svm_et

February 17, 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 - 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).

[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 cosnidered as a positive review. A rating of 1 or 2 can be considered as negative one. A review of rating 3 is considered nuetral 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 wil be set to "positive". Otherwise, it will be set to "negative".

```
In [0]: %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 [57]: import pandas as pd
         downloaded = drive.CreateFile({'id':id})
         downloaded.GetContentFile('mydata.csv')
         df3 = pd.read_csv('mydata.csv')
```

```
WARNING:googleapiclient.discovery_cache:file_cache is unavailable when using oauth2client >= 4
Traceback (most recent call last):
  File "/usr/local/lib/python3.6/dist-packages/googleapiclient/discovery_cache/__init__.py", l
    from google.appengine.api import memcache
```

```
ModuleNotFoundError: No module named 'google.appengine'

During handling of the above exception, another exception occurred:

Traceback (most recent call last):
  File "/usr/local/lib/python3.6/dist-packages/googleapiclient/discovery_cache/file_cache.py",
    from oauth2client.contrib.locked_file import LockedFile
ModuleNotFoundError: No module named 'oauth2client.contrib.locked_file'

During handling of the above exception, another exception occurred:

Traceback (most recent call last):
  File "/usr/local/lib/python3.6/dist-packages/googleapiclient/discovery_cache/file_cache.py",
    from oauth2client.locked_file import LockedFile
ModuleNotFoundError: No module named 'oauth2client.locked_file'

During handling of the above exception, another exception occurred:

Traceback (most recent call last):
  File "/usr/local/lib/python3.6/dist-packages/googleapiclient/discovery_cache/__init__.py", li
    from . import file_cache
  File "/usr/local/lib/python3.6/dist-packages/googleapiclient/discovery_cache/file_cache.py",
    'file_cache is unavailable when using oauth2client >= 4.0.0')
ImportError: file_cache is unavailable when using oauth2client >= 4.0.0
```

```
In [58]: df3.shape
```

```
Out[58]: (100000, 11)
```

```
In [0]: # 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 point
        # 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 5
        # for tsne assignment you can take 5k data points

        filtered_data = pd.read_sql_query(""" SELECT * FROM Reviews WHERE Score != 3 LIMIT 5000

        # Give reviews with Score>3 a positive rating(1), and reviews with a score<3 a negativ
        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)
```

```
In [0]: filtered_data=df3
```

```
In [0]: display = pd.read_sql_query("""
        SELECT UserId, ProductId, ProfileName, Time, Score, Text, COUNT(*)
        FROM Reviews
        GROUP BY UserId
        HAVING COUNT(*)>1
        """, con)
```

```
In [0]: print(display.shape)
        display.head()
```

```
In [0]: display[display['UserId']=='AZY10LLTJ71NX']
```

```
In [0]: display['COUNT(*)'].sum()
```

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

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

```
In [61]: #Deduplication of entries
         final=sorted_data.drop_duplicates(subset={"UserId","ProfileName","Time","Text"}, keep=
         final.shape
```

```
Out[61]: (87775, 11)
```

```
In [62]: #Checking to see how much % of data still remains
         (final['Id'].size*1.0)/(filtered_data['Id'].size*1.0)*100
```

```
Out[62]: 87.775
```

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 calcualtions

```
In [0]: display= pd.read_sql_query("""
        SELECT *
        FROM Reviews
        WHERE Score != 3 AND Id=44737 OR Id=64422
        ORDER BY ProductID
        """, con)

        display.head()
```

```
In [0]: final=final[final.HelpfulnessNumerator<=final.HelpfulnessDenominator]
```

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

```
(87773, 11)
```

```
Out[15]: 5    59521
         4    14071
         1     8886
         2     5295
         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 obsereved to be better than Porter Stemming)

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

```
In [64]: # 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)
```

```
My dogs loves this chicken but its a product from China, so we wont be buying it anymore.  Its
==================================================
The Candy Blocks were a nice visual for the Lego Birthday party but the candy has little taste
==================================================
was way to hot for my blood, took a bite and did a jig  lol
==================================================
My dog LOVES these treats. They tend to have a very strong fish oil smell. So if you are afraid
==================================================
```

```
In [65]: # 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)
```

```
My dogs loves this chicken but its a product from China, so we wont be buying it anymore.  Its
```

```
In [66]: # 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)
```

```
My dogs loves this chicken but its a product from China, so we wont be buying it anymore.  Its
==================================================
The Candy Blocks were a nice visual for the Lego Birthday party but the candy has little taste
==================================================
was way to hot for my blood, took a bite and did a jig  lol
==================================================
My dog LOVES these treats. They tend to have a very strong fish oil smell. So if you are afrai
```

```
In [0]: # 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 [68]: sent_1500 = decontracted(sent_1500)
         print(sent_1500)
         print("="*50)
```

```
was way to hot for my blood, took a bite and did a jig  lol
==================================================
```

```
In [69]: #remove words with numbers python: https://stackoverflow.com/a/18082370/4084039
         sent_0 = re.sub("\S*\d\S*", "", sent_0).strip()
         print(sent_0)
```

```
My dogs loves this chicken but its a product from China, so we wont be buying it anymore.  Its
```

```
In [70]: #remove spacial character: https://stackoverflow.com/a/5843547/4084039
         sent_1500 = re.sub('[^A-Za-z0-9]+', ' ', sent_1500)
         print(sent_1500)
```

```
was way to hot for my blood took a bite and did a jig lol
```

```
In [0]: # 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', 'ourselves
                    "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', 'ha
                    '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', 'migh
                   "mustn't", 'needn', "needn't", 'shan', "shan't", 'shouldn', "shouldn't", 'w
                   'won', "won't", 'wouldn', "wouldn't"])
```

```
In [72]: # 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%|| 87773/87773 [00:33<00:00, 2628.36it/s]
```

```
In [73]: preprocessed_reviews[1500]
```

```
Out[73]: 'way hot blood took bite jig lol'
```

[3.2] Preprocessing Review Summary

```
In [0]: def func(x):
          if x>3:
            return 1
          else:
            return 0
```

```
In [0]: x=preprocessed_reviews
        y=final['Score'].apply(func)
```

```
In [0]: from sklearn.model_selection import train_test_split
        x1,xtest,y1,ytest=train_test_split(x,y,test_size=0.3,random_state=1)
```

```
In [0]: xtrain,xcv,ytrain,ycv=train_test_split(x1,y1,test_size=0.2,random_state=1)
```

```
In [78]: print(len(xtrain))
         print(ytrain.shape)
         print(len(xtest))
         print(ytest.shape)
         print(len(xcv))
         print(ycv.shape)
```

```
49152
(49152,)
26332
(26332,)
12289
(12289,)
```

5 [4] Featurization

5.1 [4.1] BAG OF WORDS

```
In [79]: from sklearn.feature_extraction.text import CountVectorizer
         count_vect=CountVectorizer()
         xtrainonehotencoding=count_vect.fit_transform(xtrain)
         xtestonehotencoding=count_vect.transform(xtest)
         xcvonehotencoding=count_vect.transform(xcv)
         print(xtrainonehotencoding.shape)
         print(xtestonehotencoding.shape)
         print(xcvonehotencoding.shape)
```

```
(49152, 41229)
(26332, 41229)
(12289, 41229)
```

```
In [80]: vect=CountVectorizer(min_df=10,max_features=50)
         xtrainonehotencoding1=vect.fit_transform(xtrain[:20000])
         xtestonehotencoding1=vect.transform(xtest[:7000])
         xcvonehotencoding1=vect.transform(xcv[:3000])
         print(xtrainonehotencoding1.shape)
         print(xtestonehotencoding1.shape)
         print(xcvonehotencoding1.shape)
```

```
(20000, 50)
(7000, 50)
(3000, 50)
```

```
In [0]: xtrainonehotencoding11=xtrainonehotencoding1.toarray()
        xtestonehotencoding12=xtestonehotencoding1.toarray()
        xcvonehotencoding13=xcvonehotencoding1.toarray()
```

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

```
In [0]: #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/modu

        # you can choose these numebrs min_df=10, max_features=5000, of your choice
        count_vect = CountVectorizer(ngram_range=(1,2), 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 and bigrams ", final_bigram_
```

```
the type of count vectorizer  <class 'scipy.sparse.csr.csr_matrix'>
the shape of out text BOW vectorizer  (87773, 5000)
the number of unique words including both unigrams and bigrams  5000
```

5.3 [4.3] TF-IDF

```
In [0]: from sklearn.feature_extraction.text import TfidfVectorizer
        tfidf= TfidfVectorizer()
        xtraintfidfencoding=tfidf.fit_transform(xtrain)
        xtesttfidfencoding=tfidf.transform(xtest)
        xcvtfidfencoding=tfidf.transform(xcv)
        print(xtraintfidfencoding.shape)
        print(xtesttfidfencoding.shape)
        print(xcvtfidfencoding.shape)
```

```
(49152, 41229)
(26332, 41229)
(12289, 41229)
```

```
In [0]: vect=CountVectorizer(min_df=10,max_features=50)
        xtraintfidfencoding1=vect.fit_transform(xtrain[:20000])
        xtesttfidfencoding1=vect.transform(xtest[:7000])
        xcvtfidfencoding1=vect.transform(xcv[:3000])
        print(xtraintfidfencoding1.shape)
        print(xtesttfidfencoding1.shape)
        print(xcvtfidfencoding1.shape)
```

```
(20000, 50)
(7000, 50)
(3000, 50)
```

```
In [0]: xtraintfidfencoding11=xtraintfidfencoding1.toarray()
        xtesttfidfencoding12=xtesttfidfencoding1.toarray()
        xcvtfidfencoding13=xcvtfidfencoding1.toarray()
```

5.4 [4.4] Word2Vec

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

```
In [0]: # 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
        # we will provide a pickle file wich contains a dict ,
        # and it contains all our courpus words as keys and  model[word] as values
        # To use this code-snippet, download "GoogleNews-vectors-negative300.bin"
        # from https://drive.google.com/file/d/0B7XkCwpI5KDYNlNUTTlSS21pQmM/edit
        # it's 1.9GB in size.


        # http://kavita-ganesan.com/gensim-word2vec-tutorial-starter-code/#.W17SRFAzZPY
        # you can comment this whole cell
        # or change these varible according to your need

        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 occured atleast 5 times
            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'))

        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.bi
                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, t
```

```
WARNING:gensim.models.base_any2vec:consider setting layer size to a multiple of 4 for greater

[('awesome', 0.8137207627296448), ('good', 0.8112103939056396), ('terrific', 0.769643723964691
==================================================
[('best', 0.7403311729431152), ('greatest', 0.7323489189147949), ('tastiest', 0.70615011453628
In [0]: 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'))
```

```
WARNING:gensim.models.base_any2vec:consider setting layer size to a multiple of 4 for greater p
```

```
[('terrific', 0.8472586274147034), ('awesome', 0.829969048500061), ('good', 0.8249484896659851)
==================================================
[('greatest', 0.7730629444122314), ('best', 0.7612577676773071), ('tastiest', 0.71018093824386
```

```
In [0]: 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  13249
sample words  ['baby', 'food', 'convenient', 'healthy', 'great', 'worry', 'son', 'eating', 'ab
```

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

[4.4.1.1] Avg W2v

```
In [0]: # 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(xtrain): # for each review/sentence
            sent_vec = np.zeros(50) # as word vectors are of zero length 50, you might need to
            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%|| 49152/49152 [1:38:41<00:00,  8.30it/s]
```

```
49152
50
```

```
In [0]: # average Word2Vec
        # compute average word2vec for each review.
        sent_vectorstest = []; # the avg-w2v for each sentence/review is stored in this list
        for sent in tqdm(xtest): # for each review/sentence
            sent_vec = np.zeros(50) # as word vectors are of zero length 50, you might need to
            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_vectorstest.append(sent_vec)
        print(len(sent_vectorstest))
        print(len(sent_vectorstest))
```

```
100%|| 26332/26332 [55:58<00:00,  7.84it/s]

26332
26332
```

```
In [0]: # average Word2Vec
        # compute average word2vec for each review.
        sent_vectorscv = []; # the avg-w2v for each sentence/review is stored in this list
        for sent in tqdm(xcv): # for each review/sentence
            sent_vec = np.zeros(50) # as word vectors are of zero length 50, you might need to
            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_vectorscv.append(sent_vec)
        print(len(sent_vectorscv))
        print(len(sent_vectorscv))
```

```
100%|| 12289/12289 [15:12<00:00, 11.55it/s]

12289
12289
```

[4.4.1.2] TFIDF weighted W2v

```
In [0]: model = TfidfVectorizer()
        xtraintfidfw2v = model.fit_transform(preprocessed_reviews)
        #xtesttfidfw2v=model.transform(xtest)
        #xcvtfidfw2v=model.transform(xcv)
        tfidf_feat = model.get_feature_names()
        dictionary = dict(zip(model.get_feature_names(), list(model.idf_)))
```

```
In [0]: xcvtfidf_sent_vectors = []; # the tfidf-w2v for each sentence/review is stored in this
        row=0;
        for sent in tqdm(xcv): # 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.split(' '): # for each word in a review/sentence
                if word in w2v_words and word in tfidf_feat:
                    vec = w2v_model.wv[word]
                    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
            xcvtfidf_sent_vectors.append(sent_vec)
            row += 1
```

```
100%|| 12289/12289 [07:58<00:00, 25.70it/s]
```

```
In [0]: xtraintfidf_sent_vectors = []; # the tfidf-w2v for each sentence/review is stored in t
        row=0;
        for sent in tqdm(xtrain): # 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.split(' '): # for each word in a review/sentence
                if word in w2v_words and word in tfidf_feat:
                    vec = w2v_model.wv[word]
                    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
            xtraintfidf_sent_vectors.append(sent_vec)
            row += 1
```

```
100%|| 49152/49152 [1:26:29<00:00,  4.26it/s]
```

```
In [0]: xtesttfidf_sent_vectors = []; # the tfidf-w2v for each sentence/review is stored in th
        row=0;
```

```
    for sent in tqdm(xtest): # 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.split(' '): # for each word in a review/sentence
            if word in w2v_words and word in tfidf_feat:
                vec = w2v_model.wv[word]
                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
        xtesttfidf_sent_vectors.append(sent_vec)
        row += 1
```

```
100%|| 26332/26332 [1:27:35<00:00,  7.82it/s]
```

6 [5] Assignment 7: SVM

```
<li><strong>Apply SVM on these feature sets</strong>
    <ul>
        <li><font color='red'>SET 1:</font>Review text, preprocessed one converted into vectors
        <li><font color='red'>SET 2:</font>Review text, preprocessed one converted into vectors
        <li><font color='red'>SET 3:</font>Review text, preprocessed one converted into vectors
        <li><font color='red'>SET 4:</font>Review text, preprocessed one converted into vectors
    </ul>
</li>
<br>
<li><strong>Procedure</strong>
    <ul>
<li>You need to work with 2 versions of SVM
    <ul><li>Linear kernel</li>
        <li>RBF kernel</li></ul>
<li>When you are working with linear kernel, use SGDClassifier with hinge loss because it is co
<li>When you are working with SGDClassifier with hinge loss and trying to find the AUC
    score, you would have to use <a href='https://scikit-learn.org/stable/modules/generated/skl
<li>Similarly, like kdtree of knn, when you are working with RBF kernel it's better to reduce
```

the number of dimensions. You can put min_df = 10, max_features = 500 and consider a sample size of 40k points.

```
    </ul>
</li>
<br>
<li><strong>Hyper paramter tuning (find best alpha in range [10^-4 to 10^4], and the best penal
    <ul>
<li>Find the best hyper parameter which will give the maximum <a href='https://www.appliedaicou
<li>Find the best hyper paramter using k-fold cross validation or simple cross validation data
```

```
<li>Use gridsearch cv or randomsearch cv or you can also write your own for loops to do this ta
    </ul>
</li>
<br>
<li><strong>Feature importance</strong>
    <ul>
<li>When you are working on the linear kernel with BOW or TFIDF please print the top 10 best
```

features for each of the positive and negative classes.

```
    </ul>
</li>
<br>
<li><strong>Feature engineering</strong>
    <ul>
<li>To increase the performance of your model, you can also experiment with with feature engine
        <ul>
        <li>Taking length of reviews as another feature.</li>
        <li>Considering some features from review summary as well.</li>
    </ul>
    </ul>
</li>
<br>
<li><strong>Representation of results</strong>
    <ul>
<li>You need to plot the performance of model both on train data and cross validation data for
<img src='train_cv_auc.JPG' width=300px></li>
<li>Once after you found the best hyper parameter, you need to train your model with it, and fi
<img src='train_test_auc.JPG' width=300px></li>
<li>Along with plotting ROC curve, you need to print the <a href='https://www.appliedaicourse.c
<img src='confusion_matrix.png' width=300px></li>
    </ul>
</li>
<br>
<li><strong>Conclusion</strong>
    <ul>
<li>You need to summarize the results at the end of the notebook, summarize it in the table fo
    <img src='summary.JPG' width=400px>
</li>
    </ul>
```

Note: Data Leakage

1. There will be an issue of data-leakage if you vectorize the entire data and then split it into train/cv/test.
2. To avoid the issue of data-leakag, make sure to split your data first and then vectorize it.
3. While vectorizing your data, apply the method fit_transform() on you train data, and apply the method transform() on cv/test data.
4. For more details please go through this link.

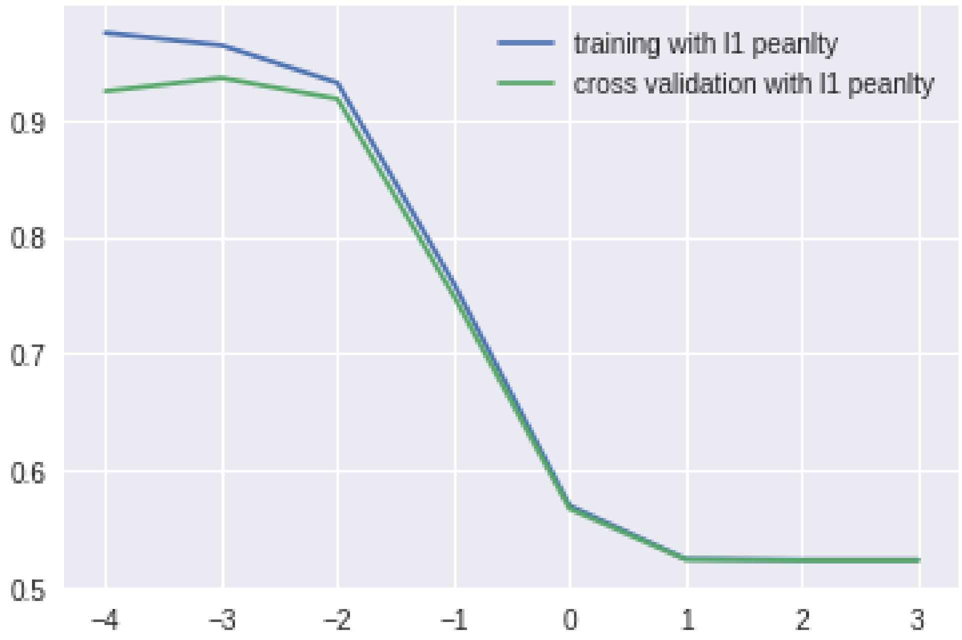
7 Applying SVM

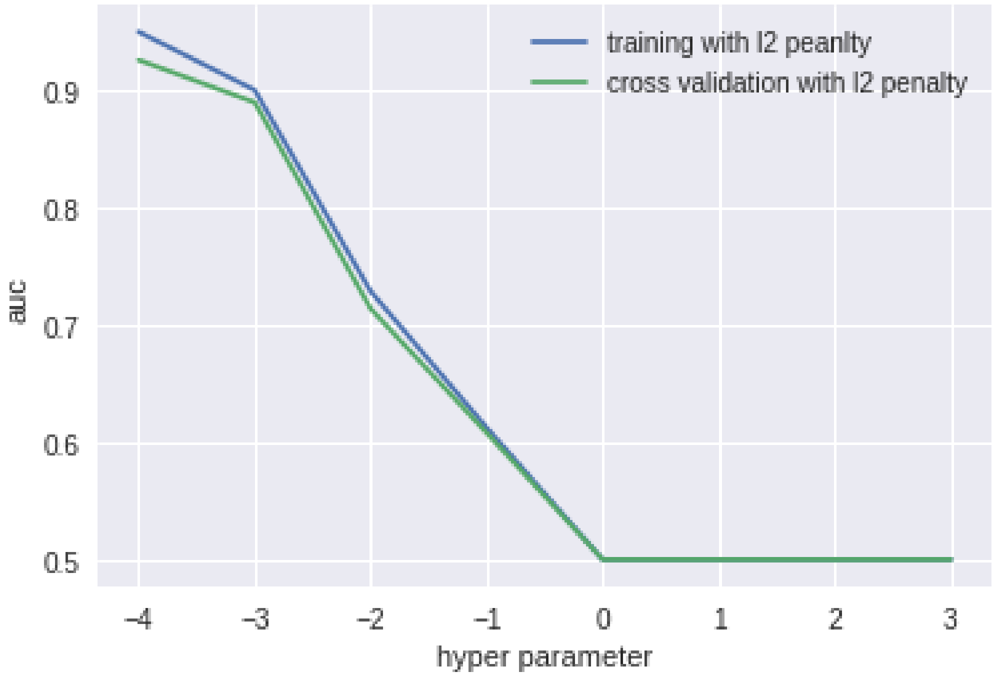
7.1 [5.1] Linear SVM

7.1.1 [5.1.1] Applying Linear SVM on BOW, SET 1

```
In [0]: #with hyper parameter tuning
        from sklearn.metrics import auc
        from sklearn.metrics import roc_auc_score
        from sklearn.svm import SVC
        from sklearn.linear_model import SGDClassifier
        from sklearn.calibration import CalibratedClassifierCV
        cvscores_with_l1=[]
        cvscores_with_l2=[]
        cvscores=[]
        cvscores1=[]
        cvscores1_with_l1=[]
        cvscores1_with_l2=[]

        alpha=[10**i for i in range(-4,4,1)]
        beta=['l1','l2']
        for i in alpha:
          for j in beta:
            svca=SGDClassifier(alpha=i,loss='hinge',penalty=j)
            svca.fit(xtrainonehotencoding,ytrain)
            svcx=CalibratedClassifierCV(svca,method='sigmoid')
            svcx.fit(xtrainonehotencoding,ytrain)
            predict1=svcx.predict_proba(xtrainonehotencoding)[:,1]
            cvscores.append(roc_auc_score(ytrain,predict1))
            predict2=svcx.predict_proba(xcvonehotencoding)[:,1]
            cvscores1.append(roc_auc_score(ycv,predict2))
            optimal_k=np.argmax(cvscores1)
        fig,ax=plt.subplots()
        fig,ax1=plt.subplots()
        cvscores_with_l1=[cvscores[x] for x in range(len(cvscores)) if x%2 != 0]
        cvscores_with_l2=[cvscores[x] for x in range(len(cvscores)) if x%2 == 0]
        ax.plot(np.log10(alpha),cvscores_with_l1,label='training with l1 peanlty')
        ax1.plot(np.log10(alpha),cvscores_with_l2,label='training with l2 peanlty')
        ax1.legend()
        ax.legend()
        cvscores1_with_l1=[cvscores1[x] for x in range(len(cvscores1)) if x%2 != 0]
        cvscores1_with_l2=[cvscores1[x] for x in range(len(cvscores1)) if x%2 == 0]
        ax.plot(np.log10(alpha),cvscores1_with_l1,label='cross validation with l1 peanlty')
        ax1.plot(np.log10(alpha),cvscores1_with_l2,label='cross validation with l2 penalty')
        ax1.legend()
        ax.legend()
        plt.xlabel('hyper parameter')
        plt.ylabel('auc')
        plt.show()
        optimal_k=np.argmax(cvscores1)
        print(cvscores1_with_l1)
        print(cvscores1_with_l2)
```





```
[0.9246012407403723, 0.9359948290783802, 0.9180348664737614, 0.749703680215667, 0.5670509560822
[0.9258863657017113, 0.8896977635705491, 0.7137514318710211, 0.6080766691121093, 0.5, 0.5, 0.5
```

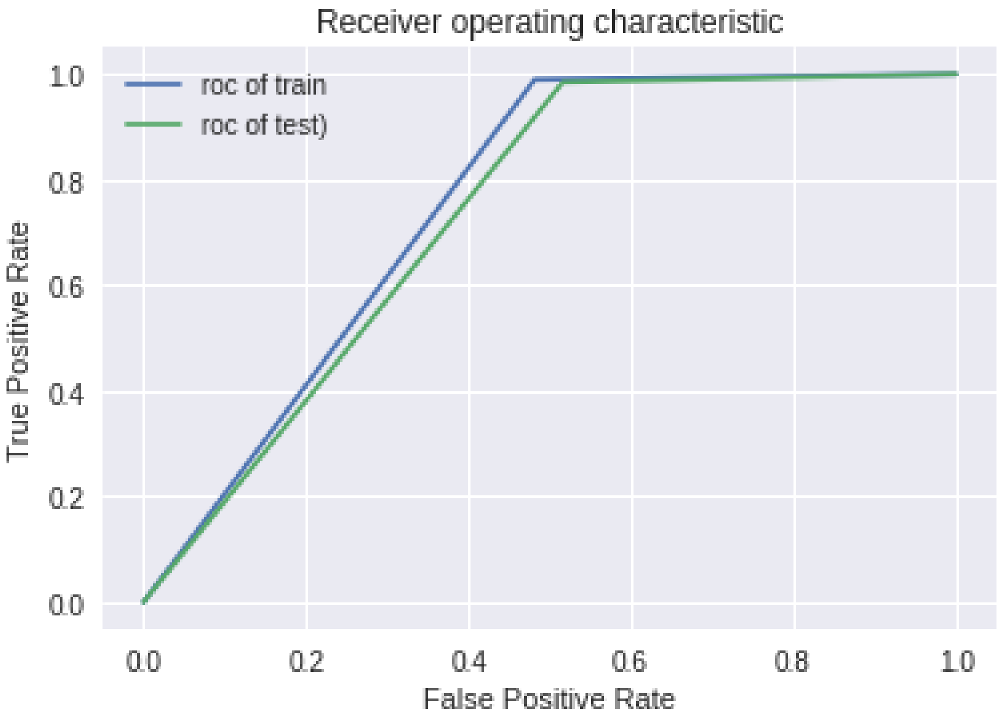
from the above scores and graph it is evident that l1 penalty with 10**-4 as alpha is best

```
In [0]: x=svca.coef_.flatten()
        y=count_vect.get_feature_names()
        print(x)
        print('printing top 10 negativew words')
        a=np.argsort(x)
        lis=[ y[i] for i in a[:10]]
        print(lis)
        print('printi9ng positive words')
        a1=np.argsort(x)[::-1]
        lis=[y[i] for i in a1[:10]]
        print(lis)
```

```
[2.44141613e-07 8.13805376e-08 4.06902688e-08 ... 6.10354032e-08
 2.03451344e-08 2.03451344e-08]
printing top 10 negativew words
['worst', 'horrible', 'return', 'awful', 'refund', 'waste', 'terrible', 'threw', 'disappointing
printi9ng positive words
['great', 'good', 'like', 'one', 'love', 'coffee', 'flavor', 'tea', 'taste', 'would']
```

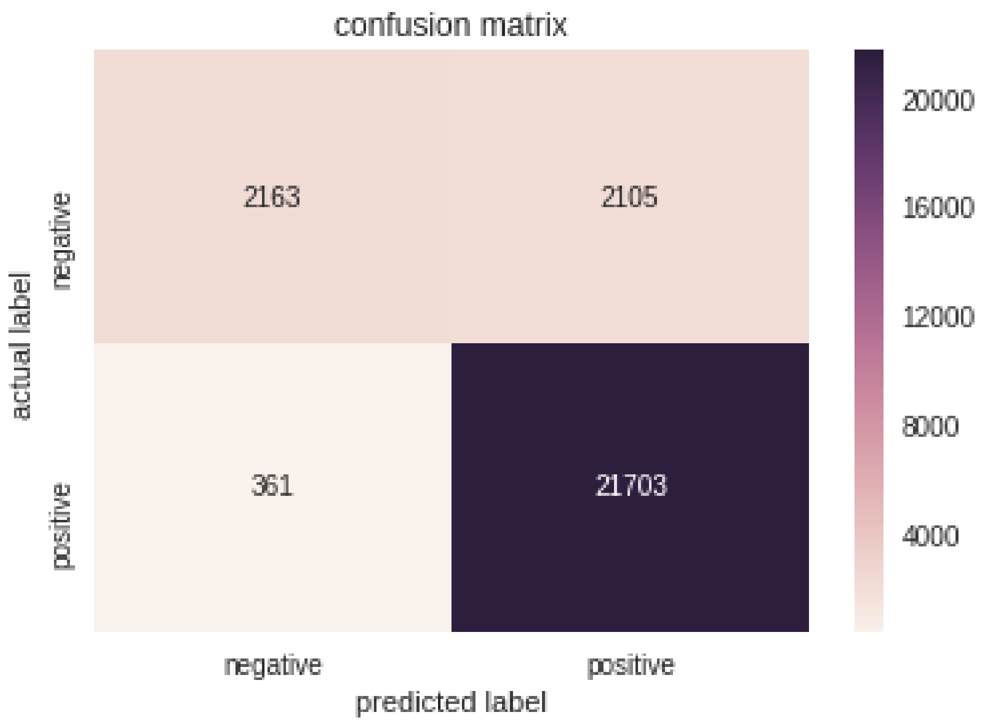
```
In [0]: from sklearn.svm import SVC
        from sklearn.metrics import accuracy_score
        from sklearn.metrics import roc_auc_score
        svce=SGDClassifier(alpha=10**-4,loss='hinge',penalty='l1')
        svce.fit(xtrainonehotencoding,ytrain)
        model=CalibratedClassifierCV(svce,method='sigmoid')
        model.fit(xtrainonehotencoding,ytrain)
        predictrain=model.predict(xtrainonehotencoding)
        fpr, tpr, thresh = metrics.roc_curve(ytrain, predictrain)
        auc = metrics.roc_auc_score(ytrain, predictrain)
        plt.plot(fpr,tpr,label="roc of train")
        plt.legend()
        predic=model.predict(xtestonehotencoding)
        fpr, tpr, thresh = metrics.roc_curve(ytest, predic)
        auc = metrics.roc_auc_score(ytest, predic)
        plt.plot(fpr,tpr,label="roc of test)")
        plt.legend()
        plt.xlabel('False Positive Rate')
        plt.ylabel('True Positive Rate')
        plt.title('Receiver operating characteristic')
        print(roc_auc_score(ytest, predic))
```

```
0.7342608968168258
```



```
In [0]: #plotting confusion matrix aftyer performing knn on top of svd data
        from sklearn.metrics import confusion_matrix
        rest=confusion_matrix(ytest,predic)
        import seaborn as sns
        classlabel=['negative','positive']

        frame=pd.DataFrame(rest,index=classlabel,columns=classlabel)
        sns.heatmap(frame,annot=True,fmt="d")
        plt.title("confusion matrix")
        plt.xlabel("predicted label")
        plt.ylabel("actual label")
        plt.show()
```

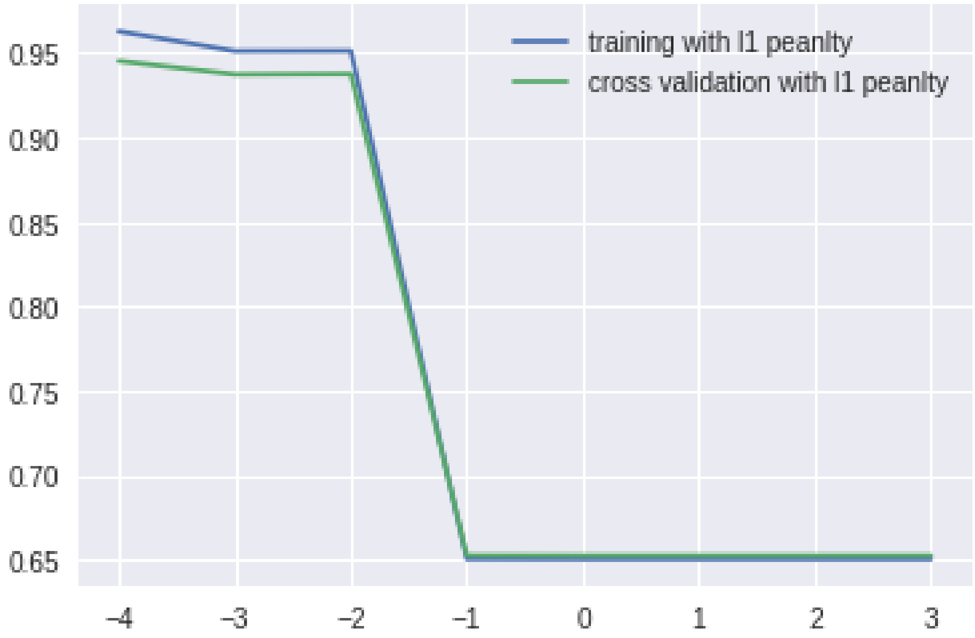


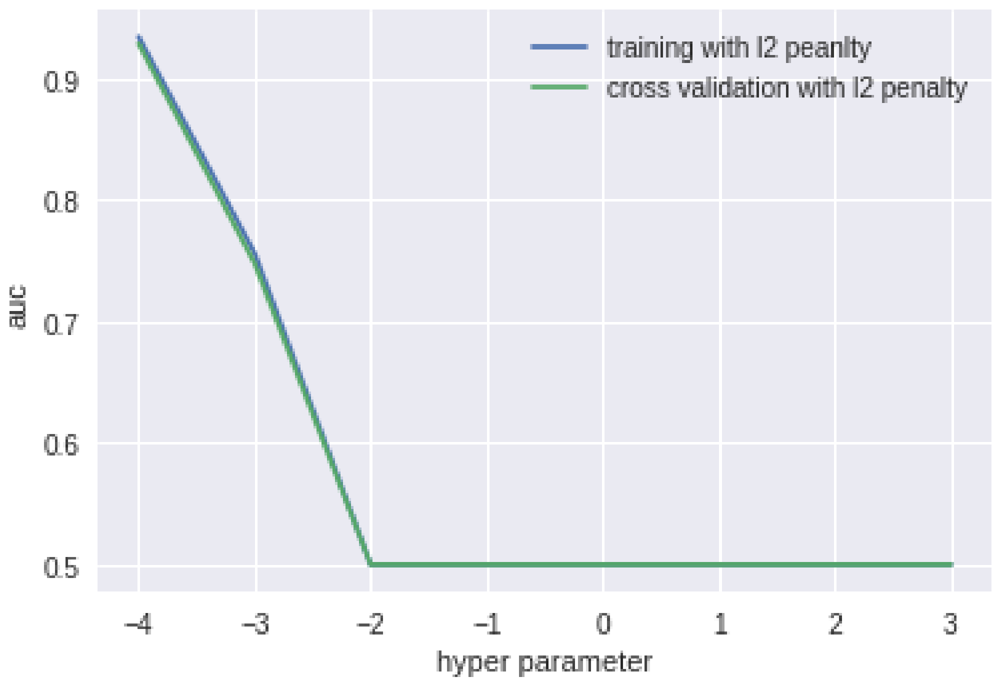
7.1.2 [5.1.2] Applying Linear SVM on TFIDF, SET 2

```
In [0]: #with hyper parameter tuning
        from sklearn.metrics import auc
        from sklearn.metrics import roc_auc_score
        from sklearn.svm import SVC
        from sklearn.linear_model import SGDClassifier
        from sklearn.calibration import CalibratedClassifierCV
        cvscores_with_l1=[]
        cvscores_with_l2=[]
        cvscores=[]
        cvscores1=[]
        cvscores1_with_l1=[]
        cvscores1_with_l2=[]

        alpha=[10**i for i in range(-4,4,1)]
        beta=['l1','l2']
        for i in alpha:
          for j in beta:
            svca=SGDClassifier(alpha=i,loss='hinge',penalty=j)
            svca.fit(xtraintfidfencoding,ytrain)
            svcx=CalibratedClassifierCV(svca,method='sigmoid')
            svcx.fit(xtraintfidfencoding,ytrain)
            predict1=svcx.predict_proba(xtraintfidfencoding)[:,1]
```

```
    cvscores.append(roc_auc_score(ytrain,predict1))
    predict2=svcx.predict_proba(xcvtfidfencoding)[:,1]
    cvscores1.append(roc_auc_score(ycv,predict2))
    optimal_k=np.argmax(cvscores1)
fig,ax=plt.subplots()
fig,ax1=plt.subplots()
cvscores_with_l1=[cvscores[x] for x in range(len(cvscores)) if x%2 != 0]
cvscores_with_l2=[cvscores[x] for x in range(len(cvscores)) if x%2 == 0]
ax.plot(np.log10(alpha),cvscores_with_l1,label='training with l1 peanlty')
ax1.plot(np.log10(alpha),cvscores_with_l2,label='training with l2 peanlty')
ax1.legend()
ax.legend()
cvscores1_with_l1=[cvscores1[x] for x in range(len(cvscores1)) if x%2 != 0]
cvscores1_with_l2=[cvscores1[x] for x in range(len(cvscores1)) if x%2 == 0]
ax.plot(np.log10(alpha),cvscores1_with_l1,label='cross validation with l1 peanlty')
ax1.plot(np.log10(alpha),cvscores1_with_l2,label='cross validation with l2 penalty')
ax1.legend()
ax.legend()
plt.xlabel('hyper parameter')
plt.ylabel('auc')
plt.show()
optimal_k=np.argmax(cvscores1)
print(cvscores1_with_l1)
print(cvscores1_with_l2)
```





```
[0.9453673195257479, 0.9374718939855762, 0.937752417848533, 0.652610490519912, 0.65251639753146
[0.9283744914590868, 0.7475720352518161, 0.5, 0.5, 0.5, 0.5, 0.5, 0.5]
```

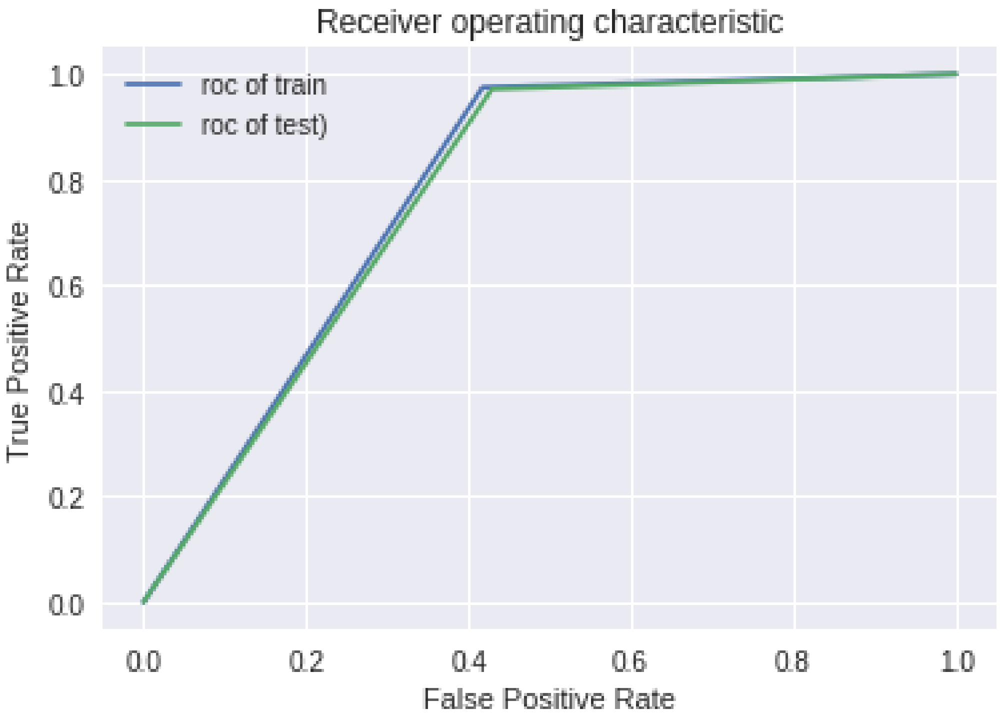
```
In [0]: x=svca.coef_.flatten()
        y=tfidf.get_feature_names()
        print(x)
        print('printing top 10 negativew words')
        a=np.argsort(x)
        lis=[ y[i] for i in a[:10]]
        print(lis)
        print('printi9ng positive words')
        a1=np.argsort(x)[::-1]
        lis=[y[i] for i in a1[:10]]
        print(lis)
```

```
[4.75867682e-08 2.56062485e-08 1.83931617e-08 ... 1.68671552e-08
 2.54810478e-09 4.31768671e-09]
printing top 10 negativew words
['worst', 'horrible', 'awful', 'return', 'waste', 'threw', 'terrible', 'disgusting', 'disappoi
printi9ng positive words
['great', 'good', 'love', 'coffee', 'tea', 'like', 'one', 'flavor', 'best', 'taste']
```

from the above graph it is evident that 10**-4 with l1 paenalty is best

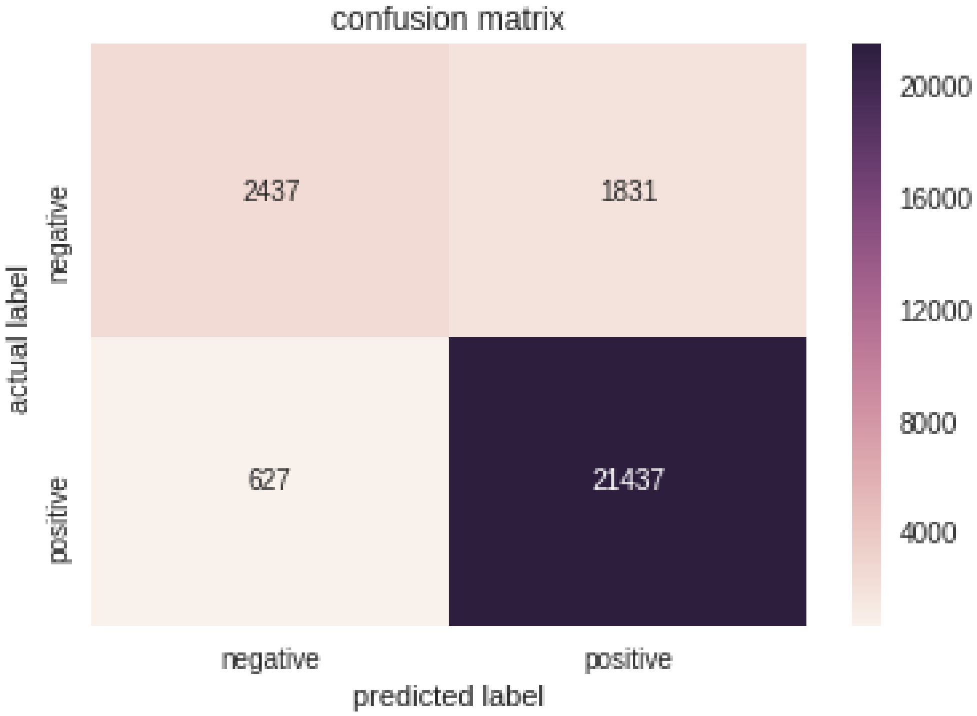
```
In [0]: from sklearn.neighbors import KNeighborsClassifier
        from sklearn.metrics import accuracy_score
        from sklearn.metrics import roc_auc_score
        svce=SGDClassifier(alpha=10**-4,loss='hinge',penalty='l1')
        svce.fit(xtraintfidfencoding,ytrain)
        model=CalibratedClassifierCV(svce,method='sigmoid')
        model.fit(xtraintfidfencoding,ytrain)
        predictrain=model.predict(xtraintfidfencoding)
        fpr, tpr, thresh = metrics.roc_curve(ytrain, predictrain)
        auc = metrics.roc_auc_score(ytrain, predictrain)
        plt.plot(fpr,tpr,label="roc of train")
        plt.legend()
        predic=model.predict(xtesttfidfencoding)
        fpr, tpr, thresh = metrics.roc_curve(ytest, predic)
        auc = metrics.roc_auc_score(ytest, predic)
        plt.plot(fpr,tpr,label="roc of test)")
        plt.legend()
        plt.xlabel('False Positive Rate')
        plt.ylabel('True Positive Rate')
        plt.title('Receiver operating characteristic')
        print(roc_auc_score(ytest, predic))
```

```
0.7712880540752878
```



```
In [0]: #plotting confusion matrix aftyer performing knn on top of svd data
        from sklearn.metrics import confusion_matrix
        rest=confusion_matrix(ytest,predic)
        import seaborn as sns
        classlabel=['negative','positive']

        frame=pd.DataFrame(rest,index=classlabel,columns=classlabel)
        sns.heatmap(frame,annot=True,fmt="d")
        plt.title("confusion matrix")
        plt.xlabel("predicted label")
        plt.ylabel("actual label")
        plt.show()
```

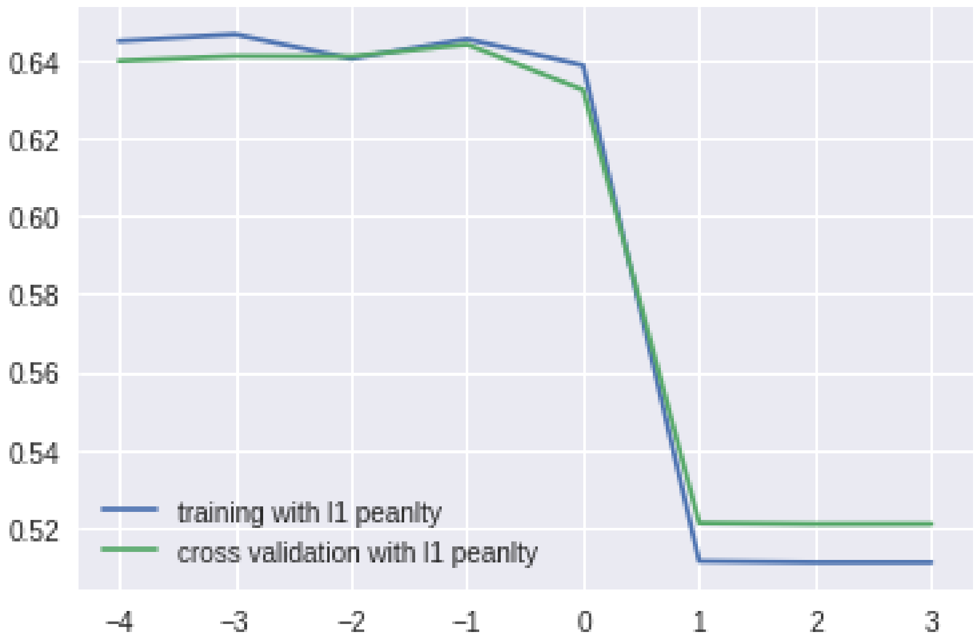


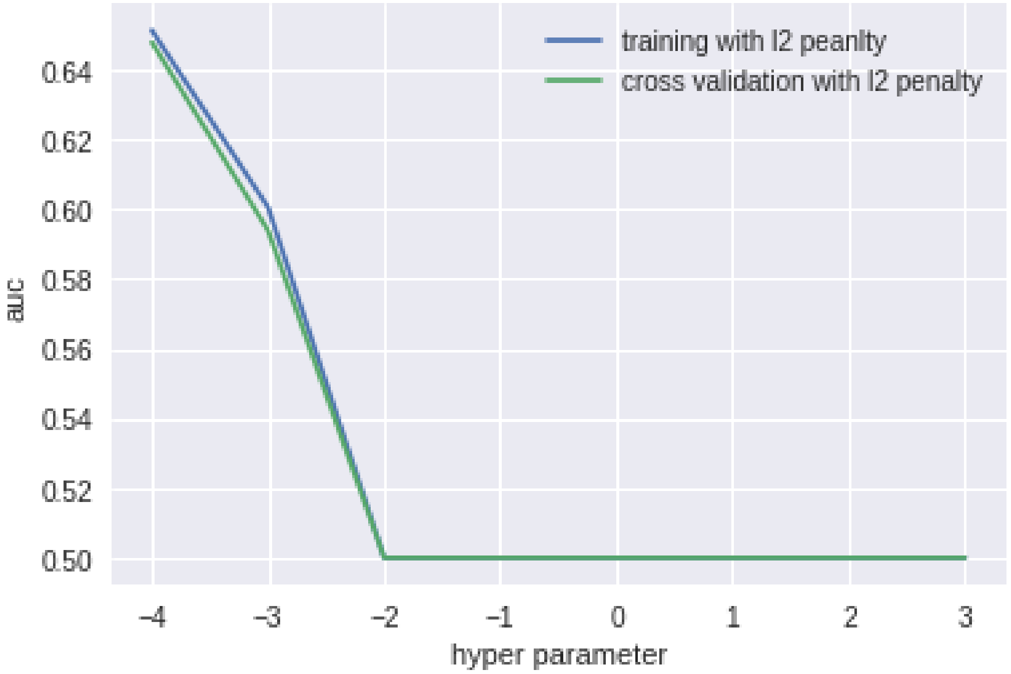
7.1.3 [5.1.3] Applying Linear SVM on AVG W2V, SET 3

```
In [0]: #with hyper parameter tuning
        from sklearn.metrics import auc
        from sklearn.metrics import roc_auc_score
        from sklearn.svm import SVC
        from sklearn.linear_model import SGDClassifier
        from sklearn.calibration import CalibratedClassifierCV
        cvscores_with_l1=[]
        cvscores_with_l2=[]
        cvscores=[]
```

```
cvscores1=[]
cvscores1_with_l1=[]
cvscores1_with_l2=[]

alpha=[10**i for i in range(-4,4,1)]
beta=['l1','l2']
for i in alpha:
  for j in beta:
    svca=SGDClassifier(alpha=i,loss='hinge',penalty=j)
    svca.fit(sent_vectors,ytrain)
    svcx=CalibratedClassifierCV(svca,method='sigmoid')
    svcx.fit(sent_vectors,ytrain)
    predict1=svcx.predict_proba(sent_vectors)[:,1]
    cvscores.append(roc_auc_score(ytrain,predict1))
    predict2=svcx.predict_proba(sent_vectorscv)[:,1]
    cvscores1.append(roc_auc_score(ycv,predict2))
    optimal_k=np.argmax(cvscores1)
fig,ax=plt.subplots()
fig,ax1=plt.subplots()
cvscores_with_l1=[cvscores[x] for x in range(len(cvscores)) if x%2 != 0]
cvscores_with_l2=[cvscores[x] for x in range(len(cvscores)) if x%2 == 0]
ax.plot(np.log10(alpha),cvscores_with_l1,label='training with l1 peanlty')
ax1.plot(np.log10(alpha),cvscores_with_l2,label='training with l2 peanlty')
ax1.legend()
ax.legend()
cvscores1_with_l1=[cvscores1[x] for x in range(len(cvscores1)) if x%2 != 0]
cvscores1_with_l2=[cvscores1[x] for x in range(len(cvscores1)) if x%2 == 0]
ax.plot(np.log10(alpha),cvscores1_with_l1,label='cross validation with l1 peanlty')
ax1.plot(np.log10(alpha),cvscores1_with_l2,label='cross validation with l2 penalty')
ax1.legend()
ax.legend()
plt.xlabel('hyper parameter')
plt.ylabel('auc')
plt.show()
optimal_k=np.argmax(cvscores1)
print(cvscores1_with_l1)
print(cvscores1_with_l2)
```



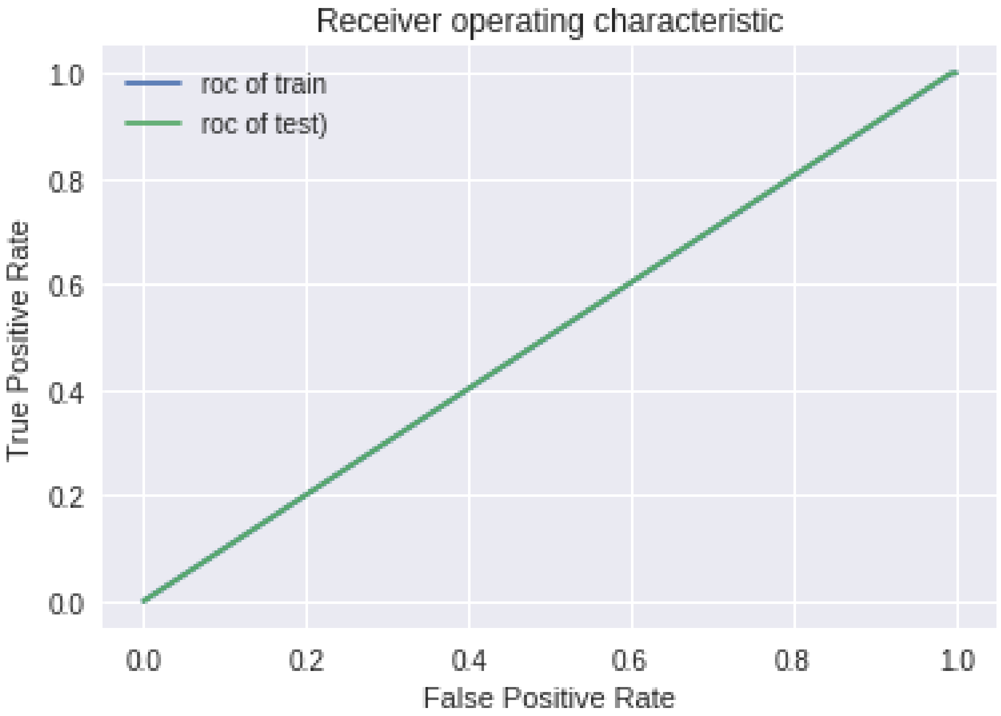


```
[0.6397218334981585, 0.6409457977857421, 0.6408536862519318, 0.6439388200357564, 0.632325191247
[0.6478143912994372, 0.5938681954541323, 0.5, 0.5, 0.5, 0.5, 0.5, 0.5]
```

from abopve graph and values it is evident that 10**-4 and l1 penalty is good

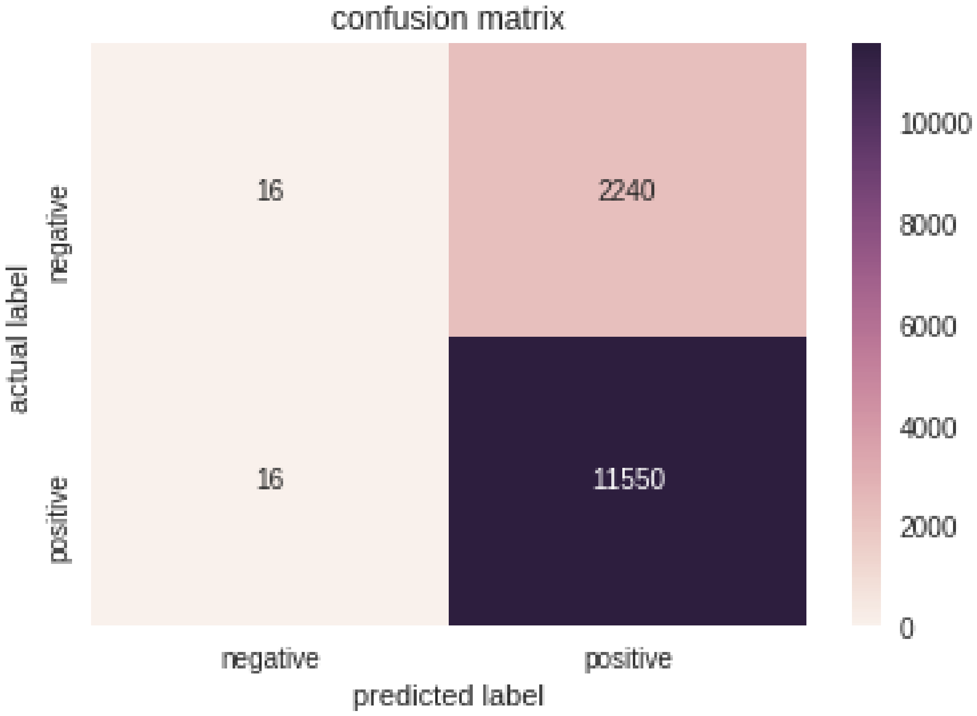
```
In [0]: from sklearn.neighbors import KNeighborsClassifier
        from sklearn.metrics import accuracy_score
        from sklearn.metrics import roc_auc_score
        svce=SGDClassifier(alpha=10**-4,loss='hinge',penalty='l1')
        svce.fit(sent_vectors,ytrain)
        model=CalibratedClassifierCV(svce,method='sigmoid')
        model.fit(sent_vectors,ytrain)
        predictrain=model.predict(sent_vectors)
        fpr, tpr, thresh = metrics.roc_curve(ytrain, predictrain)
        auc = metrics.roc_auc_score(ytrain, predictrain)
        plt.plot(fpr,tpr,label="roc of train")
        plt.legend()
        predic=model.predict(sent_vectorstest)
        fpr, tpr, thresh = metrics.roc_curve(ytest, predic)
        auc = metrics.roc_auc_score(ytest, predic)
        plt.plot(fpr,tpr,label="roc of test)")
        plt.legend()
        plt.xlabel('False Positive Rate')
        plt.ylabel('True Positive Rate')
        plt.title('Receiver operating characteristic')
        print(roc_auc_score(ytest, predic))
```

```
0.5028544167730558
```



```
In [0]: #plotting confusion matrix aftyer performing knn on top of svd data
        from sklearn.metrics import confusion_matrix
        rest=confusion_matrix(ytest,predic)
        import seaborn as sns
        classlabel=['negative','positive']

        frame=pd.DataFrame(rest,index=classlabel,columns=classlabel)
        sns.heatmap(frame,annot=True,fmt="d")
        plt.title("confusion matrix")
        plt.xlabel("predicted label")
        plt.ylabel("actual label")
        plt.show()
```

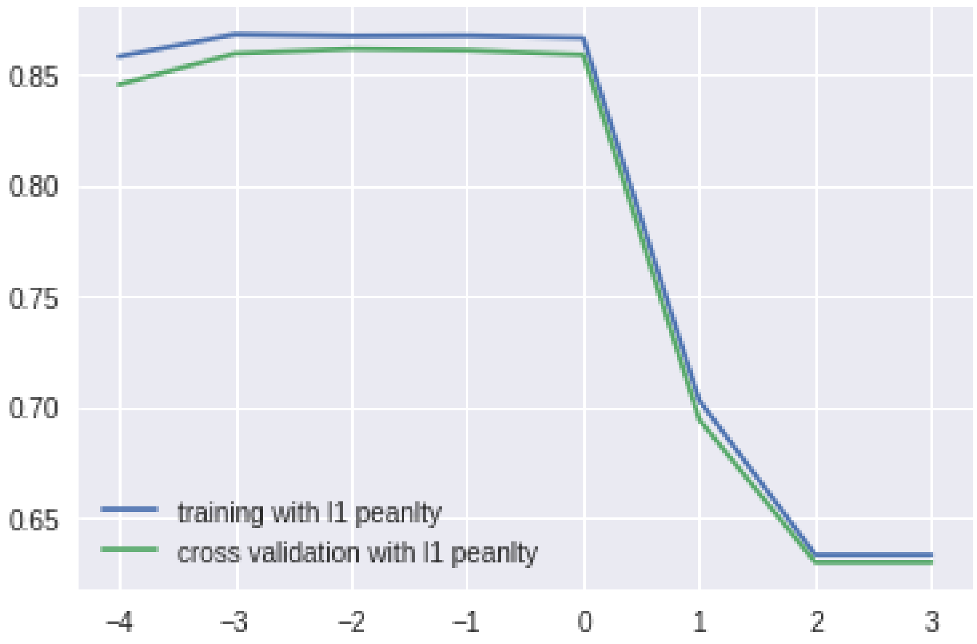


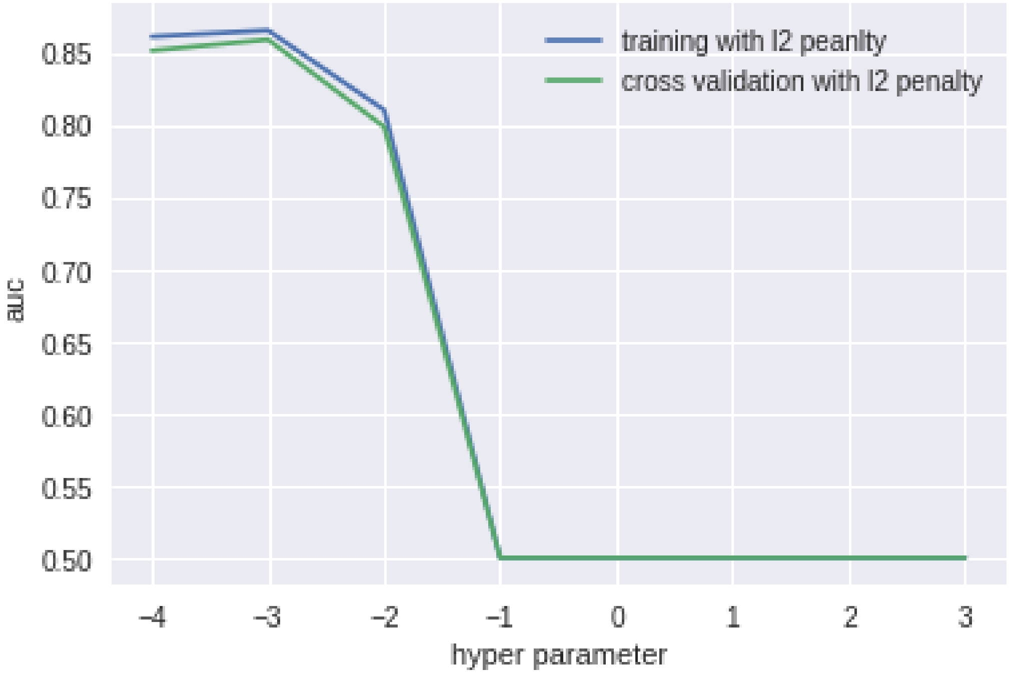
7.1.4 [5.1.4] Applying Linear SVM on TFIDF W2V, SET 4

```
In [0]: #with hyper parameter tuning
        from sklearn.metrics import auc
        from sklearn.metrics import roc_auc_score
        from sklearn.svm import SVC
        from sklearn.linear_model import SGDClassifier
        from sklearn.calibration import CalibratedClassifierCV
```

```
cvscores_with_l1=[]
cvscores_with_l2=[]
cvscores=[]
cvscores1=[]
cvscores1_with_l1=[]
cvscores1_with_l2=[]

alpha=[10**i for i in range(-4,4,1)]
beta=['l1','l2']
for i in alpha:
  for j in beta:
    svca=SGDClassifier(alpha=i,loss='hinge',penalty=j)
    svca.fit(xtraintfidf_sent_vectors,ytrain)
    svcx=CalibratedClassifierCV(svca,method='sigmoid')
    svcx.fit(xtraintfidf_sent_vectors,ytrain)
    predict1=svcx.predict_proba(xtraintfidf_sent_vectors)[:,1]
    cvscores.append(roc_auc_score(ytrain,predict1))
    predict2=svcx.predict_proba(xcvtfidf_sent_vectors)[:,1]
    cvscores1.append(roc_auc_score(ycv,predict2))
    optimal_k=np.argmax(cvscores1)
fig,ax=plt.subplots()
fig,ax1=plt.subplots()
cvscores_with_l1=[cvscores[x] for x in range(len(cvscores)) if x%2 != 0]
cvscores_with_l2=[cvscores[x] for x in range(len(cvscores)) if x%2 == 0]
ax.plot(np.log10(alpha),cvscores_with_l1,label='training with l1 peanlty')
ax1.plot(np.log10(alpha),cvscores_with_l2,label='training with l2 peanlty')
ax1.legend()
ax.legend()
plt.xlabel('hyper parameter')
plt.ylabel('auc')
cvscores1_with_l1=[cvscores1[x] for x in range(len(cvscores1)) if x%2 != 0]
cvscores1_with_l2=[cvscores1[x] for x in range(len(cvscores1)) if x%2 == 0]
ax.plot(np.log10(alpha),cvscores1_with_l1,label='cross validation with l1 peanlty')
ax1.plot(np.log10(alpha),cvscores1_with_l2,label='cross validation with l2 penalty')
ax1.legend()
ax.legend()
plt.xlabel('hyper parameter')
plt.ylabel('auc')
plt.show()
optimal_k=np.argmax(cvscores1)
print(cvscores1_with_l1)
print(cvscores1_with_l2)
```



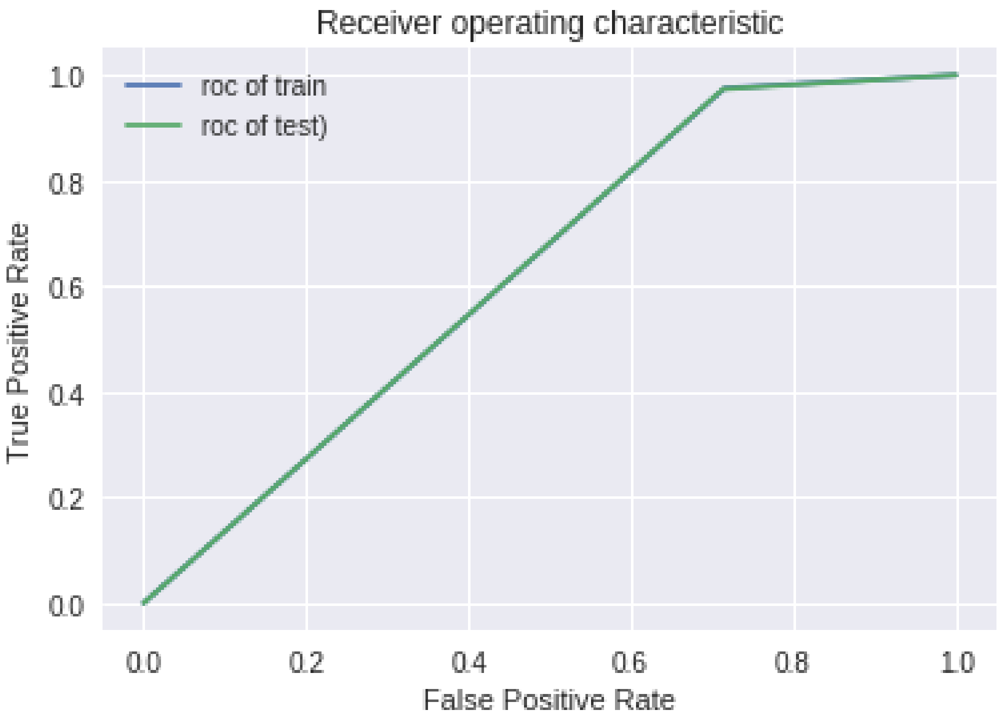


```
[0.8452044152932006, 0.8591853965857785, 0.861271077184989, 0.8606621080352936, 0.8586003592177
[0.8512713802061096, 0.8589157766569058, 0.798398326772467, 0.5, 0.5, 0.5, 0.5, 0.5]
```

from abopve graph and values it is evident that 10**-2 and l1 penalty is good

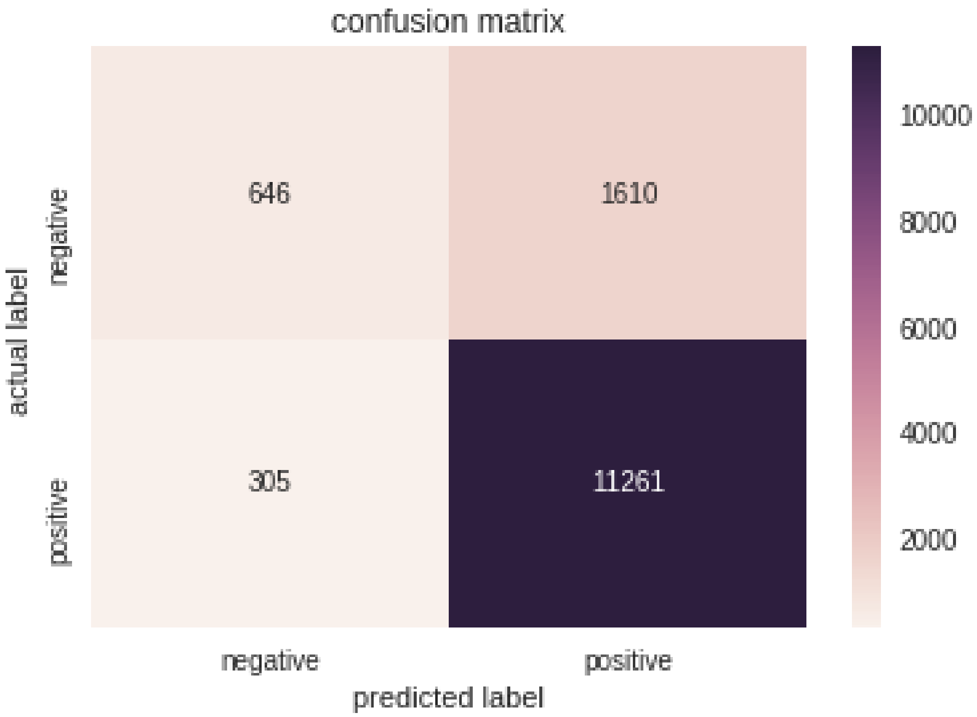
```
In [0]: from sklearn.neighbors import KNeighborsClassifier
        from sklearn.metrics import accuracy_score
        from sklearn.metrics import roc_auc_score
        svce=SGDClassifier(alpha=10**-2,loss='hinge',penalty='l1')
        svce.fit( xtraintfidf_sent_vectors,ytrain)
        model=CalibratedClassifierCV(svce,method='sigmoid')
        model.fit(xtraintfidf_sent_vectors,ytrain)
        predictrain=model.predict( xtraintfidf_sent_vectors)
        fpr, tpr, thresh = metrics.roc_curve(ytrain, predictrain)
        auc = metrics.roc_auc_score(ytrain, predictrain)
        plt.plot(fpr,tpr,label="roc of train")
        plt.legend()
        predic=model.predict( xtesttfidf_sent_vectors)
        fpr, tpr, thresh = metrics.roc_curve(ytest, predic)
        auc = metrics.roc_auc_score(ytest, predic)
        plt.plot(fpr,tpr,label="roc of test)")
        plt.legend()
        plt.xlabel('False Positive Rate')
        plt.ylabel('True Positive Rate')
        plt.title('Receiver operating characteristic')
        print(roc_auc_score(ytest, predic))
```

```
0.6299885608711275
```



```
In [0]: #plotting confusion matrix aftyer performing knn on top of svd data
        from sklearn.metrics import confusion_matrix
        rest=confusion_matrix(ytest,predic)
        import seaborn as sns
        classlabel=['negative','positive']

        frame=pd.DataFrame(rest,index=classlabel,columns=classlabel)
        sns.heatmap(frame,annot=True,fmt="d")
        plt.title("confusion matrix")
        plt.xlabel("predicted label")
        plt.ylabel("actual label")
        plt.show()
```

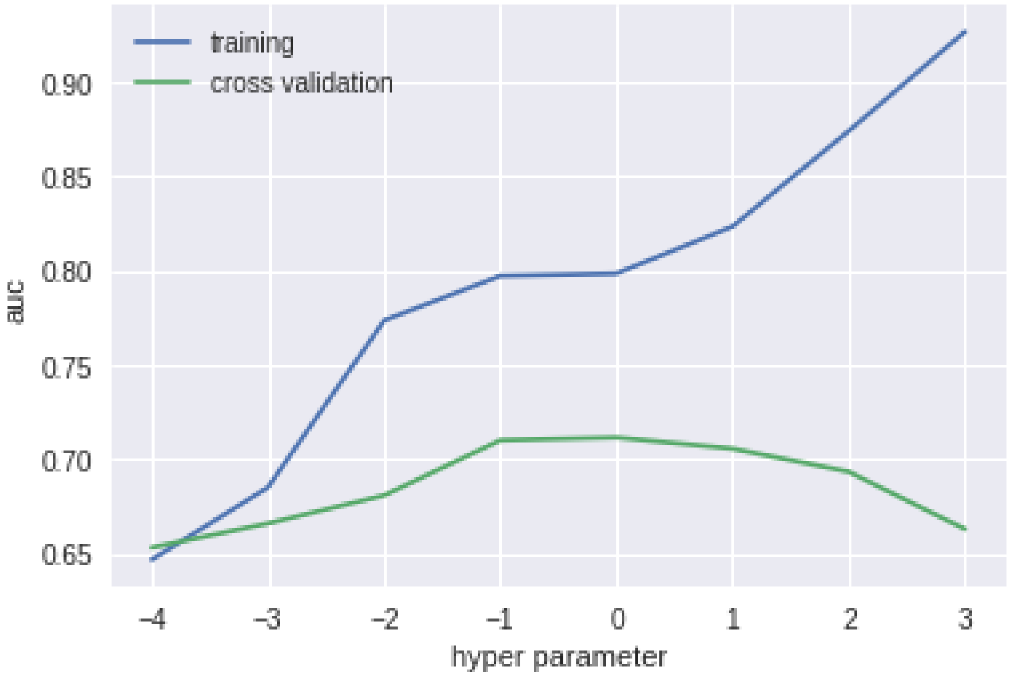


7.2 [5.2] RBF SVM

7.2.1 [5.2.1] Applying RBF SVM on BOW, SET 1

```
In [0]: ytrain1=ytrain[:20000]
        ycv1=ycv[:3000]
        ytest1=ytest[:7000]
```

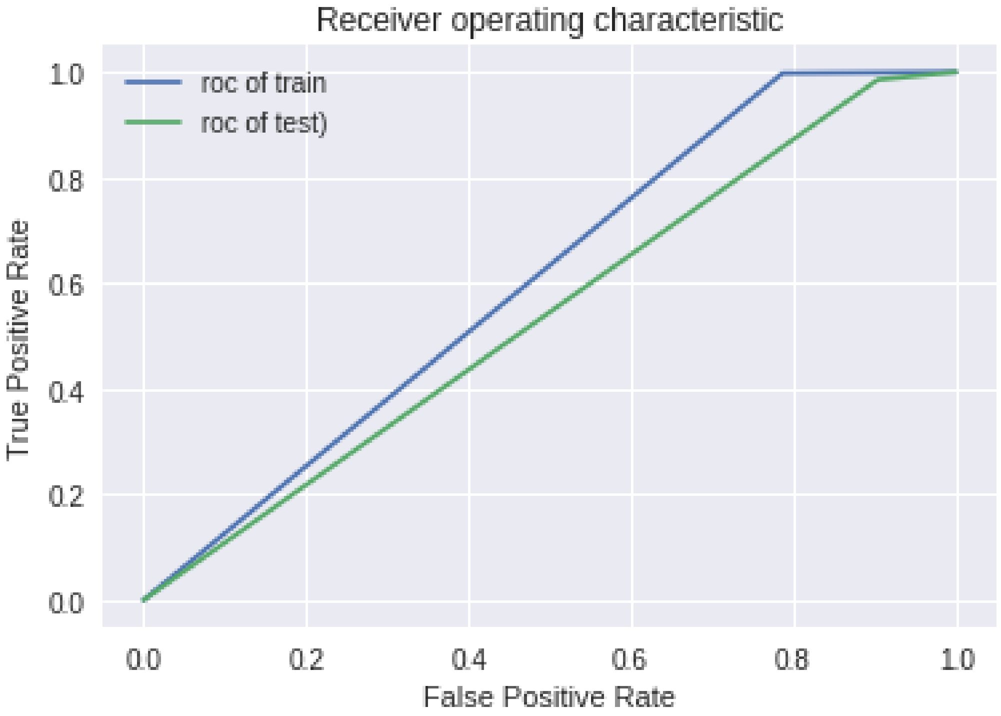
```
In [85]: #@title Default title text
         #with hyper parameter tuning
         from sklearn.metrics import auc
         from sklearn.metrics import roc_auc_score
         from sklearn.neighbors import KNeighborsClassifier
         from sklearn.calibration import CalibratedClassifierCV
         cvscores=[]
         cvscores1=[]
         from sklearn.svm import SVC
         alpha=[10**i for i in range(-4,4,1)]
         for i in alpha:
             svcx=SVC(C=i,kernel='rbf')
             svcx.fit(xtrainonehotencoding11,ytrain1)
             model=CalibratedClassifierCV(svcx,method='sigmoid')
             model.fit(xtrainonehotencoding11,ytrain1)
             predicty=model.predict(xtrainonehotencoding11)[:1]
             predict1=model.predict_proba(xtrainonehotencoding11)[:,1]
             cvscores.append(roc_auc_score(ytrain1,predict1))
             predict2=model.predict_proba(xcvonehotencoding13)[:,1]
             cvscores1.append(roc_auc_score(ycv1,predict2))
             optimal_k=np.argmax(cvscores1)
         fig,ax=plt.subplots()
         ax.plot(np.log10(alpha),cvscores,label='training')
         ax.legend()
         ax.plot(np.log10(alpha),cvscores1,label='cross validation')
         ax.legend()
         plt.xlabel('hyper parameter')
         plt.ylabel('auc')
         plt.show()
         print(cvscores)
         optimal_k=np.argmax(cvscores1)
         print(alpha[optimal_k])
         print(cvscores1)
```



```
[0.6531204007507325, 0.6659113771080596, 0.6807918472610416, 0.7099156438611416, 0.71140343770
1
[0.6531204007507325, 0.6659113771080596, 0.6807918472610416, 0.7099156438611416, 0.71140343770
```

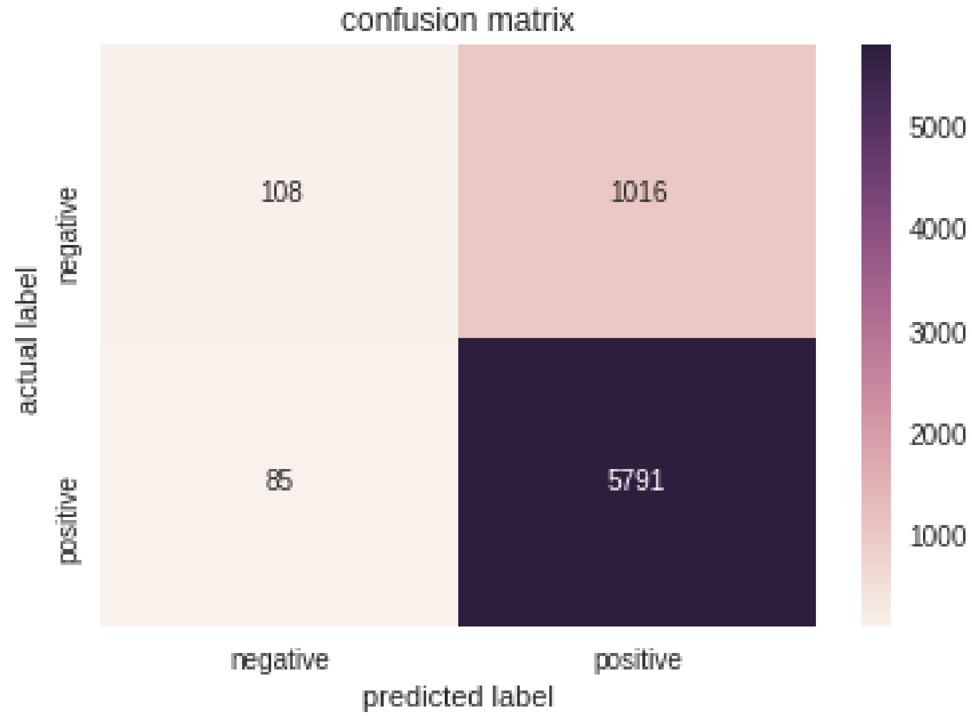
```
In [0]: from sklearn.metrics import accuracy_score
        from sklearn.metrics import roc_auc_score
        svce=SVC(C=alpha[optimal_k],kernel='rbf')
        svce.fit(xtrainonehotencoding11,ytrain1)
        predictrain=svce.predict(xtrainonehotencoding11)
        fpr, tpr, thresh = metrics.roc_curve(ytrain1, predictrain)
        auc = metrics.roc_auc_score(ytrain1, predictrain)
        plt.plot(fpr,tpr,label="roc of train")
        plt.legend()
        predic=svce.predict(xtestonehotencoding12)
        fpr, tpr, thresh = metrics.roc_curve(ytest1, predic)
        auc = metrics.roc_auc_score(ytest1, predic)
        plt.plot(fpr,tpr,label="roc of test)")
        plt.legend()
        plt.xlabel('False Positive Rate')
        plt.ylabel('True Positive Rate')
        plt.title('Receiver operating characteristic')
        print(roc_auc_score(ytest1, predic))
```

```
0.5408098931899833
```



```
In [0]: #plotting confusion matrix aftyer performing knn on top of svd data
        from sklearn.metrics import confusion_matrix
        rest=confusion_matrix(ytest1,predic)
        import seaborn as sns
        classlabel=['negative','positive']

        frame=pd.DataFrame(rest,index=classlabel,columns=classlabel)
        sns.heatmap(frame,annot=True,fmt="d")
        plt.title("confusion matrix")
        plt.xlabel("predicted label")
        plt.ylabel("actual label")
        plt.show()
```

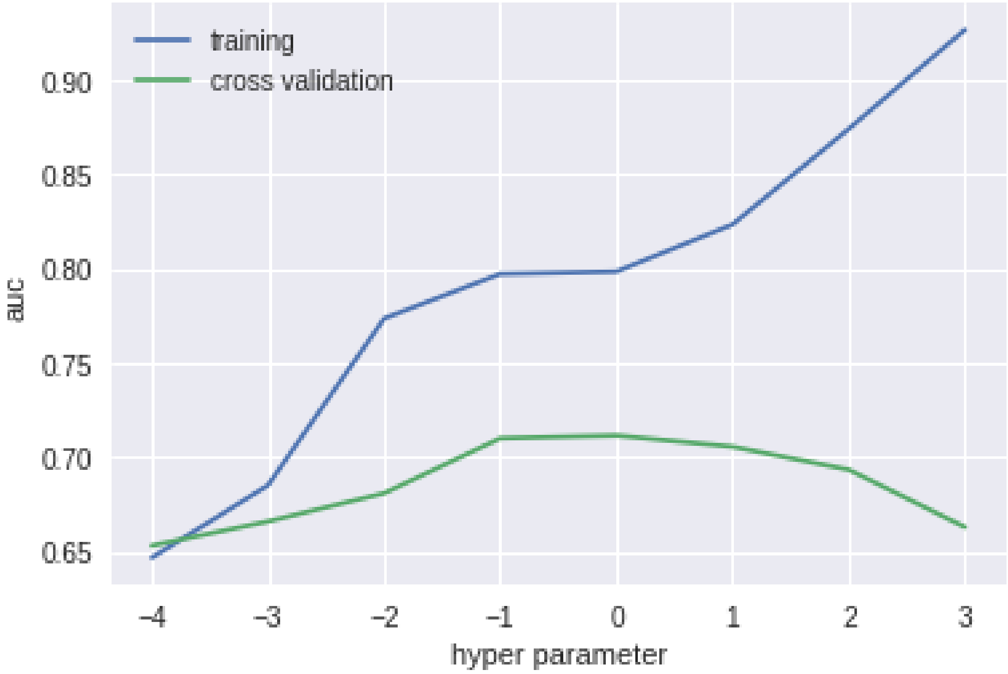


7.2.2 [5.2.2] Applying RBF SVM on TFIDF, SET 2

```
In [0]: #with hyper parameter tuning
        from sklearn.metrics import auc
        from sklearn.metrics import roc_auc_score
        from sklearn.neighbors import KNeighborsClassifier
        from sklearn.calibration import CalibratedClassifierCV
        from sklearn.svm import SVC
        cvscores=[]
        cvscores1=[]

        alpha=[10**i for i in range(-4,4,1)]
        for i in alpha:
            svcx=SVC(C=i,kernel='rbf')
            svcx.fit(xtraintfidfencoding11,ytrain1)
            model=CalibratedClassifierCV(svcx,method='sigmoid')
            model.fit(xtraintfidfencoding11,ytrain1)
            predict1=model.predict_proba(xtraintfidfencoding11)[:,1]
            cvscores.append(roc_auc_score(ytrain1,predict1))
            predict2=model.predict_proba(xcvtfidfencoding13)[:,1]
            cvscores1.append(roc_auc_score(ycv1,predict2))
            optimal_k=np.argmax(cvscores1)
        fig,ax=plt.subplots()
        ax.plot(np.log10(alpha),cvscores,label='training')
        ax.legend()
        ax.plot(np.log10(alpha),cvscores1,label='cross validation')
        ax.legend()
        plt.xlabel('hyper parameter')
        plt.ylabel('auc')
        plt.show()
        print(cvscores1)
        optimal_k=np.argmax(cvscores1)
        optimal_k=alpha[optimal_k]
        print(optimal_k)

        print(cvscores1)
```

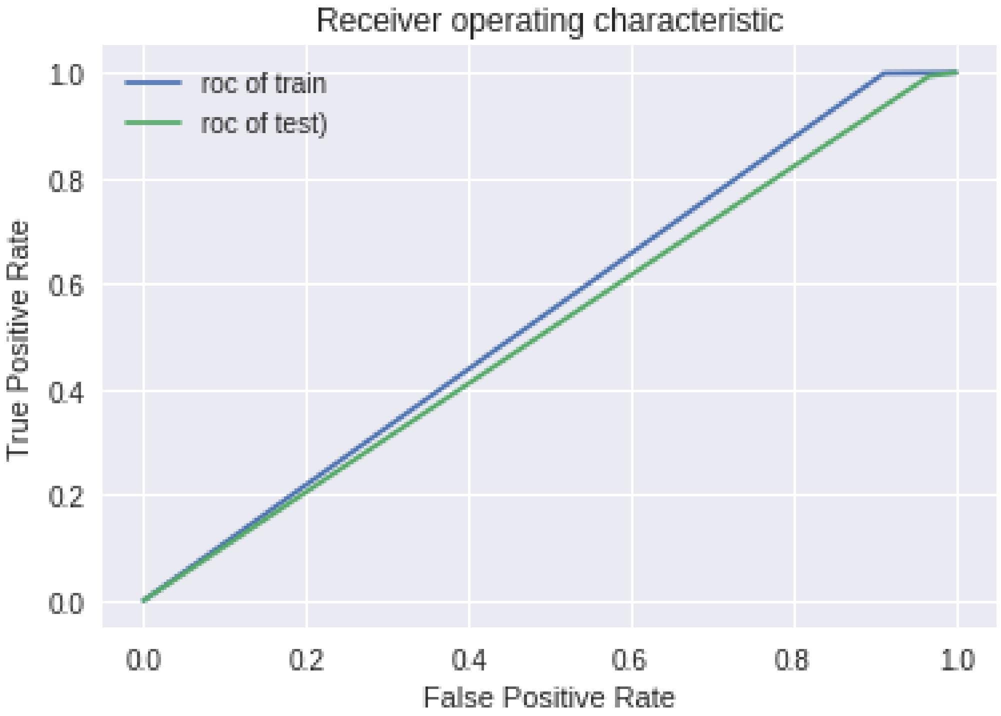


```
[0.6531204007507325, 0.6659113771080596, 0.6807918472610416, 0.7099156438611416, 0.7114034377706
1
[0.6531204007507325, 0.6659113771080596, 0.6807918472610416, 0.7099156438611416, 0.7114034377706
```

```
In [0]: from sklearn.neighbors import KNeighborsClassifier
        from sklearn.metrics import accuracy_score
        from sklearn.metrics import roc_auc_score
        svce=SVC(C=optimal_k,kernel='rbf')
        svce.fit(xtraintfidfencoding11,ytrain1)
        model=CalibratedClassifierCV(svce,method='sigmoid')
        model.fit(xtraintfidfencoding11,ytrain1)
```

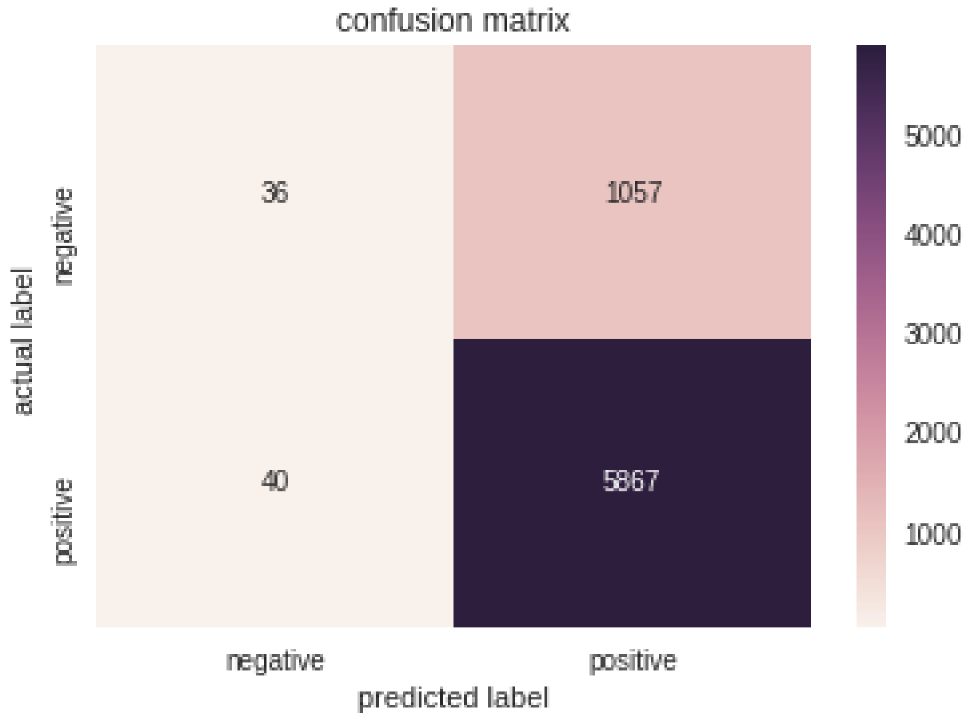
```
predictrain=model.predict(xtraintfidfencoding11)
fpr, tpr, thresh = metrics.roc_curve(ytrain1, predictrain)
auc = metrics.roc_auc_score(ytrain1, predictrain)
plt.plot(fpr,tpr,label="roc of train")
plt.legend()
predic=model.predict(xtesttfidfencoding12)
fpr, tpr, thresh = metrics.roc_curve(ytest1, predic)
auc = metrics.roc_auc_score(ytest1, predic)
plt.plot(fpr,tpr,label="roc of test)")
plt.legend()
plt.xlabel('False Positive Rate')
plt.ylabel('True Positive Rate')
plt.title('Receiver operating characteristic')
print(roc_auc_score(ytest1, predic))
```

```
0.5130826220569483
```



```
In [0]: #plotting confusion matrix aftyer performing knn on top of svd data
        from sklearn.metrics import confusion_matrix
        rest=confusion_matrix(ytest1,predic)
        import seaborn as sns
        classlabel=['negative','positive']
```

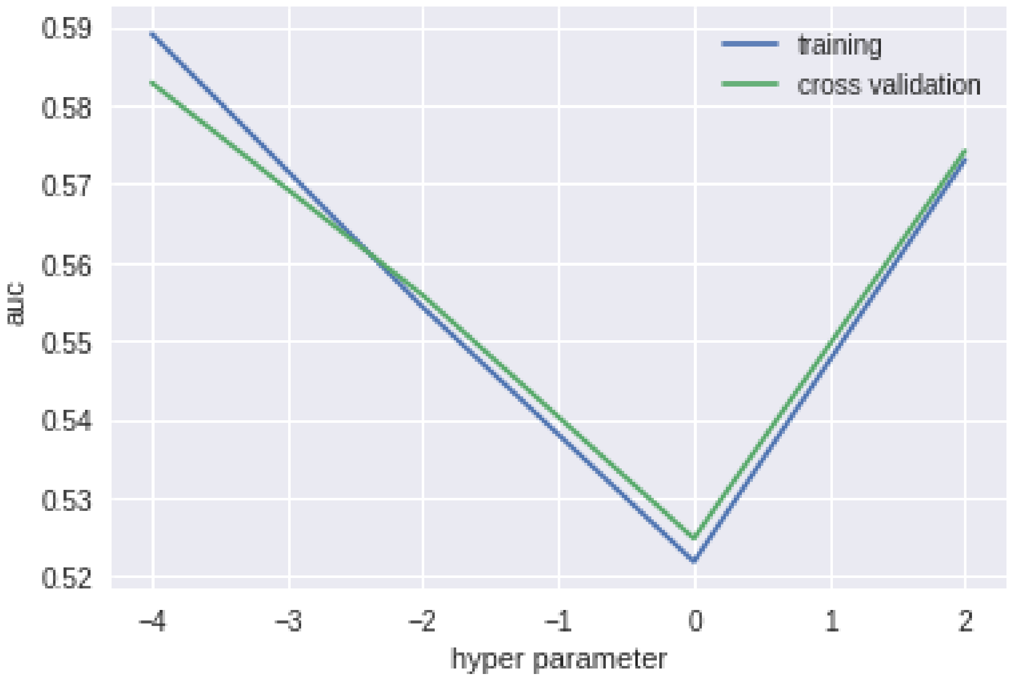
```
frame=pd.DataFrame(rest,index=classlabel,columns=classlabel)
sns.heatmap(frame,annot=True,fmt="d")
plt.title("confusion matrix")
plt.xlabel("predicted label")
plt.ylabel("actual label")
plt.show()
```



7.2.3 [5.2.3] Applying RBF SVM on AVG W2V, SET 3

```
In [60]: #with hyper parameter tuning
         from sklearn.metrics import auc
         from sklearn.metrics import roc_auc_score
         from sklearn.neighbors import KNeighborsClassifier
         cvscores=[]
         cvscores1=[]
         from sklearn.svm import SVC
         from sklearn.calibration import CalibratedClassifierCV
         alpha=[10**i for i in range(-4,4,2)]
         for i in alpha:
             svcx=SVC(C=i,kernel='rbf')
             svcx.fit(sent_vectors,ytrain)
             model=CalibratedClassifierCV(svcx,method='sigmoid')
             model.fit(sent_vectors,ytrain)
             predict1=model.predict_proba(sent_vectors)[:,1]
```

```
            cvscores.append(roc_auc_score(ytrain,predict1))
            predict2=model.predict_proba(sent_vectorscv)[:,1]
            cvscores1.append(roc_auc_score(ycv,predict2))
            optimal_k=np.argmax(cvscores1)
        fig,ax=plt.subplots()
        ax.plot(np.log10(alpha),cvscores,label='training')
        ax.legend()
        ax.plot(np.log10(alpha),cvscores1,label='cross validation')
        ax.legend()
        plt.xlabel('hyper parameter')
        plt.ylabel('auc')
        plt.show()
        print(cvscores1)
        optimal_k=np.argmax(cvscores1)
        optimal_k=alpha[optimal_k]
        print(optimal_k)
```

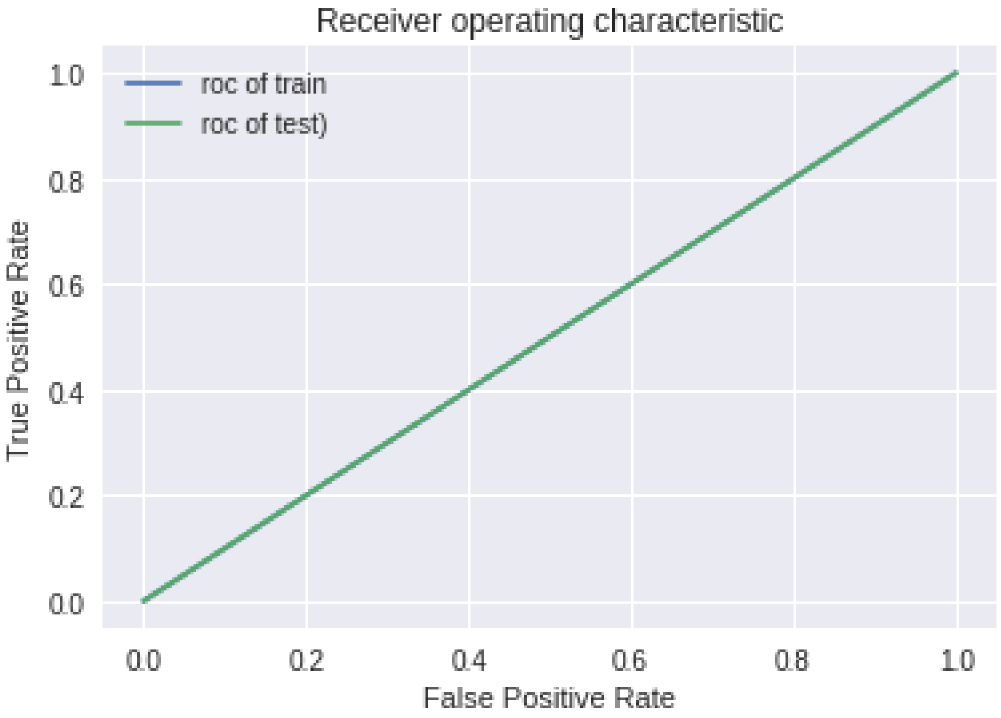


```
[0.5828309352784548, 0.5557763734358625, 0.5248100296690307, 0.5741589256901355]
0.0001
```

```
In [61]: from sklearn.neighbors import KNeighborsClassifier
         from sklearn.metrics import accuracy_score
         from sklearn.metrics import roc_auc_score
         svce=SVC(C=optimal_k,kernel='rbf')
```

```
svce.fit(sent_vectors,ytrain)
model=CalibratedClassifierCV(svce,method='sigmoid')
model.fit(sent_vectors,ytrain)
predictrain=model.predict(sent_vectors)
fpr, tpr, thresh = metrics.roc_curve(ytrain, predictrain)
auc = metrics.roc_auc_score(ytrain, predictrain)
plt.plot(fpr,tpr,label="roc of train")
plt.legend()
predic=model.predict(sent_vectorstest)
fpr, tpr, thresh = metrics.roc_curve(ytest, predic)
auc = metrics.roc_auc_score(ytest, predic)
plt.plot(fpr,tpr,label="roc of test)")
plt.legend()
plt.xlabel('False Positive Rate')
plt.ylabel('True Positive Rate')
plt.title('Receiver operating characteristic')
print(roc_auc_score(ytest, predic))
```

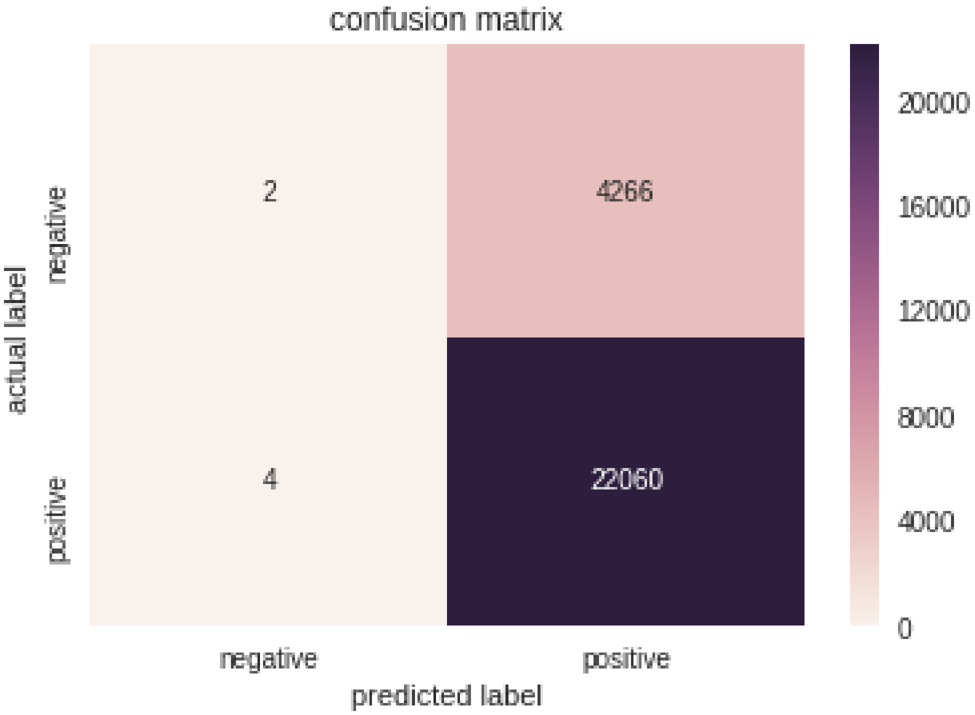
```
0.5001436563854796
```



```
In [62]: #plotting confusion matrix aftyer performing knn on top of svd data
         from sklearn.metrics import confusion_matrix
         rest=confusion_matrix(ytest,predic)
```

```
import seaborn as sns
classlabel=['negative','positive']

frame=pd.DataFrame(rest,index=classlabel,columns=classlabel)
sns.heatmap(frame,annot=True,fmt="d")
plt.title("confusion matrix")
plt.xlabel("predicted label")
plt.ylabel("actual label")
plt.show()
```



7.2.4 [5.2.4] Applying RBF SVM on TFIDF W2V, SET 4

```
In [0]: #with hyper parameter tuning
        from sklearn.metrics import auc
        from sklearn.metrics import roc_auc_score
        from sklearn.calibration import CalibratedClassifierCV
        from sklearn.svm import SVC
        cvscores=[]
        cvscores1=[]
        alpha=[10**i for i in range(-4,4,2)]
        for i in alpha:
            print('*')
            svcx=SVC(C=i,kernel='rbf')
            svcx.fit( xtraintfidf_sent_vectors,ytrain)
```

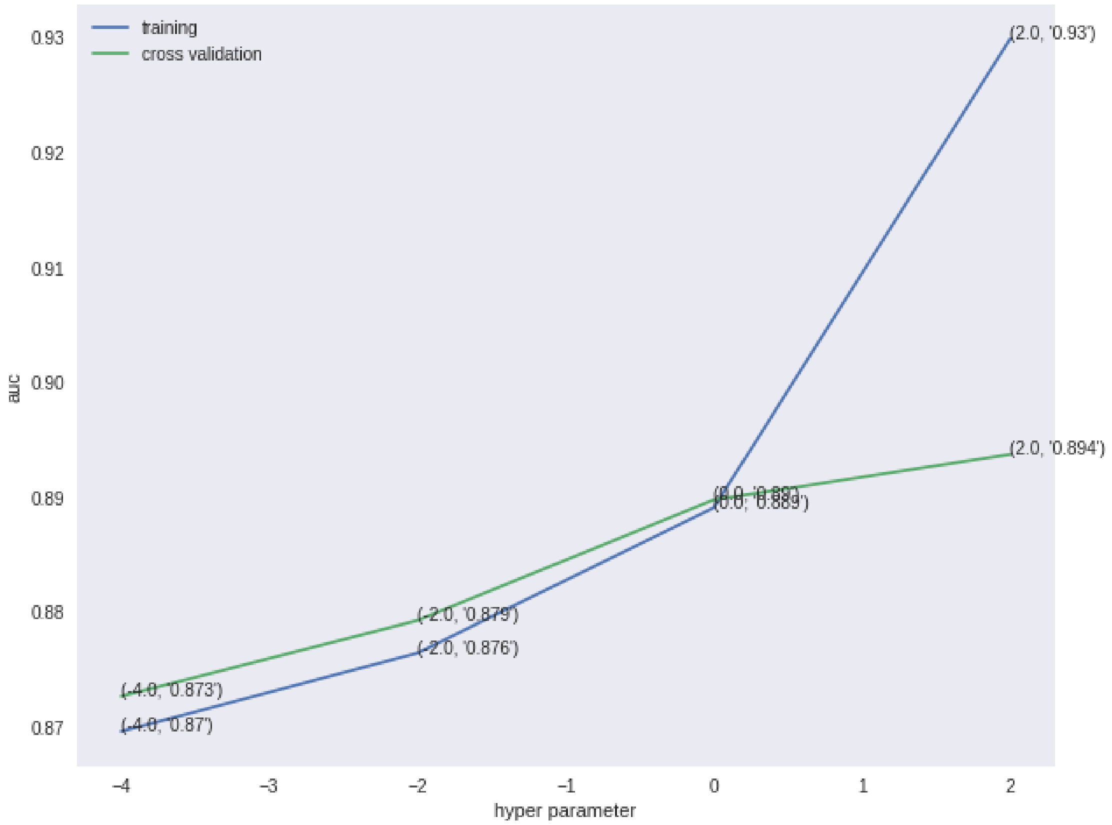
```
    model=CalibratedClassifierCV(svcx,method='sigmoid')
    model.fit(xtraintfidf_sent_vectors,ytrain)
    predict1=model.predict_proba( xtraintfidf_sent_vectors)[:,1]
    cvscores.append(roc_auc_score(ytrain,predict1))
    predict2=model.predict_proba( xcvtfidf_sent_vectors)[:,1]
    cvscores1.append(roc_auc_score(ycv,predict2))
optimal_k=np.argmax(cvscores1)
fig,ax=plt.subplots(figsize=(10,8))
ax.plot(np.log10(alpha),cvscores,label='training')
for i,txt in enumerate(np.round(cvscores,3)):
  ax.annotate((np.log10(alpha[i]),str(txt)), (np.log10(alpha[i]),cvscores[i]))
plt.grid()
ax.legend()
ax.plot(np.log10(alpha),cvscores1,label='cross validation')
for i,txt in enumerate(np.round(cvscores1,3)):
  ax.annotate((np.log10(alpha[i]),str(txt)), (np.log10(alpha[i]),cvscores1[i]))
ax.legend()
plt.xlabel('hyper parameter')
plt.ylabel('auc')
plt.show()
optimal_k=np.argmax(cvscores1)
man=alpha[optimal_k]
print(man)
print(cvscores1)
```

*

*

*

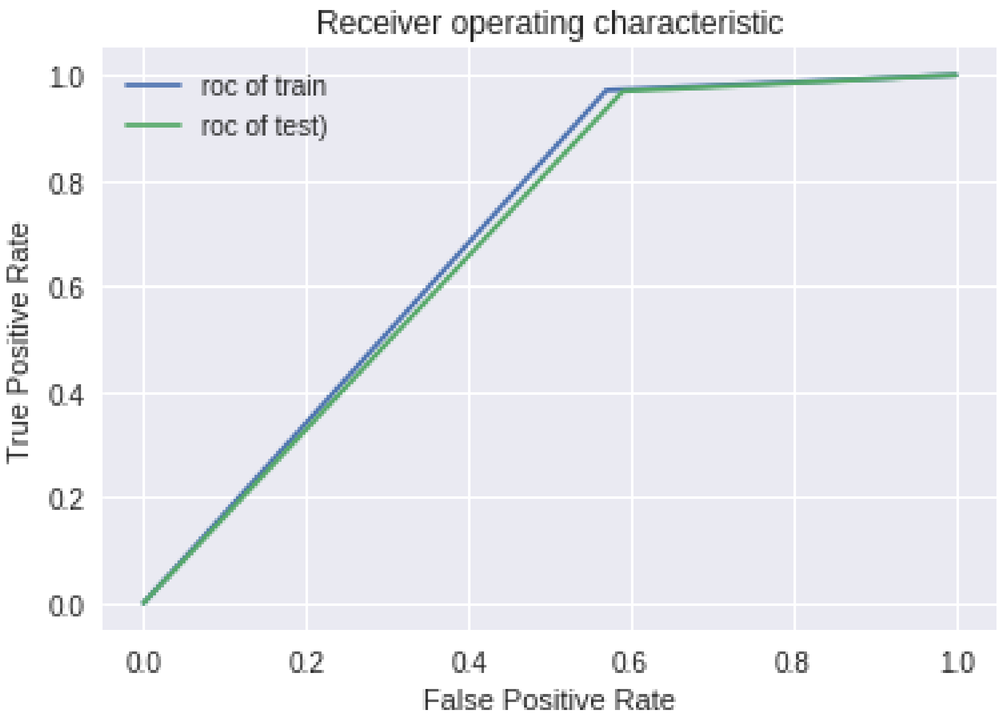
*



```
100
[0.8727287025152176, 0.8793181368769241, 0.8898152579239947, 0.8937687235296435]
```

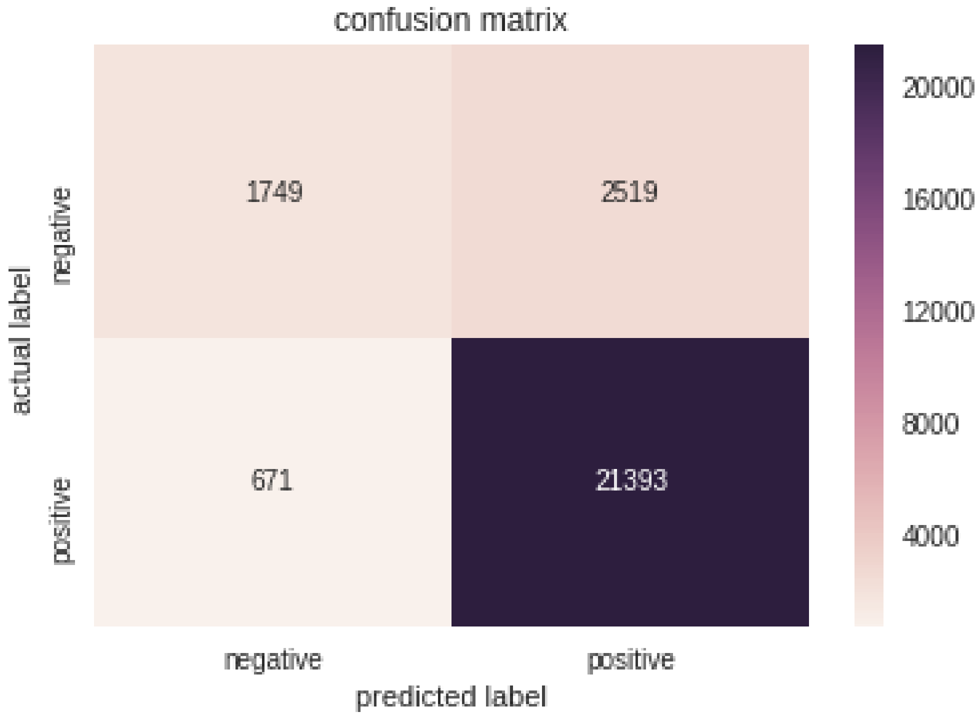
```
In [0]: from sklearn.metrics import accuracy_score
        from sklearn.metrics import roc_auc_score
        svce=SVC(C=optimal_k,kernel='rbf')
        svce.fit( xtraintfidf_sent_vectors,ytrain)
        model=CalibratedClassifierCV(svce,method='sigmoid')
        model.fit(xtraintfidf_sent_vectors,ytrain)
        predictrain=model.predict( xtraintfidf_sent_vectors)
        fpr, tpr, thresh = metrics.roc_curve(ytrain, predictrain)
        auc = metrics.roc_auc_score(ytrain, predictrain)
        plt.plot(fpr,tpr,label="roc of train")
        plt.legend()
        predic=model.predict( xtesttfidf_sent_vectors)
        fpr, tpr, thresh = metrics.roc_curve(ytest, predic)
        auc = metrics.roc_auc_score(ytest, predic)
        plt.plot(fpr,tpr,label="roc of test)")
        plt.legend()
        plt.xlabel('False Positive Rate')
        plt.ylabel('True Positive Rate')
        plt.title('Receiver operating characteristic')
        print(roc_auc_score(ytest, predic))
```

```
0.6896911421693592
```



```
In [0]: #plotting confusion matrix aftyer performing knn on top of svd data
        from sklearn.metrics import confusion_matrix
        rest=confusion_matrix(ytest,predic)
        import seaborn as sns
        classlabel=['negative','positive']

        frame=pd.DataFrame(rest,index=classlabel,columns=classlabel)
        sns.heatmap(frame,annot=True,fmt="d")
        plt.title("confusion matrix")
        plt.xlabel("predicted label")
        plt.ylabel("actual label")
        plt.show()
```



8 [6] Conclusions

```
In [86]: data = [['linear',9, 0.564], ['linear',7, 0.5],['linear',9, 0.54],['linear',9, 0.65],
         pd.DataFrame(data, columns=["model", "hyperparameter",'auc'],index=['bow','tfidf','wo
```

```
Out[86]:                        model  hyperparameter    auc
         bow                   linear               9  0.564
         tfidf                 linear               7  0.500
         word2vec              linear               9  0.540
         averageword2vec       linear               9  0.650
         bow               rbf_kernel               9  0.580
         tfidf             rbf_kernel               9  0.600
         word2vec          rbf_kernel               9  0.549
         averageword2vec   rbf_kernel               9  0.650
```