Sentiment Polarity Classification of Retail Product Reviews Using Machine Learning and Deep Learning

By Aneesh Kalita

A Dissertation submitted in partial fulfilment of the requirements for the degree
of
MSc Data Analytics
At Dublin Business School

Supervisor: Amit Sharma

Dated: August 2023

Declaration

I certify that unless otherwise specified or when it is acknowledged by references, the applied project I have submitted for the MSc Data Analytics degree is the product of my research. Additionally, this work hasn't been submitted for credit toward any other degree.

Signature: Aneesh Kalita

Dated: 28/8/2023

Acknowledgements

I would like to thank Dublin Business School for giving me the opportunity and resources to conduct this applied research project. I would also like to thank my professors for enlightening me with their knowledge and insight.

I would like to sincerely thank my supervisor Amit Sharma for motivating and guiding me towards the accomplishment of this applied research project. It is my token of gratitude towards my family and friends for offering me support along the journey.

Abstract:

A comparative study has been undertaken to compare the performance of conventional machine learning classifiers and deep learning models to classify the sentiment polarity of the Amazon musical instrument reviews dataset. Text preprocessing and the Tf-idf feature vectorisation techniques were applied to the machine learning models. The 10-fold cross-validation technique and Grid-search hyperparameter tuning were implemented to compare the performance of the Multinomial Naïve Bayes classifier, Logistic regression classifier and the Linear Support Vector classifier. The Google pre-trained Word2vec word embeddings and a custom word embedding trained using the Word2vec algorithm on the Amazon reviews dataset were used to train the artificial neural network (ANN) and recurrent neural network-long short-term neural network (RNN-LSTM) models. The RNN-LSTM trained using the custom-trained word embeddings achieved the best accuracy and F1-score of 92.31% followed by the RNN-LSTM trained using the Google word2vec embeddings with an accuracy and F1-score of 91.92%. The SVM classifier achieved the greatest accuracy of 91.82% among the machine learning classifiers.

Contents

List of Figures

List of Tables

List of Abbreviations

1. SVM-	Support Vector Machine
2. NLP-	Natural Language Processing
3. NLTK-	Natural Language Processing Toolkit
4. ANN-	Artificial Neural Network
5. RNN-LSTM-	Recurrent neural networks- Long short term memory
6. Tf-Idf-	Term Frequency- Inverse document frequency vectorisation
7. POS-	Part of Speech Tagging
8. CBOW-	Continuous Bag of words

Introduction

Natural Language Processing

To share information and knowledge, and to express one's thoughts and emotions, people use natural language to transform their thoughts into comprehensible words, phrases, and sentences. The evolution of language led to an increase in the complexity of its syntax, grammar, and the ways as to which meaning is imparted by the varying arrangement of words. Natural language processing is an interdisciplinary field encompassing linguistics, computer science and statistics, to enable machines to understand, extract meaning, analyse and perform an array of tasks involving natural language. (Khurana *et al.*, 2023)

Sentiment Analysis

The recent surge in digitalisation of e-commerce platforms, communication media, scientific journals, blogs, etcetera has led to a shift in the online presence of users, resulting in an abundance of labelled and unlabelled text available for research and development of novel processing architectures. This shift along with the COVID-19 epidemic led to customers preferring the ease and informed choice of ordering products from e-commerce websites, as they provide transparent and comprehensive details of their products. The customers can write feedback on the purchased products, enabling the companies to conduct an in-depth analysis of the customer preferences and sales distribution of the various products. Manually analysing the sentiment of the vast magnitudes of product reviews is humanly impossible calling for the need for automated machine sentiment classifiers. Achieving a satisfactory and high level of classifier accuracy enables the analysts and company decision-makers to design tailored and improved products, services and plans for varying customer groups resulting in company growth and happier customers.

Sentiment classification is a subfield of natural language processing that aims to analyse and classify the polarity of the sentiment of a unit of text (such as a word, sentence, or an entire document) expressed by a user towards an entity. Depending on the context and research/business problem, the entity may be an event, person, or product/service. (Medhat, Hassan and Korashy, 2014).

The classification of sentiment polarity can be performed at three levels of granularity:

1. Document: The sentiment of an entire document of varying sentence lengths is treated as a unit and classified.
2. Sentence: Different sentences may portray varying sentiments and each sentence's sentiment is classified.
3. Aspect: In a particular sentence, sentiment can be expressed about one or more entities. These entities may have different aspects to them, the different entities and their aspects are targeted, and the sentiments expressed about each aspect are classified. (Mabrouk, Redondo and Kayed, 2020)
   In the present research, each review of customers purchasing musical instruments from Amazon shall be treated as a document. Its sentiment shall

be classified at the document level, as aspect and sentence-level classification require more computational power.

The Sentiment of text can be classified into different polarities depending on the label polarity distribution of training corpus in the case of supervised sentiment classification- Bipolar sentiments (positive or negative), Ternary (positive, neutral, and negative) or multiple-level polarity (such as review ratings ranging from 1 to 5, 1 implying very bad to 5 implying very good).

Based on the literature, there have been different approaches to sentiment classification based on the technique and algorithm used- Lexicon-based approaches, Machine Learning or Hybrid-based approaches. The lexicon-based sentiment classification approach involves utilizing a predetermined sentiment lexicon to analyse text sentiment by matching words against labelled polarities, aggregating scores, and potentially applying thresholds for sentiment determination.

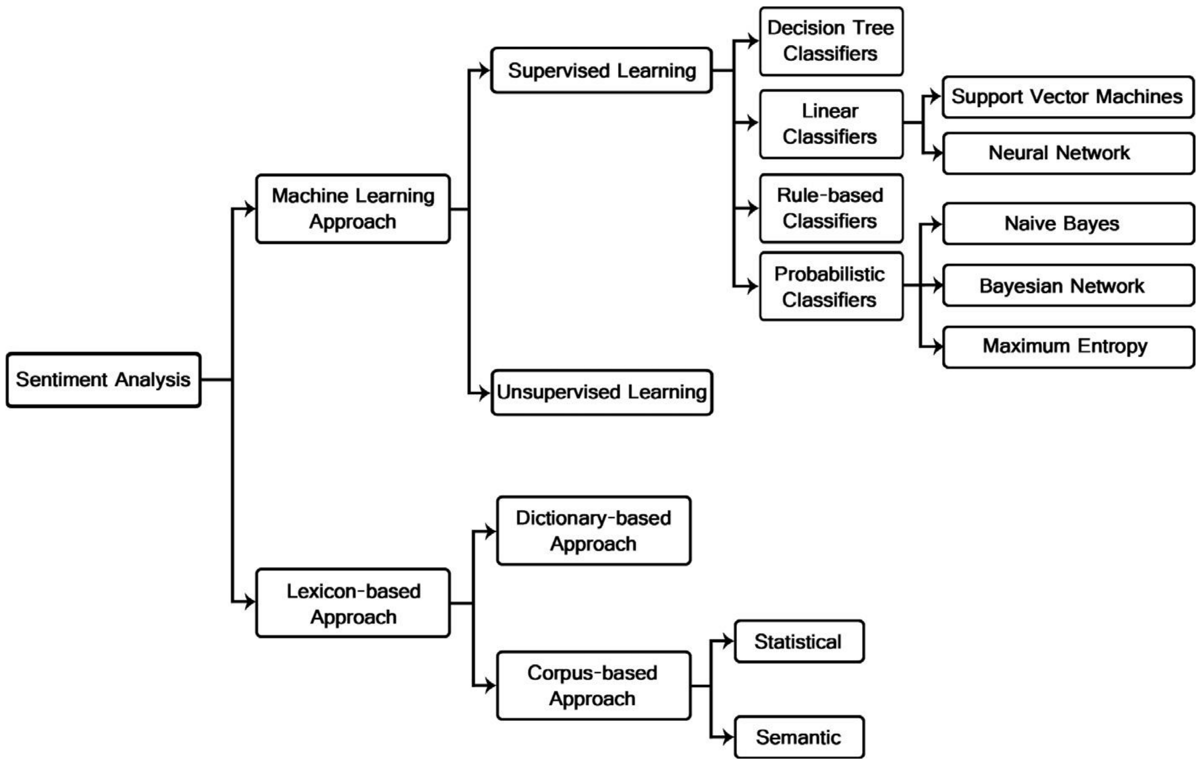


Figure 1 Sentiment classification approaches (Medhat, Hassan and Korashy, 2014)

Machine Learning approaches comprise cleaning the text data of noise and redundancy, extracting the important features and feature selection, and developing a hypothesis function capturing the patterns in the train data mapping the input to the output to generalize to new data. Machine Learning approaches particularly Neural Networks have achieved state-of-the-art performances. Other popular algorithms include Naïve Bayes classifier, decision trees, support vector machines, linear models, random forest classifier, etcetera. (Mabrouk, Redondo and Kayed, 2020)

Research Problem

The product reviews posted by customers on e-commerce websites contain both subjective and objective opinions, and the automated and accurate classification of the review sentiments adds to the knowledge base of the company. Amazon is a favoured and global e-commerce company having a sizeable customer base and records reviews in numerous languages and of varying product domains. The highly unstructured and noisy nature of text data poses difficulty in analysing the data and developing accurate predictive models to classify review sentiment. According to the literature, sentiment classification is highly dependent upon the domain of study, the text preprocessing methods employed, the feature selection, extraction, and transformation methods used, and the predictive modelling approach taken – lexical-based, machine learning-based, or a hybrid approach. Due to the diverse range of techniques, and supervised statistical, machine learning and deep learning-based predictive modelling techniques it is imperative to conduct thorough research to compare the sentiment classification performances of each of the chosen technologies.

Research Aim

In this research, a comparative study will be undertaken to compare the performance of supervised machine learning classifiers - Logistic Regression, Multinomial Naïve Bayes and linear Support Vector Machines, and supervised deep learning classifiers- Artificial Neural Network (ANN), and RNN Long Short-Term Memory model (RNN-LSTM) to classify the sentiment polarity of the Amazon musical instruments reviews dataset and evaluate in terms classification accuracy, precision, recall and F1-score. The deep learning models shall be trained on the pre-trained Google word2vec word embeddings and a custom word embedding trained on the Amazon dataset using the Word2vec algorithm.

<u>**Literature Review**</u>

In the digital era information is transacted mostly via the internet, which is inherent in the form of text in blogs, scientific domains, social media sites, and e-commerce platforms. While numbers and symbols can be easily digested by machines, the processing of natural language for analysis poses a major problem as it is unstructured and captures the contexts in a changing manner. (Adikari *et al.*, 2021 in Singh *et al.*, 2022)

The various problems encountered during sentiment polarity classification and their resolution are discussed by Erik Cambria and Soujanya Poria et al. (Cambria *et al.*, 2017). The subproblems can be divided into three categories- syntactic, semantic, and pragmatic. The syntactic layer contributes to refining and standardizing textual content, addressing issues like spelling, abbreviations, and sentence structure. It employs techniques like regular expressions, reducing different forms of words to the root word (stemming and lemmatization), sentence tokenization and tagging words to their parts of speech. The semantic layer deals with extracting meaning from the text and employs methods like Topic extraction (breaks down text into constituent topics), Named Entity Recognition to identify and categorise named entities, such as names, dates, locations, and organizations, in text and Word sense disambiguation.

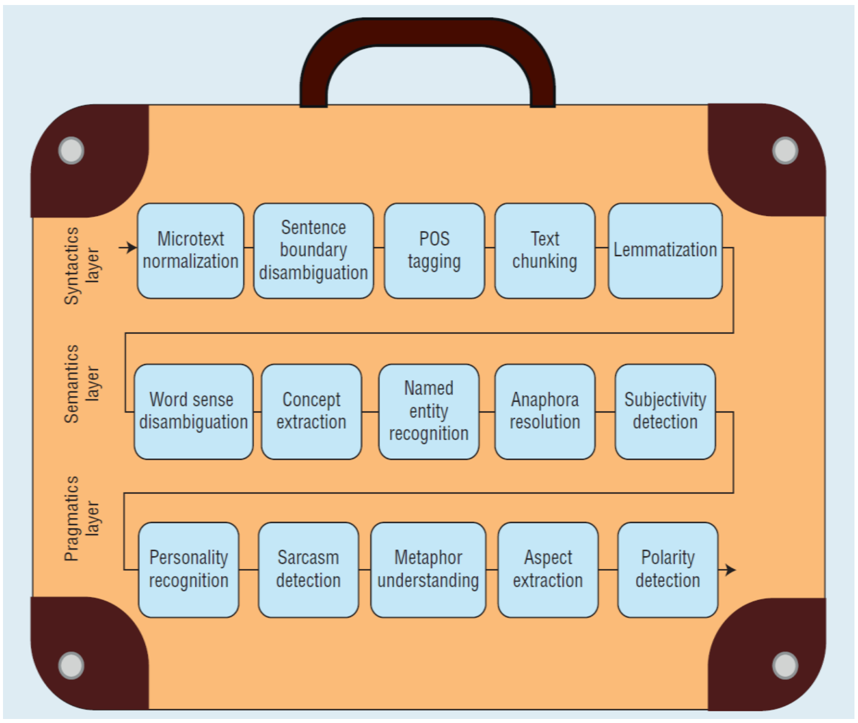


Figure 2 . Subproblems involving sentiment classification (Cambria et al., 2017)

(Jianqiang and Xiaolin, 2017) This paper discusses the effects of six text preprocessing methods on the sentiment classification performance of Logistic Regression, Naïve Bayes, Support Vector Machine and Random Forrest Classifier on the Twitter dataset. The importance of removing noise and redundancy is highlighted. The removal of URLs from the input text, and the removal of numbers improved classifier performance, fixing contractions of words expressing negation such as won't to will not and can't to cannot, and expanding abbreviations affected classifier performance. Removing stop words (common words such as - and, or, this, that, etcetera) that do not contribute to the significant variance of input texts was performed. The models were evaluated by 10-fold cross-validation and the hyperparameters of the models were tuned by using Grid Search Cross-validation of the scikit-learn library.

(Sunitha, Joseph and Akhil, 2019) conducted a study on the performance of supervised machine learning classification algorithms for sentiment classification on two different datasets – the Twitter review dataset and the US airline dataset. An advanced preprocessing stage was proposed consisting of word case standardisation, tokenisation of text, stop word filtration, filtration of tokens based on a threshold length and finally stemming of tokens into their root word. The text was vectorised using the term frequency-inverse document frequency vectorisation (Tf-Idf) and finally, the feature space was pruned using a threshold tf-idf score. This process resulted in an improvement of classification accuracy and precision by 8% and 6% with the support vector machines (SVM) classifier achieving the top accuracy of 83% followed by Naïve Bayes (80.17%) and K-nearest neighbours (81%).

(Dey *et al.*, 2020) The authors have compared the performance of two classifiers, linear support vector machines and Naïve Bayes, for the sentiment classification of Amazon product reviews. The reviews consisted of multi-class labels which were converted to a binary classification problem by ignoring neutral rating labels. The preprocessing stage consisted of tokenisation of text into word tokens, stop word removal and missing value imputation, followed by filtering important nouns based on their occurrence frequency and context. The vectors were input to the classifiers giving a precision, recall and F1-score of 83.990%, 83.997% and 83.993% for SVM , and 82.853%, 82.884% and 82.662% for Naïve Bayes.

(Arshad *et al.*, 2021) The authors have conducted a comparative study on the sentiment classification performance of linear SVM using stochastic gradient descent optimization, Naïve Bayes and logistic regression on a multi-class labelled stack-overflow question dataset. The features have been extracted from the pre-processed using tf-idf vectorisation as well as word2vec word embeddings and Document2vec embeddings. The logistic regression with the Doc2vec classifier gave the best performance with an accuracy of 80% followed by SVM(78%).

(Jiang, Hu and Jiang, 2022) The authors have proposed a hybrid model consisting of text vectorisation using the Word2vec algorithm and combined it with an LSTM model for binary text sentiment classification of the IMDB movie dataset. They have compared the performance of the Word2vec-LSTM classifier with conventional machine learning classifiers – KNN, Logistic regression, SVM and Random forest using word index tokenisation and hash table vectorisation. The Word2vec-LSTM outperformed the machine learning classifiers with an accuracy of 81.56 and an F1-score of 81.49%. The trained word2vec text vectors of 300 dimensionality were fed to an embedding layer and a single layer of LSTM was used with 128 memory units.

(Manalu, Tulus and Efendi, 2020) This research has compared the performance of the RNN-LSTM model with 100,200 and 300-dimensional word2vec word embeddings as input on a Google Playstore reviews dataset and with an embedding layer as the input. The multi-class reviews have been merged for binary sentiment classification. A dropout layer has been introduced to improve the generalization capacity of the model and an LSTM layer with 100 memory units has been used to train the classifier. Early stopping has been implemented to prevent the LSTM from overfitting the data. The LSTM model with the word2vec embedding outperformed the LSTM with a normal embedding layer with an accuracy of 86.7% and an F1-score of 85%.

TECHNOLOGIES AND LIBRARIES USED

The Pandas library (1.5.3) is used to analyse the dataset. It provides powerful data structures such as the Dataframe and series objects and functionalities such as slicing, indexing and filtered extraction of data. Seaborn and Matplotlib libraries are used for plotting the visualisations.

NumPy 1.24.4: It is a powerful numerical computing library providing array and matrix data structures allowing for performing fast calculations.

Natural Language Toolkit 3.8.1 (NLTK): It is a library used for natural language processing tasks providing modules such as word and sentence tokenisation, part of speech tagging, word lemmatisation and stop-word removal.

Scikit-learn 1.2.2: It offers tools for preprocessing data, feature selection and model evaluation. It allows for the implementation of different machine-learning techniques.

TensorFlow 2.7.0: It provides comprehensive functionalities for designing, training and developing neural network architectures and hyperparameter tuners.

Keras 2.7.0: With TensorFlow as a backend, the Keras API allows for the development and training of neural networks in a simpler method.

Gensim 4.3.1: It provides natural language processing modules and algorithms such as the Word2vec, Glove and FastText word embeddings.

Research Methodology

In this research, the CRISP-DM (Cross Industry Standard Practise for Data Mining) (Shearer, 2000) methodology has been implemented to methodically follow an effective process flow for the sentiment classification of the Amazon musical instruments review dataset. It is a standard domain-neutral comprehensive process for effectively tackling a data mining problem with six iterative steps.

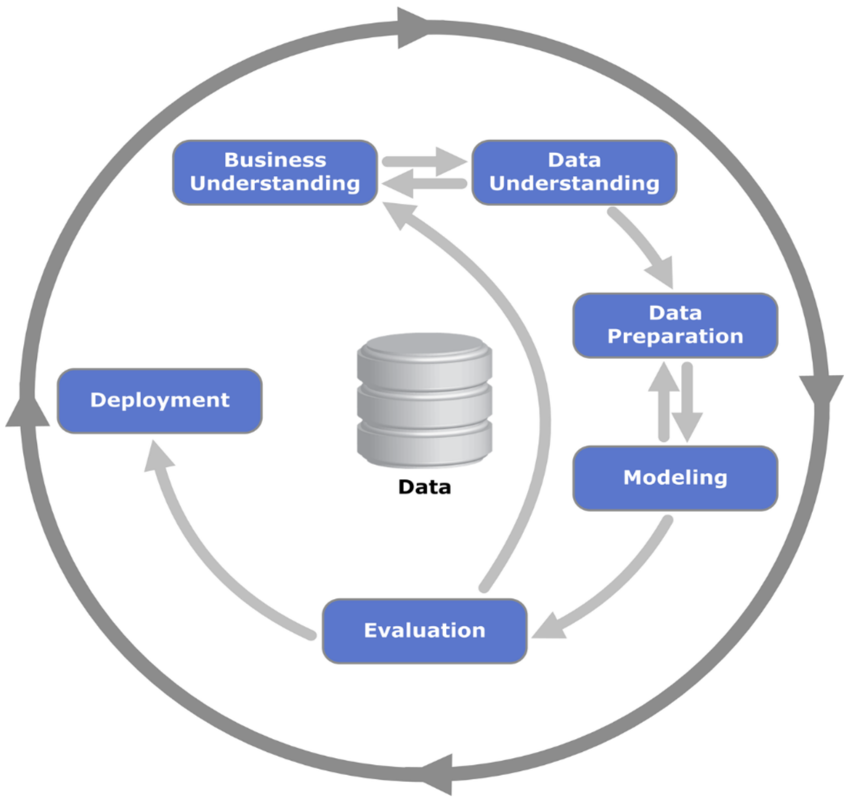


Figure 3 CRISP-DM methodology(Shearer, 2000)

The research has been broadly divided into the following subtasks:

- To acquire the Amazon musical instruments reviews dataset, analyse the dataset and the distribution of its independent features and target labels.
- To methodically clean the text dataset by employing preprocessing techniques such as character case standardisation, removal of unwanted characters, numbers, and noise, stop-word removal, part-of-speech tagging, and word lemmatization.
- To extract numeric feature vectors from the text using an n-gram count vectorizer and transform the vectors with a term frequency-inverse document frequency (tf-idf) vectorizer to weigh the n-grams according to their document discriminative power.
- To create pipelines streamlining the process flow of feature extraction and predictive model fitting. Input the pre-processed text data into the pipelines, fit the predictive models and evaluate the dataset using 10-fold cross-validation to obtain an unbiased and comprehensive evaluation of the particular machine learning model on the dataset in terms of classification accuracy, precision, recall and F1-score.

- To fine-tune the predictive models by tuning their hyperparameters using Grid Search 10-fold cross-validation and test the model by employing the fine-tuned hyperparameters for a fair assessment.
- To import the word2vec embedding word vectors pre-trained on the Google news set, extract word features by creating an embedding vector matrix. To obtain custom word embeddings by training the Word2vec algorithm on the Amazon dataset. Create the ANN, RNN and RNN-LSTM model architectures and evaluate the sentiment classification performance of the model on the pre-processed and tokenized data.
- Fine-tune the hyperparameters of the deep learning models to improve classification performance.
- Perform a comparative study of the performance of the models in terms of evaluation metrics- accuracy, precision, recall and F1-score of the predictions.

1. Business Understanding

The Amazon e-commerce website sells a wide array of products in various localities and systematically records the text feedback reviews of the customers for an in-depth analysis. These product reviews are invaluable to the company as they contain the opinions and sentiments of the customers, highlighting the strengths and weaknesses of the products as well as the satisfaction levels of the customers. As manually labelling the sentiments is an insurmountable task due to the huge magnitude of reviews generated daily, automated sentiment prediction systems must be designed that can make accurate classifications. By analysing the predicted sentiments, the inherent flaws in the products can be improved and the favoured products can be recommended to potential customers. An in-depth analysis of the varying customer bases according to product choices can be conducted to develop better marketing and distribution plans.

2. Data Understanding

The Amazon musical instruments reviews dataset is a tab-separated values file containing 15 features of which star_rating is the target label (dependant feature) containing the sentiment of each review and the rest are independent features.

The dataset is obtained from the link provided:

s3.amazonaws.com/amazon-reviews-pds/tsv/index.txt

The review_body feature contains the product review text. The dataset has a total of 904765 samples. Below is a summary of the features:

Marketplace: The sales location of the product being reviewed (Nominal)

customer_id: Unique identifier of Customer (Nominal)

review_id: Unique identifier of Review (Nominal)

product_id: Unique identifier of the product (Nominal) product_parent: Parent product category identifier of the product (Nominal)

product_title: Name of the product (Nominal)

product_category: The musical instrument category (Nominal)

star_rating: The product rating provided by the customer (sentiment score) (Ordinal)

helpful_votes: helpfulness score of the review by users (Discrete)

total_votes: Total votes provided for the review by users (Discrete)

vine: Whether the reviewer was a verified reviewer. (categorical)

verified_purchase: Whether the purchase made was verified. (categorical)

review_headline: A short review headline for the product (string)

review_body: The main review text (string)

review_date: Date of review (Date)

Some of the customer reviews with their sentiment ratings are outlined below:

Review	**Sentiment Rating**
Really bad. Bought as a midi trigger kit but the latency from the module is ridiculous. Comlete waste of $$. A toy.	1
Bridge pickup was broken. I replace d the pickup and ok now. To cheap to send back.	2
Works very good, but induces ALOT of noise.	3
nice Microphone. good delivery	4
This is an awesome mic!	5

A function was developed to count the number of reviews per year. The reviews range from the year 1999 to 2015 with the customers posting the least number of reviews in 1999 and the maximum number of reviews from 2014. This indicates a steady rise in customer growth and involvement over the years.

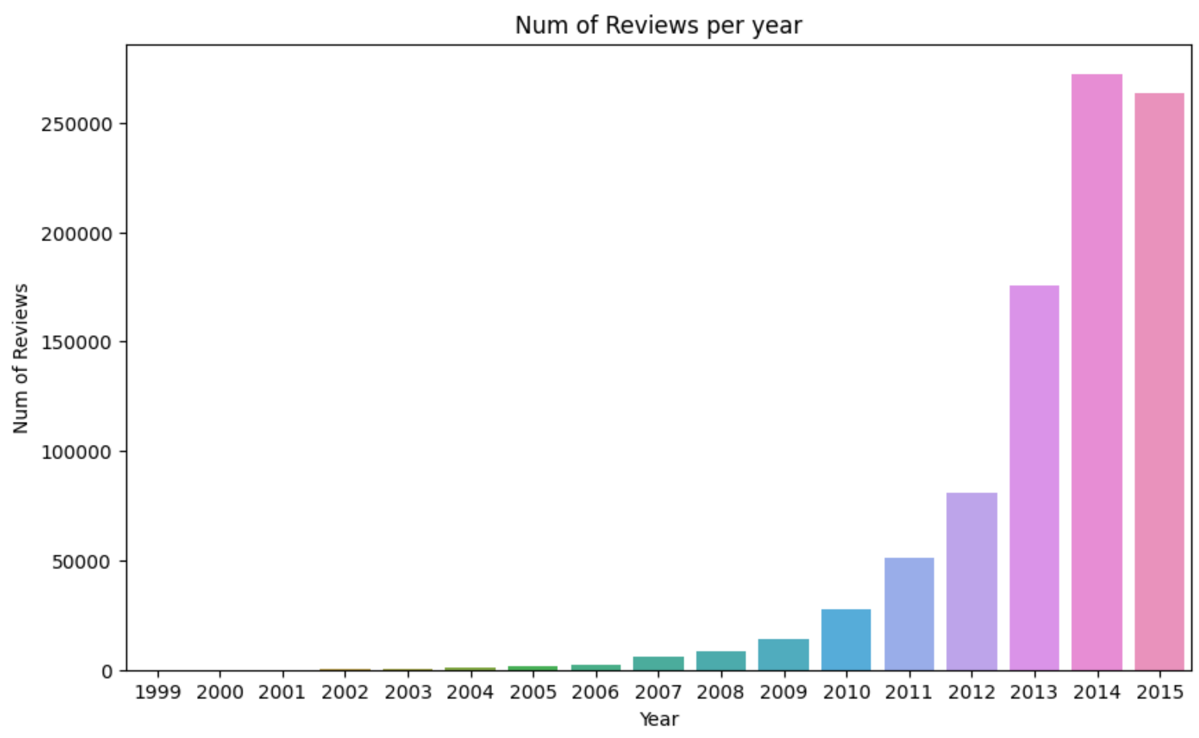


Figure 4 Number of Reviews per Year in the Amazon dataset

The Star ratings are the ratings given by the customers per product purchase and can be considered a measure of the sentiment of the customer's review of the product. From the distribution of star ratings, it is noted that there are 5 different ratings:

5: Very Positive	3: Neutral	1: Very Negative
4: Positive	2: Negative	

The distribution of the sentiment ratings is outlined below:

Star_Rating 1: 65081

Star_Rating 2: 39779

Star_Rating 3: 65509

Star_Rating 4: 151681

Star_Rating 5: 529739

A function is defined to note the different review star ratings and calculate the Percentage of star_ratings distributed in the dataset. A bar plot is plotted to visualize the Percentage of star_ratings distribution

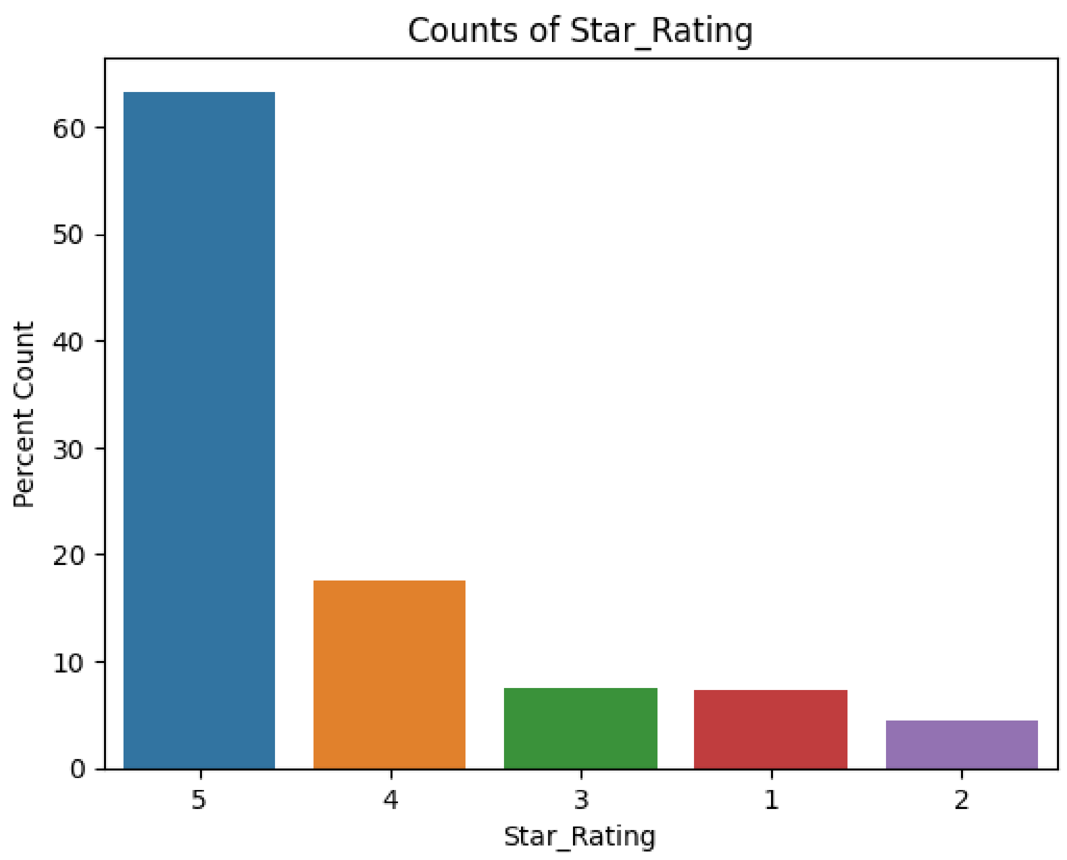


Figure 5 Distribution of target class feature star_rating

We can see from the bar plot below that the dataset is highly imbalanced in terms of Star_rating with a significant percentage of the dataset being 5-star rated reviews (62.19 %) and the minority class being negative and very negative reviews with 2 star_rating having the least reviews (4.67%). This indicates that the majority of the customers are satisfied with the quality of the products whereas a minority are not happy. Both sentiments are equally important for classification and prediction as negative sentiment reviews can be used to monitor and improve upon the products.

As the research is conducted on sentiment classification of text data using supervised predictive models, our main features of interest shall be the review_body containing the text reviews and the star_ratings labels containing the sentiment rating of the reviews.

3. Data Preprocessing

Missing values: Upon checking for missing values in the dataset, we found there are 0 missing values in the dataset. The data types of the various features were cast into dtypes consuming smaller memory such as integer-64 to int-32 and int-8 for memory efficiency and reducing the overload of predictive modelling.

Duplicate Review removal: A total of 52976 duplicate reviews were found and were dropped as these add up to redundancy.

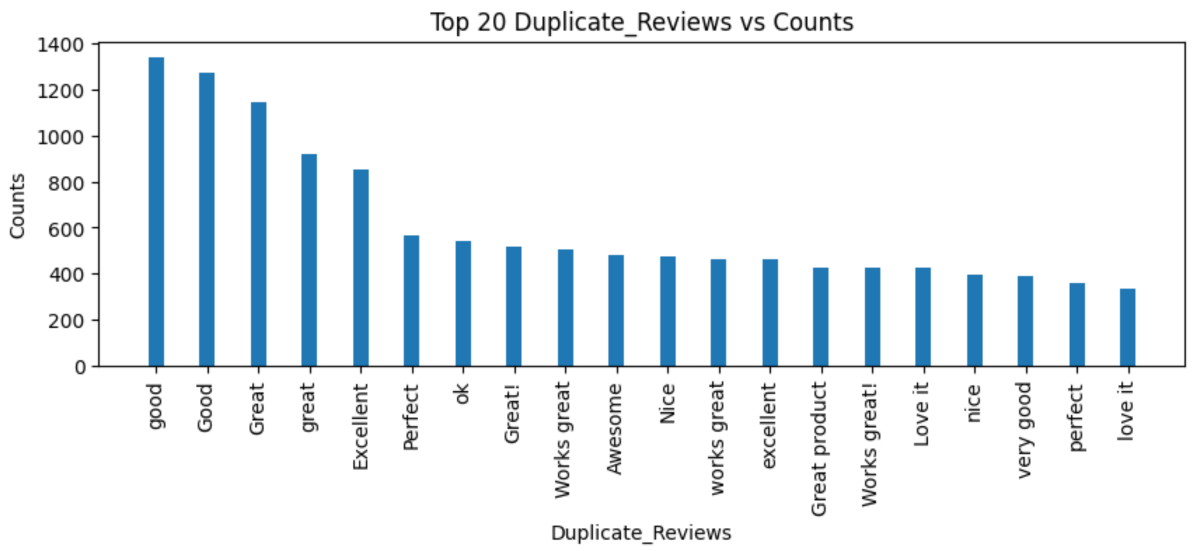


Figure 6 Top 20 Duplicate reviews in the dataset

Dropping of unwanted features:

The features review_id (identifier), review_body containing the text reviews and the target star_rating containing sentiment scores are the features of main interest. The rest of the features such as marketplace, customer_id, product_category, etcetera are dropped.

Merging of Target class labels for binary sentiment polarity classification:

According to the literature, it has been decided to research the sentiment classification of binary polar sentiments. The reviews having target star_rating 3 convey neutral sentiment and have been dropped. The reviews having sentiment scores 1 and 2 are merged to have a sentiment score of 0 indicative negative sentiment while the reviews having sentiment scores 3 and 4 are merged to yield reviews having positive sentiment of 1. The resulting data frames are concatenated and shuffled and the resulting review counts are shown below.

Star_rating 1: 681420 (positive sentiment)

Star_rating 0: 104860 (negative sentiment)

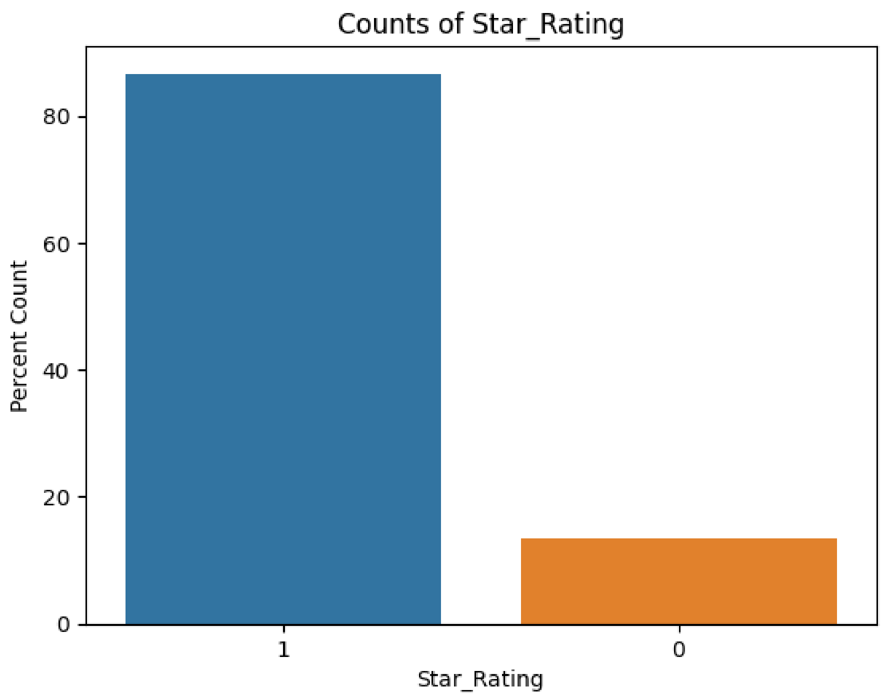


Figure 7 Merged Binary sentiment feature

<u>Balancing of target class distribution using Random Undersampler:</u>

Many machine learning models assume the target class to have a balanced distribution and giving imbalanced input data results in the model giving more weight to the majority class during training the model. As a result, the model can classify the majority class accurately, but the minority class gets classified inaccurately. (Brownlee, 2020) In this context, the dataset will be balanced using scikit-learn random undersampler to randomly pick up points from the majority class and undersampling

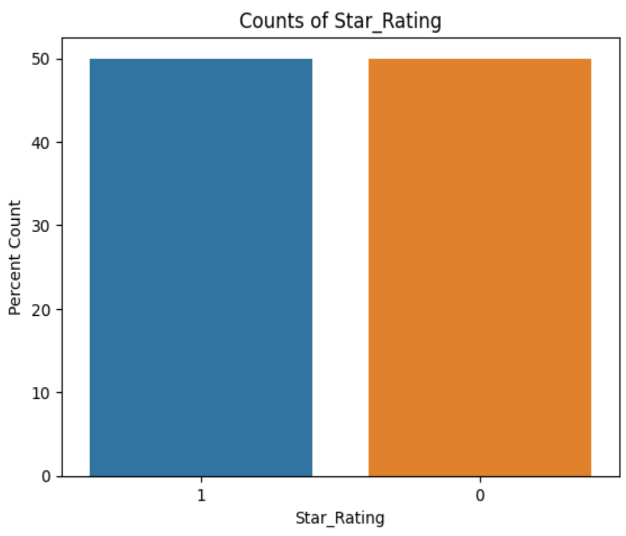


Figure 8 Balanced Target Sentiment Feature

Some of the top frequent words appearing in the balanced dataset reviews are generated using Wordcloud before cleaning text. The presence of HTML tags such as <br> is noted.

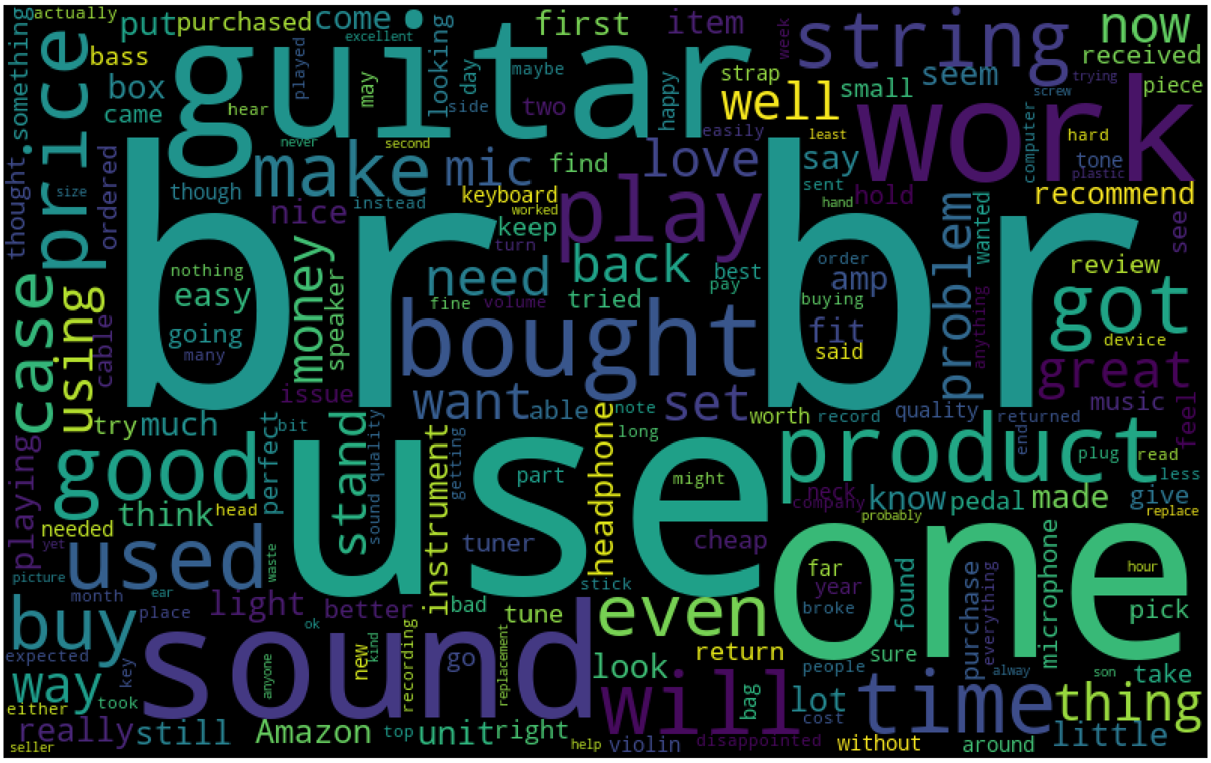


Figure 9 Top frequent words in the reviews before text cleaning

<u>Cleaning of review texts:</u>

For the predictive models to accurately classify the input reviews, they need information-rich word features that can help discriminate the reviews as positive or negative. Text containing unwanted symbols, numbers and words leads to the predictive model capturing unwanted noise which results in a deterioration of classification performance. In general, customers write reviews in informal language and use abbreviated words, slang, symbols, numbers and emoticons which does not add to classifier performance.

To clean the text a class object was developed which accepts the dataframe containing unprocessed text reviews and applies a cleaning function to clean each review and return the processed dataframe. An array of text-cleaning techniques has been used in a streamlined flow which is outlined below:

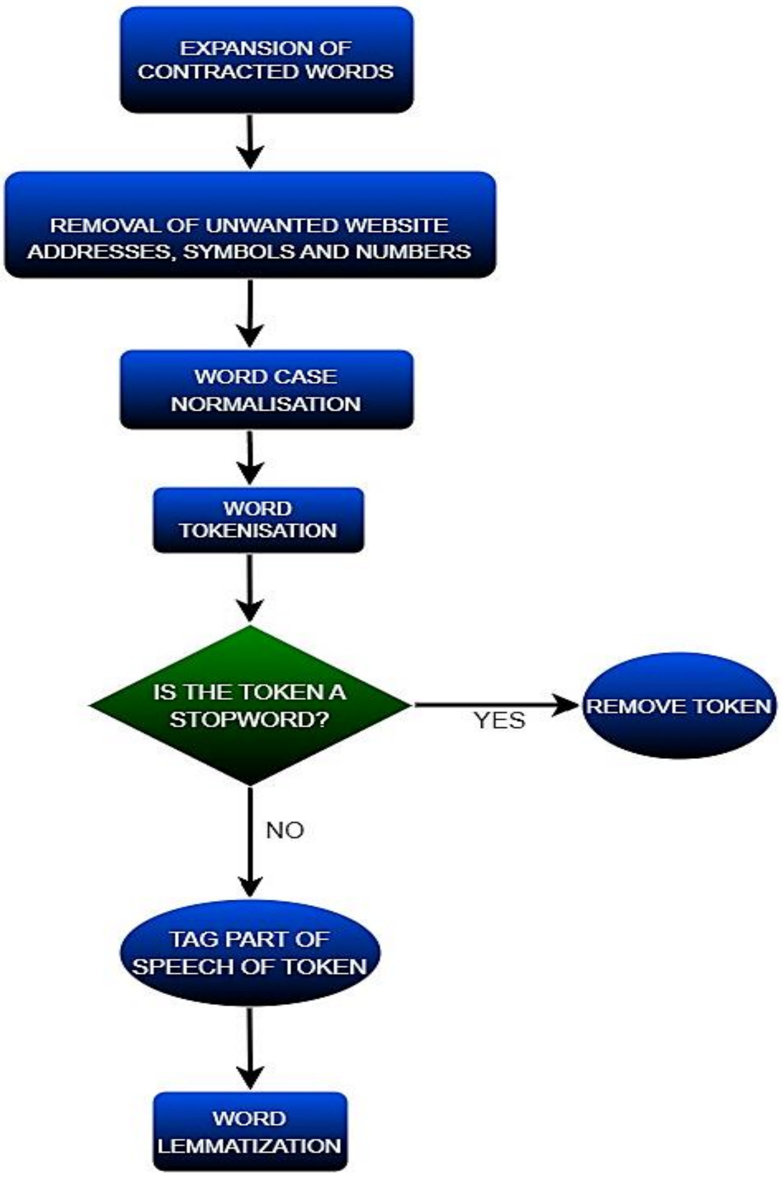

EXPANSION OF CONTRACTED WORDS
REMOVAL OF UNWANTED WEBSITE ADDRESSES, SYMBOLS AND NUMBERS
WORD CASE NORMALISATION
WORD TOKENISATION
IS THE TOKEN A STOPWORD?
YES
REMOVE TOKEN
NO
TAG PART OF SPEECH OF TOKEN
WORD LEMMATIZATION

Figure 10 Text Preprocessing Flowchart

- Fixing of contracted words: The contractions package has been used to expand contracted forms of words in the review text such as shouldn't, mustn't, I'll to should not, must not, and I will. This leads to uniformity in the corpus of words and aids in the downstream tasks such as tokenisation and lemmatisation in recognising the words.
- Removal of website addresses, symbols and numbers: Using Python regular expressions, we searched for website addresses such as 'https?://amazon.com and HTML tags such as <br>, and removed them. Numbers, punctuations and other symbols do not play a role in sentiment classification and were removed.
- Word case normalization: The words in the review texts were cast to lowercase obtaining a consistent case format, to ensure that variations in the case do not affect text analysis. This leads to a reduced dimensionality of the vocabulary and improves text consistency as the same word written in lowercase or uppercase is treated as one word. Normalizing the case of words also helps with tasks like tokenization, where words are split into individual tokens. Consistent case ensures that words are correctly identified as separate tokens, even when they appear with different cases.
- Word Tokenisation: For the transformation of text into features, we need to transform the documents into individual words. The nltk punkt tokenizer is used which recognizes sentence boundaries and tokenizes the document into individual words.

- Stopword removal: Stopwords are commonly occurring words in the English language such as 'a', 'an', 'the', 'to', etcetera which add little semantic meaning to the document. Removal of stopwords helps reduce the dimensionality of the vocabulary, reduces noise and allows the model to capture the information-rich content of words and topics. The words 'no', 'not' and 'nor' are removed from the NLTK stopword list as they denote negation.
- Part of Speech tagging (POS) and Word Lemmatization: With POS tagging, each word in a text is given a unique grammatical label that designates its syntactic category or part of speech, such as a noun, verb, adjective, or adverb, among others. This aids in the downstream task of lemmatization. We designed a function that filters stop words, and feeds the POS-tagged tokens to the NLTK Wordnet Lemmatizer which converts the inflated words such as "better" or "expecting" to their base dictionary form – "good" and "expect". This leads to consistency in the word corpus and reduces redundancy.

Upon pre-processing of reviews, a total of 6171327 words and a total of 61137 unique words are obtained (vocabulary size). We obtained a total of 209673 pre-processed reviews.

Raw Review text	Pre-processed Review
Purchased by my son, he is very happy with this Singing Bowl and the sound it produces	purchase son happy sing bowl sound produce
Product seemed to look nice but 3 plugs in back don't work I was not satisfied And the vu only works part of the time it was a waste of money	product seem look nice plug back not work not satisfied vu work part time waste money
Works okay except with my damper pedal. It keeps playing an E on odd occasions when I press the pedal. I am able to use the the damper when I hook up my old Tascam interface. Problem is, I use the Tascam on another computer.<br /><br />May have to buy an M Audio interface or another Tascam. Thing is that I don't want another box sitting on my desk.	work okay except damper pedal keep play e odd occasion press pedal able use damper hook old tascam interface problem use tascam another computer may buy audio interface another tascam thing not want another box sit desk

Table 1 Processed Review Texts

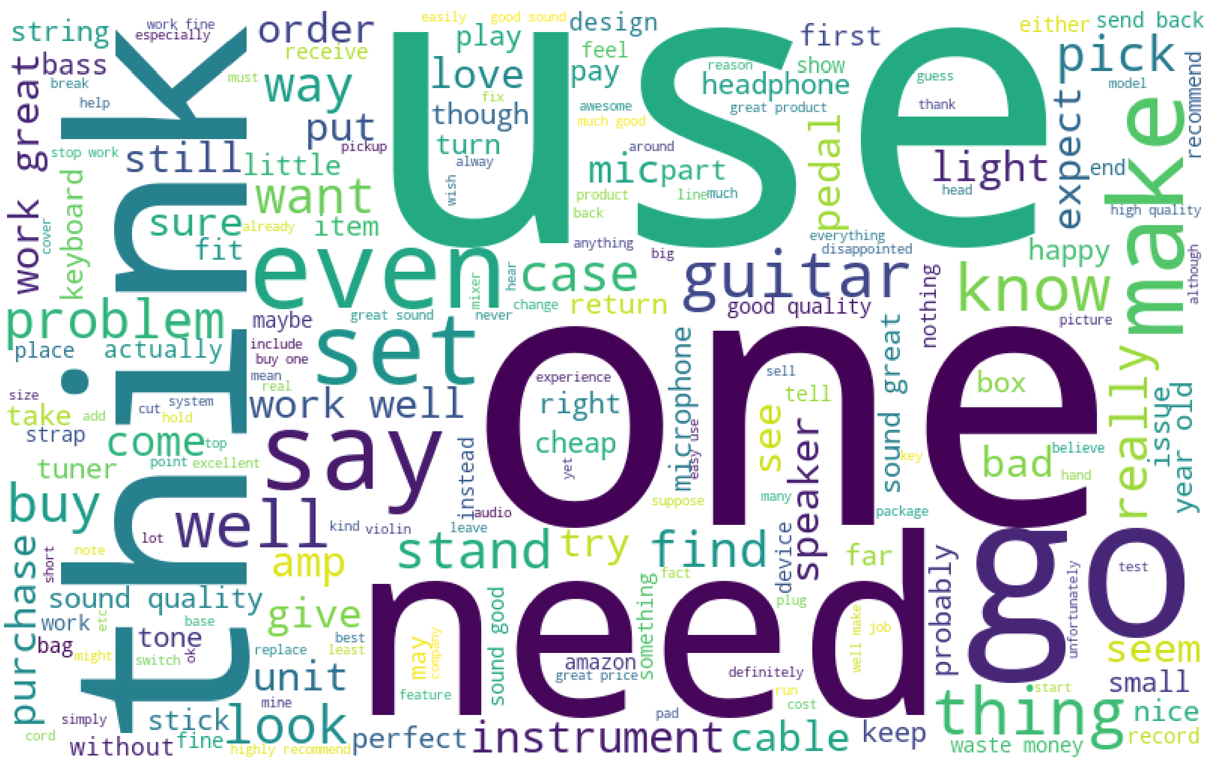


Figure 11 Common words occurring in the pre-processed review corpus

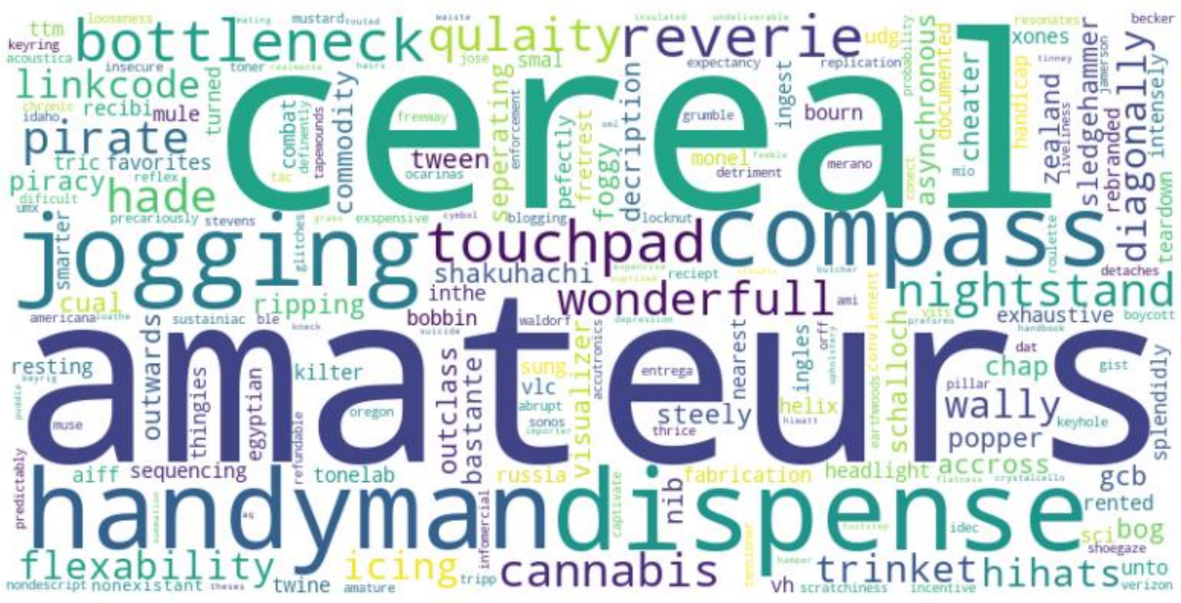


Figure 12 Terms that occur less than 10 times throughout the corpus of pre-processed reviews

4. Feature extraction and feature selection

Document texts are composed of words which cannot be directly fed to the machine learning algorithms. Such models require numeric features having a fixed dimension. To extract numeric features from the review texts we employ two techniques- Bag of N-grams and Tf-IDF vectorization.

Bag of N-grams: The "Bag of N-grams" is a text representation technique that captures the distribution of word or character sequences in a text document and represents text without considering the order of words. It creates a vocabulary (lexicon) by collecting a unique set of words from the corpus of documents. N-grams refer to the number of consecutive words which is taken together as a unit, i.e., 1-gram (one word), bi-grams (2 words) or tri-grams (3 words) N-grams capture the local word order and context in a text which is important for sentiment analysis. For each document, a document vector of vocabulary length is created by considering the counts of the N-gram appearing in that document.

Tf-IDF Vectorization: The words that occur frequently throughout the corpus do not add to the discriminative power of the features and such terms lead to the classification model placing more emphasis on the common words rather than the informative words. To reduce this ambiguity the Term Frequency-Inverse document frequency weighing technique is used. It is given by the formula:

$$\text{TF-IDF}= \text{TF}(a,b) * \text{IDF}(n,D)$$

Where TF(a,b) is the frequency of the term a in document b and

$$\text{IDF}= log\left(\frac{D}{n}\right)$$

D is the total number of documents and n is the number of documents in which the term is present.

An inverse relationship is followed such that terms that occur rarely throughout the corpus but frequently in that particular document are given more weight while placing low weight on commonly occurring terms throughout the corpus. Each document vector is normalised to have a norm of 1.

5. Data Modelling

Sentiment Classification using Machine Learning Models

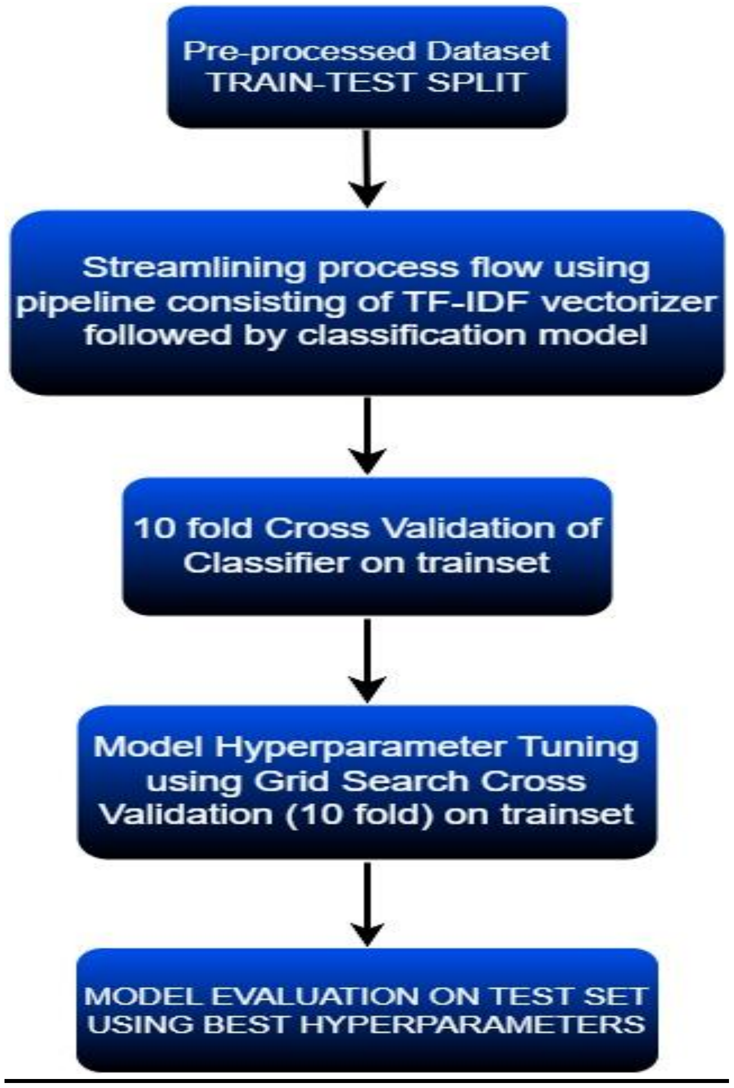

Pre-processed Dataset TRAIN-TEST SPLIT
Streamlining process flow using pipeline consisting of TF-IDF vectorizer followed by classification model
10 fold Cross Validation of Classifier on trainset
Model Hyperparameter Tuning using Grid Search Cross Validation (10 fold) on trainset
MODEL EVALUATION ON TEST SET USING BEST HYPERPARAMETERS

Figure 13 Process flow for review sentiment classification using machine learning models

For predictive modelling, the pre-processed dataset shall be split into train and test sets to obtain an unbiased evaluation of classifier performance indicating the ability to fit the trainset as well as generalize to the unseen dataset. 20% of the data points shall be used as a test set using stratified split to maintain target class distribution. A pipeline will be created consisting of the Tf-IDF vectorizer and the predictive classifier to streamline process flow during model fitting and evaluation.

Cross Validation: Using a single instance of fitting the model on the train data and evaluating it on test-set may result in an optimistic evaluation of the model. We have used 10-fold cross-validation to evaluate the baseline performances of the machine learning classifiers using the default parameters. Instead of relying on a single partition of the data, cross-validation performs 10 splits of train-validation sets with each validation set being stratified to have the same class distribution as the original class distribution, training on one subset and evaluating on the validation set, and then averages the results. A stable estimate of model performance is obtained by mitigating the impact of random variations. Testing the model on multiple validation splits of the data provides a realistic assessment of the generalisation capability of the classifier. This guards against overfitting to a specific data partition, which can occur with a single split.

Multinomial Naïve Bayes Classifier:

The Multinomial Naive Bayes classifier is well well-suited algorithm for text sentiment classification and is an adaptation of the Bayes theorem which estimates the probability of obtaining a target class feature given a set of multinomial input features such as counts of word occurrence or word occurrence frequencies. The Bayes theorem is given by the equation:

$$P(y \mid x_1, \ldots, x_n) = \frac{P(y)P(x_1, \ldots, x_n \mid y)}{P(x_1, \ldots, x_n)}$$

Where $P(y|x_1, x_2, \ldots, x_n)$ is the posterior probability of obtaining output class given a set of input text features. The classifier assumes that the word features are independent of each other. The prior probability P(y) which is the probability of obtaining the target classes is computed from the training dataset. $P(x_1, \ldots x_n|y)$ are the class probabilities which are the likelihoods of obtaining each of the input word features given the distribution of target class labels. Based on the training set, the naïve Bayes classifier builds a decision rule for predicting the class posterior probabilities and tries to maximize it using the objective function. (Raschka, 2017)

Using the default parameters of the tf-idf vectorizer and Naïve Bayes classifier, we have implemented a 10-fold cross-validation to obtain an unbiased evaluation of the classifier on Amazon pre-processed reviews.

Hyper-parameter Tuning of the Multinomial Naïve Bayes Classifier:

To implement a classifier a set of hyperparameters must be provided before the model learning process. The performance of a classifier is evaluated in terms of how well it generalizes to new data. Fine-tuning the hyperparameters helps the models from overfitting the data. The 10-fold Grid-search cross-validation technique has been implemented to tune the hyperparameters of the classifier and the tf-idf vectorizer to obtain the best classifier in terms of classifier performance. A pipeline consisting of the tf-idf vectorizer and Naïve Bayes classifier has been passed to the Grid-search cross validator.

The range of hyper-parameters to be tuned is outlined below:

- *Tf-idf N-gram range*: [(1,1), (1,2), (1,3)] The tf-idf n-gram range decides how many consecutive words together compose a feature unit. The binary and ternary ngram units are composed of two and three words and can better capture the context of the documents than the one-gram feature. However, this leads to a higher dimensionality of sparse input feature vectors.

- *Tf-idf Minimum Document frequency:* [10,100] In a corpus of documents there are rare words that must be removed while building the tf-idf feature vectors. These rare words cause the model to capture irrelevant noise and lead to overfitting the training set. The min-df parameter filters out the words if they appear in fewer documents than the set threshold of 10 and 100 documents.
- *Tf-idf Maximum Document frequency*: [0.6,0.8] Words of high frequency in the corpus add to the noise captured by the classifier. Words that appear in more than 60% or 80% of the documents shall be discarded.
- *Alpha:* [0.001,0.01,0.1, 1.0] While calculating the posterior probability for a test text feature set that does not appear in the training set, zero class conditional probabilities are obtained leading to a final estimation of zero. To avoid this a small smoothing constant alpha is added to the counts of each text feature to avoid getting zero class conditional probabilities. This allows the classifier to generalize to unseen data.

  An exhaustive search will be made over the hyperparameter space using grid search cross-validation such that the best classifier hyperparameters and tf-idf vectorisation parameters are obtained. For each hyperparameter set the model will be trained and validated using the 10 disjoint train-validation splits and the results averaged.

Logistic Regression Classifier:

Logistic regression is used to predict the probability of a target class given a set of input features by modelling the relationship between the input features to the target feature using the sigmoid function. A linear relationship between the log odds of the probability of the target feature (logits) and the weighted combination of the input features is assumed. The predicted probability of the target feature belonging to the positive class logistic sigmoid function:

$$P(Y|z) = \frac{1}{1+e^{-z}}$$

Where $z = \sum_{i=1}^{n} w_i x_i + b$

The model weights are updated during the training process using the optimization function to minimize the error between the predictions and the true target values. The output probability is obtained between 0 and 1 which is assigned as positive class if probability is greater than 0.5. (Geron, 2019)

The pre-processed review texts are fed to the pipeline consisting of tf-idf vectorizer and logistic regression classifier to implement a 10-fold cross-validation obtaining an evaluation of classifier performance using default model parameters.

Hyper-parameter tuning of Logistic Regression Classifier

For fine-tuning the classifier, the following set of hyperparameters shall be tuned:

Tf-idf N-gram range: [(1,1), (1,2), (1,3)],

Tf-idf Minimum Document frequency: [10,100],

Tf-idf Maximum Document frequency: [0.6,0.8],

Logistic Regression Regularisation constant C: [1e-4,0.001,0.01,0.1,1]

Regularisation is introduced in the classifier to force the model to take smaller feature weight updates, which prevents the model from closely fitting the sample points and can generalize well to unseen data. Hyperparameter C is inversely proportional to regularisation strength such that lower values of C introduce high regularisation in the model. A regularisation term which is the regularisation constant time the square of the weights is added to the log loss function such that with each weight update iteration they are forced to take on smaller values while allowing the model to fit the data.

The pre-processed train dataset is fed to the 10-fold Grid-search cross validator and over the total course of iterations, the best hyperparameters of the logistic classifier are obtained. The model is refit on the whole train data and evaluated on the test dataset.

Linear Support Vector Machine:

It is used for binary or multiple-class classification and is based on the concept of maximum margin classification using support vectors. The goal is to set up two parallel marginal planes passing through the sample points (support vectors) nearest to the hyperplane dividing the two target class samples. This marginal distance between the decision hyperplane and sample points is maximized while accurately classifying the sample points.

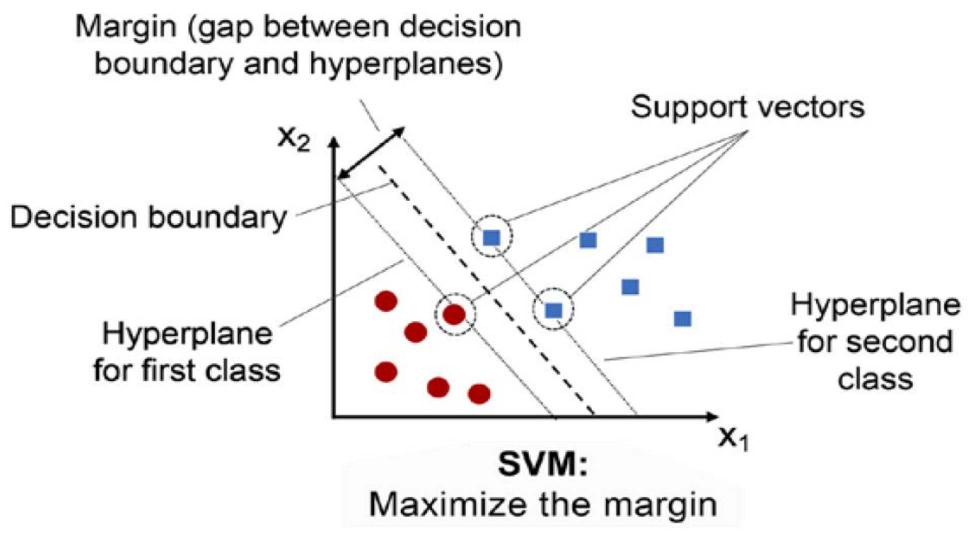

Margin (gap between decision boundary and hyperplanes)
Support vectors
x_2
Decision boundary
Hyperplane for first class
Hyperplane for second class
x_1
SVM:
Maximize the margin

Figure 14 Maximisation of Marginal plane distance between hyperplane and support vectors in Support Vector Classifier (Sebastian and Vahid, 2019)

The marginal distance between the two classes is given by the following expression:

$$\frac{2}{||w||}$$

here $||w||$ is $\sqrt{\sum_{i=1}^{p} w_i}$

w being the weights corresponding to features i to p.

This marginal distance is maximized while trying to accurately predict the sample points using the given constraint:

$$y^{(i)}(w^T x_i + b) \geq 1$$

When predicting a sample point, when the prediction is greater than 1 and has the same sign as the true label it indicates the point is above or below the marginal plane and is correctly classified. Thus, the objective function becomes the maximisation of the marginal distance from the hyperplane to the nearest sample points of each class while ensuring the data points are accurately classified using the above constraint equation. Such a separation of sample points is called hard margin classification and leads the model to overfit the data.

<u>Hyper-parameter tuning of Support Vector Classifier</u>

For fine-tuning the classifier, the following set of hyperparameters shall be tuned:

Tf-idf N-gram range: [(1,1), (1,2), (1,3)],

Tf-idf Minimum Document frequency: [10,100],

Tf-idf Maximum Document frequency: [0.6,0.8],

Regularisation constant C= [1,1e-1, 1e-2, 1e-3,1e-4]

To allow a relaxation in the penalization of predictions that fall within the marginal distance, some of the points of the points are allowed to be misclassified while retaining a wider margin. This technique is called soft margin classification introducing regularisation in the model and allowing it to better generalize to unseen data. It is controlled by the parameter C, such that higher values of C lead to the model being stricter about misclassification, whereas a lower value leads the model to develop a relaxed wider margin.

Sentiment Classification Using Deep Learning Models

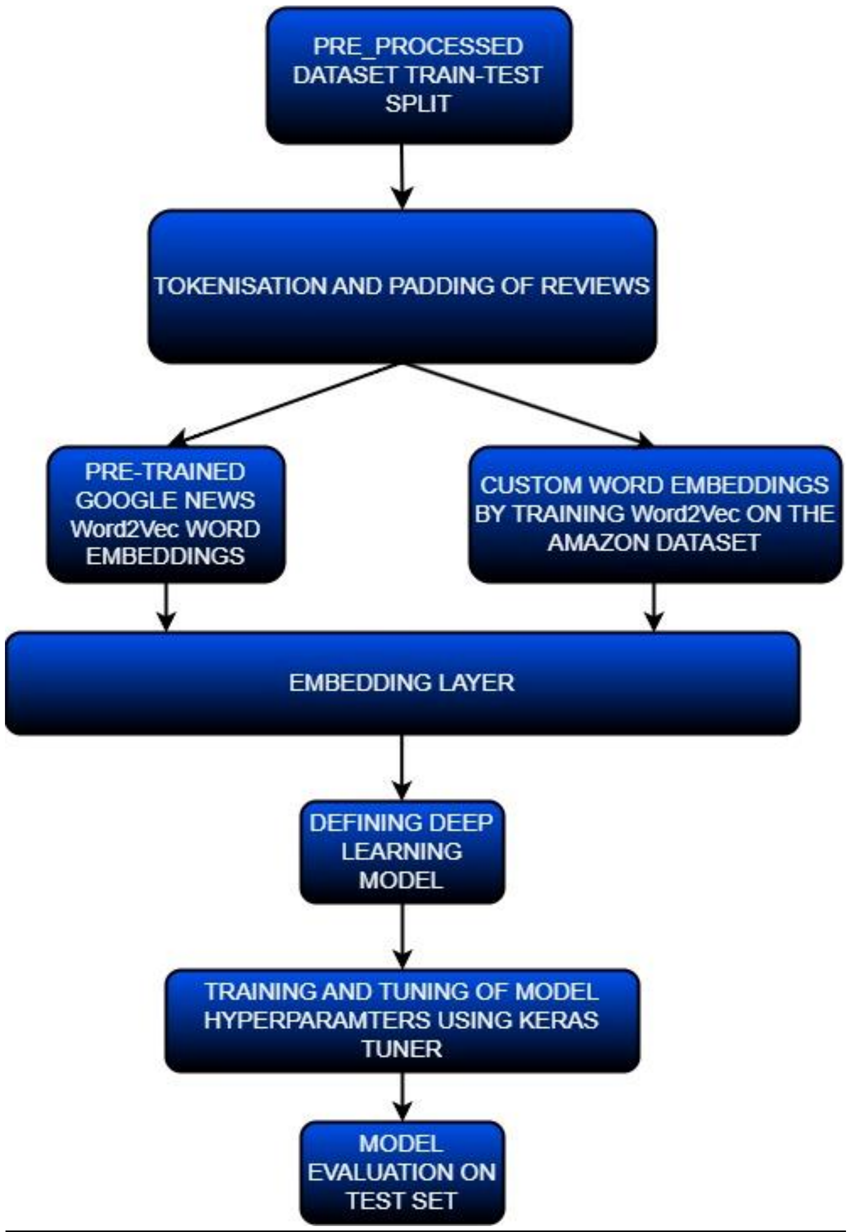

PRE_PROCESSED DATASET TRAIN-TEST SPLIT
TOKENISATION AND PADDING OF REVIEWS
PRE-TRAINED GOOGLE NEWS Word2Vec WORD EMBEDDINGS
CUSTOM WORD EMBEDDINGS BY TRAINING Word2Vec ON THE AMAZON DATASET
EMBEDDING LAYER
DEFINING DEEP LEARNING MODEL
TRAINING AND TUNING OF MODEL HYPERPARAMTERS USING KERAS TUNER
MODEL EVALUATION ON TEST SET

Figure 15 Process flow for sentiment classification of Amazon reviews using deep learning models

Train- Validation split: The pre-processed review dataset is split into stratified train(80%), and validation(20%) sets. The neural network shall be trained on the trainset, its hyperparameters tuned with the validation set and a final evaluation of classifier performance shall be performed on the test set.

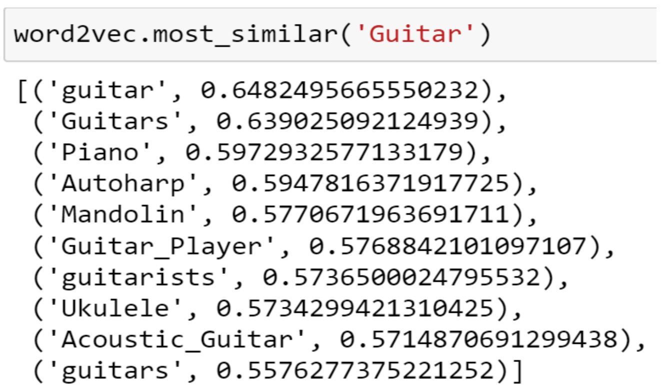
Tokenisation and Padding of reviews: The pre-processed datasets are tokenised using the Keras tokenizer. The tokenizer builds a vocabulary (a unique set of words) from the corpus of train review texts breaks each review into the substituent words and assigns an integer token number to each word corresponding to the vocabulary. The maximum length of a review is

2199 and 85% of the reviews have a maximum of 51 tokens. As the neural networks require inputs of fixed length, each tokenized review shall be padded with zeroes at the start of the vectors to convert them to have a fixed length of 50 tokens. The shape of the padded train, validation and test review sets are (146804, 50), (31458,50) and (31458, 50).

Word2Vec Embedding Matrix:

The Word2vec algorithm was proposed by (Mikolov *et al.*, 2013) which is an unsupervised technique to represent words as continuous vectors. A shallow neural network consisting of an input layer, a hidden layer and an output softmax layer is used in the word2vec algorithm. A pretrained word2vec word embedding has been used which had been obtained by training on a large corpus of Google news dataset containing 1.6 billion words.

A custom word embedding has been obtained by training the word2vec Continuous Bag of words algorithm (CBOW) on the Amazon musical reviews dataset. In CBOW, a target word is predicted from the surrounding window of context words. Depending on the window size, one hot encoded vector of the context words is passed to the hidden layer having 300 neurons (dimension of the word embedding) and the probability of the target word is obtained from a softmax layer after implementing a feedforward pass. This window is slid to cover the entire corpus thus obtaining the word embedding vectors from the weights of the hidden layer. The word embeddings obtained contain the contextual and semantic meaning of the word in a unique vector in a continuous vector space of 300 dimensions such that similar words have vectors that are closer in the vector space.



```
word2vec.most_similar('Guitar')
```

```
[('guitar', 0.6482495665550232),
 ('Guitars', 0.639025092124939),
 ('Piano', 0.5972932577133179),
 ('Autoharp', 0.5947816371917725),
 ('Mandolin', 0.5770671963691711),
 ('Guitar_Player', 0.5768842101097107),
 ('guitarists', 0.5736500024795532),
 ('Ukulele', 0.5734299421310425),
 ('Acoustic_Guitar', 0.5714870691299438),
 ('guitars', 0.5576277375221252)]
```

For each of the 54,262 words in the tokenized reviews vocabulary, we have extracted the 300-dimensional vectors from the pre-trained Google word2vec dataset corresponding to that word and created an embedding matrix, such that each i-th vector in the matrix will correspond to the i-th indexed word in the vocabulary.

We have used Google pre-trained word embeddings and the custom-trained word embeddings to train the neural networks separately.

Embedding Layer:

We will use an embedding layer as the first layer of the Artificial Neural Network (ANN) and Recurrent Neural Network – Long Short Term Memory (RNN-LSTM) models. The Keras Embedding layer shall take the sequences of integer-encoded reviews as input. The word2vec embedding matrix previously created shall be used as the weight initializer for the embedding layer. The weights shall not be updated, and the layer shall be frozen to preserve the pre-trained weights of the word2vec vectors. The input shape is (batch_size, sequence_length=50) and for each word within the review, it shall do a lookup in the word2vec embedding matrix and output a sequence of word embeddings corresponding to the input sequence. The output shape is (batch_size=128, sequence_length=50, embedding_dim=300). The input_dim parameter is set to 54262 the size of the vocabulary (number of unique words in tokenizer +1).

Artificial Neural Network (ANN):

The ANN is a powerful algorithm capable of capturing complex patterns in data which is a mathematical adaptation of the human brain. The fundamental component of an ANN is the neuron also called perceptron which takes a vector of weighted features, applies a net input function and finally an activation function to get the output.

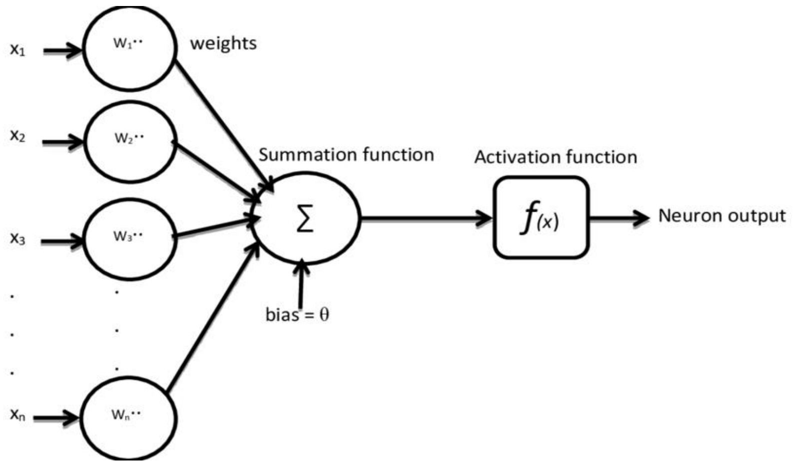


Figure 16 Basic Unit of ANN- Perceptron (Yacim and Boshoff, 2018)

For a vector of input features, unique weights small in magnitude are initialized randomly using a weight initializer for each feature-to-neuron connection. A bias term is added for the the ANN to capture a baseline offset value allowing the ANN to model a relationship with the output even when all the inputs are zero. The different weights allow the model to adjust the importance of each feature accordingly. The output of a perceptron is given by:

$$y = f\left(\sum_{i=1}^{\mathbb{N}} w_i x_i + b\right)$$

Where f is the activation function applied, w_i is the weight applied for each input x_i and bias term for the layer.

A deep neural network consists of several hidden layers, each layer containing several neurons. Each neuron in a layer is connected to each of the neurons in the next layer via multiple weighted connections. The neuron receives the weighted inputs and applies a summation and the activation function (introduces non-linearity) to pass on the output to the next layer. This is called the feedforward pass wherein the model passes the inputs through the layers to obtain an output through the final output layer. The output prediction is evaluated via the loss function to determine the gap between the actual and the predicted value. The error is backpropagated through each of the neurons updating their weights with chained differentiation with respect to the loss with the help of the optimizer. The step-size by which the model weights are updated with each iteration to reach the global minimum of the loss function is the learning rate. This feedforward and backpropagation are iterated several times (epochs) over the training data updating the weights along the way such that the model can fit complex representations of the training data.(Sebastian and Vahid, 2019)

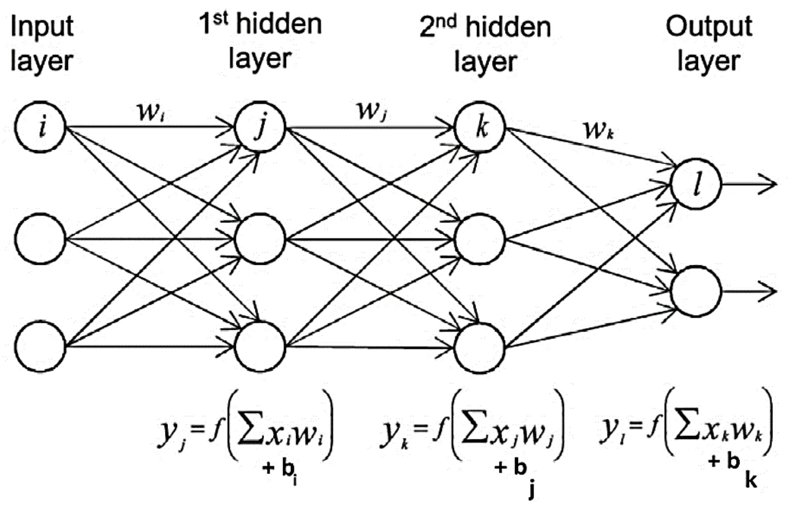

Input layer
1st hidden layer
2nd hidden layer
Output layer
i
j
k
l
w_i
w_j
w_k
$y_j=f\left(\sum x_i w_i + b_i\right)$
$y_k=f\left(\sum x_j w_j + b_j\right)$
$y_l=f\left(\sum x_k w_k + b_k\right)$

Figure 17 Architecture of an ANN(Vieira, Pinaya and Mechelli, 2017)

Keras Tuner:

The Keras Tuner is designed to tune the hyperparameters of a neural network such as the number of layers, the number of neurons in each layer and the learning rate of the optimization function for the best classifier performance with the help of a hyperparameter object hp. (O'Malley *et al.*, 2019) We have created a model function for the ANN model by initializing the Keras Sequential layers model. The first layer has been set as the embedding layer receiving the padded integer review token sequences and shall output the word2vec embeddings corresponding to each review for the downstream layers. The embedding output is flattened into a one-dimensional vector for the subsequent downstream dense layers. To tune the number of layers and the number of neurons in each layer, a loop has been created such that the Keras tuner will randomly search through the loop choosing the number of layers ranging from 1 to 20. For each layer, the tuner will randomly choose the number of neurons from a range of 32 to 512 with a step size of 32. A dropout layer has been introduced which shall randomly drop a percentage of input values with a given probability. This prevents the model from overfitting allows it to depend less on the exact learned weights and develops new internal generalized representations of the data. (Srivastava *et al.*, 2014) The dropout rate shall be tuned from a choice of 0.2, 0.3 or 0.5. The final dense output layer has an activation of sigmoid which shall output a value ranging from 0 to 1 indicating the probability of input data belonging to the target class. The random search tuner is set to search through the hyperparameter space for 10 trials such that in each trial two ANN models will be trained using the same hyperparameters and their resulting evaluation scores with the validation set will be averaged over each trial. The executions per trial have been set to 2 to avoid biases as the stochastic nature of neural networks results in varying results. The best model and hyperparameter set shall be obtained after searching through 10 trials.

Recurrent Neural Network- Long Short Term Memory (RNN-LSTM):

In machine learning algorithms and ANN, the input text features are passed as input without taking the ordering of words into context and suffer from the assumption that the features(words) are independent of each other. But in sequences of text, the order of words is of high importance. This is considered by Recurrent Neural Networks which passes one feature word vector, processes it through a dense neural network and uses this output as an additional input for the next hidden layer for processing the next word. This introduces memory into the network and the sequence of words is remembered to make the prediction. However, RNNs suffer from a problem of vanishing gradient descent and exploding gradient descent when updating the weights of the hidden layer neurons due to the long-chained differentials of the loss with respect to hidden states being developed. To address this issue, we have used the Long Short-Term Memory model.

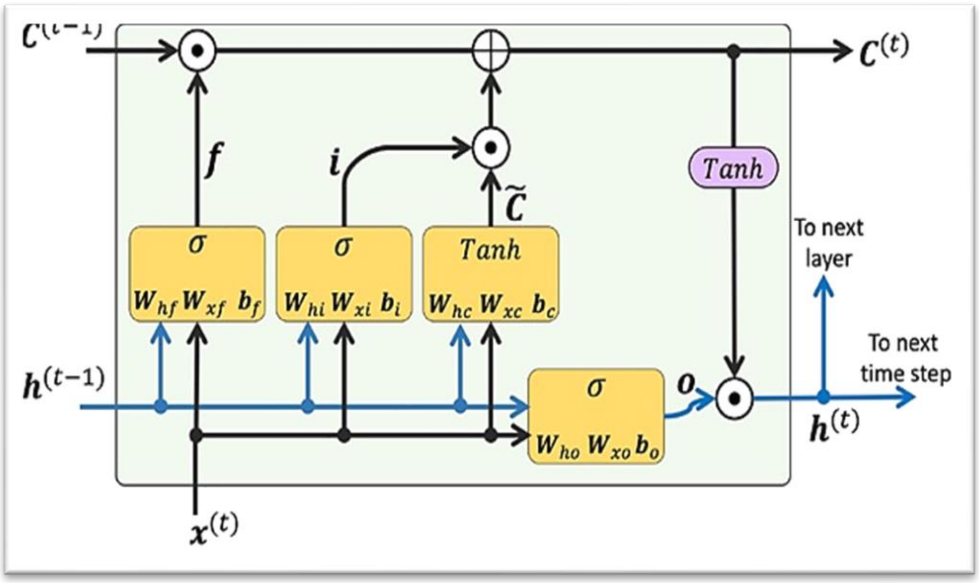


Figure 18 Architecture of an LSTM cell (Sebastian and Vahid, 2019)

The fundamental unit of an LSTM model is the memory cell which is a separate component that can store and update information over time and capture long-term word dependencies over 1000 time steps. It has two states- the cell state (long-term memory) and the hidden state (short-term memory). The updating of the cell state C is implemented without associating with any weights of the hidden layers which allows the LSTM to avoid the vanishing gradient problem. LSTM consists of three gates:

Forget gate: In the forget gate, the output vector from the previous memory cell time step h_{t-1} and the current time step input vector x_t undergoes matrix multiplication with hidden and input weight matrices (W_{hf}, W_{xf}), and the bias b_f. The output obtained is applied to the sigmoid activation function σ which squashes the output to a value between 0 and 1 and an element-wise multiplication is implemented with previous cell state C_{t-1}. This decides whether to retain or forget the previous long term cell state C_{t-1}.

Input Gate: The previous hidden states and current input vector are applied to the sigmoid function and tanh activation function and their element-wise addition output is multiplied with the previous cell state C_{t-1}. This gate decides the amount of information from the current input to be added to the long-term memory based on the short-term memory.

Output gate: This gate regulates how much of the data in the long-term memory cell should be output. To calculate the output, it considers both the previous hidden state and the current input vector. This output is passed onto the next memory cell and over the course of the total number of memory cells, the final output vector is obtained for a single review text sequence. (Hochreiter and Schmidhuber, 1997)

We have decided to tune the hyperparameters of the LSTM model using Keras Tuner. A hypermodel object will be created which will dynamically create and configure the LSTM model based on the hyperparameter space. The random search tuner will be used to search through the following hyperparameter space:

Number of LSTM Layers: [1, 2, 3]

Number of memory cells per layer: [a choice ranging from 16 to 96 with a step size of 16]

Dropout Rate: [0.2, 0.3, 0.5]

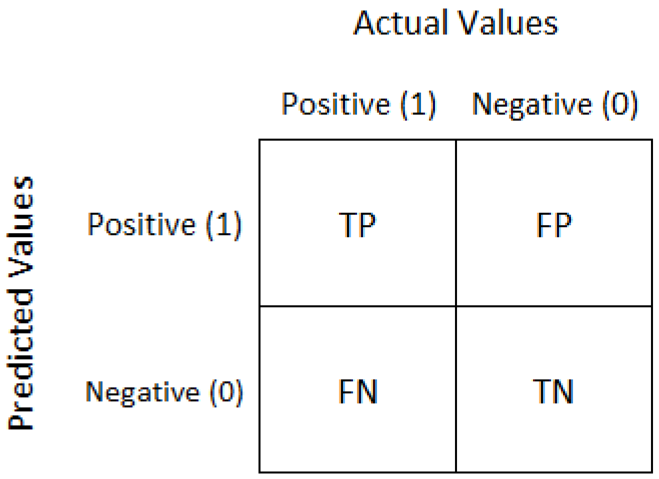
The function compiles the model using the Adam optimizer with a learning rate determined by the learning rate hyperparameter. The loss function is set to 'binary_crossentropy', which is appropriate for binary classification.

The function dynamically builds an LSTM-based neural network model with varying numbers of LSTM layers, units per layer, and other hyperparameters based on the tuning setup. It's designed to provide flexibility in architecture while leveraging hyperparameter tuning to find the optimal configuration for the given task.

6. Results and Model Evaluation

For evaluating the classification models in classifying the Amazon reviews the following evaluation metrics have been used:

1. Confusion Matrix: It is a matrix that gives an in-depth look into the model's predictions and actual results, making it easier to assess the model's accuracy and pinpoint areas for development. An evaluation is obtained as to whether the predicted values how many are actually true (TP and TN) and whether the predicted values do not match the true labels (FP and FN).



		Actual Values	
		Positive (1)	Negative (0)
Predicted Values	Positive (1)	TP	FP
	Negative (0)	FN	TN

(Narkhede, 2021)

2. *Accuracy:* The accuracy of a classifier is given by:

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

It gives a measure of the proportion of predictions made for the test set which belongs to the true label out of all the predictions made.

3. *Precision:* The precision of a classifier is given by:
$$\frac{TP}{TP + FP}$$
It is a measure of the proportion of correct predictions of the positive class made by the classifier out of all positive predictions. It is of importance if the problem requires a lower number of false positive predictions.
4. *Recall:* The recall of a classifier is given by:

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

It is a measure of the proportion of correct predictions of the positive class out of all the actual positives. It is of importance if the problem requires a higher number of positive predictions to be true of all the actual positives.

5. *F1-score*: The F1-score of a classifier is given by:

$$2 \times \left(\frac{P \times R}{P + R}\right)$$

Where P is the Precision and R is the Recall of the classifier. F1-score gives an equal weight to both precision and recall considering both false positives and false negatives.

Machine Learning Models:

Multinomial Naïve Bayes Classifier:

We implemented a 10-fold cross-validation of the naïve Bayes classifier on the whole Amazon pre-processed reviews set. The following metrics were obtained after averaging the results over the 10 folds:

Multinomial Naïve Bayes Classifier					
Accuracy	Precision	Recall	F1-score	Train Time (seconds)	Score Time (seconds)
86.63	86.63	86.63	86.63	12.3408	0.7716

Table.2 Multinomial Naïve Bayes Classifier Cross-validation scores

The precision and recall metrics follow an inverse relationship and the Multinomial Naïve Bayes classifier achieving a good performance across all four metrics indicates that the model has balanced the trade-off between False positives and True negatives and can compromise between minimizing false positives and false negatives.

The Amazon dataset was pre-processed to have a balanced distribution and it was easier for the classifier to accurately predict the reviews. The target class being binary the classifier achieved a good consistency in predicting both the classes.

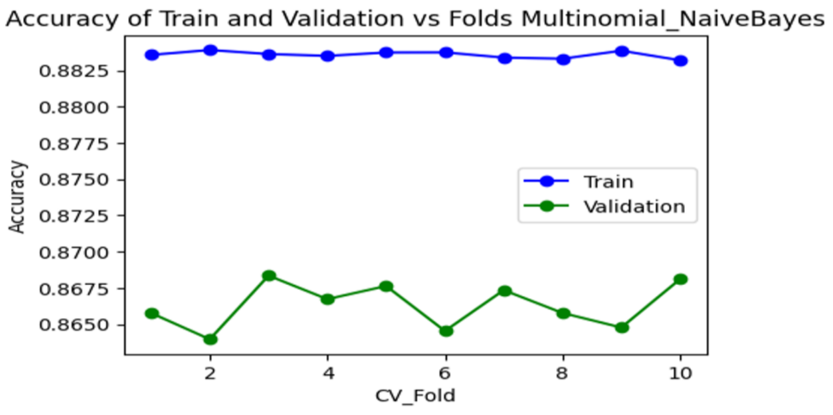


Figure 19 Cross Validation Train and Validation accuracy NB classifier vs. Fold

Grid-Search Multinomial Naïve Bayes Classifier:

We have implemented Grid-search using 10-fold cross-validation using macro F1-score as the evaluation metric to obtain the best hyperparameter set of the classifier in terms of classification performance. The following hyperparameters were found to be the best:

Smoothing constant alpha: 1.0

Tf-idf maximum document frequency: 0.6

Tf-idf minimum document frequency: 10

Tf-idf N-gram range: (1, 3)

The smoothing constant of 1 was found to be the best to generalize to unseen data and was added to the counts of input text features. This was implemented to avoid predicting zero posterior probabilities for estimating test text features that were not seen in the training set.

The ternary N-gram unit of three consecutive words was chosen as it allowed a wider window of context to be captured by the model like phrases and small topics. The semantic knowledge of an expression is better represented by groups of words.

The N-gram terms appearing in more than 60% of the documents in the corpus were discarded to prevent the model from capturing high-frequency irrelevant words and noise. The rare words and N-gram units that appeared in 10 documents or less were also discarded. This helped in improving the generalisation capacity of the model.

The best score obtained during the Grid-search cross-validation was a macro F1-score of 90.24%.

The best model obtained was used to predict the test pre-processed set and the following classification report was obtained:

Multinomial Naïve Bayes classifier using best hyperparameter set							
Class	**Accuracy**	**Precision**	**Recall**	**F1-score**	**Support**	**Train Time(sec)**	**Score Time(sec)**
Negative Class (0)	**0.9013**	0.9043	0.8977	0.9010	20972	52.2047	2.1
Positive Class (1)		0.8984	0.9050	0.9017	20972		
Macro average		**0.9014**	**0.9013**	**0.9013**	**41944**		

Table.3 Multinomial Naïve Bayes Classifier using best hyperparameters classification scores

The confusion matrix obtained from predicting the test set using the best Naïve Bayes model is shown below:

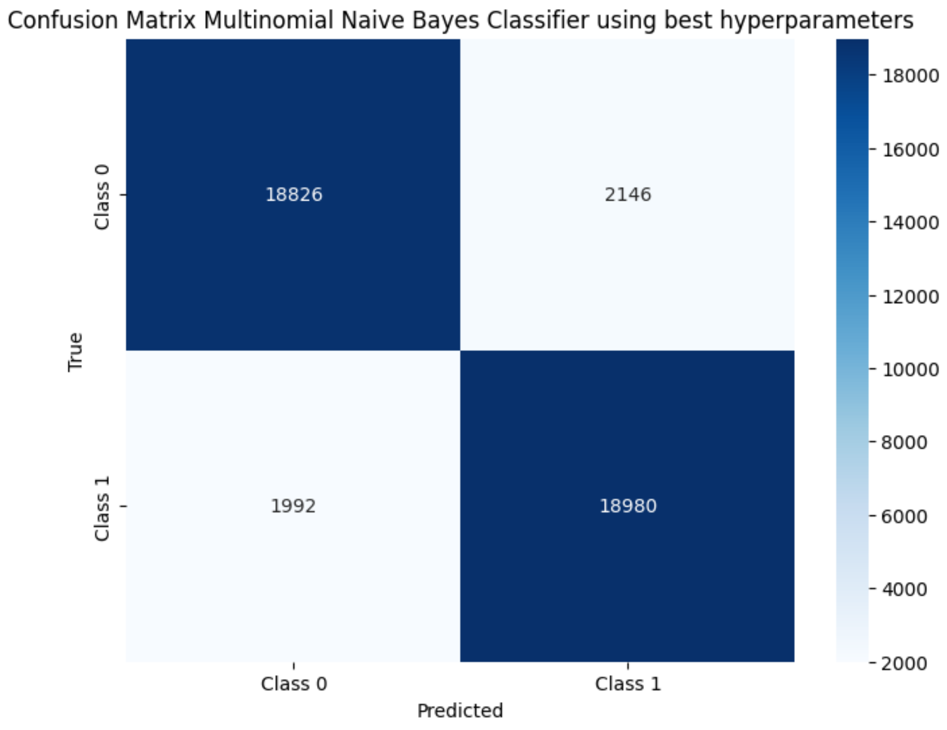


Figure 20 Confusion matrix for Naïve Bayes classifier using best hyperparameters

Logistic Regression Classifier:

We implemented a 10-fold cross-validation of the naïve Bayes classifier on the whole Amazon pre-processed reviews set. The following metrics were obtained after averaging the results over the 10 folds:

Logistic Regression Classifier					
Accuracy	Precision	Recall	F1-score	Train Time (seconds)	Score Time (seconds)
0.8983	0.8983	0.8983	0.8983	20.61	0.86

Table.4 Logistic Regression_Classifier Cross-validation scores

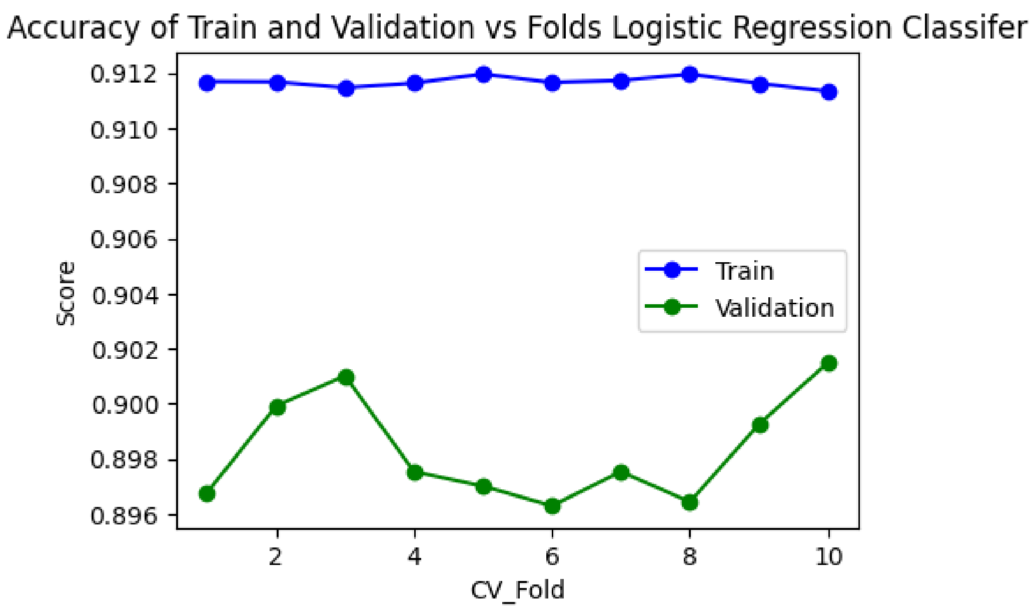


Figure 21 Cross Validation Train and Validation accuracy vs Fold (Logistic Regression classifier)

<u>Grid-Search Logistic Regression Classifier:</u>

We have implemented Grid-search using 10-fold cross-validation using macro F1-score as the evaluation metric to obtain the best hyperparameter set of the classifier in terms of classification performance. The following hyperparameters were found to be the best:

Regularisation Strength C: 1

Tf-idf maximum document frequency: 0.6

Tf-idf minimum document frequency: 10

Tf-idf N-gram range: (1, 3)

The regularisation constant of 1 was found to be the best to generalize to unseen data. It is inversely proportional to the regularisation strength and a medium constant of 1 was appropriate.

The best score obtained during the Grid-search cross-validation was a macro F1-score of 91.67%

The best model obtained was used to predict the test pre-processed set and the following classification report was obtained:

Logistic Regression classifier using best hyperparameter set							
Class	**Accuracy**	**Precision**	**Recall**	**F1-score**	**Support**	**Train Time(sec)**	**Score Time(sec)**
Negative Class (0)	**0.9162**	0.9161	0.9164	0.9163	20972	56.36	1.90
Positive Class (1)		0.9164	0.9161	0.9162	20972		
Macro average		**0.9162**	**0.9162**	**0.9162**	**41944**		

Table.5 Logistic Regression classifier using best hyperparameter classification scores

The confusion matrix obtained from predicting the test set using the best logistic regression model is shown below:

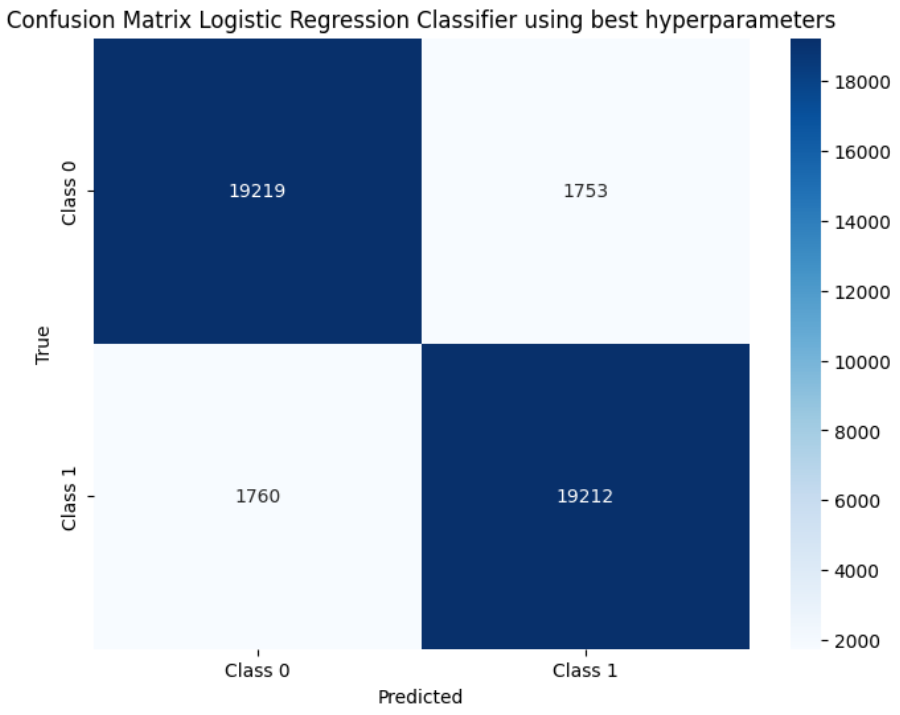


Figure 22 Confusion matrix for Logistic Regression classifier using best hyperparameters

Support Vector Classifier

We implemented a 10-fold cross-validation of the Support Vector classifier on the whole Amazon pre-processed reviews set. The following metrics were obtained after averaging the results over the 10 folds:

Linear Support Vector Classifier Cross-Validation Scores					
Accuracy	Precision	Recall	F1-score	Train Time (seconds)	Score Time (seconds)
0.8952	0.8952	0.8952	0.8952	15.61	0.74

Table.6 Linear Support Vector Classifier Cross-validation scores

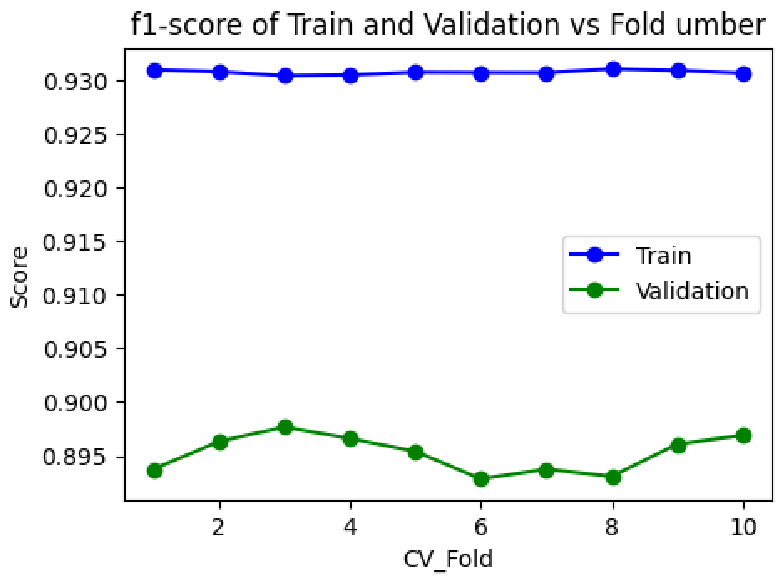


Figure 23 Cross Validation Train and Validation accuracy vs Fold (SVM classifier)

Grid-Search Support Vector Classifier:

We have implemented Grid-search using 10-fold cross-validation using macro F1-score as the evaluation metric to obtain the best hyperparameter set of the classifier in terms of classification performance. The following hyperparameters were found to be the best:

SVC Regularisation Constant C: 0.1

Tf-idf maximum document frequency: 0.6

Tf-idf minimum document frequency: 10

Tf-idf N-gram range: (1, 3)

The regularisation constant of 0.1 was found to be the best to generalize to unseen data. It is inversely proportional to the regularisation strength and allows less penalization to sample points within the support vector margin.

The best score obtained during the Grid-search cross-validation was a macro F1-score of 91.85%

The best model obtained was used to predict the test pre-processed set and the following classification report was obtained:

Support Vector Classifier using best hyperparameter set							
Class	**Accuracy**	**Precision**	**Recall**	**F1-score**	**Support**	**Train Time(sec)**	**Score Time(sec)**
Negative Class (0)	**0.9182**	0.9177	0.9187	0.9182	20972	54.93	2.00
Positive Class (1)		0.9187	0.9177	0.9182	20972		
Macro average		**0 .9182**	**0. 9182**	**0. 9182**	**41944**		

Table.7 Support Vector Classifier using best hyperparameter classification scores

The confusion matrix obtained from predicting the test set using the best logistic regression model is shown below:

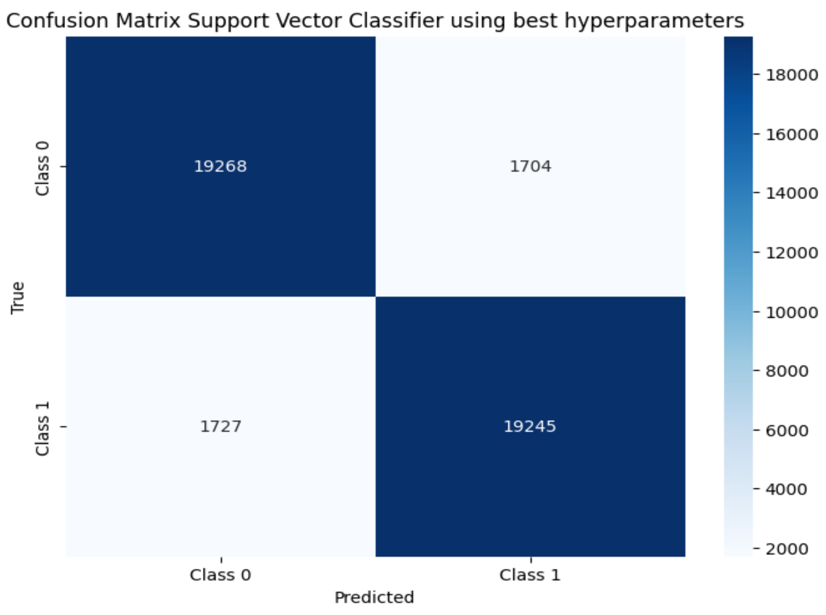


Figure 24 Confusion matrix for SVM classifier using best hyperparameters

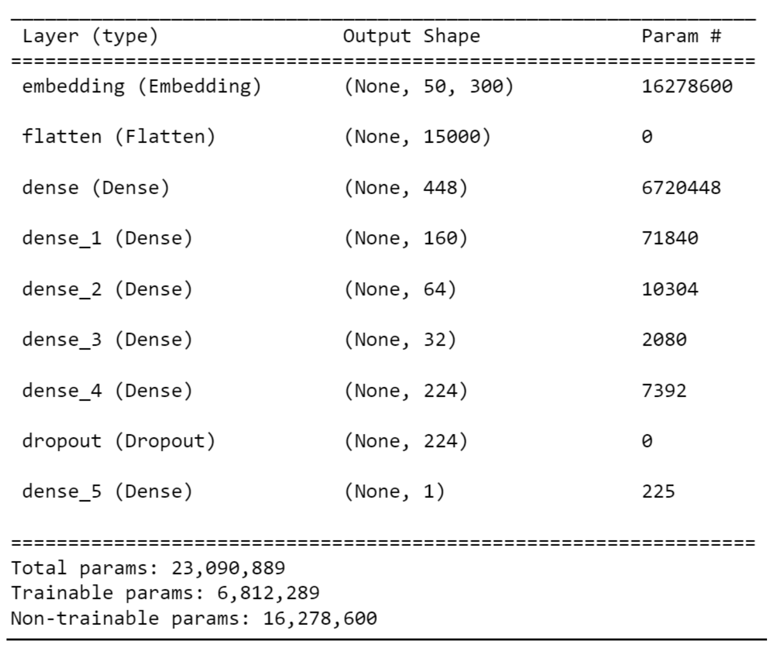
Deep Learning Models:

To obtain the word embeddings of the tokenized and padded reviews, we have used the Word2vec algorithm. Two different word embeddings were obtained- the pre-trained Google word2vec word embeddings and a word embedding matrix obtained by training the Word2vec algorithm on the Amazon Musical reviews dataset from scratch. The ANN and RNN-LSTM models were trained using these word embeddings separately and the hyperparameters were tuned using the Keras tuner. The best set of hyperparameters was obtained by randomly searching through the hyperparameter space for 10 trials. For each trial, one set of hyperparameters was chosen and the neural network was trained and evaluated two times to obtain an unbiased evaluation score. The number of epochs for the training algorithm was set to 50 with the imposition of the early stopping constraint such that if the validation accuracy was not improving after 3 training epochs, the neural network training process was stopped, and the best weights were restored.

Neural Network Evaluation using Google pre-trained Word2vec Word embeddings:

Artificial Neural Network (ANN):

After implementing tuner Random search through the hyperparameter space the ANN model built using the best set of hyperparameters is outlined below:



```
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 embedding (Embedding)       (None, 50, 300)           16278600

 flatten (Flatten)           (None, 15000)             0

 dense (Dense)               (None, 448)               6720448

 dense_1 (Dense)             (None, 160)               71840

 dense_2 (Dense)             (None, 64)                10304

 dense_3 (Dense)             (None, 32)                2080

 dense_4 (Dense)             (None, 224)               7392

 dropout (Dropout)           (None, 224)               0

 dense_5 (Dense)             (None, 1)                 225

=================================================================
Total params: 23,090,889
Trainable params: 6,812,289
Non-trainable params: 16,278,600
_________________________________________________________________
```

Figure 25 ANN using Google Word embeddings architecture summary

The best Hyperparameters obtained are outlined below:

Number of layers: 5
Number of neurons in layer 1: 448
Number of neurons in layer 2: 160
Number of neurons in layer 3: 64
Number of neurons in layer 4: 32
Number of neurons in layer 5: 224
Learning Rate of optimizer: 0.0001
Dropout Rate: 0.2

The classification report is outlined below:

ANN using Google pre-trained Word2vec embeddings					
Class	**Accuracy** (percentage)	**Precision** (percentage)	**Recall** (percentage)	**F1-score** (percentage)	**Support**
Negative Class (0)	**85.26**	83.54	87.82	85.63	20972
Positive Class (1)		87.16	82.69	84.87	20972
Macro average		**85.35**	**85.26**	**85.25**	**41944**

Table.8 ANN using Google pre-trained Word2vec embeddings classification scores

It is observed that the negative class has a higher recall than precision which indicates that ou t of the true negative labels fewer were misclassified as positive, however comparatively a gr eater number of true positive classes were misclassified as negative. The ANN classifier achi eved a decent accuracy of 85.26% which is less than the machine learning models and LSTM accuracy.

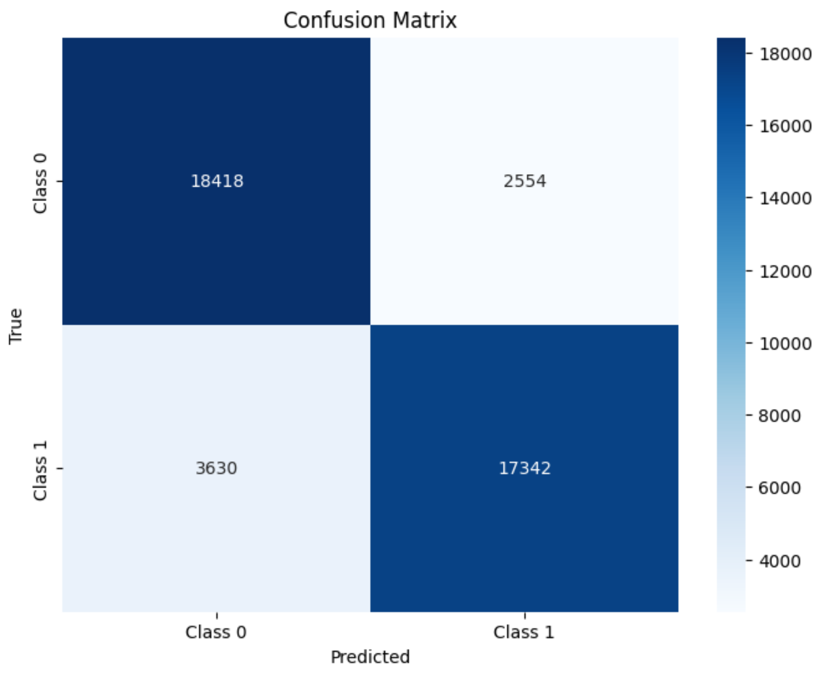
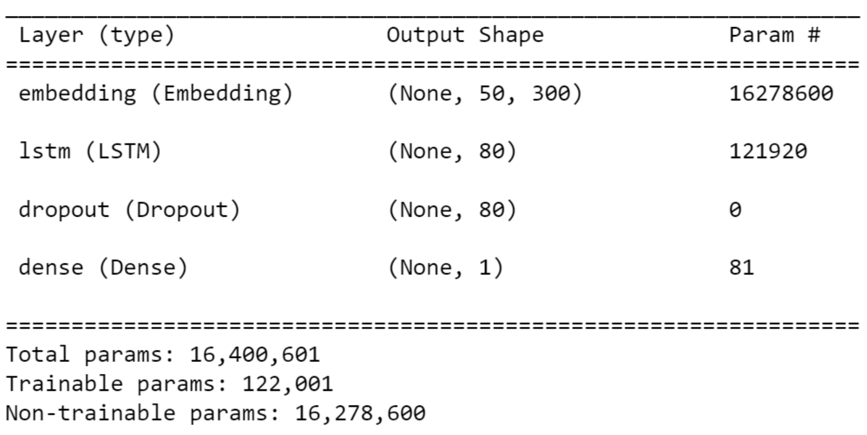


Figure 26 Confusion matrix of ANN model using Google word embeddings

It is observed that the negative class has a greater number (18418) of correctly predicted sample points than the positive class. However, a greater number of actual positives were misclassified as negative (3630).

RNN-LSTM:

After implementing tuner Random search through the hyperparameter space the LSTM model trained using the Google pre-trained word embedding was built using the best set of hyperparameters which is outlined below:



```
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 embedding (Embedding)       (None, 50, 300)           16278600

 lstm (LSTM)                 (None, 80)                121920

 dropout (Dropout)           (None, 80)                0

 dense (Dense)               (None, 1)                 81

=================================================================
Total params: 16,400,601
Trainable params: 122,001
Non-trainable params: 16,278,600
```

Figure 27 LSTM using Google Word embeddings architecture summary

A single LSTM layer was used with 80 memory cells and a dropout rate of 0.2
and an optimizer learning rate of 0.001 was used which achieved a validation accuracy of 91. 90%.

The classification report is outlined below:

RNN-LSTM using Google pre-trained Word2vec embeddings					
Class	**Accuracy** (percentage)	**Precision** (percentage)	**Recall** (percentage)	**F1-score** (percentage)	**Support**
Negative Class (0)		91.95	91.88	91.92	20972
Positive Class (1)	**91.92**	91.89	91.96	91.92	20972
Macro average		**91.92**	**91.92**	**91.92**	**41944**

Table.9 LSTM using Google pre-trained Word2vec embeddings classification scores

The LSTM model achieved an accuracy of 91.92% which outperformed the machine learning models and the ANN model trained using the pre-trained embeddings. The negative class has a higher precision of 91.95% than the positive class implying an ability of the model to accurately predict the negative class.

The confusion matrix is outlined below:

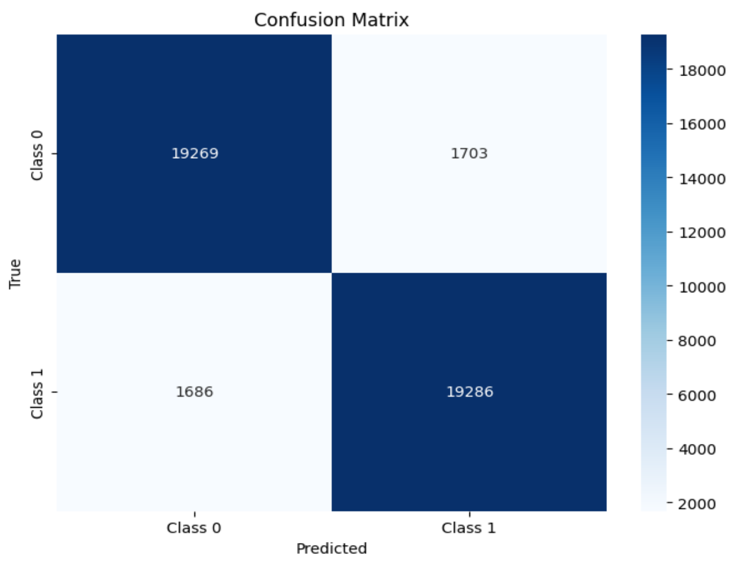


Figure 28 Confusion matrix of LSTM model using Google word embeddings

The positive class has a higher number of correctly classified samples (true positives) than the negative class (19286 samples) however it has a higher number of false positives than the negative class.

<u>Evaluation of Neural Network models using custom Word2vec Word embeddings trained on the Amazon dataset:</u>

<u>Artificial Neural Network (ANN) using custom-trained word embeddings:</u>

After implementing tuner Random search through the hyperparameter space the ANN model was built using the best set of hyperparameters the classification report is outlined below:

ANN using custom Word2vec Word embeddings trained on the Amazon dataset					
Class	**Accuracy**	**Precision**	**Recall**	**F1-score**	**Support**
	(percentage)				
Negative Class (0)	**87.98**	87.93	88.04	87.98	20972
Positive Class (1)		88.03	87.91	87.97	20972
Macro average		**87.98**	**87.98**	**87.98**	**41944**

Table.10 ANN using custom-trained Word2vec Word embeddings classification scores

The ANN classifier trained using the custom-trained embeddings achieved a higher accuracy than the ANN trained using the pre-trained Google word embeddings.

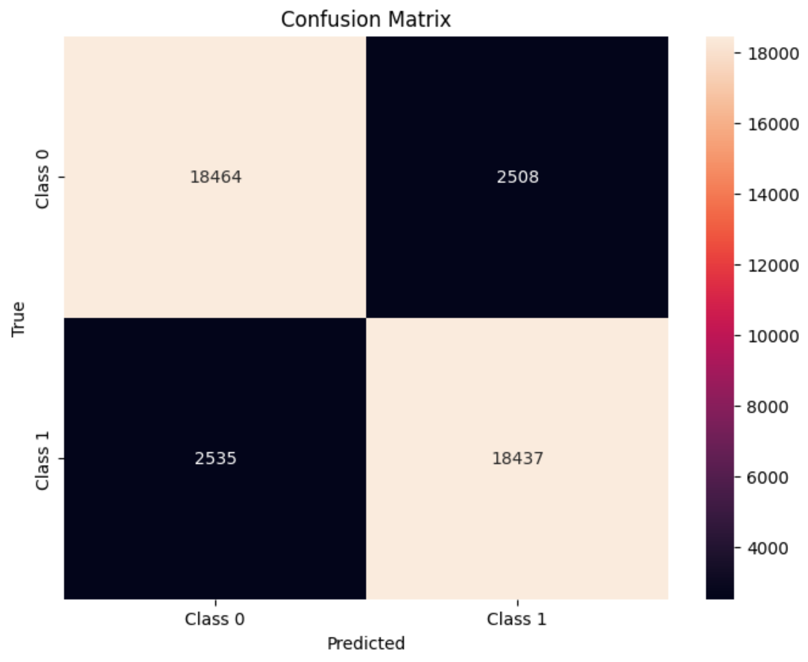
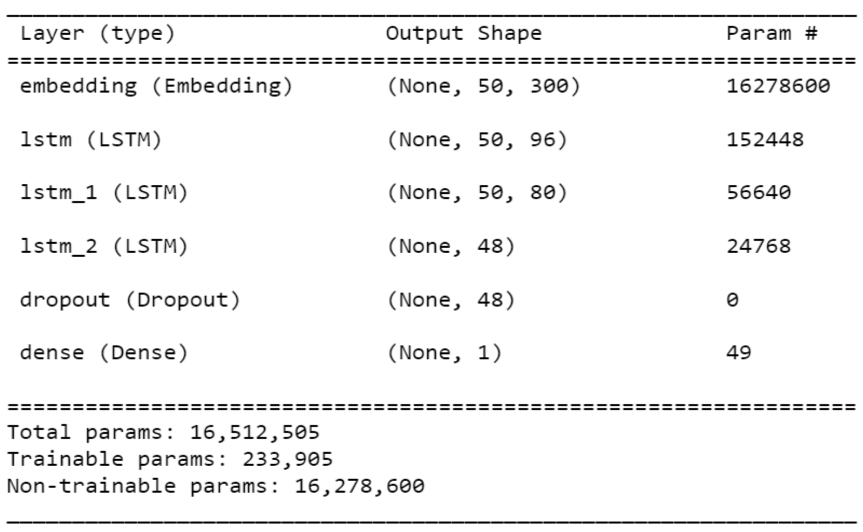


Figure 29 Confusion matrix of ANN model using custom-trained word embeddings

It is observed that the negative class has a greater number of correctly predicted sample points than the positive class (true negatives). The negative class has a lower precision than the positive class.

RNN-LSTM USING CUSTOM TRAINED WORD EMBEDDINGS:

After implementing tuner Random search through the hyperparameter space the LSTM model trained using the custom-trained word embeddings was built using the best set of hyperparameters which is outlined below. Three LSTM layers were used with 96,80 and 48 memory cells in the first, second and third layers. A dropout rate of 0.5 was used after the final LSTM layer leading to generalization of the model. An optimizer learning rate of 0.001 was used and achieved a validation accuracy of 92.30%.



```
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 embedding (Embedding)       (None, 50, 300)           16278600

 lstm (LSTM)                 (None, 50, 96)            152448

 lstm_1 (LSTM)               (None, 50, 80)            56640

 lstm_2 (LSTM)               (None, 48)                24768

 dropout (Dropout)           (None, 48)                0

 dense (Dense)               (None, 1)                 49

=================================================================
Total params: 16,512,505
Trainable params: 233,905
Non-trainable params: 16,278,600
_________________________________________________________________
```

Figure.30 LSTM using custom-trained word embeddings architecture summary

The classification report is outlined below:

RNN-LSTM using custom Word2vec embeddings trained on the Amazon dataset					
Class	**Accuracy**	**Precision**	**Recall**	**F1-score**	**Support**
	(percentage)				
Negative Class (0)	**92.31**	92.52	92.07	92.29	20972
Positive Class (1)		92.11	92.55	92.33	20972
Macro average		**92.31**	**92.31**	**92.31**	**41944**

Table.11 LSTM using custom-trained Word2vec Word embeddings classification scores

The LSTM model trained using custom word embeddings achieved an accuracy of 92.31 % which outperforms the machine learning models and the ANN and LSTM models trained u sing the pre-trained Google embeddings. This indicates custom tailoring the word embedding s to the same domain of the dataset leads to better results than using pre-trained word embedd ing. The Google word embedding was trained on a news dataset whereas the dataset used in t his research belongs to an e-commerce domain. The positive class has a higher recall than th e positive class implying a larger proportion of positive classes being correctly classified.

The confusion matrix is outlined below:

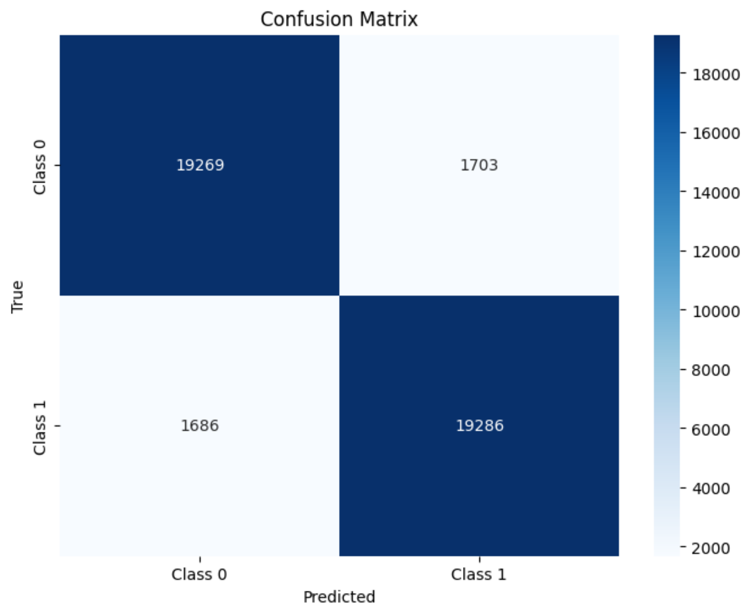


Figure 30 Confusion matrix of LSTM model using custom-trained word embeddings

The positive class has a higher number of correctly classified samples (true positives) than the negative class (19286 samples) however it has a higher number of false positives than the negative class.

	Classifier	Accuracy	Precision	Recall	F1-score
		Macro-average (percentage)			
Machine Learning	Multinomial Naïve Bayes	90.13	90.14	90.13	90.13
	Logistic Regression	91.62	91.62	91.62	91.62
	Support Vector Classifier	91.82	91.82	91.82	91.82
Deep Learning	ANN using Google Word2vec word embeddings	85.26	85.35	85.26	85.25
	LSTM using Google Word2vec word embeddings	91.92	91.92	91.92	91.92
	ANN using Custom trained Word2vec word embeddings	87.98	87.98	87.98	87.98
	LSTM using Custom trained Word2vec word embeddings	92.31	92.31	92.31	92.31

Table.12 Final classification evaluation scores of all the classifiers

Conclusion

The comprehensive analysis of customer feedback is of prime interest to e-commerce companies as it leads to an improved understanding of product regularities and customer opinions and interests. A comparative study was conducted to compare the performance of machine learning and deep learning classifiers to classify sentiments of the Amazon Muiscal instruments review dataset. Text preprocessing techniques such as noise removal, stopword removal, word lemmatisation and part-of-speech tagging were implemented. Setting the N-gram range to 3, and the maximum and minimum document frequency thresholds to 60% and 10 for the Tf-Idf vectorizer helped achieve the best results. Among the deep learning models, the LSTM model trained with the custom word embedding learned using the Word2vec algorithm on the Amazon musical reviews dataset had the best performance with an accuracy, precision, recall and F1-score of 92.31%. It outperformed all the other classifiers including the machine learning classifiers. It was followed by the LSTM model trained using the pre-trained Google Word2vec word embeddings with an accuracy, precision, recall and F1 score of 91.92%. Among the machine learning classifiers, the Support Vector Classifier trained using Tf-idf vectorisation achieved the greatest accuracy, precision, recall and F1 score of 91.82%, followed by the logistic regression classifier (trained using Tf-idf vectorisation) with an accuracy, precision, recall and F1-score of 91.62%. The Multinomial Naïve Bayes classifier achieved a decent performance but had the lowest performance score among the three machine learning classifiers. The ANN model trained using the custom Word2vec word embeddings outperformed the ANN model trained using the Google Word2vec word embeddings with accuracy, precision, recall and F1 score of 87.98%. The ANN model trained using the Google word embeddings had the lowest performance with accuracy, precision, recall and F1-score of 85.26%, 85.35%, 85.26% and 85.25%.

Limitations:

1. As the Amazon reviews dataset was large in size in terms of the number of reviews (904,765 reviews) the majority had to be undersampled to reduce computational load which led to a loss of information for the majority class. The availability of computational resources and time would allow the sentiment classification of the whole dataset to obtain an accurate model.
2. The linear support vector classifier is the simplest form of support vector classifier. Improved classification performance can be obtained by using the Gaussian, polynomial and rbf kernels which require more powerful computational resources and time.
3. The vectorisation of raw text by techniques such as Tf-Idf vectorization leads to a sparse feature space of high dimensionality. Reducing the dimensionality of features by feature selection techniques such as Principle component analysis and Singular Value Decomposition leads to improved performance capacity of the classifier. As the dataset was large in size, SVD could not be performed due to system memory overload.
4. The tokenisation technique used to convert the input text reviews into dictionary-mapped integer tokens for the deep learning models considered only the top 5000 frequent words in the corpus. As to the literature, highly frequent words add to the model capturing noise. Text tokenisation by building a vocabulary based on tf-idf frequencies would have been more efficient.
5. The number of dimensions in the Google word2vec word embeddings is high (300) and leads to the overfitting of the deep learning models. It would have been efficient to employ improved word embeddings such as Glove and FastText.

Future Work:

More advanced data preprocessing techniques such as named entity recognition and word sense disambiguation can be employed to extract representations of various entities and to extract the correct meaning of words given the ambiguous nature of meaning that a word can portray. Topic modelling techniques such as Latent semantic analysis can be used to extract the various topics embedded in the documents for improved classification performance. Transformers and attention deep learning architectures could be employed to get a better contextual understanding of the documents and achieve a higher classification accuracy.

Bibliography

Adikari, A. *et al.* (2021) 'Value co-creation for open innovation: An evidence-based study of the data-driven paradigm of social media using machine learning.', *International Journal of Information Management Data Insights*, 1(2), p. 100022. Available at: https://doi.org/10.1016/j.jjimei.2021.100022.

Arshad, W. *et al.* (2021) 'Multi-Class Text Classification: Model Comparison and Selection', in *2021 International Conference on Electrical, Communication, and Computer Engineering (ICECCE)*, pp. 1–5. Available at: https://doi.org/10.1109/ICECCE52056.2021.9514108.

Brownlee, J. (2020) 'Random Oversampling and Undersampling for Imbalanced Classification', *MachineLearningMastery.com*, 14 January. Available at: https://machinelearningmastery.com/random-oversampling-and-undersampling-for-imbalanced-classification/ (Accessed: 22 August 2023).

Cambria, E. *et al.* (2017) 'Sentiment Analysis Is a Big Suitcase', *IEEE Intelligent Systems*, 32(6), pp. 74–80. Available at: https://doi.org/10.1109/MIS.2017.4531228.

Dey, S. *et al.* (2020) 'A Comparative Study of Support Vector Machine and Naive Bayes Classifier for Sentiment Analysis on Amazon Product Reviews', in *2020 International Conference on Contemporary Computing and Applications (IC3A)*, pp. 217–220. Available at: https://doi.org/10.1109/IC3A48958.2020.233300.

Geron, A. (2019) *Hands-on Machine Learning with Scikit-Learn, Keras, and TensorFlow: Concepts, Tools, and Techniques to Build Intelligent Systems*. O'REILLY.

Hochreiter, S. and Schmidhuber, J. (1997) 'Long Short-Term Memory', *Neural Computation*, 9(8), pp. 1735–1780. Available at: https://doi.org/10.1162/neco.1997.9.8.1735.

Jiang, H., Hu, C. and Jiang, F. (2022) 'Text Sentiment Analysis of Movie Reviews Based on Word2Vec-LSTM', in *2022 14th International Conference on Advanced Computational Intelligence (ICACI)*, pp. 129–134. Available at: https://doi.org/10.1109/ICACI55529.2022.9837505.

Jianqiang, Z. and Xiaolin, G. (2017) 'Comparison Research on Text Pre-processing Methods on Twitter Sentiment Analysis', *IEEE Access*, 5, pp. 2870–2879. Available at: https://doi.org/10.1109/ACCESS.2017.2672677.

Khurana, D. *et al.* (2023) 'Natural language processing: state of the art, current trends and challenges', *Multimedia Tools and Applications*, 82(3), pp. 3713–3744. Available at: https://doi.org/10.1007/s11042-022-13428-4.

Mabrouk, A., Redondo, R.P.D. and Kayed, M. (2020) 'Deep Learning-Based Sentiment Classification: A Comparative Survey', *IEEE Access*, 8, pp. 85616–85638. Available at: https://doi.org/10.1109/ACCESS.2020.2992013.

Manalu, B.U., Tulus and Efendi, S. (2020) 'Deep Learning Performance In Sentiment Analysis', in *2020 4rd International Conference on Electrical, Telecommunication and Computer Engineering (ELTICOM)*, pp. 97–102. Available at: https://doi.org/10.1109/ELTICOM50775.2020.9230488.

Medhat, W., Hassan, A. and Korashy, H. (2014) ‘Sentiment analysis algorithms and applications: A survey’, *Ain Shams Engineering Journal*, 5(4), pp. 1093–1113. Available at: https://doi.org/10.1016/j.asej.2014.04.011.

Mikolov, T. *et al.* (2013) ‘Efficient Estimation of Word Representations in Vector Space’.

Narkhede, S. (2021) *Understanding Confusion Matrix*, *Medium*. Available at: https://towardsdatascience.com/understanding-confusion-matrix-a9ad42dcfd62 (Accessed: 26 August 2023).

O’Malley, T. *et al.* (2019) ‘KerasTuner’. Available at: https://github.com/keras-team/keras-tuner.

Raschka, S. (2017) ‘Naive Bayes and Text Classification I - Introduction and Theory’. arXiv. Available at: http://arxiv.org/abs/1410.5329 (Accessed: 26 August 2023).

Sebastian, R. and Vahid, M. (2019) *Python Machine Learning : Machine Learning and Deep Learning with Python, Scikit-learn, and TensorFlow 2, 3rd Edition.* Packt Publishing. Available at: https://search.ebscohost.com/login.aspx?direct=true&AuthType=ip,shib,cookie,url&db=nlebk&AN=2329991&site=bsi-live.

Shearer, C. (2000) ‘The CRISP-DM model: the new blueprint for data mining’, *Journal of data warehousing*, 5(4), pp. 13–22.

Singh, K.N. *et al.* (2022) ‘A novel approach for dimension reduction using word embedding: An enhanced text classification approach’, *International Journal of Information Management Data Insights*, 2(1), p. 100061. Available at: https://doi.org/10.1016/j.jjimei.2022.100061.

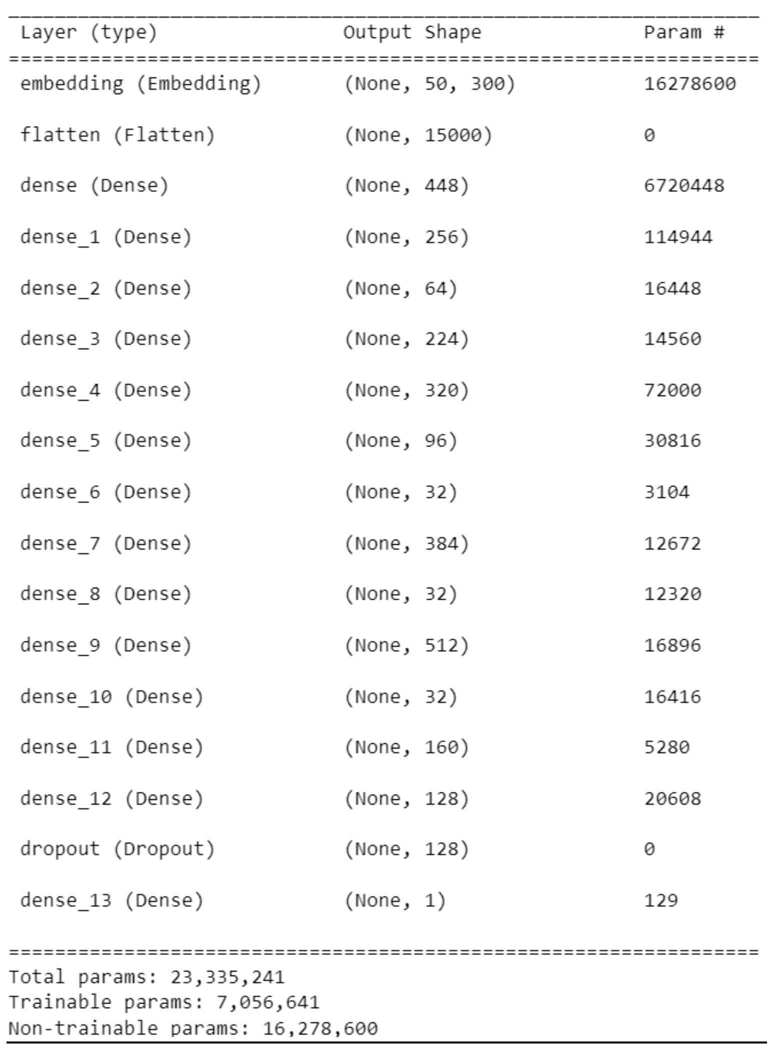
Srivastava, N. *et al.* (2014) ‘Dropout: A Simple Way to Prevent Neural Networks from Overfitting’, *Journal of Machine Learning Research*, 15(56), pp. 1929–1958.

Sunitha, P.B., Joseph, S. and Akhil, P.V. (2019) ‘A Study on the Performance of Supervised Algorithms for Classification in Sentiment Analysis’, in *TENCON 2019 - 2019 IEEE Region 10 Conference (TENCON)*, pp. 1351–1356. Available at: https://doi.org/10.1109/TENCON.2019.8929530.

Vieira, S., Pinaya, W. and Mechelli, A. (2017) ‘Using deep learning to investigate the neuroimaging correlates of psychiatric and neurological disorders: Methods and applications’, *Neuroscience & Biobehavioral Reviews*, 74. Available at: https://doi.org/10.1016/j.neubiorev.2017.01.002.

Yacim, J. and Boshoff, D. (2018) ‘Impact of Artificial Neural Networks Training Algorithms on Accurate Prediction of Property Values’, *Journal of Real Estate Research*, 40, pp. 375–418. Available at: https://doi.org/10.1080/10835547.2018.12091505.

Appendices



```
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 embedding (Embedding)       (None, 50, 300)           16278600

 flatten (Flatten)           (None, 15000)             0

 dense (Dense)               (None, 448)               6720448

 dense_1 (Dense)             (None, 256)               114944

 dense_2 (Dense)             (None, 64)                16448

 dense_3 (Dense)             (None, 224)               14560

 dense_4 (Dense)             (None, 320)               72000

 dense_5 (Dense)             (None, 96)                30816

 dense_6 (Dense)             (None, 32)                3104

 dense_7 (Dense)             (None, 384)               12672

 dense_8 (Dense)             (None, 32)                12320

 dense_9 (Dense)             (None, 512)               16896

 dense_10 (Dense)            (None, 32)                16416

 dense_11 (Dense)            (None, 160)               5280

 dense_12 (Dense)            (None, 128)               20608

 dropout (Dropout)           (None, 128)               0

 dense_13 (Dense)            (None, 1)                 129

=================================================================
Total params: 23,335,241
Trainable params: 7,056,641
Non-trainable params: 16,278,600
_________________________________________________________________
```

Figure 31 Model architecture summary of ANN trained using custom-trained word embeddings

The ARP_Artefact_Aneesh_Kalita_10617414.ipynb file contains the code for the project.

The readme file within the artefact folder has the instructions provided for successfully running the code.

All the files provided in the artefact folder have to be loaded into the current working directory before running the code.

The original raw dataset used for the project is the amazon_reviews_us_Musical_Instruments_v1_00.tsv file and can be loaded within the code.

The necessary libraries and packages can be downloaded using the code provided in the artefact file.

The balanced pre-processed dataframe has been saved for use and is the Balanced_Preprocessed_Amazon.csv file. It can be loaded within the code to train the machine-learning models.

For training the deep learning models the pre-trained Google word2vec word embedding (1.6 gb size) can be downloaded using the code in the artefact.

The custom word embedding trained using the Word2vec algorithm on the Amazon corpus has been saved and can be found inside the Custom_Word2vec_word_embedding folder. The files within the folder must be loaded into the current working directory before running the code and can be loaded using the artefact code.

The word embeddings extracted from the custom-trained word embeddings for the training dataset has been saved as an embedding matrix and is the embedding_matrix_amazon.npy numpy file. It can be loaded from within the code file.

After extracting the word embeddings from the Google pre-trained word embeddings for our training dataset the resulting embedding matrix has been saved as a numpy file - embedding_matrix__google_word2vec.npy for future use. It can be loaded from the code.

The Keras tuner must be installed from within the code and the Neural network model tuning can be implemented thereafter.