Aalborg University

10th semester

Mathematics-Economics

Master's thesis

The Impact of Sentiment on Stock Price Movements:
A WallStreetBets Case Study

1st of November 2024

AALBORG UNIVERSITY

STUDENT REPORT

AALBORG UNIVERSITY
STUDENT REPORT

Department of Mathematical Sciences
Skjernvej 4A
DK-9220 Aalborg Ø
http://math.aau.dk

Title

The Impact of Sentiment on Stock Price Movements: A WallStreetBets Case Study

Project

Master's thesis

Project period

1st September 2024 - 1st November 2024

Project group

Group 4.111c_2

Participants

Oliver Christensen

Supervisors

Sauri Arregui Orimar

Number of pages

66

Date of Completion

November 1, 2024

Abstract:

This study investigates the impact of sentiment on stock price movements using data from the WallStreetBets (WSB) subreddit. Utilizing natural language processing techniques, we extracted and analyzed sentiment in WSB posts and examined its correlation with historical stock prices. Various machine learning models were employed to explore their effectiveness in predicting stock movements. The results suggest that sentiment analysis can provide valuable insights into financial market behavior, demonstrating its potential utility in stock price prediction.

Preface

The project period is from the 1st of September 2024 to the 1st of November 2024, and is written by a 10th semester student studying Mathematics-Economics at Aalborg University. It is assumed that the reader is at least a 10th semester student on a mathematical orientated education and therefore has knowledge about, among other things, probability theory and basic measure theory.

References is written in the beginning of each section and refers to the bibliography. References are written according to the Vancouver method, and are therefore written as: [Number, page].

Signatures

Oliver Christensen

Oliver Christensen
ochri19@student.aau.dk

Nomenclature

$\lvert\cdot\rvert$	Cardinality (sets and vectors: number of elements; matrices: number of rows)
$\boldsymbol{d}$	Encoded feature vector of a document
$\mathcal{D}$	Corpus - a vector of documents
$\mathcal{T}$	Tokenization function
D	Document-Term Matrix
d	Document
S_E	Entropy
S_G	Gini Impurity
t	Token
V	Vocabulary
IG	Information Gain

Contents

1 Introduction

Sentiment analysis has emerged as a powerful tool in the domain of financial markets, providing insights into public opinion and emotional trends that can significantly influence stock prices. This project aims to leverage sentiment analysis on posts from WallStreetBets (WSB) to explore the potential of using online sentiment to predict future stock price movements. WallStreetBets is a popular subreddit on the social news and discussion website, Reddit. It is known for its high-risk, high-reward trading strategies and its often irreverent, meme-filled commentary on stock market investments. The community primarily consists of retail investors who share their experiences, strategies, and opinions on various stocks and financial markets.

The rationale behind this project lies in the premise that collective sentiment, particularly within highly active and influential online communities like WSB, can drive substantial market movements. For instance, the dramatic rise of GameStop (GME) stock in early 2021 serves as a prime example. This event was heavily influenced by the user ”DeepFuckingValue” as his posts garnered widespread attention and support, contributing to an explosive surge in the stock price. This case illustrates how sentiment and social media buzz can lead to significant market events, including so-called ”pump and dump” schemes, where coordinated buying and selling can manipulate stock prices.

By utilizing natural language processing, we can quantify the emotional tone and overall sentiment of the community towards specific stocks. This information can then be integrated into a machine learning algorithm to analyze and predict stock price movements. The algorithm will, among other things, use historical sentiment data and corresponding stock prices to learn patterns and correlations that may indicate future price changes.

The ultimate goal of this project is to determine the viability of sentiment analysis as a predictive tool for financial markets. By understanding how sentiment drives stock movements, investors and analysts can gain a valuable edge in making informed trading decisions and identifying potential market manipulations before they occur.

1.1 Problem Statement

Can sentiment signals within the WallStreetBets community be used to predict future stock price movements, and how effectively can this information be incorporated into a machine learning algorithm to predict future stock market movements?

2 | Natural Language Processing

Natural Language Processing (NLP) is a multidisciplinary field that combines linguistics, computer science, and artificial intelligence to enable machines to understand and interpret human language. In the context of sentiment analysis, NLP techniques are used to assess and categorize emotions expressed in text, allowing for the extraction of insights from opinions, reviews, and social media interactions. To leverage these techniques, it is important to establish a clear understanding of several key concepts foundational to NLP and sentiment analysis.[1]

Definition 2.1.

1. **Vocabulary**: The complete and ordered set of unique tokens in the corpus.
2. **Token**: A unit of text that has been extracted from a larger body of text. In particular, we define a token to be the smallest unit of analysis. Examples include words, characters, and expressions that consist of multiple words or characters. A tokenization function $\mathcal{T}$ is a function that can split a document of any length into vector of any dimension depending on the type of token chosen.
3. **n-gram**: A contiguous sequence of n tokens from a given text. Examples include *unigrams* ($n = 1$) and *bigrams* ($n = 2$).
4. **Document**: A sentence or passage of text.
5. **Corpus**: A collection of documents. We denote this by the vector, $\mathcal{D}$.
6. **Feature Vector**: A numerical representation of some observation. Each value corresponds to a different feature of the data point being represented. For textual features, we define $\boldsymbol{d} \in \mathbb{R}^m,\ m \leq |\boldsymbol{v}|$ as the feature vector representation of a document d, where $\boldsymbol{v}$ is the vocabulary. Each component indicates a measure of importance, frequency, or weight for the corresponding item in $\boldsymbol{v}$.
7. **Document-Term Matrix**: A matrix representation of a corpus. A Document-Term Matrix, $D \in \mathbb{R}^{n \times m}$ consists of feature vectors, where n is the number of feature vectors and the size of each feature vector (m) is constant.

Having established these foundational definitions, we can delve deeper into the essential processes that prepare textual data for analysis.

2.1 Preprocessing

Preprocessing is a crucial step in preparing textual data for sentiment analysis. It ensures that raw data is cleaned, structured, and enriched to facilitate the application of computational methods. Below are the key preprocessing steps involved in transforming textual data into a usable format. [1]

2.1.1 Text Cleaning

Raw textual data often comes in unstructured formats, containing unnecessary elements such as HTML tags, special characters, or irrelevant symbols. The process of *parsing* extracts the actual text from files, removing unwanted elements. Depending on the source, this process can be simple or tedious. After parsing, extraneous elements, including stop words such as ”the” or ”and”, are removed. Stop words contribute little to the sentiment analysis and add noise, so removing them helps reduce dimensionality which is inherently very high in textual data. In fact there are q^w unique representations of a text with w words drawn from a vocabulary consisting of q words.[2] The number of unique grammatically correct sentences, however, is significantly less.

Example 2.2. $d_1 =$ ”~~The~~ quick brown fox jumped ~~over~~ ~~the~~ lazy dog~~!~~”

2.1.2 Stemming and Lemmatization

In natural language, many words are either synonymous or simply different inflections of the same root word. This presents an opportunity to reduce the dimensionality of our vocabulary. The goal is to simplify and normalize all words within each group of similar or synonymous words to their respective base forms. What constitutes the base form of a word is not exactly trivial. Examples of text normalization techniques include *stemming* as well as *lemmatization.*

Stemming

Stemming is a process that reduces words down to their root form by removing common suffixes and prefixes. For example, words like “running,” “runner,” and “ran” would all be reduced to the stem “run”. Stemming is strictly rule-based and is context independent. While computationally efficient, this has the disadvantage of not capturing the semantic relationship with words such as ”better” and ”good”. Further, stemming does not always produce valid words, leading to loss of meaning.

Lemmatization

Lemmatization is a more sophisticated process than stemming. It reduces words to their base or dictionary form, known as the *lemma*, by considering the context and part of speech of a word. Unlike stemming, lemmatization produces valid words and is capable of capturing the semantic relationship between words. For example, the words “better” and “good” share a lemmatized form of “good.”

While lemmatization is more accurate in preserving meaning, it is computationally more expensive than stemming, as it relies on a deeper understanding of the grammatical context and requires a pre-built lexical database such as *WordNet*.

Example 2.3. Continuing Example 2.2, both stemming and lemmatization produces the same output.
$d_1 =$ ”quick brown fox jump$\cancel{ed}$ lazy dog”

2.1.3 Tokenization

Once each document has been cleaned, they are still single objects. It is very unlikely that a document of text exists more than once and therefore, it would be impossible for the machine to learn anything based on entire documents. The same way humans consider the meaning of each word seperately, it is necessary to seperate the sentence or text passage into tokens in order to establish meaning behind each word through machine learning. Text can also be split sub-words, characters or even n-grams. Sub-word tokenization enables the computer to recognize words within words, which may uncover more relationships between documents than would have otherwise been achieved. Likewise, using e.g. bigrams makes it possible for the machine to recognize which words often go together. For example, the word ”learning” is more likely to be contiguous to ”machine” than many other words.

Example 2.4. Continuing Example 2.2, we tokenize on a word-level.

$$\mathcal{T}(d_1) = \begin{bmatrix} \text{”quick”} & \text{”brown”} & \text{”fox”} & \text{”jump”} & \text{”lazy”} & \text{”dog”} \end{bmatrix}$$

2.1.4 Feature Extraction

Any method intended for machine learning or statistical processing must eventually convert text into numerical data. This is typically achieved using a *document-term matrix* (DTM), where each row represents a document, each column represents a term, and the values in the cells represent the (weighted) frequency of occurence of each term in a document[1]. A simple way to encode a document is by simply marking each word from the vocabulary by 1 if it exists in the document and 0 otherwise. To illustrate, consider the following example.

Example 2.5. We now add some extra documents to show the structure of the DTM.

$$d_1 = \text{"quick brown fox jump lazy dog"} \qquad d_3 = \text{"fox jump"}$$
$$d_2 = \text{"brown fox quick"} \qquad d_4 = \text{"dog lazy"}$$

The corpus can then be represented by the following DTM.

	Vocabulary						
Feature Vectors	"quick"	"brown"	"fox"	"jump"	"lazy"	"dog"	
$\boldsymbol{d}_1$	1	1	1	1	1	1	1
$\boldsymbol{d}_2$	1	1	1	0	0	0	0
$\boldsymbol{d}_3$	0	0	1	1	0	0	0
$\boldsymbol{d}_4$	0	0	0	0	0	1	1

Table 2.1: Document-term matrix

The type of encoding is also known as *one-hot* encoding[3]. More formally, the one-hot encoding function is the indicator function,

$$\mathbb{1}_d(t) := \begin{cases} 1 \text{ if } t \in d \\ 0 \text{ otherwise.} \end{cases}$$

Thus, the feature vectors are given by $\boldsymbol{d}_i = [\mathbb{1}_{d_i}(t)]_{t \in \boldsymbol{v}}$ for $i \in \{1, \ldots, |\mathcal{D}|\}$. This ensures, that each token has a unique representation so there is no overlap between tokens. The dimensionality of the one-hot encoded vector equals the size of the vocabulary, which can become very large for extensive vocabularies, leading to sparsity. For large vocabularies, the DTM is typically converted to some compressed sparse matrix format.

Another technique for feature extraction is what is known as *term frequency-inverse document frequency*(TF-IDF), which assigns weights to terms based on their frequency across documents, thus emphasizing terms that are more specific to individual documents. TF-IDF is the product of two statistics: Term Frequency (TF) and Inverse Document Frequency (IDF). Term Frequency measures how frequently a term (or token) appears in a document, normalized to prevent bias towards longer documents [4]:

$$\text{TF}(t, d) = \frac{f(t, d)}{|\mathcal{T}(d)|} \tag{2.1}$$

where $f(t, d) := \sum_{t' \in T(d)} \mathbb{1}_{t'=t}$ is the frequency of term t in document d and $|\mathcal{T}(d)|$ is the number of elements in the tokenized document.

Inverse Document Frequency (IDF) measures how important a term is within the entire corpus:

$$\text{IDF}(t, \mathcal{D}) = \ln\left(\frac{|\mathcal{D}|}{|\{d \in \mathcal{D} : t \in \mathcal{T}(d)\}|}\right) \tag{2.2}$$

where $|\mathcal{D}|$ is the total number of documents in the corpus $\mathcal{D}$, and $|\{d \in \mathcal{D} : t \in \mathcal{T}(d)\}|$ is the number of documents containing the term t.

The TF-IDF score for a term t in a document d is calculated as:

$$\text{TF-IDF}(t, d, \mathcal{D}) = \text{TF}(t, d) \times \text{IDF}(t, \mathcal{D}). \tag{2.3}$$

In this case, feature vectors are calculated as $\boldsymbol{d}_i = [\text{TF-IDF}(t, d_i, \mathcal{D})]_{t \in \boldsymbol{v}}$ for $i \in \{1, \dots, |\mathcal{D}|\}$.

Example 2.6. Further extending Example 2.5, we can calculate the TF and IDF scores for each document.

$$|\mathcal{T}(d_1)| = 6 \qquad |\mathcal{T}(d_2)| = 3 \qquad |\mathcal{T}(d_3)| = |\mathcal{T}(d_4)| = 2 \qquad |c| = 4$$

Since each word appears exactly once, the term frequencies happen to be equal document-wise. Note that this is not the general case.

$$\text{TF}(t, d_i) = \frac{1}{|\mathcal{T}(d_i)|} \quad \forall (t, i) \in \{(t, i) \mid t \in \mathcal{T}(d_i),\, i \in \{1, \dots, 4\}\}$$

The inverse document frequencies are independent of the individual document content.

$$\text{IDF}('fox', \mathcal{D}) = \ln\left(\frac{4}{3}\right) \approx 0.2877$$
$$\text{IDF}(t, \mathcal{D}) = \ln 2 \approx 0.6931 \quad \forall t \in \{\text{"quick", "brown", "jump", "lazy", "dog"}\}$$

By multiplying the each TF and IDF, we arrive at the following DTM.

	Vocabulary					
Feature Vectors	"quick"	"brown"	"fox"	"jump"	"lazy"	"dog"
$\boldsymbol{d}_1$	0.116	0.116	0.0480	0.116	0.116	0.116
$\boldsymbol{d}_2$	0.2310	0.2310	0.0959	0	0	0
$\boldsymbol{d}_3$	0	0	0.1438	0.3465	0	0
$\boldsymbol{d}_4$	0	0	0	0	0.3465	0.3465

Table 2.2: Document-term matrix for TF-IDF encoding

2.1.5 Metadata Enrichment

Beyond text cleaning and feature extraction, adding metadata to the text provides valuable context for sentiment analysis. Metadata such as the publication date, author, source, or even geographic location can be important for refining the analysis.[1]

For example, using *Named Entity Recognition (NER)* techniques can help extract key entities (such as companies, locations, or individuals) mentioned in the text. Enriching the dataset with this additional information allows for a more sophisticated analysis of sentiment trends associated with particular entities or time periods.

2.2 Dimensionality Reduction

Textual data are inherently high-dimensional. Structuring a text of length w words, each drawn from a vocabulary of q possible words, yields unique representations with a dimensionality of q^w [2]. As the document-term matrix is often large and sparse, *dimensionality reduction* techniques are applied to reduce the number of features while retaining the most informative aspects of the text. Techniques such as *Principal Component Analysis (PCA)* or *Singular Value Decomposition (SVD)* help reduce computational complexity and improve the performance of sentiment analysis models.

2.2.1 Principal Component Analysis (PCA)

In this section, we describe how Principal Component Analysis (PCA) can be utilized to reduce dimensionality[5]. The first step in PCA is center the DTM, $D \in \mathbb{R}^{n\times m}$. We define,

$$\bar{\boldsymbol{d}} := \frac{1}{n}\sum_{i=1}^{n} \boldsymbol{d}_i$$

and the centered DTM is given by

$$D_c := D - \bar{\boldsymbol{d}}$$

Next, the covariance matrix is computed as:

$$\text{Cov} = \frac{1}{N} D_c^\top D_c \in \mathbb{R}^{m\times m}.$$

From the covariance matrix Cov, we compute the eigenvalues $\lambda_1, \lambda_2, \dots, \lambda_m$ and the corresponding eigenvectors $\boldsymbol{v_1}, \boldsymbol{v_2}, \dots, \boldsymbol{v_m}$. The eigenvalues are sorted in descending order, and we select the top m' eigenvalues, denoted as $\lambda_{\max 1}, \lambda_{\max 2}, \dots, \lambda_{\max m'}$, where $m' < m$. This essentially selects the m' features that contribute the most to the variance. The explained variance is calculated using the formula $\frac{\lambda_i}{\sum_j^m \lambda_j}$. By plotting the cumulative explained variance against the number of principal components, one can look for an "elbow" point. This point indicates where adding more principal components results in diminishing returns.

The matrix W is constructed using the selected eigenvectors as its rows:

$$W = \begin{bmatrix} \boldsymbol{v}_{\mathbf{max\,1}} \\ \boldsymbol{v}_{\mathbf{max\,2}} \\ \vdots \\ \boldsymbol{v}_{\mathbf{max}\,\boldsymbol{d'}} \end{bmatrix} \in \mathbb{R}^{m' \times m}.$$

The final step involves projecting the original feature vectors $\boldsymbol{d_i}$ into the reduced dimensional space. This is achieved by computing the projected vector $\boldsymbol{z_i}$ for each document:

$$\boldsymbol{z_i} = W(\boldsymbol{d_i} - \bar{\boldsymbol{d}})^\top.$$

The set of projected examples is defined as $D_p := \begin{bmatrix} \boldsymbol{z_1} & \boldsymbol{z_2} & \dots & \boldsymbol{z_n} \end{bmatrix}^\top$, which represents the reduced dimensional representations of the original documents.

PCA serves as a powerful technique for dimensionality reduction, particularly useful in applications involving high-dimensional feature vectors derived from text data. However, it is important to note that the computation of the covariance matrix and the subsequent eigenvalue decomposition can be computationally intensive, particularly as the number of training examples n grows, resulting in quadratic complexity $O(N^2)$. For this reason, PCA may not always be the most efficient choice for text mining tasks where the dimensionality of the data can be extremely large.

2.2.2 Singular Value Decomposition (SVD)

The Singular Value Decomposition (SVD) is the process of decomposing a given matrix into three matrices: one matrix whose column vectors are the eigenvectors, a diagonal matrix whose diagonal elements are the eigenvalues, and another matrix whose row vectors are also eigenvectors.[6] The SVD is utilized for reducing the dimensionality of numerical vectors that represent data items. Consider n training examples, each represented as an m-dimensional vector. To perform SVD on a particular the DTM:

1. Compute the n eigenvectors from the $n \times n$ matrix $DD^\top$ and construct the $n \times n$ matrix U by arranging the eigenvectors as its column vectors. The corresponding eigenvalues must be ordered in descending magnitude.

2. Compute the n eigenvalues from the eigenvectors of the matrix $DD^\top$ and construct the $n \times n$ diagonal matrix Σ by arranging the singular values (the square roots of the eigenvalues) in descending order along its diagonal.

3. Compute the m eigenvectors from the $m \times m$ matrix $D^\top D$ and construct the $m \times m$ matrix V by arranging the eigenvectors as its row vectors. The arrangement of the columns of V must correspond to the order of the singular values in Σ.

The data matrix representing the training examples, D, is decomposed into the three matrices as follows:

$$D = U\Sigma V^T \tag{2.4}$$

This decomposition is referred to as Singular Value Decomposition (SVD).

Training examples with reduced dimensions are generated by selecting the top m' eigenvalues from the matrix Σ, where $m' < m$ and only the maximum m' eigenvalues are retained from Σ, while the remaining entries are filled with zeros. The matrix D', representing the training examples with reduced size, is constructed as follows:

$$D' = U\Sigma' V^T \tag{2.5}$$

In the matrix D', the m' non-zero column vectors are retained, and the $m - m'$ zero column vectors are omitted.

In summary, the following remarks can be made about SVD: Understanding the eigenvectors and eigenvalues of a matrix forms the basis for SVD. The SVD decomposes a matrix into three matrices consisting of its eigenvectors and eigenvalues, as shown in Equation (2.68). In Equation (2.68), U and V contain the eigenvectors arranged as column and row vectors, respectively, while Σ is the diagonal matrix with eigenvalues as its diagonal elements. Dimensionality reduction using SVD is achieved by removing the smallest eigenvalues from the matrix Σ.

2.3 Part-of-speech tagging

Part-of-speech (POS) tagging involves assigning grammatical categories to individual words in a sentence. These categories typically include nouns, verbs, adjectives, adverbs, pronouns, prepositions, conjunctions, and interjections. POS tagging is essential for understanding the syntactic structure of sentences and can significantly enhance various NLP applications, such as machine translation, information retrieval, and sentiment analysis.[7]

Importance of POS Tagging Syntactic Analysis: POS tagging provides crucial information about the syntactic role of words within a sentence, enabling more accurate parsing and understanding of sentence structure. Word Sense Disambiguation: Many words can have multiple meanings depending on their context. POS tagging helps disambiguate these meanings by considering the grammatical role of the word in the sentence. Feature Extraction: In many NLP tasks, features derived from POS tags can improve the performance of machine learning models. For example, distinguishing between nouns and verbs can help classify text more effectively. Semantic Analysis: Understanding the role of words in context can aid in extracting semantic relationships and meaning from text, leading to better insights and analysis. Several techniques are used for POS tagging, each with its strengths and weaknesses[7]:

- **Rule-Based Tagging**: Early POS taggers relied on a set of handcrafted rules to assign tags based on the context of words. While effective for specific tasks, this approach is often limited by the complexity of language and the extensive number of exceptions.

- **Statistical Tagging**: These methods, such as Hidden Markov Models (HMMs), utilize probabilities derived from large annotated corpora to predict the most likely tag sequence for a given sentence. Statistical methods typically outperform rule-based approaches but require significant training data.
- **Machine Learning Approaches**: Modern POS tagging often employs machine learning algorithms, such as Conditional Random Fields (CRFs) and neural networks. These methods can learn complex patterns in the data and generalize better to unseen examples, making them highly effective for various languages and domains.
- **Deep Learning Models**: With the rise of deep learning, recurrent neural networks (RNNs) and transformers (like BERT) have become popular for POS tagging tasks. These models can capture long-range dependencies and contextual information, resulting in state-of-the-art performance on POS tagging benchmarks.

2.4 Word Embeddings

Word embeddings are a type of word representation that allows words to be represented as dense vectors in a continuous vector space. Unlike traditional one-hot encoding, which represents words as sparse vectors with a length equal to the size of the vocabulary, word embeddings capture semantic meanings and relationships between words by positioning them in a high-dimensional space. This approach enables models to understand context, similarity, and relationships among words more effectively, making embeddings a fundamental concept in natural language processing (NLP) and machine learning. [8]

Word embeddings capture the meaning of words based on their context in a large corpus of text. Words that appear in similar contexts are positioned closer together in the vector space, allowing for a quantitative measure of semantic similarity. By representing words in a lower-dimensional space, word embeddings reduce the computational complexity associated with processing high-dimensional categorical data. This compact representation is particularly beneficial for large vocabularies. Many NLP tasks, such as text classification, sentiment analysis, and machine translation, benefit from the rich representations provided by word embeddings. They have been shown to improve the performance of machine learning models compared to traditional methods. Pre-trained word embeddings can be used across various tasks and domains, allowing models to leverage knowledge gained from large datasets. This transferability reduces the need for extensive labeled data in new tasks.

Common Techniques for Generating Word Embeddings

Several techniques are widely used to generate word embeddings, each with its approach and underlying methodology[9]:

- Word2Vec: Developed by Google, Word2Vec uses neural networks to learn word representations from large text corpora. It offers two architectures:
- Continuous Bag of Words (CBOW): Predicts a target word based on its surrounding context words.
- Skip-Gram: Predicts surrounding context words given a target word. This method has been effective in capturing fine-grained semantic relationships.
- GloVe (Global Vectors for Word Representation): Developed by Stanford, GloVe is based on matrix factorization techniques. It constructs a global word-word co-occurrence matrix from a corpus and derives embeddings by factorizing this matrix. GloVe captures both local and global statistical information, resulting in meaningful vector representations.
- FastText: Created by Facebook, FastText improves upon Word2Vec by representing words as bags of character n-grams. This approach allows it to generate embeddings for out-of-vocabulary words by combining the embeddings of their constituent n-grams. FastText is particularly useful for handling morphologically rich languages.
- BERT and Contextualized Embeddings: Unlike traditional static embeddings, models like BERT (Bidirectional Encoder Representations from Transformers) generate contextualized word embeddings. Each word's representation depends on the entire sentence context, allowing for more nuanced understanding and improved performance on various NLP tasks.

3 | Machine Learning

Machine learning (ML) has emerged as a powerful tool for natural language processing (NLP), revolutionizing the way we analyze and understand human language. NLP encompasses a broad set of tasks, from simple text classification to complex language generation, and machine learning techniques allow computers to automatically learn patterns in data, making it possible to automate and enhance many of these tasks.

In NLP, machine learning algorithms can process large volumes of text, recognize patterns in language, and generate insights that would be difficult or time-consuming for humans to produce manually. By leveraging models that learn from data, machine learning enables applications such as sentiment analysis, text classification, machine translation, speech recognition, and even chatbots.

Unlike rule-based approaches, where predefined instructions are needed to process language, ML models, particularly deep learning models like neural networks, can learn to interpret language from data alone. This adaptability makes machine learning a critical tool for analyzing unstructured data like text, where the nuances of human communication, such as slang, tone, and context, can be challenging to define with static rules.

From detecting trends in social media discussions to powering personal assistants like Siri or Alexa, machine learning has opened up new possibilities in NLP, enabling machines to better understand and respond to human language in a meaningful way.

3.1 Supervised Learning

Supervised learning is a foundational approach in machine learning, particularly useful for natural language processing tasks. In supervised learning, models are trained on labeled datasets, where each input (such as a sentence or a document) is paired with its corresponding output or label (such as a sentiment classification or topic label). The model learns to map the input to the correct output by minimizing the error between its predictions and the actual labels, eventually becoming capable of generalizing to unseen data.[10]

In NLP, supervised learning algorithms are frequently applied to tasks such as Sentiment Analysis, Text Classification, Named Entity Recognition and Part-of-Speech Tagging.

The success of supervised learning in NLP largely depends on the quality and quantity of labeled training data. Models like support vector machines (SVMs), logistic regression, and more recently, neural networks, are commonly used in these tasks. With the advent of deep learning, complex models like recurrent neural networks (RNNs), convolutional neural networks (CNNs), and transformers (such as BERT, GPT) have achieved state-of-the-art results across various supervised NLP tasks.

Despite its power, supervised learning comes with the challenge of requiring large, high-quality datasets. For many languages and domains, labeled data may be scarce or expensive to obtain. Additionally, models trained using supervised learning are often domain-specific, meaning they might not perform well when applied to different types of text than those they were trained on.

3.2 Unsupervised learning

Unlike supervised learning, unsupervised learning trains algorithms on data without the use of labeled responses. There are several methods of achieving this, such as *clustering* and *association rule learning.* [10]

Unsupervised learning is a branch of machine learning that deals with training algorithms on data without labeled responses. Unlike supervised learning, where the model learns from a dataset that contains input-output pairs, unsupervised learning seeks to find patterns and structures within data without explicit guidance. This approach is particularly useful in situations where labeled data is scarce or unavailable.

A technique the also falls under the umbrella of unsupervised learning is what is known as PCA (Principal Component Analysis). This is useful for reducing the dimensionality of the feature space and can thus also be used in conjunction with supervised learning. So while this project largely pertains the use of supervised learning algorithms, we also employ some unsupervised learning techniques.

3.3 Feature Engineering

Feature engineering is a crucial step in the machine learning pipeline that involves selecting, modifying, or creating features to improve model performance. Traditionally, this process required domain expertise to handcraft features based on prior knowledge and intuition about the data. Commonly used techniques included transforming raw data into a format suitable for modeling, extracting relevant metrics, and identifying key variables that could influence predictions.

Recent advancements in machine learning, particularly with the rise of neural networks, have shifted the focus toward automatic learning. In this paradigm, models can automatically learn complex features directly from raw data, minimizing the need for extensive manual feature engineering. For instance, deep learning architectures, such as convolutional neural networks (CNNs) and recurrent neural networks (RNNs), automatically extract hierarchical features from images and sequential data, respectively. This capability not only reduces the time and effort spent on feature design but also enables the discovery of intricate patterns that may be overlooked in traditional approaches [11, 12, 13].

Automatic learning can be applied in both supervised and unsupervised contexts. In supervised learning, models learn to associate input features with labeled outputs, optimizing their performance through backpropagation. Conversely, in unsupervised learning, models analyze unlabeled data to uncover hidden structures, employing techniques such as clustering and dimensionality reduction [14, 15]

The ability of neural networks to perform automatic feature learning has led to significant advancements across various domains, including natural language processing (NLP), image recognition, and speech analysis, ultimately contributing to more robust and accurate models [16, 17]

3.4 Tree-based Models

Tree-based models use a tree-like structure to make decisions based on feature values, making them effective for both classification and regression tasks. Their intuitive design allows for easy interpretation and visualization.

A pre-cursor to Random Forests, Decision Trees play a crucial rule in many text classification schemes.

3.4.1 Decision Trees

A Decision Tree classifier builds a tree structure where each node represents a decision based on the value of a specific feature, leading to branches that represent the possible outcomes of that decision. The following figure depicts an example of such a tree:

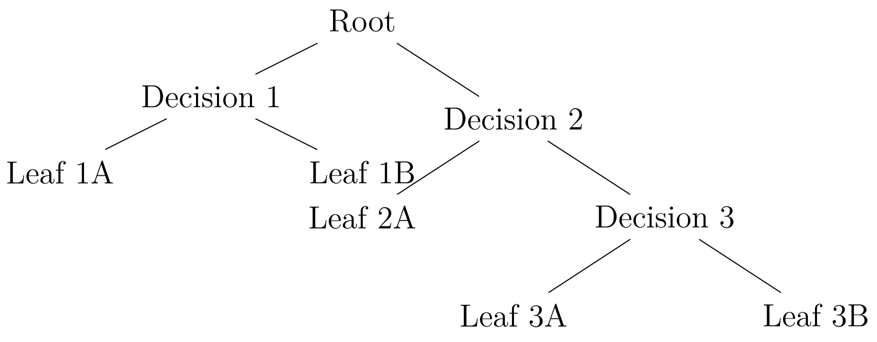


Figure 3.1: Example decision tree

- **Root Node**: The topmost node that represents the entire dataset.
- **Decision Nodes**: Intermediate nodes that represent decisions based on specific feature values.

- **Leaf Nodes**: Terminal nodes that represent the final classification or output. Each leaf node corresponds to a specific class label.

Feature Selection and Splits

Feature selection at each node involves selecting the feature that best splits the data according to a chosen splitting criterion, S. Common criteria include:

- **Gini Impurity**: An impurity measure focusing on the probability of misclassification when selecting two instances at random. It is represented as a value between 0 and 0.5, where 0 indicates that all samples fall into the same class, and 0.5 signifies that the samples are evenly distributed among all classes. Let C denote the set of unique classes. The measure is calculated as follows:[15]

$$S_G(D) = 1 - \sum_{k \in C} P(k)$$

  where $P(k) := \frac{|D^{(k)}|}{|D|}$ is the proportion of class k in the Document-Term Matrix D.

- **Entropy**: This measure quantifies the amount of uncertainty or disorder in a dataset based on the distribution of classes. It is calculated as follows:[15]

$$S_E(D) = -\sum_{k \in C} p(k) \log_2 p(k).$$

To decide the best split, we need to maximize *Ínformation Gain*, which is the (hypothetical) reduction in impurity (whether based on entropy or Gini impurity) after a split. Information gain, IG, is calculated as the difference between the impurity of the parent node and the weighted impurity of the child nodes resulting from the split.

For a corpus D split by a decision rule δ, the information gain $IG(D, \delta)$ is defined as:

$$\mathrm{IG}(D, \delta) = S(D) - S(D|\delta),$$

where the conditional impurity $S(D|\delta) = \sum_{i \in C} \frac{|D^{(i|\delta)}|}{|D|} S(D^{(i|\delta)})$ and $D^{(i|\delta)}$ denotes the subset of documents in D that fall into the i-th category after the split based on decision rule δ, and $S(D^{(i)})$ represents the impurity measure (either Gini or entropy) of the child node $D^{(i)}$. For one-hot encoded data, the decision rules can be seen as presence or absence of a word $\delta = \{D_{ij} = t\}$, whereas for frequency based encodings such as TF-IDF, decision rules are on the form $\delta = \{D_{ij} < t\}$ **or** $\delta = \{D_{ij} >= t\}$, where $t \in \mathbb{R}$ is some threshold and D_{ij} is some entry in the DTM, D.

Decision Process and Thresholds

At each decision node, the classifier evaluates all available features and their corresponding thresholds to determine the best split. The process follows these steps[15]:

1. **Determine Unique Values**: For the selected feature, identify its unique values.
2. **Generate Possible Thresholds**: If a feature has multiple unique values, thresholds can be set at:
   - Each unique value. (Categorical features)
   - Midpoints between unique values. (Continuous features)

   For example, if the feature is the TF-IDF score for a word, and the unique scores are $[0.0, 0.2, 0.4, 0.5, 0.8]$, possible thresholds would include 0.1, 0.3, 0.45, and 0.65.
3. **Evaluate Each Threshold**: For each threshold, the data is split into groups based on the decision rules (e.g., greater/less than or equal/not equal to the thresholds), and the Gini impurity or Information Gain is calculated for each group.
4. **Choose the Best Split**: The decision rule that maximizes Information Gain is chosen.

3.4.2 Random Forest

Random Forest is an ensemble learning method that combines multiple decision trees to improve classification accuracy and robustness. Each tree in the forest independently classifies data, and the Random Forest aggregates these predictions to provide a more accurate and generalized outcome than any single tree.

Random Forest Architecture

A Random Forest model consists of an ensemble of decision trees, each built on a different subset of data and a random subset of features. This process, known as **Bagging** (Bootstrap Aggregating), helps mitigate the variance of individual trees, leading to a more stable and accurate classifier. The architecture of Random Forest involves: [10]

- **Bootstrapped Sampling**: Each decision tree in the forest is trained on a random subset of the training data, typically chosen with replacement, meaning that some instances may appear multiple times, while others may not appear at all.
- **Feature Randomization**: To further reduce correlation between trees, a random subset of features is selected at each split in a tree, ensuring that no single feature overly dominates the classification outcome.
- **Voting Mechanism**: In classification, each tree in the forest casts a “vote” on the class label, and the final prediction is made based on the majority vote among all trees. For binary classification, the final class label is typically the one that receives more than 50% of the votes.

Training and Prediction in Random Forest

The Random Forest classifier trains each tree independently, allowing for highly parallelized training and prediction processes. Training and prediction follow these steps: [10]

1. **Generate Bootstrapped Samples**: For each decision tree, a random sample of data is drawn from the original training set, creating a unique dataset for each tree.
2. **Construct Individual Trees**: Using the bootstrapped sample, each tree is constructed based on random feature subsets, applying standard decision tree principles (as discussed in the previous section) to maximize information gain at each split.
3. **Aggregate Predictions**: For a given test document, each tree makes an independent prediction. The Random Forest classifier then aggregates these predictions through majority voting to determine the final class label.

3.5 Linear Models

3.5.1 Naive Bayes

The Naive Bayes (NB) classifier is a probabilistic (supervised) learning method based on Bayes' theorem, which assumes strong (naive) independence between the features. Let C denote the set of classes. For binary text classification in particular, $C = \{0, 1\}$. The goal is to classify a document, d, into a class $k \in C$ that maximizes the posterior probability $P(c = k|d) = P(c = k|\mathcal{T}(d)) = P(c = k|\boldsymbol{d})$. The probability of a document d being in class k is computed as [18]:

$$P(c = k|d) \propto P(c = k) \prod_{t \in \mathcal{T}(d)} P(t|c = k)$$

where $P(t|c = k)$ is the conditional probability of term (or encoded feature) t occurring in a document of class k, and $P(c = k)$ is the prior probability of class k. The DTM is organized in an ordered manner. Let $D^{(k)} \subseteq D$ (by row) be the (encoded) documents that belong to class k. Thus, when all terms are fully encoded, probabilities of the form $P(t|c = k)$ can be estimated using the columns of $D^{(k)}$. In particular,

$$P(t|c = k) \approx \frac{\sum_d D^{(k)}_{d,t} + \alpha}{\sum_{i,j} D^{(k)}_{i,j} + \alpha|V|}$$

where the smoothing term $\alpha = 1$ handles zero probabilities. The prior probability is simply estimated as $P(c = k) = \frac{|D^{(k)}|}{|D|}$ - the number of documents in class k out of the total number of documents.

To classify a new document d, the algorithm computes the score for each class:

$$S[k] = \log P(c = k) + \sum_{t \in \mathcal{T}(d)} \log P(t|c = k)$$

This formulation represents the logarithm of the posterior probability. Because the logarithm is a strictly monotonic increasing function, it preserves the order of probabilities while also preventing underflow issues in computational implementations. Next, the document is assigned to the class that has the highest score, determined by the maximum a posteriori (MAP) criterion, given by

$$k_{\text{MAP}} = \arg\max_{k \in C} S[k].$$

Naive Bayes classifiers are known for their simplicity and efficiency. They can be trained and applied to classify new data with a single pass over the data. Despite the strong independence assumptions, NB classifiers often perform surprisingly well in practice, particularly in text classification tasks where the independence assumptions are not strictly true. The pseudocodes below illustrates the processes involved in training and applying a Naive Bayes classifier. [18]

Algorithm 1 Train Naive Bayes Classifier

```
1:  procedure TrainMultinomialNB(D, C, V, $\alpha = 1$)
2:      $n, m :=$ Size($D$)
3:      for each $k \in C$ do
4:          $D_k :=$ DocsInClass($D, k$)
5:          $n_k :=$ NumRows($D_k$)
6:          prior[$k$] $:= n_k/n$
7:          Total$_k := \sum_{i,j} D_k[i][j] + \alpha|V|$
8:          for each $t \in V$ do
9:              $i =$ Index($t, V$)
10:             condprop[$t$][$k$] $:= \frac{\sum_d D_k[d][i] + \alpha}{\text{Total}_k}$
11:     return prior, condprob
```

Once trained, we can apply the classifier on new or unseen documents in the following manner.

Algorithm 2 Apply Naive Bayes Classifier

```
1: procedure ApplyMultinomialNB(d, C, V, prior, condprob)
2:     $T_d := \mathcal{T}(d)$
3:     for each $k \in C$ do
4:         score[$k$] := $\log \text{prior}[k] + \sum_{t \in T_d} \log \text{condprob}[t][k]$
5:     return $\arg\max_{k \in C} \text{score}[k]$
```

3.5.2 Logistic Regression

Logistic Regression is a widely used statistical method for binary classification problems. It models the probability that a given input belongs to a particular class by fitting data to a logistic curve. For the purposes of this project, we assume classification is strictly binary.

The probability that a given instance belongs to the positive class (class 1) is given by

$$P(c = 1|x) = \sigma(z),$$

where, $\sigma(z) = \frac{1}{1+e^{-z}}$, and $z = \boldsymbol{\beta}^\top \begin{bmatrix} 1 \\ \boldsymbol{d} \end{bmatrix}$ is a linear combination of input features with intercept. Here, the function, σ, is known as a *logistic function* which is also a *sigmoid function*. Further, $\boldsymbol{\beta} \in \mathbb{R}^{n+1}$ contain the coefficients of the model [19], where β_0 is the intercept.

Conversely, the probability of belonging to the negative class (class 0) is $1 - \sigma(z)$ [20].

Logistic regression estimates the parameters $\boldsymbol{\beta}$ using maximum likelihood estimation (MLE). The likelihood function for a single instance is:

$$L(\boldsymbol{\beta}) = P(y|x; \boldsymbol{\beta}) = \sigma(z)^y (1 - \sigma(z))^{1-y}$$

For a dataset with n instances, the overall likelihood is the product of the individual likelihoods:

$$L(\boldsymbol{\beta}) = \prod_{i=1}^{n} \sigma(z_i)^{y_i} (1 - \sigma(z_i))^{1-y_i}$$

The goal is to find the parameter values $\boldsymbol{\beta}$ that maximize this likelihood function. In practice, it is more convenient to work with the log-likelihood function,

$$\ln L(\boldsymbol{\beta}) = \sum_{i=1}^{n} \left[y_i \ln(\sigma(z_i)) + (1 - y_i) \ln(1 - \sigma(z_i)) \right].$$

Optimization algorithms such as gradient descent are used to find the parameters that maximize the log-likelihood [21].

Logistic Regression offers several advantages. It is easy to implement, interpret, and very efficient to train. It provides probabilities for class membership, which can be useful for decision-making. It also works well with linearly separable data and can be extended to multiclass classification problems using techniques like one-vs-rest and softmax regression.

However, logistic regression has some disadvantages. It assumes a linear relationship between the input features and the log-odds of the outcome, which may not always be true. It can struggle with complex relationships and interactions between features unless properly engineered. It is also sensitive to outliers and may require feature scaling for optimal performance [22].

3.6 Ensemble Methods

3.6.1 Gradient Boosting

Gradient boosting is an ensemble machine learning technique that builds models sequentially by combining weak learners, typically decision trees, to create a strong predictive model. The core idea is to optimize a loss function by iteratively adding models that correct the errors made by previous models.

The general process of gradient boosting can be summarized in the following steps: [10]

1. Initialize the model with a constant value, usually the mean of the target variable for regression tasks.
2. For a specified number of iterations, do the following:
    (a) Compute the pseudo-residuals, which are the gradients of the loss function with respect to the current model predictions.
    (b) Fit a new weak learner (e.g., a decision tree) to these pseudo-residuals.
    (c) Update the model by adding the new weak learner, scaled by a learning rate.
3. The final model is a weighted sum of all the weak learners.

Gradient boosting can effectively minimize various types of loss functions, making it versatile for both regression and classification tasks. The method is particularly effective when dealing with complex datasets where linear models may fail to capture underlying patterns.

Several implementations of gradient boosting exist, with some of the most popular being:

- **XGBoost** (Extreme Gradient Boosting): An optimized and efficient implementation of gradient boosting that includes regularization techniques to prevent overfitting.

- **LightGBM**: A gradient boosting framework that uses tree-based learning algorithms, designed for distributed and efficient training of large datasets.
- **CatBoost**: A gradient boosting library that handles categorical features automatically, making it user-friendly and effective for datasets with categorical variables.

Gradient boosting has gained popularity due to its high predictive performance, flexibility, and ability to handle different types of data. However, careful tuning of hyperparameters such as learning rate, number of trees, and tree depth is essential to avoid overfitting and to achieve optimal results.

3.7 Neural Networks

Neural networks are a foundational component of modern machine learning, particularly in the realm of deep learning. Inspired by the structure and function of the human brain, a neural network consists of interconnected nodes, or neurons, organized in layers. These networks are designed to recognize patterns within data by transforming inputs into outputs through a series of mathematical operations.

3.7.1 Structure

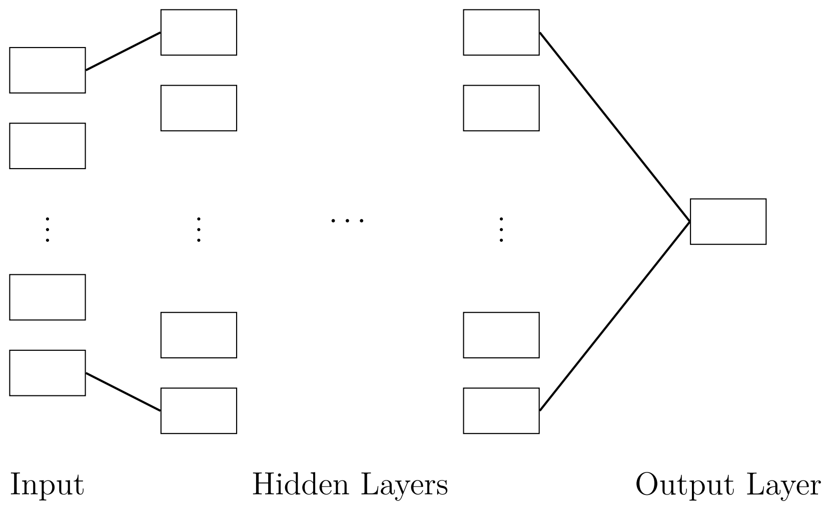


Figure 3.2: Neural Network for binary text classification

A typical neural network comprises three main types of layers:

- **Input Layer**: This layer receives the raw input data. Each neuron in the input layer corresponds to a feature in the dataset. For instance, in natural language processing, the features could be word embeddings or numerical representations of text.

- **Hidden Layers**: These layers lie between the input and output layers and consist of neurons that apply various transformations to the input data. The number of hidden layers and the number of neurons within each layer can vary, and these parameters greatly influence the model's ability to learn complex patterns. A neural network is considered *deep* if it contains 2 or more hidden layers. Each neuron in a hidden layer receives inputs from the previous layer, applies a weighted sum followed by an activation function, and passes the output to the next layer. Common activation functions include the ReLU (Rectified Linear Unit), sigmoid, and tanh, each contributing differently to the network's learning capacity.
- **Output Layer**: The output layer generates the final predictions based on the transformed data from the hidden layers. The structure of this layer often depends on the specific task; for binary classification, we use a single neuron with a sigmoid activation function. However, it is also possible to have multiple output in case of multi-class classification. Here, a *softmax* activation function is preferred when outputs are considered mutually exclusive.

3.7.2 Learning Process

Neural networks learn through a process called backpropagation, which involves two key phases: forward propagation and backward propagation. During *forward propagation*, the input data is passed through the network, resulting in an output. This output is then compared to the actual target label to compute a loss value using a loss function (such as mean squared error for regression tasks or cross-entropy loss for classification).

In the *backward propagation* phase, the network updates its weights based on the calculated loss using optimization algorithms such as stochastic gradient descent (SGD) or Adam. This iterative process adjusts the weights to minimize the loss, allowing the network to improve its predictions over time.

A critical aspect of the learning process involves the use of *activation functions*, which introduce non-linearity into the network. This non-linearity enables the network to learn complex patterns and interactions within the data. Without activation functions, a neural network with multiple layers would behave like a linear model, severely limiting its capacity to model intricate relationships.

Several activation functions are commonly used, each with its own advantages and applications:[15]

- **Sigmoid Function**: A function, $\sigma : \mathbb{R} \to \mathbb{R}$, that is bounded, differentiable, monotonically increasing, and has exactly one inflection point. The most commonly used sigmoid function is the logistic function, which is particularly useful for binary classification tasks. Its output is often limited to the interval $(0, 1)$. Another variant, the hyperbolic tangent (tanh), has a codomain of $(-1, 1)$, making it zero-centered and often leading to faster convergence. The formula for tanh is:

$$\tanh(x) = \frac{e^x - e^{-x}}{e^x + e^{-x}}$$

However, sigmoid activations can suffer from the *vanishing gradient problem*, where gradients become very small for large or small input values, slowing down learning in deeper networks.

- **ReLU (Rectified Linear Unit)**: ReLU is one of the most widely used activation functions in deep neural networks, defined as:

  $$\text{ReLU}(x) = \max(0, x)$$

  This function is computationally efficient and helps alleviate the vanishing gradient problem by not saturating in the positive domain. However, it can cause some neurons to become inactive (when $x < 0$), a phenomenon known as the *dying ReLU problem.* Variants such as Leaky ReLU and Parametric ReLU (PReLU) introduce small slopes for negative inputs to counter this issue.

- **Softmax Function**: Commonly used in the output layer for multi-class classification, the softmax function converts a vector of raw scores into probabilities. For an output vector $\mathbf{z}$, the softmax function for the i-th element is:

  $$\text{softmax}(z_i) = \frac{e^{z_i}}{\sum_j e^{z_j}}$$

  This ensures that the output values sum to 1, making them interpretable as probabilities.

Incorporating these activation functions is essential for enabling the neural network to capture and represent complex relationships in the data.

Another important technique that enhances the learning process is **batch normalization** (BN). BN stabilizes and accelerates the training process by normalizing the inputs to each layer. During training, batch normalization adjusts and scales the output of each layer so that it has a mean of zero and a variance of one across a mini-batch of data. The process can be described mathematically as follows:

Given an input x_i in a mini-batch, we compute:

1. The batch mean:

   $$\mu_{\text{batch}} = \frac{1}{m} \sum_{i=1}^{m} x_i$$

2. The batch variance:

   $$\sigma^2_{\text{batch}} = \frac{1}{m} \sum_{i=1}^{m} (x_i - \mu_{\text{batch}})^2$$

3. Normalization of each input:
$$\hat{x}_i = \frac{x_i - \mu_{\text{batch}}}{\sqrt{\sigma^2_{\text{batch}} + \epsilon}}$$

4. Scaling and shifting:
$$y_i = \gamma \hat{x}_i + \beta$$

where γ and β are learnable parameters that allow the network to scale and shift the normalized output, retaining its representational capacity. The constant ϵ is a small positive value to prevent division by zero. Batch normalization can reduce the internal covariate shift (the change in the distribution of layer inputs during training), leading to faster convergence and improved generalization.

To further enhance model robustness, *dropout* is employed as a regularization technique used to prevent overfitting in neural networks. During each training iteration, a certain percentage of neurons are randomly "dropped" (set to zero), which forces the network to rely on different subsets of neurons each time. This encourages the network to develop more robust representations.

Mathematically, for a layer with output $\mathbf{h}$ and dropout rate p, the dropout operation can be represented as:
$$\tilde{\mathbf{h}} = \mathbf{h} \odot \mathbf{r}$$

where $\mathbf{r}$ is a binary mask vector, with each element r_i sampled from a Bernoulli distribution:

$$r_i \sim \text{Bernoulli}(1 - p)$$

During inference, dropout is disabled, but the weights are scaled down by $(1 - p)$ to account for the lack of deactivated neurons. This technique helps prevent the network from becoming overly reliant on any single neuron, promoting generalization and reducing the risk of overfitting.

By incorporating activation functions, batch normalization, and dropout, neural networks can effectively learn complex patterns, converge faster, and generalize better, making them powerful tools for a wide range of tasks.

3.7.3 Convolutional Neural Networks

Convolutional Neural Networks (CNNs) are traditionally associated with image processing, but they have also gained traction in Natural Language Processing (NLP). To utilize CNNs for textual data, the input data needs to be in a grid-like format. Fortunately, this is already the case for Document-Term Matrices, which can essentially be seen as greyscale images. CNNs leverages a series of convolutional layers to automatically and adaptively learn spatial hierarchies in data. CNNs are particularly effective at capturing local patterns through convolution operations.[23]

Structure of a Convolutional Neural Network

A CNN typically comprises several types of layers arranged in a sequence to form a deep network architecture. The primary types of layers in a CNN are:

- **Convolutional Layer**: This layer performs the convolution operation, which applies a filter (or kernel) to the input data to produce a feature map. Each filter is a small matrix that slides over the input and computes dot products to capture local patterns, such as edges, textures, or shapes, in an image. Mathematically, for an input matrix X and a filter matrix K, the convolution operation at position (i, j) is:

  $$(X * K)_{i,j} = \sum_m \sum_n X_{i+m,j+n} \cdot K_{m,n},$$

  where m and n index the filter's height and width, respectively. A key advantage of this approach is that filters can detect features regardless of their location in the image, making CNNs translation-invariant.

- **Pooling Layer**: After each convolutional layer, a pooling layer is often applied to reduce the spatial dimensions of the feature map, retaining the most essential information. The most common pooling technique is max pooling, which takes the maximum value within each region, emphasizing the most prominent features. Pooling reduces computation, controls overfitting, and ensures that small translations in the image have minimal effect on the output.

- **Fully Connected Layer**: Towards the end of the network, fully connected layers are often used to map the extracted high-level features to the final output classes. Each neuron in a fully connected layer is connected to every neuron in the previous layer, aggregating information across all spatial locations to make predictions. Essentially, the now processed and flattened image is used as input neurons into a conventional neural network.

- **Activation Function**: Similar to standard neural networks, CNNs use activation functions such as ReLU to introduce non-linearity after convolution and pooling operations, allowing the network to capture complex patterns.

The output layer of a CNN is typically a softmax layer for multi-class classification tasks, producing a probability distribution across different classes.

Learning Process in CNNs

CNNs use the same learning process as traditional neural networks, involving forward and backward propagation. In the forward pass, input data moves through the convolutional, pooling, and fully connected layers, eventually producing an output. During backward propagation, CNNs adjust the weights of the filters in each layer based on the error gradients, optimizing the network to minimize loss.[23]

One key difference in CNN training is the handling of **weights** in the convolutional layers. Filters in these layers are learned automatically during training, where each filter is updated to recognize specific patterns relevant to the task, such as edges in the first layers and more complex textures in deeper layers. As the network deepens, it captures increasingly abstract representations, ultimately learning intricate features that distinguish different classes.

CNNs have several advantages that make them well-suited for computer vision tasks:

Parameter Efficiency: By sharing weights in convolutional layers, CNNs use fewer parameters, making them less prone to overfitting and more efficient to train than fully connected networks. **Translation Invariance**: The sliding window approach allows CNNs to detect patterns regardless of their position in the image. **Hierarchical Feature Learning**: The layered architecture enables CNNs to learn low-level to high-level features, capturing complex structures that aid in object recognition and classification.

Applications of CNNs

CNNs are widely used in image classification, where they classify objects within an image; object detection, where they identify the location of objects; and segmentation, where they label each pixel in an image. Beyond computer vision, CNNs are also adapted for other structured data types, including audio signal processing and certain natural language processing tasks, illustrating their versatility and power in various domains.

Convolutional Neural Networks (CNNs), originally designed for image classification, are increasingly being applied to natural language processing (NLP) tasks. This is because the textual data, effectively represented as grayscale images, reveals patterns that are yet to be understood, but can be effectively leveraged for predictive modeling.

4 | Application

The goal of this project is to develop some financial strategy that takes in a reddit post that contain a ticker symbol along with corresponding price data from the stock and outputs whether or not to invest.

Computing the market sentiment is a seperate NLP task alltogether. In order to analyze market sentiment on r/WallStreetBets, it is necessary to collect all the data of the subreddit. This can be done through means of web-scraping; however, since this is a paid service, we will be making use of already existing data dumps[24]. By downloading 643GB worth of Reddit posts from 2012 (since inception) up to and including the year 2022, we can filter out all the data that belongs to WSB by iterating through the `zst` compressed `ndjson` files. Next, we convert the data to CSV, yielding a total of 2,349,125 rows of Reddit posts. As can be seen in the following example, we have chosen 8 different fields out of the available 112 for each post.

Field	**Value**
`author`	u/DeepFuckingValue
`created`	2021-01-28 22:06
`id`	l78uct
`link_flair_text`	YOLO
`num_comments`	23,650
`score`	282,358
`selftext`	
`title`	GME YOLO update — Jan 28 2021

Despite being one of the most influential and popular posts on the subreddit, this specific post would not significantly benefit a machine learning algorithm. Although its category, `score`, and number of comments might suggest a significant stock movement, the direction is unclear due to the lack of sentiment information in the title. And the description is empty since the original post was just an image of losses and gains to the tune of several million dollars. In the following section we will analyse the data to provide further insight in the subreddit.

4.1 Data Analysis

Only the data that contains ticker symbols are conducive to making stock predictions. While there are many ticker symbols on the New York Stock Exchange, most are not active nor easily borrowable. Using the alpaca api, we can retrieve lists of ticker symbols and check for matches in the reddit posts. The table below provides an overview.

Category	**Available**	**Necessary**	**Posts**	**Mentions**
Tickers	31,590	8,370	650,104	1,130,569
→ Active	11,803	5,815	615,583	1,032,093
→ Easily Borrowable	4,556	3,551	530,194	851,373

Table 4.2: Ticker counts in the corpus

The first column shows the number of tickers that are available in the NYSE and the second shows the number of tickers that are actually present in the data. The third column is the number of reddit posts that contain ticker symbols, whereas the last column is the total number of ticker mentions. In the rows, we also consider the active tickers as well as those that are, in addition, easily borrowable.

We continue our investigation using the $3,551$ active and easily borrowable tickers present in the text. The following bar chart shows the 10 most frequently mentioned tickers.

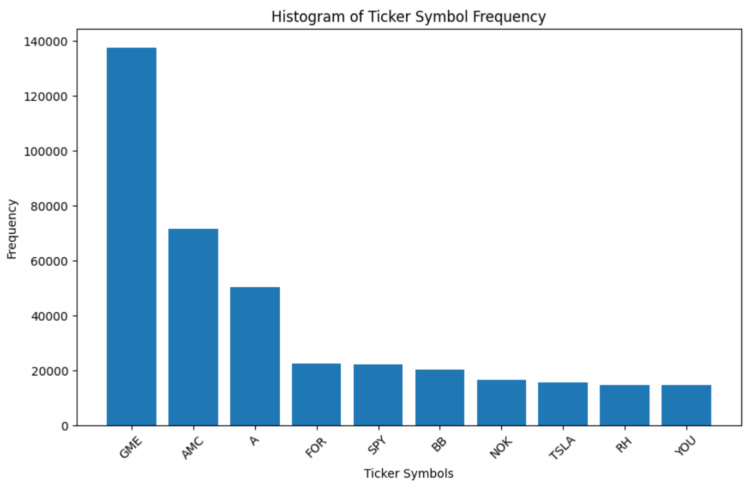


Figure 4.1: Number of ticker mentions in corpus

As can be seen on the above chart, certain words such as "A", "DD", and "FOR" are quite frequent. Here, "DD" as "Due Diligence" is much more commonly used than references to the DuPont company and it is intuitive how common English words such as "FOR" and

"A" are more likely to occur than references to the respective companies despite the use of capitalization. Therefore, we have compiled a comprehensive list of 112 tickers to exclude (Table A.2). In doing so, we may update the chart as follows;

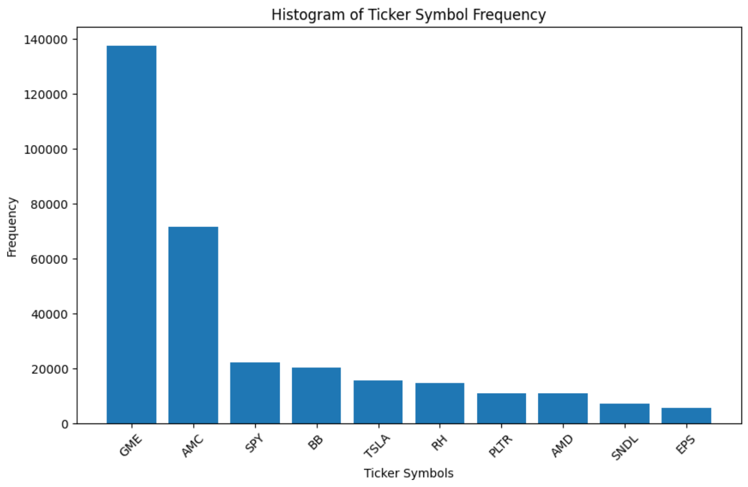
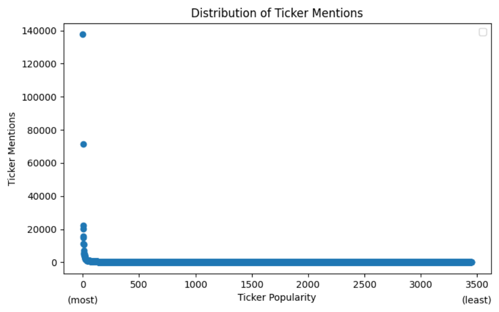


Figure 4.2: Number of ticker mentions after removal

Now, out of the remaining 3449 tickers, we can take a look at the distribution of ticker mentions by frequency rank.

(a) Raw data



(b) log-log plot with regression line

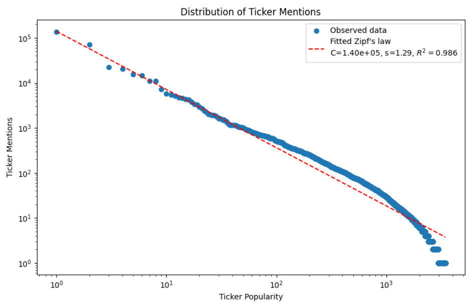


Figure 4.3: Ticker mentions by rank

Since Figure 4.3a appears to exhibit an inverse relationship, we log-transform the data, enabling us to fit a regression line in Figure 4.3b. This transformation suggests that $\ln p(x) = -s \ln x + c$ for some constants s and c. From the regression, we obtain $s \approx 1.29$ and $C \approx 140{,}000$, where $p(x) = Cx^{-s}$ with $C = e^c$ and $R^2 \approx 0.986$. This relationship is known as *Zipf's Law.*[25]

If, in addition, we limit ourselves to tickers that have been mentioned at least 50 times and exclude the top 10 most mentioned outliers, we get $s \approx 1.11$ and $R^2 \approx 0.993$ (plot available in Appendix A.3).

The reason that we have this type of relationship instead of, say, a linear relationship (without log-transforming) is due to a rich-get-richer phenomenon where the highly popular or influential stocks are discussed more frequently due to being popular in a self reinforcing cycle. Similar to compound interest, the growth of attention on a stock such as GME accumulates over time, resulting in a graph that shows increasing mentions as the stock's popularity builds.

4.2 Building The Sentiment Pipeline

Since this subreddit contains quite unique terminologies, existing tools like TextBlob are not very efficient in determining sentiment. For example, phrases such as "GME to the moon!" are not uncommon on Reddit, but since TextBlob was trained on more generic datasets, it does not recognize the positive sentiment. However, there also exists other tools such as Valence Aware Dictionary and sEntiment Reasoner (VADER), which is specifically tuned for social media and short text environments. While this specific phrase is classified as neutral, other texts of reddit are often classified correctly and can thus serve as a benchmark.

We seek to improve performance by creating our own pipeline specific to this subreddit. In this project, we will be making use of supervised learning. This involves labeled data, which on first inspection, is not directly available. However, since the data is categorized, it is not unreasonable to assume that the categories, "Gain", and "Loss" are highly related to positive and negative sentiment, respectively. This provides a supervised framework to train our model on. Therefore, for the purposes of this project we consider sentiment as binary - either positive or negative. Unfortunately, this reduces our available data to 257,901 posts (11%). This dataset is significantly larger than what could be done by manual labeling. For the remainder of this section, we will be making use of this dataset.

Category	**Total**	**Per post**
characters	5,226,616	78.79
words	1,002,780	15.09
unique words	29,090(2.9%)	0.44

Table 4.3: Summary of post and word statistics (out of 257,901 posts)

In order to build an NLP pipeline on this data, further preprocessing is necessary. This can significantly reduce noise so that the machine can extract relevant information. First, specific emojis were replaced with their text representations to preserve sentiment, while the remaining emojis, symbols, and pictograms were removed to reduce noise. HTML tags, URLs, email addresses, and text within brackets (such as [removed] and [edited]) were stripped to retain only relevant content. Ticker symbols, which do not offer semantic value, were also removed. The text was then converted to lowercase for uniformity, and contractions were expanded to their full forms for better recognition. Non-alphabetic characters, including numbers and punctuation, were filtered out, leaving only alphabetic text. Extra whitespace was collapsed into single spaces, ensuring clean tokenization. Lastly, stop words were removed to focus on more meaningful words, with analysis being conducted both with and without stop words. [1]

4.2.1 Statistics

(a) Number of characters in corpus **(b)** Classes

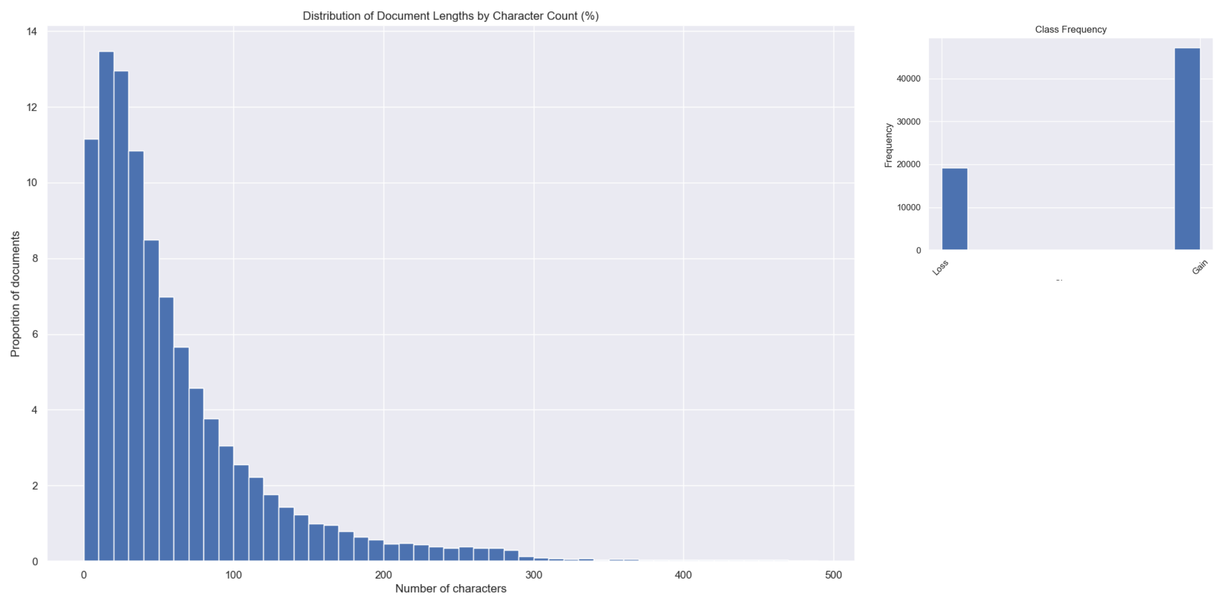


Figure 4.4: Comparison of different figures side by side.

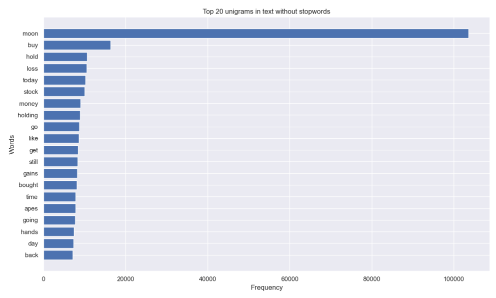
As can be seen 4.4a, the average reddit post is fairly short. The effect of this is not yet known, since shorter signals may mean less noise, yet we also have less data. This is not necessarily a bad thing, as it could also be that the corpus is simply a collection of short clear sentiment signals. In addition, subreddit has a tendency to show gains more often than losses as can be seen on 4.4b. A general rule of thumb in machine learning is that if the minority class has less than 10% of the number of datapoints of the majority class, it is considered unbalanced since this is often when algorithms start to struggle with bias towards the majority class. Therefore, it should be safe to proceed.

[1]All code is available on github[26]

4.2.2 n-grams

Since text classification can be done on a word basis, however character level classification as well as the use of bigrams, trigrams is also possible. The following bar chart provides the most common unigrams with- and without stop words.

(a) Number of characters in corpus



(b) Classes

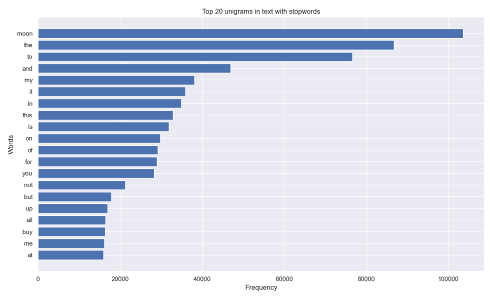


Figure 4.5: Comparison of different figures side by side.

As can be seen on the bar charts, the most common words are significantly more informative once stopwords have been removed. Followers of the subreddit may recognize that several of these words are likely part of a phrase such as ”we like the stock”, ”apes together strong”, and ”diamond/paper hands”.

Followers of the subreddit can recognize several common phrases, such as ”we like the stock”, ”apes together strong”, ”diamond/paper hands”, and the extensive use of the word moon is mainly due to the frequent use of rocket emojis. In section A.1, the most common bigrams and trigrams are available.

4.3 Model selection

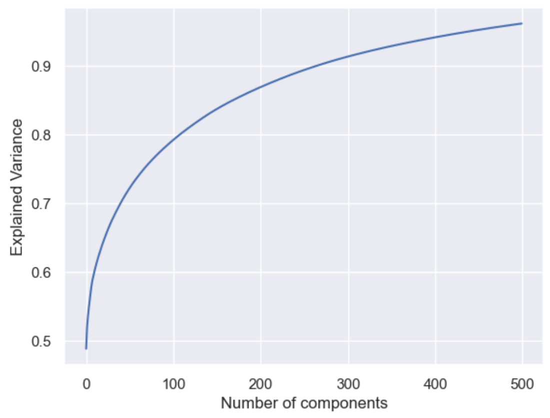
There exists a wide variety of models to choose from. In this project, we will consider 6 models with 3 different encodings. Through 5-fold cross-validation, we will be comparing Naive Bayes, Logistic regression, random forest, gradient boosting, deep neural networks and Convolutional Neural Networks.

4.3.1 Encoding Data

We start by splitting the data into a train and test set. Here, the test set consists of 20% of the data and we have used a stratified split. We will be encoding the data in 3 ways. Using word-level tokens without stop-words, we are left with a 206,320 $\times$ 46,494 training DTM with 1,127,292 elements. This is a very sparse matrix, since for each non-zero element there are roughly 8508 zero's. In order for the more complex models to be feasible, our common feature set needs to be small enough for the computer to handle. Therefore, it is necessary to removing the most rare words before applying SVD to reduce the sparsity of this matrix. We seek to find a vocabulary that retains at least 70% of elements. In the case of word-level tokens, this amounts to a vocabulary of just 773 elements and we will be utilizing both one-hot- as well as TF-IDF encoding.

The following figure shows the explained variance as a function of the number of latent features included after applying SVD.

(a) Explained Variance for one-hot encoding



(b) Explained variance for word-level TF-IDF encoding

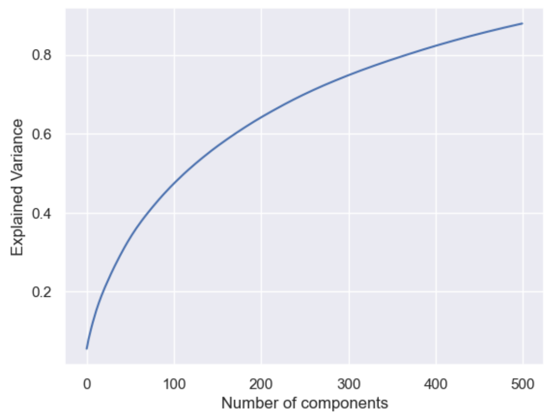


Figure 4.6: Explained Variance with word-level encodings

Since both encodings are using word-level tokens, the respective vocabularies are the same. However, the number of components required to achieve the same explained variance is much greater. Therefore, by also choosing a common threshold (0.8) of explained variance, we establish a level playing field for comparison. For word-level one-hot encoding, this results in 109 features, whereas word-level TF-IDF requires 369 features for the same threshold.

Character-level TF-IDF has the smallest vocabulary of 13,144 (yet is still 72% of all possible character combinations) different character bigrams and trigrams. Of this, the 553 most frequently used words account for 70% of the non-zero elements. As illustrated in the following figure, this encoding requires less features than word-level TF-IDF, yet still more than one-hot encoded words.

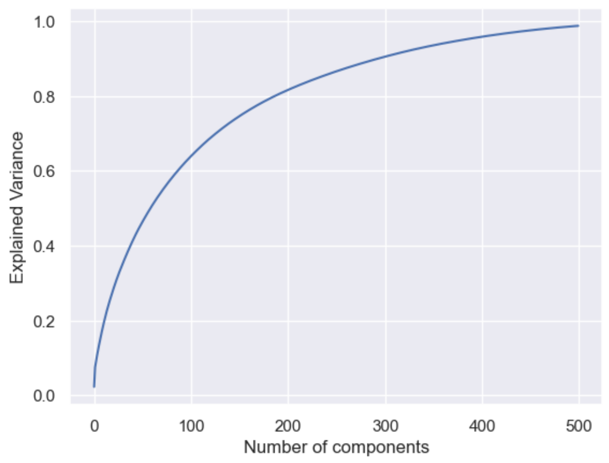


Figure 4.7: Explained variance for character level n-gram TF-IDF encoding

The following table summarizes the final feature sets for each encoding type after SVD.

Token Size	**Encoding**	**Number of Features**
Word	One-Hot	109
Word	TF-IDF	369
2-3 Characters	TF-IDF	187

Table 4.4: Encoding Summary

4.3.2 Results

The Naive Bayes Estimator cannot use any negative inputs which are often generated from Singular Vector Decompositions. Therefore, we use the original 773 words and the 553 character combinations, respectively as features. The following table summarizes our findings

		Mean		Std.	
Encoding	Model	acc	bal. acc	acc	bal. acc
C. TF-IDF (no SVD)	NB	0.712	0.624	0.001	0.001
C. TF-IDF	LR	0.720	0.645	0.002	0.002
OneHot	LR	0.721	0.627	0.001	0.001
OneHot	CNN	0.720	0.635	0.001	0.001
C. TF-IDF	CNN	0.725	0.645	0.001	0.002
C. TF-IDF	RF	0.735	0.644	0.001	0.002
TF-IDF	CNN	0.736	0.655	0.001	0.001
OneHot (no SVD)	NB	0.742	0.692	0.002	0.002
OneHot	RF	0.744	0.677	0.001	0.002
TF-IDF	LR	0.747	0.678	0.002	0.002
TF-IDF	RF	0.750	0.677	0.002	0.003
OneHot	DNN	0.750	0.687	0.002	0.003
TF-IDF (no SVD)	NB	0.751	0.674	0.003	0.003
C. TF-IDF	DNN	0.754	0.686	0.002	0.002
TF-IDF	DNN	0.763	0.701	0.002	0.002

Table 4.5: Model Accuracy (least to highest)

In Table 4.5, we notice that modelling on TF-IDF encoded data yields the best results (by accuracy) for each model. When considering balanced accuracy, there is only one exception which is One-Hot-encoded Naive Bayes. We proceed with the TF-IDF encoded DNN and we get the following classification report.

	precision	recall	f1-score	support
1	0.82	0.88	0.85	33,707
0	0.74	0.63	0.68	17,874
accuracy			0.80	51,581
macro avg	0.78	0.63	0.71	51,581
weighted avg	0.79	0.80	0.79	51,581

Table 4.6: Classification report for the sentiment analysis model.

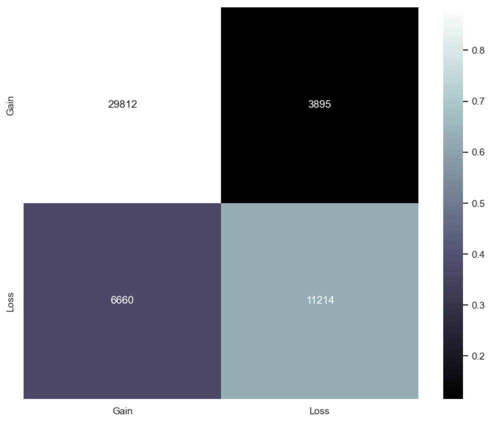


Table 4.7: Confusion Matrix

The following figures illustrates the performance over time.

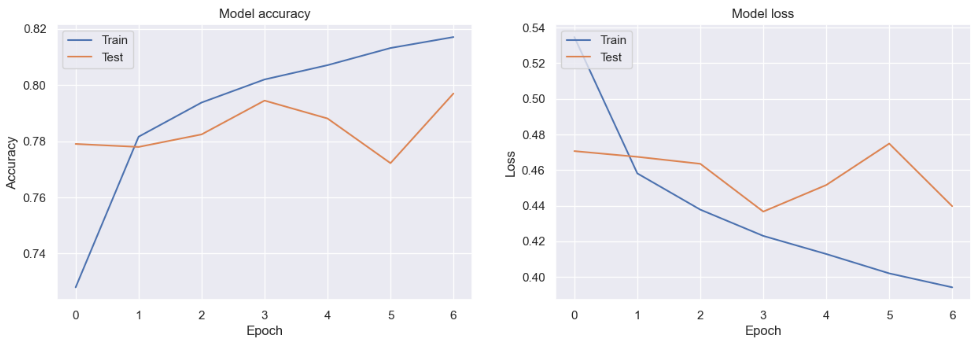


Figure 4.8: Accuracy over Time

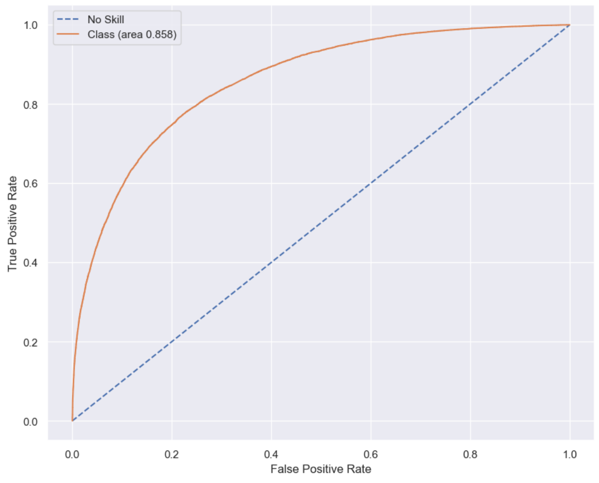


Figure 4.9: ROC for DNN

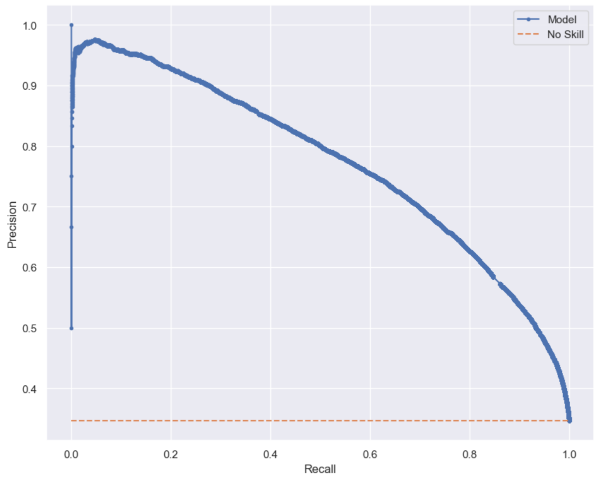


Figure 4.10: Precision curve for DNN

4.4 Constructing trading strategy

In this section we will construct a trading strategy that utilizes the text of a reddit post to generate trading signals. We define;

$$\text{signal} = \begin{cases} 0 & \text{Short position} \\ 1 & \text{Long position} \\ 2 & \text{Cover short position} \\ 3 & \text{Sell long position.} \end{cases}$$

This setup will double the size of our dataframe and is beneficial for backtesting purposes since we explicitly know exactly when to cover short positions and sell off long positions. However, it is not ideal for prediction purposes since there are 4 signals to predict. To get around this problem we simply use a fixed timeframe to predict the short- and long positions using the original dataframe. This way, we uncover the remaining trading signals by just adding 2 to the signal while keeping track on the sell/cover time in a seperate column. Repeating this process, we can build 3 models - a 1 hour model, a 30 minute model and a 10 minute model.

The strategy: If we receive a signal of 0 we enter a short position and then wait until the stock has decreased by 1% or the fixed time frame has passed. Similarly, if we receive a signal of 1 we enter a long position and wait until the stock has increased by 1% or the fixed time frame has passed.

So, to label our dataset, we gather trade data of the ticker mentioned in the reddit post up to the fixed timeframe and mark it as 0 if it has decreased by 1% and 1 if it has increased by 1%. If neither happen, then the end price determines the label (decrease marked as 0, while increase is marked as 1).

4.4.1 Data labeling

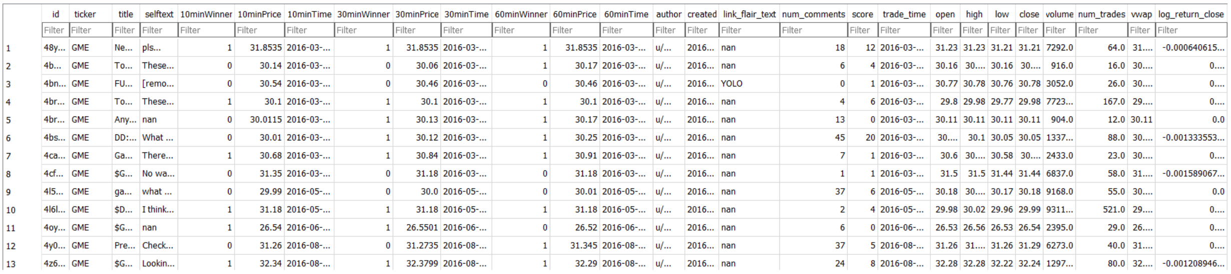
To label our data we make use of the alpaca api. In general, making a lot of api calls is a slow process. So to speed things up we made use of several optimizations. Unlike r/wallstreetbets, the NYSE is not open at all times of the day. Therefore, we can exclude the posts that fall on weekends as well as those outside the open hours (see Figure A.4 for distribution). Further, we found that sometimes tickers are mentioned before their ipo date which we can also exclude. Lastly, we incorporated a cache system to limit the number of api calls when there are a lot of posts about the same ticker in a short amount of time. This often happens during major financial events with respect to the ticker. This also means that we sort the data by ticker (from most popular to least). The optimal cache time period chosen proportional to the number of overlapping datapoints within the time frame of the respective model.

The following pseudocode illustrates this process further.

Algorithm 3 Collect and Process Daily Data

```
procedure CollectAndProcessData(data, fromDate, toDate, timeFrames, maxHours, batchSize, tickerStart)
    data := Sort(WsbPosts(data, fromDate, toDate), by='created')
    if DatabaseExists('TrainingData.db') then
        existingData := LoadFromDatabase('DataTable')
        data := RemoveDuplicates(data, existingData)
        if data is empty then
            return
    timeFrames := Sort(timeFrames, reverse=True)
    tickers := UniqueSort(data.tickers, by = 'Count')
    for each ticker in tickers do
        submissions := Filter(data, 'ticker' == ticker)
        Sort(submissions, by='created')
        Date = submission.created
        cacheTime := GetCacheTime(submissions, maxHours, max(timeFrames))
        Initialize(TradeCache, cacheTime)
        recordsToInsert := []
        ipoDate := GetIpoDate(ticker)
        for each submission in submissions do
            if Date < ipoDate or MarketClosed(Date) then Continue
            p := InitializeNewPost(submission, ticker)
            for each timeFrame in timeFrames do
                trades := GetLatestTrades(ticker, p.created, timeFrame)
                if trades is empty then
                    break
                p := SetWinnerAttributes(p, trades, timeFrame, pInc=1, pDec=1)
            Append(recordsToInsert, p)
            if ReachedBatchSize(recordsToInsert, batchSize) then
                InsertIntoDatabase(recordsToInsert)
                recordsToInsert := []
```

By running this code, we retrieve all the relevant data to generate trading signals. The following picture shows the structure of our SQL table.



	id	ticker	title	selftext	10minWinner	10minPrice	10minTime	30minWinner	30minPrice	30minTime	60minWinner	60minPrice
	Filter	Filter	Filter	Filter	Filter	Filter	Filter	Filter	Filter	Filter	Filter	Filter
1	48y...	GME	Ne...	pls...	1	31.8535	2016-03-...	1	31.8535	2016-03-...	1	31.8535
2	4b...	GME	To...	These...	0	30.14	2016-03-...	0	30.06	2016-03-...	1	30.17
3	4bn...	GME	FU...	[remo...	0	30.54	2016-03-...	0	30.46	2016-03-...	0	30.46
4	4br...	GME	To...	These...	1	30.1	2016-03-...	1	30.1	2016-03-...	1	30.1
5	4br...	GME	Any...	nan	0	30.0115	2016-03-...	1	30.13	2016-03-...	1	30.17
6	4bs...	GME	DD:...	What ...	0	30.01	2016-03-...	1	30.12	2016-03-...	1	30.25
7	4ca...	GME	Ga...	There...	1	30.68	2016-03-...	1	30.84	2016-03-...	1	30.91
8	4cf...	GME	$G...	No wa...	0	31.35	2016-03-...	0	31.18	2016-03-...	0	31.18
9	4l5...	GME	ga...	what ...	0	29.99	2016-05-...	0	30.0	2016-05-...	0	30.01
10	4l6l...	GME	$D...	I think...	1	31.18	2016-05-...	1	31.18	2016-05-...	1	31.18
11	4oy...	GME	$G...	nan	1	26.54	2016-06-...	1	26.5501	2016-06-...	0	26.52
12	4y0...	GME	Pre...	Check...	0	31.26	2016-08-...	0	31.2735	2016-08-...	1	31.345
13	4z6...	GME	$G...	Lookin...	1	32.34	2016-08-...	1	32.3799	2016-08-...	1	32.29

	60minTime	author	created	link_flair_text	num_comments	score	trade_time	open	high	low	close	volume	num_trades	vwap	log_return_close
	Filter	Filter	Filter	Filter	Filter	Filter	Filter	Filter	Filter	Filter	Filter	Filter	Filter	Filter	Filter
1	2016-03-...	u/...	2016...	nan	18	12	2016-03-...	31.23	31.23	31.21	31.21	7292.0	64.0	31....	-0.000640615...
2	2016-03-...	u/...	2016...	nan	6	4	2016-03-...	30.16	30....	30.16	30....	916.0	16.0	30....	0....
3	2016-03-...	u/...	2016...	YOLO	0	1	2016-03-...	30.77	30.78	30.76	30.78	3052.0	26.0	30....	0....
4	2016-03-...	u/...	2016...	nan	4	6	2016-03-...	29.8	29.98	29.77	29.98	7723...	167.0	29....	0....
5	2016-03-...	u/...	2016...	nan	13	0	2016-03-...	30.11	30.11	30.11	30.11	904.0	12.0	30.11	0.0
6	2016-03-...	u/...	2016...	nan	45	20	2016-03-...	30....	30.1	30.05	30.05	1337...	88.0	30....	-0.001333553...
7	2016-03-...	u/...	2016...	nan	7	1	2016-03-...	30.6	30....	30.58	30....	2433.0	23.0	30....	0....
8	2016-03-...	u/...	2016...	nan	1	1	2016-03-...	31.5	31.5	31.44	31.44	6837.0	58.0	31....	-0.001589067...
9	2016-05-...	u/...	2016...	nan	37	6	2016-05-...	30.18	30....	30.17	30.18	9168.0	55.0	30....	0.0
10	2016-05-...	u/...	2016...	nan	2	4	2016-05-...	29.98	30.02	29.96	29.99	9311...	521.0	29....	0....
11	2016-06-...	u/...	2016...	nan	6	0	2016-06-...	26.53	26.56	26.53	26.54	2395.0	29.0	26....	0....
12	2016-08-...	u/...	2016...	nan	37	5	2016-08-...	31.26	31....	31.26	31.29	6273.0	40.0	31....	0....
13	2016-08-...	u/...	2016...	nan	24	8	2016-08-...	32.28	32.28	32.22	32.24	1297...	80.0	32....	-0.001208946...

Figure 4.11: Reddit Post- and Trade Data

Note that in Figure 4.11, we also capture the time and price when the stock increases (or decreases) by 1%. The columns on the form, "xminWinner", shows whether the stock increases (1) or decreases (0) the same way as described in section 4.4 and the remaining columns depict the minute-bar trading data closest to the creation of the reddit post submission. Since the "created" column is only measured to the nearest minute, the "trade_time" column from the minute bar data happens to correspond to the exact times in the "created" column. From this operation, as the tickers became rarer, we experienced diminishing returns in terms of data points retrieved per ticker as well as computational efficiency. Therefore, we settled on a dataset consisting of 319 tickers and 283243 data points.

Calculating sentiment

Now that we have collected all the relevant data, we will use our Neural Network to measure the sentiment of each post so we can use that as a feature instead of text in our final model. Since our model essentially predicts how likely that a given post belongs to the category, 'Gain' as opposed to 'Loss', we only need to compute the sentiment for the posts that do not belong to 'Gain' or 'Loss' categories. The sentiment column is essentially conditional probabilities. The conditional probability of belonging to the category 'Gain' given that we belong to the category 'Gain' is 1, while the remaining rows are conditioned on no information and just calculated from the model. Note that we do not condition on other categories, such as 'News', 'YOLO', etc. The following figure illustrates the sentiment over time for select tickers.

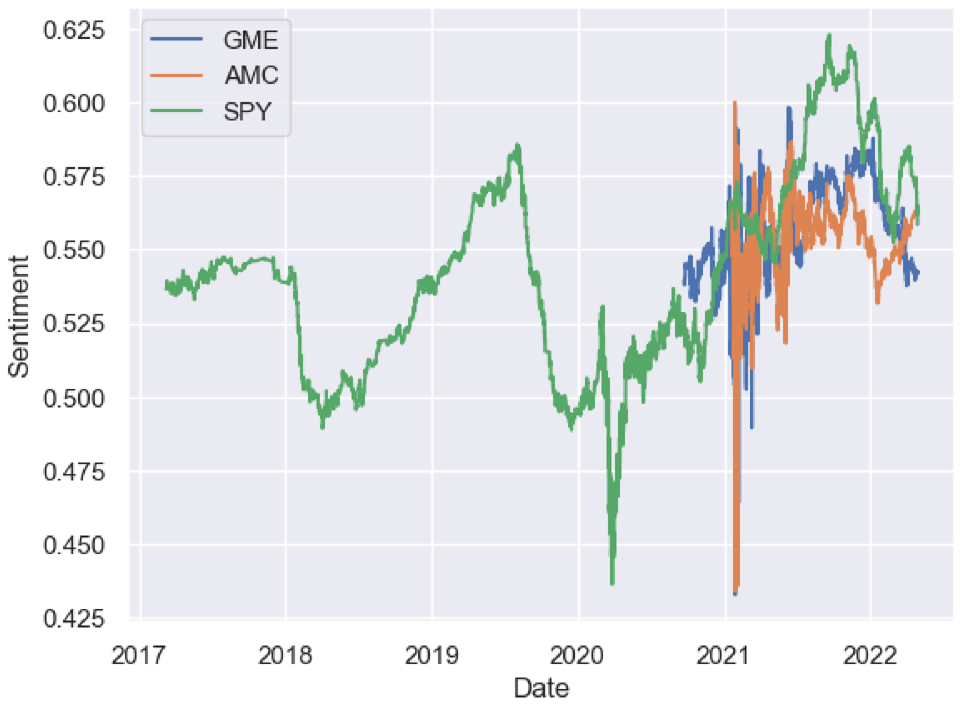


Figure 4.12: Sentiment over Time for Select Tickers (neutral = 0.5)

4.5 Testing predictability

In this section, we assess whether sentiment scores can predict stock returns. For this, we check granger causality. It is important to conducting Granger causality tests on sentiment scores and stock prices for each ticker individually, rather than aggregating the entire dataset. This is due to the sensitivity of Granger causality to the underlying time series structure, which can vary significantly across different tickers. However, instead of checking granger causality for all the tickers, we use the same tickers as shown in Figure 4.12 as a test sample to assess whether or not it makes sense to use sentiment as a feature in our final Neural Network. The granger test assumes stationarity, so it is important to ensure that the data is stationary before conducting the test. To evaluate the stationarity of our time series data, we utilize the Augmented Dickey-Fuller (ADF) test. In doing so, we establish that all sentiment scores as well as Log-Returns are stationary for GME, AMC, and SPY. For more details, see Table A.3. Note that the prices of the stocks generally do not pass the ADF test for stationarity, as indicated by their higher p-values, which reinforces the necessity of using log-returns or sentiment scores for subsequent Granger causality tests.

4.5.1 Granger Causality

We apply the Granger causality test to examine whether sentiment can predict `log_return_close` for various stock tickers. The maximum lag used is 50. From the `grangercausalitytests` function in the `statsmodels` package, we are provided with p-values at each lag using f-test, χ^2-test, lr-test and parameter f-test, respectively. Out of 30 of the most common tickers, 6 had p values below 0.05 (TSLA, NFLX,...) , suggesting the possibility that log-returns can predict sentiment, and 7 tickers (BB, MSFT...) suggesting that the sentiment scores can predict log-returns. It is important to note that while Granger causality indicates a potential predictive relationship, it does not imply causation in the strictest sense. Other factors could be at play, and sentiment might be one of several variables influencing stock returns.

4.5.2 Cross-Correlation

To further explore the relationship between sentiment scores and log-returns, we check for cross-correlation. Cross-correlation helps us understand the lagged relationship between two time series, in this case, sentiment and log-returns (see Figure A.5). By examining the cross-correlation plots, we can identify any significant lags where sentiment scores might be correlated with subsequent log-returns or vice versa. These plots can be found in Figure A.6. Note that for many of the stocks, the correlation at positive lags is negative, suggesting that higher sentiment scores may be followed by lower log-returns. This inverse relationship could indicate that when sentiment is high, it may already be priced into the stock, and subsequent returns could be lower as a result. Conversely, negative sentiment scores might precede positive log-returns, possibly due to an overreaction in sentiment that the market corrects over time.

Machine learning models can handle complex interactions and incorporate multiple features, potentially revealing hidden patterns. In the following section, we integrate our sentiment model along with several other features into an aggregated numerical pipeline. Instead of predicting log-returns, we shift our focus to using sentiment, log-returns, and additional features to predict whether the stock will increase in the given timeframes.

4.5.3 Results

Now that we have established there exists some predictability, we can perform the same cross-validation, except now we do not have any word encoding since all our features are numerical except for `ticker`.

Now that we have established there exists some predictability, we can perform the same cross-validation on each model. The difference here, is, that we no longer have any word encoding since all our features are numerical, with the exception of `ticker`. This column is expanded to a set of dummy variables, essentially one-hot encoding the tickers. Note that no longer consider Naive Bayes since it cannot handle negative inputs.

Once again, we check if the data is balanced. On the whole dataset, we have the following.

Class	10minWinner	30minWinner	60minWinner
1	168677	187250	197893
0	114566	95993	85350

Table 4.8: Trading Signal Count

As expected, longer durations mean a larger proportion of the positive class. Obviously, the likelihood that a long position is profitable increases with the timeframe of the investment.

Next we split the data into a training set and a validation set (20%) and the remaining 'training set' will later be further split randomly into K-fold train and test sets for cross-validation.

Timeframe	**Class**	**Class Weight**	**Ratio**	**Balanced**
60min	0	1.7198	0.431	yes
	1	0.705	0.431	
30min	0	1.5373	0.513	yes
	1	0.741	0.513	
10min	0	1.2815	0.679	yes
	1	0.8199	0.679	

Table 4.9: Class Weights and Ratios Across Different Timeframes

For each of these 3 timeframes, we will make use of 2 distinct sets of features totaling in 6 models per model type. In addition to Naive Bayes, we will also not consider CNNs since they perform better on either image- or textual data. In total, we will thus be comparing 24 models (120 when considering 5-fold cross-validation). Feature set 1 consists of the ticker dummy-variables, log-returns (close) as well as some sentiment-related features, including the (log) number of comments, (log) number of upvotes, and the sentiment of the full text (title and description). Feature set 2 consists of the same ticker dummy variables as well as log-returns. Here, each measure has been normalized to ensure consistent features. Essentially, we seek to show whether simple log-return based models can be enhanced using sentiment and related features. The following table provides an overview on the cross-validation results.

			Mean		Std.	
model	sentiment	timeframe	acc.	bal. acc.	acc.	bal. acc
LR	✓	60	0.696660	0.682075	0.128986	0.056008
LR	✓	30	0.667193	0.654269	0.113291	0.056820
LR	✓	10	0.594901	0.591055	0.075488	0.050164
LR		60	0.697830	0.683043	0.126264	0.054658
LR		30	0.669365	0.656033	0.110480	0.056414
LR		10	0.595744	0.591601	0.074382	0.050200
RF	✓	60	0.681448	0.630417	0.101460	0.035761
RF	✓	30	0.652171	0.612136	0.085013	0.033856
RF	✓	10	0.603458	0.586763	0.064113	0.040979
RF		60	0.657670	0.618863	0.085478	0.040967
RF		30	0.624571	0.597659	0.070246	0.037885
RF		10	0.583458	0.572266	0.048325	0.033694
GB	✓	60	0.724878	0.618882	0.095325	0.076331
GB	✓	30	0.697852	0.605067	0.085882	0.068711
GB	✓	10	0.651156	0.583735	0.067303	0.070548
GB		60	0.727041	0.622905	0.095993	0.085249
GB		30	0.701007	0.607844	0.084472	0.076842
GB		10	0.651606	0.583890	0.066266	0.070031
DNN	✓	60	0.723682	0.708350	0.003375	0.003190
DNN	✓	30	0.692521	0.678016	0.006523	0.003208
DNN	✓	10	0.616892	0.610207	0.004663	0.003927
DNN		60	0.714710	0.706459	0.005467	0.003206
DNN		30	0.689215	0.676087	0.003298	0.001302
DNN		10	0.617338	0.613325	0.011307	0.006298

As can be seen in the table, the accuracies are generally lower, which is to be expected given the increased complexity of the task at hand. While models such as Gradient Boosting which cannot take class imbalance into account, the balanced accuracies are overall above expectations. We take note of the higher timeframes always fare better across all models. Further, we note the DNN with the particularly exceptional performance of 70.8% cross-validated mean balanced accuracy. From these values, the DNN models generally fare better. Whether or not to include the sentiment features appears statistically insignificant given the standard deviations. Proceeding with the hour-based DNN model, we will backtest our strategy in the next chapter.

5 Evaluation

The final step is to backtest the DNN to evaluate its performance on historical stock price data. This involves applying the trained DNN to our out-of-sample validation set to simulate its real-world trading performance.

5.1 Backtesting

Backtesting is a critical process in evaluating the performance of a trading strategy. It involves simulating trades on historical data to assess how a trading model would have performed in real market conditions. In this project, we focused on backtesting a trading strategy developed using a Convolutional Neural Network (DNN), which was selected for its ability to capture complex patterns in time-series data.

The DNN model, trained on a combination of sentiment scores and stock market data, was used to generate trading signals. For a given model time frame, e.g. 60 minutes, should then have data of the following format.

	id	created	close	60minTime	60minPrice	ticker	signal
0	3zf0ir	15:40	103.145	16:39:59	103.470	AAPL	1
1	3zf2ed	15:55	199.235	15:56:38	204.061	SPY	1
2	3zfor4	18:27	107.900	18:46:18	108.790	NFLX	1
3	3zfov6	18:28	199.980	18:28:12	204.061	SPY	1
4	3zfzud	19:36	105.000	20:12:41	104.061	AAPL	0

Table 5.1: General format (dates removed for brevity)

For backtesting purposes, we can then retrieve the converse signals, namely covering a call position (2), and closing a long position (3). In essence, we have a secondary table with the same post id's and tickers, except the times and prices are different and we add 2 to `signal` in this table. Once we combine the two tables so that `60minTime` merges with the `created` column, and `60minPrice` merges with the `close` column, we can then re-sort by time (and id since there may be overlapping times). The following table illustrates the converted table.

	id	created	close	ticker	signal
0	3zf0ir	15:40	103.145	AAPL	1
1	3zf2ed	15:55	199.235	SPY	1
6	3zf2ed	15:56:38	204.061	SPY	3
5	3zf0ir	16:39:59	103.470	AAPL	3
2	3zfor4	18:27	107.900	NFLX	1
3	3zfov6	18:28	199.980	SPY	1
8	3zfov6	18:28:12	204.061	SPY	3
7	3zfor4	18:46:18	108.790	NFLX	3
4	3zfzud	19:36	105.000	AAPL	0
9	3zfzud	20:12:41	104.061	AAPL	2

Table 5.2: Dataframe for Backtesting (dates removed for brevity)

Now, we have a dataframe with clear trading signals that we can use for backtesting. While all code is available on github[26], the code for the backtesting logic can also be found in Appendix B.1. An important note to make, is that we will be selling and covering at the 'optimal' times. This, however, is a double-edged sword, since in case our prediction is wrong we will be selling or covering at the worst possible times. This approach simplifies are backtesting and due to this behavior, the subsequent results serve as a lower bound.

We initialize the portfolio at $1, but since our trading strategy allocates a certain percentage of the available cash for each trading signal, results are proportional to the initial investment. The following figure depicts the cumulative P&L curve using the predicted signals generated by the DNN model.

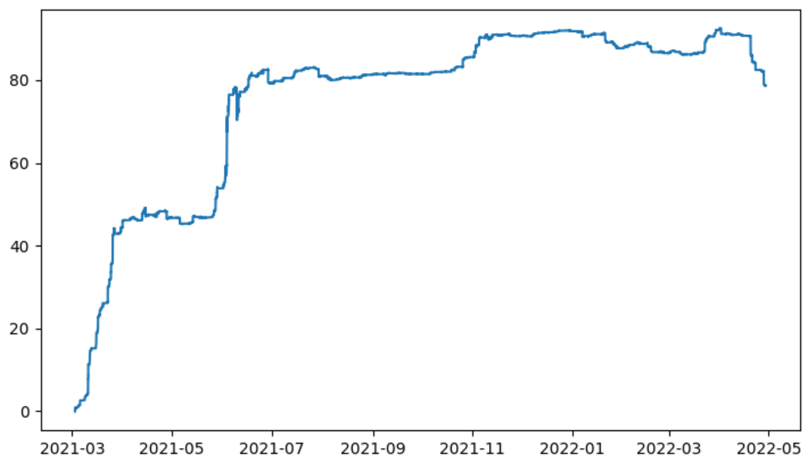


Figure 5.1: Cumulative P&L

In addition, the backtesting engine calculated several key performance metrics to evaluate the strategy, as summarized in the following table:

Metric (%)	**Value**
Total Returns	7862.60
Annualized Returns	4360.44
Volatility	1619.89
Sharpe Ratio	2.69
Average Trade Return	0.91
Win Ratio	60.52

Table 5.3: Performance Metrics using predicted signals

As seen in the table above, the win ratio is a bit lower than expected, since the 5-fold cross-validated balanced accuracy was 70.8%. This may be due to the random sampling in cross-validation as opposed to the contiguous validation data necessary to conduct a backtest. Nevertheless, our predictions yielded an accuracy of ”only” 60.93% on the validationset. Here, the win ratio aligns nicely since some returns happen to be zero. If we were to consider a profit of 0 as a ”win”, the win ratio increases to 61.3%. The total returns (7862.60%) for the predicted signals are exceptionally high, which suggests that, in this case, our strategy is highly effective in capitalizing on market movements, leveraging compounding effects over time. It is however, important to note that this is likely due to a combination of luck and general strategy, as there is a take-profit and stop-loss of 1% and because we trade very frequently. The volatility metric indicates large fluctuations in the portfolio value, underscoring the inherent risk associated with the strategy. Managing this high volatility is crucial for improving the strategy’s robustness and ensuring sustainable performance, though the sharpe ratio suggests that the trade-off is worth it. Additionally, it is likely that the high volatility may be caused by GME which fluctuated significantly throughout 2021 in particular.

The average trade return is higher than expected, since the theoretical maximum would be 1% with this strategy and because we only have an accuracy of 60.93%.

In summary, while the DNN-based trading strategy demonstrates the potential for high returns, the metrics reveal significant risks associated with the high volatility environment at the time and it is unlikely that results this exceptional can be reproduced in live trading scenarios, but this model and strategy still shows great promise for future profitability.

6 Discussion

The DNN model applied in this study demonstrates that stock price patterns can be predicted to some extent. While the granger tests showed potential for sentiment as a predictor of stock price movements, little evidence was found in practice for the models discussed in this paper. There are several avenues for enhancing the model and extending the research.

In order to analyze sentiment and perform cross-correlation analysis and granger tests, we need to measure the sentiment. This is why we needed to build our final DNN model in two stages. However, one could argue, that had we simply used the text as features, we could skip this intermediate step and the model would likely improve as a consequence.

As we saw in Table 4.8, the rate of positive class instances increased as a function of the time frame, and models with higher time frames always had better accuracies as compared to lower time frames, which begs the question at what point increases in accuracy diminishes or even falls as time frame increases. Though, assuming accuracy only increases, one should also consider the effects on PnL since longer timeframes mean less compounding effects and thus profits, so there is some balance to be found. Cross-correlation analysis indicates that significant relationships exist over longer periods. Future models could incorporate data from extended time frames, such as daily or weekly intervals, to capture these longer-term trends more effectively.

Additionally, expanding the feature set to include more financial indicators and sentiment-related metrics could improve model performance. Features such as social media engagement metrics, sentiment momentum, and volatility indices could provide additional insights into market dynamics and enhance prediction accuracy. It is important to strike a balance between feature engineering and avoiding overfitting.

Conducting a comparative analysis with other sentiment analysis tools and trading strategies could provide a benchmark for evaluating the effectiveness of the proposed model. This analysis could include traditional technical analysis methods and other machine learning-based approaches, helping to contextualize the performance of the DNN-based strategy within a broader spectrum of financial analysis techniques.

Developing real-time trading models by integrating real-time data streams from social media and stock markets can provide more dynamic and responsive trading strategies. This approach requires robust infrastructure to handle high-frequency data processing and the implementation of real-time machine learning models capable of making quick and accurate trading decisions.

Using sentiment analysis for stock trading raises several ethical concerns. One primary issue is the potential for market manipulation. Online communities, such as WallStreetBets, can significantly influence stock prices through coordinated efforts, leading to artificial inflation or deflation of stock values. This phenomenon can harm individual investors who may not fully understand the underlying market dynamics and are influenced by the hype.

Another ethical consideration is the transparency and fairness of trading strategies. Advanced sentiment analysis tools may provide certain investors with an unfair advantage over others who do not have access to similar technologies. This disparity can exacerbate inequalities in the financial markets, where those with better tools and resources can disproportionately benefit.

Future research could expand this study by integrating sentiment analysis from multiple social media platforms. Platforms like Twitter, Facebook, and financial news sites offer additional data that could enhance the accuracy and robustness of sentiment-based trading models. By comparing sentiment across different platforms, researchers can develop more comprehensive models that better capture the overall market sentiment.

To summarize, exploring more sophisticated methods to improve model accuracy and stability is essential. Techniques such as ensemble learning, transformer models, and advanced neural networks could be investigated to enhance prediction performance. These methods could provide deeper insights into the complex relationships between sentiment and stock price movements.

7 | Conclusion

This project explored the potential of using sentiment analysis on WallStreetBets posts to predict stock price movements. By leveraging a combination of natural language processing techniques and various machine learning models, we aimed to capture the influence of online sentiment on stock market behavior.

The analysis revealed evidence for as well as against the use of sentiment scores as a predictor of stock price trends, demonstrating some correlations with market movements. The integration of sentiment analysis with historical stock data allowed us to generate trading signals that offered several profitable opportunities. The performance metrics from our backtesting results indicated that models incorporating sentiment and related measures were generally within a standard deviation of simpler counterparts.

However, the study had several limitations. The unique language and slang used in WallStreetBets posts posed challenges in sentiment analysis, necessitating the development of a custom sentiment pipeline. Future research could enhance preprocessing techniques and incorporate more sophisticated natural language processing models directly applied to text data. The current model uses an hourly time horizon for predicting stock movements, but cross-correlation analysis suggests that significant relationships exist over longer periods. Future studies could explore longer time frames, such as daily or weekly intervals, to capture these trends more effectively.

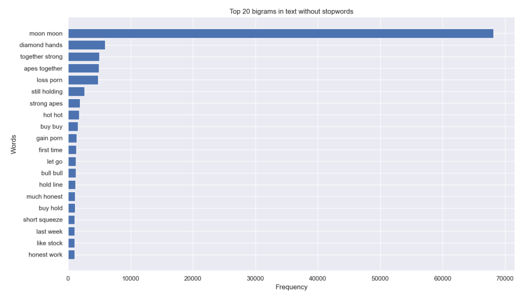
While the study utilized a comprehensive set of features, expanding the feature set to include more financial indicators and sentiment-related metrics could improve model performance. Future research should also focus on feature selection to avoid overfitting. This study primarily used historical data. Integrating real-time data streams from multiple social media platforms could enhance the model's responsiveness and accuracy. Developing infrastructure for real-time data processing and implementing real-time machine learning models would be a valuable extension of this work.

In conclusion, this project demonstrates the viability of sentiment analysis as a predictive tool for financial markets, particularly in the context of influential online communities like WallStreetBets. While the results show promise, further research is essential to harness the full potential of sentiment-based trading strategies. By continuing to refine these models and addressing the identified limitations, sentiment analysis can become a powerful asset in the toolkit of investors and financial analysts.

A Illustrations

A.1 Graphs

(a) Bigrams without stopwords



(b) Bigrams with stopwords

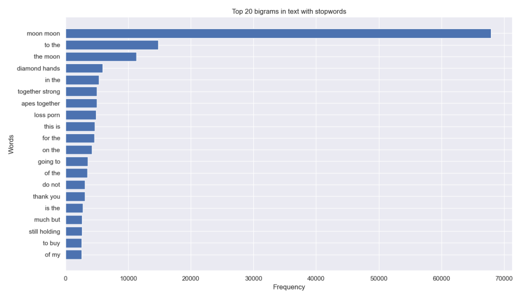
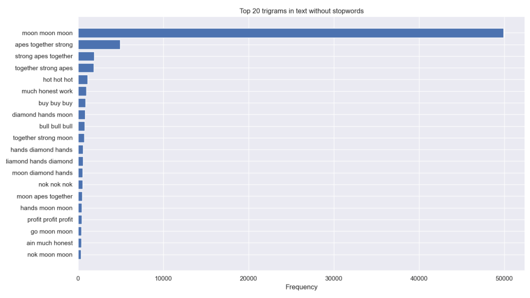


Figure A.1: Top 20 bigrams

(a) Trigrams without stopwords



(b) Trigrams with stopwords

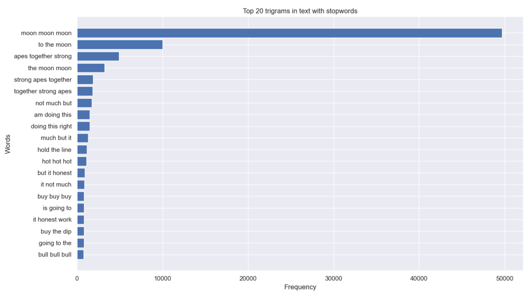


Figure A.2: Top 20 trigrams

A.2 Tables

Excluded ticker symbols							
A	YOU	IT	IPO	F	DD	C	EU
USD	AM	LINE	NOW	ALL	DAY	DO	ARE
X	USA	BY	OR	OUT	ON	LOVE	GOOD

FOR	NEXT	HAS	U	SO	BE	REAL	E
GO	WAY	EDIT	AI	SEE	AS	B	M
Z	O	K	R	S	V	J	G
H	L	W	D	T	EW	AN	HI
PUMP	FOLD	CASH	FUND	DEEP	TWO	EVER	LOW
FUN	MAX	WOW	MOD	CAR	BOOM	HEAR	API
OPEN	FAST	TV	HE	DTE	SMH	HIGH	WEED
SHIP	BUD	GAME	NICE	TECH	GRAB	WAVE	EAT
CARE	PAY	STEP	APPS	WELL	SNOW	TEAM	SPOT
SUN	MAN	TRUE	TIP	WOLF	ROOT	OF	YOLO
IS	SUB	CAN	LEG	ME	POST	RUN	SKIN
TOWN							

Table A.2: Excluded tickers for WallStreetBets NLP model. These are common words or acronyms that overlap with stock ticker symbols but are not relevant in the financial context.

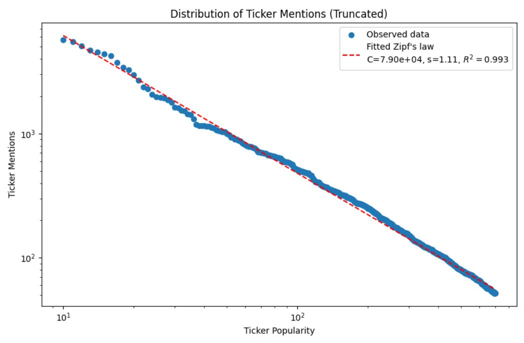


Figure A.3: Ticker mentions by rank (truncated)

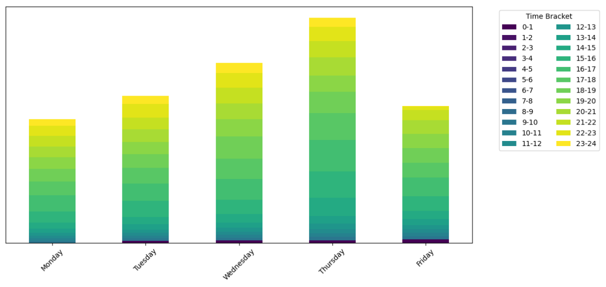


Figure A.4: Distribution of Open Market Hours

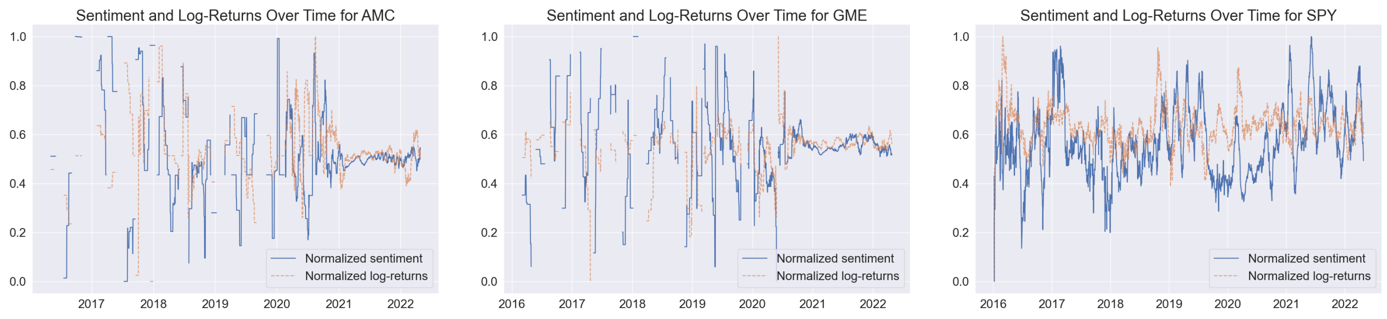


Figure A.5: Sentiment vs. Stock Prices for Various Companies (30-day moving average)

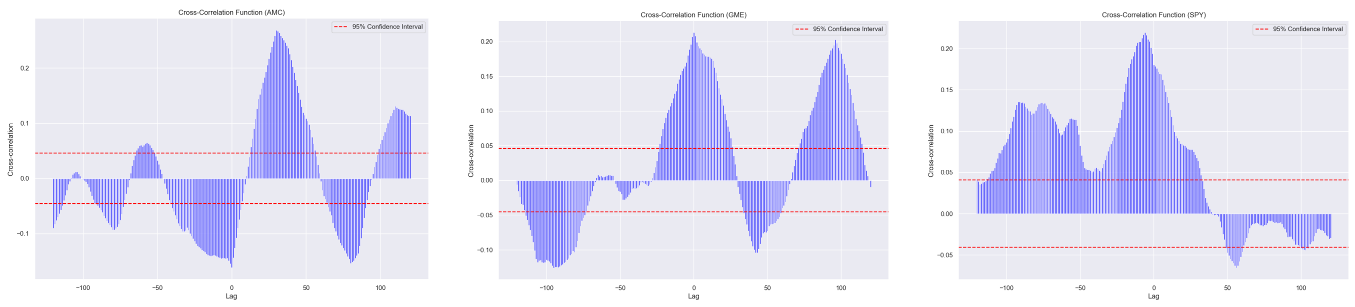


Figure A.6: Cross-Correlation between Sentiment and Stock Prices for Various Companies (30-day moving average)

Emoji	**Meaning**	**Replacement**
🚀	To The Moon!	moon
💎 🙌	Diamond Hands	diamond hands
🧻 🙌	Paper Hands	paper hands
🐂	Bullish trend	bull
🐻	Bearish trend	bear
📈	Positive trend	bull
📉	Negative trend	bear
🦍	Apes Together Strong	apes together strong
💰	Financial gain	profit
🤡	Foolish behavior	fool
🔥	Hot stock	hot
😭	Crying	cry
😎	Confidence	confident
😂	Funny	laugh
🍿	Watching drama	watch
🙈	Ignoring news	ignore
👀	Paying attention	watch
💀	Significant loss	loss
🎢	Volatility	volatile
🏴‍☠️	Risky investment	risk

Table A.1: Emoji Replacements for Sentiment Analysis

Stock	Type	Statistic	p-value	Lags	Obs	Crit. Val.	Stat Value
GME	Scores	-28.488000	0.000000	71	116438	-2.861600	-2.85e+01
	Prices	-2.421800	0.135700	63	116446	-2.861600	-2.42e+00
	Log-Returns	-33.204800	0.000000	71	116438	-2.861600	-3.32e+01
AMC	Scores	-20.960900	0.000000	59	59527	-2.861600	-2.10e+01
	Prices	-2.114300	0.238800	60	59526	-2.861600	-2.11e+00
	Log-Returns	-30.100500	0.000000	58	59528	-2.861600	-3.01e+01
SPY	Scores	-10.047600	0.000000	35	7340	-2.861900	-1.00e+01
	Prices	-1.139200	0.699200	16	7359	-2.861900	-1.14e+00
	Log-Returns	-15.823200	0.000000	29	7346	-2.861900	-1.58e+01

Table A.3: ADF Test Statistics

B | Code listings

Listing B.1: Backtesting Logic

```
class Backtest():
    def __init__(self, data, verbose=False, model=60):
        df = data.loc[:, ['id', 'created', 'close', f'{model}minTime', f'{model}minPrice
            ', 'ticker', 'signal']].copy()
        df_merged = pd.DataFrame({
            'time': pd.concat([df['created'], df[f'{model}minTime']]),
            'ticker': pd.concat([df['ticker'], df['ticker']]),
            'signal': pd.concat([df['signal'], df['signal']+2]), # Invert signal for '60
                minTime'
            'price': pd.concat([df['close'], df[f'{model}minPrice']]),
            'id': pd.concat([df['id'], df['id']]) # Add 'id' to the concatenated
                DataFrame
        })

        # Sort by 'time' and then by 'id' to break ties in time values
        df_merged = df_merged.sort_values(by=['time', 'id']).reset_index(drop=True)

        self.data = df_merged
        self.trades = []
        self.pnl_history = pd.DataFrame({'time': [], 'value': []})
        self.cash = 1
        self.portfolio = {}
        self.risk_per_trade = 0.01
        self.trade_api = alpaca_api.Alpaca()
        self.returns = pd.DataFrame({'time': [], 'value': []})
        self.entry = {}
        self.verbose = verbose

    def sell_all_positions(self, time):
        for ticker, position in self.portfolio.items():
            trade = self.trade_api.get_trades_by_time(ticker, time, time + pd.Timedelta(
                minutes=1), limit=1)
            if trade.empty:
                raise Exception('No trades within a minute')
            price = trade.p.iloc[0]
            self.cash += position * price
            self.portfolio[ticker] = 0

    def execute_trade(self, ticker, quantity, price, signal, time, old_price=None, id='
        00aaaa', i=0):
        if signal in [0, 1]: # Long or Short
            if (ticker not in self.entry) or ('prices' not in self.entry[ticker].keys()):
                self.entry[ticker] = {'prices': [price],
                                   'quantities': [quantity],
                                   'id': [id]}
            else:
                self.entry[ticker]['prices'].append(price)
                self.entry[ticker]['quantities'].append(quantity)
                self.entry[ticker]['id'].append(id)
        else: # Sell or Cover
            profit = (-quantity) * (price - old_price)
            if profit < 0:
                raise ValueError()
            pnl = pd.DataFrame({'time': [time], 'value': [profit]})
            returns = pd.DataFrame({'time': [time], 'value': [profit / abs(quantity *
                old_price)]})
            if self.pnl_history.empty:
                self.pnl_history = pnl
            else:
                self.pnl_history = pd.concat([self.pnl_history, pnl])
            if self.returns.empty:
                self.returns = returns
            else:
                self.returns = pd.concat([self.returns, returns])
            self.entry[ticker]['quantities'].pop(i)
            self.entry[ticker]['prices'].pop(i)
            self.entry[ticker]['id'].pop(i)

        self.cash -= quantity * price
        if ticker in self.portfolio:
            new_quantity = self.portfolio[ticker] + quantity
        else:
            new_quantity = quantity

        if new_quantity:
            self.portfolio[ticker] = new_quantity
        else:
            del self.portfolio[ticker]

    def print_portfolio_summary(self):
        print('--- Portfolio Summary ---')
        print('Cash:', self.cash)
        print('Positions:')
        for symbol, quantity in self.portfolio.items():
            print(symbol + ':', quantity)

    def plot_pnl(self, name="pnl_plot"):
        time, value = self.pnl_history.time, self.pnl_history.value
        fig, ax = plt.subplots(figsize=(10, 6))
        ax.plot(time, value)
        ax.set_xlabel('Date', fontsize=14)
        ax.set_xticklabels(ax.get_xticks(), fontsize=12) # Adjust the fontsize as needed
        ax.set_yticklabels(ax.get_yticks(), fontsize=12)
        plt.legend()
        plt.savefig(f"{name}.png")
        plt.close()

    def run_backtest(self):
        datasize = len(self.data)
        prev_row = self.data.iloc[-1]
        for i, row in self.data.iterrows():
            percent = (i / datasize) * 100
            sys.stdout.write(f'\rProgress: {percent:.2f}%')
            sys.stdout.flush()
            time, ticker, signal, price, id = row['time'], row['ticker'], row['signal'],
                row['price'], row['id']

            if time == prev_row['time'] and ticker == prev_row['ticker'] and signal ==
                prev_row['signal']:
                continue

            if self.verbose:
                print(f"Iteration {i}:")
                print(f"Cash Before: {self.cash}")
                print(f"Portfolio Before: {self.portfolio}")
            if ticker not in self.portfolio:
                self.portfolio[ticker] = 0
            old_price = None
            if signal == 0:
                quantity = -self.risk_per_trade * self.cash / price
                assert quantity <= 0
            elif signal == 1:
                quantity = self.risk_per_trade * self.cash / price
                assert quantity >= 0 and self.cash - quantity * price >= 0
            if signal == 2:
                i = np.where(np.array(self.entry[ticker]['id']) == id)[0][0]
                old_price = self.entry[ticker]['prices'][i]
                quantity = -self.entry[ticker]['quantities'][i]
                assert quantity >= 0 and self.cash - quantity * price >= 0
            elif signal == 3:
                i = np.where(np.array(self.entry[ticker]['id']) == id)[0][0]
                old_price = self.entry[ticker]['prices'][i]
                quantity = -self.entry[ticker]['quantities'][i]
                assert quantity <= 0

            if self.verbose:
                print(f"Signal: {signal}, Quantity: {quantity}")
            self.execute_trade(ticker, quantity, price, signal, time, old_price=old_price
                , id=id, i=i)
            if self.cash < 0:
                print("bankrupt.")
                break

            if i == len(self.data) - 1:
                self.sell_all_positions(time)
            if self.verbose:
                print(f"Cash After: {self.cash}")
                print(f"Portfolio After: {self.portfolio}\n")

            prev_row = row
        self.win_ratio = np.mean(self.pnl_history.value >= 0)

b = Backtest(df, verbose=False)

b.cash = 1
b.risk_per_trade = 0.01

b.run_backtest()
```

Bibliography

[1] Algaba A, Ardia D, Bluteau K, Borms S, Boudt K. Econometrics meets sentiment: An overview of methodology and applications. Journal of Economic Surveys. 2020;34(3):512–547.

[2] Gentzkow M, Kelly B, Taddy M. Text as data. Journal of Economic Literature. 2019;57(3):535–574.

[3] Bose B. NLP Text Encoding: A Beginner's Guide. Medium. 2021;Accessed: 2024-10-31. Accessible from: https://bishalbose294.medium.com/nlp-text-encoding-a-beginners-guide-fa332d715854.

[4] Ramos J, et al. Using TF-IDF to determine word relevance in document queries. In: Proceedings of the first instructional conference on machine learning. Citeseer; 2003. .

[5] Jolliffe IT. Principal Component Analysis. Springer Series in Statistics. 1986;Provides a comprehensive overview of PCA, including mathematical foundations, applications, and computational aspects.

[6] Golub GH, Loan CFV. Matrix Computations. 4th ed. Baltimore, MD: Johns Hopkins University Press; 2012. A comprehensive reference on numerical linear algebra, covering matrix factorizations including SVD, applications, and algorithms.

[7] Jurafsky D, Martin JH. Speech and Language Processing. In: Speech and Language Processing. 2nd ed. Prentice Hall; 2000. A foundational text in NLP that includes extensive discussions on POS tagging and its applications in various NLP tasks.

[8] Mikolov T, Sutskever I, Chen K, Corrado G, Dean J. Distributed Representations of Words and Phrases and Their Compositionality. Proceedings of the 26th International Conference on Neural Information Processing Systems - Volume 2. 2013;p. 3111–3119. This paper discusses Word2Vec, a prominent method for generating word embeddings, and its applications in capturing semantic relationships.

[9] Rao RPVN, Kumar A, Venkatakrishnan ABG, et al. A Survey on Word Embedding: Techniques and Applications. International Journal of Computer Applications. 2019;182(1):1–12.

[10] Hastie T, Tibshirani R, Friedman JH, Friedman JH. The elements of statistical learning: data mining, inference, and prediction. vol. 2. Springer; 2009.

[11] LeCun Y, Bengio Y, Hinton G. Deep Learning. Nature. 2015;521(7553):436–444.

[12] Shin H, et al. Deep Convolutional Neural Networks for Brain Image Analysis: A Comprehensive Review. NeuroImage. 2016;.

[13] Xu K, et al. Show, Attend and Tell: Neural Image Caption Generation with Visual Attention. In: Proceedings of the 32nd International Conference on Machine Learning; 2015. .

[14] Goodfellow I, Bengio Y, Courville A. Deep Learning. MIT Press; 2016.

[15] Bishop CM. Pattern Recognition and Machine Learning. Springer; 2006.

[16] Manning CD, Raghavan P, Schütze H. Introduction to Information Retrieval. MIT Press; 2008.

[17] Devlin J, Chang MW, Lee K, Toutanova K. BERT: Pre-training of Deep Bidirectional Transformers for Language Understanding. arXiv preprint arXiv:181004805. 2018;.

[18] Manning CD. Introduction to information retrieval. Syngress Publishing,; 2008.

[19] McCullagh P, Nelder JA. Generalized Linear Models. Chapman and Hall/CRC; 1989.

[20] Cox DR. The regression analysis of binary sequences. Journal of the Royal Statistical Society: Series B (Methodological). 1958;20(2):215–242.

[21] Hosmer DW, Lemeshow S, Sturdivant RX. Applied Logistic Regression. Wiley; 2013.

[22] Hastie T, Tibshirani R, Friedman J. The Elements of Statistical Learning: Data Mining, Inference, and Prediction. Springer; 2009.

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

[24] stuck_in_the_matrix, RaiderBDev. Reddit comments and submissions from 2005-06 to 2023-12; 2024. Accessed: 2024-06-03. https://academictorrents.com/details/9c263fc85366c1ef8f5bb9da0203f4c8c8db75f4.

[25] Newman ME. Power laws, Pareto distributions and Zipf's law. Contemporary physics. 2005;46(5):323–351.

[26] Christensen O. p9-1204e [code]. Gruppe 1204e; 2023. Accessible from: https://github.com/oliverch99/p9-1204e.