CSCI 544 - Applied Natural Language Processing
Homework 1
Report

By: Nikita Maid

Including three sample reviews in your report along with corresponding ratings.

In [5]:
```
pd.set_option('display.max_colwidth', None)
df.loc[3:5]
```

Out[5]:

	Review	Ratings
3	A must if you love garlic on tomato marinara sauce.	5
4	Worth every penny! Buy one now and be a pizza slice master!!	5
5	The description says "Suitable for all type of surfaces" but it is not true! We ordered this item for our large family (we like to cook a lot) but this does not work as advertised! This pressure cooker does NOT work for induction stoves. I called the manufacturer (Magefesa) and they confirmed that it will not work. We are exchanging this pot for the Stainless Steel version that is supposed to work on induction stoves. Magefesa said that the 14.3 quart pot will work on induction stoves and the description here on Amazon clearly states that it will work on induction stoves. Thankfully Amazon has a great return process so they will send a truck to pick this up and they will send the other version in a couple days.	1

Here are the three samples before the data cleaning and data preprocessing with their corresponding ratings. I have set max_colwidth so that all the content of Review is seen and not being truncated from the output.
For printing, I have just selected rows from 3 to 5.

Also, report the statistics of the ratings, i.e., how many reviews received 1 ratings, etc.

In [6]:
```
# Calculating the statistics of the rating
df.groupby(['Ratings']).count()
```

Out[6]:

Ratings	Review
1	427276
2	242187
3	349921
4	732439
5	3128400

To compute the statistics of the ratings, I have used groupby function along with count.
As we can clearly see from the output, how many reviews have received the particular rating
The groupby function helps us to group all the ratings and then the count function counts the number of reviews for that particular group. In this case, it will count the number of reviews with the rating of 1,2,3,4,5.

Include the number of reviews for each of these three classes in your report

In [9]:
```
stats = df.groupby(['Label']).Review.count()
print(stats)
```

```
Label
0.0         669463
1.0        3860839
neutral     349921
Name: Review, dtype: int64
```

The three classes here are:

1) Label 0 with Ratings lesser than 3
2) Label 1 with Ratings greater than 3
3) Label neutral with Ratings equal to 3.

As per the homework document, they are printed with a comma in the python executable file.

In your report, print the average length of the reviews in terms of character length in your dataset before and after cleaning

Before cleaning

In [11]:
```
# Printing the average character length of the Review before Data Cleaning
length_preCLeaning = df.Review.str.len().mean()
print(length_preCLeaning)
```
```
321.85835150109006
```

After cleaning

In [18]:
```
# Printing the average character length of the Review after Data Cleaning
length_afterCLeaning = df.Review.str.len().mean()
print(length_afterCLeaning)
```
```
310.38993
```

We can see the average length of the reviews have reduced by 11.4684

Print three sample reviews before and after data cleaning + preprocessing.

For this, I have included the same rows as for the first question to show the comparison.
Before data cleaning + preprocessing

In [12]:
```
df.loc[3:5]
```
Out[12]:

	Review	Ratings	Label
3	the pan still sticks so I think I will dump it	1	0
4	was not cute at all>! the colors were all faded after the first wash. as a gift is fine and cheap (i guess) but for own children, wont recommend it.	1	0
5	Wow, if you're thinking of buying this can opener, consider this: the edges of the lid after opening a can are jagged! It's very easy to cut yourself while trying to remove the lid from the strong magnet that \\"catches\\" it. The can opener looks nice and does a reasonable job of cutting, but it's probably not worth the danger of using it. Especially if children in your home are apt to use it.	2	0

After data cleaning + preprocessing

In [21]:
```
df.loc[3:5]
```
Out[21]:

	Review	Ratings	Label
3	pan still stick think dump	1	0
4	cute color faded first wash gift fine cheap guess child recommend	1	0
5	wow thinking buying opener consider edge lid opening jagged easy cut trying remove lid strong magnet catch opener look nice reasonable job cutting probably worth danger using especially child home apt use	2	0

As we can see from the two comparisons, in the after data cleaning + preprocessing picture, all the text is firstly in lower case, there are no alphanumeric, no html or url , no extra spaces, all the contractions are expanded, and stop words are removed and the sentences are lemmatized. Therefore we can see the fewer number of characters in the before and after.

Note: Please find all the required code in the PDF version of the ipynb file attached after this.

In [1]:

```
import pandas as pd
import numpy as np
import nltk
nltk.download('wordnet')
import re
from bs4 import BeautifulSoup
```

```
[nltk_data] Downloading package wordnet to
[nltk_data]     C:\Users\nikit\AppData\Roaming\nltk_data...
[nltk_data]   Package wordnet is already up-to-date!
```

In [2]:

```
#! 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_0
0.tsv.gz
```

Read Data

In [3]:

```
import csv
df = pd.read_csv('C://Users//nikit//OneDrive//Desktop//Masters//Fall2021//ANLP//Homework/
/HW1//amazon_reviews_us_Kitchen_v1_00.tsv'
                 , parse_dates=True, delimiter="\t", quoting=csv.QUOTE_NONE, encoding='u
tf-8')
df.head()
```

Out[3]:

	marketplace	customer_id	review_id	product_id	product_parent	product_title	product_category	star_rating	I
0	US	37000337	R3DT59XH7HXR9K	B00303FI0G	529320574	Arthur Court Paper Towel Holder	Kitchen	5	
1	US	15272914	R1LFS11BNASSU8	B00JCZKZN6	274237558	Olde Thompson Bavaria Glass Salt and Pepper Mi...	Kitchen	5	
2	US	36137863	R296RT05AG0AF6	B00JLIKA5C	544675303	Progressive International PL8 Professional Man...	Kitchen	5	
3	US	43311049	R3V37XDZ7ZCI3L	B000GBNB8G	491599489	Zyliss Jumbo Garlic Press	Kitchen	5	
4	US	13763148	R14GU232NQFYX2	B00VJ5KX9S	353790155	1 X Premier Pizza Cutter - Stainless Steel 14"...	Kitchen	5	

Keep Reviews and Ratings

In [4]:

```python
df = df.drop(columns=['marketplace', 'customer_id', 'review_id', 'product_id',
       'product_parent', 'product_title', 'product_category',
       'helpful_votes', 'total_votes', 'vine', 'verified_purchase',
       'review_headline','review_date'])
df.rename(columns={'star_rating': 'Ratings', 'review_body': 'Review'}, inplace=True)
columns_change = ['Review','Ratings']
df = df[columns_change]
```

In [5]:

```python
pd.set_option('display.max_colwidth', None)
df.loc[3:5]
```

Out[5]:

	Review	Ratings
3	A must if you love garlic on tomato marinara sauce.	5
4	Worth every penny! Buy one now and be a pizza slice master!!	5
5	The description says "Suitable for all type of surfaces" but it is not true! We ordered this item for our large family (we like to cook a lot) but this does not work as advertised! This pressure cooker does NOT work for induction stoves. I called the manufacturer (Magefesa) and they confirmed that it will not work. We are exchanging this pot for the Stainless Steel version that is supposed to work on induction stoves. Magefesa said that the 14.3 quart pot will work on induction stoves and the description here on Amazon clearly states that it will work on induction stoves. Thankfully Amazon has a great return process so they will send a truck to pick this up and they will send the other version in a couple days.	1

In [6]:

```python
# Calculating the statistics of the rating
df.groupby(['Ratings']).count()
```

Out[6]:

	Review
Ratings	
1	427276
2	242187
3	349921
4	732439
5	3128400

Labelling Reviews:

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

In [7]:

```python
df.loc[df['Ratings']  < 3 , 'Label'] = 0
df.loc[df['Ratings'] > 3 , 'Label'] = 1
df.loc[df['Ratings'] == 3 , 'Label'] = "neutral"
pd.options.display.float_format = '{:,.0f}'.format
```

In [8]:

```python
# Including the number of reviews for each of the three classes
stats = df.groupby(['Label']).Review.count()
print(" Statistics of three classes : Label 0 : {0}, Label 1 : {1}, Label neutral : {2}"
.format(stats[0], stats[1],stats[2]))
```

```
 Statistics of three classes : Label 0 : 669463, Label 1 : 3860839, Label neutral : 34992
```

1

In [9]:

```
stats = df.groupby(['Label']).Review.count()
print(stats)
```

```
Label
0.0          669463
1.0         3860839
neutral      349921
Name: Review, dtype: int64
```

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

In [10]:

```
positive = df.query("Label == 1").sample(100000)
negative = df.query("Label == 0").sample(100000)

frames = [positive , negative]

result = pd.concat(frames)

df = result.sample(frac = 1)

df.reset_index(inplace=True)

df = df.drop(columns = 'index')
```

In [11]:

```
# Printing the average character length of the Review before Data Cleaning
length_preCLeaning = df.Review.str.len().mean()
print(length_preCLeaning)
```

```
321.85835150109006
```

In [12]:

```
df.loc[3:5]
```

Out[12]:

	Review	Ratings	Label
3	the pan still sticks so I think I will dump it	1	0
4	was not cute at all>! the colors were all faded after the first wash. as a gift is fine and cheap (i guess) but for own children, wont recommend it.	1	0
5	Wow, if you're thinking of buying this can opener, consider this: the edges of the lid after opening a can are jagged! It's very easy to cut yourself while trying to remove the lid from the strong magnet that \\"catches\\" it. The can opener looks nice and does a reasonable job of cutting, but it's probably not worth the danger of using it. Especially if children in your home are apt to use it.	2	0

Data Cleaning

Convert the all reviews into the lower case.

In [13]:

```
df['Review'] = df['Review'].str.lower()
```

remove the HTML and URLs from the reviews

remove the HTML and URLs from the reviews

In [14]:

```
def clean_html(html):
    try:
        soup = BeautifulSoup(html, "html.parser").text
        return soup
    except:
        text = ''
        return text
import bs4

df['Review'] = df['Review'].apply(lambda x: clean_html(x))
df['Review'] = df['Review'].str.replace('<.*?>',' ', regex=True)
```

remove non-alphabetical characters

In [15]:

```
df['Review'] = df['Review'].str.replace("[^a-zA-Z0-9']"," ", regex=True)
```

Remove the extra spaces between the words

In [16]:

```
df['Review'] = df['Review'].str.replace('\s+',' ',regex=True)
```

perform contractions on the reviews.

In [17]:

```
import contractions
def contractionfunction(s):
    expanded_words = []
    expanded_words.append(contractions.fix(s))
    expanded_text = ' '.join(expanded_words)
    s = expanded_text
    return s
df['Review'] = df['Review'].apply(str)
df['Review'] = df['Review'].apply(lambda x: contractionfunction(x))
```

In [18]:

```
# Printing the average character length of the Review after Data Cleaning
length_afterCLeaning = df.Review.str.len().mean()
print(length_afterCLeaning)
```

```
310.38993
```

Pre-processing

remove the stop words

In [19]:

```
from nltk.corpus import stopwords
nltk.download('stopwords')
stop = stopwords.words('english')
df['Review'] = df['Review'].apply(lambda x: ' '.join([word for word in x.split() if word
not in (stop)]))
```

```
[nltk_data] Downloading package stopwords to
[nltk_data]     C:\Users\nikit\AppData\Roaming\nltk_data...
```

```
[nltk_data]   Package stopwords is already up-to-date!
```

perform lemmatization

In [20]:

```
from nltk.stem import WordNetLemmatizer
lemmatizer = nltk.stem.WordNetLemmatizer()
def pract_lem(sentence):
    sente = nltk.word_tokenize(sentence)
    lemmatized_string = ' '.join([lemmatizer.lemmatize(words) for words in sente])
    return lemmatized_string
import nltk
nltk.download('punkt')
df['Review']=df['Review'].apply(str)
df['Review'] = df['Review'].apply(lambda x: pract_lem(x))
```

```
[nltk_data] Downloading package punkt to
[nltk_data]     C:\Users\nikit\AppData\Roaming\nltk_data...
[nltk_data]   Package punkt is already up-to-date!
```

In [21]:

```
df.loc[3:5]
```

Out[21]:

	Review	Ratings	Label
3	**pan still stick think dump**	**1**	**0**
4	**cute color faded first wash gift fine cheap guess child recommend**	**1**	**0**
5	**wow thinking buying opener consider edge lid opening jagged easy cut trying remove lid strong magnet catch opener look nice reasonable job cutting probably worth danger using especially child home apt use**	**2**	**0**

In [22]:

```
# Printing the average character length of the Review after Data Cleaning and pre-process
ing
length_after = df.Review.str.len().mean()
print(length_after)
```

```
191.956595
```

TF-IDF Feature Extraction

In [23]:

```
from sklearn.feature_extraction.text import TfidfVectorizer
vector_data = TfidfVectorizer().fit(df['Review'])
```

In [24]:

```
vectorized_data = vector_data.transform(df['Review'])
```

In [25]:

```
vectorized_data
```

Out[25]:

```
<200000x66243 sparse matrix of type '<class 'numpy.float64'>'
 with 4968162 stored elements in Compressed Sparse Row format>
```

In [26]:

```
# Splitting the dataset into training and testing data
```

```
from sklearn.model_selection import train_test_split
X_train, X_test, y_train, y_test = train_test_split(vectorized_data, df['Label'], random
_state=50,test_size=0.2)
```

In [27]:

```
y_train=y_train.astype('int')
y_test=y_test.astype('int')
```

Perceptron

In [28]:

```
from sklearn.linear_model import Perceptron
Perceptron_model = Perceptron(max_iter=40, eta0=0.1, random_state=0)
Perceptron_model.fit(X_train, y_train)
```

Out[28]:

```
Perceptron(eta0=0.1, max_iter=40)
```

In [29]:

```
PPredictions_traindata = Perceptron_model.predict(X_train)
PPredictions_testdata = Perceptron_model.predict(X_test)
```

In [30]:

```
from sklearn import metrics
```

In [31]:

```
# Accuracy, Precision, Recall, F1 Score for training data
P_accuracy = metrics.accuracy_score(y_train,PPredictions_traindata)
P_precision = metrics.precision_score(y_train,PPredictions_traindata)
P_recall = metrics.recall_score(y_train,PPredictions_traindata)
P_f1Score = metrics.f1_score(y_train,PPredictions_traindata)
print(" Accuracy: {0} \n Precision : {1} \n Recall : {2} \n F1_Score : {3}".format(P_acc
uracy, P_precision,P_recall,P_f1Score))
```

```
 Accuracy: 0.89889375
 Precision : 0.9069056834385822
 Recall : 0.8893216974418517
 F1_Score : 0.8980276221153423
```

In [32]:

```
# Accuracy, Precision, Recall, F1 Score for testing data
P_accuracy1 = metrics.accuracy_score(y_test,PPredictions_testdata)
P_precision1 = metrics.precision_score(y_test,PPredictions_testdata)
P_recall1 = metrics.recall_score(y_test,PPredictions_testdata)
P_f1Score1 = metrics.f1_score(y_test,PPredictions_testdata)
print(" Accuracy: {0} \n Precision : {1} \n Recall : {2} \n F1_Score : {3}".format(P_acc
uracy1, P_precision1,P_recall1,P_f1Score1))
```

```
 Accuracy: 0.85655
 Precision : 0.865359814289399
 Recall : 0.8428377631512838
 F1_Score : 0.8539503156180004
```

SVM

In [33]:

```
from sklearn import svm
from sklearn.pipeline import Pipeline
from sklearn.preprocessing import StandardScaler
```

```
from sklearn.svm import LinearSVC
```

In [34]:

```
svm_model = Pipeline([('scaler' ,StandardScaler(with_mean=False)), ('Linear_svc', LinearS
VC(C=1, dual=False))])
```

In [35]:

```
svm_model.fit(X_train, y_train)
```

Out[35]:

```
Pipeline(steps=[('scaler', StandardScaler(with_mean=False)),
                ('Linear_svc', LinearSVC(C=1, dual=False))])
```

In [36]:

```
SVMPredictions_traindata = svm_model.predict(X_train)
SVMPredictions_testdata = svm_model.predict(X_test)
```

In [37]:

```
# Accuracy, Precision, Recall, F1 Score for training data
SVM_accuracy = metrics.accuracy_score(y_train,SVMPredictions_traindata)
SVM_precision = metrics.precision_score(y_train,SVMPredictions_traindata)
SVM_recall = metrics.recall_score(y_train,SVMPredictions_traindata)
SVM_f1Score = metrics.f1_score(y_train,SVMPredictions_traindata)
print(" Accuracy: {0} \n Precision : {1} \n Recall : {2} \n F1_Score : {3}".format(SVM_a
ccuracy, SVM_precision,SVM_recall,SVM_f1Score))
```

```
 Accuracy: 0.95065625
 Precision : 0.9502045193794583
 Recall : 0.9512840680674682
 F1_Score : 0.9507439872726705
```

In [38]:

```
# Accuracy, Precision, Recall, F1 Score for testing data
SVM_accuracy1 = metrics.accuracy_score(y_test,PPredictions_testdata)
SVM_precision1 = metrics.precision_score(y_test,PPredictions_testdata)
SVM_recall1 = metrics.recall_score(y_test,PPredictions_testdata)
SVM_f1Score1 = metrics.f1_score(y_test,PPredictions_testdata)
print(" Accuracy: {0} \n Precision : {1} \n Recall : {2} \n F1_Score : {3}".format(SVM_a
ccuracy1, SVM_precision1,SVM_recall1,SVM_f1Score1))
```

```
 Accuracy: 0.85655
 Precision : 0.865359814289399
 Recall : 0.8428377631512838
 F1_Score : 0.8539503156180004
```

Logistic Regression

In [39]:

```
from sklearn.linear_model import LogisticRegression
```

In [40]:

```
LR_model = LogisticRegression(random_state=0 , max_iter = 500).fit(X_train, y_train)
```

In [41]:

```
LRPredictions_traindata = LR_model.predict(X_train)
LRPredictions_testdata = LR_model.predict(X_test)
```

In [42]:

```
# Accuracy, Precision, Recall, F1 Score for training data
```

```
LR_accuracy = metrics.accuracy_score(y_train,LRPredictions_traindata)
LR_precision = metrics.precision_score(y_train,LRPredictions_traindata)
LR_recall = metrics.recall_score(y_train,LRPredictions_traindata)
LR_f1Score = metrics.f1_score(y_train,LRPredictions_traindata)
print(" Accuracy: {0} \n Precision : {1} \n Recall : {2} \n F1_Score : {3}".format(LR_ac
curacy, LR_precision,LR_recall,LR_f1Score))
```

```
 Accuracy: 0.91415625
 Precision : 0.9175286271549012
 Recall : 0.9103462052261633
 F1_Score : 0.9139233049440048
```

In [43]:

```
# Accuracy, Precision, Recall, F1 Score for testing data
LR_accuracy1 = metrics.accuracy_score(y_test,LRPredictions_testdata)
LR_precision1 = metrics.precision_score(y_test,LRPredictions_testdata)
LR_recall1 = metrics.recall_score(y_test,LRPredictions_testdata)
LR_f1Score1 = metrics.f1_score(y_test,LRPredictions_testdata)
print(" Accuracy: {0} \n Precision : {1} \n Recall : {2} \n F1_Score : {3}".format(LR_ac
curacy1, LR_precision1,LR_recall1,LR_f1Score1))
```

```
 Accuracy: 0.899225
 Precision : 0.9028017460156329
 Recall : 0.8936843691905743
 F1_Score : 0.8982199217270547
```

Naive Bayes

In [44]:

```
from sklearn.naive_bayes import MultinomialNB
MNB_model = MultinomialNB().fit(X_train, y_train)
```

In [45]:

```
MNBPredictions_traindata = MNB_model.predict(X_train)
MNBPredictions_testdata = MNB_model.predict(X_test)
```

In [46]:

```
# Accuracy, Precision, Recall, F1 Score for training data
MNB_accuracy = metrics.accuracy_score(y_train,MNBPredictions_traindata)
MNB_precision = metrics.precision_score(y_train,MNBPredictions_traindata)
MNB_recall = metrics.recall_score(y_train,MNBPredictions_traindata)
MNB_f1Score = metrics.f1_score(y_train,MNBPredictions_traindata)
print(" Accuracy: {0} \n Precision : {1} \n Recall : {2} \n F1_Score : {3}".format(MNB_a
ccuracy, MNB_precision,MNB_recall,MNB_f1Score))
```

```
 Accuracy: 0.88769375
 Precision : 0.8916090968685392
 Recall : 0.8830043572168745
 F1_Score : 0.8872858657265982
```

In [47]:

```
# Accuracy, Precision, Recall, F1 Score for testing data
MNB_accuracy1 = metrics.accuracy_score(y_test,MNBPredictions_testdata)
MNB_precision1 = metrics.precision_score(y_test,MNBPredictions_testdata)
MNB_recall1 = metrics.recall_score(y_test,MNBPredictions_testdata)
MNB_f1Score1 = metrics.f1_score(y_test,MNBPredictions_testdata)
print(" Accuracy: {0} \n Precision : {1} \n Recall : {2} \n F1_Score : {3}".format(MNB_a
ccuracy1, MNB_precision1,MNB_recall1,MNB_f1Score1))
```

```
 Accuracy: 0.868725
 Precision : 0.8733564366527368
 Recall : 0.86102597598352
 F1_Score : 0.8671473750790638
```