Twitter Sentiment Analysis

by YorkU Students

CSML1000 Machine Learning in Business Context - Blended Live Online Fall 2024

Project #3

8th November, 2024

Arslan Abakarov
Daniel (Jee Hwan) Lee
Karen Krucik, MBA(Oxon)
Aayush Bhatt
Mathi Mahalinga

Abstract

Twitter allows users to share thoughts and emotions instantly, making it a prime source for public sentiment on topics like politics, brands, and social issues. However, analyzing sentiment is challenging due to its informal language, with slang, abbreviations, and emojis, complicating traditional analysis methods. To address this, we have developed a web application to classify tweets. This automated tool provides insights into public attitudes and helps track trends and assess responses.

Background

A team from York University was tasked with developing a sentiment model that analyzes the tweets to categorize as positive, negative, neutral, or irrelevant. The model aimed to process large volumes of tweet data efficiently, using machine learning algorithms. By building this model, the team seeks to provide a valuable tool for monitoring social trends and assessing brand perception.

Objective

This project focuses on creating a sentiment model for integration with existing platforms. By analyzing large volumes of tweet data, the team identifies which tweets are positive, negative, or neutral, providing clear insights into public opinion. With this data trained into the system, the application can receive any feedback commentary, analyse and determine whether the feedback is neutral, positive, negative or irrelevant. Thus, the application is a multipurpose feedback analysis tool useful for social media data analytics, product feedback commentary, facebook comment scraping etc.

Further, it has been designed for easy adaptation and can be implemented across various platforms, equipping businesses, researchers, and organizations with a valuable tool for monitoring sentiment trends and responding effectively to changes in public attitudes.

Data Analysis

The dataset initially begins with 2 different datasets: a training dataset and a validation dataset. These were then used to build the sentiment prediction model. It comprises information from 74,682 tweet data, along with their sentiment results. Further, it includes 3 key features that serve as predictors for its sentiment value. Sourced originally from Kaggle[1], the extensive dataset provides essential and useful information, making it highly suitable for developing highly accurate and effective predictive models.

Description of features:

Column Name	Description
Tweet ID	A unique identifier for each tweet
entity	The main subject of the tweet, such as a person, brand, or topic
text	The actual content of the tweet
sentiment	The sentiment label for the tweet's content, often classified as *positive*, *negative*, *neutral, or irrelevant*

[1] https://www.kaggle.com/datasets/jp797498e/twitter-entity-sentiment-analysis/

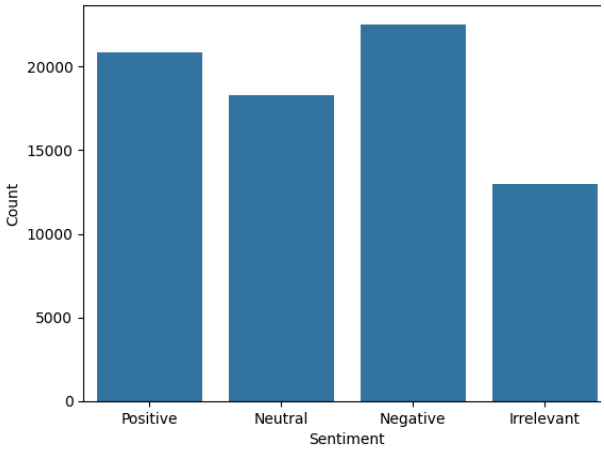
Data Exploration

The initial exploration of the dataset focuses on identifying trends and correlations between features. Key steps include analyzing the data for missing values, ensuring the dataset is clean, well-structured and ready for model development.

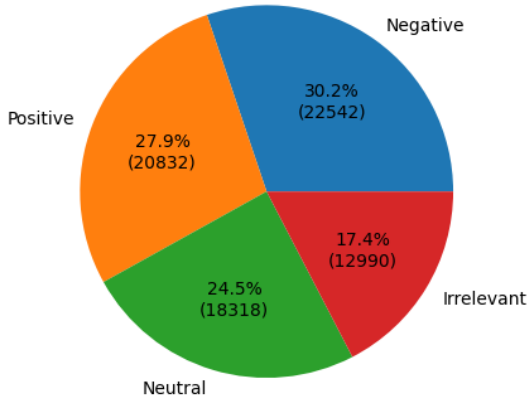
The following lists statistical analysis of the data set and assists in a high level understanding of the distribution of values:

```
Missing values in each column:
Id            0
Company       0
Sentiment     0
Tweet         0
tokenized     0
dtype: int64
Sentiment distribution:
Sentiment
Negative      22542
Positive      20832
Neutral       18318
Irrelevant    12990
Name: count, dtype: int64
Number of missing tweets: 0
```

We also graphed the distribution of the various sentiment counts providing a quick histogram of their various outputs. The following graphs indicate the representation of features across the data set. In this exploration, we cannot suggest strong correlation between each feature:

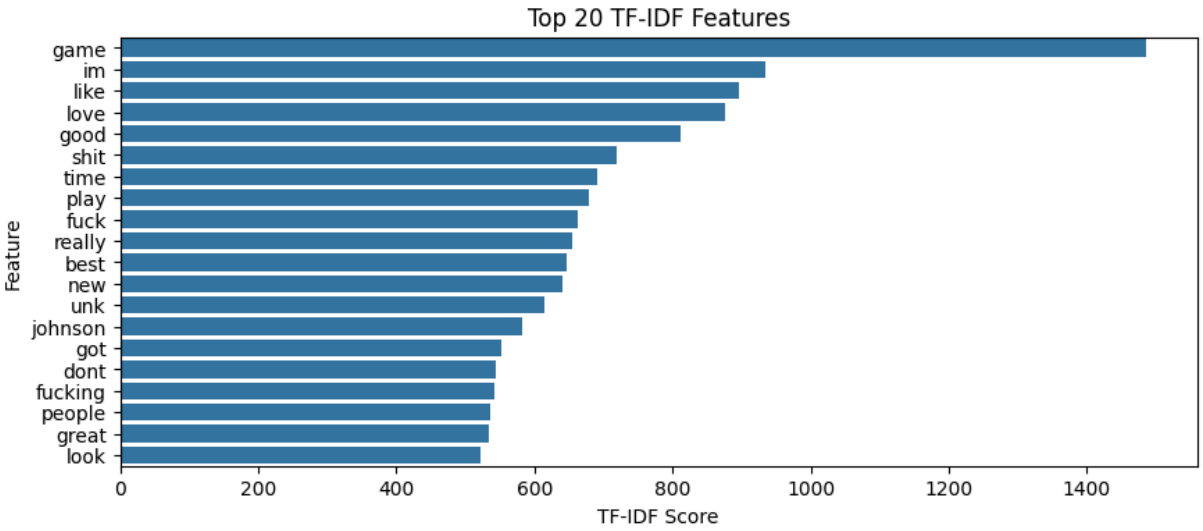


Class Distribution of Sentiment



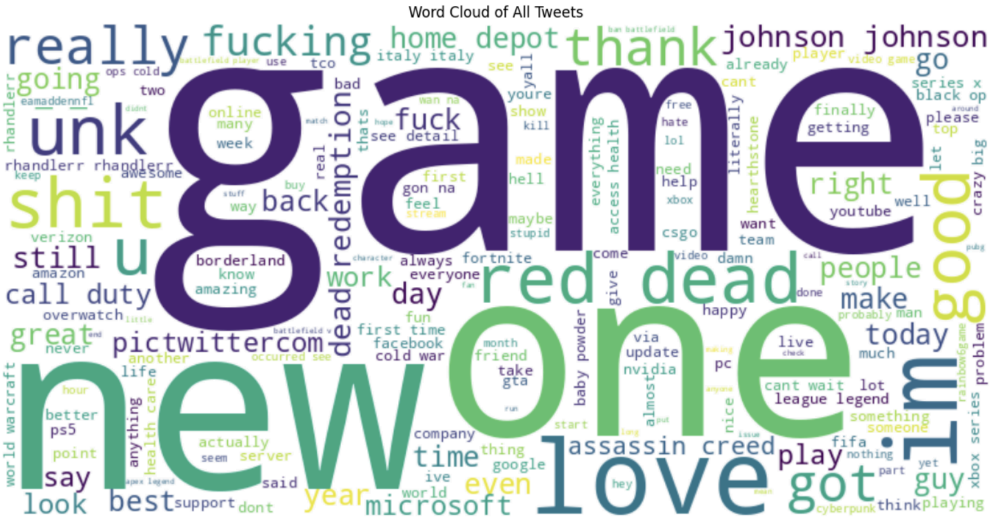
Pie Chart of Sentiment

The above histogram (left) indicates the representation of features across the data set. In this first diagram, we can see that there is not a large difference between the sentiment counts with the largest count being only approximately 1.7x the smallest count.



Top 20 TF-IDF Features

In the pie chart (right), we can further see this data represented by percentage and with the graph just above, we can see the highest frequency words in this data set.



Data Preparation and Feature Engineering

Feature engineering helps to enhance the quality and relevance of the data, which ultimately leads to better-performing models. For this diabetes prediction model, the goal is to achieve high predictability on unseen data and effective feature engineering plays a key role in that.

Key steps in this process include:

- Checking for missing values
- Data cleaning and tokenization
- Text Normalization
- Data correlation and observations

Checking for missing values

To ensure the quality of the dataset, the **isnull()** function from the Pandas library will be used to check for missing values. Missing or empty values can significantly reduce the model's performance by introducing inaccuracies and bias in predictions.

We checked for null values by summing the number of nulls in each feature. This produced the following outcome:

```
# Check for missing values in 'Tweet' column
missing_tweets = train_data['Tweet'].isnull().sum()
print(f"Number of missing tweets: {missing_tweets}")
```

```
Missing values in each column:
Id              0
Company         0
Sentiment       0
Tweet         686
dtype: int64
```

Checking Tweets for Missing Values

During data exploration, we noted that there were no missing values for most features. However, as may be seen above, there are 686 entries (tweets) which had null values. So, due to the small number relative to the dataset size, we just removed these.

Data Cleaning and Tokenization

Beyond just removing null or missing values, we also cleaned the tweet data as follows:

- Removed URLs
- Removed mentions
- Removed hashtags
- Removed punctuation
- Converted it to lowercase
- Split into individual words (tokens)

And returned these token values back to the called function.

```
# Function to clean and tokenize text
def clean_text(text):
    # Remove URLs, mentions, hashtags, and punctuation, and convert to lowercase
    text = re.sub(r'http\S+|www\S+|https\S+', '', text, flags=re.MULTILINE)  # Remove URLs
    text = re.sub(r'@\w+', '', text)  # Remove mentions
    text = re.sub(r'#\w+', '', text)  # Remove hashtags
    text = re.sub(r'[^\w\s]', '', text)  # Remove punctuations
    text = text.lower()  # Convert to lowercase
    return word_tokenize(text)
```

StopWord Removal

In Natural Language Processing (NLP), **stopwords** are common words that typically do not carry significant meaning and are often filtered out during text processing. These words are frequent in a language and are usually removed to focus on more meaningful or content-rich terms. Examples of stopwords in English include words like *"the," "is," "in," "on," "and," "it,"* etc.

By removing stopwords, NLP algorithms can process text more efficiently, as these words do not usually contribute to the overall understanding or classification of the text. However, in some cases, stopwords may still carry useful information depending on the task, so they are not always removed automatically.

As part of our efforts we used this process.

Key points about stopwords:

- **Common words**: They occur frequently but contribute little to the meaning of a sentence.
- **Filtered for efficiency**: Removing them reduces data size and improves algorithm performance.
- **Context-specific**: Stopwords vary by language and context, and some tasks may require retaining them.

Lemmatization

Lemmatization in Natural Language Processing (NLP) is the process of reducing words to their base or root form, called a **lemma**, which represents the canonical form of the word. Unlike **stemming**, which simply chops off word endings, lemmatization uses linguistic knowledge about the word's meaning and context to transform it into its proper base form.

Lemmatization, whilst more computationally demanding, produces more accurate outcomes than stemming (an alternative), so we chose to use this.

For example:

- The words *"running," "ran,"* and *"runs"* are all reduced to the lemma *"run."*
- The word *"better"* would be lemmatized to *"good"* (based on context and meaning), whereas stemming might not capture this relationship.

Feature Aggregation

TF-IDF Vectorizer

The TF-IDF Vectorizer (Term Frequency-Inverse Document Frequency) is a method for converting text into numerical features based on how important the words are in the context relative to the large collection of documents. It is especially useful for text-based tasks like sentiment analysis as it focuses on the key words, reduces noise, and handles short text well. In essence, the technique is a powerful and efficient way to convert the tweets into features that emphasize sentiment-relevant words. After performing this step, we produced the following outcome:

```
Training TF-IDF shape: (74682, 5000)
Validation TF-IDF shape: (1000, 5000)
```

means that the TF-IDF matrix has been created for both the training and validation datasets with the following details: 74,682 and 1,000 tweets were in the data for training and validation respectively. In addition, 5000 is the maximum number of unique words or terms, based on the parameter max_features=5000 in the TF-IDF Vectorizer.

Model Training and Analysis

Create Model

As the next step, we further prepare the data for machine learning by encoding the “sentiment” column, converting text or categorical sentiment values into numeric labels. Due to the requirement that data be 1 or zero, data with the -1 is removed and produces the ‘sentiment encoded’ column which is now ready for model training, with labeled and unlabeled data ultimately represented by 0,1, and -1.

Training The Model

Logistic Regression

For the next step, training the model we used Logistic Regression, solver=”saga” as without this type of solver, the data computation was found to be too heavy on its RAM demands with the system crashing. Further as the data was not a mix of labeled and unlabeled we needed to use a semi-supervised labelling approach.

SelfTrainingClassifier enables the iterative labelling of data. The Base Classifier was Logistic Regression. This has a maximum of 1000 classifications, and as mentioned previously, the saga solver which is helpful for large datasets. Finally, the fit method was called to begin the training and the data was labeled.

Rating the Model

Classification Report

The output is a classification report, which evaluates the performance of a machine learning model using linear regression algorithm. The breakdown of the report is below.

```
Accuracy: 0.793
Classification Report:
              precision    recall  f1-score   support

  Irrelevant       0.76      0.73      0.74       172
    Negative       0.75      0.88      0.81       266
     Neutral       0.89      0.70      0.78       285
    Positive       0.78      0.85      0.82       277

    accuracy                           0.79      1000
   macro avg       0.80      0.79      0.79      1000
weighted avg       0.80      0.79      0.79      1000
```

By Keys:

1. **Accuracy (0.793)**:
    - This metric represents the overall correctness of the model. It shows that 79.3% of the predictions made by the model are correct.
    - **Accuracy** is calculated as the ratio of correctly predicted instances to the total instances

2. **Precision**:
    - Precision measures the accuracy of positive predictions for each class. It tells us how many of the instances predicted as a particular class are actually that class.
    - Example: For the Neutral class, precision is 0.89, meaning 89% of instances predicted as neutral are correct.

3. **Recall**:
    - Recall, also known as sensitivity or true positive rate, measures how many actual instances of each class the model correctly identified.
    - Example: For the Negative class, recall is 0.88, meaning the model correctly identified 88% of all actual negative cases.

4. **F1-Score**:
    - The F1-score is the harmonic mean of precision and recall. It balances the trade-off between precision and recall and is useful when there is an uneven class distribution.
    - Example: The Positive class has an F1-score of 0.82, indicating a good balance between precision (0.78) and recall (0.85).

5. **Support**:
    - Support refers to the number of actual occurrences of each class in the dataset. For *example, there are 277 instances of the Positive class.*

By Class:

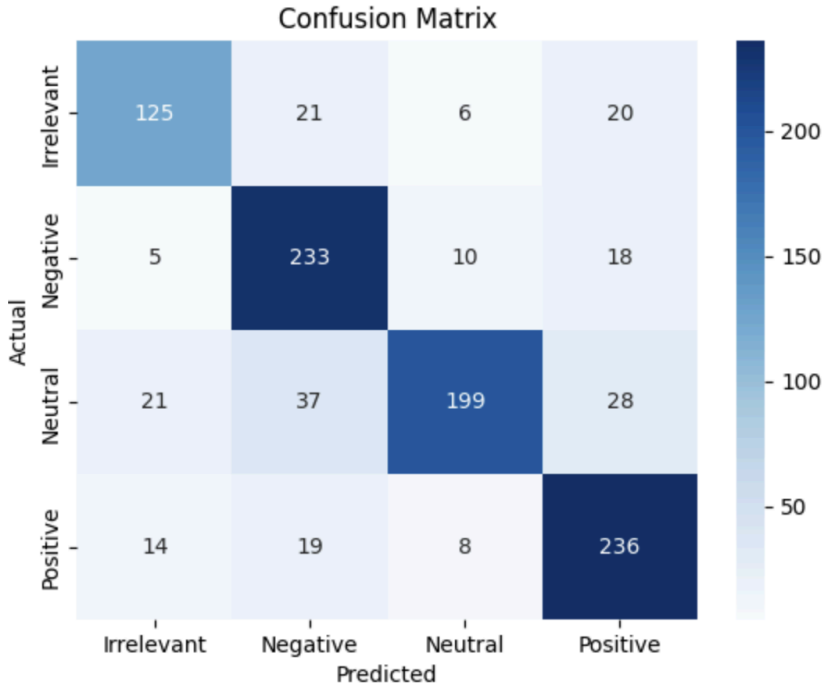
- **Irrelevant:** Precision is 0.76, meaning 76% of the predicted Irrelevant instances are correct. Recall is 0.73, indicating 73% of the actual Irrelevant instances were correctly identified. The F1-score of 0.74 shows a moderate balance between the two.

- **Negative:** High recall (0.88) shows the model is very good at identifying actual Negative cases, but slightly lower precision (0.75) means some predictions are falsely labeled as negative. The F1-score (0.81) indicates good overall performance.

- **Neutral:** Precision is the highest at 0.89, meaning when the model predicts Neutral, it is very likely to be correct. However, recall is lower at 0.70, showing that 30% of actual neutral cases were missed.

- **Positive:** The model performs well here, with both precision (0.78) and recall (0.85) leading to an F1-score of 0.82.

Confusion Matrix



The confusion matrix here shows how well the model classified tweets into four sentiment categories: Positive, Negative, Neutral, and Irrelevant. Each row represents the actual sentiment of tweets, while each column represents the predicted sentiment by the model.

Diagonal values represent correct classifications (e.g., 125 for Irrelevant-Irrelevant, 233 for Negative-Negative). All other values indicate misclassification. For example 21 irrelevant tweets were classified as negative.

Decision Tree

To ensure that our model produced the best outcome, we compared it to a different model: Decision Tree. We trained the model which produced an Accuracy on the validation set of 0.9110.

```
Accuracy: 0.911
Classification Report:
              precision    recall  f1-score   support

  Irrelevant       0.91      0.94      0.92       172
    Negative       0.87      0.95      0.91       266
     Neutral       0.95      0.87      0.91       285
    Positive       0.92      0.90      0.91       277

    accuracy                           0.91      1000
   macro avg       0.91      0.91      0.91      1000
weighted avg       0.91      0.91      0.91      1000
```

Classification Report

We then ran a classification report, which produced the above outcomes. The classes correspond to the labels the model predicts. Here's what each metric means:

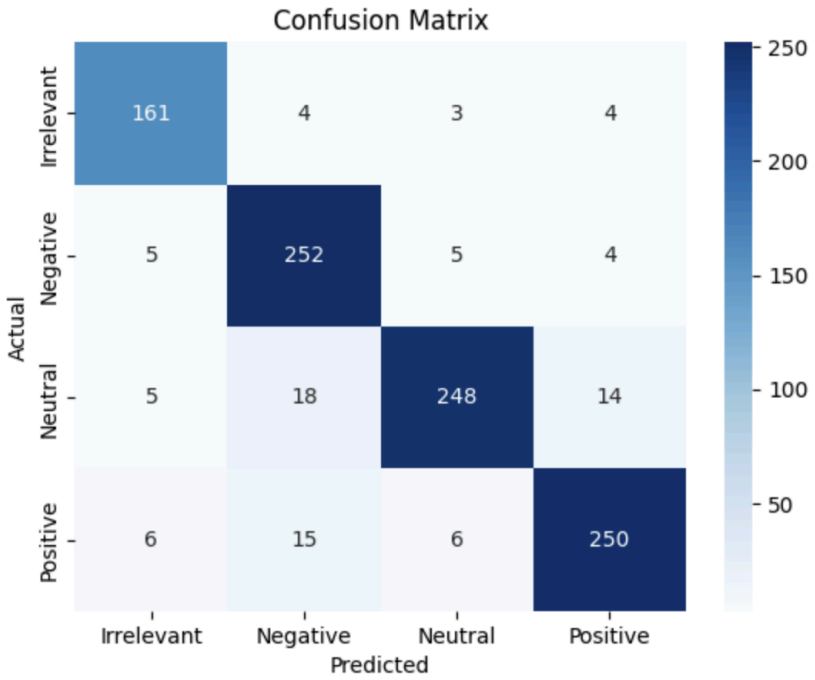
1. **Precision**:
    - Precision is the ratio of correctly predicted positive observations to the total predicted positives.
    - For example, for the "Irrelevant" class, precision is 0.91, meaning that 91% of the instances the model labeled as "Irrelevant" were actually "Irrelevant."
2. **Recall**:
    - Recall is the ratio of correctly predicted positive observations to all observations in the actual class.
    - For the "Irrelevant" class, recall is 0.94, meaning the model correctly identified 94% of all the actual "Irrelevant" examples.
3. **F1-Score**:
    - F1-score is the harmonic mean of precision and recall, giving a balanced measure of the two.
    - For the "Irrelevant" class, the F1-score is 0.92, indicating a good balance between precision and recall.
4. **Support**:
    - Support refers to the number of actual occurrences of the class in the dataset. For "Irrelevant," the support is 172, meaning there are 172 examples in the dataset with this true label.

Macro Average and Weighted Average:

- **Macro Average**:
  This is the simple average of precision, recall, and F1-score across all classes, treating each class equally. In our case, the macro averages are all around 0.91, suggesting balanced performance across all classes.
- **Weighted Average**:
  This is the average of precision, recall, and F1-score across all classes, weighted by the number of true instances for each class (i.e., considering the support).

Weighted averages help when class distribution is imbalanced, ensuring larger classes (like "Neutral" and "Positive") have more influence on the overall metric.

Confusion Matrix



Similar to the previous confusion matrix, the image above shows how well the model classified tweets into four sentiment categories: Positive, Negative, Neutral, and Irrelevant. Each row represents the actual sentiment of tweets, while each column represents the predicted sentiment by the model.

Diagonal values represent correct classifications (e.g., 161 for Irrelevant-Irrelevant, 252 for Negative-Negative). All other values indicate misclassification. For example 4 irrelevant tweets were classified as negative.

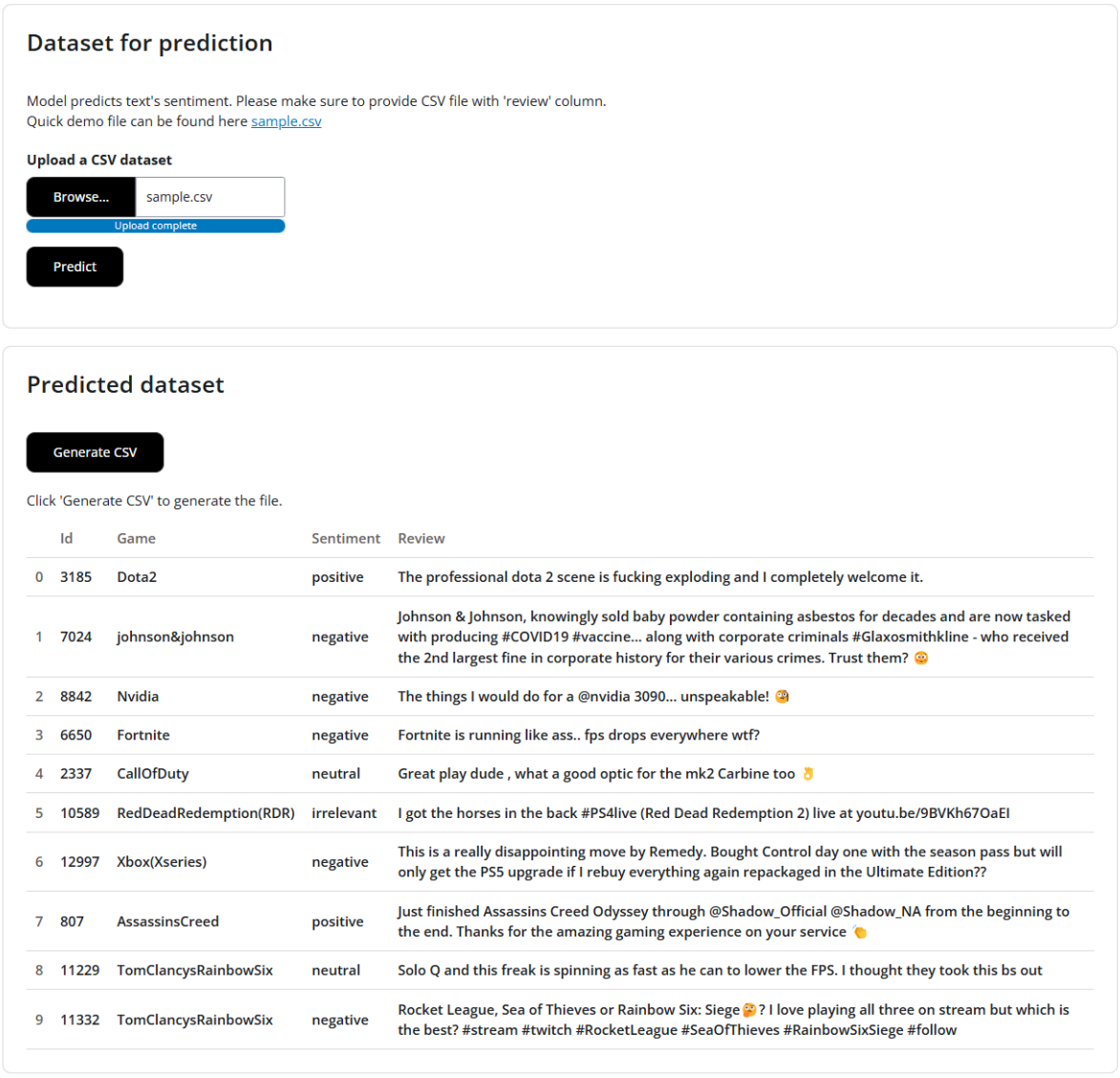
Classification Model of Simpletransformers

In addition we have reviewed the use of the classification model of the SimpleTransformers library. We decided not to proceed with this approach since it generated models that are around couple hundreds MegaBytes in size and take way longer time to train and generate in comparison to 500 kb models produced by DB Tree and other approaches that would also train within couple of minutes and predict within milliseconds depending on the dataset size.

Model Deployment

The final chosen model will be deployed for sentiment analysis based on the tweets provided.
The model was deployed using Shinyapps.io (www.shinyapps.io). Its url may be found here:

https://aabakarov.shinyapps.io/sentiment_nlp/



Its GitHub repo is public and may be found here:

https://github.com/ArslanAbakarov/nlp_sentiment

Application users just need to provide a CSV file that has a column “Review” to generate a return CSV that has also column Sentiment. Users can also preview the results on the web page of the App.

This application is simple to understand and use.

The link to the video presentation of the document and device may be found here

Conclusion

In this project, we developed an AI model that applies Natural Language Processing (NLP) to analyze sentiment from a diverse set of text-based reviews. Trained on a dataset that includes various forms of feedback, our model is designed to generalize across multiple types of review content, making it versatile for a range of applications beyond social media.

This project involved several stages, including data preprocessing to clean and structure the text, feature extraction using NLP techniques, and finally, model training and evaluation. The results demonstrate the model's ability to accurately classify sentiments across different types of reviews, showcasing its potential for analyzing customer feedback, product reviews, social commentary, and more.

Our project highlights the value of AI in automating sentiment analysis and its practical implications for understanding user feedback across various platforms. Future improvements might include refining the model on a more diverse dataset or leveraging advanced NLP architectures to enhance accuracy and adaptability.

This project provided valuable experience in NLP, data processing, and model training, essential for building effective AI-driven sentiment analysis tools.