Text Mining Project

Master’s in Data Science and Advanced Analytics

NOVA Information Management School

Universidade Nova de Lisboa

Stock Sentiment

Predicting Market Behavior from Tweets

Report

Group 25

Bruna Simões, 20240491

Catarina Ribeirinha, 20240507

Marco Galão, 20201545

Margarida Cardoso, 20240493

Spring Semester 2024-2025

TABLE OF CONTENTS

1. Introduction

The full project is available on our GitHub repository [Bib. 0] for further exploration and collaboration.

The main goal of this project is to develop a model capable of classifying stock market sentiment from tweets into three categories, that indicate neutral, positive or negative sentiment. Predicting sentiment in the context of financial markets is particularly relevant, as stock prices are heavily influenced by investor perception and their reaction to economic and financial developments. [Bib. 12]

This task has plenty of challenges typical of social media data, such as informal language, noise, short text length, links and high-class imbalance. To address these challenges, the project consists on a comprehensive text mining pipeline that includes data cleaning and exploring, several preprocessing techniques tailored to financial language and context, multiple feature engineering strategies (from *Bag of Words* and *TF-IDF* to word embeddings like *GloVe* and transformer-based embeddings such as *Text Embedding 3 Small*), and a wide range of classification models, from classical approaches like *Logistic Regression* and *XGBoost* to deep learning methods like *LSTM* and Transformers (*DistilBERT* and *Twitter RoBERTa*).

The project was structured into modular notebooks covering different stages of the workflow, from data exploration and preprocessing to feature engineering, model training, tuning, and final evaluation. [Figure 1]

2. Data Exploration

After uploading the datasets, verifying their integrity and structure was the first step. No duplicates or missing values were found in the file, and both key columns (*text* and *label*) had the correct data types (string for text and integer for label). The dataset consists of 9.543 rows, each representing a tweet about the stock market (*text* column), and each is assigned a label based on tweet content: Bearish (0) expresses a pessimistic outlook, Bullish (1) represents an optimistic outlook, and Neutral (2) expresses no clear sentiment.

The first thing that stood out was the high imbalance in the dataset, with almost 65% of tweets being categorized as Neutral [Figure 2, Table 1]. This imbalance can impact model performance, especially in detecting the minority classes. To address this issue, we implemented and tested resampling techniques and used evaluation metrics that are sensitive to class distribution (such as macro F1-score).

To dig deeper into the specifications, a temporary feature was created: *word_count.* On average, each tweet in the training CSV file contains approximately 12 words, with 75% of them having 15 words or fewer. The maximum tweet length is 31 words, which falls within a reasonable range for Twitter data. [Figure 3] However, a noteworthy observation is the presence of very short tweets, with some containing as few as 1 word. At this stage, with the text still full of stop words (such as “the”, “of”, “for”, etc.), it is not reasonable to draw insights, this analysis is postponed to the train partition until after preprocessing.

Following the word count analysis, an encoding check was performed, and 39 tweets were identified as containing emojis. Most tweets were also detected as being written in English, and the majority of those not flagged as English were, in fact, misclassified. Therefore, we chose not to filter out rows that were not identified as English.

After pre-processing the text, we re-examined the word count per tweet. Results showed a reduction in length, with most tweets being compressed to less than 15 words and with peak frequency around 7-9 words. The spread of the distribution is also tighter and it’s rare to exceed 15 words, the tail being virtually eliminated. This change in distribution confirms that the original data set contained a lot of syntactic noise that was removed during pre-processing [Figure 4]. The most frequent word is naturally "percent," followed by "stock," "market," and "quarter," the foresees as predicted by the financial nature of the data set and which dominate the language. "USA," "marketscreener" (a very frequent source referent), and "report" continue with the economic direction of the tweets. Other words such as "hedge," "fund," "share," and "dividend" suggest that most of these tweets are technical, suggesting a professional or institutional commentary as opposed to amateur or colloquial commentary. Furthermore, the presence of geopolitical indicators such as "China" and "USA" suggests the internationalized nature of the topics. [Figure 5]

It's interesting to examine the most common words in each label, both to assess the effectiveness of preprocessing and to get a better grasp of the data.

For the Neutral class, the most common words are terms like “stock”, “percent”, “market screener”, “result” and “report”. These are objective, non-directional words that state information without decidedly stating sentiment, which seems to fit the label neutral well. Directional words like "up", "down", "increase", or "decrease" are not very frequent.

On the other hand, Bullish tweets contain words with positive meaning such as “up”, “beat” (commonly used in the phrase “beat the market”), “rise”, “eps” (which means “earnings per share”), “gain”, “buy” and “revenue”. These words clearly reflect optimism and growth, validating their association with the bullish label.

For the Bearish category, the vocabulary shifts towards negative or cautionary terms such as “down”, “miss”, “fall”, “cut”, “sale” and very interestingly “oil”, which often appears in contexts involving volatility and instability. [Figure 6, 7]

Overall, the frequency of words by sentiment class confirms that the preprocessing pipeline did indeed retain semantically meaningful and sentiment-congruent lexicon, which is critical for the operation of classification models.

3. Data Preprocessing

To split the data, the hold-out method was used, with 20% of the data (1,909 observations) allocated for validation. The results of the split were saved in a *.pkl* file, which was then loaded in each notebook to ensure that the same partitions were consistently used.

In addition, a second *.pkl* file containing the raw (non-preprocessed) version of the same split was saved. This version was used specifically for transformer-based models, which rely on their own tokenization and preprocessing pipelines.

After this, and to prepare the tweets for model training, a custom function was implemented, that applies several preprocessing steps. These steps ensure standardization, removal of noise, and preservation of meaningful patterns, particularly those relevant to financial sentiment:

1. **Encoding Fix and Emoji Removal:** Encoding issues are addressed by employing the *ftfy* library to correct corrupted characters and emojis are removed.
2. **Lowercasing:** All text is converted to lowercase.
3. **Preservation of Financial Entities and Context:** Special attention was taken to preserve meaningful financial entities:
    a. **Indexes** like "S&P 500", "Dow Jones", and "Nasdaq Composite" get mapped to sp500, dowjones, and nasdaq_composite respectively.
    b. **Regions** like "US" and "UK" are normalized to USA and UK.
    c. **Quarter** references such as “Q1 2021” or “3Q2020” are standardized into a format like quarter_1 2021.
    d. **Cashtags** like $AAPL are replaced with TICKER_AAPL, preserving stock mentions intact for sentiment correlation.
    e. **Percentages** are tagged with PCT_INCREASE / PCT_DECREASE and labeled as percent values.
    f. **Monetary values** ($1200, €300) are replaced by the token costvalue.
4. **Contraction Expansion:** Contractions such as “can't”, “won't” and “isn’t” are expanded using the contractions library. This helps the models recognize better the negations and improves semantic interpretation.
5. **Removing unwanted patterns:** URLs, mentions, and special characters are removed. Hashtags lose the # but keep the critical word. Letters that are replicated are cut off (e.g., "soooo" to "soo") in an attempt to limit noise.

6. **Stopword Filtering with Exceptions:** The English stopwords in general are removed using the stopwords list of NLTK, but major direction and sentiment-bearing words like "up", "down", "crash", and "gain" are not dropped even if they exist in the stopwords list since they have meaning in the financial context.
7. **Lemmatization / Stemming:** Lemmatization is performed by default using NLTK's WordNetLemmatizer, reducing words to their base form (for example: "rising" to "rise"). Optionally, SnowballStemmer stemming can be used instead.

4. Feature Engineering

To transform the text data into structured numerical representations suitable for classification, we explored different feature engineering strategies: *Bag of Words (BoW)*, *Term Frequency - Inverse Document Frequency (TF-IDF*), *Word2Vec*, *GloVe*, *Text Embedding 3 Small* and *Twitter RoBERTa*. The choice of embeddings spans from basic to semantically informed methods, enabling a comparative analysis between conventional and more advanced text representations.

4.1 BoW

As a starting point, we implemented the *Bag of Words* model using binary encoding. In this representation, each tweet is converted into a vector that simply records whether a word appears or not - frequency is disregarded. This approach is particularly well-suited for short text like tweets, where repetition of words is rare, and presence can be more informative than count.

We limited the vocabulary size by setting *max_df* = 0.7, which excludes words that appear in more than 70% of the tweets, thereby reducing sparsity and removing low-information features.

In addition, the *ngram_range* was extended to include bigrams, enabling the model to capture richer contextual patterns. While unigrams provide a basic representation of word presence, incorporating bigrams allowed the model to capture short contextual patterns (e.g. price target, stock price [Figure 8]) that often carry sentiment in financial language. These phrase-level patterns are often more indicative of sentiment than individual tokens, especially in technical language (Manning et al., 2008) [Bib.1].

The resulting binary matrix was used as input for traditional machine learning classifiers. Although *BoW* ignores both word order and semantic relationships, it provided a simple and computationally efficient baseline. Moreover, it served as a reference point to assess the added value of more complex embedding methods later in the pipeline.

4.2 TF-IDF

To refine the *BoW* representation, we applied *TF-IDF*, which weighs words not only by how often they appear in a tweet, but also by how rare they are across the entire corpus. This weighting scheme reduces the influence of generic words and emphasizes those that are more discriminative for sentiment classification (Kouloumpis et al., 2011) [Bib.2] - particularly important in financial text, where terms like “quarter” or “fund” may be frequent but not sentiment-bearing on their own.

As in the *BoW* setup, we excluded words that appeared in more than 70% of the tweets and included bigrams in the vectorization process.

4.3 Word2Vec

Recognizing the limitations of frequency-based approaches, we shifted towards embedding models that capture semantic similarity between words. We trained a custom *Word2Vec* model using the *skip-gram algorithm* (*sg*=1), which is more effective in learning representations for infrequent terms - a useful property given the specialized terminology and linguistic variability present in financial communication.

We implemented two variations of this embedding. In the first, each tweet was tokenized and represented by averaging the vectors of its constituent words. This results in a fixed-size vector per tweet, allowing it to be directly used with machine learning classifiers such as logistic regression or random forest. Despite losing information about word order, this averaged representation captured important semantic relationships - enabling the model, for instance, to treat "drop" and "fall" as similar bearish signals, even if they appeared in different contexts (Mikolov et al., 2013).

The second variation was developed for use with *LSTM* models. Instead of collapsing the tweet into a single vector, each token was mapped to its corresponding *Word2Vec* embedding, resulting in a sequence of vectors per tweet. These sequences were then padded to the length of the longest tweet in the training set (30 tokens), ensuring a uniform input length when passing to *LSTM*, also preserving word order and enabling the model to learn temporal dependencies. This approach proves to be useful for capturing syntactic structures such as negation, repetition or intensification - elements that are often decisive in sentiment classification (Zhang & Wallace, 2017). [Bib. 3]

4.4 Glove

To further enhance semantic representation, we incorporated pre-trained *GloVe* embeddings using the *glove-twitter* model with 200 dimensions. This version of *GloVe* was trained specifically on Twitter data, making it particularly well-suited to the linguistic characteristics of our corpus - including informal expressions, abbreviations and social media-specific language.

As with Word2Vec, each tweet was represented by averaging the vectors of its constituent words. The main strength of the GloVe approach lies in its ability to embed words within a semantically rich space (Pennington, Socher, & Manning, 2014) [Bib. 4], even when those words are rare or domain specific. This proved especially useful in our setting, where financial terminology and sentiment-bearing phrases may not appear frequently enough to be well captured by simpler models.

We also extended this approach by training *LSTM* models on sequences of *GloVe* vectors, rather than averaging them.

4.4 Extra 1 - Text Embedding 3 Small

As an additional feature engineering strategy, *Text Embedding 3 Small*, a state-of-the-art model from Azure OpenAI, was employed. This model is part of OpenAI's latest generation of transformer-based embedding architectures, designed to capture rich semantic information from text by encoding it into dense vector representations. It leverages large-scale self-supervised pretraining on diverse textual data to learn contextual relationships, enabling it to generate suitable for a wide range of natural language understanding (NLU) tasks (OpenAI, 2023) [Bib. 5].

Among the available options provided by the university (*text-embedding-ada-002*, *text-embedding-3-small*, and *text-embedding-3-large*), this model was selected for its optimal trade-off between semantic richness, computational efficiency, and scalability. It produces more informative embeddings than *ada-002* while being significantly more lightweight and cost-effective than *3-large*.

The access to the model was established via the Azure OpenAI API, using credentials securely stored in a *.env* file. Embeddings were generated in batches of 32, with a one-second delay between requests to respect the API's rate limits. To ensure efficiency and reproducibility, the embedding process was managed through a caching mechanism using pickle files. The *embedding_te3s()* function coordinated the embedding generation or retrieval and subsequently applied a downstream classifier to the resulting feature vectors.

4.5 Extra 2 - Twitter RoBERTa Embeddings

As an additional feature engineering strategy, the *Twitter RoBERTa* model from *Hugging Face*'s Transformers library was utilized. Specifically, the *cardiffnlp/twitter-roberta-base* variant was chosen due to its pretraining on a large corpus of Twitter data, which makes it particularly well-suited for handling informal and social media text. This specialization allows the model to capture the unique linguistic patterns, slang, and contextual nuances present in tweets more effectively than general-purpose language models (Barbieri et al., 2020) [Bib. 6].

The implementation of *Twitter RoBERTa* embeddings followed a similar approach to the *TE3 Small* method. Text inputs were tokenized and processed in batches of 32 to generate dense vector representations from the model's *CLS* token output. To optimize efficiency and reproducibility, embeddings were cached using pickle files, preventing redundant computation in subsequent runs. The function *embedding_roberta()* orchestrated the embedding generation or retrieval and applied a downstream classifier to the resulting feature vectors.

5. Classification Models

After converting the text data appropriately to use as inputs for our models, we proceeded to apply a range of classification algorithms, including: *KNN*, *Naïve Bayes*, *Random Forest*, *Logistic Regression*, *XGBoost*, *LSTM (Long Short-Term Memory)*, *DistilBERT* (Transformer-encoder model), *Twitter RoBERTa* (Transformer-encoder model), and *GPT-4o* (Transformer-decoder model). The selection of these models was guided by a balance between simplicity and complexity: we aimed to implement both simple, interpretable models like *Logistic Regression* and *Naïve Bayes*, as well as more advanced and computationally demanding architectures, such as encoder-and decoder-based Transformer models.

5.1 Classical models

Given that several models were implemented and tested, to provide a clear and concise overview of the classical machine learning methods (*KNN*, *Naïve Bayes*, *Random Forest*, *Logistic Regression* and *XGBoost*) their description and explanation were summarized in Table 2.

5.2 LSTM

Regarding the *LSTM* network, this type of model is explicitly designed to overcome the challenges of long-term dependencies in sequential data. Standard *RNNs (Recurrent Neural Networks)* struggle with issues like vanishing gradients and difficulty retaining information from earlier time steps. *LSTMs* are one of the solutions presented by introducing a more complex architecture based on a cell state and multiple gates that regulate information flow. *LSTMs* allow through the gates, to distinguish which information should be discarded, what new information should be added and what information is passed to the next hidden state. This idea of retaining context across many time steps is fundamental, especially for tasks as ours, where the final classification is the result of understanding the full context.

5.3 Transformer (Encoder): DistilBERT

For our initial implementation of a Transformer-based model, we used *DistilBERT* (*distilbert-base-uncased*), a pre-trained model introduced in class, to directly classify tweets. The fine-tuning process involved adapting the pretrained model according to the three stock sentiment categories. The implementation followed standard *Hugging Face Trainer* workflows and consisted of the following steps:

1. **Dataset preparation**: Texts and labels were converted into *Hugging Face DatasetDict* objects.
2. **Tokenization**: Texts were tokenized using the model's associated tokenizer with truncation and padding.
3. **Model adaptation**: The model was loaded with a new classification head for 3 output labels.
4. **Training configuration** (Table 3): A set of training parameters was defined to balance performance and generalization.
5. **Evaluation**: Model performance was monitored using macro F1-score, precision, recall, and accuracy with the best model selected based on the lowest validation loss to ensure optimal generalization.

The training was performed using the *Trainer API*, and the best model was saved along with the tokenizer for downstream use.

We selected *DistilBERT* not only for its strong performance but also for its efficiency, which made it well-suited to our computational constraints.

DistilBERT is a compressed and more efficient version of *BERT*. It retains much of *BERT's* accuracy while being smaller, faster and requiring fewer parameters, resulting in quicker inference and reduced computational cost. Despite its smaller size, *DistilBERT* still delivers performance comparable to larger Transformer models, making it an ideal choice for our objective. This efficiency is achieved through knowledge distillation, since during training, the model learns to reproduce the behavior of *BERT* (a complex and powerful model) in a more compact and faster format.

5.4 Extra 1 - Transformer (Encoder): Twitter RoBERTa

To explore additional transformer-based classification approaches, the *Twitter RoBERTa* model (*cardiffnlp/twitter-roberta-base*) was fine-tuned on the target sentiment classification task, accordingly to the process described already. This model, previously employed in Section 4.5 for embedding extraction, was now utilized as a full encoder-based classifier by adapting it to the specific downstream problem.

Twitter RoBERTa is a variant of *RoBERTa* (Liu et al., 2019) [Bib 7] pretrained on a large corpus of English tweets, making it well-suited for handling noisy and domain-specific language typically found in social media. As discussed in Section 4.5, this model was selected due to its domain adaptation, semantic richness, and lightweight architecture compared to larger alternatives.

The fine-tuning process for *Twitter RoBERTa* followed the same steps previously applied to *DistilBERT*, including dataset preparation, tokenization, model adaptation with a classification head, training with the *Hugging Face Trainer API*, and evaluation based on macro F1-score, precision, recall, and accuracy.

5.5 Extra 2 - Transformer (Decoder): GPT-4o

In contrast to encoder-based models like *Twitter RoBERTa*, this approach explores decoder-style language modeling using *GPT-4o*. This model, a state-of-the-art autoregressive transformer, was employed via prompt-based inference to predict stock sentiment categories for tweets.

GPT-4o was selected from the models made available by the university, specifically due to its advanced reasoning capabilities and strong performance on instruction-based tasks. In this context, it was applied by prompting the model with a task-specific system instruction and a few labeled examples (few-shot learning). This enabled classification without any gradient-based fine-tuning, reducing computational cost while still leveraging the model's generalization capabilities (OpenAI, 2024) [Bib. 8].

The classification pipeline followed these main steps:

1. **Few-shot prompt construction** (Table 4): A small set of representative labeled tweets (one per class) was selected to guide the model's predictions.
2. **Prompt-based inference** (Table 5): Tweets were classified using the *chat.completions* API by sending structured prompts that included the system instruction, few-shot examples, and the target tweet.
3. **Batch processing, delay and caching:** To handle multiple tweets efficiently, the process was run in batches with a delay between requests to respect API rate limits, and caching was enabled to avoid redundant API calls.

Classification was performed using a deterministic prompt with temperature = 0, and the model was instructed to return only the class label (0, 1, or 2) for each input tweet. Additionally, A fallback mechanism returned "unknown" in case of API errors, such as content filtering, for example:

Error on input 'How to Make a Killing During t…': Error code: 400 - content_filter triggered (violence).

6. Evaluation and Results

The models were evaluated using four key metrics: macro F1-score, precision, recall, and accuracy. Given the class imbalance in the dataset, macro F1-Score was treated as the primary performance indicator, as it equally weights all classes regardless of their frequency.

The overall modeling strategy consisted of three stages:

1. **Baseline Modeling**: Initial models were trained using default configurations to address imbalance. At this stage, the focus was on macro F1-score performance on the validation set, without strong emphasis on overfitting, as untuned tree-based models often exhibit high variance by design. (Table 6)

2. **Hyperparameter Tuning**:

   The top-performing simpler models, such as *XGBoost*, *Logistic Regression*, and *LSTM*, were trained on oversampled and non-oversampled data to assess this class imbalance strategy and were tuned using grid search. This allowed for a comprehensive evaluation of key hyperparameters across a reasonable search space, given their relatively lower training cost.

   For the Transformer encoder models (*DistilBERT* and *Twitter RoBERTa*), hyperparameter tuning was conducted using random search with 15 trials per model via the *Hugging Face Trainer API* and *Optuna* integration. Although literature recommends at least 60 trials for robust search (O'Reilly Media, n.d.) [Bib. 9], a smaller number was used due to computational constraints. In addition to hyperparameter tuning, training was performed on oversampled data, and dropout regularization was applied to both hidden layers and attention mechanisms to mitigate overfitting. The search spaces explored are summarized in Table 7, 8, 9 and 10.

   Finally, for the Transformer decoder-based model (*GPT-4o*), no gradient-based tuning was performed. Since *GPT-4o* is accessed via an API and operates in a few-shot inference setting, it cannot be fine-tuned in the traditional sense. Instead, performance was controlled through prompt design.

3. **Final Predictions**: The best hyperparameter configuration per model was selected and retrained on the full training set. Lastly, the final predictions on the test set were generated to simulate real-world deployment and stored for evaluation against true labels.

6.1 Baseline Modeling

6.1.1 Traditional Models

The baseline experiments were designed to evaluate the interaction between embedding strategies and model architectures prior to any tuning. It is important to note that not all embedding-model combinations were tested:

- *Naïve Bayes* was only tested with *BoW* and *TF-IDF* embeddings due to its incompatibility with dense representations, which rely on continuous vector spaces and would require assumptions that break *Naïve Bayes*' independence model.
- *LSTM* models were only tested with *GloVe* and *Word2Vec*, since sequence modeling is meaningful only when each word is mapped to a dense vector. Advanced embeddings such as *TE3 Small* and *Twitter RoBERTa* already encode contextual information at the sentence level, making sequential modeling with *LSTM* redundant and meaningless.

Overall, model effectiveness depended strongly on the embedding used. Frequency based embeddings like *BoW* and *TF-IDF* produced strong results on the validation set [Figure 10] especially when paired with simpler models. For instance, both *Naïve Bayes* with *BoW* bigrams and *KNN* with *TF-IDF* unigrams achieved validation F1-Scores around 70% [Figure 11], confirming that even basic representations can yield good performance when matched with models that align well with their structural assumptions – *Naïve Bayes*, for example, is inherently suited to sparse count-based inputs and KNN benefits from the locality structure imposed by *TF-IDF* vectors. Notably, unigrams tended to be more stable across models, while bigrams, while added useful short-term context, harmed performance in some cases – especially with *XGBoost* and *KNN*.

In contrast, *XGBoost* – while highly effective in other settings – showed poor compatibility with *TF-IDF* embeddings, reaching only F1-Scores of 26% and 9% on the validation, with unigrams and bigrams respectively. However, *XGBoost* delivered strong results with dense embeddings. Notably, the combination with ***TE3 Small*** [Figure 11] yielded a validation F1-Score of 77% – the highest among all baseline configurations. It also performed well with *BoW* unigrams (F1-Score = 71%).

Pretrained dense embedding like *GloVe* also proved valuable, especially when paired with models capable of capturing sequence information*. **LSTM + GloVe*** [Figure 11] reached a validation F1-Score of 74%. Similarly, ***Logistic Regression* with *TE3 Small*** [Figure 11] matched *XGBoost*'s top result with an F1 of 77%, showing that even linear models can perform well when supported by strong pretrained features.

Word2Vec, on the other hand, was among the weakest embedding strategies overall. Most models scored below 53% in validation F1-Score, with *Logistic Regression* and *LSTM* reaching as low as 43% and 44%, respectively. This underperformance is likely due to the simplistic averaging approach used to generate sentence-level representations, which discards word order and may wash out sentiment from tweets.

RoBERTa embeddings offered competitive results across several configurations, reaching validation F1-scores up to 72% when combined with Logistic Regression. While not outperforming *TE3 Small*, *RoBERTa* consistently provided dense representations that led to stable performance across model types. Unlike *Word2Vec*, it retains contextual information, which may explain its superiority despite being used in frozen form.

Another important pattern across the experiments was the prevalence of overfitting. Many models, especially tree-based ones and *KNN*, achieved near-perfect scores on the training set, regardless of the embedding used. For instance, several *KNN* and *XGBoost* models reported training F1-Scores above 99%, while their validation performance dropped significantly. This pattern reinforces the need for regularization and hyperparameter tuning to control variance.

6.1.2 Pretrained Models

This section evaluates the pretrained models used - *DistilBERT, RoBERTa* and *GPT-4o* – in their default configurations, without applying oversampling, regularization or hyperparameter tuning.

DistilBERT [Figure 12] achieved a strong performance, with a validation F1 macro of 82%, accuracy of 86% and relatively balanced precision and recall (both 82%). Despite being a smaller model, outperformed all traditional models. However, the training scores (F1-Score = 99%) indicate some degree of overfitting.

RoBERTa [Figure 13] slightly improved results upon *DistiBERT* in every metric. It reached a validation F1 of 0.86, with precision and recall above 84% and accuracy of 88%. Its stronger performance highlights the benefit of deeper contextualization and more extensive pretraining. However, the training scores again approached 100% across all metrics, suggesting overfitting remains an issue.

The confusion matrices of these two models [Figure 15 and 16] further reveal that *RoBERTa* handled all classes more evenly, particularly distinguishing better between the minority classes (bullish and bearish), whereas *DistilBERT*, despite similar overall accuracy, exhibited a slight bias toward the majority neutral class, misclassifying more bearish and bullish tweets.

In contrast to these transformers*, GPT-4o* was used in a few-shot inference setup via API. The results were substantially weaker, with a macro F1 of 34%, accuracy below 45% and relatively low precision and recall. This underperformance is likely due to the lack of task-specific training.

Overall, ***RoBERTa*** and ***DistilBERT*** emerged as the best performing models, with overfitting being the main limitation observed, an issue that must be addressed through proper regularization and tuning, as discussed in the next section.

6.2 Hyperparameter Tuning

6.2.1 Traditional Models

In order to perform hyperparameter tuning, we first evaluated the performance of all previously described models in combination with their respective feature engineering techniques. The results showed that the majority of models exhibited overfitting, which is not surprising given the imbalance in the dataset.

Three models were selected for hyperparameter tuning: *Logistic Regression* with *TE3 Small* – the top-performing model with minimal overfitting*; XGBoost* with *TE3 Small* – selected due to its strong baseline performance, with the goal of reducing overfitting; and *LSTM* with *GloVe* – also selected based on promising results, in an attempt to improve generalization.

Surprisingly, the Logistic Regression model outperformed both. We hypothesize that this is due to the use of the 'ovr' (one-vs-rest) strategy, which treats the problem as three separate binary classification tasks - one per class. This decomposition appears to allow the model to specialize better in each sentiment category, ultimately leading to improved performance. This model yielded consistently strong results across all evaluation metrics, with low overfitting between the training and validation sets. As shown in Figure 17, the model achieved a macro F1-score of 82.91% on the validation set, a 6% difference to the training score of 89.60%, indicating a good generalization capability. Precision and recall also remained high, with validation values of 85.21% and 80.99% respectively, suggesting that the model is both accurate in its predictions and robust across classes despite the class imbalance. The overall accuracy reached 87.01% on the validation set, compared to 91.97% on the training data, reinforcing that the model is not overfitting significantly. These results validate the effectiveness of combining a simple yet interpretable classifier like Logistic Regression with semantically rich transformer-based embeddings.

On the other hand, both *XGBoost* with *TE3 Small* and *LSTM* with *GloVe* displayed signs of significant overfitting. As illustrated in Figure 18 and 19, both models achieved near-perfect scores on the training set across all metrics, including macro F1-score, precision, recall, and accuracy. However, their validation performance dropped considerably: *XGBoost* achieved a validation F1-score of 81.59% and accuracy of 86.22%, while *LSTM* scored a validation F1-score of 74.22% and accuracy of 81.19%. Despite our efforts to mitigate overfitting in *XGBoost* by reducing depth, introducing regularization, and limiting the number of splits, the model did not show meaningful improvements and was therefore discarded. The same outcome occurred with the *LSTM* model.

The same tuning was run over the same models but with oversampled data, to assess if results would improve and overfit would diminish. That was not the case, as observed in Figures 20, 21 and 22. Although *XGBoost* and *LSTM* showed minimal improvement in performance the overfitting was very similar to the models ran on non-oversampled data.

6.2.2 Pretrained Models

To mitigate the overfitting during the baseline evaluations, a set of hyperparameter tuning experiments was conducted on *DistilBERT* and *RoBERTa*. Specifically, we explored combinations of weighted loss functions, oversampling of minority classes and dropout regularization, followed by hyperparameter tuning based on the following methodology:

- **Class weights**: We computed a weighted loss function, assigning higher penalty to errors on underrepresented classes. Weights were set as the inverse of each label's frequency in the training set, resulting in the final weight vector (6.6153, 4.9646, 1.5447) for Bearish, Bullish and Neutral classes, respectively.
- **Oversampling**: Minority classes were balanced by resampling their examples with replacement until all classes matched the size of the majority class, helping the model learn equally from all labels.
- **Dropout**: Applied a dropout rate of 30% to hidden and attention layers (*hidden_dropout_prob=0.3, attention_probs_dropout_prob=0.3*) to prevent overfit.
- **Hyperparameter Tuning**: Conducted using *Optuna* with 10 trials to optimize performance. The tuning explored combinations of learning rate, batch size, number of epochs, and weight decay. The full search space configuration is summarized in Table 10.

The performance results of *RoBERTa* under these strategies are shown in Figures 23, 24 and 26 while *DistilBERT* results are presented in Figures 27, 28 and 30.

After applying class weighting, oversampling and dropout, both models demonstrated strong performance. *DistilBERT*, achieved a validation F1-Score of 80.85%, with precision and recall of 80.16% and 81.63%, respectively, reflecting a balanced performance. However, the training F1-Score remained nearly perfect at 99.89%, still indicating overfitting. While the introduction of dropout helped reduce the train-validation gap, it did not significantly alleviate the model's tendency to memorize the training data.

RoBERTa, on the other hand, showed a more favorable response to regularization, with clear improvements in both performance and generalization. It achieved a F1-Score of 86.79% on the training set and 82.94% on validation set, outperforming *DistilBERT*. The key difference lies in recall: this model reached 86.07%, which is 5.44% higher than *DistilBERT's* 81.63%, meaning it was able to correctly identify approximately 5.44% more true positive.

This gain in recall is further evidenced by the confusion matrices [Figures 25 and 29]. While both models were highly accurate for the majority class (neutral), *RoBERTa* misclassified fewer bullish and bearish tweets.

Given the results, hyperparameter tuning for both models was applied on top of the regularization strategies already in place. Both models exhibited improved generalization and more consistent validation scores. For *DistilBERT*, the validation F1-Score remained stable at 80.12%, while the training F1 dropped slightly to 99.33%, indicating a modest reduction in overfitting.

In contrast, *RoBERTa* benefitted more substantially from tuning. It achieved a validation F1-Score of 83.50%, with a reduced training F1-Score of 88.32%, substantially narrowing the train-validation gap. Precision and recall reached 81.63% and 85.82%, respectively, highlighting its strength in correctly identifying minority class instances and handling class imbalance effectively. This combination of high validation performance and controlled overfitting makes *RoBERTa* the most robust and reliable model in our pipeline, and thus the one selected for final evaluation and deployment.

6.3 Final Predictions

The final predictions on the test set were generated using the *Twitter RoBERTa* transformer encoder model, identified in the previous section as the best-performing approach. The distribution of predicted labels was 21.82% of Bearish (0), 25.04% of Bullish (1) and 53.14% of Neutral.

7. Challenges and Future Improvements

Despite promising results from some models, the project encountered several challenges that highlight opportunities for future enhancement.

Class Imbalance

One of the primary challenges was the substantial class imbalance, with Neutral tweets constituting nearly two-thirds of the dataset. While oversampling, dropout and metric selection helped mitigate this imbalance, future work could explore advanced data augmentation techniques, such as *Back Translation*. This involves translating an original tweet into another language (e.g., French) and back to English, preserving semantic content while enriching linguistic variety (e.g., transforming "The market is bearish" into "The market is going down") (Xie et al., 2019) [Bib.10]. Resampling strategies like *SMOTE, BorderLineSMOTE* and ensemble methods tailored for imbalanced data (e.g., *Balanced Random Forest*) were carried out and tested preliminary, but extending and exploiting these trials can be beneficial.

Model Overfitting

Models like *XGBoost* and *LSTM* demonstrated strong training performance but suffered from overfitting. This suggests the need for further regularization strategies, more diverse training data, or architectures better suited to capturing financial sentiment while resisting noise. Although cross-validation was not performed in this study, it could be incorporated in future work to provide more reliable performance estimates and better detect overfitting.

Leveraging Large Language Models for Financial Texts

The prompt-based approach using *GPT-4o* showed limited performance, likely due to the informal, domain-specific nature of financial tweets, which often include abbreviations, tickers, slang, and sarcasm. Such language is not well-represented in general-purpose pretraining corpora. Future work could explore fine-tuning decoder models on finance-specific datasets or employing *Retrieval-Augmented Generation (RAG)*, which enables models to ground responses in relevant external documents, such as financial news or reports (Lewis et al., 2020) [Bib.11].

Implement a different approach

Classifying tweets into three distinct classes in a single step is challenging, especially due to nuances in the language used or even the presence of mixed emotions, particularly in "neutral" tweets, without leaning clearly in one direction. This can confuse the model, especially when trained to handle all three labels simultaneously. Our proposed alternative would be to restructure the task into two distinct stages:

- A first stage, where the model is designed to detect sentiment, to determine whether a tweet expresses any sentiment (bullish or bearish) versus being neutral.
- A second stage applied only to the tweets identified as containing sentiment, allowing the model to distinguish between bullish (positive) and bearish (negative) sentiment.

This approach allows each model to concentrate on a simpler and more targeted objective, improving its ability to handle ambiguous cases, particularly when neutral tweets contain mixed emotional signals. It also provides greater flexibility, as each stage can leverage different models or feature representations that are best suited to their specific task.

Although this strategy introduces additional complexity and the risk of misclassifications in the first stage, it has the potential to enhance overall accuracy by breaking down a complex three-class classification problem into two more manageable binary tasks.

8. Conclusion

This project addressed the challenge of classifying stock market sentiment from tweets through advanced NLP and machine learning techniques. Despite the complexity of financial and social media language and the severe class imbalance, the results showed that combining adequate preprocessing techniques with powerful embeddings and pre-trained models can yield strong performance.

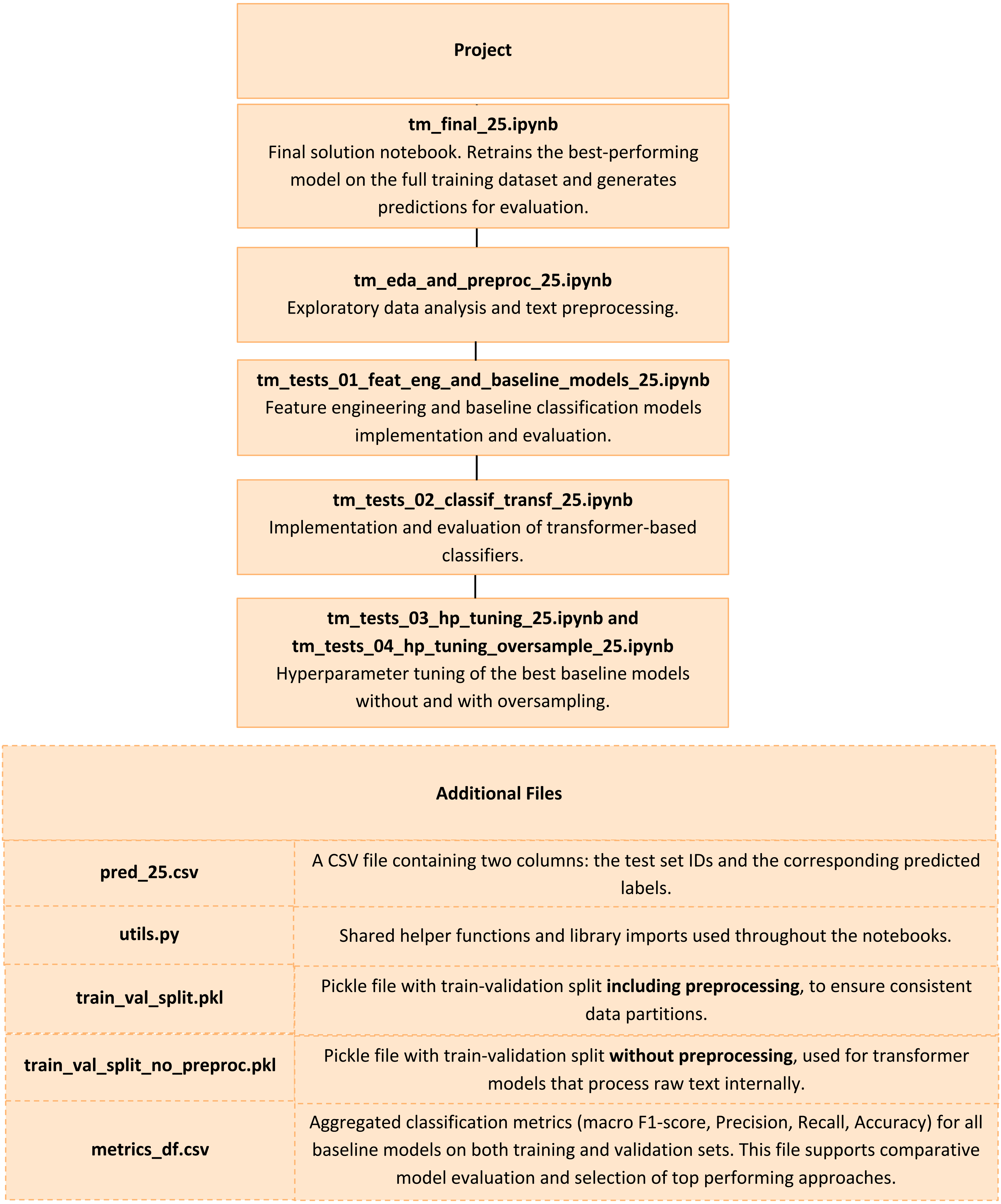
Indeed, a key insight from our work was the importance of embedding choice in model effectiveness. Simpler frequency-based representations like *BoW* and *TF-IDF* performed surprisingly well when paired with models aligned to their assumptions, like *Naïve Bayes* and *KNN*. Yet, when deeper context was required, transformer-based embeddings such as *TE3 Small* and *Twitter RoBERTa* substantially outperformed other methods, especially when combined with easily interpretable models like *Logistic Regression*.

Additionally, transformer models, particularly *DistilBERT* and *RoBERTa*, as expected elevated performance. After applying class weighting, oversampling and dropout regularization, *RoBERTa* achieved the best overall results. It showed notable improvements in recall and generalization, outperforming *DistilBERT* and traditional models. Despite some remaining overfitting, the use of dropout and hyperparameter tuning narrowed the performance gap between training and validation sets, indicating a more stable learning. Interestingly, *GPT-4o*, despite its advanced architecture, underperformed, possibly due to the limitations of few-shot inference without task-specific tuning.

Another key takeaway was the limited benefit of oversampling and hyperparameter tuning for certain models like *LSTM* and *XGBoost*, which continued to exhibit overfitting regardless of the configuration. This validated that model complexity alone does not guarantee better results, rather the alignment between embeddings, architecture and regularization is what triggers better performance.

Overall, despite improvements, persisting issues remain such as overfitting and ineffectiveness in solving class imbalance that underline the need for more refined augmentation techniques or reframing the model as a 3-step classification task, where each model is specialized in predicting one class. Ultimately, the *Twitter RoBERTa* model fine-tuned with regularization and class balancing was selected for final test predictions, although simpler models like Logistic Regression alongside with high-quality embeddings also performed competitively.

ANNEX A: VISUAL REPRESENTATIONS



Project

tm_final_25.ipynb
Final solution notebook. Retrains the best-performing model on the full training dataset and generates predictions for evaluation.

tm_eda_and_preproc_25.ipynb
Exploratory data analysis and text preprocessing.

tm_tests_01_feat_eng_and_baseline_models_25.ipynb
Feature engineering and baseline classification models implementation and evaluation.

tm_tests_02_classif_transf_25.ipynb
Implementation and evaluation of transformer-based classifiers.

tm_tests_03_hp_tuning_25.ipynb and tm_tests_04_hp_tuning_oversample_25.ipynb
Hyperparameter tuning of the best baseline models without and with oversampling.

Additional Files

pred_25.csv	A CSV file containing two columns: the test set IDs and the corresponding predicted labels.
utils.py	Shared helper functions and library imports used throughout the notebooks.
train_val_split.pkl	Pickle file with train-validation split **including preprocessing**, to ensure consistent data partitions.
train_val_split_no_preproc.pkl	Pickle file with train-validation split **without preprocessing**, used for transformer models that process raw text internally.
metrics_df.csv	Aggregated classification metrics (macro F1-score, Precision, Recall, Accuracy) for all baseline models on both training and validation sets. This file supports comparative model evaluation and selection of top performing approaches.

Figure 1 – Project Structure Diagram

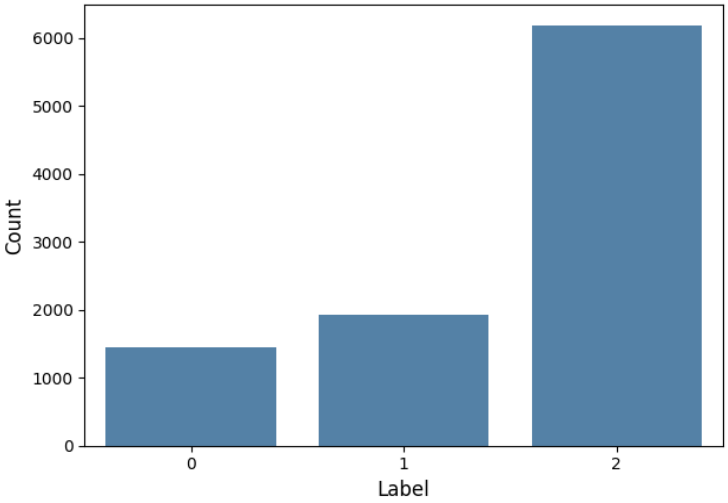


Figure 2 - Distribution of Label Values

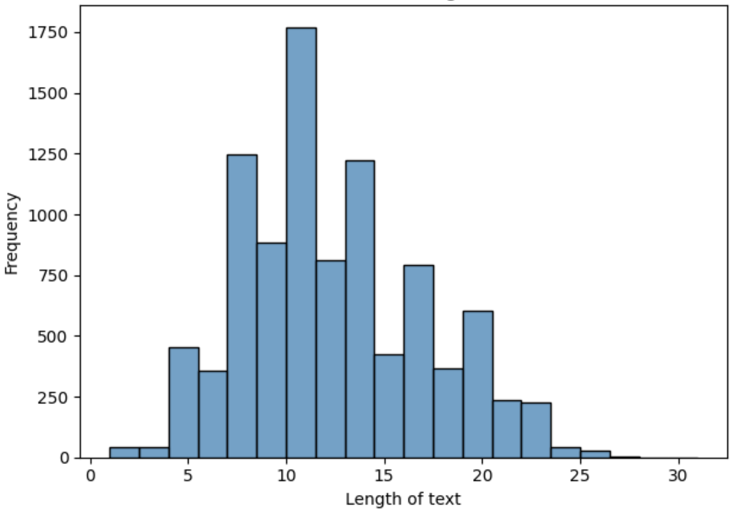


Figure 3 - Word Count Distribution before Pre-Processing

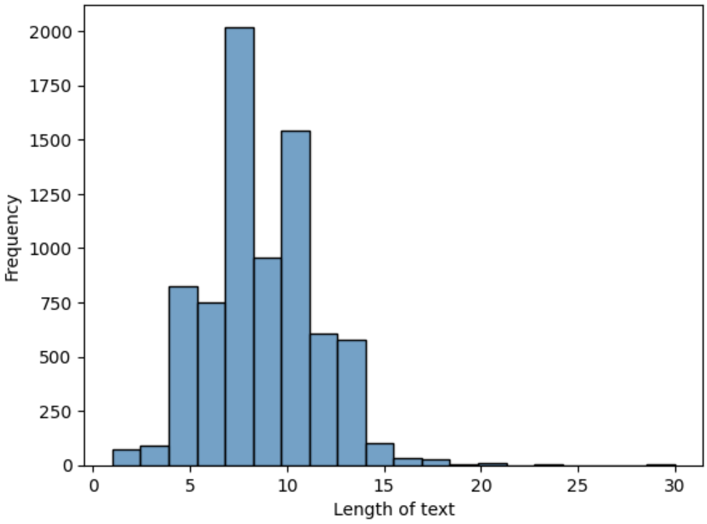


Figure 4 - Word Count distribution after Pre-Processing

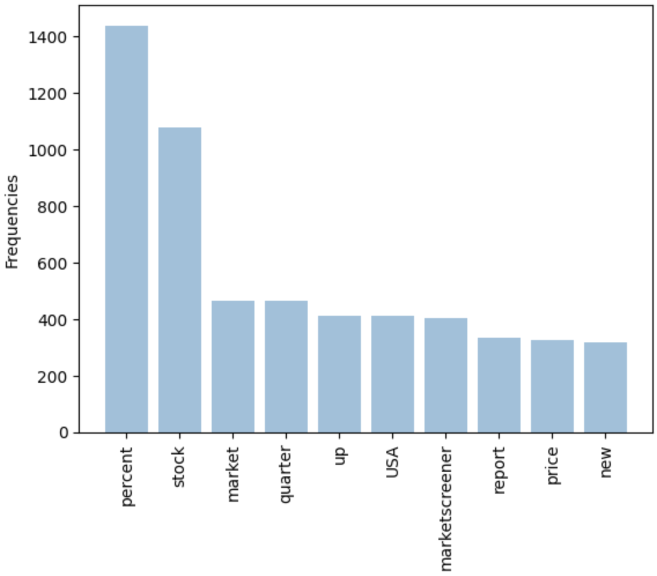


Figure 5 - Most popular words in the train split

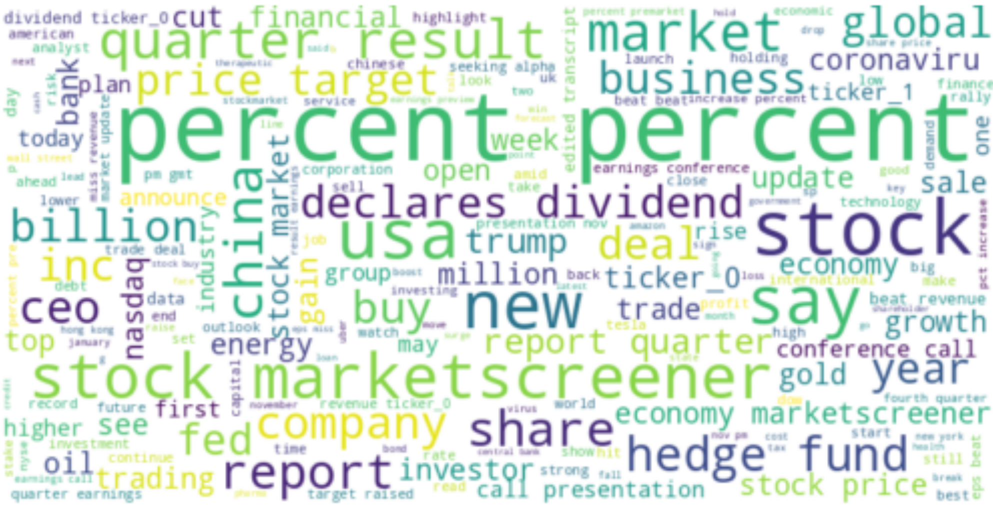


Figure 6 - Word cloud for the train split

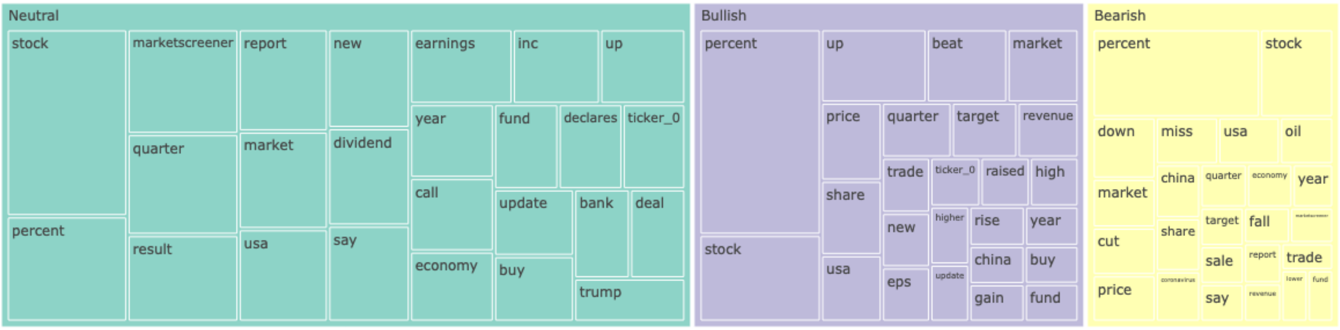


Figure 7 - Tree Map of word frequency for each label

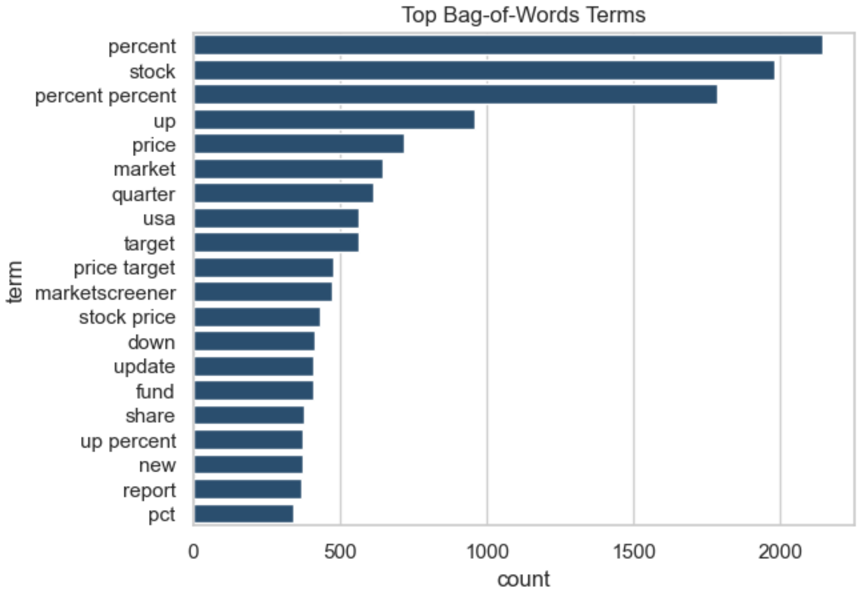


Figure 8 - Top unigrams and bigrams extracted using BoW

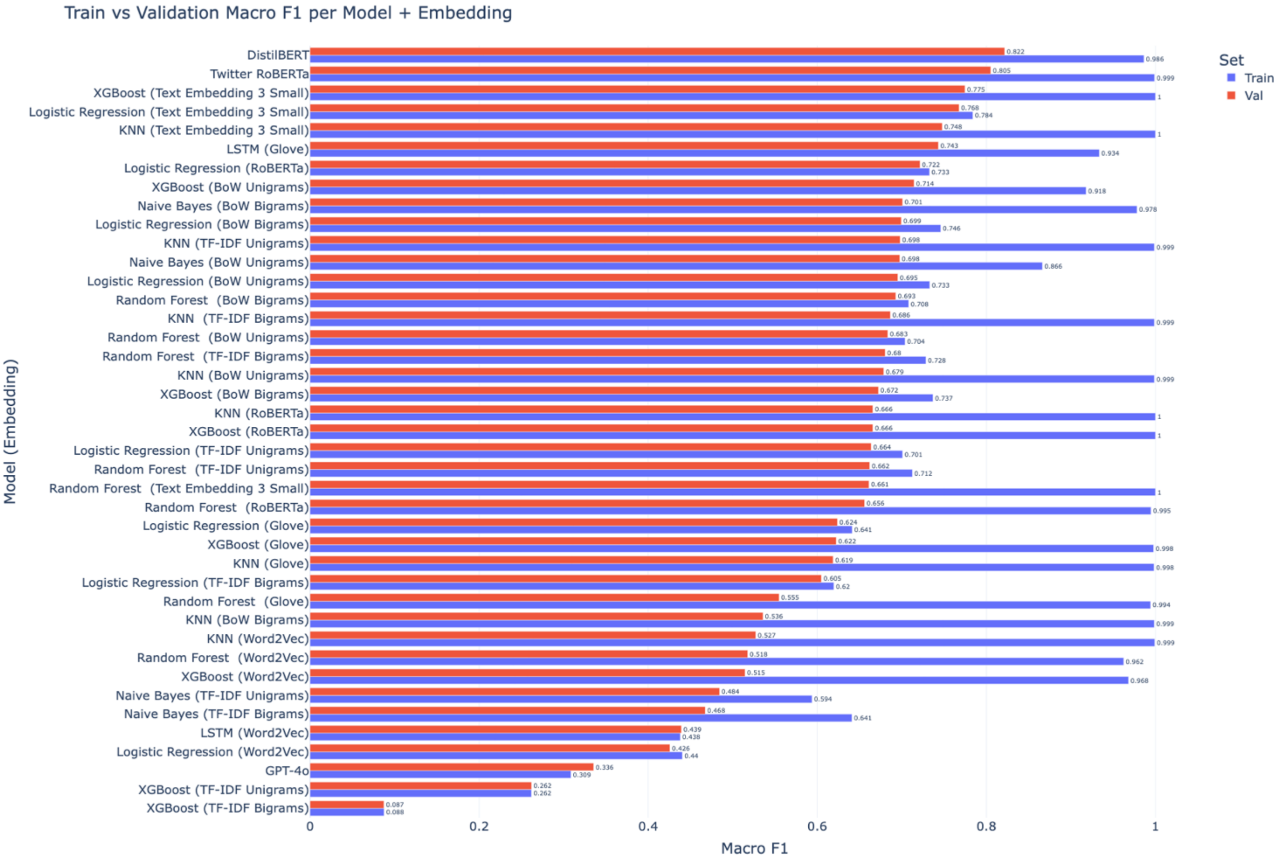


Figure 9 - Performance of Classical Models, LSTM and Transformers with the different Feature Engineering Techniques, using Macro F1-Score

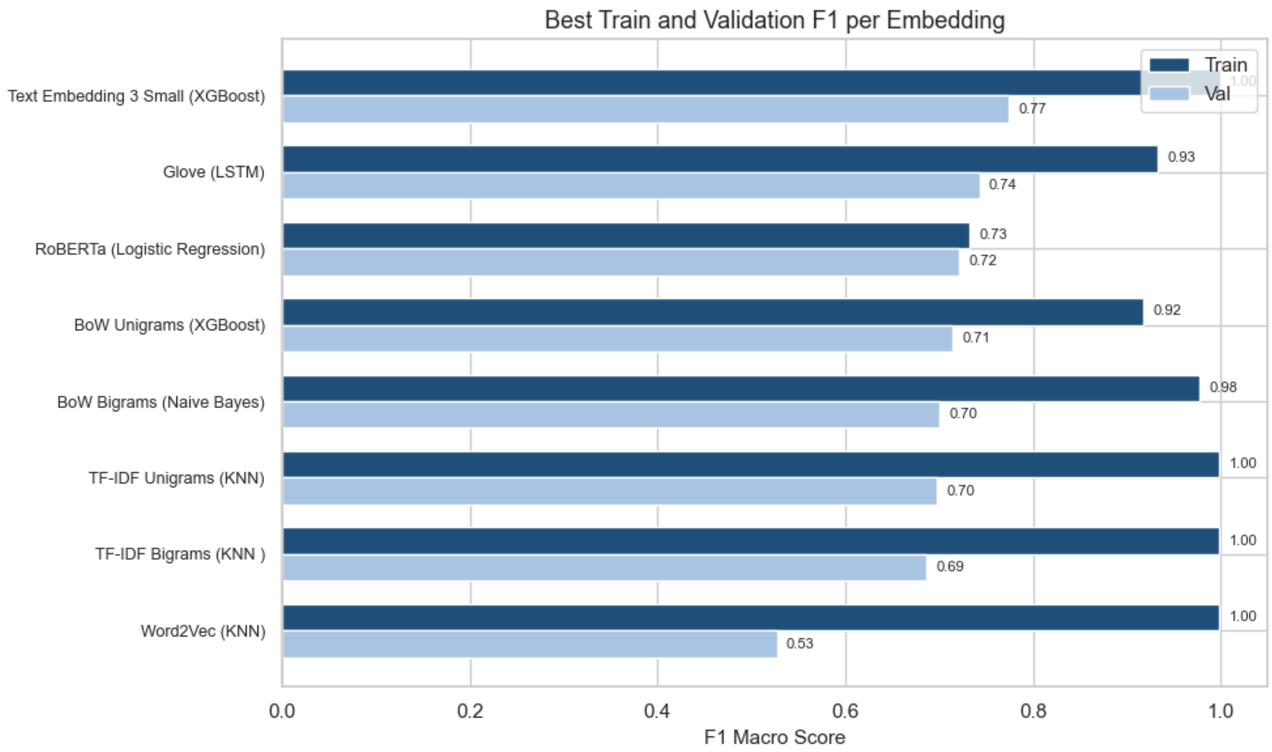


Figure 10 - Best train and validation F1-Score per embedding strategy

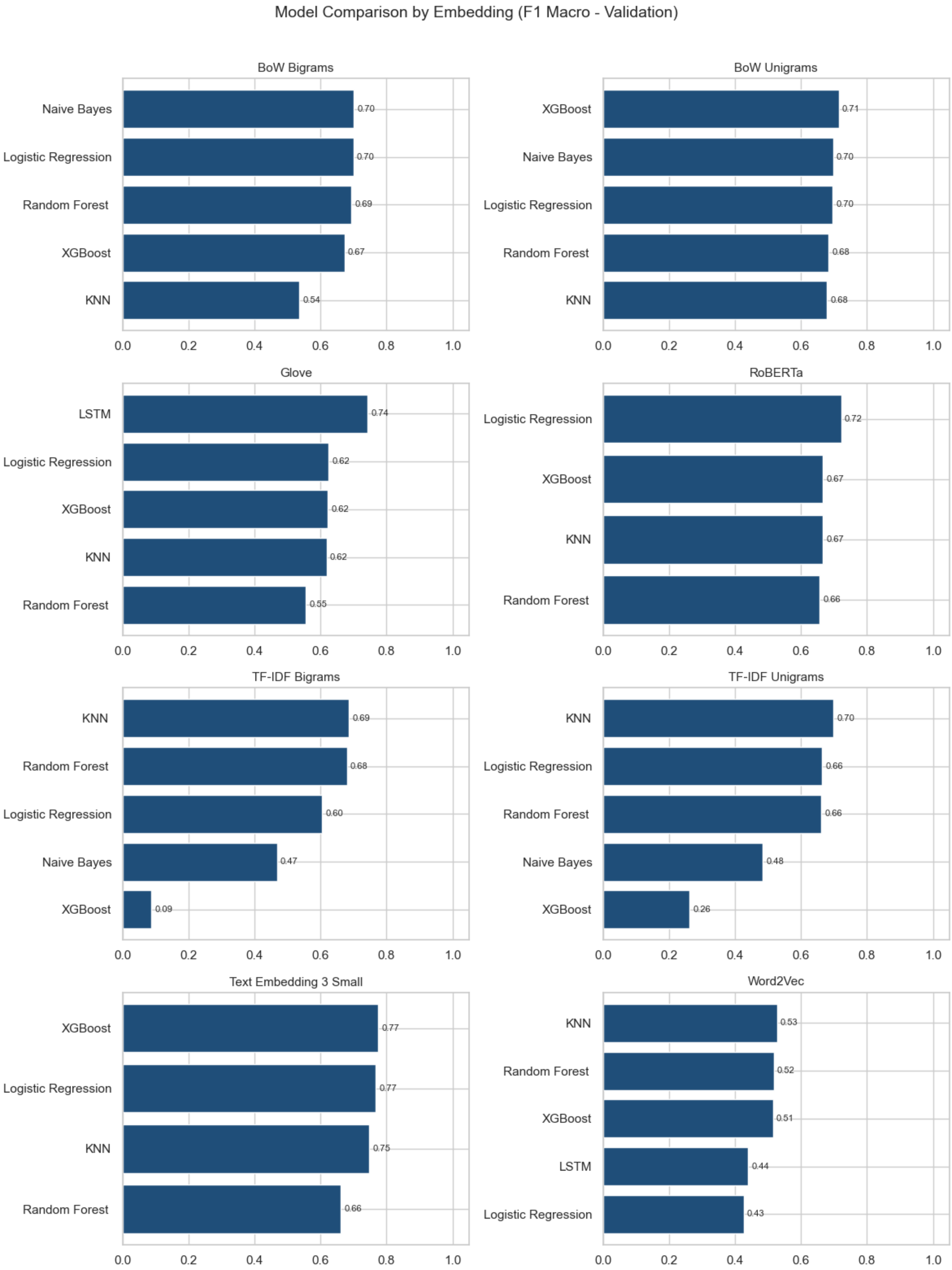


Figure 11 - Validation F1 scores of different models across each embedding strategy

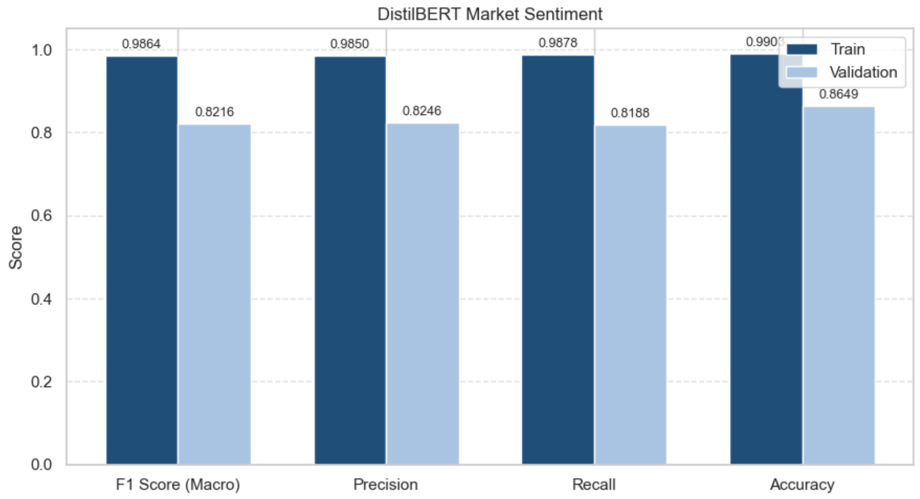


Figure 12 – DistilBERT Metrics before Hyperparameter Tuning

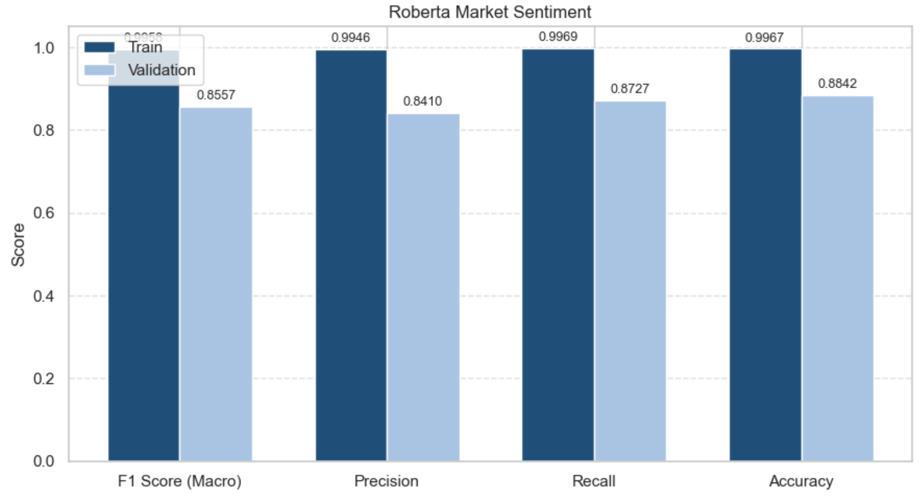


Figure 13 – Roberta Metrics before Hyperparameter Tuning

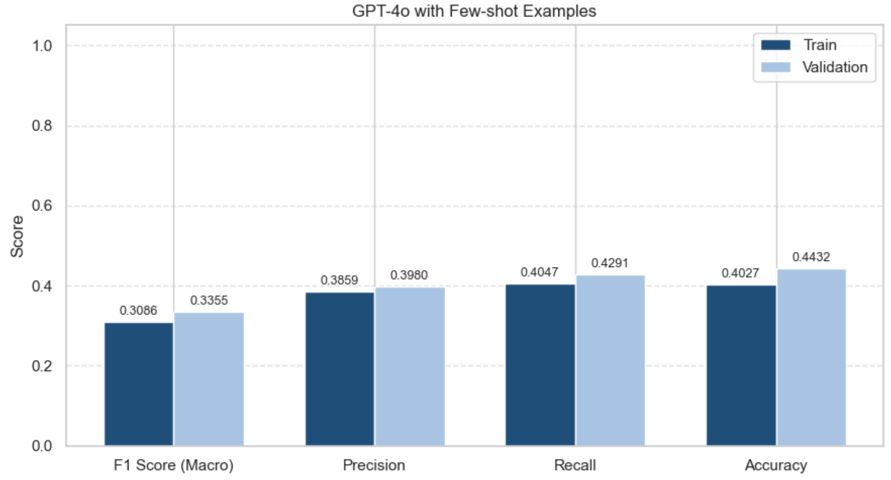


Figure 14 – GPT-4o Metrics before Hyperparameter Tuning

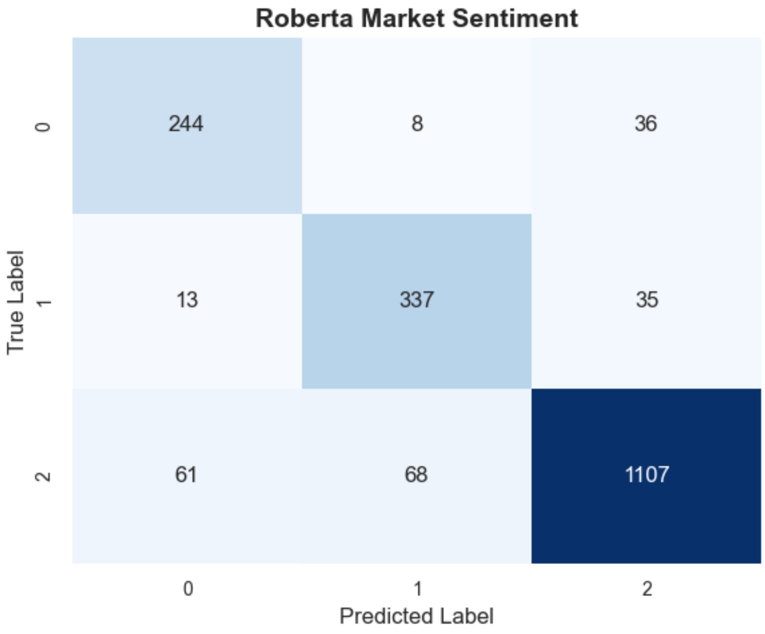


Figure 15 – Roberta Confusion Matrix before Hyperparameter Tuning

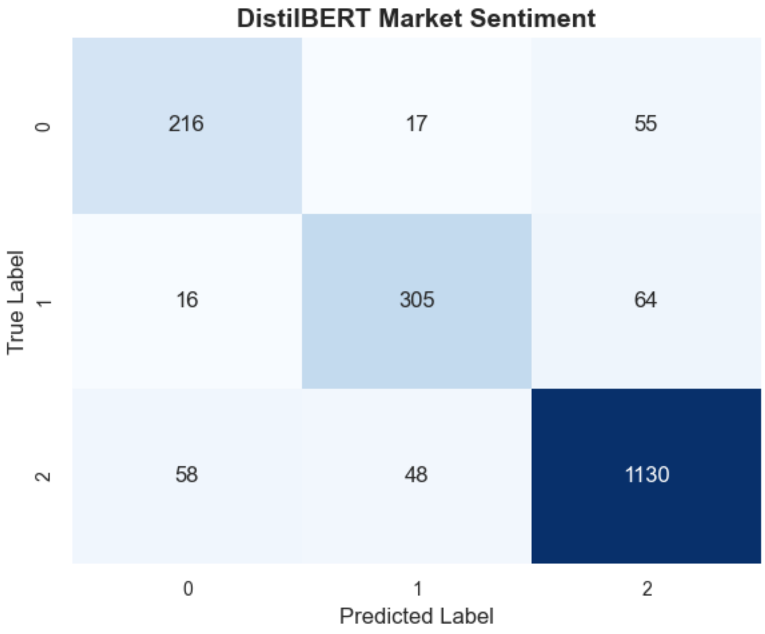


Figure 16 – DistilBERT Confusion Matrix before Hyperparameter Tuning

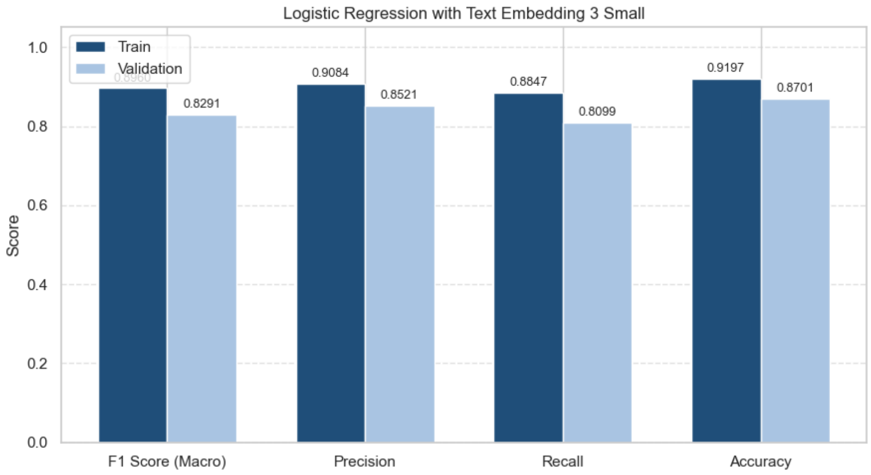


Figure 17 - Logistic Regression with text Embedding 3 Small Metrics after Hyperparameter Tuning

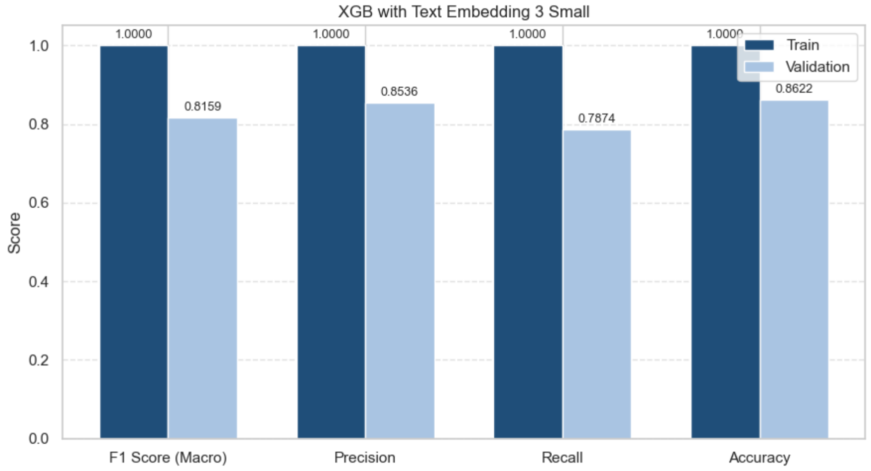


Figure 18 - XGB with text Embedding 3 Small Metrics after Hyperparameter Tuning

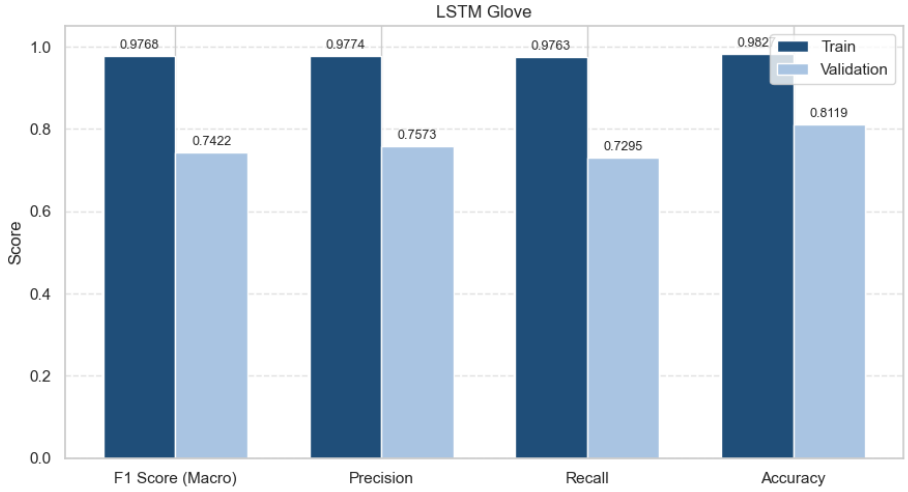


Figure 19 - LSTM with Glove Metrics after Hyperparameter Tuning

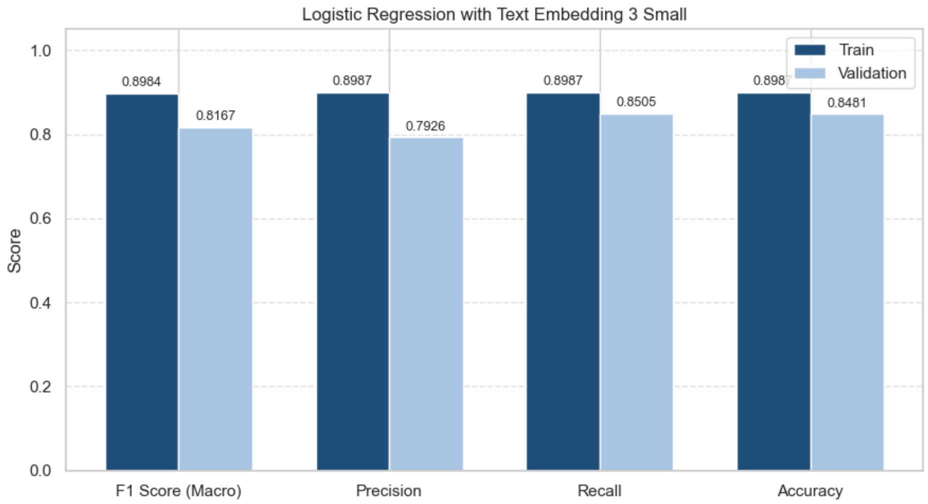


Figure 20 - Logistic Regression with text Embedding 3 Small Metrics after Hyperparameter Tuning with Oversampled Data

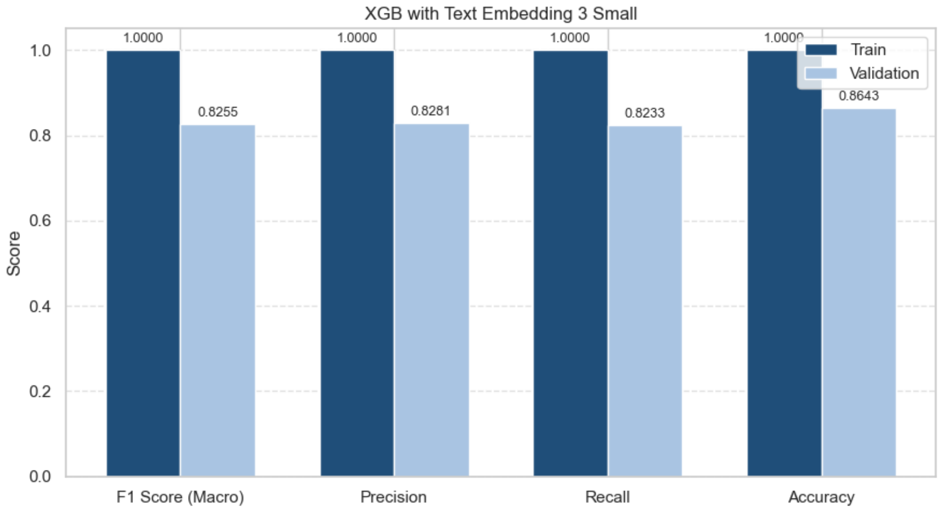


Figure 21 - XGB with text Embedding 3 Small Metrics after Hyperparameter Tuning with Oversampled Data

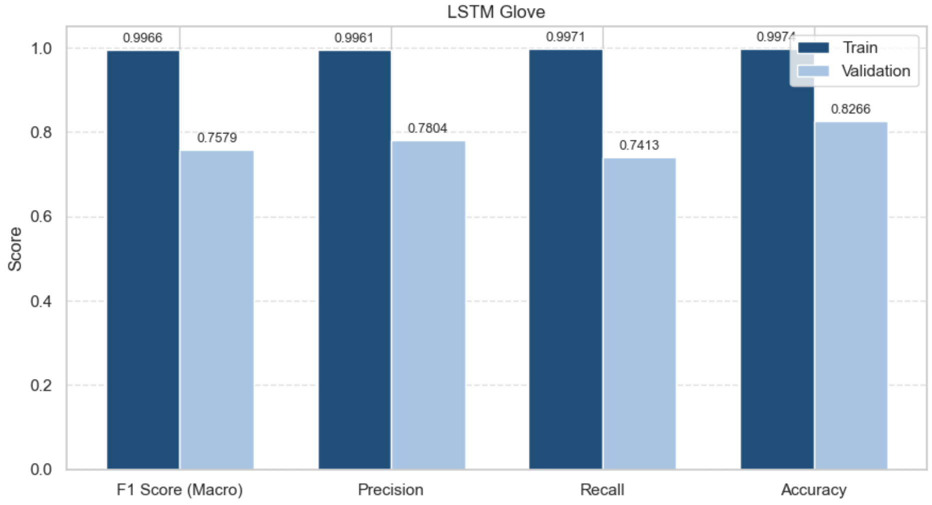


Figure 22 - LSTM with Glove Metrics after Hyperparameter Tuning with Oversampled Data

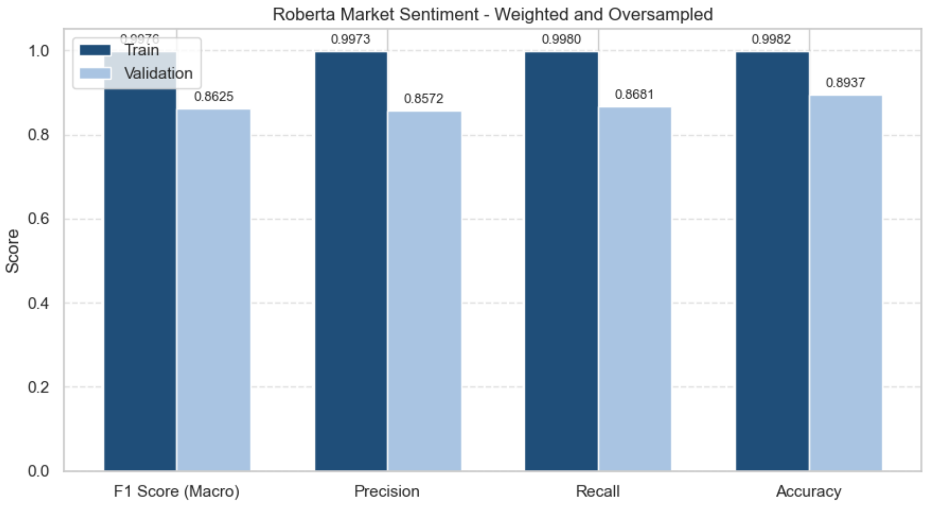


Figure 23 – RoBERTa metrics after applying Class Weighting and Oversampling

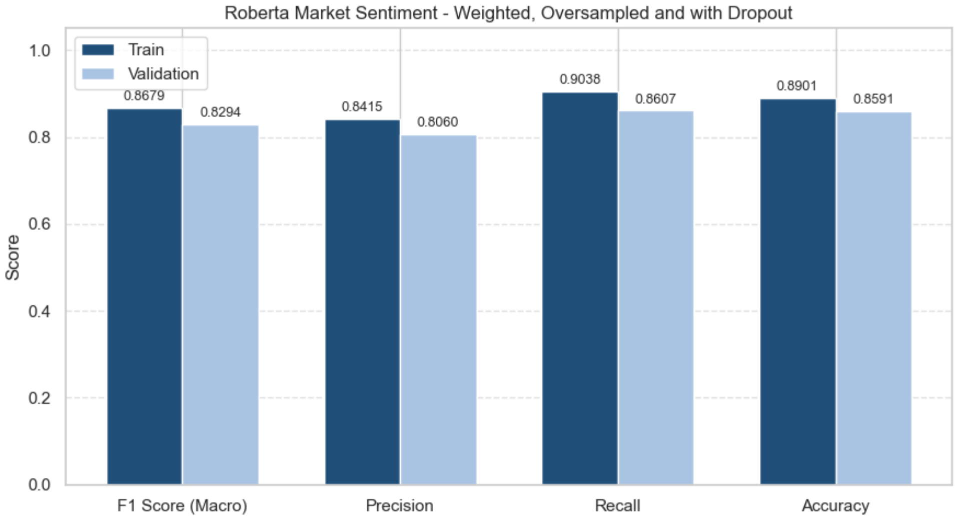


Figure 24 - RoBERTa metrics after applying Class Weighting, Oversampling and Dropout (30%)

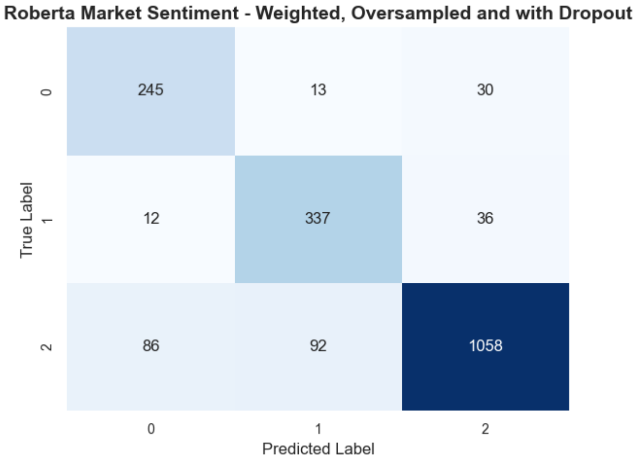


Figure 25 – RoBERTa Confusion Matrix after applying Class Weighting, Oversampling and Dropout (30%)

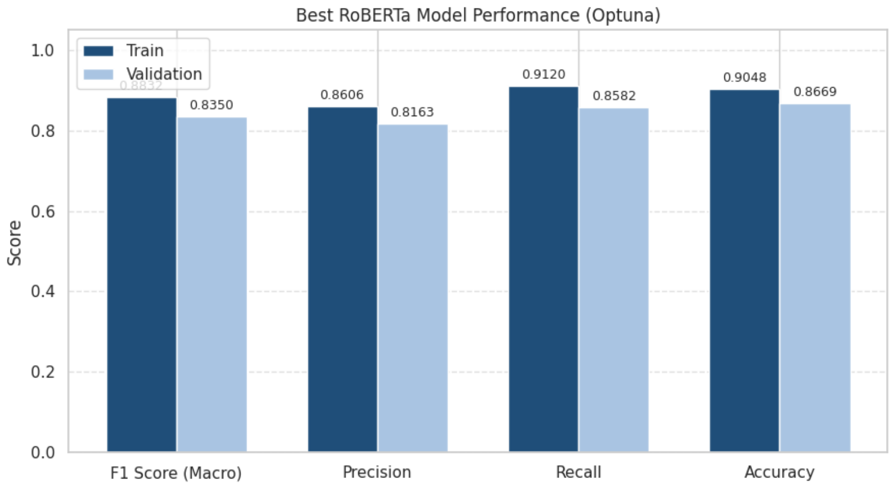


Figure 26 - RoBERTa metrics after Hyperparameter tuning

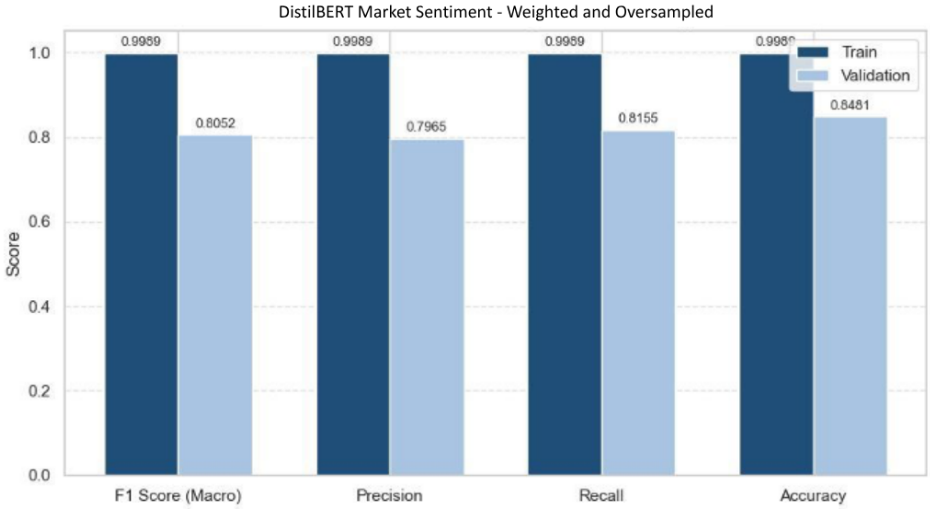


Figure 27 – DistilBERT metrics after applying Class Weighting and Oversampling

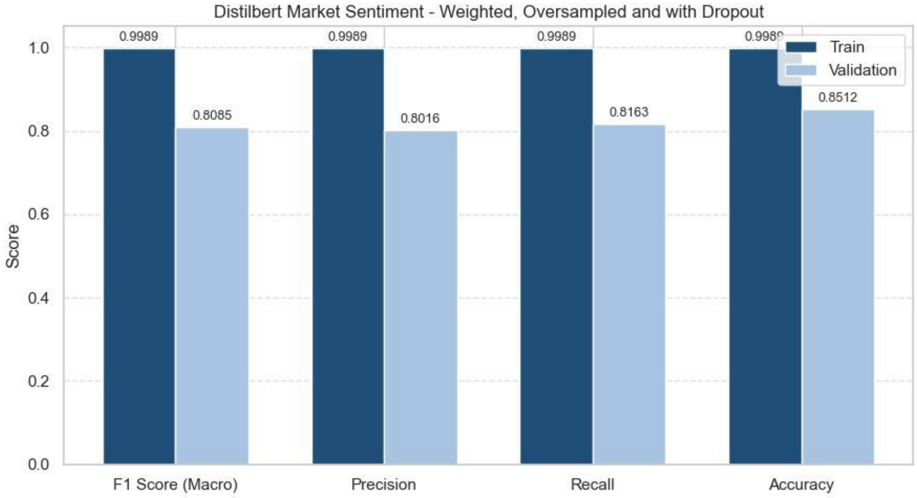


Figure 28 – DistilBERT metrics after applying Class Weighting, Oversampling and Dropout (30%)

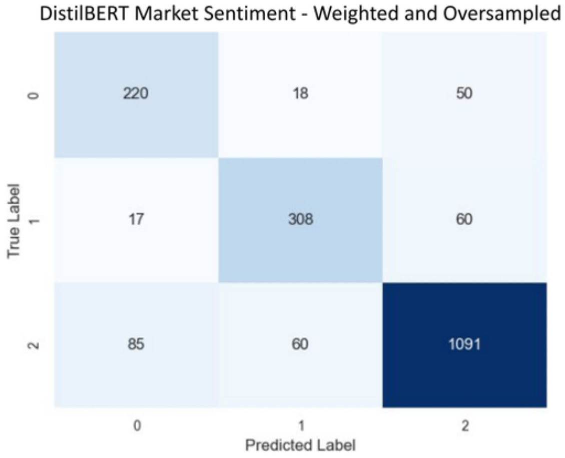


Figure 29 – DistilBERT Confusion Matrix after applying Class Weighting, Oversampling and Dropout (30%)

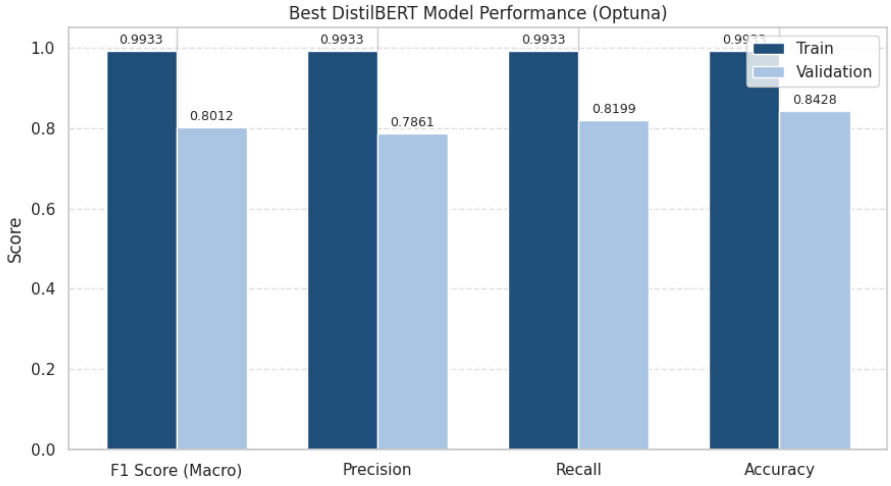


Figure 30 – DistilBERT metrics after Hyperparameter tuning

ANNEX B: TABLES

LABEL	PROPORTION
BEARISH (0)	64.7386%
BULLISH (1)	20.1509%
NEUTRAL (2)	15.1106%

Table 1 - Proportion of Label Distribution

MODEL	DESCRIPTION/EXPLANATION
KNN	KNN is an instance-based learning algorithm. It classifies data points based on the majority label among the *k* nearest neighbors in the feature space. In our case, KNN uses the vector representations of tweets to find the most similar examples in the training set and classify it likewise. It is a simple and interpretable method, however due to the sparse and high-dimensional nature of text data (especially with BoW or TF-IDF), its performance may be suboptimal.
Naïve Bayes	Naïve Bayes is a probabilistic classifier, based on the Bayes' Theorem, with a strong assumption of feature independence. This assumption makes it work particularly well with high-dimensional data, having often a surprisingly good performance, specially when using BoW or TF-IDF vectors. However, feature independence is rare in reality, which may lead to poor performance.
Random Forest	Random Forest is an ensemble learning method, it basically builds multiple decision trees during training and outputs the majority vote (for classification tasks). It is a more robust approach to deal with overfitting and non-linear relationships, since it can capture more complex interactions in textual data compared to linear models. However, in contrast to linear models, Random Forests are generally less interpretable and slower.
Logistic Regression	Logistic Regression is a linear classifier that models the probability of class membership using a function, normally softmax in multiclass. It is a highly interpretable model, commonly used in NLP tasks. It performs particularly well when the data is linearly separable and despite its limitations in capturing non-linear patterns, it can be highly effective.
XGBoost	XGBoost (Extreme Gradient Boosting) is an efficient and powerful machine learning algorithm based on gradient boosting. It builds an ensemble of decision trees sequentially, where each tree corrects the errors of the previous one. XGBoost is known for its speed, scalability and high performance, often outperforming other models in tasks like classification and regression. It uses regularization techniques to prevent overfitting and handles missing data well. Additionally, XGBoost is widely used in machine learning competitions and practical applications.

Table 2 – Description/Explanation of Classical Models

PARAMETER	VALUE	JUSTIFICATION
learning_rate	2e-5	standard fine-tuning learning rate for transformers, balancing convergence speed and stability.
per_device_train_batch_size	16	moderate batch size to fit most gpu memory constraints while maintaining efficient training.
per_device_eval_batch_size	32	larger batch size for evaluation to speed up validation without impacting training memory.
num_train_epochs	5	enough epochs to allow learning without overfitting; commonly effective for transformer fine-tuning.
weight_decay	0.01	regularization to prevent overfitting by penalizing large weights.
logging_steps	100	logs training status every 100 steps to provide timely updates without excessive overhead.
eval_strategy	"epoch"	evaluate model performance at the end of each epoch for consistent progress tracking.
save_strategy	"epoch"	save model checkpoints at each epoch to preserve best models and enable recovery.
load_best_model_at_end	true	automatically load the best model based on the specified metric after training completes.
metric_for_best_model	"loss"	use validation loss to select the best model checkpoint, as lower loss indicates better fit.
greater_is_better	false	because lower loss values are preferable, this is set to false.

Table 3 – Training Configuration of Encoder Transformer Models

TWEET	LABEL
Phathom Pharmaceuticals EPS Of -$9.30	0
JPMorgan likes Bausch Health in premarket analyst action	1
Penn Station Stinks, But the Neighborhood is Looking Up	2

Table 4 – Few-Shot Examples for GPT-4o Prompt Construction

ROLE	CONTENT
system	You are a financial sentiment classification assistant. Your task is to analyze short social media texts (tweets) that may influence or reflect investor sentiment regarding the stock market. Based on the content, classify each tweet into one of the following categories: 0, 1, 2. - Bearish (0): Suggests negative or pessimistic sentiment about the market or a stock. - Bullish (1): Suggests positive or optimistic sentiment. - Neutral (2): Does not express a clear opinion or is irrelevant to market sentiment. Respond only with the correct category label - no explanation.
user	Phathom Pharmaceuticals EPS of -$9.30
assistant	0
user	JPMorgan likes Bausch Health in premarket analyst action
assistant	1
user	Penn Station Stinks, But the Neighborhood is Looking Up
assistant	2
user	

Table 5 – Structure of Prompt Sent to GPT-4o for Sentiment Classification

MODELS	PARAMETERS	EMBEDDINGS APPLIED

KNN	n_neighbors = 10
	metric = 'cosine'
	weights = 'distance'
		- BoW (Unigrams/ Bigrams);
		- TF-IDF (Unigrams/ Bigrams);
		- Word2Vec (Minimum count=1/ Minimum count based on percentile);
		- Glove;
		- Text Embedding 3 Small;
		- Twitter RoBERTa.

Naïve Bayes	alpha = 1	- BoW (Unigrams/ Bigrams); - TF-IDF (Unigrams/ Bigrams). **Note**: Word2Vec, Glove and Extra methods weren't used because they can generate negative numbers and Naive Bayes is based on probabilities/ word frequencies, which must be non-negative.

Random Forest	n_estimators = 200
	criterion = 'gini'
	max_depth = 20
	min_samples_split = 10
	min_samples_leaf = 3
	max_features = 'sqrt'
	class_weight = 'balanced'
		- BoW (Unigrams/ Bigrams);
		- TF-IDF (Unigrams/ Bigrams);
		- Word2Vec (Minimum count=1/ Minimum count based on percentile);
		- Glove;
		- Text Embedding 3 Small;
		- Twitter RoBERTa.

Logistic Regression	penalty = 'elasticnet'
	solver = 'saga'
	l1_ratio = 0.5
	C = 0.15
	class_weight = 'balanced'
	max_iter=300
	multi_class='multinomial'
		- BoW (Unigrams/ Bigrams);
		- TF-IDF (Unigrams/ Bigrams);
		- Word2Vec (Minimum count=1/ Minimum count based on percentile);
		- Glove;
		- Text Embedding 3 Small;
		- Twitter RoBERTa.

XGBoost	objective='multi:softmax'
	num_class=3
	eval_metric='mlogloss'
	use_label_encoder=False
	learning_rate = 0.05
	max_depth_xgboost = 6
	n_estimators_xgboost = 300
	subsample = 0.8
		- BoW (Unigrams/ Bigrams);
		- TF-IDF (Unigrams/ Bigrams);
		- Word2Vec (Minimum count=1/ Minimum count based on percentile);
		- Glove;

	colsample_bytree = 1
	scale_pos_weight = 1
		- Text Embedding 3 Small;
		- Twitter RoBERTa.

LSTM	batch_size=16
	epochs=10
	sg=1
	learning_rate_lstm=0.001
	optimizer=Adam(learning_rate=learning_rate_lstm)
	loss='categorical_crossentropy'
	metrics=['categorical_accuracy', Precision(name='precision'), Recall(name='recall'), AUC(name='auc', multi_label=True)]
	units=64
	dropout=0.3
		- Word2Vec;
		- Glove.
		
		Note: BoW and TF-IDF are not suitable for LSTM due to lack of sequential structure.

Table 6 – Baseline Modeling Parameters and Embeddings Applied (Classical Models and LSTM)

PENALTY	SOLVER	C	L1-RATIO	CLASS WEIGHT	MULTICLASS
L1	saga	0.01, 0.1, 1, 10	-	none, balanced	ovr, multinomial
L2	saga	0.01, 0.1, 1, 10	-	none, balanced	ovr, multinomial
elasticnet	saga	0.01, 0.1, 1, 10	0.0, 0.5, 1.0	none, balanced	ovr, multinomial

Table 7 - Hyperparameter Search Space Logistic Regression

LEARNING RATE	MAX_DEPTH	N_ESTIMATORS	SUBSAMPLE	COLSAMPLE_BYTREE	SCALE_POS_WEIGHT
0.2, 0.1	4, 6, 8	100, 500	0.7, 1	0.8, 1	1

Table 8 - Hyperparameter Search Space XGBoost

UNITS	DROPOUT	LR
64	0.3	0.005, 0.001
64	0.4	0.005, 0.001
128	0.3	0.005, 0.001
128	0.4	0.005, 0.001

Table 9 - Hyperparameter Search Space LSTM

HYPERPARAMETER	VALUES EXPLORED
Learning Rate	Log-uniform: 1e-5 - 5e-5
Number Of Epochs	Integer: 4 - 6
Weight Decay	Float: 0.01 - 0.15
Batch Size (per Device)	8, 16, 32
Warmup Ratio	Float: 0.0 - 0.2

Table 10 - Hyperparameter Search Space for Transformer Encoders (DistilBERT and Twitter RoBERTa)

MODEL	EMBEDDING	F1 MACRO		PRECISION		RECALL		ACCURACY	
		Train	Val	Train	Val	Train	Val	Train	Val
KNN	BoW Unigrams	0.9987	0.6786	0.9987	0.7388	0.9988	0.6541	0.9991	0.7873
	BoW Bigrams	0.9986	0.5358	0.9983	0.546	0.9989	0.6052	0.999	0.5783
	TF-IDF Unigrams	0.9987	0.698	0.9983	0.7478	0.9992	0.6726	0.9991	0.7968
	TF-IDF Bigrams	0.9987	0.6864	0.9985	0.7389	0.999	0.6605	0.9991	0.7889
	Word2Vec	0.9991	0.5272	0.9988	0.5824	0.9993	0.5121	0.9993	0.7087
	GloVe	0.9983	0.6187	0.9976	0.66	0.999	0.6041	0.9987	0.7606
	TE 3 Small	1.0	0.7478	1.0	0.753	1.0	0.7435	1.0	0.8119
	RoBERTa	1.0	0.6657	1.0	0.7013	1.0	0.6495	1.0	0.7842
NAÏVE BAYES	BoW Unigrams	0.8663	0.6975	0.8853	0.7606	0.852	0.6655	0.8993	0.7957
	BoW Bigrams	0.9782	0.7009	0.976	0.7919	0.9807	0.66	0.9827	0.803
	TF-IDF Unigrams	0.5938	0.4843	0.8906	0.8237	0.5505	0.4669	0.7685	0.7203
	TF-IDF Bigrams	0.641	0.4675	**0.9106**	**0.8439**	0.5876	0.4539	0.7891	0.714
RANDOM FOREST	BoW Unigrams	0.7038	0.6834	0.7045	0.694	0.704	0.6753	0.7699	0.7643
	BoW Bigrams	**0.7081**	**0.6928**	0.717	0.7109	0.7009	0.6791	0.7796	0.7768
	TF-IDF Unigrams	0.7125	0.662	0.7087	0.6613	0.7166	0.6627	0.7756	0.7444
	TF-IDF Bigrams	0.7285	0.6803	0.7426	0.6857	0.7169	0.6755	0.7954	0.7611
	Word2Vec	0.9624	0.5177	0.95	0.5491	0.9764	0.5101	0.9675	0.6894
	GloVe	0.9942	0.5548	0.9917	0.7006	0.9968	0.5277	0.995	0.7402
	TE 3 Small	1.0	0.6611	1.0	0.8641	1.0	0.6006	1.0	0.7852
	RoBERTa	0.9947	0.6559	0.9921	0.7589	0.9974	0.6206	0.9953	0.7873
LOGISTIC REGRESSION	BoW Unigrams	0.733	0.6953	0.7157	0.6851	0.7575	0.7094	0.7865	0.7643
	BoW Bigrams	0.746	0.6994	0.731	0.6932	0.7663	0.7094	0.799	0.7695
	TF-IDF Unigrams	0.7009	0.6637	0.6934	0.6572	0.7096	0.6716	0.7683	0.7386
	TF-IDF Bigrams	0.6196	0.6049	0.6555	0.6164	0.5976	0.5963	0.732	0.7093
	Word2Vec	0.4404	0.4257	0.437	0.4261	0.4532	0.4459	0.597	0.5616
	GloVe	0.6413	0.6238	0.6239	0.6079	0.685	0.6599	0.6993	0.6888
	TE 3 Small	0.784	0.7677	0.7542	0.7396	0.8359	0.8162	0.8145	0.8004
	RoBERTa	0.7327	0.7215	0.7114	0.7083	0.7696	0.7474	0.7837	0.7816
XGBOOST	BoW Unigrams	0.918	0.7145	0.9423	0.747	0.8975	0.6914	0.9388	0.7988
	BoW Bigrams	0.7368	0.6723	0.8747	0.8175	0.6778	0.6195	0.8256	0.7873
	TF-IDF Unigrams	0.2619	0.262	0.2157	0.2158	0.3333	0.3333	0.6472	0.6475
	TF-IDF Bigrams	0.0876	0.0874	0.0504	0.0503	0.3333	0.3333	0.1512	0.1509
	Word2Vec	0.9682	0.5146	0.9851	0.6012	0.9529	0.5005	0.9748	0.7145
	GloVe	0.9978	0.6224	0.9982	0.7391	0.9974	0.5837	0.9983	0.7685
	TE 3 Small	1.0	0.7747	1.0	0.8514	1.0	0.7293	1.0	0.8402
	RoBERTa	1.0	0.6657	1.0	0.7013	1.0	0.6495	1.0	0.7842
LSTM	Word2Vec	0.4379	0.4393	0.7558	0.4242	0.4638	0.4653	0.7033	0.7035
	GloVe	**0.9336**	**0.7432**	0.9349	0.7617	0.9325	0.7288	0.9506	0.8151
DISTILBERT	-	0.9864	0.8216	0.9850	0.8246	0.9878	0.8188	0.9903	0.8649
TWITTER ROBERTA	-	0.9989	0.8052	0.9989	0.7965	0.9989	0.8155	0.9989	0.8481
GPT-4O	-	0.3355	0.3859	0.3980	0.4047	0.4291	0.4027	0.4432	0.3355

Table 11 - Hyperparameter Search Space for Transformer Encoders (DistilBERT and Twitter RoBERTa)

References

[Bib. 0] Galão, M., & Collaborators. (2024). Text Mining Project [GitHub repository]. GitHub. Link

[Bib. 1] Manning, C. D., Raghavan, P., & Schütze, H. (2008). Introduction to Information Retrieval. Cambridge University Press.

[Bib. 2] Twitter sentiment analysis: The good the bad and the OMG! In Proceedings of the Fifth International Conference on Weblogs and Social Media (ICWSM)

[Bib. 3] Zhang, Y., & Wallace, B. C. (2017). A Sensitivity Analysis of (and Practitioners' Guide to) Convolutional Neural Networks for Sentence Classification.

[Bib. 4] Pennington, J., Socher, R., & Manning, C. D. (2014). GloVe: Global Vectors for Word Representation. In Proceedings of the 2014 Conference on Empirical Methods in Natural Language Processing (EMNLP) (pp. 1532–1543). Association for Computational Linguistics.

[Bib. 5] OpenAI. (2023). Text embedding models. OpenAI API Documentation. Retrieved from https://platform.openai.com/docs/guides/embeddings

[Bib. 6] Barbieri, F., Camacho-Collados, J., Espinosa Anke, L., & Neves, L. (2020). "TweetEval: Unified Benchmark and Comparative Evaluation for Tweet Classification".

[Bib. 7] Liu, Y., Ott, M., Goyal, N., Du, J., Joshi, M., Chen, D., … & Stoyanov, V. (2019). RoBERTa: A Robustly Optimized BERT Pretraining Approach.

[Bib. 8] OpenAI. (2024). GPT-4o: The first Omni model. Retrieved from https://openai.com/index/hello-gpt-4o/

[Bib. 9] O'Reilly Media. (n.d.). Evaluating Machine Learning Models. Retrieved from https://www.oreilly.com/content/evaluating-machine-learning-models/

[Bib. 10] Xie, Q., Dai, Z., Hovy, E., Luong, M. T., & Le, Q. V. (2019). *Unsupervised Data Augmentation for Consistency Training*.

[Bib. 11] Lewis, P., Perez, E., Piktus, A., Petroni, F., Karpukhin, V., Goyal, N., … & Riedel, S. (2020). Retrieval-Augmented Generation for Knowledge-Intensive NLP Tasks. Advances in Neural Information Processing Systems (NeurIPS).

[Bib. 12] Baker, M., & Wurgler, J. (2007). Investor sentiment in the stock market. Journal of Economic Perspectives, 21(2), 129–151.