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 [40]:
```
%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 [42]:

```
# 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 50
0000 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 Sco
re != 3 LIMIT 500000""", con)
# for tsne assignment you can take 5k data points
```

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

# Give reviews with Score>3 a positive rating(1), and reviews with a sc
ore<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 (500000, 10)
```

Out[42]:

	Id	ProductId	UserId	ProfileName	HelpfulnessNumerator	Helpfulnes
0	1	B001E4KFG0	A3SGXH7AUHU8GW	delmartian	1	1
1	2	B00813GRG4	A1D87F6ZCVE5NK	dll pa	0	0

	Id	ProductId	UserId	ProfileName	HelpfulnessNumerator	Helpfulnes
2	3	B000LQOCH0	ABXLMWJIXXAIN	Natalia Corres "Natalia Corres"	1	1

In [43]:
```
display = pd.read_sql_query("""
SELECT UserId, ProductId, ProfileName, Time, Score, Text, COUNT(*)
FROM Reviews
GROUP BY UserId
HAVING COUNT(*)>1
""", con)
```

In [44]:
```
print(display.shape)
display.head()
```

(80668, 7)

Out[44]:

	UserId	ProductId	ProfileName	Time	Score	Text	COU
0	#oc-R115TNMSPFT9I7	B007Y59HVM	Breyton	1331510400	2	Overall its just OK when considering the price...	2
1	#oc-R11D9D7SHXIJB9	B005HG9ET0	Louis E. Emory "hoppy"	1342396800	5	My wife has recurring extreme muscle spasms, u...	3

	UserId	ProductId	ProfileName	Time	Score	Text	COU
2	#oc-R11DNU2NBKQ23Z	B007Y59HVM	Kim Cieszykowski	1348531200	1	This coffee is horrible and unfortunately not ...	2
3	#oc-R11O5J5ZVQE25C	B005HG9ET0	Penguin Chick	1346889600	5	This will be the bottle that you grab from the...	3
4	#oc-R12KPBODL2B5ZD	B007OSBE1U	Christopher P. Presta	1348617600	1	I didnt like this coffee. Instead of telling y...	2

In [45]:
```
display[display['UserId']=='AZY10LLTJ71NX']
```

Out[45]:

	UserId	ProductId	ProfileName	Time	Score	Text
80638	AZY10LLTJ71NX	B006P7E5ZI	undertheshrine "undertheshrine"	1334707200	5	I was recommended to try green tea extract to ...

In [46]:
```
display['COUNT(*)'].sum()
```

Out[46]: 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 [47]:
```
display= pd.read_sql_query("""
SELECT *
FROM Reviews
WHERE Score != 3 AND UserId="AR5J8UI46CURR"
ORDER BY ProductID
""", con)
display.head()
```

Out[47]:

	Id	ProductId	UserId	ProfileName	HelpfulnessNumerator	Helpfuln
0	78445	B000HDL1RQ	AR5J8UI46CURR	Geetha Krishnan	2	2
1	138317	B000HDOPYC	AR5J8UI46CURR	Geetha Krishnan	2	2

	Id	ProductId	UserId	ProfileName	HelpfulnessNumerator	Helpfuln
2	138277	B000HDOPYM	AR5J8UI46CURR	Geetha Krishnan	2	2
3	73791	B000HDOPZG	AR5J8UI46CURR	Geetha Krishnan	2	2
4	155049	B000PAQ75C	AR5J8UI46CURR	Geetha Krishnan	2	2

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 [48]:
```
#Sorting data according to ProductId in ascending order
sorted_data=filtered_data.sort_values('ProductId', axis=0, ascending=True, inplace=False, kind='quicksort', na_position='last')
```

In [49]:
```
#Deduplication of entries
final=sorted_data.drop_duplicates(subset={"UserId","ProfileName","Time","Text"}, keep='first', inplace=False)
final.shape
```

Out[49]:
```
(348262, 10)
```

In [50]:
```
#Checking to see how much % of data still remains
(final['Id'].size*1.0)/(filtered_data['Id'].size*1.0)*100
```

Out[50]:
```
69.6524
```

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 [51]:
```
display= pd.read_sql_query("""
SELECT *
FROM Reviews
WHERE Score != 3 AND Id=44737 OR Id=64422
ORDER BY ProductID
""", con)

display.head()
```

Out[51]:

	Id	ProductId	UserId	ProfileName	HelpfulnessNumerator	Helpfuln
0	64422	B000MIDROQ	A161DK06JJMCYF	J. E. Stephens "Jeanne"	3	1
1	44737	B001EQ55RW	A2V0I904FH7ABY	Ram	3	2

In [52]:
```
final=final[final.HelpfulnessNumerator<=final.HelpfulnessDenominator]
```

In [53]:
```
#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()
```

```
(348260, 10)
```

Out[53]:
```
1    293516
0     54744
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 [54]:

```
# 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)
```

```
This book was purchased as a birthday gift for a 4 year old boy. He squ
ealed with delight and hugged it when told it was his to keep and he di
```

```
d not have to return it to the library.
==================================================
I've purchased both the Espressione Espresso (classic) and the 100% Ara
bica.  My vote is definitely with the 100% Arabica.  The flavor has mor
e bite and flavor (much more like European coffee than American).
==================================================
This is a great product. It is very healthy for all of our dogs, and it
is the first food that they all love to eat. It helped my older dog los
e weight and my 10 year old lab gain the weight he needed to be health
y.
==================================================
I find everything I need at Amazon so I always look there first. Chocol
ate tennis balls for a tennis party, perfect! They were the size of mal
ted milk balls. Unfortunately, they arrived 3 days after the party. The
caveat here is, not everything from Amazon may arrive at an impressive
2 or 3 days. This shipment took 8 days from the Candy/Cosmetic Depot ba
ck east to southern California.
==================================================
```

In [55]:

```python
# remove urls from text python: https://stackoverflow.com/a/40823105/40
84039
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)
```

```
This book was purchased as a birthday gift for a 4 year old boy. He squ
ealed with delight and hugged it when told it was his to keep and he di
d not have to return it to the library.
```

In [56]:

```python
# https://stackoverflow.com/questions/16206380/python-beautifulsoup-how
-to-remove-all-tags-from-an-element
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)
```

```
This book was purchased as a birthday gift for a 4 year old boy. He squ
ealed with delight and hugged it when told it was his to keep and he di
d not have to return it to the library.
==================================================
I've purchased both the Espressione Espresso (classic) and the 100% Ara
bica.  My vote is definitely with the 100% Arabica.  The flavor has mor
e bite and flavor (much more like European coffee than American).
==================================================
This is a great product. It is very healthy for all of our dogs, and it
is the first food that they all love to eat. It helped my older dog los
e weight and my 10 year old lab gain the weight he needed to be health
y.
==================================================
I find everything I need at Amazon so I always look there first. Chocol
ate tennis balls for a tennis party, perfect! They were the size of mal
ted milk balls. Unfortunately, they arrived 3 days after the party. The
caveat here is, not everything from Amazon may arrive at an impressive
2 or 3 days. This shipment took 8 days from the Candy/Cosmetic Depot ba
ck east to southern California.
```

In [57]:

```
# 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 [58]:

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

```
This is a great product. It is very healthy for all of our dogs, and it
is the first food that they all love to eat. It helped my older dog los
e weight and my 10 year old lab gain the weight he needed to be health
y.
==================================================
```

In [59]:

```
#remove words with numbers python: https://stackoverflow.com/a/1808237
0/4084039
sent_0 = re.sub("\S*\d\S*", "", sent_0).strip()
print(sent_0)
```

```
This book was purchased as a birthday gift for a  year old boy. He sque
aled with delight and hugged it when told it was his to keep and he did
not have to return it to the library.
```

In [60]:

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

```
This is a great product It is very healthy for all of our dogs and it i
s the first food that they all love to eat It helped my older dog lose
weight and my 10 year old lab gain the weight he needed to be healthy
```

In [61]:

```
# https://gist.github.com/sebleier/554280
# we are removing the words from the stop words list: 'no', 'nor', 'no
t'
# <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', 'o
urs', 'ourselves', 'you', "you're", "you've",\
            "you'll", "you'd", 'your', 'yours', 'yourself', 'yourselve
s', 'he', 'him', 'his', 'himself', \
            'she', "she's", 'her', 'hers', 'herself', 'it', "it's", 'it
s', 'itself', 'they', 'them', 'their',\
            'theirs', 'themselves', 'what', 'which', 'who', 'whom', 'th
is', 'that', "that'll", 'these', 'those', \
            'am', 'is', 'are', 'was', 'were', 'be', 'been', 'being', 'h
ave', 'has', 'had', 'having', 'do', 'does', \
            'did', 'doing', 'a', 'an', 'the', 'and', 'but', 'if', 'or',
 'because', 'as', 'until', 'while', 'of', \
            'at', 'by', 'for', 'with', 'about', 'against', 'between',
'into', 'through', 'during', 'before', 'after',\
            'above', 'below', 'to', 'from', 'up', 'down', 'in', 'out',
'on', 'off', 'over', 'under', 'again', 'further',\
            'then', 'once', 'here', 'there', 'when', 'where', 'why', 'h
ow', 'all', 'any', 'both', 'each', 'few', 'more',\
            'most', 'other', 'some', 'such', 'only', 'own', 'same', 's
o', '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', "is
n'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 [62]:

```
# 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%|██████████████████████████████████████████████████
█████| 348260/348260 [02:24<00:00, 2411.42it/s]
```

In [63]:

```
preprocessed_reviews[1500]
```

Out[63]:

```
'great product healthy dogs first food love eat helped older dog lose w
eight year old lab gain weight needed healthy'
```

[3.2] Preprocessing Review Summary

In [64]:

```
## Similartly you can do preprocessing for review summary also.
```

[4] Featurization

[4.1] BAG OF WORDS

In [65]:
```
#BoW
count_vect = CountVectorizer() #in scikit-learn
count_vect.fit(preprocessed_reviews)
print("some feature names ", count_vect.get_feature_names()[:10])
print('='*50)

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

```
some feature names  ['aa', 'aaa', 'aaaa', 'aaaaa', 'aaaaaa', 'aaaaaaaaa
aa', 'aaaaaaaaaaaa', 'aaaaaaaaaaaaa', 'aaaaaaaaaaaaaa', 'aaaaaaaaaaaaaa
a']
==================================================
the type of count vectorizer  <class 'scipy.sparse.csr.csr_matrix'>
the shape of out text BOW vectorizer  (348260, 113898)
the number of unique words  113898
```

[4.2] Bi-Grams and n-Grams.

In [66]:
```
#bi-gram, tri-gram and n-gram

#removing stop words like "not" should be avoided before building n-gra
ms
# count_vect = CountVectorizer(ngram_range=(1,2))
# please do read the CountVectorizer documentation http://scikit-learn.
org/stable/modules/generated/sklearn.feature_extraction.text.CountVecto
rizer.html

# you can choose these numebrs min_df=10, max_features=5000, of your ch
oice
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_s
hape())
```

```
print("the number of unique words including both unigrams and bigrams "
, final_bigram_counts.get_shape()[1])
```

```
the type of count vectorizer  <class 'scipy.sparse.csr.csr_matrix'>
the shape of out text BOW vectorizer  (348260, 5000)
the number of unique words including both unigrams and bigrams  5000
```

[4.3] TF-IDF

In [67]:

```
tf_idf_vect = TfidfVectorizer(ngram_range=(1,2), min_df=10)
tf_idf_vect.fit(preprocessed_reviews)
print("some sample features(unique words in the corpus)",tf_idf_vect.ge
t_feature_names()[0:10])
print('='*50)

final_tf_idf = tf_idf_vect.transform(preprocessed_reviews)
print("the type of count vectorizer ",type(final_tf_idf))
print("the shape of out text TFIDF vectorizer ",final_tf_idf.get_shape
())
print("the number of unique words including both unigrams and bigrams "
, final_tf_idf.get_shape()[1])
```

```
some sample features(unique words in the corpus) ['aa', 'aaa', 'aaaaa',
'aaah', 'aafco', 'ab', 'aback', 'abandon', 'abandoned', 'abbey']
==================================================
the type of count vectorizer  <class 'scipy.sparse.csr.csr_matrix'>
the shape of out text TFIDF vectorizer  (348260, 194764)
the number of unique words including both unigrams and bigrams  194764
```

[4.4] Word2Vec

In [68]:

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

In [69]:
```python
# 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 val
ues
# To use this code-snippet, download "GoogleNews-vectors-negative300.bi
n"
# from https://drive.google.com/file/d/0B7XkCwpI5KDYNlNUTTlSS21pQmM/edi
t
# it's 1.9GB in size.


# http://kavita-ganesan.com/gensim-word2vec-tutorial-starter-code/#.W17
SRFAzZPY
# 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.bin', binary=True)
        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_trai
n_w2v = True, to train your own w2v ")
```

```
[('fantastic', 0.8856132626533508), ('terrific', 0.8841899633407593),
('excellent', 0.8641712665557861), ('awesome', 0.863487958908081), ('go
od', 0.8613229393959045), ('wonderful', 0.8187414407730103), ('perfec
t', 0.7689666152000427), ('nice', 0.7594492435455322), ('fabulous', 0.7
563694715499878), ('amazing', 0.7314964532852173)]
==================================================
[('nastiest', 0.8637551665306091), ('disgusting', 0.7737479209899902),
('greatest', 0.7718157172203064), ('best', 0.7317373752593994), ('horri
ble', 0.705586314201355), ('vile', 0.694715142250061), ('horrid', 0.689
056396484375), ('terrible', 0.6835475564002991), ('awful', 0.6760625243
186951), ('tastiest', 0.657644510269165)]
```

In [70]:

```
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  32908
sample words  ['book', 'purchased', 'birthday', 'gift', 'year', 'old',
'boy', 'squealed', 'delight', 'hugged', 'told', 'keep', 'not', 'retur
n', 'library', 'daughter', 'loves', 'really', 'rosie', 'books', 'introd
uced', 'cd', 'performed', 'carole', 'king', 'also', 'available', 'amazo
n', 'later', 'knows', 'songs', 'far', 'go', 'one', 'johnny', 'around',
'chicken', 'soup', 'w', 'rice', 'well', 'written', 'clever', 'art', 'wo
rk', 'maurice', 'sendak', 'plus', 'cheap', 'highly']
```

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

[4.4.1.1] Avg W2v

In [71]:

```
# average Word2Vec
# compute average word2vec for each review.
sent_vectors = []; # the avg-w2v for each sentence/review is stored in
```

```
 this list
for sent in tqdm(list_of_sentance): # for each review/sentence
    sent_vec = np.zeros(50) # as word vectors are of zero length 50, yo
u might need to change this to 300 if you use google's w2v
    cnt_words =0; # num of words with a valid vector in the sentence/re
view
    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%|██████████████████████████████████████████████████████████████████
███████| 348260/348260 [20:42<00:00, 280.25it/s]
```

```
348260
50
```

[4.4.1.2] TFIDF weighted W2v

In [ ]:

```
# S = ["abc def pqr", "def def def abc", "pqr pqr def"]
model = TfidfVectorizer()
tf_idf_matrix = model.fit_transform(preprocessed_reviews)
# we are converting a dictionary with word as a key, and the idf as a v
alue
dictionary = dict(zip(model.get_feature_names(), list(model.idf_)))
```

In [ ]:

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

tfidf_sent_vectors = []; # the tfidf-w2v for each sentence/review is st
```

```
ored in this list
row=0;
for sent in tqdm(list_of_sentance): # for each review/sentence
    sent_vec = np.zeros(50) # as word vectors are of zero length
    weight_sum =0; # num of words with a valid vector in the sentence/r
eview
    for word in sent: # for each word in a review/sentence
        if word in w2v_words and word in tfidf_feat:
            vec = w2v_model.wv[word]
#             tf_idf = tf_idf_matrix[row, tfidf_feat.index(word)]
            # to reduce the computation we are
            # dictionary[word] = idf value of word in whole courpus
            # sent.count(word) = tf valeus of word in this review
            tf_idf = dictionary[word]*(sent.count(word)/len(sent))
            sent_vec += (vec * tf_idf)
            weight_sum += tf_idf
    if weight_sum != 0:
        sent_vec /= weight_sum
    tfidf_sent_vectors.append(sent_vec)
    row += 1
```

[5] Assignment 5: Apply Logistic Regression

1. **Apply Logistic Regression 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. **Hyper paramter tuning (find best hyper parameters corresponding the algorithm that you choose)**

    - 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

3. **Pertubation Test**

- Get the weights W after fit your model with the data X i.e Train data.
- Add a noise to the X (X' = X + e) and get the new data set X' (if X is a sparse matrix, X.data+=e)
- Fit the model again on data X' and get the weights W'
- Add a small eps value(to eliminate the divisible by zero error) to W and W' i.e W=W+10^-6 and W' = W'+10^-6
- Now find the % change between W and W' (| (W-W') / (W) |)*100)
- Calculate the 0th, 10th, 20th, 30th, ...100th percentiles, and observe any sudden rise in the values of percentage_change_vector
- Ex: consider your 99th percentile is 1.3 and your 100th percentiles are 34.6, there is sudden rise from 1.3 to 34.6, now calculate the 99.1, 99.2, 99.3,..., 100th percentile values and get the proper value after which there is sudden rise the values, assume it is 2.5
- Print the feature names whose % change is more than a threshold x(in our example it's 2.5)

4. **Sparsity**

- Calculate sparsity on weight vector obtained after using L1 regularization

NOTE: Do sparsity and multicollinearity for any one of the vectorizers. Bow or tf-idf is recommended.

5. **Feature importance**

- Get top 10 important features for both positive and negative classes separately.

6. **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.

7. **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.

8. **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.

Applying Logistic Regression

[5.1] Logistic Regression on BOW, SET 1

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

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

In [74]:
```
connect=sqlite3.connect('final.sqlite')
```

In [75]:
```
cleaned_data= pd.read_sql_query("select * from cleaned", connect)
```

In [76]:
```
cleaned_data.shape
```

Out[76]:
```
(348260, 12)
```

In [77]:
```
cleaned_data['Time'] = pd.to_datetime(cleaned_data['Time'], unit = 's')
cleaned_data= cleaned_data.sort_values(by='Time')
```

In [78]:
```
cleaned_data.head()
```

Out[78]:

	index	Id	ProductId	UserId	ProfileName	HelpfulnessNumerato
26	138706	150524	0006641040	ACITT7DI6IDDL	shari zychinski	0

	index	Id	ProductId	UserId	ProfileName	HelpfulnessNumerato
31	138683	150501	0006641040	AJ46FKXOVC7NR	Nicholas A Mesiano	2
424	417839	451856	B00004CXX9	AIUWLEQ1ADEG5	Elizabeth Medina	0
400	346055	374359	B00004CI84	A344SMIA5JECGM	Vincent P. Ross	1
423	417838	451855	B00004CXX9	AJH6LUC1UT1ON	The Phantom of the Opera	0

In [79]:
```
from sklearn.model_selection import train_test_split
from sklearn.model_selection import GridSearchCV
from sklearn.naive_bayes import MultinomialNB
from sklearn.metrics import accuracy_score
from sklearn.model_selection import cross_val_score
```

In [173]:
```
X = cleaned_data['processed_review']
Y = cleaned_data['Score']
```

In [174]:
```
Y.value_counts()
```

Out[174]:
```
1    293516
0     54744
Name: Score, dtype: int64
```

In [175]:
```
X_train,X_test,Y_train,Y_test = train_test_split(X,Y,test_size=0.3,shuf
fle=False)
```

In [176]:
```
print("The Shape of train data is :",X_train.shape, Y_train.shape)
print("The Shape of test data is :",X_test.shape, Y_test.shape)
```

```
The Shape of train data is : (243782,) (243782,)
The Shape of test data is : (104478,) (104478,)
```

In [414]:
```
#We will make the model learn the vocab from train dataset. We will use
 countvectorizer to serve our purpose
vec= CountVectorizer(min_df = 50)
bow_train = vec.fit_transform(X_train)
bow_test = vec.transform(X_test)


#normalize the data
from sklearn import preprocessing
bow_train = preprocessing.normalize(bow_train)
bow_test = preprocessing.normalize(bow_test)
```

In [178]:
```
print("After Vectorization")
print(bow_train.shape,Y_train.shape)
print(bow_test.shape,Y_test.shape)
```

```
After Vectorization
(243782, 8485) (243782,)
(104478, 8485) (104478,)
```

In [179]:
```
# https://scikit-learn.org/stable/modules/generated/sklearn.linear_mode
l.LogisticRegression.html
from sklearn.model_selection import TimeSeriesSplit
from sklearn.linear_model import LogisticRegression
```

```
lr = LogisticRegression(penalty='l1')
#we will create a dictonary of values using which we will test for n_ne
ighbors
para_grid= {'C':[10000,7000,5000,2000,500,100,50,10,5,1,0.1,0.5,0.01,0.
05,0.005,0.001,0.0005,0.0001,0.00005,0.00001]}
#We will use the gridsearch to test all aforementioned values for alpha
# https://scikit-learn.org/stable/modules/generated/sklearn.model_selec
tion.TimeSeriesSplit.html
ts_cv = TimeSeriesSplit(n_splits =10)
# https://scikit-learn.org/stable/modules/generated/sklearn.model_selec
tion.GridSearchCV.html
lr_gsv= GridSearchCV(lr,para_grid,cv=ts_cv,scoring='roc_auc',n_jobs=-1)
#we will be fitting the optimised k values to the train_bow data
lr_gsv.fit(bow_train,Y_train)
```

Out[179]:

```
GridSearchCV(cv=TimeSeriesSplit(max_train_size=None, n_splits=10),
       error_score='raise-deprecating',
       estimator=LogisticRegression(C=1.0, class_weight=None, dual=Fals
e, fit_intercept=True,
          intercept_scaling=1, max_iter=100, multi_class='warn',
          n_jobs=None, penalty='l1', random_state=None, solver='warn',
          tol=0.0001, verbose=0, warm_start=False),
       fit_params=None, iid='warn', n_jobs=-1,
       param_grid={'C': [10000, 7000, 5000, 2000, 500, 100, 50, 10, 5,
1, 0.1, 0.5, 0.01, 0.05, 0.005, 0.001, 0.0005, 0.0001, 5e-05, 1e-05]},
       pre_dispatch='2*n_jobs', refit=True, return_train_score='warn',
       scoring='roc_auc', verbose=0)
```

In [180]:

```
#The best C value obtained from the gridsearchCV is
best_C= lr_gsv.best_params_
print("Best  Hyperparameter:", best_C)
print("The accuracy obtained by using the hyperparameter is:", lr_gsv.b
est_score_*100)
```

```
Best  Hyperparameter: {'C': 5}
The accuracy obtained by using the hyperparameter is: 94.93468569042966
```

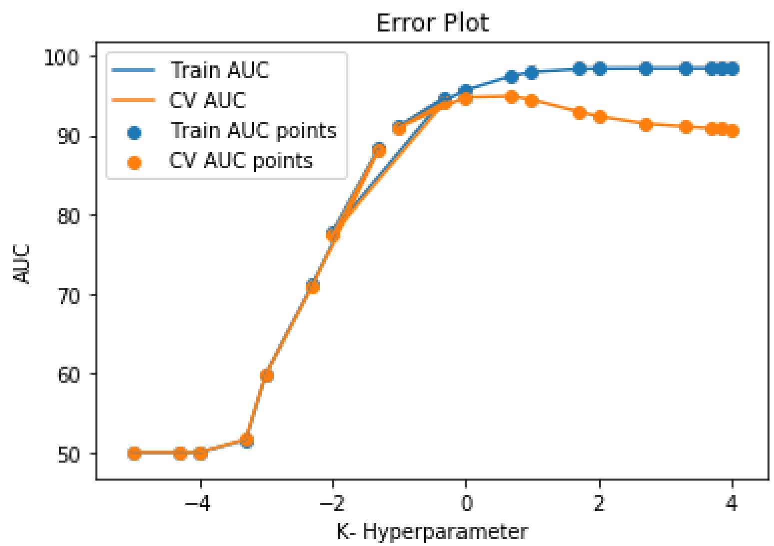
In [181]:

```
train_auc = lr_gsv.cv_results_['mean_train_score']
```

```
cv_auc = lr_gsv.cv_results_['mean_test_score']
for key in para_grid.values():
    key= np.log10(key) # used log scale on X axis for ease of understan
ding
    print(key)
plt.plot(key,train_auc*100,label='Train AUC')
plt.plot(key,cv_auc*100,label='CV AUC')
plt.scatter(key,train_auc*100,label='Train AUC points')
plt.scatter(key,cv_auc*100,label='CV AUC points')
plt.legend()
plt.xlabel('K- Hyperparameter')
plt.ylabel('AUC')
plt.title('Error Plot')
```

```
[ 4.          3.84509804  3.69897     3.30103     2.69897     2.
  1.69897     1.          0.69897     0.          -1.         -0.30103
 -2.         -1.30103    -2.30103    -3.         -3.30103    -4.
 -4.30103    -5.        ]
```

Out[181]: Text(0.5, 1.0, 'Error Plot')



In [182]:
```
#we will select the best hyperparameter and train the model using the p
arameter
```

```
lr = LogisticRegression(penalty = 'l1', C= 5, n_jobs=-1)
lr.fit(bow_train, Y_train)
y_train_pred= lr.predict_proba(bow_train)[:,1]
y_test_pred=lr.predict_proba(bow_test)[:,1]
```

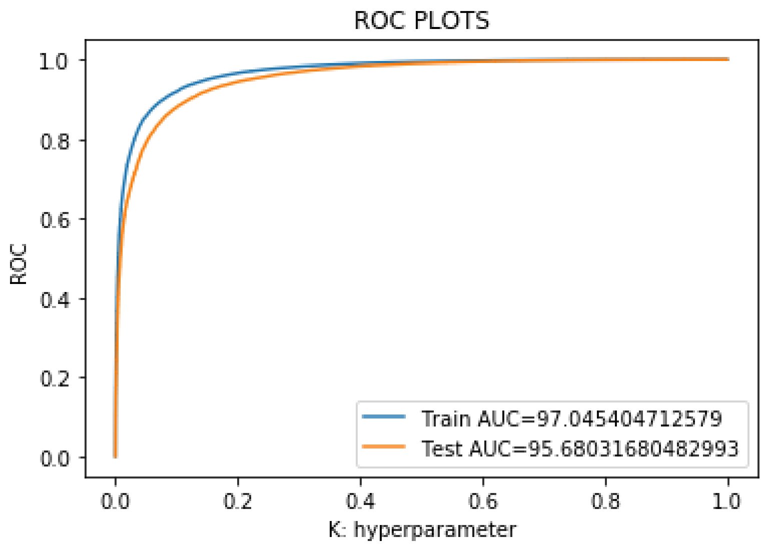
In [183]:

```
train_fpr,train_tpr,thersholds=roc_curve(Y_train,y_train_pred)
test_fpr,test_tpr,thersholds=roc_curve(Y_test,y_test_pred)
```

In [184]:

```
plt.plot(train_fpr,train_tpr,label="Train AUC=" +str(auc(train_fpr,trai
n_tpr)*100))
plt.plot(test_fpr,test_tpr,label="Test AUC=" +str(auc(test_fpr,test_tpr
)*100))
plt.xlabel("K: hyperparameter")
plt.ylabel("ROC")
plt.title("ROC PLOTS")
plt.legend()
plt.legend()
```

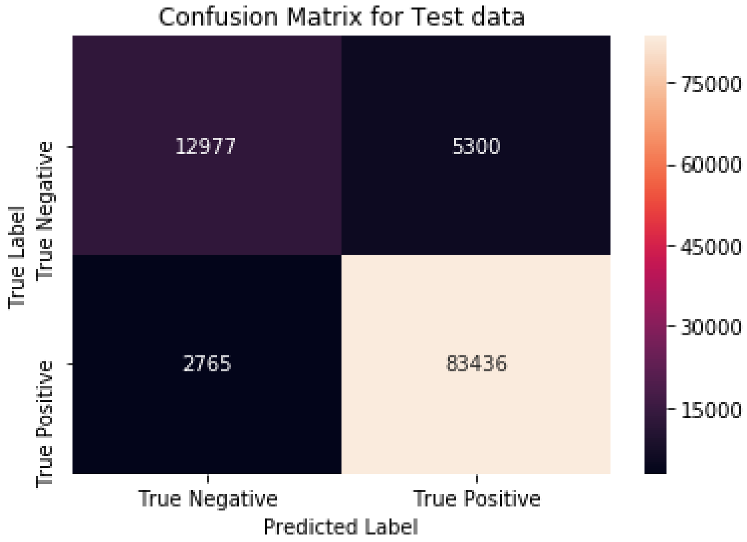
Out[184]: <matplotlib.legend.Legend at 0x252fcfe9860>



In [185]:
```
import seaborn as sns
from sklearn.metrics import classification_report
```

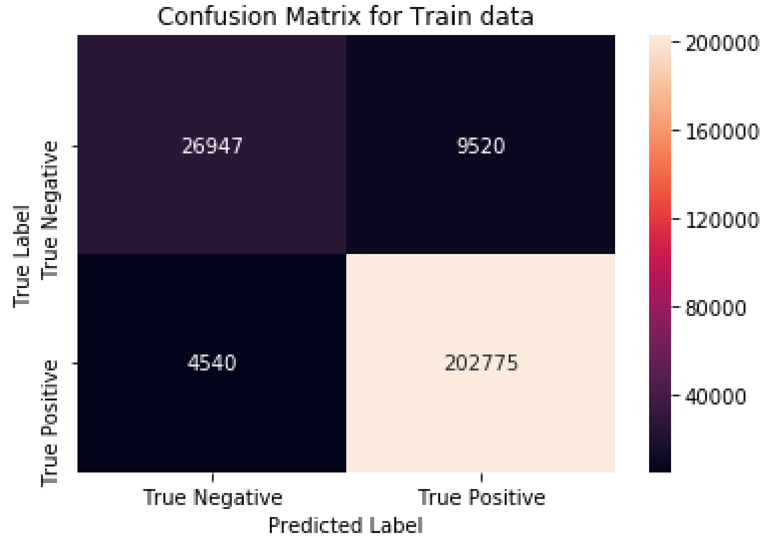
In [186]:
```
class_label = ["True Negative", "True Positive"]
cm=pd.DataFrame(confusion_matrix(Y_test,lr.predict(bow_test)),index=cla
ss_label,columns=class_label)
sns.heatmap(cm,annot=True,fmt="g")
plt.title("Confusion Matrix for Test data")
plt.xlabel("Predicted Label")
plt.ylabel("True Label")
```

Out[186]:
```
Text(33.0, 0.5, 'True Label')
```



In [187]:
```
class_label = ["True Negative", "True Positive"]
cm=pd.DataFrame(confusion_matrix(Y_train,lr.predict(bow_train)),index=c
lass_label,columns=class_label)
sns.heatmap(cm,annot=True,fmt="g")
plt.title("Confusion Matrix for Train data")
plt.xlabel("Predicted Label")
plt.ylabel("True Label")
```

Out[187]:
```
Text(33.0, 0.5, 'True Label')
```



In [188]:
```
print(classification_report(Y_test,lr.predict(bow_test)))
```

```
              precision    recall  f1-score   support

           0       0.82      0.71      0.76     18277
           1       0.94      0.97      0.95     86201

   micro avg       0.92      0.92      0.92    104478
   macro avg       0.88      0.84      0.86    104478
weighted avg       0.92      0.92      0.92    104478
```

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

In [189]:
```
# Please write all the code with proper documentation
```

In [190]:
```
w_before_error = lr.coef_
```

In [192]:
```
print(w_before_error)
```

```
[[0.32915868 1.92045268 0.         ... 0.         0.16918511 0.
]]
```

In [193]:
```
len(w_before_error[0])
```

Out[193]:
```
8485
```

In [194]:
```
bow_train.shape
```

Out[194]:
```
(243782, 8485)
```

In [195]:
```
print(np.count_nonzero(w_before_error))
```

```
6117
```

In [196]:
```
#sparsity percentage = number of zero elements / Total number of elemen
ts

sparse_percent = (np.count_nonzero(w_before_error))/ (len(w_before_erro
r[0]))
print(sparse_percent)
```

```
0.7209192692987625
```

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

In [197]:
```
# Please write all the code with proper documentation
```

In [198]:
```
# https://scikit-learn.org/stable/modules/generated/sklearn.linear_mode
l.LogisticRegression.html
from sklearn.model_selection import TimeSeriesSplit
from sklearn.linear_model import LogisticRegression

lr = LogisticRegression(penalty='l2')
```

```
#we will create a dictonary of values using which we will test for n_ne
ighbors
para_grid= {'C':[10000,7000,5000,2000,500,100,50,10,5,1,0.1,0.5,0.01,0.
05,0.005,0.001,0.0005,0.0001,0.00005,0.00001]}
#We will use the gridsearch to test all aforementioned values for alpha
# https://scikit-learn.org/stable/modules/generated/sklearn.model_selec
tion.TimeSeriesSplit.html
ts_cv = TimeSeriesSplit(n_splits =10)
# https://scikit-learn.org/stable/modules/generated/sklearn.model_selec
tion.GridSearchCV.html
lr_gsv= GridSearchCV(lr,para_grid,cv=ts_cv,scoring='roc_auc',n_jobs=-1)
#we will be fitting the optimised k values to the train_bow data
lr_gsv.fit(bow_train,Y_train)
```

Out[198]:

```
GridSearchCV(cv=TimeSeriesSplit(max_train_size=None, n_splits=10),
       error_score='raise-deprecating',
       estimator=LogisticRegression(C=1.0, class_weight=None, dual=Fals
e, fit_intercept=True,
          intercept_scaling=1, max_iter=100, multi_class='warn',
          n_jobs=None, penalty='l2', random_state=None, solver='warn',
          tol=0.0001, verbose=0, warm_start=False),
       fit_params=None, iid='warn', n_jobs=-1,
       param_grid={'C': [10000, 7000, 5000, 2000, 500, 100, 50, 10, 5,
1, 0.1, 0.5, 0.01, 0.05, 0.005, 0.001, 0.0005, 0.0001, 5e-05, 1e-05]},
       pre_dispatch='2*n_jobs', refit=True, return_train_score='warn',
       scoring='roc_auc', verbose=0)
```

In [199]:

```
#The best C value obtained from the gridsearchCV is
best_C= lr_gsv.best_params_
print("Best  Hyperparameter:", best_C)
print("The accuracy obtained by using the hyperparameter is:", lr_gsv.b
est_score_*100)
```

```
Best  Hyperparameter: {'C': 5}
The accuracy obtained by using the hyperparameter is: 95.1633218228176
```

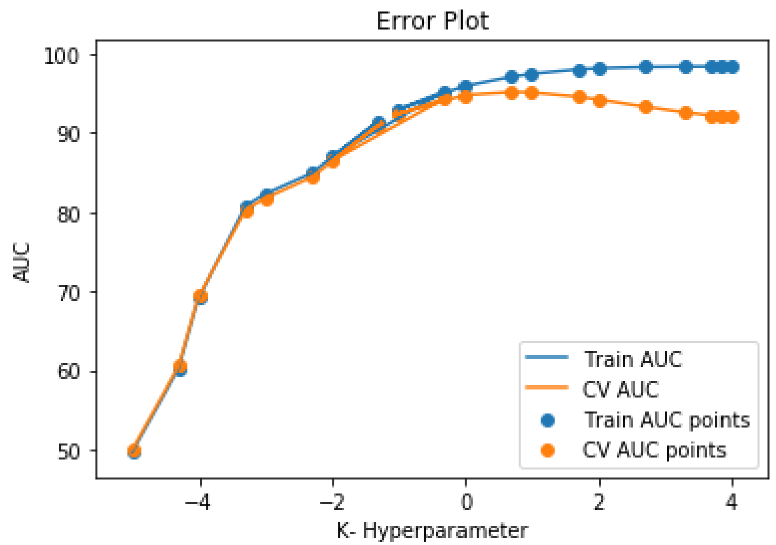
In [200]:

```
train_auc = lr_gsv.cv_results_['mean_train_score']
cv_auc = lr_gsv.cv_results_['mean_test_score']
for key in para_grid.values():
```

```
    key= np.log10(key) # used log scale on X axis for ease of understan
ding
    print(key)
plt.plot(key,train_auc*100,label='Train AUC')
plt.plot(key,cv_auc*100,label='CV AUC')
plt.scatter(key,train_auc*100,label='Train AUC points')
plt.scatter(key,cv_auc*100,label='CV AUC points')
plt.legend()
plt.xlabel('K- Hyperparameter')
plt.ylabel('AUC')
plt.title('Error Plot')
```

```
[ 4.          3.84509804  3.69897     3.30103     2.69897     2.
  1.69897     1.          0.69897     0.          -1.         -0.30103
 -2.         -1.30103    -2.30103    -3.          -3.30103    -4.
 -4.30103    -5.        ]
```

Out[200]: Text(0.5, 1.0, 'Error Plot')



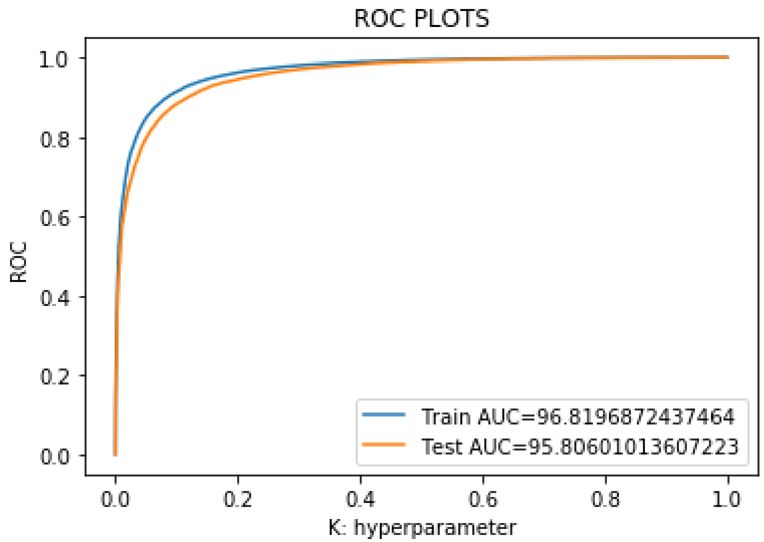
In [201]:
```
#we will select the best hyperparameter and train the model using the p
arameter

lr = LogisticRegression(penalty = 'l2', C= 5, n_jobs=-1)
```

```
lr.fit(bow_train, Y_train)
y_train_pred= lr.predict_proba(bow_train)[:,1]
y_test_pred=lr.predict_proba(bow_test)[:,1]
```

In [202]:
```
train_fpr,train_tpr,thersholds=roc_curve(Y_train,y_train_pred)
test_fpr,test_tpr,thersholds=roc_curve(Y_test,y_test_pred)
```

In [203]:
```
plt.plot(train_fpr,train_tpr,label="Train AUC=" +str(auc(train_fpr,train_tpr)*100))
plt.plot(test_fpr,test_tpr,label="Test AUC=" +str(auc(test_fpr,test_tpr)*100))
plt.xlabel("K: hyperparameter")
plt.ylabel("ROC")
plt.title("ROC PLOTS")
plt.legend()
plt.legend()
```

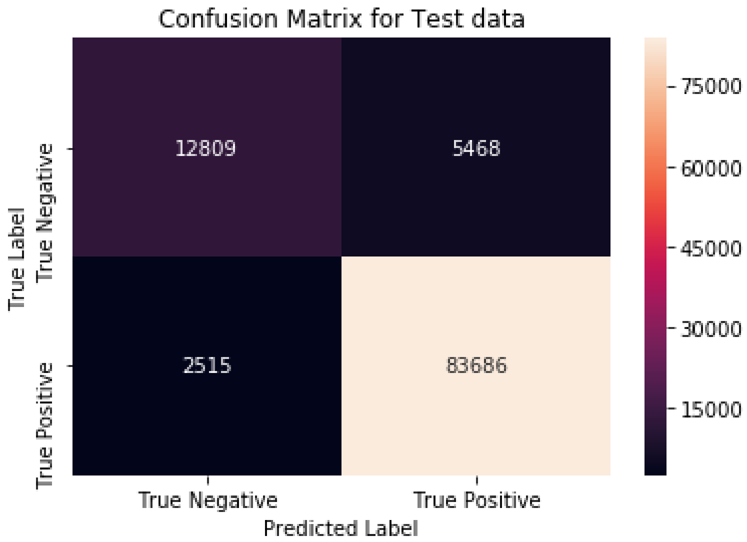
Out[203]: <matplotlib.legend.Legend at 0x252fb7709b0>



In [204]:
```
class_label = ["True Negative", "True Positive"]
cm=pd.DataFrame(confusion_matrix(Y_test,lr.predict(bow_test)),index=cla
```

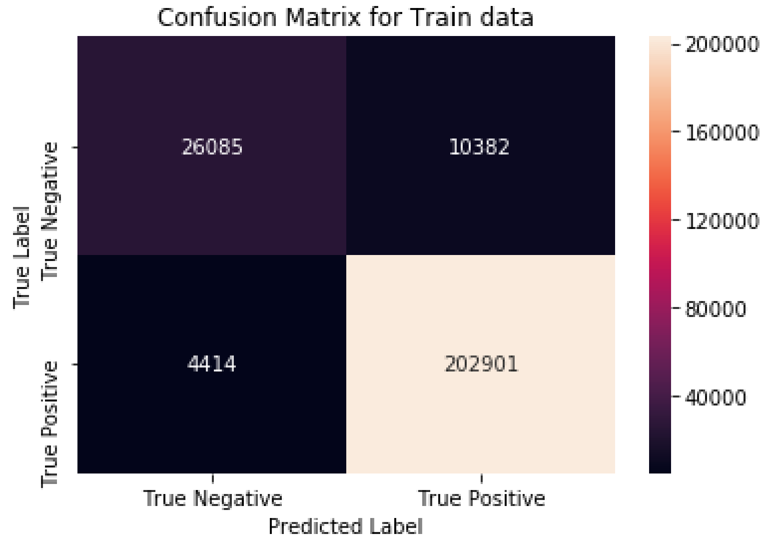
```
ss_label,columns=class_label)
sns.heatmap(cm,annot=True,fmt="g")
plt.title("Confusion Matrix for Test data")
plt.xlabel("Predicted Label")
plt.ylabel("True Label")
```

Out[204]: Text(33.0, 0.5, 'True Label')



In [205]:
```
class_label = ["True Negative", "True Positive"]
cm=pd.DataFrame(confusion_matrix(Y_train,lr.predict(bow_train)),index=c
lass_label,columns=class_label)
sns.heatmap(cm,annot=True,fmt="g")
plt.title("Confusion Matrix for Train data")
plt.xlabel("Predicted Label")
plt.ylabel("True Label")
```

Out[205]: Text(33.0, 0.5, 'True Label')



In [206]:
```
print(classification_report(Y_test,lr.predict(bow_test)))
```

```
              precision    recall  f1-score   support

           0       0.84      0.70      0.76     18277
           1       0.94      0.97      0.95     86201

   micro avg       0.92      0.92      0.92    104478
   macro avg       0.89      0.84      0.86    104478
weighted avg       0.92      0.92      0.92    104478
```

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

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

In [207]:
```
# a. Get the weights W after fit your model with the data X.
w_before_error = lr.coef_
```

In [218]:
```
w_before_error
```

Out[218]:
```
array([[ 0.63396744,  1.86660572,  0.44267429, ..., -0.59728292,
         0.51586997, -0.51198272]])
```

In [209]:
```
# b. Add a noise to the X (X' = X + e) and get the new data set X' (if
 X is a sparse matrix, X.data+=e)

epsilon = 0.001
bow_train.data+=epsilon
```

In [211]:
```
#c. We fit the model again on data X' and get the weights W'
e_lr = LogisticRegression(penalty = 'l2', C= 5, n_jobs=-1)
e_lr.fit(bow_train, Y_train)
```

Out[211]:
```
LogisticRegression(C=5, class_weight=None, dual=False, fit_intercept=Tr
ue,
          intercept_scaling=1, max_iter=100, multi_class='warn', n_jobs
=-1,
          penalty='l2', random_state=None, solver='warn', tol=0.0001,
          verbose=0, warm_start=False)
```

In [212]:
```
#weight vectors after adding noise
w_after_error = e_lr.coef_
```

In [217]:
```
w_after_error
```

Out[217]:
```
array([[ 0.62839668,  1.85521157,  0.45477524, ..., -0.61813027,
         0.51536592, -0.52220339]])
```

In [264]:
```
#  find the % change between W and W', percentage_change_vector = (| (W
-W') / (W) |)*100

percent_change_vector = np.absolute((np.absolute(w_before_error - w_aft
er_error)/(w_before_error)))*100
percent_change_vector
```

Out[264]:
```
array([[0.8787145 , 0.61042078, 2.73359961, ..., 3.49036565, 0.0977089
```

```
5,
        1.99629227]])
```

In [412]:
```
percent_change_vector[0,0:10]
```

Out[412]:
```
array([0.8787145 , 0.61042078, 2.73359961, 1.56349123, 0.20635745,
       0.11012808, 0.65666579, 0.72192608, 0.65761129, 1.24013916])
```

In [269]:
```
percent_change_vector.shape
```

Out[269]:
```
(1, 8485)
```

In [303]:
```
np.percentile(percent_change_vector,98)
```

Out[303]:
```
11.87159924571811
```

In [308]:
```
np.percentile(percent_change_vector,99.4)
```

Out[308]:
```
31.923413547794496
```

We found there is a drastic increase in the change in weight vector from 98 to 99.4.

In [419]:
```
# g.    print the feature names whose % change is more than a threshold
 x(in our example it's 2.5)


#use vectorizer.get_feature_names() to get all the feature names and th
en use np.argsort(clf.coef_[0])[-10:] to get last 10

all_features = vec.get_feature_names()
weight_values = lr.coef_

required_features = np.where(percent_change_vector>np.percentile(percen
t_change_vector,99.4))[1]

list=[]
for i in required_features:
    list.append(all_features[i])
```

```
print(list)
print("No.Of.Features",len(required_features))
```

```
['accompany', 'acting', 'assured', 'balls', 'battle', 'business', 'carr
ot', 'cocoa', 'commitment', 'cozy', 'crackers', 'crew', 'cuts', 'dange
r', 'devoted', 'dunking', 'elevated', 'flower', 'foil', 'gas', 'georgi
a', 'glove', 'hardness', 'hectic', 'holders', 'indulging', 'jersey', 'k
idney', 'meant', 'meanwhile', 'mesh', 'miami', 'mixture', 'mononitrat
e', 'pallet', 'pomeranian', 'pooch', 'premium', 'recognize', 'rural',
'sachet', 'seasoned', 'slow', 'strain', 'tall', 'tapioca', 'tennessee',
'transformed', 'tzu', 'understood', 'unsuspecting']
No.Of.Features 51
```

[5.1.3] Feature Importance on BOW, SET 1

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

In [ ]:

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

In [328]:

```
# https://stackoverflow.com/questions/11116697/how-to-get-most-informat
ive-features-for-scikit-learn-classifiers
def show_most_informative_features(vectorizer, clf, n=10):
    feature_names = vectorizer.get_feature_names()
    coefs_with_fns = sorted(zip(clf.coef_[0], feature_names))
    top =  coefs_with_fns[:-(n + 1):-1]
    for  (coef_2, fn_2) in top:
        print(coef_2, fn_2)
print("Top 10 important features of positive class" )
show_most_informative_features(vec, lr)
```

```
Top 10 important features of positive class
11.93444862707039 pleasantly
10.541322293934403 hooked
9.852664441058494 beat
9.80439619617664 skeptical
9.589867045301164 delicious
9.08901117650765 perfect
```

```
8.6111633696274 excellent
8.604900714097868 amazing
8.417262374602116 highly
8.307718033470485 pleased
```

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

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

In [327]:
```
# https://stackoverflow.com/questions/11116697/how-to-get-most-informat
ive-features-for-scikit-learn-classifiers
def show_most_informative_features(vectorizer, clf, n=10):
    feature_names = vectorizer.get_feature_names()
    coefs_with_fns = sorted(zip(clf.coef_[0], feature_names))
    top = coefs_with_fns[:n]
    for (coef_1, fn_1) in top:
        print( coef_1, fn_1)
print("Top 10 important features of negative class" )
show_most_informative_features(vec, lr)
```

```
Top 10 important features of negative class
-15.341563568282965 worst
-13.326577509540087 disappointing
-13.148736717048337 disappointment
-10.94811967042364 terrible
-10.884158884065132 awful
-9.71091625760773 undrinkable
-9.665038881682237 disgusting
-9.57233262630617 threw
-9.485389472902147 horrible
-9.451215806122635 tasteless
```

[5.2] Logistic Regression on TFIDF, SET 2

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

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

In [329]:
```
#We will make the model learn the vocab from train dataset. We will use
 countvectorizer to serve our purpose
tfidf= TfidfVectorizer(ngram_range =(1,2))
tfidf_train = tfidf.fit_transform(X_train)
tfidf_test = tfidf.transform(X_test)


#normalize the data
from sklearn import preprocessing
tfidf_train = preprocessing.normalize(tfidf_train)
tfidf_test = preprocessing.normalize(tfidf_test)
```

In [330]:
```
print("After Vectorization")
print(tfidf_train.shape,Y_train.shape)
print(tfidf_test.shape,Y_test.shape)
```

```
After Vectorization
(243782, 2939515) (243782,)
(104478, 2939515) (104478,)
```

In [332]:
```
lr_tfidf = LogisticRegression(penalty='l1')
#we will create a dictonary of values using which we will test for n_ne
ighbors
para_grid= {'C':[10000,7000,5000,2000,500,100,50,10,5,1,0.1,0.5,0.01,0.
05,0.005,0.001,0.0005,0.0001,0.00005,0.00001]}
#We will use the gridsearch to test all aforementioned values for alpha
# https://scikit-learn.org/stable/modules/generated/sklearn.model_selec
tion.TimeSeriesSplit.html
ts_cv = TimeSeriesSplit(n_splits =10)
# https://scikit-learn.org/stable/modules/generated/sklearn.model_selec
tion.GridSearchCV.html
lr_gsv= GridSearchCV(lr_tfidf,para_grid,cv=ts_cv,scoring='roc_auc',n_jo
bs=-1)
```

```
#we will be fitting the optimised k values to the train_bow data
lr_gsv.fit(tfidf_train,Y_train)
```

Out[332]:

```
GridSearchCV(cv=TimeSeriesSplit(max_train_size=None, n_splits=10),
       error_score='raise-deprecating',
       estimator=LogisticRegression(C=1.0, class_weight=None, dual=Fals
e, fit_intercept=True,
          intercept_scaling=1, max_iter=100, multi_class='warn',
          n_jobs=None, penalty='l1', random_state=None, solver='warn',
          tol=0.0001, verbose=0, warm_start=False),
       fit_params=None, iid='warn', n_jobs=-1,
       param_grid={'C': [10000, 7000, 5000, 2000, 500, 100, 50, 10, 5,
1, 0.1, 0.5, 0.01, 0.05, 0.005, 0.001, 0.0005, 0.0001, 5e-05, 1e-05]},
       pre_dispatch='2*n_jobs', refit=True, return_train_score='warn',
       scoring='roc_auc', verbose=0)
```

In [333]:

```
#The best C value obtained from the gridsearchCV is
best_C= lr_gsv.best_params_
print("Best  Hyperparameter:", best_C)
print("The accuracy obtained by using the hyperparameter is:", lr_gsv.b
est_score_*100)
```

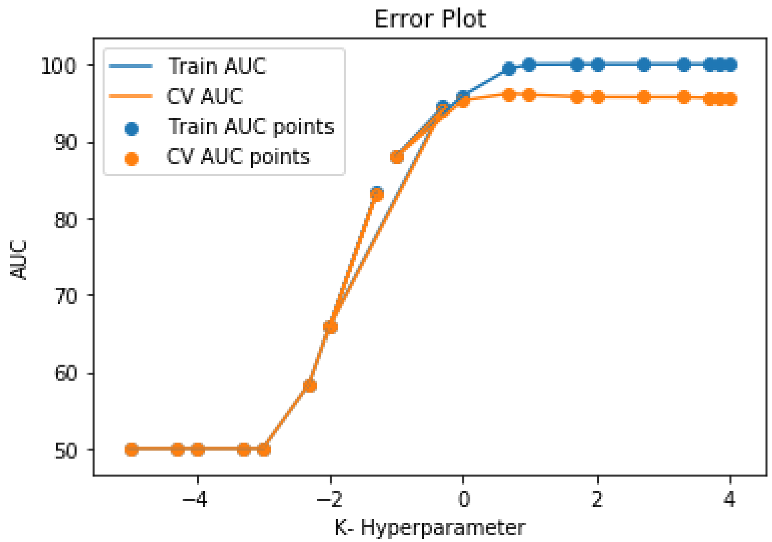
```
Best  Hyperparameter: {'C': 5}
The accuracy obtained by using the hyperparameter is: 96.2142470003127
```

In [334]:

```
train_auc = lr_gsv.cv_results_['mean_train_score']
cv_auc = lr_gsv.cv_results_['mean_test_score']
for key in para_grid.values():
    key= np.log10(key) # used log scale on X axis for ease of understan
ding
    print(key)
plt.plot(key,train_auc*100,label='Train AUC')
plt.plot(key,cv_auc*100,label='CV AUC')
plt.scatter(key,train_auc*100,label='Train AUC points')
plt.scatter(key,cv_auc*100,label='CV AUC points')
plt.legend()
plt.xlabel('K- Hyperparameter')
plt.ylabel('AUC')
plt.title('Error Plot')
```

```
[ 4.          3.84509804  3.69897     3.30103     2.69897     2.
  1.69897     1.          0.69897     0.         -1.         -0.30103
 -2.         -1.30103    -2.30103    -3.         -3.30103    -4.
 -4.30103    -5.        ]
```

Out[334]: `Text(0.5, 1.0, 'Error Plot')`



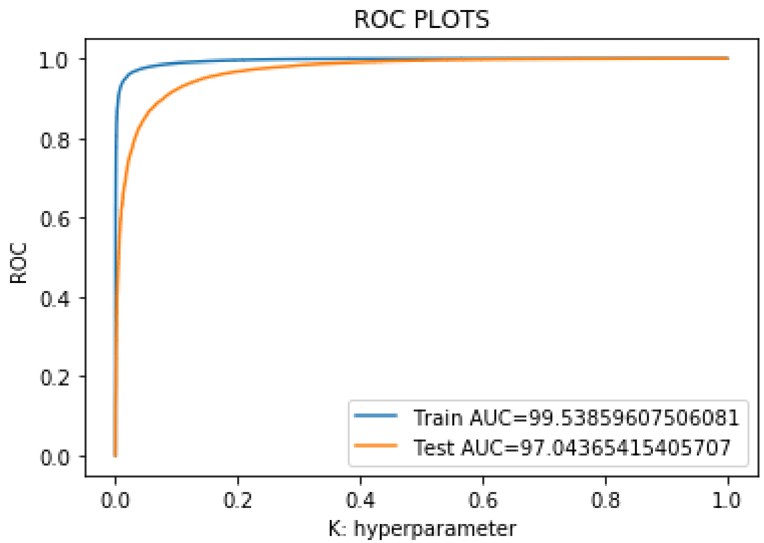
In [335]:
```
#we will select the best hyperparameter and train the model using the p
arameter

lr = LogisticRegression(penalty = 'l1', C= 5, n_jobs=-1)
lr.fit(tfidf_train, Y_train)
y_train_pred= lr.predict_proba(tfidf_train)[:,1]
y_test_pred=lr.predict_proba(tfidf_test)[:,1]
```

In [336]:
```
train_fpr,train_tpr,thersholds=roc_curve(Y_train,y_train_pred)
test_fpr,test_tpr,thersholds=roc_curve(Y_test,y_test_pred)
```

In [337]:
```
plt.plot(train_fpr,train_tpr,label="Train AUC=" +str(auc(train_fpr,trai
n_tpr)*100))
plt.plot(test_fpr,test_tpr,label="Test AUC=" +str(auc(test_fpr,test_tpr
```

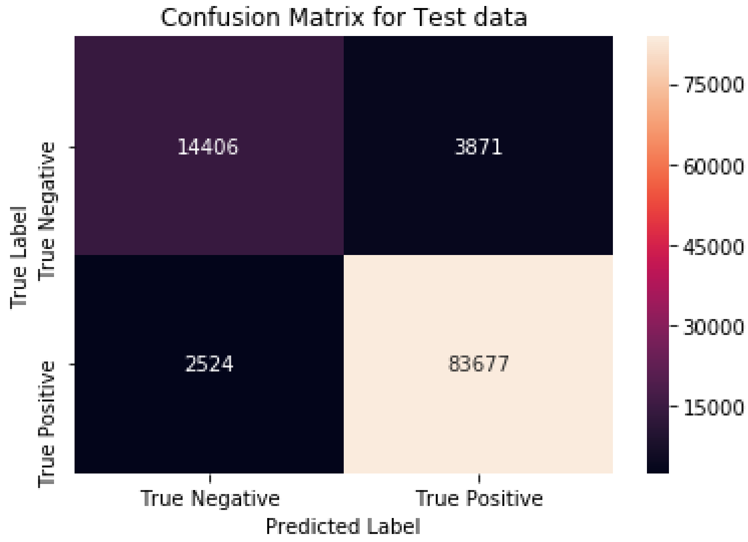
```
)*100))
plt.xlabel("K: hyperparameter")
plt.ylabel("ROC")
plt.title("ROC PLOTS")
plt.legend()
plt.legend()
```

Out[337]: <matplotlib.legend.Legend at 0x2530cfb6eb8>



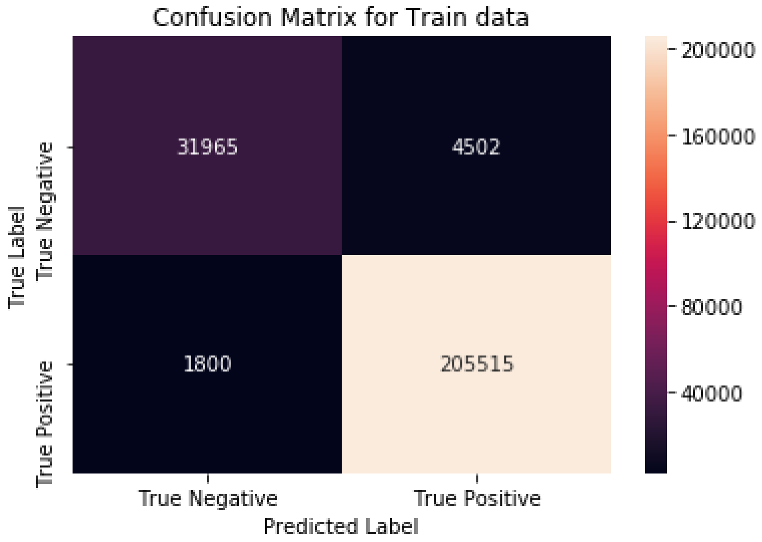
In [338]:
```
class_label = ["True Negative", "True Positive"]
cm=pd.DataFrame(confusion_matrix(Y_test,lr.predict(tfidf_test)),index=c
lass_label,columns=class_label)
sns.heatmap(cm,annot=True,fmt="g")
plt.title("Confusion Matrix for Test data")
plt.xlabel("Predicted Label")
plt.ylabel("True Label")
```

Out[338]: Text(33.0, 0.5, 'True Label')



In [339]:
```
class_label = ["True Negative", "True Positive"]
cm=pd.DataFrame(confusion_matrix(Y_train,lr.predict(tfidf_train)),index
=class_label,columns=class_label)
sns.heatmap(cm,annot=True,fmt="g")
plt.title("Confusion Matrix for Train data")
plt.xlabel("Predicted Label")
plt.ylabel("True Label")
```

Out[339]:
```
Text(33.0, 0.5, 'True Label')
```



In [340]:
```
print(classification_report(Y_test,lr.predict(tfidf_test)))
```

```
              precision    recall  f1-score   support

           0       0.85      0.79      0.82     18277
           1       0.96      0.97      0.96     86201

   micro avg       0.94      0.94      0.94    104478
   macro avg       0.90      0.88      0.89    104478
weighted avg       0.94      0.94      0.94    104478
```

[5.2.2] Applying Logistic Regression with L2 regularization on TFIDF, SET 2

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

In [341]:
```
lr_tfidf_l2 = LogisticRegression(penalty='l2')
#we will create a dictonary of values using which we will test for n_ne
ighbors
```

```
para_grid= {'C':[10000,7000,5000,2000,500,100,50,10,5,1,0.1,0.5,0.01,0.
05,0.005,0.001,0.0005,0.0001,0.00005,0.00001]}
#We will use the gridsearch to test all aforementioned values for alpha
# https://scikit-learn.org/stable/modules/generated/sklearn.model_selec
tion.TimeSeriesSplit.html
ts_cv = TimeSeriesSplit(n_splits =10)
# https://scikit-learn.org/stable/modules/generated/sklearn.model_selec
tion.GridSearchCV.html
lr_gsv_l2= GridSearchCV(lr_tfidf_l2,para_grid,cv=ts_cv,scoring='roc_au
c',n_jobs=-1)
#we will be fitting the optimised k values to the train_bow data
lr_gsv_l2.fit(tfidf_train,Y_train)
```

Out[341]:

```
GridSearchCV(cv=TimeSeriesSplit(max_train_size=None, n_splits=10),
       error_score='raise-deprecating',
       estimator=LogisticRegression(C=1.0, class_weight=None, dual=Fals
e, fit_intercept=True,
          intercept_scaling=1, max_iter=100, multi_class='warn',
          n_jobs=None, penalty='l2', random_state=None, solver='warn',
          tol=0.0001, verbose=0, warm_start=False),
       fit_params=None, iid='warn', n_jobs=-1,
       param_grid={'C': [10000, 7000, 5000, 2000, 500, 100, 50, 10, 5,
1, 0.1, 0.5, 0.01, 0.05, 0.005, 0.001, 0.0005, 0.0001, 5e-05, 1e-05]},
       pre_dispatch='2*n_jobs', refit=True, return_train_score='warn',
       scoring='roc_auc', verbose=0)
```

In [360]:

```
#The best C value obtained from the gridsearchCV is
best_C= lr_gsv_l2.best_params_
print("Best  Hyperparameter:", best_C)
print("The accuracy obtained by using the hyperparameter is:", lr_gsv_l
2.best_score_*100)
```

```
Best  Hyperparameter: {'C': 100}
The accuracy obtained by using the hyperparameter is: 96.54518475730295
```

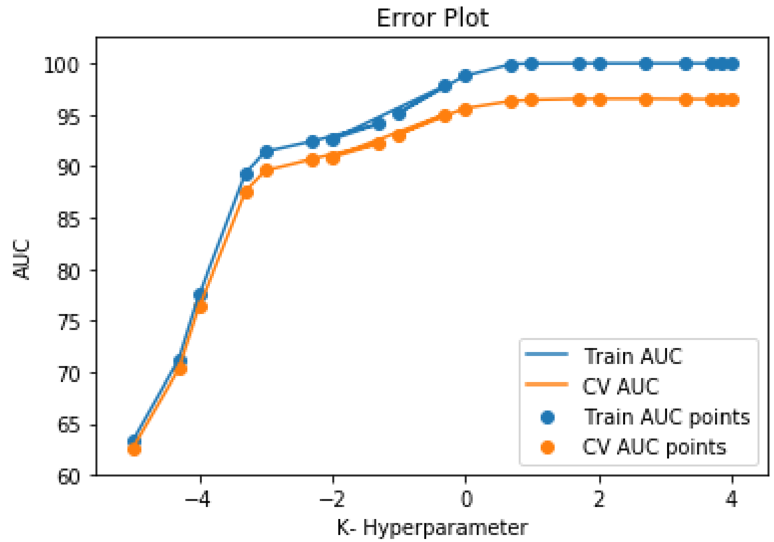
In [374]:

```
train_auc = lr_gsv_l2.cv_results_['mean_train_score']
cv_auc = lr_gsv_l2.cv_results_['mean_test_score']
for key in para_grid.values():
    key= np.log10(key) # used log scale on X axis for ease of understan
```

```
ding
    print(key)
plt.plot(key,train_auc*100,label='Train AUC')
plt.plot(key,cv_auc*100,label='CV AUC')
plt.scatter(key,train_auc*100,label='Train AUC points')
plt.scatter(key,cv_auc*100,label='CV AUC points')
plt.legend()
plt.xlabel('K- Hyperparameter')
plt.ylabel('AUC')
plt.title('Error Plot')
```

```
[ 4.          3.84509804  3.69897     3.30103     2.69897     2.
  1.69897     1.          0.69897     0.         -1.         -0.30103
 -2.         -1.30103    -2.30103    -3.         -3.30103    -4.
 -4.30103    -5.        ]
```

Out[374]: Text(0.5, 1.0, 'Error Plot')



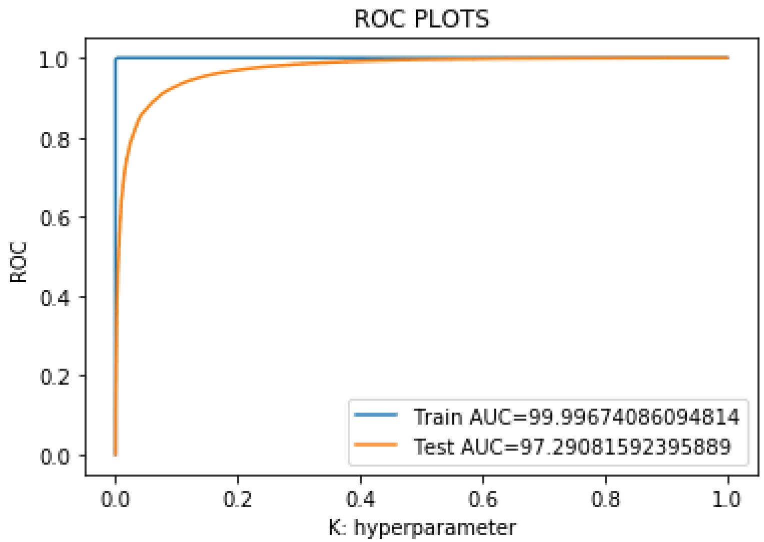
In [359]:
```
#we will select the best hyperparameter and train the model using the p
arameter

lr = LogisticRegression(penalty = 'l2', C= 100, n_jobs=-1)
lr.fit(tfidf_train, Y_train)
```

```
y_train_pred= lr.predict_proba(tfidf_train)[:,1]
y_test_pred=lr.predict_proba(tfidf_test)[:,1]
```

In [361]:
```
train_fpr,train_tpr,thersholds=roc_curve(Y_train,y_train_pred)
test_fpr,test_tpr,thersholds=roc_curve(Y_test,y_test_pred)
```

In [362]:
```
plt.plot(train_fpr,train_tpr,label="Train AUC=" +str(auc(train_fpr,trai
n_tpr)*100))
plt.plot(test_fpr,test_tpr,label="Test AUC=" +str(auc(test_fpr,test_tpr
)*100))
plt.xlabel("K: hyperparameter")
plt.ylabel("ROC")
plt.title("ROC PLOTS")
plt.legend()
plt.legend()
```

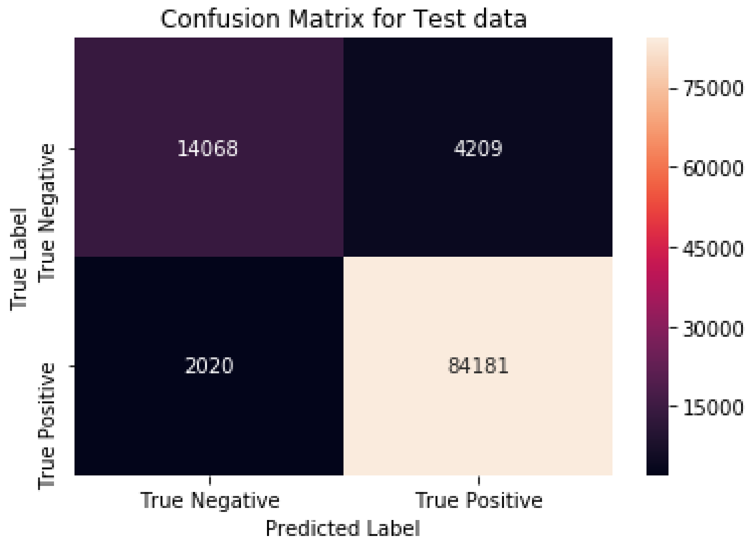
Out[362]: <matplotlib.legend.Legend at 0x253b33a35c0>



In [363]:
```
class_label = ["True Negative", "True Positive"]
cm=pd.DataFrame(confusion_matrix(Y_test,lr.predict(tfidf_test)),index=c
lass_label,columns=class_label)
```

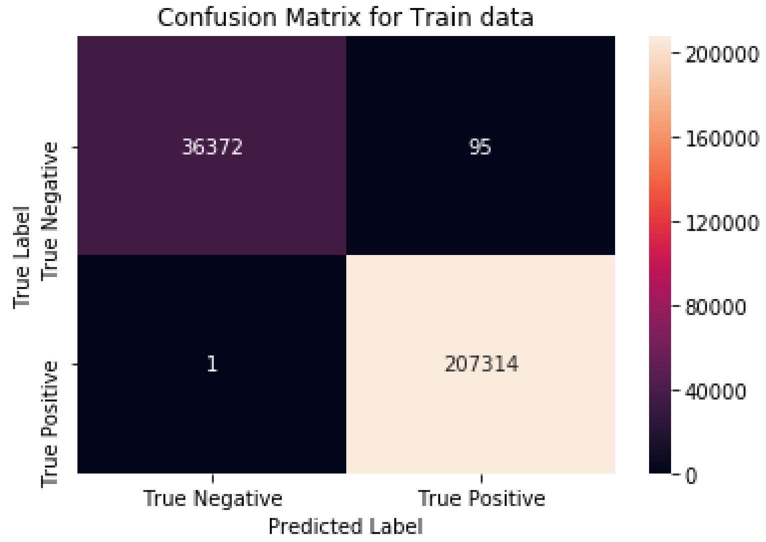
```
sns.heatmap(cm,annot=True,fmt="g")
plt.title("Confusion Matrix for Test data")
plt.xlabel("Predicted Label")
plt.ylabel("True Label")
```

Out[363]: Text(33.0, 0.5, 'True Label')



In [364]:
```
class_label = ["True Negative", "True Positive"]
cm=pd.DataFrame(confusion_matrix(Y_train,lr.predict(tfidf_train)),index
=class_label,columns=class_label)
sns.heatmap(cm,annot=True,fmt="g")
plt.title("Confusion Matrix for Train data")
plt.xlabel("Predicted Label")
plt.ylabel("True Label")
```

Out[364]: Text(33.0, 0.5, 'True Label')



In [365]:
```
print(classification_report(Y_test,lr.predict(tfidf_test)))
```

```
              precision    recall  f1-score   support

           0       0.87      0.77      0.82     18277
           1       0.95      0.98      0.96     86201

   micro avg       0.94      0.94      0.94    104478
   macro avg       0.91      0.87      0.89    104478
weighted avg       0.94      0.94      0.94    104478
```

[5.2.3] Feature Importance on TFIDF, SET 2

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

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

In [350]:
```
# https://stackoverflow.com/questions/11116697/how-to-get-most-informat
ive-features-for-scikit-learn-classifiers
def show_most_informative_features(vectorizer, clf, n=10):
    feature_names = vectorizer.get_feature_names()
    coefs_with_fns = sorted(zip(clf.coef_[0], feature_names))
    top =  coefs_with_fns[:-(n + 1):-1]
    for  (coef_2, fn_2) in top:
        print(coef_2, fn_2)
print("Top 10 important features of positive class" )
show_most_informative_features(tfidf, lr)
```

```
Top 10 important features of positive class
24.60445414426393 great
21.0451856914382 delicious
19.378803744568785 best
17.18331714788054 not disappointed
17.153713507721775 perfect
17.15134939448816 good
15.491112491736846 excellent
14.625954353878695 loves
13.952860806781834 wonderful
13.92165254334985 love
```

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

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

In [351]:
```
# https://stackoverflow.com/questions/11116697/how-to-get-most-informat
ive-features-for-scikit-learn-classifiers
def show_most_informative_features(vectorizer, clf, n=10):
    feature_names = vectorizer.get_feature_names()
    coefs_with_fns = sorted(zip(clf.coef_[0], feature_names))
    top = coefs_with_fns[:n]
    for (coef_1, fn_1) in top:
        print( coef_1, fn_1)
print("Top 10 important features of negative class" )
show_most_informative_features(tfidf, lr)
```

```
Top 10 important features of negative class
-19.5138357416574 worst
-18.625822608825764 disappointed
-17.609838379216477 not worth
-16.74789769835786 disappointing
-16.131435800675828 not recommend
-15.875137244293708 not good
-15.425312325387498 terrible
-15.171164158181037 awful
-14.536659739544058 disappointment
-13.753862605974044 horrible
```

[5.3] Logistic Regression on AVG W2V, SET 3

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

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

In [352]:
```
# Train your own Word2Vec model using your own text corpus
i=0
sentance_train=[]
for sentance in X_train:
    sentance_train.append(sentance.split())
```

In [353]:
```
# to train the W2V model on the provided list of sentences
w2v_model=Word2Vec(sentance_train,min_count=5,size=50, workers=4)
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  28027
sample words  ['witty', 'little', 'book', 'makes', 'son', 'laugh', 'lou
d', 'car', 'driving', 'along', 'always', 'sing', 'refrain', 'learned',
'whales', 'india', 'drooping', 'roses', 'love', 'new', 'words', 'introd
```

```
uces', 'silliness', 'classic', 'willing', 'bet', 'still', 'able', 'memo
ry', 'college', 'remember', 'seeing', 'show', 'aired', 'television', 'y
ears', 'ago', 'child', 'sister', 'later', 'bought', 'day', 'thirty', 's
omething', 'used', 'series', 'books', 'songs', 'student', 'teaching']
```

In [ ]:

```
# we will convert the test data into W2V
# average Word2Vec
# compute average word2vec for each review.
vectors_train = []; # the avg-w2v for each sentence/review is stored in
 this list
for sent in tqdm(sentance_train): # for each review/sentence
    sent_vec = np.zeros(50) # as word vectors are of zero length 50, yo
u might need to change this to 300 if you use google's w2v
    cnt_words =0; # num of words with a valid vector in the sentence/re
view
    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
    vectors_train.append(sent_vec)
print(len(vectors_train))
print(len(vectors_train[0]))
```

In [355]:

```
# Converting the test data text
i=0
sentance_test=[]
for sentance in X_test:
    sentance_test.append(sentance.split())
```

In [ ]:

```
# we will convert the test data into W2V
# average Word2Vec
# compute average word2vec for each review.
vectors_test = []; # the avg-w2v for each sentence/review is stored in
 this list
for sent in tqdm(sentance_test): # for each review/sentence
```

```
    sent_vec = np.zeros(50) # as word vectors are of zero length 50, yo
u might need to change this to 300 if you use google's w2v
    cnt_words =0; # num of words with a valid vector in the sentence/re
view
    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
    vectors_test.append(sent_vec)
print(len(vectors_test))
print(len(vectors_test[0]))
```

In [366]:
```
lr_avg = LogisticRegression(penalty='l1')
#we will create a dictonary of values using which we will test for n_ne
ighbors
para_grid= {'C':[10000,7000,5000,2000,500,100,50,10,5,1,0.1,0.5,0.01,0.
05,0.005,0.001,0.0005,0.0001,0.00005,0.00001]}
#We will use the gridsearch to test all aforementioned values for alpha
# https://scikit-learn.org/stable/modules/generated/sklearn.model_selec
tion.TimeSeriesSplit.html
ts_cv = TimeSeriesSplit(n_splits =10)
# https://scikit-learn.org/stable/modules/generated/sklearn.model_selec
tion.GridSearchCV.html
lr_gsv_avg= GridSearchCV(lr_avg,para_grid,cv=ts_cv,scoring='roc_auc',n_
jobs=-1)
#we will be fitting the optimised k values to the train_bow data
lr_gsv_avg.fit(vectors_train,Y_train)
```

Out[366]:
```
GridSearchCV(cv=TimeSeriesSplit(max_train_size=None, n_splits=10),
       error_score='raise-deprecating',
       estimator=LogisticRegression(C=1.0, class_weight=None, dual=Fals
e, fit_intercept=True,
          intercept_scaling=1, max_iter=100, multi_class='warn',
          n_jobs=None, penalty='l1', random_state=None, solver='warn',
          tol=0.0001, verbose=0, warm_start=False),
       fit_params=None, iid='warn', n_jobs=-1,
```

```
             param_grid={'C': [10000, 7000, 5000, 2000, 500, 100, 50, 10, 5,
1, 0.1, 0.5, 0.01, 0.05, 0.005, 0.001, 0.0005, 0.0001, 5e-05, 1e-05]},
             pre_dispatch='2*n_jobs', refit=True, return_train_score='warn',
             scoring='roc_auc', verbose=0)
```

In [368]:

```
#The best C value obtained from the gridsearchCV is
best_C= lr_gsv_avg.best_params_
print("Best  Hyperparameter:", best_C)
print("The accuracy obtained by using the hyperparameter is:", lr_gsv_a
vg.best_score_*100)
```

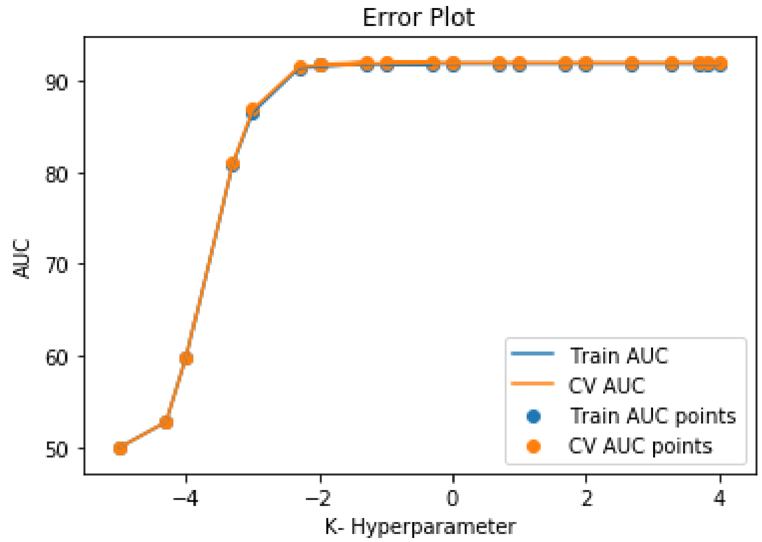
```
Best  Hyperparameter: {'C': 0.5}
The accuracy obtained by using the hyperparameter is: 91.92869890453761
```

In [370]:

```
train_auc = lr_gsv_avg.cv_results_['mean_train_score']
cv_auc = lr_gsv_avg.cv_results_['mean_test_score']
for key in para_grid.values():
    key= np.log10(key) # used log scale on X axis for ease of understan
ding
    print(key)
plt.plot(key,train_auc*100,label='Train AUC')
plt.plot(key,cv_auc*100,label='CV AUC')
plt.scatter(key,train_auc*100,label='Train AUC points')
plt.scatter(key,cv_auc*100,label='CV AUC points')
plt.legend()
plt.xlabel('K- Hyperparameter')
plt.ylabel('AUC')
plt.title('Error Plot')
```

```
[ 4.          3.84509804  3.69897     3.30103     2.69897     2.
  1.69897     1.          0.69897     0.         -1.         -0.30103
 -2.         -1.30103    -2.30103    -3.         -3.30103    -4.
 -4.30103    -5.        ]
```

Out[370]: `Text(0.5, 1.0, 'Error Plot')`

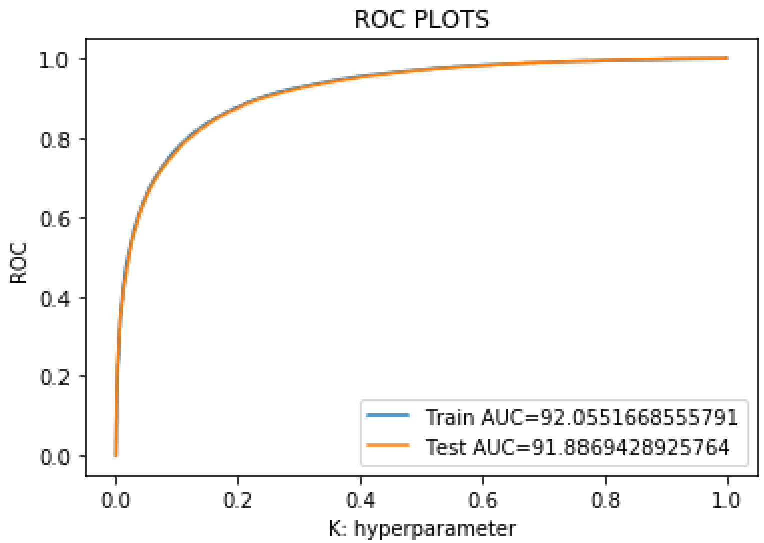


In [371]:
```
#we will select the best hyperparameter and train the model using the p
arameter

lr = LogisticRegression(penalty = 'l1', C= 0.5, n_jobs=-1)
lr.fit(vectors_train, Y_train)
y_train_pred= lr.predict_proba(vectors_train)[:,1]
y_test_pred=lr.predict_proba(vectors_test)[:,1]
```

In [372]:
```
train_fpr,train_tpr,thersholds=roc_curve(Y_train,y_train_pred)
test_fpr,test_tpr,thersholds=roc_curve(Y_test,y_test_pred)
```

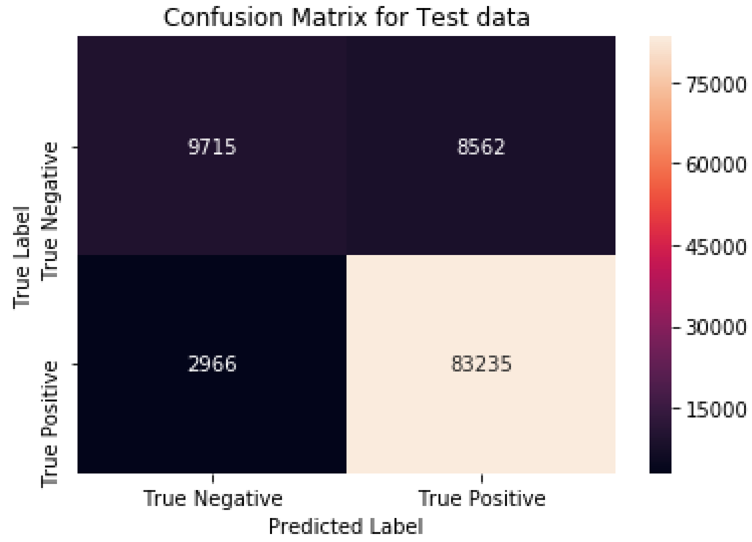
In [373]:
```
plt.plot(train_fpr,train_tpr,label="Train AUC=" +str(auc(train_fpr,trai
n_tpr)*100))
plt.plot(test_fpr,test_tpr,label="Test AUC=" +str(auc(test_fpr,test_tpr
)*100))
plt.xlabel("K: hyperparameter")
plt.ylabel("ROC")
plt.title("ROC PLOTS")
plt.legend()
plt.legend()
```

Out[373]: <matplotlib.legend.Legend at 0x253c381a320>



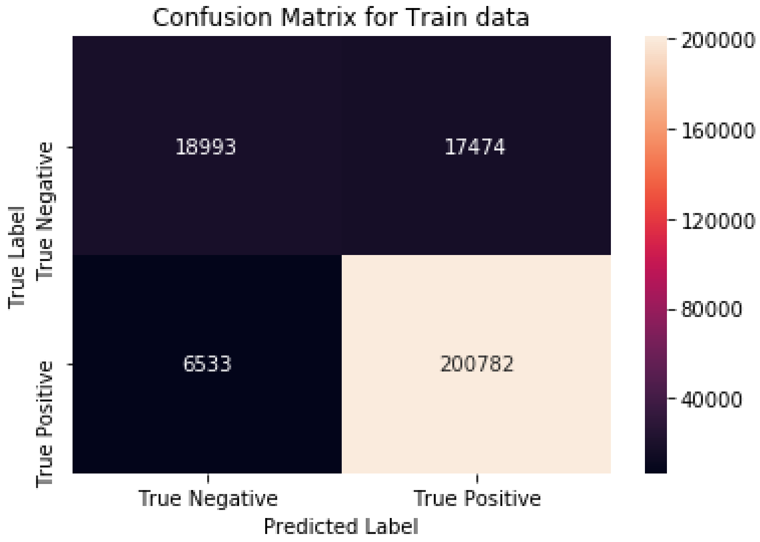
In [375]:
```
class_label = ["True Negative", "True Positive"]
cm=pd.DataFrame(confusion_matrix(Y_test,lr.predict(vectors_test)),index
=class_label,columns=class_label)
sns.heatmap(cm,annot=True,fmt="g")
plt.title("Confusion Matrix for Test data")
plt.xlabel("Predicted Label")
plt.ylabel("True Label")
```

Out[375]: Text(33.0, 0.5, 'True Label')



In [376]:
```
class_label = ["True Negative", "True Positive"]
cm=pd.DataFrame(confusion_matrix(Y_train,lr.predict(vectors_train)),ind
ex=class_label,columns=class_label)
sns.heatmap(cm,annot=True,fmt="g")
plt.title("Confusion Matrix for Train data")
plt.xlabel("Predicted Label")
plt.ylabel("True Label")
```

Out[376]: Text(33.0, 0.5, 'True Label')



In [377]:
```
print(classification_report(Y_test,lr.predict(vectors_test)))
```

```
              precision    recall  f1-score   support

           0       0.77      0.53      0.63     18277
           1       0.91      0.97      0.94     86201

   micro avg       0.89      0.89      0.89    104478
   macro avg       0.84      0.75      0.78    104478
weighted avg       0.88      0.89      0.88    104478
```

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

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

In [390]:
```
lr_avg = LogisticRegression(penalty='l2')
#we will create a dictonary of values using which we will test for n_ne
ighbors
```

```
para_grid= {'C':[10000,7000,5000,2000,500,100,50,10,5,1,0.1,0.5,0.01,0.05,0.005,0.001,0.0005,0.0001,0.00005,0.00001]}
#We will use the gridsearch to test all aforementioned values for alpha
# https://scikit-learn.org/stable/modules/generated/sklearn.model_selection.TimeSeriesSplit.html
ts_cv = TimeSeriesSplit(n_splits =10)
# https://scikit-learn.org/stable/modules/generated/sklearn.model_selection.GridSearchCV.html
lr_gsv_avg= GridSearchCV(lr_avg,para_grid,cv=ts_cv,scoring='roc_auc',n_jobs=-1)
#we will be fitting the optimised k values to the train_bow data
lr_gsv_avg.fit(vectors_train,Y_train)
```

Out[390]:

```
GridSearchCV(cv=TimeSeriesSplit(max_train_size=None, n_splits=10),
       error_score='raise-deprecating',
       estimator=LogisticRegression(C=1.0, class_weight=None, dual=False, fit_intercept=True,
          intercept_scaling=1, max_iter=100, multi_class='warn',
          n_jobs=None, penalty='l2', random_state=None, solver='warn',
          tol=0.0001, verbose=0, warm_start=False),
       fit_params=None, iid='warn', n_jobs=-1,
       param_grid={'C': [10000, 7000, 5000, 2000, 500, 100, 50, 10, 5, 1, 0.1, 0.5, 0.01, 0.05, 0.005, 0.001, 0.0005, 0.0001, 5e-05, 1e-05]},
       pre_dispatch='2*n_jobs', refit=True, return_train_score='warn',
       scoring='roc_auc', verbose=0)
```

In [391]:

```
#The best C value obtained from the gridsearchCV is
best_C= lr_gsv_avg.best_params_
print("Best  Hyperparameter:", best_C)
print("The accuracy obtained by using the hyperparameter is:", lr_gsv_avg.best_score_*100)
```

```
Best  Hyperparameter: {'C': 0.05}
The accuracy obtained by using the hyperparameter is: 91.93185396683194
```

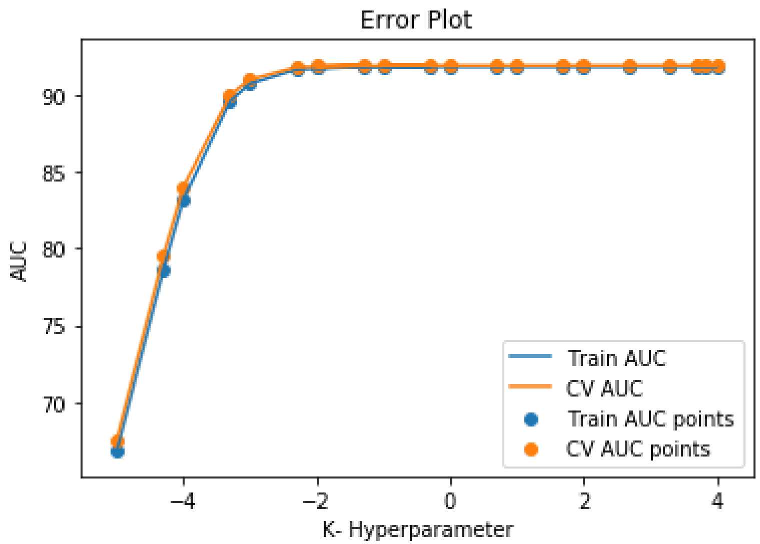
In [392]:

```
train_auc = lr_gsv_avg.cv_results_['mean_train_score']
cv_auc = lr_gsv_avg.cv_results_['mean_test_score']
for key in para_grid.values():
    key= np.log10(key) # used log scale on X axis for ease of understan
```

```
ding
    print(key)
plt.plot(key,train_auc*100,label='Train AUC')
plt.plot(key,cv_auc*100,label='CV AUC')
plt.scatter(key,train_auc*100,label='Train AUC points')
plt.scatter(key,cv_auc*100,label='CV AUC points')
plt.legend()
plt.xlabel('K- Hyperparameter')
plt.ylabel('AUC')
plt.title('Error Plot')
```

```
[ 4.          3.84509804  3.69897     3.30103     2.69897     2.
  1.69897     1.          0.69897     0.         -1.         -0.30103
 -2.         -1.30103    -2.30103    -3.         -3.30103    -4.
 -4.30103    -5.        ]
```

Out[392]: Text(0.5, 1.0, 'Error Plot')



In [393]:

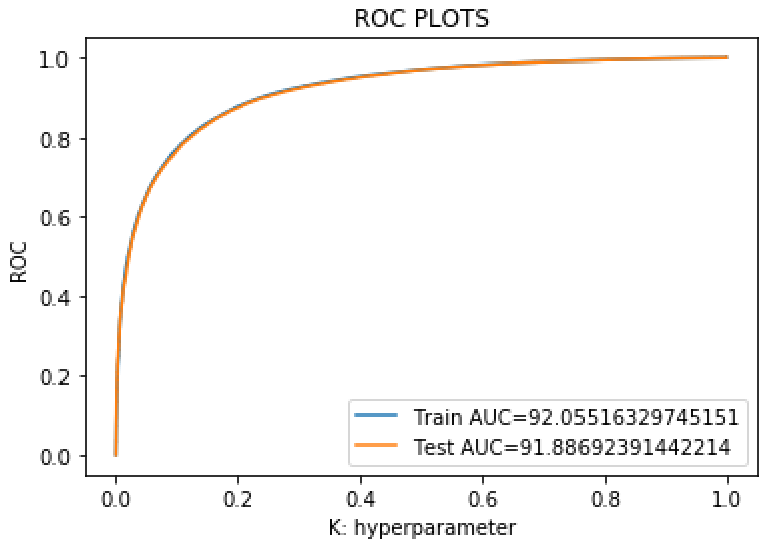
```
#we will select the best hyperparameter and train the model using the p
arameter

lr = LogisticRegression(penalty = 'l2', C= 0.05, n_jobs=-1)
lr.fit(vectors_train, Y_train)
```

```
y_train_pred= lr.predict_proba(vectors_train)[:,1]
y_test_pred=lr.predict_proba(vectors_test)[:,1]
```

In [394]:
```
train_fpr,train_tpr,thersholds=roc_curve(Y_train,y_train_pred)
test_fpr,test_tpr,thersholds=roc_curve(Y_test,y_test_pred)
```

In [395]:
```
plt.plot(train_fpr,train_tpr,label="Train AUC=" +str(auc(train_fpr,train_tpr)*100))
plt.plot(test_fpr,test_tpr,label="Test AUC=" +str(auc(test_fpr,test_tpr)*100))
plt.xlabel("K: hyperparameter")
plt.ylabel("ROC")
plt.title("ROC PLOTS")
plt.legend()
plt.legend()
```

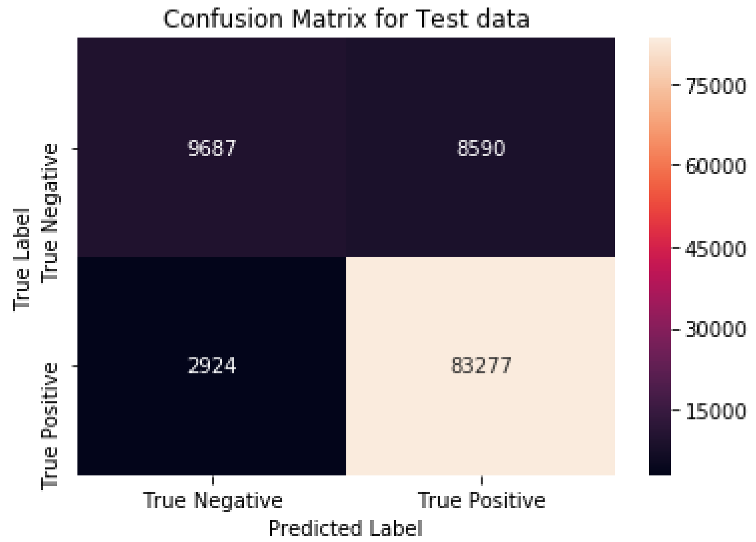
Out[395]: <matplotlib.legend.Legend at 0x25302f8fa90>



In [396]:
```
class_label = ["True Negative", "True Positive"]
cm=pd.DataFrame(confusion_matrix(Y_test,lr.predict(vectors_test)),index=class_label,columns=class_label)
```

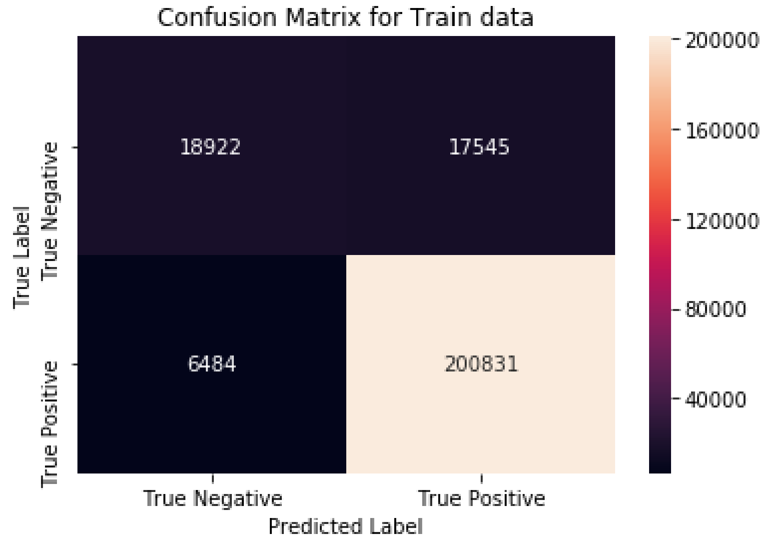
```
sns.heatmap(cm,annot=True,fmt="g")
plt.title("Confusion Matrix for Test data")
plt.xlabel("Predicted Label")
plt.ylabel("True Label")
```

Out[396]: Text(33.0, 0.5, 'True Label')



In [397]:
```
class_label = ["True Negative", "True Positive"]
cm=pd.DataFrame(confusion_matrix(Y_train,lr.predict(vectors_train)),ind
ex=class_label,columns=class_label)
sns.heatmap(cm,annot=True,fmt="g")
plt.title("Confusion Matrix for Train data")
plt.xlabel("Predicted Label")
plt.ylabel("True Label")
```

Out[397]: Text(33.0, 0.5, 'True Label')



In [398]:
```
print(classification_report(Y_test,lr.predict(vectors_test)))
```

```
              precision    recall  f1-score   support

           0       0.77      0.53      0.63     18277
           1       0.91      0.97      0.94     86201

   micro avg       0.89      0.89      0.89    104478
   macro avg       0.84      0.75      0.78    104478
weighted avg       0.88      0.89      0.88    104478
```

[5.4] Logistic Regression on TFIDF W2V, SET 4

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

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

In [378]:
```
model = TfidfVectorizer()
tf_idf_matrix = model.fit_transform(X_train)
# we are converting a dictionary with word as a key, and the idf as a v
alue
dictionary = dict(zip(model.get_feature_names(), list(model.idf_)))
```

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

tfidf_sent_vectors = []; # the tfidf-w2v for each sentence/review is st
ored in this list
row=0;
for sent in tqdm(sentance_train): # 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/r
eview
    for word in sent: # for each word in a review/sentence
        if word in w2v_words and word in tfidf_feat:
            vec = w2v_model.wv[word]
#             tf_idf = tf_idf_matrix[row, tfidf_feat.index(word)]
            # to reduce the computation we are
            # dictionary[word] = idf value of word in whole courpus
            # sent.count(word) = tf valeus of word in this review
            tf_idf = dictionary[word]*(sent.count(word)/len(sent))
            sent_vec += (vec * tf_idf)
            weight_sum += tf_idf
    if weight_sum != 0:
        sent_vec /= weight_sum
    tfidf_sent_vectors.append(sent_vec)
    row += 1
```

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

```
tfidf_sent_vectors_test = []; # the tfidf-w2v for each sentence/review
 is stored in this list
row=0;
for sent in tqdm(sentance_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/r
eview
    for word in sent: # for each word in a review/sentence
        if word in w2v_words and word in tfidf_feat:
            vec = w2v_model.wv[word]
#              tf_idf = tf_idf_matrix[row, tfidf_feat.index(word)]
            # to reduce the computation we are
            # dictionary[word] = idf value of word in whole courpus
            # sent.count(word) = tf valeus of word in this review
            tf_idf = dictionary[word]*(sent.count(word)/len(sent))
            sent_vec += (vec * tf_idf)
            weight_sum += tf_idf
    if weight_sum != 0:
        sent_vec /= weight_sum
    tfidf_sent_vectors_test.append(sent_vec)
    row += 1
```

In [381]:
```
lr_avg = LogisticRegression(penalty='l1')
#we will create a dictonary of values using which we will test for n_ne
ighbors
para_grid= {'C':[10000,7000,5000,2000,500,100,50,10,5,1,0.1,0.5,0.01,0.
05,0.005,0.001,0.0005,0.0001,0.00005,0.00001]}
#We will use the gridsearch to test all aforementioned values for alpha
# https://scikit-learn.org/stable/modules/generated/sklearn.model_selec
tion.TimeSeriesSplit.html
ts_cv = TimeSeriesSplit(n_splits =10)
# https://scikit-learn.org/stable/modules/generated/sklearn.model_selec
tion.GridSearchCV.html
lr_gsv_w2v= GridSearchCV(lr_avg,para_grid,cv=ts_cv,scoring='roc_auc',n_
jobs=-1)
#we will be fitting the optimised k values to the train_bow data
lr_gsv_w2v.fit(tfidf_sent_vectors,Y_train)
```

Out[381]:
```
GridSearchCV(cv=TimeSeriesSplit(max_train_size=None, n_splits=10),
```

```
             error_score='raise-deprecating',
             estimator=LogisticRegression(C=1.0, class_weight=None, dual=Fals
e, fit_intercept=True,
                intercept_scaling=1, max_iter=100, multi_class='warn',
                n_jobs=None, penalty='l1', random_state=None, solver='warn',
                tol=0.0001, verbose=0, warm_start=False),
             fit_params=None, iid='warn', n_jobs=-1,
             param_grid={'C': [10000, 7000, 5000, 2000, 500, 100, 50, 10, 5,
1, 0.1, 0.5, 0.01, 0.05, 0.005, 0.001, 0.0005, 0.0001, 5e-05, 1e-05]},
             pre_dispatch='2*n_jobs', refit=True, return_train_score='warn',
             scoring='roc_auc', verbose=0)
```

In [382]:
```
#The best C value obtained from the gridsearchCV is
best_C= lr_gsv_w2v.best_params_
print("Best  Hyperparameter:", best_C)
print("The accuracy obtained by using the hyperparameter is:", lr_gsv_w
2v.best_score_*100)
```

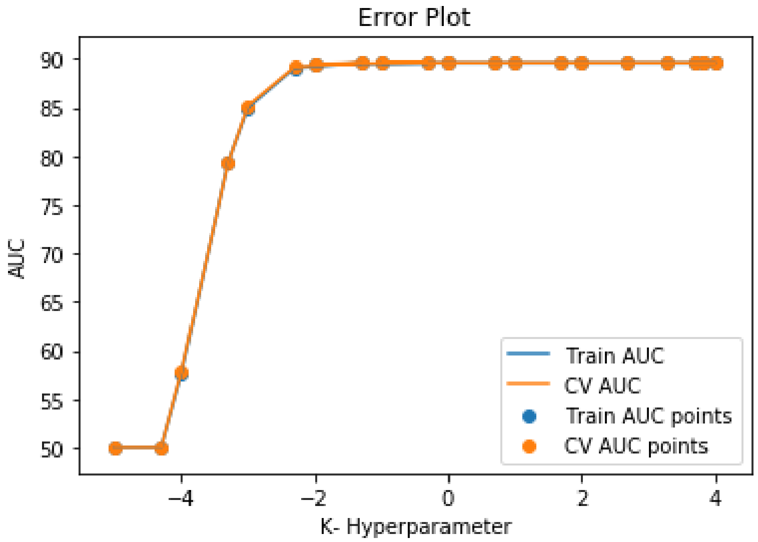
```
Best  Hyperparameter: {'C': 0.5}
The accuracy obtained by using the hyperparameter is: 89.61585985248975
```

In [383]:
```
train_auc = lr_gsv_w2v.cv_results_['mean_train_score']
cv_auc = lr_gsv_w2v.cv_results_['mean_test_score']
for key in para_grid.values():
    key= np.log10(key) # used log scale on X axis for ease of understan
ding
    print(key)
plt.plot(key,train_auc*100,label='Train AUC')
plt.plot(key,cv_auc*100,label='CV AUC')
plt.scatter(key,train_auc*100,label='Train AUC points')
plt.scatter(key,cv_auc*100,label='CV AUC points')
plt.legend()
plt.xlabel('K- Hyperparameter')
plt.ylabel('AUC')
plt.title('Error Plot')
```

```
[ 4.           3.84509804  3.69897      3.30103      2.69897      2.
  1.69897      1.           0.69897      0.          -1.          -0.30103
```

```
 -2.          -1.30103    -2.30103    -3.          -3.30103    -4.
 -4.30103    -5.         ]
```

Out[383]: Text(0.5, 1.0, 'Error Plot')



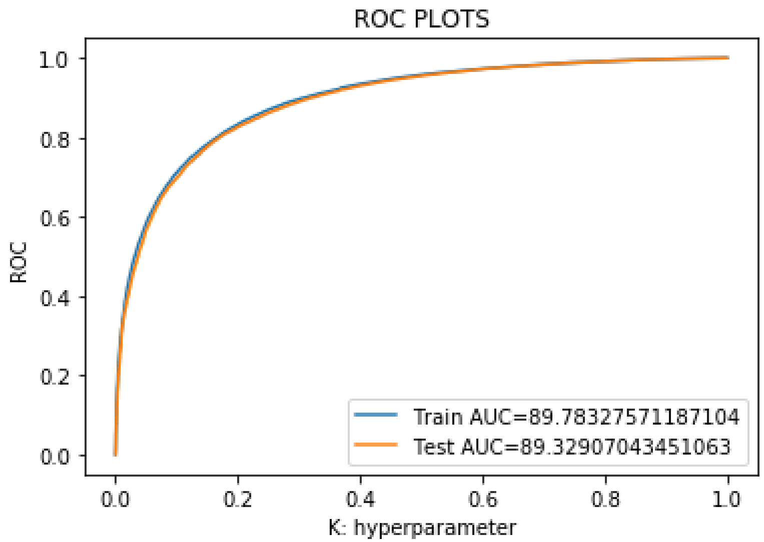
In [384]:
```
#we will select the best hyperparameter and train the model using the p
arameter

lr = LogisticRegression(penalty = 'l1', C= 0.5, n_jobs=-1)
lr.fit(tfidf_sent_vectors, Y_train)
y_train_pred= lr.predict_proba(tfidf_sent_vectors)[:,1]
y_test_pred=lr.predict_proba(tfidf_sent_vectors_test)[:,1]
```

In [385]:
```
train_fpr,train_tpr,thersholds=roc_curve(Y_train,y_train_pred)
test_fpr,test_tpr,thersholds=roc_curve(Y_test,y_test_pred)
```

In [386]:
```
plt.plot(train_fpr,train_tpr,label="Train AUC=" +str(auc(train_fpr,trai
n_tpr)*100))
plt.plot(test_fpr,test_tpr,label="Test AUC=" +str(auc(test_fpr,test_tpr
)*100))
plt.xlabel("K: hyperparameter")
```

```
plt.ylabel("ROC")
plt.title("ROC PLOTS")
plt.legend()
plt.legend()
```

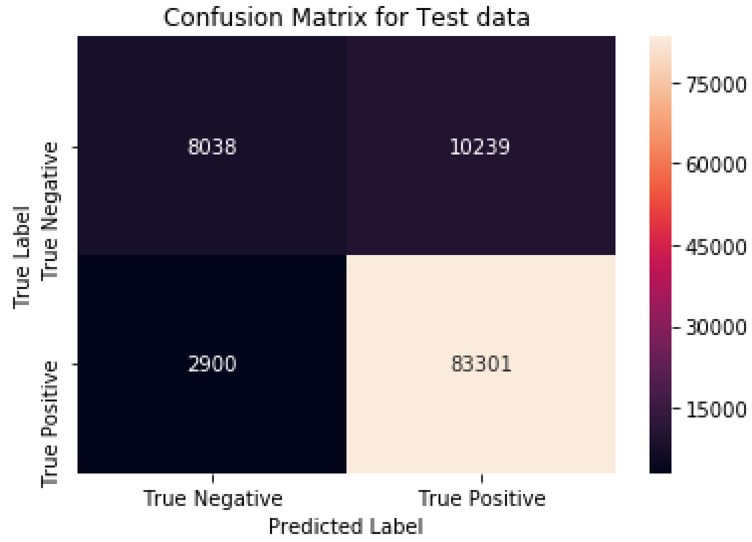
Out[386]: <matplotlib.legend.Legend at 0x253031c9be0>



In [387]:

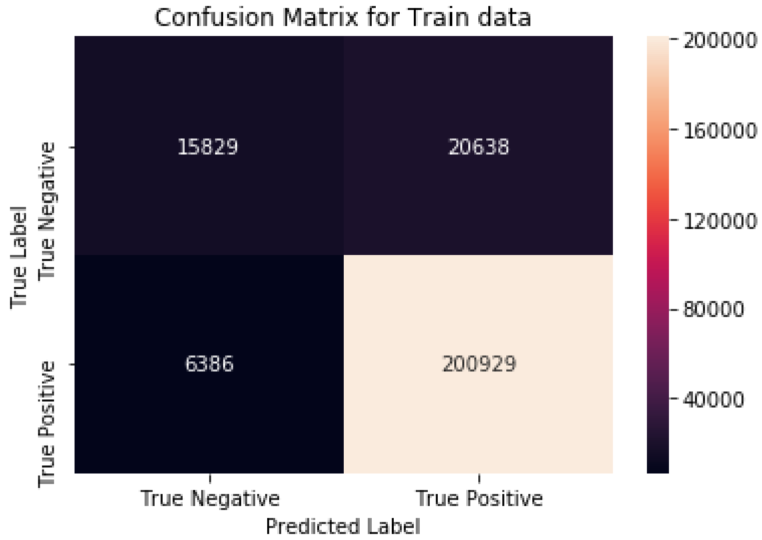
```
class_label = ["True Negative", "True Positive"]
cm=pd.DataFrame(confusion_matrix(Y_test,lr.predict(tfidf_sent_vectors_t
est)),index=class_label,columns=class_label)
sns.heatmap(cm,annot=True,fmt="g")
plt.title("Confusion Matrix for Test data")
plt.xlabel("Predicted Label")
plt.ylabel("True Label")
```

Out[387]: Text(33.0, 0.5, 'True Label')



In [388]:
```
class_label = ["True Negative", "True Positive"]
cm=pd.DataFrame(confusion_matrix(Y_train,lr.predict(tfidf_sent_vectors
)),index=class_label,columns=class_label)
sns.heatmap(cm,annot=True,fmt="g")
plt.title("Confusion Matrix for Train data")
plt.xlabel("Predicted Label")
plt.ylabel("True Label")
```

Out[388]:
```
Text(33.0, 0.5, 'True Label')
```



In [389]:
```
print(classification_report(Y_test,lr.predict(tfidf_sent_vectors_test
)))
```

```
              precision    recall  f1-score   support

           0       0.73      0.44      0.55     18277
           1       0.89      0.97      0.93     86201

   micro avg       0.87      0.87      0.87    104478
   macro avg       0.81      0.70      0.74    104478
weighted avg       0.86      0.87      0.86    104478
```

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

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

In [399]:
```
lr_avg = LogisticRegression(penalty='l2')
#we will create a dictonary of values using which we will test for n_ne
```

```
ighbors
para_grid= {'C':[10000,7000,5000,2000,500,100,50,10,5,1,0.1,0.5,0.01,0.
05,0.005,0.001,0.0005,0.0001,0.00005,0.00001]}
#We will use the gridsearch to test all aforementioned values for alpha
# https://scikit-learn.org/stable/modules/generated/sklearn.model_selec
tion.TimeSeriesSplit.html
ts_cv = TimeSeriesSplit(n_splits =10)
# https://scikit-learn.org/stable/modules/generated/sklearn.model_selec
tion.GridSearchCV.html
lr_gsv_w2v= GridSearchCV(lr_avg,para_grid,cv=ts_cv,scoring='roc_auc',n_
jobs=-1)
#we will be fitting the optimised k values to the train_bow data
lr_gsv_w2v.fit(tfidf_sent_vectors,Y_train)
```

Out[399]:

```
GridSearchCV(cv=TimeSeriesSplit(max_train_size=None, n_splits=10),
       error_score='raise-deprecating',
       estimator=LogisticRegression(C=1.0, class_weight=None, dual=Fals
e, fit_intercept=True,
          intercept_scaling=1, max_iter=100, multi_class='warn',
          n_jobs=None, penalty='l2', random_state=None, solver='warn',
          tol=0.0001, verbose=0, warm_start=False),
       fit_params=None, iid='warn', n_jobs=-1,
       param_grid={'C': [10000, 7000, 5000, 2000, 500, 100, 50, 10, 5,
1, 0.1, 0.5, 0.01, 0.05, 0.005, 0.001, 0.0005, 0.0001, 5e-05, 1e-05]},
       pre_dispatch='2*n_jobs', refit=True, return_train_score='warn',
       scoring='roc_auc', verbose=0)
```

In [400]:

```
#The best C value obtained from the gridsearchCV is
best_C= lr_gsv_w2v.best_params_
print("Best  Hyperparameter:", best_C)
print("The accuracy obtained by using the hyperparameter is:", lr_gsv_w
2v.best_score_*100)
```

```
Best  Hyperparameter: {'C': 0.5}
The accuracy obtained by using the hyperparameter is: 89.6131741334993
```

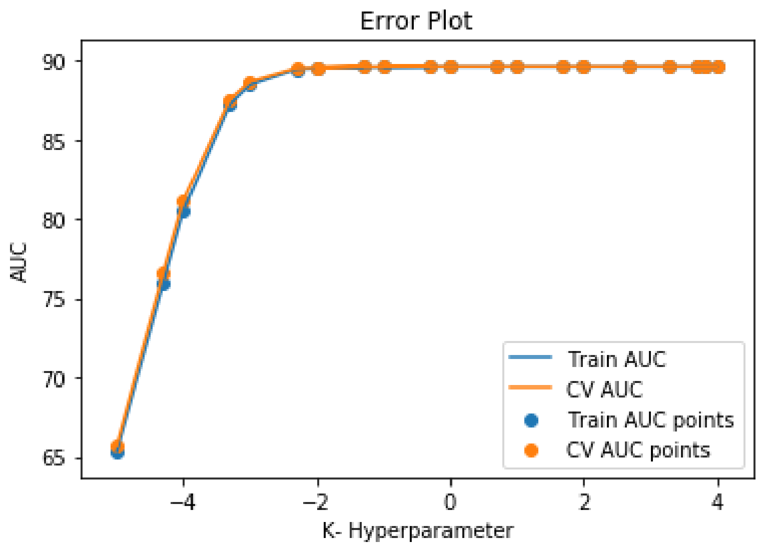
In [401]:

```
train_auc = lr_gsv_w2v.cv_results_['mean_train_score']
cv_auc = lr_gsv_w2v.cv_results_['mean_test_score']
for key in para_grid.values():
```

```
    key= np.log10(key) # used log scale on X axis for ease of understan
ding
    print(key)
plt.plot(key,train_auc*100,label='Train AUC')
plt.plot(key,cv_auc*100,label='CV AUC')
plt.scatter(key,train_auc*100,label='Train AUC points')
plt.scatter(key,cv_auc*100,label='CV AUC points')
plt.legend()
plt.xlabel('K- Hyperparameter')
plt.ylabel('AUC')
plt.title('Error Plot')
```

```
[ 4.          3.84509804  3.69897     3.30103     2.69897     2.
  1.69897     1.          0.69897     0.         -1.         -0.30103
 -2.         -1.30103    -2.30103    -3.         -3.30103    -4.
 -4.30103    -5.        ]
```

Out[401]: Text(0.5, 1.0, 'Error Plot')



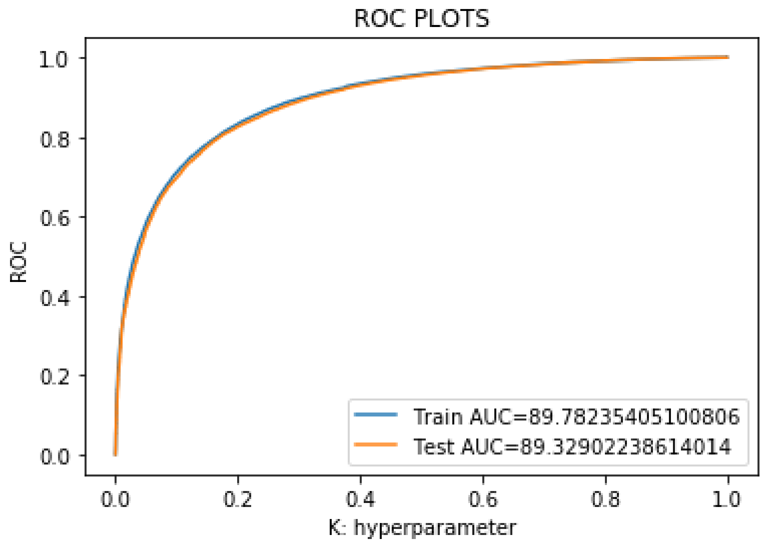
In [402]:
```
#we will select the best hyperparameter and train the model using the p
arameter

lr = LogisticRegression(penalty = 'l2', C= 0.5, n_jobs=-1)
```

```
lr.fit(tfidf_sent_vectors, Y_train)
y_train_pred= lr.predict_proba(tfidf_sent_vectors)[:,1]
y_test_pred=lr.predict_proba(tfidf_sent_vectors_test)[:,1]
```

In [403]:
```
train_fpr,train_tpr,thersholds=roc_curve(Y_train,y_train_pred)
test_fpr,test_tpr,thersholds=roc_curve(Y_test,y_test_pred)
```

In [404]:
```
plt.plot(train_fpr,train_tpr,label="Train AUC=" +str(auc(train_fpr,trai
n_tpr)*100))
plt.plot(test_fpr,test_tpr,label="Test AUC=" +str(auc(test_fpr,test_tpr
)*100))
plt.xlabel("K: hyperparameter")
plt.ylabel("ROC")
plt.title("ROC PLOTS")
plt.legend()
plt.legend()
```

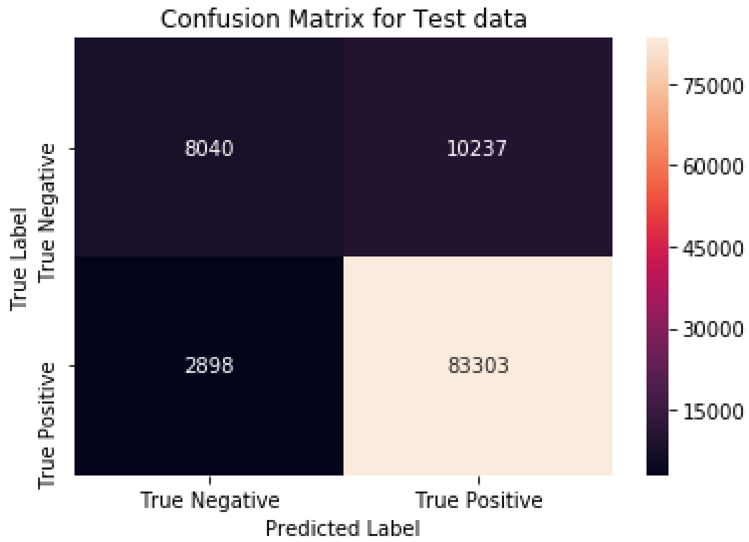
Out[404]: <matplotlib.legend.Legend at 0x2530323c9e8>



In [405]:
```
class_label = ["True Negative", "True Positive"]
cm=pd.DataFrame(confusion_matrix(Y_test,lr.predict(tfidf_sent_vectors_t
```

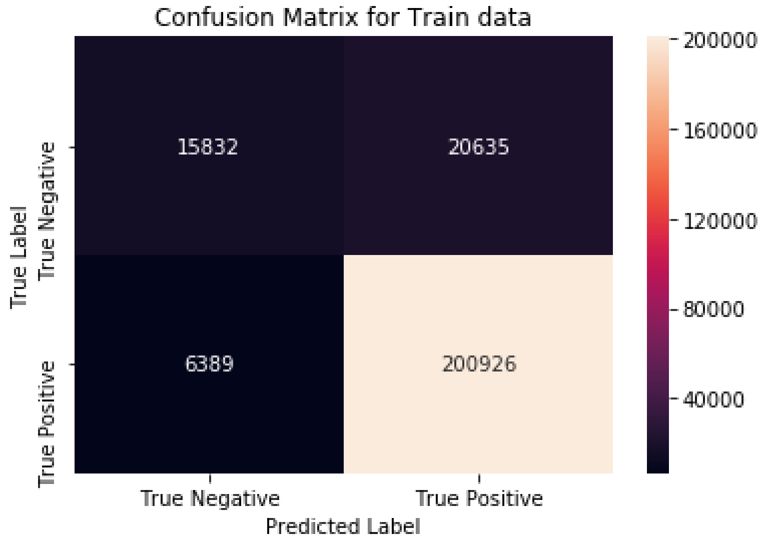
```
est)),index=class_label,columns=class_label)
sns.heatmap(cm,annot=True,fmt="g")
plt.title("Confusion Matrix for Test data")
plt.xlabel("Predicted Label")
plt.ylabel("True Label")
```

Out[405]: Text(33.0, 0.5, 'True Label')



In [406]:
```
class_label = ["True Negative", "True Positive"]
cm=pd.DataFrame(confusion_matrix(Y_train,lr.predict(tfidf_sent_vectors
)),index=class_label,columns=class_label)
sns.heatmap(cm,annot=True,fmt="g")
plt.title("Confusion Matrix for Train data")
plt.xlabel("Predicted Label")
plt.ylabel("True Label")
```

Out[406]: Text(33.0, 0.5, 'True Label')



In [407]:
```
print(classification_report(Y_test,lr.predict(tfidf_sent_vectors_test
)))
```

```
              precision    recall  f1-score   support

           0       0.74      0.44      0.55     18277
           1       0.89      0.97      0.93     86201

   micro avg       0.87      0.87      0.87    104478
   macro avg       0.81      0.70      0.74    104478
weighted avg       0.86      0.87      0.86    104478
```

[6] Conclusions

In [ ]:
```
# Please compare all your models using Prettytable library
```

In [408]:
```
from prettytable import PrettyTable
```

```
tab = PrettyTable()
```

In [409]:

```
tab.field_names = ["Vectorizer", "Regularizer", "Hyperparameter", "AUC"
]

tab.add_row(["BOW", "L1", 5, 94.93])
tab.add_row(["TFIDF", "L1", 5, 96.21])
tab.add_row(["W2V", "L1", 0.5, 91.92])
tab.add_row(["TFIDFW2V", "L1", 0.5, 89.61])
tab.add_row(["BOW", "L2", 5, 95.16])
tab.add_row(["TFIDF", "L2", 100, 96.54])
tab.add_row(["W2V", "L2", 0.05, 91.93])
tab.add_row(["TFIDFW2V", "L2", 0.5, 89.62])
```

In [410]:

```
print(tab)
```

```
+------------+-------------+----------------+-------+
| Vectorizer | Regularizer | Hyperparameter |  AUC  |
+------------+-------------+----------------+-------+
|    BOW     |      L1     |       5        | 94.93 |
|   TFIDF    |      L1     |       5        | 96.21 |
|    W2V     |      L1     |      0.5       | 91.92 |
|  TFIDFW2V  |      L1     |      0.5       | 89.61 |
|    BOW     |      L2     |       5        | 95.16 |
|   TFIDF    |      L2     |      100       | 96.54 |
|    W2V     |      L2     |      0.05      | 91.93 |
|  TFIDFW2V  |      L2     |      0.5       | 89.62 |
+------------+-------------+----------------+-------+
```

Conclusion

1. After text preprocessing, we split the data into test and train dataset.
2. We used BOW, TFIDF, AVG W2V, TFIDF W2V vectorizer on the review text.
3. Used logistic regression as an estimator in GridsearchCV to find the optimal hyperparameterby using both L1 and L2 regularization.

4. After obtaining the optimal C, we trained the model using the hyperparameter and we obtained the precision and recall on predicted data.
5. Performed perturbation test on BOW vectorised review column. Found there was a drastic change in weight vector when the percentile value changed from 98 to 99.4%.
6. Printed all the 51 highly correlated vectors which causes multicollinearity
7. For all the bulit models, we found the F1 score to be good except for TFIDF W2V vectoriser where the F1 score was around 0.63 for negative class.