Bilenkin550Week3_Exercise_3.2

March 29, 2025

3.2 Exercise: Sentiment Analysis and Preprocessing Text

Part 1: Using the TextBlob Sentiment Analyzer

1. Import the movie review data as a data frame and ensure that the data is loaded properly.

[64]:
```python
import pandas as pd

# Loading and defining the file path to a variable
file_path = r"C:\Users\maxim\OneDrive\Desktop\BU\DSC 550\labeledTrainData.tsv"

# Loading the dataset from local folder
df = pd.read_csv(file_path, delimiter='\t')

# Displaying the first 5 rows
df.head()
```

[64]:
```
       id  sentiment                                             review
0  5814_8          1  With all this stuff going down at the moment w…
1  2381_9          1  \The Classic War of the Worlds\" by Timothy Hi…
2  7759_3          0  The film starts with a manager (Nicholas Bell)…
3  3630_4          0  It must be assumed that those who praised this…
4  9495_8          1  Superbly trashy and wondrously unpretentious 8…
```

2. How many of each positive and negative reviews are there?

[65]:
```python
# Counting the number of positive and negative reviews
review_counts = df['sentiment'].value_counts()

# Mapping sentiment values to labels and formatting counts with commas
review_counts_with_labels = review_counts.rename({1: 'Positive', 0: 'Negative'})

# Formatting the counts with commas
formatted_counts = review_counts_with_labels.apply(lambda x: f'{x:,}')

# Displaying the counts with labels and formatted numbers
print(formatted_counts)
```

```
sentiment
Positive    12,500
```

```
Negative    12,500
Name: count, dtype: object
```

3. Use TextBlob to classify each movie review as positive or negative. Assume that a polarity score greater than or equal to zero is a positive sentiment and less than 0 is a negative sentiment.

[66]:
```python
# Importing necessary package
from textblob import TextBlob

# Function to classify sentiment based on polarity
def classify_sentiment(review):
    # Creating a TextBlob object
    blob = TextBlob(review)

    # Getting the polarity score and classify positive or negative
    polarity = blob.sentiment.polarity
    if polarity >= 0:
        return 'Positive'
    else:
        return 'Negative'

# Applying the classify_sentiment function to each review
df['predicted_sentiment'] = df['review'].apply(classify_sentiment)

# Displaying the first 5 rows with the predicted sentiment
df[['review', 'sentiment', 'predicted_sentiment']].head()
```

[66]:
```
                                              review  sentiment  \
0  With all this stuff going down at the moment w…          1
1  \The Classic War of the Worlds\" by Timothy Hi…          1
2  The film starts with a manager (Nicholas Bell)…          0
3  It must be assumed that those who praised this…          0
4  Superbly trashy and wondrously unpretentious 8…          1

  predicted_sentiment
0            Positive
1            Positive
2            Negative
3            Positive
4            Negative
```

4. Check the accuracy of this model. Is this model better than random guessing?

[67]:
```python
# Importing necessary package for evaluation
from sklearn.metrics import accuracy_score

# Mapping 'Positive' and 'Negative' to 1 and 0 for comparison
```

```
df['predicted_sentiment_numeric'] = df['predicted_sentiment'].map({'Positive':␣
↪1, 'Negative': 0})

# Calculating the accuracy
accuracy = accuracy_score(df['sentiment'], df['predicted_sentiment_numeric'])

# Printing the accuracy
print(f"Accuracy of the model: {accuracy * 100:.2f}%")
```

```
Accuracy of the model: 68.52%
```

Based on the above result, the model achieves an accuracy rate of 68%, which is better than random guessing (50/50) in a binary classification task. However, although the model's accuracy exceeds random guessing, its predictive power is not particularly impressive. The model may benefit from exploring other libraries and packages with higher predictive capabilities.

5. For up to five points extra credit, use another prebuilt text sentiment analyzer, e.g., VADER, and repeat steps (3) and (4).

Importing VADER and classifying sentiments

[68]:
```
# Importing necessary package for VADER
from vaderSentiment.vaderSentiment import SentimentIntensityAnalyzer

# Initializing the SentimentIntensityAnalyzer
analyzer = SentimentIntensityAnalyzer()

# Function to classify sentiment using VADER
def classify_sentiment_vader(review):
    # Getting the sentiment score using VADER
    sentiment_score = analyzer.polarity_scores(review)

    # Classifying sentiment as Positive or Negative based on the compound score
    if sentiment_score['compound'] >= 0:
        return 'Positive'
    else:
        return 'Negative'

# Applying the classify_sentiment_vader function to each review
df['predicted_sentiment_vader'] = df['review'].apply(classify_sentiment_vader)

# Displaying the first 5 rows with the predicted sentiment using VADER
df[['review', 'sentiment', 'predicted_sentiment_vader']].head()
```

[68]:
```
                                              review  sentiment  \
0  With all this stuff going down at the moment w…          1
1  \The Classic War of the Worlds\" by Timothy Hi…          1
2  The film starts with a manager (Nicholas Bell)…          0
3  It must be assumed that those who praised this…          0
```

```
4  Superbly trashy and wondrously unpretentious 8…          1

  predicted_sentiment_vader
0                  Negative
1                  Positive
2                  Negative
3                  Negative
4                  Positive
```

Checking the accuracy of the VADER model

```
[69]: from sklearn.metrics import accuracy_score

      # Mapping the predicted sentiment values (VADER)
      df['predicted_sentiment_vader'] = df['predicted_sentiment_vader'].
       ↪map({'Positive': 1, 'Negative': 0})

      # Comparing the predicted sentiment from VADER with the actual sentiment
      accuracy_vader = accuracy_score(df['sentiment'],␣
       ↪df['predicted_sentiment_vader'])

      print(f'Accuracy of VADER model: {accuracy_vader * 100:.2f}%')
```

```
Accuracy of VADER model: 69.40%
```

Accuracy of VADER Model:

The VADER model achieved an accuracy of 69.40%, which is slightly higher than the TextBlob model's accuracy of 68%. Although both models outperform random guessing (50%) in a binary classification task, the VADER model shows a marginal improvement. However, the predictive power of both models still leaves room for improvement. Further exploration of more advanced models or techniques might yield better results for sentiment classification.

Part 2: Prepping Text for a Custom Model

1. Convert all text to lowercase letters.

```
[70]: df['review'] = df['review'].str.lower()



      # Checking the first 2 rows
      df.head(2)
```

```
[70]:        id  sentiment                                            review  \
      0  5814_8          1  with all this stuff going down at the moment w…
      1  2381_9          1  \the classic war of the worlds\" by timothy hi…

        predicted_sentiment  predicted_sentiment_numeric  predicted_sentiment_vader
      0            Positive                            1                          0
      1            Positive                            1                          1
```

2. Remove punctuation and special characters from the text.

```
[71]: df['review'] = df['review'].apply(lambda x: ''.join([char for char in x if char␣
       ↪not in string.punctuation]))

      # Checking the first 2 rows
      df.head(2)
```

```
[71]:       id  sentiment                                             review  \
      0  5814_8          1  with all this stuff going down at the moment w…
      1  2381_9          1  the classic war of the worlds by timothy hines…

        predicted_sentiment  predicted_sentiment_numeric  predicted_sentiment_vader
      0            Positive                            1                          0
      1            Positive                            1                          1
```

3. Remove stop words.

```
[72]: # Ensuring stopwords dataset is loaded
      from nltk.corpus import stopwords

      # Loading the stopwords
      stop_words = stopwords.words('english')

      # Removing stopwords from the review text
      df['review'] = df['review'].apply(lambda x: ' '.join([word for word in x.
       ↪split() if word not in stop_words]))

      # Checking the first 2 rows
      df.head(2)
```

```
[72]:       id  sentiment                                             review  \
      0  5814_8          1  stuff going moment mj ive started listening mu…
      1  2381_9          1  classic war worlds timothy hines entertaining …

        predicted_sentiment  predicted_sentiment_numeric  predicted_sentiment_vader
      0            Positive                            1                          0
      1            Positive                            1                          1
```

4. Apply NLTK's PorterStemmer.

```
[73]: # Importing PorterStemmer from nltk
      from nltk.stem import PorterStemmer

      # Creating a PorterStemmer object
      stemmer = PorterStemmer()

      # Applying the stemmer to each word in the review
```

```
df['review'] = df['review'].apply(lambda x: ' '.join([stemmer.stem(word) for␣
 ↪word in x.split()]))

# Checking the first 2 rows after stemming
df.head(2)
```

[73]:
```
       id  sentiment                                             review  \
0  5814_8          1  stuff go moment mj ive start listen music watc…
1  2381_9          1  classic war world timothi hine entertain film …

  predicted_sentiment  predicted_sentiment_numeric  predicted_sentiment_vader
0            Positive                            1                          0
1            Positive                            1                          1
```

5. Create a bag-of-words matrix from your stemmed text (output from (4)), where each row is a word-count vector for a single movie review (see sections 5.3 & 6.8 in the Machine Learning with Python Cookbook). Display the dimensions of your bag-of-words matrix. The number of rows in this matrix should be the same as the number of rows in your original data frame.

[74]:
```
# Importing CountVectorizer from sklearn
from sklearn.feature_extraction.text import CountVectorizer

# Initializing the CountVectorizer
vectorizer = CountVectorizer()

# Creating the Bag-of-Words (BoW) matrix
X_bow = vectorizer.fit_transform(df['review'])

# Getting the dimensions of the BoW matrix
num_reviews, num_unique_words = X_bow.shape

# Displaying the dimensions with explanations and formatted numbers
print(f"Number of reviews in the dataset: {num_reviews:,}")
print(f"Number of unique words in the Bag-of-Words matrix: {num_unique_words:
 ↪,}")
```

```
Number of reviews in the dataset: 25,000
Number of unique words in the Bag-of-Words matrix: 92,379
```

6. Create a term frequency-inverse document frequency (tf-idf) matrix from your stemmed text, for your movie reviews (see section 6.9 in the Machine Learning with Python Cookbook). Display the dimensions of your tf-idf matrix. These dimensions should be the same as your bag-of-words matrix.

[75]:
```
# Importing TfidfVectorizer from sklearn
from sklearn.feature_extraction.text import TfidfVectorizer

# Initializing the TfidfVectorizer
tfidf_vectorizer = TfidfVectorizer()
```

```
# Creating the TF-IDF matrix
X_tfidf = tfidf_vectorizer.fit_transform(df['review'])

# Getting the dimensions of the TF-IDF matrix
num_reviews_tfidf, num_unique_words_tfidf = X_tfidf.shape

# Displaying the dimensions with explanations and formatted numbers
print(f"Number of reviews in the dataset: {num_reviews_tfidf:,}")
print(f"Number of unique words in the TF-IDF matrix: {num_unique_words_tfidf:
 ↪,}")
```

```
Number of reviews in the dataset: 25,000
Number of unique words in the TF-IDF matrix: 92,379
```