Amazon_Fine_Food_Reviews_Analysis_KNN

January 18, 2019

1 Amazon Fine Food Reviews Analysis

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

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

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

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

Attribute Information:

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

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

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

2 [1]. Reading Data

2.1 [1.1] Loading the data

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

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

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

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


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

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

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

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

from tqdm import tqdm
import os
```

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 point
# you can change the number to any other number based on your computing power

# filtered_data = pd.read_sql_query(""" SELECT * FROM Reviews WHERE Score != 3 LIMIT 5
```

```
filtered_data = pd.read_sql_query(""" SELECT * FROM Reviews WHERE Score != 3 LIMIT 5000

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

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

```
Number of data points in our data (50000, 10)
```

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

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

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

```
In [3]: display = pd.read_sql_query("""
        SELECT UserId, ProductId, ProfileName, Time, Score, Text, COUNT(*)
        FROM Reviews
        GROUP BY UserId
        HAVING COUNT(*)>1
        """, con)
```

```
In [4]: print(display.shape)
        display.head()
```

```
(80668, 7)
```

```
Out[4]:                UserId   ProductId             ProfileName        Time  Score  \
        0  #oc-R115TNMSPFT9I7  B007Y59HVM                 Breyton  1331510400      2
```

```
1  #oc-R11D9D7SHXIJB9  B005HG9ET0  Louis E. Emory "hoppy"  1342396800      5
2  #oc-R11DNU2NBKQ23Z  B007Y59HVM        Kim Cieszykowski  1348531200      1
3  #oc-R1105J5ZVQE25C  B005HG9ET0            Penguin Chick  1346889600      5
4  #oc-R12KPBODL2B5ZD  B0070SBE1U   Christopher P. Presta  1348617600      1

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

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

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

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

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

Out[6]: 393063

3 [2] Exploratory Data Analysis

3.1 [2.1] Data Cleaning: Deduplication

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

In [7]:
```
display= pd.read_sql_query("""
SELECT *
FROM Reviews
WHERE Score != 3 AND UserId="AR5J8UI46CURR"
ORDER BY ProductID
""", con)
display.head()
```

Out[7]:
```
       Id   ProductId         UserId      ProfileName  HelpfulnessNumerator  \
0   78445  B000HDL1RQ  AR5J8UI46CURR  Geetha Krishnan                     2
1  138317  B000HDOPYC  AR5J8UI46CURR  Geetha Krishnan                     2
2  138277  B000HDOPYM  AR5J8UI46CURR  Geetha Krishnan                     2
3   73791  B000HDOPZG  AR5J8UI46CURR  Geetha Krishnan                     2
4  155049  B000PAQ75C  AR5J8UI46CURR  Geetha Krishnan                     2

   HelpfulnessDenominator  Score        Time  \
0                       2      5  1199577600
```

```
1                        2       5  1199577600
2                        2       5  1199577600
3                        2       5  1199577600
4                        2       5  1199577600

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

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

As it can be seen above that same user has multiple reviews with same values for HelpfulnessNumerator, HelpfulnessDenominator, Score, Time, Summary and Text and on doing analysis it was found that ProductId=B000HDOPZG was Loacker Quadratini Vanilla Wafer Cookies, 8.82-Ounce Packages (Pack of 8) ProductId=B000HDL1RQ was Loacker Quadratini Lemon Wafer Cookies, 8.82-Ounce Packages (Pack of 8) and so on

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

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

```
In [8]: #Sorting data according to ProductId in ascending order
        sorted_data=filtered_data.sort_values('ProductId', axis=0, ascending=True, inplace=Fals
```

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

```
Out[9]: (46072, 10)
```

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

```
Out[10]: 92.144
```

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

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

         display.head()
```

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

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

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

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

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

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

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

```
(46071, 10)
```

```
Out[13]: 1    38479
         0     7592
         Name: Score, dtype: int64
```

4 [3] Preprocessing

4.1 [3.1]. Preprocessing Review Text

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

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

1. Begin by removing the html tags

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

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

In [14]:
```
# printing some random reviews
sent_0 = final['Text'].values[0]
print(sent_0)
print("="*50)

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

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

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

```
My dogs loves this chicken but its a product from China, so we wont be buying it anymore.  Its
==================================================
this is yummy, easy and unusual. it makes a quick, delicous pie, crisp or cobbler. home made is
==================================================
Great flavor, low in calories, high in nutrients, high in protein! Usually protein powders are
==================================================
For those of you wanting a high-quality, yet affordable green tea, you should definitely give t
==================================================
```

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)
```

```
My dogs loves this chicken but its a product from China, so we wont be buying it anymore.  Its
```

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

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

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

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

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

```
My dogs loves this chicken but its a product from China, so we wont be buying it anymore.  Its
==================================================
this is yummy, easy and unusual. it makes a quick, delicous pie, crisp or cobbler. home made is
==================================================
Great flavor, low in calories, high in nutrients, high in protein! Usually protein powders are
==================================================
For those of you wanting a high-quality, yet affordable green tea, you should definitely give t
```

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)
    phrase = re.sub(r"wont","will not",phrase) # in some words aposthepe is missing
    phrase = re.sub(r"its","it is",phrase)
    phrase = re.sub(r"Its","It is",phrase)
    phrase = re.sub(r"isnt","is 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_0 = decontracted(sent_0)
print(sent_0)
print("="*50)
```

```
My dogs loves this chicken but it is a product from China, so we will not be buying it anymore
==================================================
```

In [19]:
```
#remove words with numbers python: https://stackoverflow.com/a/18082370/4084039
sent_4900 = re.sub("\S*\d\S*", "", sent_4900).strip()
print(sent_4900)
```

```
For those of you wanting a high-quality, yet affordable green tea, you should definitely give t
```

In [20]:
```
#remove special character: https://stackoverflow.com/a/5843547/4084039
sent_4900 = re.sub('[^A-Za-z0-9]+', ' ', sent_4900)
print(sent_4900)
```

```
For those of you wanting a high quality yet affordable green tea you should definitely give thi
```

In [21]:
```
# https://gist.github.com/sebleier/554280
# we are removing the words from the stop words list: 'no', 'nor', 'not'
# <br /><br /> ==> after the above steps, we are getting "br br"
# we are including them into stop words list
# instead of <br /> if we have <br/> these tags would have revmoved in the 1st step

stopwords= set(['br', 'the', 'i', 'me', 'my', 'myself', 'we', 'our', 'ours', 'ourselve
            "you'll", "you'd", 'your', 'yours', 'yourself', 'yourselves', 'he', 'him'
            'she', "she's", 'her', 'hers', 'herself', 'it', "it's", 'its', 'itself',
            'theirs', 'themselves', 'what', 'which', 'who', 'whom', 'this', 'that', "
            'am', 'is', 'are', 'was', 'were', 'be', 'been', 'being', 'have', 'has', 'h
            'did', 'doing', 'a', 'an', 'the', 'and', 'but', 'if', 'or', 'because', 'as
            'at', 'by', 'for', 'with', 'about', 'against', 'between', 'into', 'through
            'above', 'below', 'to', 'from', 'up', 'down', 'in', 'out', 'on', 'off', 'o
            'then', 'once', 'here', 'there', 'when', 'where', 'why', 'how', 'all', 'an
            'most', 'other', 'some', 'such', 'only', 'own', 'same', 'so', 'than', 'too
            's', 't', 'can', 'will', 'just', 'don', "don't", 'should', "should've", 'n
            've', 'y', 'ain', 'aren', "aren't", 'couldn', "couldn't", 'didn', "didn't"
            "hadn't", 'hasn', "hasn't", 'haven', "haven't", 'isn', "isn't", 'ma', 'mig
            "mustn't", 'needn', "needn't", 'shan', "shan't", 'shouldn', "shouldn't",
            'won', "won't", 'wouldn', "wouldn't"])
```

In [22]:
```
# Combining all the above stundents
import itertools
```

```
from tqdm import tqdm
preprocessed_reviews = []
# tqdm is for printing the status bar
for sentence in tqdm(final['Text'].values):
    sentence = re.sub(r"http\S+", "", sentence)
    sentence = BeautifulSoup(sentence, 'lxml').get_text()
    sentence = decontracted(sentence)
    sentence = re.sub("\S*\d\S*", "", sentence).strip()
    sentence = re.sub('[^A-Za-z]+', ' ', sentence)

    #https://www.analyticsvidhya.com/blog/2014/11/text-data-cleaning-steps-python/
    # This removes words such as aawwww or happpyyy or awsooommmee etc
    sentence = ''.join(''.join(s)[:2] for _, s in itertools.groupby(sentence))

    # https://gist.github.com/sebleier/554280
    sentence = ' '.join(e.lower() for e in sentence.split() if e.lower() not in stopwo
    preprocessed_reviews.append(sentence.strip())
```

```
100%|| 46071/46071 [00:36<00:00, 1271.93it/s]
```

In [23]:
```
smpl = " aaaaww aaaww aaa aawwww"
sent = ''.join(''.join(s)[:2] for _, s in itertools.groupby(smpl))
print(sent)
```

```
 aaww aaww aa aaww
```

In [24]:
```
preprocessed_reviews[4900]
```

Out[24]:
```
'wanting high quality yet affordable green tea definitely give one try let first star
```

[3.2] Preprocessing Review Summary

In [25]:
```
## Similartly performing preprocessing for review summary also.

preprocessed_summary=[]

for sent in tqdm(final['Summary'].values):
    sent = re.sub(r"http\S+","",sent)
    sent = BeautifulSoup(sent,'lxml').get_text()
    sent = decontracted(sent)
    sent = re.sub(r"\S+\d\S+","",sent).strip()
    sent = re.sub(r"[^A-Za-z0]+"," ",sent)

    #https://www.analyticsvidhya.com/blog/2014/11/text-data-cleaning-steps-python/
    # This removes words such as aawwww or happpyyy or awsooommmee etc
    sent = ''.join(''.join(s)[:2] for _, s in itertools.groupby(sent))
```

```
            # https://gist.github.com/sebleier/554280
            sent = ' '.join(w.lower() for w in sent.split() if w.lower() not in stopwords)
            preprocessed_summary.append(sent.strip())
```

```
100%|| 46071/46071 [00:17<00:00, 2563.52it/s]
```

In [26]:
```
preprocessed_summary[4900]
```

Out[26]:
```
'truly well balanced green tea'
```

In [27]:
```
final['Summary'].values[4900]
```

Out[27]:
```
'A Truly Well-Balanced Green Tea'
```

In [28]:
```
# Removing those words which are of lenght 2
# This will remove non relevant words then we will perform featurization
cleaned_reviews = []
for sent in tqdm(preprocessed_reviews):
    sentence = ' '.join(w for w in sent.split() if len(w)>2)
    cleaned_reviews.append(sentence.strip())
```

```
100%|| 46071/46071 [00:00<00:00, 69422.22it/s]
```

In [29]:
```
print(cleaned_reviews[0])
```

```
dogs loves chicken product china not buying anymore hard find chicken products made usa one no
```

In [30]:
```
final["Cleaned_review"] = cleaned_reviews
final.head(5)
```

Out[30]:
```
          Id   ProductId          UserId        ProfileName  \
22620  24750  2734888454  A13ISQV0U9GZIC           Sandikaye
22621  24751  2734888454  A1C298ITT645B6  Hugh G. Pritchard
2546    2774  B00002NCJC  A196AJHU9EASJN       Alex Chaffee
2547    2775  B00002NCJC  A13RRPGE79XFFH           reader48
1145    1244  B00002Z754  A3B8RCEI0FXFI6          B G Chase

       HelpfulnessNumerator  HelpfulnessDenominator  Score        Time  \
22620                     1                       1      0  1192060800
22621                     0                       0      1  1195948800
2546                      0                       0      1  1282953600
2547                      0                       0      1  1281052800
1145                     10                      10      1   962236800

                          Summary  \
22620               made in china
22621           Dog Lover Delites
```

```
2546                    thirty bucks?
2547                     Flies Begone
1145   WOW Make your own 'slickers' !

                                                          Text  \
22620  My dogs loves this chicken but its a product f...
22621  Our dogs just love them.  I saw them in a pet ...
2546   Why is this $[...] when the same product is av...
2547   We have used the Victor fly bait for 3 seasons...
1145   I just received my shipment and could hardly w...

                                                Cleaned_review
22620  dogs loves chicken product china not buying an...
22621  dogs love saw pet store tag attached regarding...
2546   product available victor traps unreal course t...
2547   used victor fly bait seasons not beat great pr...
1145   received shipment could hardly wait try produc...
```

5 [4] Featurization

5.1 [4.1] BAG OF WORDS

In [31]:
```
#BoW
count_vect1 = CountVectorizer() #in scikit-learn
count_vect1.fit(cleaned_reviews)
print("some feature names ", count_vect1.get_feature_names()[:10])
print('='*50)

final_counts = count_vect1.transform(cleaned_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  ['aaa', 'aaah', 'aaahh', 'aaaww', 'aachen', 'aadp', 'aaf', 'aafco', 'aah',
==================================================
the type of count vectorizer  <class 'scipy.sparse.csr.csr_matrix'>
the shape of out text BOW vectorizer  (46071, 38772)
the number of unique words  38772
```

5.2 [4.2] Bi-Grams and n-Grams.

In [33]:
```
#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/mod
```

```
# you can choose these numebrs min_df=10, max_features=5000, of your choice
count_vect2 = CountVectorizer(ngram_range=(1,2), min_df=5)
final_bigram_counts = count_vect2.fit_transform(cleaned_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_
bigram_features = count_vect2.get_feature_names()
print(bigram_features[:10])
```

```
the type of count vectorizer  <class 'scipy.sparse.csr.csr_matrix'>
the shape of out text BOW vectorizer  (46071, 59655)
the number of unique words including both unigrams and bigrams  59655
['aafco', 'aback', 'abandon', 'abandoned', 'abdominal', 'ability', 'ability buy', 'able', 'able
```

5.3 [4.3] TF-IDF

```
In [34]: # tfidf on unigrams
         tf_idf_vect1 = TfidfVectorizer()
         tf_idf_vect1.fit(cleaned_reviews)
         print("some sample features(unique words in the corpus)",tf_idf_vect1.get_feature_name
         print('='*50)

         final_tf_idf1 = tf_idf_vect1.transform(cleaned_reviews)
         print("the type of count vectorizer ",type(final_tf_idf1))
         print("the shape of out text TFIDF vectorizer ",final_tf_idf1.get_shape())
         print("the number of unique words including both unigrams and bigrams ", final_tf_idf1
```

```
some sample features(unique words in the corpus) ['aaa', 'aaah', 'aaahh', 'aaaww', 'aachen', 'a
==================================================
the type of count vectorizer  <class 'scipy.sparse.csr.csr_matrix'>
the shape of out text TFIDF vectorizer  (46071, 38772)
the number of unique words including both unigrams and bigrams  38772
```

```
In [38]: # Tfidf on bigrams
         tf_idf_vect2 = TfidfVectorizer(ngram_range=(1,2),min_df=3)
         tf_idf_vect2.fit(cleaned_reviews)
         print("some sample features(unique words in the corpus)",tf_idf_vect2.get_feature_name
         print('='*50)

         final_tf_idf2 = tf_idf_vect2.transform(cleaned_reviews)
         print("the type of count vectorizer ",type(final_tf_idf2))
         print("the shape of out text TFIDF vectorizer ",final_tf_idf2.get_shape())
         print("the number of unique words including both unigrams and bigrams ", final_tf_idf2
```

```
some sample features(unique words in the corpus) ['aafco', 'aback', 'abandon', 'abandoned', 'ab
==================================================
the type of count vectorizer  <class 'scipy.sparse.csr.csr_matrix'>
```

```
the shape of out text TFIDF vectorizer  (46071, 115699)
the number of unique words including both unigrams and bigrams  115699
```

5.4 [4.4] Word2Vec

In [39]:
```
# Train your own Word2Vec model using your own text corpus
i=0
list_of_sentence=[]
for sentence in tqdm(cleaned_reviews):
    list_of_sentence.append(sentence.split())
```

```
100%|| 46071/46071 [00:00<00:00, 133611.51it/s]
```

In [40]:
```
outfile = open("list_of_sentence","wb")
pickle.dump(list_of_sentence,outfile)
outfile.close()
```

In [42]:
```
# 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
# 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_sentence,min_count=3,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.b
        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, t
```

```
[('awesome', 0.8226403594017029), ('fantastic', 0.8149124979972839), ('good', 0.808935999870300
==================================================
[('nastiest', 0.7852265238761902), ('greatest', 0.7365906238555908), ('best', 0.732491374015808
```

```
In [43]: 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  16667
sample words  ['dogs', 'loves', 'chicken', 'product', 'china', 'not', 'buying', 'anymore', 'ha
```

5.5 [4.4.1] Converting text into vectors using Avg W2V, TFIDF-W2V

[4.4.1.1] Avg W2v

```
In [44]: # 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 list_of_sentence: # for each review/sentence
             sent_vec = np.zeros(50) # as word vectors are of zero length 50, you might need to
             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]))
```

```
46071
50
```

[4.4.1.2] TFIDF weighted W2v

```
In [161]: # S = ["abc def pqr", "def def def abc", "pqr pqr def"]
          model = TfidfVectorizer()
          tf_idf_matrix = model.fit_transform(cleaned_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 [142]: # 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 = tfid

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

```
In [47]: print(len(tfidf_sent_vectors))
         print(len(tfidf_sent_vectors[0]))
```

```
46071
50
```

6 [5] Assignment 3: KNN

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)

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.

```
</li>
<li><font color='red'>SET 6:</font>Review text, preprocessed one converted into vector
<pre>
```

```
            tf_idf_vect = TfidfVectorizer(min_df=10, max_features=500)
            tf_idf_vect.fit(preprocessed_reviews)
        </pre>
        </li>
        <li><font color='red'>SET 3:</font>Review text, preprocessed one converted into vectors
        <li><font color='red'>SET 4:</font>Review text, preprocessed one converted into vectors
    </ul>
</li>
<br>
<li><strong>The hyper paramter tuning(find best K)</strong>
    <ul>
<li>Find the best hyper parameter which will give the maximum <a href='https://www.appliedaicou
<li>Find the best hyper paramter using k-fold cross validation or simple cross validation data
<li>Use gridsearch cv or randomsearch cv or you can also write your own for loops to do this ta
    </ul>
</li>
<br>
<li>
<strong>Representation of results</strong>
    <ul>
<li>You need to plot the performance of model both on train data and cross validation data for
<img src='train_cv_auc.JPG' width=300px></li>
<li>Once after you found the best hyper parameter, you need to train your model with it, and fi
<img src='train_test_auc.JPG' width=300px></li>
<li>Along with plotting ROC curve, you need to print the <a href='https://www.appliedaicourse.c
<img src='confusion_matrix.png' width=300px></li>
    </ul>
</li>
<br>
<li><strong>Conclusion</strong>
    <ul>
<li>You need to summarize the results at the end of the notebook, summarize it in the table for
    <img src='summary.JPG' width=400px>
</li>
    </ul>
```

Note: Data Leakage

1. There will be an issue of data-leakage if you vectorize the entire data and then split it into train/cv/test.
2. To avoid the issue of data-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.

6.1 [5.1] Applying KNN brute force

6.1.1 [5.1.1] Applying KNN brute force on BOW, SET 1

```
In [82]: from sklearn.cross_validation import train_test_split

         # cleaned_reviews contains all the required reviews
         # Splitting cleaned_reviews into train and test dataset

         X = cleaned_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(len(X_train),len(Y_train),len(X_test),len(Y_test))
```

```
C:\Users\rites\Anaconda3\lib\site-packages\sklearn\cross_validation.py:41: DeprecationWarning:
  "This module will be removed in 0.20.", DeprecationWarning)


32249 32249 13822 13822
```

```
In [84]: # Now we will vectorize train and test datasets separately using BagofWords
         # Use fit_transform to vectorize train dataset and transform to vectorize test datase
         count_vect2 = CountVectorizer()
         X_train = count_vect2.fit_transform(X_train)
         X_test = count_vect2.transform(X_test)
         print(X_train.shape,X_test.shape)
```

```
(32249, 32938) (13822, 32938)
```

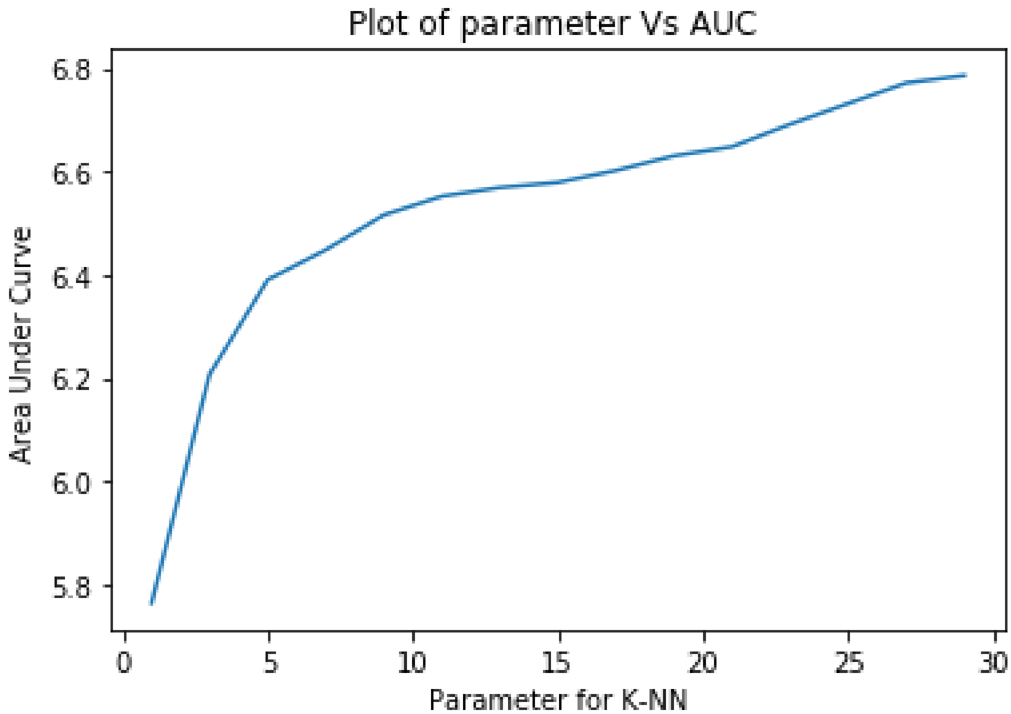
```
In [89]: # We will do time based splitting and do 10 fold cross validation
         # This is done as reviews keeps changing with time and hence time based splitting is
         from sklearn.model_selection import TimeSeriesSplit
         from sklearn.metrics import roc_auc_score
         from sklearn.metrics import accuracy_score

         # Time series object
         tscv = TimeSeriesSplit(n_splits=10)

         from sklearn.neighbors import KNeighborsClassifier
         from sklearn.metrics import accuracy_score

         param_list = [1,3,5,7,9,11,13,15,17,19,21,23,25,27,29]
         acc_list = []
         auc_list = []

         for k in range(1,30,2):
             # KNN Classifier
```

```
            clf = KNeighborsClassifier(n_neighbors=k,algorithm='brute',leaf_size=30)
            i=0
            acc=0.0
            auc=0.0
            for train_index,test_index in tscv.split(X_train):
                x_train = X_train[0:train_index[-1]][:] # row 0 to train_index(excluding)
                y_train = Y_train[0:train_index[-1]][:] # row 0 to train_index(excluding)
                x_test = X_train[train_index[-1]:test_index[-1]][:] # row from train_index to
                y_test = Y_train[train_index[-1]:test_index[-1]][:] # row from train_index to

                clf.fit(x_train,y_train)

                predict_y = clf.predict(x_test)
                predict_probab = clf.predict_proba(x_test)[:,1]
                i += 1
                acc += accuracy_score(y_test,predict_y,normalize=True) * float(100)
                auc += roc_auc_score(y_test,predict_probab)

            acc_list.append(acc)
            auc_list.append(auc)
            print("Cross Validation Accuracy for k = {:d} is {:.2f}% and auc is {:.2f}".forma
```

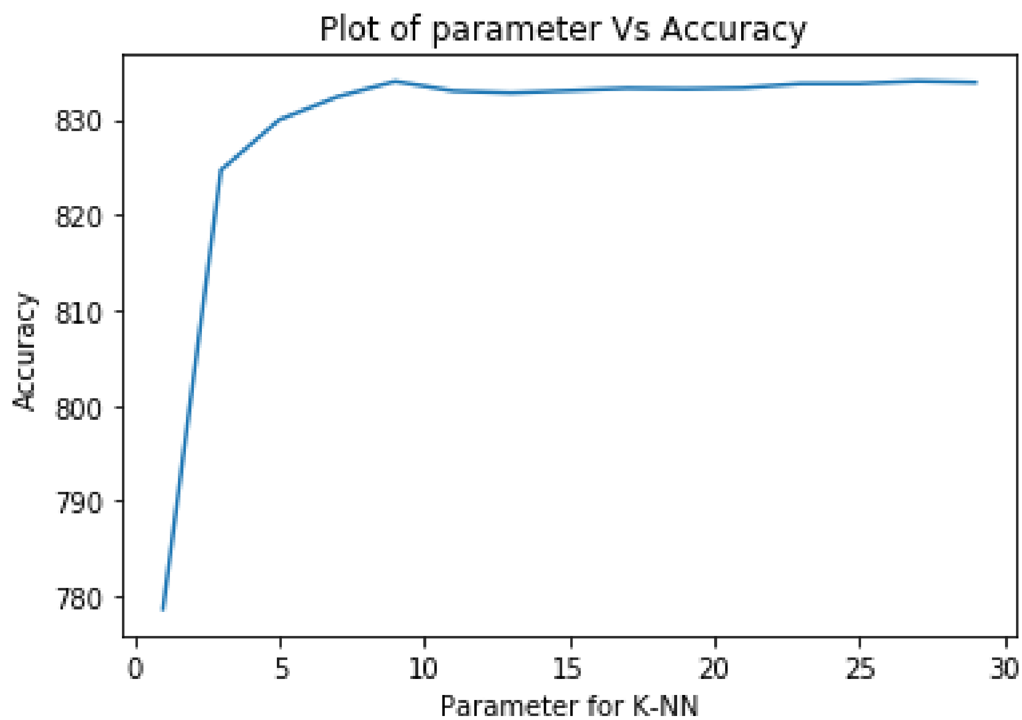
```
Cross Validation Accuracy for k = 1 is 77.87% and auc is 0.58
Cross Validation Accuracy for k = 3 is 82.47% and auc is 0.62
Cross Validation Accuracy for k = 5 is 83.00% and auc is 0.64
Cross Validation Accuracy for k = 7 is 83.24% and auc is 0.64
Cross Validation Accuracy for k = 9 is 83.40% and auc is 0.65
Cross Validation Accuracy for k = 11 is 83.30% and auc is 0.66
Cross Validation Accuracy for k = 13 is 83.28% and auc is 0.66
Cross Validation Accuracy for k = 15 is 83.30% and auc is 0.66
Cross Validation Accuracy for k = 17 is 83.33% and auc is 0.66
Cross Validation Accuracy for k = 19 is 83.32% and auc is 0.66
Cross Validation Accuracy for k = 21 is 83.33% and auc is 0.66
Cross Validation Accuracy for k = 23 is 83.37% and auc is 0.67
Cross Validation Accuracy for k = 25 is 83.37% and auc is 0.67
Cross Validation Accuracy for k = 27 is 83.40% and auc is 0.68
Cross Validation Accuracy for k = 29 is 83.39% and auc is 0.68
```

In [90]:
```
import matplotlib.pyplot as plt

# Plotting graph of auc and parameter

plt.plot(param_list,auc_list)
plt.xlabel("Parameter for K-NN")
plt.ylabel("Area Under Curve")
```

```
plt.title("Plot of parameter Vs AUC ")
plt.show()
```




```
In [91]: # Plotting graph of accuracy and parameter

         plt.plot(param_list,acc_list)
         plt.xlabel("Parameter for K-NN")
         plt.ylabel("Accuracy")
         plt.title("Plot of parameter Vs Accuracy ")
         plt.show()
```



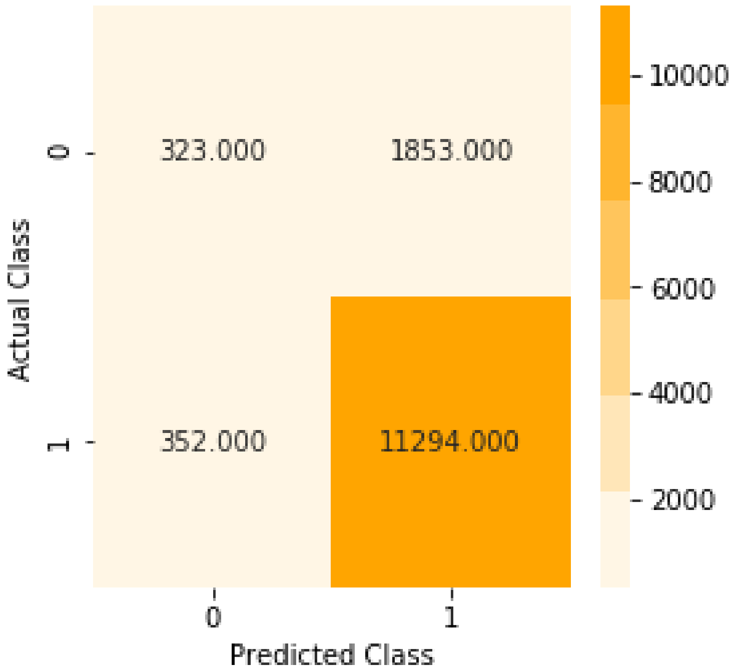

```
In [92]: # Training final model on best auc and taking k = 21

         final_clf = KNeighborsClassifier(n_neighbors=21,algorithm='brute',leaf_size=30)

         final_clf.fit(X_train,Y_train)

         predict_y = final_clf.predict(X_test)
         predict_probab = final_clf.predict_proba(X_test)[:,1] # This returns only probability

         acc = accuracy_score(Y_test,predict_y,normalize=True)* float(100)
         auc = roc_auc_score(Y_test,predict_probab)
         print("Final Accuracy is {:.2f}% and auc is {:.2f}".format(acc,auc))

Final Accuracy is 84.05% and auc is 0.67


In [53]: outfile = open("BOW_burte","wb")
         pickle.dump(final_clf,outfile)
         outfile.close()

In [79]: # Plotting confusion matrix of this model
         from sklearn.metrics import confusion_matrix
         import seaborn as sns
         import matplotlib.pyplot as plt
```

```
def confusion_matrix_plot(y_test,predict_y):
    C = confusion_matrix(y_test,predict_y) # This matrix contains true +ve , false +v

    # Calculating precision matrix

    P = (C/C.sum(axis=0))
    # Explaination
    # axis = 0 will calculate sum about columns and divide each element in that colum
    # C = [1,4
    #      2,7]
    # C.sum(axis=1) is [3,11]
    # P = [1/3 , 4/11
    #      2/3 , 3/11]

    # Calculating recall matrix

    R = (((C.T)/(C.sum(axis=1))).T)
    # Explaination
    # axis = 1 will calculate sum about rows and divide all the elements in that row 
    # C = [1 , 4
    #      2 , 7]
    # C.T will be [1 , 2
    #              4 , 7]
    # C.sum(axis=0) is [5,9]
    # (C.T)/(C.sum(axis=1)) will be [1/5 , 4/5
    #                                2/9 , 7/9]

    labels = [0,1] # This list contains class labels

    # cmap object which contains color code
    cmap = sns.light_palette("orange")

    #Plotting confusion matrix
    print("=========== Confusion matrix ==========")
    plt.figure(figsize=(4,4))
    sns.heatmap(C,annot=True,cmap=cmap,fmt=".3f",xticklabels=labels,yticklabels=label
    plt.xlabel("Predicted Class")
    plt.ylabel("Actual Class")
    plt.show()

    # Plotting Precision confusion matrix
    print("=========== Precision Matrix ==========")
    plt.figure(figsize=(4,4))
    sns.heatmap(P,annot=True,cmap=cmap,fmt=".3f",xticklabels=labels,yticklabels=label
    plt.xlabel("Predicted Class")
    plt.ylabel("Actual Class")
    plt.show()
```

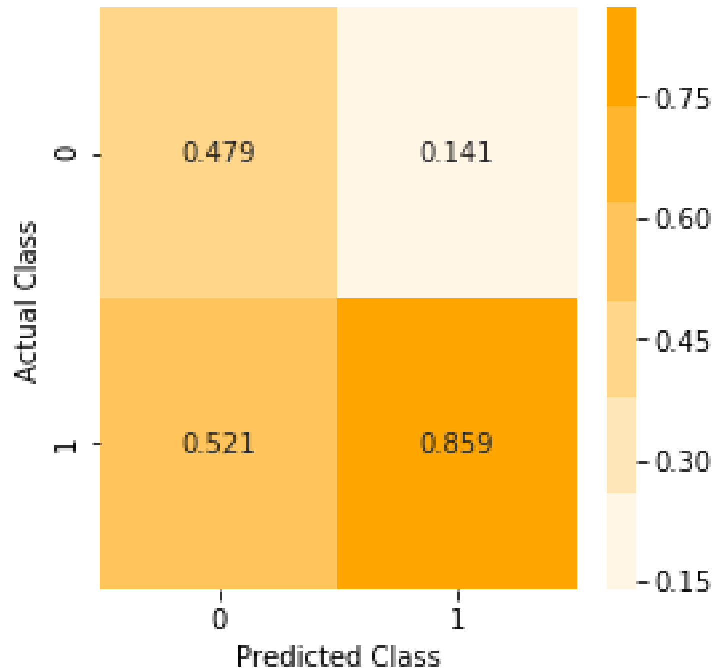
```
    # Plotting Recall Matrix
    print("=========== Recall Matrix =============")
    plt.figure(figsize=(4,4))
    sns.heatmap(R,annot=True,cmap=cmap,fmt=".3f",xticklabels=labels,yticklabels=labels
    plt.xlabel("Predicted Class")
    plt.ylabel("Actual Class")
    plt.show()
```

```
In [93]: # Calling confusion_matrix_plot

         confusion_matrix_plot(Y_test,predict_y)
```

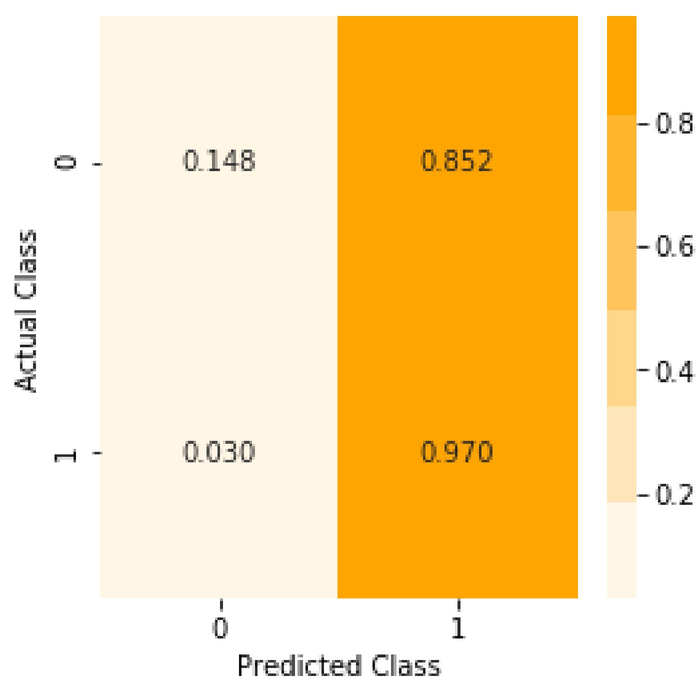
```
=========== Confusion matrix ==========
```




```
=========== Precision Matrix ==========
```



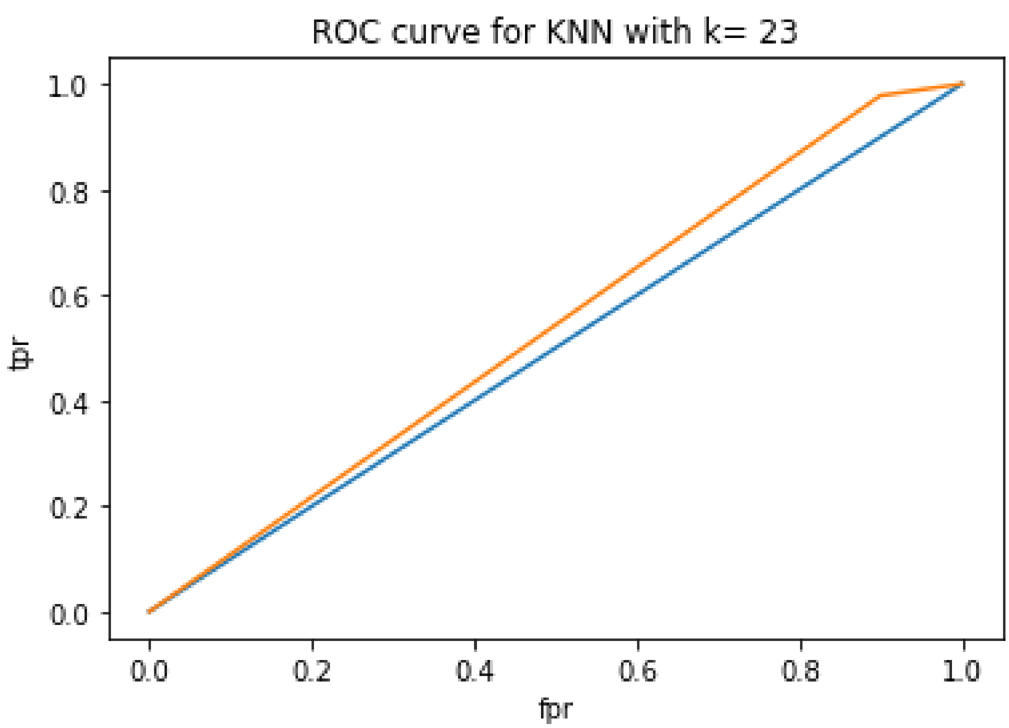

```
=========== Recall Matrix ============
```



In [94]:
```
from sklearn.metrics import classification_report

print(classification_report(Y_test,predict_y))
```

```
             precision    recall  f1-score   support

          0       0.48      0.15      0.23      2176
          1       0.86      0.97      0.91     11646

avg / total       0.80      0.84      0.80     13822
```

In [58]:
```
# Plotting ROC Curve
from sklearn.metrics import roc_curve

fpr,tpr,threshold = roc_curve(Y_test,predict_y)

plt.plot([0,1],[0,1])
plt.plot(fpr,tpr)
plt.xlabel("fpr")
plt.ylabel("tpr")
plt.title("ROC curve for KNN with k= 23")
plt.show()
```



In [59]:
```
# Printing area under curve

print("Area under ROC curve is ={:.3f} ".format(roc_auc_score(Y_test,predict_probab)))
```

```
Area under ROC curve is =0.620
```

7 Observations

1. Precision and recall for +ve class is good but for -ve class is very bad .

7.0.1 [5.1.2] Applying KNN brute force on TFIDF

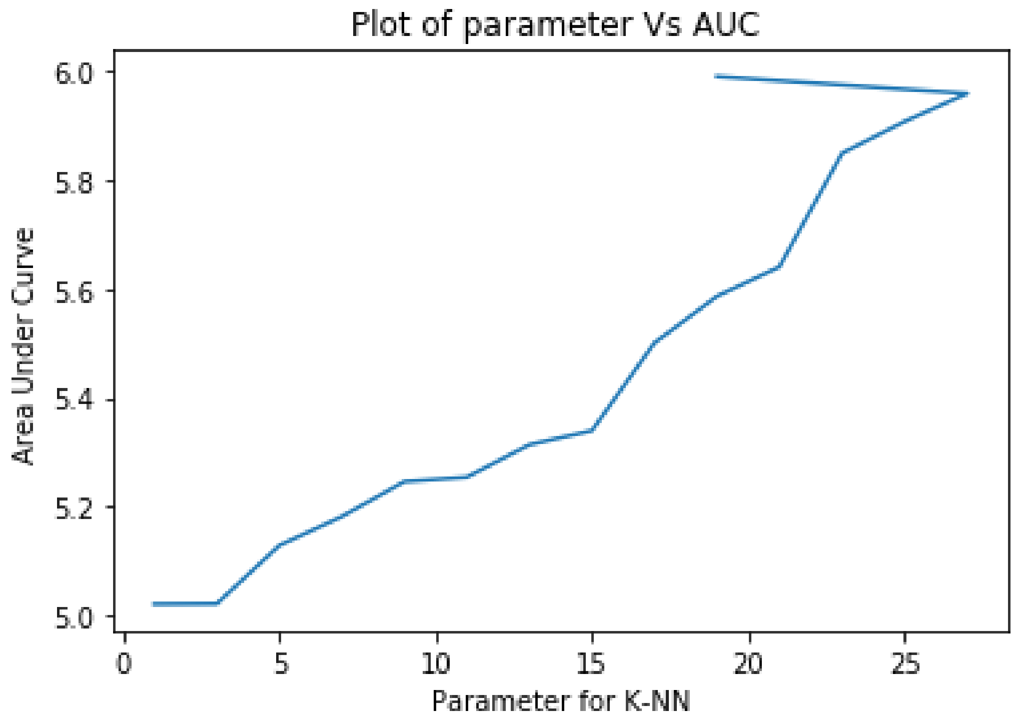
In [109]:
```
# In this section Tfidf will be used for vectorization
# Splitting datasets into train and test datasets

X_train,X_test,Y_train,Y_test = train_test_split(X,Y,test_size=0.3,random_state=42)

# Now we will vectorize train and test datasets separately using Tfidf
# Use fit_transform to vectorize train dataset and transform to vectorize test datas
X_train = tf_idf_vect2.fit_transform(X_train)
X_test = tf_idf_vect2.transform(X_test)
```

In [111]:
```
# Performing time series split cross validation

acc_list=[]
auc_list=[]

for k in range(1,30,2):
    # KNN Classifier
    clf = KNeighborsClassifier(n_neighbors=k,algorithm='brute',leaf_size=30)
    i=0
    acc=0.0
    auc=0.0
    for train_index,test_index in tscv.split(X_train):
        x_train = X_train[0:train_index[-1]][:] # row 0 to train_index(excluding)
        y_train = Y_train[0:train_index[-1]][:] # row 0 to train_index(excluding)
        x_test = X_train[train_index[-1]:test_index[-1]][:] # row from train_index t
        y_test = Y_train[train_index[-1]:test_index[-1]][:] # row from train_index t

        clf.fit(x_train,y_train)

        predict_y = clf.predict(x_test)
        predict_probab = clf.predict_proba(x_test)[:,1]
        i += 1
        acc += accuracy_score(y_test,predict_y,normalize=True) * float(100)
        auc += roc_auc_score(y_test,predict_probab)
```

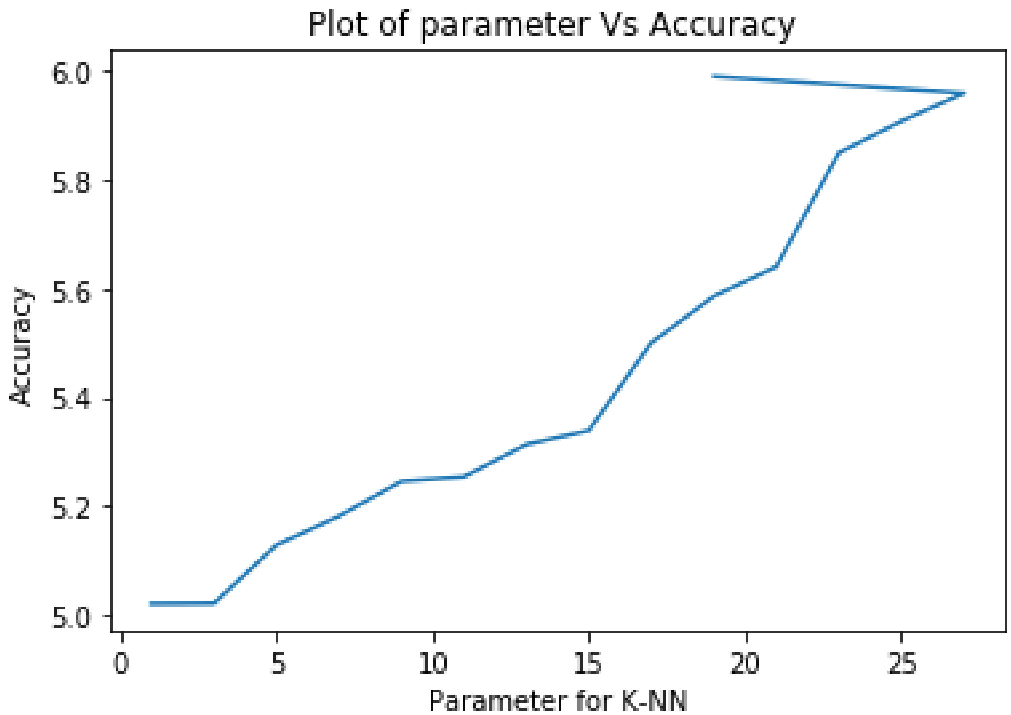
```
            auc_list.append(auc)
            acc_list.append(acc)
            print("Cross Validation Accuracy for k = {:d} is {:.2f}% and auc is {:.2f}".forma
```

```
Cross Validation Accuracy for k = 1 is 83.24% and auc is 0.50
Cross Validation Accuracy for k = 3 is 83.23% and auc is 0.50
Cross Validation Accuracy for k = 5 is 83.19% and auc is 0.51
Cross Validation Accuracy for k = 7 is 83.22% and auc is 0.52
Cross Validation Accuracy for k = 9 is 83.24% and auc is 0.52
Cross Validation Accuracy for k = 11 is 83.24% and auc is 0.53
Cross Validation Accuracy for k = 13 is 83.25% and auc is 0.53
Cross Validation Accuracy for k = 15 is 83.28% and auc is 0.53
Cross Validation Accuracy for k = 17 is 83.27% and auc is 0.55
Cross Validation Accuracy for k = 19 is 83.27% and auc is 0.56
Cross Validation Accuracy for k = 21 is 83.28% and auc is 0.56
Cross Validation Accuracy for k = 23 is 83.27% and auc is 0.58
Cross Validation Accuracy for k = 25 is 83.26% and auc is 0.59
Cross Validation Accuracy for k = 27 is 83.25% and auc is 0.60
Cross Validation Accuracy for k = 29 is 83.26% and auc is 0.60
```

In [112]:
```
import matplotlib.pyplot as plt

# Plotting graph of auc and parameter

plt.plot(param_list,auc_list)
plt.xlabel("Parameter for K-NN")
plt.ylabel("Area Under Curve")
plt.title("Plot of parameter Vs AUC ")
plt.show()
```




```
In [113]: # Plotting graph of auc and parameter

          plt.plot(param_list,auc_list)
          plt.xlabel("Parameter for K-NN")
          plt.ylabel("Accuracy")
          plt.title("Plot of parameter Vs Accuracy ")
          plt.show()
```



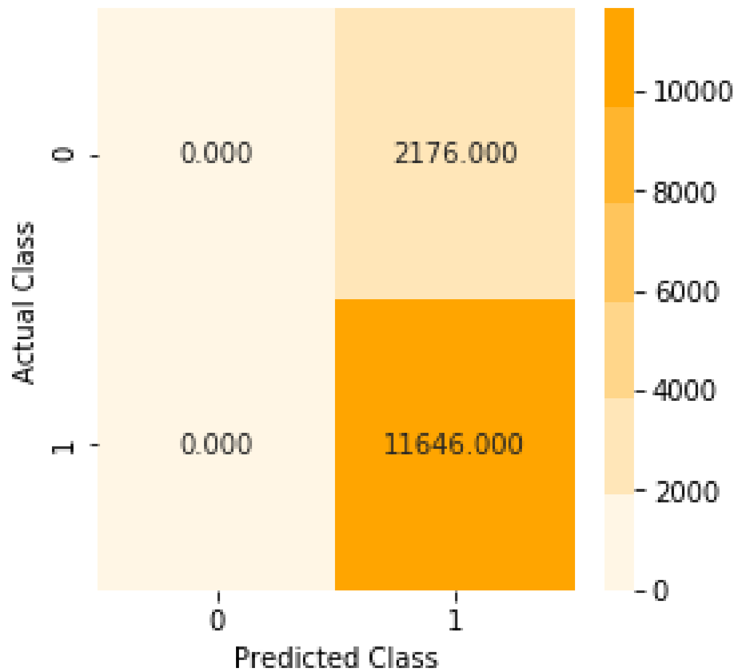

```
In [63]: # Training final model on best auc and taking k = 25

         final_clf = KNeighborsClassifier(n_neighbors=25,algorithm='brute',leaf_size=30)

         final_clf.fit(X_train,Y_train)

         predict_y = final_clf.predict(X_test)
         predict_probab = final_clf.predict_proba(X_test)[:,1] # This returns only probability

         acc = accuracy_score(Y_test,predict_y,normalize=True)* float(100)
         auc = roc_auc_score(Y_test,predict_probab)
         print("Final Accuracy is {:.2f}% and auc is {:.2f}".format(acc,auc))

         #print("For k = 29 final accuracy is {:.2f}% and auc is {:.3f}%".format(acc,auc))
```

```
Final Accuracy is 84.26% and auc is 0.51
```

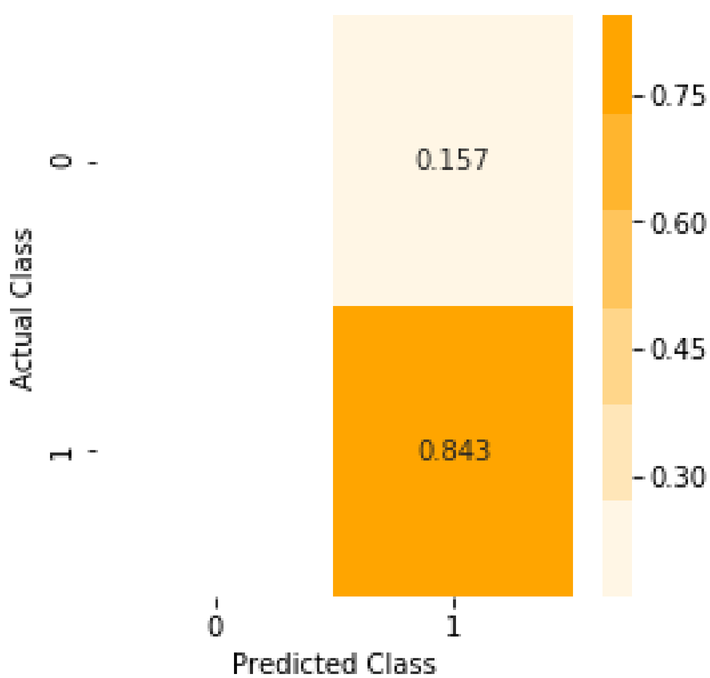
```
In [ ]: outfile = open("BOW_burte","wb")
        pickle.dump(final_clf,outfile)
        outfile.close()
```

```
In [64]: # Calling confusion_matrix_plot

         confusion_matrix_plot(Y_test,predict_y)
```

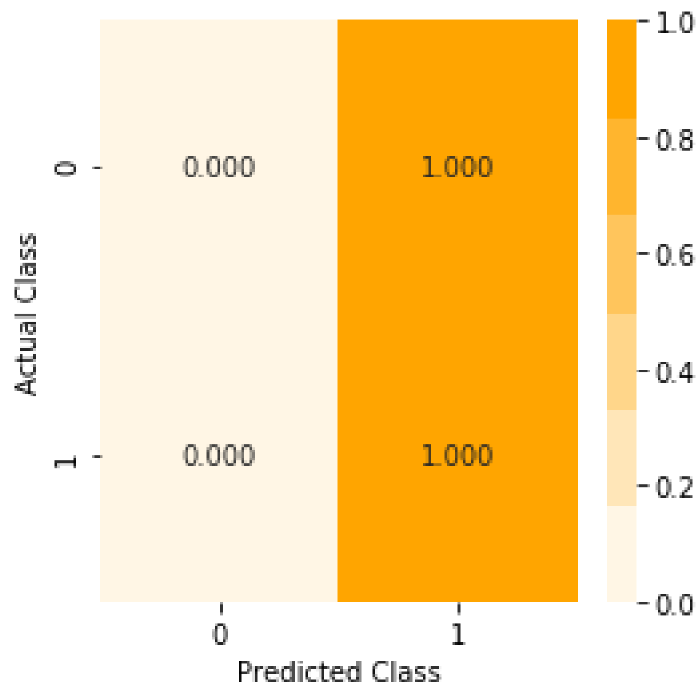
=========== Confusion matrix ==========



=========== Precision Matrix ==========



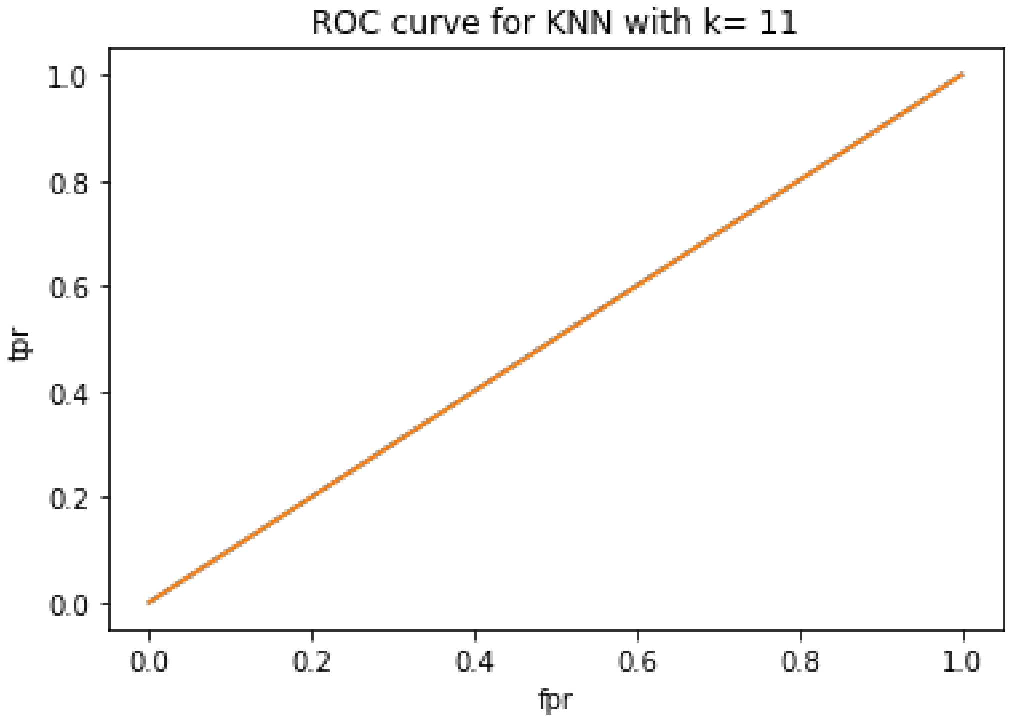
```
=========== Recall Matrix =============
```




```
In [65]: from sklearn.metrics import classification_report

         print(classification_report(Y_test,predict_y))

             precision    recall  f1-score   support

          0       0.00      0.00      0.00      2176
          1       0.84      1.00      0.91     11646

avg / total       0.71      0.84      0.77     13822
```

```
In [66]: # Plotting ROC Curve
         from sklearn.metrics import roc_curve

         fpr,tpr,threshold = roc_curve(Y_test,predict_y)

         plt.plot([0,1],[0,1])
```

```
plt.plot(fpr,tpr)
plt.xlabel("fpr")
plt.ylabel("tpr")
plt.title("ROC curve for KNN with k= 11")
plt.show()
```




```
In [67]: # Printing area under curve

         print("Area under ROC curve is ={:.3f} ".format(roc_auc_score(Y_test,predict_probab))
```

```
Area under ROC curve is =0.510
```

7.0.2 [5.1.3] Applying KNN brute force on AVG W2V, SET 3

```
In [95]: # In this section avg_w2v will be used for vectorization
         # Splitting datasets into train and test datasets

         X_train,X_test,Y_train,Y_test = train_test_split(list_of_sentence,Y,test_size=0.3,ranc
         print(X_train[0])
```

```
['like', 'one', 'alot', 'literally', 'tastes', 'like', 'chocolate', 'brownie', 'nuts', 'not',
```

```
In [96]: # Now we will vectorize train dataset usin avg_w2v
         train_sent_vectors = []; # the avg-w2v for each sentence/review is stored in this lis
         for sent in X_train: # for each review/sentence
             sent_vec = np.zeros(50) # as word vectors are of zero length 50, you might need t
             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
             train_sent_vectors.append(sent_vec)
         print(len(train_sent_vectors))
         print(len(train_sent_vectors[0]))
```

```
32249
50
```

```
In [97]: # Vectorization of test dataset using avg_w2v

         test_sent_vectors = []; # the avg-w2v for each sentence/review is stored in this list
         for sent in X_test: # for each review/sentence
             sent_vec = np.zeros(50) # as word vectors are of zero length 50, you might need t
             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
             test_sent_vectors.append(sent_vec)
         print(len(test_sent_vectors))
         print(len(test_sent_vectors[0]))
```

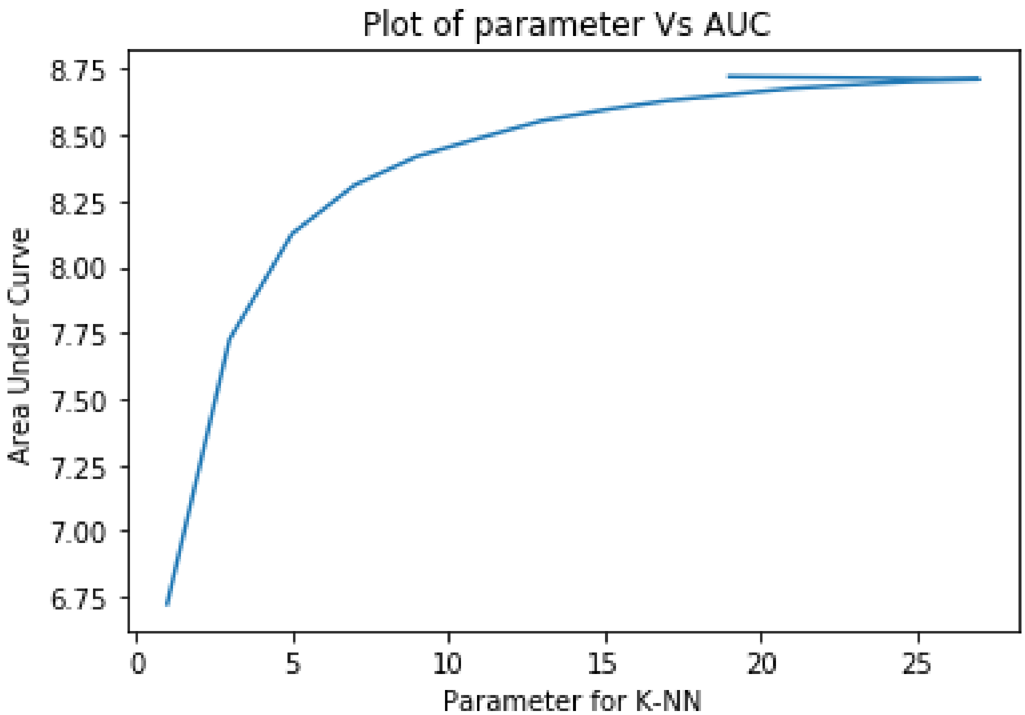
```
13822
50
```

```
In [102]: # 10 fold cross validation using time series splitting

          auc_list=[]
          acc_list=[]

          for k in range(1,30,2):
              # KNN Classifier
              clf = KNeighborsClassifier(n_neighbors=k,algorithm='brute',leaf_size=30)
```

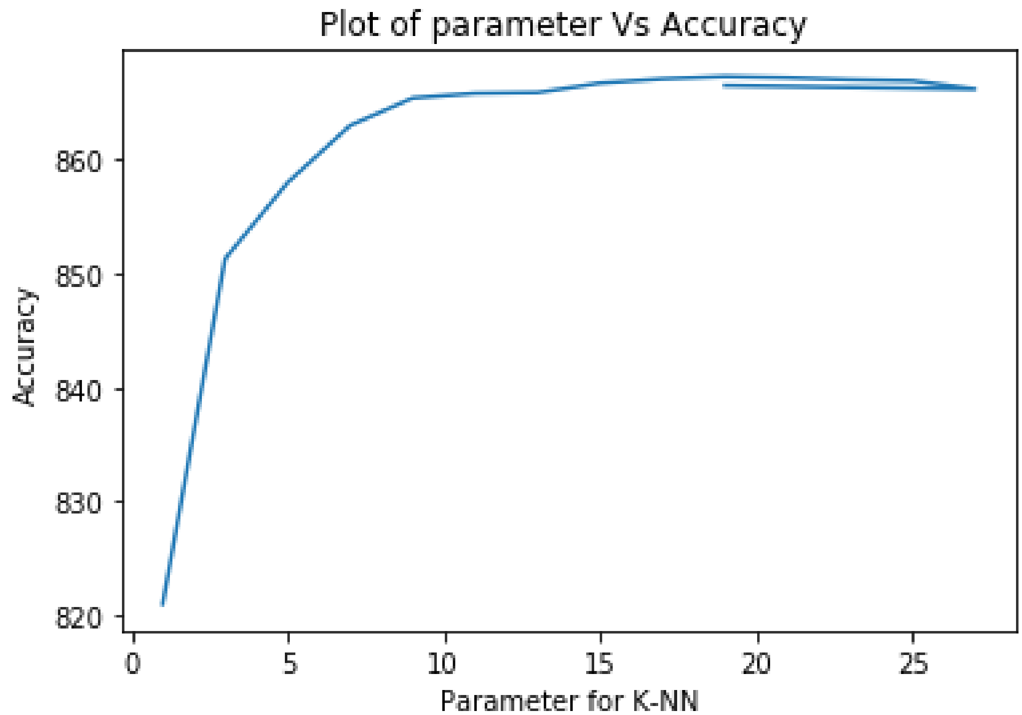
```
    i=0
    acc=0.0
    auc=0.0
    for train_index,test_index in tscv.split(train_sent_vectors):
        x_train = train_sent_vectors[0:train_index[-1]][:] # row 0 to train_index(ex
        y_train = Y_train[0:train_index[-1]][:] # row 0 to train_index(excluding)
        x_test = train_sent_vectors[train_index[-1]:test_index[-1]][:] # row from tra
        y_test = Y_train[train_index[-1]:test_index[-1]][:] # row from train_index to

        clf.fit(x_train,y_train)

        predict_y = clf.predict(x_test)
        predict_probab = clf.predict_proba(x_test)[:,1]
        i += 1
        acc += accuracy_score(y_test,predict_y,normalize=True) * float(100)
        auc += roc_auc_score(y_test,predict_probab)
    auc_list.append(auc)
    acc_list.append(acc)
    print("Cross Validation Accuracy for k = {:d} is {:.2f}% and auc is {:.2f}".forma
```

```
Cross Validation Accuracy for k = 1 is 82.09% and auc is 0.67
Cross Validation Accuracy for k = 3 is 85.13% and auc is 0.77
Cross Validation Accuracy for k = 5 is 85.80% and auc is 0.81
Cross Validation Accuracy for k = 7 is 86.30% and auc is 0.83
Cross Validation Accuracy for k = 9 is 86.54% and auc is 0.84
Cross Validation Accuracy for k = 11 is 86.58% and auc is 0.85
Cross Validation Accuracy for k = 13 is 86.59% and auc is 0.86
Cross Validation Accuracy for k = 15 is 86.67% and auc is 0.86
Cross Validation Accuracy for k = 17 is 86.71% and auc is 0.86
Cross Validation Accuracy for k = 19 is 86.73% and auc is 0.87
Cross Validation Accuracy for k = 21 is 86.72% and auc is 0.87
Cross Validation Accuracy for k = 23 is 86.70% and auc is 0.87
Cross Validation Accuracy for k = 25 is 86.69% and auc is 0.87
Cross Validation Accuracy for k = 27 is 86.62% and auc is 0.87
Cross Validation Accuracy for k = 29 is 86.65% and auc is 0.87
```

In [103]:
```
import matplotlib.pyplot as plt

# Plotting graph of auc and parameter

plt.plot(param_list,auc_list)
plt.xlabel("Parameter for K-NN")
plt.ylabel("Area Under Curve")
plt.title("Plot of parameter Vs AUC ")
plt.show()
```



In [104]:
```
# Plotting graph of accuracy and parameter

plt.plot(param_list,acc_list)
plt.xlabel("Parameter for K-NN")
plt.ylabel("Accuracy")
plt.title("Plot of parameter Vs Accuracy ")
plt.show()
```

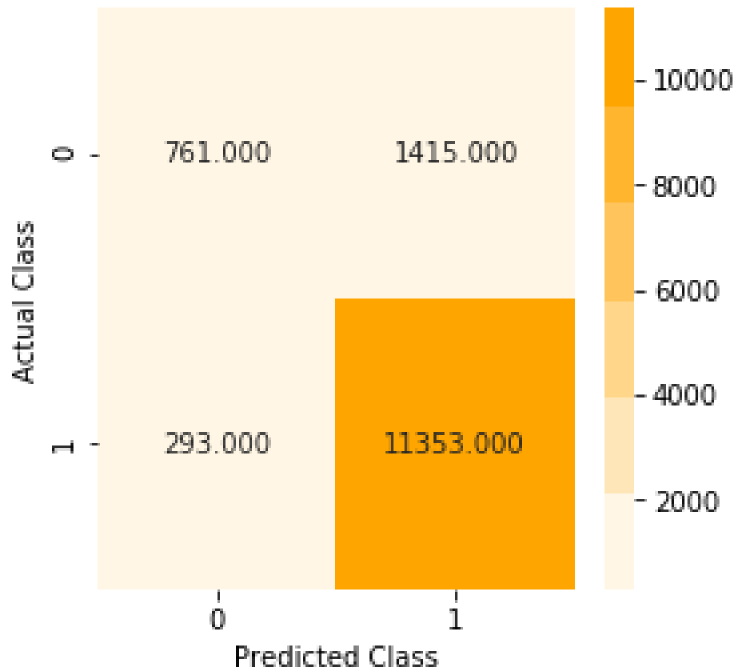



```
In [105]: # Training final model on best auc and taking k = 17

          final_clf = KNeighborsClassifier(n_neighbors=17,algorithm='brute',leaf_size=30)

          final_clf.fit(train_sent_vectors,Y_train)

          predict_y = final_clf.predict(test_sent_vectors)
          predict_probab = final_clf.predict_proba(test_sent_vectors)[:,1] # This returns only

          acc = accuracy_score(Y_test,predict_y,normalize=True)* float(100)
          auc = roc_auc_score(Y_test,predict_probab)
          print("Final Accuracy is {:.2f}% and auc is {:.2f}".format(acc,auc))

          #print("For k = 29 final accuracy is {:.2f}% and auc is {:.2f}%".format(acc,auc))
```

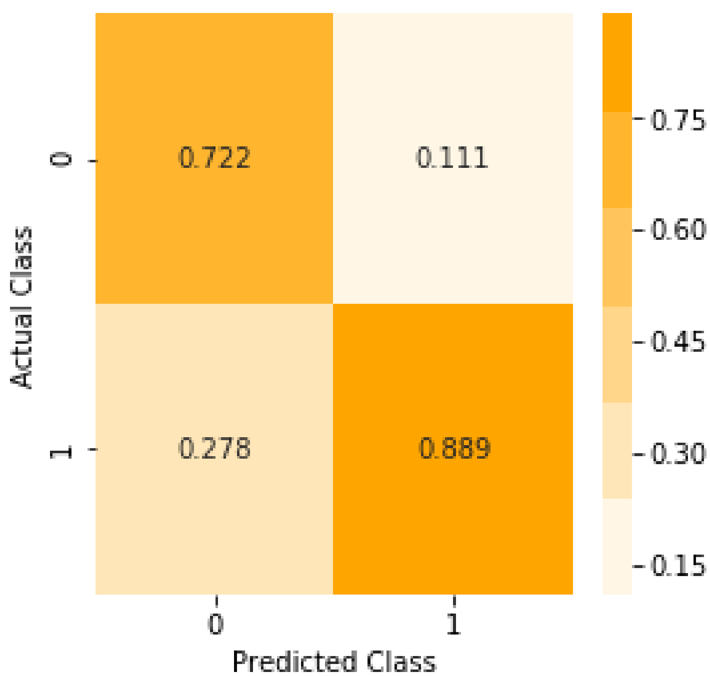
```
Final Accuracy is 87.77% and auc is 0.87
```

```
In [77]: # Calling confusion_matrix_plot

         confusion_matrix_plot(Y_test,predict_y)
```

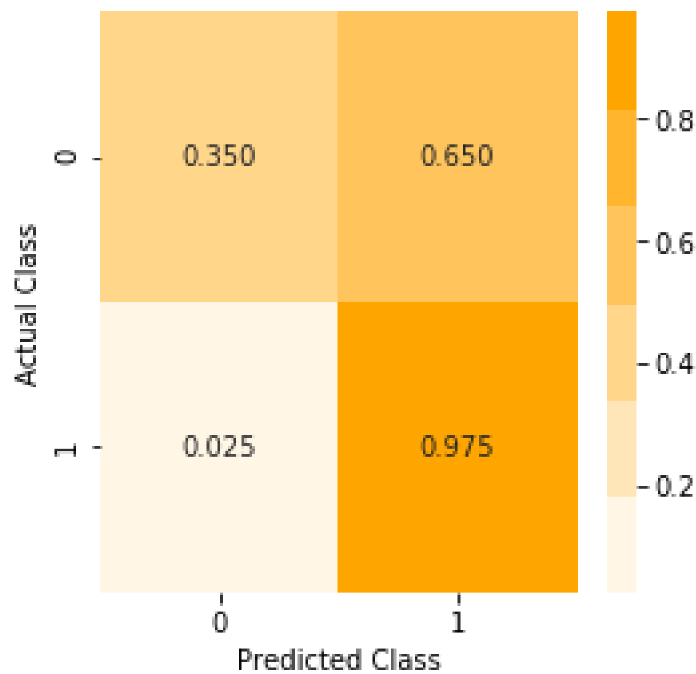
```
=========== Confusion matrix ==========
```




```
=========== Precision Matrix ==========
```



```
=========== Recall Matrix =============
```

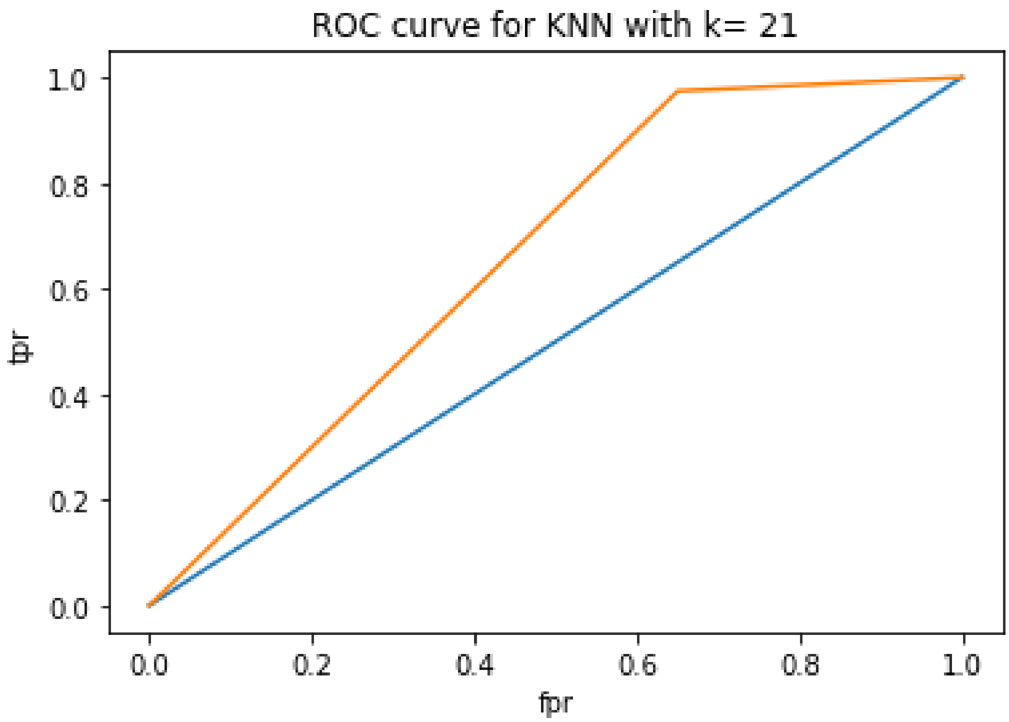



```
In [78]: from sklearn.metrics import classification_report

         print(classification_report(Y_test,predict_y))
```

```
             precision    recall  f1-score   support

          0       0.72      0.35      0.47      2176
          1       0.89      0.97      0.93     11646

avg / total       0.86      0.88      0.86     13822
```

```
In [80]: # Plotting ROC Curve
         from sklearn.metrics import roc_curve

         fpr,tpr,threshold = roc_curve(Y_test,predict_y)

         plt.plot([0,1],[0,1])
         plt.plot(fpr,tpr)
         plt.xlabel("fpr")
         plt.ylabel("tpr")
```

```
plt.title("ROC curve for KNN with k= 21")
plt.show()
```




```
In [81]: # Printing area under curve

         print("Area under ROC curve is ={:.3f} ".format(roc_auc_score(Y_test,predict_probab))
```

```
Area under ROC curve is =0.878
```

7.0.3 [5.1.4] Applying KNN brute force on TFIDF W2V

```
In [173]: X_train,X_test,Y_train,Y_test = train_test_split(cleaned_reviews,Y,test_size=0.3,ranc

In [174]: model = TfidfVectorizer()
          model.fit_transform(cleaned_reviews)
          tfidf_feat = model.get_feature_names()

In [175]: # Vectorizing train dataset

          # TF-IDF weighted Word2Vec
           # tfidf words/col-names
          # final_tf_idf is the sparse matrix with row= sentence, col=word and cell_val = tfid
```

```
train_tfidf_sent_vectors = []; # the tfidf-w2v for each sentence/review is stored in
row=0;
for sent in X_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 += 1
    if weight_sum != 0:
        sent_vec /= weight_sum
    train_tfidf_sent_vectors.append(sent_vec)
    row += 1
```

In [176]:
```
outfile = open("tfidf_w2v_train_vect","wb")
pickle.dump(train_tfidf_sent_vectors,outfile)
outfile.close()
```

In [146]:
```
infile = open("tfidf_w2v_train_vect","rb")
train_tfidf_sent_vectors = pickle.load(infile)
infile.close()
```

In [177]:
```
# TF-IDF weighted Word2Vec
model.fit_transform(X_test)
tfidf_feat = model.get_feature_names()
test_tfidf_sent_vectors = []; # the tfidf-w2v for each sentence/review is stored in
row=0;
for sent in X_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 values of word in this review
            tf_idf = dictionary[word]*(sent.count(word)/len(sent))
            sent_vec += (vec * tf_idf)
            weight_sum +=1
    if weight_sum != 0:
        sent_vec /= weight_sum
```

```
            test_tfidf_sent_vectors.append(sent_vec)
            row += 1
```

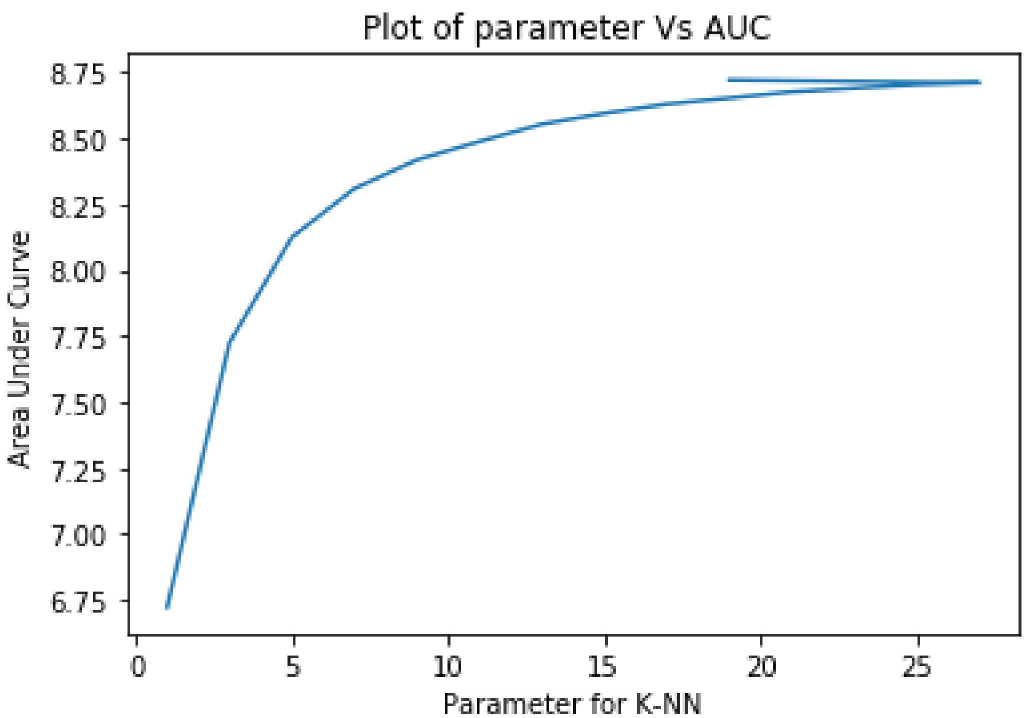
In [178]:
```
outfile = open("tfidf_w2v_test_vect","wb")
pickle.dump(test_tfidf_sent_vectors,outfile)
outfile.close()
```

In [179]:
```
# 10 fold cross validation using time series splitting

acc_list = []
auc_list = []

for k in range(1,30,2):
    # KNN Classifier
    clf = KNeighborsClassifier(n_neighbors=k,algorithm='brute',leaf_size=30)
    i=0
    acc=0.0
    auc=0.0
    for train_index,test_index in tscv.split(train_tfidf_sent_vectors):
        x_train = train_sent_vectors[0:train_index[-1]][:] # row 0 to train_index(ex
        y_train = Y_train[0:train_index[-1]][:] # row 0 to train_index(excluding)
        x_test = train_sent_vectors[train_index[-1]:test_index[-1]][:] # row from tra
        y_test = Y_train[train_index[-1]:test_index[-1]][:] # row from train_index t

        clf.fit(x_train,y_train)

        predict_y = clf.predict(x_test)
        predict_probab = clf.predict_proba(x_test)[:,1]
        i += 1
        acc += accuracy_score(y_test,predict_y,normalize=True) * float(100)
        auc += roc_auc_score(y_test,predict_probab)

    acc_list.append(acc)
    auc_list.append(auc)
    print("Cross Validation Accuracy for k = {:d} is {:.2f}% and auc is {:.2f}".forma
```

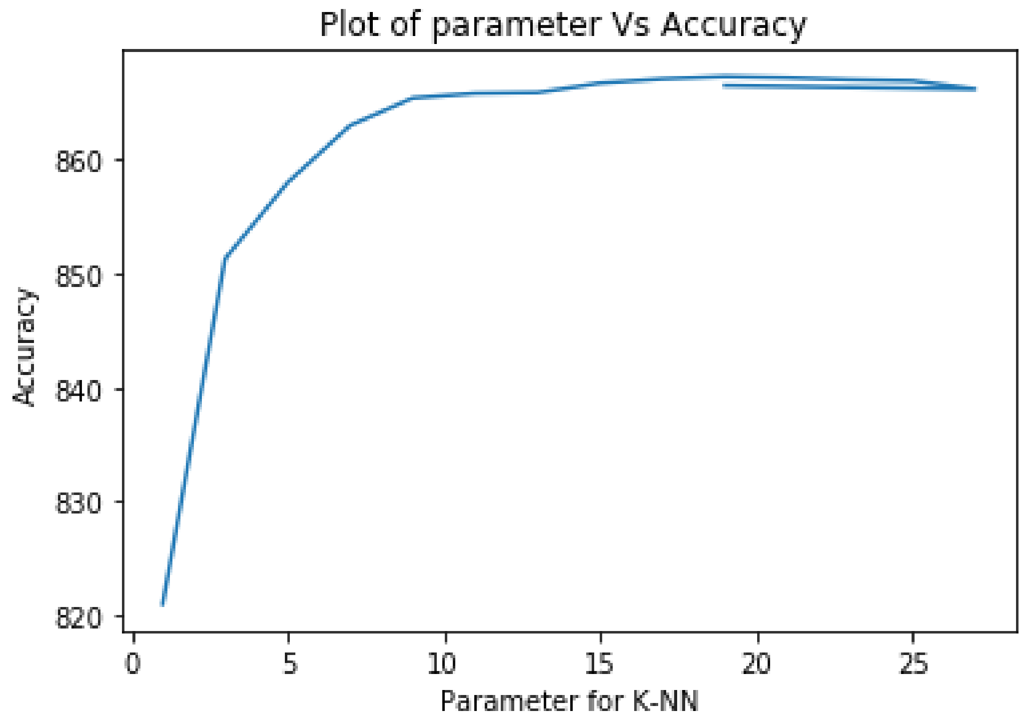
```
Cross Validation Accuracy for k = 1 is 82.09% and auc is 0.67
Cross Validation Accuracy for k = 3 is 85.13% and auc is 0.77
Cross Validation Accuracy for k = 5 is 85.80% and auc is 0.81
Cross Validation Accuracy for k = 7 is 86.30% and auc is 0.83
Cross Validation Accuracy for k = 9 is 86.54% and auc is 0.84
Cross Validation Accuracy for k = 11 is 86.58% and auc is 0.85
Cross Validation Accuracy for k = 13 is 86.59% and auc is 0.86
Cross Validation Accuracy for k = 15 is 86.67% and auc is 0.86
Cross Validation Accuracy for k = 17 is 86.71% and auc is 0.86
Cross Validation Accuracy for k = 19 is 86.73% and auc is 0.87
Cross Validation Accuracy for k = 21 is 86.72% and auc is 0.87
Cross Validation Accuracy for k = 23 is 86.70% and auc is 0.87
```

```
Cross Validation Accuracy for k = 25 is 86.69% and auc is 0.87
Cross Validation Accuracy for k = 27 is 86.62% and auc is 0.87
Cross Validation Accuracy for k = 29 is 86.65% and auc is 0.87
```

In [180]:
```python
import matplotlib.pyplot as plt

# Plotting graph of auc and parameter

plt.plot(param_list,auc_list)
plt.xlabel("Parameter for K-NN")
plt.ylabel("Area Under Curve")
plt.title("Plot of parameter Vs AUC ")
plt.show()
```



In [181]:
```python
plt.plot(param_list,acc_list)
plt.xlabel("Parameter for K-NN")
plt.ylabel("Accuracy")
plt.title("Plot of parameter Vs Accuracy ")
plt.show()
```

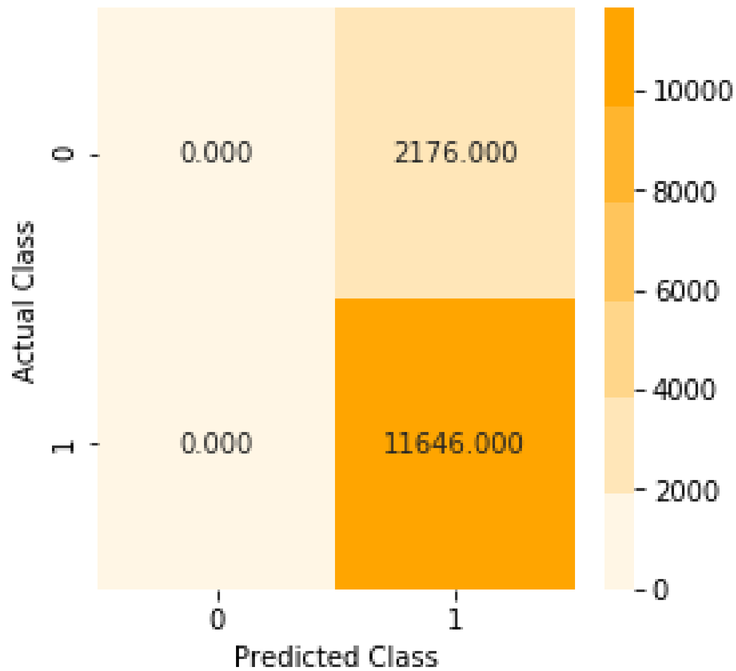



```
In [182]: # Training final model on best auc and taking k = 17

          final_clf = KNeighborsClassifier(n_neighbors=21,algorithm='brute',leaf_size=30)

          final_clf.fit(train_tfidf_sent_vectors,Y_train)

          predict_y = final_clf.predict(test_tfidf_sent_vectors)
          predict_probab = final_clf.predict_proba(test_tfidf_sent_vectors)[:,1] # This return

          acc = accuracy_score(Y_test,predict_y,normalize=True)* float(100)
          auc = roc_auc_score(Y_test,predict_probab)
          print("Final Accuracy is {:.2f}% and auc is {:.2f}".format(acc,auc))
```

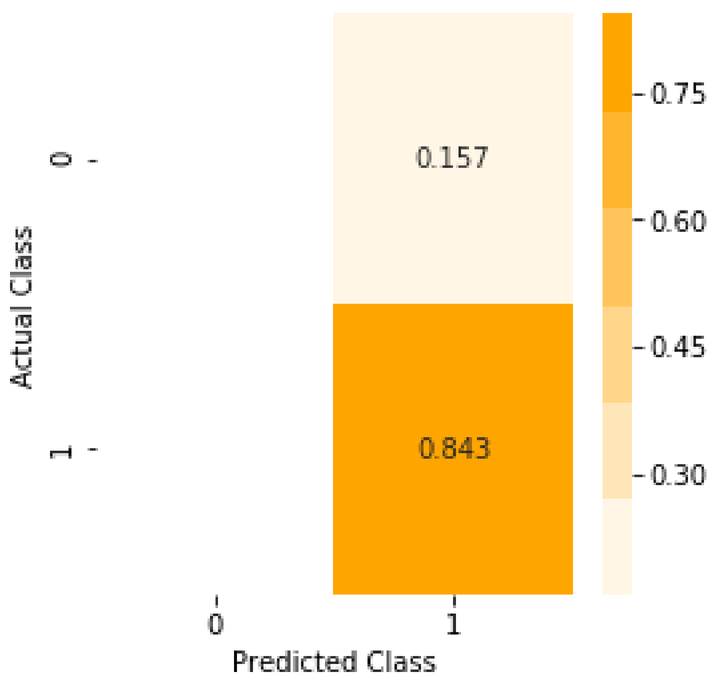
```
Final Accuracy is 84.26% and auc is 0.50
```

```
In [183]: # Calling confusion_matrix_plot

          confusion_matrix_plot(Y_test,predict_y)
```

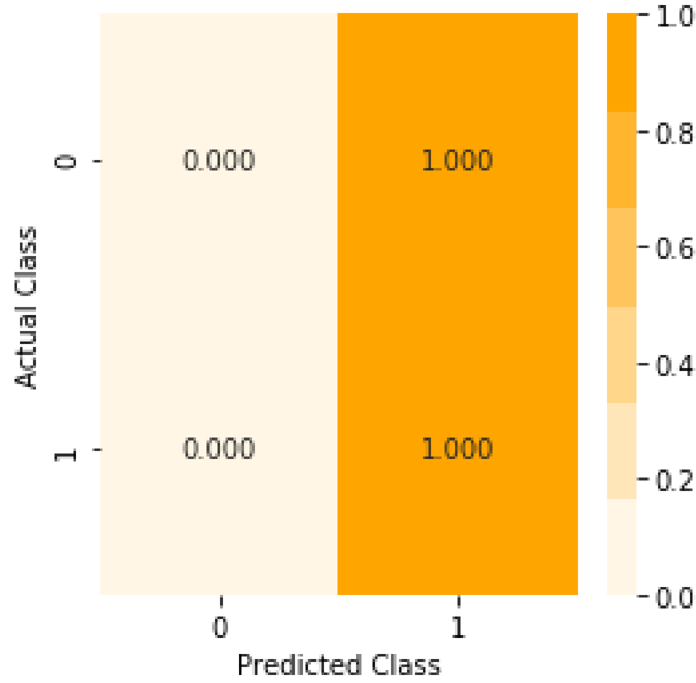
```
=========== Confusion matrix ==========
```




```
=========== Precision Matrix ==========
```



```
=========== Recall Matrix =============
```

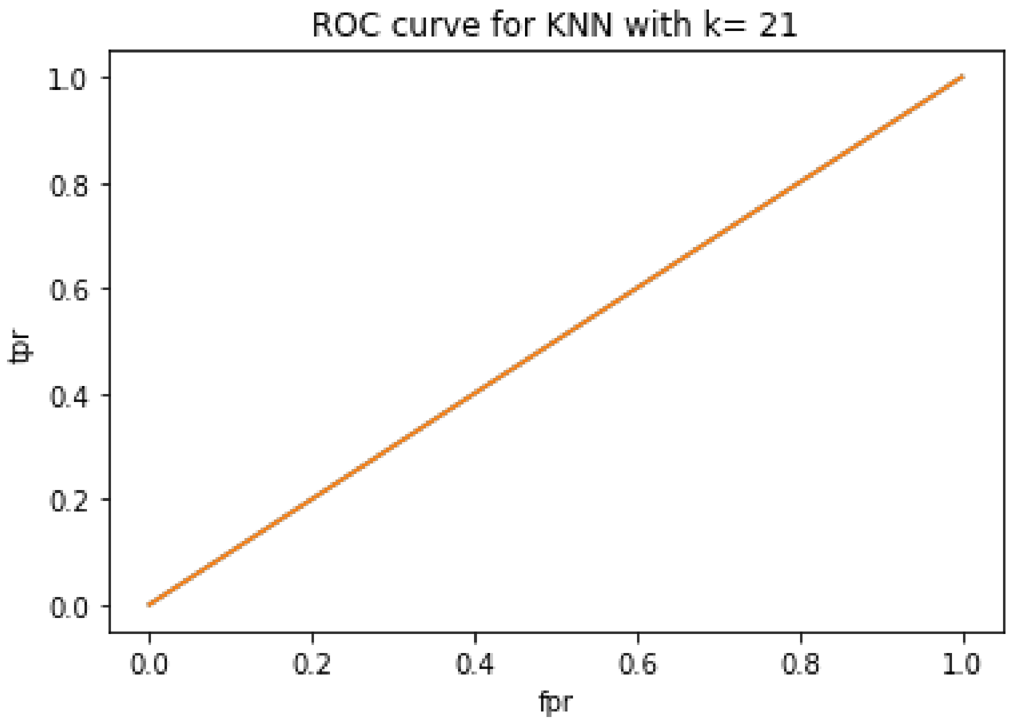


In [184]:
```python
from sklearn.metrics import classification_report

print(classification_report(Y_test,predict_y))
```

```
             precision    recall  f1-score   support

          0       0.00      0.00      0.00      2176
          1       0.84      1.00      0.91     11646

avg / total       0.71      0.84      0.77     13822
```

In [185]:
```python
# Plotting ROC Curve
from sklearn.metrics import roc_curve

fpr,tpr,threshold = roc_curve(Y_test,predict_y)

plt.plot([0,1],[0,1])
plt.plot(fpr,tpr)
plt.xlabel("fpr")
plt.ylabel("tpr")
```

```
plt.title("ROC curve for KNN with k= 21")
plt.show()

# Printing area under curve

print("Area under ROC curve is ={:.3f} ".format(roc_auc_score(Y_test,predict_probab))
```




```
Area under ROC curve is =0.500
```

7.1 [5.2] Applying KNN kd-tree

7.1.1 [5.2.1] Applying KNN kd-tree on BOW, SET 5

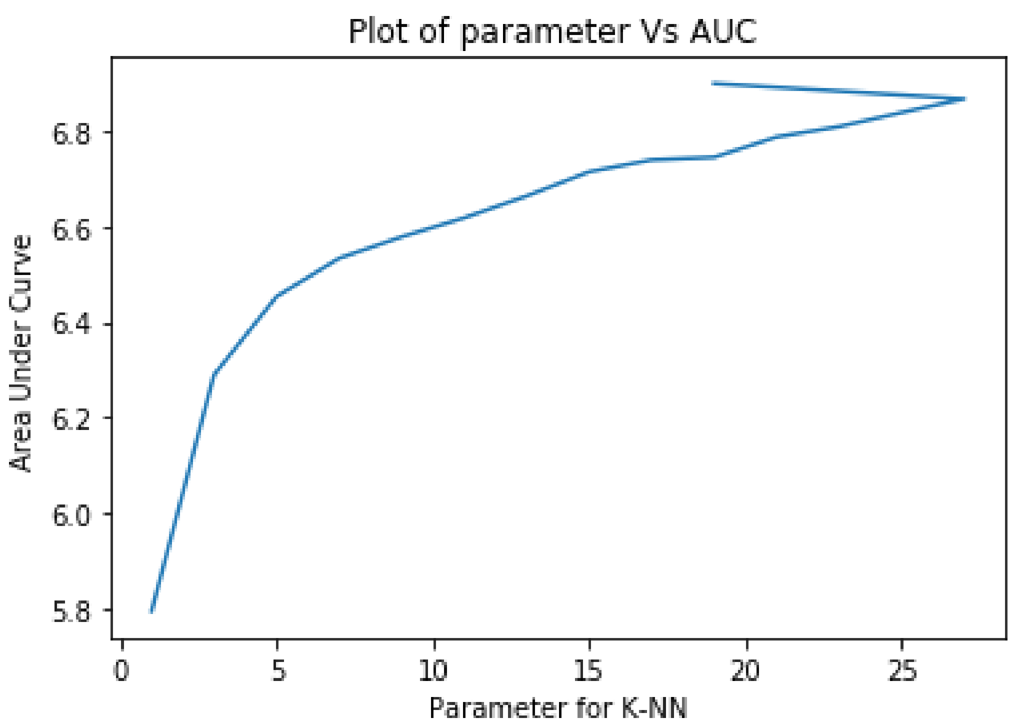
```
In [60]: # Kd tree works slow for high dimensional data.
         # Therefore taking top 5000 features
         count_vect = CountVectorizer(max_features=5000)

         X = cleaned_reviews
         # Splitting data into train and test dataset
         X_train,X_test,Y_train,Y_test = train_test_split(X,Y,test_size=0.3,random_state=42)
         print(len(X_train),len(X_test))
```

```
32249 13822
```

```
In [61]: # Vectorizing train and test dataset seperately
         X_train = count_vect.fit_transform(X_train)
         X_train.shape

         X_test = count_vect.transform(X_test)
         X_test.shape

Out[61]: (13822, 5000)

In [68]: # Performing 10 fold cross validation on time split data
         from sklearn.neighbors import KNeighborsClassifier
         from sklearn.model_selection import TimeSeriesSplit

         tscv = TimeSeriesSplit(n_splits=10)

         param_list=[1,3,5,7,9,11,13,15,17,19,21,23,25,27,19]
         auc_list = []
         acc_list = []

         for k in range(1,30,2):
             clf = KNeighborsClassifier(n_neighbors=k,algorithm='kd_tree',leaf_size=40)
             acc = 0.0
             auc = 0.0
             i=0
             for train_index,test_index in tscv.split(X_train):
                 x_train = X_train[0:train_index[-1]][:] # row 0 to train_index(excluding)
                 y_train = Y_train[0:train_index[-1]][:] # row 0 to train_index(excluding)
                 x_test = X_train[train_index[-1]:test_index[-1]][:] # row from train_index to
                 y_test = Y_train[train_index[-1]:test_index[-1]][:] # row from train_index to

                 clf.fit(x_train,y_train)

                 predict_y = clf.predict(x_test)
                 predict_probab = clf.predict_proba(x_test)[:,1]
                 i += 1
                 acc += accuracy_score(y_test,predict_y,normalize=True) * float(100)
                 auc += roc_auc_score(y_test,predict_probab)

             auc_list.append(auc)
             acc_list.append(acc)
             print("Cross Validation Accuracy for k = {:d} is {:.2f}% and auc is {:.2f}".forma


Cross Validation Accuracy for k = 1 is 77.79% and auc is 0.58
Cross Validation Accuracy for k = 3 is 82.71% and auc is 0.63
Cross Validation Accuracy for k = 5 is 83.07% and auc is 0.65
Cross Validation Accuracy for k = 7 is 83.23% and auc is 0.65
Cross Validation Accuracy for k = 9 is 83.31% and auc is 0.66
```

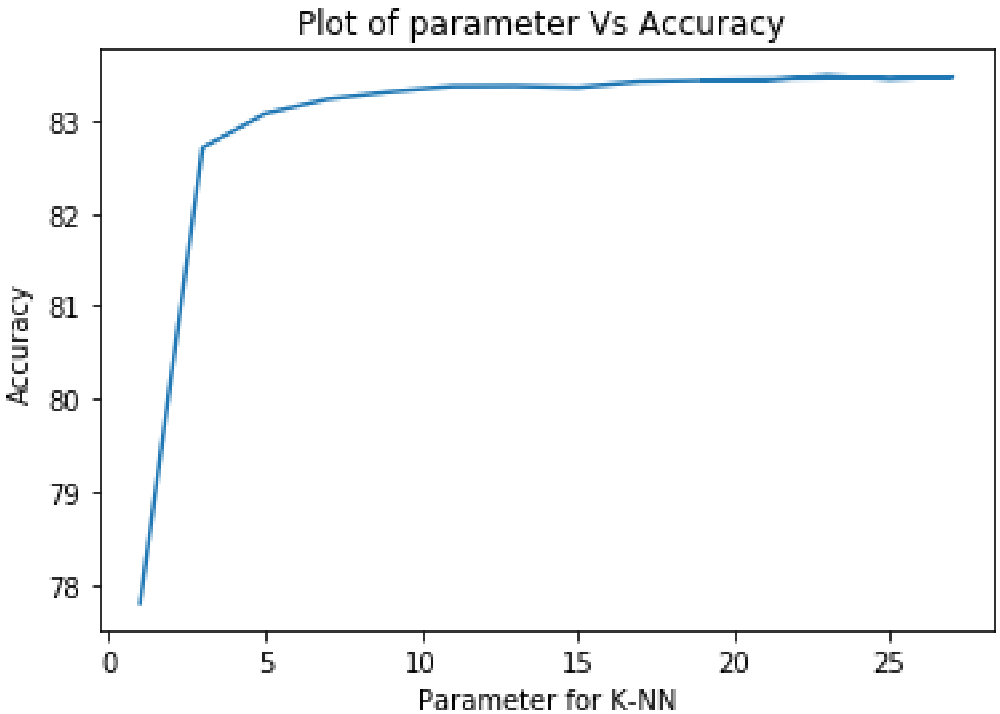
```
Cross Validation Accuracy for k = 11 is 83.37% and auc is 0.66
Cross Validation Accuracy for k = 13 is 83.37% and auc is 0.67
Cross Validation Accuracy for k = 15 is 83.35% and auc is 0.67
Cross Validation Accuracy for k = 17 is 83.42% and auc is 0.67
Cross Validation Accuracy for k = 19 is 83.43% and auc is 0.67
Cross Validation Accuracy for k = 21 is 83.43% and auc is 0.68
Cross Validation Accuracy for k = 23 is 83.48% and auc is 0.68
Cross Validation Accuracy for k = 25 is 83.44% and auc is 0.68
Cross Validation Accuracy for k = 27 is 83.47% and auc is 0.69
Cross Validation Accuracy for k = 29 is 83.44% and auc is 0.69
```

In [69]:
```
import matplotlib.pyplot as plt

# Plotting graph of auc and parameter

plt.plot(param_list,auc_list)
plt.xlabel("Parameter for K-NN")
plt.ylabel("Area Under Curve")
plt.title("Plot of parameter Vs AUC ")
plt.show()
```



In [73]:
```
# Plotting graph of accuracy and parameter
```

```
acc_list = [x*10.0 for x in acc_list]

plt.plot(param_list,acc_list)
plt.xlabel("Parameter for K-NN")
plt.ylabel("Accuracy")
plt.title("Plot of parameter Vs Accuracy ")
plt.show()
```

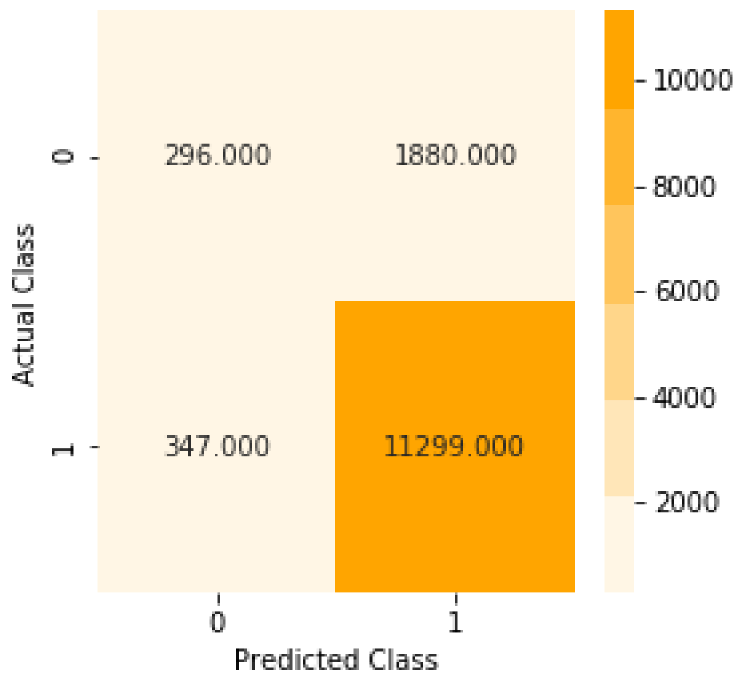



```
In [75]: # Testing final model for k = 25

         final_clf = KNeighborsClassifier(n_neighbors=25,algorithm='kd_tree',leaf_size=40)
         final_clf.fit(X_train,Y_train)

         predict_y = final_clf.predict(X_test)
         predict_probab = final_clf.predict_proba(X_test)[:,1]

         acc = accuracy_score(Y_test,predict_y,normalize=True)* float(100)
         auc = roc_auc_score(Y_test,predict_probab)

         print("Accuracy of model for k = 25 is {:.2f} and AUC is {:.2f}".format(acc,auc))
```

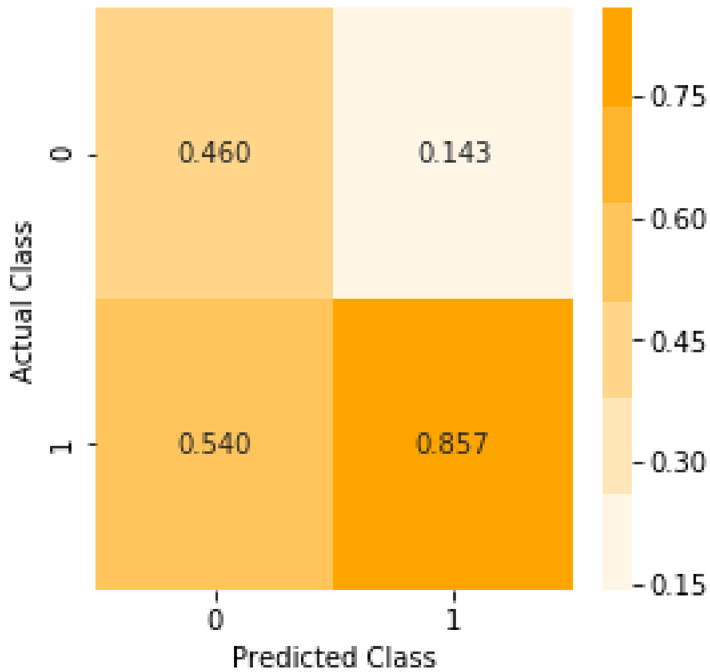
```
Accuracy of model for k = 25 is 83.89 and AUC is 0.68
```

```
In [76]: # Plotting confusion matrix plot
         confusion_matrix_plot(Y_test,predict_y)
```

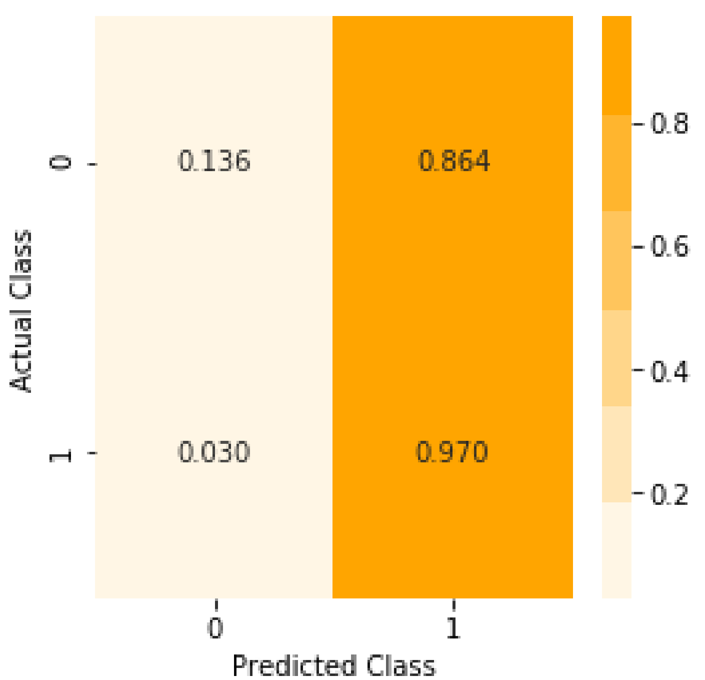
```
=========== Confusion matrix ==========
```




```
=========== Precision Matrix ==========
```



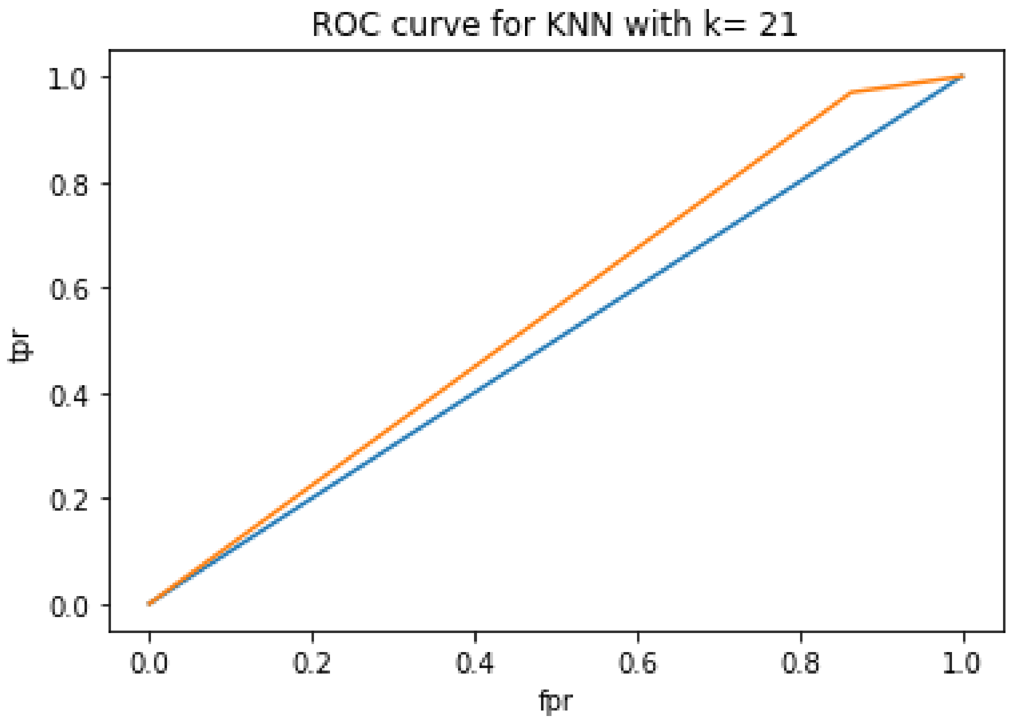

```
=========== Recall Matrix =============
```



In [77]:
```
from sklearn.metrics import classification_report

print(classification_report(Y_test,predict_y))
```

```
             precision    recall  f1-score   support

          0       0.46      0.14      0.21      2176
          1       0.86      0.97      0.91     11646

avg / total       0.79      0.84      0.80     13822
```

In [78]:
```
# Plotting ROC Curve
from sklearn.metrics import roc_curve

fpr,tpr,threshold = roc_curve(Y_test,predict_y)

plt.plot([0,1],[0,1])
plt.plot(fpr,tpr)
plt.xlabel("fpr")
plt.ylabel("tpr")
plt.title("ROC curve for KNN with k= 21")
plt.show()

# Printing area under curve

print("Area under ROC curve is ={:.3f} ".format(roc_auc_score(Y_test,predict_probab)))
```




```
Area under ROC curve is =0.681
```

```
In [79]: # Printing area under curve

         print("Area under ROC curve is ={:.3f} ".format(roc_auc_score(Y_test,predict_probab)))
```

```
Area under ROC curve is =0.681
```

7.1.2 [5.2.2] Applying KNN kd-tree on TFIDF, SET 6

```
In [80]: # In this section Tfidf will be used for vectorization
         # Splitting datasets into train and test datasets

         tf_idf_vect = TfidfVectorizer(max_features=5000)

         X_train,X_test,Y_train,Y_test = train_test_split(X,Y,test_size=0.3,random_state=42)

         # Now we will vectorize train and test datasets separately using Tfidf
         # Use fit_transform to vectorize train dataset and transform to vectorize test datase
         X_train = tf_idf_vect.fit_transform(X_train)
         X_test = tf_idf_vect.transform(X_test)
```

In [87]: # Performing 10 fold cross validation on time series split data

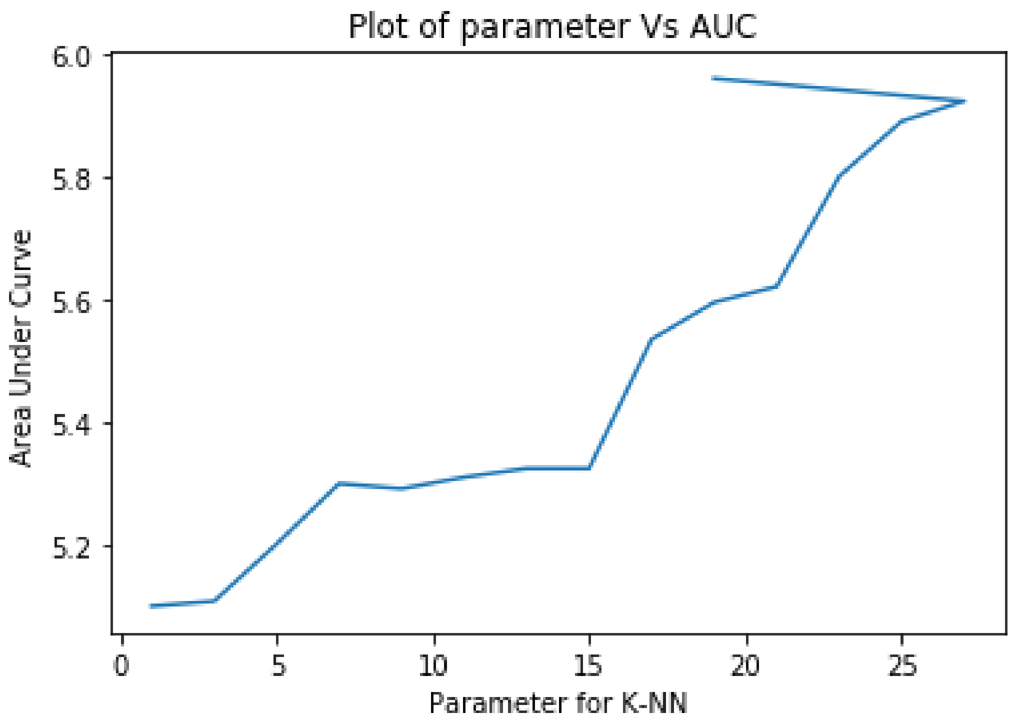
```
auc_list=[]
acc_list=[]
for k in range(1,30,2):
    # KNN Classifier
    clf = KNeighborsClassifier(n_neighbors=k,algorithm='kd_tree',leaf_size=40)
    i=0
    acc=0.0
    auc=0.0

    for train_index,test_index in tscv.split(X_train):
        x_train = X_train[0:train_index[-1]][:] # row 0 to train_index(excluding)
        y_train = Y_train[0:train_index[-1]][:] # row 0 to train_index(excluding)
        x_test = X_train[train_index[-1]:test_index[-1]][:] # row from train_index to
        y_test = Y_train[train_index[-1]:test_index[-1]][:] # row from train_index to

        clf.fit(x_train,y_train)

        predict_y = clf.predict(x_test)
        predict_probab = clf.predict_proba(x_test)[:,1]
        i += 1
        acc += accuracy_score(y_test,predict_y,normalize=True) * float(100)
        auc += roc_auc_score(y_test,predict_probab)

    auc_list.append(auc)
    acc_list.append(acc)

    print("Cross Validation Accuracy for k = {:d} is {:.2f}% and auc is {:.2f}".forma
```

```
Cross Validation Accuracy for k = 1 is 82.72% and auc is 0.51
Cross Validation Accuracy for k = 3 is 83.21% and auc is 0.51
Cross Validation Accuracy for k = 5 is 83.17% and auc is 0.52
Cross Validation Accuracy for k = 7 is 83.24% and auc is 0.53
Cross Validation Accuracy for k = 9 is 83.25% and auc is 0.53
Cross Validation Accuracy for k = 11 is 83.25% and auc is 0.53
Cross Validation Accuracy for k = 13 is 83.27% and auc is 0.53
Cross Validation Accuracy for k = 15 is 83.25% and auc is 0.53
Cross Validation Accuracy for k = 17 is 83.25% and auc is 0.55
Cross Validation Accuracy for k = 19 is 83.25% and auc is 0.56
Cross Validation Accuracy for k = 21 is 83.25% and auc is 0.56
Cross Validation Accuracy for k = 23 is 83.25% and auc is 0.58
Cross Validation Accuracy for k = 25 is 83.25% and auc is 0.59
Cross Validation Accuracy for k = 27 is 83.24% and auc is 0.59
Cross Validation Accuracy for k = 29 is 83.24% and auc is 0.60
```

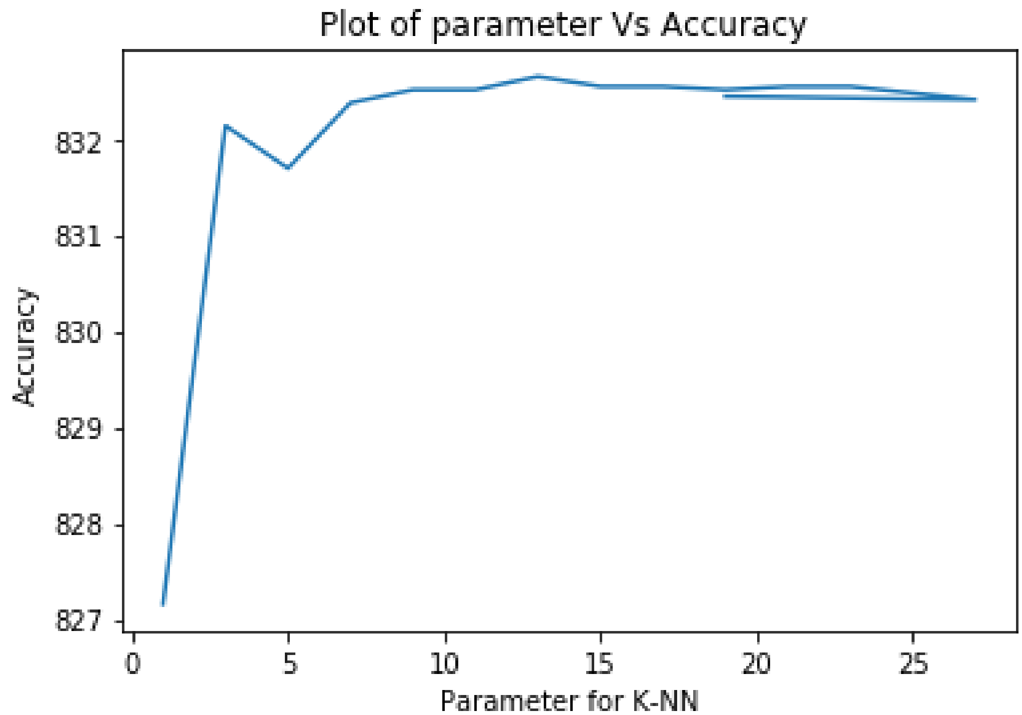
In [88]:

```
import matplotlib.pyplot as plt
```

```
# Plotting graph of auc and parameter

plt.plot(param_list,auc_list)
plt.xlabel("Parameter for K-NN")
plt.ylabel("Area Under Curve")
plt.title("Plot of parameter Vs AUC ")
plt.show()
```




```
In [89]: import matplotlib.pyplot as plt

         # Plotting graph of accuracy and parameter

         plt.plot(param_list,acc_list)
         plt.xlabel("Parameter for K-NN")
         plt.ylabel("Accuracy")
         plt.title("Plot of parameter Vs Accuracy ")
         plt.show()
```

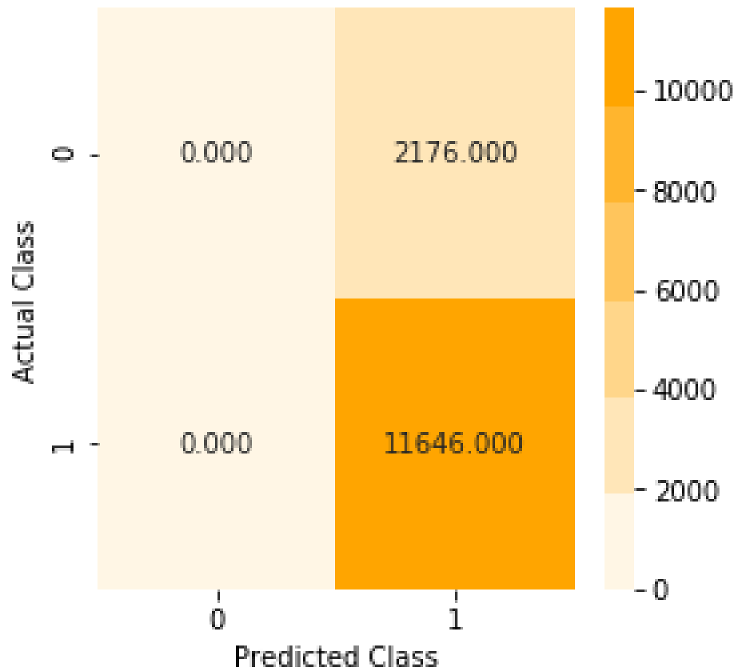



```
In [90]: # Training final model on best auc and taking k = 25

         final_clf = KNeighborsClassifier(n_neighbors=25,algorithm='brute',leaf_size=30)

         final_clf.fit(X_train,Y_train)

         predict_y = final_clf.predict(X_test)
         predict_probab = final_clf.predict_proba(X_test)[:,1] # This returns only probability

         acc = accuracy_score(Y_test,predict_y,normalize=True)* float(100)
         auc = roc_auc_score(Y_test,predict_probab)
         print("Final Accuracy is {:.2f}% and auc is {:.2f}".format(acc,auc))

         #print("For k = 25 final accuracy is {:.2f}% and auc is {:.3f}%".format(acc,auc))
```

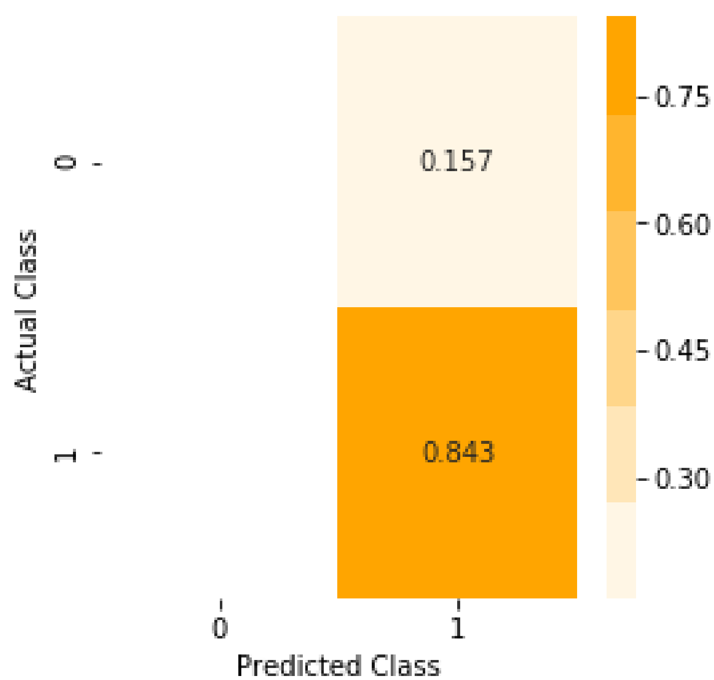
```
Final Accuracy is 84.26% and auc is 0.53
```

```
In [91]: # Calling confusion_matrix_plot

         confusion_matrix_plot(Y_test,predict_y)
```

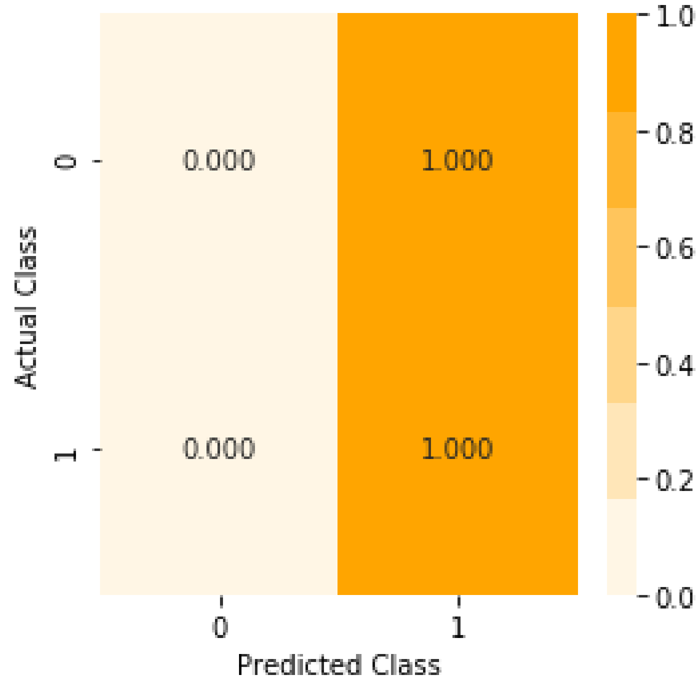
```
=========== Confusion matrix ==========
```




```
=========== Precision Matrix ==========
```



```
=========== Recall Matrix =============
```

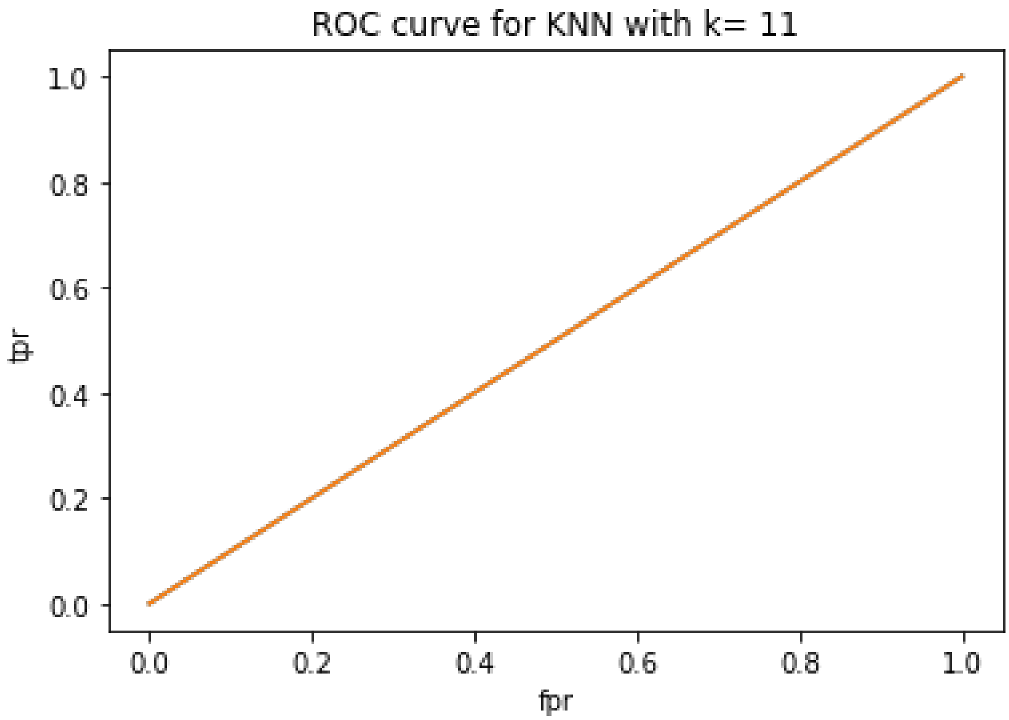



```
In [92]: from sklearn.metrics import classification_report

         print(classification_report(Y_test,predict_y))
```

```
             precision    recall  f1-score   support

          0       0.00      0.00      0.00      2176
          1       0.84      1.00      0.91     11646

avg / total       0.71      0.84      0.77     13822
```

```
In [93]: # Plotting ROC Curve
         from sklearn.metrics import roc_curve

         fpr,tpr,threshold = roc_curve(Y_test,predict_y)

         plt.plot([0,1],[0,1])
         plt.plot(fpr,tpr)
         plt.xlabel("fpr")
         plt.ylabel("tpr")
```

```
plt.title("ROC curve for KNN with k= 11")
plt.show()
```




```
In [94]: # Printing area under curve

         print("Area under ROC curve is ={:.3f} ".format(roc_auc_score(Y_test,predict_probab))
```

```
Area under ROC curve is =0.528
```

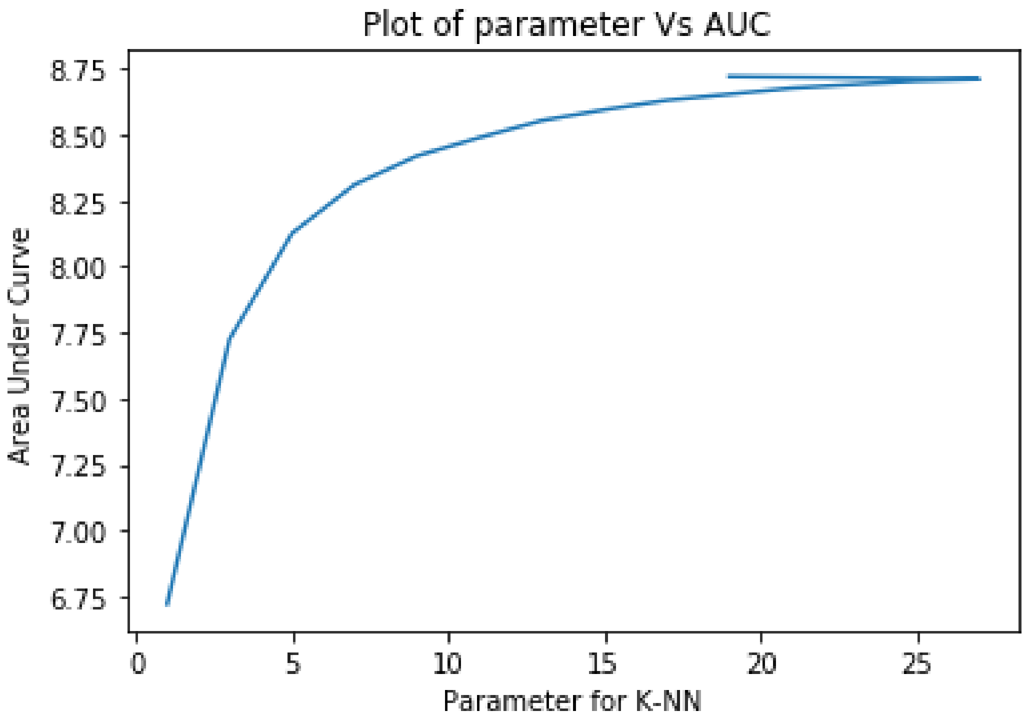
7.1.3 [5.2.3] Applying KNN kd-tree on AVG W2V, SET 3

```
In [106]: # 10 fold cross validation using time series splitting
          # Here X_train is train_sent_vectors and X_test is test_sent_vectors
          # Here vectorization results of previous section is used . Only KNN algorithm is cha

          auc_list=[]
          acc_list=[]
          for k in range(1,30,2):
              # KNN Classifier
              clf = KNeighborsClassifier(n_neighbors=k,algorithm='kd_tree',leaf_size=40,n_jobs=
              i=0
              acc=0.0
              auc=0.0
```

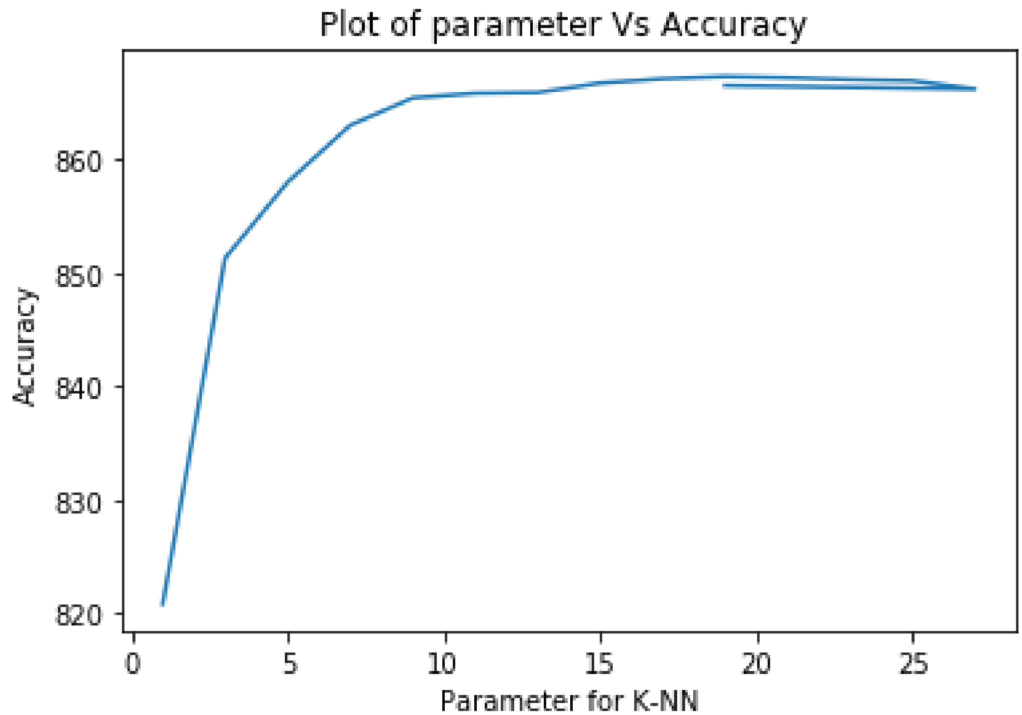
```
        for train_index,test_index in tscv.split(train_sent_vectors):
            x_train = train_sent_vectors[0:train_index[-1]][:] # row 0 to train_index(ex
            y_train = Y_train[0:train_index[-1]][:] # row 0 to train_index(excluding)
            x_test = train_sent_vectors[train_index[-1]:test_index[-1]][:] # row from tr
            y_test = Y_train[train_index[-1]:test_index[-1]][:] # row from train_index t

            clf.fit(x_train,y_train)

            predict_y = clf.predict(x_test)
            predict_probab = clf.predict_proba(x_test)[:,1]
            i += 1
            acc += accuracy_score(y_test,predict_y,normalize=True) * float(100)
            auc += roc_auc_score(y_test,predict_probab)

        auc_list.append(auc)
        acc_list.append(acc)
        print("Cross Validation Accuracy for k = {:d} is {:.2f}% and auc is {:.2f}".forma
```

```
Cross Validation Accuracy for k = 1 is 82.08% and auc is 0.67
Cross Validation Accuracy for k = 3 is 85.13% and auc is 0.77
Cross Validation Accuracy for k = 5 is 85.80% and auc is 0.81
Cross Validation Accuracy for k = 7 is 86.30% and auc is 0.83
Cross Validation Accuracy for k = 9 is 86.54% and auc is 0.84
Cross Validation Accuracy for k = 11 is 86.58% and auc is 0.85
Cross Validation Accuracy for k = 13 is 86.59% and auc is 0.86
Cross Validation Accuracy for k = 15 is 86.67% and auc is 0.86
Cross Validation Accuracy for k = 17 is 86.71% and auc is 0.86
Cross Validation Accuracy for k = 19 is 86.73% and auc is 0.87
Cross Validation Accuracy for k = 21 is 86.72% and auc is 0.87
Cross Validation Accuracy for k = 23 is 86.70% and auc is 0.87
Cross Validation Accuracy for k = 25 is 86.69% and auc is 0.87
Cross Validation Accuracy for k = 27 is 86.62% and auc is 0.87
Cross Validation Accuracy for k = 29 is 86.65% and auc is 0.87
```

```
In [107]: import matplotlib.pyplot as plt

          # Plotting graph of auc and parameter

          plt.plot(param_list,auc_list)
          plt.xlabel("Parameter for K-NN")
          plt.ylabel("Area Under Curve")
          plt.title("Plot of parameter Vs AUC ")
          plt.show()
```



In [108]:
```
# Plotting graph of auc and parameter

plt.plot(param_list,acc_list)
plt.xlabel("Parameter for K-NN")
plt.ylabel("Accuracy")
plt.title("Plot of parameter Vs Accuracy ")
plt.show()
```

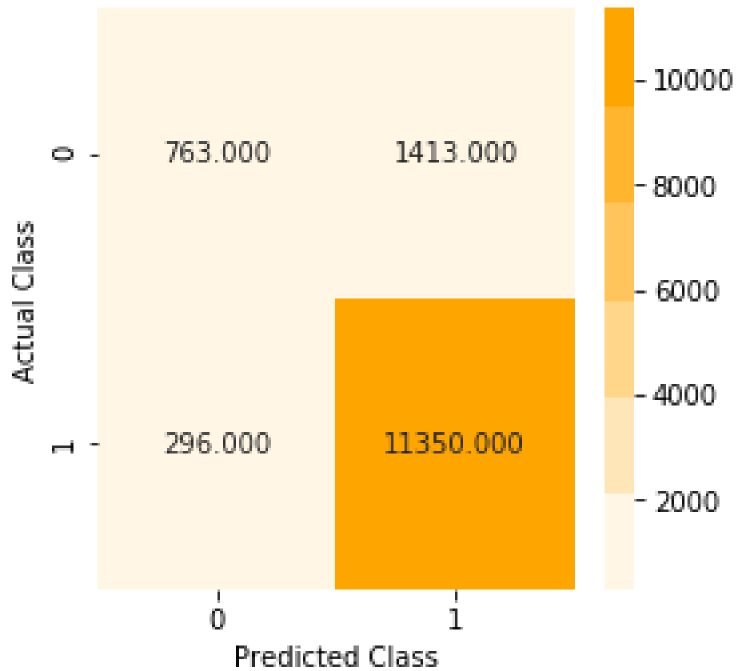


In [118]:
```
# Training the final model
final_clf = KNeighborsClassifier(n_neighbors=21,algorithm='kd_tree',leaf_size=30)

final_clf.fit(train_sent_vectors,Y_train)

predict_y = final_clf.predict(test_sent_vectors)
predict_probab = final_clf.predict_proba(test_sent_vectors)[:,1] # Returns the class

acc = accuracy_score(Y_test,predict_y,normalize=True)*float(100)
auc = roc_auc_score(Y_test,predict_probab)

print("Accuracy of model with k = 21 is {:.2f}% and auc is {:.2f}".format(acc,auc))
```

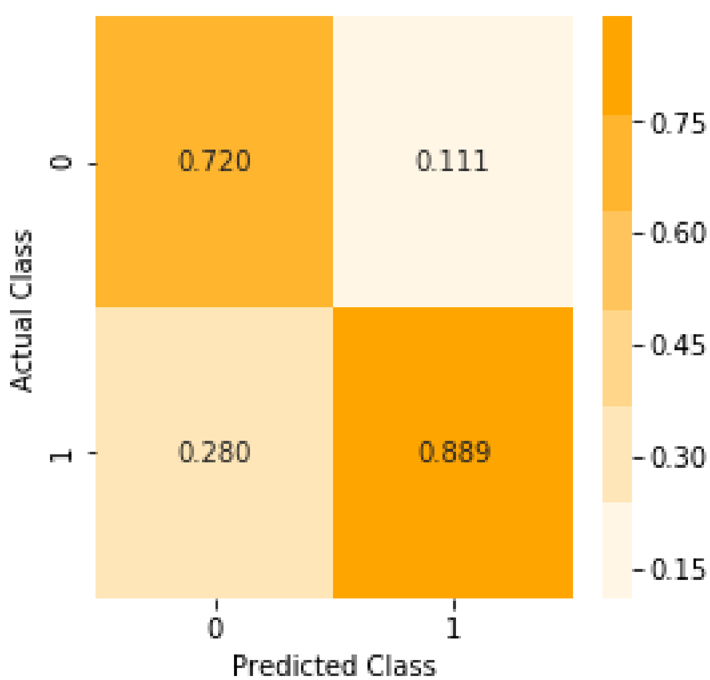
```
Accuracy of model with k = 21 is 87.64% and auc is 0.88
```

In [119]:
```
# Plotting confusion matrix , precision and recall matrix

confusion_matrix_plot(Y_test,predict_y)
```

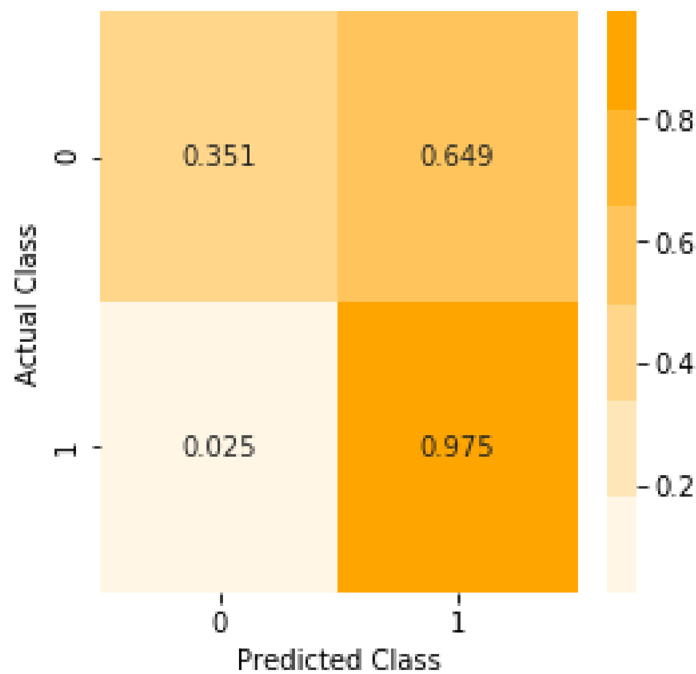
```
=========== Confusion matrix ==========
```



=========== Precision Matrix ==========



```
=========== Recall Matrix =============
```

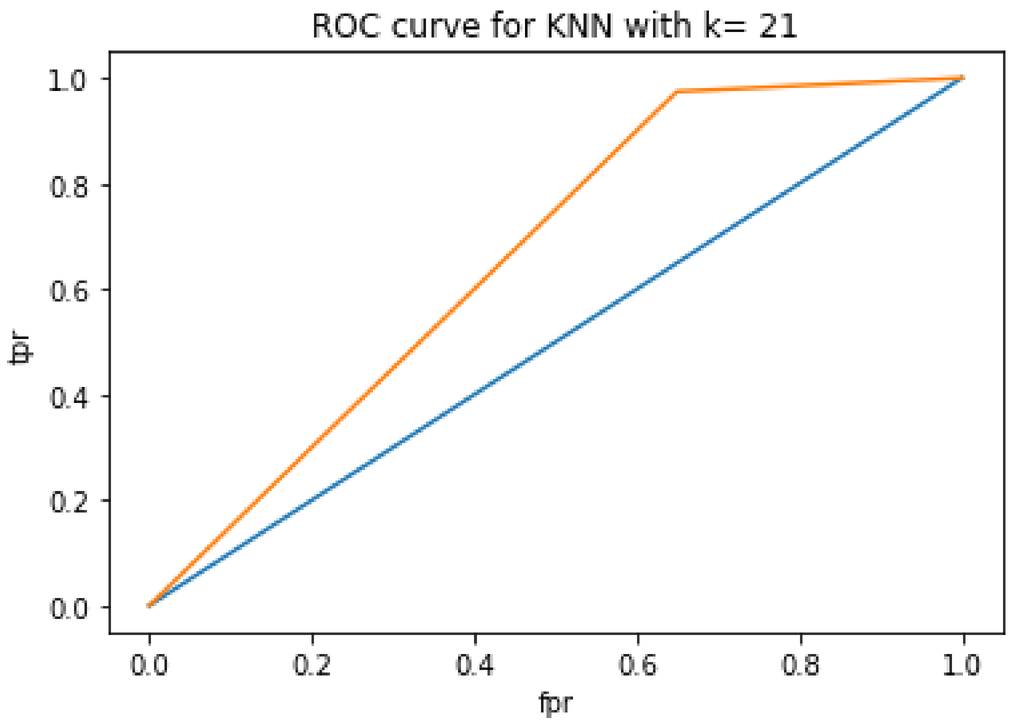



```
In [120]: # Printing the classification report
          print(classification_report(Y_test,predict_y))
```

```
             precision    recall  f1-score   support

          0       0.72      0.35      0.47      2176
          1       0.89      0.97      0.93     11646

avg / total       0.86      0.88      0.86     13822
```

```
In [121]: # Plotting ROC Curve
          from sklearn.metrics import roc_curve

          fpr,tpr,threshold = roc_curve(Y_test,predict_y)

          plt.plot([0,1],[0,1])
          plt.plot(fpr,tpr)
          plt.xlabel("fpr")
          plt.ylabel("tpr")
          plt.title("ROC curve for KNN with k= 21")
```

```
plt.show()

# Printing area under curve

print("Area under ROC curve is ={:.3f} ".format(roc_auc_score(Y_test,predict_probab))
```




```
Area under ROC curve is =0.878
```

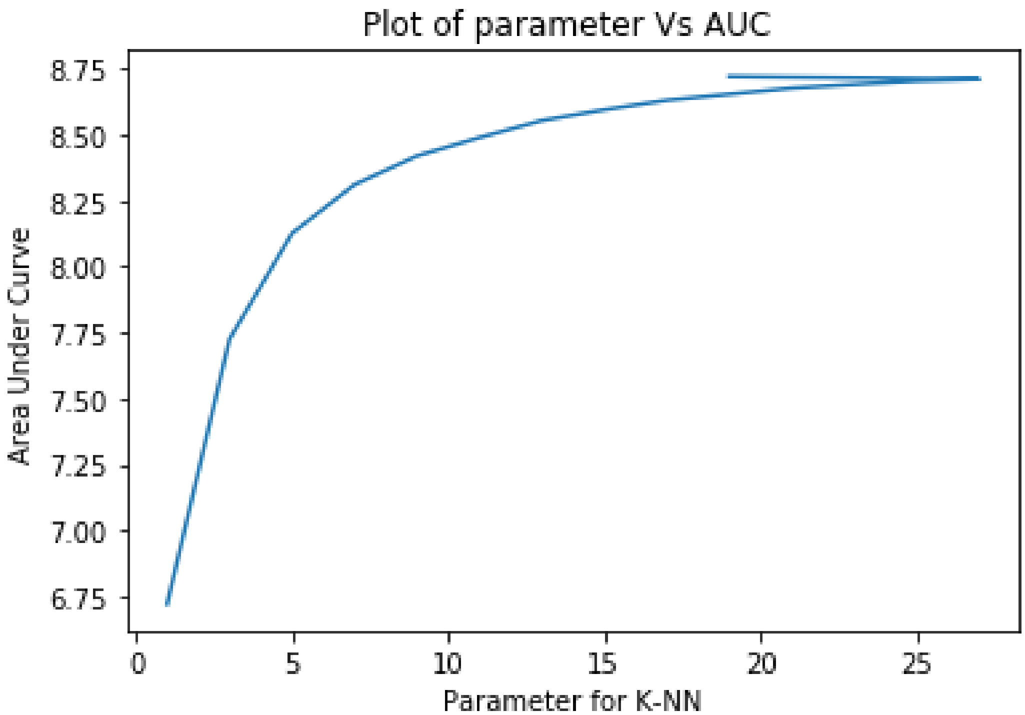
7.1.4 [5.2.4] Applying KNN kd-tree on TFIDF W2V, SET 4

```
In [163]: # 10 fold cross validation on time series splitting
          # Here we are using vectorized dataset from previous section

          acc_list = []
          auc_list = []

          for k in range(1,30,2):
              # KNN Classifier
              clf = KNeighborsClassifier(n_neighbors=k,algorithm='kd_tree',leaf_size=40)
              i=0
              acc=0.0
              auc=0.0
              for train_index,test_index in tscv.split(train_tfidf_sent_vectors):
```

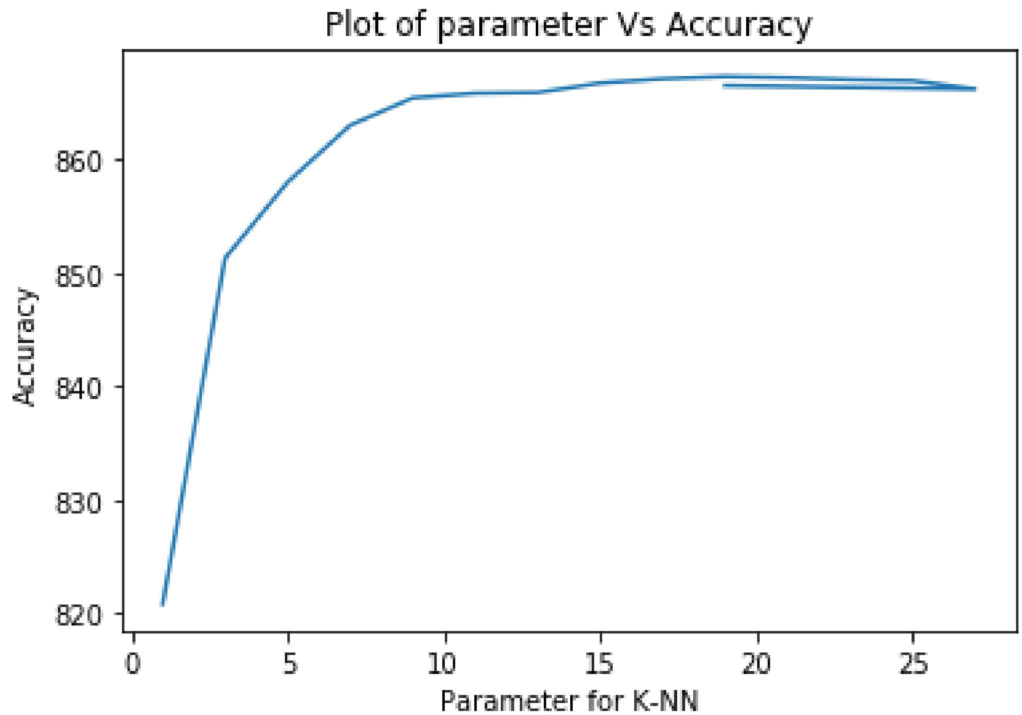
```
            x_train = train_sent_vectors[0:train_index[-1]][:] # row 0 to train_index(ex
            y_train = Y_train[0:train_index[-1]][:] # row 0 to train_index(excluding)
            x_test = train_sent_vectors[train_index[-1]:test_index[-1]][:] # row from tr
            y_test = Y_train[train_index[-1]:test_index[-1]][:] # row from train_index t

            clf.fit(x_train,y_train)

            predict_y = clf.predict(x_test)
            predict_probab = clf.predict_proba(x_test)[:,1]
            i += 1
            acc += accuracy_score(y_test,predict_y,normalize=True) * float(100)
            auc += roc_auc_score(y_test,predict_probab)

        acc_list.append(acc)
        auc_list.append(auc)
        print("Cross Validation Accuracy for k = {:d} is {:.2f}% and auc is {:.2f}".forma
```

```
Cross Validation Accuracy for k = 1 is 82.08% and auc is 0.67
Cross Validation Accuracy for k = 3 is 85.13% and auc is 0.77
Cross Validation Accuracy for k = 5 is 85.80% and auc is 0.81
Cross Validation Accuracy for k = 7 is 86.30% and auc is 0.83
Cross Validation Accuracy for k = 9 is 86.54% and auc is 0.84
Cross Validation Accuracy for k = 11 is 86.58% and auc is 0.85
Cross Validation Accuracy for k = 13 is 86.59% and auc is 0.86
Cross Validation Accuracy for k = 15 is 86.67% and auc is 0.86
Cross Validation Accuracy for k = 17 is 86.71% and auc is 0.86
Cross Validation Accuracy for k = 19 is 86.73% and auc is 0.87
Cross Validation Accuracy for k = 21 is 86.72% and auc is 0.87
Cross Validation Accuracy for k = 23 is 86.70% and auc is 0.87
Cross Validation Accuracy for k = 25 is 86.69% and auc is 0.87
Cross Validation Accuracy for k = 27 is 86.62% and auc is 0.87
Cross Validation Accuracy for k = 29 is 86.65% and auc is 0.87
```

In [164]:
```
# Plotting graph of auc and parameter

plt.plot(param_list,auc_list)
plt.xlabel("Parameter for K-NN")
plt.ylabel("Area Under Curve")
plt.title("Plot of parameter Vs AUC ")
plt.show()
```



In [165]:
```
# Plotting graph of acc and parameter

plt.plot(param_list,acc_list)
plt.xlabel("Parameter for K-NN")
plt.ylabel("Accuracy")
plt.title("Plot of parameter Vs Accuracy")
plt.show()
```

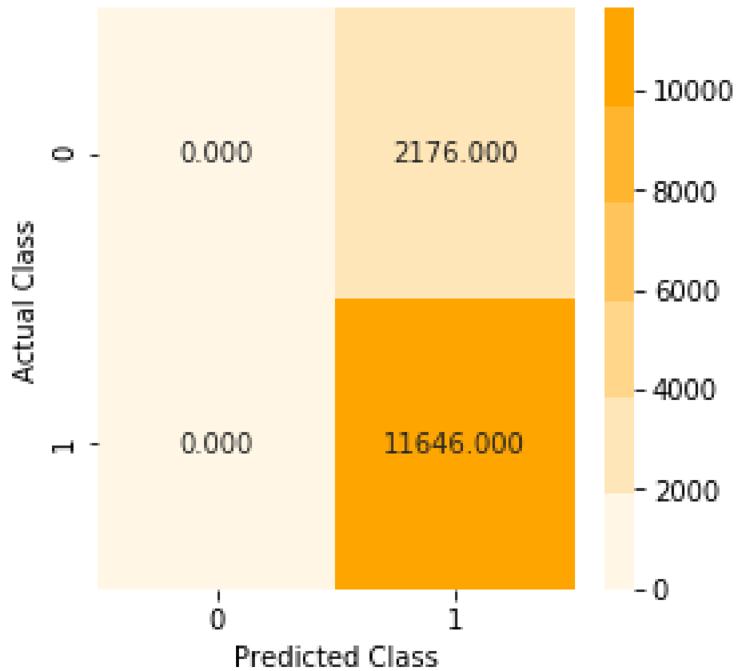


In [170]:
```
# Training the final model
final_clf = KNeighborsClassifier(n_neighbors=21,algorithm='kd_tree',leaf_size=40)

final_clf.fit(train_tfidf_sent_vectors,Y_train)

predict_y = final_clf.predict(test_tfidf_sent_vectors)
predict_probab = final_clf.predict_proba(test_tfidf_sent_vectors)[:,1] # Returns the

acc = accuracy_score(Y_test,predict_y,normalize=True)*float(100)
auc = roc_auc_score(Y_test,predict_probab)

print("Accuracy of model with k = 21 is {:.2f}% and auc is {:.2f}".format(acc,auc))
```

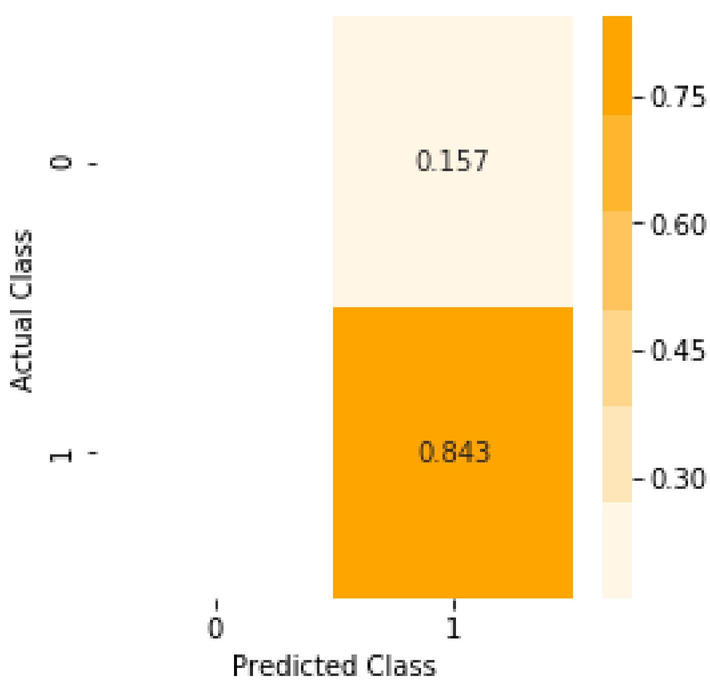
```
Accuracy of model with k = 21 is 84.26% and auc is 0.50
```

In [171]:
```
# Plotting confusion matrix , precision and recall matrix

confusion_matrix_plot(Y_test,predict_y)

# Printing the classification report
print(classification_report(Y_test,predict_y))
```

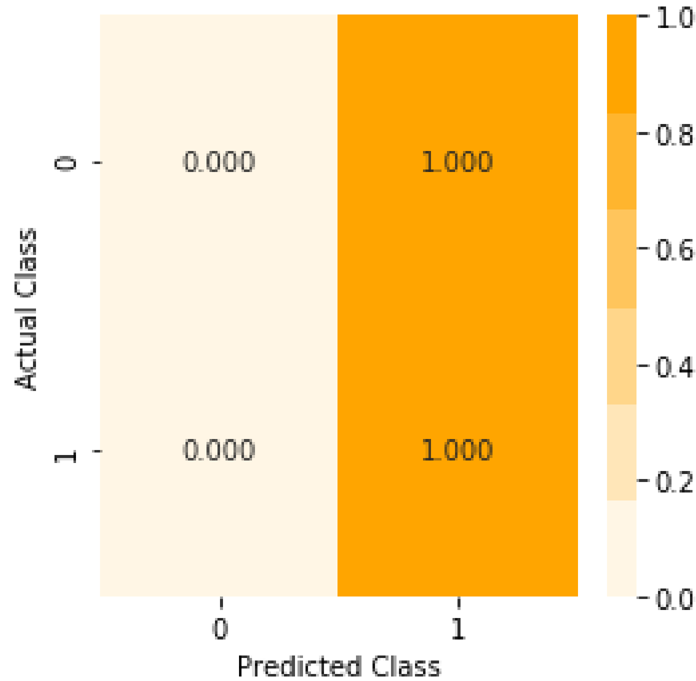
```
=========== Confusion matrix ==========
```




```
=========== Precision Matrix ==========
```



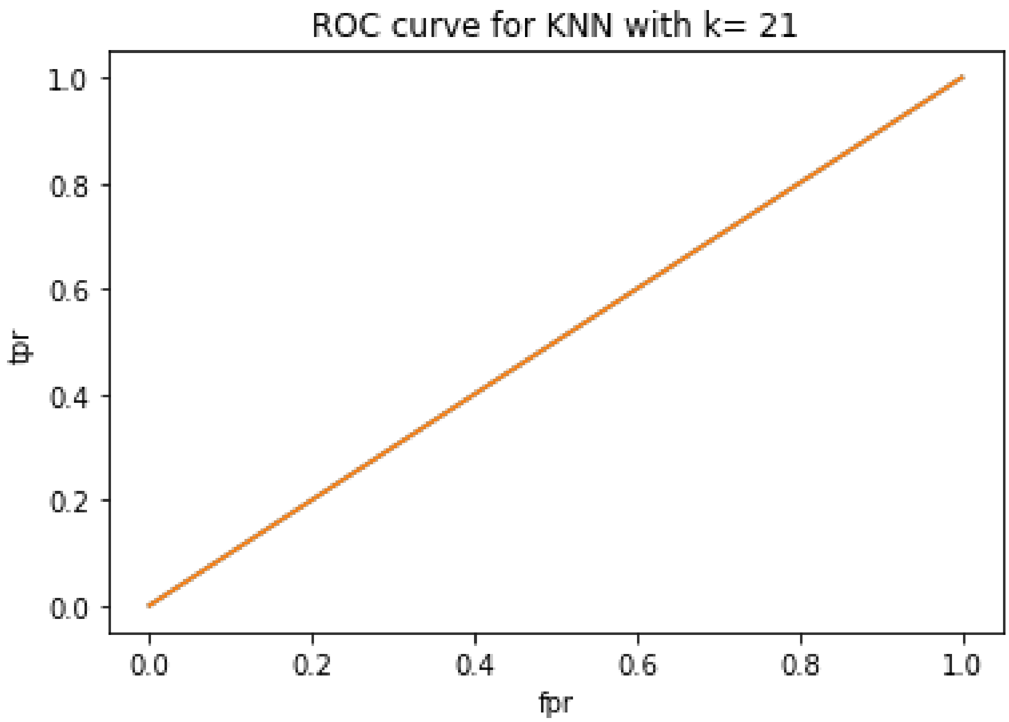
```
=========== Recall Matrix =============
```




```
             precision    recall  f1-score   support

          0       0.00      0.00      0.00      2176
          1       0.84      1.00      0.91     11646

avg / total       0.71      0.84      0.77     13822
```

In [172]:
```python
# Plotting ROC Curve
from sklearn.metrics import roc_curve

fpr,tpr,threshold = roc_curve(Y_test,predict_y)

plt.plot([0,1],[0,1])
plt.plot(fpr,tpr)
plt.xlabel("fpr")
plt.ylabel("tpr")
plt.title("ROC curve for KNN with k= 21")
plt.show()

# Printing area under curve
```

```
print("Area under ROC curve is ={:.3f} ".format(roc_auc_score(Y_test,predict_probab)
```




```
Area under ROC curve is =0.500
```

The model that we have trained are performing good on positive class but not that good on negative class

Therefore we are trying to analyse positive and negative reviews seperately and remove some most common words.

```
In [31]: # Seperating positive reviews and negative reviews for analysis
         positive_review = final[final.Score == 1]
         negative_review = final[final.Score == 0]
         print(positive_review.shape)
         print(negative_review.shape)
```

```
(38479, 11)
(7592, 11)
```

```
In [32]: positive_text = positive_review["Cleaned_review"]
         negative_text = negative_review["Cleaned_review"]
         print(positive_text.shape)
         print(negative_text.shape)
```

```
(38479,)
(7592,)
```

```
In [33]: pos_word_count = {}
         for sent in positive_text:
             for word in sent.split():
                 if word not in pos_word_count:
                     pos_word_count[word] = 1
                 else:
                     pos_word_count[word] += 1
```

```
In [60]: import collections
         pos_word_counter = collections.Counter(pos_word_count)

         words_most_com = []
         words_least_com = []

         for word,count in pos_word_counter.most_common(200):
             words_most_com.append(word)
             print(word,count)
```

```
not 35145
like 15694
good 13981
great 12935
one 11207
taste 10185
tea 9412
love 9368
coffee 9368
would 9174
flavor 9113
product 8827
food 7418
get 6950
really 6542
best 5976
amazon 5860
much 5799
use 5786
also 5730
time 5701
little 5625
price 5289
find 5120
make 5116
chocolate 5111
```

well 4988
tried 4987
buy 4758
dog 4519
try 4510
better 4448
even 4385
eat 4232
first 3782
sugar 3726
found 3651
cup 3596
drink 3528
bag 3519
used 3499
water 3473
sweet 3406
delicious 3383
free 3368
could 3352
bit 3251
made 3216
day 3166
favorite 3121
recommend 3121
store 3098
two 3065
way 3056
since 3038
bought 3027
mix 3005
nice 2982
think 2921
hot 2868
give 2857
dogs 2848
loves 2834
tastes 2828
treats 2809
many 2783
order 2782
still 2724
every 2711
makes 2703
always 2668
flavors 2659
perfect 2646
add 2642

without 2633
know 2619
easy 2617
box 2605
organic 2597
years 2584
want 2514
right 2495
milk 2485
got 2455
healthy 2443
never 2427
lot 2397
quality 2370
keep 2352
brand 2332
treat 2324
stuff 2304
ever 2297
less 2294
snack 2286
fresh 2200
enough 2183
small 2179
chips 2168
say 2164
enjoy 2161
something 2113
need 2102
definitely 2101
salt 2101
put 2092
cat 2074
eating 2069
old 2044
regular 2041
different 2035
happy 2022
using 2007
long 1998
tasty 1984
high 1963
highly 1962
far 1960
strong 1931
back 1930
wonderful 1929
excellent 1921

size 1915
buying 1913
hard 1911
whole 1908
ordered 1904
local 1898
sure 1896
low 1895
products 1886
ingredients 1852
popcorn 1840
green 1832
dark 1829
though 1817
chicken 1811
oil 1809
however 1786
thing 1771
big 1760
natural 1760
shipping 1745
calories 1733
pretty 1725
work 1722
looking 1704
take 1702
quite 1688
see 1683
fat 1683
butter 1668
diet 1643
people 1624
year 1619
actually 1617
last 1616
rice 1616
foods 1610
gluten 1608
stores 1595
texture 1580
pack 1554
feel 1552
package 1551
bags 1549
around 1542
per 1539
cats 1536
new 1533

```
full 1533
anything 1513
tasting 1499
another 1499
cups 1493
bread 1486
almost 1485
sauce 1480
real 1472
brands 1469
bar 1468
purchase 1459
going 1450
grocery 1448
family 1448
worth 1446
cookies 1446
expensive 1440
half 1437
loved 1434
added 1425
may 1422
bars 1414
protein 1393
usually 1392
morning 1388
home 1387
getting 1386
tasted 1381
fruit 1354
```

In [64]:
```
# Collecting most important positive words together
imp_pos_words = ['good','great','love','loves','loved','best','really','well','better
                 'recommend','nice','loves','tastes','treats','perfect','organic','en
                 'wonderful','excellent','pretty']
```

In [61]:
```
# Calculating average len of positive reviews
len_sum = 0
for sent in positive_text:
    len_sum += len(sent.split())

print("avg len of reviews",len_sum/38479)
```

```
avg len of reviews 37.480235972868314
```

In [62]:
```
len_sum = 0
for sent in negative_text:
```

```
            len_sum += len(sent.split())

        print("avg len of reviews",len_sum/38479)
```

```
avg len of reviews 8.557524883702799
```

```
In [35]: neg_word_count = {}
         for sent in negative_text:
             for word in sent.split():
                 if word not in neg_word_count:
                     neg_word_count[word] = 1
                 else:
                     neg_word_count[word] += 1
```

```
In [63]: import collections
         neg_word_counter = collections.Counter(neg_word_count)

         for word,count in neg_word_counter.most_common(200):
             print(word,count)
```

```
not 12513
like 4189
would 3254
taste 3114
product 2989
one 2498
good 1953
flavor 1869
coffee 1737
food 1557
even 1529
get 1527
tea 1497
amazon 1348
buy 1334
much 1318
really 1307
could 1253
first 1116
tried 1103
dog 1095
time 1092
bought 1073
box 1033
water 1027
better 1002
bad 994
made 987
```

try 946
eat 933
chocolate 927
also 881
know 875
great 855
bag 837
use 833
love 830
drink 820
two 816
sugar 813
make 808
little 805
thought 799
got 787
way 764
price 751
well 747
used 744
something 740
never 739
ordered 731
still 727
think 722
order 722
tastes 716
find 695
back 663
disappointed 663
cup 663
old 654
money 650
ingredients 631
away 627
brand 626
found 616
products 608
tasted 608
however 601
since 597
give 578
stuff 575
received 570
package 568
different 567
dogs 566
store 566

say 564
many 553
reviews 550
new 547
sweet 539
see 537
sure 531
want 531
hot 526
another 516
company 511
item 510
coconut 507
smell 504
mix 499
ever 494
treats 494
going 492
quality 491
ginger 491
maybe 474
purchased 473
may 472
people 471
buying 470
thing 466
bit 463
less 461
free 458
almost 455
recommend 455
cans 453
looking 451
hard 443
anything 443
small 443
bags 442
purchase 440
best 433
nothing 429
organic 426
though 425
actually 424
eating 424
flavors 421
put 416
lot 415
natural 411

texture 411
said 409
last 408
half 407
whole 407
real 406
high 391
salt 390
strong 390
shipping 388
cat 386
chicken 385
problem 385
milk 379
day 378
plastic 375
oil 372
pack 371
gave 371
right 370
years 369
opened 368
trying 367
review 365
look 364
pretty 363
enough 357
using 357
cookies 352
cups 351
every 350
juice 350
case 348
chips 348
take 347
work 345
regular 344
tasting 342
big 342
might 341
read 339
waste 338
far 336
bottle 334
wanted 332
either 331
boxes 331
local 330

```
packaging 329
low 329
awful 327
popcorn 324
horrible 322
probably 321
worth 319
size 319
dry 317
came 317
green 317
long 316
bitter 316
instead 311
jerky 310
least 309
kind 309
return 306
without 298
three 296
getting 296
took 295
went 290
rather 288
several 288
unfortunately 288
need 286
worst 286
```

In [65]:
```
# Collecting important negative words
neg_imp_words = ['not','like','bad','great','love','disappointed','different','best',
                 'bitter','jerky','return','unfortunately','worst']
```

Observations

Clearly we can see that top most occuring word in both positive and negative reviews are "not" and "like".

In [67]:
```
# Therefore we are removing "not" word from positive review set and like "from" both
# And after this we will analyse how does our models perform

cleaned_pos_text = []
for sent in positive_text:
    sentence = []
    s = ""
    for word in sent.split():
        if(word == "not"):
            continue
```

```
            if(word in imp_pos_words):
                sentence.append(word)
            else:
                continue
        s = " ".join(e.lower() for e in sentence)
        cleaned_pos_text.append(s.strip())

    print(len(cleaned_pos_text))
```

38479

```
In [68]: # For negative reviews
         cleaned_neg_text = []
         for sent in negative_text:
             sentence = []
             for word in sent.split():
                 if(word == "like"):
                     continue
                 if(word in neg_imp_words):
                     sentence = "".join(word.lower())
                 else:
                     continue

             s = " ".join(e.lower() for e in sentence)
             cleaned_neg_text.append(s.strip())


         print(len(cleaned_neg_text))
```

7592

```
In [69]: # Storing class labels of both positive and negative data points

         positive_score = positive_review["Score"]
         negative_score = negative_review["Score"]

         # Now we will combine positive and negative reviews

         review_lst = cleaned_pos_text + cleaned_neg_text

         review_label = list(positive_score) + list(negative_score)
```

```
In [70]: # Converting them into dataframe
         final_dataset = pd.DataFrame({"Text":review_lst , "Class":review_label})
         final_dataset.head()
```

Out[70]:

	Class	Text
0	1	love

```
1      1       pretty
2      1        great
3      1         love
4      1  really good
```

```
In [71]: from sklearn.model_selection import train_test_split

         # Splitting data into train and test
         X = final_dataset["Text"]
         Y = final_dataset["Class"]

         X_train,X_test,Y_train,Y_test = train_test_split(X,Y,test_size=0.3,random_state=42)
         print(X_train.shape,X_test.shape)
         print(Y_train.value_counts(),Y_test.value_counts())
```

```
(32249,) (13822,)
1    26945
0     5304
Name: Class, dtype: int64 1    11534
0     2288
Name: Class, dtype: int64
```

Using bag of words

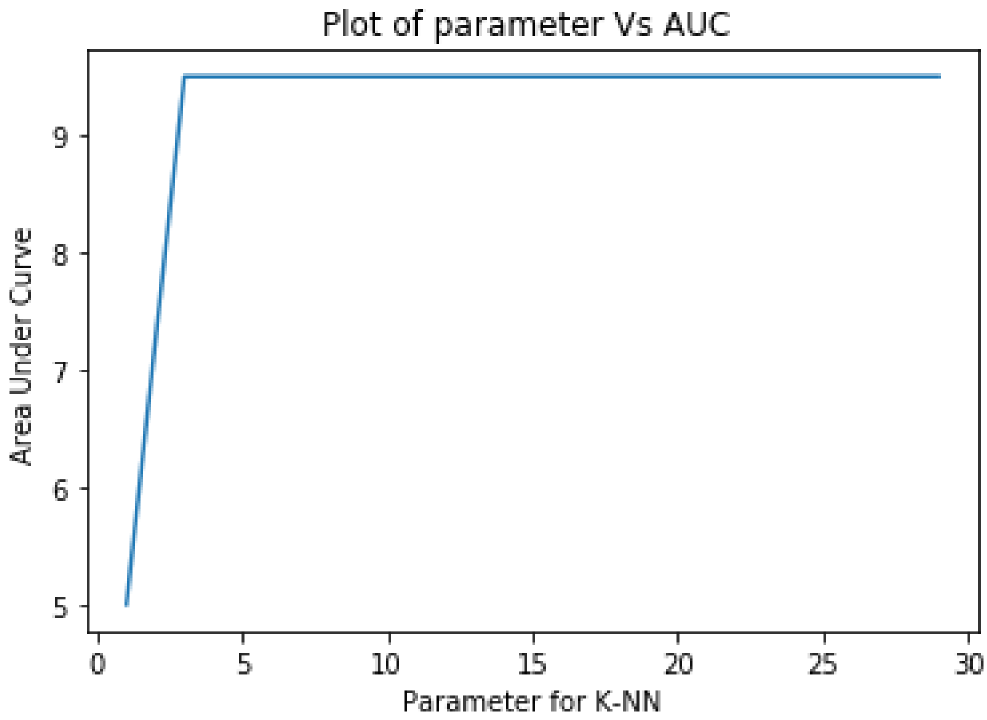
```
In [72]: # BagOfWords for bigrams and unigrams
         bow_model = CountVectorizer(ngram_range=(1,2)) # Initializing the model

         # vectorizing train dataset
         vect_train = bow_model.fit_transform(X_train)

         # vectorizing test dataset
         vect_test = bow_model.transform(X_test)
```

```
In [73]: from sklearn.neighbors import KNeighborsClassifier
         from sklearn.model_selection import TimeSeriesSplit
         from sklearn.metrics import accuracy_score
         from sklearn.metrics import roc_auc_score


         # Initializing time series splitter
         tscv = TimeSeriesSplit(n_splits=10)

         # Now we will perform 10 fold cross validation on time split data

         acc_list = []
         auc_list = []

         for k in range(1,30,2):
```

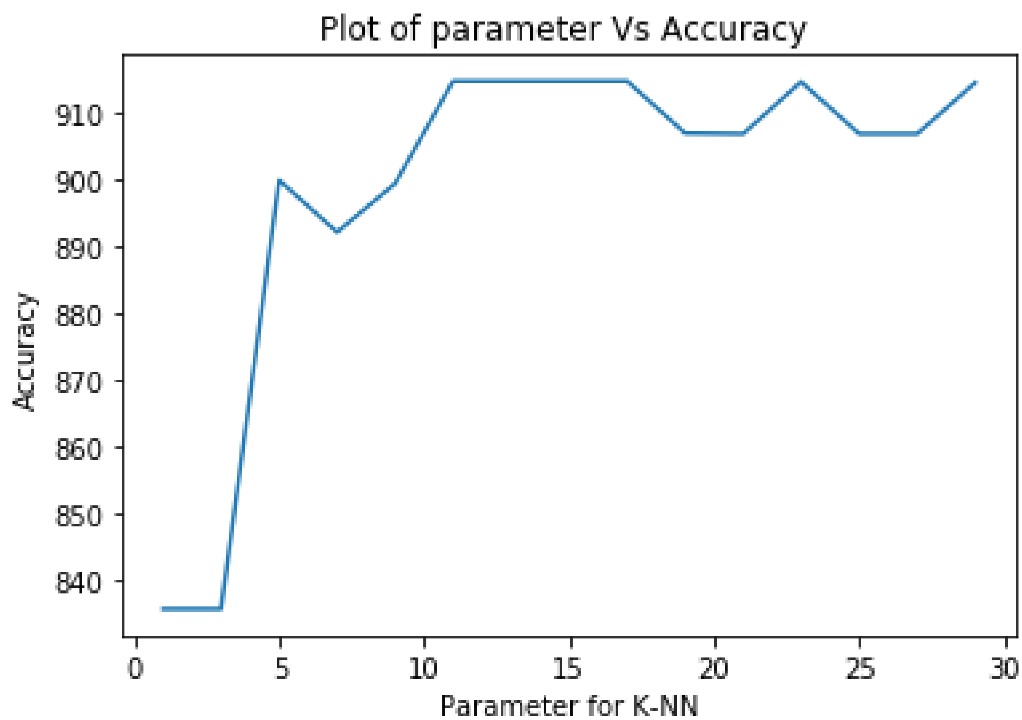
```
            # KNN Classifier
            clf = KNeighborsClassifier(n_neighbors=k,algorithm='brute',leaf_size=40)
            i=0
            acc=0.0
            auc=0.0
            for train_index,test_index in tscv.split(vect_train):
                x_train = vect_train[0:train_index[-1]][:] # row 0 to train_index(excluding)
                y_train = Y_train[0:train_index[-1]][:] # row 0 to train_index(excluding)
                x_test = vect_train[train_index[-1]:test_index[-1]][:] # row from train_index
                y_test = Y_train[train_index[-1]:test_index[-1]][:] # row from train_index to

                clf.fit(x_train,y_train)

                predict_y = clf.predict(x_test)
                predict_probab = clf.predict_proba(x_test)[:,1]
                i += 1
                acc += accuracy_score(y_test,predict_y,normalize=True) * float(100)
                auc += roc_auc_score(y_test,predict_probab)

            acc_list.append(acc)
            auc_list.append(auc)
            print("Cross Validation Accuracy for k = {:d} is {:.2f}% and auc is {:.2f}".forma
```

```
Cross Validation Accuracy for k = 1 is 83.57% and auc is 0.50
Cross Validation Accuracy for k = 3 is 83.57% and auc is 0.95
Cross Validation Accuracy for k = 5 is 89.98% and auc is 0.95
Cross Validation Accuracy for k = 7 is 89.21% and auc is 0.95
Cross Validation Accuracy for k = 9 is 89.93% and auc is 0.95
Cross Validation Accuracy for k = 11 is 91.46% and auc is 0.95
Cross Validation Accuracy for k = 13 is 91.46% and auc is 0.95
Cross Validation Accuracy for k = 15 is 91.46% and auc is 0.95
Cross Validation Accuracy for k = 17 is 91.46% and auc is 0.95
Cross Validation Accuracy for k = 19 is 90.69% and auc is 0.95
Cross Validation Accuracy for k = 21 is 90.68% and auc is 0.95
Cross Validation Accuracy for k = 23 is 91.45% and auc is 0.95
Cross Validation Accuracy for k = 25 is 90.68% and auc is 0.95
Cross Validation Accuracy for k = 27 is 90.68% and auc is 0.95
Cross Validation Accuracy for k = 29 is 91.44% and auc is 0.95
```

In [74]:
```
# Plotting graph of auc and parameter

param_list = [1,3,5,7,9,11,13,15,17,19,21,23,25,27,29]

plt.plot(param_list,auc_list)
plt.xlabel("Parameter for K-NN")
plt.ylabel("Area Under Curve")
plt.title("Plot of parameter Vs AUC ")
plt.show()
```



In [75]:
```
# Plotting graph of accuracy and parameter

plt.plot(param_list,acc_list)
plt.xlabel("Parameter for K-NN")
plt.ylabel("Accuracy")
plt.title("Plot of parameter Vs Accuracy ")
plt.show()
```

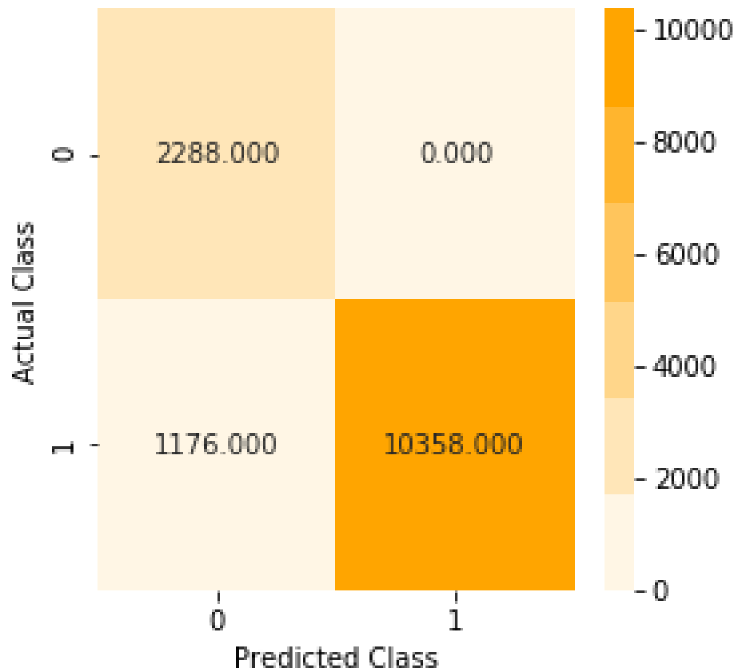


In [77]:
```
# Training the final model
final_clf = KNeighborsClassifier(n_neighbors=13,algorithm='brute',leaf_size=40)

final_clf.fit(vect_train,Y_train)

predict_y = final_clf.predict(vect_test)
predict_probab = final_clf.predict_proba(vect_test)[:,1] # Returns the class probabal

acc = accuracy_score(Y_test,predict_y,normalize=True)*float(100)
auc = roc_auc_score(Y_test,predict_probab)

print("Accuracy of model with k = 13 is {:.2f}% and auc is {:.2f}".format(acc,auc))
```

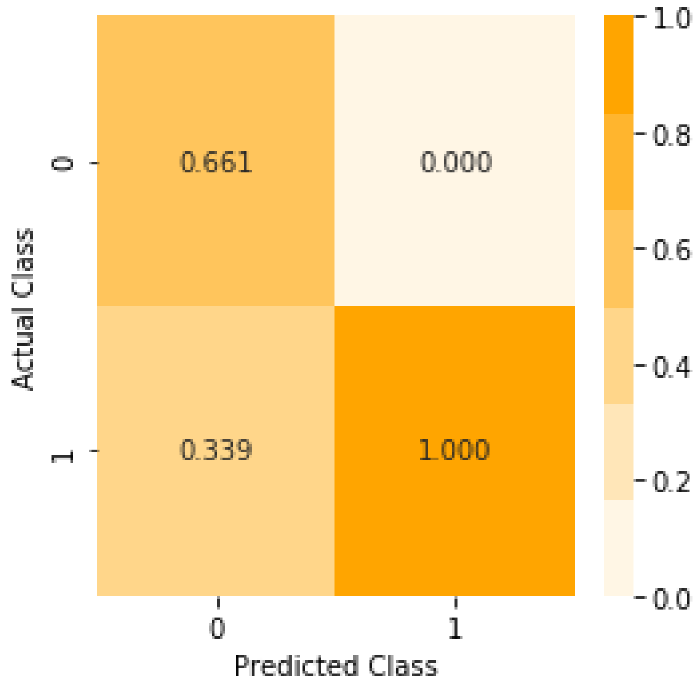
```
Accuracy of model with k = 13 is 91.49% and auc is 0.95
```

In [80]:
```
from sklearn.metrics import classification_report
# Plotting confusion matrix , precision and recall matrix

confusion_matrix_plot(Y_test,predict_y)

# Printing the classification report
print(classification_report(Y_test,predict_y))
```

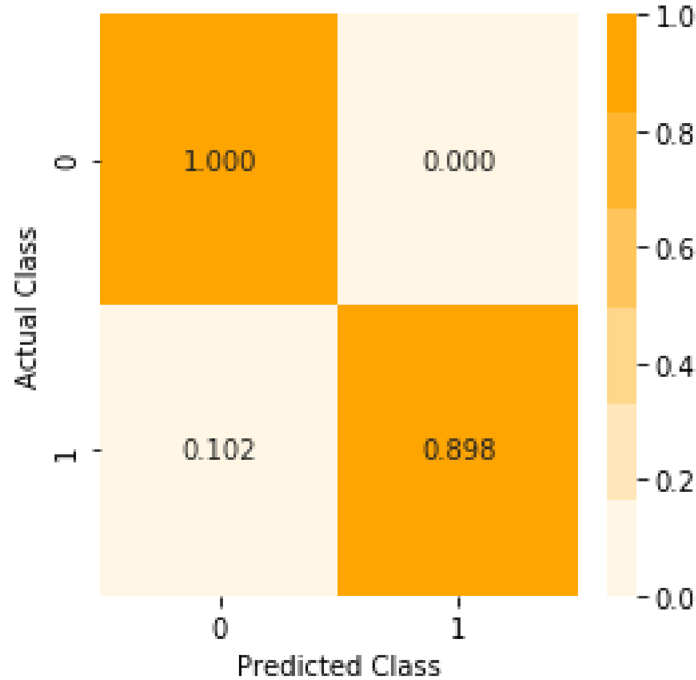
=========== Confusion matrix ==========



=========== Precision Matrix ==========



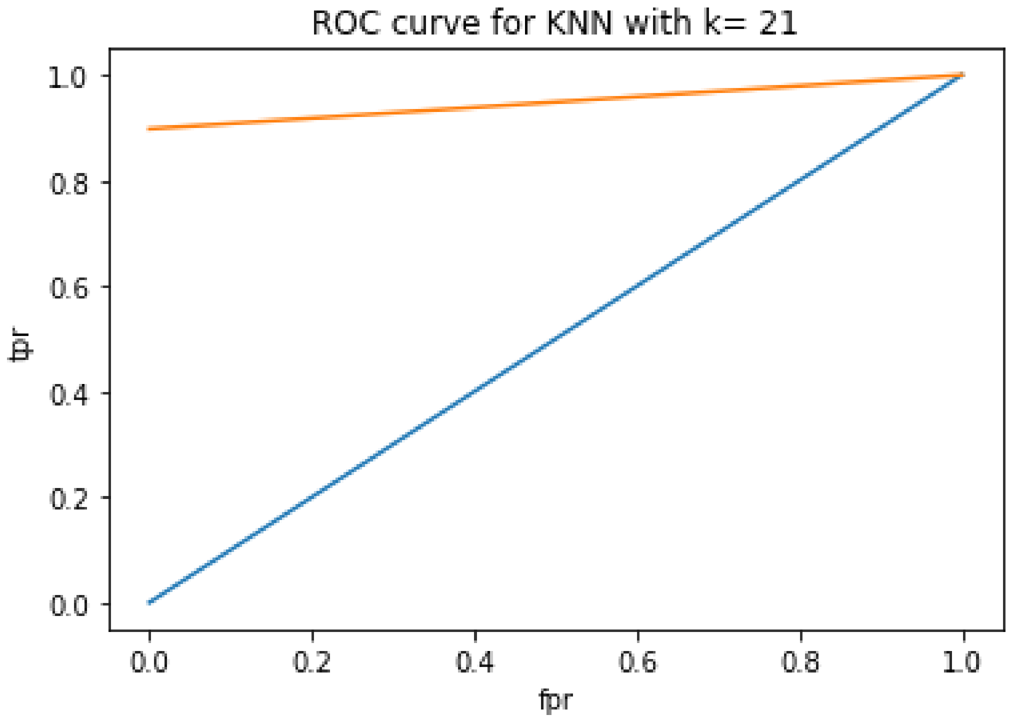
```
=========== Recall Matrix =============
```




```
             precision    recall  f1-score   support

          0       0.66      1.00      0.80      2288
          1       1.00      0.90      0.95     11534

avg / total       0.94      0.91      0.92     13822
```

```
In [81]: # Plotting ROC Curve
         from sklearn.metrics import roc_curve

         fpr,tpr,threshold = roc_curve(Y_test,predict_y)

         plt.plot([0,1],[0,1])
         plt.plot(fpr,tpr)
         plt.xlabel("fpr")
         plt.ylabel("tpr")
         plt.title("ROC curve for KNN with k= 21")
```

```
plt.show()

# Printing area under curve

print("Area under ROC curve is ={:.3f} ".format(roc_auc_score(Y_test,predict_probab))
```




```
Area under ROC curve is =0.949
```

Observation
Surpringly the accuracy and area under ROC increased drastically when only most important words were used as features

8 [6] Conclusions

```
In [95]: from prettytable import PrettyTable

         x = PrettyTable()


         x.field_names = ["Model Type","Best K","Accuracy","AUC","Precision","Recall","f1-score

         x.add_row(["BOW","21","84.05%","0.67","0.80","0.84","0.80"])
```

```python
x.add_row(["TfIdf","25","84.26%","0.51","0.71","0.84","0.77"])
x.add_row(["Avg W2V","17","87.77","0.87","0.86","0.88","0.86"])
x.add_row(["TfIdf Weighted W2V","21","84.26","0.50","0.71","0.84","0.77"])
x.add_row(["BOW on improved features","13","91.49","0.95","0.94","0.91","0.92"])
print(x)
```

```
+--------------------------+--------+----------+------+-----------+--------+----------+
|        Model Type        | Best K | Accuracy | AUC  | Precision | Recall | f1-score |
+--------------------------+--------+----------+------+-----------+--------+----------+
|           BOW            |   21   |  84.05%  | 0.67 |    0.80   |  0.84  |   0.80   |
|          TfIdf           |   25   |  84.26%  | 0.51 |    0.71   |  0.84  |   0.77   |
|         Avg W2V          |   17   |  87.77   | 0.87 |    0.86   |  0.88  |   0.86   |
|    TfIdf Weighted W2V    |   21   |  84.26   | 0.50 |    0.71   |  0.84  |   0.77   |
| BOW on improved features |   13   |  91.49   | 0.95 |    0.94   |  0.91  |   0.92   |
+--------------------------+--------+----------+------+-----------+--------+----------+
```