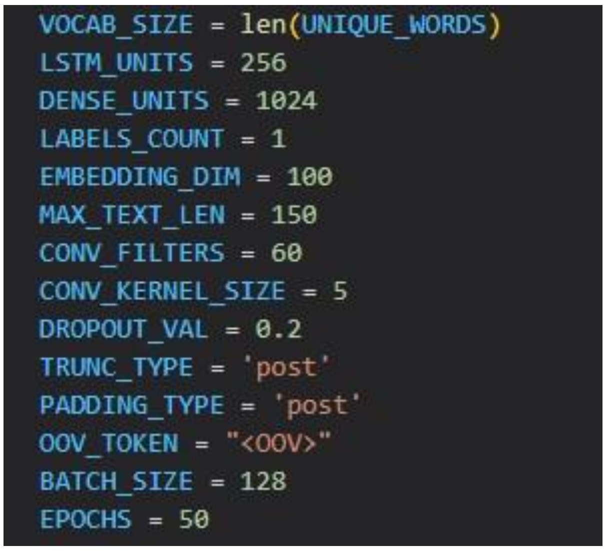


Figure 37 LSTM Variant 1 for Sarcasm Detection Architecture



```
VOCAB_SIZE = len(UNIQUE_WORDS)
LSTM_UNITS = 256
DENSE_UNITS = 1024
LABELS_COUNT = 1
EMBEDDING_DIM = 100
MAX_TEXT_LEN = 150
CONV_FILTERS = 60
CONV_KERNEL_SIZE = 5
DROPOUT_VAL = 0.2
TRUNC_TYPE = 'post'
PADDING_TYPE = 'post'
OOV_TOKEN = "<OOV>"
BATCH_SIZE = 128
EPOCHS = 50
```

Figure 38 LSTM Variant 1 for Sarcasm Detection Hyperparameters

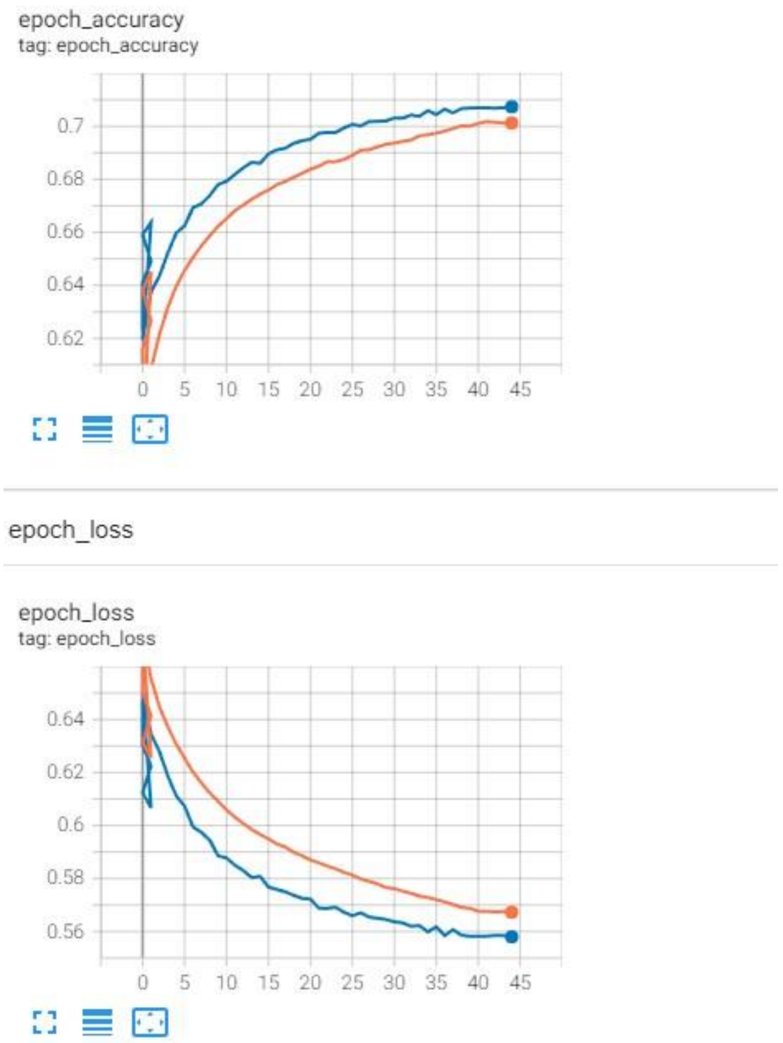


Figure 39 LSTM Variant 1 for Sarcasm Detection Learning Curve

5.1.5 Best Performing Word Embedding

It is important to mention, that all models except LSTM variants 7 and 27 were using Word2Vec instead of GloVe, and still managed to become the best performing models, therefore ending the word embedding comparisons with Word2Vec being the best performing word embedding, with GloVe in second place, TF-IDF in third place, and FastText in last place.

5.2 Evaluation

5.2.1 Comparison with the Related Work

5.2.1.1 Sentiment Analysis

Models	Accuracy Scores
SVM By Haque [12]	0.94
Logistic Regression [14]	0.86
Ensemble [18]	0.78
My Proposed XGBoost Model	0.85
My Proposed LSTM Model	0.77

Table 16 Related Work vs Proposed Model for Sentiment Analysis

5.2.1.2 Sarcasm Detection

Models	F1-Score
Adaboost With Decision Tree [19]	0.67
LSTM [20]	0.79
My Proposed XGBoost Model	0.71
My Proposed LSTM Model	0.71

Table 17 Related Work vs Proposed Model for Sarcasm Detection

5.2.1.3 Explanation of the results

SVM By Haque [12] was able to achieve the highest result out of the Sentiment Analysis models. However, it is important to mention that the datasets that were used in these experiments were no more than 48600 rows of data, with two of the three dataset splits containing less than 5000 rows each. While achieving such a high accuracy is an important feat, it is important to mention that the datasets were not diverse enough to provide challenges for the models to face. This is something that must be taken into consideration as both of the proposed models were on a dataset that is several times larger than the dataset used by this author.

In addition, The Logistic Regression model proposed in [14] was a binary classification problem, where the model was only detecting positive and negative sentiments only. While this is not a big problem to discuss, it is important to mention that these results were not done on the same types of labels such as the other datasets, which all used three labels.

Lastly, for the proposed models in sarcasm detection, while the comparisons with the related work were fair, it is important to note that the sarcasm detection models were not fully optimized, and due to computational, and time constraints, only three LSTM models were trained for this problem. In the future, further optimization is required to bring this model up to a more plausible position.

6 Discussion of Results

6.1 Limitations in Sentiment Analysis and Sarcasm Detection

Although in the previous section, multiple models from both the proposed solutions, and the related work achieved great results, it is still difficult for models to currently predict sentiment, and sarcasm without either predicting some sentiments better than others, or overall detecting all sentiments at a slightly above average level. Furthermore, there is always a trade-off between overall accuracy and individual scores for each sentiment. In sarcasm detection, this issue can be remedied by further optimizing models, however in sentiment analysis, even in other related work, accuracies are often inflated by high predictions in one class, even though another class is not performing as well. This can also be seen in this project, with some of the proposed models before CUST suffering the same fate as the related work. Therefore, even though CUST was a massive improvement for the deep learning models, it might not always be the best case for when the priority lies in allowing the model to get really good at detecting a certain sentiment, rather than detecting the different sentiments that exist.

6.2 Weakness of Sentiment Polarity Scores

Sentiment Polarity scores were highly varied between the two sentiment analysis datasets, and between SMOTE and CUST versions of the dataset. To be clear, Sentiment Polarity as a sentiment analyser was not reliable for detecting sentiment in large amounts of text. From the smaller dataset experiments, it was very clear that sentiment polarity was meant to detect sentiments of shorter reviews, and it was important to not overload it with many reviews. Machine Learning, and Deep Learning models on the other hand, are well equipped to handle long reviews, in large amounts of data. Therefore, they are seen as the superior options when compared to Sentiment Analysers.

6.3 Why Some Models Perform Better Than Others

One noticeable observation about the “Comparison with the Related Work” Section is that while they are models that fall within the range of the proposed models, some models reached way higher scores for reasons mentioned previously. However, it is also vital to include the fact that there were very few papers regarding the exact dataset that was used for the larger dataset portion of the project.

Therefore, it is difficult to compare these results without some biases cropping up. Eventually, it all comes down to the data, and model architectures that were proposed. Also, as discussed previously, several papers use datasets that are either imbalanced, and therefore would have an inflated accuracy due a model's bias towards one or several classes more than the others, or datasets that are very small, that the model does not face any challenges during training and can learn the simple information with no problems. While the proposed model does not suffer from these issues, it does suffer from a different set of problems such as insufficient optimization, and lack of further experimentation with other architectures.

7 Conclusions and Future Work

7.1 Summary

Sentiment Analysis, and Sarcasm Detection are well-known NLP problems that have shown that traditional Sentiment Polarity calculations, and approaches do not produce acceptable results. Considering these previously conducted experiments, machine learning, and deep learning algorithms have been proven to achieve remarkable results, and the proposed models for both problems have reached a high degree of proficiency, and capacity to solve this problem well. Furthermore, after training several models on a dataset consisting of 6990280 rows and experimenting with CUST to improve the results by utilizing 2075683 rows from the data, several high performing models emerged such as XGBoost with about 85% accuracy score, LSTM with 77% for sentiment analysis, and 71% F1-Score for both XGBoost, and LSTM models for sarcasm detection. Further experimentation is required in order to improve the results to reach the state-of-the-art.

7.2 Web Application

A web application solution was developed, in order to add a Graphical User Interface to the models' predictions. The web application, and the utilized models can be found in the project directory under the "web_app" directory.

7.3 Future Work and Improvements

7.3.1 Pre-Trained Architectures

While many architectures were covered in this project, there are several that should be included, but were ultimately not implemented due to computational, and time constraints. These architectures include:

1. LeBERT
2. AlBERT
3. Other Pre-trained Models using similar architectures to the ones proposed in this project such as RNN or LSTM

7.3.2 Optimization

While RNN and LSTM models were greatly optimized for the larger dataset of this project, many other models were not optimized, and their base models were used in the comparisons. As a way to further improve upon the proposed results, it is imperative that more models are optimized, and their hyperparameters are tuned to the max in order to truly discover what their true level of potential can reach.

7.3.3 LSTM Variant 22

LSTM Variant 22 was supposed to be the model that is trained on the whole dataset from start to finish. However, due to shortages in computational power, the model only trained on 4 epochs. In the future, this model should be trained on the whole dataset for several more epochs in order to truly access its potential.

7.3.4 Further Experimentation Required

While the models in the dataset have achieved good results, the literature shows that there are still better results out there. Therefore, more experimentation is required in order to achieve the current state-of-the-art.

7.4 Individual Contributions

7.4.1 CUST

CUST was introduced in this project as an experiment in order to improve the machine learning, and deep learning models' ability to detect the neutral class as the models were underperforming in that regard. While CUST may not be perfect, and may not be applicable to any problem, for this problem specifically, it was effective in improving the models' ability to predict the neutral class, even if there was a trade-off between the neutral class scores, and the overall accuracy scores.

7.4.2 Experimentation with a wide swath of architectures

In this project, many different architectures were experimented with to find the most optimal one. While no model could surpass the state-of-the-art, now there is a wide range of models that were trained to perform sentiment analysis, and sarcasm detection, and the results of these models can be examined for future research to investigate it further.

7.4.3 XGBoost and LSTM Larger Dataset State of the Art

Since there was no research found concerning the use of the proposed larger dataset, for now, it will be claimed that the XGBoost, and LSTM Variant models are the current machine learning, and deep learning state-of-the-art models when trained on this large of a dataset. While several other models have been trained on large dataset sizes, none came close to the proposed larger dataset, even with modified with CUST.

References

[1] P. J. Kiger, "Why the nile river was so important to ancient egypt," History.com, 12-Jul-2021. [Online]. Available: https://www.history.com/news/ancient-egypt-nile-river.

[2] Administrator, "Compstat Research Paper," iResearchNet, 04-Jan-2021. [Online]. Available: https://www.iresearchnet.com/research-paper-examples/technology-research-paper/compstat-research-paper/.

[3] Y. Joris Toonders, "Data is the new oil of the Digital Economy," Wired, 07-Aug-2015. [Online]. Available: https://www.wired.com/insights/2014/07/data-new-oil-digital-economy/.

[4] K. R. Chowdhary, "Natural language processing," Fundamentals of Artificial Intelligence, pp. 603–649, Apr. 2020.

[5] H. Kaur, V. Mangat, and Nidhi, "A survey of sentiment analysis techniques," 2017 International Conference on I-SMAC (IoT in Social, Mobile, Analytics and Cloud) (I-SMAC), Oct. 2017.

[6] K. Mouthami, K. N. Devi, and V. M. Bhaskaran, "Sentiment analysis and classification based on textual reviews," 2013 International Conference on Information Communication and Embedded Systems (ICICES), Apr. 2013.

[7] A. S. Neogi, K. A. Garg, R. K. Mishra, and Y. K. Dwivedi, "Sentiment analysis and classification of Indian Farmers' protest using Twitter data," International Journal of Information Management Data Insights, vol. 1, no. 2, p. 100019, Nov. 2021.

[8] "Removing stop words with NLTK in python," GeeksforGeeks, 22-Aug-2022. [Online]. Available: https://www.geeksforgeeks.org/removing-stop-words-nltk-python/.

[9] T. Ganegedara, "Light on math ML: Intuitive Guide to Understanding Glove embeddings," Medium, https://towardsdatascience.com/light-on-math-ml-intuitive-guide-to-understanding-glove-embeddings-b13b4f19c010#:~:text=The%20advantage%20of%20GloVe%20is,occurrence)%20to%20obtain%20word%20vectors.

[10] D. E. Cahyani and I. Patasik, "Performance comparison of TF-IDF and word2vec models for emotion text classification," Bulletin of Electrical Engineering and Informatics, vol. 10, no. 5, pp. 2780–2788, Oct. 2021.

[11] "5 things you need to know about sentiment analysis and classification," KDnuggets, Mar-2018. [Online]. Available: https://www.kdnuggets.com/2018/03/5-things-sentiment-analysis-classification.html.

[12] T. U. Haque, N. N. Saber, and F. M. Shah, "Sentiment analysis on large scale Amazon product reviews," 2018 IEEE International Conference on Innovative Research and Development (ICIRD), May 2018.

[13] E. Park, J. Kang, D. Choi, and J. Han, “Understanding customers' hotel revisiting behaviour: A sentiment analysis of online feedback reviews,” Current Issues in Tourism, vol. 23, no. 5, pp. 605–611, Nov. 2018.

[14] H. Pouransari, “[PDF] deep learning for sentiment analysis of movie reviews: Semantic scholar,” , 2015. [Online]. Available: https://www.semanticscholar.org/paper/Deep-learning-for-sentiment-analysis-of-movie-Pouransari/8a156da566f15041cff94048d7a24226a19967e2.

[15] S. Liu, “Sentiment Analysis of Yelp Reviews: A Comparison of Techniques and Models,”, arXiv, Apr. 2020.

[16] C. Li and J. Zhang, “[PDF] prediction of yelp review star rating using sentiment analysis: Semantic scholar,” [PDF] Prediction of Yelp Review Star Rating using Sentiment Analysis | Semantic Scholar, 2014. [Online]. Available: https://www.semanticscholar.org/paper/Prediction-of-Yelp-Review-Star-Rating-using-Li-Zhang/205201b657a894eee015be9f1b4269508fca83e8#extracted.

[17] Y. Xu, X. Wu, and Q. Wang, “[PDF] sentiment analysis of yelp ' s ratings based on text: Semantic scholar,” [PDF] Sentiment Analysis of Yelp ' s Ratings Based on Text | Semantic Scholar, 2014. [Online]. Available: https://www.semanticscholar.org/paper/Sentiment-Analysis-of-Yelp-%E2%80%98-s-Ratings-Based-on-Xu-Wu/3335c3d66f514b97fcef0321188635014531d8fa.

[18] H. S. and R. Ramathmika, “Sentiment Analysis of Yelp reviews by machine learning,” *2019 International Conference on Intelligent Computing and Control Systems (ICCS)*, May 2019. doi:10.1109/iccs45141.2019.9065812

[19] J. Lemmens, B. Burtenshaw, E. Lotfi, I. Markov, and W. Daelemans, “SARCASM detection using an ensemble approach,” *Proceedings of the Second Workshop on Figurative Language Processing*, 2020. doi:10.18653/v1/2020.figlang-1.36

[20] S. Khotijah, J. Tirtawangsa, and A. A. Suryani, “Using LSTM for context based approach of sarcasm detection in Twitter,” *Proceedings of the 11th International Conference on Advances in Information Technology*, Jul. 2020. doi:10.1145/3406601.3406624

[21] N. Bahrawi, “Sentiment analysis using random forest algorithm-online social media based,” *Journal of Information Technology and Its Utilization*, vol. 2, no. 2, p. 29, 2019. doi:10.30818/jitu.2.2.2695

[22] Mullen, T. and Collier, N. (2004) Sentiment analysis using support vector machines with diverse information sources, ACL Anthology. Available at: https://aclanthology.org/W04-3253.

[23] J. Kazmaier and J. H. van Vuuren, “The power of ensemble learning in sentiment analysis,” *Expert Systems with Applications*, vol. 187, p. 115819, 2022. doi:10.1016/j.eswa.2021.115819

[24] G. Guo, H. Wang, D. Bell, Y. Bi, and K. Greer, “KNN model-based approach in classification,” *On The Move to Meaningful Internet Systems 2003: CoopIS, DOA, and ODBASE*, pp. 986–996, 2003. doi:10.1007/978-3-540-39964-3_62

[25] Abbas, M. et al. (2019) Multinomial Naive Bayes Classification Model for Sentiment Analysis, 19(3). doi:10.13140/RG.2.2.30021.40169.

[26] N. Silva, E. Hruschka, and E. Hruschka, “BIOCOM USP: Tweet sentiment analysis with Adaptive Boosting Ensemble,” *Proceedings of the 8th International Workshop on Semantic Evaluation (SemEval 2014)*, 2014. doi:10.3115/v1/s14-2017

[27] M. Seyfioğlu and M. Demirezen, “A hierarchical approach for sentiment analysis and categorization of Turkish written Customer Relationship Management Data,” *Proceedings of the 2017 Federated Conference on Computer Science and Information Systems*, 2017. doi:10.15439/2017f204

[28] “Word vectorization using glove,” Medium, https://medium.com/analytics-vidhya/word-vectorization-using-glove-76919685ee0b.

[29] A. Patel and A. K. Tiwari, “Sentiment analysis by using recurrent neural network,” *SSRN Electronic Journal*, 2019. doi:10.2139/ssrn.3349572

[30] Dr. G. S. N. Murthy, Shanmukha Rao Allu, Bhargavi Andhavarapu, and Mounika Bagadi, Mounika Belusonti, “Text based sentiment analysis using LSTM,” *International Journal of Engineering Research and*, vol. V9, no. 05, 2020. doi:10.17577/ijertv9is050290

[31] D. A. Alboaneen, H. Tianfield, and Y. Zhang, “Sentiment analysis via multi-layer perceptron trained by Meta-heuristic optimisation,” *2017 IEEE International Conference on Big Data (Big Data)*, 2017. doi:10.1109/bigdata.2017.8258507

[32] R. Horev, “Bert explained: State of the art language model for NLP,” Medium, https://towardsdatascience.com/bert-explained-state-of-the-art-language-model-for-nlp-f8b21a9b6270.

[33] “Sklearn.metrics.f1_score,” scikit, https://scikit-learn.org/stable/modules/generated/sklearn.metrics.f1_score.html.

[34] K. Leung, “Micro, Macro & weighted averages of F1 score, clearly explained,” Medium, https://towardsdatascience.com/micro-macro-weighted-averages-of-f1-score-clearly-explained-b603420b292f

Appendix I

Smaller Dataset: https://www.kaggle.com/datasets/sripaadsrinivasan/yelp-coffee-reviews

Larger Dataset: https://www.yelp.com/dataset

News Headlines: https://www.kaggle.com/datasets/rmisra/news-headlines-dataset-for-sarcasm-detection?select=Sarcasm_Headlines_Dataset_v2.json

Sarcasm on Reddit: https://www.kaggle.com/datasets/danofer/sarcasm?select=train-balanced-sarcasm.csv

MUStARD: https://github.com/soujanyaporia/MUStARD/blob/master/data/sarcasm_data.json

Note: the MUStARD dataset was forked, and the dataset was pulled out manually. This is important to mention for anyone planning to download this dataset off of the github repo.

Appendix II

String

NumPy

NLTK

Punkt from NLTK

Stopwords from NLTK

Averaged Perceptron Tagger from NLTK

Wordnet from NLTK

Sci-kit Learn

SentimentIntensityAnalyzer from NLTK.Vader

XGBClassifier from Source: https://pypi.org/project/xgboost/

Matplotlib

WordCloud

Pandas

Collections

Seaborn

Requests

OS

Pickle

Itertools

Patoolib

Archive

Imblearn

Gensim

TensorFlow

Keras

Keras Backend

Datetime