SENTIMENT ANALYSIS WEB APP

Submitted By

ABHINAND AS

DEEPU PRASAD H

THEJUIN P

JITHIN VM

1. Introduction

This Streamlit web application is designed to perform sentiment analysis on text data, enabling users to identify the underlying sentiment in textual inputs. The application supports both training from custom CSV datasets and real-time prediction on user-entered text. Sentiment analysis is widely used in areas such as customer feedback monitoring, social media analysis, and review aggregation. With the growing amount of textual data generated online, understanding the emotional tone behind user expressions is critical. The application uses a traditional machine learning approach, leveraging TF-IDF for feature extraction and a Multinomial Naive Bayes classifier for prediction. Users upload datasets that contain text labeled with sentiments like 'positive', 'negative', or 'neutral'.

2. Requirements

The following Python libraries are required:

streamlit

Pandas

scikit-learn

numpy

(optional) matplotlib / seaborn for visualizations

3. Model Setup

The model setup process begins with loading a dataset provided by the user in the form of a CSV file. The file must include two important columns: one containing the actual text (usually under the column name 'selected_text'), and the other containing the corresponding sentiment labels (e.g., 'positive', 'negative', or 'neutral'). The application validates these columns and proceeds only if both are present. Sentiment labels are then converted into numeric format using label encoding, a necessary step for training machine learning models. The dataset is split into training and test subsets using an 80:20 ratio to ensure fair evaluation. TF-IDF vectorization is applied to convert text into numerical form. This process highlights the importance of each word in a document relative to the entire dataset, helping the model to identify significant patterns. Once the features are prepared, a Multinomial Naive Bayes classifier is trained on the training data. This classifier is particularly suited for discrete features like word counts. After training, the model is ready to evaluate and predict sentiments.

4. Data Preprocessing

Data preprocessing plays a crucial role in ensuring that the model receives clean and meaningful input. The first step in preprocessing involves handling missing values. Rows with null values in either the text or sentiment columns are dropped to prevent issues during model training. The sentiment labels are then converted into numerical format using a label encoder, which transforms categorical values into machine-readable numbers. Next, the dataset is split into training and testing sets using stratified sampling to maintain the distribution of sentiment classes. After splitting, the text is transformed using TF-IDF vectorization. TF-IDF, or Term Frequency-Inverse Document Frequency, is a statistical method that converts text into numerical features by evaluating how important a word is to a document in a collection. Words that appear frequently in one document but rarely across others are given more weight, helping the model to distinguish between different sentiments. The TF-IDF vectorizer used in this app is limited to 5000 features and filters out English stopwords. This dimensionality reduction improves model performance and training speed without sacrificing accuracy.

5. Model Training

The model training process is handled using the Multinomial Naive Bayes classifier, a simple yet effective algorithm for text classification problems. Once the text data is vectorized using TF-IDF, the training features and labels are fed into the model. The classifier learns the relationship between word patterns and sentiment categories during the fitting process. It calculates the conditional probability of a sentiment given a particular word or combination of words. Multinomial Naive Bayes is highly efficient and works well with large sparse datasets, making it ideal for document classification. After training, the model is used to predict sentiments for the test dataset, allowing us to evaluate its generalization performance. The training process is fast, and the model occupies minimal memory, which is beneficial for real-time applications. This approach lays the foundation for building more complex NLP systems in the future, such as ensemble models or deep learning-based classifiers. For many use cases, this classic machine learning model offers a reliable and interpretable solution.

6. Model Evaluation

After the model is trained on the TF-IDF-transformed training data, its performance is evaluated on the test set using a variety of metrics. The most important metric is Accuracy, which measures the percentage of correctly predicted labels. However, accuracy alone may not be sufficient, especially in imbalanced datasets. Therefore, additional metrics such as Precision, Recall, and F1-score are also computed. Precision quantifies the number of true positive predictions made among all positive predictions. Recall measures how many actual positive cases were correctly predicted. F1-score is the harmonic mean of precision and recall, giving a better sense of overall model performance in uneven class distributions. These metrics are computed using the scikit-learn `precision_recall_fscore_support` and `classification_report` functions. In the app, the evaluation results are displayed in two ways: a summary block fixed at the bottom-right of the interface, and a detailed classification report shown inside an expandable section. This dual presentation allows users to quickly review high-level metrics or dive into class-wise performance details. By providing clear evaluation metrics, users can make informed decisions about the quality of their model and decide if improvements or retraining are necessary.

7. User Interaction

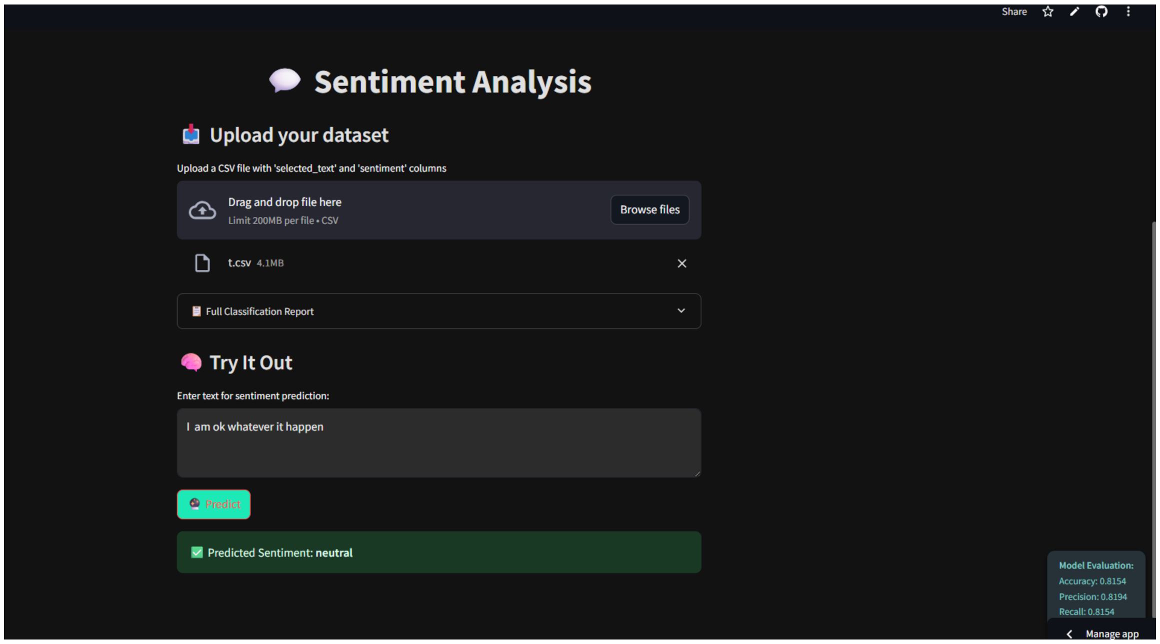
User interaction is a key focus of this Streamlit application. The interface is designed to be intuitive and accessible, even for users without technical expertise. Upon launching the app, users are prompted to upload a CSV file containing text samples and their associated sentiments. After successful upload and processing, users are presented with real-time model performance metrics such as accuracy and F1 score. Beyond static evaluation, the application includes a dynamic section where users can input their own text. This text is then transformed using the trained TF-IDF vectorizer, and the model predicts its sentiment class. The prediction is immediately shown on screen with a confirmation message. This interactivity allows users to experiment with various inputs and better understand how the model responds. Features such as text validation, visual feedback, and styled UI elements contribute to a smooth and engaging user experience. Overall, the app bridges the gap between machine learning models and end-users, making sentiment analysis approachable and insightful.

8. Output

The output section of the application is where the results of both model evaluation and real-time prediction are displayed. For model evaluation, a metrics box is shown in the bottom-right corner summarizing key performance indicators like accuracy, precision, recall, and F1-score. These results help users assess the effectiveness of the classifier based on their uploaded dataset. For real-time interaction, the application features a text area where users can enter custom text. When the 'Predict' button is clicked, the entered text is processed and classified using the trained model. The predicted sentiment is then displayed in a prominent styled message using `st.success()`, making it easy to interpret. This functionality is particularly useful for testing model behavior on various phrases or feedback entries. Whether the text is neutral, positive, or negative, the app highlights the result clearly. The dual output mechanism—evaluation metrics and live prediction—adds practical value to the app by combining analytical validation with hands-on experimentation.

Examples:

Neutral sentence:



Sentiment Analysis

Upload your dataset

Upload a CSV file with 'selected_text' and 'sentiment' columns

Drag and drop file here
Limit 200MB per file • CSV

t.csv 4.1MB

Full Classification Report

Try It Out

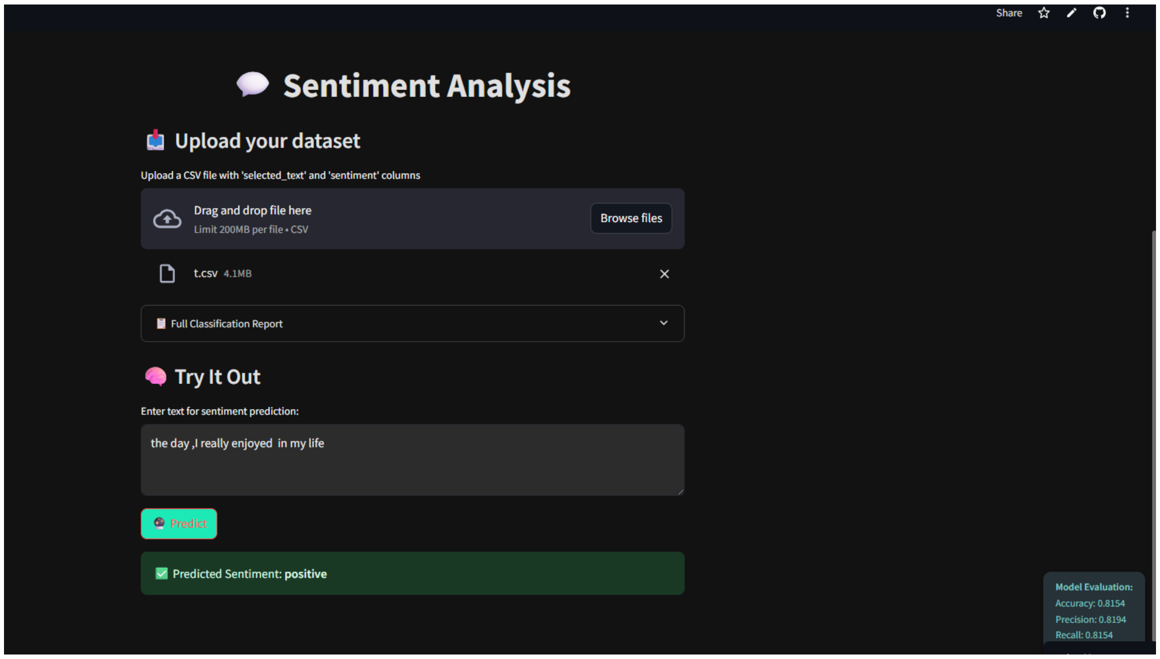
Enter text for sentiment prediction:

I am ok whatever it happen

Predicted Sentiment: **neutral**

Model Evaluation:
Accuracy: 0.8154
Precision: 0.8194
Recall: 0.8154

Positive Sentence:



Sentiment Analysis

Upload your dataset

Upload a CSV file with 'selected_text' and 'sentiment' columns

Drag and drop file here
Limit 200MB per file • CSV

t.csv 4.1MB

Full Classification Report

Try It Out

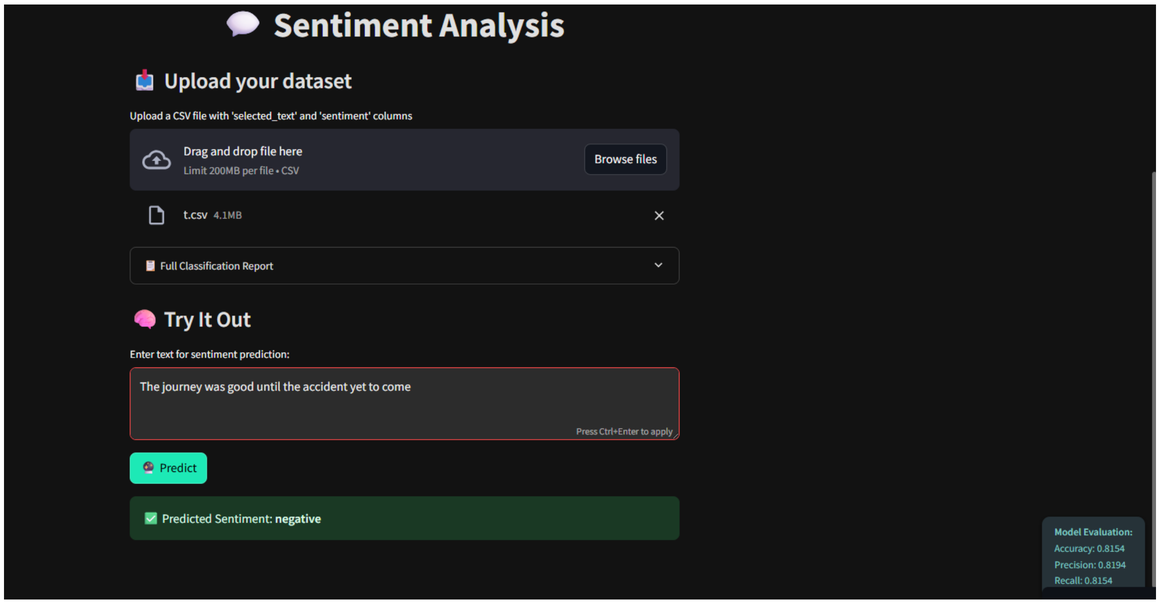
Enter text for sentiment prediction:

the day ,I really enjoyed in my life

Predicted Sentiment: **positive**

Model Evaluation:
Accuracy: 0.8154
Precision: 0.8194
Recall: 0.8154

Negative sentence:



Sentiment Analysis

Upload your dataset

Upload a CSV file with 'selected_text' and 'sentiment' columns

Drag and drop file here
Limit 200MB per file • CSV

Browse files

t.csv 4.1MB

Full Classification Report

Try It Out

Enter text for sentiment prediction:

The journey was good until the accident yet to come

Press Ctrl+Enter to apply

Predict

Predicted Sentiment: **negative**

Model Evaluation:
Accuracy: 0.8154
Precision: 0.8194
Recall: 0.8154

9. Step-by-Step Code Explanation

The code behind the sentiment analysis app is structured in a logical and modular fashion to promote readability and functionality.

1. import Libraries: The app begins by importing essential libraries such as pandas for data handling, scikit-learn for machine learning, and Streamlit for building the UI.

2. App Title and Theme : A custom HTML/CSS-based dark theme is applied using `st.markdown`, along with a title and an image to enhance visual presentation.

3.File Upload: Users are prompted to upload a CSV file containing `selected_text` and `sentiment` columns. This is handled by `st.file_uploader`.

4. Data Cleaning & Label Encoding: The dataset is validated for missing entries. Sentiments are encoded numerically using `LabelEncoder` to prepare for machine learning.

5. Train-Test Split & TF-IDF: Data is split into training and test sets (80:20). The text is converted to a numerical format using TF-IDF with a feature limit and stopword removal.

6. Model Training: A `MultinomialNB` classifier is trained on the TF-IDF vectors. This model is chosen for its suitability for high-dimensional text data.

7. Model Evaluation: The trained model is used to predict test data. Evaluation metrics

(accuracy, precision, recall, F1) are calculated and displayed.

8. Text Input for Prediction: Users can input new text into a text box. After clicking Predict, the app uses the model to classify the input and display the sentiment.

9. **Display Results: The output includes a success message for the sentiment, and model metrics are shown in a styled block for clarity.

The app's structure ensures seamless flow from dataset input to result interpretation, encouraging user engagement and learning.

Conclusion

This sentiment analysis application provides a practical, user-friendly platform for text classification using traditional machine learning methods. It serves as a bridge between technical implementation and real-world use, allowing users to upload datasets, train models, and test inputs—all within an interactive Streamlit interface. The application leverages TF-IDF vectorization to transform text data and employs a Multinomial Naive Bayes classifier for prediction. With evaluation metrics readily available and a built-in text input feature, the app is ideal for experimenting with sentiment analysis tasks. It can serve as an educational tool for students learning NLP, a prototyping platform for developers, or a lightweight solution for small-scale projects. The modular design also makes it easy to extend, whether by integrating advanced classifiers like SVMs or neural networks, or by adding visualizations and dashboards. The app can be deployed publicly using Streamlit Cloud, making it accessible across devices and operating systems. In summary, the project is a well-rounded introduction to practical natural language processing with real-world utility.