Amazon Fine Food Reviews Analysis

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

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

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

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

Attribute Information:

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

Objective:

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

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

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

[1]. Reading Data

[1.1] Loading the data

The dataset is available in two forms

1. .csv file
2. SQLite Database

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

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

In [1]:

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


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

```
import seaborn as sns
from sklearn.feature_extraction.text import TfidfTransformer
from sklearn.feature_extraction.text import TfidfVectorizer

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

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

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

from tqdm import tqdm
import os
```

In [2]:

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

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

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

filtered_data = pd.read_sql_query(""" SELECT * FROM Reviews WHERE Score != 3 LIMIT 20000""", con)

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

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

```
Number of data points in our data (20000, 10)
```

Out[2]:

	Id	ProductId	UserId	ProfileName	HelpfulnessNumerator	HelpfulnessDenominator	Score	Time	Summary
0	1	B001E4KFG0	A3SGXH7AUHU8GW	delmartian	1	1	1	1303862400	Good Quality Dog Food
1	2	B00813GRG4	A1D87F6ZCVE5NK	dll pa	0	0	0	1346976000	Not as Advertised
2	3	B000LQOCH0	ABXLMWJIXXAIN	Natalia Corres "Natalia Corres"	1	1	1	1219017600	"Delight" says it all

Id	ProductId	UserId	ProfileName	HelpfulnessNumerator	HelpfulnessDenominator	Score	Time	Summary

In [3]:

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

In [4]:

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

```
(80668, 7)
```

Out[4]:

	UserId	ProductId	ProfileName	Time	Score	Text	COUNT(*)
0	#oc-R115TNMSPFT9I7	B005ZBZLT4	Breyton	1331510400	2	Overall its just OK when considering the price...	2
1	#oc-R11D9D7SHXIJB9	B005HG9ESG	Louis E. Emory "hoppy"	1342396800	5	My wife has recurring extreme muscle spasms, u...	3
2	#oc-R11DNU2NBKQ23Z	B005ZBZLT4	Kim Cieszykowski	1348531200	1	This coffee is horrible and unfortunately not ...	2
3	#oc-R11O5J5ZVQE25C	B005HG9ESG	Penguin Chick	1346889600	5	This will be the bottle that you grab from the...	3
4	#oc-R12KPBODL2B5ZD	B007OSBEV0	Christopher P. Presta	1348617600	1	I didnt like this coffee. Instead of telling y...	2

In [5]:

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

Out[5]:

	UserId	ProductId	ProfileName	Time	Score	Text	COUNT(*)
80638	AZY10LLTJ71NX	B001ATMQK2	undertheshrine "undertheshrine"	1296691200	5	I bought this 6 pack because for the price tha...	5

In [6]:

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

Out[6]:

```
393063
```

[2] Exploratory Data Analysis

[2.1] Data Cleaning: Deduplication

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

In [7]:

```
display= pd.read_sql_query("""
SELECT *
FROM Reviews
WHERE Score != 3 AND UserId="AR5J8UI46CURR"
```

```
ORDER BY ProductID
""", con)
display.head()
```

Out[7]:

	Id	ProductId	UserId	ProfileName	HelpfulnessNumerator	HelpfulnessDenominator	Score	Time	Summ
0	78445	B000HDL1RQ	AR5J8UI46CURR	Geetha Krishnan	2	2	5	1199577600	LOACK QUADRAT VANIL WAFE
1	138317	B000HDOPYC	AR5J8UI46CURR	Geetha Krishnan	2	2	5	1199577600	LOACK QUADRAT VANIL WAFE
2	138277	B000HDOPYM	AR5J8UI46CURR	Geetha Krishnan	2	2	5	1199577600	LOACK QUADRAT VANIL WAFE
3	73791	B000HDOPZG	AR5J8UI46CURR	Geetha Krishnan	2	2	5	1199577600	LOACK QUADRAT VANIL WAFE
4	155049	B000PAQ75C	AR5J8UI46CURR	Geetha Krishnan	2	2	5	1199577600	LOACK QUADRAT VANIL WAFE

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

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

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

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

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

In [8]:

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

In [9]:

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

Out[9]:

```
(19354, 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]:

Out[10]:

```
96.77
```

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

In [11]:

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

display.head()
```

Out[11]:

	Id	ProductId	UserId	ProfileName	HelpfulnessNumerator	HelpfulnessDenominator	Score	Time	Summary
0	64422	B000MIDROQ	A161DK06JJMCYF	J. E. Stephens "Jeanne"	3	1	5	1224892800	Bought This for My Son at College
1	44737	B001EQ55RW	A2V0I904FH7ABY	Ram	3	2	4	1212883200	Pure cocoa taste with crunchy almonds inside

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()
```

```
(19354, 10)
```

Out[13]:

```
1    16339
0     3015
Name: Score, dtype: int64
```

[3] Preprocessing

[3.1]. Preprocessing Review Text

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

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

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

5. Convert the word to lowercase
6. Remove Stopwords
7. Finally Snowball Stemming the word (it was obsereved to be better than Porter Stemming)

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

In [14]:

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

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

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

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

```
We have used the Victor fly bait for 3 seasons.  Can't beat it.  Great product!
==================================================
I received this box with great anticipation since they don't sell these on the west coast. I got t
he package, opened the box and was EXTREMELY disappointed. The cookies looked like a gorilla shook
the box to death and left most of the box filled with crumbs. AND THERE WAS A RODENT SIZED HOLE ON
THE SIDE OF THE BOX!!!!!!!! So, needless to say I will not NOT be reordering these again.
==================================================
I have two cats. My big boy has eaten these and never had a problem...as a matter of fact he has n
ever vomited or had a hair ball since I adopted him at 2 months. My girl cat throws up every time
she eats this particular flavor. Since I treat them equally these are no longer purchased. I hate
to see my girl sick so I just recommend you watch your cats after you give them these treats. If n
ot a problem...carry on.
==================================================
I was always a fan of Dave's, so I bought this at a local store to try Blair's and I'm glad I did.
The jalepeno sause is very mild (for me) but one of the most delicious condiments I've ever tasted
.  The Afterdeath is a bit painful, but still very tasty on rice & beans, burritos, or any chicken
dish I've tried it on.  The Sudden Death kicked my ass when I underestimated it, but now a few dro
ps in a dish or pot are just right if I want heat without changing flavor much.
==================================================
```

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)
```

```
We have used the Victor fly bait for 3 seasons.  Can't beat it.  Great product!
```

In [16]:

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

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

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

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

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

```
We have used the Victor fly bait for 3 seasons.  Can't beat it.  Great product!
==================================================
I received this box with great anticipation since they don't sell these on the west coast. I got t
he package, opened the box and was EXTREMELY disappointed. The cookies looked like a gorilla shook
the box to death and left most of the box filled with crumbs. AND THERE WAS A RODENT SIZED HOLE ON
THE SIDE OF THE BOX!!!!!!!! So, needless to say I will not NOT be reordering these again.
==================================================
I have two cats. My big boy has eaten these and never had a problem...as a matter of fact he has n
ever vomited or had a hair ball since I adopted him at 2 months. My girl cat throws up every time
she eats this particular flavor. Since I treat them equally these are no longer purchased. I hate
to see my girl sick so I just recommend you watch your cats after you give them these treats. If n
ot a problem...carry on.
==================================================
I was always a fan of Dave's, so I bought this at a local store to try Blair's and I'm glad I did.
The jalepeno sause is very mild (for me) but one of the most delicious condiments I've ever tasted
.  The Afterdeath is a bit painful, but still very tasty on rice & beans, burritos, or any chicken
dish I've tried it on.  The Sudden Death kicked my ass when I underestimated it, but now a few dro
ps in a dish or pot are just right if I want heat without changing flavor much.
```

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)
```

```
I have two cats. My big boy has eaten these and never had a problem...as a matter of fact he has n
ever vomited or had a hair ball since I adopted him at 2 months. My girl cat throws up every time
she eats this particular flavor. Since I treat them equally these are no longer purchased. I hate
to see my girl sick so I just recommend you watch your cats after you give them these treats. If n
ot a problem...carry on.
==================================================
```

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)
```

```
We have used the Victor fly bait for  seasons.  Can't beat it.  Great product!
```

In [20]:

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

```
I have two cats My big boy has eaten these and never had a problem as a matter of fact he has neve
r vomited or had a hair ball since I adopted him at 2 months My girl cat throws up every time she
eats this particular flavor Since I treat them equally these are no longer purchased I hate to see
my girl sick so I just recommend you watch your cats after you give them these treats If not a pro
blem carry on
```

In [21]:

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

stopwords= set(['br', 'the', 'i', 'me', 'my', 'myself', 'we', 'our', 'ours', 'ourselves', 'you', "y
ou're", "you've",\
            "you'll", "you'd", 'your', 'yours', 'yourself', 'yourselves', 'he', 'him', 'his',
'himself', \
            'she', "she's", 'her', 'hers', 'herself', 'it', "it's", 'its', 'itself', 'they', 'them',
'their',\
            'theirs', 'themselves', 'what', 'which', 'who', 'whom', 'this', 'that', "that'll",
'these', 'those', \
            'am', 'is', 'are', 'was', 'were', 'be', 'been', 'being', 'have', 'has', 'had', 'having',
'do', 'does', \
            'did', 'doing', 'a', 'an', 'the', 'and', 'but', 'if', 'or', 'because', 'as', 'until', '
while', 'of', \
            'at', 'by', 'for', 'with', 'about', 'against', 'between', 'into', 'through', 'during',
'before', 'after',\
            'above', 'below', 'to', 'from', 'up', 'down', 'in', 'out', 'on', 'off', 'over', 'under'
, 'again', 'further',\
            'then', 'once', 'here', 'there', 'when', 'where', 'why', 'how', 'all', 'any', 'both', 'e
ach', 'few', 'more',\
            'most', 'other', 'some', 'such', 'only', 'own', 'same', 'so', 'than', 'too', 'very', \
            's', 't', 'can', 'will', 'just', 'don', "don't", 'should', "should've", 'now', 'd', 'll'
, 'm', 'o', 're', \
            've', 'y', 'ain', 'aren', "aren't", 'couldn', "couldn't", 'didn', "didn't", 'doesn', "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 [22]:

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

```
100%|██████████| 19354/19354 [00:12<00:00, 1526.25it/s]
```

In [23]:

```
preprocessed_reviews[1500]
```

Out[23]:

```
'two cats big boy eaten never problem matter fact never vomited hair ball since adopted months gir
```

```
l cat throws every time eats particular flavor since treat equally no longer purchased hate see gi
rl sick recommend watch cats give treats not problem carry'
```

[3.2] Preprocessing Review Summary

In [24]:

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

[4] Featurization

[4.1] BAG OF WORDS

In [25]:

```
#BoW
count_vect = CountVectorizer() #in scikit-learn
count_vect.fit(preprocessed_reviews)
print("some feature names ", count_vect.get_feature_names()[:10])
print('='*50)

final_counts = count_vect.transform(preprocessed_reviews)
print("the type of count vectorizer ",type(final_counts))
print("the shape of out text BOW vectorizer ",final_counts.get_shape())
print("the number of unique words ", final_counts.get_shape()[1])
```

```
some feature names  ['aa', 'aaaa', 'aaaaa', 'aaaand', 'aafco', 'aahhhs', 'aahs', 'ab', 'aback', 'a
bandon']
==================================================
the type of count vectorizer  <class 'scipy.sparse.csr.csr_matrix'>
the shape of out text BOW vectorizer  (19354, 25997)
the number of unique words  25997
```

[4.2] Bi-Grams and n-Grams.

In [26]:

```
#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  (19354, 5000)
the number of unique words including both unigrams and bigrams  5000
```

[4.3] TF-IDF

In [27]:

```
tf_idf_vect = TfidfVectorizer(ngram_range=(1,2), min_df=10)
tf_idf_vect.fit(preprocessed_reviews)
print("some sample features(unique words in the corpus)",tf_idf_vect.get_feature_names()[0:10])
print('='*50)
```

```
final_tf_idf = tf_idf_vect.transform(preprocessed_reviews)
print("the type of count vectorizer ",type(final_tf_idf))
print("the shape of out text TFIDF vectorizer ",final_tf_idf.get_shape())
print("the number of unique words including both unigrams and bigrams ", final_tf_idf.get_shape()[
1])
```

```
some sample features(unique words in the corpus) ['ability', 'able', 'able buy', 'able eat', 'able
find', 'able get', 'able give', 'able make', 'able order', 'able purchase']
==================================================
the type of count vectorizer  <class 'scipy.sparse.csr.csr_matrix'>
the shape of out text TFIDF vectorizer  (19354, 11762)
the number of unique words including both unigrams and bigrams  11762
```

[4.4] Word2Vec

In [28]:

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

In [29]:

```
# Using Google News Word2Vectors

# in this project we are using a pretrained model by google
# its 3.3G file, once you load this into your memory
# it occupies ~9Gb, so please do this step only if you have >12G of ram
# we will provide a pickle file wich contains a dict ,
# and it contains all our courpus words as keys and  model[word] as values
# To use this code-snippet, download "GoogleNews-vectors-negative300.bin"
# from https://drive.google.com/file/d/0B7XkCwpI5KDYNlNUTTlSS21pQmM/edit
# it's 1.9GB in size.


# http://kavita-ganesan.com/gensim-word2vec-tutorial-starter-code/#.W17SRFAzZPY
# you can comment this whole cell
# or change these varible according to your need

is_your_ram_gt_16g=False
want_to_use_google_w2v = False
want_to_train_w2v = True

if want_to_train_w2v:
    # min_count = 5 considers only words that occured atleast 5 times
    w2v_model=Word2Vec(list_of_sentance,min_count=5,size=50, workers=4)
    print(w2v_model.wv.most_similar('great'))
    print('='*50)
    print(w2v_model.wv.most_similar('worst'))

elif want_to_use_google_w2v and is_your_ram_gt_16g:
    if os.path.isfile('GoogleNews-vectors-negative300.bin'):
        w2v_model=KeyedVectors.load_word2vec_format('GoogleNews-vectors-negative300.bin', binary=Tr
ue)
        print(w2v_model.wv.most_similar('great'))
        print(w2v_model.wv.most_similar('worst'))
    else:
        print("you don't have gogole's word2vec file, keep want_to_train_w2v = True, to train your
own w2v ")
```

```
[('awesome', 0.8064208030700684), ('excellent', 0.8037704825401306), ('good', 0.8028221130371094),
('wonderful', 0.7886734008789062), ('fantastic', 0.7870103716850281), ('amazing',
0.7722129821777344), ('perfect', 0.7130043506622314), ('delicious', 0.7130036354064941),
('decent', 0.6866195797920227), ('quick', 0.668171763420105)]
==================================================
[('personal', 0.8243658542633057), ('closest', 0.8083181381225586), ('hooked',
0.7997307777404785), ('britt', 0.7980998158454895), ('jamaica', 0.7949360609054565),
('disappointing', 0.7936862111091614), ('addicted', 0.7923240065574646), ('purely',
0.7896387577056885), ('surpasses', 0.7816001176834106), ('greatest', 0.7772828340530396)]
```

In [30]:

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

```
number of words that occured minimum 5 times  8370
sample words  ['used', 'fly', 'bait', 'seasons', 'ca', 'not', 'beat', 'great', 'product',
'available', 'traps', 'course', 'total', 'pretty', 'stinky', 'right', 'nearby', 'really', 'good',
'idea', 'final', 'outstanding', 'use', 'car', 'window', 'everybody', 'asks', 'bought', 'made', 'tw
o', 'thumbs', 'received', 'shipment', 'could', 'hardly', 'wait', 'try', 'love', 'call', 'instead',
'stickers', 'removed', 'easily', 'daughter', 'designed', 'signs', 'printed', 'reverse', 'windows',
'beautifully']
```

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

[4.4.1.1] Avg W2v

In [31]:

```
# 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 this
to 300 if you use google's w2v
    cnt_words =0; # num of words with a valid vector in the sentence/review
    for word in sent: # for each word in a review/sentence
        if word in w2v_words:
            vec = w2v_model.wv[word]
            sent_vec += vec
            cnt_words += 1
    if cnt_words != 0:
        sent_vec /= cnt_words
    sent_vectors.append(sent_vec)
print(len(sent_vectors))
print(len(sent_vectors[0]))
```

```
100%|██████████| 19354/19354 [00:42<00:00, 459.00it/s]
```

```
19354
50
```

[4.4.1.2] TFIDF weighted W2v

In [32]:

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

In [33]:

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

tfidf_sent_vectors = []; # the tfidf-w2v for each sentence/review is stored in this list
row=0;
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)
    row += 1
```

```
100%|██████████| 19354/19354 [06:04<00:00, 53.07it/s]
```

[5] Assignment 3: KNN

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

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

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

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

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

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

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

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

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

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

4. **Representation of results**

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

5. **Conclusion**

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

Note: Data Leakage

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

[5.1] Applying KNN brute force

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

In [34]:

```
#hiding all the warnings
warnings.filterwarnings('ignore')

# Breaking the data into train and test
from sklearn.model_selection import train_test_split

X = preprocessed_reviews
y = final['Score']
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.3, random_state=42)

print('shape of X:',len(X),'shape of y:',len(y))
```

```
shape of X: 19354 shape of y: 19354
```

In [35]:

```
#BoW

count_vect = CountVectorizer() #in scikit-learn
count_vect.fit(X_train)
print("some feature names ", count_vect.get_feature_names()[:10])
print('='*50)

final_counts = count_vect.transform(X_train)
print("the type of count vectorizer ",type(final_counts))
print("the shape of out text BOW vectorizer ",final_counts.get_shape())
print("the number of unique words ", final_counts.get_shape()[1])

final_test=count_vect.transform(X_test)
```

```
some feature names  ['aa', 'aaaa', 'aaaaa', 'aafco', 'aback', 'abandon', 'abandoned',
'abandoning', 'abates', 'abbott']
==================================================
the type of count vectorizer  <class 'scipy.sparse.csr.csr_matrix'>
the shape of out text BOW vectorizer  (13547, 22164)
the number of unique words  22164
```

In [36]:

```
# Shape of train and test data
print("X_train shape: {0}   y_train shape: {1}".format(len(X_train), len(y_train)))
print("X_test shape:  {0}    y_test shape: {1}".format(len(X_test), len(y_test)))

train_per=len(X_train)/len(X)*100
test_per=len(X_test)/len(X)*100
import math
print("\nPercentage of data in Train : {0:.1f} % and Test :{1:.1f} % ".format(train_per,test_per))
```

```
X_train shape: 13547   y_train shape: 13547
X_test shape:  5807    y_test shape: 5807
```

```
Percentage of data in Train : 70.0 % and Test :30.0 %
```

In [37]:

```
X_train=final_counts
X_test=final_test
```

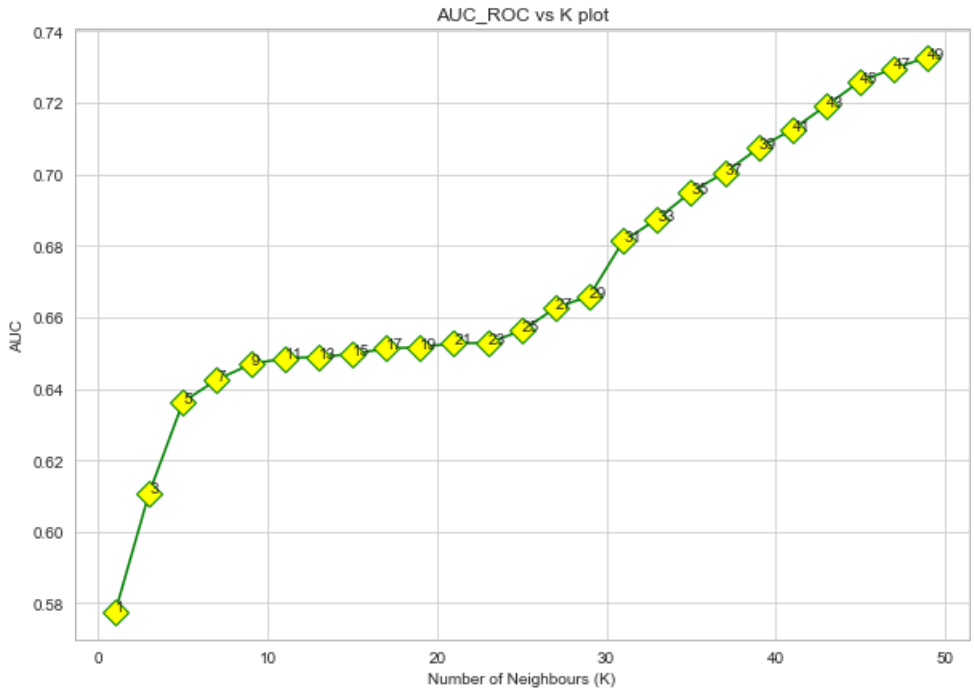
K fold cross validation

In [39]:

```
# applying knn and finding optimal k
from sklearn.neighbors import KNeighborsClassifier
from sklearn.model_selection import cross_val_score
cv_list=[]
neighbors=range(1,50,2)
for K in neighbors:
    K_value = K
    neigh = KNeighborsClassifier(n_neighbors = K_value, weights='uniform', algorithm='brute')
    scores=cross_val_score(neigh, X_train, y_train, cv=5, scoring='roc_auc') # 5 fold cross validat
ion
    cv_list.append(scores.mean())
```

In [40]:

```
# plot of k vs auc_roc
plt.figure(figsize = (10,7))

sns.set_style("whitegrid");

plt.plot(neighbors, cv_list, color='green',  marker='D',markerfacecolor='yellow', markersize=12)
ind=0
for val in neighbors:
    plt.annotate('%s' % val,xy=(val,cv_list[ind]),xycoords='data')
    ind=ind+1

plt.title("AUC_ROC vs K plot")
plt.xlabel('Number of Neighbours (K)')
plt.ylabel('AUC')
plt.show()
```



In [41]:

```
# optimal k
opt k=31
```

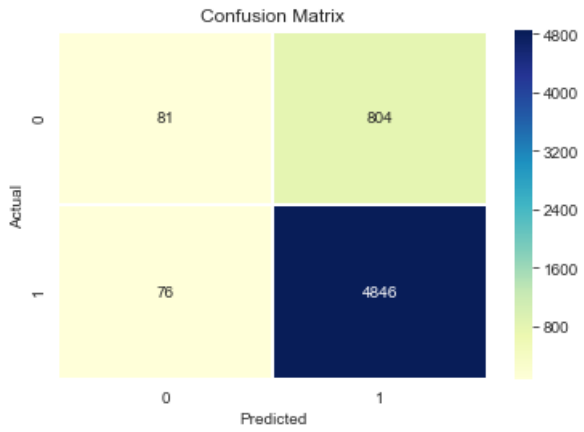
```
val=math.floor(opt_k/2)+1
print("Maximum AUC score is: ",cv_list[val],"for k=",opt_k)
```

```
Maximum AUC score is:  0.6874412358621292 for k= 31
```

Confusion Matrix

In [42]:

```
import seaborn as sb
from sklearn.metrics import confusion_matrix

knn_optimal = KNeighborsClassifier(n_neighbors=opt_k , algorithm = 'brute')
knn_optimal.fit(X_train, y_train)

conf_matrix = confusion_matrix(y_test,knn_optimal.predict(X_test))
df_cmatrix = pd.DataFrame(conf_matrix) #, columns=['Negative', 'Positive'])
sb.heatmap(df_cmatrix, annot=True, fmt='d',cmap="YlGnBu", linewidths=1.4)

plt.title("Confusion Matrix")
plt.xlabel("Predicted")
plt.ylabel("Actual")
plt.show()
```



AUC CURVE

In [44]:

```
# calculating train_auc
from sklearn.metrics import roc_auc_score

neighbors=list(range(1,50,2))

train_auc_list=[]
for k in neighbors:
    neigh = KNeighborsClassifier(n_neighbors=k , algorithm = 'brute')
    neigh.fit(X_train, y_train)
    # predict probabilities
    probs = neigh.predict_proba(X_train)
    # keep probabilities for the positive outcome only
    probs = probs[:, 1]
    # calculate AUC
    train_auc = roc_auc_score(y_train, probs)
    train_auc_list.append(train_auc)
```

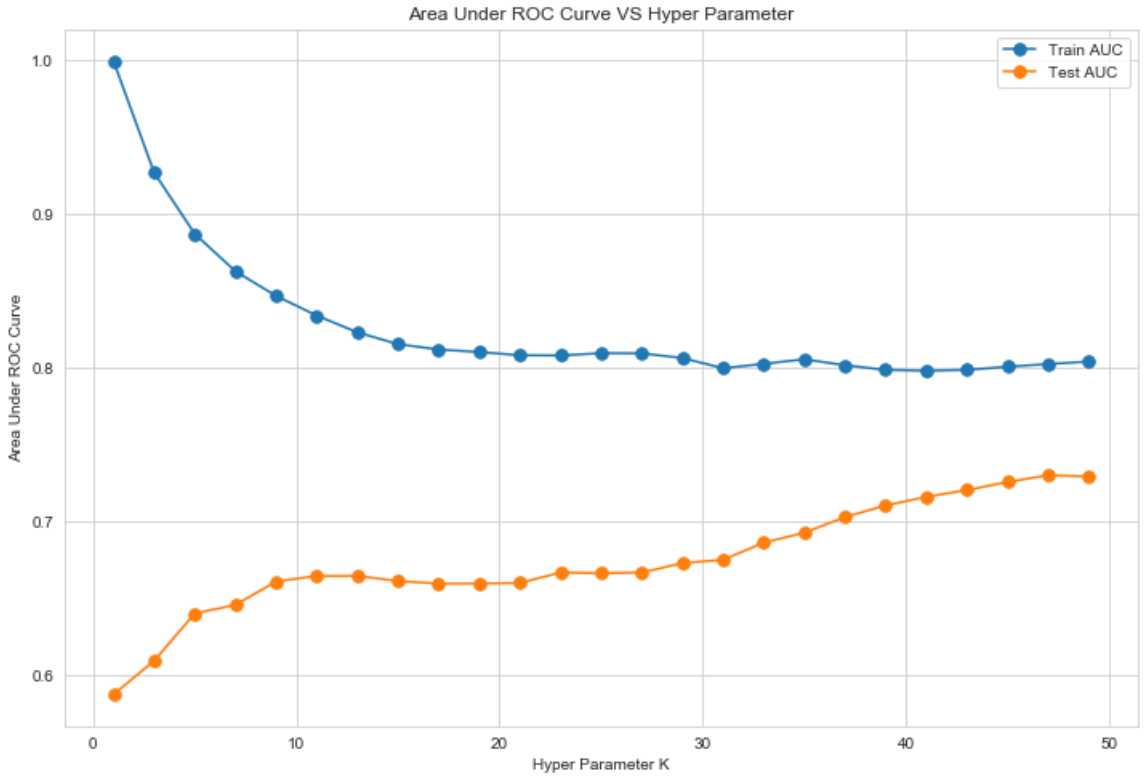
In [45]:

```
# calculating test_auc
neighbors=list(range(1,50,2))

test_auc_list=[]
for k in neighbors:
    neigh = KNeighborsClassifier(n_neighbors=k , algorithm = 'brute')
```

```
    neigh.fit(X_train, y_train)
    # predict probabilities
    probs = neigh.predict_proba(X_test)
    # keep probabilities for the positive outcome only
    probs = probs[:, 1]
    # calculate AUC
    test_auc = roc_auc_score(y_test, probs)
    test_auc_list.append(test_auc)
```

In [46]:

```
plt.figure(figsize = (12,8))
plt.plot(neighbors, train_auc_list, marker='.',label="Train AUC",markersize=15)
plt.plot(neighbors, test_auc_list, marker='.',label="Test AUC",markersize=15)
plt.legend()
plt.xlabel("Hyper Parameter K")
plt.ylabel("Area Under ROC Curve")
plt.title("Area Under ROC Curve VS Hyper Parameter")
plt.show()
```



In [47]:

```
knn_optimal = KNeighborsClassifier(n_neighbors=opt_k , algorithm = 'brute')
knn_optimal.fit(X_train, y_train)
```

Out[47]:

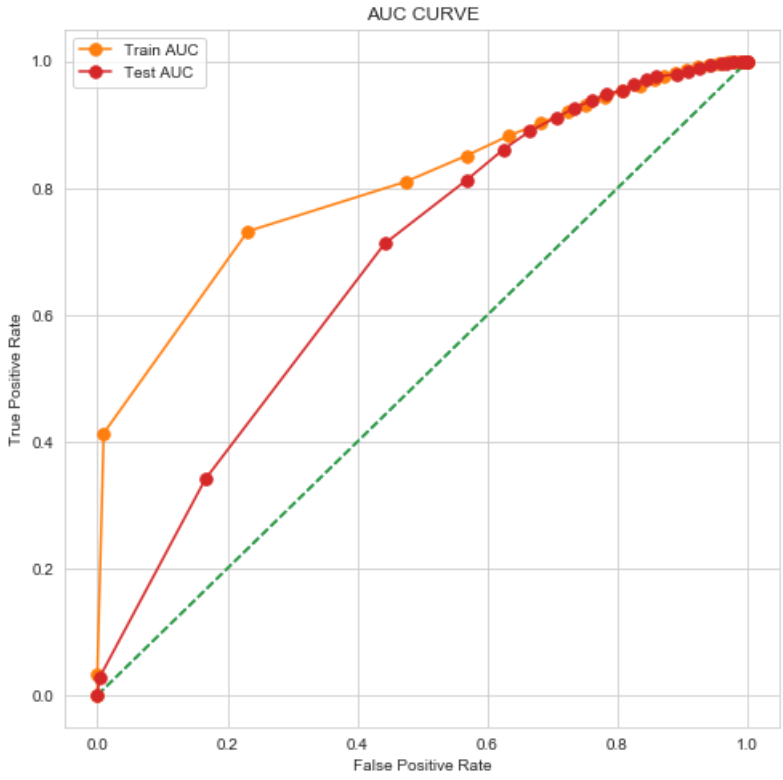
```
KNeighborsClassifier(algorithm='brute', leaf_size=30, metric='minkowski',
           metric_params=None, n_jobs=None, n_neighbors=31, p=2,
           weights='uniform')
```

In [48]:

```
from matplotlib import pyplot
from sklearn.metrics import roc_auc_score
from sklearn.metrics import roc_curve, auc

plt.figure(figsize = (8,8))
# predict probabilities
probs = knn_optimal.predict_proba(X_train)
# keep probabilities for the positive outcome only
probs = probs[:, 1]
# calculate AUC
train_auc = roc_auc_score(y_train, probs)
```

```
print('Train AUC: %f' % train_auc)
# calculate roc curve
fpr, tpr, thresholds = roc_curve(y_train, probs)
# plot no skill
pyplot.plot([0, 1], [0, 1], linestyle='--')
# plot the roc curve for the model
pyplot.plot(fpr, tpr, marker='.',label="Train AUC",markersize=15)


# predict probabilities
probs = knn_optimal.predict_proba(X_test)
# keep probabilities for the positive outcome only
probs = probs[:, 1]
# calculate AUC
test_auc = roc_auc_score(y_test, probs)
print('Test AUC: %f' % test_auc)
# calculate roc curve
fpr, tpr, thresholds = roc_curve(y_test, probs)
# plot no skill
pyplot.plot([0, 1], [0, 1], linestyle='--')
# plot the roc curve for the model
pyplot.plot(fpr, tpr, marker='.',label="Test AUC",markersize=15)

plt.legend()
plt.xlabel("False Positive Rate")
plt.ylabel("True Positive Rate")
plt.title("AUC CURVE")

# show the plot
pyplot.show()
```

```
Train AUC: 0.799941
Test AUC: 0.675245
```



Conclusion

In [49]:

```
bow_brute_par=opt_k
bow_brute_auc=np.round(test_auc,4)
print("Vectorizer: BOW \t Model: Brute")
print("Best Hyper parameter: ",bow_brute_par)
print("AUC: ",bow_brute_auc)
```

```
Vectorizer: BOW   Model: Brute
Best Hyper parameter:  31
AUC:  0.6752
```

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

In [50]:

```
#hiding all the warnings
warnings.filterwarnings('ignore')

# Breaking the data into train and test
from sklearn.model_selection import train_test_split

X = preprocessed_reviews
y = final['Score']
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.3, random_state=42)

print('shape of X:',len(X),'shape of y:',len(y))
```

```
shape of X: 19354 shape of y: 19354
```

In [51]:

```
tf_idf_vect = TfidfVectorizer(ngram_range=(1,2), min_df=5)
tf_idf_vect.fit(X_train)
X_train_tfidf= tf_idf_vect.transform(X_train)

X_test_tfidf=tf_idf_vect.transform(X_test)
```

In [52]:

```
# Shape of train and test data
print("X_train shape: {0}   y_train shape: {1}".format(len(X_train), len(y_train)))
print("X_test shape:  {0}    y_test shape: {1}".format(len(X_test), len(y_test)))

train_per=len(X_train)/len(X)*100
test_per=len(X_test)/len(X)*100
import math
print("\nPercentage of data in Train : {0:.1f} % and Test :{1:.1f} % ".format(train_per,test_per))
```

```
X_train shape: 13547   y_train shape: 13547
X_test shape:  5807    y_test shape: 5807

Percentage of data in Train : 70.0 % and Test :30.0 %
```

K fold cross validation

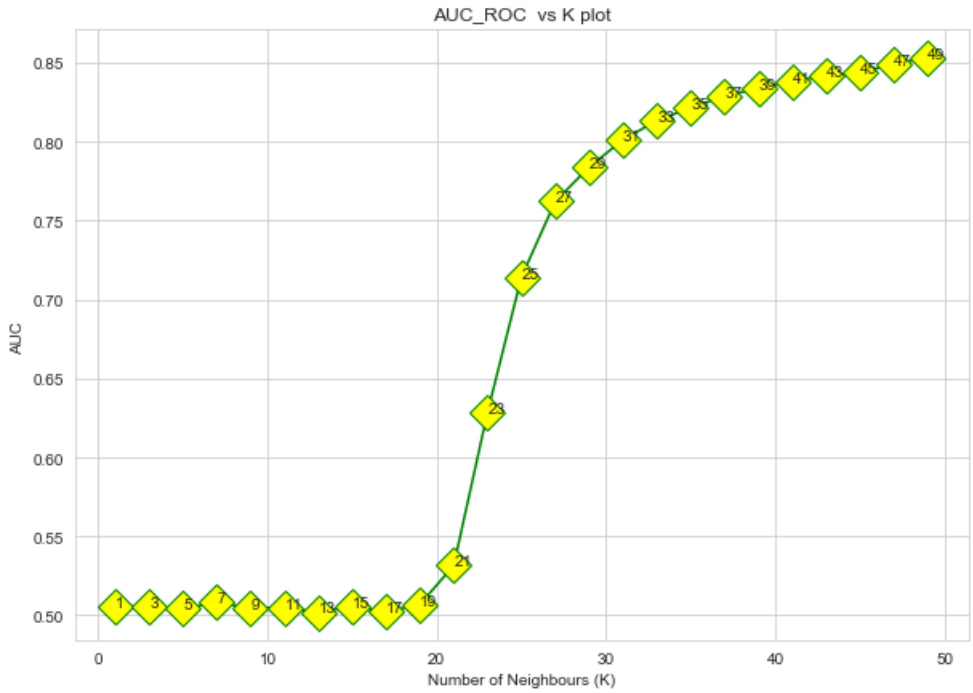
In [53]:

```
# applying knn and finding optimal k

from sklearn.model_selection import cross_val_score
cv_list=[]
neighbors=range(1,50,2)
for K in neighbors:
    K_value = K
    neigh = KNeighborsClassifier(n_neighbors = K_value, weights='uniform', algorithm='brute')
    scores=cross_val_score(neigh, X_train_tfidf, y_train, cv=5, scoring='roc_auc') # 5 fold cross v
alidation
    cv_list.append(scores.mean())
```

In [54]:

```
# plot of k vs roc_auc
plt.figure(figsize = (10,7))

sns.set_style("whitegrid");
```

```
sns.set_style( whitegrid );

plt.plot(neighbors, cv_list, color='green',  marker='D',markerfacecolor='yellow', markersize=15)
ind=0
for val in neighbors:
    plt.annotate('%s' % val,xy=(val,cv_list[ind]),xycoords='data')
    ind=ind+1

plt.title("AUC_ROC  vs K plot")
plt.xlabel('Number of Neighbours (K)')
plt.ylabel('AUC ')
plt.show()
```



In [55]:

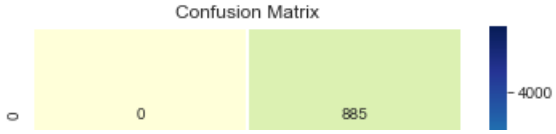
```
# optimal k
opt_k=29
val=math.floor(opt_k/2)+1
print("Maximum AUC score is: ",cv_list[val],"for k=",opt_k)
```

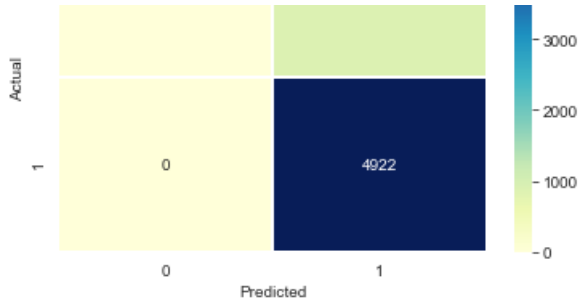
```
Maximum AUC score is:  0.8011585038338367 for k= 29
```

Confusion Matrix

In [56]:

```
import seaborn as sb
from sklearn.metrics import confusion_matrix

knn_optimal = KNeighborsClassifier(n_neighbors=opt_k , algorithm = 'brute')
knn_optimal.fit(X_train_tfidf, y_train)

conf_matrix = confusion_matrix(y_test,knn_optimal.predict(X_test_tfidf))
df_cmatrix = pd.DataFrame(conf_matrix) #, columns=['Negative', 'Positive'])
sb.heatmap(df_cmatrix, annot=True, fmt='d',cmap="YlGnBu", linewidths=1.4)

plt.title("Confusion Matrix")
plt.xlabel("Predicted")
plt.ylabel("Actual")
plt.show()
```





AUC CURVE

In [57]:

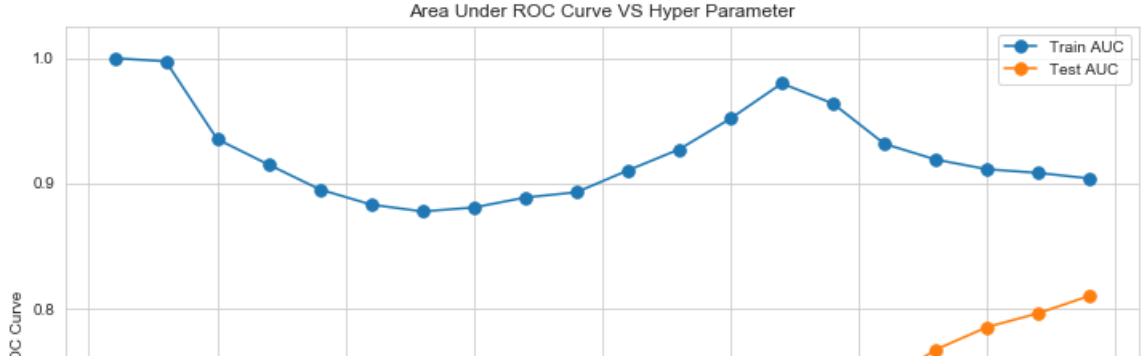
```
# calculating train_auc
neighbors=list(range(1,40,2))

train_auc_list=[]
for k in neighbors:
    neigh = KNeighborsClassifier(n_neighbors=k , algorithm = 'brute')
    neigh.fit(X_train_tfidf, y_train)
    # predict probabilities
    probs = neigh.predict_proba(X_train_tfidf)
    # keep probabilities for the positive outcome only
    probs = probs[:, 1]
    # calculate AUC
    train_auc = roc_auc_score(y_train, probs)
    train_auc_list.append(train_auc)
```

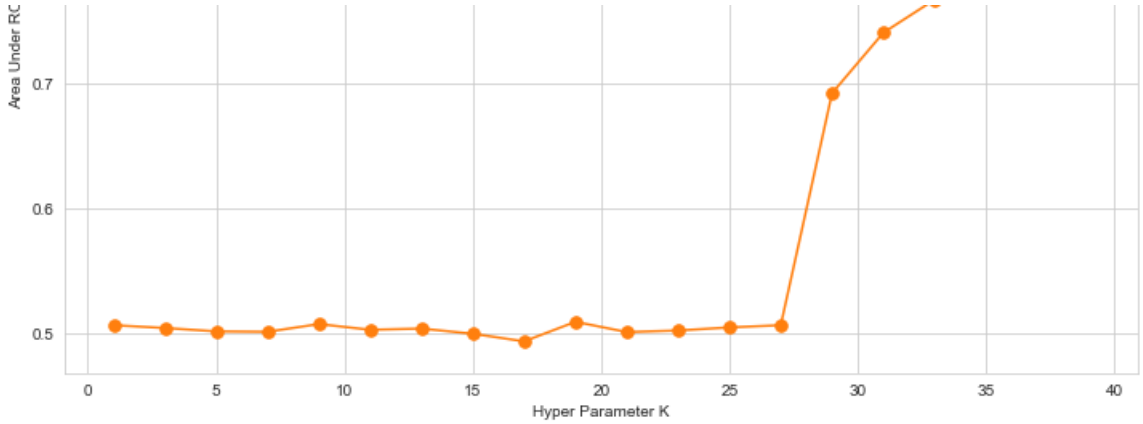
In [58]:

```
# calculating test_auc
neighbors=list(range(1,40,2))

test_auc_list=[]
for k in neighbors:
    neigh = KNeighborsClassifier(n_neighbors=k , algorithm = 'brute')
    neigh.fit(X_train_tfidf, y_train)
    # predict probabilities
    probs = neigh.predict_proba(X_test_tfidf)
    # keep probabilities for the positive outcome only
    probs = probs[:, 1]
    # calculate AUC
    test_auc = roc_auc_score(y_test, probs)
    test_auc_list.append(test_auc)
```

In [59]:

```
plt.figure(figsize = (12,8))
plt.plot(neighbors, train_auc_list, marker='.',label="Train AUC",markersize=15)
plt.plot(neighbors, test_auc_list, marker='.',label="Test AUC",markersize=15)
plt.legend()
plt.xlabel("Hyper Parameter K")
plt.ylabel("Area Under ROC Curve")
plt.title("Area Under ROC Curve VS Hyper Parameter")
plt.show()
```





In [60]:

```
knn_optimal = KNeighborsClassifier(n_neighbors=opt_k , algorithm = 'brute')
knn_optimal.fit(X_train_tfidf, y_train)
```

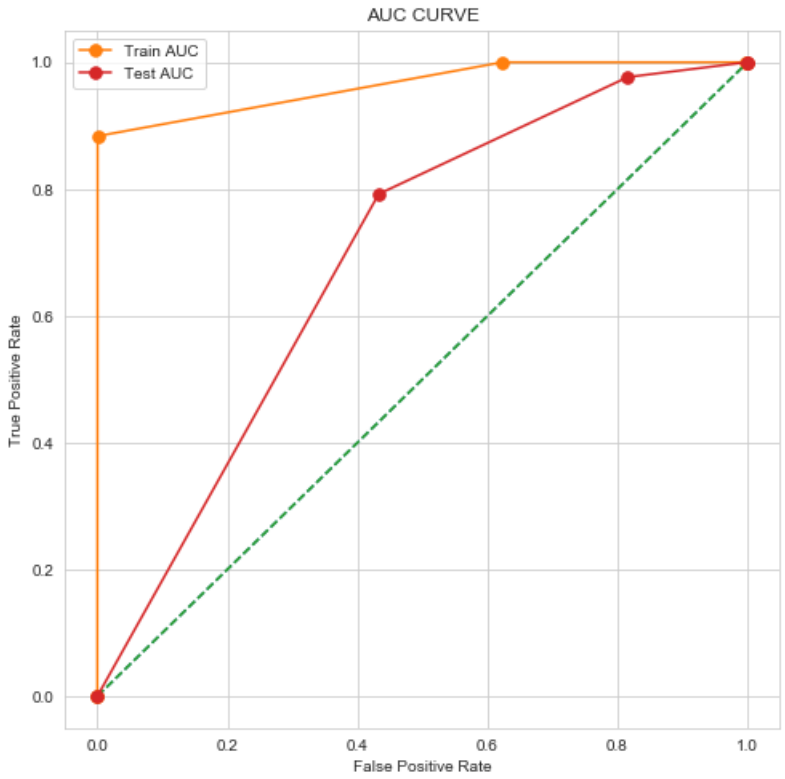
Out[60]:

```
KNeighborsClassifier(algorithm='brute', leaf_size=30, metric='minkowski',
           metric_params=None, n_jobs=None, n_neighbors=29, p=2,
           weights='uniform')
```

In [61]:

```
from matplotlib import pyplot
from sklearn.metrics import roc_auc_score
from sklearn.metrics import roc_curve, auc

plt.figure(figsize = (8,8))
# predict probabilities
probs = knn_optimal.predict_proba(X_train_tfidf)
# keep probabilities for the positive outcome only
probs = probs[:, 1]
# calculate AUC
train_auc = roc_auc_score(y_train, probs)
print('Train AUC: %f' % train_auc)
# calculate roc curve
fpr, tpr, thresholds = roc_curve(y_train, probs)
# plot no skill
pyplot.plot([0, 1], [0, 1], linestyle='--')
# plot the roc curve for the model
pyplot.plot(fpr, tpr, marker='.',label="Train AUC",markersize=15)


# predict probabilities
probs = knn_optimal.predict_proba(X_test_tfidf)
# keep probabilities for the positive outcome only
probs = probs[:, 1]
# calculate AUC
test_auc = roc_auc_score(y_test, probs)
print('Test AUC: %f' % test_auc)
# calculate roc curve
fpr, tpr, thresholds = roc_curve(y_test, probs)
# plot no skill
pyplot.plot([0, 1], [0, 1], linestyle='--')
# plot the roc curve for the model
pyplot.plot(fpr, tpr, marker='.',label="Test AUC",markersize=15)

plt.legend()
plt.xlabel("False Positive Rate")
plt.ylabel("True Positive Rate")
plt.title("AUC CURVE")

# show the plot
pyplot.show()
```

```
Train AUC: 0.963478
Test AUC: 0.692222
```



Conclusion

In [62]:

```
tfidf_brute_par=opt_k
tfidf_brute_auc=np.round(test_auc,4)
print("Vectorizer: TFIDF \t Model: Brute")
print("Best Hyper parameter: ",tfidf_brute_par)
print("AUC: ",tfidf_brute_auc)
```

```
Vectorizer: TFIDF   Model: Brute
Best Hyper parameter:  29
AUC:  0.6922
```

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

In [63]:

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

In [64]:

```
#hiding all the warnings
warnings.filterwarnings('ignore')

# Breaking the data into train and test
from sklearn.model_selection import train_test_split

X = preprocessed_reviews
y = final['Score']
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.3, random_state=42)

print('shape of X:',len(X),'shape of y:',len(y))
```

```
shape of X: 19354 shape of y: 19354
```

In [65]:

```
# Train your own Word2Vec model using your own text corpus
```

```
i=0
list_of_sentance=[]
for sentance in X_train:
    list_of_sentance.append(sentance.split())
```

In [66]:

```
is_your_ram_gt_16g=False
want_to_use_google_w2v = False
want_to_train_w2v = True

if want_to_train_w2v:
    # min_count = 5 considers only words that occured atleast 5 times
    w2v_model=Word2Vec(list_of_sentance,min_count=5,size=50, workers=4)
    print(w2v_model.wv.most_similar('great'))
    print('='*50)
    print(w2v_model.wv.most_similar('worst'))

elif want_to_use_google_w2v and is_your_ram_gt_16g:
    if os.path.isfile('GoogleNews-vectors-negative300.bin'):
        w2v_model=KeyedVectors.load_word2vec_format('GoogleNews-vectors-negative300.bin', binary=Tr
ue)
        print(w2v_model.wv.most_similar('great'))
        print(w2v_model.wv.most_similar('worst'))
    else:
        print("you don't have gogole's word2vec file, keep want_to_train_w2v = True, to train your
own w2v ")
```

```
[('good', 0.8283370733261108), ('excellent', 0.8188117146492004), ('fantastic',
0.7660717368125916), ('awesome', 0.7503769397735596), ('amazing', 0.7432704567909241), ('quick', 0
.7414237856864929), ('wonderful', 0.7364493608474731), ('super', 0.715154767036438), ('love',
0.6900370121002197), ('decent', 0.6776210069656372)]
==================================================
[('insisted', 0.9500042796134949), ('amongst', 0.949163019657135), ('reordered',
0.9451670050621033), ('owned', 0.9373623132705688), ('occur', 0.926796555519104), ('personal', 0.9
237515926361084), ('europe', 0.9225219488143921), ('closest', 0.9200869798660278), ('among',
0.9194868803024292), ('avid', 0.919314980506897)]
```

In [67]:

```
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])
print(len(w2v_words))
# length=len(w2v_words)
```

```
number of words that occured minimum 5 times  7000
sample words  ['good', 'variety', 'chocolate', 'bad', 'everyone', 'says', 'flavors', 'great',
'gives', 'family', 'friends', 'latte', 'steamer', 'favorite', 'hazelnut', 'stick', 'time', 'buy',
'horrible', 'step', 'pop', 'change', 'formula', 'use', 'cheap', 'barley', 'malt', 'sugar', 'high',
'glycemic', 'place', 'honey', 'amount', 'herbs', 'effective', 'original', 'huge',
'disappointment', 'shame', 'outstanding', 'products', 'reviewer', 'recommended', 'go', 'way', 'ste
er', 'clear', 'recommending', 'anything', 'prince']
7000
```

In [68]:

```
# 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 this
to 300 if you use google's w2v
    cnt_words =0; # num of words with a valid vector in the sentence/review
    for word in sent: # for each word in a review/sentence
        if word in w2v_words:
            vec = w2v_model.wv[word]
            sent_vec += vec
            cnt_words += 1
    if cnt_words != 0:
        sent_vec /= cnt_words
    sent_vectors.append(sent_vec)
print(len(sent_vectors))
```

```
print(len(sent_vectors))
print(len(sent_vectors[0]))
sent_vectors=np.array(sent_vectors)
```

```
100%|██████████| 13547/13547 [00:23<00:00, 577.62it/s]
```

```
13547
50
```

In [69]:

```
sent_vec_test=[]
sentance_test=[]
for sentance in X_test:
    sentance_test.append(sentance.split())


for sent in tqdm(sentance_test): # for each review/sentence
    sent_vec = np.zeros(50) # as word vectors are of zero length 50, you might need to change this to 300 if you use google's w2v
    cnt_words =0; # num of words with a valid vector in the sentence/review
    for word in sent: # for each word in a review/sentence
        if word in w2v_words:
            vec = w2v_model.wv[word]
            sent_vec += vec
            cnt_words += 1
    if cnt_words != 0:
        sent_vec /= cnt_words
    sent_vec_test.append(sent_vec)
print(len(sent_vec_test))
print(len(sent_vec_test[0]))
sent_vec_test=np.array(sent_vec_test)
```

```
100%|██████████| 5807/5807 [00:11<00:00, 506.63it/s]
```

```
5807
50
```

In [70]:

```
# Shape of train and test data
print("X_train shape: {0}   y_train shape: {1}".format(len(X_train), len(y_train)))
print("X_test shape:  {0}    y_test shape: {1}".format(len(X_test), len(y_test)))

train_per=len(X_train)/len(X)*100
test_per=len(X_test)/len(X)*100
import math
print("\nPercentage of data in Train : {0:.1f} % and Test :{1:.1f} % ".format(train_per,test_per))
```

```
X_train shape: 13547   y_train shape: 13547
X_test shape:  5807    y_test shape: 5807

Percentage of data in Train : 70.0 % and Test :30.0 %
```

In [71]:

```
X_train=sent_vectors
X_test=sent_vec_test
```

K fold cross validation

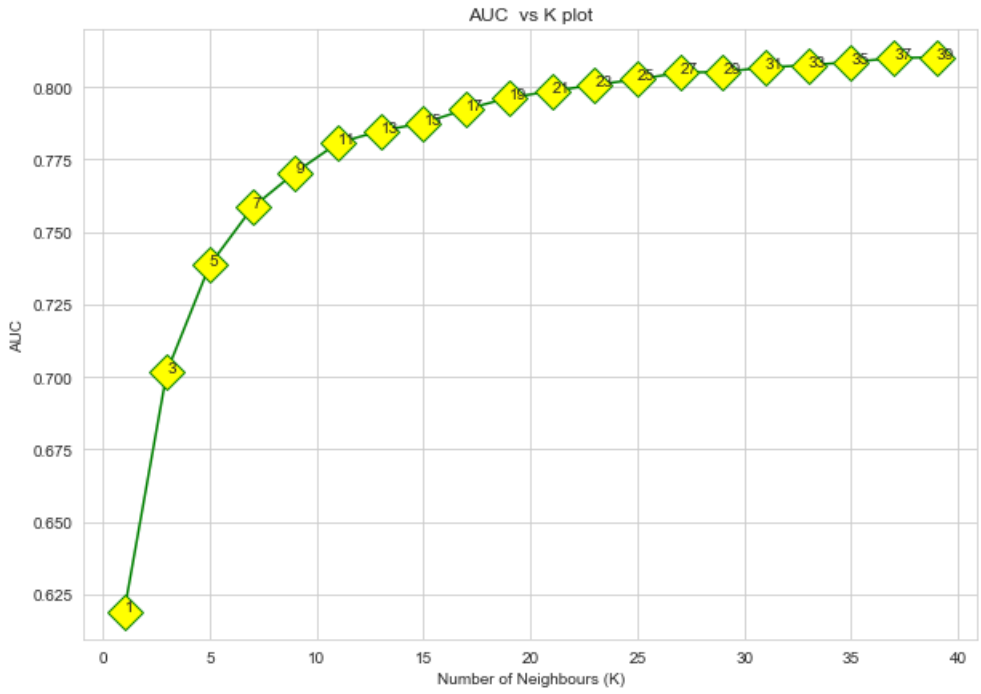
In [73]:

```
# applying knn and finding optimal k

from sklearn.model_selection import cross_val_score
cv_list=[]
neighbors=range(1,40,2)
```

```
for K in neighbors:
    K_value = K
    neigh = KNeighborsClassifier(n_neighbors = K_value, weights='uniform', algorithm='brute')
    scores=cross_val_score(neigh, X_train, y_train, cv=5, scoring='roc_auc') # 5 fold cross validat
ion
    cv_list.append(scores.mean())
```

In [74]:

```
# plot of k vs accuracy
plt.figure(figsize = (10,7))

sns.set_style("whitegrid");

plt.plot(neighbors, cv_list, color='green',  marker='D',markerfacecolor='yellow', markersize=15)
ind=0
for val in neighbors:
    plt.annotate('%s' % val,xy=(val,cv_list[ind]),xycoords='data')
    ind=ind+1

plt.title("AUC  vs K plot")
plt.xlabel('Number of Neighbours (K)')
plt.ylabel('AUC')
plt.show()
```



In [75]:

```
# optimal k
opt_k=15
val=math.floor(opt_k/2)+1
print("Maximum AUC score is: ",cv_list[val],"for k=",opt_k)
```

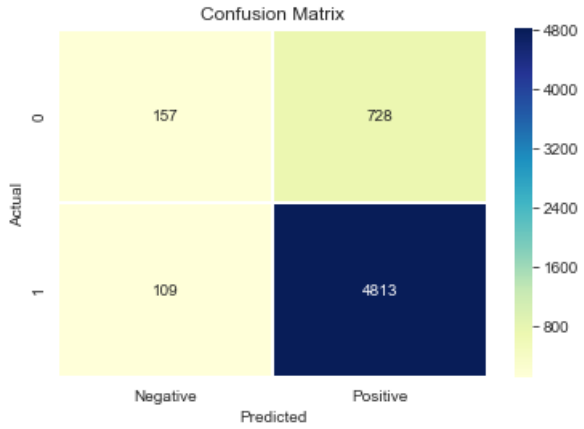
```
Maximum AUC score is:  0.7923499716883969 for k= 15
```

Confusion Matrix

In [76]:

```
import seaborn as sb
from sklearn.metrics import confusion_matrix

knn_optimal = KNeighborsClassifier(n_neighbors=opt_k , algorithm = 'brute')
knn_optimal.fit(X_train, y_train)

conf_matrix = confusion_matrix(y_test,knn_optimal.predict(X_test))
```

```
df_cmatrix = pd.DataFrame(conf_matrix, columns=['Negative', 'Positive'])
sb.heatmap(df_cmatrix, annot=True, fmt='d',cmap="YlGnBu", linewidths=1.4)

plt.title("Confusion Matrix")
plt.xlabel("Predicted")
plt.ylabel("Actual")
plt.show()
```



AUC CURVE

In [77]:

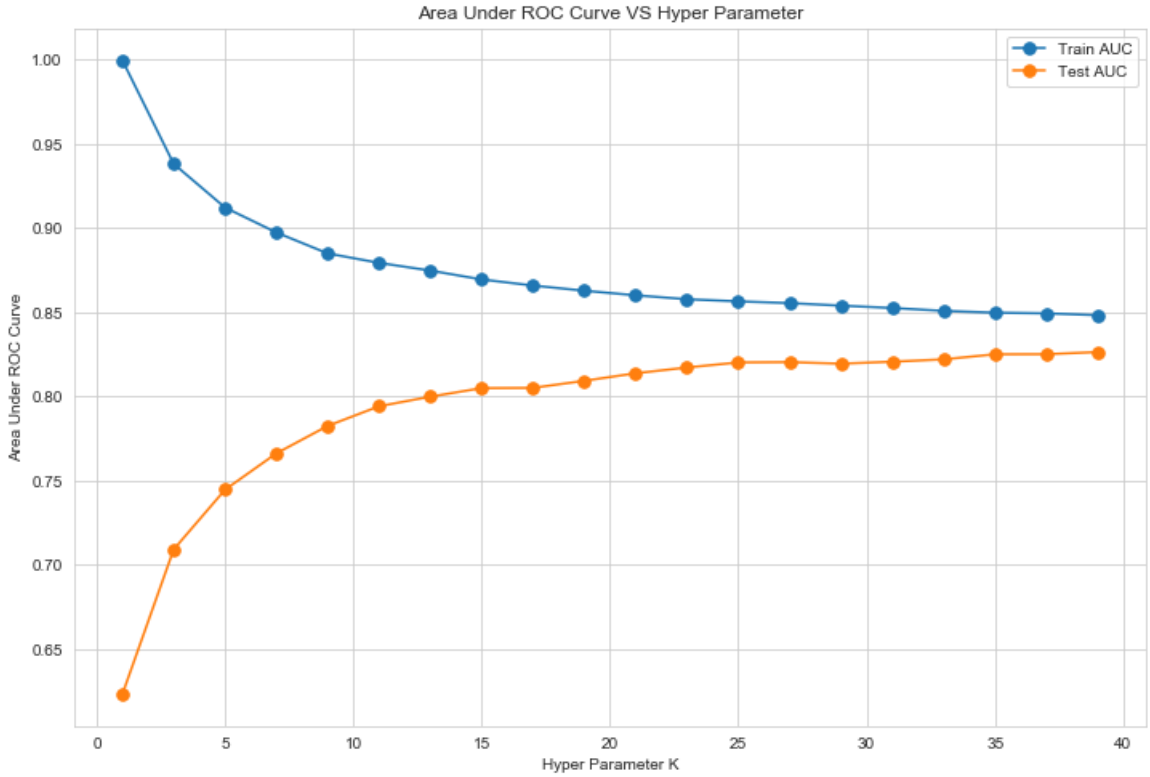
```
# calculating train_auc
neighbors=list(range(1,40,2))

train_auc_list=[]
for k in neighbors:
    neigh = KNeighborsClassifier(n_neighbors=k , algorithm = 'brute')
    neigh.fit(X_train, y_train)
    # predict probabilities
    probs = neigh.predict_proba(X_train)
    # keep probabilities for the positive outcome only
    probs = probs[:, 1]
    # calculate AUC
    train_auc = roc_auc_score(y_train, probs)
    train_auc_list.append(train_auc)
```

In [78]:

```
# calculating test_auc
neighbors=list(range(1,40,2))

test_auc_list=[]
for k in neighbors:
    neigh = KNeighborsClassifier(n_neighbors=k , algorithm = 'brute')
    neigh.fit(X_train, y_train)
    # predict probabilities
    probs = neigh.predict_proba(X_test)
    # keep probabilities for the positive outcome only
    probs = probs[:, 1]
    # calculate AUC
    test_auc = roc_auc_score(y_test, probs)
    test_auc_list.append(test_auc)
```

In [79]:

```
plt.figure(figsize = (12,8))
plt.plot(neighbors, train_auc_list, marker='.',label="Train AUC",markersize=15)
plt.plot(neighbors, test_auc_list, marker='.',label="Test AUC",markersize=15)
plt.legend()
plt.xlabel("Hyper Parameter K")
plt.ylabel("Area Under ROC Curve")
plt.title("Area Under ROC Curve VS Hyper Parameter")
plt.show()
```



In [80]:

```
knn_optimal = KNeighborsClassifier(n_neighbors=opt_k , algorithm = 'brute')
knn_optimal.fit(X_train, y_train)
```

Out[80]:

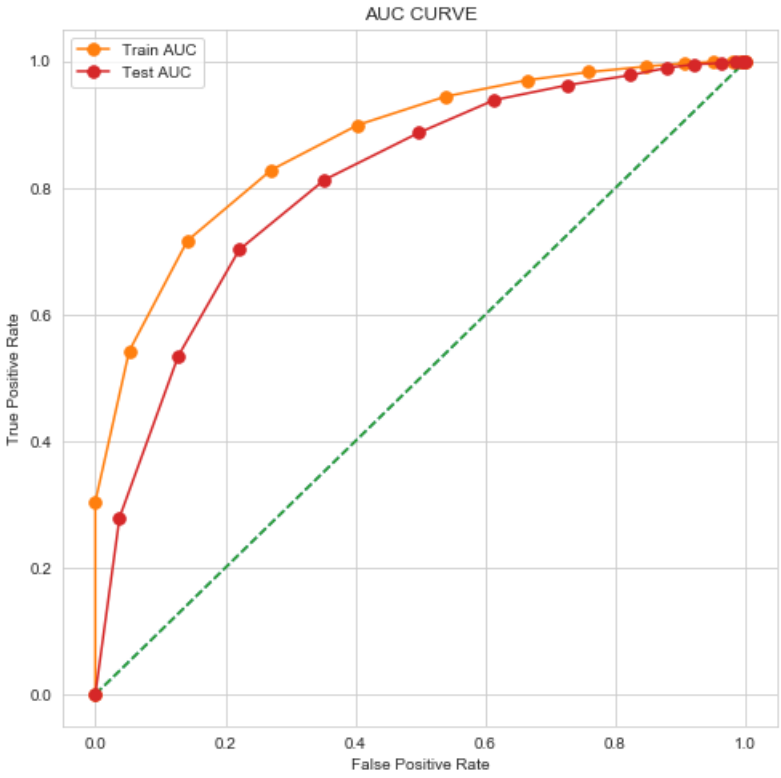
```
KNeighborsClassifier(algorithm='brute', leaf_size=30, metric='minkowski',
           metric_params=None, n_jobs=None, n_neighbors=15, p=2,
           weights='uniform')
```

In [81]:

```
from matplotlib import pyplot
from sklearn.metrics import roc_auc_score
from sklearn.metrics import roc_curve, auc

plt.figure(figsize = (8,8))
# predict probabilities
probs = knn_optimal.predict_proba(X_train)
# keep probabilities for the positive outcome only
probs = probs[:, 1]
# calculate AUC
train_auc = roc_auc_score(y_train, probs)
print('Train AUC: %f' % train_auc)
# calculate roc curve
fpr, tpr, thresholds = roc_curve(y_train, probs)
# plot no skill
pyplot.plot([0, 1], [0, 1], linestyle='--')
# plot the roc curve for the model
pyplot.plot(fpr, tpr, marker='.',label="Train AUC",markersize=15)


# predict probabilities
probs = knn_optimal.predict_proba(X_test)
# keep probabilities for the positive outcome only
probs = probs[:, 1]
# calculate AUC
test_auc = roc_auc_score(y_test, probs)
print('Test AUC: %f' % test_auc)
# calculate roc curve
fpr, tpr, thresholds = roc_curve(y_test, probs)
# plot no skill
pyplot.plot([0, 1], [0, 1], linestyle='--')
# plot the roc curve for the model
```

```
pyplot.plot(fpr, tpr, marker='.',label="Test AUC",markersize=15)

plt.legend()
plt.xlabel("False Positive Rate")
plt.ylabel("True Positive Rate")
plt.title("AUC CURVE")

# show the plot
pyplot.show()
```

```
Train AUC: 0.869517
Test AUC: 0.804948
```



Conclusion

In [82]:

```
w2vec_brute_par=opt_k
w2vec_brute_auc=np.round(test_auc,4)
print("Vectorizer: W2Vec \t Model: Brute")
print("Best Hyper parameter: ",w2vec_brute_par)
print("AUC: ",w2vec_brute_auc)
```

```
Vectorizer: W2Vec    Model: Brute
Best Hyper parameter:  15
AUC:  0.8049
```

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

In [83]:

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

In [84]:

```
#hiding all the warnings
warnings.filterwarnings('ignore')

# Breaking the data into train and test
from sklearn.model_selection import train_test_split
```

```
X = preprocessed_reviews
y = final['Score']
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.3, random_state=42)

print('shape of X:',len(X),'shape of y:',len(y))
```

```
shape of X: 19354 shape of y: 19354
```

In [85]:

```
# Shape of train and test data
print("X_train shape: {0}   y_train shape: {1}".format(len(X_train), len(y_train)))
print("X_test shape:  {0}    y_test shape: {1}".format(len(X_test), len(y_test)))

train_per=len(X_train)/len(X)*100
test_per=len(X_test)/len(X)*100
import math
print("\nPercentage of data in Train : {0:.1f} % and Test :{1:.1f} % ".format(train_per,test_per))
```

```
X_train shape: 13547   y_train shape: 13547
X_test shape:  5807    y_test shape: 5807

Percentage of data in Train : 70.0 % and Test :30.0 %
```

In [86]:

```
model = TfidfVectorizer()
tf_idf_matrix = model.fit(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
i=0
list_of_sentance_train=[]
for sentance in X_train:
    list_of_sentance_train.append(sentance.split())
tfidf_feat = tf_idf_vect.get_feature_names() # tfidf words/col-names
# final_tf_idf is the sparse matrix with row= sentence, col=word and cell_val = tfidf

tfidf_sent_vectors_train = []; # the tfidf-w2v for each sentence/review is stored in this list
row=0;
for sent in tqdm(list_of_sentance_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
    tfidf_sent_vectors_train.append(sent_vec)
    row += 1
```

```
100%|██████████| 13547/13547 [26:32<00:00,  8.51it/s]
```

In [87]:

```
i=0
list_of_sentance_test=[]
for sentance in X_test:
    list_of_sentance_test.append(sentance.split())
tfidf_feat = tf_idf_vect.get_feature_names() # tfidf words/col-names
# final_tf_idf is the sparse matrix with row= sentence, col=word and cell_val = tfidf

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

```
for sent in tqdm(list_of_sentance_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
    tfidf_sent_vectors_test.append(sent_vec)
    row += 1
```

```
100%|██████████| 5807/5807 [01:39<00:00, 58.15it/s]
```

In [88]:

```
# X_train=tfidf_sent_vectors
# X_test=tfidf_sent_vectors_test

X_train=tfidf_sent_vectors_train
X_test=tfidf_sent_vectors_test
```

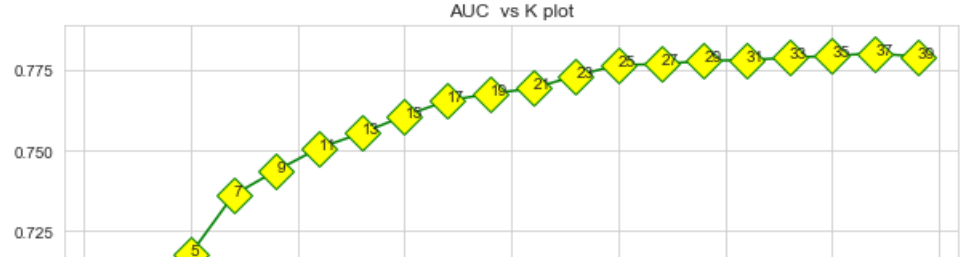
K fold cross validation

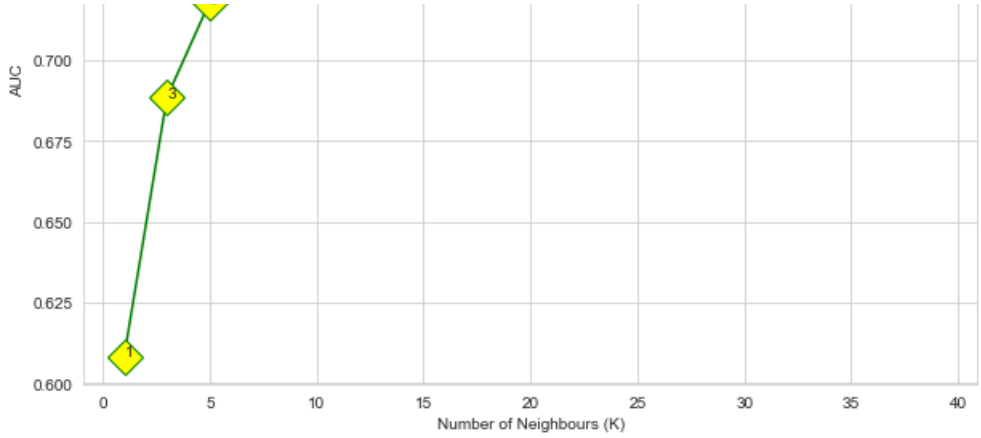
In [89]:

```
# applying knn and finding optimal k

from sklearn.model_selection import cross_val_score
cv_list=[]
neighbors=range(1,40,2)
for K in neighbors:
    K_value = K
    neigh = KNeighborsClassifier(n_neighbors = K_value, weights='uniform', algorithm='brute')
    scores=cross_val_score(neigh, X_train, y_train, cv=5, scoring='roc_auc') # 5 fold cross validat
ion
    cv_list.append(scores.mean())
```

In [90]:

```
# plot of k vs accuracy
plt.figure(figsize = (10,7))

sns.set_style("whitegrid");

plt.plot(neighbors, cv_list, color='green',  marker='D',markerfacecolor='yellow', markersize=15)
ind=0
for val in neighbors:
    plt.annotate('%s' % val,xy=(val,cv_list[ind]),xycoords='data')
    ind=ind+1

plt.title("AUC  vs K plot")
plt.xlabel('Number of Neighbours (K)')
plt.ylabel('AUC')
plt.show()
```





In [91]:

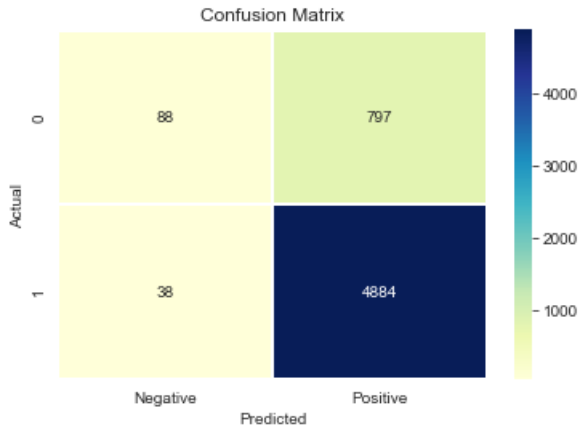
```
# optimal k
opt_k=31
val=math.floor(opt_k/2)+1
print("Maximum AUC score is: ",cv_list[val],"for k=",opt_k)
```

```
Maximum AUC score is:  0.7786659634981625 for k= 31
```

Confusion Matrix

In [92]:

```
import seaborn as sb
from sklearn.metrics import confusion_matrix

knn_optimal = KNeighborsClassifier(n_neighbors=opt_k , algorithm = 'brute')
knn_optimal.fit(X_train, y_train)

conf_matrix = confusion_matrix(y_test,knn_optimal.predict(X_test))
df_cmatrix = pd.DataFrame(conf_matrix, columns=['Negative', 'Positive'])
sb.heatmap(df_cmatrix, annot=True, fmt='d',cmap="YlGnBu", linewidths=1.4)

plt.title("Confusion Matrix")
plt.xlabel("Predicted")
plt.ylabel("Actual")
plt.show()
```



AUC CURVE

In [93]:

```
# calculating train_auc
neighbors=list(range(1,40,2))

train_auc_list=[]
for k in neighbors:
```

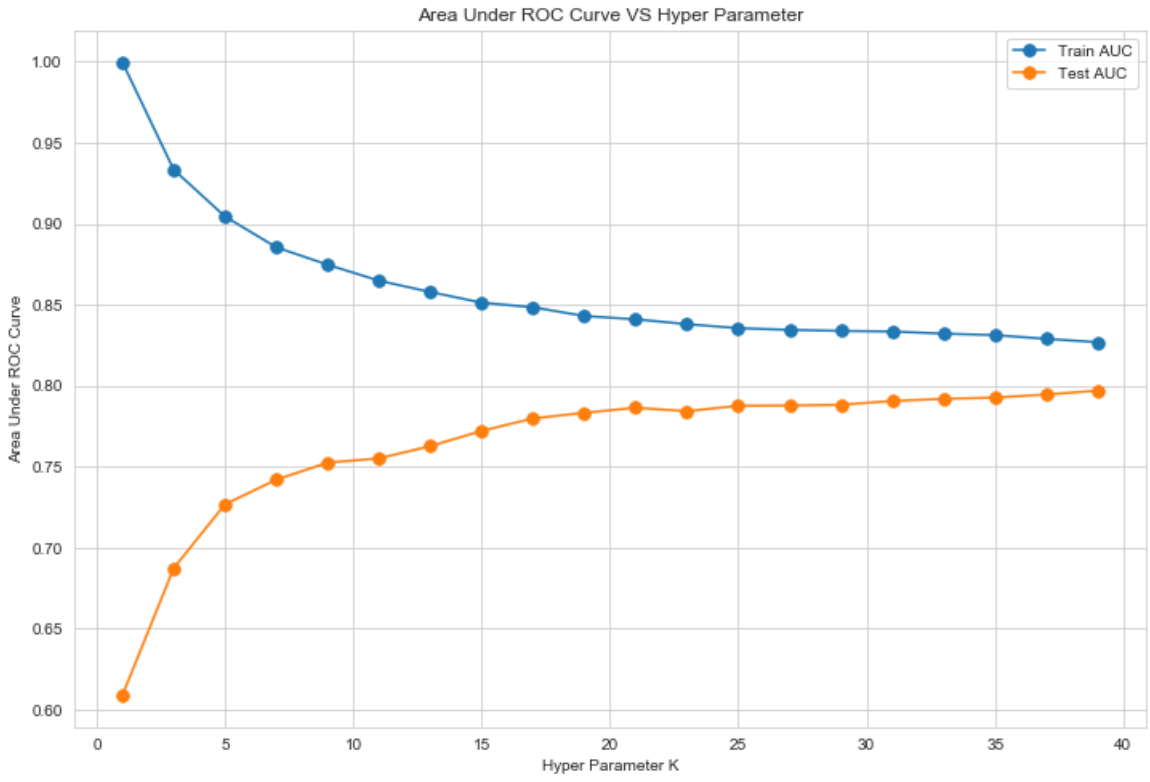
```
    neigh = KNeighborsClassifier(n_neighbors=k , algorithm = 'brute')
    neigh.fit(X_train, y_train)
    # predict probabilities
    probs = neigh.predict_proba(X_train)
    # keep probabilities for the positive outcome only
    probs = probs[:, 1]
    # calculate AUC
    train_auc = roc_auc_score(y_train, probs)
    train_auc_list.append(train_auc)
```

In [94]:

```
# calculating test_auc
neighbors=list(range(1,40,2))

test_auc_list=[]
for k in neighbors:
    neigh = KNeighborsClassifier(n_neighbors=k , algorithm = 'brute')
    neigh.fit(X_train, y_train)
    # predict probabilities
    probs = neigh.predict_proba(X_test)
    # keep probabilities for the positive outcome only
    probs = probs[:, 1]
    # calculate AUC
    test_auc = roc_auc_score(y_test, probs)
    test_auc_list.append(test_auc)
```

In [95]:

```
plt.figure(figsize = (12,8))
plt.plot(neighbors, train_auc_list, marker='.',label="Train AUC",markersize=15)
plt.plot(neighbors, test_auc_list, marker='.',label="Test AUC",markersize=15)
plt.legend()
plt.xlabel("Hyper Parameter K")
plt.ylabel("Area Under ROC Curve")
plt.title("Area Under ROC Curve VS Hyper Parameter")
plt.show()
```



In [96]:

```
knn_optimal = KNeighborsClassifier(n_neighbors=opt_k , algorithm = 'brute')
knn_optimal.fit(X_train, y_train)
```

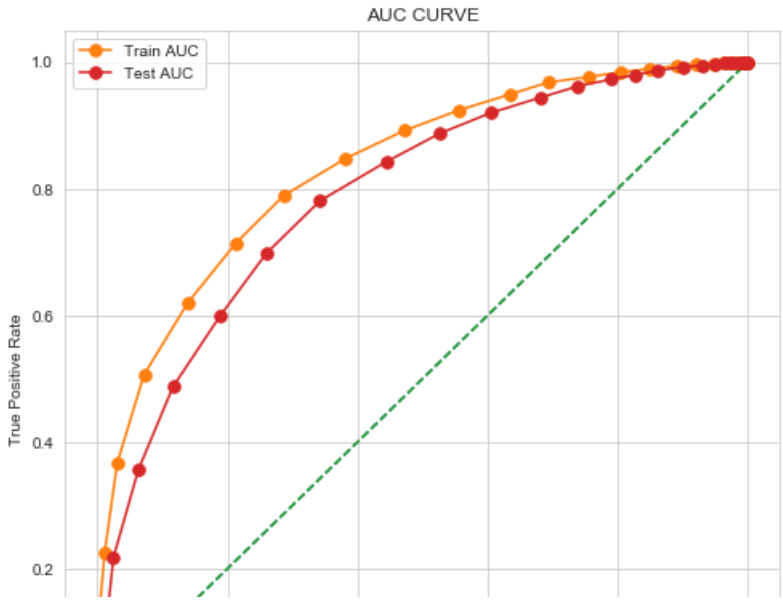
Out[96]:

```
KNeighborsClassifier(algorithm='brute', leaf_size=30, metric='minkowski',
           metric_params=None, n_jobs=None, n_neighbors=31, p=2,
           weights='uniform')
```

In [97]:

```
from matplotlib import pyplot
from sklearn.metrics import roc_auc_score
from sklearn.metrics import roc_curve, auc

plt.figure(figsize = (8,8))
# predict probabilities
probs = knn_optimal.predict_proba(X_train)
# keep probabilities for the positive outcome only
probs = probs[:, 1]
# calculate AUC
train_auc = roc_auc_score(y_train, probs)
print('Train AUC: %f' % train_auc)
# calculate roc curve
fpr, tpr, thresholds = roc_curve(y_train, probs)
# plot no skill
pyplot.plot([0, 1], [0, 1], linestyle='--')
# plot the roc curve for the model
pyplot.plot(fpr, tpr, marker='.',label="Train AUC",markersize=15)


# predict probabilities
probs = knn_optimal.predict_proba(X_test)
# keep probabilities for the positive outcome only
probs = probs[:, 1]
# calculate AUC
test_auc = roc_auc_score(y_test, probs)
print('Test AUC: %f' % test_auc)
# calculate roc curve
fpr, tpr, thresholds = roc_curve(y_test, probs)
# plot no skill
pyplot.plot([0, 1], [0, 1], linestyle='--')
# plot the roc curve for the model
pyplot.plot(fpr, tpr, marker='.',label="Test AUC",markersize=15)

plt.legend()
plt.xlabel("False Positive Rate")
plt.ylabel("True Positive Rate")
plt.title("AUC CURVE")

# show the plot
pyplot.show()
```

```
Train AUC: 0.833422
Test AUC: 0.790504
```



Conclusion

In [98]:

```
tfidf_w2vec_brute_par=opt_k
tfidf_w2vec_bow_brute_auc=np.round(test_auc,4)
print("Vectorizer: TFIDF W2VEC \t Model: Brute")
print("Best Hyper parameter: ",tfidf_w2vec_brute_par)
print("AUC: ",tfidf_w2vec_bow_brute_auc)
```

```
Vectorizer: TFIDF W2VEC   Model: Brute
Best Hyper parameter:  31
AUC:  0.7905
```

[5.2] Applying KNN kd-tree

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

In [99]:

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

In [100]:

```
#hiding all the warnings
warnings.filterwarnings('ignore')

# Breaking the data into train and test
from sklearn.model_selection import train_test_split

X = preprocessed_reviews
y = final['Score']
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.3, random_state=42)

print('shape of X:',len(X),'shape of y:',len(y))
```

```
shape of X: 19354 shape of y: 19354
```

In [101]:

```
#BoW using kd tree


count_vect = CountVectorizer(min_df=10, max_features=500) #in scikit-learn
count_vect.fit(X_train)
print("some feature names ", count_vect.get_feature_names()[:10])
print('='*50)

final_counts = count_vect.transform(X_train)
print("the type of count vectorizer ",type(final_counts))
print("the shape of out text BOW vectorizer ",final_counts.get_shape())
print("the number of unique words ", final_counts.get_shape()[1])

final_test=count_vect.transform(X_test)
```

```
some feature names  ['able', 'absolutely', 'actually', 'add', 'added', 'aftertaste', 'ago',
'almost', 'also', 'alternative']
==================================================
the type of count vectorizer  <class 'scipy.sparse.csr.csr_matrix'>
```

```
the shape of out text BOW vectorizer  (13547, 500)
the number of unique words  500
```

In [102]:

```
# Shape of train and test data
print("X_train shape: {0}   y_train shape: {1}".format(len(X_train), len(y_train)))
print("X_test shape:  {0}    y_test shape: {1}".format(len(X_test), len(y_test)))

train_per=len(X_train)/len(X)*100
test_per=len(X_test)/len(X)*100
import math
print("\nPercentage of data in Train : {0:.1f} % and Test :{1:.1f} % ".format(train_per,test_per))
```

```
X_train shape: 13547   y_train shape: 13547
X_test shape:  5807    y_test shape: 5807

Percentage of data in Train : 70.0 % and Test :30.0 %
```

In [103]:

```
X_train=final_counts.toarray()
X_test=final_test.toarray()
```

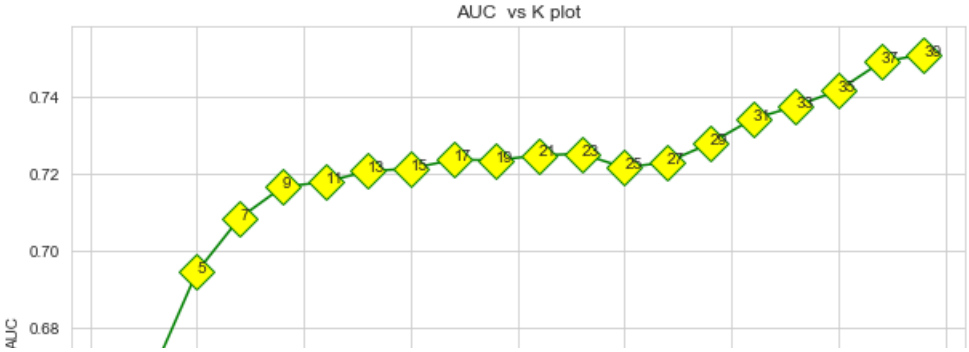
K fold cross validation

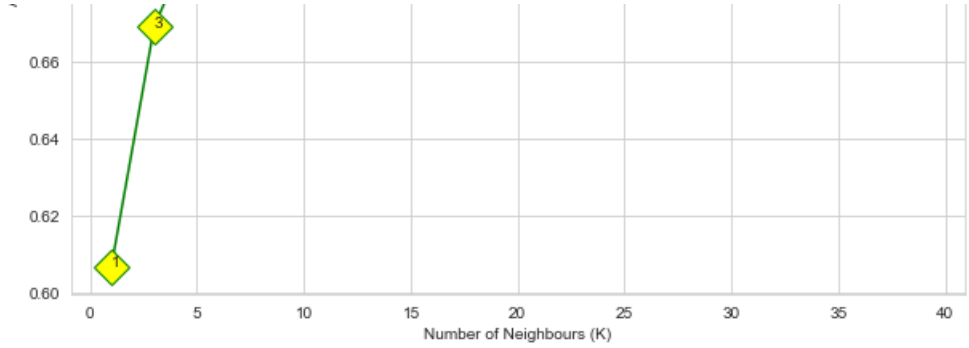
In [104]:

```
# applying knn and finding optimal k

from sklearn.model_selection import cross_val_score
cv_list=[]
neighbors=range(1,40,2)
for K in neighbors:
    K_value = K
    neigh = KNeighborsClassifier(n_neighbors = K_value, weights='uniform', algorithm='kd_tree')
    scores=cross_val_score(neigh, X_train, y_train, cv=5, scoring='roc_auc') # 5 fold cross validat
ion
    cv_list.append(scores.mean())
```

In [105]:

```
# plot of k vs accuracy
plt.figure(figsize = (10,7))

sns.set_style("whitegrid");

plt.plot(neighbors, cv_list, color='green',  marker='D',markerfacecolor='yellow', markersize=15)
ind=0
for val in neighbors:
    plt.annotate('%s' % val,xy=(val,cv_list[ind]),xycoords='data')
    ind=ind+1

plt.title("AUC  vs K plot")
plt.xlabel('Number of Neighbours (K)')
plt.ylabel('AUC')
plt.show()
```





In [106]:

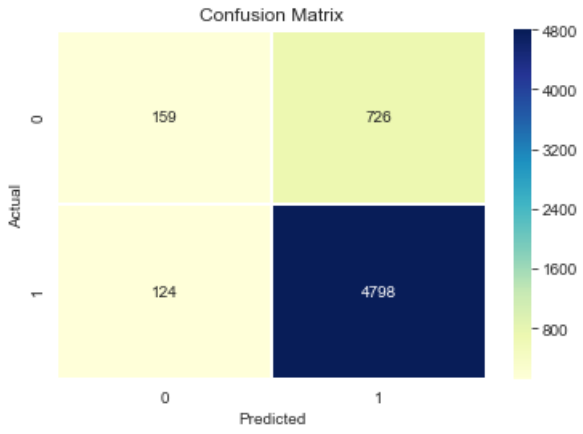
```
# optimal k
opt_k=15
val=math.floor(opt_k/2)+1
print("Maximum AUC score is: ",cv_list[val],"for k=",opt_k)
```

```
Maximum AUC score is:  0.7235218029613486 for k= 15
```

Confusion Matrix

In [107]:

```
import seaborn as sb
from sklearn.metrics import confusion_matrix

knn_optimal = KNeighborsClassifier(n_neighbors=opt_k , algorithm = 'kd_tree')
knn_optimal.fit(X_train, y_train)

conf_matrix = confusion_matrix(y_test,knn_optimal.predict(X_test))
df_cmatrix = pd.DataFrame(conf_matrix) #, columns=['Negative', 'Positive'])
sb.heatmap(df_cmatrix, annot=True, fmt='d',cmap="YlGnBu", linewidths=1.4)

plt.title("Confusion Matrix")
plt.xlabel("Predicted")
plt.ylabel("Actual")
plt.show()
```



AUC CURVE

In [108]:

```
# calculating train_auc
neighbors=list(range(1,40,2))

train_auc_list=[]
for k in neighbors:
    neigh = KNeighborsClassifier(n_neighbors=k , algorithm = 'brute')
    neigh.fit(X_train, y_train)
    # predict probabilities
```

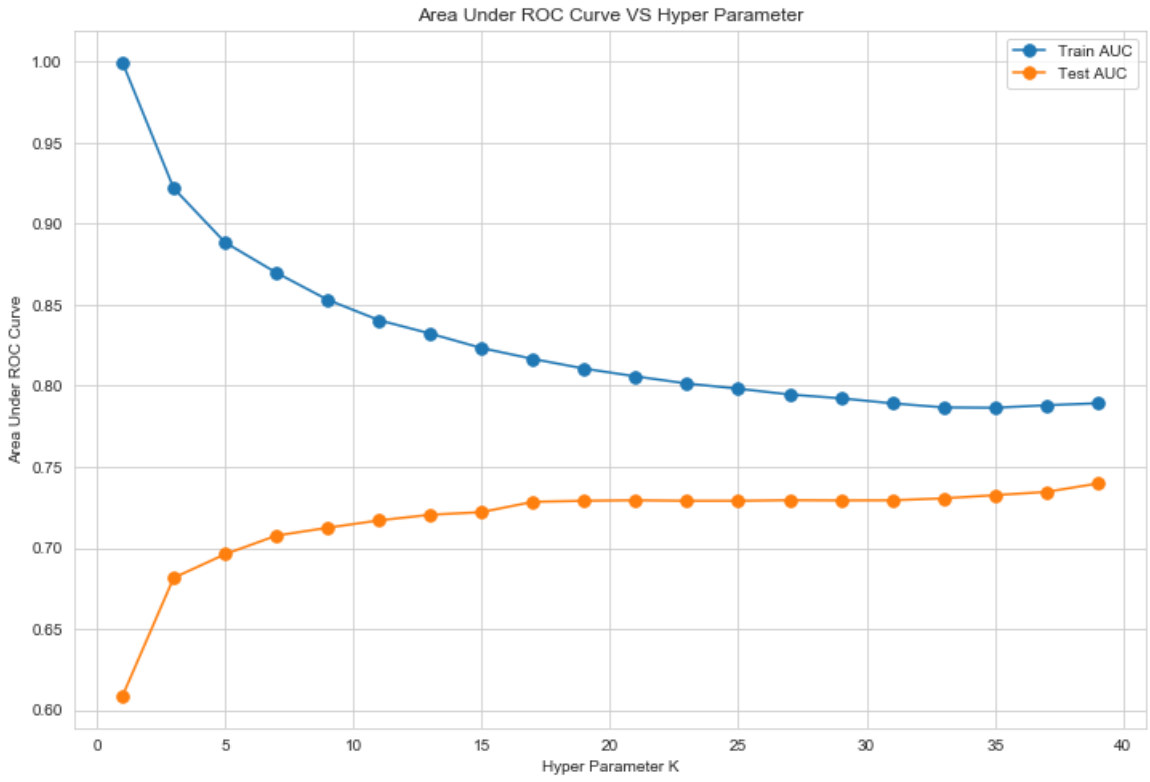
```
    probs = neigh.predict_proba(X_train)
    # keep probabilities for the positive outcome only
    probs = probs[:, 1]
    # calculate AUC
    train_auc = roc_auc_score(y_train, probs)
    train_auc_list.append(train_auc)
```

In [109]:

```
# calculating test_auc
neighbors=list(range(1,40,2))

test_auc_list=[]
for k in neighbors:
    neigh = KNeighborsClassifier(n_neighbors=k , algorithm = 'brute')
    neigh.fit(X_train, y_train)
    # predict probabilities
    probs = neigh.predict_proba(X_test)
    # keep probabilities for the positive outcome only
    probs = probs[:, 1]
    # calculate AUC
    test_auc = roc_auc_score(y_test, probs)
    test_auc_list.append(test_auc)
```

In [110]:

```
plt.figure(figsize = (12,8))
plt.plot(neighbors, train_auc_list, marker='.',label="Train AUC",markersize=15)
plt.plot(neighbors, test_auc_list, marker='.',label="Test AUC",markersize=15)
plt.legend()
plt.xlabel("Hyper Parameter K")
plt.ylabel("Area Under ROC Curve")
plt.title("Area Under ROC Curve VS Hyper Parameter")
plt.show()
```



In [111]:

```
knn_optimal = KNeighborsClassifier(n_neighbors=opt_k , algorithm = 'kd_tree')
knn_optimal.fit(X_train, y_train)
```

Out[111]:

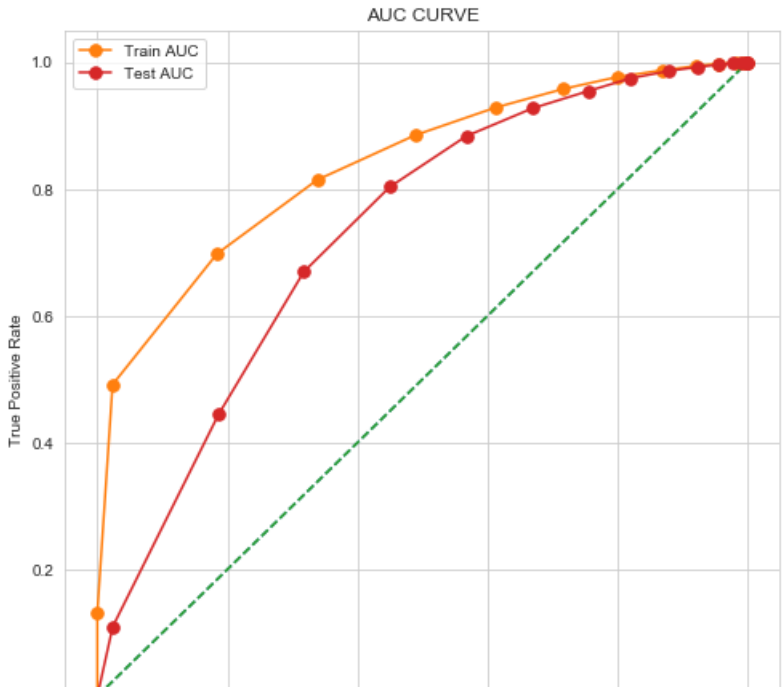
```
KNeighborsClassifier(algorithm='kd_tree', leaf_size=30, metric='minkowski',
```

```
metric_params=None, n_jobs=None, n_neighbors=15, p=2,
weights='uniform')
```

In [112]:

```
from matplotlib import pyplot
from sklearn.metrics import roc_auc_score
from sklearn.metrics import roc_curve, auc

plt.figure(figsize = (8,8))
# predict probabilities
probs = knn_optimal.predict_proba(X_train)
# keep probabilities for the positive outcome only
probs = probs[:, 1]
# calculate AUC
train_auc = roc_auc_score(y_train, probs)
print('Train AUC: %f' % train_auc)
# calculate roc curve
fpr, tpr, thresholds = roc_curve(y_train, probs)
# plot no skill
pyplot.plot([0, 1], [0, 1], linestyle='--')
# plot the roc curve for the model
pyplot.plot(fpr, tpr, marker='.',label="Train AUC",markersize=15)


# predict probabilities
probs = knn_optimal.predict_proba(X_test)
# keep probabilities for the positive outcome only
probs = probs[:, 1]
# calculate AUC
test_auc = roc_auc_score(y_test, probs)
print('Test AUC: %f' % test_auc)
# calculate roc curve
fpr, tpr, thresholds = roc_curve(y_test, probs)
# plot no skill
pyplot.plot([0, 1], [0, 1], linestyle='--')
# plot the roc curve for the model
pyplot.plot(fpr, tpr, marker='.',label="Test AUC",markersize=15)

plt.legend()
plt.xlabel("False Positive Rate")
plt.ylabel("True Positive Rate")
plt.title("AUC CURVE")

# show the plot
pyplot.show()
```

```
Train AUC: 0.836504
Test AUC: 0.729737
```



Conclusion

In [113]:

```
bow_kdtree_par=opt_k
bow_kdtree_auc=np.round(test_auc,4)
print("Vectorizer: BOW \t Model: KD TREE")
print("Best Hyper parameter: ",bow_kdtree_par)
print("AUC: ",bow_kdtree_auc)
```

```
Vectorizer: BOW   Model: KD TREE
Best Hyper parameter:  15
AUC:  0.7297
```

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

In [114]:

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

In [115]:

```
#hiding all the warnings
warnings.filterwarnings('ignore')

# Breaking the data into train and test
from sklearn.model_selection import train_test_split

X = preprocessed_reviews
y = final['Score']
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.3, random_state=42)

print('shape of X:',len(X),'shape of y:',len(y))
```

```
shape of X: 19354 shape of y: 19354
```

In [116]:

```
tf_idf_vect = TfidfVectorizer(ngram_range=(1,2))
tf_idf_vect.fit(X_train)
X_train_tfidf= tf_idf_vect.transform(X_train)

X_test_tfidf=tf_idf_vect.transform(X_test)
```

In [117]:

```
# Shape of train and test data
print("X_train shape: {0}   y_train shape: {1}".format(len(X_train), len(y_train)))
print("X_test shape:  {0}    y_test shape: {1}".format(len(X_test), len(y_test)))

train_per=len(X_train)/len(X)*100
test_per=len(X_test)/len(X)*100
import math
print("\nPercentage of data in Train : {0:.1f} % and Test :{1:.1f} % ".format(train_per,test_per))
```

```
X_train shape: 13547   y_train shape: 13547
X_test shape:  5807    y_test shape: 5807

Percentage of data in Train : 70.0 % and Test :30.0 %
```

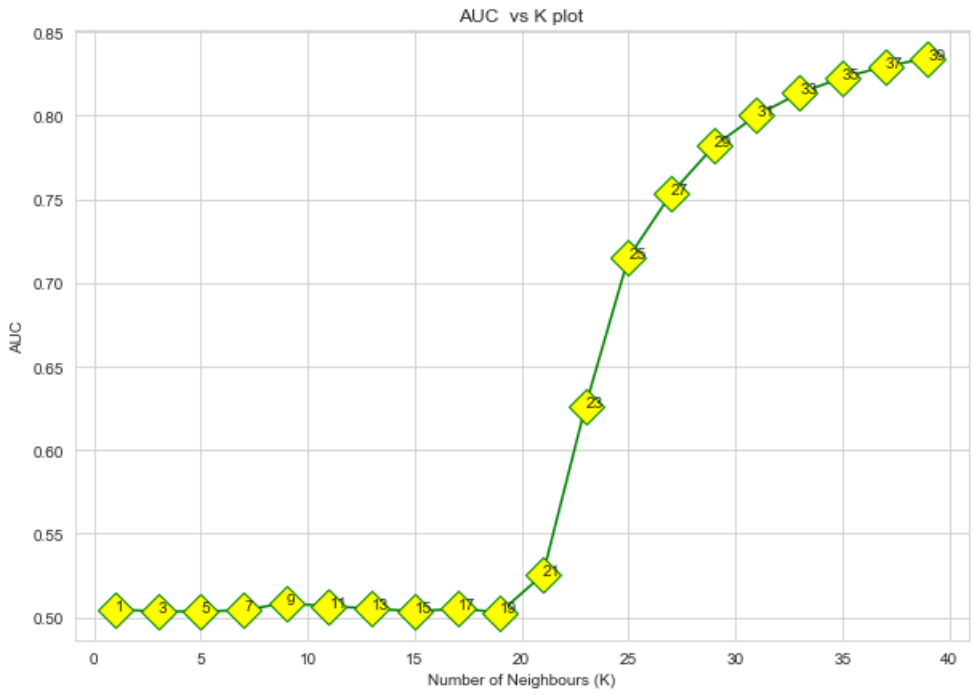
K fold cross validation

In [118]:

```
# applying knn and finding optimal k

from sklearn.model_selection import cross_val_score
cv_list=[]
neighbors=range(1,40,2)
for K in neighbors:
    K_value = K
    neigh = KNeighborsClassifier(n_neighbors = K_value, weights='uniform', algorithm='kd_tree')
    scores=cross_val_score(neigh, X_train_tfidf, y_train, cv=5, scoring='roc_auc') # 5 fold cross v
alidation
    cv_list.append(scores.mean())
```

In [119]:

```
# plot of k vs accuracy
plt.figure(figsize = (10,7))

sns.set_style("whitegrid");

plt.plot(neighbors, cv_list, color='green',  marker='D',markerfacecolor='yellow', markersize=15)
ind=0
for val in neighbors:
    plt.annotate('%s' % val,xy=(val,cv_list[ind]),xycoords='data')
    ind=ind+1

plt.title("AUC  vs K plot")
plt.xlabel('Number of Neighbours (K)')
plt.ylabel('AUC')
plt.show()
```



In [167]:

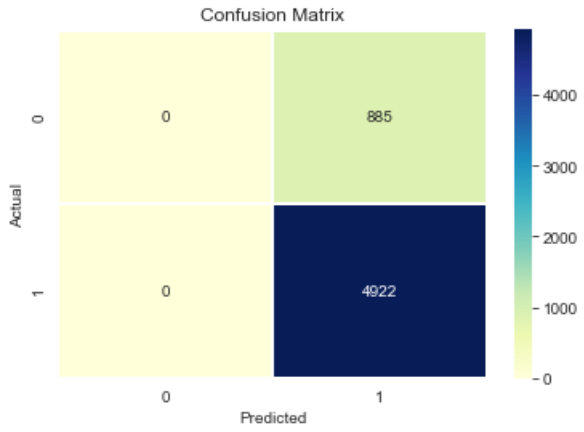
```
# optimal k
opt_k=31
val=math.floor(opt_k/2)+1
print("Maximum AUC score is: ",cv_list[val],"for k=",opt_k)
```

```
Maximum AUC score is:  0.7698399723838025 for k= 31
```

Confusion Matrix

In [168]:

```
import seaborn as sb
from sklearn.metrics import confusion_matrix

knn_optimal = KNeighborsClassifier(n_neighbors=opt_k , algorithm = 'kd_tree')
knn_optimal.fit(X_train_tfidf, y_train)

conf_matrix = confusion_matrix(y_test,knn_optimal.predict(X_test_tfidf))
df_cmatrix = pd.DataFrame(conf_matrix) #, columns=['Negative', 'Positive'])
sb.heatmap(df_cmatrix, annot=True, fmt='d',cmap="YlGnBu", linewidths=1.4)

plt.title("Confusion Matrix")
plt.xlabel("Predicted")
plt.ylabel("Actual")
plt.show()
```



AUC CURVE

In [122]:

```
# calculating train_auc
neighbors=list(range(1,40,2))

train_auc_list=[]
for k in neighbors:
    neigh = KNeighborsClassifier(n_neighbors=k , algorithm = 'brute')
    neigh.fit(X_train_tfidf, y_train)
    # predict probabilities
    probs = neigh.predict_proba(X_train_tfidf)
    # keep probabilities for the positive outcome only
    probs = probs[:, 1]
    # calculate AUC
    train_auc = roc_auc_score(y_train, probs)
    train_auc_list.append(train_auc)
```

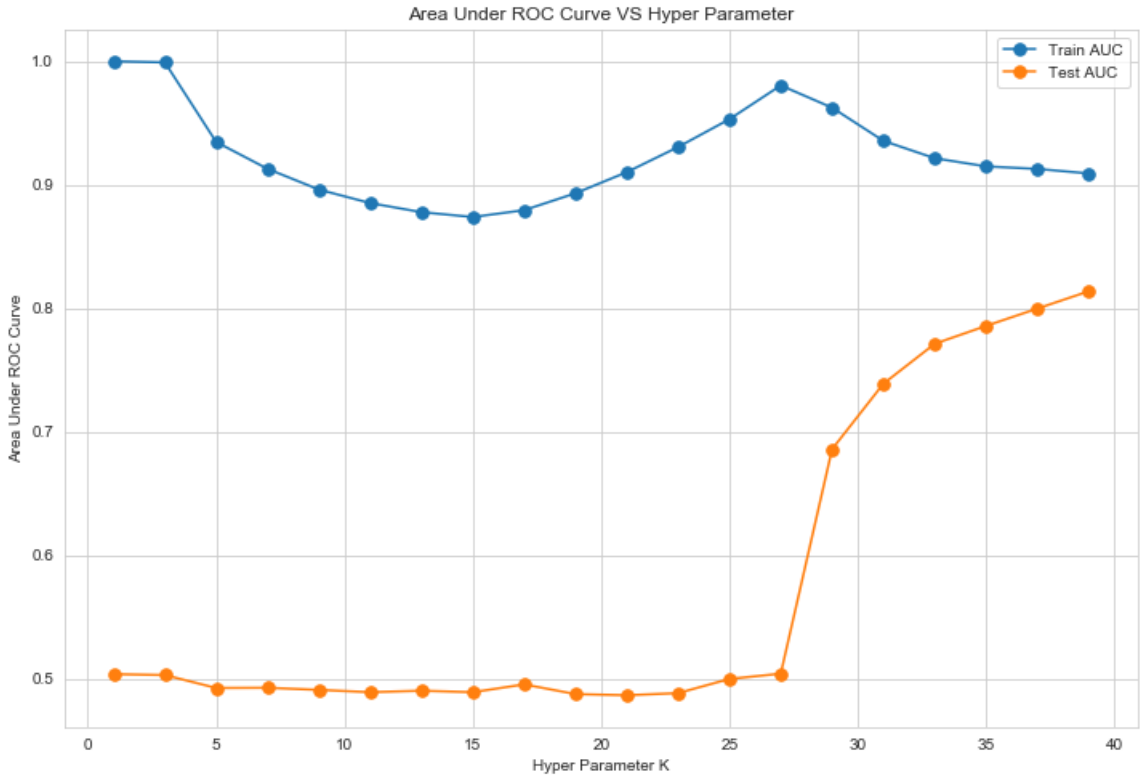
In [123]:

```
# calculating test_auc
neighbors=list(range(1,40,2))

test_auc_list=[]
for k in neighbors:
    neigh = KNeighborsClassifier(n_neighbors=k , algorithm = 'brute')
    neigh.fit(X_train_tfidf, y_train)
    # predict probabilities
    probs = neigh.predict_proba(X_test_tfidf)
    # keep probabilities for the positive outcome only
    probs = probs[:, 1]
    # calculate AUC
    test_auc = roc_auc_score(y_test, probs)
    test_auc_list.append(test_auc)
```

In [124]:

```
plt.figure(figsize = (12,8))
plt.plot(neighbors, train_auc_list, marker='.', label="Train AUC",markersize=15)
```

```
plt.plot(neighbors, train_auc_list, marker='.',label="Train AUC",markersize=15)
plt.plot(neighbors, test_auc_list, marker='.',label="Test AUC",markersize=15)
plt.legend()
plt.xlabel("Hyper Parameter K")
plt.ylabel("Area Under ROC Curve")
plt.title("Area Under ROC Curve VS Hyper Parameter")
plt.show()
```



In [169]:

```
knn_optimal = KNeighborsClassifier(n_neighbors=opt_k , algorithm = 'kd_tree')
knn_optimal.fit(X_train_tfidf, y_train)
```

Out[169]:

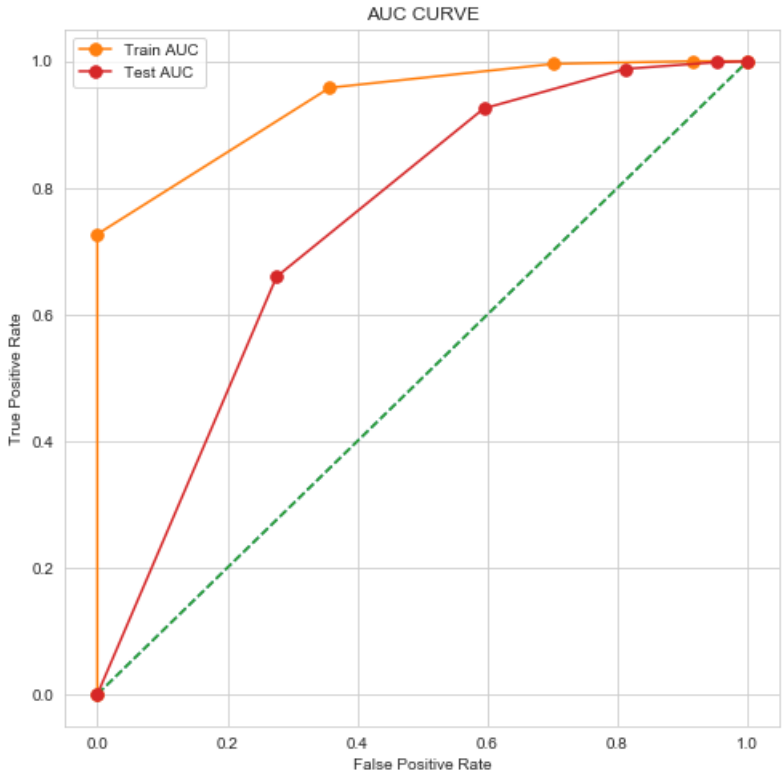
```
KNeighborsClassifier(algorithm='kd_tree', leaf_size=30, metric='minkowski',
           metric_params=None, n_jobs=None, n_neighbors=31, p=2,
           weights='uniform')
```

In [170]:

```
from matplotlib import pyplot
from sklearn.metrics import roc_auc_score
from sklearn.metrics import roc_curve, auc

plt.figure(figsize = (8,8))
# predict probabilities
probs = knn_optimal.predict_proba(X_train_tfidf)
# keep probabilities for the positive outcome only
probs = probs[:, 1]
# calculate AUC
train_auc = roc_auc_score(y_train, probs)
print('Train AUC: %f' % train_auc)
# calculate roc curve
fpr, tpr, thresholds = roc_curve(y_train, probs)
# plot no skill
pyplot.plot([0, 1], [0, 1], linestyle='--')
# plot the roc curve for the model
pyplot.plot(fpr, tpr, marker='.',label="Train AUC",markersize=15)


# predict probabilities
probs = knn_optimal.predict_proba(X_test_tfidf)
# keep probabilities for the positive outcome only
probs = probs[:, 1]
```

```
# calculate AUC
test_auc = roc_auc_score(y_test, probs)
print('Test AUC: %f' % test_auc)
# calculate roc curve
fpr, tpr, thresholds = roc_curve(y_test, probs)
# plot no skill
pyplot.plot([0, 1], [0, 1], linestyle='--')
# plot the roc curve for the model
pyplot.plot(fpr, tpr, marker='.',label="Test AUC",markersize=15)

plt.legend()
plt.xlabel("False Positive Rate")
plt.ylabel("True Positive Rate")
plt.title("AUC CURVE")

# show the plot
pyplot.show()
```

```
Train AUC: 0.935298
Test AUC: 0.738616
```



Conclusion

In [171]:

```
tfidf_kdtree_par=opt_k
tfidf_kdtree_auc=np.round(test_auc,4)
print("Vectorizer: TFIDF \t Model: KD TREE")
print("Best Hyper parameter: ",tfidf_kdtree_par)
print("AUC: ",tfidf_kdtree_auc)
```

```
Vectorizer: TFIDF   Model: KD TREE
Best Hyper parameter:  31
AUC:  0.7386
```

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

In [128]:

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

In [129]:

```
#hiding all the warnings
warnings.filterwarnings('ignore')

# Breaking the data into train and test
from sklearn.model_selection import train_test_split

X = preprocessed_reviews
y = final['Score']
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.3, random_state=42)

print('shape of X:',len(X),'shape of y:',len(y))
```

```
shape of X: 19354 shape of y: 19354
```

In [130]:

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

In [131]:

```
is_your_ram_gt_16g=False
want_to_use_google_w2v = False
want_to_train_w2v = True

if want_to_train_w2v:
    # min_count = 5 considers only words that occured atleast 5 times
    w2v_model=Word2Vec(list_of_sentance,min_count=5,size=50, workers=4)
    print(w2v_model.wv.most_similar('great'))
    print('='*50)
    print(w2v_model.wv.most_similar('worst'))

elif want_to_use_google_w2v and is_your_ram_gt_16g:
    if os.path.isfile('GoogleNews-vectors-negative300.bin'):
        w2v_model=KeyedVectors.load_word2vec_format('GoogleNews-vectors-negative300.bin', binary=Tr
ue)
        print(w2v_model.wv.most_similar('great'))
        print(w2v_model.wv.most_similar('worst'))
    else:
        print("you don't have gogole's word2vec file, keep want_to_train_w2v = True, to train your
own w2v ")
```

```
[('good', 0.8574337363243103), ('excellent', 0.8093309998512268), ('fantastic',
0.7782913446426392), ('awesome', 0.7498130798339844), ('wonderful', 0.7497345209121704),
('amazing', 0.7462096810340881), ('quick', 0.7188891172409058), ('super', 0.712965190410614),
('decent', 0.6978143453598022), ('love', 0.6887656450271606)]
==================================================
[('world', 0.9303915500640869), ('disappointing', 0.9301894903182983), ('among',
0.9276330471038818), ('reordered', 0.9245986938476562), ('occur', 0.924347460269928), ('mediocre',
0.9234187006950378), ('saltines', 0.9205341339111328), ('raving', 0.9177750945091248), ('vegas', 0
.915202796459198), ('experiences', 0.9148045778274536)]
```

In [132]:

```
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])
print(len(w2v_words))
# length=len(w2v_words)
```

```
number of words that occured minimum 5 times  7000
sample words  ['good', 'variety', 'chocolate', 'bad', 'everyone', 'says', 'flavors', 'great',
'gives', 'family', 'friends', 'latte', 'steamer', 'favorite', 'hazelnut', 'stick', 'time', 'buy',
'horrible', 'step', 'pop', 'change', 'formula', 'use', 'cheap', 'barley', 'malt', 'sugar', 'high',
'glycemic', 'place', 'honey', 'amount', 'herbs', 'effective', 'original', 'huge',
'disappointment', 'shame', 'outstanding', 'products', 'reviewer', 'recommended', 'go', 'way', 'ste
```

```
er', 'clear', 'recommending', 'anything', 'prince']
7000
```

In [133]:

```
# 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 this
to 300 if you use google's w2v
    cnt_words =0; # num of words with a valid vector in the sentence/review
    for word in sent: # for each word in a review/sentence
        if word in w2v_words:
            vec = w2v_model.wv[word]
            sent_vec += vec
            cnt_words += 1
    if cnt_words != 0:
        sent_vec /= cnt_words
    sent_vectors.append(sent_vec)
print(len(sent_vectors))
print(len(sent_vectors[0]))
sent_vectors=np.array(sent_vectors)
```

```
100%|██████████| 13547/13547 [00:17<00:00, 773.32it/s]
```

```
13547
50
```

In [134]:

```
sent_vec_test=[]
sentance_test=[]
for sentance in X_test:
    sentance_test.append(sentance.split())


for sent in tqdm(sentance_test): # for each review/sentence
    sent_vec = np.zeros(50) # as word vectors are of zero length 50, you might need to change this
to 300 if you use google's w2v
    cnt_words =0; # num of words with a valid vector in the sentence/review
    for word in sent: # for each word in a review/sentence
        if word in w2v_words:
            vec = w2v_model.wv[word]
            sent_vec += vec
            cnt_words += 1
    if cnt_words != 0:
        sent_vec /= cnt_words
    sent_vec_test.append(sent_vec)
print(len(sent_vec_test))
print(len(sent_vec_test[0]))
sent_vec_test=np.array(sent_vec_test)
```

```
100%|██████████| 5807/5807 [00:08<00:00, 713.68it/s]
```

```
5807
50
```

In [135]:

```
# Shape of train and test data
print("X_train shape: {0}   y_train shape: {1}".format(len(X_train), len(y_train)))
print("X_test shape:  {0}    y_test shape: {1}".format(len(X_test), len(y_test)))

train_per=len(X_train)/len(X)*100
test_per=len(X_test)/len(X)*100
import math
print("\nPercentage of data in Train : {0:.1f} % and Test :{1:.1f} % ".format(train_per,test_per))
```

```
X_train shape: 13547   y_train shape: 13547
X_test shape:  5807    y_test shape: 5807

Percentage of data in Train : 70.0 % and Test :30.0 %
```

In [136]:

```
X_train=sent_vectors
X_test=sent_vec_test
```

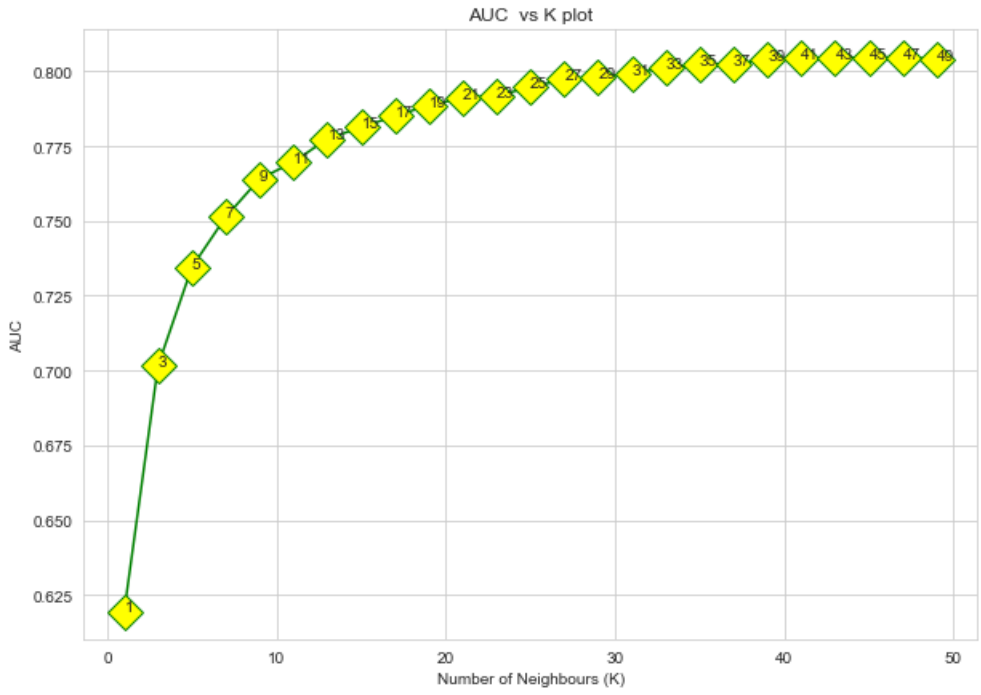
K fold cross validation

In [138]:

```
# applying knn and finding optimal k

from sklearn.model_selection import cross_val_score
cv_list=[]
neighbors=range(1,50,2)
for K in neighbors:
    K_value = K
    neigh = KNeighborsClassifier(n_neighbors = K_value, weights='uniform', algorithm='kd_tree')
    scores=cross_val_score(neigh, X_train, y_train, cv=5, scoring='roc_auc') # 5 fold cross validat
ion
    cv_list.append(scores.mean())
```

In [139]:

```
# plot of k vs accuracy
plt.figure(figsize = (10,7))

sns.set_style("whitegrid");

plt.plot(neighbors, cv_list, color='green',  marker='D',markerfacecolor='yellow', markersize=15)
ind=0
for val in neighbors:
    plt.annotate('%s' % val,xy=(val,cv_list[ind]),xycoords='data')
    ind=ind+1

plt.title("AUC  vs K plot")
plt.xlabel('Number of Neighbours (K)')
plt.ylabel('AUC')
plt.show()
```



In [140]:

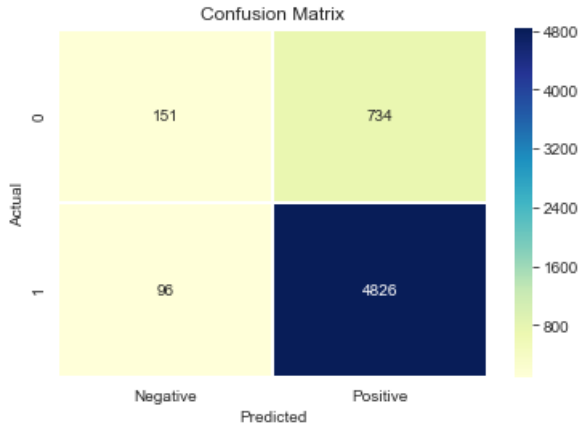
```
# optimal k
opt_k=15
val=math.floor(opt_k/2)+1
print("Maximum AUC score is: ",cv_list[val],"for k=",opt_k)
```

```
Maximum AUC score is:  0.7850203491301104 for k= 15
```

Confusion Matrix

In [141]:

```
import seaborn as sb
from sklearn.metrics import confusion_matrix

knn_optimal = KNeighborsClassifier(n_neighbors=opt_k , algorithm = 'kd_tree')
knn_optimal.fit(X_train, y_train)

conf_matrix = confusion_matrix(y_test,knn_optimal.predict(X_test))
df_cmatrix = pd.DataFrame(conf_matrix, columns=['Negative', 'Positive'])
sb.heatmap(df_cmatrix, annot=True, fmt='d',cmap="YlGnBu", linewidths=1.4)

plt.title("Confusion Matrix")
plt.xlabel("Predicted")
plt.ylabel("Actual")
plt.show()
```



AUC CURVE

In [142]:

```
# calculating train_auc
neighbors=list(range(1,40,2))

train_auc_list=[]
for k in neighbors:
    neigh = KNeighborsClassifier(n_neighbors=k , algorithm = 'brute')
    neigh.fit(X_train, y_train)
    # predict probabilities
    probs = neigh.predict_proba(X_train)
    # keep probabilities for the positive outcome only
    probs = probs[:, 1]
    # calculate AUC
    train_auc = roc_auc_score(y_train, probs)
    train_auc_list.append(train_auc)
```

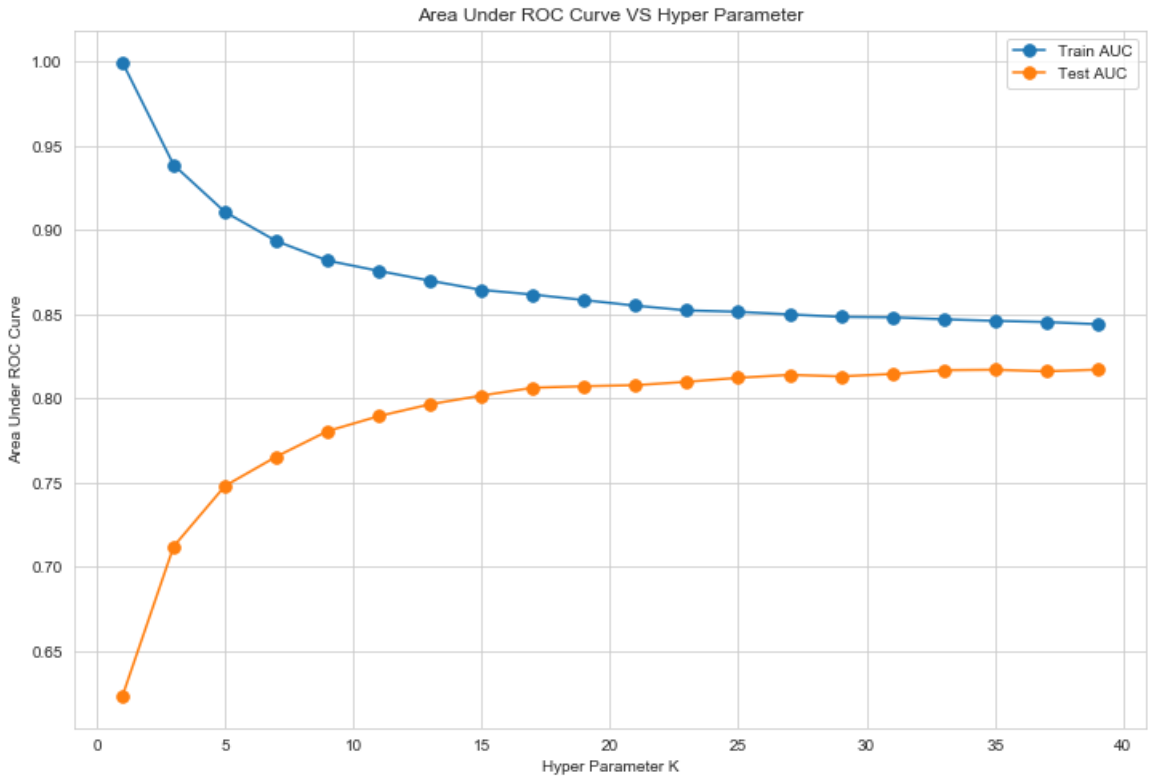
In [143]:

```
# calculating test_auc
neighbors=list(range(1,40,2))

test_auc_list=[]
```

```
for k in neighbors:
    neigh = KNeighborsClassifier(n_neighbors=k , algorithm = 'brute')
    neigh.fit(X_train, y_train)
    # predict probabilities
    probs = neigh.predict_proba(X_test)
    # keep probabilities for the positive outcome only
    probs = probs[:, 1]
    # calculate AUC
    test_auc = roc_auc_score(y_test, probs)
    test_auc_list.append(test_auc)
```

In [144]:

```
plt.figure(figsize = (12,8))
plt.plot(neighbors, train_auc_list, marker='.',label="Train AUC",markersize=15)
plt.plot(neighbors, test_auc_list, marker='.',label="Test AUC",markersize=15)
plt.legend()
plt.xlabel("Hyper Parameter K")
plt.ylabel("Area Under ROC Curve")
plt.title("Area Under ROC Curve VS Hyper Parameter")
plt.show()
```



In [145]:

```
knn_optimal = KNeighborsClassifier(n_neighbors=opt_k , algorithm = 'kd_tree')
knn_optimal.fit(X_train, y_train)
```

Out[145]:

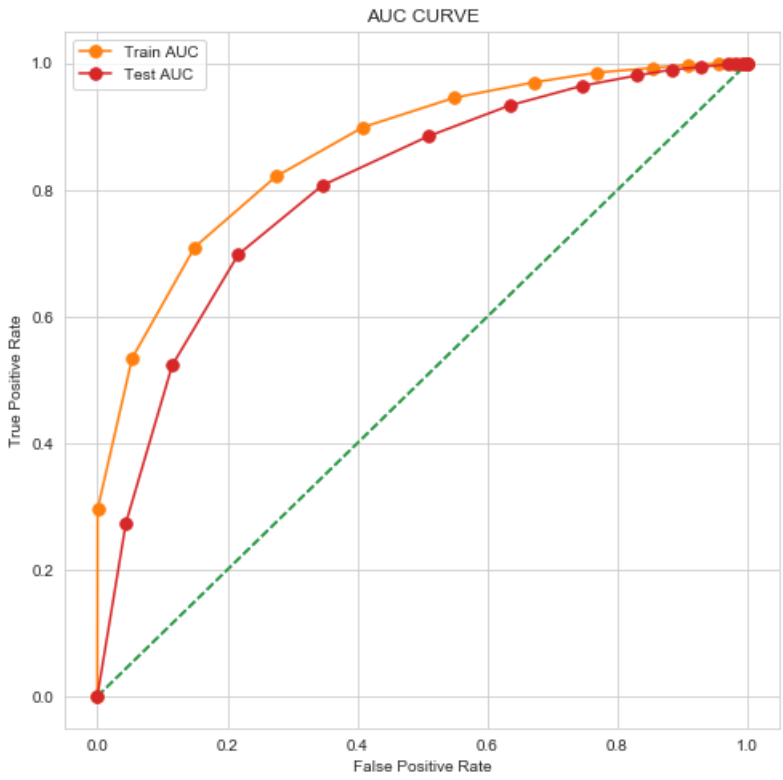
```
KNeighborsClassifier(algorithm='kd_tree', leaf_size=30, metric='minkowski',
           metric_params=None, n_jobs=None, n_neighbors=15, p=2,
           weights='uniform')
```

In [146]:

```
from matplotlib import pyplot
from sklearn.metrics import roc_auc_score
from sklearn.metrics import roc_curve, auc

plt.figure(figsize = (8,8))
# predict probabilities
probs = knn_optimal.predict_proba(X_train)
# keep probabilities for the positive outcome only
probs = probs[:, 1]
```

```
# calculate AUC
train_auc = roc_auc_score(y_train, probs)
print('Train AUC: %f' % train_auc)
# calculate roc curve
fpr, tpr, thresholds = roc_curve(y_train, probs)
# plot no skill
pyplot.plot([0, 1], [0, 1], linestyle='--')
# plot the roc curve for the model
pyplot.plot(fpr, tpr, marker='.',label="Train AUC",markersize=15)


# predict probabilities
probs = knn_optimal.predict_proba(X_test)
# keep probabilities for the positive outcome only
probs = probs[:, 1]
# calculate AUC
test_auc = roc_auc_score(y_test, probs)
print('Test AUC: %f' % test_auc)
# calculate roc curve
fpr, tpr, thresholds = roc_curve(y_test, probs)
# plot no skill
pyplot.plot([0, 1], [0, 1], linestyle='--')
# plot the roc curve for the model
pyplot.plot(fpr, tpr, marker='.',label="Test AUC",markersize=15)

plt.legend()
plt.xlabel("False Positive Rate")
plt.ylabel("True Positive Rate")
plt.title("AUC CURVE")

# show the plot
pyplot.show()
```

```
Train AUC: 0.864475
Test AUC: 0.801727
```



Conclusion

In [147]:

```
w2vec_kdtree_par=opt_k
w2vec_kdtree_auc=np.round(test_auc,4)
print("Vectorizer: W2VEC \t Model: KD TREE")
print("Best Hyper parameter: " w2vec_kdtree_par)
```

```
print("Best Hyper parameter:  ",w2vec_kdtree_par)
print("AUC: ",w2vec_kdtree_auc)
```

```
Vectorizer: W2VEC   Model: KD TREE
Best Hyper parameter:  15
AUC:  0.8017
```

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

In [148]:

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

In [149]:

```
#hiding all the warnings
warnings.filterwarnings('ignore')

# Breaking the data into train and test
from sklearn.model_selection import train_test_split

X = preprocessed_reviews
y = final['Score']
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.3, random_state=42)

print('shape of X:',len(X),'shape of y:',len(y))
```

```
shape of X: 19354 shape of y: 19354
```

In [150]:

```
# Shape of train and test data
print("X_train shape: {0}   y_train shape: {1}".format(len(X_train), len(y_train)))
print("X_test shape:  {0}    y_test shape: {1}".format(len(X_test), len(y_test)))

train_per=len(X_train)/len(X)*100
test_per=len(X_test)/len(X)*100
import math
print("\nPercentage of data in Train : {0:.1f} % and Test :{1:.1f} % ".format(train_per,test_per))
```

```
X_train shape: 13547   y_train shape: 13547
X_test shape:  5807    y_test shape: 5807

Percentage of data in Train : 70.0 % and Test :30.0 %
```

In [151]:

```
model = TfidfVectorizer()
tf_idf_matrix = model.fit(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
i=0
list_of_sentance_train=[]
for sentance in X_train:
    list_of_sentance_train.append(sentance.split())
tfidf_feat = tf_idf_vect.get_feature_names() # tfidf words/col-names
# final_tf_idf is the sparse matrix with row= sentence, col=word and cell_val = tfidf

tfidf_sent_vectors_train = []; # the tfidf-w2v for each sentence/review is stored in this list
row=0;
for sent in tqdm(list_of_sentance_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 values 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_train.append(sent_vec)
    row += 1
```

```
100%|██████████| 13547/13547 [2:33:50<00:00,  2.07it/s]
```

In [152]:

```
i=0
list_of_sentance_test=[]
for sentance in X_test:
    list_of_sentance_test.append(sentance.split())
tfidf_feat = tf_idf_vect.get_feature_names() # tfidf words/col-names
# final_tf_idf is the sparse matrix with row= sentence, col=word and cell_val = tfidf

tfidf_sent_vectors_test = []; # the tfidf-w2v for each sentence/review is stored in this list
row=0;
for sent in tqdm(list_of_sentance_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
    tfidf_sent_vectors_test.append(sent_vec)
    row += 1
```

```
100%|██████████| 5807/5807 [40:47<00:00,  2.87it/s]
```

In [153]:

```
# X_train=tfidf_sent_vectors
# X_test=tfidf_sent_vectors_test

X_train=tfidf_sent_vectors_train
X_test=tfidf_sent_vectors_test
```

K fold cross validation

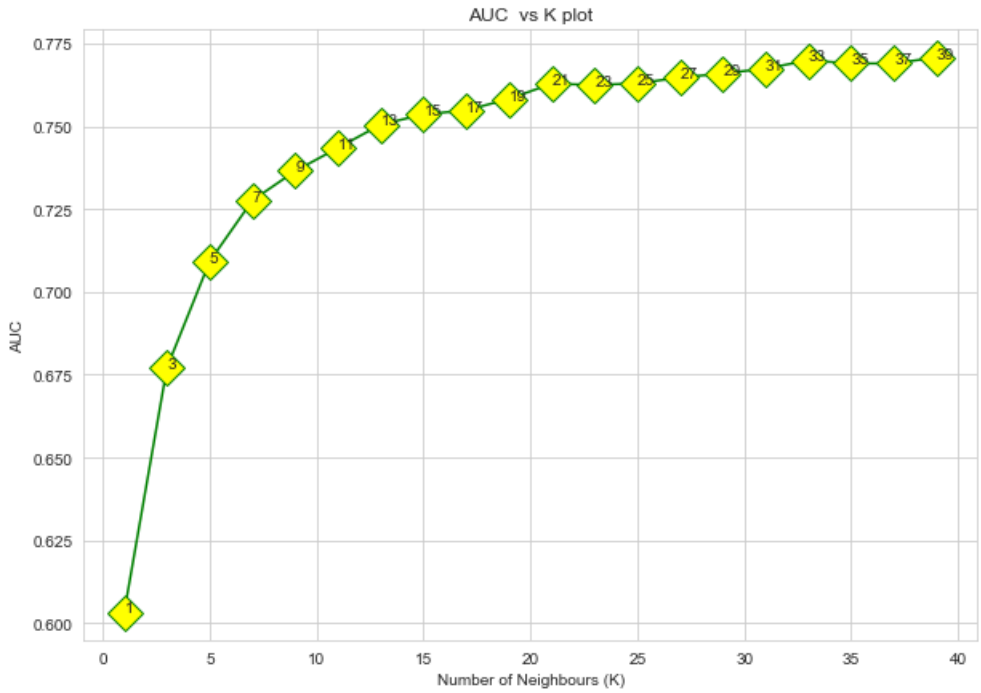
In [154]:

```
# applying knn and finding optimal k

from sklearn.model_selection import cross_val_score
cv_list=[]
neighbors=range(1,40,2)
for K in neighbors:
    K_value = K
    neigh = KNeighborsClassifier(n_neighbors = K_value, weights='uniform', algorithm='kd_tree')
    scores=cross_val_score(neigh, X_train, y_train, cv=5, scoring='roc_auc') # 5 fold cross validat
ion
    cv_list.append(scores.mean())
```

In [155]:

```
# plot of k vs accuracy
plt.figure(figsize = (10,7))
```

```
sns.set_style("whitegrid");

plt.plot(neighbors, cv_list, color='green',  marker='D',markerfacecolor='yellow', markersize=15)
ind=0
for val in neighbors:
    plt.annotate('%s' % val,xy=(val,cv_list[ind]),xycoords='data')
    ind=ind+1

plt.title("AUC  vs K plot")
plt.xlabel('Number of Neighbours (K)')
plt.ylabel('AUC')
plt.show()
```



In [156]:

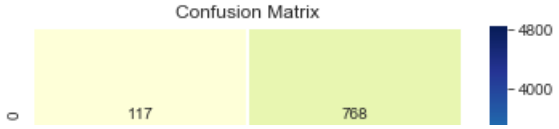
```
# optimal k
opt_k=15
val=math.floor(opt_k/2)+1
print("Maximum AUC score is: ",cv_list[val],"for k=",opt_k)
```

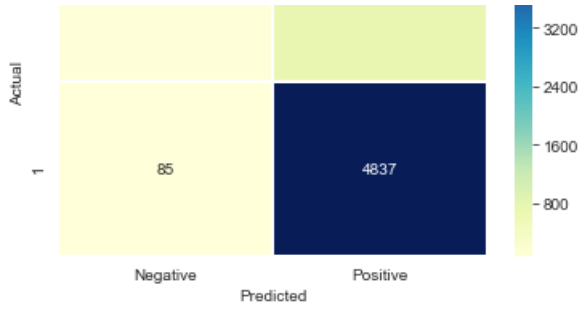
```
Maximum AUC score is:  0.7547944891542069 for k= 15
```

Confusion Matrix

In [157]:

```
import seaborn as sb
from sklearn.metrics import confusion_matrix

knn_optimal = KNeighborsClassifier(n_neighbors=opt_k , algorithm = 'kd_tree')
knn_optimal.fit(X_train, y_train)

conf_matrix = confusion_matrix(y_test,knn_optimal.predict(X_test))
df_cmatrix = pd.DataFrame(conf_matrix, columns=['Negative', 'Positive'])
sb.heatmap(df_cmatrix, annot=True, fmt='d',cmap="YlGnBu", linewidths=1.4)

plt.title("Confusion Matrix")
plt.xlabel("Predicted")
plt.ylabel("Actual")
plt.show()
```





AUC CURVE

In [158]:

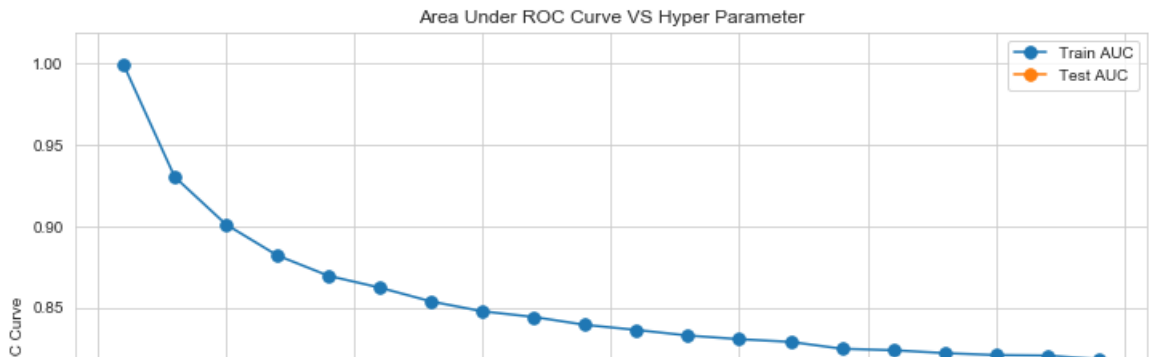
```
# calculating train_auc
neighbors=list(range(1,40,2))

train_auc_list=[]
for k in neighbors:
    neigh = KNeighborsClassifier(n_neighbors=k , algorithm = 'brute')
    neigh.fit(X_train, y_train)
    # predict probabilities
    probs = neigh.predict_proba(X_train)
    # keep probabilities for the positive outcome only
    probs = probs[:, 1]
    # calculate AUC
    train_auc = roc_auc_score(y_train, probs)
    train_auc_list.append(train_auc)
```

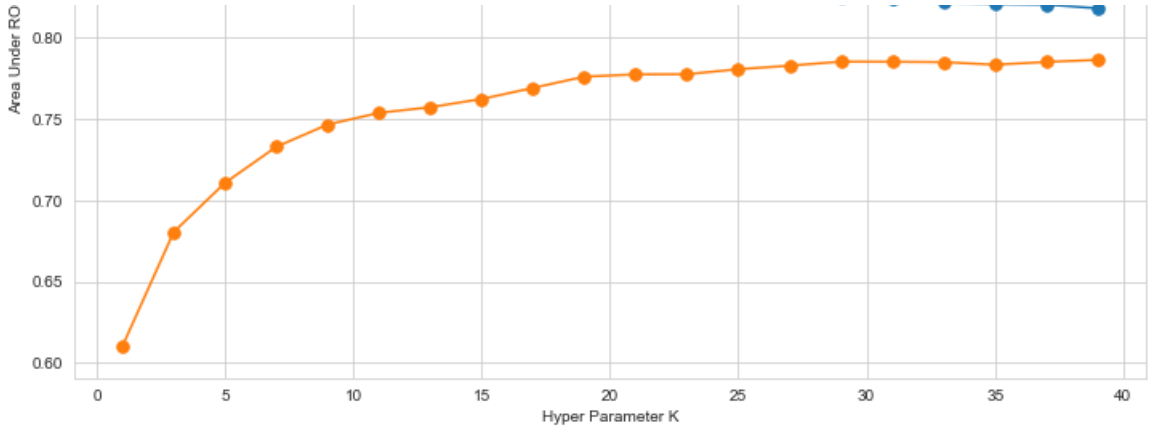
In [159]:

```
# calculating test_auc
neighbors=list(range(1,40,2))

test_auc_list=[]
for k in neighbors:
    neigh = KNeighborsClassifier(n_neighbors=k , algorithm = 'brute')
    neigh.fit(X_train, y_train)
    # predict probabilities
    probs = neigh.predict_proba(X_test)
    # keep probabilities for the positive outcome only
    probs = probs[:, 1]
    # calculate AUC
    test_auc = roc_auc_score(y_test, probs)
    test_auc_list.append(test_auc)
```

In [160]:

```
plt.figure(figsize = (12,8))
plt.plot(neighbors, train_auc_list, marker='.',label="Train AUC",markersize=15)
plt.plot(neighbors, test_auc_list, marker='.',label="Test AUC",markersize=15)
plt.legend()
plt.xlabel("Hyper Parameter K")
plt.ylabel("Area Under ROC Curve")
plt.title("Area Under ROC Curve VS Hyper Parameter")
plt.show()
```





In [161]:

```
knn_optimal = KNeighborsClassifier(n_neighbors=opt_k , algorithm = 'kd_tree')
knn_optimal.fit(X_train, y_train)
```

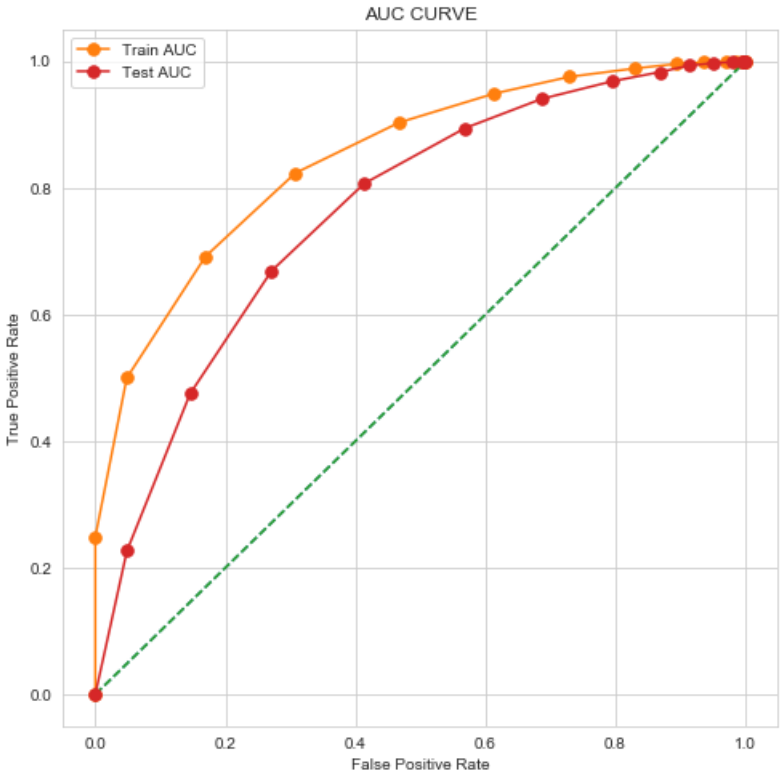
Out[161]:

```
KNeighborsClassifier(algorithm='kd_tree', leaf_size=30, metric='minkowski',
           metric_params=None, n_jobs=None, n_neighbors=15, p=2,
           weights='uniform')
```

In [162]:

```
from matplotlib import pyplot
from sklearn.metrics import roc_auc_score
from sklearn.metrics import roc_curve, auc

plt.figure(figsize = (8,8))
# predict probabilities
probs = knn_optimal.predict_proba(X_train)
# keep probabilities for the positive outcome only
probs = probs[:, 1]
# calculate AUC
train_auc = roc_auc_score(y_train, probs)
print('Train AUC: %f' % train_auc)
# calculate roc curve
fpr, tpr, thresholds = roc_curve(y_train, probs)
# plot no skill
pyplot.plot([0, 1], [0, 1], linestyle='--')
# plot the roc curve for the model
pyplot.plot(fpr, tpr, marker='.',label="Train AUC",markersize=15)


# predict probabilities
probs = knn_optimal.predict_proba(X_test)
# keep probabilities for the positive outcome only
probs = probs[:, 1]
# calculate AUC
test_auc = roc_auc_score(y_test, probs)
print('Test AUC: %f' % test_auc)
# calculate roc curve
fpr, tpr, thresholds = roc_curve(y_test, probs)
# plot no skill
pyplot.plot([0, 1], [0, 1], linestyle='--')
# plot the roc curve for the model
pyplot.plot(fpr, tpr, marker='.',label="Test AUC",markersize=15)

plt.legend()
plt.xlabel("False Positive Rate")
plt.ylabel("True Positive Rate")
plt.title("AUC CURVE")

# show the plot
pyplot.show()
```

```
Train AUC: 0.847733
Test AUC: 0.762348
```



Conclusion

In [163]:

```
tfidf_w2vec_kdtree_par=opt_k
tfidf_w2vec_kdtree_auc=np.round(test_auc,4)
print("Vectorizer: TFIDF W2VEC \t Model: KD TREE")
print("Best Hyper parameter: ",tfidf_w2vec_kdtree_par)
print("AUC: ",tfidf_w2vec_kdtree_auc)
```

```
Vectorizer: TFIDF W2VEC   Model: KD TREE
Best Hyper parameter:  15
AUC:  0.7623
```

[6] Conclusions

In [164]:

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

In [172]:

```
# Constructing a table to summerize all the above results

from prettytable import PrettyTable

x = PrettyTable()

x.field_names = ["Vectorizer", "Model", "Hyper Parameter", "AUC"]

x.add_row(["BOW", "Brute", bow_brute_par, bow_brute_auc])
x.add_row(["TFIDF", "Brute", tfidf_brute_par, tfidf_brute_auc])
x.add_row(["W2VEC", "Brute", w2vec_brute_par,w2vec_brute_auc])
x.add_row(["TFIDF W2VEC", "Brute", tfidf_w2vec_brute_par,tfidf_w2vec_bow_brute_auc])
x.add_row(["BOW", "KD Tree", bow_kdtree_par,bow_kdtree_auc])
x.add_row(["TFIDF", "KD Tree", tfidf_kdtree_par, tfidf_kdtree_auc])
x.add_row(["W2VEC", "KD Tree", w2vec_kdtree_par, w2vec_kdtree_auc])
x.add_row(["TFIDF W2VEC", "KD Tree", tfidf_w2vec_kdtree_par,tfidf_w2vec_kdtree_auc])
print("="*100)
print('Conclusions'.center(100,' '))
```

```
print("Conclusions".center(100," "))
print("="*100)
print(x.get_string(border=True, padding_width=7))
# print("#"*100)
```

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

                                            Conclusions
====================================================================================================

+-------------------------+---------------------+-----------------------------+--------------------
|       Vectorizer        |        Model        |       Hyper Parameter       |        AUC
+-------------------------+---------------------+-----------------------------+--------------------
|           BOW           |        Brute        |              31             |       0.6752
|          TFIDF          |        Brute        |              29             |       0.6922
|          W2VEC          |        Brute        |              15             |       0.8049
|       TFIDF W2VEC       |        Brute        |              31             |       0.7905
|           BOW           |       KD Tree       |              15             |       0.7297
|          TFIDF          |       KD Tree       |              31             |       0.7386
|          W2VEC          |       KD Tree       |              15             |       0.8017
|       TFIDF W2VEC       |       KD Tree       |              15             |       0.7623
+-------------------------+---------------------+-----------------------------+--------------------
```

References:

1. http://dataaspirant.com/2016/12/30/k-nearest-neighbor-implementation-scikit-learn/
2. https://stackabuse.com/k-nearest-neighbors-algorithm-in-python-and-scikit-learn/
3. https://kevinzakka.github.io/2016/07/13/k-nearest-neighbor/
4. https://stackoverflow.com/questions/15148149/annotate-a-plot-using-matplotlib
5. https://scikit-learn.org/stable/auto_examples/model_selection/plot_confusion_matrix.html
6. https://scikit-learn.org/stable/modules/generated/sklearn.metrics.roc_curve.html
7. https://machinelearningmastery.com/roc-curves-and-precision-recall-curves-for-classification-in-python/
8. http://zetcode.com/python/prettytable/