TEXT CLASSIFICATION SYSTEM

Sentiment Analysis

OCTOBER 16, 2024

Team Members

Name	Tasks
Nabil Sherif Nabil Ibrahim	Data handling and preprocessing (revising and modifying) Model building and training (implementing) Model Deployment using FastAPI (implementing) Documentation Powerpoint
Ahmed Mohamed Abdelalem Omar	Data handling and preprocessing (implementing) Model Deployment using FastAPI (implementing) Documentation Powerpoint
Shaimaa Ali Mohamed Mohamed	Data handling and preprocessing (implementing) Model Deployment using FastAPI (revising) Streamlit Demo (implementing) Documentation Powerpoint
Ahmed Maher Abdel Sameia	Data handling and preprocessing (revising and modifying) Model building and training (implementing) Streamlit Demo (implementing) Documentation Powerpoint
Mohamed Anwar Abdelhady Mohamed	Data handling and preprocessing (implementing) Streamlit Demo (implementing) Documentation Powerpoint

Table of Contents

Introduction

In this project, we develop a sentiment analysis model to classify tweets into two categories: positive and negative. By leveraging machine learning techniques, we aim to provide insights into public sentiment on various topics expressed through Twitter.

Purpose and Objectives:

The primary objective of this project is to build an effective model that accurately predicts whether a tweet conveys a positive or negative sentiment. This capability can be valuable for businesses, researchers, and social media analysts who want to gauge public opinion and sentiment trends in real-time. Our specific goals include:

- **Data Collection**: We utilize the Sentiment140 dataset, which consists of 1.6 million tweets labeled with their respective sentiments. This extensive dataset provides a rich source of diverse opinions, allowing our model to learn from a wide range of expressions and contexts. The dataset is particularly useful for training and evaluating the model's performance in classifying sentiments accurately.

- **Model Development**: Design and train a machine learning model capable of classifying sentiments based on textual data.

- **Deployment**: Create a FastAPI application to serve the model and enable users to make sentiment predictions easily.

What is Sentiment Analysis?

Sentiment analysis is a natural language processing (NLP) technique that involves determining the emotional tone behind a body of text. In this project, we focus on classifying sentiments expressed in tweets as either positive or negative.

This binary classification approach is particularly useful for understanding public reactions and opinions, especially during events or discussions that generate strong feelings.

By utilizing advanced modeling techniques, including Bidirectional LSTMs and convolutional neural networks (CNNs), we aim to create a robust sentiment classification model. This model will not only enhance our understanding of social media dynamics but also enable organizations to respond effectively to public sentiment.

Dataset Description

The dataset used in this project is the Sentiment140 dataset, which contains a total of 1,600,000 tweets extracted using the Twitter API. This extensive dataset has been annotated to facilitate sentiment detection, making it a valuable resource for training and evaluating sentiment analysis models.

Context

The Sentiment140 dataset offers a rich set of data points that can be utilized to classify sentiments expressed in tweets. Each tweet is annotated with a sentiment polarity: 0 for negative sentiments, 2 for neutral sentiments, and 4 for positive sentiments.

However, for the purposes of this project, we focus on binary sentiment classification, specifically distinguishing between positive and negative sentiments.

Content

The dataset consists of the following six fields:

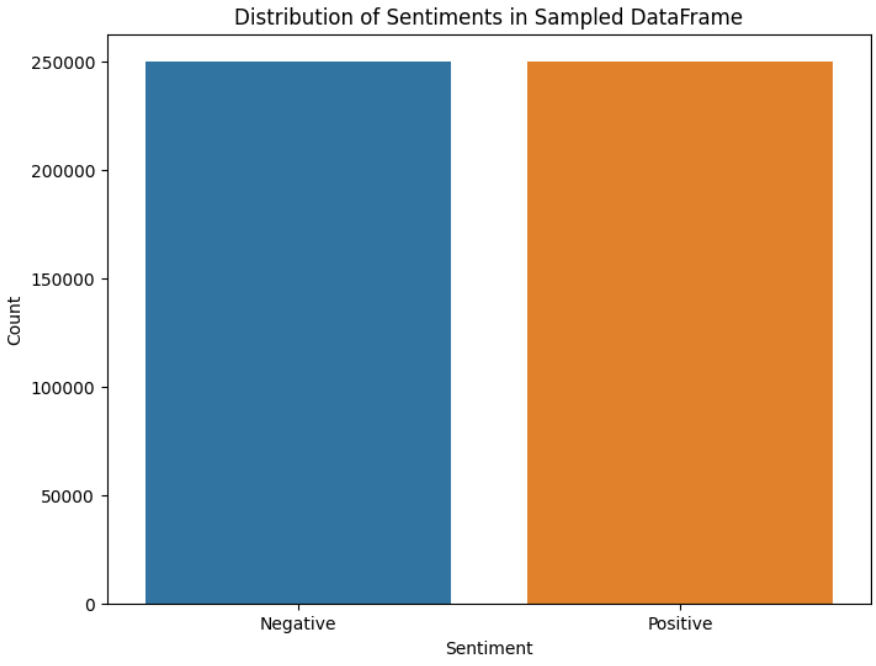
1. **target**: The polarity of the tweet, with values indicating sentiment:
    - **0** = Negative
    - **2** = Neutral (not used in this project)
    - **4** = Positive
2. **ids**: The unique identifier for the tweet (e.g., 2087).
3. **date**: The timestamp of when the tweet was posted (e.g., Sat May 16 23:58:44 UTC 2009).
4. **flag**: The query associated with the tweet. If no query exists, this field is marked as **NO_QUERY**.
5. **user**: The username of the individual who tweeted (e.g., robotickilldozr).
6. **text**: The content of the tweet itself (e.g., Lyx is cool).

This dataset serves as the foundation for training our sentiment analysis model, enabling us to understand and classify the sentiments expressed by Twitter users accurately.

Code Implementation

This section outlines the key steps involved in the implementation of the sentiment analysis model using the Sentiment140 dataset.

1. **Dataset Loading and Preparation**:
    - The dataset, which contains 1.6 million tweets
    - The dataset contains these columns: **sentiment**, **id**, **date**, **query**, **user_id**, and **text**.
    - The columns **id**, **date**, **query**, and **user_id** are dropped, leaving **sentiment** and **text** for further processing.
    - The sentiment values are mapped to "**Positive**" and "**Negative**" for binary classification.
2. **Sampling**:
    - To balance the dataset and reduce processing time, a random sample of 500,000 tweets is selected from the original dataset, maintaining the distribution of sentiment categories.
3. **Data Visualization**:
    - The distribution of sentiments in the sampled data is visualized using bar charts to ensure a balanced dataset.



4. **Text Preprocessing**:
    - Several preprocessing steps are applied to clean the text data, including:
        - Removing usernames (mentions).
        - Tokenizing text into words.
        - Normalizing tokens to lowercase.
        - Filtering out non-alphanumeric tokens and stop words.
        - Lemmatizing tokens based on part of speech.
        - Joining the cleaned tokens back into a single string.

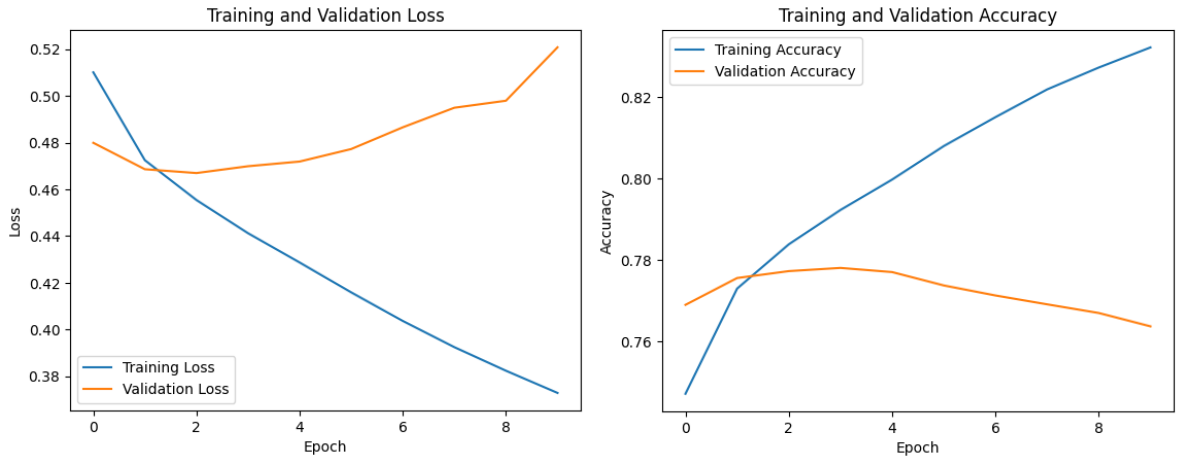
5. **Data Splitting**:
    - The pre-processed data is split into training and testing sets, with 80% used for training and 20% for testing.

6. **Text Vectorization**:
    - The FastText model is trained on the training data to generate word embeddings.
    - The text data is tokenized and converted to sequences, which are then padded to ensure uniform length.

7. **Model Building**:
    - Sequential Model Construction:
        - **Embedding Layer**: Initialized with the embedding matrix from FastText, using a vocabulary size and embedding dimension specific to our dataset.
        - **Single LSTM Layer**: Incorporating dropout to capture dependencies and provide regularization.
        - **Dense Layer**: A fully connected layer with fewer units to reduce complexity.
        - **Output Layer**: A sigmoid activation function for binary classification.

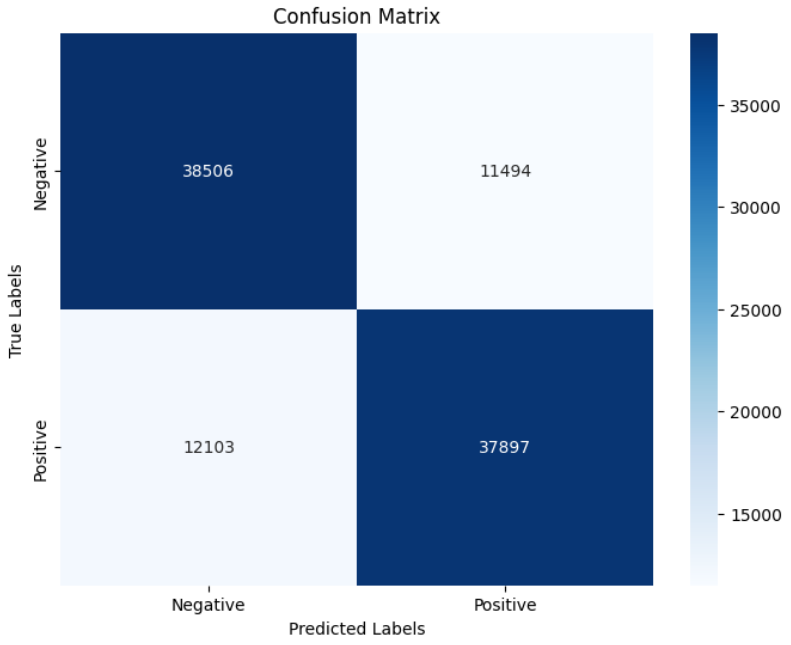
This model is compiled using the **Adam optimizer** and **binary cross-entropy loss**, ensuring accuracy is tracked during training. The simplified architecture balances model performance and computational efficiency.

8. **Model Training**:
    - The model is compiled and trained on the training data using the following parameters:
        - **Epochs**: 5, which defines the number of complete passes through the training dataset.
        - **Batch Size**: 64, which indicates the number of samples processed before the model is updated.
        - **Validation Split**: 0.2, meaning 20% of the training data is used for validation to monitor the model's performance during training.
    - The training history, including metrics like accuracy and loss, is stored for further analysis.



9. **Evaluation**:

- The model's performance is evaluated on the test data using a confusion matrix and classification report, providing insights into its accuracy and effectiveness.



10. **Model Saving**:

- The trained model is saved in HDF5 format for future use, allowing for easy loading and deployment.

API Development with FastAPI

Overview of FastAPI

FastAPI is a modern, high-performance web framework for building APIs with Python. It leverages standard Python type hints and offers several benefits, making it ideal for deploying machine learning models.

Benefits of Using FastAPI for Deployment

1. **Performance**: FastAPI is one of the fastest Python frameworks, thanks to its support for asynchronous programming.
2. **Ease of Use**: With its simple syntax, FastAPI minimizes boilerplate code, enhancing developer productivity.
3. **Interactive API Documentation**: Automatically generated Swagger UI and ReDoc documentation make API testing and interaction straightforward.
4. **Data Validation**: Pydantic ensures robust data validation, improving reliability.
5. **Type Hints and Editor Support**: Python type hints enhance code readability and help catch errors early.
6. **Asynchronous Support**: Asynchronous request handling boosts performance, especially for I/O-bound operations.
7. **Dependency Injection**: Simplifies dependency management, making the code modular and testable.

FastAPI Implementation

In this section, we developed an API using FastAPI to facilitate sentiment analysis based on the trained model. We started by saving our sentiment analysis model and tokenizer, which allows us to load them efficiently when handling requests.

We implemented the same preprocessing techniques used during the model training process to ensure consistency in text handling. This involved several steps:

1. **Usernames Removal:** We removed any mentions or usernames from the input text to focus on the content.
2. **Tokenization:** The text was tokenized into individual words for further processing.
3. **Normalization:** We converted all tokens to lowercase to maintain uniformity.
4. **Cleaning:** Non-alphanumeric tokens were filtered out to retain only meaningful words.
5. **Stop Words Removal:** Common stop words that do not contribute to sentiment were eliminated.
6. **Lemmatization:** Tokens were lemmatized to their base form based on their parts of speech, improving the model's understanding of the text.
7. **Meaningful Tokens Filtering:** Finally, tokens were filtered to retain only those that were sufficiently long and meaningful.

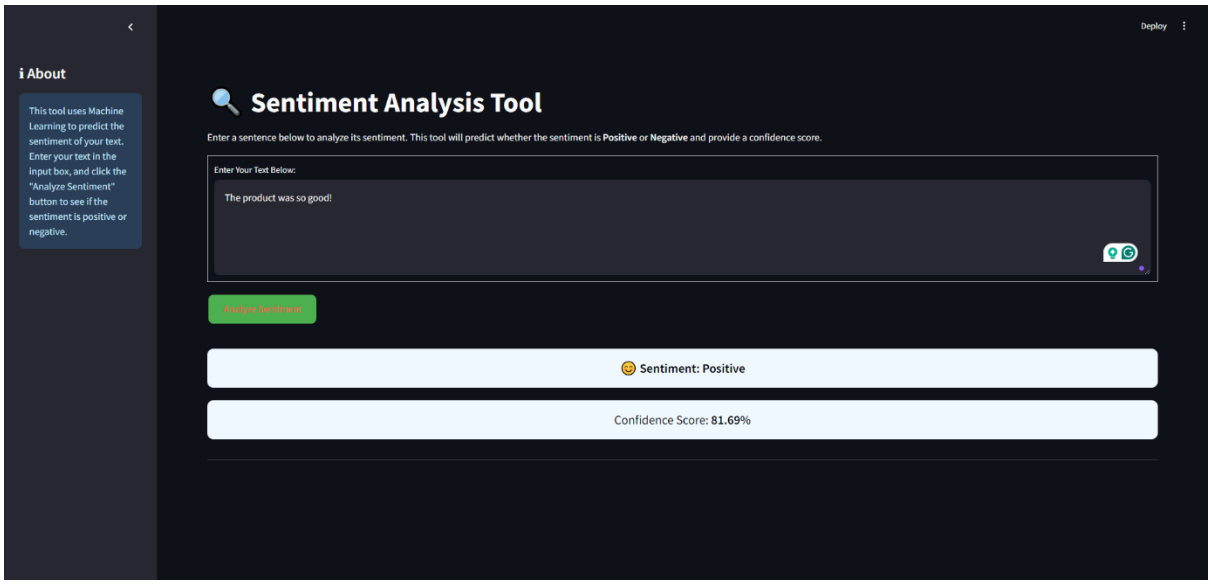
We exposed a prediction endpoint that accepts user input and returns the predicted sentiment (positive or negative) along with a confidence score for the prediction. This structured approach ensures that the API is efficient, reliable, and user-friendly, allowing us to deploy our sentiment analysis model seamlessly.

Deployment with Streamlit

To enhance user interaction and accessibility, we deployed the sentiment analysis model using Streamlit. This framework enabled us to create a user-friendly interface, allowing users to input text and analyze its sentiment with ease.

The application features a responsive design with clear instructions for users, guiding them to enter their text for analysis. Upon submission, the application communicates with the FastAPI backend, sending the user input for sentiment prediction.

Results are displayed in a visually appealing format, indicating whether the sentiment is positive or negative, along with a confidence score. Custom styling improves the overall user experience, making the tool intuitive and engaging. This deployment approach leverages Streamlit's strengths in rapid application development while providing an interactive platform for sentiment analysis.



AWS Flowchart Implementation