Sentiment Analysis Using Logistic Regression

1. Introduction

This document describes how to use a Logistic Regression model to analyze sentiment on the Sentiment140 dataset. Model training, evaluation, data preprocessing, and environment setup are the steps.

2. Environment Setup

2.1 Installing Kaggle and Downloading the Dataset

1. Install Kaggle in Google Colab:

```
!pip install kaggle
```

2. Create a Kaggle API token in the Kaggle settings, download the kaggle.json file, and upload it to Colab:

```
!mkdir -p ~/.kaggle
!cp kaggle.json ~/.kaggle/
!chmod 600 ~/.kaggle/kaggle.json
```

3. Download the Sentiment140 dataset:

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

4. Extract the dataset using the ZipFile library in Python:

```
from zipfile import ZipFile
dataset = '/content/sentiment140.zip'
with ZipFile(dataset, 'r') as zip:
  zip.extractall()
  print('The dataset is extracted')
```

2.2 Required Libraries

Import the necessary libraries for data manipulation, text processing, and model building:

```
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
```

3. Data Loading and Preprocessing

3.1 Downloading Stopwords

Download stopwords from NLTK and print the English stopwords:

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

3.2 Loading Data

Load the data from the CSV file into a pandas DataFrame and check its shape:

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

Print the first 5 rows using df.head(). Rename columns if necessary:

```
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()
```

3.3 Data Cleaning

Check for null values in each column:

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

Focus on the 'target' and 'text' columns, checking their distribution:

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

If the target column is imbalanced, consider balancing it for better model performance.

3.4 Stemming Function

Define a stemming function to clean and stem the text:

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

```
    stemmed_content = stemmed_content.lower()
    stemmed_content = stemmed_content.split()
    stemmed_content = [PorterStemmer().stem(word) for word in stemmed_content if not
word in stopwords.words('english')]
    stemmed_content = ' '.join(stemmed_content)
    return stemmed_content
```

Apply the stemming function to all tweets:

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

4. Model Training and Evaluation

4.1 Data and Labels Separation

Separate the data (tweets) and labels (target):

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

4.2 Word Cloud Visualization

Create word clouds for positive and negative texts:

```
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()
```

4.3 Model Creation and Evaluation

Split the data into training and test sets, and train a Logistic Regression model:

```
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)
```

Evaluate the model's accuracy on the training and testing data:

```
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 testing data: ', testing_data_accuracy*100)
```