NLP Assignment 1 Report

-Rhea Anand (3720700662)

1. Dataset Preparation

The data is read into a dataframe text_df
\t is used for specifying that the tsv is tab separated data.

Q. Report Sample Reviews along with corresponding rating from the data frame:

```
 ************ 3 Sample Reviews ***********

                                                      review_body  star_rating
2250839  This was outstanding. Looked so great on my ta...          5.0
1383289  It's a timer. It counts down time. I also use ...          4.0
3685721  I was very satisfied the product and the speed...          5.0
```

Q. Report how many times each rating occurs in the dataset:

```
 ************ Frequency of each rating  ************

5.0    3124595
4.0     731701
1.0     426870
3.0     349539
2.0     241939
Name: star_rating, dtype: int64
```

Q. Below we see how many times each sentiment (positive, negative, neutral) occur

```
************ Frequency of each sentiment ************

positive    3856296
negative     668809
neutral      349539
Name: sentiment, dtype: int64
```

Q. The frequencies after removing neutral sentiment:

```
**** Frequency of each sentiment after removing neutral reviews ****

positive    3856296
negative     668809
Name: sentiment, dtype: int64
```

2. Data Cleaning

First convert the reviews to lower case using .lower() function
Then use beautifulsoup library to remove HTML tags and regular expression remove the
Using regular expressions remove the digits and special characters
Using contractions library convert the contractions for example I'm to I am
Convert to lower case again because i'm gets converted to I am.

Q. The Average character length before data cleaning:

```
Avg. character length before cleaning : 322.651215
```

3. Preprocessing

Using NLTK library stopwords collection to remove stop words by checking if any wor d in review is present in the stop word list
Using NLTK library Wordnetlemmatizer lemmatize each word in the dataset

Q. Report the Average character length before data cleaning and preproc essing:

```
Avg. character length before cleaning : 322.651215
```

Q. Report three sample reviews before data cleaning + preprocessing:

```
Avg. character length before cleaning : 322.651215

************ 3 Sample Reviews before cleaning ***********

4096630    I saw this kettle in a friend's house (she rec...
3426962    Heavy duty and non stick. Just the right size ...
1950150    Not what  I thought I needed but would be grea...
Name: review_body, dtype: object
```

Q. Report the Average character length after data-cleaning and before p reprocessing:

```
Avg. character length before preprocessing 309.412685
```

Q. Report the Average character length after data-cleaning and preprocessing
Print 3 sample reviews after preprocessing

```
Avg. character length after data cleaning + preprocessing : 189.412555

 ******* 3 Sample Reviews after data cleaning + preprocessing ********

                                                 review_body  star_rating  label
3252743  happy drawer insert shave sand paper bit fit d...          5.0      1
1971228  produce ton fine grind setting consistency get...          2.0      0
4587690  bought plate also black small salad dessert pl...          1.0      0
```

4. Feature Extraction

Using Sklearn train_test_split function split the data into 80% training and 20% test s ets.
Test_size = 0.2 specifies 20% of the dataset be taken into the test set and the rest 80% be taken into the training set .
TF-IDF is used for converting the word reviews into vector form input vectors.
fit and transform on train data and only transform test data respectively.
By giving the max_features we limit each review getting converted to a vector of max 2000 words/features.
min_df specifies min frequency of a word selected as a feature i.e the word has to occ ur atleast once
max_df ensures that a word used in more than 70% of the reviews is not considered a s a feature

5. Perceptron

Train a perceptron on normalized data.
Normalize the data using standardScaler
With_mean = False parameter is required because the data is a sparse dataset
Training parameters : 100 iterations and a learning rate (eta0) of 0.1

Q. Report the Accuracy, Precision, Recall, and f1-score on train data o f the perceptron model:

```
 ************ Evaluation metrics on training data ***********

Training Accuracy: 0.84600
Training F1 Score: 0.84675
Training Precision Score: 0.84122
Training Recall Score: 0.85235
```

Q. Report the Accuracy, Precision, Recall, and f1-score on test data of the perceptron model:

```
 ************ Evaluation metrics on test data ***********

Testing Accuracy: 0.84325
Testing F1 Score: 0.84506
Testing Precision Score: 0.84107
Testing Recall Score: 0.84909
```

6. SVM

Train an SVM model using the SKLearn implementation.
LinearSVC() is used for handling larger dataset.

Q. Report the Accuracy, Precision, Recall, and f1-score on train data o f the SVM model:

```
************ Evaluation metrics on training data ***********

Training Accuracy: 0.89763
Training F1 Score: 0.89708
Training Precision Score: 0.90037
Training Recall Score: 0.89382
```

Q. Report the Accuracy, Precision, Recall, and f1-score on test data of the SVM model:

```
************ Evaluation metrics on test data ***********

Testing Accuracy: 0.89470
Testing F1 Score: 0.89523
Testing Precision Score: 0.89684
Testing Recall Score: 0.89363
```

7. Logistic Regression

Using the Sklearn implementation of logistic regression model with default paramete rs fit a model on the training data.

Q. Report the Accuracy, Precision, Recall, and f1-score on train data o f the Logistic Regression model:

```
************ Evaluation metrics on training data ***********

Training Accuracy: 0.89706
Training F1 Score: 0.89635
Training Precision Score: 0.90100
Training Recall Score: 0.89174
```

Q. Report the Accuracy, Precision, Recall, and f1-score on test data of the Logistic Regression model:

```
************ Evaluation metrics on test data ***********

Testing Accuracy: 0.89570
Testing F1 Score: 0.89606
Testing Precision Score: 0.89911
Testing Recall Score: 0.89304
```

8. Multinomial Naïve Bayes

Using the Sklearn implementation of Multinomial Naïve Bayes model with default pa rameters fit a model on the training data.

Q. Report the Accuracy, Precision, Recall, and f1-score on train data o f the Multinomial Naïve Bayes model:

```
************ Evaluation metrics on training data ***********

Training Accuracy: 0.86606
```

```
Training F1 Score: 0.86594
Training Precision Score: 0.86526
Training Recall Score: 0.86661
```

Q. Report the Accuracy, Precision, Recall, and f1-score on test data of the Multinomial Naïve Bayes model:

```
 ************ Evaluation metrics on test data ***********

Testing Accuracy: 0.86460
Testing F1 Score: 0.86583
Testing Precision Score: 0.86387
Testing Recall Score: 0.86781
```

assignment1.py file sample output:

- Statistics of three classes (with comma between them)
- Average length of reviews before and after data cleaning (with comma between them)
- Average length of reviews before and after data preprocessing (with comma between them)
- Accuracy, Precision, Recall, and f1-score for training and testing split (in the mentioned order) for Perceptron (with comma between them)
- Accuracy, Precision, Recall, and f1-score for training and testing split (in the mentioned order) for SVM
- Accuracy, Precision, Recall, and f1-score for training and testing split (in the mentioned order) for Logistic Regression (with comma between them)
- Accuracy, Precision, Recall, and f1-score for training and testing split (in the mentioned order) for Naive Bayes (with comma between them)

3856296 , 668809 , 349539
322.651215 , 309.412685
309.412685 , 189.412555
0.84600, 0.84122, 0.85235, 0.84675, 0.84325, 0.84107, 0.84909, 0.84506
0.89763, 0.90037, 0.89382, 0.89708, 0.89470, 0.89684, 0.89363, 0.89523
0.89706, 0.90100, 0.89174, 0.89635, 0.89570, 0.89911, 0.89304, 0.89606
0.86606, 0.86526, 0.86661, 0.86594, 0.86460, 0.86387, 0.86781, 0.86583

HW1-CSCI544

September 9, 2021

[19]:
```
import pandas as pd
import numpy as np
import nltk    # stemming, lemmatization etc
nltk.download('wordnet')
nltk.download('stopwords')
import re      # for removing urls etc
import urllib
import contractions # won't to will not, don't to do not
from bs4 import BeautifulSoup # remove html content
import sklearn
import warnings
warnings.filterwarnings("ignore")
```

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

[20]:
```
#! pip install bs4 # in case you don't have it installed

# Dataset: https://s3.amazonaws.com/amazon-reviews-pds/tsv/
 ↪amazon_reviews_us_Kitchen_v1_00.tsv.gz
```

0.1 Read Data

[21]:
```
text_df = pd.read_csv('data.tsv', error_bad_lines = False, sep = '\t',␣
 ↪warn_bad_lines=False)
# \t because its tsv file
# it will throw an error if number of rows are not in correct allignment
# error_bad_lines = False will ignore such lines, when True (default)it will␣
 ↪throw an error
```

0.2 Keep Reviews and Ratings

[22]:
```
# Keep rating and reviews only
text_df = text_df[['review_body','star_rating']]

# drop na rows from ratings and reviews
text_df.dropna(subset = ["review_body"],inplace=True)
text_df.dropna(subset = ["star_rating"],inplace=True)

# Print 3 reviews
print("\n ************ 3 Sample Reviews ***********\n")
print(text_df.sample(3))

print("\n ************ Frequency of each rating  ************\n")

# Print frequency of the rating
count = text_df['star_rating'].value_counts() # how many rows for each rating␣
↪1,2,3,4,5
print(count)
```

```
 ************ 3 Sample Reviews ***********

                                               review_body  star_rating
2250839  This was outstanding. Looked so great on my ta…          5.0
1383289  It's a timer. It counts down time. I also use …          4.0
3685721  I was very satisfied the product and the speed…          5.0

 ************ Frequency of each rating  ************

5.0    3124595
4.0     731701
1.0     426870
3.0     349539
2.0     241939
Name: star_rating, dtype: int64
```

1 Labelling Reviews:

1.1 The reviews with rating 4,5 are labelled to be 1 and 1,2 are labelled as 0. Discard the reviews with rating 3’

[23]:
```
# 1/2 rating negative sentiment
# if 3 discard because its neutral
# if its 4/5 positive sentiment
```

```
text_df['label'] = np.where(text_df["star_rating"] >= 4, 1, 0)     # create␣
↪positive and negative sentiment label
text_df['sentiment'] = np.where(text_df["star_rating"] >= 4, "positive",␣
↪"negative")     # create positive and negative sentiment label
text_df['label'] = np.where(text_df["star_rating"] == 3, -1,text_df['label'])
text_df['sentiment'] = np.where(text_df["star_rating"] == 3,␣
↪"neutral",text_df['sentiment'])
text_df = text_df[['star_rating','review_body','label','sentiment']]          ␣
↪# copying to a new data frame


count = text_df['sentiment'].value_counts()                          # counting␣
↪frequency of each label
print("\n************ Frequency of each sentiment ************\n")
print(count)

text_df = text_df[text_df.star_rating != 3]                            # delete␣
↪the rows with neutral rating 3.
count = text_df['sentiment'].value_counts()                          # counting␣
↪frequency of each label after neutral is dropped
print("\n**** Frequency of each sentiment after removing neutral reviews␣
↪****\n")
print(count)
text_df = text_df[['review_body','star_rating','label']]
```

```
************ Frequency of each sentiment ************

positive    3856296
negative     668809
neutral      349539
Name: sentiment, dtype: int64

**** Frequency of each sentiment after removing neutral reviews ****

positive    3856296
negative     668809
Name: sentiment, dtype: int64
```

We select 200000 reviews randomly with 100,000 positive and 100,000 negative reviews.

[24]:
```
sm0 = text_df.label[text_df.label.eq(0)].sample(100000,random_state=80).index  ␣
↪ #randomly select 100000 positive reviews
sm1 = text_df.label[text_df.label.eq(1)].sample(100000, random_state=80).index ␣
↪  #randomly select 100000 negative reviews
```

```
text_df = text_df.loc[sm0.union(sm1)]     # combine into one dataset

avg_char = text_df['review_body'].apply(lambda a :len(str(a))).mean()  #␣
↪finding avg no. of characters in a review
print(" Avg. character length before cleaning :",avg_char)

print("\n ************ 3 Sample Reviews before cleaning ***********\n")
print(text_df['review_body'].sample(3))
```

```
 Avg. character length before cleaning : 322.651215

 ************ 3 Sample Reviews before cleaning ***********

4096630    I saw this kettle in a friend's house (she rec…
3426962    Heavy duty and non stick. Just the right size …
1950150    Not what  I thought I needed but would be grea…
Name: review_body, dtype: object
```

2 Data Cleaning

2.1 Convert the all reviews into the lower case.

[25]:
```
text_df["review_body"] = text_df["review_body"].str.lower()     # convert␣
↪everything to lower case
```

2.2 remove the HTML and URLs from the reviews

[26]:
```
# Removing html tags using beautiful soup like <br> tags
text_df["review_body"] = text_df["review_body"].apply(lambda x:␣
↪BeautifulSoup(str(x)).get_text())

# Removing urls from reviews
text_df["review_body"] = text_df["review_body"].apply(lambda x: re.
↪sub(r'\s*(https?://|www\.)+\S+(\s+|$)', " ", str(x), flags=re.UNICODE))
```

2.3 remove non-alphabetical characters

[27]:
```
# Removing Digits from the review_body
text_df["review_body"] = text_df["review_body"].apply(lambda x: re.
↪sub(r"[^\D']+", " ", str(x), flags=re.UNICODE)) # remove all numbers

# Removing Special Characters
text_df["review_body"] = text_df["review_body"].apply(lambda x: re.
↪sub(r"[^\w']+", " ", str(x), flags=re.UNICODE)) # remove all special␣
↪characters
```

2.4 Remove the extra spaces between the words

[28]:
```
#remove more than one spaces
text_df["review_body"] = text_df["review_body"].apply(lambda x: re.sub(r'\s+',' ', str(x), flags = re.UNICODE))
```

2.5 perform contractions on the reviews.

[29]:
```
def contractionfunction(s):
    s = s.apply(lambda x: contractions.fix(x))
    return s

text_df_onecol = contractionfunction(text_df["review_body"])
text_df["review_body"] = text_df_onecol
text_df["review_body"] = text_df["review_body"].str.lower()      # convert everything to lower case again because contractions adds I
```

3 Pre-processing

3.1 remove the stop words

[30]:
```
avg_char = text_df['review_body'].apply(lambda a :len(str(a))).mean()   # finding avg no. of characters in a review
print("Avg. character length before preprocessing", avg_char)

from nltk.corpus import stopwords
# storing all the stop words
stop_words = stopwords.words('english')

# remove stop words from each review
text_df["review_body"] = text_df["review_body"].apply(lambda x: " ".join([item for item in str(x).split() if item not in stop_words]))
```

```
Avg. character length before preprocessing 309.412685
```

3.2 perform lemmatization

[31]:
```
from nltk.stem import WordNetLemmatizer
lemmatizer = WordNetLemmatizer()

text_df["review_body"] = text_df["review_body"].apply(lambda x: " ".join([lemmatizer.lemmatize(item)  for item in str(x).split()]))

avg_char_after = text_df['review_body'].apply(lambda x : (len(str(x)))).mean()
print("Avg. character length after data cleaning + preprocessing :", avg_char_after)
```

```
print("\n ************ 3 Sample Reviews after data cleaning + preprocessing␣
↪***********\n")
print(text_df.sample(3))
```

```
Avg. character length after data cleaning + preprocessing : 189.412555

 ************ 3 Sample Reviews after data cleaning + preprocessing ***********

                                              review_body  star_rating  label
3252743  happy drawer insert shave sand paper bit fit d…         5.0      1
1971228  produce ton fine grind setting consistency get…         2.0      0
4587690  bought plate also black small salad dessert pl…         1.0      0
```

4 TF-IDF Feature Extraction

[32]:
```
from sklearn.feature_extraction.text import TfidfVectorizer

# fit and transform on train data and only transform test data respectively.
# converting each review to a vector of max 2000 words
# min_df specifies min frequency of a word selected as a feature i.e the word␣
↪has to occur atleast once
# max_df ensures that a word used in more than 70% of the reviews is not␣
↪considered as a feature

from sklearn.model_selection import train_test_split
#Split train and test into 80 20 split
X_train, X_test, y_train, y_test =␣
↪train_test_split(text_df["review_body"],text_df["label"], test_size=0.2,␣
↪random_state=90)

tfidfconverter = TfidfVectorizer(max_features=2000, min_df=1, max_df=0.7)

# fit decides the features based on the train dataset whose retrictions were␣
↪described in the above TfidfVectorizer function
X_train = tfidfconverter.fit_transform(X_train)
X_test = tfidfconverter.transform(X_test)
```

5 Perceptron

[33]:
```
from sklearn.linear_model import Perceptron
from sklearn.preprocessing import StandardScaler
from sklearn.metrics import accuracy_score,f1_score,precision_score,recall_score

# standard scalera is a function used to normalize the review vectors
sc = StandardScaler(with_mean=False)
```

```python
# using the normalizing function to create a normalized training dataset
X_train_std = sc.fit_transform(X_train)

# normalize the test data using the same scaler
X_test_std = sc.transform(X_test)

# Create a perceptron object with the parameters: 40 iterations (epochs) over
↪the data, and a learning rate of 0.1
ppn = Perceptron(max_iter=100, eta0=0.1, random_state=0)

# Train the perceptron
ppn.fit(X_train_std, y_train)


print("\n ************ Evaluation metrics on training data ***********\n")

y_pred_train = ppn.predict(X_train_std)

print('Training Accuracy: %.5f' % accuracy_score(y_train, y_pred_train))
print('Training F1 Score: %.5f' % f1_score(y_train, y_pred_train))
print('Training Precision Score: %.5f' % precision_score(y_train, y_pred_train))
print('Training Recall Score: %.5f' % recall_score(y_train, y_pred_train))

y_pred_test = ppn.predict(X_test_std)

print("\n ************ Evaluation metrics on test data ***********\n")

print('Testing Accuracy: %.5f' % accuracy_score(y_test, y_pred_test))
print('Testing F1 Score: %.5f' % f1_score(y_test, y_pred_test))
print('Testing Precision Score: %.5f' % precision_score(y_test, y_pred_test))
print('Testing Recall Score: %.5f' % recall_score(y_test, y_pred_test))
```

```
 ************ Evaluation metrics on training data ***********

Training Accuracy: 0.84600
Training F1 Score: 0.84675
Training Precision Score: 0.84122
Training Recall Score: 0.85235

 ************ Evaluation metrics on test data ***********

Testing Accuracy: 0.84325
Testing F1 Score: 0.84506
Testing Precision Score: 0.84107
Testing Recall Score: 0.84909
```

6 SVM

[34]:
```python
# from sklearn import svm

from sklearn import svm

#Create a svm Classifier
svm_clf = svm.LinearSVC() # Linear Kernel

#Train the model using the training sets
svm_clf.fit(X_train, y_train)



print("\n ************ Evaluation metrics on training data ***********\n")

y_pred_train = svm_clf.predict(X_train)

print('Training Accuracy: %.5f' % accuracy_score(y_train, y_pred_train))
print('Training F1 Score: %.5f' % f1_score(y_train, y_pred_train))
print('Training Precision Score: %.5f' % precision_score(y_train, y_pred_train))
print('Training Recall Score: %.5f' % recall_score(y_train, y_pred_train))

y_pred_test = svm_clf.predict(X_test)

print("\n ************ Evaluation metrics on test data ***********\n")

print('Testing Accuracy: %.5f' % accuracy_score(y_test, y_pred_test))
print('Testing F1 Score: %.5f' % f1_score(y_test, y_pred_test))
print('Testing Precision Score: %.5f' % precision_score(y_test, y_pred_test))
print('Testing Recall Score: %.5f' % recall_score(y_test, y_pred_test))
```

```
 ************ Evaluation metrics on training data ***********

Training Accuracy: 0.89763
Training F1 Score: 0.89708
Training Precision Score: 0.90037
Training Recall Score: 0.89382

 ************ Evaluation metrics on test data ***********

Testing Accuracy: 0.89470
Testing F1 Score: 0.89523
Testing Precision Score: 0.89684
Testing Recall Score: 0.89363
```

7 Logistic Regression

[35]:
```python
from sklearn.linear_model import LogisticRegression

# instantiate the model (using the default parameters)
logreg = LogisticRegression()

# fit the model with data
logreg.fit(X_train,y_train)


print("\n ************ Evaluation metrics on training data ***********\n")

y_pred_train = logreg.predict(X_train)

print('Training Accuracy: %.5f' % accuracy_score(y_train, y_pred_train))
print('Training F1 Score: %.5f' % f1_score(y_train, y_pred_train))
print('Training Precision Score: %.5f' % precision_score(y_train, y_pred_train))
print('Training Recall Score: %.5f' % recall_score(y_train, y_pred_train))

y_pred_test = logreg.predict(X_test)

print("\n ************ Evaluation metrics on test data ***********\n")

print('Testing Accuracy: %.5f' % accuracy_score(y_test, y_pred_test))
print('Testing F1 Score: %.5f' % f1_score(y_test, y_pred_test))
print('Testing Precision Score: %.5f' % precision_score(y_test, y_pred_test))
print('Testing Recall Score: %.5f' % recall_score(y_test, y_pred_test))
```

```
 ************ Evaluation metrics on training data ***********

Training Accuracy: 0.89706
Training F1 Score: 0.89635
Training Precision Score: 0.90100
Training Recall Score: 0.89174

 ************ Evaluation metrics on test data ***********

Testing Accuracy: 0.89570
Testing F1 Score: 0.89606
Testing Precision Score: 0.89911
Testing Recall Score: 0.89304
```

8 Naive Bayes

[36]:
```python
from sklearn.naive_bayes import MultinomialNB

#Create a Multinomial Naive Bayes model with default parameters
model = MultinomialNB()

# Train the model using the training set
model.fit(X_train, y_train)

#Predict Output
y_pred = model.predict(X_test) # 0:Overcast, 2:Mild


print("\n ************ Evaluation metrics on training data ***********\n")

y_pred_train = model.predict(X_train)

print('Training Accuracy: %.5f' % accuracy_score(y_train, y_pred_train))
print('Training F1 Score: %.5f' % f1_score(y_train, y_pred_train))
print('Training Precision Score: %.5f' % precision_score(y_train, y_pred_train))
print('Training Recall Score: %.5f' % recall_score(y_train, y_pred_train))

y_pred_test = model.predict(X_test)

print("\n ************ Evaluation metrics on test data ***********\n")

print('Testing Accuracy: %.5f' % accuracy_score(y_test, y_pred_test))
print('Testing F1 Score: %.5f' % f1_score(y_test, y_pred_test))
print('Testing Precision Score: %.5f' % precision_score(y_test, y_pred_test))
print('Testing Recall Score: %.5f' % recall_score(y_test, y_pred_test))
```

```
 ************ Evaluation metrics on training data ***********

Training Accuracy: 0.86606
Training F1 Score: 0.86594
Training Precision Score: 0.86526
Training Recall Score: 0.86661

 ************ Evaluation metrics on test data ***********

Testing Accuracy: 0.86460
Testing F1 Score: 0.86583
Testing Precision Score: 0.86387
Testing Recall Score: 0.86781
```

[ ]: