Amazon Fine Food Reviews Analysis-Copy1

March 21, 2021

1 Amazon Fine Food Reviews

Data Source: https://www.kaggle.com/snap/amazon-fine-food-reviews

Attribute Information:

Number of attributes-10 1. Id 2. ProductId - unique identifier for the product 3. UserId - unqiue identifier for the user 4. ProfileName 5. HelpfulnessNumerator - number of users who found the review helpful 6. HelpfulnessDenominator - number of users who indicated whether they found the review helpful or not 7. Score - rating between 1 and 5 8. Time - timestamp for the review 9. Summary - brief summary of the review 10. Text - text of the review

Objective: Given a review, determine whether the review is positive (Rating of 4 or 5) or negative (rating of 1 or 2). (We will ignore reviews with rating of 3)

In [1]:
```
%matplotlib inline
import warnings
warnings.filterwarnings("ignore")
```

```
import sqlite3
import pandas as pd
import numpy as np
import nltk
import string
import matplotlib.pyplot as plt
import seaborn as sns
from sklearn.feature_extraction.text import TfidfTransformer
from sklearn.feature_extraction.text import TfidfVectorizer

from sklearn.feature_extraction.text import CountVectorizer
from sklearn.metrics import confusion_matrix
from sklearn import metrics
from sklearn.metrics import roc_curve, auc
from nltk.stem.porter import PorterStemmer

import re
# Tutorial about Python regular expressions: https://pymotw.com/2/re/
```

```
import string
from nltk.corpus import stopwords
from nltk.stem import PorterStemmer
from nltk.stem.wordnet import WordNetLemmatizer

from gensim.models import Word2Vec
from gensim.models import KeyedVectors
import pickle

from tqdm import tqdm
import os
```

2 [1]. Loading and Reading Data

```
In [2]: con = sqlite3.connect('Amazon_Food_Review\database.sqlite')

        #ignoring reviews with rating of 3 and selecting first 5000 reviews
        filtered_data = pd.read_sql_query(""" SELECT * FROM Reviews WHERE Score != 3 LIMIT 500

        # Gives reviews with Score>3 a positive rating, and reviews with a score<3 a negative
        def partition(x):
            if x < 3:
                return 0
            return 1

        #changing reviews rating to positive/negative
        actualScore = filtered_data['Score']
        positiveNegative = actualScore.map(partition)
        filtered_data['Score'] = positiveNegative
        print("Number of data points in our data", filtered_data.shape)
        filtered_data.head(5)
```

```
Number of data points in our data (5000, 10)
```

Out[2]:

	Id	ProductId	UserId	ProfileName	\
0	1	B001E4KFG0	A3SGXH7AUHU8GW	delmartian	
1	2	B00813GRG4	A1D87F6ZCVE5NK	dll pa	
2	3	B000LQOCH0	ABXLMWJIXXAIN	Natalia Corres "Natalia Corres"	
3	4	B000UA0QIQ	A395BORC6FGVXV	Karl	
4	5	B006K2ZZ7K	A1UQRSCLF8GW1T	Michael D. Bigham "M. Wassir"	

	HelpfulnessNumerator	HelpfulnessDenominator	Score	Time	\
0	1	1	1	1303862400	
1	0	0	0	1346976000	
2	1	1	1	1219017600	
3	3	3	0	1307923200	
4	0	0	1	1350777600	

```
                  Summary                                               Text
0  Good Quality Dog Food  I have bought several of the Vitality canned d...
1      Not as Advertised  Product arrived labeled as Jumbo Salted Peanut...
2  "Delight" says it all  This is a confection that has been around a fe...
3         Cough Medicine  If you are looking for the secret ingredient i...
4            Great taffy  Great taffy at a great price.  There was a wid...
```

3 Exploratory Data Analysis

3.1 [2] Data Cleaning: Deduplication

It is observed (as shown in the table below) that the reviews data had many duplicate entries. Hence it was necessary to remove duplicates in order to get unbiased results for the analysis of the data. Following is an example:

A product can have variants depending upon the quantity and flavours. But we will consider all the variants of a product as the same product as review to one of its variant is repeated for all its variant. Therefore we need to remove duplicate reviews of a product.

In [3]:
```
#checking for duplicates
display = pd.read_sql_query("""
SELECT UserId, ProductId, ProfileName, Time, Score, Text, COUNT(*)
FROM Reviews
GROUP BY UserId
HAVING COUNT(*)>1
""", con)

print(display.shape)
display.head()
```

```
(80668, 7)
```

Out[3]:
```
           UserId   ProductId             ProfileName        Time  Score  \
0  #oc-R115TNMSPFT9I7  B007Y59HVM                 Breyton  1331510400      2
1  #oc-R11D9D7SHXIJB9  B005HG9ET0  Louis E. Emory "hoppy"  1342396800      5
2  #oc-R11DNU2NBKQ23Z  B007Y59HVM        Kim Cieszykowski  1348531200      1
3  #oc-R11O5J5ZVQE25C  B005HG9ET0           Penguin Chick  1346889600      5
4  #oc-R12KPBODL2B5ZD  B007OSBE1U   Christopher P. Presta  1348617600      1

                                                Text  COUNT(*)
0  Overall its just OK when considering the price...         2
1  My wife has recurring extreme muscle spasms, u...         3
2  This coffee is horrible and unfortunately not ...         2
3  This will be the bottle that you grab from the...         3
4  I didnt like this coffee. Instead of telling y...         2
```

In [6]:
```
#Sorting data according to ProductId in ascending order
sorted_data=filtered_data.sort_values('ProductId', axis=0, ascending=True, inplace=Fals
```

```
In [7]: #Deduplication of entries
        final=sorted_data.drop_duplicates(subset={"UserId","ProfileName","Time","Text"}, keep=
        final.shape

Out[7]: (4986, 10)
```

Practically HelpfulnessNumerator shoud be smaller than HelpfulnessDenominator, therfore we can also check this constraint to make sure given data make sense.

```
In [8]: display= pd.read_sql_query("""
        SELECT *
        FROM Reviews
        WHERE HelpfulnessNumerator > HelpfulnessDenominator
        ORDER BY ProductID
        """, con)

        display.head()

Out[8]:       Id   ProductId          UserId               ProfileName  \
        0  64422  B000MIDROQ  A161DK06JJMCYF  J. E. Stephens "Jeanne"
        1  44737  B001EQ55RW  A2V0I904FH7ABY                      Ram

           HelpfulnessNumerator  HelpfulnessDenominator  Score        Time  \
        0                     3                       1      5  1224892800
        1                     3                       2      4  1212883200

                                              Summary  \
        0            Bought This for My Son at College
        1  Pure cocoa taste with crunchy almonds inside

                                                    Text
        0  My son loves spaghetti so I didn't hesitate or...
        1  It was almost a 'love at first bite' - the per...

In [9]: final=final[final.HelpfulnessNumerator<=final.HelpfulnessDenominator]

In [10]: print(final.shape)

         #Number of positive and negative points
         final['Score'].value_counts()

(4986, 10)


Out[10]: 1    4178
         0     808
         Name: Score, dtype: int64
```

4 [3]. Text Preprocessing.

We can print some random reviews to observe the text and identify what text preprocessing may be needed.

```
In [76]: sent_0 = final['Text'].values[0]
         print(sent_0)
         print("="*50)

         sent_1000 = final['Text'].values[1000]
         print(sent_1000)
         print("="*50)

         sent_1500 = final['Text'].values[1500]
         print(sent_1500)
         print("="*50)

         sent_4900 = final['Text'].values[4900]
         print(sent_4900)
         print("="*50)
```

```
Why is this $[...] when the same product is available for $[...] here?<br />http://www.amazon.
==================================================
I recently tried this flavor/brand and was surprised at how delicious these chips are.  The bes
==================================================
Wow.  So far, two two-star reviews.  One obviously had no idea what they were ordering; the otl
==================================================
love to order my coffee on amazon.  easy and shows up quickly.<br />This k cup is great coffee
==================================================
```

We can observe that there are html tags, URLs, punctuations etc.
Therefore in text preprocessing we will do the following-

1. remove html tags and URLs
2. remove punctuations and special characters
3. remove if any alphanumeric words are present as they will not make much sense
4. removing words with length less than equal to 2 as they do not make much meaning to text.
5. converting words to lowercase
6. Remove stopwords
7. Snowball stemming

```
In [77]: #Removing URLs
         # remove urls from text python: https://stackoverflow.com/a/40823105/4084039

         sent_0 = re.sub(r"http\S+", "", sent_0)
         sent_1000 = re.sub(r"http\S+", "", sent_1000)
         sent_150 = re.sub(r"http\S+", "", sent_1500)
         sent_4900 = re.sub(r"http\S+", "", sent_4900)
```

```
print(sent_0)
print(sent_1000)
```

```
Why is this $[...] when the same product is available for $[...] here?<br /> /><br />The Victo
I recently tried this flavor/brand and was surprised at how delicious these chips are.  The bes
```

In [78]:
```
#Removing all tags
# https://stackoverflow.com/questions/16206380/python-beautifulsoup-how-to-remove-all
from bs4 import BeautifulSoup

print(sent_0)
soup = BeautifulSoup(sent_0, 'lxml')
text = soup.get_text()
print(text)
print("="*50)

soup = BeautifulSoup(sent_1000, 'lxml')
text = soup.get_text()
print(text)
print("="*50)

soup = BeautifulSoup(sent_1500, 'lxml')
text = soup.get_text()
print(text)
print("="*50)

soup = BeautifulSoup(sent_4900, 'lxml')
text = soup.get_text()
print(text)
```

```
Why is this $[...] when the same product is available for $[...] here?<br /> /><br />The Victo
Why is this $[...] when the same product is available for $[...] here? />The Victor M380 and M
==================================================
I recently tried this flavor/brand and was surprised at how delicious these chips are.  The bes
==================================================
Wow.  So far, two two-star reviews.  One obviously had no idea what they were ordering; the otl
==================================================
love to order my coffee on amazon.  easy and shows up quickly.This k cup is great coffee.  dca
```

In [79]:
```
# https://stackoverflow.com/a/47091490/4084039
import re

def decontracted(phrase):
    # specific
    phrase = re.sub(r"won't", "will not", phrase)
    phrase = re.sub(r"can\'t", "can not", phrase)
```

```
    # general
    phrase = re.sub(r"n\'t", " not", phrase)
    phrase = re.sub(r"\'re", " are", phrase)
    phrase = re.sub(r"\'s", " is", phrase)
    phrase = re.sub(r"\'d", " would", phrase)
    phrase = re.sub(r"\'ll", " will", phrase)
    phrase = re.sub(r"\'t", " not", phrase)
    phrase = re.sub(r"\'ve", " have", phrase)
    phrase = re.sub(r"\'m", " am", phrase)
    return phrase
```

```
In [80]: sent_1500 = decontracted(sent_1500)
         print(sent_1500)
         print("="*50)
```

```
Wow.  So far, two two-star reviews.  One obviously had no idea what they were ordering; the otl
==================================================
```

```
In [81]: #remove words with numbers python: https://stackoverflow.com/a/18082370/4084039
         sent_0 = re.sub("\S*\d\S*", "", sent_0).strip()
         print(sent_0)
```

```
Why is this $[...] when the same product is available for $[...] here?<br /> /><br />The Victo
```

```
In [82]: #remove spacial character: https://stackoverflow.com/a/5843547/4084039
         sent_1500 = re.sub('[^A-Za-z0-9]+', ' ', sent_1500)
         print(sent_1500)
```

```
Wow So far two two star reviews One obviously had no idea what they were ordering the other wa
```

```
In [83]: # https://gist.github.com/sebleier/554280
         # we are removing the words from the stop words list: 'no', 'nor', 'not'
         # <br /><br /> ==> after the above steps, we are getting "br br"
         # we are including them into stop words list
         # instead of <br /> if we have <br/> these tags would have revmoved in the 1st step

         stopwords= set(['br', 'the', 'i', 'me', 'my', 'myself', 'we', 'our', 'ours', 'ourselv
                     "you'll", "you'd", 'your', 'yours', 'yourself', 'yourselves', 'he', 'him'
                     'she', "she's", 'her', 'hers', 'herself', 'it', "it's", 'its', 'itself',
                     'theirs', 'themselves', 'what', 'which', 'who', 'whom', 'this', 'that', "t
                     'am', 'is', 'are', 'was', 'were', 'be', 'been', 'being', 'have', 'has', 'h
                     'did', 'doing', 'a', 'an', 'the', 'and', 'but', 'if', 'or', 'because', 'as
                     'at', 'by', 'for', 'with', 'about', 'against', 'between', 'into', 'throug
                     'above', 'below', 'to', 'from', 'up', 'down', 'in', 'out', 'on', 'off', 'o
                     'then', 'once', 'here', 'there', 'when', 'where', 'why', 'how', 'all', 'a
```

```
                    'most', 'other', 'some', 'such', 'only', 'own', 'same', 'so', 'than', 'too
                    's', 't', 'can', 'will', 'just', 'don', "don't", 'should', "should've", 'n
                    've', 'y', 'ain', 'aren', "aren't", 'couldn', "couldn't", 'didn', "didn't"
                    "hadn't", 'hasn', "hasn't", 'haven', "haven't", 'isn', "isn't", 'ma', 'mig
                    "mustn't", 'needn', "needn't", 'shan', "shan't", 'shouldn', "shouldn't",
                    'won', "won't", 'wouldn', "wouldn't"])
```

```
In [84]: # Combining all the above stundents
         from tqdm import tqdm
         preprocessed_reviews = []
         # tqdm is for printing the status bar
         for sentance in tqdm(final['Text'].values):
             sentance = re.sub(r"http\S+", "", sentance)
             sentance = BeautifulSoup(sentance, 'lxml').get_text()
             sentance = decontracted(sentance)
             sentance = re.sub("\S*\d\S*", "", sentance).strip()
             sentance = re.sub('[^A-Za-z]+', ' ', sentance)
             # https://gist.github.com/sebleier/554280
             sentance = ' '.join(e.lower() for e in sentance.split() if e.lower() not in stopwo
             preprocessed_reviews.append(sentance.strip())
```

```
100%|| 4986/4986 [00:02<00:00, 1884.96it/s]
```

```
In [85]: preprocessed_reviews[1500]
```

```
Out[85]: 'wow far two two star reviews one obviously no idea ordering wants crispy cookies hey
```

[3.2] Preprocess Summary

```
In [86]: ## Similartly you can do preprocessing for review summary also.
```

5 [4] Featurization

5.1 [4.1] BAG OF WORDS

```
In [134]: #BoW
          count_vect = CountVectorizer() #in scikit-learn
          count_vect.fit(preprocessed_reviews)
          print("some feature names ", count_vect.get_feature_names()[:10])
          print('='*50)

          final_counts = count_vect.transform(preprocessed_reviews)
          print("the type of count vectorizer ",type(final_counts))
          print("the shape of out text BOW vectorizer ",final_counts.get_shape())
          print("the number of unique words ", final_counts.get_shape()[1])
```

```
some feature names  ['aa', 'aahhhs', 'aback', 'abandon', 'abates', 'abbott', 'abby', 'abdominal
==================================================
```

```
the type of count vectorizer  <class 'scipy.sparse.csr.csr_matrix'>
the shape of out text BOW vectorizer  (4986, 12997)
the number of unique words  12997
```

```
In [135]: final_counts.shape

          print(final_counts)
```

```
  (0, 733)        1
  (0, 2604)        1
  (0, 4460)        1
  (0, 4766)        1
  (0, 7479)        1
  (0, 8733)        1
  (0, 8815)        1
  (0, 9617)        1
  (0, 10946)        1
  (0, 11819)        1
  (0, 11889)        1
  (0, 12185)        1
  (0, 12403)        1
  (1, 810)        1
  (1, 937)        1
  (1, 1521)        1
  (1, 4460)        1
  (1, 5013)        1
  (1, 7610)        1
  (1, 8815)        1
  (1, 10034)        1
  (1, 12265)        1
  (1, 12403)        1
  (2, 944)        1
  (2, 1567)        1
  :        :
  (4985, 1244)        1
  (4985, 1508)        1
  (4985, 1626)        1
  (4985, 2166)        1
  (4985, 3503)        1
  (4985, 4900)        1
  (4985, 6520)        1
  (4985, 6693)        1
  (4985, 6924)        1
  (4985, 7610)        4
  (4985, 7788)        1
  (4985, 8693)        1
  (4985, 9076)        1
```

```
  (4985, 10239)        1
  (4985, 10370)        1
  (4985, 10462)        1
  (4985, 11210)        1
  (4985, 11219)        1
  (4985, 11429)        1
  (4985, 11622)        1
  (4985, 11890)        1
  (4985, 12265)        1
  (4985, 12274)        1
  (4985, 12569)        1
  (4985, 12587)        1
```

5.2 [4.2] Bi-Grams and n-Grams.

In [89]:
```
#bi-gram, tri-gram and n-gram

        #removing stop words like "not" should be avoided before building n-grams
        # count_vect = CountVectorizer(ngram_range=(1,2))
        # please do read the CountVectorizer documentation http://scikit-learn.org/stable/mod
        # you can choose these numebrs min_df=10, max_features=5000, of your choice

        count_vect = CountVectorizer(ngram_range=(1,1), min_df=10, max_features=5000)
        final_bigram_counts = count_vect.fit_transform(preprocessed_reviews)
        print("the type of count vectorizer ",type(final_bigram_counts))
        print("the shape of out text BOW vectorizer ",final_bigram_counts.get_shape())
        print("the number of unique words including both unigrams ", final_bigram_counts.get_s

        count_vect = CountVectorizer(ngram_range=(2,2), min_df=10, max_features=5000)
        final_bigram_counts = count_vect.fit_transform(preprocessed_reviews)
        print("the shape of out text BOW vectorizer ",final_bigram_counts.get_shape())
        print("the number of unique words including both bigrams ", final_bigram_counts.get_sh

        count_vect = CountVectorizer(ngram_range=(1,2), min_df=10, max_features=5000)
        final_bigram_counts = count_vect.fit_transform(preprocessed_reviews)
        print("the shape of out text BOW vectorizer ",final_bigram_counts.get_shape())
        print("the number of unique words including both unigrams and bigrams ", final_bigram_
```

```
the type of count vectorizer  <class 'scipy.sparse.csr.csr_matrix'>
the shape of out text BOW vectorizer  (4986, 2218)
the number of unique words including both unigrams  2218
the shape of out text BOW vectorizer  (4986, 926)
the number of unique words including both bigrams  926
the shape of out text BOW vectorizer  (4986, 3144)
the number of unique words including both unigrams and bigrams  3144
```

5.3 [4.3] TF-IDF

```
In [90]: tf_idf_vect = TfidfVectorizer(ngram_range=(1,2), min_df=10)
         tf_idf_vect.fit(preprocessed_reviews)
         print("some sample features(unique words in the corpus)",tf_idf_vect.get_feature_names
         print('='*50)

         final_tf_idf = tf_idf_vect.transform(preprocessed_reviews)
         print("the type of count vectorizer ",type(final_tf_idf))
         print("the shape of out text TFIDF vectorizer ",final_tf_idf.get_shape())
         print("the number of unique words including both unigrams and bigrams ", final_tf_idf
```

```
some sample features(unique words in the corpus) ['ability', 'able', 'able find', 'able get',
==================================================
the type of count vectorizer  <class 'scipy.sparse.csr.csr_matrix'>
the shape of out text TFIDF vectorizer  (4986, 3144)
the number of unique words including both unigrams and bigrams  3144
```

5.4 [4.4] Word2Vec

```
In [91]: # Train your own Word2Vec model using your own text corpus
         i=0
         list_of_sentance=[]
         for sentance in preprocessed_reviews:
             list_of_sentance.append(sentance.split())
```

```
In [146]: # http://kavita-ganesan.com/gensim-word2vec-tutorial-starter-code/#.W17SRFAzZPY

          w2v_model=Word2Vec(list_of_sentance,min_count=5,size=50, workers=4)
          print(w2v_model.wv.most_similar('great'))
          print('='*50)
          print(w2v_model.wv.most_similar('worst'))
```

```
[('excellent', 0.9968411326408386), ('especially', 0.9964055418968201), ('intake', 0.995507657
==================================================
[('experience', 0.9994966983795166), ('awful', 0.9994459748268127), ('gold', 0.999439537525177
```

```
In [93]: w2v_words = list(w2v_model.wv.vocab)
         print("number of words that occured minimum 5 times ",len(w2v_words))
         print("sample words ", w2v_words[0:50])
```

```
number of words that occured minimum 5 times  3817
sample words  ['product', 'available', 'course', 'total', 'pretty', 'stinky', 'right', 'nearby
```

5.5 [4.4.1] Converting text into vectors using wAvg W2V, TFIDF-W2V

[4.4.1.1] Avg W2v

In [94]:
```
# average Word2Vec
# compute average word2vec for each review.
sent_vectors = []; # the avg-w2v for each sentence/review is stored in this list
for sent in tqdm(list_of_sentance): # for each review/sentence
    sent_vec = np.zeros(50) # as word vectors are of zero length 50, you might need t
    cnt_words =0; # num of words with a valid vector in the sentence/review
    for word in sent: # for each word in a review/sentence
        if word in w2v_words:
            vec = w2v_model.wv[word]
            sent_vec += vec
            cnt_words += 1
    if cnt_words != 0:
        sent_vec /= cnt_words
    sent_vectors.append(sent_vec)
print(len(sent_vectors))
print(len(sent_vectors[0]))
```

```
100%|| 4986/4986 [00:06<00:00, 796.95it/s]

4986
50
```

[4.4.1.2] TFIDF weighted W2v

In [95]:
```
# S = ["abc def pqr", "def def def abc", "pqr pqr def"]
model = TfidfVectorizer()
model.fit(preprocessed_reviews)
# we are converting a dictionary with word as a key, and the idf as a value
dictionary = dict(zip(model.get_feature_names(), list(model.idf_)))
```

In [96]:
```
# TF-IDF weighted Word2Vec
tfidf_feat = model.get_feature_names() # tfidf words/col-names
# final_tf_idf is the sparse matrix with row= sentence, col=word and cell_val = tfidf

tfidf_sent_vectors = []; # the tfidf-w2v for each sentence/review is stored in this l
row=0;
for sent in tqdm(list_of_sentance): # for each review/sentence
    sent_vec = np.zeros(50) # as word vectors are of zero length
    weight_sum =0; # num of words with a valid vector in the sentence/review
    for word in sent: # for each word in a review/sentence
        if word in w2v_words and word in tfidf_feat:
            vec = w2v_model.wv[word]
#             tf_idf = tf_idf_matrix[row, tfidf_feat.index(word)]
            # to reduce the computation we are
            # dictionary[word] = idf value of word in whole courpus
```

```
                    # sent.count(word) = tf valeus of word in this review
                    tf_idf = dictionary[word]*(sent.count(word)/len(sent))
                    sent_vec += (vec * tf_idf)
                    weight_sum += tf_idf
            if weight_sum != 0:
                sent_vec /= weight_sum
            tfidf_sent_vectors.append(sent_vec)
            row += 1
```

```
100%|| 4986/4986 [00:35<00:00, 139.65it/s]
```

6 KNN on BOG

```
In [97]: type(tfidf_sent_vectors)
```

```
Out[97]: list
```

```
In [98]: tfidf_sent_vectors[1]
```

```
Out[98]: array([-0.64786634,  0.0156483 ,  0.56561308,  0.04209164, -0.55496904,
                -0.18347869,  0.60473286, -0.50051789,  0.40580555,  0.06499073,
                -0.25700069, -0.01015892,  0.38030735,  0.26323603,  0.03702686,
                -0.03800285,  0.15453433, -0.35752802,  0.06056566, -0.29540167,
                -0.01408095,  0.27466547, -0.25305963, -0.42376791, -0.11983078,
                 0.05320868, -0.42532639,  0.75630956,  0.13839521, -0.70666067,
                -0.24815073, -0.06387648,  0.19691942, -0.25955613,  0.24401081,
                 0.25294161, -0.05304605, -0.23085085, -0.19603802, -0.00149323,
                -0.2459258 ,  0.25613215,  0.24925459,  0.29020263, -0.00579971,
                -0.70504623,  0.44378778, -0.15514344,  0.23615679,  0.01952577])
```

```
In [99]: y = final['Score'].values
```

```
In [100]: len(tfidf_sent_vectors)
```

```
Out[100]: 4986
```

```
In [101]: test = 'this is a bad useless expensive wrong costly poor '
          test = [test]
          bog_test = count_vect.transform(test)
          rating = 1
```

```
In [102]: type(bog_test)
```

```
Out[102]: scipy.sparse.csr.csr_matrix
```

```
In [103]: print(bog_test)
```

```
  (0, 152)        1
  (0, 873)        1
  (0, 2093)        1
  (0, 3121)        1
```

In [104]:
```
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
from sklearn.model_selection import train_test_split
from sklearn.neighbors import KNeighborsClassifier
from sklearn.metrics import accuracy_score
from sklearn.model_selection import cross_val_score
from collections import Counter
from sklearn.metrics import accuracy_score
from sklearn import model_selection
```

In [144]:
```
# knn = KNeighborsClassifier(10)
# knn.fit(final_counts, y)
# pred = knn.predict(bog_test)
# print(pred)
```

In [106]:
```
X_1, X_test, y_1, y_test = model_selection.train_test_split(final_counts, y, test_si

# split the train data set into cross validation train and cross validation test
X_tr, X_cv, y_tr, y_cv = model_selection.train_test_split(X_1, y_1, test_size=0.3,ra

for i in range(1,30,2):
    # instantiate learning model (k = 30)
    knn = KNeighborsClassifier(n_neighbors=i)

    # fitting the model on crossvalidation train
    knn.fit(X_tr, y_tr)

    # predict the response on the crossvalidation train
    pred = knn.predict(X_cv)

    # evaluate CV accuracy
    acc = accuracy_score(y_cv, pred, normalize=True) * float(100)
    print('\nCV accuracy for k = %d is %d%%' % (i, acc))

knn = KNeighborsClassifier(1)
knn.fit(X_tr,y_tr)
pred = knn.predict(X_test)
acc = accuracy_score(y_test, pred, normalize=True) * float(100)
print('\n****Test accuracy for k = 1 is %d%%' % (acc))
```

```
CV accuracy for k = 1 is 75%
```

```
CV accuracy for k = 3 is 80%

CV accuracy for k = 5 is 82%

CV accuracy for k = 7 is 82%

CV accuracy for k = 9 is 83%

CV accuracy for k = 11 is 83%

CV accuracy for k = 13 is 83%

CV accuracy for k = 15 is 83%

CV accuracy for k = 17 is 83%

CV accuracy for k = 19 is 82%

CV accuracy for k = 21 is 82%

CV accuracy for k = 23 is 82%

CV accuracy for k = 25 is 82%

CV accuracy for k = 27 is 82%

CV accuracy for k = 29 is 82%

****Test accuracy for k = 1 is 77%
```

In [110]:

```
Accuracy of the model at optimal hyperparameter alpha = 0 is:  85.026738%
f1 score value for   the model is: 0.7960278482695767
precision score  for   the model is: 0.8579197824609109
```

In [111]:
```
#final_counts=final_counts.toarray() #converting spare matrix to dense matrix
```

In [143]:
```
# from sklearn.naive_bayes import GaussianNB
# X_train, X_test, y_train, y_test =model_selection.train_test_split(final_counts, y
# gnb = GaussianNB()
# y_pred = gnb.fit(X_train, y_train).predict(X_test)

# acc = accuracy_score(y_test, y_pred, normalize=True) * float(100)
# print("Number of mislabeled points out of a total %d points : %d"
#       % (X_test.shape[0], (y_test != y_pred).sum()))
```

```
In [137]: from sklearn.naive_bayes import GaussianNB
          from sklearn.naive_bayes import MultinomialNB

          alpha_values = [0.00001,0.0001,0.001,0.01,0.1,1,10,100,1000,10000,100000]#alpha from
          for i in alpha_values:
              X_train, X_test, y_train, y_test =model_selection.train_test_split(final_counts,
              nb = MultinomialNB(alpha=i).fit(X_train, y_train)
              y_pred = nb.predict(X_test)
              acc = accuracy_score(y_test, y_pred, normalize=True) * float(100)
              print("Number of mislabeled points out of a total %d points and aplha= %f : %d" %
              print("accuracy= %f "%(((X_test.shape[0]-(y_test != y_pred).sum()))/X_test.shape
```

```
Number of mislabeled points out of a total 998 points and aplha= 0.000010 : 107
accuracy= 0.892786
Number of mislabeled points out of a total 998 points and aplha= 0.000100 : 104
accuracy= 0.895792
Number of mislabeled points out of a total 998 points and aplha= 0.001000 : 101
accuracy= 0.898798
Number of mislabeled points out of a total 998 points and aplha= 0.010000 : 93
accuracy= 0.906814
Number of mislabeled points out of a total 998 points and aplha= 0.100000 : 101
accuracy= 0.898798
Number of mislabeled points out of a total 998 points and aplha= 1.000000 : 104
accuracy= 0.895792
Number of mislabeled points out of a total 998 points and aplha= 10.000000 : 136
accuracy= 0.863727
Number of mislabeled points out of a total 998 points and aplha= 100.000000 : 136
accuracy= 0.863727
Number of mislabeled points out of a total 998 points and aplha= 1000.000000 : 136
accuracy= 0.863727
Number of mislabeled points out of a total 998 points and aplha= 10000.000000 : 136
accuracy= 0.863727
Number of mislabeled points out of a total 998 points and aplha= 100000.000000 : 136
accuracy= 0.863727
```

```
In [138]: optimal_alpha=0.010000
```

6.1 performance metrics

6.2 Naive Bayes

```
In [140]: from sklearn.metrics import plot_confusion_matrix
          from sklearn.metrics import precision_score
          from sklearn.metrics import accuracy_score, precision_recall_fscore_support, f1_score
          nb = MultinomialNB(alpha=optimal_alpha)
          nb=nb.fit(X_train, y_train)
          Y_pred = nb.predict(X_test)
```

```
        Y_test_accuracy = accuracy_score(y_test, Y_pred, normalize=True, sample_weight=None)*
        print('Accuracy of the model at optimal hyperparameter alpha = %d is:  %f%%' % (optim

        #print('Confusion matrix for the model is:')
        #plot_confusion_matrix(nb,X=y_test , y_true=Y_pred)

        f1score= f1_score(y_test, Y_pred, average='weighted')
        print('f1 score value for   the model is: %s'% f1score)


        precisionscore=precision_score(y_test, Y_pred)
        print('precision score  for   the model is: %s'% precisionscore)
```

```
Accuracy of the model at optimal hyperparameter alpha = 0 is:  90.681363%
f1 score value for   the model is: 0.903822434689302
precision score  for   the model is: 0.9364358683314415
```

6.3 KNN

```
In [141]: from sklearn.metrics import plot_confusion_matrix
          from sklearn.metrics import precision_score
          from sklearn.metrics import accuracy_score, precision_recall_fscore_support, f1_score

          X_1, X_test, y_1, y_test = model_selection.train_test_split(final_counts, y, test_size
          knn = KNeighborsClassifier(n_neighbors=9)
          knn.fit(X_tr, y_tr)
          pred = knn.predict(X_test)


          Y_test_accuracy = accuracy_score(y_test, pred, normalize=True) * float(100)
          print('Accuracy of the model at optimal hyperparameter alpha = %d is:  %f%%' % (optim

          #print('Confusion matrix for the model is:')
          #plot_confusion_matrix(y_test, Y_pred)

          f1score= f1_score(y_test, pred, average='weighted')
          print('f1 score value for   the model is: %s'% f1score)


          precisionscore=precision_score(y_test, pred)
          print('precision score  for   the model is: %s'% precisionscore)
```

```
Accuracy of the model at optimal hyperparameter alpha = 0 is:  85.026738%
f1 score value for   the model is: 0.7960278482695767
precision score  for   the model is: 0.8579197824609109
```

7 #KNN on tf-idf

In [ ]:
```
X_1, X_test, y_1, y_test = cross_validation.train_test_split(final_tf_idf, y, test_size

# split the train data set into cross validation train and cross validation test
X_tr, X_cv, y_tr, y_cv = cross_validation.train_test_split(X_1, y_1, test_size=0.3)

for i in range(1,30,2):
    # instantiate learning model (k = 30)
    knn = KNeighborsClassifier(n_neighbors=i)

    # fitting the model on crossvalidation train
    knn.fit(X_tr, y_tr)

    # predict the response on the crossvalidation train
    pred = knn.predict(X_cv)

    # evaluate CV accuracy
    acc = accuracy_score(y_cv, pred, normalize=True) * float(100)
    print('\nCV accuracy for k = %d is %d%%' % (i, acc))

knn = KNeighborsClassifier(1)
knn.fit(X_tr,y_tr)
pred = knn.predict(X_test)
acc = accuracy_score(y_test, pred, normalize=True) * float(100)
print('\n****Test accuracy for k = 1 is %d%%' % (acc))
```

8 KNN on AvgW2V

In [ ]:
```
X_1, X_test, y_1, y_test = cross_validation.train_test_split(sent_vectors, y, test_size

# split the train data set into cross validation train and cross validation test
X_tr, X_cv, y_tr, y_cv = cross_validation.train_test_split(X_1, y_1, test_size=0.3)

for i in range(1,30,2):
    # instantiate learning model (k = 30)
    knn = KNeighborsClassifier(n_neighbors=i)

    # fitting the model on crossvalidation train
    knn.fit(X_tr, y_tr)

    # predict the response on the crossvalidation train
    pred = knn.predict(X_cv)

    # evaluate CV accuracy
    acc = accuracy_score(y_cv, pred, normalize=True) * float(100)
    print('\nCV accuracy for k = %d is %d%%' % (i, acc))
```

```
knn = KNeighborsClassifier(1)
knn.fit(X_tr,y_tr)
pred = knn.predict(X_test)
acc = accuracy_score(y_test, pred, normalize=True) * float(100)
print('\n****Test accuracy for k = 1 is %d%%' % (acc))
```

9 KNN on tf-idf weighted W2V

```
In [ ]: X_1, X_test, y_1, y_test = cross_validation.train_test_split(tfidf_sent_vectors, y, tes

        # split the train data set into cross validation train and cross validation test
        X_tr, X_cv, y_tr, y_cv = cross_validation.train_test_split(X_1, y_1, test_size=0.3)

        for i in range(1,30,2):
            # instantiate learning model (k = 30)
            knn = KNeighborsClassifier(n_neighbors=i)

            # fitting the model on crossvalidation train
            knn.fit(X_tr, y_tr)

            # predict the response on the crossvalidation train
            pred = knn.predict(X_cv)

            # evaluate CV accuracy
            acc = accuracy_score(y_cv, pred, normalize=True) * float(100)
            print('\nCV accuracy for k = %d is %d%%' % (i, acc))

        knn = KNeighborsClassifier(1)
        knn.fit(X_tr,y_tr)
        pred = knn.predict(X_test)
        acc = accuracy_score(y_test, pred, normalize=True) * float(100)
        print('\n****Test accuracy for k = 1 is %d%%' % (acc))
```