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.

Connecting to Final dataset, that was created after text pre-processing.

In [6]:

```
import warnings
warnings.filterwarnings('ignore')

#General Libraries
import os
import sqlite3
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
import seaborn as sns
from collections import Counter
import pickle

#Scikit-learn packages
from sklearn.feature_extraction.text import TfidfTransformer
from sklearn.feature_extraction.text import TfidfVectorizer
from sklearn.feature_extraction.text import CountVectorizer
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import StandardScaler
import string
from nltk.corpus import stopwords
from nltk.stem import PorterStemmer
from nltk.stem.wordnet import WordNetLemmatizer


#Scikit-learn scoring metrics
```

```
from sklearn import metrics
from sklearn.metrics import roc_curve, auc
from sklearn.metrics import roc_auc_score
from sklearn.metrics import f1_score
from sklearn.metrics import classification_report
from sklearn.metrics import confusion_matrix
from sklearn.metrics import accuracy_score

#Gensim model Libraries
from gensim.models import Word2Vec
from gensim.models import KeyedVectors
import pickle
from tqdm import tqdm

import warnings
warnings.filterwarnings('ignore')

# importing specific libraries
from sklearn.model_selection import GridSearchCV
from sklearn.model_selection import RandomizedSearchCV
from scipy.sparse import coo_matrix, hstack,vstack

import warnings
warnings.filterwarnings('ignore')
```

In [7]:

```
# 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""", 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 (525814, 10)
```

Out[7]:

	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
[illegible]	[illegible]	[illegible]	[illegible]	Natalia Corres	[illegible]	[illegible]	[illegible]	[illegible]	"Delight"

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

In [8]:

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

In [9]:

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

```
(80668, 7)
```

Out[9]:

	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 [10]:

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

Out[10]:

	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 [11]:

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

Out[11]:

```
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 [12]:

```
display= pd.read_sql_query("""
SELECT *
```

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

Out[12]:

	Id	ProductId	UserId	ProfileName	HelpfulnessNumerator	HelpfulnessDenominator	Score	Time	Summ
0	78445	B000HDL1RQ	AR5J8UI46CURR	Geetha Krishnan	2	2	5	1199577600	LOACK QUADRAT VANII WAFE
1	138317	B000HDOPYC	AR5J8UI46CURR	Geetha Krishnan	2	2	5	1199577600	LOACK QUADRAT VANII WAFE
2	138277	B000HDOPYM	AR5J8UI46CURR	Geetha Krishnan	2	2	5	1199577600	LOACK QUADRAT VANII WAFE
3	73791	B000HDOPZG	AR5J8UI46CURR	Geetha Krishnan	2	2	5	1199577600	LOACK QUADRAT VANII WAFE
4	155049	B000PAQ75C	AR5J8UI46CURR	Geetha Krishnan	2	2	5	1199577600	LOACK QUADRAT VANII 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 [13]:

```
#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 [14]:

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

Out[14]:

```
(364173, 10)
```

In [15]:

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

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

Out[15]:

```
69.25890143662969
```

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 [16]:

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

display.head()
```

Out[16]:

	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 [17]:

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

In [18]:

```
#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()
```

```
(364171, 10)
```

Out[18]:

```
1    307061
0     57110
Name: Score, dtype: int64
```

In [19]:

```
final.head(50)
```

Out[19]:

	Id	ProductId	UserId	ProfileName	HelpfulnessNumerator	HelpfulnessDenominator	Score	Time
138706	150524	0006641040	ACITT7DI6IDDL	shari zychinski	0	0	1	939340800

	Id	ProductId	UserId	ProfileName	HelpfulnessNumerator	HelpfulnessDenominator	Score	Time	
138688	150506	0006641040	A2IW4PEEKO2R0U	Tracy	1	1	1	1194739200	
138689	150507	0006641040	A1S4A3IQ2MU7V4	sally sue "sally sue"	1	1	1	1191456000	ri
138690	150508	0006641040	AZGXZ2UUK6X	Catherine Hallberg " (Kate)"	1	1	1	1076025600	
138691	150509	0006641040	A3CMRKGE0P909G	Teresa	3	4	1	1018396800	A t
138693	150511	0006641040	A1C9K534BCI9GO	Laura Purdie Salas	0	0	1	1344211200	ar
138694	150512	0006641040	A1DJXZA5V5FFVA	A. Conway	0	0	1	1338249600	
138695	150513	0006641040	ASH0DZQQF6AIZ	tessarat	0	0	1	1325721600	
138696	150514	0006641040	A2ONB6ZA292PA	Rosalind Matzner	0	0	1	1313884800	
138697	150515	0006641040	A2RTT81R6Y3R7X	Lindylu	0	0	1	1303171200	
138687	150505	0006641040	A2PTSM496CF40Z	Jason A. Teeple "Nobody made a greater mistak...	1	1	1	1210809600	
138698	150516	0006641040	A3OI7ZGH6WZJ5G	Mary Jane Rogers "Maedchen"	0	0	1	1293840000	
138700	150518	0006641040	AK1L4EJBA23JF	L. M. Kraus	0	0	1	1288224000	
138701	150519	0006641040	A12HY5OZ2QNK4N	Elizabeth H. Roessner	0	0	1	1256774400	
138702	150520	0006641040	ADBFSA9KTQANE	James L. Hammock "Pucks Buddy"	0	0	1	1256688000	
138703	150521	0006641040	A3RMCRB2NDTDYP	Carol Carruthers	0	0	1	1243468800	T
138704	150522	0006641040	A1S3C5OFU508P3	Charles Ashbacher	0	0	1	1219536000	C e
138705	150523	0006641040	A2P4F2UO0UMP8C	Elizabeth A. Curry "Lovely Librarian"	0	0	1	1096675200	

	Id	ProductId	UserId	ProfileName	HelpfulnessNumerator	HelpfulnessDenominator	Score	Time	D
138707	150525	0006641040	A2QID6VCFTY51R	Rick	1	2	1	1025481600	s a
138708	150526	0006641040	A3E9QZFE9KXH8J	R. Mitchell	11	18	0	1129507200	
138709	150529	0006641040	A25ACLV5KPB4W	Matt Hetling "Matt"	0	1	1	1108425600	
138699	150517	0006641040	ABW4IC5G5G8B5	kevin clark	0	0	1	1291075200	
138686	150504	0006641040	AQEYF1AXARWJZ	Les Sinclair "book maven"	1	1	1	1212278400	
138692	150510	0006641040	AM1MNZMYMS7D8	Dr. Joshua Grossman	0	0	1	1348358400	Pr
138680	150498	0006641040	A3SJWISOCP31TR	R. J. Wells	2	2	1	1176336000	A
138677	150494	0006641040	AYZ0PR5QZROD1	Mother of 3 girls	3	3	1	1173312000	
138678	150496	0006641040	A3KKR87BJ0C595	Gretchen Goodfellow "Lover of children's lit"	3	3	1	1111363200	Y c u
138685	150503	0006641040	A3R5XMPFU8YZ4D	Her Royal Motherliness "Nana"	1	1	1	1233964800	
138684	150502	0006641040	AVFMJ50HNO21J	Jane Doe	1	1	1	1324944000	
138679	150497	0006641040	A1HKYQOFC8ZZCH	Maria Apolloni "lanarossa"	2	2	0	1334707200	T
138683	150501	0006641040	AJ46FKXOVC7NR	Nicholas A Mesiano	2	2	1	940809600	gr s
138676	150493	0006641040	AMX0PJKV4PPNJ	E. R. Bird "Ramseelbird"	71	72	1	1096416000	c C
138682	150500	0006641040	A1IJKK6Q1GTEAY	A Customer	2	2	1	1009324800	
138681	150499	0006641040	A3E7R866M94L0C	L. Barker "simienwolf"	2	2	1	1065830400	e
476617	515426	141278509X	AB1A5EGHHVA9M	CHelmic	1	1	1	1332547200	

	Id	ProductId	UserId	ProfileName	HelpfulnessNumerator	HelpfulnessDenominator	Score	Time	
22621	24751	2734888454	A1C298ITT645B6	Hugh G. Pritchard	0	0	1	1195948800	
22620	24750	2734888454	A13ISQV0U9GZIC	Sandikaye	1	1	0	1192060800	
284375	308077	2841233731	A3QD68O22M2XHQ	LABRNTH	0	0	1	1345852800	G b
157850	171161	7310172001	AFXMWPNS1BLU4	H. Sandler	0	0	1	1229385600	
157849	171160	7310172001	A74C7IARQEM1R	stucker	0	0	1	1230076800	
157833	171144	7310172001	A1V5MY8V9AWUQB	Cheryl Sapper "champagne girl"	0	0	1	1244764800	- h
157832	171143	7310172001	A2SWO60IW01VPX	Sam	0	0	1	1252022400	M
157837	171148	7310172001	A3TFTWTG2CC1GA	J. Umphress	0	0	1	1240272000	
157831	171142	7310172001	A2ZO1AYFVQYG44	Cindy Rellie "Rellie"	0	0	1	1254960000	I c I l
157830	171141	7310172001	AZ40270J4JBZN	Zhinka Chunmee "gamer from way back in the 70's"	0	0	1	1264291200	D
157829	171140	7310172001	ADXXVGRCGQQUO	Richard Pearlstein	0	0	1	1264377600	
157828	171139	7310172001	A13MS1JQG2ADOJ	C. Perrone	0	0	1	1265760000	
157827	171138	7310172001	A13LAE0YTXA11B	Dita Vyslouzilova "dita"	0	0	1	1269216000	
157848	171159	7310172001	A16GY2RCF410DT	LB	0	0	1	1231718400	
157834	171145	7310172001	A1L8DNQYY69L2Z	R. Flores	0	0	1	1243728000	

[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 [20]:

```
# 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)
```

```
this witty little book makes my son laugh at loud. i recite it in the car as we're driving along a
nd he always can sing the refrain. he's learned about whales, India, drooping roses:  i love all t
he new words this book  introduces and the silliness of it all.  this is a classic book i am
willing to bet my son will STILL be able to recite from memory when he is  in college
==================================================
I was really looking forward to these pods based on the reviews.  Starbucks is good, but I prefer
bolder taste.... imagine my surprise when I ordered 2 boxes - both were expired! One expired back
in 2005 for gosh sakes.  I admit that Amazon agreed to credit me for cost plus part of shipping, b
ut geez, 2 years expired!!!  I'm hoping to find local San Diego area shoppe that carries pods so t
hat I can try something different than starbucks.
==================================================
Great ingredients although, chicken should have been 1st rather than chicken broth, the only thing
I do not think belongs in it is Canola oil. Canola or rapeseed is not someting a dog would ever fi
nd in nature and if it did find rapeseed in nature and eat it, it would poison them. Today's Food
industries have convinced the masses that Canola oil is a safe and even better oil than olive or v
irgin coconut, facts though say otherwise. Until the late 70's it was poisonous until they figured
out a way to fix that. I still like it but it could be better.
==================================================
Can't do sugar.  Have tried scores of SF Syrups.  NONE of them can touch the excellence of this
product.<br /><br />Thick, delicious.  Perfect.  3 ingredients: Water, Maltitol, Natural Maple
Flavor.  PERIOD.  No chemicals.  No garbage.<br /><br />Have numerous friends & family members
hooked on this stuff.  My husband & son, who do NOT like "sugar free" prefer this over major label
regular syrup.<br /><br />I use this as my SWEETENER in baking: cheesecakes, white brownies,
muffins, pumpkin pies, etc... Unbelievably delicious...<br /><br />Can you tell I like it? :)
==================================================
```

In [21]:

```
# remove urls from text python: https://stackoverflow.com/a/40823105/4084039
import re
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)
```

```
this witty little book makes my son laugh at loud. i recite it in the car as we're driving along a
nd he always can sing the refrain. he's learned about whales, India, drooping roses:  i love all t
he new words this book  introduces and the silliness of it all.  this is a classic book i am
willing to bet my son will STILL be able to recite from memory when he is  in college
```

In [22]:

```
# 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)
```

```
this witty little book makes my son laugh at loud. i recite it in the car as we're driving along a
nd he always can sing the refrain. he's learned about whales, India, drooping roses:  i love all t
he new words this book  introduces and the silliness of it all.  this is a classic book i am
willing to bet my son will STILL be able to recite from memory when he is  in college
==================================================
I was really looking forward to these pods based on the reviews.  Starbucks is good, but I prefer
bolder taste.... imagine my surprise when I ordered 2 boxes - both were expired! One expired back
in 2005 for gosh sakes.  I admit that Amazon agreed to credit me for cost plus part of shipping, b
ut geez, 2 years expired!!!  I'm hoping to find local San Diego area shoppe that carries pods so t
hat I can try something different than starbucks.
==================================================
Great ingredients although, chicken should have been 1st rather than chicken broth, the only thing
I do not think belongs in it is Canola oil. Canola or rapeseed is not someting a dog would ever fi
nd in nature and if it did find rapeseed in nature and eat it, it would poison them. Today's Food
industries have convinced the masses that Canola oil is a safe and even better oil than olive or v
irgin coconut, facts though say otherwise. Until the late 70's it was poisonous until they figured
out a way to fix that. I still like it but it could be better.
==================================================
Can't do sugar.  Have tried scores of SF Syrups.  NONE of them can touch the excellence of this
product.Thick, delicious.  Perfect.  3 ingredients: Water, Maltitol, Natural Maple Flavor.
PERIOD.  No chemicals.  No garbage.Have numerous friends & family members hooked on this stuff.  M
y husband & son, who do NOT like "sugar free" prefer this over major label regular syrup.I use thi
s as my SWEETENER in baking: cheesecakes, white brownies, muffins, pumpkin pies, etc...
Unbelievably delicious...Can you tell I like it? :)
```

In [23]:

```
# 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 [24]:

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

```
Great ingredients although, chicken should have been 1st rather than chicken broth, the only thing
I do not think belongs in it is Canola oil. Canola or rapeseed is not someting a dog would ever fi
nd in nature and if it did find rapeseed in nature and eat it, it would poison them. Today is Food
industries have convinced the masses that Canola oil is a safe and even better oil than olive or v
irgin coconut, facts though say otherwise. Until the late 70 is it was poisonous until they
figured out a way to fix that. I still like it but it could be better.
==================================================
```

In [25]:

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

```
this witty little book makes my son laugh at loud. i recite it in the car as we're driving along a
nd he always can sing the refrain. he's learned about whales, India, drooping roses:  i love all t
he new words this book  introduces and the silliness of it all.  this is a classic book i am
willing to bet my son will STILL be able to recite from memory when he is  in college
```

In [26]:

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

```
Great ingredients although chicken should have been 1st rather than chicken broth the only thing I
do not think belongs in it is Canola oil Canola or rapeseed is not someting a dog would ever find
in nature and if it did find rapeseed in nature and eat it it would poison them Today is Food indu
stries have convinced the masses that Canola oil is a safe and even better oil than olive or virgi
n coconut facts though say otherwise Until the late 70 is it was poisonous until they figured out
a way to fix that I still like it but it could be better
```

In [27]:

```
# 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", "k","n","x","e"])
```

In [28]:

```python
# 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%|████████████████████████████████████████████████████████████████████████| 364171/364171
[02:00<00:00, 3026.75it/s]
```

In [29]:

```python
preprocessed_reviews[1500]
```

Out[29]:

```
'great ingredients although chicken rather chicken broth thing not think belongs canola oil canola
rapeseed not someting dog would ever find nature find rapeseed nature eat would poison today food
industries convinced masses canola oil safe even better oil olive virgin coconut facts though say
otherwise late poisonous figured way fix still like could better'
```

[3.2] Preprocessing Review Summary

In [30]:

```python
## Similartly you can do preprocessing for review summary also.
preprocessed_review_summary = []
# tqdm is for printing the status bar
for sentance in tqdm(final['Summary'].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_review_summary.append(sentance.strip())
```

```
100%|████████████████████████████████████████████████████████████████████████| 364171/364171
[01:23<00:00, 4368.60it/s]
```

[3.3] Adding Review Length

In [31]:

```python
## Similartly you can do preprocessing for review summary also.
new_review_len = []
# tqdm is for printing the status bar
for sentance in tqdm(final['Text'].values):
    rev_len = len(sentance)
    new_review_len.append(rev_len)
```

```
100%|█████████████████████████████████████████████████████████████████████| 364171/364171
[00:00<00:00, 1431297.88it/s]
```

[3.4]Creating new DS with New Features

In [32]:

```
final['Cleaned_text'] = preprocessed_reviews
final['Cleaned_summary'] = preprocessed_review_summary
final['Review_length'] = new_review_len
```

In [33]:

```
final.head(2)
```

Out[33]:

	Id	ProductId	UserId	ProfileName	HelpfulnessNumerator	HelpfulnessDenominator	Score	Time	Su
138706	150524	0006641040	ACITT7DI6IDDL	shari zychinski	0	0	1	939340800	E educ
138688	150506	0006641040	A2IW4PEEKO2R0U	Tracy	1	1	1	1194739200	Lo boo th

In [57]:

```
# store final table into an SQlLite table for future.
conn = sqlite3.connect('final.sqlite')
c=conn.cursor()
conn.text_factory = str
final.to_sql('Reviews', conn,  schema=None, if_exists='replace', \
            index=True, index_label=None, chunksize=None, dtype=None)
conn.close()
```

In [58]:

```
con = sqlite3.connect("final.sqlite")
final = pd.read_sql_query("""SELECT * FROM Reviews""",con)
```

In [59]:

```
final.shape
```

Out[59]:

(364171, 16)

In [60]:

```
final.head(2)
```

Out[60]:

	level_0	index	Id	ProductId	UserId	ProfileName	HelpfulnessNumerator	HelpfulnessDenominator	Score	
0	0	138706	150524	0006641040	ACITT7DI6IDDL	shari zychinski	0	0	1	9393

	level_0	index	Id	ProductId	UserId	ProfileName	HelpfulnessNumerator	HelpfulnessDenominator	Score
1	1	138688	150506	0006641040	A2IW4PEEKO2R0U	Tracy	1	1	1	11947

[3.5] Merge features (Cleaned_Text,Cleaned_summary and Review_length) for feature Engineering

In [34]:

```
range(len(final['Cleaned_text']))
```

Out[34]:

```
range(0, 364171)
```

In [50]:

```
train = []
for i in range(len(final['Cleaned_text'])):
    train.append(final['Cleaned_summary'].values[i] + " " + final['Cleaned_text'].values[i])
```

In [51]:

```
final['Cleaned_text'].values[1]
```

Out[51]:

```
'grew reading sendak books watching really rosie movie incorporates love son loves however miss ha
rd cover version paperbacks seem kind flimsy takes two hands keep pages open'
```

In [52]:

```
final['Cleaned_summary'].values[1]
```

Out[52]:

```
'love book miss hard cover version'
```

In [53]:

```
final['Review_length'].values[1]
```

Out[53]:

```
260
```

In [54]:

```
train[1]
```

Out[54]:

```
'love book miss hard cover version grew reading sendak books watching really rosie movie
incorporates love son loves however miss hard cover version paperbacks seem kind flimsy takes two
hands keep pages open'
```

In [55]:

```
final['train'] = train
```

In [56]:

```
final['train'].values[1]
```

```
final['train'].values[1]
```

Out[56]:

```
'love book miss hard cover version grew reading sendak books watching really rosie movie incorporates love son loves however miss hard cover version paperbacks seem kind flimsy takes two hands keep pages open'
```

Spliting Train and test data.

In [61]:

```
S100_sample_data = final.sample(n=50000)
S100_sorted_data = S100_sample_data.sort_values('Time',ascending=True)
```

In [62]:

```
S100_sorted_data.head(3)
```

Out[62]:

	level_0	index	Id	ProductId	UserId	ProfileName	HelpfulnessNumerator	HelpfulnessDenominator	Score	
30	30	138683	150501	0006641040	AJ46FKXOVC7NR	Nicholas A Mesiano	2	2	1	94
308	308	346041	374343	B00004CI84	A1B2IZU1JLZA6	Wes	19	23	0	94
215	215	70688	76882	B00002N8SM	A32DW342WBJ6BX	Buttersugar	0	0	1	94

In [63]:

```
#Splitting the dataset to Train and Test.
S100_Train, S100_Test = train_test_split(S100_sorted_data, test_size=0.3, random_state=0)
S100_Y_train = S100_Train['Score']
S100_Y_test = S100_Test['Score']
S100_Y_train.shape,S100_Y_test.shape
```

Out[63]:

```
((35000,), (15000,))
```

1. Creating Vectorizers

SET 1 - BOW Vectorizers

In [64]:

```
S100_count_vect = CountVectorizer() #in scikit-learn
# BOW Vectorizer: Train data
S100_BOW_X_train = S100_count_vect.fit_transform(S100_Train['train'].values)

#BOW Vectorizer: Test data
S100_BOW_X_test = S100_count_vect.transform(S100_Test['train'].values)

print(S100_BOW_X_train.shape,S100_BOW_X_test.shape)
```

```
(35000, 37818) (15000, 37818)
```

SET 2 TFIDF Vectorizers

In [90]:

```
S100_tfidf_vect = TfidfVectorizer()
# TFIDF: Train data
S100_tfidf_X_train = S100_tfidf_vect.fit_transform(S100_Train['train'].values)

# TFIDF: Test data
S100_tfidf_X_test = S100_tfidf_vect.transform(S100_Test['train'].values)

print(S100_tfidf_X_train.shape,S100_tfidf_X_test.shape)
```

```
(35000, 37818) (15000, 37818)
```

SET 3 Word2Vec Vectorizers

In [91]:

```
import gensim
#Train data
i=0
list_of_train = []
for sentence in S100_Train['train'].values:
    list_of_train.append(sentence.split())

w2v_model_train = gensim.models.Word2Vec(list_of_train,min_count=5,size=50,workers=4)
words_train=list(w2v_model_train.wv.vocab)

# average Word2Vec
# compute average word2vec for each review.
S100_W2V_train = []; # the avg-w2v for each sentence/review is stored in this list
for sent in tqdm(list_of_train): # for each review/sentence
    sent_vec = np.zeros(50) # as word vectors are of zero length
    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 words_train:
            vec = w2v_model_train.wv[word]
            sent_vec += vec
            cnt_words += 1
    if cnt_words != 0:
        sent_vec /= cnt_words
    S100_W2V_train.append(sent_vec)
print(len(S100_W2V_train))
print(len(S100_W2V_train[0]))
```

```
100%|████████████████████████████████████████████████████████████████████████████| 35000/35000 [01:
49<00:00, 320.65it/s]
```

```
35000
50
```

In [92]:

```
#Test data

#i=0
list_of_test = []
for sentence in S100_Test['train'].values:
    list_of_test.append(sentence.split())

# average Word2Vec
# compute average word2vec for each review.
S100_W2V_test = []; # the avg-w2v for each sentence/review is stored in this list
for sent in tqdm(list_of_test): # for each review/sentence
    sent_vec = np.zeros(50) # as word vectors are of zero length
```

```
    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 words_train:
            vec = w2v_model_train.wv[word]
            sent_vec += vec
            cnt_words += 1
    if cnt_words != 0:
        sent_vec /= cnt_words
    S100_W2V_test.append(sent_vec)
print(len(S100_W2V_test))
print(len(S100_W2V_test[0]))
```

```
100%|██████████████████████████████████████████████████████████████████████████| 15000/15000 [00:
50<00:00, 299.92it/s]
```

```
15000
50
```

SET 4 TFIDF-Word2Vec Vectorizers

In [97]:

```
tf_idf_vect = TfidfVectorizer()
# Please write all the code with proper documentation
# S = ["abc def pqr", "def def def abc", "pqr pqr def"]
model = TfidfVectorizer()
model.fit(final['train'])
# we are converting a dictionary with word as a key, and the idf as a value
dictionary = dict(zip(model.get_feature_names(), list(model.idf_)))

# TF-IDF weighted Word2Vec
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

#standardized_weight_w2v = StandardScaler().fit_transform(tfidf_sent_vectors)
#print(standardized_weight_w2v.shape)
```

In [102]:

```
def tfidfw2v(test):
    '''
    Returns tfidf word2vec
    '''
    tfidf_sent_vectors = []; # the tfidf-w2v for each sentence/review is stored in this list
    i=0
    list_of_sentance=[]
    for sentance in test:
        list_of_sentance.append(sentance.split())

    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 words_train and word in tfidf_feat:
                vec = w2v_model_train.wv[word]
                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)

    return tfidf_sent_vectors
```

In [103]:

```
S100_tfidf_W2V_train = tfidfw2v(S100_Train['train'])
S100_tfidf_W2V_test  = tfidfw2v(S100_Test['train'])
```

```
100%|█████████████████████████████████████████████████████████████████████████| 35000/35000
[1:17:19<00:00,  7.09it/s]
```

```
[1:17:19<00:00,  7.09it/s]
100%|██████████████████████████████████████████████████| 15000/15000 [41:39<00:00,  8.37it/s]
```

In [104]:

```
len(S100_tfidf_W2V_train)
```

Out[104]:

```
35000
```

In [105]:

```
len(S100_tfidf_W2V_train[0])
```

Out[105]:

```
50
```

Export and import Vectorizers

In [106]:

```
import pickle

def savetofile(obj,filename):
    pickle.dump(obj,open(filename+".p","wb"))

def openfromfile(filename):
    temp = pickle.load(open(filename+".p","rb"))
    return temp
```

Exporting vectorizers to local for future use

In [107]:

```
###########---------- exporting 100K samples vectorizers.
# Predict labels
savetofile(S100_Y_train,'S100_Y_train')
savetofile(S100_Y_test,'S100_Y_test')

#BOW
savetofile(S100_BOW_X_train,'S100_BOW_X_train')
savetofile(S100_BOW_X_test,'S100_BOW_X_test')


#TFIDF
savetofile(S100_tfidf_X_train,'S100_tfidf_X_train')
savetofile(S100_tfidf_X_test,'S100_tfidf_X_test')

# Avg Word2Vec

savetofile(S100_W2V_train,'S100_W2V_train')
savetofile(S100_W2V_test,'S100_W2V_test')

# TFIDF - Word2Vec

savetofile(S100_tfidf_W2V_train,'S100_tfidf_W2V_train')
savetofile(S100_tfidf_W2V_test,'S100_tfidf_W2V_test')
```

Dump of Vectorizers object for future use

In [108]:

```
# 100K sample
savetofile(S100_count_vect,'S100_count_vect')
savetofile(S100_tfidf_vect,'S100_tfidf_vect')
```

```
saveToFile(S100_tfidf_vect, 'S100_tfidf_vect')
```

Importing Vectors and Vectorizers

In [109]:

```
###########---------- importing 100K samples vectorizers.
# Predict labels
S100_Y_train = openfromfile('S100_Y_train')
S100_Y_test  = openfromfile('S100_Y_test')

#BOW
S100_BOW_X_train  = openfromfile('S100_BOW_X_train')
S100_BOW_X_test  = openfromfile('S100_BOW_X_test')


#TFIDF
S100_tfidf_X_train  = openfromfile('S100_tfidf_X_train')
S100_tfidf_X_test  = openfromfile('S100_tfidf_X_test')

# Avg Word2Vec

S100_W2V_train  = openfromfile('S100_W2V_train')
S100_W2V_test  = openfromfile('S100_W2V_test')

# TFIDF - Word2Vec
S100_tfidf_W2V_train = openfromfile('S100_tfidf_W2V_train')
S100_tfidf_W2V_test  = openfromfile('S100_tfidf_W2V_test')

# Vectorizers
# Sample 100K
S100_count_vect = openfromfile('S100_count_vect')
S100_tfidf_vect = openfromfile('S100_tfidf_vect')
```

Adding Review length as a new feature

BOW

In [80]:

```
bow_A = S100_BOW_X_train
bow_B = S100_BOW_X_test

len_train = S100_Train['Review_length']
len_test  = S100_Test['Review_length']
```

In [81]:

```
bow_A.shape,bow_B.shape
```

Out[81]:

```
((35000, 37818), (15000, 37818))
```

In [82]:

```
#bow_A = coo_matrix(A)
len_A = coo_matrix(len_train)
#bow_B = coo_matrix(B)
len_B = coo_matrix(len_test)
```

In [83]:

```
len_A = len_A.reshape(35000,1)
len_B = len_B.reshape(15000,1)
print(len_A.shape)
print(len_B.shape)
```

```
(35000, 1)
(15000, 1)
```

In [84]:

```
S100_BOW_X_train = hstack([len_A,bow_A])
S100_BOW_X_test = hstack([len_B,bow_B])
S100_BOW_X_train.shape,S100_BOW_X_test.shape
```

Out[84]:

```
((35000, 37819), (15000, 37819))
```

TFIDF

In [110]:

```
A = S100_tfidf_X_train
B = S100_tfidf_X_test
tfidf_A = coo_matrix(A)
tfidf_B = coo_matrix(B)
S100_tfidf_X_train.shape,S100_tfidf_X_test.shape
```

Out[110]:

```
((35000, 37818), (15000, 37818))
```

In [111]:

```
S100_tfidf_X_train = hstack([len_A,tfidf_A])
S100_tfidf_X_test = hstack([len_B,tfidf_B])
S100_tfidf_X_train.shape,S100_tfidf_X_test.shape
```

Out[111]:

```
((35000, 37819), (15000, 37819))
```

W2V

In [112]:

```
A = S100_W2V_train
B = S100_W2V_test
W2V_A = coo_matrix(A)
W2V_B = coo_matrix(B)
W2V_A.shape,W2V_B.shape
```

Out[112]:

```
((35000, 50), (15000, 50))
```

In [113]:

```
S100_W2V_train = hstack([len_A,W2V_A])
S100_W2V_test = hstack([len_B,W2V_B])
S100_W2V_train.shape,S100_W2V_test.shape
```

Out[113]:

```
((35000, 51), (15000, 51))
```

TFDIF W2V

In [114]:

```
A = S100_tfidf_W2V_train
B = S100_tfidf_W2V_test
tfidf_W2V_A = coo_matrix(A)
tfidf_W2V_B = coo_matrix(B)
tfidf_W2V_A.shape,tfidf_W2V_B.shape
```

Out[114]:

```
((35000, 50), (15000, 50))
```

In [115]:

```
S100_tfidf_W2V_train = hstack([len_A,tfidf_W2V_A])
S100_tfidf_W2V_test = hstack([len_B,tfidf_W2V_B])
S100_tfidf_W2V_train.shape,S100_tfidf_W2V_test.shape
```

Out[115]:

```
((35000, 51), (15000, 51))
```

** ASSIGNMENT BEGINS HERE **

[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

Importing Relevant libraries

In [70]:

```
from sklearn.ensemble import RandomForestClassifier
import xgboost as xgb
from wordcloud import WordCloud
```

Random Forest Common training and test

In [116]:

```
### Random forest have deep trees.
RF_n_est = [5,10,50,100,200,500,1000]
RF_depth = [2,3,4,5,6,7,8,9,10]
```

In [68]:

```
def RFTrainModel(X_train,Y_train,n_est,depth):
    # Tuning parameter for GridSearchCV
    tuned_parameters = dict(n_estimators = n_est,max_depth = depth)

    # Decission tree classiffier object
    model = RandomForestClassifier(class_weight='balanced')

    #GridSearchCV for Cross Validation
    clf = GridSearchCV(model, tuned_parameters, scoring = 'roc_auc', cv=2)
    clf.fit(X_train,Y_train)

    # Fetching best hyperparameters.
    best_estimator, best_depth = clf.best_params_.get('n_estimators'), clf.best_params_.get('max_de
pth')

    # Fetching Train AUC scores and CV AUC scores
    Train_AUC = clf.cv_results_.get('mean_train_score')
    CV_AUC = clf.cv_results_.get('mean_test_score')

    print(clf.best_params_)
    print(clf.best_score_)
    print(clf.best_estimator_)
    #print(clf.grid_scores_)

    return Train_AUC, CV_AUC,best_estimator,best_depth
```

In [77]:

```
def RFTestModel(X_train,Y_train,X_test,Y_test,best_estimator,best_depth):
    # Retrain the model with best hyper-parameters.
    model =
RandomForestClassifier(class_weight='balanced',max_depth=best_depth,n_estimators=best_estimator,ra
```

```
RandomForestClassifier(class_weight='balanced',max_depth=best_depth,n_estimators=best_estimator,ra
ndom_state=0)
    model.fit(X_train,Y_train)

    # Calculate test scores.
     # 1. To calculate AUC values, you have to use predict_proba not just predict,
     # if predict_proba is not available in respective classifier you can go through
calibratedclassifierCV
     # (https://scikit-
learn.org/stable/modules/generated/sklearn.calibration.CalibratedClassifierCV.html)
     # (ex: clf.predict_proba(bow_cv_data)[:, 1]).

    #-------------------------
    # Decision trees provides probabilistic outputs, hence fetch these for plotting.
    train_pred = model.predict_proba(X_train)[:,1]
    test_pred = model.predict_proba(X_test)[:,1]
    train_fpr, train_tpr, train_threshold = roc_curve(Y_train, train_pred )
    test_fpr, test_tpr, test_threshold = roc_curve(Y_test, test_pred )

    # ROC_AUC_SCORE
    test_score = roc_auc_score(Y_test,test_pred)
    print('\nTest AUC score = ' +str(test_score))

    pred_label = model.predict(X_test)

    return test_score,train_fpr, train_tpr,test_fpr, test_tpr,pred_label,model
```

XGBoost Common training and test

In [138]:

```
#GBDT have shallow trees.
XG_n_est = [5,10,50,100,200,500,1000]
XG_depth = [2,3,4,5,6,7,8,9,10]
```

In [142]:

```
def XGTrainModel(X_train,Y_train,n_est,depth):
    # Tuning parameter for GridSearchCV
    tuned_parameters = dict(n_estimators = n_est,max_depth = depth)

    # Decission tree classiffier object
    model = xgb.XGBClassifier()

    #GridSearchCV for Cross Validation
    clf = GridSearchCV(model, tuned_parameters, scoring = 'roc_auc', cv=2)
    clf.fit(X_train,Y_train)

    # Fetching best hyperparameters.
    best_estimator, best_depth = clf.best_params_.get('n_estimators'), clf.best_params_.get('max_de
pth')

    # Fetching Train AUC scores and CV AUC scores
    Train_AUC = clf.cv_results_.get('mean_train_score')
    CV_AUC = clf.cv_results_.get('mean_test_score')

    print(clf.best_params_)
    print(clf.best_score_)
    print(clf.best_estimator_)
    #print(clf.grid_scores_)

    return Train_AUC, CV_AUC,best_estimator,best_depth
```

In [140]:

```
def XGTestModel(X_train,Y_train,X_test,Y_test,best_estimator,best_depth):
    # Retrain the model with best hyper-parameters.
    model = xgb.XGBClassifier(max_depth=best_depth,n_estimators=best_estimator,random_state=0)
    model.fit(X_train,Y_train)

    # Calculate test scores.
     # 1. To calculate AUC values, you have to use predict_proba not just predict,
     # if predict_proba is not available in respective classifier you can go through
```

```
calibratedclassifierCV
     # (https://scikit-
learn.org/stable/modules/generated/sklearn.calibration.CalibratedClassifierCV.html)
     # (ex: clf.predict_proba(bow_cv_data)[:, 1]).

    #-------------------------
    # Decision trees provides probabilistic outputs, hence fetch these for plotting.
    train_pred = model.predict_proba(X_train)[:,1]
    test_pred = model.predict_proba(X_test)[:,1]
    train_fpr, train_tpr, train_threshold = roc_curve(Y_train, train_pred )
    test_fpr, test_tpr, test_threshold = roc_curve(Y_test, test_pred )

    # ROC_AUC_SCORE
    test_score = roc_auc_score(Y_test,test_pred)
    print('\nTest AUC score = ' +str(test_score))

    pred_label = model.predict(X_test)

    return test_score,train_fpr, train_tpr,test_fpr, test_tpr,pred_label,model
```

Train and Cross validation Comparision plots

In [118]:

```
def CompareTrainCV(Train_AUC,CV_AUC,n_est,depth,best_estimator,best_depth):


    ###   ---------- Heat Maps to print the confusion matrix with AUC scores.   ** Code Reference (
ithub.
    #***************************************************************************     ---------------
-----
    df_heatmap = pd. DataFrame(Train_AUC. reshape(7, 9), index=n_est, columns=depth )
    fig = plt. figure(figsize=(10, 10))
    heatmap = sns. heatmap(df_heatmap, annot=True)
    plt. ylabel('# of Base Models' , size=18)
    plt. xlabel('Depth' , size=18)
    plt. title("Train Data", size=24)
    plt. show()

    df_heatmap = pd. DataFrame(CV_AUC. reshape(7, 9), index=n_est, columns=depth )
    fig = plt. figure(figsize=(10, 10))
    heatmap = sns. heatmap(df_heatmap, annot=True)
    plt. ylabel('# of Base Models' , size=18)
    plt. xlabel('Depth' , size=18)
    plt. title("CV Data", size=24)
    plt. show()

    #***************************************************************************

    print('\nThe best optimal number of estimators are ' +str(best_estimator) +' at best optimal de
pth of ' +str(best_depth))
```

Train and Test Comparision plots

In [74]:

```
def CompareTrainTest(Y_test,test_score,train_fpr, train_tpr,test_fpr, test_tpr,pred_label):

    ## ROC Curve for Train and Test data

    plt.figure(1)
    plt.plot([0, 1], [0, 1], 'k--')
    plt.plot(train_fpr, train_tpr, label='Train')
    plt.plot(test_fpr, test_tpr, label='Test')
    plt.xlabel('False positive rate')
    plt.ylabel('True positive rate')
    plt.title('ROC curve')
    plt.legend(loc='best')
    plt.show()

    ## Confusion Matrix
```

```
    # Confusion Matrix to identify, if the model is baised towards positive points or not.
    cm = confusion_matrix(Y_test, pred_label)
    class_label = ["negative", "positive"]
    df_cm = pd.DataFrame(cm, index = class_label, columns = class_label)
    sns.heatmap(df_cm, annot = True, fmt = "d")
    plt.title("Confusion Matrix")
    plt.xlabel("Predicted Label")
    plt.ylabel("Actual Label")
    plt.show()
```

[5.1.1] Applying Random Forests on BOW, SET 1

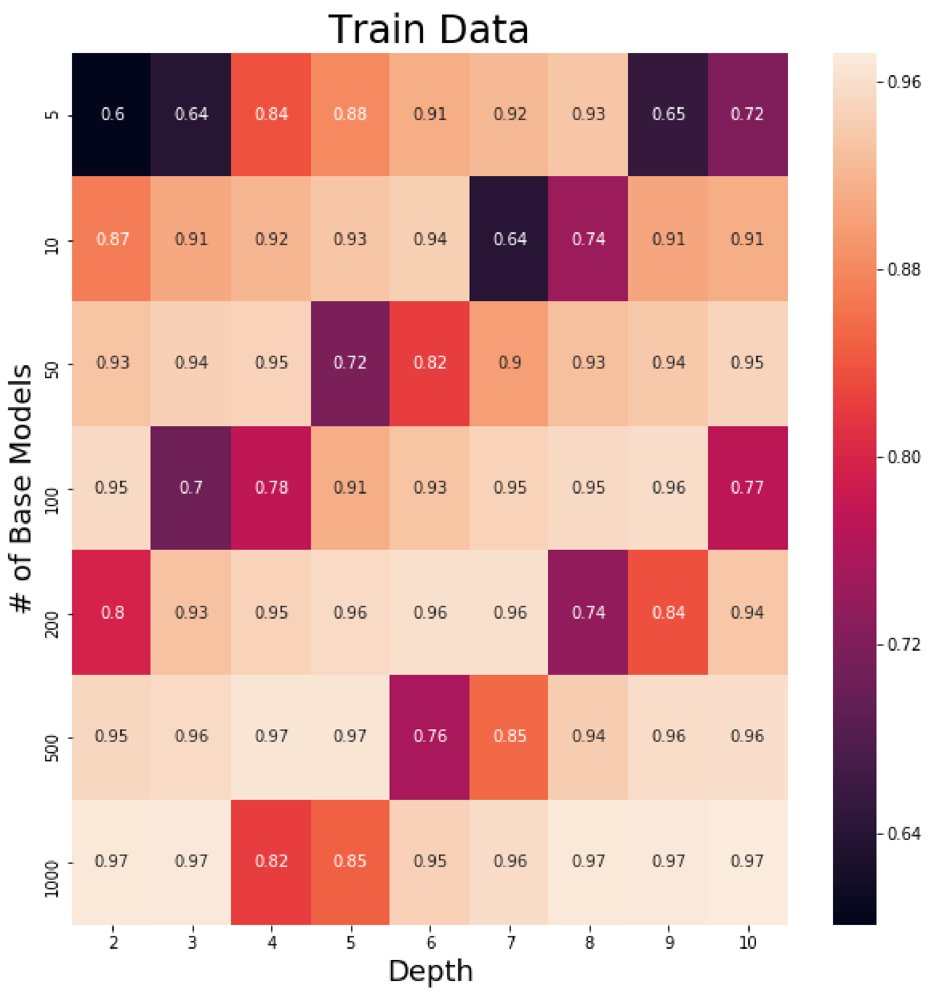
In [117]:

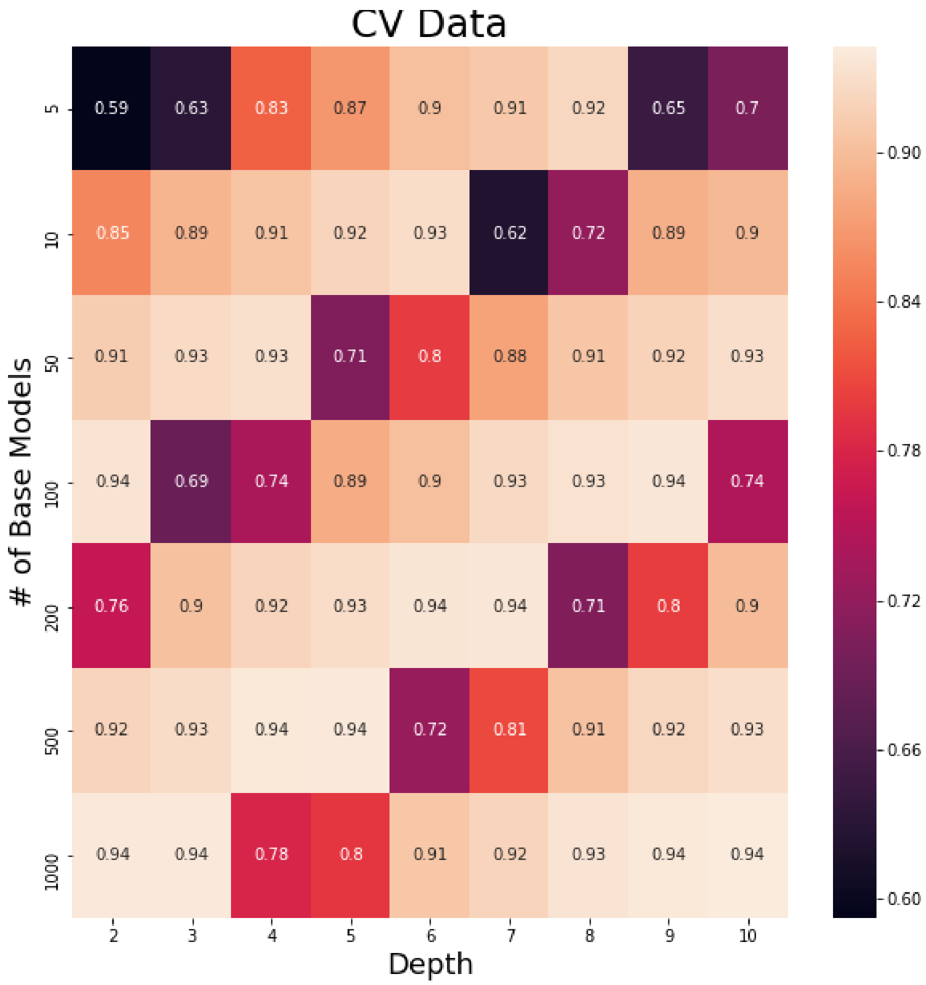
```
# Train model with cross validation
RF_bow_Train_AUC, RF_bow_CV_AUC,RF_bow_best_estimator, RF_bow_best_depth =
RFTrainModel(S100_BOW_X_train, S100_Y_train, RF_n_est,RF_depth)
```

```
{'max_depth': 10, 'n_estimators': 1000}
0.942389929832294
RandomForestClassifier(bootstrap=True, class_weight='balanced',
            criterion='gini', max_depth=10, max_features='auto',
            max_leaf_nodes=None, min_impurity_decrease=0.0,
            min_impurity_split=None, min_samples_leaf=1,
            min_samples_split=2, min_weight_fraction_leaf=0.0,
            n_estimators=1000, n_jobs=None, oob_score=False,
            random_state=None, verbose=0, warm_start=False)
```

In [119]:

```
## Train Vs Cross Validation (Prevention of Overfitting and underfitting of the model)
CompareTrainCV(RF_bow_Train_AUC,
RF_bow_CV_AUC,RF_n_est,RF_depth,RF_bow_best_estimator,RF_bow_best_depth)
```






```
The best optimal number of estimators are 1000 at best optimal depth of 10
```

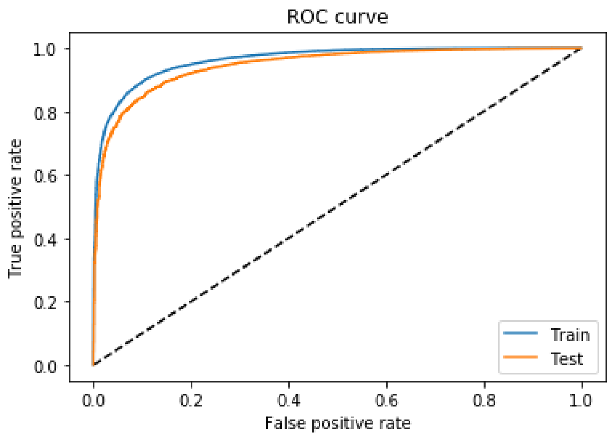
In [120]:

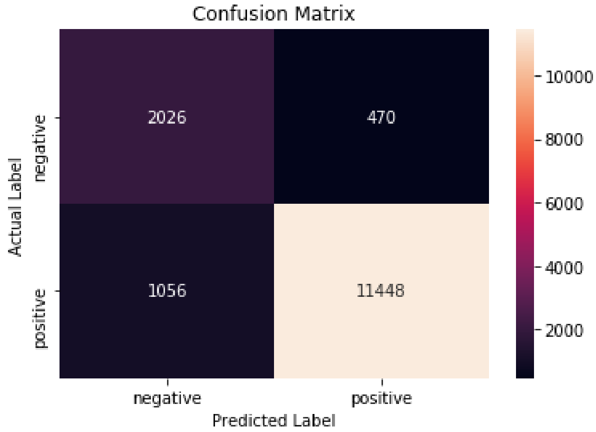
```
## Retrain the best model and test
RF_bow_test_score,RF_bow_train_fpr, RF_bow_train_tpr,RF_bow_test_fpr,
RF_bow_test_tpr,RF_bow_pred_label,RF_bow_model =
RFTestModel(S100_BOW_X_train,S100_Y_train,S100_BOW_X_test,S100_Y_test,RF_bow_best_estimator,RF_bow_
best_depth)
```

```
Test AUC score = 0.9458545220657595
```

In [121]:

```
## Train Vs Test Scores performance
CompareTrainTest(S100_Y_test,RF_bow_test_score,RF_bow_train_fpr, RF_bow_train_tpr,RF_bow_test_fpr,
RF_bow_test_tpr,RF_bow_pred_label)
```





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

In [122]:

```
n = 20
top_features = []
top_feature_names = S100_count_vect.get_feature_names()
coefs = sorted(zip(RF_bow_model.feature_importances_, top_feature_names))
top = coefs[:-(n + 1):-1]
#print('\033[1m' + "feature_importances\tfeatures" + '\033[0m')
#print("="*35)
for (coef1, feat1) in top:
    #print("%.4f\t\t\t%-15s" % (coef1, feat1))
    top_features.append(feat1)

# printing word cloud
wordcloud = WordCloud(background_color='white',width=1600,height=800).generate(" ".join(top_feature
s))
fig = plt.figure(figsize=(15,15))
plt.imshow(wordcloud)
plt.axis('off')
plt.tight_layout(pad=0)
fig.savefig("bow_RF_top_features.png")
plt.show()
```

[5.1.3] Applying Random Forests on TFIDF, SET 2

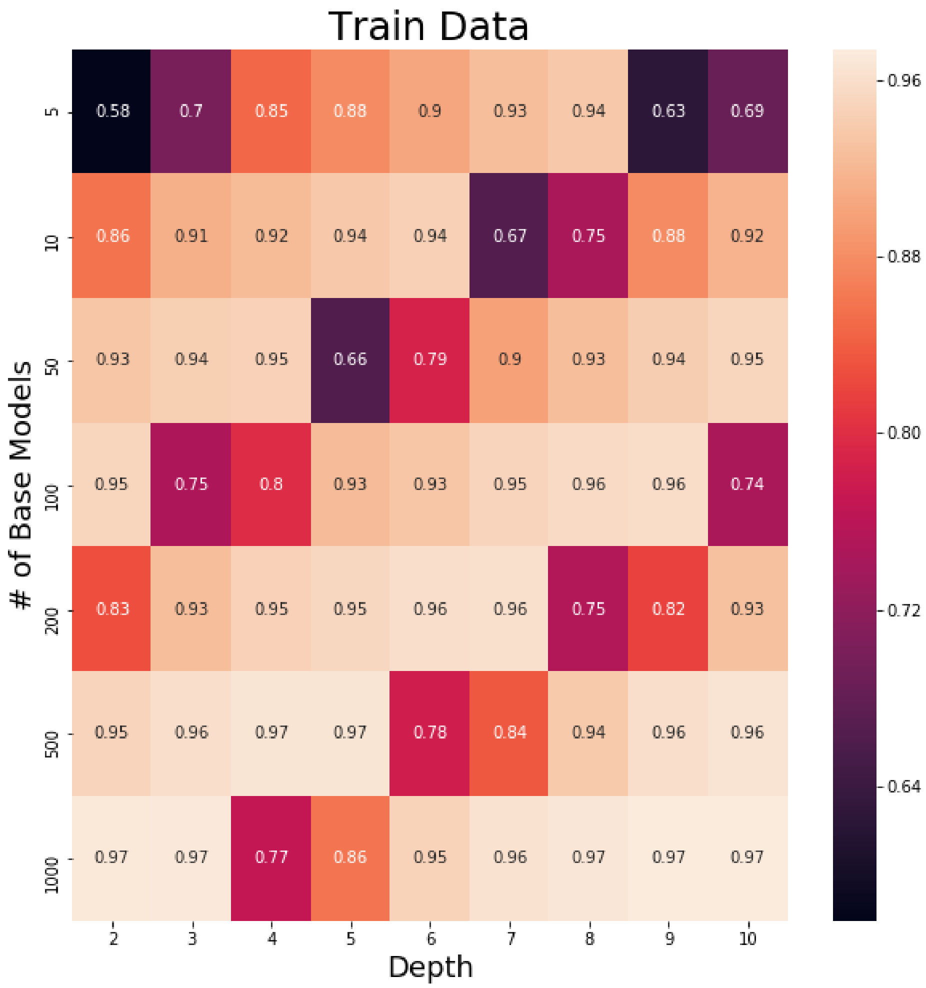
In [123]:

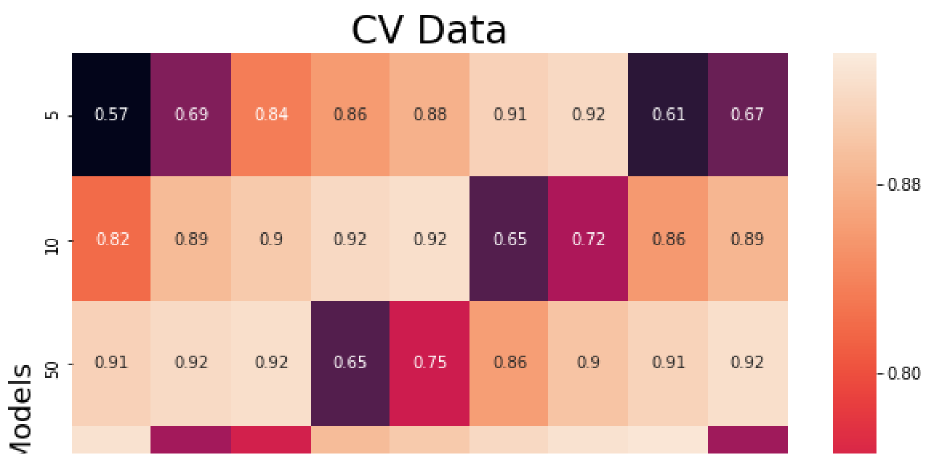
```
# Train model with cross validation
RF_tfidf_Train_AUC,RF_tfidf_CV_AUC,RF_tfidf_best_estimator,RF_tfidf_best_depth =
RFTrainModel(S100_tfidf_X_train, S100_Y_train, RF_n_est,RF_depth)
```

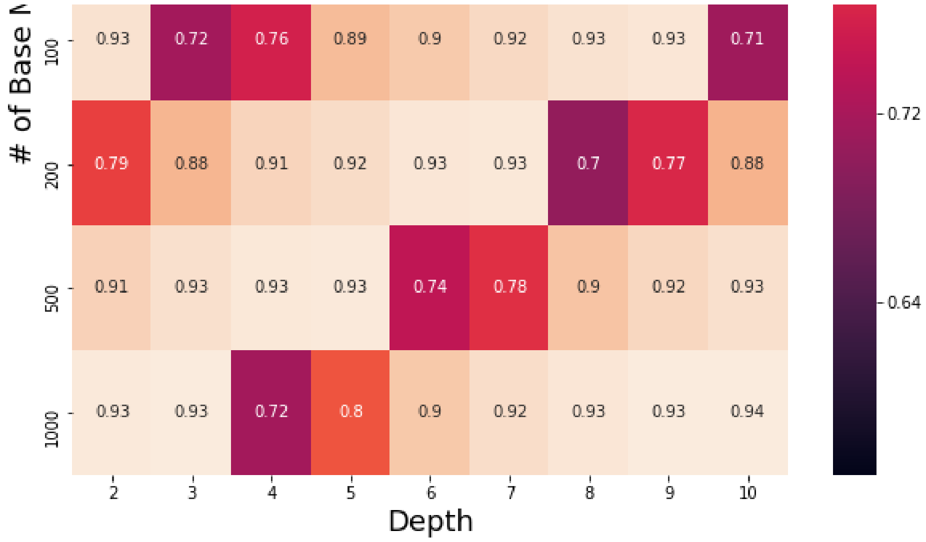
```
{'max_depth': 10, 'n_estimators': 1000}
0.9356991057374777
RandomForestClassifier(bootstrap=True, class_weight='balanced',
            criterion='gini', max_depth=10, max_features='auto',
            max_leaf_nodes=None, min_impurity_decrease=0.0,
            min_impurity_split=None, min_samples_leaf=1,
            min_samples_split=2, min_weight_fraction_leaf=0.0,
            n_estimators=1000, n_jobs=None, oob_score=False,
            random_state=None, verbose=0, warm_start=False)
```

In [124]:

```
## Train Vs Cross Validation (Prevention of Overfitting and underfitting of the model)
CompareTrainCV(RF_tfidf_Train_AUC,
RF_tfidf_CV_AUC,RF_n_est,RF_depth,RF_tfidf_best_estimator,RF_tfidf_best_depth)
```








```
The best optimal number of estimators are 1000 at best optimal depth of 10
```

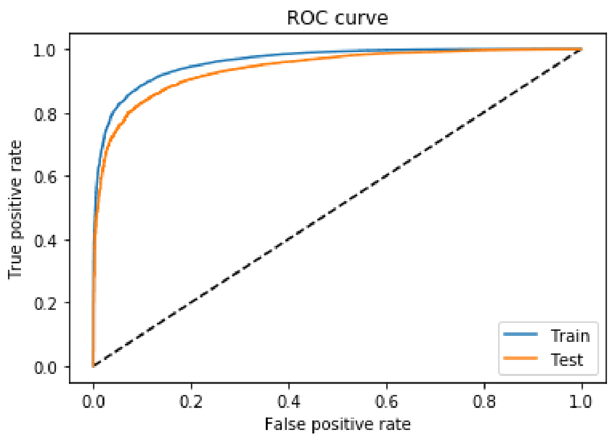
In [125]:

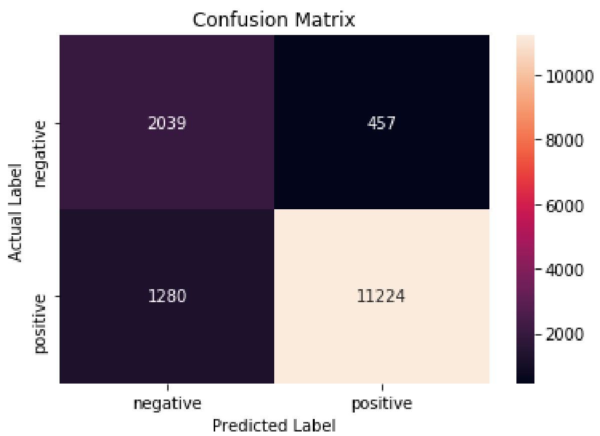
```
## Retrain the best model and test
RF_tfidf_test_score,RF_tfidf_train_fpr, RF_tfidf_train_tpr,RF_tfidf_test_fpr, RF_tfidf_test_tpr,RF
_tfidf_pred_label,RF_tfidf_model = RFTestModel(S100_tfidf_X_train,S100_Y_train,S100_tfidf_X_test,S
100_Y_test,RF_tfidf_best_estimator,RF_tfidf_best_depth)
```

```
Test AUC score = 0.9387311925568433
```

In [126]:

```
## Train Vs Test Scores performance
CompareTrainTest(S100_Y_test,RF_tfidf_test_score,RF_tfidf_train_fpr,
RF_tfidf_train_tpr,RF_tfidf_test_fpr, RF_tfidf_test_tpr,RF_tfidf_pred_label)
```





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

In [127]:

```
n = 20
top_features = []
top_feature_names = S100_tfidf_vect.get_feature_names()
coefs = sorted(zip(RF_tfidf_model.feature_importances_, top_feature_names))
top = coefs[:-(n + 1):-1]
#print('\033[1m' + "feature_importances\tfeatures" + '\033[0m')
#print("="*35)
for (coef1, feat1) in top:
    #print("%.4f\t\t\t%-15s" % (coef1, feat1))
    top_features.append(feat1)

# printing word cloud
wordcloud = WordCloud(background_color='white',width=1600,height=800).generate(" ".join(top_feature
s))
fig = plt.figure(figsize=(15,15))
plt.imshow(wordcloud)
plt.axis('off')
plt.tight_layout(pad=0)
fig.savefig("tfidf_RF_top_features.png")
plt.show()
```

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

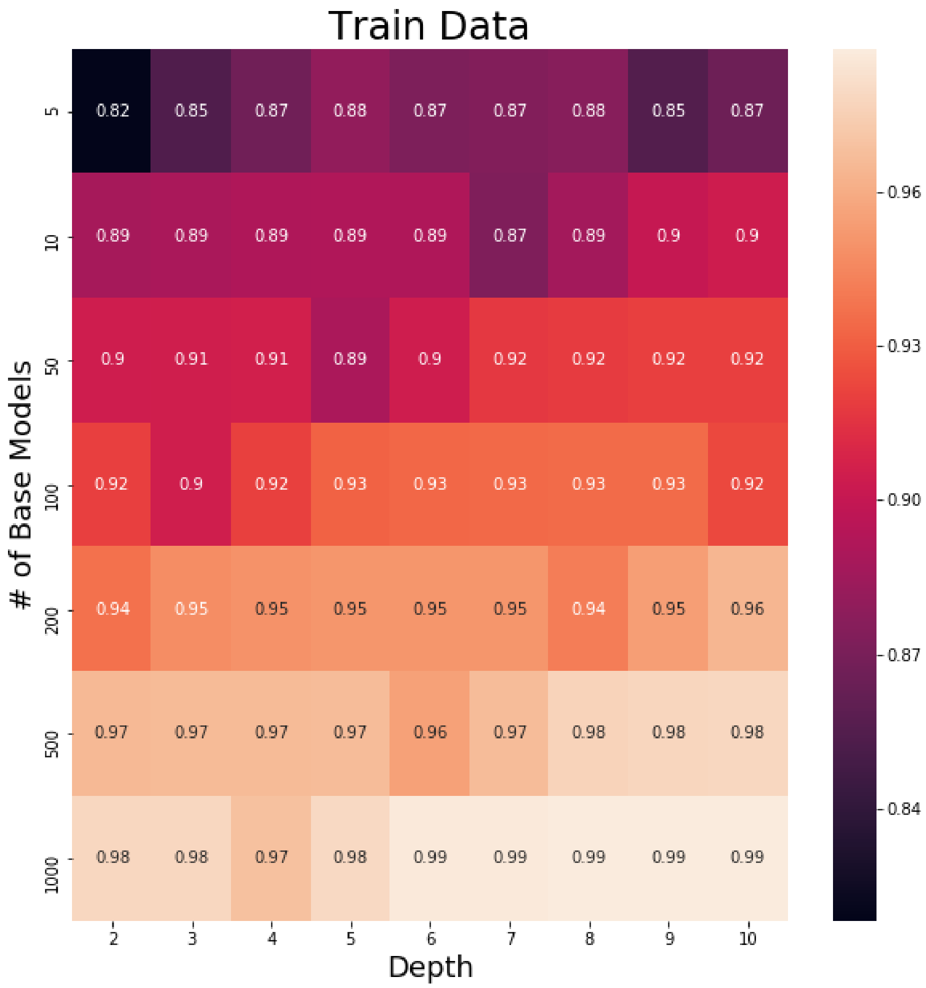
In [128]:

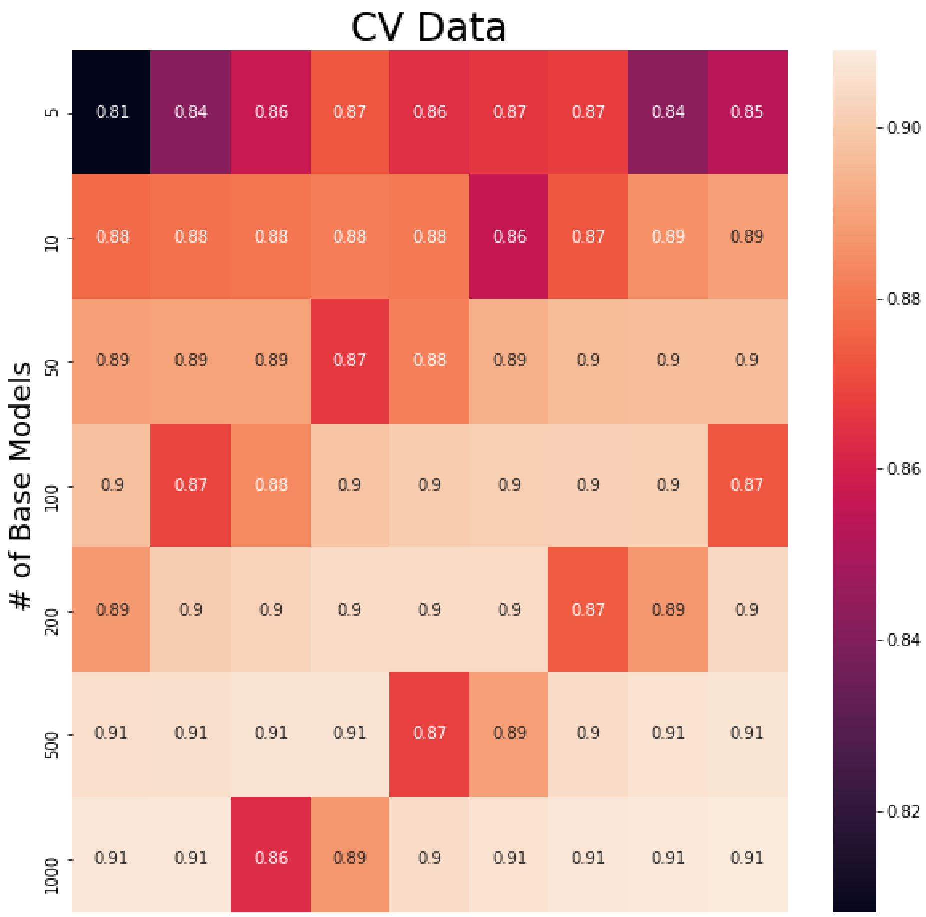
```
# Train model with cross validation
RF_W2V_Train_AUC,RF_W2V_CV_AUC,RF_W2V_best_estimator,RF_W2V_best_depth =
RFTrainModel(S100_W2V_train, S100_Y_train, RF_n_est,RF_depth)
```

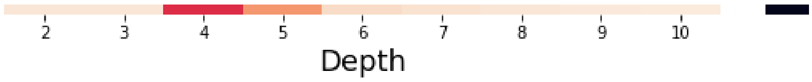
```
{'max_depth': 10, 'n_estimators': 1000}
0.9089251901329499
RandomForestClassifier(bootstrap=True, class_weight='balanced',
            criterion='gini', max_depth=10, max_features='auto',
            max_leaf_nodes=None, min_impurity_decrease=0.0,
            min_impurity_split=None, min_samples_leaf=1,
            min_samples_split=2, min_weight_fraction_leaf=0.0,
            n_estimators=1000, n_jobs=None, oob_score=False,
            random_state=None, verbose=0, warm_start=False)
```

In [129]:

```
## Train Vs Cross Validation (Prevention of Overfitting and underfitting of the model)
CompareTrainCV(RF_W2V_Train_AUC,
RF_W2V_CV_AUC,RF_n_est,RF_depth,RF_W2V_best_estimator,RF_W2V_best_depth)
```








```
The best optimal number of estimators are 1000 at best optimal depth of 10
```

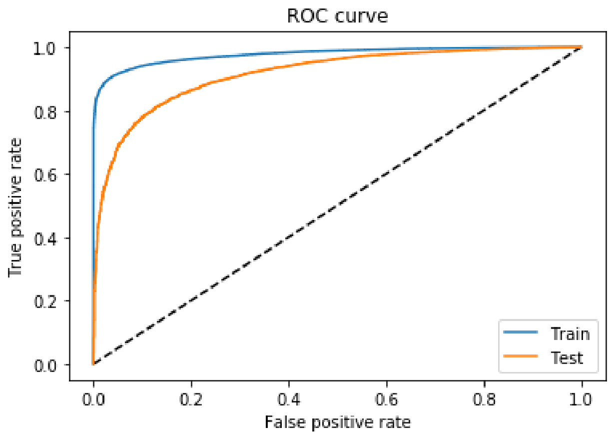
In [130]:

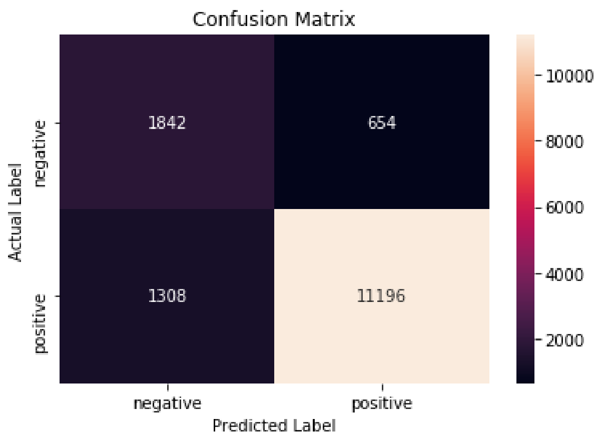
```
## Retrain the best model and test
RF_W2V_test_score,RF_W2V_train_fpr, RF_W2V_train_tpr,RF_W2V_test_fpr,
RF_W2V_test_tpr,RF_W2V_pred_label,RF_W2V_model =
RFTestModel(S100_W2V_train,S100_Y_train,S100_W2V_test,S100_Y_test,RF_W2V_best_estimator,RF_W2V_best
_depth)
```

```
Test AUC score = 0.9161853142891712
```

In [131]:

```
## Train Vs Test Scores performance
CompareTrainTest(S100_Y_test,RF_W2V_test_score,RF_W2V_train_fpr, RF_W2V_train_tpr,RF_W2V_test_fpr,
RF_W2V_test_tpr,RF_W2V_pred_label)
```





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

In [132]:

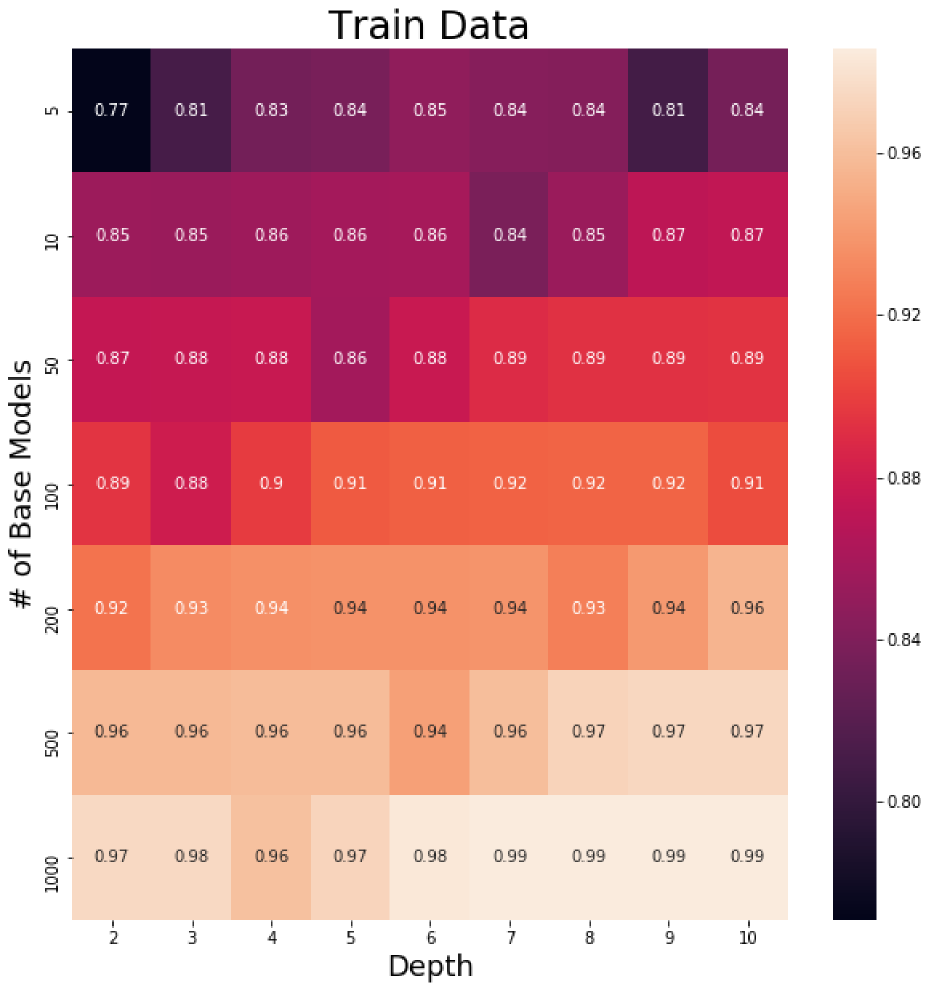
```
# Train model with cross validation
RF_tfidf_W2V_Train_AUC,RF_tfidf_W2V_CV_AUC,RF_tfidf_W2V_best_estimator,RF_tfidf_W2V_best_depth = R
FTrainModel(S100_tfidf_W2V_train, S100_Y_train, RF_n_est,RF_depth)
```

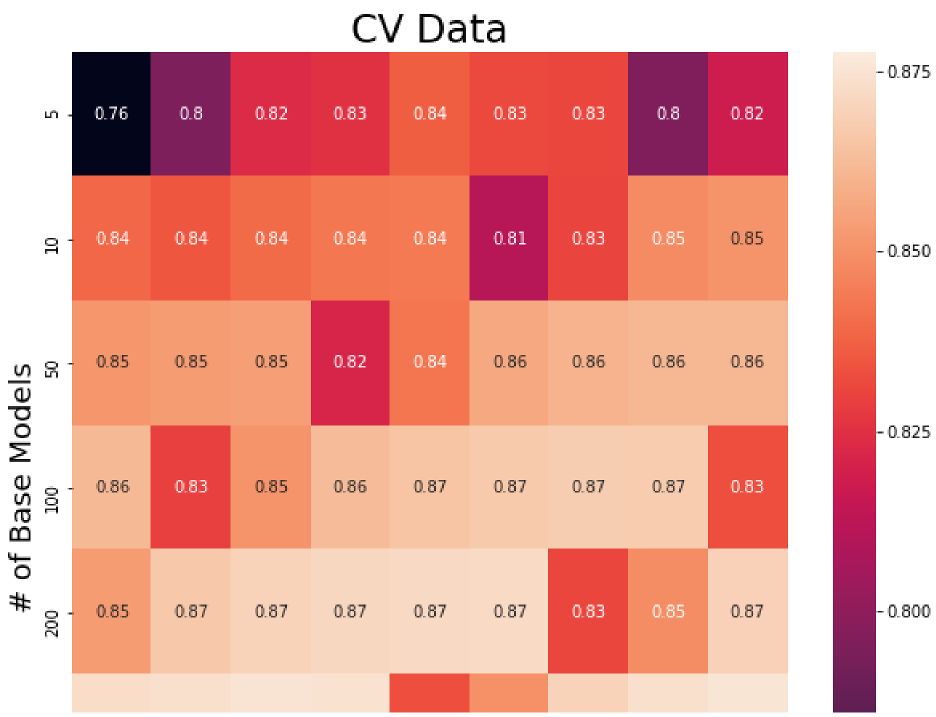
```
{'max_depth': 10, 'n_estimators': 1000}
0.8775345836386327
RandomForestClassifier(bootstrap=True, class_weight='balanced',
            criterion='gini', max_depth=10, max_features='auto',
            max_leaf_nodes=None, min_impurity_decrease=0.0,
```

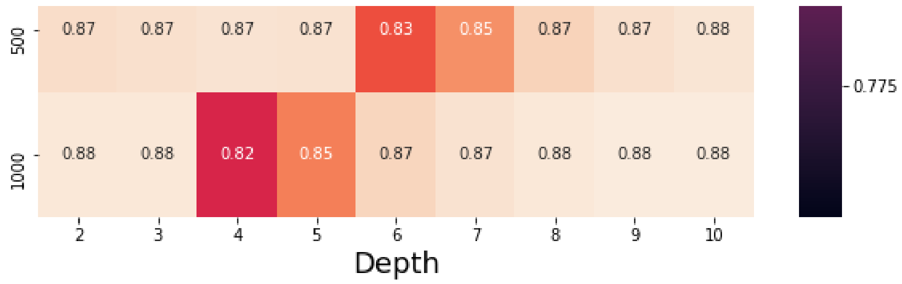
```
            min_impurity_split=None, min_samples_leaf=1,
            min_samples_split=2, min_weight_fraction_leaf=0.0,
            n_estimators=1000, n_jobs=None, oob_score=False,
            random_state=None, verbose=0, warm_start=False)
```

In [133]:

```
## Train Vs Cross Validation (Prevention of Overfitting and underfitting of the model)
CompareTrainCV(RF_tfidf_W2V_Train_AUC,
RF_tfidf_W2V_CV_AUC,RF_n_est,RF_depth,RF_tfidf_W2V_best_estimator,RF_tfidf_W2V_best_depth)
```








```
The best optimal number of estimators are 1000 at best optimal depth of 10
```

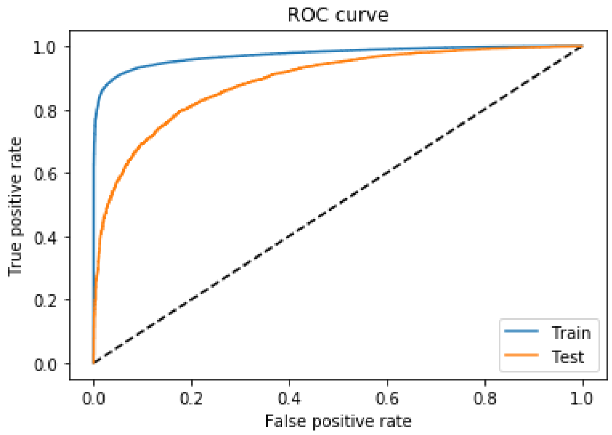
In [135]:

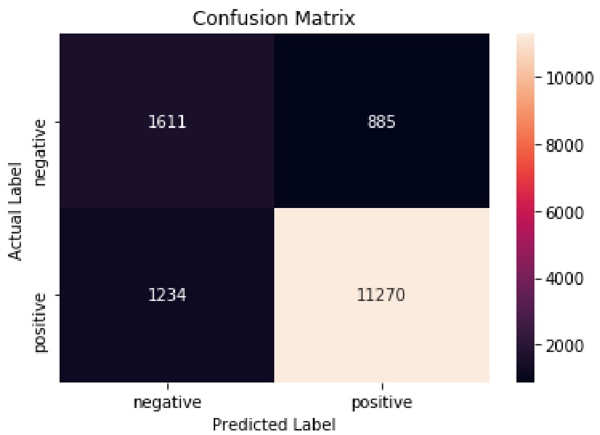
```
## Retrain the best model and test
RF_tfidf_W2V_test_score,RF_tfidf_W2V_train_fpr, RF_tfidf_W2V_train_tpr,RF_tfidf_W2V_test_fpr, RF_t
fidf_W2V_test_tpr,RF_tfidf_W2V_pred_label,RF_tfidf_W2V_model = RFTestModel(S100_tfidf_W2V_train,S1
00_Y_train,S100_tfidf_W2V_test,S100_Y_test,RF_tfidf_W2V_best_estimator,RF_tfidf_W2V_best_depth)
```

```
Test AUC score = 0.8895953294945618
```

In [136]:

```
## Train Vs Test Scores performance
CompareTrainTest(S100_Y_test,RF_tfidf_W2V_test_score,RF_tfidf_W2V_train_fpr,
RF_tfidf_W2V_train_tpr,RF_tfidf_W2V_test_fpr, RF_tfidf_W2V_test_tpr,RF_tfidf_W2V_pred_label)
```



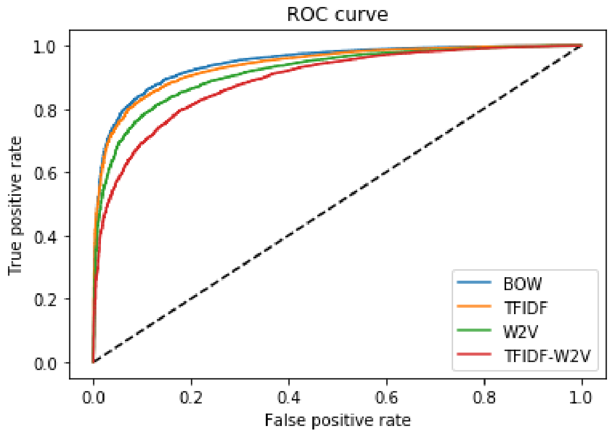


ROC Plot to compare the best versions of all Random Forest four models

In [137]:

```
plt.figure(1)
plt.plot([0, 1], [0, 1], 'k--')
plt.plot(RF_bow_test_fpr, RF_bow_test_tpr, label='BOW')
```

```
plt.plot(RF_tfidf_test_fpr, RF_tfidf_test_tpr, label='TFIDF')
plt.plot(RF_W2V_test_fpr, RF_W2V_test_tpr, label='W2V')
plt.plot(RF_tfidf_W2V_test_fpr, RF_tfidf_W2V_test_tpr, label='TFIDF-W2V')
plt.xlabel('False positive rate')
plt.ylabel('True positive rate')
plt.title('ROC curve')
plt.legend(loc='best')
plt.show()
```



[5.2] Applying GBDT using XGBOOST

[5.2.1] Applying XGBOOST on BOW, SET 1

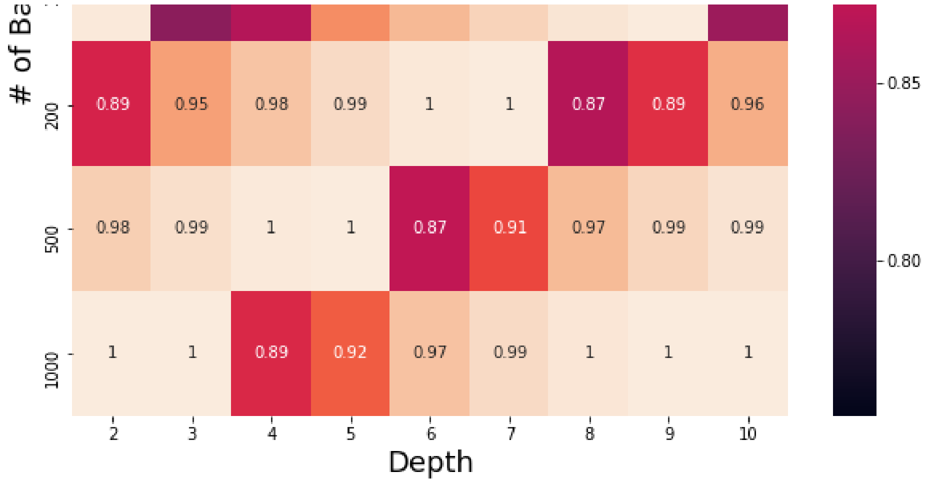
In [144]:

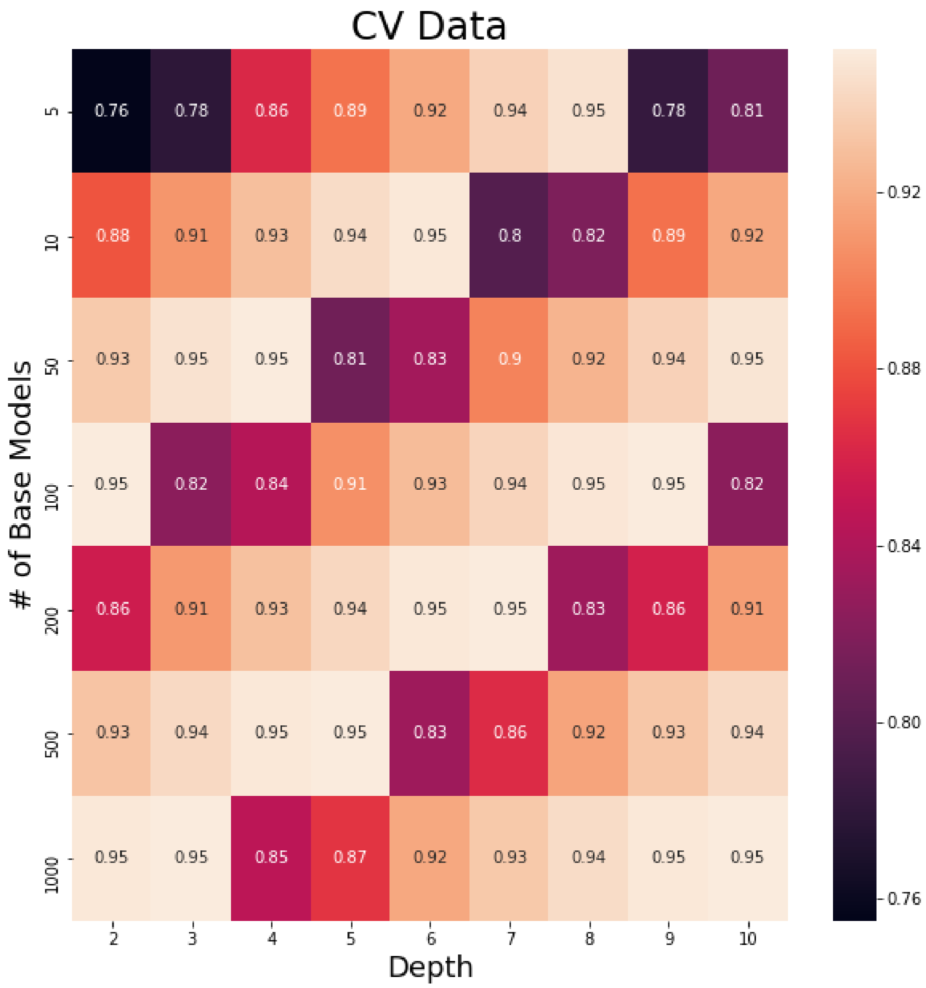
```
# Train model with cross validation
XG_bow_Train_AUC, XG_bow_CV_AUC,XG_bow_best_estimator, XG_bow_best_depth =
XGTrainModel(S100_BOW_X_train, S100_Y_train, XG_n_est,XG_depth)
```

```
{'max_depth': 6, 'n_estimators': 1000}
0.9520841964539748
XGBClassifier(base_score=0.5, booster='gbtree', colsample_bylevel=1,
       colsample_bytree=1, gamma=0, learning_rate=0.1, max_delta_step=0,
       max_depth=6, min_child_weight=1, missing=None, n_estimators=1000,
       n_jobs=1, nthread=None, objective='binary:logistic', random_state=0,
       reg_alpha=0, reg_lambda=1, scale_pos_weight=1, seed=None,
       silent=True, subsample=1)
```

In [145]:

```
## Train Vs Cross Validation (Prevention of Overfitting and underfitting of the model)
CompareTrainCV(XG_bow_Train_AUC,
XG_bow_CV_AUC,XG_n_est,XG_depth,XG_bow_best_estimator,XG_bow_best_depth)
```






```
The best optimal number of estimators are 1000 at best optimal depth of 6
```

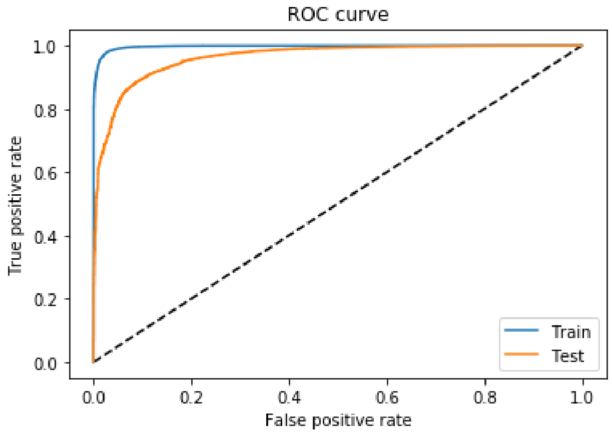
In [146]:

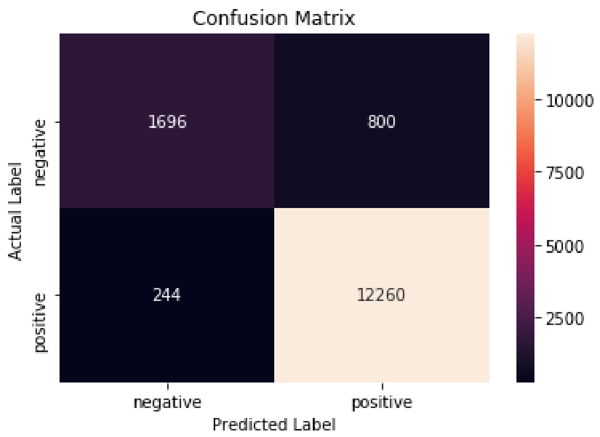
```
## Retrain the best model and test
XG_bow_test_score,XG_bow_train_fpr, XG_bow_train_tpr,XG_bow_test_fpr,
XG_bow_test_tpr,XG_bow_pred_label,XG_bow_model =
XGTestModel(S100_BOW_X_train,S100_Y_train,S100_BOW_X_test,S100_Y_test,XG_bow_best_estimator,XG_bow_
best_depth)
```

```
Test AUC score = 0.9628041783039685
```

In [147]:

```
## Train Vs Test Scores performance
CompareTrainTest(S100_Y_test,XG_bow_test_score,XG_bow_train_fpr, XG_bow_train_tpr,XG_bow_test_fpr,
XG_bow_test_tpr,XG_bow_pred_label)
```





[5.2.2] Applying XGBOOST on TFIDF, SET 2

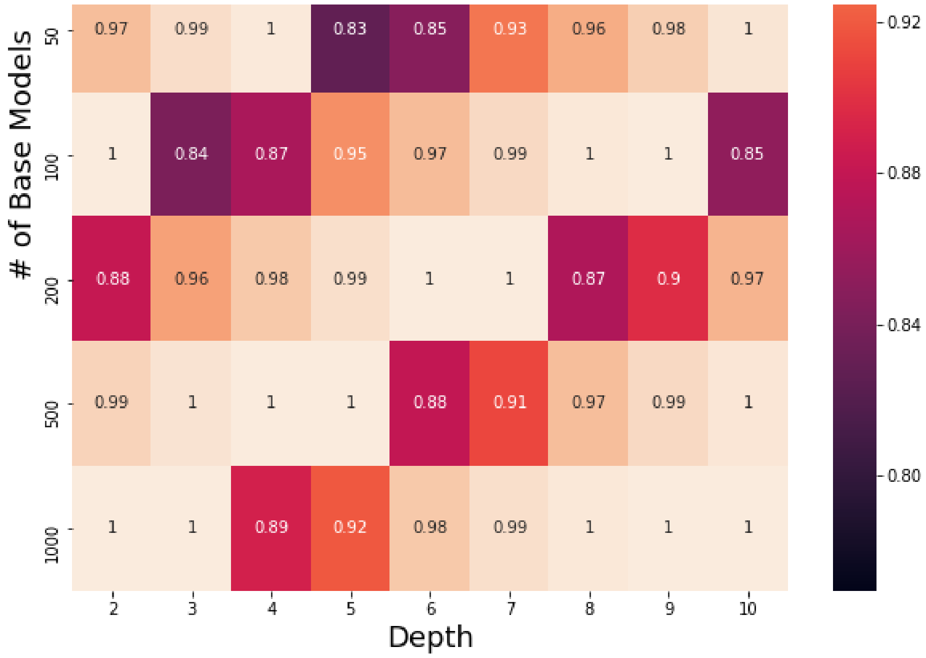
In [148]:

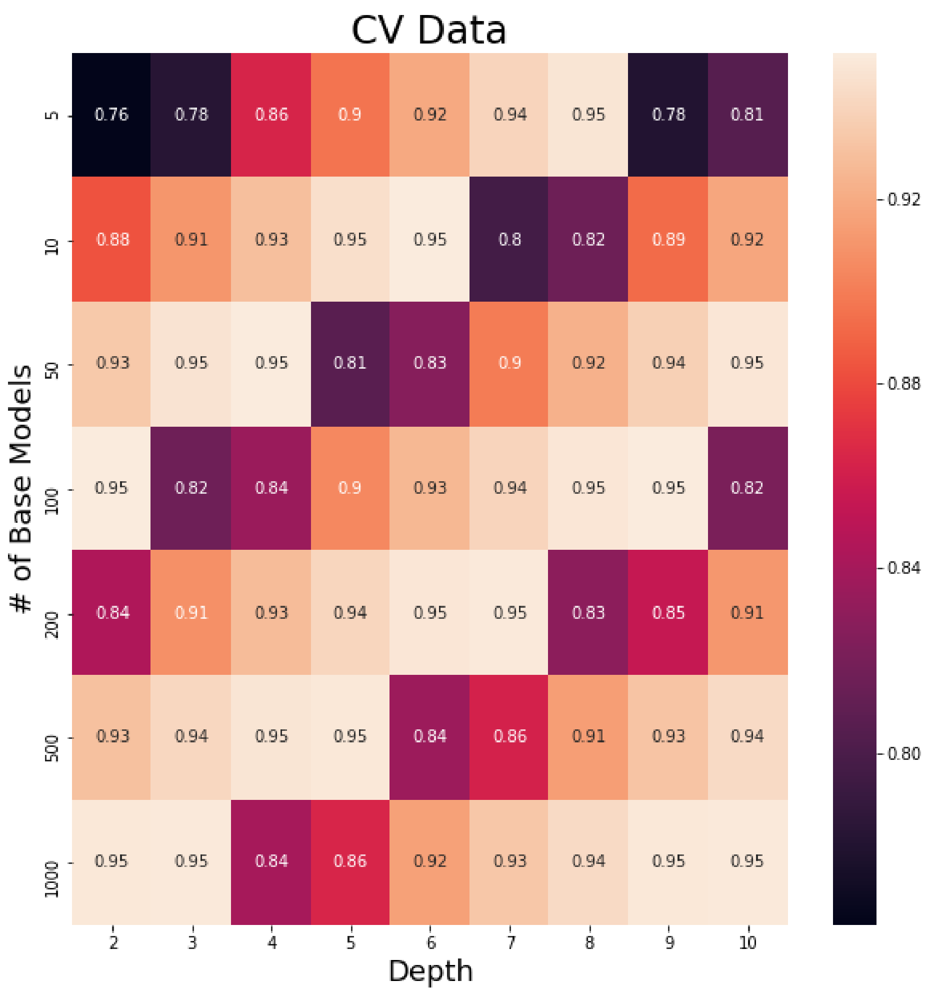
```
# Train model with cross validation
XG_tfidf_Train_AUC,XG_tfidf_CV_AUC,XG_tfidf_best_estimator,XG_tfidf_best_depth =
XGTrainModel(S100_tfidf_X_train, S100_Y_train, XG_n_est,XG_depth)
```

```
{'max_depth': 4, 'n_estimators': 1000}
0.9513209369194056
XGBClassifier(base_score=0.5, booster='gbtree', colsample_bylevel=1,
       colsample_bytree=1, gamma=0, learning_rate=0.1, max_delta_step=0,
       max_depth=4, min_child_weight=1, missing=None, n_estimators=1000,
       n_jobs=1, nthread=None, objective='binary:logistic', random_state=0,
       reg_alpha=0, reg_lambda=1, scale_pos_weight=1, seed=None,
       silent=True, subsample=1)
```

In [149]:

```
## Train Vs Cross Validation (Prevention of Overfitting and underfitting of the model)
CompareTrainCV(XG_tfidf_Train_AUC,
XG_tfidf_CV_AUC,XG_n_est,XG_depth,XG_tfidf_best_estimator,XG_tfidf_best_depth)
```






```
The best optimal number of estimators are 1000 at best optimal depth of 4
```

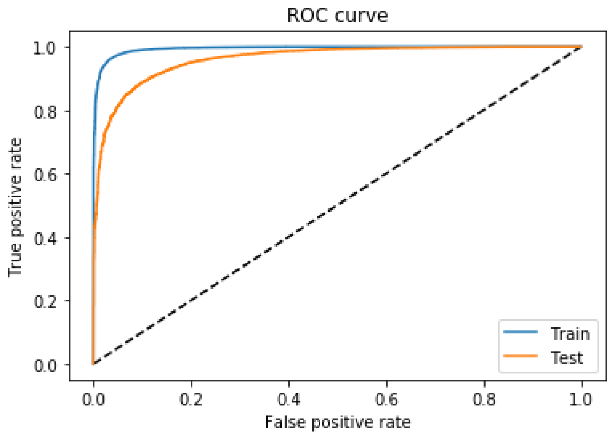
In [150]:

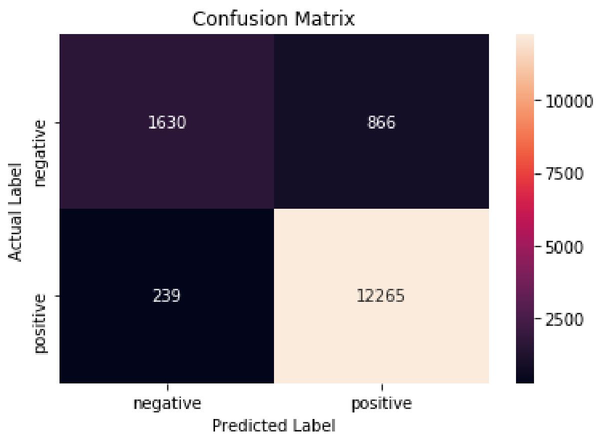
```
## Retrain the best model and test
XG_tfidf_test_score,XG_tfidf_train_fpr, XG_tfidf_train_tpr,XG_tfidf_test_fpr, XG_tfidf_test_tpr,XG_tfidf_pred_label,XG_tfidf_model = XGTestModel(S100_tfidf_X_train,S100_Y_train,S100_tfidf_X_test,S100_Y_test,XG_tfidf_best_estimator,XG_tfidf_best_depth)
```

```
Test AUC score = 0.9613300666863527
```

In [151]:

```
#Train Vs Test Scores performance
CompareTrainTest(S100_Y_test,XG_tfidf_test_score,XG_tfidf_train_fpr,
XG_tfidf_train_tpr,XG_tfidf_test_fpr, XG_tfidf_test_tpr,XG_tfidf_pred_label)
```





[5.2.3] Applying XGBOOST on AVG W2V, SET 3

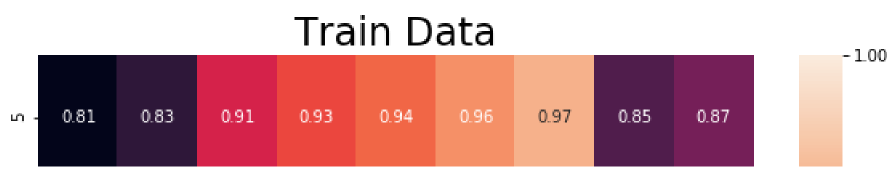
In [153]:

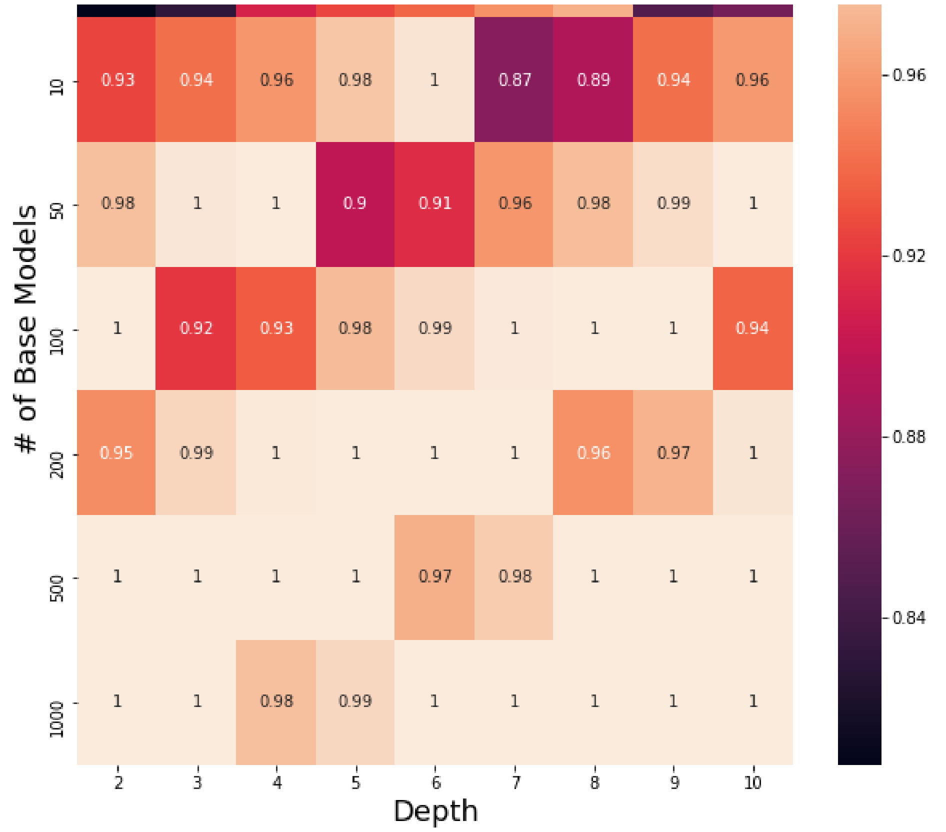
```
# Train model with cross validation
#W2V_train = np.array(S100_W2V_train)
XG_W2V_Train_AUC,XG_W2V_CV_AUC,XG_W2V_best_estimator,XG_W2V_best_depth =
XGTrainModel(S100_W2V_train, S100_Y_train, XG_n_est,XG_depth)
```

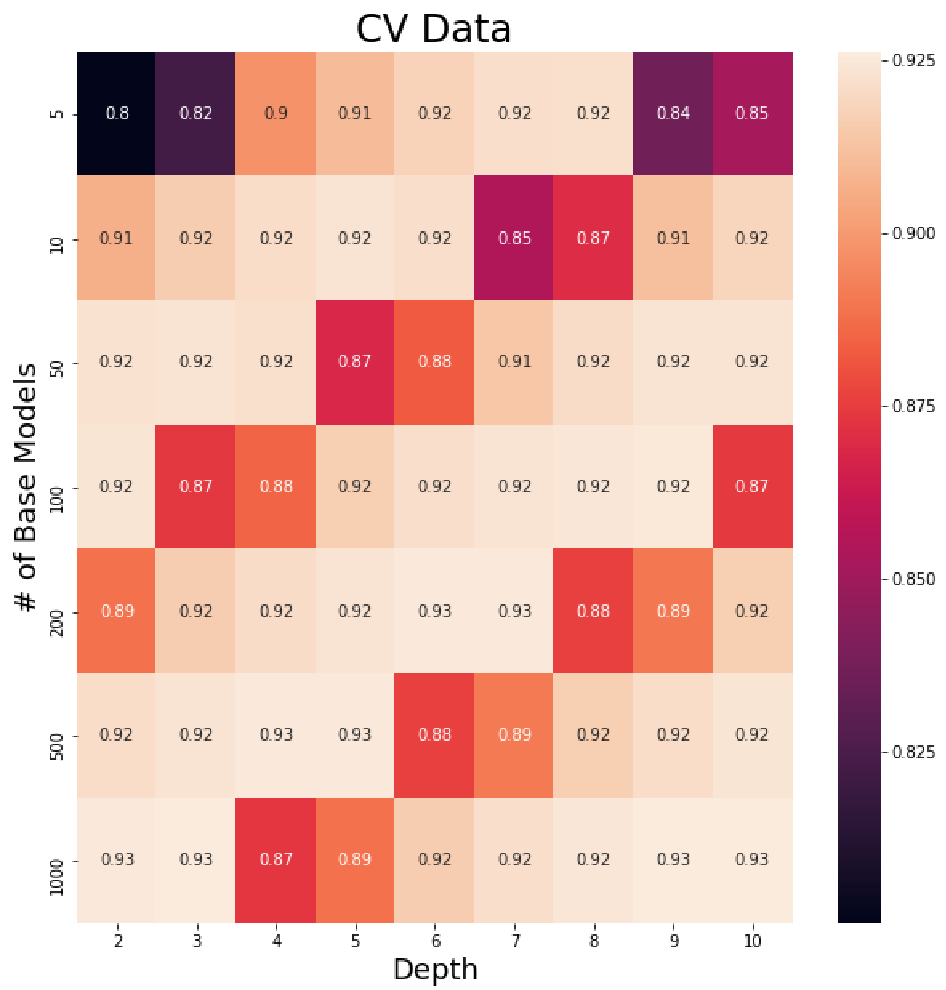
```
{'max_depth': 10, 'n_estimators': 1000}
0.9261247140282896
XGBClassifier(base_score=0.5, booster='gbtree', colsample_bylevel=1,
       colsample_bytree=1, gamma=0, learning_rate=0.1, max_delta_step=0,
       max_depth=10, min_child_weight=1, missing=None, n_estimators=1000,
       n_jobs=1, nthread=None, objective='binary:logistic', random_state=0,
       reg_alpha=0, reg_lambda=1, scale_pos_weight=1, seed=None,
       silent=True, subsample=1)
```

In [154]:

```
## Train Vs Cross Validation (Prevention of Overfitting and underfitting of the model)
CompareTrainCV(XG_W2V_Train_AUC,
XG_W2V_CV_AUC,XG_n_est,XG_depth,XG_W2V_best_estimator,XG_W2V_best_depth)
```








```
The best optimal number of estimators are 1000 at best optimal depth of 10
```

In [155]:

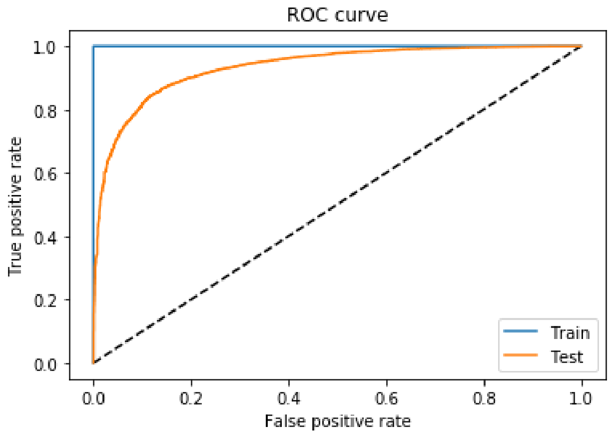
```
## Retrain the best model and test
#W2V_test = np.array(S100_W2V_test)
```

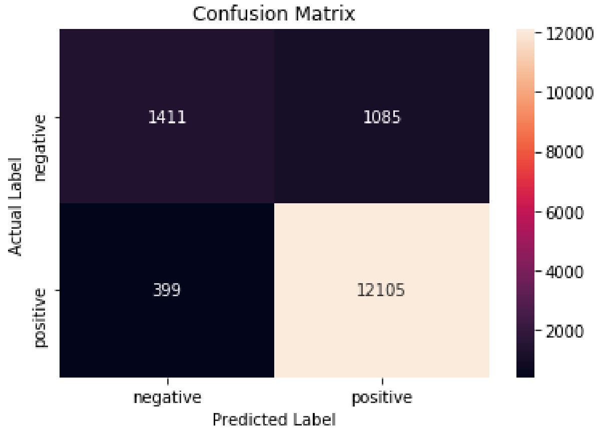
```
XG_W2V_test_score,XG_W2V_train_fpr, XG_W2V_train_tpr,XG_W2V_test_fpr,
XG_W2V_test_tpr,XG_W2V_pred_label,XG_W2V_model =
XGTestModel(S100_W2V_train,S100_Y_train,S100_W2V_test,S100_Y_test,XG_W2V_best_estimator,XG_W2V_best
_depth)
```

```
Test AUC score = 0.9336270726700789
```

In [156]:

```
## Train Vs Test Scores performance
CompareTrainTest(S100_Y_test,XG_W2V_test_score,XG_W2V_train_fpr, XG_W2V_train_tpr,XG_W2V_test_fpr,
XG_W2V_test_tpr,XG_W2V_pred_label)
```





[5.2.4] Applying XGBOOST on TFIDF W2V, SET 4

In [157]:

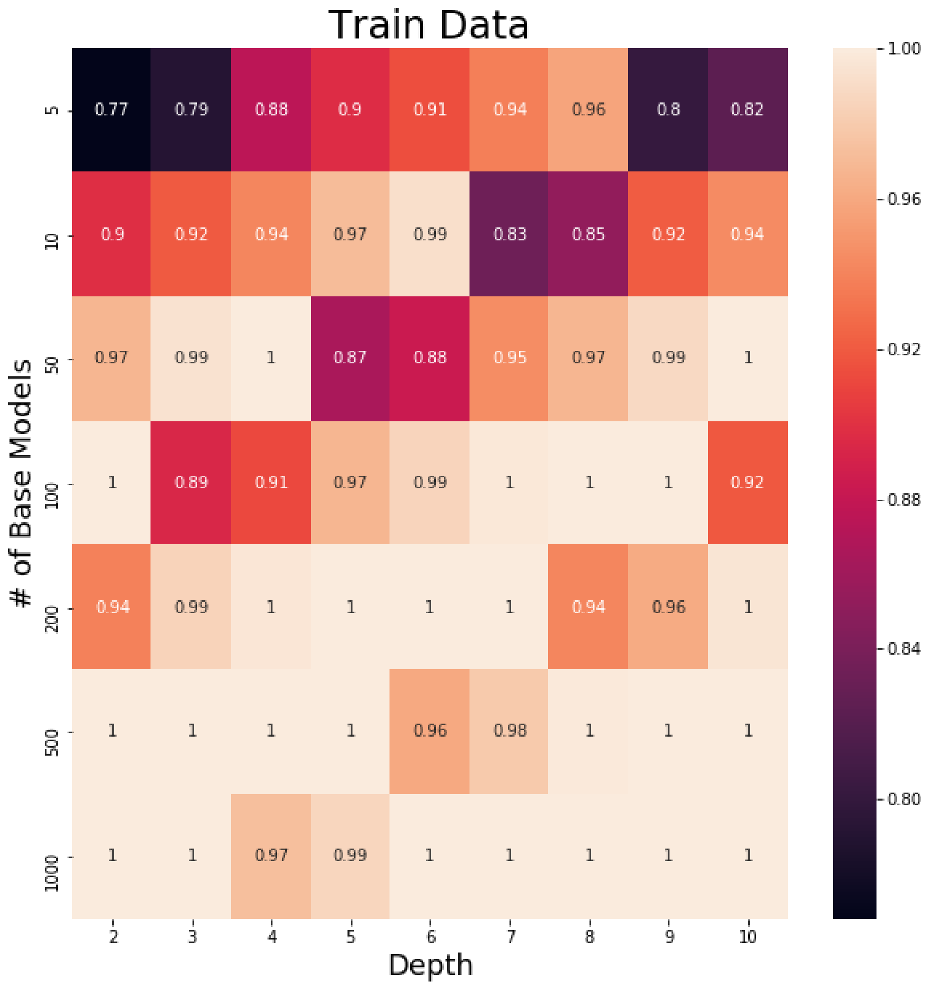
```
# Train model with cross validation
#tfidf_W2V_train = np.array(S100_tfidf_W2V_train)
XG_tfidf_W2V_Train_AUC,XG_tfidf_W2V_CV_AUC,XG_tfidf_W2V_best_estimator,XG_tfidf_W2V_best_depth = X
GTrainModel(S100_tfidf_W2V_train, S100_Y_train, XG_n_est,XG_depth)
```

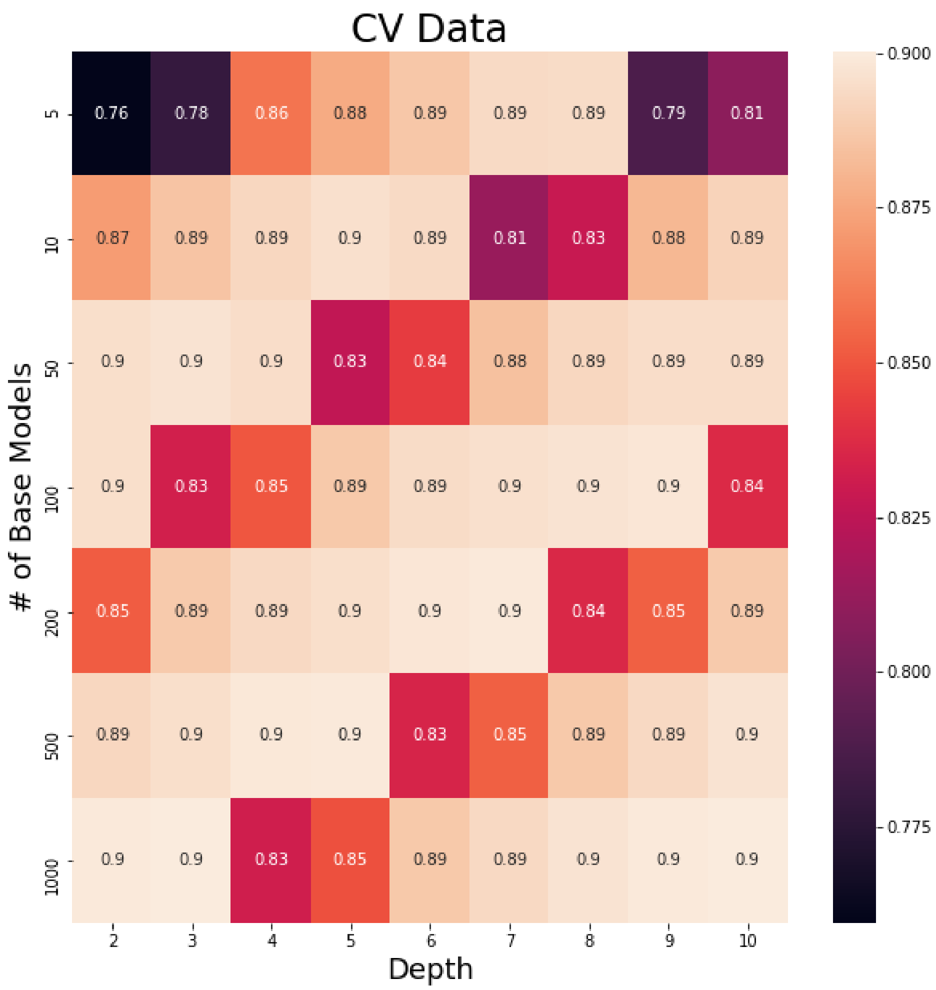
```
{'max_depth': 9, 'n_estimators': 1000}
0.9001146649614619
XGBClassifier(base_score=0.5, booster='gbtree', colsample_bylevel=1,
       colsample_bytree=1, gamma=0, learning_rate=0.1, max_delta_step=0,
       max_depth=9, min_child_weight=1, missing=None, n_estimators=1000,
       n_jobs=1, nthread=None, objective='binary:logistic', random_state=0,
       reg_alpha=0, reg_lambda=1, scale_pos_weight=1, seed=None,
       silent=True, subsample=1)
```

In [158]:

```
## Train Vs Cross Validation (Prevention of Overfitting and underfitting of the model)
CompareTrainCV(XG_tfidf_W2V_Train_AUC,
```

```
XG_tfidf_W2V_CV_AUC,XG_n_est,XG_depth,XG_tfidf_W2V_best_estimator,XG_tfidf_W2V_best_depth)
```





```
The best optimal number of estimators are 1000 at best optimal depth of 9
```

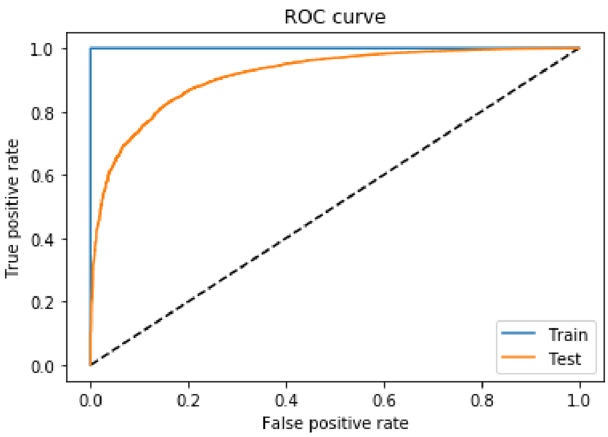
In [159]:

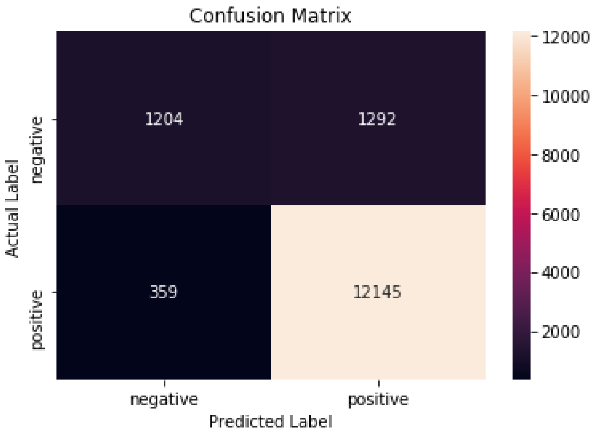
```
## Retrain the best model and test
#tfidf_W2V_test = np.array(S100_tfidf_W2V_test)
XG_tfidf_W2V_test_score,XG_tfidf_W2V_train_fpr, XG_tfidf_W2V_train_tpr,XG_tfidf_W2V_test_fpr, XG_t
fidf_W2V_test_tpr,XG_tfidf_W2V_pred_label,XG_tfidf_W2V_model = XGTestModel(S100_tfidf_W2V_train,S1
00_Y_train,S100_tfidf_W2V_test,S100_Y_test,XG_tfidf_W2V_best_estimator,XG_tfidf_W2V_best_depth)
```

```
Test AUC score = 0.9149581108404284
```

In [160]:

```
## Train Vs Test Scores performance
CompareTrainTest(S100_Y_test,XG_tfidf_W2V_test_score,XG_tfidf_W2V_train_fpr,
XG_tfidf_W2V_train_tpr,XG_tfidf_W2V_test_fpr, XG_tfidf_W2V_test_tpr,XG_tfidf_W2V_pred_label)
```

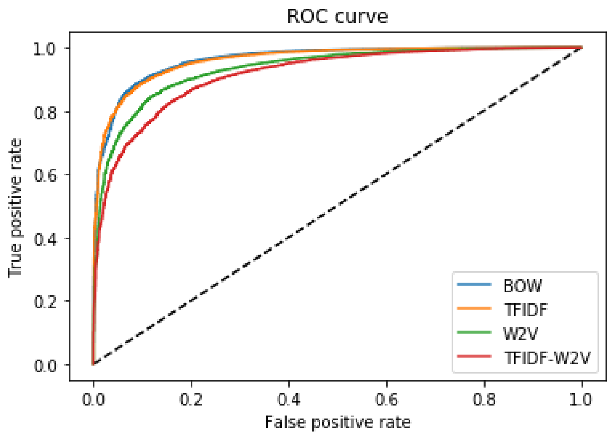




ROC Plot to compare the best versions of all XGBoost four models

In [161]:

```
plt.figure(1)
plt.plot([0, 1], [0, 1], 'k--')
plt.plot(XG_bow_test_fpr, XG_bow_test_tpr, label='BOW')
plt.plot(XG_tfidf_test_fpr, XG_tfidf_test_tpr, label='TFIDF')
plt.plot(XG_W2V_test_fpr, XG_W2V_test_tpr, label='W2V')
plt.plot(XG_tfidf_W2V_test_fpr, XG_tfidf_W2V_test_tpr, label='TFIDF-W2V')
plt.xlabel('False positive rate')
plt.ylabel('True positive rate')
plt.title('ROC curve')
plt.legend(loc='best')
plt.show()
```



[6] Conclusions

In [162]:

```
all_test_auc  = []
all_test_auc.append('%2f' % RF_bow_test_score )
all_test_auc.append('%2f' % XG_bow_test_score )
all_test_auc.append('%2f' % RF_tfidf_test_score )
all_test_auc.append('%2f' % XG_tfidf_test_score )
all_test_auc.append('%2f' % RF_W2V_test_score )
all_test_auc.append('%2f' % XG_W2V_test_score )
all_test_auc.append('%2f' % RF_tfidf_W2V_test_score )
all_test_auc.append('%2f' % XG_tfidf_W2V_test_score )
print(all_test_auc)
```

```
['0.945855', '0.962804', '0.938731', '0.961330', '0.916185', '0.933627', '0.889595', '0.914958']
```

In [163]:

```
all_best_depths = []
all_best_depths.append(RF_bow_best_depth)
all_best_depths.append(XG_bow_best_depth)
all_best_depths.append(RF_tfidf_best_depth)
all_best_depths.append(XG_tfidf_best_depth)
all_best_depths.append(RF_W2V_best_depth)
all_best_depths.append(XG_W2V_best_depth)
all_best_depths.append(RF_tfidf_W2V_best_depth)
all_best_depths.append(XG_tfidf_W2V_best_depth)
```

In [164]:

```
all_best_estimators = []
all_best_estimators.append(RF_bow_best_estimator)
all_best_estimators.append(XG_bow_best_estimator)
all_best_estimators.append(RF_tfidf_best_estimator)
all_best_estimators.append(XG_tfidf_best_estimator)
all_best_estimators.append(RF_W2V_best_estimator)
all_best_estimators.append(XG_W2V_best_estimator)
all_best_estimators.append(RF_tfidf_W2V_best_estimator)
all_best_estimators.append(XG_tfidf_W2V_best_estimator)
```

In [167]:

```
print("Summary of Results from Models W/O Feature Engineering")
from prettytable import PrettyTable
from prettytable import from_csv
with open("WithoutFE.csv", "r") as fp:
    y = from_csv(fp)

print(y)
```

```
Summary of Results from Models W/O Feature Engineering
```

```
+------------+--------------+-------+-------+----------------+
| Vectorizer |     Model    | Depth | Split | Test AUC Score |
+------------+--------------+-------+-------+----------------+
|    BOW     | Random Forest |  1000 |  100  |    0.911212    |
|    BOW     |    XGBoost   |   8   |  500  |    0.939093    |
|   TF-IDF   | Random Forest |  100  |  100  |    0.917859    |
|   TF-IDF   |    XGBoost   |   8   |  500  |    0.941874    |
|    W2V     | Random Forest |   10  |  100  |    0.890502    |
|    W2V     |    XGBoost   |   10  |  500  |    0.918853    |
| TFIDF-W2V  | Random Forest |  1000 |  100  |    0.86844     |
| TFIDF-W2V  |    XGBoost   |   10  |  500  |    0.899244    |
+------------+--------------+-------+-------+----------------+
```

In [168]:

```python
print("Summary of Results from Models WITH Feature Engineering")
Vectorizer = ['BOW','BOW','TF-IDF','TF-IDF','W2V','W2V','TFIDF-W2V','TFIDF-W2V']
Model = ['Random Forest','XGBoost','Random Forest','XGBoost','Random Forest','XGBoost','Random For
est','XGBoost']



x = PrettyTable(['Vectorizer', 'Model', 'Depth','Split','Test AUC Score'])
for i in range(8):
    x.add_row([Vectorizer[i], Model[i], all_best_depths[i], all_best_estimators[i], all_test_auc[i]
])


print(x)
```

```
Summary of Results from Models WITH Feature Engineering
+------------+--------------+-------+-------+----------------+
| Vectorizer |     Model    | Depth | Split | Test AUC Score |
+------------+--------------+-------+-------+----------------+
|    BOW     | Random Forest |   10  |  1000 |    0.945855    |
|    BOW     |    XGBoost   |   6   |  1000 |    0.962804    |
|   TF-IDF   | Random Forest |   10  |  1000 |    0.938731    |
|   TF-IDF   |    XGBoost   |   4   |  1000 |    0.961330    |
|    W2V     | Random Forest |   10  |  1000 |    0.916185    |
|    W2V     |    XGBoost   |   10  |  1000 |    0.933627    |
| TFIDF-W2V  | Random Forest |   10  |  1000 |    0.889595    |
| TFIDF-W2V  |    XGBoost   |   9   |  1000 |    0.914958    |
+------------+--------------+-------+-------+----------------+
```

Observation:

After considering Review Summay and Review text, the performance of all the models is boosted by more than 2%, which is a significant difference. Hence it is worth to add these features for this dataset analysis/modeling.