Master's in Data Science and Advanced Analytics

NOVA Information Management School
Universidade Nova de Lisboa

Stock Sentiment
Predicting market behavior from tweets

Group 08

Alexandre Gonçalves, 20240738

Bráulio Damba, 20240007

Hugo Fonseca, 20240520

Ricardo Pereira, 20240745

Victoria Goon, 20240550

Spring Semester 2024-2025

Table of Contents

1. Introduction & Background

The stock market plays a fundamental role in modern economies, and investors continually seek methods to predict market trends effectively. Recently, sentiment analysis of social media content, particularly Twitter posts, has gained popularity among investors due to its ability to quickly capture market sentiment.

In recent years, advancements in Natural Language Processing (NLP) have significantly improved the accuracy of sentiment analysis, traditionally categorizing text sentiment as positive, negative, or neutral. In financial contexts, however, sentiment analysis often adopts specialized categories: "bullish" for expected price increases, "bearish" for expected declines, and "neutral" when prices are anticipated to remain stable.

In this project, we applied NLP techniques to a dataset containing several tweets, where the goal is to predict if a certain tweet is “bullish”, “bearish” or “neutral”. This project follows the following structure: The exploratory data analysis is in Section 2. Section 3 details the preprocessing steps implemented. Section 4 details the feature extraction techniques used. Section 5 contains the modelling stage and different steps that we tried to reach the final model selected. Finally, the project’s conclusions and future work will be provided in Section 7 and 8 respectively. The code used for this project can be found in the GitHub repository.

2. Exploratory Data Analysis

The dataset consists of 9,543 tweets focused on financial topics, each represented by a single row with two columns: text (the tweet content) and label (the assigned class). There is no missing data in either column. On average, tweets contain 8 to 15 words, and the overall corpus includes around 31,000 unique words. The word cloud in *Figure 2.1* helps visualize the most frequent terms throughout the corpus with each different class showing an average of 11–12 words per tweet, as detailed in Figure 2.2, which show the distribution of words per sentiment label. The distribution plots for word and character counts can be found in *Figure 2.3* and *Figure 2.4*., respectively. Tweets are classified into three categories: Bearish (0), Bullish (1), and Neutral (2). The class distribution is notably imbalanced, with 1,442 Bearish, 1,923 Bullish, and 6,178 Neutral tweets over half the corpus falls into the Neutral class (*Figure 2.5*).

Linguistic patterns across labels reveal clear distinctions. Neutral tweets tend to use formal, objective language common in financial reporting, including terms like stock, results, and dividend, along with phrases such as earnings conference and edited transcript. Bullish tweets are dominated by optimistic financial signals, with words like beats, raised, and higher, and phrases such as price target raised and eps beats. Bearish tweets reflect concern or negative sentiment, often mentioning misses, cut, and coronavirus, or phrases like target cut and hedge funds dumping.

Noise in the data includes shortened links (e.g., *https://t.co/...*), which appear roughly 5,000 times as seen in *Figure 2.6*. These links, automatically generated by Twitter (X), are not human-readable and typically redirect to financial or news websites. Hashtags and emojis are rare, reinforcing the corpus’s professional tone. Cash tags (e.g., $AAPL) are frequent, as many tweets mention specific companies. Roughly 93% of the tweets are classified as English. The remaining 7% are labeled as other languages, but manual inspection suggests most are still predominantly written in English, with only minor non-English or domain-specific terms indicating likely misclassification by the language detection model.

Content-wise, more than 85% of the tweets center on financial disclosures, such as earnings reports, dividend announcements, and market updates, frequently referencing companies like Boeing, Timken, and Procter & Gamble. The tone is informative, likely intended for investors, analysts, or financial enthusiasts.

3. Pre-processing

We applied a series of rule-based preprocessing techniques before splitting the corpus. First, we filtered tweets with less than 3 words, as seen in *Table 3.1* and *Table 3.2*, and removed them since they add no sentimental value, and then we proceeded to convert all text to lowercase to merge word variants such as "APPLE", "apple", "APple", thereby reducing dimensionality, an approach shown to be effective by *Uysal and Günal (2014).* We also expanded contractions such as “won’t” and “don’t” with “will not” and “do not”, respectively, to ensure proper tokenization, avoiding issues where tokens like “won” and “t” might incorrectly be created. To standardize non-informative elements, we replaced URLs, user mentions with placeholders (“URL”, “USER”) as proposed by *Agarwal et al. (2011)*. Numerical values were removed, since they do not contain any sentiment, and repeated punctuation was normalized for example “Really???” to “Really?”, reducing vocabulary noise and enhancing generalization (Balahur, 2013). These steps, which don’t involve learning from the data, don’t incur in data leakage.

We used NLTK’s TweetTokenizer, which is optimized for social media text. Unlike standard tokenizer, TweetTokenizer keeps hashtags (e.g #apple) and emoticons (e.g., :-)) together as single tokens, rather than splitting them into separate parts as word tokenize does (like ['#', 'apple'] or [':', '-', ')']

After tokenization, we removed punctuation and optionally filtered out stopwords that are highly frequent function words (such as articles and prepositions), which carry little sentimental value.

We then applied either lemmatization or stemming to further normalize the tokens. Lemmatization reduces words to their base or dictionary form by converting for example “running” and “ran” to “run”. This process reduces the number of unique word forms and simplifies the feature space has shown by *Guzman and Maalej (2014)* to be effective for user sentiment extraction. For Stemming, on the other hand, trims suffixes, so that words like “running,” “runner,” and “ran” may all be reduced to “run”. We used the Porter Stemmer *(Porter, 1980*).

Our pipeline applies lemmatization by default when both lemmatization and stemming are available. While both methods help reduce vocabulary size and improve efficiency, lemmatization tends to preserve the semantic meaning of words more effectively, leading to better results in sentiment analysis tasks (*Guzman and Maalej, 2014*; *Symeonidis et al., 2018*). Although stemming can be useful in certain situations, it sometimes results in a loss of meaning and does not consistently improve classification performance, particularly for deep learning models like CNNs (*Annett and Kondrak, 2008*).

To identify the most effective pre-processing strategy, we systematically varied three main options in our pipeline simultaneously: the removal of stopwords, the application of lemmatization, and the application of stemming. By considering all possible combinations of these options, we generated eight distinct pre-processing variants—for example: with or without stopword removal, with or without lemmatization, and with or without stemming, in every possible combination. All other pre-processing steps were kept constant throughout this process. We evaluated these combinations using simple statistical feature extraction methods such as TF-IDF and BoW and classical ML models, to determine which pre-processing approach yielded the best performance. The pipeline we followed is shown in *Figure 3.1*. Based on the results in *Tables 3.3* and *3.4*, the "No Lemmatization, Stemming, With Stopwords" combination consistently achieved the highest or near-highest macro F1 scores across both Bag-of-Words and TF-IDF models, and was the preferred choice for two of the three top classifiers (Logistic Regression and XGB) when using majority voting. Consequently, we selected this configuration for the remainder of the project, as performance differences among the other strategies were minimal, as shown by the F1 Macro Average score in *Tables 3.1* and *3.2* and retraining all models on every variant would have been unnecessarily resource intensive.

4. Feature Engineering

To address feature extraction, we explored five main categories of feature engineering approaches following the pipeline, illustrated in *Figure 4.1*. We began with statistical methods, using Bag of Words and TF-IDF. Next, we implemented fixed word embedding encoders such as Word2Vec, GloVe-Twitter, and FastText.

We then moved to sentence encoders such as all-mpnet-base-v2, all-distilroberta-v1, and both all-miniLM variants. Following this, we used transformer-based models (BERT-large-uncased, RoBERTa-large, and Bart-large) for feature extraction. Finally, we experimented with domain-specific transformers including FinBERT, BertTweet, and FinTwitBERT, that are pretrained specifically on financial or social media text, also to feature extraction.

4.1. Statistical Methods

We applied the bag-of-words model using the CountVectorizer (CV), which converts a collection of text documents into a matrix of token counts. In this representation, each sentence becomes a vector where each position indicates how many times a specific word appears. The vocabulary for this process can be built from the corpus or provided externally, and it determines which words become features.

However, CountVectorizer has some limitations. It ignores the order of words, which means important context is lost, something that matters for tasks like sentiment analysis. Moreover, it also treats all words equally, so frequent but unimportant words can dominate the features, while more meaningful words might be overlooked. To partially address this, we also experimented with N-Gram vectorization (e.g., bigrams), which considers word pairs or sequences to capture more context.

We then applied TF-IDF (Term frequency- inverse document frequency), which evaluates the relevance of a word in a document within a corpus of documents. It addresses the feature-vanishing issue of CV algorithms by re-weighting the count frequencies of the tokens in the sentence according to the number of appearances of each token. The algorithm works by multiplying two metrics: term frequency (TF), which calculates the number of occurrences of a term in the sequence and inverse document frequency (IDF), as seen in *Figure 4.2* , which penalizes the terms that appear in more sentences within the corpus where *t* denotes the term, *d* denotes the document, *D* denotes the corpus of documents and *N* is the total number of documents.

To avoid the curse of dimensionality and overfitting, we limited the feature space to the top 15,000 most frequent n-grams for both CountVectorizer and TF-IDF representations.

4.2. Fixed Word Embedding Encoders

To overcome the drawbacks of BoW and TF-IDF, we used word embeddings, which represent words as dense vectors in high-dimensional space based on their context, following the distributional hypothesis (*Z. S. Harris 1954*) (i.e., words used in similar contexts tend to have similar meanings).

For our project, we used Word2vec, Fast-Text and GloVe Twitter embeddings, which were trained on 2 billion tweets, containing 27 billion tokens and covering a vocabulary of 1.2 million unique words. These embeddings are available in various vector dimensions (25, 50, 100, and 200) and are uncased, meaning they do not distinguish between uppercase and lowercase letters.

Word2Vec, represents a key milestone in NLP, offering two main architectures for creating word embeddings: Continuous Bag-of-Words (CBOW), which predicts a target word from surrounding context words, and Skip-Gram, which predicts context words from a given target word (*Figure 4.3*). As a result, Skip-Gram typically performs better on larger datasets, capturing broader textual relationships.

However, Word2Vec has some limitations, notably its inability to handle unknown or out-of-vocabulary (OOV) words, since it cannot generate vectors for words it hasn't seen during training. Additionally, Word2Vec doesn't account for morphological similarities, resulting in entirely separate representations for related words like agree/agreement or worth/worthwhile (*Bojanowski et al. 2017*).

In our project, since our corpus is small, we trained our Word2Vec models using *gensim's* default CBOW architecture (sg=0), with a vector size of 200, a context window of 10, a minimum count of 1 (including all words), and up to 32 tokens per tweet (maximum sequence length). Moreover, training was parallelized across 7 CPU cores.

GloVe overcomes the drawbacks of Word2Vec in the training phase, improving the generated embeddings. It emphasizes the importance of considering the co-occurrence probabilities between the words rather than single word occurrence probabilities themselves (Pennington et al. 2014). The model combines two classes of methods for distributed word representations: global matrix factorization and Skip-grams are used to extract better features by examining the relationships between words. The global matrix factorization method can capture the statistics and relationships between words. On the other hand, Word2Vec Skip-grams method is efficient in extracting the local context and capturing the word analogy. Both methods are successfully incorporated into the GloVe encoder, thus outperforming Word2Vec in many NLP tasks.

FastText extends Word2Vec by incorporating sub-word information. Instead of treating each word as a single unit, it represents words as a collection of character n-grams. For example, with n=3, the word "finance" would be represented by n-grams such as <fi, fin, ina, nan, anc, nce, ce> and e>. We trained FastText embeddings from scratch using Gensim's implementation on our specific tweet dataset, using a vector size of 200 and a maximum sequence length of 32.

4.3. Sentence Encoders

To capture sentence-level meaning, we employed sentence encoders, which produce fixed-length embeddings that represent both syntactic and semantic properties of entire texts. While early approaches averaged word embeddings to form sentence vectors, these methods ignored word order and relationships. Modern sentence encoders, by contrast, use transformer-based architectures with attention mechanisms that model context and interdependencies across the input sequence.

Models like all-mpnet-base-v2 belong to the Sentence Transformers library (*Reimers et al. 2019*) and are specifically fine-tuned to generate fixed-size embeddings for entire sentences or short texts. Unlike basic word-level averaging, these models leverage attention mechanisms (*Y. Liu et al. 2019*) to capture the contextual relationships and dependencies between words.

One limitation of sentence encoders, however, lies in their compression strategy: regardless of input length, all information is condensed into a single fixed-size vector. While this enables fast comparison and retrieval, it can lead to loss of fine-grained details, particularly for longer texts.

In this project, we employed several state-of-the-art sentence encoders from the Sentence Transformers library to produce fixed-length vector representations of tweets. Specifically, we used all-mpnet-base-v2, which builds on MPNet and combines masked language modeling with permuted language modeling for improved contextual understanding, all-distilroberta-v1, a lighter and faster variant of RoBERTa that retains strong semantic performance, and all-MiniLM models (L6-v2 and L12-v2), which include 6 and 12 transformer layers, respectively, offering a trade-off between efficiency and representational power.

4.4. Transformers

Unlike sentence transformers that are specifically fine-tuned to generate fixed-length embeddings that capture the overall meaning of entire sentences, standard transformer models such as BERT are pre-trained for general-purpose language modeling tasks. When using transformer models for feature extraction, there are two common strategies to obtain a fixed length embedding for a sentence: using the [CLS] token (*Ledesma et. al 2024*) which is designed to capture aggregate sentence information for classification task or applying mean pooling (*Ledesma et. al 2024)* over all token embeddings, treating each word equally and averaging their representations.

In our project, we opted for mean pooling. This decision is based on findings from the literature such as Sentence-BERT (*Reimers et. al, 2019*) which demonstrate that mean pooling generally creates more robust sentence embeddings, particularly when the transformer model has not been fine-tuned for the target task.

4.5. Domain-Specific Transformers

For domain-specific sentence embeddings, we employed three transformer models specifically pre-trained on financial or social media text, namely the FinBERT model, which is a BERT-based language model that has been further pre-trained and fine-tuned on financial news and company reports (Dogu Araci *2019*). We also used BERTweet, which is a RoBERTa-based model that has been pre-trained on a large-scale English Twitter dataset (*Nguyen et. al 2020*), and we also used FinTwitBERT, which is a version of FinBert that has been further fine-tuned on financial tweets (*Yang et. al 2020*).

5. Classification Models

We based our modeling approach, based on the *Figure 5.1***.**

Firstly, in our evaluation analysis, we used four machine-learning classifiers: Support Vector Classifier (SVC), as a representative of Support Vector Machines (SVM), and an Extreme Gradient Boosting (XGB), as a representative of gradient-boosted decision trees. We used these classifiers firstly with the statistical methods (BoW and TF-IDF).

Bidirectional RNN networks are often used to collect features from both directions. A forward RNN $\vec{h}$ gathers token features from the start (x_1) to the end (x_n), while the backward RNN processes the tokens in reverse direction, from (xn) to (x1). The resulting hidden state h uses both sets of features concatenating, the forward RNN and the backward RNN.

Furthermore, we build an additional group of GRU and LSTM networks, which support bidirectional feature extraction, to assess their performance as described in [*K. Mishev et al 2019*] and we use the cross-entropy loss function when training the models. The ADAM (Adaptive Learning Rate) optimization algorithm is used to find optimal weights in the networks, for the baseline models. We use a maximum of one hundred training epochs for all DL models. Moreover, we impose early stopping when the validation loss does not diminish after ten epochs to prevent overfitting.

In order to improve the architecture of previous DNN net works, several mechanisms have been introduced: one of them is the Attention mechanism (*D. Bahdanau et. al 2014*) , which helps RNN networks focus on specific parts of the input sequence, improving the prediction. The Attention mechanism is widely used in encoder-decoder architectures due to its ability to highlight important parts of the contextual information. We've implemented three types of attention layers, as shown in Table namely the simple attention layers for our baseline models, and then when we did hyperparamter tuning we introduced Bahdanau (Additive) and a Multi-Head Self-Attention, that can be further analyzed and consulted in *Table 5.1*.

It is important to mention, that we did not use BiLSTM or BiGRU architectures in combination with sentence encoders or transformer-based embeddings. This is because sentence encoders and transformers are designed to output fixed-length vector representations that already capture the relevant context of the entire sentence, making additional sequential modeling with RNNs unnecessary and potentially redundant. Instead, these representations are best leveraged by traditional machine learning classifiers such as SVM or XGBoost. Conversely, using LSTM or BiGRU networks with fixed word embedding encoders (such as Word2Vec, FastText, or GloVe) is appropriate, as these embeddings provide token-level representations without capturing the sequential dependencies present in the input text. This distinction is why we paired traditional ML classifiers with statistical features like BoW and TF-IDF, as these approaches do not exploit sequential dependencies, while we used LSTM and BiGRU architectures specifically with word-level embeddings to model sequence information for sentiment analysis.

6. Fine Tunning NLP Transformers

We fine-tuned pre-trained transformer models (BERT-base, BART-base, and FinTwitBERT) for sequence classification using the Hugging Face Transformers library. Each model was loaded with its corresponding tokenizer, and texts were tokenized with padding/truncation. Fine-tuning was conducted using the Hugging Face Trainer, with early stopping based on validation loss, a batch size of 8, and up to 100 epochs. The model with the lowest validation loss was selected for evaluation.

6.1. BERT

BERT, introduced by *Devlin et al. (2018)*, is a language model based on the transformer architecture. Unlike previous models that processed language in only one direction, BERT learns deep, bidirectional representations, meaning it considers the context of a word from both its left and right. It is pre-trained on large unlabeled text corpora such as Wikipedia and BookCorpus using two tasks: Masked Language Modeling (MLM), where random words in a sentence are masked and the model learns to predict them, and Next Sentence Prediction (NSP), which helps the model understand the relationship between sentences.

BERT comes in two common versions: BERT-base (12 layers, 768 hidden size, 110M parameters) and BERT-large (24 layers, 1024 hidden size, 340M parameters). For this project, we've used the bert-base architecture.

6.2. BART

BART, developed by Facebook AI in 2019, is a transformer model that combines the strengths of BERT and GPT architectures. Its encoder is similar to BERT, capturing bidirectional context, while its decoder is auto regressive like GPT. BART's architecture consists of an encoder-decoder structure connected by cross-attention, which enables the decoder to focus on relevant parts of the input when generating output. For classification tasks, such as sentiment analysis, BART is fine-tuned by feeding the input sentence to the encoder and using the decoder's end-of-sequence (EOS) token representation to predict the class. We applied this structure to fine-tune BART for financial sentiment classification.

6.3. FinTwitBERT

FinTwitBERT is a transformer model specialized for financial tweets. It builds on the FinBERT model but is pre-trained on a corpus of 10 million financial tweets, allowing it to better capture the nuances and informal language common in financial social media. It includes special tokenization to handle Twitter-specific elements like user mentions (@USER) and URLs ([URL]) (*Akkerman, S. 2022*).

7. RESULTS AND DISCUSSION

This section presents the results and key findings of our models, organized by the feature extraction strategy, classification method, and fine-tuning approach.

7.1. EVALUATION METRICS USED

As discussed in the EDA, our corpus exhibits a significant class imbalance. In such cases, relying solely on accuracy can be misleading, as a model could achieve high accuracy by predicting the majority class for most samples while failing to correctly classify the minority classes. To provide a fair evaluation on this imbalanced dataset, we focus on the macro-averaged F1 score as our main metric. Macro F1 treats each class equally, ensuring that minority classes such as Bearish and Bullish are properly accounted for, rather than being overshadowed by the majority Neutral class.

In addition, we report macro-averaged precision, recall, and overall accuracy for a comprehensive overview. However, all our model comparisons and selections are based primarily on macro F1, as it offers a balanced assessment regardless of class distribution.

7.2. Statistical Methods

In *Table 7.1* we report on the performance of the traditional machine learning models using the Bag of Words feature extraction technique. Logistic Regression, SVC, and XGB all achieved similar macro F1 scores (around 0.75), showing that linear and ensemble methods can perform well with text features. However, despite XGB's slightly higher accuracy and precision, its lower recall suggests that it favors the majority class and may miss more minority class examples. On the other hand, KNN performed much worse, as expected for imbalanced datasets, since its predictions are dominated by the majority class among the nearest neighbors.

In *Table 7.2*, we present the performance of the traditional machine learning models using TF-IDF as the feature extraction technique. Logistic regression with TF-IDF achieved a slightly higher f1-macro than when compared to BoW (0.7704 vs. 0.7530). This can be explained by the fact that tf-idf down weights the background noise, when compared to BoW. The score for all the other models remains as for most classifiers, BoW and TF-IDF offer similar performance because both representations encode enough information for these algorithms to separate classes.

7.3. Fixed word embedding encoders

In *Table 7.3*, *Table 7.4* and *Table 7.5*, we present the performance for the several RNNs we implemented using the different encoding methods : Word2Vec, Fast-Text, and Glove-Twitter, respectively.

Word2vec, as stated above was trained in our corpus, and as our corpus is small, probably it is the reason why this method performs poorly across all models. Nonetheless, the performance is a bit superior for BiGRU, achieving a 0.4578 f1 macro.

Fast-text performs even poorly when compared to word2vec, across all models, which wasn't expected, as usually as report in the literature (*Bojanowski et al. 2017)* fast-text outperforms or matches word2vec. This can probably be explained by the fact that our best-preprocessing variant is too normalized, which removes the morphological variety that Fast-Text leverages. Moreover, our training corpus is small and so aligned with the previous fact , fast-text may not have enough data to learn useful n-gram patterns. The best model using fast-text was BiGRU with the simple attention layer, howerver the f1 macro differs only 0.0129 from the model without the attention layer.

GloVe-Twitter embeddings consistently outperformed both Word2Vec and FastText across all RNN architectures. This strong performance can be attributed to several factors. First, GloVe-Twitter was pre-trained on a massive corpus of Twitter data, capturing the unique vocabulary, slang, and informal writing styles used on social media platforms. This domain relevance is crucial, as it ensures that the embeddings already capture patterns specific to tweets, including the kinds of abbreviations and context switches typical in financial discussions online. In contrast, the Word2Vec and FastText embeddings were trained solely on our own dataset, which, while domain-specific, is much smaller and therefore unable to capture as rich a semantic and syntactic structure. As a result, the GloVe-Twitter embeddings provide a more appropriate representation of the input text. The best model using Glove-Twitter was BiGRU + Attention having 0.7785 f1 macro average, while BiLSTM with or without attention achieve more or less the same performance.

A key finding from both the Word2Vec and GloVe-Twitter results is that BiGRU models show a much greater improvement from the addition of an attention mechanism compared to BiLSTM models. This likely stems from the GRU's simpler memory design, which means it depends more on external cues, like attention, to focus on the most relevant parts of the input sequence. In contrast, LSTMs already have a more sophisticated internal memory, allowing them to track important information over longer sequences even without attention. Our observation is consistent with previous research, which demonstrates that attention layers can significantly improve the performance of RNN-based models (Li et. all 2018) by helping them better capture relevant information from the input. However, it's important to note that there is no definitive evidence in the literature that GRUs always benefit more than LSTMs from attention; our finding is primarily empirical and based on our corpus.

7.4. Sentence Encoders

In Table we 7.6, 7.7, 7.8 and 7.9 we present the performance of classifier models for the embeddings creating by pre trained models, respectively mp_net_base_v2, distilroberta_v1, all-mini-l12-v2 and all-minilm-l6-v2 (Sentence Transformers, n.d).

Across all tested sentence encoder models , we observe several consistent patterns in performance. SVC and MLP achieved the best macro F1 scores, consistently outperforming Logistic Regression, XGBoost, and KNN. For instance, all models using sentence encoders reached macro F1 scores above 0.73, even in a class-imbalanced context. KNN and Logistic Regression fell behind, likely because sentence embeddings work best with models capable of capturing non-linear semantic patterns. The best overall result in this category was SVC with distilroberta_v1 (macro F1 = 0.7439), slightly edging out both MiniLM L12 and MiniLM L6 models by a narrow margin (0.0021).

7.5. Transformers

7.5.1. Feature Extraction

Table 7.10, 7.11, 7.12 shows the performance of models using transformer encoders namely BERT-Large-Uncased, RoBERTAa-Large-Uncased and BART-Large respectively.

Perfomance improved with higher quality encoders. When using BERT-Large-uncased and RoBERTa-Large-uncased as sentence encoders, both SVC and MLP consistently deliver the highest macro F1 scores and overall accuracy. For instance, with RoBERTa-Large, both SVC (F1 Macro 0.7622) and MLP (F1 Macro 0.7596) demonstrate robust performance. Logistic Regression and XGBoost achieved reasonable performance even if it weren't as good, while KNN again showed the weakest results, indicating its limitations in high-dimensional semantic spaces.

7.5.2. Fine Tunning

Table 7.13 and *7.14*, report the performance of fine tuning on transformers, namely BERT and BART respectively, to obtain the classification. Both BERT and BART demonstrate strong performance, with BART achieving the highest macro F1 score (0.7824) and accuracy (0.8417) after 5 epochs, compared to BERT's macro F1 of 0.770 and accuracy of 0.8375 in 4 epochs. These results confirm that fine-tuning directly on the classification task significantly enhances performance over feature extraction alone.

7.6. Domain specific transformers

7.6.1. Feature Extraction

Table 7.15, Table 7.16 and *Table 7.17* report the performance of FinBert, BERT-Tweet and FinTWITBert. We observe that when using domain-specific transformer models for feature extraction, such as these models, we observe that performance is generally strong and highly competitive with generic, large-scale transformer encoders. Across all three models, MLP and SVC classifiers consistently achieve the best macro F1 scores, typically ranging from 0.71 to 0.76, with FinBERT + MLP delivering the highest macro F1 (0.7596) and accuracy (0.8239) among all tested combinations. Among the three models, FinBERT achieves the strongest results, followed by BERT-Tweet and FinTwitBERT, indicating that pre-training on financial or social media-specific corpora provide a benefit for sentiment analysis.

7.6.2. Fine Tunning

Table 7.18 report the performance on FinTwitBERT.

Given that FinTwitBERT is pre-trained on financial tweets, we thought that it would outperform the general-purpose transformer models once fine-tuned. However, its F1 score, and accuracy were similar to BERT and BART. The main advantage of FinTwitBERT emerged in training stability, as evidenced by its much lower validation loss. In our corpus, the domain-specific pre-training offer benefits, particularly reducing overfitting.

8. Hyperparameter tunning and handling imbalance

After evaluating our baseline models, we decided to perform hyperparameter tuning for the best model that was not a transformer, and for the best fine-tuned transformer. We decided to do this, grounded on the fact that we wanted to test whether these big architectural differences had such an impact or not in our corpus.

Up until now, in the validation set our best model that wasn't a transformer was BiGRU with a simple attention layer and with GloVe Twitter Embeddings, achieving a F1 Macro average of 0.7785. Firstly, we tested the impact of the different embedding dimensions, as it can be seen in *Table 8.1*, and we got 200D as the best dimensions for GloVe Twitter, being largely superior to the other embedding dimensions, and we noticed that the more we reduce the size of the embeddings the worse the performance becomes. The next step was testing different attention layers, namely the Badahanau (Additive) layer *(D. Bahdanau, et. al 2014),* and the multi-head self-attention layer (*Ashish Vaswani,et. al 2017*) that are explained in detail in the *Table 8.2*. From this step, we conclude that using other layers different from the simple, deteriorate the performance of our model, so we stick to the simple attention layer. Next, we tested different parameter combinations, namely the ones present in *Table 8.3*, using 50 bayesian search trials, as it builds improvements on top of the previously best found hyperparameters and it is more efficient than random search.

The best combination of hyperparameters found is present in *Table 8.4* and the results with these combinations, can be found in *Table 8.5*, were slightly worse (0.7713) than the default setting (0.7785), which suggests that we didn't explore well the parameter space, and as our dataset is small the randomness in our validation split can make some hyperparameter settings look worse by chance.

Next, we performed hyperparameter tuning for the best previously found tunned transformer, which was BART, and we performed this search using 20 trials of bayesian search (using the Tree-structured Parzen Estimator), and we tested the parameters that are present in *Table 8*.6, and we got the best hyperparameters found in *Table 8.7.* The tunned model performed really well, being our best model up until now, as visible in *Table 8.8*, and obtained an improvement of almost 0.06 f1 macro average points when compared to the BART non tunned model. Next, as we already identified the class imbalance, and it was not possible to apply smote directly to raw text, and we were using raw text as an input to BART, we decided not to use it , and instead we used random undersampling which first identifies the minority class, and then it randomly selects, without replacement to avoid duplicate samples, an equal number of samples from each class , matching the size of the minority class. The performance of this method, is present in *Table 8.9* was a lot worse than the tuned model (0.7868 vs. 0.8417).

9. Performance on test set and error analysis

We evaluated the performance of our best model (tuned BART) in the validation and test set, and we conclude that the model generalizes well for the test set, as it shows consistent results across all metrics, in both validation and test, and in test only has less 0.026 of f1 macro average than validation (*Figure 9.1*).

Analyzing the confusion matrix on test set, present in *Figure 9.2.* , we get that the model most accurately predicts the Neutral class, correctly classifying 558 Neutral instances. However, 38 Neutral samples were incorrectly labeled as Bearish and 22 as Bullish. For the Bearish class, 108 out of 144 instances were correctly identified, but 28 were misclassified as Neutral and 8 as Bullish. Similarly, 153 Bullish samples were correctly predicted, while 36 were mistaken for Neutral and 4 for Bearish.

To further analyze the reason of this difference, as it is visible in *Table 9.1*, we can see that for instance, some tweets labeled as Bearish were predicted as Neutral, such as *"OPEC+ deal is much weaker than it looks"* or *"Fed policymakers working to limit damage as pandemic puts U.S. economy on pause".* These tweets mention negative situations, but they do so in a somewhat factual or understated tone, which may not provide strong enough signals for the model to confidently predict a Bearish sentiment.

For true bullish tweets, as seen in *Table 9.2* for example in *"$TOPS moving good today. Just play the movement. But don't get stuck holding the bag."* starts with a positive statement about a stock's movement but then includes a cautionary note. The mixed tone might confuse the model, leading it to classify the tweet as Neutral due to the lack of clear bullish sentiment. Additionally for the tweet, *"One analyst says TD could unlock 'significant value' by selling its minority stake in Ameritrade after its purchase…,"* the Bullish perspective is implied by the phrase *"unlock significant value"*. However, the model might focus on the factual reporting of an analyst's opinion rather than the positive implication, resulting in a Neutral prediction.

10. Conclusion and future work

This project demonstrates that fine-tuned transformer models, particularly BART, are the most effective for financial sentiment classification on Twitter, outperforming both traditional machine learning and RNN-based approaches. Nonetheless, we give an explicit mention to the fact that RNNs such as BiGRU performed better than we were expecting, and in the case of not having enough computational resources, this could be a good approach to use when doing this kind of sentiment analysis. Moreover, the logistic regression with such a simple method like TF-IDF that runs in seconds or few minutes in good hardware, can also be an alternative. We also mention that text pre-processing can have to some extent impact, so it should be done and properly tested when one wants to obtain the best results.

There were important limitations. Preprocessing choices were based on what worked for traditional ML models, which may not be optimal for deep learning or transformers. Computational resources limited our ability to fully explore preprocessing and hyperparameter settings. We only tested 20 trials for hyperparameter tuning with transformers, compared to 50 trials for BiGRU + Simple Attention, making the search effort and computational cost unequal. Although transformers outperformed BiGRU + Attention even with fewer trials, it is likely that additional tuning could have improved their performance even further. Notably, hyperparameter tuning for BiGRU + Attention actually led to worse results than the default model, suggesting that our search space was not well explored, more trials would be necessary to mitigate this. Finally, random undersampling was used to balance the classes, but this approach discarded a large amount of data and resulted in lower overall performance. More advanced strategies for handling class imbalance, as well as more extensive and model-specific tuning could further improve results in future work. Model ensembling, combining predictions from both transformer-based and traditional models, could also be considered to further boost robustness and performance.

References

1) Uysal, A.K. and Gunal, S. (2014) The Impact of Preprocessing on Text Classification. Information Processing & Management, 50, 104-112.
2) Agarwal et al., (2011) , Sentiment Analysis of Twitter Data
3) Alexandra Balahur (2013), Sentiment Analysis in Social Media Texts.
4) Guzman and Maalej (2014) , How Do Users Like This Feature? A Fine Grained Sentiment Analysis of App Reviews
5) Porter, M. (1980) The Porter Stemming Algorithm.
6) Symeonidis et al., (2018) , A comparative evaluation of pre-processing techniques and their interactions for twitter sentiment analysis
7) Annett, M., Kondrak, G. (2008) , A Comparison of Sentiment Analysis Techniques: Polarizing Movie Blogs
8) Z. S. Harris, Distributional structure, Word, vol. 10, nos. 23, pp. 146162, Aug. 1954.
9) P. Bojanowski, E. Grave, A. Joulin, and T. Mikolov, Enriching word vectors with subword information, Trans. Assoc. Comput. Linguistics, vol. 5, pp. 135146, Dec. 2017.
10) J. Pennington, R. Socher, and C. Manning, Glove: Global vectors for word representation, in Proc. Conf. Empirical Methods Natural Lang. Process. (EMNLP), 2014, pp. 15321543.
11) Reimers, N., & Gurevych, I. (2019). Sentence-BERT: Sentence Embeddings using Siamese BERT-Networks.
12) Ledesma Roque, D. A., Kolesnikova, O., & Menchaca Méndez, R. (2024). Exploring the interpretability of the BERT model for semantic similarity.
13) Dogu Araci (*2019)* FinBERT: Financial Sentiment Analysis with Pre-trained Language Models
14) Dat Quoc Nguyen, Thanh Vu, Anh Tuan Nguyen (2020) , BERTweet: A pre-trained language model for English Tweets
15) Yang, Y., Uy, M. C. S., & Huang, A. (2020). FinBERT: A Pretrained Language Model for Financial Communications
16) K. Mishev et al., Performance evaluation of word and sentence embeddings for finance headlines sentiment analysis, in ICT Innovations 2019.
17) D. Bahdanau, K. Cho, and Y. Bengio (2014) , Neural machine translation by jointly learning to align and translate
18) Devlin, J., Chang, M.-W., Lee, K., & Toutanova, K. (2019). BERT: Pre-training of Deep Bidirectional Transformers for Language Understanding
19) Akkerman, S. (2022). FinTwitBERT: A domain-specific language model for financial Twitter data. Available at: https://huggingface.co/StephanAkkerman/FinTwitBERT
20) Ashish Vaswani, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N. Gomez, Łukasz Kaiser & Illia Polosukhin (2017), Attention Is All You Need. In Advances in Neural Information Processing Systems 30 (NeurIPS 2017). Introduces the Transformer architecture, including multi-head self-attention
21) Enriching Word Vectors with Subword Information
22) Ashish Vaswani et al.
23) Bojanowski, P., Grave, E., Joulin, A., & Mikolov, T. (2016). Enriching Word Vectors with Subword Information. Transactions of the Association for Computational Linguistics, 5, 135–146.
24) Li, B., Liu, T., Zhao, Z., & Zhang, X. (2018). Attention-Based Recurrent Neural Network for Sequence Labeling. Proceedings of the 27th International Joint Conference on Artificial Intelligence (IJCAI 2018), 4730–4736.
25) Sentence Transformers. (n.d.). Pretrained Models. Retrieved from https://www.sbert.net/docs/sentence_transformer/pretrained_models.html

APPENDIX

APPENDIX 1 – EXTRA WORK

Feature Engineering

We implemented and fine-tuned FinTwitBERT (encoder).

Classification Model

We implemented and fine-tuned BART (encoder-decoder).

Appendix 2 – EDA

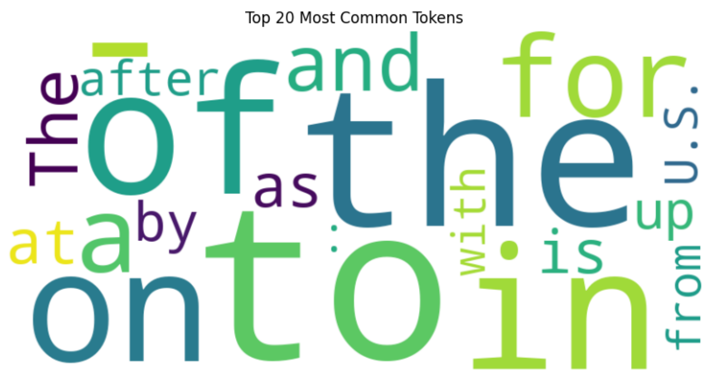

Figure 2.1 - Word cloud showing the most common tokens in the corpus

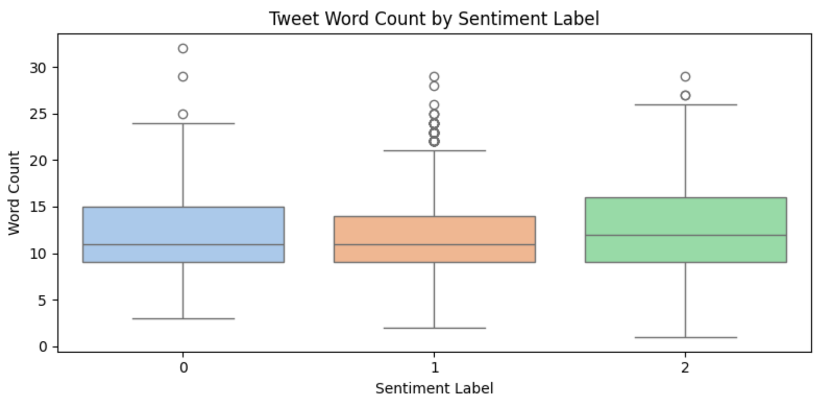

Figure 2.2 - Distribution of words per tweet per label.

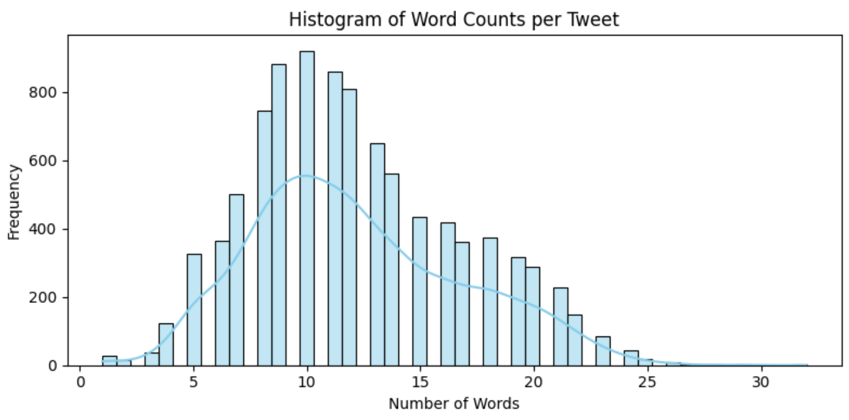

Figure 2.3 - Histogram showing the count and distribution of words per tweet

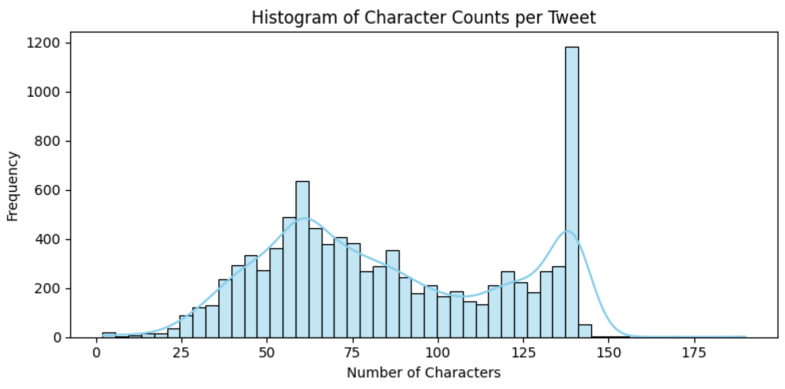

Figure 2.4 - Histogram showing the count and distribution of characters per tweet

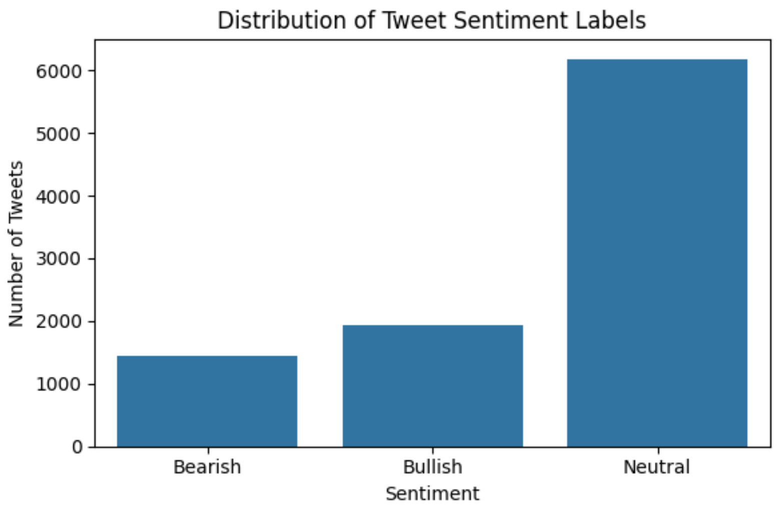

Figure 2.5 – Distribution of tweet sentiment labels

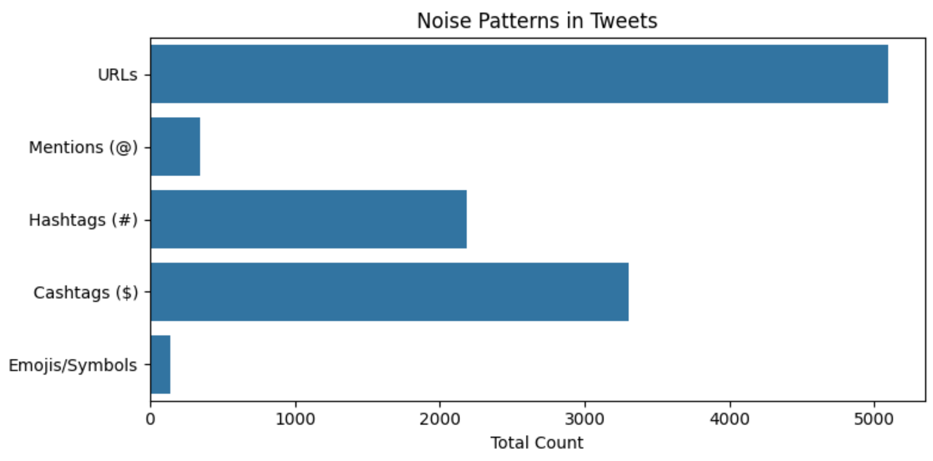

Figure 2.6 - Counts of different noise patterns found in the corpus

Appendix 3 – Pre Processing

Index	*Tweet*	*Count of Words*
6841	Nasdaq climbs 0.3%	3
6849	Nasdaq up 0.2%	3
7800	$AIKI boucing nicely…..	3
6819	Futures up https://t.co/diz7v5lmVb	3

Table 3.1 - Samples of tweets that have 3 words or less

Index	*Tweet*	*Count of Words*
4360	Damn	1
4440	F5	1
4914	Next week.	2
4681	https://t.co/575AH1YRkF	1

Table 3.2 - Samples of tweets that have 2 words or less

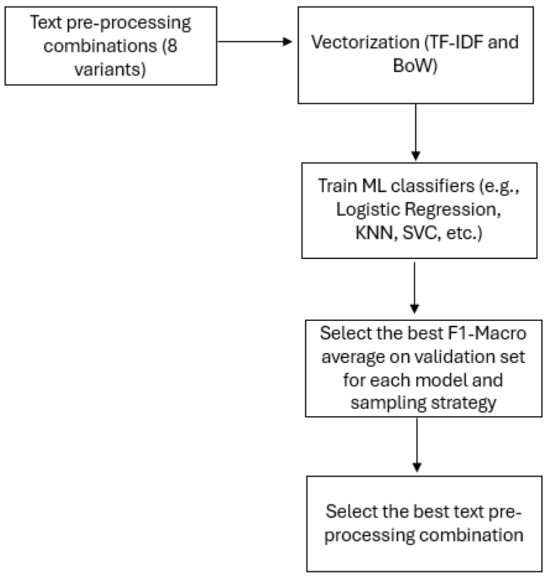
Text pre-processing combinations (8 variants)
Vectorization (TF-IDF and BoW)
Train ML classifiers (e.g., Logistic Regression, KNN, SVC, etc.)
Select the best F1-Macro average on validation set for each model and sampling strategy
Select the best text pre-processing combination

Figure 3.1 – Pipeline followed to choose the best text variant

Variant	*Model*	*F1-Macro Average*
No Lemmatization, Stemming, With Stopwords	Logistic Regression	**0.7298**
No Lemmatization, No Stemming, With Stopwords	Logistic Regression	0.7283
Lemmatization, No Stemming, With Stopwords	Logistic Regression	0.7240
Lemmatization, No Stemming, With Stopwords	SVC	0.7088

Lemmatization, Stemming, With Stopwords	SVC	0.7088
No Lemmatization, Stemming, With Stopwords	SVC	0.7061
No Lemmatization, Stemming, With Stopwords	XGB	0.7064
Lemmatization, Stemming, With Stopwords	XGB	0.6847
Lemmatization, No Stemming, With Stopwords	XGB	0.6839

Table 3.3 – F1 Macro Average of Text Variants with Bag of Words for Traditional ML Models

Variant	*Model*	*F1-Macro Average*
No Lemmatization, Stemming, With Stopwords	SVC	**0.7319**
No Lemmatization, No Stemming, With Stopwords	SVC	0.7186
Lemmatization, No Stemming, With Stopwords	SVC	0.7168
No Lemmatization, Stemming, With Stopwords	Logistic Regression	0.7107
Lemmatization, No Stemming, No Stopwords	Logistic Regression	0.7034
Lemmatization, Stemming, No Stopwords	Logistic Regression	0.7034
No Lemmatization, Stemming, With Stopwords	XGB	0.6871
Lemmatization, No Stemming, With Stopwords	XGB	0.6784
Lemmatization, Stemming, With Stopwords	XGB	0.6784

Table 3.4 – F1 Macro Average of Text Variants with TF-IDF for Traditional ML Models

Appendix 4 – Feature Engineering

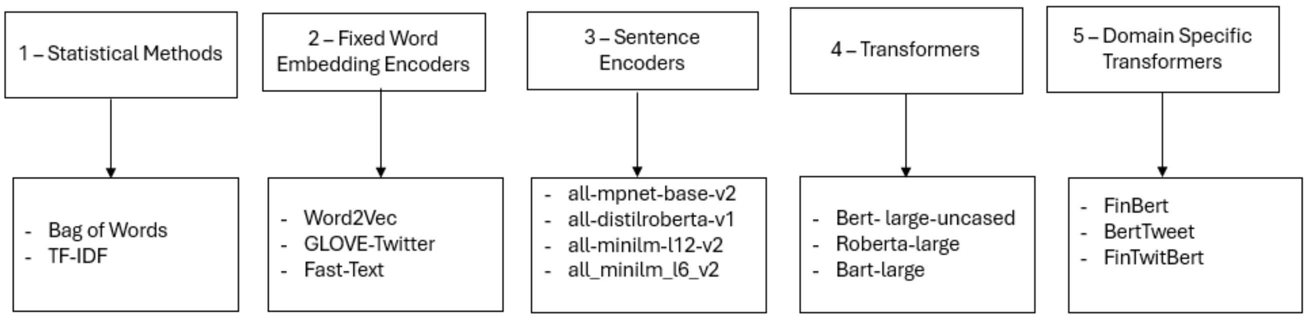

Figure 4.1 – Feature Engineering pipeline

$$
\begin{aligned}
tfidf(t,d,D) &= tf(t,d)\cdot idf(t,D)\\
tf(t,d) &= log(1+freq(t,d))\\
idf(t,D) &= log(\frac{N}{count(d\in D: t\in d)})
\end{aligned}
$$

Figure 4.2 – TF-IDF approach

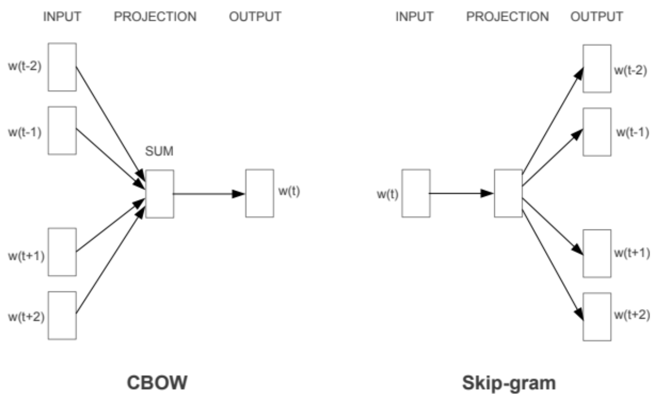

Figure 4.3 – Architectures of Word2Vec

Appendix 5 – Classification Models

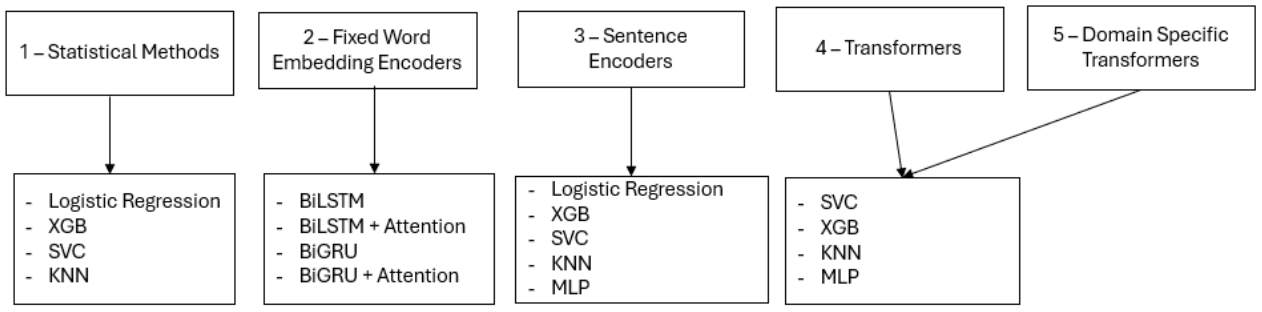

Figure 5.1 – Classification models used

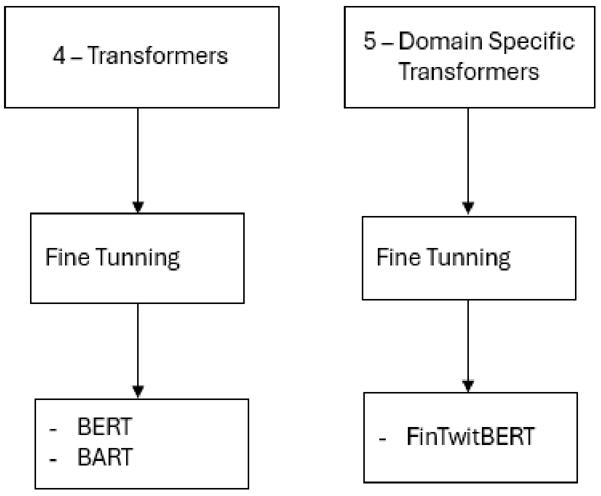

Figure 5.2 – Fine Tunning for transformers

Attention Layer	*Description*
Simple	Calculates a set of attention weights over the hidden states and produces a context vector as a weighted sum.
Bahdanau (Additive)	Introduces an alignment model that scores each encoder hidden state based on the decoder's current state, using a small neural network.
Multi-Head Self-Attention	Projects the input into multiple lower-dimensional subspaces (heads) and computes attention independently in each. The outputs from all heads are then concatenated and linearly transformed

Table 5.1 – Attention layers used

Appendix 7 - RESULTS

1 – Statistical Methods

Model	*F1 Macro*	*F1 Precision*	*F1 Recall*	*Accuracy*
SVC	**0.7514**	0.7687	0.7371	0.8207
XGB	0.7499	0.8301	0.7075	0.8301
Logistic Regression	0.7530	0.7615	0.7454	0.8218
KNN	0.4464	0.7711	0.4354	0.6991

Table 7.1 – Results for BoW with traditional ML models

Model	*F1 Macro*	*F1 Precision*	*F1 Recall*	*Accuracy*
Logistic Regression	**0.7704**	0.7585	0.7844	0.8249
SVC	0.7552	0.7985	0.7280	0.8270
XGB	0.7262	0.8255	0.6780	0.8144
KNN	0.4225	0.7909	0.4197	0.6907

Table 7.2 – Results for TF-IDF with traditional ML models

2 – Fixed word embedding encoders

Model	*F1 Macro*	*F1 Precision*	*F1 Recall*	*Accuracy*
BiGRU	**0.4578**	0.5296	0.4564	0.6908
BiGRU + Simple Attention	0.4323	0.5330	0.4530	0.7002
BiLSTM	0.4395	0.5766	0.4518	0.7002
BiLSTM + Simple Attention	0.3824	0.4163	0.4062	0.6824

Table 7.3 – Results for Word2Vec using DL models

Model	*F1 Macro*	*F1 Precision*	*F1 Recall*	*Accuracy*
BiGRU + Simple Attention	**0.4082**	0.6358	0.4200	0.6792
BiLSTM + Simple Attention	0.3953	0.4061	0.4177	0.6792
BiLSTM	0.3910	0.3911	0.4147	0.6688
BiGRU	0.3424	0.3906	0.3751	0.6614

Table 7.4 – Results for Fast-Text using DL models

Model	*F1 Macro*	*F1 Precision*	*F1 Recall*	*Accuracy*
BiGRU + Simple Attention	**0.7785**	0.7882	0.7696	0.8344
BiLSTM + Simple Attention	0.7624	0.7893	0.7425	0.8281
BiLSTM	0.7618	0.7855	0.7442	0.8312
BiGRU	0.6939	0.7605	0.6571	0.7925

Table 7.5 – Results for GloVe Twitter using DL models

3 – Sentence Encoders

Model	*F1 Macro*	*F1 Precision*	*F1 Recall*	*Accuracy*
SVC	**0.7418**	0.7260	0.7622	0.7935
MLP	0.7140	0.7435	0.6922	0.7914
Logistic Regression	0.6737	0.6521	0.7270	0.7170
XGB	0.6307	0.7233	0.5935	0.7600
KNN	0.6625	0.6710	0.6571	0.7589

Table 7.6 – Results for mpnet_base_v2 using tradional ML models

Model	*F1 Macro*	*F1 Precision*	*F1 Recall*	*Accuracy*
SVC	**0.7439**	0.7250	0.7700	0.7935
MLP	0.7141	0.7585	0.6858	0.8019
Logistic Regression	0.6977	0.6740	0.7447	0.7432
XGB	0.6920	0.7941	0.6464	0.7600
KNN	0.6676	0.6699	0.6654	0.7589

Table 7.7 – Results for distilroberta_v1 using traditional ML models

Model	*F1 Macro*	*F1 Precision*	*F1 Recall*	*Accuracy*
SVC	**0.7418**	0.7260	0.7622	0.7935
MLP	0.7389	0.7776	0.7120	0.8124
Logistic Regression	0.6737	0.6521	0.7270	0.7170
KNN	0.6625	0.6699	0.6654	0.7589
XGB	0.6307	0.7941	0.6464	0.7600

Table 7.8 – Results for all-minilm_l12_v2

Model	*F1 Macro*	*F1 Precision*	*F1 Recall*	*Accuracy*
SVC	**0.7389**	0.7776	0.7120	0.8124
MLP	0.7079	0.7315	0.6904	0.7945
Logistic Regression	0.6803	0.6583	0.7239	0.7306
XGB	0.6383	0.7619	0.5984	0.7715
KNN	0.6320	0.6425	0.6269	0.7358

Table 7.9 – Results for all-minilm_l6_v2

4 – Transformers

1) Feature Extraction

Model	*F1 Macro*	*F1 Precision*	*F1 Recall*	*Accuracy*
MLP	**0.6512**	0.6684	0.6396	0.7526
SVC	0.6370	0.6210	0.6810	0.6992
Logistic Regression	0.6350	0.6199	0.6694	0.7044
XGB	0.5961	0.7202	0.5561	0.7453
KNN	0.5669	0.5740	0.5612	0.6939

Table 7.10 - Results for BERT-Large-Uncased using feature extraction

Model	*F1 Macro*	*F1 Precision*	*F1 Recall*	*Accuracy*
MLP	**0.7442**	0.7693	0.7253	0.8166
Logistic Regression	0.7253	0.7021	0.7651	0.7736
XGB	0.6778	0.7811	0.6342	0.7893
SVC	0.6368	0.6499	0.6263	0.7442
KNN	0.6330	0.6416	0.6269	0.7463

Table 7.11 – Results for RoBERTa-Large-Uncased using feature extraction

Model	*F1 Macro*	*F1 Precision*	*F1 Recall*	*Accuracy*
SVC	**0.7622**	0.7447	0.7854	0.8166
MLP	0.7596	0.7750	0.7464	0.8239
Logistic Regression	0.6945	0.6745	0.7276	0.7516
XGB	0.6702	0.7575	0.6314	0.7862
KNN	0.6222	0.6221	0.6240	0.7348

Table 7.12 – Results for BART-Large using feature extraction

2) Fine Tunning

Model	*F1 Macro*	*F1 Precision*	*F1 Recall*	*Accuracy*	*Epochs*	*Validation Loss*
BERT	**0.7770**	0.7944	0.7637	0.8375	4	0.7603

Table 7.13 – Results for BERT with fine-tunning

Model	*F1 Macro*	*F1 Precision*	*F1 Recall*	*Accuracy*	*Epochs*	*Validation Loss*
BART	**0.7824**	0.8126	0.7592	0.8417	5	0.7407

Table 7.14 – Results for BART with fine-tunning

5 – Domain Specific Transformers

1) Feature Extraction

Model	*F1 Macro*	*F1 Precision*	*F1 Recall*	*Accuracy*
MLP	**0.7596**	0.7750	0.7464	0.8239
SVC	0.7160	0.6961	0.7474	0.7704
XGB	0.7067	0.7655	0.6722	0.7987
KNN	0.6958	0.7041	0.6902	0.7746
Logistic Regression	0.6865	0.6658	0.7263	0.7442

Table 7.15 – Results for FinBert using feature extraction

Model	*F1 Macro*	*F1 Precision*	*F1 Recall*	*Accuracy*
MLP	**0.7171**	0.7261	0.7094	0.7935
Logistic Regression	0.7001	0.6784	0.7423	0.7537
SVC	0.6961	0.6761	0.7370	0.7516
XGB	0.6327	0.7306	0.5937	0.7662
KNN	0.6248	0.6251	0.6248	0.7285

Table 7.16 – Results for BERT-Tweet using feature extraction

Model	*F1 Macro*	*F1 Precision*	*F1 Recall*	*Accuracy*
SVC	**0.7173**	0.7009	0.7441	0.7746
MLP	0.7166	0.7214	0.7143	0.7893
XGB	0.6983	0.7692	0.6590	0.7883
KNN	0.6929	0.7101	0.6807	0.7715
Logistic Regression	0.6604	0.6401	0.7022	0.7138

Table 7.17 – Results for FinTWITBERT using feature extraction

2) Fine Tunning

Model	*F1 Macro*	*F1 Precision*	*F1 Recall*	*Accuracy*	*Epochs*	*Validation Loss*
FinTwitBert	**0.7708**	0.8226	0.7367	0.8386	4	0.6443

Table 7.18 – Results for FinTWITBERT using fine-tunning

Appendix 8 – Hyperparameter Tuning

1) BiGRU + Attention Layer (with GloVe embeddings)

Model	*GloVe Embeddings dimension*	*Attention Layer*	*F1 Macro*	*F1 Precision*	*F1 Recall*	*Accuracy*
BiGRU	25 D	Simple	0.6430	0.7017	0.6118	0.7589
	50 D		0.7223	0.7537	0.6993	0.8040
	100 D		0.7415	0.7796	0.7146	0.8176
	200 D		**0.7785**	0.7882	0.7696	0.8344

Table 8.1 – Glove embeddings dimension

Model	*Attention Layer*	*GloVe Embeddings Dimension*	*F1 Macro*	*F1 Precision*	*F1 Recall*	*Accuracy*
BiGRU	**Simple**	200 D	**0.7785**	0.7882	0.7696	0.8344
	Bahdanau (Additive)	200 D	0.7638	0.7962	0.7395	0.8291
	Multi-Head Self-Attention	200 D	0.7525	0.7870	0.7273	0.8249

Table 8.2 – Attention layers

Hyperparameter	*Values tested*
embedding_dropout	0 , 0.2
num_layers	1 , 2
units	64, 128
dropout	0.2 , 0.4
recurrent_dropout	0.2 , 0.4
l2	0 , 1e-4
extra_dense	True, False
optimizer	adam, rmsprop, nadam
learning_rate	1e-3, 5e-4

Table 8.3 – Search space for hyperparameters for Bigru

Hyperparameter	***Value***
embedding_dropout	0.0
num_layers	1
units	128
dropout	0.4
recurrent_dropout	0.2
l2	0.0
extra_dense	True
optimizer	nadam
learning_rate	0.0005

Table 8.4 – Chosen hyperparameters for BiGRU

Model	*GloVe Embeddings dimension*	*Attention Layer*	*F1 Macro*	*F1 Precision*	*F1 Recall*	*Accuracy*
BiGRU	200 D	Simple	**0.7713**	0.7894	0.7560	0.8344

Table 8.5 – Results of the tuned model, after 50 trials of Bayesian search

2) Hyperparameter tuning for the best transformer model – BART

Hyperparameter	***Values Tested***
learning_rate	1e-5, 2e-5, 3e-5, 5e-5
per_device_train_batch_size	8, 16
num_train_epochs	3, 4, 5
weight_decay	0.0, 0.01
warmup_steps	0, 100, 500
gradient_accumulation_steps	1, 2
dropout	0.1, 0.2

Table 8.6 – Different values for hyperparameters for BART

Hyperparameter	***Value***
learning_rate	2e-05
per_device_train_batch_size	16
num_train_epochs	5
weight_decay	0.0
warmup_steps	0
gradient_accumulation_steps	1
dropout	0.1

Table 8.7 – Values obtained from the hyperparameter tuning

Model	*F1 Macro*	*F1 Precision*	*F1 Recall*	*Accuracy*	*Epochs*	*Validation Loss*
BART	**0.8417**	0.8408	0.8441	0.8763	5	0.4280

Table 8.8 – Results of the tuned BART after 20 trials of Bayesian Search (Tree-structured Parzen Estimator (TPE))

Model	*Sampling Technique*	*F1 Macro*	*F1 Precision*	*F1 Recall*	*Accuracy*	*Epochs*	*Validation Loss*
BART	None	**0.8417**	0.8408	0.8441	0.8763	5	0.4280
	Random undersampling	0.7868	0.7608	0.8303	0.8229	5	0.547270

Table 8.9 – Results of the tuned BART with and without random undersampling

Appendix 9 – Evaluation on test set and error analysis

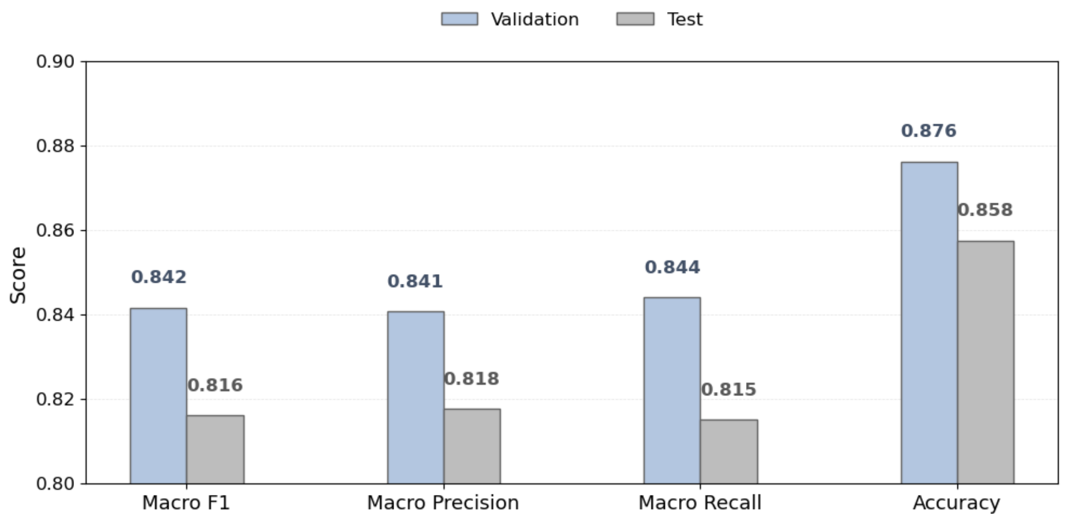

Figure 9.1 – Performance metrics for Validation and Test Set

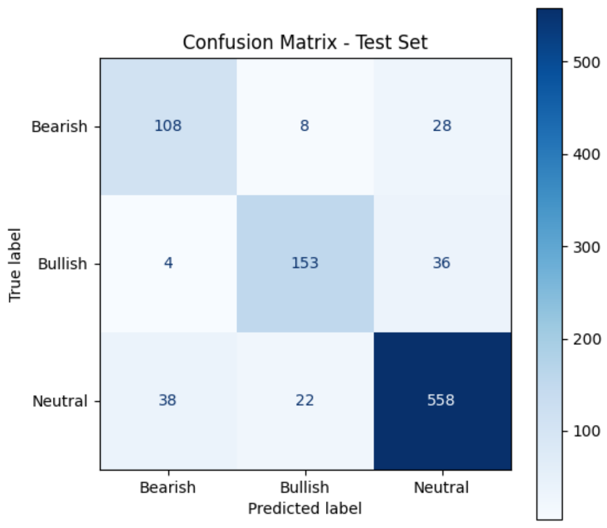

Figure 9.2 – Confusion Matrix for the predictions on Test Set

Index	*Tweet*	*Predicted*	*True Label*
2971	OPEC+ deal is much weaker than it looks	Neutral	Bearish
403	Fed policymakers working to limit damage as pandemic puts U.S. economy on pause	Neutral	Bearish
7698	$WDC - Western Digital May Have Trouble Pushing Stock Price Higher. Subscribe to Seeking Alpha for more:	Neutral	Bullish

Table 9.1 - Examples of tweets that were “Bearish” but were predicted as other classes

Index	*Tweet*	*Predicted*	*True Label*
3853	October Architectural Billings Index: Finally, Signs Of Strength.	Neutral	Bullish
7919	$TOPS moving good today. Just play the movement. But don't get stuck holding the bag.	Neutral	Bullish
295	One analyst says TD could unlock "significant value" by selling its minority stake in Ameritrade after its purchase…	Neutral	Bullish

Table 9.2 - Examples of tweets that were “Bullish” but were predicted as other classes

Index	*Tweet*	*Predicted*	*True Label*
1253	Apple canceled the premiere of "The Banker" from the AFI festival as it looks into unspecified "concerns" about the…	Bearish	Neutral
4127	Another bridge has collapsed in Italy	Bearish	Neutral
1434	Emirates Pares Back Wide-Body Airbus Order Amid Fleet Review	Bearish	Neutral

Table 9.3 - Examples of tweets that were “Neutral” but were predicted as other classes