Amazon Fine Food Reviews Analysis

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

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

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

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

Attribute Information:

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

Objective:

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

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

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

[1]. Reading Data

[1.1] Loading the data

The dataset is available in two forms

1. .csv file
2. SQLite Database

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

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

In [2]:

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


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

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

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

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

from tqdm import tqdm
import os
```

In [3]:

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

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

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


#filtered_data.merge(filtered_data_positive,left_on='Score',right_on='Score')
print("Number of data points in our data", filtered_data.shape)
# Give reviews with Score>3 a positive rating(1), and reviews with a score<3 a negative rating(0).
def partition(x):
    if x < 3:
        return 0
    return 1

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

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

Out[3]:

	Id	ProductId	UserId	ProfileName	HelpfulnessNumerator	HelpfulnessDenominator	Score	Tim
0	1	B001E4KFG0	A3SGXH7AUHU8GW	delmartian	1	1	1	130386240
1	2	B00813GRG4	A1D87F6ZCVE5NK	dll pa	0	0	0	134697600

	Id	ProductId	UserId	ProfileName	HelpfulnessNumerator	HelpfulnessDenominator	Score	Time
2	3	B000LQOCH0	ABXLMWJIXXAIN	Natalia Corres "Natalia Corres"	1	1	1	121901760

In [4]:

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

In [5]:

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

```
(80668, 7)
```

Out[5]:

	UserId	ProductId	ProfileName	Time	Score	Text	COUNT(*)
0	#oc-R115TNMSPFT9I7	B007Y59HVM	Breyton	1331510400	2	Overall its just OK when considering the price...	2
1	#oc-R11D9D7SHXIJB9	B005HG9ET0	Louis E. Emory "hoppy"	1342396800	5	My wife has recurring extreme muscle spasms, u...	3
2	#oc-R11DNU2NBKQ23Z	B007Y59HVM	Kim Cieszykowski	1348531200	1	This coffee is horrible and unfortunately not ...	2
3	#oc-R11O5J5ZVQE25C	B005HG9ET0	Penguin Chick	1346889600	5	This will be the bottle that you grab from the...	3
4	#oc-R12KPBODL2B5ZD	B007OSBE1U	Christopher P. Presta	1348617600	1	I didnt like this coffee. Instead of telling y...	2

In [6]:

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

Out[6]:

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

In [7]:

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

Out[7]:

```
393063
```

[2] Exploratory Data Analysis

[2.1] Data Cleaning: Deduplication

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

In [8]:

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

Out[8]:

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

As it can be seen above that same user has multiple reviews with same values for HelpfulnessNumerator, HelpfulnessDenominator, Score, Time, Summary and Text and on doing analysis it was found that

ProductId=B000HDOPZG was Loacker Quadratini Vanilla Wafer Cookies, 8.82-Ounce Packages (Pack of 8)

ProductId=B000HDL1RQ was Loacker Quadratini Lemon Wafer Cookies, 8.82-Ounce Packages (Pack of 8) and so on

It was inferred after analysis that reviews with same parameters other than ProductId belonged to the same product just having different flavour or quantity. Hence in order to reduce redundancy it was decided to eliminate the rows having same parameters.

The method used for the same was that we first sort the data according to ProductId and then just keep the first similar product review and delelte the others. for eg. in the above just the review for ProductId=B000HDL1RQ remains. This method ensures that there is only one representative for each product and deduplication without sorting would lead to possibility of different representatives still existing for the same product.

In [9]:

```
#Sorting data according to ProductId in ascending order
sorted_data=filtered_data.sort_values('ProductId', axis=0, ascending=True, inplace=False, kind='qui
cksort', na_position='last')
```

In [10]:

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

Out[10]:

```
(364173, 10)
```

In [11]:

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

Out[11]:

```
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 [12]:

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

display.head()
```

Out[12]:

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

In [13]:

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

In [14]:

```
#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[14]:

```
1    307061
0     57110
Name: Score, dtype: int64
```

[3] Preprocessing

[3.1]. Preprocessing Review Text

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

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

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

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

In [15]:

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

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

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

sent_4900 = final['Text'].values[3000]
print(sent_4900)
print("="*50)
```

```
this witty little book makes my son laugh at loud. i recite it in the car as we're driving along a
nd he always can sing the refrain. he's learned about whales, India, drooping roses:  i love all t
he new words this book  introduces and the silliness of it all.  this is a classic book i am
willing to bet my son will STILL be able to recite from memory when he is  in college
==================================================
I was really looking forward to these pods based on the reviews.  Starbucks is good, but I prefer
bolder taste.... imagine my surprise when I ordered 2 boxes - both were expired! One expired back
in 2005 for gosh sakes.  I admit that Amazon agreed to credit me for cost plus part of shipping, b
ut geez, 2 years expired!!!  I'm hoping to find local San Diego area shoppe that carries pods so t
hat I can try something different than starbucks.
==================================================
Great ingredients although, chicken should have been 1st rather than chicken broth, the only thing
I do not think belongs in it is Canola oil. Canola or rapeseed is not someting a dog would ever fi
nd in nature and if it did find rapeseed in nature and eat it, it would poison them. Today's Food
industries have convinced the masses that Canola oil is a safe and even better oil than olive or v
irgin coconut, facts though say otherwise. Until the late 70's it was poisonous until they figured
out a way to fix that. I still like it but it could be better.
==================================================
I first had this tea at a vegetarian Vietnamese restaurant. I ordered green tea, and as the
waitress came out of the kitchen with the pot I could smell it already. It has the most heavenly a
roma; in fact, I find just the smell relaxing. The taste is also very nice and unexpected. I find,
now that I know I like jasmine green tea, that there are very few "jasmine" teas where you can act
ually taste the jasmine. Usually the only ones that can do that are the very expensive loose teas
from tea shoppes, and Stassen's tea. It's altogether a lovely experience, and I wish that I could
get it in decaf too.
==================================================
```

In [16]:

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

print(sent_0)
```

```
this witty little book makes my son laugh at loud. i recite it in the car as we're driving along a
nd he always can sing the refrain. he's learned about whales, India, drooping roses:  i love all t
he new words this book  introduces and the silliness of it all.  this is a classic book i am
willing to bet my son will STILL be able to recite from memory when he is  in college
```

In [17]:

```
# https://stackoverflow.com/questions/16206380/python-beautifulsoup-how-to-remove-all-tags-from-an
-element
from bs4 import BeautifulSoup

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

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

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

```
this witty little book makes my son laugh at loud. i recite it in the car as we're driving along a
nd he always can sing the refrain. he's learned about whales, India, drooping roses:  i love all t
he new words this book  introduces and the silliness of it all.  this is a classic book i am
willing to bet my son will STILL be able to recite from memory when he is  in college
==================================================
I was really looking forward to these pods based on the reviews.  Starbucks is good, but I prefer
bolder taste.... imagine my surprise when I ordered 2 boxes - both were expired! One expired back
in 2005 for gosh sakes.  I admit that Amazon agreed to credit me for cost plus part of shipping, b
ut geez, 2 years expired!!!  I'm hoping to find local San Diego area shoppe that carries pods so t
hat I can try something different than starbucks.
==================================================
Great ingredients although, chicken should have been 1st rather than chicken broth, the only thing
I do not think belongs in it is Canola oil. Canola or rapeseed is not someting a dog would ever fi
nd in nature and if it did find rapeseed in nature and eat it, it would poison them. Today's Food
industries have convinced the masses that Canola oil is a safe and even better oil than olive or v
irgin coconut, facts though say otherwise. Until the late 70's it was poisonous until they figured
out a way to fix that. I still like it but it could be better.
==================================================
```

In [18]:

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

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

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

In [19]:

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

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

In [20]:

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

```
this witty little book makes my son laugh at loud. i recite it in the car as we're driving along a
nd he always can sing the refrain. he's learned about whales, India, drooping roses:  i love all t
he new words this book  introduces and the silliness of it all.  this is a classic book i am
willing to bet my son will STILL be able to recite from memory when he is  in college
```

In [21]:

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

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

In [22]:

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

stopwords= set(['br', 'the', 'i', 'me', 'my', 'myself', 'we', 'our', 'ours', 'ourselves', 'you', "y
ou're", "you've",\
            "you'll", "you'd", 'your', 'yours', 'yourself', 'yourselves', 'he', 'him', 'his',
'himself', \
            'she', "she's", 'her', 'hers', 'herself', 'it', "it's", 'its', 'itself', 'they', 'them',
'their',\
            'theirs', 'themselves', 'what', 'which', 'who', 'whom', 'this', 'that', "that'll",
'these', 'those', \
            'am', 'is', 'are', 'was', 'were', 'be', 'been', 'being', 'have', 'has', 'had', 'having',
'do', 'does', \
            'did', 'doing', 'a', 'an', 'the', 'and', 'but', 'if', 'or', 'because', 'as', 'until', '
while', 'of', \
            'at', 'by', 'for', 'with', 'about', 'against', 'between', 'into', 'through', 'during',
'before', 'after',\
            'above', 'below', 'to', 'from', 'up', 'down', 'in', 'out', 'on', 'off', 'over', 'under'
, 'again', 'further',\
            'then', 'once', 'here', 'there', 'when', 'where', 'why', 'how', 'all', 'any', 'both', 'e
ach', 'few', 'more',\
            'most', 'other', 'some', 'such', 'only', 'own', 'same', 'so', 'than', 'too', 'very', \
            's', 't', 'can', 'will', 'just', 'don', "don't", 'should', "should've", 'now', 'd', 'll'
, 'm', 'o', 're', \
            've', 'y', 'ain', 'aren', "aren't", 'couldn', "couldn't", 'didn', "didn't", 'doesn', "d
esn't", 'hadn',\
            "hadn't", 'hasn', "hasn't", 'haven', "haven't", 'isn', "isn't", 'ma', 'mightn',
"mightn't", 'mustn',\
            "mustn't", 'needn', "needn't", 'shan', "shan't", 'shouldn', "shouldn't", 'wasn',
"wasn't", 'weren', "weren't", \
            'won', "won't", 'wouldn', "wouldn't"])
```

In [23]:

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

```
100%|██████████████████████████████████████████████████████████████████████████| 364171/364171
[03:34<00:00, 1698.34it/s]
```

In [24]:

```
print(len(preprocessed_reviews))
```

```
364171
```

[3.2] Preprocessing Review Summary

In [25]:

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

In [26]:

```
from sklearn.model_selection import train_test_split

data_BF = preprocessed_reviews[:20000]
scores_BF = final['Score'][:20000]

data_KDT = preprocessed_reviews[:10000]
scores_KDT = final['Score'][:10000]

data_train_BF,data_test_BF,scores_train_BF,scores_test_BF = train_test_split(data_BF, scores_BF,
test_size=0.2, random_state=1)
data_train_BF,data_cv_BF,scores_train_BF,scores_cv_BF = train_test_split(data_train_BF,
scores_train_BF, test_size=0.25, random_state=1)

data_train_KDT,data_test_KDT,scores_train_KDT,scores_test_KDT = train_test_split(data_KDT, scores_K
DT, test_size=0.2, random_state=1)
data_train_KDT,data_cv_KDT,scores_train_KDT,scores_cv_KDT = train_test_split(data_train_KDT, scores
_train_KDT, test_size=0.25, random_state=1)

print("Length of data_train_BF : ",len(data_train_BF))
print("Length of data_cv_BF : ",len(data_cv_BF))
print("Length of data_test_BF : ",len(data_test_BF))
print("Length of scores_train_BF : ",len(scores_train_BF))
print("Length of scores_cv_BF : ",len(scores_cv_BF))
print("Length of scores_test_BF : ",len(scores_test_BF))
print("Length of data_train_KDT : ",len(data_train_KDT))
print("Length of data_cv_KDT : ",len(data_cv_KDT))
print("Length of data_test_KDT : ",len(data_test_KDT))
print("Length of scores_train_KDT : ",len(scores_train_KDT))
print("Length of scores_cv_KDT : ",len(scores_cv_KDT))
print("Length of scores_test_KDT : ",len(scores_test_KDT))
```

```
Length of data_train_BF :  12000
Length of data_cv_BF :  4000
Length of data_test_BF :  4000
Length of scores_train_BF :  12000
Length of scores_cv_BF :  4000
Length of scores_test_BF :  4000
Length of data_train_KDT :  6000
Length of data_cv_KDT :  2000
Length of data_test_KDT :  2000
Length of scores_train_KDT :  6000
Length of scores_cv_KDT :  2000
Length of scores_test_KDT :  2000
```

[4] Featurization

[4.1] BAG OF WORDS

In [27]:

```
#BoW
#Brute
count_vect = CountVectorizer() #in scikit-learn
count_vect.fit(data_train_BF)
final_counts_BF = count_vect.fit_transform(data_train_BF)
bow_train_BF = final_counts_BF
bow_cv_BF = count_vect.transform(data_cv_BF)
bow_test_BF = count_vect.transform(data_test_BF)

#KDTree
count_vect = CountVectorizer(min_df=10,max_features=500)
count_vect.fit(data_train_KDT)
final_counts_KDT = count_vect.fit_transform(data_train_KDT)
bow_train_KDT = final_counts_KDT.toarray()
bow_cv_KDT = count_vect.transform(data_cv_KDT).toarray()
bow_test_KDT = count_vect.transform(data_test_KDT).toarray()
```

[4.2] Bi-Grams and n-Grams.

In [28]:

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

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

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

```
the type of count vectorizer  <class 'scipy.sparse.csr.csr_matrix'>
the shape of out text BOW vectorizer  (364171, 5000)
the number of unique words including both unigrams and bigrams  5000
```

[4.3] TF-IDF

In [29]:

```
#TF-IDF
#BOW
tf_idf_vect = TfidfVectorizer(ngram_range=(1,2), min_df=10)
tf_idf_vect = TfidfVectorizer(ngram_range=(1,2), min_df=10)
tf_idf_vect.fit(data_train_BF)
final_tf_idf_BF = tf_idf_vect.fit_transform(data_train_BF)
tf_idf_train_BF = final_tf_idf_BF
tf_idf_cv_BF = tf_idf_vect.transform(data_cv_BF)
tf_idf_test_BF = tf_idf_vect.transform(data_test_BF)
#KDTree
tf_idf_vect = TfidfVectorizer(ngram_range=(1,2), min_df=10,max_features=500)
tf_idf_vect.fit(data_train_KDT)
final_tf_idf_KDT = tf_idf_vect.fit_transform(data_train_KDT)
tf_idf_train_KDT = final_tf_idf_KDT.toarray()
tf_idf_cv_KDT = tf_idf_vect.transform(data_cv_KDT).toarray()
tf_idf_test_KDT = tf_idf_vect.transform(data_test_KDT).toarray()
```

[4.4] Word2Vec

In [30]:

```
# Train your own Word2Vec model using your own text corpus
w2v_model_BF=Word2Vec(data_train_BF,min_count=5,size=50, workers=4)
w2v_words_BF = list(w2v_model_BF.wv.vocab)

w2v_model_KDT=Word2Vec(data_train_KDT,min_count=5,size=50, workers=4)
w2v_words_KDT = list(w2v_model_KDT.wv.vocab)
```

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

[4.4.1.1] Avg W2v

In [31]:

```
def avg_W2V(list_of_sentance,w2v_model,w2v_words):
    # average Word2Vec
    # compute average word2vec for each review.
    sent_vectors = []; # the avg-w2v for each sentence/review is stored in this list
    for sent in tqdm(list_of_sentance): # for each review/sentence
        sent_vec = np.zeros(50) # as word vectors are of zero length 50, you might need to change t
his to 300 if you use google's w2v
        cnt_words =0; # num of words with a valid vector in the sentence/review
        for word in sent: # for each word in a review/sentence
            if word in w2v_words:
                vec = w2v_model.wv[word]
                sent_vec += vec
                cnt_words += 1
        if cnt_words != 0:
            sent_vec /= cnt_words
        sent_vectors.append(sent_vec)
    return sent_vectors

#Brute
avgw2v_train_BF = avg_W2V(data_train_BF,w2v_model_BF,w2v_words_BF)
avg_w2v_cv_BF = avg_W2V(data_cv_BF,w2v_model_BF,w2v_words_BF)
avg_w2v_test_BF = avg_W2V(data_test_BF,w2v_model_BF,w2v_words_BF)
#KDTree
avgw2v_train_KDT = avg_W2V(data_train_KDT,w2v_model_KDT,w2v_words_KDT)
avg_w2v_cv_KDT = avg_W2V(data_cv_KDT,w2v_model_KDT,w2v_words_KDT)
avg_w2v_test_KDT = avg_W2V(data_test_KDT,w2v_model_KDT,w2v_words_KDT)
```

```
100%|████████████████████████████████████████████████████████████████████████████| 12000/12000 [00:
17<00:00, 671.73it/s]
100%|██████████████████████████████████████████████████████████████████████████████| 4000/4000
[00:05<00:00, 678.90it/s]
100%|██████████████████████████████████████████████████████████████████████████████| 4000/4000
[00:05<00:00, 686.93it/s]
100%|██████████████████████████████████████████████████████████████████████████████| 6000/6000
[00:10<00:00, 599.11it/s]
100%|██████████████████████████████████████████████████████████████████████████████| 2000/2000
[00:03<00:00, 657.64it/s]
100%|██████████████████████████████████████████████████████████████████████████████| 2000/2000
[00:02<00:00, 672.70it/s]
```

[4.4.1.2] TFIDF weighted W2v

In [32]:

```
# S = ["abc def pqr", "def def def abc", "pqr pqr def"]
#Brute
model_BF = TfidfVectorizer()
tf_idf_matrix_BF = model_BF.fit_transform(data_train_BF)
# we are converting a dictionary with word as a key, and the idf as a value
dictionary_BF = dict(zip(model_BF.get_feature_names(), list(model_BF.idf_)))
# TF-IDF weighted Word2Vec
tfidf_feat_BF = model_BF.get_feature_names() # tfidf words/col-names
#KDTree
model_KDT = TfidfVectorizer()
tf_idf_matrix_KDT = model_KDT.fit_transform(data_train_KDT)
# we are converting a dictionary with word as a key, and the idf as a value
dictionary_KDT = dict(zip(model_KDT.get_feature_names(), list(model_KDT.idf_)))
# TF-IDF weighted Word2Vec
tfidf_feat_KDT = model_KDT.get_feature_names() # tfidf words/col-names
```

In [33]:

```
# final_tf_idf is the sparse matrix with row= sentence, col=word and cell_val = tfidf

def tf_idf_w2v(list_of_sentance,w2v_model,w2v_words,tfidf_feat,dictionary):
    tfidf_sent_vectors = []; # the tfidf-w2v for each sentence/review is stored in this list

    for sent in tqdm(list_of_sentance): # for each review/sentence
        sent_vec = np.zeros(50) # as word vectors are of zero length
        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
        tfidf_sent_vectors.append(sent_vec)

    return tfidf_sent_vectors
#Brute
tf_idf_w2v_train_BF =
tf_idf_w2v(data_train_BF,w2v_model_BF,w2v_words_BF,tfidf_feat_BF,dictionary_BF)
tf_idf_w2v_cv_BF = tf_idf_w2v(data_cv_BF,w2v_model_BF,w2v_words_BF,tfidf_feat_BF,dictionary_BF)
tf_idf_w2v_test_BF = tf_idf_w2v(data_test_BF,w2v_model_BF,w2v_words_BF,tfidf_feat_BF,dictionary_BF
)
#KDTree
tf_idf_w2v_train_KDT =
tf_idf_w2v(data_train_KDT,w2v_model_KDT,w2v_words_KDT,tfidf_feat_KDT,dictionary_KDT)
tf_idf_w2v_cv_KDT =
tf_idf_w2v(data_cv_KDT,w2v_model_KDT,w2v_words_KDT,tfidf_feat_KDT,dictionary_KDT)
tf_idf_w2v_test_KDT =
tf_idf_w2v(data_test_KDT,w2v_model_KDT,w2v_words_KDT,tfidf_feat_KDT,dictionary_KDT)
```

```
100%|████████████████████████████████████████████████████████████| 12000/12000 [30
:31<00:00,  6.01it/s]
100%|██████████████████████████████████████████████████████████████| 4000/4000 [10
:46<00:00,  6.19it/s]
100%|██████████████████████████████████████████████████████████████| 4000/4000 [08
:59<00:00,  5.42it/s]
100%|██████████████████████████████████████████████████████████████| 6000/6000 [08
:49<00:00, 16.93it/s]
100%|██████████████████████████████████████████████████████████████| 2000/2000 [02
:52<00:00, 10.46it/s]
100%|██████████████████████████████████████████████████████████████| 2000/2000 [02
:49<00:00, 11.78it/s]
```

[5] Assignment 3: KNN

1. **Apply Knn(brute force version) on these feature sets**

    - SET 1:Review text, preprocessed one converted into vectors using (BOW)
    - SET 2:Review text, preprocessed one converted into vectors using (TFIDF)
    - SET 3:Review text, preprocessed one converted into vectors using (AVG W2v)
    - SET 4:Review text, preprocessed one converted into vectors using (TFIDF W2v)

2. **Apply Knn(kd tree version) on these feature sets**
    NOTE: sklearn implementation of kd-tree accepts only dense matrices, you need to convert the sparse matrices of CountVectorizer/TfidfVectorizer into dense matices. You can convert sparse matrices to dense using .toarray() attribute. For more information please visit this link

    - SET 5:Review text, preprocessed one converted into vectors using (BOW) but with restriction on maximum features generated.

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

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

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

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

3. **The hyper paramter tuning(find best K)**

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

4. **Representation of results**

    - You need to plot the performance of model both on train data and cross validation data for each hyper parameter, like shown in the figure
    - Once after you found the best hyper parameter, you need to train your model with it, and find the AUC on test data and plot the ROC curve on both train and test.
    - Along with plotting ROC curve, you need to print the confusion matrix with predicted and original labels of test data points

5. **Conclusion**

    - You need to summarize the results at the end of the notebook, summarize it in the table format. To print out a table please refer to this prettytable library link

Note: Data Leakage

1. There will be an issue of data-leakage if you vectorize the entire data and then split it into train/cv/test.
2. To avoid the issue of data-leakag, make sure to split your data first and then vectorize it.
3. While vectorizing your data, apply the method fit_transform() on you train data, and apply the method transform() on cv/test data.
4. For more details please go through this link.

[5.1] Applying KNN brute force

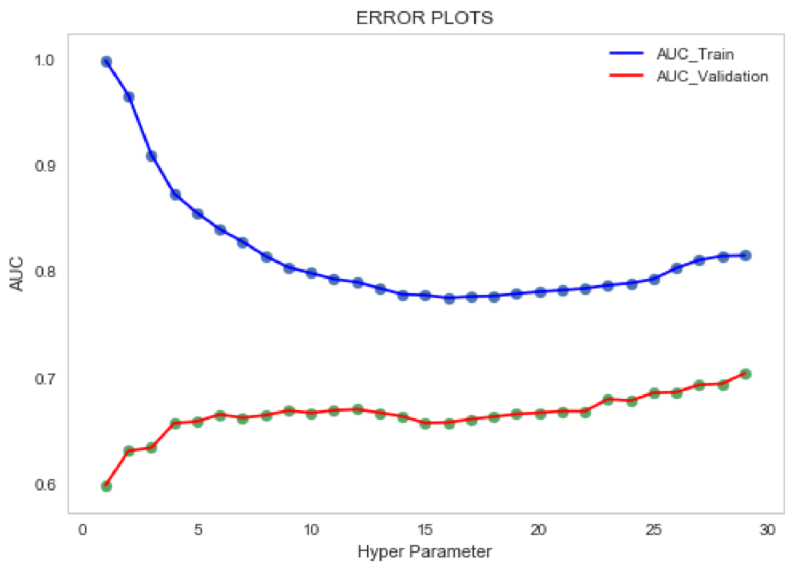
In [34]:

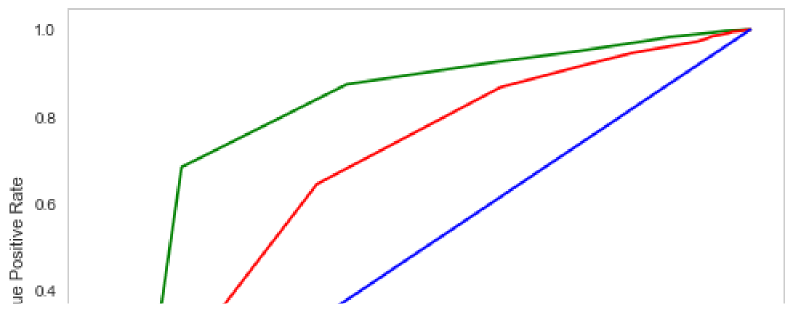
```
from sklearn.neighbors import KNeighborsClassifier

def get_AUC(X_train,y_train,X_cv,y_cv,alg,k_range):
    """This function apply the knn model with given algorithm
       on train and cv data and return AUC values for train and cross validation"""
    auc_train = []
    auc_cv = []
    # applying KNN Brute Force algorithm on list of hyper parameters to find best k using simple l
oop
    for k in k_range:

        knn = KNeighborsClassifier(algorithm = alg,n_neighbors=k)
        knn.fit(X_train,y_train)
        prob_train = knn.predict_proba(X_train)
        fpr, tpr, threshold = roc_curve(y_train, prob_train[:, 1])
        auc_train.append(auc(fpr,tpr))

        prob_cv = knn.predict_proba(X_cv)
        fpr, tpr, threshold = roc_curve(y_cv, prob_cv[:, 1])
        auc_cv.append(auc(fpr,tpr))

    return auc_train,auc_cv

def plot_AUC_Curves(auc_tran,auc_cv,k_range):
    """This function plots the auc curves for the given auc values and k_range"""
    sns.set_style("whitegrid",{'axes.grid' : False})
    plt.plot(k_range,auc_train,"b-",label = "AUC_Train")
    plt.plot(k_range,auc_cv,"r-",label = "AUC_Validation")
    plt.scatter(k_range,auc_train)
    plt.scatter(k_range,auc_cv)
    plt.legend()
    plt.xlabel("Hyper Parameter")
    plt.ylabel("AUC")
    plt.title("ERROR PLOTS")
    plt.show()

def apply_roc_curve(X_train,y_train,X_test,y_test,alg,optimal_k):
    """This function apply knn model on train and predict labels for test data
       and also find FPR and TPR for train and test data.
       Returns the predicted labels,FPR and TPR values"""
    knn = KNeighborsClassifier(algorithm = alg,n_neighbors=optimal_k)
    knn.fit(X_train,y_train)
    prob_train = knn.predict_proba(X_train)
    fpr_train, tpr_train, threshold = roc_curve(y_train, prob_train[:, 1])
    prob_test = knn.predict_proba(X_test)
    fpr_test, tpr_test, threshold = roc_curve(y_test, prob_test[:, 1])

    # predict the class labels
    pred_train = knn.predict(X_train)
    pred_test = knn.predict(X_test)

    return fpr_train,tpr_train,fpr_test,tpr_test,pred_train,pred_test

def plot_roc_curve(fpr_train,tpr_train,fpr_test,tpr_test):
    """This function plot the roc curves for train and test data"""
    # plot ROC curves for train and test data
    plt.plot(fpr_train,tpr_train,"g-",label = "AUC_Train : "+str(auc(fpr_train, tpr_train)))
    plt.plot(fpr_test,tpr_test,"r-",label = "AUC_Test : "+str(auc(fpr_test, tpr_test)))
    plt.plot([0,1],[0,1],"b-")
    plt.legend(loc="lower right")
    plt.xlabel("False Positive Rate")
    plt.ylabel("True Positive Rate")
    plt.show()

def plot_Confusion_Matrix(actual_labels,predict_labels,title):
    """This function plot the confusion matrix"""
    # Reference : https://seaborn.pydata.org/generated/seaborn.heatmap.html
    cm = confusion_matrix(actual_labels, predict_labels)
    classNames = ['NO','YES']
    cm_data = pd.DataFrame(cm,index = classNames,
                  columns = classNames)
    plt.figure(figsize = (5,4))
    sns.heatmap(cm_data, annot=True,fmt="d")
    plt.title(title)
    plt.ylabel('Actual label')
    plt.xlabel('Predicted label')
    plt.show()
```

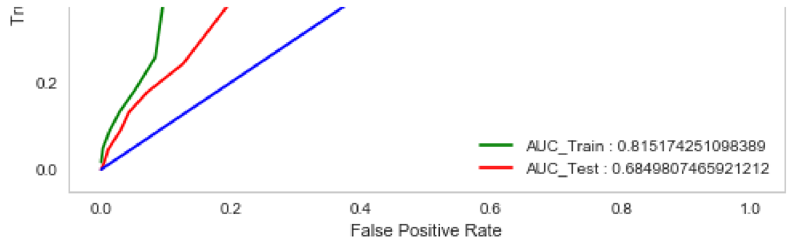
[5.1.1] Applying KNN brute force on BOW, SET 1

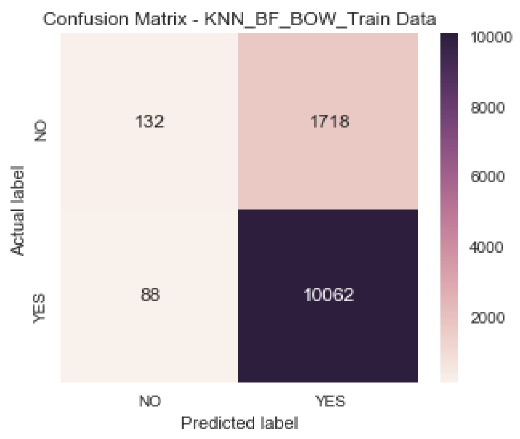
In [35]:

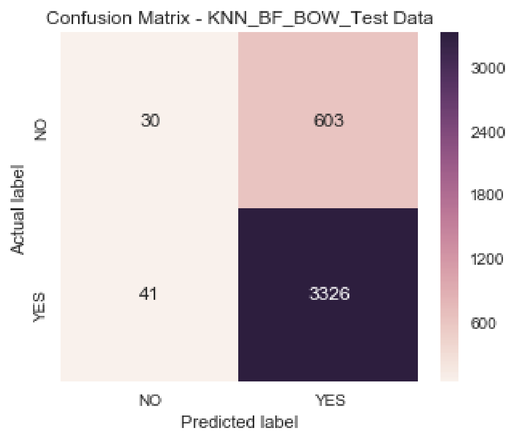
```
# Please write all the code with proper documentation

# list of hyper parameters
k_range = range(1,30)

# applying KNN Brute Force algorithm on list of hyper parameters to find best k using simple loop
auc_train,auc_cv = get_AUC(bow_train_BF,scores_train_BF,bow_cv_BF,scores_cv_BF,'brute',k_range)

# plot AUC curves for train and cross validation data
plot_AUC_Curves(auc_train,auc_cv,k_range)

# Best hyper parameter K
optimal_k_KNN_BF_BOW = auc_cv.index(max(auc_cv)) + 1

# apply KNN Brute force algorithm with best K
fpr_train,tpr_train,fpr_test,tpr_test,pred_train,pred_test =
apply_roc_curve(bow_train_BF,scores_train_BF,bow_test_BF,scores_test_BF,'brute',optimal_k_KNN_BF_BO
W)

#AUC
auc_KNN_BF_BOW = auc(fpr_test,tpr_test)

# plot ROC curves for train and test data
plot_roc_curve(fpr_train,tpr_train,fpr_test,tpr_test)

# Confusion matrix
plot_Confusion_Matrix(scores_train_BF,pred_train,'Confusion Matrix - KNN_BF_BOW_Train Data')
plot_Confusion_Matrix(scores_test_BF,pred_test,'Confusion Matrix - KNN_BF_BOW_Test Data')

print("Hyper parameter : ",optimal_k_KNN_BF_BOW)
print("AUC : %.2f" %auc_KNN_BF_BOW)
```











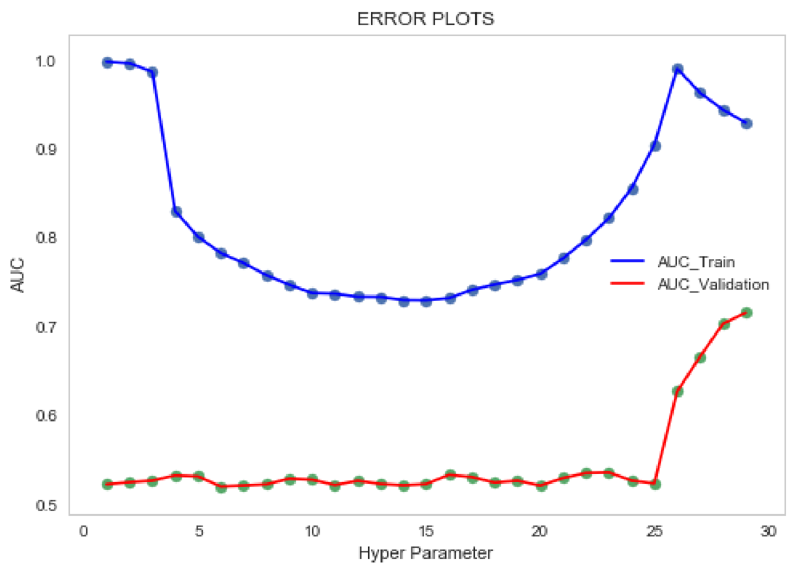
```
Hyper parameter :  29
AUC : 0.68
```

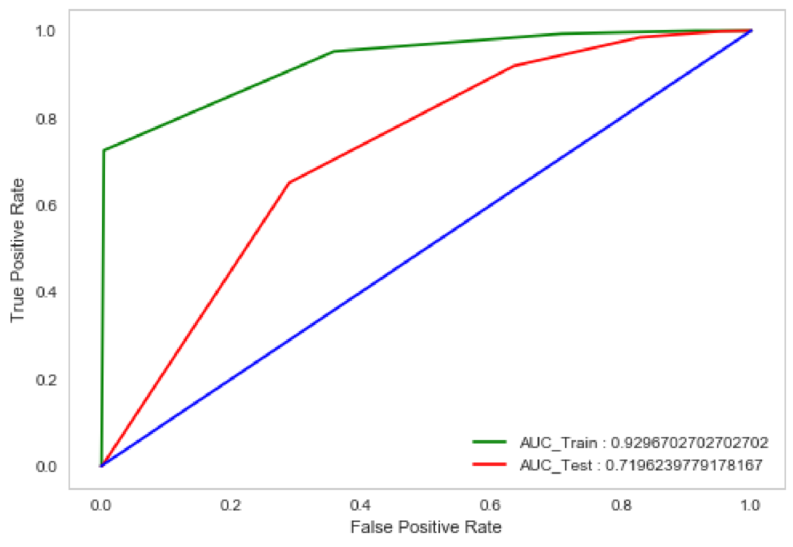
[5.1.2] Applying KNN brute force on TFIDF, SET 2

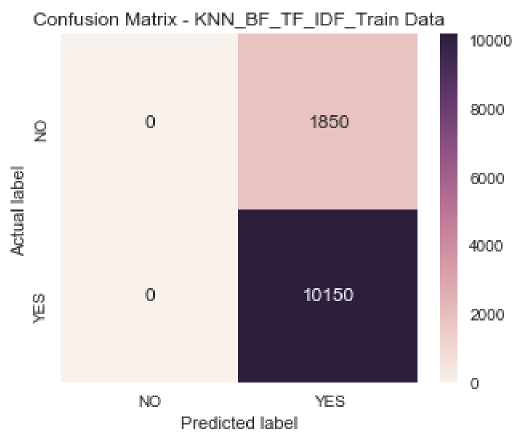
In [36]:

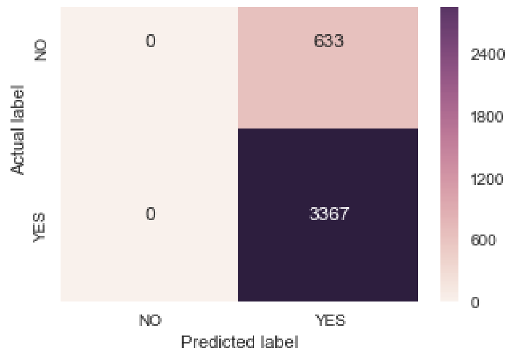
```
# Please write all the code with proper documentation

# list of hyper parameters
k_range = range(1,30)

# applying KNN Brute Force algorithm on list of hyper parameters to find best k using simple loop
auc_train,auc_cv =
get_AUC(tf_idf_train_BF,scores_train_BF,tf_idf_cv_BF,scores_cv_BF,'brute',k_range)

# plot AUC curves for train and cross validation data
plot_AUC_Curves(auc_train,auc_cv,k_range)

# Best hyper parameter K
optimal_k_KNN_BF_TF_IDF = auc_cv.index(max(auc_cv)) + 1

# apply KNN Brute force algorithm with best K
fpr_train,tpr_train,fpr_test,tpr_test,pred_train,pred_test =
apply_roc_curve(tf_idf_train_BF,scores_train_BF,tf_idf_test_BF,scores_test_BF,'brute',optimal_k_KNN
_BF_TF_IDF)

#AUC
auc_KNN_BF_TF_IDF = auc(fpr_test,tpr_test)
```

```
# plot ROC curves for train and test data
plot_roc_curve(fpr_train,tpr_train,fpr_test,tpr_test)

# Confusion matrix
plot_Confusion_Matrix(scores_train_BF,pred_train,'Confusion Matrix - KNN_BF_TF_IDF_Train Data')
plot_Confusion_Matrix(scores_test_BF,pred_test,'Confusion Matrix - KNN_BF_TF_IDF_Test Data')

print("Hyper parameter : ",optimal_k_KNN_BF_TF_IDF)
print("AUC : %.2f" %auc_KNN_BF_TF_IDF)
```







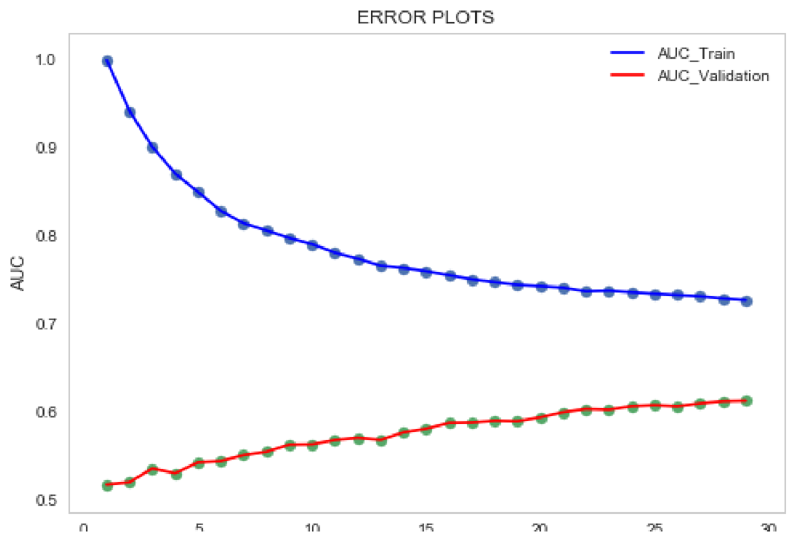


```
Hyper parameter :  29
AUC : 0.72
```

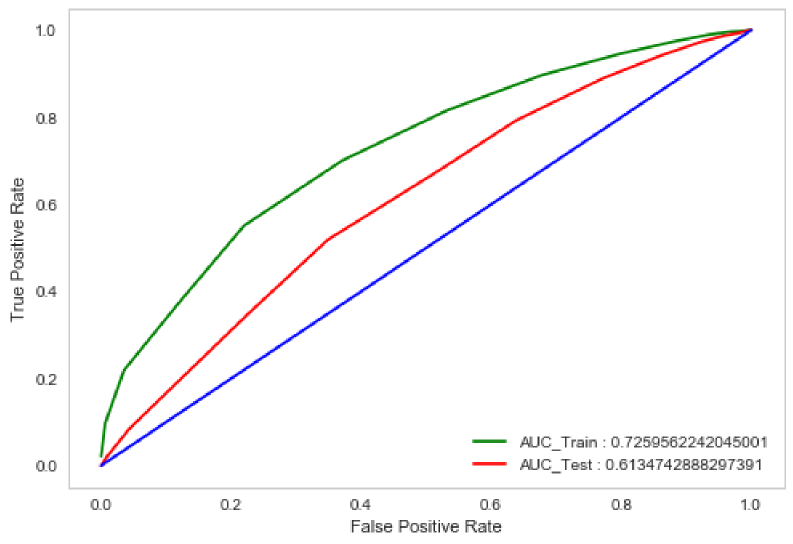
[5.1.3] Applying KNN brute force on AVG W2V, SET 3

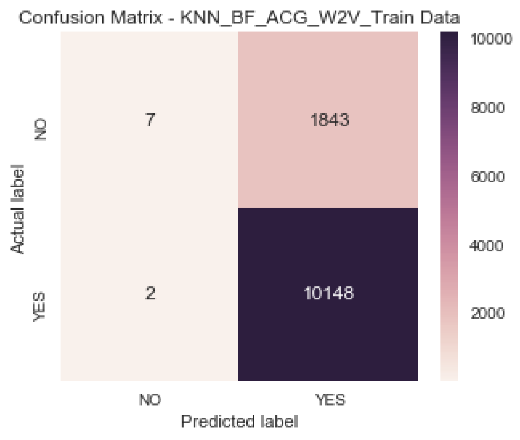
In [37]:

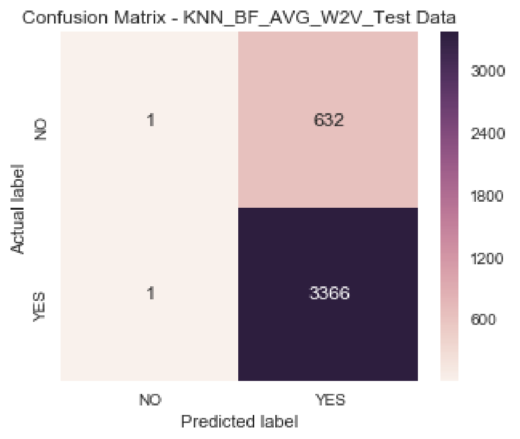
```
# Please write all the code with proper documentation

# list of hyper parameters
k_range = range(1,30)

# applying KNN Brute Force algorithm on list of hyper parameters to find best k using simple loop
auc_train,auc_cv =
get_AUC(avgw2v_train_BF,scores_train_BF,avg_w2v_cv_BF,scores_cv_BF,'brute',k_range)

# plot AUC curves for train and cross validation data
plot_AUC_Curves(auc_train,auc_cv,k_range)

# Best hyper parameter K
optimal_k_KNN_BF_AVG_W2V = auc_cv.index(max(auc_cv)) + 1

# apply KNN Brute force algorithm with best K
fpr_train,tpr_train,fpr_test,tpr_test,pred_train,pred_test =
apply_roc_curve(avgw2v_train_BF,scores_train_BF,avg_w2v_test_BF,scores_test_BF,'brute',optimal_k_KN
N_BF_AVG_W2V)

#AUC
auc_KNN_BF_AVG_W2V = auc(fpr_test,tpr_test)

# plot ROC curves for train and test data
plot_roc_curve(fpr_train,tpr_train,fpr_test,tpr_test)

# Confusion matrix
plot_Confusion_Matrix(scores_train_BF,pred_train,'Confusion Matrix - KNN_BF_ACG_W2V_Train Data')
plot_Confusion_Matrix(scores_test_BF,pred_test,'Confusion Matrix - KNN_BF_AVG_W2V_Test Data')

print("Hyper parameter : ",optimal_k_KNN_BF_AVG_W2V)
print("AUC : %.2f" %auc_KNN_BF_AVG_W2V)
```



Hyper Parameter





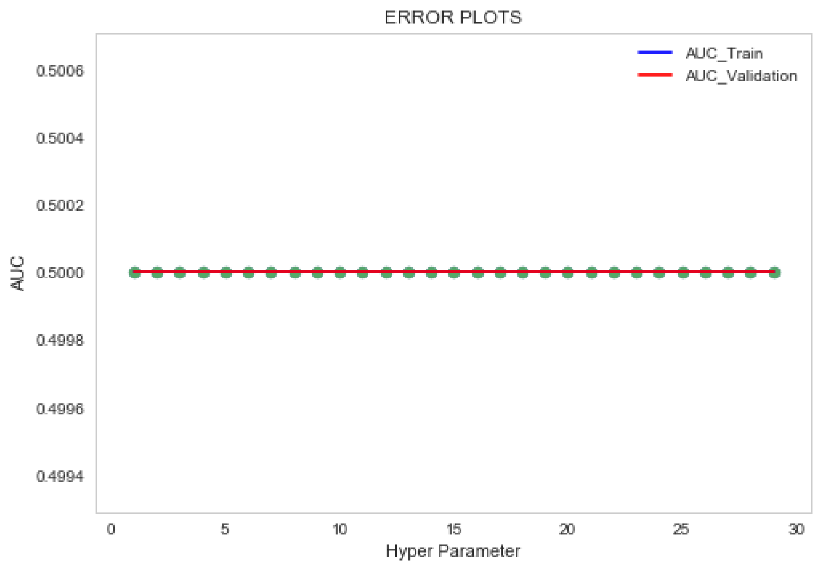


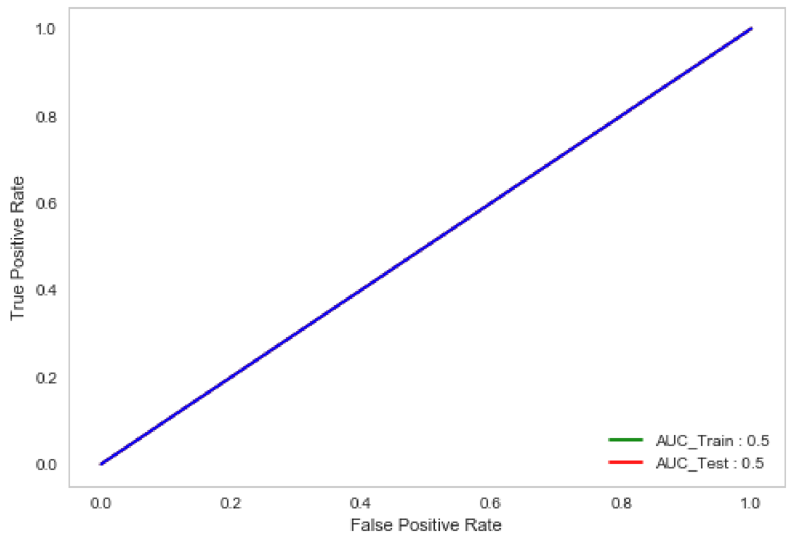
```
Hyper parameter :  29
AUC : 0.61
```

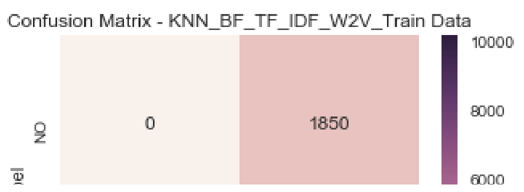
[5.1.4] Applying KNN brute force on TFIDF W2V, SET 4

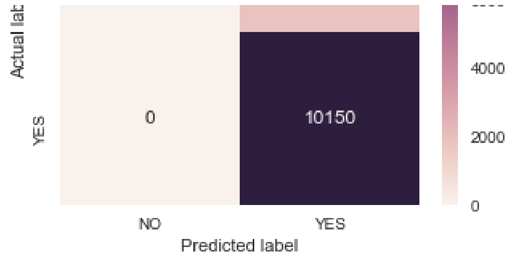
In [38]:

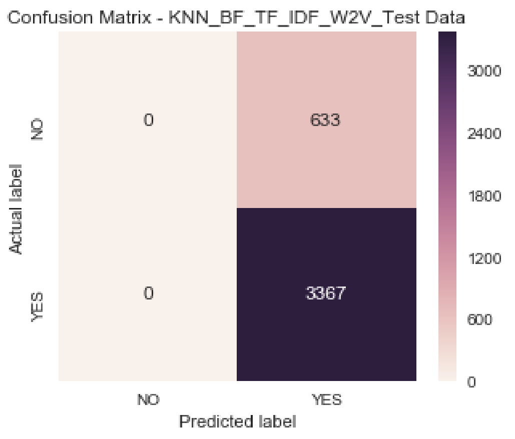
```
# Please write all the code with proper documentation

# list of hyper parameters
k_range = range(1,30)

# applying KNN Brute Force algorithm on list of hyper parameters to find best k using simple loop
auc_train,auc_cv =
get_AUC(tf_idf_w2v_train_BF,scores_train_BF,tf_idf_w2v_cv_BF,scores_cv_BF,'brute',k_range)
get_AUC(tf_idf_w2v_train_BF,scores_train_BF,tf_idf_w2v_cv_BF,scores_cv_BF,'brute',k_range)

# plot AUC curves for train and cross validation data
plot_AUC_Curves(auc_train,auc_cv,k_range)

# Best hyper parameter K
optimal_k_KNN_BF_TF_IDF_W2V = auc_cv.index(max(auc_cv)) + 1

# apply KNN Brute force algorithm with best K
fpr_train,tpr_train,fpr_test,tpr_test,pred_train,pred_test = apply_roc_curve(tf_idf_w2v_train_BF,s
cores_train_BF,tf_idf_w2v_test_BF,scores_test_BF,'brute',optimal_k_KNN_BF_TF_IDF_W2V)

#AUC
auc_KNN_BF_TF_IDF_W2V = auc(fpr_test,tpr_test)

# plot ROC curves for train and test data
plot_roc_curve(fpr_train,tpr_train,fpr_test,tpr_test)

# Confusion matrix
plot_Confusion_Matrix(scores_train_BF,pred_train,'Confusion Matrix - KNN_BF_TF_IDF_W2V_Train Data'
)
plot_Confusion_Matrix(scores_test_BF,pred_test,'Confusion Matrix - KNN_BF_TF_IDF_W2V_Test Data')

print("Hyper parameter : ",optimal_k_KNN_BF_TF_IDF_W2V)
print("AUC : %.2f" %auc_KNN_BF_TF_IDF_W2V)
```











```
Hyper parameter :  1
AUC : 0.50
```

[5.2] Applying KNN kd-tree

In [ ]:

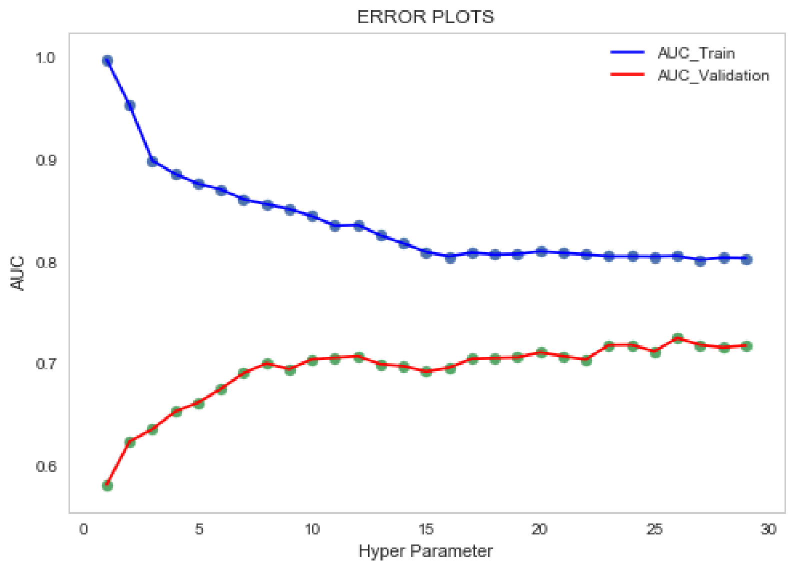
```

```

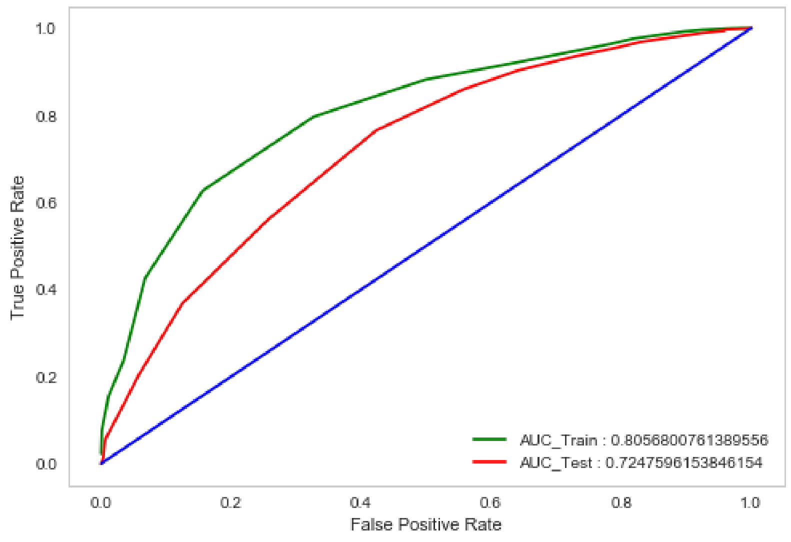
[5.2.1] Applying KNN kd-tree on BOW, SET 5

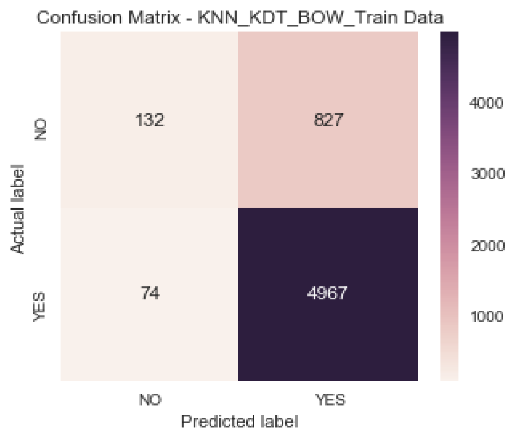
In [39]:

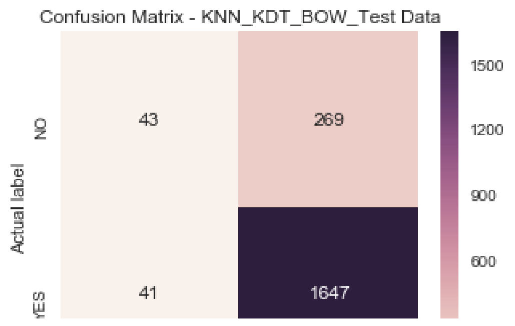
```
# Please write all the code with proper documentation

# list of hyper parameters
k_range = range(1,30)

# applying KNN Brute Force algorithm on list of hyper parameters to find best k using simple loop
auc_train,auc_cv =
get_AUC(bow_train_KDT,scores_train_KDT,bow_cv_KDT,scores_cv_KDT,'kd_tree',k_range)

# plot AUC curves for train and cross validation data
plot_AUC_Curves(auc_train,auc_cv,k_range)

# Best hyper parameter K
optimal_k_KNN_KDT_BOW = auc_cv.index(max(auc_cv)) + 1

# apply KNN Brute force algorithm with best K
fpr_train,tpr_train,fpr_test,tpr_test,pred_train,pred_test =
apply_roc_curve(bow_train_KDT,scores_train_KDT,bow_test_KDT,scores_test_KDT,'kd_tree',optimal_k_KNN
_KDT_BOW)

#AUC
auc_KNN_KDT_BOW = auc(fpr_test,tpr_test)

# plot ROC curves for train and test data
plot_roc_curve(fpr_train,tpr_train,fpr_test,tpr_test)

# Confusion matrix
plot_Confusion_Matrix(scores_train_KDT,pred_train,'Confusion Matrix - KNN_KDT_BOW_Train Data')
plot_Confusion_Matrix(scores_test_KDT,pred_test,'Confusion Matrix - KNN_KDT_BOW_Test Data')
```

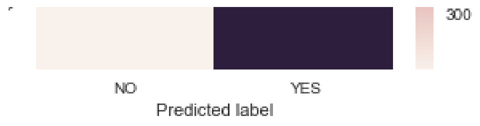
```
print("Hyper parameter : ",optimal_k_KNN_KDT_BOW)
print("AUC : %.2f" %auc_KNN_KDT_BOW)
```









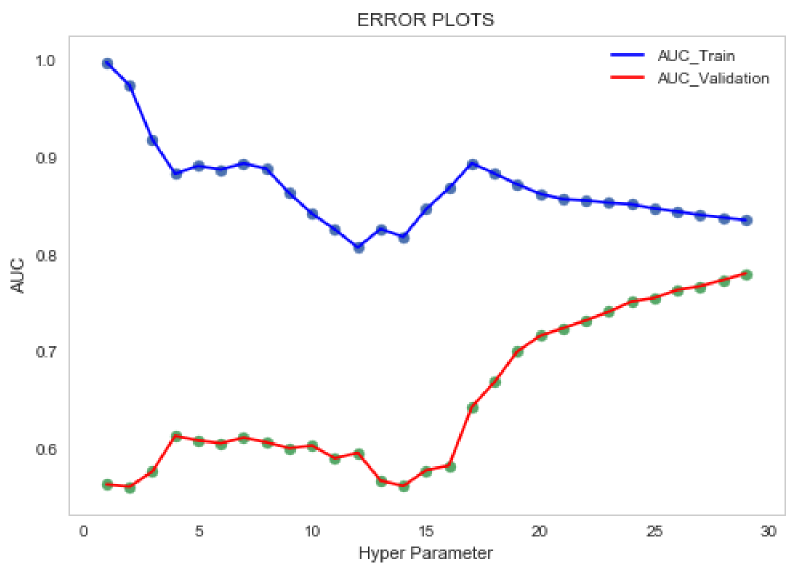


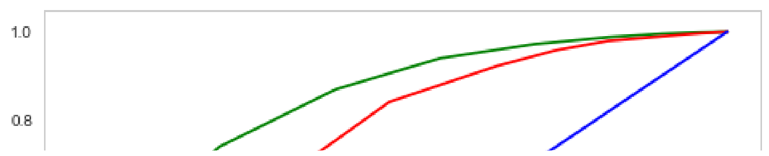
```
Hyper parameter :  26
AUC : 0.72
```

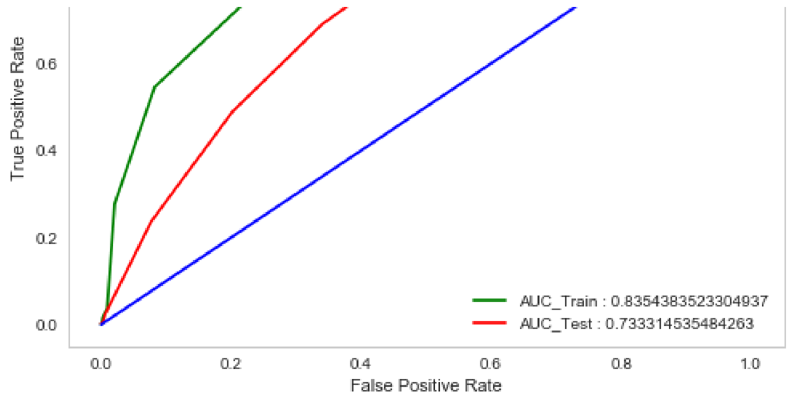
[5.2.2] Applying KNN kd-tree on TFIDF, SET 6

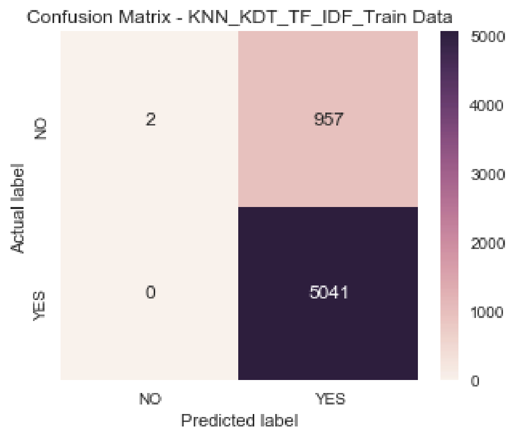
In [40]:

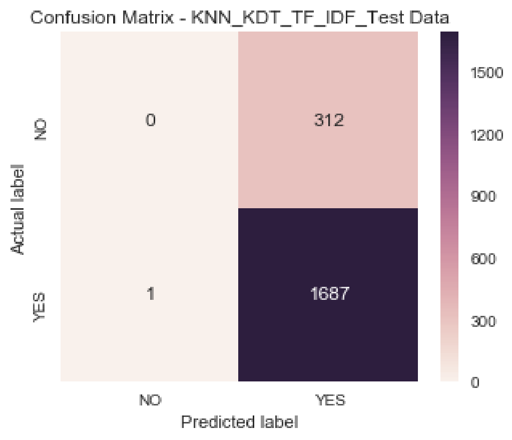
```
# Please write all the code with proper documentation

# list of hyper parameters
k_range = range(1,30)

# applying KNN Brute Force algorithm on list of hyper parameters to find best k using simple loop
auc_train,auc_cv = get_AUC(tf_idf_train_KDT,scores_train_KDT,tf_idf_cv_KDT,scores_cv_KDT,'kd_tree'
,k_range)

# plot AUC curves for train and cross validation data
plot_AUC_Curves(auc_train,auc_cv,k_range)

# Best hyper parameter K
optimal_k_KNN_KDT_TF_IDF = auc_cv.index(max(auc_cv)) + 1

# apply KNN Brute force algorithm with best K
fpr_train,tpr_train,fpr_test,tpr_test,pred_train,pred_test =
apply_roc_curve(tf_idf_train_KDT,scores_train_KDT,tf_idf_test_KDT,scores_test_KDT,'kd_tree',optimal
_k_KNN_KDT_TF_IDF)

#AUC
auc_KNN_KDT_TF_IDF = auc(fpr_test,tpr_test)

# plot ROC curves for train and test data
plot_roc_curve(fpr_train,tpr_train,fpr_test,tpr_test)

# Confusion matrix
plot_Confusion_Matrix(scores_train_KDT,pred_train,'Confusion Matrix - KNN_KDT_TF_IDF_Train Data')
plot_Confusion_Matrix(scores_test_KDT,pred_test,'Confusion Matrix - KNN_KDT_TF_IDF_Test Data')

print("Hyper parameter : ",optimal_k_KNN_KDT_TF_IDF)
print("AUC : %.2f" %auc_KNN_KDT_TF_IDF)
```











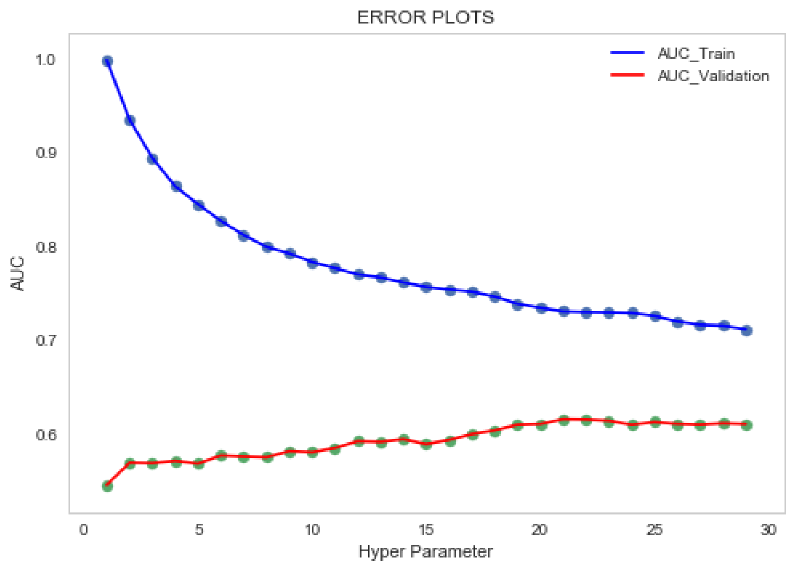
```
Hyper parameter :  29
AUC : 0.73
```

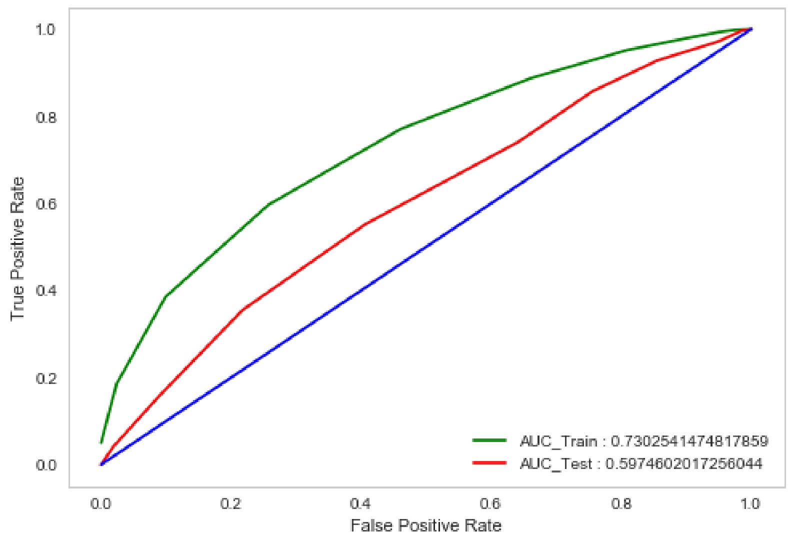
[5.2.3] Applying KNN kd-tree on AVG W2V, SET 3

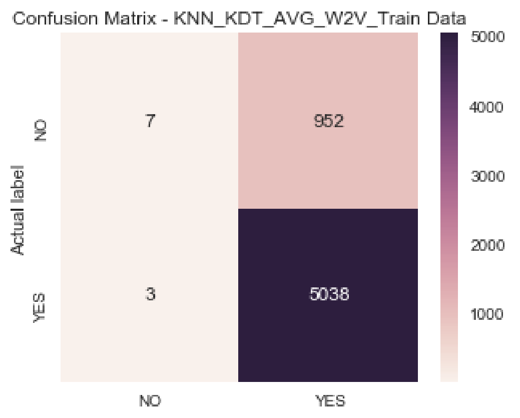
In [41]:

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

# list of hyper parameters
k_range = range(1,30)

# applying KNN Brute Force algorithm on list of hyper parameters to find best k using simple loop
auc_train,auc_cv =
get_AUC(avgw2v_train_KDT,scores_train_KDT,avg_w2v_cv_KDT,scores_cv_KDT,'kd_tree',k_range)

# plot AUC curves for train and cross validation data
plot_AUC_Curves(auc_train,auc_cv,k_range)

# Best hyper parameter K
optimal_k_KNN_KDT_AVG_W2V = auc_cv.index(max(auc_cv)) + 1

# apply KNN Brute force algorithm with best K
```

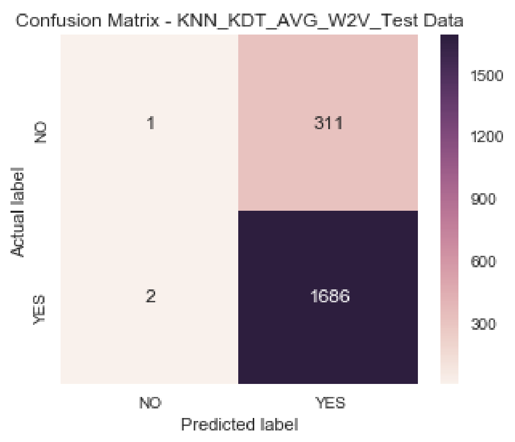
```
fpr_train,tpr_train,fpr_test,tpr_test,pred_train,pred_test =
apply_roc_curve(avgw2v_train_KDT,scores_train_KDT,avg_w2v_test_KDT,scores_test_KDT,'kd_tree',optima
l_k_KNN_KDT_AVG_W2V)

#AUC
auc_KNN_KDT_AVG_W2V = auc(fpr_test,tpr_test)

# plot ROC curves for train and test data
plot_roc_curve(fpr_train,tpr_train,fpr_test,tpr_test)

# Confusion matrix
plot_Confusion_Matrix(scores_train_KDT,pred_train,'Confusion Matrix - KNN_KDT_AVG_W2V_Train Data')
plot_Confusion_Matrix(scores_test_KDT,pred_test,'Confusion Matrix - KNN_KDT_AVG_W2V_Test Data')

print("Hyper parameter : ",optimal_k_KNN_KDT_AVG_W2V)
print("AUC : %.2f" %auc_KNN_KDT_AVG_W2V)
```







Predicted label

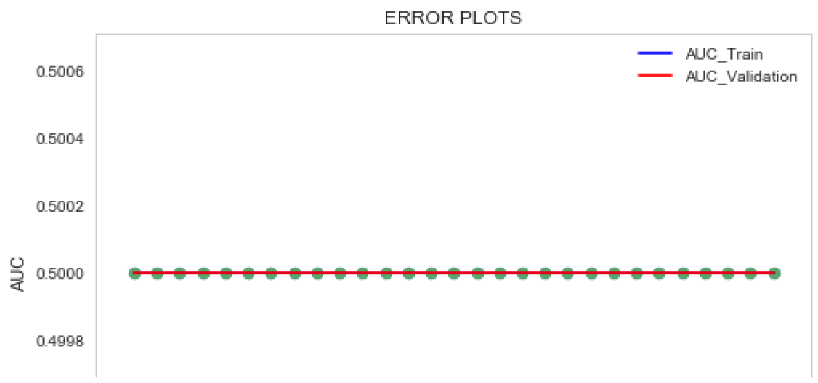


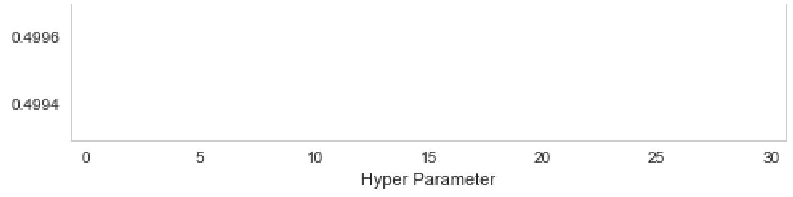
```
Hyper parameter :  21
AUC : 0.60
```

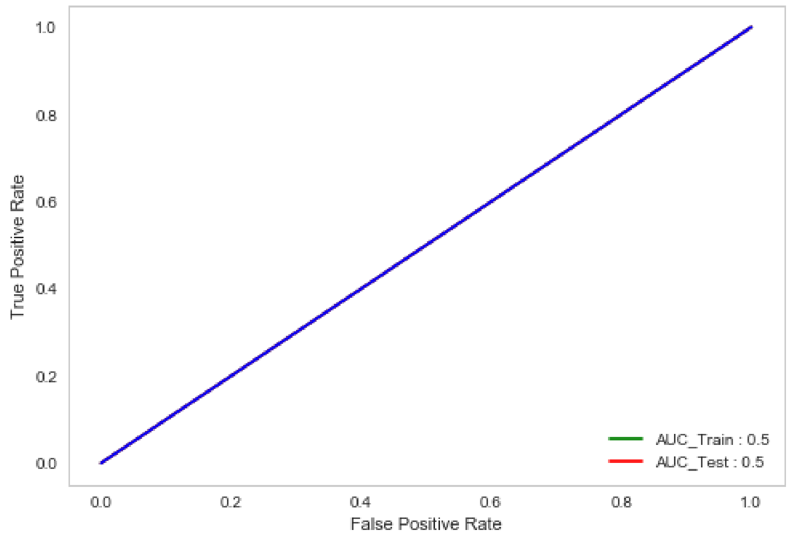
[5.2.4] Applying KNN kd-tree on TFIDF W2V, SET 4

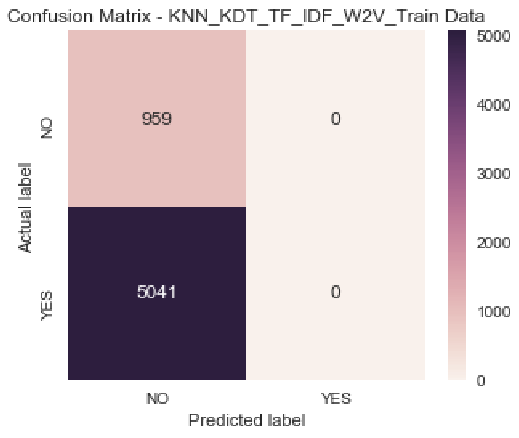
In [42]:

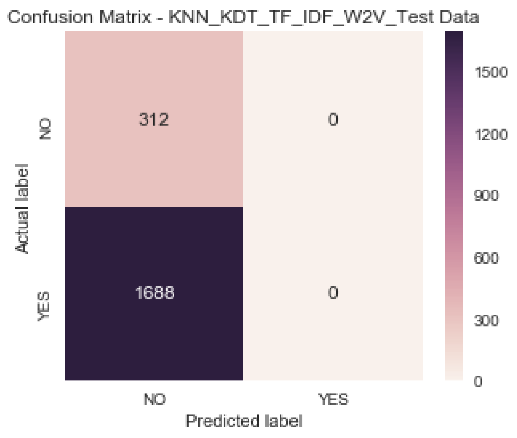
```
# Please write all the code with proper documentation

# list of hyper parameters
k_range = range(1,30)

# applying KNN Brute Force algorithm on list of hyper parameters to find best k using simple loop
auc_train,auc_cv = get_AUC(tf_idf_w2v_train_KDT,scores_train_KDT,tf_idf_w2v_cv_KDT,scores_cv_KDT,'
kd_tree',k_range)

# plot AUC curves for train and cross validation data
plot_AUC_Curves(auc_train,auc_cv,k_range)

# Best hyper parameter K
optimal_k_KNN_KDT_TF_IDF_W2V = auc_cv.index(max(auc_cv)) + 1

# apply KNN Brute force algorithm with best K
fpr_train,tpr_train,fpr_test,tpr_test,pred_train,pred_test = apply_roc_curve(tf_idf_w2v_train_KDT,
scores_train_KDT,tf_idf_w2v_test_KDT,scores_test_KDT,'kd_tree',optimal_k_KNN_KDT_TF_IDF_W2V)

#AUC
auc_KNN_KDT_TF_IDF_W2V = auc(fpr_test,tpr_test)

# plot ROC curves for train and test data
plot_roc_curve(fpr_train,tpr_train,fpr_test,tpr_test)

# Confusion matrix
plot_Confusion_Matrix(scores_train_KDT,pred_train,'Confusion Matrix - KNN_KDT_TF_IDF_W2V_Train Dat
a')
plot_Confusion_Matrix(scores_test_KDT,pred_test,'Confusion Matrix - KNN_KDT_TF_IDF_W2V_Test Data')

print("Hyper parameter : ",optimal_k_KNN_KDT_TF_IDF_W2V)
print("AUC : %.2f" %auc_KNN_KDT_TF_IDF_W2V)
```











```
Hyper parameter :  1
AUC : 0.50
```

[6] Conclusions

In [43]:

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

```
from prettytable import PrettyTable

table = PrettyTable()
table.field_names = ["Vectorizer","Model","Hyper parameter","AUC"]
table.add_row(["BOW","Brute",optimal_k_KNN_BF_BOW,round(auc_KNN_BF_BOW,2)])
table.add_row(["TFIDF","Brute",optimal_k_KNN_BF_TF_IDF,round(auc_KNN_BF_TF_IDF,2)])
table.add_row(["W2V","Brute",optimal_k_KNN_BF_AVG_W2V,round(auc_KNN_BF_AVG_W2V,2)])
table.add_row(["TFIDFW2V","Brute",optimal_k_KNN_BF_TF_IDF_W2V,round(auc_KNN_BF_TF_IDF_W2V,2)])
table.add_row(["BOW","kd_tree",optimal_k_KNN_KDT_BOW,round(auc_KNN_KDT_BOW,2)])
table.add_row(["TFIDF","kd_tree",optimal_k_KNN_KDT_TF_IDF,round(auc_KNN_KDT_TF_IDF,2)])
table.add_row(["W2V","kd_tree",optimal_k_KNN_KDT_AVG_W2V,round(auc_KNN_KDT_AVG_W2V,2)])
table.add_row(["TFIDFW2V","kd_tree",optimal_k_KNN_KDT_TF_IDF_W2V,round(auc_KNN_KDT_TF_IDF_W2V,2)])

print(table.get_string(title="Results"))
```

```
+-------------------------------------------------+
|                     Results                     |
+------------+---------+-----------------+------+
| Vectorizer |  Model  | Hyper parameter | AUC  |
+------------+---------+-----------------+------+
|    BOW     |  Brute  |        29       | 0.68 |
|   TFIDF    |  Brute  |        29       | 0.72 |
|    W2V     |  Brute  |        29       | 0.61 |
|  TFIDFW2V  |  Brute  |        1        | 0.5  |
|    BOW     | kd_tree |        26       | 0.72 |
|   TFIDF    | kd_tree |        29       | 0.73 |
|    W2V     | kd_tree |        21       | 0.6  |
|  TFIDFW2V  | kd_tree |        1        | 0.5  |
+------------+---------+-----------------+------+
```

In [ ]: