AI Sentiment analysis documentation

Filip Šarík 243537, **Razvan Nica 242715**:
Model selection, testing, training, feature engineering, hyper-tuning, model deployment
Erik Hriňa 230395:
Model testing, frontend development, backend development, app design, app testing

The goal of this project was to create a model for sentiment analysis and additionally create its app. In the first meeting, we divided our work and responsibilities, deciding on how our model should work and what it should predict. We all agreed on a multiclass prediction model, more specifically for **three classes** (positive, neutral, negative). With this, we started to look for a **suitable dataset** for this project and found one that fulfills all these requirements.

Model Development

The first stage of this project was selecting models for testing and then evaluating each of them. Before that, we checked the metadata of our dataset first. It contained 4 columns with strings. We looked into the **distribution of target classes** which proved to be mostly **balanced**. Then we checked for missing values and found out that only one row had a missing text. There was no point in keeping it, so we dropped it.

In the next stage, we researched what is required for these models to process the text data. We read articles, watched videos, and decided on our approach. First, we imported all the necessary libraries for the transformation of the text. We downloaded a list of **stopwords** in **English** from **NLTK**. Stopwords are words that don't have any emotional value, so it is common practice to remove them from the strings. Some examples of these words are: 'i', 'me', 'my', 'myself', 'we', 'our', etc.

Then, we **replaced** the sentiment results of the target class with **numerical values** (0:neutral, 1: positive, -1: negative). The next step in this stage was "**stemming**", which means getting the **root** of a word, for example, words like "responded" or "actress" get transformed into "respond" and "act".To achieve this, we used **PortStemmer** and used it as a function for cleaning the text. This **function** first removes all the characters, except for letters, then it transforms all the letters to lowercase, splits these words into tokens, applies stemming, and then joins these tokens back into a string. We applied this function to the text column of our dataset and then dropped the unnecessary columns.

The third stage was preparing the data for the machine learning models. This stage included **splitting the data** and transforming the text tokens into numbers that the model could handle. Then, we tried using different classification models to see which one performed best.

The first model we tested was **Logistic Regression**. With only a few parameters set, the training accuracy of this model was **73%**, and the accuracy of the **test data** was **69%**. The next model we tested was a **MultinomialNB classifier** for which we didn't have to set any parameters to get accuracies of **72%** on the training set and **63%** on the **test data**. Next, we tested an **XGBClassifier** with a **few parameters** tuned at random. The model achieved a **71%** accuracy on the **testing set**.

We also tried implementing a **neural network** classification model. The accuracy of the NN model obtained for the testing data was about **62%**, which was significantly lower

compared to the **XGB Classification Model**. Between these four models, we have chosen the one with the best performance on the testing data, the **XGB Classifier**.

Furthermore, we also tried implementing a **pre-trained model** to compare it to ours. We chose the **RoBERTa model**[1] developed by **HuggingFace**. Since the model was pre-trained, there was no need to split the data into testing and training sets. Instead, we used all the data to **evaluate** the model's **performance**. The accuracy score for the entire dataset was around **70%** which was still **lower** than the **XGB Classification Model** we implemented.

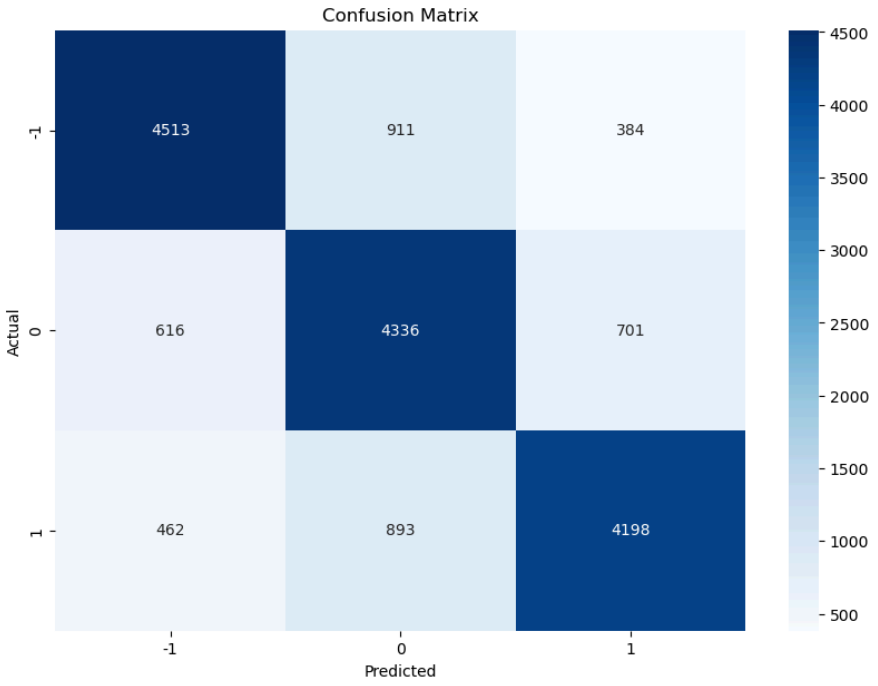
The fourth stage consisted of trying to **improve** this model and combining all the **preprocessing** steps into a **pipeline**.

First, we have found **two additional datasets** that were suitable. This has **increased** the number of **data** we can work with. We went from **28,000** to over **100,000**, and after some preprocessing, like **dropping duplicates** and **missing values**, this went down to **85,000**. Before combining all these data frames into one, we **replaced** the sentiment values with **numerical** ones and **dropped** unnecessary **columns**. There was also another class called "Irrelevant" so we also had to remove instances of that class.

Eventually, we ended up with a **balanced** dataset of around **28,000** records from each **class**. Next, we started creating the **pipeline**. We imported and downloaded everything that was needed, and defined the function for preprocessing the text, however, this time using a **different tokenizing** method from the **NLTK** library. We also implemented a function that would transform **emojis** into text, which could have a big impact on the predictions. We again defined the target and features, split the dataset, and **encoded target variables** due to our previous experiments that showed **XGBClassifier** having **issues** with a negative value as a class, so we **mapped** them to positive: 2, neutral: 1, and negative: 0.

Finally, we defined the pipeline with **TfidfVectorizer** and the model. The entire preprocessing function is a **parameter** inside the **vectorizer**, and we increased the number of **n_estimators** of the XGBClassifier from 500 to 600, which slightly increased the model's performance.

We retrained the model, evaluated it, and printed the accuracy scores. The model had **82%** accuracy on training data and **75%** accuracy on testing data. The last change we made was to **increase** n_estimators to **800**, and this further improved the model's accuracy to **84%** on training data and **76%** on test set. With this satisfactory model, we **exported** the pipeline and checked the **confusion matrix** on how it is doing for all of the classes and it only had difficulty with distinguishing between the **neutral** class and the other two, which can be sometimes confusing even for us humans to do. We had a spare dataset for **validation**, so we decided to use it on this model and the result was an **accuracy** of **90%**. Due to the limited time frame, we all agreed that this model could be used in the application.



Application Development

Frontend

The app's frontend was built with **Flutter**. Using **Dart**, I created the pages, widgets, and logic to make the app work. The app has two pages: **Home** and **Results**.

1. **Home Page:**
    - Users can enter text into a text field (maximum 500 characters).
    - The app requires at least one character to proceed, as testing showed the model can handle very short inputs.
    - When the user clicks the "Send" button, they are taken to the Results page.
2. **Results Page:**
    - Shows the user's text, the detected sentiment, and a pie chart showing the confidence for each possible result.
    - If something goes wrong (e.g., server errors or timeouts), an error message is shown and the user stays on the Home page.

Backend

The backend was built using **Python's Flask**. It has one endpoint (*/predict*) that handles POST requests with the user's text. The backend:

- Processes the text using a machine learning pipeline to predict whether the sentiment is negative (0), neutral (1), or positive (2).
- Calculates the confidence for each category and sends a response with:
    - **prediction:** the sentiment (as a number).
    - **probabilities:** confidence levels (as a list).
- If there's an issue, it sends a 400 status code with an error message.

Demo

We made a video[2] showing the app working on a local machine. This app keeps things simple and clear while providing meaningful sentiment analysis for the user.

References

[1] RoBERTa: https://huggingface.co/docs/transformers/model_doc/roberta
[2] App Demonstration: https://github.com/ErikHrina230395/sentiment-analysis/blob/main/Documentation/app-demo.mp4
[3] GitHub Repository: https://github.com/ErikHrina230395/sentiment-analysis
[4] Dataset 1: https://www.kaggle.com/datasets/yasserh/twitter-tweets-sentiment-dataset
[5] Dataset 2: https://www.kaggle.com/datasets/jp797498e/twitter-entity-sentiment-analysis
[6] Dataset 3: https://www.kaggle.com/datasets/mdismielhossenabir/sentiment-analysis