SENTIMENT ANALYSIS PROJECT

GROUP MEMBERS

Ashhal Aamir – 29114
Saad Khan – 29162
Mohammad Faraz Ansari – 29201
Mohammad Ibrahim – 29079

WHAT IS SENTIMENT ANALYSIS

Sentiment analysis is a natural language processing technique used to determine the emotional tone behind a piece of text—whether it's positive, negative, or neutral. The goal is to extract subjective information (opinions, emotions, attitudes) from data to understand public perception, customer feedback, or personal sentiment trends.

OBJECTIVE/NEED

It has various uses in our everyday lives such as in the career and workplace, where it could be used for employee feedback analysis, resume & interview analysis, analyse job market trends. It can be used in business and analysing customer experiences, like companies can track social media sentiment to handle PR and monitor how their products are performing from reviews and complaints. It is a handy tool for Market research as well like analysing product reviews on Amazon and Yelp, identifying strengths and weaknesses. It can be used to tract social media mood such as on Twitter by analysing tweets, to detect negativity. It can even be used in finance and investigating. Our project is aimed at analysing general comments, to check if they are positive or negative.

MODULE DESCRIPTIONS

TRAIN.PY

This module is responsible for loading, preprocessing, and training a sentiment analysis model using a Naïve Bayes classifier with TF-IDF vectorization. It also handles dataset balancing and model evaluation before saving the trained model for future use. Worked on by Faraz, Ashhal and Ibrahim (error testing).

KEY LIBRARIES used are pandas for data loading and manipulation, scikit-learn – Model training (MultinomialNB), text vectorization (TfidfVectorizer), and evaluation (accuracy_score), and finally joblib for saving and loading the trained model.

KEY FUNCTIONS:

`Load_data(path):` Input: Reads a CSV file containing text data and sentiment scores. Preprocessing: Filters rows with missing values. Converts numerical scores (1-5) into sentiment labels: Positive (Score 4), Negative (Score 2), Neutral scores (3) are discarded. Balancing: Undersamples the majority class to ensure an equal number of positive and negative samples, preventing model bias. Output: Returns balanced lists of texts (Text) and corresponding labels (Sentiment).

`Train_model ():` Data Splitting: Uses train_test_split to divide data into 80% training and 20% testing sets. Model Pipeline: TF-IDF Vectorizer: Converts text into numerical features, removing stopwords and considering unigrams and bigrams (ngram_range=(1, 2)). Multinomial Naïve Bayes Classifier: Efficient for text classification with discrete features (e.g., word counts). Training & Evaluation: Fits the model on training data. Computes and prints accuracy on the test set. Model Persistence: Saves the trained pipeline to model.pkl using joblib.

OUTPUT:

Prints class distribution after balancing (e.g., {'Positive': 5000, 'Negative': 5000}). Displays test accuracy (e.g., Accuracy on test set: 0.85). Saves the trained model as model.pkl.

APP.PY

This module provides a graphical user interface (GUI) for the sentiment analysis model, allowing users to input a text review and receive an instant sentiment prediction (Positive or Negative). It loads the pre-trained model (model.pkl) and displays results in an interactive window. Worked on by Saad.

KEY LIBRARIES used are tkinter to create the GUI window and widgets. Joblib which loads the pre trained model.

KEY COMPONENTS:

Model Loading: Uses joblib to load the pre-trained Naïve Bayes + TF-IDF pipeline from model.pkl.

GUI Setup (Using tkinter)

Window Configuration:

Title: "Sentiment Analysis"

Text input field (Text widget) for user reviews.

Button ("Predict Sentiment") to trigger classification.

Label (result_label) to display predictions.

Prediction Function (predict ())

Input Handling: Extracts text from the input field and checks for empty input (shows a warning if empty).

Sentiment Prediction: Passes the review text to the model and retrieves the predicted sentiment.

Output Display: Updates result_label with the prediction (e.g., "Sentiment: Positive").

UTILS.PY

This module handles data loading, text preprocessing, and model serialization/deserialization. It prepares raw text data for training and provides utility functions to save/load trained models and vectorizers. Worked on by Ashhal.

KEY LIBRARIES: pandas for data loading and manipulation, re for regular expressions for text cleaning and joblib for efficient serialization of python objects.

Key Functions & Workflow

`load_data(filepath)`: Input: Reads a CSV file containing review texts (Text) and ratings (Score).
Preprocessing: Drops rows with missing values. Converts numerical scores into binary labels: 1 (Positive) for scores 4. 0 (Negative) for scores 2. Neutral scores (3) are filtered out.
Output: Returns two lists: Text: Raw review texts. label: Binary sentiment labels (0 or 1).

`clean_text(text)`:

Text Normalization: Removes URLs (http\S+).
Strips special characters (keeps only letters and whitespace).
Collapses multiple spaces into one.
Converts text to lowercase.
Output: Returns a cleaned string (e.g., "I loved this product").

`save_model(model, vectorizer, filepath)` Purpose: Saves a trained model and its vectorizer as a single .pkl file.
Usage Example: `save_model(classifier, tfidf_vectorizer, 'sentiment_model.pkl')`

`load_model(filepath)`: Purpose: Loads a saved model and vectorizer from disk.
Returns: A tuple (model, vectorizer) for prediction tasks.

PREDICT.PY

This module provides a command-line interface (CLI) for real-time sentiment analysis. It loads the pre-trained model and allows users to input text reviews interactively, returning immediate sentiment predictions (Positive/Negative). Worked on by Ibrahim.

KEY LIBRARIES: joblib for loading the serialized model pipeline.

Key Functions & Workflow

`predict_sentiment(text)`
Input: Raw text string (e.g., a product review).
Process: Loads the pre-trained model (model.pkl) using joblib.
Runs prediction on the input text.
Output: Returns the predicted sentiment ("Positive" or "Negative").

Interactive Loop

User Input: Prompts the user to enter a review or type `exit` to quit.
Prediction & Output: Displays the sentiment result in the console (e.g., Predicted Sentiment: Positive).
Exit Condition: Loop terminates when the user types `exit`.

PROBLEMS ENCOUNTERED

Initially, the sentiment analysis model was trained on a small dataset, which led to unreliable predictions due to insufficient data. To address this, we switched to a larger dataset containing 40,000 reviews. However, this introduced a new problem: the dataset was heavily skewed toward positive reviews, causing the model to bias its predictions toward the "Positive" class while ignoring negative sentiments. To resolve this imbalance, we undersampled the majority class (Positive) to match the number of negative samples, ensuring an equal distribution of both sentiments. This adjustment significantly improved the model's ability to distinguish between positive and negative reviews, leading to more accurate and unbiased predictions.

We attempted to develop the sentiment analysis model without relying on built-in libraries such as scikit-learn, aiming to implement core algorithms like TF-IDF and Naive Bayes from scratch. However, this approach proved highly complex and inefficient, requiring extensive code to handle text preprocessing, feature extraction, and classification logic. Challenges included:

- Sentence tokenization and organization – Manually breaking down and structuring text data led to inconsistencies.
- Algorithmic complexity – Reimplementing vectorization and probability calculations introduced errors and reduced performance.
- Maintenance difficulties – The custom code became hard to debug.

Ultimately, we transitioned to optimized libraries (scikit-learn, NLTK) for reliability and efficiency. This shift streamlined development, improved accuracy, and allowed focus on refining the model rather than rebuilding foundational components.

During development, we experimented with replacing the binary (positive/negative) scoring system with a more granular sentiment scale (e.g., 1–5 ratings or intensity-based classifications). The goal was to capture nuanced emotions like "mildly positive" or "strongly negative" for higher precision. However, this approach introduced challenges:

- Increased complexity in label interpretation and model training.
- Ambiguity in mid-range scores (e.g., distinguishing "neutral" vs. "mixed" sentiments).
- Time constraints prevented proper validation and tuning of the multi-class system.

Due to these hurdles, we reverted to the simpler binary classification to ensure project feasibility while maintaining reasonable accuracy. Future work could revisit a refined multi-level scoring system with expanded datasets and deeper analysis.

UPGRADES FOR THE FUTURE

1. Data-Level Improvements

- Expand & Balance Dataset: Collecting and using more diverse reviews (from platforms like Twitter, Amazon and Yelp).
- Utilizing data augmentation techniques and experimenting with oversampling.
- Implementing granular sentiment labels such as using a 5 star rating system, adding emotion detection like anger, joy, happiness and a sarcasm detector.

2. Model Upgrades

- Advanced Algorithms: Switching to transformer models like BERT via HuggingFace for improved accuracy.
- Implement deep learning so that the model can retain context for improved reviews.
- Probabilistic Outputs: Display a confidence score along with the sentiment like Positive: 70% and flag low confidence predictions for human reviews.

3. General Improvements

- Updating the GUI and using Tweepy to scrape live reviews of Twitter.