Homework 1 - Text Representation and Text Classification for Sentiment Analysis

1. Dataset Preparation

In this step of the homework, I have carried out the following tasks:

a. Read the dataset using the pandas read_table function : The dataset is read using the read_table() function which creates the .tsv into the dataframe. The dataset has some rows, which are not in consistency with the dataset format (they have more number of columns then specified). These lines have been skipped while reading the dataset.

b. Create a three-class classification task: In this step, I have first dropped all the columns except 'review_body' and 'star_rating'. I have further dropped the rows in which the 'review_body' or 'star_rating' is null. To form the 3-class classification problem, I have simply checked if int(star_rating) falls in 1,2,3,4,5 and assigned class 1 for 1 and 2, 2 for 3 and 3 for 4 and 5. To select 20000 random instances for every class, I have used sample function to create three datasets - each having 20000 rows for every class. Concatenating these datasets results in a dataset of 60000 rows (every class has 20000 samples). I have not split the dataset into train and test dataset at this stage because I preferred cleaning and preprocessing the dataset first and then splitting it.

2. Data Cleaning

For the task of data cleaning, I have carried out the following steps. The task has been carried out using python's regular expressions.

a. .lower() function has converted the 'review_body' to lowercase.

b. HTML and URL tags have been replaced by ‘’ using the replace function. Regular expressions has been used to represent the patterns of the URL and HTML tags
c. Contractions is performed using the contractions library of python. It has a collection of all contractions that needed to be done.
d. Following contractions, I have replaced the non-alphabetic characters with a empty string using regex.
e. Extra spaces have been removed by splitting the review_body at white space and then joining them with a single space

RESULT:-
270.51321666666666 , 259.66218333333336

3. PRE-PROCESSING

There are two major tasks that have been carried out for the preprocessing of the ‘review_body’ text - removal of stopwords and lemmatization.

a. Removal of stopwords - The nltk package of python provides the stopwords. Stopwords are commonly used words that appear in the text e.g is, this etc. These words are removed because for the purpose of sentiment analysis, these words do not contribute to the sentiment of the sentence. I have removed the stopwords from the ‘review_body’ and used the english stopwords from the nltk stopwords.
b. Lemmatization - Lemmatization is simply reducing the word to their lemma i.e the root. The words studying, studies all reduce to their lemma, study. WordNetLemmatizer from the nltk package is used for lemmatization. The lemmatization has a pos parameter which accepts the part of speech.

RESULT:-
259.66218333333336 , 157.6374920751443

4. Feature Extraction

After data cleaning and preprocessing, some rows in 'review_body' are empty. I have considered to remove these rows from the dataset prior to feature extraction. Since the machine learning algorithms accept numerical feature vectors, we cannot use 'review_body' as a feature. We basically have to extract features from the text in the 'review_body'. One of the most common feature extraction techniques for text classification is TF-IDF vectorization. At this step, I have first split the dataset into train and test dataset. The TF-IDF vectorization is fit on the training data and then both the train and test data are transformed by the fit TF-IDF vectorizer. After this step we have X_train, y_train, X_test and y_test as has been expected by the assignment.

5. Perceptron

For this step, I have used a single layer perceptron from the sklearn.linear_model. A perceptron uses the perceptron algorithm for weight and bias updation as discussed in the lectures. Since the homework is focussed on data preparation and cleaning, I have not tuned any parameters for boosting the performance of the perceptron

RESULTS:

0.6221631568186465 , 0.7598002496878902 , 0.6841276978417267

0.6368214885282597 , 0.2894201424211597 , 0.39797167336947015

0.635593220338983 , 0.833127622809183 , 0.7210768080333297

0.6315259552286298 , 0.6274493383060776 , 0.6010587264148421

6. SVM

Support Vector Machines are maximum margin classifiers. They build a hyperplane between classes such that the margin between them is maximum. These maximum margin points are called support vectors. For this part of the homework, I have used the sklearn inbuilt LinearSVC.

RESULTS:

0.6802257230190454 , 0.7223470661672909 , 0.7006539113586825

0.6 , 0.5439979654120041 , 0.5706282513005202

0.7448441247002399 , 0.7667242656134288 , 0.7556258362729595

0.6750232825730951 , 0.677689765730908 , 0.6756359996440541

7. Logistic Regression

Logistic Regression is a linear classifier. For the logistic regression we have used the inbuilt sklearn linear model implementation, LogisticRegression.

RESULTS:

0.6900457941672692 , 0.7148564294631711 , 0.7022320333578612

0.5916120576671036 , 0.5740081383519837 , 0.5826771653543308

0.7604373757455268 , 0.7553690446803258 , 0.7578947368421052

0.6806984091932998 , 0.6814112041651601 , 0.6809346451847658

8. Multinomial Naive Bayes

Naive Bayes is a generative model and it models on joint probabilities. It calculates the P(label | features) based on P(features, labels) and P(labels). It is based upon the bayes rule. We have used the sklearn implementation of Multinomial Naive Bayes.

RESULTS:

0.7038021802712044 , 0.6609238451935081 , 0.6816894154004635

0.5615580802225829 , 0.615971515768057 , 0.5875075803517285

0.7442514052120592 , 0.7190817082201926 , 0.7314500941619586

0.6698705552352822 , 0.6653256897272526 , 0.6668823633047168

HW1_CSCI544_Naseela_Pervez

January 26, 2023

[1]:
```
# Python Version :
import pandas as pd
import numpy as np
import nltk
nltk.download('wordnet', quiet = True)
import re
from bs4 import BeautifulSoup
```

0.1 Read Data

[2]:
```
data = pd.read_table('https://s3.amazonaws.com/amazon-reviews-pds/tsv/
↪amazon_reviews_us_Beauty_v1_00.tsv.gz', on_bad_lines = 'skip',verbose =␣
↪False)
```

```
/usr/local/lib/python3.8/dist-packages/IPython/core/interactiveshell.py:3326:
DtypeWarning: Columns (7) have mixed types.Specify dtype option on import or set
low_memory=False.
  exec(code_obj, self.user_global_ns, self.user_ns)
```

0.2 Keep Reviews and Ratings

[3]:
```
new_data = data[['review_body','star_rating']]
```

We form three classes and select 20000 reviews randomly from each class.

[4]:
```
new_data = new_data.dropna()
```

[5]:
```
def get_new_rating(rating):
  if int(rating) == 1 or int(rating) == 2:
    return 1
  elif int(rating) == 3:
    return 2
  else:
    return 3

new_data['class'] = new_data["star_rating"].apply(get_new_rating)
```

[6]:
```
random_sample_class1 = new_data[new_data['class']==1].
 ↪sample(n=20000,replace=False, random_state=42)
random_sample_class2 = new_data[new_data['class']==2].
 ↪sample(n=20000,replace=False, random_state=42)
random_sample_class3 = new_data[new_data['class']==3].
 ↪sample(n=20000,replace=False, random_state=42)

class_samples = [random_sample_class1,random_sample_class2,random_sample_class3]

df_class = pd.concat(class_samples).reset_index(drop=True)
```

1 Data Cleaning

[7]:
```
before_cleaning = df_class['review_body'].str.len().mean()
```

[8]:
```
# Convert all reviews to lowercase
df_class['review_body'] = df_class['review_body'].str.lower()
```

[9]:
```
# Remove URLs
import re
url_pattern = r"http[s]\S+"
df_class['review_body'] = df_class['review_body'].replace(to_replace =␣
 ↪url_pattern, value = '',regex=True)
```

[10]:
```
# Remove HTML tags
html_pattern = r"<[^<]*>"
df_class['review_body'] = df_class['review_body'].replace(to_replace =␣
 ↪html_pattern, value = '',regex=True)
```

[11]:
```
# Perform contraction
import contractions
```

[12]:
```
df_class['review_body'] = df_class['review_body'].apply(contractions.fix)
```

[13]:
```
# Remove non-alphabetic characters
apha_str = r"[^a-z ]+"
df_class['review_body'] = df_class['review_body'].replace(to_replace =␣
 ↪apha_str, value = '',regex=True,)
```

[14]:
```
# Remove Extra Spaces
df_class['review_body'] = df_class['review_body'].apply(lambda x: ' '.join(x.
 ↪split()))
```

[15]:
```
after_cleaning = df_class['review_body'].str.len().mean()

print(before_cleaning,',',after_cleaning)
```

270.51321666666666 , 259.66218333333336

2 Pre-processing

2.1 remove the stop words

[16]:
```
before_preprocessing = df_class['review_body'].str.len().mean()
```

[17]:
```
nltk.download('stopwords', quiet = True)
from nltk.corpus import stopwords

stopwords_eng = stopwords.words('english')

# stopwords_all = stopwords_eng + extra_words
```

[18]:
```
ignore_stopwords = ["no", "not",'nor','very']
for kw in ignore_stopwords:
  stopwords_eng.remove(kw)
```

[19]:
```
df_class['review_body'] = df_class['review_body'].apply(lambda x: ' '.join([w␣
 ↪for w in x.split() if w not in stopwords_eng]))
```

[20]:
```
##strip again
df_class['review_body'] = df_class['review_body'].apply(lambda x:x.strip())
```

[21]:
```
## remove empty reviews
df_class = df_class[df_class['review_body']!='']
```

2.2 perform lemmatization

[22]:
```
# pos tagging
nltk.download('wordnet', quiet = True)
nltk.download('averaged_perceptron_tagger', quiet = True)
from nltk.corpus import wordnet
from nltk import pos_tag

def mapper(tags):

  if tags.startswith('N'):
    return wordnet.NOUN
  elif tags.startswith('VB'):
    return wordnet.VERB
  elif tags.startswith('JJ'):
    return wordnet.ADJ
  elif tags.startswith('RB'):
```

```
    return wordnet.ADV
  else:
    return None
```

[23]:
```
nltk.download('omw-1.4', quiet = True)
from nltk.stem import WordNetLemmatizer
```

[24]:
```
df_class[df_class["review_body"].apply(lambda x:len(x.strip()))==0]
```

[24]:
```
Empty DataFrame
Columns: [review_body, star_rating, class]
Index: []
```

[25]:
```
word_lemmatizer = WordNetLemmatizer()

def lemmatization(text):


  words = text.split()
  # print(text, words)
  tags_out =  pos_tag(words)
  words, tags = list(zip(*tags_out))

  return " ".join([word_lemmatizer.lemmatize(word, pos=mapper(tag)) if␣
↪mapper(tag) is not None else word for word, tag in zip(words, tags)])
```

[26]:
```
df_class['review_body'] = df_class['review_body'].apply(lemmatization)
```

[27]:
```
after_preprocessing =  df_class['review_body'].str.len().mean()

print(before_preprocessing, ',', after_preprocessing)
```

```
259.66218333333336 , 157.6374920751443
```

3 TF-IDF Feature Extraction

[28]:
```
# Train test split

from sklearn.model_selection import train_test_split

# X = data_df.drop(['class'],axis=1)
X = df_class['review_body']

# y = data_df['class']
y = df_class['class']

X_train,X_test,y_train,y_test = train_test_split(X,y,test_size = 0.2)
```

[29]:
```
from sklearn.feature_extraction.text import TfidfVectorizer

tfidf = TfidfVectorizer(max_features = 1000,ngram_range=(1,5))

features_train = tfidf.fit_transform(X_train).toarray()

features_test = tfidf.transform(X_test).toarray()
```

[30]:
```
X_train = pd.DataFrame(data = features_train)
X_test = pd.DataFrame(data = features_test)
```

4 Perceptron

[31]:
```
from sklearn.linear_model import Perceptron

# perceptron_classifier = Perceptron(alpha=0.8, max_iter=10, verbose=1, eta0=1,␣
↪tol=None)
perceptron_classifier = Perceptron()
perceptron_classifier.fit(X_train,y_train)
```

[31]:
```
Perceptron()
```

[32]:
```
preds = perceptron_classifier.predict(X_test)

from sklearn.metrics import classification_report, precision_score
```

[33]:
```
rep = classification_report(y_test, preds,output_dict=True)
```

[34]:
```
print("\n",rep['1']['precision'],',',rep['1']['recall'],',',rep['1']['f1-score'])
print("\n",rep['2']['precision'],',',rep['2']['recall'],',',rep['2']['f1-score'])
print("\n",rep['3']['precision'],',',rep['3']['recall'],',',rep['3']['f1-score'])
print("\n",rep['macro avg']['precision'],',',rep['macro␣
↪avg']['recall'],',',rep['macro avg']['f1-score'])
```

0.6221631568186465 , 0.7598002496878902 , 0.6841276978417267

0.6368214885282597 , 0.2894201424211597 , 0.39797167336947015

0.635593220338983 , 0.833127622809183 , 0.7210768080333297

0.6315259552286298 , 0.6274493383060776 , 0.6010587264148421

5 SVM

[35]:
```
from sklearn.svm import LinearSVC
svc = LinearSVC()
svc.fit(X_train,y_train)
```

[35]:
```
LinearSVC()
```

[36]:
```
preds = svc.predict(X_test)

rep = classification_report(y_test, preds,output_dict=True)
```

[37]:
```
print("\n",rep['1']['precision'],',',rep['1']['recall'],',',rep['1']['f1-score'])
print("\n",rep['2']['precision'],',',rep['2']['recall'],',',rep['2']['f1-score'])
print("\n",rep['3']['precision'],',',rep['3']['recall'],',',rep['3']['f1-score'])
print("\n",rep['macro avg']['precision'],',',rep['macro␣
 ↪avg']['recall'],',',rep['macro avg']['f1-score'])
```

0.6802257230190454 , 0.7223470661672909 , 0.7006539113586825

0.6 , 0.5439979654120041 , 0.5706282513005202

0.7448441247002399 , 0.7667242656134288 , 0.7556258362729595

0.6750232825730951 , 0.677689765730908 , 0.6756359996440541

6 Logistic Regression

[38]:
```
from sklearn.linear_model import LogisticRegression

logreg_classifier = LogisticRegression(max_iter = 1000)

logreg_classifier.fit(X_train,y_train)
```

[38]:
```
LogisticRegression(max_iter=1000)
```

[39]:
```
preds = logreg_classifier.predict(X_test)


rep = classification_report(y_test, preds,output_dict=True)
```

[40]:
```
print("\n",rep['1']['precision'],',',rep['1']['recall'],',',rep['1']['f1-score'])
print("\n",rep['2']['precision'],',',rep['2']['recall'],',',rep['2']['f1-score'])
print("\n",rep['3']['precision'],',',rep['3']['recall'],',',rep['3']['f1-score'])
print("\n",rep['macro avg']['precision'],',',rep['macro␣
 ↪avg']['recall'],',',rep['macro avg']['f1-score'])
```

```
0.6900457941672692 , 0.7148564294631711 , 0.7022320333578612

0.5916120576671036 , 0.5740081383519837 , 0.5826771653543308

0.7604373757455268 , 0.7553690446803258 , 0.7578947368421052

0.6806984091932998 , 0.6814112041651601 , 0.6809346451847658
```

7 Naive Bayes

[41]:
```
from sklearn.naive_bayes import MultinomialNB

gnb = MultinomialNB()

gnb.fit(X_train,y_train)
```

[41]:
```
MultinomialNB()
```

[42]:
```
preds = gnb.predict(X_test)


rep = classification_report(y_test, preds,output_dict=True)
```

[43]:
```
print("\n",rep['1']['precision'],',',rep['1']['recall'],',',rep['1']['f1-score'])
print("\n",rep['2']['precision'],',',rep['2']['recall'],',',rep['2']['f1-score'])
print("\n",rep['3']['precision'],',',rep['3']['recall'],',',rep['3']['f1-score'])
print("\n",rep['macro avg']['precision'],',',rep['macro␣
 ↪avg']['recall'],',',rep['macro avg']['f1-score'])
```

```
0.7038021802712044 , 0.6609238451935081 , 0.6816894154004635

0.5615580802225829 , 0.615971515768057 , 0.5875075803517285

0.7442514052120592 , 0.7190817082201926 , 0.7314500941619586
```

```
0.6698705552352822 , 0.6653256897272526 , 0.6668823633047168
```

[43]: