Sentiment Analysis and Machine Learning Model Documentation

Introduction

This documentation provides an overview of the sentiment analysis and machine learning pipeline implemented in the provided Jupyter Notebook. The project leverages natural language processing (NLP) techniques to analyze sentiment from textual data and builds a machine learning model to predict sentiments.

It seems you uploaded a new file, but I can use the context of the previous notebook file to create the documentation as they may relate to the same project. Here's a comprehensive documentation based on the structure and content typically found in a sentiment analysis and machine learning project:

Sentiment Analysis and Machine Learning Model Documentation

This documentation provides an overview of the sentiment analysis and machine learning pipeline implemented in the provided Jupyter Notebook. The project leverages natural language processing (NLP) techniques to analyze sentiment from textual data and builds a machine learning model to predict sentiments.

Project Structure

1. **Environment Setup**
    - **Objective:** Install and import necessary libraries to ensure the notebook can execute all operations required for sentiment analysis and model training.
    - **Tools Used:**
        - pandas for data manipulation.
        - nltk (Natural Language Toolkit) for sentiment analysis using VADER (Valence Aware Dictionary for Sentiment Reasoning).
        - scikit-learn for machine learning tasks.
2. **Data Loading**

- **Objective:** Load the dataset for analysis.
- **Method:**
  - The data is read from a CSV file (generic_sentiment_dataset_50k.csv) into a Pandas DataFrame.
  - The data structure is explored using basic DataFrame operations like .head().

3. Data Preprocessing

- **Objective:** Clean and prepare the data for sentiment analysis and model training.
- **Steps:**
  - Handle missing values by filling NaNs in the text column.
  - Convert text data into a suitable format for analysis.

4. Sentiment Analysis Using VADER

- **Objective:** Analyze the sentiment of text data using the VADER sentiment analyzer.
- **Method:**
  - VADER is initialized, and a custom function is defined to classify sentiment based on the compound score provided by VADER.
  - The function is applied to the text data, and the results are compared to the actual sentiment labels in the dataset.

5. Model Training and Testing

- **Objective:** Train a machine learning model to predict sentiment based on the text data.
- **Steps:**
  - Convert textual data into numerical data using CountVectorizer.
  - Split the data into training and testing sets.
  - Train a Naive Bayes classifier (MultinomialNB) on the training data.
  - Test the model on the test set and generate predictions.

6. Model Evaluation

- **Objective:** Evaluate the performance of the trained model.
- **Metrics:**

        - Generate a classification report that includes precision, recall, and F1-score.
        - Create a confusion matrix to visualize the model's performance.
        - Visualize the confusion matrix using a heatmap for better interpretability.
7. **Result Storage**
    - **Objective:** Save the results of the sentiment analysis and model predictions.
    - **Method:**
        - The DataFrame with the original, predicted sentiments, and their comparison is saved to a new CSV file for further analysis.

Requirements

- Python packages:
    - pandas
    - nltk
    - scikit-learn
    - matplotlib and seaborn for visualization (if used)

HOW CODE IS EXECUTED :

Introduction

This Python script is designed as a simple web application for performing sentiment analysis on textual data. The application uses Streamlit for the user interface and scikit-learn for the machine learning model. The user can input text, and the app predicts whether the sentiment is positive, neutral, or negative.

Key Components

1. **Libraries Imported**
    - **Streamlit (st)**: Used to create the web interface for the sentiment analysis application.
    - **Pandas (pd)**: Used for data manipulation and handling.
    - **CountVectorizer from scikit-learn**: Converts text data into numerical features suitable for machine learning.
    - **Multinomial Naive Bayes (MultinomialNB)**: A machine learning algorithm used to classify the sentiment.

Code:

```
import streamlit as st
import pandas as pd
from sklearn.feature_extraction.text import CountVectorizer
from sklearn.naive_bayes import MultinomialNB
```

2.Application Title

- Sets the title of the web app to "Sentiment Analysis App".

Code:

```
st.title("Sentiment Analysis App")
```

3.Data Loading

- **Objective:** Load the dataset used for training the sentiment analysis model.
- **Method:** The script attempts to load a CSV file located at C:\\Users\\dream\\Downloads\\generic_sentiment_dataset_50k.csv. If the file is not found, it displays an error message and stops execution.

Code:

```
file_path = 'C:\\Users\\dream\\Downloads\\generic_sentiment_dataset_50k.csv'

try:
    df = pd.read_csv(file_path)
except FileNotFoundError:
    st.error("Data file not found. Please check the path and try again.")
    st.stop()
```

4.Dataset Preview

- Displays the first few rows of the dataset in the app to give users an overview of the data being used.

Code:

```
st.write("### Preview of the Dataset")
st.write(df.head())
```

5. Data Validation

- **Objective:** Ensure the dataset contains the necessary columns (text and sentiment).

- **Action:** If the required columns are missing, the app shows an error and stops execution.

Code:

```
if 'text' not in df.columns or 'sentiment' not in df.columns:
    st.error("The dataset must contain 'text' and 'sentiment' columns.")
    st.stop()
```

6. **Data Preprocessing**

- **Objective:** Prepare the text data for model training.
- **Steps:**
    - Handle missing values in the text column by filling them with empty strings.
    - Split the dataset into features (X) and labels (y).

Code:

```
df['text'] = df['text'].fillna('')
X = df['text']
y = df['sentiment']
```

7. **Text Vectorization**

- **Objective:** Convert the text data into a numerical format using CountVectorizer.
- **Method:** The CountVectorizer is initialized with English stop words removed, and the text data is transformed into vectors.

Code:

```
vectorizer = CountVectorizer(stop_words='english')
X_vectorized = vectorizer.fit_transform(X)
```

8. **Model Training**

- **Objective:** Train a Naive Bayes classifier on the vectorized text data.
- **Method:** The MultinomialNB model is trained using the vectorized text data and corresponding sentiment labels.

Code:

```
model = MultinomialNB()
model.fit(X_vectorized, y)
```

9. **User Input for Sentiment Analysis**

- **Objective:** Allow users to input text and analyze its sentiment.
- **Method:** A text area is provided for user input, and a button triggers the sentiment analysis.

Code:

```
st.write("### Write a Review")
user_input = st.text_area("Enter your review here:", "")
```

10. **Sentiment Prediction**

- **Objective:** Predict the sentiment of the user's input text.
- **Steps:**
    - The input text is vectorized using the trained CountVectorizer.
    - The sentiment is predicted using the trained Naive Bayes model.
    - The result is displayed as Positive, Neutral, or Negative based on the prediction.

Code:

```
if st.button("Analyze Sentiment"):
    if user_input:
        input_vectorized = vectorizer.transform([user_input])
        prediction = model.predict(input_vectorized)
```

```
        st.write("### Sentiment Analysis Result")
        if prediction[0] == 'positive':
            st.success("The sentiment is Positive! 😊")
        elif prediction[0] == 'neutral':
            st.info("The sentiment is Neutral. 😐")
        else:
            st.error("The sentiment is Negative! 😔")
    else:
        st.warning("Please enter a review to analyze.")
```

Conclusion

This script provides a straightforward sentiment analysis tool that allows users to input text and receive a sentiment classification in real-time. The model is trained on a dataset of labeled text, and predictions are made using a Naive Bayes classifier. The application is built using Streamlit, making it easy to deploy and interact with.

WHAT I HAVE LEARNT?

1. **Sentiment Analysis Workflow:** The project is designed to perform sentiment analysis on text data. It uses both rule-based sentiment analysis (VADER) and machine learning models (Naive Bayes).
2. **Streamlit App Structure:** The project includes a Streamlit app that allows users to input text and receive real-time sentiment predictions. The app uses pandas for data handling, CountVectorizer to convert text into numerical features, and MultinomialNB for sentiment classification.
3. **Common Issues in Cloud Deployment:**
    - The importance of using **relative paths** for files when deploying to cloud platforms.
    - How to handle missing files (e.g., CSV dataset) in cloud deployments and the use of st.file_uploader() in Streamlit to allow file uploads.
4. **Dependency Management:** The need for a requirements.txt file that lists all necessary Python libraries (e.g., scikit-learn, pandas, streamlit) to ensure the app runs properly on both local and cloud environments.

These insights highlight the process of building, testing, and deploying a machine learning-based sentiment analysis tool, and the common issues developers might encounter, especially in deployment scenarios.

Thanking You:

Thank you for taking the time to explore this project. Your interest and feedback are highly appreciated.

Acknowledgments

I would like to acknowledge the resources and documentation provided by the open-source community, especially the developers behind scikit-learn, pandas, and Streamlit, which made this project possible.

References

- scikit-learn Documentation
- Streamlit Documentation
- pandas Documentation

  Natural Language Toolkit (nltk)

Contact Information

For any questions, feedback, or collaboration inquiries, feel free to contact me:

Gmail: naledushyanth@gmail.com

GitHub: https://github.com/dreammast