Amazon Fine Food Reviews Analysis

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

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

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

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

Attribute Information:

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

Objective:

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

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

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

[1]. Reading Data

[1.1] Loading the data

The dataset is available in two forms

1. .csv file
2. SQLite Database

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

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

In [1]:

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


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

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

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

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

from tqdm import tqdm
import os
```

In [2]:

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

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

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

filtered_data = pd.read_sql_query(""" SELECT * FROM Reviews WHERE Score != 3 LIMIT 100000""", 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 (100000, 10)
```

Out[2]:

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

	Id	ProductId	UserId	ProfileName	HelpfulnessNumerator	HelpfulnessDenominator	Score	Time	Summary
2	3	B000LQOCH0	ABXLMWJIXXAIN	Corres "Natalia Corres"	1	1	1	1219017600	"Delight" says it all

In [3]:

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

In [4]:

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

```
(80668, 7)
```

Out[4]:

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

In [5]:

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

Out[5]:

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

In [6]:

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

Out[6]:

```
393063
```

[2] Exploratory Data Analysis

[2.1] Data Cleaning: Deduplication

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

In [7]:

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

Out[7]:

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

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

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

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

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

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

In [8]:

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

In [9]:

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

Out[9]:

```
(87775, 10)
```

In [10]:

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

```
(final['Id'].size*1.0)/(filtered_data['Id'].size*1.0)*100
```

Out[10]:

```
87.775
```

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

In [11]:

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

display.head()
```

Out[11]:

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

In [12]:

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

In [13]:

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

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

```
(87773, 10)
```

Out[13]:

```
1    73592
0    14181
Name: Score, dtype: int64
```

[3] Preprocessing

[3.1]. Preprocessing Review Text

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

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

1. Begin by removing the html tags

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

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

In [14]:

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

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

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

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

```
My dogs loves this chicken but its a product from China, so we wont be buying it anymore.  Its ver
y hard to find any chicken products made in the USA but they are out there, but this one isnt.  It
s too bad too because its a good product but I wont take any chances till they know what is going
on with the china imports.
==================================================
The Candy Blocks were a nice visual for the Lego Birthday party but the candy has little taste to
it.  Very little of the 2 lbs that I bought were eaten and I threw the rest away.  I would not buy
the candy again.
==================================================
was way to hot for my blood, took a bite and did a jig  lol
==================================================
My dog LOVES these treats. They tend to have a very strong fish oil smell. So if you are afraid of
the fishy smell, don't get it. But I think my dog likes it because of the smell. These treats are
really small in size. They are great for training. You can give your dog several of these without
worrying about him over eating. Amazon's price was much more reasonable than any other retailer. Y
ou can buy a 1 pound bag on Amazon for almost the same price as a 6 ounce bag at other retailers.
It's definitely worth it to buy a big bag if your dog eats them a lot.
==================================================
```

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 ver
y hard to find any chicken products made in the USA but they are out there, but this one isnt.  It
s too bad too because its a good product but I wont take any chances till they know what is going
on with the china imports.
```

In [16]:

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

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

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

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

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

```
My dogs loves this chicken but its a product from China, so we wont be buying it anymore.  Its ver
y hard to find any chicken products made in the USA but they are out there, but this one isnt.  It
s too bad too because its a good product but I wont take any chances till they know what is going
on with the china imports.
==================================================
The Candy Blocks were a nice visual for the Lego Birthday party but the candy has little taste to
it.  Very little of the 2 lbs that I bought were eaten and I threw the rest away.  I would not buy
the candy again.
==================================================
was way to hot for my blood, took a bite and did a jig  lol
==================================================
My dog LOVES these treats. They tend to have a very strong fish oil smell. So if you are afraid of
the fishy smell, don't get it. But I think my dog likes it because of the smell. These treats are
really small in size. They are great for training. You can give your dog several of these without
worrying about him over eating. Amazon's price was much more reasonable than any other retailer. Y
ou can buy a 1 pound bag on Amazon for almost the same price as a 6 ounce bag at other retailers.
It's definitely worth it to buy a big bag if your dog eats them a lot.
```

In [17]:

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

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

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

In [18]:

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

```
was way to hot for my blood, took a bite and did a jig  lol
==================================================
```

In [19]:

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

```
My dogs loves this chicken but its a product from China, so we wont be buying it anymore.  Its ver
y hard to find any chicken products made in the USA but they are out there, but this one isnt.  It
s too bad too because its a good product but I wont take any chances till they know what is going
on with the china imports.
```

In [20]:

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

```
was way to hot for my blood took a bite and did a jig lol
```

In [21]:

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

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

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

```
100%|██████████| 87773/87773 [00:31<00:00, 2752.33it/s]
```

In [23]:

```
preprocessed_reviews[1500]
```

Out[23]:

```
'way hot blood took bite jig lol'
```

[3.2] Preprocessing Review Summary

Similartly you can do preprocessing for review summary also.

[4] Featurization

[4.1] BAG OF WORDS

In [24]:

```
#BoW
count_vect = CountVectorizer(max_features = 5000,min_df = 10) #in scikit-learn
count_vect.fit(preprocessed_reviews)
print("some feature names ", count_vect.get_feature_names()[:10])
print('='*50)

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

```
some feature names  ['ability', 'able', 'absolute', 'absolutely', 'absorb', 'absorbed', 'acai', 'a
ccept', 'acceptable', 'accepted']
==================================================
the type of count vectorizer  <class 'scipy.sparse.csr.csr_matrix'>
the shape of out text BOW vectorizer  (87773, 5000)
the number of unique words  5000
```

[4.2] Bi-Grams and n-Grams.

In [25]:

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

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

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

```
the type of count vectorizer  <class 'scipy.sparse.csr.csr_matrix'>
the shape of out text BOW vectorizer  (87773, 5000)
the number of unique words including both unigrams and bigrams  5000
```

[4.3] TF-IDF

In [26]:

```
tf_idf_vect = TfidfVectorizer(ngram_range=(1,2), min_df=10,max_features=5000)
tf_idf_vect.fit(preprocessed_reviews)
print("some sample features(unique words in the corpus)",tf_idf_vect.get_feature_names()[0:10])
print('='*50)

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

```
some sample features(unique words in the corpus) ['ability', 'able', 'able buy', 'able find',
'able get', 'absolute', 'absolute favorite', 'absolutely', 'absolutely best', 'absolutely
delicious']
==================================================
the type of count vectorizer  <class 'scipy.sparse.csr.csr_matrix'>
the shape of out text TFIDF vectorizer  (87773, 5000)
the number of unique words including both unigrams and bigrams  5000
```

[4.4] Word2Vec

In [27]:

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

In [28]:

```
# 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

want_to_train_w2v = True

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

```
[('fantastic', 0.835064172744751), ('excellent', 0.8148547410964966), ('terrific',
0.8135321140289307), ('awesome', 0.8106702566146851), ('good', 0.8045200109481812), ('wonderful',
0.7591359615325928), ('perfect', 0.7456346750259399), ('nice', 0.731300950050354), ('amazing', 0.7
099001407623291), ('fabulous', 0.7019394636154175)]
==================================================
[('greatest', 0.8424437046051025), ('tastiest', 0.7769253253936768), ('best', 0.7433720827102661),
('nastiest', 0.6567928791046143), ('smoothest', 0.6418704390525818), ('disgusting',
0.6383175253868103), ('surpass', 0.6206839084625244), ('closest', 0.6079548001289368),
('freshest', 0.6070331335067749), ('horrible', 0.5871541500091553)]
```

In [29]:

```
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  17386
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', 'infestation', 'literally', 'everywhere', 'flying', 'around', 'kitchen',
'bought', 'hoping', 'least', 'got', 'rid', 'weeks', 'fly', 'stuck', 'squishing', 'buggers', 'succe
'bought', 'hoping', 'least', 'get', 'fid', 'weeks', 'fly', 'stuck', 'squishing', 'buggers', 'succe
ss', 'rate']
```

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

[4.4.1.1] Avg W2v

In [30]:

```
# 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%|██████████| 87773/87773 [03:34<00:00, 408.61it/s]
```

```
87773
50
```

[4.4.1.2] TFIDF weighted W2v

In [31]:

```
# S = ["abc def pqr", "def def def abc", "pqr pqr def"]
model = TfidfVectorizer(min_df=10, max_features=5000)
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 [32]:

```
# 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%|██████████| 87773/87773 [07:59<00:00, 183.19it/s]
```

[5] Assignment 7: SVM

1. **Apply SVM on these feature sets**

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

2. **Procedure**

    - You need to work with 2 versions of SVM
        - Linear kernel
        - RBF kernel
    - When you are working with linear kernel, use SGDClassifier' with hinge loss because it is computationally less expensive.
    - When you are working with 'SGDClassifier' with hinge loss and trying to find the AUC score, you would have to use CalibratedClassifierCV
    - Similarly, like kdtree of knn, when you are working with RBF kernel it's better to reduce the number of dimensions. You can put min_df = 10, max_features = 500 and consider a sample size of 40k points.

3. **Hyper paramter tuning (find best alpha in range [10^-4 to 10^4], and the best penalty among 'l1', 'l2')**

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

4. **Feature importance**

    - When you are working on the linear kernel with BOW or TFIDF please print the top 10 best features for each of the positive and negative classes.

5. **Feature engineering**

    - To increase the performance of your model, you can also experiment with with feature engineering like :
        - Taking length of reviews as another feature.
        - Considering some features from review summary as well.

6. **Representation of results**

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

7. **Conclusion**

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

Note: Data Leakage

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

Applying SVM

Applying SVM

[5.1] Linear SVM

[5.1.1] Applying Linear SVM on BOW, SET 1

In [34]:

```
# ============================== loading libraries ===========================================
import pdb
import numpy as np
import pandas as pd
import seaborn as sns
import matplotlib.pyplot as plt
import sklearn
from sklearn.calibration import CalibratedClassifierCV
from sklearn.model_selection import TimeSeriesSplit
from sklearn.model_selection import train_test_split
from sklearn.model_selection import cross_val_score
from sklearn.metrics import accuracy_score
from sklearn.metrics import roc_auc_score
from sklearn.metrics import confusion_matrix
from collections import Counter
import scikitplot.metrics as skplt
from sklearn import linear_model
from sklearn.svm import SVC
from sklearn.model_selection import GridSearchCV
# ===============================================================================================
```

In [100]:

```
#Spliting entire data to train,test and cross validation
X=np.array(preprocessed_reviews)
y = np.array(final['Score'])

#https://scikit-learn.org/stable/modules/generated/sklearn.model_selection.TimeSeriesSplit.html
tscv = TimeSeriesSplit(n_splits=4)
for train_index, test_index in tscv.split(X):
    X_1, X_test = X[train_index], X[test_index]
    y_1, y_test = y[train_index], y[test_index]

tscv = TimeSeriesSplit(n_splits=3)
for train_index, test_index in tscv.split(X_1):
    X_tr, X_cv = X_1[train_index], X_1[test_index]
    y_tr, y_cv = y_1[train_index], y_1[test_index]

#converting Reviews to Bag of words after splitting to avoid data leakage problem
count_vect = CountVectorizer(max_features = 5000,min_df = 10)
final_X_tr=count_vect.fit_transform(X_tr)
final_X_test=count_vect.transform(X_test)
final_X_cv=count_vect.transform(X_cv)
```

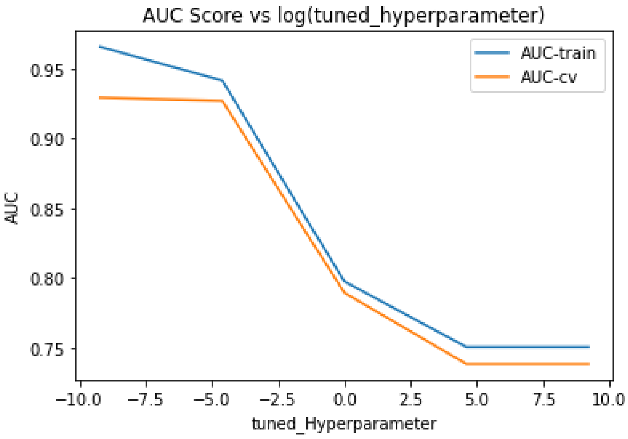
In [101]:

```
#Calculating for finding Best alpha
#predic_proba reference:
#https://stackoverflow.com/questions/37089177/probability-prediction-method-of-
kneighborsclassifier-returns-only-0-and-1
#https://discuss.analyticsvidhya.com/t/what-is-the-difference-between-predict-and-predict-proba/67
376/3
roc_tr=[]
roc_cv=[]
max_auc_score=0
best_alpha=0
tuned_parameters =[10**-4, 10**-2, 10**0, 10**2, 10**4]
for i in tuned_parameters:
    clf=linear_model.SGDClassifier(alpha=i,loss='hinge',penalty='l2',class_weight='balanced',random
_state=0)
    clf.fit(final_X_tr,y_tr)
    clf_sgd = CalibratedClassifierCV(base_estimator=clf,method='sigmoid')
    clf_sgd.fit(final_X_tr,y_tr)
    # predict the response on the traininig
    pred_tr = clf_sgd.predict_proba(final_X_tr)
    pred_tr=(pred_tr)[:,1]
    roc_tr.append(roc_auc_score(y_tr,pred_tr))

    # predict the response on the crossvalidation
    pred_cv = clf_sgd.predict_proba(final_X_cv)
    pred_cv=(pred_cv)[:,1]
    roc_cv.append(roc_auc_score(y_cv,pred_cv))

    #finding best c using loop
    if roc_auc_score(y_cv,pred_cv)>max_auc_score:
        best_alpha=i
        max_auc_score=roc_auc_score(y_cv,pred_cv)

print(best_alpha)
print(max_auc_score)
alpha_bow_linearsvm=best_alpha
auc_bow_linearsvm=max_auc_score
```

```
0.0001
0.929299078259954
```

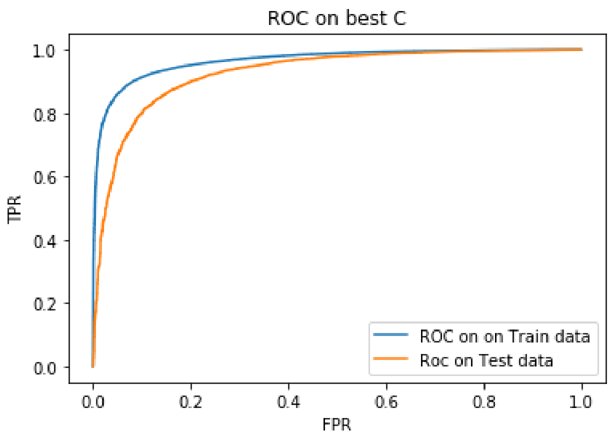
In [102]:

```
# plotting curve between between AUC of cv and train with log of tuned parameter
logalpha=np.log(tuned_parameters)
plt.plot(logalpha,roc_tr,label="AUC-train")
plt.plot(logalpha,roc_cv ,label="AUC-cv")
plt.legend()
plt.xlabel('tuned_Hyperparameter')
plt.ylabel('AUC')
plt.title('AUC Score vs log(tuned_hyperparameter)')
plt.show()
```



In [103]:

```
#1)Training the model using best C
clf=linear_model.SGDClassifier(alpha=alpha_bow_linearsvm,loss='hinge',penalty='l2',class_weight='balanced',random_state=0)
clf.fit(final_X_tr,y_tr)
clf_sgd = CalibratedClassifierCV(base_estimator=clf,method='sigmoid')
clf_sgd.fit(final_X_tr,y_tr)
#predicting probability on Test data
pred_test = clf_sgd.predict_proba(final_X_test)
pred_test=(pred_test)[:,1]
#predicting probablity of Training data
pred_tr = clf_sgd.predict_proba(final_X_tr)
pred_tr=(pred_tr)[:,1]


#2)Plotting Roc Curve
#Reference for finding fpr an tpr :
#https://www.programcreek.com/python/example/81207/sklearn.metrics.roc_curve
fpr_tr, tpr_tr, threshold_train = metrics.roc_curve(y_tr, pred_tr)
fpr_test, tpr_test, threshold_test = metrics.roc_curve(y_test, pred_test)
plt.plot(fpr_tr,tpr_tr, label="ROC on on Train data")
plt.plot(fpr_tr,tpr_tr ,label="ROC on on Train data")
plt.plot(fpr_test,tpr_test ,label="Roc on Test data")
plt.legend()
plt.title('ROC on best C')
plt.xlabel('FPR')
plt.ylabel('TPR')
plt.show()
```

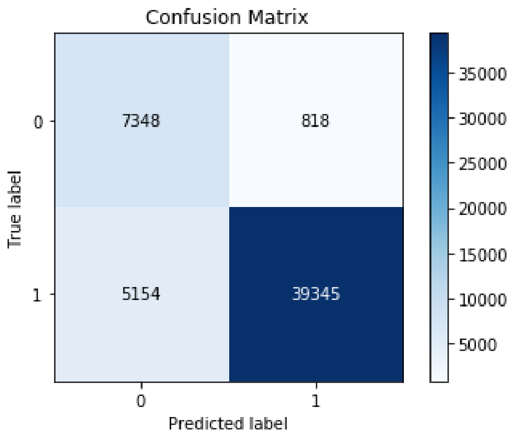


In [104]:

```
#plotting the confusion matrix on train data
#Reference:
#https://scikit-learn.org/stable/modules/generated/sklearn.metrics.confusion_matrix.html

prediction=clf.predict(final_X_tr)
skplt.plot_confusion_matrix(y_tr ,prediction)
```

Out[104]:

```
<matplotlib.axes._subplots.AxesSubplot at 0x7f7684647dd8>
```

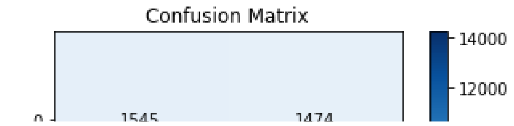


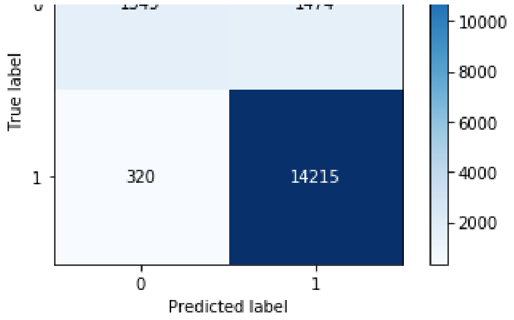
In [105]:

```
#plotting the confusion matrix on test data
#Reference:
#https://scikit-learn.org/stable/modules/generated/sklearn.metrics.confusion_matrix.html

prediction=clf_sgd.predict(final_X_test)
skplt.plot_confusion_matrix(y_test ,prediction)
```

Out[105]:

```
<matplotlib.axes._subplots.AxesSubplot at 0x7f768bac3c88>
```





Print the best feature for each of the positive and negative classes

In [106]:

```
#Reference for top features is from statistics of machine learning by pratap dangeti:
#https://books.google.co.in/books?id=C-dDDwAAQBAJ&pg=PA216&lpg=PA216&dq=coefs_with_fns%5B:-(n+%2B+
1):-
1&source=bl&ots=j18t1prZXo&sig=ACfU3U2yz8v4v3SOiSrT7fBpJauJKe80DQ&hl=en&sa=X&ved=2ahUKEwjzwK69x6Dj
XMBHYHqADIQ6AEwAXoECAgQAQ#v=onepage&q=coefs_with_fns%5B%3A-(n%20%2B%201)%3A-1&f=false
n=10
feature_names = count_vect.get_feature_names()
coefs_with_fns = sorted(zip(clf.coef_[0], feature_names))
top_n_coefs = zip(coefs_with_fns[:n], coefs_with_fns[:-(n + 1):-1])
print("\tNegative-feature\t\t\t\tPositive-feature")
print("--------------------------------------------------------------------------------------------------
-------")
for (coef_1, fn_1), (coef_2, fn_2) in top_n_coefs:
        print("\t%.4f\t%-15s\t\t\t\t%.4f\t%-15s" % (coef_1, fn_1, coef_2, fn_2))
```

```
 Negative-feature     Positive-feature
----------------------------------------------------------------------------------------------------
 -4.3400 worst              3.0470 pleasantly
 -4.0801 disappointing      3.0437 hooked
 -3.2219 disappointment     3.0261 perfect
 -3.0317 tasteless          3.0080 beat
 -2.9841 terrible           2.8794 excellent
 -2.9175 shame              2.8695 complaint
 -2.8247 hopes              2.8421 delicious
 -2.7073 died               2.7812 yummy
 -2.6852 horrible           2.7737 amazing
 -2.6424 distilled          2.7124 worried
```

[5.1.2] Applying Linear SVM on TFIDF, SET 2

In [107]:

```
#Spliting entire data to train,test and cross validation
X=np.array(preprocessed_reviews)
y = np.array(final['Score'])

##https://scikit-learn.org/stable/modules/generated/sklearn.model_selection.TimeSeriesSplit.html
tscv = TimeSeriesSplit(n_splits=4)
for train_index, test_index in tscv.split(X):
    X_1, X_test = X[train_index], X[test_index]
    y_1, y_test = y[train_index], y[test_index]

tscv = TimeSeriesSplit(n_splits=3)
for train_index, test_index in tscv.split(X_1):
    X_tr, X_cv = X_1[train_index], X_1[test_index]
    y_tr, y_cv = y_1[train_index], y_1[test_index]

#converting Reviews to Bag of words after splitting to avoid data leakage problem
tf_idf_vect = TfidfVectorizer(ngram_range=(1,2),min_df=10,max_features=2000)
final_X_tr=tf_idf_vect.fit_transform(X_tr)
final_X_test=tf_idf_vect.transform(X_test)
final_X_cv=tf_idf_vect.transform(X_cv)
```

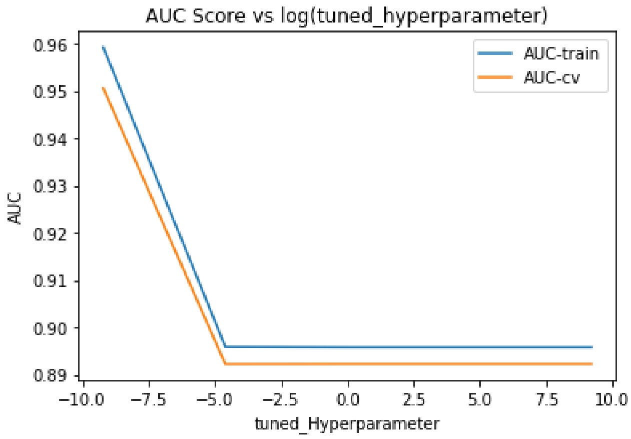
In [108]:

```
#Calculating for finding Best alpha
#predic_proba reference:
#https://stackoverflow.com/questions/37089177/probability-prediction-method-of-
kneighborsclassifier-returns-only-0-and-1
#https://discuss.analyticsvidhya.com/t/what-is-the-difference-between-predict-and-predict-proba/67
376/3
roc_tr=[]
roc_cv=[]
max_auc_score=0
best_alpha=0
tuned_parameters =[10**-4, 10**-2, 10**0, 10**2, 10**4]
for i in tuned_parameters:
    clf=linear_model.SGDClassifier(alpha=i,loss='hinge',penalty='l2',class_weight='balanced',random
_state=0)
    clf.fit(final_X_tr,y_tr)
    clf_sgd = CalibratedClassifierCV(base_estimator=clf,method='sigmoid',cv='prefit')
    clf_sgd.fit(final_X_tr,y_tr)

    # predict the response on the traininig
    pred_tr = clf_sgd.predict_proba(final_X_tr)
    pred_tr=(pred_tr)[:,1]
    roc_tr.append(roc_auc_score(y_tr,pred_tr))

    # predict the response on the crossvalidation
    pred_cv = clf_sgd.predict_proba(final_X_cv)
    pred_cv=(pred_cv)[:,1]
    roc_cv.append(roc_auc_score(y_cv,pred_cv))

    #finding best c using loop
    if roc_auc_score(y_cv,pred_cv)>max_auc_score:
        best_alpha=i
        max_auc_score=roc_auc_score(y_cv,pred_cv)

print(best_alpha)
print(max_auc_score)
alpha_tfidf_linearsvm=best_alpha
auc_tfidf_linearsvm=max_auc_score
```

```
0.0001
0.9506286808935704
```

In [109]:

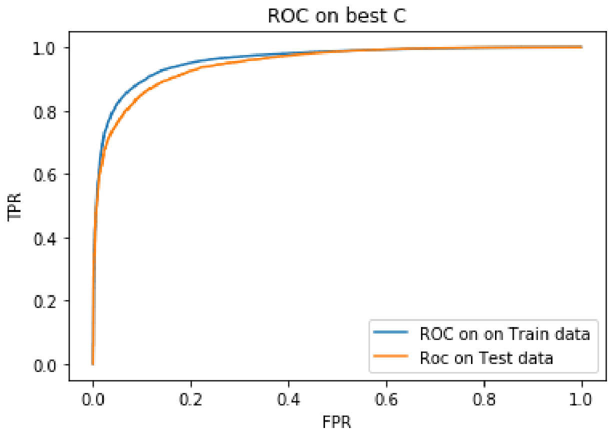
```
# plotting curve between between AUC of cv and train with log of tuned parameter
logalpha=np.log(tuned_parameters)
plt.plot(logalpha,roc_tr,label="AUC-train")
plt.plot(logalpha,roc_cv ,label="AUC-cv")
plt.legend()
plt.xlabel('tuned_Hyperparameter')
plt.ylabel('AUC')
plt.title('AUC Score vs log(tuned_hyperparameter)')
plt.show()
```



In [110]:

```
#1)Training the model using best C
```

```
clf=linear_model.SGDClassifier(alpha=alpha_tfidf_linearsvm,loss='hinge',penalty='l2',class_weight=
'balanced',random_state=0)
clf.fit(final_X_tr,y_tr)
clf_sgd = CalibratedClassifierCV(base_estimator=clf,method='sigmoid')
clf_sgd.fit(final_X_tr,y_tr)
#predicting probability on Test data
pred_test = clf_sgd.predict_proba(final_X_test)
pred_test=(pred_test)[:,1]
#predicting probablity of Training data
pred_tr = clf_sgd.predict_proba(final_X_tr)
pred_tr=(pred_tr)[:,1]


#2)Plotting Roc Curve
#Reference for finding fpr an tpr :
#https://www.programcreek.com/python/example/81207/sklearn.metrics.roc_curve
fpr_tr, tpr_tr, threshold_train = metrics.roc_curve(y_tr, pred_tr)
fpr_test, tpr_test, threshold_test = metrics.roc_curve(y_test, pred_test)
plt.plot(fpr_tr,tpr_tr ,label="ROC on on Train data")
plt.plot(fpr_test,tpr_test ,label="Roc on Test data")
plt.legend()
plt.title('ROC on best C')
plt.xlabel('FPR')
plt.ylabel('TPR')
plt.show()
```

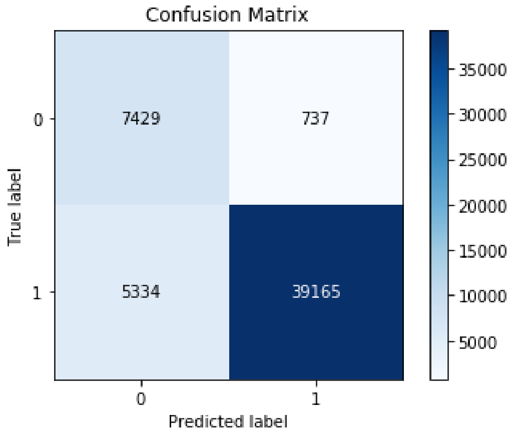


In [111]:

```
#plotting the confusion matrix on train data
#Reference:
#https://scikit-learn.org/stable/modules/generated/sklearn.metrics.confusion_matrix.html

prediction=clf.predict(final_X_tr)
skplt.plot_confusion_matrix(y_tr ,prediction)
```

Out[111]:

```
<matplotlib.axes._subplots.AxesSubplot at 0x7f76b95a0668>
```

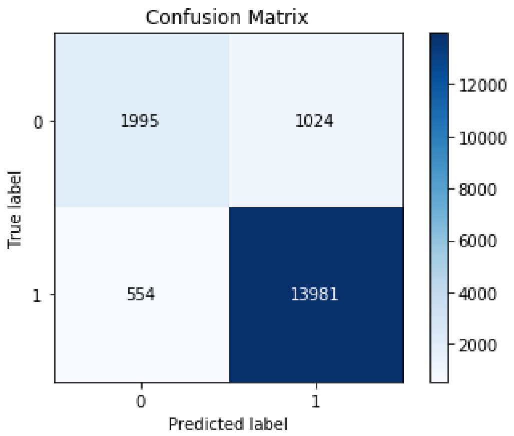


In [112]:

```
#plotting the confusion matrix on test data
#Reference:
#https://scikit-learn.org/stable/modules/generated/sklearn.metrics.confusion_matrix.html

prediction=clf_sgd.predict(final_X_test)
skplt.plot_confusion_matrix(y_test ,prediction)
```

Out[112]:

```
<matplotlib.axes._subplots.AxesSubplot at 0x7f768b78f128>
```



In [113]:

```
#Reference for top features is from statistics of machine learning by pratap dangeti:
#https://books.google.co.in/books?id=C-dDDwAAQBAJ&pg=PA216&lpg=PA216&dq=coefs_with_fns%5B:-(n+%2B+
1):-
1&source=bl&ots=j18t1prZXo&sig=ACfU3U2yz8v4v3SOiSrT7fBpJauJKe80DQ&hl=en&sa=X&ved=2ahUKEwjzwK69x6Dj
XMBHYHqADIQ6AEwAXoECAgQAQ#v=onepage&q=coefs_with_fns%5B%3A-(n%20%2B%201)%3A-1&f=false
n=10
feature_names = count_vect.get_feature_names()
coefs_with_fns = sorted(zip(clf.coef_[0], feature_names))
top_n_coefs = zip(coefs_with_fns[:n], coefs_with_fns[:-(n + 1):-1])
print("\tNegative-feature\t\t\t\tPositive-feature")
print("-----------------------------------------------------------------------------------------
-------")
for (coef_1, fn_1), (coef_2, fn_2) in top_n_coefs:
        print("\t%.4f\t%-15s\t\t\t\t%.4f\t%-15s" % (coef_1, fn_1, coef_2, fn_2))
```

```
 Negative-feature    Positive-feature
----------------------------------------------------------------------------------------------------
 -4.1010 bottled           5.3462 chicago
 -3.6591 decrease          4.7120 blues
 -3.5492 gums              4.1923 americans
 -3.5259 bottles           3.8956 coupons
 -3.3053 desired           3.8487 donate
 -3.0847 fun               3.5042 business
 -2.8367 golean            3.3669 cheers
 -2.8352 delighted         3.3084 guaranteed
 -2.7977 department        3.2349 cough
 -2.7617 coats             3.0967 degree
```

Print the best feature for each of the positive and negative classes

[5.1.3] Applying Linear SVM on AVG W2V, SET 3

In [114]:

```
#Spliting entire data to train,test and cross validation
X=np.array(preprocessed_reviews)
y = np.array(final['Score'])

#https://scikit-learn.org/stable/modules/generated/sklearn.model_selection.TimeSeriesSplit.html
```

```
tscv = TimeSeriesSplit(n_splits=4)
for train_index, test_index in tscv.split(X):
    X_1, X_test = X[train_index], X[test_index]
    y_1, y_test = y[train_index], y[test_index]

tscv = TimeSeriesSplit(n_splits=3)
for train_index, test_index in tscv.split(X_1):
    X_tr, X_cv = X_1[train_index], X_1[test_index]
    y_tr, y_cv = y_1[train_index], y_1[test_index]

#converting Reviews to Bag of words after splitting to avoid data leakage problem
count_vect = CountVectorizer(min_df=10,max_features=5000)
final_X_tr=count_vect.fit_transform(X_tr)
final_X_test=count_vect.transform(X_test)
final_X_cv=count_vect.transform(X_cv)

# average Word2Vec
# compute average word2vec for each review.
list_of_sentance_tr=[]
for sentance in X_tr:
    list_of_sentance_tr.append(sentance.split())
final_X_tr = []; # the avg-w2v for each sentence/review is stored in this list
for sent in tqdm(list_of_sentance_tr): # 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
    final_X_tr.append(sent_vec)


list_of_sentance_cv=[]
for sentance in X_cv:
    list_of_sentance_cv.append(sentance.split())
final_X_cv = []; # the avg-w2v for each sentence/review is stored in this list
for sent in tqdm(list_of_sentance_cv): # 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
    final_X_cv.append(sent_vec)


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

```
100%|██████████| 52665/52665 [01:58<00:00, 443.75it/s]
100%|██████████| 17554/17554 [00:41<00:00, 427.28it/s]
100%|██████████| 17554/17554 [00:45<00:00, 387.21it/s]
```

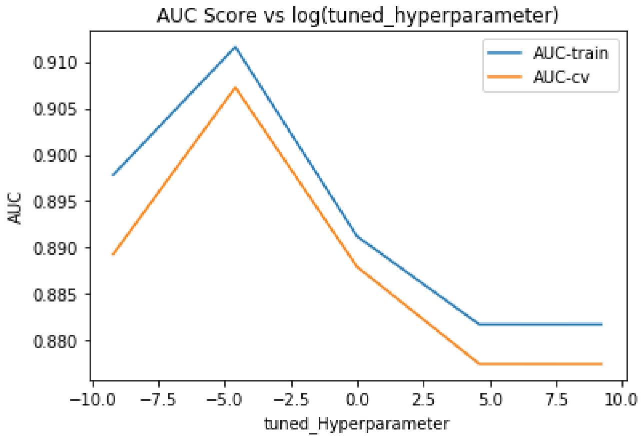
In [115]:

```
#Calculating for finding Best alpha
#predic_proba reference:
#https://stackoverflow.com/questions/37089177/probability-prediction-method-of-
kneighborsclassifier-returns-only-0-and-1
#https://discuss.analyticsvidhya.com/t/what-is-the-difference-between-predict-and-predict-proba/67
376/3
roc_tr=[]
roc_cv=[]
max_auc_score=0
best_alpha=0
tuned_parameters =[10**-4, 10**-2, 10**0, 10**2, 10**4]
for i in tuned_parameters:
    clf=linear_model.SGDClassifier(alpha=i,loss='hinge',penalty='l2',class_weight='balanced',random
_state=0)
    clf.fit(final_X_tr,y_tr)
    clf_sgd = CalibratedClassifierCV(base_estimator=clf,method='sigmoid',cv='prefit')
    clf_sgd.fit(final_X_tr,y_tr)

    # predict the response on the traininig
    pred_tr = clf_sgd.predict_proba(final_X_tr)
    pred_tr=(pred_tr)[:,1]
    roc_tr.append(roc_auc_score(y_tr,pred_tr))

    # predict the response on the crossvalidation
    pred_cv = clf_sgd.predict_proba(final_X_cv)
    pred_cv=(pred_cv)[:,1]
    roc_cv.append(roc_auc_score(y_cv,pred_cv))

    #finding best c using loop
    if roc_auc_score(y_cv,pred_cv)>max_auc_score:
        best_alpha=i
        max_auc_score=roc_auc_score(y_cv,pred_cv)

print(best_alpha)
print(max_auc_score)
alpha_avgw2v_linearsvm=best_alpha
auc_avgw2v_linearsvm=max_auc_score
```

```
0.01
0.9072313480757692
```

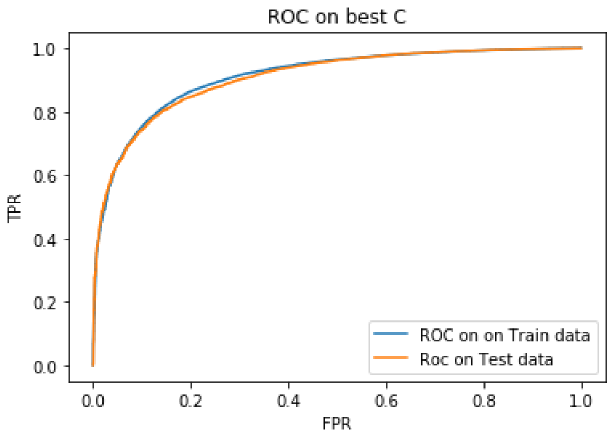
In [116]:

```
# plotting curve between between AUC of cv and train with log of tuned parameter
logalpha=np.log(tuned_parameters)
plt.plot(logalpha,roc_tr,label="AUC-train")
plt.plot(logalpha,roc_cv ,label="AUC-cv")
plt.legend()
plt.xlabel('tuned_Hyperparameter')
plt.ylabel('AUC')
plt.title('AUC Score vs log(tuned_hyperparameter)')
plt.show()
```



In [117]:

```
#1)Training the model using best C
clf=linear_model.SGDClassifier(alpha=alpha_avgw2v_linearsvm,loss='hinge',penalty='l2',class_weight
='balanced',random_state=0)
clf.fit(final_X_tr,y_tr)
clf_sgd = CalibratedClassifierCV(base_estimator=clf,method='sigmoid')
clf_sgd.fit(final_X_tr,y_tr)
#predicting probability on Test data
pred_test = clf_sgd.predict_proba(final_X_test)
pred_test=(pred_test)[:,1]
#predicting probablity of Training data
pred_tr = clf_sgd.predict_proba(final_X_tr)
pred_tr=(pred_tr)[:,1]


#2)Plotting Roc Curve
#Reference for finding fpr an tpr :
#https://www.programcreek.com/python/example/81207/sklearn.metrics.roc_curve
fpr_tr, tpr_tr, threshold_train = metrics.roc_curve(y_tr, pred_tr)
fpr_test, tpr_test, threshold_test = metrics.roc_curve(y_test, pred_test)
plt.plot(fpr_tr,tpr_tr ,label="ROC on on Train data")
plt.plot(fpr_test,tpr_test ,label="Roc on Test data")
plt.legend()
plt.title('ROC on best C')
plt.xlabel('FPR')
plt.ylabel('TPR')
plt.show()
```

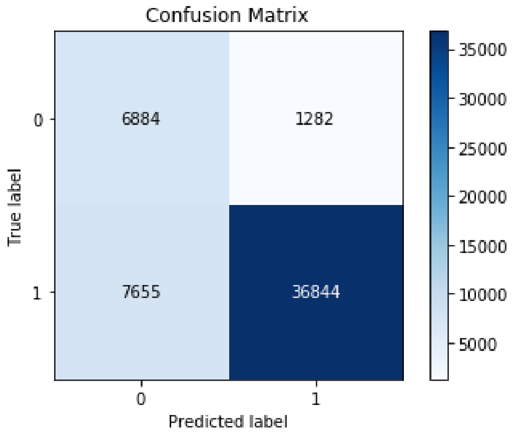


In [118]:

```
#plotting the confusion matrix on train data
#Reference:
#https://scikit-learn.org/stable/modules/generated/sklearn.metrics.confusion_matrix.html

prediction=clf.predict(final_X_tr)
skplt.plot_confusion_matrix(y_tr ,prediction)
```

Out[118]:

```
<matplotlib.axes._subplots.AxesSubplot at 0x7f7684eea518>
```

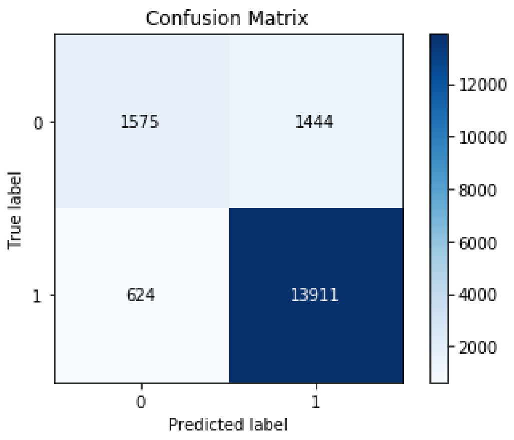


In [119]:

```
#plotting the confusion matrix on test data
#Reference:
#https://scikit-learn.org/stable/modules/generated/sklearn.metrics.confusion_matrix.html

prediction=clf_sgd.predict(final_X_test)
skplt.plot_confusion_matrix(y_test ,prediction)
```

Out[119]:

```
<matplotlib.axes._subplots.AxesSubplot at 0x7f7679f56ac8>
```



[5.1.4] Applying Linear SVM on TFIDF W2V, SET 4

In [35]:

```
# S = ["abc def pqr", "def def def abc", "pqr pqr def"]
X=preprocessed_reviews
y=np.array(final['Score'])
X_1, X_test, y_1, y_test = train_test_split(X, y, test_size=0.3, random_state=0)
X_tr, X_cv, y_tr, y_cv = train_test_split(X_1, y_1, test_size=0.3)

model = TfidfVectorizer(min_df=10, max_features=5000)
tf_idf_matrix = model.fit_transform(X_tr)
# 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 [36]:

```
# 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

list_of_sentance_tr=[]
for sentance in X_tr:
    list_of_sentance_tr.append(sentance.split())
final_X_tr = []; # the tfidf-w2v for each sentence/review is stored in this list
row=0;
for sent in tqdm(list_of_sentance_tr): # 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
    final_X_tr.append(sent_vec)
    row += 1


list_of_sentance_cv=[]
for sentance in X_cv:
    list_of_sentance_cv.append(sentance.split())
final_X_cv = []; # the tfidf-w2v for each sentence/review is stored in this list
row=0;
for sent in tqdm(list_of_sentance_cv): # 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
    final_X_cv.append(sent_vec)
    row += 1


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

```
100%|██████████| 43008/43008 [03:54<00:00, 183.45it/s]
100%|██████████| 18433/18433 [01:40<00:00, 182.76it/s]
100%|██████████| 26332/26332 [02:22<00:00, 184.54it/s]
```

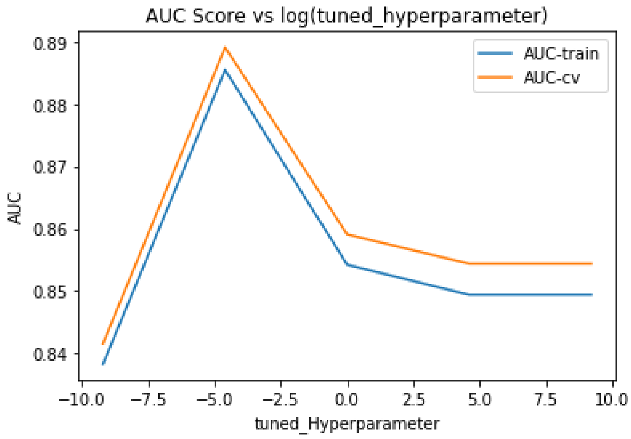
In [37]:

```
#Calculating for finding Best alpha
#predic_proba reference:
#https://stackoverflow.com/questions/37089177/probability-prediction-method-of-
kneighborsclassifier-returns-only-0-and-1
#https://discuss.analyticsvidhya.com/t/what-is-the-difference-between-predict-and-predict-proba/67
376/3
roc_tr=[]
roc_cv=[]
max_auc_score=0
best_alpha=0
tuned_parameters =[10**-4, 10**-2, 10**0, 10**2, 10**4]
for i in tuned_parameters:
    clf=linear_model.SGDClassifier(alpha=i,loss='hinge',penalty='l2',class_weight='balanced',random
_state=0)
    clf.fit(final_X_tr,y_tr)
    clf_sgd = CalibratedClassifierCV(base_estimator=clf,method='sigmoid',cv='prefit')
    clf_sgd.fit(final_X_tr,y_tr)

    # predict the response on the training
    # predict the response on the training
    pred_tr = clf_sgd.predict_proba(final_X_tr)
    pred_tr=(pred_tr)[:,1]
    roc_tr.append(roc_auc_score(y_tr,pred_tr))

    # predict the response on the crossvalidation
    pred_cv = clf_sgd.predict_proba(final_X_cv)
    pred_cv=(pred_cv)[:,1]
    roc_cv.append(roc_auc_score(y_cv,pred_cv))

    #finding best c using loop
    if roc_auc_score(y_cv,pred_cv)>max_auc_score:
        best_alpha=i
        max_auc_score=roc_auc_score(y_cv,pred_cv)

print(best_alpha)
print(max_auc_score)
alpha_tfidfw2v_linearsvm=best_alpha
auc_tfidfw2v_linearsvm=max_auc_score
```

```
0.01
0.8891595082376444
```

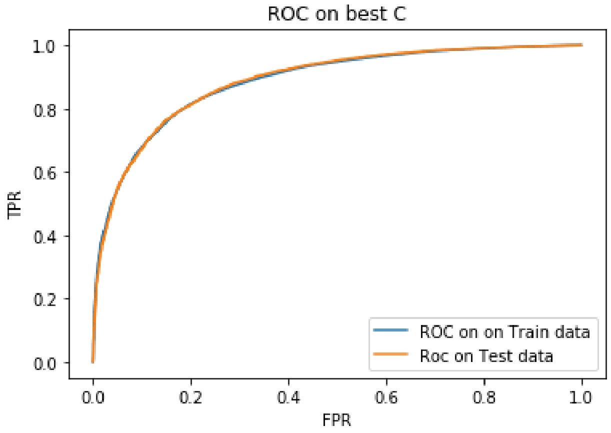
In [38]:

```
# plotting curve between between AUC of cv and train with log of tuned parameter
logalpha=np.log(tuned_parameters)
plt.plot(logalpha,roc_tr,label="AUC-train")
plt.plot(logalpha,roc_cv ,label="AUC-cv")
plt.legend()
plt.xlabel('tuned_Hyperparameter')
plt.ylabel('AUC')
plt.title('AUC Score vs log(tuned_hyperparameter)')
plt.show()
```



In [39]:

```
#1)Training the model using best C
clf=linear_model.SGDClassifier(alpha=alpha_tfidfw2v_linearsvm,loss='hinge',penalty='l2',class_weigh
t='balanced',random_state=0)
clf.fit(final_X_tr,y_tr)
clf_sgd = CalibratedClassifierCV(base_estimator=clf,method='sigmoid')
clf_sgd.fit(final_X_tr,y_tr)
#predicting probability on Test data
pred_test = clf_sgd.predict_proba(final_X_test)
pred_test=(pred_test)[:,1]
#predicting probablity of Training data
pred_tr = clf_sgd.predict_proba(final_X_tr)
pred_tr=(pred_tr)[:,1]


#2)Plotting Roc Curve
#Reference for finding fpr an tpr :
#https://www.programcreek.com/python/example/81207/sklearn.metrics.roc_curve
fpr_tr, tpr_tr, threshold_train = metrics.roc_curve(y_tr, pred_tr)
fpr_test, tpr_test, threshold_test = metrics.roc_curve(y_test, pred_test)
plt.plot(fpr_tr,tpr_tr ,label="ROC on on Train data")
plt.plot(fpr_tr,tpr_tr ,label="ROC on on Train data")
plt.plot(fpr_test,tpr_test ,label="Roc on Test data")
plt.legend()
plt.title('ROC on best C')
plt.xlabel('FPR')
plt.ylabel('TPR')
plt.show()
```

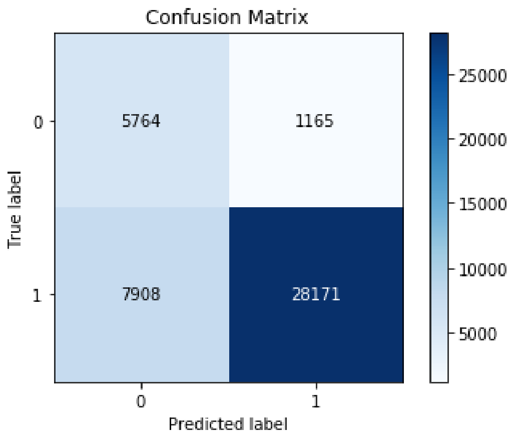


In [40]:

```
#plotting the confusion matrix on train data
#Reference:
#https://scikit-learn.org/stable/modules/generated/sklearn.metrics.confusion_matrix.html

prediction=clf.predict(final_X_tr)
skplt.plot_confusion_matrix(y_tr ,prediction)
```

Out[40]:

```
<matplotlib.axes._subplots.AxesSubplot at 0x7f687c81b3c8>
```

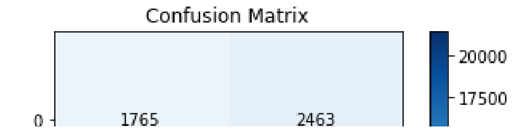


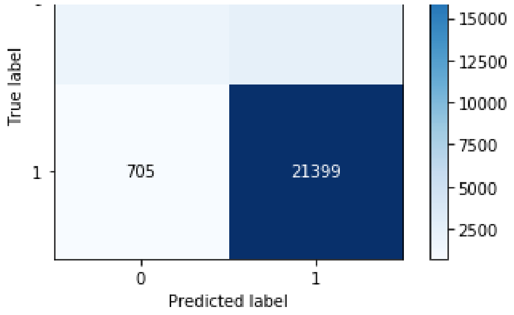
In [41]:

```
#plotting the confusion matrix on test data
#Reference:
#https://scikit-learn.org/stable/modules/generated/sklearn.metrics.confusion_matrix.html

prediction=clf_sgd.predict(final_X_test)
skplt.plot_confusion_matrix(y_test ,prediction)
```

Out[41]:

```
<matplotlib.axes._subplots.AxesSubplot at 0x7f687c71f940>
```





[5.2] RBF SVM

[5.2.1] Applying RBF SVM on BOW, SET 1

In [72]:

```
X=np.array(preprocessed_reviews)
y = np.array(final['Score'])
# Taking only 15k datapoints because of memory constraints
X=X[:15000]
y=y[:15000]

## split the data set into train and test
X_1, X_test, y_1, y_test = train_test_split(X, y, test_size=0.3, random_state=1)

# split the train data set into cross validation train and cross validation test
X_tr, X_cv, y_tr, y_cv = train_test_split(X_1, y_1, test_size=0.3,random_state=1)


#converting Reviews to Bag of words after splitting to avoid data leakage problem
count_vect = CountVectorizer(max_features = 500,min_df = 10)
final_X_tr=count_vect.fit_transform(X_tr)
final_X_test=count_vect.transform(X_test)
final_X_cv=count_vect.transform(X_cv)
```

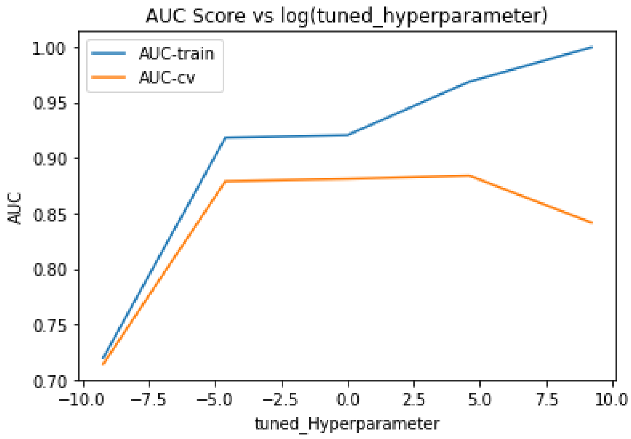
In [73]:

```
#Calculating for finding Best alpha
#predic_proba reference:
#https://stackoverflow.com/questions/37089177/probability-prediction-method-of-
kneighborsclassifier-returns-only-0-and-1
#https://discuss.analyticsvidhya.com/t/what-is-the-difference-between-predict-and-predict-proba/67
376/3
roc_tr=[]
roc_cv=[]
max_auc_score=0
best_alpha=0
tuned_parameters =[10**-4, 10**-2, 10**0, 10**2, 10**4]
for i in tuned_parameters:
    clf=SVC(C=i,probability=True)
    # fitting the model on train data
    clf.fit(final_X_tr,y_tr)

    # predict the response on the traininig
    pred_tr = clf.predict_proba(final_X_tr)
    pred_tr=(pred_tr)[:,1]
    roc_tr.append(roc_auc_score(y_tr,pred_tr))

    # predict the response on the crossvalidation
    pred_cv = clf.predict_proba(final_X_cv)
    pred_cv=(pred_cv)[:,1]
    roc_cv.append(roc_auc_score(y_cv,pred_cv))

    #finding best c using loop
    if roc_auc_score(y_cv,pred_cv)>max_auc_score:
        best_alpha=i
        max_auc_score=roc_auc_score(y_cv,pred_cv)

print(best_alpha)
print(max_auc_score)
print(max_auc_score)
alpha_bow_rbfsvm=best_alpha
auc_bow_rbfsvm=max_auc_score
```

```
100
0.8838945130663491
```

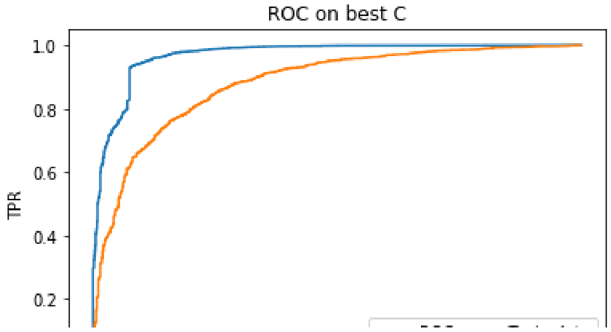
In [74]:

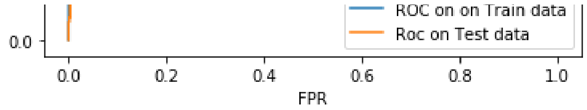
```
# plotting curve between between AUC of cv and train with log of tuned parameter
logalpha=np.log(tuned_parameters)
plt.plot(logalpha,roc_tr,label="AUC-train")
plt.plot(logalpha,roc_cv ,label="AUC-cv")
plt.legend()
plt.xlabel('tuned_Hyperparameter')
plt.ylabel('AUC')
plt.title('AUC Score vs log(tuned_hyperparameter)')
plt.show()
```



In [75]:

```
#1)Training the model using best C
clf=SVC(C=alpha_bow_rbfsvm,probability=True)
clf.fit(final_X_tr,y_tr)
#predicting probability on Test data
pred_test = clf.predict_proba(final_X_test)
pred_test=(pred_test)[:,1]
#predicting probablity of Training data
pred_tr = clf.predict_proba(final_X_tr)
pred_tr=(pred_tr)[:,1]

#2)Plotting Roc Curve
#Reference for finding fpr an tpr :
#https://www.programcreek.com/python/example/81207/sklearn.metrics.roc_curve
fpr_tr, tpr_tr, threshold_train = metrics.roc_curve(y_tr, pred_tr)
fpr_test, tpr_test, threshold_test = metrics.roc_curve(y_test, pred_test)
plt.plot(fpr_tr,tpr_tr ,label="ROC on on Train data")
plt.plot(fpr_test,tpr_test ,label="Roc on Test data")
plt.legend()
plt.title('ROC on best C')
plt.xlabel('FPR')
plt.ylabel('TPR')
plt.show()
```



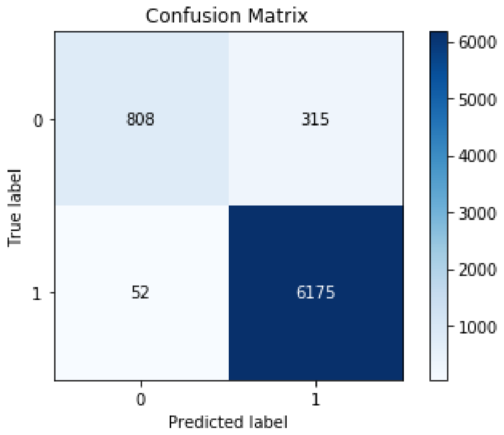


In [76]:

```
#plotting the confusion matrix on train data
#Reference:
#https://scikit-learn.org/stable/modules/generated/sklearn.metrics.confusion_matrix.html

prediction=clf.predict(final_X_tr)
skplt.plot_confusion_matrix(y_tr ,prediction)
```

Out[76]:

```
<matplotlib.axes._subplots.AxesSubplot at 0x7f768f6869e8>
```

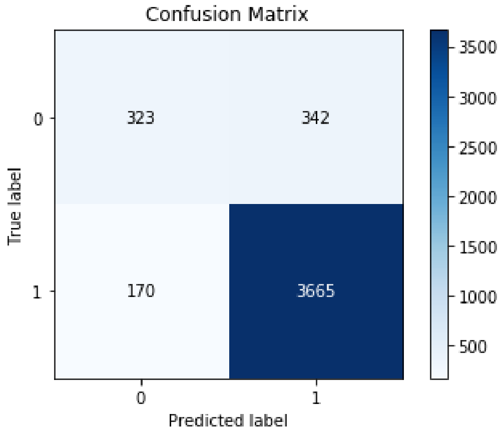


In [77]:

```
#plotting the confusion matrix on test data
#Reference:
#https://scikit-learn.org/stable/modules/generated/sklearn.metrics.confusion_matrix.html

prediction=clf.predict(final_X_test)
skplt.plot_confusion_matrix(y_test ,prediction)
```

Out[77]:

```
<matplotlib.axes._subplots.AxesSubplot at 0x7f7685f1c8d0>
```



[5.2.2] Applying RBF SVM on TFIDF, SET 2

In [78]:

```
X=np.array(preprocessed_reviews)
```

```
y = np.array(final['Score'])
# Taking only 15k datapoints because of memory constraints
X=X[:15000]
y=y[:15000]

## split the data set into train and test
X_1, X_test, y_1, y_test = train_test_split(X, y, test_size=0.3, random_state=1)

# split the train data set into cross validation train and cross validation test
X_tr, X_cv, y_tr, y_cv = train_test_split(X_1, y_1, test_size=0.3,random_state=1)

#converting Reviews to Bag of words after splitting to avoid data leakage problem
tf_idf_vect = TfidfVectorizer(ngram_range=(1,2),min_df=10,max_features=500)
final_X_tr=tf_idf_vect.fit_transform(X_tr)
final_X_test=tf_idf_vect.transform(X_test)
final_X_cv=tf_idf_vect.transform(X_cv)
```

In [79]:

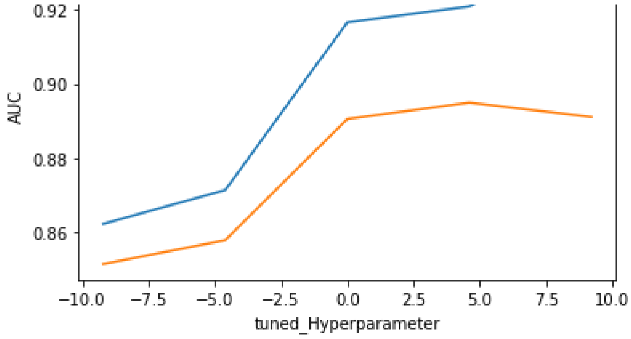
```
#Calculating for finding Best alpha
#predic_proba reference:
#https://stackoverflow.com/questions/37089177/probability-prediction-method-of-
kneighborsclassifier-returns-only-0-and-1
#https://discuss.analyticsvidhya.com/t/what-is-the-difference-between-predict-and-predict-proba/67
376/3
roc_tr=[]
roc_cv=[]
max_auc_score=0
best_alpha=0
tuned_parameters =[10**-4, 10**-2, 10**0, 10**2, 10**4]
for i in tuned_parameters:
    clf=SVC(C=i,probability=True)
    # fitting the model on train data
    clf.fit(final_X_tr,y_tr)

    # predict the response on the traininig
    pred_tr = clf.predict_proba(final_X_tr)
    pred_tr=(pred_tr)[:,1]
    roc_tr.append(roc_auc_score(y_tr,pred_tr))

    # predict the response on the crossvalidation
    pred_cv = clf.predict_proba(final_X_cv)
    pred_cv=(pred_cv)[:,1]
    roc_cv.append(roc_auc_score(y_cv,pred_cv))

    #finding best c using loop
    if roc_auc_score(y_cv,pred_cv)>max_auc_score:
        best_alpha=i
        max_auc_score=roc_auc_score(y_cv,pred_cv)

print(best_alpha)
print(max_auc_score)
alpha_tfidf_rbfsvm=best_alpha
auc_tfidf_rbfsvm=max_auc_score
```

```
100
0.8949367752322688
```

In [80]:

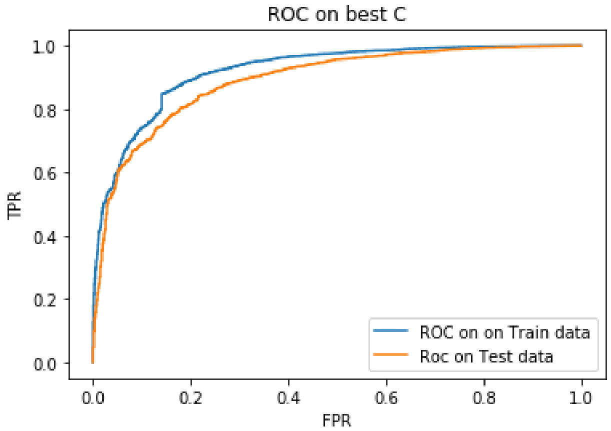
```
# plotting curve between between AUC of cv and train with log of tuned parameter
logalpha=np.log(tuned_parameters)
plt.plot(logalpha,roc_tr,label="AUC-train")
plt.plot(logalpha,roc_cv ,label="AUC-cv")
plt.legend()
plt.xlabel('tuned_Hyperparameter')
plt.ylabel('AUC')
plt.title('AUC Score vs log(tuned_hyperparameter)')
plt.show()
```

AUC Score vs log(tuned_hyperparameter)
0.94
AUC-train
AUC-cv



In [81]:

```
#1)Training the model using best C
clf=SVC(C=alpha_tfidf_rbfsvm,probability=True)
clf.fit(final_X_tr,y_tr)
#predicting probability on Test data
pred_test = clf.predict_proba(final_X_test)
pred_test=(pred_test)[:,1]
#predicting probablity of Training data
pred_tr = clf.predict_proba(final_X_tr)
pred_tr=(pred_tr)[:,1]

#2)Plotting Roc Curve
#Reference for finding fpr an tpr :
#https://www.programcreek.com/python/example/81207/sklearn.metrics.roc_curve
fpr_tr, tpr_tr, threshold_train = metrics.roc_curve(y_tr, pred_tr)
fpr_test, tpr_test, threshold_test = metrics.roc_curve(y_test, pred_test)
plt.plot(fpr_tr,tpr_tr ,label="ROC on on Train data")
plt.plot(fpr_test,tpr_test ,label="Roc on Test data")
plt.legend()
plt.title('ROC on best C')
plt.xlabel('FPR')
plt.ylabel('TPR')
plt.show()
```

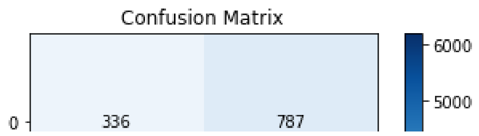


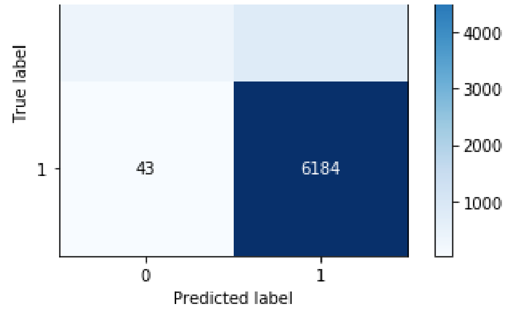
In [82]:

```
#plotting the confusion matrix on train data
#Reference:
#https://scikit-learn.org/stable/modules/generated/sklearn.metrics.confusion_matrix.html

prediction=clf.predict(final_X_tr)
skplt.plot_confusion_matrix(y_tr ,prediction)
```

Out[82]:

```
<matplotlib.axes._subplots.AxesSubplot at 0x7f76835a1668>
```



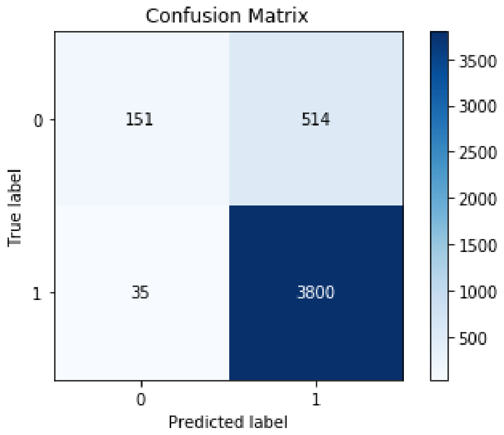


In [84]:

```
#plotting the confusion matrix on test data
#Reference:
#https://scikit-learn.org/stable/modules/generated/sklearn.metrics.confusion_matrix.html

prediction=clf.predict(final_X_test)
skplt.plot_confusion_matrix(y_test ,prediction)
```

Out[84]:

```
<matplotlib.axes._subplots.AxesSubplot at 0x7f76b9474390>
```



[5.2.3] Applying RBF SVM on AVG W2V, SET 3

In [85]:

```
X=np.array(preprocessed_reviews)
y = np.array(final['Score'])
# Taking only 15k datapoints because of memory constraints
X=X[:15000]
y=y[:15000]

## split the data set into train and test
X_1, X_test, y_1, y_test = train_test_split(X, y, test_size=0.3, random_state=1)

# split the train data set into cross validation train and cross validation test
X_tr, X_cv, y_tr, y_cv = train_test_split(X_1, y_1, test_size=0.3,random_state=1)

#converting Reviews to Bag of words after splitting to avoid data leakage problem
count_vect = CountVectorizer(min_df=10,max_features=500)
final_X_tr=count_vect.fit_transform(X_tr)
final_X_test=count_vect.transform(X_test)
final_X_cv=count_vect.transform(X_cv)

# average Word2Vec
# compute average word2vec for each review.
list_of_sentance_tr=[]
for sentance in X_tr:
    list_of_sentance_tr.append(sentance.split())
final_X_tr = []; # the avg-w2v for each sentence/review is stored in this list
for sent in tqdm(list_of_sentance_tr): # 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
    final_X_tr.append(sent_vec)


list_of_sentance_cv=[]
for sentance in X_cv:
    list_of_sentance_cv.append(sentance.split())
final_X_cv = []; # the avg-w2v for each sentence/review is stored in this list
for sent in tqdm(list_of_sentance_cv): # 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
    final_X_cv.append(sent_vec)


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

```
100%|██████████| 7350/7350 [00:16<00:00, 458.06it/s]
100%|██████████| 3150/3150 [00:06<00:00, 459.43it/s]
100%|██████████| 4500/4500 [00:09<00:00, 459.39it/s]
```

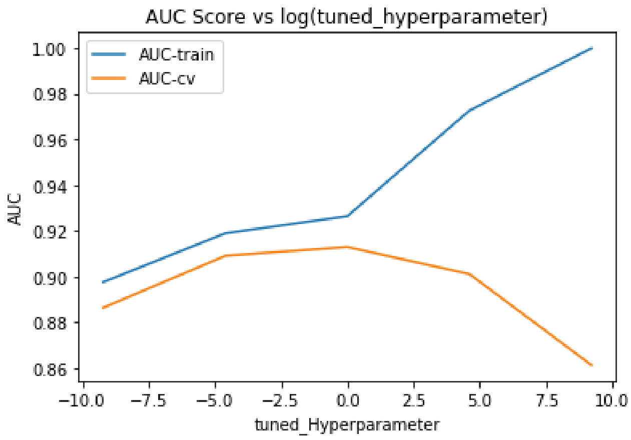
In [86]:

```
#Calculating for finding Best alpha
#predic_proba reference:
#https://stackoverflow.com/questions/37089177/probability-prediction-method-of-
kneighborsclassifier-returns-only-0-and-1
#https://discuss.analyticsvidhya.com/t/what-is-the-difference-between-predict-and-predict-proba/67
376/3
roc_tr=[]
roc_cv=[]
max_auc_score=0
best_alpha=0
tuned_parameters =[10**-4, 10**-2, 10**0, 10**2, 10**4]
for i in tuned_parameters:
    clf=SVC(C=i,probability=True)
    # fitting the model on train data
    clf.fit(final_X_tr,y_tr)

    # predict the response on the traininig
    pred_tr = clf.predict_proba(final_X_tr)
    pred_tr=(pred_tr)[:,1]
    roc_tr.append(roc_auc_score(y_tr,pred_tr))

    # predict the response on the crossvalidation
    pred_cv = clf.predict_proba(final_X_cv)
    pred_cv = clf.predict_proba(final_X_cv)
    pred_cv=(pred_cv)[:,1]
    roc_cv.append(roc_auc_score(y_cv,pred_cv))

    #finding best c using loop
    if roc_auc_score(y_cv,pred_cv)>max_auc_score:
        best_alpha=i
        max_auc_score=roc_auc_score(y_cv,pred_cv)

print(best_alpha)
print(max_auc_score)
alpha_avgw2v_rbfsvm=best_alpha
auc_avgw2v_rbfsvm=max_auc_score
```

```
1
0.9130136072623471
```

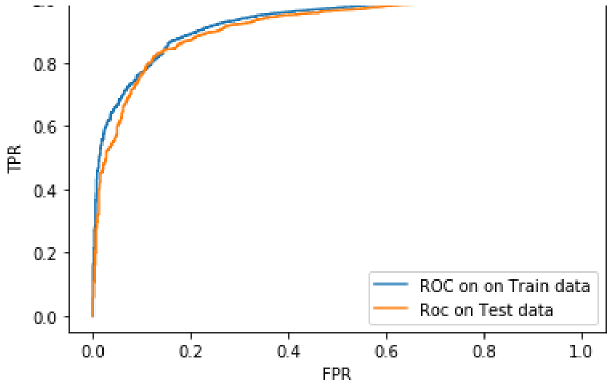
In [87]:

```
# plotting curve between between AUC of cv and train with log of tuned parameter
logalpha=np.log(tuned_parameters)
plt.plot(logalpha,roc_tr,label="AUC-train")
plt.plot(logalpha,roc_cv ,label="AUC-cv")
plt.legend()
plt.xlabel('tuned_Hyperparameter')
plt.ylabel('AUC')
plt.title('AUC Score vs log(tuned_hyperparameter)')
plt.show()
```



In [88]:

```
#1)Training the model using best C
clf=SVC(C=alpha_avgw2v_rbfsvm,probability=True)
clf.fit(final_X_tr,y_tr)
#predicting probability on Test data
pred_test = clf.predict_proba(final_X_test)
pred_test=(pred_test)[:,1]
#predicting probablity of Training data
pred_tr = clf.predict_proba(final_X_tr)
pred_tr=(pred_tr)[:,1]

#2)Plotting Roc Curve
#Reference for finding fpr an tpr :
#https://www.programcreek.com/python/example/81207/sklearn.metrics.roc_curve
fpr_tr, tpr_tr, threshold_train = metrics.roc_curve(y_tr, pred_tr)
fpr_test, tpr_test, threshold_test = metrics.roc_curve(y_test, pred_test)
plt.plot(fpr_tr,tpr_tr ,label="ROC on on Train data")
plt.plot(fpr_test,tpr_test ,label="Roc on Test data")
plt.legend()
plt.title('ROC on best C')
plt.xlabel('FPR')
plt.ylabel('TPR')
plt.show()
```

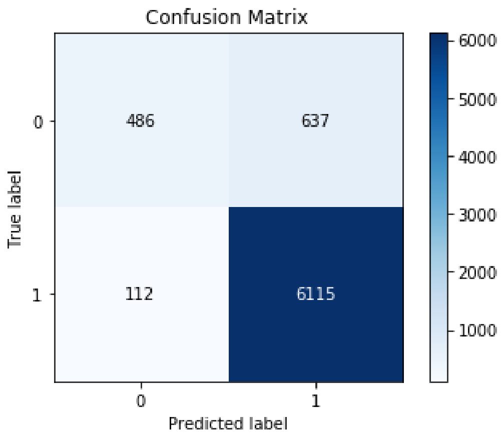


In [89]:

```
#plotting the confusion matrix on train data
#Reference:
#https://scikit-learn.org/stable/modules/generated/sklearn.metrics.confusion_matrix.html

prediction=clf.predict(final_X_tr)
skplt.plot_confusion_matrix(y_tr ,prediction)
```

Out[89]:

```
<matplotlib.axes._subplots.AxesSubplot at 0x7f7683c23f28>
```

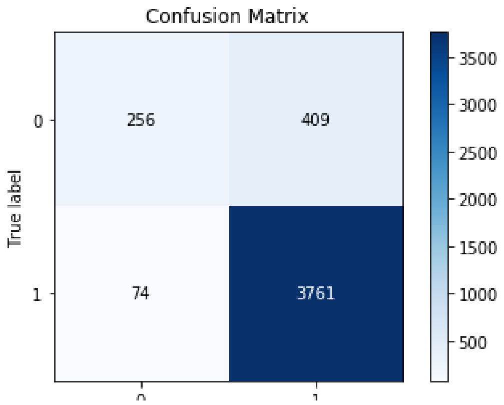


In [90]:

```
#plotting the confusion matrix on test data
#Reference:
#https://scikit-learn.org/stable/modules/generated/sklearn.metrics.confusion_matrix.html

prediction=clf.predict(final_X_test)
skplt.plot_confusion_matrix(y_test ,prediction)
```

Out[90]:

```
<matplotlib.axes._subplots.AxesSubplot at 0x7f7684455710>
```



[5.2.4] Applying RBF SVM on TFIDF W2V, SET 4

In [42]:

```
# S = ["abc def pqr", "def def def abc", "pqr pqr def"]
X=preprocessed_reviews
y=np.array(final['Score'])
# Taking only 15k datapoints because of memory constraints
X=X[:15000]
y=y[:15000]
X_1, X_test, y_1, y_test = train_test_split(X, y, test_size=0.3, random_state=0)
X_tr, X_cv, y_tr, y_cv = train_test_split(X_1, y_1, test_size=0.3)

model = TfidfVectorizer(min_df=10, max_features=5000)
tf_idf_matrix = model.fit_transform(X_tr)
# 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 [43]:

```
# 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

list_of_sentance_tr=[]
for sentance in X_tr:
    list_of_sentance_tr.append(sentance.split())
final_X_tr = []; # the tfidf-w2v for each sentence/review is stored in this list
row=0;
for sent in tqdm(list_of_sentance_tr): # 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
    final_X_tr.append(sent_vec)
    row += 1


list_of_sentance_cv=[]
for sentance in X_cv:
    list_of_sentance_cv.append(sentance.split())
final_X_cv = []; # the tfidf-w2v for each sentence/review is stored in this list
row=0;
for sent in tqdm(list_of_sentance_cv): # 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
    final_X_cv.append(sent_vec)
    row += 1
```

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

```
100%|██████████| 7350/7350 [00:31<00:00, 235.69it/s]
100%|██████████| 3150/3150 [00:12<00:00, 249.87it/s]
100%|██████████| 4500/4500 [00:17<00:00, 250.16it/s]
```

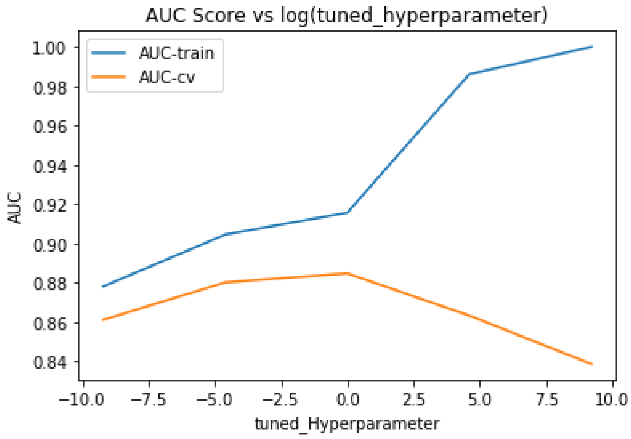
In [44]:

```
#Calculating for finding Best alpha
#predic_proba reference:
#https://stackoverflow.com/questions/37089177/probability-prediction-method-of-
kneighborsclassifier-returns-only-0-and-1
#https://discuss.analyticsvidhya.com/t/what-is-the-difference-between-predict-and-predict-proba/67
376/3
roc_tr=[]
roc_cv=[]
max_auc_score=0
best_alpha=0
tuned_parameters =[10**-4, 10**-2, 10**0, 10**2, 10**4]
for i in tuned_parameters:
    clf=SVC(C=i,probability=True)
    # fitting the model on train data
    clf.fit(final_X_tr,y_tr)

    # predict the response on the traininig
    pred_tr = clf.predict_proba(final_X_tr)
    pred_tr=(pred_tr)[:,1]
    roc_tr.append(roc_auc_score(y_tr,pred_tr))

    # predict the response on the crossvalidation
    pred_cv = clf.predict_proba(final_X_cv)
    pred_cv=(pred_cv)[:,1]
    roc_cv.append(roc_auc_score(y_cv,pred_cv))

    #finding best c using loop
    if roc_auc_score(y_cv,pred_cv)>max_auc_score:
        best_alpha=i
        max_auc_score=roc_auc_score(y_cv,pred_cv)

print(best_alpha)
print(max_auc_score)
alpha_tfidfw2v_rbfsvm=best_alpha
auc_tfidfw2v_rbfsvm=max_auc_score
```

```
1
0.8847405454328328
```

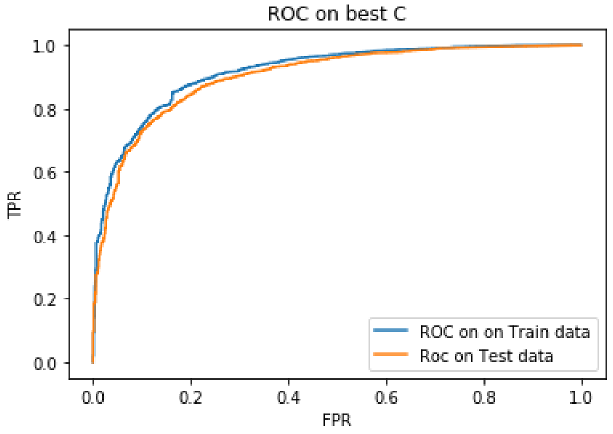
In [45]:

```
# plotting curve between between AUC of cv and train with log of tuned parameter
logalpha=np.log(tuned_parameters)
plt.plot(logalpha,roc_tr,label="AUC-train")
plt.plot(logalpha,roc_cv ,label="AUC-cv")
plt.legend()
plt.xlabel('tuned_Hyperparameter')
plt.ylabel('AUC')
plt.title('AUC Score vs log(tuned_hyperparameter)')
plt.show()
```



In [46]:

```
#1)Training the model using best C
clf=SVC(C=alpha_tfidfw2v_rbfsvm,probability=True)
clf.fit(final_X_tr,y_tr)
#predicting probability on Test data
pred_test = clf.predict_proba(final_X_test)
pred_test=(pred_test)[:,1]
#predicting probablity of Training data
pred_tr = clf.predict_proba(final_X_tr)
pred_tr=(pred_tr)[:,1]

#2)Plotting Roc Curve
#Reference for finding fpr an tpr :
#https://www.programcreek.com/python/example/81207/sklearn.metrics.roc_curve
fpr_tr, tpr_tr, threshold_train = metrics.roc_curve(y_tr, pred_tr)
fpr_test, tpr_test, threshold_test = metrics.roc_curve(y_test, pred_test)
plt.plot(fpr_tr,tpr_tr ,label="ROC on on Train data")
plt.plot(fpr_test,tpr_test ,label="Roc on Test data")
plt.legend()
plt.title('ROC on best C')
plt.xlabel('FPR')
plt.ylabel('TPR')
plt.show()
```



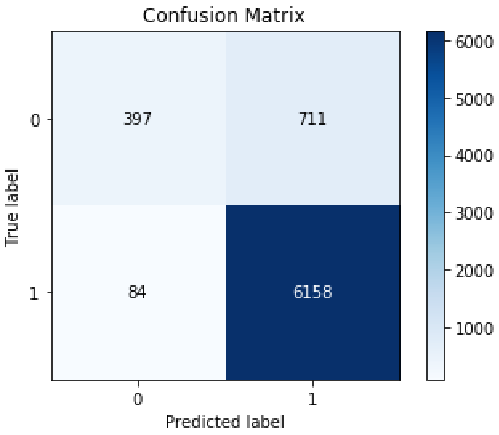
In [47]:

```
#plotting the confusion matrix on train data
#Reference:
#https://scikit-learn.org/stable/modules/generated/sklearn.metrics.confusion_matrix.html

prediction=clf.predict(final_X_tr)
```

```
skplt.plot_confusion_matrix(y_tr ,prediction)
```

Out[47]:

```
<matplotlib.axes._subplots.AxesSubplot at 0x7f68a4ace320>
```

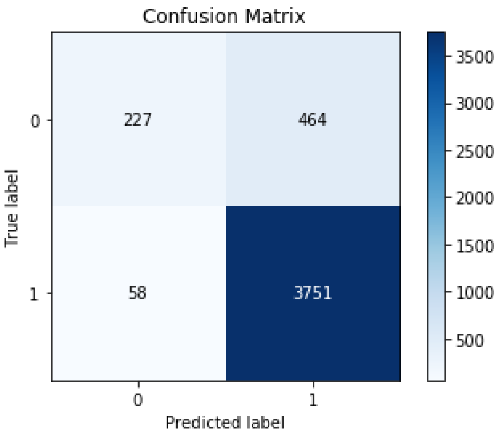


In [48]:

```
#plotting the confusion matrix on test data
#Reference:
#https://scikit-learn.org/stable/modules/generated/sklearn.metrics.confusion_matrix.html

prediction=clf.predict(final_X_test)
skplt.plot_confusion_matrix(y_test ,prediction)
```

Out[48]:

```
<matplotlib.axes._subplots.AxesSubplot at 0x7f68a4a2ca20>
```



[6] Conclusions

In [127]:

```
# Please compare all your models using Prettytable library
from prettytable import PrettyTable
x = PrettyTable()
x.field_names = ["Vectorizer", "Model_name", "Hyperameter-best-alpha","AUC"]
x.add_row(["BOW","LinearSVM",0.0001,0.929])
x.add_row(["TFIDF","LinearSVM",0.0001,0.950])
x.add_row(["AwgW2V","LinearSVM",0.01,0.907])
x.add_row(["TFIDF-W2V","LinearSVM",0.01,0.889])
x.add_row(["BOW","RbfSVM",100,0.883])
x.add_row(["TFIDF","RbfSVM",100,0.894])
x.add_row(["AwgW2V","RbfSVM",1,0.913])
x.add_row(["TFIDF-W2V","RbfSVM",1,0.884])
print(x)
```

```
+------------+------------+-----------------------+---------------------+
| Vectorizer | Model_name | Hyperamete-best-alpha |         AUC         |
+------------+------------+-----------------------+---------------------+
|    BOW     | LinearSVM  |         0.0001        |  0.929299078259954  |
|   TFIDF    | LinearSVM  |         0.0001        | 0.9506286808935704  |
|   AwgW2V   | LinearSVM  |          0.01         | 0.9072313480757692  |
| TFIDF-W2V  | LinearSVM  |          0.01         | 0.8850630350014701  |
|    BOW     |   RbfSVM   |          100          | 0.8838945130663491  |
|   TFIDF    |   RbfSVM   |          100          | 0.8949367752322688  |
|   AwgW2V   |   RbfSVM   |           1           | 0.9130136072623471  |
| TFIDF-W2V  |   RbfSVM   |           1           | 0.8649945523143604  |
+------------+------------+-----------------------+---------------------+
```