03 Amazon Fine Food Reviews Analysis_KNN

April 11, 2019

1 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.

2 [1]. Reading Data

2.1 [1.1] Loading the data

The dataset is available in two forms 1. .csv file 2. SQLite Database

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

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

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


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

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

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

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

from tqdm import tqdm
import os

from sklearn.model_selection import train_test_split
from sklearn.model_selection import cross_val_score
from sklearn.neighbors import KNeighborsClassifier
from sklearn.metrics import classification_report, confusion_matrix, roc_auc_score
from sklearn.model_selection import GridSearchCV
from imblearn.over_sampling import SMOTE
from imblearn.over_sampling import RandomOverSampler
from collections import Counter
```

In [2]:
```
# using SQLite Table to read data.
```

```
con = sqlite3.connect(os.path.join( os.getcwd(), '..', '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 5
# for tsne assignment you can take 5k data points

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

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

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

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

Out[2]:

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

   HelpfulnessNumerator  HelpfulnessDenominator  Score        Time  \
0                     1                       1      1  1303862400
1                     0                       0      0  1346976000
2                     1                       1      1  1219017600

                 Summary                                               Text
0  Good Quality Dog Food  I have bought several of the Vitality canned d...
1      Not as Advertised  Product arrived labeled as Jumbo Salted Peanut...
2  "Delight" says it all  This is a confection that has been around a fe...
```

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  \
        0  #oc-R115TNMSPFT9I7  B007Y59HVM                     Breyton  1331510400      2
        1  #oc-R11D9D7SHXIJB9  B005HG9ET0  Louis E. Emory "hoppy"  1342396800      5
        2  #oc-R11DNU2NBKQ23Z  B007Y59HVM            Kim Cieszykowski  1348531200      1
        3  #oc-R11O5J5ZVQE25C  B005HG9ET0               Penguin Chick  1346889600      5
        4  #oc-R12KPBODL2B5ZD  B007OSBE1U     Christopher P. Presta  1348617600      1

                                                        Text  COUNT(*)
        0  Overall its just OK when considering the price...         2
        1  My wife has recurring extreme muscle spasms, u...         3
        2  This coffee is horrible and unfortunately not ...         2
        3  This will be the bottle that you grab from the...         3
        4  I didnt like this coffee. Instead of telling y...         2
```

```
In [5]: display[display['UserId']=='AZY10LLTJ71NX']
```

```
Out[5]:               UserId   ProductId                     ProfileName        Time  \
        80638  AZY10LLTJ71NX  B006P7E5ZI  undertheshrine "undertheshrine"  1334707200

               Score                                               Text  COUNT(*)
        80638      5  I was recommended to try green tea extract to ...         5
```

```
In [6]: display['COUNT(*)'].sum()
```

```
Out[6]: 393063
```

3 [2] Exploratory Data Analysis

3.1 [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  \
        0   78445  B000HDL1RQ  AR5J8UI46CURR  Geetha Krishnan                     2
        1  138317  B000HDOPYC  AR5J8UI46CURR  Geetha Krishnan                     2
        2  138277  B000HDOPYM  AR5J8UI46CURR  Geetha Krishnan                     2
        3   73791  B000HDOPZG  AR5J8UI46CURR  Geetha Krishnan                     2
        4  155049  B000PAQ75C  AR5J8UI46CURR  Geetha Krishnan                     2

           HelpfulnessDenominator  Score        Time  \
        0                       2      5  1199577600
        1                       2      5  1199577600
        2                       2      5  1199577600
        3                       2      5  1199577600
        4                       2      5  1199577600

                                     Summary  \
        0  LOACKER QUADRATINI VANILLA WAFERS
        1  LOACKER QUADRATINI VANILLA WAFERS
        2  LOACKER QUADRATINI VANILLA WAFERS
        3  LOACKER QUADRATINI VANILLA WAFERS
        4  LOACKER QUADRATINI VANILLA WAFERS

                                                        Text
        0  DELICIOUS WAFERS. I FIND THAT EUROPEAN WAFERS ...
        1  DELICIOUS WAFERS. I FIND THAT EUROPEAN WAFERS ...
        2  DELICIOUS WAFERS. I FIND THAT EUROPEAN WAFERS ...
        3  DELICIOUS WAFERS. I FIND THAT EUROPEAN WAFERS ...
        4  DELICIOUS WAFERS. I FIND THAT EUROPEAN WAFERS ...
```

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=Fals
```

```
In [9]: #Deduplication of entries
        final=sorted_data.drop_duplicates(subset={"UserId","ProfileName","Time","Text"}, keep=
        final.shape
```

Out[9]: (364173, 10)

In [10]:
```
#Checking to see how much % of data still remains
(final['Id'].size*1.0)/(filtered_data['Id'].size*1.0)*100
```

Out[10]: 69.25890143662969

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

In [11]:
```
display= pd.read_sql_query("""
SELECT *
FROM Reviews
WHERE Score != 3 AND Id=44737 OR Id=64422
ORDER BY ProductID
""", con)

display.head()
```

Out[11]:
```
      Id   ProductId          UserId              ProfileName  \
0  64422  B000MIDROQ  A161DK06JJMCYF  J. E. Stephens "Jeanne"
1  44737  B001EQ55RW  A2V0I904FH7ABY                      Ram

   HelpfulnessNumerator  HelpfulnessDenominator  Score        Time  \
0                     3                       1      5  1224892800
1                     3                       2      4  1212883200

                                     Summary  \
0            Bought This for My Son at College
1  Pure cocoa taste with crunchy almonds inside

                                                Text
0  My son loves spaghetti so I didn't hesitate or...
1  It was almost a 'love at first bite' - the per...
```

In [12]:
```
final=final[final.HelpfulnessNumerator<=final.HelpfulnessDenominator]
```

In [13]:
```
#Before starting the next phase of preprocessing lets see the number of entries left
print(final.shape)

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

```
(364171, 10)
```

Out[13]:
```
1    307061
0     57110
Name: Score, dtype: int64
```

4 [3] Preprocessing

4.1 [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)
```

```
this witty little book makes my son laugh at loud. i recite it in the car as we're driving alo
==================================================
I was really looking forward to these pods based on the reviews.  Starbucks is good, but I pre
==================================================
Great ingredients although, chicken should have been 1st rather than chicken broth, the only t
==================================================
Can't do sugar.  Have tried scores of SF Syrups.  NONE of them can touch the excellence of thi
==================================================
```

```
In [15]: # remove urls from text python: https://stackoverflow.com/a/40823105/4084039
         sent_0 = re.sub(r"http\S+", "", sent_0)
         sent_1000 = re.sub(r"http\S+", "", sent_1000)
```

```
sent_150 = re.sub(r"http\S+", "", sent_1500)
sent_4900 = re.sub(r"http\S+", "", sent_4900)

print(sent_0)
```

```
this witty little book makes my son laugh at loud. i recite it in the car as we're driving alo
```

```
In [16]: # https://stackoverflow.com/questions/16206380/python-beautifulsoup-how-to-remove-all
         from bs4 import BeautifulSoup

         soup = BeautifulSoup(sent_0, 'lxml')
         text = soup.get_text()
         print(text)
         print("="*50)

         soup = BeautifulSoup(sent_1000, 'lxml')
         text = soup.get_text()
         print(text)
         print("="*50)

         soup = BeautifulSoup(sent_1500, 'lxml')
         text = soup.get_text()
         print(text)
         print("="*50)

         soup = BeautifulSoup(sent_4900, 'lxml')
         text = soup.get_text()
         print(text)
```

```
this witty little book makes my son laugh at loud. i recite it in the car as we're driving alo
==================================================
I was really looking forward to these pods based on the reviews.  Starbucks is good, but I pre
==================================================
Great ingredients although, chicken should have been 1st rather than chicken broth, the only t
==================================================
Can't do sugar.  Have tried scores of SF Syrups.  NONE of them can touch the excellence of thi
```

```
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 ingredients although, chicken should have been 1st rather than chicken broth, the only t
==================================================
```

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)
```

```
this witty little book makes my son laugh at loud. i recite it in the car as we're driving alo
```

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 ingredients although chicken should have been 1st rather than chicken broth the only thi
```

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', 'ourselve
            "you'll", "you'd", 'your', 'yours', 'yourself', 'yourselves', 'he', 'him'
            'she', "she's", 'her', 'hers', 'herself', 'it', "it's", 'its', 'itself',
            'theirs', 'themselves', 'what', 'which', 'who', 'whom', 'this', 'that', "
            'am', 'is', 'are', 'was', 'were', 'be', 'been', 'being', 'have', 'has', '
            'did', 'doing', 'a', 'an', 'the', 'and', 'but', 'if', 'or', 'because', 'a
            'at', 'by', 'for', 'with', 'about', 'against', 'between', 'into', 'throug
            'above', 'below', 'to', 'from', 'up', 'down', 'in', 'out', 'on', 'off', '
            'then', 'once', 'here', 'there', 'when', 'where', 'why', 'how', 'all', 'a
            'most', 'other', 'some', 'such', 'only', 'own', 'same', 'so', 'than', 'to
            's', 't', 'can', 'will', 'just', 'don', "don't", 'should', "should've", '
            've', 'y', 'ain', 'aren', "aren't", 'couldn', "couldn't", 'didn', "didn't
```

```
                "hadn't", 'hasn', "hasn't", 'haven', "haven't", 'isn', "isn't", 'ma', 'mi
                "mustn't", 'needn', "needn't", 'shan', "shan't", 'shouldn', "shouldn't",
                'won', "won't", 'wouldn', "wouldn't"])
```

[3.2] Preprocessing Review Text and Summary

```
In [22]: preprocessed_reviews = []
         preprocessed_summary = []
         # Sampling the data and preprocessing
         def data_sampling_preprocessing(final, no_of_samples):

             final = final.sample(n=no_of_samples)

             # 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 s
                 preprocessed_reviews.append(sentance.strip())

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

                 preprocessed_summary.append(summary.strip())


             final['CleanedText'] = preprocessed_reviews #adding a column of CleanedText which
             final['CleanedText'] = final['CleanedText'].astype('str')

             final['CleanedSummary'] = preprocessed_summary #adding a column of CleanedSummary
             final['CleanedSummary'] = final['CleanedSummary'].astype('str')
```

```python
        final['Text_Summary'] = final['CleanedSummary'] + final['CleanedText']

        # # store final table into an SQlLite table for future.
        # conn = sqlite3.connect('final.sqlite')
        # c=conn.cursor()
        # conn.text_factory = str
        # final.to_sql('Reviews', conn,  schema=None, if_exists='replace', \
        #              index=True, index_label=None, chunksize=None, dtype=None)
        # conn.close()

        return final['Text_Summary'], final['Score']
```

5 [5] Assignment 3: KNN

```
<li><strong>Apply Knn(brute force version) on these feature sets</strong>
    <ul>
        <li><font color='red'>SET 1:</font>Review text, preprocessed one converted into vectors
        <li><font color='red'>SET 2:</font>Review text, preprocessed one converted into vectors
        <li><font color='red'>SET 3:</font>Review text, preprocessed one converted into vectors
        <li><font color='red'>SET 4:</font>Review text, preprocessed one converted into vectors
    </ul>
</li>
<br>
<li><strong>Apply Knn(kd tree version) on these feature sets</strong>
    <br><font color='red'>NOTE: </font>sklearn implementation of kd-tree accepts only dense ma
    <ul>
        <li><font color='red'>SET 5:</font>Review text, preprocessed one converted into vectors
        <pre>
        count_vect = CountVectorizer(min_df=10, max_features=500)
        count_vect.fit(preprocessed_reviews)
        </pre>
        </li>
        <li><font color='red'>SET 6:</font>Review text, preprocessed one converted into vectors
        <pre>
            tf_idf_vect = TfidfVectorizer(min_df=10, max_features=500)
            tf_idf_vect.fit(preprocessed_reviews)
        </pre>
        </li>
        <li><font color='red'>SET 3:</font>Review text, preprocessed one converted into vectors
        <li><font color='red'>SET 4:</font>Review text, preprocessed one converted into vectors
    </ul>
</li>
<br>
<li><strong>The hyper paramter tuning(find best K)</strong>
    <ul>
<li>Find the best hyper parameter which will give the maximum <a href='https://www.appliedaico
```

```
<li>Find the best hyper paramter using k-fold cross validation or simple cross validation data
<li>Use gridsearch cv or randomsearch cv or you can also write your own for loops to do this ta
    </ul>
</li>
<br>
<li>
<strong>Representation of results</strong>
    <ul>
<li>You need to plot the performance of model both on train data and cross validation data for
<img src='train_cv_auc.JPG' width=300px></li>
<li>Once after you found the best hyper parameter, you need to train your model with it, and fi
<img src='train_test_auc.JPG' width=300px></li>
<li>Along with plotting ROC curve, you need to print the <a href='https://www.appliedaicourse.c
<img src='confusion_matrix.png' width=300px></li>
    </ul>
</li>
<br>
<li><strong>Conclusion</strong>
    <ul>
<li>You need to summarize the results at the end of the notebook, summarize it in the table fo
    <img src='summary.JPG' width=400px>
</li>
    </ul>
```

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-leakage, 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.

Function to create a pickle file and read from pickle file

5.1 [5.1] Applying KNN brute force

In [23]:
```python
# Sort 'Time' column
final = final.sort_values(by='Time', ascending=True)
```

In [24]:
```python
# Source: https://docs.python.org/3/library/pickle.html

# Saving data to pickle file
def topicklefile(obj, file_name):
    pickle.dump(obj,open(file_name+'.pkl', 'wb'))
```

In [25]:
```python
# Data from pickle file
def frompicklefile(file_name):
```

```
        data = pickle.load(open(file_name+'.pkl', 'rb'))
        return data
```

In [26]:
```
# Source: https://scikit-learn.org/stable/modules/generated/sklearn.model_selection.t
# https://beckernick.github.io/oversampling-modeling/
# Train Test split for train and test data
def data_split(final, no_of_samples):
    X, y = data_sampling_preprocessing(final, no_of_samples)
    # split the data set into train and test
    X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.3, random_s

    topicklefile(X_train, 'X_train')
    topicklefile(X_test, 'X_test')
    topicklefile(y_train, 'y_train')
    topicklefile(y_test, 'y_test')

    return X_train, X_test, y_train, y_test
```

In [27]:
```
def apply_avgw2v_train_test(X_train, X_test):

    # Training own Word2Vec model using your own text corpus
    list_of_sent_train = []
    for sent in X_train:#final['Text_Summary'].values:
        list_of_sent_train.append(sent.split())
    list_of_sent_test = []
    for sent in X_test:#final['Text_Summary'].values:
        list_of_sent_test.append(sent.split())

    # min_count = 5 considers only words that occured atleast 5 times
    w2v_model=Word2Vec(list_of_sent_train,min_count=5,size=50, workers=8)

    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])

    # compute average word2vec for each review for train data
    avgw2v_train = []; # the avg-w2v for each sentence/review is stored in this list
    for sent in tqdm(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 w2v_words:
                vec = w2v_model.wv[word]
                sent_vec += vec
                cnt_words += 1
        if cnt_words != 0:
            sent_vec /= cnt_words
        avgw2v_train.append(sent_vec)
```

```
#     print(len(avgw2v_train))
#     print(len(avgw2v_train[0]))

    # compute average word2vec for each review for test data
    avgw2v_test = []; # the avg-w2v for each sentence/review is stored in this list
    for sent in tqdm(list_of_sent_test): # for each review/sentence
        sent_vec = np.zeros(50) # as word vectors are of zero length
        cnt_words =0; # num of words with a valid vector in the sentence/review
        for word in sent: # for each word in a review/sentence
            if word in w2v_words:
                vec = w2v_model.wv[word]
                sent_vec += vec
                cnt_words += 1
        if cnt_words != 0:
            sent_vec /= cnt_words
        avgw2v_test.append(sent_vec)
#     print(len(avgw2v_test))
#     print(len(avgw2v_test[0]))

    return avgw2v_train, avgw2v_test
```

In [28]:
```
def apply_tfidfw2v_train_test(X_train, X_test):

    # Training own Word2Vec model using your own text corpus
    list_of_sent_train = []
    for sent in X_train:#final['Text_Summary'].values:
        list_of_sent_train.append(sent.split())
    list_of_sent_test = []
    for sent in X_test:#final['Text_Summary'].values:
        list_of_sent_test.append(sent.split())

    # min_count = 5 considers only words that occured atleast 5 times
    w2v_model=Word2Vec(list_of_sent_train,min_count=5,size=50, workers=16)

    w2v_words = list(w2v_model.wv.vocab)

    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_)))

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

    tfidfw2v_train = []; # the tfidf-w2v for each sentence/review is stored in this l
    row=0;
```

```
for sent in tqdm(list_of_sent_train): # for each review/sentence
    sent_vec = np.zeros(50) # as word vectors are of zero length
    weight_sum =0; # num of words with a valid vector in the sentence/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
    tfidfw2v_train.append(sent_vec)
    row += 1


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

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

tfidfw2v_test = []; # the tfidf-w2v for each sentence/review is stored in this li
row=0;
for sent in tqdm(list_of_sent_test): # for each review/sentence
    sent_vec = np.zeros(50) # as word vectors are of zero length
    weight_sum =0; # num of words with a valid vector in the sentence/review
    for word in sent: # for each word in a review/sentence
        if word in w2v_words and word in tfidf_feat:
            vec = w2v_model.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
    tfidfw2v_test.append(sent_vec)
    row += 1
```

```
        return tfidfw2v_train, tfidfw2v_test
```

In [29]:
```
# Applying BOW on train and test data and creating the
from sklearn.preprocessing import StandardScaler
from scipy.sparse import hstack

#Standardize 'bow_train' data features by removing the mean and scaling to unit varia
std_scalar1 = StandardScaler(copy=True, with_mean=False, with_std=True)
std_scalar2 = StandardScaler(copy=True, with_mean=True, with_std=True)

#Source: https://medium.com/@saeedAR/smote-and-near-miss-in-python-machine-learning-i
sm = SMOTE()

def apply_vectorizers_train_test(algo, model_name):

    if algo == 'brute' and model_name == 'BOW':
        train_data, test_data, y_train, y_test = data_split(final, 50000)

        topicklefile(train_data,'train_data')
        topicklefile(test_data,'test_data')
        topicklefile(y_train,'y_train')
        topicklefile(y_test,'y_test')

    elif algo == 'kd_tree' and model_name == 'BOW':
        train_data, test_data, y_train, y_test = data_split(final, 20000)

        topicklefile(train_data,'train_data')
        topicklefile(test_data,'test_data')
        topicklefile(y_train,'y_train')
        topicklefile(y_test,'y_test')

    else:
        print('else statement')
        train_data = frompicklefile('train_data')
        test_data = frompicklefile('test_data')
        y_train = frompicklefile('y_train')
        y_test = frompicklefile('y_test')

        print(train_data.shape)
        print(y_train.shape)


    if model_name == 'BOW':
        #Applying BoW on Train data
        count_vect = CountVectorizer()

        #Applying BoW on Test data
        train_vect = count_vect.fit_transform(train_data)
```

```python
        #Applying BoW on Test data similar to the bow_train data
        test_vect = count_vect.transform(test_data)

        # Standardise train data
        train_vect = std_scalar1.fit_transform(train_vect)
        # Standardize the unseen bow_test data
        test_vect = std_scalar1.transform(test_vect)

        train_vect, bow_y_train = sm.fit_sample(train_vect, y_train)

        topicklefile(train_vect, 'train_vect')
        topicklefile(bow_y_train, 'bow_y_train')
        topicklefile(test_vect, 'test_vect')


        print("'train_vect' and 'test_vect' are the pickle files.")
        return count_vect

    elif model_name == 'TF-IDF':
        #Applying TF-IDF on Train data
        count_vect = TfidfVectorizer(ngram_range=(1,2), min_df=10)

        #Applying BoW on Test data
        train_vect = count_vect.fit_transform(train_data)

        #Applying BoW on Test data similar to the bow_train data
        test_vect = count_vect.transform(test_data)

        # Standardise train data
        train_vect = std_scalar1.fit_transform(train_vect)
        # Standardize the unseen bow_test data
        test_vect = std_scalar1.transform(test_vect)

        train_vect, tfidf_y_train = sm.fit_sample(train_vect, y_train)

        topicklefile(train_vect, 'train_vect')
        topicklefile(tfidf_y_train, 'tfidf_y_train')
        topicklefile(test_vect, 'test_vect')

        print("'train_vect' and 'test_vect' are the pickle files.")
        return count_vect

    elif model_name == 'AvgW2V':
        train_vect, test_vect = apply_avgw2v_train_test(train_data, test_data)

        # Standardise train data
        train_vect = std_scalar2.fit_transform(train_vect)
```

```
        # Standardize the unseen bow_test data
        test_vect = std_scalar2.transform(test_vect)

        train_vect, avgw2v_y_train = sm.fit_sample(train_vect, y_train)

        topicklefile(train_vect, 'train_vect')
        topicklefile(avgw2v_y_train, 'avgw2v_y_train')
        topicklefile(test_vect, 'test_vect')
        print("'train_vect' and 'test_vect' are the pickle files.")

    elif model_name == 'TF-IDF W2V':
        train_vect, test_vect = apply_tfidfw2v_train_test(train_data, test_data)

        # Standardise train data
        train_vect = std_scalar2.fit_transform(train_vect)
        # Standardize the unseen bow_test data
        test_vect = std_scalar2.transform(test_vect)

        train_vect, tfidfw2v_y_train = sm.fit_sample(train_vect, y_train)

        topicklefile(train_vect, 'train_vect')
        topicklefile(tfidfw2v_y_train, 'tfidfw2v_y_train')
        topicklefile(test_vect, 'test_vect')
        print("'train_vect' and 'test_vect' are the pickle files.")

    else:
        #Error Message
        print('Model specified is not valid! Please check.')
```

```
In [30]: def applying_knn(algo, k_values, train_data, y_train):

             parameters = {'n_neighbors':k_values}
             knn_clf = KNeighborsClassifier(algorithm=algo, n_jobs=-1)

             clf = GridSearchCV(knn_clf, parameters, cv=10, scoring= 'roc_auc', n_jobs=-1, retu
             clf.fit(train_data, y_train)

             k_optimal = clf.best_params_.get('n_neighbors')

             train_auc= clf.cv_results_['mean_train_score']
             cv_auc = clf.cv_results_['mean_test_score']

             return clf, k_optimal, train_auc, cv_auc
```

```
In [31]: def train_cv_error_plot(k_values, train_auc, cv_auc):
             plt.plot(k_values, train_auc, label='Train AUC')

             plt.plot(k_values, cv_auc, label='CV AUC')
```

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

```
In [32]: def optimal_knn(algo, k_optimal, train_data, y_train):
             knn_optimal = KNeighborsClassifier(n_neighbors=k_optimal, algorithm=algo, n_jobs=
             knn_optimal.fit(train_data, y_train)

             return knn_optimal
```

```
In [33]: # Confusion Matrix
         def cm_fig(knn_optimal, y_test, test_vec):
             cm = pd.DataFrame(confusion_matrix(y_test, knn_optimal.predict(test_vec)))
             # print(confusion_matrix(y_test, y_pred))

             plt.figure(1, figsize=(18,5))
             plt.subplot(121)
             plt.title("Confusion Matrix")
             sns.set(font_scale=1.4)
             sns.heatmap(cm, cmap= 'gist_earth', annot=True, annot_kws={'size':15}, fmt='g')
```

```
In [34]: #Reference: https://stackoverflow.com/questions/52910061/implementing-roc-curves-for-
         def error_plot(knn_optimal, train_vec, y_train, test_vec, y_test):
             train_fpr, train_tpr, thresholds = roc_curve(y_train, knn_optimal.predict_proba(t
             test_fpr, test_tpr, thresholds = roc_curve(y_test, knn_optimal.predict_proba(test

             plt.plot(train_fpr, train_tpr, label="train AUC = %0.3f" %auc(train_fpr, train_tp
             plt.plot(test_fpr, test_tpr, label="test AUC = %0.3f" %auc(test_fpr, test_tpr))
             plt.plot([0.0, 1.0], [0.0, 1.0],'k--')
             plt.legend()
             plt.xlabel("C: hyperparameter")
             plt.ylabel("AUC")
             plt.title("ERROR PLOTS")
             plt.show()

             return auc(test_fpr, test_tpr)
```

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

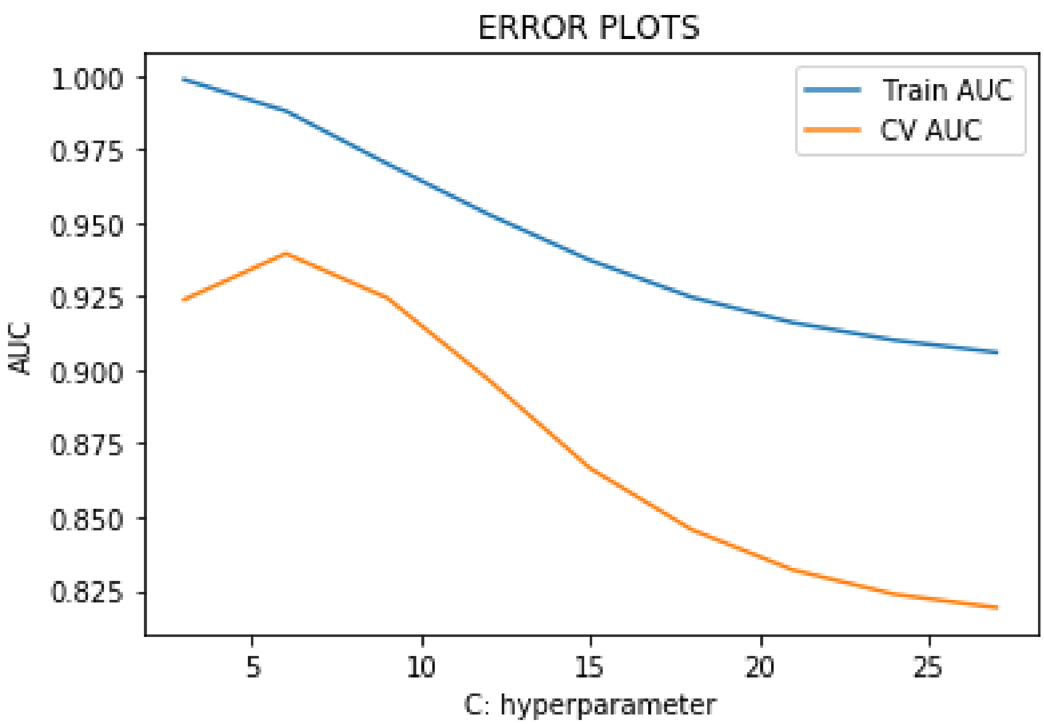
```
In [35]: count_vect = apply_vectorizers_train_test('brute','BOW')
```

```
100%|| 50000/50000 [00:19<00:00, 2554.22it/s]
100%|| 50000/50000 [00:12<00:00, 3859.11it/s]


'train_vect' and 'test_vect' are the pickle files.
```

In [36]:
```
bow_train_vect = frompicklefile('train_vect')
bow_test_vect = frompicklefile('test_vect')
bow_y_train = frompicklefile('bow_y_train')
y_test = frompicklefile('y_test')
```

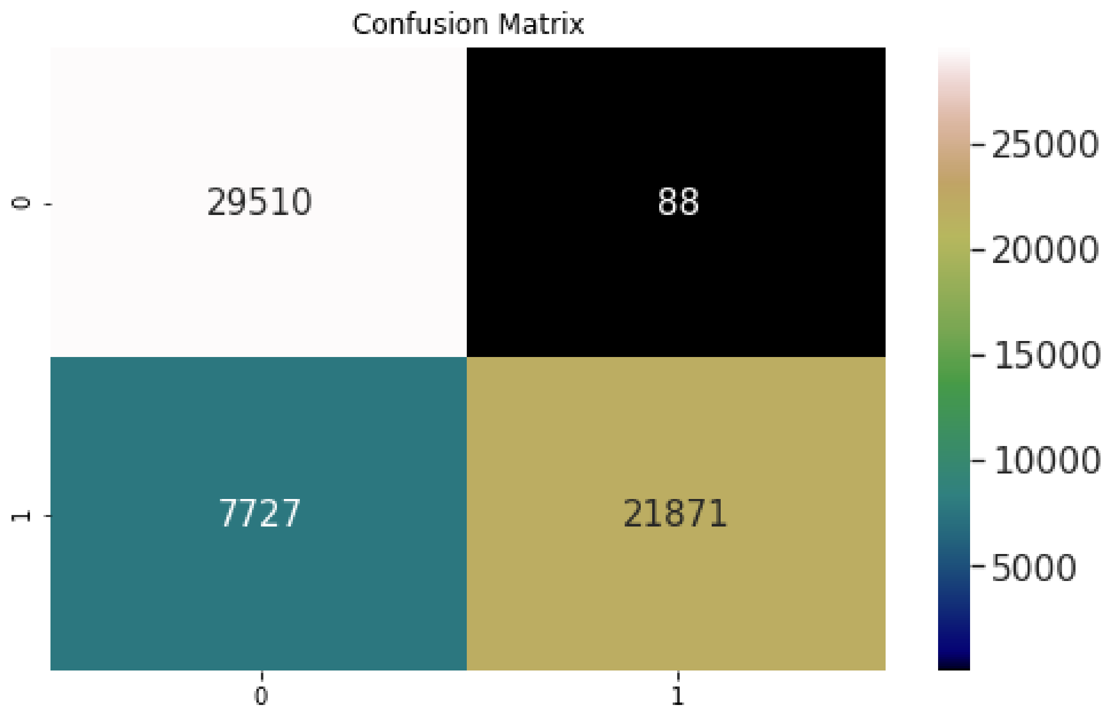
In [37]:
```
k_values = [i for i in range(3,30,3)]
clf, k_optimal, train_auc, cv_auc = applying_knn('brute', k_values, bow_train_vect, bo

k_optimal_bow1 = k_optimal

print('The optimal K is {}' .format(k_optimal_bow1))

train_cv_error_plot(k_values, train_auc, cv_auc)
```

```
The optimal K is 6
```

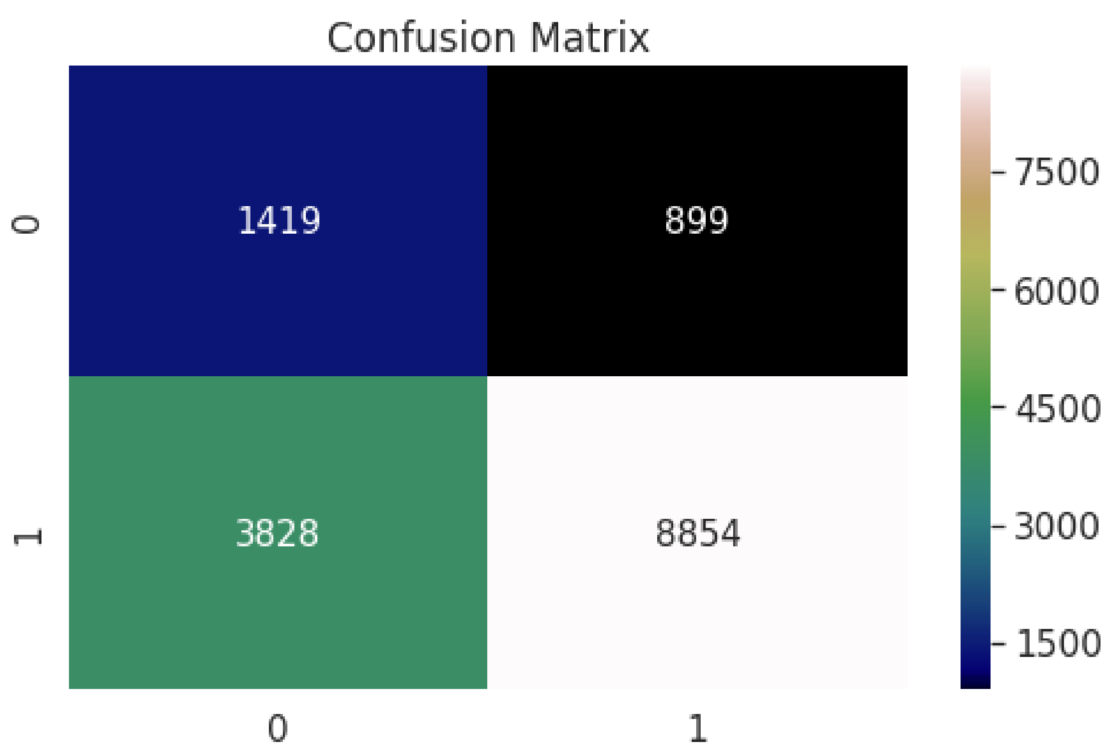


In [38]:
```
knn_optimal = optimal_knn('brute', k_optimal, bow_train_vect, bow_y_train)
```

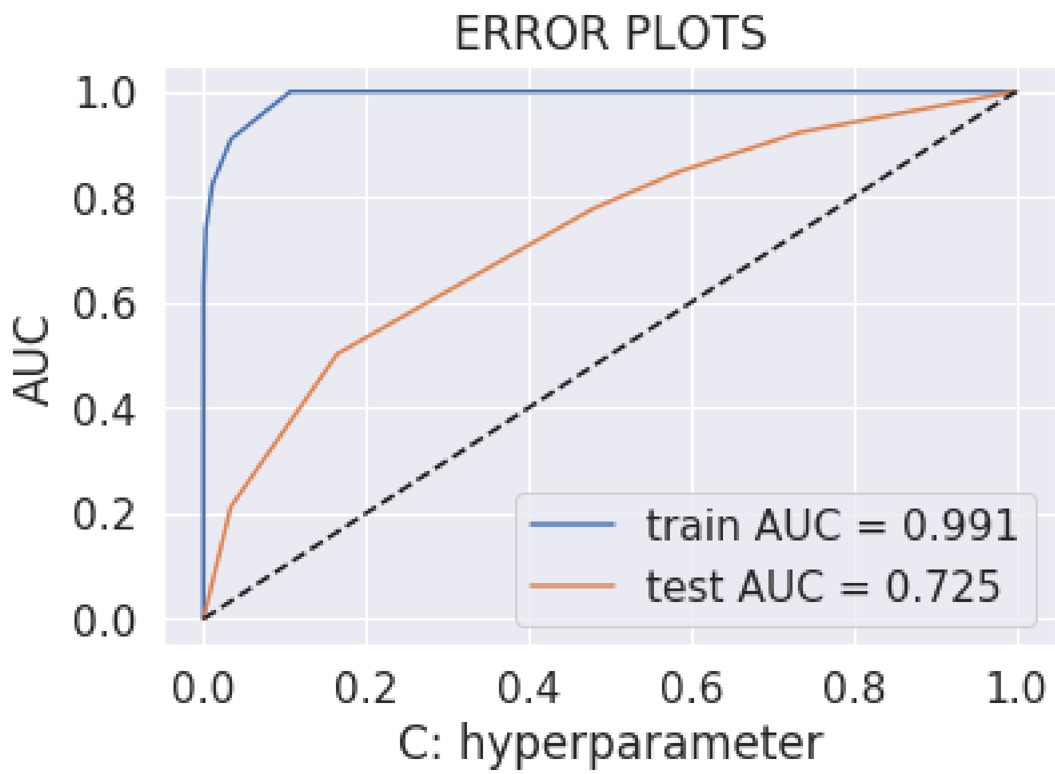
In [39]:
```
cm_fig(knn_optimal, bow_y_train, bow_train_vect)
```



In [40]: cm_fig(knn_optimal, y_test, bow_test_vect)



```
In [41]: auc_bow1 = error_plot(knn_optimal, bow_train_vect, bow_y_train, bow_test_vect, y_test)
```



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

```
In [42]: count_vect = apply_vectorizers_train_test('brute', 'TF-IDF')
```

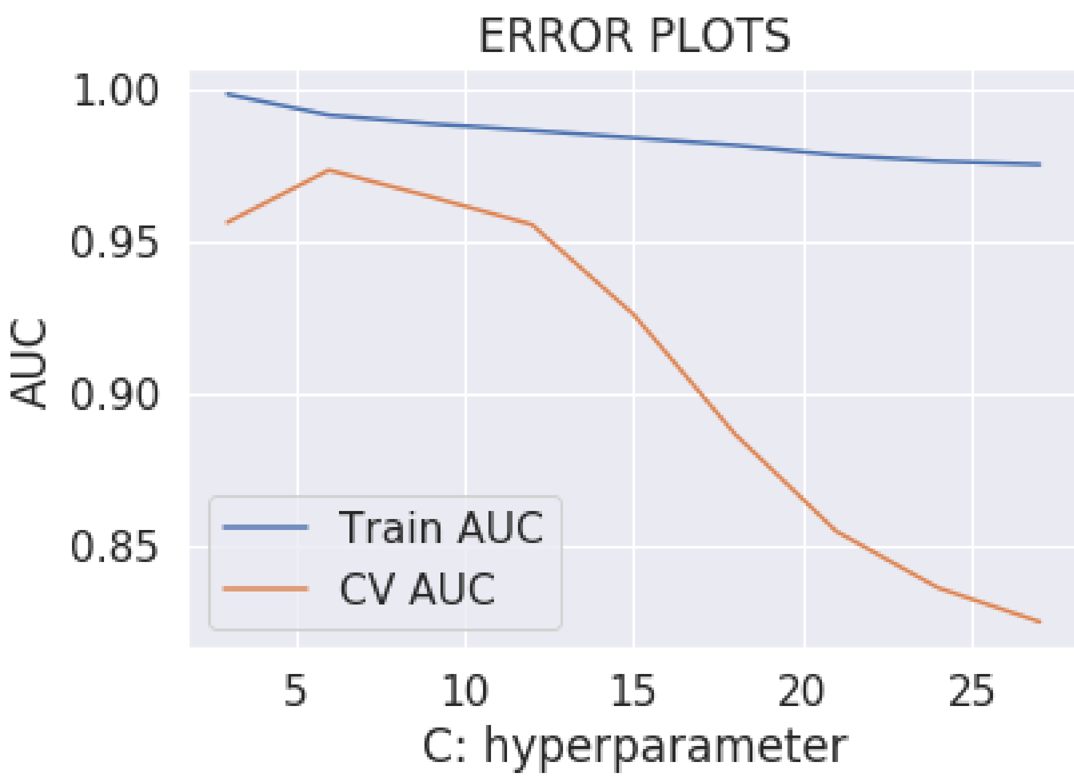
```
else statement
(35000,)
(35000,)
'train_vect' and 'test_vect' are the pickle files.
```

```
In [43]: tfidf_train_vect = frompicklefile('train_vect')
         tfidf_test_vect = frompicklefile('test_vect')
         tfidf_y_train = frompicklefile('tfidf_y_train')
         y_test = frompicklefile('y_test')
```

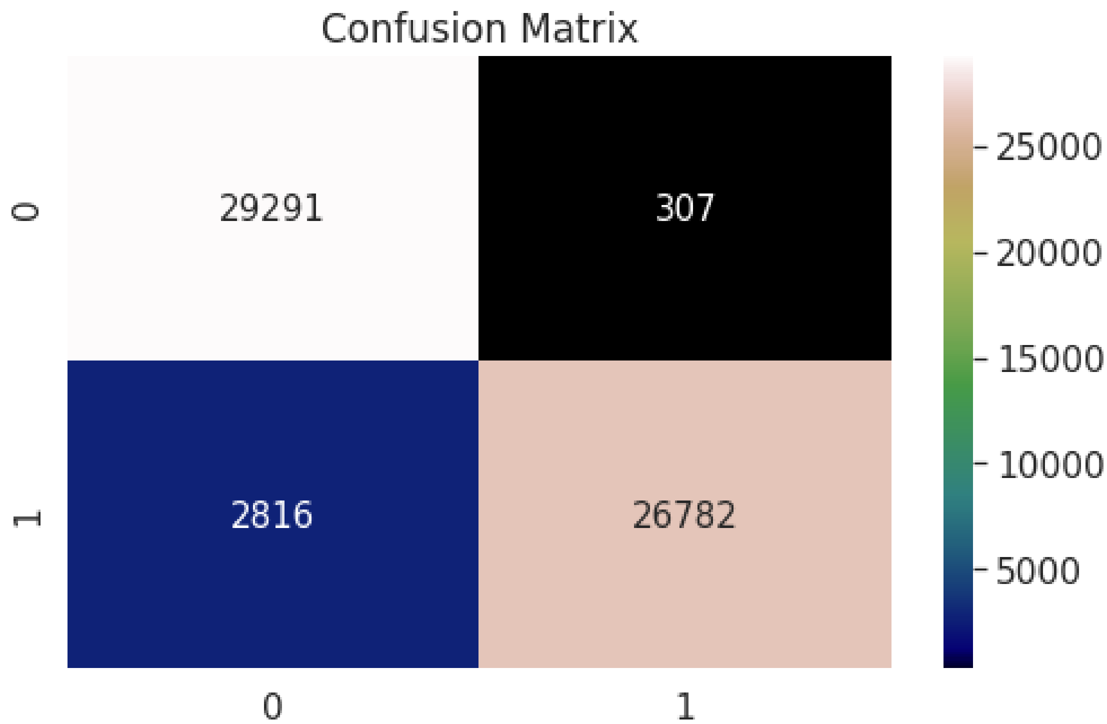
```
In [44]: k_values = [i for i in range(3,30,3)]
         clf, k_optimal, train_auc, cv_auc = applying_knn('brute', k_values, tfidf_train_vect,
```

```
k_optimal_tfidf1 = k_optimal

print('The optimal K is {}' .format(k_optimal_tfidf1))

train_cv_error_plot(k_values, train_auc, cv_auc)
```

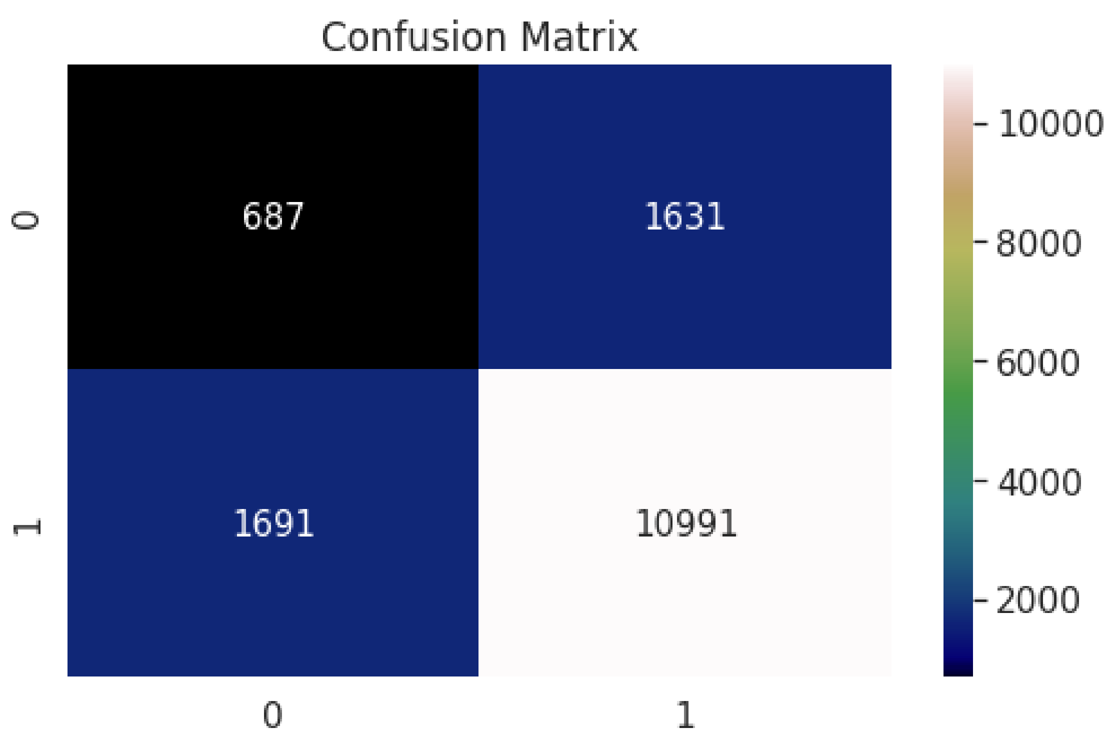
```
The optimal K is 6
```



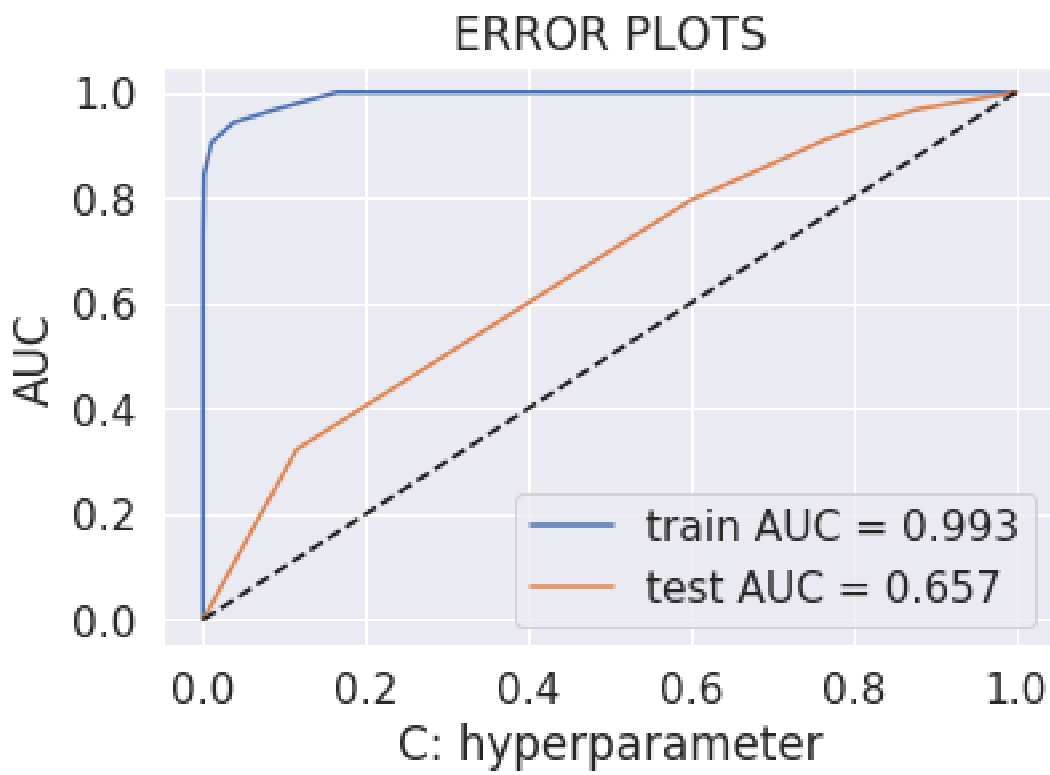

```
In [45]: knn_optimal = optimal_knn('brute', k_optimal, tfidf_train_vect, tfidf_y_train)

In [46]: cm_fig(knn_optimal, tfidf_y_train, tfidf_train_vect)
```



In [47]: `cm_fig(knn_optimal, y_test, tfidf_test_vect)`



```
In [48]: auc_tfidf1 = error_plot(knn_optimal, tfidf_train_vect, tfidf_y_train, tfidf_test_vect
```



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

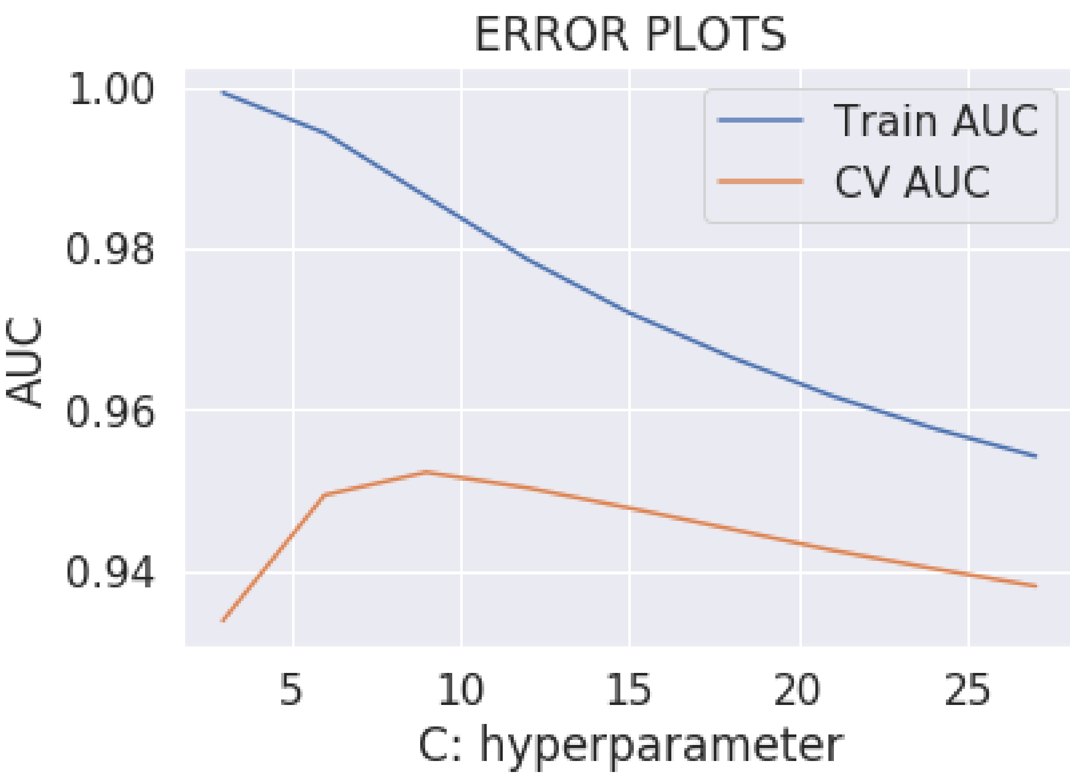
```
In [49]: count_vect = apply_vectorizers_train_test('brute', 'AvgW2V')
```

```
else statement
(35000,)
(35000,)


100%|| 35000/35000 [05:55<00:00, 106.26it/s]
100%|| 15000/15000 [02:33<00:00, 84.10it/s]


'train_vect' and 'test_vect' are the pickle files.
```

```
In [50]: avgw2v_train_vect = frompicklefile('train_vect')
         avgw2v_test_vect = frompicklefile('test_vect')
         avgw2v_y_train = frompicklefile('avgw2v_y_train')
         y_test = frompicklefile('y_test')
```

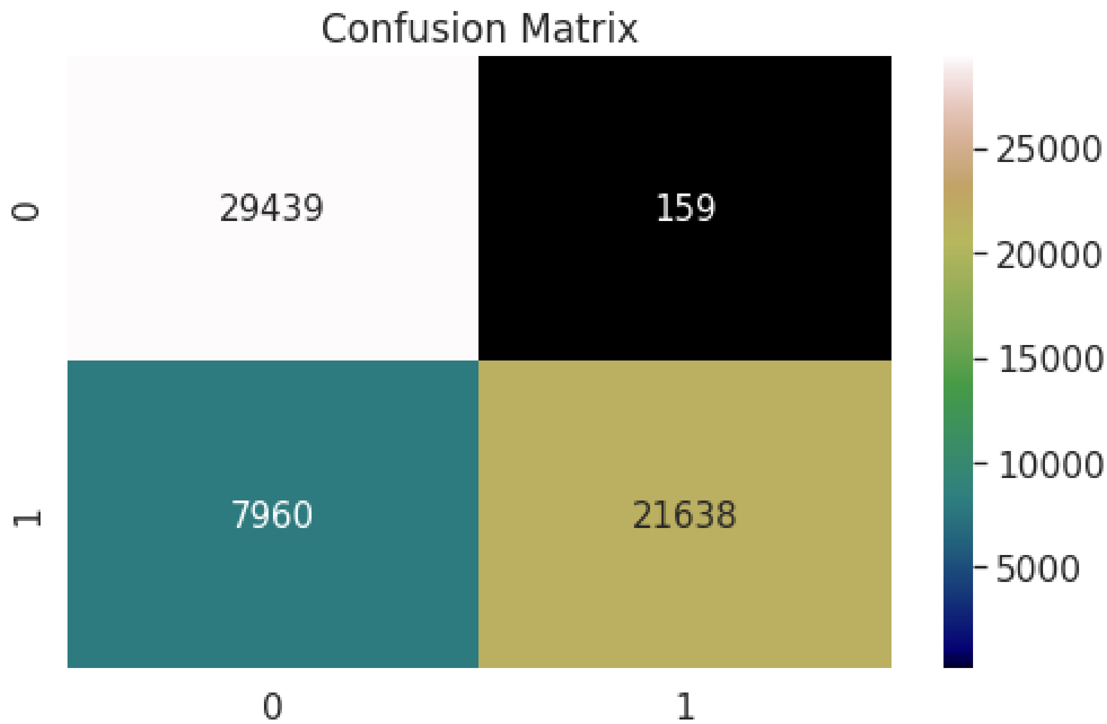
In [51]:
```
k_values = [i for i in range(3,30,3)]
clf, k_optimal, train_auc, cv_auc = applying_knn('brute', k_values, avgw2v_train_vect

k_optimal_avgw2v1 = k_optimal

print('The optimal K is {}' .format(k_optimal_avgw2v1))

train_cv_error_plot(k_values, train_auc, cv_auc)
```

```
The optimal K is 9
```

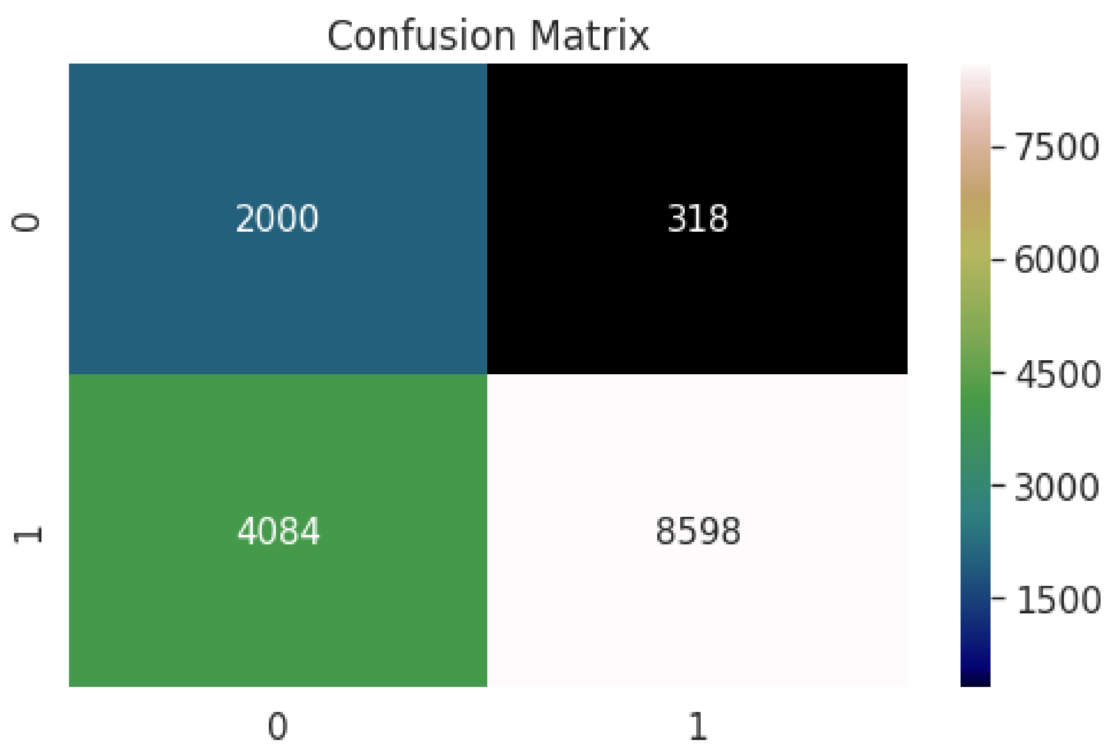


In [52]:
```
knn_optimal = optimal_knn('brute', k_optimal, avgw2v_train_vect, avgw2v_y_train)
```

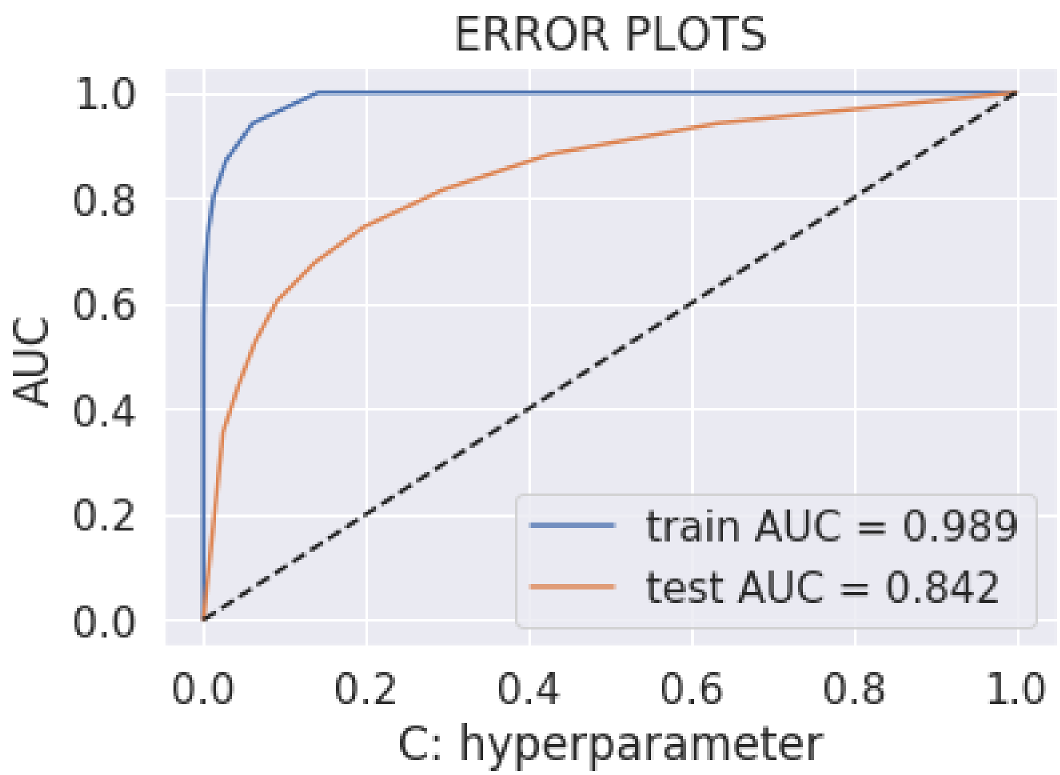
In [53]:
```
cm_fig(knn_optimal, avgw2v_y_train, avgw2v_train_vect)
```



In [54]: `cm_fig(knn_optimal, y_test, avgw2v_test_vect)`



```
In [55]: auc_avgw2v1 = error_plot(knn_optimal, avgw2v_train_vect, avgw2v_y_train, avgw2v_test_v
```



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

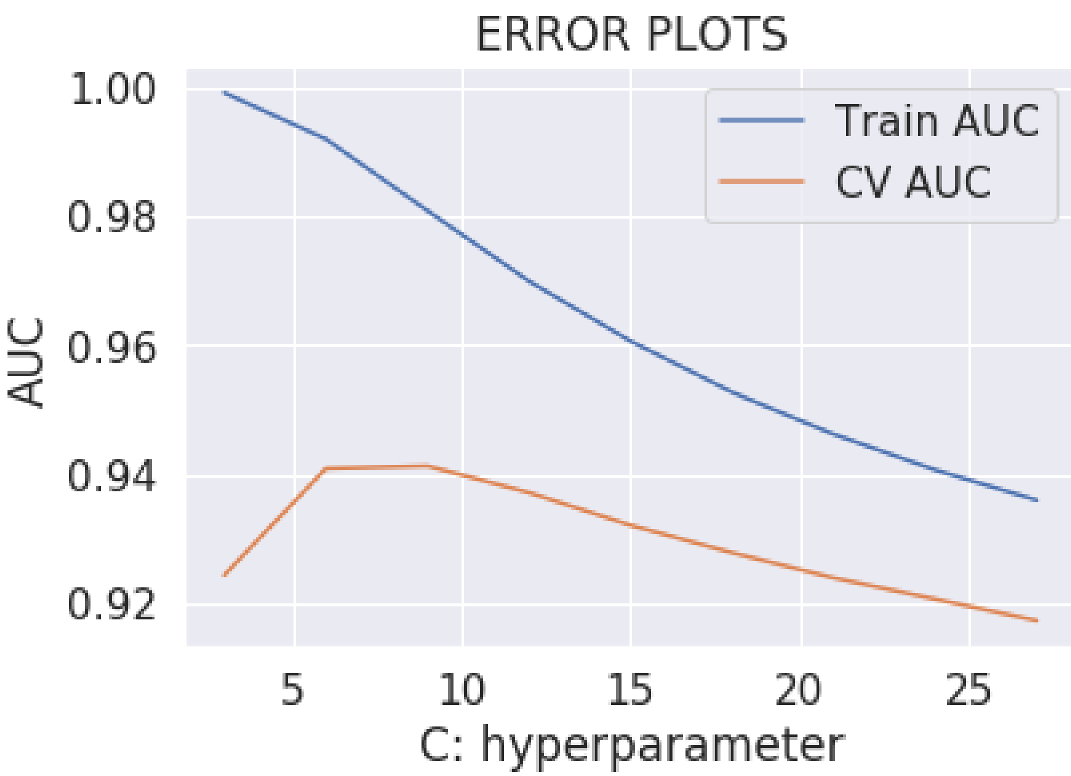
```
In [56]: count_vect = apply_vectorizers_train_test('brute', 'TF-IDF W2V')
```

```
else statement
(35000,)
(35000,)


100%|| 35000/35000 [25:37<00:00, 22.76it/s]
100%|| 15000/15000 [10:52<00:00, 22.99it/s]


'train_vect' and 'test_vect' are the pickle files.
```

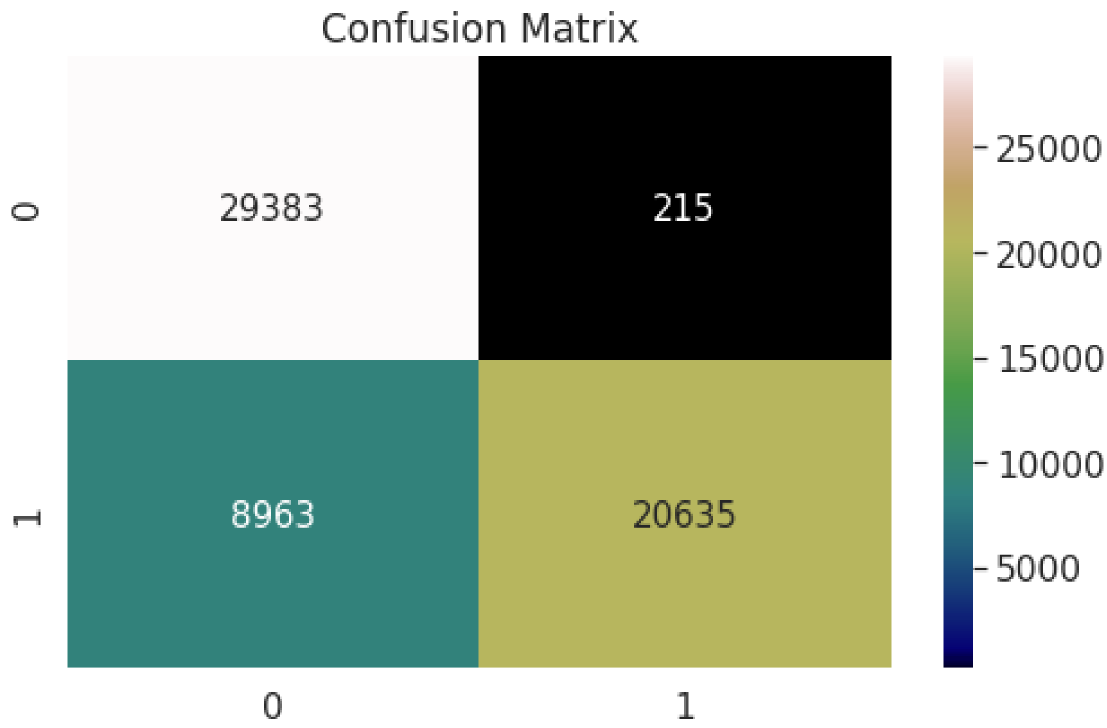
```
In [57]: tfidfw2v_train_vect = frompicklefile('train_vect')
         tfidfw2v_test_vect = frompicklefile('test_vect')
         tfidfw2v_y_train = frompicklefile('tfidfw2v_y_train')
         y_test = frompicklefile('y_test')
```

```
In [58]: k_values = [i for i in range(3,30,3)]
         clf, k_optimal, train_auc, cv_auc = applying_knn('brute', k_values, tfidfw2v_train_vec

         k_optimal_tfidfw2v1 = k_optimal

         print('The optimal K is {}' .format(k_optimal_tfidfw2v1))

         train_cv_error_plot(k_values, train_auc, cv_auc)
```

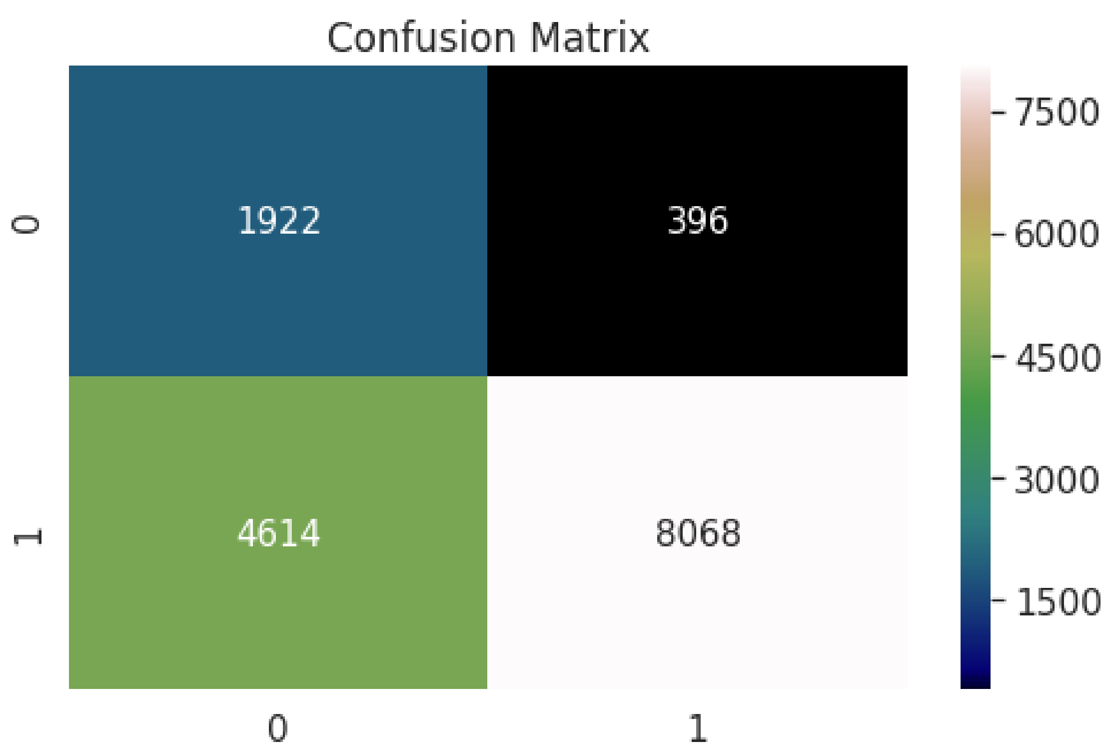
```
The optimal K is 9
```



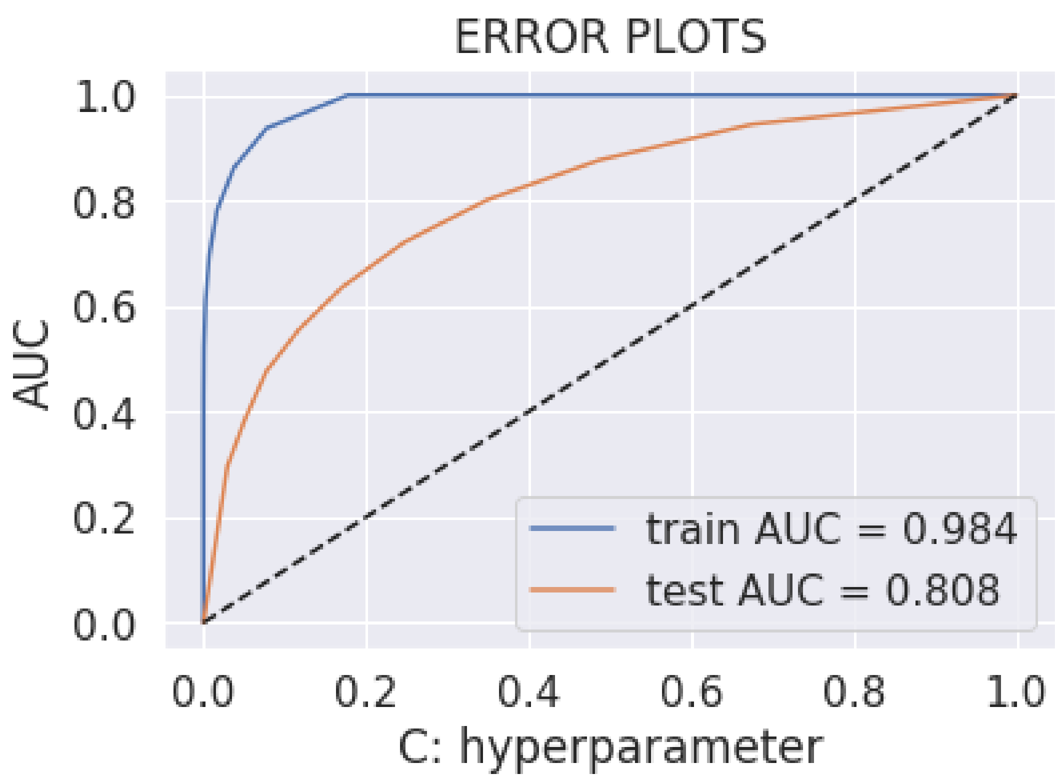

```
In [59]: knn_optimal = optimal_knn('brute', k_optimal, tfidfw2v_train_vect, tfidfw2v_y_train)

In [60]: cm_fig(knn_optimal, tfidfw2v_y_train, tfidfw2v_train_vect)
```




```
In [61]: cm_fig(knn_optimal, y_test, tfidfw2v_test_vect)
```



```
In [62]: auc_tfidfw2v1 = error_plot(knn_optimal, tfidfw2v_train_vect, tfidfw2v_y_train, tfidfw2
```



5.2 [5.2] Applying KNN kd-tree

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

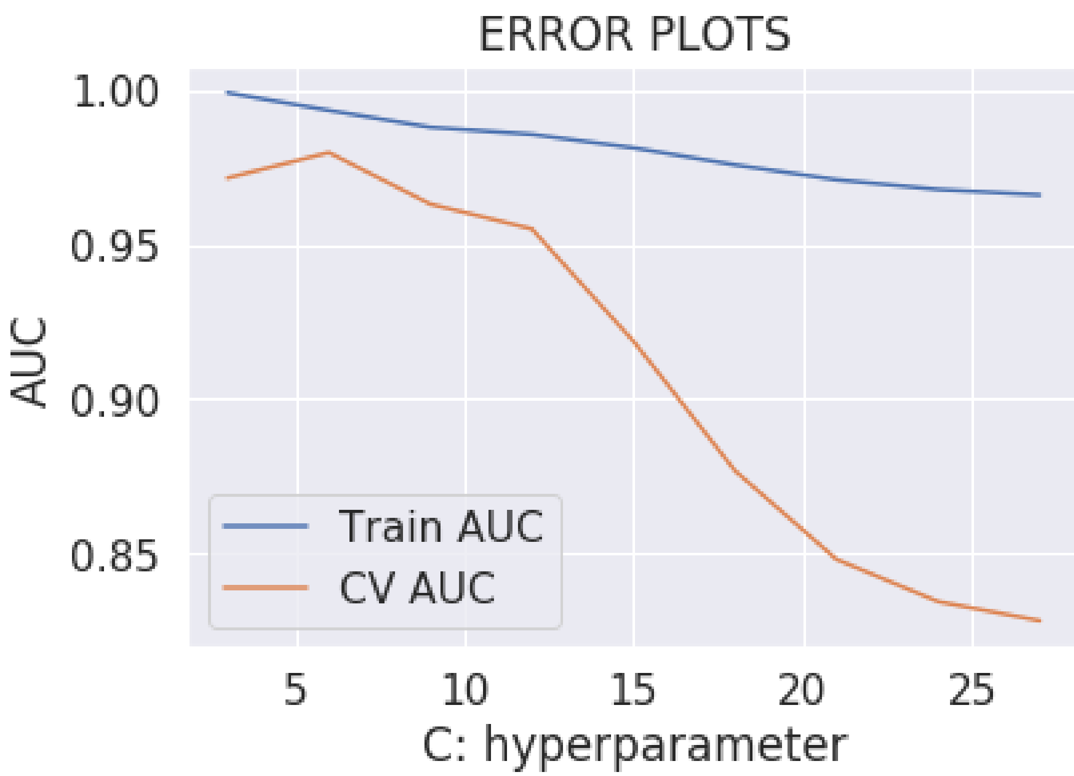
```
In [63]: count_vect = apply_vectorizers_train_test('kd_tree','BOW')
```

```
100%|| 20000/20000 [00:08<00:00, 2444.31it/s]
100%|| 20000/20000 [00:05<00:00, 3783.23it/s]


'train_vect' and 'test_vect' are the pickle files.
```

```
In [64]: bow_train_vect = frompicklefile('train_vect')
         bow_test_vect = frompicklefile('test_vect')
         bow_y_train = frompicklefile('bow_y_train')
         y_test = frompicklefile('y_test')
```

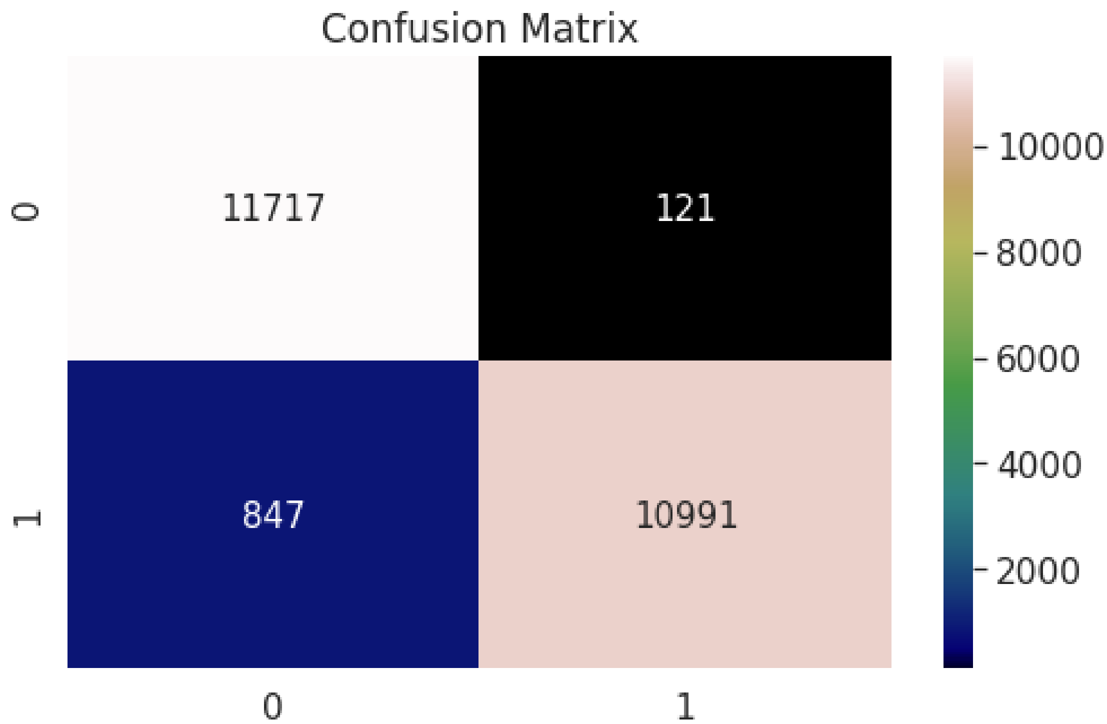
```
In [65]: k_values = [i for i in range(3,30,3)]
         clf, k_optimal, train_auc, cv_auc = applying_knn('kd_tree', k_values, bow_train_vect,

         k_optimal_bow2 = k_optimal

         print('The optimal K is {}' .format(k_optimal_bow2))

         train_cv_error_plot(k_values, train_auc, cv_auc)
```

```
The optimal K is 6
```

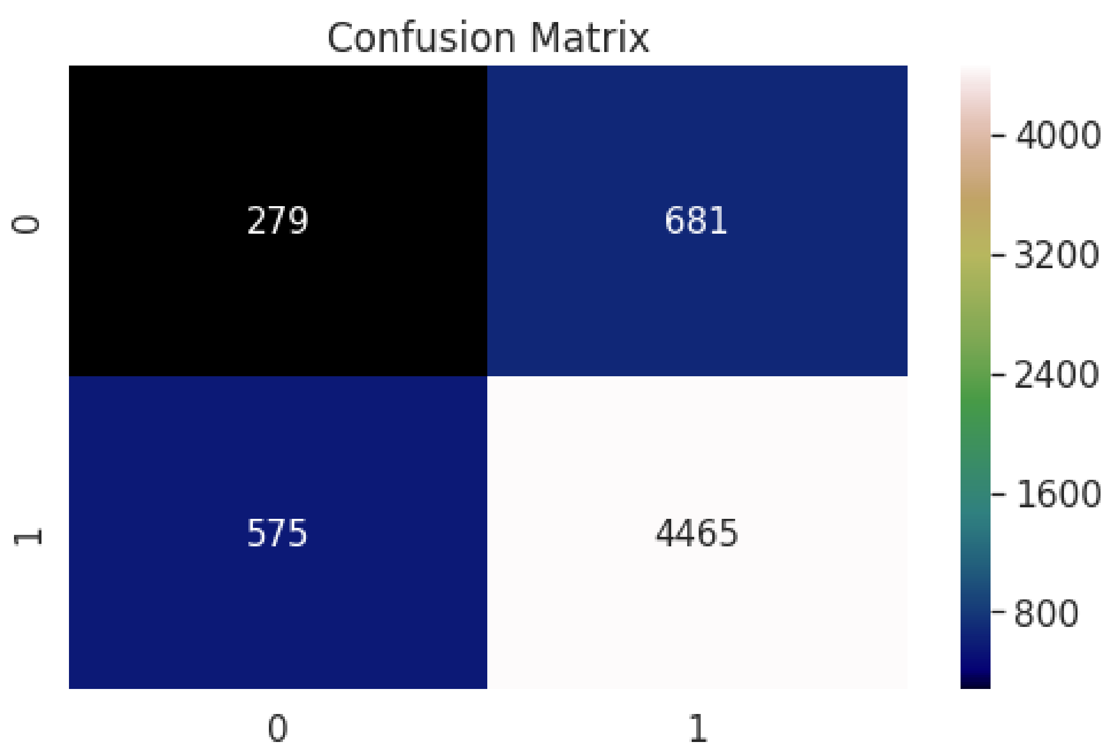



```
In [66]: knn_optimal = optimal_knn('kd_tree', k_optimal, bow_train_vect, bow_y_train)
```

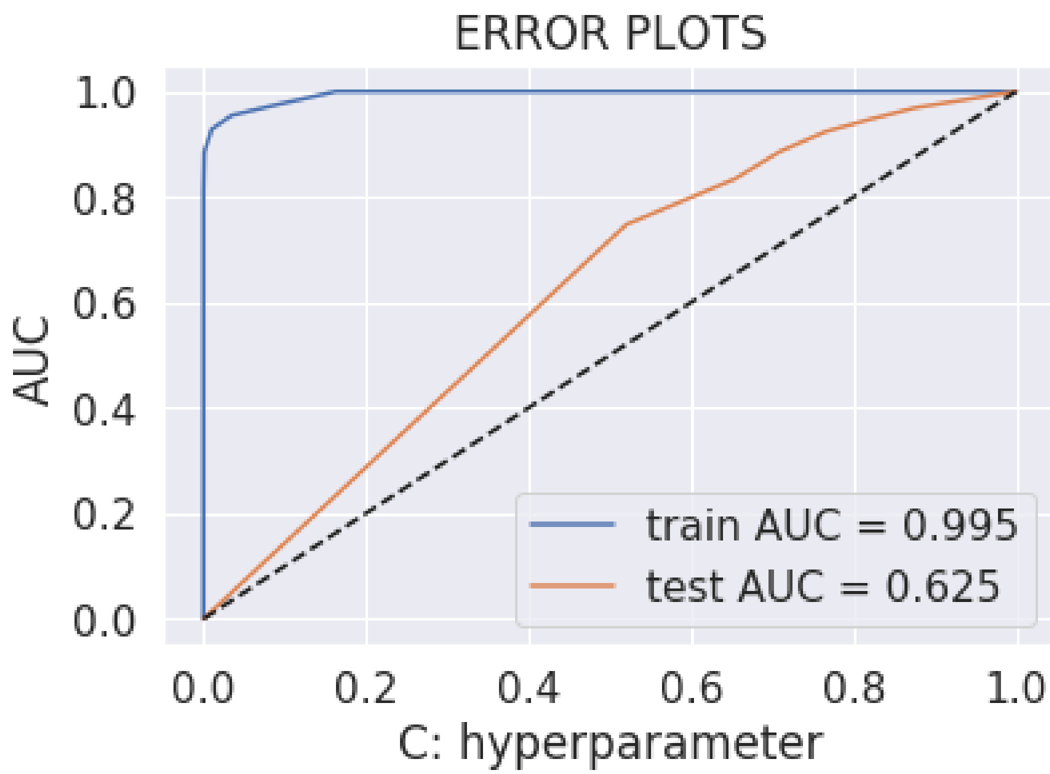
```
In [67]: cm_fig(knn_optimal, bow_y_train, bow_train_vect)
```



In [68]: cm_fig(knn_optimal, y_test, bow_test_vect)



```
In [69]: auc_bow2 = error_plot(knn_optimal, bow_train_vect, bow_y_train, bow_test_vect, y_test
```



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

```
In [70]: count_vect = apply_vectorizers_train_test('kd_tree','TF-IDF')
```

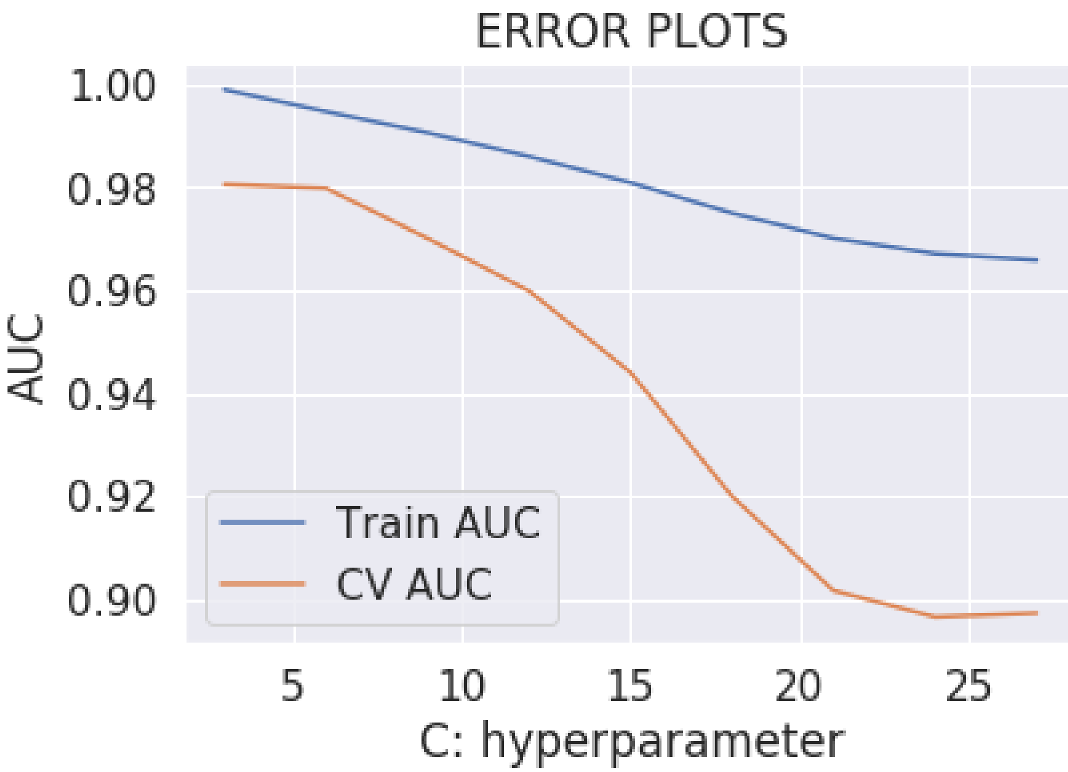
```
else statement
(14000,)
(14000,)
'train_vect' and 'test_vect' are the pickle files.
```

```
In [71]: tfidf_train_vect = frompicklefile('train_vect')
         tfidf_test_vect = frompicklefile('test_vect')
         tfidf_y_train = frompicklefile('tfidf_y_train')
         y_test = frompicklefile('y_test')
```

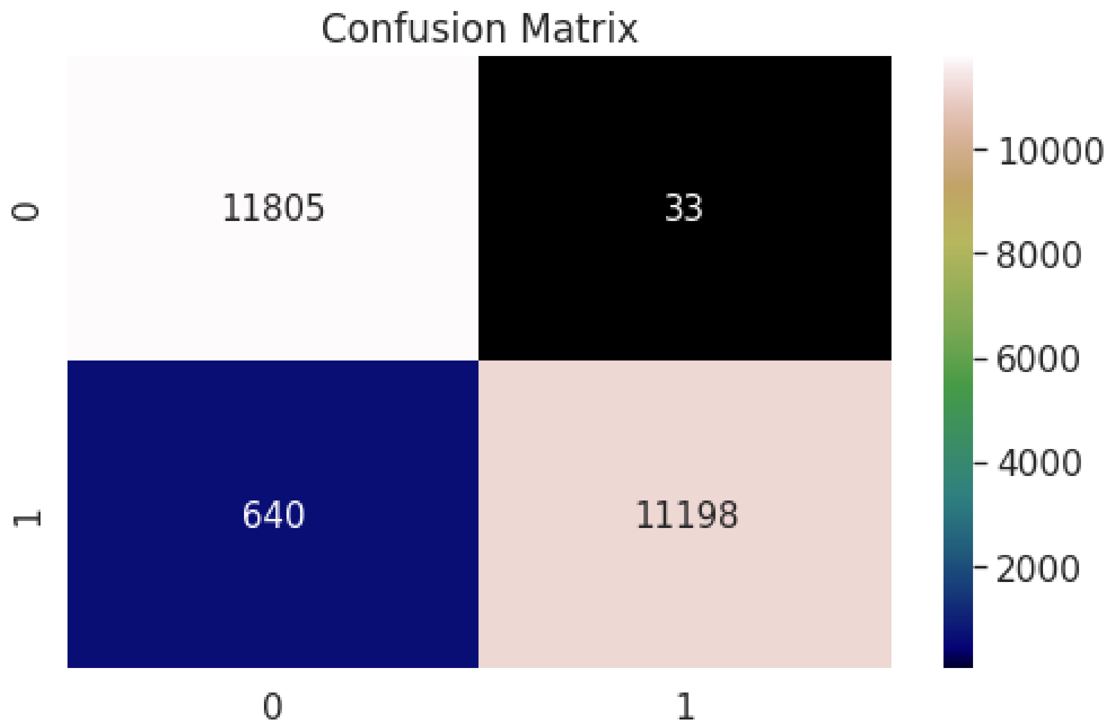
```
In [72]: k_values = [i for i in range(3,30,3)]
         clf, k_optimal, train_auc, cv_auc = applying_knn('kd_tree', k_values, tfidf_train_vec
```

```
k_optimal_tfidf2 = k_optimal

print('The optimal K is {}' .format(k_optimal_tfidf2))

train_cv_error_plot(k_values, train_auc, cv_auc)
```

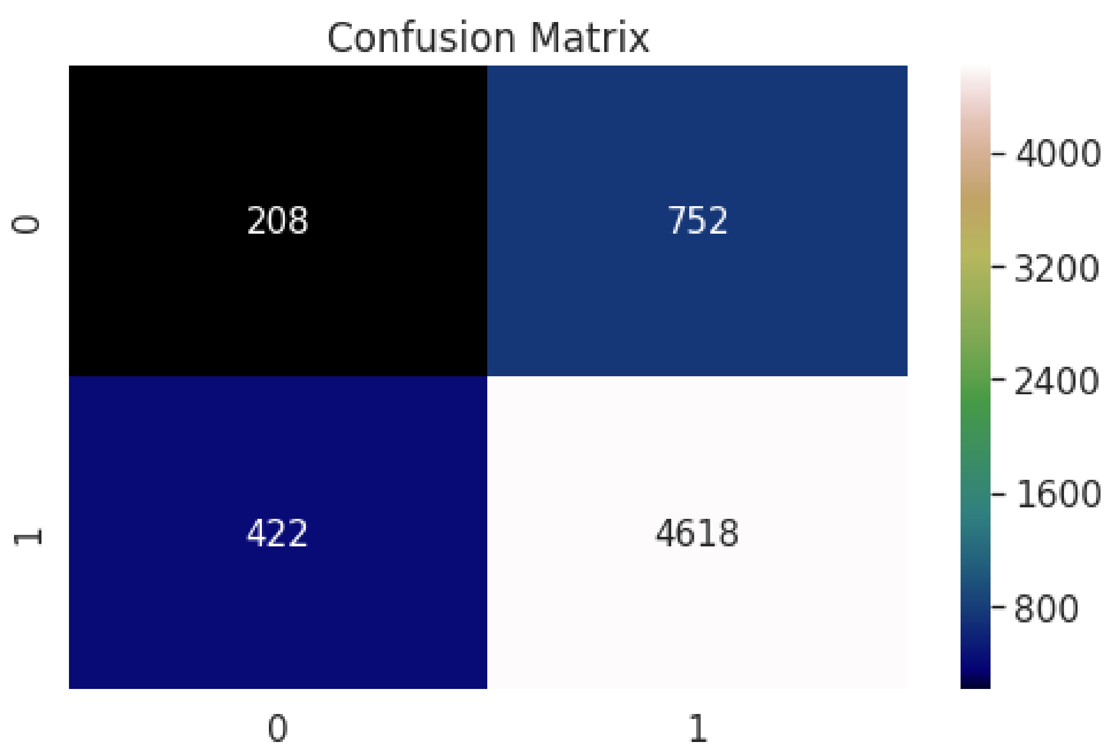
```
The optimal K is 3
```



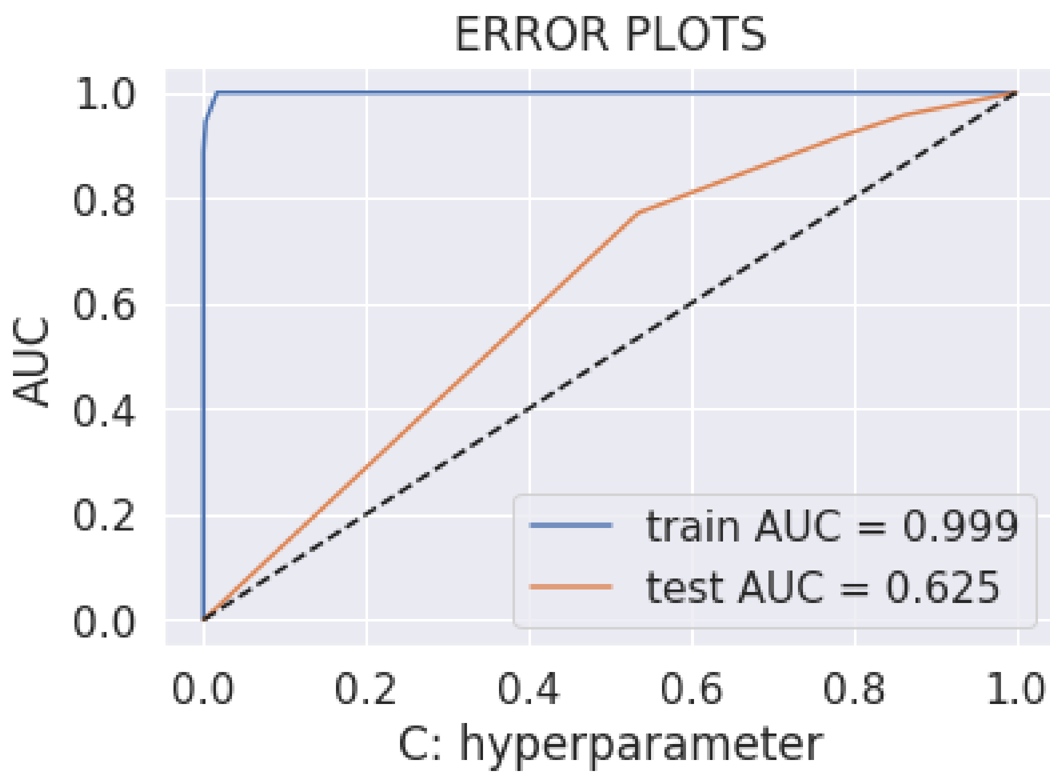

```
In [73]: knn_optimal = optimal_knn('kd_tree', k_optimal, tfidf_train_vect, tfidf_y_train)

In [74]: cm_fig(knn_optimal, tfidf_y_train, tfidf_train_vect)
```




```
In [75]: cm_fig(knn_optimal, y_test, tfidf_test_vect)
```



```
In [76]: auc_tfidf2 = error_plot(knn_optimal, tfidf_train_vect, tfidf_y_train, tfidf_test_vect
```



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

```
In [77]: count_vect = apply_vectorizers_train_test('kd_tree','AvgW2V')
```

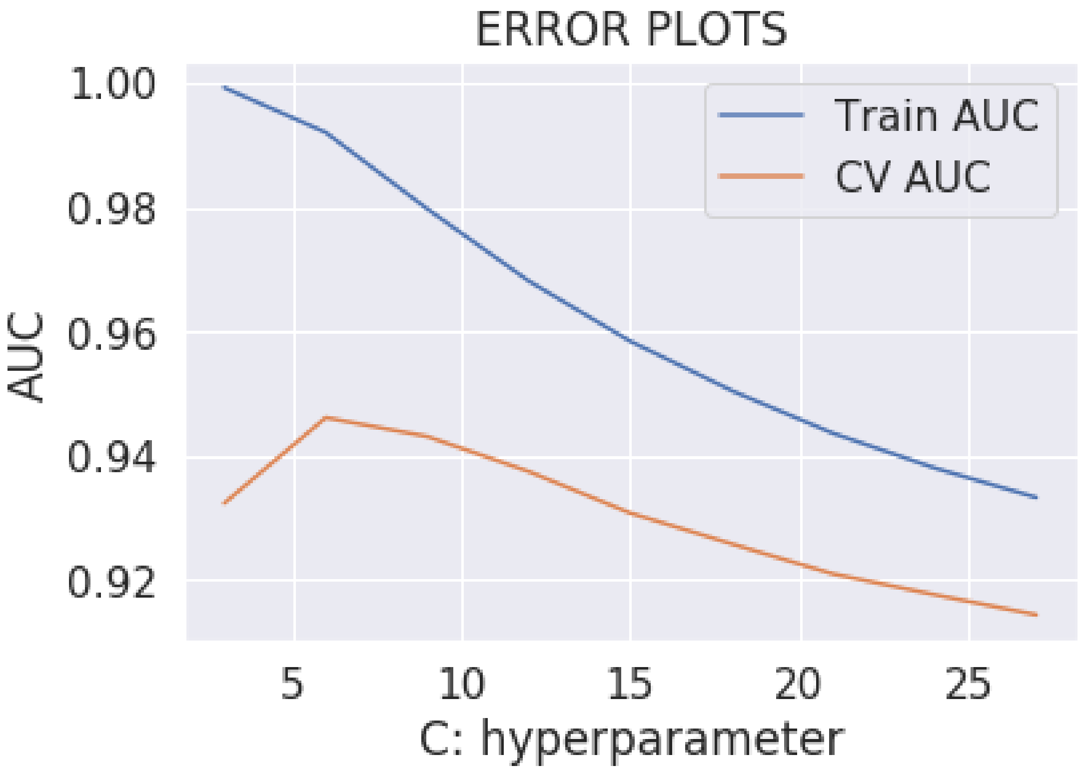
```
else statement
(14000,)
(14000,)


100%|| 14000/14000 [01:15<00:00, 184.94it/s]
100%|| 6000/6000 [00:32<00:00, 184.66it/s]
```

'train_vect' and 'test_vect' are the pickle files.

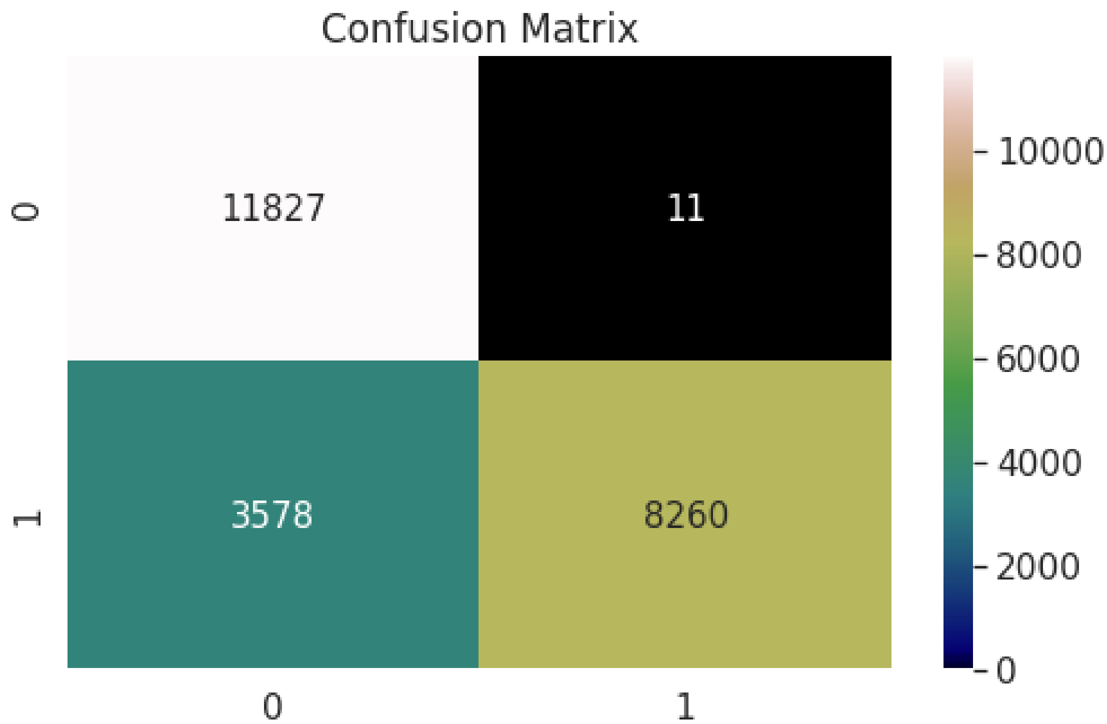
```
In [78]: avgw2v_train_vect = frompicklefile('train_vect')
         avgw2v_test_vect = frompicklefile('test_vect')
         avgw2v_y_train = frompicklefile('avgw2v_y_train')
         y_test = frompicklefile('y_test')
```

```
In [79]: k_values = [i for i in range(3,30,3)]
         clf, k_optimal, train_auc, cv_auc = applying_knn('kd_tree', k_values, avgw2v_train_vec

         k_optimal_avgw2v2 = k_optimal

         print('The optimal K is {}' .format(k_optimal_avgw2v2))

         train_cv_error_plot(k_values, train_auc, cv_auc)
```

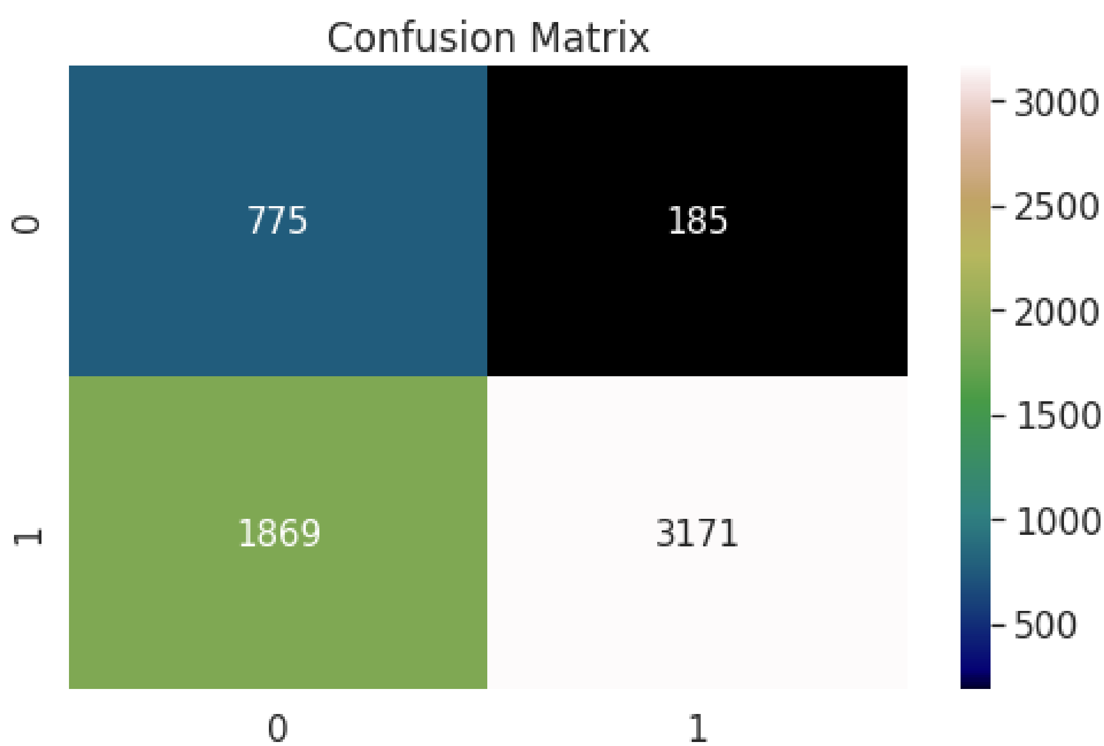
```
The optimal K is 6
```



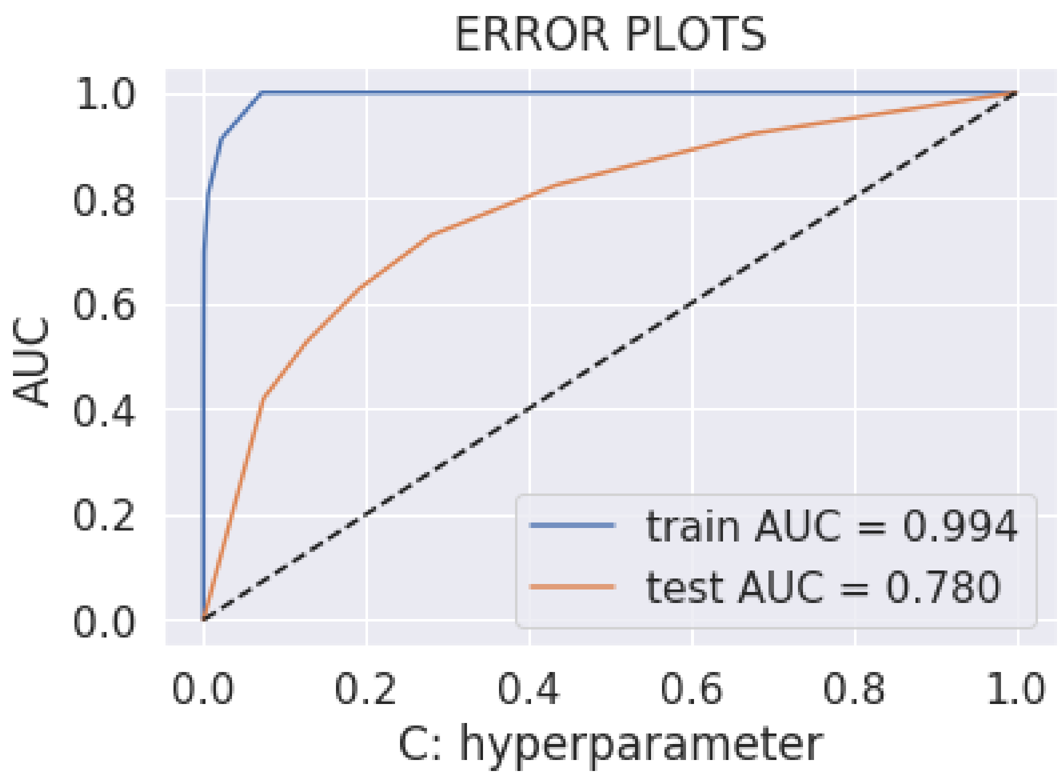

```
In [80]: knn_optimal = optimal_knn('kd_tree', k_optimal, avgw2v_train_vect, avgw2v_y_train)

In [81]: cm_fig(knn_optimal, avgw2v_y_train, avgw2v_train_vect)
```




```
In [82]: cm_fig(knn_optimal, y_test, avgw2v_test_vect)
```



```
In [83]: auc_avgw2v2 = error_plot(knn_optimal, avgw2v_train_vect, avgw2v_y_train, avgw2v_test_y
```



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

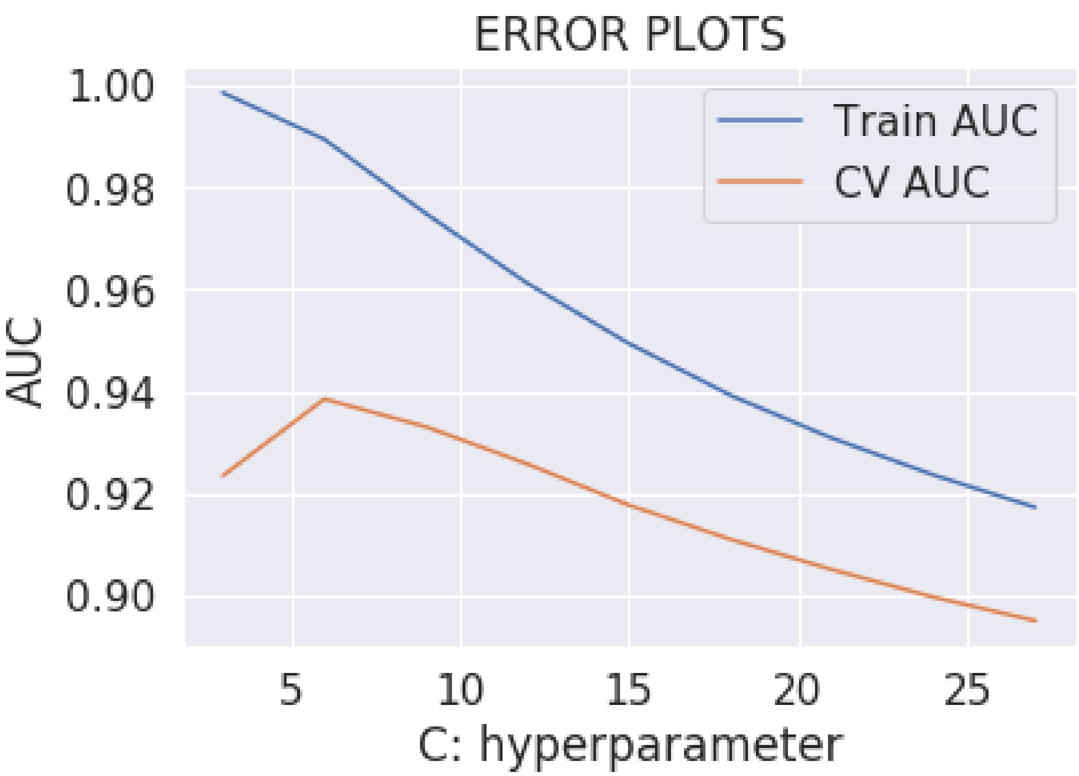
```
In [84]: count_vect = apply_vectorizers_train_test('kd_tree','TF-IDF W2V')
```

```
else statement
(14000,)
(14000,)


100%|| 14000/14000 [05:42<00:00, 40.90it/s]
100%|| 6000/6000 [02:27<00:00, 40.74it/s]


'train_vect' and 'test_vect' are the pickle files.
```

```
In [85]: tfidfw2v_train_vect = frompicklefile('train_vect')
         tfidfw2v_test_vect = frompicklefile('test_vect')
         tfidfw2v_y_train = frompicklefile('tfidfw2v_y_train')
         y_test = frompicklefile('y_test')
```

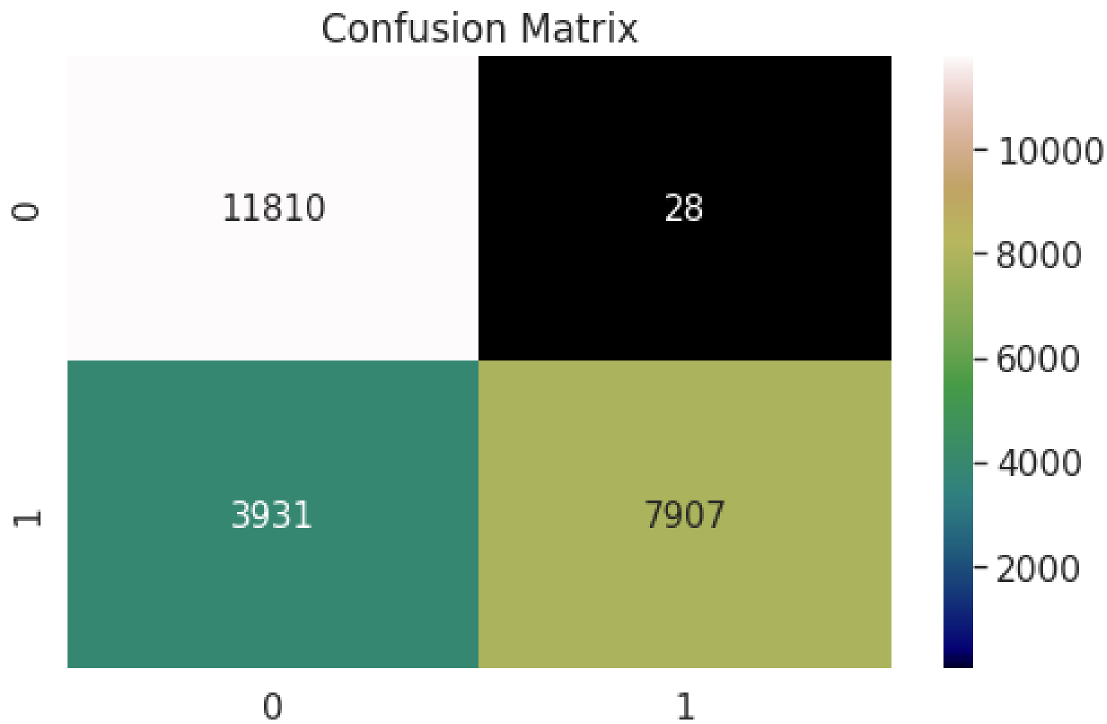
```
In [86]: k_values = [i for i in range(3,30,3)]
         clf, k_optimal, train_auc, cv_auc = applying_knn('kd_tree', k_values, tfidfw2v_train_v

         k_optimal_tfidfw2v2 = k_optimal

         print('The optimal K is {}' .format(k_optimal_tfidfw2v2))

         train_cv_error_plot(k_values, train_auc, cv_auc)
```

```
The optimal K is 6
```

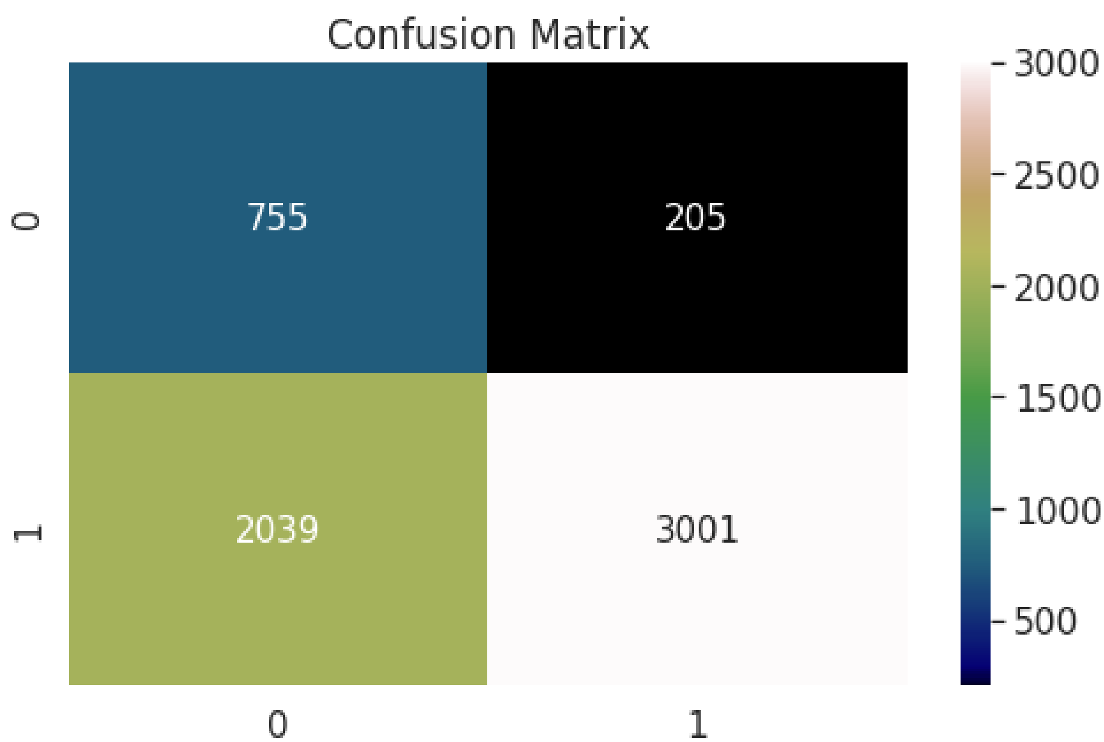



```
In [87]: knn_optimal = optimal_knn('kd_tree', k_optimal, tfidfw2v_train_vect, tfidfw2v_y_train
```

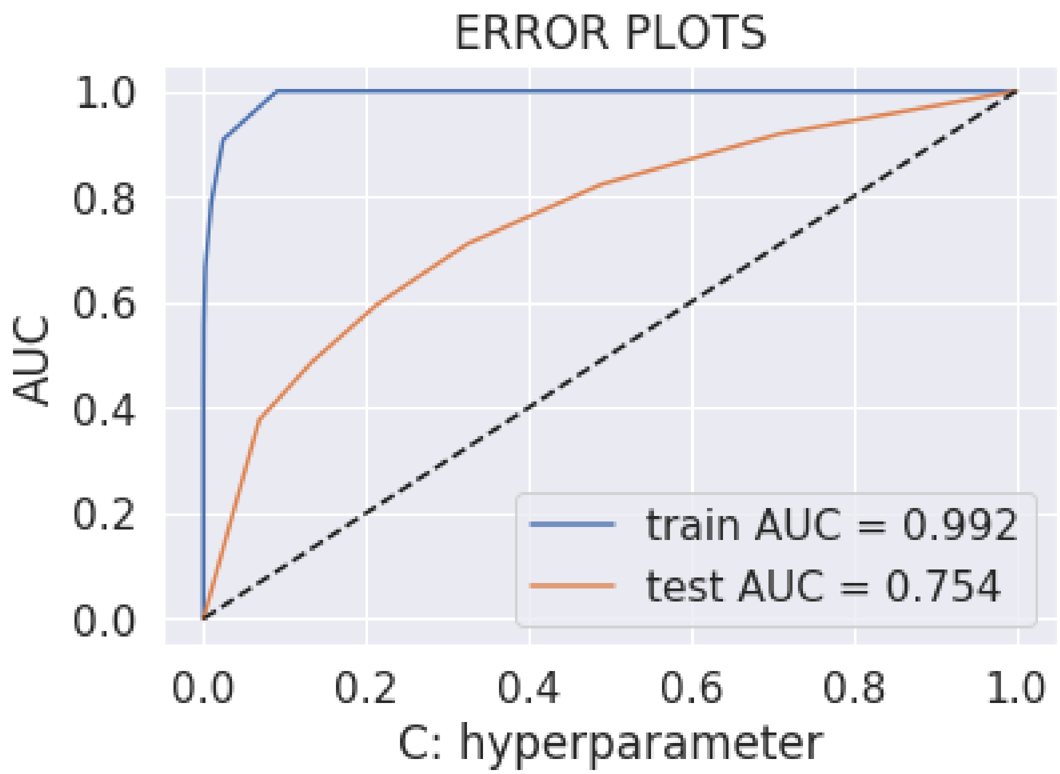
```
In [88]: cm_fig(knn_optimal, tfidfw2v_y_train, tfidfw2v_train_vect)
```




```
In [89]: cm_fig(knn_optimal, y_test, tfidfw2v_test_vect)
```



```
In [90]: auc_tfidfw2v2 = error_plot(knn_optimal, tfidfw2v_train_vect, tfidfw2v_y_train, tfidfw2
```



6 [6] Conclusions

```
In [91]: #Source: http://zetcode.com/python/prettytable/
         from prettytable import PrettyTable

         model_metric = PrettyTable()

         model_metric = PrettyTable(["Model Name", "Algorithm", 'Hyperparameter', 'Test AUC'])

         model_metric.add_row(["Bag of Words","Brute Force", k_optimal_bow1, auc_bow1])
         model_metric.add_row(["TF-IDF","Brute Force", k_optimal_tfidf1, auc_tfidf1])
         model_metric.add_row(["Avg W2V","Brute Force", k_optimal_avgw2v1, auc_avgw2v1])
         model_metric.add_row(["TF-IDF W2V","Brute Force", k_optimal_tfidfw2v1, auc_tfidfw2v1]
         model_metric.add_row(["Bag of Words","KD-tree", k_optimal_bow2, auc_bow2])
         model_metric.add_row(["TF-IDF","KD-tree", k_optimal_tfidf2, auc_tfidf2])
         model_metric.add_row(["Avg W2V","KD-tree", k_optimal_avgw2v2, auc_avgw2v2])
         model_metric.add_row(["TF-IDF W2V","KD-tree", k_optimal_tfidfw2v2, auc_tfidfw2v2])
```

```python
print(model_metric.get_string(start=0, end=8))
```

```
+--------------+-------------+----------------+--------------------+
|  Model Name  |  Algorithm  | Hyperparameter |      Test AUC      |
+--------------+-------------+----------------+--------------------+
| Bag of Words | Brute Force |       6        | 0.725350373964907  |
|    TF-IDF    | Brute Force |       6        | 0.6565892409792116 |
|   Avg W2V    | Brute Force |       9        | 0.8423776560475338 |
|  TF-IDF W2V  | Brute Force |       9        | 0.8078694314321018 |
| Bag of Words |   KD-tree   |       6        | 0.625309089781746  |
|    TF-IDF    |   KD-tree   |       3        | 0.624642650462963  |
|   Avg W2V    |   KD-tree   |       6        | 0.7801007564484127 |
|  TF-IDF W2V  |   KD-tree   |       6        | 0.7535802124669312 |
+--------------+-------------+----------------+--------------------+
```

7 [7] Observations

1. Time Elapsed: Brute Force takes significantly less time compared to KD-Tree to train for all the algorithms. Hence Brute Force based models were trained & tested on 50k data and kd_tree were trained & tested on 20k.

2. All the models are trying to slightly overfit on the training data to some extent.

3. SOMTE(Synthetic Minority Oversampling Technique) is used to generate new datapoints for the minority class.

4. Test AUC scores ranges from 0.62 to 0.84 for all the models and the Hyperparameter K ranges from 3 to 9.