# Amazon Fine Food Reviews Analysis

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

EDA: https://nycdatascience.com/blog/student-works/amazon-fine-foods-visualization/ (https://nycdatascience.com/blog/student-works/amazon-fine-foods-visualization/)

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

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

Attribute Information:

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

**Objective:**

Given a review, determine whether the review is positive (rating of 4 or 5) or negative (rating of 1 or 2).

[Q] How to determine if a review is positive or negative?

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

# [1]. Reading Data

# [1.1] Loading the data

The dataset is available in two forms

1. .csv file
2. SQLite Database

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

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

In [1]:

```
%matplotlib inline
import warnings
warnings.filterwarnings("ignore")


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

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

import re
# Tutorial about Python regular expressions: https://pymotw.com/2/re/
import string
from nltk.corpus import stopwords
from nltk.stem import PorterStemmer
from nltk.stem.wordnet import WordNetLemmatizer

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

from tqdm import tqdm
import os
```

In [2]:

```
# using SQLite Table to read data.
con = sqlite3.connect('database.sqlite')

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

# filtered_data = pd.read_sql_query(""" SELECT * FROM Reviews WHERE Score != 3 LIMIT 50
0000""", con)
# for tsne assignment you can take 5k data points

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

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

#changing reviews with score less than 3 to be positive and vice-versa
actualScore = filtered_data['Score']
positiveNegative = actualScore.map(partition)
filtered_data['Score'] = positiveNegative
print("Number of data points in our data", filtered_data.shape)
filtered_data.head(3)
```

```
Number of data points in our data (525814, 10)
```

Out[2]:

| | Id | ProductId | UserId | ProfileName | HelpfulnessNumerator | Helpfu |
|---|---|---|---|---|---|---|
| **0** | 1 | B001E4KFG0 | A3SGXH7AUHU8GW | delmartian | 1 | 1 |
| **1** | 2 | B00813GRG4 | A1D87F6ZCVE5NK | dll pa | 0 | 0 |
| **2** | 3 | B000LQOCH0 | ABXLMWJIXXAIN | Natalia Corres "Natalia Corres" | 1 | 1 |

In [3]:

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

In [4]:

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

```
(80668, 7)
```

Out[4]:

| | UserId | ProductId | ProfileName | Time | Score | Text | C |
|---|---|---|---|---|---|---|---|
| **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 |
| **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 [5]:

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

Out[5]:

| | UserId | ProductId | ProfileName | Time | Score | |
|---|---|---|---|---|---|---|
| **80638** | AZY10LLTJ71NX | B006P7E5ZI | undertheshrine "undertheshrine" | 1334707200 | 5 | I was recomme to try gree tea extrac ... |

In [6]:

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

Out[6]:

393063

# [2] Exploratory Data Analysis

## [2.1] Data Cleaning: Deduplication

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

In [7]:

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

Out[7]:

| | Id | ProductId | UserId | ProfileName | HelpfulnessNumerator | Help |
|---|---|---|---|---|---|---|
| **0** | 78445 | B000HDL1RQ | AR5J8UI46CURR | Geetha Krishnan | 2 | 2 |
| **1** | 138317 | B000HDOPYC | AR5J8UI46CURR | Geetha Krishnan | 2 | 2 |
| **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 [8]:

```
#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 [9]:

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

Out[9]:

```
(364173, 10)
```

In [10]:

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

Out[10]:

```
69.25890143662969
```

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

In [11]:

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

display.head()
```

Out[11]:

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

In [12]:

```
final=final[final.HelpfulnessNumerator<=final.HelpfulnessDenominator]
```

In [13]:

```
#Before starting the next phase of preprocessing lets see the number of entries left
print(final.shape)

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

```
(364171, 10)
```

Out[13]:

```
1    307061
0     57110
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 [15]:

```
# 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 witty little book makes my son laugh at loud. i recite it in the car
as we're driving along and he always can sing the refrain. he's learned ab
out whales, India, drooping roses:  i love all the new words this book  in
troduces and the silliness of it all.  this is a classic book i am  willin
g to bet my son will STILL be able to recite from memory when he is  in co
llege
==================================================
I was really looking forward to these pods based on the reviews.  Starbuck
s is good, but I prefer bolder taste.... imagine my surprise when I ordere
d 2 boxes - both were expired! One expired back in 2005 for gosh sakes.  I
admit that Amazon agreed to credit me for cost plus part of shipping, but
geez, 2 years expired!!!  I'm hoping to find local San Diego area shoppe t
hat carries pods so that I can try something different than starbucks.
==================================================
Great ingredients although, chicken should have been 1st rather than chick
en broth, the only thing I do not think belongs in it is Canola oil. Canol
a or rapeseed is not someting a dog would ever find in nature and if it di
d find rapeseed in nature and eat it, it would poison them. Today's Food i
ndustries have convinced the masses that Canola oil is a safe and even bet
ter oil than olive or virgin coconut, facts though say otherwise. Until th
e late 70's it was poisonous until they figured out a way to fix that. I s
till like it but it could be better.
==================================================
Can't do sugar.  Have tried scores of SF Syrups.  NONE of them can touch t
he excellence of this product.<br /><br />Thick, delicious.  Perfect.  3 i
ngredients: Water, Maltitol, Natural Maple Flavor.  PERIOD.  No chemicals.
No garbage.<br /><br />Have numerous friends & family members hooked on th
is stuff.  My husband & son, who do NOT like "sugar free" prefer this over
major label regular syrup.<br /><br />I use this as my SWEETENER in bakin
g: cheesecakes, white brownies, muffins, pumpkin pies, etc... Unbelievably
delicious...<br /><br />Can you tell I like it? :)
==================================================
```

In [16]:

```
# remove urls from text python: https://stackoverflow.com/a/40823105/4084039
sent_0 = re.sub(r"http\S+", "", sent_0)
sent_1000 = re.sub(r"http\S+", "", sent_1000)
sent_150 = re.sub(r"http\S+", "", sent_1500)
sent_4900 = re.sub(r"http\S+", "", sent_4900)

print(sent_0)
```

```
this witty little book makes my son laugh at loud. i recite it in the car
as we're driving along and he always can sing the refrain. he's learned ab
out whales, India, drooping roses:  i love all the new words this book  in
troduces and the silliness of it all.  this is a classic book i am  willin
g to bet my son will STILL be able to recite from memory when he is  in co
llege
```

In [17]:

```
# https://stackoverflow.com/questions/16206380/python-beautifulsoup-how-to-remove-all-t
ags-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 witty little book makes my son laugh at loud. i recite it in the car
as we're driving along and he always can sing the refrain. he's learned ab
out whales, India, drooping roses:  i love all the new words this book  in
troduces and the silliness of it all.  this is a classic book i am  willin
g to bet my son will STILL be able to recite from memory when he is  in co
llege
==================================================
I was really looking forward to these pods based on the reviews.  Starbuck
s is good, but I prefer bolder taste.... imagine my surprise when I ordere
d 2 boxes - both were expired! One expired back in 2005 for gosh sakes.  I
admit that Amazon agreed to credit me for cost plus part of shipping, but
geez, 2 years expired!!!  I'm hoping to find local San Diego area shoppe t
hat carries pods so that I can try something different than starbucks.
==================================================
Great ingredients although, chicken should have been 1st rather than chick
en broth, the only thing I do not think belongs in it is Canola oil. Canol
a or rapeseed is not someting a dog would ever find in nature and if it di
d find rapeseed in nature and eat it, it would poison them. Today's Food i
ndustries have convinced the masses that Canola oil is a safe and even bet
ter oil than olive or virgin coconut, facts though say otherwise. Until th
e late 70's it was poisonous until they figured out a way to fix that. I s
till like it but it could be better.
==================================================
Can't do sugar.  Have tried scores of SF Syrups.  NONE of them can touch t
he excellence of this product.Thick, delicious.  Perfect.  3 ingredients:
Water, Maltitol, Natural Maple Flavor.  PERIOD.  No chemicals.  No garbag
e.Have numerous friends & family members hooked on this stuff.  My husband
& son, who do NOT like "sugar free" prefer this over major label regular s
yrup.I use this as my SWEETENER in baking: cheesecakes, white brownies, mu
ffins, pumpkin pies, etc... Unbelievably delicious...Can you tell I like i
t? :)
```

In [18]:

```
# 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 [19]:

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

```
Great ingredients although, chicken should have been 1st rather than chick
en broth, the only thing I do not think belongs in it is Canola oil. Canol
a or rapeseed is not someting a dog would ever find in nature and if it di
d find rapeseed in nature and eat it, it would poison them. Today is Food
industries have convinced the masses that Canola oil is a safe and even be
tter oil than olive or virgin coconut, facts though say otherwise. Until t
he late 70 is it was poisonous until they figured out a way to fix that. I
still like it but it could be better.
==================================================
```

In [20]:

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

```
this witty little book makes my son laugh at loud. i recite it in the car
as we're driving along and he always can sing the refrain. he's learned ab
out whales, India, drooping roses:  i love all the new words this book  in
troduces and the silliness of it all.  this is a classic book i am  willin
g to bet my son will STILL be able to recite from memory when he is  in co
llege
```

In [21]:

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

```
Great ingredients although chicken should have been 1st rather than chicke
n broth the only thing I do not think belongs in it is Canola oil Canola o
r rapeseed is not someting a dog would ever find in nature and if it did f
ind rapeseed in nature and eat it it would poison them Today is Food indus
tries have convinced the masses that Canola oil is a safe and even better
oil than olive or virgin coconut facts though say otherwise Until the late
70 is it was poisonous until they figured out a way to fix that I still li
ke it but it could be better
```

In [22]:

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

stopwords= set(['br', 'the', 'i', 'me', 'my', 'myself', 'we', 'our', 'ours', 'ourselve
s', 'you', "you're", "you've",\
            "you'll", "you'd", 'your', 'yours', 'yourself', 'yourselves', 'he', 'him',
'his', 'himself', \
            'she', "she's", 'her', 'hers', 'herself', 'it', "it's", 'its', 'itself', 't
hey', 'them', 'their',\
            'theirs', 'themselves', 'what', 'which', 'who', 'whom', 'this', 'that', "th
at'll", 'these', 'those', \
            'am', 'is', 'are', 'was', 'were', 'be', 'been', 'being', 'have', 'has', 'ha
d', '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', 'ov
er', 'under', 'again', 'further',\
            'then', 'once', 'here', 'there', 'when', 'where', 'why', 'how', 'all', 'an
y', 'both', 'each', 'few', 'more',\
            'most', 'other', 'some', 'such', 'only', 'own', 'same', 'so', 'than', 'too'
, 'very', \
            's', 't', 'can', 'will', 'just', 'don', "don't", 'should', "should've", 'no
w', 'd', 'll', 'm', 'o', 're', \
            've', 'y', 'ain', 'aren', "aren't", 'couldn', "couldn't", 'didn', "didn't",
'doesn', "doesn't", 'hadn',\
            "hadn't", 'hasn', "hasn't", 'haven', "haven't", 'isn', "isn't", 'ma', 'migh
tn', "mightn't", 'mustn',\
            "mustn't", 'needn', "needn't", 'shan', "shan't", 'shouldn', "shouldn't", 'w
asn', "wasn't", 'weren', "weren't", \
            'won', "won't", 'wouldn', "wouldn't"])
```

In [23]:

```
# 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 stopwor
ds)
    preprocessed_reviews.append(sentance.strip())
```

```
100%|██████████████████████████████| 364171/364171 [03:02<00:00, 1992.68
it/s]
```

In [24]:

```
preprocessed_reviews[1500]
```

Out[24]:

```
'great ingredients although chicken rather chicken broth thing not think b
elongs canola oil canola rapeseed not someting dog would ever find nature
find rapeseed nature eat would poison today food industries convinced mass
es canola oil safe even better oil olive virgin coconut facts though say o
therwise late poisonous figured way fix still like could better'
```

## [3.2] Preprocessing Review Summary

In [0]:

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

# [4] Featurization

# [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 (https://www.appliedaicourse.com/course/applied-ai-course-online/lessons/receiver-operating-characteristic-curve-roc-curve-and-auc-1/) 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 (https://www.appliedaicourse.com/course/applied-ai-course-online/lessons/confusion-matrix-tpr-fpr-fnr-tnr-1/) with predicted and original labels of test data points. Please visualize your confusion matrices using seaborn heatmaps.
  (https://seaborn.pydata.org/generated/seaborn.heatmap.html)
  (https://seaborn.pydata.org/generated/seaborn.heatmap.html)
(https://seaborn.pydata.org/generated/seaborn.heatmap.html)

(https://seaborn.pydata.org/generated/seaborn.heatmap.html)

8. **Conclusion** (https://seaborn.pydata.org/generated/seaborn.heatmap.html)

   (https://seaborn.pydata.org/generated/seaborn.heatmap.html)
   - 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 (https://seaborn.pydata.org/generated/seaborn.heatmap.html) link (http://zetcode.com/python/prettytable/)

**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. (https://soundcloud.com/applied-ai-course/leakage-bow-and-tfidf)

# Applying Logistic Regression

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

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

In [14]:

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

# =========== Loading Libraries ==================

import warnings
warnings.filterwarnings("ignore")
from sklearn.cross_validation import train_test_split
from sklearn.metrics import accuracy_score
from sklearn.cross_validation import cross_val_score
from collections import Counter
from sklearn import cross_validation
```

In [380]:

```
df = pd.DataFrame({'Text':preprocessed_reviews})
X = df['Text'][:100000].values
y = final['Score'][:100000].values
```

In [381]:

```
# split the data set into train and test
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.33) # this is ran
dom splitting
X_train, X_cv, y_train, y_cv = train_test_split(X_train, y_train, test_size=0.33) # thi
s is random splitting
```

In [65]:

```
print(X_train.shape, y_train.shape)
print(X_cv.shape, y_cv.shape)
print(X_test.shape, y_test.shape)
```

```
(44890,) (44890,)
(22110,) (22110,)
(33000,) (33000,)
```

In [66]:

```
from sklearn.feature_extraction.text import CountVectorizer
vectorizer = CountVectorizer()
vectorizer.fit(X_train) # fit has to happen only on train data

# we use the fitted CountVectorizer to convert the text to vector
X_train_bow = vectorizer.transform(X_train)
X_cv_bow = vectorizer.transform(X_cv)
X_test_bow = vectorizer.transform(X_test)

print("After vectorizations")
print(X_train_bow.shape, y_train.shape)
print(X_cv_bow.shape, y_cv.shape)
print(X_test_bow.shape, y_test.shape)
```

```
After vectorizations
(44890, 40475) (44890,)
(22110, 40475) (22110,)
(33000, 40475) (33000,)
```

# Simple cross validation

In [70]:

```
from sklearn.linear_model import LogisticRegression
K = [10**-4, 10**-3, 10**-2, 10**-1, 1, 10**1, 10**2, 10**3, 10**4]
#for i in range(1,50,2):
for i in K:
    clf = LogisticRegression(C=i)
    clf.fit(X_train_bow, y_train)

    # predict the response on the crossvalidation train
    pred = clf.predict(X_cv_bow)

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

```
CV accuracy for lambda = 0.000100 is 84%

CV accuracy for lambda = 0.001000 is 86%

CV accuracy for lambda = 0.010000 is 90%

CV accuracy for lambda = 0.100000 is 91%

CV accuracy for lambda = 1.000000 is 91%

CV accuracy for lambda = 10.000000 is 90%

CV accuracy for lambda = 100.000000 is 89%

CV accuracy for lambda = 1000.000000 is 88%

CV accuracy for lambda = 10000.000000 is 88%
```

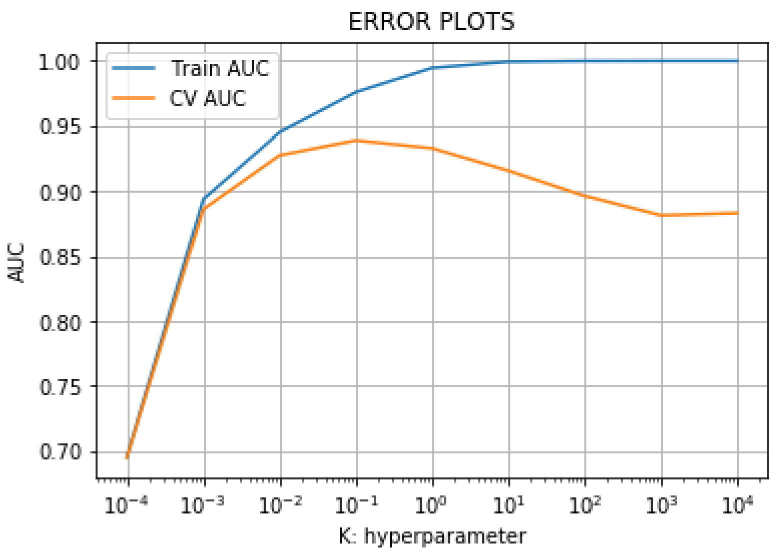
## Simple for loop

In [71]:

```
from sklearn.metrics import roc_auc_score
import matplotlib.pyplot as plt

train_auc = []
cv_auc = []
#K = [1, 5, 10, 15, 21, 31, 41, 51]
K = [10**-4, 10**-3, 10**-2, 10**-1, 1, 10**1, 10**2, 10**3, 10**4]
for i in K:
    neigh = LogisticRegression(C=i)
    neigh.fit(X_train_bow, y_train)


    #y_train_pred = []
    #for i in range(0, X_train.shape[0], 1000):
    #    y_train_pred.extend(neigh.predict_proba(X_train_bow[i:i+1000])[:,1])

    #y_cv_pred = []
    #for i in range(0, X_cv.shape[0], 1000):
    #   y_cv_pred.extend(neigh.predict_proba(X_cv_bow[i:i+1000])[:,1])

    y_train_pred =  neigh.predict_proba(X_train_bow)[:,1]
    y_cv_pred =  neigh.predict_proba(X_cv_bow)[:,1]

    train_auc.append(roc_auc_score(y_train,y_train_pred))
    cv_auc.append(roc_auc_score(y_cv, y_cv_pred))


plt.semilogx(K, train_auc, label='Train AUC')
plt.semilogx(K, cv_auc, label='CV AUC')
plt.legend()
plt.grid()
plt.xlabel("K: hyperparameter")
plt.ylabel("AUC")
plt.title("ERROR PLOTS")
```

Out[71]:

Text(0.5,1,'ERROR PLOTS')

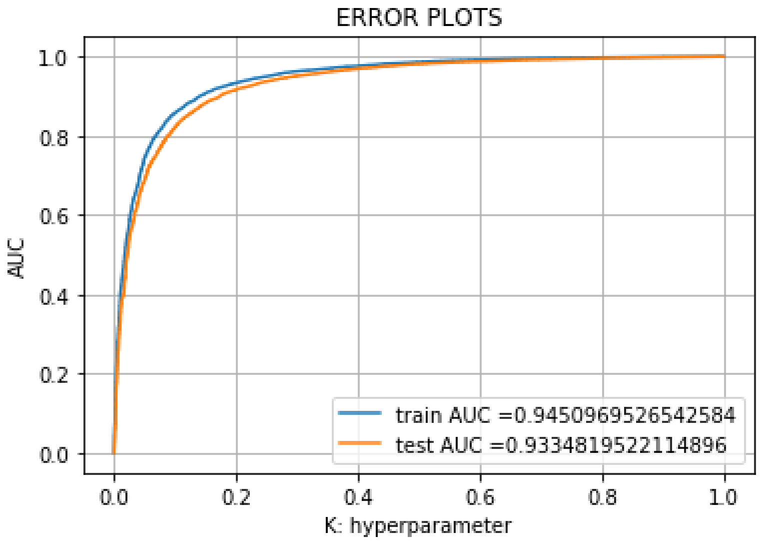



OBSERVATION: The best value of lambda is 10**-2.

## Testing with Test data

In [149]:

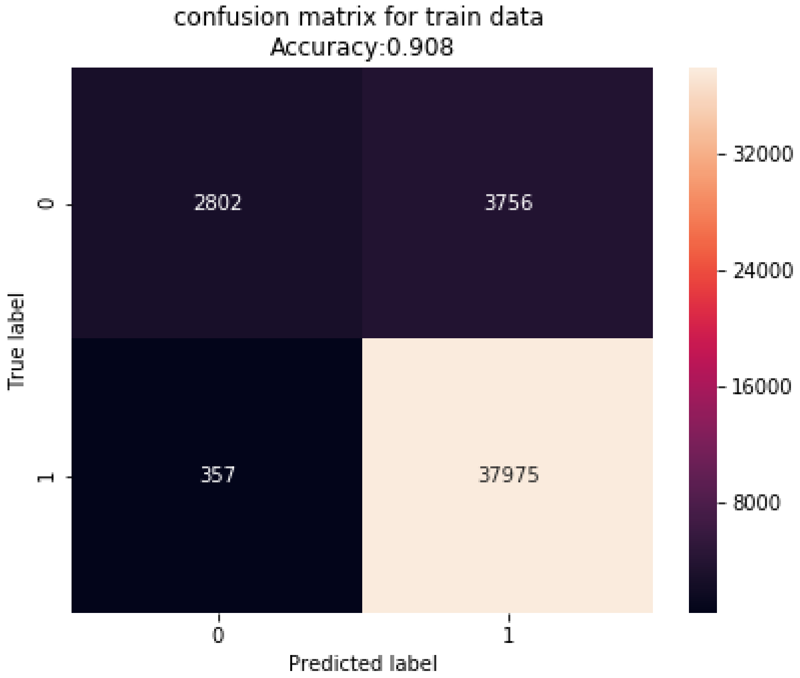
```
# https://scikit-learn.org/stable/modules/generated/sklearn.metrics.roc_curve.html#skle
arn.metrics.roc_curve
from sklearn.metrics import roc_curve, auc


neigh = LogisticRegression(C=10**-2)
neigh.fit(X_train_bow, y_train)
# roc_auc_score(y_true, y_score) the 2nd parameter should be probability estimates of t
he positive class
# not the predicted outputs

train_fpr, train_tpr, thresholds = roc_curve(y_train, neigh.predict_proba(X_train_bow)
[:,1])
test_fpr, test_tpr, thresholds = roc_curve(y_test, neigh.predict_proba(X_test_bow)[:,1
])

plt.plot(train_fpr, train_tpr, label="train AUC ="+str(auc(train_fpr, train_tpr)))
plt.plot(test_fpr, test_tpr, label="test AUC ="+str(auc(test_fpr, test_tpr)))
plt.legend()
plt.grid()
plt.xlabel("K: hyperparameter")
plt.ylabel("AUC")
plt.title("ERROR PLOTS")
plt.show()

print("="*100)

from sklearn.metrics import confusion_matrix
print("Train confusion matrix")
print(confusion_matrix(y_train, neigh.predict(X_train_bow)))
print("Test confusion matrix")
print(confusion_matrix(y_test, neigh.predict(X_test_bow)))
```

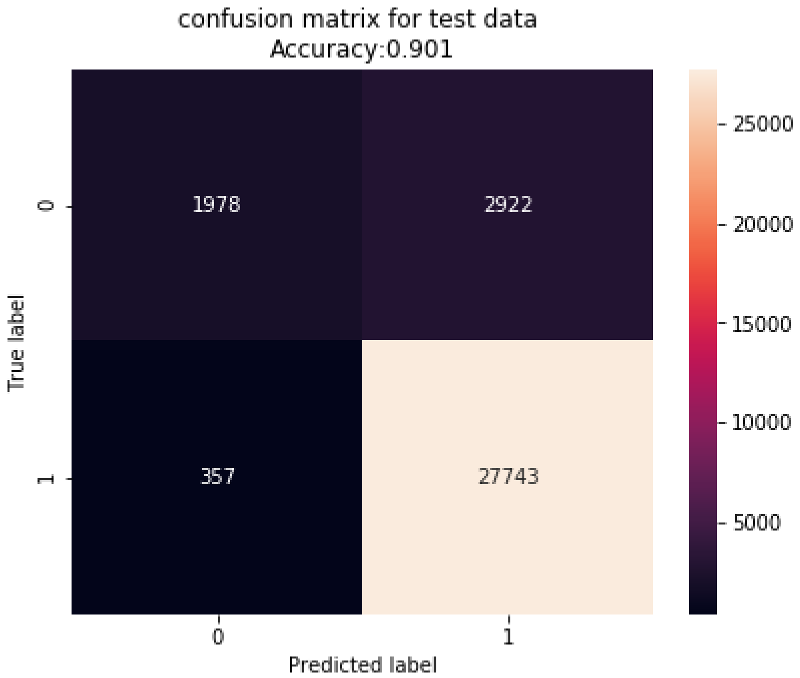



```
==========================================================================
==========================
Train confusion matrix
[[ 2802  3756]
 [  357 37975]]
Test confusion matrix
[[ 1978  2922]
 [  357 27743]]
```

In [141]:

```
# Creates a confusion matrix for train data
cm = confusion_matrix(y_train, neigh.predict(X_train_bow))
cm_df = pd.DataFrame(cm)
plt.figure(figsize=(6.5,5))
sns.heatmap(cm_df, annot=True, fmt="d")
plt.title('confusion matrix for train data \nAccuracy:{0:.3f}'.format(accuracy_score(y_
train, neigh.predict(X_train_bow))))
plt.ylabel('True label')
plt.xlabel('Predicted label')
plt.show()


# Creates a confusion matrix for test data
cm = confusion_matrix(y_test, neigh.predict(X_test_bow))
cm_df = pd.DataFrame(cm)
plt.figure(figsize=(6.5,5))
sns.heatmap(cm_df, annot=True, fmt="d")
plt.title('confusion matrix for test data \nAccuracy:{0:.3f}'.format(accuracy_score(y_t
est, neigh.predict(X_test_bow))))
plt.ylabel('True label')
plt.xlabel('Predicted label')
plt.show()
```




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

In [77]:

```
l2 = neigh.coef_
print("The number of non-zero element before L1 regularization is {}".format(np.count_n
onzero(l2)))
# Please write all the code with proper documentation

clf = LogisticRegression(C=10**-2, penalty='l1');
clf.fit(X_train_bow, y_train)
w = clf.coef_
print("The number of non-zero element after L1 regularization is {}".format(np.count_no
nzero(w)))
```

```
The number of non-zero element before L1 regularization is 40475
The number of non-zero element after L1 regularization is 77
```

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

#### Simple cross validation:

In [78]:

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

from sklearn.linear_model import LogisticRegression
K = [10**-4, 10**-3, 10**-2, 10**-1, 1, 10**1, 10**2, 10**3, 10**4]
#for i in range(1,50,2):
for i in K:
    clf = LogisticRegression(C=i, penalty='l1')
    clf.fit(X_train_bow, y_train)

    # predict the response on the crossvalidation train
    pred = clf.predict(X_cv_bow)

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

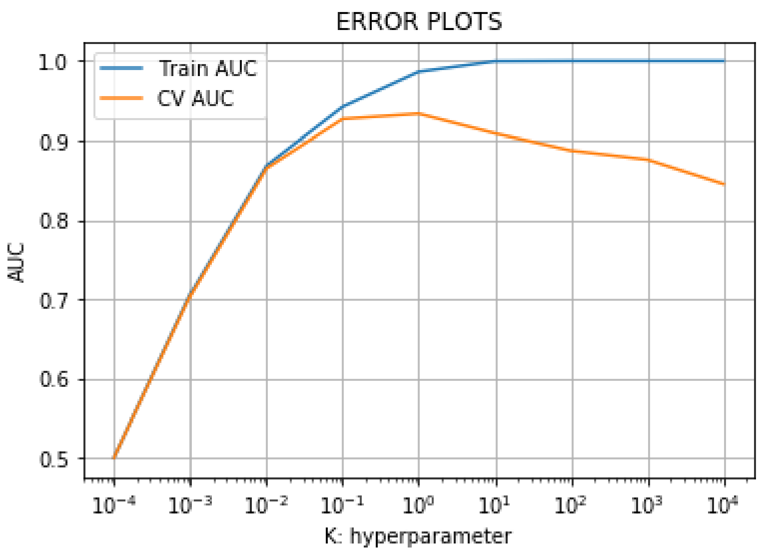
```
CV accuracy for lambda = 0.000100 is 84%

CV accuracy for lambda = 0.001000 is 84%

CV accuracy for lambda = 0.010000 is 86%

CV accuracy for lambda = 0.100000 is 91%

CV accuracy for lambda = 1.000000 is 91%

CV accuracy for lambda = 10.000000 is 90%

CV accuracy for lambda = 100.000000 is 89%

CV accuracy for lambda = 1000.000000 is 88%

CV accuracy for lambda = 10000.000000 is 87%
```

In [79]:

```
from sklearn.metrics import roc_auc_score
import matplotlib.pyplot as plt

train_auc = []
cv_auc = []
#K = [1, 5, 10, 15, 21, 31, 41, 51]
K = [10**-4, 10**-3, 10**-2, 10**-1, 1, 10**1, 10**2, 10**3, 10**4]
for i in K:
    neigh = LogisticRegression(C=i, penalty='l1')
    neigh.fit(X_train_bow, y_train)


    #y_train_pred = []
    #for i in range(0, X_train.shape[0], 1000):
    #    y_train_pred.extend(neigh.predict_proba(X_train_bow[i:i+1000])[:,1])

    #y_cv_pred = []
    #for i in range(0, X_cv.shape[0], 1000):
    #   y_cv_pred.extend(neigh.predict_proba(X_cv_bow[i:i+1000])[:,1])

    y_train_pred =  neigh.predict_proba(X_train_bow)[:,1]
    y_cv_pred =  neigh.predict_proba(X_cv_bow)[:,1]

    train_auc.append(roc_auc_score(y_train,y_train_pred))
    cv_auc.append(roc_auc_score(y_cv, y_cv_pred))


plt.semilogx(K, train_auc, label='Train AUC')
plt.semilogx(K, cv_auc, label='CV AUC')
plt.legend()
plt.grid()
plt.xlabel("K: hyperparameter")
plt.ylabel("AUC")
plt.title("ERROR PLOTS")
```

Out[79]:

Text(0.5,1,'ERROR PLOTS')

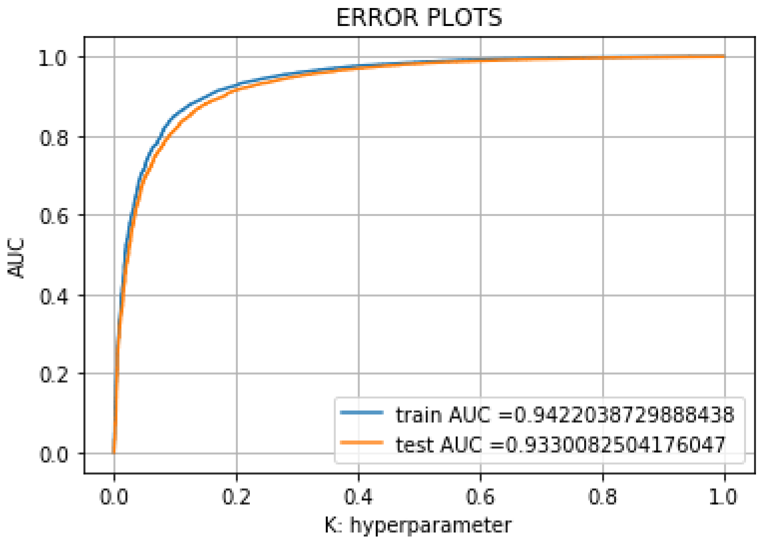



OBSERVATION: The best value of lambda is 10**-1.

## Testing:

In [157]:

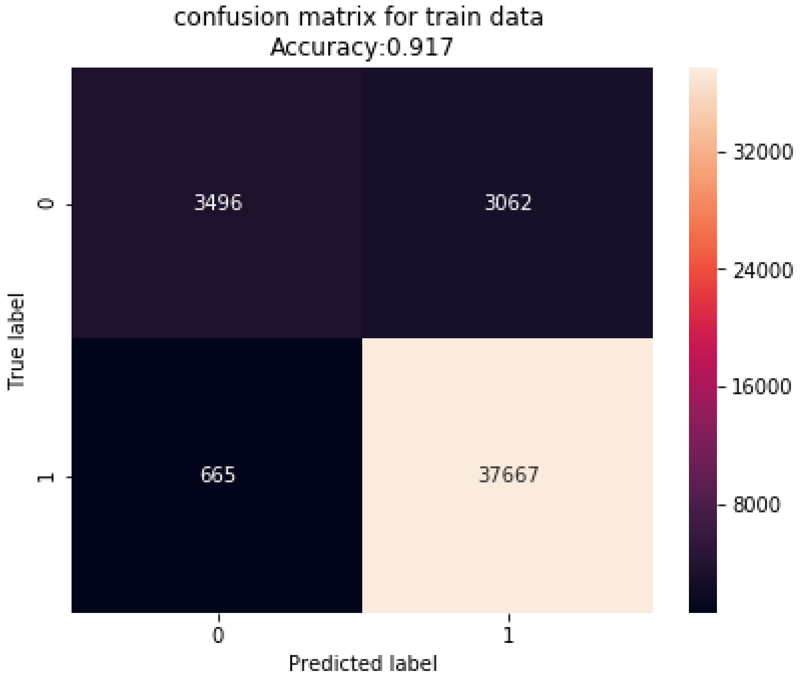
```
# https://scikit-learn.org/stable/modules/generated/sklearn.metrics.roc_curve.html#skle
arn.metrics.roc_curve
from sklearn.metrics import roc_curve, auc


neigh = LogisticRegression(C=10**-1, penalty='l1')
neigh.fit(X_train_bow, y_train)
# roc_auc_score(y_true, y_score) the 2nd parameter should be probability estimates of t
he positive class
# not the predicted outputs

train_fpr, train_tpr, thresholds = roc_curve(y_train, neigh.predict_proba(X_train_bow)
[:,1])
test_fpr, test_tpr, thresholds = roc_curve(y_test, neigh.predict_proba(X_test_bow)[:,1
])

plt.plot(train_fpr, train_tpr, label="train AUC ="+str(auc(train_fpr, train_tpr)))
plt.plot(test_fpr, test_tpr, label="test AUC ="+str(auc(test_fpr, test_tpr)))
plt.legend()
plt.grid()
plt.xlabel("K: hyperparameter")
plt.ylabel("AUC")
plt.title("ERROR PLOTS")
plt.show()

print("="*100)

from sklearn.metrics import confusion_matrix
print("Train confusion matrix")
print(confusion_matrix(y_train, neigh.predict(X_train_bow)))
print("Test confusion matrix")
print(confusion_matrix(y_test, neigh.predict(X_test_bow)))
```

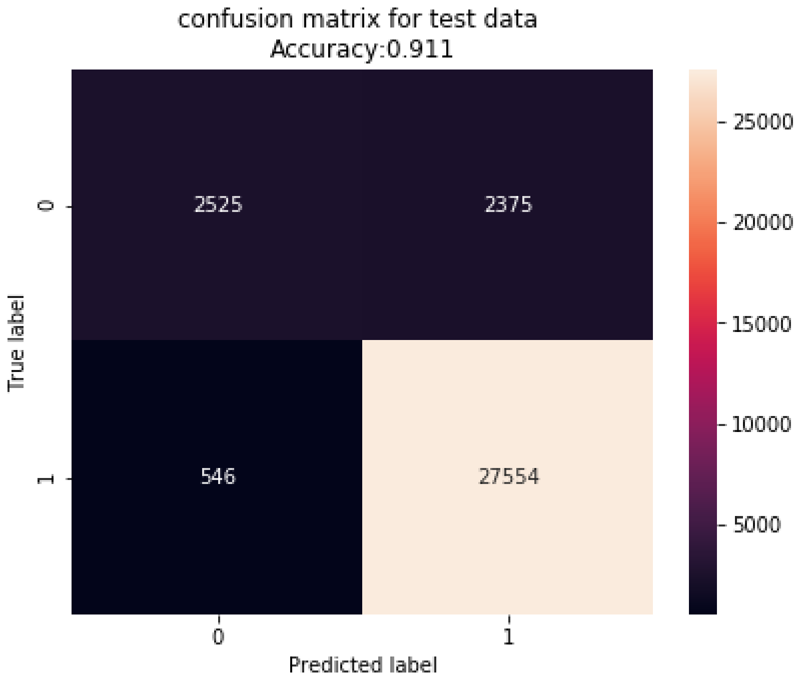



```
====================================================================================================
Train confusion matrix
[[ 3496  3062]
 [  665 37667]]
Test confusion matrix
[[ 2525  2375]
 [  546 27554]]
```

In [81]:

```
# Creates a confusion matrix for train data
cm = confusion_matrix(y_train, neigh.predict(X_train_bow))
cm_df = pd.DataFrame(cm)
plt.figure(figsize=(6.5,5))
sns.heatmap(cm_df, annot=True, fmt="d")
plt.title('confusion matrix for train data \nAccuracy:{0:.3f}'.format(accuracy_score(y_
train, neigh.predict(X_train_bow))))
plt.ylabel('True label')
plt.xlabel('Predicted label')
plt.show()


# Creates a confusion matrix for test data
cm = confusion_matrix(y_test, neigh.predict(X_test_bow))
cm_df = pd.DataFrame(cm)
plt.figure(figsize=(6.5,5))
sns.heatmap(cm_df, annot=True, fmt="d")
plt.title('confusion matrix for test data \nAccuracy:{0:.3f}'.format(accuracy_score(y_t
est, neigh.predict(X_test_bow))))
plt.ylabel('True label')
plt.xlabel('Predicted label')
plt.show()
```




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

In [196]:

```
X_train_bow.data+=2 #adding noise
```

In [200]:

```
clf = LogisticRegression(C=10**-1, penalty='l1')
clf.fit(X_train_bow, y_train)
```

Out[200]:

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

In [283]:

```
w1 = neigh.coef_.T
w2 = clf.coef_.T
```

In [288]:

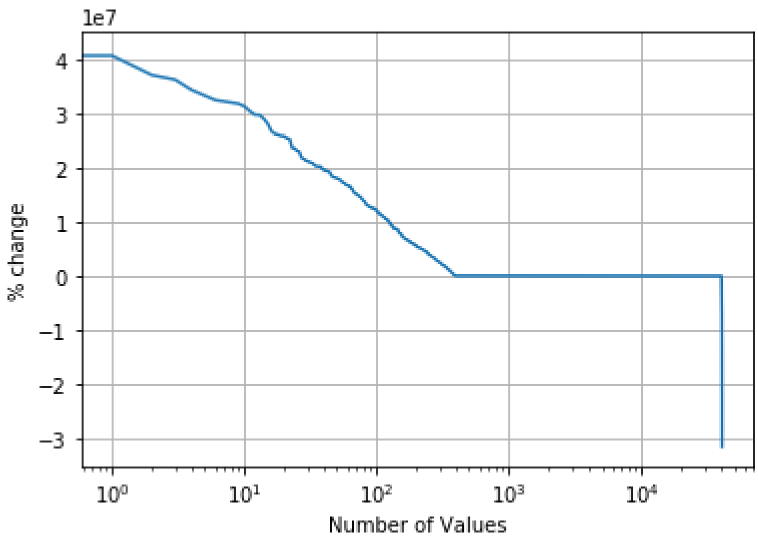
```
w1 = w1 + 10**-6
w2 = w2 + 10**-6
```

In [289]:

```
change = []
for i in range(len(w1)):
    change.append(((w1.item(i)-w2.item(i))/w1.item(i))*100)
```

In [301]:

```
plt.semilogx(change)
plt.xlabel("Number of Values")
plt.ylabel("% change")
plt.grid()
```




Observation: There are around 10**3 points that are multicollinear as depicted above in the graph.

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

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

In [312]:

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

#positive

a = neigh.coef_[0]
a_std = np.argsort(a)
print(np.take(vectorizer.get_feature_names(), a_std)[::-1][:10])
```

```
['delicious' 'perfect' 'beat' 'excellent' 'loves' 'amazing' 'highly'
 'pleased' 'awesome' 'wonderful']
```

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

In [313]:

```
#negative

b = neigh.coef_[0]
b_std = np.argsort(b)
print(np.take(vectorizer.get_feature_names(), b_std)[:10])
```

```
['worst' 'disappointing' 'terrible' 'awful' 'threw' 'horrible'
 'disappointed' 'disappointment' 'tasteless' 'bland']
```

# [5.2] Logistic Regression on TFIDF, SET 2

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

In [332]:

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

tf_idf_vect = TfidfVectorizer(ngram_range=(1,2), min_df=10)
tf_idf_vect.fit(X_train) # fit has to happen only on train data

# we use the fitted CountVectorizer to convert the text to vector
X_train_tf = tf_idf_vect.transform(X_train)
X_cv_tf = tf_idf_vect.transform(X_cv)
X_test_tf = tf_idf_vect.transform(X_test)

print("After vectorizations")
print(X_train_tf.shape, y_train.shape)
print(X_cv_tf.shape, y_cv.shape)
print(X_test_tf.shape, y_test.shape)
```

```
After vectorizations
(44890, 25492) (44890,)
(22110, 25492) (22110,)
(33000, 25492) (33000,)
```

# Simple cross validation:

In [334]:

```
from sklearn.linear_model import LogisticRegression
K = [10**-4, 10**-3, 10**-2, 10**-1, 1, 10**1, 10**2, 10**3, 10**4]
#for i in range(1,50,2):
for i in K:
    clf = LogisticRegression(C=i, penalty='l1')
    clf.fit(X_train_tf, y_train)

    # predict the response on the crossvalidation train
    pred = clf.predict(X_cv_tf)

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

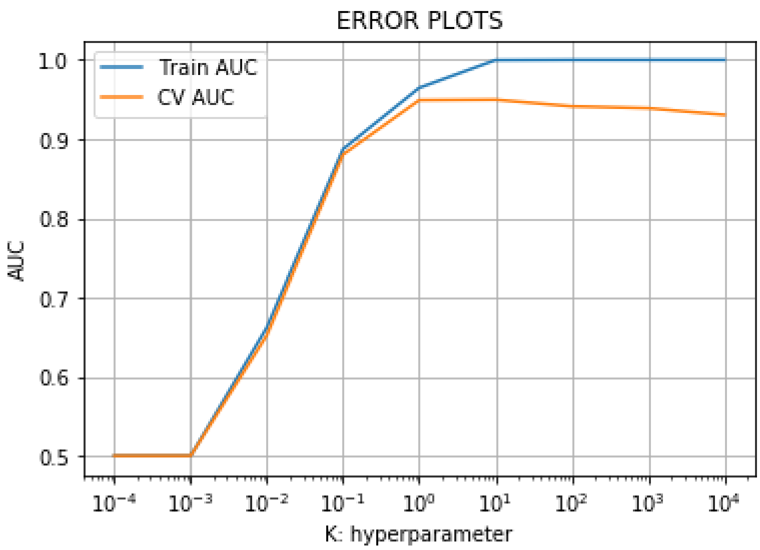
```
CV accuracy for lambda = 0.000100 is 84%

CV accuracy for lambda = 0.001000 is 84%

CV accuracy for lambda = 0.010000 is 84%

CV accuracy for lambda = 0.100000 is 87%

CV accuracy for lambda = 1.000000 is 92%

CV accuracy for lambda = 10.000000 is 92%

CV accuracy for lambda = 100.000000 is 91%

CV accuracy for lambda = 1000.000000 is 91%

CV accuracy for lambda = 10000.000000 is 91%
```

In [335]:

```
from sklearn.metrics import roc_auc_score
import matplotlib.pyplot as plt

train_auc = []
cv_auc = []
#K = [1, 5, 10, 15, 21, 31, 41, 51]
K = [10**-4, 10**-3, 10**-2, 10**-1, 1, 10**1, 10**2, 10**3, 10**4]
for i in K:
    neigh = LogisticRegression(C=i, penalty='l1')
    neigh.fit(X_train_tf, y_train)

    y_train_pred =  neigh.predict_proba(X_train_tf)[:,1]
    y_cv_pred =  neigh.predict_proba(X_cv_tf)[:,1]

    train_auc.append(roc_auc_score(y_train,y_train_pred))
    cv_auc.append(roc_auc_score(y_cv, y_cv_pred))


plt.semilogx(K, train_auc, label='Train AUC')
plt.semilogx(K, cv_auc, label='CV AUC')
plt.legend()
plt.grid()
plt.xlabel("K: hyperparameter")
plt.ylabel("AUC")
plt.title("ERROR PLOTS")
```

Out[335]:

```
Text(0.5,1,'ERROR PLOTS')
```

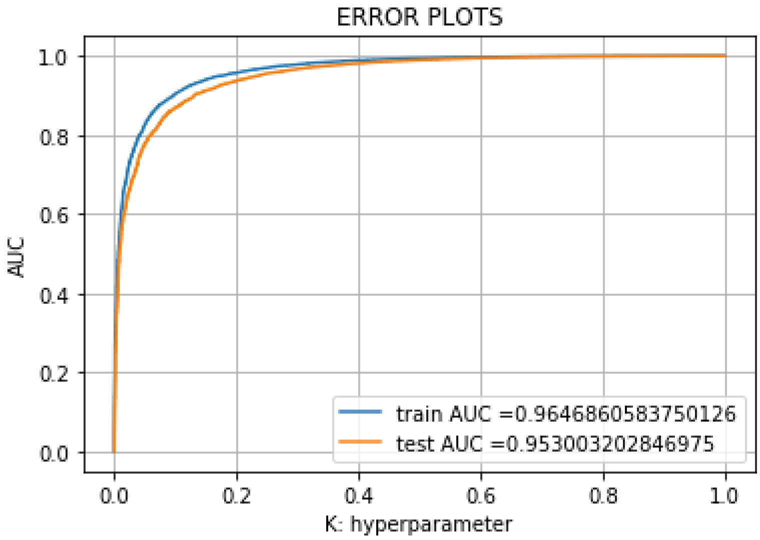



OBSERVATION: The best value of lambda is 1.

## Testing:

In [336]:

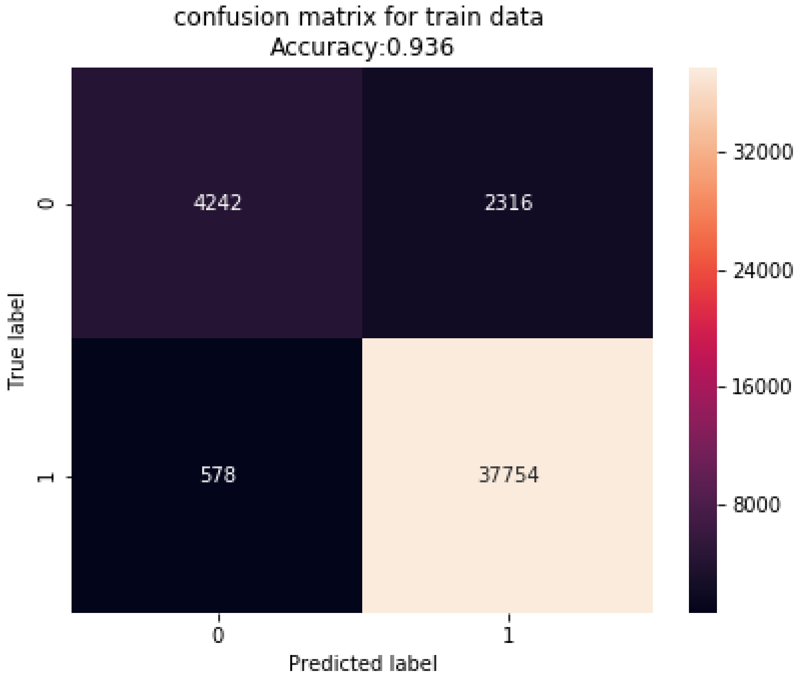
```
# https://scikit-learn.org/stable/modules/generated/sklearn.metrics.roc_curve.html#skle
arn.metrics.roc_curve
from sklearn.metrics import roc_curve, auc


neigh = LogisticRegression(C=1, penalty='l1')
neigh.fit(X_train_tf, y_train)
# roc_auc_score(y_true, y_score) the 2nd parameter should be probability estimates of t
he positive class
# not the predicted outputs

train_fpr, train_tpr, thresholds = roc_curve(y_train, neigh.predict_proba(X_train_tf)
[:,1])
test_fpr, test_tpr, thresholds = roc_curve(y_test, neigh.predict_proba(X_test_tf)[:,1])

plt.plot(train_fpr, train_tpr, label="train AUC ="+str(auc(train_fpr, train_tpr)))
plt.plot(test_fpr, test_tpr, label="test AUC ="+str(auc(test_fpr, test_tpr)))
plt.legend()
plt.grid()
plt.xlabel("K: hyperparameter")
plt.ylabel("AUC")
plt.title("ERROR PLOTS")
plt.show()

print("="*100)

from sklearn.metrics import confusion_matrix
print("Train confusion matrix")
print(confusion_matrix(y_train, neigh.predict(X_train_tf)))
print("Test confusion matrix")
print(confusion_matrix(y_test, neigh.predict(X_test_tf)))
```

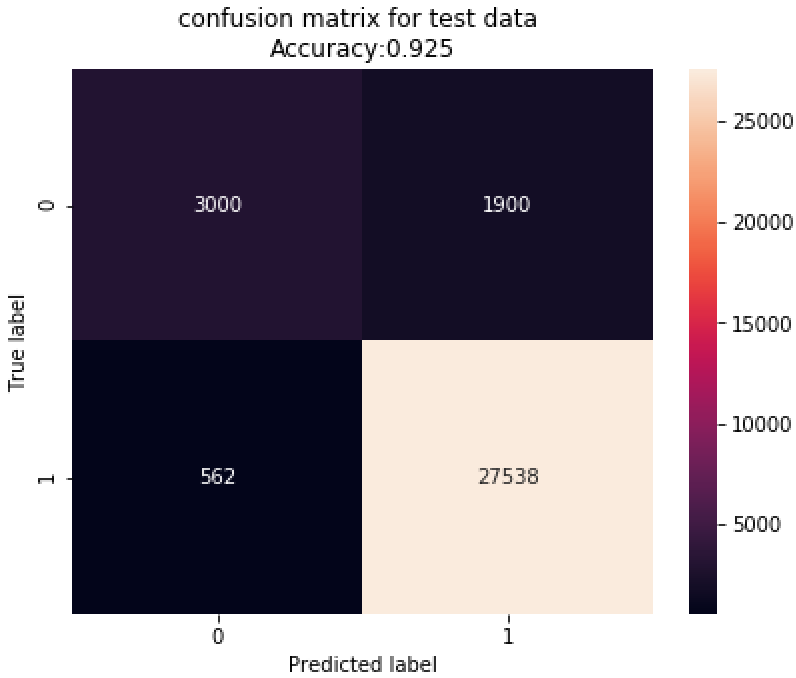



```
================================================================================
==========================
Train confusion matrix
[[ 4242  2316]
 [  578 37754]]
Test confusion matrix
[[ 3000  1900]
 [  562 27538]]
```

In [337]:

```
# Creates a confusion matrix for train data
cm = confusion_matrix(y_train, neigh.predict(X_train_tf))
cm_df = pd.DataFrame(cm)
plt.figure(figsize=(6.5,5))
sns.heatmap(cm_df, annot=True, fmt="d")
plt.title('confusion matrix for train data \nAccuracy:{0:.3f}'.format(accuracy_score(y_
train, neigh.predict(X_train_tf))))
plt.ylabel('True label')
plt.xlabel('Predicted label')
plt.show()


# Creates a confusion matrix for test data
cm = confusion_matrix(y_test, neigh.predict(X_test_tf))
cm_df = pd.DataFrame(cm)
plt.figure(figsize=(6.5,5))
sns.heatmap(cm_df, annot=True, fmt="d")
plt.title('confusion matrix for test data \nAccuracy:{0:.3f}'.format(accuracy_score(y_t
est, neigh.predict(X_test_tf))))
plt.ylabel('True label')
plt.xlabel('Predicted label')
plt.show()
```


confusion matrix for train data
Accuracy:0.936
4242
2316
578
37754
0
1
True label
0
1
Predicted label
32000
24000
16000
8000


confusion matrix for test data
Accuracy:0.925
3000
1900
562
27538
0
1
True label
0
1
Predicted label
25000
20000
15000
10000
5000

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

## Simple Cross Validation:

In [338]:

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

from sklearn.linear_model import LogisticRegression
K = [10**-4, 10**-3, 10**-2, 10**-1, 1, 10**1, 10**2, 10**3, 10**4]
#for i in range(1,50,2):
for i in K:
    clf = LogisticRegression(C=i)
    clf.fit(X_train_tf, y_train)

    # predict the response on the crossvalidation train
    pred = clf.predict(X_cv_tf)

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

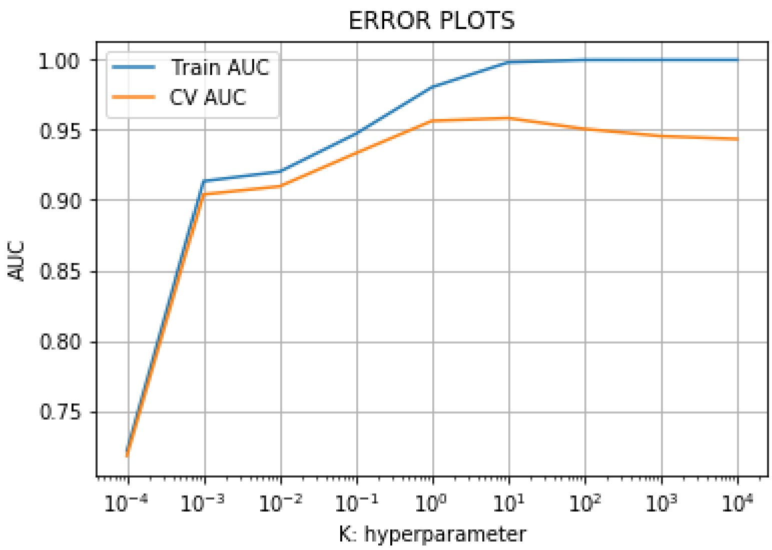
```
CV accuracy for lambda = 0.000100 is 84%

CV accuracy for lambda = 0.001000 is 84%

CV accuracy for lambda = 0.010000 is 84%

CV accuracy for lambda = 0.100000 is 86%

CV accuracy for lambda = 1.000000 is 92%

CV accuracy for lambda = 10.000000 is 93%

CV accuracy for lambda = 100.000000 is 92%

CV accuracy for lambda = 1000.000000 is 92%

CV accuracy for lambda = 10000.000000 is 91%
```

In [339]:

```
from sklearn.metrics import roc_auc_score
import matplotlib.pyplot as plt

train_auc = []
cv_auc = []
#K = [1, 5, 10, 15, 21, 31, 41, 51]
K = [10**-4, 10**-3, 10**-2, 10**-1, 1, 10**1, 10**2, 10**3, 10**4]
for i in K:
    neigh = LogisticRegression(C=i)
    neigh.fit(X_train_tf, y_train)

    y_train_pred =  neigh.predict_proba(X_train_tf)[:,1]
    y_cv_pred =  neigh.predict_proba(X_cv_tf)[:,1]

    train_auc.append(roc_auc_score(y_train,y_train_pred))
    cv_auc.append(roc_auc_score(y_cv, y_cv_pred))


plt.semilogx(K, train_auc, label='Train AUC')
plt.semilogx(K, cv_auc, label='CV AUC')
plt.legend()
plt.grid()
plt.xlabel("K: hyperparameter")
plt.ylabel("AUC")
plt.title("ERROR PLOTS")
```

Out[339]:

Text(0.5,1,'ERROR PLOTS')

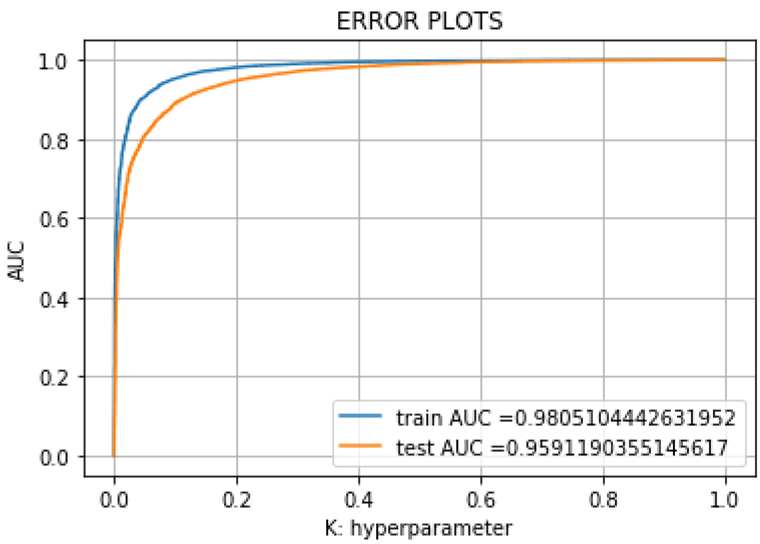



OBSERVATION: The best value of lambda is 1.

# Testing :

In [340]:

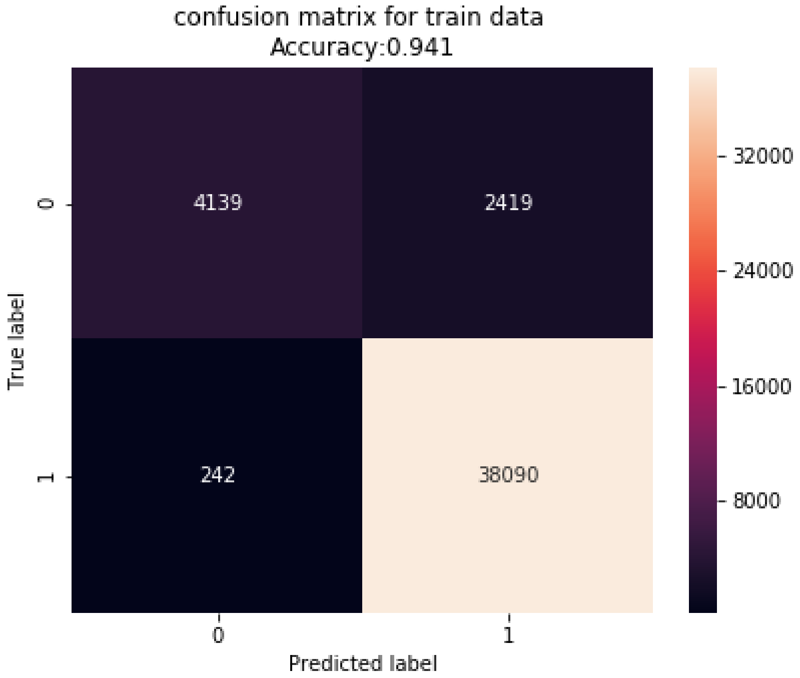
```
# https://scikit-learn.org/stable/modules/generated/sklearn.metrics.roc_curve.html#skle
arn.metrics.roc_curve
from sklearn.metrics import roc_curve, auc


neigh = LogisticRegression(C=1)
neigh.fit(X_train_tf, y_train)
# roc_auc_score(y_true, y_score) the 2nd parameter should be probability estimates of t
he positive class
# not the predicted outputs

train_fpr, train_tpr, thresholds = roc_curve(y_train, neigh.predict_proba(X_train_tf)
[:,1])
test_fpr, test_tpr, thresholds = roc_curve(y_test, neigh.predict_proba(X_test_tf)[:,1])

plt.plot(train_fpr, train_tpr, label="train AUC ="+str(auc(train_fpr, train_tpr)))
plt.plot(test_fpr, test_tpr, label="test AUC ="+str(auc(test_fpr, test_tpr)))
plt.legend()
plt.grid()
plt.xlabel("K: hyperparameter")
plt.ylabel("AUC")
plt.title("ERROR PLOTS")
plt.show()

print("="*100)

from sklearn.metrics import confusion_matrix
print("Train confusion matrix")
print(confusion_matrix(y_train, neigh.predict(X_train_tf)))
print("Test confusion matrix")
print(confusion_matrix(y_test, neigh.predict(X_test_tf)))
```

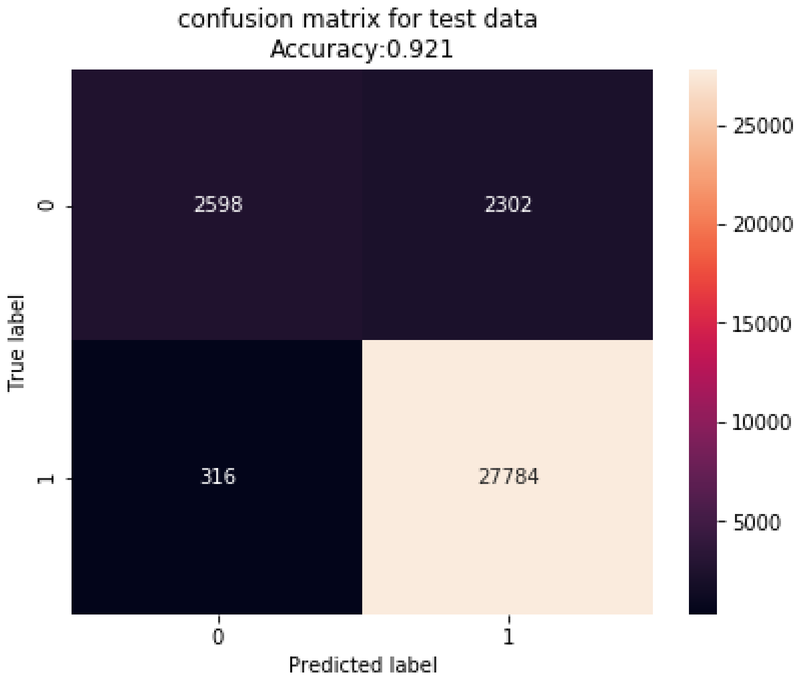



```
================================================================================
==========================
Train confusion matrix
[[ 4139  2419]
 [  242 38090]]
Test confusion matrix
[[ 2598  2302]
 [  316 27784]]
```

In [341]:

```
# Creates a confusion matrix for train data
cm = confusion_matrix(y_train, neigh.predict(X_train_tf))
cm_df = pd.DataFrame(cm)
plt.figure(figsize=(6.5,5))
sns.heatmap(cm_df, annot=True, fmt="d")
plt.title('confusion matrix for train data \nAccuracy:{0:.3f}'.format(accuracy_score(y_
train, neigh.predict(X_train_tf))))
plt.ylabel('True label')
plt.xlabel('Predicted label')
plt.show()


# Creates a confusion matrix for test data
cm = confusion_matrix(y_test, neigh.predict(X_test_tf))
cm_df = pd.DataFrame(cm)
plt.figure(figsize=(6.5,5))
sns.heatmap(cm_df, annot=True, fmt="d")
plt.title('confusion matrix for test data \nAccuracy:{0:.3f}'.format(accuracy_score(y_t
est, neigh.predict(X_test_tf))))
plt.ylabel('True label')
plt.xlabel('Predicted label')
plt.show()
```




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

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

In [343]:

```
#positive

a = neigh.coef_[0]
a_std = np.argsort(a)
print(np.take(tf_idf_vect.get_feature_names(), a_std)[::-1][:10])
```

```
['great' 'best' 'delicious' 'love' 'good' 'perfect' 'loves' 'excellent'
 'wonderful' 'favorite']
```

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

In [344]:

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

b = neigh.coef_[0]
b_std = np.argsort(b)
print(np.take(tf_idf_vect.get_feature_names(), b_std)[:10])
```

```
['disappointed' 'not' 'worst' 'terrible' 'awful' 'horrible'
 'not recommend' 'disappointing' 'not buy' 'stale']
```

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

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

In [32]:

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

In [33]:

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

elif want_to_use_google_w2v and is_your_ram_gt_16g:
    if os.path.isfile('GoogleNews-vectors-negative300.bin'):
        w2v_model=KeyedVectors.load_word2vec_format('GoogleNews-vectors-negative300.bi
n', 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_train_w2v = True, to
train your own w2v ")
```

```
[('good', 0.8069188594818115), ('excellent', 0.806404173374176), ('wonderf
ul', 0.786535918712616), ('amazing', 0.7736546397209167), ('fantastic', 0.
757821798324585), ('perfect', 0.7529041171073914), ('delicious', 0.6939063
668251038), ('awesome', 0.6899166703224182), ('terrific', 0.68481177091598
51), ('outstanding', 0.6703290343284607)]
==================================================
[('theater', 0.771755576133728), ('best', 0.7685863375663757), ('closest',
0.7642993330955505), ('tastiest', 0.7582318186759949), ('funniest', 0.7348
730564117432), ('worse', 0.7309377193450928), ('disgusting', 0.72418618202
20947), ('eaten', 0.7240061163902283), ('awful', 0.7225335240364075), ('e
h', 0.7166944146156311)]
```

In [34]:

```
w2v_words = list(w2v_model.wv.vocab)
```

In [348]:

```
# average Word2Vec
# compute average word2vec for each review.
X_train_aw2v = []; # the avg-w2v for each sentence/review is stored in this list
for sent in list_of_sentance: # for each review/sentence
    sent_vec = np.zeros(50) # as word vectors are of zero length 50, you 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/review
    for word in sent: # for each word in a review/sentence
        if word in (w2v_words):
            vec = w2v_model.wv[word]
            sent_vec += vec
            cnt_words += 1
    if cnt_words != 0:
        sent_vec /= cnt_words
    X_train_aw2v.append(sent_vec)
```

In [349]:

```
# Train your own Word2Vec model using your own text corpus
i=0
X_cv_w2v = []
for sentance in X_cv:
    X_cv_w2v.append(sentance.split())
```

In [350]:

```
# Training for cv

X_cv_aw2v = []; # the avg-w2v for each sentence/review is stored in this list
for sent in X_cv_w2v: # for each review/sentence
    sent_vec = np.zeros(50) # as word vectors are of zero length 50, you 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/review
    for word in sent: # for each word in a review/sentence
        if word in (w2v_words):
            vec = w2v_model.wv[word]
            sent_vec += vec
            cnt_words += 1
    if cnt_words != 0:
        sent_vec /= cnt_words
    X_cv_aw2v.append(sent_vec)
```

In [351]:

```
# Train your own Word2Vec model using your own text corpus
i=0
X_test_w2v = []
for sentance in X_test:
    X_test_w2v.append(sentance.split())
```

In [352]:

```
# Training for test

X_test_aw2v = []; # the avg-w2v for each sentence/review is stored in this list
for sent in X_test_w2v: # for each review/sentence
    sent_vec = np.zeros(50) # as word vectors are of zero length 50, you 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/review
    for word in sent: # for each word in a review/sentence
        if word in (w2v_words):
            vec = w2v_model.wv[word]
            sent_vec += vec
            cnt_words += 1
    if cnt_words != 0:
        sent_vec /= cnt_words
    X_test_aw2v.append(sent_vec)
```

In [353]:

```
print(len(X_train_aw2v), y_train.shape)
print(len(X_cv_aw2v), y_cv.shape)
print(len(X_test_aw2v), y_test.shape)
```

```
44890 (44890,)
22110 (22110,)
33000 (33000,)
```

## Simple cross validation:

In [354]:

```
from sklearn.linear_model import LogisticRegression
K = [10**-4, 10**-3, 10**-2, 10**-1, 1, 10**1, 10**2, 10**3, 10**4]
#for i in range(1,50,2):
for i in K:
    clf = LogisticRegression(C=i, penalty='l1')
    clf.fit(X_train_aw2v, y_train)

    # predict the response on the crossvalidation train
    pred = clf.predict(X_cv_aw2v)

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

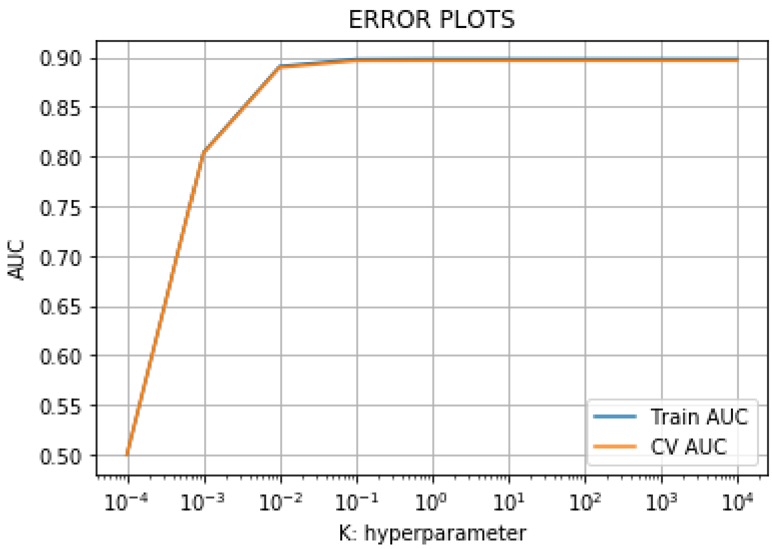
```
CV accuracy for lambda = 0.000100 is 84%

CV accuracy for lambda = 0.001000 is 84%

CV accuracy for lambda = 0.010000 is 87%

CV accuracy for lambda = 0.100000 is 88%

CV accuracy for lambda = 1.000000 is 88%

CV accuracy for lambda = 10.000000 is 88%

CV accuracy for lambda = 100.000000 is 88%

CV accuracy for lambda = 1000.000000 is 88%

CV accuracy for lambda = 10000.000000 is 88%
```

In [356]:

```
from sklearn.metrics import roc_auc_score
import matplotlib.pyplot as plt

train_auc = []
cv_auc = []
#K = [1, 5, 10, 15, 21, 31, 41, 51]
K = [10**-4, 10**-3, 10**-2, 10**-1, 1, 10**1, 10**2, 10**3, 10**4]
for i in K:
    neigh = LogisticRegression(C=i, penalty = 'l1')
    neigh.fit(X_train_aw2v, y_train)

    y_train_pred =  neigh.predict_proba(X_train_aw2v)[:,1]
    y_cv_pred =  neigh.predict_proba(X_cv_aw2v)[:,1]

    train_auc.append(roc_auc_score(y_train,y_train_pred))
    cv_auc.append(roc_auc_score(y_cv, y_cv_pred))


plt.semilogx(K, train_auc, label='Train AUC')
plt.semilogx(K, cv_auc, label='CV AUC')
plt.legend()
plt.grid()
plt.xlabel("K: hyperparameter")
plt.ylabel("AUC")
plt.title("ERROR PLOTS")
```

Out[356]:

```
Text(0.5,1,'ERROR PLOTS')
```

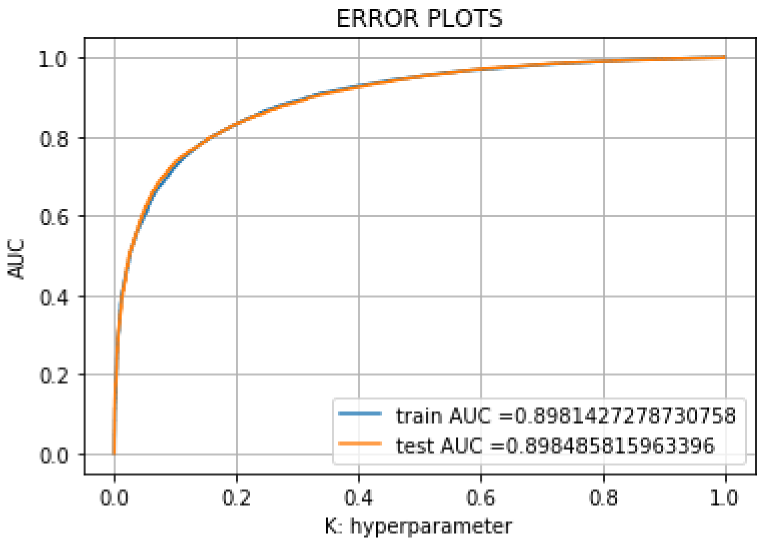



Observation:- Best value of lambda is 10**-1

# Testing :

In [358]:

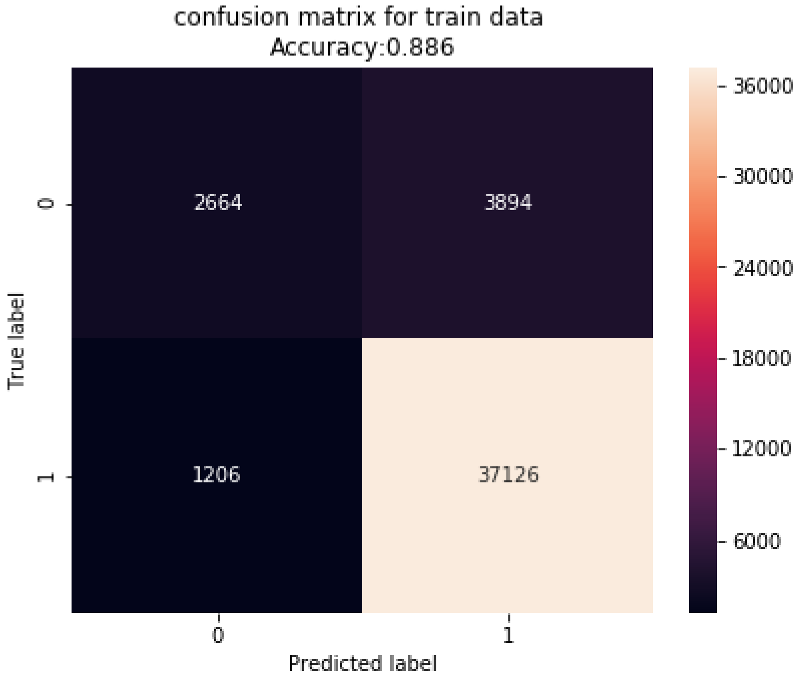
```
# https://scikit-learn.org/stable/modules/generated/sklearn.metrics.roc_curve.html#skle
arn.metrics.roc_curve
from sklearn.metrics import roc_curve, auc


neigh = LogisticRegression(C=10**-1, penalty = 'l1')
neigh.fit(X_train_aw2v, y_train)
# roc_auc_score(y_true, y_score) the 2nd parameter should be probability estimates of t
he positive class
# not the predicted outputs

train_fpr, train_tpr, thresholds = roc_curve(y_train, neigh.predict_proba(X_train_aw2v)
[:,1])
test_fpr, test_tpr, thresholds = roc_curve(y_test, neigh.predict_proba(X_test_aw2v)[:,1
])

plt.plot(train_fpr, train_tpr, label="train AUC ="+str(auc(train_fpr, train_tpr)))
plt.plot(test_fpr, test_tpr, label="test AUC ="+str(auc(test_fpr, test_tpr)))
plt.legend()
plt.grid()
plt.xlabel("K: hyperparameter")
plt.ylabel("AUC")
plt.title("ERROR PLOTS")
plt.show()

print("="*100)

from sklearn.metrics import confusion_matrix
print("Train confusion matrix")
print(confusion_matrix(y_train, neigh.predict(X_train_aw2v)))
print("Test confusion matrix")
print(confusion_matrix(y_test, neigh.predict(X_test_aw2v)))
```

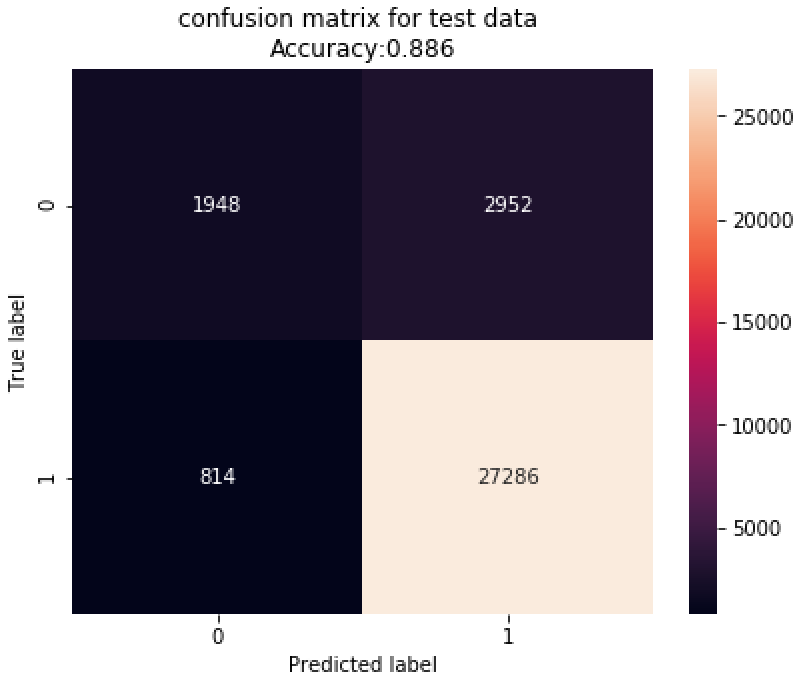



```
============================================================================
=========================
Train confusion matrix
[[ 2664  3894]
 [ 1206 37126]]
Test confusion matrix
[[ 1948  2952]
 [  814 27286]]
```

In [359]:

```
# Creates a confusion matrix for train data
cm = confusion_matrix(y_train, neigh.predict(X_train_aw2v))
cm_df = pd.DataFrame(cm)
plt.figure(figsize=(6.5,5))
sns.heatmap(cm_df, annot=True, fmt="d")
plt.title('confusion matrix for train data \nAccuracy:{0:.3f}'.format(accuracy_score(y_
train, neigh.predict(X_train_aw2v))))
plt.ylabel('True label')
plt.xlabel('Predicted label')
plt.show()


# Creates a confusion matrix for test data
cm = confusion_matrix(y_test, neigh.predict(X_test_aw2v))
cm_df = pd.DataFrame(cm)
plt.figure(figsize=(6.5,5))
sns.heatmap(cm_df, annot=True, fmt="d")
plt.title('confusion matrix for test data \nAccuracy:{0:.3f}'.format(accuracy_score(y_t
est, neigh.predict(X_test_aw2v))))
plt.ylabel('True label')
plt.xlabel('Predicted label')
plt.show()
```


confusion matrix for train data
Accuracy:0.886
2664
3894
1206
37126
True label
Predicted label
0
1
0
1
36000
30000
24000
18000
12000
6000


confusion matrix for test data
Accuracy:0.886
1948
2952
814
27286
True label
Predicted label
0
1
0
1
25000
20000
15000
10000
5000

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

In [360]:

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

from sklearn.linear_model import LogisticRegression
K = [10**-4, 10**-3, 10**-2, 10**-1, 1, 10**1, 10**2, 10**3, 10**4]
#for i in range(1,50,2):
for i in K:
    clf = LogisticRegression(C=i)
    clf.fit(X_train_aw2v, y_train)

    # predict the response on the crossvalidation train
    pred = clf.predict(X_cv_aw2v)

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

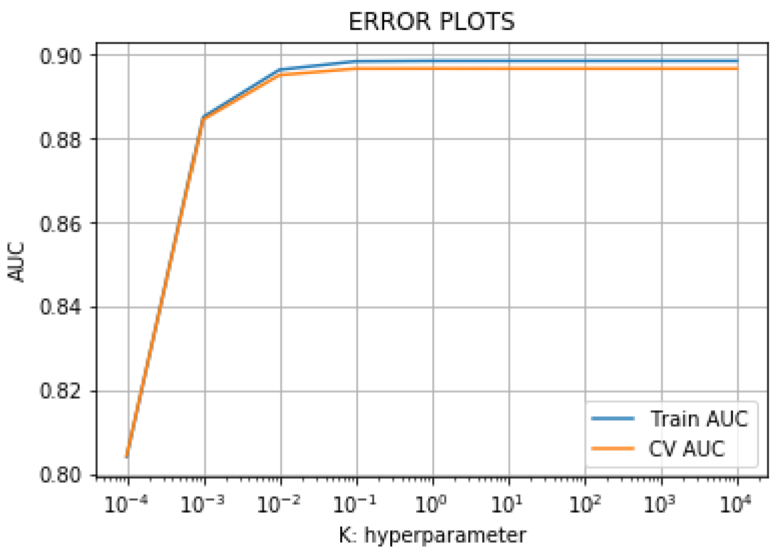
```
CV accuracy for lambda = 0.000100 is 84%

CV accuracy for lambda = 0.001000 is 86%

CV accuracy for lambda = 0.010000 is 88%

CV accuracy for lambda = 0.100000 is 88%

CV accuracy for lambda = 1.000000 is 88%

CV accuracy for lambda = 10.000000 is 88%

CV accuracy for lambda = 100.000000 is 88%

CV accuracy for lambda = 1000.000000 is 88%

CV accuracy for lambda = 10000.000000 is 88%
```

In [361]:

```
from sklearn.metrics import roc_auc_score
import matplotlib.pyplot as plt

train_auc = []
cv_auc = []
#K = [1, 5, 10, 15, 21, 31, 41, 51]
K = [10**-4, 10**-3, 10**-2, 10**-1, 1, 10**1, 10**2, 10**3, 10**4]
for i in K:
    neigh = LogisticRegression(C=i)
    neigh.fit(X_train_aw2v, y_train)

    y_train_pred =  neigh.predict_proba(X_train_aw2v)[:,1]
    y_cv_pred =  neigh.predict_proba(X_cv_aw2v)[:,1]

    train_auc.append(roc_auc_score(y_train,y_train_pred))
    cv_auc.append(roc_auc_score(y_cv, y_cv_pred))


plt.semilogx(K, train_auc, label='Train AUC')
plt.semilogx(K, cv_auc, label='CV AUC')
plt.legend()
plt.grid()
plt.xlabel("K: hyperparameter")
plt.ylabel("AUC")
plt.title("ERROR PLOTS")
```

Out[361]:

```
Text(0.5,1,'ERROR PLOTS')
```

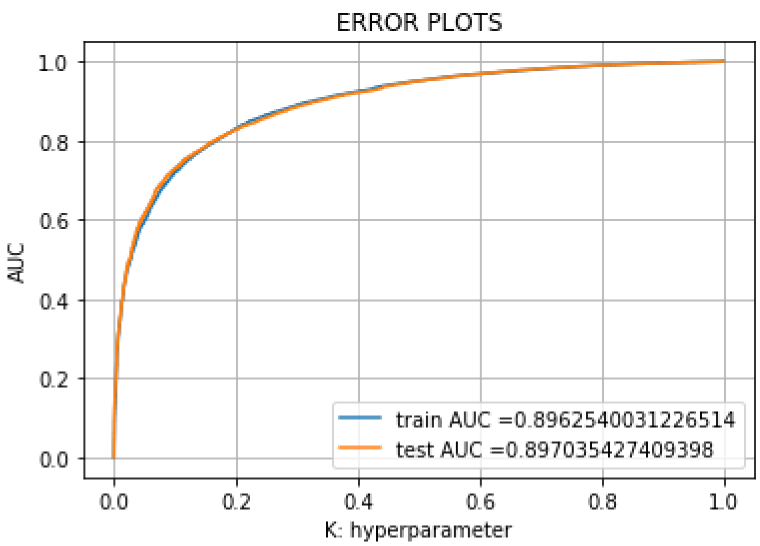



Observation: Best lambda = 10**-2

## Testing:

In [362]:

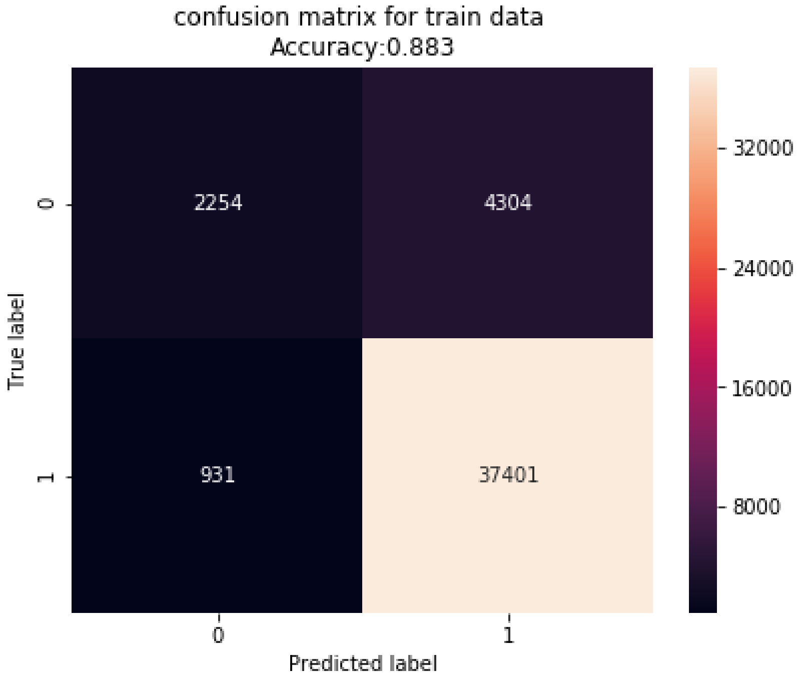
```
# https://scikit-learn.org/stable/modules/generated/sklearn.metrics.roc_curve.html#skle
arn.metrics.roc_curve
from sklearn.metrics import roc_curve, auc


neigh = LogisticRegression(C=10**-2)
neigh.fit(X_train_aw2v, y_train)
# roc_auc_score(y_true, y_score) the 2nd parameter should be probability estimates of t
he positive class
# not the predicted outputs

train_fpr, train_tpr, thresholds = roc_curve(y_train, neigh.predict_proba(X_train_aw2v)
[:,1])
test_fpr, test_tpr, thresholds = roc_curve(y_test, neigh.predict_proba(X_test_aw2v)[:,1
])

plt.plot(train_fpr, train_tpr, label="train AUC ="+str(auc(train_fpr, train_tpr)))
plt.plot(test_fpr, test_tpr, label="test AUC ="+str(auc(test_fpr, test_tpr)))
plt.legend()
plt.grid()
plt.xlabel("K: hyperparameter")
plt.ylabel("AUC")
plt.title("ERROR PLOTS")
plt.show()

print("="*100)

from sklearn.metrics import confusion_matrix
print("Train confusion matrix")
print(confusion_matrix(y_train, neigh.predict(X_train_aw2v)))
print("Test confusion matrix")
print(confusion_matrix(y_test, neigh.predict(X_test_aw2v)))
```

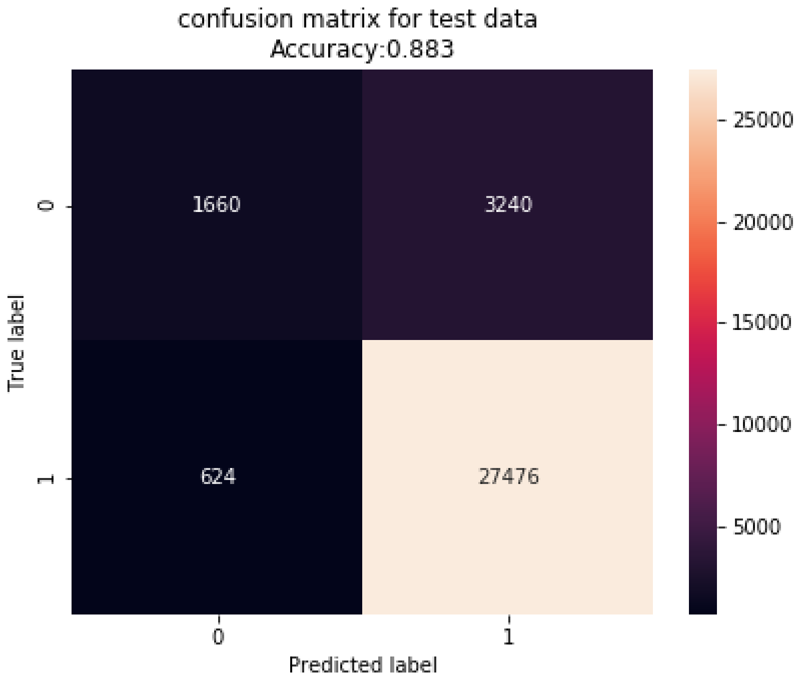



```
===========================================================================
=========================
Train confusion matrix
[[ 2254  4304]
 [  931 37401]]
Test confusion matrix
[[ 1660  3240]
 [  624 27476]]
```

In [363]:

```
# Creates a confusion matrix for train data
cm = confusion_matrix(y_train, neigh.predict(X_train_aw2v))
cm_df = pd.DataFrame(cm)
plt.figure(figsize=(6.5,5))
sns.heatmap(cm_df, annot=True, fmt="d")
plt.title('confusion matrix for train data \nAccuracy:{0:.3f}'.format(accuracy_score(y_
train, neigh.predict(X_train_aw2v))))
plt.ylabel('True label')
plt.xlabel('Predicted label')
plt.show()


# Creates a confusion matrix for test data
cm = confusion_matrix(y_test, neigh.predict(X_test_aw2v))
cm_df = pd.DataFrame(cm)
plt.figure(figsize=(6.5,5))
sns.heatmap(cm_df, annot=True, fmt="d")
plt.title('confusion matrix for test data \nAccuracy:{0:.3f}'.format(accuracy_score(y_t
est, neigh.predict(X_test_aw2v))))
plt.ylabel('True label')
plt.xlabel('Predicted label')
plt.show()
```


confusion matrix for train data
Accuracy:0.883
True label
Predicted label
0
1
2254
4304
931
37401
32000
24000
16000
8000


confusion matrix for test data
Accuracy:0.883
True label
Predicted label
0
1
1660
3240
624
27476
25000
20000
15000
10000
5000

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

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

In [29]:

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

In [30]:

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

In [35]:

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

X_train_tfw2v = []; # the tfidf-w2v for each sentence/review is stored in this list
row=0;
for sent in list_of_sentance: # for each review/sentence
    sent_vec = np.zeros(50) # as word vectors are of zero length
    weight_sum =0; # num of words with a valid vector in the sentence/review
    for word in sent: # for each word in a review/sentence
        if word in w2v_words and word in tfidf_feat:
            vec = w2v_model.wv[word]
#             tf_idf = tf_idf_matrix[row, tfidf_feat.index(word)]
            # to reduce the computation we are
            # dictionary[word] = idf value of word in whole courpus
            # sent.count(word) = tf valeus of word in this review
            tf_idf = dictionary[word]*(sent.count(word)/len(sent))
            sent_vec += (vec * tf_idf)
            weight_sum += tf_idf
    if weight_sum != 0:
        sent_vec /= weight_sum
    X_train_tfw2v.append(sent_vec)
    row += 1
```

In [36]:

```
i=0
X_cv_w2v = []
for sentance in X_cv:
    X_cv_w2v.append(sentance.split())
```

In [37]:

```
tf_idf_matrix = model.transform(X_cv)
# we are converting a dictionary with word as a key, and the idf as a value
dictionary = dict(zip(model.get_feature_names(), list(model.idf_)))

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

X_cv_tfw2v = []; # the tfidf-w2v for each sentence/review is stored in this list
row=0;
for sent in X_cv_w2v: # for each review/sentence
    sent_vec = np.zeros(50) # as word vectors are of zero length
    weight_sum =0; # num of words with a valid vector in the sentence/review
    for word in sent: # for each word in a review/sentence
        if word in w2v_words and word in tfidf_feat:
            vec = w2v_model.wv[word]
#             tf_idf = tf_idf_matrix[row, tfidf_feat.index(word)]
            # to reduce the computation we are
            # dictionary[word] = idf value of word in whole courpus
            # sent.count(word) = tf valeus of word in this review
            tf_idf = dictionary[word]*(sent.count(word)/len(sent))
            sent_vec += (vec * tf_idf)
            weight_sum += tf_idf
    if weight_sum != 0:
        sent_vec /= weight_sum
    X_cv_tfw2v.append(sent_vec)
    row += 1
```

In [38]:

```
i=0
X_test_w2v = []
for sentance in X_test:
    X_test_w2v.append(sentance.split())
```

In [39]:

```
tf_idf_matrix = model.transform(X_train)
# we are converting a dictionary with word as a key, and the idf as a value
dictionary = dict(zip(model.get_feature_names(), list(model.idf_)))


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

X_test_tfw2v = []; # the tfidf-w2v for each sentence/review is stored in this list
row=0;
for sent in X_test_w2v: # for each review/sentence
    sent_vec = np.zeros(50) # as word vectors are of zero length
    weight_sum =0; # num of words with a valid vector in the sentence/review
    for word in sent: # for each word in a review/sentence
        if word in w2v_words and word in tfidf_feat:
            vec = w2v_model.wv[word]
#             tf_idf = tf_idf_matrix[row, tfidf_feat.index(word)]
            # to reduce the computation we are
            # dictionary[word] = idf value of word in whole courpus
            # sent.count(word) = tf valeus of word in this review
            tf_idf = dictionary[word]*(sent.count(word)/len(sent))
            sent_vec += (vec * tf_idf)
            weight_sum += tf_idf
    if weight_sum != 0:
        sent_vec /= weight_sum
    X_test_tfw2v.append(sent_vec)
    row += 1
```

In [40]:

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

from sklearn.linear_model import LogisticRegression
K = [10**-4, 10**-3, 10**-2, 10**-1, 1, 10**1, 10**2, 10**3, 10**4]
#for i in range(1,50,2):
for i in K:
    clf = LogisticRegression(C=i, penalty='l1', class_weight = 'balanced')
    clf.fit(X_train_tfw2v, y_train)

    # predict the response on the crossvalidation train
    pred = clf.predict(X_cv_tfw2v)

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

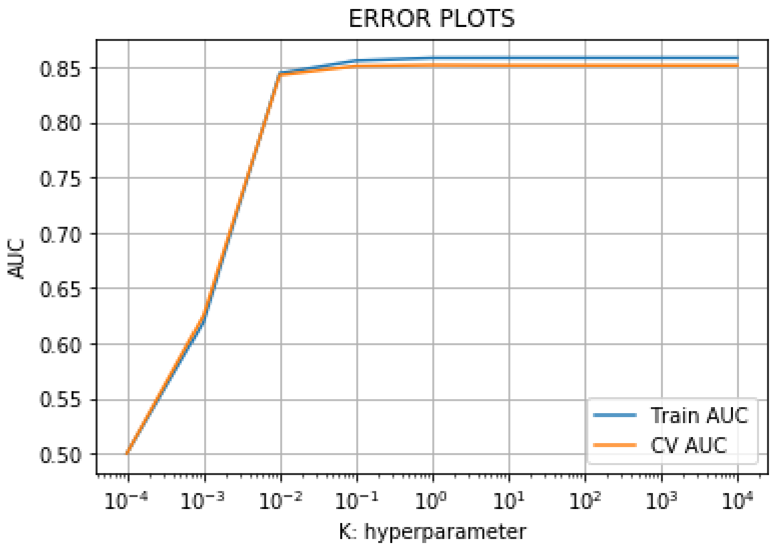
```
CV accuracy for lambda = 0.000100 is 15%

CV accuracy for lambda = 0.001000 is 43%

CV accuracy for lambda = 0.010000 is 75%

CV accuracy for lambda = 0.100000 is 76%

CV accuracy for lambda = 1.000000 is 76%

CV accuracy for lambda = 10.000000 is 76%

CV accuracy for lambda = 100.000000 is 76%

CV accuracy for lambda = 1000.000000 is 76%

CV accuracy for lambda = 10000.000000 is 76%
```

In [41]:

```
from sklearn.metrics import roc_auc_score
import matplotlib.pyplot as plt

train_auc = []
cv_auc = []
#K = [1, 5, 10, 15, 21, 31, 41, 51]
K = [10**-4, 10**-3, 10**-2, 10**-1, 1, 10**1, 10**2, 10**3, 10**4]
for i in K:
    neigh = LogisticRegression(C=i,  penalty='l1', class_weight = 'balanced')
    neigh.fit(X_train_tfw2v, y_train)

    y_train_pred =  neigh.predict_proba(X_train_tfw2v)[:,1]
    y_cv_pred =  neigh.predict_proba(X_cv_tfw2v)[:,1]

    train_auc.append(roc_auc_score(y_train,y_train_pred))
    cv_auc.append(roc_auc_score(y_cv, y_cv_pred))


plt.semilogx(K, train_auc, label='Train AUC')
plt.semilogx(K, cv_auc, label='CV AUC')
plt.legend()
plt.grid()
plt.xlabel("K: hyperparameter")
plt.ylabel("AUC")
plt.title("ERROR PLOTS")
```

Out[41]:

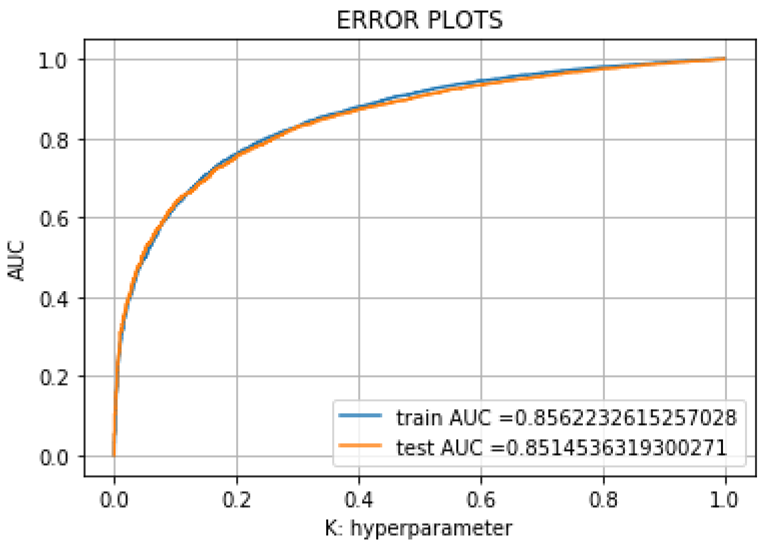
Text(0.5,1,'ERROR PLOTS')




Observation: Best lambda = 10**-1

In [42]:

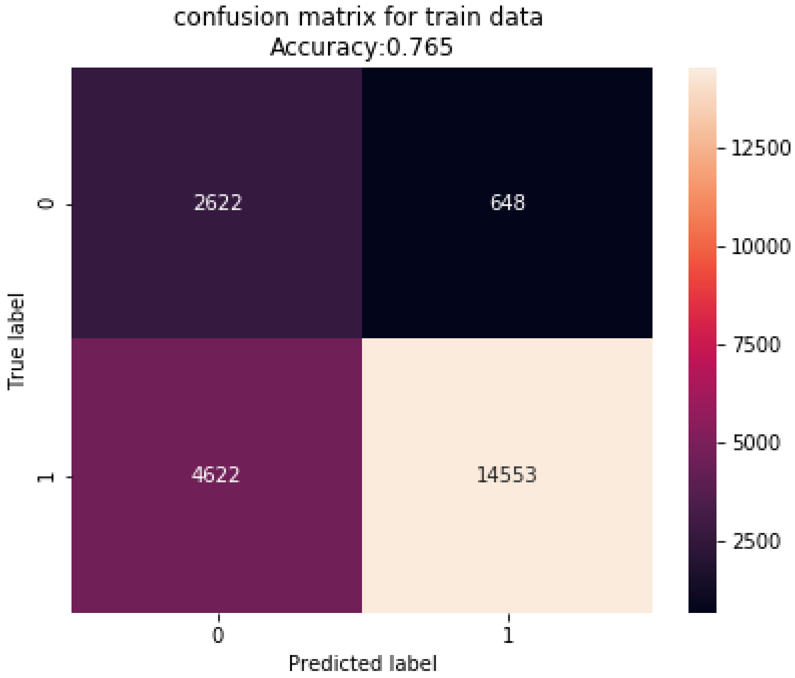
```
# https://scikit-learn.org/stable/modules/generated/sklearn.metrics.roc_curve.html#skle
arn.metrics.roc_curve
from sklearn.metrics import roc_curve, auc


neigh = LogisticRegression(C=10**-1, penalty='l1', class_weight = 'balanced')
neigh.fit(X_train_tfw2v, y_train)
# roc_auc_score(y_true, y_score) the 2nd parameter should be probability estimates of t
he positive class
# not the predicted outputs

train_fpr, train_tpr, thresholds = roc_curve(y_train, neigh.predict_proba(X_train_tfw2v
)[:,1])
test_fpr, test_tpr, thresholds = roc_curve(y_test, neigh.predict_proba(X_test_tfw2v)[:,
1])

plt.plot(train_fpr, train_tpr, label="train AUC ="+str(auc(train_fpr, train_tpr)))
plt.plot(test_fpr, test_tpr, label="test AUC ="+str(auc(test_fpr, test_tpr)))
plt.legend()
plt.grid()
plt.xlabel("K: hyperparameter")
plt.ylabel("AUC")
plt.title("ERROR PLOTS")
plt.show()

print("="*100)

from sklearn.metrics import confusion_matrix
print("Train confusion matrix")
print(confusion_matrix(y_train, neigh.predict(X_train_tfw2v)))
print("Test confusion matrix")
print(confusion_matrix(y_test, neigh.predict(X_test_tfw2v)))
```

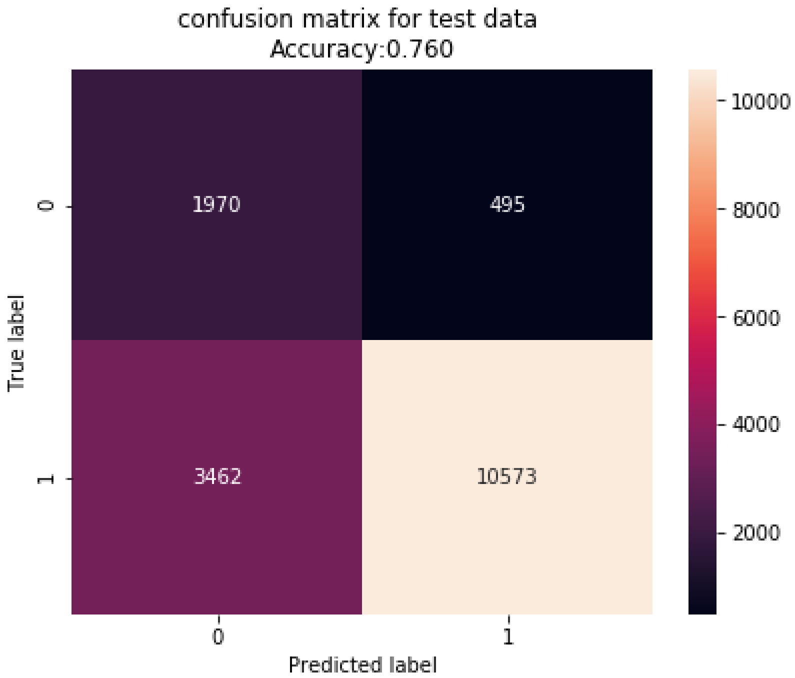



```
================================================================================
============================
Train confusion matrix
[[ 2622   648]
 [ 4622 14553]]
Test confusion matrix
[[ 1970   495]
 [ 3462 10573]]
```

In [44]:

```
# Creates a confusion matrix for train data
cm = confusion_matrix(y_train, neigh.predict(X_train_tfw2v))
cm_df = pd.DataFrame(cm)
plt.figure(figsize=(6.5,5))
sns.heatmap(cm_df, annot=True, fmt="d")
plt.title('confusion matrix for train data \nAccuracy:{0:.3f}'.format(accuracy_score(y_
train, neigh.predict(X_train_tfw2v))))
plt.ylabel('True label')
plt.xlabel('Predicted label')
plt.show()


# Creates a confusion matrix for test data
cm = confusion_matrix(y_test, neigh.predict(X_test_tfw2v))
cm_df = pd.DataFrame(cm)
plt.figure(figsize=(6.5,5))
sns.heatmap(cm_df, annot=True, fmt="d")
plt.title('confusion matrix for test data \nAccuracy:{0:.3f}'.format(accuracy_score(y_t
est, neigh.predict(X_test_tfw2v))))
plt.ylabel('True label')
plt.xlabel('Predicted label')
plt.show()
```


confusion matrix for train data
Accuracy:0.765
2622
648
4622
14553
True label
Predicted label
0
1
0
1
12500
10000
7500
5000
2500


confusion matrix for test data
Accuracy:0.760
1970
495
3462
10573
True label
Predicted label
0
1
0
1
10000
8000
6000
4000
2000

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

In [45]:

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

from sklearn.linear_model import LogisticRegression
K = [10**-4, 10**-3, 10**-2, 10**-1, 1, 10**1, 10**2, 10**3, 10**4]
#for i in range(1,50,2):
for i in K:
    clf = LogisticRegression(C=i, class_weight = 'balanced')
    clf.fit(X_train_tfw2v, y_train)

    # predict the response on the crossvalidation train
    pred = clf.predict(X_cv_tfw2v)

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

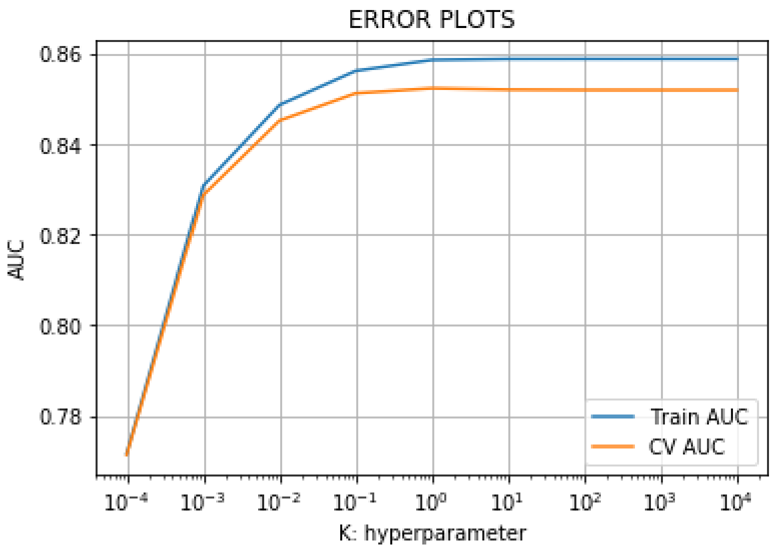
```
CV accuracy for lambda = 0.000100 is 69%

CV accuracy for lambda = 0.001000 is 73%

CV accuracy for lambda = 0.010000 is 75%

CV accuracy for lambda = 0.100000 is 76%

CV accuracy for lambda = 1.000000 is 76%

CV accuracy for lambda = 10.000000 is 76%

CV accuracy for lambda = 100.000000 is 76%

CV accuracy for lambda = 1000.000000 is 76%

CV accuracy for lambda = 10000.000000 is 76%
```

In [46]:

```
from sklearn.metrics import roc_auc_score
import matplotlib.pyplot as plt

train_auc = []
cv_auc = []
#K = [1, 5, 10, 15, 21, 31, 41, 51]
K = [10**-4, 10**-3, 10**-2, 10**-1, 1, 10**1, 10**2, 10**3, 10**4]
for i in K:
    neigh = LogisticRegression(C=i, class_weight = 'balanced')
    neigh.fit(X_train_tfw2v, y_train)

    y_train_pred =  neigh.predict_proba(X_train_tfw2v)[:,1]
    y_cv_pred =  neigh.predict_proba(X_cv_tfw2v)[:,1]

    train_auc.append(roc_auc_score(y_train,y_train_pred))
    cv_auc.append(roc_auc_score(y_cv, y_cv_pred))


plt.semilogx(K, train_auc, label='Train AUC')
plt.semilogx(K, cv_auc, label='CV AUC')
plt.legend()
plt.grid()
plt.xlabel("K: hyperparameter")
plt.ylabel("AUC")
plt.title("ERROR PLOTS")
```

Out[46]:

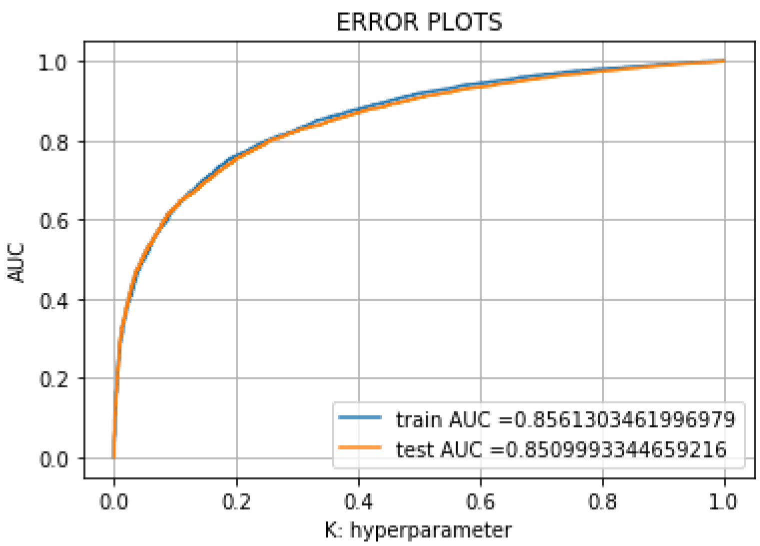
```
Text(0.5,1,'ERROR PLOTS')
```




Best hyperparameter is 10**-1

In [47]:

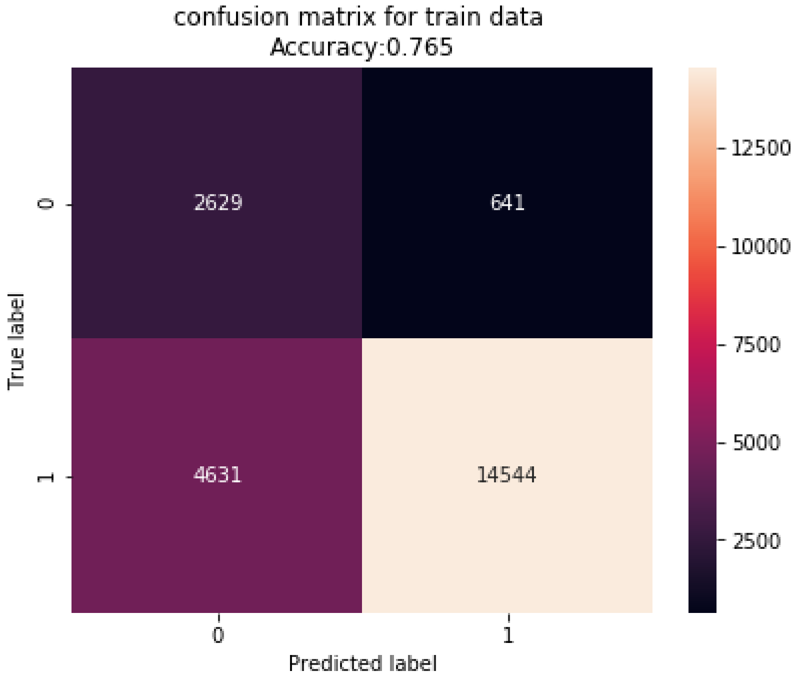
```
# https://scikit-learn.org/stable/modules/generated/sklearn.metrics.roc_curve.html#skle
arn.metrics.roc_curve
from sklearn.metrics import roc_curve, auc


neigh = LogisticRegression(C=10**-1, class_weight = 'balanced')
neigh.fit(X_train_tfw2v, y_train)
# roc_auc_score(y_true, y_score) the 2nd parameter should be probability estimates of t
he positive class
# not the predicted outputs

train_fpr, train_tpr, thresholds = roc_curve(y_train, neigh.predict_proba(X_train_tfw2v
)[:,1])
test_fpr, test_tpr, thresholds = roc_curve(y_test, neigh.predict_proba(X_test_tfw2v)[:,
1])

plt.plot(train_fpr, train_tpr, label="train AUC ="+str(auc(train_fpr, train_tpr)))
plt.plot(test_fpr, test_tpr, label="test AUC ="+str(auc(test_fpr, test_tpr)))
plt.legend()
plt.grid()
plt.xlabel("K: hyperparameter")
plt.ylabel("AUC")
plt.title("ERROR PLOTS")
plt.show()

print("="*100)

from sklearn.metrics import confusion_matrix
print("Train confusion matrix")
print(confusion_matrix(y_train, neigh.predict(X_train_tfw2v)))
print("Test confusion matrix")
print(confusion_matrix(y_test, neigh.predict(X_test_tfw2v)))
```

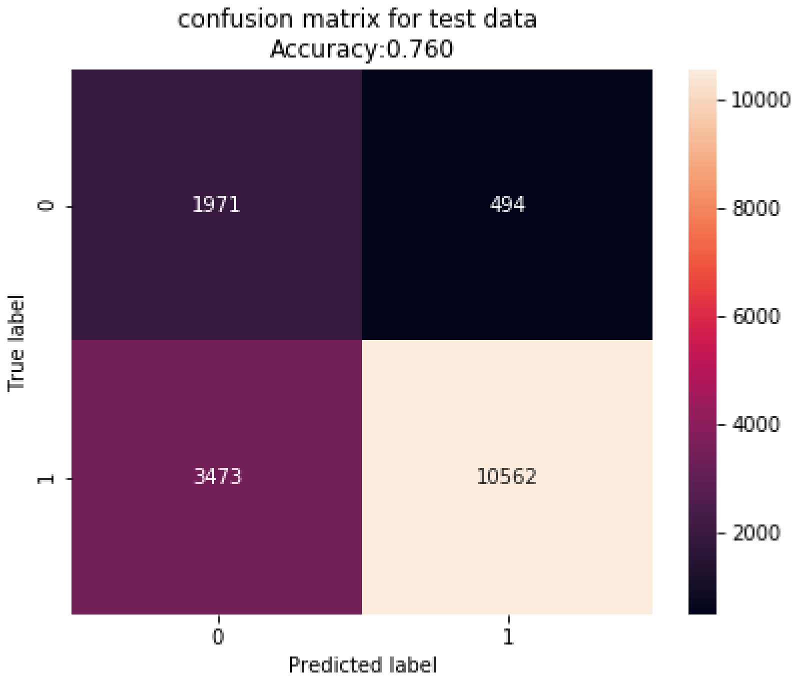



```
==========================================================================
==========================
Train confusion matrix
[[ 2629   641]
 [ 4631 14544]]
Test confusion matrix
[[ 1971   494]
 [ 3473 10562]]
```

In [48]:

```
# Creates a confusion matrix for train data
cm = confusion_matrix(y_train, neigh.predict(X_train_tfw2v))
cm_df = pd.DataFrame(cm)
plt.figure(figsize=(6.5,5))
sns.heatmap(cm_df, annot=True, fmt="d")
plt.title('confusion matrix for train data \nAccuracy:{0:.3f}'.format(accuracy_score(y_
train, neigh.predict(X_train_tfw2v))))
plt.ylabel('True label')
plt.xlabel('Predicted label')
plt.show()


# Creates a confusion matrix for test data
cm = confusion_matrix(y_test, neigh.predict(X_test_tfw2v))
cm_df = pd.DataFrame(cm)
plt.figure(figsize=(6.5,5))
sns.heatmap(cm_df, annot=True, fmt="d")
plt.title('confusion matrix for test data \nAccuracy:{0:.3f}'.format(accuracy_score(y_t
est, neigh.predict(X_test_tfw2v))))
plt.ylabel('True label')
plt.xlabel('Predicted label')
plt.show()
```


confusion matrix for train data
Accuracy:0.765
2629
641
4631
14544
True label
0
1
Predicted label
0
1
12500
10000
7500
5000
2500


confusion matrix for test data
Accuracy:0.760
1971
494
3473
10562
True label
0
1
Predicted label
0
1
10000
8000
6000
4000
2000

# [6] Conclusions

In [49]:

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

from prettytable import PrettyTable

x = PrettyTable()

x.field_names = ["Model", "Hyperparameter(alpha)", "Train AUC", "Test AUC"]

x.add_row(["BOW (L1)", 10**-1, 0.94, 0.93])
x.add_row(["BOW (L2)", 10**-2, 0.94, 0.93])
x.add_row(["TF-IDF (L1)", 1, 0.96, 0.95])
x.add_row(["TF-IDF (L2)", 1, 0.98, 0.95])
x.add_row(["AVG W2V (L1)", 10**-1, 0.89, 0.88])
x.add_row(["AVG W2V (L2)", 10**-2, 0.89, 0.88])
x.add_row(["TFIDF W2V (L1)", 10**-1, 0.856, 0.851])
x.add_row(["TFIDF W2V (L2)", 10**-1, 0.856, 0.850])


print(x)
```

```
+----------------+-----------------------+-----------+----------+
|     Model      | Hyperparameter(alpha) | Train AUC | Test AUC |
+----------------+-----------------------+-----------+----------+
|    BOW (L1)    |          0.1          |    0.94   |   0.93   |
|    BOW (L2)    |          0.01         |    0.94   |   0.93   |
|  TF-IDF (L1)   |           1           |    0.96   |   0.95   |
|  TF-IDF (L2)   |           1           |    0.98   |   0.95   |
|  AVG W2V (L1)  |          0.1          |    0.89   |   0.88   |
|  AVG W2V (L2)  |          0.01         |    0.89   |   0.88   |
| TFIDF W2V (L1) |          0.1          |   0.856   |  0.851   |
| TFIDF W2V (L2) |          0.1          |   0.856   |   0.85   |
+----------------+-----------------------+-----------+----------+
```