Ravi_Chandra_Reddy_Basireddy_CSCI544_HW1

September 8, 2022

1 CSCI544 Homework1 - Ravi Chandra Reddy Basireddy

Python Version : 3.9

1.1 Imports

- Pandas: To work with dataframes.
- NLTK: A Natural Language Toolkit used for processing textual data.
- RE: Regular Expressions used for handling word findings & substitutions.
- BS4: BeautifulSoup Library is a parser that can handle HTML Tags and Links.
- Contractions: A library to contract and de-contract Contractions.
- String: A library to handle strings.
- Warning: A library to handle console warnings.

[1]:
```
import pandas as pd
import nltk
nltk.download('wordnet',quiet=True)
import re
from bs4 import BeautifulSoup
import contractions
import string
import warnings
warnings.filterwarnings(action='ignore')

#!pip install bs4
#!pip install contractions
#!pip install nltk
#!pip install string
#!pip install pandas
#!pip install warnings
#!pip install sklearn
```

1.2 Read Data

- Read Data from a TSV file where the data is seperated using tabs.
- we are only intrested in Star Rating and Review Body.
- Star Rating: Rating given by the customers in the range of 1 to 5.
- Review Body: Review given by the customers in the textual format.

[2]:
```
amazon_reviews=pd.read_csv('data.tsv',␣
↪sep='\t',usecols=['star_rating','review_body'],low_memory=False)
```

1.3 Keep Reviews and Ratings

- Already completed at the reading data step.
- Dropping NaN Values which has no meaning to the rating.
- Dropping Duplicates which are repeated.
- Printing out the first five values to know what kind of data is in dataframe.

[3]:
```
amazon_reviews=amazon_reviews.dropna()
amazon_reviews=amazon_reviews.drop_duplicates()
amazon_reviews.head(5)
```

[3]:
```
  star_rating                                        review_body
0           5  so beautiful even tho clearly not high end …
1           5  Great product.. I got this set for my mother, …
2           5  Exactly as pictured and my daughter's friend l…
3           5  Love it. Fits great. Super comfortable and nea…
4           5  Got this as a Mother's Day gift for my Mom and…
```

1.4 We select 20000 reviews randomly from each rating class.

- filtering out the data with respective labels.
- sampling 20k reviews from each class.
- Combining all the data from diffent classes to create a vector of Dimension (100000,2).

[4]:
```
star_one=amazon_reviews[amazon_reviews.star_rating=='1']
star_one=star_one.sample(n=20000)
star_two=amazon_reviews[amazon_reviews.star_rating=='2']
star_two=star_two.sample(n=20000)
star_three=amazon_reviews[amazon_reviews.star_rating=='3']
star_three=star_three.sample(n=20000)
star_four=amazon_reviews[amazon_reviews.star_rating=='4']
star_four=star_four.sample(n=20000)
star_five=amazon_reviews[amazon_reviews.star_rating=='5']
star_five=star_five.sample(n=20000)
sampled_reviews=pd.
↪concat([star_one,star_two,star_three,star_four,star_five],ignore_index=True)
sampled_reviews.shape
```

[4]:
```
(100000, 2)
```

2 Data Cleaning

- Cleaning the data inorder to make the models better, as better data will always result better Prediction.

3 Pre-processing

1. Removing URL.
2. Removing HTML Tags.
3. Removing All the characters except for A-Z&a-z.
4. Removing any html text left with BeautifulSoup Library.
5. Removing Contractions.
6. Removing Punctuation.
7. Removing extra Spaces.
8. Converting the text to lowecase.

[5]:
```
def remove_punctuation(review):
    return ''.join([words for words in review if words not in string.
 ↪punctuation ])
```

[6]:
```
def clean_review(review):
    review = re.sub(r"http\S+", "", review)
    review = re.sub('<.*?>+', '', review)
    review = re.sub('[^A-Za-z]+', ' ', review)
    review = BeautifulSoup(review, "html.parser").get_text()
    review = contractions.fix(review)
    review = remove_punctuation(review)
    review = re.sub("\S*\d\S*", "", review).strip()
    review = review.lower()
    return review
```

3.0.1 Calculating the Average Length of Reviews by Character

[7]:
```
def average_count(sampled_reviews):
    number_of_sentences=len(sampled_reviews)
    return sum(map(len,sampled_reviews))/number_of_sentences
```

[8]:
```
beforeCleaning=average_count(sampled_reviews['review_body'])
sampled_reviews['review_body']=sampled_reviews['review_body'].apply(lambda␣
 ↪review:clean_review(review))
afterCleaning=average_count(sampled_reviews['review_body'])
print("Average character length of the reviews Before and After␣
 ↪Cleaning",beforeCleaning,',',afterCleaning)
```

```
Average character length of the reviews Before and After Cleaning 198.21343 ,
190.29407
```

3.1 remove the stop words

Removing the stop words such as “The” which have no meaning to them.

[9]:
```
from nltk.corpus import stopwords
stop_words = set(stopwords.words('english'))
```

```
beforePreprocessing=average_count(sampled_reviews['review_body'])
sampled_reviews['review_body']= sampled_reviews['review_body'].apply(lambda␣
↪review: " ".join([word for word in review.split() if word not in␣
↪stop_words]))
```

3.2 perform lemmatization

Converting all the words in the reviews to single form so that they can be matched by similarity.

[10]:
```
from nltk.stem import WordNetLemmatizer
lemmatizer = WordNetLemmatizer()
sampled_reviews['review_body']= sampled_reviews['review_body'].apply(lambda␣
↪review: ' '.join(lemmatizer.lemmatize(e) for e in nltk.
↪word_tokenize(review)))
afterPreprocessing=average_count(sampled_reviews['review_body'])
print("Average character length of the reviews Before and After␣
↪Preprocessing",beforePreprocessing,',',afterPreprocessing)
```

```
Average character length of the reviews Before and After Preprocessing 190.29407
, 112.67224
```

4 TF-IDF Feature Extraction

- Converting Reviews to Count Vectors using a concept known as TF-IDF.
- It is the relation between Term Frequency and Inverse Document Frequency.
- Term Frequency is the frequency of the word in a corpus.
- Inverse Document Frequecy is the Frequency of a Word in that particular Document.
- TF - IDF tells about the Frequency of a Word in that Particular Document With Respect to the Entire Corpus.
- N-Grams are combination of words in that particular document. Bi-gram Example (really-appreciate).

[11]:
```
from sklearn.feature_extraction.text import TfidfVectorizer
tf_idf_vect=TfidfVectorizer(ngram_range=(1,3))
final_tf_idf=tf_idf_vect.fit_transform(sampled_reviews['review_body'].values)
```

4.1 Train Test Split

- We split the data in the split of 80:20 which is 80% of the for Training and 20% of the Data for Testing.
- We use train test split in order to train the model and test performance of the model.

[12]:
```
from sklearn.model_selection import train_test_split
xtrain, xtest, ytrain, ytest = train_test_split(final_tf_idf,␣
↪sampled_reviews['star_rating'], test_size = 0.2)
```

4.2 Classification Metrics

A Function that gives you information on Accuracy, Precision, Recall and F1-score.

[13]:
```
from sklearn.metrics import classification_report, confusion_matrix,␣
↪accuracy_score

def metrics(prediction, actual):
    print('\nAccuracy:', accuracy_score(actual, prediction))
    print('\nclassification_report\n')
    print(classification_report(actual, prediction))
```

5 Perceptron

- Perceptron is a two class classification Model.
- It uses a concept of Neuron, which has an activation function, which activates only when crossing a certain threshold.
- We used random_state to randomize the data.
- We used n_jobs to Run Parallel on All Cores.
- The Accuracy of this model on this data is in the range of 45-47.

[14]:
```
from sklearn.linear_model import Perceptron
perceptronModel = Perceptron(random_state=0,n_jobs=-1)
perceptronModel.fit(xtrain, ytrain)
predictions=perceptronModel.predict(xtest)
metrics(predictions, ytest)
```

```
Accuracy: 0.45805

classification_report

              precision    recall  f1-score   support

           1       0.56      0.55      0.56      3965
           2       0.36      0.35      0.36      4051
           3       0.37      0.30      0.33      4044
           4       0.40      0.41      0.41      3932
           5       0.55      0.69      0.61      4008

    accuracy                           0.46     20000
   macro avg       0.45      0.46      0.45     20000
weighted avg       0.45      0.46      0.45     20000
```

6 SVM

- SVM uses a concept of boundary, which helps it to detect and avoid outliers.

- SVM have diffrent form of Kernel: Linear, Poly and more, which can be used fir different firms of data.
- We used C=0.1 which is a regularization parameter.
- The Accuracy of this model on this data is in the range of 50-52.

[15]:
```
from sklearn.svm import LinearSVC
SVM = LinearSVC(C=0.1)
SVM.fit(xtrain, ytrain)
predictions=SVM.predict(xtest)
metrics(predictions, ytest)
```

```
Accuracy: 0.51835

classification_report

              precision    recall  f1-score   support

           1       0.54      0.71      0.62      3965
           2       0.43      0.31      0.36      4051
           3       0.46      0.37      0.41      4044
           4       0.49      0.41      0.45      3932
           5       0.60      0.79      0.68      4008

    accuracy                           0.52     20000
   macro avg       0.50      0.52      0.50     20000
weighted avg       0.50      0.52      0.50     20000
```

7 Logistic Regression

- Logistic Regression tells the likely hood between the classes.
- Logistic regression uses logaritms to compress the data between 0 and 1 which act similar to probabilty.
- We used solver as Saga which is fastest and best for huge data. They are other solvers like newton-cg, lbfgs & more.
- We used random_state to randomize the data.
- We used n_jobs to Run Parallel on All Cores.
- We used max iterations as 200.
- The Accuracy of this model on this data is in the range of 51-53.

[16]:
```
from sklearn.linear_model import LogisticRegression
logisticModel=LogisticRegression(solver='saga',random_state=0,n_jobs=-1,max_iter=200)
logisticModel.fit(xtrain, ytrain)
predictions=logisticModel.predict(xtest)
metrics(predictions, ytest)
```

```
Accuracy: 0.52425

classification_report

              precision    recall  f1-score   support

           1       0.57      0.65      0.61      3965
           2       0.42      0.37      0.40      4051
           3       0.44      0.42      0.43      4044
           4       0.49      0.45      0.47      3932
           5       0.65      0.73      0.69      4008

    accuracy                           0.52     20000
   macro avg       0.52      0.52      0.52     20000
weighted avg       0.52      0.52      0.52     20000
```

8 Naive Bayes

- Naive Bayes uses the principle of Bayes Theorem.
- Bayes Theorem makes use of Conditional Probability.
- Naive Bayes is one of the fastest as it computes probabilities which require little to none computing.
- The Accuracy of this model on this data is in the range of 51-53.

[17]:
```python
from sklearn import naive_bayes
naiveBayesModel = naive_bayes.MultinomialNB()
naiveBayesModel.fit(xtrain, ytrain)
predictions=naiveBayesModel.predict(xtest)
metrics(predictions, ytest)
```

```
Accuracy: 0.5143

classification_report

              precision    recall  f1-score   support

           1       0.59      0.64      0.61      3965
           2       0.42      0.36      0.39      4051
           3       0.44      0.36      0.39      4044
           4       0.44      0.50      0.47      3932
           5       0.65      0.71      0.68      4008

    accuracy                           0.51     20000
   macro avg       0.51      0.52      0.51     20000
weighted avg       0.51      0.51      0.51     20000
```