NOVA IMS Information Management School

TEXT MINING

SENTIMENT ANALYSIS

SPRING SEMESTER 24/25

Group 34

Diogo Duarte, 20240525
Philippe Dutranoit, 20240518
Rodrigo Sardinha, 20211627
Rui Luz, 20211628

Table of Contents

GitHub Repository: https://github.com/philippedut/Text_mining

1. Introduction

In an era where financial markets are increasingly influenced by online discourse, especially on social media platforms like Twitter, the ability to assess public sentiment in real time has become a valuable tool for investors and analysts. Market sentiment, the overall attitude of investors toward a particular market or asset, is often categorized as Bullish, expecting price increases, Bearish, expecting price decreases, or neutral. These sentiment shifts are not only reflected in financial indicators but also in textual signals across online platforms.

This project addresses the challenge of predicting stock market sentiment from tweets, using Natural Language Processing (NLP) techniques. The goal is to develop and evaluate a machine learning model capable of classifying individual tweets into one of three sentiment categories: Bearish (0), Bullish (1), or Neutral (2). This model should learn from labeled tweet data and apply that knowledge to new, unseen tweets. Our aim is to build a solution that not only performs well but is also interpretable and grounded in the concepts studied during the course.

2. Data Exploration

Understanding the structure and content of the dataset is a critical first step in building an effective NLP model, especially when dealing with unstructured text data like tweets. Exploring the data allows us to uncover important patterns, identify potential quality issues, and assess the distribution of sentiment labels, all of which directly impact the design of preprocessing steps and the selection of modeling strategies.

In this chapter, we perform a detailed analysis of the labeled tweet dataset used for training. We examine aspects such as class distribution, tweet length, vocabulary characteristics, and the presence of noise or anomalies. These insights helped shape our decisions throughout the pipeline, from feature engineering to classification model selection.

2.1. Overview

The training dataset consists of 9 543 tweets, each labeled with one of three sentiment categories: Bearish (0), Bullish (1), or Neutral (2). Each entry includes two columns: *text*, containing the raw tweet, and *label*, representing the associated sentiment. These labels reflect the market attitude expressed in each tweet:

- Bearish (0): Represents investor caution or pessimism, often driven by concerns that the market will decline after a period of growth or due to negative signals.

- Bullish (1): Reflects investor optimism, typically triggered by positive market trends or expectations of continued growth. It implies confidence that prices will rise.
- Neutral (2): Indicates a lack of clear positive or negative sentiment. These tweets may express balanced views, uncertainty, or content unrelated to market direction.

2.2. Sentiment Analysis

To understand the balance of sentiment categories in the dataset, we analyzed the distribution of labels across the tweets. This step is crucial for identifying potential imbalances that could bias the model during training and affect performance on underrepresented classes. Additionally, to ensure better interpretation, we added a new column, *sentiment*, that contains the name of the label as a string, instead of an integer.

The table below presents the absolute and relative frequency of each sentiment label:

Table 1 – Frequency Table of Sentiments

Sentiment	**Absolute**	**Relative**
Bearish	1154	0.151
Bullish	1538	0.202
Neutral	4942	0.647

To better illustrate this distribution, the following bar plot shows the number of tweets per sentiment class:

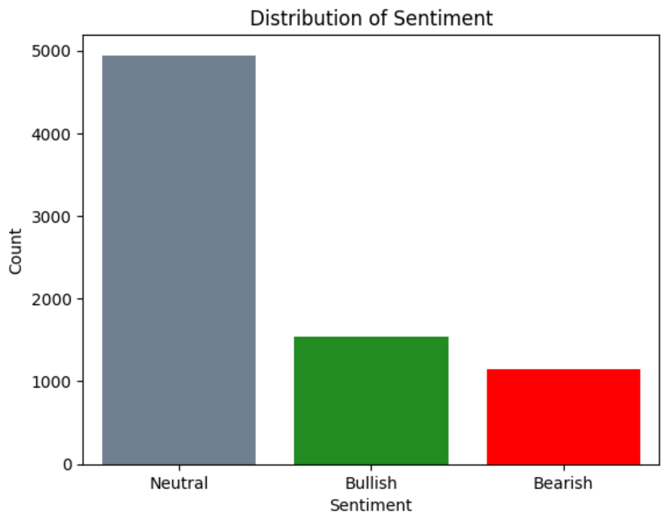

Figure 1 – Bar plot of sentiment distribution

The outputs show that most tweets are associated with Neutral sentiments, while Bullish and Bearish appear to be somewhat in balance. This skew may bias the model toward predicting Neutral sentiment unless addressed during training and highlights the need to monitor performance carefully across all classes.

During the initial exploration, we identified an opportunity to enhance the quality of our analysis by creating a new column *scale* with the sentiment labels put into a scale that reflects their natural progression from negative to positive: Bearish = 0, Neutral = 1, and Bullish = 2. While the original labels are categorical, this mapping allowed us to treat sentiment as an ordinal variable in some contexts, enabling the use of location metrics, such as median and quartiles, cumulative frequency analysis, and more expressive visualizations like boxplots.

This simple but impactful transformation gave us a deeper understanding of the sentiment distribution context and its statistical behavior across the dataset, adding interpretability and analytical depth to our exploratory process.

Table 2 – Cumulative Frequency Table of Sentiments

Sentiment	**Absolute**	**Relative**
Bearish	1154	0.151
Neutral	6096	0.799
Bullish	7634	1.0

2.3. Tweet Analysis

We analyzed the textual characteristics of the tweets to uncover patterns and take informed decisions during model design. This included measuring tweet length, identifying common terms, and comparing content across sentiment classes.

First, we calculated the number of words per tweet to assess message length. Most tweets are short, typically ranging between 5 and 20 words. A histogram suggested a somewhat right-skewed normal distribution, with a few extremely short or long tweets.

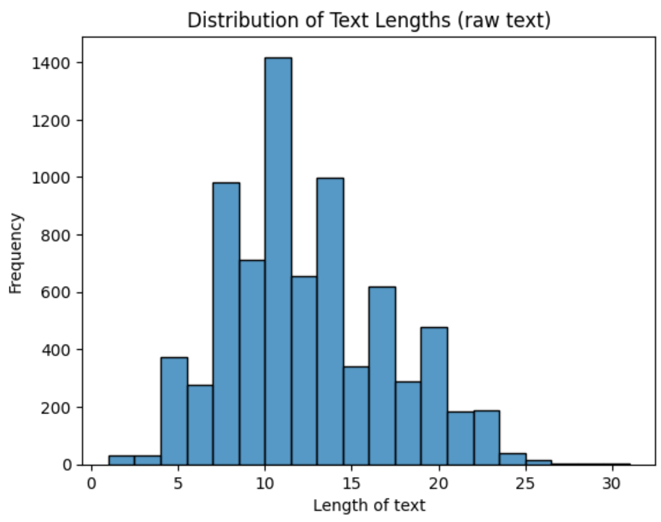

Figure 2 – Histogram of tweet length distribution

Next, we added a binary feature *has_link*, indicating whether a tweet includes a hyperlink. This allowed us to explore whether sentiment distributions differ in tweets with links. Pie charts revealed that Neutral tweets were more frequent in both cases, though Bullish posts were relatively more common among tweets without links.

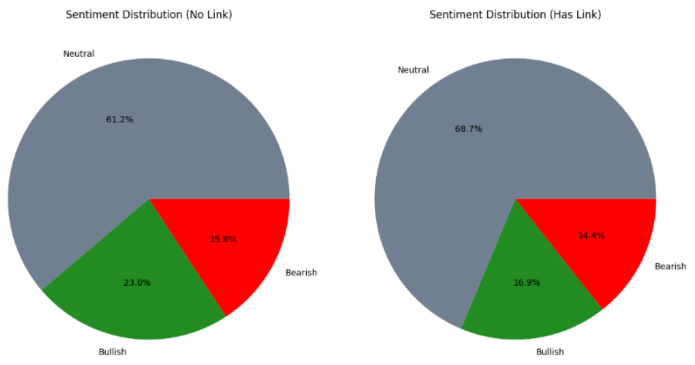

Figure 3 – Pie Charts with sentiment comparison on link existence

We then applied a custom preprocessing function that included lowercasing, URL and mention removal, hashtag cleaning, punctuation stripping, stop-word filtering, and lemmatization. This standardized the text and prepared it for feature extraction.

Using the cleaned text, we implemented a Bag-of-Words model to identify the most frequent terms overall and by sentiment class, having generated word clouds for each group. Some noticeable terms to be commonly included in Bullish tweets, thus bringing optimistic sentiment, are “new”, “price target”, “beat”, “gain”, “rise” and “target raised”. On the other hand, tweets categorized as Bearish contained terms such as “china”, “cut”, “coronavirus”, “oil”, “lower” and “hedge fund”. Neutral tweets had more general or ambiguous language.

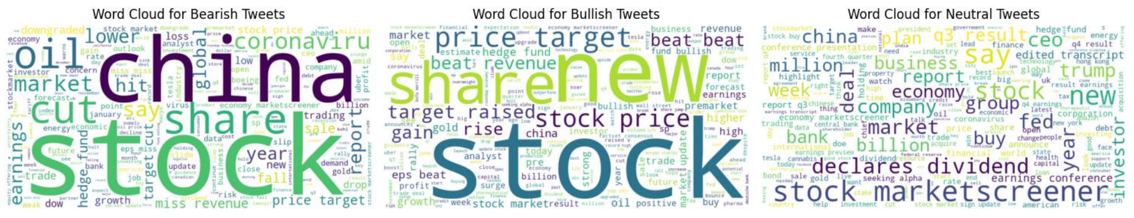

Figure 4 – Sentiment-based Word Clouds

Finally, our analysis of vocabulary patterns by sentiment class confirmed that distinct linguistic signals are associated with each market stance. Bullish tweets tend to emphasize action and optimism, Bearish ones reflect concern or warning, and Neutral tweets often adopt a more descriptive or uncertain tone. These textual distinctions provide valuable patterns not only for model training but also for understanding how sentiment is communicated in financial posts on social media.

3. Data Preprocessing

Before training our models, we applied a set of preprocessing steps to clean and prepare the tweets. Since the dataset consists of informal and noisy text from Twitter, this step was essential to remove irrelevant elements and standardize the input across different approaches.

All models used the same data split: training and validation sets were pre-generated and loaded from CSV files (*X_train.csv*, *X_val.csv*, etc.), so that all models would be evaluated on the same data, ensuring a fair comparison between them.

Although each model had its own specific preprocessing pipeline, especially when it came to tokenization and input formatting, most of them shared a common initial cleaning process. All tweets were lowercased to reduce vocabulary size and ensure consistency. We also removed several noisy elements typical of Twitter data: links (URLs), user mentions and unnecessary punctuation. In the case of hashtags, we removed just the symbol and kept the word, so something like *#bullish* was treated simply as bullish. In addition, numbers and excessive whitespace were cleaned up to simplify the input without losing meaning.

The sentiment labels were also kept consistent across all models: 0 = Bearish, 1 = Bullish, and 2 = Neutral. In models that required it, such as the LSTM, the labels were encoded using *OneHotEncoder* to match the expected format for classification.

While this basic cleaning was applied across the board, each model adapted the preprocessing to its needs:

- Classical models (e.g., *XGBoost*, *Logistic Regression*) applied more aggressive cleaning, such as removing stop-words and applying lemmatization. This helped reduce noise and improve performance when using vector-based representations like TF-IDF and Bag-of-Words.

- The LSTM model used *Keras'* Tokenizer to convert each cleaned tweet into a sequence of integers. Padding was then applied to ensure all sequences were the same length. This approach had been covered in class and worked well in combination with pre-trained embeddings like *GloVe*.
- BERT-based models used tokenizers from *HuggingFace*, which handled preprocessing internally based on the model architecture. Some of these models, like *cardiffnlp/twitter-xlm-roberta-base-sentiment*, had been explored during class, while others, such as *bert-base-uncased* and *bertweet-base*, were additional models chosen by the group to compare performance across different pretrained transformers.

Overall, this preprocessing stage ensured that all models were trained and evaluated on consistent, cleaned inputs, while still allowing flexibility for each approach to tailoring the data to its own requirements.

4. Feature Engineering

Like the preprocessing, different techniques were applied based on the nature of each pipeline. We began with the Bag-of-Words (BoW) model, which utilizes *CountVectorizer* to construct features by counting the frequency of each word in a given text. While this method provides a straightforward representation, its simplicity often overlooks the importance or relevance of words across documents. To address this limitation, we transitioned to the TF-IDF technique, which enhances feature representation by reducing the weight of common words and emphasizing more informative ones.

However, due to the limitations of such frequency-based approaches, particularly their inability to capture contextual or semantic relationships between words, we explored more advanced techniques in later iterations. Specifically, we adopted word embeddings to represent text in a dense, continuous vector space. Using pre-trained *GloVe* embeddings, we mapped each word to a high-dimensional vector that captures its meaning and relationships to other words, allowing our LSTM model to learn from sequential dependencies in the text.

To further enhance our feature representations and move beyond the constraints of static embeddings, we turned to contextual embeddings powered by transformer models. In this approach, each sentence was passed through a pre-trained model, and the resulting sentence-level embedding. Unlike traditional methods, these embeddings dynamically adjust based on the context in which words appear, offering a much richer understanding of language. We used this method to generate dense feature vectors that encapsulate the semantics of full sentences, which were then fed into various machine learning classifiers.

The first transformer model introduced in class for this purpose was *cardiffnlp/twitter-xlm-roberta-base-sentiment*, which provided a multilingual and social media-focused foundation. Building on this, we explored additional pre-trained models that were not covered in class, including:

- *cardiffnlp/twitter-roberta-base-sentiment-latest*, a *RoBERTa*-based model specifically fine-tuned on Twitter data to capture informal and emotionally charged language.
- *finiteautomata/bertweet-base-sentiment-analysis*, a *BERT*-based model trained on over 850 million English tweets and fine-tuned for sentiment analysis.

5. Classification Models

In this section, we describe the different pipelines developed to address the problem introduced in the introduction. Our approach began with relatively simple methods and gradually evolved toward more sophisticated techniques specifically designed for natural language processing (NLP).

5.1. Frequency Based (tm_tests_01_34.ipynb)

In this pipeline, we applied a range of text preprocessing techniques. Initially, we used a basic cleaning approach, like the one used during the exploratory data analysis, which involved removing noise and standardizing the text. To enhance the feature extraction process, we also experimented with different pre-trained tokenizers, believing they could improve model performance. Specifically, we used *bert-base-uncased* and *cardiffnlp/twitter-roberta-base-sentiment-latest*, both accessed through *HuggingFace*'s *AutoTokenizer*. For feature representation, we relied on two main methods: TF-IDF and Bag-of-Words

The pipeline uses two different classifiers: *K-Nearest Neighbors* (KNN), which was introduced in class, and *XGBoost*, which we included as an additional model. This second classifier is a powerful gradient boosting algorithm that's well known for its strong performance, especially on structured data. It works by building an ensemble of decision trees in sequence, improving each one based on the errors of the previous, and includes built-in regularization to help prevent overfitting.

Taking everything into account we had 12 different methods that could be used. For each of them we created a grid search on the training set using the same Grid depending on the model used (around 50 different combinations for each model). Once the best parameters were found, we made a last run for each model and evaluated them on the validation data that was kept apart throughout the overall process.

5.2. LSTM (tm_tests_02_34.ipynb)

Following the previous experiments, we tested a neural approach using a Long Short-Term Memory (LSTM) model, known for its ability to process sequential text data. After preprocessing the data, as described in Section 3, we used *Keras*' built-in Tokenizer to convert each tweet into a sequence of integers, where each unique word was mapped to a unique index based on frequency. To ensure that all input sequences had the same shape, we applied padding with a fixed maximum length. This was a crucial step for enabling the data to be passed into the LSTM model.

The architecture consisted of an embedding layer (initialized with pre-trained *GloVe*-Twitter embeddings), followed by a single LSTM layer with 128 units, and a fully connected dense layer with a *softmax* activation function to output the sentiment prediction. The embedding layer was set as non-trainable to preserve the integrity of the pre-trained vectors.

We trained the model using categorical cross-entropy as the loss function and *Adam* as the optimizer, with a learning rate of 0,001. The labels were encoded using *OneHotEncoder*. Additionally, as an attempt to avoid overfitting, we implemented early stopping based on validation loss. We set a maximum of 20 epochs, although training usually stopped earlier thanks to the callback. Throughout the process, we monitored both accuracy and macro-F1 score to evaluate performance across all classes.

Overall, this approach allowed us to explore how a sequential model like LSTM performs on informal, noisy financial text and how well it can leverage word embeddings to capture meaning in short messages like tweets.

5.3. Pre-trained Transformer Models (BERT) (tm_tests_03_34.ipynb)

This section of the project is divided into two parts. Firstly, we evaluate pre-trained Transformer models using *Hugging Face's* pipeline *sentiment-analysis*.

In this part, the models are used solely in inference mode, this means that they are not fine-tuned or trained on our dataset. Instead, the pipeline receives the raw tweet texts directly from the dataset as input, automatically handling tokenization using the appropriate tokenizer associated with each pre-trained model. Since Transformer models rely on full text context to capture meaning, we didn’t apply traditional preprocessing techniques as done in the previous pipelines.

The second part of this section involves extracting embeddings from a pre-trained language model and using them to train multiple classifiers. To understand how different learning approaches perform on Transformer features, we tested different models. Logistic Regression which predicts classes based on weighted input features. Support Vector Machine attempts to find the optimal boundary that maximizes the margin between classes. *Random Forest* builds decision trees and combines their outputs to make predictions, while Gradient Boosting builds trees sequentially with the objective to correct the errors of the previous trees. *XGBoost*, also a tree-based model, adds regularization and optimization to try improve generalization. Finally, the MLP Classifier, which uses a small neural network to capture non-linear patterns in the data.

6. Evaluation and Results

In order to evaluate the performance of each model, we relied on a set of complementary evaluation metrics. Our main baselines were accuracy and F1 score. Accuracy provides a general overview of how many predictions were correct across all classes, but it can be misleading in the presence of class imbalance. Therefore, we placed particular emphasis on the macro-average F1 score, which considers both precision and recall for each class and gives equal weight to all classes, regardless of their frequency. This made it especially suitable for our imbalanced dataset.

In addition to these core metrics, we also analyzed the classification report, which includes precision, recall, and F1 score per class. Precision indicates how many of the predicted labels were correct, while recall measures how many of the actual labels were correctly identified. Together, these metrics provided a more nuanced view of model performance, particularly in capturing minority classes.

6.1. Frequency Based

When we first ran this pipeline, we didn't expect great results, especially since the models weren't built specifically for NLP. But the evaluation results were surprisingly good. *XGBoost* consistently outperformed KNN, particularly on the validation F1-score, which was key due to our imbalanced labels. The best setup combined Bag-of-Words with the *bert-base-uncased* tokenizer, reaching a validation F1 of 0,710 (table 1).

KNN, on the other hand, often overfit the training data, scoring perfectly on the train set but dropping significantly on validation, sometimes below 0.45. It struggled when paired with high-dimensional features like TF-IDF or transformer-based tokenizers. *XGBoost* handled these better, showing stronger generalization and overall better performance. In the end, this pipeline proved to be a strong baseline for our work.

6.2. LSTM

The LSTM model performed well overall, reaching a validation accuracy of 0,8025 and a macro F1 score of 0,7013. Performance was strongest on the Neutral class (F1 = 0.88), which represented the majority of tweets in the dataset, while the Bearish and Bullish classes obtained F1 scores of 0,57 and 0,65, respectively.

The training process showed smooth convergence, with both accuracy and loss curves confirming that the model learned effectively without memorizing the training data.

While class imbalance posed a challenge, especially for the minority Bearish class, the model was able to generalize well and capture meaningful patterns in the tweet content using only word embeddings and a relatively simple architecture.

6.3. Pre-trained Transformer Models (BERT)

In the first part of our evaluation, the pre-trained model that performed better was *cardiffnlp/twitter-roberta-base-sentiment-latest*, achieving an accuracy of 0.683 and a macro F1-score of 0.587. While *cardiffnlp/twitter-xlm-roberta-base-sentiment* had the lowest performance with 0.576 accuracy and 0.449 F1. Since these models are not trained in our dataset, we cannot study overfitting/underfitting. Therefore, their performance reflects only how well each model generalizes on the validation set.

In the second part, the best classifier was the MLP Classifier, reaching 0,838 validation accuracy and a macro F1-score of 0,784. The worst performance came from the *Random Forest*, which overfitted with perfect training scores but only 0.721 validation F1 and 0.807 accuracy. This overfitting trend was also visible, in *XGBoost*, while Logistic Regression and SVC showed a more balanced generalization.

Overall, these results demonstrate that even simple classifiers, when combined with strong contextual embeddings, can achieve competitive performance on the target classification task.

7. Final Model

For our final approach, we selected the model that delivered the best overall results. This setup used the pre-trained *RoBERTa* tokenizer and embeddings from *cardiffnlp/twitter-roberta-base-sentiment-latest*, combined with a *Multi-Layer Perceptron* (MLP) classifier. Although the model showed clear signs of overfitting on the validation set, it still achieved the highest accuracy and F1 score among all models tested. The second-best model, which used the same embeddings but with *XGBoost* as the classifier, also showed similar overfitting. Given that both models had this limitation, we decided to proceed with the one that performed best overall.

To strengthen our final model choice, we decided to run a grid search on top of it. While it's not ideal to tune only one model and ideally, all classifiers should undergo the same process, we chose to do so due to computational constraints. Still, we felt it was a good way to get the most out of our model and make sure it was performing at its full potential.

Table 3 – Final Model evaluation metrics

Train Accuracy	**Train F1-Score**	**Val Accuracy**	**Val F1-Score**
1,000	1,000	0,833	0,779

These results confirmed that, despite some overfitting, our final model was able to generalize reasonably well and delivered the strongest overall performance in our experiments.

8. Conclusion

Throughout this project, we gained a much deeper understanding of how NLP works in practice. We built and compared several different pipelines to explore how various models and approaches handle the same problem. One of the biggest surprises was seeing how well *XGBoost* performed when combined with a pre-trained tokenizer and a simple frequency-based encoder like Bag-of words. Initially, we assumed that achieving strong results would require more specialized deep learning models, but our experiments showed that even traditional methods can be highly effective when paired with smart feature engineering.

For our final model, we chose a combination of pre-trained embeddings and a classifier on top to make our predictions. While this setup delivered the best performance overall, it also came with limitations, most notably, overfitting on the validation set. Although other models may have generalized better, they also consistently scored lower. In future iterations, our main goal would be to address this overfitting issue. One potential improvement would be to move beyond frozen embeddings, since we used only frozen pre-trained models for tokenization and embedding, allowing parts of the model to fine-tune during training could help adapt it better to our specific task.

Overall, most of our top-performing models were capped at around 0,75 F1 score and 0,80 accuracy. Breaking through that performance ceiling will likely require more task-specific fine-tuning or the integration of more advanced techniques in future work.

APPENDIX

Appendix Table 1 - Frequency based results

	Train Accuracy	Train F1-score	Val Accuracy	Val F1-score
XGboost, bow, simple cleaning	0.845	0.770	0.778	0.659
XGboost, bow, “bert-base-uncased"	0.925	0.896	0.813	0.710
XGboost, bow, “cardiffnlp/twitter-roberta-base-sentiment-latest"	0.903	0.862	0.796	0.685
XGboost, tf-idf, simple cleaning	0.866	0.805	0.767	0.644
XGboost, tf-idf, “bert-base-uncased"	0.949	0.931	0.802	0.6863
XGboost, tf-idf, “cardiffnlp/twitter-roberta-base-sentiment-latest"	0.929	0.902	0.787	0.6714
KNN, bow, simple cleaning	0.999	0.999	0.696	0.439
KNN, bow, “bert-base-uncased"	1.000	1.000	0.700	0.4813
KNN, bow, “cardiffnlp/twitter-roberta-base-sentiment-latest"	0.755	0.594	0.701	0.469
KNN, tf-idf, simple cleaning	0.674	0.354	0.6689	0.341
KNN, tf-idf, “bert-base-uncased"	0.814	0.727	0.767	0.649
KNN, tf-idf, “cardiffnlp/twitter-roberta-base-sentiment-latest"	1.000	1.000	0.77	0.647

Appendix Table 2 – LSTM results

	Train Accuracy	Train F1-score	Val Accuracy	Val F1-score
LSTM	0.866	Unknown	0.802	0.701

Appendix Table 3 – Pre-trained Transformer Models results

	Train Accuracy	Train F1-score	Val Accuracy	Val F1-score
“cardiffnlp/twitter-roberta-base-sentiment-latest", MLP	1.000	1.000	0.838	0.784
“cardiffnlp/twitter-roberta-base-sentiment-latest", XGB	1.000	1.000	0.834	0.773
“cardiffnlp/twitter-roberta-base-sentiment-latest", LogisticRegression	0.896	0.863	0.8167	0.758
“cardiffnlp/twitter-roberta-base-sentiment-latest", SVC	0.847	0.787	0.823	0.755
“cardiffnlp/twitter-roberta-base-sentiment-latest",GradiantBossting	0.895	0.858	0.823	0.754
“cardiffnlp/twitter-roberta-base-sentiment-latest" ,RandomForest	1.000	1.000	0.807	0.720