

# 05 Amazon Fine Food Reviews Analysis\_Logistic Regression

March 26, 2019

## 1 Amazon Fine Food Reviews Analysis

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

EDA: <https://nycdatascience.com/blog/student-works/amazon-fine-foods-visualization/>

The Amazon Fine Food Reviews dataset consists of reviews of fine foods from Amazon.

Number of reviews: 568,454 Number of users: 256,059 Number of products: 74,258 Timespan: Oct 1999 - Oct 2012 Number of Attributes/Columns in data: 10

Attribute Information:

1. Id
2. ProductId - unique identifier for the product
3. UserId - unique 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).

[Q] How to determine if a review is positive or negative? [Ans] We could use Score/Rating. A rating of 4 or 5 can be considered as a positive review. A rating of 1 or 2 can be considered as negative one. A review of rating 3 is considered neutral and such reviews are ignored from our analysis. This is an approximate and proxy way of determining the polarity (positivity/negativity) of a review.

## 2 [1]. Reading Data

### 2.1 [1.1] Loading the data

The dataset is available in two forms 1. .csv file 2. SQLite Database

In order to load the data, We have used the SQLITE dataset as it is easier to query the data and visualise the data efficiently.

Here as we only want to get the global sentiment of the recommendations (positive or negative), we will purposefully ignore all Scores equal to 3. If the score is above 3, then the recommendation will be set to "positive". Otherwise, it will be set to "negative".

```
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

In [2]: # using SQLite Table to read data.
con = sqlite3.connect('database.sqlite')

# filtering only positive and negative reviews i.e.
# not taking into consideration those reviews with Score=3
# SELECT * FROM Reviews WHERE Score != 3 LIMIT 500000, will give top 500000 data points.
# you can change the number to any other number based on your computing power

# filtered_data = pd.read_sql_query(""" SELECT * FROM Reviews WHERE Score != 3 LIMIT 500000 """, con)
```

```

# for tsne assignment you can take 5k data points

filtered_data = pd.read_sql_query(""" SELECT * FROM Reviews WHERE Score != 3 LIMIT 150000

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

#changing reviews with score less than 3 to be positive and vice-versa
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(3)

```

Number of data points in our data (150000, 10)

```

Out[2]:
   Id  ProductId  UserId  ProfileName \
0   1  B001E4KFG0  A3SGXH7AUHU8GW  delmartian
1   2  B00813GRG4  A1D87F6ZCVE5NK  dll pa
2   3  B000LQOCHO  ABXLMWJIXXAIN  Natalia Corres "Natalia Corres"

   HelpfulnessNumerator  HelpfulnessDenominator  Score  Time \
0                      1                      1      1  1303862400
1                      0                      0      0  1346976000
2                      1                      1      1  1219017600

   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...

```

```

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

```

```

In [4]: print(display.shape)
display.head()

```

(80668, 7)

```

Out[4]:
   UserId  ProductId  ProfileName  Time  Score \
0  #oc-R115TNMSPFT9I7  B007Y59HVM  Breyton  1331510400  2

```

1	#oc-R11D9D7SHXIJB9	B005HG9ETO	Louis E. Emory "hoppy"	1342396800	5
2	#oc-R11DNU2NBKQ23Z	B007Y59HVM	Kim Cieszykowski	1348531200	1
3	#oc-R1105J5ZVQE25C	B005HG9ETO	Penguin Chick	1346889600	5
4	#oc-R12KPB0DL2B5ZD	B0070SBE1U	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 [5]: display[display['UserId']=='AZY10LLTJ71NX']
```

```
Out [5]:
```

	UserId	ProductId	ProfileName	Time \
80638	AZY10LLTJ71NX	B006P7E5ZI	undertheshrine "undertheshrine"	1334707200

	Score	Text	COUNT(*)
80638	5	I was recommended to try green tea extract to ...	5

```
In [6]: display['COUNT(*)'].sum()
```

```
Out [6]: 393063
```

### 3 [2] Exploratory Data Analysis

#### 3.1 [2.1] 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:

```
In [7]: display= pd.read_sql_query("""
SELECT *
FROM Reviews
WHERE Score != 3 AND UserId="AR5J8UI46CURR"
ORDER BY ProductID
""", con)
display.head()
```

```
Out [7]:
```

	Id	ProductId	UserId	ProfileName	HelpfulnessNumerator \
0	78445	B000HDL1RQ	AR5J8UI46CURR	Geetha Krishnan	2
1	138317	B000HDOPYC	AR5J8UI46CURR	Geetha Krishnan	2
2	138277	B000HDOPYM	AR5J8UI46CURR	Geetha Krishnan	2
3	73791	B000HDOPZG	AR5J8UI46CURR	Geetha Krishnan	2
4	155049	B000PAQ75C	AR5J8UI46CURR	Geetha Krishnan	2

	HelpfulnessDenominator	Score	Time \
0	2	5	1199577600

1	2	5	1199577600
2	2	5	1199577600
3	2	5	1199577600
4	2	5	1199577600

	Summary \
0	LOACKER QUADRATINI VANILLA WAFERS
1	LOACKER QUADRATINI VANILLA WAFERS
2	LOACKER QUADRATINI VANILLA WAFERS
3	LOACKER QUADRATINI VANILLA WAFERS
4	LOACKER QUADRATINI VANILLA WAFERS

	Text
0	DELICIOUS WAFERS. I FIND THAT EUROPEAN WAFERS ...
1	DELICIOUS WAFERS. I FIND THAT EUROPEAN WAFERS ...
2	DELICIOUS WAFERS. I FIND THAT EUROPEAN WAFERS ...
3	DELICIOUS WAFERS. I FIND THAT EUROPEAN WAFERS ...
4	DELICIOUS WAFERS. I FIND THAT EUROPEAN WAFERS ...

As it can be seen above that same user has multiple reviews with same values for HelpfulnessNumerator, HelpfulnessDenominator, Score, Time, Summary and Text and on doing analysis it was found that ProductId=B000HDOPZG was Loacker Quadratini Vanilla Wafer Cookies, 8.82-Ounce Packages (Pack of 8) ProductId=B000HDL1RQ was Loacker Quadratini Lemon Wafer Cookies, 8.82-Ounce Packages (Pack of 8) and so on

It was inferred after analysis that reviews with same parameters other than ProductId belonged to the same product just having different flavour or quantity. Hence in order to reduce redundancy it was decided to eliminate the rows having same parameters.

The method used for the same was that we first sort the data according to ProductId and then just keep the first similar product review and delete the others. for eg. in the above just the review for ProductId=B000HDL1RQ remains. This method ensures that there is only one representative for each product and deduplication without sorting would lead to possibility of different representatives still existing for the same product.

```
In [8]: #Sorting data according to ProductId in ascending order
sorted_data=filtered_data.sort_values('ProductId', axis=0, ascending=True, inplace=False)
```

```
In [9]: #Deduplication of entries
final=sorted_data.drop_duplicates(subset={"UserId","ProfileName","Time","Text"}, keep='first')
final.shape
```

```
Out[9]: (126359, 10)
```

```
In [10]: #Checking to see how much % of data still remains
(final['Id'].size*1.0)/(filtered_data['Id'].size*1.0)*100
```

```
Out[10]: 84.23933333333333
```

Observation:- It was also seen that in two rows given below the value of HelpfulnessNumerator is greater than HelpfulnessDenominator which is not practically possible hence these two rows too are removed from calculations

```

In [11]: display= pd.read_sql_query("""
SELECT *
FROM Reviews
WHERE Score != 3 AND Id=44737 OR Id=64422
ORDER BY ProductID
""", con)

display.head()

Out[11]:
```

	Id	ProductId	UserId	ProfileName	\
0	64422	B000MIDR0Q	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 [12]: final=final[final.HelpfulnessNumerator<=final.HelpfulnessDenominator]

In [13]: #Before starting the next phase of preprocessing lets see the number of entries left
print(final.shape)

#How many positive and negative reviews are present in our dataset?
final['Score'].value_counts()

(126357, 10)

Out[13]: 1    106326
         0     20031
         Name: Score, dtype: int64

```

## 4 [3] Preprocessing

### 4.1 [3.1]. Preprocessing Review Text

Now that we have finished deduplication our data requires some preprocessing before we go on further with analysis and making the prediction model.

Hence in the Preprocessing phase we do the following in the order below:-

1. Begin by removing the html tags

2. Remove any punctuations or limited set of special characters like , or . or # etc.
3. Check if the word is made up of english letters and is not alpha-numeric
4. Check to see if the length of the word is greater than 2 (as it was researched that there is no adjective in 2-letters)
5. Convert the word to lowercase
6. Remove Stopwords
7. Finally Snowball Stemming the word (it was observed to be better than Porter Stemming)

After which we collect the words used to describe positive and negative reviews

```
In [14]: # printing some random reviews
        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)
```

```
I grew up reading these Sendak books, and watching the Really Rosie movie that incorporates the
=====
Its about time Spanish products started getting their due.. The most famous (rightly so) Spanish
=====
I love this stuff. I nuke a mug a milk until it's very hot, drop in 2 of the triangles, stir u
=====
There's nothing like the scent of real lavender! Just a whiff smells so good. Besides enjoying
=====
```

```
In [15]: # 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)
```

```
I grew up reading these Sendak books, and watching the Really Rosie movie that incorporates the
```

```
In [16]: # https://stackoverflow.com/questions/16206380/python-beautifulsoup-how-to-remove-all
        from bs4 import BeautifulSoup
```

```

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)

```

I grew up reading these Sendak books, and watching the Really Rosie movie that incorporates the  
=====

Its about time Spanish products started getting their due.. The most famous (rightly so) Spani  
=====

I love this stuff. I nuke a mug a milk until it's very hot, drop in 2 of the triangles, stir u  
=====

There's nothing like the scent of real lavender! Just a whiff smells so good. Besides enjoying

```
In [17]: # 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 [18]: sent_1500 = decontracted(sent_1500)
```



```
print(sent_1500)
print("="*50)
```

I love this stuff. I nuke a mug a milk until it is very hot, drop in 2 of the triangles, stir  
=====

```
In [19]: #remove words with numbers python: https://stackoverflow.com/a/18082370/4084039
sent_0 = re.sub("\S*\d\S*", "", sent_0).strip()
print(sent_0)
```

I grew up reading these Sendak books, and watching the Really Rosie movie that incorporates the

```
In [20]: #remove spacial character: https://stackoverflow.com/a/5843547/4084039
sent_1500 = re.sub('[^A-Za-z0-9]+', ' ', sent_1500)
print(sent_1500)
```

I love this stuff I nuke a mug a milk until it is very hot drop in 2 of the triangles stir unt

```
In [21]: # 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 reumoved 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', 'l
    'did', 'doing', 'a', 'an', 'the', 'and', 'but', 'if', 'or', 'because', 'as
    'at', 'by', 'for', 'with', 'about', 'against', 'between', 'into', 'through
    'above', 'below', 'to', 'from', 'up', 'down', 'in', 'out', 'on', 'off', 'o
    'then', 'once', 'here', 'there', 'when', 'where', 'why', 'how', 'all', 'an
    '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', 'mi
    "mustn't", 'needn', "needn't", 'shan', "shan't", 'shouldn', "shouldn't",
    'won', "won't", 'wouldn', "wouldn't"])
```

```
In [22]: # Combining all the above stundents
from tqdm import tqdm
preprocessed_reviews = []
# tqdm is for printing the status bar
for sentence in tqdm(final['Text'].values):
    sentence = re.sub(r"http\S+", "", sentence)
```

```

sentence = BeautifulSoup(sentence, 'lxml').get_text()
sentence = decontracted(sentence)
sentence = re.sub("\S*\d\S*", "", sentence).strip()
sentence = re.sub('[^A-Za-z]+', ' ', sentence)
# https://gist.github.com/sebleier/554280
sentence = ' '.join(e.lower() for e in sentence.split() if e.lower() not in stopwords)
preprocessed_reviews.append(sentence.strip())

```

100%|| 126357/126357 [00:59<00:00, 2128.06it/s]

In [23]: preprocessed\_reviews[1500]

Out[23]: 'love stuff nuke mug milk hot drop triangles stir chocolate melts froth aerolatte simp

## 5 [4] Featurization

### 5.1 [4.1] BAG OF WORDS

In [24]: `from sklearn.model_selection import train_test_split`

```

# X_train, X_test, y_train, y_test = train_test_split(X, Y, test_size=0.33, shuffle=False)
X=np.asarray(preprocessed_reviews[:50000])
Y=final['Score'].values
print(X.shape, " ", Y.shape)
X_train, X_test, y_train, y_test = train_test_split(X, Y[:50000], test_size=0.33) # train
X_train, X_cv, y_train, y_cv = train_test_split(X_train, y_train, test_size=0.33) # test

```

(50000,) (126357,)

In [25]: `#BoW`

```

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

final_countsXtrain = count_vect.transform(X_train)
final_countsXtest = count_vect.transform(X_test)
final_countsXcv = count_vect.transform(X_cv)
print("the shape of out text BOW vectorizer xtrain ", final_countsXtrain.get_shape())
print("the shape of out text BOW vectorizer xtest ", final_countsXtest.get_shape())
print("the shape of out text BOW vectorizer xcv ", final_countsXcv.get_shape())

```

some feature names ['aa', 'aaaa', 'aaah', 'aachen', 'aafco', 'aagh', 'aahs', 'aap', 'ab', 'ab']  
=====

the shape of out text BOW vectorizer xtrain (22445, 29261)  
the shape of out text BOW vectorizer xtest (16500, 29261)  
the shape of out text BOW vectorizer xcv (11055, 29261)

## 5.2 [4.3] TF-IDF

```
In [26]: tf_idf_vect = TfidfVectorizer(ngram_range=(1,2), min_df=10)
        tf_idf_vect.fit(X_train)
        print("some sample features(unique words in the corpus)",tf_idf_vect.get_feature_names())
        print('='*50)

        final_tf_idfXtrain = tf_idf_vect.transform(X_train)
        final_tf_idfXtest = tf_idf_vect.transform(X_test)
        final_tf_idfXcv = tf_idf_vect.transform(X_cv)
        #print("the type of count vectorizer ",type(final_tf_idf))
        print("the shape of out text TFIDF vectorizer xtrain ",final_tf_idfXtrain.get_shape())
        print("the shape of out text TFIDF vectorizer xtest ",final_tf_idfXtest.get_shape())
        print("the shape of out text TFIDF vectorizer xcv ",final_tf_idfXcv.get_shape())

some sample features(unique words in the corpus) ['abdominal', 'ability', 'able', 'able buy',
=====
the shape of out text TFIDF vectorizer xtrain (22445, 12990)
the shape of out text TFIDF vectorizer xtest (16500, 12990)
the shape of out text TFIDF vectorizer xcv (11055, 12990)
```

## 5.3 [4.4] Word2Vec

```
In [27]: # Train your own Word2Vec model using your own text corpus
        i=0
        list_of_sentancetrain=[]
        list_of_senancetest=[]
        list_of_senancexcv=[]
        for sentence in X_train:
            list_of_sentancetrain.append(sentence.split())
        for sentence in X_test:
            list_of_senancetest.append(sentence.split())
        for sentence in X_cv:
            list_of_senancexcv.append(sentence.split())

In [28]: # Using Google News Word2Vectors

        # in this project we are using a pretrained model by google
        # its 3.3G file, once you load this into your memory
        # it occupies ~9Gb, so please do this step only if you have >12G of ram

        is_your_ram_gt_16g=False
        want_to_use_google_w2v = False
        want_to_train_w2v = True

        if want_to_train_w2v:
            # min_count = 5 considers only words that occurred atleast 5 times
            w2v_model=Word2Vec(list_of_sentancetrain,min_count=5,size=50, workers=4)
```

```

print(w2v_model.wv.most_similar('great'))
print('='*50)
print(w2v_model.wv.most_similar('worst'))

elif want_to_use_google_w2v and is_your_ram_gt_16g:
    if os.path.isfile('GoogleNews-vectors-negative300.bin'):
        w2v_model=KeyedVectors.load_word2vec_format('GoogleNews-vectors-negative300.b
        print(w2v_model.wv.most_similar('great'))
        print(w2v_model.wv.most_similar('worst'))
    else:
        print("you don't have gogole's word2vec file, keep want_to_train_w2v = True, 1

[('good', 0.8249876499176025), ('fantastic', 0.8009868860244751), ('awesome', 0.78993320465087
=====
[('coolest', 0.7907428741455078), ('eaten', 0.7868819236755371), ('hottest', 0.784995138645172

In [29]: 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 9299

sample words ['bought', 'toy', 'put', 'one', 'dogs', 'crate', 'filled', 'handful', 'kibble',

## 5.4 [4.4.1] Converting text into vectors using Avg W2V, TFIDF-W2V

### [4.4.1.1] Avg W2v

```

In [30]: # average Word2Vec
# compute average word2vec for each review.
sent_vectorsxtrain = []; # the avg-w2v for each sentence/review is stored in this list
for sent in tqdm(list_of_sentancextrain): # for each review/sentence
    sent_vec = np.zeros(50)
    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_vectorsxtrain.append(sent_vec)

100%|| 22445/22445 [00:35<00:00, 626.13it/s]

```

```

In [31]: sent_vectorsxtest = []; # the avg-w2v for each sentence/review is stored in this list
for sent in tqdm(list_of_sentancetest): # for each review/sentence

```

```

sent_vec = np.zeros(50)
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_vectorsxtest.append(sent_vec)
sent_vectorsxcv = []; # the avg-w2v for each sentence/review is stored in this list
for sent in tqdm(list_of_sentancexcv): # for each review/sentence
    sent_vec = np.zeros(50)
    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_vectorsxcv.append(sent_vec)
print(" w2v xtrain ",len(sent_vectorsxtrain)," ",len(sent_vectorsxtrain[0]))
print(" w2v xtest  ",len(sent_vectorsxtest)," ",len(sent_vectorsxtest[0]))
print(" w2v xcv    ",len(sent_vectorsxcv)," ",len(sent_vectorsxcv[0]))

```

```

100%|| 16500/16500 [00:27<00:00, 607.93it/s]
100%|| 11055/11055 [00:18<00:00, 602.54it/s]

```

```

w2v xtrain  22445   50
w2v xtest   16500   50
w2v xcv     11055   50

```

```

In [32]: sent_vectorsxcv = []; # the avg-w2v for each sentence/review is stored in this list
for sent in tqdm(list_of_sentancexcv): # for each review/sentence
    sent_vec = np.zeros(50)
    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_vectorsxcv.append(sent_vec)
print(" w2v xtrain ",len(sent_vectorsxtrain)," ",len(sent_vectorsxtrain[0]))

```

```
print(" w2v xtest ",len(sent_vectorsxtest)," ",len(sent_vectorsxtest[0]))
print(" w2v xcv ",len(sent_vectorsxcv)," ",len(sent_vectorsxcv[0]))
```

100%|| 11055/11055 [00:18<00:00, 593.97it/s]

```
w2v xtrain  22445   50
w2v xtest   16500   50
w2v xcv     11055   50
```

#### [4.4.1.2] TFIDF weighted W2v

```
In [33]: # S = ["abc def pqr", "def def def abc", "pqr pqr def"]
model = TfidfVectorizer()
tfidf_matrix = model.fit_transform(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 [34]: # 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_vectorsxtrain = []; # the tfidf-w2v for each sentence/review is stored in
row=0;
for sent in tqdm(list_of_sentancextrain): # for each review/sentence
    sent_vec = np.zeros(50)
    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 corpus
            # sent.count(word) = tf value 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_vectorsxtrain.append(sent_vec)
    row += 1
```

100%|| 22445/22445 [26:08<00:00, 14.31it/s]

```
In [35]: tfidf_sent_vectorsxtest = []; # the tfidf-w2v for each sentence/review is stored in t
row=0;
for sent in tqdm(list_of_sentancextest): # 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 corpus
# sent.count(word) = tf value 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_vectorsxtest.append(sent_vec)
row += 1

```

100%|| 16500/16500 [18:52<00:00, 14.56it/s]

```

In [36]: tfidf_sent_vectorsxcv = []; # the tfidf-w2v for each sentence/review is stored in this
row=0;
for sent in tqdm(list_of_sentancexcv): # 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 corpus
# sent.count(word) = tf value 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_vectorsxcv.append(sent_vec)
    row += 1

```

100%|| 11055/11055 [13:22<00:00, 13.78it/s]

## 6 Applying Logistic Regression

### 6.1 [5.1] Logistic Regression on BOW, SET 1

#### 6.1.1 [5.1.1] Applying Logistic Regression with L1 regularization on BOW, SET 1

```
In [68]: #code source: http://occam.olin.edu/sites/default/files/DataScienceMaterials/machine_
from sklearn.model_selection import train_test_split
from sklearn.grid_search import GridSearchCV
from sklearn.datasets import *
from sklearn.linear_model import LogisticRegression
from sklearn.preprocessing import StandardScaler

import warnings
warnings.filterwarnings("ignore")

scaler=StandardScaler(with_mean=False)
x_train=scaler.fit_transform(final_countsXtrain)
x_test=scaler.fit_transform(final_countsXtest)
x_cv=scaler.fit_transform(final_countsXcv)
tuned_parameters = [{'C': [0.000001,0.00001,0.0001, 0.001, 0.01, 0.1, 1, 10, 100, 1000]}]

#Using GridSearchCV
model = GridSearchCV(LogisticRegression(penalty='l1'), tuned_parameters, scoring = 'r2')
model.fit(x_train, y_train)

print(model.best_estimator_)
print(model.score(x_test, y_test))

LogisticRegression(C=0.1, class_weight=None, dual=False, fit_intercept=True,
    intercept_scaling=1, max_iter=100, multi_class='ovr', n_jobs=1,
    penalty='l1', random_state=None, solver='liblinear', tol=0.0001,
    verbose=0, warm_start=False)
0.9210901402248803
```

```
In [38]: # Please write all the code with proper documentation
from sklearn.metrics import roc_curve, auc

best_c = 0.1
model = LogisticRegression(penalty='l1' , C=best_c)
model.fit(x_train, y_train)
# roc_auc_score(y_true, y_score) the 2nd parameter should be probability estimates of
# not the predicted outputs

train_fpr, train_tpr, thresholds = roc_curve(y_train, model.predict_proba(x_train)[:,-1])
test_fpr, test_tpr, thresholds = roc_curve(y_test, model.predict_proba(x_test)[:,-1])

plt.plot(train_fpr, train_tpr, label="train AUC =" +str(auc(train_fpr, train_tpr)))
```



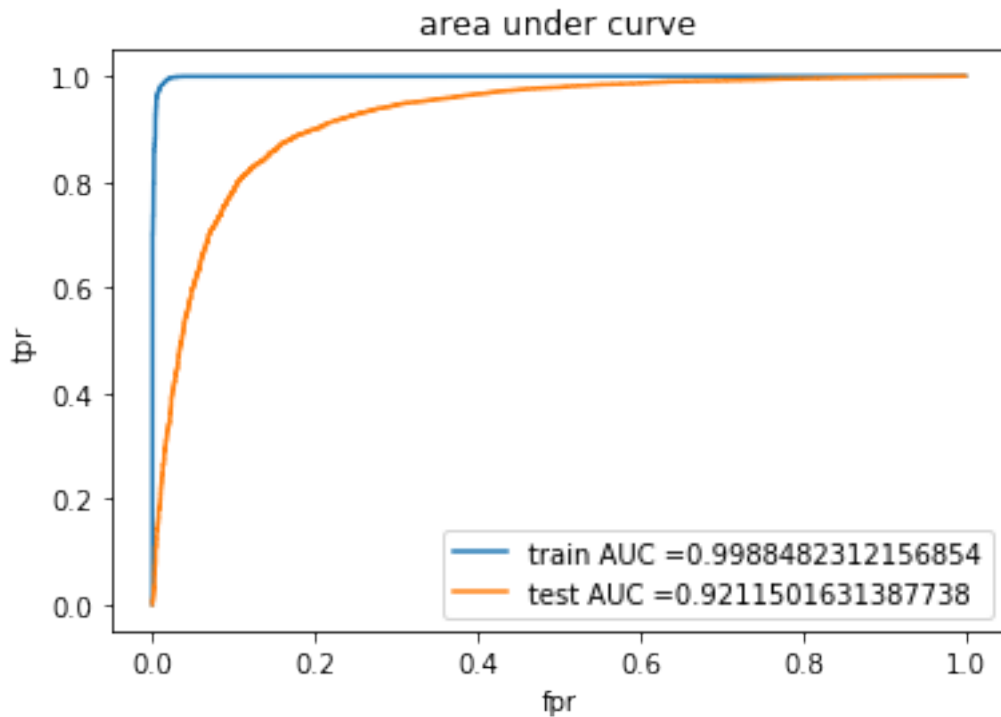
```

plt.plot(test_fpr, test_tpr, label="test AUC =" + str(auc(test_fpr, test_tpr)))
plt.legend()
plt.xlabel("fpr")
plt.ylabel("tpr")
plt.title("area under curve")
plt.show()

print("="*100)

from sklearn.metrics import confusion_matrix
print("Train confusion matrix")
print(confusion_matrix(y_train, model.predict(x_train)))
print("Test confusion matrix")
print(confusion_matrix(y_test, model.predict(x_test)))

```



```

=====
Train confusion matrix
[[ 3252   164]
 [    6 19023]]
Test confusion matrix
[[ 1565   1017]
 [   485 13433]]

```

### [5.1.1.1] Calculating sparsity on weight vector obtained using L1 regularization on BOW, SET 1

In [39]: *# Please write all the code with proper documentation*

```
wt=model.coef_  
z=np.count_nonzero(wt)  
spar=z/np.size(wt)*100  
print("weight vector is ",100-spar,"% sparsity on wt vector using L1 regularization")  
print(wt.shape)
```

```
weight vector is 83.26783090119955 % sparsity on wt vector using L1 regularization  
(1, 29261)
```

### 6.1.2 [5.1.2] Applying Logistic Regression with L2 regularization on BOW, SET 1

In [40]: *# Please write all the code with proper documentation*

```
import warnings  
warnings.filterwarnings("ignore")  
scaler=StandardScaler(with_mean=False)  
x_train=scaler.fit_transform(final_countsXtrain)  
x_test=scaler.fit_transform(final_countsXtest)  
x_cv=scaler.fit_transform(final_countsXcv)  
tuned_parameters = [{'C': [0.000001,0.00001,0.0001, 0.001, 0.01, 0.1, 1, 10, 100, 1000]}]  
  
model = GridSearchCV(LogisticRegression(penalty='l2'), tuned_parameters, scoring = 'r2')  
model.fit(x_train, y_train)  
  
print(model.best_estimator_)  
print(model.score(x_test, y_test))
```

```
LogisticRegression(C=0.001, class_weight=None, dual=False, fit_intercept=True,  
    intercept_scaling=1, max_iter=100, multi_class='ovr', n_jobs=1,  
    penalty='l2', random_state=None, solver='liblinear', tol=0.0001,  
    verbose=0, warm_start=False)  
0.8968042069801555
```

In [41]: best\_c = 0.001

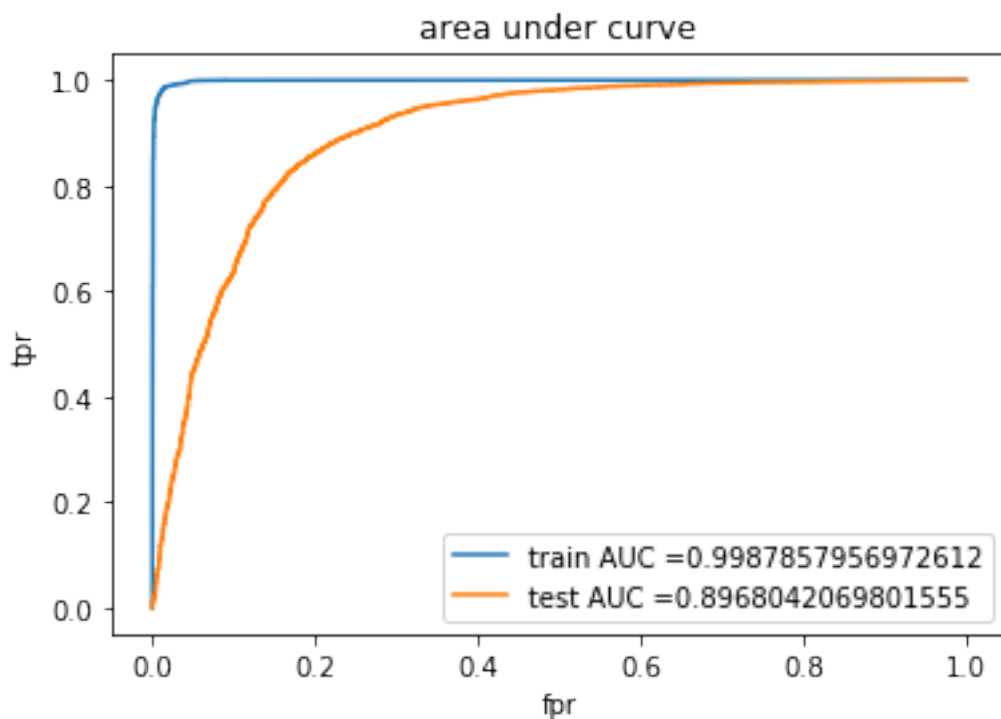
```
model = LogisticRegression(penalty='l2' , C=best_c)  
model.fit(x_train, y_train)  
# roc_auc_score(y_true, y_score) the 2nd parameter should be probability estimates of  
# not the predicted outputs  
wt1=model.coef_  
train_fpr, train_tpr, thresholds = roc_curve(y_train, model.predict_proba(x_train)[:,-1])  
test_fpr, test_tpr, thresholds = roc_curve(y_test, model.predict_proba(x_test)[:,-1])  
  
plt.plot(train_fpr, train_tpr, label="train AUC =" +str(auc(train_fpr, train_tpr)))  
plt.plot(test_fpr, test_tpr, label="test AUC =" +str(auc(test_fpr, test_tpr)))  
plt.legend()
```

```

plt.xlabel("fpr")
plt.ylabel("tpr")
plt.title("area under curve")
plt.show()

print("="*100)
print(x_train.shape)
from sklearn.metrics import confusion_matrix
print("Train confusion matrix")
print(confusion_matrix(y_train, model.predict(x_train)))
print("Test confusion matrix")
print(confusion_matrix(y_test, model.predict(x_test)))

```



```

=====
(22445, 29261)
Train confusion matrix
[[ 3121   295]
 [    7 19022]]
Test confusion matrix
[[ 1231  1351]
 [   216 13702]]

```

### [5.1.2.1] Performing pertubation test (multicollinearity check) on BOW, SET 1

```

In [42]: # Please write all the code with proper documentation
         from scipy.sparse import csr_matrix
         epsilon = np.random.normal(0, 0.01)
         xt=csr_matrix(x_train)
         xt.data=xt.data+epsilon

In [43]: print(xt.shape)

(22445, 29261)

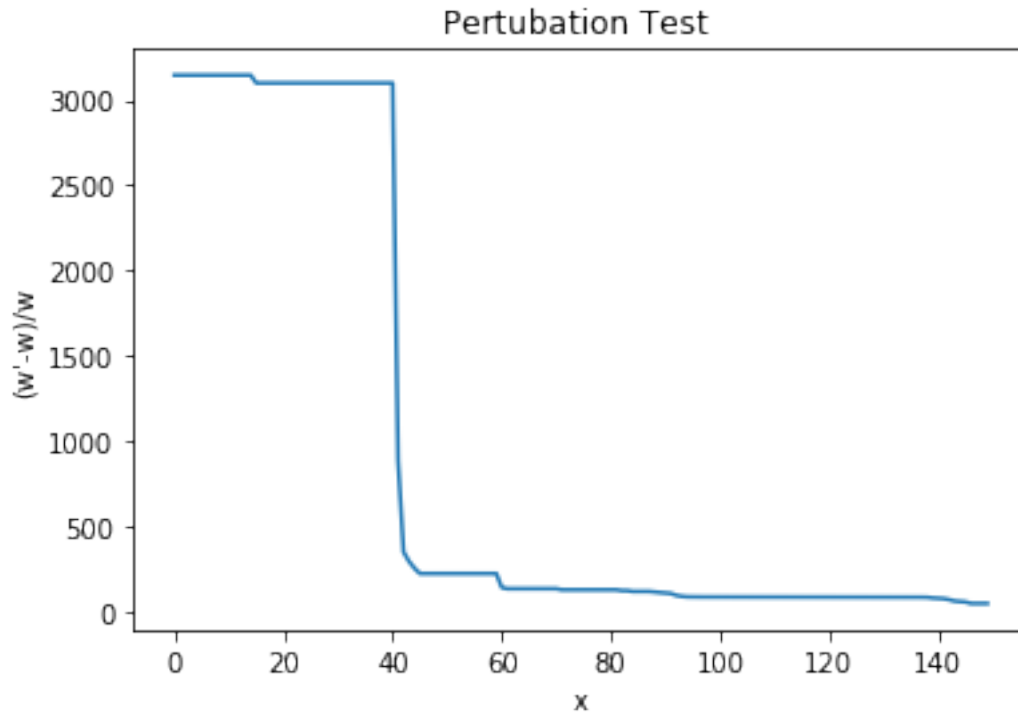
In [44]: best_c = 0.001
         model = LogisticRegression(penalty='l2' , C=best_c)
         model.fit(xt, y_train)
         # roc_auc_score(y_true, y_score) the 2nd parameter should be probability estimates of
         # not the predicted outputs
         wt2=model.coef_

In [ ]:

In [45]: de=np.subtract(wt1,wt2)
         q=np.empty(de.size)
         q.fill(0.000001)
         wt1n=np.add(wt1,q)
         de=np.divide(de,wt1n)
         dey=(np.absolute(de)*100)
         dex=np.arange(0,np.size(dey),1)
         dex=np.reshape(dex,(1,-1))
         dey=np.reshape(dey,(1,-1))
         dc={}
         c=0
         for i in dey[0,:]:
             dc[str(i)]=c
             c=c+1

In [46]: dey=-np.sort(-dey,axis=1)
         #np.reshape(dey,(1,-1))
         plt.plot(dex[0,:150],dey[0,:150])
         plt.title("Pertubation Test ")
         plt.ylabel("(w'-w)/w")
         plt.xlabel("x")
         plt.show()
         print(dey)

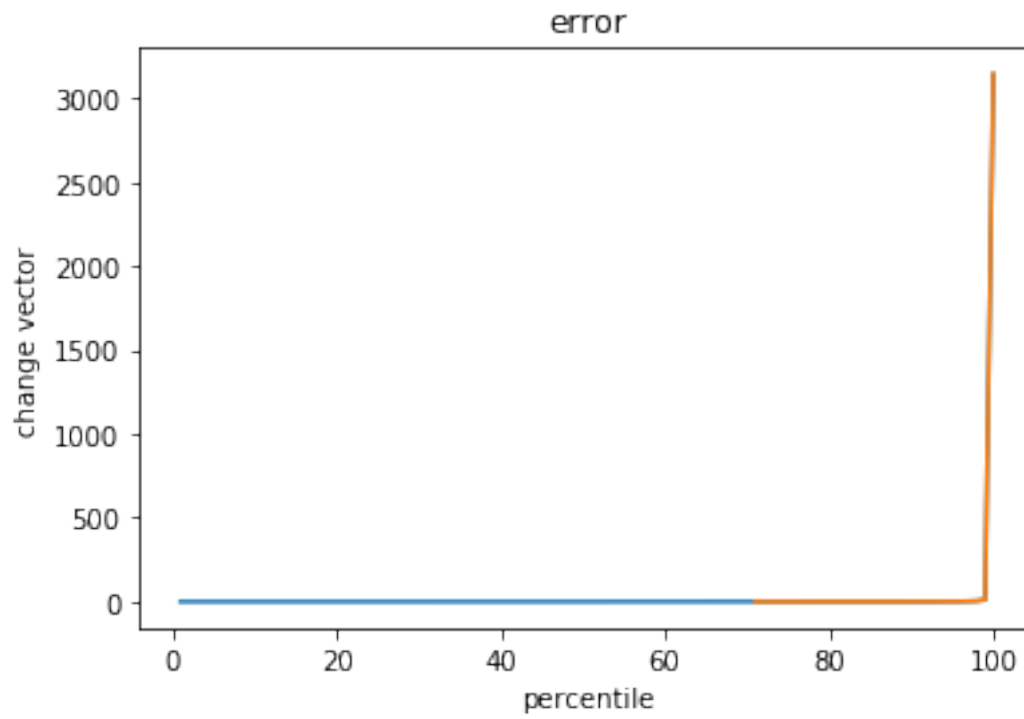
```

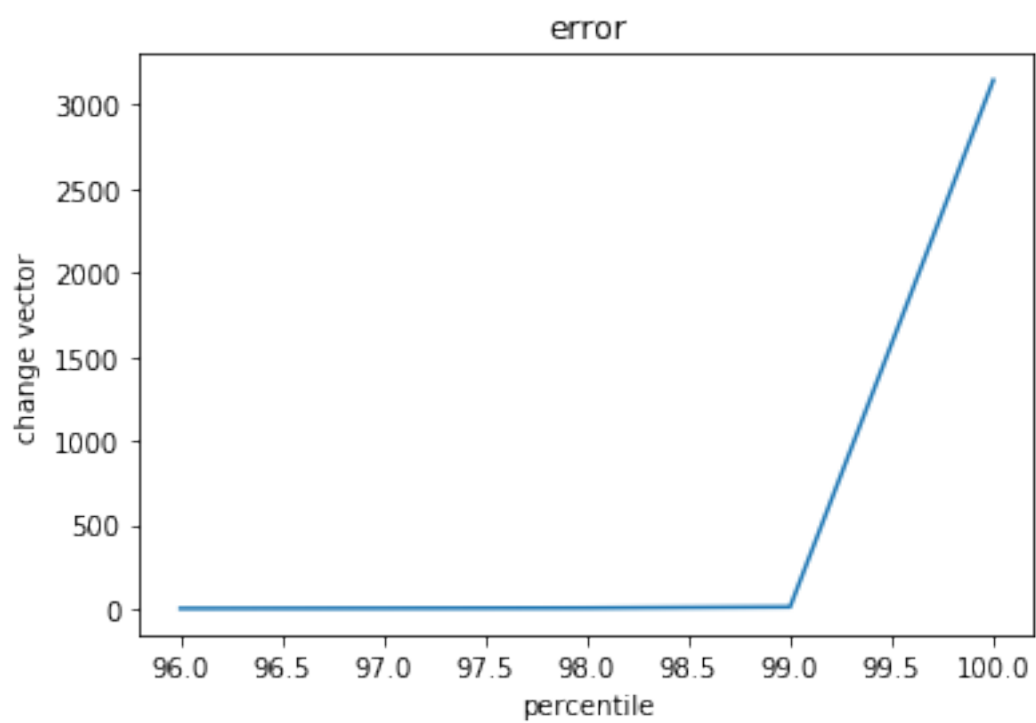
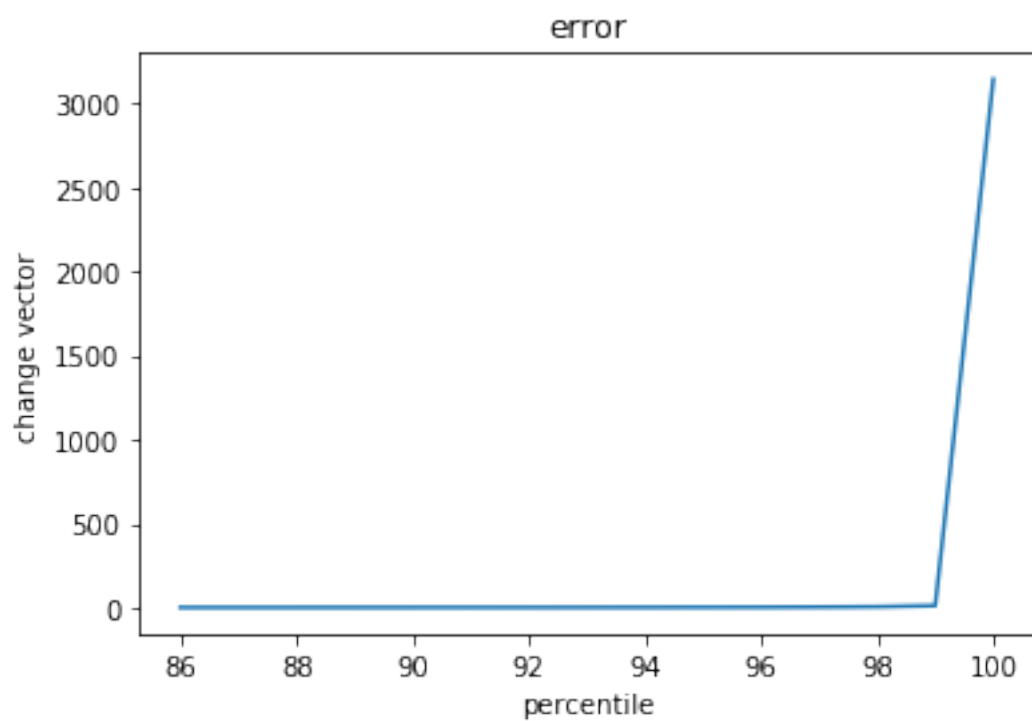


```
[[3.14520257e+03 3.14520257e+03 3.14520257e+03 ... 3.65269630e-05
 2.32428972e-05 6.19215359e-06]]
```

```
In [47]: y=[]
         for i in range(1,101):
             y.append(np.percentile(dey,i))
         y=np.array(y).reshape(-1,1)
         #print(y)
         x=np.arange(1,101,1).reshape(-1,1)
         #print(y.shape)
         #print(x.shape)
         plt.plot(x,y)
         plt.plot(x[70:],y[70:])
         plt.xlabel("percentile")
         plt.ylabel("change vector")
         plt.title("error")
         plt.show()
         plt.plot(x[85:],y[85:])
         plt.xlabel("percentile")
         plt.ylabel("change vector")
         plt.title("error")
         plt.show()
         plt.plot(x[95:],y[95:])
```

```
plt.xlabel("percentile")  
plt.ylabel("change vector")  
plt.title("error")  
plt.show()
```



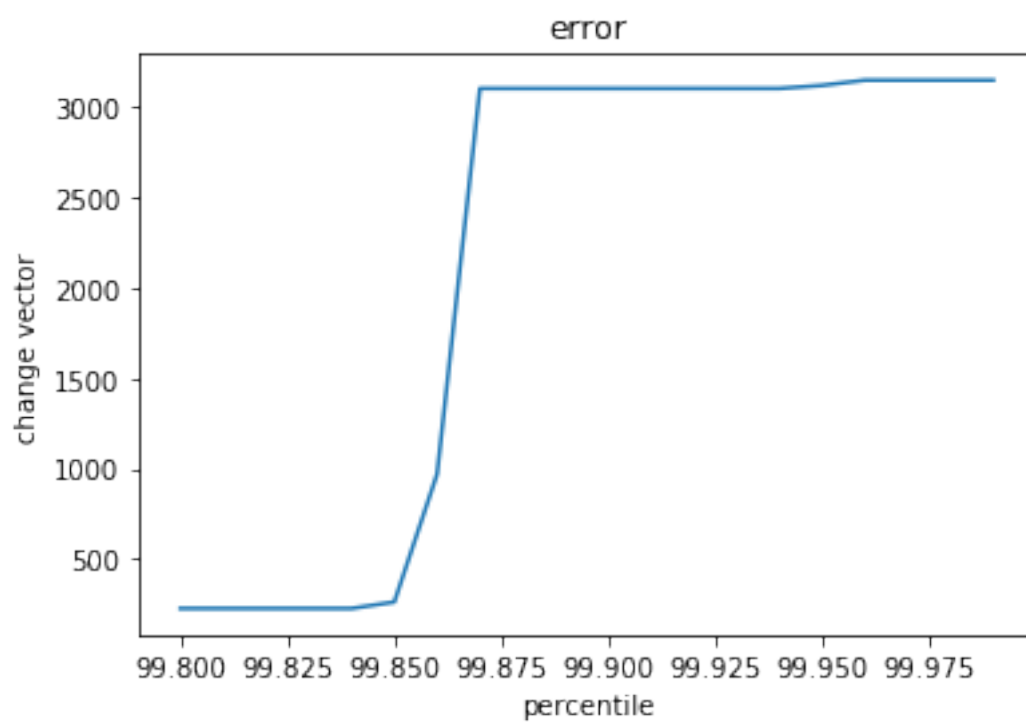
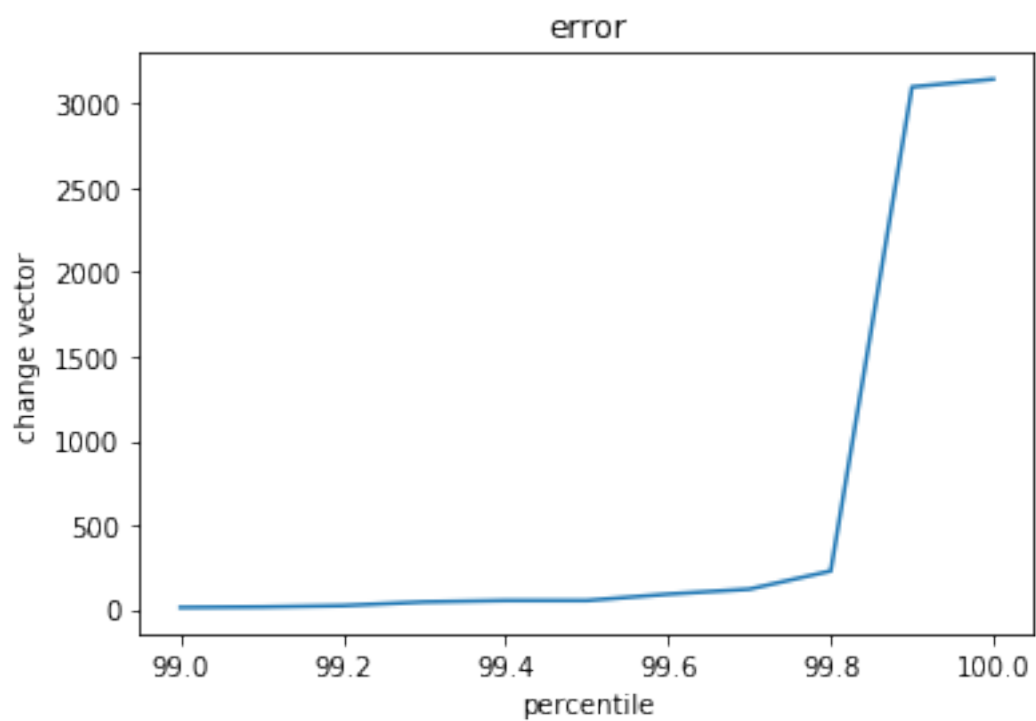


```

In [48]: y=[]
         r=99
         for i in range(11):
             y.append(np.percentile(dey,r))
             r=r+0.1
         y=np.array(y).reshape(-1,1)
         #print(y)
         x=np.arange(99,100.1,0.1).reshape(-1,1)
         #print(y.shape)
         #print(x.shape)
         plt.plot(x,y)
         plt.xlabel("percentile")
         plt.ylabel("change vector")
         plt.title("error")
         plt.show()
         y=[]
         r=99.8
         for i in range(20):
             y.append(np.percentile(dey,r))
             r=r+0.01
         y=np.array(y).reshape(-1,1)
         #print(y)
         x=np.arange(99.8,100,0.01).reshape(-1,1)
         #print(y.shape)
         #print(x.shape)
         plt.plot(x[:20],y)
         plt.xlabel("percentile")
         plt.ylabel("change vector")
         plt.title("error")
         plt.show()

```





```
In [67]: print("The features having collinearity are")
          t=np.percentile(dey,99.850)
          for i in dey[0,:]:
              if i>=t:
                  print(count_vect.get_feature_names()[dc[str(i)]])
```

[illegible]

26

xvi  
xvi  
forward  
monarch  
show

## 7 [5.1.3] Feature Importance on BOW, SET 1

### [5.1.3.1] Top 10 important features of positive class from SET 1

```
In [50]: # Please write all the code with proper documentation
         #wtvec=np.array(wt1)
         d={}
         c=0
         for i in wt1[0,:]:
             d[str(i)]=c
             c=c+1
         wtvec=-np.sort(-wt1)
         for i in range(10):
             print(count_vect.get_feature_names()[d[str(wtvec[0,i])]))
```

great  
love  
best  
good  
loves  
delicious  
excellent  
favorite  
perfect  
nice

### [5.1.3.2] Top 10 important features of negative class from SET 1

```
In [51]: # Please write all the code with proper documentation
         d={}
         c=0
         for i in wt1[0,:]:
             d[str(i)]=c
             c=c+1
         wtvec=np.sort(wt1)
         for i in range(10):
             print(count_vect.get_feature_names()[d[str(wtvec[0,i])]))
```

disappointed  
not

```
worst
awful
waste
horrible
terrible
return
bad
disappointment
```

## 7.1 [5.2] Logistic Regression on TFIDF, SET 2

### 7.1.1 [5.2.1] Applying Logistic Regression with L1 regularization on TFIDF, SET 2

```
In [52]: # Please write all the code with proper documentation
scaler=StandardScaler(with_mean=False)
x_train=scaler.fit_transform(final_tf_idfXtrain)
x_test=scaler.fit_transform(final_tf_idfXtest)
x_cv=scaler.fit_transform(final_tf_idfXcv)
tuned_parameters = [{'C': [0.000001,0.00001,0.0001, 0.001, 0.01, 0.1, 1, 10, 100, 1000]}]

#Using GridSearchCV
model = GridSearchCV(LogisticRegression(penalty='l1'), tuned_parameters, scoring = 'r2')
model.fit(x_train, y_train)

print(model.best_estimator_)
print(model.score(x_test, y_test))

LogisticRegression(C=0.01, class_weight=None, dual=False, fit_intercept=True,
    intercept_scaling=1, max_iter=100, multi_class='ovr', n_jobs=1,
    penalty='l1', random_state=None, solver='liblinear', tol=0.0001,
    verbose=0, warm_start=False)
0.9410177198104779

In [53]: best_c = 0.01
model = LogisticRegression(penalty='l1' , C=best_c)
model.fit(x_train, y_train)
# roc_auc_score(y_true, y_score) the 2nd parameter should be probability estimates of
# not the predicted outputs

train_fpr, train_tpr, thresholds = roc_curve(y_train, model.predict_proba(x_train)[:,-1])
test_fpr, test_tpr, thresholds = roc_curve(y_test, model.predict_proba(x_test)[:,-1])

plt.plot(train_fpr, train_tpr, label="train AUC =" +str(auc(train_fpr, train_tpr)))
plt.plot(test_fpr, test_tpr, label="test AUC =" +str(auc(test_fpr, test_tpr)))
plt.legend()
plt.xlabel("fpr")
plt.ylabel("tpr")
```

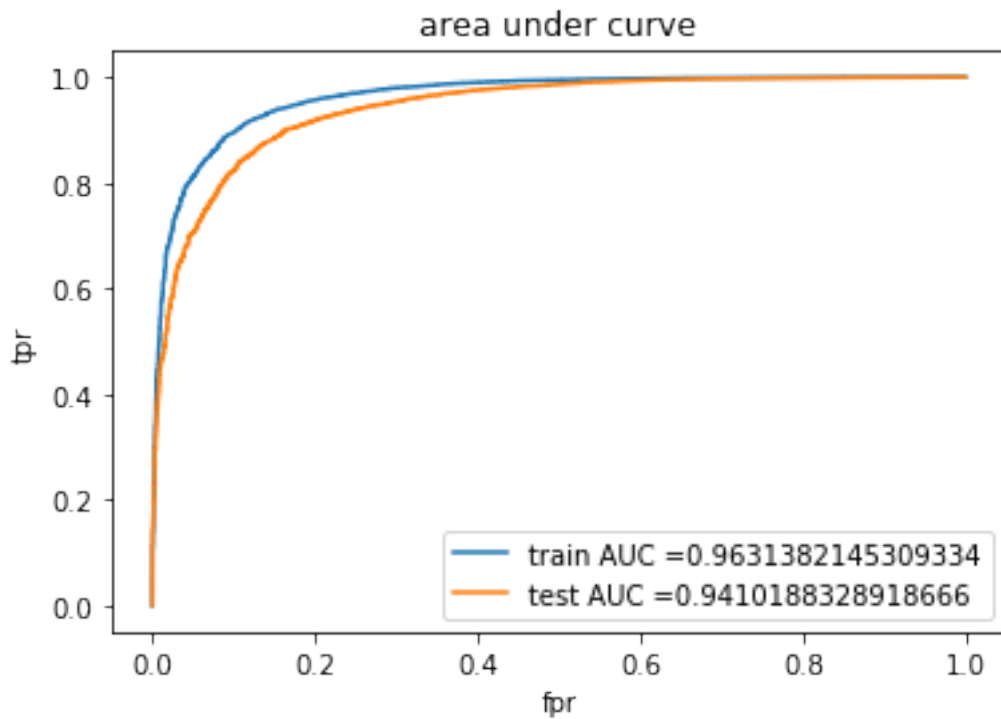
```

plt.title("area under curve")
plt.show()

print("="*100)

from sklearn.metrics import confusion_matrix
print("Train confusion matrix")
print(confusion_matrix(y_train, model.predict(x_train)))
print("Test confusion matrix")
print(confusion_matrix(y_test, model.predict(x_test)))

```



```

=====
Train confusion matrix
[[ 1940  1476]
 [  151 18878]]
Test confusion matrix
[[ 1288  1294]
 [  183 13735]]

```

### 7.1.2 [5.2.2] Applying Logistic Regression with L2 regularization on TFIDE, SET 2

```

In [54]: # Please write all the code with proper documentation
scaler=StandardScaler(with_mean=False)

```

```

x_train=scaler.fit_transform(final_tf_idfXtrain)
x_test=scaler.fit_transform(final_tf_idfXtest)
x_cv=scaler.fit_transform(final_tf_idfXcv)
tuned_parameters = [{'C': [0.000001,0.00001,0.0001, 0.001, 0.01, 0.1, 1, 10, 100, 1000]}]

#Using GridSearchCV
model = GridSearchCV(LogisticRegression(penalty='l2'), tuned_parameters, scoring = 'r
model.fit(x_train, y_train)

print(model.best_estimator_)
print(model.score(x_test, y_test))

LogisticRegression(C=0.001, class_weight=None, dual=False, fit_intercept=True,
    intercept_scaling=1, max_iter=100, multi_class='ovr', n_jobs=1,
    penalty='l2', random_state=None, solver='liblinear', tol=0.0001,
    verbose=0, warm_start=False)
0.9338242921998929

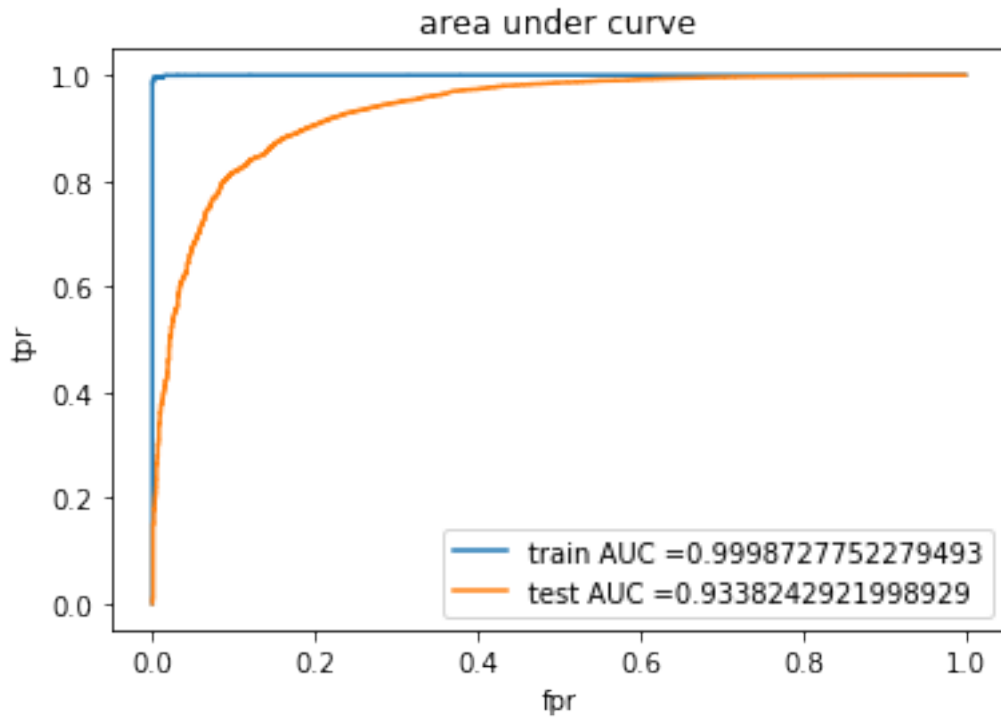
In [55]: best_c = 0.001
model = LogisticRegression(penalty='l2' , C=best_c)
model.fit(x_train, y_train)
# roc_auc_score(y_true, y_score) the 2nd parameter should be probability estimates of
# not the predicted outputs
wt1=model.coef_
train_fpr, train_tpr, thresholds = roc_curve(y_train, model.predict_proba(x_train)[:,1])
test_fpr, test_tpr, thresholds = roc_curve(y_test, model.predict_proba(x_test)[:,1])

plt.plot(train_fpr, train_tpr, label="train AUC =" +str(auc(train_fpr, train_tpr)))
plt.plot(test_fpr, test_tpr, label="test AUC =" +str(auc(test_fpr, test_tpr)))
plt.legend()
plt.xlabel("fpr")
plt.ylabel("tpr")
plt.title("area under curve")
plt.show()

print("="*100)

from sklearn.metrics import confusion_matrix
print("Train confusion matrix")
print(confusion_matrix(y_train, model.predict(x_train)))
print("Test confusion matrix")
print(confusion_matrix(y_test, model.predict(x_test)))

```



=====

Train confusion matrix

```
[[ 3331   85]
 [    9 19020]]
```

Test confusion matrix

```
[[ 1543  1039]
 [   366 13552]]
```

### 7.1.3 [5.2.3] Feature Importance on TFIDF, SET 2

#### [5.2.3.1] Top 10 important features of positive class from SET 2

```
In [56]: # Please write all the code with proper documentation
d={}
c=0
for i in wt1[0,:]:
    d[str(i)]=c
    c=c+1
wtvec=-np.sort(-wt1)
for i in range(10):
    print(tf_idf_vect.get_feature_names()[d[str(wtvec[0,i])]])
```

great  
love

best  
good  
delicious  
perfect  
loves  
excellent  
nice  
favorite

### [5.2.3.2] Top 10 important features of negative class from SET 2

```
In [57]: # Please write all the code with proper documentation
d={}
c=0
for i in wt1[0,:]:
    d[str(i)]=c
    c=c+1
wtvec=np.sort(wt1)
for i in range(10):
    print(tf_idf_vect.get_feature_names()[d[str(wtvec[0,i])])])
```

disappointed  
not buy  
worst  
disappointment  
not worth  
two stars  
awful  
not recommend  
terrible  
return

## 7.2 [5.3] Logistic Regression on AVG W2V, SET 3

### 7.2.1 [5.3.1] Applying Logistic Regression with L1 regularization on AVG W2V SET 3

```
In [58]: # Please write all the code with proper documentation
scaler=StandardScaler(with_mean=False)
x_train=scaler.fit_transform(sent_vectorsxtrain)
x_test=scaler.fit_transform(sent_vectorsxtest)
x_cv=scaler.fit_transform(sent_vectorsxcv)
tuned_parameters = [{'C': [0.000001,0.00001,0.0001, 0.001, 0.01, 0.1, 1, 10, 100, 1000]}]

#Using GridSearchCV
model = GridSearchCV(LogisticRegression(penalty='l1'), tuned_parameters, scoring = 'r2')
model.fit(x_train, y_train)
```



```

print(model.best_estimator_)
print(model.score(x_test, y_test))

LogisticRegression(C=1, class_weight=None, dual=False, fit_intercept=True,
    intercept_scaling=1, max_iter=100, multi_class='ovr', n_jobs=1,
    penalty='l1', random_state=None, solver='liblinear', tol=0.0001,
    verbose=0, warm_start=False)
0.8851669688868151

```

```

In [59]: best_c = 1
model = LogisticRegression(penalty='l1' , C=best_c)
model.fit(x_train, y_train)
# roc_auc_score(y_true, y_score) the 2nd parameter should be probability estimates of
# not the predicted outputs

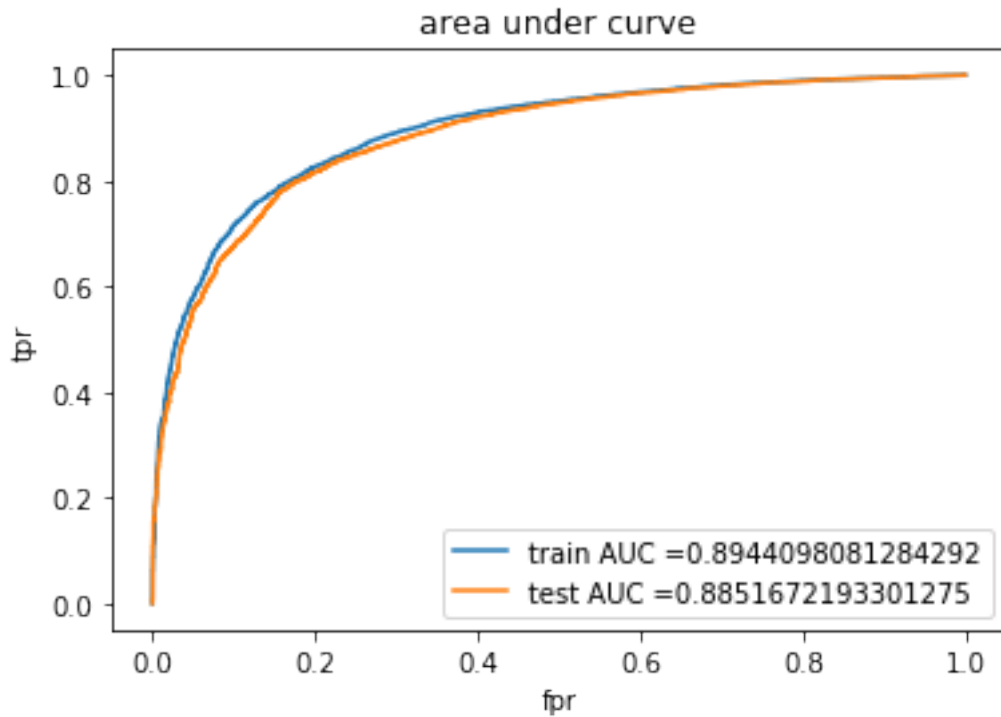
train_fpr, train_tpr, thresholds = roc_curve(y_train, model.predict_proba(x_train)[: , 1])
test_fpr, test_tpr, thresholds = roc_curve(y_test, model.predict_proba(x_test)[: , 1])

plt.plot(train_fpr, train_tpr, label="train AUC =" + str(auc(train_fpr, train_tpr)))
plt.plot(test_fpr, test_tpr, label="test AUC =" + str(auc(test_fpr, test_tpr)))
plt.legend()
plt.xlabel("fpr")
plt.ylabel("tpr")
plt.title("area under curve")
plt.show()

print("="*100)

from sklearn.metrics import confusion_matrix
print("Train confusion matrix")
print(confusion_matrix(y_train, model.predict(x_train)))
print("Test confusion matrix")
print(confusion_matrix(y_test, model.predict(x_test)))

```



=====

Train confusion matrix

```
[[ 1404  2012]
 [   634 18395]]
```

Test confusion matrix

```
[[ 1077  1505]
 [   501 13417]]
```

## 7.2.2 [5.3.2] Applying Logistic Regression with L2 regularization on AVG W2V, SET 3

In [60]: *# Please write all the code with proper documentation*

*#Using GridSearchCV*

```
model = GridSearchCV(LogisticRegression(penalty='l2'), tuned_parameters, scoring = 'r
model.fit(x_train, y_train)
```

```
print(model.best_estimator_)
print(model.score(x_test, y_test))
```

```
LogisticRegression(C=1, class_weight=None, dual=False, fit_intercept=True,
    intercept_scaling=1, max_iter=100, multi_class='ovr', n_jobs=1,
    penalty='l2', random_state=None, solver='liblinear', tol=0.0001,
    verbose=0, warm_start=False)
```

0.8851956307325779

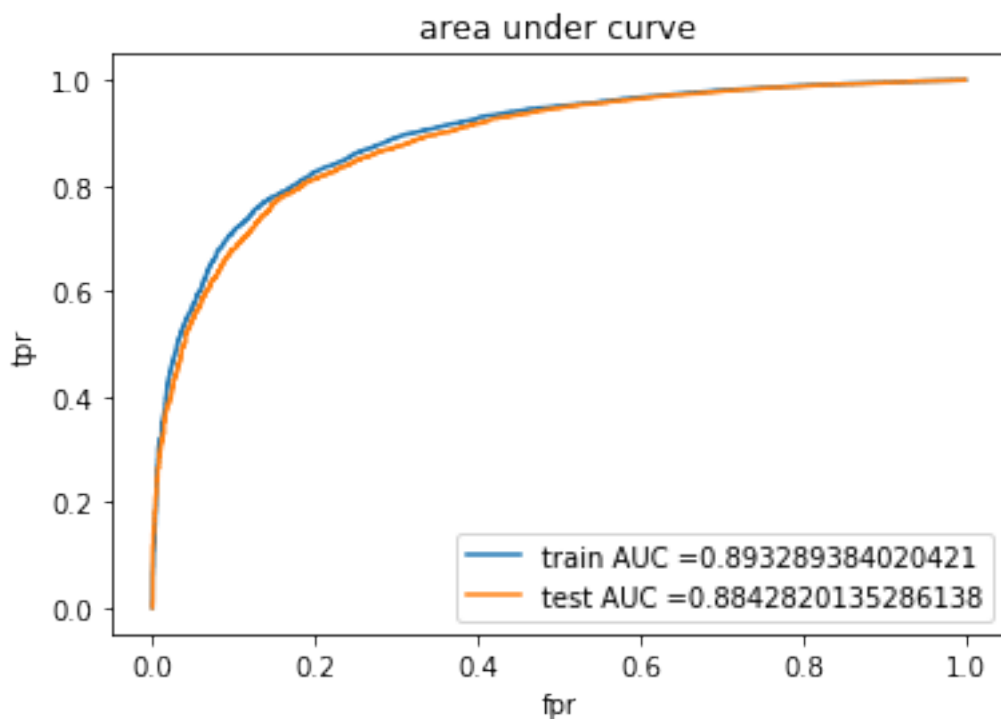
```
In [61]: best_c = 0.1
model = LogisticRegression(penalty='l2' , C=best_c)
model.fit(x_train, y_train)
# roc_auc_score(y_true, y_score) the 2nd parameter should be probability estimates of
# not the predicted outputs

train_fpr, train_tpr, thresholds = roc_curve(y_train, model.predict_proba(x_train)[: ,1])
test_fpr, test_tpr, thresholds = roc_curve(y_test, model.predict_proba(x_test)[: ,1])

plt.plot(train_fpr, train_tpr, label="train AUC =" +str(auc(train_fpr, train_tpr)))
plt.plot(test_fpr, test_tpr, label="test AUC =" +str(auc(test_fpr, test_tpr)))
plt.legend()
plt.xlabel("fpr")
plt.ylabel("tpr")
plt.title("area under curve")
plt.show()

print("="*100)

from sklearn.metrics import confusion_matrix
print("Train confusion matrix")
print(confusion_matrix(y_train, model.predict(x_train)))
print("Test confusion matrix")
print(confusion_matrix(y_test, model.predict(x_test)))
```



```
=====
Train confusion matrix
[[ 1373  2043]
 [   626 18403]]
Test confusion matrix
[[ 1009  1573]
 [   471 13447]]
```

## 7.3 [5.4] Logistic Regression on TFIDF W2V, SET 4

### 7.3.1 [5.4.1] Applying Logistic Regression with L1 regularization on TFIDF W2V, SET 4

```
In [62]: # Please write all the code with proper documentation
scaler=StandardScaler(with_mean=False)
x_train=scaler.fit_transform(tfidf_sent_vectorsxtrain)
x_test=scaler.fit_transform(tfidf_sent_vectorsxtest)
x_cv=scaler.fit_transform(tfidf_sent_vectorsxcv)
tuned_parameters = [{'C': [0.000001,0.00001,0.0001, 0.001, 0.01, 0.1, 1, 10, 100, 1000]}]

#Using GridSearchCV
model = GridSearchCV(LogisticRegression(penalty='l1'), tuned_parameters, scoring = 'roc_auc')
model.fit(x_train, y_train)

print(model.best_estimator_)
```

```
LogisticRegression(C=10000, class_weight=None, dual=False, fit_intercept=True,
    intercept_scaling=1, max_iter=100, multi_class='ovr', n_jobs=1,
    penalty='l1', random_state=None, solver='liblinear', tol=0.0001,
    verbose=0, warm_start=False)
```

```
In [63]: best_c = 1
model = LogisticRegression(penalty='l1' , C=best_c)
model.fit(x_train, y_train)
# roc_auc_score(y_true, y_score) the 2nd parameter should be probability estimates of
# not the predicted outputs

train_fpr, train_tpr, thresholds = roc_curve(y_train, model.predict_proba(x_train)[:,-1])
test_fpr, test_tpr, thresholds = roc_curve(y_test, model.predict_proba(x_test)[:,-1])

plt.plot(train_fpr, train_tpr, label="train AUC =" +str(auc(train_fpr, train_tpr)))
plt.plot(test_fpr, test_tpr, label="test AUC =" +str(auc(test_fpr, test_tpr)))
plt.legend()
plt.xlabel("fpr")
plt.ylabel("tpr")
```

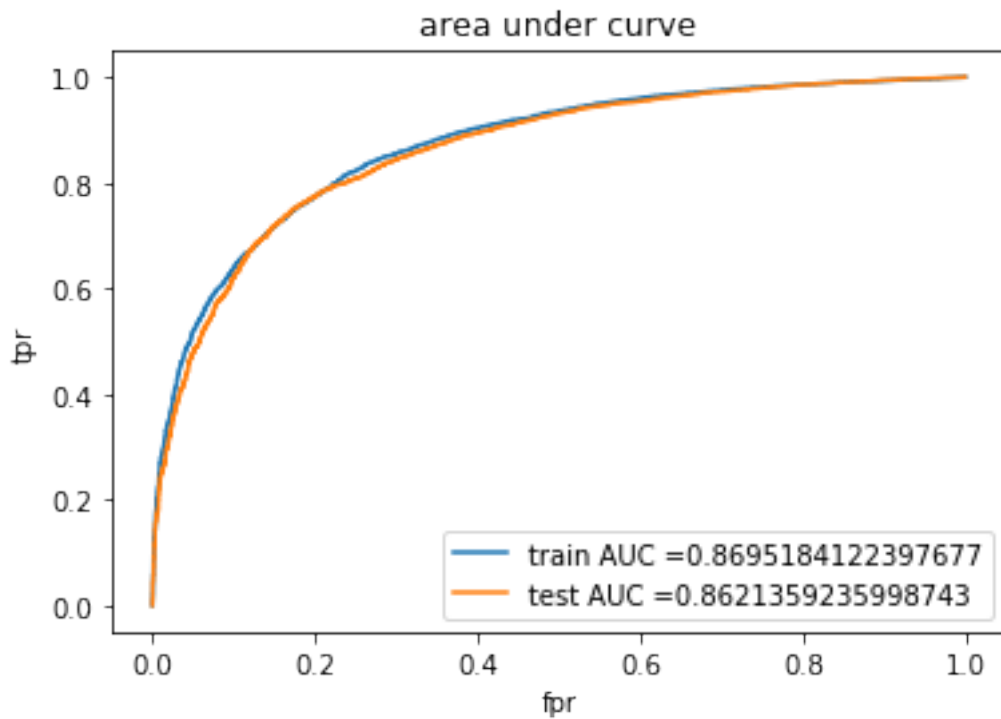
```

plt.title("area under curve")
plt.show()

print("="*100)

from sklearn.metrics import confusion_matrix
print("Train confusion matrix")
print(confusion_matrix(y_train, model.predict(x_train)))
print("Test confusion matrix")
print(confusion_matrix(y_test, model.predict(x_test)))

```



```

=====
Train confusion matrix
[[ 1120  2296]
 [  538 18491]]
Test confusion matrix
[[  845  1737]
 [  439 13479]]

```

### 7.3.2 [5.4.2] Applying Logistic Regression with L2 regularization on TFIDF W2V, SET 4

```

In [64]: # Please write all the code with proper documentation
         model = GridSearchCV(LogisticRegression(penalty='l2'), tuned_parameters, scoring = 'r

```

```

model.fit(x_train, y_train)

print(model.best_estimator_)
print(model.score(x_test, y_test))

LogisticRegression(C=10, class_weight=None, dual=False, fit_intercept=True,
                    intercept_scaling=1, max_iter=100, multi_class='ovr', n_jobs=1,
                    penalty='l2', random_state=None, solver='liblinear', tol=0.0001,
                    verbose=0, warm_start=False)
0.8621971987303303

In [65]: best_c = 0.1
model = LogisticRegression(penalty='l2' , C=best_c)
model.fit(x_train, y_train)
# roc_auc_score(y_true, y_score) the 2nd parameter should be probability estimates of
# not the predicted outputs

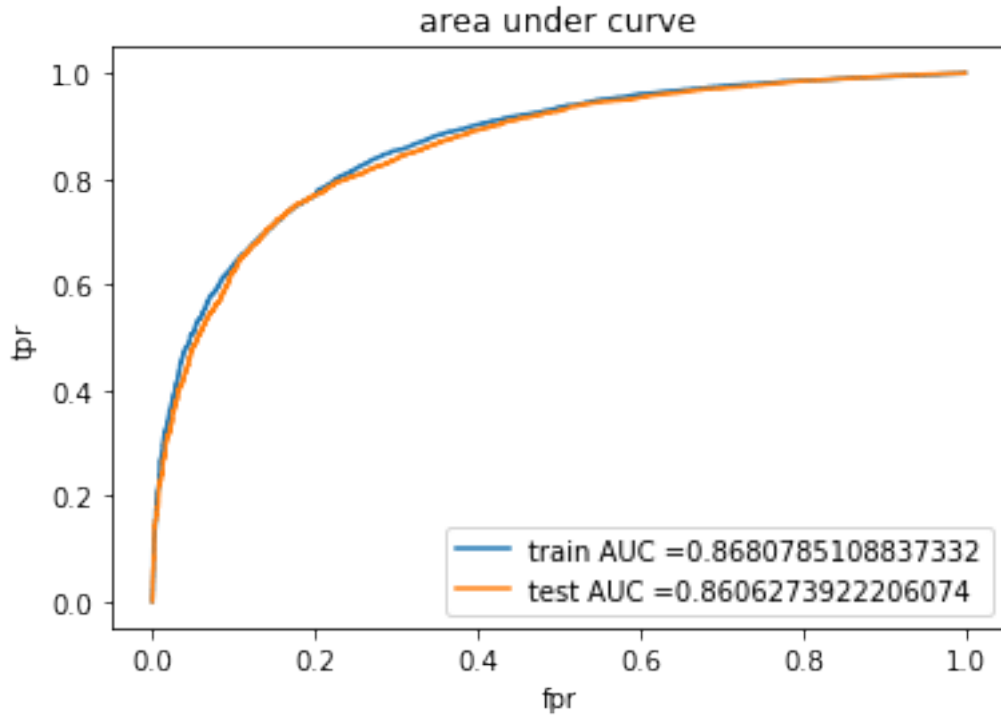
train_fpr, train_tpr, thresholds = roc_curve(y_train, model.predict_proba(x_train)[:,1])
test_fpr, test_tpr, thresholds = roc_curve(y_test, model.predict_proba(x_test)[:,1])

plt.plot(train_fpr, train_tpr, label="train AUC =" + str(auc(train_fpr, train_tpr)))
plt.plot(test_fpr, test_tpr, label="test AUC =" + str(auc(test_fpr, test_tpr)))
plt.legend()
plt.xlabel("fpr")
plt.ylabel("tpr")
plt.title("area under curve")
plt.show()

print("="*100)

from sklearn.metrics import confusion_matrix
print("Train confusion matrix")
print(confusion_matrix(y_train, model.predict(x_train)))
print("Test confusion matrix")
print(confusion_matrix(y_test, model.predict(x_test)))

```



=====

Train confusion matrix

```
[[ 1092  2324]
 [   529 18500]]
```

Test confusion matrix

```
[[   775   1807]
 [   401 13517]]
```

## 8 [6] Conclusions

In [66]: `from prettytable import PrettyTable`

```
x = PrettyTable()
```

```
x.field_names = ["Vectorizer", "Model", "Hyperparameter - k", "AUC"]
```

```
x.add_row(["BOW", "Logistic Regression with L1 regularization", 0.1,0.917])
```

```
x.add_row(["TFIDF", "Logistic Regression with L1 regularization", 0.001,0.9007])
```

```
x.add_row(["Avg- W2Vec", "Logistic Regression with L1 regularization", 0.01,0.943])
```

```
x.add_row(["TFIDF- W2Vec", "Logistic Regression with L1 regularization", 0.001,0.939])
```

```
x.add_row(["BOW", "Logistic Regression with L2 regularization", 1,0.890])
```

```
x.add_row(["TFIDF", "Logistic Regression with L2 regularization", 0.1,0.890])
```

```

x.add_row(["Avg- W2Vec", "Logistic Regression with L2 regularization", 1,0.867])
x.add_row(["TFIDF- W2Vec", "Logistic Regression with L2 regularization", 0.1,0.867])

print(x)

```

Vectorizer	Model	Hyperparameter - k	AUC
BOW	Logistic Regression with L1 regularization	0.1	0.917
TFIDF	Logistic Regression with L1 regularization	0.001	0.9007
Avg- W2Vec	Logistic Regression with L1 regularization	0.01	0.943
TFIDF- W2Vec	Logistic Regression with L1 regularization	0.001	0.939
BOW	Logistic Regression with L2 regularization	1	0.89
TFIDF	Logistic Regression with L2 regularization	0.1	0.89
Avg- W2Vec	Logistic Regression with L2 regularization	1	0.867
TFIDF- W2Vec	Logistic Regression with L2 regularization	0.1	0.867