Amazon Fine Food Reviews Analysis

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

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

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

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

Attribute Information:

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

Objective:

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

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

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

[1]. Reading Data

[1.1] Loading the data

The dataset is available in two forms

1. .csv file
2. SQLite Database

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

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

In [1]:

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


import sqlite3
import pandas as pd
import numpy as np
import nltk
import string
import matplotlib.pyplot as plt
```

```python
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 sklearn.metrics import roc_auc_score
from tqdm import tqdm
import os
from sklearn.model_selection import train_test_split
```

```
C:\ProgramData\Anaconda3\lib\site-packages\gensim\utils.py:1209: UserWarning: detected Windows; al
iasing chunkize to chunkize_serial
  warnings.warn("detected Windows; aliasing chunkize to chunkize_serial")
```

In [2]:

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

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

# filtered_data = pd.read_sql_query(""" SELECT * FROM Reviews WHERE Score != 3 LIMIT 500000""", co
n)
# for tsne assignment you can take 5k data points

filtered_data = pd.read_sql_query(""" SELECT * FROM Reviews WHERE Score != 3 LIMIT 50000""", 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 (50000, 10)
```

Out[2]:

	Id	ProductId	UserId	ProfileName	HelpfulnessNumerator	HelpfulnessDenominator	Score	Time
0	1	B001E4KFG0	A3SGXH7AUHU8GW	delmartian	1	1	1	130386240(

	Id	ProductId	UserId	ProfileName	HelpfulnessNumerator	HelpfulnessDenominator	Score	Time
1	2	B00813GRG4	A1D87F6ZCVE5NK	dll pa	0	0	0	134697600
2	3	B000LQOCH0	ABXLMWJIXXAIN	Natalia Corres "Natalia Corres"	1	1	1	121901760

In [3]:

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

In [4]:

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

```
(80668, 7)
```

Out[4]:

	UserId	ProductId	ProfileName	Time	Score	Text	COUNT(*)
0	#oc-R115TNMSPFT9I7	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	Text	COUNT(*)
80638	AZY10LLTJ71NX	B006P7E5ZI	undertheshrine "undertheshrine"	1334707200	5	I was recommended to try green tea extract to ...	5

In [6]:

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

Out[6]:

```
393063
```

[2] Exploratory Data Analysis

[2.1] Data Cleaning: Deduplication

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

In [7]:

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

Out[7]:

	Id	ProductId	UserId	ProfileName	HelpfulnessNumerator	HelpfulnessDenominator	Score	Ti
0	78445	B000HDL1RQ	AR5J8UI46CURR	Geetha Krishnan	2	2	5	1199577€
1	138317	B000HDOPYC	AR5J8UI46CURR	Geetha Krishnan	2	2	5	1199577€
2	138277	B000HDOPYM	AR5J8UI46CURR	Geetha Krishnan	2	2	5	1199577€
3	73791	B000HDOPZG	AR5J8UI46CURR	Geetha Krishnan	2	2	5	1199577€
4	155049	B000PAQ75C	AR5J8UI46CURR	Geetha Krishnan	2	2	5	1199577€

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='qui
cksort', na_position='last')
```

In [9]:

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

Out[9]:

```
(46072, 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]:

```
92.144
```

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

In [11]:

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

display.head()
```

Out[11]:

	Id	ProductId	UserId	ProfileName	HelpfulnessNumerator	HelpfulnessDenominator	Score	Ti
0	64422	B000MIDROQ	A161DK06JJMCYF	J. E. Stephens "Jeanne"	3	1	5	1224892$
1	44737	B001EQ55RW	A2V0I904FH7ABY	Ram	3	2	4	12128832

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()
```

```
(46071, 10)
```

Out[13]:

```
1    38479
0     7592
Name: Score, dtype: int64
```

[3] Preprocessing

[3.1]. Preprocessing Review Text

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

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

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

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

In [14]:

```
# printing some random reviews
sent_0 = final['Text'].values[0]
print(sent_0)
print("="*50)

sent_1000 = final['Text'].values[1000]
print(sent_1000)
print("="*50)

sent_1500 = final['Text'].values[1500]
print(sent_1500)
print("="*50)

sent_4900 = final['Text'].values[4900]
print(sent_4900)
print("="*50)
```

```
My dogs loves this chicken but its a product from China, so we wont be buying it anymore.  Its ver
y hard to find any chicken products made in the USA but they are out there, but this one isnt.  It
s too bad too because its a good product but I wont take any chances till they know what is going
on with the china imports.
==================================================
this is yummy, easy and unusual. it makes a quick, delicous pie, crisp or cobbler. home made is be
tter, but a heck of a lot more work. this is great to have on hand for last minute dessert needs w
here you really want to impress wih your creativity in cooking! recommended.
==================================================
Great flavor, low in calories, high in nutrients, high in protein! Usually protein powders are hig
h priced and high in calories, this one is a great bargain and tastes great, I highly recommend fo
r the lady gym rats, probably not "macho" enough for guys since it is soy based...
==================================================
For those of you wanting a high-quality, yet affordable green tea, you should definitely give this
one a try. Let me first start by saying that everyone is looking for something different for their
ideal tea, and I will attempt to briefly highlight what makes this tea attractive to a wide range
of tea drinkers (whether you are a beginner or long-time tea enthusiast).  I have gone through ove
r 12 boxes of this tea myself, and highly recommend it for the following reasons:<br /><br />-Qual
ity:  First, this tea offers a smooth quality without any harsh or bitter after tones, which often
turns people off from many green teas.  I've found my ideal brewing time to be between 3-5
minutes, giving you a light but flavorful cup of tea.  However, if you get distracted or forget ab
out your tea and leave it brewing for 20+ minutes like I sometimes do, the quality of this tea is
```

```
such that you still get a smooth but deeper flavor without the bad after taste.  The leaves themse
lves are whole leaves (not powdered stems, branches, etc commonly found in other brands), and the
high-quality nylon bags also include chunks of tropical fruit and other discernible ingredients.
This isn't your standard cheap paper bag with a mix of unknown ingredients that have been ground d
own to a fine powder, leaving you to wonder what it is you are actually drinking.<br /><br />-Tast
e:  This tea offers notes of real pineapple and other hints of tropical fruits, yet isn't sweet or
artificially flavored.  You have the foundation of a high-quality young hyson green tea for those
true "tea flavor" lovers, yet the subtle hints of fruit make this a truly unique tea that I believ
e most will enjoy.  If you want it sweet, you can add sugar, splenda, etc but this really is not n
ecessary as this tea offers an inherent warmth of flavor through it's ingredients.<br /><br />-Pri
ce:  This tea offers an excellent product at an exceptional price (especially when purchased at th
e prices Amazon offers).  Compared to other brands which I believe to be of similar quality
(Mighty Leaf, Rishi, Two Leaves, etc.), Revolution offers a superior product at an outstanding pri
ce.  I have been purchasing this through Amazon for less per box than I would be paying at my loca
l grocery store for Lipton, etc.<br /><br />Overall, this is a wonderful tea that is comparable, a
nd even better than, other teas that are priced much higher.  It offers a well-balanced cup of gre
en tea that I believe many will enjoy.  In terms of taste, quality, and price, I would argue you w
on't find a better combination that that offered by Revolution's Tropical Green Tea.
==================================================
```

In [15]:

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

print(sent_0)
```

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

In [16]:

```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)
```

```
My dogs loves this chicken but its a product from China, so we wont be buying it anymore.  Its ver
y hard to find any chicken products made in the USA but they are out there, but this one isnt.  It
s too bad too because its a good product but I wont take any chances till they know what is going
on with the china imports.
==================================================
this is yummy, easy and unusual. it makes a quick, delicous pie, crisp or cobbler. home made is be
tter, but a heck of a lot more work. this is great to have on hand for last minute dessert needs w
here you really want to impress wih your creativity in cooking! recommended.
==================================================
Great flavor, low in calories, high in nutrients, high in protein! Usually protein powders are hig
h priced and high in calories, this one is a great bargain and tastes great, I highly recommend fo
r the lady gym rats, probably not "macho" enough for guys since it is soy based...
==================================================
For those of you wanting a high-quality, yet affordable green tea, you should definitely give this
```

one a try. Let me first start by saying that everyone is looking for something different for their ideal tea, and I will attempt to briefly highlight what makes this tea attractive to a wide range of tea drinkers (whether you are a beginner or long-time tea enthusiast). I have gone through ove r 12 boxes of this tea myself, and highly recommend it for the following reasons:-Quality: First, this tea offers a smooth quality without any harsh or bitter after tones, which often turns people off from many green teas. I've found my ideal brewing time to be between 3-5 minutes, giving you a light but flavorful cup of tea. However, if you get distracted or forget about your tea and lea ve it brewing for 20+ minutes like I sometimes do, the quality of this tea is such that you still get a smooth but deeper flavor without the bad after taste. The leaves themselves are whole leave s (not powdered stems, branches, etc commonly found in other brands), and the high-quality nylon b ags also include chunks of tropical fruit and other discernible ingredients. This isn't your standard cheap paper bag with a mix of unknown ingredients that have been ground down to a fine po wder, leaving you to wonder what it is you are actually drinking.-Taste: This tea offers notes of real pineapple and other hints of tropical fruits, yet isn't sweet or artificially flavored. You have the foundation of a high-quality young hyson green tea for those true "tea flavor" lovers, ye t the subtle hints of fruit make this a truly unique tea that I believe most will enjoy. If you w ant it sweet, you can add sugar, splenda, etc but this really is not necessary as this tea offers an inherent warmth of flavor through it's ingredients.-Price: This tea offers an excellent product at an exceptional price (especially when purchased at the prices Amazon offers). Compared to other brands which I believe to be of similar quality (Mighty Leaf, Rishi, Two Leaves, etc.), R evolution offers a superior product at an outstanding price. I have been purchasing this through Amazon for less per box than I would be paying at my local grocery store for Lipton, etc.Overall, this is a wonderful tea that is comparable, and even better than, other teas that are priced much higher. It offers a well-balanced cup of green tea that I believe many will enjoy. In terms of t aste, quality, and price, I would argue you won't find a better combination that that offered by R evolution's Tropical Green Tea.

In [17]:

```
# https://stackoverflow.com/a/47091490/4084039
import re

def decontracted(phrase):
    # specific
    phrase = re.sub(r"won't", "will not", phrase)
    phrase = re.sub(r"can\'t", "can not", phrase)

    # general
    phrase = re.sub(r"n\'t", " not", phrase)
    phrase = re.sub(r"\'re", " are", phrase)
    phrase = re.sub(r"\'s", " is", phrase)
    phrase = re.sub(r"\'d", " would", phrase)
    phrase = re.sub(r"\'ll", " will", phrase)
    phrase = re.sub(r"\'t", " not", phrase)
    phrase = re.sub(r"\'ve", " have", phrase)
    phrase = re.sub(r"\'m", " am", phrase)
    return phrase
```

In [18]:

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

Great flavor, low in calories, high in nutrients, high in protein! Usually protein powders are hig h priced and high in calories, this one is a great bargain and tastes great, I highly recommend fo r the lady gym rats, probably not "macho" enough for guys since it is soy based...
==

In [19]:

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

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

In [20]:

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

```
Great flavor low in calories high in nutrients high in protein Usually protein powders are high pr
iced and high in calories this one is a great bargain and tastes great I highly recommend for the
lady gym rats probably not macho enough for guys since it is soy based
```

In [21]:

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

stopwords= set(['br', 'the', 'i', 'me', 'my', 'myself', 'we', 'our', 'ours', 'ourselves', 'you', "y
ou'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', 'they', 'them',
'their',\
            'theirs', 'themselves', 'what', 'which', 'who', 'whom', 'this', 'that', "that'll",
'these', 'those', \
            'am', 'is', 'are', 'was', 'were', 'be', 'been', 'being', 'have', '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', 'how', 'all', 'any', 'both', 'e
ach', 'few', 'more',\
            'most', 'other', 'some', 'such', 'only', 'own', 'same', 'so', 'than', 'too', 'very', \
            's', 't', 'can', 'will', 'just', 'don', "don't", 'should', "should've", 'now', 'd', 'll'
, 'm', 'o', 're', \
            've', 'y', 'ain', 'aren', "aren't", 'couldn', "couldn't", 'didn', "didn't", 'doesn', "do
esn't", 'hadn',\
            "hadn't", 'hasn', "hasn't", 'haven', "haven't", 'isn', "isn't", 'ma', 'mightn',
"mightn't", 'mustn',\
            "mustn't", 'needn', "needn't", 'shan', "shan't", 'shouldn', "shouldn't", 'wasn',
"wasn't", 'weren', "weren't", \
            'won', "won't", 'wouldn', "wouldn't"])
```

In [22]:

```
# Combining all the above stundents
from tqdm import tqdm
preprocessed_reviews = []
# tqdm is for printing the status bar
for sentance in tqdm(final['Text'].values):
    sentance = re.sub(r"http\S+", "", sentance)
    sentance = BeautifulSoup(sentance, 'lxml').get_text()
    sentance = decontracted(sentance)
    sentance = re.sub("\S*\d\S*", "", sentance).strip()
    sentance = re.sub('[^A-Za-z]+', ' ', sentance)
    # https://gist.github.com/sebleier/554280
    sentance = ' '.join(e.lower() for e in sentance.split() if e.lower() not in stopwords)
    preprocessed_reviews.append(sentance.strip())
```

```
100%|████████████████████████████████████████████████████████████████████████████████| 46071/46071
[00:20<00:00, 2301.95it/s]
```

In [23]:

```
preprocessed_reviews[1500]
```

Out[23]:

```
'great flavor low calories high nutrients high protein usually protein powders high priced high ca
lories one great bargain tastes great highly recommend lady gym rats probably not macho enough guy
s since soy based'
```

[3.2] Preprocessing Review Summary

Feature Engineering

- adding the no. of words in each review as a feature.

In [58]:

```
preprocessed_reviews_fe = []
for i in preprocessed_reviews:
    count = 0;
    for j in i:
        count += 1;
    i = i + " " + str(count);
    preprocessed_reviews_fe.append(i)
```

In [26]:

```
preprocessed_reviews_fe[1]
```

Out[26]:

```
'dogs love saw pet store tag attached regarding made china satisfied safe 72'
```

[4] Featurization

In [27]:

```
from sklearn.model_selection import train_test_split

X_train, X_test, y_train, y_test = train_test_split(preprocessed_reviews_fe, final['Score'], test_size=0.33) # this is random splitting
X_train, X_cv, y_train, y_cv = train_test_split(X_train, y_train, test_size=0.33) # this is random splitting
```

[4.1] BAG OF WORDS

In [33]:

```
#BoW
count_vect = CountVectorizer(min_df=10, max_features=500) #in scikit-learn

X_train_vect = count_vect.fit_transform(X_train)
X_train_vect = X_train_vect.toarray()
X_cv_vect = count_vect.transform(X_cv)
X_cv_vect = X_cv_vect.toarray()
X_test_vect = count_vect.transform(X_test)
X_test_vect = X_test_vect.toarray()
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  ['able', 'absolutely', 'acid', 'actually', 'add', 'added', 'adding',
'aftertaste', 'ago', 'almonds']
==================================================
the type of count vectorizer  <class 'scipy.sparse.csr.csr_matrix'>
the shape of out text BOW vectorizer  (46071, 500)
the number of unique words  500
```

[4.2] Bi-Grams and n-Grams.

In [ ]:

```
#bi-gram, tri-gram and n-gram

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

# you can choose these numebrs min_df=10, max_features=5000, of your choice
count_vect = CountVectorizer(ngram_range=(1,2), min_df=10, max_features=5000)
final_bigram_counts = count_vect.fit_transform(preprocessed_reviews)
print("the type of count vectorizer ",type(final_bigram_counts))
print("the shape of out text BOW vectorizer ",final_bigram_counts.get_shape())
print("the number of unique words including both unigrams and bigrams ", final_bigram_counts.get_s
hape()[1])
```

[4.3] TF-IDF

In [34]:

```
#tf_idf_vect.fit(preprocessed_reviews)
tf_idf = TfidfVectorizer(min_df=10, max_features=500)
X_train_vect_tfidf = tf_idf.fit_transform(X_train)
X_train_vect_tfidf = X_train_vect_tfidf.toarray()
X_test_vect_tfidf = tf_idf.transform(X_test)
X_test_vect_tfidf = X_test_vect_tfidf.toarray()
X_cv_vect_tfidf = tf_idf.transform(X_cv)
X_cv_vect_tfidf = X_cv_vect_tfidf.toarray()
```

[4.4] Word2Vec

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

[4.4.1.1] Avg W2v

In [30]:

```
# average Word2Vec
# compute average word2vec for each review.
i=0
list_of_sent=[]
X_train_list=[]
X_test_list=[]
X_cv_list=[]
for sent in X_train:
    X_train_list.append(sent.split())

for sent in X_cv:
    X_cv_list.append(sent.split())

for sent in X_test:
    X_test_list.append(sent.split())

w2v_model=Word2Vec(X_train_list,min_count=0,size=50, workers=4)
w2v_words = list(w2v_model.wv.vocab)

X_train_vectors = [];
for sent in tqdm(X_train_list):
    sent_vec = np.zeros(50)
    cnt_words =0;
    for word in sent:
        if word in w2v_words:
            vec = w2v_model.wv[word]
            sent_vec += vec
            cnt_words += 1
```

```
            cnt_words += 1
    if cnt_words != 0:
        sent_vec /= cnt_words
    X_train_vectors.append(sent_vec)


X_cv_vectors = []
for sent in tqdm(X_cv_list):
    sent_vec = np.zeros(50)
    cnt_words =0;
    for word in sent:
        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_vectors.append(sent_vec)

X_test_vectors = []
for sent in tqdm(X_test_list):
    sent_vec = np.zeros(50)
    cnt_words =0;
    for word in sent:
        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_vectors.append(sent_vec)
```

```
100%|████████████████████████████████████████████████████████████████████████████| 20680/20680 [00:51<00:00, 398.90it/s]
100%|████████████████████████████████████████████████████████████████████████████| 10187/10187 [00:29<00:00, 345.32it/s]
100%|████████████████████████████████████████████████████████████████████████████| 15204/15204 [00:43<00:00, 345.89it/s]
```

[4.4.1.2] TFIDF weighted W2v

In [31]:

```
dictionary = dict(zip(tf_idf.get_feature_names(), list(tf_idf.idf_)))
tfidf_feat = tf_idf.get_feature_names()
tfidf_X_train_vectors = [];
tfidf_X_test_vectors = [];
tfidf_X_cv_vectors = [];

for sent in tqdm(X_train_list):
    sent_vec = np.zeros(50)
    weight_sum =0;
    for word in sent:
        if word in (w2v_words and tfidf_feat):
            vec = w2v_model.wv[word]
            tf_idf_count = dictionary[word]*sent.count(word)
            sent_vec += (vec * tf_idf_count)
            weight_sum += tf_idf_count
    if weight_sum != 0:
        sent_vec /= weight_sum
    tfidf_X_train_vectors.append(sent_vec)

for sent in tqdm(X_test_list):
    sent_vec = np.zeros(50)
    weight_sum =0;
    for word in sent:
        if word in (w2v_words and tfidf_feat):
            vec = w2v_model.wv[word]
            tf_idf_count = dictionary[word]*sent.count(word)
            sent_vec += (vec * tf_idf_count)
            weight_sum += tf_idf_count
    if weight_sum != 0:
        sent_vec /= weight_sum
    tfidf_X_test_vectors.append(sent_vec)
```

```
for sent in tqdm(X_cv_list):
    sent_vec = np.zeros(50)
    weight_sum =0;
    for word in sent:
        if word in (w2v_words and tfidf_feat):
            vec = w2v_model.wv[word]
            tf_idf_count = dictionary[word]*sent.count(word)
            sent_vec += (vec * tf_idf_count)
            weight_sum += tf_idf_count
    if weight_sum != 0:
        sent_vec /= weight_sum
    tfidf_X_cv_vectors.append(sent_vec)
```

```
100%|████████████████████████████████████████████████████████████████████████████| 20680/20680 [04
:02<00:00, 85.26it/s]
100%|████████████████████████████████████████████████████████████████████████████| 15204/15204 [03
:00<00:00, 84.39it/s]
100%|████████████████████████████████████████████████████████████████████████████| 10187/10187 [01
:59<00:00, 85.63it/s]
```

[5] Assignment 3: KNN

1. **Apply Knn(brute force version) 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. **Apply Knn(kd tree version) on these feature sets**
   NOTE: sklearn implementation of kd-tree accepts only dense matrices, you need to convert the sparse matrices of CountVectorizer/TfidfVectorizer into dense matices. You can convert sparse matrices to dense using .toarray() attribute. For more information please visit this link
   - SET 5:Review text, preprocessed one converted into vectors using (BOW) but with restriction on maximum features generated.

     ```
     count_vect = CountVectorizer(min_df=10, max_features=500)
     count_vect.fit(preprocessed_reviews)
     ```

   - SET 6:Review text, preprocessed one converted into vectors using (TFIDF) but with restriction on maximum features generated.

     ```
     tf_idf_vect = TfidfVectorizer(min_df=10, max_features=500)
     tf_idf_vect.fit(preprocessed_reviews)
     ```

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

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

4. **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

5. **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.](#)

[5.1] Applying KNN brute force

[5.1.1] Applying KNN brute force on BOW, SET 1

In [35]:

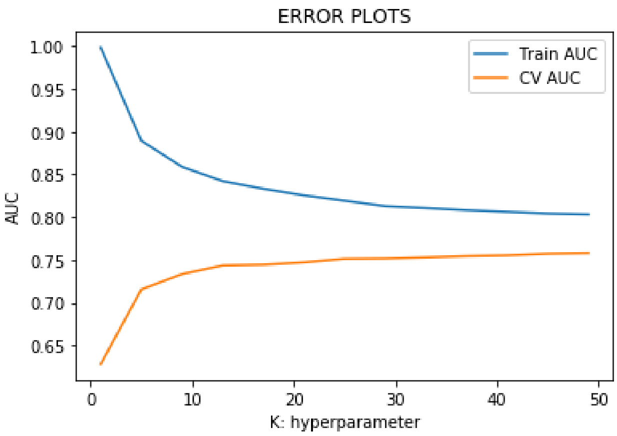
```
# Please write all the code with proper documentation
from sklearn.neighbors import KNeighborsClassifier
from sklearn.metrics import accuracy_score
score = []
ind = []
train_auc = []
cv_auc = []
print(len(y_cv))
for i in tqdm(range(1,50,4)):
    knn = KNeighborsClassifier(n_neighbors=i,algorithm = 'brute')
    knn.fit(X_train_vect, y_train)
    pred = knn.predict(X_cv_vect)
    acc = accuracy_score(y_cv, pred, normalize=True) * float(100)
    score.append(acc)
    ind.append(i)
    y_train_pred =  knn.predict_proba(X_train_vect)[:,1]
    y_cv_pred =  knn.predict_proba(X_cv_vect)[:,1]
    train_auc.append(roc_auc_score(y_train,y_train_pred))
    cv_auc.append(roc_auc_score(y_cv, y_cv_pred))
optimal_k_accuracy = ind[score.index(max(score))]
print('\nThe optimal number of neighbors is (according to accuracy): %d.' % optimal_k_accuracy)
optimal_k_auc = ind[cv_auc.index(max(cv_auc))]
print('\nThe optimal number of neighbors is (according to auc curve (max auc)): %d.' %
optimal_k_auc)

plt.plot(range(1,50,4), train_auc, label='Train AUC')
plt.plot(range(1,50,4), cv_auc, label='CV AUC')
plt.legend()
plt.xlabel("K: hyperparameter")
plt.ylabel("AUC")
plt.title("ERROR PLOTS")
plt.show()

plt.scatter(range(1,50,4), train_auc, label='Train AUC')
plt.scatter(range(1,50,4), cv_auc, label='CV AUC')
plt.legend()
plt.xlabel("K: hyperparameter")
plt.ylabel("AUC")
plt.title("ERROR PLOTS")
plt.show()
```

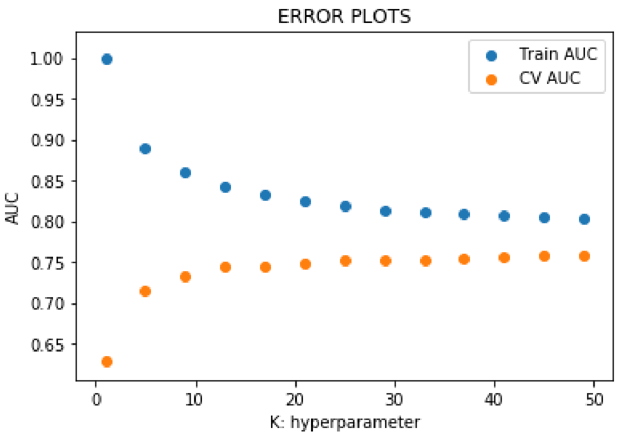
```
10187
```

```
  0%|                                                                        | 0
[00:00<?, ?it/s]
  8%|██████                                                        | 1/13 [00:
05:27, 27.28s/it]
 15%|█████████████                                                 | 2/13
```

```
[01:01<05:22, 29.29s/it]
 23%|██████                                    | 3/13 [01:
<04:58, 29.81s/it]
 31%|████████                                  | 4/13 [02:
<04:32, 30.28s/it]
 38%|██████████                                | 5/13
[02:34<04:03, 30.44s/it]
 46%|████████████                              | 6/13
[03:05<03:34, 30.66s/it]
 54%|██████████████                            | 7/13 [03:
7<03:05, 30.89s/it]
 62%|████████████████                          | 8/13 [04:
9<02:37, 31.43s/it]
 69%|██████████████████                        | 9/13
[04:46<02:11, 32.96s/it]
 77%|████████████████████                     | 10/13 [05:
25<01:44, 34.71s/it]
 85%|██████████████████████                   | 11/13 [06:
01<01:10, 35.08s/it]
 92%|████████████████████████                 | 12/13
[06:38<00:35, 35.85s/it]
100%|██████████████████████████| 13/13
[07:14<00:00, 35.83s/it]
```

```
The optimal number of neighbors is (according to accuracy): 49.

The optimal number of neighbors is (according to auc curve (max auc)): 49.
```



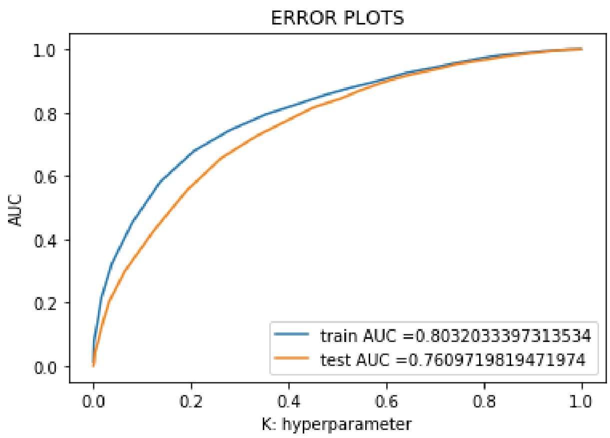


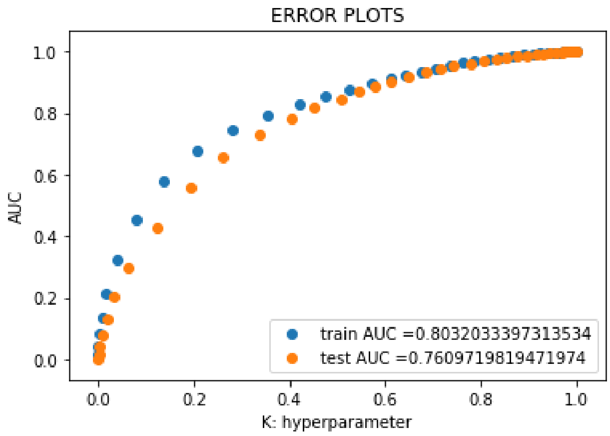
In [36]:

```
knn = KNeighborsClassifier(optimal_k_accuracy,algorithm = 'brute')
knn.fit(X_train_vect,y_train)
pred = knn.predict(X_test_vect)
acc = accuracy_score(y_test, pred, normalize=True) * float(100)
print('\n****Test accuracy for k = %d is %f%%' % (optimal_k_accuracy,acc))

train_fpr, train_tpr, thresholds = roc_curve(y_train, knn.predict_proba(X_train_vect)[:,1])
test_fpr, test_tpr, thresholds = roc_curve(y_test, knn.predict_proba(X_test_vect)[:,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.legend()
plt.xlabel("K: hyperparameter")
plt.ylabel("AUC")
plt.title("ERROR PLOTS")
plt.show()

plt.scatter(train_fpr, train_tpr, label="train AUC ="+str(auc(train_fpr, train_tpr)))
plt.scatter(test_fpr, test_tpr, label="test AUC ="+str(auc(test_fpr, test_tpr)))
plt.legend()
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")
sns.heatmap(confusion_matrix(y_train, knn.predict(X_train_vect)))
print("Test confusion matrix")
sns.heatmap(confusion_matrix(y_test, knn.predict(X_test_vect)))
```

```
****Test accuracy for k = 49 is 83.925283%
```

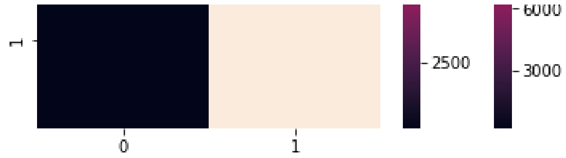





```
====================================================================================================

Train confusion matrix
Test confusion matrix
```

Out[36]:

```
<matplotlib.axes._subplots.AxesSubplot at 0x27170bed518>
```



[5.1.2] Applying KNN brute force on TFIDF, SET 2

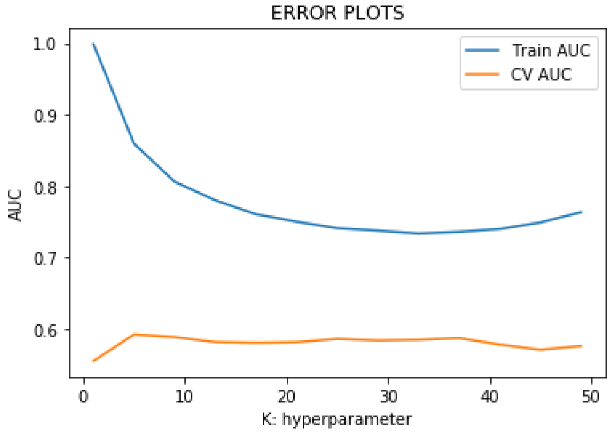
In [37]:

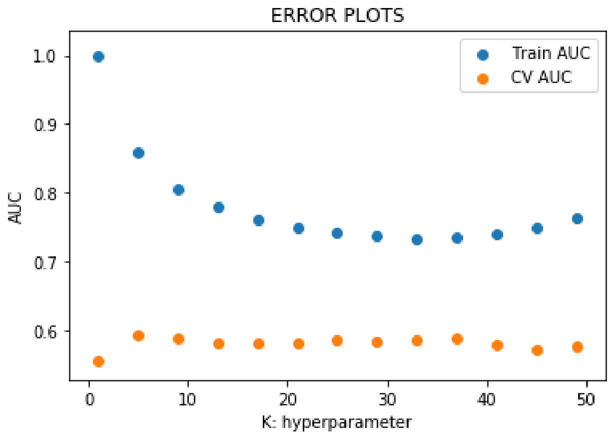
```
# Please write all the code with proper documentation
score = []
ind = []
train_auc = []
cv_auc = []
for i in tqdm(range(1,50,4)):
    knn = KNeighborsClassifier(n_neighbors=i,algorithm = 'brute')
    knn.fit(X_train_vect_tfidf, y_train)
    pred = knn.predict(X_cv_vect_tfidf)
    acc = accuracy_score(y_cv, pred, normalize=True) * float(100)
    score.append(acc)
    ind.append(i)
    y_train_pred =  knn.predict_proba(X_train_vect_tfidf)[:,1]
    y_cv_pred =  knn.predict_proba(X_cv_vect_tfidf)[:,1]
    train_auc.append(roc_auc_score(y_train,y_train_pred))
    cv_auc.append(roc_auc_score(y_cv, y_cv_pred))

optimal_k_accuracy = ind[score.index(max(score))]
print('\nThe optimal number of neighbors is (according to accuracy): %d.' % optimal_k_accuracy)
optimal_k_auc = ind[cv_auc.index(max(cv_auc))]
print('\nThe optimal number of neighbors is (according to auc curve (max auc)): %d.' %
optimal_k_auc)


plt.plot(range(1,50,4), train_auc, label='Train AUC')
plt.plot(range(1,50,4), cv_auc, label='CV AUC')
plt.legend()
plt.xlabel("K: hyperparameter")
plt.ylabel("AUC")
plt.title("ERROR PLOTS")
plt.show()

plt.scatter(range(1,50,4), train_auc, label='Train AUC')
plt.scatter(range(1,50,4), cv_auc, label='CV AUC')
plt.legend()
plt.xlabel("K: hyperparameter")
plt.ylabel("AUC")
plt.title("ERROR PLOTS")
plt.show()
```

```
  0%|                                                                     | C
[00:00<?, ?it/s]
  8%|███▌                                                           | 1/13 [00:
05:02, 25.23s/it]
 15%|███████                                                        | 2/13
[00:56<04:57, 27.06s/it]
 23%|██████████▌                                                    | 3/13 [01:
<04:51, 29.18s/it]
 31%|██████████████                                                 | 4/13 [02:
<04:30, 30.09s/it]
 38%|█████████████████▌                                             | 5/13
[02:33<04:02, 30.35s/it]
 46%|█████████████████████                                          | 6/13
[03:06<03:36, 30.99s/it]
 54%|████████████████████████▌                                      | 7/13 [03:
6<03:05, 30.86s/it]
 62%|████████████████████████████                                   | 8/13 [04:
9<02:37, 31.47s/it]
 69%|███████████████████████████████▌                               | 9/13
[04:43<02:08, 32.23s/it]
 77%|███████████████████████████████████                           | 10/13 [05:
15<01:36, 32.09s/it]
 85%|██████████████████████████████████████▌                       | 11/13 [05:
46<01:03, 31.72s/it]
```

```
 92%|█████████████████████████████████████████████████████████████         | 12/13
[06:16<00:31, 31.27s/it]
100%|██████████████████████████████████████████████████████████████████| 13/13
[06:46<00:00, 30.99s/it]
```

```
The optimal number of neighbors is (according to accuracy): 17.

The optimal number of neighbors is (according to auc curve (max auc)): 5.
```



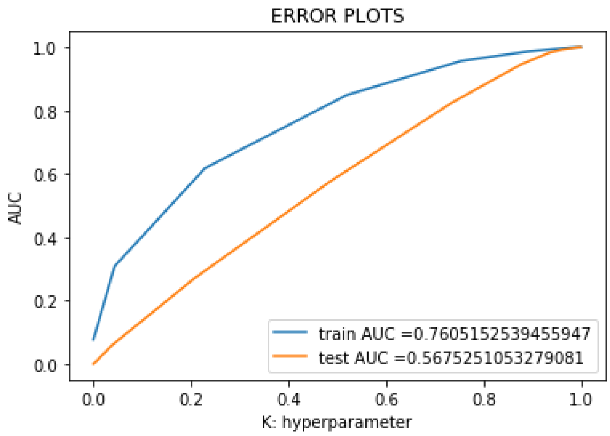


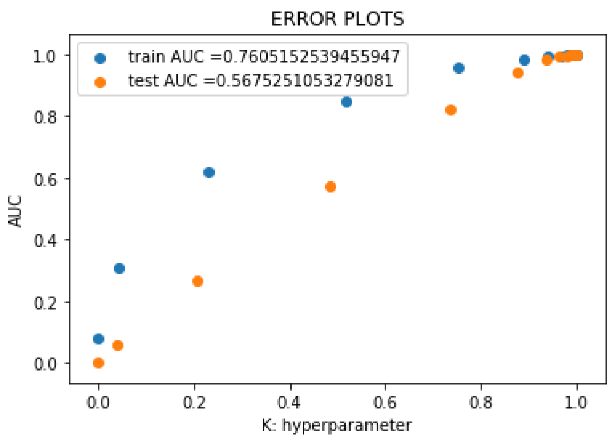
In [39]:

```
knn = KNeighborsClassifier(optimal_k_accuracy,algorithm = 'brute')
knn.fit(X_train_vect_tfidf,y_train)
pred = knn.predict(X_test_vect_tfidf)
acc = accuracy_score(y_test, pred, normalize=True) * float(100)
print('\n****Test accuracy for k = %d is %f%%' % (optimal_k_accuracy,acc))

train_fpr, train_tpr, thresholds = roc_curve(y_train, knn.predict_proba(X_train_vect_tfidf)[:,1])
test_fpr, test_tpr, thresholds = roc_curve(y_test, knn.predict_proba(X_test_vect_tfidf)[:,1])

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

plt.scatter(train_fpr, train_tpr, label="train AUC ="+str(auc(train_fpr, train_tpr)))
plt.scatter(test_fpr, test_tpr, label="test AUC ="+str(auc(test_fpr, test_tpr)))
plt.legend()
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")
sns.heatmap(confusion_matrix(y_train, knn.predict(X_train_vect_tfidf)))
print("Test confusion matrix")
```

```
print('Test confusion matrix')
sns.heatmap(confusion_matrix(y_test, knn.predict(X_test_vect_tfidf)))
```

```
****Test accuracy for k = 17 is 83.300447%
```

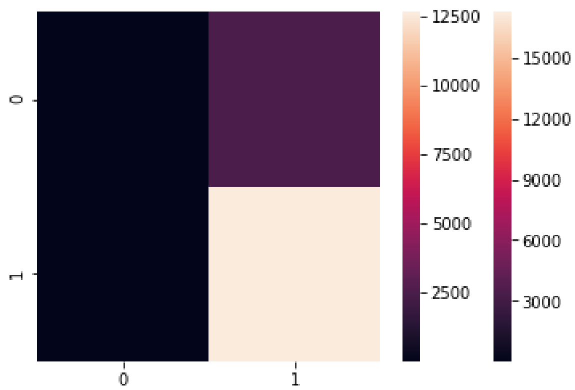





```
====================================================================================================

Train confusion matrix
Test confusion matrix
```

Out[39]:

```
<matplotlib.axes._subplots.AxesSubplot at 0x2716fb6e160>
```



[5.1.3] Applying KNN brute force on AVG W2V, SET 3

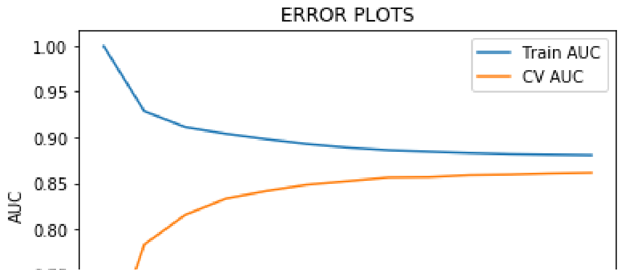
In [40]:

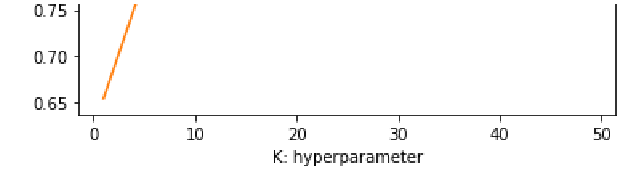
```
# Please write all the code with proper documentation
score = []
ind = []
train_auc = []
cv_auc = []
for i in tqdm(range(1,50,4)):
    knn = KNeighborsClassifier(n_neighbors=i,algorithm = 'brute')
```

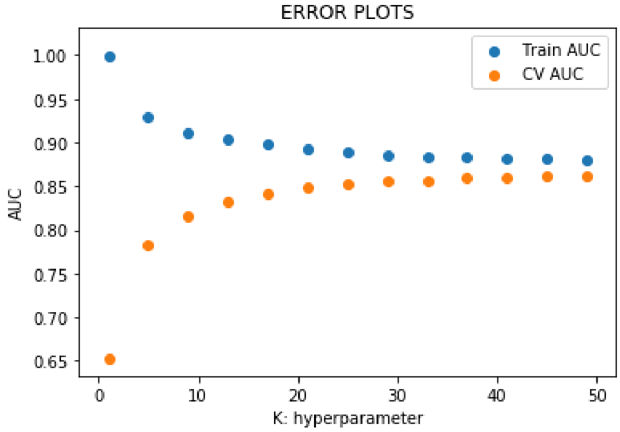
```
    knn.fit(X_train_vectors, y_train)
    pred = knn.predict(X_cv_vectors)
    acc = accuracy_score(y_cv, pred, normalize=True) * float(100)
    score.append(acc)
    ind.append(i)
    y_train_pred =  knn.predict_proba(X_train_vectors)[:,1]
    y_cv_pred =  knn.predict_proba(X_cv_vectors)[:,1]
    train_auc.append(roc_auc_score(y_train,y_train_pred))
    cv_auc.append(roc_auc_score(y_cv, y_cv_pred))
optimal_k_accuracy = ind[score.index(max(score))]
print('\nThe optimal number of neighbors is (according to accuracy): %d.' % optimal_k_accuracy)
optimal_k_auc = ind[cv_auc.index(max(cv_auc))]
print('\nThe optimal number of neighbors is (according to auc curve (max auc)): %d.' %
optimal_k_auc)


plt.plot(range(1,50,4), train_auc, label='Train AUC')
plt.plot(range(1,50,4), cv_auc, label='CV AUC')
plt.legend()
plt.xlabel("K: hyperparameter")
plt.ylabel("AUC")
plt.title("ERROR PLOTS")
plt.show()

plt.scatter(range(1,50,4), train_auc, label='Train AUC')
plt.scatter(range(1,50,4), cv_auc, label='CV AUC')
plt.legend()
plt.xlabel("K: hyperparameter")
plt.ylabel("AUC")
plt.title("ERROR PLOTS")
plt.show()
```

```
  0%|                                                                  | C
[00:00<?, ?it/s]
  8%|█████                                                       | 1/13 [00:
02:56, 14.72s/it]
 15%|██████████                                                  | 2/13
[00:33<02:55, 15.94s/it]
 23%|███████████████                                             | 3/13 [00:
<02:48, 16.90s/it]
 31%|████████████████████                                        | 4/13 [01:
<02:40, 17.87s/it]
 38%|█████████████████████████                                   | 5/13
[01:32<02:27, 18.43s/it]
 46%|██████████████████████████████                              | 6/13
[01:50<02:08, 18.39s/it]
 54%|███████████████████████████████████                         | 7/13 [02:
9<01:50, 18.42s/it]
 62%|████████████████████████████████████████                    | 8/13 [02:
7<01:31, 18.30s/it]
 69%|█████████████████████████████████████████████               | 9/13
[02:45<01:12, 18.18s/it]
 77%|██████████████████████████████████████████████████         | 10/13 [03:
03<00:54, 18.16s/it]
 85%|███████████████████████████████████████████████████████    | 11/13 [03:
21<00:36, 18.06s/it]
 92%|████████████████████████████████████████████████████████████ | 12/13
[03:39<00:18, 18.10s/it]
100%|█████████████████████████████████████████████████████████████████| 13/13
[03:58<00:00, 18.32s/it]
```

```
The optimal number of neighbors is (according to accuracy): 25.

The optimal number of neighbors is (according to auc curve (max auc)): 49.
```



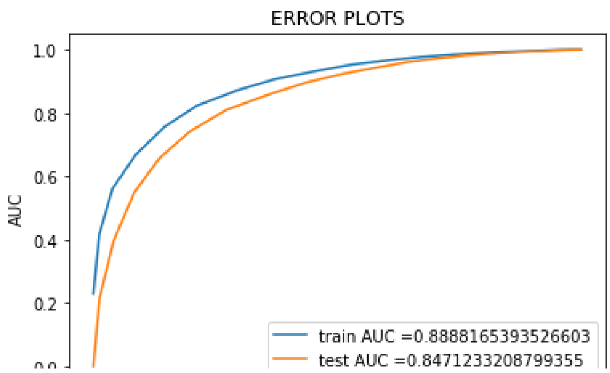


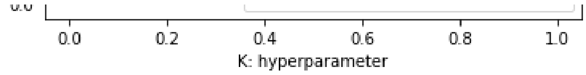


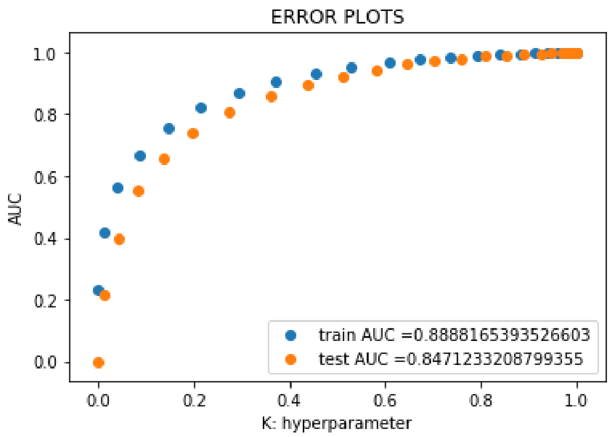
In [41]:

```
knn = KNeighborsClassifier(optimal_k_accuracy,algorithm = 'brute')
knn.fit(X_train_vectors,y_train)
pred = knn.predict(X_test_vectors)
acc = accuracy_score(y_test, pred, normalize=True) * float(100)
print('\n****Test accuracy for k = %d is %f%%' % (optimal_k_accuracy,acc))


train_fpr, train_tpr, thresholds = roc_curve(y_train, knn.predict_proba(X_train_vectors)[:,1])
test_fpr, test_tpr, thresholds = roc_curve(y_test, knn.predict_proba(X_test_vectors)[:,1])

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

plt.scatter(train_fpr, train_tpr, label="train AUC ="+str(auc(train_fpr, train_tpr)))
plt.scatter(test_fpr, test_tpr, label="test AUC ="+str(auc(test_fpr, test_tpr)))
plt.legend()
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")
sns.heatmap(confusion_matrix(y_train, knn.predict(X_train_vectors)))
print("Test confusion matrix")
sns.heatmap(confusion_matrix(y_test, knn.predict(X_test_vectors)))
```

```
****Test accuracy for k = 25 is 85.701131%
```



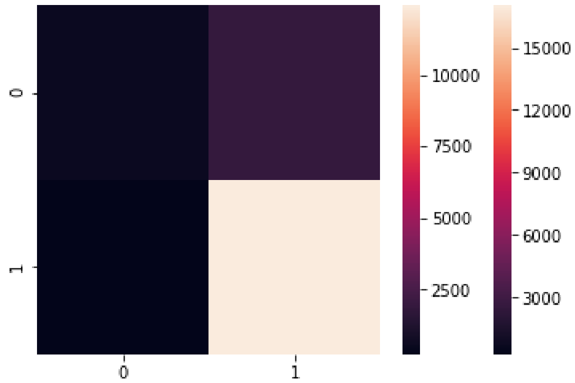





```
==========================================================================================

Train confusion matrix
Test confusion matrix
```

Out[41]:

```
<matplotlib.axes._subplots.AxesSubplot at 0x271714600b8>
```



[5.1.4] Applying KNN brute force on TFIDF W2V, SET 4

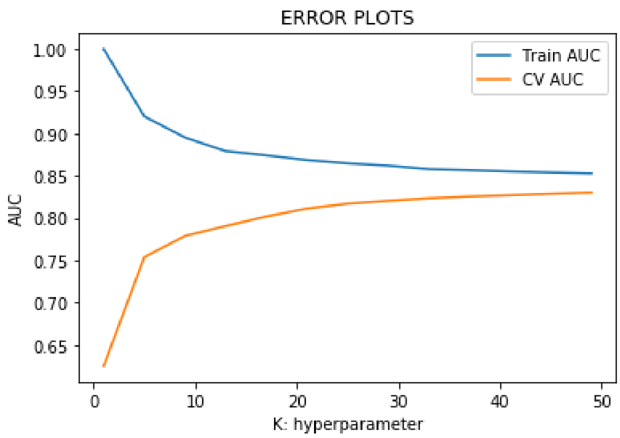
In [42]:

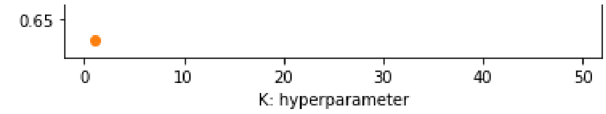
```
# Please write all the code with proper documentation
# Please write all the code with proper documentation
score = []
ind = []
train_auc = []
cv_auc = []
for i in tqdm(range(1,50,4)):
    knn = KNeighborsClassifier(n_neighbors=i,algorithm = 'brute')
    knn.fit(tfidf_X_train_vectors, y_train)
    pred = knn.predict(tfidf_X_cv_vectors)
    acc = accuracy_score(y_cv, pred, normalize=True) * float(100)
    score.append(acc)
    ind.append(i)
    y_train_pred =  knn.predict_proba(tfidf_X_train_vectors)[:,1]
    y_cv_pred =  knn.predict_proba(tfidf_X_cv_vectors)[:,1]
    train_auc.append(roc_auc_score(y_train,y_train_pred))
    cv_auc.append(roc_auc_score(y_cv, y_cv_pred))
optimal_k_accuracy = ind[score.index(max(score))]
print('\nThe optimal number of neighbors is (according to accuracy): %d.' % optimal_k_accuracy)
optimal_k_auc = ind[cv_auc.index(max(cv_auc))]
print('\nThe optimal number of neighbors is (according to auc curve (max auc)): %d.' %
optimal_k_auc)

plt.plot(range(1,50,4), train_auc, label='Train AUC')
plt.plot(range(1,50,4), cv_auc, label='CV AUC')
plt.legend()
```

```
plt.xlabel("K: hyperparameter")
plt.ylabel("AUC")
plt.title("ERROR PLOTS")
plt.show()

plt.scatter(range(1,50,4), train_auc, label='Train AUC')
plt.scatter(range(1,50,4), cv_auc, label='CV AUC')
plt.legend()
plt.xlabel("K: hyperparameter")
plt.ylabel("AUC")
plt.title("ERROR PLOTS")
plt.show()
```

```
  0%|                                                                    | C
[00:00<?, ?it/s]
  8%|████                                                              | 1/13 [00:
02:46, 13.90s/it]
 15%|████████                                                          | 2/13
[00:31<02:45, 15.07s/it]
 23%|████████████                                                      | 3/13 [00:
<02:38, 15.82s/it]
 31%|████████████████                                                  | 4/13 [01:
<02:27, 16.42s/it]
 38%|████████████████████                                              | 5/13
[01:24<02:14, 16.83s/it]
 46%|████████████████████████                                          | 6/13
[01:42<01:59, 17.10s/it]
 54%|████████████████████████████                                      | 7/13 [02:
0<01:43, 17.32s/it]
 62%|████████████████████████████████                                  | 8/13 [02:
8<01:27, 17.53s/it]
 69%|████████████████████████████████████                              | 9/13
[02:36<01:10, 17.68s/it]
 77%|████████████████████████████████████████                         | 10/13 [02:
54<00:53, 17.76s/it]
 85%|████████████████████████████████████████████                     | 11/13 [03:
12<00:35, 17.93s/it]
 92%|████████████████████████████████████████████████                 | 12/13
[03:30<00:17, 17.92s/it]
100%|████████████████████████████████████████████████████| 13/13
[03:48<00:00, 17.98s/it]
```

```
The optimal number of neighbors is (according to accuracy): 13.

The optimal number of neighbors is (according to auc curve (max auc)): 49.
```

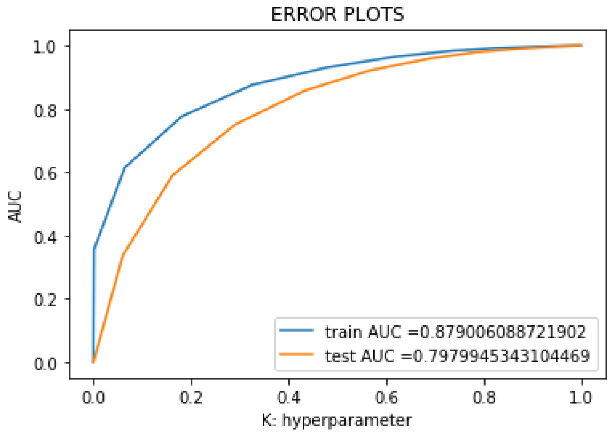


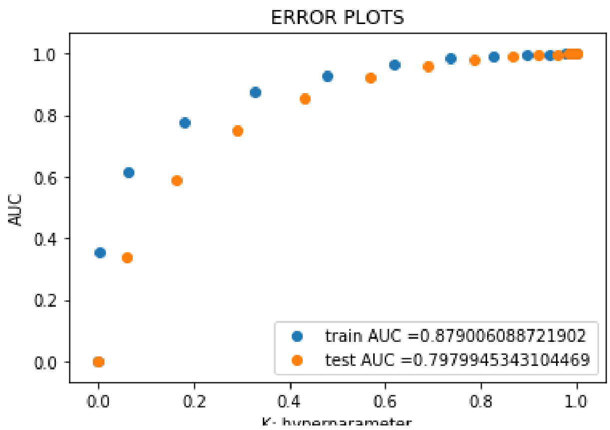


In [43]:

```
knn = KNeighborsClassifier(optimal_k_accuracy,algorithm = 'brute')
knn.fit(tfidf_X_train_vectors,y_train)
pred = knn.predict(tfidf_X_test_vectors)
acc = accuracy_score(y_test, pred, normalize=True) * float(100)
print('\n****Test accuracy for k = %d is %f%%' % (optimal_k_accuracy,acc))

train_fpr, train_tpr, thresholds = roc_curve(y_train, knn.predict_proba(tfidf_X_train_vectors)[:,1]
)
test_fpr, test_tpr, thresholds = roc_curve(y_test, knn.predict_proba(tfidf_X_test_vectors)[:,1])

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

plt.scatter(train_fpr, train_tpr, label="train AUC ="+str(auc(train_fpr, train_tpr)))
plt.scatter(test_fpr, test_tpr, label="test AUC ="+str(auc(test_fpr, test_tpr)))
plt.legend()
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")
sns.heatmap(confusion_matrix(y_train, knn.predict(tfidf_X_train_vectors)))
print("Test confusion matrix")
sns.heatmap(confusion_matrix(y_test, knn.predict(tfidf_X_test_vectors)))
```

```
****Test accuracy for k = 13 is 85.096027%
```



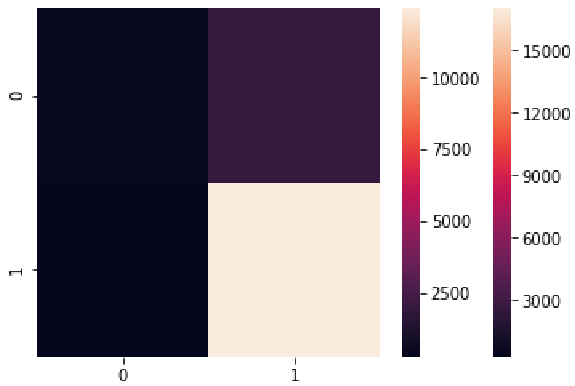


K: hyperparameter

```
===================================================================================================

Train confusion matrix
Test confusion matrix
```

Out[43]:

```
<matplotlib.axes._subplots.AxesSubplot at 0x271718d9d30>
```



[5.2] Applying KNN kd-tree

In [44]:

```
X_train, X_test, y_train, y_test = train_test_split(preprocessed_reviews_fe[1:20000], final['Score'
][1:20000], test_size=0.33) # this is random splitting
X_train, X_cv, y_train, y_cv = train_test_split(X_train, y_train, test_size=0.33) # this is random
splitting
```

BOW (for kd-tree)

In [45]:

```
#BoW
count_vect = CountVectorizer(min_df=10, max_features=500) #in scikit-learn

X_train_vect = count_vect.fit_transform(X_train)
X_train_vect = X_train_vect.toarray()
X_cv_vect = count_vect.transform(X_cv)
X_cv_vect = X_cv_vect.toarray()
X_test_vect = count_vect.transform(X_test)
X_test_vect = X_test_vect.toarray()
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  ['able', 'absolutely', 'actually', 'add', 'added', 'adding', 'ago', 'almost',
'also', 'although']
==================================================
the type of count vectorizer  <class 'scipy.sparse.csr.csr_matrix'>
the shape of out text BOW vectorizer  (46071, 500)
the number of unique words  500
```

TF-IDF (for kd-tree)

In [46]:

```
#tf_idf_vect.fit(preprocessed_reviews)
tf_idf = TfidfVectorizer(min_df=10, max_features=500)
```

```
tf_idf = TfidfVectorizer(min_df=10, max_features=500)
X_train_vect_tfidf = tf_idf.fit_transform(X_train)
X_train_vect_tfidf = X_train_vect_tfidf.toarray()
X_test_vect_tfidf = tf_idf.transform(X_test)
X_test_vect_tfidf = X_test_vect_tfidf.toarray()
X_cv_vect_tfidf = tf_idf.transform(X_cv)
X_cv_vect_tfidf = X_cv_vect_tfidf.toarray()
```

Avg Word2Vec (for kd-tree)

In [47]:

```
# average Word2Vec
# compute average word2vec for each review.
i=0
list_of_sent=[]
X_train_list=[]
X_test_list=[]
X_cv_list=[]
for sent in X_train:
    X_train_list.append(sent.split())

for sent in X_cv:
    X_cv_list.append(sent.split())

for sent in X_test:
    X_test_list.append(sent.split())

w2v_model=Word2Vec(X_train_list,min_count=0,size=50, workers=4)
w2v_words = list(w2v_model.wv.vocab)

X_train_vectors = [];
for sent in tqdm(X_train_list):
    sent_vec = np.zeros(50)
    cnt_words =0;
    for word in sent:
        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_vectors.append(sent_vec)


X_cv_vectors = []
for sent in tqdm(X_cv_list):
    sent_vec = np.zeros(50)
    cnt_words =0;
    for word in sent:
        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_vectors.append(sent_vec)

X_test_vectors = []
for sent in tqdm(X_test_list):
    sent_vec = np.zeros(50)
    cnt_words =0;
    for word in sent:
        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_vectors.append(sent_vec)
```

```
  0%|                                                                  | 0/8
[00:00<?, ?it/s]
  1%|█                                                   | 89/8977 [00:0
```

```
0:10, 883.29it/s]
  2%|██                                                              | 191/8977 [00:0
0:09, 897.52it/s]
  3%|███                                                             | 275/8977 [00:0
0:10, 854.33it/s]
  4%|███                                                             | 353/8977 [00:0
0:10, 806.45it/s]
  5%|████                                                            | 434/8977
[00:00<00:10, 785.51it/s]
  6%|████                                                            | 511/8977
[00:00<00:11, 759.15it/s]
  7%|█████                                                           | 586/8977
[00:00<00:11, 735.54it/s]
  7%|██████                                                          | 653/8977
[00:00<00:12, 661.66it/s]
  8%|██████                                                          | 731/8977
[00:00<00:12, 676.32it/s]
  9%|███████                                                         | 815/8977
[00:01<00:11, 701.46it/s]
 10%|████████                                                        | 884/8977
[00:01<00:11, 678.39it/s]
 11%|████████                                                        | 952/8977
[00:01<00:12, 660.35it/s]
 11%|█████████                                                      | 1018/8977
[00:01<00:12, 642.19it/s]
 12%|██████████                                                     | 1095/8977
[00:01<00:11, 659.58it/s]
 13%|██████████                                                     | 1166/8977
[00:01<00:11, 656.42it/s]
 14%|███████████                                                    | 1232/8977
[00:01<00:12, 639.59it/s]
 14%|███████████                                                    | 1297/8977
[00:01<00:12, 625.31it/s]
 15%|████████████                                                   | 1360/8977
[00:01<00:12, 609.66it/s]
 16%|████████████                                                   | 1422/8977
[00:02<00:12, 596.17it/s]
 17%|█████████████                                                  | 1482/8977
[00:02<00:12, 580.86it/s]
 17%|█████████████                                                  | 1546/8977 [00:0
00:12, 582.15it/s]
 18%|██████████████                                                 | 1605/8977 [00:0
00:12, 568.65it/s]
 19%|███████████████                                                | 1672/8977 [00:0
00:12, 581.16it/s]
 19%|███████████████                                                | 1733/8977 [00:0
00:12, 573.94it/s]
 20%|███████████████                                                | 1791/8977 [00:0
00:13, 535.86it/s]
 21%|████████████████                                               | 1857/8977 [00:0
00:12, 554.50it/s]
 21%|█████████████████                                              | 1914/8977 [00:0
00:16, 428.92it/s]
 22%|█████████████████                                              | 1962/8977 [00:0
00:17, 398.16it/s]
 22%|█████████████████                                              | 2006/8977 [00:0
00:17, 399.40it/s]
 23%|██████████████████                                             | 2059/8977 [00:0
00:16, 421.64it/s]
 23%|██████████████████                                             | 2104/8977 [00:0
00:16, 418.52it/s]
 24%|███████████████████                                            | 2148/8977 [00:0
00:16, 413.53it/s]
 24%|███████████████████                                            | 2191/8977 [00:0
00:16, 407.21it/s]
 25%|████████████████████                                           | 2245/8977 [00:0
00:15, 429.81it/s]
 26%|████████████████████                                           | 2300/8977 [00:0
00:14, 449.40it/s]
 26%|█████████████████████                                          | 2361/8977 [00:0
00:13, 477.21it/s]
 27%|█████████████████████                                          | 2412/8977 [00:0
00:13, 473.74it/s]
 27%|█████████████████████                                          | 2461/8977 [00:0
00:14, 445.87it/s]
 28%|██████████████████████                                         | 2513/8977 [00:0
00:14, 454.35it/s]
```

```
 29%|██████████████████████▎                                                       | 2560/8977 [00:0
00:15, 427.62it/s]
 29%|██████████████████████▋                                                       | 2604/8977 [00:0
00:17, 370.07it/s]
 29%|██████████████████████▉                                                       | 2643/8977 [00:0
00:18, 350.51it/s]
 30%|███████████████████████▎                                                      | 2680/8977 [00:0
00:18, 346.76it/s]
 30%|███████████████████████▋                                                      | 2721/8977 [00:0
00:17, 354.74it/s]
 31%|████████████████████████                                                      | 2767/8977 [00:0
00:16, 372.13it/s]
 31%|████████████████████████▍                                                     | 2815/8977 [00:0
00:15, 389.92it/s]
 32%|████████████████████████▊                                                     | 2855/8977 [00:0
00:16, 382.34it/s]
 32%|█████████████████████████▏                                                    | 2894/8977 [00:0
00:16, 374.23it/s]
 33%|█████████████████████████▌                                                    | 2932/8977 [00:0
00:16, 365.78it/s]
 33%|█████████████████████████▊                                                    | 2969/8977 [00:0
00:19, 314.41it/s]
 33%|██████████████████████████                                                    | 3003/8977 [00:0
00:19, 313.34it/s]
 34%|██████████████████████████▌                                                   | 3051/8977
[00:05<00:17, 342.76it/s]
 34%|██████████████████████████▉                                                   | 3097/8977
[00:06<00:16, 362.91it/s]
 35%|███████████████████████████▍                                                  | 3148/8977
[00:06<00:14, 388.67it/s]
 36%|███████████████████████████▊                                                  | 3201/8977
[00:06<00:13, 413.21it/s]
 36%|████████████████████████████▎                                                 | 3255/8977
[00:06<00:13, 434.47it/s]
 37%|████████████████████████████▊                                                 | 3310/8977
[00:06<00:12, 452.96it/s]
 37%|█████████████████████████████▏                                                | 3357/8977
[00:06<00:13, 426.76it/s]
 38%|█████████████████████████████▌                                                | 3401/8977
[00:06<00:13, 419.12it/s]
 38%|█████████████████████████████▉                                                | 3444/8977
[00:06<00:13, 410.99it/s]
 39%|██████████████████████████████▎                                               | 3486/8977
[00:06<00:13, 402.51it/s]
 39%|██████████████████████████████▋                                               | 3529/8977
[00:07<00:13, 399.66it/s]
 40%|███████████████████████████████                                               | 3581/8977
[00:07<00:12, 419.62it/s]
 40%|███████████████████████████████▍                                              | 3624/8977
[00:07<00:14, 377.47it/s]
 41%|███████████████████████████████▉                                              | 3672/8977
[00:07<00:13, 394.01it/s]
 41%|████████████████████████████████▎                                             | 3713/8977
[00:07<00:13, 388.07it/s]
 42%|████████████████████████████████▌                                             | 3753/8977
[00:07<00:13, 381.09it/s]
 42%|█████████████████████████████████                                             | 3807/8977
[00:07<00:12, 409.10it/s]
 43%|█████████████████████████████████▍                                            | 3849/8977
[00:07<00:12, 401.24it/s]
 43%|█████████████████████████████████▊                                            | 3890/8977
[00:08<00:14, 360.53it/s]
 44%|██████████████████████████████████▎                                           | 3946/8977
[00:08<00:12, 390.68it/s]
 45%|██████████████████████████████████▊                                           | 4001/8977 [00:0
<00:11, 418.71it/s]
 45%|███████████████████████████████████▏                                          | 4048/8977 [00:0
<00:11, 421.96it/s]
 46%|███████████████████████████████████▌                                          | 4092/8977 [00:0
<00:12, 398.22it/s]
 46%|███████████████████████████████████▉                                          | 4133/8977 [00:0
<00:13, 358.85it/s]
 46%|████████████████████████████████████▎                                         | 4171/8977 [00:0
<00:13, 355.35it/s]
 47%|████████████████████████████████████▌                                         | 4208/8977 [00:0
<00:14, 335.20it/s]
 47%|█████████████████████████████████████                                         | 4259/8977 [00:0
```

```
<00:12, 366.08it/s]
 48%|                                                  | 4303/8977 [00:0
<00:12, 376.24it/s]
 48%|                                                  | 4343/8977 [00:0
<00:12, 373.03it/s]
 49%|                                                  | 4384/8977 [00:0
<00:12, 373.58it/s]
 49%|                                                  | 4437/8977 [00:0
<00:11, 401.15it/s]
 50%|                                                  | 4497/8977 [00:0
<00:10, 436.34it/s]
 51%|                                                  | 4543/8977 [00:0
<00:10, 431.50it/s]
 51%|                                                  | 4588/8977 [00:0
<00:12, 361.27it/s]
 52%|                                                  | 4627/8977 [00:0
<00:12, 344.94it/s]
 52%|                                                  | 4665/8977 [00:1
<00:12, 345.69it/s]
 52%|                                                  | 4702/8977 [00:1
<00:12, 343.44it/s]
 53%|                                                  | 4738/8977 [00:1
<00:12, 338.49it/s]
 53%|                                                  | 4775/8977 [00:1
<00:12, 338.44it/s]
 54%|                                                  | 4812/8977 [00:1
<00:12, 338.40it/s]
 54%|                                                  | 4852/8977 [00:1
<00:11, 346.16it/s]
 55%|                                                  | 4899/8977 [00:1
<00:11, 367.61it/s]
 55%|                                                  | 4947/8977 [00:1
<00:10, 386.44it/s]
 56%|                                                  | 4988/8977 [00:1
<00:10, 382.90it/s]
 56%|                                                  | 5038/8977 [00:1
<00:09, 402.52it/s]
 57%|                                                  | 5087/8977 [00:1
<00:09, 415.17it/s]
 57%|                                                  | 5135/8977 [00:1
<00:09, 421.82it/s]
 58%|                                                  | 5178/8977 [00:1
<00:09, 412.79it/s]
 58%|                                                  | 5220/8977 [00:1
<00:09, 403.72it/s]
 59%|                                                  | 5263/8977 [00:1
<00:09, 400.50it/s]
 59%|                                                  | 5308/8977 [00:1
<00:09, 403.72it/s]
 60%|                                                  | 5349/8977 [00:1
<00:09, 377.58it/s]
 60%|                                                  | 5389/8977 [00:1
<00:09, 373.95it/s]
 61%|                                                  | 5435/8977 [00:1
<00:09, 386.82it/s]
 61%|                                                  | 5476/8977 [00:1
<00:09, 383.16it/s]
 61%|                                                  | 5517/8977
[00:12<00:09, 380.52it/s]
 62%|                                                  | 5561/8977
[00:12<00:08, 386.80it/s]
 63%|                                                  | 5611/8977
[00:12<00:08, 405.53it/s]
 63%|                                                  | 5661/8977
[00:12<00:07, 419.76it/s]
 64%|                                                  | 5704/8977
[00:12<00:08, 393.75it/s]
 64%|                                                  | 5744/8977
[00:12<00:09, 339.03it/s]
 64%|                                                  | 5780/8977
[00:12<00:09, 336.01it/s]
 65%|                                                  | 5815/8977
[00:13<00:10, 292.19it/s]
 65%|                                                  | 5847/8977
[00:13<00:11, 269.25it/s]
 66%|                                                  | 5894/8977
[00:13<00:10, 303.22it/s]
```

```
 66%|███████████████████████████████████████████████████                          | 5927/8977
[00:13<00:10, 302.78it/s]
 66%|███████████████████████████████████████████████████                          | 5960/8977
[00:13<00:09, 302.46it/s]
 67%|████████████████████████████████████████████████████                         | 5992/8977
[00:13<00:09, 299.43it/s]
 67%|████████████████████████████████████████████████████                         | 6023/8977
[00:13<00:10, 270.38it/s]
 67%|████████████████████████████████████████████████████                         | 6052/8977
[00:13<00:10, 268.79it/s]
 68%|████████████████████████████████████████████████████                         | 6088/8977
[00:14<00:10, 284.44it/s]
 68%|█████████████████████████████████████████████████████                        | 6125/8977
[00:14<00:09, 298.37it/s]
 69%|█████████████████████████████████████████████████████                        | 6156/8977
[00:14<00:10, 269.75it/s]
 69%|█████████████████████████████████████████████████████                        | 6191/8977
[00:14<00:09, 283.11it/s]
 69%|█████████████████████████████████████████████████████                        | 6224/8977
[00:14<00:09, 288.45it/s]
 70%|██████████████████████████████████████████████████████                       | 6268/8977
[00:14<00:08, 315.21it/s]
 70%|██████████████████████████████████████████████████████                       | 6307/8977
[00:14<00:08, 326.58it/s]
 71%|███████████████████████████████████████████████████████                      | 6351/8977 [00:1
4<00:07, 346.13it/s]
 71%|███████████████████████████████████████████████████████                      | 6387/8977 [00:1
4<00:07, 340.86it/s]
 72%|███████████████████████████████████████████████████████                      | 6426/8977
[00:15<00:07, 345.43it/s]
 72%|███████████████████████████████████████████████████████                      | 6462/8977 [00:1
5<00:07, 340.29it/s]
 72%|███████████████████████████████████████████████████████                      | 6497/8977 [00:1
5<00:07, 319.65it/s]
 73%|████████████████████████████████████████████████████████                     | 6539/8977 [00:1
5<00:07, 336.58it/s]
 73%|████████████████████████████████████████████████████████                     | 6574/8977 [00:1
5<00:07, 331.43it/s]
 74%|█████████████████████████████████████████████████████████                    | 6608/8977 [00:1
5<00:08, 275.60it/s]
 74%|█████████████████████████████████████████████████████████                    | 6638/8977 [00:1
5<00:08, 275.21it/s]
 74%|█████████████████████████████████████████████████████████                    | 6679/8977 [00:1
5<00:07, 299.06it/s]
 75%|██████████████████████████████████████████████████████████                   | 6711/8977 [00:1
6<00:07, 297.09it/s]
 75%|██████████████████████████████████████████████████████████                   | 6752/8977 [00:1
6<00:07, 316.81it/s]
 76%|██████████████████████████████████████████████████████████                   | 6785/8977 [00:1
6<00:08, 255.40it/s]
 76%|██████████████████████████████████████████████████████████                   | 6830/8977 [00:1
6<00:07, 288.19it/s]
 76%|██████████████████████████████████████████████████████████                   | 6863/8977 [00:1
6<00:07, 292.13it/s]
 77%|███████████████████████████████████████████████████████████                  | 6907/8977 [00:1
6<00:06, 318.28it/s]
 77%|███████████████████████████████████████████████████████████                  | 6948/8977 [00:1
6<00:06, 333.38it/s]
 78%|████████████████████████████████████████████████████████████                 | 6996/8977 [00:1
6<00:05, 359.29it/s]
 78%|████████████████████████████████████████████████████████████                 | 7034/8977 [00:1
7<00:06, 302.56it/s]
 79%|█████████████████████████████████████████████████████████████                | 7075/8977 [00:1
7<00:05, 321.15it/s]
 79%|█████████████████████████████████████████████████████████████                | 7118/8977 [00:1
7<00:05, 339.69it/s]
 80%|██████████████████████████████████████████████████████████████               | 7159/8977 [00:1
7<00:05, 349.54it/s]
 80%|██████████████████████████████████████████████████████████████               | 7208/8977 [00:1
7<00:04, 374.22it/s]
 81%|██████████████████████████████████████████████████████████████               | 7247/8977 [00:1
7<00:04, 368.75it/s]
 81%|██████████████████████████████████████████████████████████████               | 7287/8977 [00:1
7<00:04, 367.84it/s]
 82%|███████████████████████████████████████████████████████████████              | 7325/8977 [00:1
7<00:05, 318.83it/s]
 82%|███████████████████████████████████████████████████████████████              | 7366/8977 [00:1
```

```
7<00:04, 333.81it/s]
 82%|█████████████████████████████████████████████████████████████████         | 7401/8977 [00:1
8<00:04, 329.55it/s]
 83%|██████████████████████████████████████████████████████████████████        | 7436/8977 [00:1
8<00:04, 326.53it/s]
 83%|██████████████████████████████████████████████████████████████████        | 7479/8977 [00:1
8<00:04, 344.02it/s]
 84%|██████████████████████████████████████████████████████████████████        | 7521/8977 [00:1
8<00:04, 355.12it/s]
 84%|███████████████████████████████████████████████████████████████████       | 7559/8977 [00:1
8<00:04, 352.78it/s]
 85%|███████████████████████████████████████████████████████████████████       | 7608/8977 [00:1
8<00:03, 376.82it/s]
 85%|███████████████████████████████████████████████████████████████████       | 7647/8977 [00:1
8<00:03, 370.51it/s]
 86%|████████████████████████████████████████████████████████████████████      | 7685/8977 [00:1
8<00:03, 333.40it/s]
 86%|████████████████████████████████████████████████████████████████████      | 7729/8977 [00:1
8<00:03, 351.46it/s]
 87%|█████████████████████████████████████████████████████████████████████     | 7773/8977 [00:1
9<00:03, 365.31it/s]
 87%|█████████████████████████████████████████████████████████████████████     | 7811/8977 [00:1
9<00:03, 344.38it/s]
 88%|██████████████████████████████████████████████████████████████████████    | 7871/8977 [00:1
9<00:02, 387.67it/s]
 88%|██████████████████████████████████████████████████████████████████████    | 7915/8977 [00:1
9<00:02, 391.96it/s]
 89%|██████████████████████████████████████████████████████████████████████    | 7956/8977 [00:1
9<00:02, 386.67it/s]
 89%|██████████████████████████████████████████████████████████████████████    | 7996/8977 [00:1
9<00:03, 289.79it/s]
 89%|███████████████████████████████████████████████████████████████████████   | 8033/8977 [00:1
9<00:03, 302.81it/s]
 90%|███████████████████████████████████████████████████████████████████████   | 8067/8977 [00:2
0<00:03, 270.99it/s]
 90%|███████████████████████████████████████████████████████████████████████   | 8098/8977 [00:2
0<00:03, 274.61it/s]
 91%|████████████████████████████████████████████████████████████████████████  | 8140/8977 [00:2
0<00:02, 300.20it/s]
 91%|████████████████████████████████████████████████████████████████████████  | 8173/8977 [00:2
0<00:02, 277.00it/s]
 91%|████████████████████████████████████████████████████████████████████████  | 8212/8977
[00:20<00:02, 296.88it/s]
 92%|█████████████████████████████████████████████████████████████████████████ | 8257/8977
[00:20<00:02, 323.94it/s]
 92%|█████████████████████████████████████████████████████████████████████████ | 8302/8977
[00:20<00:01, 346.02it/s]
 93%|█████████████████████████████████████████████████████████████████████████ | 8346/8977
[00:20<00:01, 361.18it/s]
 93%|██████████████████████████████████████████████████████████████████████████| 8384/8977
[00:20<00:01, 356.95it/s]
 94%|██████████████████████████████████████████████████████████████████████████| 8421/8977
[00:21<00:01, 351.15it/s]
 94%|██████████████████████████████████████████████████████████████████████████| 8459/8977
[00:21<00:01, 350.03it/s]
 95%|███████████████████████████████████████████████████████████████████████████| 8507/8977
[00:21<00:01, 372.56it/s]
 95%|███████████████████████████████████████████████████████████████████████████| 8546/8977
[00:21<00:01, 367.63it/s]
 96%|████████████████████████████████████████████████████████████████████████████| 8588/8977
[00:21<00:01, 372.40it/s]
 96%|████████████████████████████████████████████████████████████████████████████| 8626/8977
[00:21<00:00, 364.54it/s]
 97%|█████████████████████████████████████████████████████████████████████████████| 8685/8977
[00:21<00:00, 403.83it/s]
 97%|█████████████████████████████████████████████████████████████████████████████| 8727/8977
[00:21<00:00, 380.78it/s]
 98%|█████████████████████████████████████████████████████████████████████████████| 8767/8977 [00:
21<00:00, 376.14it/s]
 98%|██████████████████████████████████████████████████████████████████████████████| 8806/8977 [00:
22<00:00, 370.05it/s]
 99%|██████████████████████████████████████████████████████████████████████████████| 8844/8977 [00:
22<00:00, 347.31it/s]
 99%|██████████████████████████████████████████████████████████████████████████████| 8891/8977 [00:
22<00:00, 368.52it/s]
 99%|██████████████████████████████████████████████████████████████████████████████| 8929/8977
[00:22<00:00, 361.93it/s]
```

```
100%|| 8975/8977 [00:
22<00:00, 377.74it/s]
100%|| 8977/8977
[00:22<00:00, 399.27it/s]
  0%|                                                        | 0/4
[00:00<?, ?it/s]
  1%|| 30/4422 [00:0
0:16, 274.31it/s]
  2%|| 73/4422 [00:0
0:14, 301.67it/s]
  3%|| 111/4422 [00:0
0:13, 314.08it/s]
  3%|| 146/4422 [00:0
0:13, 315.84it/s]
  4%|| 174/4422 [00:0
0:14, 295.15it/s]
  5%|| 210/4422
[00:00<00:14, 292.97it/s]
  5%|| 237/4422
[00:00<00:15, 277.43it/s]
  6%|| 268/4422
[00:00<00:14, 279.21it/s]
  7%|| 308/4422
[00:00<00:13, 300.54it/s]
  8%|| 341/4422
[00:01<00:13, 300.90it/s]
  9%|| 381/4422
[00:01<00:12, 317.80it/s]
  9%|| 413/4422
[00:01<00:12, 309.80it/s]
 10%|| 455/4422
[00:01<00:12, 328.87it/s]
 11%|| 495/4422
[00:01<00:11, 339.12it/s]
 12%|| 550/4422
[00:01<00:10, 363.88it/s]
 13%|| 588/4422
[00:01<00:12, 304.83it/s]
 14%|| 621/4422
[00:01<00:12, 303.90it/s]
 15%|| 665/4422
[00:02<00:11, 327.96it/s]
 16%|| 702/4422
[00:02<00:11, 331.00it/s]
 17%|| 751/4422 [00:0
00:10, 359.14it/s]
 18%|| 793/4422 [00:0
00:09, 366.26it/s]
 19%|| 831/4422 [00:0
00:10, 345.07it/s]
 20%|| 867/4422 [00:0
00:10, 325.71it/s]
 20%|| 901/4422 [00:0
00:11, 307.36it/s]
 21%|| 935/4422 [00:0
00:11, 308.41it/s]
 22%|| 982/4422 [00:0
00:10, 336.95it/s]
 23%|| 1023/4422 [00:0
00:09, 347.50it/s]
 24%|| 1073/4422 [00:0
00:08, 374.44it/s]
 25%|| 1121/4422 [00:0
00:08, 391.70it/s]
 26%|| 1164/4422 [00:0
00:08, 392.14it/s]
 27%|| 1204/4422 [00:0
00:08, 383.83it/s]
 28%|| 1257/4422 [00:0
00:07, 409.37it/s]
 29%|| 1299/4422 [00:0
00:09, 340.33it/s]
 30%|| 1339/4422 [00:0
00:09, 333.97it/s]
 31%|| 1383/4422 [00:0
00:08, 351.90it/s]
 32%|| 1432/4422 [00:0
```

```
00:07, 376.10it/s]
 34%|██████████████████████████                                                    | 1485/4422 [00:0
00:07, 403.20it/s]
 35%|███████████████████████████                                                   | 1527/4422
[00:04<00:08, 337.42it/s]
 35%|███████████████████████████                                                   | 1568/4422
[00:04<00:08, 347.84it/s]
 36%|████████████████████████████                                                  | 1605/4422
[00:04<00:08, 344.83it/s]
 37%|█████████████████████████████                                                 | 1643/4422
[00:04<00:08, 345.60it/s]
 38%|██████████████████████████████                                                | 1688/4422
[00:04<00:07, 363.04it/s]
 39%|██████████████████████████████                                                | 1726/4422
[00:05<00:07, 343.06it/s]
 40%|███████████████████████████████                                               | 1762/4422
[00:05<00:08, 311.31it/s]
 41%|████████████████████████████████                                              | 1803/4422
[00:05<00:07, 327.99it/s]
 42%|████████████████████████████████                                              | 1848/4422
[00:05<00:07, 349.25it/s]
 43%|█████████████████████████████████                                             | 1891/4422
[00:05<00:07, 361.36it/s]
 44%|██████████████████████████████████                                            | 1929/4422
[00:05<00:07, 342.01it/s]
 44%|██████████████████████████████████                                            | 1965/4422 [00:0
<00:09, 249.96it/s]
 45%|███████████████████████████████████                                           | 1996/4422 [00:0
<00:09, 259.15it/s]
 46%|████████████████████████████████████                                          | 2032/4422 [00:0
<00:08, 276.81it/s]
 47%|████████████████████████████████████                                          | 2074/4422 [00:0
<00:07, 302.11it/s]
 48%|█████████████████████████████████████                                         | 2115/4422 [00:0
<00:07, 320.80it/s]
 49%|██████████████████████████████████████                                        | 2155/4422 [00:0
<00:06, 333.08it/s]
 50%|██████████████████████████████████████                                        | 2195/4422 [00:0
<00:06, 342.24it/s]
 51%|███████████████████████████████████████                                       | 2242/4422 [00:0
<00:05, 364.51it/s]
 52%|████████████████████████████████████████                                      | 2297/4422 [00:0
<00:05, 397.22it/s]
 53%|█████████████████████████████████████████                                     | 2339/4422 [00:0
<00:05, 393.17it/s]
 54%|██████████████████████████████████████████                                    | 2385/4422 [00:0
<00:05, 401.01it/s]
 55%|███████████████████████████████████████████                                   | 2438/4422 [00:0
<00:04, 422.90it/s]
 56%|███████████████████████████████████████████                                   | 2482/4422 [00:0
<00:05, 367.58it/s]
 57%|████████████████████████████████████████████                                  | 2521/4422 [00:0
<00:05, 348.94it/s]
 58%|█████████████████████████████████████████████                                 | 2558/4422 [00:0
<00:05, 345.68it/s]
 59%|██████████████████████████████████████████████                                | 2594/4422 [00:0
<00:05, 326.10it/s]
 60%|██████████████████████████████████████████████                                | 2641/4422 [00:0
<00:05, 351.45it/s]
 61%|███████████████████████████████████████████████                               | 2681/4422 [00:0
<00:04, 355.62it/s]
 62%|████████████████████████████████████████████████                              | 2721/4422
[00:07<00:04, 358.59it/s]
 62%|█████████████████████████████████████████████████                             | 2763/4422
[00:07<00:04, 365.86it/s]
 63%|█████████████████████████████████████████████████                             | 2801/4422
[00:08<00:05, 305.80it/s]
 64%|██████████████████████████████████████████████████                            | 2838/4422
[00:08<00:05, 314.88it/s]
 65%|███████████████████████████████████████████████████                           | 2882/4422
[00:08<00:04, 336.84it/s]
 66%|███████████████████████████████████████████████████                           | 2919/4422
[00:08<00:04, 337.28it/s]
 67%|████████████████████████████████████████████████████                          | 2961/4422
[00:08<00:04, 350.06it/s]
 68%|█████████████████████████████████████████████████████                         | 2997/4422
[00:08<00:04, 343.52it/s]
```

```
 69%|█████████████████████████████████████████████████████                         | 3032/4422
[00:08<00:04, 308.36it/s]
 69%|█████████████████████████████████████████████████████                         | 3064/4422
[00:08<00:04, 290.54it/s]
 70%|██████████████████████████████████████████████████████                        | 3095/4422
[00:09<00:04, 288.37it/s]
 71%|███████████████████████████████████████████████████████                       | 3131/4422
[00:09<00:04, 299.51it/s]
 72%|████████████████████████████████████████████████████████                      | 3163/4422
[00:09<00:04, 297.39it/s]
 72%|████████████████████████████████████████████████████████                      | 3199/4422 [00:0
9<00:03, 306.26it/s]
 73%|████████████████████████████████████████████████████████                      | 3242/4422 [00:0
9<00:03, 328.02it/s]
 74%|█████████████████████████████████████████████████████████                     | 3276/4422 [00:0
9<00:03, 296.14it/s]
 75%|██████████████████████████████████████████████████████████                    | 3308/4422 [00:0
9<00:03, 295.06it/s]
 76%|███████████████████████████████████████████████████████████                   | 3341/4422 [00:0
9<00:03, 297.03it/s]
 76%|███████████████████████████████████████████████████████████                   | 3382/4422 [00:0
9<00:03, 316.77it/s]
 77%|████████████████████████████████████████████████████████████                  | 3415/4422 [00:1
0<00:03, 312.10it/s]
 78%|████████████████████████████████████████████████████████████                  | 3453/4422 [00:1
0<00:03, 321.93it/s]
 79%|█████████████████████████████████████████████████████████████                 | 3504/4422 [00:1
0<00:02, 354.90it/s]
 80%|██████████████████████████████████████████████████████████████                | 3546/4422 [00:1
0<00:02, 363.16it/s]
 81%|███████████████████████████████████████████████████████████████               | 3593/4422 [00:1
0<00:02, 380.86it/s]
 82%|████████████████████████████████████████████████████████████████              | 3648/4422 [00:1
0<00:01, 410.76it/s]
 84%|█████████████████████████████████████████████████████████████████             | 3698/4422 [00:1
0<00:01, 423.42it/s]
 85%|██████████████████████████████████████████████████████████████████            | 3742/4422 [00:1
0<00:01, 416.86it/s]
 86%|███████████████████████████████████████████████████████████████████           | 3794/4422 [00:1
0<00:01, 432.86it/s]
 87%|███████████████████████████████████████████████████████████████████           | 3838/4422 [00:1
1<00:01, 404.96it/s]
 88%|████████████████████████████████████████████████████████████████████          | 3880/4422 [00:1
1<00:01, 381.48it/s]
 89%|█████████████████████████████████████████████████████████████████████         | 3920/4422 [00:1
1<00:01, 332.59it/s]
 89%|█████████████████████████████████████████████████████████████████████         | 3955/4422 [00:1
1<00:01, 328.71it/s]
 90%|██████████████████████████████████████████████████████████████████████        | 3990/4422 [00:1
1<00:01, 312.42it/s]
 91%|███████████████████████████████████████████████████████████████████████       | 4027/4422 [00:1
1<00:01, 319.60it/s]
 92%|███████████████████████████████████████████████████████████████████████       | 4060/4422
[00:11<00:01, 300.61it/s]
 93%|████████████████████████████████████████████████████████████████████████      | 4101/4422
[00:11<00:01, 319.61it/s]
 93%|████████████████████████████████████████████████████████████████████████      | 4134/4422
[00:12<00:00, 314.03it/s]
 94%|█████████████████████████████████████████████████████████████████████████     | 4176/4422
[00:12<00:00, 332.19it/s]
 95%|██████████████████████████████████████████████████████████████████████████    | 4214/4422
[00:12<00:00, 336.63it/s]
 96%|██████████████████████████████████████████████████████████████████████████    | 4249/4422
[00:12<00:00, 331.47it/s]
 97%|███████████████████████████████████████████████████████████████████████████   | 4287/4422
[00:12<00:00, 336.11it/s]
 98%|████████████████████████████████████████████████████████████████████████████  | 4330/4422 [00:
12<00:00, 351.41it/s]
 99%|█████████████████████████████████████████████████████████████████████████████ | 4366/4422 [00:
12<00:00, 344.25it/s]
100%|█████████████████████████████████████████████████████████████████████████████▉| 4408/4422 [00:
12<00:00, 329.19it/s]
100%|██████████████████████████████████████████████████████████████████████████████| 4422/4422
[00:12<00:00, 342.60it/s]
  0%|                                                                                | 0/6
[00:00<?, ?it/s]
  0%|▍                                                                               | 31/6600 [00:0
```

```
  0%|▏                                                                               | 31/6600 [00:0
0:23, 283.44it/s]
  1%|▌                                                                               | 58/6600 [00:0
0:24, 271.38it/s]
  1%|█                                                                               | 92/6600 [00:0
0:23, 282.14it/s]
  2%|██                                                                             | 142/6600 [00:0
0:20, 318.75it/s]
  3%|██                                                                             | 169/6600 [00:0
0:23, 278.95it/s]
  3%|██                                                                             | 204/6600 [00:0
0:22, 290.12it/s]
  4%|███                                                                            | 236/6600 [00:0
0:21, 290.73it/s]
  4%|███                                                                            | 272/6600 
[00:00<00:21, 301.28it/s]
  5%|████                                                                           | 320/6600 
[00:01<00:18, 332.56it/s]
  5%|████                                                                           | 355/6600 
[00:01<00:18, 328.70it/s]
  6%|█████                                                                          | 389/6600 
[00:01<00:19, 323.14it/s]
  6%|█████                                                                          | 427/6600 
[00:01<00:18, 330.07it/s]
  7%|██████                                                                         | 468/6600 
[00:01<00:18, 329.45it/s]
  8%|██████                                                                         | 502/6600 
[00:01<00:18, 323.65it/s]
  8%|██████                                                                         | 540/6600 
[00:01<00:18, 330.44it/s]
  9%|███████                                                                        | 577/6600 
[00:01<00:18, 332.63it/s]
  9%|███████                                                                        | 615/6600 
[00:01<00:17, 336.94it/s]
 10%|████████                                                                       | 651/6600 
[00:02<00:17, 334.57it/s]
 11%|████████                                                                       | 701/6600 
[00:02<00:16, 363.85it/s]
 11%|█████████                                                                      | 750/6600 
[00:02<00:15, 385.58it/s]
 12%|█████████                                                                      | 798/6600 
[00:02<00:14, 400.16it/s]
 13%|██████████                                                                     | 839/6600 
[00:02<00:17, 332.58it/s]
 13%|██████████                                                                     | 883/6600 
[00:02<00:16, 350.82it/s]
 14%|███████████                                                                    | 935/6600 
[00:02<00:14, 380.77it/s]
 15%|████████████                                                                   | 978/6600 
[00:02<00:14, 384.26it/s]
 15%|████████████                                                                  | 1018/6600 
[00:02<00:16, 347.63it/s]
 16%|████████████                                                                  | 1055/6600 
[00:03<00:16, 330.39it/s]
 17%|█████████████                                                                 | 1094/6600 
[00:03<00:16, 324.66it/s]
 17%|█████████████                                                                 | 1136/6600 [00:0
00:16, 340.44it/s]
 18%|██████████████                                                                | 1180/6600 [00:0
00:15, 356.91it/s]
 18%|██████████████                                                                | 1217/6600 [00:0
00:16, 336.17it/s]
 19%|███████████████                                                               | 1252/6600 [00:0
00:16, 317.09it/s]
 20%|███████████████                                                               | 1289/6600 [00:0
00:16, 323.07it/s]
 20%|████████████████                                                              | 1333/6600 [00:0
00:15, 343.36it/s]
 21%|████████████████                                                              | 1369/6600 [00:0
00:16, 324.65it/s]
 21%|█████████████████                                                             | 1416/6600 [00:0
00:14, 350.35it/s]
 22%|█████████████████                                                             | 1453/6600 [00:0
00:15, 332.07it/s]
 23%|██████████████████                                                            | 1488/6600 [00:0
00:19, 259.81it/s]
 23%|██████████████████                                                            | 1524/6600 [00:0
00:18, 277.34it/s]
```

```
00:18, 277.34it/s]
 24%|██████████████████                                                            | 1565/6600 [00:0
00:16, 300.82it/s]
 24%|███████████████████                                                           | 1609/6600 [00:0
00:15, 325.37it/s]
 25%|███████████████████                                                           | 1655/6600 [00:0
00:14, 349.08it/s]
 26%|████████████████████                                                          | 1704/6600 [00:0
00:13, 373.86it/s]
 26%|████████████████████                                                          | 1746/6600 [00:0
00:12, 376.85it/s]
 27%|█████████████████████                                                         | 1786/6600 [00:0
00:13, 357.78it/s]
 28%|█████████████████████                                                         | 1823/6600 [00:0
00:13, 351.72it/s]
 28%|█████████████████████                                                         | 1859/6600 [00:0
00:13, 344.63it/s]
 29%|██████████████████████                                                        | 1895/6600 [00:0
00:14, 325.44it/s]
 29%|██████████████████████                                                        | 1929/6600 [00:0
00:15, 294.75it/s]
 30%|███████████████████████                                                       | 1960/6600 [00:0
00:15, 291.27it/s]
 30%|███████████████████████                                                       | 1996/6600 [00:0
00:18, 252.16it/s]
 31%|████████████████████████                                                      | 2035/6600 [00:0
00:16, 276.45it/s]
 31%|████████████████████████                                                      | 2075/6600 [00:0
00:15, 298.30it/s]
 32%|█████████████████████████                                                     | 2118/6600 [00:0
00:13, 321.58it/s]
 33%|█████████████████████████                                                     | 2158/6600 [00:0
00:13, 333.66it/s]
 33%|█████████████████████████                                                     | 2194/6600 [00:0
00:13, 332.30it/s]
 34%|██████████████████████████                                                    | 2229/6600 [00:0
00:13, 314.68it/s]
 35%|██████████████████████████                                                    | 2283/6600
[00:06<00:12, 353.02it/s]
 35%|███████████████████████████                                                   | 2327/6600
[00:06<00:11, 366.49it/s]
 36%|███████████████████████████                                                   | 2366/6600
[00:07<00:11, 363.47it/s]
 36%|████████████████████████████                                                  | 2404/6600
[00:07<00:11, 358.51it/s]
 37%|████████████████████████████                                                  | 2446/6600
[00:07<00:11, 365.80it/s]
 38%|█████████████████████████████                                                 | 2494/6600
[00:07<00:10, 385.03it/s]
 38%|█████████████████████████████                                                 | 2537/6600
[00:07<00:10, 387.44it/s]
 39%|██████████████████████████████                                                | 2587/6600
[00:07<00:09, 406.02it/s]
 40%|███████████████████████████████                                               | 2629/6600
[00:07<00:09, 399.16it/s]
 40%|███████████████████████████████                                               | 2670/6600
[00:07<00:10, 391.40it/s]
 41%|████████████████████████████████                                              | 2710/6600
[00:07<00:10, 383.33it/s]
 42%|████████████████████████████████                                              | 2749/6600
[00:08<00:10, 374.90it/s]
 42%|████████████████████████████████                                              | 2787/6600
[00:08<00:11, 322.52it/s]
 43%|█████████████████████████████████                                             | 2829/6600
[00:08<00:11, 326.45it/s]
 44%|██████████████████████████████████                                            | 2881/6600
[00:08<00:10, 360.33it/s]
 44%|██████████████████████████████████                                            | 2934/6600 [00:0
<00:09, 390.36it/s]
 45%|███████████████████████████████████                                           | 2980/6600 [00:0
<00:09, 398.97it/s]
 46%|███████████████████████████████████                                           | 3026/6600 [00:0
<00:08, 405.08it/s]
 46%|████████████████████████████████████                                          | 3068/6600 [00:0
<00:08, 398.52it/s]
 47%|████████████████████████████████████                                          | 3119/6600 [00:0
<00:08, 416.68it/s]
 48%|█████████████████████████████████████                                         | 3166/6600 [00:0
```

```
 48%|█████████████████████████████████████                                         | 3166/6600 [00:0
<00:08, 420.54it/s]
 49%|██████████████████████████████████████                                        | 3209/6600 [00:0
<00:08, 411.93it/s]
 49%|██████████████████████████████████████                                        | 3251/6600 [00:0
<00:08, 385.79it/s]
 50%|███████████████████████████████████████                                       | 3291/6600 [00:0
<00:08, 379.54it/s]
 51%|███████████████████████████████████████                                       | 3334/6600 [00:0
<00:08, 383.52it/s]
 51%|████████████████████████████████████████                                      | 3373/6600 [00:0
<00:08, 375.04it/s]
 52%|████████████████████████████████████████                                      | 3411/6600 [00:0
<00:09, 350.37it/s]
 52%|█████████████████████████████████████████                                     | 3464/6600 [00:0
<00:08, 382.13it/s]
 53%|█████████████████████████████████████████                                     | 3504/6600 [00:0
<00:08, 377.06it/s]
 54%|██████████████████████████████████████████                                    | 3546/6600 [00:1
<00:08, 379.12it/s]
 54%|██████████████████████████████████████████                                    | 3595/6600 [00:1
<00:07, 397.46it/s]
 55%|███████████████████████████████████████████                                   | 3641/6600 [00:1
<00:07, 404.13it/s]
 56%|████████████████████████████████████████████                                  | 3690/6600 [00:1
<00:06, 416.37it/s]
 57%|████████████████████████████████████████████                                  | 3743/6600 [00:1
<00:06, 418.64it/s]
 57%|█████████████████████████████████████████████                                 | 3786/6600 [00:1
<00:06, 410.66it/s]
 58%|█████████████████████████████████████████████                                 | 3832/6600 [00:1
<00:06, 413.46it/s]
 59%|██████████████████████████████████████████████                                | 3874/6600 [00:1
<00:06, 404.17it/s]
 59%|██████████████████████████████████████████████                                | 3915/6600 [00:1
<00:07, 377.85it/s]
 60%|███████████████████████████████████████████████                               | 3954/6600 [00:1
<00:07, 371.22it/s]
 60%|███████████████████████████████████████████████                               | 3992/6600 [00:1
<00:07, 348.13it/s]
 61%|████████████████████████████████████████████████                              | 4036/6600 [00:1
<00:07, 362.79it/s]
 62%|████████████████████████████████████████████████                              | 4073/6600
[00:11<00:07, 355.08it/s]
 62%|█████████████████████████████████████████████████                             | 4115/6600
[00:11<00:06, 363.30it/s]
 63%|█████████████████████████████████████████████████                             | 4165/6600
[00:11<00:06, 387.15it/s]
 64%|██████████████████████████████████████████████████                            | 4205/6600
[00:11<00:06, 364.12it/s]
 64%|██████████████████████████████████████████████████                            | 4243/6600
[00:11<00:06, 358.95it/s]
 65%|███████████████████████████████████████████████████                           | 4280/6600
[00:12<00:06, 352.50it/s]
 66%|███████████████████████████████████████████████████                           | 4332/6600
[00:12<00:05, 382.15it/s]
 66%|████████████████████████████████████████████████████                          | 4372/6600
[00:12<00:06, 361.12it/s]
 67%|████████████████████████████████████████████████████                          | 4410/6600
[00:12<00:06, 341.86it/s]
 67%|█████████████████████████████████████████████████████                         | 4452/6600
[00:12<00:06, 340.09it/s]
 68%|█████████████████████████████████████████████████████                         | 4493/6600
[00:12<00:06, 349.83it/s]
 69%|██████████████████████████████████████████████████████                        | 4537/6600
[00:12<00:05, 364.08it/s]
 69%|██████████████████████████████████████████████████████                        | 4574/6600
[00:12<00:05, 355.83it/s]
 70%|███████████████████████████████████████████████████████                       | 4632/6600
[00:12<00:04, 394.80it/s]
 71%|███████████████████████████████████████████████████████                       | 4678/6600
[00:13<00:04, 402.20it/s]
 72%|████████████████████████████████████████████████████████                      | 4720/6600
[00:13<00:04, 379.76it/s]
 72%|████████████████████████████████████████████████████████                      | 4760/6600 [00:1
3<00:05, 345.08it/s]
 73%|█████████████████████████████████████████████████████████                     | 4806/6600 [00:1
3<00:04, 364.72it/s]
```

```
3<00:04, 364.73it/s]
 73%|█████████████████████████████████████████████████████████           | 4847/6600 [00:1
3<00:04, 367.72it/s]
 74%|█████████████████████████████████████████████████████████▉          | 4901/6600 [00:1
3<00:04, 398.21it/s]
 75%|██████████████████████████████████████████████████████████▍         | 4943/6600 [00:1
3<00:04, 393.72it/s]
 76%|███████████████████████████████████████████████████████████         | 4996/6600 [00:1
3<00:03, 417.19it/s]
 77%|███████████████████████████████████████████████████████████▊        | 5058/6600 [00:1
3<00:03, 453.08it/s]
 77%|████████████████████████████████████████████████████████████▎       | 5105/6600 [00:1
4<00:03, 393.36it/s]
 78%|████████████████████████████████████████████████████████████▉       | 5154/6600 [00:1
4<00:03, 408.30it/s]
 79%|█████████████████████████████████████████████████████████████▌      | 5213/6600 [00:1
4<00:03, 440.43it/s]
 80%|██████████████████████████████████████████████████████████████▏     | 5263/6600 [00:1
4<00:03, 445.32it/s]
 81%|██████████████████████████████████████████████████████████████▊     | 5318/6600 [00:1
4<00:02, 461.16it/s]
 81%|███████████████████████████████████████████████████████████████▍    | 5366/6600 [00:1
4<00:02, 434.95it/s]
 82%|███████████████████████████████████████████████████████████████▉    | 5411/6600 [00:1
4<00:02, 409.27it/s]
 83%|████████████████████████████████████████████████████████████████▍   | 5454/6600 [00:1
4<00:02, 404.30it/s]
 83%|████████████████████████████████████████████████████████████████▉   | 5496/6600 [00:1
5<00:02, 398.00it/s]
 84%|█████████████████████████████████████████████████████████████████▍  | 5537/6600 [00:1
5<00:02, 390.77it/s]
 84%|█████████████████████████████████████████████████████████████████▉  | 5577/6600 [00:1
5<00:02, 382.91it/s]
 85%|██████████████████████████████████████████████████████████████████▍ | 5621/6600 [00:1
5<00:02, 388.53it/s]
 86%|██████████████████████████████████████████████████████████████████▉ | 5661/6600 [00:1
5<00:02, 381.40it/s]
 87%|███████████████████████████████████████████████████████████████████▍| 5710/6600 [00:1
5<00:02, 399.21it/s]
 87%|████████████████████████████████████████████████████████████████████| 5757/6600 [00:1
5<00:02, 407.90it/s]
 88%|████████████████████████████████████████████████████████████████████▌| 5799/6600 [00:1
5<00:02, 383.18it/s]
 88%|█████████████████████████████████████████████████████████████████████ | 5839/6600 [00:1
5<00:02, 377.78it/s]
 89%|█████████████████████████████████████████████████████████████████████▍| 5878/6600 [00:1
6<00:01, 371.16it/s]
 90%|█████████████████████████████████████████████████████████████████████▉| 5916/6600 [00:1
6<00:01, 348.10it/s]
 90%|██████████████████████████████████████████████████████████████████████▎| 5952/6600 [00:1
6<00:02, 244.25it/s]
 91%|██████████████████████████████████████████████████████████████████████▉| 5999/6600 [00:1
6<00:02, 280.59it/s]
 92%|███████████████████████████████████████████████████████████████████████▍| 6044/6600
[00:16<00:01, 310.18it/s]
 92%|███████████████████████████████████████████████████████████████████████▊| 6080/6600
[00:16<00:01, 315.64it/s]
 93%|████████████████████████████████████████████████████████████████████████▎| 6121/6600
[00:16<00:01, 331.22it/s]
 93%|████████████████████████████████████████████████████████████████████████▊| 6164/6600
[00:16<00:01, 334.95it/s]
 94%|█████████████████████████████████████████████████████████████████████████▎| 6205/6600
[00:17<00:01, 320.65it/s]
 95%|█████████████████████████████████████████████████████████████████████████▋| 6239/6600
[00:17<00:01, 317.65it/s]
 95%|██████████████████████████████████████████████████████████████████████████▏| 6283/6600
[00:17<00:00, 339.06it/s]
 96%|██████████████████████████████████████████████████████████████████████████▋| 6318/6600
[00:17<00:00, 333.12it/s]
 96%|███████████████████████████████████████████████████████████████████████████▎| 6369/6600
[00:17<00:00, 364.34it/s]
 97%|███████████████████████████████████████████████████████████████████████████▋| 6409/6600
[00:17<00:00, 364.75it/s]
 98%|████████████████████████████████████████████████████████████████████████████▏| 6447/6600 [00:
17<00:00, 359.26it/s]
 98%|████████████████████████████████████████████████████████████████████████████▌| 6484/6600 [00:
17<00:00, 323.78it/s]
```

```
 99%|██████████████████████████████████████████████████████████████████████████| 6518/6600 [00:
18<00:00, 319.80it/s]
 99%|██████████████████████████████████████████████████████████████████████████| 6564/6600
[00:18<00:00, 344.57it/s]
100%|███████████████████████████████████████████████████████████████████████████| 6600/6600
[00:18<00:00, 362.56it/s]
```

TFIDF-Word2Vec (for kd-tree)

In [48]:

```
dictionary = dict(zip(tf_idf.get_feature_names(), list(tf_idf.idf_)))
tfidf_feat = tf_idf.get_feature_names()
tfidf_X_train_vectors = [];
tfidf_X_test_vectors = [];
tfidf_X_cv_vectors = [];

for sent in tqdm(X_train_list):
    sent_vec = np.zeros(50)
    weight_sum =0;
    for word in sent:
        if word in (w2v_words and tfidf_feat):
            vec = w2v_model.wv[word]
            tf_idf_count = dictionary[word]*sent.count(word)
            sent_vec += (vec * tf_idf_count)
            weight_sum += tf_idf_count
    if weight_sum != 0:
        sent_vec /= weight_sum
    tfidf_X_train_vectors.append(sent_vec)

for sent in tqdm(X_test_list):
    sent_vec = np.zeros(50)
    weight_sum =0;
    for word in sent:
        if word in (w2v_words and tfidf_feat):
            vec = w2v_model.wv[word]
            tf_idf_count = dictionary[word]*sent.count(word)
            sent_vec += (vec * tf_idf_count)
            weight_sum += tf_idf_count
    if weight_sum != 0:
        sent_vec /= weight_sum
    tfidf_X_test_vectors.append(sent_vec)

for sent in tqdm(X_cv_list):
    sent_vec = np.zeros(50)
    weight_sum =0;
    for word in sent:
        if word in (w2v_words and tfidf_feat):
            vec = w2v_model.wv[word]
            tf_idf_count = dictionary[word]*sent.count(word)
            sent_vec += (vec * tf_idf_count)
            weight_sum += tf_idf_count
    if weight_sum != 0:
        sent_vec /= weight_sum
    tfidf_X_cv_vectors.append(sent_vec)
```

```
  0%|                                                                              | 0/8
[00:00<?, ?it/s]
  2%|█                                                                   | 138/8977 [00:0
:07, 1261.99it/s]
  3%|██                                                                  | 297/8977 [00:0
:06, 1314.01it/s]
  5%|███                                                                 | 472/8977
[00:00<00:06, 1388.49it/s]
  7%|████                                                                | 652/8977
[00:00<00:05, 1456.82it/s]
 10%|█████                                                               | 869/8977
[00:00<00:05, 1583.04it/s]
 12%|██████                                                             | 1067/8977
[00:00<00:04, 1645.03it/s]
 14%|███████                                                            | 1261/8977
[00:00<00:04, 1681.66it/s]
 16%|████████                                                           | 1433/8977
[00:00<00:04, 1647.38it/s]
```

```
 18%|█████████████                                                     | 1593/8977 [00:00
0:04, 1587.38it/s]
 20%|███████████████                                                   | 1759/8977 [00:01
0:04, 1564.96it/s]
 21%|████████████████                                                  | 1914/8977 [00:01
0:04, 1450.92it/s]
 23%|█████████████████                                                 | 2060/8977 [00:01
0:04, 1414.07it/s]
 25%|██████████████████                                                | 2202/8977 [00:01
0:04, 1377.25it/s]
 27%|████████████████████                                              | 2384/8977 [00:01
0:04, 1452.37it/s]
 29%|█████████████████████                                             | 2564/8977 [00:01
0:04, 1505.46it/s]
 30%|███████████████████████                                           | 2730/8977 [00:01
0:04, 1509.15it/s]
 32%|████████████████████████                                          | 2914/8977
[00:01<00:03, 1557.26it/s]
 34%|██████████████████████████                                        | 3097/8977
[00:01<00:03, 1590.33it/s]
 36%|███████████████████████████                                       | 3257/8977
[00:02<00:03, 1482.01it/s]
 38%|█████████████████████████████                                     | 3408/8977
[00:02<00:04, 1387.67it/s]
 40%|██████████████████████████████                                    | 3571/8977
[00:02<00:03, 1416.97it/s]
 42%|███████████████████████████████                                   | 3742/8977
[00:02<00:03, 1457.97it/s]
 43%|█████████████████████████████████                                 | 3890/8977
[00:02<00:03, 1424.90it/s]
 45%|██████████████████████████████████                                | 4034/8977 [00:02
00:03, 1275.16it/s]
 46%|███████████████████████████████████                               | 4166/8977 [00:02
00:04, 1200.45it/s]
 48%|████████████████████████████████████                              | 4290/8977 [00:02
00:04, 1129.29it/s]
 50%|██████████████████████████████████████                            | 4455/8977 [00:03
00:03, 1221.08it/s]
 51%|███████████████████████████████████████                           | 4591/8977 [00:03
00:03, 1227.72it/s]
 53%|████████████████████████████████████████                          | 4717/8977 [00:03
00:03, 1204.00it/s]
 54%|█████████████████████████████████████████                         | 4840/8977 [00:03
00:03, 1179.04it/s]
 56%|██████████████████████████████████████████                        | 4985/8977 [00:03
00:03, 1219.56it/s]
 57%|███████████████████████████████████████████                       | 5120/8977 [00:03
00:03, 1223.93it/s]
 59%|████████████████████████████████████████████                      | 5252/8977 [00:03
00:03, 1218.82it/s]
 60%|██████████████████████████████████████████████                    | 5407/8977 [00:03
00:02, 1272.26it/s]
 62%|███████████████████████████████████████████████                   | 5577/8977
[00:03<00:02, 1345.52it/s]
 64%|█████████████████████████████████████████████████                 | 5761/8977
[00:04<00:02, 1436.92it/s]
 66%|██████████████████████████████████████████████████                | 5939/8977
[00:04<00:02, 1489.25it/s]
 68%|████████████████████████████████████████████████████              | 6107/8977
[00:04<00:01, 1503.00it/s]
 70%|█████████████████████████████████████████████████████             | 6292/8977
[00:04<00:01, 1555.00it/s]
 72%|███████████████████████████████████████████████████████           | 6455/8977 [00:04
<00:01, 1535.02it/s]
 74%|████████████████████████████████████████████████████████          | 6616/8977 [00:04
<00:01, 1515.60it/s]
 75%|█████████████████████████████████████████████████████████         | 6773/8977 [00:04
<00:01, 1490.67it/s]
 77%|███████████████████████████████████████████████████████████       | 6948/8977 [00:04
<00:01, 1521.89it/s]
 79%|████████████████████████████████████████████████████████████      | 7101/8977 [00:04
<00:01, 1418.37it/s]
 81%|█████████████████████████████████████████████████████████████     | 7245/8977 [00:05
<00:01, 1386.25it/s]
 82%|███████████████████████████████████████████████████████████████   | 7386/8977 [00:05
<00:01, 1355.64it/s]
 84%|████████████████████████████████████████████████████████████████  | 7523/8977 [00:05
```

```
<00:01, 1265.72it/s]
 85%|████████████████████████████████████████████████████████████████           | 7664/8977 [00:05
<00:01, 1272.68it/s]
 87%|█████████████████████████████████████████████████████████████████          | 7793/8977 [00:05
<00:00, 1189.48it/s]
 89%|██████████████████████████████████████████████████████████████████         | 7956/8977
[00:05<00:00, 1266.18it/s]
 90%|███████████████████████████████████████████████████████████████████        | 8086/8977
[00:05<00:00, 1139.81it/s]
 91%|████████████████████████████████████████████████████████████████████       | 8211/8977
[00:05<00:00, 1140.75it/s]
 93%|█████████████████████████████████████████████████████████████████████      | 8339/8977
[00:05<00:00, 1149.47it/s]
 94%|██████████████████████████████████████████████████████████████████████     | 8476/8977
[00:06<00:00, 1176.83it/s]
 96%|████████████████████████████████████████████████████████████████████████   | 8625/8977
[00:06<00:00, 1226.96it/s]
 98%|█████████████████████████████████████████████████████████████████████████  | 8791/8977
[00:06<00:00, 1301.81it/s]
100%|███████████████████████████████████████████████████████████████████████████| 8977/8977
[00:06<00:00, 1398.11it/s]
  0%|                                                                                     | 0/6
[00:00<?, ?it/s]
  2%|█                                                                          | 143/6600 [00:00
:04, 1307.52it/s]
  5%|███                                                                        | 310/6600
[00:00<00:04, 1366.43it/s]
  7%|█████                                                                      | 438/6600
[00:00<00:04, 1301.05it/s]
  9%|██████                                                                     | 578/6600
[00:00<00:04, 1294.68it/s]
 11%|████████                                                                   | 724/6600
[00:00<00:04, 1306.51it/s]
 13%|█████████                                                                  | 847/6600
[00:00<00:04, 1245.66it/s]
 16%|███████████                                                               | 1024/6600
[00:00<00:04, 1338.12it/s]
 17%|█████████████                                                             | 1150/6600 [00:00
0:04, 1276.29it/s]
 19%|██████████████                                                            | 1273/6600 [00:00
0:04, 1226.66it/s]
 21%|███████████████                                                           | 1393/6600 [00:01
0:04, 1184.74it/s]
 23%|█████████████████                                                         | 1511/6600 [00:01
0:04, 1150.87it/s]
 25%|██████████████████                                                        | 1665/6600 [00:01
0:04, 1217.60it/s]
 28%|████████████████████                                                      | 1853/6600 [00:01
0:03, 1334.35it/s]
 30%|██████████████████████                                                    | 2000/6600 [00:01
0:03, 1337.27it/s]
 33%|████████████████████████                                                  | 2188/6600
[00:01<00:03, 1432.30it/s]
 36%|███████████████████████████                                               | 2387/6600
[00:01<00:02, 1529.98it/s]
 39%|█████████████████████████████                                             | 2589/6600
[00:01<00:02, 1613.03it/s]
 42%|███████████████████████████████                                           | 2762/6600
[00:01<00:02, 1603.54it/s]
 45%|█████████████████████████████████                                         | 2960/6600
[00:02<00:02, 1660.46it/s]
 48%|████████████████████████████████████                                      | 3176/6600 [00:02
00:01, 1743.78it/s]
 51%|██████████████████████████████████████                                    | 3362/6600 [00:02
00:01, 1730.62it/s]
 54%|████████████████████████████████████████                                  | 3546/6600 [00:02
00:01, 1715.86it/s]
 57%|██████████████████████████████████████████                                | 3772/6600 [00:02
00:01, 1807.88it/s]
 60%|█████████████████████████████████████████████                             | 3970/6600 [00:02
00:01, 1808.64it/s]
 63%|███████████████████████████████████████████████                           | 4170/6600
[00:02<00:01, 1814.03it/s]
 66%|█████████████████████████████████████████████████                         | 4365/6600
[00:02<00:01, 1804.59it/s]
 69%|███████████████████████████████████████████████████                       | 4557/6600
[00:02<00:01, 1789.60it/s]
```

```
 72%|██████████████████████████████████████████████████████▋                    | 4782/6600
[00:03<00:00, 1862.29it/s]
 76%|█████████████████████████████████████████████████████████▏                 | 5008/6600 [00:03
<00:00, 1919.17it/s]
 79%|███████████████████████████████████████████████████████████▊               | 5228/6600 [00:03
<00:00, 1945.98it/s]
 82%|██████████████████████████████████████████████████████████████▏            | 5439/6600 [00:03
<00:00, 1940.94it/s]
 85%|████████████████████████████████████████████████████████████████▍          | 5638/6600 [00:03
<00:00, 1902.86it/s]
 89%|██████████████████████████████████████████████████████████████████▉        | 5856/6600
[00:03<00:00, 1929.10it/s]
 92%|█████████████████████████████████████████████████████████████████████▏     | 6050/6600
[00:03<00:00, 1879.10it/s]
 95%|███████████████████████████████████████████████████████████████████████▎   | 6241/6600
[00:03<00:00, 1837.23it/s]
 98%|█████████████████████████████████████████████████████████████████████████▌ | 6438/6600
[00:03<00:00, 1826.29it/s]
100%|███████████████████████████████████████████████████████████████████████████| 6600/6600
[00:04<00:00, 1637.13it/s]
  0%|                                                                                | 0/4
[00:00<?, ?it/s]
  4%|███                                                                         | 178/4422 [00:00
:02, 1627.57it/s]
  8%|██████▏                                                                     | 360/4422
[00:00<00:02, 1638.36it/s]
 13%|█████████▉                                                                  | 580/4422
[00:00<00:02, 1734.92it/s]
 18%|█████████████▍                                                              | 782/4422
[00:00<00:02, 1767.10it/s]
 22%|████████████████▌                                                           | 966/4422 [00:00
0:01, 1740.80it/s]
 27%|████████████████████                                                       | 1180/4422 [00:00
0:01, 1799.83it/s]
 31%|███████████████████████▏                                                   | 1367/4422 [00:00
0:01, 1771.85it/s]
 35%|██████████████████████████▌                                                | 1565/4422
[00:00<00:01, 1783.22it/s]
 40%|█████████████████████████████▋                                             | 1753/4422
[00:00<00:01, 1763.47it/s]
 44%|████████████████████████████████▉                                          | 1945/4422
[00:01<00:01, 1761.09it/s]
 48%|███████████████████████████████████▉                                       | 2119/4422 [00:01
00:01, 1706.35it/s]
 53%|███████████████████████████████████████▌                                   | 2329/4422 [00:01
00:01, 1765.32it/s]
 57%|██████████████████████████████████████████▊                                | 2524/4422 [00:01
00:01, 1770.58it/s]
 62%|██████████████████████████████████████████████▎                            | 2730/4422 [00:01
00:00, 1803.02it/s]
 66%|█████████████████████████████████████████████████▋                         | 2930/4422
[00:01<00:00, 1798.37it/s]
 71%|████████████████████████████████████████████████████▉                      | 3118/4422
[00:01<00:00, 1773.79it/s]
 75%|████████████████████████████████████████████████████████▍                  | 3325/4422 [00:01
<00:00, 1807.86it/s]
 80%|████████████████████████████████████████████████████████████               | 3539/4422 [00:01
<00:00, 1850.09it/s]
 85%|███████████████████████████████████████████████████████████████▉           | 3767/4422 [00:02
<00:00, 1914.73it/s]
 90%|███████████████████████████████████████████████████████████████████▌       | 3983/4422
[00:02<00:00, 1932.42it/s]
 95%|███████████████████████████████████████████████████████████████████████▎   | 4202/4422
[00:02<00:00, 1952.91it/s]
100%|██████████████████████████████████████████████████████████████████████████▊| 4414/4422
[00:02<00:00, 1948.53it/s]
100%|███████████████████████████████████████████████████████████████████████████| 4422/4422
[00:02<00:00, 1835.83it/s]
```

[5.2.1] Applying KNN kd-tree on BOW, SET 5

In [49]:

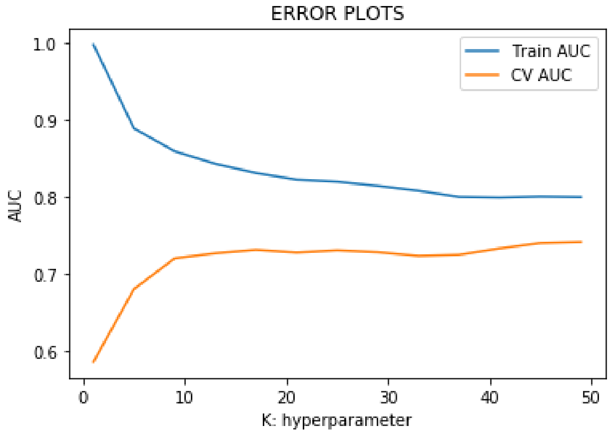
```
# Please write all the code with proper documentation
from sklearn.neighbors import KNeighborsClassifier
```

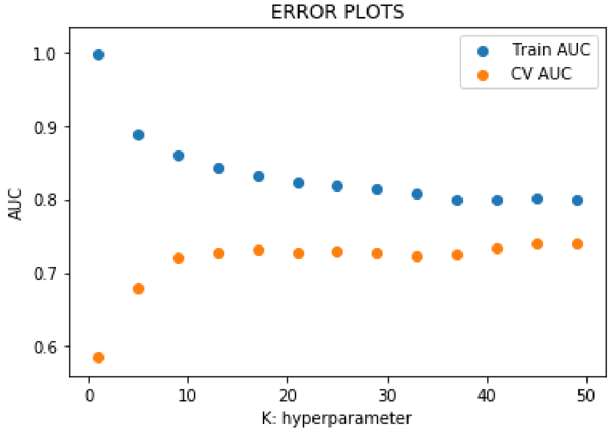
```python
from sklearn.neighbors import KNeighborsClassifier
from sklearn.metrics import accuracy_score
score = []
ind = []
train_auc = []
cv_auc = []
print(len(y_cv))
for i in tqdm(range(1,50,4)):
    knn = KNeighborsClassifier(n_neighbors=i,algorithm = 'kd_tree')
    knn.fit(X_train_vect, y_train)
    pred = knn.predict(X_cv_vect)
    acc = accuracy_score(y_cv, pred, normalize=True) * float(100)
    score.append(acc)
    ind.append(i)
    y_train_pred =  knn.predict_proba(X_train_vect)[:,1]
    y_cv_pred =  knn.predict_proba(X_cv_vect)[:,1]
    train_auc.append(roc_auc_score(y_train,y_train_pred))
    cv_auc.append(roc_auc_score(y_cv, y_cv_pred))
optimal_k_accuracy = ind[score.index(max(score))]
print('\nThe optimal number of neighbors is (according to accuracy): %d.' % optimal_k_accuracy)
optimal_k_auc = ind[cv_auc.index(max(cv_auc))]
print('\nThe optimal number of neighbors is (according to auc curve (max auc)): %d.' %
optimal_k_auc)

plt.plot(range(1,50,4), train_auc, label='Train AUC')
plt.plot(range(1,50,4), cv_auc, label='CV AUC')
plt.legend()
plt.xlabel("K: hyperparameter")
plt.ylabel("AUC")
plt.title("ERROR PLOTS")
plt.show()

plt.scatter(range(1,50,4), train_auc, label='Train AUC')
plt.scatter(range(1,50,4), cv_auc, label='CV AUC')
plt.legend()
plt.xlabel("K: hyperparameter")
plt.ylabel("AUC")
plt.title("ERROR PLOTS")
plt.show()
```

```
4422
```

```
  0%|                                                                                  | 0
[00:00<?, ?it/s]
  8%|██████                                                                     | 1/13 [01:4
1:45, 108.77s/it]
 15%|████████████                                                               | 2/13
[04:10<21:46, 118.75s/it]
 23%|███████████████████                                                        | 3/13
[06:33<20:59, 125.94s/it]
 31%|█████████████████████████                                                  | 4/13 [09:0
20:07, 134.12s/it]
 38%|████████████████████████████████                                           | 5/13
[11:38<18:35, 139.40s/it]
 46%|██████████████████████████████████████                                     | 6/13
[14:17<16:57, 145.37s/it]
 54%|████████████████████████████████████████████                               | 7/13 [16:4
<14:39, 146.60s/it]
 62%|███████████████████████████████████████████████████                        | 8/13 [19:2
<12:25, 149.17s/it]
 69%|█████████████████████████████████████████████████████████                  | 9/13
[22:01<10:09, 152.25s/it]
 77%|███████████████████████████████████████████████████████████████           | 10/13 [24:3
8<07:40, 153.46s/it]
 85%|█████████████████████████████████████████████████████████████████████     | 11/13 [27:1
2<05:07, 153.69s/it]
 92%|███████████████████████████████████████████████████████████████████████████| 12/13 [29:3
9<02:31, 151.70s/it]
100%|██████████████████████████████████████████████████████████████████████████████████| 13/13
[32:05<00:00, 149.93s/it]
```

```
The optimal number of neighbors is (according to accuracy): 13.

The optimal number of neighbors is (according to auc curve (max auc)): 49.
```

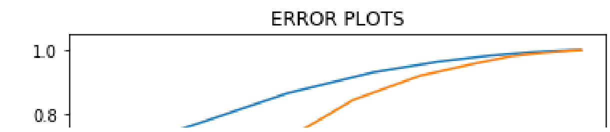


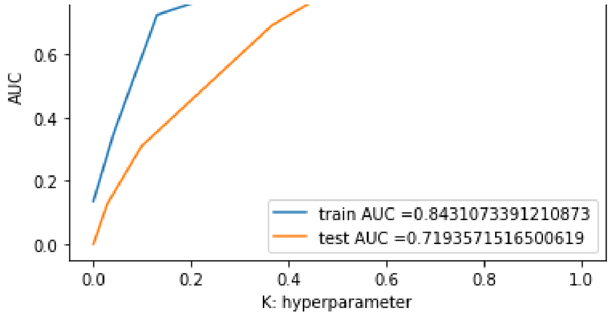


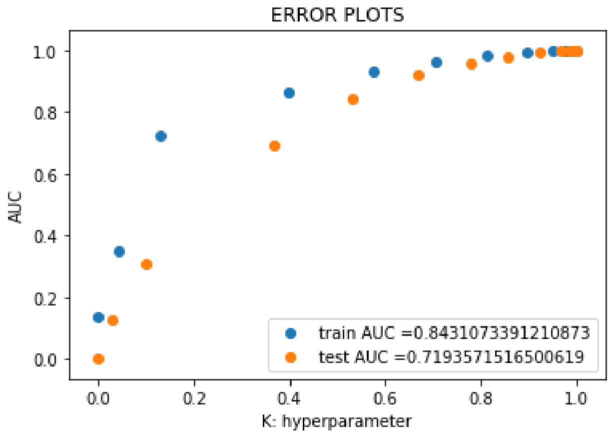
In [50]:

```
knn = KNeighborsClassifier(optimal_k_accuracy,algorithm = 'kd_tree')
knn.fit(X_train_vect,y_train)
pred = knn.predict(X_test_vect)
acc = accuracy_score(y_test, pred, normalize=True) * float(100)
print('\n****Test accuracy for k = %d is %f%%' % (optimal_k_accuracy,acc))

train_fpr, train_tpr, thresholds = roc_curve(y_train, knn.predict_proba(X_train_vect)[:,1])
test_fpr, test_tpr, thresholds = roc_curve(y_test, knn.predict_proba(X_test_vect)[:,1])

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

plt.scatter(train_fpr, train_tpr, label="train AUC ="+str(auc(train_fpr, train_tpr)))
plt.scatter(test_fpr, test_tpr, label="test AUC ="+str(auc(test_fpr, test_tpr)))
plt.legend()
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")
sns.heatmap(confusion_matrix(y_train, knn.predict(X_train_vect)))
print("Test confusion matrix")
sns.heatmap(confusion_matrix(y_test, knn.predict(X_test_vect)))
```

```
****Test accuracy for k = 13 is 84.606061%
```



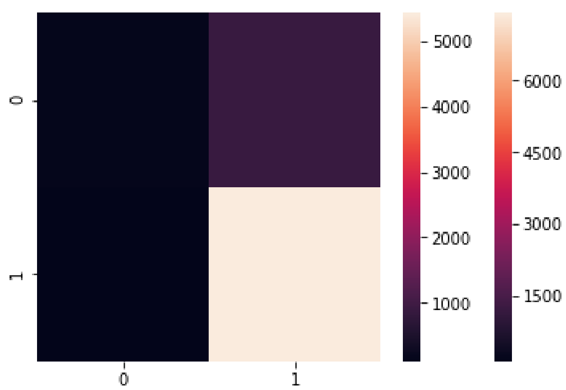





```
====================================================================================================

Train confusion matrix
Test confusion matrix
```

Out[50]:

```
<matplotlib.axes._subplots.AxesSubplot at 0x271701dfb70>
```



[5.2.2] Applying KNN kd-tree on TFIDF, SET 6

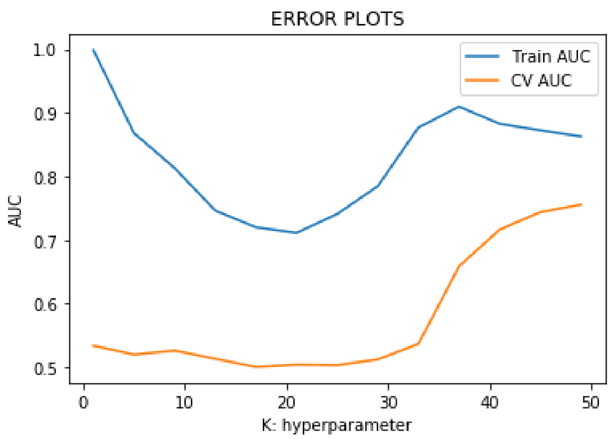
In [51]:

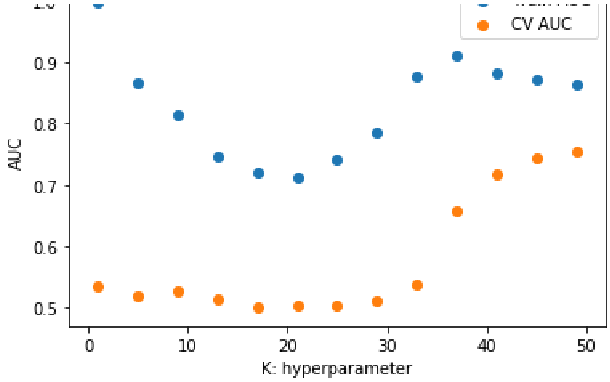
```python
# Please write all the code with proper documentation
score = []
ind = []
train_auc = []
cv_auc = []
for i in tqdm(range(1,50,4)):
    knn = KNeighborsClassifier(n_neighbors=i,algorithm = 'kd_tree')
    knn.fit(X_train_vect_tfidf, y_train)
    pred = knn.predict(X_cv_vect_tfidf)
    acc = accuracy_score(y_cv, pred, normalize=True) * float(100)
    score.append(acc)
    ind.append(i)
    y_train_pred =  knn.predict_proba(X_train_vect_tfidf)[:,1]
    y_cv_pred =  knn.predict_proba(X_cv_vect_tfidf)[:,1]
    train_auc.append(roc_auc_score(y_train,y_train_pred))
    cv_auc.append(roc_auc_score(y_cv, y_cv_pred))
```

```python
optimal_k_accuracy = ind[score.index(max(score))]
print('\nThe optimal number of neighbors is (according to accuracy): %d.' % optimal_k_accuracy)
optimal_k_auc = ind[cv_auc.index(max(cv_auc))]
print('\nThe optimal number of neighbors is (according to auc curve (max auc)): %d.' %
optimal_k_auc)


plt.plot(range(1,50,4), train_auc, label='Train AUC')
plt.plot(range(1,50,4), cv_auc, label='CV AUC')
plt.legend()
plt.xlabel("K: hyperparameter")
plt.ylabel("AUC")
plt.title("ERROR PLOTS")
plt.show()

plt.scatter(range(1,50,4), train_auc, label='Train AUC')
plt.scatter(range(1,50,4), cv_auc, label='CV AUC')
plt.legend()
plt.xlabel("K: hyperparameter")
plt.ylabel("AUC")
plt.title("ERROR PLOTS")
plt.show()
```

```
  0%|                                                                   | C
[00:00<?, ?it/s]
  8%|█████                                                   | 1/13 [01:
17:41, 88.47s/it]
 15%|█████████                                                | 2/13
[03:59<19:38, 107.15s/it]
 23%|██████████████                                           | 3/13
[06:28<19:57, 119.75s/it]
 31%|███████████████████                                      | 4/13 [08:5
19:18, 128.78s/it]
 38%|████████████████████████                                 | 5/13
[11:28<18:02, 135.25s/it]
 46%|█████████████████████████████                            | 6/13
[14:07<16:37, 142.48s/it]
 54%|██████████████████████████████████                       | 7/13 [16:4
<14:34, 145.70s/it]
 62%|███████████████████████████████████████                  | 8/13 [19:1
<12:14, 146.94s/it]
 69%|████████████████████████████████████████████             | 9/13
[21:40<09:51, 147.84s/it]
 77%|█████████████████████████████████████████████████        | 10/13 [24:1
0<07:25, 148.52s/it]
 85%|██████████████████████████████████████████████████████   | 11/13 [26:4
0<04:57, 148.92s/it]
 92%|███████████████████████████████████████████████████████████ | 12/13 [29:1
1<02:29, 149.40s/it]
100%|████████████████████████████████████████████████████████████████| 13/13
[31:41<00:00, 149.65s/it]
```

```
The optimal number of neighbors is (according to accuracy): 13.

The optimal number of neighbors is (according to auc curve (max auc)): 49.
```

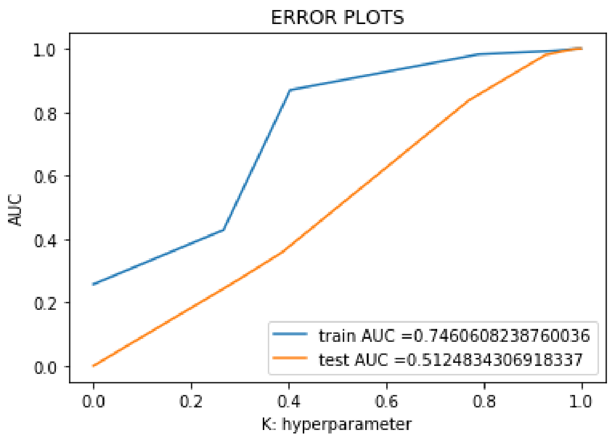


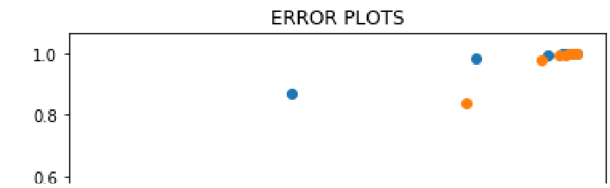


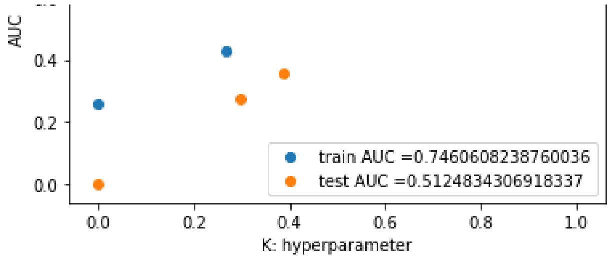
In [52]:

```
knn = KNeighborsClassifier(optimal_k_accuracy,algorithm = 'kd_tree')
knn.fit(X_train_vect_tfidf,y_train)
pred = knn.predict(X_test_vect_tfidf)
acc = accuracy_score(y_test, pred, normalize=True) * float(100)
print('\n****Test accuracy for k = %d is %f%%' % (optimal_k_accuracy,acc))

train_fpr, train_tpr, thresholds = roc_curve(y_train, knn.predict_proba(X_train_vect_tfidf)[:,1])
test_fpr, test_tpr, thresholds = roc_curve(y_test, knn.predict_proba(X_test_vect_tfidf)[:,1])

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

plt.scatter(train_fpr, train_tpr, label="train AUC ="+str(auc(train_fpr, train_tpr)))
plt.scatter(test_fpr, test_tpr, label="test AUC ="+str(auc(test_fpr, test_tpr)))
plt.legend()
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")
sns.heatmap(confusion_matrix(y_train, knn.predict(X_train_vect_tfidf)))
print("Test confusion matrix")
sns.heatmap(confusion_matrix(y_test, knn.predict(X_test_vect_tfidf)))
```

```
****Test accuracy for k = 13 is 84.045455%
```



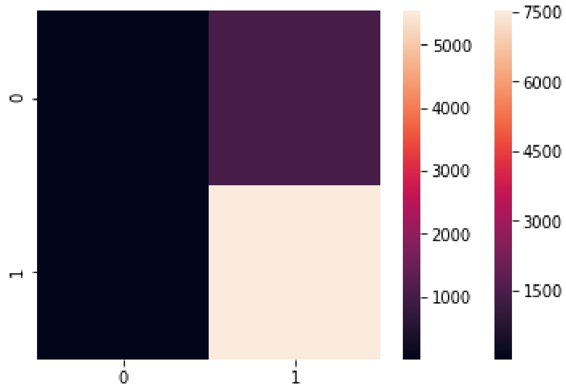





```
====================================================================================================

Train confusion matrix
Test confusion matrix
```

Out[52]:

```
<matplotlib.axes._subplots.AxesSubplot at 0x27171bf1f98>
```



[5.2.3] Applying KNN kd-tree on AVG W2V, SET 3

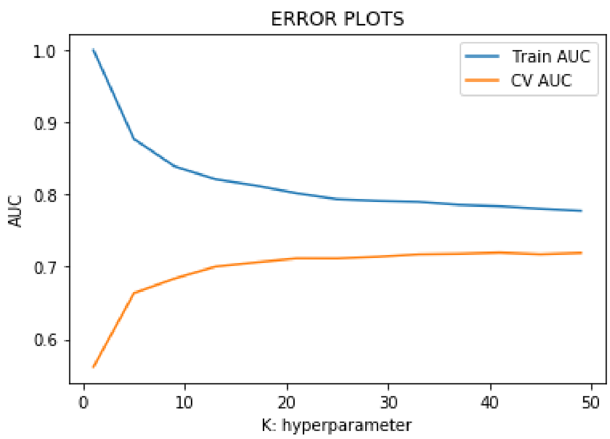
In [53]:

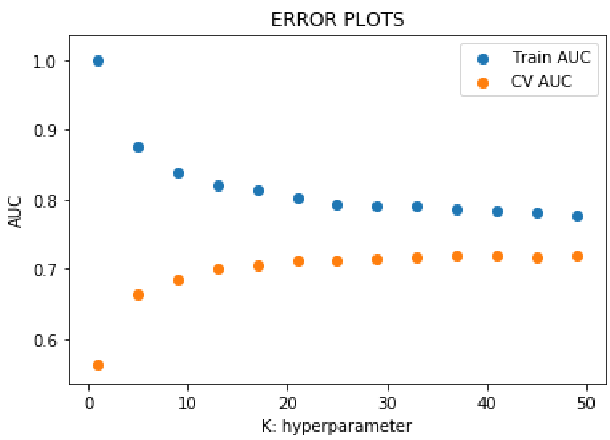
```
# Please write all the code with proper documentation
score = []
ind = []
train_auc = []
cv_auc = []
for i in tqdm(range(1,50,4)):
    knn = KNeighborsClassifier(n_neighbors=i,algorithm = 'kd_tree')
    knn.fit(X_train_vectors, y_train)
    pred = knn.predict(X_cv_vectors)
    acc = accuracy_score(y_cv, pred, normalize=True) * float(100)
    score.append(acc)
    ind.append(i)
    y_train_pred =  knn.predict_proba(X_train_vectors)[:,1]
    y_cv_pred =  knn.predict_proba(X_cv_vectors)[:,1]
    train_auc.append(roc_auc_score(y_train,y_train_pred))
    cv_auc.append(roc_auc_score(y_cv, y_cv_pred))
optimal_k_accuracy = ind[score.index(max(score))]
print('\nThe optimal number of neighbors is (according to accuracy): %d.' % optimal_k_accuracy)
optimal_k_auc = ind[cv_auc.index(max(cv_auc))]
print('\nThe optimal number of neighbors is (according to auc curve (max auc)): %d.' %
optimal_k_auc)


plt.plot(range(1,50,4), train_auc, label='Train AUC')
plt.plot(range(1,50,4), cv_auc, label='CV AUC')
plt.legend()
plt.xlabel("K: hyperparameter")
plt.ylabel("AUC")
plt.title("ERROR PLOTS")
plt.show()

plt.scatter(range(1,50,4), train_auc, label='Train AUC')
plt.scatter(range(1,50,4), cv_auc, label='CV AUC')
plt.legend()
plt.xlabel("K: hyperparameter")
plt.ylabel("AUC")
```

```
plt.title("ERROR PLOTS")
plt.show()
```

```
  0%|                                                                                        | C
[00:00<?, ?it/s]
  8%|██████                                                                        | 1/13 [00:
00:30,  2.53s/it]
 15%|█████████████                                                                 | 2/13
[00:08<00:39,  3.60s/it]
 23%|███████████████████                                                           | 3/13 [00:
<00:45,  4.56s/it]
 31%|██████████████████████████                                                    | 4/13 [00:
<00:49,  5.47s/it]
 38%|████████████████████████████████                                              | 5/13
[00:30<00:49,  6.20s/it]
 46%|███████████████████████████████████████                                       | 6/13
[00:38<00:47,  6.76s/it]
 54%|█████████████████████████████████████████████                                 | 7/13 [00:
8<00:45,  7.51s/it]
 62%|████████████████████████████████████████████████████                          | 8/13 [00:
7<00:39,  7.94s/it]
 69%|██████████████████████████████████████████████████████████                    | 9/13
[01:06<00:33,  8.30s/it]
 77%|████████████████████████████████████████████████████████████████             | 10/13 [01:
15<00:25,  8.59s/it]
 85%|███████████████████████████████████████████████████████████████████████      | 11/13 [01:
24<00:17,  8.79s/it]
 92%|█████████████████████████████████████████████████████████████████████████████ | 12/13
[01:34<00:08,  9.00s/it]
100%|████████████████████████████████████████████████████████████████████████████████████| 13/13
[01:44<00:00,  9.21s/it]
```

```
The optimal number of neighbors is (according to accuracy): 17.

The optimal number of neighbors is (according to auc curve (max auc)): 41.
```



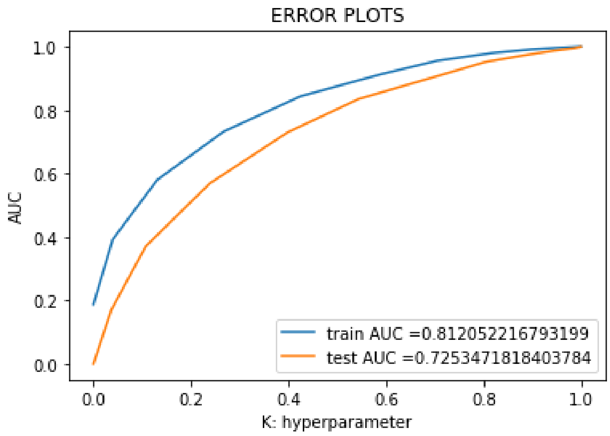


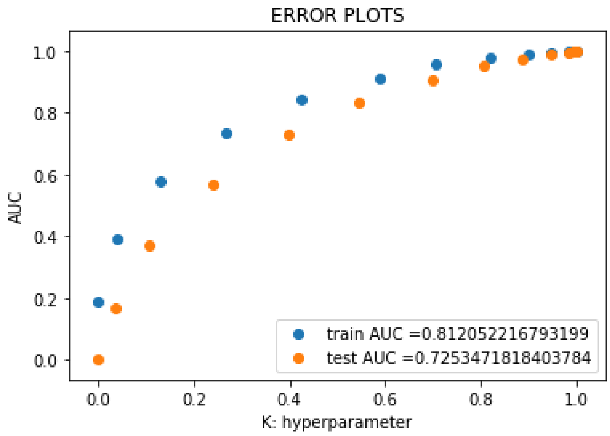
In [54]:

```
knn = KNeighborsClassifier(optimal_k_accuracy,algorithm = 'kd_tree')
knn.fit(X_train_vectors,y_train)
pred = knn.predict(X_test_vectors)
```

```
acc = accuracy_score(y_test, pred, normalize=True) * float(100)
print('\n****Test accuracy for k = %d is %f%%' % (optimal_k_accuracy,acc))


train_fpr, train_tpr, thresholds = roc_curve(y_train, knn.predict_proba(X_train_vectors)[:,1])
test_fpr, test_tpr, thresholds = roc_curve(y_test, knn.predict_proba(X_test_vectors)[:,1])

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

plt.scatter(train_fpr, train_tpr, label="train AUC ="+str(auc(train_fpr, train_tpr)))
plt.scatter(test_fpr, test_tpr, label="test AUC ="+str(auc(test_fpr, test_tpr)))
plt.legend()
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")
sns.heatmap(confusion_matrix(y_train, knn.predict(X_train_vectors)))
print("Test confusion matrix")
sns.heatmap(confusion_matrix(y_test, knn.predict(X_test_vectors)))
```

```
****Test accuracy for k = 17 is 83.757576%
```

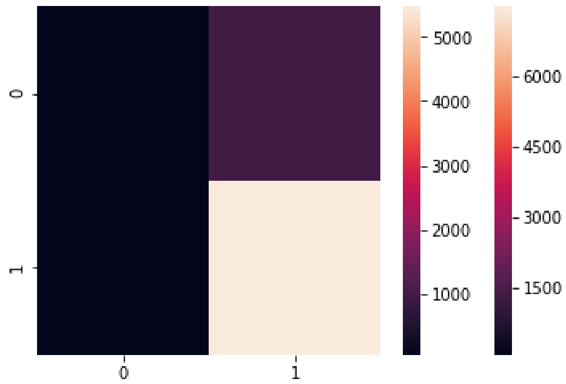





```
====================================================================================================

Train confusion matrix
Test confusion matrix
```

Out[54]:

```
<matplotlib.axes._subplots.AxesSubplot at 0x27172e05898>
```



[5.2.4] Applying KNN kd-tree on TFIDF W2V, SET 4

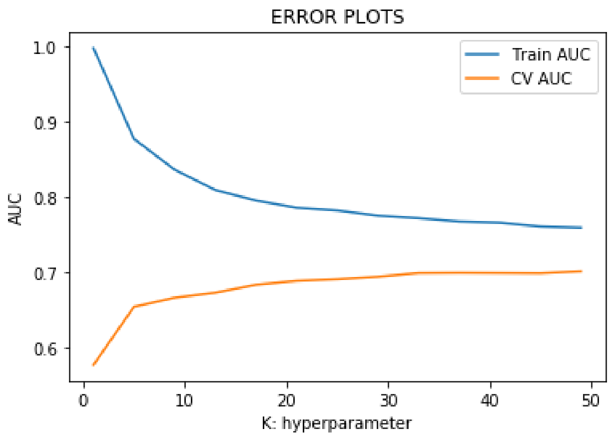
In [55]:

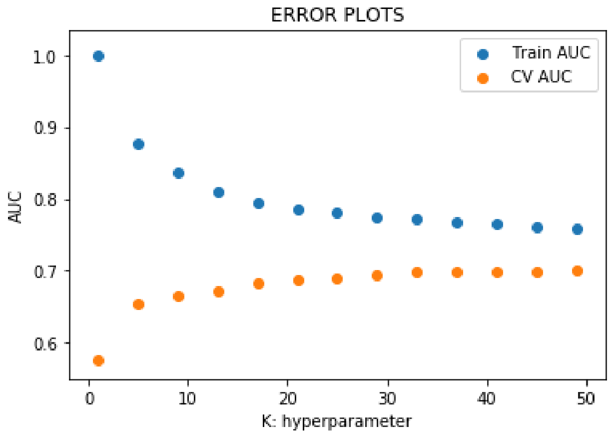
```
# Please write all the code with proper documentation
# Please write all the code with proper documentation
score = []
ind = []
train_auc = []
cv_auc = []
for i in tqdm(range(1,50,4)):
    knn = KNeighborsClassifier(n_neighbors=i,algorithm = 'kd_tree')
    knn.fit(tfidf_X_train_vectors, y_train)
    pred = knn.predict(tfidf_X_cv_vectors)
    acc = accuracy_score(y_cv, pred, normalize=True) * float(100)
    score.append(acc)
    ind.append(i)
    y_train_pred =  knn.predict_proba(tfidf_X_train_vectors)[:,1]
    y_cv_pred =  knn.predict_proba(tfidf_X_cv_vectors)[:,1]
    train_auc.append(roc_auc_score(y_train,y_train_pred))
    cv_auc.append(roc_auc_score(y_cv, y_cv_pred))
optimal_k_accuracy = ind[score.index(max(score))]
print('\nThe optimal number of neighbors is (according to accuracy): %d.' % optimal_k_accuracy)
optimal_k_auc = ind[cv_auc.index(max(cv_auc))]
print('\nThe optimal number of neighbors is (according to auc curve (max auc)): %d.' %
optimal_k_auc)

plt.plot(range(1,50,4), train_auc, label='Train AUC')
plt.plot(range(1,50,4), cv_auc, label='CV AUC')
plt.legend()
plt.xlabel("K: hyperparameter")
plt.ylabel("AUC")
plt.title("ERROR PLOTS")
plt.show()

plt.scatter(range(1,50,4), train_auc, label='Train AUC')
plt.scatter(range(1,50,4), cv_auc, label='CV AUC')
plt.legend()
plt.xlabel("K: hyperparameter")
plt.ylabel("AUC")
plt.title("ERROR PLOTS")
plt.show()
```

```
  0%|                                                                      | C
[00:00<?, ?it/s]
  8%|██████                                                      | 1/13 [00:
00:28,  2.39s/it]
 15%|███████████                                                 | 2/13
[00:08<00:37,  3.40s/it]
 23%|██████████████████                                          | 3/13 [00:
<00:44,  4.41s/it]
 31%|████████████████████████                                    | 4/13 [00:
<00:46,  5.18s/it]
 38%|██████████████████████████████                              | 5/13
[00:30<00:48,  6.11s/it]
 46%|████████████████████████████████████                        | 6/13
[00:37<00:45,  6.54s/it]
 54%|███████████████████████████████████████████                 | 7/13 [00:
6<00:42,  7.07s/it]
```

```
6<00:42,  7.07s/it]
 62%|                                                        | 8/13 [00:
4<00:37,  7.43s/it]
 69%|                                                        | 9/13
[01:02<00:30,  7.60s/it]
 77%|                                                        | 10/13 [01:
11<00:24,  8.06s/it]
 85%|                                                        | 11/13 [01:
20<00:16,  8.37s/it]
 92%|                                                        | 12/13
[01:29<00:08,  8.53s/it]
100%|                                                        | 13/13
[01:39<00:00,  8.88s/it]
```

```
The optimal number of neighbors is (according to accuracy): 49.

The optimal number of neighbors is (according to auc curve (max auc)): 49.
```



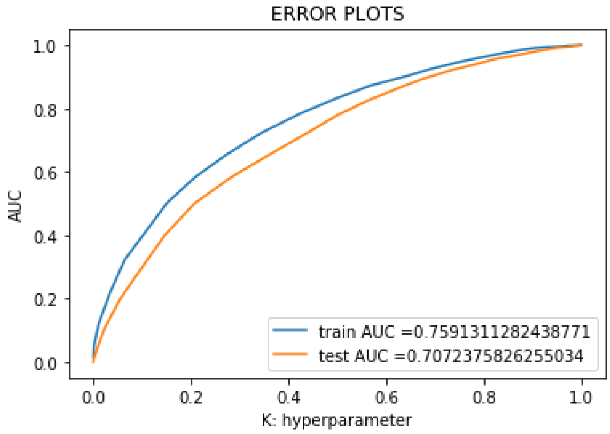


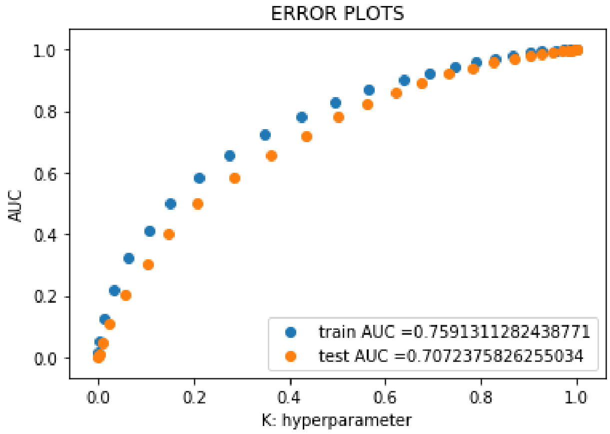
In [56]:

```
knn = KNeighborsClassifier(optimal_k_accuracy,algorithm = 'kd_tree')
knn.fit(tfidf_X_train_vectors,y_train)
pred = knn.predict(tfidf_X_test_vectors)
acc = accuracy_score(y_test, pred, normalize=True) * float(100)
print('\n****Test accuracy for k = %d is %f%%' % (optimal_k_accuracy,acc))

train_fpr, train_tpr, thresholds = roc_curve(y_train, knn.predict_proba(tfidf_X_train_vectors)[:,1]
)
test_fpr, test_tpr, thresholds = roc_curve(y_test, knn.predict_proba(tfidf_X_test_vectors)[:,1])

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

plt.scatter(train_fpr, train_tpr, label="train AUC ="+str(auc(train_fpr, train_tpr)))
plt.scatter(test_fpr, test_tpr, label="test AUC ="+str(auc(test_fpr, test_tpr)))
plt.legend()
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")
sns.heatmap(confusion_matrix(y_train, knn.predict(tfidf_X_train_vectors)))
print("Test confusion matrix")
sns.heatmap(confusion_matrix(y_test, knn.predict(tfidf_X_test_vectors)))
```

```
****Test accuracy for k = 49 is 83.909091%
```

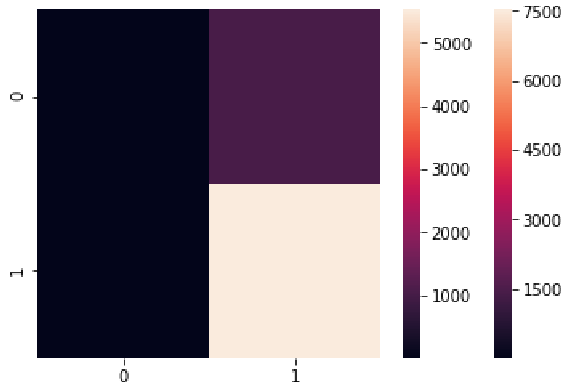





```
====================================================================================================

Train confusion matrix
Test confusion matrix
```

Out[56]:

```
<matplotlib.axes._subplots.AxesSubplot at 0x27171c6b0b8>
```



[6] Conclusions

In [57]:

```
# Please compare all your models using Prettytable library
from prettytable import PrettyTable
x = PrettyTable()
x.field_names = ["Vectorizer", "Model", "Hyperparameter", "AUC"]
x.add_row(["BOW", "BRUTE", 49, 0.7609])
x.add_row(["TFIDF", "BRUTE", 17, 0.567])
x.add_row(["W2V", "BRUTE", 25, 0.8471])
x.add_row(["TFIDFW2V", "BRUTE", 13, 0.7979])
x.add_row(["BOW", "KD_TREE", 13, 0.719])
x.add_row(["TFIDF", "KD_TREE", 13, 0.512])
x.add_row(["W2V", "KD_TREE", 17, 0.7253])
x.add_row(["TFIDFW2V", "KD_TREE", 49, 0.707])
print(x)
```

```
+------------+---------+----------------+--------+
| Vectorizer |  Model  | Hyperparameter |  AUC   |
+------------+---------+----------------+--------+
|    BOW     |  BRUTE  |       49       | 0.7609 |
|   TFIDF    |  BRUTE  |       17       | 0.567  |
|    W2V     |  BRUTE  |       25       | 0.8471 |
|  TFIDFW2V  |  BRUTE  |       13       | 0.7979 |
|    BOW     | KD_TREE |       13       | 0.719  |
|   TFIDF    | KD_TREE |       13       | 0.512  |
|    W2V     | KD_TREE |       17       | 0.7253 |
|  TFIDFW2V  | KD_TREE |       49       | 0.707  |
+------------+---------+----------------+--------+
```

- The best algorithm of the above is word2vec using bag of words
- Tfidf tend to perform bad in both brute-force and kd-tree
- The data set is surely imbalanced
- Feature engineering by adding the no. of words in the review as a feature tend to improve the results