C550-T301-Data_mining_2241_week3_Samanta_rajib

September 17, 2023

0.1 Class : C550-T301 Data Mining (2241-1)

0.2 Name : Rajib Samanta

0.2.1 Assignment : Week 3

Download the labeled training dataset from this link: Bag of Words Meets Bags of Popcorn. https://www.kaggle.com/competitions/word2vec-nlp-tutorial/data

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.
2. How many of each positive and negative reviews are there?
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.
4. Check the accuracy of this model. Is this model better than random guessing?
5. For up to five points extra credit, use another prebuilt text sentiment analyzer, e.g., VADER, and repeat steps (3) and (4).

[59]:
```
# Import Libraries
import pandas as pd
import os
#pip install textblob
from textblob import TextBlob
# pip install vaderSentiment
from vaderSentiment.vaderSentiment import SentimentIntensityAnalyzer
import re
# pip install nltk
import nltk
from nltk.corpus import stopwords
from nltk.stem import PorterStemmer
from sklearn.feature_extraction.text import CountVectorizer
from sklearn.feature_extraction.text import TfidfVectorizer
```

[60]:
```
#pip install nltk
```

[61]:
```
# Read the labeled training dataset  file ('labeledTrainData.tsv') from local:
directory = '/Users/rajibsamanta/Documents/Rajib/College/Sem6_fall_2023/Week3'
# Set the working directory
```

```
os.chdir(directory)
print(os.getcwd())
# 1. Import the movie review data as a data frame and ensure that the data is␣
 ↪loaded properly.

file_name = "labeledTrainData.tsv"

# Load the dataset into a pandas DataFrame
df = pd.read_csv(file_name, delimiter='\t', quoting=3)

# Display few records.
df.head()
```

```
/Users/rajibsamanta/Documents/Rajib/College/Sem6_fall_2023/Week3
```

[61]:

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

[62]:
```
# 2. How many of each positive and negative reviews are there?

# Count the number of positive (label=1) and negative (label=0) reviews
positive_reviews = df[df['sentiment'] == 1].shape[0]
negative_reviews = df[df['sentiment'] == 0].shape[0]

print("Number of Positive Reviews:", positive_reviews)
print("Number of Negative Reviews:", negative_reviews)
```

```
Number of Positive Reviews: 12500
Number of Negative Reviews: 12500
```

[63]:
```
# 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.

# Function to classify sentiment based on polarity score
def classify_sentiment(polarity):
    if polarity >= 0:
        return "Positive"
    else:
        return "Negative"

# Apply sentiment analysis using TextBlob and classify sentiment
df['polarity'] = df['review'].apply(lambda x: TextBlob(x).sentiment.polarity)
#df['sentiment'] = df['polarity'].apply(classify_sentiment)
```

```
df['predicted_sentiment'] = df['polarity'].apply(classify_sentiment)

# Print the first few rows of the DataFrame with sentiment labels
print(df.head())
```

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

   polarity predicted_sentiment
0  0.001277            Positive
1  0.256349            Positive
2 -0.053941            Negative
3  0.134753            Positive
4 -0.024290            Negative
```

[64]:
```
# map the orginal dataset for positive and negative
df['sentiment'] = df['sentiment'].map({1: 'Positive',0: 'Negative'})
print(df.head())
```

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

   polarity predicted_sentiment
0  0.001277            Positive
1  0.256349            Positive
2 -0.053941            Negative
3  0.134753            Positive
4 -0.024290            Negative
```

[65]:
```
# 4. Check the accuracy of this model. Is this model better than random␣
 ↪guessing?


# Calculate accuracy by comparing predicted sentiment with actual sentiment
correct_predictions = (df['sentiment'] == df['predicted_sentiment']).sum()
print(f"correct_predictions: {correct_predictions * 100:.2f}%")
total_predictions = len(df)
print(f"total_predictions: {total_predictions * 100:.2f}%")
accuracy = correct_predictions / total_predictions
```

```
print(f"Accuracy: {accuracy * 100:.2f}%")
```

```
correct_predictions: 1712600.00%
total_predictions: 2500000.00%
Accuracy: 68.50%
```

0.2.2 The accuracy is 68.50% significantly better than random guessing (where random guessing would give us around 50% accuracy in a binary sentiment classification)

[66]:
```
# 5. For up to five points extra credit, use another prebuilt text sentiment␣
 ↪analyzer, e.g., VADER, and repeat steps (3) and (4).
# Function to classify sentiment based on TextBlob polarity
# Load the dataset into a pandas DataFrame agaian
df = pd.read_csv(file_name, delimiter='\t', quoting=3)

def classify_sentiment_textblob(polarity):
    return "Positive" if polarity >= 0 else "Negative"

# Apply TextBlob sentiment analysis
df['textblob_polarity'] = df['review'].apply(lambda x: TextBlob(x).sentiment.
 ↪polarity)
df['textblob_sentiment'] = df['textblob_polarity'].
 ↪apply(classify_sentiment_textblob)

# Initialize the VADER sentiment analyzer
vader_analyzer = SentimentIntensityAnalyzer()

# Function to classify sentiment based on VADER sentiment scores
def classify_sentiment_vader(review):
    sentiment_scores = vader_analyzer.polarity_scores(review)
    compound_score = sentiment_scores['compound']
    return "Positive" if compound_score >= 0 else "Negative"

# Apply VADER sentiment analysis
df['vader_sentiment'] = df['review'].apply(classify_sentiment_vader)

# map the orginal dataset for positive and negative
df['sentiment'] = df['sentiment'].map({1: 'Positive',0: 'Negative'})

# Calculate accuracy for TextBlob
textblob_accuracy = (df['sentiment'] == df['textblob_sentiment']).sum() /␣
 ↪len(df)

# Calculate accuracy for VADER
vader_accuracy = (df['sentiment'] == df['vader_sentiment']).sum() / len(df)
```

```
print(f"TextBlob Accuracy: {textblob_accuracy * 100:.2f}%")
print(f"VADER Accuracy: {vader_accuracy * 100:.2f}%")
```

```
TextBlob Accuracy: 68.50%
VADER Accuracy: 69.41%
```

The accuracy of both TextBlob and VADER in classifying sentiments are mostly same.

0.2.3 Part 2: Prepping Text for a Custom Model

If you want to run your own model to classify text, it needs to be in proper form to do so. The following steps will outline a procedure to do this on the movie reviews text. 1. Convert all text to lowercase letters. 2. Remove punctuation and special characters from the text. 3. Remove stop words. 4. Apply NLTK's PorterStemmer. 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. 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.

[67]:
```
# Load the dataset into a pandas DataFrame agaian
df = pd.read_csv(file_name, delimiter='\t', quoting=3)
# 1. Convert all text to lowercase letters.
# Convert the 'review' column to lowercase
df['review'] = df['review'].str.lower()

# Print the first few rows of the DataFrame to verify the conversion
print(df[['review']].head())
```

```
                                              review
0  "with all this stuff going down at the moment …
1  "\"the classic war of the worlds\" by timothy …
2  "the film starts with a manager (nicholas bell…
3  "it must be assumed that those who praised thi…
4  "superbly trashy and wondrously unpretentious …
```

[68]:
```
# 2. Remove punctuation and special characters from the text.
# Function to remove punctuation and special characters


def remove_punctuation(text):
    # Define a regular expression pattern to match non-alphanumeric characters
    pattern = r'[^a-zA-Z0-9\s]'
```

```
    # Use the re.sub() function to replace matched characters with an empty␣
 ↪string
    return re.sub(pattern, '', text)

# Apply the remove_punctuation function to the 'review' column
df['review'] = df['review'].apply(remove_punctuation)

# Print the first few rows of the DataFrame to verify the removal of punctuation
print(df.head())
```

```
         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…
2  "7759_3"          0  the film starts with a manager nicholas bell g…
3  "3630_4"          0  it must be assumed that those who praised this…
4  "9495_8"          1  superbly trashy and wondrously unpretentious 8…
```

[69]:
```
# 3. Remove stop words.

# Download NLTK stop words
nltk.download('stopwords')
# Function to remove stop words
def remove_stopwords(text):
    # Tokenize the text into words
    words = text.split()

    # Remove stop words
    stop_words = set(stopwords.words("english"))

    filtered_words = [word for word in words if word.lower() not in stop_words]

    # Join the filtered words back into a single string
    return " ".join(filtered_words)

#print(stop_words)
# Apply the remove_stopwords function to the 'review' column
df['review'] = df['review'].apply(remove_stopwords)

# Print the first few rows of the DataFrame to verify the removal of stop words
print(df[['review']].head())
```

```
[nltk_data] Downloading package stopwords to
[nltk_data]     /Users/rajibsamanta/nltk_data…
[nltk_data]   Package stopwords is already up-to-date!
```

```
                                              review
0  stuff going moment mj ive started listening mu…
1  classic war worlds timothy hines entertaining …
2  film starts manager nicholas bell giving welco…
```

```
3  must assumed praised film greatest filmed oper…
4  superbly trashy wondrously unpretentious 80s e…
```

[70]:
```
# 4. Apply NLTK's PorterStemmer.

# Download NLTK stop words and initialize the Porter Stemmer
nltk.download('stopwords')
stemmer = PorterStemmer()


# Function to remove punctuation, remove stop words, and apply stemming
def preprocess_and_stem(text):
    # Remove punctuation
    text = re.sub(r'[^a-zA-Z0-9\s]', '', text)

    # Tokenize the text into words
    words = text.split()

    # Remove stop words
    stop_words = set(stopwords.words("english"))
    filtered_words = [word for word in words if word.lower() not in stop_words]

    # Apply stemming using the Porter Stemmer
    stemmed_words = [stemmer.stem(word) for word in filtered_words]

    # Join the stemmed words back into a single string
    return " ".join(stemmed_words)

# Apply the preprocess_and_stem function to the 'review' column
df['review'] = df['review'].apply(preprocess_and_stem)

# Print the first few rows of the DataFrame to verify the preprocessing and␣
 ↪stemming
print(df.head())
```

```
[nltk_data] Downloading package stopwords to
[nltk_data]     /Users/rajibsamanta/nltk_data…
[nltk_data]   Package stopwords is already up-to-date!
```

	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 …
2	"7759_3"	0	film start manag nichola bell give welcom inve…
3	"3630_4"	0	must assum prais film greatest film opera ever…
4	"9495_8"	1	superbl trashi wondrous unpretenti 80 exploit …

[71]:
```

```

```
# 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.
# Initialize the CountVectorizer
vectorizer = CountVectorizer()

# Fit and transform the 'review' column to create the BoW matrix
bow_matrix = vectorizer.fit_transform(df['review'])

# Display the dimensions of the BoW matrix
print("Dimensions of the Bag-of-Words (BoW) Matrix:", bow_matrix.shape)
```

```
Dimensions of the Bag-of-Words (BoW) Matrix: (25000, 92226)
```

[72]:
```
# 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.

# Initialize the TfidfVectorizer
tfidf_vectorizer = TfidfVectorizer()

# Fit and transform the 'review' column to create the TF-IDF matrix
tfidf_matrix = tfidf_vectorizer.fit_transform(df['review'])

# Display the dimensions of the TF-IDF matrix
print("Dimensions of the TF-IDF Matrix:", tfidf_matrix.shape)
```

```
Dimensions of the TF-IDF Matrix: (25000, 92226)
```

[ ]:
```

```