Bachelor's Studies

Field: Quantitative Methods in Economics and Information Systems

Antoni Łopacz
Student No.: 121572

Quantifying Investor Sentiment: A Machine Learning Analysis of Bitcoin-Related Reddit Posts

Bachelor's thesis
Under the scientific supervision of
Bartosz Pankratz, PhD
Written at the Institute of Econometrics

Warsaw, 2025

Contents

Chapter 1

Introduction

Cryptocurrencies, along with the underlying blockchain technology, have revolutionized and changed the financial market forever. Operating independently of centralized financial institutions, they provide a decentralized system that enables uninterrupted, continuous trading across global markets. Unlike the conventional banking system, which relies on intermediaries and is subject to government control, blockchain ensures immutability and peer-to-peer transactions. This independence enhances financial inclusivity, privacy, and global accessibility, but also introduces drawbacks such as price volatility, lack of regulatory oversight, and vulnerability to illicit activities.

The primary distinction between traditional and cryptocurrency markets lies in asset valuation. Traditional financial instruments derive their value from tangible assets, earnings, and established financial metrics, whereas cryptocurrency valuations are largely speculative, driven by factors such as potential future adoption and technological innovation. As a result, investor sentiment plays a significantly greater role in influencing cryptocurrencies prices compared to traditional markets. This highlights the potential value of leveraging market sentiment data for price forecasting, whether as a source of informational insight or as an input for predictive models. However, it also raises an important question, what exactly constitutes market sentiment data? Unlike historical market data, which is readily available, sentiment data is more difficult to obtain. Furthermore, there is no clear or universally accepted metric for measuring current market sentiment.

One approach to quantifying market sentiment is to analyze social media content using artificial intelligence tools, ranging from simple models to advanced transformer-based neural networks. This study adopts this method by applying natural language processing and sentiment analysis techniques to a dataset of Reddit posts about Bitcoin. Numerical representations of posts are generated, and models such as support vector machines and multilayer perceptron are employed to assess the emotional tone, classifying posts on a five-point scale ranging from highly negative to highly positive.

This study begins with an introduction to cryptocurrencies in Chapter 2, explaining their fundamental mechanisms, the transformative impact they have had on financial systems, and their advantages and limitations. Additionally, an overview of the cryptocurrency market, supported by relevant data, is provided to contextualize its significance. Chapter 3 then focuses on the dataset used in this study, detailing its origin, structure, and the processes involved in its creation and pre-processing, with particular emphasis on the distribution of the target variable. In Chapter 4, the theoretical foundations underpinning this study are examined, offering an in-depth analysis of the architectures, models, and algorithms employed. The methods for transforming Reddit posts into numerical representations for sentiment classification are explored, along with the models used to derive sentiment labels from these representations, with key concepts explained intuitively and supported by necessary mathematical formulations. Finally, Chapter 5 presents and analyzes the experimental results, evaluating model performance across different architectures, highlighting their strengths and weaknesses, and comparing their effectiveness in sentiment prediction to draw conclusions based on the findings.

Chapter 2

Cryptocurrencies

In this chapter, cryptocurrencies will be examined, including their origins and mode of operation. The mechanisms of decentralization and blockchain technology will be described and various data about the cryptocurrency market will be presented. The chapter will conclude by outlining the purpose of this study and presenting the research hypotheses that guide the analysis.

2.1. Understanding Cryptocurrencies

Cryptocurrencies, which can be defined as digital currencies secured by cryptography (Binance Academy, 2023), are a relatively recent innovation that has revolutionized the financial landscape by introducing a new type of currency with distinctive features that set them apart from traditional currencies. A fundamental characteristic of cryptocurrencies is decentralization, which allows users to maintain full control over their assets by eliminating the need for a central authority, thereby reducing the risk of manipulation or centralized control. Blockchain technology further strengthens these systems by providing transparency and immutability, as all transactions are recorded on a public, tamper-proof ledger where modifications are virtually impossible. Additionally, their borderless nature facilitates seamless global transactions and remittances, making them particularly appealing for international use. Many cryptocurrencies also possess a limited supply, such as Bitcoin, which is capped at 21 million coins (Crypto.com, 2025). This inherent scarcity mitigates inflation and enhances cryptocurrencies potential as a store of value, contributing to sustained demand over time.

2.2. How Cryptocurrencies Work

Understanding how cryptocurrencies work requires a basic understanding of cryptography, its role in securing digital transactions, and blockchain, the core technology that forms the

foundation of the entire system. Cryptography can be defined as the practice of creating and implementing coded algorithms to protect and obscure transmitted information, allowing access only to those with the required authorization and decryption ability (IBM, 2025). Cryptocurrencies employ cryptographic techniques, including hash functions, public-key cryptography, and digital signatures, to secure transactions, ensure data integrity, and regulate the creation of additional units. The specifics of this topic fall beyond the scope of this study. However, to provide a basic understanding of cryptographic applications, the core algorithm SHA-256, or Secure Hash Algorithm 256-bit, will be introduced.

2.2.1. Secure Hash Algorithm (SHA-256)

SHA-256 is a cryptographic hash function developed by the National Security Agency (NSA) and first published in 2001 (Golden, 2020). It transforms input data of any length into a fixed-size, 256-bit hash value, essentially generating a sequence of 256 ones and zeros for any given input. However, its two key features make it both highly useful and secure. The first feature is its use of logical operations to process input data, ensuring that even a slight change in the input produces a drastically different hash, a property known as the avalanche effect (Upadhyay et al., 2022). This makes it highly effective for verifying data integrity, as any alteration to the input data results in a completely different hash value. The second feature is its resistance to preimage attacks, meaning it is computationally infeasible to reconstruct the original input from its hash output.

In the context of cryptocurrencies like Bitcoin, SHA-256 serves several critical functions. It is integral to the mining process, where it is used in Bitcoin's proof-of-work mechanism to validate new blocks and secure the network. Additionally, SHA-256 is responsible for generating unique transaction identifiers, ensuring that each transaction is securely recorded and cannot be altered. It also plays a crucial role in the creation of Bitcoin addresses, providing cryptographic security and anonymity. Furthermore, SHA-256 is used to hash each block in the blockchain, linking blocks together and ensuring the integrity and immutability of the ledger (FasterCapital, 2024).

2.2.2. Blockchain

Blockchain can be described as a special kind of database maintained by a distributed network of computers, called nodes. The origins of blockchain date back to the early 1990s when computer scientist Stuart Haber and physicist W. Scott Stornetta employed cryptographic techniques in a chain of blocks as a way to secure digital documents from data tampering (Binance Academy, 2024). The idea of using blockchain technology for cryptocurrencies originated with their inception, traceable to 2008, when an individual or group operating under the pseudonym

Satoshi Nakamoto published the whitepaper *Bitcoin: A Peer-to-Peer Electronic Cash System*, introducing Bitcoin as the first decentralized cryptocurrency (Nakamoto, 2008).

At its core, blockchain is a decentralized and distributed ledger technology designed to securely record and verify transactions across a network, and it operates as follows: every node maintains a local copy of the blockchain and updates it whenever a new block is added to the ledger. Each block contains cryptographic hash, examined in Section 2.2.1, that links it to the previous block, forming an immutable chain of records. Computer nodes independently verify the integrity and authenticity of transaction data using these cryptographic hashes. Through the application of the network's consensus mechanism, a set of rules that dictate how nodes reach agreement on the state of the blockchain, only valid transactions are recorded. The use of cryptographic hashes ensures that misbehaving nodes or attempts to validate invalid transactions are promptly identified and excluded from the network, preserving the integrity and reliability of the system.

While blockchain technology is primarily associated with cryptocurrencies, it is equally well-suited for recording various other types of digital data and has applications across a wide range of industries. Financial services have embraced blockchain for its security and immutability, with companies like American Express integrating it into payment networks. Beyond finance, blockchain is being used in supply chain management, such as De Beers' Tracr platform for tracking diamonds, and in healthcare, where solutions like Medicalchain enhance patient data security (IBM Blockchain, 2019). These examples highlight blockchain's broad potential beyond digital currencies.

2.3. Bitcoin Market

Current capitalization of the entire cryptocurrency market is more than $3.6 trillion, with Bitcoin being the largest, holding a market capitalization of over $2 billion (CoinMarketCap, 2025), establishing it as a globally recognized and widely adopted digital currency. The exact number of Bitcoin holders cannot be determined, but estimates suggest it exceeds 100 million. This is particularly impressive considering that the cryptocurrency market emerged just over a decade ago.

Bitcoin's inception in 2009 saw its value at a fraction of a cent. The first recorded transaction, which established its initial monetary value, valued one Bitcoin at approximately $0.0009 (Forbes Advisor, 2024). At the time of writing this thesis, specifically in December 2024, just 15 years after its inception, Bitcoin broke through the $100,000 mark for the first time (Investing.com, 2025).

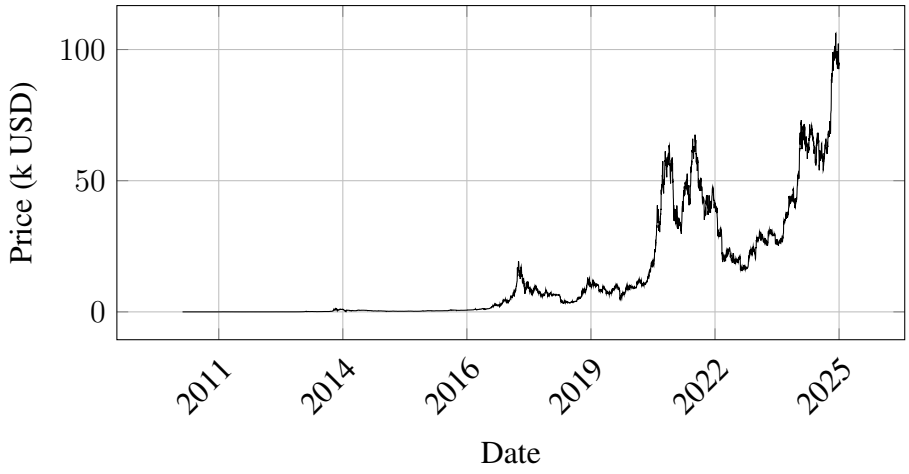

Figure 2.1: Bitcoin price over time.

Data Source: Investing.com (Investing.com, 2025).

Its rapid price growth and increasing popularity have made it a global phenomenon. Compared to last year, the number of Bitcoin millionaires soared by 111% to 85,400 (Henley & Partners, 2024b). The possibility of making quick and substantial profits attracts many individuals to invest in cryptocurrencies. Estimates indicate that as much as 34% of individuals in the largest global cohort of cryptocurrency holders, aged 24 to 35, own cryptocurrency (Henley & Partners, 2024a). However, most investors, drawn by the popularity and promise of potential profits, have lost money. According to research published in 2022 by the Bank of International Settlements (BIS), an estimated 73% to 81% of retail Bitcoin buyers are likely operating at a loss on their investments (Auer et al., 2023). It is crucial to recognize that Bitcoin's volatile nature, while capable of creating millionaires overnight, can just as easily lead to financial ruin.

2.4. Research Hypotheses

Given the data presented above, it is clear that every investor strives to eliminate the uncertainty and risk associated with Bitcoin investment. In an ideal scenario, this would be achieved by knowing the actual future prices of Bitcoin and securing immense profits. Unfortunately, unless potential investors know a flawless fortune teller or they are handed the Bitcoin equivalent of a time-transported sports almanac, like the one Biff Tannen received, knowing future prices remains an impossible dream. Instead, investors make do with what they have, forming predictions based on available sources, such as market data and past trends, to anticipate what lies ahead. This approach was borrowed from older, more traditional markets, such as the stock market. However, the cryptocurrency market exhibits significantly higher volatility. For instance, Bitcoin's historical volatility exceeds 75% annually, compared to just 15–20% for the S&P 500 index (Bonacua, 2024). The key distinction, however, lies in the fact that the

cryptocurrency market is not tied to physical assets. Their value is completely derived from factors such as utility and perceived value, making investor sentiment and belief in the underlying technology far more critical to this market.

This study explores the possibility of quantifying market sentiment by analysis of social media content, more precisely Bitcoin-related Reddit posts, using various machine learning models and exploring their effectiveness. It is hypothesized that simpler models can provide sufficient results while minimizing computational demands and associated financial costs. Additionally, it is expected that the tokenization and embedding methods applied will have a significant impact on the performance of sentiment classification models.

Chapter 3

Reddit-Based Sentiment Dataset

Data used in this research comes form the publicly available dataset, created by researchers from the University of Bologna, which is based on an existing dataset of Reddit posts and comments from Kaggle (Sero, 2021). The original dataset contains 250,569 posts and 3,756,097 comments collected from Reddit in August 2021 (Lexyr, 2021) and was filtered and pre-processed by the researchers.

3.1. Initial Data Pre-processing

The original Kaggle dataset did not include additional pre-processing steps related to post filtering or text cleaning. Therefore, a preliminary pre-processing phase was conducted to select only posts and comments that could potentially correlate with cryptocurrency price fluctuations. Posts tagged as 'deleted' or 'removed,' constituting approximately 49% of the original dataset, were excluded due to their lack of relevant textual content. Additionally, comments containing blacklisted words linked to scams, phishing, or advertisements (e.g., 'giveaway' or 'pump join') were also removed to ensure data relevance and integrity.

Subsequently, a series of traditional text normalization steps was applied to the remaining posts. These operations involved merging the title and body of each post, removing URLs and special symbols, and eliminating stopwords. This pre-processing phase reduced the dataset to 121,593 posts and 2,755,329 comments. Lastly, since the presence of bots is common on Reddit, a spam and bot detection procedure was implemented using the heuristic that assumes spam and bot-generated content often consists of short, frequently repeated sentences. Applying this filtering further reduced the dataset to 55,002 posts, ensuring higher data quality and reliability for analysis. (Seroyizhko et al., 2022)

3.2. Sentiment and Emotion Variables

The original Kaggle dataset includes comment-level sentiment annotations, however, no details are provided about how these annotations were produced. To enrich the dataset, researchers incorporated additional unsupervised labels by extracting sentiment and emotion features from Reddit posts and comments using multiple state-of-the-art tools. Among these enhancements is the feature “BERT-Sentiment,” which serves as the target variable for model training in this study. This feature ranges from 1 (negative) to 5 (positive) and was generated using a BERT model pre-trained on multilingual product reviews for sentiment analysis.

3.3. Further Data Processing

The distribution of the "BERT-Sentiment" variable in the dataset was initially highly imbalanced, with the majority of posts labeled as 1 (62.87%), a considerable portion labeled as 5 (26.86%), and the less frequent remaining labels 2 (4.61%), 4 (3.79%), and 3 (1.88%). To address this imbalance, a random downsampling approach was applied, where 14,773 rows with label 1 were removed, resulting in both labels 1 and 5 representing 41.97% each. The frequencies of the remaining labels, 2 (7.20%), 4 (5.92%), and 3 (2.94%), remained lower but increased relative to the original distribution. The final dataset comprises 35,196 observations, which were split into three subsets, namely the training set (21,117), validation set (7,039), and test set (7,040).

Chapter 4

From Text to Sentiment Prediction

In Chapter 4, a detailed description of the analytical tools and techniques used in the experiments will be presented. All models utilized in this study can be categorized into four main architectures, representing different combinations of the embedding methods (Word2Vec, BERT) and the machine learning models employing these embeddings as inputs (support vector machine, multilayer perceptron). First, the tokenization and embedding generation processes will be explained, focusing on Word2Vec and the transformer-based model BERT. This section will describe how meaningful numerical representations of Reddit posts are obtained. Subsequently, the methods used to evaluate sentiment from these numerical representations will be discussed, including an explanation of the support vector machine and the multilayer perceptron.

4.1. Word Embedding

Computers cannot inherently understand words, thus, words must be represented numerically for analytical tasks. However, simple word indexing with integers is insufficient, as it fails to capture the semantic depth and associations that words carry. When hearing a word, people intuitively grasp meanings that are not explicitly stated. For example, “king” is often associated with concepts like “male,” “monarch,” and “powerful.” To enable computers to capture such relationships, words can be represented in continuous vector space using an appropriate mathematical function. Therefore, the key point is defining a mapping between words and their real-valued representations so the meaning and features of the word can be captured.

In the following sections, two different ideas for obtaining this mapping will be explained, but first, the conceptual differences between them will be briefly examined. Both Word2Vec and the BERT model generate word representations, but the information encoded in the vector differs.

Word2Vec captures semantic and syntactic relationships by generating a single, fixed vector for each word. Encoding the semantics of a word involves capturing its meaning, the concepts

it represents, and its relationships with other words. This process reflects how words relate to broader contexts, such as associations with synonyms, antonyms, or conceptual similarities. Furthermore, encoding the syntactics involves capturing the structure of a given language, reflecting grammatical aspects such as word order, tense, and subject-verb agreement. To conclude, training a Word2Vec model can be viewed as creating a dictionary where each word is represented by a numerical vector. Each word's "definition" is expressed as a set of numbers, with similar words having closer vector representations, allowing the model to identify linguistic patterns and similarities effectively.

The more complex BERT architecture introduces a different approach to numerical word representation. Instead of assigning a single, static vector to a word, BERT generates contextual embeddings, which means the representation of a word varies depending on its use in a sentence and surrounding context. This allows BERT to capture both semantic relationships and syntactic structures, while also providing a deeper contextual understanding by considering the entire sentence during encoding.

4.2. Tokenization

Before delving deeper into proper embedding generation, the concept of tokenization must be introduced. Tokenization is a process of dividing text into smaller units, called tokens, which can be transformed into numerical vectors. While it can be intuitively described as splitting a sentence into words, this is a simplification, as tokens can vary significantly depending on the tokenization approach. Architectures used in this study differ in the way tokens are generated and utilized.

4.2.1. NLTK Word Tokenizer

For architectures based on the Word2Vec model, the word tokenizer recommended by the NLTK library, `nltk.tokenize.word_tokenize`, was used. It is a rule-based tokenizer that applies a predefined set of rules to split text into tokens using a combination of unsupervised machine learning and regular expressions. In fact, it combines two tokenizers: first, the `PunktSentenceTokenizer` segments the text into sentences, and then an enhanced version of the `TreebankWordTokenizer` tokenizes each sentence into words. This two-step process ensures more precise sentence and word segmentation. (NLTK documentation, 2023c)

The `PunktSentenceTokenizer` is an unsupervised machine learning-based tokenizer that builds a model for sentence boundary detection. It identifies sentence breaks by learning patterns in text and develops a model for recognizing abbreviations, collocations, and words that frequently appear at the beginning of sentences. (NLTK documentation, 2023a)

The `TreebankWordTokenizer` uses regular expressions to tokenize text following the Penn Tree-

bank guidelines. It splits standard contractions (e.g., “don’t” → “do n’t”), treats most punctuation as separate tokens, separates commas and single quotes when followed by whitespace, and distinguishes periods at the end of sentences. (NLTK documentation, 2023b)

4.2.2. WordPiece Tokenization

Architectures based on the BERT model utilize WordPiece, a tokenization algorithm developed by Google. Unlike the previously described word-based tokenizer, WordPiece is subword-based, breaking text into smaller subword units rather than full words. This approach addresses several issues associated with word-based tokenization, such as the need for an enormous vocabulary size and the high number of out-of-vocabulary tokens (Hugging Face, nd).

The WordPiece vocabulary consists of a set of root words and subwords, containing approximately 30k tokens (Devlin et al., 2019), which is relatively small compared to the 3 million tokens in the vocabulary of the pre-trained Word2Vec model used in this study (Google Code Archive, 2013). This smaller vocabulary size offers significant advantages, including improved memory efficiency due to the reduced number of embeddings to store, and decreased computational complexity, as fewer trainable parameters are required during BERT model training.

The subword approach also improves generalization across similar words and reduces the need to learn separate embeddings for every variation. For example, the words “playing,” “played,” and “player” would not be tokenized into three separate tokens because WordPiece captures the common root “play” and appends the subword components “##ing,” “##ed,” and “##er.” This allows the model to effectively tokenize morphological variations and compound words, enhancing its ability to understand and represent linguistic structures more efficiently.

One of the key advantages of the subword approach is that, despite its relatively modest vocabulary size, the number of out-of-vocabulary tokens is minimal and often zero in practice. Since the WordPiece tokenizer is trained on a fixed vocabulary, a word-based tokenizer would replace any word absent from this vocabulary with an unknown word token (`[UNK]`), leading to significant information loss. The subword decomposition strategy overcomes this limitation by splitting unfamiliar words into smaller, recognizable tokens already present in the vocabulary, thereby retaining more information.

4.2.3. Tokenization Process

The WordPiece tokenizer operates by iteratively decomposing words into smaller units based on a pre-defined vocabulary. This vocabulary, created during model training by Google, was constructed by identifying the most frequently occurring subword units in a large corpus of text. These subwords include complete words as well as partial components, such as prefixes, suffixes, and infixes.

When tokenizing a new sentence, each word is initially treated as a single token and checked against the vocabulary. If a word is not found, it is split into smaller subwords using a longest-match-first strategy. The tokenizer iteratively selects the longest substring of the word that exists in the vocabulary and appends it as a token, continuing this process until the entire word is tokenized. To differentiate subwords

from complete words, subwords that do not appear at the beginning of a word are prefixed with "##." For example, the word "playing" would be tokenized into "play" and "##ing." In rare cases where a word cannot be decomposed into any recognizable subwords, it is assigned the `[UNK]` token. (Hugging Face, nd).

4.3. Word2Vec

Word2Vec is a neural network-based algorithm developed by Tomáš Mikolov and his team at Google, introduced in the 2013 paper *Efficient Estimation of Word Representations in Vector Space.* (Mikolov et al., 2013a). It is based on the concept of distributional semantics, which suggests that a word's meaning can be inferred from the words frequently appearing near it. The paper presented two shallow, two-layer neural network architectures, Continuous Bag of Words (CBOW) and Skip-Gram, designed to learn distributed word representations efficiently. They aim to obtain static vector representations by utilizing the conditional probabilities between a target word and its context words (the surrounding words in a specified window) in the training data, either by predicting the target word from its context (CBOW) or predicting the context words from the target word (Skip-Gram). This approach allows Word2Vec to capture semantic similarities by positioning contextually related words closer together in the vector space.

In this study, 300-dimensional pre-trained Word2Vec embeddings are used, trained by Google on approximately 100 billion words from the Google News dataset, resulting in a vocabulary of 3 million vectors representing various words and phrases (Google Code Archive, 2013). Therefore, further analysis will focus on the solutions and architecture used to generate them, specifically the Skip-Gram with Negative Sampling (SGNS) approach. The original Skip-Gram architecture will first be explained, followed by an examination of the Negative Sampling technique.

4.3.1. Skip-Gram

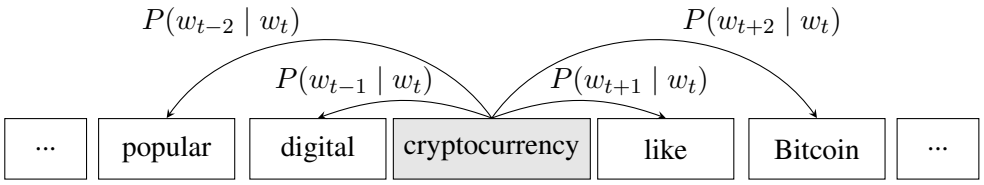

Figure 4.1: Skip-Gram context window example.

The Skip-Gram (SG) architecture focuses on predicting context words based on a target word, also called the center word. It trains on large text corpora by maximizing the likelihood of correctly predicting words that frequently occur within a specified context window around the center word. The context window size, a model hyperparameter, determines how many words around the target word are considered during training. For a window of size C, the number of calculated conditional probabilities is $2C$, as it

is calculated for C words before the center word and C after it. Context window size typically ranges between 1 and 10. The Skip-Gram training objective can be expressed as:

$$L(\theta) = \prod_{t=1}^{T} P(w_{t-C}, \ldots, w_{t-1}, w_{t+1}, \ldots, w_{t+C} \mid w_t) \tag{4.1}$$

$$L(\theta) = \prod_{t=1}^{T} \prod_{\substack{-C \leq j \leq C \\ j \neq 0}} P(w_{t+j} \mid w_t) \tag{4.2}$$

where θ represents all the model parameters, T is the corpus length, and w_t is the target word. For reasons such as easier computation, avoiding numerical underflow, and simplifying gradient calculations, the final minimized loss function is derived by transforming equation 4.2 into the negative log-likelihood form:

$$J(\theta) = -\frac{1}{T} \sum_{t=1}^{T} \sum_{\substack{-C \leq j \leq C \\ j \neq 0}} \log P(w_{t+j} \mid w_t) \tag{4.3}$$

where the probability of a context word w_{t+j} occurring given a target word w_t is:

$$P(w_{t+j} \mid w_t) = \frac{\exp(\vec{v}_{w_{t+j}} \cdot \vec{u}_{w_t})}{\sum_{w=1}^{V} \exp(\vec{v}_w \cdot \vec{u}_{w_I})} \tag{4.4}$$

where V is the vocabulary size. The numerator calculates the similarity score between the target word and the context word using the dot product, while the denominator sums over all words in the vocabulary to ensure the output is a valid probability distribution. This normalization step, known as the softmax function, converts raw similarity scores into probabilities, making the model's predictions interpretable and ensuring the sum of all probabilities equals one. However, in practice, using the softmax function would be computationally very expensive, with training complexity per each training example:

$$Q = 2C \times (D + D \times V) \tag{4.5}$$

where the dominating term is $D \times V$. It arises from calculating the denominator in equation 4.4, as the softmax function requires iterating over every word in the vocabulary (V being the vocabulary size) and computing the dot product with the target word vector (D representing the dimensionality of the word vectors). This results in high computational complexity, making the process expensive for large datasets.

To address this, the original Word2Vec paper proposed the use of hierarchical softmax, a softmax approximation method utilizing a binary tree structure, with the vocabulary organized as a Huffman Tree. This reduces the number of operations from being proportional to the vocabulary size to a logarithmic scale, significantly decreasing the training complexity 4.5:

$$Q = 2C \times (D + D \times \log_2(V)) \tag{4.6}$$

4.3.2. Skip-Gram with Negative Sampling

The solution used to train the `word2vec-google-news-300` model was a further enhancement to the hierarchical softmax approach, introduced in the paper *Distributed Representations of Words and Phrases and their Compositionality* (Mikolov et al., 2013b), published by Tomáš Mikolov and his team at Google approximately one month after the original Word2Vec paper. This method, called negative sampling, simplifies the computation by replacing the softmax function with sigmoid functions, sacrificing the probability distribution properties but significantly reducing the computational complexity while still achieving effective word representations.

This approach does not process all context words simultaneously but pairs each target word with individual context words. For each target word and its $2C$ positive context words, the model randomly samples k negative words from the vocabulary that are not within the context window. The training objective for a single word pair is to maximize:

$$L = \log P(w_c \mid w_t) - \sum_{j=1}^{k} \log P(w_{n_j} \mid w_t) \tag{4.7}$$

by maximizing $P(w_c \mid w_t)$, the conditional probability of the actual context word given the target word, while minimizing $P(w_{n_j} \mid w_t)$, the conditional probabilities of the k randomly sampled negative words. In this method, the conditional probability is also defined as the similarity score between two words and computed using the dot product, so the model aims to minimize the loss function:

$$J(\theta) = -\frac{1}{T} \sum_{t=1}^{T} \sum_{w_c \in C_t} \left[\log \sigma(\vec{v}_{w_c} \cdot \vec{u}_{w_t}) + \sum_{n=1}^{k} \mathbb{E}_{w_n \sim P(w)} \log \sigma(-\vec{v}_{w_n} \cdot \vec{u}_{w_t}) \right] \tag{4.8}$$

where k negative words are sampled from $P(w)$, which is a unigram distribution of words raised to the $3/4$ power. This means that each word's probability of being chosen as a negative sample is proportional to its frequency in the corpus, adjusted by taking the frequency to the $3/4$ power. This adjustment balances the selection process, ensuring that both common and rare words are appropriately represented during training.

When it comes to computational efficiency, the improvement arises from the fact that the number of comparisons required during training is significantly reduced, as only a limited number of negative samples are used instead of iterating over the entire vocabulary, resulting in a training complexity per example:

$$Q = 2C \times (D + D \times k) \tag{4.9}$$

To summarize, using negative sampling instead of hierarchical softmax reduces the number of operations from being logarithmically proportional to the vocabulary size to being directly proportional to the number of negative samples drawn, which for large datasets can be as low as 2 to 5, according to the authors' experiments.

4.3.3. Implementation Details

The Word2Vec word embeddings provided by Google and used in this study were generated using Skip-Gram with Negative Sampling architecture. Each tokenized Reddit post was converted into an array of these embeddings, and the final sentence representation was derived by averaging all word embeddings in the sentence array. This representation was subsequently used as input for the SVM and MLP models to derive sentiment information.

4.4. BERT

Another method for obtaining numerical sentence representations used in this study involves the pre-trained Bidirectional Encoder Representations from Transformers (BERT), specifically the ‘bert-base-uncased‘ model. This Encoder-Only Transformer model generates dynamic word embeddings by considering both left and right contexts of a word, producing context-sensitive sentence representations. It was developed by Google researchers Jacob Devlin, Ming-Wei Chang, Kenton Lee, and Kristina Toutanova and was first introduced in the 2018 paper *BERT: Pre-training of Deep Bidirectional Transformers for Language Understanding* (Devlin et al., 2019).

BERT significantly impacted natural language processing by leveraging transfer learning and providing a unified framework for multiple NLP tasks, achieved through pre-training on large datasets and fine-tuning for specific tasks. Its open-source release, including the architecture and pre-trained weights, further contributed to progress in the field by making state-of-the-art tools accessible to the research community.

This chapter explores the concept of contextual embeddings, tracing their progression from static embeddings like Word2Vec to advanced contextual representations exemplified by BERT. The BERT architecture, based on the Transformer framework, will be introduced with a focus on its Encoder-Only design and the self-attention mechanism that enables context embedding. The chapter also examines BERT’s pre-training process, highlighting its dual objectives of Masked Language Modeling (MLM) and Next Sentence Prediction (NSP), as well as the specifications of the BERT Base pre-trained model. Finally, it details how BERT is applied to support sentiment analysis tasks in this study.

4.4.1. Contextual Embeddings

Word embeddings generated by the Word2Vec model, described in Section 4.3, are static, meaning each unique token is assigned a single fixed vector representation. Consequently, identical tokens receive the same embedding regardless of context. However, language often exhibits polysemy, where a word can have multiple meanings depending on its context. To understand polysemy, consider the example of the word "great" used in different contexts:

“Bitcoin experienced a great surge in price last week.” → Large price increase.

“This is a great blockchain project with innovative security features.” → High-quality technology.

"Bitcoin had a great impact on the financial world." → Significant impact on global finance.

"I had a great experience using this exchange." → Good user experience.

"Oh great, my chart's looking less like gains and more like a ski slope." → Sarcastic use.

Furthermore, contextual relationships can extend beyond immediate neighboring words, influencing meaning across longer distances within a text, sometimes even between separate sentences. Consider the example of the word "model" in sentence:

"I work with models." → ?

By this sentence alone, it is impossible to determine whether the speaker is a photographer working for a fashion magazine or an engineer employed at a technology company.

Static embeddings fail to capture these context-dependent variations in word meaning, which limits their effectiveness in handling complex NLP tasks. To address this issue, the idea of contextual embeddings was introduced, where word representations dynamically adjust based on the sentence context. Regarding to example of "great", static Word2Vec embeddings would represent it with the same vector in all sentences, whereas in the case of contextual embeddings generated by BERT, a separate vector would be generated for the word in each sentence, with the cosine similarity between vectors reflecting the similarity between word meanings across contexts.

4.4.2. Evolution of Contextual Embeddings

Contextual embeddings evolved significantly over time, starting with the publication of *Semi-Supervised Sequence Learning* (Dai and Le, 2015), published by the Google Brain team in 2015, which introduced early contextual representations without relying on pre-trained word vectors. This paper introduces a method for semi-supervised sequence learning by pre-training an RNN language model on unlabeled data and then fine-tuning it on labeled data for specific tasks.

Another milestone occurred in 2017 with the publication of *Attention Is All You Need* (Vaswani et al., 2017), one of the most influential papers in the fields of machine learning and natural language processing, with over 150,000 citations as of January 2025 (Google Scholar, 2025). It revolutionized deep learning and sequence modeling with the Transformer architecture, which introduced a self-attention mechanism that enables parallelized computations, improves training efficiency, and, most importantly for contextual embeddings, effectively handles long-range dependencies.

A major breakthrough followed in 2018 with the Allen Institute for Artificial Intelligence and the University of Washington's publication of *ELMo: Deep Contextualized Word Representations* (Peters et al., 2018), which introduced ELMo (Embeddings from Language Models), generating dynamic word embeddings using a bidirectional LSTM and considering both left and right context. However, the training process was inefficient, and its reliance on LSTMs limited scalability for long-range dependencies. Additionally, the context was not truly bidirectional, as the model technically learned left-to-right and right-to-left context separately before concatenating them, resulting in a slight loss of contextual integrity.

Another important publication was OpenAI's *Improving Language Understanding by Generative Pre-Training (GPT)* (Radford et al., 2018), laying the foundation for subsequent developments in the GPT series of language models. It impacted the field by introducing the idea of generative pre-training, a technique where a model is first trained on a large corpus of text data in an unsupervised manner, learning general language patterns and structures, and later being fine-tuned on specific tasks. Soon after, BERT was developed, combining concepts from Transformers, ELMo, and GPT, introducing bidirectional context modeling and significantly advancing NLP performance.

To understand how BERT incorporates context into embeddings, it is essential to examine its architecture and the attention mechanism, which will be addressed in the following subsection.

4.4.3. Encoder-Only Transformer Architecture

The original Transformer architecture (Vaswani et al., 2017), was designed as a sequence-to-sequence model combining an encoder and a decoder to perform tasks like machine translation. The encoder processes the input sequence to generate contextualized representations, while the decoder generates the output sequence based on these representations and previous outputs. Over time, researchers discovered that the encoder and decoder could be utilized independently for various tasks. The encoder in the Transformer architecture is a fundamental component designed to process input sequences efficiently and capture long-range dependencies without relying on recurrence or convolution.

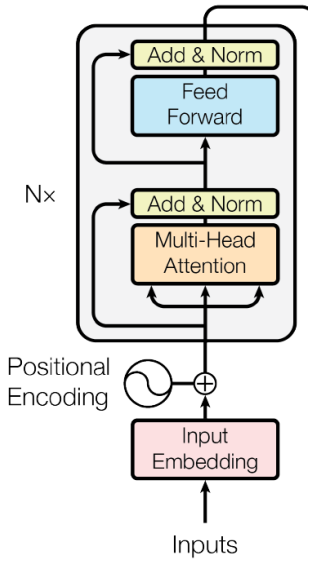

Figure 4.2: Encoder architecture.

Source: Attention Is All You Need (Vaswani et al., 2017).

It consists of a stack of identical layers, each featuring two core sub-layers: a multi-head self-attention mechanism and a position-wise fully connected feed-forward network. The self-attention mechanism enables the model to focus on various parts of the input sequence simultaneously, identifying relationships between all tokens within the maximum context window size, which, in the case of BERT, is

512 tokens (Devlin et al., 2019). An examination of how it functions is provided in Subsection 4.4.4. The feed-forward network, applied independently at each position, enhances the model's capacity for complex transformations. Residual connections and layer normalization are incorporated to improve gradient flow and stability during training. To compensate for the lack of inherent sequence-order awareness, positional encodings using sine and cosine functions are added to the input embeddings.

This architectural design enables substantial parallelization during training, addressing the limitations of sequential processing and significantly improving performance in tasks that require long-range contextual understanding.

4.4.4. Self-Attention Mechanism

The self-attention mechanism allows the model to evaluate the significance of each word in a sentence relative to every other word and then update the word embeddings, enabling words to pass information to whichever other words they are relevant to. It is important to note that the model does not actually process words but rather tokens, which were explained in Section 4.2. However, for the sake of clarity and intuitive explanation, the terms "words" and "tokens" will be used interchangeably.

This mechanism computes attention scores by generating three vectors for each word: Query, Key, and Value. These vectors are projections of the input embeddings into a subspace, where Query and Key have a dimension of d_k, and Value has a dimension of d_v. The attention score between two words is determined by the dot product of their Query and Key vectors, followed by a softmax function to normalize the scores into weights. These weights represent the significance of each word relative to others in the sentence. The weights are then applied to the Value vectors to produce a weighted sum, effectively capturing each word's contextual representation (Vaswani et al., 2017). This process can be intuitively understood as integrating information from various tokens and embedding it into each word's representation to capture contextual relationships effectively.

In practice, computations for every token are performed in parallel. All Query, Key, and Value vectors are packed into matrices Q, K, and V, respectively and the matrix of outputs is computed as:

$$\text{Attention}(Q, K, V) = \text{softmax}\left(\frac{QK^\top}{\sqrt{d_k}}\right)V$$

Additionally, the dot products are scaled by dividing by the square root of the dimension of the Key vectors ($\sqrt{d_k}$). This normalization prevents excessively large attention scores, ensuring numerical stability and improving the optimization process.

Multi-head attention

Multi-head attention is an extension of the self-attention mechanism that enables the model to focus on multiple aspects of the input simultaneously. Instead of computing a single attention score for each token pair, the model splits the input embeddings into multiple subspaces, each processed by a separate attention head. Each head independently computes self-attention using its own set of learned parameters, capturing different types of relationships, such as syntax, semantics, or long-range dependencies. The

outputs from all heads are then concatenated and linearly transformed to produce the final attention representation. This parallel processing allows the model to capture complex patterns and relationships in the data more effectively than a single-head attention mechanism.

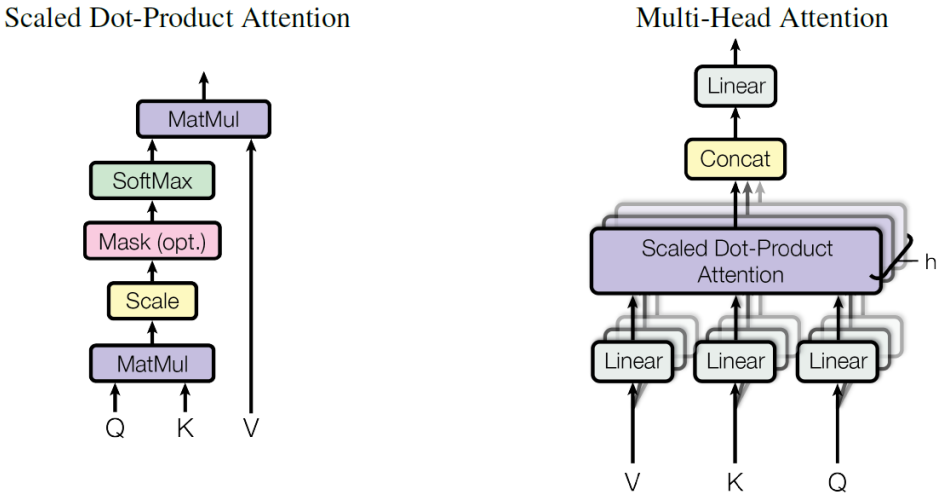

Figure 4.3: On the left is the scaled dot-product attention, the mathematical approach employed to calculate attention scores. On the right is the multi-head attention mechanism, which integrates multiple attention heads to capture diverse representation subspaces.

Source: Attention Is All You Need (Vaswani et al., 2017).

4.4.5. BERT Base Architecture

Building upon the foundational Transformer architecture, the BERT Base model distinguishes itself with specific configurations tailored for deep bidirectional language understanding. It comprises 12 encoder layers, each with 12 self-attention heads, facilitating the capture of intricate contextual relationships within text. The hidden size of 768 dimensions per token representation enables the model to encode substantial semantic information. Additionally, the intermediate size of 3,072 in the feed-forward networks allows for complex transformations and feature extraction. This architecture results in approximately 110 million parameters, balancing model capacity and computational efficiency (Devlin et al., 2019).

4.4.6. Pre-training Process

The BERT Base model leverages transfer learning by being pre-trained on a large dataset and requiring only fine-tuning for specific tasks, a process that significantly reduces computational demands compared to pre-training the entire model. BERT Base was specifically pre-trained on the BookCorpus, containing approximately 800 million words from books, and English Wikipedia, comprising around 2.5 billion words of text. This extensive dataset allowed BERT to capture a broad range of semantic

and syntactic relationships in language. The pre-training process utilized two complementary objectives, Masked Language Modeling (MLM) and Next Sentence Prediction (NSP), which enabled the model to effectively learn distinct aspects of language understanding. In the following part of this subsection, the input, its architectural design, and the training objectives will be examined.

Input Representation

BERT's input is purposefully structured to support learning through its pre-training objectives. Each input sequence begins with a special classification token `[CLS]`, followed by the tokens of the first sentence, a separator token `[SEP]`, the tokens of the second sentence, and concludes with another `[SEP]` token. Each token is represented by the sum of three embeddings: token embeddings, segment embeddings, and position embeddings. Token embeddings are derived using WordPiece tokenization, as discussed in Section 4.2.2. Segment embeddings differentiate tokens from the first and second sentences, assigning "A" for the first sentence and "B" for the second. Position embeddings encode the positional index of each token within the sequence, enabling the model to account for the sequential structure of the input effectively.

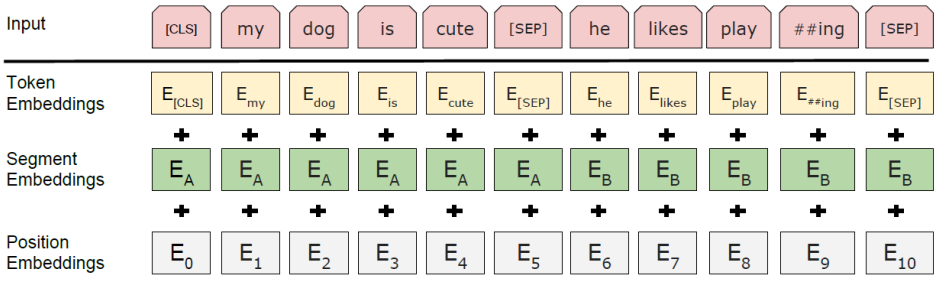

Figure 4.4: BERT input representation.

Source: BERT: Pre-training of Deep Bidirectional Transformers for Language Understanding (Devlin et al., 2019).

Masked Language Modeling (MLM)

MLM allows the model to grasp token-level relationships by predicting missing words in a sequence. In this task, 15% of the tokens in the input sequence are randomly selected for prediction. However, these tokens are not always replaced by the special `[MASK]` token. To maintain alignment with real-world data during fine-tuning, 80% of the selected tokens are masked, 10% are replaced by random tokens, and the remaining 10% are left unchanged. This randomness ensures that the model learns to predict tokens not solely based on the presence of the `[MASK]` token but also based on its understanding of surrounding context and intrinsic word relationships. For example, if the input sentence is "The cat sat on the `[MASK]`," the model uses its bidirectional nature to infer the masked token from both preceding ("The cat sat on") and succeeding ("the") contexts.

The output of the MLM task is a probability distribution over the entire vocabulary for each masked token. The predicted token is compared against the true token using a cross-entropy loss, which calculates the difference between the predicted probability distribution and the true one. The loss is averaged over all masked tokens in the batch, ensuring that the model learns to accurately predict the masked tokens.

Next Sentence Prediction (NSP)

NSP is a binary classifier designed to help the model understand sentence-level relationships, by identifying whether one sentence follows another logically. During pre-training, sentence pairs are constructed such that 50% of the time, the second sentence directly follows the first in the original corpus (labeled as `IsNext`), and 50% of the time, a random sentence from the corpus is selected as the second sentence (labeled as `NotNext`). For example, a pair like [“The sun is shining.”, “The sky is clear and blue.”] might represent an `IsNext` case, while [“The sun is shining.”, “I had breakfast today.”] would represent a `NotNext` case.

The output of the NSP is a binary label for each sentence pair, indicating whether the second sentence follows the first. The model produces logits for the two labels, which are converted into probabilities using a softmax function. The loss for this task is computed using cross-entropy, and averaged across the batch, ensuring that the model learns to discern sentence-level coherence effectively.

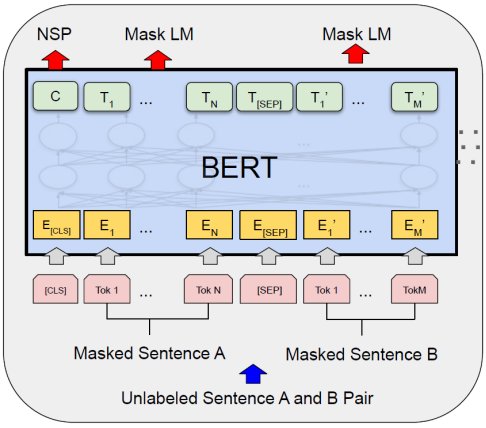

Figure 4.5: Pre-training procedure for BERT.

Source: BERT: Pre-training of Deep Bidirectional Transformers for Language Understanding (Devlin et al., 2019).

This three-layer representation of input data, combined with the dual training objectives, enables BERT to achieve a nuanced understanding of language. The MLM task captures token-level relationships, while the NSP task ensures sentence-level coherence, making BERT highly effective for a wide range of downstream natural language processing tasks.

4.4.7. Implementation Details

The embeddings utilized in this study were derived from the pre-trained BERT Base model, specifically bert-base-uncased, available at Hugging Face (Hugging Face, 2025).

Reddit posts were tokenized using the WordPiece tokenizer, with the `[CLS]` token prepended to each sequence. The `[CLS]` token acts as an aggregate representation of the entire sequence, effectively encoding its overall meaning. After processing the tokenized input through BERT's layers, the output corresponding to the `[CLS]` token was extracted as the post representation. This representation was subsequently used as input for the SVM and MLP models to derive sentiment information.

4.5. Support Vector Machine

Support Vector Machine (SVM) is a supervised machine learning algorithm designed primarily for classification tasks, developed in the early 1990s by Vladimir Vapnik and his coworkers at AT&T Bell Laboratories (The Franklin Institute, 2012). SVM builds upon the concepts of the maximal margin classifier and the support vector classifier, also known as the soft margin classifier. Its primary objective is to determine the optimal hyperplane that separates data points from different classes in a high-dimensional space. This hyperplane is chosen to maximize the margin, which is the distance between the hyperplane and the nearest data points from any class, known as support vectors. A notable strength of SVM is its ability to handle cases where classes are not linearly separable. It is achieved through the use of kernel functions, which map the data into higher-dimensional spaces, enabling the algorithm to learn non-linear decision boundaries. These attributes make SVM a highly versatile and effective tool for addressing a wide range of classification problems.

Section 4.5 provides a comprehensive introduction to SVM, beginning with the foundational concepts of the maximal margin classifier and the support vector classifier. The theoretical framework behind SVM is then explored, detailing how it constructs optimal decision boundaries and how kernel functions extend its capabilities to handle non-linearly separable data. Following this, the application of SVM to multi-class classification problems is examined, explaining the strategy for adapting a fundamentally binary classifier to multi-class settings. The chapter concludes with an overview of implementation details, outlining the practical steps taken to apply SVM models in this study.

4.5.1. Maximal Margin Classifier

The objective of SVM is to construct a classifier based on a separating hyperplane. However, when the data is linearly separable, infinitely many hyperplanes can separate the classes, requiring a criterion to select the most appropriate one. The SVM approach is rooted in the concept of the maximal margin classifier, which seeks to maximize the distance between the separating hyperplane, known as the decision boundary, and the nearest data points from each class, called support vectors. Since the objective of finding the separating hyperplane involves maximizing this distance, known as the margin, deriving a mathematical representation of the margin becomes essential.

To begin, let us examine the linear equation that will represent the decision hyperplane:

$$\vec{w} \cdot \vec{x} + b = 0 \tag{4.10}$$

where $\vec{w}$ is the normal vector to the hyperplane, meaning it is perpendicular to the decision boundary and defines its orientation, $\vec{x}$ represents any point on the hyperplane and b is a scalar that shifts the hyperplane away from the origin along the direction of $\vec{w}$. Therefore, an observation $\vec{x_i}$ would be classified into one class if $\vec{w} \cdot \vec{x_i} + b > 0$, indicating that it lies above the hyperplane, and into the other class if $\vec{w} \cdot \vec{x_i} + b < 0$, indicating that it lies below.

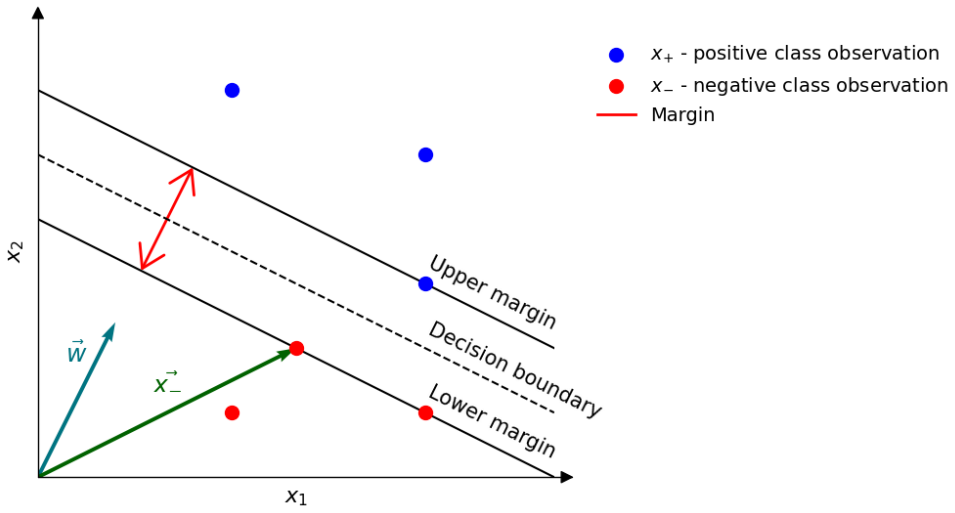

Figure 4.6: A simple representation of the SVM problem. Separation of two classes: positive (blue dots) and negative (red dots). The dashed line represents the decision boundary, while the two solid parallel lines denote the upper and lower margins. The vector $\vec{w}$ is normal to the decision boundary. The margin, depicted as the red line, represents the distance between the decision boundary and the nearest data points from each class.

All support vectors lie on the margin hyperplanes, with those from one class positioned on the upper margin and those from the other class on the lower margin. Consequently, all remaining observations are farther away from the decision boundary. By enforcing the constraint that the upper margin satisfies $\vec{w} \cdot \vec{x}_+ + b = 1$ and lower satisfies $\vec{w} \cdot \vec{x}_- + b = -1$, the classification rule during training can be expressed as:

$$\begin{aligned} \vec{w} \cdot \vec{x}_+ + b \geq 1 \quad &\text{for every positive observation } \vec{x}_+ \\ \vec{w} \cdot \vec{x}_- + b \leq -1 \quad &\text{for every negative observation } \vec{x}_- \end{aligned} \tag{4.11}$$

Equivalently, assuming that $y_t = 1$ for every $\vec{x}_+$ and $y_t = -1$ for every $\vec{x}_-$, all corresponding linear constraints are satisfied with:

$$y_t(\vec{w} \cdot \vec{x}_t + b) \geq 1 \quad \forall t \tag{4.12}$$

Training objective is maximizing the margin, which is equal to $\frac{2}{\|\vec{w}\|}$. It can be derived as the distance between the margin hyperplanes, however, here it will be presented purely as the distance between support vectors projected perpendicularly onto the margin hyperplanes.

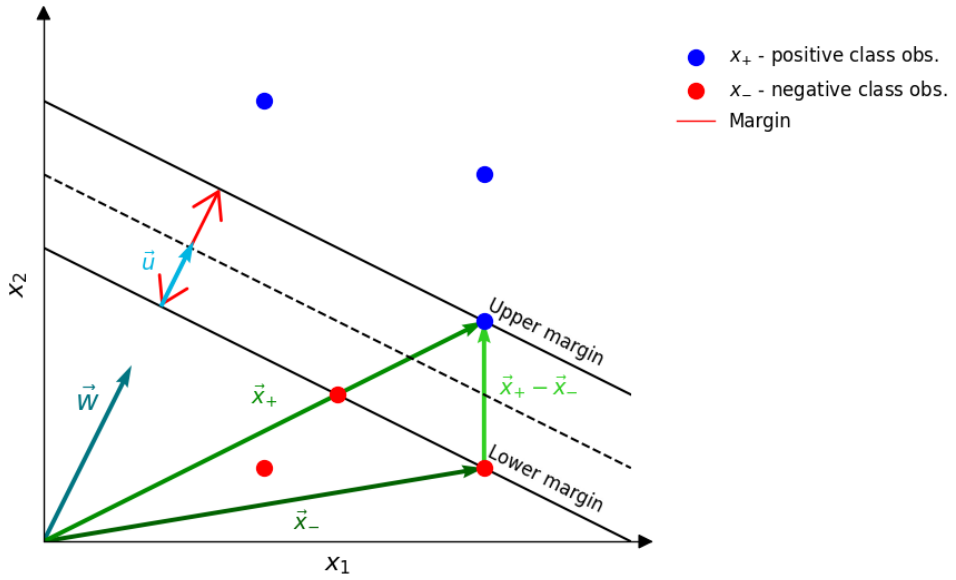

Figure 4.7: The margin can be determined by projecting the vector $\vec{x_-} - \vec{x_+}$ onto the normal vector $\vec{w}$.

If vector $\vec{x_-}$ represents a support vector from the negative class, and $\vec{x_+}$ a positive class, then $\vec{x_-} - \vec{x_+}$ represents a vector pointing from the lower margin to the upper margin. Nonetheless, the magnitude of this vector does not represent the distance between the margins, as it is not necessarily perpendicular to them. Instead, the dot product of $\vec{x_-} - \vec{x_+}$ and a unit vector perpendicular to the separating hyperplane would yield the margin, as it provides the magnitude of $\vec{x_-} - \vec{x_+}$ in the direction perpendicular to the hyperplane. By definition $\vec{w}$ is perpendicular, thus $\frac{\vec{w}}{\|\vec{w}\|}$ would be the perpendicular unit vector. Then the margin can be presented as:

$$\frac{\vec{w}}{\|\vec{w}\|} \cdot (\vec{x_-} - \vec{x_+}) \tag{4.13}$$

Considering Equation 4.11, for support vectors it holds that:

$$w \cdot x_+ = 1 - b \quad \text{and} \quad -w \cdot x_- = 1 + b \tag{4.14}$$

Thus, by combining Equations 4.13 and 4.14, the margin can be expressed as:

$$\frac{1}{\|\vec{w}\|} \cdot (1 - b + 1 + b) = \frac{2}{\|\vec{w}\|} \tag{4.15}$$

Equation 4.15 implies that the margin depends solely on the support vectors and is maximized by minimizing the magnitude of the vector $\vec{w}$. In practice, the term $\frac{1}{2}\|\vec{w}\|^2$ is minimized, as it simplifies the computational process while achieving the same objective.

The optimization is performed employing the method of Lagrange multipliers, under constraints 4.12 with the following objective function:

$$\begin{aligned} &\mathcal{L}(\vec{w}, b, \vec{\alpha}) = \frac{1}{2}\|\vec{w}\|^2 - \sum_{i=1}^{N} \alpha_i \left[y_i(\vec{w} \cdot \vec{x}_i + b) - 1 \right] \\ &\text{subject to} \quad \alpha_i \geq 0, \quad \forall i \in \{1, \ldots, N\} \end{aligned} \tag{4.16}$$

where α_i are the Lagrange multipliers associated with each constraint, and they are non-zero only for support vectors, indicating that the constraints are active solely for observations lying on the margin hyperplanes. This represents a dual optimization approach, in which minimization is performed with respect to $\vec{w}$ and b, while maximization is conducted with respect to α. Furthermore, as the objective is to find the extremum, the derivative of Equation 4.16 is taken with respect to $\vec{w}$ and b, set to zero, and the results are substituted, reformulating the equation into its final form for minimization with respect to α:

$$\begin{aligned} \mathcal{L}(\vec{\alpha}) &= \frac{1}{2}\sum_{i=1}^{N}\sum_{j=1}^{N}\alpha_i\alpha_j y_i y_j(\vec{x}_i \cdot \vec{x}_j) - \sum_{i=1}^{N}\alpha_i \\ \text{subject to} \quad & \alpha_i \geq 0, \quad \forall i \in \{1,\ldots,N\} \\ & \sum_{i=1}^{N}\alpha_i y_i = 0 \quad (\frac{\partial \mathcal{L}}{\partial b}) \end{aligned} \tag{4.17}$$

where the indices i and j represent the indices of training observations, used to account for interactions between them. As all vectors that are not support vectors do not contribute to the formulation of $\vec{w}$ and have $\alpha = 0$, for most iterations, Equation 4.17 evaluates to 0, effectively involving only the pairs of support vectors. The key aspects here are that $\vec{w}$ is a linear sum of the vectors in the sample set, and that optimization depends solely on the dot products of the support vectors.

The final solution is obtained by identifying the optimal values of α_i, which determine the support vectors. These support vectors are then used to calculate the optimal values of $\vec{w}$ and b, thereby defining the separating hyperplane and completing the classification task.

4.5.2. Support Vector Classifier

Support vector classifier, also known as the soft margin classifier, enhances the model to effectively handle real-world data that is not perfectly linearly separable. Unlike the maximal margin classifier, which requires all data points to be distinctly separated by a clear boundary, soft margin introduces flexibility by allowing some misclassifications and points to lie within the margin. This is achieved through the use of hinge loss, which quantifies the penalty for each data point based on its position relative to the margin, formulated in Equation 4.18. Specifically, hinge loss assigns zero loss to correctly classified points that lie outside the margin, a partial loss to correctly classified points within the margin, and a significant loss to misclassified points.

$$L_{\text{hinge}} = \max\left(0, 1 - y_i(\vec{w} \cdot \vec{x}_i + b)\right) \tag{4.18}$$

The mathematical formulation of soft margin SVM involves minimizing the average hinge loss across all training samples while simultaneously maximizing the margin by minimizing the squared magnitude of the weight vector $\vec{w}$, and can be formulated in its primal form as:

$$J = \frac{1}{N}\sum_{i=1}^{N}\max\left(0, 1 - y_i(\vec{w} \cdot \vec{x}_i + b)\right) + \lambda\|\vec{w}\|^2 \tag{4.19}$$

where λ is a regularization parameter that controls $\vec{w}$ and balances the trade-off between achieving a large margin and minimizing classification errors, allowing the model to be both robust and adaptable to

noisy, overlapping data. Graphically, hinge loss is depicted as a function that remains zero for confident correct classifications, increases linearly to 1 for points within the margin, and linearly escalates further for misclassified points. Without delving into details, the dual objective can be formulated as:

$$\begin{aligned} \mathcal{L}(\vec{\alpha}) &= \sum_{i=1}^{N} \alpha_i \; - \; \frac{1}{4\lambda} \sum_{i,j=1}^{N} \alpha_i \alpha_j y_i y_j (\mathbf{x}_i \cdot \mathbf{x}_j) \\ \text{subject to} \quad 0 \; &\leq \; \alpha_i \; \leq \; 1, \quad \sum_{i=1}^{N} \alpha_i \, y_i \; = \; 0 \end{aligned} \tag{4.20}$$

By incorporating these mechanisms, soft margin SVM provides a more practical and resilient framework for classification tasks in complex, real-world scenarios where data imperfections are inevitable.

4.5.3. Kernel Functions

Introducing the soft margin enabled the classification of data that is not perfectly linearly separable, while introducing the kernel functions allows for classification that is not linearly separable in any form. Kernel functions provide a powerful way to handle complex and non-linearly separable data by implicitly mapping points into a high-dimensional space without explicitly performing the transformation. Rather than defining a specific feature map $T(\vec{x})$ and working in an exponentially larger space, the kernel trick allows to compute inner products between transformed data points directly from their original vectors. This avoids the computational burden of computing and storing high-dimensional representations, enabling the model to capture rich, non-linear patterns. Intuitively, a kernel function $K(\vec{x}_i, \vec{x}_j)$ acts as a shortcut: instead of computing $T(\vec{x}_i) \cdot T(\vec{x}_j)$ explicitly, it produces the same result by applying a function of the original inner products $\vec{x}_i \cdot \vec{x}_j$. For example, a second-degree polynomial kernel can implicitly create interaction and quadratic terms that would normally require expanding the feature vector.

By incorporating the kernel function, the dual formulation presented in Equation 4.20, where the optimization problem depends only on inner products, is modified by replacing the term $\vec{x}_i \cdot \vec{x}_j$ with the kernel $K(\vec{x}_i, \vec{x}_j)$. This results in:

$$\begin{aligned} \mathcal{L}(\boldsymbol{\alpha}) &= \sum_{i=1}^{N} \alpha_i \; - \; \frac{1}{4\lambda} \sum_{i=1}^{N} \sum_{j=1}^{N} \alpha_i \, \alpha_j \, y_i \, y_j \, K(\vec{x}_i, \vec{x}_j) \\ \text{subject to} \quad 0 \; &\leq \; \alpha_i \; \leq \; 1, \quad \sum_{i=1}^{N} \alpha_i \, y_i = 0 \end{aligned} \tag{4.21}$$

Thus, by leveraging cleverly designed kernel functions, this modification allows for capturing relationships in a higher-dimensional space without the computational cost of explicitly constructing that space. This capability enables SVM to model highly flexible decision boundaries, making it an effective classification tool.

4.5.4. Radial Basis Function

The Radial Basis Function (RBF) kernel, also known as the Gaussian kernel, is a widely used kernel function and was implemented in all SVM models used in this study. It significantly enhances SVM

capability to classify complex and non-linearly separable data, as the RBF kernel allows to implicitly map input features into an infinite-dimensional space. This is achieved through the kernel trick, as outlined in Section 4.5.3, eliminating the need to explicitly perform any transformation.

The kernel's effectiveness stems from its ability to assign greater influence to nearby points while diminishing the impact of distant ones, resembling a weighted nearest neighbor model where the classification of a new observation is heavily influenced by its closest training points. Mathematically, the RBF kernel function is defined as:

$$K(\vec{x}_i, \vec{x}_j) = \exp\left(-\gamma \|\vec{x}_i - \vec{x}_j\|^2\right) \tag{4.22}$$

where γ is a parameter that controls the spread of the kernel and determines how much influence a single training example has. This modification not only preserves the computational efficiency but also enables the SVM to handle datasets with significant overlap and noise by adjusting the influence of each training point.

It has been stated that the implementation of the RBF kernel provides a representation of an infinite-dimensional relationship, thus, the process enabling this representation will be examined. The RBF kernel is effectively representing an infinite sum of polynomial kernels through the use of the Taylor series expansion. Specifically, the RBF kernel defined in Equation 4.22 can be expressed as as:

$$\exp\left(-\gamma(\|\vec{x}_i\|^2 + \|\vec{x}_j\|^2)\right) \exp\left(2\gamma \vec{x}_i \cdot \vec{x}_j\right) \tag{4.23}$$

where the term last term can be expanded as an infinite series:

$$\exp\big(2\gamma\, \vec{x}_i \cdot \vec{x}_j\big) = 1 + \frac{\big(2\gamma\, \vec{x}_i \cdot \vec{x}_j\big)}{1!} + \frac{\big(2\gamma\, \vec{x}_i \cdot \vec{x}_j\big)^2}{2!} + \cdots = \sum_{d=0}^{\infty} \frac{\big(2\gamma\, \vec{x}_i \cdot \vec{x}_j\big)^d}{d!} \tag{4.24}$$

Each term $\frac{\left(2\gamma\, \vec{x}_i \cdot \vec{x}_j\right)^d}{d!}$ captures a polynomial relationship of degree d, similar to the polynomial kernel $(\vec{x}_i \cdot \vec{x}_j + r)^d$ with $r = 0$. Consequently, as Equation 4.24 represents an infinite sum of these terms, it results in an infinite-dimensional feature mapping without explicitly computing each individual term. By implementing the RBF as the kernel function in Equation 4.21, the resulting objective function is:

$$\begin{aligned} \mathcal{L}(\boldsymbol{\alpha}) &= \sum_{i=1}^{N} \alpha_i \;-\; \frac{1}{4\lambda} \sum_{i=1}^{N} \sum_{j=1}^{N} \alpha_i\, \alpha_j\, y_i\, y_j\, \exp\Big(-\gamma\, \|\mathbf{x}_i - \mathbf{x}_j\|^2\Big) \\ \text{subject to} \quad & 0 \;\leq\; \alpha_i \;\leq\; 1, \quad \sum_{i=1}^{N} \alpha_i\, y_i = 0 \end{aligned} \tag{4.25}$$

4.5.5. One-vs-Rest Multiclass Classification

Up to this point in Section 4.5, the concepts have been discussed in the context of binary classification. However, this study addresses a five-class classification problem. In the following subsection, the One-vs-Rest (OvR) strategy, employed to tackle this challenge and adapt SVM for multi-class classification, will be examined.

The One-vs-Rest strategy decomposes a multi-class problem into several binary classification tasks. Therefore, when dealing with five classes, the OvR method involves training five separate SVM classifiers, each designed to distinguish one particular class from the remaining four. Therefore, the training objective of each classifier is to maximize the margin between its target class and the aggregate of the other classes, analogously to the clasical binary classification.

When making predictions on new data, the observation is evaluated by all five classifiers, each providing a decision score that reflects its confidence in the instance belonging to its respective class. Mathematically, each classifier i computes a decision function $f_i(\vec{x})$ for an input feature vector $\vec{x}$:

$$f_i(\vec{x}) = \vec{w}_i^\top \vec{x} + b_i \tag{4.26}$$

where $\vec{w}_i$ is the weight vector and b_i is the bias term for classifier i. This decision function measures the distance of the instance from the separating hyperplane specific to class i. A higher value of $f_i(\vec{x})$ indicates greater confidence that the instance belongs to class i. The class corresponding to the highest decision score is then selected as the predicted label for the observaiton:

$$\hat{y} = \arg \max_{i \in \{1,2,3,4,5\}} f_i(\vec{x}) \tag{4.27}$$

4.5.6. Implementation Details

All SVM models used in this study first standardize the numerical representations of Reddit posts, derived from Word2Vec and BERT, achieving zero mean and unit variance. The standardized features are then passed into an SVM classifier. Various SVM configurations are explored by tuning hyperparameters, including the kernel type (linear and RBF), the regularization parameter C (0.1, 1, 10), and γ, which controls the influence of individual samples (0.001, 0.01, 0.1, 1).

4.6. Multilayer Perceptron

Another approach used for obtaining sentiment labels from numerical Reddit post representations is the multilayer perceptron (MLP), also known as a feedforward network. An MLP is a type of neural network composed of fully connected dense layers that progressively transform the input data through many linear operations followed by activation functions, what enables the network to capture complex, non-linear relationships. The architecture is characterized as "multi-layer" because it includes an input layer, one or more hidden layers, and an output layer. In this study, the output layer is designed with five neurons, each corresponding to one of the five sentiment labels.

The training of an MLP involves optimizing its parameters through iterative updates guided by gradients computed from the loss function, progressively improving the model's performance. In Section 4.6, an overview of the training process will be presented, highlighting the key components that enable the MLP to learn from data, with a deeper examination of the loss function and optimization algorithm

utilized in this study. The Section will conclude with implementation details, outlining the specific configurations and techniques applied in this study.

4.6.1. MLP Training

The training process of an MLP revolves around optimizing its parameters to minimize a predefined loss function, thereby enhancing its prediction accuracy. This process begins with forward propagation, where input data is passed through the network to produce an output. The network's performance is then evaluated using a loss function, which quantifies the discrepancy between the predicted output and the actual target values. Next, backpropagation is employed to compute gradients of the loss function with respect to each weight and bias in the network. These gradients indicate the direction and magnitude of the adjustments needed to reduce the loss. An optimization algorithm, such as Adam, uses these gradients to iteratively update the parameters, gradually steering the network towards a set of optimal values. This cycle of forward propagation, loss computation, backpropagation, and parameter updating is repeated over multiple epochs, continuously refining the network's performance until the loss converges or other stopping criteria are met.

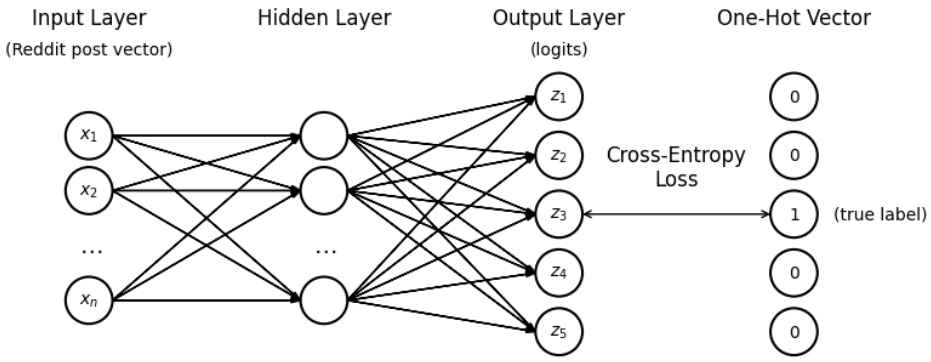

Figure 4.8: Simplified architecture of the implemented MLP models.

4.6.2. Categorical Cross-Entropy Loss

Categorical Cross-Entropy Loss is a loss function widely used for multi-class classification tasks, measuring the dissimilarity between the predicted probability distribution and the true distribution of target labels. During training, it penalizes the model heavily when it assigns a low probability to the correct class, reinforcing the need to predict higher probabilities for accurate labels. For a single training example, the loss is defined as:

$$L = -\sum_{i=1}^{C} p_i \log \hat{p}_i \tag{4.28}$$

where p_i represents the ground truth probabilities and $\hat{p}_i$ denotes the model-assigned probabilities for each of the C classes. In practice, the ground truth is provided as a one-hot encoded vector, where the

correct class has a probability of 1, while all others are 0. Consequently, for a given observation, only the log probability of the true class contributes to the loss, simplifying the expression to the negative logarithm of the predicted probability for that class. In practice, the loss is typically computed directly from the logits rather than explicitly applying the softmax function beforehand. When extended to a batch of N examples, the loss function is computed as the average over the batch:

$$L = -\frac{1}{N}\sum_{i=1}^{N}\log\left(\frac{\exp(z_{i,y_i})}{\sum_{j=1}^{C}\exp(z_{i,j})}\right) \tag{4.29}$$

where z_{i,y_i} represents the logit for the correct class of the i-th observation, $z_{i,j}$ is the logit for class j for that observation, and C is the total number of classes. This formulation effectively integrates the softmax transformation and the negative log-likelihood calculation into a single, numerically stable operation.

4.6.3. Adam Optimization Algorithm

The Adam optimizer, short for Adaptive Moment Estimation, is an optimization algorithm designed for training deep neural networks by combining the advantages of momentum-based and adaptive learning rate methods. Introduced in 2015 in the paper *Adam: A Method for Stochastic Optimization* (Kingma and Ba, 2015), it builds upon two foundational algorithms: AdaGrad, which adapts learning rates based on past squared gradients, and RMSProp, which stabilizes updates by maintaining an exponentially decaying average of past gradients. By integrating these approaches, Adam accelerates convergence while maintaining robustness by effectively managing gradient updates.

One of the key concepts is the momentum method, which smooths out updates by maintaining an exponentially decaying average of past gradients, which is given by:

$$m_t = \beta_1 m_{t-1} + (1-\beta_1) g_t \tag{4.30}$$

where g_t is the gradient at time step t and β_1 is the momentum hyperparameter. This approach addresses one of the main challenges in standard gradient descent, namely erratic updates caused by noisy loss surfaces or steep valleys. By maintaining a running average of gradients, the momentum method stabilizes parameter updates and reduces oscillations in high-curvature regions. When gradients consistently point in the same direction, momentum builds up speed, allowing the optimizer to move past small dips and traverse flat regions more effectively. This acceleration enhances optimization efficiency, helping the model escape local minima and achieve faster, more stable convergence.

Furthermore, an essential advancement is the introduction of adaptive learning rates, which allow the algorithm to adjust the learning rate for each parameter dynamically based on the magnitude of past gradients. This is achieved by computing an exponentially decaying average of past squared gradients:

$$v_t = \beta_2 v_{t-1} + (1-\beta_2) g_t^2 \tag{4.31}$$

where g_t is the gradient at time step t and β_2 is the decay rate hyperparameter. This formulation ensures that more recent gradients have a greater influence on the average while older gradients decay exponentially in their impact. The term v_t serves as an estimate of the uncentered variance of the gradients, effectively capturing the variability in the gradient signal for each parameter.

Adam combines these two approaches by using bias-corrected estimates of these moments,

$$\hat{m}_t = \frac{m_t}{1-\beta_1^t} \quad \text{and} \quad \hat{v}_t = \frac{v_t}{1-\beta_2^t} \tag{4.32}$$

to update the parameters according to

$$\theta_{t+1} = \theta_t - \alpha \frac{\hat{m}_t}{\sqrt{\hat{v}_t} + \epsilon} \tag{4.33}$$

where α is the learning rate and ϵ is a small constant for numerical stability.

4.6.4. Implementation Details

In this study, categorical Cross-Entropy Loss was used as the loss function, while the Adam optimizer with a learning rate of 0.001 and weight decay of 1e-5 was employed for parameter updates. A `ReduceLROnPlateau` scheduler was implemented to decrease the learning rate by a factor of 0.1 when the validation loss plateaued, with a patience of 2 epochs. Some models incorporated regularization and stabilization techniques, including dropout and batch normalization. Gradient clipping with a maximum norm of 5.0 was also applied. The training loop ran for up to 20 epochs, with early stopping triggered if validation loss showed no improvement for 3 consecutive epochs.

Chapter 5

Evaluation of Sentiment Classification Models

Chapter 5 focuses on the analysis of the results obtained in this study, evaluating the performance of different sentiment classification models. The chapter introduces the evaluation metrics used to measure model effectiveness, ensuring a clear understanding of the performance assessment criteria. Additionally, the naming conventions applied to distinguish between various model configurations are explained. Following this, the results for selected models from each of the four main architecture groups:

- Word2Vec + SVM
- Word2Vec + MLP
- BERT + SVM
- BERT + MLP

are presented, offering insight into their effectiveness across various evaluation metrics. Subsequently, the best-performing model from each group is analyzed in greater depth, providing a detailed assessment of their respective strengths and limitations.

5.1. Evaluation Metrics

Evaluation metrics are quantitative measures used to assess the performance of machine learning models, providing insight into their effectiveness in making accurate predictions. They help compare different models, diagnose potential weaknesses, and ensure that a model generalizes well to unseen data. In this section, the evaluation metrics employed in this study will be examined.

Accuracy

Accuracy is a metric that quantifies the overall correctness of a classifier by calculating the ratio of correct predictions to the total number of predictions:

$$\text{Accuracy} = \frac{\text{Number of Correct Predictions}}{\text{Total Number of Predictions}} \tag{5.1}$$

F1-Score

While accuracy provides an overall measure of correctness, it does not account for class imbalances, where one class may dominate the predictions. To address this, the F1-Score is used, which balances precision and recall. Precision measures the proportion of correctly predicted positive instances among all predicted positive instances, while recall measures the proportion of correctly predicted positive instances among all actual positive instances. Mathematically, these metrics are defined as:

$$\text{Precision} = \frac{\text{True Positives}}{\text{True Positives} + \text{False Positives}} \quad \text{Recall} = \frac{\text{True Positives}}{\text{True Positives} + \text{False Negatives}} \tag{5.2}$$

The F1-Score is the harmonic mean of precision and recall, ensuring that both are given equal importance:

$$\text{F1-Score} = 2 \times \frac{\text{Precision} \times \text{Recall}}{\text{Precision} + \text{Recall}} \tag{5.3}$$

Macro Average F1-Score

The Macro Average F1-Score is computed by first determining the F1-Score for each class and then taking the simple arithmetic mean over all classes. Formally, if there are C classes and $F1_i$ is the F1-Score for class i, then

$$\text{Macro Avg F1} = \frac{1}{C} \sum_{i=1}^{C} F1_i \tag{5.4}$$

This metric treats all classes equally, regardless of their frequency.

Weighted Average F1-Score

In contrast, the Weighted Average F1-Score calculates the F1-Score for each class and then averages these scores, weighting each by the number of true instances in that class:

$$\text{Weighted Avg F1} = \frac{\sum_{i=1}^{C} n_i \times F1_i}{\sum_{i=1}^{C} n_i} \tag{5.5}$$

where n_i is the number of true samples for class i. This approach accounts for class imbalance by giving more influence to classes with a higher number of instances.

5.2. Model Naming Convention

In this study, a systematic model naming convention was employed to succinctly convey each model's key attributes. The base naming structure follows the format:

`<EmbeddingType>_<ArchitectureType>_<Configuration>`.

where `<EmbeddingType>` indicates the input representation (e.g., W2V for Word2Vec, BERT), `<ArchitectureType>` specifies the model architecture (e.g., MLP, SVM), and `<Configuration>` provides model-specific details.

For MLP models, the `<Configuration>` follows the format:

`<LayerConfig>_<Activation>_<Extras>`.

where <LayerConfig> specifies the number of layers (e.g., 3L for a three-layer network), <Activation> denotes the activation function used (e.g., ReLU), and <Extras> includes additional components such as dropout (e.g., DO30 for 30% dropout) or batch normalization (e.g., BN).

For SVM models, the <Configuration> consists of key hyperparameters in the following order:

<C>_<Gamma>_<Kernel>.

where <C> represents the regularization parameter, <Gamma> specifies the gamma value, and <Kernel> denotes the kernel type used (e.g., linear, RBF).

5.3. Model Evaluation

Word2Vec + SVM

The results of SVM classifiers using Word2Vec-generated input predominantly fall within the 0.6–0.7 accuracy range, with the highest accuracy of 0.76 observed for $C = 10$ and $\gamma = 0.01$. However, despite this relatively fair accuracy, the macro F1-scores remain low, highlighting an imbalance in performance across different classes. In contrast, the weighted F1-scores are notably higher, indicating that the models perform significantly better on majority classes.

Model	Accuracy	Macro Avg F1	Weighted Avg F1
W2V_SVM_C_10_gamma_0.01_rbf	0.76	0.52	0.74
W2V_SVM_C_10_gamma_0.001_rbf	0.75	0.54	0.74
W2V_SVM_C_1_gamma_0.01_rbf	0.75	0.49	0.72
W2V_SVM_C_1_gamma_0.001_rbf	0.72	0.46	0.69
W2V_SVM_C_10_gamma_1_linear	0.69	0.48	0.68

Table 5.1: Evaluation metrics for Word2Vec SVM Models.

Word2Vec + MLP

Switching from an SVM to a multi-layer perceptron classifier did not significantly impact performance when using the same Word2Vec embeddings, yielding results within similar ranges. The most effective models were characterized by deeper architectures and the inclusion of batch normalization layers. While a simple one-layer MLP achieved only 0.68 accuracy and a macro F1-score of 0.29, expanding the network to five layers and integrating batch normalization improved accuracy to 0.77 and raised the macro F1-score to 0.52.

Model	Accuracy	Macro Avg F1	Weighted Avg F1
W2V_MLP_1L_ReLU	0.68	0.29	0.62
W2V_MLP_3L_ReLU	0.70	0.3	0.64
W2V_MLP_3L_ReLU_DO30	0.71	0.31	0.65
W2V_MLP_5L_ReLU	0.70	0.3	0.65
W2V_MLP_5L_ReLU_BN2	0.74	0.49	0.73
W2V_MLP_5L_ReLU_BN4	0.77	0.52	0.75

Table 5.2: Evaluation metrics for Word2Vec MLP Models.

BERT + SVM

Models that utilized embeddings generated directly from the pre-trained BERT-base model, without any fine-tuning, as input to an SVM, yielded relatively poor results. Accuracy remained around 0.45 regardless of hyperparameter settings. Both macro and weighted F1-scores were also quite low, indicating that the classifier did not effectively leverage the richer contextual embeddings provided by BERT. This outcome highlights that BERT-generated embeddings, when not fine-tuned for the specific task and used within a traditional SVM pipeline, do not necessarily translate into strong classification performance.

Model	Accuracy	Macro Avg F1	Weighted Avg F1
BERT_SVM_C_1_gamma_0.001_rbf	0.45	0.19	0.41
BERT_SVM_C_10_gamma_0.001_rbf	0.44	0.21	0.41
BERT_SVM_C_10_gamma_0.01_rbf	0.44	0.2	0.39
BERT_SVM_C_1_gamma_0.01_rbf	0.45	0.19	0.39
BERT_SVM_C_0.1_gamma_0.001_rbf	0.44	0.18	0.39

Table 5.3: Evaluation metrics for BERT SVM Models.

BERT + MLP

When the BERT-base model is fine-tuned alongside a small MLP head, performance improves significantly. Accuracy surpasses 0.80, reaching up to 0.85, with macro F1 exceeding 0.70 and weighted F1 above 0.80, making this the best-performing configuration overall. The highest scores are achieved with deeper MLPs incorporating batch normalization, however, even a simple one-layer MLP yields remarkably strong results. This underscores the effectiveness of allowing BERT's internal parameters to adapt to the classification task rather than relying on purely static embeddings.

Model	Accuracy	Macro Avg F1	Weighted Avg F1
BERT_MLP_1L_ReLU	0.82	0.63	0.81
BERT_MLP_3L_ReLU	0.85	0.72	0.85
BERT_MLP_3L_ReLU_BN	0.84	0.72	0.84
BERT_MLP_5L_ReLU_BN	0.85	0.73	0.85

Table 5.4: Evaluation metrics for BERT MLP Models.

5.4. Architectures Comparison

This section presents a comparative analysis of the best-performing models from each architecture group, assessing their overall effectiveness and differences in performance. Additionally, a detailed examination of label-specific results is conducted to evaluate how well each model classifies different sentiment categories.

Model	Accuracy	Macro Avg F1	Weighted Avg F1
W2V_SVM_C_10_gamma_0.001_rbf	0.75	0.54	0.74
W2V_MLP_5L_ReLU_BN4	0.77	0.52	0.75
BERT_SVM_C_1_gamma_0.001_rbf	0.45	0.19	0.41
BERT_MLP_5L_ReLU_BN	0.85	0.73	0.85

Table 5.5: Evaluation metrics for best-performing model from each architecture group.

An analysis of the overall metrics clearly indicates that fine-tuning BERT with an MLP produces the most robust results. While Word2Vec-based models perform worse, they still achieve strong results with significantly simpler architectures and lower computational requirements. Additionally, the findings highlight that using BERT solely as a fixed feature extractor, without adapting its internal parameters, fails to fully utilize its contextual capabilities for this task, as evidenced by the BERT-SVM model performing substantially worse than all other approaches.

Furthermore, a deeper examination of label-specific classification performance reinforces that the fine-tuned BERT model combined with an MLP outperforms all other models.

Class	#	W2V_SVM			W2V_MLP			BERT_SVM			BERT_MLP		
		Precision	Recall	F1-Score	Precision	Recall	F1-Score	Precision	Recall	F1-Score	Precision	Recall	F1-Score
1	2969	0.78	0.82	0.80	0.80	0.79	0.80	0.45	0.56	0.50	0.81	0.95	0.87
2	519	0.48	0.43	0.45	0.46	0.52	0.49	0.00	0.00	0.00	0.63	0.67	0.65
3	206	0.36	0.17	0.23	0.00	0.00	0.00	0.00	0.00	0.00	0.79	0.36	0.50
4	404	0.66	0.40	0.50	0.67	0.38	0.49	0.00	0.00	0.00	0.87	0.62	0.72
5	2942	0.81	0.86	0.84	0.79	0.88	0.83	0.46	0.51	0.48	0.94	0.85	0.89
Avg.	-	0.62	0.54	0.56	0.54	0.51	0.52	0.18	0.22	0.20	0.81	0.69	0.73
W. Avg.	-	0.75	0.77	0.76	0.74	0.76	0.75	0.38	0.45	0.41	0.86	0.85	0.85

Table 5.6: Label-specific evaluation metrics for the best-performing model from each architecture group.

The advantage of the fine-tuned BERT model with an MLP becomes particularly evident when evaluating the performance on labels 2, 3, and 4, which together accounted for just over 16% of the dataset. This model was the only one capable of capturing the essential linguistic nuances necessary to achieve strong classification results for these underrepresented classes, attaining an average F1-score of approximately 62%. The next best-performing model, Word2Vec SVM, lagged significantly behind with an F1-score below 39%, followed by Word2Vec MLP, which achieved less than 33%. Meanwhile, the non-fine-tuned BERT SVM model failed to classify any instances of these underrepresented labels.

A particularly informative case is label 3, which represents a neutral sentiment. Only two models managed to classify any observations from this category correctly on the test set: the Word2Vec SVM model, which correctly classified 17% of label 3 instances, and the fine-tuned BERT MLP, which achieved a recall of 36%.

Chapter 6

Conclusions

The objective of this study was to explore the possibility of quantifying market sentiment through the analysis of Bitcoin-related Reddit posts. As discussed in Chapter 2, it is evident that the cryptocurrency market is a well-established, significant, and rapidly growing financial sector and that the ability to derive insights from market sentiment presents a potentially lucrative advantage. In Chapter 4, various approaches for obtaining numerical representations of posts were examined, alongside detailed explanations of their application in sentiment classification. Finally, in Chapter 5, the results were analyzed, providing an evaluation of model performance and highlighting the differences between architectures and their effectiveness in sentiment prediction.

In this final Chapter 6, the research hypotheses formulated at the beginning of the study are assessed in light of the obtained results, providing a conclusive evaluation of their validity. Additionally, potential directions for future research are discussed, outlining opportunities for further exploration and improvement in sentiment analysis and its applications in financial markets.

6.1. Review of Research Hypotheses

This study was guided by two primary research hypotheses, each of which was examined through experimental evaluation and performance analysis of sentiment classification models. The following section presents an assessment of these hypotheses in light of the obtained results.

Utility of Simpler Architectures

The first hypothesis proposed that simpler models could yield sufficiently accurate results while reducing computational demands and financial costs. Although the best-performing model in this study was the fine-tuned BERT MLP model, representing the most sophisticated architecture, experiments demonstrated that simpler models based on Word2Vec embeddings also achieved competitive results. While deep Transformer-based models are necessary when achieving the highest classification effectiveness is the primary objective, particularly for capturing nuanced sentiment across all classes, simpler models remain viable alternatives for specific tasks. If the primary goal is to determine whether a text expresses

a positive or negative sentiment, then a model leveraging Word2Vec embeddings with an SVM or MLP classifier can provide adequate performance while significantly reducing computational overhead. This confirms that in certain scenarios, particularly those where computational efficiency is a priority, simpler models can be both practical and effective.

Impact of Tokenization and Embedding Methods on Model Performance

The second hypothesis posited that the choice of tokenization and embedding methods would have a significant impact on the performance of sentiment classification models. The results strongly support this assumption, as different embedding techniques led to substantial variations in model effectiveness. A striking example is the performance improvement observed when transitioning from static Word2Vec-generated embeddings to contextual embeddings derived from a fine-tuned BERT model. In the case of MLP classifiers, this transition resulted in an approximate 10 percentage point increase in accuracy, highlighting the critical role of contextual information in enhancing sentiment classification. However, the study also revealed that more complex embedding techniques do not always guarantee better performance. For instance, the SVM classifier performed significantly worse when using embeddings generated by a pre-trained BERT model without fine-tuning, demonstrating that adapting the model's internal parameters to the specific classification task is crucial for leveraging its full potential.

6.2. Future Research Directions

The findings of this study offer valuable insights for investors by demonstrating the significant role sentiment plays in the cryptocurrency market and its potential as a market indicator. One possible avenue for future research is integrating sentiment analysis into real-time data pipelines. By employing web scraping techniques to continuously collect social media content, sentiment classification could be performed almost instantaneously, enabling dynamic market monitoring. Such a system could assist traders by providing real-time sentiment indicators that reflect shifts in market mood, potentially serving as an early warning system for price fluctuations.

Additionally, sentiment data could be incorporated as an input feature in predictive models for price forecasting. Future research could explore the integration of sentiment metrics with traditional financial indicators, such as trading volume, volatility, and order book data, to enhance market prediction models. A more detailed analysis of the relationship between sentiment trends and price movements could provide deeper insights into how investor sentiment drives market dynamics.

Another promising research direction involves refining the embedding techniques used for sentiment classification. While this study utilized pre-trained Word2Vec embeddings, an interesting extension would be training a Word2Vec model directly on a cryptocurrency-related dataset. This could yield word representations more attuned to domain-specific terminology, potentially improving classification performance. Similarly, the poor performance of the pre-trained BERT model with an SVM classifier suggests that a fine-tuned BERT + SVM approach could be tested to determine whether adapting the model's internal parameters to the specific task would yield better results.

Bibliography

Auer, R., Cornelli, G., Doerr, S., Frost, J., and Gambacorta, L. (2023). Crypto trading and bitcoin prices: Evidence from a new database of retail adoption. BIS Working Papers 1049, Monetary and Economic Department, Bank for International Settlements. (Accessed: 11 January 2025).

Binance Academy (2023). What is cryptocurrency and how does it work? Available at: https://academy.binance.com/en/articles/what-is-a-cryptocurrency. (Accessed: 22 November 2024).

Binance Academy (2024). What is blockchain and how does it work? Available at: https://academy.binance.com/en/articles/what-is-blockchain-and-how-does-it-work. (Accessed: 21 December 2024).

Bonacua, A. (2024). Volatility differences across crypto assets. Available at: https://www.markets.com/education-centre/volatility-differences-across-crypto-assets/. (Accessed: 11 January 2025).

CoinMarketCap (2025). Today's cryptocurrency prices by market cap. Available at: https://coinmarketcap.com/. (Accessed: 23 December 2024).

Crypto.com (2025). How many bitcoins are there? Available at: https://crypto.com/bitcoin/how-many-bitcoins-are-there. (Accessed: 23 January 2025).

Dai, A. and Le, Q. (2015). Semi-supervised sequence learning. In *Advances in Neural Information Processing Systems*, volume 28, pages 3079–3087. Available at: https://proceedings.neurips.cc/paper/2015/file/7137debd45ae4d0ab9aa953017286b20-Paper.pdf (Accessed: 19 January 2025).

Devlin, J., Chang, M.-W., Lee, K., and Toutanova, K. (2019). Bert: Pre-training of deep bidirectional transformers for language understanding. In *Proceedings of the 2019 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies, Volume 1 (Long and Short Papers)*, pages 4171–4186. Available at: https://aclanthology.org/N19-1423.pdf (Accessed: 17 January 2025).

FasterCapital (2024). Sha-256: Cracking the code: Sha-256 and its role in bitcoin mining. Available at: https://www.fastercapital.com/content/

SHA-256--Cracking-the-Code--SHA-256-and-Its-Role-in-Bitcoin-Mining.html. (Accessed: 21 January 2025).

Forbes Advisor (2024). Bitcoin price history 2009 to 2024. Available at: https://www.forbes.com/advisor/in/investing/cryptocurrency/bitcoin-price-history-chart/. (Accessed: 7 January 2025).

Golden (2020). Sha-256. Available at: https://golden.com/wiki/SHA-256-XKEJ8AB. (Accessed: 14 December 2024).

Google Code Archive (2013). Word2vec project information. Available at: https://code.google.com/archive/p/word2vec/. (Accessed: 17 January 2025).

Google Scholar (2025). Ashish Vaswani Google Scholar page. Available at: https://scholar.google.com/citations?hl=en&user=oR9sCGYAAAAJ. (Accessed: 20 January 2025).

Henley & Partners (2024a). Crypto high-net-worth individuals: Multi-billion dollar opportunity. Available at: https://www.henleyglobal.com/publications/crypto-wealth-report-2024/crypto-high-net-worth-individuals-multi-billion-dollar-opportunity. (Accessed: 11 January 2025).

Henley & Partners (2024b). Crypto wealth report 2024. Available at: https://www.henleyglobal.com/publications/crypto-wealth-report-2024. (Accessed: 11 January 2025).

Hugging Face (2025). Bert base uncased model. Available at: https://huggingface.co/google-bert/bert-base-uncased. (Accessed: 19 January 2025).

Hugging Face (n.d.). Nlp course chapter 6: The huggingface tokenizers library. Available at: https://huggingface.co/learn/nlp-course/chapter6/6. (Accessed: 17 January 2025).

IBM (2025). Cryptography. Available at: https://www.ibm.com/think/topics/cryptography. (Accessed: 22 January 2025).

IBM Blockchain (2019). Blockchain beyond cryptocurrency. Available at: https://www.ibm.com/think/insights/blockchain-beyond-cryptocurrency. (Accessed: 24 January 2025).

Investing.com (2025). Bitcoin historical data. Available at: https://www.investing.com/crypto/bitcoin/historical-data. (Accessed: 21 December 2024).

Kingma, D. and Ba, J. (2015). Adam: A method for stochastic optimization. arXiv preprint arXiv:1412.6980. Available at: https://arxiv.org/abs/1412.6980 (Accessed: 24 January 2025).

Lexyr (2021). Reddit cryptocurrency data for august 2021. Available at: https://www.kaggle.com/datasets/pavellexyr/reddit-cryptocurrency-data-for-august-2021. Accessed: 26 November 2024.

Mikolov, T., Chen, K., Corrado, G., and Dean, J. (2013a). Efficient estimation of word representations in vector space. arXiv preprint arXiv:1301.3781. Available at: https://arxiv.org/abs/1301.3781 (Accessed: 18 January 2025).

Mikolov, T., Sutskever, I., Chen, K., Corrado, G., and Dean, J. (2013b). Distributed representations of words and phrases and their compositionality. arXiv preprint. Available at: https://arxiv.org/abs/1310.4546 (Accessed: 18 January 2025).

Nakamoto, S. (2008). Bitcoin: A peer-to-peer electronic cash system. Available at: https://bitcoin.org/en/bitcoin-paper. (Accessed: 22 December 2024).

NLTK documentation (2023a). nltk.tokenize.punktsentencetokenizer. Available at: https://www.nltk.org/api/nltk.tokenize.PunktSentenceTokenizer.html#nltk.tokenize.PunktSentenceTokenizer. Accessed: 12 January 2025.

NLTK documentation (2023b). nltk.tokenize.treebankwordtokenizer. Available at: https://www.nltk.org/api/nltk.tokenize.TreebankWordTokenizer.html#nltk.tokenize.TreebankWordTokenizer. Accessed: 12 January 2025.

NLTK documentation (2023c). nltk.tokenize.word_tokenize. Available at: https://www.nltk.org/api/nltk.tokenize.word_tokenize.html. Accessed: 12 January 2025.

Peters, M., Neumann, M., Iyyer, M., Gardner, M., Clark, C., Lee, K., and Zettlemoyer, L. (2018). Deep contextualized word representations. In *Proceedings of the 2018 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies*, volume 1, pages 2227–2237. Available at: https://www.aclweb.org/anthology/N18-1202/ (Accessed: 19 January 2025).

Radford, A., Narasimhan, K., Salimans, T., and Sutskever, I. (2018). Improving language understanding by generative pre-training. OpenAI. Available at: https://cdn.openai.com/research-covers/language-unsupervised/language_understanding_paper.pdf (Accessed: 19 January 2025).

Sero, P. (2021). Bitcoin reddit sentiment dataset. Available at: https://www.kaggle.com/datasets/paulsero/bitcoin-reddit-sentiment-dataset?select=reddit_sentiment_august2021.csv. Accessed: 26 November 2024.

Seroyizhko, P., Zhexenova, Z., Shafiq, M. Z., Merizzi, F., Galassi, A., and Ruggeri, F. (2022). A sentiment and emotion annotated dataset for bitcoin price forecasting based on reddit posts. In *Proceedings of the Fourth Workshop on Financial Technology and Natural Language Processing (FinNLP)*, pages

203–210, Abu Dhabi, United Arab Emirates (Hybrid). Association for Computational Linguistics. Available at: https://aclanthology.org/2022.finnlp-1.27/ (Accessed: 26 November 2024).

The Franklin Institute (2012). Vladimir vapnik. Available at: https://fi.edu/en/awards/laureates/vladimir-vapnik. (Accessed: 19 January 2025).

Upadhyay, D., Gaikwad, N., Zaman, M., and Sampalli, S. (2022). Investigating the avalanche effect of various cryptographically secure hash functions and hash-based applications. *IEEE Access*, 10:112472–112486.

Vaswani, A., Shazeer, N., Parmar, N., Uszkoreit, J., Jones, L., Gomez, A., Kaiser, , and Polosukhin, I. (2017). Attention is all you need. In *Advances in Neural Information Processing Systems*, volume 30. Curran Associates, Inc. Available at: https://papers.nips.cc/paper/7181-attention-is-all-you-need.pdf (Accessed: 18 January 2025).

List of Figures

List of Tables

Source Code and Data Accessibility

Source Code

The code for this study, including data preprocessing, model training, and evaluation, is available at: https://github.com/antek-lopacz/sentiment_analysis_bitcoin.

Data

The dataset used in this study, from which Reddit posts and the target variable were obtained, is the Bitcoin Reddit Sentiment dataset. It is available at: https://www.kaggle.com/datasets/paulsero/bitcoin-reddit-sentiment-dataset?select=reddit_sentiment_august2021.csv

Summary

Cryptocurrencies have transformed financial markets, expanding rapidly in scale and influence. Unlike traditional assets, their valuations are largely speculative, driven by innovation and potential adoption. As a result, investor sentiment significantly impacts cryptocurrency prices, making sentiment analysis a valuable tool for market forecasting.

This study examines the quantification of market sentiment through Bitcoin-related Reddit posts. Sentiment classification is performed using two embedding techniques, Word2Vec and BERT, combined with two classification models: support vector machines and multi-layer perceptrons.

The findings reveal two key insights. First, while the fine-tuned BERT-MLP model achieved the highest accuracy, simpler Word2Vec-based models performed competitively, offering computationally efficient alternatives. Second, tokenization and embedding choices strongly influenced performance, with fine-tuned BERT embeddings significantly improving accuracy. However, advanced embeddings did not always yield better results, as SVMs performed poorly with BERT embeddings without fine-tuning. This study underscores the trade-offs between computational complexity and classification accuracy, offering guidance on selecting modeling approaches for sentiment analysis in cryptocurrency markets.