Decoding Emotions: Sentiment Analysis on Twitter Data

Nicolás Romeu García

1674080

Abstract—This project explores the analysis and classification of sentiments expressed in tweets analyzing a dataset with over 27,000 different texts. By focusing on text data, the project aims to understand how individuals convey positive, neutral, or negative emotions in social media messages, more specifically, in Twitter. Through preprocessing and feature engineering, the raw textual data is transformed into structured representations suitable for machine learning models. The primary goal is to build an effective sentiment classification system that accurately predicts the sentiment of a given tweet. The project evaluates multiple classification algorithms and word vectorizing methods assesses their performance using the F1-score as the main metric. Aside predicting, the project also seeks to help you visualize the emotional content of tweets through techniques like WordClouds and comparative frequency analysis. These visualizations provide insights into common themes across different sentiment categories. Ultimately, the project combines natural language processing [6], machine learning, and data visualization to contribute to understanding online communication and developing practical tools for sentiment analysis.

Keywords—*Sentiment Classification, Data Visualization, Natural Language Processing, Twitter Data*

Contents

1. Introduction

In the digital age, platforms like Twitter (now known as X) provide a wealth of user-generated content where individuals express their thoughts and emotions on a wide range of topics. Understanding the sentiments behind these texts has become an essential task for applications in business, social sciences, and public opinion research. This project investigates sentiment analysis using a dataset with over than **27,000 tweets**, aiming to classify them as **positive, neutral, or negative**.

The study employs **natural language processing (NLP) techniques** to clean and structure raw text data, preparing it for machine learning algorithms. By comparing various **classification models** and **feature extraction methods**, the project seeks to identify the best approaches for accurate sentiment prediction, while key metrics, such as the **F1-score**, guide the evaluation of the models.

Beyond the classification, another important part of this project is the in-depth comparison between the **selected text column** (default column that had the dataset with the words that best encapsulate the sentiment) and the **predicted text**, a new column of the dataset that is created by calculating **which words are the most important according to the model and feature extraction method with the best accuracy**. This analysis provides insights into the consistency and discrepancies between the author's key words and the ones predicted by the model by examining word frequencies and distribution differences.

2. Sentiment Classification

Sentiment classification involves categorizing tweets into three distinct classes: positive, neutral, and negative. In this section, we analyze the dataset to identify patterns, preprocess the text to ensure consistency, and apply various machine learning algorithms to optimize classification performance.

2.1. Data Cleaning and Preprocessing

The raw dataset has over 27,000 tweets, presenting diverse text inputs that require cleaning and standardization for effective analysis. The preprocessing steps focus on removing inconsistencies, reducing noise, and basically making the data more readable for the model.

Originally, the dataset was structured with the following columns:

1. **tweetID**: Unique identifier for each tweet
2. **text**: The text of the tweet (without being processed)
3. **Sentiment**: The sentiment related to the tweet
4. **Selected_text**: Text supporting the sentiment of the tweet

The first step was to drop some of the columns in the dataset. The justification to why dropping those columns is the following: **tweetID** is dropped as it's a unique identifier for each tweet, so it doesn't give any important information in order to classify. On the other hand, **Selected_text** is dropped because it already has the words that support the sentiment, so it feels like some sort of 'cheating' to use it to predict, as the model wouldn't really have to 'think' much to predict correctly.

The last steps are mapping the labels, using **0 for negative, 1 for neutral and 2 for positive** and to finish cleaning the dataset, I created a function that removes user mentions, hashtags, URLs, and special characters, while converting emojis into descriptive text to retain their sentiment-related context. Furthermore, tweets are converted to lowercase, and stopwords are filtered out, minimizing irrelevant information.

```python
def preprocess_tweet(tweet):
    if not isinstance(tweet, str):
        return ""
    tweet = re.sub(r"@\w+", "", tweet)
    tweet = re.sub(r"#\w+", "", tweet)
    tweet = re.sub(r"http\S+|www\S+", "", tweet)
    tweet = emoji.demojize(tweet)
    tweet = re.sub(r"[^a-zA-Z\s\*]", "", tweet)
    tweet = tweet.lower().strip()
    tweet_2 = " ".join([word for word in tweet.split()
     if word not in stop_words])
    if tweet_2 == "":
        return tweet
    return tweet_2
```

Python Code 1. Text cleaning function

2.2. Data Visualization

Once we have our dataset clean, let's get into the data visualization. Data visualization plays a crucial role in understanding the distribution and structure of the dataset.

The analysis begins with a radar plot to represent the distribution of tweets across positive, neutral, and negative sentiments. This visualization highlights potential class imbalances, which are critical to address during model training.

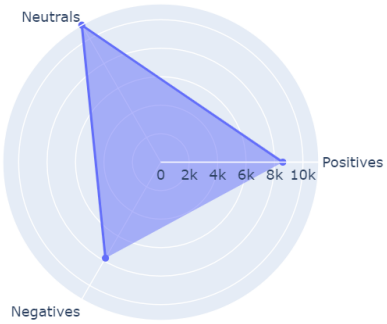

Figure 1. Radar plot of sentiment distribution

It is also performed a word frequency analysis to identify the most common words associated with each sentiment. These frequencies are visualized through **WordClouds** [4], offering an intuitive view of the linguistic patterns and themes prevalent in the dataset. A **WordCloud** is a visual representation of text data where the frequency of each word is depicted by its size. What this basically means is that words that appear more frequently are displayed larger, while less frequent words appear smaller. You can customize them to create shapes, change color schemes and even font styles. This is the function that I wrote to create this plots.

```python
def plot_wordcloud(text, title, title_size):
    words = text
    allwords = []
    for wordlist in words:
        allwords += wordlist
    mostcommon = FreqDist(allwords).most_common(140)

    wordcloud = WordCloud(
        width=1200,
        height=800,
        background_color='white',
        stopwords=set(STOPWORDS),
        max_words=150,
        scale=3,
        mask=mask,
        contour_width=0.5,
        contour_color='white',
    ).generate_from_frequencies(dict(mostcommon))

    fig = plt.figure(figsize=(18, 18), facecolor='white')
    plt.imshow(wordcloud, interpolation='bilinear')
    plt.axis('off')
    plt.title(title,
              fontdict={
                  'size': title_size,
                  'verticalalignment': 'bottom'
              })
    plt.tight_layout(pad=0)
    plt.show()
```

Python Code 2. WordCloud plot function

Lastly, this is how this plots would look for each sentiment:

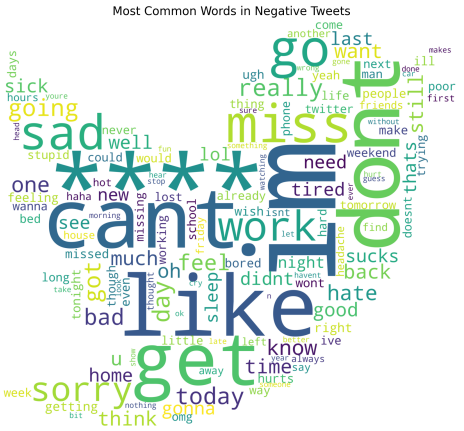

Figure 2. WordCloud for Negative Sentiment

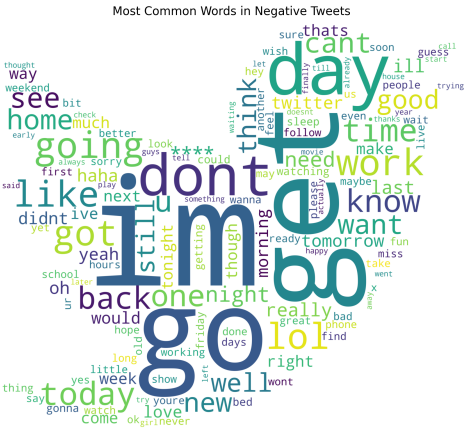

Figure 3. WordCloud for Neutral Sentiment

Figure 4. WordCloud for Positive Sentiment

A thing to note here is that once the frequency analysis was done, I realized that even if 'im' is not a stopword itself, it is very common on the 3 different sentiments. Because this word doesn't really bring any information itself, I decided to modify the function **preprocess_tweet** a little so it also deletes it when doing the preprocessing.

```python
    stopwords = stop_words.copy()
    stopwords.add("im")
    tweet_2 = " ".join([word for word in tweet.split()
     if word not in stopwords])
```

Python Code 3. Lines added to the function preprocess_tweet

It might also be noticeable that the * are not being deleted. This is the case because actually on the original text cuss words are censored, so they instead of putting the word itself, they put the *, and because cuss words are often related with a negative connotation, I decided to leave them as I thought it would be helpful to have them.

2.3. Metric Selection

Choosing the right evaluation metric is critical to evaluate the performance of sentiment classification models. Given the class imbalance in the dataset, with some sentiment categories occurring more frequently than others, accuracy alone is insufficient as it may not reflect the model's ability to correctly classify minority classes.

To address this, the **macro-averaged F1-score** (also known as F1_macro) is selected as the primary evaluation metric. This metric provides a **harmonic mean of precision and recall**, ensuring that the model performs well in identifying positive and negative classes, even in the presence of imbalances.

2.4. Classification

This section focuses on building sentiment classification models using machine learning. Using two different feature extraction techniques, **Term Frequency-Inverse Document Frequency** (TF-IDF) [5] and **Bag of Words** (BoW) [1], textual data is transformed into numerical representations that can be processed by machine learning algorithms. The primary objective is to compare the performance of these methods and determine their effectiveness in predicting sentiment categories.

To achieve this, **Multinomial Naive Bayes, Bernoulli Naive Bayes, Complement Naive Bayes, and Logistic Regression models** are applied with both vectorization techniques while also doing cross-validation. The objective of this is to find out if there is a combination of a feature extraction technique and a predictive model that outperforms the rest, so later on when searching hyperparameters the efforts can focused on that specific combination.

2.4.1. Classification with TF-IDF

TF-IDF (Term Frequency-Inverse Document Frequency) is a text vectorization technique that assigns importance to words based on how frequently they appear in individual documents relative to their frequency across the entire dataset. This method helps emphasize terms that are more distinctive to specific tweets while down-weighting commonly used words.

The first step is to do the vectorization and for this dataset, the maximum number of significant terms is fixed at 5,000 for efficiency purposes. Once the vectorization is done, the data gets split into 10 different folds by using **Stratified K-Fold**. to perform cross-validation. The four different models will use those folds to predict and lastly evaluate their performance using f1_macro as the main metric.

To help us visualize the results, a plot like this is created with each model's score and it's cross validation means and errors.

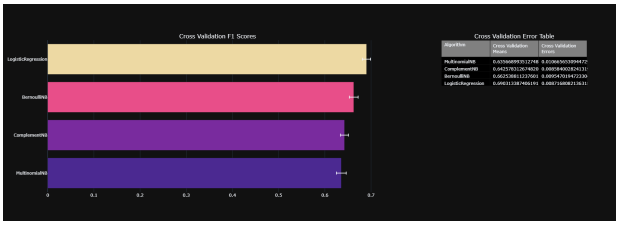

Figure 5. TF-IDF Cross Validation F1 Scores

2.4.2. Classification with BoW

Bag of Words (BoW) is a simpler text vectorization method that represents text as a collection of word counts or binary occurrences, disregarding word order and context. While it lacks the nuanced weighting of TF-IDF, it can still provide meaningful representations of text, particularly for sentiment analysis tasks.

The classification part is very similar to the one done for TF-IDF if not practically the same. For the vectorization the maximum vocabulary size is also set in 5,000 words, again, for computational efficiency purposes. Once the vectorization is done, data is also divided in 10 different folds using **Stratified K-Fold**. The models will use once again the cross-validation technique to predict and evaluate their performance, using also the f1_macro as the main metric for performance measuring.

To help us visualize the results, the same plot than before is created but for Bag of Words.

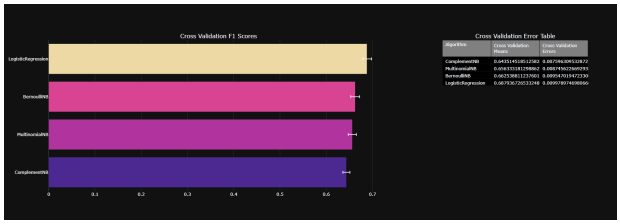

Figure 6. BoW Cross Validation F1 Scores

> **Information**
>
> As we can see, there is no significant difference in the scores between the feature extraction techniques, making it necessary to perform hyperparameter tuning for all combinations.

2.5. Hyperparameters search

To improve the performance of the sentiment classification models, a search is performed to find the best hyperparameters for each combination of feature extraction method (TF-IDF and Bag of Words) and predictive model.

In order to evaluate different hyperparameter combinations and their performance, we use **GridSearchCV** combined with **Stratified K-Fold cross-validation**. For Naive Bayes models (Multinomial, Bernoulli, and Complement), the smoothing parameter α is fine-tuned to control how probabilities are adjusted for unseen words. For Logistic Regression, hyperparameters such as regularization strength (C), penalty type (L1 or L2), and solver type are explored. The best scores for each vectorizing method are the following:

Results for BoW:

- **MultinomialNB:** Best F1 Score: 0.6637 | Best Parameter: $\alpha = 2.$
- **BernoulliNB:** Best F1 Score: 0.6604 | Best Parameter: $\alpha = 1.5$
- **ComplementNB:** Best F1 Score: 0.6501 | Best Parameter: $\alpha = 2.$
- **LogisticRegression:** Best F1 Score: 0.7081 | Best Parameters: $C = 1.0$, class_weight = balanced, penalty = ℓ_1, solver = liblinear

Results for TF-IDF:

- **MultinomialNB:** Best F1 Score: 0.6332 | Best Parameter: $\alpha = 1.$
- **BernoulliNB:** Best F1 Score: 0.6604 | Best Parameters: $\alpha = 1.5$
- **ComplementNB:** Best F1 Score: 0.6622 | Best Parameters: $\alpha = 2.$
- **LogisticRegression:** Best F1 Score: 0.7106 | Best Parameters: $C = 1.$, class_weight = balanced, penalty = ℓ_1, solver = liblinear

3. Results Analysis

In this section, we present a detailed analysis of the performance of the sentiment classification models, focusing on both the predicted results and their comparison with the selected text given by default. The goal is to assess the models' effectiveness in predicting sentiment categories and explore how well they align with the expected outcomes. Additionally, we will visualize the results to gain a deeper understanding of the model's performance and its ability to differentiate between various sentiment categories.

3.1. Creation of the Predicted Text

After performing the hyperparameter search, we concluded that the best model is **Logistic Regression** for **TF-IDF**, so this is the combination I'm going to be using to search for the most important words predicted.

This is actually done with the purpose of creating a column called **predicted_selected_text** that includes the words that the model found the most important to classify each tweet. This is done thanks to the implementation of a function that I created called **extract_keywords**, which works by following the next steps:

1. **Vectorizing:** The function starts by transforming the input tweet into the selected representation using the pre-fitted vectorizer (In this case, TF-IDF).

2. **Predicting:** Using the model (Logistic regression in this case), it predicts the sentiment of the tweet. The class index of the predicted sentiment is used to find the corresponding coefficients for that class, which reflect the importance of each word.

3. **Contribution Calculation:** It then calculates the contribution of each word to the sentiment prediction. The contribution is computed by multiplying the model's coefficient for each feature by the corresponding value in the tweet's TF-IDF vector.

4. **Keyword Selection:** To focus on the most influential words, the function selects the top keywords based on their contributions. A percentile threshold is applied, where only words with contributions above a certain percentile are retained.

5. **Keyword Ordering:** The selected keywords are then ordered according to their appearance in the original tweet, ensuring that the final output preserves the structure of the text.

This is an example of how the 20 first rows of my dataset after calculated the predicted selected words and setting a threshold of 40%:

	cleaned_text	selected_text	predicted_selected_text
0	id responded going	I'd have responded, if I were going	id
1	sooo sad miss san diego	Sooo SAD	sad
2	boss bullying	bullying me	boss
3	interview leave alone	leave me alone	alone
4	sons **** couldnt put releases already bought	Sons of ****,	couldnt
5	shameless plugging best rangers forum earth	http://www.dothebouncy.com/smf - some shameles...	best
6	feedings baby fun smiles coos	fun	baby fun
7	soooo high	Soooo high	high
8	both of you	Both of you	you
9	journey wow u became cooler hehe possible	Wow... u just became cooler.	wow cooler hehe
10	much love hopeful reckon chances minimal p nev...	as much as i love to be hopeful, i reckon the ...	love cake
11	really really like song love story taylor swift	like	really really like love
12	sharpie running dangerously low ink	DANGERously	sharpie running dangerously low ink
13	want go music tonight lost voice	lost	lost
14	test test lg env	test test from the LG enV2	test test lg env
15	uh oh sunburned	Uh oh, I am sunburned	sunburned
16	sok trying plot alternatives speak *sigh*	*sigh*	sok trying plot alternatives speak *sigh*
17	ive sick past days thus hair looks wierd didnt...	sick	sick didnt
18	back home gonna miss every one	onna	gonna miss
19	hes	Hes just not that into you	hes

Figure 7. Plot of the first 20 rows of the dataset

3.2. Visualazing the Comparison

This next and last part of the work consists on comparing both **predicte_selected_text** and **selected_text** by exploring different visualization techniques to highlight discrepancies and agreements between both columns [2].

The first and maybe most basic thing that we wanted to visualize is the difference between the number of words that appear on the selected text vs the predicted text:

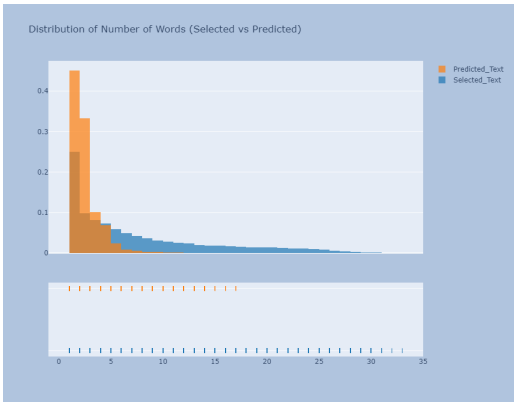

Figure 8. Distribution of Number of Words (Selected vs Predicted)

We can see that there is a big difference between the number of words used, mostly because there isn't any predicted text longer than 17 words, but there are way more than in selected text with less than 4 words, which probably means that the model tends to simplify the sentiment expressions by focusing on shorter, more concise phrases.

Another interesting visualization involves examining the distribution of the difference in word counts between the selected and predicted texts. To achieve this, a kernel density plot was created to illustrate the distribution of word count differences. To make it more illustrative, the plot is also separated by sentiment categories, making the analysis more significant and interesting.

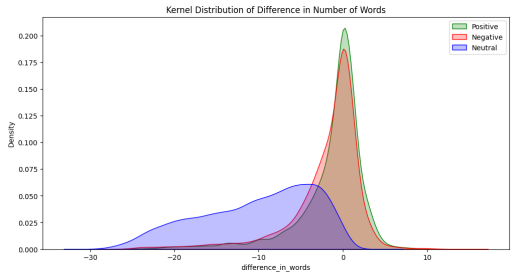

Figure 9. Kernel Distribution of Difference in Number of Words

As we can see, positive and negative sentiment have a very similar ammount of words, as there is a big peak around the 0 for both sentiments. On the other side, the neutral one has a very different distribution, which could mean that the model struggles to consistently predict the correct number of words for negative sentiments, possibly beacuse of the complexity in how negative emotions are expressed in the text.

Another interesting insight would be showing the **Jaccard Score** [3] between the selected and predicted text. This socre measures similarity between two sets by dividing the size of their intersection by the size of their union. This is how the kernel density plot for the Jaccard Score looks like:

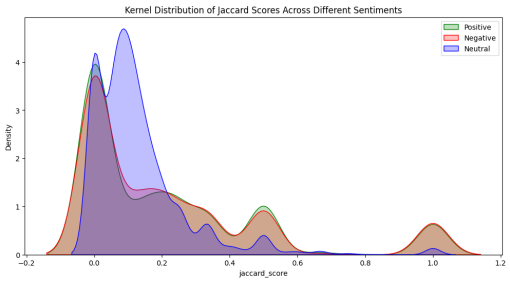

Figure 10. Kernel Distribution of Jaccard Scores

In this plot, it is noticeable that the three sentiments have actually pretty similar distributions, with peaks 0, 0.5 and 1, even though neutral has an extra peak at around 0.1, which could mean that the model occasionally assigns partial matches for neutral sentiment, possibly capturing less relevant or incomplete portions of the selected text compared to positive and negative sentiments.

We visualized the distribution of how many words there are in each column, but we haven't seen yet which words are the most common for each column, so first let's visualize the first 20 most common words for each one:

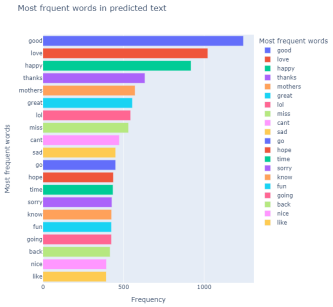

Figure 11. Caption for the first image.

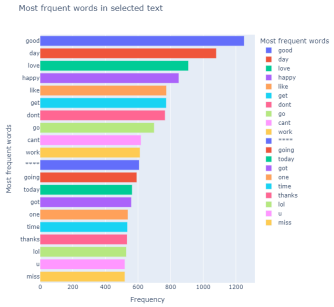

Figure 12. Caption for the second image.

Information

To visualize the most common words in the selected text, the stopwords were eliminated by using the function with we applied the preprocess to the raw text.

Lastly, even having done this last analysis to see the most common words for each column, it became obvius that an even insightful approach would be identifying the most frequent words not just for each text but also **for each sentiment**. To achieve this, I decided to create donut plots, which provide a dynamic and visually engaging representation of the most common words associated with each sentiment:

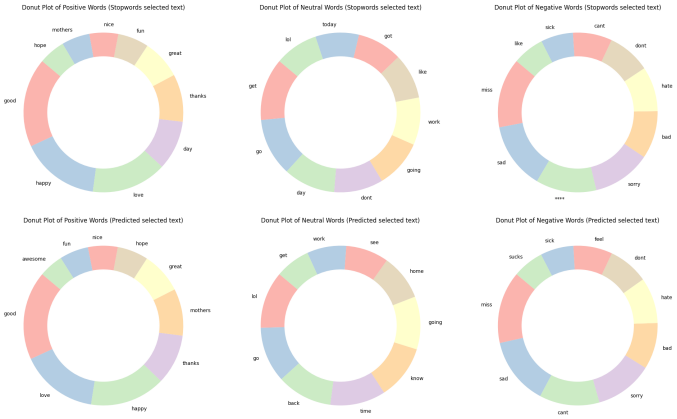

Figure 13. Donut Plot for each Sentiment

4. Future improvements

1. **Deep Learning:** The models used were pretty basic, so a possible next step could be to explore neural networks like Recurrent Neural Networks (RNNs), Long Short-Term Memory (LSTM) networks, or Transformers (e.g., BERT) for better handling of text sequences and context.

2. **Advanced Feature Engineering:** One of the reasons the model wasn't working so well could be because the feature extraction methods that are used don't work so well, so a possible thing to implement in the future is the usage of pre-trained word embeddings (e.g., Word2Vec, GloVe) to capture semantic relationships between words, which can improve the model's understanding of sentiment.

5. Code and Data Availability:

The code for this project is available in this GitHub Repository, and the data for this project is from this Kaggle dataset

References

[1] *Bag of Words Vectorizer*. [Online]. Available: https://scikit-learn.org/1.5/modules/generated/sklearn.feature_extraction.text.CountVectorizer.html.

[2] *Creating and Updating Figures in Python*. [Online]. Available: https://plotly.com/python/creating-and-updating-figures/.

[3] *Jaccard Score*. [Online]. Available: https://scikit-learn.org/1.5/modules/generated/sklearn.metrics.jaccard_score.html.

[4] *Masks for WordClouds*. [Online]. Available: https://www.kaggle.com/datasets/andreshg/masksforwordclouds?select=twitter_mask.png.

[5] *TF-IDF Vectorizer*. [Online]. Available: https://scikit-learn.org/1.5/modules/generated/sklearn.feature_extraction.text.TfidfVectorizer.html.

[6] *What is NLP (natural language processing)?* [Online]. Available: https://www.ibm.com/topics/natural-language-processing?regionCode=US&languageCode=en&cm-history=US-en.