[1]. Reading Data

[1.1] Loading the data

The dataset is available in two forms

1. .csv file
2. SQLite Database

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

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

In [1]:

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


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

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

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

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

from tqdm import tqdm
import os
```

In [2]:

```
# using SQLite Table to read data.
con = sqlite3.connect('D:/AAIC/Data_Sets/amazon-fine-food-reviews/database.sqlite')

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

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

filtered_data = pd.read_sql_query(""" SELECT * FROM Reviews WHERE Score != 3 LIMIT 55000""", con)

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

```
    if x < 3:
        return 0
    return 1

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

```
Number of data points in our data (55000, 10)
```

Out[2]:

	Id	ProductId	UserId	ProfileName	HelpfulnessNumerator	HelpfulnessDenominator	Score	Time
0	1	B001E4KFG0	A3SGXH7AUHU8GW	delmartian	1	1	1	130386240(
1	2	B00813GRG4	A1D87F6ZCVE5NK	dll pa	0	0	0	134697600(
2	3	B000LQOCH0	ABXLMWJIXXAIN	Natalia Corres "Natalia Corres"	1	1	1	121901760(

In [3]:

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

In [4]:

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

```
(80668, 7)
```

Out[4]:

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

	UserId	ProductId	ProfileName	Time	Score	Text	COUNT(*)
4	#oc-R12KPBODL2B5ZD	B007OSBE1U	Christopher P. Presta	1348617600	5	I didnt like this coffee. Instead of telling y...	2

In [5]:

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

Out[5]:

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

In [6]:

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

Out[6]:

```
393063
```

[2] Exploratory Data Analysis

[2.1] Data Cleaning: Deduplication

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

In [7]:

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

Out[7]:

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

	Id	ProductId	UserId	ProfileName	HelpfulnessNumerator	HelpfulnessDenominator	Score	Ti
4	155049	B000PAQ75C	AR5J8UI46CURR	Geetha Krishnan	2	2	5	1199577

In [8]:

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

In [9]:

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

Out[9]:

```
(50251, 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]:

```
91.36545454545455
```

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

In [11]:

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

display.head()
```

Out[11]:

	Id	ProductId	UserId	ProfileName	HelpfulnessNumerator	HelpfulnessDenominator	Score	Ti
0	64422	B000MIDROQ	A161DK06JJMCYF	J. E. Stephens "Jeanne"	3	1	5	12248928
1	44737	B001EQ55RW	A2V0I904FH7ABY	Ram	3	2	4	12128832

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()
```

```
(50250, 10)
```

Out[13]:

```
1    42026
0     8224
Name: Score, dtype: int64
```

Observations:

1. The given data set is an imbalanced dataset.

[3] Preprocessing

[3.1]. Preprocessing Review Text

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

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

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

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

In [14]:

```
# https://stackoverflow.com/a/47091490/4084039
import re
from bs4 import BeautifulSoup
def decontracted(phrase):
    # specific
    phrase = re.sub(r"won't", "will not", phrase)
    phrase = re.sub(r"can\'t", "can not", phrase)

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

In [15]:

```
# https://gist.github.com/sebleier/554280
```

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

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

In [16]:

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

```
100%|██████████████████████████████████| 50250/50250 [00:25<00:00, 1958.82it/s]
```

In [17]:

```
preprocessed_reviews[1500]
```

Out[17]:

```
'never ever tasted finer delicious chocolate entire life thing would ask leonidas belgian
chocolates make chocolates whole almonds would best worlds'
```

Preparing Data points (into Train and Test)

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

In [273]:

```
#divide preprocessed review data to Train and Test dataset using train_test_split
from sklearn.model_selection import train_test_split
from sklearn import preprocessing

#Breaking into Train and test
```

```
X_train, X_test, Y_train, Y_test = train_test_split(preprocessed_reviews,final['Score'].values,test
_size=0.3,shuffle=False)
#print(X_train.shape, Y_tra)
pd.DataFrame(Y_train)[0].value_counts()
```

Out[273]:

```
1    29552
0     5623
Name: 0, dtype: int64
```

In [274]:

```
NoofDataPoint_Train = len(X_train);
X_train = X_train[:NoofDataPoint_Train]
Y_train = Y_train[:NoofDataPoint_Train]
NoOfDataPoints_Test = len(X_test);
X_test = X_test[0:NoOfDataPoints_Test]
Y_test = Y_test[:NoOfDataPoints_Test]

#Let No of ngramRange be 1 = UniGram
uni_gram = CountVectorizer(max_features=2000)
X_train = uni_gram.fit_transform(X_train)
X_test = uni_gram.transform(X_test)

#Normalizing the  Data
X_train = preprocessing.normalize(X_train)
X_test = preprocessing.normalize(X_test)
```

[4] Featurization

[4.1] BAG OF WORDS

4.1.1 Applying KNN brute force on BOW

In [141]:

```
Max_No_Of_Neighbours = 100;
K_Value_List_odd = list(range(1,Max_No_Of_Neighbours,2))
```

In [275]:

```
X_train_sub, X_cv_bow, Y_train_sub, y_cv = train_test_split(X_train,Y_train,test_size=0.3,shuffle=F
alse)
print(X_train_sub.shape,X_cv_bow.shape)
```

```
(24622, 2000) (10553, 2000)
```

In [30]:

```
def batch_predict(clf, data):
    # roc_auc_score(y_true, y_score) the 2nd parameter should be probability estimates of the posi
tive class
    # not the predicted outputs

    y_data_pred = []
    tr_loop = data.shape[0] - data.shape[0]%1000
    # consider you X_tr shape is 49041, then your cr_loop will be 49041 - 49041%1000 = 49000
    # in this for loop we will iterate until the last 1000 multiplier
    for i in range(0, tr_loop, 1000):
        y_data_pred.extend(clf.predict_proba(data[i:i+1000])[:,1])
    # we will be predicting for the last data points
    y_data_pred.extend(clf.predict_proba(data[tr_loop:])[:,1])

    return y_data_pred
```

In [31]:

```
def batch_predict_val(clf, data):
    # roc_auc_score(y_true, y_score) the 2nd parameter should be probability estimates of the posi
tive class
    # not the predicted outputs

    y_data_pred = []
    tr_loop = data.shape[0] - data.shape[0]%1000
    # consider you X_tr shape is 49041, then your cr_loop will be 49041 - 49041%1000 = 49000
    # in this for loop we will iterate until the last 1000 multiplier
    for i in range(0, tr_loop, 1000):
        y_data_pred.extend(clf.predict(data[i:i+1000])[:,1])
    # we will be predicting for the last data points
    y_data_pred.extend(clf.predict(data[tr_loop:])[:,1])

    return y_data_pred
```

In [26]:

```
%%time
from sklearn.neighbors import KNeighborsClassifier

Y_data_pred_bow_tr=[]
Y_data_pred_bow_cv=[]

for K in tqdm(K_Value_List_odd):
    neigh_bow = KNeighborsClassifier(n_neighbors=K)
    neigh_bow.fit(X_train_sub, Y_train_sub)
    Y_data_pred_bow_tr.append(batch_predict(neigh_bow, X_train_sub))
    Y_data_pred_bow_cv.append(batch_predict(neigh_bow, X_cv_bow))
    # roc_auc_score(y_true, y_score) the 2nd parameter should be probability estimates of the posi
tive class
    # not the predicted outputs
```

```
100%|██████████████████████████████████████████| 50/50 [36:20<00:00, 45.02s/it]
```

```
Wall time: 36min 20s
```

In [217]:

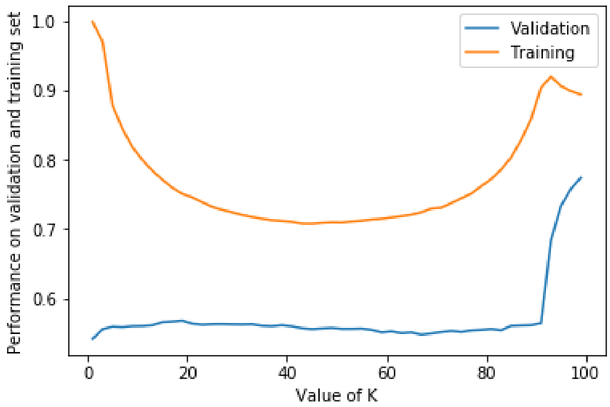
```
from sklearn.metrics import roc_curve, auc

def Score_Provider(Y_train,Y_pred_data_prob_tr, Y_cv,Y_pred_data_prob_cv):
    '''
    input: Y_train             - Actaual output
           Y_pred_data_prob_tr - Predicted probabilities
           Y_cv                - Actual output of test data
           Y_pred_data_prob_cv - Predicted output of test data
    '''
    roc_tr=[]
    auc_tr=[]
    roc_cv=[]
    auc_cv=[]
    for_opt_k =[]

    Y_data_pred_cv_avg=[]
    Y_data_pred_tr_avg=[]

    for i in range(len(K_Value_List_odd)):
        #roc_tr will be having fpr, tpr and thresholds
        roc_tr.append(roc_curve(Y_train,Y_pred_data_prob_tr[i]))
        auc_tr.append(auc(roc_tr[i][0],roc_tr[i][1]))
        roc_cv.append(roc_curve(Y_cv,Y_pred_data_prob_cv[i]))
        auc_cv.append(auc(roc_cv[i][0],roc_cv[i][1]))
        for_opt_k.append(abs(auc_tr[i] - auc_cv[i]))

    K_best_Value = K_Value_List_odd[for_opt_k.index(min(for_opt_k))]
    return K_best_Value, auc_tr, auc_cv, roc_tr, roc_cv

K_Value_bow_bru,auc_bow_tr,auc_bow_cv,roc_bow_tr, roc_bow_cv =
Score_Provider(Y_train_sub,Y_data_pred_bow_tr, y_cv,Y_data_pred_bow_cv)
```

In [33]:

```
plt.plot(K_Value_List_odd, auc_bow_cv, label="Validation")
plt.plot(K_Value_List_odd, auc_bow_tr, label="Training")
plt.xlabel("Value of K");
plt.ylabel("Performance on validation and training set")
plt.legend()
```

Out[33]:

```
<matplotlib.legend.Legend at 0x484b5b8d0>
```



In [25]:

```
from sklearn.neighbors import KNeighborsClassifier

knn_to_fit_the_best = KNeighborsClassifier(n_neighbors=K_Value_bow_bru,algorithm='brute')
knn_to_fit_the_best.fit(X_train, Y_train)
```

Out[25]:

```
KNeighborsClassifier(algorithm='brute', leaf_size=30, metric='minkowski',
           metric_params=None, n_jobs=1, n_neighbors=99, p=2,
           weights='uniform')
```

In [33]:

```
%%time
Y_prob_test = batch_predict(knn_to_fit_the_best, X_test)
Y_prob_train = batch_predict(knn_to_fit_the_best, X_train)
```

```
Wall time: 1min 30s
```

%%time Y_pred_train = batch_predict_val(knn_to_fit_the_best, X_train) Y_pred_test = batch_predict_val(knn_to_fit_the_best, X_test)

In [52]:

```
fpr, tpr, threshholds = roc_curve(Y_train,Y_prob_train)
fpr_test, tpr_test, threshholds_test = roc_curve(Y_test,Y_prob_test)
```

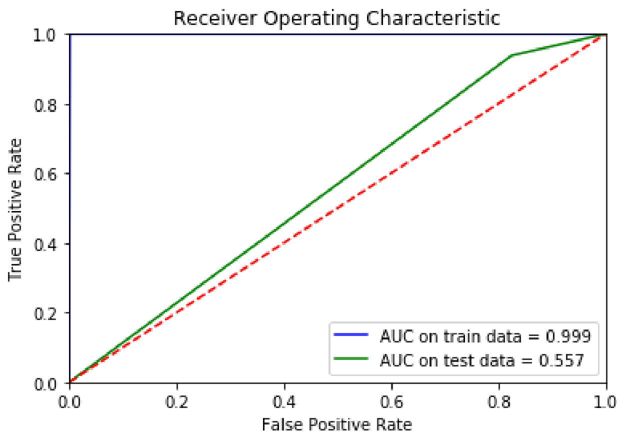
In [53]:

```
from sklearn.metrics import auc

#computing AUC value on prediction over train data
roc_auc_train_bow_br = auc(fpr, tpr)
print(roc_auc_train_bow_br)
#computing AUC value on prediction over test data
roc_auc_test_bow_br = auc(fpr_test, tpr_test)
print(roc_auc_test_bow_br)
```

```
0.9991997154543838
0.5566258787135373
```

In [54]:

```
#Plotting ROC Curve
import matplotlib.pyplot as plt
plt.title('Receiver Operating Characteristic')
plt.plot(fpr, tpr, 'b', label = 'AUC on train data = %0.3f' % roc_auc_train_bow_br)

plt.plot(fpr_test, tpr_test, 'g', label = 'AUC on test data = %0.3f' % roc_auc_test_bow_br)
plt.legend(loc = 'lower right')
plt.plot([0, 1], [0, 1],'r--')
plt.xlim([0, 1])
plt.ylim([0, 1])
plt.ylabel('True Positive Rate')
plt.xlabel('False Positive Rate')
plt.show()
```

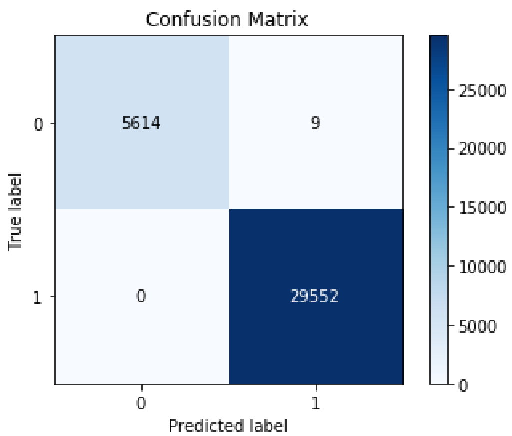


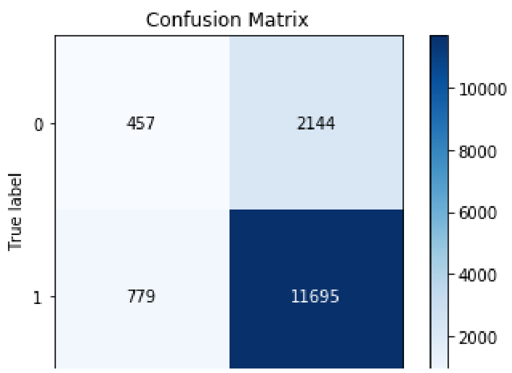
In [55]:

```
#confusion matirx
import scikitplot.metrics as skplt
# for train data prediction
skplt.plot_confusion_matrix(Y_train, Y_prob_train)
#for test data prediction
skplt.plot_confusion_matrix(Y_test, Y_prob_test )
```

Out[55]:

```
<matplotlib.axes._subplots.AxesSubplot at 0x48403ecc0>
```





------------------------- End of Brute - BoW ------------------------------------

4.1.2 BOW - Kd-Tree

In [26]:

```
Max_No_Of_Neighbours = 100;
K_Value_List_odd = list(range(1,Max_No_Of_Neighbours,2))
```

In [39]:

```
%%time
from sklearn.neighbors import KNeighborsClassifier

Y_data_pred_bow_kd_tr=[]
Y_data_pred_bow_kd_cv=[]

for K in tqdm(K_Value_List_odd):
    neigh_bow = KNeighborsClassifier(n_neighbors=K, algorithm='kd_tree')
    neigh_bow.fit(X_train_sub, Y_train_sub)
    Y_data_pred_bow_kd_tr.append(batch_predict(neigh_bow, X_train_sub))
    Y_data_pred_bow_kd_cv.append(batch_predict(neigh_bow, X_cv_bow))
    # roc_auc_score(y_true, y_score) the 2nd parameter should be probability estimates of the posi
tive class
    # not the predicted outputs
```

```
100%|██████████████████████████████████████████| 50/50 [36:02<00:00, 42.87s/it]
```

```
Wall time: 36min 2s
```

In [143]:

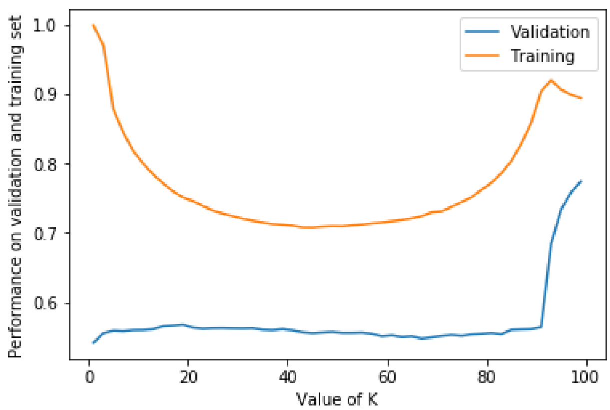
```
K_Value_bow_kd, auc_bow_kd_tr,auc_bow_kd_cv,roc_bow_kd_tr, roc_bow_kd_cv = Score_Provider(Y_train_s
ub,Y_data_pred_bow_kd_tr, y_cv,Y_data_pred_bow_kd_cv)
```

In [144]:

```
plt.plot(K_Value_List_odd, auc_bow_kd_cv, label="Validation")
plt.plot(K_Value_List_odd, auc_bow_kd_tr, label="Training")
plt.xlabel("Value of K");
plt.ylabel("Performance on validation and training set")
plt.legend()
```

Out[144]:

```
<matplotlib.legend.Legend at 0x8880a764e0>
```



In [145]:

```
%%time
from sklearn.neighbors import KNeighborsClassifier
from sklearn.metrics import auc

knn_to_fit_the_best = KNeighborsClassifier(n_neighbors=K_Value_bow_kd,algorithm='kd_tree')
knn_to_fit_the_best.fit(X_train, Y_train)

Y_prob_train = batch_predict(knn_to_fit_the_best, X_train)
Y_prob_test = batch_predict(knn_to_fit_the_best, X_test)

fpr, tpr, threshholds = roc_curve(Y_train,Y_prob_train)
fpr_test, tpr_test, threshholds_test = roc_curve(Y_test,Y_prob_test)

#computing AUC value on prediction over train data
roc_auc_train_bow_kd = auc(fpr, tpr)
print(roc_auc_train_bow_kd)
#computing AUC value on prediction over test data
roc_auc_test_bow_kd = auc(fpr_test, tpr_test)
print(roc_auc_test_bow_kd)
```

```
0.769132276328341
0.570084969354481
Wall time: 1min 36s
```

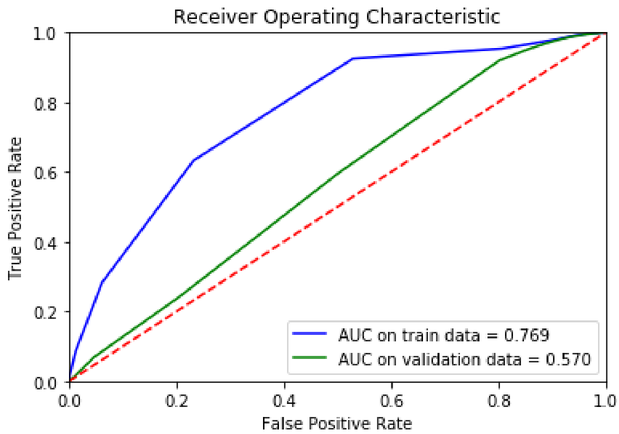
In [276]:

```
print(K_Value_bow_kd)
```

```
99
```

In [178]:

```
#Plotting ROC Curve
import matplotlib.pyplot as plt
plt.title('Receiver Operating Characteristic')
plt.plot(fpr, tpr, 'b', label = 'AUC on train data = %0.3f' % roc_auc_train_bow_kd)

plt.plot(fpr_test, tpr_test, 'g', label = 'AUC on validation data = %0.3f' % roc_auc_test_bow_kd)
plt.legend(loc = 'lower right')
plt.plot([0, 1], [0, 1],'r--')
plt.xlim([0, 1])
plt.ylim([0, 1])
plt.ylabel('True Positive Rate')
plt.xlabel('False Positive Rate')
plt.show()
```



In [175]:

```
data = X_train
clf=knn_to_fit_the_best

y_data_pred = []
tr_loop = data.shape[0] - data.shape[0]%1000
# consider you X_tr shape is 49041, then your cr_loop will be 49041 - 49041%1000 = 49000
# in this for loop we will iterate until the last 1000 multiplier
```

```python
for i in tqdm(range(0, tr_loop, 1000)):
    y_data_pred.extend(clf.predict(data[i:i+1000]))
    # we will be predicting for the last data points
y_data_pred.extend(clf.predict(data[tr_loop:]))
```

```
  0%|                                          | 0/35 [00:00<?, ?it/s]
  3%|█▎                                        | 1/35 [00:02<01:23,  2.44s/it]
  6%|██▍                                       | 2/35 [00:04<01:13,  2.24s/it]
  9%|███▋                                      | 3/35 [00:05<01:06,  2.08s/it]
 11%|████▉                                     | 4/35 [00:07<01:01,  1.98s/it]
 14%|██████▏                                   | 5/35 [00:09<00:57,  1.91s/it]
 17%|███████▎                                  | 6/35 [00:11<00:59,  2.04s/it]
 20%|████████▌                                 | 7/35 [00:13<00:54,  1.95s/it]
 23%|█████████▊                                | 8/35 [00:15<00:50,  1.89s/it]
 26%|███████████                               | 9/35 [00:16<00:48,  1.85s/it]
 29%|████████████▎                            | 10/35 [00:18<00:45,  1.80s/it]
 31%|█████████████▌                           | 11/35 [00:20<00:43,  1.81s/it]
 34%|██████████████▋                          | 12/35 [00:22<00:45,  1.96s/it]
 37%|███████████████▉                         | 13/35 [00:24<00:41,  1.88s/it]
 40%|█████████████████▏                       | 14/35 [00:26<00:38,  1.83s/it]
 43%|██████████████████▍                      | 15/35 [00:27<00:36,  1.80s/it]
 46%|███████████████████▋                     | 16/35 [00:29<00:33,  1.78s/it]
 49%|████████████████████▉                    | 17/35 [00:32<00:34,  1.94s/it]
 51%|██████████████████████                   | 18/35 [00:33<00:32,  1.89s/it]
 54%|███████████████████████▎                 | 19/35 [00:35<00:29,  1.85s/it]
 57%|████████████████████████▌                | 20/35 [00:37<00:27,  1.83s/it]
 60%|█████████████████████████▊               | 21/35 [00:39<00:25,  1.83s/it]
 63%|███████████████████████████              | 22/35 [00:41<00:24,  1.92s/it]
 66%|████████████████████████████▎            | 23/35 [00:43<00:25,  2.09s/it]
 69%|█████████████████████████████▍           | 24/35 [00:45<00:22,  2.07s/it]
 71%|██████████████████████████████▋          | 25/35 [00:47<00:20,  2.04s/it]
 74%|███████████████████████████████▉         | 26/35 [00:49<00:18,  2.02s/it]
 77%|█████████████████████████████████▏       | 27/35 [00:52<00:17,  2.20s/it]
 80%|██████████████████████████████████▍      | 28/35 [00:54<00:14,  2.11s/it]
 83%|███████████████████████████████████▋     | 29/35 [00:56<00:12,  2.06s/it]
 86%|████████████████████████████████████▊    | 30/35 [00:58<00:10,  2.03s/it]
 89%|██████████████████████████████████████   | 31/35 [01:00<00:07,  1.99s/it]
 91%|███████████████████████████████████████▎ | 32/35 [01:02<00:06,  2.20s/it]
 94%|████████████████████████████████████████▌| 33/35 [01:04<00:04,  2.07s/it]
 97%|█████████████████████████████████████████▊| 34/35 [01:06<00:01,  1.96s/it]
```

```
100%|████████████████████████████████████████████| 35/35 [01:07<00:00,  1.89s/it]
```

In [174]:

```python
test_data=X_test
clf=knn_to_fit_the_best

y_test_data_pred = []
test_loop = test_data.shape[0] - test_data.shape[0]%1000
# consider you X_tr shape is 49041, then your cr_loop will be 49041 - 49041%1000 = 49000
# in this for loop we will iterate until the last 1000 multiplier

for i in tqdm(range(0, test_loop, 1000)):
    y_test_data_pred.extend(clf.predict(test_data[i:i+1000]))
    # we will be predicting for the last data points
y_test_data_pred.extend(clf.predict(test_data[test_loop:]))
```

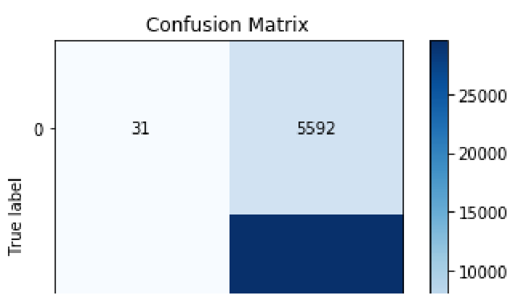
```
  0%|                                            | 0/15 [00:00<?, ?it/s]
  7%|███                                         | 1/15 [00:01<00:26,  1.90s/it]
 13%|██████                                      | 2/15 [00:04<00:26,  2.07s/it]
 20%|█████████                                   | 3/15 [00:06<00:24,  2.00s/it]
 27%|████████████                                | 4/15 [00:07<00:21,  1.92s/it]
 33%|███████████████                             | 5/15 [00:09<00:19,  1.92s/it]
 40%|██████████████████                          | 6/15 [00:11<00:17,  1.91s/it]
 47%|█████████████████████                       | 7/15 [00:14<00:16,  2.05s/it]
 53%|███████████████████████                     | 8/15 [00:15<00:13,  1.96s/it]
 60%|██████████████████████████                  | 9/15 [00:17<00:11,  1.90s/it]
 67%|█████████████████████████████               | 10/15 [00:19<00:09,  1.90s/it]
 73%|████████████████████████████████            | 11/15 [00:21<00:07,  1.85s/it]
 80%|███████████████████████████████████         | 12/15 [00:23<00:05,  1.87s/it]
 87%|██████████████████████████████████████      | 13/15 [00:25<00:04,  2.00s/it]
 93%|█████████████████████████████████████████   | 14/15 [00:27<00:01,  1.97s/it]
100%|████████████████████████████████████████████| 15/15 [00:29<00:00,  1.91s/it]
```

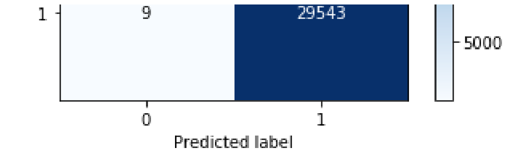
In [176]:

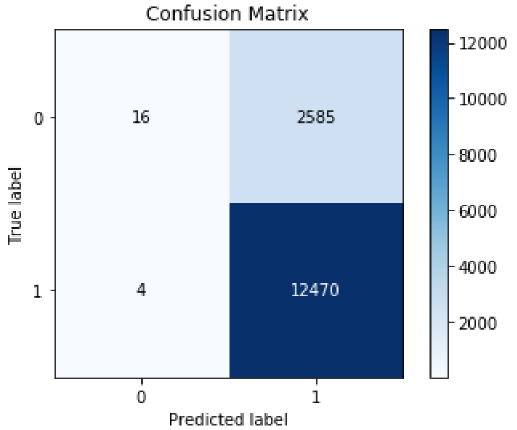
```python
#confusion matirx
import scikitplot.metrics as skplt
# for train data prediction
skplt.plot_confusion_matrix(Y_train, y_data_pred)
#for test data prediction
skplt.plot_confusion_matrix(Y_test, y_test_data_pred )
```

Out[176]:

```
<matplotlib.axes._subplots.AxesSubplot at 0x88a3532128>
```







In [177]:

```
from sklearn.metrics import roc_auc_score

roc_auc_score(Y_test,y_test_data_pred)
```

Out[177]:

```
0.5029154066062947
```

-------------------------------- End Of BoW-KdTree -------------------------------------

[4.2] Bi-Grams and n-Grams.

In [ ]:

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

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

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

[4.3] TF-IDF

Preparing datapoints into Train, test and validation set

In [202]:

```
#divide preprocessed review data to Train and Test dataset using train_test_split
from sklearn.model_selection import train_test_split
from sklearn import preprocessing

#Breaking into Train and test
X_train, X_test, Y_train, Y_test = train_test_split(preprocessed_reviews,final['Score'].values,test
_size=0.3,shuffle=False)
X_train_div, X_CV, Y_train_div, Y_CV = train_test_split(X_train,Y_train,test_size=0.2,shuffle=False
```

```
)
```

In [203]:

```
#Using uni- and bi-grams
tfidf = TfidfVectorizer(ngram_range=(1,2), max_features=500)
#Taking only 500 dimensions
X_train_div = tfidf.fit_transform(X_train_div)
X_train = tfidf.fit_transform(X_train)
#Normalize Data
X_train = preprocessing.normalize(X_train)
X_train_div = preprocessing.normalize(X_train_div)
X_test = tfidf.transform(X_test)
X_CV = tfidf.transform(X_CV)
#Normalize Data
X_test = preprocessing.normalize(X_test)
X_CV = preprocessing.normalize(X_CV)

X_train_div = X_train_div[:,:]
X_test = X_test[:,:]
X_CV = X_CV[:,:]

print("The total train dataset:",X_train.shape)
print("Train Data Size: ",X_train_div.shape)
print("Test Data Size: ",X_CV.shape)
```

```
The total train dataset: (35175, 500)
Train Data Size:  (28140, 500)
Test Data Size:  (7035, 500)
```

4.3.1 Tf-Idf - KdTree

In [181]:

```
Max_No_Of_Neighbours = 100;
K_Value_List_odd = list(range(1,Max_No_Of_Neighbours,2))
```

In [72]:

```
%%time
from sklearn.neighbors import KNeighborsClassifier

Y_data_pred_tf_kd_tr=[]
Y_data_pred_tf_kd_cv=[]

for K in tqdm(K_Value_List_odd):
    neigh_bow = KNeighborsClassifier(n_neighbors=K, algorithm='kd_tree')
    neigh_bow.fit(X_train_div, Y_train_div)
    Y_data_pred_tf_kd_tr.append(batch_predict(neigh_bow, X_train_div))
    Y_data_pred_tf_kd_cv.append(batch_predict(neigh_bow, X_CV))
    # roc_auc_score(y_true, y_score) the 2nd parameter should be probability estimates of the posi
tive class
    # not the predicted outputs
```

```
100%|██████████████████████████████████████████| 50/50 [39:38<00:00, 47.18s/it]
```

```
Wall time: 39min 38s
```

In [218]:

```
K_Value_tf_kd, auc_tf_kd_tr,auc_tf_kd_cv,roc_tf_kd_tr, roc_tf_kd_cv =
Score_Provider(Y_train_div,Y_data_pred_tf_kd_tr, Y_CV,Y_data_pred_tf_kd_cv)
```

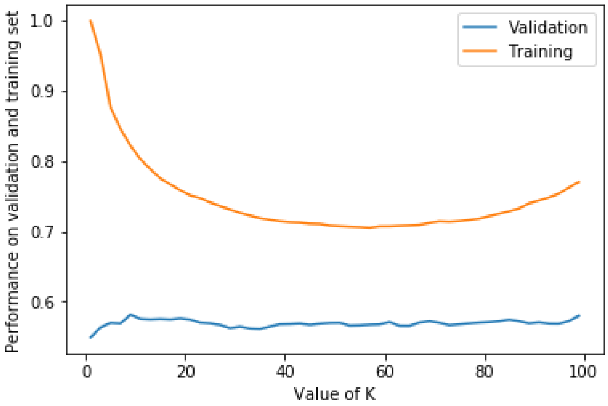
In [219]:

```
plt.plot(K_Value_List_odd, auc_tf_kd_cv, label="Validation")
plt.plot(K_Value_List_odd, auc_tf_kd_tr, label="Training")
plt.xlabel("Value of K");
plt.ylabel("Performance on validation and training set")
```

```
plt.legend()
```

Out[219]:

```
<matplotlib.legend.Legend at 0x88c1bedfd0>
```



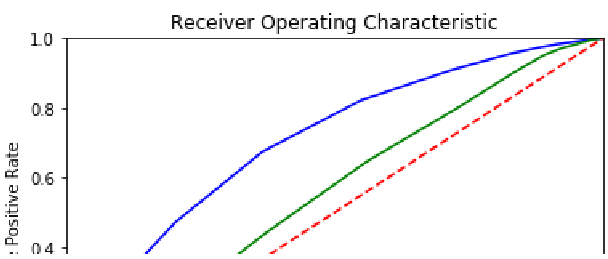
In [220]:

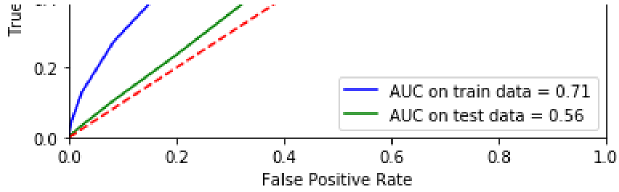
```
%%time
from sklearn.neighbors import KNeighborsClassifier
from sklearn.metrics import auc

knn_to_fit_the_best = KNeighborsClassifier(n_neighbors=K_Value_tf_kd,algorithm='kd_tree')
knn_to_fit_the_best.fit(X_train, Y_train)

Y_prob_train = batch_predict(knn_to_fit_the_best, X_train)
Y_prob_test = batch_predict(knn_to_fit_the_best, X_test)

fpr, tpr, threshholds = roc_curve(Y_train,Y_prob_train)
fpr_test, tpr_test, threshholds_test = roc_curve(Y_test,Y_prob_test)

#computing AUC value on prediction over train data
roc_auc_train_tf_kd = auc(fpr, tpr)
print(roc_auc_train_tf_kd)
#computing AUC value on prediction over test data
roc_auc_test_tf_kd = auc(fpr_test, tpr_test)
print(roc_auc_test_tf_kd)
```

```
0.7061900358291382
0.5561884752580639
Wall time: 1min 29s
```

In [221]:

```
#Plotting ROC Curve
import matplotlib.pyplot as plt
plt.title('Receiver Operating Characteristic')
plt.plot(fpr, tpr, 'b', label = 'AUC on train data = %0.2f' % roc_auc_train_tf_kd)

plt.plot(fpr_test, tpr_test, 'g', label = 'AUC on test data = %0.2f' % roc_auc_test_tf_kd)
plt.legend(loc = 'lower right')
plt.plot([0, 1], [0, 1],'r--')
plt.xlim([0, 1])
plt.ylim([0, 1])
plt.ylabel('True Positive Rate')
plt.xlabel('False Positive Rate')
plt.show()
```





In [223]:

```
data = X_train
clf=knn_to_fit_the_best

y_data_pred = []
tr_loop = data.shape[0] - data.shape[0]%1000
# consider you X_tr shape is 49041, then your cr_loop will be 49041 - 49041%1000 = 49000
# in this for loop we will iterate until the last 1000 multiplier

for i in range(0, tr_loop, 1000):
    y_data_pred.extend(clf.predict(data[i:i+1000]))
    # we will be predicting for the last data points
y_data_pred.extend(clf.predict(data[tr_loop:]))
```

In [224]:

```
test_data=X_test
clf=knn_to_fit_the_best

y_test_data_pred = []
test_loop = test_data.shape[0] - test_data.shape[0]%1000
# consider you X_tr shape is 49041, then your cr_loop will be 49041 - 49041%1000 = 49000
# in this for loop we will iterate until the last 1000 multiplier

for i in range(0, test_loop, 1000):
    y_test_data_pred.extend(clf.predict(test_data[i:i+1000]))
    # we will be predicting for the last data points
y_test_data_pred.extend(clf.predict(test_data[test_loop:]))
print("============================================")
```

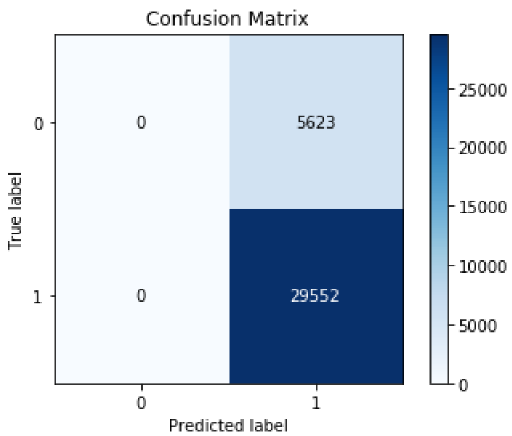
```
============================================
```

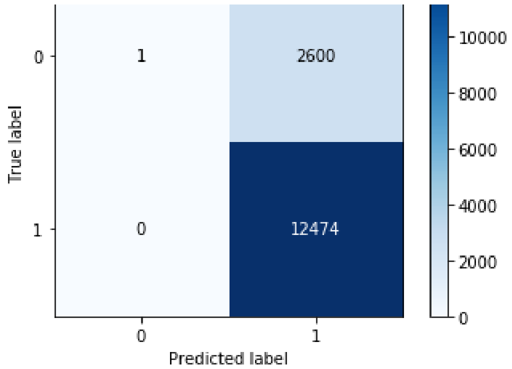
In [225]:

```
#confusion matirx
import scikitplot.metrics as skplt
# for train data prediction
skplt.plot_confusion_matrix(Y_train, y_data_pred)
#for test data prediction
skplt.plot_confusion_matrix(Y_test, y_test_data_pred )
```

Out[225]:

```
<matplotlib.axes._subplots.AxesSubplot at 0x88c4ea54a8>
```





In [226]:

```
roc_auc_score(Y_test,y_test_data_pred)
```

Out[226]:

```
0.5001922337562477
```

4.3.2 TfIdf - Brute

In [89]:

```
%%time
from sklearn.neighbors import KNeighborsClassifier

Y_data_pred_tf_br_tr=[]
Y_data_pred_tf_br_cv=[]

for K in tqdm(K_Value_List_odd):
    neigh_bow = KNeighborsClassifier(n_neighbors=K, algorithm='brute')
    neigh_bow.fit(X_train_div, Y_train_div)
    Y_data_pred_tf_br_tr.append(batch_predict(neigh_bow, X_train_div))
    Y_data_pred_tf_br_cv.append(batch_predict(neigh_bow, X_CV))
    # roc_auc_score(y_true, y_score) the 2nd parameter should be probability estimates of the posi
tive class
    # not the predicted outputs
```

```
100%|██████████████████████████████████████████| 50/50 [38:10<00:00, 46.08s/it]
```

```
Wall time: 38min 10s
```

In [227]:

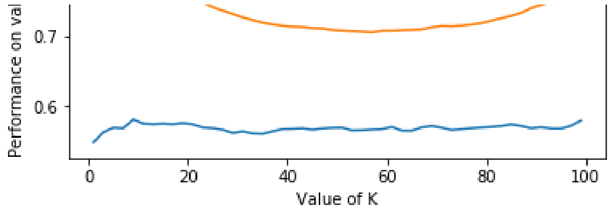
```
K_Value_tf_br, auc_tf_br_tr,auc_tf_br_cv,roc_tf_br_tr, roc_tf_br_cv = Score_Provider(Y_train_div, Y
_data_pred_tf_br_tr, Y_CV, Y_data_pred_tf_br_cv)
```

In [228]:

```
plt.plot(K_Value_List_odd, auc_tf_br_cv, label="Validation")
plt.plot(K_Value_List_odd, auc_tf_br_tr, label="Training")
plt.xlabel("Value of K");
plt.ylabel("Performance on validation and training set")
plt.legend()
```

Out[228]:

```
<matplotlib.legend.Legend at 0x88c382d6a0>
```



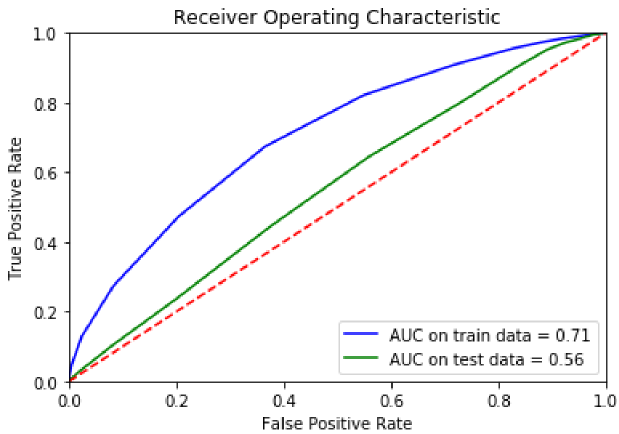
In [229]:

```
%%time
from sklearn.neighbors import KNeighborsClassifier
from sklearn.metrics import auc

knn_to_fit_the_best = KNeighborsClassifier(n_neighbors=K_Value_tf_br,algorithm='kd_tree')
knn_to_fit_the_best.fit(X_train, Y_train)

Y_prob_train = batch_predict(knn_to_fit_the_best, X_train)
Y_prob_test = batch_predict(knn_to_fit_the_best, X_test)

fpr, tpr, threshholds = roc_curve(Y_train,Y_prob_train)
fpr_test, tpr_test, threshholds_test = roc_curve(Y_test,Y_prob_test)

#computing AUC value on prediction over train data
roc_auc_train_tf_br = auc(fpr, tpr)
print(roc_auc_train_tf_br)
#computing AUC value on prediction over test data
roc_auc_test_tf_br = auc(fpr_test, tpr_test)
print(roc_auc_test_tf_br)
```

```
0.7061900358291382
0.5561884752580639
Wall time: 1min 29s
```

In [230]:

```
#Plotting ROC Curve
import matplotlib.pyplot as plt
plt.title('Receiver Operating Characteristic')
plt.plot(fpr, tpr, 'b', label = 'AUC on train data = %0.2f' % roc_auc_train_tf_br)

plt.plot(fpr_test, tpr_test, 'g', label = 'AUC on test data = %0.2f' % roc_auc_test_tf_br)
plt.legend(loc = 'lower right')
plt.plot([0, 1], [0, 1],'r--')
plt.xlim([0, 1])
plt.ylim([0, 1])
plt.ylabel('True Positive Rate')
plt.xlabel('False Positive Rate')
plt.show()
```



In [231]:

```
data = X_train
clf=knn_to_fit_the_best

y_data_pred = []
tr_loop = data.shape[0] - data.shape[0]%1000
```

```
tr_loop = data.shape[0] - data.shape[0]%1000
# consider you X_tr shape is 49041, then your cr_loop will be 49041 - 49041%1000 = 49000
# in this for loop we will iterate until the last 1000 multiplier

for i in range(0, tr_loop, 1000):
    y_data_pred.extend(clf.predict(data[i:i+1000]))
    # we will be predicting for the last data points
y_data_pred.extend(clf.predict(data[tr_loop:]))
```

In [232]:

```
test_data=X_test
clf=knn_to_fit_the_best

y_test_data_pred = []
test_loop = test_data.shape[0] - test_data.shape[0]%1000
# consider you X_tr shape is 49041, then your cr_loop will be 49041 - 49041%1000 = 49000
# in this for loop we will iterate until the last 1000 multiplier

for i in range(0, test_loop, 1000):
    y_test_data_pred.extend(clf.predict(test_data[i:i+1000]))
    # we will be predicting for the last data points
y_test_data_pred.extend(clf.predict(test_data[test_loop:]))
print("============================================")
```

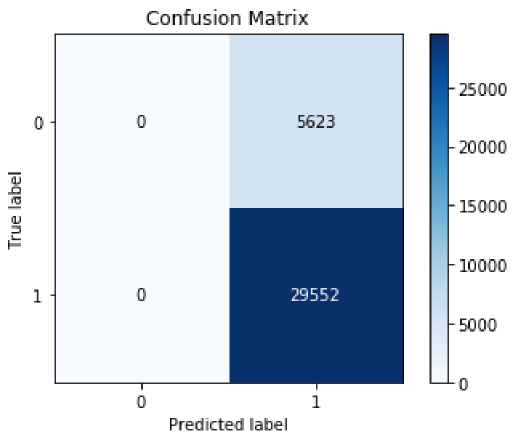
```
============================================
```

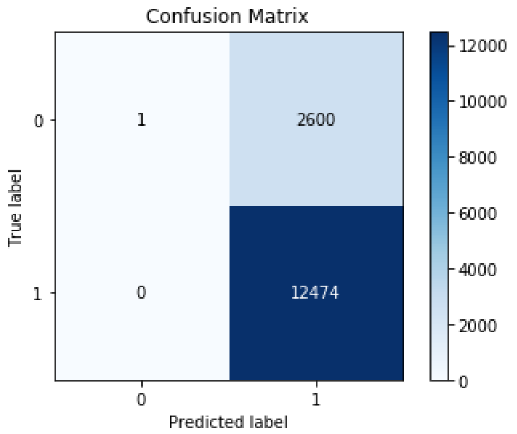
In [233]:

```
#confusion matirx
import scikitplot.metrics as skplt
# for train data prediction
skplt.plot_confusion_matrix(Y_train, y_data_pred)
#for test data prediction
skplt.plot_confusion_matrix(Y_test, y_test_data_pred)
```

Out[233]:

```
<matplotlib.axes._subplots.AxesSubplot at 0x88c3376c88>
```





In [234]:

```
roc_auc_score(Y_test,y_test_data_pred)
```

Out[234]:

```
0.5001922337562477
```

[4.4] Word2Vec

In [90]:

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

```
100%|██████████████████████████████████| 50250/50250 [00:00<00:00, 93509.38it/s]
```

In [91]:

```
# Using Google News Word2Vectors

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


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

is_your_ram_gt_16g=False
want_to_use_google_w2v = False
want_to_train_w2v = True

if want_to_train_w2v:
    # min_count = 5 considers only words that occured atleast 5 times
    w2v_model=Word2Vec(list_of_sentance,min_count=5,size=50, workers=4)
    print(w2v_model.wv.most_similar('great'))
    print('='*500)
    print(w2v_model.wv.most_similar('worst'))

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

```
[('awesome', 0.8535417318344116), ('terrific', 0.807737410068512), ('fantastic', 0.8068745732307434), ('wonderful', 0.8040031790733337), ('amazing', 0.7949562668800354), ('perfect', 0.7941107749938965), ('good', 0.789454996585846), ('excellent', 0.7841684818267822), ('fabulous', 0.7118127346038818), ('nice', 0.6893609166145325)]
====================================================================================================
====================================================================================================
====================================================================================================
====================================================================================================
====================================================================================================
[('greatest', 0.7174200415611267), ('nastiest', 0.716022253036499), ('best', 0.702441930770874), ('tastiest', 0.6704314947128296), ('awful', 0.663917064666748), ('disgusting', 0.6562853455543518), ('experienced', 0.6376103162765503), ('hardly', 0.6375899314880371), ('worse', 0.6312829256057739), ('horrible', 0.6296736598014832)]
```

In [92]:

```
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  13354
sample words  ['dogs', 'loves', 'chicken', 'product', 'china', 'wont', 'buying', 'anymore',
'hard', 'find', 'products', 'made', 'usa', 'one', 'isnt', 'bad', 'good', 'take', 'chances',
'till', 'know', 'going', 'imports', 'love', 'saw', 'pet', 'store', 'tag', 'attached', 'regarding',
'satisfied', 'safe', 'available', 'victor', 'traps', 'unreal', 'course', 'total', 'fly', 'pretty',
'stinky', 'right', 'nearby', 'used', 'bait', 'seasons', 'ca', 'not', 'beat', 'great']
```

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

[4.4.1.1] Avg W2v - Brute

In [93]:

```
# average Word2Vec
# compute average word2vec for each review.
sent_vectors = []; # the avg-w2v for each sentence/review is stored in this list
for sent in tqdm(list_of_sentance): # for each review/sentence
    sent_vec = np.zeros(50) # as word vectors are of zero length 50, you might need to change this
to 300 if you use google's w2v
    cnt_words =0; # num of words with a valid vector in the sentence/review
    for word in sent: # for each word in a review/sentence
        if word in w2v_words:
            vec = w2v_model.wv[word]
            sent_vec += vec
            cnt_words += 1
    if cnt_words != 0:
        sent_vec /= cnt_words
    sent_vectors.append(sent_vec)
print(len(sent_vectors))
print(len(sent_vectors[0]))
```

```
100%|██████████████████████████████████████| 50250/50250 [02:34<00:00, 325.70it/s]
```

```
50250
50
```

In [236]:

```
from sklearn import preprocessing
from sklearn.model_selection import train_test_split

sent_vectors = preprocessing.normalize(sent_vectors)

#Not shuffling the data as we want it on time basis
X_train, X_test, Y_train,Y_test = train_test_split(sent_vectors,final['Score'].values,test_size=0.3
,shuffle=False)
X_train_div, X_CV, Y_train_div, Y_CV = train_test_split(X_train,Y_train,test_size=0.2,shuffle=False
)
print("The training and test dataset shape is:", X_train_div.shape,X_CV.shape)
```

```
The training and test dataset shape is: (28140, 50) (7035, 50)
```

In [95]:

```
%%time
from sklearn.neighbors import KNeighborsClassifier

Y_data_pred_Aw2v_br_tr=[]
Y_data_pred_Aw2v_br_cv=[]

for K in tqdm(K_Value_List_odd):
```

```
    neigh_bow = KNeighborsClassifier(n_neighbors=K, algorithm='brute')
    neigh_bow.fit(X_train_div, Y_train_div)
    Y_data_pred_Aw2v_br_tr.append(batch_predict(neigh_bow, X_train_div))
    Y_data_pred_Aw2v_br_cv.append(batch_predict(neigh_bow, X_CV))
    # roc_auc_score(y_true, y_score) the 2nd parameter should be probability estimates of the posi
tive class
    # not the predicted outputs
```

```
100%|██████████████████████████████████████████| 50/50 [21:05<00:00, 26.04s/it]
```

```
Wall time: 21min 5s
```

In [237]:

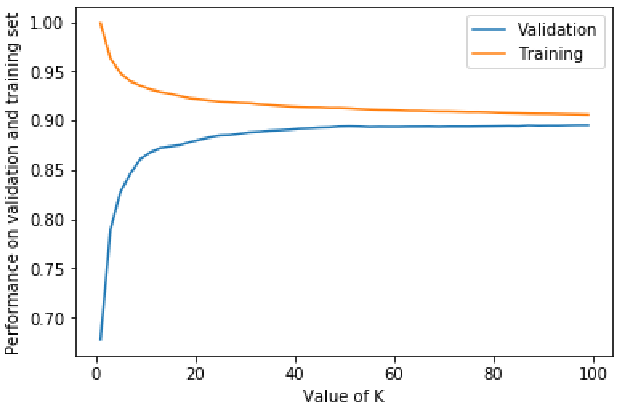
```
K_Value_Aw2v_br, auc_Aw2v_br_tr,auc_Aw2v_br_cv,roc_Aw2v_br_tr, roc_Aw2v_br_cv = Score_Provider(Y_tr
ain_div, Y_data_pred_Aw2v_br_tr, Y_CV, Y_data_pred_Aw2v_br_cv)
```

In [238]:

```
plt.plot(K_Value_List_odd, auc_Aw2v_br_cv, label="Validation")
plt.plot(K_Value_List_odd, auc_Aw2v_br_tr, label="Training")
plt.xlabel("Value of K");
plt.ylabel("Performance on validation and training set")
plt.legend()
```

Out[238]:

```
<matplotlib.legend.Legend at 0x88c3d44b00>
```



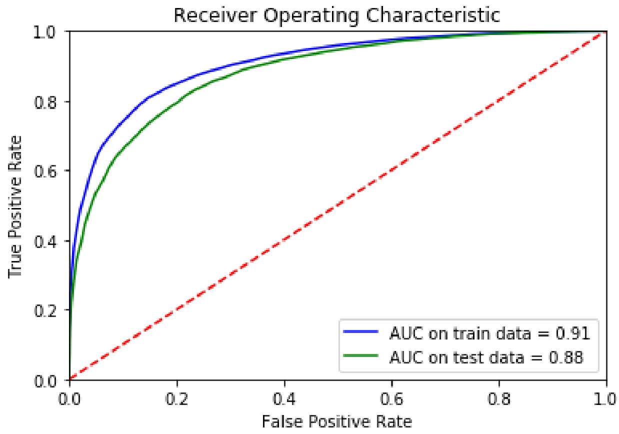
In [239]:

```
%%time
from sklearn.neighbors import KNeighborsClassifier
from sklearn.metrics import auc

knn_to_fit_the_best = KNeighborsClassifier(n_neighbors=K_Value_Aw2v_br,algorithm='kd_tree')
knn_to_fit_the_best.fit(X_train, Y_train)

Y_prob_train = batch_predict(knn_to_fit_the_best, X_train)
Y_prob_test = batch_predict(knn_to_fit_the_best, X_test)

fpr, tpr, threshholds = roc_curve(Y_train,Y_prob_train)
fpr_test, tpr_test, threshholds_test = roc_curve(Y_test,Y_prob_test)

#computing AUC value on prediction over train data
roc_auc_train_Aw2v_br = auc(fpr, tpr)
print(roc_auc_train_Aw2v_br)
#computing AUC value on prediction over test data
roc_auc_test_Aw2v_br = auc(fpr_test, tpr_test)
print(roc_auc_test_Aw2v_br)
print(K_Value_Aw2v_br)
```

```
0.9067235907544242
0.8811440599214534
99
Wall time: 6min 5s
```

In [240]:

```
#Plotting ROC Curve
import matplotlib.pyplot as plt
plt.title('Receiver Operating Characteristic')
plt.plot(fpr, tpr, 'b', label = 'AUC on train data = %0.2f' % roc_auc_train_Aw2v_br)

plt.plot(fpr_test, tpr_test, 'g', label = 'AUC on test data = %0.2f' % roc_auc_test_Aw2v_br)
plt.legend(loc = 'lower right')
plt.plot([0, 1], [0, 1],'r--')
plt.xlim([0, 1])
plt.ylim([0, 1])
plt.ylabel('True Positive Rate')
plt.xlabel('False Positive Rate')
plt.show()
```



In [241]:

```
data = X_train
clf=knn_to_fit_the_best

y_data_pred = []
tr_loop = data.shape[0] - data.shape[0]%1000
# consider you X_tr shape is 49041, then your cr_loop will be 49041 - 49041%1000 = 49000
# in this for loop we will iterate until the last 1000 multiplier

for i in range(0, tr_loop, 1000):
    y_data_pred.extend(clf.predict(data[i:i+1000]))
    # we will be predicting for the last data points
y_data_pred.extend(clf.predict(data[tr_loop:]))
```

In [242]:

```
test_data=X_test
clf=knn_to_fit_the_best

y_test_data_pred = []
test_loop = test_data.shape[0] - test_data.shape[0]%1000
# consider you X_tr shape is 49041, then your cr_loop will be 49041 - 49041%1000 = 49000
# in this for loop we will iterate until the last 1000 multiplier

for i in range(0, test_loop, 1000):
    y_test_data_pred.extend(clf.predict(test_data[i:i+1000]))
    # we will be predicting for the last data points
y_test_data_pred.extend(clf.predict(test_data[test_loop:]))
print("============================================")
```

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

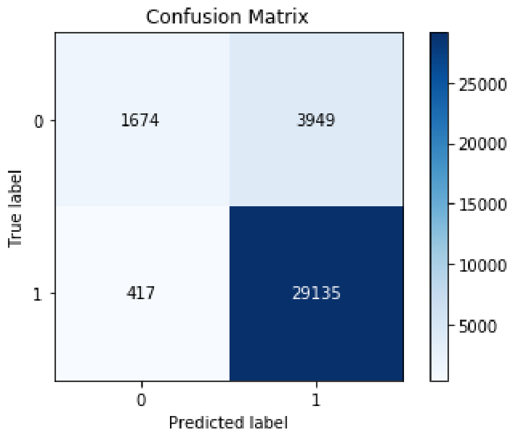
In [243]:

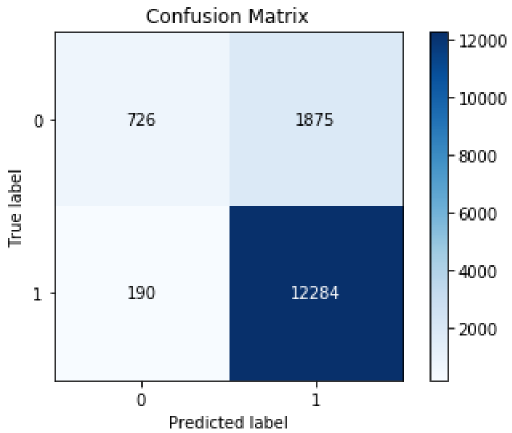
```
#confusion matirx
import scikitplot.metrics as skplt
# for train data prediction
skplt.plot_confusion_matrix(Y_train, y_data_pred)
```

```
#for test data prediction
skplt.plot_confusion_matrix(Y_test, y_test_data_pred )
```

Out[243]:

```
<matplotlib.axes._subplots.AxesSubplot at 0x88c170f4a8>
```





In [244]:

```
roc_auc_score(Y_test, y_test_data_pred)
```

Out[244]:

```
0.6319458660865812
```

---------------------------- End of Avg W2V - Brute ----------------------------------

Avg W2V - KdTree

In [96]:

```
%%time
from sklearn.neighbors import KNeighborsClassifier

Y_data_pred_Aw2v_kd_tr=[]
Y_data_pred_Aw2v_kd_cv=[]

for K in tqdm(K_Value_List_odd):
    neigh_bow = KNeighborsClassifier(n_neighbors=K, algorithm='kd_tree')
    neigh_bow.fit(X_train_div, Y_train_div)
    Y_data_pred_Aw2v_kd_tr.append(batch_predict(neigh_bow, X_train_div))
    Y_data_pred_Aw2v_kd_cv.append(batch_predict(neigh_bow, X_CV))
    # roc_auc_score(y_true, y_score) the 2nd parameter should be probability estimates of the posi
tive class
    # not the predicted outputs
```

```
100%|████████████████████████████████████████| 50/50 [2:11:57<00:00, 167.96s/it]
```

```
Wall time: 2h 11min 57s
```

In [245]:

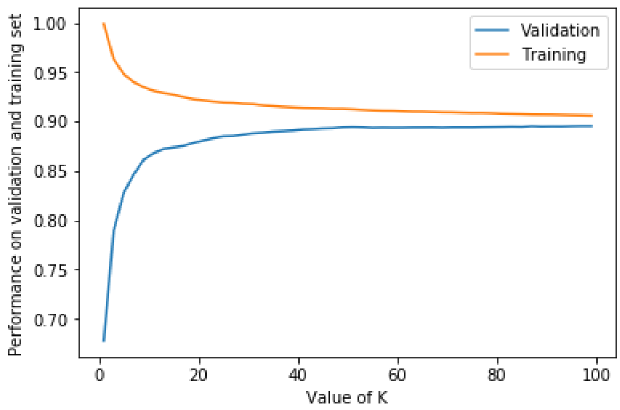
```
K_Value_Aw2v_kd, auc_Aw2v_kd_tr,auc_Aw2v_kd_cv,roc_Aw2v_kd_tr, roc_Aw2v_kd_cv = Score_Provider(Y_tr
ain_div, Y_data_pred_Aw2v_kd_tr, Y_CV, Y_data_pred_Aw2v_kd_cv)
```

In [246]:

```
plt.plot(K_Value_List_odd, auc_Aw2v_kd_cv, label="Validation")
plt.plot(K_Value_List_odd, auc_Aw2v_kd_tr, label="Training")
plt.xlabel("Value of K");
plt.ylabel("Performance on validation and training set")
plt.legend()
```

Out[246]:

```
<matplotlib.legend.Legend at 0x88c177afd0>
```



In [247]:

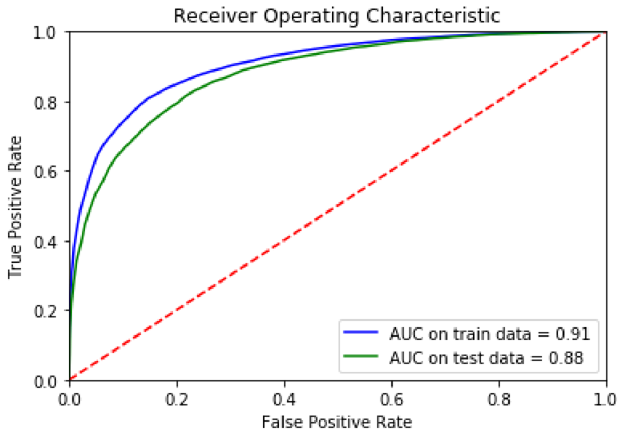
```
%%time
from sklearn.neighbors import KNeighborsClassifier
from sklearn.metrics import auc

knn_to_fit_the_best = KNeighborsClassifier(n_neighbors=K_Value_Aw2v_kd,algorithm='kd_tree')
knn_to_fit_the_best.fit(X_train, Y_train)

Y_prob_train = batch_predict(knn_to_fit_the_best, X_train)
Y_prob_test = batch_predict(knn_to_fit_the_best, X_test)

fpr, tpr, threshholds = roc_curve(Y_train,Y_prob_train)
fpr_test, tpr_test, threshholds_test = roc_curve(Y_test,Y_prob_test)

#computing AUC value on prediction over train data
roc_auc_train_Aw2v_kd = auc(fpr, tpr)
print(roc_auc_train_Aw2v_kd)
#computing AUC value on prediction over test data
roc_auc_test_Aw2v_kd = auc(fpr_test, tpr_test)
print(roc_auc_test_Aw2v_kd)
print(K_Value_Aw2v_kd)
```

```
0.9067235907544242
0.8811440599214534
99
Wall time: 6min 18s
```

In [248]:

```
#Plotting ROC Curve
import matplotlib.pyplot as plt
plt.title('Receiver Operating Characteristic')
plt.plot(fpr, tpr, 'b', label = 'AUC on train data = %0.2f' % roc_auc_train_Aw2v_kd)
```

```
plt.plot(fpr_test, tpr_test, 'g', label = 'AUC on test data = %0.2f' % roc_auc_test_Aw2v_kd)
plt.legend(loc = 'lower right')
plt.plot([0, 1], [0, 1],'r--')
plt.xlim([0, 1])
plt.ylim([0, 1])
plt.ylabel('True Positive Rate')
plt.xlabel('False Positive Rate')
plt.show()
```



In [249]:

```
data = X_train
clf=knn_to_fit_the_best

y_data_pred = []
tr_loop = data.shape[0] - data.shape[0]%1000
# consider you X_tr shape is 49041, then your cr_loop will be 49041 - 49041%1000 = 49000
# in this for loop we will iterate until the last 1000 multiplier

for i in range(0, tr_loop, 1000):
    y_data_pred.extend(clf.predict(data[i:i+1000]))
    # we will be predicting for the last data points
y_data_pred.extend(clf.predict(data[tr_loop:]))
```

In [250]:

```
test_data=X_test
clf=knn_to_fit_the_best

y_test_data_pred = []
test_loop = test_data.shape[0] - test_data.shape[0]%1000
# consider you X_tr shape is 49041, then your cr_loop will be 49041 - 49041%1000 = 49000
# in this for loop we will iterate until the last 1000 multiplier

for i in range(0, test_loop, 1000):
    y_test_data_pred.extend(clf.predict(test_data[i:i+1000]))
    # we will be predicting for the last data points
y_test_data_pred.extend(clf.predict(test_data[test_loop:]))
print("===========================================")
```

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

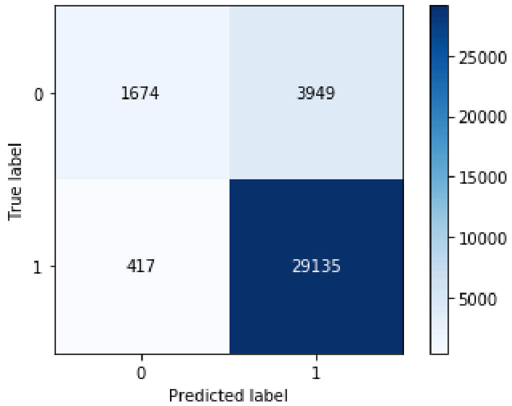
In [251]:

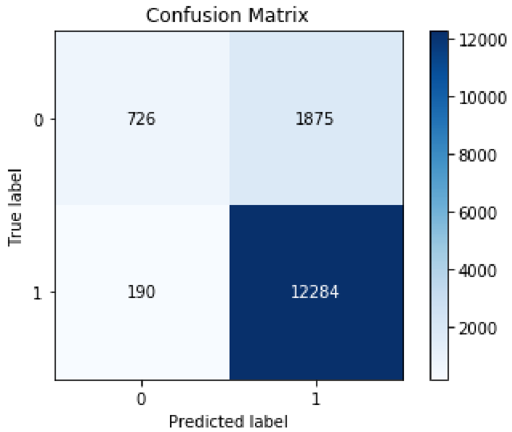
```
#confusion matirx
import scikitplot.metrics as skplt
# for train data prediction
skplt.plot_confusion_matrix(Y_train, y_data_pred)
#for test data prediction
skplt.plot_confusion_matrix(Y_test, y_test_data_pred )
```

Out[251]:

```
<matplotlib.axes._subplots.AxesSubplot at 0x88c3eb2a58>
```

Confusion Matrix





TFIDF W2V

In [97]:

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

In [84]:

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

tfidf_sent_vectors = []; # the tfidf-w2v for each sentence/review is stored in this list
row=0;
for sent in tqdm(list_of_sentance): # for each review/sentence
    sent_vec = np.zeros(50) # as word vectors are of zero length
    weight_sum =0; # num of words with a valid vector in the sentence/review
    for word in sent: # for each word in a review/sentence
        if word in w2v_words and word in tfidf_feat:
            vec = w2v_model.wv[word]
#               tf_idf = tf_idf_matrix[row, tfidf_feat.index(word)]
            # to reduce the computation we are
            # dictionary[word] = idf value of word in whole courpus
            # sent.count(word) = tf valeus of word in this review
            tf_idf = dictionary[word]*(sent.count(word)/len(sent))
            sent_vec += (vec * tf_idf)
            weight_sum += tf_idf
    if weight_sum != 0:
        sent_vec /= weight_sum
    tfidf_sent_vectors.append(sent_vec)
    row += 1
```

```
100%|██████████████████████████████████| 50250/50250 [30:59<00:00, 27.02it/s]
```

TfIdf - W2v- Brute

In [252]:

```
from sklearn import preprocessing
from sklearn.model_selection import train_test_split

tfidf_sent_vectors = preprocessing.normalize(tfidf_sent_vectors)

#Not shuffling the data as we want it on time basis
X_train, X_test, Y_train,Y_test = train_test_split(tfidf_sent_vectors,final['Score'].values,test_si
ze=0.3,shuffle=False)
X_train_div, X_CV, Y_train_div, Y_CV = train_test_split(X_train,Y_train,test_size=0.2,shuffle=False
)
print("The training and test dataset shape is:", X_train_div.shape,X_CV.shape)
```

```
The training and test dataset shape is: (28140, 50) (7035, 50)
```

In [99]:

```
%%time
from sklearn.neighbors import KNeighborsClassifier

Y_data_pred_Tw2v_br_tr=[]
Y_data_pred_Tw2v_br_cv=[]

for K in tqdm(K_Value_List_odd):
    neigh_bow = KNeighborsClassifier(n_neighbors=K, algorithm='brute')
    neigh_bow.fit(X_train_div, Y_train_div)
    Y_data_pred_Tw2v_br_tr.append(batch_predict(neigh_bow, X_train_div))
    Y_data_pred_Tw2v_br_cv.append(batch_predict(neigh_bow, X_CV))
    # roc_auc_score(y_true, y_score) the 2nd parameter should be probability estimates of the posi
tive class
    # not the predicted outputs
```

```
100%|██████████████████████████████████████████| 50/50 [21:59<00:00, 26.75s/it]
```

```
Wall time: 21min 59s
```

In [255]:

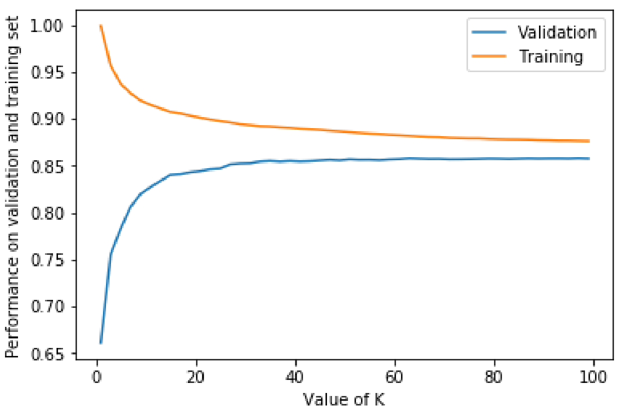
```
K_Value_Tw2v_br, auc_Tw2v_br_tr, auc_Tw2v_br_cv,roc_Tw2v_br_tr, roc_Tw2v_br_cv = Score_Provider(Y_t
rain_div, Y_data_pred_Tw2v_br_tr, Y_CV, Y_data_pred_Tw2v_br_cv)
```

In [256]:

```
plt.plot(K_Value_List_odd, auc_Tw2v_br_cv, label="Validation")
plt.plot(K_Value_List_odd, auc_Tw2v_br_tr, label="Training")
plt.xlabel("Value of K");
plt.ylabel("Performance on validation and training set")
plt.legend()
```

Out[256]:

```
<matplotlib.legend.Legend at 0x8884f60f98>
```



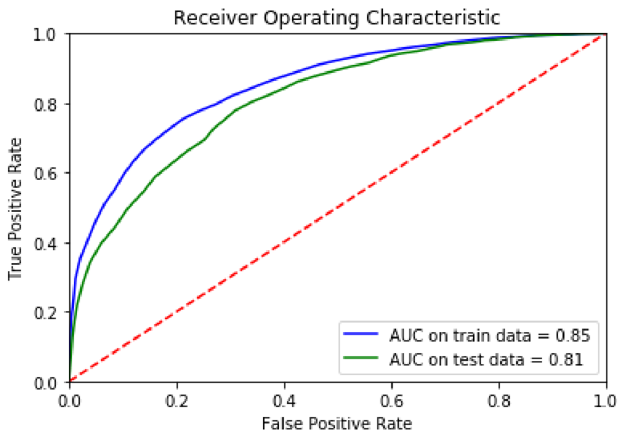
In [257]:

```
%%time
from sklearn.neighbors import KNeighborsClassifier
from sklearn.metrics import auc

knn_to_fit_the_best = KNeighborsClassifier(n_neighbors=K_Value_Tw2v_br,algorithm='kd_tree')
knn_to_fit_the_best.fit(X_train, Y_train)

Y_prob_train = batch_predict(knn_to_fit_the_best, X_train)
Y_prob_test = batch_predict(knn_to_fit_the_best, X_test)

fpr, tpr, threshholds = roc_curve(Y_train,Y_prob_train)
fpr_test, tpr_test, threshholds_test = roc_curve(Y_test,Y_prob_test)

#computing AUC value on prediction over train data
roc_auc_train_Tw2v_br = auc(fpr, tpr)
print(roc_auc_train_Tw2v_br)
#computing AUC value on prediction over test data
roc_auc_test_Tw2v_br = auc(fpr_test, tpr_test)
print(roc_auc_test_Tw2v_br)
print(K_Value_Tw2v_br)
```

```
0.8779949468407512
0.8489076117231954
97
Wall time: 5min 24s
```

In [176]:

```
#Plotting ROC Curve
import matplotlib.pyplot as plt
plt.title('Receiver Operating Characteristic')
plt.plot(fpr, tpr, 'b', label = 'AUC on train data = %0.2f' % roc_auc_train_Tw2v_br)

plt.plot(fpr_test, tpr_test, 'g', label = 'AUC on test data = %0.2f' % roc_auc_test_Tw2v_br)
plt.legend(loc = 'lower right')
plt.plot([0, 1], [0, 1],'r--')
plt.xlim([0, 1])
plt.ylim([0, 1])
plt.ylabel('True Positive Rate')
plt.xlabel('False Positive Rate')
plt.show()
```



In [260]:

```
data = X_train
clf=knn_to_fit_the_best

y_data_pred = []
tr_loop = data.shape[0] - data.shape[0]%1000
# consider you X_tr shape is 49041, then your cr_loop will be 49041 - 49041%1000 = 49000
# in this for loop we will iterate until the last 1000 multiplier

for i in range(0, tr_loop, 1000):
    y_data_pred.extend(clf.predict(data[i:i+1000]))
```

```
    # we will be predicting for the last data points
y_data_pred.extend(clf.predict(data[tr_loop:]))
```

In [261]:

```
test_data=X_test
clf=knn_to_fit_the_best

y_test_data_pred = []
test_loop = test_data.shape[0] - test_data.shape[0]%1000
# consider you X_tr shape is 49041, then your cr_loop will be 49041 - 49041%1000 = 49000
# in this for loop we will iterate until the last 1000 multiplier

for i in range(0, test_loop, 1000):
    y_test_data_pred.extend(clf.predict(test_data[i:i+1000]))
    # we will be predicting for the last data points
y_test_data_pred.extend(clf.predict(test_data[test_loop:]))
print("===========================================")
```

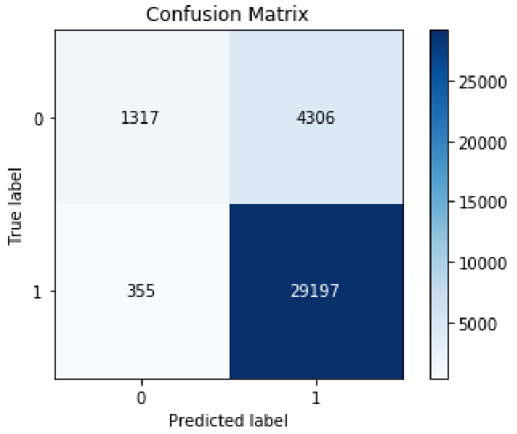
```
===========================================
```

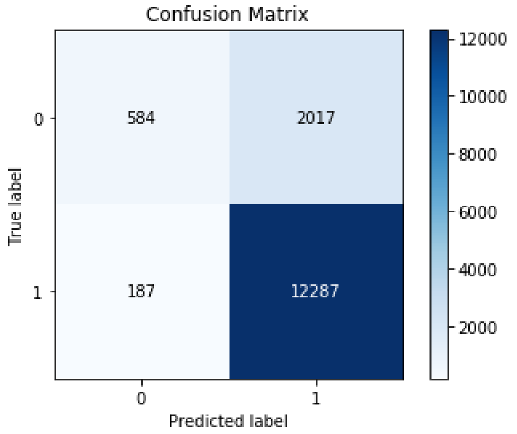
In [262]:

```
#confusion matirx
import scikitplot.metrics as skplt
# for train data prediction
skplt.plot_confusion_matrix(Y_train, y_data_pred)
#for test data prediction
skplt.plot_confusion_matrix(Y_test, y_test_data_pred )
```

Out[262]:

```
<matplotlib.axes._subplots.AxesSubplot at 0x88c4f45470>
```





Avg W2V TfIdf - KdTree

In [100]:

```
%%time
from sklearn.neighbors import KNeighborsClassifier

Y_data_pred_Tw2v_kd_tr=[]
Y_data_pred_Tw2v_kd_cv=[]

for K in tqdm(K_Value_List_odd):
    neigh_bow = KNeighborsClassifier(n_neighbors=K, algorithm='kd_tree')
    neigh_bow.fit(X_train_div, Y_train_div)
    Y_data_pred_Tw2v_kd_tr.append(batch_predict(neigh_bow, X_train_div))
    Y_data_pred_Tw2v_kd_cv.append(batch_predict(neigh_bow, X_CV))
    # roc_auc_score(y_true, y_score) the 2nd parameter should be probability estimates of the posi
tive class
    # not the predicted outputs
```

```
100%|██████████████████████████████████████████| 50/50 [2:36:52<00:00, 163.47s/it]
```

```
Wall time: 2h 36min 52s
```

In [263]:

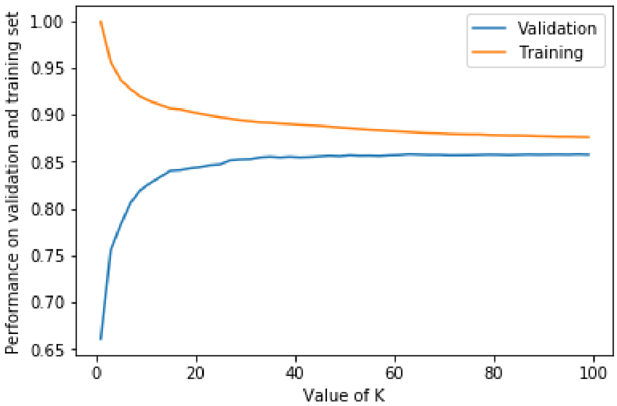
```
K_Value_Tw2v_kd, auc_Tw2v_br_kd, auc_Tw2v_kd_cv,roc_Tw2v_kd_tr, roc_Tw2v_kd_cv = Score_Provider(Y_t
rain_div, Y_data_pred_Tw2v_kd_tr, Y_CV, Y_data_pred_Tw2v_kd_cv)
```

In [264]:

```
plt.plot(K_Value_List_odd, auc_Tw2v_kd_cv, label="Validation")
plt.plot(K_Value_List_odd, auc_Tw2v_br_kd, label="Training")
plt.xlabel("Value of K");
plt.ylabel("Performance on validation and training set")
plt.legend()
```

Out[264]:

```
<matplotlib.legend.Legend at 0x88c5228828>
```



In [267]:

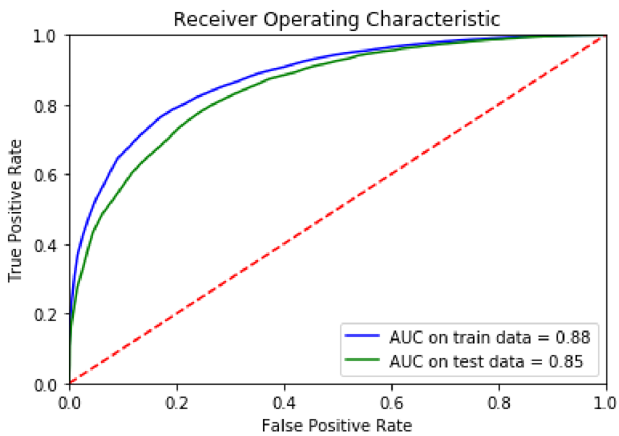
```
%%time
from sklearn.neighbors import KNeighborsClassifier
from sklearn.metrics import auc

knn_to_fit_the_best = KNeighborsClassifier(n_neighbors=K_Value_Tw2v_kd,algorithm='kd_tree')
knn_to_fit_the_best.fit(X_train, Y_train)

Y_prob_train = batch_predict(knn_to_fit_the_best, X_train)
Y_prob_test = batch_predict(knn_to_fit_the_best, X_test)

fpr, tpr, threshholds = roc_curve(Y_train,Y_prob_train)
fpr_test, tpr_test, threshholds_test = roc_curve(Y_test,Y_prob_test)

#computing AUC value on prediction over train data
roc_auc_train_Tw2v_kd = auc(fpr, tpr)
print(roc_auc_train_Tw2v_kd)
#computing AUC value on prediction over test data
roc_auc_test_Tw2v_kd = auc(fpr_test, tpr_test)
print(roc_auc_test_Tw2v_kd)
```

```
print(K_Value_Tw2v_kd)
```

```
0.8779949468407512
0.8489076117231954
97
Wall time: 5min 26s
```

In [268]:

```
#Plotting ROC Curve
import matplotlib.pyplot as plt
plt.title('Receiver Operating Characteristic')
plt.plot(fpr, tpr, 'b', label = 'AUC on train data = %0.2f' % roc_auc_train_Tw2v_kd)

plt.plot(fpr_test, tpr_test, 'g', label = 'AUC on test data = %0.2f' % roc_auc_test_Tw2v_kd)
plt.legend(loc = 'lower right')
plt.plot([0, 1], [0, 1],'r--')
plt.xlim([0, 1])
plt.ylim([0, 1])
plt.ylabel('True Positive Rate')
plt.xlabel('False Positive Rate')
plt.show()
```



In [269]:

```
data = X_train
clf=knn_to_fit_the_best

y_data_pred = []
tr_loop = data.shape[0] - data.shape[0]%1000
# consider you X_tr shape is 49041, then your cr_loop will be 49041 - 49041%1000 = 49000
# in this for loop we will iterate until the last 1000 multiplier

for i in range(0, tr_loop, 1000):
    y_data_pred.extend(clf.predict(data[i:i+1000]))
    # we will be predicting for the last data points
y_data_pred.extend(clf.predict(data[tr_loop:]))
```

In [270]:

```
test_data=X_test
clf=knn_to_fit_the_best

y_test_data_pred = []
test_loop = test_data.shape[0] - test_data.shape[0]%1000
# consider you X_tr shape is 49041, then your cr_loop will be 49041 - 49041%1000 = 49000
# in this for loop we will iterate until the last 1000 multiplier

for i in range(0, test_loop, 1000):
    y_test_data_pred.extend(clf.predict(test_data[i:i+1000]))
    # we will be predicting for the last data points
y_test_data_pred.extend(clf.predict(test_data[test_loop:]))
print("============================================")
```

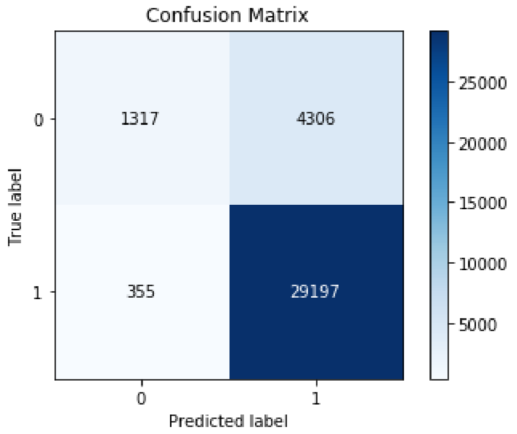
```
============================================
```

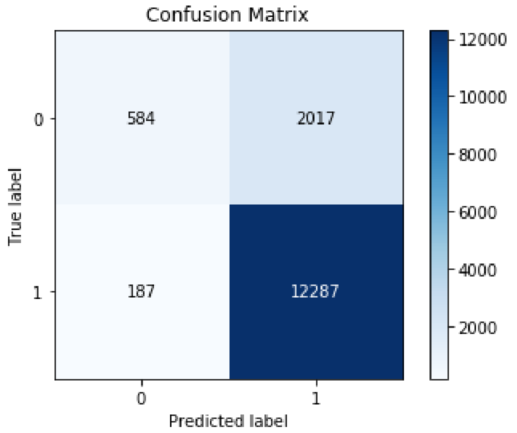
In [271]:

```
#confusion matirx
import scikitplot.metrics as skplt
# for train data prediction
skplt.plot_confusion_matrix(Y_train, y_data_pred)
#for test data prediction
skplt.plot_confusion_matrix(Y_test, y_test_data_pred )
```

Out[271]:

```
<matplotlib.axes._subplots.AxesSubplot at 0x88a1a134a8>
```





In [279]:

```
from prettytable import PrettyTable
X = PrettyTable()

X.field_names = ["vectorizer","Algorithm","Optimal-K Value","AUC on train","AUC on Test"]
X.add_row(["BoW","Brute",K_Value_bow_kd, roc_auc_train_bow_br, roc_auc_test_bow_br])
X.add_row(["BoW","Kd-Tree",K_Value_bow_kd,roc_auc_train_bow_kd, roc_auc_test_bow_kd])
X.add_row(["TfIdf","Brute",K_Value_tf_br,roc_auc_test_tf_br, roc_auc_test_tf_kd])
X.add_row(["TfIdf","Kd-Tree",K_Value_tf_kd,roc_auc_train_tf_kd, roc_auc_test_tf_kd])
X.add_row(["Avg W2V","Brute",K_Value_Aw2v_br,roc_auc_train_Aw2v_br, roc_auc_test_Aw2v_br])
X.add_row(["Avg W2V","Kd-Tree",K_Value_Aw2v_kd,roc_auc_train_Aw2v_kd, roc_auc_test_Aw2v_kd])
X.add_row(["Avg TfIdf W2V","Brute",K_Value_Tw2v_br,roc_auc_train_Tw2v_br, roc_auc_test_Tw2v_br])
X.add_row(["Avg TfIdf W2V","Kd-Tree",K_Value_Tw2v_kd,roc_auc_train_Tw2v_kd,roc_auc_test_Tw2v_kd])
print(X)
```

```
+---------------+-----------+-----------------+--------------------+--------------------+
|   vectorizer  | Algorithm | Optimal-K Value |    AUC on train    |    AUC on Test     |
+---------------+-----------+-----------------+--------------------+--------------------+
|      BoW      |   Brute   |        99       | 0.9991997154543838 | 0.5566258787135373 |
|      BoW      |  Kd-Tree  |        99       | 0.769132276328341  | 0.570084969354481  |
|     TfIdf     |   Brute   |        61       | 0.5561884752580639 | 0.5561884752580639 |
|     TfIdf     |  Kd-Tree  |        61       | 0.7061900358291382 | 0.5561884752580639 |
|    Avg W2V    |   Brute   |        99       | 0.9067235907544242 | 0.8811440599214534 |
|    Avg W2V    |  Kd-Tree  |        99       | 0.9067235907544242 | 0.8811440599214534 |
| Avg TfIdf W2V |   Brute   |        97       | 0.8779949468407512 | 0.8489076117231954 |
| Avg TfIdf W2V |  Kd-Tree  |        97       | 0.8779949468407512 | 0.8489076117231954 |
```

```
+----------------+-----------+------------------+---------------------+---------------------+
```

- In BoW though KNN had good AUC score on train data. It was overfitting and predicted more of as class 1
- Kd-Tree had significantly played role in AUC score of TfIdf where it has 0.2 more than Brute algorithm performance on train dataset. Anyway both had same performance on validation set.
- Average W2V really had shown great performance on both train and test data set.
- Average-TfIdf-W2V had also shown good performance on both, but has lesser score than former.