MDSAA

Mestrado em

Data Science and Advanced Analytics

Stock Sentiment
Predicting market behaviour from tweets

Text Mining

Group 11

Inês Major: 20240486
Luís Semedo: 20240852
Pedro Santos: 20240295
Rafael Bernardo: 20240510
Rodrigo Miranda: 20240490

NOVA Information Management School
Instituto Superior de Estatística e Gestão de Informação
Universidade Nova de Lisboa
June, 2025

ABSTRACT

This project explores the use of Natural Language Processing (NLP) techniques to classify financial tweets into three sentiment categories: Bullish, Bearish, and Neutral. Starting with exploratory data analysis, we identified key patterns and cleaned the text data to remove noise such as URLs, emojis, and irrelevant characters. Two preprocessing pipelines were developed: one for traditional machine learning models (e.g., KNN, SVM, CNN-BiLSTM) and another tailored to Transformer-based models (e.g., RoBERTa, DeBERTa). Model performance was evaluated using metrics such as accuracy and macro F1-score. In addition, a Streamlit app was created to deploy a simple chatbot, allowing both individual and batch sentiment predictions. This pipeline demonstrates how unstructured data from social media can be transformed into actionable insights for financial sentiment analysis.

Keywords: Financial Sentiment Analysis, NLP, Tweets, Text Classification, Transformers, Streamlit, Machine Learning, Deep Learning, RoBERTa, DeBERTa, Bullish, Bearish, Neutral.

TABLE OF CONTENTS

1. INTRODUCTION

The influence of social media on financial markets has become increasingly evident in recent years. Platforms like Twitter enable real-time dissemination of investor opinions, market rumors, and breaking news, offering a rich, albeit noisy, source of information. As such, this project aims to develop a Natural Language Processing (NLP) pipeline capable of classifying tweets into three distinct sentiment categories: Bearish, Bullish, and Neutral. This classification can be used as a proxy for market sentiment, providing valuable insights into investor behavior.

Our corpus, composed of over 9,500 labeled tweets for training and nearly 2,400 unlabeled tweets for testing, was most likely collected directly from Twitter. Consequently, the dataset is highly informal, filled with abbreviations, emojis, hashtags, and various forms of noise. Addressing these challenges required a thoughtful combination of exploratory data analysis (EDA), preprocessing, feature engineering, and model experimentation.

Throughout this report, we investigate a broad range of NLP techniques and model architectures, from traditional methods such as Bag-of-Words and TF-IDF, to more advanced neural architectures like *CNN-BiLSTM*, and Transformer-based models such as *RoBERTa* and *DeBERTa*. Furthermore, we developed a custom Streamlit application that integrates our final model with Azure OpenAI's GPT interface, offering both single and batch prediction functionalities.

This report is organized as follows: we begin with an in-depth EDA to understand the structure and patterns in the data, followed by a detailed explanation of our preprocessing strategies tailored to different model families. We then explore various feature engineering approaches and evaluate the performance of multiple classifiers. Lastly, we present the development of our Streamlit app and conclude with key takeaways and future improvements.

Our goal is not only to deliver a performant sentiment classifier but also to demonstrate the effectiveness of a well-designed NLP pipeline in transforming unstructured textual data into actionable financial insights.

2. EDA

The dataset used in this project comprises of both a train and test corpus with respectively 9543 tweets and 2387 for the latter. Our train corpus contains 3 different labels: Neutral, Bullish and Bearish, which is precisely the central point of this work, classify financial tweets and provide sentiment analysis.

2.1. TARGET IMBALANCE

Besides the usual descriptive analysis of the label feature, and checking if the dataset contains any missing values, the first key insight is crucial for the procedure along the rest of our work, target imbalance. From Figure 1, this analysis becomes evident, precisely 6178 of the instances are classified as "*Neutral*", around 65% of total classifications. Evidently, we can expect that most of our models when in doubt might tend to predict tweets as Neutral, furthermore it might prove useful to formulate hypothesis on how our preprocessing can enhance and bring out the nature of both "*Bearish*" and "*Bullish*" labels. The latter however are much more balanced with 1923 and 1442 instances respectively.

2.2. Stopword Analysis per Class

For this analysis, a temporary data frame was created, and the stopwords package from *nltk_data* was installed. The rationale was the following: what kind of words or patterns are relevant to each class? The results were proven to be quite interesting. The Bearish class (Figure 2) demonstrated prominence in words such as *"coronavirus"*, *"china"*, and *"misses"*, which could in fact reflect market concerns or instability. The Bullish class (Figure 3), on the other hand, highlights terms like *"beats"*, *"target"*, and *"revenue"*, suggesting positive sentiment and performance expectations. To better visualize these lexical differences, a graph was produced showing the most common words per class, after removing stopwords and less informative terms (Figure 5). This representation offers a clearer view of the vocabulary distribution across classes and reinforces the notion that word usage carries strong sentiment signals. Additionally, the frequent presence of non-alphanumeric characters (e.g., hyphens, colons, and hashtags) suggests potential sources of noise. Their removal during preprocessing may contribute to cleaner input data and improved downstream model performance.

We conclude this section with a succinct insight from the Tri-Gram Analysis. By simply visualizing 30 trigrams, we quickly observed the presence of the word "https" in most of the instances, indicating the presence of links, which semantically might not be helpful.

2.3. Character Patterns – Digits/Punctuation/URLs/Emojis

Through the use of REGEX patterns, we essentially wanted to comprehend how many of our tweets contain several characteristics or patterns such as digits, punctuation, URLs, mentions, hashtags, etc... For URLs, around 46.9% of our tweets contain them, which is quite abundant (specifically 4470 instances), in terms of mentions and hashtags the percentages are much lower, around 3% and 9.4% respectively. Furthermore, we have aggregated the following analysis: Digit, Punctuation, Upper Case and Emoji Counts. By computing the mean and aggregating per class we have managed to unravel interesting insights (Figure 6). Punctuation is naturally most prominent occurrence per class and groups, digits follow suit, where Upper case and emoji counts show much lower occurrence. Interestingly, digit count appears to be most prominent in Bullish class, perhaps case of some target stock prices or sharing gains.

In addition, the same analysis was extended to examine the frequency of URLs, user mentions, and hashtags across sentiment classes. The Bullish class exhibited notably fewer occurrences of these elements compared to the other classes (Figure 7). To further contextualize these patterns, a complementary variant of the analysis was performed, offering a more detailed view of the distribution of such features. Moreover, a binary feature was engineered to identify tweets containing potential sources of textual noise: such as symbols, emojis, punctuation marks, and other special characters. The outcome was conclusive: approximately 85% of the tweets include these noisy elements, a result consistent with the informal and unstructured nature of content typically found on Twitter (Figure 10).

2.4. TEXT ANALYSIS

This section explores additional insights derived from the dataset. An initial analysis of tweet length, in terms of both word and character count, revealed notable patterns (Figure 11). Most tweets contain approximately 10 to 15 words, with the word count distribution closely resembling a normal distribution. In contrast, the character count displays a pronounced spike around 130 characters, with over 1,000 instances, suggesting potential character limit effects typical of Twitter content. Descriptive statistics further revealed that some tweets consist of only a single word. Upon inspection, these tweets were found to belong exclusively to the "Neutral" class and largely comprised stock tickers or user mentions, offering little semantic value. As such, removing these instances may contribute to noise reduction. Figure 9 corroborates the presence of outliers in word count, reinforcing the potential benefit of minimal-length tweet filtering in preprocessing.

On the contrary, some models such as Transformers or others might benefit from this type of behavior. As we will in section 3, specific preprocessing will be elaborated for some attempts, whereas for most Transformer approaches they already benefit from internal preprocessing and are built on top of previous schematics and assumptions.

Delving deeper into the analysis of actual words contained in our tweets, the result is as expected. Without the elimination of stop words, these are the most frequent words in our corpus, without exception ("to", "the", "of", "-", etc...), (Figure 11).

2.5. MAIN TAKEAWAYS

Exploratory Data Analysis is essential for understanding the dataset, identifying potential challenges, and uncovering meaningful patterns. Given that the corpus likely originates from Twitter, it exhibits typical noise, such as special characters, URLs, and mentions. Replacing URLs and mentions with standardized tokens (e.g., "URL", "USER") may help retain structural context without semantic loss. Hashtags can often be reduced to the keywords they contain, while selective punctuation: such as "!" or "?", may carry sentiment cues, unlike more neutral symbols like periods or dashes.

3. PREPROCESSING

In light of the previous section, several important insights were drawn. Regarding preprocessing, it is important to acknowledge that multiple alternative strategies could have been explored. Nevertheless, existing literature highlights a variety of methods which, in most cases of classical sentiment analysis, such as in our case, are strongly recommended and widely adopted.

3.1. TRAIN/VALIDATION SPLIT:

The first step involved splitting the corpus into training and validation sets in order to evaluate model performance on an "unseen" subset. A standard 80/20 split was applied to resemble the structure of the "test.csv" file, which contains approximately 2,000 rows. Stratification was also ensured to preserve the class distribution across both sets, and a random state of 42 was used to guarantee reproducibility.

3.2. Cleaning & Preprocessing:

Preprocessing plays a pivotal role in sentiment analysis, particularly when dealing with noisy social media data such as tweets. However, the preprocessing strategy must be tailored to the nature of the model in use.

For this reason, we adopted **two distinct preprocessing pipelines**, depending on the family of models:

- **Classical models** such as KNN, SVM, and LSTM required more aggressive preprocessing, including stopword removal, lemmatization, and stemming, in line with standard NLP practices.
- **Transformer-based encoders** such as BERT, RoBERTa, and DeBERTa, on the other hand, benefit from their internal tokenization mechanisms and pretrained embeddings, which allow them to handle raw and noisy text more effectively. For these models, we opted for a lightweight preprocessing approach designed to preserve contextual richness while injecting weak supervision cues into the input.

Classical Models:

Our pipeline includes a function that wraps all the different preprocessing steps in order to apply it sequentially to both splits. The procedure starts of simply, with basic cleaning, most of the noise that was collected in section 2. Lowercasing the text is crucial to normalize words and improve generalization. We follow suit with the following procedure: The URLs were replaced with a <URL> mark, allowing the model to comprehend that this is a unique token, and the same procedure was used for the Mentions, replaced with <USER>. Essentially, we are looking to reduce dimensionality, reduce noise and improve generalization, and this is once again standard practice across the industry especially in cases of sentiment analysis projects scraped from data with Twitter.

Furthermore, we initially suggested that selecting punctuation could be interesting to maintain sentiment in “!” or “?”. Empirical tests on a baseline performance run indicated no apparent significant differences, hence, we will proceed with removing punctuation altogether. Finally, we remove the “#” symbol from our text, keeping only the words after it.

Second step of the preprocessing pipeline is tokenization, a mandatory stage that splits words into so called “tokens”, initial tests were developed on a baseline run using the classical *nltk.word_tokenize()* method, which demonstrated superior performance with our pipeline versus the *TweetTokenizer* package, designed for social media text. Third step removes stop words from our analysis, and step four and five lemmatizes and stemmatizes the text, a standard procedure in the space. We apply this procedure to our splits and we are essentially ready to experiment with different feature engineering and model combinations.

Encoders:

Unlike traditional models that rely heavily on extensive text cleaning and normalization, transformer-based architectures such as *BERT*, *RoBERTa*, or *DeBERTa* benefit from rich pre-trained embeddings and subword tokenization strategies, which make them particularly robust to the informal and noisy language commonly found on social media. Consequently, the preprocessing applied to encoder models in our pipeline was deliberately minimal, with a focus on preserving potentially valuable contextual information while injecting weakly supervised cues directly into the text.

The cleaning process primarily involved replacing URLs and redundant spacing using simple regular expressions. URLs were substituted with a generic [URL] token, enabling the model to learn that such tokens refer to external links while maintaining the original sentence structure. No further steps such as punctuation removal or lowercasing were applied, as these are internally handled by the tokenizer of each transformer model.

In addition to this minimal preprocessing, we extracted five binary features for each tweet, indicating the presence of specific elements: URLs, mentions (e.g., @user), hashtags (e.g., #market), hashtags (e.g., $AAPL), and emojis. Rather than treating these as separate features, we opted to embed them back into the text using natural language cues. For example, a tweet containing both a hashtag and a URL would be enriched as: “This tweet contains a hashtag. This tweet contains a URL. $AAPL is down today. [URL]”. This strategy aims to support the model in associating structural tweet patterns with specific sentiments, which is particularly useful in short texts where explicit sentiment words may be sparse.

The final output of this process was the creation of the *bert_text_enriched* column, which served as the consistent input for all transformer-based experiments. No stemming, lemmatization, or stopword removal was performed, as these techniques are generally counterproductive in transformer-based models due to their reliance on intact token sequences and contextual embeddings.

4. FEATURE ENGINEERING & MODEL EVALUATION (CLASSICAL MODELS)

4.1. BoW & KNN

This configuration is perhaps the most straight forward and simple approach to the problem at hand. Widely recognized technique and perhaps one of the oldest and out-of-date as well is the *Bag-of-Words* as a feature engineering method. Paired up with a *KNN* model, which is a simple but effective model for some tasks, we managed to develop a solid baseline for the task. The results are not astonishing at all, yet for different values of k, we managed to find that k=13 maximizes accuracy score (Figure 12). Given the simplicity of the approach, we developed a *GridSearch* and discovered the best parameters, which provided quite similar results.

In addition, we also simulated the same configurations but using *TF-IDF* instead of *BoW* paired up with the same *KNN* parameters. Furthermore, k=13 appeared to also maximize accuracy, and we saw here a boost in 2% in terms of accuracy score (78%), and F1-Macro scores (68%) Figure 13.

4.2. TF-IDF & SVM

In light of the above, given TF-IDF's boost in results, we decided to take a quick peek and see how this feature engineering method performed when paired with perhaps a more robust model. For this section, given the computational resources available, a comprehensive *GridSearch* was developed in order to maximize performance, but quickly we realized that the results were deceptive, the model was quickly memorizing completely the training set.

As for the *SVM* tuning the "*C*" parameter was crucial to add regularization, additionally, setting *class_weights* as "balanced" helped in achieving much better generalization. Furthermore, as for the *TF-IDF*, containing *max_features* to 2000 as opposed to higher values, was a quick and easy fix to combat overfitting. Ultimately, we managed to be withing the 5-6% differential in terms of training and holdout set which is considered a good rule of thumb, and F1-macro achieves a solid 72% on the holdout set as well as 80% accuracy as opposed to 81% and 86% respectively on the train set (Figure 15). This serves as our new baseline and an important remark on the importance of checking and regularizing overfitting.

4.3. WORD2VEC & CNN + BILSTM

With the aim of applying some of the concept lectures in class, we also took inspiration from (Xuemei, Xiaoyan, & Raga, 2022), which applied to their sentiment classification model from online comments of COVID-19 a GloVe-CNN-BiLSTM approach, gathering quite fascinating results of around 95% accuracy, sharing some of the parameters and architecture applied. According to the paper there are several advantages to this approach as such "*It can not only take advantage of CNN to extract local features but also take advantage of BiLSTM to consider the global features of text sequence*".

With the aid of this paper, we developed a similar approach, first things first we defined *Word2Vec* with a *vector_size* of 200, setting it to skip-gram to better capture semantics, a window of 3 usually better suited for tweets where context is more local. Preprocessing, tokenization and padding was applied, and training architecture consists of a block of 2 *Conv1D* layers, interconnected with a *BatchNormalization* layer, and a *GlobalMaxPooling* layer to flatten outputs. Secondly, a BiLSTM block was constructed with 2 Bidirectional layers of LSTMs, each with a *dropout rate* of 0.15. Both these blocks are merged and followed by *Dropout layer* of 0.45, and a *Dense* layer suited to our necessities with a *SoftMax* activation function. For training, *Adam* optimizer was used, and we set learning rate to 1e-5. In this case, for simplicity, accuracy was the only metric tracked just to check the possible performance of the model. An important note is that not exactly the same parameters were used as in the paper, and based on several runs, filters and neurons as well as dropout and learning rates were tweaked iteratively to maximize generalization. Moreover, generalization was successfully achieved with around 73% accuracy across sets (Figure 16), at the end of 45 epochs.

4.3.1. GLOVE & CNN + BILSTM

Similarly to the previous section, we decided to implement a variation using *GloVe*. The advantage of *GloVe* is that unlike *Word2Vec*, it does not rely solely on local statistics like the local context information of words but incorporates global statistics to obtain word vectors. Likewise, the source this architecture was inspired by also used *GloVe*, and results were promising. With pretty much a similar architecture except for a *Dropout* Rate set to 0.6, around 77% accuracy was achieved across sets during 45 epochs (Figure 17).

As future sections regarding *Transformers* provide robust results, we did not dedicate extensive time to this section, either in terms of boosting performance or in terms of tracking relevant metrics such as f1_macro, precision and recall.

5. FEATURE ENGINEERING TRANSFORMER-BASED ENCODERS

In Transformer-based approaches, feature engineering is inherently embedded within the model architecture itself.

However, a crucial preprocessing step remains necessary: formatting the input text in a way that is compatible with these architectures. To address this, we implemented a function called *prepare_hf_datasets*, specifically designed to tokenize and structure the input data for encoder-only transformer models such as *RoBERTa* or *DeBERTa*. Each split of the dataset (training and validation) is converted into a *HuggingFace* Dataset object, where the column containing the enriched text (bert_text_enriched) is renamed to the standard "text" label expected by most tokenizers. The core of the function lies in the *tokenize_function*, which applies the selected model's tokenizer while ensuring truncation and padding are correctly handled.

Although we experimented with different values for the *max_length* parameter, including 256 and 512 tokens, the performance gains were marginal and did not justify the significantly increased computational cost. Therefore, we chose to retain a maximum sequence length of 128 tokens, balancing efficiency and performance.

6. CLASSIFICATION MODELS TRANSFORMER-BASED ENCODERS

Encoders:

For all Transformer-based models tested in this study, namely *RoBERTa-Base*, *RoBERTa-Large*, *DeBERTa-v3-base*, *DeBERTa-v3-large*, *BERTweet*, *MPNet-base*, and *ALBERT-base-v2*, we used a unified training function to streamline the fine-tuning process. This function, *train_encoder_model*, receives the dataset and the Hugging Face model checkpoint as input, and performs the full training cycle using the Trainer class from the transformers library.

Within this function, the appropriate tokenizer is loaded for the selected checkpoint, with the *use_fast* option set to False for *BERTweet* due to tokenizer compatibility issues. The dataset is then tokenized using the *prepare_hf_datasets* function, and a sequence classification model is instantiated with *num_labels*=3, corresponding to the three sentiment classes in our task.

Training arguments include early stopping with patience of 3 epochs, evaluation and saving at each epoch, a learning rate of 2e-5, and a batch size of 32. Importantly, the best model is automatically restored at the end of training based on the macro F1-score on the validation set. This function ensures consistency across all encoder models and was used to train each of the models listed in the interface shown.

Given their superior performance in the initial training runs*, RoBERTa-Large* and *DeBERTa-v3-Large* were selected for hyperparameter optimization. To this end, we employed *Optuna*, a hyperparameter optimization framework that integrates seamlessly with Hugging Face's Trainer. Unlike traditional grid search, which exhaustively tests all parameter combinations from a predefined list, *Optuna* uses a smarter, sampling-based approach that dynamically selects the most promising regions of the hyperparameter space based on previous trial results. This enables a more efficient search, particularly when dealing with expensive models like *RoBERTa-Large* and *DeBERTa-Large*.

For each model, we defined a search space including the learning rate (sampled on a log scale), weight decay, and batch size. The objective metric was the macro F1-score on the validation set, and early stopping was used to prevent overfitting and reduce computational cost. Each trial involved initializing a fresh model, training it for a maximum of 10 epochs, and evaluating it after every epoch.

Lastly, after identifying *RoBERTa-Large* and *DeBERTa-v3-Large* as the two best-performing individual models, we also built a simple ensemble by averaging their predicted probabilities. This ensemble aimed to leverage the complementary strengths of both models and further improve overall classification performance. However, for the final prediction on the test set, this ensemble was not used, as our interpretation of the project guidelines is that the final pipeline should include a single classification model only.

7. EVALUATION TRANSFORMER-BASED ENCODERS

Encoders

The performance results for all tested Transformer-based encoders are presented in Table 1, where we report a comprehensive set of metrics including Accuracy, Precision, Recall, and F1-Score (both under macro and weighted averaging strategies). While our primary optimization objective was to maximize the macro F1-score, given the presence of class imbalance in the dataset, other metrics were also considered to ensure overall robustness and consistency of the models.

Among all evaluated models, *RoBERTa-Large* and *DeBERTa-v3-Large* consistently achieved the highest scores across most evaluation criteria and both models benefited from dedicated hyperparameter tuning using *Optuna*, as described previously. Although *DeBERTa-v3-Large* slightly outperformed *RoBERTa-Large* in some isolated runs, we observed that results were not entirely stable across executions and that minor fluctuations were common, likely due to random initialization and training stochasticity. Therefore, the final model choice was somewhat subjective, based not only on metric comparisons but also on training stability, reproducibility, and integration within the pipeline.

As a result, we selected *RoBERTa-Large* as the final model for test set prediction. On the validation set, *RoBERTa-Large* achieved an overall accuracy of 91%, with a macro F1-score of 0.89 and a weighted F1-score of 0.91. Class-wise, the model performed particularly well on the dominant class Neutral (F1-score of 0.94), while still maintaining strong results on Bullish (0.89) and Bearish (0.85).

8. EXTRA WORK

8.1 APP (CHATBOT)

Link for repository for this app: https://github.com/inesnmajor/tmapp2

Link for the Streamlit app: https://textminingproject.streamlit.app/

To complement our project, we built a lightweight web application using Streamlit to streamline both exploratory testing and systematic evaluation of our stock-sentiment classifier.

Upon launch, the app reads its configuration: Azure OpenAI endpoint, API key, deployment name and API version from a local .env file, ensuring that sensitive credentials never appear in the code.

The user interface consists of two tabs. The first tab, "Single Classification," presents a simple text area where analysts can paste in an individual market comment or tweet. Behind the scenes, before calling the model, the application injects a concise "system" prompt instructing the ChatGPT deployment to respond with exactly one of three labels: Bullish, Bearish, or Neutral, without any additional explanation. The model's single-word reply is then displayed prominently, allowing for rapid, ad-hoc sanity checks.

The second tab, "Batch Evaluation," enables end-to-end performance analysis on larger datasets. Users upload a CSV file containing two columns: text (the market statement) and label (an integer code for the true sentiment). The app maps these integer codes to their corresponding string labels, then iterates through each row, invoking the same system-prompted model call for each example. If any input triggers Azure's content-policy filter (for violence or other disallowed content), that row is silently skipped and counted, preventing a single problematic example from aborting the entire run.

Once all valid predictions are collected, the app computes standard classification metrics: accuracy, per-class precision, recall and F1 then generates a confusion matrix. The results are rendered in-page using a compact Matplotlib plot for the confusion matrix and an interactive Pandas DataFrame for the detailed classification report(Figure 18 and Figure 19).

All inference in the app is performed via Azure OpenAI's Chat Completion API, leveraging our "ChatGPT" deployment (backed by GPT-3.5-Turbo). In our initial zero-shot experiments we achieved roughly 69% accuracy (Table 2). By fine-tuning the sampling parameters: lowering the temperature to 0.2 and setting *top_p* to 0.9, we were able to boost overall accuracy to 71.04% (Table 3). We did not explore additional parameter sweeps or alternative model variants, the primary goal of this project was to prototype the Streamlit interface and demonstrate a novel integration of Azure OpenAI for real-time sentiment analysis.

Finally, the user can download a consolidated CSV report that begins with the number of filtered (skipped) texts, followed by the full classification-report table and the confusion-matrix table. This design provides a seamless workflow for both quick tests and large-scale model validation, all within a single, user-friendly interface.

8.2 ENCODERS

Firstly, we tested seven distinct encoder-only architectures, including RoBERTa (base and large), DeBERTa-v3 (base and large), BERTweet, MPNet-base, and ALBERT-base-v2, allowing us to explore a broad range of language representations and training regimes.

To further optimize performance, we employed Optuna for hyperparameter tuning of the two best-performing models: RoBERTa-Large and DeBERTa-v3-Large. This involved searching over learning rate, batch size, weight decay and optimizer parameters across almost 100 trials, significantly improving performance beyond default configurations. The integration with Hugging Face's Trainer and the definition of a custom F1-macro objective function enabled a more efficient and targeted search strategy compared to traditional grid search.

Additionally, we implemented a simple but effective ensemble model by averaging the predicted probabilities of RoBERTa-Large and DeBERTa-Large. While this ensemble was not used in the final submission due to project constraints requiring a single model, it served as a validation of the complementary nature of both models and their consistent high performance.

9. CONCLUSION

This project set out to tackle the complex task of inferring market sentiment from noisy and unstructured Twitter data using a variety of Natural Language Processing techniques. Throughout the process, we explored multiple paths: from classical machine learning with handcrafted features, to deep learning architectures leveraging pre-trained language models, demonstrating a clear evolution in performance and sophistication.

Our results underscore the importance of thoughtful preprocessing, tailored feature engineering, and appropriate model selection when working with informal text data. While early approaches such as BoW + KNN or TF-IDF + SVM provided us with solid baselines, it was the Transformer-based models, particularly RoBERTa-Large, that delivered state-of-the-art results, achieving both high accuracy and balanced class-wise performance. This confirms the growing dominance of pretrained language models in sentiment classification tasks, especially when dealing with short, context-sensitive texts like tweets.

Beyond model development, we also designed a practical Streamlit application that enables real-time inference through Azure OpenAI, bridging the gap between research and deployment. This application not only validates our pipeline but also showcases the feasibility of integrating advanced NLP models into user-friendly tools for analysts and decision-makers.

In sum, this project reinforced our understanding of the entire text mining pipeline, from exploratory analysis to deployment and highlighted the value of combining statistical rigor with creative engineering. Future work could explore few-shot learning with GPT-style models or apply domain adaptation techniques to further tailor transformer encoders to financial discourse. Nonetheless, our current pipeline represents a robust and deployable solution for stock sentiment analysis grounded in real-world social media data.

10. BIBLIOGRAPHY

Xuemei, S., Xiaoyan, L., & Raga, R. C. (22 de 10 de 2022). GloVe-CNN-BiLSTM Model for Sentiment Analysis on Text Reviews.

11. APPENDIX – EXPLORATORY DATA ANALYSIS

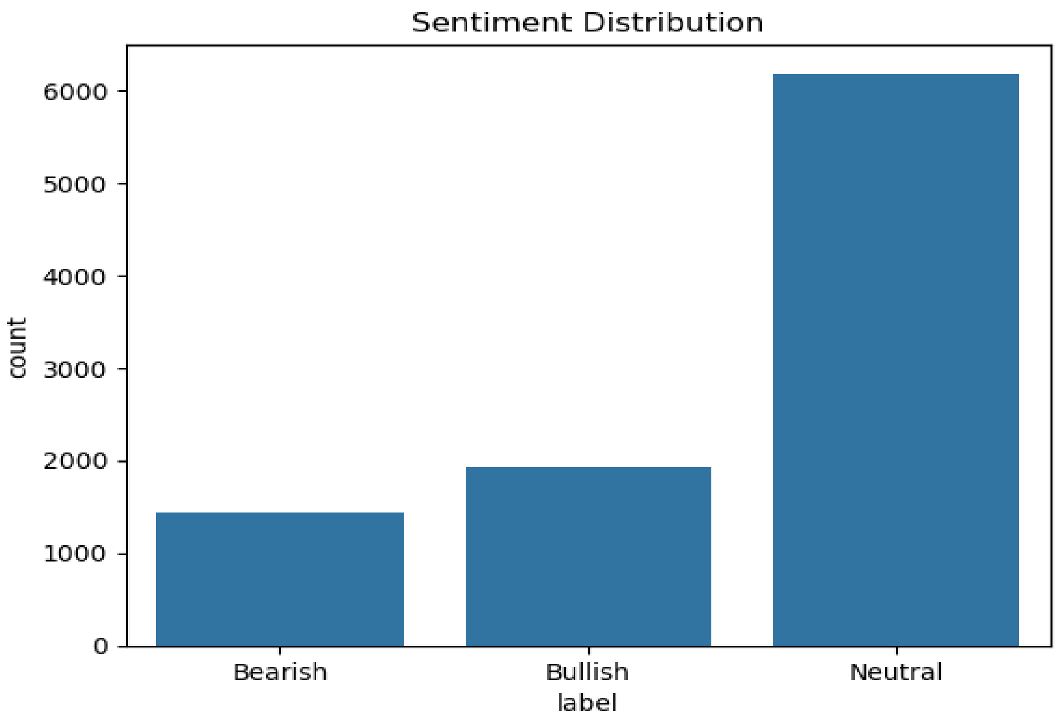
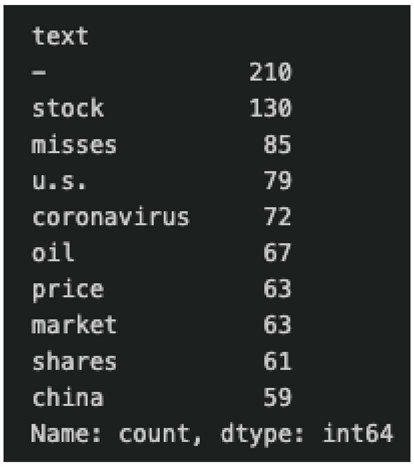
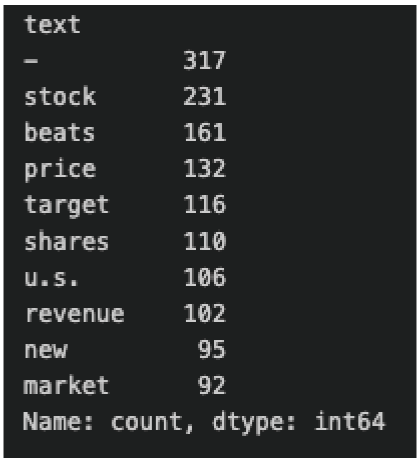


Figure 1 - Target Imbalance Analysis



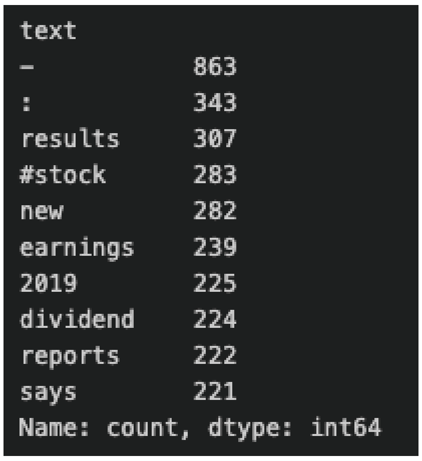
```
text
–                 210
stock             130
misses             85
u.s.               79
coronavirus        72
oil                67
price              63
market             63
shares             61
china              59
Name: count, dtype: int64
```

Figure 2 - Prominent Words Excluding Stopwords – Bearish Class



```
text
-          317
stock      231
beats      161
price      132
target     116
shares     110
u.s.       106
revenue    102
new         95
market      92
Name: count, dtype: int64
```

Figure 3 - Prominent Words Excluding Stopwords – Bullish Class



```
text
-           863
:           343
results     307
#stock      283
new         282
earnings    239
2019        225
dividend    224
reports     222
says        221
Name: count, dtype: int64
```

Figure 4 - Prominent Words Excluding Stopwords – Neutral Class

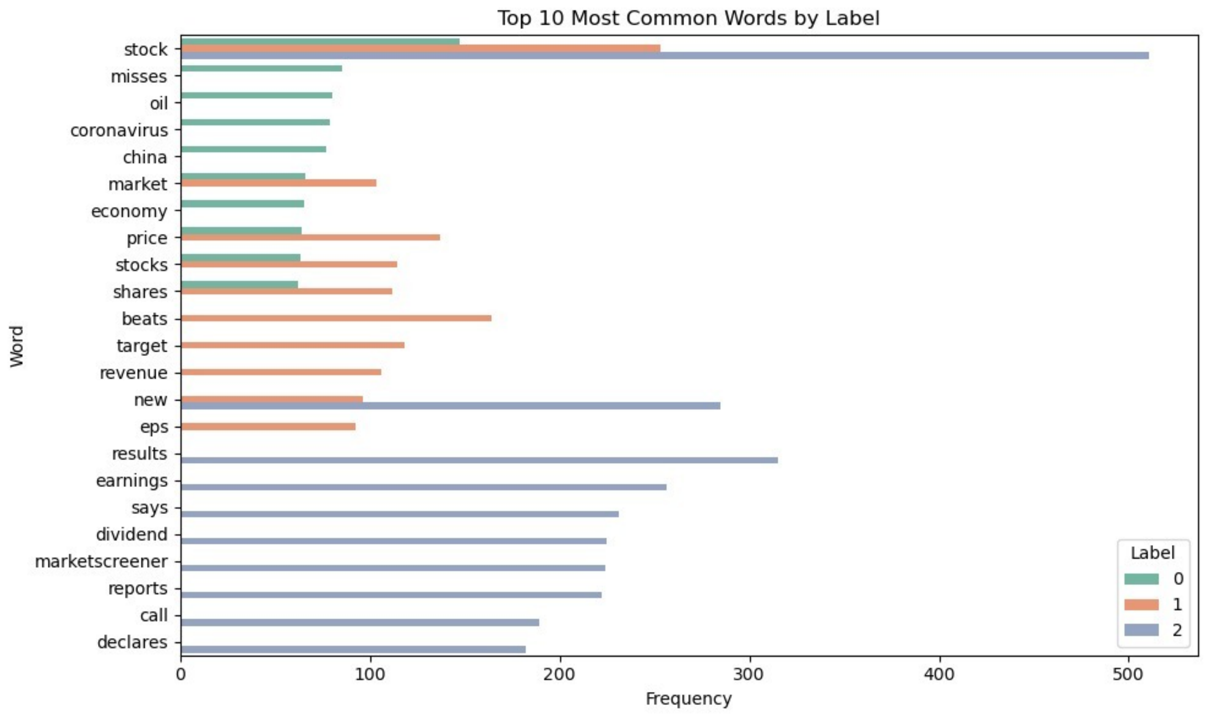
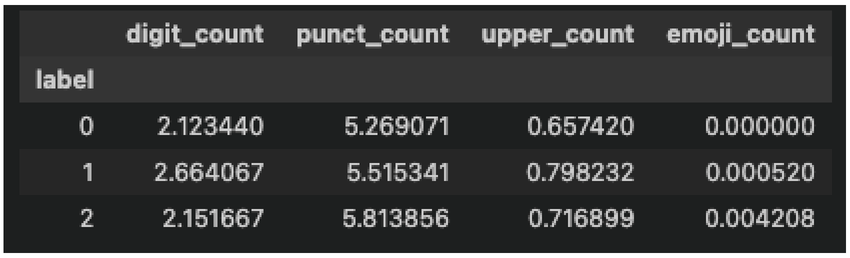
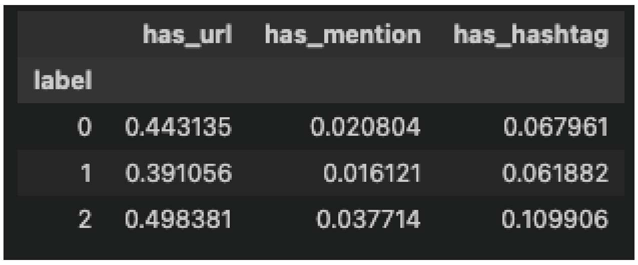


Figure 5 - Top 10 most common Words by Label



	digit_count	punct_count	upper_count	emoji_count
label				
0	2.123440	5.269071	0.657420	0.000000
1	2.664067	5.515341	0.798232	0.000520
2	2.151667	5.813856	0.716899	0.004208

Figure 6 - Group Aggregation per Class – Digits/Punctuation/Uppercase/Emoji Count



	has_url	has_mention	has_hashtag
label			
0	0.443135	0.020804	0.067961
1	0.391056	0.016121	0.061882
2	0.498381	0.037714	0.109906

Figure 7 - Group Aggregation per Class – Digits/Punctuation/Uppercase/Emoji Count

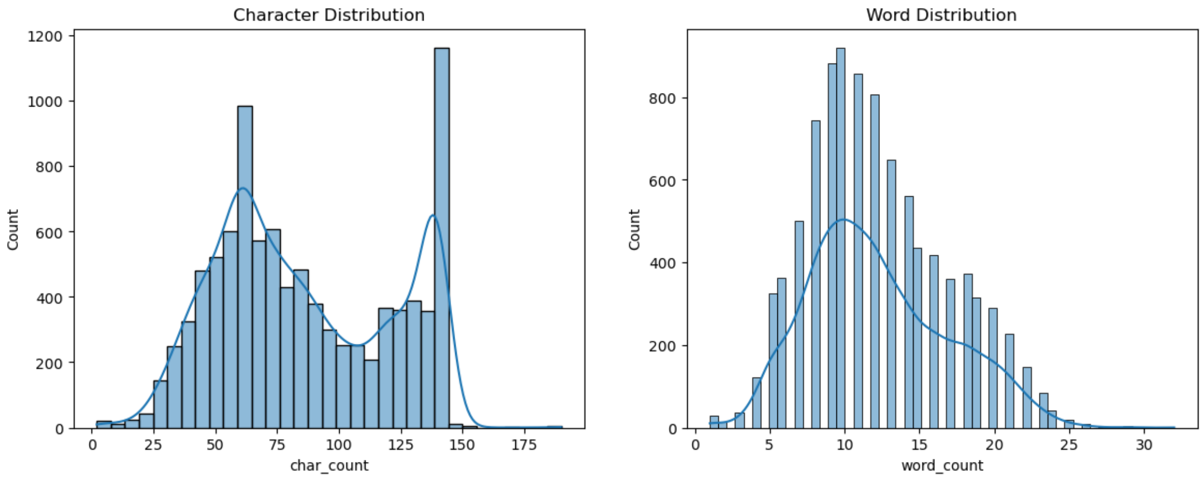


Figure 8 - Character & Word Count - Distributions

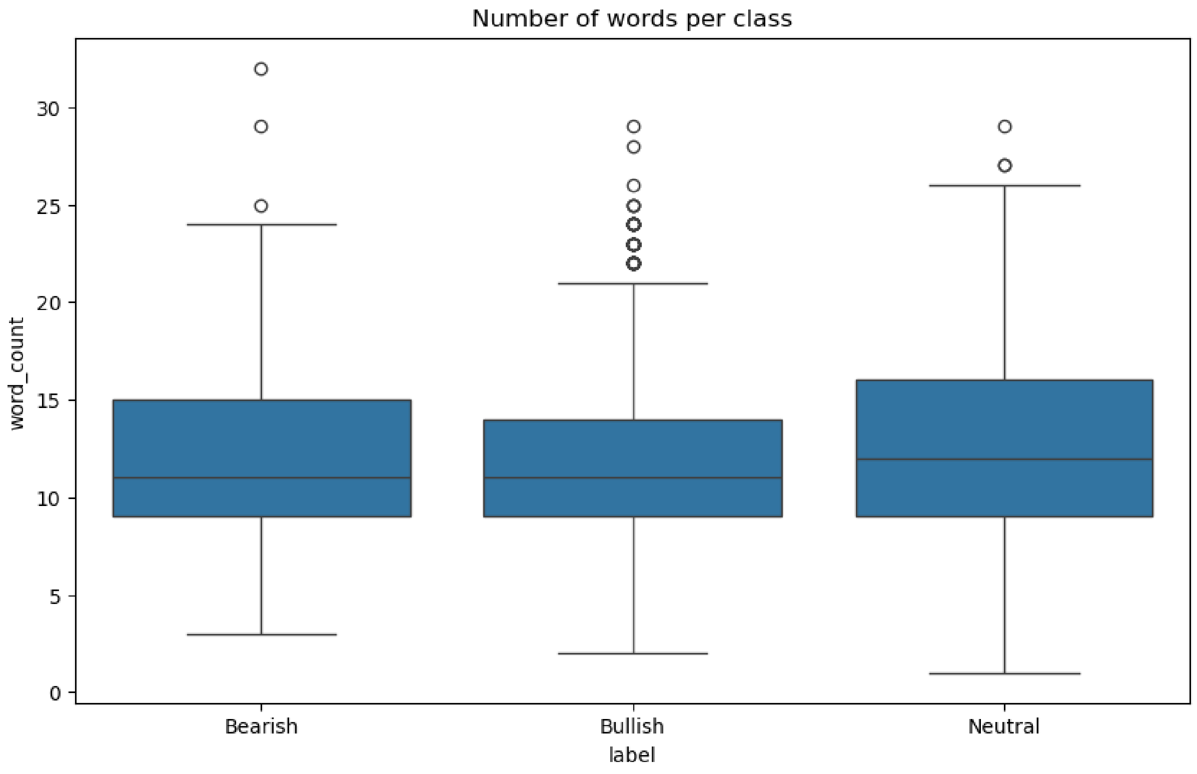
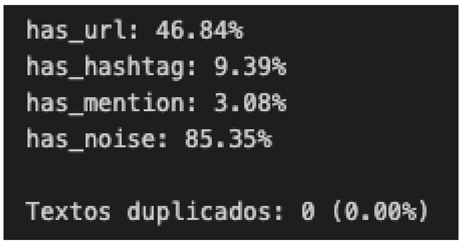


Figure 9 - Boxplot of Word Count per Class



```
has_url: 46.84%
has_hashtag: 9.39%
has_mention: 3.08%
has_noise: 85.35%

Textos duplicados: 0 (0.00%)
```

Figure 10 - Statistics and Special Characters Checks

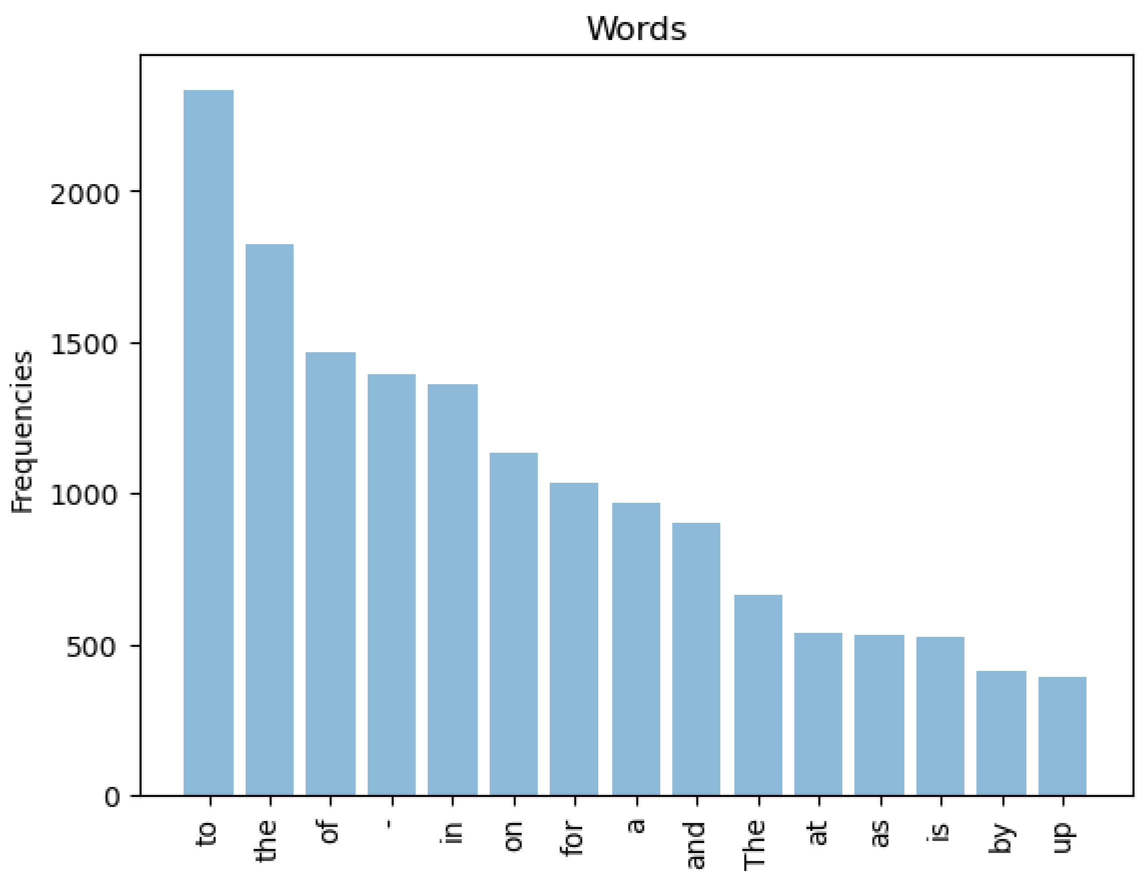


Figure 11 - Most Frequent Words in Corpus

12. APPENDIX – FEATURE ENGINEERING & MODEL EVALUATION (CLASSICAL MODELS)

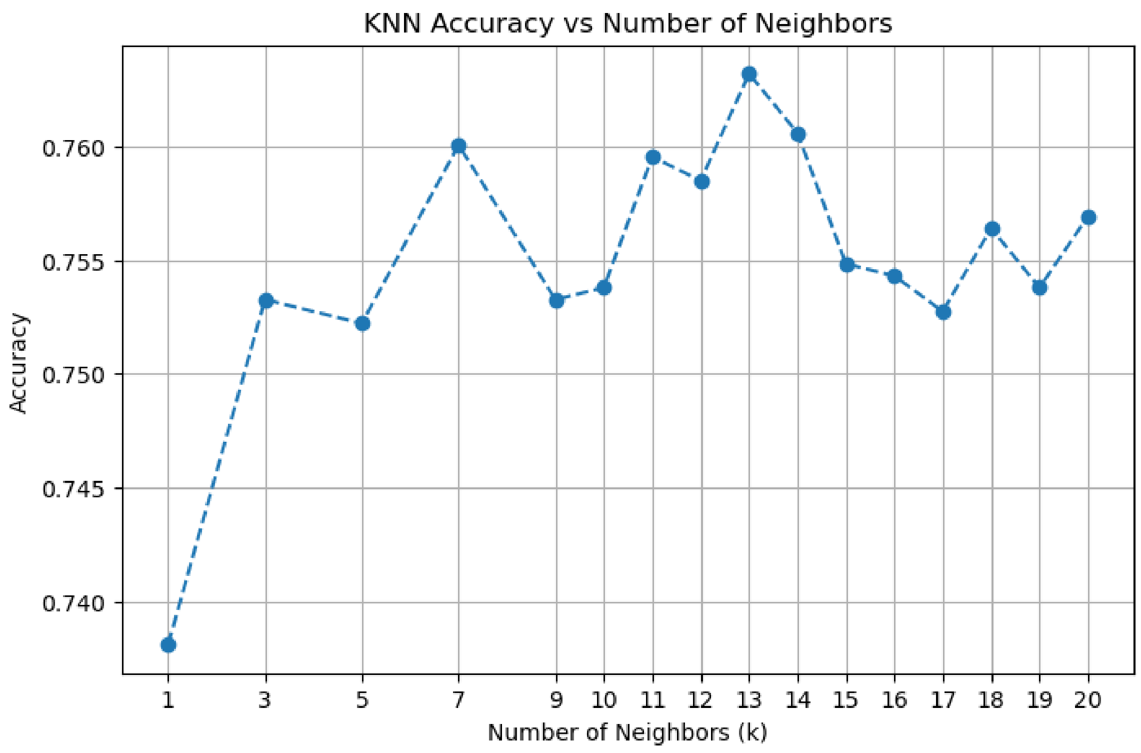


Figure 12 – KNN vs Number of Neighbors for BoW & KNN

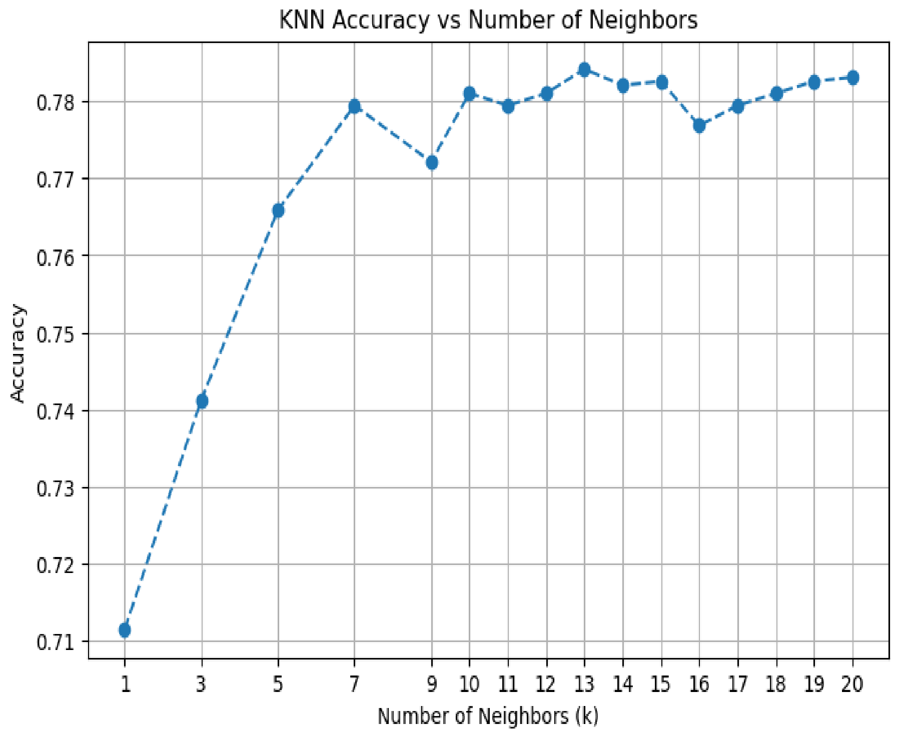
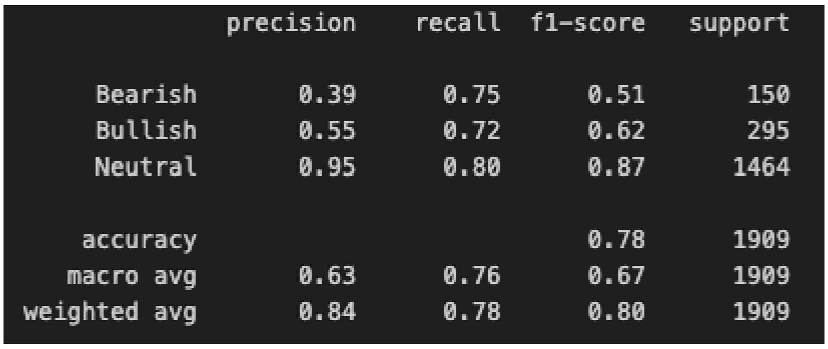


Figure 13 - KNN vs Number of Neighbors for TF-IDF & KNN

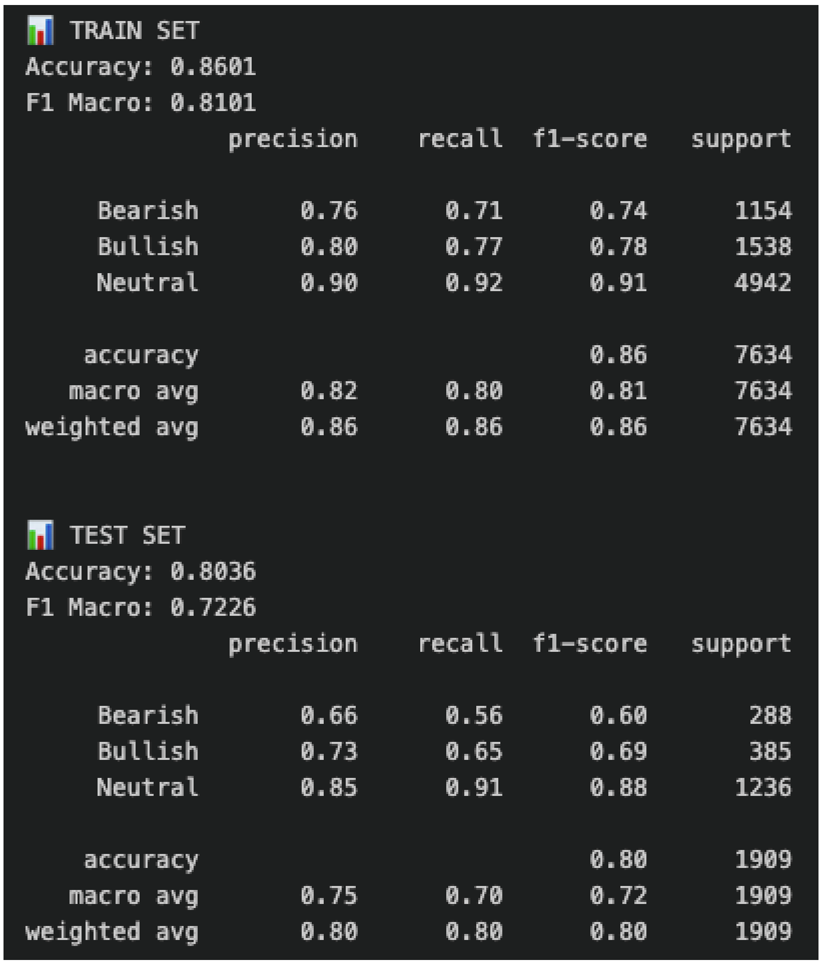


```
              precision    recall  f1-score   support

     Bearish       0.39      0.75      0.51       150
     Bullish       0.55      0.72      0.62       295
     Neutral       0.95      0.80      0.87      1464

    accuracy                           0.78      1909
   macro avg       0.63      0.76      0.67      1909
weighted avg       0.84      0.78      0.80      1909
```

Figure 14 - Performance of TF-IDF & KNN Configuration



```
📊 TRAIN SET
Accuracy: 0.8601
F1 Macro: 0.8101
              precision    recall  f1-score   support

     Bearish       0.76      0.71      0.74      1154
     Bullish       0.80      0.77      0.78      1538
     Neutral       0.90      0.92      0.91      4942

    accuracy                           0.86      7634
   macro avg       0.82      0.80      0.81      7634
weighted avg       0.86      0.86      0.86      7634


📊 TEST SET
Accuracy: 0.8036
F1 Macro: 0.7226
              precision    recall  f1-score   support

     Bearish       0.66      0.56      0.60       288
     Bullish       0.73      0.65      0.69       385
     Neutral       0.85      0.91      0.88      1236

    accuracy                           0.80      1909
   macro avg       0.75      0.70      0.72      1909
weighted avg       0.80      0.80      0.80      1909
```

Figure 15 - TF-IDF & SVM Performance Across Sets

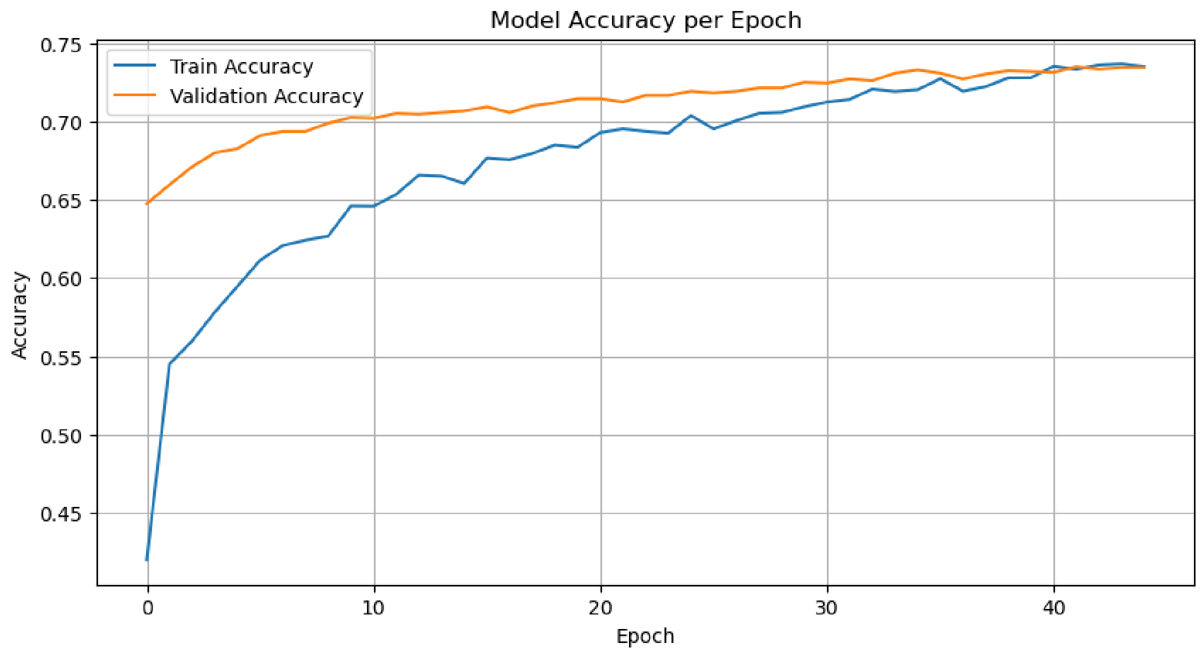


Figure 16 - Word2Vec & CNN + BiLSTM Performance

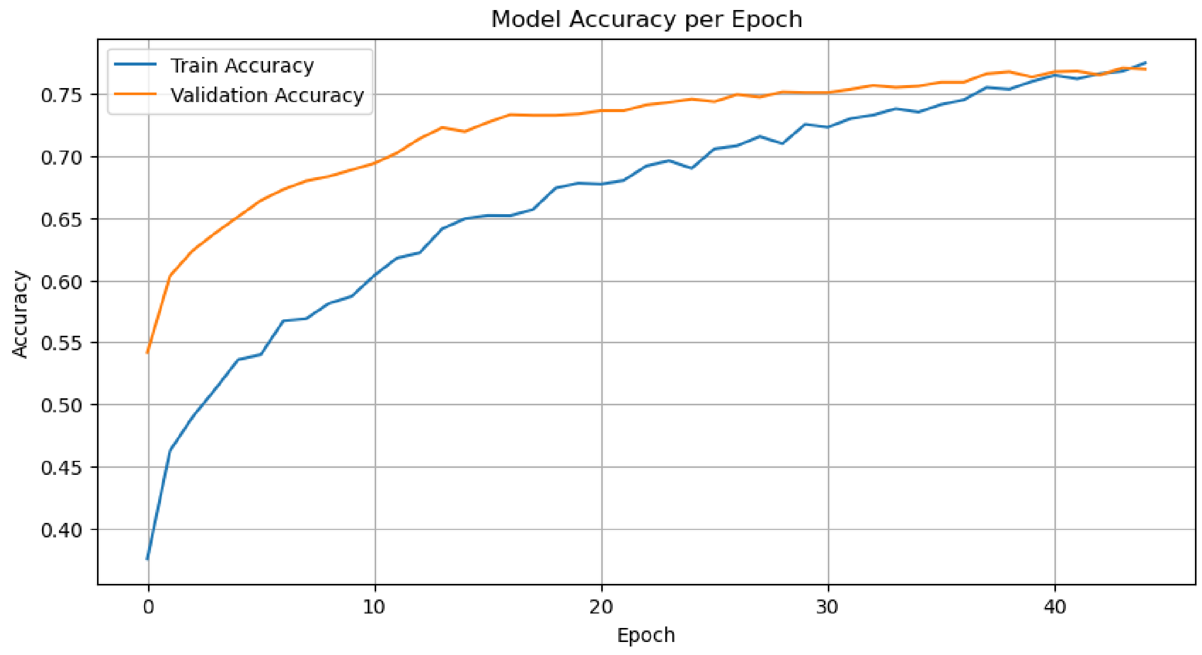


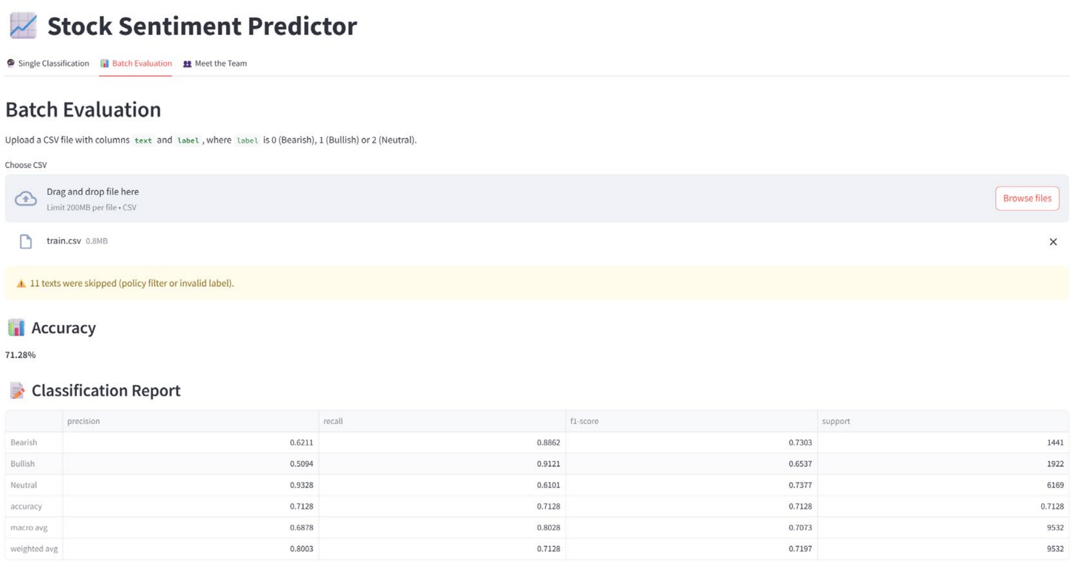
Figure 17 - GloVe + CNN + BiLSTM Performance

13. APPENDIX – EVALUATION TRANSFORMERS

Model	Accuracy	Precision Macro	Recall Macro	F1 Macro	Precision Weighted	Recall Weighted	F1 Weighted
RoBERTa-Base	0.9067	0.8748	0.892	0.8831	0.908341	0.9068	0.9073
RoBERTa-Large	0.9083	0.8978	0.87	0.8831	0.9079	0.9083	0.9076
Grid Search RoBERTa-Large	0.9089	0.8822	0.8857	0.8883	0.9092	0.9089	0.909
DeBERTa-v3-Base	0.8999	0.8715	0.8784	0.8744	0.9013	0.8999	0.9
DeBERTa-v3-Large	0.9109	0.8876	0.8818	0.8844	0.9106	0.911	0.9106
Grid Search DeBERTa-v3-Large	0.9115	0.8962	0.8907	0.8928	0.9157	0.9151	0.9151
BERTweet	0.8816	0.8409	0.8582	0.8547	0.8843	0.8816	0.8826
MPNet-Base	0.8952	0.8689	0.8749	0.8692	0.8996	0.8952	0.8958
ALBERT-Base-v2	0.8758	0.8409	0.8449	0.8428	0.8762	0.8758	0.876
Ensemble	0.92	0.9	0.88	0.89	0.92	0.92	0.92

Table 1 - Results of the Transformer-Based Encoders

14. APPENDIX – APP



Stock Sentiment Predictor

Batch Evaluation

Upload a CSV file with columns `text` and `label` , where `label` is 0 (Bearish), 1 (Bullish) or 2 (Neutral).

Choose CSV

train.csv 0.8MB

⚠ 11 texts were skipped (policy filter or invalid label).

Accuracy

71.28%

Classification Report

	precision	recall	f1-score	support
Bearish	0.6211	0.8862	0.7303	1441
Bullish	0.5094	0.9121	0.6537	1922
Neutral	0.9328	0.6101	0.7377	6169
accuracy	0.7128	0.7128	0.7128	0.7128
macro avg	0.6878	0.8028	0.7073	9532
weighted avg	0.8003	0.7128	0.7197	9532

Figure 18 - Example Run for Train.csv

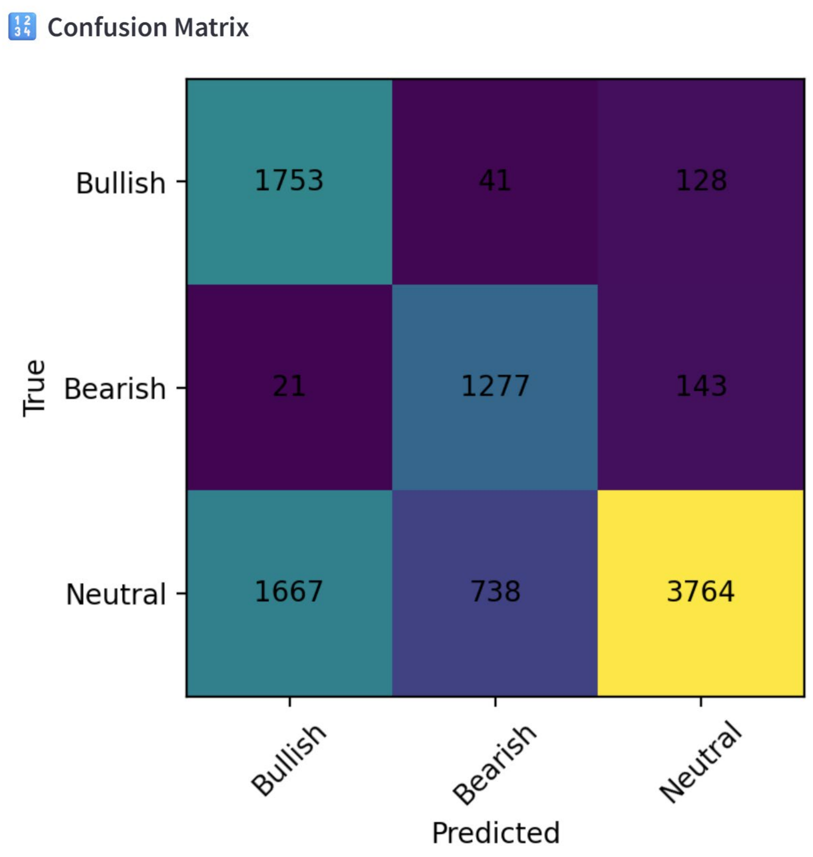


Figure 19 - Confusion Matrix for Test.csv

Skipped texts: 11				
Classification Report				
	Precision	Recall	F1-Score	Support
Bearish	0.5998	0.8632	0.7078	1441.0
Bullish	0.4982	0.9011	0.6417	1922.0
Neutral	0.9190	0.5924	0.7204	6169.0
accuracy	0.6956			
macro avg	0.3361	0.3928	0.3450	9532.0
weighted avg	0.7859	0.6956	0.7026	9532.0

Table 2 - Classification Report for first try

Skipped texts: 11				
Classification Report				
	Precision	Recall	F1-Score	Support
Bearish	0.6149	0.8820	0.7246	1441.0
Bullish	0.5106	0.9136	0.6551	1922.0
Neutral	0.9302	0.6070	0.7346	6169.0
accuracy	0.7104			
macro avg	0.6852	0.8009	0.7048	9532.0
weighted avg	0.7979	0.7104	0.7171	9532.0

Table 3 - Classification Report for second try