Brand tracking based on Sentiment Analysis using Twitter Data

Andrea Toval Borrell

Abstract – Brand sentiment monitoring is crucial for businesses to understand customer perceptions. However, the real-time analysis of social media data can be challenging. Fortunately, advanced deep learning models based on transformers offer exceptional solutions to address this issue. Hence, the objective of this project is to fine-tune two widely used transformer models, BERT and RoBERTa, to achieve state-of-the-art sentiment results. Additionally, a user-friendly interface will be developed using Streamlit, providing an effective platform to showcase the outcomes of the sentiment analysis. This interface will help companies to assess the overall sentiment of the public, enabling them to obtain valuable insights into their customers' views on their products and services.

Keywords – Real-time analysis, monitoring, sentiment analysis, social media, advanced algorithms, transformers, user-friendly interface, valuable insights.

Resumen – El monitoreo del sentimiento de marcas es crucial para que las empresas entiendan las percepciones de los clientes. Sin embargo, el análisis en tiempo real de los datos de las redes sociales puede ser complicado. Afortunadamente, los modelos avanzados de deep learning basados en transformers ofrecen soluciones excepcionales para abordar este problema. Por lo tanto, el objetivo de este proyecto es hacer un fine-tuning de dos modelos de transformers muy conocidos, BERT y RoBERTa, para lograr resultados de sentimiento de última generación. Además, se desarrollará una interfaz fácil de usar usando Streamlit, proporcionando una plataforma efectiva para mostrar los resultados del análisis de sentimiento. Esta interfaz ayudará a las empresas a evaluar el sentimiento general del público, lo que les permitirá obtener información valiosa sobre las opiniones de sus clientes sobre sus productos y servicios.

Palabras clave – Análisis en tiempo real, monitoreo, análisis de sentimientos, redes sociales, algoritmos avanzados, transformers, interfaz fácil de usar, información valiosa.

1 Introduction

Nowadays important brands such as Nike, Apple, Samsung, Microsoft, Netflix, and others define the marketplace in an outstanding manner. These brands do not only have an extraordinarily high reputation, but they also create an identity for people who decide to use them, influencing their behaviour in ways that extend way beyond the point of sale. However, new products or changes in the terms of their services may be prone to leave behind bad reviews or even an atrocious reputation, inducing their clients to shift to alternative brands, hence their competition.

For this reason, monitoring how well a brand is doing is an essential task for any current company, since detecting potential problem areas could help to mitigate them before they have a chance to escalate to a major concern. Additionally, brand tracking offers the possibility not only to foresee imminent complications, but also to acknowledge how customers perceive and interact with your brand, being able to evaluate your strategies and improve based on their feedback.

In general, tracking brand performance is crucial for businesses to preserve a strong and favourable brand identity, attract and keep customers, and remain competitive in the market. Analyzing data from a wide range of sources, such as social media, online reviews, surveys, and feedback forms, is vital to comprehend how a brand is performing and what efforts might be taken to enhance it.

However, it is important to keep in mind that a brand may be remembered or viewed for good or bad reasons, as you analyze brand awareness, perception, or reputation. Because of this, when you evaluate all these factors, you should also take the brand sentiment into account. Brand sentiment can be obtained through sentiment analysis, which is the process of recognizing and categorizing the opinions conveyed in a piece of text using natural language processing, especially to determine the polarity of views on a particular topic, which might be positive, negative,

• Contact E-mail: 1566145@uab.cat
• Work tutored by: Eduardo César Galobardes (Computer Architecture and Operating Systems)
• Course 2022/23

or neutral. It is very helpful in business or marketing, to understand how these opinions and feelings can drive consumer behaviour [1].

Taking everything into account, it is important for businesses to monitor how positive or negative consumers' perception of their brand are, to be proactive and respond in the event of a possible misstep. This can be accomplished by creating a real-time system based on sentiment analysis.

Therefore, the problem at hand is to develop a simple and intuitive interface using Streamlit to display sentiment results. To achieve this, the most advanced deep learning models, BERT and RoBERTa, are going to be used to perform the sentiment analysis task.

The main contributions of this work include:

1. **Data recollection**: This step includes collecting a wide range of tweets with annotated sentiment labels related to brands or products. Additionally, to ensure an effective analysis, we meticulously prepare and balance the gathered data.
2. **Fine-tuning BERT and RoBERTa models**: This involves adapting the models' weights to specifically understand and classify sentiment from tweets. By fine-tuning these models on a labelled dataset, their ability to accurately predict sentiment will be enhanced.
3. **Model evaluation and selection**: Once the fine-tuning process is complete, the performance of both models will be evaluated by measuring metrics such as accuracy, precision, and recall. In this way, we can assess the models' effectiveness in predicting sentiment.
4. **Interface development with Streamlit**: An interface, with several metrics and visualizations, will be developed using Streamlit. Providing users with a simple and interactive way to display sentiment results of a specific brand.

The rest of this document is organized as follows. In Section 2, an overview of the current state-of-art in sentiment analysis models will be presented. Section 3 will outline the primary objectives of the project. Moving on to Section 4, a detailed explanation of the data recollection process for sentiment analysis will be provided. Section 5 will delve into the specifics of BERT and RoBERTa models, highlighting their architectures, pre-training process and significance in sentiment analysis. Section 6 will outline the necessary steps to utilize these prior models for sentiment classification. Next, in Section 7, a demonstration of a dashboard using Streamlit will be showcased, displaying the sentiment analysis results. Lastly, Section 8 will present the concluding remarks and summarize the key findings of the entire project.

2 State-of-the-art

Over the last few years, the AI and machine learning community has made significant advancements in building larger models that have demonstrated impressive capabilities, particularly in the field of Natural Language Processing (NLP). The introduction of the transformer architecture by Google in 2017 [2] has played a crucial role in driving this rapid progress in NLP. Since its publication, numerous variations of transformers have emerged, and one particularly renowned model is BERT, which has established its dominance in various NLP tasks and some of the highest-performing models are based on BERT's foundations.

Without a doubt, the publication of the paper *"Attention Is All You Need"* and its code, along with its impressive performance in tasks like machine translation, has led some experts in the field to consider transformers as potential substitutes for LSTMs. This inclination was further reinforced by the fact that they are better at handling long-term dependencies. While LSTMs can capture some dependencies over shorter distances by relying on past hidden states, their sequential nature and susceptibility to gradient issues limit their ability to capture long-range dependencies effectively. In contrast, transformers outperform these architectures by avoiding recursion and processing sentences as a whole rather than word by word. They also use multi-head attention mechanisms to learn relationships between words, enabling effective modelling of long-term dependencies [3].

Overall, **transformer-based models**, such as **BERT** (Bidirectional Encoder Representations from Transformers), **GPT** (Generative Pre-trained Transformer) and **RoBERTa** (Robustly Optimized BERT Approach), dominate the current state-of-the-art in many text processing tasks, including sentiment classification. These models are highly accurate and are capable of capturing the intricate word associations necessary for successful sentiment analysis.

3 Objectives

With millions of tweets added every day, Twitter is an enormous data source for accessing user's opinions, experiences, and discussions of current issues

and events through messages (tweets). Due to its global reach, it has become a globally preferred platform to perform sentiment analysis, which can be beneficial for companies to understand people's opinions on products, services, organizations, situations, government actions, and decisions.

The main purpose of this paper is to explain all the steps required to create a demo of a simple dashboard, that uses Twitter as a data source, to display the sentiment of any brand or product in real-time.

Sentiment analysis can be done in many ways. There are libraries like *TextBlob*, *NLTK*, and *Vader* that offer convenient pre-built functions for sentiment analysis on pre-processed tweets. However, these methods are quite unchallenging and boring, and the output might not be particularly accurate for brand sentiment analysis.

To overcome this limitation, the use of state-of-the-art models is essential. By fine-tuning pre-trained models for analyzing Twitter data, we can obtain accurate sentiment results. Therefore, businesses can gain valuable insights of how their brand is being perceived by their customers and the public at large in the present moment, enabling them to make informed decisions and take appropriate actions to maintain a positive brand image.

The most crucial objectives can be defined as:

- Fine-tuning two deep learning models (BERT and RoBERTa) for sentiment analysis using a recollection of well-labelled tweets
- Compare both models to assess its performance and suitability for the task.
- Scrape tweets to analyze the sentiment of the target audience of the brand.
- Perform and visualize sentiment results for a specific brand using a pre-trained deep learning model with real-time data from Twitter.

4 Data recollection

To effectively train a deep learning model, having a sufficiently large, labelled dataset is essential. With this goal in mind, I opted to merge multiple datasets related to Twitter users' opinions on brands or products, resulting in a vast compilation of labelled tweets. Some of the datasets used are about Apple products, and others well-known ones are **Sentiment140** [4] and **SemEval** (Semantic Evaluation) [5], the latter being made available from previous task competitions.

The next step involves conducting some exploratory analysis to gain a deeper insight into our data. Through this exploratory analysis, I examined those tweets that surpassed the prior Twitter character limit of 280 characters, as these datasets were created before 2022, as well as shorter tweets. This was done to evaluate the quality of the tweets and exclude those that lacked meaningful sentiment content from the analysis. In addition, I utilized visualizations to analyze the most common words in the dataset, which offered me valuable insights into typical sentiment words (sorry, love, thank, hate, etc), the informal language and slang (lol, xd, xoxo, etc.) and themes present in the data.

During the analysis, I discovered a class imbalance issue, primarily related to neutral instances. Addressing this class imbalance is important since it can significantly affect the performance and accuracy of the classification model. The imbalance can lead to biasing the model towards the majority class, thereby disregarding the minority class. Therefore, I decided to perform data augmentation. There are various methods for augmenting data, such as oversampling, undersampling, using synthetic data generation methods such as SMOTE (Synthetic Minority Oversampling Technique) [6] or doing back-translations [7].

Among these techniques, I have made the decision to use **back-translations** to increase the amount of neutral instances. Essentially, this method involves translating the tweet from one language to another and then back to the original language. Through this process, new text can be generated that conveys the same meaning as the original, but with different wording. This can effectively increase the size of the dataset and potentially improve the model's performance. Finally, after some rounds of back-translations, I was able to obtain around 160k instances of neutral tweets.

5 BERT and RoBERTa models

BERT and RoBERTa are two of the most widely used transformer-based models for natural language processing tasks. These models utilize self-supervised learning and are pre-trained on large amounts of unlabelled text data to understand the context and deep meaning of words.

5.1 Architectures

BERT and RoBERTa are built upon the **transformer architecture**. Specifically, they consist of a stack of identical Transformer encoder layers, where each

one has a multi-head self-attention mechanism and a fully connected feed-forward network. These layers allow to capture the contextual relationships between words in a sentence effectively. When considering the architectures of the base models, they consist of 12 layers of encoders, a hidden size of 768, 12 attention heads and 110M parameters. For a better understanding, we can assume both pre-trained models are black boxes that produce 768-dimensional vectors for each token in a sequence [8].

5.2 Pre-training process

One of the main reasons for the good performance of BERT on different NLP tasks is the use of **semi-supervised learning**. This means the model is trained for a specific task that enables it to learn and recognize patterns in the language. As a result, the trained model has language-processing capabilities that can be used to enhance other models that require training in a supervised way [9].

The BERT pre-training was accomplished using a **masked language model** and **next sentence prediction**. The masked language model randomly masks words in the input and asks the model to predict the missing word. This helps BERT learn context from both the left and right sides of the masked token, adding bidirectionality to the model. The next sentence prediction is basically predicting the likelihood of a sentence Y being the consecutive to sentence X. This task is included to improve the model's understanding of relationships between sentences.

Researchers at Facebook revisited the pre-training process that was used to develop BERT and identified several areas that could be improved. Their efforts resulted in the development of a "robustly optimized" version of BERT that they named RoBERTa [10]. One notable distinction between the two approaches lies in their masking strategies. BERT's uses a static language masking strategy, while RoBERTa introduces diversity into the training process by adopting a dynamic masking approach. In RoBERTa, every single example when it is presented to the model is masked in a potentially different way, via some random function.

RoBERTa also makes modifications to important hyperparameters used in BERT. These changes include eliminating BERT's next-sentence pre-training objective, training with larger mini-batches and higher learning rates. By doing so, it enhances the masked language modelling objective compared to BERT, resulting in improved performance on downstream tasks. Additionally, RoBERTa explores the use of significantly more data and a longer training duration, surpassing BERT in both scale and training time [10].

5.3 Benefits of using pre-trained models

After trying to train an LSTM model and obtaining unsatisfactory results, I decided to rely on state-of-the-art models. Specifically, I opted to fine-tune BERT and RoBERTa models.

Fine-tuning is a process that takes a pre-trained model, which has already been trained for a specific task, and tweaking it to perform a second similar task. This approach allows us to take advantage of the knowledge and capabilities that the model has already acquired, without having to develop the learning process from scratch. Fine-tuning can be applied to the entire neural network or selectively to certain layers. When fine-tuning specific layers, the remaining ones are "frozen", meaning they are not updated during the backpropagation step.

Using these pre-trained models with the fine-tuning approach, offers numerous advantages in terms of results and time efficiency. Foremost, these models' weights already encode a lot of information about our language. Consequently, training our fine-tuned model requires significantly less time compared to training from scratch [11]. It is like having the bottom layers of our network extensively trained, and now we only need to gently tune them while using their output as features for our classification task. For this reason, the authors of BERT recommend only 2-4 epochs of training for fine-tuning on a specific NLP task. In contrast to the hundreds of GPU hours that would be needed to train the original BERT model or a LSTM from scratch.

6 BERT and RoBERTa for sentiment classification

In the following section, we will outline the step-by-step process for utilizing BERT and RoBERTa for sentiment classification tasks.

6.1 Preprocessing

The pre-processing of Twitter data is an important step because it can improve the performance of the model by reducing noise and making the text data more consistent. Tweets usually contain elements such as URLs, mentions and hashtags that may not

be relevant to the sentiment analysis task and can be removed. Other common techniques include lowercasing, punctuation removal, replacing emojis, expanding contractions, stop words removal, lemmatization, etc. Nevertheless, it is important to highlight that the required pre-processing may vary depending on the model utilized.

When fine-tuning BERT and RoBERTa models for sentiment analysis, applying a minimal preprocessing is sufficient. It is unnecessary to perform all the text transformation steps because BERT and RoBERTa were trained on complete sentences. Basic transformations such as removing HTML tags, URLs, mentions, hashtags, expanding contractions, decoding emojis, converting to lowercase, removing some special characters, and applying a simple spell correction are enough for my purposes.

6.2 Tokenization

In order to perform sentiment analysis using BERT and RoBERTa models, the text sentences need to be tokenized. This involves splitting the sentences into tokens and mapping them to their corresponding indices in the tokenizer vocabulary. This process can be performed by the tokenizer included in both models. The output of these tokenizers, which serves as the input features for the models, consists of **token IDs**, which are integer representations of the input tokens, and an **attention mask**. The attention mask is a sequence of ones and zeroes that provides information to the model about which tokens come from the input sentence (real tokens) and which ones are padding tokens. Token type IDs can also be generated, particularly useful in question-answering tasks to identify which token belongs to which segment. Furthermore, the special tokens that can be incorporated into the model are the following [12]:

- **[CLS]**: Token at the beginning of a sentence, that references a classification task.
- **[SEP]**: Token for the endings of sentences.
- **[PAD]**: Token for padding.
- **[UNK]**: Any unknown token not appearing in its vocabulary.

It is important to note that BERT and RoBERTa have constraints regarding input length. To ensure efficient backpropagation during training, all inputs to the model must have the same length. Therefore, sentences need to be padded or truncated to a fixed length, with a maximum of 512 tokens. After this tokenization, these inputs are fed into the model to generate token embeddings. This process involves indexing a matrix of size vocabulary-size x 768 (H), where the weights of this matrix would be learned while training.

One crucial distinction between BERT and RoBERTa generation of word embeddings compared to other advanced techniques like *Word2Vec* is their consideration of contextual information. While *Word2Vec* successfully captures the semantic meaning of words, it does not account for the context in which words are used. For instance, *Word2Vec* would assign the same embedding vector to the same two words with different meanings. In contrast, BERT and RoBERTa utilize contextual information and would generate different embedding vectors based on its surrounding words.

6.3 Fine-tuning

To perform the fine-tuning, we must load the pre-trained model weights, and then insert an additional layer on top of the pre-trained model to convert the output to our specific task. In this case, since we are performing sentiment classification, we will be inserting a fully-connected layer on top of its output layer that is configured to output the probabilities of the classes we are trying to predict. Moreover, a dropout layer will also be added to improve generalization. The Figure 1 shows an example of how BERT is used for text-classification:

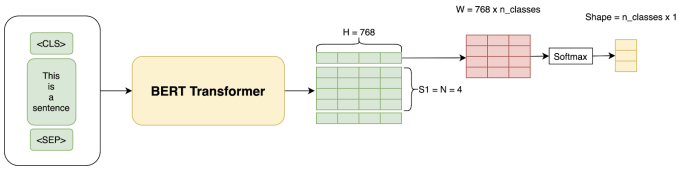

Fig. 1: BERT fine-tuning process.

6.4 Hypertuning and LRFinder

BERT and RoBERTa are pre-trained models that have already undergone extensive hyperparameter tuning during their pre-training phase. As a result, when fine-tuning these models for a specific task, there might not be a need for an extensive hyperparameter tuning. However, it can still be beneficial to adjust important parameters such as batch size, learning rate, and dropout rate. The recommended hyperparameters suggested by the authors were taken into consideration when fine-tuning the models [13]:

- **Batch size**: 16 or 32
- **Learning rate (Adam)**: 5e-5, 3e-5 or 2e-5

- **Number of epochs**: 2, 3, 4

One way to do this adjustment is by using a hyperparameter optimization library like **Optuna** [14]. Optuna is a powerful framework for defining and optimizing hyperparameters in machine learning models. It allows you to define a search space for your hyperparameters and uses Bayesian optimization to find the optimal values. Consequently, I have developed an implementation of Optuna to facilitate this process.

However, the hyperparameter tuning was very time-consuming, especially since it had to deal with large amounts of data. In such cases, it may be more practical to focus on adjusting the most important hyperparameters. For example, the learning rate, which can have a significant impact on the model performance.

The learning rate plays a vital role in determining the weight adjustments of our network in response to the loss gradient [15]. It measures how much a model can "learn" from a new mini-batch of training data, meaning how much we update the model weights with the information coming from each new mini-batch. The higher the learning rate, the bigger the steps we take along the trajectory to the minimum of the loss function, which can cause overshooting. Whereas the lowest the learning rate, the tiniest the steps, which can take a long time to reach the minimum and result in minimal improvement, also known as *loss plateau*. Between these two extremes, there exists a good learning rate. Hence, it is an important hyperparameter.

One way to find the optimal learning rate is by using a **Learning Rate Finder** [15, 17], which helps you find the best value by training the model with different learning rates and observing the loss. This idea comes from a paper called *"Cyclical Learning Rates for Training Neural Networks"* by Leslie Smith in 2015 [18], where she introduces the concept of **cyclical learning rate**, that involves increasing and decreasing in turns the learning rate during training. One important thing covered in the paper is the **LR Range test**, which consists of running a short training session in which the learning rate is increased (linearly) between two boundary values *min-lr* and *max-lr*. Initially, with a small learning rate the network will gradually converge, leading to a progressive reduction in the loss values. At some point, the learning rate will get too large and cause the network to diverge.

The Figure 2 shows the LRFinder plot, that displays loss values versus tested learning rates. Usually, the best value is somewhere around the middle of the steepest descending loss curve. However, in this case, the red dot of the graphic, which indicates the optimal value chosen by PyTorch, is at the beginning. This makes sense, considering a paper called *"How to Fine-Tune BERT for Text Classification"* [19], in which it is stated that lower learning rates, such as 2e-5, are necessary to make BERT overcome the catastrophic forgetting problem. This issue entails that the pre-trained knowledge is erased during the learning of new knowledge, primarily because of aggressive learning rates. Consequently, the model does not converge properly.

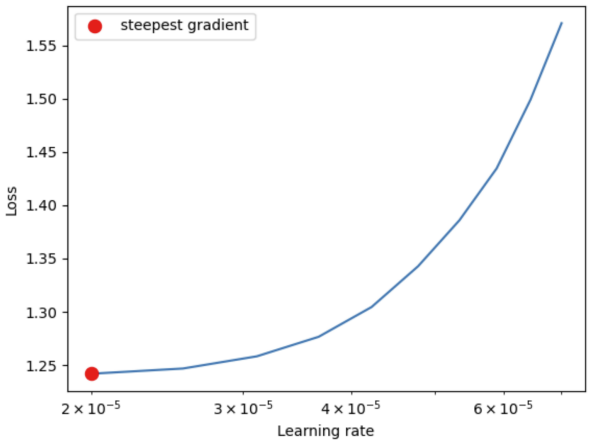

Fig. 2: Learning rate suggested for BERT.

Regarding the padding length, I plotted a histogram based on the number of tokens in tweets to analyze its distribution. Most of the tweets contain less than 100 tokens. But I opted for a maximum length of 128 to be on the safe side.

The table 1 displays the selected hyperparameters for each model:

Model	Epochs	Batch size	Lr	Dropout
BERT	3	16	2e-5	0.5
RoBERTa	3	32	2e-5	0.5

Table 1: Model hyperparameters: BERT vs. RoBERTa

As we can visualize, I selected a relatively high dropout rate, which is effective in mitigating overfitting issues in large and complex models or when dealing with noisy and diverse datasets, similar to the one I have. Additionally, during the Optuna hyperparameter tuning process, I experimented with three distinct dropout rates (0.1, 0.3, 0.5), and it was determined that 0.5 yielded superior results.

6.5 RESULTS

The sentiment analysis dataset used in this fine-tuning consists of a total of 1.764.293 instances. The Table 2 shows the number of instances for each class, as we can see there's an imbalanced class issue, even though we have performed data augmentation of neutral data:

	Number of instances
Positive	804.413
Negative	792.878
Neutral	167.002

Table 2: Count of instances for each class.

To train and evaluate the model, the dataset is divided into training, validation, and testing sets. The training set contains 90% of the instances, which corresponds to 1,587,863 instances. While the validation and testing sets each contain 5% of the instances, which amounts to 88,215 instances for each set.

To evaluate how well each model performed, we will look at a summary of performance metrics created using the classification report from *sklearn.metrics*. This summary includes the precision, recall, and F1-score [20] for each predicted class as well as the overall accuracy of the classifier. The evaluation results of both models are displayed in the Table 3:

	Precision	Recall	F1-Score	Instances
BERT				
Positive	0.86	0.85	0.85	40173
Negative	0.86	0.86	0.86	39585
Neutral	0.84	0.83	0.84	8457
Average	0.85	0.85	0.85	88215
Accuracy			0.86	88215
RoBERTa				
Positive	0.87	0.86	0.86	40173
Negative	0.86	0.88	0.87	39585
Neutral	0.84	0.80	0.82	8457
Average	0.86	0.85	0.85	88215
Accuracy			0.86	88215

Table 3: Comparison of BERT and RoBERTa Models for Sentiment Analysis

When evaluating the performance of BERT and RoBERTa on the classification task, both models yield similar results overall. The classification report shows that RoBERTa achieves slightly better results compared to BERT in certain metrics, except when handling neutral data. However, it's important to note that these improvements are minimal. The overall similarity in results between BERT and RoBERTa indicates that both models are capable of capturing and understanding the underlying patterns in the data, as well as making accurate predictions.

Another explored approach, involved freezing the majority of the network and re-training (adjusting the weights) only specific layers (last two layers of BERT, the pooler layer, and the fully-connected layer). However, the results from this partial unfreezing (displayed in the Table 4) were not as favourable as when the entire model was fine-tuned. Despite the fact that initial layers capture general characteristics, while later layers capture domain-specific information. This indicates that retaining the pre-trained knowledge and enabling the whole model to adapt to the specific task leads to a better performance compared to selectively unfreezing the last layers.

Unfreezing BERT last 2 layers + pooler + classifier				
	Precision	Recall	F1-Score	Instances
Positive	0.84	0.82	0.83	40173
Negative	0.82	0.86	0.84	39585
Neutral	0.82	0.72	0.77	8457
Average	0.83	0.80	0.81	88215
Accuracy			0.83	88215

Table 4: Results of freezing the whole BERT, except certain layers.

7 DASHBOARD DEMO

To build a dashboard demo to display sentiment results, there are three necessary steps: scrape tweets with Twitter API, find a suitable way to store results, and utilize an easy-to-use interface like Streamlit to load the model and display the sentiment results. Each step is thoroughly detailed in the subsections that follow.

7.1 TWITTER API FOR SCRAPING TWEETS

Discovering a method to fetch and store tweets is a crucial step towards building an interface that displays the results of our sentiment analysis model.

Researchers have extensively relied on the free access to Twitter's API to gather valuable data and gain insights into the conversations taking place on the platform. This access has been essential for comprehending the trending topics and discussions within the online community. Unfortunately, recent updates to Twitter's policies have resulted in substantial pricing changes, making the API access unaffordable for many organizations that have heavily relied on it for conducting their research [21].

The new API access tiers that Twitter has launched are the following [22]:

- **Free**: For write-only use cases and testing the Twitter API.
- **Basic**: Costs $100 per month and the read limit is 10,000 tweets a month.
- **Pro**: Costs $5000 per month, and includes the search and filtered stream, as well as the ability to fetch 1,000,000 tweets a month.
- **Enterprise**: Costs $42,000 per month, and gives access to 50 million tweets a month.

The enterprise plan is incredibly expensive for companies and start-ups in their early stages. For these newly established businesses, the most viable option would be to rely on the Pro tier. However, even with the Pro tier, priced at an equivalent of $60,000 per year, it is unlikely to attract many customers as the cost may still exceed their budget.

Additionally, the recently introduced costs are not feasible for the academic community and many Twitter-based products, leading to their potential shutdown. This will significantly eliminate any opportunities for conducting research within this domain.

Despite all this, I have decided to continue using the Twitter API, utilizing a simple object called *Cursor* [23] to retrieve tweets. While this *Cursor* object can retrieve a collection of tweets matching a specific query, it comes with the inconvenience of limited requests, allowing only 180 requests every 15 minutes [24]. A potential future optimization could involve investing in a more advanced API to gain access to a greater volume of data. However, currently, this endeavour appears to be quite expensive, and the more practical approach seems to be opting for a Pro subscription.

7.2 SQLite3 to store results

Once I finished working with the Twitter API, I needed to select a proper and manageable database to store the results for further analysis. After investigating, **SQLite3** [25] seemed to be a good alternative, since it is an easy-to-use module in Python that provides a lightweight disk-based database that does not require a separate server process. With this module, you can create, connect to, and manage SQLite databases in your Python programs. Additionally, you can execute SQL commands to create tables, insert data, query data, and more.

The main process to store a tweet is the following: once a tweet is retrieved with the *Cursor* object, we preprocess it and pass it through our model to calculate its sentiment. This sentiment is then saved along with other relevant metadata, such as the tweet, timestamp, number of likes, retweets, replies, etc., using an *insert* operation on the corresponding table.

It is important to note that the tweets being retrieved pertain to a specific brand. Therefore, when we need to search for a different brand, we execute a *delete* statement to truncate the table, effectively erasing all existing rows. This ensures that any previous data associated with other products is completely removed, to only display the newly entered data that references the input brand.

7.3 Streamlit to display results

Streamlit [26] simplifies the process of building a web app, using just a single Python file, eliminating the need to write complex *HTML*, *CSS*, or *JavaScript* code. With Streamlit, we can create interactive dashboards that provide real-time visualization and monitoring of several measures.

The dashboard I have created offers a thorough analysis of tweet sentiment. In the following paragraphs, we will explore the different components of the dashboard and their respective functionalities.

Firstly, we have a **filter panel**, shown in the Figure 3, which allows us to customize the criteria for fetching tweets. Within this panel, we can set filters such as the brand of interest, the desired number of tweets, minimum reply count, minimum retweet count, minimum like count, specific date range, and the option to exclude retweets or replies. These flexible filters enable users to narrow down the specific set of tweets they want to analyze.

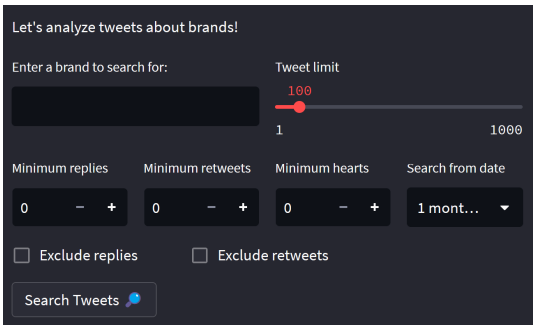

Fig. 3: Filters to fetch tweets.

On the other hand, the dashboard presents some key performance indicators (KPIs) that provide an overview of the sentiment analysis results, shown in the Figure 4. They include the mean polarity score,

which indicates the average sentiment of the analyzed tweets, and the number of tweets categorized as positive, negative, and neutral. These metrics give users a quick understanding of the sentiment distribution in the corresponding tweets.

Furthermore, I have incorporated various visualizations to enhance the dashboard's presentation and insights (shown in Figure 4 and 5):

- **Scatter plot**: it visually represents each tweet on a scatter plot, with the polarity value on the y-axis and the date on the x-axis. This visualization allows users to observe the sentiment of individual tweets and identify any patterns or trends over time.
- **Pie chart**: it provides a clear visualization of the distribution of tweets across the different sentiment classes: positive, negative, and neutral. By presenting the percentage of tweets in each class, users can quickly grasp the overall sentiment balance within the scraped data.
- **Bar chart**: it displays the top terms found in the analyzed tweets. Presenting the most frequent topics or keywords related to the brand or product of interest, can easily help to identify the main themes being discussed. In other words, this graphic enables users to obtain valuable insights regarding the prevailing topics and discussions surrounding the brand or product.

Overall, this Streamlit dashboard offers users a simple and informative interface to analyze sentiment in tweets. Users can gain a solid understanding of the sentiment trends, distribution, and important topics within the scraped Twitter data by combining filters, KPIs, and informative visualizations.

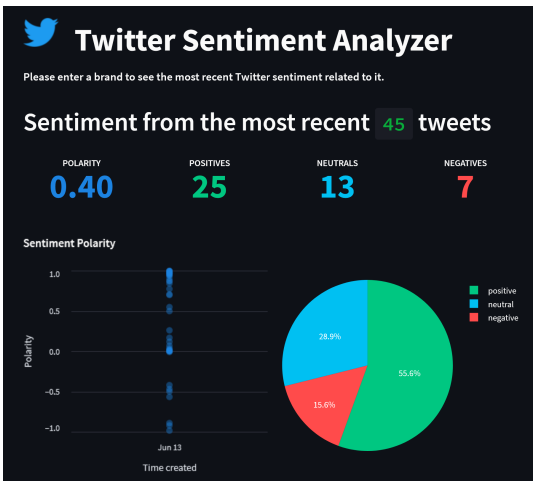

Fig. 4: Metrics and visualizations.

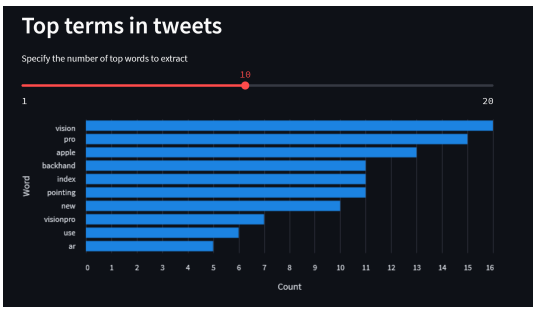

Fig. 5: Visualization of top terms in tweets.

8 Conclusions

Throughout this project, I have had the opportunity to learn a lot about the fascinating field of Natural Language Processing (NLP) and explore state-of-the-art models such as BERT and RoBERTa. This journey of learning has been immensely enriching, and I also had the opportunity to learn from Ph.D. students, who generously shared their valuable insights and knowledge about the subject.

One aspect that amazes me is the remarkable language capabilities showed by BERT and RoBERTa. By simply adding a single fully-connected layer on top of those models, I achieved performance that is nearly state-of-the-art. Moreover, despite the enormous size and complexity of these models, it is quite impressive to know that fine-tuning for just 2–4 epochs yielded such good results.

Additionally, it is worth highlighting that the development of a basic web application, which is capable of analyzing real-time tweets for any given brand, is a useful tool for businesses and researchers to monitor social media trends and sentiment. Furthermore, this web app can be conveniently customized and expanded to incorporate additional features and functionalities, to enhance the analysis task.

Using this tool to analyze social media data and extract insights regarding customer brand perception is immensely valuable. Thereby, businesses can proactively stay ahead of the competition and make well-informed decisions.

References

[1] Shreyashi. "Twitter Sentiment Analysis: A Brief Overview With Real Example." Gramener Blog, 2022, https://blog.gramener.com/twitter-sentiment-analysis/.

[2] Vaswani, A., Shazeer, N., Parmar, N., Uszkoreit, J., Jones, L., Gomez, A. N., Kaiser, L., & Polosukhin, I. Attention is all you need. Advances in neural information processing systems, Volume 30, 12 Jun. 2017, https://arxiv.org/abs/1706.03762.

[3] Alammar, Jay. “The Illustrated Transformer – Jay Alammar – Visualizing Machine Learning One Concept at a Time.” Jay Alammar – Visualizing Machine Learning One Concept at a Time., https://jalammar.github.io/illustrated-transformer/.

[4] “For Academics - sentiment140 - a Twitter Sentiment Analysis Tool.” Sentiment140, http://help.sentiment140.com/for-students.

[5] “Semeval-2017 Task 4.” Sentiment Analysis in Twitter ¡ SemEval-2017 Task 4, https://alt.qcri.org/semeval2017/task4/#.

[6] Jayawardena, Numal. “How to Deal with Imbalanced Data.” Medium, Towards Data Science, 9 July 2020, https://towardsdatascience.com/how-to-deal-with-imbalanced-data-34ab7db9b100.

[7] Keita, Zoumana. “Data Augmentation in NLP Using Back Translation with Marianmt.” Medium, Towards Data Science, 21 Nov. 2022, https://towardsdatascience.com/data-augmentation-in-nlp-using-back-translation-with-marianmt-a8939dfea50a.

[8] Vu, Kevin. “BERT Transformers: How Do They Work? - DZone.” Dzone.Com, DZone, 6 Apr. 2021, https://dzone.com/articles/bert-transformers-how-do-they-work.

[9] Alammar, Jay. “The Illustrated Bert, Elmo, and Co. (How NLP Cracked Transfer Learning).” The Illustrated BERT, ELMo, and Co. (How NLP Cracked Transfer Learning) – Jay Alammar – Visualizing Machine Learning One Concept at a Time., https://jalammar.github.io/illustrated-bert/.

[10] “Roberta: An Optimized Method for Pretraining Self-Supervised NLP Systems.” Meta AI, https://ai.facebook.com/blog/roberta-an-optimized-method-for-pretraining-self-supervised-nlp-systems/.

[11] BERT Fine-Tuning Tutorial with PyTorch, Chris McCormick, 22 July 2019, http://mccormickml.com/2019/07/22/BERT-fine-tuning/.

[12] Valkov, Venelin. “Multi-Label Text Classification with Bert and Pytorch Lightning.” Curiousily, https://curiousily.com/posts/multi-label-text-classification-with-bert-and-pytorch-lightning/. Accessed 16 June 2023.

[13] PySnacks. “A Tutorial on Using BERT for Text Classification w Fine Tuning.” PySnacks, 23 Jan. 2020, https://pysnacks.com/machine-learning/bert-text-classification-with-fine-tuning/.

[14] Nik. “Python Optuna: A Guide to Hyperparameter Optimization • Datagy.” Datagy, 21 Apr. 2023, https://datagy.io/python-optuna/.

[15] Salimath, Ashwath. “How to Use the Learning Rate Finder in TensorFlow — by Ashwath Salimath — Octavian — Medium.” Medium, Octavian, 9 Apr. 2019, https://medium.com/octavian-ai/how-to-use-the-learning-rate-finder-in-tensorflow-126210de9489.

[16] davidtvs. “GitHub - Davidtvs/Pytorch-Lr-Finder: A Learning Rate Range Test Implementation in PyTorch.” GitHub, https://github.com/davidtvs/pytorch-lr-finder.

[17] “Learning Rate Finder - PyTorch Lightning 1.5.10 Documentation.” Lightning AI, lightning.ai/docs/pytorch/1.5.10/advanced/lr_finder.html.

[18] Smith, Leslie N. ”Cyclical learning rates for training neural networks.” 2017 IEEE winter conference on applications of computer vision (WACV). IEEE, 2017, https://arxiv.org/abs/1506.01186.

[19] Sun, Chi, et al. ”How to fine-tune bert for text classification?.” Chinese Computational Linguistics: 18th China National Conference, CCL 2019, Kunming, China, October 18–20, 2019, Proceedings 18. Springer International Publishing, 2019, https://arxiv.org/abs/1905.05583.

[20] Harikrishnan. “Confusion Matrix, Accuracy, Precision, Recall, F1 Score — by Harikrishnan N B — Analytics Vidhya — Medium.” Medium, Analytics Vidhya, 10 Dec. 2019, https://medium.com/analytics-vidhya/confusion-matrix-accuracy-precision-recall-f1-score-ade299cf63cd.

[21] Stokel-Walker, Chris. “Twitter’s $42,000-per-Month API Prices out Nearly Everyone.” Wired, 10 Mar. 2023, https://www.wired.com/story/twitter-data-api-prices-out-nearly-everyone/.

[22] “Twitter API - Products - Twitter Developer Platform.” Twitter, developer.twitter.com/en/products/twitter-api.

[23] Raj, Piyush. “Get Data from Twitter API in Python - Step by Step Guide.” Data Science Parichay, 15 Dec. 2021, https://datascienceparichay.com/article/get-data-from-twitter-api-in-python-step-by-step-guide/.

[24] “Rate Limits — Docs — Twitter Developer Platform.” Twitter, https://developer.twitter.com/en/docs/twitter-api/rate-limits.

[25] DavidMuller. “How To Use the Sqlite3 Module in Python 3 — DigitalOcean.” DigitalOcean — The Cloud for Builders, DigitalOcean, 2 June 2020, https://www.digitalocean.com/community/tutorials/how-to-use-the-sqlite3-module-in-python-3.

[26] RS, AbdulMajedRaja. “How to Build a Real-Time Live Dashboard with Streamlit.” Streamlit, 30 Nov. 2022, https://blog.streamlit.io/how-to-build-a-real-time-live-dashboard-with-streamlit/.