Amazon Fine Food Reviews Analysis

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

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

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

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

Attribute Information:

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

Objective:

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

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

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

[1]. Reading Data

[1.1] Loading the data

The dataset is available in two forms

1. .csv file
2. SQLite Database

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

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

In [1]:

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


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

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

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

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

from tqdm import tqdm
import os
```

In [2]:

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

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

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

filtered_data = pd.read_sql_query(""" SELECT * FROM Reviews WHERE Score != 3 LIMIT 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
2	3	B000LQOCH0	ABXLMWJIXXAIN	Natalia Corres "Natalia	1	1	1	1219017600	"Delight" says it all

Id	ProductId	UserId	ProfileName	HelpfulnessNumerator	HelpfulnessDenominator	Score	Time	Summary

In [3]:

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

In [4]:

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

```
(80668, 7)
```

Out[4]:

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

In [5]:

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

Out[5]:

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

In [6]:

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

Out[6]:

```
393063
```

[2] Exploratory Data Analysis

[2.1] Data Cleaning: Deduplication

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

In [7]:

```
display= pd.read_sql_query("""
SELECT *
FROM Reviews
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 VANI WAFE
1	138317	B000HDOPYC	AR5J8UI46CURR	Geetha Krishnan	2	2	5	1199577600	LOACK QUADRAT VANI WAFE
2	138277	B000HDOPYM	AR5J8UI46CURR	Geetha Krishnan	2	2	5	1199577600	LOACK QUADRAT VANI WAFE
3	73791	B000HDOPZG	AR5J8UI46CURR	Geetha Krishnan	2	2	5	1199577600	LOACK QUADRAT VANI WAFE
4	155049	B000PAQ75C	AR5J8UI46CURR	Geetha Krishnan	2	2	5	1199577600	LOACK QUADRAT VANI 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
```

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 [00:33<00:00, 2633.66it/s]
```

In [23]:

```
preprocessed_reviews[1500]
```

Out[23]:

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

[3.2] Preprocessing Review Summary

In [24]:

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

[4] Featurization

[4.1] BAG OF WORDS

In [25]:

```
#BoW
count_vect = CountVectorizer(max_features = 2000,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  ['able', 'absolute', 'absolutely', 'according', 'acid', 'acidic', 'across', 'a
ctual', 'actually', 'add']
==================================================
the type of count vectorizer  <class 'scipy.sparse.csr.csr_matrix'>
the shape of out text BOW vectorizer  (87773, 2000)
the number of unique words  2000
```

[4.2] Bi-Grams and n-Grams.

In [26]:

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

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

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

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

[4.3] TF-IDF

In [27]:

```
tf_idf_vect = TfidfVectorizer(ngram_range=(1,2), min_df=10,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 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 [28]:

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

In [29]:

```
# Using Google News Word2Vectors

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


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

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', 0.8363029956817627), ('awesome', 0.8307173252105713), ('good', 0.8269831538200378),
('excellent', 0.8168129920959473), ('terrific', 0.8040965795516968), ('wonderful',
0.7746241092681885), ('perfect', 0.7741086483001709), ('nice', 0.7501264214515686), ('amazing', 0.
721499502658844), ('fabulous', 0.7197446823120117)]
==================================================
[('greatest', 0.8233102560043335), ('tastiest', 0.7471528649330139), ('best', 0.7351601123809814),
('disgusting', 0.6649123430252075), ('closest', 0.658866822719574), ('nastiest',
0.6477102041244507), ('horrible', 0.6151102781295776), ('superior', 0.5948395133018494), ('awful',
0.593022346496582), ('smoothest', 0.5925673246383667)]
```

In [30]:

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

```
number of words that occured minimum 5 times  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', 'get', 'rid', '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 [31]:

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

```
100%|██████████| 87773/87773 [04:23<00:00, 333.31it/s]
```

```
87773
50
```

[4.4.1.2] TFIDF weighted W2v

In [32]:

```
# 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 [33]:

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

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

```
100%|██████████| 87773/87773 [09:42<00:00, 150.59it/s]
```

[5] Assignment 9: Random Forests

1. **Apply Random Forests & GBDT 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. **The hyper paramter tuning (Consider two hyperparameters: n_estimators & max_depth)**
   - 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

3. **Feature importance**
   - Get top 20 important features and represent them in a word cloud. Do this for BOW & TFIDF.

4. **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.

5. **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
     with X-axis as **n_estimators**, Y-axis as **max_depth**, and Z-axis as **AUC Score** , we have given the notebook which explains how to plot this 3d plot, you can find it in the same drive *3d_scatter_plot.ipynb*

(or)

   - 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
     seaborn heat maps with rows as **n_estimators**, columns as **max_depth**, and values inside the cell representing **AUC Score**
   - You choose either of the plotting techniques out of 3d plot or heat map
   - 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.

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

Note: Data Leakage

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

[5.1] Applying RF

[5.1.1] Applying Random Forests on BOW, SET 1

In [36]:

```
import numpy as np
import pandas as pd
import math
import matplotlib.pyplot as plt
from sklearn.model_selection  import train_test_split
from sklearn.metrics import accuracy_score
from sklearn.model_selection  import cross_val_score
from collections import Counter
from sklearn.metrics import accuracy_score
from sklearn import model_selection
from sklearn.metrics import roc_auc_score
from sklearn.calibration import CalibratedClassifierCV
from sklearn.tree import DecisionTreeClassifier
from sklearn.ensemble import RandomForestClassifier
import scikitplot.metrics as skplt
from wordcloud import WordCloud
from xgboost import XGBClassifier
from sklearn.model_selection import RandomizedSearchCV
```

In [37]:

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

## 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()
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)
```

Finding the optimal depth,base learners and best auc score

In [91]:

```
#Finding the optimal number of base learners and depths
base_learners = [5,10,50,100,200,500,1000]
depths= [2,3,4,5,6,7,8,9,10]
tuned_parameters={'n_estimators': base_learners, 'max_depth':depths}
clf= RandomForestClassifier(max_features='sqrt')
rf_randomsearch=RandomizedSearchCV(clf,tuned_parameters,scoring='roc_auc',n_jobs=-1)
rf_randomsearch.fit(final_X_tr,y_tr)
optimal_baselearners_rfbow=rf_randomsearch.best_estimator_.n_estimators
optimal_depth_rfbow=rf_randomsearch.best_estimator_.max_depth
optimal_aucscore_rfbow=rf_randomsearch.best_score_
print("optimal number of baselearners",optimal_baselearners_rfbow)
print("optimal max_depth",optimal_depth_rfbow)
print("Best AUCScore:", optimal_aucscore_rfbow)
```

```
optimal number of baselearners 200
optimal max_depth 10
Best AUCScore: 0.8806200238297764
```

Finding the auc scores of train and cv data

In [92]:

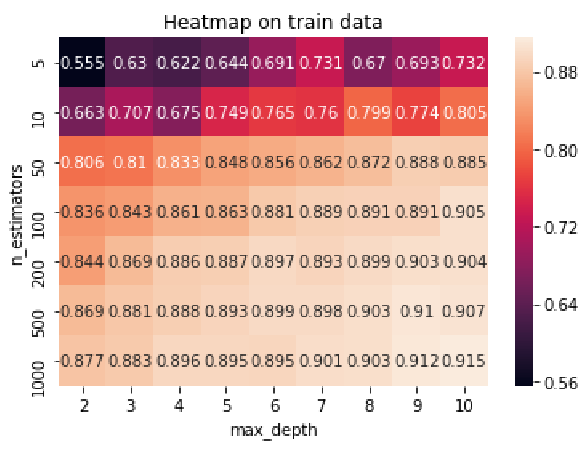
```
#finding the auc scores on train and cv data
roc_tr=[]
roc_cv=[]
```

```
baselearner=[]
depth=[]
for i in base_learners:
    for j in depths:
        clf=RandomForestClassifier(max_features='sqrt',max_depth=j,n_estimators=i)

        #fitting the model
        clf.fit(final_X_tr,y_tr)

        # predicting 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))

        # predicting 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))

        #appending for plotting
        baselearner.append(i)
        depth.append(j)
```

Plotting the heat map on the train data

In [99]:

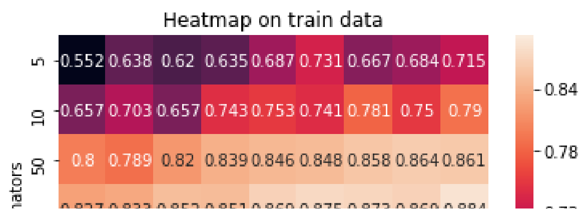
```
#plotting the heatmap on train data
#https://cmdlinetips.com/2019/01/how-to-make-heatmap-with-seaborn-in-python/
data = pd.DataFrame({'n_estimators': baselearner, 'max_depth': depth, 'AUC': roc_tr})
scores = data.pivot("n_estimators", "max_depth", "AUC")
ax = sns.heatmap(scores,annot=True,fmt='.3g')
plt.title('Heatmap on train data')
plt.show()
```

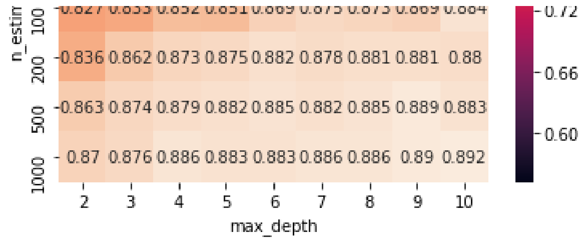


Plotting the heatmap on cv data

In [94]:

```
#plotting the heatmap on cv data
#https://cmdlinetips.com/2019/01/how-to-make-heatmap-with-seaborn-in-python/
data = pd.DataFrame({'n_estimators': baselearner, 'max_depth': depth, 'AUC': roc_cv})
scores = data.pivot("n_estimators", "max_depth", "AUC")
plot = sns.heatmap(scores,annot=True,fmt='.3g')
plt.title('Heatmap on train data')
plt.show()
```

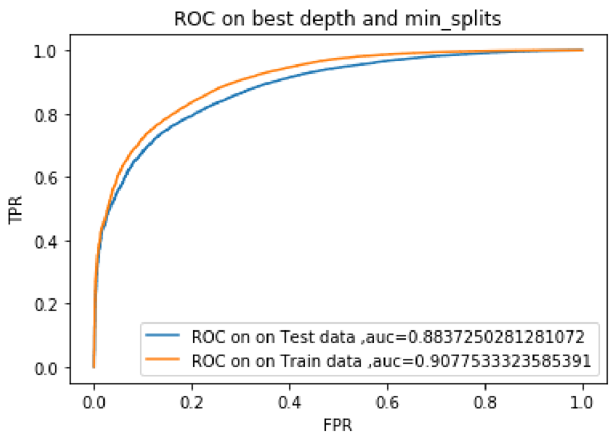




Training the model with obtained best depths and best number of estimators

In [95]:

```
#1)Training the model with obtained best depths and best number of estimators
clf=RandomForestClassifier(max_features='sqrt',max_depth=optimal_depth_rfbow,n_estimators=optimal_b
aselearners_rfbow)
# fitting the model on train data
clf.fit(final_X_tr,y_tr)
#predicting probablity of success Training data
pred_tr = clf.predict_proba(final_X_tr)
pred_tr=(pred_tr)[:,1]
#predicting probability of success on Test data
pred_test = clf.predict_proba(final_X_test)
pred_test=(pred_test)[:,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_test,tpr_test ,label='ROC on on Test data
,auc='+str(roc_auc_score(y_test,pred_test)))
plt.plot(fpr_tr,tpr_tr ,label='ROC on on Train data ,auc='+str(roc_auc_score(y_tr,pred_tr)))
plt.legend()
plt.title('ROC on best depth and min_splits')
plt.xlabel('FPR')
plt.ylabel('TPR')
plt.show()
```

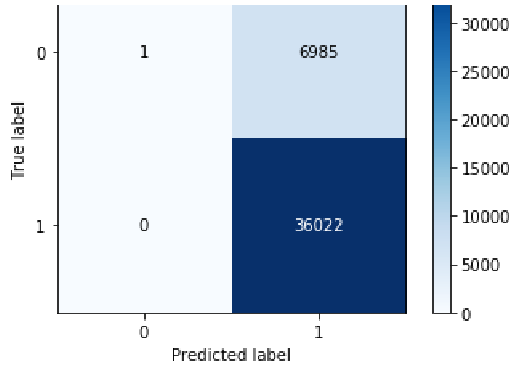


In [96]:

```
#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[96]:

```
<matplotlib.axes._subplots.AxesSubplot at 0x7f46b7413940>
```

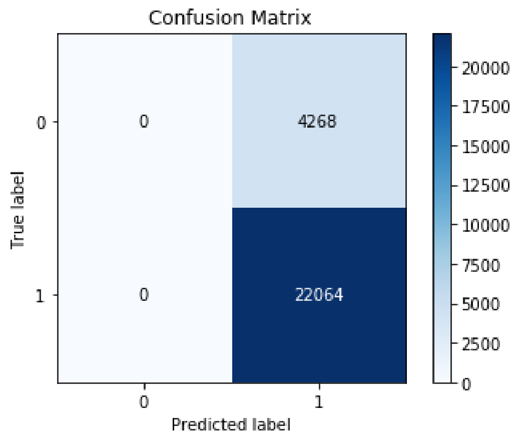


In [97]:

```
#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[97]:

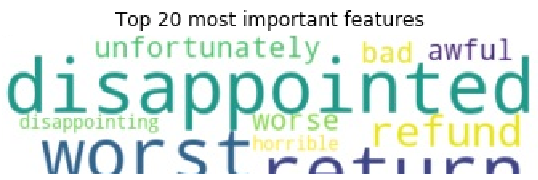
```
<matplotlib.axes._subplots.AxesSubplot at 0x7f46bd873b00>
```



[5.1.2] Wordcloud of top 20 important features from SET 1

In [98]:

```
#https://www.datacamp.com/community/tutorials/wordcloud-python
#getting feature_names and values to get top 20 features
feature_names = count_vect.get_feature_names()
feature_values = clf.feature_importances_
feature_values_df = pd.DataFrame({'word': feature_names, 'coeficient': feature_values})
dframe = feature_values_df.sort_values("coeficient", ascending = False)[:20]
text = " ".join(word for word in dframe.word)

#generating a wordcloud image
wordcloud = WordCloud(background_color ='white').generate(text)

# plotting the wordcloud
plt.imshow(wordcloud, interpolation = 'bilinear')
plt.axis("off")
plt.title("Top 20 most important features")
plt.show()
```



[5.1.3] Applying Random Forests on TFIDF, SET 2

In [100]:

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

## 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=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 [101]:

```
#Finding the optimal number of base learners and depths
base_learners = [5,10,50,100,200,500,1000]
depths= [2,3,4,5,6,7,8,9,10]
tuned_parameters={'n_estimators': base_learners, 'max_depth':depths}
clf= RandomForestClassifier(max_features='sqrt')
rf_randomsearch=RandomizedSearchCV(clf,tuned_parameters,scoring='roc_auc',n_jobs=-1)
rf_randomsearch.fit(final_X_tr,y_tr)
optimal_baselearners_rftfidf=rf_randomsearch.best_estimator_.n_estimators
optimal_depth_rftfidf=rf_randomsearch.best_estimator_.max_depth
optimal_aucscore_rftfidf=rf_randomsearch.best_score_
print("optimal number of baselearners",optimal_baselearners_rftfidf)
print("optimal max_depth",optimal_depth_rftfidf)
print("Best AUCScore:", optimal_aucscore_rftfidf)
```

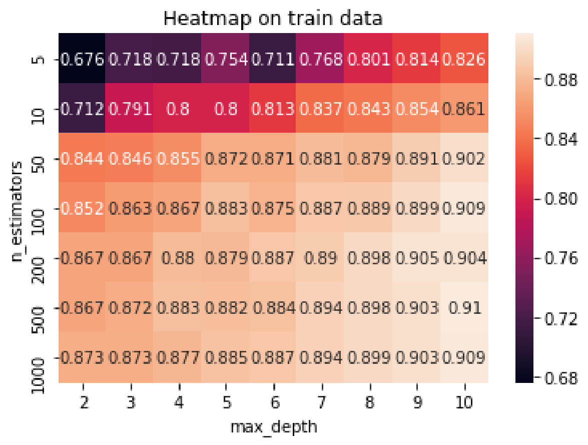
```
optimal number of baselearners 500
optimal max_depth 8
Best AUCScore: 0.8735470650427336
```

In [102]:

```
#finding the auc scores on train and cv data
roc_tr=[]
roc_cv=[]
baselearner=[]
depth=[]
for i in base_learners:
    for j in depths:
        clf=RandomForestClassifier(max_features='sqrt',max_depth=j,n_estimators=i)

        #fitting the model
        clf.fit(final_X_tr,y_tr)

        # predicting 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))

        # predicting 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))

        #appending for plotting
        baselearner.append(i)
        depth.append(j)
```

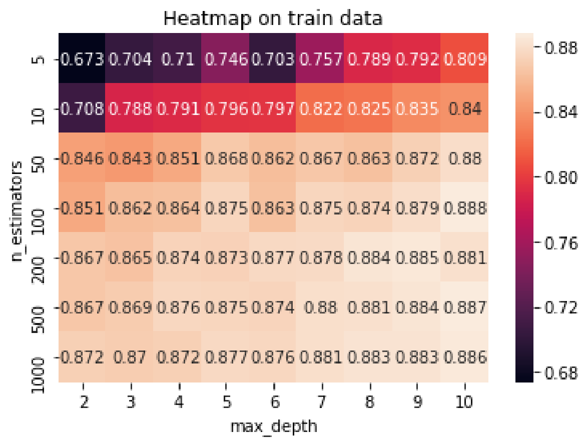
In [103]:

```
#plotting the heatmap on train data
#https://cmdlinetips.com/2019/01/how-to-make-heatmap-with-seaborn-in-python/
data = pd.DataFrame({'n_estimators': baselearner, 'max_depth': depth, 'AUC': roc_tr})
scores = data.pivot("n_estimators", "max_depth", "AUC")
plot = sns.heatmap(scores,annot=True,fmt='.3g')
plt.title('Heatmap on train data')
plt.show()
```



In [ ]:

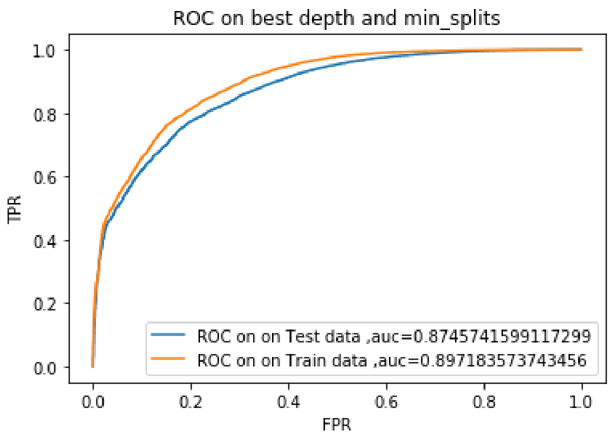
```
#plotting the heatmap on cv data
#https://cmdlinetips.com/2019/01/how-to-make-heatmap-with-seaborn-in-python/
data = pd.DataFrame({'n_estimators': baselearner, 'max_depth': depth, 'AUC': roc_cv})
scores = data.pivot("n_estimators", "max_depth", "AUC")
plot = sns.heatmap(scores,annot=True,fmt='.3g')
plt.title('Heatmap on train data')
plt.show()
```



In [ ]:

```
#1)Training the model with obtained best depths and best number of estimators
clf=RandomForestClassifier(max_features='sqrt',max_depth=optimal_depth_rftfidf,n_estimators=optimal
_baselearners_rftfidf)
# fitting the model on train data
clf.fit(final_X_tr,y_tr)
#predicting probablity of success Training data
pred_tr = clf.predict_proba(final_X_tr)
pred_tr=(pred_tr)[:,1]
#predicting probability of success on Test data
pred_test = clf.predict_proba(final_X_test)
pred_test=(pred_test)[:,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_test,tpr_test ,label='ROC on on Test data
,auc='+str(roc_auc_score(y_test,pred_test)))
plt.plot(fpr_tr,tpr_tr ,label='ROC on on Train data ,auc='+str(roc_auc_score(y_tr,pred_tr)))
plt.legend()
plt.title('ROC on best depth and min_splits')
plt.xlabel('FPR')
plt.ylabel('TPR')
plt.show()
```

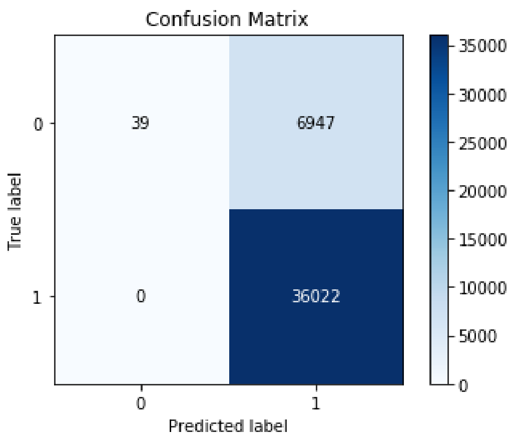


In [ ]:

```
#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[ ]:

```
<matplotlib.axes._subplots.AxesSubplot at 0x7f46bde33fd0>
```

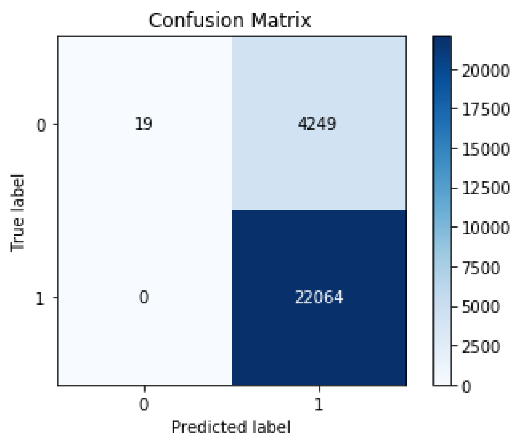


In [ ]:

```
#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[ ]:

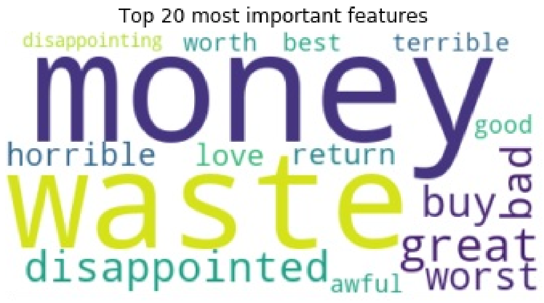
```
<matplotlib.axes._subplots.AxesSubplot at 0x7f46ef998c18>
```



[5.1.4] Wordcloud of top 20 important features from SET 2

In [51]:

```
#https://www.datacamp.com/community/tutorials/wordcloud-python
#getting feature_names and values to get top 20 features
feature_names = tf_idf_vect.get_feature_names()
feature_values = clf.feature_importances_
feature_values_df = pd.DataFrame({'word': feature_names, 'coeficient': feature_values})
dframe = feature_values_df.sort_values("coeficient", ascending = False)[:20]
text = " ".join(word for word in dframe.word)

#generating a wordcloud image
wordcloud = WordCloud(background_color ='white').generate(text)

# plotting the wordcloud
plt.imshow(wordcloud, interpolation = 'bilinear')
plt.axis("off")
plt.title("Top 20 most important features")
plt.show()
```



[5.1.5] Applying Random Forests on AVG W2V, SET 3

In [ ]:

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

## 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=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%|██████████| 43008/43008 [02:10<00:00, 330.47it/s]
100%|██████████| 18433/18433 [00:56<00:00, 328.67it/s]
100%|██████████| 26332/26332 [01:21<00:00, 324.53it/s]
```

In [ ]:

```
#Finding the optimal number of base learners and depths
base_learners = [5,10,50,100,200,500,1000]
depths= [2,3,4,5,6,7,8,9,10]
tuned_parameters={'n_estimators': base_learners, 'max_depth':depths}
clf= RandomForestClassifier(max_features='sqrt')
rf_randomsearch=RandomizedSearchCV(clf,tuned_parameters,scoring='roc_auc',n_jobs=-1)
rf_randomsearch.fit(final_X_tr,y_tr)
optimal_baselearners_rfavgw2v=rf_randomsearch.best_estimator_.n_estimators
optimal_depth_rfavgw2v=rf_randomsearch.best_estimator_.max_depth
optimal_aucscore_rfavgw2v=rf_randomsearch.best_score_
print("optimal number of baselearners",optimal_baselearners_rfavgw2v)
print("optimal max_depth",optimal_depth_rfavgw2v)
print("Best AUCScore:", optimal_aucscore_rfavgw2v)
```

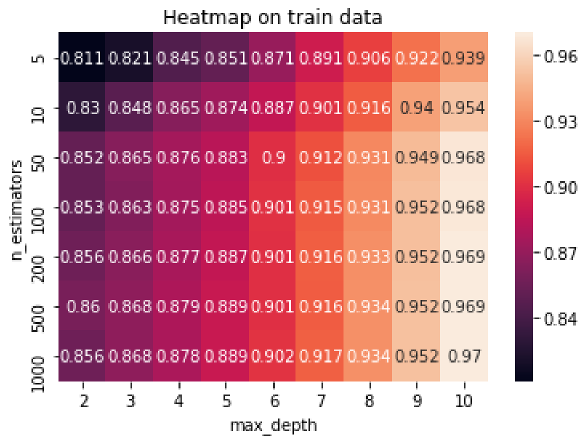
```
optimal number of baselearners 500
optimal max_depth 10
Best AUCScore: 0.8949304138149061
```

In [ ]:

```python
#finding the auc scores on train and cv data
roc_tr=[]
roc_cv=[]
baselearner=[]
depth=[]
for i in base_learners:
    for j in depths:
        clf=RandomForestClassifier(max_features='sqrt',max_depth=j,n_estimators=i)

        #fitting the model
        clf.fit(final_X_tr,y_tr)

        # predicting 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))

        # predicting 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))

        #appending for plotting
        baselearner.append(i)
        depth.append(j)
```

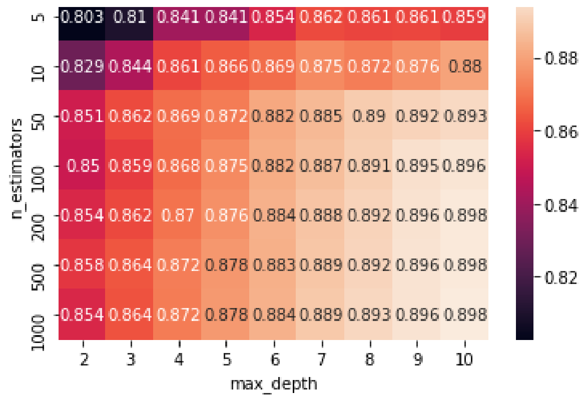
In [ ]:

```python
#plotting the heatmap on train data
#https://cmdlinetips.com/2019/01/how-to-make-heatmap-with-seaborn-in-python/
data = pd.DataFrame({'n_estimators': baselearner, 'max_depth': depth, 'AUC': roc_tr})
scores = data.pivot("n_estimators", "max_depth", "AUC")
plot = sns.heatmap(scores,annot=True,fmt='.3g')
plt.title('Heatmap on train data')
plt.show()
```



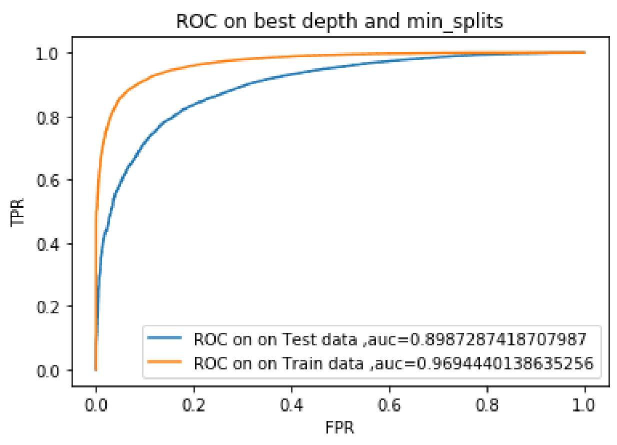
In [ ]:

```python
#plotting the heatmap on cv data
#https://cmdlinetips.com/2019/01/how-to-make-heatmap-with-seaborn-in-python/
data = pd.DataFrame({'n_estimators': baselearner, 'max_depth': depth, 'AUC': roc_cv})
scores = data.pivot("n_estimators", "max_depth", "AUC")
plot = sns.heatmap(scores,annot=True,fmt='.3g')
plt.title('Heatmap on train data')
plt.show()
```



In [ ]:

```
#1)Training the model with obtained best depths and best number of estimators
clf=RandomForestClassifier(max_features='sqrt',max_depth=optimal_depth_rfavgw2v,n_estimators=optim
al_baselearners_rfavgw2v)
# fitting the model on train data
clf.fit(final_X_tr,y_tr)
#predicting probablity of success Training data
pred_tr = clf.predict_proba(final_X_tr)
pred_tr=(pred_tr)[:,1]
#predicting probability of success on Test data
pred_test = clf.predict_proba(final_X_test)
pred_test=(pred_test)[:,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_test,tpr_test ,label='ROC on on Test data
,auc='+str(roc_auc_score(y_test,pred_test)))
plt.plot(fpr_tr,tpr_tr ,label='ROC on on Train data ,auc='+str(roc_auc_score(y_tr,pred_tr)))
plt.legend()
plt.title('ROC on best depth and min_splits')
plt.xlabel('FPR')
plt.ylabel('TPR')
plt.show()
```

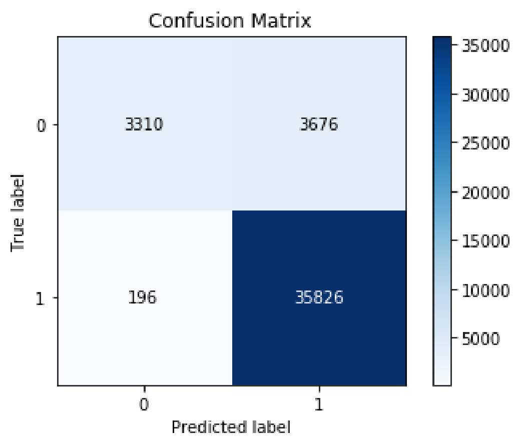


In [ ]:

```
#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[ ]:

```
<matplotlib.axes._subplots.AxesSubplot at 0x7f469cfba550>
```

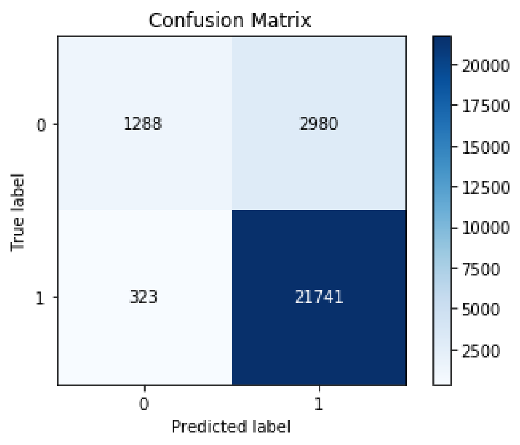


In [ ]:

```
#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[ ]:

```
<matplotlib.axes._subplots.AxesSubplot at 0x7f469db9f240>
```



[5.1.6] Applying Random Forests on TFIDF W2V, SET 4

In [37]:

```
# 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 [04:04<00:00, 176.02it/s]
100%|██████████| 18433/18433 [01:43<00:00, 177.57it/s]
100%|██████████| 26332/26332 [02:30<00:00, 174.89it/s]
```

In [38]:

```
#Finding the optimal number of base learners and depths
base_learners = [5,10,50,100,200,500,1000]
depths= [2,3,4,5,6,7,8,9,10]
tuned_parameters={'n_estimators': base_learners, 'max_depth':depths}
clf= RandomForestClassifier(max_features='sqrt')
rf_randomsearch=RandomizedSearchCV(clf,tuned_parameters,scoring='roc_auc',n_jobs=-1)
rf_randomsearch.fit(final_X_tr,y_tr)
optimal_baselearners_rftfidfw2v=rf_randomsearch.best_estimator_.n_estimators
optimal_depth_rftfidfw2v=rf_randomsearch.best_estimator_.max_depth
optimal_aucscore_rftfidfw2v=rf_randomsearch.best_score_
print("optimal number of baselearners",optimal_baselearners_rftfidfw2v)
print("optimal max_depth",optimal_depth_rftfidfw2v)
print("Best AUCScore:", optimal_aucscore_rftfidfw2v)
```

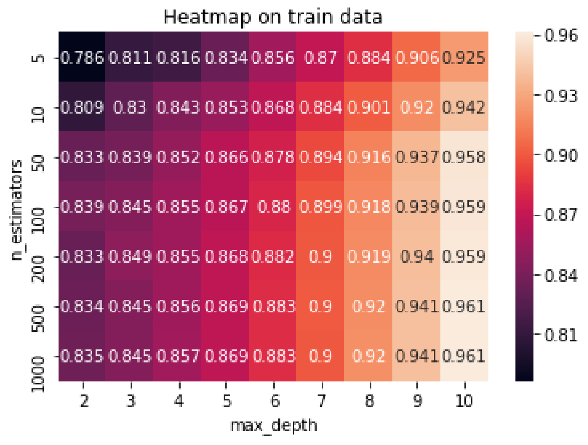
```
optimal number of baselearners 500
optimal number of baselearners 500
optimal max_depth 10
Best AUCScore: 0.8728810655022399
```

In [39]:

```
#finding the auc scores on train and cv data
roc_tr=[]
roc_cv=[]
baselearner=[]
depth=[]
for i in base_learners:
    for j in depths:
        clf=RandomForestClassifier(max_features='sqrt',max_depth=j,n_estimators=i)

        #fitting the model
        clf.fit(final_X_tr,y_tr)

        # predicting 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))

        # predicting 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))

        #appending for plotting
        baselearner.append(i)
        depth.append(j)
```

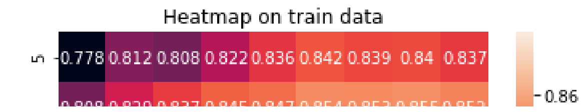
In [40]:

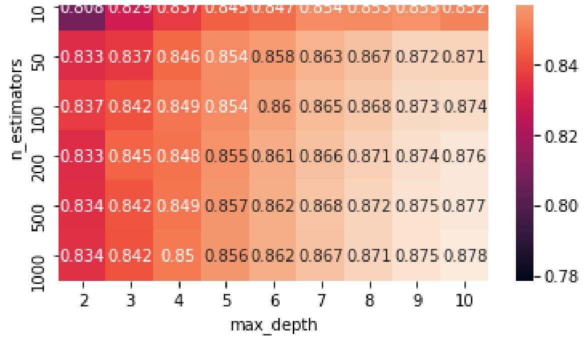
```
#plotting the heatmap on train data
#https://cmdlinetips.com/2019/01/how-to-make-heatmap-with-seaborn-in-python/
data = pd.DataFrame({'n_estimators': baselearner, 'max_depth': depth, 'AUC': roc_tr})
scores = data.pivot("n_estimators", "max_depth", "AUC")
plot = sns.heatmap(scores,annot=True,fmt='.3g')
plt.title('Heatmap on train data')
plt.show()
```



In [41]:

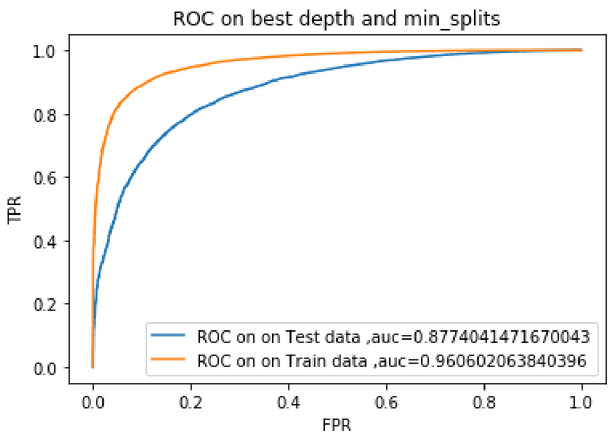
```
#plotting the heatmap on cv data
#https://cmdlinetips.com/2019/01/how-to-make-heatmap-with-seaborn-in-python/
data = pd.DataFrame({'n_estimators': baselearner, 'max_depth': depth, 'AUC': roc_cv})
scores = data.pivot("n_estimators", "max_depth", "AUC")
plot = sns.heatmap(scores,annot=True,fmt='.3g')
plt.title('Heatmap on train data')
plt.show()
```





In [42]:

```
#1)Training the model with obtained best depths and best number of estimators
clf=RandomForestClassifier(max_features='sqrt',max_depth=optimal_depth_rftfidfw2v,n_estimators=opt
imal_baselearners_rftfidfw2v)
# fitting the model on train data
clf.fit(final_X_tr,y_tr)
#predicting probablity of success Training data
pred_tr = clf.predict_proba(final_X_tr)
pred_tr=(pred_tr)[:,1]
#predicting probability of success on Test data
pred_test = clf.predict_proba(final_X_test)
pred_test=(pred_test)[:,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_test,tpr_test ,label='ROC on on Test data
,auc='+str(roc_auc_score(y_test,pred_test)))
plt.plot(fpr_tr,tpr_tr ,label='ROC on on Train data ,auc='+str(roc_auc_score(y_tr,pred_tr)))
plt.legend()
plt.title('ROC on best depth and min_splits')
plt.xlabel('FPR')
plt.ylabel('TPR')
plt.show()
```

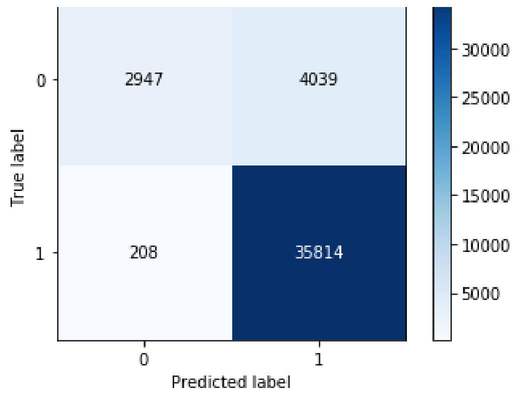


In [43]:

```
#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[43]:

```
<matplotlib.axes._subplots.AxesSubplot at 0x7f4f7de625c0>
```

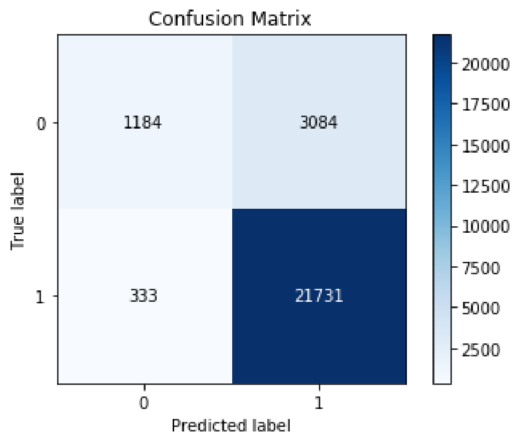


In [44]:

```
#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[44]:

```
<matplotlib.axes._subplots.AxesSubplot at 0x7f4f71658a90>
```



[5.2] Applying GBDT using XGBOOST

[5.2.1] Applying XGBOOST on BOW, SET 1

In [38]:

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

## 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 = 2000,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 [37]:

```
#Finding the optimal number of base learners and depths
```

```
#finding the optimal number of base learners and depths
base_learners = [5,10,50,100,200,500,1000]
depths= [2,3,4,5,6,7,8,9,10]
tuned_parameters={'n_estimators': base_learners, 'max_depth':depths}
clf= XGBClassifier()
xgb_randomsearch=RandomizedSearchCV(clf,tuned_parameters,scoring='roc_auc',n_jobs=-1)
xgb_randomsearch.fit(final_X_tr,y_tr)
optimal_baselearners_xgbtbow=xgb_randomsearch.best_estimator_.n_estimators
optimal_depth_xgbtbow=xgb_randomsearch.best_estimator_.max_depth
optimal_aucscore_xgbtbow=xgb_randomsearch.best_score_
print("optimal number of baselearners",optimal_baselearners_xgbtbow)
print("optimal max_depth",optimal_depth_xgbtbow)
print("Best AUCScore:", optimal_aucscore_xgbtbow)
```

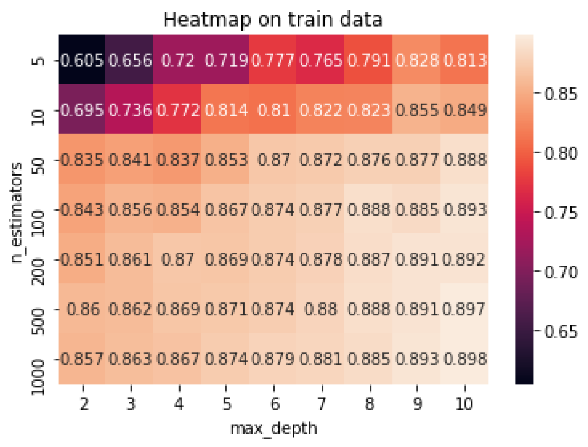
```
optimal number of baselearners 500
optimal max_depth 10
Best AUCScore: 0.9322985286626816
```

In [38]:

```
#finding the auc scores on train and cv data
roc_tr=[]
roc_cv=[]
baselearner=[]
depth=[]
for i in base_learners:
    for j in depths:
        clf=RandomForestClassifier(max_depth=j,n_estimators=i)

        #fitting the model
        clf.fit(final_X_tr,y_tr)

        # predicting 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))

        # predicting 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))

        #appending for plotting
        baselearner.append(i)
        depth.append(j)
```

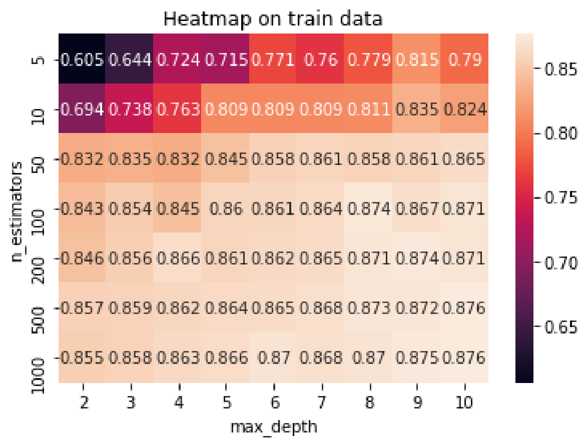
In [39]:

```
#plotting the heatmap on train data
#https://cmdlinetips.com/2019/01/how-to-make-heatmap-with-seaborn-in-python/
data = pd.DataFrame({'n_estimators': baselearner, 'max_depth': depth, 'AUC': roc_tr})
scores = data.pivot("n_estimators", "max_depth", "AUC")
plot = sns.heatmap(scores,annot=True,fmt='.3g')
plt.title('Heatmap on train data')
plt.show()
```



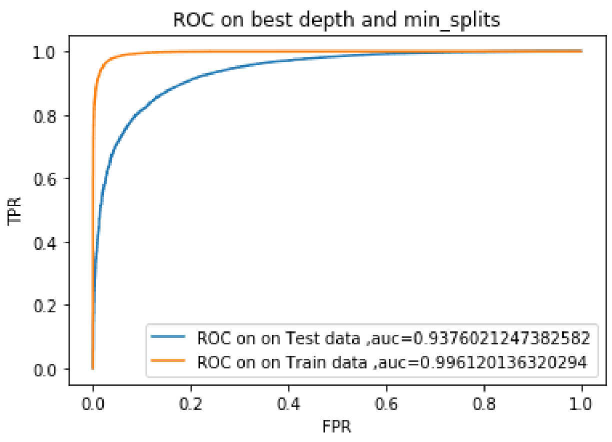
In [40]:

```
#plotting the heatmap on cv data
#https://cmdlinetips.com/2019/01/how-to-make-heatmap-with-seaborn-in-python/
data = pd.DataFrame({'n_estimators': baselearner, 'max_depth': depth, 'AUC': roc_cv})
scores = data.pivot("n_estimators", "max_depth", "AUC")
plot = sns.heatmap(scores,annot=True,fmt='.3g')
plt.title('Heatmap on train data')
plt.show()
```



In [41]:

```
#1)Training the model using best min_splits and best depth
clf = XGBClassifier(max_depth = optimal_depth_xgbtbow, n_estimators = optimal_baselearners_xgbtbow)
# fitting the model on train data
clf.fit(final_X_tr,y_tr)
#predicting probablity of success Training data
pred_tr = clf.predict_proba(final_X_tr)
pred_tr=(pred_tr)[:,1]
#predicting probablity of success on Test data
pred_test = clf.predict_proba(final_X_test)
pred_test=(pred_test)[:,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_test,tpr_test ,label='ROC on on Test data
,auc='+str(roc_auc_score(y_test,pred_test)))
plt.plot(fpr_tr,tpr_tr ,label='ROC on on Train data ,auc='+str(roc_auc_score(y_tr,pred_tr)))
plt.legend()
plt.title('ROC on best depth and min_splits')
plt.xlabel('FPR')
plt.ylabel('TPR')
plt.show()
```

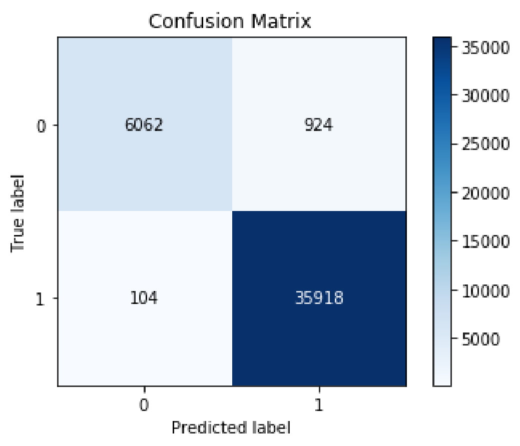


In [42]:

```
#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[42]:

```
<matplotlib.axes._subplots.AxesSubplot at 0x7ff8816962b0>
```

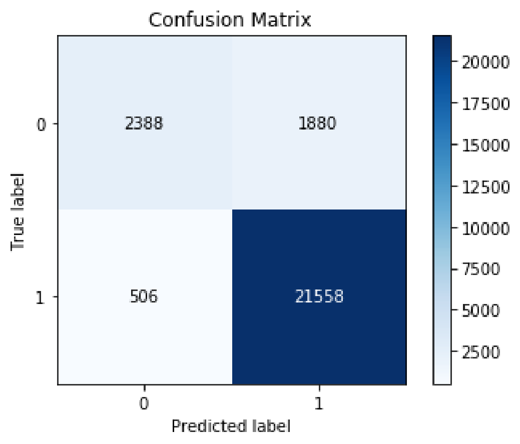


In [43]:

```
#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[43]:

```
<matplotlib.axes._subplots.AxesSubplot at 0x7ff87ae04cf8>
```



[5.2.2] Applying XGBOOST on TFIDF, SET 2

In [44]:

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

## 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=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 [45]:

```
#Finding the optimal number of base learners and depths
base_learners = [5,10,50,100,200,500,1000]
depths= [2,3,4,5,6,7,8,9,10]
tuned_parameters={'n_estimators': base_learners, 'max_depth':depths}
clf= XGBClassifier()
xgb_randomsearch=RandomizedSearchCV(clf,tuned_parameters,scoring='roc_auc',n_jobs=-1)
xgb_randomsearch.fit(final_X_tr,y_tr)
optimal_baselearners_xgbttfidf=xgb_randomsearch.best_estimator_.n_estimators
optimal_depth_xgbttfidf=xgb_randomsearch.best_estimator_.max_depth
optimal_aucscore_xgbttfidf=xgb_randomsearch.best_score_
print("optimal number of baselearners",optimal_baselearners_xgbttfidf)
print("optimal max_depth",optimal_depth_xgbttfidf)
print("Best AUCScore:", optimal_aucscore_xgbttfidf)
```

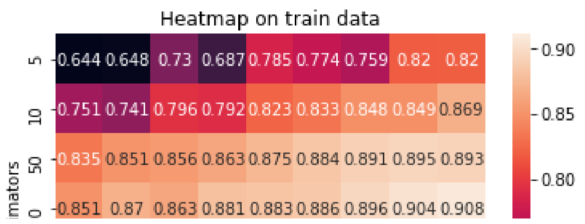
```
optimal number of baselearners 1000
optimal max_depth 5
Best AUCScore: 0.9402924514116242
```

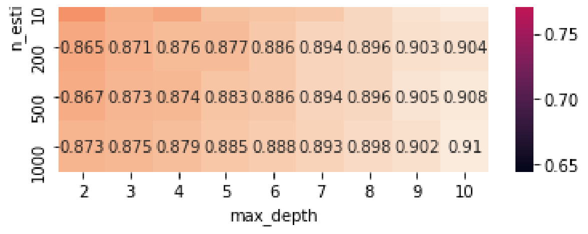
In [46]:

```
#finding the auc scores on train and cv data
roc_tr=[]
roc_cv=[]
baselearner=[]
depth=[]
for i in base_learners:
    for j in depths:
        clf=RandomForestClassifier(max_depth=j,n_estimators=i)

        #fitting the model
        clf.fit(final_X_tr,y_tr)

        # predicting 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))

        # predicting 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))

        #appending for plotting
        baselearner.append(i)
        depth.append(j)
```

In [47]:

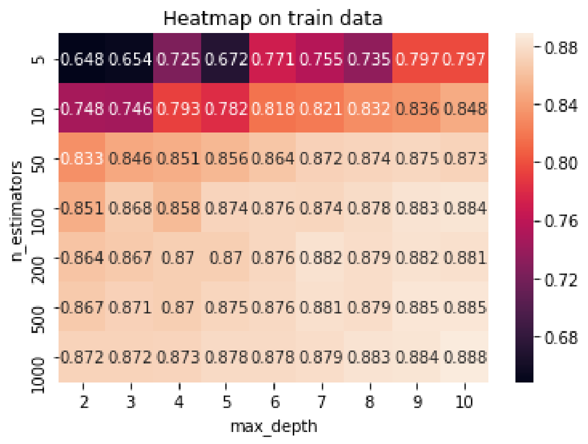
```
#plotting the heatmap on train data
#https://cmdlinetips.com/2019/01/how-to-make-heatmap-with-seaborn-in-python/
data = pd.DataFrame({'n_estimators': baselearner, 'max_depth': depth, 'AUC': roc_tr})
scores = data.pivot("n_estimators", "max_depth", "AUC")
plot= sns.heatmap(scores,annot=True,fmt='.3g')
plt.title('Heatmap on train data')
plt.show()
```





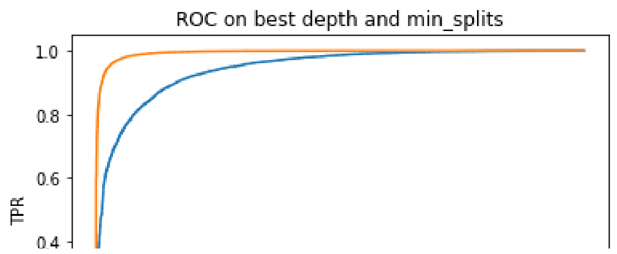
In [48]:

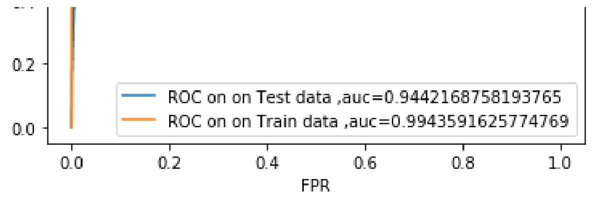
```
#plotting the heatmap on cv data
#https://cmdlinetips.com/2019/01/how-to-make-heatmap-with-seaborn-in-python/
data = pd.DataFrame({'n_estimators': baselearner, 'max_depth': depth, 'AUC': roc_cv})
scores = data.pivot("n_estimators", "max_depth", "AUC")
plot = sns.heatmap(scores,annot=True,fmt='.3g')
plt.title('Heatmap on train data')
plt.show()
```



In [49]:

```
#1)Training the model using best min_splits and best depth
clf = XGBClassifier(max_depth = optimal_depth_xgbttfidf, n_estimators =
optimal_baselearners_xgbttfidf)
# fitting the model on train data
clf.fit(final_X_tr,y_tr)
#predicting probablity of success Training data
pred_tr = clf.predict_proba(final_X_tr)
pred_tr=(pred_tr)[:,1]
#predicting probability of success on Test data
pred_test = clf.predict_proba(final_X_test)
pred_test=(pred_test)[:,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_test,tpr_test ,label='ROC on on Test data
,auc='+str(roc_auc_score(y_test,pred_test)))
plt.plot(fpr_tr,tpr_tr ,label='ROC on on Train data ,auc='+str(roc_auc_score(y_tr,pred_tr)))
plt.legend()
plt.title('ROC on best depth and min_splits')
plt.xlabel('FPR')
plt.ylabel('TPR')
plt.show()
```



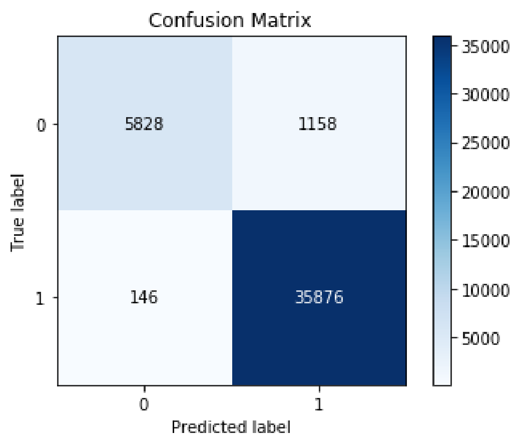


In [50]:

```
#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[50]:

```
<matplotlib.axes._subplots.AxesSubplot at 0x7ff8730e5d30>
```

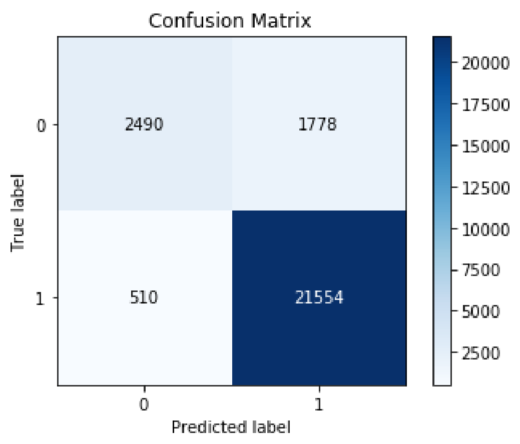


In [51]:

```
#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[51]:

```
<matplotlib.axes._subplots.AxesSubplot at 0x7ff878119ac8>
```



[5.2.3] Applying XGBOOST on AVG W2V, SET 3

In [52]:

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

## 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=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%|██████████| 43008/43008 [01:52<00:00, 382.24it/s]
100%|██████████| 18433/18433 [00:47<00:00, 390.23it/s]
100%|██████████| 26332/26332 [01:08<00:00, 381.94it/s]
```

In [53]:

```
#converting to numpy array
final_X_tr=np.array(final_X_tr)
final_X_cv=np.array(final_X_cv)
final_X_test=np.array(final_X_test)

#Finding the optimal number of base learners and depths
base_learners = [5,10,50,100,200,500,1000]
depths= [2,3,4,5,6,7,8,9,10]
tuned_parameters={'n_estimators': base_learners, 'max_depth':depths}
clf= XGBClassifier()
xgb_randomsearch=RandomizedSearchCV(clf,tuned_parameters,scoring='roc_auc',n_jobs=-1)
xgb_randomsearch.fit(final_X_tr,y_tr)
optimal_baselearners_xgbtavgw2v=xgb_randomsearch.best_estimator_.n_estimators
optimal_depth_xgbtavgw2v=xgb_randomsearch.best_estimator_.max_depth
optimal_aucscore_xgbtavgw2v=xgb_randomsearch.best_score_
print("optimal number of baselearners",optimal_baselearners_xgbtavgw2v)
print("optimal max_depth",optimal_depth_xgbtavgw2v)
print("Best AUCScore:", optimal_aucscore_xgbtavgw2v)
```

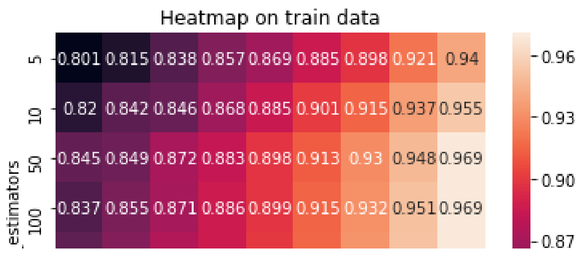
```
optimal number of baselearners 500
optimal max_depth 7
Best AUCScore: 0.9105260795281419
```

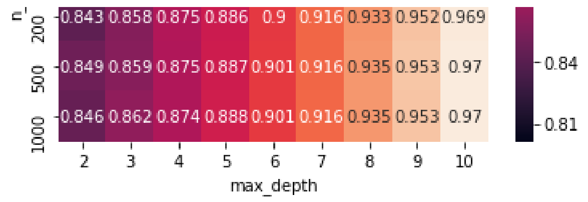
In [54]:

```
#finding the auc scores on train and cv data
roc_tr=[]
roc_cv=[]
baselearner=[]
depth=[]
for i in base_learners:
    for j in depths:
        clf=RandomForestClassifier(max_depth=j,n_estimators=i)

        #fitting the model
        clf.fit(final_X_tr,y_tr)

        # predicting 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))

        # predicting 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))

        #appending for plotting
        baselearner.append(i)
        depth.append(j)
```

In [55]:

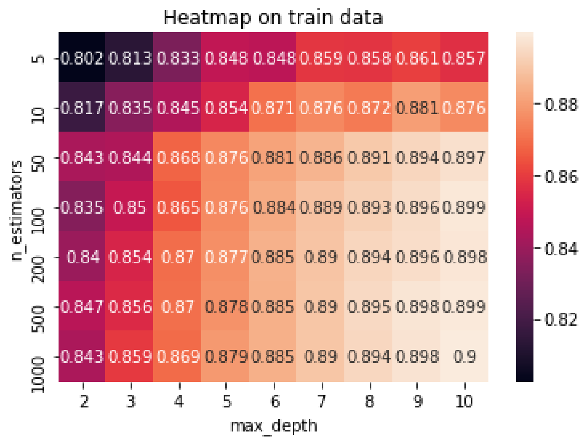
```
#plotting the heatmap on train data
#https://cmdlinetips.com/2019/01/how-to-make-heatmap-with-seaborn-in-python/
data = pd.DataFrame({'n_estimators': baselearner, 'max_depth': depth, 'AUC': roc_tr})
scores = data.pivot("n_estimators", "max_depth", "AUC")
plot = sns.heatmap(scores,annot=True,fmt='.3g')
plt.title('Heatmap on train data')
plt.show()
```





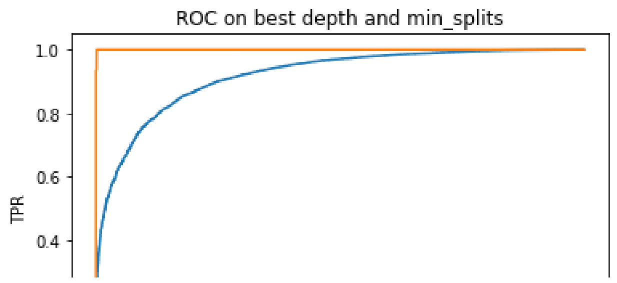
In [56]:

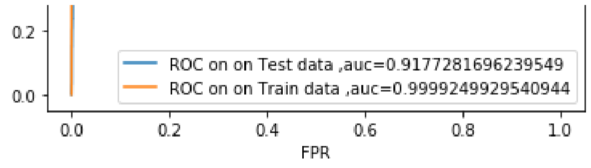
```
#plotting the heatmap on cv data
#https://cmdlinetips.com/2019/01/how-to-make-heatmap-with-seaborn-in-python/
data = pd.DataFrame({'n_estimators': baselearner, 'max_depth': depth, 'AUC': roc_cv})
scores = data.pivot("n_estimators", "max_depth", "AUC")
plot = sns.heatmap(scores,annot=True,fmt='.3g')
plt.title('Heatmap on train data')
plt.show()
```



In [57]:

```
#1)Training the model using best min_splits and best depth
clf = XGBClassifier(max_depth = optimal_depth_xgbtavgw2v, n_estimators =
optimal_baselearners_xgbtavgw2v)
# fitting the model on train data
clf.fit(final_X_tr,y_tr)
#predicting probablity of success Training data
pred_tr = clf.predict_proba(final_X_tr)
pred_tr=(pred_tr)[:,1]
#predicting probability of success on Test data
pred_test = clf.predict_proba(final_X_test)
pred_test=(pred_test)[:,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_test,tpr_test ,label='ROC on on Test data
,auc='+str(roc_auc_score(y_test,pred_test)))
plt.plot(fpr_tr,tpr_tr ,label='ROC on on Train data ,auc='+str(roc_auc_score(y_tr,pred_tr)))
plt.legend()
plt.title('ROC on best depth and min_splits')
plt.xlabel('FPR')
plt.ylabel('TPR')
plt.show()
```



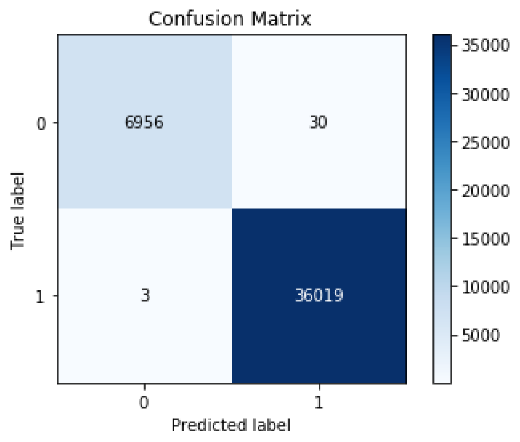


In [58]:

```
#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[58]:

```
<matplotlib.axes._subplots.AxesSubplot at 0x7ff879893780>
```

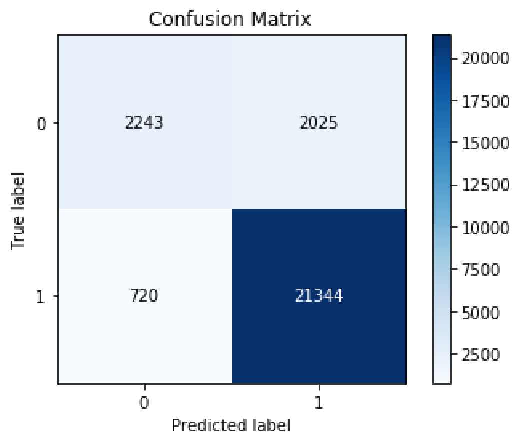


In [59]:

```
#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[59]:

```
<matplotlib.axes._subplots.AxesSubplot at 0x7ff881496e10>
```



[5.2.4] Applying XGBOOST on TFIDF W2V, SET 4

In [39]:

```python
# 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 [04:35<00:00, 156.37it/s]
```

```
100%|██████████| 18433/18433 [01:55<00:00, 159.80it/s]
100%|██████████| 26332/26332 [02:45<00:00, 159.20it/s]
```

In [40]:

```
#converting to numpy array
final_X_tr=np.array(final_X_tr)
final_X_cv=np.array(final_X_cv)
final_X_test=np.array(final_X_test)

#Finding the optimal number of base learners and depths
base_learners = [5,10,50,100,200,500,1000]
depths= [2,3,4,5,6,7,8,9,10]
tuned_parameters={'n_estimators': base_learners, 'max_depth':depths}
clf= XGBClassifier()
xgb_randomsearch=RandomizedSearchCV(clf,tuned_parameters,scoring='roc_auc',n_jobs=-1)
xgb_randomsearch.fit(final_X_tr,y_tr)
optimal_baselearners_xgbttfidfw2v=xgb_randomsearch.best_estimator_.n_estimators
optimal_depth_xgbttfidfw2v=xgb_randomsearch.best_estimator_.max_depth
optimal_aucscore_xgbttfidfw2v=xgb_randomsearch.best_score_
print("optimal number of baselearners",optimal_baselearners_xgbttfidfw2v)
print("optimal max_depth",optimal_depth_xgbttfidfw2v)
print("Best AUCScore:", optimal_aucscore_xgbttfidfw2v)
```

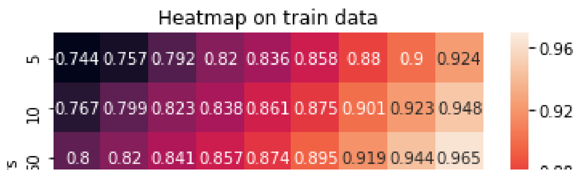
```
optimal number of baselearners 1000
optimal max_depth 2
Best AUCScore: 0.8895269699694236
```

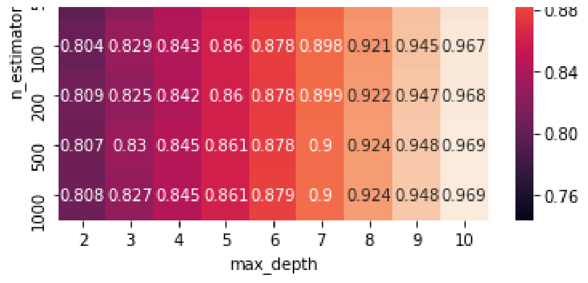
In [41]:

```
#finding the auc scores on train and cv data
roc_tr=[]
roc_cv=[]
baselearner=[]
depth=[]
for i in base_learners:
    for j in depths:
        clf=RandomForestClassifier(max_depth=j,n_estimators=i)

        #fitting the model
        clf.fit(final_X_tr,y_tr)

        # predicting 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))

        # predicting 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))

        #appending for plotting
        baselearner.append(i)
        depth.append(j)
```

In [42]:

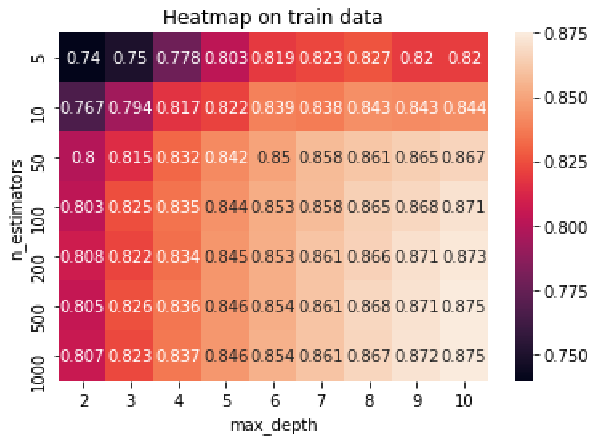
```
#plotting the heatmap on train data
#https://cmdlinetips.com/2019/01/how-to-make-heatmap-with-seaborn-in-python/
data = pd.DataFrame({'n_estimators': baselearner, 'max_depth': depth, 'AUC': roc_tr})
scores = data.pivot("n_estimators", "max_depth", "AUC")
plot = sns.heatmap(scores,annot=True,fmt='.3g')
plt.title('Heatmap on train data')
plt.show()
```





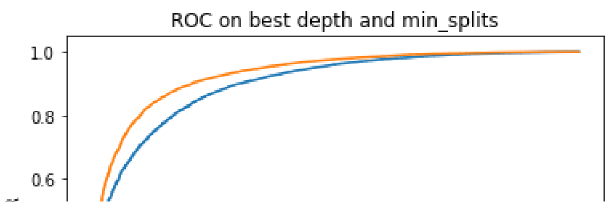
In [43]:

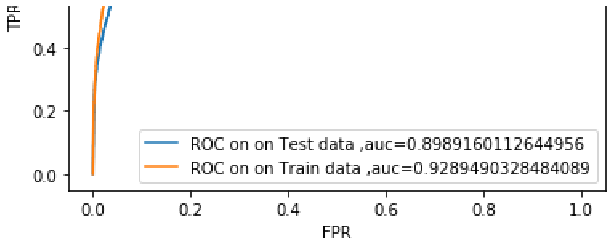
```
#plotting the heatmap on train data
#https://cmdlinetips.com/2019/01/how-to-make-heatmap-with-seaborn-in-python/
data = pd.DataFrame({'n_estimators': baselearner, 'max_depth': depth, 'AUC': roc_cv})
scores = data.pivot("n_estimators", "max_depth", "AUC")
plot = sns.heatmap(scores,annot=True,fmt='.3g')
plt.title('Heatmap on train data')
plt.show()
```



In [44]:

```
#1)Training the model using best min_splits and best depth
clf = XGBClassifier(max_depth = optimal_depth_xgbttfidfw2v, n_estimators =
optimal_baselearners_xgbttfidfw2v)
# fitting the model on train data
clf.fit(final_X_tr,y_tr)
#predicting probablity of success Training data
pred_tr = clf.predict_proba(final_X_tr)
pred_tr=(pred_tr)[:,1]
#predicting probability of success on Test data
pred_test = clf.predict_proba(final_X_test)
pred_test=(pred_test)[:,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_test,tpr_test ,label='ROC on on Test data
,auc='+str(roc_auc_score(y_test,pred_test)))
plt.plot(fpr_tr,tpr_tr ,label='ROC on on Train data ,auc='+str(roc_auc_score(y_tr,pred_tr)))
plt.legend()
plt.title('ROC on best depth and min_splits')
plt.xlabel('FPR')
plt.ylabel('TPR')
plt.show()
```



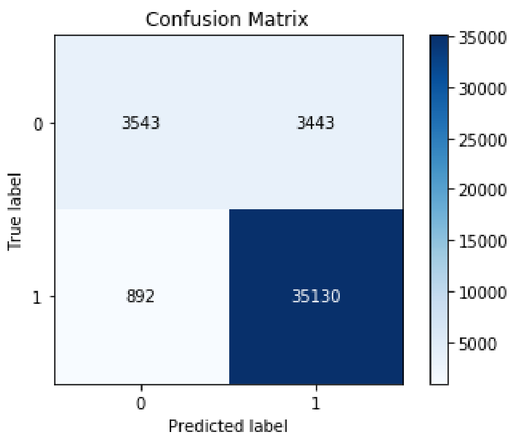


In [45]:

```
#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[45]:

```
<matplotlib.axes._subplots.AxesSubplot at 0x7fcca5d79da0>
```

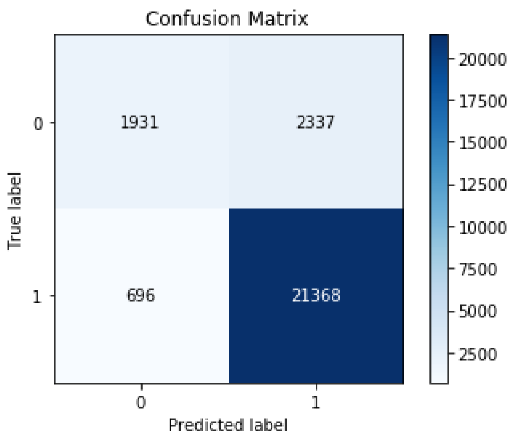


In [46]:

```
#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[46]:

```
<matplotlib.axes._subplots.AxesSubplot at 0x7fcc94c35a90>
```



[6] Conclusions

[6] Conclusions

In [47]:

```
# Please compare all your models using Prettytable library
#since each model takes lot to time to run,instead of using variables,manually entering the obtain
ed values
from prettytable import PrettyTable
x = PrettyTable()
x.field_names = ["Vectorizer", "Model_name", "optimalbaselearner","optimal depth","AUC"]
x.add_row(["BOW","Randomforest",200,10,0.880])
x.add_row(["TFIDF","Randomforest",500,8,0.873])
x.add_row(["AwgW2V","Randomforest",500,10,0.894])
x.add_row(["TFIDF-W2V","Randomforest",500,10,0.872])
x.add_row(["BOW","Xgboost",500,10,0.932])
x.add_row(["TFIDF","Xgboost",1000,5,0.840])
x.add_row(["AwgW2V","Xgboost",500,7,0.810])
x.add_row(["TFIDF-W2V","Xgboost",1000,2,0.889])
print(x)
```

```
+------------+--------------+--------------------+---------------+-------+
| Vectorizer |  Model_name  | optimalbaselearner | optimal depth |  AUC  |
+------------+--------------+--------------------+---------------+-------+
|    BOW     | Randomforest |        200         |       10      |  0.88 |
|   TFIDF    | Randomforest |        500         |       8       | 0.873 |
|   AwgW2V   | Randomforest |        500         |       10      | 0.894 |
| TFIDF-W2V  | Randomforest |        500         |       10      | 0.872 |
|    BOW     |   Xgboost    |        500         |       10      | 0.932 |
|   TFIDF    |   Xgboost    |        1000        |       5       |  0.84 |
|   AwgW2V   |   Xgboost    |        500         |       7       |  0.81 |
| TFIDF-W2V  |   Xgboost    |        1000        |       2       | 0.889 |
+------------+--------------+--------------------+---------------+-------+
```