Twitter Sentiment Analysis Project Documentation

Overview:

This project focuses on analyzing the sentiment of tweets using machine learning. The dataset used for this project is the Sentiment140 dataset from Kaggle, which contains 1.6 million labeled tweets. The main objective is to predict the sentiment of a tweet as either positive or negative.

Steps and Implementation:

1. Installing Required Libraries

We begin by installing the Kaggle library in Google Colab, which allows us to download datasets directly from Kaggle.

Code:

```
!pip install Kaggle
```

2. Kaggle API Setup

To download datasets from Kaggle, a Kaggle API token is required:

1. Generate a new API token in the Kaggle settings, which will download a JSON file (`kaggle.json`).

2. Upload this JSON file to your Colab environment and set up Kaggle authentication.

Code:

```
!mkdir -p ~/.kaggle

!cp kaggle.json ~/.kaggle/

!chmod 600 ~/.kaggle/kaggle.json
```

3. Downloading the Dataset

Once authenticated, download the Sentiment140 dataset using the Kaggle API command copied from the Kaggle dataset page.

Code:

```
!kaggle datasets download -d kazanova/sentiment140
```

4. Extracting the Dataset

The downloaded dataset is in a ZIP format. We extract it using the `ZipFile` module.

Code:

```
from zipfile import ZipFile

dataset = '/content/sentiment140.zip'

with ZipFile(dataset, 'r') as zip:

    zip.extractall()

    print('The dataset is extracted')
```

5. Importing Necessary Libraries

Several libraries are required for data processing, cleaning, and modeling. Here's a brief description of each:

Numpy: For numerical operations.

Pandas: For data manipulation and analysis.

Re: For regular expression operations.

nltk: For natural language processing, specifically stopwords removal and stemming.

sklearn: For machine learning tasks like data splitting, vectorization, model building, and evaluation.

Code:

```
import numpy as np
import pandas as pd
import re
from nltk.corpus import stopwords
from nltk.stem.porter import PorterStemmer
from sklearn.feature_extraction.text import TfidfVectorizer
from sklearn.model_selection import train_test_split
from sklearn.linear_model import LogisticRegression
from sklearn.metrics import accuracy_score
```

6. Downloading Stopwords

Stopwords are common words that do not contribute much to the meaning of a sentence. We download the list of English stopwords using `nltk`.

python

Code:

```
import nltk
nltk.download('stopwords')
print(stopwords.words('english'))
```

7. Loading the Dataset

The dataset is loaded into a Pandas DataFrame, and its shape (number of rows and columns) is checked.

Code:

```
df = pd.read_csv('/content/training.1600000.processed.noemoticon.csv', encoding='ISO-8859-1')
df.shape
```

8. Renaming Columns

Since the dataset lacks column names, we manually assign them.

Code:

```
column_names = ['target', 'id', 'date', 'flag', 'user', 'text']

df = pd.read_csv('/content/training.1600000.processed.noemoticon.csv', encoding='ISO-8859-1', names=column_names)

df.head()
```

9. Data Cleaning

We check for null values in the dataset to ensure data integrity.

Code:

```
df.isnull().sum()
```

10. Target Column Distribution

The `target` column, which indicates the sentiment (positive or negative), is critical. We analyze its distribution to ensure balance, as an imbalanced dataset can affect model performance.

Code:

```
df['target'].value_counts()
```

11. Visualizing Sentiment Distribution

We plot the distribution of sentiments using Seaborn and Matplotlib.

Code:

```
import seaborn as sns

import matplotlib.pyplot as plt

sns.countplot(x='target', data=df, palette='Set2')

plt.xlabel('Sentiment')

plt.ylabel('Count')
plt.title('Sentiment Distribution')

plt.show()
```

12. Text Preprocessing with Stemming

A stemming function is defined to preprocess the tweet texts by:

- Removing non-alphabetical characters.
- Converting text to lowercase.
- Removing stopwords.
- Applying stemming to reduce words to their root form.

Code:

```
port_stem = PorterStemmer()


def stemming(content):

    stemmed_content = re.sub('[^a-zA-Z]', ' ', content)

    stemmed_content = stemmed_content.lower()

    stemmed_content = stemmed_content.split()

    stemmed_content = [port_stem.stem(word) for word in stemmed_content if
not word in stopwords.words('english')]

    stemmed_content = ' '.join(stemmed_content)

    return stemmed_content

df['stemmed_content'] = df['text'].apply(stemming)
```

13. Splitting Data and Labels

The dataset is split into features (tweets) and labels (sentiments).

Code:

```
X = df['stemmed_content'].values

Y = df['target'].values
```

14. Generating Word Clouds

Word clouds for positive and negative sentiments are generated to visualize the most common words.

Code:

```
from wordcloud import WordCloud

positive_text = ' '.join(df[df['target'] == 'positive']['stemmed_content'])

negative_text = ' '.join(df[df['target'] == 'negative']['stemmed_content'])

wordcloud = WordCloud(max_words=100,
background_color='white').generate(positive_text)

plt.figure(figsize=(10, 7))

plt.imshow(wordcloud, interpolation='bilinear')

plt.axis('off')

plt.title('Positive Tweets Word Cloud')

plt.show()


wordcloud = WordCloud(max_words=100,
background_color='white').generate(negative_text)

plt.figure(figsize=(10, 7))

plt.imshow(wordcloud, interpolation='bilinear')

plt.axis('off')

plt.title('Negative Tweets Word Cloud')

plt.show()
```

15. Building the Sentiment Analysis Model

We split the dataset into training and testing sets, vectorize the text data, and build a logistic regression model to predict tweet sentiments.

Code:

```
X_train, X_test, Y_train, Y_test = train_test_split(X, Y, test_size=0.2, stratify=Y,
random_state=2)

vectorizer = TfidfVectorizer()

X_train = vectorizer.fit_transform(X_train)

X_test = vectorizer.transform(X_test)


model = LogisticRegression(max_iter=1000)

model.fit(X_train, Y_train)
```

<u>Evaluating the model</u>

Code:

```
X_train_prediction = model.predict(X_train)

training_data_accuracy = accuracy_score(X_train_prediction, Y_train)

print('Accuracy score on the training data:', training_data_accuracy * 100)


X_test_prediction = model.predict(X_test)

testing_data_accuracy = accuracy_score(X_test_prediction, Y_test)

print('Accuracy score on the testing data:', testing_data_accuracy * 100)
```

Conclusion:

The logistic regression model achieved an accuracy of 81.01% on the training data and 77.80% on the testing data. This project demonstrates a basic approach to sentiment analysis using machine learning, with potential improvements including the use of more sophisticated models and additional text preprocessing techniques.