HW1-CSCI544

September 7, 2021

[166]:
```
#importing libraries
import pandas as pd
import numpy as np
import nltk
#nltk.download('wordnet')
import re
from bs4 import BeautifulSoup
import contractions
from nltk.corpus import stopwords
from nltk.stem import WordNetLemmatizer
from nltk.tokenize import word_tokenize
from sklearn.feature_extraction.text import TfidfVectorizer
from sklearn.linear_model import Perceptron
from sklearn.metrics import accuracy_score, precision_recall_fscore_support as␣
↪score
from sklearn.svm import LinearSVC as SVC
from sklearn.linear_model import LogisticRegression
from sklearn.naive_bayes import MultinomialNB
import warnings
warnings.filterwarnings('ignore')
```

[167]:
```
#! 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

[168]:
```
file_path = '/Users/vanshdhar/Desktop/amazon_reviews_us_Kitchen_v1_00.tsv'
input_data = pd.read_csv(file_path,␣
↪sep='\t',error_bad_lines=False,warn_bad_lines=False)
input_data =input_data.dropna()
```

0.2 Keep Reviews and Ratings

[169]:
```
input_data = input_data[['review_body','star_rating']]

#printing sample reviews and rating statistics
print('Sample Reviews:')
print(input_data.head(3))
print('\nReviews with 1.0 Rating: {} '.format((input_data.loc[␣
 ↪input_data['star_rating'] == 1.0 ]).shape[0]))
print('Reviews with 2.0 Rating: {} '.format((input_data.loc[␣
 ↪input_data['star_rating'] == 2.0 ]).shape[0]))
print('Reviews with 3.0 Rating: {} '.format((input_data.loc[␣
 ↪input_data['star_rating'] == 3.0 ]).shape[0]))
print('Reviews with 4.0 Rating: {} '.format((input_data.loc[␣
 ↪input_data['star_rating'] == 4.0 ]).shape[0]))
print('Reviews with 5.0 Rating: {} '.format((input_data.loc[␣
 ↪input_data['star_rating'] == 5.0 ]).shape[0]))
```

```
Sample Reviews:
                                         review_body  star_rating
0                 Beautiful.  Looks great on counter.          5.0
1   I personally have 5 days sets and have also bo…         5.0
2   Fabulous and worth every penny. Used for clean…         5.0

Reviews with 1.0 Rating: 426852
Reviews with 2.0 Rating: 241931
Reviews with 3.0 Rating: 349533
Reviews with 4.0 Rating: 731693
Reviews with 5.0 Rating: 3124553
```

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'

[170]:
```
#printing statistics of three classes
rating_less_three = (input_data.loc[ input_data['star_rating'] < 3.0 ]).shape[0]
rating_eq_three = (input_data.loc[ input_data['star_rating'] == 3.0 ]).shape[0]
rating_more_three = (input_data.loc[ input_data['star_rating'] > 3.0 ]).shape[0]

print('Statistics of three classes- Reviews with Positive sentiment : {},␣
 ↪Reviews with Negative sentiment : {}, Reviews with Neutral Sentiment : {},'.
 ↪format(rating_more_three,rating_less_three,rating_eq_three))

#labelling reviews
input_data = input_data.loc[ input_data['star_rating'] != 3.0 ]
input_data['binary_label'] = input_data['star_rating']
```

```
input_data.loc[input_data['star_rating'] > 3.0, 'binary_label'] = 1
input_data.loc[input_data['star_rating'] < 3.0, 'binary_label'] = 0
```

```
Statistics of three classes- Reviews with Positive sentiment : 3856246, Reviews
with Negative sentiment : 668783, Reviews with Neutral Sentiment : 349533,
```

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

[171]:
```
#random sampling positive and negative reviews
positive_data = input_data.loc[ input_data['binary_label'] == 1 ]
positive_data = positive_data.sample(n=100000, random_state=65)
negative_data = input_data.loc[ input_data['binary_label'] == 0 ]
negative_data = negative_data.sample(n=100000, random_state=65)

input_data = pd.concat([positive_data,negative_data])
input_data = input_data.sample(frac = 1, random_state=65).reset_index(drop=True)

#splitting the dataset into training and testing
training_dataset = input_data.sample(frac = 0.8, random_state=65)
testing_dataset = input_data.drop(training_dataset.index)
training_dataset = training_dataset.reset_index(drop=True)
testing_dataset = testing_dataset.reset_index(drop=True)
```

2 Data Cleaning

2.1 Convert the all reviews into the lower case.

[172]:
```
#printing sample reviews before data cleaning and pre-processing
sample_reviews_before_data_cleaning = training_dataset['review_body'].head(3)
#print(training_dataset['review_body'].head(3))
#print(testing_dataset['review_body'].head(3))
print('\nSample reviews before Data cleaning:\n')
for idx,rev in enumerate(list(sample_reviews_before_data_cleaning)):
    print('{}. {}'.format(idx+1,rev))

#printing average length of reviews before data cleaning
training_data_Length_bef_clean = training_dataset['review_body'].str.len()
testing_data_Length_bef_clean = testing_dataset['review_body'].str.len()

avg_len_before_data = (sum(training_data_Length_bef_clean) +␣
 ↪sum(testing_data_Length_bef_clean)) /␣
 ↪(len(training_dataset)+len(testing_dataset))

#Converting reviews in lower case
training_dataset['review_body'] = training_dataset['review_body'].str.lower()
testing_dataset['review_body'] = testing_dataset['review_body'].str.lower()
```

```
Sample reviews before Data cleaning:

1. So happy I made the purchase. The carbonating of water is much much easier
with this one. I have an older version that you screw your bottles onto and this
just saves so much time. Plus the indicator lights make it just to easy. Both
features promote use for me which is a plus.
2. I recommend this mug as a great alternative to getting cups every time you by
a coffee/tea. It helps with recycling and environment.
3. This strainer looks cool, but the metal rim around the top of the darned
thing catches flecks of food when you go to dump it out. So it's kind of a bear
to clean. Tell you the truth, I went back to my old enamel one from the 1930's.
Nice try, WalterDrake.
```

2.2 remove the HTML and URLs from the reviews

[173]:
```
#removing HTML tags
training_dataset['review_body'] = training_dataset['review_body'].apply(lambda␣
 ↪x: BeautifulSoup(x, 'html.parser').get_text())
testing_dataset['review_body'] = testing_dataset['review_body'].apply(lambda x:␣
 ↪BeautifulSoup(x, 'html.parser').get_text())

#removing URL tags
training_dataset['review_body'] = training_dataset['review_body'].apply(lambda␣
 ↪x: re.sub(r'^https?:\/\/.*[\r\n]*', '', x))
testing_dataset['review_body'] = testing_dataset['review_body'].apply(lambda x:␣
 ↪re.sub(r'^https?:\/\/.*[\r\n]*', '', x))
#training_dataset
```

2.3 remove non-alphabetical characters

[174]:
```
training_dataset['review_body'] = training_dataset['review_body'].str.
 ↪replace('[^a-zA-Z ]', ' ')
testing_dataset['review_body'] = testing_dataset['review_body'].str.
 ↪replace('[^a-zA-Z ]', ' ')
```

2.4 Remove the extra spaces between the words

[175]:
```
training_dataset['review_body'] = training_dataset['review_body'].apply(lambda␣
 ↪x: re.sub(' +', ' ', x))
testing_dataset['review_body'] = testing_dataset['review_body'].apply(lambda x:␣
 ↪re.sub(' +', ' ', x))
```

2.5 perform contractions on the reviews.

[176]:
```
#!pip install contractions

#performing contractions
def Contractionfunction(text):
    expanded_words = []
    for word in text.split():
        expanded_words.append(contractions.fix(word))

    expanded_text = ' '.join(expanded_words)
    return expanded_text

training_dataset['review_body'] = training_dataset['review_body'].apply(lambda␣
↪x: Contractionfunction(x))
testing_dataset['review_body'] = testing_dataset['review_body'].apply(lambda x:␣
↪Contractionfunction(x))

#printing average length of reviews after data cleaning
training_data_Length_aft_clean = training_dataset['review_body'].str.len()
testing_data_Length_aft_clean = testing_dataset['review_body'].str.len()

avg_len_after_data = (sum(training_data_Length_aft_clean) +␣
↪sum(testing_data_Length_aft_clean)) /␣
↪(len(training_dataset)+len(testing_dataset))

print('Average length of reviews in Dataset before Data Cleaning  : {}, Average␣
↪length of reviews in Dataset after Data Cleaning  : {}'.
↪format(avg_len_before_data,avg_len_after_data))
```

```
Average length of reviews in Dataset before Data Cleaning  : 321.8445, Average
length of reviews in Dataset after Data Cleaning  : 306.76644
```

3 Pre-processing

3.1 remove the stop words

[177]:
```
#calculating average length of reviews before pre-processing
training_data_Length_bef_process = training_dataset['review_body'].str.len()
testing_data_Length_bef_process = testing_dataset['review_body'].str.len()

avg_len_bef_process_data = (sum(training_data_Length_bef_process) +␣
↪sum(testing_data_Length_bef_process)) /␣
↪(len(training_dataset)+len(testing_dataset))
#avg_len_testing_data = testing_data_Length_bef_process.sum() /␣
↪len(testing_dataset)
```

```
#nltk.download('stopwords')
#removing stop words
StopWords = stopwords.words('english')

training_dataset['review_body'] = training_dataset['review_body'].apply(lambda␣
 ↪x: ' '.join([word for word in x.split() if word not in (StopWords)]))
testing_dataset['review_body'] = testing_dataset['review_body'].apply(lambda x:␣
 ↪' '.join([word for word in x.split() if word not in (StopWords)]))
```

3.2 perform lemmatization

[178]:
```
#nltk.download('punkt')
#performing lemmatization
lemmatizer = WordNetLemmatizer()

def lemmatize_text(text):
    lemmatize_tokens = [lemmatizer.lemmatize(w) for w in word_tokenize(text)]
    return ' '.join(lemmatize_tokens)

training_dataset['review_body'] = training_dataset['review_body'].apply(lambda␣
 ↪x: lemmatize_text(x))
testing_dataset['review_body'] = testing_dataset['review_body'].apply(lambda x:␣
 ↪lemmatize_text(x))

#training_dataset['review_body']

#calculating average length of reviews after pre-processing
training_data_Length_aft_process = training_dataset['review_body'].str.len()
testing_data_Length_aft_process = testing_dataset['review_body'].str.len()

avg_len_aft_process_data = ( sum(training_data_Length_aft_process) +␣
 ↪sum(testing_data_Length_aft_process)) / (len(training_dataset) +␣
 ↪len(testing_dataset))

#printing average length of reviews before and after pre-processing
print('Average length of reviews in  Dataset before Pre-Processing  : {},␣
 ↪Average length of reviews in Dataset after Pre-Processing  : {}'.
 ↪format(avg_len_bef_process_data,avg_len_aft_process_data))
#print('Average length of reviews in testing dataset after Pre-Processing  :␣
 ↪{}'.format(avg_len_aft_process_data))

#printing sample reviews after data cleaning and pre-processing
sample_reviews_after_pre_processing = training_dataset['review_body'].head(3)

#print(sample_reviews_before_data_cleaning)
```

```
print('\nSample reviews after Data Pre-Processing:\n')
for idx,rev in enumerate(list(sample_reviews_after_pre_processing)):
    print('{}. {}'.format(idx+1,rev))
#print(sample_reviews_after_pre_processing)
```

```
Average length of reviews in  Dataset before Pre-Processing  : 306.76644,
Average length of reviews in Dataset after Pre-Processing  : 188.24344

Sample reviews after Data Pre-Processing:

1. happy made purchase carbonating water much much easier one older version
screw bottle onto save much time plus indicator light make easy feature promote
use plus
2. recommend mug great alternative getting cup every time coffee tea help
recycling environment
3. strainer look cool metal rim around top darned thing catch fleck food go dump
kind bear clean tell truth went back old enamel one nice try walterdrake
```

4 TF-IDF Feature Extraction

[179]:
```
vectoriser = TfidfVectorizer(min_df=0.001)
training_dataset['review_body']  = list(vectoriser.
 ↪fit_transform(training_dataset['review_body']).toarray())
testing_dataset['review_body']  = list(vectoriser.
 ↪transform(testing_dataset['review_body']).toarray())
```

5 Perceptron

[180]:
```
X_train = training_dataset['review_body'].tolist()
X_test = testing_dataset['review_body'].tolist()
y_train = training_dataset['binary_label'].tolist()
y_test = testing_dataset['binary_label'].tolist()

perceptModel = Perceptron(tol=1e-3, max_iter = 75, random_state=0)
perceptModel.fit(X_train, y_train)
#accuracy = clf.score(X_test, y_test)
y_train_pred = perceptModel.predict(X_train)
y_test_pred = perceptModel.predict(X_test)

precision_train,recall_train,fscore_train, _ =␣
 ↪score(y_train,y_train_pred,average='binary')
accuracy_train = accuracy_score(y_train,y_train_pred)

precision_test,recall_test,fscore_test, _ =␣
 ↪score(y_test,y_test_pred,average='binary')
accuracy_test = accuracy_score(y_test,y_test_pred)
```

```
print ('Perceptron Training Data Accuracy : {}, Precision : {}, Recall : {},␣
↪F-score : {},  Perceptron Testing Data Accuracy : {}, Precision : {}, Recall␣
↪: {}, F-score : {}'.
↪format(accuracy_train,precision_train,recall_train,fscore_train,accuracy_test,precision_tes
```

```
Perceptron Training Data Accuracy : 0.875175, Precision : 0.8565322695540676,
Recall : 0.9014394242303079, F-score : 0.8784122732253744,  Perceptron Testing
Data Accuracy : 0.865575, Precision : 0.8439488944415633, Recall : 0.896484375,
F-score : 0.8694237353991112
```

6 SVM

[184]:
```
svclassifier = SVC(max_iter = 5000,random_state=0)
#svclassifier = SVC(kernel='linear',tol=1e-3, max_iter = 500,␣
↪random_state=0)#500
svclassifier.fit(X_train, y_train)
#y_pred = svclassifier.predict(X_test)
y_train_pred = svclassifier.predict(X_train)
y_test_pred = svclassifier.predict(X_test)

precision_train,recall_train,fscore_train, _ =␣
↪score(y_train,y_train_pred,average='binary')
accuracy_train = accuracy_score(y_train,y_train_pred)

precision_test,recall_test,fscore_test, _ =␣
↪score(y_test,y_test_pred,average='binary')
accuracy_test = accuracy_score(y_test,y_test_pred)

print ('SVM Training Data Accuracy : {}, Precision : {}, Recall : {}, F-score :␣
↪{},  SVM Testing Data Accuracy : {}, Precision : {}, Recall : {}, F-score :␣
↪{}'.
↪format(accuracy_train,precision_train,recall_train,fscore_train,accuracy_test,precision_tes
```

```
SVM Training Data Accuracy : 0.90211875, Precision : 0.905415107886483, Recall :
0.898140743702519, F-score : 0.9017632557818606,  SVM Testing Data Accuracy :
0.89305, Precision : 0.8914670658682635, Recall : 0.8946814903846154, F-score :
0.8930713857228555
```

7 Logistic Regression

[182]:
```
logisticRegr = LogisticRegression(tol=2e-3, max_iter = 50, random_state=0)
logisticRegr.fit(X_train, y_train)

y_train_pred = logisticRegr.predict(X_train)
y_test_pred = logisticRegr.predict(X_test)
```

```
precision_train,recall_train,fscore_train, _ =␣
 ↪score(y_train,y_train_pred,average='binary')
accuracy_train = accuracy_score(y_train,y_train_pred)

precision_test,recall_test,fscore_test, _ =␣
 ↪score(y_test,y_test_pred,average='binary')
accuracy_test = accuracy_score(y_test,y_test_pred)

print ('Logistic Regression Training Data Accuracy : {}, Precision : {}, Recall␣
 ↪: {}, F-score : {},  Logistic Regression Testing Data Accuracy : {},␣
 ↪Precision : {}, Recall : {}, F-score : {}'.
 ↪format(accuracy_train,precision_train,recall_train,fscore_train,accuracy_test,precision_tes
```

```
Logistic Regression Training Data Accuracy : 0.9011, Precision :
0.9048832164657217, Recall : 0.8965163934426229, F-score : 0.9006803745826115,
Logistic Regression Testing Data Accuracy : 0.893175, Precision :
0.8922769307673082, Recall : 0.8939302884615384, F-score : 0.8931028444199834
```

8 Naive Bayes

[183]:
```
NBayesModel = MultinomialNB()
NBayesModel.fit(X_train, y_train)

y_train_pred = NBayesModel.predict(X_train)
y_test_pred = NBayesModel.predict(X_test)

precision_train,recall_train,fscore_train, _ =␣
 ↪score(y_train,y_train_pred,average='binary')
accuracy_train = accuracy_score(y_train,y_train_pred)

precision_test,recall_test,fscore_test, _ =␣
 ↪score(y_test,y_test_pred,average='binary')
accuracy_test = accuracy_score(y_test,y_test_pred)

print ('Naive Bayes Training Data Accuracy : {}, Precision : {}, Recall : {},␣
 ↪F-score : {},  Naive Bayes Testing Data Accuracy : {}, Precision : {},␣
 ↪Recall : {}, F-score : {}'.
 ↪format(accuracy_train,precision_train,recall_train,fscore_train,accuracy_test,precision_tes
```

```
Naive Bayes Training Data Accuracy : 0.86963125, Precision : 0.8662828845558651,
Recall : 0.8743252698920432, F-score : 0.8702854975218741,  Naive Bayes Testing
Data Accuracy : 0.866125, Precision : 0.860654523915297, Recall :
0.8731971153846154, F-score : 0.866880453426803
```

[ ]:
```
```