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.

[1]. Reading Data

[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 [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 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
```

```
D:\AAnaconda\lib\site-packages\gensim\utils.py:1212: UserWarning: detected Windows; aliasing chunkize t
o chunkize_serial
  warnings.warn("detected Windows; aliasing chunkize to chunkize_serial")
```

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 100000""", con)

# 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 (100000, 10)
```

Out[2]:

	Id	ProductId	UserId	ProfileName	HelpfulnessNumerator	HelpfulnessDenominator	Score	Ti
0	1	B001E4KFG0	A3SGXH7AUHU8GW	delmartian	1	1	1	1303862

	Id	ProductId	UserId	ProfileName	HelpfulnessNumerator	HelpfulnessDenominator	Score	Ti
1	2	B00813GRG4	A1D87F6ZCVE5NK	dll pa	0	0	0	1346976
2	3	B000LQOCH0	ABXLMWJIXXAIN	Natalia Corres "Natalia Corres"	1	1	1	1219017

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	Text	COUNT(*)
0	#oc-R115TNMSPFT9I7	B005ZBZLT4	Breyton	1331510400	2	Overall its just OK when considering the price...	2
1	#oc-R11D9D7SHXIJB9	B005HG9ESG	Louis E. Emory "hoppy"	1342396800	5	My wife has recurring extreme muscle spasms, u...	3
2	#oc-R11DNU2NBKQ23Z	B005ZBZLT4	Kim Cieszykowski	1348531200	1	This coffee is horrible and unfortunately not ...	2
3	#oc-R11O5J5ZVQE25C	B005HG9ESG	Penguin Chick	1346889600	5	This will be the bottle that you grab from the...	3
4	#oc-R12KPBODL2B5ZD	B007OSBEV0	Christopher P. Presta	1348617600	1	I didnt like this coffee. Instead of telling y...	2

In [5]:

```
display[display['UserId']=='AZY10LLTJ71NX']
```

Out[5]:

	UserId	ProductId	ProfileName	Time	Score	Text	COUNT(*)
80638	AZY10LLTJ71NX	B001ATMQK2	undertheshrine "undertheshrine"	1296691200	5	I bought this 6 pack because for the price tha...	5

In [6]:

```
display['COUNT(*)'].sum()
```

Out[6]:

```
393063
```

[2] Exploratory Data Analysis

[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	HelpfulnessDenominator	Score	
0	78445	B000HDL1RQ	AR5J8UI46CURR	Geetha Krishnan	2	2	5	11995
1	138317	B000HDOPYC	AR5J8UI46CURR	Geetha Krishnan	2	2	5	11995
2	138277	B000HDOPYM	AR5J8UI46CURR	Geetha Krishnan	2	2	5	11995
3	73791	B000HDOPZG	AR5J8UI46CURR	Geetha Krishnan	2	2	5	11995
4	155049	B000PAQ75C	AR5J8UI46CURR	Geetha Krishnan	2	2	5	11995

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 [8]:

```
#Sorting data according to ProductId in ascending order
sorted_data=filtered_data.sort_values('ProductId', axis=0, ascending=True, inplace=False, kind='quickso
rt', na_position='last')
```

In [9]:

```
#Deduplication of entries
final=sorted_data.drop_duplicates(subset={"UserId","ProfileName","Time","Text"}, keep='first', inplace=
False)
final.shape
```

Out[9]:

```
(87775, 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]:

```
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 [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	HelpfulnessNumerator	HelpfulnessDenominator	Score	
0	64422	B000MIDROQ	A161DK06JJMCYF	J. E. Stephens "Jeanne"	3	1	5	12248
1	44737	B001EQ55RW	A2V0I904FH7ABY	Ram	3	2	4	12128

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()
```

```
(87773, 10)
```

Out[13]:

```
1    73592
0    14181
Name: Score, dtype: int64
```

[3] Preprocessing

[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 [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)
```

```
My dogs loves this chicken but its a product from China, so we wont be buying it anymore.  Its very har
d to find any chicken products made in the USA but they are out there, but this one isnt.  Its too bad
too because its a good product but I wont take any chances till they know what is going on with the chi
na imports.
==================================================
The Candy Blocks were a nice visual for the Lego Birthday party but the candy has little taste to it.
Very little of the 2 lbs that I bought were eaten and I threw the rest away.  I would not buy the candy
again.
==================================================
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 of the
fishy smell, don't get it. But I think my dog likes it because of the smell. These treats are really sm
all in size. They are great for training. You can give your dog several of these without worrying about
him over eating. Amazon's price was much more reasonable than any other retailer. You can buy a 1 pound
bag on Amazon for almost the same price as a 6 ounce bag at other retailers. It's definitely worth it t
o buy a big bag if your dog eats them a lot.
==================================================
```

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)
```

```
My dogs loves this chicken but its a product from China, so we wont be buying it anymore.  Its very har
d to find any chicken products made in the USA but they are out there, but this one isnt.  Its too bad
too because its a good product but I wont take any chances till they know what is going on with the chi
na imports.
```

In [16]:

```
# https://stackoverflow.com/questions/16206380/python-beautifulsoup-how-to-remove-all-tags-from-an-elem
ent
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 very har
d to find any chicken products made in the USA but they are out there, but this one isnt.  Its too bad
too because its a good product but I wont take any chances till they know what is going on with the chi
na imports.
==================================================
The Candy Blocks were a nice visual for the Lego Birthday party but the candy has little taste to it.
Very little of the 2 lbs that I bought were eaten and I threw the rest away.  I would not buy the candy
again.
==================================================
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 of the
fishy smell, don't get it. But I think my dog likes it because of the smell. These treats are really sm
all in size. They are great for training. You can give your dog several of these without worrying about
him over eating. Amazon's price was much more reasonable than any other retailer. You can buy a 1 pound
bag on Amazon for almost the same price as a 6 ounce bag at other retailers. It's definitely worth it t
o buy a big bag if your dog eats them a lot.
```

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)
```

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

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)
```

```
My dogs loves this chicken but its a product from China, so we wont be buying it anymore.  Its very har
d to find any chicken products made in the USA but they are out there, but this one isnt.  Its too bad
too because its a good product but I wont take any chances till they know what is going on with the chi
na imports.
```

In [20]:

```
#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 [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 revmoved in the 1st step

stopwords= set(['br', 'the', 'i', 'me', 'my', 'myself', 'we', 'our', 'ours', 'ourselves', 'you', "you'r
e", "you've",\
            "you'll", "you'd", 'your', 'yours', 'yourself', 'yourselves', 'he', 'him', 'his', 'himself'
, \
            'she', "she's", 'her', 'hers', 'herself', 'it', "it's", 'its', 'itself', 'they', 'them', 't
heir',\
            'theirs', 'themselves', 'what', 'which', 'who', 'whom', 'this', 'that', "that'll", 'these',
'those', \
            'am', 'is', 'are', 'was', 'were', 'be', 'been', 'being', 'have', 'has', 'had', 'having', 'd
o', 'does', \
            'did', 'doing', 'a', 'an', 'the', 'and', 'but', 'if', 'or', 'because', 'as', 'until', 'whil
e', 'of', \
            'at', 'by', 'for', 'with', 'about', 'against', 'between', 'into', 'through', 'during', 'bef
ore', 'after',\
            'above', 'below', 'to', 'from', 'up', 'down', 'in', 'out', 'on', 'off', 'over', 'under', 'a
gain', 'further',\
            'then', 'once', 'here', 'there', 'when', 'where', 'why', 'how', 'all', 'any', 'both', 'each
', 'few', 'more',\
            'most', 'other', 'some', 'such', 'only', 'own', 'same', 'so', 'than', 'too', 'very', \
            's', 't', 'can', 'will', 'just', 'don', "don't", 'should', "should've", 'now', 'd', 'll', '
m', 'o', 're', \
            've', 'y', 'ain', 'aren', "aren't", 'couldn', "couldn't", 'didn', "didn't", 'doesn', "doesn
't", 'hadn',\
            "hadn't", 'hasn', "hasn't", 'haven', "haven't", 'isn', "isn't", 'ma', 'mightn', "mightn't",
'mustn',\
            "mustn't", 'needn', "needn't", 'shan', "shan't", 'shouldn', "shouldn't", 'wasn', "wasn't",
'weren', "weren'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 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 stopwords)
    preprocessed_reviews.append(sentance.strip())
```

```
100%|██████████| 87773/87773 [00:48<00:00, 1812.89it/s]
```

In [23]:

```
final['CleanedText'] = preprocessed_reviews
y = final['Score']
```

In [24]:

```
preprocessed_reviews[1500]
```

Out[24]:

```
'way hot blood took bite jig lol'
```

[5] Assignment 7: SVM

1. **Apply SVM on these feature sets**

    - SET 1:Review text, preprocessed one converted into vectors using (BOW)
    - SET 2:Review text, preprocessed one converted into vectors using (TFIDF)
    - SET 3:Review text, preprocessed one converted into vectors using (AVG W2v)
    - SET 4:Review text, preprocessed one converted into vectors using (TFIDF W2v)

2. **Procedure**

    - You need to work with 2 versions of SVM
        - Linear kernel
        - RBF kernel
    - When you are working with linear kernel, use SGDClassifier' with hinge loss because it is computationally less expensive.
    - When you are working with 'SGDClassifier' with hinge loss and trying to find the AUC score, you would have to use CalibratedClassifierCV
    - 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.

3. **Hyper paramter tuning (find best alpha in range [10^-4 to 10^4], and the best penalty among 'l1', 'l2')**

    - Find the best hyper parameter which will give the maximum AUC value
    - Find the best hyper paramter using k-fold cross validation or simple cross validation data
    - Use gridsearch cv or randomsearch cv or you can also write your own for loops to do this task of hyperparameter tuning

4. **Feature importance**

    - 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.

5. **Feature engineering**

- To increase the performance of your model, you can also experiment with with feature engineering like :
  - Taking length of reviews as another feature.
  - Considering some features from review summary as well.

6. **Representation of results**

- You need to plot the performance of model both on train data and cross validation data for each hyper parameter, like shown in the figure.
- Once after you found the best hyper parameter, you need to train your model with it, and find the AUC on test data and plot the ROC curve on both train and test.
- Along with plotting ROC curve, you need to print the confusion matrix with predicted and original labels of test data points. Please visualize your confusion matrices using seaborn heatmaps.

7. **Conclusion**

- You need to summarize the results at the end of the notebook, summarize it in the table format. To print out a table please refer to this prettytable library link

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.

In [25]:

```
#Splitting data into train and test:

from sklearn.model_selection import train_test_split
from sklearn.metrics import accuracy_score
from sklearn.model_selection import cross_val_score
from collections import Counter
from sklearn.metrics import accuracy_score
from sklearn import model_selection

X_train, X_test, y_train, y_test = train_test_split(final, y, test_size=0.2)
print(X_train.shape, y_train.shape)
print(X_test.shape, y_test.shape)

#Splitting train data into train and cv(60:20)
X_tr, X_cv, y_tr, y_cv = train_test_split(X_train, y_train, test_size=0.2)
print(X_tr.shape, y_tr.shape)
print(X_cv.shape, y_cv.shape)
```

```
(70218, 11) (70218,)
(17555, 11) (17555,)
(56174, 11) (56174,)
(14044, 11) (14044,)
```

In [26]:

```
#Applying BoW
model = CountVectorizer()
model.fit(X_tr['CleanedText'])
train_bow = model.transform(X_tr['CleanedText'])
cv_bow = model.transform(X_cv['CleanedText'])
test_bow = model.transform(X_test['CleanedText'])
print(test_bow.shape)
print(cv_bow.shape)
print(train_bow.shape)
```

```
(17555, 44245)
(14044, 44245)
(56174, 44245)
```

In [27]:

```
#Applying tf_idf vectorization
tf_idf_vect = TfidfVectorizer(ngram_range=(1,2))
tf_idf_vect.fit(X_tr['Text'])
train_tf_idf = tf_idf_vect.transform(X_tr['Text'])
test_tf_idf = tf_idf_vect.transform(X_test['Text'])
cv_tf_idf = tf_idf_vect.transform(X_cv['Text'])

print(test_tf_idf.shape)
print(train_tf_idf.shape)
print(cv_tf_idf.shape)
```

```
(17555, 859723)
(56174, 859723)
(14044, 859723)
```

In [28]:

```
# Word2Vec model for train/test and cv dataset
i=0
list_of_sent=[]
for sent in X_tr['CleanedText'].values:
    list_of_sent.append(sent.split())

print(X_tr['CleanedText'].values[0])
print("******************************************************************")
print(list_of_sent[0])


# Word2Vec model for test and CV
i=0
list_of_sent_cv=[]
for sent in X_cv['CleanedText'].values:
    list_of_sent_cv.append(sent.split())

print(X_cv['CleanedText'].values[0])
print("******************************************************************")
print(list_of_sent_cv[0])


i=0
list_of_sent_test=[]
for sent in X_test['CleanedText'].values:
    list_of_sent_test.append(sent.split())

print(X_test['CleanedText'].values[0])
print("******************************************************************")
print(list_of_sent_test[0])


w2v_model_train=Word2Vec(list_of_sent,min_count=5,size=50, workers=5)
w2v_model_test=Word2Vec(list_of_sent_test,min_count=5,size=50, workers=5)
w2v_model_cv=Word2Vec(list_of_sent_cv,min_count=5,size=50, workers=5)


w2v_words = list(w2v_model_train.wv.vocab)
print("number of words that occured minimum 5 times ",len(w2v_words))
print("sample words ", w2v_words[0:50])


# average Word2Vec
# compute average word2vec for each review Train dataset
sent_vectors = []; # the avg-w2v for each sentence/review is stored in this list
for sent in tqdm(list_of_sent): # for each review/sentence
    sent_vec = np.zeros(50) # as word vectors are of zero length
    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_train.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))
print(len(sent_vectors[0]))


# average Word2Vec
# compute average word2vec for each review - test dataset
sent_vectors_test = []; # the avg-w2v for each sentence/review is stored in this list
for sent in tqdm(list_of_sent_test): # for each review/sentence
    sent_vec_test = np.zeros(50) # as word vectors are of zero length
    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_train.wv[word]
            sent_vec += vec
            cnt_words += 1
    if cnt_words != 0:
        sent_vec /= cnt_words
    sent_vectors_test.append(sent_vec)
print(len(sent_vectors_test))
print(len(sent_vectors_test[0]))


# average Word2Vec
# compute average word2vec for each review - cv dataset
sent_vectors_cv = []; # the avg-w2v for each sentence/review is stored in this list
for sent in tqdm(list_of_sent_cv): # for each review/sentence
    sent_vec_cv = np.zeros(50) # as word vectors are of zero length
    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_train.wv[word]
            sent_vec += vec
            cnt_words += 1
    if cnt_words != 0:
        sent_vec /= cnt_words
    sent_vectors_cv.append(sent_vec)
print(len(sent_vectors_cv))
print(len(sent_vectors_cv[0]))
```

```
tryed lot beef jerky must say greatest eaten try say finger licking good nt chew like piece leather eit
her bettie ohio
*****************************************************************
['tryed', 'lot', 'beef', 'jerky', 'must', 'say', 'greatest', 'eaten', 'try', 'say', 'finger', 'licking'
, 'good', 'nt', 'chew', 'like', 'piece', 'leather', 'either', 'bettie', 'ohio']
kids allergic eggs nuts dairy two also picky eaters cookie mix easy prepare tastes wonderful work excep
tionally well rolled cut cookies always hand major holidays beware go quickly
*****************************************************************
['kids', 'allergic', 'eggs', 'nuts', 'dairy', 'two', 'also', 'picky', 'eaters', 'cookie', 'mix', 'easy'
, 'prepare', 'tastes', 'wonderful', 'work', 'exceptionally', 'well', 'rolled', 'cut', 'cookies', 'alway
s', 'hand', 'major', 'holidays', 'beware', 'go', 'quickly']
think grape one next day woke find roommate ate rest entire box guess pretty good
*****************************************************************
['think', 'grape', 'one', 'next', 'day', 'woke', 'find', 'roommate', 'ate', 'rest', 'entire', 'box', 'g
uess', 'pretty', 'good']
number of words that occured minimum 5 times  14170
sample words  ['tryed', 'lot', 'beef', 'jerky', 'must', 'say', 'greatest', 'eaten', 'try', 'finger', 'l
icking', 'good', 'nt', 'chew', 'like', 'piece', 'leather', 'either', 'ohio', 'person', 'never', 'drank'
, 'tea', 'tried', 'many', 'different', 'teas', 'liked', 'also', 'moderate', 'stomach', 'issues', 'thoug
ht', 'would', 'admit', 'took', 'couple', 'cups', 'get', 'hooked', 'found', 'taste', 'wonderful', 'calmi
ng', 'effect', 'big', 'help', 'drink', 'one', 'cup']
```

```
100%|██████████| 56174/56174 [02:59<00:00, 312.46it/s]
```

```
56174
50
```

```
100%|██████████| 17555/17555 [00:58<00:00, 301.09it/s]
```

```
17555
50
```

```
100%|██████████| 14044/14044 [00:47<00:00, 293.35it/s]
```

```
14044
50
```

In [29]:

```python
# S = ["abc def pqr", "def def def abc", "pqr pqr def"]
model = TfidfVectorizer()
tf_idf_matrix = model.fit_transform(X_tr['CleanedText'].values)
# 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 [30]:

```python
# TF-IDF weighted Word2Vec for train dataset
tfidf_feat = model.get_feature_names() # tfidf words/col-names
# final_tf_idf is the sparse matrix with row= sentence, col=word and cell_val = tfidf

tfidf_sent_vectors = []; # the tfidf-w2v for each sentence/review is stored in this list
row=0;
for sent in tqdm(list_of_sent): # 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_train.wv[word]
#             tf_idf = tf_idf_matrix[row, tfidf_feat.index(word)]
            # to reduce the computation we are
            # dictionary[word] = idf value of word in whole courpus
            # sent.count(word) = tf valeus of word in this review
            tf_idf = dictionary[word]*(sent.count(word)/len(sent))
            sent_vec += (vec * tf_idf)
            weight_sum += tf_idf
    if weight_sum != 0:
        sent_vec /= weight_sum
    tfidf_sent_vectors.append(sent_vec)
    row += 1
```

```
100%|██████████| 56174/56174 [47:47<00:00, 19.59it/s]
```

In [31]:

```python
# TF-IDF weighted Word2Vec for test dataset
tfidf_feat = model.get_feature_names() # tfidf words/col-names
# final_tf_idf is the sparse matrix with row= sentence, col=word and cell_val = tfidf

tfidf_sent_vectors_test = []; # the tfidf-w2v for each sentence/review is stored in this list
row=0;
for sent in tqdm(list_of_sent_test): # 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_train.wv[word]
#             tf_idf = tf_idf_matrix[row, tfidf_feat.index(word)]
            # to reduce the computation we are
            # dictionary[word] = idf value of word in whole courpus
            # sent.count(word) = tf valeus of word in this review
            tf_idf = dictionary[word]*(sent.count(word)/len(sent))
            sent_vec += (vec * tf_idf)
            weight_sum += tf_idf
    if weight_sum != 0:
        sent_vec /= weight_sum
    tfidf_sent_vectors_test.append(sent_vec)
    row += 1


# TF-IDF weighted Word2Vec for cv dataset
tfidf_feat = model.get_feature_names() # tfidf words/col-names
# final_tf_idf is the sparse matrix with row= sentence, col=word and cell_val = tfidf

tfidf_sent_vectors_cv = []; # the tfidf-w2v for each sentence/review is stored in this list
row=0;
for sent in tqdm(list_of_sent_cv): # for each review/sentence
```

```
for sent in tqdm(list_of_sent_cv): # 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_train.wv[word]
#             tf_idf = tf_idf_matrix[row, tfidf_feat.index(word)]
            # to reduce the computation we are
            # dictionary[word] = idf value of word in whole courpus
            # sent.count(word) = tf valeus of word in this review
            tf_idf = dictionary[word]*(sent.count(word)/len(sent))
            sent_vec += (vec * tf_idf)
            weight_sum += tf_idf
    if weight_sum != 0:
        sent_vec /= weight_sum
    tfidf_sent_vectors_cv.append(sent_vec)
    row += 1
```

```
100%|██████████| 17555/17555 [12:22<00:00, 28.21it/s]
100%|██████████| 14044/14044 [09:33<00:00, 24.48it/s]
```

Applying SVM

[5.1] Linear SVM

[5.1.1] Applying Linear SVM on BOW, SET 1

In [32]:

```
from sklearn import svm
from sklearn.model_selection import GridSearchCV
from sklearn.svm import LinearSVC
from sklearn import linear_model
from sklearn.calibration import CalibratedClassifierCV,calibration_curve
alphas = np.array([0.00001, 0.0001, 0.001, 0.01, 1, 10, 100, 1000])
base_estimator = linear_model.SGDClassifier(loss='hinge',penalty='l1', random_state=0, class_weight='ba
lanced')
scoring = {'AUC': 'roc_auc'}
grid = GridSearchCV(estimator=base_estimator,param_grid=dict(alpha=alphas),scoring = scoring, refit = '
AUC')
grid.fit(train_bow, y_tr)
print(grid)
# summarize the results of the grid search
(grid.best_score_)
print(grid.best_estimator_.alpha)
results_tr_bow = grid.cv_results_
#print(results_bow)
```

```
GridSearchCV(cv='warn', error_score='raise-deprecating',
       estimator=SGDClassifier(alpha=0.0001, average=False, class_weight='balanced',
       early_stopping=False, epsilon=0.1, eta0=0.0, fit_intercept=True,
       l1_ratio=0.15, learning_rate='optimal', loss='hinge', max_iter=None,
       n_iter=None, n_iter_no_change=5, n_jobs=None, penalty='l1',
       power_t=0.5, random_state=0, shuffle=True, tol=None,
       validation_fraction=0.1, verbose=0, warm_start=False),
       fit_params=None, iid='warn', n_jobs=None,
       param_grid={'alpha': array([1.e-05, 1.e-04, 1.e-03, 1.e-02, 1.e+00, 1.e+01, 1.e+02, 1.e+03])},
       pre_dispatch='2*n_jobs', refit='AUC', return_train_score='warn',
       scoring={'AUC': 'roc_auc'}, verbose=0)
1e-05
```

In [33]:

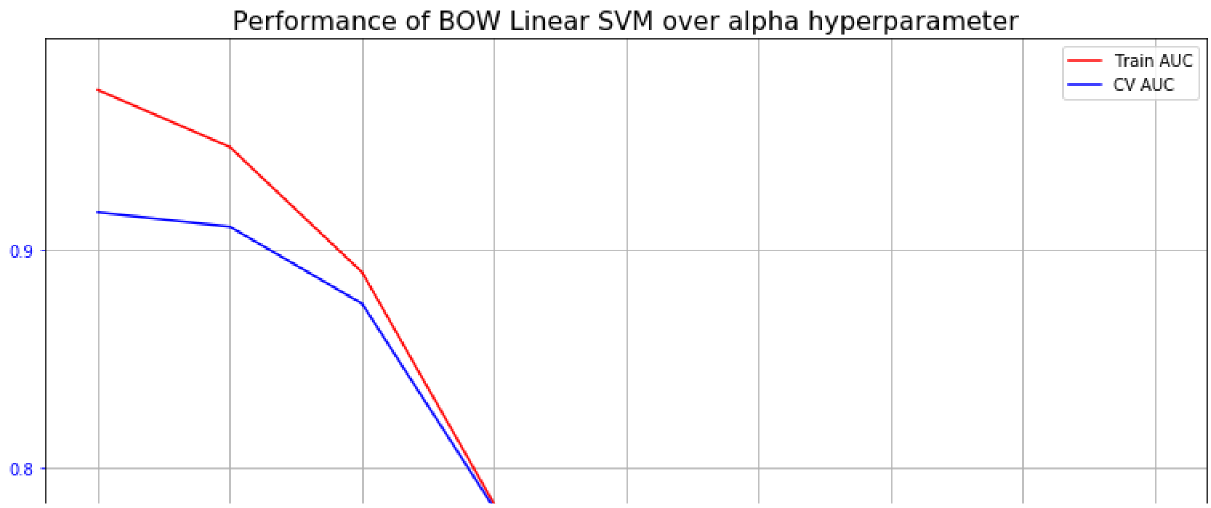
```
alphas = np.array([0.00001, 0.0001, 0.001, 0.01, 1, 10, 100, 1000])
base_estimator = linear_model.SGDClassifier(loss='hinge',penalty='l1', random_state=0, class_weight='ba
lanced')
scoring = {'AUC': 'roc_auc'}
```

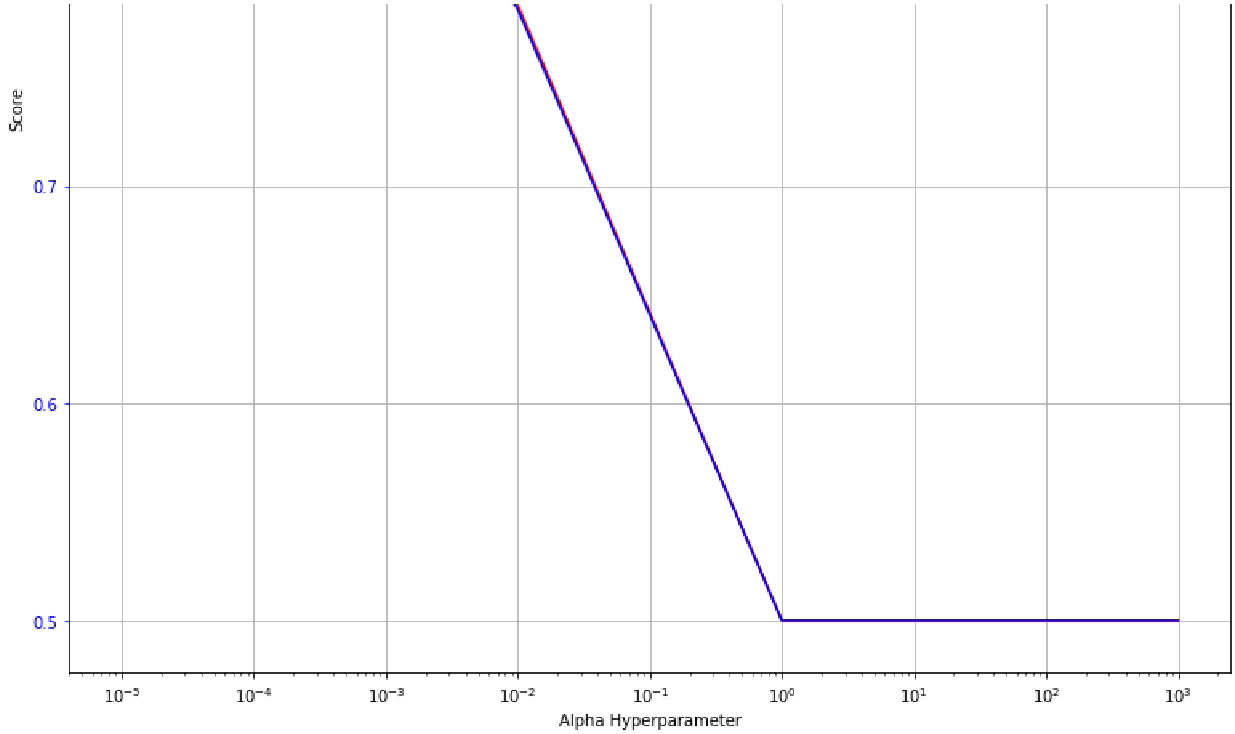
```
grid = GridSearchCV(estimator=base_estimator,param_grid=dict(alpha=alphas),scoring = scoring, refit = '
AUC')
grid.fit(cv_bow, y_cv)
print(grid)
# summarize the results of the grid search
(grid.best_score_)
print(grid.best_estimator_.alpha)
results_cv_bow = grid.cv_results_
#print(results_bow)
```

```
GridSearchCV(cv='warn', error_score='raise-deprecating',
       estimator=SGDClassifier(alpha=0.0001, average=False, class_weight='balanced',
       early_stopping=False, epsilon=0.1, eta0=0.0, fit_intercept=True,
       l1_ratio=0.15, learning_rate='optimal', loss='hinge', max_iter=None,
       n_iter=None, n_iter_no_change=5, n_jobs=None, penalty='l1',
       power_t=0.5, random_state=0, shuffle=True, tol=None,
       validation_fraction=0.1, verbose=0, warm_start=False),
       fit_params=None, iid='warn', n_jobs=None,
       param_grid={'alpha': array([1.e-05, 1.e-04, 1.e-03, 1.e-02, 1.e+00, 1.e+01, 1.e+02, 1.e+03])},
       pre_dispatch='2*n_jobs', refit='AUC', return_train_score='warn',
       scoring={'AUC': 'roc_auc'}, verbose=0)
1e-05
```

In [34]:

```
#Performance over alpha hyperparameter
plt.figure(figsize=(13, 13))
plt.title("Performance of BOW Linear SVM over alpha hyperparameter",
          fontsize=16)

#ax.set_xlim(0, 402)
#ax.set_ylim(0.73, 1)

# Get the regular numpy array from the MaskedArray
X_axis = np.array(results_tr_bow['param_alpha'].data, dtype=float)
Y_axis_train = results_tr_bow['mean_train_AUC']
Y_axis_CV = results_tr_bow['mean_test_AUC']
#Y_axis = np.array(sorted(results_bow['mean_train_AUC']).data, dtype=float)
#fig, ax = plt.subplots()
ax = plt.gca()
ax.set_xscale('log')
#ax=plt.subplots()
curve1, = ax.plot(X_axis, Y_axis_train, label="Train AUC", color='r')
curve2, = ax.plot(X_axis, Y_axis_CV, label="CV AUC", color='b')
curves = [curve1, curve2]
ax.legend()
ax.set_ylabel("Score")
ax.set_xlabel("Alpha Hyperparameter")
ax.tick_params(axis='y', colors=curve1.get_color())
ax.tick_params(axis='y', colors=curve2.get_color())
#ax.plot(X_axis,Y_axis_CV)
plt.legend(loc="best")
plt.grid()
plt.show()
```





Observation:

```
The graph shows that for best alpha value of 1e-05, AUC score for test data is 0.91.
As alpha increases, AUC score decreases and reaches a value of 0.5.
```

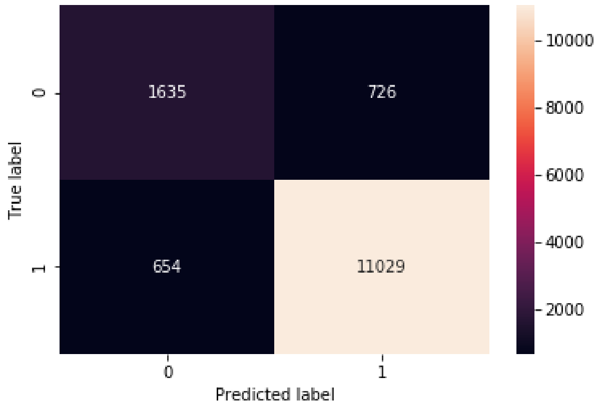
In [35]:

```
# After finding the best hyperparameter value for BOW, applying SVM on train dataset and predicting
# accuracy/AUC score for cv dataset
from sklearn import svm
from sklearn.model_selection import GridSearchCV
from sklearn.svm import LinearSVC
from sklearn import linear_model
from sklearn.calibration import CalibratedClassifierCV,calibration_curve
clf = linear_model.SGDClassifier(alpha = 1e-05,loss='hinge',penalty='l1', random_state=0, class_weight=
'balanced')
scoring = {'AUC': 'roc_auc'}
clf.fit(train_bow,y_tr)

#Caliberate the classifier.
clf_calibrated=CalibratedClassifierCV(clf, cv='prefit', method='isotonic')
cclf=clf_calibrated.fit(train_bow, y_tr).predict(cv_bow)
pred_cv = clf_calibrated.predict_proba(cv_bow)[:,1]
#pred_cv = np.argmax(log_pred,axis = 1)
print('alpha value = ',1e-05)
fpr, tpr, thresholds = roc_curve(y_cv,pred_cv)
roc_auc_cv = auc(fpr, tpr)
print('Area under the ROC curve : %f', + roc_auc_cv)

#Plotting confusion matrix
import seaborn as sns
conf_mat = confusion_matrix(y_cv, cclf)
print(conf_mat)
#conf_normalized = conf_mat.astype('int') / conf_mat.sum(axis=1)[:, np.newaxis]
sns.heatmap(conf_mat, annot=True, fmt ='g')
plt.ylabel('True label')
plt.xlabel('Predicted label')
```

```
alpha value =  1e-05
Area under the ROC curve : %f 0.9219105233069419
[[ 1635   726]
 [  654 11029]]
```

Out[35]:

```
Text(0.5, 15.0, 'Predicted label')
```



Observation for cv dataset:

1. Out of 2272 negative data points, BOW predicts 1421 correctly and the remaining 851 incorrect
.
2. Similarly, out of 11772 positive data points, 446 are misclassified.
3. AUC score is above 0.5

In [36]:

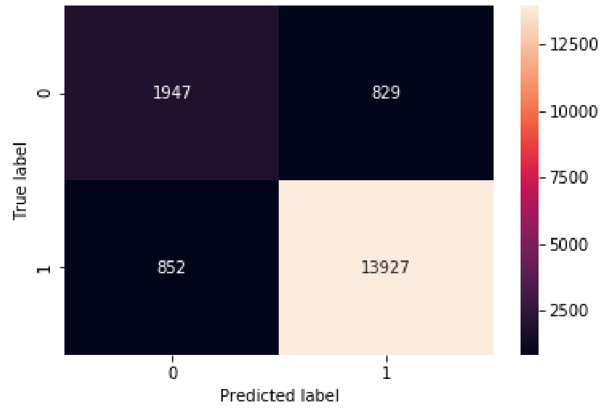
```
# After finding the best hyperparameter value for BOW which is 1e-05, applying linear SVM on train data
set and predicting
# accuracy/AUC score for cv dataset
from sklearn import svm
from sklearn.model_selection import GridSearchCV
from sklearn.svm import LinearSVC
from sklearn import linear_model
from sklearn.calibration import CalibratedClassifierCV,calibration_curve
clf = linear_model.SGDClassifier(alpha = 1e-05,loss='hinge',penalty='l1', random_state=0,class_weight =
"balanced")
scoring = {'AUC': 'roc_auc'}
clf.fit(train_bow,y_tr)

#Caliberate the classifier.
clf_calibrated=CalibratedClassifierCV(clf, cv='prefit', method='isotonic')
cclf = clf_calibrated.fit(train_bow, y_tr).predict(test_bow)
pred_test = clf_calibrated.predict_proba(test_bow)[:,1]
print('alpha value = ',1e-05)
fpr, tpr, thresholds = roc_curve(y_test,pred_test)
roc_auc_test = auc(fpr, tpr)
print('Area under the ROC curve : %f', + roc_auc_test)

#Plotting confusion matrix
import seaborn as sns
conf_mat = confusion_matrix(y_test, cclf)
print(conf_mat)
#conf_normalized = conf_mat.astype('int') / conf_mat.sum(axis=1)[:, np.newaxis]
sns.heatmap(conf_mat, annot=True, fmt ='g')
plt.ylabel('True label')
plt.xlabel('Predicted label')

#Plot ROC Curve
plt.figure(0).clf()
fpr, tpr, thresholds = roc_curve(y_test,pred_test)
roc_auc_test = auc(fpr, tpr)
plt.plot(fpr,tpr,label="Test Data, auc="+str(roc_auc_test))

fpr, tpr, thresh = roc_curve(y_cv, pred_cv)
roc_auc_cv = auc(fpr, tpr)
plt.plot(fpr,tpr,label="Train Data, auc="+str(roc_auc_cv))
plt.title('ROC curve for Train and Test Dataset - BOW')
plt.xlabel('True Positive Rate')
plt.ylabel('False Positive Rate')
plt.legend(loc=0)
```

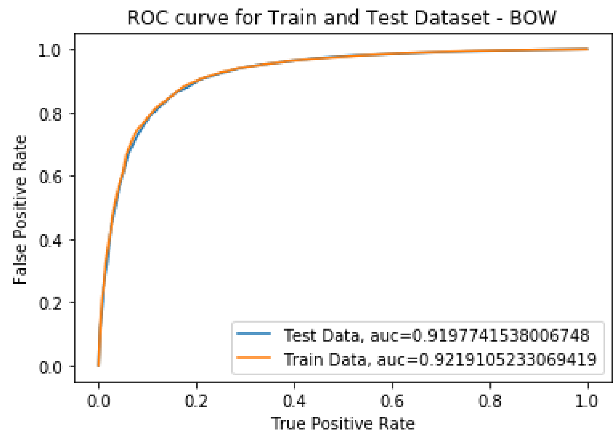
```
alpha value =  1e-05
Area under the ROC curve : %f 0.9197741538006748
```

```
Area under the ROC curve : %f 0.9197741538006748
[[ 1947   829]
 [  852 13927]]
```

Out[36]:

<matplotlib.legend.Legend at 0x4980ab70>



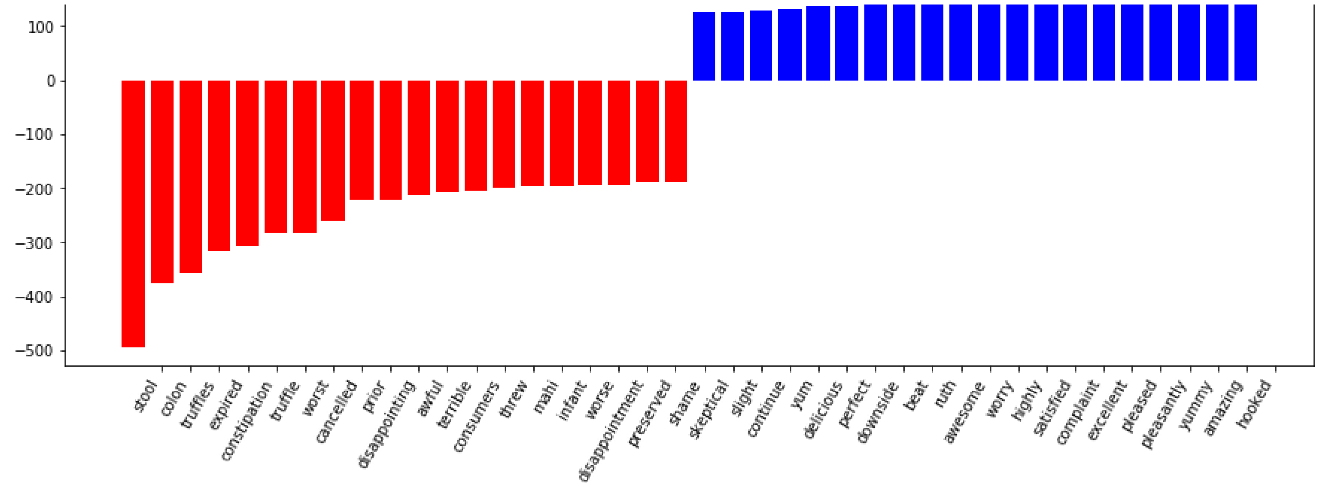


Observation for test dataset:

1. Out of 2889 negative data points, BOW predicts 1820 correctly and the remaining 1069 incorrect.
2. Similarly, out of 14666 positive data points, 624 are misclassified.
3. Accuracy 90% and AUC score is above 0.5

In [37]:

```
#Finding the top 20 features in BOW:
import matplotlib.pyplot as plt

def plot_coefficients(classifier, feature_names, top_features=20):
    coef = classifier.coef_.ravel()
    top_positive_coefficients = np.argsort(coef)[-top_features:]
    top_negative_coefficients = np.argsort(coef)[:top_features]
    top_coefficients = np.hstack([top_negative_coefficients, top_positive_coefficients])
 # create plot
    plt.figure(figsize=(15, 5))
    colors = ['red' if c < 0 else 'blue' for c in coef[top_coefficients]]
    plt.bar(np.arange(2 * top_features), coef[top_coefficients], color=colors)
    feature_names = np.array(feature_names)
    plt.xticks(np.arange(1, 1 + 2 * top_features), feature_names[top_coefficients], rotation=60, ha='right')
    plt.show()
    print(feature_names[top_positive_coefficients])
    print(feature_names[top_negative_coefficients])

plot_coefficients(clf, model.get_feature_names(), top_features=20)
```




```
['skeptical' 'slight' 'continue' 'yum' 'delicious' 'perfect' 'downside'
 'beat' 'ruth' 'awesome' 'worry' 'highly' 'satisfied' 'complaint'
 'excellent' 'pleased' 'pleasantly' 'yummy' 'amazing' 'hooked']
['stool' 'colon' 'truffles' 'expired' 'constipation' 'truffle' 'worst'
 'cancelled' 'prior' 'disappointing' 'awful' 'terrible' 'consumers'
 'threw' 'mahi' 'infant' 'worse' 'disappointment' 'preserved' 'shame']
```

[5.1.2] Applying Linear SVM on TFIDF, SET 2

In [38]:

```
# Please write all the code with proper documentation
alphas = np.array([0.00001, 0.0001, 0.001, 0.01, 1, 10, 100, 1000])
base_estimator = linear_model.SGDClassifier(loss='hinge',penalty='l2', random_state=0,class_weight = "b
alanced")
scoring = {'AUC': 'roc_auc'}
grid = GridSearchCV(estimator=base_estimator,param_grid=dict(alpha=alphas),scoring = scoring, refit = '
AUC')
grid.fit(train_tf_idf, y_tr)
print(grid)
# summarize the results of the grid search
(grid.best_score_)
print(grid.best_estimator_.alpha)
results_tr_tfidf = grid.cv_results_
#print(results_tfidf)
```

```
GridSearchCV(cv='warn', error_score='raise-deprecating',
       estimator=SGDClassifier(alpha=0.0001, average=False, class_weight='balanced',
       early_stopping=False, epsilon=0.1, eta0=0.0, fit_intercept=True,
       l1_ratio=0.15, learning_rate='optimal', loss='hinge', max_iter=None,
       n_iter=None, n_iter_no_change=5, n_jobs=None, penalty='l2',
       power_t=0.5, random_state=0, shuffle=True, tol=None,
       validation_fraction=0.1, verbose=0, warm_start=False),
       fit_params=None, iid='warn', n_jobs=None,
       param_grid={'alpha': array([1.e-05, 1.e-04, 1.e-03, 1.e-02, 1.e+00, 1.e+01, 1.e+02, 1.e+03])},
       pre_dispatch='2*n_jobs', refit='AUC', return_train_score='warn',
       scoring={'AUC': 'roc_auc'}, verbose=0)
1e-05
```

In [35]:

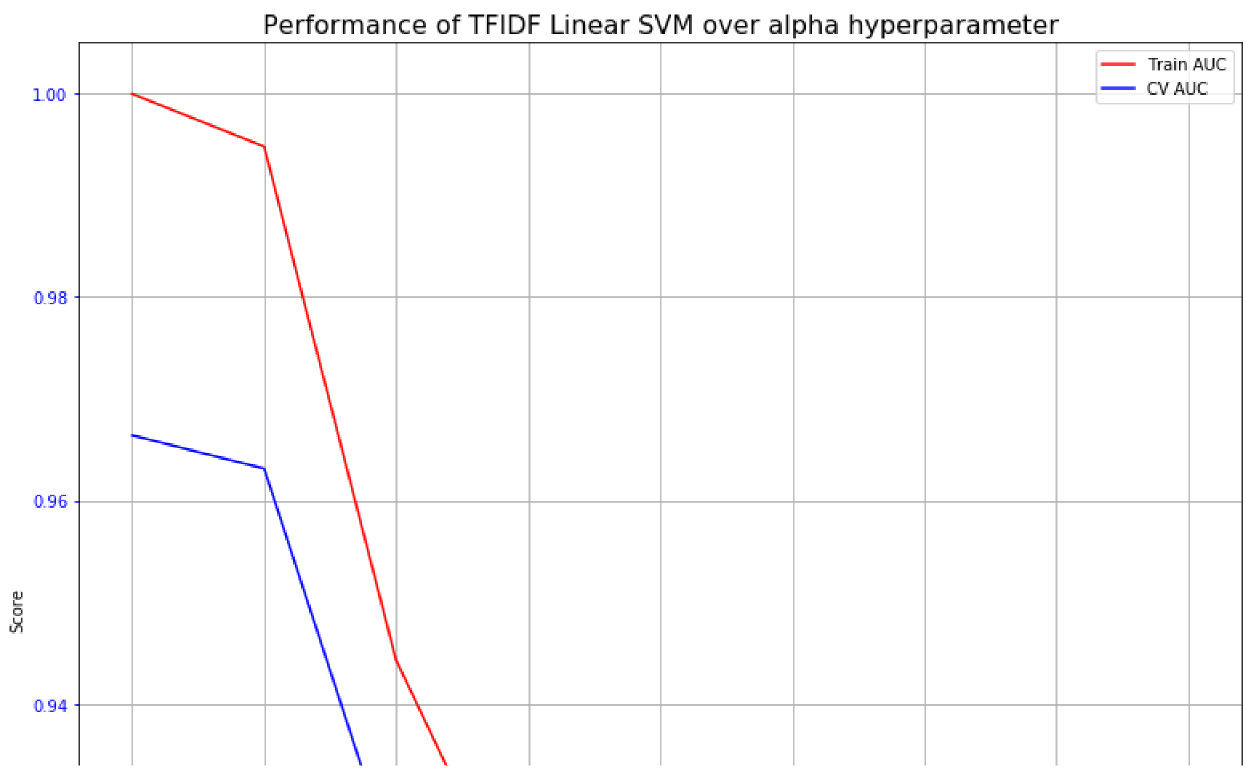
```
# Please write all the code with proper documentation
alphas = np.array([0.00001, 0.0001, 0.001, 0.01, 1, 10, 100, 1000])
base_estimator = linear_model.SGDClassifier(loss='hinge',penalty='l2', random_state=0,class_weight = "b
alanced")
scoring = {'AUC': 'roc_auc'}
grid = GridSearchCV(estimator=base_estimator,param_grid=dict(alpha=alphas),scoring = scoring, refit = '
AUC')
grid.fit(cv_tf_idf, y_cv)
print(grid)
# summarize the results of the grid search
(grid.best_score_)
print(grid.best_estimator_.alpha)
results_cv_tfidf = grid.cv_results_
```

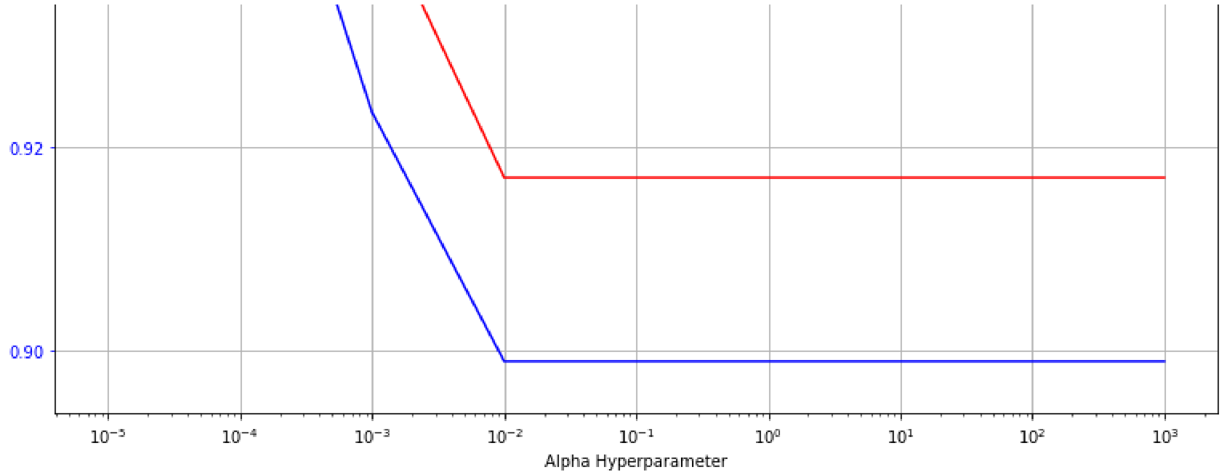
```
#print(results_tfidf)
```

```
GridSearchCV(cv='warn', error_score='raise-deprecating',
       estimator=SGDClassifier(alpha=0.0001, average=False, class_weight='balanced',
       early_stopping=False, epsilon=0.1, eta0=0.0, fit_intercept=True,
       l1_ratio=0.15, learning_rate='optimal', loss='hinge', max_iter=None,
       n_iter=None, n_iter_no_change=5, n_jobs=None, penalty='l2',
       power_t=0.5, random_state=0, shuffle=True, tol=None,
       validation_fraction=0.1, verbose=0, warm_start=False),
       fit_params=None, iid='warn', n_jobs=None,
       param_grid={'alpha': array([1.e-05, 1.e-04, 1.e-03, 1.e-02, 1.e+00, 1.e+01, 1.e+02, 1.e+03])},
       pre_dispatch='2*n_jobs', refit='AUC', return_train_score='warn',
       scoring={'AUC': 'roc_auc'}, verbose=0)
0.0001
```

In [39]:

```
#Performance over alpha hyperparameter
plt.figure(figsize=(13, 13))
plt.title("Performance of TFIDF Linear SVM over alpha hyperparameter",
          fontsize=16)

#ax.set_xlim(0, 402)
#ax.set_ylim(0.73, 1)

# Get the regular numpy array from the MaskedArray
X_axis = np.array(results_tr_tfidf['param_alpha'].data, dtype=float)
Y_axis_train = results_tr_tfidf['mean_train_AUC']
Y_axis_CV = results_tr_tfidf['mean_test_AUC']
#Y_axis = np.array(sorted(results_bow['mean_train_AUC']).data, dtype=float)
#fig, ax = plt.subplots()
ax = plt.gca()
ax.set_xscale('log')
#ax=plt.subplots()
curve1, = ax.plot(X_axis, Y_axis_train, label="Train AUC", color='r')
curve2, = ax.plot(X_axis, Y_axis_CV, label="CV AUC", color='b')
curves = [curve1, curve2]
ax.legend()
ax.set_ylabel("Score")
ax.set_xlabel("Alpha Hyperparameter")
ax.tick_params(axis='y', colors=curve1.get_color())
ax.tick_params(axis='y', colors=curve2.get_color())
#ax.plot(X_axis,Y_axis_CV)
#plt.legend(loc="best")
plt.grid()
plt.show()
```





In [ ]:

```
Observation:

    The graph shows that for best alpha value of 1e-05, AUC score for test data 0.97 respectively.
    As alpha increases, AUC score decreases but the accuracy curve is reaching stable value of 85%(at a
lpha=1e-02)
    AUC score approaches to 0.5 as alpha increases.
```

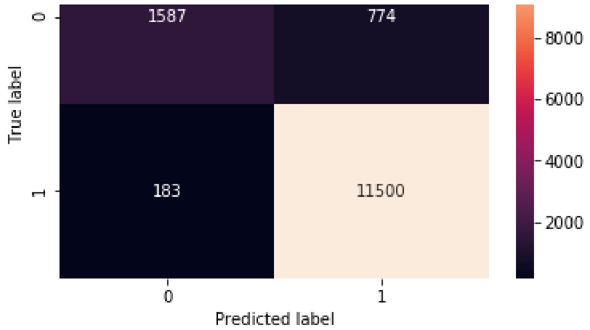
In [40]:

```
# After finding the best hyperparameter value for TFIDF which is 1e-05, applying linear SVM on train da
taset and predicting
# accuracy/AUC score for cv dataset
from sklearn import svm
from sklearn.model_selection import GridSearchCV
from sklearn.svm import LinearSVC
from sklearn import linear_model
from sklearn.calibration import CalibratedClassifierCV,calibration_curve
clf = linear_model.SGDClassifier(alpha = 1e-05,loss='hinge',penalty='l2', random_state=0,class_weight =
"balanced")
scoring = {'AUC': 'roc_auc'}
clf.fit(train_tf_idf,y_tr)

#Caliberate the classifier.
clf_calibrated=CalibratedClassifierCV(clf, cv='prefit', method='isotonic')
cclf = clf_calibrated.fit(train_tf_idf, y_tr).predict(cv_tf_idf)
pred_cv = clf_calibrated.predict_proba(cv_tf_idf)[:,1]
#pred_cv = np.argmax(log_pred,axis = 1)
print('alpha value = ',1e-05)
fpr, tpr, thresholds = roc_curve(y_cv,pred_cv)
roc_auc_cv = auc(fpr, tpr)
print('Area under the ROC curve : %f', + roc_auc_cv)

#Plotting confusion matrix
import seaborn as sns
conf_mat = confusion_matrix(y_cv, cclf)
print(conf_mat)
#conf_normalized = conf_mat.astype('int') / conf_mat.sum(axis=1)[:, np.newaxis]
sns.heatmap(conf_mat, annot=True, fmt ='g')
plt.ylabel('True label')
plt.xlabel('Predicted label')
```

```
alpha value =  1e-05
Area under the ROC curve : %f 0.9576169873340873
[[ 1587   774]
 [  183 11500]]
```

Out[40]:

```
Text(0.5, 15.0, 'Predicted label')
```



Observation for test dataset:

```
1. Out of 2267 negative data points, TFiDF predicts 1783 correctly and the remaining 484 incorre
ct.
2. Similarly, out of 11777 positive data points, 319 are misclassified.
3. Accuracy 94% and AUC score is above 0.5
```

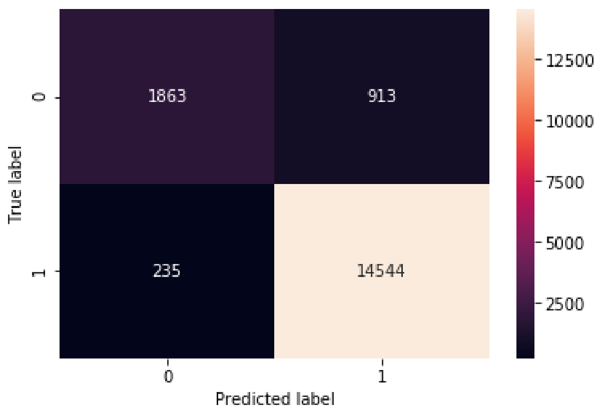
In [41]:

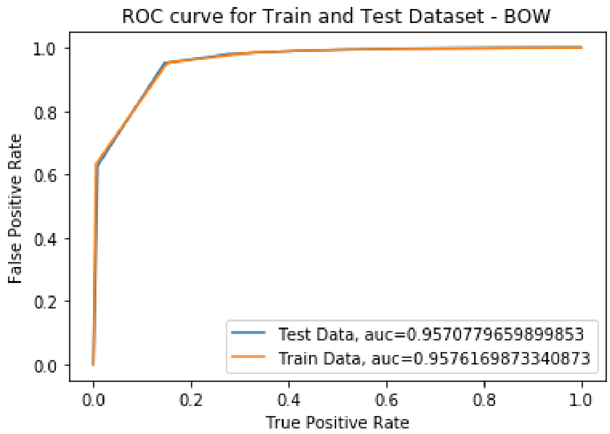
```python
# After finding the best hyperparameter value for TFIDF which is 1e-05, applying linear SVM on train da
taset and predicting
# accuracy/AUC score for cv dataset
from sklearn import svm
from sklearn.model_selection import GridSearchCV
from sklearn.svm import LinearSVC
from sklearn import linear_model
from sklearn.calibration import CalibratedClassifierCV,calibration_curve
clf = linear_model.SGDClassifier(alpha = 1e-05,loss='hinge',penalty='l2', random_state=0, class_weight
= "balanced")
scoring = {'AUC': 'roc_auc'}
clf.fit(train_tf_idf,y_tr)

#Caliberate the classifier.
clf_calibrated=CalibratedClassifierCV(clf, cv='prefit', method='isotonic')
cclf = clf_calibrated.fit(train_tf_idf, y_tr).predict(test_tf_idf)
pred_test = clf_calibrated.predict_proba(test_tf_idf)[:,1]
#pred_test = np.argmax(log_pred,axis = 1)
print('alpha value = ',1e-05)
fpr, tpr, thresholds = roc_curve(y_test,pred_test)
roc_auc_test = auc(fpr, tpr)
print('Area under the ROC curve : %f', + roc_auc_test)

#Plotting confusion matrix
import seaborn as sns
conf_mat = confusion_matrix(y_test, cclf)
print(conf_mat)
#conf_normalized = conf_mat.astype('int') / conf_mat.sum(axis=1)[:, np.newaxis]
sns.heatmap(conf_mat, annot=True, fmt ='g')
plt.ylabel('True label')
plt.xlabel('Predicted label')

#Plot ROC Curve
plt.figure(0).clf()
fpr, tpr, thresholds = roc_curve(y_test,pred_test)
roc_auc_test = auc(fpr, tpr)
plt.plot(fpr,tpr,label="Test Data, auc="+str(roc_auc_test))

fpr, tpr, thresh = roc_curve(y_cv, pred_cv)
roc_auc_cv = auc(fpr, tpr)
plt.plot(fpr,tpr,label="Train Data, auc="+str(roc_auc_cv))
plt.title('ROC curve for Train and Test Dataset - BOW')
plt.xlabel('True Positive Rate')
plt.ylabel('False Positive Rate')
plt.legend(loc=0)
```

```
alpha value =  1e-05
Area under the ROC curve : %f 0.9570779659899853
[[ 1863   913]
 [  235 14544]]
```

Out[41]:

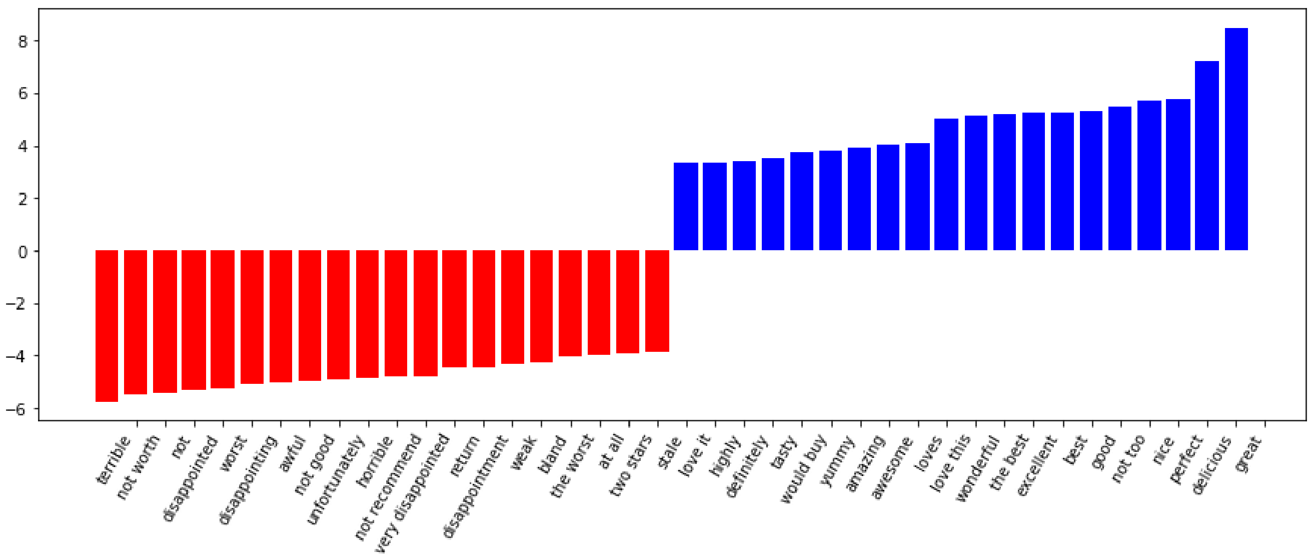
<matplotlib.legend.Legend at 0x4b2d26d8>





In [55]:

```
plot_coefficients(clf, tf_idf_vect.get_feature_names(), top_features=20)
```




```
['love it' 'highly' 'definitely' 'tasty' 'would buy' 'yummy' 'amazing'
 'awesome' 'loves' 'love this' 'wonderful' 'the best' 'excellent' 'best'
 'good' 'not too' 'nice' 'perfect' 'delicious' 'great']
['terrible' 'not worth' 'not' 'disappointed' 'worst' 'disappointing'
 'awful' 'not good' 'unfortunately' 'horrible' 'not recommend'
 'very disappointed' 'return' 'disappointment' 'weak' 'bland' 'the worst'
 'at all' 'two stars' 'stale']
```

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

In [42]:

```
# Please write all the code with proper documentation
alphas = np.array([0.00001, 0.0001, 0.001, 0.01, 1, 10, 100, 1000])
base_estimator = linear_model.SGDClassifier(loss='hinge',penalty='l2', random_state=0, class_weight='ba
lanced')
scoring = {'AUC': 'roc_auc'}
grid = GridSearchCV(estimator=base_estimator,param_grid=dict(alpha=alphas),scoring = scoring, refit = '
AUC')
grid.fit(sent_vectors, y_tr)
print(grid)
# summarize the results of the grid search
(grid.best_score_)
print(grid.best_estimator_.alpha)
results_avg_w2v = grid.cv_results_
#print(results)
```

```
GridSearchCV(cv='warn', error_score='raise-deprecating',
       estimator=SGDClassifier(alpha=0.0001, average=False, class_weight='balanced',
       early_stopping=False, epsilon=0.1, eta0=0.0, fit_intercept=True,
       l1_ratio=0.15, learning_rate='optimal', loss='hinge', max_iter=None,
       n_iter=None, n_iter_no_change=5, n_jobs=None, penalty='l2',
       power_t=0.5, random_state=0, shuffle=True, tol=None,
       validation_fraction=0.1, verbose=0, warm_start=False),
       fit_params=None, iid='warn', n_jobs=None,
       param_grid={'alpha': array([1.e-05, 1.e-04, 1.e-03, 1.e-02, 1.e+00, 1.e+01, 1.e+02, 1.e+03])},
       pre_dispatch='2*n_jobs', refit='AUC', return_train_score='warn',
       scoring={'AUC': 'roc_auc'}, verbose=0)
0.01
```

In [43]:

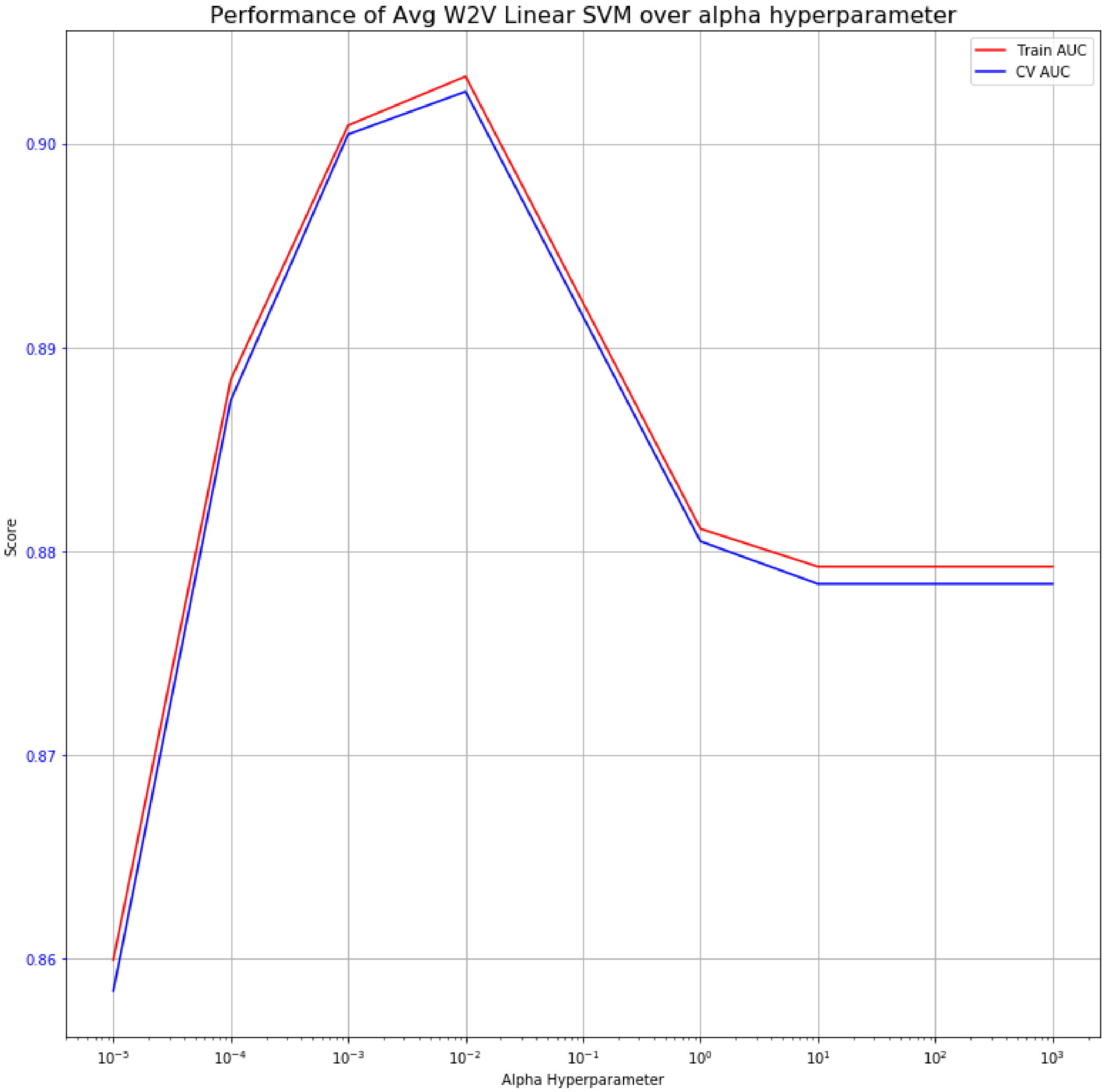
```
# Please write all the code with proper documentation
alphas = np.array([0.00001, 0.0001, 0.001, 0.01, 1, 10, 100, 1000])
base_estimator = linear_model.SGDClassifier(loss='hinge',penalty='l2', random_state=0, class_weight='ba
lanced')
scoring = {'AUC': 'roc_auc'}
grid = GridSearchCV(estimator=base_estimator,param_grid=dict(alpha=alphas),scoring = scoring, refit = '
AUC')
grid.fit(sent_vectors_cv, y_cv)
print(grid)
# summarize the results of the grid search
(grid.best_score_)
print(grid.best_estimator_.alpha)
results_avg_w2v_cv = grid.cv_results_
#print(results)
```

```
GridSearchCV(cv='warn', error_score='raise-deprecating',
       estimator=SGDClassifier(alpha=0.0001, average=False, class_weight='balanced',
       early_stopping=False, epsilon=0.1, eta0=0.0, fit_intercept=True,
       l1_ratio=0.15, learning_rate='optimal', loss='hinge', max_iter=None,
       n_iter=None, n_iter_no_change=5, n_jobs=None, penalty='l2',
       power_t=0.5, random_state=0, shuffle=True, tol=None,
       validation_fraction=0.1, verbose=0, warm_start=False),
       fit_params=None, iid='warn', n_jobs=None,
       param_grid={'alpha': array([1.e-05, 1.e-04, 1.e-03, 1.e-02, 1.e+00, 1.e+01, 1.e+02, 1.e+03])},
       pre_dispatch='2*n_jobs', refit='AUC', return_train_score='warn',
       scoring={'AUC': 'roc_auc'}, verbose=0)
1e-05
```

In [44]:

```
#Performance over alpha hyperparameter
plt.figure(figsize=(13, 13))
plt.title("Performance of Avg W2V Linear SVM over alpha hyperparameter",
          fontsize=16)
X_axis = np.array(results_avg_w2v['param_alpha'].data, dtype=float)
Y_axis_train = results_avg_w2v['mean_train_AUC']
Y_axis_CV = results_avg_w2v['mean_test_AUC']
ax = plt.gca()
ax.set_xscale('log')
curve1, = ax.plot(X_axis, Y_axis_train, label="Train AUC", color='r')
curve2, = ax.plot(X_axis, Y_axis_CV, label="CV AUC", color='b')
curves = [curve1, curve2]
```

```
ax.legend()
ax.set_ylabel("Score")
ax.set_xlabel("Alpha Hyperparameter")
ax.tick_params(axis='y', colors=curve1.get_color())
ax.tick_params(axis='y', colors=curve2.get_color())
#ax.plot(X_axis,Y_axis_CV)
#plt.legend(loc="best")
plt.grid()
plt.show()
```



Observation:

```
The graph shows that for best alpha value of 0.01, AUC score 0.9.
As alpha increases, AUC score decreases.
AUC score approaches to 0.5 as alpha increases.
```

In [46]:

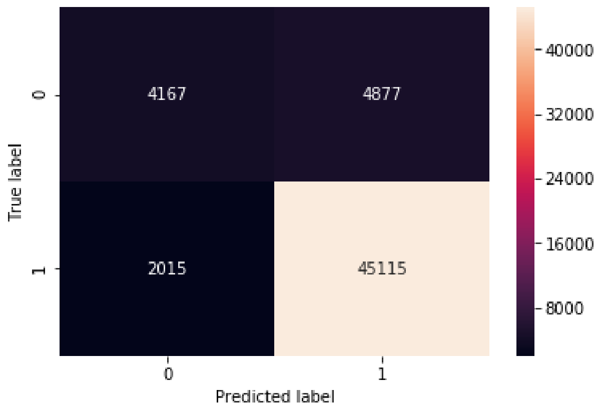
```
# After finding the best hyperparameter value for AVG W2V which is 0.001, applying linear SVM on train
dataset and predicting
# accuracy/AUC score for cv dataset
from sklearn import svm
from sklearn.model_selection import GridSearchCV
from sklearn.svm import LinearSVC
from sklearn import linear_model
from sklearn.calibration import CalibratedClassifierCV,calibration_curve
clf = linear_model.SGDClassifier(alpha = 0.01,loss='hinge',penalty='l1', random_state=0, class_weight='
balanced')
```

```
"balanced")
scoring = {'AUC': 'roc_auc'}
clf.fit(sent_vectors,y_tr)

#Caliberate the classifier.
clf_calibrated=CalibratedClassifierCV(clf,method='isotonic')
cclf = clf_calibrated.fit(sent_vectors, y_tr).predict(sent_vectors)
pred_cv = clf_calibrated.predict_proba(sent_vectors)[:,1]
#pred_cv = np.argmax(log_pred,axis = 1)
print('alpha value = ',0.01)
fpr, tpr, thresholds = roc_curve(y_tr,pred_cv)
roc_auc_cv = auc(fpr, tpr)
print('Area under the ROC curve : %f', + roc_auc_cv)

#Plotting confusion matrix
import seaborn as sns
conf_mat = confusion_matrix(y_tr, cclf)
print(conf_mat)
#conf_normalized = conf_mat.astype('int') / conf_mat.sum(axis=1)[:, np.newaxis]
sns.heatmap(conf_mat, annot=True, fmt ='g')
plt.ylabel('True label')
plt.xlabel('Predicted label')
```

```
alpha value =  0.01
Area under the ROC curve : %f 0.8927773833242635
[[ 4167  4877]
 [ 2015 45115]]
```

Out[46]:

```
Text(0.5, 15.0, 'Predicted label')
```



In [48]:

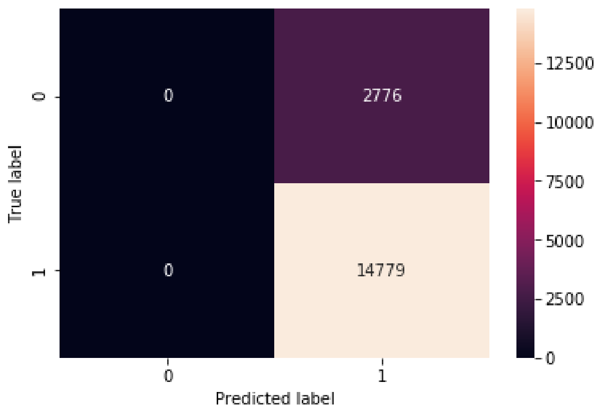
```
# After finding the best hyperparameter value for AVG W2V, applying Decision tree on train dataset and predicting
# accuracy/AUC score for test dataset
from sklearn import svm
from sklearn.model_selection import GridSearchCV
from sklearn.svm import LinearSVC
from sklearn import linear_model
from sklearn.calibration import CalibratedClassifierCV,calibration_curve
clf = linear_model.SGDClassifier(alpha = 0.01,loss='hinge',penalty='l2', random_state=0,class_weight = "balanced")
scoring = {'AUC': 'roc_auc'}
clf.fit(sent_vectors,y_tr)

#Caliberate the classifier.
clf_calibrated=CalibratedClassifierCV(clf, cv='prefit', method='isotonic')
cclf = clf_calibrated.fit(sent_vectors, y_tr).predict(sent_vectors_test)
pred_test = clf_calibrated.predict_proba(sent_vectors_test)[:,1]
fpr, tpr, thresholds = roc_curve(y_test,pred_test)
roc_auc_test = auc(fpr, tpr)
print('Area under the ROC curve : %f', + roc_auc_test)

#Plotting confusion matrix
import seaborn as sns
conf_mat = confusion_matrix(y_test, cclf)
print(conf_mat)
```

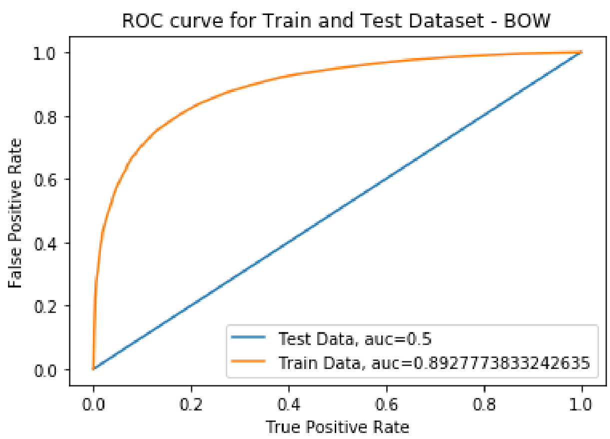
```
#conf_normalized = conf_mat.astype('int') / conf_mat.sum(axis=1)[:, np.newaxis]
sns.heatmap(conf_mat, annot=True, fmt ='g')
plt.ylabel('True label')
plt.xlabel('Predicted label')

#Plot ROC Curve
plt.figure(0).clf()
fpr, tpr, thresholds = roc_curve(y_test,pred_test)
roc_auc_test = auc(fpr, tpr)
plt.plot(fpr,tpr,label="Test Data, auc="+str(roc_auc_test))

fpr, tpr, thresh = roc_curve(y_tr, pred_cv)
roc_auc_cv = auc(fpr, tpr)
plt.plot(fpr,tpr,label="Train Data, auc="+str(roc_auc_cv))
plt.title('ROC curve for Train and Test Dataset - BOW')
plt.xlabel('True Positive Rate')
plt.ylabel('False Positive Rate')
plt.legend(loc=0)
```

```
Area under the ROC curve : %f 0.5
[[    0  2776]
 [    0 14779]]
```

Out[48]:

```
<matplotlib.legend.Legend at 0x4a80f9b0>
```





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

In [49]:

```
# Please write all the code with proper documentation
from sklearn import svm
from sklearn.model_selection import GridSearchCV
from sklearn.svm import LinearSVC
from sklearn import linear_model
from sklearn.calibration import CalibratedClassifierCV,calibration_curve
alphas = np.array([0.00001, 0.0001, 0.001, 0.01, 1, 10, 100, 1000])
base_estimator = linear_model.SGDClassifier(loss='hinge',penalty='l2', random_state=0, class_weight="ba
lanced")
```

```
scoring = {'AUC': 'roc_auc'}
grid = GridSearchCV(estimator=base_estimator,param_grid=dict(alpha=alphas),scoring = scoring, refit = '
AUC')
grid.fit(tfidf_sent_vectors, y_tr)
print(grid)
# summarize the results of the grid search
(grid.best_score_)
print(grid.best_estimator_.alpha)
results_tfidf_w2v_tr = grid.cv_results_
#print(results)
```

```
GridSearchCV(cv='warn', error_score='raise-deprecating',
       estimator=SGDClassifier(alpha=0.0001, average=False, class_weight='balanced',
       early_stopping=False, epsilon=0.1, eta0=0.0, fit_intercept=True,
       l1_ratio=0.15, learning_rate='optimal', loss='hinge', max_iter=None,
       n_iter=None, n_iter_no_change=5, n_jobs=None, penalty='l2',
       power_t=0.5, random_state=0, shuffle=True, tol=None,
       validation_fraction=0.1, verbose=0, warm_start=False),
       fit_params=None, iid='warn', n_jobs=None,
       param_grid={'alpha': array([1.e-05, 1.e-04, 1.e-03, 1.e-02, 1.e+00, 1.e+01, 1.e+02, 1.e+03])},
       pre_dispatch='2*n_jobs', refit='AUC', return_train_score='warn',
       scoring={'AUC': 'roc_auc'}, verbose=0)
0.01
```

In [50]:

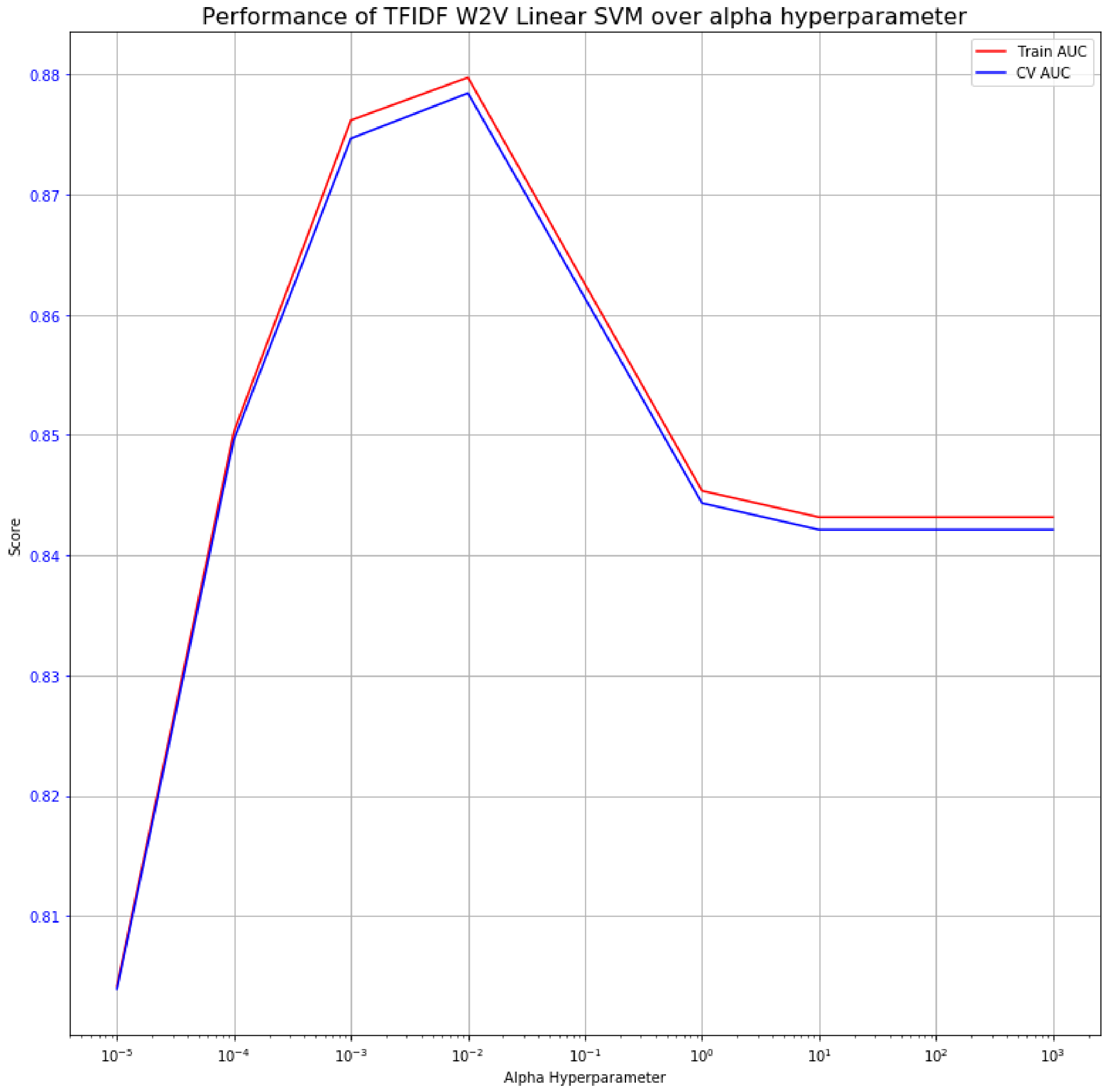
```
# Please write all the code with proper documentation
from sklearn import svm
from sklearn.model_selection import GridSearchCV
from sklearn.svm import LinearSVC
from sklearn import linear_model
from sklearn.calibration import CalibratedClassifierCV,calibration_curve
alphas = np.array([0.00001, 0.0001, 0.001, 0.01, 1, 10, 100, 1000])
base_estimator = linear_model.SGDClassifier(loss='hinge',penalty='l2', random_state=0, class_weight="ba
lanced")
scoring = {'AUC': 'roc_auc'}
grid = GridSearchCV(estimator=base_estimator,param_grid=dict(alpha=alphas),scoring = scoring, refit = '
AUC')
grid.fit(tfidf_sent_vectors_cv, y_cv)
print(grid)
# summarize the results of the grid search
(grid.best_score_)
print(grid.best_estimator_.alpha)
results_tfidf_w2v_cv = grid.cv_results_
#print(results)
```

```
GridSearchCV(cv='warn', error_score='raise-deprecating',
       estimator=SGDClassifier(alpha=0.0001, average=False, class_weight='balanced',
       early_stopping=False, epsilon=0.1, eta0=0.0, fit_intercept=True,
       l1_ratio=0.15, learning_rate='optimal', loss='hinge', max_iter=None,
       n_iter=None, n_iter_no_change=5, n_jobs=None, penalty='l2',
       power_t=0.5, random_state=0, shuffle=True, tol=None,
       validation_fraction=0.1, verbose=0, warm_start=False),
       fit_params=None, iid='warn', n_jobs=None,
       param_grid={'alpha': array([1.e-05, 1.e-04, 1.e-03, 1.e-02, 1.e+00, 1.e+01, 1.e+02, 1.e+03])},
       pre_dispatch='2*n_jobs', refit='AUC', return_train_score='warn',
       scoring={'AUC': 'roc_auc'}, verbose=0)
0.01
```

In [51]:

```
#Performance over alpha hyperparameter
plt.figure(figsize=(13, 13))
plt.title("Performance of TFIDF W2V Linear SVM over alpha hyperparameter",
          fontsize=16)
X_axis = np.array(results_tfidf_w2v_tr['param_alpha'].data, dtype=float)
Y_axis_train = results_tfidf_w2v_tr['mean_train_AUC']
Y_axis_CV = results_tfidf_w2v_tr['mean_test_AUC']
ax = plt.gca()
ax.set_xscale('log')
curve1, = ax.plot(X_axis, Y_axis_train, label="Train AUC", color='r')
curve2, = ax.plot(X_axis, Y_axis_CV, label="CV AUC", color='b')
curves = [curve1, curve2]
ax.legend()
```

```
ax.legend()
ax.set_ylabel("Score")
ax.set_xlabel("Alpha Hyperparameter")
ax.tick_params(axis='y', colors=curve1.get_color())
ax.tick_params(axis='y', colors=curve2.get_color())
#ax.plot(X_axis,Y_axis_CV)
#plt.legend(loc="best")
plt.grid()
plt.show()
```



In [53]:

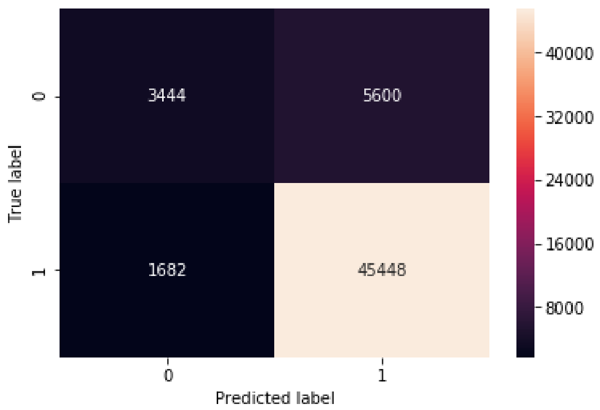
```
# After finding the best hyperparameter value for TFIDF W2V which is 0.01, applying linear SVM on train
dataset and predicting
# accuracy/AUC score for cv dataset
from sklearn import svm
from sklearn.model_selection import GridSearchCV
from sklearn.svm import LinearSVC
from sklearn import linear_model
from sklearn.calibration import CalibratedClassifierCV,calibration_curve
clf = linear_model.SGDClassifier(alpha = 0.001,loss='hinge',penalty='l2', random_state=0, class_weight=
"balanced")
scoring = {'AUC': 'roc_auc'}
clf.fit(tfidf_sent_vectors,y_tr)

#Caliberate the classifier.
clf_calibrated=CalibratedClassifierCV(clf, cv='prefit', method='isotonic')
cclf = clf_calibrated.fit(tfidf_sent_vectors, y_tr).predict(tfidf_sent_vectors)
pred_cv = clf_calibrated.predict_proba(tfidf_sent_vectors)[:,1]
fpr, tpr, thresholds = roc_curve(y_tr,pred_cv)
```

```
roc_auc_cv = auc(fpr, tpr)
print('Area under the ROC curve : %f', + roc_auc_cv)

#Plotting confusion matrix
import seaborn as sns
conf_mat = confusion_matrix(y_tr, cclf)
print(conf_mat)
#conf_normalized = conf_mat.astype('int') / conf_mat.sum(axis=1)[:, np.newaxis]
sns.heatmap(conf_mat, annot=True, fmt ='g')
plt.ylabel('True label')
plt.xlabel('Predicted label')
```

```
Area under the ROC curve : %f 0.8746821900390697
[[ 3444  5600]
 [ 1682 45448]]
```

Out[53]:

```
Text(0.5, 15.0, 'Predicted label')
```



In [54]:

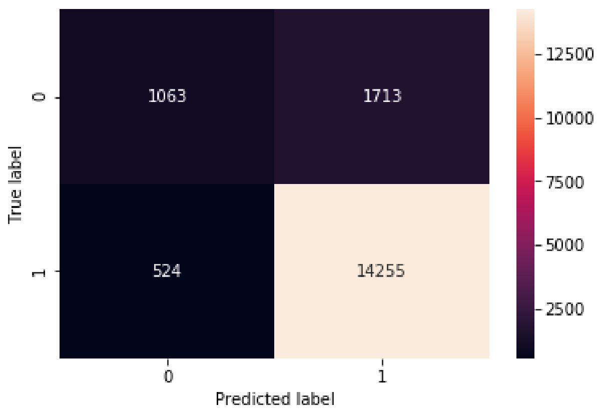
```
# After finding the best hyperparameter value for TFIDF W2V which is 0.01, applying linear SVM on train
dataset and predicting
# accuracy/AUC score for test dataset
from sklearn import svm
from sklearn.model_selection import GridSearchCV
from sklearn.svm import LinearSVC
from sklearn import linear_model
from sklearn.calibration import CalibratedClassifierCV,calibration_curve
clf = linear_model.SGDClassifier(alpha = 0.001,loss='hinge',penalty='l2', random_state=0, class_weight=
'balanced')
scoring = {'AUC': 'roc_auc'}
clf.fit(tfidf_sent_vectors,y_tr)

#Caliberate the classifier.
clf_calibrated=CalibratedClassifierCV(clf, cv='prefit', method='isotonic')
cclf = clf_calibrated.fit(tfidf_sent_vectors, y_tr).predict(tfidf_sent_vectors_test)
pred_test = clf_calibrated.predict_proba(tfidf_sent_vectors_test)[:,1]
#pred_test = np.argmax(log_pred,axis = 1)
print('alpha value = ',0.01)
fpr, tpr, thresholds = roc_curve(y_test,pred_test)
roc_auc_test = auc(fpr, tpr)
print('Area under the ROC curve : %f', + roc_auc_test)

#Plotting confusion matrix
import seaborn as sns
conf_mat = confusion_matrix(y_test, cclf)
print(conf_mat)
#conf_normalized = conf_mat.astype('int') / conf_mat.sum(axis=1)[:, np.newaxis]
sns.heatmap(conf_mat, annot=True, fmt ='g')
plt.ylabel('True label')
plt.xlabel('Predicted label')

#Plot ROC Curve
plt.figure(0).clf()
fpr, tpr, thresholds = roc_curve(y_test,pred_test)
roc_auc_test = auc(fpr, tpr)
```

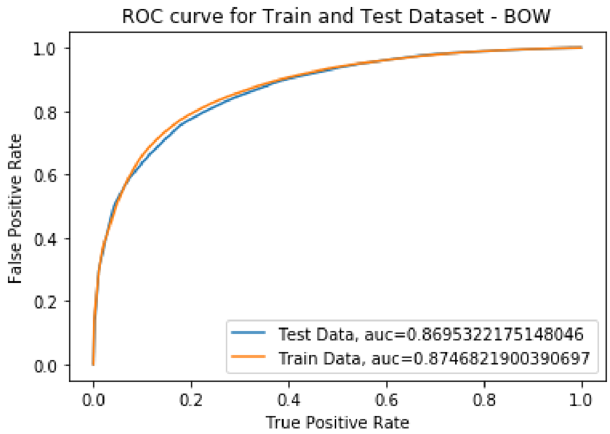
```
plt.plot(fpr,tpr,label="Test Data, auc="+str(roc_auc_test))

fpr, tpr, thresh = roc_curve(y_tr, pred_cv)
roc_auc_cv = auc(fpr, tpr)
plt.plot(fpr,tpr,label="Train Data, auc="+str(roc_auc_cv))
plt.title('ROC curve for Train and Test Dataset - BOW')
plt.xlabel('True Positive Rate')
plt.ylabel('False Positive Rate')
plt.legend(loc=0)
```

```
alpha value =  0.01
Area under the ROC curve : %f 0.8695322175148046
[[ 1063  1713]
 [  524 14255]]
```

Out[54]:

```
<matplotlib.legend.Legend at 0x453c7908>
```





[5.2] RBF SVM

[5.2.1] Applying RBF SVM on BOW, SET 1

In [3]:

```
# Please write all the code with proper documentation
```

[5.2.2] Applying RBF SVM on TFIDF, SET 2

In [3]:

```
# Please write all the code with proper documentation
```

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

In [3]:

```
# Please write all the code with proper documentation
```

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

In [3]:

```
# Please write all the code with proper documentation
```

[6] Conclusions

In [55]:

```
# Please compare all your models using Prettytable library

from prettytable import PrettyTable
table = PrettyTable(["model","alpha value","ROC"])
table.add_row(["Linear SVM using BoW", "1e-05","0.81"])
table.add_row(["Linear SVM using TFIDF", "1e-05","0.86"])
table.add_row(["Linear SVM using AVG W2V", "0.01","0.5"])
table.add_row(["Linear SVM using TFIDF W2V", "0.01","0.86"])
print(table)
```

```
+----------------------------+-------------+------+
|           model            | alpha value | ROC  |
+----------------------------+-------------+------+
|    Linear SVM using BoW    |    1e-05    | 0.81 |
|   Linear SVM using TFIDF   |    1e-05    | 0.86 |
|  Linear SVM using AVG W2V  |     0.01    | 0.5  |
| Linear SVM using TFIDF W2V |     0.01    | 0.86 |
+----------------------------+-------------+------+
```

Observation:

Overall, linear SVM works well when BOW/TFIDF/TFIDF W2V is used. While using Average W2V vectorization technique, though accuracy is 85%, AUC score is very low 0.5. This in turn overfits the data, misclassifying entire negative class as positive, since majority data points are positive.