Project 3 Group 18
Maiah Jaffa @jaffamaiah, Andres Portillo @andresporcruz, Martin Wall @wall-martin
GitHub Repository: https://github.com/Andresporcruz/social-media-sentiment-analysis.git
VIDEO LINK

Problem Statement:
In this project, we developed a sentiment analysis model that quantifies the sentiment in an input statement, based on social media data. This project works with data specific to public opinions on various topics, events, and entities. Through training a model with social media data from text-based platforms like X we quantify input statements as either positive or negative.

Motivation:
Society has shifted in recent years, and consumers and constituents can share their opinions on a variety of topics with ease. It is vital that organizations take advantage of the opportunity to analyze this data and make informed decisions that will benefit those they serve. Sentiment analysis insights can be useful in gauging public opinion on products, services, or policies. Further this work can aid organizations in identifying and addressing customer complaints promptly, monitoring and responding to PR crises effectively, and understanding trends and shifts in public perception over time.

Features:
This project allows for users to input a statement and receive a score of 1 or 0, with 1 indicating positive sentiment and 0 indicating negative sentiment. To train the two models we used to analyze sentiment in statements we used a dataset containing 1.6 million tweets from X and cleaned the dataset, extracting just the statement. 1.2 million of these statements are then assigned sentiment scores of either 1 or 0 and used the rest of the dataset to test the models. The user can select two different algorithms to calculate the sentiment on an input the Simple Algorithm or the Naive Bayes Algorithm. Another feature, visualize, allows users to view a Word Cloud that outputs the most used words the algorithms recognized from the dataset. The output Word Cloud can be sorted to view positive or negative words in the dataset according to either algorithm, allowing users to better understand the words the algorithms quantify as positive or negative.

Data:

Sentiment140, a Kaggle dataset, is the main dataset utilized in this study. A sentiment score has been assigned to each of the 1.6 million tweets in this dataset. The fields listed below are part of the schema:

- **Tweet_ID (int)**: A unique identifier for each tweet.
- **Timestamp (datetime)**: The date and time when the tweet was posted.
- **Text (string)**: The content of the tweet.
- **Sentiment (int)**: The sentiment associated with a tweet, where 0 indicates negative sentiment and 1 indicates positive sentiment.

Tools:

- Language: Python
    - **NLTK:** For natural language processing tasks.
    - **scikit-learn:** For implementing machine learning algorithms.
    - **pandas:** For data manipulation and analysis.
    - **matplotlib:** For data visualization.
    - **seaborn:** For data visualization.
    - **wordcloud:** For generating word cloud visualizations.
- Language: C++
    - **SFML:** For creating an interactive dashboard and handling the UI elements.
    - **Cstdio:** For passing input to and receiving output from python scripts.

Algorithms implemented:

We developed two different algorithms to calculate the sentiment of statements, our Simple Algorithm and our Naive Bayes Algorithm.

The Simple Algorithm calls custom_word_weighting_algorithm.py which looks at the phrases in the dataset with an associated positive sentiment then looks at the words included in those phrases and counts the number of times those words are used in the dataset of positive phrases. The same function is then done with the negative phrases. Then the words are assigned with a sentiment weight of 0 for negative sentiment and 1 for positive sentiment based on how many times they are found within the positive and negative datasets. Based on the sentiment weight of the words within an input statement the algorithm assesses the weights of the words and outputs the sentiment of a statement. The Naive Bayes Algorithm calls custom_naive_bayes.py which computes the probability distributions of words in the dataset to predict the sentiment of a statement. Whereas our simple algorithm only looks at the frequency of words appearing in either positive or negative sentiment phrases, our custom version of the popular Naïve Bayes algorithm uses

Bayes' theorem to weight a particular phrase based on prevalent features seen in phrases of positive or negative sentiment. It employs a probabilistic approach that considers not just the presence of individual words but their likelihoods across the entire phrase, allowing for a more holistic determination of the sentiment. This algorithm also uses smoothing to effectively account for words that appear in a particular phrase but do not appear anywhere in the training data (the simple algorithm has no way to assign sentiments to words that were not present in the training data).

Distribution of Responsibility and Roles:

Andres Portillo:

- **GitHub Setup**: Set up the project repository on GitHub, managing version control and collaboration among team members.
- **Backend Development**: Built and maintained the backend of the project, ensuring robust functionality and integration with the front-end.
- **Model Training**: Trained various sentiment analysis models, including the custom word weighting algorithm and the Multinomial Naive Bayes algorithm.
- **Data Processing**: Handled data preprocessing, including cleaning, vectorization, and splitting the dataset for training and testing.

Martin Wall:

- **Frontend Development**: Created the user interface and connected it to the backend.
- **Visualizations**: Generated data sets from the testing data to be used in creating word clouds for visual data display.
- **Video**: Organized and recorded the video to demonstrate the user interface of the project.

Maiah Jaffa:

- **Report:** Managed and wrote the final report, working with both team members to come up with a cohesive report.
- **Leadership:** Organized team meetings and made sure that everyone submitted necessary parts of the project by the deadline.

Changes Made after Initial Proposal:

Initially we were interested in classifying statements into three categories: positive, neutral, and negative. Providing users with a more nuanced understanding of sentiment in statements through capturing and classifying statements that aren't inherently sentimental or not sentimental. However, in progressing through our project we prioritized reaching an accuracy rate as close to 85% accurate as possible. In development we found that in our efforts to achieve a higher accuracy rate it would be best to use binary classifications. The ambiguity of classifying neutral statements led to a much more complex model with decreased performance of 50% accuracy. In using binary classification with our 2 algorithms, we reached 75% accuracy in custom_word_weighting_algorithm.py and 76% accuracy in custom_naive_bayes.py. Initially, we planned to implement two different machine learning algorithms, but that was beyond the project's scope. Instead, we implemented our Simple Algorithm and the custom simplified version of the Naive Bayes Algorithm. Our simple algorithm is much less complex than a machine learning algorithm which saved us time during the implementation of it and our custom Naïve Bayes algorithm does not make use of complex concepts such as Laplace smoothing, which would have made it very difficult to implement the Naïve Bayes algorithm in its entirety from scratch.

Big O Worst Case Time Complexity Analysis:

1. Data Loading and Preprocessing

- Function: load_data
    - Time Complexity: O(n)
    - Explanation: Cleaning the text by loading a CSV file and going through each row iteratively. In relation to the dataset's n rows, the complexity is linear.
- Function: clean_text
    - Time Complexity: O(m)
    - Explanation: Iterating over each character in the text and cleaning the text entry as it does. Where m is the length of a text entry, both regular expression matching and string manipulations are applied.
- Function: preprocess_data
    - Time Complexity: O(n * m)
    - Explanation: Each of the n text items are passed to the clean_text() function, resulting in an overall complexity of O(n * m), where m is the maximum length of a text entry.

2. Model Training and Prediction

- Function: train_model (for Custom Word Weighting Algorithm)
    - Time Complexity: O(n * m)
    - Explanation: Updates word counts in dictionary by iterating through n text entries of length m. Since the average time for inserting and checking a word in a dictionary is O(1), the total complexity is usually (n * m), but is $O((n * m)^2)$ in the worst case due to the worst-case time complexity of the dictionary.
- Function: predict (for Custom Word Weighting Algorithm)

    - Time Complexity: O(p * q)
    - Explanation: The algorithm determines scores based on word counts for p text entries, each of length q. The outcome is O(p * q), where q is the maximum length of the prediction texts.
- Function: train_model (for Naive Bayes using TF-IDF)
    - Time Complexity: O(n * m + n * log(n))
    - Explanation: The document-term matrix is created in O(n * m) via the TF-IDF vectorization. O(n * m + n * log(n)) is the usual outcome of training the Naive Bayes model, which entails operations logarithmic for each feature and operations linear in the number of documents.
- Function: predict (for Naive Bayes using TF-IDF)
    - Time Complexity: O(p * q)
    - Explanation: O(p * q) is the result of the model's predictions for p text entries of length q, each based on precomputed probabilities.

As a group, how was the overall experience for the project? This project was a great learning experience and offered a lot of room for growth both in code production and collaboration.

Did you have any challenges? If so, describe. It was challenging to get started with the project, stay coordinated and motivated as a group, and to connect the frontend to the backend.

If you were to start once again as a group, any changes you would make to the project and/or workflow? If we were to start again, we would start a bit earlier and communicate from the start about the strengths and weaknesses of each member. Additionally, we would try to identify ways to avoid machine learning algorithms because implementing a custom version of one took a lot of groundwork.

Comment on what each of the members learned through this process. We learned a lot about how to work as a team. Additionally, Andres learned about machine learning algorithms, Martin learned about connecting the frontend and backend of projects, and Maiah learned about sentiment analysis and how the math and code behind it works.

References

- https://www.kaggle.com/datasets/kazanova/sentiment140
- https://www.datacamp.com/tutorial/naive-bayes-scikit-learn
- https://www.youtube.com/watch?v=EK9uEfR53n4
- https://cplusplus.com/reference/cstdio/