Data Science Project

Technical Report

GROUP ID: APOPHIS

Research Question:

How does the performance of neural networks compare to logistic regression in sentiment analysis on combined Amazon review and Twitter data?

Robbert Bernecker (2823756)

Pooya Mers (2734368)

Duy Phan (2802376)

Erin Breur (2754718)

June 26, 2025

E_EDS2_DSPT

Bachelor Econometrics and Data Science

Contents

Abstract

We investigate the performance of Neural Network(NN) models compared to Logistic regression(LR) in sentiment classification on the combined dataset of Amazon reviews and Twitter tweets. Our classification task is to map each document to ternary labels(positive, neutral, and negative). The datasets contain 205000 reviews and around 50000 tweets. A preprocessing pipeline is used to remove noisy and uninformative data from the raw input. Two feature engineering techniques were applied to transform preprocessed text into numeric vectors: TF-IDF for LR and Word Embeddings for the NN models. We consider two neural network models: Bidirectional Long Short-Term (Bi-LSTM) and Convolutional Neural Networks (CNN). We evaluate the performance by using accuracy and averages of the F1-scores as our metrics. Our analysis results show that while LR performs best for predicting the neutral sentiment in our dataset, Bi-LSTM yields the highest overall accuracy at 72.8%, slightly followed by CNN at 72.2%, and LR at 71.4%.

Keywords: *Sentiment classification, TF-IDF, Word-Embeddings, Logistic Regression, CNN, LSTM, Amazon reviews, Twitter*

1 Introduction

Sentiment classification is the process of classifying a block of text as positive, negative, or neutral. The goal is to capture the opinion of individuals in a way that enables businesses to expand and allows organizations to quantitatively analyze the tone of the user based on text. By monitoring consumer sentiment, they can make marketing and operations decisions that lead to customer satisfaction and increased brand reputation. With the growth of online platforms in terms of users, it is no longer plausible for linguistic experts to evaluate the sentiment of texts one by one. Furthermore, expert-created linguistic baselines are not able to perform consistently in all types of corpora, due to a lack of adaptability to context, limited vocabulary, and inability to capture relationships outside the pre-set rules. Automatic classification by data-driven models does not face these issues, as they can learn directly from the data.

In this report, we investigate the research question: "How does the performance of neural networks compare to logistic regression in sentiment analysis on combined Amazon review and Twitter data?" We explore Supervised Machine Learning (ML) models that perform ternary sentiment classification. The datasets used are the Amazon Review Dataset Ni et al. (2019) and the Twitter dataset Nakov et al. (2016). The literature review in section 2 provides insight into existing literature and approaches. We introduce the mathematical framework for the classification task in section 3. In section 4, we begin

with preprocessing to clean the data and select important features. Thereafter, we proceed with encoding in section 5 to convert the text into interpretable numerical features (numeric vectors) for the ML classifiers. Then, we introduce, discuss, and train three supervised models: Logistic Regression (LR), Convolutional Neural Networks (CNN), and Long Short-Term Memory (LSTM) in section 6. Lastly, we evaluate the results of the three models and discuss which one performs best for different sentiment classification purposes in section 7.

2 Literature Review

2.1 Preprocessing

The data preprocessing step transforms raw data input into a structured format, which is used in training and evaluating the classification model. This stage involves several chosen steps, one of which is Stopword Removal, which encompasses a process of removing redundant words that reduces dimensionality (Garg and Sharma, 2022). The study of Pradha et al. (2019), regarding effective and proper text data preprocessing techniques for Sentiment Analysis, mentions that words such as ”a”, ”an”, and ”the” do not provide any meaning to the text and can be removed without loss of accuracy. For instance, Agarwal et al. (2011) performed Sentiment Analysis of Twitter data and made use of a dictionary of stop words to identify and remove up to 38.3% of all total words. They also introduced other approaches, including the labeling of emoticons with an appurtenant sentiment. An example would be ”:)” being labeled ’positive’. Another important preprocessing step involves translating acronyms and internet slang into their full English forms to improve semantic understanding. For instance, the acronym “lol” is typically expanded to “laughing out loud”, providing clearer sentiment cues for classification models (Baldwin et al., 2013).

Lemmatization and Stemming are two preprocessing techniques that seek to further shrink dimensionality. Lemmatization is used to map each word to its base or dictionary form, known as its ’lemma’. This reduces dimensionality, since words that share the same lemma are afterwards treated as the same word. Unlike Stemming, which only removes suffixes from words, Lemmatization uses vocabulary and grammar rules to return a meaningful root word. However, according to Pradha et al. (2019) and Saraswati et al. (2025), stemming does have a far better computing time than Lemmatization. Their research also mentions that a combination of preprocessing techniques involving Stemming, Tokenization, and removal of special characters and punctuation performed best in their settings. However, the multilingual sentiment analysis by Garg and Sharma (2022) showed that

the use of Stemming can generally degrade the performance of the text classification model when dealing with large and unbalanced datasets. Their analysis, moreover, corroborates the earlier statement on the expensiveness of Lemmatization, as it concludes that while Lemmatization imposes positive impacts on the classification performance on small datasets, it has an undesirable computational expense. Garg and Sharma emphasize a more careful consideration of the necessity of Lemmatization and Stemming techniques in sentiment analysis, and explain the tradeoff that takes place when choosing the best combination of techniques to improve accuracy, reduce dimension, under the constraints of computation. Lastly, Shi et al. (2016) indicated that the Stratified Sampling method achieves a higher accuracy than simple random sampling due to convenient organizational management.

2.2 Encoding

For models to be able to process texts, such as reviews and tweets, they must be converted into a numerical form. This section describes key NLP techniques that are used for this end: Tokenization/Vectorization, TF–IDF weighting, word embeddings, and dimensionality reduction. Tokenization breaks down text into smaller units called tokens (e.g., words or phrases). During Vectorization, each unique token is mapped to a number, and the input text documents get converted into vectors containing numerical values. The simplest method for Vectorization is the bag-of-words model, which counts token frequencies. While straightforward, this approach is high-dimensional and sparse, leading to computational inefficiency and limited semantic understanding Manning et al. (2008). Afterwards, Term Frequency–Inverse Document Frequency (TF–IDF) improves on raw counts by emphasizing sentiment-rich but rare terms, while downplaying common neutral terms Manning et al. (2008). Word embeddings such as Word2Vec Mikolov et al. (2013b) and GloVe Pennington et al. (2014) represent words as dense vectors in a lower-dimensional space. These vectors beautifully capture semantic relationships: similar words (e.g., great, excellent) are placed close together. Embeddings generalize better than sparse vectors and reduce dimensionality while retaining contextual meaning. High-dimensional vectors can be compressed using methods like PCA and t-SNE. PCA preserves global variance and creates uncorrelated components, whereas t-SNE preserves local structure, making it ideal for visualizing clusters in sentiment data (van der Maaten and Hinton, 2008).

2.3 Classification

A classical example of a regression-based method that is relevant for Sentiment Analysis is logistic regression. The paper by Salloum et al. (2024) explores how logistic regression remains a strong tool for sentiment classification due to its simplicity, interpretability, and efficiency. The model output is a probability estimate for a document belonging to a certain category. Because of its interpretability, it additionally allows us to have insight into how hyperparameter selection behaves and allows us to investigate the estimation process, which is ultimately why we have opted for this method as a benchmark.

The Support Vector Machine (SVM) model finds the optimal hyperplane that separates different categories in a high-dimensional space, maximizing the margin between classes to improve classification accuracy Sebastiani (2002a). It would similarly have sufficed as a benchmark, due to its simple and interpretable nature. However, in comparison, logistic regression provides us with more options for tuning and simulating. While the results obtained by Pang et al. (2002) outperform expert-generated keyword baselines, the accuracies of models are low compared to their application in topic-based categorization. The findings of Pang et al. indicate that while probability-based models may be simple to implement and interpret, they likely will not produce high predictive accuracy.

Cliche (2017a) uses Convolutional Neural Networks (CNNs) and Long Short-Term Memory (LSTM), which is a type of recurrent neural network (RNN), to train a classifier for tweets. The approach employed by Cliche was an ensemble of multiple CNNs and multiple LSTMs to reduce variance and boost accuracy. We take inspiration from this and train both CNN and LSTM models, after which we make an attempt at an ensemble ourselves. CNNs perform well for sentiment analysis tasks because they can capture long-term dependencies in the vocabulary and syntax Abid et al. (2019). While typical RNNs struggle with remembering long-term relationships in the sequence, LSTM can retain these relationships, which leads to increased performance Wankhade et al. (2022).

Studies such as the one by Garg and Sharma mentioned above combine stop-word removal, TF–IDF, and Logistic Regression, whereas others, such as the study by Cliche, pair Lemmatization, Word2Vec embeddings, and neural network models. The paper by Almeida and Xexéo (2023) discusses how word embeddings allow ML models to capture more information in smaller dimensions and understand the relationship between words in a sequence better. Furthermore, Wardana et al. (2024) investigates that Logistic Regression pairs well with TF-IDF because it processes the high-dimensional but interpretable features that TF-IDF encodes in an efficient manner. Finally, we consider the investigations made in Miyajiwala et al. (2022), in regard to stop word removal. Miyajiwala et al. find that Neural Network models are sensitive to removing stop words, and doing so may reduce model performance. We use this guidance and the discussed approach to construct

the steps in preprocessing our data.

3 Classification Task

The objective is to find the best-performing ternary classifier model that can predict the sentiment of a document (a review or a tweet). More formally, let $D = \{d_1, ..., d_n\}$ be the domain of documents and $C = \{+1, 0, -1\}$, where c_i corresponds to positive, neutral, and negative sentiments, respectively. Assume there exists an unknown target function $g : \mathcal{D} \to \mathcal{C}$ that describes the true sentiment for each document. The learning algorithm receives a labeled dataset:

$$\mathcal{S} = \{(d_i, c_i)\}_{i=1}^{n}, \quad d_i \in \mathcal{D},\ c_i \in \mathcal{C}$$

and aims to find a function f that approximates g as closely as possible, not only on $\mathcal{S}$ but also on unseen data (out-of-sample generalization).

In sentiment analysis, documents must often go through preprocessing to discard information that does not improve the performance of the classification model. Additionally, for certain machine learning models, learning f from raw text is infeasible. The raw text inputs must be converted to an interpretable numeric format (Sebastiani, 2002b). Thus, an **encoder** Ψ is used to map documents to a feature space:

$$\Psi : \mathcal{D} \to \mathcal{X} \subseteq \mathbb{R}^K$$

where K is the size of the vocabulary or the dimension of the feature space. Each document d is represented as a vector $\mathbf{x} = \Psi(d)$.

Figure 1 illustrates the approach we take in our analysis. The justification for the choices in each step is explained in detail in their respective sections. First, the preprocessing of the documents in D. Then, feature extraction is performed to convert the text into interpretable features for the classifiers. In the next step, three machine learning models are introduced, justified, and trained. Lastly, the performance of each model is evaluated, which aids the discussion of criteria for selecting the most appropriate model in the specific application on this dataset.

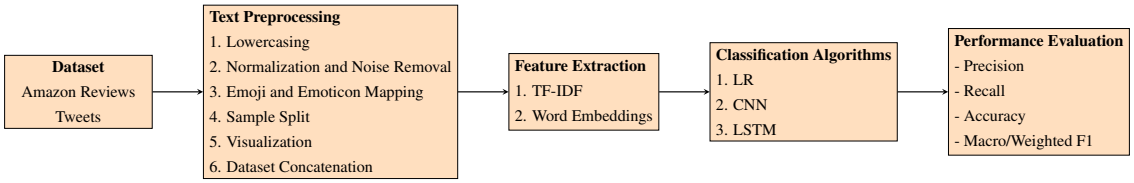


Figure 1: Proposed Approach for Sentiment Classification

3.1 Notation and Mathematical Framework

The following notation will be used throughout the report:

- $\mathcal{D} = \{d_1, \ldots, d_n\}$: Collection of documents.
- $\mathcal{C} = \{+1, 0, -1\}$: Sentiment label set.
- $\mathbf{y} \in \mathcal{C}^n$: Vector of true labels.
- $\mathcal{V} = \{t_1, t_2, \ldots, t_{|\mathcal{V}|}\}$: Vocabulary of unique terms.
- $\Psi : \mathcal{D} \to \mathbb{R}^{|\mathcal{V}|}$: Encoder function mapping documents to feature vectors.
- $\mathbf{X} \in \mathbb{R}^{n \times |\mathcal{V}|}$: Document-term feature matrix, where each row $\mathbf{x}_i = \Psi(d_i)$ is a feature representation of document d_i.
- g: True (unknown) target function describing the true sentiment for each document
- $h : \mathbb{R}^{|\mathcal{V}|} \to \mathcal{C}$: Classifier function.
- $f = h \circ \Psi$: End-to-end classification function that estimates g.
- Tr: Training set
- Va: Validation set
- Te: Test set

4 Dataset & Preprocessing Steps

4.1 Dataset description

The dataset is comprised of two parts: 205,000 Amazon reviews and 50,083 Twitter tweets. Both reviews and tweets are pre-classified (labeled) in three categories as being ”positive”, negative”, or ”neutral”. The original labels for Amazon reviews are 1-5

ratings, which do not directly convey information about sentiment. Following the methodology in Liu (2012), the ratings are mapped to three-way categories [-1, 0, 1] to align with the sentiment classification problem.

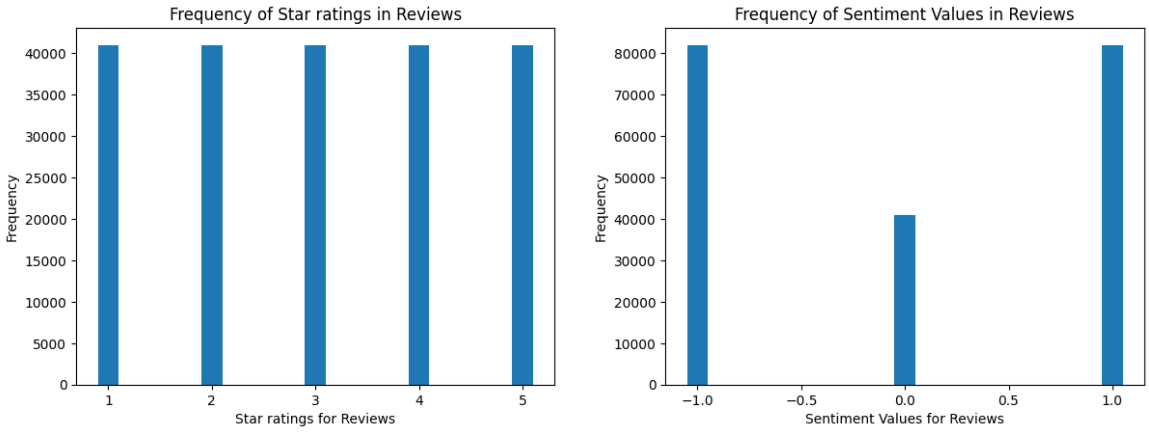
- $\phi : \{1, 2, 3, 4, 5\} \rightarrow \mathcal{C}$: Star rating to sentiment label mapping, defined as:

$$\phi(s) = \begin{cases} -1, & \text{if } s \in \{1, 2\} \\ 0, & \text{if } s = 3 \\ +1, & \text{if } s \in \{4, 5\} \end{cases}$$

Here, ϕ maps star ratings to sentiment labels in $\mathcal{C}$, where -1 indicates negative, 0 indicates neutral, and $+1$ indicates positive sentiment.

This method is simple, but it has a major drawback that the star ratings may not exactly correspond to sentiment. This drawback restricts the models from achieving perfect predictive accuracy because the review data goes through a trivial classification(mapping) process that is not error-free. Furthermore, we can see in Figure 2 that the neutral sentiment class now has half the data compared to the other classes. This means that the neutral sentiment may be underrepresented in the classifiers, leading to fewer correct predictions.

Figure 3 shows the frequency of the three sentiment classes in the tweets data. The neutral sentiment has the most documents, followed closely by the positive sentiment, while the negative sentiment has many fewer documents. As a result, the negative class may be underrepresented in the classifiers, harming model performance.



(a) Original Data

(b) Data After Mapping To Ternary

Figure 2: Frequency of data in Reviews: before and after mapping to ternary values.

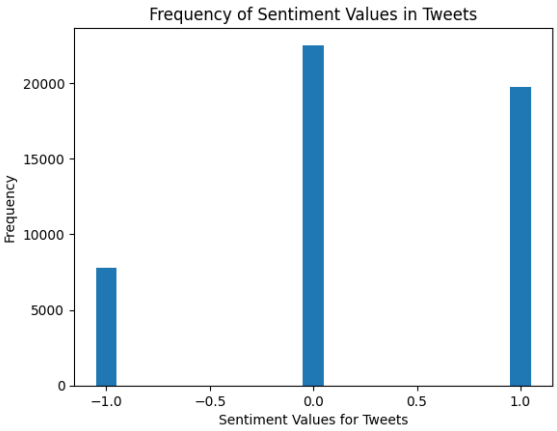


Figure 3: Frequency of data in Tweets.

4.2 Preprocessing and feature extraction

Visual inspection of the data shows that many documents have to go through text cleaning and feature extraction. Figure 4 illustrates the steps that have been taken to prepare the data for encoding:

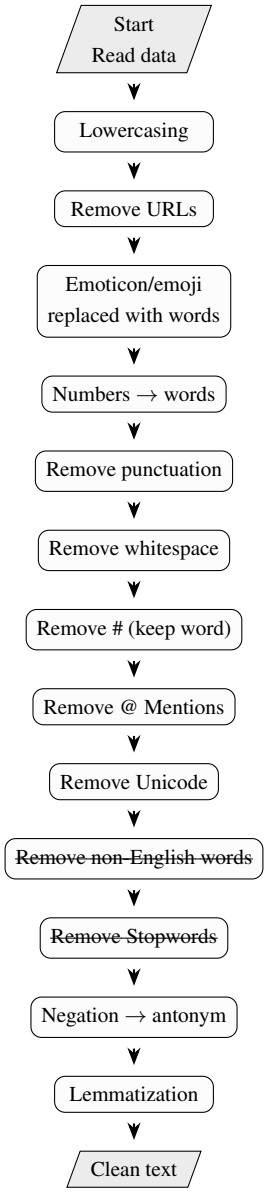


Figure 4: Text Preprocessing Pipeline (Crossed-out steps are considered but not eventually implemented)

1. All text is lower-cased, since capitalized words and expressions do not convey much information, and if untreated, stay differentiable from their lower-case counterparts. According to Pradha et al. (2019), the main purpose of converting text to all lowercase is that words such as ”Hello” and ”hello” are not treated as different words, as they are the same word and have the same meaning. This preprocessing technique also helps in reducing the word size that the dictionary needs to hold. Converting all the letters to lowercase, combined with other preprocessing methods, can eliminate redundant words in the dataset to ensure the preprocessed text is suitable for the classification of Amazon Book Reviews (Srujan et al., 2018). In addition, several researchers have highlighted that the overall combination of lowering cases

with other preprocessing approaches involving stop word removal, lemmatization or stemming, and part-of-speech has shown a great effect on the accuracy of the machine learning models for sentiment classification (Garg and Sharma (2022)).

2. One of the common preprocessing techniques is URL removal from the tweet, including HTTP, HTTPS, and the URL for a picture in the tweet (Pradha et al. (2019), and Saraswati et al. (2025)). It is much easier for machine learning models to process text after the removal of punctuation such as URLs, special characters, and HTML tags, because these add noise to the text without conveying any semantic information. To summarize, this stage of data preprocessing can ensure the analysis solely focuses on the semantic meaning of the text (Saraswati et al. (2025)). The research of Palomino and Aider (2022) highlighted the importance of the complete removal of URLs from Twitter data, as it is impossible to estimate the sentiment of an online source, web page, picture, or other things associated with URLs.

3. Emoticons and emojis are replaced with their associated semantic meaning. The study of sentiment analysis on Twitter by Agarwal et al. (2011) introduces a new approach for preprocessing Twitter data by using an emoticon dictionary to assign all the moods of the users indicated by punctuation and letters, and the corresponding emotional states. For instance, ":=(" is labeled as negative while ":)" is labeled as positive. Furthermore, Liu et al. (2021) argues that the effectiveness of replacing emojis with their associated meanings can improve the performance of sentiment analysis algorithms. There is also a similar result regarding the study on the sentiment analysis of Tweets including emoji data, by LeCompte and Chen (2017), that there is an increase in the classification accuracy for Naive Bayes and Support Vector Machine when incorporating emoji for Twitter data classification.

4. Numbers are replaced by words, despite the accounts of Nandwani and Verma (2021) that entirely removing numbers can enhance the model performance accuracy. Mehanna and Mahmuddin (2021) argues that keeping numbers and replacing them with words in some specific cases can convey the sentiment of a sentence. For instance, in the sentence "I watched this movie 3 times", the number 3 indicates that the film is very interesting and appealing. If that number is removed, the sentence becomes more neutral in sentiment already. The same could be illustrated for sentences containing numbers, for which the sentence has a negative sentiment (Mehanna and Mahmuddin, 2021).

5. Removal of punctuation or any non-alphanumeric words from the original text is also considered in the data cleaning phase. Non-textual elements such as $, ":", and

”,” are considered noise that harms the accuracy of machine learning models and precise sentiments, according to the studies on text data preprocessing for sentiment analysis by Pradha et al. (2019) and Garg and Sharma (2022). However, the thorough evaluation of effective text preprocessing in SA of Palomino and Aider (2022) argues that the punctuation should be eliminated after the step of converting the emoticons into words, because emoticons such as ”:)”, ”:D”, and ”;)” contain punctuation and would otherwise be removed by the punctuation removal step, despite carrying relevant sentiment. The approach of removing punctuation only after handling the emoticons in text preprocessing also increases the speed and improves the performance of the sentiment analysis (Kim, 2018). Furthermore, question marks (”?”) and exclamation marks (”!”) are not removed, as Devitt and Ahmad (2007) points out that they may contribute to the meaning of the text.

6. The removal of redundant whitespace is necessary for text preprocessing to ensure a more effective computational process. This is because whitespace does not contribute any meaning to the text (Pradha et al., 2019). The classification of Amazon Book Reviews considers this, for example, by removing ”/t” (or tab) in order to reduce noise, seeing as they do not indicate any sentiment (Srujan et al., 2018). Identification and removal of unnecessary spaces between characters could help prevent inappropriate token splits. For instance, ”New” and ”York” can be treated as two separate tokens that can confuse the classifier rather than keeping ”New York” as a single term (Palomino and Aider, 2022).

7. Hashtags are usually used to mark some particular trending topics that are primarily done to increase the visibility of users’ tweets. A Removal of hashtag characters, except for the keywords and phrases included in the hashtags, is considered. For example, the symbol # is extracted from # happy but not the word happy since it can reflect the mood of users, and this can ensure better sentiment classification accuracy as stated by Palomino and Aider (2022). However, Naseem et al. (2021) argue that although hashtag symbols may express sentiment, and give extra information regarding some particular topic, they are only useful for human beings and do not provide any information for machine classification such that hashtags can be considered as noise in raw data, and they need to be handled properly by excluding the hashtag symbol ”#” from the phrases and keywords to improve the classifier to understand the content of tweets easily. An example of this is # happylife is split into two separate words, ”happy” and ”life”.

8. User mentions are omitted from tweets as they offer no insight into the text sentiment (Pradha et al., 2019). The removal of user mentions ”@username” is cru-

cial in text preprocessing as they are considered as noisy and redundant data and lack sentiment-related values. The result obtained from the study of Naseem et al. (2021) demonstrated that the combined preprocessing techniques involving the removal of URLs, user mentions, and hashtag symbols increase the performance of SVM, CNN, and other classifiers.

9. Unicode sequences are the unwanted strings or leftovers contributing more noise to the data. The experimental results also illustrated that the removal of all Unicode strings, URLs, user mentions, and hashtag symbols # is conducted in the first step, which achieves the best-performing preprocessing sequences (Naseem et al., 2021). Symeonidis et al. (2018) emphasizes that Unicode strings such as "\u002c" and "\x06" are leftovers from data collection, and cleaning up these strange characters acts as a baseline in the context of text preprocessing.

10. Non-English words and characters are abundant in both datasets, although the tweets suffer from this issue more, possibly due to the more informal nature of the corpus platform. As suggested by Symeonidis et al. (2018), identifying and removing these words may increase model performance. However, by trial and error, we saw that performing this step significantly lowered the performance of all models. This process removes all words that are not in a given English dictionary. Therefore, there are likely slang, misspellings, and abbreviations in the documents that get removed unintentionally in this process, although they contain information pertinent to sentiment classification. As a result, this step is not performed in the final pipeline.

11. Removing stop words is a common practice in preprocessing text classification tasks to keep the relevant text for sentimental purposes, but certain negation words are kept to use inside bi-grams. Stop words are mostly prepositions, articles, pronouns, and conjunctions like **the**, **a**, **am**, **are**, **on**, and **at**, **and** are highly repeated in sentences; however, these words generally do not contribute much to the classification task and semantic meaning, and their removal is considered (Naseem et al., 2021; Srujan et al., 2018). However, negations such as **not**, **wouldn't**, **can't** expressing the opposite meaning of words and phrases are kept and handled properly by replacing them with antonyms where the word "not" is searched in the sentence and check whether there is an antonym of the next word (Symeonidis et al., 2018; Palomino and Aider, 2022). For instance, the phrase **not good** will be substituted with the word **bad**. The transformation of negated words for tweets with the prefix "not" can improve the result of sentiment classification (Naseem et al., 2021).

12. Words are then lemmatized to unite different words from the same base into their root forms. Lemmatization is a preprocessing technique used to convert different

forms of words to the base word, and it shows significant improvement in the performance of CNN and LSTM classifiers for sentiment analysis in the case of Twitter datasets (Naseem et al., 2021).

4.3 Corpora Visualization and Analysis

After the data is cleaned, different corpus statistics can be visualized.

Uni-gram frequency: In figure 5, the frequency of uni-grams in the reviews dataset is plotted against those in the tweets dataset. There is an interesting distinction between the two datasets. The uni-grams in the reviews tend to be adjectives such as ”great” or verbs such as ”use” that relate to quality or functionality, where as the uni-grams in the tweets are nouns that are about time, such as ”tomorrow”, of verbs such as ”go” that convey information about events and activity.

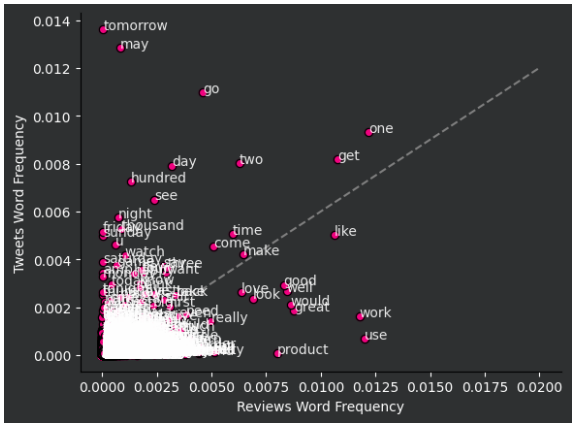
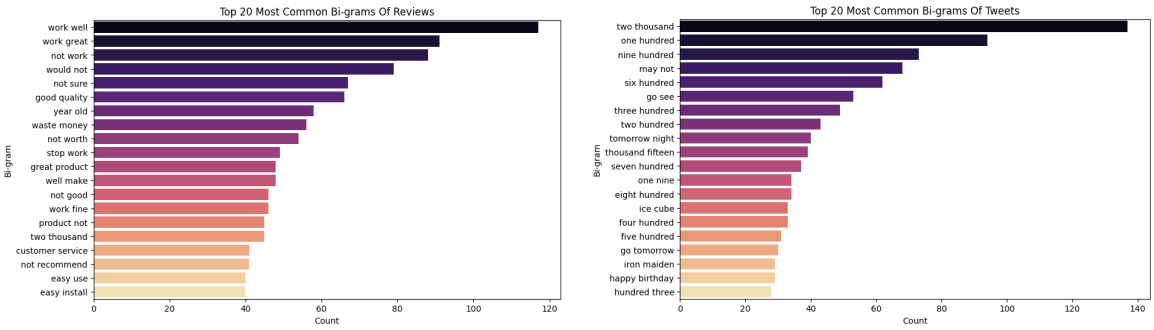


Figure 5: The frequency of words in tweets plotted against tweets.

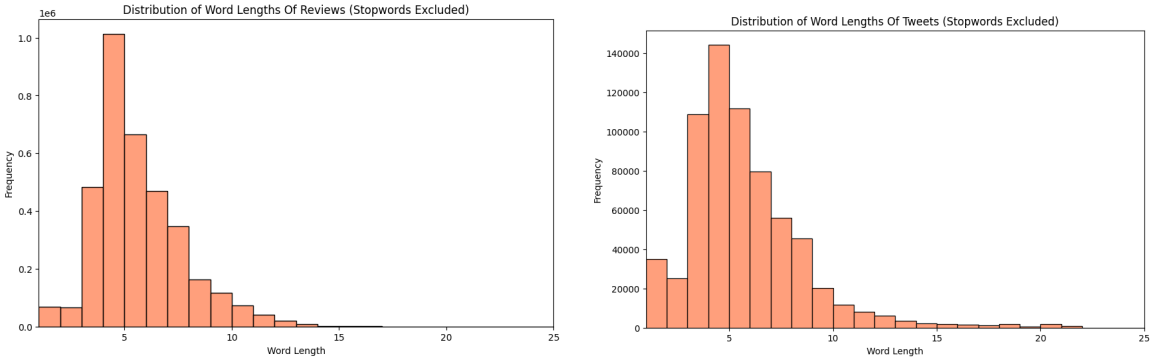
Bi-gram frequency: The bi-grams provide additional information about the corpora. The most common bi-grams from the reviews are those that discuss functionality or quality, such as ”work well”, or share an opinion, such as ”not worth” or ”waste money”. On the other hand, the most common bigrams from tweets are numbers, such as ”two thousand”. The reason for this may be that numbers are more common in dialogue where events or people are discussed, in comparison to reviews, where product feedback is given. Additionally, there are also bigrams concerning decisions found in tweets, such as ”gon na” or ”tomorrow night”.



(a) Reviews (b) Tweets

Figure 6: The most common bigrams of reviews and tweets.

Word lengths: As visible in figure 7, the length of uni-grams is similar in both datasets, and most words are between three and eight letters long in both. However, there are more short (between 1 and 3 characters) unigrams for tweets, which may be due to abbreviations or colloquialization of words in the more informal social media platform. Additionally, the distribution of word lengths for tweets is much wider, which may be due to links, numbers, or spam text that went undetected in the preprocessing step. Table 1 also shows the statistics for the token lengths across both datasets. While the token lengths of the two datasets both have a median of 5 characters, tweets have a slightly higher mean (5.24 compared to 5.15), and also have a longer maximum length (56 compared to 42).



(a) Reviews (b) Tweets

Figure 7: Word lengths distributions of reviews and tweets.

Dataset	Mean	Median	Max	Min
Reviews	5.15	5.0	42	1
Tweets	5.24	5.0	56	1

Table 1: Token Length Statistics for Reviews and Tweets (Stopwords Excluded)

4.4 Dataset Concatenation

Two approaches are considered for sentiment classification with the two datasets: the first is to concatenate (combine) the two datasets into a large dataset and train a model on the combined dataset, and the second is to train a model on each dataset separately. Since the task objective is to perform sentiment prediction on unseen documents, the source of the unseen documents is not available. Therefore, the second approach requires that the documents first be 'pre-classified' into either tweets or reviews, and the pre-classified documents can then go through the sentiment classification model appropriate for the assigned dataset. This method comes with three major drawbacks and implications:

1. The pre-classification process may mistakenly classify the origin of the document, which would likely lower the accuracy of the sentiment classification process that comes after. This drawback can be alleviated if there were a third 'common' dataset, where documents with a pre-defined *low* certainty of belonging to either tweets or reviews would get assigned. However, this alternative approach suffers from the following drawback even more.

2. There would be fewer data points available for each model corresponding to the two datasets, which may hinder overall model performance.

3. Aside from the size of the data, the imbalance in the size of the datasets for reviews and tweets is a major problem. Since there are four times as many review documents as there are tweets, the pre-classification process would likely be biased towards over-classifying documents as reviews.

To put these three implications to test, a Domain-Adversarial Neural Network was trained (see Ganin et al. (2016)), although the details of this approach are not in the scope of this paper. Using this approach, the model learns the common features between the two domains and discards domain-specific information, aiming to create a better predictor for the entire corpus. Within the DANN architecture, a Convolutional Neural Network (CNN) is used as the feature extractor, optimized for sentiment classification and adversarial domain discrimination through a gradient reversal layer. The results (Appendix A.1) confirm our suspicions that the size, imbalance, and the increased complexity result in a worse-performing model, compared (across all macro metrics) to the models to be discussed in section 6.

To conclude, since the size of the data set is not large for sentiment classification, this second approach is avoided, and instead, the first approach(training a model on the combined dataset) is taken: $D = D_{tweets} \cup D_{reviews}$.

4.5 Sample Split

After preprocessing, visualization, and concatenation of the datasets, the dataset D is divided into a training dataset Tr with 70% of the data, a 15% validation set, Va, and a 15% testing set, Te, using stratified sampling. Tr will be used to train the models, Va will be used to tune hyperparameters when applicable, and Te will be used to test the performance of the models. Stratified sampling ensures that the distribution of samples across three categories (-1,0,1) is the same in the train and test sets, maintaining representation of the population and reducing the potential bias in the analysis.

5 Feature Engineering

In order to estimate the unknown target function $g : \mathcal{D} \rightarrow \mathcal{C}$, we used an encoder Ψ to map each document $(d_j \in \mathcal{D})$ to a K-dimensional feature space $\mathbf{x} = \Psi(d) \in \mathbb{R}^K$, a classifer is then applied $h : \mathbb{R}^K \rightarrow \mathcal{C}$ to make prediction for the final model $f(d) = (h \circ \Psi)(d)$.

To use text data in Machine Learning for sentiment analysis, it must first be tokenized, a process that segments the text into smaller units called tokens—typically words, subwords, or punctuation marks—depending on the tokenization strategy used (Sebastiani, 2002b). Thereafter, the tokens must be converted into numerical representations in a process called vectorization, which transforms text data into a format suitable for machine learning algorithms. TF-IDF and Word Embeddings are two common approaches used in sentiment classification.

5.1 Term Frequency-Inverse Document Frequency (TF-IDF)

To train the multinomial logistic regression model, we utilized Term Frequency-Inverse Document Frequency (TF-IDF) to provide our model with informative and numerical inputs. In this subsection, we provide a brief definition of TF-IDF with mathematical notation, details of our engineered features, and regularization, dimensionality reduction techniques, with motivation for our choices.

TF-IDF is a statistical measure used in natural language processing, such as sentiment analysis, to assess the importance of a word in a document relative to a collection of documents or corpus. The structure of TF-IDF involves two components, where the first part is the Term Frequency (TF), measuring how often a word appears in a document. A higher frequency suggests greater importance. If a term appears frequently in a document, it is likely related to the content of the document. Inverse Document Frequency (IDF) is the second component that reduces the weight of common words while increasing the weight of rare words. Therefore, we decided to use TF-IDF for vectorization to balance

common and rare words and highlight the most meaningful terms.

We applied TF-IDF to the preprocessed text. It can be expressed in mathematical terms as follows:

Let $\mathcal{V} = \{t_1, t_2, \ldots, t_{|\mathcal{V}|}\}$ denote the vocabulary of unique terms. For each document $d_j \in \mathcal{D}$, the TF-IDF encoder is:

$$\Psi_{\text{TF-IDF}}(d_j) = \mathbf{x}_j = \left[w(t_1, d_j), w(t_2, d_j), \ldots, w(t_{|\mathcal{V}|}, d_j)\right]$$

and

$$w(t_k, d_j) = \text{tf}(t_k, d_j) \cdot \log\left(\frac{N}{\text{df}(t_k)}\right).$$

Where w indicates the weight value of the term t_k and document d_j, the total number of document in the corpus defines as N, matrix $\mathbf{X} \in \mathbb{R}^{n \times |\mathcal{V}|}$ includes all vectors $\mathbf{x}_j$, and $tf(\cdot)$ and $df(\cdot)$ denote the term frequency representing the number of times and number of the documents in a particular term, respectively (Kamyab et al., 2021).

5.1.1 Feature Construction:

Unigrams and bigrams are used first, where unigrams are individual tokens such as "good", "bad", "happy", and bigrams are pairs of words like "not good", and "I hate", etc, conveying sufficient sentiment. These sequences of single and double tokens from the given text can keep the model simple and avoid highly complex dimensionality (Ahuja et al., 2019). The vocabulary size is limited to 29000 terms through the TF-IDF score. Specifically, we eliminated rare terms appearing in less than 5 documents through the syntax $mindf = 5$ and $maxdf = 0.8$, and discarded very frequent terms appearing beyond 80% of documents. This helps reduce unnecessary terms, thus speeding up the training process. Then we ran 5-fold cross-validation to try different values of max features with a range of 20000, 31000, and 1000 to select the highest F1 weighted score as the optimal value to achieve the best model performance. The TF-IDF vectors are normalized by applying cosine normalization, which can be illustrated by the following function:

$$w_{kj} = \frac{\text{w}(t_k, d_j)}{\sqrt{\sum_{s=1}^{|\mathcal{V}|}(\text{w}(t_s, d_j))^2}}$$

This cosine normalization technique ensures that the preprocessed texts of different lengths such as short vs long tweets and reviews can be comparably represented, and avoid the dominance of long text over short ones.

5.1.2 Regularization, and Dimensionality Reduction:

In this subsection, we applied an L2 regularization for the logistic regression model to impose a penalty term on large weights of uninformative features. This method helps models effectively shrink noisy terms and avoid the effect of overfitting (Salehi et al., 2019). In terms of dimensionality reduction, we set thresholds for the hyperparameters within vectorization. In particular, we set a maximum of 29000 words for vocabulary size, and removed the least and the most frequent terms in the document, ensuring a drop in the dimensionality and focusing on key features (Chen et al., 2019).

5.2 Word Embeddings

To train Convolutional Neural Network (CNN) and Long Short-Term Memory (LSTM) models, we used word embeddings. Specifically, a trained Word2Vec model is applied to train the tokens for LSTM, and a Keras embedding layer for CNN. A short definition of word embeddings - a feature extraction technique, our construction features, reasons for our choices, how we implemented dimensionality reduction, and vocabulary limitation are used, would be all discussed in this subsection.

Word embeddings are numerical representations of words in a lower-dimensional space that capture semantic and syntactic information. They map words into a continuous vector space to allow words with similar meanings or contexts to have the same representation. Thus, similarity can be evaluated based on similar vector representations, and this advantage helps the machine learning models capture more information in fewer dimensions and understand the relationship between words better.

5.2.1 Constructed Features:

A Word2Vec model for LSTM is trained by using only the training tokens. We also created an embedding matrix to store each word, which is then represented by a 50-dimensional vector. From the training data, we extracted the 20,000 most frequent words from the total vocabulary size. Then we take an average of the token vectors for each document, which can be expressed by the following equation for the word embedding encoder:

$$\Psi_{\text{emb}}(d_j) = \frac{1}{|d_j|} \sum_{t \in d_j} \overrightarrow{\text{embed}(t)}$$

Where $\text{embed}(t) \in \mathbb{R}^d$ is an embedding function ,d_j represents each document and $|d_j|$ stands for the number of tokens in each document d_j. This step produces a feature vector defined by a fixed length to represent the whole text (Stalidis et al. (2015)). On

the other hand, we applied the CNN model with a Keras embedding layer that receives an input dimension of 20000 features and yields output with reduced dimension to 100; this embedding layer will be learned during the training process (Kim, 2014).

There are several reasons why we decided to apply these approaches. One of them is to ensure a lower dimensionality for the feature size since word embeddings are densely distributed, fixed-length word vectors. The second reason is that all words that have similar semantic meaning are collected together in a continuous vector space. Taking an average of each document can provide a single compact vector to represent the meaning of each text. These advantages help machine learning models capture more information in smaller dimensions and understand the relationship between words in a sequence better, thus making the sentiment modeling more efficient and accurate (Almeida and Xexéo, 2023).

5.2.2 Dimensionality Reduction for Word Embeddings

The dimensionality reduction occurs in the case of Word2Vec for LSTM, where it selects the top 20000 most frequent words, and Keras for CNN, where it reduces the dimension of the feature size from 20000 to 100. This typically filters out uninformative tokens to save memory usage and increases the speed of the training process to ensure stabilization and improvement for the sentiment modeling process (Raunak, 2017).

5.3 Comparison between TF-IDF and Word Embeddings:

While TF-IDF is applied for logistic regression as a feature extraction method that tends to create sparse representations of text where each word has a unique slot, leading to high-dimensional vectors, word embeddings typically embed words in a multidimensional space, giving continuously dense vectors acting as a compact summary to represent the meaning of words. A traditional method like IF-IDF treats words and isolated tokens; they rarely capture the relationship between words and their context. For example, it is possible that ”good” and ”great” are treated as completely different words despite their similar meanings. Meanwhile, word embeddings can capture the surrounding words and how they connect, since this approach places words with the same meanings close to each other. Furthermore, word embeddings can capture synonyms and polysemy effectively, which better their understanding of the exact meaning of a word in a particular context. Thus, word embeddings tend to outperform TF-IDF in model performance since the classifiers capture more information in fewer dimensions. In section 7, word embeddings are applied to CNN and LSTM models, achieving a higher accuracy in sentiment classification than logistic regression using TF-IDF (Zhan, 2025; Rudkowsky et al., 2018).

5.4 Conclusion:

Thus, the resulting numerical feature vectors $\mathbf{x} = \Psi(d)$ from the encoding process of TF-IDF and word embeddings are used as inputs for individual classification algorithms.

$$h : \mathbb{R}^{|\mathcal{V}|} \rightarrow \mathcal{C}$$

Then we can approximate the target function $g : \mathcal{D} \rightarrow \mathcal{C}$ describing the true sentiment label $c_j \in \mathcal{C}$ for each document $d_j \in \mathcal{D}$.

6 Classification Algorithms

Our goal is to train a model

$$f = h \circ \Psi : \mathcal{D} \rightarrow \mathcal{C}, \quad \mathcal{C} = \{-1, 0, +1\}$$

using the training set Tr and optimize it (when applicable) using the validation Va that assigns each document $d_j \in \mathcal{D}$ to a corresponding sentiment label $c_j \in \mathcal{C}$.

6.1 Logistic Regression (LR)

6.1.1 Motivation and justification

The logistic regression model is a good starting point to form a baseline due to their simple architecture, high computational efficiency. Although the Logistic Regression model has a better ability to process the linear relationship between the input features and output labels, the presence of increasingly complex language structure shows the disadvantage of LR (Lin et al., 2023).

The logistic regression model is easier to implement, interpret, and is very efficient to train. It can easily extend to multiple classes, such as multinomial regression. It is very fast at classifying unknown records, especially for many simple datasets, and performs well when the dataset is linearly separable. Nevertheless, the logistic regression can overfit in high-dimensional datasets if the number of observations is less than the number of features. The limitation of Logistic Regression is that it sticks solely to the assumption of linearity; thus, non-linear and complex relationships cannot be handled properly when linearly separable data is rarely found in real-world scenarios. More powerful and compact classification algorithms, such as Neural Networks, can outperform logistic regression (Hosmer Jr et al., 2013).

6.1.2 Model explanation

A model we have employed is a Logistic Regression for Multi-Class Text Sentiment Classification. The classification is performed using a supervised machine learning pipeline centered around multinomial logistic regression with class weighting. Logistic regression is particularly appropriate for problems where interpretability, efficiency, and linear decision boundaries are desirable.

The high interpretability is an advantage, and is found through the fact that each coefficient directly reflects the influence of a specific feature on the log-odds of a given sentiment class (Hastie et al., 2009). For example, the word "excellent" could receive a strong positive coefficient, meaning its presence increases the log-odds of the text being classified as positive sentiment. In contrast, the word "okay" might receive a coefficient close to zero.

Unlike binary logistic regression, which learns a single decision boundary, the multinomial formulation extends the model to learn separate decision functions for each class, which allows the classifier to simultaneously consider all sentiment categories, rather than decomposing the task into multiple binary subproblems (Sebastiani, 2002b).

In multinomial logistic regression, the model computes

$$P(y = c \mid \mathbf{x}) = \frac{e^{\mathbf{w}_c \cdot \mathbf{x}}}{\sum_{k=1}^{K} e^{\mathbf{w}_k \cdot \mathbf{x}}}$$

where X is the input feature vector, Wc is the weight vector for sentiment class c, and K=3 is the total number of classes. For each sentiment class, it computes a linear activation $\mathbf{w}_c \cdot \mathbf{x}$, the dot product of class weights and input features. The softmax function is applied to the linear activation, which normalizes them into probabilities that sum to 1 (Bishop, 2006).

The predicted class is chosen as the one with the highest estimated probability.

$$\hat{y} = \arg\max_{c} \; P(y = c \mid \mathbf{x})$$

The model operates over a feature space constructed via TF-IDF vectorization, which is discussed in Section 5, where each feature represents the weighted presence of a term in the input text document. The classification model is implemented using `sklearn.linear_model.Logi` (Pedregosa et al., 2011).

To prevent overfitting and reduce computational cost, we aimed to reduce the dimensionality of the TF-IDF feature space by using the `max_features` parameter. This hyperparameter limits the number of features retained in the TF-IDF matrix to the top-k

most informative ones, based on term frequency across the corpus. By discarding both rare and overly common terms, the resulting representation becomes more robust and generalizable. In our experiments, we found that setting `max_features` between 20,000 and 30,000 improved model performance the most, particularly by boosting weighted average F1-score.

To identify the optimal value for `max_features`, we performed hyperparameter tuning by means of 5-fold stratified cross-validation. For each setting, the model was trained and evaluated on different folds, and the average weighted F1-score was recorded. Figure 8 plots the relationship between `max_features` and cross-validated F1 performance. The plot shows that the peak performance occurred around `max_features` = 29000. This tuning allowed us to select an optimal feature limit to balance expressiveness and generalization while optimizing prediction accuracy.

Figure 8: Weighted F1-Score based on the number of features used

Training is performed using the Limited Memory BFGS (LBFGS) solver, which approximates the Broyden–Fletcher–Goldfarb–Shanno algorithm (BFGS) with the use of less memory (Liu and Nocedal, 1989). It is well-suited for the minimization of convex loss functions such as our own multinomial log-likelihood. Moreover, it is more efficient for high-dimensional feature spaces like those produced by TF-IDF vectorization, and it converges reliably for moderate-to-large datasets (Pedregosa et al., 2011).

The contribution of each category to the loss function can be balanced through the `class_weight` parameter, by adjusting according to its relative frequency in the training data. As a result, underrepresented classes (e.g., neutral sentiments) are less likely to be overshadowed by majority classes during optimization (learn developers, 2024). In our specific case, neutral sentiments are not underrepresented. Rather, they are harder to discern than positive or negative sentiments in NLP by their nature, as neutral sentiments often lack emotionally

charged words. This is why balancing weights still proves beneficial (Zhang et al., 2018).

The combined dataset of both Tweets and Amazon reviews is split into a training and test set using stratified sampling, to preserve the class proportions of both datasets (Neyman, 1934).

We added an `is_tweet` dummy variable to adjust for domain-specific language patterns, which increased the accuracy. However, the infrastructure of the testing platform for this specific case study does not provide information about whether the test document is a tweet or a review, whereas the dataset for training and validating does. It is still valuable to know that incorporating this information enhances accuracy, seeing as in real-world applications, source information like this is readily available in most cases.

As Hastie et al. (2009) mentions in 'The Elements of Statistical Learning', while more complex models may outperform logistic regression in raw accuracy, the interpretability and training efficiency of LR make it a valuable baseline and a strong choice for production pipelines.

6.2 Convolutional Neural Networks(CNN)

6.2.1 Motivation and justification

Our reasoning to switch from Logistic Regression to Convolutional Neural Network is that CNN can capture non-linear relationships and text dependencies more accurately, with automatic feature extraction helping CNN absorb important features, capture local and global dependencies in the input data, and understand hierarchical features that LR can miss. This allows CNN to achieve a higher sentiment classification accuracy (LeCun et al., 2015; Kim, 2014).

The first obvious advantage of CNN is the ability to capture both local and global dependencies in the text input. CNN can also extract important features from input data without manual feature extraction. Convolutional Neural Networks excel at extracting hierarchical features from text, capturing local patterns essential for understanding complex language structures. However, CNN requires lots of data and computing power. Also, it can overfit when the training dataset is small or the model is complex. Lastly, CNN can be vulnerable to corrupt input data and make inaccurate predictions (LeCun et al., 2015; Kim, 2014).

6.2.2 Model explanation

A convolutional classifier

$$h_{\text{CNN}} : \mathbb{R}^{d \times |\mathcal{V}|} \to \mathcal{C}$$

is applied through 1D convolutional and pooling layers, followed by fully connected layers and a softmax output:

$$\hat{\mathbf{y}} = \text{softmax}(h_{\text{CNN}}(\mathbf{x})).$$

Convolutional Neural Networks(CNNs) are a type of feed-forward neural network that is widely used in image recognition tasks. Unlike simple Artificial Neural Network(ANN) models that only consist of input layers, hidden layers, and output layers, CNNs utilize multiple layers to learn hierarchical feature representations. The structure of our model builds on Kim (2014) and Cliche (2017b). Figure 9 depicts the structure of the CNN model used. Thereafter, different layers that were considered or implemented in the model are discussed.

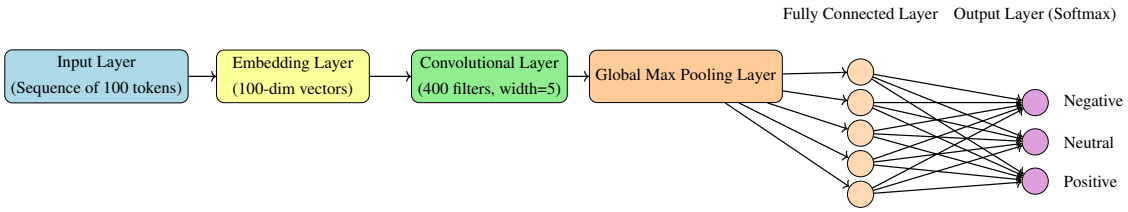


Figure 9: Constructed CNN architecture for text classification. The model processes input tokens through an embedding layer, a convolutional layer, and a global max pooling layer. The output is passed to a fully connected (dense) layer with ReLU activation, followed by a softmax layer for three-class sentiment prediction.

- Embedding layer: this layer projects discrete tokens (t_j) into a low-dimensional vector space (E-dimension) to obtain semantically well-designed word vectors (e_j), where words with similar meanings are located close, and the words with opposite meanings are located far apart.

  More formally, each input document $d_j \in \mathcal{D}$ is mapped via the encoder Ψ to a sequence of word embeddings $\mathbf{X}$:

  $$\mathbf{X} = \Psi(d) = [\mathbf{e}_1, \mathbf{e}_2, \ldots, \mathbf{e}_{|\mathcal{V}|}]^T, \quad \mathbf{e}_i \in \mathbb{R}^d.$$

  where $\mathbf{E} \in \mathbb{R}^{d \times |\mathcal{V}|}$ is the embedding matrix, $|\mathcal{V}|$ is vocabulary size, and d is embedding dimension. Mikolov et al. (2013a)

  The Embedding layer is useful for sentiment classification, since it collects words with similar semantics in the same vectors. Dimensionality reduction is also performed in this layer by creating dense representations for features rather than a

sparse vector space. This layer is optional, as the embedding process can instead be done outside the model in a process such as Word2Vec. Here, the embedding was performed during training using an embedding layer to exploit the structure of the CNN that allows documents to get encoded internally, which allows the model to understand the end-to-end representation of the features inside the data.

- Convolutional Layer: this layer performs a convolution operation on the input sequence of word embeddings to extract local features and produce a set of feature maps Krichen (2023). In this report, we use a one-dimensional convolution, as textual data is naturally represented as a sequence rather than a multi-dimensional grid.

  The output of the convolution operation at position i using filter m is given by:

$$c_{m,i} = \sigma\left(\sum_{j=0}^{k-1} \mathbf{w}_m[j] \cdot \mathbf{e}_{i+j} + b_m\right)$$

  Kim (2014)

  Where:

  - $c_{m,i}$: Activation at position i from filter m.
  - $\sigma(\cdot)$: Non-linear activation function, such as ReLU.
  - k: Filter (kernel) size.
  - $\mathbf{w}_m \in \mathbb{R}^{k \times d}$: Weight vector of filter m, where d is the embedding dimension.
  - $\mathbf{e}_{i+j} \in \mathbb{R}^d$: Word embedding at position $i + j$.
  - $b_m \in \mathbb{R}$: Bias term for filter m.

  The convolutional layer is applied after the embedding layer to extract the locally key features, such as unigrams and bigrams, that are strong signals of sentiment. The complex relationship between words can also be understood better by the activation function $ReLU$, which highlights the importance of analysis in the relationship between words in a sentence, for sentiment classification (Kim, 2014).

- Pooling Layer: this layer inputs the set of feature maps c_f that was outputted in the previous layer as input, and performs dimension reduction to increase computation speeds, reduce memory, and prevent overfitting. The operation is performed independently on each feature map and reduces the data by taking the maximum(max-pooling) or average value(average-pooling) of non-overlapping regions. In our

model, we opt for max-pooling, since in classification tasks, the presence of strong features is crucial.

More formally, this layer outputs:

$$p_f = \max_i c_{f,i}$$

Boureau et al. (2010)

which reduces the feature map c_f to a single scalar per filter.

Pooling plays a crucial role in vector compression, since it gathers all information in a single scalar, keeping the most crucial features instead of the noisy ones. In the pooling process, the precise location of important features is considered less important, and instead, counting the occurrences provides more information (Gholamalinezhad and Khosravi, 2020).

- Fully-connected layer: this layer is a traditional NN layer that connects every neuron in the previous layer to every neuron in the current layer (Krichen, 2023). This layer serves as the connection mechanism after the pooling layer to generate the output layer.

$$h = \sigma(\mathbf{W}\mathbf{p} + \mathbf{b})$$

LeCun et al. (1998)

where $\mathbf{W}$ and $\mathbf{b}$ are weights and biases of the dense layer, and $\mathbf{p} = [p_1, p_2, \ldots, p_F]^T$ is the pooled vector.

A fully-connected layer is an intermediate step before the output layer. In particular, it synthesizes all the processed features from the convolutional and pooling layers and applies a non-linear transformation via $ReLU$ to help the models understand the complex relationship between these features, and thus make a final sentiment decision (Kiranyaz et al., 2021).

- Activation layer: this layer introduces non-linearity to the network, allowing it to learn more complex features (Krichen, 2023). The activation function determines which information should be transmitted from one neuron to the next. It is applied to the output value of neurons in the previous layer and passes the processed value to the next layer (Li et al., 2022). Two activation functions are used in the model:

ReLU, used to activate neurons from the pooling to the fully-connected layer:

$$\sigma(x) = \max(0, x)$$

Nair and Hinton (2010)

Softmax, used to create a three-class sentiment prediction:

$$\sigma(\mathbf{z})_i = \frac{e^{z_i}}{\sum_j e^{z_j}}$$

Bridle (1990)

The two types of functions in the activation layer increase predictive performance in sentiment classification. $ReLU$, a nonlinear mapping function, allows CNN to dive deep into complex patterns from the features instead of linear layers. In addition, the Softmax function produces a probability distribution to perform the final sentiment classification, which enables the activation layer to transform a complex pattern into a comprehensive probability interpretation for the sentiment decision (Zhang et al., 2018).

- Drop-out layer: this layer drops out a set percentage of neurons in the previous layers, during the training of the current layer, which prevents overfitting and helps to learn the robust features of the data (Srivastava et al., 2014). In our sentiment classification task, the datasets are medium-sized and noisy. It ensures a more robust representation of the n-grams process through the model. However, using a dropout layer resulted in a decrease in the validation accuracy in our model, so it was not used.

The combined layers compose our model f_{CNN} to estimate g.

6.3 Hyperparameter estimation

The layers require several hyperparameters to be configured. The embedding layer requires the number of *embedding dimensions* to determine the size of the word-embedding vectors. The convolutional layer requires the *number of filters* (kernels) and *kernel size* to control the width of the convolution kernel. Lastly, the fully-connected layer requires the number of *hidden dimensions* to control the number of neurons in the dense layer after pooling.

To compute the optimal hyperparameters for the model, automated tuning was used to train multiple models with different hyperparameters for a few epochs. The best-performing models(in terms of validation accuracy) were kept, and they were subsequently trained with more epochs. Consequently, the best-performing model was selected with the optimal hyperparameters in table 2.

Hyperparameter	Type	Values Tried	Optimal Value
Embedding Dimensions	Integer	50, 100, 150, 200	100
Number of Filters	Integer	100, 150, 200, 250, 300, 350, 400	400
Kernel Size	Categorical	3, 5, 7	5
Hidden Dimensions	Integer	100, 150, 200, 250, 300, 350, 400	150

Table 2: Hyperparameter search space and optimal values found via tuning

The optimal hyperparameters are then used to construct the CNN model. Additionally, using an ensemble of three CNN models for three sets of convolutional filter sizes: [1,2,3], [3,4,5], and [5,6,7] was considered. However, given our data and the general structure of the model, the ensemble underperformed the unit CNN model. The possible reasons for this may be the higher number of parameters to optimize in an ensemble or that there is less training data per model.

6.4 Bidirectional Long Short-Term Memory (Bi-LSTM)

6.4.1 Motivation and justification

Long Short-Term Memory, particularly bidirectional LSTM (Bi-LSTM), is our chosen final model for sentiment classification. LSTM differs from LR and CNN since it can handle long-term dependencies, performs well on sequence data, and can handle different variable-length inputs. However, Bi-LSTM is chosen as the final model to capture the long-term dependencies in sequential data in both forward and backward directions, which combines the power of LSTM with bidirectional processing, allowing the model to capture both past and future context of the input sequence (Mahadevaswamy and Swathi, 2023).

Bi-LSTM captures not only the context that comes before a specific time step but also the context that follows. Bi-LSTM can capture richer dependencies in the input sequence by considering both past and future information. The architecture of bidirectional LSTM is flexible and can be customized by adding extra layers, such as convolutional or attention layers, to improve performance. Bi-LSTM often performs better than traditional LSTM on tasks like sentiment analysis. However, Bi-LSTM can be computationally expensive due to the need to process the input sequence in both directions. This can make it impractical to implement a Bi-LSTM in a resource-constrained environment. Bidirectional LSTM requires large amounts of training data to learn meaningful representations of the input sequence. Without sufficient training data, the model may overfit to the training set or fail to generalize to new data (Nama, 2021; Huang et al., 2015).

6.4.2 Model explanation

Long Short-Term Memory (LSTM) networks are a type of recurrent neural network (RNN). Unlike traditional RNN language models that suffer from the exploding and vanishing gradient problem during the backpropagation training stage, LSTM overcomes this problem by using the memory cell along with input, forget, and output gates to selectively retain or discard the information (Cliche, 2017b). This highlights that RNN with LSTM performs better in many experiments in the case of structures with conjunctions involving "although...but...", "not only... but also...", "however", and so on. LSTM can also identify long statements and achieve multi-classification for text emotional attributes more accurately than conventional RNNs (Li and Qian, 2016).

Similar to the case of CNN, we use the same sequence as input for LSTM

$$\mathbf{X} = \Psi(d) = [\mathbf{e}_1, \mathbf{e}_2, \ldots, \mathbf{e}_{|\mathcal{V}|}]^T, \quad \mathbf{e}_i \in \mathbb{R}^d.$$

We let $h_{\text{LSTM}} : \mathbb{R}^{d \times |\mathcal{V}|} \to \mathcal{C}$ defines the LSTM classifier. The prediction is expressed as follows:

$$\hat{\mathbf{y}} = \text{softmax}(h_{\text{LSTM}}(\mathbf{X})).$$

The embedding layer is processed as follows:

$$\text{Embedding(input dimension=}|\mathcal{V}|, \text{ output dimension=}d)$$

An LSTM comprises three gates to control the usage and update of the text history information involving the input gates, the forget gates, and the output gates, respectively. The memory cell and these three gates enable LSTM to read, save, and update the long-distance history information. The calculation process of LSTM networks includes 4 main steps.

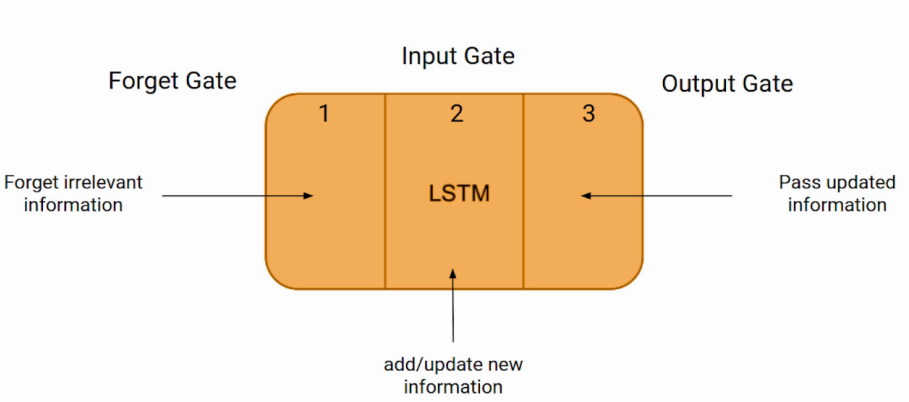


Figure 10: Illustration of Three Main Gates of LSTM
Input Gate, Forget Gate, and Output Gate

$$f_t = \sigma(W_f \cdot x_t + U_f \cdot h_{t-1} + b_f) \quad (1)$$
$$i_t = \sigma(W_i \cdot x_t + U_i \cdot h_{t-1} + b_i) \quad (2)$$
$$o_t = \sigma(W_o \cdot x_t + U_o \cdot h_{t-1} + b_o) \quad (3)$$
$$c_t = f_t \circ c_{t-1} + i_t \circ \tanh(W_c \cdot x_t + U_c \cdot h_{t-1} + b_c) \quad (4)$$
$$h_t = o_t \circ \tanh(c_t) \quad (5)$$

The first step occurs in the forget gate f_t, identifying which information from the cell state should be discarded. The second step is the input gates i_t that decide what information to store in the cell state. The third step is to update the old cell state and use the information from the input gates and forget gates to calculate the updated values of the LSTM cell state c_t. Next, the output gates o_t determine the value of the output based on the state of the cell. In the final stage, the output of the whole cell is updated, which can be illustrated in Figure 10. Note that in the above functions, h_t is the regular hidden state, σ is the sigmoid function, and $\circ$ is the Hadamard product (Cliche, 2017b).

However, there is a disadvantage of LSTM in that it accurately and sufficiently post-processes word information because a sentence is processed in only one direction. Therefore, we implement a bidirectional LSTM to capture more contextual information. A bidirectional LSTM consists of two separate LSTM layers where one LSTM reads the sentence forward and the other reads it backward, and then the outputs of the two LSTMs are stacked together, thus making Bi-LSTMs particularly effective for tasks for understanding both the past and future context, and its bidirectional architecture provides better sentiment classification results than its unidirectional counterparts (Cliche, 2017b; Derra and Baier, 2020).

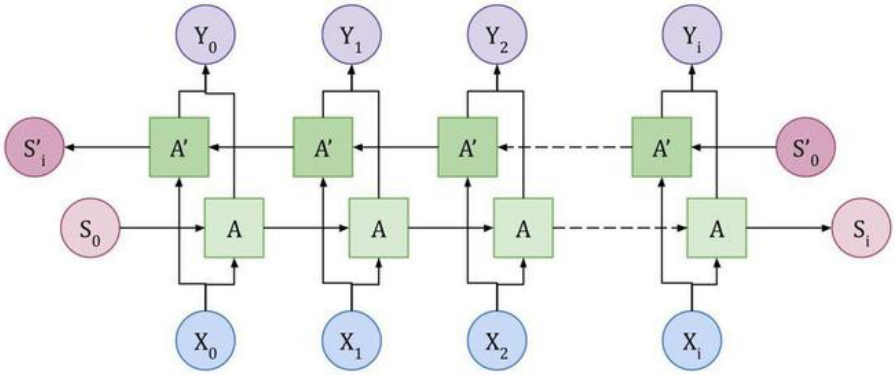


Figure 11: Illustration of the Bidirectional LSTM structure showing forward and backward sequence processing.

Note: X_i is the input token, Y_i is the output token, A and A' are the forward and backward LSTM units, and the final output of Y_i is the combination of A and A' LSTM nodes.

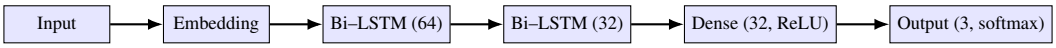


Figure 12: LSTM-Based Sentiment Classification Architecture. The model processes embedded input through two bidirectional LSTM layers, followed by a dense layer and a 3-class softmax output.

Our architecture for Bi-LSTM is illustrated in Figure 12. We first used Word2Vec to turn each word into a number and give a list of word vectors that have numeric representations that are then used in training. Next, the preprocessed data input goes through 2 layers of LSTM, and we use Bi-LSTM, an extension of the traditional LSTM network allows the sequences to be processed both forward and backward to better understand the context. Specifically, the first LSTM layer has 64 memory cells and there are 32 units of memory cell in the second LSTM layer to store and process information to store and process the information. After 2 layers of LSTM, the obtained results are passed to a dense classifier that accounts for 32 units of memory cell and provides three outputs: positive, negative, and neutral. The key components of LSTM are input, forget, and output gates make the model keep and update useful parts of information and discard what is not meaningful in a sentence (Wang et al., 2018), and (Sunny et al., 2020).

The two layers of Bi-LSTM. The first LSTM layer contains 64 bits, focusing on capturing key local features from the processed input, such as short phrases with negations showing the sentiment clue. The second LSTM layer with 32 bits builds on these local features after the first layer to provide a more refined and robust representation of features that are useful for the final classification (Wang et al., 2018).

Table 3: Choice of Hyperparameters and Training process for Bi-LSTM

Parameters	Choice and Justification	References
Units per LSTM Layer	First layer contains 64 units to capture larger contextual embedding and 32 units for the second layer to reduce dimensionality and enhance feature representation.	Reimers and Gurevych (2017)
Embedding Layer	Freezing embeddings using Word2vec to improve generalization performance.	Welch et al. (2020)
Batch Size	A batch size of 64 gives the optimal validation performance.	Glenn et al. (2023)
Optimizer & Learning Rate	Adam optimizer is applied with learning rate of 0.0001	Aslan (2022)
Early Stopping Criteria	Stops if there is no improvement in validation loss with the chosen 3 epochs.	Aslan (2022)
Validation Split	Extracted 10 % of training set manage generalization.	Standard Regularization Technique

Our chosen hyperparameters for two LSTM layers are 64 and 32 units, respectively, for a bidirectional architecture. The reason why the first LSTM layer contains 64 units and then reduces to 32 units for the second LSTM layer is to ensure the first layer can have the ability to capture a broader and deeper representation of the input sequence, and the second layer can take into account refining and compressing this information. Moreover, using fewer units of memory cells can avoid the risk of the overfitting problem and rising computational cost. Our design is based on the conclusion from Reimers and Gurevych (2017), who found that one and two LSTM layers with around 50 to 100 yield robust performances. We also decided to freeze the word embeddings that were trained by using Word2Vec. According to Welch et al. (2020), freezing pre-trained input embeddings can improve language model performance and provide better generalization performance than fine-tuning, yielding high efficiency in medium-sized datasets.

Bi-LSTMs are trained using the following procedures. Firstly, a batch size of 64 is applied, and this aligns with the results of Glenn et al. (2023), demonstrating that a batch size of 64 yields optimal validation performance while smaller and larger size values have lower validation accuracy. Additionally, we followed the study of Aslan (2022) where an

Adam optimizer with a learning rate of 0.0001 is used to ensure a stable convergence rate. We also applied an early-stopping criterion with an epoch set equal to 3 to regulate validation loss efficiently and avoid overtraining. We then conducted a validation split of 10 % during the training process to keep track of generalization performance. These methods above not only ensure the Bi-LSTM model learn quickly without overfitting, but they also ensure to stop the training process when this model stops improving.

7 Performance Evaluation

7.1 Performance Metrics

The task objective is to compare the performance of LR, CNN, and Bi-LSTM in out-of-sample prediction of sentiment, when applied on the test set Te, across the classes in $\mathcal{C}$.

7.1.1 Estimation of metrics

For each category c_i: The contingency table 4 defines the terms that link the true judgment to the classifier judgment, where TP_i is the number of documents that both the classifier and true judgments agree that belong to c_i, TN_i is the number of documents that both the classifier and true judgments agree that do not belong to c_i, FN_i is the number of documents that the classifier predicts does not belong to c_i while the true judgment is that it belongs to c_i. FP_i is similar to FN_i, but the classifier and true judgments are reversed. In the ternary context, given that the premise is single-label classification, for each class c_i, a *one-vs-rest* approach is adopted: c_i is treated as the positive class and all others as negative.

Category c_i	**true judgments**	
	YES	**NO**
classifier judgments YES	TP_i	FP_i
classifier judgments NO	FN_i	TN_i

Table 4: The contingency table for category c_i (Sebastiani, 2002b)

Given that the defined true model is $g(d_x, c_i)$, and the estimated model is $f(d_x, c_i)$,

different conditional and unconditional metrics can be defined.

Accuracy (α) is the unconditional probability of true prediction for all classes: $P(f(d_x) = g(d_x))$. While accuracy is simple to compute, it comes with the disadvantage that it doesn't convey information about the performance in each sentiment class (Grandini et al., 2020).

Precision (π) with respect to a classification c_i is defined as the probability that if a random document d_x is classified under c_i, this decision is correct: $P(g(d_x, c_i) = T \mid f(d_x, c_i) = T)$ (Sebastiani, 2002b). For example, for a random document classified by the model to have a *positive* sentiment, precision measures the proportion that it has a true positive sentiment.

Recall (ρ) with respect to a classification c_i is defined as the probability that the model classifies a document d_x under c_i, given that it truly belongs to c_i: $P(f(d_x, c_i) = T \mid g(d_x, c_i) = T)$ (Sebastiani, 2002b). For example, for a random document that has a true positive sentiment, recall measures the probability that the model classifies it under $c_i = 1$.

F1-score is the harmonic mean of precision and recall, used to yield a univariate (single number) performance measure (Christen et al., 2023).

In practice, these measures can be estimated for the test set Te using the terms in the contingency table 4:

- **Precision:** $\pi_i = \dfrac{TP_i}{TP_i + FP_i}$
- **Recall:** $\rho_i = \dfrac{TP_i}{TP_i + FN_i}$
- **F1-score:** $F1_i = \dfrac{2 \cdot \pi_i \cdot \rho_i}{\pi_i + \rho_i}$
- **Accuracy:** $\alpha = \dfrac{\sum_{i=1}^{|C|} TP_i}{\sum_{i=1}^{|C|}(TP_i + FP_i + FN_i)}$

7.1.2 Averaging Strategies

To compute an estimate of the unconditional F1-score (w.r.t. c_i), two averaging strategies are considered: Macro-averaging and Weighted averaging. To compute the averaging, the arithmetic (weighted) mean over the harmonic means of recall and precision is taken, instead of the alternative approach, which comprises taking the harmonic mean over arithmetic means (See Opitz and Burst (2021)). The unconditional and conditional estimates of precision, recall, and F1-score can then be compared across the three models.

- **Macro-averaging**: $F1^M = \frac{1}{|\mathcal{C}|}\sum_{i=1}^{|\mathcal{C}|} F1_i$. Macro-averaging takes the average of $F1$ without assigning different weights based on $F1_i$.

- **Weighted-averaging**: $F1^{\text{W}} = \sum_{i=1}^{|\mathcal{C}|} w_i \cdot F1_i$, where $w_i = \frac{support_i}{\sum_{i=1}^{|C|} support_i}$. Weighted-averaging takes the weighted-average of $F1$ by using the proportion of the support for $F1_i$ to the sum of all supports.

7.2 Comparing the three models

7.2.1 Comparison Metrics

Following the approach of Cliche (2017b) and considering the findings of Hinojosa Lee et al. (2024) that recommend using multiple comparison metrics, we select Accuracy, Macro-average F1(herein M-F1), and Weighted-average F1(herein W-F1) to evaluate the performance of our models. Accuracy computes the overall ability of the model in making predictions. M-F1 can reflect on the performance of smaller classes better, given that future out-of-sample classifications may be performed on an imbalanced dataset. On the other hand, W-F1 assigns larger weights for classes with larger data, which helps consider the overall performance when some classes are expected to be larger than others.

In terms of accuracy, the **Bi-LSTM**, CNN, and LR achieved a score of 72.8%, 72.2%, and 71.4%, respectively. The **LR model** achieved the highest M-F1 score of 0.698, closely followed by Bi-LSTM with 0.696, both outperforming CNN, which scored 0.686. CNN also achieved the lowest W-F1 score out of the three at 0.713. However, **Bi-LSTM** outperformed LR here, achieving 0.722, compared to 0.719. Bi-LSTM achieves this higher score because its F1-scores for the positive and negative classes are higher than LR, even though its F1-score for the neutral class is lower. Since these two classes have more data compared to the neutral class, W-F1 assigns the highest weights to their F1-scores, leading Bi-LSTM to outperform LR. Although CNN also achieves higher scores, compared to LR, for classes with larger weights, the difference is not large enough to result in a higher W-F1 score.

The pros and cons subsection of the models in section 6 provides some insight into the reasoning behind the performance differences in each model.

- Bi-LSTM can handle sequences in both directions, which allows for deeper feature learning. This explains why Bi-LSTM achieves the highest performance across the models.

- LR performs well when the classes are linearly separable. This allowed the weight-balancing process for the classes to get implemented without information loss, leading to it to predict well for the neutral sentiment class.

- The convolutional layer in CNN works well to extract features; however, it is prone to overfitting towards the less-represented classes, leading to lower overall prediction accuracy.

Class	Precision	Recall	F1-Score	Support
Negative	0.740	0.802	0.770	13468
Neutral	0.555	0.426	0.482	9528
Positive	0.780	0.836	0.807	15267
Accuracy			0.722	38263
Macro Avg	0.692	0.688	**0.686**	38263
Weighted Avg	0.710	0.722	**0.713**	38263

Table 5: CNN classification report showing precision, recall, and F1-score for each class.

Class	Precision	Recall	F1-Score	Support
Negative	0.722	0.842	0.777	13468
Neutral	0.557	0.457	0.502	9528
Positive	0.824	0.795	0.810	15267
Accuracy			0.728	38263
Macro Avg	0.701	0.698	**0.696**	38263
Weighted Avg	0.722	0.728	**0.722**	38263

Table 6: Bi-LSTM classification report showing precision, recall, and F1-score for each class.

Class	Precision	Recall	F1-Score	Support
Negative	0.768	0.749	0.758	13468
Neutral	0.503	0.590	0.543	9528
Positive	0.831	0.759	0.793	15267
Accuracy			0.714	38263
Macro Avg	0.701	0.699	**0.698**	38263
Weighted Avg	0.727	0.714	**0.719**	38263

Table 7: LR classification report showing precision, recall, and F1-score for each class.

7.2.2 Confusion Matrices

The confusion matrices in Figure 13 provide visual insight into inter-class differences in the prediction accuracy of the three models.

- **Neutral**: LR performed best in predicting the neutral sentiment, outperforming both Bi-LSTM and CNN. In regard to incorrect classifications, LR, like Bi-LSTM, predicted more incorrect negative sentiments; the predictions of CNN are more evenly split.

- **Positive**: Here, CNN performed best, classifying the most true negative sentiments correctly, followed by Bi-LSTM and LR. However, CNN also predicted marginally higher incorrect negative sentiments compared to LR.

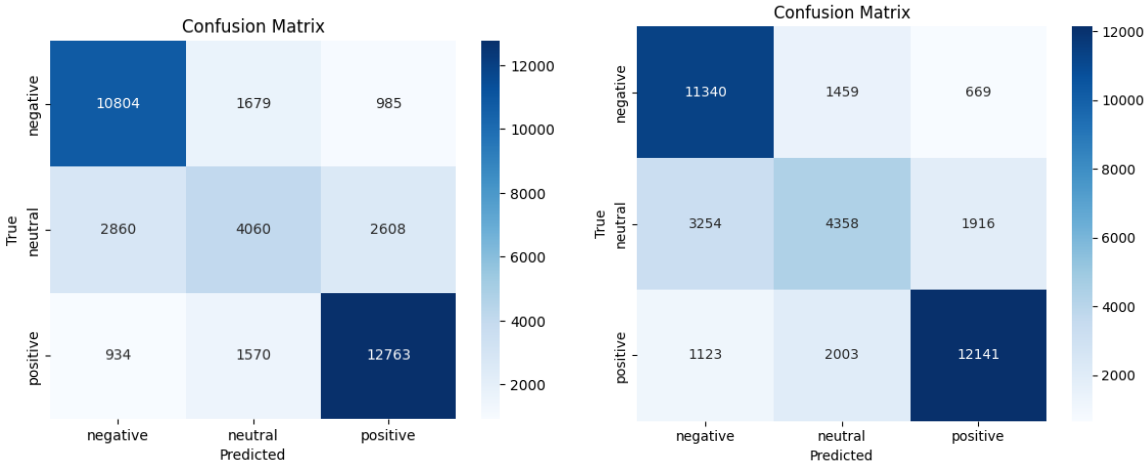
- **Negative**: Bi-LSTM performed best at identifying the true negative sentiment, followed by CNN and LR.

  Each of the models outperforms the others in one of the classes: LR for neutral, CNN for positive, and Bi-LSTM for negative. Therefore, for selecting the "best" model, we must consider which aspects of the classification task are most prioritized.

In some sentiment classification tasks, predicting the polar opposite(negative vs positive) sentiment is highly undesirable. An example of this would be using an automated process that predicts the positive sentiment of a user to ask for a 'follow' on a company's Twitter or review platform. If the true sentiment of a document were to be negative, asking for a 'follow' would not be appropriate and may result in more dissatisfaction for the user. As a result, in such scenarios, the LR model may be preferred, as it predicts the lowest classifications that are the polar opposite of the true classification. Furthermore, LR achieves the highest M-F1 score, making it desirable when predicting datasets with under-represented classes, as is the case in both tweets and reviews.

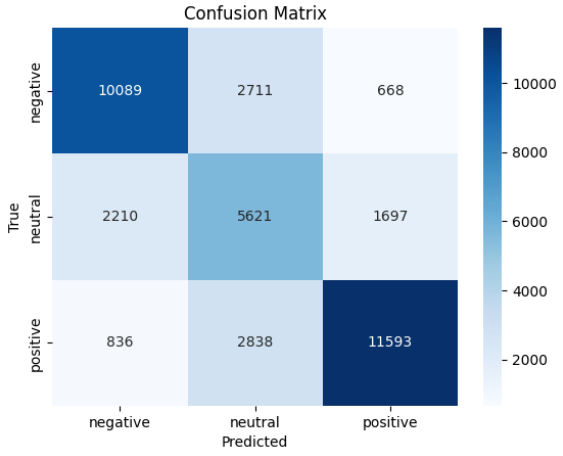
On the other hand, if the goal is to maximize overall prediction accuracy while focusing on performance for the most frequent classes, Bi-LSTM would be the better choice, as it yields the highest accuracy and W-F1.

The goal of our analysis was to find the highest-performing model for ternary sentiment classification for tweets and Amazon reviews. Since Bi-LSTM achieves the highest accuracy and W-F1, we select it as our best model for this task. Although the LR model achieves the highest M-F1 due to high performance in predicting the neutral sentiment, the number of documents with a neutral sentiment in D is low, given that only reviews with a 3-star rating are assigned this sentiment. As a result, the predictions of Bi-LSTM more accurately reflect the dataset's overall sentiment distribution. Furthermore, negative and positive sentiments play a larger role in the decision-making process of businesses, and their correct prediction may be prioritized.



(a) CNN

(b) Bi-LSTM



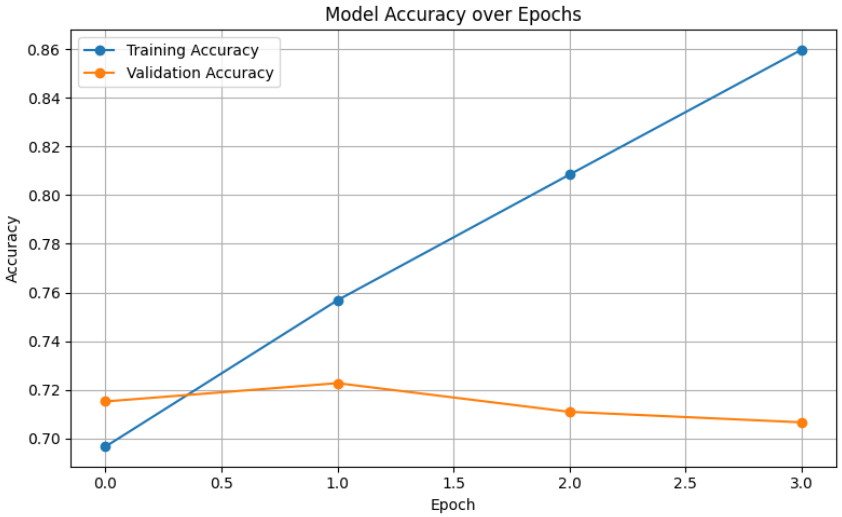
(c) LR

Figure 13: Confusion Matrices for CNN, Bi-LSTM, and LR

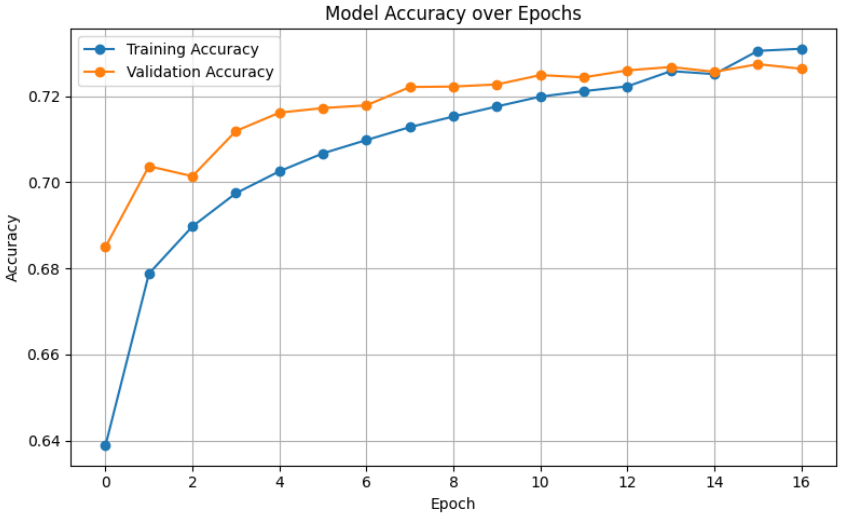
7.2.3 A Note: Efficiency Comparison

For the two neural network models, CNN and Bi-LSTM, the model accuracy across different epochs can be compared in Figure 14. Bi-LSTM starts with a very low training accuracy and a low validation accuracy. Across epochs, both training and validation accuracies gradually increase until around the 11th epoch, where the validation accuracy begins to decline. CNN, however, reaches its maximum validation accuracy at the first epoch. While training accuracy increases quickly across the next epochs, validation accuracy does not recover.

The comparison shows that CNN can achieve strong performance in prediction early, while Bi-LSTM needs more passes through the training data to optimize. Although Bi-LSTM was selected as the better sentiment classifier between the two, the computation cost of training becomes a significant consideration for large datasets. Furthermore, using a data center-class GPU to train the models, the average computation duration of an epoch was 20 seconds for CNN, considerably lower than that of Bi-LSTM, which was 45 seconds. Therefore, in classification tasks where speed and scalability are prioritized, the marginal gains in accuracy may not always justify the large gap in computation and time costs.



(a) CNN



(b) Bi-LSTM

Figure 14: Model accuracy over Epochs for CNN and Bi-LSTM.

In this paper, we presented the preprocessing, feature engineering, and classification approaches that we used to compare the performance of Logistic Regression to two typical Neural Networks. Our goal was to train the models on the text data input of tweets and reviews, investigate strategies to boost performance, and compare the final results. We saw that the neural network models involving CNN and Bi-LSTM achieve a higher accuracy than LR. This performance gap may be attributed to the structure of CNN and Bi-LSTM, which allowed them to capture complex patterns and nonlinear relationships. Furthermore, due to their use of word embeddings, they could capture deeper semantic and syntactic relations from the documents, enabling better generalization and understanding of context within the text. These inter-document relations are not readily accessible to TF-

IDF, on which our LR model was built. However, we also saw that in unbalanced datasets, LR can achieve better results in terms of Weighted-Average of the F1-scores and is less susceptible to overfitting on over-represented classes.

8 Future Improvement:

To increase efficiency in the running time of the training process, dimensionality reduction, and prediction performance of sentiment classifiers, we would implement other feature selection techniques for future improvement. One of them is a chi-squared test for feature selection to automatically gather the top informative and relevant terms for three-way sentiment labels involving positive, negative, and neutral (Abdulkhaliq and Darwesh, 2020). Another potential method to reduce feature dimensionality for TF-IDF is to implement a matrix decomposition method like Principal Component Analysis (PCA) and Singular Value Decomposition (SVD) (Kadhim et al., 2017). Furthermore, Glove would be used to better understand the global context for more complex sentiment analysis tasks, and FastText to deal with misspellings, rare typos, messy, and specialized text (Çano and Morisio, 2019). Furthermore, we would like to implement Bidirectional Encoder Representations from Transformers (BERT) using a bidirectional approach, considering both the left and right context of words in a sentence, instead of analyzing the text sequentially. BERT looks at all the words in a sentence simultaneously (Devlin et al., 2019).

A Tables

A.1 DANN model results

Class	Twitter				Amazon			
	Precision	Recall	F1-Score	Support	Precision	Recall	F1-Score	Support
Negative	0.560	0.400	0.460	1,037	0.720	0.750	0.730	2,516
Neutral	0.640	0.690	0.660	2,897	0.360	0.270	0.310	1,228
Positive	0.690	0.700	0.690	2,549	0.760	0.810	0.780	2,499
Accuracy			0.650	6,483			0.680	6,243
Macro Avg	0.630	0.600	0.610	6,483	0.610	0.610	0.610	6,243
Weighted Avg	0.640	0.650	0.640	6,483	0.660	0.680	0.670	6,243

Table 8: DANN classification report for Twitter and Amazon subsets showing precision, recall, and F1-score.

B Use of AI

We used AI tools of Chat GPT, Perplexity, and Copilot to help us in fixing Python errors, optimizing the running time of the code, organizing functions, making plots, and finding relevant research papers in some sections.

Examples:

1. ”Please find a relevant academic journal for TF-IDF and word embeddings”
2. ”Please fix the error of corrupted numpy for this block of code”
3. ”def remove_isolated_apostrophes(text): return re.sub(r”*’*”, ’ ’, text) does this remove the apostrophe from ”don’t?”
4. ”Compare the performance of CNN vs BERT in sentiment analysis”
5. ”What is the trade-off to remove stopwords?”
6. ”What is the typical accuracy attained in sentiment analysis? Compare binary vs ternary variables, and Twitter vs reviews data. Provide sources.”
7. ”How should I detect negation in sentiment analysis”
8. ”when to use latex cite vs citep”
9. What does the number of epochs measure
10. sh-5.1$ du -sh /home/pzs340/* show hidden

References

Abdulkhaliq, S. S. and Darwesh, A. M. (2020). Sentiment analysis using hybrid feature selection techniques. *UHD Journal of Science and Technology*, 4(1):29–40.

Abid, F., Alam, M., Yasir, M., and Chen, L. (2019). Sentiment analysis through recurrent variants latterly on convolutional neural network of twitter. *Future Generation Computer Systems*, 95.

Agarwal, A., Xie, B., Vovsha, I., Rambow, O., and Passonneau, R. J. (2011). Sentiment analysis of twitter data. In *Proceedings of the workshop on language in social media (LSM 2011)*, pages 30–38.

Ahuja, R., Chug, A., Kohli, S., Gupta, S., and Ahuja, P. (2019). The impact of features extraction on the sentiment analysis. *Procedia Computer Science*, 152:341–348.

Almeida, F. and Xexéo, G. (2023). Word embeddings: A survey.

Aslan, S. (2022). A novel tcnn–bi-lstm deep learning model for predicting sentiments of tweets about covid-19 vaccines. *Concurrency and Computation: Practice and Experience*, 34(28):e7387.

Baldwin, T., Cook, P., Han, B., Harwood, A., Karunasekera, S., and Moshtaghi, M. (2013). A support framework for linguistic processing of social media text. *Information Processing & Management*, 49(2):318–331.

Bishop, C. M. (2006). *Pattern Recognition and Machine Learning*. Springer.

Boureau, Y.-L., Ponce, J., and LeCun, Y. (2010). A theoretical analysis of feature pooling in visual recognition. In *Proceedings of the 27th International Conference on Machine Learning (ICML-10)*, pages 111–118. Omnipress.

Bridle, J. S. (1990). Probabilistic interpretation of feedforward classification network outputs, with relationships to statistical pattern recognition. In *Neurocomputing: Algorithms, Architectures and Applications*, pages 227–236. Springer.

Çano, E. and Morisio, M. (2019). Word embeddings for sentiment analysis: a comprehensive empirical survey. *arXiv preprint arXiv:1902.00753*.

Chen, W., Su, Y., Shen, Y., Chen, Z., Yan, X., and Wang, W. (2019). How large a vocabulary does text classification need? a variational approach to vocabulary selection. *arXiv preprint arXiv:1902.10339*.

Christen, P., Hand, D. J., and Kirielle, N. (2023). A review of the f-measure: Its history, properties, criticism, and alternatives. *ACM Comput. Surv.*, 56(3).

Cliche, M. (2017a). Bb twtr at semeval-2017 task 4: Twitter sentiment analysis with cnns and lstms. In *Proceedings of the 11th International Workshop on Semantic Evaluation (SemEval-2017)*, pages 573–580, Vancouver, Canada. Association for Computational Linguistics.

Cliche, M. (2017b). Bb_twtr at semeval-2017 task 4: Twitter sentiment analysis with cnns and lstms. *arXiv preprint arXiv:1704.06125*.

Derra, N. D. and Baier, D. (2020). Working in detail: How lstm hyperparameter selection influences sentiment analysis results. *Arch. Data Sci. Ser. A*, 6(1):10.

Devitt, A. and Ahmad, K. (2007). Sentiment polarity identification in financial news: A cohesion-based approach. In *Proceedings of the 45th Annual Meeting of the Association of Computational Linguistics (ACL)*, pages 984–991, Prague, Czech Republic.

Devlin, J., Chang, M.-W., Lee, K., and Toutanova, K. (2019). Bert: Pre-training of deep bidirectional transformers for language understanding. In *Proceedings of the 2019 conference of the North American chapter of the association for computational linguistics: human language technologies, volume 1 (long and short papers)*, pages 4171–4186.

Ganin, Y., Ustinova, E., Ajakan, H., Germain, P., Larochelle, H., Laviolette, F., Marchand, M., and Lempitsky, V. (2016). Domain-adversarial training of neural networks.

Garg, N. and Sharma, K. (2022). Text pre-processing of multilingual for sentiment analysis based on social network data. *International Journal of Electrical & Computer Engineering (2088-8708)*, 12(1).

Gholamalinezhad, H. and Khosravi, H. (2020). Pooling methods in deep neural networks, a review. *arXiv preprint arXiv:2009.07485*.

Glenn, A., LaCasse, P., and Cox, B. (2023). Emotion classification of indonesian tweets using bidirectional lstm. *Neural Computing and Applications*, 35(13):9567–9578.

Grandini, M., Bagli, E., and Visani, G. (2020). Metrics for multi-class classification: an overview.

Hastie, T., Tibshirani, R., and Friedman, J. (2009). *The Elements of Statistical Learning: Data Mining, Inference, and Prediction*. Springer, 2nd edition.

Hinojosa Lee, M. C., Braet, J., and Springael, J. (2024). Performance metrics for multilabel emotion classification: Comparing micro, macro, and weighted f1-scores. *Applied Sciences*, 14(21).

Hosmer Jr, D. W., Lemeshow, S., and Sturdivant, R. X. (2013). *Applied logistic regression*. John Wiley & Sons.

Huang, Z., Xu, W., and Yu, K. (2015). Bidirectional lstm-crf models for sequence tagging. *arXiv preprint arXiv:1508.01991*.

Kadhim, A. I., Cheah, Y.-N., Hieder, I. A., and Ali, R. A. (2017). Improving tf-idf with singular value decomposition (svd) for feature extraction on twitter. In *3rd international engineering conference on developments in civil and computer engineering applications*.

Kamyab, M., Liu, G., and Adjeisah, M. (2021). Attention-based cnn and bi-lstm model based on tf-idf and glove word embedding for sentiment analysis. *Applied Sciences*, 11(23):11255.

Kim, K. (2018). An improved semi-supervised dimensionality reduction using feature weighting: application to sentiment analysis. *Expert Systems with Applications*, 109:49–65.

Kim, Y. (2014). Convolutional neural networks for sentence classification. In *Proceedings of the 2014 Conference on Empirical Methods in Natural Language Processing (EMNLP)*, pages 1746–1751. Association for Computational Linguistics.

Kiranyaz, S., Avci, O., Abdeljaber, O., Ince, T., Gabbouj, M., and Inman, D. J. (2021). 1d convolutional neural networks and applications: A survey. *Mechanical systems and signal processing*, 151:107398.

Krichen, M. (2023). Convolutional neural networks: A survey. *Computers*, 12(8):151.

learn developers, S. (2024). sklearn.linear_model.logisticregression. Accessed: 2025-06-16.

LeCompte, T. and Chen, J. (2017). Sentiment analysis of tweets including emoji data. In *2017 International Conference on Computational Science and Computational Intelligence (CSCI)*, pages 793–798. IEEE.

LeCun, Y., Bengio, Y., and Hinton, G. (2015). Deep learning. *nature*, 521(7553):436–444.

LeCun, Y., Bottou, L., Bengio, Y., and Haffner, P. (1998). Gradient-based learning applied to document recognition. *Proceedings of the IEEE*, 86(11):2278–2324.

Li, D. and Qian, J. (2016). Text sentiment analysis based on long short-term memory. In *2016 First IEEE International Conference on Computer Communication and the Internet (ICCCI)*, pages 471–475. IEEE.

Li, Z., Liu, F., Yang, W., Peng, S., and Zhou, J. (2022). A survey of convolutional neural networks: Analysis, applications, and prospects. *IEEE Transactions on Neural Networks and Learning Systems*, 33(12):6999–7019.

Lin, Y.-C., Chen, S.-A., Liu, J.-J., and Lin, C.-J. (2023). Linear classifier: An often-forgotten baseline for text classification. *arXiv preprint arXiv:2306.07111*.

Liu, B. (2012). Sentiment analysis and opinion mining. volume 5.

Liu, C., Fang, F., Lin, X., Cai, T., Tan, X., Liu, J., and Lu, X. (2021). Improving sentiment analysis accuracy with emoji embedding. *Journal of Safety Science and Resilience*, 2(4):246–252.

Liu, D. C. and Nocedal, J. (1989). On the limited memory bfgs method for large scale optimization. *Mathematical Programming*, 45(1):503–528.

Mahadevaswamy, U. and Swathi, P. (2023). Sentiment analysis using bidirectional lstm network. *Procedia Computer Science*, 218:45–56.

Manning, C. D., Raghavan, P., and Schütze, H. (2008). *Introduction to Information Retrieval*. Cambridge University Press, Cambridge, UK.

Mehanna, Y. S. and Mahmuddin, M. (2021). The effect of pre-processing techniques on the accuracy of sentiment analysis using bag-of-concepts text representation. *SN Computer Science*, 2(4):237.

Mikolov, T., Chen, K., Corrado, G., and Dean, J. (2013a). Efficient estimation of word representations in vector space. *arXiv preprint arXiv:1301.3781*.

Mikolov, T., Sutskever, I., Chen, K., Corrado, G. S., and Dean, J. (2013b). Distributed representations of words and phrases and their compositionality. In *Advances in Neural Information Processing Systems 26 (NIPS 2013)*, pages 3111–3119.

Miyajiwala, A., Ladkat, A., Jagadale, S., and Joshi, R. (2022). On sensitivity of deep learning based text classification algorithms to practical input perturbations. *CoRR*, abs/2201.00318.

Nair, V. and Hinton, G. E. (2010). Rectified linear units improve restricted boltzmann machines. In *Proceedings of the 27th International Conference on Machine Learning (ICML-10)*, pages 807–814. Omnipress.

Nakov, P., Rosenthal, S., Kiritchenko, S., Mohammad, S. M., Kozareva, Z., Ritter, A., Stoyanov, V., and Zhu, X. (2016). Developing a successful semeval task in sentiment analysis of twitter and other social media texts. *Language Resources and Evaluation*, 50(1):35–65.

Nama, A. (2021). Understanding bidirectional lstm for sequential data processing. *Medium*.

Nandwani, P. and Verma, R. (2021). A review on sentiment analysis and emotion detection from text. *Social network analysis and mining*, 11(1):81.

Naseem, U., Razzak, I., and Eklund, P. W. (2021). A survey of pre-processing techniques to improve short-text quality: a case study on hate speech detection on twitter. *Multimedia Tools and Applications*, 80:35239–35266.

Neyman, J. (1934). On the two different aspects of the representative method: the method of stratified sampling and the method of purposive selection. *Journal of the Royal Statistical Society*, 97(4):558–625.

Ni, J., Li, J., and McAuley, J. (2019). Justifying recommendations using distantly-labeled reviews and fined-grained aspects. In *Proceedings of the 2019 Conference on Empirical Methods in Natural Language Processing (EMNLP)*.

Opitz, J. and Burst, S. (2021). Macro f1 and macro f1.

Palomino, M. A. and Aider, F. (2022). Evaluating the effectiveness of text pre-processing in sentiment analysis. *Applied Sciences*, 12(17):8765.

Pang, B., Lee, L., and Vaithyanathan, S. (2002). Thumbs up? sentiment classification using machine learning techniques. In *Proceedings of the 2002 Conference on Empirical Methods in Natural Language Processing (EMNLP 2002)*, pages 79–86. Association for Computational Linguistics.

Pedregosa, F., Varoquaux, G., Gramfort, A., Michel, V., Thirion, B., Grisel, O., Blondel, M., Prettenhofer, P., Weiss, R., Dubourg, V., et al. (2011). Scikit-learn: Machine learning in python. *Journal of Machine Learning Research*, 12:2825–2830.

Pennington, J., Socher, R., and Manning, C. D. (2014). Glove: Global vectors for word representation. In *Proceedings of the 2014 Conference on Empirical Methods in Natural Language Processing (EMNLP)*, pages 1532–1543, Doha, Qatar. Association for Computational Linguistics.

Pradha, S., Halgamuge, M. N., and Vinh, N. T. Q. (2019). Effective text data preprocessing technique for sentiment analysis in social media data. In *2019 11th international conference on knowledge and systems engineering (KSE)*, pages 1–8. IEEE.

Raunak, V. (2017). Simple and effective dimensionality reduction for word embeddings. *arXiv preprint arXiv:1708.03629*.

Reimers, N. and Gurevych, I. (2017). Optimal hyperparameters for deep lstm-networks for sequence labeling tasks. *arXiv preprint arXiv:1707.06799*.

Rudkowsky, E., Haselmayer, M., Wastian, M., Jenny, M., Emrich, Š., and Sedlmair, M. (2018). More than bags of words: Sentiment analysis with word embeddings. *Communication Methods and Measures*, 12(2-3):140–157.

Salehi, F., Abbasi, E., and Hassibi, B. (2019). The impact of regularization on high-dimensional logistic regression. *Advances in Neural Information Processing Systems*, 32.

Salloum, S. A., Alfaisal, R., Basiouni, A., Shaalan, K., and Salloum, A. (2024). Effectiveness of logistic regression for sentiment analysis of tweets about the metaverse. In *Breaking Barriers with Generative Intelligence: Proceedings of the International Conference on Breaking Barriers with Generative Intelligence (BBGI 2024)*, volume 2162 of *Lecture Notes in Networks and Systems*, pages 32–41. Springer.

Saraswati, N. W. S., Yanti, C. P., Muku, I. D. M. K., and Dewi, D. A. P. R. (2025). Evaluation analysis of the necessity of stemming and lemmatization in text classification. *MATRIK: Jurnal Manajemen, Teknik Informatika dan Rekayasa Komputer*, 24(2):321–332.

Sebastiani, F. (2002a). Machine learning in automated text categorization. *ACM Computing Surveys*, 34(1):1–47.

Sebastiani, F. (2002b). Machine learning in automated text categorization. *ACM computing surveys (CSUR)*, 34(1):1–47.

Shi, H., Li, X., Liu, H., and Zhu, L. (2016). Research on the attribute classification of sentiment target based on the stratified sampling. In *2016 12th International Conference on*

Natural Computation, Fuzzy Systems and Knowledge Discovery (ICNC-FSKD), pages 1180–1187. IEEE.

Srivastava, N., Hinton, G., Krizhevsky, A., Sutskever, I., and Salakhutdinov, R. (2014). Dropout: A simple way to prevent neural networks from overfitting. *Journal of Machine Learning Research*, 15(1):1929–1958.

Srujan, K., Nikhil, S., Raghav Rao, H., Karthik, K., Harish, B., and Keerthi Kumar, H. (2018). Classification of amazon book reviews based on sentiment analysis. In *Information Systems Design and Intelligent Applications: Proceedings of Fourth International Conference INDIA 2017*, pages 401–411. Springer.

Stalidis, P., Giatsoglou, M., Diamantaras, K., Sarigiannidis, G., and Chatzisavvas, K. C. (2015). Machine learning sentiment prediction based on hybrid document representation. *arXiv preprint arXiv:1511.09107*.

Sunny, M. A. I., Maswood, M. M. S., and Alharbi, A. G. (2020). Deep learning-based stock price prediction using lstm and bi-directional lstm model. In *2020 2nd novel intelligent and leading emerging sciences conference (NILES)*, pages 87–92. IEEE.

Symeonidis, S., Effrosynidis, D., and Arampatzis, A. (2018). A comparative evaluation of pre-processing techniques and their interactions for twitter sentiment analysis. *Expert Systems with Applications*, 110:298–310.

van der Maaten, L. and Hinton, G. E. (2008). Visualizing data using t-sne. *Journal of Machine Learning Research*, 9:2579–2605.

Wang, J., Peng, B., and Zhang, X. (2018). Using a stacked residual lstm model for sentiment intensity prediction. *Neurocomputing*, 322:93–101.

Wankhade, M., Rao, A. C. S., and Kulkarni, C. (2022). A survey on sentiment analysis methods, applications, and challenges. *Artificial Intelligence Review*, 55:5731–5780.

Wardana, N. S., Aditiawan, F. P., and Sari, A. P. (2024). Logistic regression classification with tf-idf and fasttext for sentiment analysis of linkedin reviews. *VISA: Journal of Vision and Ideas*, 4(3):1359–.

Welch, C., Mihalcea, R., and Kummerfeld, J. K. (2020). Improving low compute language modeling with in-domain embedding initialisation. *arXiv preprint arXiv:2009.14109*.

Zhan, Z. (2025). Comparative analysis of tf-idf and word2vec in sentiment analysis: A case of food reviews. In *ITM Web of Conferences*, volume 70, page 02013. EDP Sciences.

Zhang, L., Wang, S., and Liu, B. (2018). Deep learning for sentiment analysis: A survey. *Wiley Interdisciplinary Reviews: Data Mining and Knowledge Discovery*, 8(4):e1253.