[1]. Reading Data

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.

[1.1] 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 CSV dataset

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

In [1]:
```
%matplotlib inline
import warnings
warnings.filterwarnings("ignore")


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

from sklearn.metrics import accuracy_score
import seaborn as sn
from sklearn.metrics import classification_report

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

from sklearn.neighbors import KNeighborsClassifier
from sklearn.metrics import roc_auc_score
import matplotlib.pyplot as plt
from sklearn.model_selection import validation_curve

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

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

from tqdm import tqdm
import os
```

```
C:\Users\Adarsh\Anaconda3\lib\site-packages\gensim\utils.py:1212: UserWarnin
g: detected Windows; aliasing chunkize to chunkize_serial
  warnings.warn("detected Windows; aliasing chunkize to chunkize_serial")
```

In [2]:
```
#loading train and test and validation dataset

file=open("x_train.pkl","rb")
x_train=pickle.load(file) # loading 'train' dataset

file=open("x_cv.pkl",'rb')
x_cv=pickle.load(file) # loading 'validation' dataset
```

```
file=open("x_test.pkl",'rb')
x_test=pickle.load(file) # loading 'test' dataset

file=open("y_train.pkl","rb")
y_train=pickle.load(file) # loading 'train' dataset

file=open("y_cv.pkl",'rb')
y_cv=pickle.load(file) # loading 'validation' dataset

file=open("y_test.pkl",'rb')
y_test=pickle.load(file) # loading 'test' dataset

#loading train_bow and test _bow
file=open('x_train_bow.pkl','rb')
x_train_bow=pickle.load(file)

file=open('x_test_bow.pkl','rb')
x_test_bow=pickle.load(file)

file=open('x_cv_bow.pkl','rb')
x_cv_bow=pickle.load(file)

#loading train_tf_idf and test_tf_idf
file=open('train_tf_idf.pkl','rb')
train_tf_idf=pickle.load(file)

file=open('cv_tf_idf.pkl','rb')
cv_tf_idf=pickle.load(file)

file=open('test_tf_idf.pkl','rb')
test_tf_idf=pickle.load(file)

#loading train_w2v and test_w2v
file=open('train_w2v.pkl','rb')
train_w2v=pickle.load(file)

file=open('cv_w2v.pkl','rb')
cv_w2v=pickle.load(file)

file=open('test_w2v.pkl','rb')
test_w2v=pickle.load(file)

#loading train_tf_idf_w2v and test_tf_idf_w2v
file=open('train_tf_idf_w2v.pkl','rb')
train_tf_idf_w2v=pickle.load(file)

file=open('cv_tf_idf_w2v.pkl','rb')
cv_tf_idf_w2v=pickle.load(file)

file=open('test_tf_idf_w2v.pkl','rb')
test_tf_idf_w2v=pickle.load(file)
```

In [2]:

```
# using csv Table to read data.

dataset=pd.read_csv("Reviews.csv")

print(dataset.shape)
dataset.head(3)
```

```
(568454, 10)
```

Out[2]:

	Id	ProductId	UserId	ProfileName	HelpfulnessNumerator	HelpfulnessDenominator	S
0	1	B001E4KFG0	A3SGXH7AUHU8GW	delmartian	1	1	
1	2	B00813GRG4	A1D87F6ZCVE5NK	dll pa	0	0	
2	3	B000LQOCH0	ABXLMWJIXXAIN	Natalia Corres "Natalia Corres"	1	1	

In [3]:

```
# 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 da
ta points
# you can change the number to any other number based on your computing power

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

# taking reviews whose score is not equal to 3
filtered_dataset=dataset[dataset['Score']!=3]
filtered_dataset.shape

#creating a function to filter the reviews (if score>3 --> positive , if score
<3 --> negative)
def partition(x):
    if x>3:
        return 1
    return 0

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

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

Out[3]:

	Id	ProductId	UserId	ProfileName	HelpfulnessNumerator	HelpfulnessDenominator	S
0	1	B001E4KFG0	A3SGXH7AUHU8GW	delmartian	1	1	

1	2	B00813GRG4	A1D87F6ZCVE5NK	dll pa	0	0
2	3	B000LQOCH0	ABXLMWJIXXAIN	Natalia Corres "Natalia Corres"	1	1

[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:

observation:

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 [4]:

```
# sorting the value
sorted_data=filtered_dataset.sort_values(by='Id',inplace=True )
#finding the dublicate values using 'df.dublicated'
filtered_dataset[filtered_dataset.duplicated(subset={'ProfileName','HelpfulnessNumerator','HelpfulnessDenominator','Score','Time'})].shape
#alternate way to drop dublicate values
dataset_no_dup=filtered_dataset.drop_duplicates(subset={'ProfileName','Score','Time','Summary'},keep='first')
print(f"before {dataset.shape}")
print(f"after removing duplicate values-->shape = {dataset_no_dup.shape}")
# %age of no. of review reamin in data set
print('percentage of data reamin after removing duplicate values and removing reviews with neutral scores % .2f'
      %((dataset_no_dup.size/dataset.size)*100))
```

```
before (568454, 10)
after removing duplicate values-->shape = (363255, 10)
percentage of data reamin after removing duplicate values and removing revie
ws with neutral scores  63.90
```

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 [5]:
```
# removing reviews where "HelpfulnessNumerator>HelpfulnessDenominator"
final=dataset_no_dup[dataset_no_dup['HelpfulnessNumerator']<=dataset_no_dup['H
elpfulnessDenominator']]
```

In [6]:
```
#Before starting the next phase of preprocessing lets see the number of entrie
s left
print(final.shape)

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

```
(363253, 10)
```

Out[6]:
```
1    306222
0     57031
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 [7]:
```
# https://stackoverflow.com/questions/16206380/python-beautifulsoup-how-to-rem
ove-all-tags-from-an-element
from bs4 import BeautifulSoup

# 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


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

stopwords= set(['br', 'the', 'i', 'me', 'my', 'myself', 'we', 'our', 'ours',
'ourselves', 'you', "you'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', 'it
self', 'they', 'them', 'their',\
            'theirs', 'themselves', 'what', 'which', 'who', 'whom', 'this', 't
hat', "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', 'becau
se', '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', 'a
ll', 'any', 'both', 'each', 'few', 'more',\
            'most', 'other', 'some', 'such', 'only', 'own', 'same', 'so', 'tha
n', 'too', 'very', \
            's', 't', 'can', 'will', 'just', 'don', "don't", 'should', "shoul
d've", 'now', 'd', 'll', 'm', 'o', 're', \
            've', 'y', 'ain', 'aren', "aren't", 'couldn', "couldn't", 'didn',
"didn't", 'doesn', "doesn't", 'hadn',\
            "hadn't", 'hasn', "hasn't", 'haven', "haven't", 'isn', "isn't", 'm
a', 'mightn', "mightn't", 'mustn',\
            "mustn't", 'needn', "needn't", 'shan', "shan't", 'shouldn', "shoul
dn't", 'wasn', "wasn't", 'weren', "weren't", \
            'won', "won't", 'wouldn', "wouldn't"])
```

In [8]:

```
# 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 i
n stopwords)
    preprocessed_reviews.append(sentance.strip())
```

```
100%|██████████████████████████████████████████████████
█| 363253/363253 [02:53<00:00, 2096.04it/s]
```

In [9]:
```
preprocessed_reviews[:5]
```

Out[9]:
```
['bought several vitality canned dog food products found good quality produc
t looks like stew processed meat smells better labrador finicky appreciates
product better',
 'product arrived labeled jumbo salted peanuts peanuts actually small sized
unsalted not sure error vendor intended represent product jumbo',
 'confection around centuries light pillowy citrus gelatin nuts case filbert
s cut tiny squares liberally coated powdered sugar tiny mouthful heaven not
chewy flavorful highly recommend yummy treat familiar story c lewis lion wit
ch wardrobe treat seduces edmund selling brother sisters witch',
 'looking secret ingredient robitussin believe found got addition root beer
extract ordered good made cherry soda flavor medicinal',
 'great taffy great price wide assortment yummy taffy delivery quick taffy l
over deal']
```

In [10]:
```
final['preprocessed_reviews']=preprocessed_reviews
```

In [11]:
```
# splitting the dataset in train , cv and test

n=final.shape[0] #size of final dataset

train=final.iloc[:round(0.60*n),:]
cv=final.iloc[round(0.60*n):round(0.80*n),:]
test=final.iloc[round(0.80*n):round(1.0*n),:]
```

In [12]:
```
from sklearn.model_selection import train_test_split
x_training,x_test,y_training,y_test=train_test_split(preprocessed_reviews,fina
l["Score"] ,test_size=0.20, random_state=42)
x_train,x_cv,y_train,y_cv=train_test_split(x_training,y_training, test_size=0.
25, random_state=42)
```

In [13]:
```
len(x_train),len(y_train),len(x_test),len(y_test),len(x_cv),len(y_cv)
```

Out[13]:
```
(217951, 217951, 72651, 72651, 72651, 72651)
```

In [14]:
```
# saving train and test dataset using pickle for fututre use

'''file=open("x_train.pkl","wb")
pickle.dump(x_train,file)
file.close()

file=open('x_cv.pkl','wb')
pickle.dump(x_cv,file)
file.close

file=open("x_test.pkl",'wb')
pickle.dump(x_test,file)
file.close()
```

```
file=open("y_train.pkl","wb")
pickle.dump(y_train,file)
file.close()

file=open('y_cv.pkl','wb')
pickle.dump(y_cv,file)
file.close

file=open("y_test.pkl",'wb')
pickle.dump(y_test,file)
file.close()

'''
```

In [2]:

```
#loading train and test and validation dataset

file=open("x_train.pkl","rb")
x_train=pickle.load(file) # loading 'train' dataset

file=open("x_cv.pkl",'rb')
x_cv=pickle.load(file) # loading 'validation' dataset

file=open("x_test.pkl",'rb')
x_test=pickle.load(file) # loading 'test' dataset

file=open("y_train.pkl","rb")
y_train=pickle.load(file) # loading 'train' dataset

file=open("y_cv.pkl",'rb')
y_cv=pickle.load(file) # loading 'validation' dataset

file=open("y_test.pkl",'rb')
y_test=pickle.load(file) # loading 'test' dataset
```

[4] Featurization

[4.1] BAG OF WORDS

In [15]:

```
#BoW

count_vect = CountVectorizer() #in scikit-learn

x_train_bow=count_vect.fit_transform(x_train)
print("some feature names ", count_vect.get_feature_names()[:10])
print('='*100)

# transform cv and test dataset
x_cv_bow=count_vect.transform(x_cv)
x_test_bow=count_vect.transform(x_test)
```

```
some feature names  ['aa', 'aaa', 'aaaa', 'aaaaa', 'aaaaaa', 'aaaaaaaaaaaa',
'aaaaaaaaaaaaa', 'aaaaaaaaaaaaaa', 'aaaaaaaaaaaaaaa', 'aaaaaaaaaaaaaaaaaaaaa
aaaaaaaaaaaaaaaaaaaaaa']


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

=========================

In [16]:
```
x_train_bow.shape,x_test_bow.shape,x_cv_bow.shape
```

Out[16]:
```
((217951, 90000), (72651, 90000), (72651, 90000))
```

In [17]:
```
# saving train_bow and test_bow dataset using pickle for future use

'''file=open("x_train_bow.pkl","wb")
pickle.dump(x_train_bow,file)
file.close()

file=open("x_test_bow.pkl",'wb')
pickle.dump(x_test_bow,file)
file.close()

file=open("x_cv_bow.pkl",'wb')
pickle.dump(x_cv_bow,file)
file.close()
'''
```

In [3]:
```
#loading train_bow and test _bow
file=open('x_train_bow.pkl','rb')
x_train_bow=pickle.load(file)

file=open('x_test_bow.pkl','rb')
x_test_bow=pickle.load(file)

file=open('x_cv_bow.pkl','rb')
x_cv_bow=pickle.load(file)
```

[4.3] TF-IDF

In [19]:
```
# tf-idf "from sklearn.feature_extraction.text.TfidfVectorizer"
tf_idf=TfidfVectorizer()

train_tf_idf=tf_idf.fit_transform(x_train)
cv_tf_idf=tf_idf.transform(x_cv)
test_tf_idf=tf_idf.transform(x_test)
```

In [20]:
```
from sklearn.preprocessing import StandardScaler

sc=StandardScaler(with_mean=False)

train_tf_idf=sc.fit_transform(train_tf_idf)
cv_tf_idf=sc.transform(cv_tf_idf)
test_tf_idf=sc.transform(test_tf_idf)
```

In [21]:
```
# saving train_tf_idf and test_tf_idf dataset using pickle for fututre use

'''file=open("train_tf_idf.pkl","wb")
pickle.dump(train_tf_idf,file)
file.close()

file=open("cv_tf_idf.pkl",'wb')
pickle.dump(cv_tf_idf,file)
file.close()
```

```
file=open("test_tf_idf.pkl",'wb')
pickle.dump(test_tf_idf,file)
file.close()

'''
```

In [4]:
```
#loading train_tf_idf and test_tf_idf
file=open('train_tf_idf.pkl','rb')
train_tf_idf=pickle.load(file)

file=open('cv_tf_idf.pkl','rb')
cv_tf_idf=pickle.load(file)

file=open('test_tf_idf.pkl','rb')
test_tf_idf=pickle.load(file)
```

[4.4] Word2Vec

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

[4.4.1.1] Avg W2v

In [6]:
```
# converting our text-->vector using w2v with 50-dim
# more the dimension of each word = better the semantic of word
# using lib from "gensim.models.Word2Vec"
# to run w2v we need list of list of the words as w2v covert each world into n
umber of dim


# for train_w2v
list_of_sent_train=[]
for sent in x_train:
    list_of_sent_train.append((str(sent)).split())
w2v_model=Word2Vec(list_of_sent_train,min_count=5,size=50)

# vocablary of w2v model of amazon dataset
vocab=w2v_model.wv.vocab
len(vocab)

#------------------------------------------------------------------------------
--------------------

# for test_w2v
list_of_sent_cv=[]
for sent in x_cv:
    list_of_sent_cv.append((str(sent)).split())

#------------------------------------------------------------------------------
--------------------

# for test_w2v
list_of_sent_test=[]
for sent in x_test:
    list_of_sent_test.append((str(sent)).split())
```

In [7]:

```
'''
    -->procedure to make avg w2v of each reviews
    1. find the w2v of each word
    2. sum-up w2v of each word in a sentence
    3. divide the total w2v of sentence by total no. of words in the sentence
'''

# average Word2Vec
# compute average word2vec for each review.
train_w2v = []; # the avg-w2v for each sentence/review in train dataset is sto
red in this list

for sent in list_of_sent_train: # for each review/sentence
    sent_vec = np.zeros(50) # as word vectors are of zero length
    cnt_words =0; # num of words with a valid vector in the sentence/review
    for word in sent: # for each word in a review/sentence
        if word in vocab:
            vec = w2v_model.wv[word]
            sent_vec += vec
            cnt_words += 1
    if cnt_words != 0:
        sent_vec /= cnt_words
    train_w2v.append(sent_vec)

print(len(train_w2v))

#---------------------------------------------------------------------------
------

cv_w2v = []; # the avg-w2v for each sentence/review in test dataset is stored
 in this list

for sent in list_of_sent_cv: # for each review/sentence
    sent_vec = np.zeros(50) # as word vectors are of zero length
    cnt_words =0; # num of words with a valid vector in the sentence/review
    for word in sent: # for each word in a review/sentence
        if word in vocab:
            vec = w2v_model.wv[word]
            sent_vec += vec
            cnt_words += 1
    if cnt_words != 0:
        sent_vec /= cnt_words
    cv_w2v.append(sent_vec)

print(len(cv_w2v))


#---------------------------------------------------------------------------
------

test_w2v = []; # the avg-w2v for each sentence/review in test dataset is store
d in this list

for sent in list_of_sent_test: # for each review/sentence
    sent_vec = np.zeros(50) # as word vectors are of zero length
    cnt_words =0; # num of words with a valid vector in the sentence/review
    for word in sent: # for each word in a review/sentence
        if word in vocab:
            vec = w2v_model.wv[word]
            sent_vec += vec
            cnt_words += 1
```

```
    if cnt_words != 0:
        sent_vec /= cnt_words
    test_w2v.append(sent_vec)

print(len(test_w2v))
```

```
217951
72651
72651
```

In [9]:
```
from sklearn.preprocessing import StandardScaler
sc=StandardScaler(with_mean=True)

train_w2v=sc.fit_transform(train_w2v)
cv_w2v=sc.transform(cv_w2v)
test_w2v=sc.transform(test_w2v)
```

In [10]:
```
# saving train_w2v and test_w2v dataset using pickle for fututre use

'''file=open("train_w2v.pkl","wb")
pickle.dump(train_w2v,file)
file.close()

file=open("cv_w2v.pkl",'wb')
pickle.dump(cv_w2v,file)
file.close()

file=open("test_w2v.pkl",'wb')
pickle.dump(test_w2v,file)
file.close()
'''
```

In [5]:
```
#loading train_w2v and test_w2v
file=open('train_w2v.pkl','rb')
train_w2v=pickle.load(file)

file=open('cv_w2v.pkl','rb')
cv_w2v=pickle.load(file)

file=open('test_w2v.pkl','rb')
test_w2v=pickle.load(file)
```

In [21]:
```
train_w2v[0]
```

Out[21]:
```
array([-0.0080363 , -0.55650226, -2.24174777,  1.02574886,  0.12327979,
       -0.19916425, -1.06522974,  0.89144715, -1.13231167,  2.54008377,
        0.8032532 ,  0.3404576 ,  1.6792167 , -0.98081078,  1.08851643,
       -0.72007858, -0.65714762, -0.56007184,  0.01985994,  2.12137305,
       -0.09203752, -0.23671867, -1.63326771,  1.04496922,  0.45004579,
        0.3219116 ,  0.78335079,  0.54301334, -2.4968575 ,  0.35478244,
        1.46397278, -0.01982212, -0.1817636 ,  1.35729521, -0.61338792,
       -1.68822842, -0.84256537, -0.59978494,  0.40587478, -0.49775708,
        0.31289323,  0.34938107, -0.18756661, -2.25982333,  0.01440547,
       -0.97699964, -0.10107761,  0.28043456,  1.88480264, -0.81507891])
```

In [12]:
```
w2v_words = list(w2v_model.wv.vocab)
print("number of words that occured minimum 5 times ",len(w2v_words))
print("sample words ", w2v_words[0:50])
```

```
number of words that occured minimum 5 times  26749
```

```
sample words  ['really', 'nice', 'seasoning', 'bought', 'product', 'sam', 'c
lub', 'happy', 'able', 'purchase', 'cannot', 'get', 'anymore', 'use', 'meat
s', 'spaghetti', 'coffee', 'not', 'bad', 'defiantly', 'anything', 'special',
'price', 'could', 'ethical', 'fair', 'trade', 'organic', 'shade', 'grown',
'etc', 'taste', 'ok', 'stick', 'dean', 'beans', 'smooth', 'flavorful', 'medi
um', 'roast', 'pleasantly', 'surprised', 'k', 'cup', 'would', 'deal', 'dre
w', 'glad', 'dogs', 'love']
```

[4.4.1.2] TFIDF weighted W2v

In [13]:

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

In [14]:

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

train_tf_idf_w2v = []; # the tfidf-w2v for each sentence/review is stored in t
his list
row=0;

#list_of_sentence_train=list_of_sent_train[:30000] # reducing the size of trai
n list due to computational constrain

for sent in tqdm(list_of_sent_train[:60000]): # for each review/sentence
    sent_vec = np.zeros(50) # as word vectors are of zero length
    weight_sum =0; # num of words with a valid vector in the sentence/review
    for word in sent: # for each word in a review/sentence
        if word in w2v_words and word in tfidf_feat:
            vec = w2v_model.wv[word]
#            tf_idf = tf_idf_matrix[row, tfidf_feat.index(word)]
            # to reduce the computation we are
            # dictionary[word] = idf value of word in whole courpus
            # sent.count(word) = tf valeus of word in this review
            tf_idf = dictionary[word]*(sent.count(word)/len(sent))
            sent_vec += (vec * tf_idf)
            weight_sum += tf_idf
    if weight_sum != 0:
        sent_vec /= weight_sum
    train_tf_idf_w2v.append(sent_vec)
    row += 1

len(train_tf_idf_w2v)
```

```
100%|██████████| 60000/60000 [51:13<00:00, 26.40it/s]
```

Out[14]: 60000

In [15]:

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

cv_tf_idf_w2v = []; # the tfidf-w2v for each sentence/review is stored in this
 list
```

```
row=0;

#list_of_sentence_train=list_of_sent_train[:30000] # reducing the size of trai
n list due to computational constrain

for sent in tqdm(list_of_sent_cv[:20000]): # for each review/sentence
    sent_vec = np.zeros(50) # as word vectors are of zero length
    weight_sum =0; # num of words with a valid vector in the sentence/review
    for word in sent: # for each word in a review/sentence
        if word in w2v_words and word in tfidf_feat:
            vec = w2v_model.wv[word]
#             tf_idf = tf_idf_matrix[row, tfidf_feat.index(word)]
            # to reduce the computation we are
            # dictionary[word] = idf value of word in whole courpus
            # sent.count(word) = tf valeus of word in this review
            tf_idf = dictionary[word]*(sent.count(word)/len(sent))
            sent_vec += (vec * tf_idf)
            weight_sum += tf_idf
    if weight_sum != 0:
        sent_vec /= weight_sum
    cv_tf_idf_w2v.append(sent_vec)
    row += 1

len(cv_tf_idf_w2v)
```

```
100%|██████████| 20000/20000 [17:19<00:00, 19.23it/s]
```

Out[15]: 20000

In [16]:

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

test_tf_idf_w2v = []; # the tfidf-w2v for each sentence/review is stored in th
is list
row=0;

#list_of_sentence_train=list_of_sent_train[:30000] # reducing the size of trai
n list due to computational constrain

for sent in tqdm(list_of_sent_test[:20000]): # for each review/sentence
    sent_vec = np.zeros(50) # as word vectors are of zero length
    weight_sum =0; # num of words with a valid vector in the sentence/review
    for word in sent: # for each word in a review/sentence
        if word in w2v_words and word in tfidf_feat:
            vec = w2v_model.wv[word]
#             tf_idf = tf_idf_matrix[row, tfidf_feat.index(word)]
            # to reduce the computation we are
            # dictionary[word] = idf value of word in whole courpus
            # sent.count(word) = tf valeus of word in this review
            tf_idf = dictionary[word]*(sent.count(word)/len(sent))
            sent_vec += (vec * tf_idf)
            weight_sum += tf_idf
    if weight_sum != 0:
        sent_vec /= weight_sum
    test_tf_idf_w2v.append(sent_vec)
    row += 1

len(test_tf_idf_w2v)
```

```
100%|██████████| 20000/20000 [17:31<00:00, 19.03it/s]
```

Out[16]: 20000

In [15]:
```
from sklearn.preprocessing import StandardScaler
sc=StandardScaler(with_mean=True)

train_tf_idf_w2v=sc.fit_transform(train_tf_idf_w2v)
cv_tf_idf_w2v=sc.transform(cv_tf_idf_w2v)
test_tf_idf_w2v=sc.transform(test_tf_idf_w2v)
```

In [16]:
```
# saving train_tf_idf_w2v and test_tf_idf_w2v dataset using pickle for fututre
 use

'''file=open("train_tf_idf_w2v.pkl","wb")
pickle.dump(train_tf_idf_w2v,file)
file.close()

file=open("cv_tf_idf_w2v.pkl",'wb')
pickle.dump(cv_tf_idf_w2v,file)
file.close()

file=open("test_tf_idf_w2v.pkl",'wb')
pickle.dump(test_tf_idf_w2v,file)
file.close()

'''
```

In [6]:
```
#loading train_tf_idf_w2v and test_tf_idf_w2v
file=open('train_tf_idf_w2v.pkl','rb')
train_tf_idf_w2v=pickle.load(file)

file=open('cv_tf_idf_w2v.pkl','rb')
cv_tf_idf_w2v=pickle.load(file)

file=open('test_tf_idf_w2v.pkl','rb')
test_tf_idf_w2v=pickle.load(file)
```

[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

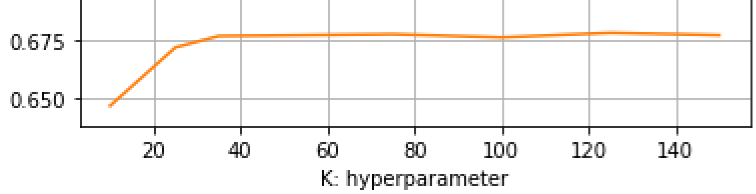
1. There will be an issue of data-leakage if you vectorize the entire data and then split it into train/cv/test.
2. To avoid the issue of data-leakag, make sure to split your data first and then vectorize it.
3. While vectorizing your data, apply the method fit_transform() on you train data, and apply 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 [22]:
```
# Please write all the code with proper documentation
```

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

"""
y_true : array, shape = [n_samples] or [n_samples, n_classes]
True binary labels or binary label indicators.

y_score : array, shape = [n_samples] or [n_samples, n_classes]
Target scores, can either be probability estimates of the positive class, conf
idence values, or non-thresholded measure of
decisions (as returned by "decision_function" on some classifiers).
For binary y_true, y_score is supposed to be the score of the class with great
er label.

"""

train_auc = []
cv_auc = []
K = [10,25,35,50,75,100,125,150]
for i in K:
    neigh = KNeighborsClassifier(n_neighbors=i,algorithm='brute',n_jobs=-2)
    neigh.fit(train_bow[:60000], y_train[:60000])
    # roc_auc_score(y_true, y_score) the 2nd parameter should be probability e
stimates of the positive class
    # not the predicted outputs


    #predicting on train and cv using blocks
    y_train_pred = []
    for i in range(0, train_bow[:60000].shape[0], 1000):
        y_train_pred.extend(neigh.predict_proba(train_bow[i:i+1000])[:,1])

    y_cv_pred = []
    for i in range(0, cv_bow[:20000].shape[0], 1000):
        y_cv_pred.extend(neigh.predict_proba(cv_bow[i:i+1000])[:,1])


    train_auc.append(roc_auc_score(y_train[:60000],y_train_pred))
    cv_auc.append(roc_auc_score(y_cv[:20000], y_cv_pred))



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

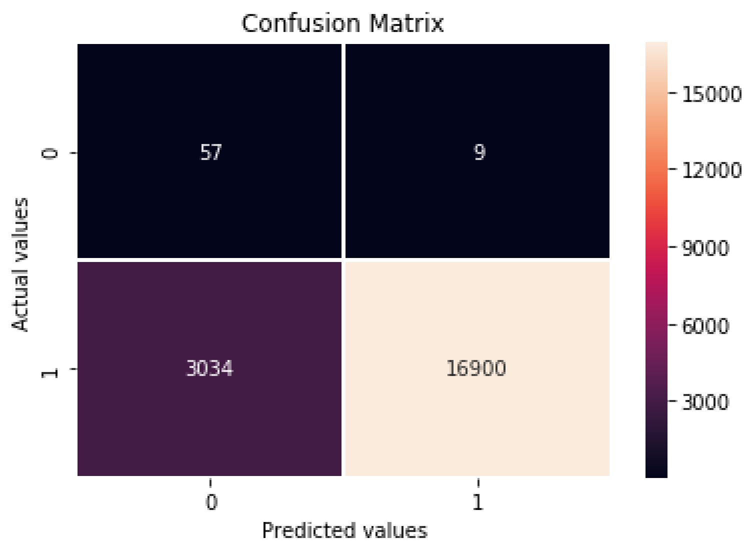


In [21]:

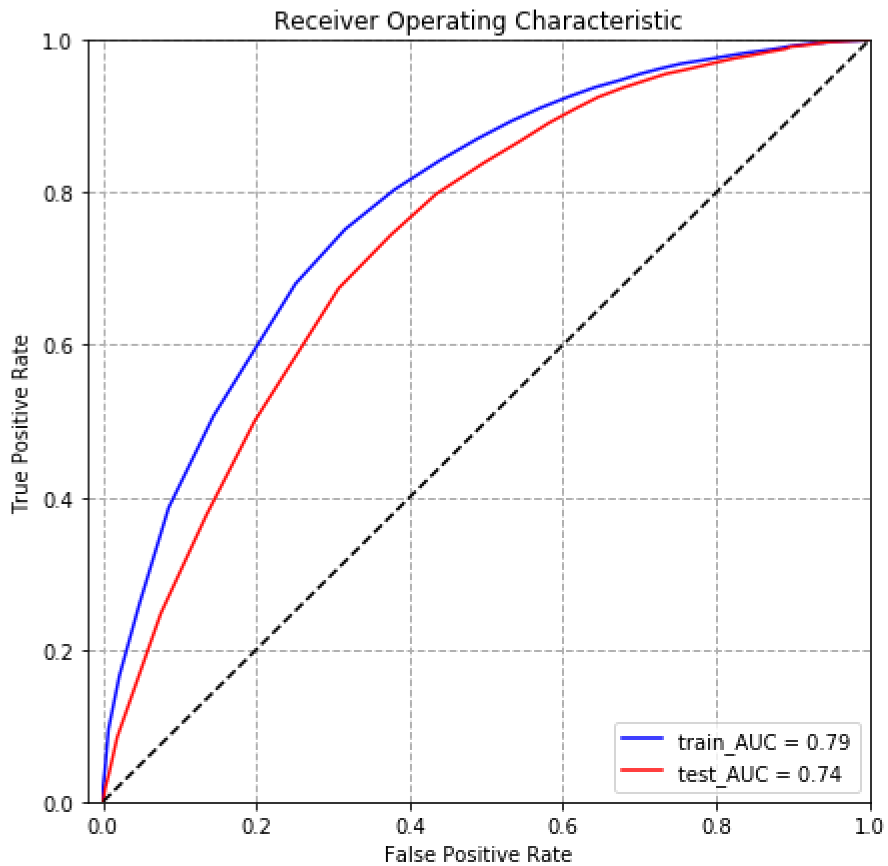
```
# Test dataset

#using optimum_k to find generalistion ROC AUC accuracy
k=125 #optimum 'K'

clf=KNeighborsClassifier(n_neighbors=k,weights='uniform', algorithm='brute')
clf.fit(train_bow[:60000],y_train[:60000])


y_pred = []
for i in range(0, test_bow[:20000].shape[0], 1000):
    y_pred.extend(clf.predict(test_bow[i:i+1000]))

y_pred_proba = []
for i in range(0, test_bow[:20000].shape[0], 1000):
    y_pred_proba.extend(clf.predict_proba(test_bow[i:i+1000])[:,1])


accuracy=accuracy_score(y_test[:20000],y_pred)
roc_auc=roc_auc_score(y_test[:20000],y_pred_proba)
print(f'\ngenearalisation roc_auc on best k-value at k = {k} is {roc_auc:.2f}'
)
print(f'\nmisclassification percentage is {(1-accuracy):.2f}%')


#ploting confusion matrix
sn.heatmap(confusion_matrix(y_pred,y_test[:20000]),annot=True, fmt="d",linewid
ths=.5)
plt.title('Confusion Matrix')
plt.xlabel('Predicted values')
plt.ylabel('Actual values')
plt.show()
print("\n\nclassification report:\n",classification_report(y_test[:20000],y_pr
ed))


# ROC Curve (reference:stack overflow with little modification)

y_train_pred_proba = []
for i in range(0, train_bow[:60000].shape[0], 1000):
    y_train_pred_proba.extend(clf.predict_proba(train_bow[i:i+1000])[:,1])


train_fpr, train_tpr, train_threshold =roc_curve(y_train[:60000], y_train_pred
_proba)
test_fpr, test_tpr, test_threshold =roc_curve(y_test[:20000], y_pred_proba)

roc_auc_train = roc_auc_score(y_train[:60000],y_train_pred_proba)
roc_auc_test = roc_auc_score(y_test[:20000],y_pred_proba)

plt.figure(figsize=(7,7))
plt.title('Receiver Operating Characteristic')
plt.plot(train_fpr, train_tpr, 'b', label = 'train_AUC = %0.2f' % roc_auc_trai
n)
```

```
plt.plot(test_fpr, test_tpr, 'r', label = 'test_AUC = %0.2f' % roc_auc_test)
plt.legend(loc = 'lower right')
plt.plot([0, 1], [0, 1],'k--')
plt.xlim([-0.02, 1])
plt.ylim([0, 1])
plt.ylabel('True Positive Rate')
plt.xlabel('False Positive Rate')
plt.grid(linestyle='--', linewidth=1)
plt.show()
```

```
genearalisation roc_auc on best k-value at k = 125 is 0.74

misclassification percentage is 0.15%
```




```
classification report:
             precision    recall  f1-score   support

          0       0.86      0.02      0.04      3091
          1       0.85      1.00      0.92     16909

avg / total       0.85      0.85      0.78     20000
```

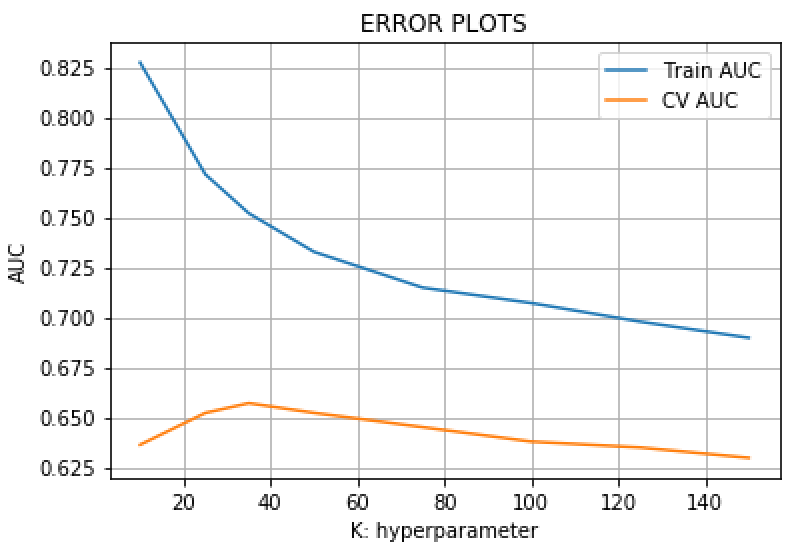


[5.1.2] Applying KNN brute force on TFIDF, SET 2

In [24]:

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

from sklearn.neighbors import KNeighborsClassifier
from sklearn.metrics import roc_auc_score
import matplotlib.pyplot as plt

"""
y_true : array, shape = [n_samples] or [n_samples, n_classes]
True binary labels or binary label indicators.

y_score : array, shape = [n_samples] or [n_samples, n_classes]
Target scores, can either be probability estimates of the positive class, conf
idence values, or non-thresholded measure of
decisions (as returned by “decision_function” on some classifiers).
For binary y_true, y_score is supposed to be the score of the class with great
er label.

"""

train_auc = []
cv_auc = []
K = [10,25,35,50,75,100,125,150]
for i in K:
    neigh = KNeighborsClassifier(n_neighbors=i)
    neigh.fit(train_tf_idf[:60000], y_train[:60000])
    # roc_auc_score(y_true, y_score) the 2nd parameter should be probability e
stimates of the positive class
    # not the predicted outputs

    #predicting on train and cv using blocks
    y_train_pred = []
```

```
    for i in range(0, train_tf_idf[:60000].shape[0], 1000):
        y_train_pred.extend(neigh.predict_proba(train_tf_idf[i:i+1000])[:,1])

    y_cv_pred = []
    for i in range(0, cv_tf_idf[:20000].shape[0], 1000):
        y_cv_pred.extend(neigh.predict_proba(cv_tf_idf[i:i+1000])[:,1])


    train_auc.append(roc_auc_score(y_train[:60000],y_train_pred))
    cv_auc.append(roc_auc_score(y_cv[:20000], y_cv_pred))



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

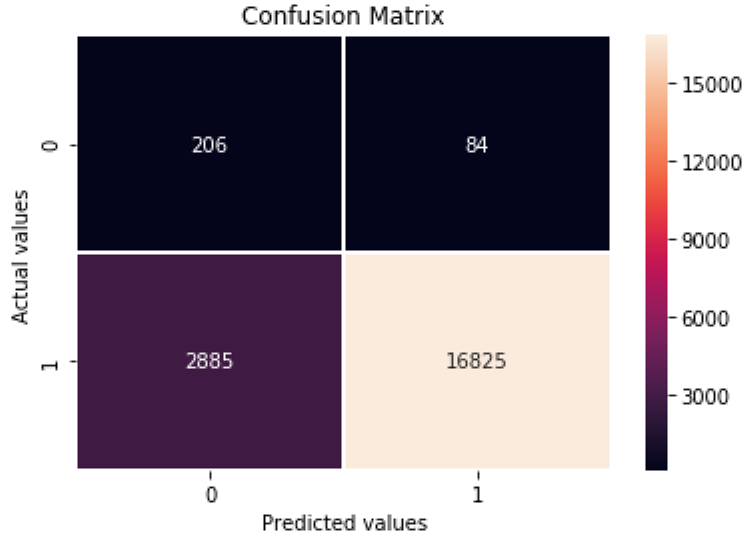


In [25]:

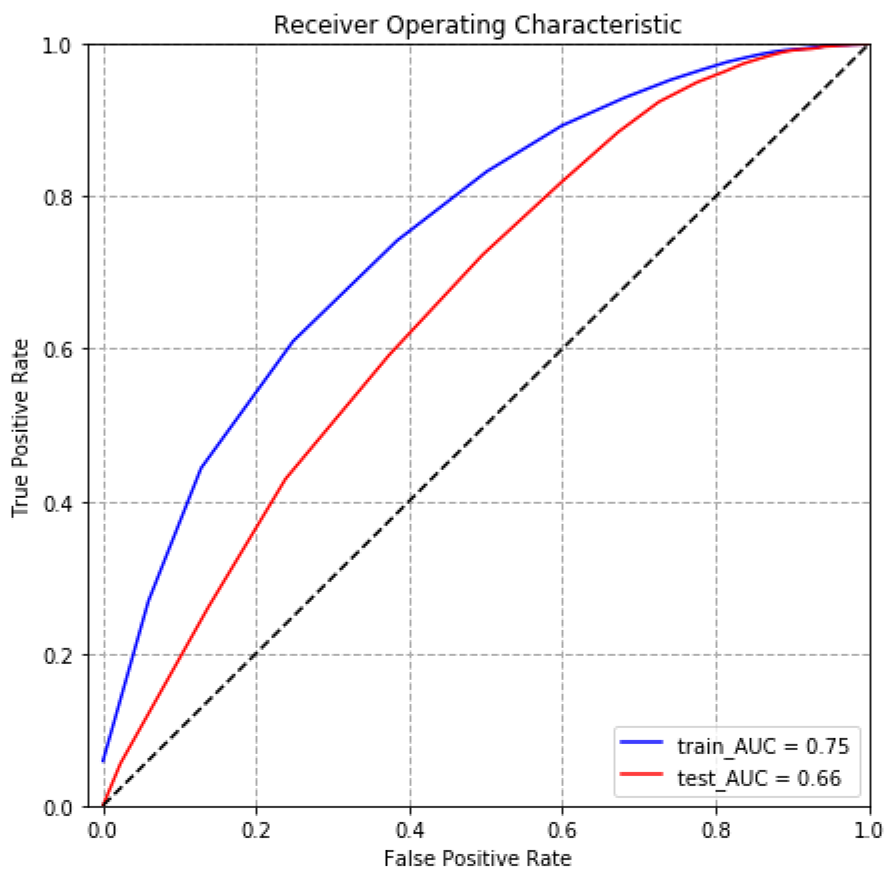
```
# Test dataset

#using optimum_k to find generalistion ROC AUC accuracy

k=35 #optimum 'K'

clf=KNeighborsClassifier(n_neighbors=k,weights='uniform', algorithm='brute')
clf.fit(train_tf_idf[:60000],y_train[:60000])


y_pred = []
for i in range(0, test_tf_idf[:20000].shape[0], 1000):
    y_pred.extend(clf.predict(test_tf_idf[i:i+1000]))

y_pred_proba = []
for i in range(0, test_tf_idf[:20000].shape[0], 1000):
    y_pred_proba.extend(clf.predict_proba(test_tf_idf[i:i+1000])[:,1])



accuracy=accuracy_score(y_test[:20000],y_pred)
roc_auc=roc_auc_score(y_test[:20000],y_pred_proba)
```

```
print(f'\ngenearalisation roc_auc on best k-value at k = {k} is {roc_auc:.2f}'
)
print(f'\nmisclassification percentage is {(1-accuracy):.2f}%')


#ploting confusion matrix
sn.heatmap(confusion_matrix(y_pred,y_test[:20000]),annot=True, fmt="d",linewid
ths=.5)
plt.title('Confusion Matrix')
plt.xlabel('Predicted values')
plt.ylabel('Actual values')
plt.show()
print("\n\nclassification report:\n",classification_report(y_test[:20000],y_pr
ed))


# ROC Curve (reference:stack overflow with little modification)

y_train_pred_proba = []
for i in range(0, train_tf_idf[:60000].shape[0], 1000):
    y_train_pred_proba.extend(clf.predict_proba(train_tf_idf[i:i+1000])[:,1])


train_fpr, train_tpr, train_threshold =roc_curve(y_train[:60000], y_train_pred
_proba)
test_fpr, test_tpr, test_threshold =roc_curve(y_test[:20000], y_pred_proba)

roc_auc_train = roc_auc_score(y_train[:60000],y_train_pred_proba)
roc_auc_test = roc_auc_score(y_test[:20000],y_pred_proba)


plt.figure(figsize=(7,7))
plt.title('Receiver Operating Characteristic')
plt.plot(train_fpr, train_tpr, 'b', label = 'train_AUC = %0.2f' % roc_auc_trai
n)
plt.plot(test_fpr, test_tpr, 'r', label = 'test_AUC = %0.2f' % roc_auc_test)
plt.legend(loc = 'lower right')
plt.plot([0, 1], [0, 1],'k--')
plt.xlim([-0.02, 1])
plt.ylim([0, 1])
plt.ylabel('True Positive Rate')
plt.xlabel('False Positive Rate')
plt.grid(linestyle='--', linewidth=1)
plt.show()
```

```
genearalisation roc_auc on best k-value at k = 35 is 0.66

misclassification percentage is 0.15%
```



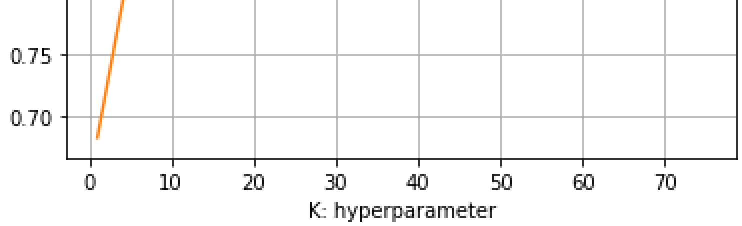

```
classification report:
             precision    recall  f1-score   support

          0       0.71      0.07      0.12      3091
          1       0.85      1.00      0.92     16909

avg / total       0.83      0.85      0.80     20000
```



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

In [22]:
```
# Please write all the code with proper documentation

from sklearn.neighbors import KNeighborsClassifier
from sklearn.metrics import roc_auc_score
```

```
import matplotlib.pyplot as plt

"""
y_true : array, shape = [n_samples] or [n_samples, n_classes]
True binary labels or binary label indicators.

y_score : array, shape = [n_samples] or [n_samples, n_classes]
Target scores, can either be probability estimates of the positive class, conf
idence values, or non-thresholded measure of
decisions (as returned by "decision_function" on some classifiers).
For binary y_true, y_score is supposed to be the score of the class with great
er label.

"""

train_auc = []
cv_auc = []
K = [1, 5,10,15,25,35,50,75]
for i in K:
    neigh = KNeighborsClassifier(n_neighbors=i)
    neigh.fit(train_w2v[:60000], y_train[:60000])
    # roc_auc_score(y_true, y_score) the 2nd parameter should be probability e
stimates of the positive class
    # not the predicted outputs


    y_train_pred = []
    for i in range(0, len(train_w2v[:60000]), 1000):
        y_train_pred.extend(neigh.predict_proba(train_w2v[i:i+1000])[:,1])

    y_cv_pred = []
    for i in range(0, len(cv_w2v[:20000]), 1000):
        y_cv_pred.extend(neigh.predict_proba(cv_w2v[i:i+1000])[:,1])


    train_auc.append(roc_auc_score(y_train[:60000],y_train_pred))
    cv_auc.append(roc_auc_score(y_cv[:20000], y_cv_pred))



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

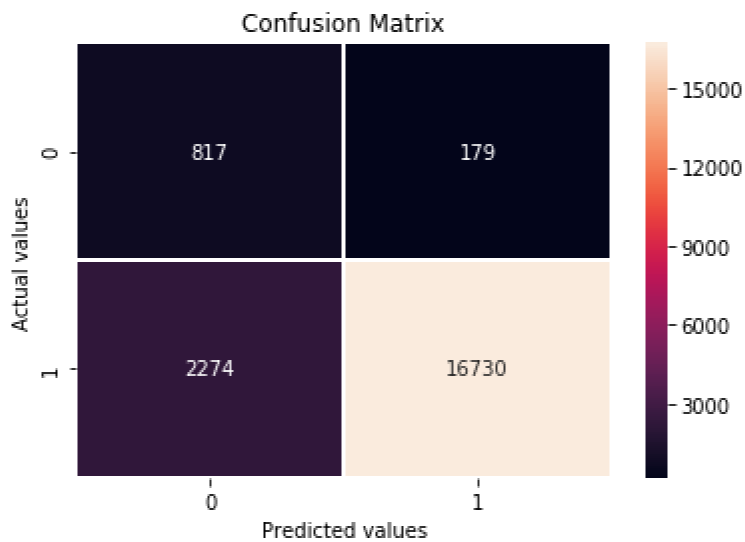


In [3]:

```
# Test dataset

#using optimum_k to find generalistion ROC AUC accuracy

k=75 #optimum 'K'

clf=KNeighborsClassifier(n_neighbors=k,weights='uniform', algorithm='brute')
clf.fit(train_w2v[:60000],y_train[:60000])


y_pred = []
for i in range(0, test_w2v[:20000].shape[0], 1000):
    y_pred.extend(clf.predict(test_w2v[i:i+1000]))

y_pred_proba = []
for i in range(0, test_w2v[:20000].shape[0], 1000):
    y_pred_proba.extend(clf.predict_proba(test_w2v[i:i+1000])[:,1])



accuracy=accuracy_score(y_test[:20000],y_pred)
roc_auc=roc_auc_score(y_test[:20000],y_pred_proba)
print(f'\ngenearalisation roc_auc on best k-value at k = {k} is {roc_auc:.2f}'
)
print(f'\nmisclassification percentage is {(1-accuracy):.2f}%')


#ploting confusion matrix
sn.heatmap(confusion_matrix(y_pred,y_test[:20000]),annot=True, fmt="d",linewid
ths=.5)
plt.title('Confusion Matrix')
plt.xlabel('Predicted values')
plt.ylabel('Actual values')
plt.show()
print("\n\nclassification report:\n",classification_report(y_test[:20000],y_pr
ed))


# ROC Curve (reference:stack overflow with little modification)

y_train_pred_proba = []
for i in range(0, train_w2v[:60000].shape[0], 1000):
    y_train_pred_proba.extend(clf.predict_proba(train_w2v[i:i+1000])[:,1])


train_fpr, train_tpr, train_threshold =roc_curve(y_train[:60000], y_train_pred
_proba)
test_fpr, test_tpr, test_threshold =roc_curve(y_test[:20000], y_pred_proba)

roc_auc_train = roc_auc_score(y_train[:60000],y_train_pred_proba)
roc_auc_test = roc_auc_score(y_test[:20000],y_pred_proba)
```

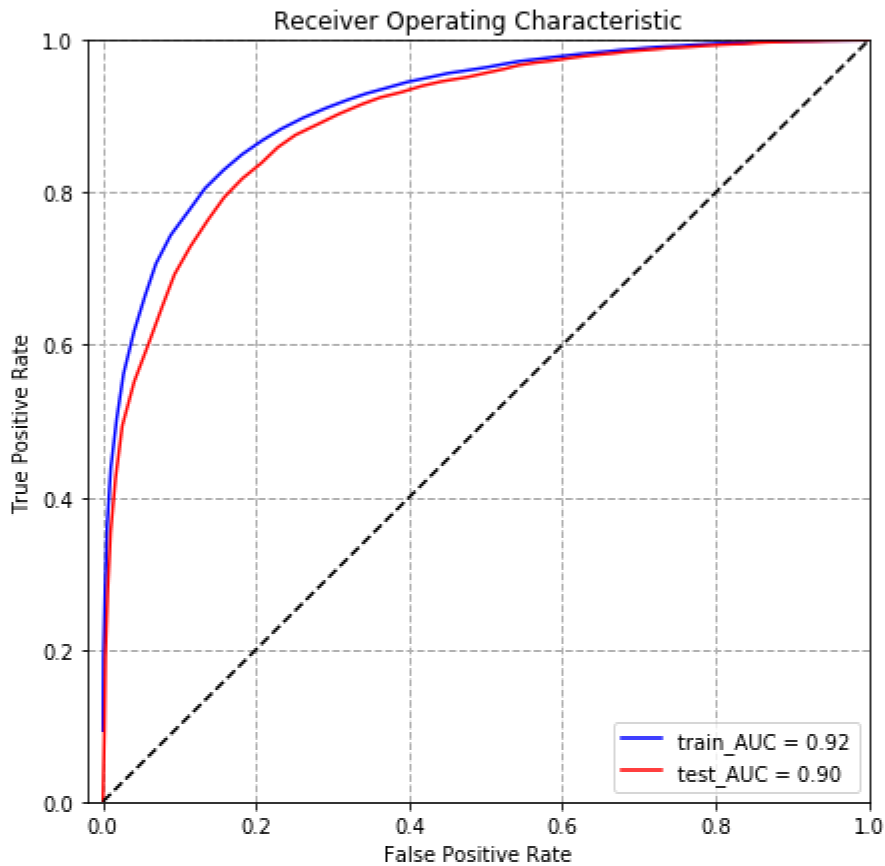
```
plt.figure(figsize=(7,7))
plt.title('Receiver Operating Characteristic')
plt.plot(train_fpr, train_tpr, 'b', label = 'train_AUC = %0.2f' % roc_auc_trai
n)
plt.plot(test_fpr, test_tpr, 'r', label = 'test_AUC = %0.2f' % roc_auc_test)
plt.legend(loc = 'lower right')
plt.plot([0, 1], [0, 1],'k--')
plt.xlim([-0.02, 1])
plt.ylim([0, 1])
plt.ylabel('True Positive Rate')
plt.xlabel('False Positive Rate')
plt.grid(linestyle='--', linewidth=1)
plt.show()
```

```
genearalisation roc_auc on best k-value at k = 75 is 0.90

misclassification percentage is 0.12%
```



classification report:

	precision	recall	f1-score	support
0	0.82	0.26	0.40	3091
1	0.88	0.99	0.93	16909
avg / total	0.87	0.88	0.85	20000

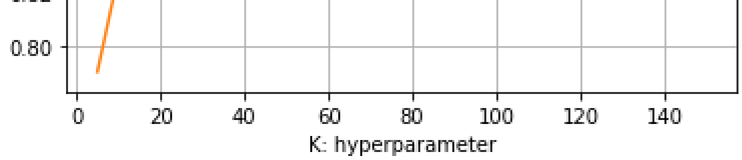


[5.1.4] Applying KNN brute force on TFIDF W2V, SET 4

In [21]:

```
# Test dataset

#using optimum_k to find generalistion ROC AUC accuracy

k=125 #optimum 'K'

clf=KNeighborsClassifier(n_neighbors=k,weights='uniform', algorithm='brute',n_
jobs=-2)
clf.fit(train_tf_idf_w2v[:60000],y_train[:60000])


y_pred = []
for i in range(0, test_tf_idf_w2v[:20000].shape[0], 1000):
    y_pred.extend(clf.predict(test_tf_idf_w2v[i:i+1000]))

y_pred_proba = []
for i in range(0, test_tf_idf_w2v[:20000].shape[0], 1000):
    y_pred_proba.extend(clf.predict_proba(test_tf_idf_w2v[i:i+1000])[:,1])



accuracy=accuracy_score(y_test[:20000],y_pred)
roc_auc=roc_auc_score(y_test[:20000],y_pred_proba)
print(f'\ngenearalisation roc_auc on best k-value at k = {k} is {roc_auc:.2f}'
)
print(f'\nmisclassification percentage is {(1-accuracy):.2f}%')


#ploting confusion matrix
sn.heatmap(confusion_matrix(y_pred,y_test[:20000]),annot=True, fmt="d",linewid
ths=.5)
```

```
plt.title('Confusion Matrix')
plt.xlabel('Predicted values')
plt.ylabel('Actual values')
plt.show()
print("\n\nclassification report:\n",classification_report(y_test[:20000],y_pr
ed))


# ROC Curve (reference:stack overflow with little modification)

y_train_pred_proba = []
for i in range(0, train_tf_idf_w2v[:60000].shape[0], 1000):
    y_train_pred_proba.extend(clf.predict_proba(train_tf_idf_w2v[i:i+1000])[:,
1])


train_fpr, train_tpr, train_threshold =roc_curve(y_train[:60000], y_train_pred
_proba)
test_fpr, test_tpr, test_threshold =roc_curve(y_test[:20000], y_pred_proba)

roc_auc_train = roc_auc_score(y_train[:60000],y_train_pred_proba)
roc_auc_test = roc_auc_score(y_test[:20000],y_pred_proba)


'''
# ROC Curve (reference:stack overflow with little modification)
y_train_pred_proba=clf.predict_proba(train_tf_idf[:20000]) #storing score of x
_train for ROC-AUC score

train_fpr, train_tpr, train_threshold =roc_curve(y_train[:60000], y_train_pred
_proba[:,1])
test_fpr, test_tpr, test_threshold =roc_curve(y_test[:20000], y_pred_proba[:,
1])

roc_auc_train = roc_auc_score(y_train[:60000],y_train_pred_proba[:,1])
roc_auc_test = roc_auc_score(y_test[:20000],y_pred_proba[:,1])
'''
plt.figure(figsize=(7,7))
plt.title('Receiver Operating Characteristic')
plt.plot(train_fpr, train_tpr, 'b', label = 'train_AUC = %0.2f' % roc_auc_trai
n)
plt.plot(test_fpr, test_tpr, 'r', label = 'test_AUC = %0.2f' % roc_auc_test)
plt.legend(loc = 'lower right')
plt.plot([0, 1], [0, 1],'k--')
plt.xlim([-0.02, 1])
plt.ylim([0, 1])
plt.ylabel('True Positive Rate')
plt.xlabel('False Positive Rate')
plt.grid(linestyle='--', linewidth=1)
plt.show()
```

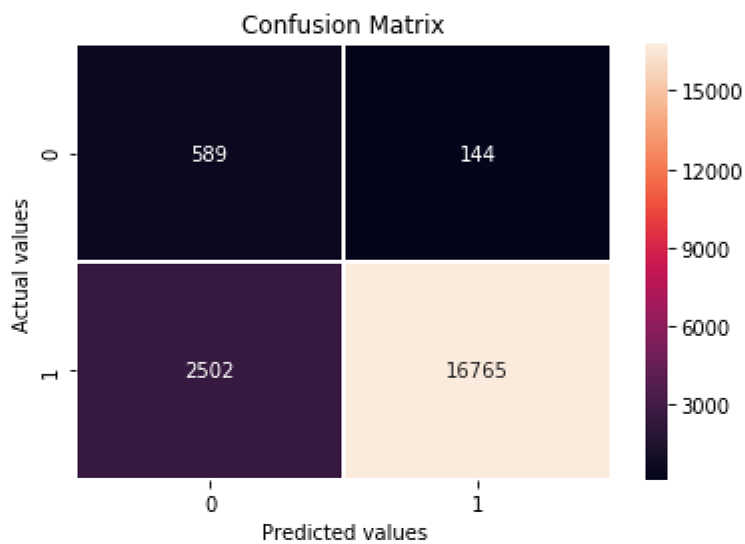


In [4]:

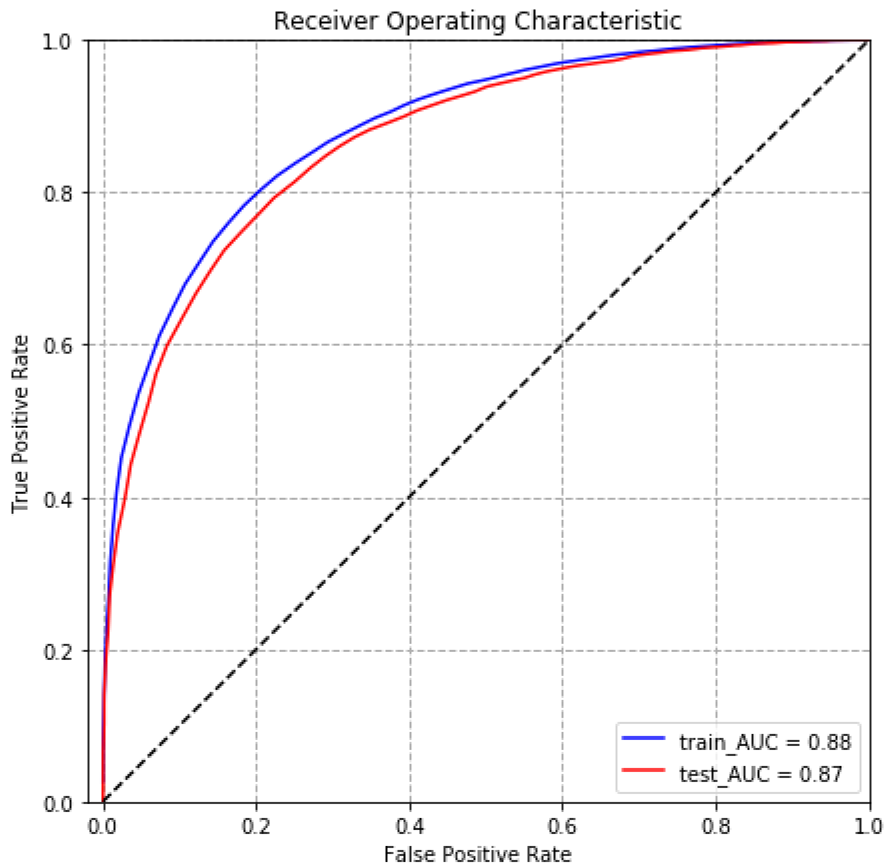
```
# Test dataset

#using optimum_k to find generalistion ROC AUC accuracy

k=125 #optimum 'K'

clf=KNeighborsClassifier(n_neighbors=k,weights='uniform', algorithm='brute')
clf.fit(train_tf_idf_w2v[:60000],y_train[:60000])


y_pred = []
for i in range(0, test_tf_idf_w2v[:20000].shape[0], 1000):
    y_pred.extend(clf.predict(test_tf_idf_w2v[i:i+1000]))

y_pred_proba = []
for i in range(0, test_tf_idf_w2v[:20000].shape[0], 1000):
    y_pred_proba.extend(clf.predict_proba(test_tf_idf_w2v[i:i+1000])[:,1])



accuracy=accuracy_score(y_test[:20000],y_pred)
roc_auc=roc_auc_score(y_test[:20000],y_pred_proba)
print(f'\ngenearalisation roc_auc on best k-value at k = {k} is {roc_auc:.2f}'
)
print(f'\nmisclassification percentage is {(1-accuracy):.2f}%')


#ploting confusion matrix
sn.heatmap(confusion_matrix(y_pred,y_test[:20000]),annot=True, fmt="d",linewid
ths=.5)
plt.title('Confusion Matrix')
plt.xlabel('Predicted values')
plt.ylabel('Actual values')
plt.show()
print("\n\nclassification report:\n",classification_report(y_test[:20000],y_pr
ed))


# ROC Curve (reference:stack overflow with little modification)

y_train_pred_proba = []
for i in range(0, train_tf_idf_w2v[:60000].shape[0], 1000):
    y_train_pred_proba.extend(clf.predict_proba(train_tf_idf_w2v[i:i+1000])[:,
1])


train_fpr, train_tpr, train_threshold =roc_curve(y_train[:60000], y_train_pred
_proba)
test_fpr, test_tpr, test_threshold =roc_curve(y_test[:20000], y_pred_proba)

roc_auc_train = roc_auc_score(y_train[:60000],y_train_pred_proba)
roc_auc_test = roc_auc_score(y_test[:20000],y_pred_proba)


plt.figure(figsize=(7,7))
```

```
plt.title('Receiver Operating Characteristic')
plt.plot(train_fpr, train_tpr, 'b', label = 'train_AUC = %0.2f' % roc_auc_trai
n)
plt.plot(test_fpr, test_tpr, 'r', label = 'test_AUC = %0.2f' % roc_auc_test)
plt.legend(loc = 'lower right')
plt.plot([0, 1], [0, 1],'k--')
plt.xlim([-0.02, 1])
plt.ylim([0, 1])
plt.ylabel('True Positive Rate')
plt.xlabel('False Positive Rate')
plt.grid(linestyle='--', linewidth=1)
plt.show()
```

```
genearalisation roc_auc on best k-value at k = 125 is 0.87

misclassification percentage is 0.13%
```




```
classification report:
             precision    recall  f1-score   support

          0       0.80      0.19      0.31      3091
          1       0.87      0.99      0.93     16909

avg / total       0.86      0.87      0.83     20000
```



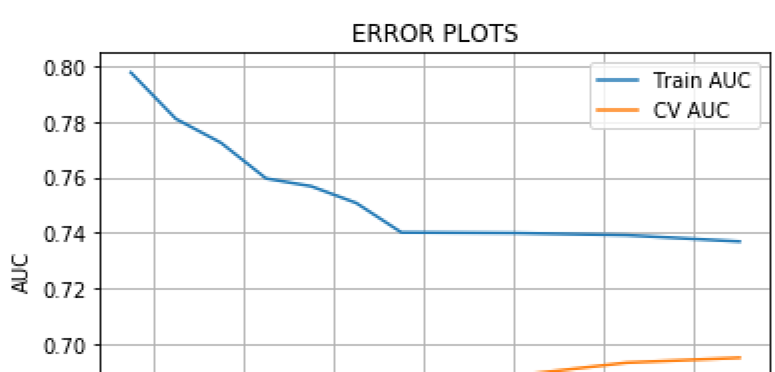
[5.2] Applying KNN kd-tree

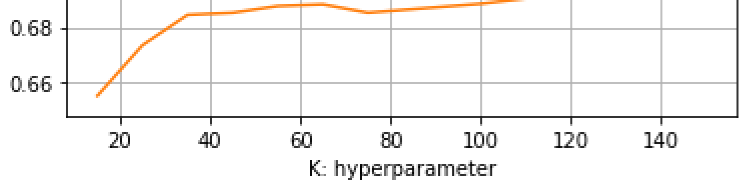
[5.2.1] Applying KNN kd-tree on BOW, SET 5

In [8]:
```
from sklearn.feature_extraction.text  import CountVectorizer
count_vect=CountVectorizer(max_df=2000, min_df=50, max_features=500)

train_bow=count_vect.fit_transform(x_train[:60000])
train_bow=train_bow.toarray()
# transform cv and test dataset
cv_bow=count_vect.transform(x_cv[:20000]).toarray()
test_bow=count_vect.transform(x_test[:20000]).toarray()
```

In [5]:
```
# Please write all the code with proper documentation

from sklearn.neighbors import KNeighborsClassifier
from sklearn.metrics import roc_auc_score
import matplotlib.pyplot as plt

"""
y_true : array, shape = [n_samples] or [n_samples, n_classes]
True binary labels or binary label indicators.

y_score : array, shape = [n_samples] or [n_samples, n_classes]
Target scores, can either be probability estimates of the positive class, conf
idence values, or non-thresholded measure of
decisions (as returned by “decision_function” on some classifiers).
For binary y_true, y_score is supposed to be the score of the class with great
er label.

"""
```

```
train_auc = []
cv_auc = []
K = [15,25,35,45,55,65,75,100,125,150]
for i in K:
    neigh = KNeighborsClassifier(n_neighbors=i,algorithm='kd_tree',n_jobs=-1)
    neigh.fit(train_bow[:60000], y_train[:60000])
    # roc_auc_score(y_true, y_score) the 2nd parameter should be probability e
stimates of the positive class
    # not the predicted outputs


    y_train_pred = []
    for i in range(0, train_bow[:60000].shape[0], 1000):
        y_train_pred.extend(neigh.predict_proba(train_bow[i:i+1000])[:,1]) # t
his is a pseudo code

    y_cv_pred = []
    for i in range(0, cv_bow[:20000].shape[0], 1000):
        y_cv_pred.extend(neigh.predict_proba(cv_bow[i:i+1000])[:,1]) # this is
 a pseudo code


    train_auc.append(roc_auc_score(y_train[:60000],y_train_pred))
    cv_auc.append(roc_auc_score(y_cv[:20000], y_cv_pred))


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



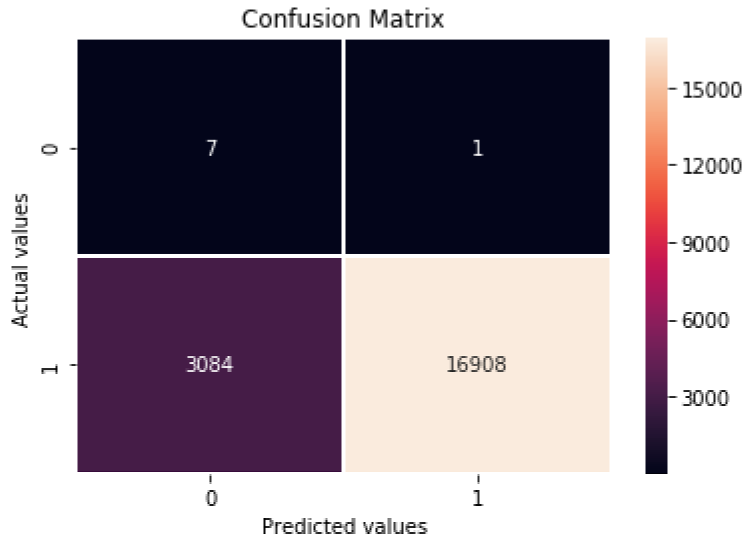


In [9]:

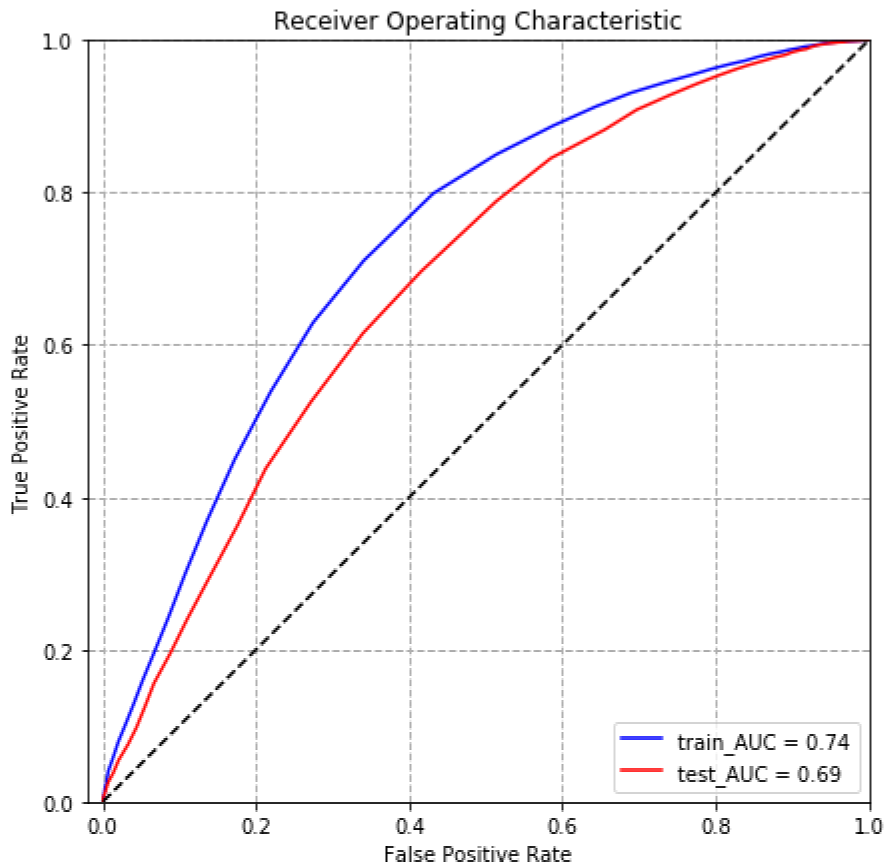
```
# Test dataset

#usin`g optimum_k to find generalistion ROC AUC accuracy
k=150 #optimum 'K'

clf=KNeighborsClassifier(n_neighbors=k,weights='uniform', algorithm='kd_tree',
n_jobs=-2)
clf.fit(train_bow[:60000],y_train[:60000])


y_pred = []
for i in range(0, test_bow[:20000].shape[0], 1000):
    y_pred.extend(clf.predict(test_bow[i:i+1000]))

y_pred_proba = []
for i in range(0, test_bow[:20000].shape[0], 1000):
    y_pred_proba.extend(clf.predict_proba(test_bow[i:i+1000])[:,1])


accuracy=accuracy_score(y_test[:20000],y_pred)
roc_auc=roc_auc_score(y_test[:20000],y_pred_proba)
print(f'\ngenearalisation roc_auc on best k-value at k = {k} is {roc_auc:.2f}'
)
print(f'\nmisclassification percentage is {(1-accuracy):.2f}%')


#ploting confusion matrix
sn.heatmap(confusion_matrix(y_pred,y_test[:20000]),annot=True, fmt="d",linewid
ths=.5)
plt.title('Confusion Matrix')
plt.xlabel('Predicted values')
plt.ylabel('Actual values')
plt.show()
print("\n\nclassification report:\n",classification_report(y_test[:20000],y_pr
ed))


# ROC Curve (reference:stack overflow with little modification)

y_train_pred_proba = []
for i in range(0, train_bow[:60000].shape[0], 1000):
    y_train_pred_proba.extend(clf.predict_proba(train_bow[i:i+1000])[:,1])


train_fpr, train_tpr, train_threshold =roc_curve(y_train[:60000], y_train_pred
_proba)
test_fpr, test_tpr, test_threshold =roc_curve(y_test[:20000], y_pred_proba)

roc_auc_train = roc_auc_score(y_train[:60000],y_train_pred_proba)
roc_auc_test = roc_auc_score(y_test[:20000],y_pred_proba)

plt.figure(figsize=(7,7))
plt.title('Receiver Operating Characteristic')
plt.plot(train_fpr, train_tpr, 'b', label = 'train_AUC = %0.2f' % roc_auc_trai
n)
```

```
plt.plot(test_fpr, test_tpr, 'r', label = 'test_AUC = %0.2f' % roc_auc_test)
plt.legend(loc = 'lower right')
plt.plot([0, 1], [0, 1],'k--')
plt.xlim([-0.02, 1])
plt.ylim([0, 1])
plt.ylabel('True Positive Rate')
plt.xlabel('False Positive Rate')
plt.grid(linestyle='--', linewidth=1)
plt.show()
```

```
genearalisation roc_auc on best k-value at k = 150 is 0.69

misclassification percentage is 0.15%
```




```
classification report:
             precision    recall  f1-score   support

          0       0.88      0.00      0.00      3091
          1       0.85      1.00      0.92     16909

avg / total       0.85      0.85      0.78     20000
```



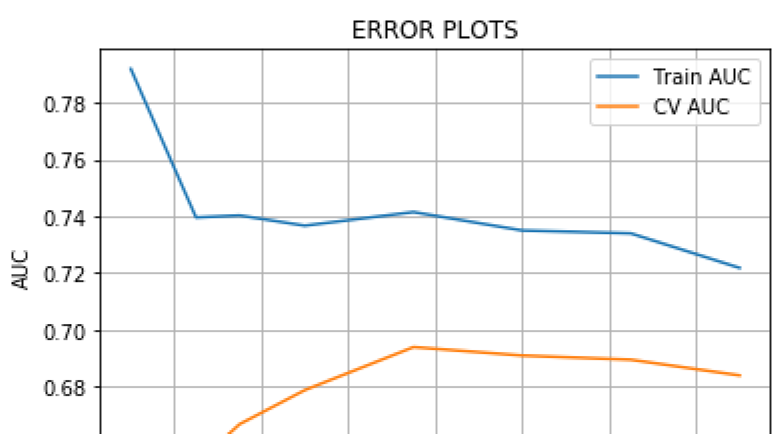
[5.2.2] Applying KNN kd-tree on TFIDF, SET 6

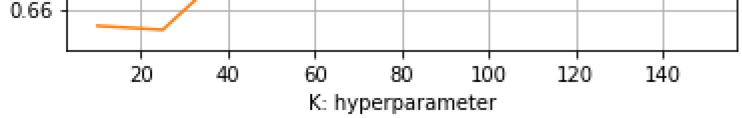
In [3]:
```
# tf-idf "from sklearn.feature_extraction.text.TfidfVectorizer"
tf_idf=TfidfVectorizer(max_df=2000, min_df=50, max_features=500)

train_tf_idf=tf_idf.fit_transform(x_train).toarray()
cv_tf_idf=tf_idf.transform(x_cv).toarray()
test_tf_idf=tf_idf.transform(x_test).toarray()
```

In [4]:
```
from sklearn.preprocessing import StandardScaler

sc=StandardScaler(with_mean=False)

train_tf_idf=sc.fit_transform(train_tf_idf)
cv_tf_idf=sc.transform(cv_tf_idf)
test_tf_idf=sc.transform(test_tf_idf)
```

In [5]:
```
# Please write all the code with proper documentation

from sklearn.neighbors import KNeighborsClassifier
from sklearn.metrics import roc_auc_score
import matplotlib.pyplot as plt

"""
y_true : array, shape = [n_samples] or [n_samples, n_classes]
True binary labels or binary label indicators.

y_score : array, shape = [n_samples] or [n_samples, n_classes]
Target scores, can either be probability estimates of the positive class, conf
idence values, or non-thresholded measure of
decisions (as returned by "decision_function" on some classifiers).
For binary y_true, y_score is supposed to be the score of the class with great
```

```
er label.

"""

train_auc = []
cv_auc = []
K = [10,25,35,50,75,100,125,150]
for i in K:
    neigh = KNeighborsClassifier(n_neighbors=i, algorithm='kd_tree',n_jobs=-1)
    neigh.fit(train_tf_idf[:60000], y_train[:60000])
    # roc_auc_score(y_true, y_score) the 2nd parameter should be probability e
stimates of the positive class
    # not the predicted outputs



    #predicting on train and cv using blocks
    y_train_pred = []
    for i in range(0, train_tf_idf[:60000].shape[0], 1000):
        y_train_pred.extend(neigh.predict_proba(train_tf_idf[i:i+1000])[:,1])

    y_cv_pred = []
    for i in range(0, cv_tf_idf[:20000].shape[0], 1000):
        y_cv_pred.extend(neigh.predict_proba(cv_tf_idf[i:i+1000])[:,1])


    train_auc.append(roc_auc_score(y_train[:60000],y_train_pred))
    cv_auc.append(roc_auc_score(y_cv[:20000], y_cv_pred))



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

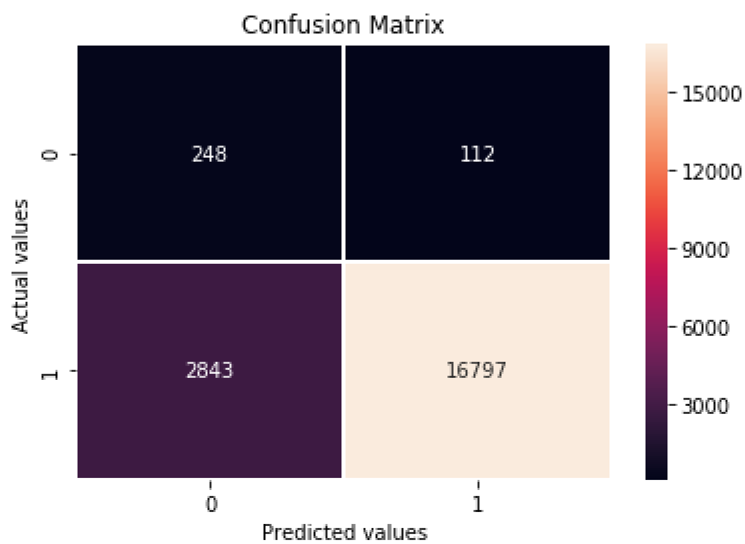




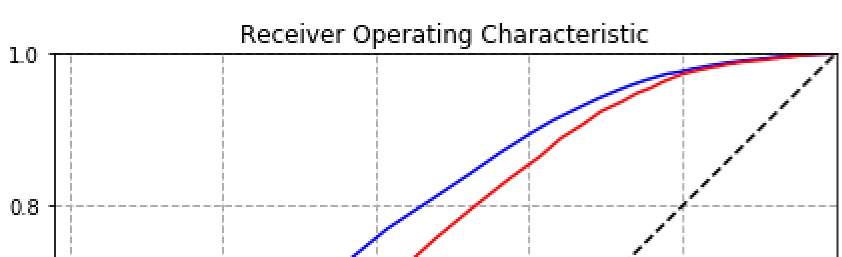
In [6]:
```
# Test dataset

#using optimum_k to find generalistion ROC AUC accuracy

k=75 #optimum 'K'

clf=KNeighborsClassifier(n_neighbors=k,weights='uniform', algorithm='kd_tree')
clf.fit(train_tf_idf[:60000],y_train[:60000])


y_pred = []
for i in range(0, test_tf_idf[:20000].shape[0], 1000):
    y_pred.extend(clf.predict(test_tf_idf[i:i+1000]))

y_pred_proba = []
for i in range(0, test_tf_idf[:20000].shape[0], 1000):
    y_pred_proba.extend(clf.predict_proba(test_tf_idf[i:i+1000])[:,1])



accuracy=accuracy_score(y_test[:20000],y_pred)
roc_auc=roc_auc_score(y_test[:20000],y_pred_proba)
print(f'\ngenearalisation roc_auc on best k-value at k = {k} is {roc_auc:.2f}'
)
print(f'\nmisclassification percentage is {(1-accuracy):.2f}%')


#ploting confusion matrix
sn.heatmap(confusion_matrix(y_pred,y_test[:20000]),annot=True, fmt="d",linewidths=.5)
plt.title('Confusion Matrix')
plt.xlabel('Predicted values')
plt.ylabel('Actual values')
plt.show()
print("\n\nclassification report:\n",classification_report(y_test[:20000],y_pred))


# ROC Curve (reference:stack overflow with little modification)

y_train_pred_proba = []
for i in range(0, train_tf_idf[:60000].shape[0], 1000):
    y_train_pred_proba.extend(clf.predict_proba(train_tf_idf[i:i+1000])[:,1])


train_fpr, train_tpr, train_threshold =roc_curve(y_train[:60000], y_train_pred_proba)
test_fpr, test_tpr, test_threshold =roc_curve(y_test[:20000], y_pred_proba)

roc_auc_train = roc_auc_score(y_train[:60000],y_train_pred_proba)
roc_auc_test = roc_auc_score(y_test[:20000],y_pred_proba)


'''
# ROC Curve (reference:stack overflow with little modification)
y_train_pred_proba=clf.predict_proba(train_tf_idf[:20000]) #storing score of x
```

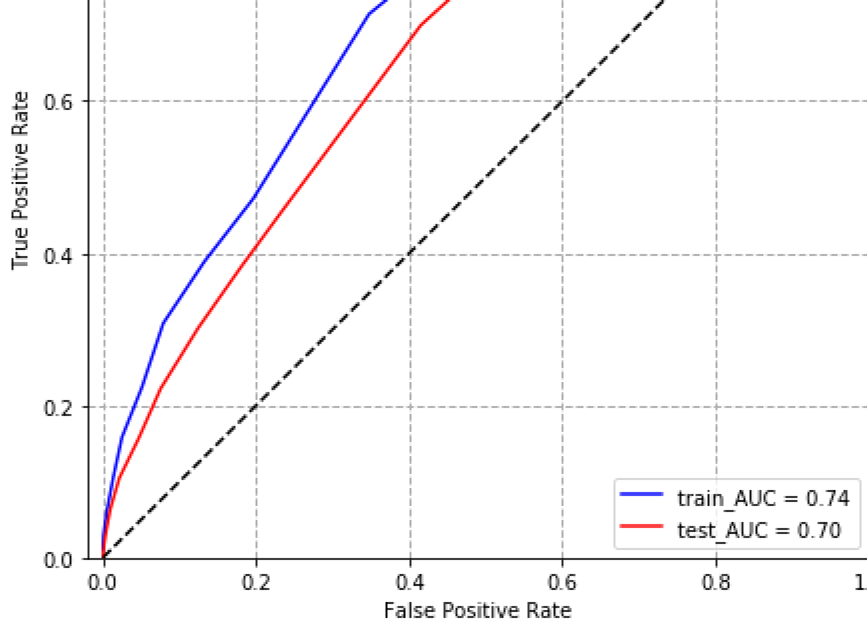
```
_train for ROC-AUC score

train_fpr, train_tpr, train_threshold =roc_curve(y_train[:60000], y_train_pred
_proba[:,1])
test_fpr, test_tpr, test_threshold =roc_curve(y_test[:20000], y_pred_proba[:,
1])

roc_auc_train = roc_auc_score(y_train[:60000],y_train_pred_proba[:,1])
roc_auc_test = roc_auc_score(y_test[:20000],y_pred_proba[:,1])
'''
plt.figure(figsize=(7,7))
plt.title('Receiver Operating Characteristic')
plt.plot(train_fpr, train_tpr, 'b', label = 'train_AUC = %0.2f' % roc_auc_trai
n)
plt.plot(test_fpr, test_tpr, 'r', label = 'test_AUC = %0.2f' % roc_auc_test)
plt.legend(loc = 'lower right')
plt.plot([0, 1], [0, 1],'k--')
plt.xlim([-0.02, 1])
plt.ylim([0, 1])
plt.ylabel('True Positive Rate')
plt.xlabel('False Positive Rate')
plt.grid(linestyle='--', linewidth=1)
plt.show()
```

```
genearalisation roc_auc on best k-value at k = 75 is 0.70

misclassification percentage is 0.15%
```




```
classification report:
             precision    recall  f1-score   support

          0       0.69      0.08      0.14      3091
          1       0.86      0.99      0.92     16909

avg / total       0.83      0.85      0.80     20000
```

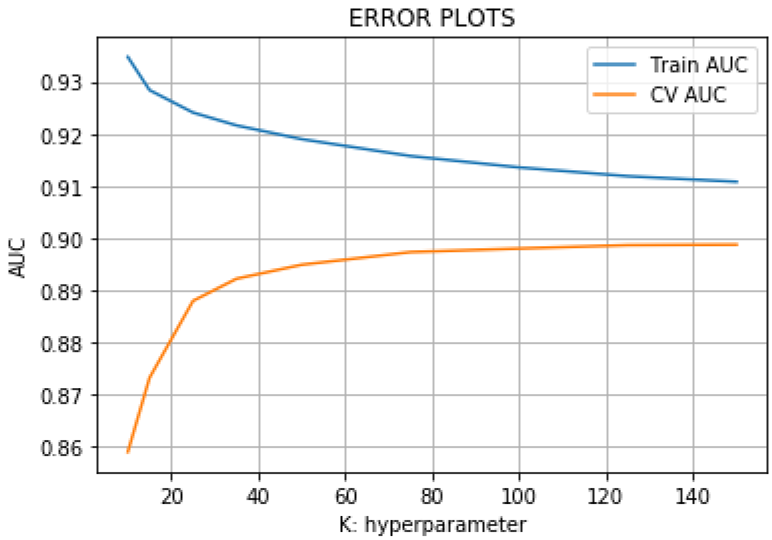




[5.2.3] Applying KNN kd-tree on AVG W2V, SET 3

In [15]:
```
# Please write all the code with proper documentation
# Please write all the code with proper documentation

from sklearn.neighbors import KNeighborsClassifier
from sklearn.metrics import roc_auc_score
import matplotlib.pyplot as plt

"""
y_true : array, shape = [n_samples] or [n_samples, n_classes]
True binary labels or binary label indicators.

y_score : array, shape = [n_samples] or [n_samples, n_classes]
Target scores, can either be probability estimates of the positive class, conf
idence values, or non-thresholded measure of
decisions (as returned by “decision_function” on some classifiers).
For binary y_true, y_score is supposed to be the score of the class with great
er label.

"""

train_auc = []
cv_auc = []
K = [10,15,25,35,50,75,100,125,150]
for i in K:
    neigh = KNeighborsClassifier(n_neighbors=i,algorithm='kd_tree',n_jobs=-1)
    neigh.fit(train_w2v[:60000], y_train[:60000])
    # roc_auc_score(y_true, y_score) the 2nd parameter should be probability e
stimates of the positive class
    # not the predicted outputs


    y_train_pred = []
    for i in range(0, len(train_w2v[:60000]), 1000):
        y_train_pred.extend(neigh.predict_proba(train_w2v[i:i+1000])[:,1])

    y_cv_pred = []
    for i in range(0, len(cv_w2v[:20000]), 1000):
        y_cv_pred.extend(neigh.predict_proba(cv_w2v[i:i+1000])[:,1])
```

```
    train_auc.append(roc_auc_score(y_train[:60000],y_train_pred))
    cv_auc.append(roc_auc_score(y_cv[:20000], y_cv_pred))



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

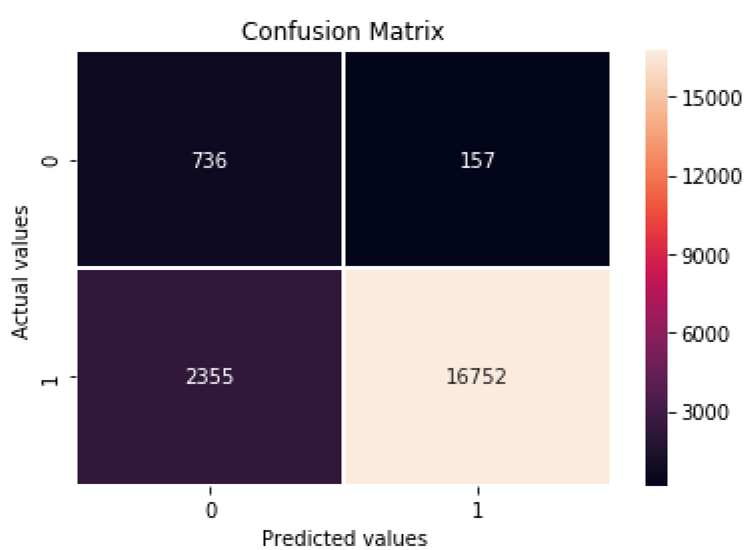


In [20]:

```
# Test dataset

#using optimum_k to find generalistion ROC AUC accuracy

k=125 #optimum 'K'

clf=KNeighborsClassifier(n_neighbors=k,weights='uniform', algorithm='kd_tree')
clf.fit(train_w2v[:60000],y_train[:60000])


y_pred = []
for i in range(0, test_w2v[:20000].shape[0], 1000):
    y_pred.extend(clf.predict(test_w2v[i:i+1000]))

y_pred_proba = []
for i in range(0, test_w2v[:20000].shape[0], 1000):
    y_pred_proba.extend(clf.predict_proba(test_w2v[i:i+1000])[:,1])



accuracy=accuracy_score(y_test[:20000],y_pred)
roc_auc=roc_auc_score(y_test[:20000],y_pred_proba)
print(f'\ngenearalisation roc_auc on best k-value at k = {k} is {roc_auc:.2f}'
)
print(f'\nmisclassification percentage is {(1-accuracy):.2f}%')


#ploting confusion matrix
```

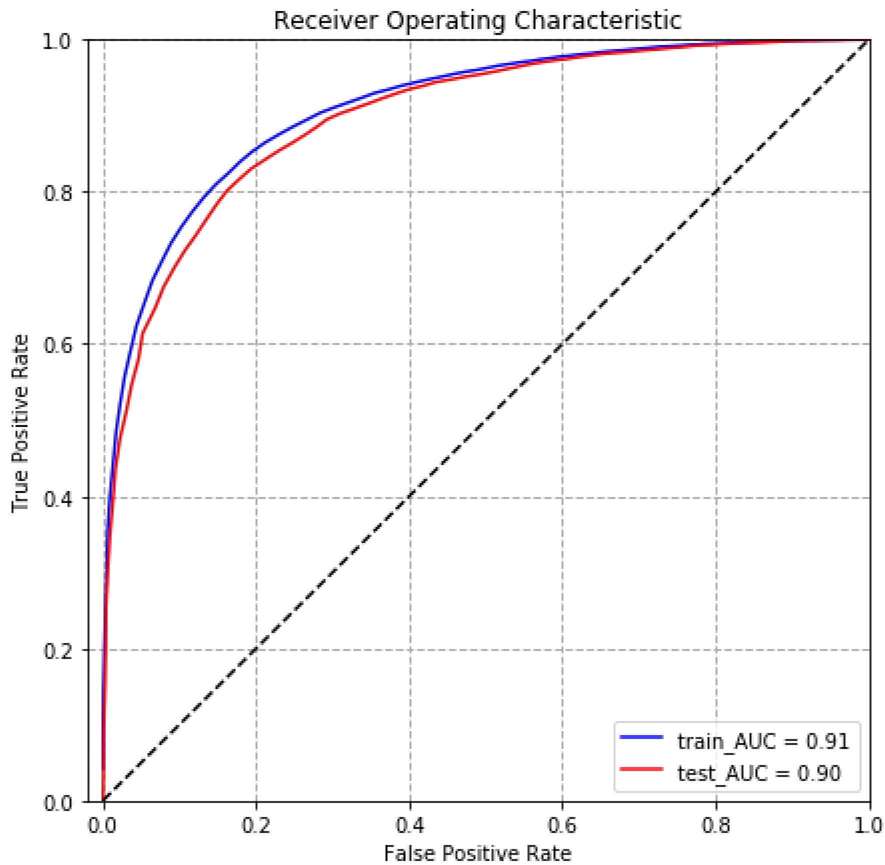
```
sn.heatmap(confusion_matrix(y_pred,y_test[:20000]),annot=True, fmt="d",linewidths=.5)
plt.title('Confusion Matrix')
plt.xlabel('Predicted values')
plt.ylabel('Actual values')
plt.show()
print("\n\nclassification report:\n",classification_report(y_test[:20000],y_pred))


# ROC Curve (reference:stack overflow with little modification)

y_train_pred_proba = []
for i in range(0, train_w2v[:60000].shape[0], 1000):
    y_train_pred_proba.extend(clf.predict_proba(train_w2v[i:i+1000])[:,1])


train_fpr, train_tpr, train_threshold =roc_curve(y_train[:60000], y_train_pred_proba)
test_fpr, test_tpr, test_threshold =roc_curve(y_test[:20000], y_pred_proba)

roc_auc_train = roc_auc_score(y_train[:60000],y_train_pred_proba)
roc_auc_test = roc_auc_score(y_test[:20000],y_pred_proba)


plt.figure(figsize=(7,7))
plt.title('Receiver Operating Characteristic')
plt.plot(train_fpr, train_tpr, 'b', label = 'train_AUC = %0.2f' % roc_auc_train)
plt.plot(test_fpr, test_tpr, 'r', label = 'test_AUC = %0.2f' % roc_auc_test)
plt.legend(loc = 'lower right')
plt.plot([0, 1], [0, 1],'k--')
plt.xlim([-0.02, 1])
plt.ylim([0, 1])
plt.ylabel('True Positive Rate')
plt.xlabel('False Positive Rate')
plt.grid(linestyle='--', linewidth=1)
plt.show()
```

```
genearalisation roc_auc on best k-value at k = 125 is 0.90

misclassification percentage is 0.13%
```




```
classification report:
```

```
             precision    recall  f1-score   support

          0       0.82      0.24      0.37      3091
          1       0.88      0.99      0.93     16909

avg / total       0.87      0.87      0.84     20000
```

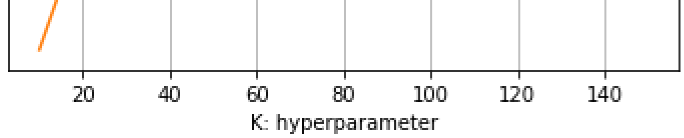


[5.2.4] Applying KNN kd-tree on TFIDF W2V, SET 4

In [16]:

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

from sklearn.neighbors import KNeighborsClassifier
from sklearn.metrics import roc_auc_score
import matplotlib.pyplot as plt

"""
y_true : array, shape = [n_samples] or [n_samples, n_classes]
True binary labels or binary label indicators.

y_score : array, shape = [n_samples] or [n_samples, n_classes]
Target scores, can either be probability estimates of the positive class, conf
idence values, or non-thresholded measure of
decisions (as returned by "decision_function" on some classifiers).
```

```
For binary y_true, y_score is supposed to be the score of the class with great
er label.

"""

train_auc = []
cv_auc = []
K = [10,15,25,35,50,75,100,125,150]
for i in K:
    neigh = KNeighborsClassifier(n_neighbors=i,algorithm='kd_tree',n_jobs=-1)
    neigh.fit(train_tf_idf_w2v[:60000], y_train[:60000])
    # roc_auc_score(y_true, y_score) the 2nd parameter should be probability e
stimates of the positive class
    # not the predicted outputs


    y_train_pred = []
    for i in range(0, len(train_tf_idf_w2v[:60000]), 1000):
        y_train_pred.extend(neigh.predict_proba(train_tf_idf_w2v[i:i+1000])[:,
1])

    y_cv_pred = []
    for i in range(0, len(cv_tf_idf_w2v[:20000]), 1000):
        y_cv_pred.extend(neigh.predict_proba(cv_tf_idf_w2v[i:i+1000])[:,1])


    train_auc.append(roc_auc_score(y_train[:60000],y_train_pred))
    cv_auc.append(roc_auc_score(y_cv[:20000], y_cv_pred))



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



In [ ]:
```
# Test dataset

#using optimum_k to find generalistion ROC AUC accuracy

k=125 #optimum 'K'

clf=KNeighborsClassifier(n_neighbors=k,weights='uniform', algorithm='kd_tree')
clf.fit(train_tf_idf_w2v[:60000],y_train[:60000])


y_pred = []
for i in range(0, test_tf_idf_w2v[:20000].shape[0], 1000):
    y_pred.extend(clf.predict(test_tf_idf_w2v[i:i+1000]))

y_pred_proba = []
for i in range(0, test_tf_idf_w2v[:20000].shape[0], 1000):
    y_pred_proba.extend(clf.predict_proba(test_tf_idf_w2v[i:i+1000])[:,1])



accuracy=accuracy_score(y_test[:20000],y_pred)
roc_auc=roc_auc_score(y_test[:20000],y_pred_proba)
print(f'\ngenearalisation roc_auc on best k-value at k = {k} is {roc_auc:.2f}')
print(f'\nmisclassification percentage is {(1-accuracy):.2f}%')


#ploting confusion matrix
sn.heatmap(confusion_matrix(y_pred,y_test[:20000]),annot=True, fmt="d",linewidths=.5)
plt.title('Confusion Matrix')
plt.xlabel('Predicted values')
plt.ylabel('Actual values')
plt.show()
print("\n\nclassification report:\n",classification_report(y_test[:20000],y_pred))


# ROC Curve (reference:stack overflow with little modification)

y_train_pred_proba = []
for i in range(0, train_tf_idf_w2v[:60000].shape[0], 1000):
    y_train_pred_proba.extend(clf.predict_proba(train_tf_idf_w2v[i:i+1000])[:,1])


train_fpr, train_tpr, train_threshold =roc_curve(y_train[:60000], y_train_pred_proba)
test_fpr, test_tpr, test_threshold =roc_curve(y_test[:20000], y_pred_proba)

roc_auc_train = roc_auc_score(y_train[:60000],y_train_pred_proba)
roc_auc_test = roc_auc_score(y_test[:20000],y_pred_proba)


plt.figure(figsize=(7,7))
plt.title('Receiver Operating Characteristic')
```

```
plt.plot(train_fpr, train_tpr, 'b', label = 'train_AUC = %0.2f' % roc_auc_trai
n)
plt.plot(test_fpr, test_tpr, 'r', label = 'test_AUC = %0.2f' % roc_auc_test)
plt.legend(loc = 'lower right')
plt.plot([0, 1], [0, 1],'k--')
plt.xlim([-0.02, 1])
plt.ylim([0, 1])
plt.ylabel('True Positive Rate')
plt.xlabel('False Positive Rate')
plt.grid(linestyle='--', linewidth=1)
plt.show()
```

[6] Conclusions

In [3]:

```
from prettytable import PrettyTable

x = PrettyTable()

x.field_names = ["text featurization", "optimum_k","accuracy(%age)", "ROC AUC
(%age) ", "algorithm-KNN",]

x.add_row(["BOW", 125, 85,74, "brute"])
x.add_row(["TF_IDF", 35, 85,66, "brute"])
x.add_row(["W2V", 75, 88,90, "brute"])
x.add_row(["Tf_IDF W2V",125,87, 87, "brute"])
x.add_row(["BOW", 150, 85,69, "kd_tree"])
x.add_row(["TF_IDF", 75, 85,70,"kd_tree"])
x.add_row(["W2V",125, 87, 90,"kd_tree"])
x.add_row(["Tf_IDF W2V", 125, 87, 87,"kd_tree"])

print(x)
```

```
+--------------------+-----------+----------------+----------------+--------
-------+
| text featurization | optimum_k | accuracy(%age) | ROC AUC(%age)  | algorit
hm-KNN |
+--------------------+-----------+----------------+----------------+--------
-------+
|        BOW         |    125    |       85       |       74       |      bru
te     |
|       TF_IDF       |     35    |       85       |       66       |      bru
te     |
|        W2V         |     75    |       88       |       90       |      bru
te     |
|     Tf_IDF W2V     |    125    |       87       |       87       |      bru
te     |
|        BOW         |    150    |       85       |       69       |     kd_t
ree    |
|       TF_IDF       |     75    |       85       |       70       |     kd_t
ree    |
|        W2V         |    125    |       87       |       90       |     kd_t
ree    |
|     Tf_IDF W2V     |    125    |       87       |       87       |     kd_t
ree    |
+--------------------+-----------+----------------+----------------+--------
-------+
```

Observation:

1. Time complexity and space complexity of KNN is large therefore it takes quite much time to execute with large dimension

2. W2V Text Featurization tend to works best using KNN algorithm with ROC_AUC=90 %

3. we need to try other ML algorithms as well and compare results.

In [ ]: