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 [0]:

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


import sqlite3
import pandas as pd
import numpy as np
import nltk
import string
```

```python
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 [0]:

```python
# 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 50
0000 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 Sco
re != 3 LIMIT 500000""", con)
# for tsne assignment you can take 5k data points
```

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

# Give reviews with Score>3 a positive rating(1), and reviews with a sc
ore<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 (5000, 10)
```

Out[0]:

	Id	ProductId	UserId	ProfileName	HelpfulnessNumerator	Helpfulnes
0	1	B001E4KFG0	A3SGXH7AUHU8GW	delmartian	1	1
1	2	B00813GRG4	A1D87F6ZCVE5NK	dll pa	0	0

	Id	ProductId	UserId	ProfileName	HelpfulnessNumerator	Helpfulnes
2	3	B000LQOCH0	ABXLMWJIXXAIN	Natalia Corres "Natalia Corres"	1	1

In [0]:
```
display = pd.read_sql_query("""
SELECT UserId, ProductId, ProfileName, Time, Score, Text, COUNT(*)
FROM Reviews
GROUP BY UserId
HAVING COUNT(*)>1
""", con)
```

In [0]:
```
print(display.shape)
display.head()
```

(80668, 7)

Out[0]:

	UserId	ProductId	ProfileName	Time	Score	Text	COU
0	#oc-R115TNMSPFT9I7	B007Y59HVM	Breyton	1331510400	2	Overall its just OK when considering the price...	2
1	#oc-R11D9D7SHXIJB9	B005HG9ET0	Louis E. Emory "hoppy"	1342396800	5	My wife has recurring extreme muscle spasms, u...	3

	UserId	ProductId	ProfileName	Time	Score	Text	COU
2	#oc-R11DNU2NBKQ23Z	B007Y59HVM	Kim Cieszykowski	1348531200	1	This coffee is horrible and unfortunately not ...	2
3	#oc-R11O5J5ZVQE25C	B005HG9ET0	Penguin Chick	1346889600	5	This will be the bottle that you grab from the...	3
4	#oc-R12KPBODL2B5ZD	B007OSBE1U	Christopher P. Presta	1348617600	1	I didnt like this coffee. Instead of telling y...	2

In [0]:
```
display[display['UserId']=='AZY10LLTJ71NX']
```

Out[0]:

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

In [0]:
```
display['COUNT(*)'].sum()
```

Out[0]: 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 [0]:

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

Out[0]:

	Id	ProductId	UserId	ProfileName	HelpfulnessNumerator	Helpfuln
0	78445	B000HDL1RQ	AR5J8UI46CURR	Geetha Krishnan	2	2
1	138317	B000HDOPYC	AR5J8UI46CURR	Geetha Krishnan	2	2

	Id	ProductId	UserId	ProfileName	HelpfulnessNumerator	Helpfuln
2	138277	B000HDOPYM	AR5J8UI46CURR	Geetha Krishnan	2	2
3	73791	B000HDOPZG	AR5J8UI46CURR	Geetha Krishnan	2	2
4	155049	B000PAQ75C	AR5J8UI46CURR	Geetha Krishnan	2	2

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

In [0]:
```
#Deduplication of entries
final=sorted_data.drop_duplicates(subset={"UserId","ProfileName","Time","Text"}, keep='first', inplace=False)
final.shape
```

Out[0]:
```
(4986, 10)
```

In [0]:
```
#Checking to see how much % of data still remains
(final['Id'].size*1.0)/(filtered_data['Id'].size*1.0)*100
```

Out[0]:
```
99.72
```

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 [0]:
```
display= pd.read_sql_query("""
SELECT *
FROM Reviews
WHERE Score != 3 AND Id=44737 OR Id=64422
ORDER BY ProductID
""", con)

display.head()
```

Out[0]:

	Id	ProductId	UserId	ProfileName	HelpfulnessNumerator	Helpfuln
0	64422	B000MIDROQ	A161DK06JJMCYF	J. E. Stephens "Jeanne"	3	1
1	44737	B001EQ55RW	A2V0I904FH7ABY	Ram	3	2

In [0]:

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

In [0]:

```
#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()
```

```
(4986, 10)
```

Out[0]:

```
1    4178
0     808
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 [0]:

```
# 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)
```

```
Why is this $[...] when the same product is available for $[...] here?<
br />http://www.amazon.com/VICTOR-FLY-MAGNET-BAIT-REFILL/dp/B00004RBDY<
```

```
br /><br />The Victor M380 and M502 traps are unreal, of course -- tota
l fly genocide. Pretty stinky, but only right nearby.
==================================================
I recently tried this flavor/brand and was surprised at how delicious t
hese chips are.  The best thing was that there were a lot of "brown" ch
ips in the bsg (my favorite), so I bought some more through amazon and 
shared with family and friends.  I am a little disappointed that there 
are not, so far, very many brown chips in these bags, but the flavor is
still very good.  I like them better than the yogurt and green onion fl
avor because they do not seem to be as salty, and the onion flavor is b
etter.  If you haven't eaten Kettle chips before, I recommend that you 
try a bag before buying bulk.  They are thicker and crunchier than Lays
but just as fresh out of the bag.
==================================================
Wow.  So far, two two-star reviews.  One obviously had no idea what the
y were ordering; the other wants crispy cookies.  Hey, I'm sorry; but t
hese reviews do nobody any good beyond reminding us to look  before ord
ering.<br /><br />These are chocolate-oatmeal cookies.  If you don't li
ke that combination, don't order this type of cookie.  I find the combo
quite nice, really.  The oatmeal sort of "calms" the rich chocolate fla
vor and gives the cookie sort of a coconut-type consistency.  Now let's
also remember that tastes differ; so, I've given my opinion.<br /><br /
>Then, these are soft, chewy cookies -- as advertised.  They are not "c
rispy" cookies, or the blurb would say "crispy," rather than "chewy." 
I happen to like raw cookie dough; however, I don't see where these tas
te like raw cookie dough.  Both are soft, however, so is this the confu
sion?  And, yes, they stick together.  Soft cookies tend to do that.  T
hey aren't individually wrapped, which would add to the cost.  Oh yeah,
chocolate chip cookies tend to be somewhat sweet.<br /><br />So, if you
want something hard and crisp, I suggest Nabiso's Ginger Snaps.  If you
want a cookie that's soft, chewy and tastes like a combination of choco
late and oatmeal, give these a try.  I'm here to place my second order.
==================================================
love to order my coffee on amazon.  easy and shows up quickly.<br />Thi
s k cup is great coffee.  dcaf is very good as well
==================================================
```

In [0]:
```
# remove urls from text python: https://stackoverflow.com/a/40823105/40
84039
```

```
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)
```

```
Why is this $[...] when the same product is available for $[...] here?<
br /> /><br />The Victor M380 and M502 traps are unreal, of course -- t
otal fly genocide. Pretty stinky, but only right nearby.
```

In [0]:

```
# 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)
```

```
Why is this $[...] when the same product is available for $[...] here?
/>The Victor M380 and M502 traps are unreal, of course -- total fly gen
ocide. Pretty stinky, but only right nearby.
==================================================
I recently tried this flavor/brand and was surprised at how delicious t
hese chips are.  The best thing was that there were a lot of "brown" ch
```

```
hese chips are.  The best thing was that there were a lot of brown ch
ips in the bsg (my favorite), so I bought some more through amazon and
shared with family and friends.  I am a little disappointed that there
are not, so far, very many brown chips in these bags, but the flavor is
still very good.  I like them better than the yogurt and green onion fl
avor because they do not seem to be as salty, and the onion flavor is b
etter.  If you haven't eaten Kettle chips before, I recommend that you
try a bag before buying bulk.  They are thicker and crunchier than Lays
but just as fresh out of the bag.
==================================================
Wow.  So far, two two-star reviews.  One obviously had no idea what the
y were ordering; the other wants crispy cookies.  Hey, I'm sorry; but t
hese reviews do nobody any good beyond reminding us to look  before ord
ering.These are chocolate-oatmeal cookies.  If you don't like that comb
ination, don't order this type of cookie.  I find the combo quite nice,
really.  The oatmeal sort of "calms" the rich chocolate flavor and give
s the cookie sort of a coconut-type consistency.  Now let's also rememb
er that tastes differ; so, I've given my opinion.Then, these are soft,
chewy cookies -- as advertised.  They are not "crispy" cookies, or the
blurb would say "crispy," rather than "chewy."  I happen to like raw co
okie dough; however, I don't see where these taste like raw cookie doug
h.  Both are soft, however, so is this the confusion?  And, yes, they s
tick together.  Soft cookies tend to do that.  They aren't individually
wrapped, which would add to the cost.  Oh yeah, chocolate chip cookies
tend to be somewhat sweet.So, if you want something hard and crisp, I s
uggest Nabiso's Ginger Snaps.  If you want a cookie that's soft, chewy
and tastes like a combination of chocolate and oatmeal, give these a tr
y.  I'm here to place my second order.
==================================================
love to order my coffee on amazon.  easy and shows up quickly.This k cu
p is great coffee.  dcaf is very good as well
```

In [0]:

```python
# 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)
```

```python
    # 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 [0]:
```python
sent_1500 = decontracted(sent_1500)
print(sent_1500)
print("="*50)
```

```
Wow.  So far, two two-star reviews.  One obviously had no idea what the
y were ordering; the other wants crispy cookies.  Hey, I am sorry; but
these reviews do nobody any good beyond reminding us to look  before or
dering.<br /><br />These are chocolate-oatmeal cookies.  If you do not
like that combination, do not order this type of cookie.  I find the co
mbo quite nice, really.  The oatmeal sort of "calms" the rich chocolate
flavor and gives the cookie sort of a coconut-type consistency.  Now le
t is also remember that tastes differ; so, I have given my opinion.<br
/><br />Then, these are soft, chewy cookies -- as advertised.  They are
not "crispy" cookies, or the blurb would say "crispy," rather than "che
wy."  I happen to like raw cookie dough; however, I do not see where th
ese taste like raw cookie dough.  Both are soft, however, so is this th
e confusion?  And, yes, they stick together.  Soft cookies tend to do t
hat.  They are not individually wrapped, which would add to the cost.
Oh yeah, chocolate chip cookies tend to be somewhat sweet.<br /><br />S
o, if you want something hard and crisp, I suggest Nabiso is Ginger Sna
ps.  If you want a cookie that is soft, chewy and tastes like a combina
tion of chocolate and oatmeal, give these a try.  I am here to place my
second order.
==================================================
```

In [0]:
```python
#remove words with numbers python: https://stackoverflow.com/a/1808237
0/4084039
```

```
sent_0 = re.sub("\S*\d\S*", "", sent_0).strip()
print(sent_0)
```

```
Why is this $[...] when the same product is available for $[...] here?<
br /> /><br />The Victor  and  traps are unreal, of course -- total fly
genocide. Pretty stinky, but only right nearby.
```

In [0]:
```
#remove spacial character: https://stackoverflow.com/a/5843547/4084039
sent_1500 = re.sub('[^A-Za-z0-9]+', ' ', sent_1500)
print(sent_1500)
```

```
Wow So far two two star reviews One obviously had no idea what they wer
e ordering the other wants crispy cookies Hey I am sorry but these revi
ews do nobody any good beyond reminding us to look before ordering br b
r These are chocolate oatmeal cookies If you do not like that combinati
on do not order this type of cookie I find the combo quite nice really
The oatmeal sort of calms the rich chocolate flavor and gives the cooki
e sort of a coconut type consistency Now let is also remember that tast
es differ so I have given my opinion br br Then these are soft chewy co
okies as advertised They are not crispy cookies or the blurb would say
crispy rather than chewy I happen to like raw cookie dough however I do
not see where these taste like raw cookie dough Both are soft however s
o is this the confusion And yes they stick together Soft cookies tend t
o do that They are not individually wrapped which would add to the cost
Oh yeah chocolate chip cookies tend to be somewhat sweet br br So if yo
u want something hard and crisp I suggest Nabiso is Ginger Snaps If you
want a cookie that is soft chewy and tastes like a combination of choco
late and oatmeal give these a try I am here to place my second order
```

In [0]:
```
# https://gist.github.com/sebleier/554280
# we are removing the words from the stop words list: 'no', 'nor', 'no
t'
# <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', 'o
urs', 'ourselves', 'you', "you're", "you've",\
```

```
            "you'll", "you'd", 'your', 'yours', 'yourself', 'yourselve
s', 'he', 'him', 'his', 'himself', \
            'she', "she's", 'her', 'hers', 'herself', 'it', "it's", 'it
s', 'itself', 'they', 'them', 'their',\
            'theirs', 'themselves', 'what', 'which', 'who', 'whom', 'th
is', 'that', "that'll", 'these', 'those', \
            'am', 'is', 'are', 'was', 'were', 'be', 'been', 'being', 'h
ave', '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', 'h
ow', 'all', 'any', 'both', 'each', 'few', 'more',\
            'most', 'other', 'some', 'such', 'only', 'own', 'same', 's
o', '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', "doesn't", 'hadn',\
            "hadn't", 'hasn', "hasn't", 'haven', "haven't", 'isn', "is
n'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 [0]:

```
# 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%|██████████| 4986/4986 [00:01<00:00, 3137.37it/s]
```

In [0]:
```
preprocessed_reviews[1500]
```

Out[0]:
```
'wow far two two star reviews one obviously no idea ordering wants cris
py cookies hey sorry reviews nobody good beyond reminding us look order
ing chocolate oatmeal cookies not like combination not order type cooki
e find combo quite nice really oatmeal sort calms rich chocolate flavor
gives cookie sort coconut type consistency let also remember tastes dif
fer given opinion soft chewy cookies advertised not crispy cookies blur
b would say crispy rather chewy happen like raw cookie dough however no
t see taste like raw cookie dough soft however confusion yes stick toge
ther soft cookies tend not individually wrapped would add cost oh yeah
chocolate chip cookies tend somewhat sweet want something hard crisp su
ggest nabiso ginger snaps want cookie soft chewy tastes like combinatio
n chocolate oatmeal give try place second order'
```

[3.2] Preprocessing Review Summary

In [0]:
```
## Similartly you can do preprocessing for review summary also.
```

[4] Featurization

[4.1] BAG OF WORDS

In [0]:
```
#BoW
count_vect = CountVectorizer() #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  ['aa', 'aahhhs', 'aback', 'abandon', 'abates', 'abb
ott', 'abby', 'abdominal', 'abiding', 'ability']
==================================================
the type of count vectorizer  <class 'scipy.sparse.csr.csr_matrix'>
the shape of out text BOW vectorizer  (4986, 12997)
the number of unique words  12997
```

[4.2] Bi-Grams and n-Grams.

In [0]:

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

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

# you can choose these numebrs min_df=10, max_features=5000, of your ch
oice
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_s
hape())
print("the number of unique words including both unigrams and bigrams "
, final_bigram_counts.get_shape()[1])
```

```
the type of count vectorizer  <class 'scipy.sparse.csr.csr_matrix'>
the shape of out text BOW vectorizer  (4986, 3144)
the number of unique words including both unigrams and bigrams  3144
```

the number of unique words including both unigrams and bigrams 3144

[4.3] TF-IDF

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

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

```
some sample features(unique words in the corpus) ['ability', 'able', 'a
ble find', 'able get', 'absolute', 'absolutely', 'absolutely deliciou
s', 'absolutely love', 'absolutely no', 'according']
==================================================
the type of count vectorizer  <class 'scipy.sparse.csr.csr_matrix'>
the shape of out text TFIDF vectorizer  (4986, 3144)
the number of unique words including both unigrams and bigrams  3144
```

[4.4] Word2Vec

In [0]:
```
# 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 [0]:
```
# 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 val
ues
# To use this code-snippet, download "GoogleNews-vectors-negative300.bi
n"
# from https://drive.google.com/file/d/0B7XkCwpI5KDYNlNUTTlSS21pQmM/edi
t
# it's 1.9GB in size.


# http://kavita-ganesan.com/gensim-word2vec-tutorial-starter-code/#.W17
SRFAzZPY
# you can comment this whole cell
# or change these varible according to your need

is_your_ram_gt_16g=False
want_to_use_google_w2v = False
want_to_train_w2v = True

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

elif want_to_use_google_w2v and is_your_ram_gt_16g:
    if os.path.isfile('GoogleNews-vectors-negative300.bin'):
        w2v_model=KeyedVectors.load_word2vec_format('GoogleNews-vectors
-negative300.bin', binary=True)
        print(w2v_model.wv.most_similar('great'))
        print(w2v_model.wv.most_similar('worst'))
    else:
        print("you don't have gogole's word2vec file, keep want_to_trai
n_w2v = True, to train your own w2v ")
```

```
[('snack', 0.9951335191726685), ('calorie', 0.9946465492248535), ('wond
```

```
erful', 0.9946032166481018), ('excellent', 0.9944332838058472), ('espec
ially', 0.9941144585609436), ('baked', 0.9940600395202637), ('salted',
0.994047224521637), ('alternative', 0.9937226176261902), ('tasty', 0.99
36816692352295), ('healthy', 0.9936649799346924)]
==================================================
[('varieties', 0.9994194507598877), ('become', 0.9992934465408325), ('p
opcorn', 0.9992750883102417), ('de', 0.9992610216140747), ('miss', 0.99
92451071739197), ('melitta', 0.999218761920929), ('choice', 0.999210238
4567261), ('american', 0.9991837739944458), ('beef', 0.999178051948547
4), ('finish', 0.9991567134857178)]
```

In [0]:
```
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  3817
sample words  ['product', 'available', 'course', 'total', 'pretty', 'st
inky', 'right', 'nearby', 'used', 'ca', 'not', 'beat', 'great', 'receiv
ed', 'shipment', 'could', 'hardly', 'wait', 'try', 'love', 'call', 'ins
tead', 'removed', 'easily', 'daughter', 'designed', 'printed', 'use',
'car', 'windows', 'beautifully', 'shop', 'program', 'going', 'lot', 'fu
n', 'everywhere', 'like', 'tv', 'computer', 'really', 'good', 'idea',
'final', 'outstanding', 'window', 'everybody', 'asks', 'bought', 'mad
e']
```

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

[4.4.1.1] Avg W2v

In [0]:
```
# 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, yo
```

```
u 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/re
view
    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%|██████████████████████████████████████████████████
██████████| 4986/4986 [00:03<00:00, 1330.47it/s]
```

```
4986
50
```

[4.4.1.2] TFIDF weighted W2v

In [0]:

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

In [0]:

```
# 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 ce
ll_val = tfidf

tfidf_sent_vectors = []; # the tfidf-w2v for each sentence/review is st
ored 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/r
eview
    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%|██████████████████████████████████████████████████████████████
██████████| 4986/4986 [00:20<00:00, 245.63it/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.](#)

In [0]:

```
# using SQLite Table to read data.
con = sqlite3.connect('final.sqlite')
final = pd.read_sql_query(""" SELECT * FROM Reviews WHERE Score != 3
 """, con)
final = final[:40000]
```

In [6]:

```
final.shape
```

Out[6]: (40000, 12)

In [0]:

```
from sklearn.model_selection import train_test_split
##Sorting data according to Time in ascending order for Time Based Spli
tting
time_sorted_data = final.sort_values('Time', axis=0, ascending=True, in
```

```
place=False, kind='quicksort', na_position='last')

x = time_sorted_data['CleanedText'].values
y = time_sorted_data['Score']

# split the data set into train and test
X_train, X_test, Y_train, Y_test = train_test_split(x, y, test_size=0.3
, random_state=0,shuffle=False)
```

[5.1] Applying RF

[5.1.1] Applying Random Forests on BOW, SET 1

In [8]:

```
# Please write all the code with proper documentation
#BoW
count_vect = CountVectorizer(min_df = 1000)
X_train_vec = count_vect.fit_transform(X_train)
X_test_vec = count_vect.transform(X_test)
print("the type of count vectorizer :",type(X_train_vec))
print("the shape of out text BOW vectorizer : ",X_train_vec.get_shape
())
print("the number of unique words :", X_train_vec.get_shape()[1])
```

```
the type of count vectorizer : <class 'scipy.sparse.csr.csr_matrix'>
the shape of out text BOW vectorizer :  (28000, 172)
the number of unique words : 172
```

In [0]:

```
from sklearn.preprocessing import StandardScaler
sc = StandardScaler(with_mean=False)
X_train_vec_standardized = sc.fit_transform(X_train_vec)
X_test_vec_standardized = sc.transform(X_test_vec)
```

In [10]:

```
# Importing libraries
from sklearn.ensemble import RandomForestClassifier
from sklearn.model_selection import GridSearchCV
```

```
from sklearn.metrics import accuracy_score,confusion_matrix,f1_score,pr
ecision_score,recall_score

base_learners = [5, 10, 50, 100, 200, 500, 1000]
depth = [2, 3, 4, 5, 6, 7, 8, 9, 10]

param_grid = {'n_estimators': base_learners,'max_depth':depth}
RFC = RandomForestClassifier(max_features='sqrt')
model = GridSearchCV(RFC, param_grid, scoring = 'roc_auc', cv=3 , n_job
s = -1,pre_dispatch=2)
model.fit(X_train_vec_standardized, Y_train)
print("Model with best parameters :\n",model.best_estimator_)
print("Accuracy of the model : ",model.score(X_test_vec_standardized, Y
_test))
```

```
Model with best parameters :
 RandomForestClassifier(bootstrap=True, class_weight=None, criterion='g
ini',
            max_depth=10, max_features='sqrt', 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=Non
e,
            oob_score=False, random_state=None, verbose=0,
            warm_start=False)
Accuracy of the model :  0.8133993758960044
```

In [11]:

```
# Cross-Validation errors
cv_errors = [1-i for i in model.cv_results_['mean_test_score']]
training_scores=[1-i for i in model.cv_results_['mean_train_score']]

# Optimal value of number of base learners
optimal_learners = model.best_estimator_.n_estimators
print("The optimal number of base learners is : ",optimal_learners)


optimal_depth=model.best_estimator_.max_depth
print("The optimal number of depth is : ",optimal_depth)
```

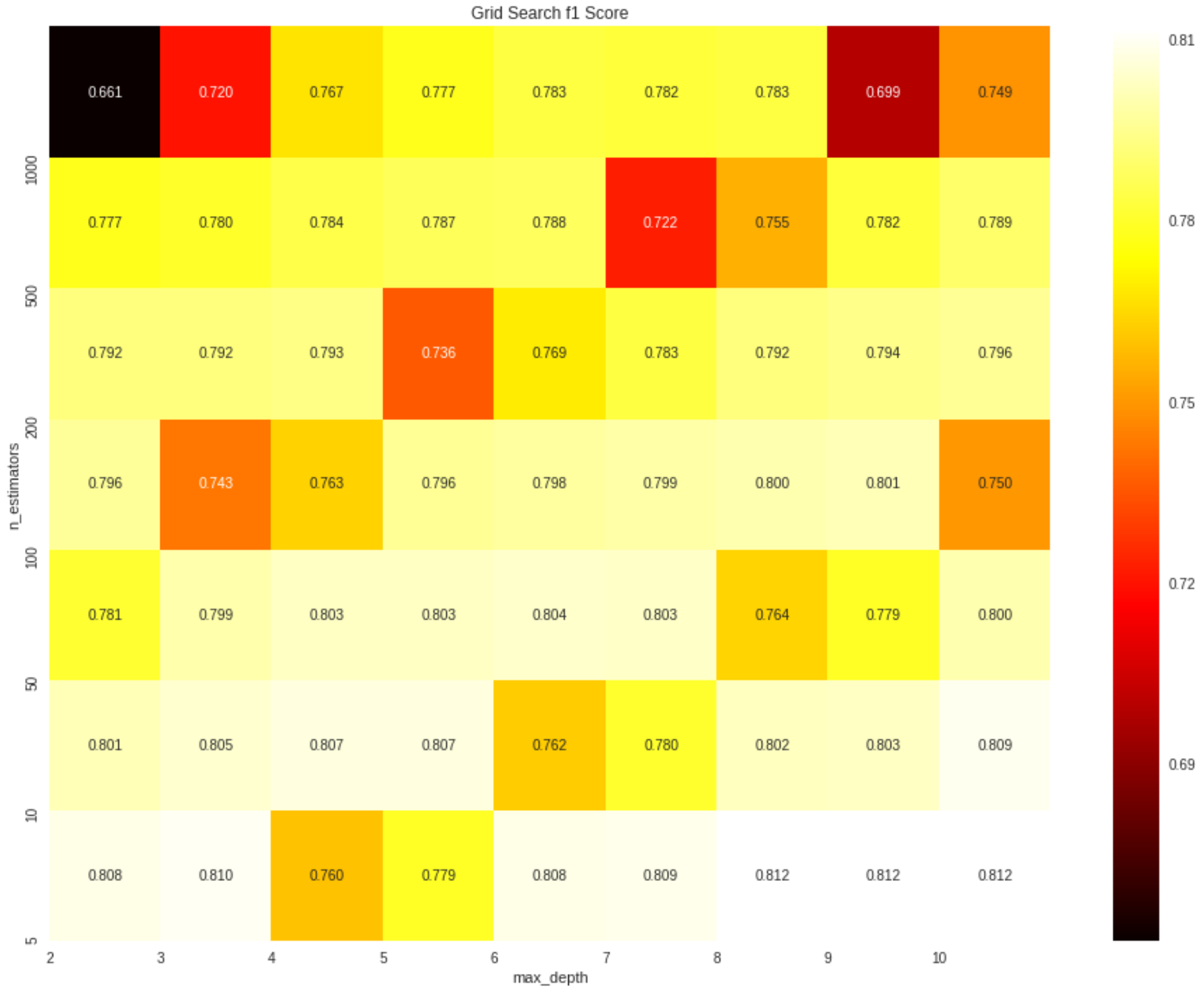
```
The optimal number of base learners is :  1000
```

The optimal number of base learners is : 1000
The optimal number of depth is : 10

In [0]:
```
# RandomForestClassifier with Optimal number of base learners
rf = RandomForestClassifier(n_estimators=optimal_learners, max_features
='sqrt', n_jobs=-1,max_depth=optimal_depth)
rf.fit(X_train_vec_standardized,Y_train)
predictions = rf.predict(X_test_vec_standardized)
predictions1 = rf.predict(X_train_vec_standardized)

# Variables that will be used for  making table in Conclusion part of t
his assignment
bow_rf_learners = optimal_learners
bow_rf_depth = optimal_depth
bow_rf_train_acc = model.score(X_test_vec_standardized, Y_test)*100
bow_rf_test_acc = accuracy_score(Y_test, predictions) * 100
```

In [13]:
```
import seaborn as sns
print("Best HyperParameter: ",model.best_params_)
print(model.best_score_)
scores = model.cv_results_['mean_test_score'].reshape(len(base_learners
),len(depth))

plt.figure(figsize=(16, 12))
sns.heatmap(scores, annot=True, cmap=plt.cm.hot, fmt=".3f", xticklabels
=base_learners, yticklabels=depth)
plt.ylabel('n_estimators')
plt.xlabel('max_depth')
plt.yticks(np.arange(len(base_learners)), base_learners)
plt.xticks(np.arange(len(depth)), depth)
plt.title('Grid Search f1 Score')
plt.show()
```

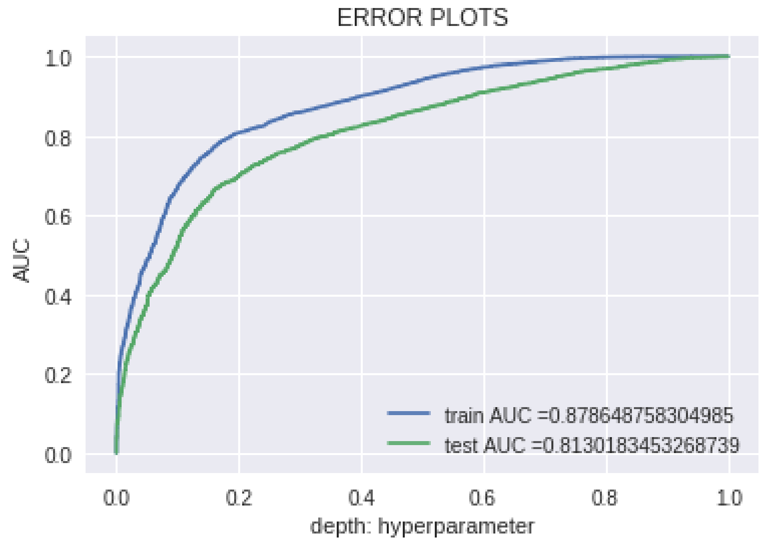
Best HyperParameter: {'max_depth': 10, 'n_estimators': 1000}
0.8120366959186399



In [14]:

```
train_fpr, train_tpr, thresholds = roc_curve(Y_train, rf.predict_proba(
X_train_vec_standardized)[:,1])
test_fpr, test_tpr, thresholds = roc_curve(Y_test, rf.predict_proba(X_t
est_vec_standardized)[:,1])

plt.plot(train_fpr, train_tpr, label="train AUC ="+str(auc(train_fpr, t
rain_tpr)))
plt.plot(test_fpr, test_tpr, label="test AUC ="+str(auc(test_fpr, test_
tpr)))
```

```
plt.legend()
plt.xlabel("depth: hyperparameter")
plt.ylabel("AUC")
plt.title("ERROR PLOTS")
plt.show()
```



In [15]:

```
# evaluate accuracy on test data
acc = accuracy_score(Y_test, predictions) * 100
print('\nThe Test Accuracy of the DecisionTreeClassifier for depth = %d
 is %f%%' % (optimal_depth, acc))
print('\nThe Test Accuracy of the DecisionTreeClassifier for depth = %d
 is %f%%' % (optimal_learners, acc))

# evaluate precision
acc = precision_score(Y_test, predictions, pos_label = 1)
print('\nThe Test Precision of the DecisionTreeClassifier for depth = %
d is %f' % (optimal_depth, acc))
print('\nThe Test Precision of the DecisionTreeClassifier for depth = %
d is %f' % (optimal_learners, acc))

# evaluate recall
acc = recall_score(Y_test, predictions, pos_label = 1)
```

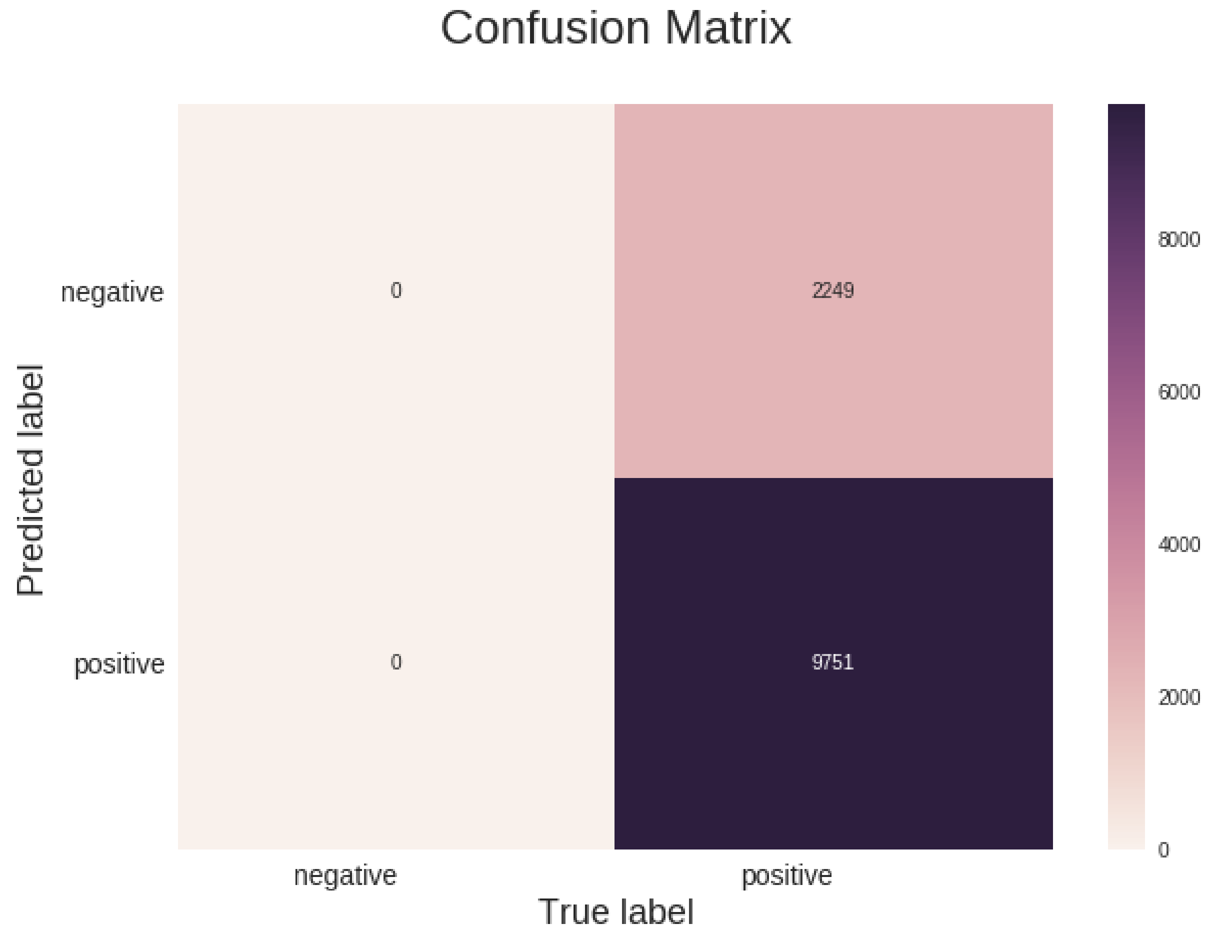
```
print('\nThe Test Recall of the DecisionTreeClassifier for depth = %d i
s %f' % (optimal_depth, acc))
print('\nThe Test Recall of the DecisionTreeClassifier for depth = %d i
s %f' % (optimal_learners, acc))

# evaluate f1-score
acc = f1_score(Y_test, predictions, pos_label = 1)
print('\nThe Test F1-Score of the DecisionTreeClassifier for depth = %d
 is %f' % (optimal_depth, acc))
print('\nThe Test F1-Score of the DecisionTreeClassifier for depth = %d
 is %f' % (optimal_learners, acc))
```

```
The Test Accuracy of the DecisionTreeClassifier for depth = 10 is 81.25
8333%

The Test Accuracy of the DecisionTreeClassifier for depth = 1000 is 81.
258333%

The Test Precision of the DecisionTreeClassifier for depth = 10 is 0.81
2583

The Test Precision of the DecisionTreeClassifier for depth = 1000 is 0.
812583

The Test Recall of the DecisionTreeClassifier for depth = 10 is 1.00000
0

The Test Recall of the DecisionTreeClassifier for depth = 1000 is 1.000
000

The Test F1-Score of the DecisionTreeClassifier for depth = 10 is 0.896
602

The Test F1-Score of the DecisionTreeClassifier for depth = 1000 is 0.8
96602
```

In [16]:

```
# Code for drawing seaborn heatmaps on test data
class_names = ['negative','positive']
df_heatmap = pd.DataFrame(confusion_matrix(Y_test, predictions), index=
```

```
class_names, columns=class_names )
fig = plt.figure(figsize=(10,7))
heatmap = sns.heatmap(df_heatmap, annot=True, fmt="d")

# Setting tick labels for heatmap
heatmap.yaxis.set_ticklabels(heatmap.yaxis.get_ticklabels(), rotation=0
, ha='right', fontsize=14)
heatmap.xaxis.set_ticklabels(heatmap.xaxis.get_ticklabels(), rotation=0
, ha='right', fontsize=14)
plt.ylabel('Predicted label',size=18)
plt.xlabel('True label',size=18)
plt.title("Confusion Matrix\n",size=24)
plt.show()
```



In [17]:

```
# evaluate accuracy on train data
acc = accuracy_score(Y_train, predictions1) * 100
print('\nThe Train Accuracy of the DecisionTreeClassifier for depth = %
d is %f%%' % (optimal_depth, acc))
print('\nThe Train Accuracy of the DecisionTreeClassifier for depth = %
d is %f%%' % (optimal_learners, acc))

# evaluate precision
acc = precision_score(Y_train, predictions1, pos_label = 1)
print('\nThe Train Precision of the DecisionTreeClassifier for depth =
```

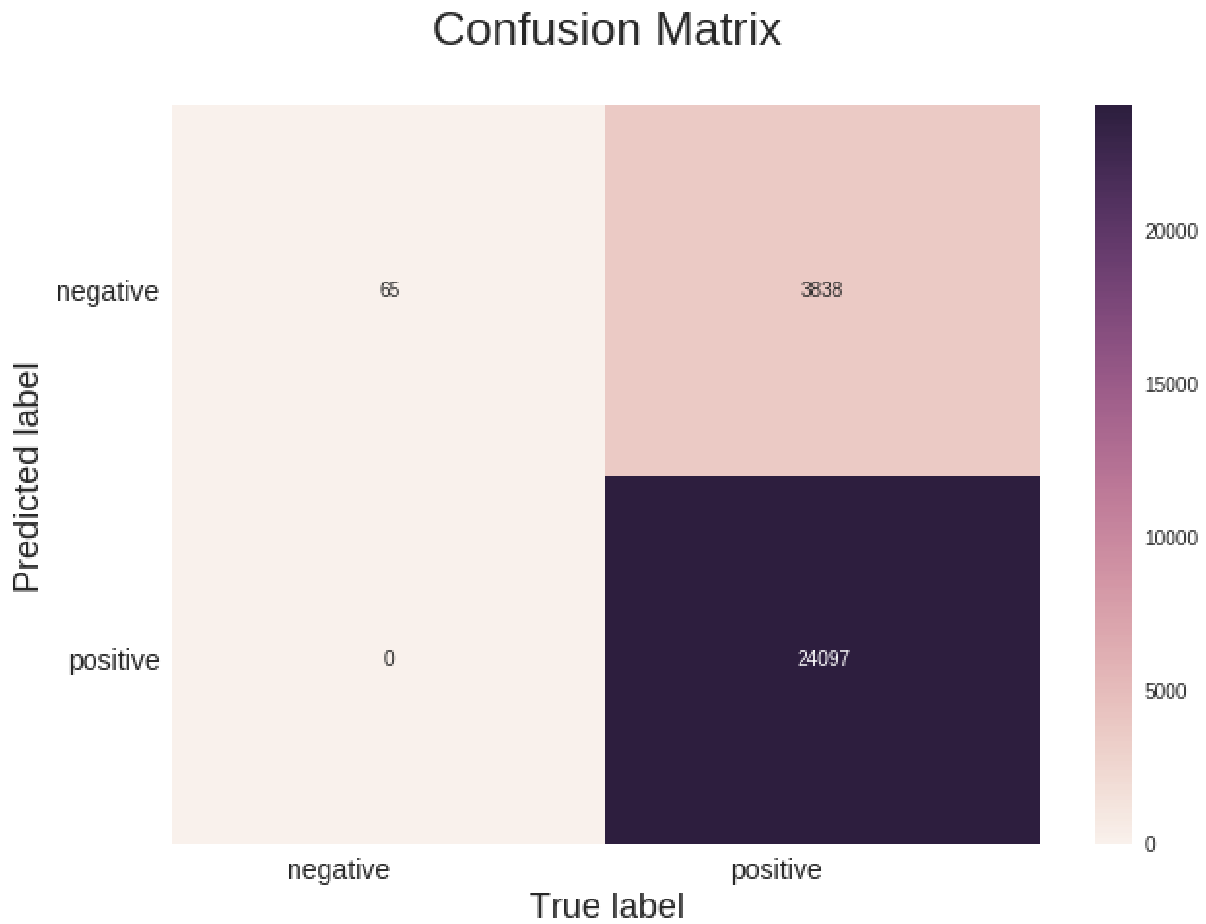
```
%d is %f' % (optimal_depth, acc))
print('\nThe Train Precision of the DecisionTreeClassifier for depth =
%d is %f' % (optimal_learners, acc))

# evaluate recall
acc = recall_score(Y_train, predictions1, pos_label = 1)
print('\nThe Train Recall of the DecisionTreeClassifier for depth = %d
 is %f' % (optimal_depth, acc))
print('\nThe Train Recall of the DecisionTreeClassifier for depth = %d
 is %f' % (optimal_learners, acc))


# evaluate f1-score
acc = f1_score(Y_train, predictions1, pos_label = 1)
print('\nThe Train F1-Score of the DecisionTreeClassifier for depth = %
d is %f' % (optimal_depth, acc))
print('\nThe Train F1-Score of the DecisionTreeClassifier for depth = %
d is %f' % (optimal_learners, acc))
```

```
The Train Accuracy of the DecisionTreeClassifier for depth = 10 is 86.2
92857%

The Train Accuracy of the DecisionTreeClassifier for depth = 1000 is 8
6.292857%

The Train Precision of the DecisionTreeClassifier for depth = 10 is 0.8
62610

The Train Precision of the DecisionTreeClassifier for depth = 1000 is
0.862610

The Train Recall of the DecisionTreeClassifier for depth = 10 is 1.0000
00

The Train Recall of the DecisionTreeClassifier for depth = 1000 is 1.00
0000

The Train F1-Score of the DecisionTreeClassifier for depth = 10 is 0.92
6238
```

```
The Train F1-Score of the DecisionTreeClassifier for depth = 1000 is 0.
926238
```

In [18]:

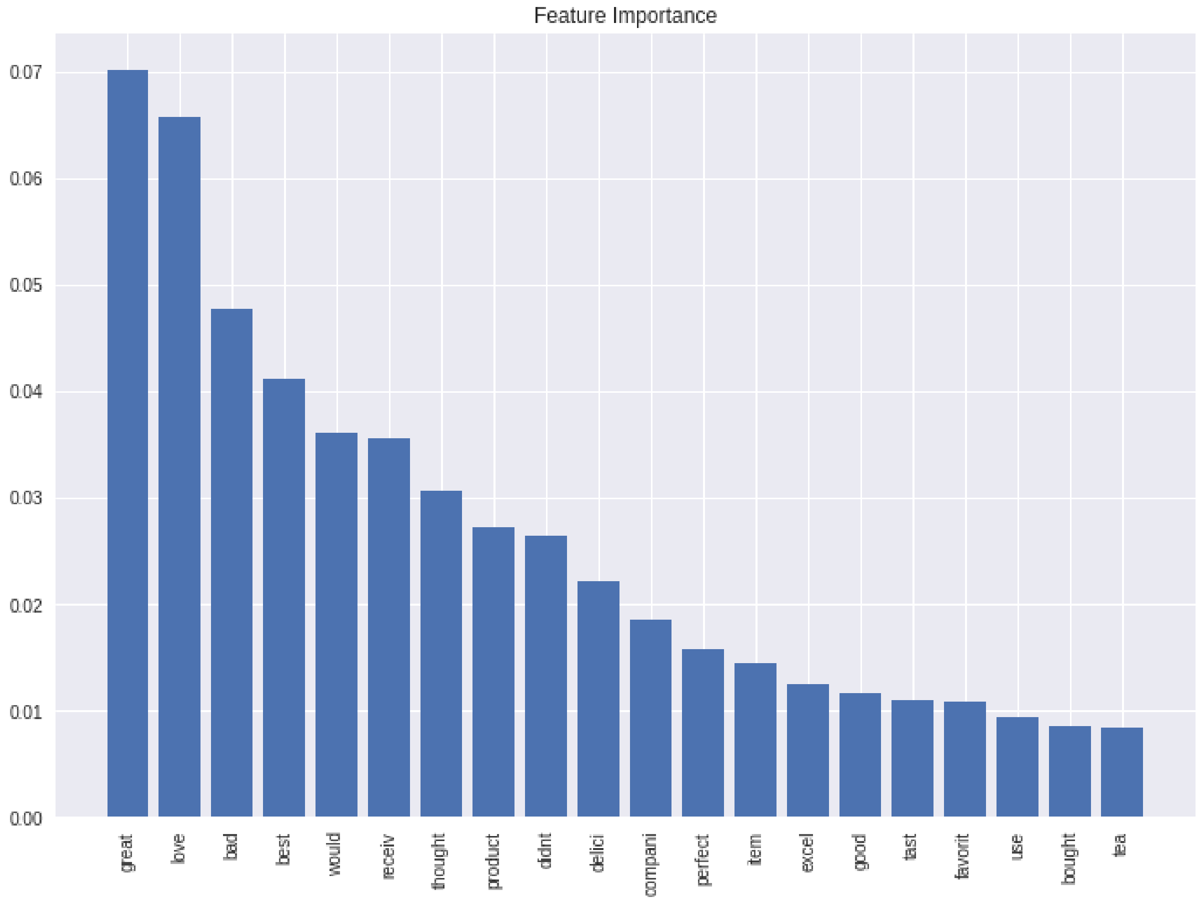
```
# Code for drawing seaborn heatmaps
class_names = ['negative','positive']
df_heatmap = pd.DataFrame(confusion_matrix(Y_train, predictions1), inde
x=class_names, columns=class_names )
fig = plt.figure(figsize=(10,7))
heatmap = sns.heatmap(df_heatmap, annot=True, fmt="d")

# Setting tick labels for heatmap
heatmap.yaxis.set_ticklabels(heatmap.yaxis.get_ticklabels(), rotation=0
, ha='right', fontsize=14)
heatmap.xaxis.set_ticklabels(heatmap.xaxis.get_ticklabels(), rotation=0
, ha='right', fontsize=14)
plt.ylabel('Predicted label',size=18)
plt.xlabel('True label',size=18)
plt.title("Confusion Matrix\n",size=24)
plt.show()
```



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

In [19]:
```
# Calculate feature importances from decision trees
importances = rf.feature_importances_

# Sort feature importances in descending order
indices = np.argsort(importances)[::-1][:20]

# Rearrange feature names so they match the sorted feature importances
```

```python
names = count_vect.get_feature_names()

sns.set(rc={'figure.figsize':(11.7,8.27)})

# Create plot
plt.figure()

# Create plot title
plt.title("Feature Importance")

# Add bars
plt.bar(range(20), importances[indices])

# Add feature names as x-axis labels
names = np.array(names)
plt.xticks(range(20), names[indices], rotation=90)

# Show plot
plt.show()
# uni_gram.get_feature_names()
```



In [20]:
```
#feature importance
rf = RandomForestClassifier(n_estimators=optimal_learners, max_features
='sqrt', n_jobs=-1,max_depth=optimal_depth)
rf.fit(X_train_vec_standardized,Y_train)
print('top 25 words and their IG---')
top=rf.feature_importances_
s=np.argsort(top)[-20:]
feature=count_vect.get_feature_names()
y=[]
for i in range(20):
    index=s[i]
```

```
    y.append(feature[index])
    print(feature[index],'\t\t:\t\t',round(top[index],5))
```

```
top 25 words and their IG---
bought          :               0.00866
review          :               0.00906
use             :               0.00984
favorit                 :               0.00996
tast            :               0.01086
good            :               0.01143
excel           :               0.01239
item            :               0.01402
perfect                 :               0.01489
compani                 :               0.01877
delici          :               0.02135
didnt           :               0.02815
product                 :               0.02904
thought                 :               0.0324
would           :               0.03543
receiv          :               0.03668
best            :               0.04254
bad             :               0.04698
love            :               0.06362
great           :               0.06729
```

In [21]:

```
from wordcloud import WordCloud, STOPWORDS
import matplotlib.pyplot as plt
import pandas as pd

comment_words = ' '
stopwords = set(STOPWORDS)

# iterate through the csv file
for val in y:

    # typecaste each val to string
    val = str(val)

    # split the value
```

```
    tokens = val.split()

    # Converts each token into lowercase
    for i in range(len(tokens)):
        tokens[i] = tokens[i].lower()

    for words in tokens:
        comment_words = comment_words + words + ' '


wordcloud = WordCloud(width = 800, height = 800,
                background_color ='white',
                stopwords = stopwords,
                min_font_size = 10).generate(comment_words)

# plot the WordCloud image
plt.figure(figsize = (8, 8), facecolor = None)
plt.imshow(wordcloud)
plt.axis("off")
plt.tight_layout(pad = 0)

plt.show()
```

[5.1.3] Applying Random Forests on TFIDF, SET 2

In [22]:
```
# Please write all the code with proper documentation
tf_idf_vect = TfidfVectorizer(min_df=1000)
X_train_vec = tf_idf_vect.fit_transform(X_train)
X_test_vec = tf_idf_vect.transform(X_test)
print("the type of count vectorizer :",type(X_train_vec))
print("the shape of out text TFIDF vectorizer : ",X_train_vec.get_shape
())
print("the number of unique words :", X_train_vec.get_shape()[1])

# Data-preprocessing: Standardizing the data
sc = StandardScaler(with_mean=False)
X_train_vec_standardized = sc.fit_transform(X_train_vec)
X_test_vec_standardized = sc.transform(X_test_vec)
```

```
the type of count vectorizer : <class 'scipy.sparse.csr.csr_matrix'>
the shape of out text TFIDF vectorizer :  (28000, 172)
the number of unique words : 172
```

In [23]:
```
# Importing libraries
from sklearn.ensemble import RandomForestClassifier
from sklearn.model_selection import GridSearchCV
from sklearn.metrics import accuracy_score,confusion_matrix,f1_score,pr
ecision_score,recall_score

base_learners = [5, 10, 50, 100, 200, 500, 1000]
depth = [2, 3, 4, 5, 6, 7, 8, 9, 10]

param_grid = {'n_estimators': base_learners,'max_depth':depth}
RFC = RandomForestClassifier(max_features='sqrt')
model = GridSearchCV(RFC, param_grid, scoring = 'roc_auc', cv=3 , n_job
s = -1,pre_dispatch=2)
model.fit(X_train_vec_standardized, Y_train)
print("Model with best parameters :\n",model.best_estimator_)
print("Accuracy of the model : ",model.score(X_test_vec_standardized, Y
_test))
```

```
Accuracy of the model :  0.8162999916233467
```

In [24]:
```
# Cross-Validation errors
```

```
cv_errors = [1-i for i in model.cv_results_['mean_test_score']]
training_scores=[1-i for i in model.cv_results_['mean_train_score']]

# Optimal value of number of base learners
optimal_learners = model.best_estimator_.n_estimators
print("The optimal number of base learners is : ",optimal_learners)


optimal_depth=model.best_estimator_.max_depth
print("The optimal number of depth is : ",optimal_depth)
```

```
The optimal number of base learners is :  1000
The optimal number of depth is :  10
```

In [0]:

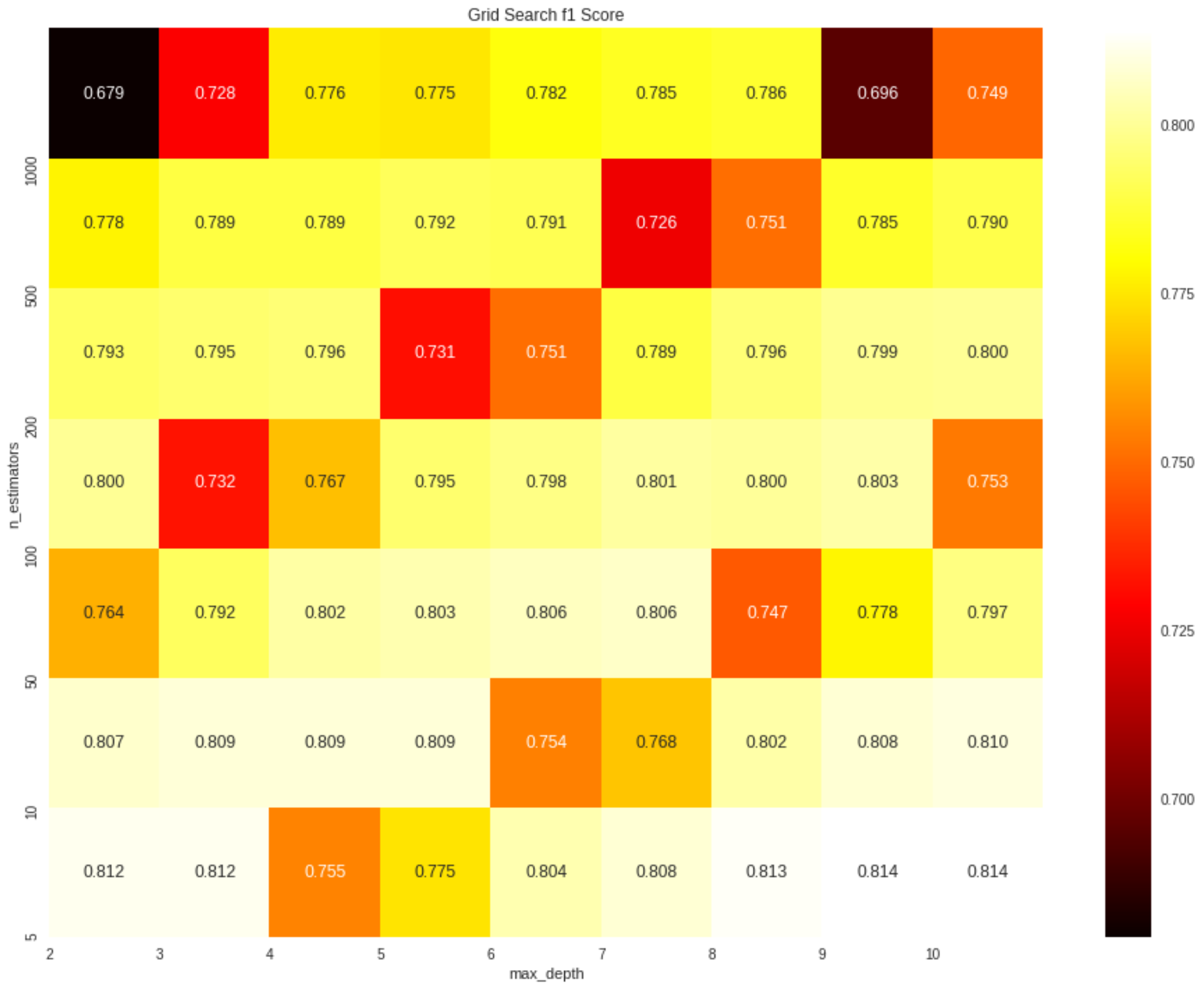
```
# RandomForestClassifier with Optimal number of base learners
rf = RandomForestClassifier(n_estimators=optimal_learners, max_features
='sqrt', n_jobs=-1,max_depth=optimal_depth)
rf.fit(X_train_vec_standardized,Y_train)
predictions = rf.predict(X_test_vec_standardized)
predictions1 = rf.predict(X_train_vec_standardized)

# Variables that will be used for  making table in Conclusion part of t
his assignment
tfidf_rf_learners = optimal_learners
tfidf_rf_depth = optimal_depth
tfidf_rf_train_acc = model.score(X_test_vec_standardized, Y_test)*100
tfidf_rf_test_acc = accuracy_score(Y_test, predictions) * 100
```

In [26]:

```
print("Best HyperParameter: ",model.best_params_)
print(model.best_score_)
scores = model.cv_results_['mean_test_score'].reshape(len(base_learners
),len(depth))

plt.figure(figsize=(16, 12))
sns.heatmap(scores, annot=True, cmap=plt.cm.hot, fmt=".3f", xticklabels
=base_learners, yticklabels=depth)
plt.ylabel('n_estimators')
plt.xlabel('max_depth')
plt.yticks(np.arange(len(base_learners)), base_learners)
```

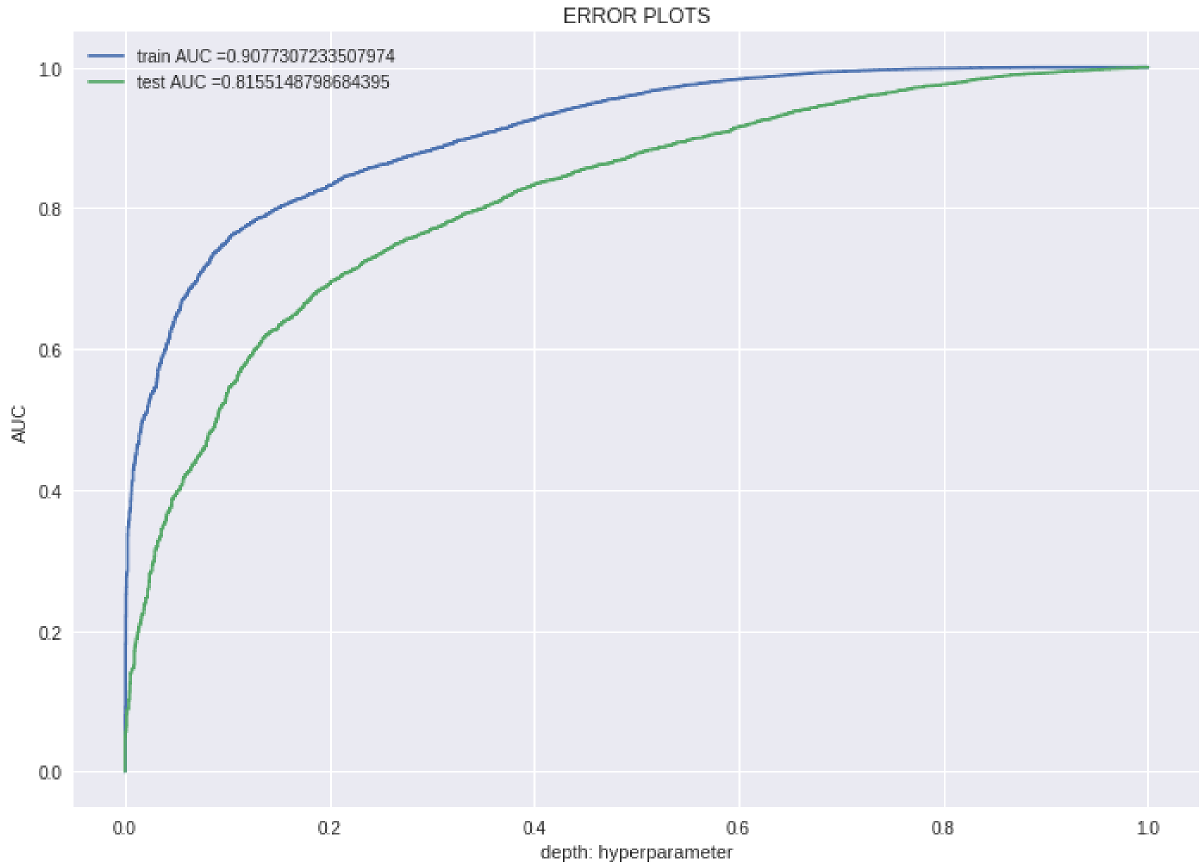
```
plt.xticks(np.arange(len(depth)), depth)
plt.title('Grid Search f1 Score')
plt.show()
```

```
Best HyperParameter:  {'max_depth': 10, 'n_estimators': 1000}
0.8143275448457207
```



In [27]:
```
train_fpr, train_tpr, thresholds = roc_curve(Y_train, rf.predict_proba(
X_train_vec_standardized)[:,1])
test_fpr, test_tpr, thresholds = roc_curve(Y_test, rf.predict_proba(X_t
```

```
est_vec_standardized)[:,1])

plt.plot(train_fpr, train_tpr, label="train AUC ="+str(auc(train_fpr, t
rain_tpr)))
plt.plot(test_fpr, test_tpr, label="test AUC ="+str(auc(test_fpr, test_
tpr)))
plt.legend()
plt.xlabel("depth: hyperparameter")
plt.ylabel("AUC")
plt.title("ERROR PLOTS")
plt.show()
```



In [28]:

```
# evaluate accuracy on test data
```

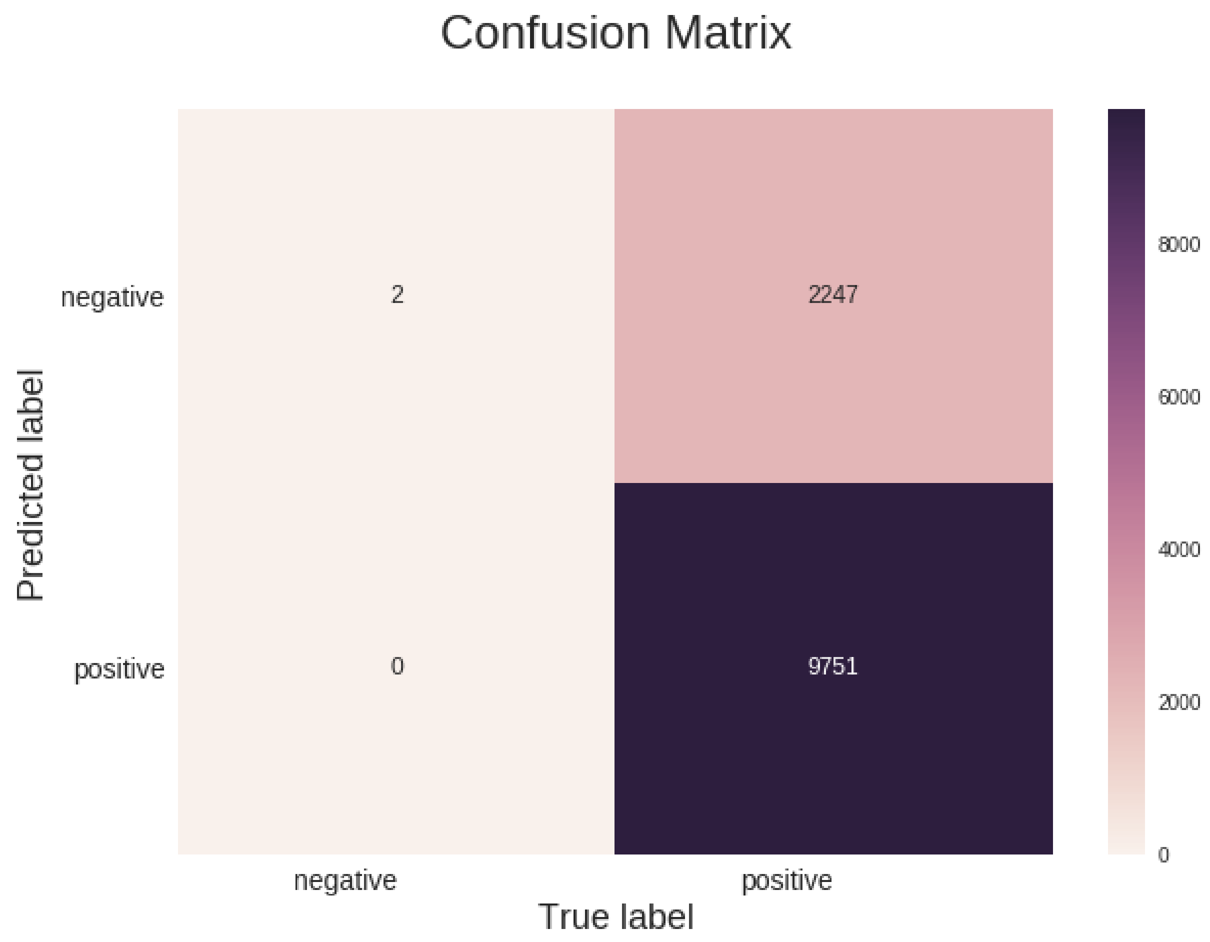
```
acc = accuracy_score(Y_test, predictions) * 100
print('\nThe Test Accuracy of the DecisionTreeClassifier for depth = %d
 is %f%%' % (optimal_depth, acc))
print('\nThe Test Accuracy of the DecisionTreeClassifier for depth = %d
 is %f%%' % (optimal_learners, acc))

# evaluate precision
acc = precision_score(Y_test, predictions, pos_label = 1)
print('\nThe Test Precision of the DecisionTreeClassifier for depth = %
d is %f' % (optimal_depth, acc))
print('\nThe Test Precision of the DecisionTreeClassifier for depth = %
d is %f' % (optimal_learners, acc))

# evaluate recall
acc = recall_score(Y_test, predictions, pos_label = 1)
print('\nThe Test Recall of the DecisionTreeClassifier for depth = %d i
s %f' % (optimal_depth, acc))
print('\nThe Test Recall of the DecisionTreeClassifier for depth = %d i
s %f' % (optimal_learners, acc))

# evaluate f1-score
acc = f1_score(Y_test, predictions, pos_label = 1)
print('\nThe Test F1-Score of the DecisionTreeClassifier for depth = %d
 is %f' % (optimal_depth, acc))
print('\nThe Test F1-Score of the DecisionTreeClassifier for depth = %d
 is %f' % (optimal_learners, acc))
```

```
The Test Accuracy of the DecisionTreeClassifier for depth = 10 is 81.27
5000%

The Test Accuracy of the DecisionTreeClassifier for depth = 1000 is 81.
275000%

The Test Precision of the DecisionTreeClassifier for depth = 10 is 0.81
2719

The Test Precision of the DecisionTreeClassifier for depth = 1000 is 0.
812719

The Test Recall of the DecisionTreeClassifier for depth = 10 is 1.00000
```

```
0

The Test Recall of the DecisionTreeClassifier for depth = 1000 is 1.000
000

The Test F1-Score of the DecisionTreeClassifier for depth = 10 is 0.896
685

The Test F1-Score of the DecisionTreeClassifier for depth = 1000 is 0.8
96685
```

In [29]:

```
# Code for drawing seaborn heatmaps on test data
class_names = ['negative','positive']
df_heatmap = pd.DataFrame(confusion_matrix(Y_test, predictions), index=
class_names, columns=class_names )
fig = plt.figure(figsize=(10,7))
heatmap = sns.heatmap(df_heatmap, annot=True, fmt="d")

# Setting tick labels for heatmap
heatmap.yaxis.set_ticklabels(heatmap.yaxis.get_ticklabels(), rotation=0
, ha='right', fontsize=14)
heatmap.xaxis.set_ticklabels(heatmap.xaxis.get_ticklabels(), rotation=0
, ha='right', fontsize=14)
plt.ylabel('Predicted label',size=18)
plt.xlabel('True label',size=18)
plt.title("Confusion Matrix\n",size=24)
plt.show()
```



In [30]:
```
# evaluate accuracy on train data
acc = accuracy_score(Y_train, predictions1) * 100
print('\nThe Train Accuracy of the DecisionTreeClassifier for depth = %
d is %f%%' % (optimal_depth, acc))
print('\nThe Train Accuracy of the DecisionTreeClassifier for depth = %
d is %f%%' % (optimal_learners, acc))

# evaluate precision
acc = precision_score(Y_train, predictions1, pos_label = 1)
print('\nThe Train Precision of the DecisionTreeClassifier for depth =
```

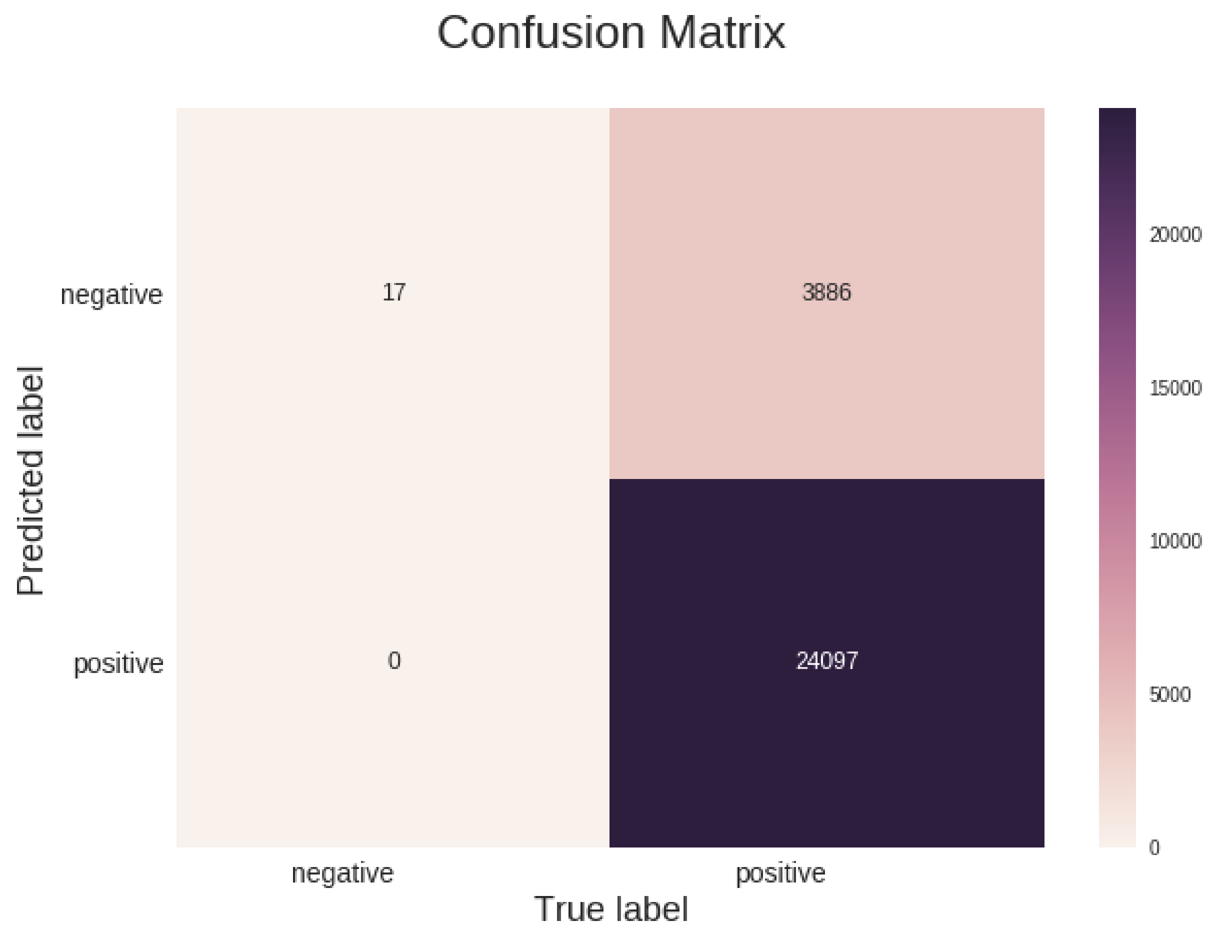
```
%d is %f' % (optimal_depth, acc))
print('\nThe Train Precision of the DecisionTreeClassifier for depth =
%d is %f' % (optimal_learners, acc))

# evaluate recall
acc = recall_score(Y_train, predictions1, pos_label = 1)
print('\nThe Train Recall of the DecisionTreeClassifier for depth = %d
 is %f' % (optimal_depth, acc))
print('\nThe Train Recall of the DecisionTreeClassifier for depth = %d
 is %f' % (optimal_learners, acc))


# evaluate f1-score
acc = f1_score(Y_train, predictions1, pos_label = 1)
print('\nThe Train F1-Score of the DecisionTreeClassifier for depth = %
d is %f' % (optimal_depth, acc))
print('\nThe Train F1-Score of the DecisionTreeClassifier for depth = %
d is %f' % (optimal_learners, acc))
```

```
The Train Accuracy of the DecisionTreeClassifier for depth = 10 is 86.1
21429%

The Train Accuracy of the DecisionTreeClassifier for depth = 1000 is 8
6.121429%

The Train Precision of the DecisionTreeClassifier for depth = 10 is 0.8
61130

The Train Precision of the DecisionTreeClassifier for depth = 1000 is
0.861130

The Train Recall of the DecisionTreeClassifier for depth = 10 is 1.0000
00

The Train Recall of the DecisionTreeClassifier for depth = 1000 is 1.00
0000

The Train F1-Score of the DecisionTreeClassifier for depth = 10 is 0.92
5384
```

```
The Train F1-Score of the DecisionTreeClassifier for depth = 1000 is 0.
925384
```

In [31]:

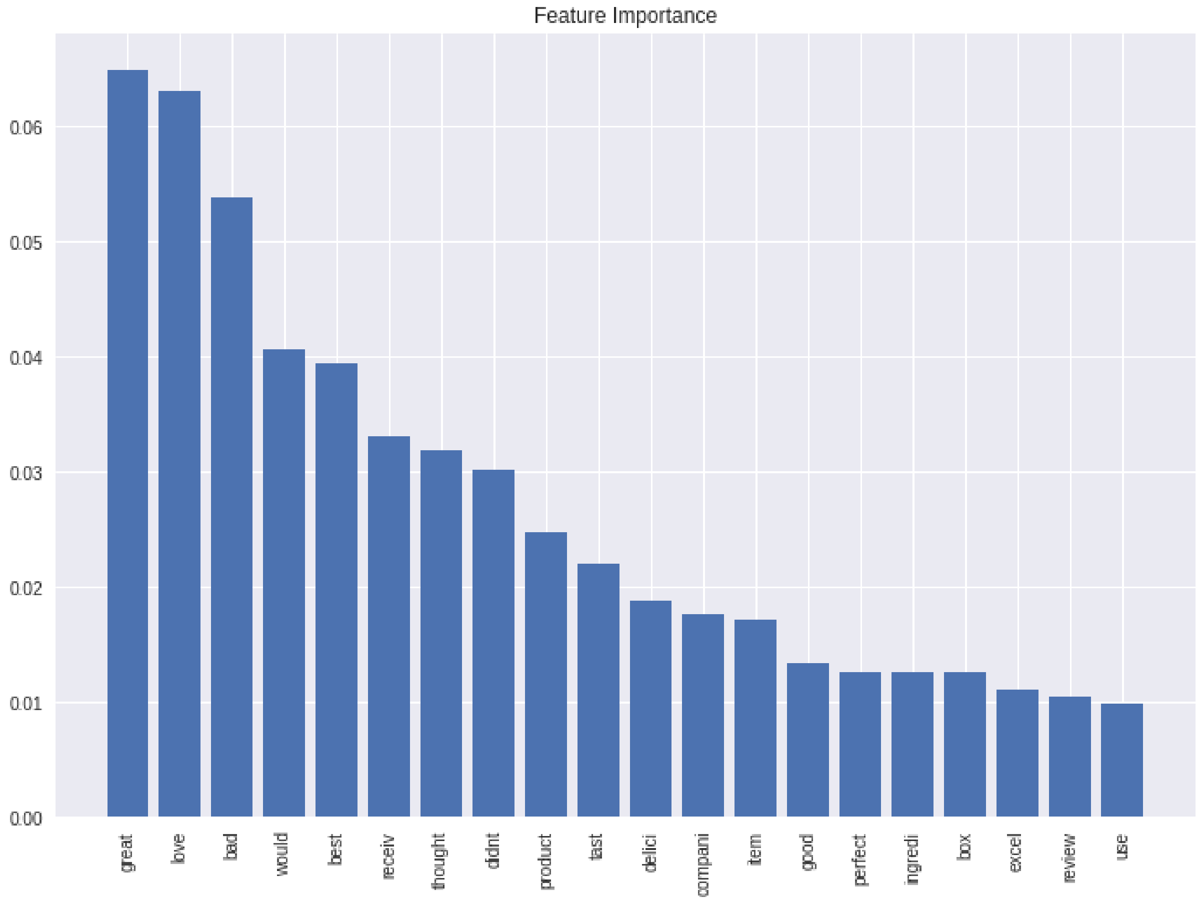
```
# Code for drawing seaborn heatmaps
class_names = ['negative','positive']
df_heatmap = pd.DataFrame(confusion_matrix(Y_train, predictions1), inde
x=class_names, columns=class_names )
fig = plt.figure(figsize=(10,7))
heatmap = sns.heatmap(df_heatmap, annot=True, fmt="d")

# Setting tick labels for heatmap
heatmap.yaxis.set_ticklabels(heatmap.yaxis.get_ticklabels(), rotation=0
, ha='right', fontsize=14)
heatmap.xaxis.set_ticklabels(heatmap.xaxis.get_ticklabels(), rotation=0
, ha='right', fontsize=14)
plt.ylabel('Predicted label',size=18)
plt.xlabel('True label',size=18)
plt.title("Confusion Matrix\n",size=24)
plt.show()
```



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

In [32]:
```
# Please write all the code with proper documentation
# Calculate feature importances from decision trees
importances = rf.feature_importances_

# Sort feature importances in descending order
indices = np.argsort(importances)[::-1][:20]
```

```python
# Rearrange feature names so they match the sorted feature importances
names = tf_idf_vect.get_feature_names()

sns.set(rc={'figure.figsize':(11.7,8.27)})

# Create plot
plt.figure()

# Create plot title
plt.title("Feature Importance")

# Add bars
plt.bar(range(20), importances[indices])

# Add feature names as x-axis labels
names = np.array(names)
plt.xticks(range(20), names[indices], rotation=90)

# Show plot
plt.show()
# uni_gram.get_feature_names()
```



In [33]:
```
#feature importance
rf = RandomForestClassifier(n_estimators=optimal_learners, max_features
='sqrt', n_jobs=-1,max_depth=optimal_depth)
rf.fit(X_train_vec_standardized,Y_train)
print('top 25 words and their IG---')
top=rf.feature_importances_
s=np.argsort(top)[-20:]
feature=tf_idf_vect.get_feature_names()
y=[]
for i in range(20):
    index=s[i]
```

```
    y.append(feature[index])
    print(feature[index],'\t\t:\t\t',round(top[index],5))
```

```
top 25 words and their IG---
review		:		0.00975
look		:		0.00977
excel		:		0.01069
box		:		0.01122
perfect		:		0.01231
ingredi		:		0.0129
good		:		0.01377
item		:		0.01739
compani		:		0.018
delici		:		0.01911
tast		:		0.02084
product		:		0.02455
didnt		:		0.03054
thought		:		0.03202
receiv		:		0.03206
best		:		0.03979
would		:		0.04191
bad		:		0.04941
love		:		0.06494
great		:		0.07242
```

In [34]:

```
from wordcloud import WordCloud, STOPWORDS
import matplotlib.pyplot as plt
import pandas as pd

comment_words = ' '
stopwords = set(STOPWORDS)

# iterate through the csv file
for val in y:

    # typecaste each val to string
    val = str(val)

    # split the value
```

```
    tokens = val.split()

    # Converts each token into lowercase
    for i in range(len(tokens)):
        tokens[i] = tokens[i].lower()

    for words in tokens:
        comment_words = comment_words + words + ' '


wordcloud = WordCloud(width = 800, height = 800,
                background_color ='white',
                stopwords = stopwords,
                min_font_size = 10).generate(comment_words)

# plot the WordCloud image
plt.figure(figsize = (8, 8), facecolor = None)
plt.imshow(wordcloud)
plt.axis("off")
plt.tight_layout(pad = 0)

plt.show()
```

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

In [35]:
```
# Please write all the code with proper documentation
# List of sentence in X_train text
sent_of_train=[]
for sent in X_train:
    sent_of_train.append(sent.split())

# List of sentence in X_est text
sent_of_test=[]
for sent in X_test:
    sent_of_test.append(sent.split())

# Train your own Word2Vec model using your own train text corpus
# min_count = 5 considers only words that occured atleast 5 times
w2v_model=Word2Vec(sent_of_train,min_count=5,size=50, workers=4)

w2v_words = list(w2v_model.wv.vocab)
print("number of words that occured minimum 5 times ",len(w2v_words))
```

```
number of words that occured minimum 5 times  7799
```

In [0]:
```
# compute average word2vec for each review for X_train .
train_vectors = [];
for sent in sent_of_train:
    sent_vec = np.zeros(50)
    cnt_words =0;
    for word in sent: #
        if word in w2v_words:
            vec = w2v_model.wv[word]
            sent_vec += vec
            cnt_words += 1
    if cnt_words != 0:
        sent_vec /= cnt_words
    train_vectors.append(sent_vec)
```

In [0]:
```
# compute average word2vec for each review for X_test .
test_vectors = [];
for sent in sent_of_test:
    sent_vec = np.zeros(50)
    cnt_words =0;
```

```
    for word in sent: #
        if word in w2v_words:
            vec = w2v_model.wv[word]
            sent_vec += vec
            cnt_words += 1
    if cnt_words != 0:
        sent_vec /= cnt_words
    test_vectors.append(sent_vec)

X_train_vec = train_vectors
X_test_vec = test_vectors
X_train_vec_standardized = sc.fit_transform(X_train_vec)
X_test_vec_standardized = sc.transform(X_test_vec)
```

In [38]:

```
# Importing libraries
from sklearn.ensemble import RandomForestClassifier
from sklearn.model_selection import GridSearchCV
from sklearn.metrics import accuracy_score,confusion_matrix,f1_score,pr
ecision_score,recall_score

base_learners = [5, 10, 50, 100, 200, 500, 1000]
depth = [2, 3, 4, 5, 6, 7, 8, 9, 10]
param_grid = {'n_estimators': base_learners,'max_depth':depth}
RFC = RandomForestClassifier(max_features='sqrt')
model = GridSearchCV(RFC, param_grid, scoring = 'roc_auc', cv=3 , n_job
s = -1,pre_dispatch=2)
model.fit(X_train_vec_standardized, Y_train)
print("Model with best parameters :\n",model.best_estimator_)
print("Accuracy of the model : ",model.score(X_test_vec_standardized, Y
_test))
```

```
Model with best parameters :
 RandomForestClassifier(bootstrap=True, class_weight=None, criterion='g
ini',
            max_depth=10, max_features='sqrt', 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=Non
e,
```

```
                oob_score=False, random_state=None, verbose=0,
                warm_start=False)
Accuracy of the model :  0.8557855383395139
```

In [39]:
```
# Cross-Validation errors
cv_errors = [1-i for i in model.cv_results_['mean_test_score']]
training_scores=[1-i for i in model.cv_results_['mean_train_score']]

# Optimal value of number of base learners
optimal_learners = model.best_estimator_.n_estimators
print("The optimal number of base learners is : ",optimal_learners)


optimal_depth=model.best_estimator_.max_depth
print("The optimal number of depth is : ",optimal_depth)
```

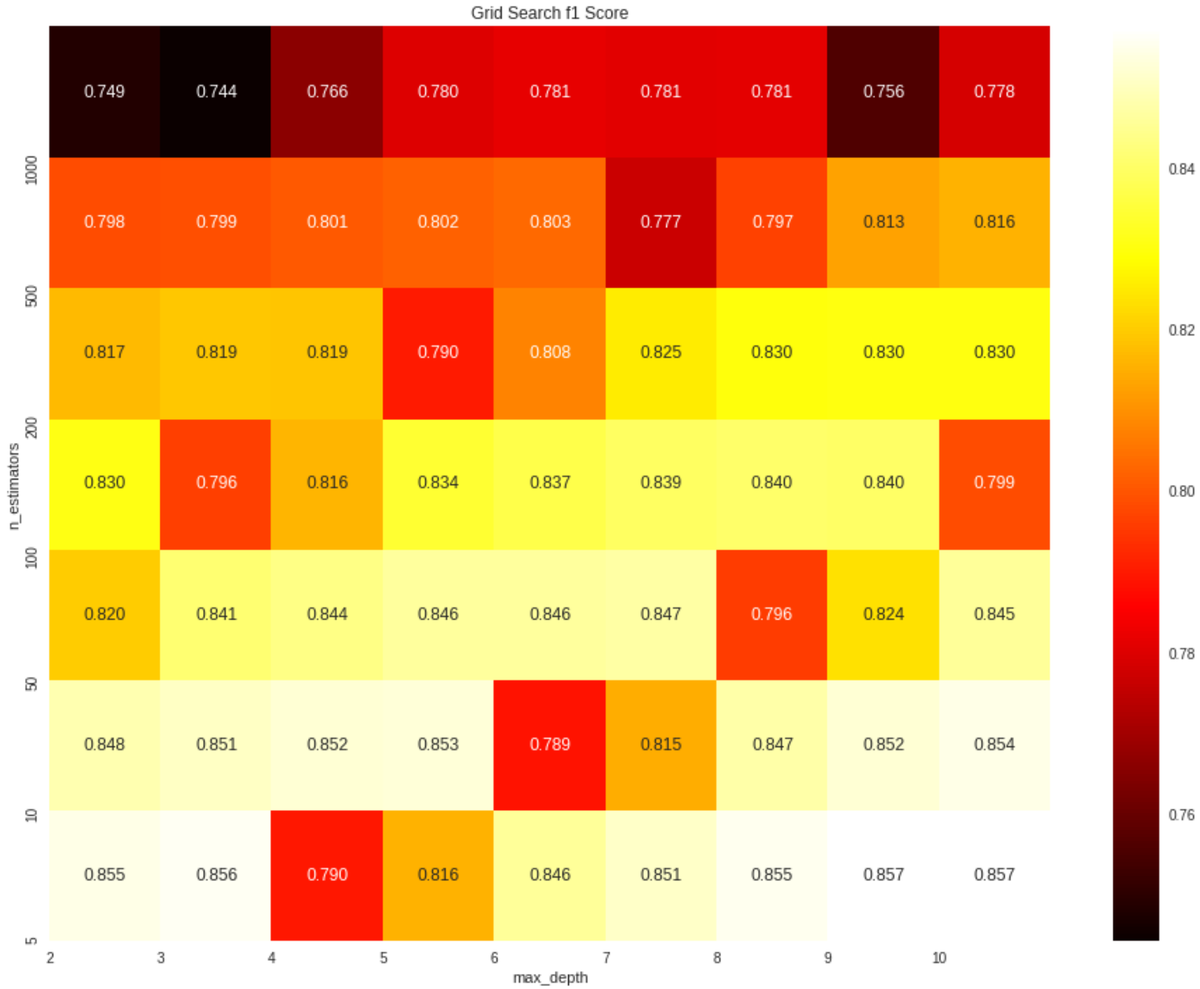
```
The optimal number of base learners is :  1000
The optimal number of depth is :  10
```

In [0]:
```
# RandomForestClassifier with Optimal number of base learners
rf = RandomForestClassifier(n_estimators=optimal_learners, max_features
='sqrt', n_jobs=-1,max_depth=optimal_depth)
rf.fit(X_train_vec_standardized,Y_train)
predictions = rf.predict(X_test_vec_standardized)
predictions1 = rf.predict(X_train_vec_standardized)

# Variables that will be used for  making table in Conclusion part of t
his assignment
avg_w2v_rf_learners = optimal_learners
avg_w2v_rf_depth = optimal_depth
avg_w2v_rf_train_acc = model.score(X_test_vec_standardized, Y_test)*100
avg_w2v_rf_test_acc = accuracy_score(Y_test, predictions) * 100
```

In [41]:
```
print("Best HyperParameter: ",model.best_params_)
print(model.best_score_)
scores = model.cv_results_['mean_test_score'].reshape(len(base_learners
),len(depth))
```

```
plt.figure(figsize=(16, 12))
sns.heatmap(scores, annot=True, cmap=plt.cm.hot, fmt=".3f", xticklabels
=base_learners, yticklabels=depth)
plt.ylabel('n_estimators')
plt.xlabel('max_depth')
plt.yticks(np.arange(len(base_learners)), base_learners)
plt.xticks(np.arange(len(depth)), depth)
plt.title('Grid Search f1 Score')
plt.show()
```

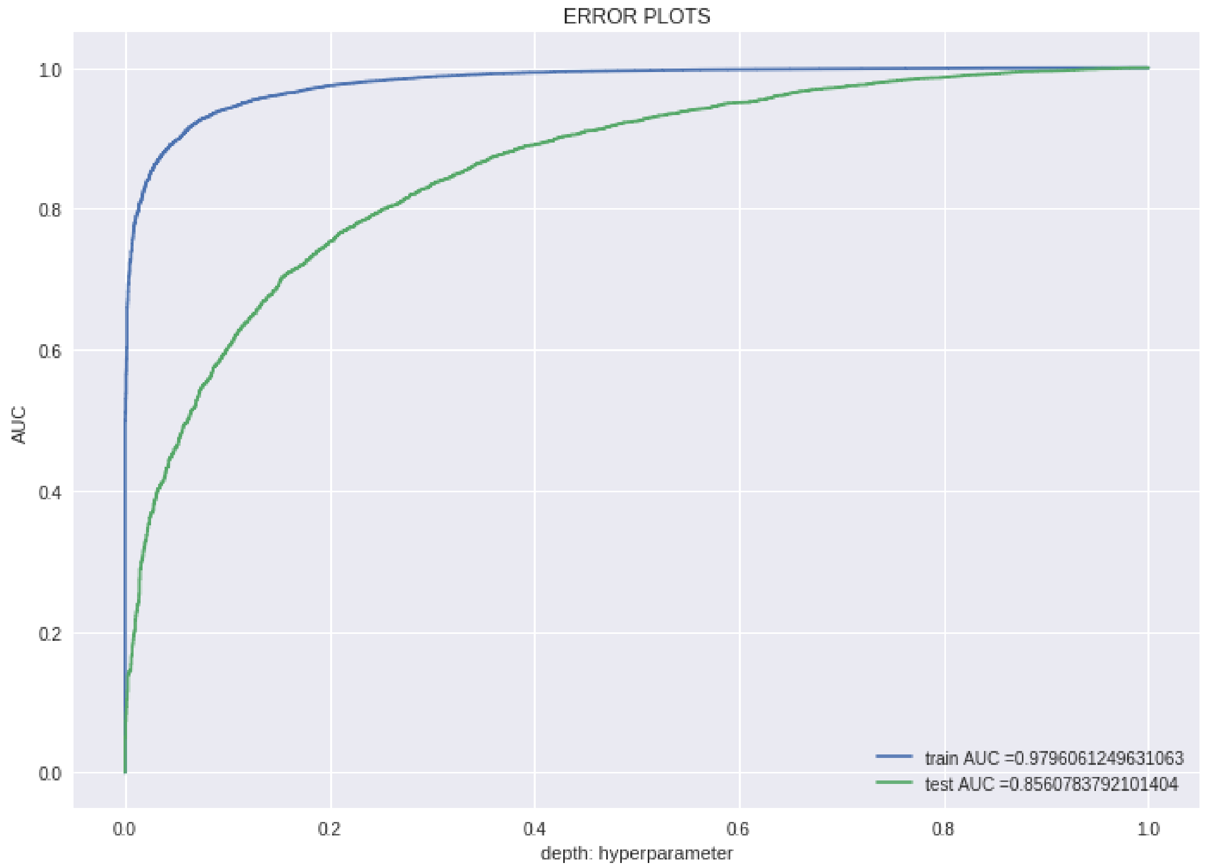
```
Best HyperParameter:  {'max_depth': 10, 'n_estimators': 1000}
0.8573943392129025
```



In [42]:

```
train_fpr, train_tpr, thresholds = roc_curve(Y_train, rf.predict_proba(
X_train_vec_standardized)[:,1])
test_fpr, test_tpr, thresholds = roc_curve(Y_test, rf.predict_proba(X_t
est_vec_standardized)[:,1])

plt.plot(train_fpr, train_tpr, label="train AUC ="+str(auc(train_fpr, t
rain_tpr)))
plt.plot(test_fpr, test_tpr, label="test AUC ="+str(auc(test_fpr, test_
tpr)))
```

```
plt.legend()
plt.xlabel("depth: hyperparameter")
plt.ylabel("AUC")
plt.title("ERROR PLOTS")
plt.show()
```



In [43]:
```
# evaluate accuracy on test data
acc = accuracy_score(Y_test, predictions) * 100
print('\nThe Test Accuracy of the DecisionTreeClassifier for depth = %d
 is %f%%' % (optimal_depth, acc))
print('\nThe Test Accuracy of the DecisionTreeClassifier for depth = %d
 is %f%%' % (optimal_learners, acc))
```

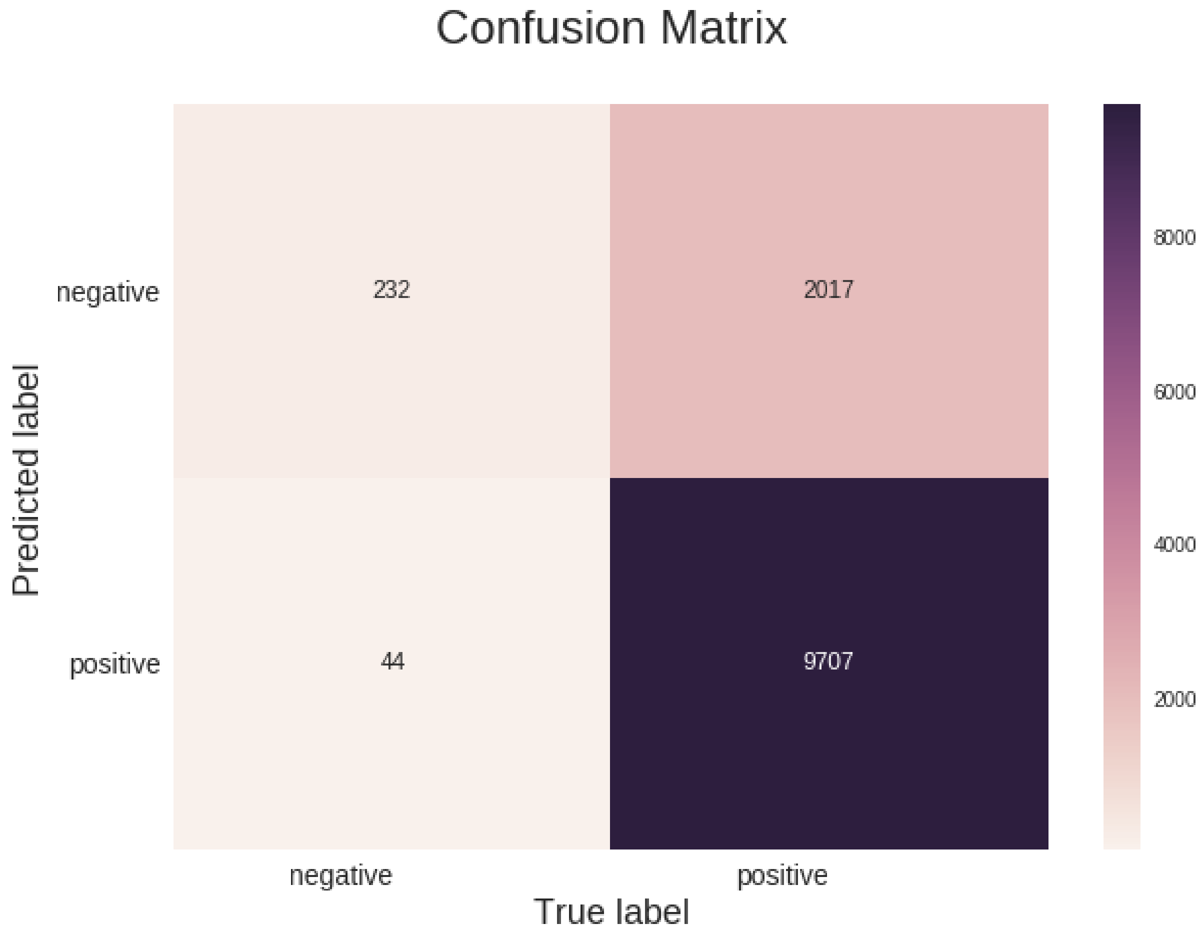
```
# evaluate precision
acc = precision_score(Y_test, predictions, pos_label = 1)
print('\nThe Test Precision of the DecisionTreeClassifier for depth = %d is %f' % (optimal_depth, acc))
print('\nThe Test Precision of the DecisionTreeClassifier for depth = %d is %f' % (optimal_learners, acc))

# evaluate recall
acc = recall_score(Y_test, predictions, pos_label = 1)
print('\nThe Test Recall of the DecisionTreeClassifier for depth = %d is %f' % (optimal_depth, acc))
print('\nThe Test Recall of the DecisionTreeClassifier for depth = %d is %f' % (optimal_learners, acc))

# evaluate f1-score
acc = f1_score(Y_test, predictions, pos_label = 1)
print('\nThe Test F1-Score of the DecisionTreeClassifier for depth = %d is %f' % (optimal_depth, acc))
print('\nThe Test F1-Score of the DecisionTreeClassifier for depth = %d is %f' % (optimal_learners, acc))
```

```
The Test Accuracy of the DecisionTreeClassifier for depth = 10 is 82.825000%

The Test Accuracy of the DecisionTreeClassifier for depth = 1000 is 82.825000%

The Test Precision of the DecisionTreeClassifier for depth = 10 is 0.827960

The Test Precision of the DecisionTreeClassifier for depth = 1000 is 0.827960

The Test Recall of the DecisionTreeClassifier for depth = 10 is 0.995488

The Test Recall of the DecisionTreeClassifier for depth = 1000 is 0.995488

The Test F1-Score of the DecisionTreeClassifier for depth = 10 is 0.904
```

```
028

The Test F1-Score of the DecisionTreeClassifier for depth = 1000 is 0.9
04028
```

In [44]:

```
# Code for drawing seaborn heatmaps on test data
class_names = ['negative','positive']
df_heatmap = pd.DataFrame(confusion_matrix(Y_test, predictions), index=
class_names, columns=class_names )
fig = plt.figure(figsize=(10,7))
heatmap = sns.heatmap(df_heatmap, annot=True, fmt="d")

# Setting tick labels for heatmap
heatmap.yaxis.set_ticklabels(heatmap.yaxis.get_ticklabels(), rotation=0
, ha='right', fontsize=14)
heatmap.xaxis.set_ticklabels(heatmap.xaxis.get_ticklabels(), rotation=0
, ha='right', fontsize=14)
plt.ylabel('Predicted label',size=18)
plt.xlabel('True label',size=18)
plt.title("Confusion Matrix\n",size=24)
plt.show()
```



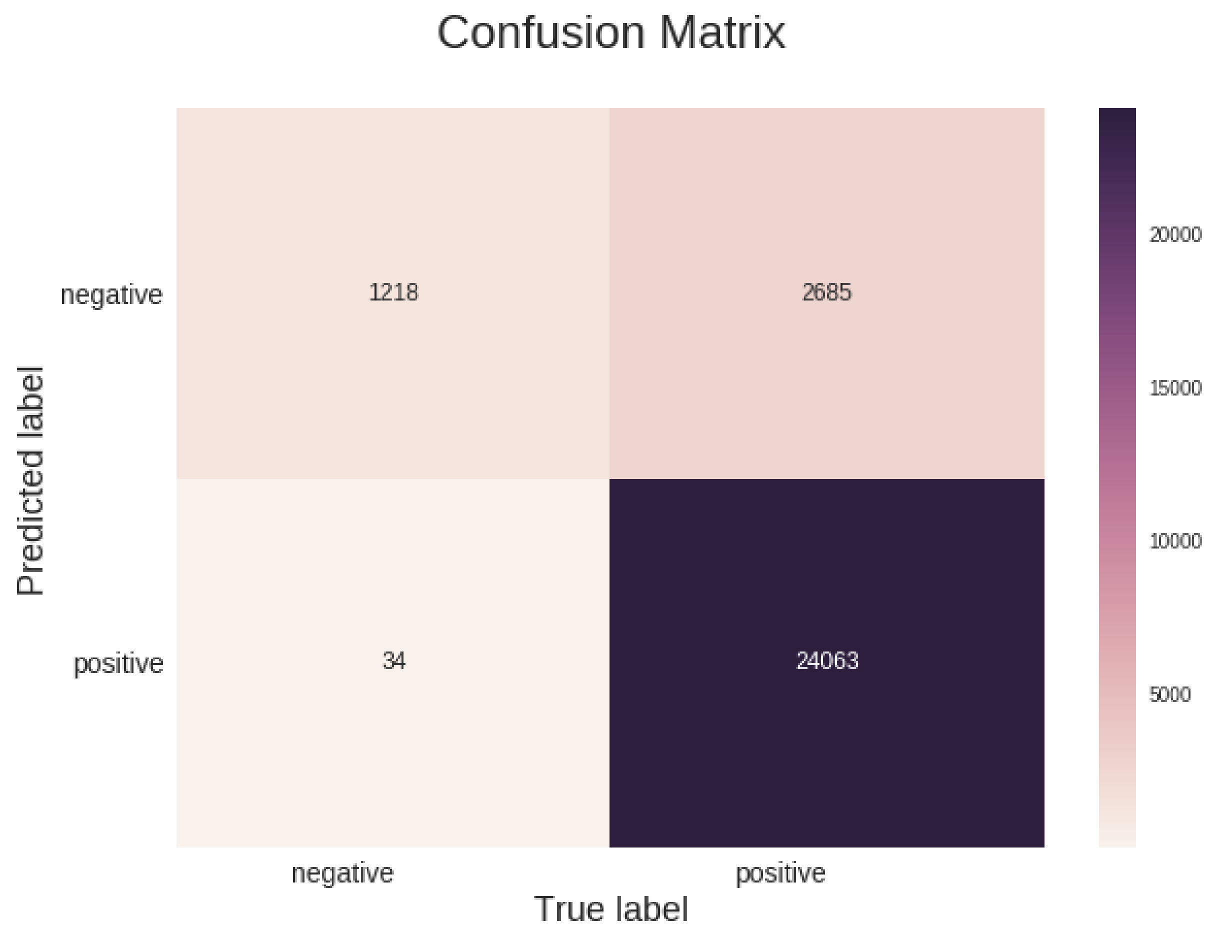
In [45]:
```
# evaluate accuracy on train data
acc = accuracy_score(Y_train, predictions1) * 100
print('\nThe Train Accuracy of the DecisionTreeClassifier for depth = %
d is %f%%' % (optimal_depth, acc))
print('\nThe Train Accuracy of the DecisionTreeClassifier for depth = %
d is %f%%' % (optimal_learners, acc))

# evaluate precision
acc = precision_score(Y_train, predictions1, pos_label = 1)
print('\nThe Train Precision of the DecisionTreeClassifier for depth =
```

```
%d is %f' % (optimal_depth, acc))
print('\nThe Train Precision of the DecisionTreeClassifier for depth =
%d is %f' % (optimal_learners, acc))

# evaluate recall
acc = recall_score(Y_train, predictions1, pos_label = 1)
print('\nThe Train Recall of the DecisionTreeClassifier for depth = %d
 is %f' % (optimal_depth, acc))
print('\nThe Train Recall of the DecisionTreeClassifier for depth = %d
 is %f' % (optimal_learners, acc))


# evaluate f1-score
acc = f1_score(Y_train, predictions1, pos_label = 1)
print('\nThe Train F1-Score of the DecisionTreeClassifier for depth = %
d is %f' % (optimal_depth, acc))
print('\nThe Train F1-Score of the DecisionTreeClassifier for depth = %
d is %f' % (optimal_learners, acc))
```

```
The Train Accuracy of the DecisionTreeClassifier for depth = 10 is 90.2
89286%

The Train Accuracy of the DecisionTreeClassifier for depth = 1000 is 9
0.289286%

The Train Precision of the DecisionTreeClassifier for depth = 10 is 0.8
99619

The Train Precision of the DecisionTreeClassifier for depth = 1000 is
0.899619

The Train Recall of the DecisionTreeClassifier for depth = 10 is 0.9985
89

The Train Recall of the DecisionTreeClassifier for depth = 1000 is 0.99
8589

The Train F1-Score of the DecisionTreeClassifier for depth = 10 is 0.94
6524
```

```
The Train F1-Score of the DecisionTreeClassifier for depth = 1000 is 0.
946524
```

In [46]:
```
# Code for drawing seaborn heatmaps
class_names = ['negative','positive']
df_heatmap = pd.DataFrame(confusion_matrix(Y_train, predictions1), inde
x=class_names, columns=class_names )
fig = plt.figure(figsize=(10,7))
heatmap = sns.heatmap(df_heatmap, annot=True, fmt="d")

# Setting tick labels for heatmap
heatmap.yaxis.set_ticklabels(heatmap.yaxis.get_ticklabels(), rotation=0
, ha='right', fontsize=14)
heatmap.xaxis.set_ticklabels(heatmap.xaxis.get_ticklabels(), rotation=0
, ha='right', fontsize=14)
plt.ylabel('Predicted label',size=18)
plt.xlabel('True label',size=18)
plt.title("Confusion Matrix\n",size=24)
plt.show()
```



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

In [0]:

```
# Please write all the code with proper documentation
# TF-IDF weighted Word2Vec
tf_idf_vect = TfidfVectorizer()

# final_tf_idf1 is the sparse matrix with row= sentence, col=word and c
ell_val = tfidf
final_tf_idf1 = tf_idf_vect.fit_transform(X_train)
```

```
# tfidf words/col-names
tfidf_feat = tf_idf_vect.get_feature_names()

# compute TFIDF Weighted Word2Vec for each review for X_test .
tfidf_test_vectors = [];
row=0;
for sent in sent_of_test:
    sent_vec = np.zeros(50)
    weight_sum =0;
    for word in sent:
        if word in w2v_words:
            vec = w2v_model.wv[word]
            # obtain the tf_idfidf of a word in a sentence/review
            tf_idf = final_tf_idf1[row, tfidf_feat.index(word)]
            sent_vec += (vec * tf_idf)
            weight_sum += tf_idf
    if weight_sum != 0:
        sent_vec /= weight_sum
    tfidf_test_vectors.append(sent_vec)
    row += 1
```

In [0]:

```
# compute TFIDF Weighted Word2Vec for each review for X_train .
tfidf_train_vectors = [];
row=0;
for sent in sent_of_train:
    sent_vec = np.zeros(50)
    weight_sum =0;
    for word in sent:
        if word in w2v_words:
            vec = w2v_model.wv[word]
            # obtain the tf_idfidf of a word in a sentence/review
            tf_idf = final_tf_idf1[row, tfidf_feat.index(word)]
            sent_vec += (vec * tf_idf)
            weight_sum += tf_idf
    if weight_sum != 0:
        sent_vec /= weight_sum
    tfidf_train_vectors.append(sent_vec)
    row += 1
```

```
X_train_vec = tfidf_train_vectors
X_test_vec = tfidf_test_vectors
X_train_vec_standardized = sc.fit_transform(X_train_vec)
X_test_vec_standardized = sc.transform(X_test_vec)
```

In [49]:
```
# Importing libraries
from sklearn.ensemble import RandomForestClassifier
from sklearn.model_selection import GridSearchCV
from sklearn.metrics import accuracy_score,confusion_matrix,f1_score,pr
ecision_score,recall_score

base_learners = [5, 10, 50, 100, 200, 500, 1000]
depth = [2, 3, 4, 5, 6, 7, 8, 9, 10]

param_grid = {'n_estimators': base_learners,'max_depth':depth}
RFC = RandomForestClassifier(max_features='sqrt')
model = GridSearchCV(RFC, param_grid, scoring = 'roc_auc', cv=3 , n_job
s = -1,pre_dispatch=2)
model.fit(X_train_vec_standardized, Y_train)
print("Model with best parameters :\n",model.best_estimator_)
print("Accuracy of the model : ",model.score(X_test_vec_standardized, Y
_test))
```

```
Model with best parameters :
 RandomForestClassifier(bootstrap=True, class_weight=None, criterion='g
ini',
            max_depth=10, max_features='sqrt', 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=500, n_jobs=Non
e,
            oob_score=False, random_state=None, verbose=0,
            warm_start=False)
Accuracy of the model :  0.591744714625842
```

In [50]:
```
# Cross-Validation errors
cv_errors = [1-i for i in model.cv_results_['mean_test_score']]
training_scores=[1-i for i in model.cv_results_['mean_train_score']]
```

```
# Optimal value of number of base learners
optimal_learners = model.best_estimator_.n_estimators
print("The optimal number of base learners is : ",optimal_learners)


optimal_depth=model.best_estimator_.max_depth
print("The optimal number of depth is : ",optimal_depth)
```

```
The optimal number of base learners is :  500
The optimal number of depth is :  10
```

In [0]:

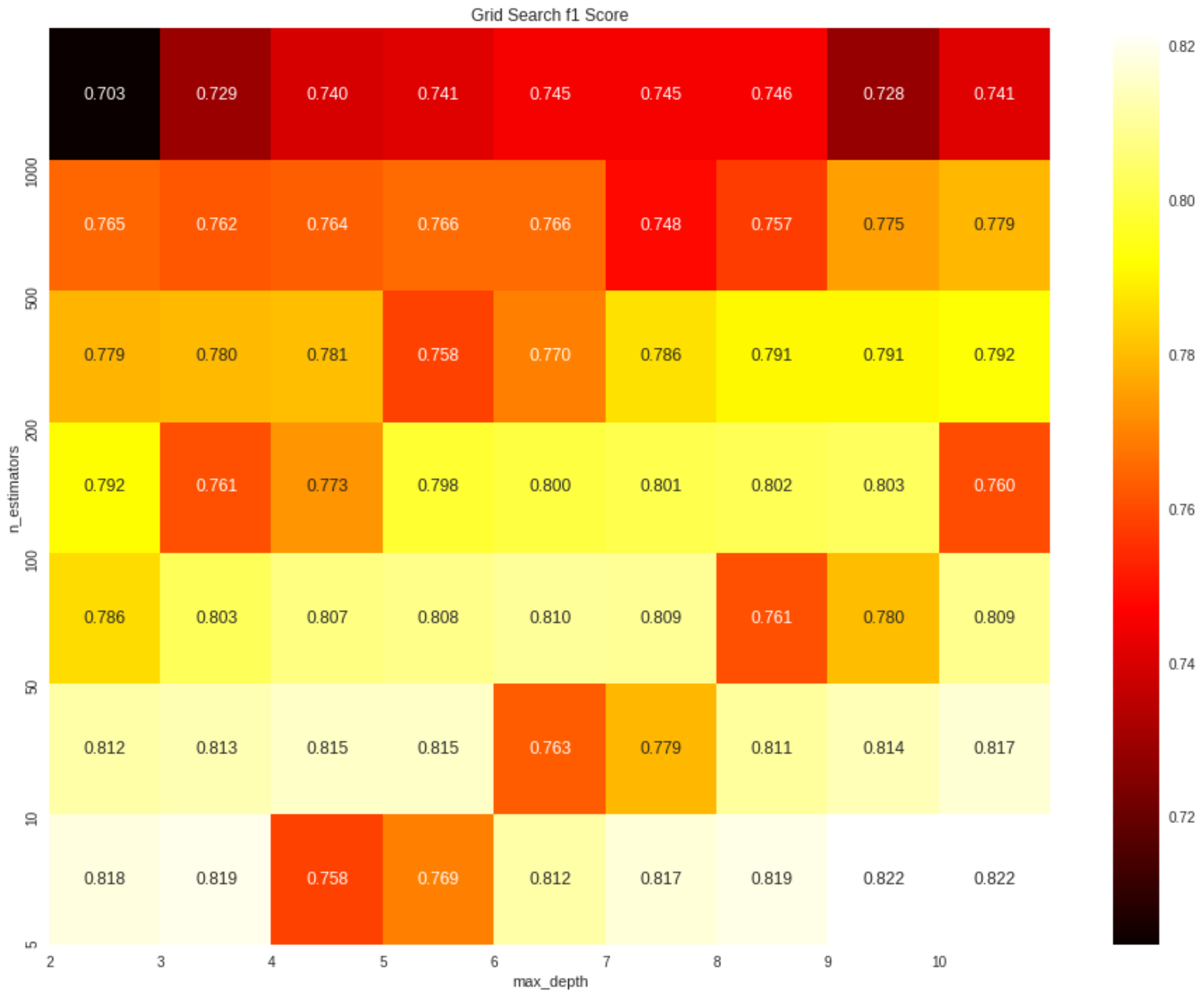
```
# RandomForestClassifier with Optimal number of base learners
rf = RandomForestClassifier(n_estimators=optimal_learners, max_features
='sqrt', n_jobs=-1,max_depth=optimal_depth)
rf.fit(X_train_vec_standardized,Y_train)
predictions = rf.predict(X_test_vec_standardized)
predictions1 = rf.predict(X_train_vec_standardized)

# Variables that will be used for  making table in Conclusion part of t
his assignment
tfidf_avg_w2v_rf_learners = optimal_learners
tfidf_avg_w2v_rf_depth = optimal_depth
tfidf_avg_w2v_rf_train_acc = model.score(X_test_vec_standardized, Y_tes
t)*100
tfidf_avg_w2v_rf_test_acc = accuracy_score(Y_test, predictions) * 100
```

In [52]:

```
print("Best HyperParameter: ",model.best_params_)
print(model.best_score_)
scores = model.cv_results_['mean_test_score'].reshape(len(base_learners
),len(depth))

plt.figure(figsize=(16, 12))
sns.heatmap(scores, annot=True, cmap=plt.cm.hot, fmt=".3f", xticklabels
=base_learners, yticklabels=depth)
plt.ylabel('n_estimators')
plt.xlabel('max_depth')
plt.yticks(np.arange(len(base_learners)), base_learners)
plt.xticks(np.arange(len(depth)), depth)
```

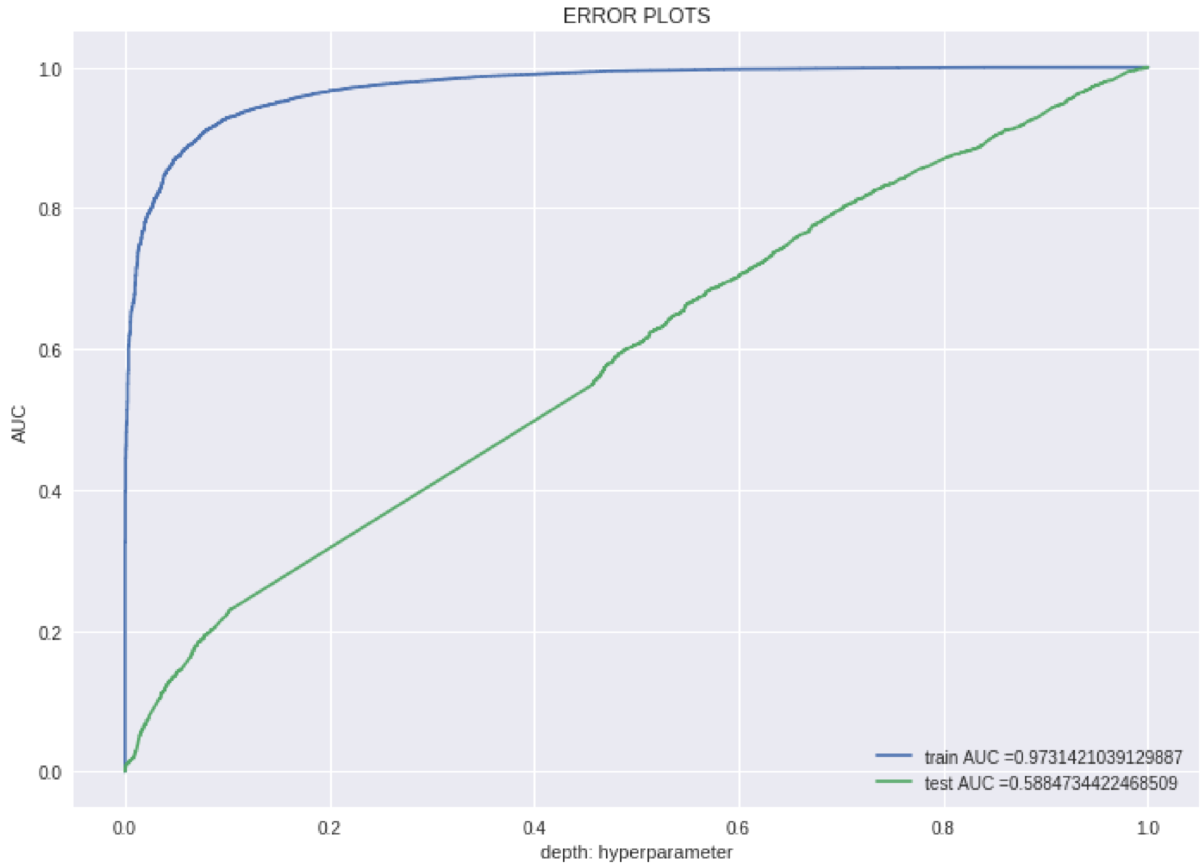
```
plt.title('Grid Search f1 Score')
plt.show()
```

```
Best HyperParameter:  {'max_depth': 10, 'n_estimators': 500}
0.8221183698038219
```



In [53]:
```
train_fpr, train_tpr, thresholds = roc_curve(Y_train, rf.predict_proba(
X_train_vec_standardized)[:,1])
test_fpr, test_tpr, thresholds = roc_curve(Y_test, rf.predict_proba(X_t
```

```
est_vec_standardized)[:,1])

plt.plot(train_fpr, train_tpr, label="train AUC ="+str(auc(train_fpr, t
rain_tpr)))
plt.plot(test_fpr, test_tpr, label="test AUC ="+str(auc(test_fpr, test_
tpr)))
plt.legend()
plt.xlabel("depth: hyperparameter")
plt.ylabel("AUC")
plt.title("ERROR PLOTS")
plt.show()
```



In [54]:
```
# evaluate accuracy on test data
```

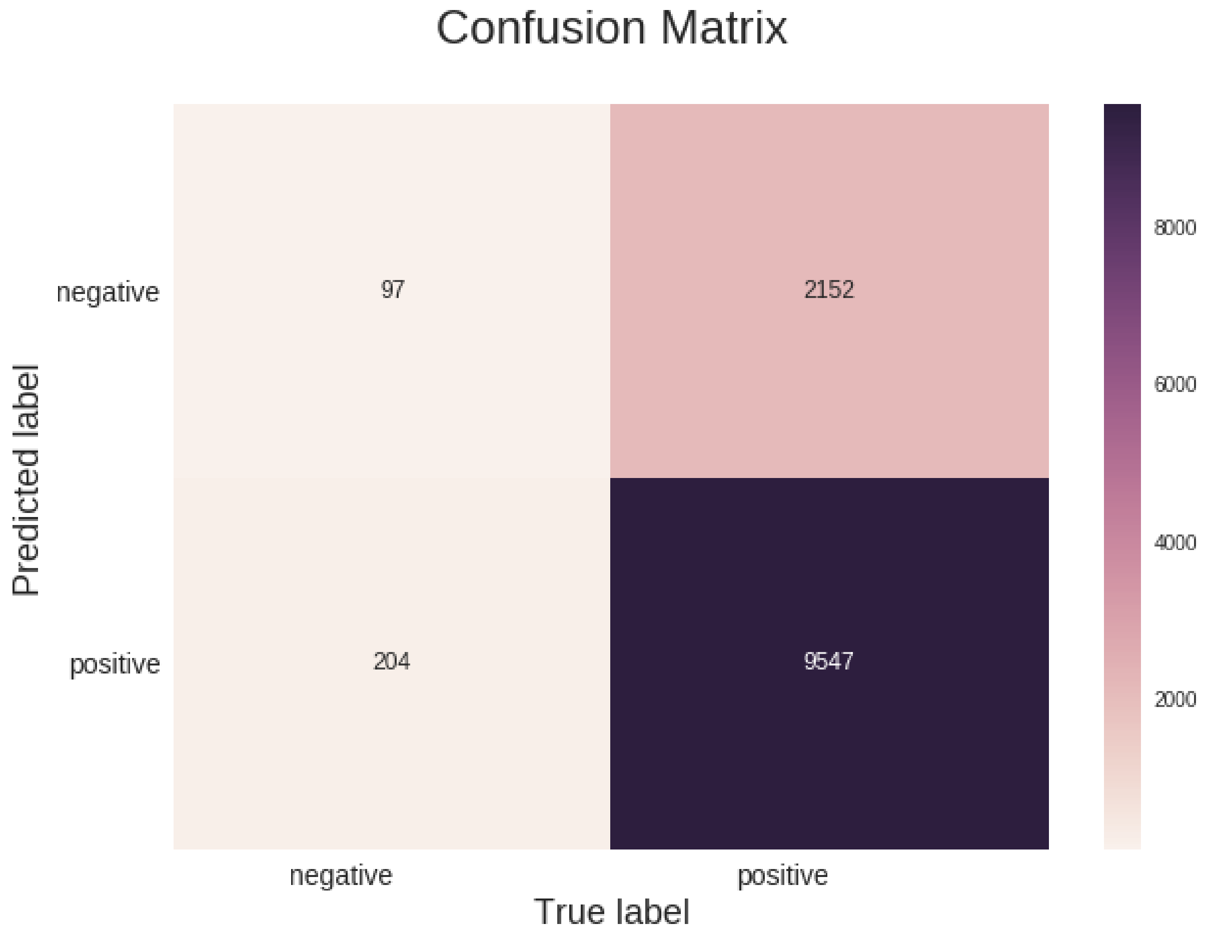
```
acc = accuracy_score(Y_test, predictions) * 100
print('\nThe Test Accuracy of the DecisionTreeClassifier for depth = %d
 is %f%%' % (optimal_depth, acc))
print('\nThe Test Accuracy of the DecisionTreeClassifier for depth = %d
 is %f%%' % (optimal_learners, acc))

# evaluate precision
acc = precision_score(Y_test, predictions, pos_label = 1)
print('\nThe Test Precision of the DecisionTreeClassifier for depth = %
d is %f' % (optimal_depth, acc))
print('\nThe Test Precision of the DecisionTreeClassifier for depth = %
d is %f' % (optimal_learners, acc))

# evaluate recall
acc = recall_score(Y_test, predictions, pos_label = 1)
print('\nThe Test Recall of the DecisionTreeClassifier for depth = %d i
s %f' % (optimal_depth, acc))
print('\nThe Test Recall of the DecisionTreeClassifier for depth = %d i
s %f' % (optimal_learners, acc))

# evaluate f1-score
acc = f1_score(Y_test, predictions, pos_label = 1)
print('\nThe Test F1-Score of the DecisionTreeClassifier for depth = %d
 is %f' % (optimal_depth, acc))
print('\nThe Test F1-Score of the DecisionTreeClassifier for depth = %d
 is %f' % (optimal_learners, acc))
```

```
The Test Accuracy of the DecisionTreeClassifier for depth = 10 is 80.36
6667%

The Test Accuracy of the DecisionTreeClassifier for depth = 500 is 80.3
66667%

The Test Precision of the DecisionTreeClassifier for depth = 10 is 0.81
6053

The Test Precision of the DecisionTreeClassifier for depth = 500 is 0.8
16053

The Test Recall of the DecisionTreeClassifier for depth = 10 is 0.97907
```

```
9

The Test Recall of the DecisionTreeClassifier for depth = 500 is 0.9790
79

The Test F1-Score of the DecisionTreeClassifier for depth = 10 is 0.890
163

The Test F1-Score of the DecisionTreeClassifier for depth = 500 is 0.89
0163
```

In [55]:

```
# Code for drawing seaborn heatmaps on test data
class_names = ['negative','positive']
df_heatmap = pd.DataFrame(confusion_matrix(Y_test, predictions), index=
class_names, columns=class_names )
fig = plt.figure(figsize=(10,7))
heatmap = sns.heatmap(df_heatmap, annot=True, fmt="d")

# Setting tick labels for heatmap
heatmap.yaxis.set_ticklabels(heatmap.yaxis.get_ticklabels(), rotation=0
, ha='right', fontsize=14)
heatmap.xaxis.set_ticklabels(heatmap.xaxis.get_ticklabels(), rotation=0
, ha='right', fontsize=14)
plt.ylabel('Predicted label',size=18)
plt.xlabel('True label',size=18)
plt.title("Confusion Matrix\n",size=24)
plt.show()
```



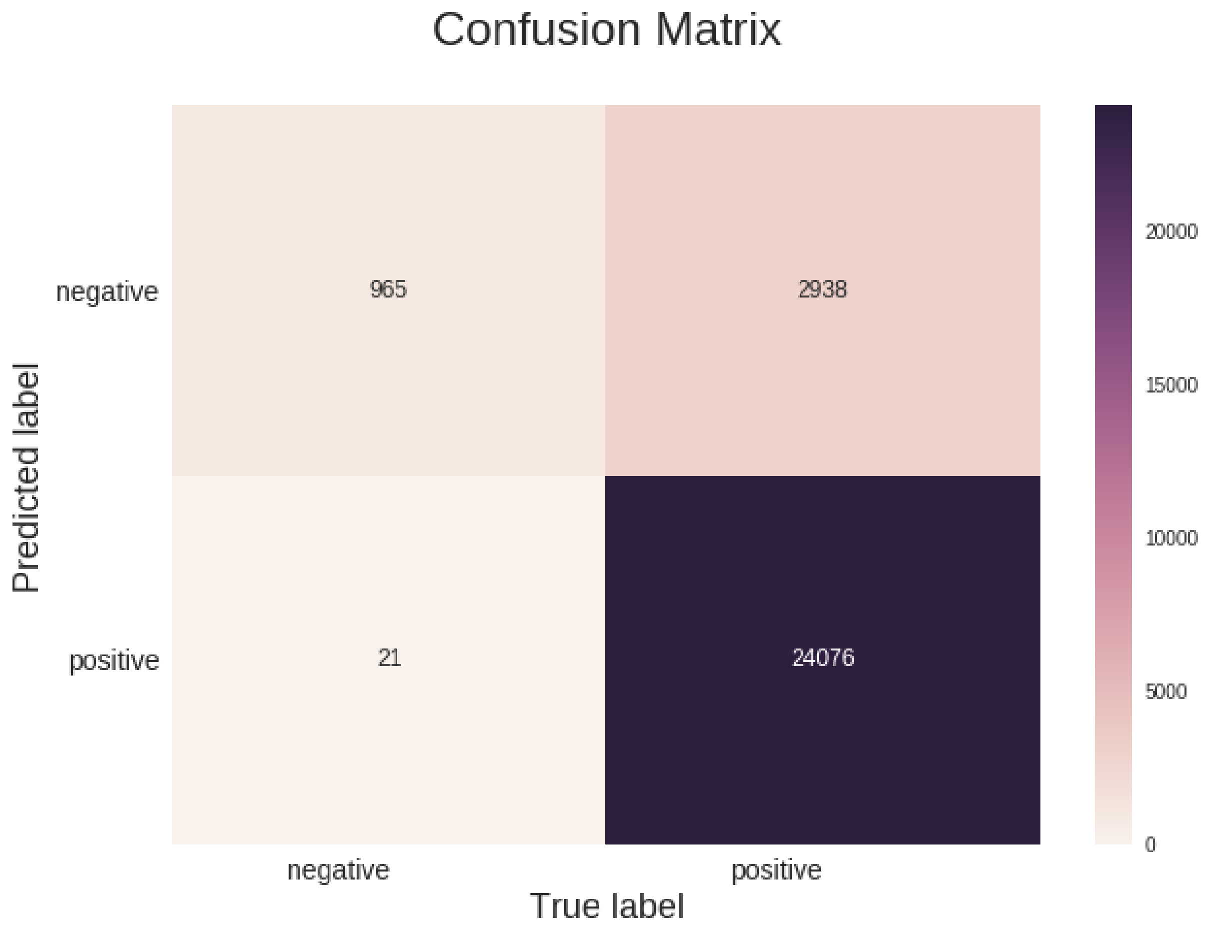
In [56]:
```
# evaluate accuracy on train data
acc = accuracy_score(Y_train, predictions1) * 100
print('\nThe Train Accuracy of the DecisionTreeClassifier for depth = %
d is %f%%' % (optimal_depth, acc))
print('\nThe Train Accuracy of the DecisionTreeClassifier for depth = %
d is %f%%' % (optimal_learners, acc))

# evaluate precision
acc = precision_score(Y_train, predictions1, pos_label = 1)
print('\nThe Train Precision of the DecisionTreeClassifier for depth =
```

```
%d is %f' % (optimal_depth, acc))
print('\nThe Train Precision of the DecisionTreeClassifier for depth = %d is %f' % (optimal_learners, acc))

# evaluate recall
acc = recall_score(Y_train, predictions1, pos_label = 1)
print('\nThe Train Recall of the DecisionTreeClassifier for depth = %d is %f' % (optimal_depth, acc))
print('\nThe Train Recall of the DecisionTreeClassifier for depth = %d is %f' % (optimal_learners, acc))


# evaluate f1-score
acc = f1_score(Y_train, predictions1, pos_label = 1)
print('\nThe Train F1-Score of the DecisionTreeClassifier for depth = %d is %f' % (optimal_depth, acc))
print('\nThe Train F1-Score of the DecisionTreeClassifier for depth = %d is %f' % (optimal_learners, acc))
```

```
The Train Accuracy of the DecisionTreeClassifier for depth = 10 is 89.432143%

The Train Accuracy of the DecisionTreeClassifier for depth = 500 is 89.432143%

The Train Precision of the DecisionTreeClassifier for depth = 10 is 0.891242

The Train Precision of the DecisionTreeClassifier for depth = 500 is 0.891242

The Train Recall of the DecisionTreeClassifier for depth = 10 is 0.999129

The Train Recall of the DecisionTreeClassifier for depth = 500 is 0.999129

The Train F1-Score of the DecisionTreeClassifier for depth = 10 is 0.942106
```

```
The Train F1-Score of the DecisionTreeClassifier for depth = 500 is 0.9
42106
```

In [57]:
```
# Code for drawing seaborn heatmaps
class_names = ['negative','positive']
df_heatmap = pd.DataFrame(confusion_matrix(Y_train, predictions1), inde
x=class_names, columns=class_names )
fig = plt.figure(figsize=(10,7))
heatmap = sns.heatmap(df_heatmap, annot=True, fmt="d")

# Setting tick labels for heatmap
heatmap.yaxis.set_ticklabels(heatmap.yaxis.get_ticklabels(), rotation=0
, ha='right', fontsize=14)
heatmap.xaxis.set_ticklabels(heatmap.xaxis.get_ticklabels(), rotation=0
, ha='right', fontsize=14)
plt.ylabel('Predicted label',size=18)
plt.xlabel('True label',size=18)
plt.title("Confusion Matrix\n",size=24)
plt.show()
```



[5.2] Applying GBDT using XGBOOST

In [0]:
```
from sklearn.model_selection import train_test_split
##Sorting data according to Time in ascending order for Time Based Spli
tting
time_sorted_data = final.sort_values('Time', axis=0, ascending=True, in
place=False, kind='quicksort', na_position='last')

x = time_sorted_data['CleanedText'].values
```

```
y = time_sorted_data['Score']

# split the data set into train and test
X_train, X_test, Y_train, Y_test = train_test_split(x, y, test_size=0.3
, random_state=0,shuffle=False)
```

In [66]:
```
# Please write all the code with proper documentation
#BoW
count_vect = CountVectorizer(min_df = 1000)
X_train_vec = count_vect.fit_transform(X_train)
X_test_vec = count_vect.transform(X_test)
print("the type of count vectorizer :",type(X_train_vec))
print("the shape of out text BOW vectorizer : ",X_train_vec.get_shape
())
print("the number of unique words :", X_train_vec.get_shape()[1])
```

```
the type of count vectorizer : <class 'scipy.sparse.csr.csr_matrix'>
the shape of out text BOW vectorizer :  (28000, 172)
the number of unique words : 172
```

In [0]:
```
from sklearn.preprocessing import StandardScaler
sc = StandardScaler(with_mean=False)
X_train_vec_standardized = sc.fit_transform(X_train_vec)
X_test_vec_standardized = sc.transform(X_test_vec)
```

[5.2.1] Applying XGBOOST on BOW, SET 1

In [68]:
```
# Please write all the code with proper documentation
from sklearn.ensemble import GradientBoostingClassifier

base_learners = [5, 10, 50, 100, 200, 500, 1000]
depth = [2, 3, 4, 5, 6, 7, 8, 9, 10]

param_grid = {'n_estimators': base_learners,  'max_depth': depth}
GBC = GradientBoostingClassifier(max_features='sqrt',subsample=0.1)
model = GridSearchCV(GBC, param_grid, scoring = 'roc_auc', cv=3, n_jobs
 = -1,pre_dispatch=2)
```

```
model.fit(X_train_vec_standardized, Y_train)
print("Model with best parameters :\n",model.best_estimator_)
print("Accuracy of the model : ",model.score(X_test_vec_standardized, Y
_test))
```

```
Model with best parameters :
 GradientBoostingClassifier(criterion='friedman_mse', init=None,
              learning_rate=0.1, loss='deviance', max_depth=2,
              max_features='sqrt', 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=500,
              n_iter_no_change=None, presort='auto', random_state=None,
              subsample=0.1, tol=0.0001, validation_fraction=0.1,
              verbose=0, warm_start=False)
Accuracy of the model :  0.8385698056803377
```

In [0]:

```
gb = GradientBoostingClassifier(n_estimators=optimal_learners, max_dept
h=optimal_depth, max_features='sqrt', subsample=0.1)
gb.fit(X_train_vec_standardized,Y_train)
prediction = gb.predict(X_test_vec_standardized)
prediction1 = gb.predict(X_train_vec_standardized)
```

In [73]:

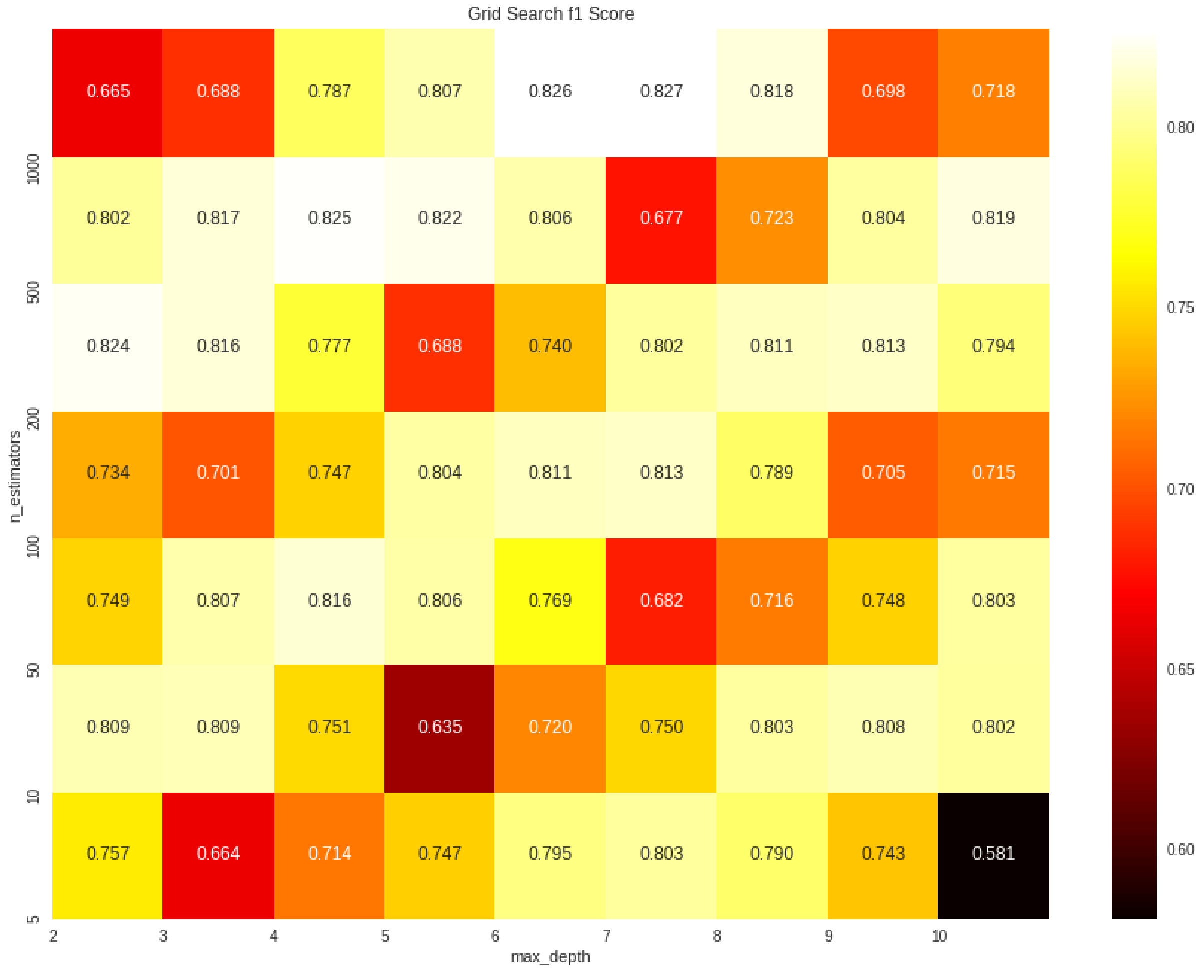
```
# Cross-Validation errors
cv_errors = [1-i for i in model.cv_results_['mean_test_score']]
training_scores=[1-i for i in model.cv_results_['mean_train_score']]

# Optimal value of number of base learners
optimal_learners = model.best_estimator_.n_estimators
print("The optimal number of base learners is : ",optimal_learners)


optimal_depth=model.best_estimator_.max_depth
print("The optimal number of depth is : ",optimal_depth)

# Variables that will be used for  making table in Conclusion part of t
his assignment
bow_gbdt_learners = optimal_learners
bow_gbdt_depth = optimal_depth
```

```
bow_gbdt_train_acc = model.score(X_test_vec, Y_test)*100
bow_gbdt_test_acc = accuracy_score(Y_test, predictions) * 100
```

```
The optimal number of base learners is :  500
The optimal number of depth is :  2
```

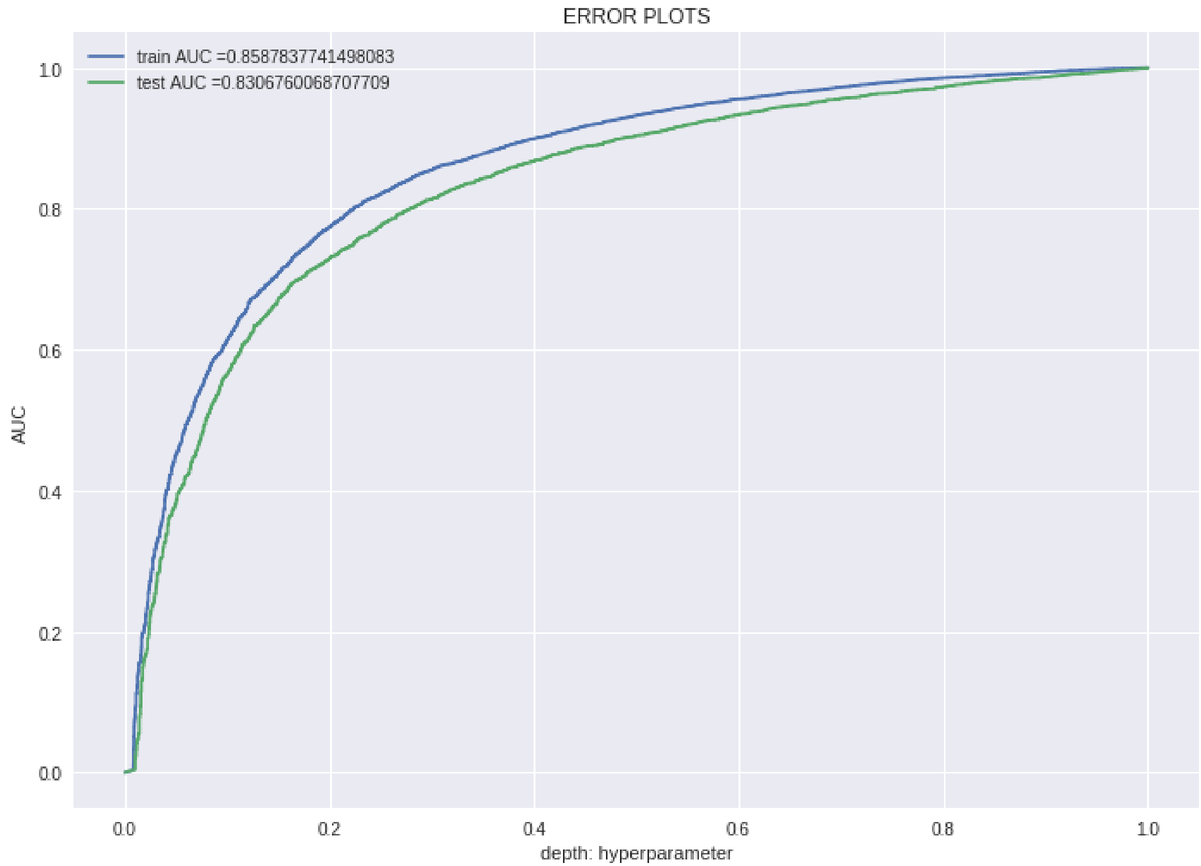
In [74]:

```
scores = model.cv_results_['mean_test_score'].reshape(len(base_learners
),len(depth))

plt.figure(figsize=(15, 11))
sns.heatmap(scores, annot=True, cmap=plt.cm.hot, fmt=".3f", xticklabels
=base_learners, yticklabels=depth)
plt.ylabel('n_estimators')
plt.xlabel('max_depth')
plt.yticks(np.arange(len(base_learners)), base_learners)
plt.xticks(np.arange(len(depth)), depth)
plt.title('Grid Search f1 Score')
plt.show()
```



In [75]:
```
train_fpr, train_tpr, thresholds = roc_curve(Y_train, gb.predict_proba(
X_train_vec_standardized)[:,1])
test_fpr, test_tpr, thresholds = roc_curve(Y_test, gb.predict_proba(X_t
est_vec_standardized)[:,1])

plt.plot(train_fpr, train_tpr, label="train AUC ="+str(auc(train_fpr, t
rain_tpr)))
plt.plot(test_fpr, test_tpr, label="test AUC ="+str(auc(test_fpr, test_
tpr)))
```

```
plt.legend()
plt.xlabel("depth: hyperparameter")
plt.ylabel("AUC")
plt.title("ERROR PLOTS")
plt.show()
```



In [76]:

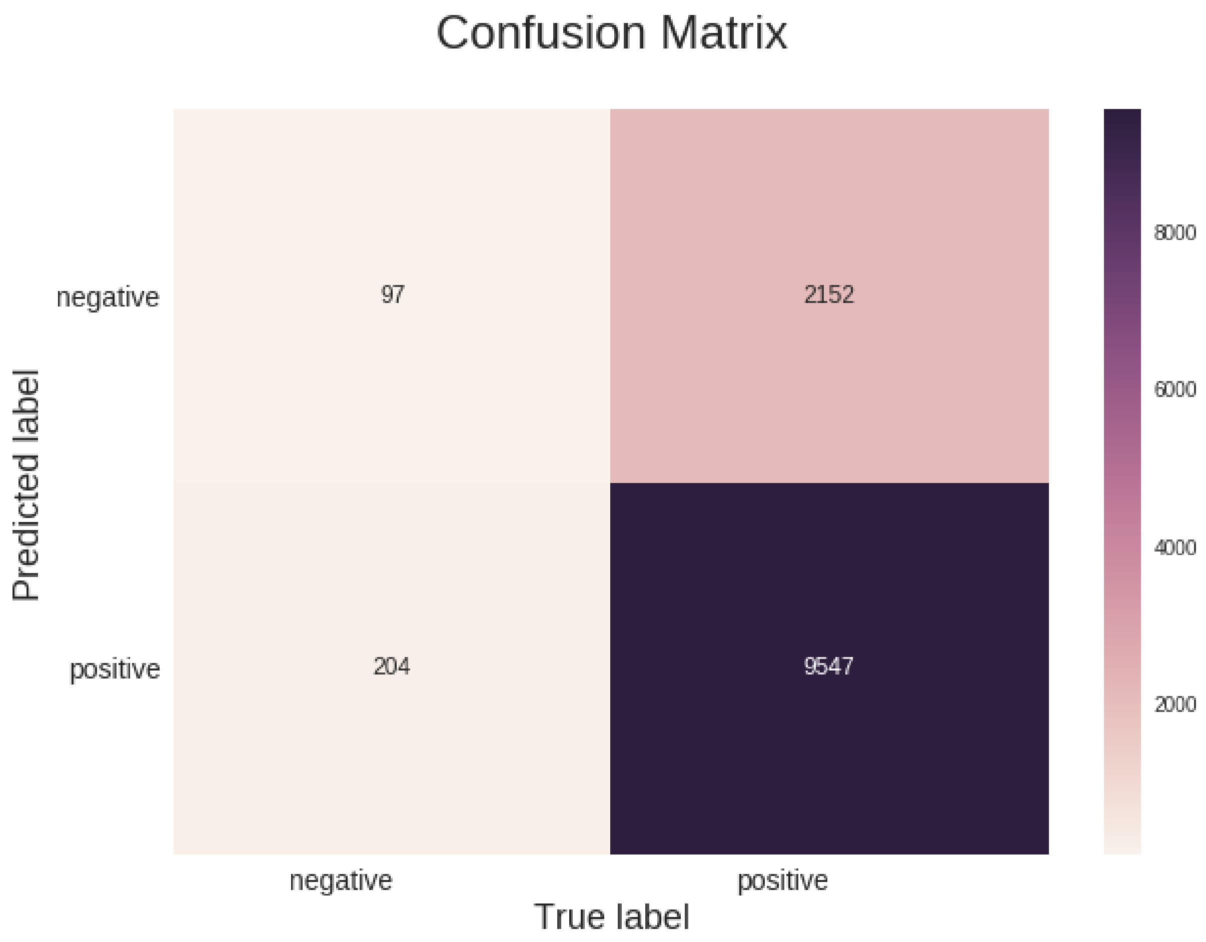
```
# evaluate accuracy on test data
acc = accuracy_score(Y_test, predictions) * 100
print('\nThe Test Accuracy of the DecisionTreeClassifier for depth = %d
 is %f%%' % (optimal_depth, acc))
print('\nThe Test Accuracy of the DecisionTreeClassifier for depth = %d
 is %f%%' % (optimal_learners, acc))
```

```python
# evaluate precision
acc = precision_score(Y_test, predictions, pos_label = 1)
print('\nThe Test Precision of the DecisionTreeClassifier for depth = %d is %f' % (optimal_depth, acc))
print('\nThe Test Precision of the DecisionTreeClassifier for depth = %d is %f' % (optimal_learners, acc))

# evaluate recall
acc = recall_score(Y_test, predictions, pos_label = 1)
print('\nThe Test Recall of the DecisionTreeClassifier for depth = %d is %f' % (optimal_depth, acc))
print('\nThe Test Recall of the DecisionTreeClassifier for depth = %d is %f' % (optimal_learners, acc))

# evaluate f1-score
acc = f1_score(Y_test, predictions, pos_label = 1)
print('\nThe Test F1-Score of the DecisionTreeClassifier for depth = %d is %f' % (optimal_depth, acc))
print('\nThe Test F1-Score of the DecisionTreeClassifier for depth = %d is %f' % (optimal_learners, acc))
```

```
The Test Accuracy of the DecisionTreeClassifier for depth = 2 is 80.366667%

The Test Accuracy of the DecisionTreeClassifier for depth = 500 is 80.366667%

The Test Precision of the DecisionTreeClassifier for depth = 2 is 0.816053

The Test Precision of the DecisionTreeClassifier for depth = 500 is 0.816053

The Test Recall of the DecisionTreeClassifier for depth = 2 is 0.979079

The Test Recall of the DecisionTreeClassifier for depth = 500 is 0.979079

The Test F1-Score of the DecisionTreeClassifier for depth = 2 is 0.890163
```

```
The Test F1-Score of the DecisionTreeClassifier for depth = 500 is 0.89
0163
```

In [77]:
```
# Code for drawing seaborn heatmaps on test data
class_names = ['negative','positive']
df_heatmap = pd.DataFrame(confusion_matrix(Y_test, predictions), index=
class_names, columns=class_names )
fig = plt.figure(figsize=(10,7))
heatmap = sns.heatmap(df_heatmap, annot=True, fmt="d")

# Setting tick labels for heatmap
heatmap.yaxis.set_ticklabels(heatmap.yaxis.get_ticklabels(), rotation=0
, ha='right', fontsize=14)
heatmap.xaxis.set_ticklabels(heatmap.xaxis.get_ticklabels(), rotation=0
, ha='right', fontsize=14)
plt.ylabel('Predicted label',size=18)
plt.xlabel('True label',size=18)
plt.title("Confusion Matrix\n",size=24)
plt.show()
```



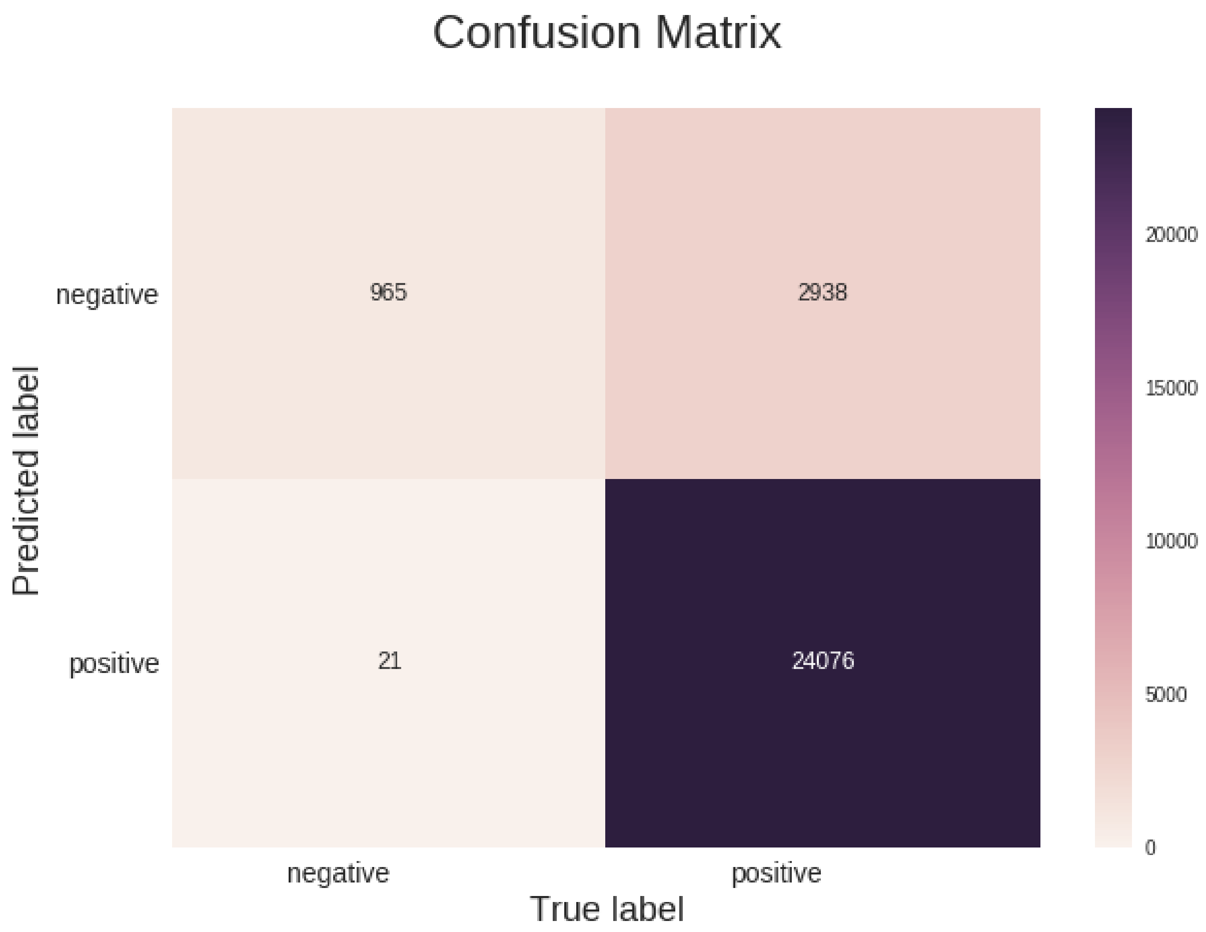
In [78]:
```
# evaluate accuracy on train data
acc = accuracy_score(Y_train, predictions1) * 100
print('\nThe Train Accuracy of the DecisionTreeClassifier for depth = %
d is %f%%' % (optimal_depth, acc))
print('\nThe Train Accuracy of the DecisionTreeClassifier for depth = %
d is %f%%' % (optimal_learners, acc))

# evaluate precision
acc = precision_score(Y_train, predictions1, pos_label = 1)
print('\nThe Train Precision of the DecisionTreeClassifier for depth =
```

```
%d is %f' % (optimal_depth, acc))
print('\nThe Train Precision of the DecisionTreeClassifier for depth = %d is %f' % (optimal_learners, acc))

# evaluate recall
acc = recall_score(Y_train, predictions1, pos_label = 1)
print('\nThe Train Recall of the DecisionTreeClassifier for depth = %d is %f' % (optimal_depth, acc))
print('\nThe Train Recall of the DecisionTreeClassifier for depth = %d is %f' % (optimal_learners, acc))


# evaluate f1-score
acc = f1_score(Y_train, predictions1, pos_label = 1)
print('\nThe Train F1-Score of the DecisionTreeClassifier for depth = %d is %f' % (optimal_depth, acc))
print('\nThe Train F1-Score of the DecisionTreeClassifier for depth = %d is %f' % (optimal_learners, acc))
```

```
The Train Accuracy of the DecisionTreeClassifier for depth = 2 is 89.432143%

The Train Accuracy of the DecisionTreeClassifier for depth = 500 is 89.432143%

The Train Precision of the DecisionTreeClassifier for depth = 2 is 0.891242

The Train Precision of the DecisionTreeClassifier for depth = 500 is 0.891242

The Train Recall of the DecisionTreeClassifier for depth = 2 is 0.999129

The Train Recall of the DecisionTreeClassifier for depth = 500 is 0.999129

The Train F1-Score of the DecisionTreeClassifier for depth = 2 is 0.942106
```

```
The Train F1-Score of the DecisionTreeClassifier for depth = 500 is 0.9
42106
```

In [79]:
```
# Code for drawing seaborn heatmaps
class_names = ['negative','positive']
df_heatmap = pd.DataFrame(confusion_matrix(Y_train, predictions1), inde
x=class_names, columns=class_names )
fig = plt.figure(figsize=(10,7))
heatmap = sns.heatmap(df_heatmap, annot=True, fmt="d")

# Setting tick labels for heatmap
heatmap.yaxis.set_ticklabels(heatmap.yaxis.get_ticklabels(), rotation=0
, ha='right', fontsize=14)
heatmap.xaxis.set_ticklabels(heatmap.xaxis.get_ticklabels(), rotation=0
, ha='right', fontsize=14)
plt.ylabel('Predicted label',size=18)
plt.xlabel('True label',size=18)
plt.title("Confusion Matrix\n",size=24)
plt.show()
```



[5.2.2] Applying XGBOOST on TFIDF, SET 2

In [80]:
```
# Please write all the code with proper documentation
tf_idf_vect = TfidfVectorizer(min_df=1000)
X_train_vec = tf_idf_vect.fit_transform(X_train)
X_test_vec = tf_idf_vect.transform(X_test)
print("the type of count vectorizer :",type(X_train_vec))
print("the shape of out text TFIDF vectorizer : ",X_train_vec.get_shape
())
```

```
print("the number of unique words :", X_train_vec.get_shape()[1])

# Data-preprocessing: Standardizing the data
sc = StandardScaler(with_mean=False)
X_train_vec_standardized = sc.fit_transform(X_train_vec)
X_test_vec_standardized = sc.transform(X_test_vec)
```

```
the type of count vectorizer : <class 'scipy.sparse.csr.csr_matrix'>
the shape of out text TFIDF vectorizer :  (28000, 172)
the number of unique words : 172
```

In [81]:

```
# Please write all the code with proper documentation
from sklearn.ensemble import GradientBoostingClassifier

base_learners = [5, 10, 50, 100, 200, 500, 1000]
depth = [2, 3, 4, 5, 6, 7, 8, 9, 10]

param_grid = {'n_estimators': base_learners,  'max_depth': depth}
GBC = GradientBoostingClassifier(max_features='sqrt',subsample=0.1)
model = GridSearchCV(GBC, param_grid, scoring = 'roc_auc', cv=3, n_jobs
 = -1,pre_dispatch=2)
model.fit(X_train_vec_standardized, Y_train)
print("Model with best parameters :\n",model.best_estimator_)
print("Accuracy of the model : ",model.score(X_test_vec_standardized, Y
_test))
```

```
Model with best parameters :
 GradientBoostingClassifier(criterion='friedman_mse', init=None,
              learning_rate=0.1, loss='deviance', max_depth=2,
              max_features='sqrt', 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=500,
              n_iter_no_change=None, presort='auto', random_state=None,
              subsample=0.1, tol=0.0001, validation_fraction=0.1,
              verbose=0, warm_start=False)
Accuracy of the model :  0.8293466634449003
```

In [0]:

```
gb = GradientBoostingClassifier(n_estimators=optimal_learners, max_dept
```

```
h=optimal_depth, max_features='sqrt', subsample=0.1)
gb.fit(X_train_vec_standardized,Y_train)
prediction = gb.predict(X_test_vec_standardized)
prediction1 = gb.predict(X_train_vec_standardized)
```

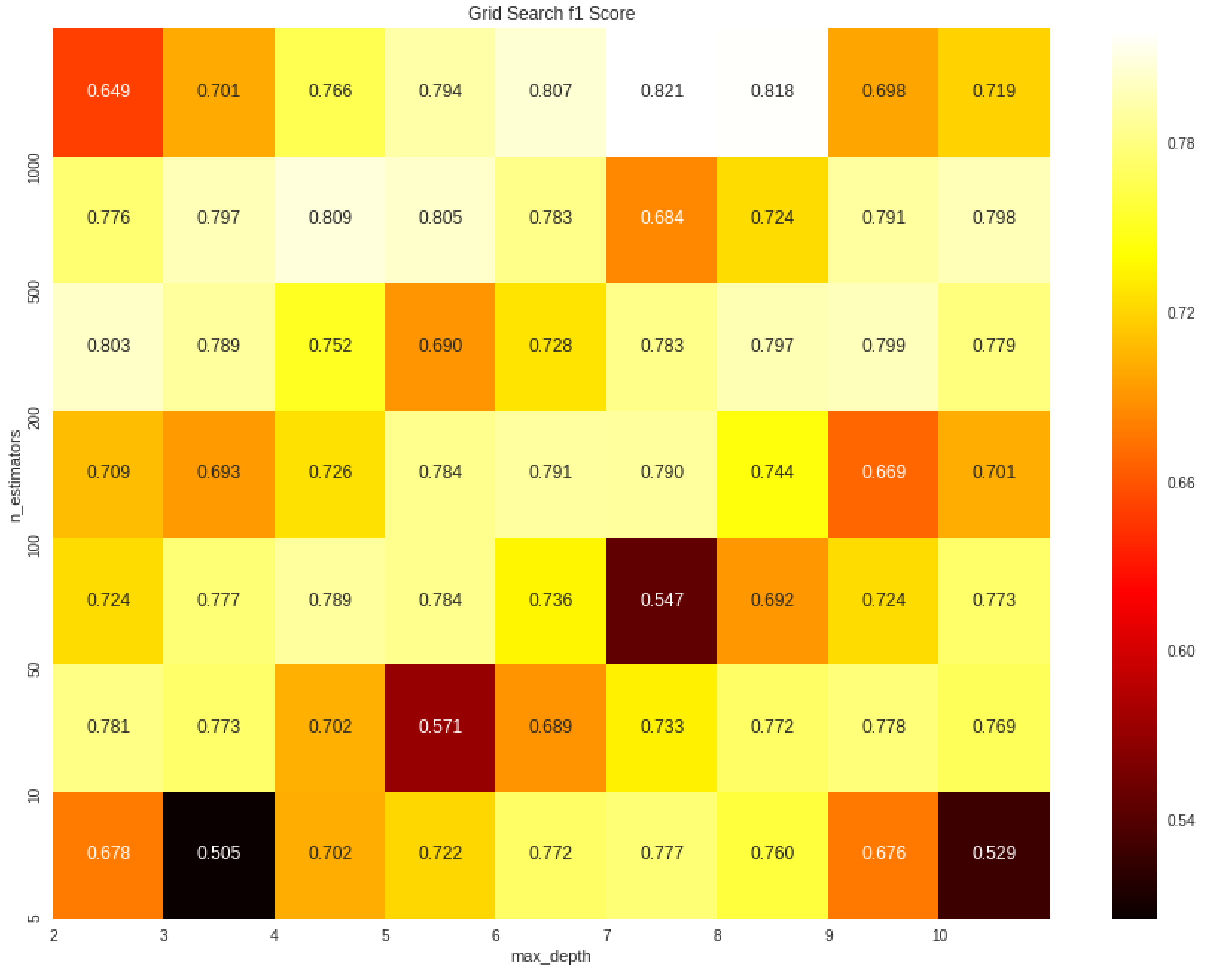
In [84]:

```
# Cross-Validation errors
cv_errors = [1-i for i in model.cv_results_['mean_test_score']]
training_scores=[1-i for i in model.cv_results_['mean_train_score']]

# Optimal value of number of base learners
optimal_learners = model.best_estimator_.n_estimators
print("The optimal number of base learners is : ",optimal_learners)


optimal_depth=model.best_estimator_.max_depth
print("The optimal number of depth is : ",optimal_depth)

# Variables that will be used for  making table in Conclusion part of t
his assignment
tfidf_gbdt_learners = optimal_learners
tfidf_gbdt_depth = optimal_depth
tfidf_gbdt_train_acc = model.score(X_test_vec, Y_test)*100
tfidf_gbdt_test_acc = accuracy_score(Y_test, predictions) * 100
```

```
The optimal number of base learners is :  500
The optimal number of depth is :  2
```

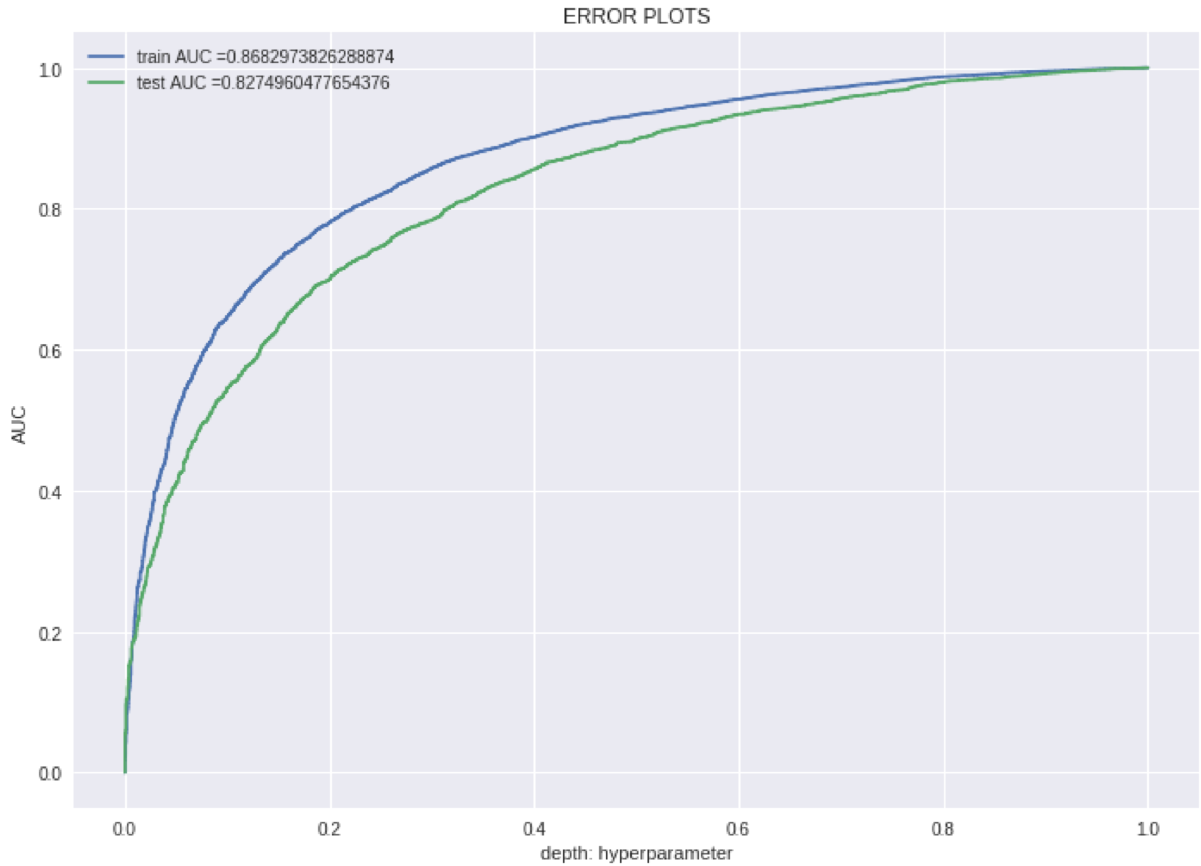
In [85]:

```
scores = model.cv_results_['mean_test_score'].reshape(len(base_learners
),len(depth))

plt.figure(figsize=(15, 11))
sns.heatmap(scores, annot=True, cmap=plt.cm.hot, fmt=".3f", xticklabels
=base_learners, yticklabels=depth)
plt.ylabel('n_estimators')
plt.xlabel('max_depth')
plt.yticks(np.arange(len(base_learners)), base_learners)
plt.xticks(np.arange(len(depth)), depth)
plt.title('Grid Search f1 Score')
plt.show()
```



In [86]:
```python
train_fpr, train_tpr, thresholds = roc_curve(Y_train, gb.predict_proba(X_train_vec_standardized)[:,1])
test_fpr, test_tpr, thresholds = roc_curve(Y_test, gb.predict_proba(X_test_vec_standardized)[:,1])

plt.plot(train_fpr, train_tpr, label="train AUC ="+str(auc(train_fpr, train_tpr)))
plt.plot(test_fpr, test_tpr, label="test AUC ="+str(auc(test_fpr, test_tpr)))
```

```
plt.legend()
plt.xlabel("depth: hyperparameter")
plt.ylabel("AUC")
plt.title("ERROR PLOTS")
plt.show()
```



In [87]:
```
# evaluate accuracy on test data
acc = accuracy_score(Y_test, predictions) * 100
print('\nThe Test Accuracy of the DecisionTreeClassifier for depth = %d
 is %f%%' % (optimal_depth, acc))
print('\nThe Test Accuracy of the DecisionTreeClassifier for depth = %d
 is %f%%' % (optimal_learners, acc))
```

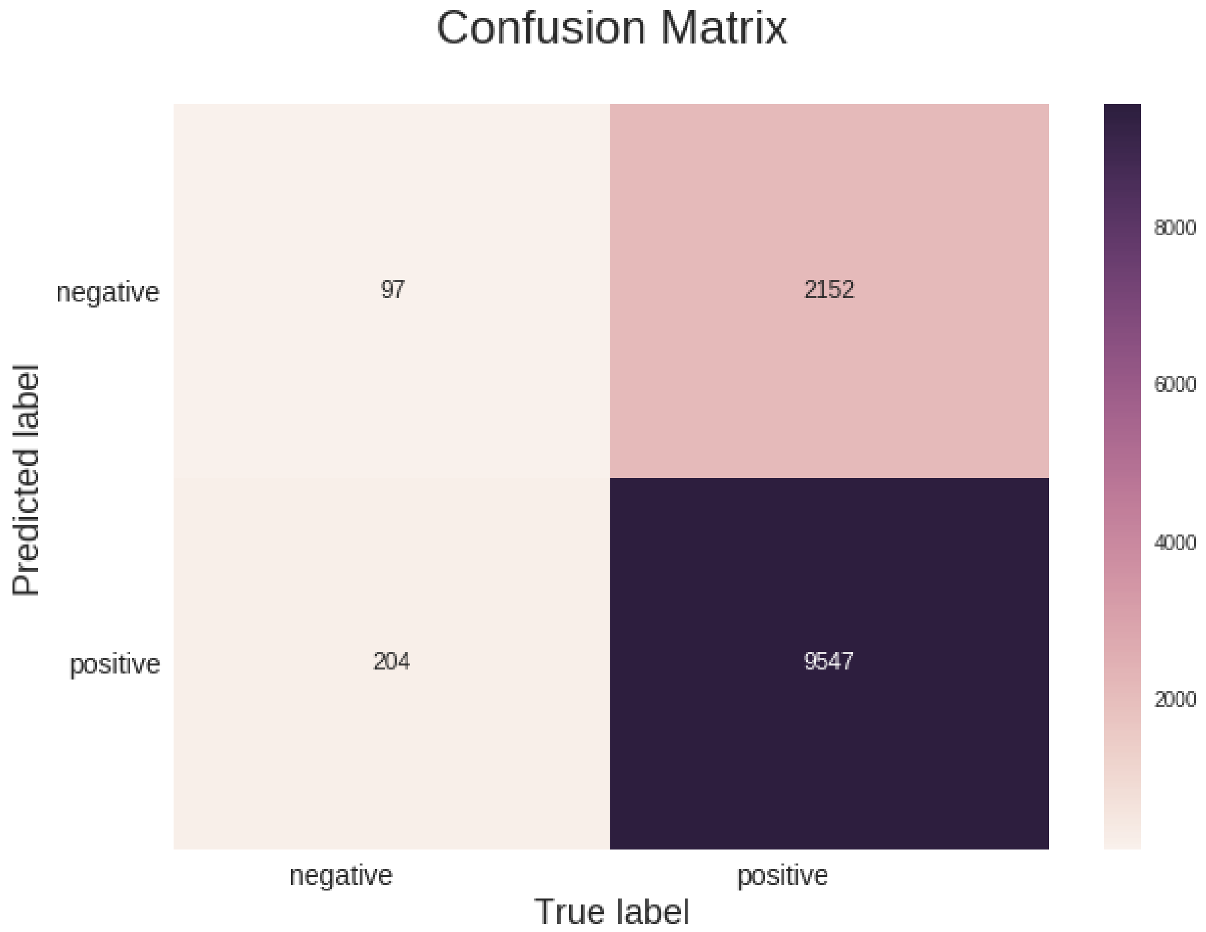
```
# evaluate precision
acc = precision_score(Y_test, predictions, pos_label = 1)
print('\nThe Test Precision of the DecisionTreeClassifier for depth = %
d is %f' % (optimal_depth, acc))
print('\nThe Test Precision of the DecisionTreeClassifier for depth = %
d is %f' % (optimal_learners, acc))

# evaluate recall
acc = recall_score(Y_test, predictions, pos_label = 1)
print('\nThe Test Recall of the DecisionTreeClassifier for depth = %d i
s %f' % (optimal_depth, acc))
print('\nThe Test Recall of the DecisionTreeClassifier for depth = %d i
s %f' % (optimal_learners, acc))

# evaluate f1-score
acc = f1_score(Y_test, predictions, pos_label = 1)
print('\nThe Test F1-Score of the DecisionTreeClassifier for depth = %d
 is %f' % (optimal_depth, acc))
print('\nThe Test F1-Score of the DecisionTreeClassifier for depth = %d
 is %f' % (optimal_learners, acc))
```

```
The Test Accuracy of the DecisionTreeClassifier for depth = 2 is 80.366
667%

The Test Accuracy of the DecisionTreeClassifier for depth = 500 is 80.3
66667%

The Test Precision of the DecisionTreeClassifier for depth = 2 is 0.816
053

The Test Precision of the DecisionTreeClassifier for depth = 500 is 0.8
16053

The Test Recall of the DecisionTreeClassifier for depth = 2 is 0.979079

The Test Recall of the DecisionTreeClassifier for depth = 500 is 0.9790
79

The Test F1-Score of the DecisionTreeClassifier for depth = 2 is 0.8901
63
```

```
The Test F1-Score of the DecisionTreeClassifier for depth = 500 is 0.89
0163
```

In [88]:

```
# Code for drawing seaborn heatmaps on test data
class_names = ['negative','positive']
df_heatmap = pd.DataFrame(confusion_matrix(Y_test, predictions), index=
class_names, columns=class_names )
fig = plt.figure(figsize=(10,7))
heatmap = sns.heatmap(df_heatmap, annot=True, fmt="d")

# Setting tick labels for heatmap
heatmap.yaxis.set_ticklabels(heatmap.yaxis.get_ticklabels(), rotation=0
, ha='right', fontsize=14)
heatmap.xaxis.set_ticklabels(heatmap.xaxis.get_ticklabels(), rotation=0
, ha='right', fontsize=14)
plt.ylabel('Predicted label',size=18)
plt.xlabel('True label',size=18)
plt.title("Confusion Matrix\n",size=24)
plt.show()
```



In [89]:
```
# evaluate accuracy on train data
acc = accuracy_score(Y_train, predictions1) * 100
print('\nThe Train Accuracy of the DecisionTreeClassifier for depth = %
d is %f%%' % (optimal_depth, acc))
print('\nThe Train Accuracy of the DecisionTreeClassifier for depth = %
d is %f%%' % (optimal_learners, acc))

# evaluate precision
acc = precision_score(Y_train, predictions1, pos_label = 1)
print('\nThe Train Precision of the DecisionTreeClassifier for depth =
```

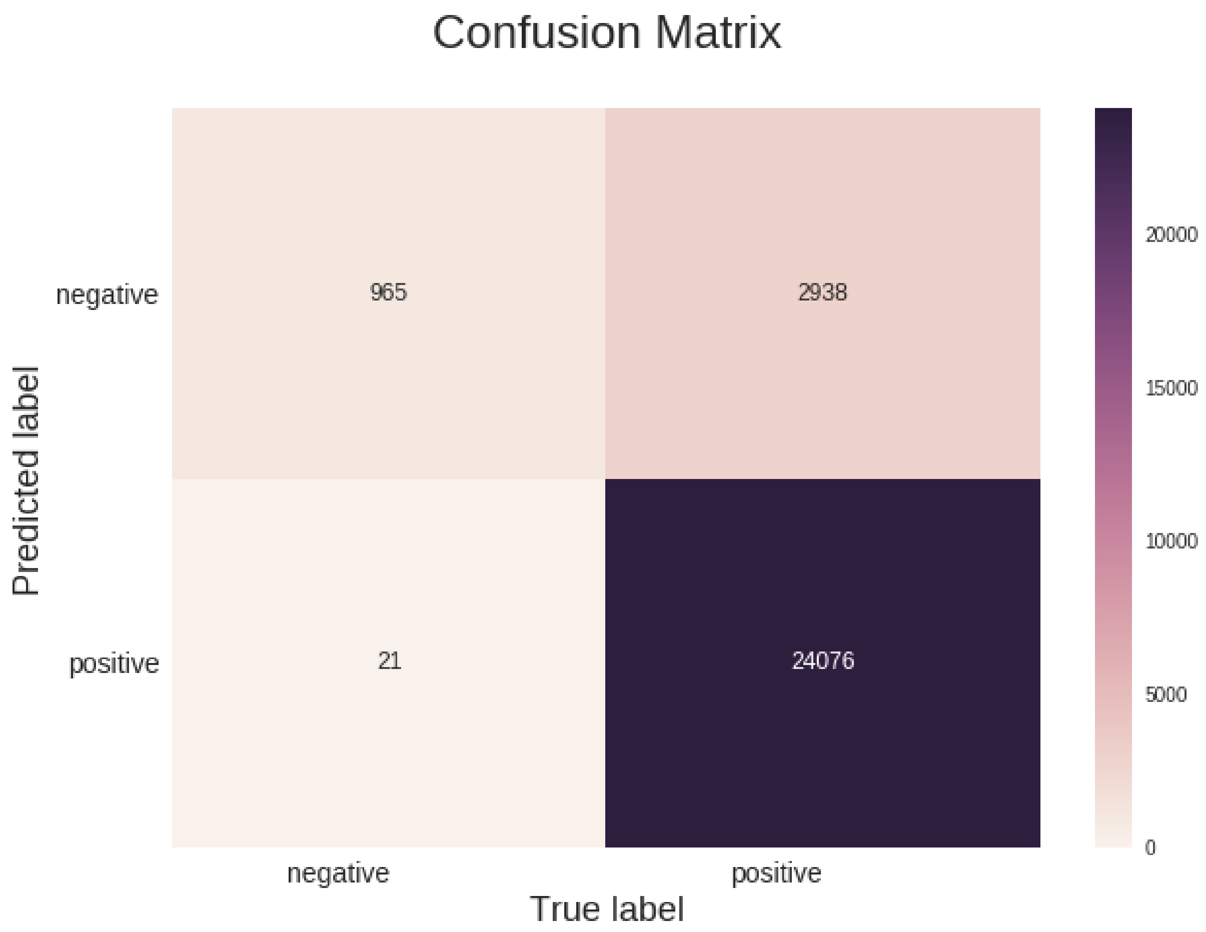
```
%d is %f' % (optimal_depth, acc))
print('\nThe Train Precision of the DecisionTreeClassifier for depth =
%d is %f' % (optimal_learners, acc))

# evaluate recall
acc = recall_score(Y_train, predictions1, pos_label = 1)
print('\nThe Train Recall of the DecisionTreeClassifier for depth = %d
 is %f' % (optimal_depth, acc))
print('\nThe Train Recall of the DecisionTreeClassifier for depth = %d
 is %f' % (optimal_learners, acc))


# evaluate f1-score
acc = f1_score(Y_train, predictions1, pos_label = 1)
print('\nThe Train F1-Score of the DecisionTreeClassifier for depth = %
d is %f' % (optimal_depth, acc))
print('\nThe Train F1-Score of the DecisionTreeClassifier for depth = %
d is %f' % (optimal_learners, acc))
```

```
The Train Accuracy of the DecisionTreeClassifier for depth = 2 is 89.43
2143%

The Train Accuracy of the DecisionTreeClassifier for depth = 500 is 89.
432143%

The Train Precision of the DecisionTreeClassifier for depth = 2 is 0.89
1242

The Train Precision of the DecisionTreeClassifier for depth = 500 is 0.
891242

The Train Recall of the DecisionTreeClassifier for depth = 2 is 0.99912
9

The Train Recall of the DecisionTreeClassifier for depth = 500 is 0.999
129

The Train F1-Score of the DecisionTreeClassifier for depth = 2 is 0.942
106
```

```
The Train F1-Score of the DecisionTreeClassifier for depth = 500 is 0.9
42106
```

In [90]:

```
# Code for drawing seaborn heatmaps
class_names = ['negative','positive']
df_heatmap = pd.DataFrame(confusion_matrix(Y_train, predictions1), inde
x=class_names, columns=class_names )
fig = plt.figure(figsize=(10,7))
heatmap = sns.heatmap(df_heatmap, annot=True, fmt="d")

# Setting tick labels for heatmap
heatmap.yaxis.set_ticklabels(heatmap.yaxis.get_ticklabels(), rotation=0
, ha='right', fontsize=14)
heatmap.xaxis.set_ticklabels(heatmap.xaxis.get_ticklabels(), rotation=0
, ha='right', fontsize=14)
plt.ylabel('Predicted label',size=18)
plt.xlabel('True label',size=18)
plt.title("Confusion Matrix\n",size=24)
plt.show()
```



[5.2.3] Applying XGBOOST on AVG W2V, SET 3

In [91]:
```
# Please write all the code with proper documentation
# List of sentence in X_train text
sent_of_train=[]
for sent in X_train:
    sent_of_train.append(sent.split())

# List of sentence in X_est text
```

```
sent_of_test=[]
for sent in X_test:
    sent_of_test.append(sent.split())

# Train your own Word2Vec model using your own train text corpus
# min_count = 5 considers only words that occured atleast 5 times
w2v_model=Word2Vec(sent_of_train,min_count=5,size=50, workers=4)

w2v_words = list(w2v_model.wv.vocab)
print("number of words that occured minimum 5 times ",len(w2v_words))
```

```
number of words that occured minimum 5 times  7799
```

In [0]:

```
# compute average word2vec for each review for X_train .
train_vectors = [];
for sent in sent_of_train:
    sent_vec = np.zeros(50)
    cnt_words =0;
    for word in sent: #
        if word in w2v_words:
            vec = w2v_model.wv[word]
            sent_vec += vec
            cnt_words += 1
    if cnt_words != 0:
        sent_vec /= cnt_words
    train_vectors.append(sent_vec)
```

In [0]:

```
# compute average word2vec for each review for X_test .
test_vectors = [];
for sent in sent_of_test:
    sent_vec = np.zeros(50)
    cnt_words =0;
    for word in sent: #
        if word in w2v_words:
            vec = w2v_model.wv[word]
            sent_vec += vec
            cnt_words += 1
    if cnt_words != 0:
        sent_vec /= cnt_words
```

```
    test_vectors.append(sent_vec)

X_train_vec = train_vectors
X_test_vec = test_vectors
X_train_vec_standardized = sc.fit_transform(X_train_vec)
X_test_vec_standardized = sc.transform(X_test_vec)
```

In [94]:

```
# Please write all the code with proper documentation
from sklearn.ensemble import GradientBoostingClassifier

base_learners = [5, 10, 50, 100, 200, 500, 1000]
depth = [2, 3, 4, 5, 6, 7, 8, 9, 10]

param_grid = {'n_estimators': base_learners,  'max_depth': depth}
GBC = GradientBoostingClassifier(max_features='sqrt',subsample=0.1)
model = GridSearchCV(GBC, param_grid, scoring = 'roc_auc', cv=3, n_jobs
 = -1,pre_dispatch=2)
model.fit(X_train_vec_standardized, Y_train)
print("Model with best parameters :\n",model.best_estimator_)
print("Accuracy of the model : ",model.score(X_test_vec_standardized, Y
_test))
```

```
Model with best parameters :
 GradientBoostingClassifier(criterion='friedman_mse', init=None,
              learning_rate=0.1, loss='deviance', max_depth=2,
              max_features='sqrt', 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=500,
              n_iter_no_change=None, presort='auto', random_state=None,
              subsample=0.1, tol=0.0001, validation_fraction=0.1,
              verbose=0, warm_start=False)
Accuracy of the model :  0.8554032309805395
```

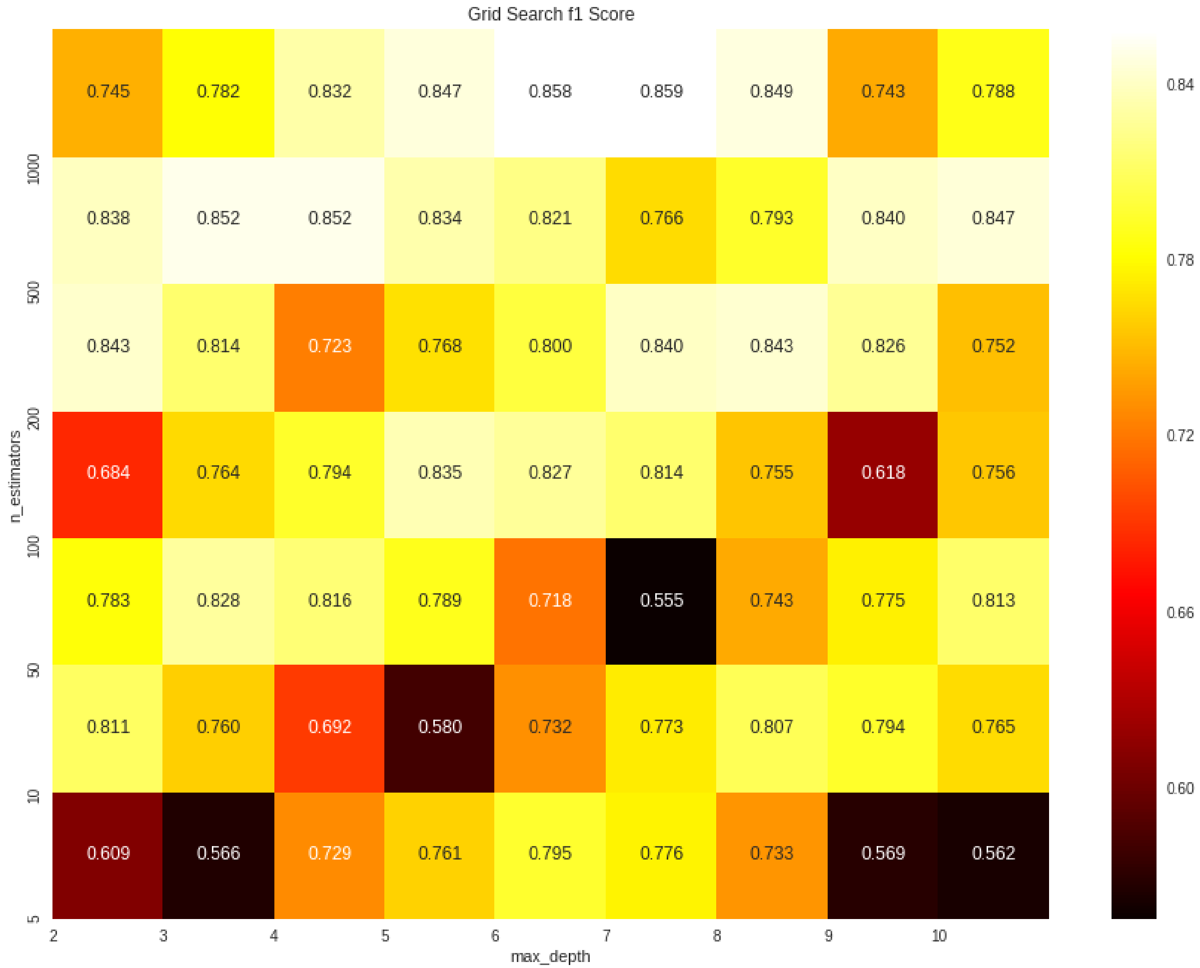
In [0]:

```
gb = GradientBoostingClassifier(n_estimators=optimal_learners, max_dept
h=optimal_depth, max_features='sqrt', subsample=0.1)
gb.fit(X_train_vec_standardized,Y_train)
prediction = gb.predict(X_test_vec_standardized)
prediction1 = gb.predict(X_train_vec_standardized)
```

In [97]:
```
# Cross-Validation errors
cv_errors = [1-i for i in model.cv_results_['mean_test_score']]
training_scores=[1-i for i in model.cv_results_['mean_train_score']]

# Optimal value of number of base learners
optimal_learners = model.best_estimator_.n_estimators
print("The optimal number of base learners is : ",optimal_learners)


optimal_depth=model.best_estimator_.max_depth
print("The optimal number of depth is : ",optimal_depth)

# Variables that will be used for  making table in Conclusion part of t
his assignment
avg_w2v_gbdt_learners = optimal_learners
avg_w2v_gbdt_depth = optimal_depth
avg_w2v_gbdt_train_acc = model.score(X_test_vec, Y_test)*100
avg_w2v_gbdt_test_acc = accuracy_score(Y_test, predictions) * 100
```

```
The optimal number of base learners is :  500
The optimal number of depth is :  2
```

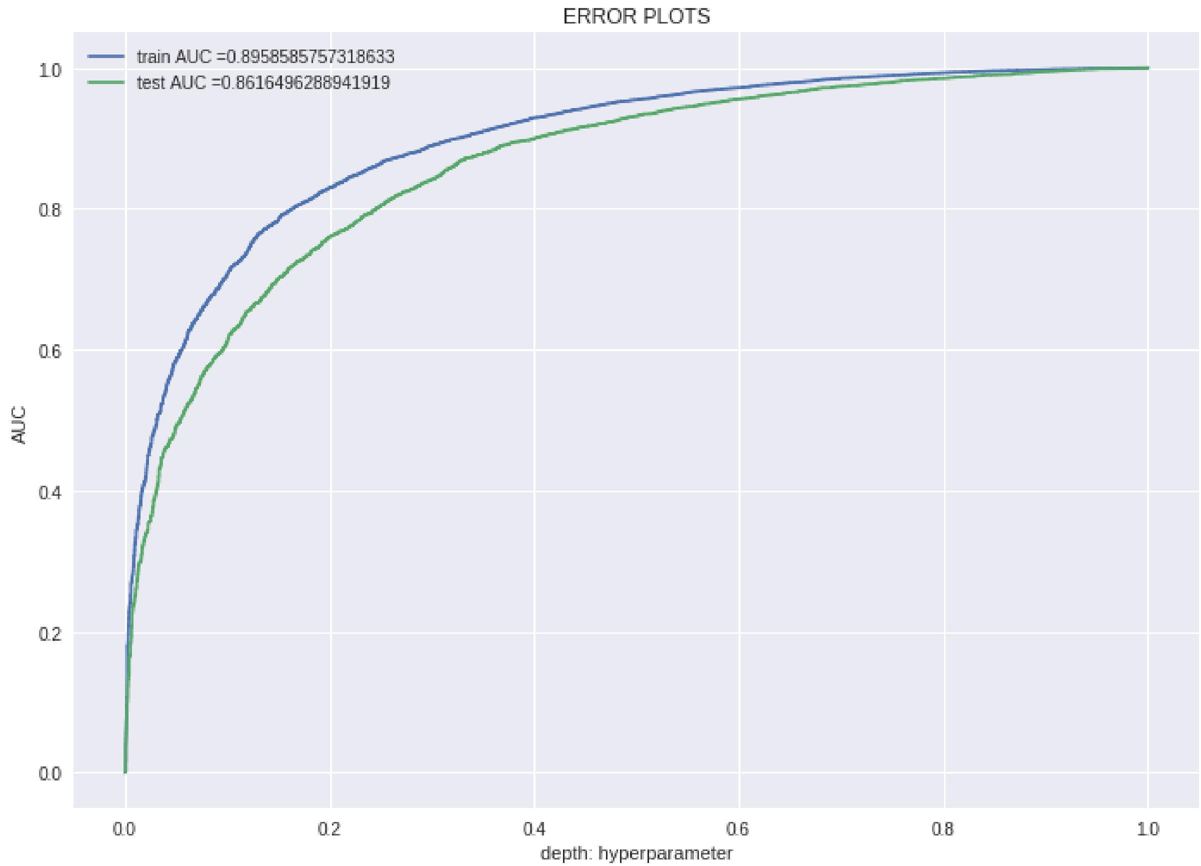
In [98]:
```
scores = model.cv_results_['mean_test_score'].reshape(len(base_learners
),len(depth))

plt.figure(figsize=(15, 11))
sns.heatmap(scores, annot=True, cmap=plt.cm.hot, fmt=".3f", xticklabels
=base_learners, yticklabels=depth)
plt.ylabel('n_estimators')
plt.xlabel('max_depth')
plt.yticks(np.arange(len(base_learners)), base_learners)
plt.xticks(np.arange(len(depth)), depth)
plt.title('Grid Search f1 Score')
plt.show()
```



In [99]:

```
train_fpr, train_tpr, thresholds = roc_curve(Y_train, gb.predict_proba(
X_train_vec_standardized)[:,1])
test_fpr, test_tpr, thresholds = roc_curve(Y_test, gb.predict_proba(X_t
est_vec_standardized)[:,1])

plt.plot(train_fpr, train_tpr, label="train AUC ="+str(auc(train_fpr, t
rain_tpr)))
plt.plot(test_fpr, test_tpr, label="test AUC ="+str(auc(test_fpr, test_
tpr)))
```

```
plt.legend()
plt.xlabel("depth: hyperparameter")
plt.ylabel("AUC")
plt.title("ERROR PLOTS")
plt.show()
```



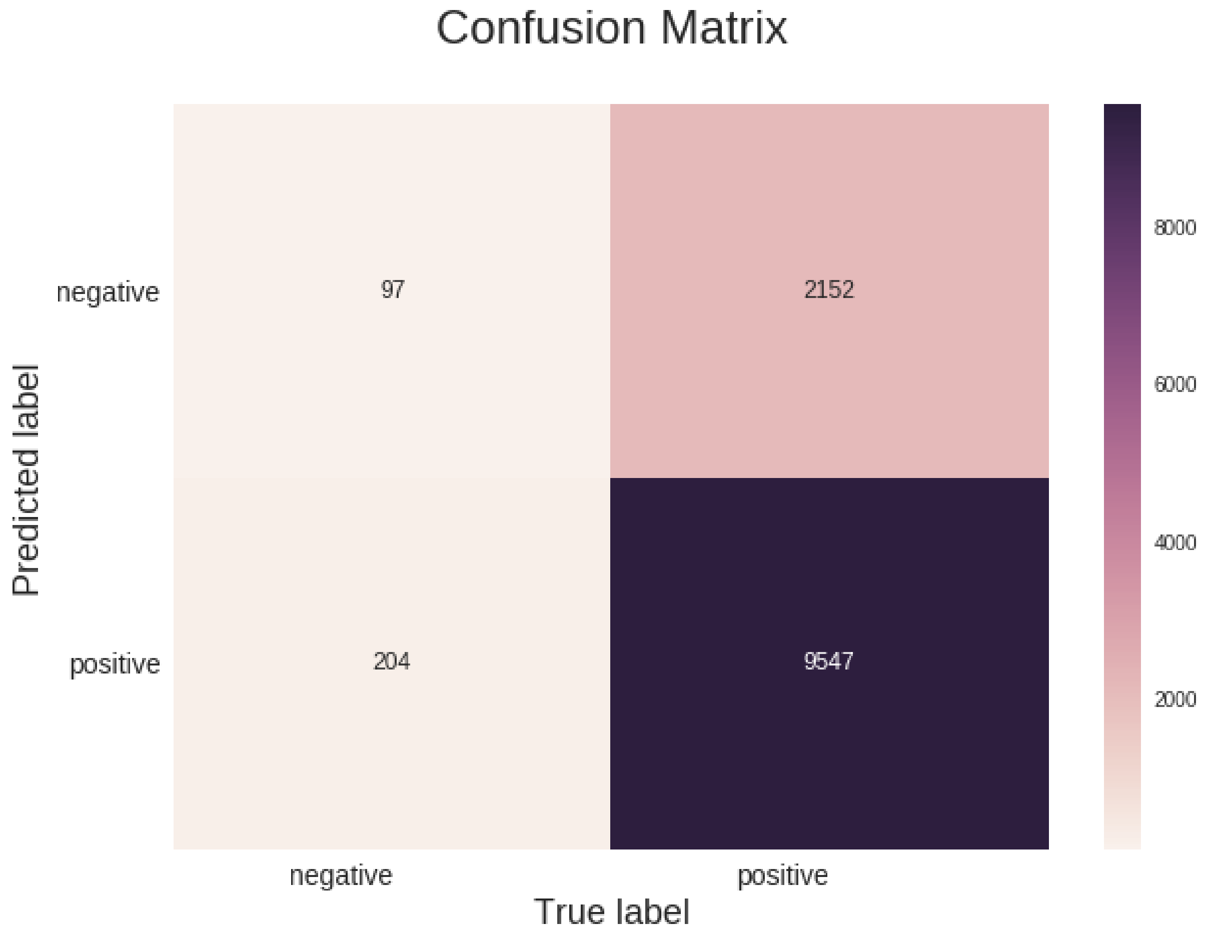
In [100]:
```
# evaluate accuracy on test data
acc = accuracy_score(Y_test, predictions) * 100
print('\nThe Test Accuracy of the DecisionTreeClassifier for depth = %d
 is %f%%' % (optimal_depth, acc))
print('\nThe Test Accuracy of the DecisionTreeClassifier for depth = %d
 is %f%%' % (optimal_learners, acc))
```

```
# evaluate precision
acc = precision_score(Y_test, predictions, pos_label = 1)
print('\nThe Test Precision of the DecisionTreeClassifier for depth = %d is %f' % (optimal_depth, acc))
print('\nThe Test Precision of the DecisionTreeClassifier for depth = %d is %f' % (optimal_learners, acc))

# evaluate recall
acc = recall_score(Y_test, predictions, pos_label = 1)
print('\nThe Test Recall of the DecisionTreeClassifier for depth = %d is %f' % (optimal_depth, acc))
print('\nThe Test Recall of the DecisionTreeClassifier for depth = %d is %f' % (optimal_learners, acc))

# evaluate f1-score
acc = f1_score(Y_test, predictions, pos_label = 1)
print('\nThe Test F1-Score of the DecisionTreeClassifier for depth = %d is %f' % (optimal_depth, acc))
print('\nThe Test F1-Score of the DecisionTreeClassifier for depth = %d is %f' % (optimal_learners, acc))
```

```
The Test Accuracy of the DecisionTreeClassifier for depth = 2 is 80.366667%

The Test Accuracy of the DecisionTreeClassifier for depth = 500 is 80.366667%

The Test Precision of the DecisionTreeClassifier for depth = 2 is 0.816053

The Test Precision of the DecisionTreeClassifier for depth = 500 is 0.816053

The Test Recall of the DecisionTreeClassifier for depth = 2 is 0.979079

The Test Recall of the DecisionTreeClassifier for depth = 500 is 0.979079

The Test F1-Score of the DecisionTreeClassifier for depth = 2 is 0.890163
```

```
The Test F1-Score of the DecisionTreeClassifier for depth = 500 is 0.89
0163
```

In [101]:
```
# Code for drawing seaborn heatmaps on test data
class_names = ['negative','positive']
df_heatmap = pd.DataFrame(confusion_matrix(Y_test, predictions), index=
class_names, columns=class_names )
fig = plt.figure(figsize=(10,7))
heatmap = sns.heatmap(df_heatmap, annot=True, fmt="d")

# Setting tick labels for heatmap
heatmap.yaxis.set_ticklabels(heatmap.yaxis.get_ticklabels(), rotation=0
, ha='right', fontsize=14)
heatmap.xaxis.set_ticklabels(heatmap.xaxis.get_ticklabels(), rotation=0
, ha='right', fontsize=14)
plt.ylabel('Predicted label',size=18)
plt.xlabel('True label',size=18)
plt.title("Confusion Matrix\n",size=24)
plt.show()
```



In [102]:
```
# evaluate accuracy on train data
acc = accuracy_score(Y_train, predictions1) * 100
print('\nThe Train Accuracy of the DecisionTreeClassifier for depth = %
d is %f%%' % (optimal_depth, acc))
print('\nThe Train Accuracy of the DecisionTreeClassifier for depth = %
d is %f%%' % (optimal_learners, acc))

# evaluate precision
acc = precision_score(Y_train, predictions1, pos_label = 1)
print('\nThe Train Precision of the DecisionTreeClassifier for depth =
```

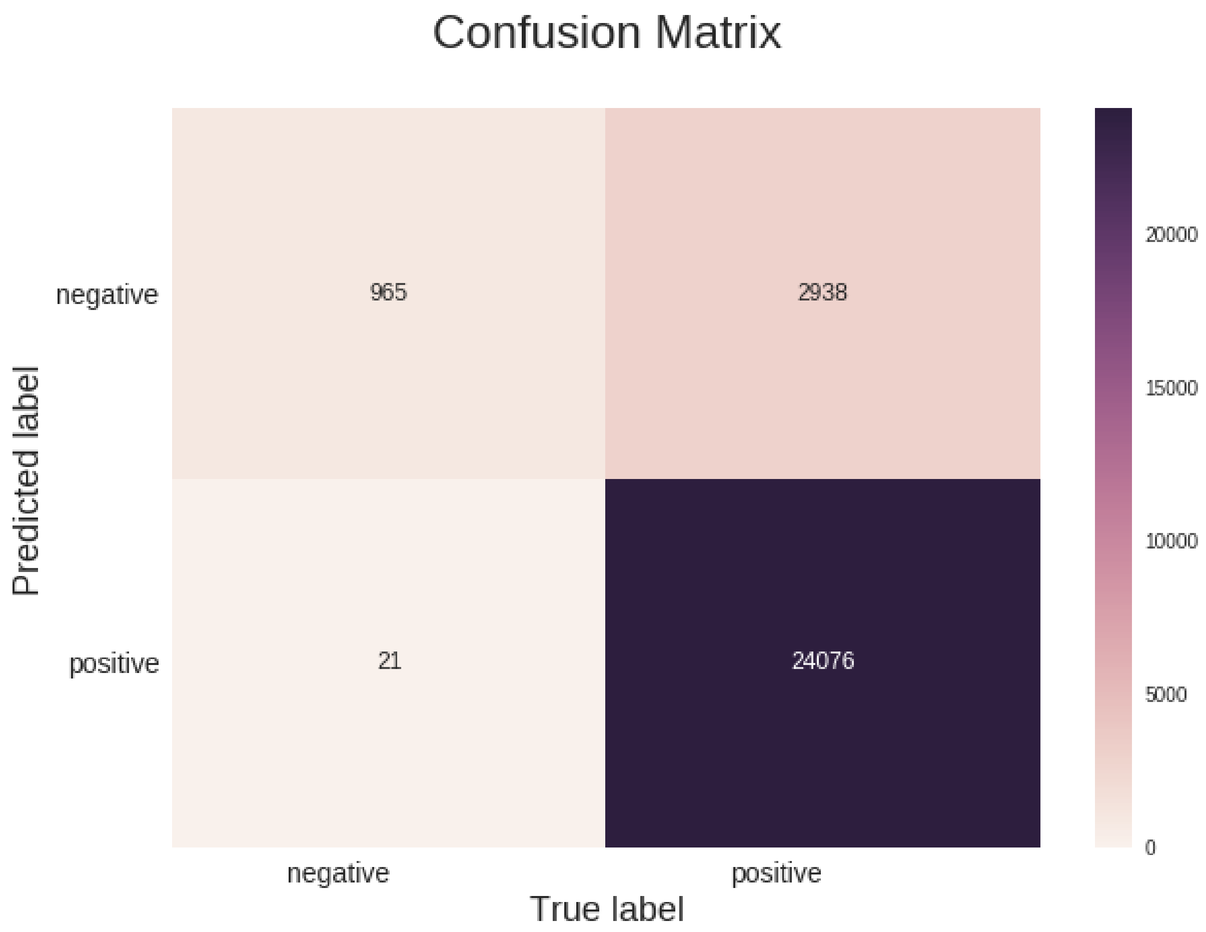
```
%d is %f' % (optimal_depth, acc))
print('\nThe Train Precision of the DecisionTreeClassifier for depth =
%d is %f' % (optimal_learners, acc))

# evaluate recall
acc = recall_score(Y_train, predictions1, pos_label = 1)
print('\nThe Train Recall of the DecisionTreeClassifier for depth = %d
 is %f' % (optimal_depth, acc))
print('\nThe Train Recall of the DecisionTreeClassifier for depth = %d
 is %f' % (optimal_learners, acc))


# evaluate f1-score
acc = f1_score(Y_train, predictions1, pos_label = 1)
print('\nThe Train F1-Score of the DecisionTreeClassifier for depth = %
d is %f' % (optimal_depth, acc))
print('\nThe Train F1-Score of the DecisionTreeClassifier for depth = %
d is %f' % (optimal_learners, acc))
```

```
The Train Accuracy of the DecisionTreeClassifier for depth = 2 is 89.43
2143%

The Train Accuracy of the DecisionTreeClassifier for depth = 500 is 89.
432143%

The Train Precision of the DecisionTreeClassifier for depth = 2 is 0.89
1242

The Train Precision of the DecisionTreeClassifier for depth = 500 is 0.
891242

The Train Recall of the DecisionTreeClassifier for depth = 2 is 0.99912
9

The Train Recall of the DecisionTreeClassifier for depth = 500 is 0.999
129

The Train F1-Score of the DecisionTreeClassifier for depth = 2 is 0.942
106
```

```
The Train F1-Score of the DecisionTreeClassifier for depth = 500 is 0.9
42106
```

In [103]:

```
# Code for drawing seaborn heatmaps
class_names = ['negative','positive']
df_heatmap = pd.DataFrame(confusion_matrix(Y_train, predictions1), inde
x=class_names, columns=class_names )
fig = plt.figure(figsize=(10,7))
heatmap = sns.heatmap(df_heatmap, annot=True, fmt="d")

# Setting tick labels for heatmap
heatmap.yaxis.set_ticklabels(heatmap.yaxis.get_ticklabels(), rotation=0
, ha='right', fontsize=14)
heatmap.xaxis.set_ticklabels(heatmap.xaxis.get_ticklabels(), rotation=0
, ha='right', fontsize=14)
plt.ylabel('Predicted label',size=18)
plt.xlabel('True label',size=18)
plt.title("Confusion Matrix\n",size=24)
plt.show()
```



[5.2.4] Applying XGBOOST on TFIDF W2V, SET 4

In [0]:
```
# Please write all the code with proper documentation
# TF-IDF weighted Word2Vec
tf_idf_vect = TfidfVectorizer()

# final_tf_idf1 is the sparse matrix with row= sentence, col=word and c
ell_val = tfidf
final_tf_idf1 = tf_idf_vect.fit_transform(X_train)
```

```
# tfidf words/col-names
tfidf_feat = tf_idf_vect.get_feature_names()

# compute TFIDF Weighted Word2Vec for each review for X_test .
tfidf_test_vectors = [];
row=0;
for sent in sent_of_test:
    sent_vec = np.zeros(50)
    weight_sum =0;
    for word in sent:
        if word in w2v_words:
            vec = w2v_model.wv[word]
            # obtain the tf_idfidf of a word in a sentence/review
            tf_idf = final_tf_idf1[row, tfidf_feat.index(word)]
            sent_vec += (vec * tf_idf)
            weight_sum += tf_idf
    if weight_sum != 0:
        sent_vec /= weight_sum
    tfidf_test_vectors.append(sent_vec)
    row += 1
```

In [0]:

```
# compute TFIDF Weighted Word2Vec for each review for X_train .
tfidf_train_vectors = [];
row=0;
for sent in sent_of_train:
    sent_vec = np.zeros(50)
    weight_sum =0;
    for word in sent:
        if word in w2v_words:
            vec = w2v_model.wv[word]
            # obtain the tf_idfidf of a word in a sentence/review
            tf_idf = final_tf_idf1[row, tfidf_feat.index(word)]
            sent_vec += (vec * tf_idf)
            weight_sum += tf_idf
    if weight_sum != 0:
        sent_vec /= weight_sum
    tfidf_train_vectors.append(sent_vec)
    row += 1
```

```
X_train_vec = tfidf_train_vectors
X_test_vec = tfidf_test_vectors
X_train_vec_standardized = sc.fit_transform(X_train_vec)
X_test_vec_standardized = sc.transform(X_test_vec)
```

In [112]:

```
# Please write all the code with proper documentation
# Please write all the code with proper documentation
from sklearn.ensemble import GradientBoostingClassifier

base_learners = [5, 10, 50, 100, 200, 500, 1000]
depth = [2, 3, 4, 5, 6, 7, 8, 9, 10]

param_grid = {'n_estimators': base_learners,  'max_depth': depth}
GBC = GradientBoostingClassifier(max_features='sqrt',subsample=0.1)
model = GridSearchCV(GBC, param_grid, scoring = 'roc_auc', cv=3, n_jobs
 = -1,pre_dispatch=2)
model.fit(X_train_vec_standardized, Y_train)
print("Model with best parameters :\n",model.best_estimator_)
print("Accuracy of the model : ",model.score(X_test_vec_standardized, Y
_test))
```

```
Model with best parameters :
 GradientBoostingClassifier(criterion='friedman_mse', init=None,
              learning_rate=0.1, loss='deviance', max_depth=2,
              max_features='sqrt', 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=500,
              n_iter_no_change=None, presort='auto', random_state=None,
              subsample=0.1, tol=0.0001, validation_fraction=0.1,
              verbose=0, warm_start=False)
Accuracy of the model :  0.591282015106339
```

In [0]:

```
gb = GradientBoostingClassifier(n_estimators=optimal_learners, max_dept
h=optimal_depth, max_features='sqrt', subsample=0.1)
gb.fit(X_train_vec_standardized,Y_train)
prediction = gb.predict(X_test_vec_standardized)
prediction1 = gb.predict(X_train_vec_standardized)
```

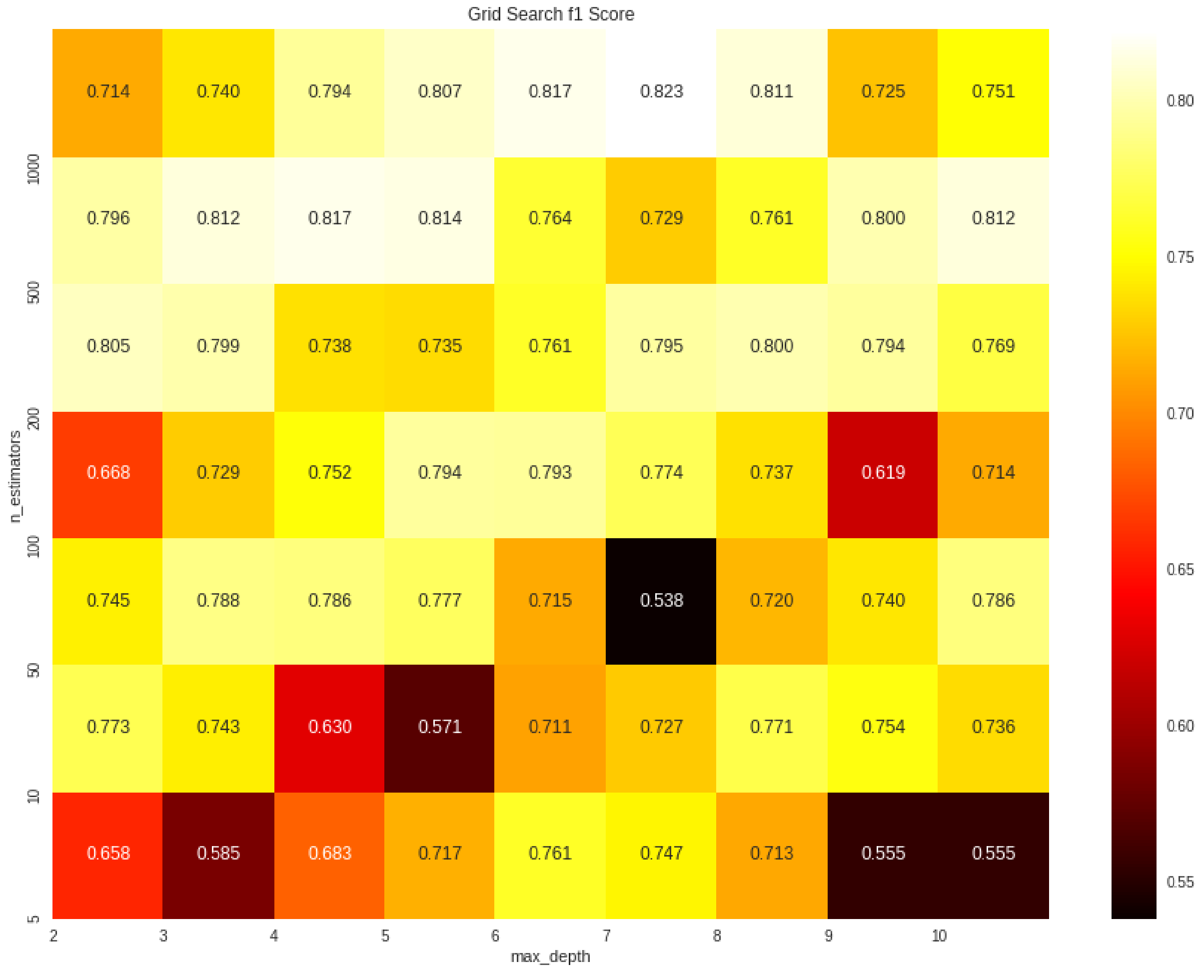
In [114]:

```
# Cross-Validation errors
cv_errors = [1-i for i in model.cv_results_['mean_test_score']]
training_scores=[1-i for i in model.cv_results_['mean_train_score']]

# Optimal value of number of base learners
optimal_learners = model.best_estimator_.n_estimators
print("The optimal number of base learners is : ",optimal_learners)


optimal_depth=model.best_estimator_.max_depth
print("The optimal number of depth is : ",optimal_depth)

# Variables that will be used for  making table in Conclusion part of t
his assignment
tfidf_avg_w2v_gbdt_learners = optimal_learners
tfidf_avg_w2v_gbdt_depth = optimal_depth
tfidf_avg_w2v_gbdt_train_acc = model.score(X_test_vec, Y_test)*100
tfidf_avg_w2v_gbdt_test_acc = accuracy_score(Y_test, predictions) * 100
```

```
The optimal number of base learners is :  500
The optimal number of depth is :  2
```

In [115]:

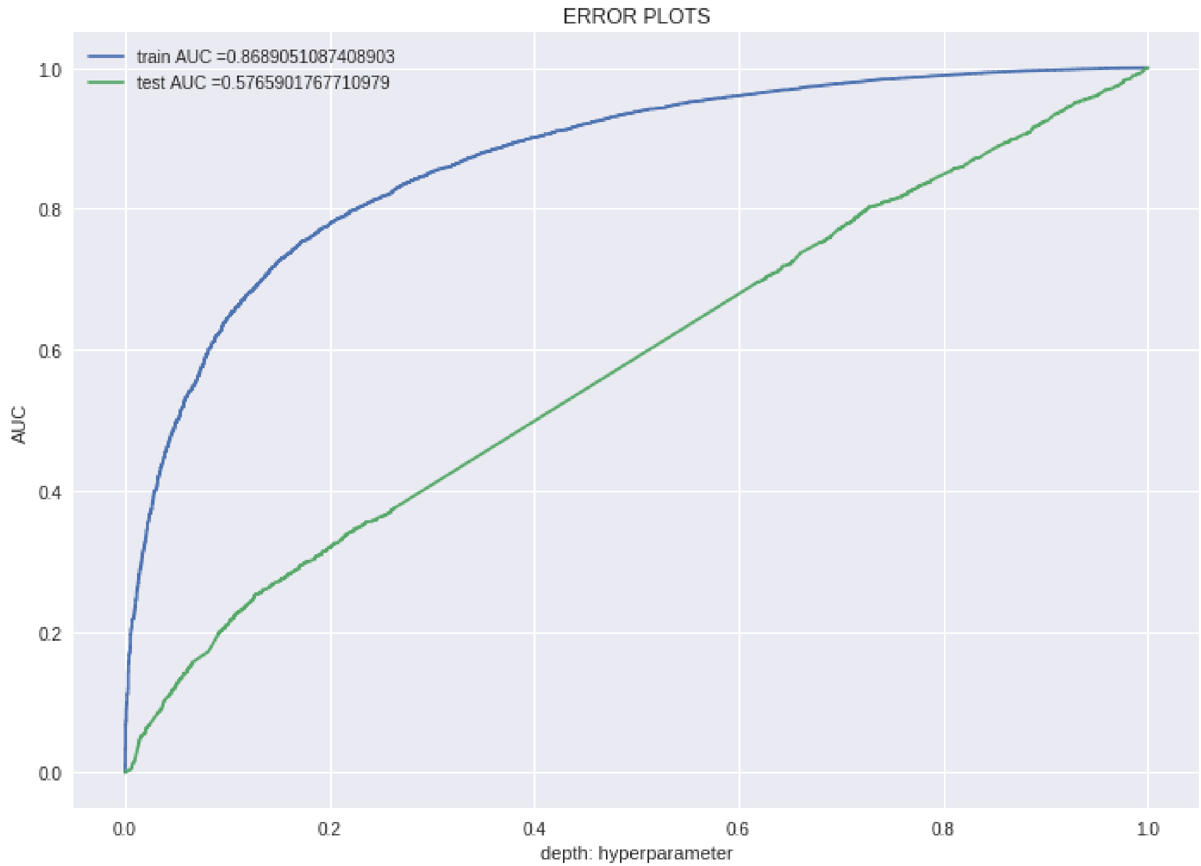
```
scores = model.cv_results_['mean_test_score'].reshape(len(base_learners
),len(depth))

plt.figure(figsize=(15, 11))
sns.heatmap(scores, annot=True, cmap=plt.cm.hot, fmt=".3f", xticklabels
=base_learners, yticklabels=depth)
plt.ylabel('n_estimators')
plt.xlabel('max_depth')
plt.yticks(np.arange(len(base_learners)), base_learners)
plt.xticks(np.arange(len(depth)), depth)
plt.title('Grid Search f1 Score')
plt.show()
```



In [116]:

```
train_fpr, train_tpr, thresholds = roc_curve(Y_train, gb.predict_proba(
X_train_vec_standardized)[:,1])
test_fpr, test_tpr, thresholds = roc_curve(Y_test, gb.predict_proba(X_t
est_vec_standardized)[:,1])

plt.plot(train_fpr, train_tpr, label="train AUC ="+str(auc(train_fpr, t
rain_tpr)))
plt.plot(test_fpr, test_tpr, label="test AUC ="+str(auc(test_fpr, test_
tpr)))
```

```
plt.legend()
plt.xlabel("depth: hyperparameter")
plt.ylabel("AUC")
plt.title("ERROR PLOTS")
plt.show()
```



In [117]:
```
# evaluate accuracy on test data
acc = accuracy_score(Y_test, predictions) * 100
print('\nThe Test Accuracy of the DecisionTreeClassifier for depth = %d
 is %f%%' % (optimal_depth, acc))
print('\nThe Test Accuracy of the DecisionTreeClassifier for depth = %d
 is %f%%' % (optimal_learners, acc))
```

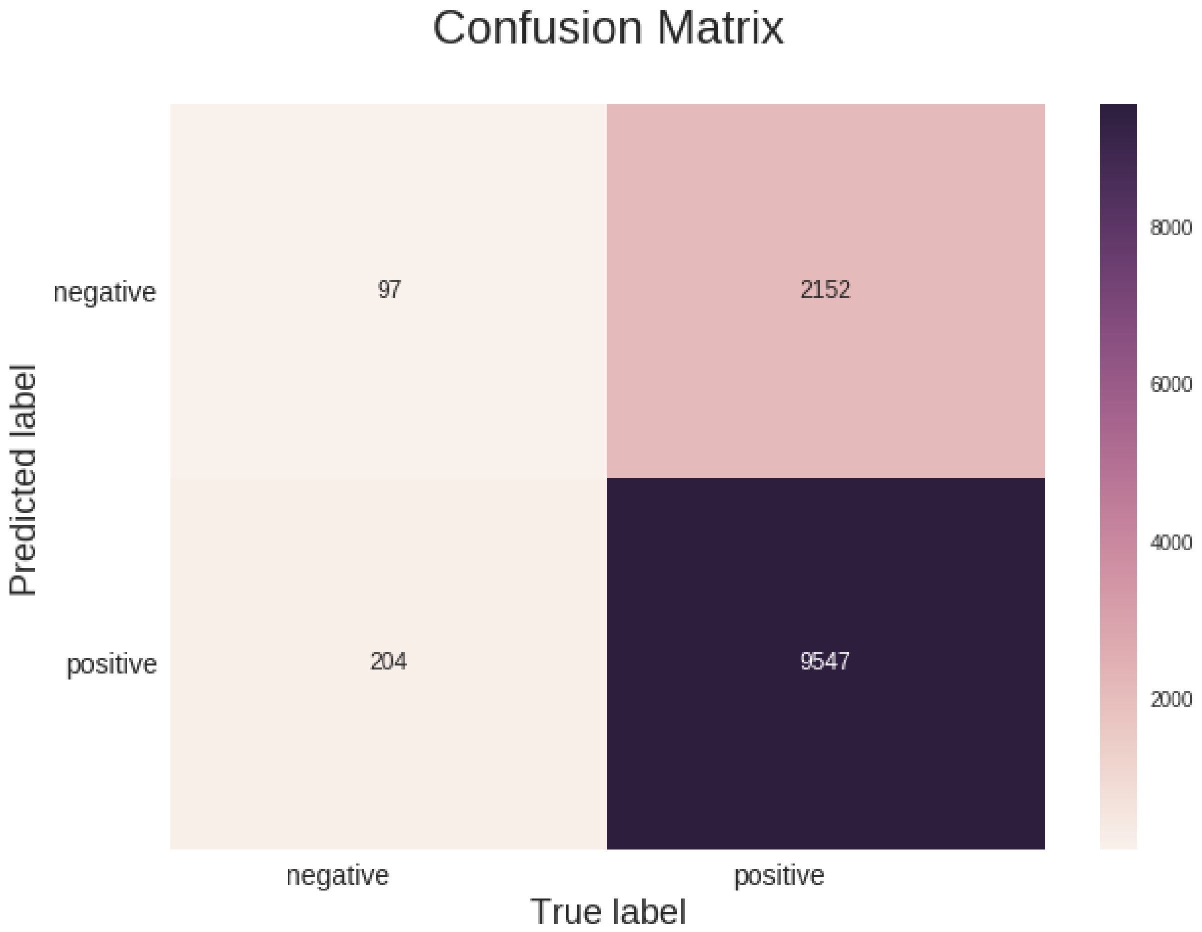
```
# evaluate precision
acc = precision_score(Y_test, predictions, pos_label = 1)
print('\nThe Test Precision of the DecisionTreeClassifier for depth = %
d is %f' % (optimal_depth, acc))
print('\nThe Test Precision of the DecisionTreeClassifier for depth = %
d is %f' % (optimal_learners, acc))

# evaluate recall
acc = recall_score(Y_test, predictions, pos_label = 1)
print('\nThe Test Recall of the DecisionTreeClassifier for depth = %d i
s %f' % (optimal_depth, acc))
print('\nThe Test Recall of the DecisionTreeClassifier for depth = %d i
s %f' % (optimal_learners, acc))

# evaluate f1-score
acc = f1_score(Y_test, predictions, pos_label = 1)
print('\nThe Test F1-Score of the DecisionTreeClassifier for depth = %d
 is %f' % (optimal_depth, acc))
print('\nThe Test F1-Score of the DecisionTreeClassifier for depth = %d
 is %f' % (optimal_learners, acc))
```

```
The Test Accuracy of the DecisionTreeClassifier for depth = 2 is 80.366
667%

The Test Accuracy of the DecisionTreeClassifier for depth = 500 is 80.3
66667%

The Test Precision of the DecisionTreeClassifier for depth = 2 is 0.816
053

The Test Precision of the DecisionTreeClassifier for depth = 500 is 0.8
16053

The Test Recall of the DecisionTreeClassifier for depth = 2 is 0.979079

The Test Recall of the DecisionTreeClassifier for depth = 500 is 0.9790
79

The Test F1-Score of the DecisionTreeClassifier for depth = 2 is 0.8901
63
```

```
The Test F1-Score of the DecisionTreeClassifier for depth = 500 is 0.89
0163
```

In [118]:

```
# Code for drawing seaborn heatmaps on test data
class_names = ['negative','positive']
df_heatmap = pd.DataFrame(confusion_matrix(Y_test, predictions), index=
class_names, columns=class_names )
fig = plt.figure(figsize=(10,7))
heatmap = sns.heatmap(df_heatmap, annot=True, fmt="d")

# Setting tick labels for heatmap
heatmap.yaxis.set_ticklabels(heatmap.yaxis.get_ticklabels(), rotation=0
, ha='right', fontsize=14)
heatmap.xaxis.set_ticklabels(heatmap.xaxis.get_ticklabels(), rotation=0
, ha='right', fontsize=14)
plt.ylabel('Predicted label',size=18)
plt.xlabel('True label',size=18)
plt.title("Confusion Matrix\n",size=24)
plt.show()
```



In [119]:

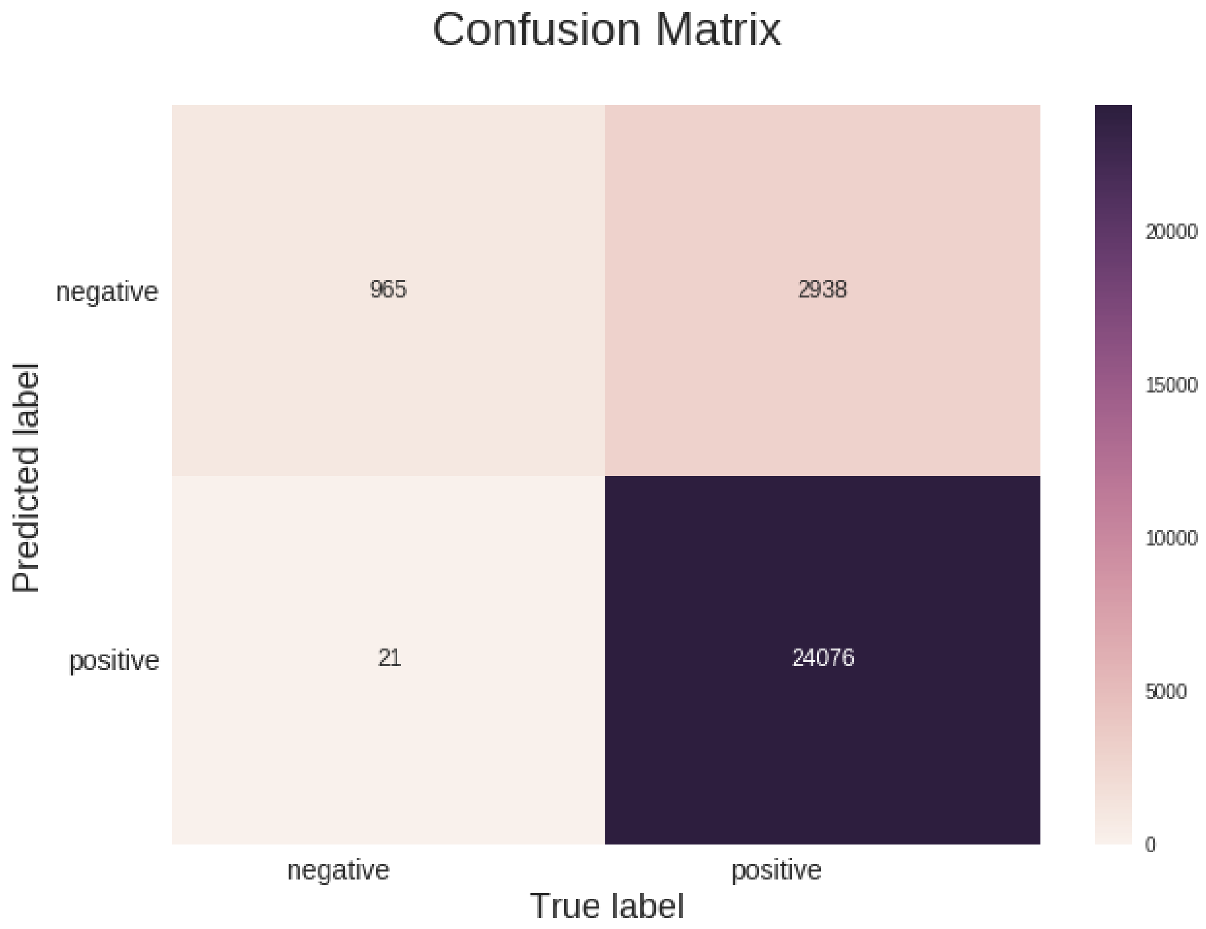
```
# evaluate accuracy on train data
acc = accuracy_score(Y_train, predictions1) * 100
print('\nThe Train Accuracy of the DecisionTreeClassifier for depth = %
d is %f%%' % (optimal_depth, acc))
print('\nThe Train Accuracy of the DecisionTreeClassifier for depth = %
d is %f%%' % (optimal_learners, acc))

# evaluate precision
acc = precision_score(Y_train, predictions1, pos_label = 1)
print('\nThe Train Precision of the DecisionTreeClassifier for depth =
```

```
%d is %f' % (optimal_depth, acc))
print('\nThe Train Precision of the DecisionTreeClassifier for depth =
%d is %f' % (optimal_learners, acc))

# evaluate recall
acc = recall_score(Y_train, predictions1, pos_label = 1)
print('\nThe Train Recall of the DecisionTreeClassifier for depth = %d
 is %f' % (optimal_depth, acc))
print('\nThe Train Recall of the DecisionTreeClassifier for depth = %d
 is %f' % (optimal_learners, acc))


# evaluate f1-score
acc = f1_score(Y_train, predictions1, pos_label = 1)
print('\nThe Train F1-Score of the DecisionTreeClassifier for depth = %
d is %f' % (optimal_depth, acc))
print('\nThe Train F1-Score of the DecisionTreeClassifier for depth = %
d is %f' % (optimal_learners, acc))
```

```
The Train Accuracy of the DecisionTreeClassifier for depth = 2 is 89.43
2143%

The Train Accuracy of the DecisionTreeClassifier for depth = 500 is 89.
432143%

The Train Precision of the DecisionTreeClassifier for depth = 2 is 0.89
1242

The Train Precision of the DecisionTreeClassifier for depth = 500 is 0.
891242

The Train Recall of the DecisionTreeClassifier for depth = 2 is 0.99912
9

The Train Recall of the DecisionTreeClassifier for depth = 500 is 0.999
129

The Train F1-Score of the DecisionTreeClassifier for depth = 2 is 0.942
106
```

```
The Train F1-Score of the DecisionTreeClassifier for depth = 500 is 0.9
42106
```

In [120]:
```
# Code for drawing seaborn heatmaps
class_names = ['negative','positive']
df_heatmap = pd.DataFrame(confusion_matrix(Y_train, predictions1), inde
x=class_names, columns=class_names )
fig = plt.figure(figsize=(10,7))
heatmap = sns.heatmap(df_heatmap, annot=True, fmt="d")

# Setting tick labels for heatmap
heatmap.yaxis.set_ticklabels(heatmap.yaxis.get_ticklabels(), rotation=0
, ha='right', fontsize=14)
heatmap.xaxis.set_ticklabels(heatmap.xaxis.get_ticklabels(), rotation=0
, ha='right', fontsize=14)
plt.ylabel('Predicted label',size=18)
plt.xlabel('True label',size=18)
plt.title("Confusion Matrix\n",size=24)
plt.show()
```



procedure

- STEP 1 :- Text Preprocessing
- STEP 2:- Time-based splitting of whole dataset into train_data and test_data
- STEP 3:- Training the vectorizer on train_data and later applying same vectorizer on both train_data and test_data to transform them into vectors
- STEP 4:- Using Random Forest/GBDT as an estimator in GridSearchCV in order to find optimal value of base_learners,depth .

- STEP 5:- Once , we get optimal value of base_learners then train Random Forest again with this optimal value of base_learners,depth and make predictions on test_data
- STEP 6:- Draw seabornheatmap
- STEP 7 :- Evaluate : Accuracy , F1-Score , Precision , Recall for test and train
- STEP 8:- Draw Seaborn Heatmap for Confusion Matrix for test and train.

[6] Conclusions

In [124]:

```
# Please compare all your models using Prettytable library
# Creating table using PrettyTable library
from prettytable import PrettyTable

# Table for Random Forest
names_rf = ['Random Forest for BoW','Random Forest for TFIDF','Random F
orest for Avg_Word2Vec','Random Forest for tfidf_Word2Vec']

learners_rf = [bow_rf_learners,tfidf_rf_learners,avg_w2v_rf_learners,tf
idf_avg_w2v_rf_learners]

optimal_depth = [bow_rf_depth,tfidf_rf_depth,avg_w2v_rf_depth,tfidf_avg
_w2v_rf_depth]

train_acc_rf = [bow_rf_train_acc,tfidf_rf_train_acc,avg_w2v_rf_train_ac
c,tfidf_avg_w2v_rf_train_acc]

test_acc_rf = [bow_rf_test_acc,tfidf_rf_test_acc,avg_w2v_rf_test_acc,tf
idf_avg_w2v_rf_test_acc]

numbering_rf = [1,2,3,4]

# Initializing prettytable
ptable = PrettyTable()

# Adding columns
ptable.add_column("S.NO.",numbering_rf)
ptable.add_column("MODEL",names_rf)
```

```
ptable.add_column("Base Learners ",learners_rf)
ptable.add_column("Optimal_Depth",optimal_depth)
ptable.add_column("Training Accuracy",train_acc_rf)
ptable.add_column("Test Accuracy",test_acc_rf)

print('\t\t\t\t\tTABLE FOR RANDOM FOREST')
# Printing the Table
print(ptable)
print("\n\n")




# Table for Gradient Boosting Decision Tree (GBDT)
names = ['GBDT for BoW','GBDT for TFIDF','GBDT for Avg_Word2Vec','GBDT
 for tfidf_Word2Vec']

base_learners = [bow_gbdt_learners,tfidf_gbdt_learners,avg_w2v_gbdt_lea
rners,tfidf_avg_w2v_gbdt_learners]

optimal_depth = [bow_gbdt_depth,tfidf_gbdt_depth,avg_w2v_gbdt_depth,tfi
df_avg_w2v_gbdt_depth]

train_acc = [bow_gbdt_train_acc,tfidf_gbdt_train_acc,avg_w2v_gbdt_train
_acc,tfidf_avg_w2v_gbdt_train_acc]

test_acc = [bow_gbdt_test_acc,tfidf_gbdt_test_acc,avg_w2v_gbdt_test_acc
,tfidf_avg_w2v_gbdt_test_acc]

numbering = [1,2,3,4]

# Initializing prettytable
table = PrettyTable()

# Adding columns
table.add_column("S.NO.",numbering)
table.add_column("MODEL",names)
table.add_column("Base Learners ",base_learners)
table.add_column("Optimal_Depth",optimal_depth)
```

```
table.add_column("Training Accuracy",train_acc)
table.add_column("Test Accuracy",test_acc)

print('\t\t\t\tTABLE FOR GRADIENT BOOSTING DECISION TREE (GBDT)')
# Printing the Table
print(table)
```

```
                                        TABLE FOR RANDOM FOREST
+-------+----------------------------------+----------------+----------
-----+-------------------+------------------+
| S.NO. |              MODEL               | Base Learners  | Optimal_D
epth | Training Accuracy |   Test Accuracy   |
+-------+----------------------------------+----------------+----------
-----+-------------------+------------------+
|   1   |       Random Forest for BoW      |      1000      |       10
     | 81.33993758960044 | 81.25833333333333 |
|   2   |      Random Forest for TFIDF     |      1000      |       10
     | 81.62999916233467 | 81.27499999999999 |
|   3   |   Random Forest for Avg_Word2Vec  |      1000      |       10
     | 85.57855383395139 |       82.825      |
|   4   | Random Forest for tfidf_Word2Vec |      500       |       10
     |  59.1744714625842 | 80.36666666666666 |
+-------+----------------------------------+----------------+----------
-----+-------------------+------------------+



                                TABLE FOR GRADIENT BOOSTING DECISION TR
EE (GBDT)
+-------+-------------------------+----------------+----------------+---
----------------+-------------------+
| S.NO. |          MODEL          | Base Learners  | Optimal_Depth | Tr
aining Accuracy  |   Test Accuracy   |
+-------+-------------------------+----------------+----------------+---
----------------+-------------------+
|   1   |       GBDT for BoW      |      500       |       2        | 7
5.31048679026388  | 80.36666666666666 |
|   2   |      GBDT for TFIDF     |      500       |       2        |  7
1.2819435149085  | 80.36666666666666 |
|   3   |   GBDT for Avg_Word2Vec  |      500       |       2        | 8
```

```
2.97019986184222  | 80.36666666666666 |
|   4   | GBDT for tfidf_Word2Vec |      500       |       2       | 6
0.052182400920316 | 80.36666666666666 |
+-------+-------------------------+---------------+---------------+---
-----------------+-------------------+
```

applying feature engineering only to gbdt(bow,tfidf)due to computation issues similarly we can implement for other two

In [105]:

```
conn = sqlite3.connect('featureeng.sqlite')
final1 = pd.read_sql_query(""" SELECT * FROM Reviews WHERE Score != 3
 """, conn)
final1.head()
```

Out[105]:

	level_0	index	Id	ProductId	UserId	ProfileName	HelpfulnessNu
0	0	138706	150524	0006641040	ACITT7DI6IDDL	shari zychinski	0
1	1	138688	150506	0006641040	A2IW4PEEKO2R0U	Tracy	1

	level_0	index	Id	ProductId	UserId	ProfileName	HelpfulnessNur
2	2	138689	150507	0006641040	A1S4A3IQ2MU7V4	sally sue "sally sue"	1
3	3	138690	150508	0006641040	AZGXZ2UUK6X	Catherine Hallberg " (Kate)"	1
4	4	138691	150509	0006641040	A3CMRKGE0P909G	Teresa	3

In [107]:
```
print(final1['CleanedText'][0])
```

```
witti littl book make son laugh loud recit car drive along alway sing r
efrain hes learn whale india droop love new word book introduc silli cl
assic book will bet son still abl recit memori colleg witti littl book
make son laugh loud recit car drive along alway sing refrain hes learn
whale india droop love new word book introduc silli classic book will b
et son still abl recit memori colleg everi book educ everi book educ
```

In [131]:
```
final1.shape
```

Out[131]:
```
(20000, 14)
```

In [0]:
```
final1=final1[:20000]
```

In [0]:
```
from sklearn.model_selection import train_test_split
##Sorting data according to Time in ascending order for Time Based Spli
tting
time_sorted_data1 = final1.sort_values('Time', axis=0, ascending=True,
inplace=False, kind='quicksort', na_position='last')

x = time_sorted_data1['CleanedText'].values
y = time_sorted_data1['Score']

# split the data set into train and test
X_train, X_test, Y_train, Y_test = train_test_split(x, y, test_size=0.3
, random_state=0,shuffle=False)
```

gbdt bow

In [133]:
```
# Please write all the code with proper documentation
#BoW
count_vect = CountVectorizer(min_df = 1000)
X_train_vec = count_vect.fit_transform(X_train)
X_test_vec = count_vect.transform(X_test)
print("the type of count vectorizer :",type(X_train_vec))
print("the shape of out text BOW vectorizer : ",X_train_vec.get_shape
())
print("the number of unique words :", X_train_vec.get_shape()[1])
```

```
the type of count vectorizer : <class 'scipy.sparse.csr.csr_matrix'>
the shape of out text BOW vectorizer :  (14000, 65)
the number of unique words : 65
```

In [0]:
```
from sklearn.preprocessing import StandardScaler
sc = StandardScaler(with_mean=False)
```

```
X_train_vec_standardized = sc.fit_transform(X_train_vec)
X_test_vec_standardized = sc.transform(X_test_vec)
```

In [135]:

```
# Please write all the code with proper documentation
from sklearn.ensemble import GradientBoostingClassifier

base_learners = [5, 10, 50, 100, 200, 500, 1000]
depth = [2, 3, 4, 5, 6, 7, 8, 9, 10]

param_grid = {'n_estimators': base_learners,  'max_depth': depth}
GBC = GradientBoostingClassifier(max_features='sqrt',subsample=0.1)
model = GridSearchCV(GBC, param_grid, scoring = 'roc_auc', cv=3, n_jobs
 = -1,pre_dispatch=2)
model.fit(X_train_vec_standardized, Y_train)
print("Model with best parameters :\n",model.best_estimator_)
print("Accuracy of the model : ",model.score(X_test_vec_standardized, Y
_test))
```

```
Model with best parameters :
 GradientBoostingClassifier(criterion='friedman_mse', init=None,
              learning_rate=0.1, loss='deviance', max_depth=3,
              max_features='sqrt', 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=100,
              n_iter_no_change=None, presort='auto', random_state=None,
              subsample=0.1, tol=0.0001, validation_fraction=0.1,
              verbose=0, warm_start=False)
Accuracy of the model :  0.8169878479376892
```

In [0]:

```
gb = GradientBoostingClassifier(n_estimators=optimal_learners, max_dept
h=optimal_depth, max_features='sqrt', subsample=0.1)
gb.fit(X_train_vec_standardized,Y_train)
prediction = gb.predict(X_test_vec_standardized)
prediction1 = gb.predict(X_train_vec_standardized)
```

In [142]:

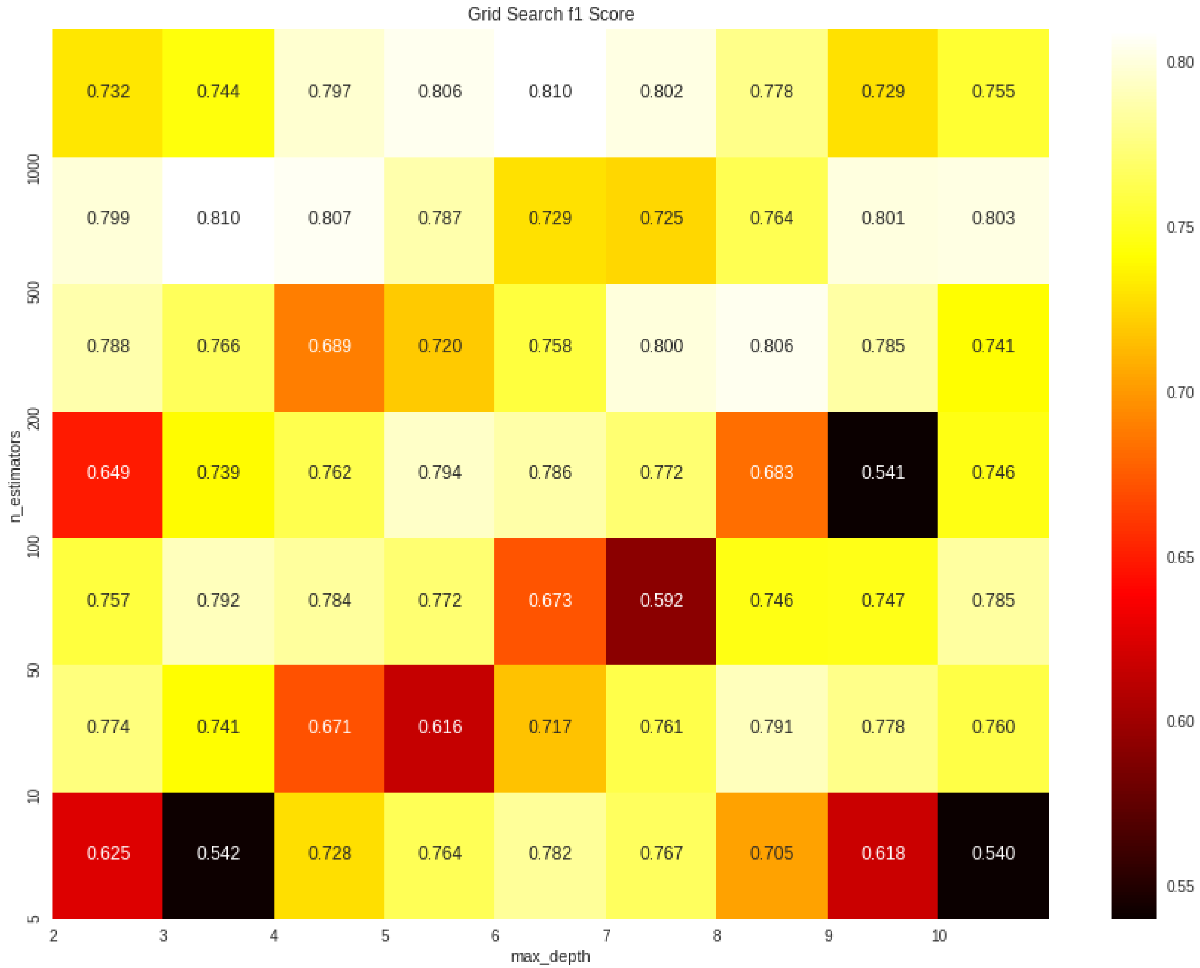
```
# Cross-Validation errors
cv_errors = [1-i for i in model.cv_results_['mean_test_score']]
```

```
training_scores=[1-i for i in model.cv_results_['mean_train_score']]

# Optimal value of number of base learners
optimal_learners = model.best_estimator_.n_estimators
print("The optimal number of base learners is : ",optimal_learners)


optimal_depth=model.best_estimator_.max_depth
print("The optimal number of depth is : ",optimal_depth)

# Variables that will be used for  making table in Conclusion part of t
his assignment
tfidf_gbdt_learners = optimal_learners
tfidf_gbdt_depth = optimal_depth
tfidf_gbdt_train_acc = model.score(X_test_vec_standardized, Y_test)*100
tfidf_gbdt_test_acc = accuracy_score(Y_test, prediction) * 100
```

```
The optimal number of base learners is :  100
The optimal number of depth is :  3
```

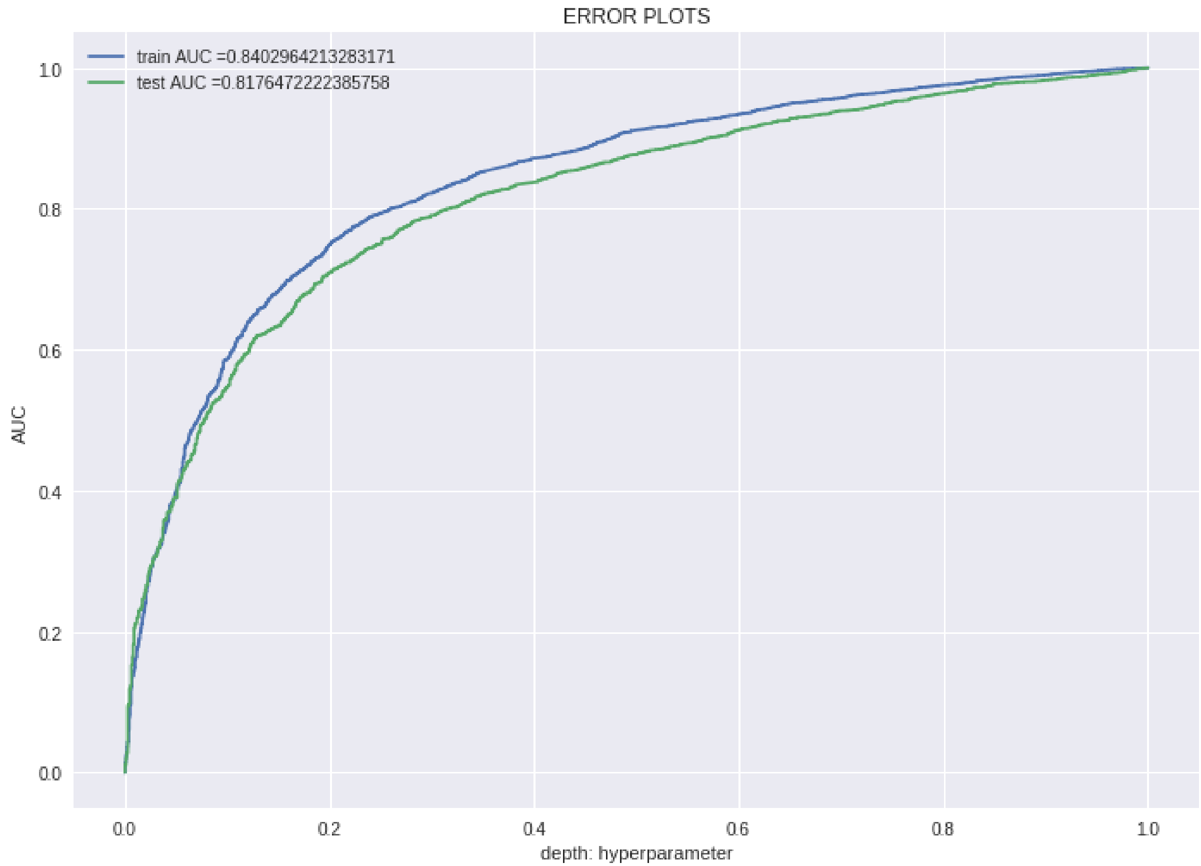
In [143]:

```
scores = model.cv_results_['mean_test_score'].reshape(len(base_learners
),len(depth))

plt.figure(figsize=(15, 11))
sns.heatmap(scores, annot=True, cmap=plt.cm.hot, fmt=".3f", xticklabels
=base_learners, yticklabels=depth)
plt.ylabel('n_estimators')
plt.xlabel('max_depth')
plt.yticks(np.arange(len(base_learners)), base_learners)
plt.xticks(np.arange(len(depth)), depth)
plt.title('Grid Search f1 Score')
plt.show()
```



In [144]:
```
train_fpr, train_tpr, thresholds = roc_curve(Y_train, gb.predict_proba(
X_train_vec_standardized)[:,1])
test_fpr, test_tpr, thresholds = roc_curve(Y_test, gb.predict_proba(X_t
est_vec_standardized)[:,1])

plt.plot(train_fpr, train_tpr, label="train AUC ="+str(auc(train_fpr, t
rain_tpr)))
plt.plot(test_fpr, test_tpr, label="test AUC ="+str(auc(test_fpr, test_
tpr)))
```

```
plt.legend()
plt.xlabel("depth: hyperparameter")
plt.ylabel("AUC")
plt.title("ERROR PLOTS")
plt.show()
```



In [147]:
```
# evaluate accuracy on test data
acc = accuracy_score(Y_test, prediction) * 100
print('\nThe Test Accuracy of the DecisionTreeClassifier for depth = %d
 is %f%%' % (optimal_depth, acc))
print('\nThe Test Accuracy of the DecisionTreeClassifier for depth = %d
 is %f%%' % (optimal_learners, acc))
```

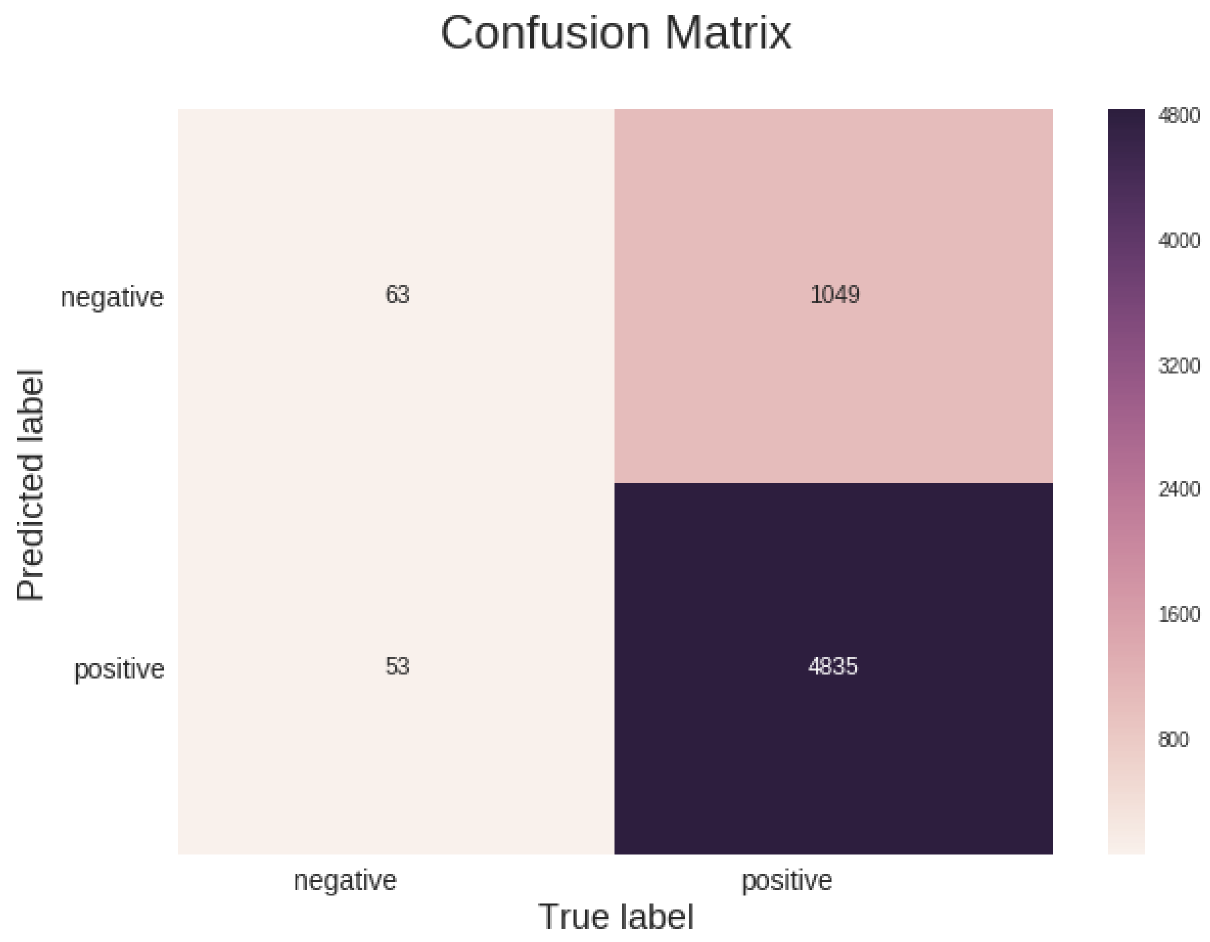
```
# evaluate precision
acc = precision_score(Y_test, prediction, pos_label = 1)
print('\nThe Test Precision of the DecisionTreeClassifier for depth = %
d is %f' % (optimal_depth, acc))
print('\nThe Test Precision of the DecisionTreeClassifier for depth = %
d is %f' % (optimal_learners, acc))

# evaluate recall
acc = recall_score(Y_test, prediction, pos_label = 1)
print('\nThe Test Recall of the DecisionTreeClassifier for depth = %d i
s %f' % (optimal_depth, acc))
print('\nThe Test Recall of the DecisionTreeClassifier for depth = %d i
s %f' % (optimal_learners, acc))

# evaluate f1-score
acc = f1_score(Y_test, prediction, pos_label = 1)
print('\nThe Test F1-Score of the DecisionTreeClassifier for depth = %d
 is %f' % (optimal_depth, acc))
print('\nThe Test F1-Score of the DecisionTreeClassifier for depth = %d
 is %f' % (optimal_learners, acc))
```

```
The Test Accuracy of the DecisionTreeClassifier for depth = 3 is 81.633
333%

The Test Accuracy of the DecisionTreeClassifier for depth = 100 is 81.6
33333%

The Test Precision of the DecisionTreeClassifier for depth = 3 is 0.821
720

The Test Precision of the DecisionTreeClassifier for depth = 100 is 0.8
21720

The Test Recall of the DecisionTreeClassifier for depth = 3 is 0.989157

The Test Recall of the DecisionTreeClassifier for depth = 100 is 0.9891
57

The Test F1-Score of the DecisionTreeClassifier for depth = 3 is 0.8976
98
```

```
The Test F1-Score of the DecisionTreeClassifier for depth = 100 is 0.89
7698
```

In [151]:

```
# Code for drawing seaborn heatmaps on test data
class_names = ['negative','positive']
df_heatmap = pd.DataFrame(confusion_matrix(Y_test, prediction), index=c
lass_names, columns=class_names )
fig = plt.figure(figsize=(10,7))
heatmap = sns.heatmap(df_heatmap, annot=True, fmt="d")

# Setting tick labels for heatmap
heatmap.yaxis.set_ticklabels(heatmap.yaxis.get_ticklabels(), rotation=0
, ha='right', fontsize=14)
heatmap.xaxis.set_ticklabels(heatmap.xaxis.get_ticklabels(), rotation=0
, ha='right', fontsize=14)
plt.ylabel('Predicted label',size=18)
plt.xlabel('True label',size=18)
plt.title("Confusion Matrix\n",size=24)
plt.show()
```



In [148]:
```
# evaluate accuracy on train data
acc = accuracy_score(Y_train, prediction1) * 100
print('\nThe Train Accuracy of the DecisionTreeClassifier for depth = %
d is %f%%' % (optimal_depth, acc))
print('\nThe Train Accuracy of the DecisionTreeClassifier for depth = %
d is %f%%' % (optimal_learners, acc))

# evaluate precision
acc = precision_score(Y_train, prediction1, pos_label = 1)
print('\nThe Train Precision of the DecisionTreeClassifier for depth =
```

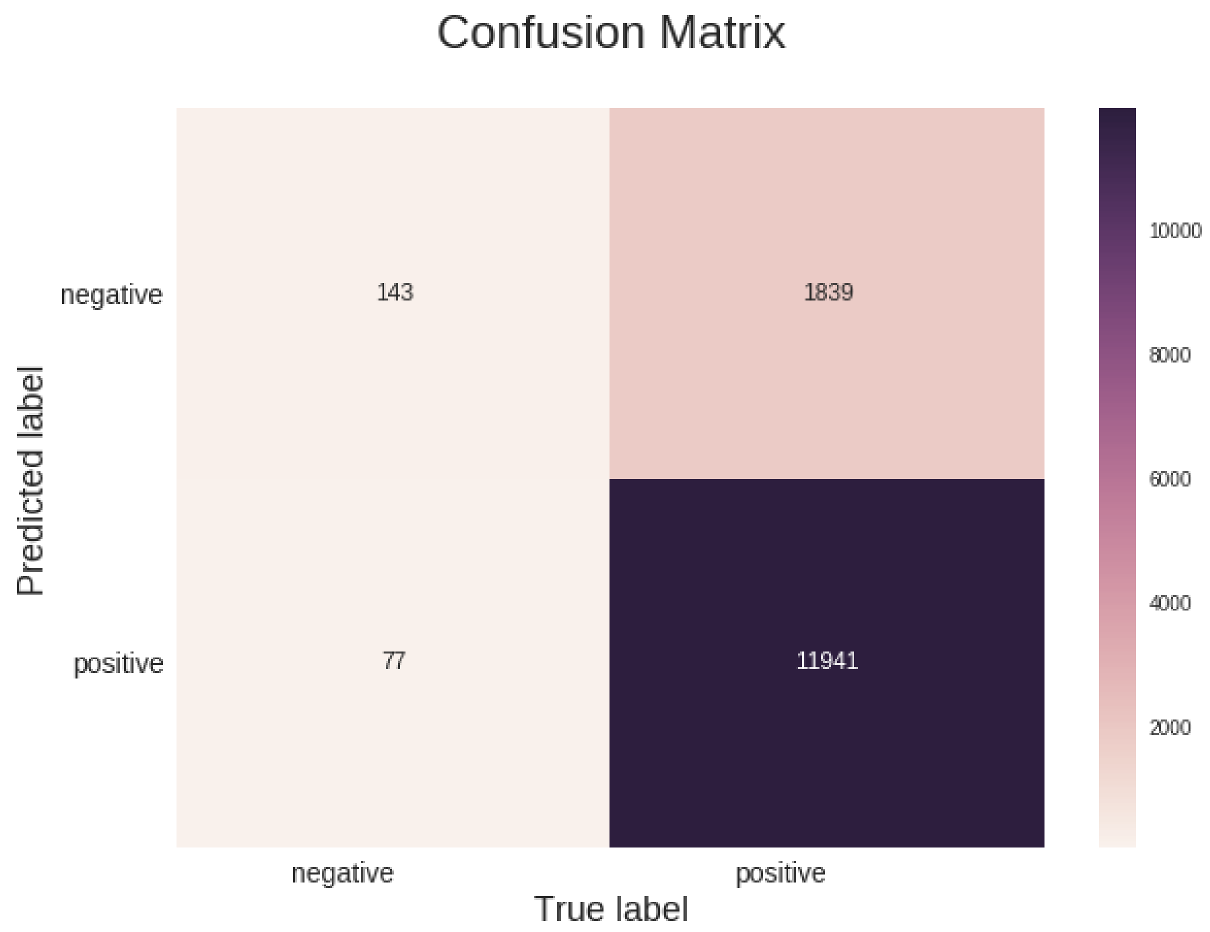
```
%d is %f' % (optimal_depth, acc))
print('\nThe Train Precision of the DecisionTreeClassifier for depth =
%d is %f' % (optimal_learners, acc))

# evaluate recall
acc = recall_score(Y_train, prediction1, pos_label = 1)
print('\nThe Train Recall of the DecisionTreeClassifier for depth = %d
 is %f' % (optimal_depth, acc))
print('\nThe Train Recall of the DecisionTreeClassifier for depth = %d
 is %f' % (optimal_learners, acc))


# evaluate f1-score
acc = f1_score(Y_train, prediction1, pos_label = 1)
print('\nThe Train F1-Score of the DecisionTreeClassifier for depth = %
d is %f' % (optimal_depth, acc))
print('\nThe Train F1-Score of the DecisionTreeClassifier for depth = %
d is %f' % (optimal_learners, acc))
```

```
The Train Accuracy of the DecisionTreeClassifier for depth = 3 is 86.31
4286%

The Train Accuracy of the DecisionTreeClassifier for depth = 100 is 86.
314286%

The Train Precision of the DecisionTreeClassifier for depth = 3 is 0.86
6546

The Train Precision of the DecisionTreeClassifier for depth = 100 is 0.
866546

The Train Recall of the DecisionTreeClassifier for depth = 3 is 0.99359
3

The Train Recall of the DecisionTreeClassifier for depth = 100 is 0.993
593

The Train F1-Score of the DecisionTreeClassifier for depth = 3 is 0.925
731
```

```
The Train F1-Score of the DecisionTreeClassifier for depth = 100 is 0.9
25731
```

In [150]:

```
# Code for drawing seaborn heatmaps
class_names = ['negative','positive']
df_heatmap = pd.DataFrame(confusion_matrix(Y_train, prediction1), index
=class_names, columns=class_names )
fig = plt.figure(figsize=(10,7))
heatmap = sns.heatmap(df_heatmap, annot=True, fmt="d")

# Setting tick labels for heatmap
heatmap.yaxis.set_ticklabels(heatmap.yaxis.get_ticklabels(), rotation=0
, ha='right', fontsize=14)
heatmap.xaxis.set_ticklabels(heatmap.xaxis.get_ticklabels(), rotation=0
, ha='right', fontsize=14)
plt.ylabel('Predicted label',size=18)
plt.xlabel('True label',size=18)
plt.title("Confusion Matrix\n",size=24)
plt.show()
```



gbdt tfidf

In [152]:
```
# Please write all the code with proper documentation
tf_idf_vect = TfidfVectorizer(min_df=1000)
X_train_vec = tf_idf_vect.fit_transform(X_train)
X_test_vec = tf_idf_vect.transform(X_test)
print("the type of count vectorizer :",type(X_train_vec))
print("the shape of out text TFIDF vectorizer : ",X_train_vec.get_shape
())
print("the number of unique words :", X_train_vec.get_shape()[1])
```

```
# Data-preprocessing: Standardizing the data
sc = StandardScaler(with_mean=False)
X_train_vec_standardized = sc.fit_transform(X_train_vec)
X_test_vec_standardized = sc.transform(X_test_vec)
```

```
the type of count vectorizer : <class 'scipy.sparse.csr.csr_matrix'>
the shape of out text TFIDF vectorizer :  (14000, 65)
the number of unique words : 65
```

In [153]:

```
# Please write all the code with proper documentation
from sklearn.ensemble import GradientBoostingClassifier

base_learners = [5, 10, 50, 100, 200, 500, 1000]
depth = [2, 3, 4, 5, 6, 7, 8, 9, 10]

param_grid = {'n_estimators': base_learners,  'max_depth': depth}
GBC = GradientBoostingClassifier(max_features='sqrt',subsample=0.1)
model = GridSearchCV(GBC, param_grid, scoring = 'roc_auc', cv=3, n_jobs
 = -1,pre_dispatch=2)
model.fit(X_train_vec_standardized, Y_train)
print("Model with best parameters :\n",model.best_estimator_)
print("Accuracy of the model : ",model.score(X_test_vec_standardized, Y
_test))
```

```
Model with best parameters :
 GradientBoostingClassifier(criterion='friedman_mse', init=None,
              learning_rate=0.1, loss='deviance', max_depth=2,
              max_features='sqrt', 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=200,
              n_iter_no_change=None, presort='auto', random_state=None,
              subsample=0.1, tol=0.0001, validation_fraction=0.1,
              verbose=0, warm_start=False)
Accuracy of the model :  0.8163696845306079
```

In [0]:

```
gb = GradientBoostingClassifier(n_estimators=optimal_learners, max_dept
h=optimal_depth, max_features='sqrt', subsample=0.1)
```

```
gb.fit(X_train_vec_standardized,Y_train)
predictions = gb.predict(X_test_vec_standardized)
predictions1 = gb.predict(X_train_vec_standardized)
```

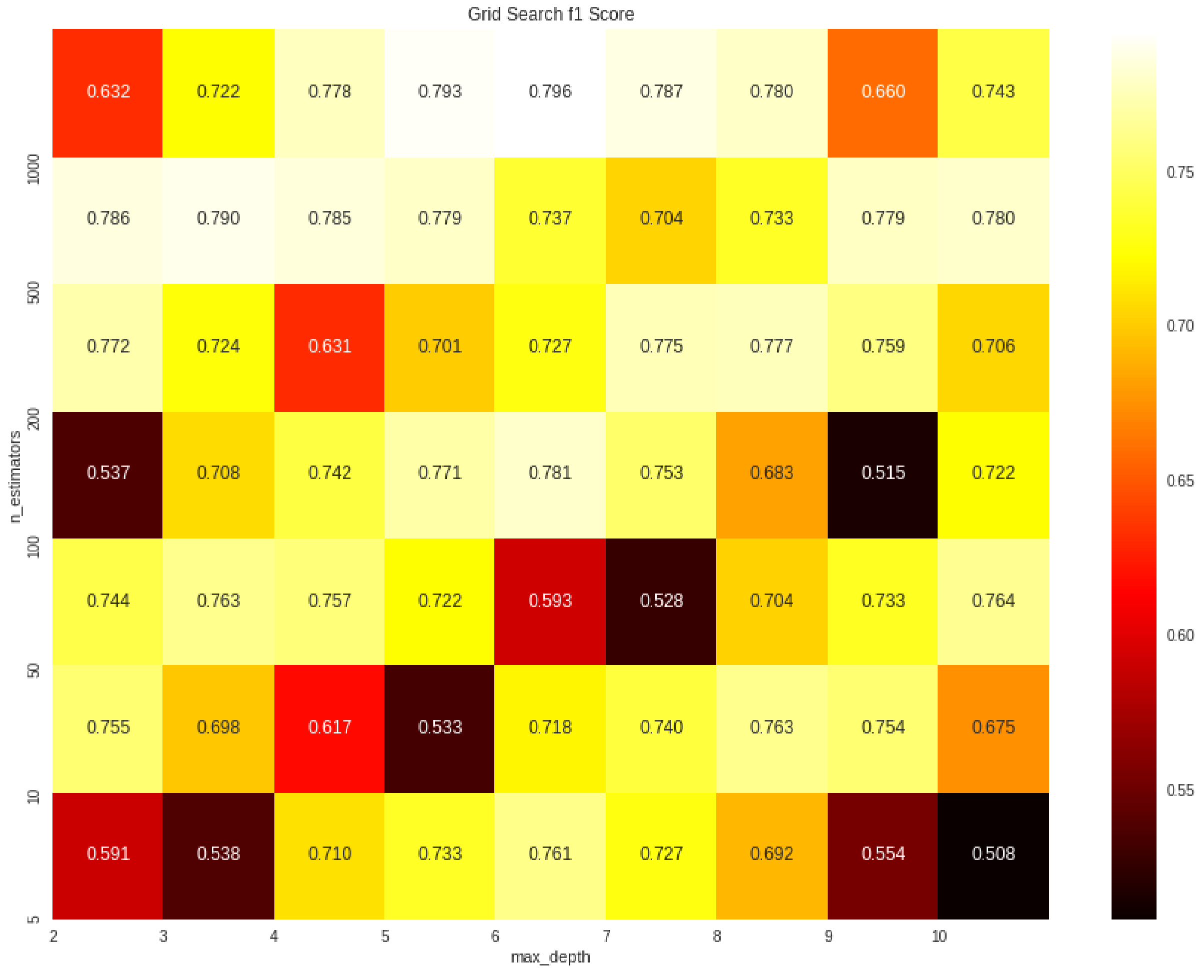
In [155]:

```
# Cross-Validation errors
cv_errors = [1-i for i in model.cv_results_['mean_test_score']]
training_scores=[1-i for i in model.cv_results_['mean_train_score']]

# Optimal value of number of base learners
optimal_learners = model.best_estimator_.n_estimators
print("The optimal number of base learners is : ",optimal_learners)


optimal_depth=model.best_estimator_.max_depth
print("The optimal number of depth is : ",optimal_depth)

# Variables that will be used for  making table in Conclusion part of t
his assignment
tfidf_gbdt_learners = optimal_learners
tfidf_gbdt_depth = optimal_depth
tfidf_gbdt_train_acc = model.score(X_test_vec, Y_test)*100
tfidf_gbdt_test_acc = accuracy_score(Y_test, predictions) * 100
```

```
The optimal number of base learners is :  200
The optimal number of depth is :  2
```

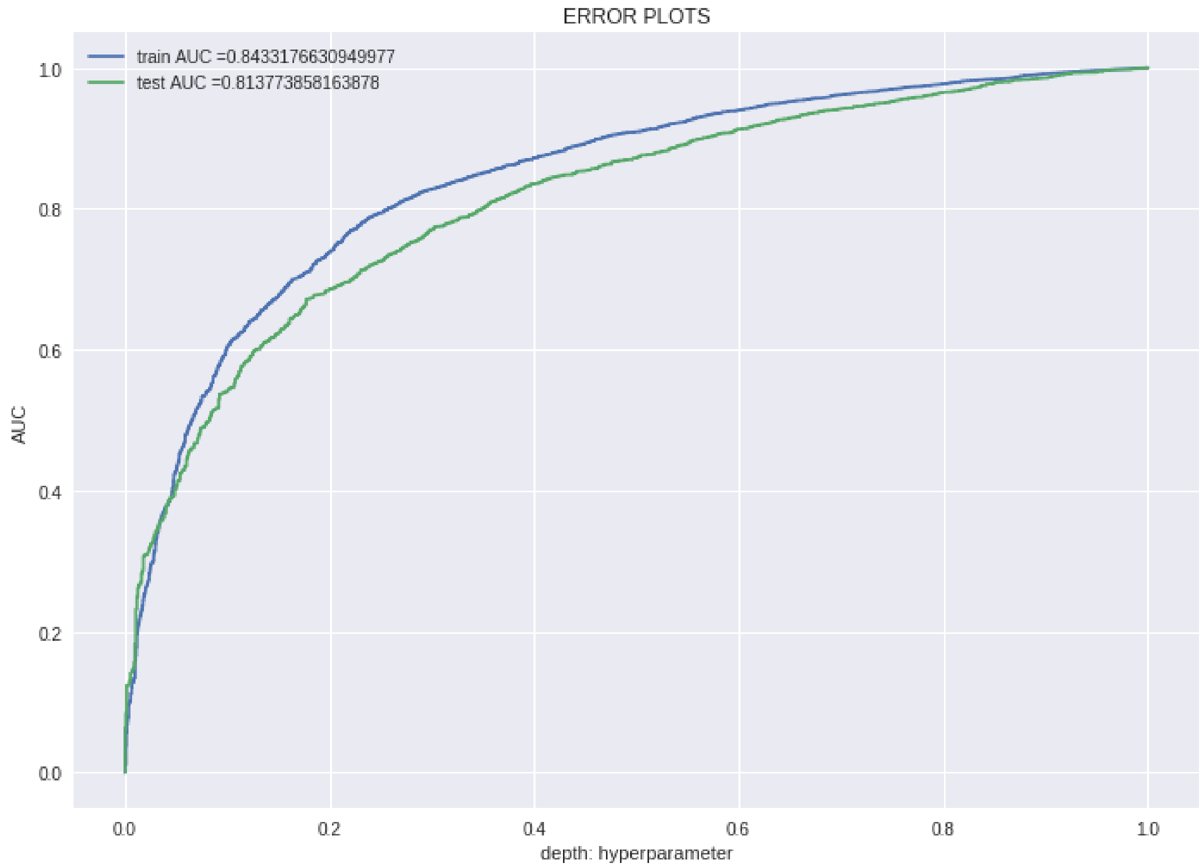
In [156]:

```
scores = model.cv_results_['mean_test_score'].reshape(len(base_learners
),len(depth))

plt.figure(figsize=(15, 11))
sns.heatmap(scores, annot=True, cmap=plt.cm.hot, fmt=".3f", xticklabels
=base_learners, yticklabels=depth)
plt.ylabel('n_estimators')
plt.xlabel('max_depth')
plt.yticks(np.arange(len(base_learners)), base_learners)
plt.xticks(np.arange(len(depth)), depth)
plt.title('Grid Search f1 Score')
plt.show()
```



In [157]:
```
train_fpr, train_tpr, thresholds = roc_curve(Y_train, gb.predict_proba(
X_train_vec_standardized)[:,1])
test_fpr, test_tpr, thresholds = roc_curve(Y_test, gb.predict_proba(X_t
est_vec_standardized)[:,1])

plt.plot(train_fpr, train_tpr, label="train AUC ="+str(auc(train_fpr, t
rain_tpr)))
plt.plot(test_fpr, test_tpr, label="test AUC ="+str(auc(test_fpr, test_
tpr)))
```

```
plt.legend()
plt.xlabel("depth: hyperparameter")
plt.ylabel("AUC")
plt.title("ERROR PLOTS")
plt.show()
```



In [158]:

```
# evaluate accuracy on test data
acc = accuracy_score(Y_test, predictions) * 100
print('\nThe Test Accuracy of the DecisionTreeClassifier for depth = %d
 is %f%%' % (optimal_depth, acc))
print('\nThe Test Accuracy of the DecisionTreeClassifier for depth = %d
 is %f%%' % (optimal_learners, acc))
```

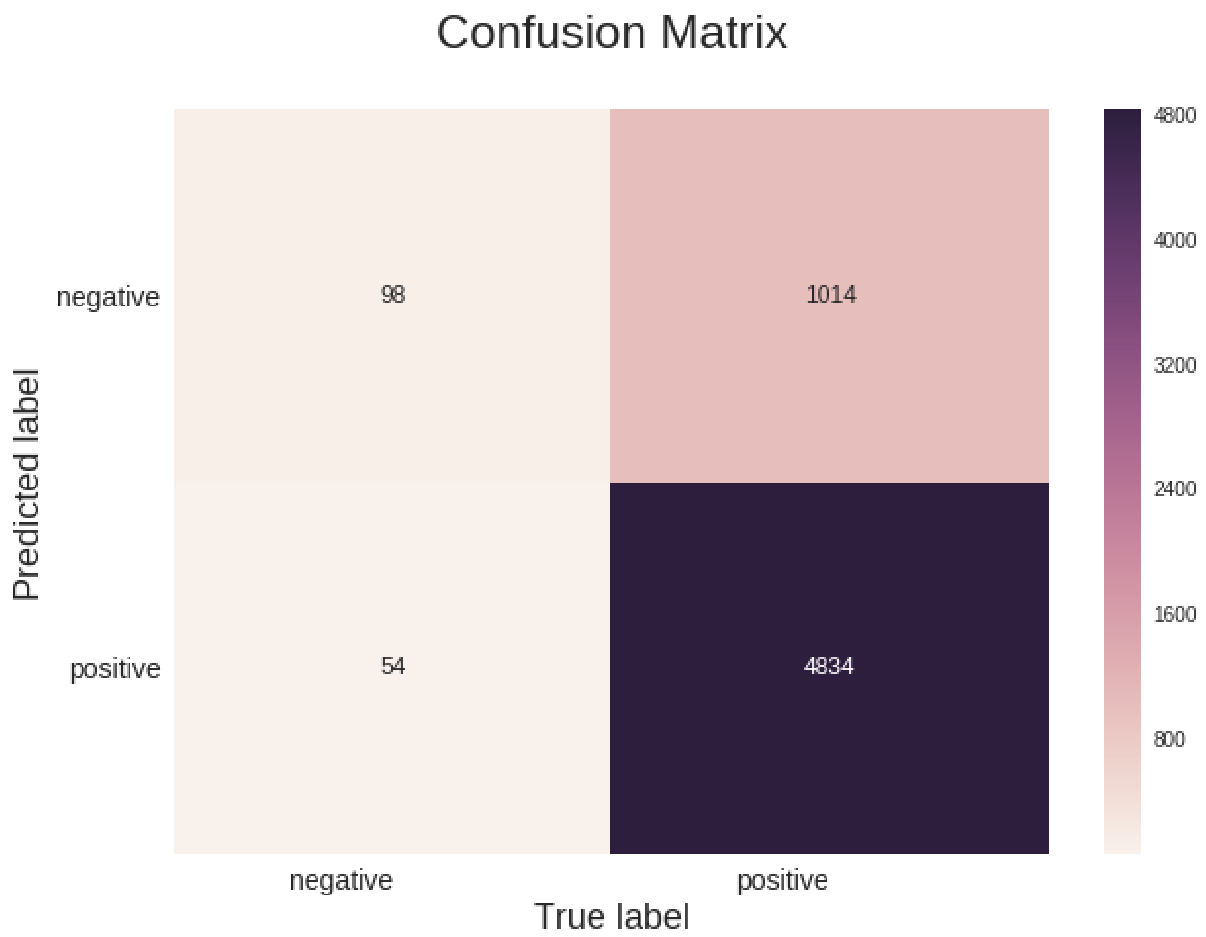
```
# evaluate precision
acc = precision_score(Y_test, predictions, pos_label = 1)
print('\nThe Test Precision of the DecisionTreeClassifier for depth = %
d is %f' % (optimal_depth, acc))
print('\nThe Test Precision of the DecisionTreeClassifier for depth = %
d is %f' % (optimal_learners, acc))

# evaluate recall
acc = recall_score(Y_test, predictions, pos_label = 1)
print('\nThe Test Recall of the DecisionTreeClassifier for depth = %d i
s %f' % (optimal_depth, acc))
print('\nThe Test Recall of the DecisionTreeClassifier for depth = %d i
s %f' % (optimal_learners, acc))

# evaluate f1-score
acc = f1_score(Y_test, predictions, pos_label = 1)
print('\nThe Test F1-Score of the DecisionTreeClassifier for depth = %d
 is %f' % (optimal_depth, acc))
print('\nThe Test F1-Score of the DecisionTreeClassifier for depth = %d
 is %f' % (optimal_learners, acc))
```

```
The Test Accuracy of the DecisionTreeClassifier for depth = 2 is 82.200
000%

The Test Accuracy of the DecisionTreeClassifier for depth = 200 is 82.2
00000%

The Test Precision of the DecisionTreeClassifier for depth = 2 is 0.826
607

The Test Precision of the DecisionTreeClassifier for depth = 200 is 0.8
26607

The Test Recall of the DecisionTreeClassifier for depth = 2 is 0.988953

The Test Recall of the DecisionTreeClassifier for depth = 200 is 0.9889
53

The Test F1-Score of the DecisionTreeClassifier for depth = 2 is 0.9005
22
```

```
The Test F1-Score of the DecisionTreeClassifier for depth = 200 is 0.90
0522
```

In [159]:

```python
# Code for drawing seaborn heatmaps on test data
class_names = ['negative','positive']
df_heatmap = pd.DataFrame(confusion_matrix(Y_test, predictions), index=
class_names, columns=class_names )
fig = plt.figure(figsize=(10,7))
heatmap = sns.heatmap(df_heatmap, annot=True, fmt="d")

# Setting tick labels for heatmap
heatmap.yaxis.set_ticklabels(heatmap.yaxis.get_ticklabels(), rotation=0
, ha='right', fontsize=14)
heatmap.xaxis.set_ticklabels(heatmap.xaxis.get_ticklabels(), rotation=0
, ha='right', fontsize=14)
plt.ylabel('Predicted label',size=18)
plt.xlabel('True label',size=18)
plt.title("Confusion Matrix\n",size=24)
plt.show()
```



In [160]:
```
# evaluate accuracy on train data
acc = accuracy_score(Y_train, predictions1) * 100
print('\nThe Train Accuracy of the DecisionTreeClassifier for depth = %d is %f%%' % (optimal_depth, acc))
print('\nThe Train Accuracy of the DecisionTreeClassifier for depth = %d is %f%%' % (optimal_learners, acc))

# evaluate precision
acc = precision_score(Y_train, predictions1, pos_label = 1)
print('\nThe Train Precision of the DecisionTreeClassifier for depth =
```

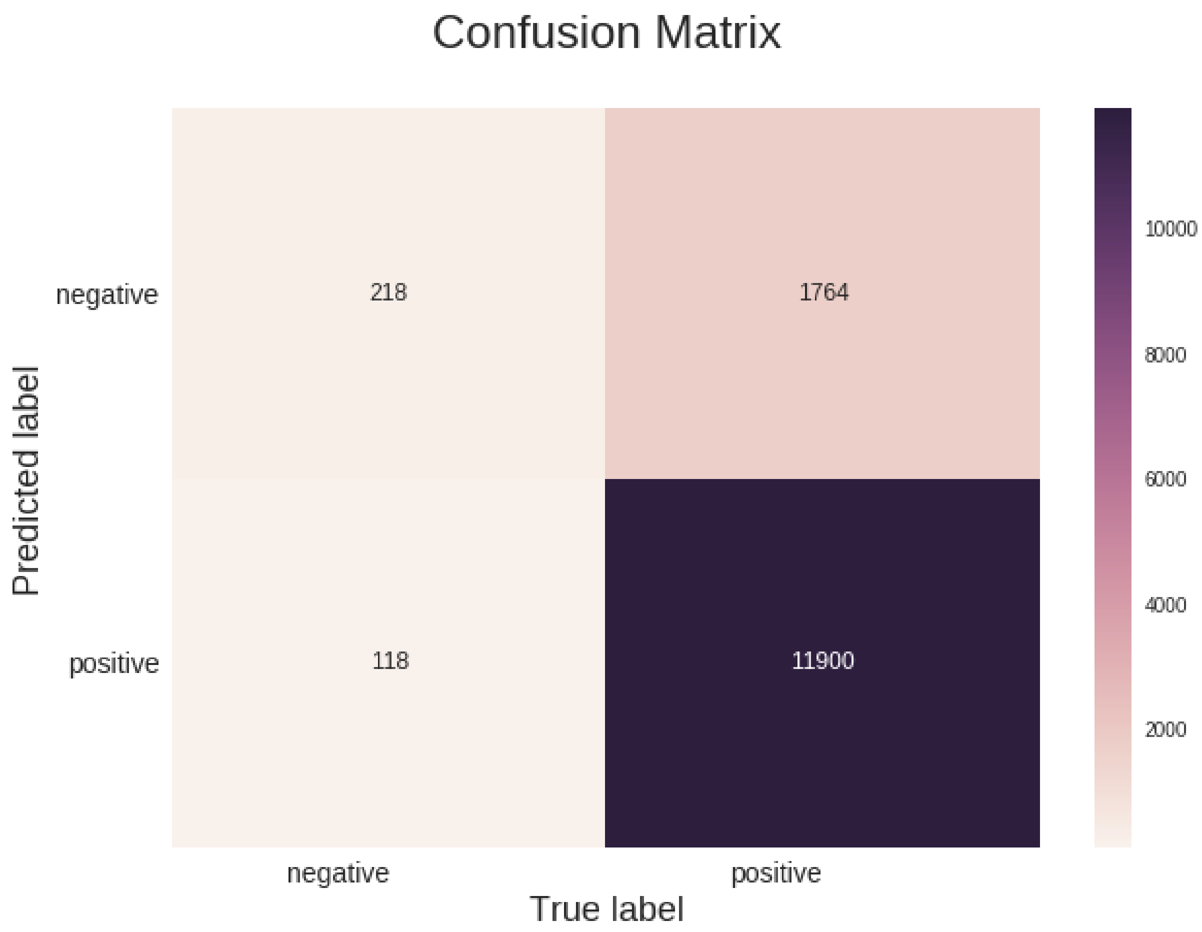
```
%d is %f' % (optimal_depth, acc))
print('\nThe Train Precision of the DecisionTreeClassifier for depth =
%d is %f' % (optimal_learners, acc))

# evaluate recall
acc = recall_score(Y_train, predictions1, pos_label = 1)
print('\nThe Train Recall of the DecisionTreeClassifier for depth = %d
 is %f' % (optimal_depth, acc))
print('\nThe Train Recall of the DecisionTreeClassifier for depth = %d
 is %f' % (optimal_learners, acc))


# evaluate f1-score
acc = f1_score(Y_train, predictions1, pos_label = 1)
print('\nThe Train F1-Score of the DecisionTreeClassifier for depth = %
d is %f' % (optimal_depth, acc))
print('\nThe Train F1-Score of the DecisionTreeClassifier for depth = %
d is %f' % (optimal_learners, acc))
```

```
The Train Accuracy of the DecisionTreeClassifier for depth = 2 is 86.55
7143%

The Train Accuracy of the DecisionTreeClassifier for depth = 200 is 86.
557143%

The Train Precision of the DecisionTreeClassifier for depth = 2 is 0.87
0902

The Train Precision of the DecisionTreeClassifier for depth = 200 is 0.
870902

The Train Recall of the DecisionTreeClassifier for depth = 2 is 0.99018
1

The Train Recall of the DecisionTreeClassifier for depth = 200 is 0.990
181

The Train F1-Score of the DecisionTreeClassifier for depth = 2 is 0.926
719
```

```
The Train F1-Score of the DecisionTreeClassifier for depth = 200 is 0.9
26719
```

In [161]:

```
# Code for drawing seaborn heatmaps
class_names = ['negative','positive']
df_heatmap = pd.DataFrame(confusion_matrix(Y_train, predictions1), inde
x=class_names, columns=class_names )
fig = plt.figure(figsize=(10,7))
heatmap = sns.heatmap(df_heatmap, annot=True, fmt="d")

# Setting tick labels for heatmap
heatmap.yaxis.set_ticklabels(heatmap.yaxis.get_ticklabels(), rotation=0
, ha='right', fontsize=14)
heatmap.xaxis.set_ticklabels(heatmap.xaxis.get_ticklabels(), rotation=0
, ha='right', fontsize=14)
plt.ylabel('Predicted label',size=18)
plt.xlabel('True label',size=18)
plt.title("Confusion Matrix\n",size=24)
plt.show()
```



In [163]:
```
# Table for Gradient Boosting Decision Tree (GBDT)
names = ['GBDT for BoW','GBDT for TFIDF']

base_learners = [bow_gbdt_learners,tfidf_gbdt_learners]

optimal_depth = [bow_gbdt_depth,tfidf_gbdt_depth]

train_acc = [bow_gbdt_train_acc,tfidf_gbdt_train_acc]

test_acc = [bow_gbdt_test_acc,tfidf_gbdt_test_acc]
```

```
numbering = [1,2]

# Initializing prettytable
table = PrettyTable()

# Adding columns
table.add_column("S.NO.",numbering)
table.add_column("MODEL",names)
table.add_column("Base Learners ",base_learners)
table.add_column("Optimal_Depth",optimal_depth)
table.add_column("Training Accuracy",train_acc)
table.add_column("Test Accuracy",test_acc)

print('\t\t\t\tTABLE FOR GRADIENT BOOSTING DECISION TREE (GBDT)')
# Printing the Table
print(table)
```

```
                                TABLE FOR GRADIENT BOOSTING DECISION TR
EE (GBDT)
+-------+----------------+----------------+---------------+------------
-------+------------------+
| S.NO. |     MODEL      | Base Learners  | Optimal_Depth | Training Ac
curacy |   Test Accuracy   |
+-------+----------------+----------------+---------------+------------
-------+------------------+
|   1   |  GBDT for BoW  |      500       |       2       | 75.31048679
026388 | 80.36666666666666 |
|   2   | GBDT for TFIDF |      200       |       2       | 75.25522237
692661 | 82.19999999999999 |
+-------+----------------+----------------+---------------+------------
-------+------------------+
```

after feature engineering accuracy got slightly increased