Amazon Fine Food Reviews Analysis

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

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

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

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

Attribute Information:

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

Objective:

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

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

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

[1]. Reading Data

[1.1] Loading the data

The dataset is available in two forms

1. .csv file
2. SQLite Database

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

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

In [1]:

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


import sqlite3
import pandas as pd
import numpy as np
import nltk
import string
import matplotlib.pyplot as plt
```

```
import seaborn as sns
from sklearn.feature_extraction.text import TfidfTransformer
from sklearn.feature_extraction.text import TfidfVectorizer

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

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

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

from tqdm import tqdm
import os
```

```
C:\Users\hemant\AnacondaNew\lib\site-packages\smart_open\ssh.py:34: UserWarning: paramiko missing, opening SSH/SCP/SFTP paths will be disabled.  `pip install paramiko` to suppress
  warnings.warn('paramiko missing, opening SSH/SCP/SFTP paths will be disabled.  `pip install paramiko` to suppress')
C:\Users\hemant\AnacondaNew\lib\site-packages\gensim\utils.py:1197: UserWarning: detected Windows; aliasing chunkize to chunkize_serial
  warnings.warn("detected Windows; aliasing chunkize to chunkize_serial")
```

In [2]:

```
# using SQLite Table to read data.
con = sqlite3.connect(r'G:\database_assignment\Logistic_regression\database5.sqlite')

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

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

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

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

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

```
Number of data points in our data (525814, 10)
```

Out[2]:

	Id	ProductId	UserId	ProfileName	HelpfulnessNumerator	HelpfulnessDenominator	Score	Time	Summary
0	1	B001E4KFG0	A3SGXH7AUHU8GW	delmartian	1	1	1	1303862400	Good Quality Dog Food

	Id	ProductId	UserId	ProfileName	HelpfulnessNumerator	HelpfulnessDenominator	Score	Time	Summary
1	2	B00813GRG4	A1D87F6ZCVE5NK	dll pa	0	0	0	1346976000	Not as Advertised
2	3	B000LQOCH0	ABXLMWJIXXAIN	Natalia Corres "Natalia Corres"	1	1	1	1219017600	"Delight" says it all

In [3]:

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

In [4]:

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

```
(80668, 7)
```

Out[4]:

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

In [5]:

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

Out[5]:

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

In [6]:

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

Out[6]:

```
393063
```

[2] Exploratory Data Analysis

[2.1] Data Cleaning: Deduplication

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

In [7]:

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

Out[7]:

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

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

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

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

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

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

In [8]:

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

In [9]:

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

Out[9]:

```
(364173, 10)
```

In [10]:

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

Out[10]:

```
69.25890143662969
```

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

In [11]:

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

display.head()
```

Out[11]:

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

In [12]:

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

In [13]:

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

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

```
(364171, 10)
```

Out[13]:

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

[3] Preprocessing

[3.1]. Preprocessing Review Text

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

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

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

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

In [14]:

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

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

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

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

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

In [15]:

```python
# remove urls from text python: https://stackoverflow.com/a/40823105/4084039
sent_0 = re.sub(r"http\S+", "", sent_0)
sent_1000 = re.sub(r"http\S+", "", sent_1000)
sent_150 = re.sub(r"http\S+", "", sent_1500)
sent_4900 = re.sub(r"http\S+", "", sent_4900)

print(sent_0)
```

```
this witty little book makes my son laugh at loud. i recite it in the car as we're driving along a
nd he always can sing the refrain. he's learned about whales, India, drooping roses:  i love all t
```

```
he new words this book  introduces and the silliness of it all.  this is a classic book i am
willing to bet my son will STILL be able to recite from memory when he is  in college
```

In [16]:

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

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

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

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

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

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

In [17]:

```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)

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

In [18]:

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

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

In [19]:

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

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

In [20]:

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

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

In [21]:

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

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

```
            "mustn't", 'needn', "needn't", 'shan', "shan't", 'shouldn', "shouldn't", 'wasn',
"wasn't", 'weren', "weren't", \
            'won', "won't", 'wouldn', "wouldn't"])
```

In [22]:

```
# Combining all the above stundents
if not os.path.isfile('final.sqlite'):

    from tqdm import tqdm
    final_string=[]
    # 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)
        final_string.append(sentance.strip())

         #############---- storing the data into .sqlite file ------########################
    final['CleanedText']=final_string #adding a column of CleanedText which displays the data afte
r pre-processing of the review
    final['CleanedText']=final['CleanedText'].str.decode("utf-8")
        # store final table into an SQlLite table for future.
    conn = sqlite3.connect('final.sqlite')
    c=conn.cursor()
    conn.text_factory = str
    final05.to_sql('Reviews', conn,  schema=None, if_exists='replace', \
                 index=True, index_label=None, chunksize=None, dtype=None)
    conn.close()
```

In [23]:

```
if os.path.isfile('final.sqlite'):
    conn = sqlite3.connect('final.sqlite')
    final1 = pd.read_sql_query(""" SELECT * FROM Reviews WHERE Score != 3 """, conn)
    conn.close()
else:
    print("Please the above cell")
```

In [24]:

```
final1.head(3)
final1['CleanedText'].head(5)
```

Out[24]:

```
0    witti littl book make son laugh loud recit car...
1    grew read sendak book watch realli rosi movi i...
2    fun way children learn month year learn poem t...
3    great littl book read nice rhythm well good re...
4    book poetri month year goe month cute littl po...
Name: CleanedText, dtype: object
```

[3.2] Preprocessing Review Summary

In [25]:

```
sorted_sample = final1.sort_values('Time', axis=0, ascending=True, inplace=False, kind='quicksort',
na_position='last')
sample_60000 = sorted_sample.iloc[0:100000]
final.shape
y = sample_60000['Score']
```

In [26]:

```
sample_60000.shape
```

```
sample_60000.shape
```

Out[26]:

```
(100000, 12)
```

In [27]:

```
sample_60000["length"] = sample_60000['Text'].apply(len)
```

In [28]:

```
sample_60000.shape
```

Out[28]:

```
(100000, 13)
```

In [29]:

```
y.shape
```

Out[29]:

```
(100000,)
```

In [30]:

```
sample_60000.head(3)
```

Out[30]:

	index	Id	ProductId	UserId	ProfileName	HelpfulnessNumerator	HelpfulnessDenominator	Score	Time
0	138706	150524	0006641040	ACITT7DI6IDDL	shari zychinski	0	0	1	939340800
30	138683	150501	0006641040	AJ46FKXOVC7NR	Nicholas A Mesiano	2	2	1	940809600
424	417839	451856	B00004CXX9	AIUWLEQ1ADEG5	Elizabeth Medina	0	0	1	944092800

In [31]:

```
sample_60000['Score'].value_counts()
```

Out[31]:

```
1    87729
0    12271
Name: Score, dtype: int64
```

In [32]:

```
from sklearn.model_selection import train_test_split

x_train, x_ts, y_train, y_ts = train_test_split(sample_60000, y, test_size=0.33) # this is random s
plitting
```

In [33]:

```
x_train.shape
```

Out[33]:

```
(67000, 13)
```

In [34]:

```
y_train.shape
```

Out[34]:

```
(67000,)
```

[4] Featurization

[4.1] BAG OF WORDS

[4.2] Bi-Grams and n-Grams.

In [35]:

```
#bi-gram, tri-gram and n-gram
from sklearn import preprocessing


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

# you can choose these numebrs min_df=10, max_features=5000, of your choice
count_vect = CountVectorizer(ngram_range=(1, 2),min_df=10) #in scikit-learn
x_tr_final_counts_bigram = count_vect.fit_transform(x_train['CleanedText'].values)
#x_cv_final_counts_bigram = count_vect.transform(x_cv['CleanedText'].values)
x_ts_final_counts_bigram = count_vect.transform(x_ts['CleanedText'].values)


print("the type of count vectorizer ",type(x_tr_final_counts_bigram))
print("the shape of out text BOW vectorizer ",x_tr_final_counts_bigram.get_shape())
print("the number of unique words ", x_tr_final_counts_bigram.get_shape()[1])

#print("the type of count vectorizer ",type(x_cv_final_counts_bigram))
#print("the shape of out text BOW vectorizer ",x_cv_final_counts_bigram.get_shape())
#print("the number of unique words ", x_cv_final_counts_bigram.get_shape()[1])

print("the type of count vectorizer ",type(x_ts_final_counts_bigram))
print("the shape of out text BOW vectorizer ",x_ts_final_counts_bigram.get_shape())
print("the number of unique words ", x_ts_final_counts_bigram.get_shape()[1])


x_tr_final_counts_bigram = preprocessing.normalize(x_tr_final_counts_bigram)
#x_cv_final_counts_bigram = preprocessing.normalize(x_cv_final_counts_bigram)
x_ts_final_counts_bigram = preprocessing.normalize(x_ts_final_counts_bigram)
```

```
the type of count vectorizer  <class 'scipy.sparse.csr.csr_matrix'>
the shape of out text BOW vectorizer  (67000, 38798)
the number of unique words  38798
the type of count vectorizer  <class 'scipy.sparse.csr.csr_matrix'>
the shape of out text BOW vectorizer  (33000, 38798)
the number of unique words  38798
```

[4.3] TF-IDF

In [36]:

```
tf_idf_vect = TfidfVectorizer(ngram_range=(1, 2),min_df=10)
x_tr_final_counts_tfidf = tf_idf_vect.fit_transform(x_train['CleanedText'].values)
#x_cv_final_counts_tfidf = tf_idf_vect.transform(x_cv['CleanedText'].values)
x_ts_final_counts_tfidf = tf_idf_vect.transform(x_ts['CleanedText'].values)


print("the type of count vectorizer ",type(x_tr_final_counts_tfidf))
print("the shape of out text TFIDF vectorizer ",x_tr_final_counts_tfidf.get_shape())
print("the number of unique words including both unigrams and bigrams ", x_tr_final_counts_tfidf.g
et_shape()[1])

#print("the type of count vectorizer ",type(x_cv_final_counts_tfidf))
#print("the shape of out text TFIDF vectorizer ",x_cv_final_counts_tfidf.get_shape())
#print("the number of unique words including both unigrams and bigrams ",
x_cv_final_counts_tfidf.get_shape()[1])

print("the type of count vectorizer ",type(x_ts_final_counts_tfidf))
print("the shape of out text TFIDF vectorizer ",x_ts_final_counts_tfidf.get_shape())
print("the number of unique words including both unigrams and bigrams ", x_ts_final_counts_tfidf.g
et_shape()[1])


x_tr_final_counts_tfidf = preprocessing.normalize(x_tr_final_counts_tfidf)
#x_cv_final_counts_tfidf = preprocessing.normalize(x_cv_final_counts_tfidf)
x_ts_final_counts_tfidf = preprocessing.normalize(x_ts_final_counts_tfidf)
```

```
the type of count vectorizer  <class 'scipy.sparse.csr.csr_matrix'>
the shape of out text TFIDF vectorizer  (67000, 38798)
the number of unique words including both unigrams and bigrams  38798
the type of count vectorizer  <class 'scipy.sparse.csr.csr_matrix'>
the shape of out text TFIDF vectorizer  (33000, 38798)
the number of unique words including both unigrams and bigrams  38798
```

[4.4] Word2Vec

In [37]:

```
# Train your own Word2Vec model using your own text corpus
i=0
list_of_sentance_train=[]
for sentance in x_train['CleanedText'].values:
    list_of_sentance_train.append(sentance.split())
```

In [38]:

```
# Train your own Word2Vec model using your own text corpus
#i=0
#list_of_sentance_cv=[]
#for sentance in x_cv['CleanedText'].values:
#    list_of_sentance_cv.append(sentance.split())
```

In [39]:

```
# Train your own Word2Vec model using your own text corpus
i=0
list_of_sentance_ts=[]
for sentance in x_ts['CleanedText'].values:
    list_of_sentance_ts.append(sentance.split())
```

In [40]:

```
print(len(list_of_sentance_train))
#print(len(list_of_sentance_cv))
print(len(list_of_sentance_ts))
```

```
67000
33000
```

In [41]:

```
def convertByteStringtoString(sentlist):
    for x in sentlist:
        for i in range(len(x)):
            x[i] = x[i]

    return sentlist
```

In [42]:

```
list_of_sentance_train = convertByteStringtoString(list_of_sentance_train)
#list_of_sentance_cv = convertByteStringtoString(list_of_sentance_cv)
list_of_sentance_ts = convertByteStringtoString(list_of_sentance_ts)
```

In [43]:

```
# min_count = 5 considers only words that occured atleast 5 times
w2v_model=Word2Vec(list_of_sentance_train,min_count=5,size=50, workers=4)
```

In [44]:

```
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  10544
sample words  ['dog', 'love', 'seem', 'keep', 'teeth', 'clean', 'ship', 'fast', 'great', 'price',
'primari', 'ingredi', 'potato', 'flake', 'guess', 'itll', 'fill', 'noth', 'els', 'realli',
'second', 'chicken', 'meal', 'that', 'sourc', 'protein', 'even', 'real', 'organ', 'doesnt',
'mean', 'good', 'still', 'low', 'qualiti', 'absolut', 'littl', 'cracker', 'much', 'healthier', 'al
tern', 'goldfish', 'snack', 'expir', 'date', 'give', 'year', 'consum', 'case', 'order']
```

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

[4.4.1.1] Avg W2v

In [45]:

```
# average Word2Vec
# compute average word2vec for each review.

train_avgw2v = []; # the avg-w2v for each sentence/review is stored in this list
for sent in tqdm(list_of_sentance_train): # for each review/sentence
    sent_vec = np.zeros(50) # as word vectors are of zero length 50, you might need to change this
to 300 if you use google's w2v
    cnt_words =0; # num of words with a valid vector in the sentence/review
    for word in sent: # for each word in a review/sentence
        if word in w2v_words:
            vec = w2v_model.wv[word]
            sent_vec += vec
            cnt_words += 1
    if cnt_words != 0:
        sent_vec /= cnt_words
    train_avgw2v.append(sent_vec)
print(len(train_avgw2v))
print(len(train_avgw2v[0]))
```

```
100%|███████████████████████████████████| 67000/67000 [02:00<00:00, 557.78it/s]
```

```
67000
50
```

In [46]:

```
"""
cv_avgw2v = []; # the avg-w2v for each sentence/review is stored in this list
for sent in tqdm(list_of_sentance_cv): # for each review/sentence
    sent_vec = np.zeros(50) # as word vectors are of zero length 50, you might need to change this
```

```
    sent_vec = np.zeros(50) # as word vectors are of zero length 50, you might need to change this
to 300 if you use google's w2v
    cnt_words =0; # num of words with a valid vector in the sentence/review
    for word in sent: # for each word in a review/sentence
        if word in w2v_words:
            vec = w2v_model.wv[word]
            sent_vec += vec
            cnt_words += 1
    if cnt_words != 0:
        sent_vec /= cnt_words
    cv_avgw2v.append(sent_vec)
print(len(cv_avgw2v))
print(len(cv_avgw2v[0]))
"""
```

Out[46]:

```
"\ncv_avgw2v = []; # the avg-w2v for each sentence/review is stored in this list\nfor sent in tqdm(list_of_sentance_cv): # for each review/sentence\n    sent_vec = np.zeros(50) # as word vectors are of zero length 50, you might need to change this to 300 if you use google's w2v\n    cnt_words =0; # num of words with a valid vector in the sentence/review\n    for word in sent: # for each word in a review/sentence\n        if word in w2v_words:\n            vec = w2v_model.wv[word]\n            sent_vec += vec\n            cnt_words += 1\n    if cnt_words != 0:\n        sent_vec /= cnt_words\n    cv_avgw2v.append(sent_vec)\nprint(len(cv_avgw2v))\nprint(len(cv_avgw2v[0]))\n"
```

In [47]:

```
test_avgw2v = []; # the avg-w2v for each sentence/review is stored in this list
for sent in tqdm(list_of_sentance_ts): # for each review/sentence
    sent_vec = np.zeros(50) # as word vectors are of zero length 50, you might need to change this
to 300 if you use google's w2v
    cnt_words =0; # num of words with a valid vector in the sentence/review
    for word in sent: # for each word in a review/sentence
        if word in w2v_words:
            vec = w2v_model.wv[word]
            sent_vec += vec
            cnt_words += 1
    if cnt_words != 0:
        sent_vec /= cnt_words
    test_avgw2v.append(sent_vec)
print(len(test_avgw2v))
print(len(test_avgw2v[0]))
```

```
100%|██████████████████████████████████████████| 33000/33000 [01:00<00:00, 541.83it/s]
```

```
33000
50
```

In [48]:

```
train_avgw2v = preprocessing.normalize(train_avgw2v)
#cv_avgw2v = preprocessing.normalize(cv_avgw2v)
test_avgw2v = preprocessing.normalize(test_avgw2v)
```

In [49]:

```
train_avgw2v = np.array(train_avgw2v)
#cv_avgw2v = np.array(cv_avgw2v)
test_avgw2v = np.array(test_avgw2v)
```

In [50]:

```
np.isnan(train_avgw2v).any()
```

Out[50]:

```
False
```

In [51]:

```
#np.isnan(cv_avgw2v).any()
```

In [52]:

```
np.isnan(test_avgw2v).any()
```

Out[52]:

```
False
```

[4.4.1.2] TFIDF weighted W2v

In [53]:

```
# S = ["abc def pqr", "def def def abc", "pqr pqr def"]
model = TfidfVectorizer()
x_tr_final_counts_TFIDF_w2v = model.fit_transform(x_train['CleanedText'].values)
#x_cv_final_counts_TFIDF_w2v = model.transform(x_cv['CleanedText'].values)
x_ts_final_counts_TFIDF_w2v = model.transform(x_ts['CleanedText'].values)

# we are converting a dictionary with word as a key, and the idf as a value
dictionary = dict(zip(model.get_feature_names(), list(model.idf_)))
```

In [54]:

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

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

```
100%|██████████████████████████████| 67000/67000 [23:56<00:00, 46.65it/s]
```

In [ ]:

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

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

```
            sent_vec += (vec * tf_idf)
            weight_sum += tf_idf
    if weight_sum != 0:
        sent_vec /= weight_sum
    tfidf_sent_vectors_cv.append(sent_vec)
    row += 1
 """
```

In [55]:

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

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

```
100%|████████████████████████████████████| 33000/33000 [11:41<00:00, 47.06it/s]
```

In [56]:

```
tfidf_sent_vectors = preprocessing.normalize(tfidf_sent_vectors)
#tfidf_sent_vectors_cv = preprocessing.normalize(tfidf_sent_vectors_cv)
tfidf_sent_vectors_ts = preprocessing.normalize(tfidf_sent_vectors_ts)
```

In [57]:

```
tfidf_sent_vectors = np.array(tfidf_sent_vectors)
#tfidf_sent_vectors_cv = np.array(tfidf_sent_vectors_cv)
tfidf_sent_vectors_ts = np.array(tfidf_sent_vectors_ts)
```

In [58]:

```
np.isnan(tfidf_sent_vectors).any()
```

Out[58]:

```
False
```

In [59]:

```
#np.isnan(tfidf_sent_vectors_cv).any()
```

In [60]:

```
np.isnan(tfidf_sent_vectors_ts).any()
```

Out[60]:

```
False
```

In [61]:

```
#To show how Time Series Split splits the data
from sklearn.model_selection import TimeSeriesSplit
tscv = TimeSeriesSplit(n_splits=7)
for train, cv in tscv.split(x_tr_final_counts_bigram):
    print("%s %s" % (train, cv))
#    print(x_tr_final_counts_bigram[train].shape,x_tr_final_counts_bigram[cv].shape)
```

```
[    0     1     2 ... 8372 8373 8374] [ 8375  8376  8377 ... 16747 16748 16749]
[    0     1     2 ... 16747 16748 16749] [16750 16751 16752 ... 25122 25123 25124]
[    0     1     2 ... 25122 25123 25124] [25125 25126 25127 ... 33497 33498 33499]
[    0     1     2 ... 33497 33498 33499] [33500 33501 33502 ... 41872 41873 41874]
[    0     1     2 ... 41872 41873 41874] [41875 41876 41877 ... 50247 50248 50249]
[    0     1     2 ... 50247 50248 50249] [50250 50251 50252 ... 58622 58623 58624]
[    0     1     2 ... 58622 58623 58624] [58625 58626 58627 ... 66997 66998 66999]
```

[5] Assignment 8: Decision Trees

1. **Apply Decision Trees 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 (best `depth` in range [1, 5, 10, 50, 100, 500, 100], and the best `min_samples_split` in range [5, 10, 100, 500])**
    - 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. **Graphviz**
    - Visualize your decision tree with Graphviz. It helps you to understand how a decision is being made, given a new vector.
    - Since feature names are not obtained from word2vec related models, visualize only BOW & TFIDF decision trees using Graphviz
    - Make sure to print the words in each node of the decision tree instead of printing its index.
    - Just for visualization purpose, limit max_depth to 2 or 3 and either embed the generated images of graphviz in your notebook, or directly upload them as .png files.
4. **Feature importance**
    - Find the top 20 important features from both feature sets Set 1 and Set 2 using `feature_importances_` method of Decision Tree Classifier and print their corresponding feature names
5. **Feature engineering**
    - To increase the performance of your model, you can also experiment with with feature engineering like :
        - Taking length of reviews as another feature.
        - Considering some features from review summary as well.
6. **Representation of results**
    - You need to plot the performance of model both on train data and cross validation data for each hyper parameter, like shown in the figure.
    - Once after you found the best hyper parameter, you need to train your model with it, and find the AUC on test data and plot the ROC curve on both train and test.
    - Along with plotting ROC curve, you need to print the confusion matrix with predicted and original labels of test data points. Please visualize your confusion matrices using seaborn heatmaps.
7. **Conclusion**
    - You need to summarize the results at the end of the notebook, summarize it in the table format. To print out a table please refer to this prettytable library link

Note: Data Leakage

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

Applying Decision Trees

[5.1] Applying Decision Trees on BOW, SET 1

In [62]:

```
# Please write all the code with proper documentation
from sklearn.linear_model import LogisticRegression
from sklearn.metrics import f1_score
from sklearn.metrics import roc_auc_score
from sklearn.metrics import accuracy_score
from math import log
from sklearn.model_selection import RandomizedSearchCV
from sklearn.model_selection import GridSearchCV
from sklearn.tree import DecisionTreeClassifier


alpha_values = np.arange(7)
#C = np.array([0.0001,0.001,0.01,0.1,1,10,100,500,1000,10000])
max_depth = np.array([1, 5, 10, 50, 100, 500, 1000])
min_samples_split = np.array([5, 10, 100, 500])
cv_auc1 = np.empty(len(alpha_values))
train_auc1 = np.empty(len(alpha_values))


neigh = DecisionTreeClassifier()
#params we need to try on classifier
param_grid = {'max_depth':[1, 5, 10, 50, 100, 500, 1000],
             'min_samples_split':[5, 10, 100, 500] ,'class_weight':['balanced']}
tscv = TimeSeriesSplit(n_splits=10) #For time based splitting
clf = RandomizedSearchCV(neigh,param_grid,cv=tscv,verbose=1)
clf.fit(x_tr_final_counts_bigram,y_train)


train_auc1 = clf.cv_results_['mean_train_score']
train_auc_std= clf.cv_results_['std_train_score']
cv_auc1 = clf.cv_results_['mean_test_score']
cv_auc_std= clf.cv_results_['std_test_score']


print("Best max_depth is:-",clf.best_estimator_.max_depth)
print("Best min_samples_split is:-",clf.best_estimator_.min_samples_split)


"""
d = max(cv_auc)

i = np.where(cv_auc == d)

i = i[0][0]
max_depth_value = float(max_depth[i])
print("Best max_depth is:-",max_depth_value)


max_depth = np.log(max_depth)

plt.plot(max_depth, train_auc, label='Train AUC')
plt.plot(max_depth, cv_auc, label='CV AUC')
plt.legend()
plt.xlabel("max_depth: hyperparameter")
plt.ylabel("AUC")
plt.title("Performance PLOT")
plt.show()
"""
```

```
Fitting 10 folds for each of 10 candidates, totalling 100 fits
```

```
[Parallel(n_jobs=1)]: Using backend SequentialBackend with 1 concurrent workers.
[Parallel(n_jobs=1)]: Done 100 out of 100 | elapsed: 22.5min finished
```

```
Best max_depth is:- 500
Best min_samples_split is:- 5
```

Out[62]:

```
'\nd = max(cv_auc)\n\ni = np.where(cv_auc == d)\n\ni = i[0][0]\nmax_depth_value =
float(max_depth[i])\nprint("Best max_depth is:-",max_depth_value)      \n    \n     \nmax_depth = np.l
og(max_depth)     \n     \nplt.plot(max_depth, train_auc, label=\'Train AUC\')\nplt.plot(max_depth, c
v_auc, label=\'CV AUC\')\nplt.legend()\nplt.xlabel("max_depth:
hyperparameter")\nplt.ylabel("AUC")\nplt.title("Performance PLOT")\nplt.show()\n'
```

In [63]:

```python
best_max_depth = clf.best_estimator_.max_depth
best_min_samples_split = clf.best_estimator_.min_samples_split
```

In [65]:

```python
import plotly.offline as offline
import plotly.graph_objs as go
offline.init_notebook_mode()
import numpy as np

trace1 = go.Scatter3d(x=max_depth,y=min_samples_split,z=train_auc1, name = 'train')
trace2 = go.Scatter3d(x=max_depth,y=min_samples_split,z=cv_auc1, name = 'Cross validation')
data = [trace1, trace2]

layout = go.Layout(scene = dict(
        xaxis = dict(title='n_estimators'),
        yaxis = dict(title='max_depth'),
        zaxis = dict(title='AUC'),))

fig = go.Figure(data=data, layout=layout)
offline.iplot(fig, filename="Bowt.png")
```

In [66]:

```
# LogisticRegression with best best "C" for l1 penalty of bow
model = DecisionTreeClassifier(max_depth = best_max_depth ,min_samples_split =
best_min_samples_split,class_weight='balanced')
model.fit(x_tr_final_counts_bigram,y_train)
#pred = model.predict_proba(x_ts_final_counts_bigram)
pred=model.predict(x_ts_final_counts_bigram)
    # evaluate CV AUC
auc_score_bowT_l1 = roc_auc_score(y_true=np.array(y_ts),
y_score=model.predict_proba(x_ts_final_counts_bigram)[:,1])*100
auc_score_bowT_lambda_l1 = best_max_depth
print('\nThe AUCScore of the DecisionTreeClassifier of best_max_depth = %f and min_samples_split =
%f is %f%%' % (best_max_depth,best_min_samples_split, auc_score_bowT_l1))
```

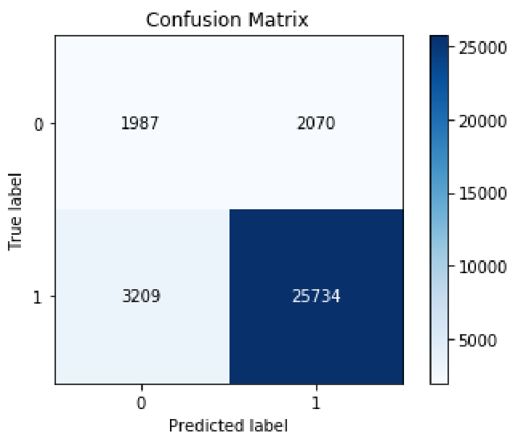
```
The AUCScore of the DecisionTreeClassifier of best_max_depth = 500.000000 and min_samples_split =
5.000000 is 69.079770%
```

In [67]:

```
import scikitplot.metrics as skplt
skplt.plot_confusion_matrix(y_ts ,pred)
```

Out[67]:

```
<matplotlib.axes._subplots.AxesSubplot at 0xef8028ea90>
```



False Positive rate --> when it is actually -ve, how often does it predicted +ve = fp/actual-ve = 2070/4057 = .51

In [146]:

```
# FPR for bowt_l1
bowt_FPR_l1 = .51
```

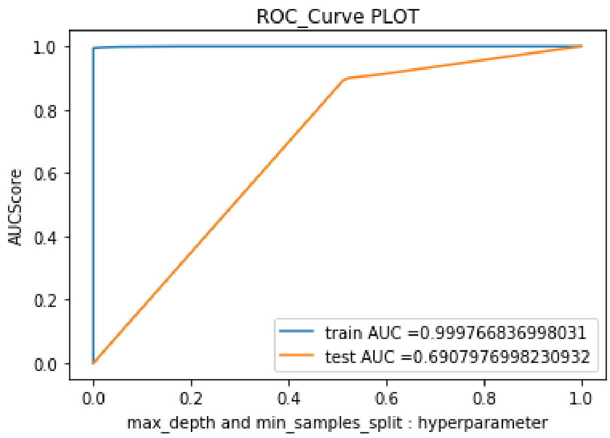
In [69]:

```
#classification report
from sklearn.metrics import classification_report
print(classification_report(y_ts, pred))
```

```
              precision    recall  f1-score   support

           0       0.38      0.49      0.43      4057
           1       0.93      0.89      0.91     28943

   micro avg       0.84      0.84      0.84     33000
   macro avg       0.65      0.69      0.67     33000
weighted avg       0.86      0.84      0.85     33000
```

In [70]:

```
from sklearn.metrics import roc_curve, auc

train_fpr, train_tpr, thresholds = roc_curve(y_train, model.predict_proba(x_tr_final_counts_bigram
)[:,1])
test_fpr, test_tpr, thresholds = roc_curve(y_ts, model.predict_proba(x_ts_final_counts_bigram)[:,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("max_depth and min_samples_split : hyperparameter")
plt.ylabel("AUCScore")
plt.title("ROC_Curve PLOT")
plt.show()
```



[5.1.1] Top 20 important features from SET 1

In [80]:

```
features = count_vect.get_feature_names()
#print("some sample features(unique words in the corpus)",features[0:10])
```

In [83]:

```
feat_log  = model.feature_importances_
feat_log
```

Out[83]:

```
array([0., 0., 0., ..., 0., 0., 0.])
```

In [85]:

```
feature_prob = pd.DataFrame([feat_log], columns = features)
feature_prob_tr = feature_prob.T
feature_prob_tr.shape
```

Out[85]:

```
(38798, 1)
```

In [90]:

```
print("Top 20 Features:-\n",feature_prob_tr[0].sort_values(ascending = False)[0:20])
```

```
Top 20 Features:-
 great            0.060248
love             0.037390
best             0.033525
disappoint       0.029303
delici           0.022029
good             0.018339
```

```
tast              0.014205
favorit           0.013439
excel             0.013097
perfect           0.012489
bad               0.011147
nice              0.009110
thought           0.008226
high recommend    0.008120
would             0.007799
money             0.007114
use               0.007012
wonder            0.006779
find              0.006579
product           0.006352
Name: 0, dtype: float64
```

[5.1.2] Graphviz visualization of Decision Tree on BOW, SET 1

In [144]:

```
# Please write all the code with proper documentation
# Importing libraries
from sklearn import tree
import pydotplus
from IPython.display import Image
from IPython.display import SVG
from graphviz import Source
from IPython.display import display
from sklearn.externals.six import StringIO

count_vect = CountVectorizer()
x_tr_final_counts_bigram = count_vect.fit_transform(x_train['CleanedText'].values)
names=count_vect.get_feature_names()
#names = names[0:50]
dt = DecisionTreeClassifier(class_weight= 'balanced',max_depth=3, min_samples_split=500)
dt.fit(x_tr_final_counts_bigram,y_train)

target = ['negative','positive']
# Create DOT data
dot_data = StringIO()
data = tree.export_graphviz(dt,out_file = "bowt_graphvizfile1.png",class_names=target,feature_names
=names,filled=True,rounded=True,special_characters=True)
```

[5.2] Applying Decision Trees on TFIDF, SET 2

In [96]:

```
#To show how Time Series Split splits the data
from sklearn.model_selection import TimeSeriesSplit
tscv1 = TimeSeriesSplit(n_splits=10)
for train, cv in tscv1.split(x_tr_final_counts_tfidf):
    print("%s %s" % (train, cv))
#    print(x_tr_final_counts_bigram[train].shape,x_tr_final_counts_bigram[cv].shape)
```

```
[    0     1     2 ... 6097 6098 6099] [ 6100  6101  6102 ... 12187 12188 12189]
[    0     1     2 ... 12187 12188 12189] [12190 12191 12192 ... 18277 18278 18279]
[    0     1     2 ... 18277 18278 18279] [18280 18281 18282 ... 24367 24368 24369]
[    0     1     2 ... 24367 24368 24369] [24370 24371 24372 ... 30457 30458 30459]
[    0     1     2 ... 30457 30458 30459] [30460 30461 30462 ... 36547 36548 36549]
[    0     1     2 ... 36547 36548 36549] [36550 36551 36552 ... 42637 42638 42639]
[    0     1     2 ... 42637 42638 42639] [42640 42641 42642 ... 48727 48728 48729]
[    0     1     2 ... 48727 48728 48729] [48730 48731 48732 ... 54817 54818 54819]
[    0     1     2 ... 54817 54818 54819] [54820 54821 54822 ... 60907 60908 60909]
[    0     1     2 ... 60907 60908 60909] [60910 60911 60912 ... 66997 66998 66999]
```

In [97]:

```
# Please write all the code with proper documentation
from sklearn.linear_model import LogisticRegression
from sklearn.metrics import f1_score
```

```
from sklearn.metrics import roc_auc_score
from sklearn.metrics import accuracy_score
from math import log
from sklearn.model_selection import RandomizedSearchCV
from sklearn.model_selection import GridSearchCV
from sklearn.tree import DecisionTreeClassifier


alpha_values = np.arange(7)
#C = np.array([0.0001,0.001,0.01,0.1,1,10,100,500,1000,10000])
max_depth = np.array([1, 5, 10, 50, 100, 500, 1000])
min_samples_split = np.array([5, 10, 100, 500])
cv_auc2 = np.empty(len(alpha_values))
train_auc2 = np.empty(len(alpha_values))


neigh = DecisionTreeClassifier()
#params we need to try on classifier
param_grid = {'max_depth':[1, 5, 10, 50, 100, 500, 1000],
              'min_samples_split':[5, 10, 100, 500] ,'class_weight':['balanced']}
tscv1 = TimeSeriesSplit(n_splits=10) #For time based splitting
clf = RandomizedSearchCV(neigh,param_grid,cv=tscv1,verbose=1)
clf.fit(x_tr_final_counts_tfidf,y_train)


train_auc2= clf.cv_results_['mean_train_score']
train_auc_std= clf.cv_results_['std_train_score']
cv_auc2 = clf.cv_results_['mean_test_score']
cv_auc_std= clf.cv_results_['std_test_score']


print("Best max_depth is:-",clf.best_estimator_.max_depth)
print("Best min_samples_split is:-",clf.best_estimator_.min_samples_split)


"""
d = max(cv_auc)

i = np.where(cv_auc == d)

i = i[0][0]
max_depth_value = float(max_depth[i])
print("Best max_depth is:-",max_depth_value)


max_depth = np.log(max_depth)

plt.plot(max_depth, train_auc, label='Train AUC')
plt.plot(max_depth, cv_auc, label='CV AUC')
plt.legend()
plt.xlabel("max_depth: hyperparameter")
plt.ylabel("AUC")
plt.title("Performance PLOT")
plt.show()
"""
```

```
Fitting 10 folds for each of 10 candidates, totalling 100 fits
```

```
[Parallel(n_jobs=1)]: Using backend SequentialBackend with 1 concurrent workers.
[Parallel(n_jobs=1)]: Done 100 out of 100 | elapsed: 24.0min finished
```

```
Best max_depth is:- 500
Best min_samples_split is:- 10
```

Out[97]:

```
'\nd = max(cv_auc)\n\ni = np.where(cv_auc == d)\n\ni = i[0][0]\nmax_depth_value =
float(max_depth[i])\nprint("Best max_depth is:-",max_depth_value)     \n    \n    \nmax_depth = np.l
og(max_depth)     \n     \nplt.plot(max_depth, train_auc, label=\'Train AUC\')\nplt.plot(max_depth, c
v_auc, label=\'CV AUC\')\nplt.legend()\nplt.xlabel("max_depth:
hyperparameter")\nplt.ylabel("AUC")\nplt.title("Performance PLOT")\nplt.show()\n'
```

In [98]:

```
import plotly.offline as offline
import plotly.graph_objs as go
offline.init_notebook_mode()
import numpy as np

trace1 = go.Scatter3d(x=max_depth,y=min_samples_split,z=train_auc2, name = 'train')
trace2 = go.Scatter3d(x=max_depth,y=min_samples_split,z=cv_auc2, name = 'Cross validation')
data = [trace1, trace2]

layout = go.Layout(scene = dict(
        xaxis = dict(title='n_estimators'),
        yaxis = dict(title='max_depth'),
        zaxis = dict(title='AUC'),))

fig = go.Figure(data=data, layout=layout)
offline.iplot(fig, filename='Tfidf.png')
```

In [99]:

```
best_max_depth1 = clf.best_estimator_.max_depth
best_min_samples_split1 = clf.best_estimator_.min_samples_split
```

In [100]:

```
# LogisticRegression with best best "C" for l1 penalty of bow
model = DecisionTreeClassifier(max_depth = best_max_depth1 ,min_samples_split =
best_min_samples_split1,class_weight='balanced')
model.fit(x_tr_final_counts_tfidf,y_train)
#pred = model.predict_proba(x_ts_final_counts_bigram)
pred=model.predict(x_ts_final_counts_tfidf)
    # evaluate CV AUC
auc_score_tfidf_l1 = roc_auc_score(y_true=np.array(y_ts),
y_score=model.predict_proba(x_ts_final_counts_tfidf)[:,1])*100
auc_score_tfidf_lambda_l1 = best_max_depth1
#print('\nThe AUCScore of the DecisionTreeClassifier of best_max_depth = %f is %f%%' %
(best_max_depth1, auc_score_tfidf_l1))
print('\nThe AUCScore of the DecisionTreeClassifier of best_max_depth = %f and min_samples_split =
%f is %f%%' % (best_max_depth1,best_min_samples_split1, auc_score_tfidf_l1))
```

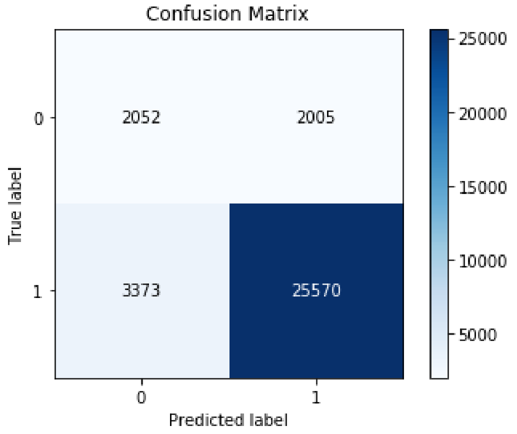
```
The AUCScore of the DecisionTreeClassifier of best_max_depth = 500.000000 and min_samples_split =
10.000000 is 70.082279%
```

In [101]:

```python
import scikitplot.metrics as skplt
skplt.plot_confusion_matrix(y_ts ,pred)
```

Out[101]:

```
<matplotlib.axes._subplots.AxesSubplot at 0xef807d3f98>
```



False Positive rate --> when it is actually -ve, how often does it predicted +ve = fp/actual-ve = 2005/4057 = .49

In [147]:

```python
# FPR for tfidf_l1
tfidf_FPR_l1 = .49
```

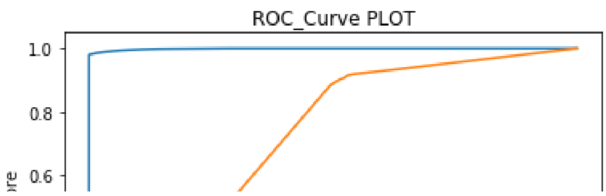
In [103]:

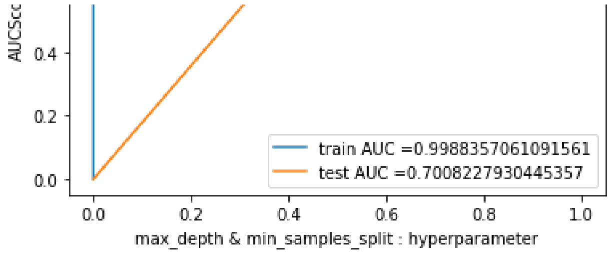
```python
print(classification_report(y_ts, pred))
```

```
              precision    recall  f1-score   support

           0       0.38      0.51      0.43      4057
           1       0.93      0.88      0.90     28943

   micro avg       0.84      0.84      0.84     33000
   macro avg       0.65      0.69      0.67     33000
weighted avg       0.86      0.84      0.85     33000
```

In [104]:

```python
from sklearn.metrics import roc_curve, auc

train_fpr, train_tpr, thresholds = roc_curve(y_train, model.predict_proba(x_tr_final_counts_tfidf)
[:,1])
test_fpr, test_tpr, thresholds = roc_curve(y_ts, model.predict_proba(x_ts_final_counts_tfidf)[:,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("max_depth & min_samples_split : hyperparameter")
plt.ylabel("AUCScore")
plt.title("ROC_Curve PLOT")
plt.show()
```





[5.2.1] Top 20 important features from SET 2

In [105]:

```
features = tf_idf_vect.get_feature_names()
#print("some sample features(unique words in the corpus)",features[0:10])
```

In [106]:

```
feat_log  = model.feature_importances_
feat_log
```

Out[106]:

```
array([0., 0., 0., ..., 0., 0., 0.])
```

In [107]:

```
feature_prob = pd.DataFrame([feat_log], columns = features)
feature_prob_tr = feature_prob.T
feature_prob_tr.shape
```

Out[107]:

```
(38798, 1)
```

In [108]:

```
print("Top 20 Features:-\n",feature_prob_tr[0].sort_values(ascending = False)[0:20])
```

```
Top 20 Features:-
 great              0.062336
love               0.036186
best               0.035834
disappoint         0.030264
delici             0.023692
good               0.016276
favorit            0.013635
excel              0.013538
tast               0.013195
perfect            0.011906
bad                0.009924
thought            0.009836
would              0.009262
nice               0.009032
high recommend     0.008801
money              0.008479
easi               0.007217
wonder             0.006324
find               0.005831
howev              0.005830
Name: 0, dtype: float64
```

[5.2.2] Graphviz visualization of Decision Tree on TFIDF, SET 2

In [145]:

```
# Please write all the code with proper documentation
```

```
# Importing libraries
from sklearn import tree
import pydotplus
from IPython.display import Image
from IPython.display import SVG
from graphviz import Source
from IPython.display import display
from sklearn.externals.six import StringIO

tf_idf_vect = TfidfVectorizer()
x_tr_final_counts_tfidf = tf_idf_vect.fit_transform(x_train['CleanedText'].values)
names=tf_idf_vect.get_feature_names()
#names = names[0:50]
dt = DecisionTreeClassifier(class_weight= 'balanced',max_depth=3, min_samples_split=500)
dt.fit(x_tr_final_counts_tfidf,y_train)

target = ['negative','positive']
# Create DOT data
dot_data = StringIO()
data = tree.export_graphviz(dt,out_file =
"tfidf_graphvizfile1.png",class_names=target,feature_names=names,filled=True,rounded=True,special_c
haracters=True)
```

[5.3] Applying Decision Trees on AVG W2V, SET 3

In [110]:

```
#To show how Time Series Split splits the data
from sklearn.model_selection import TimeSeriesSplit
tscv2 = TimeSeriesSplit(n_splits=10)
for train, cv in tscv2.split(train_avgw2v):
    print("%s %s" % (train, cv))
#    print(x_tr_final_counts_bigram[train].shape,x_tr_final_counts_bigram[cv].shape)
```

```
[    0    1    2 ... 6097 6098 6099] [ 6100  6101  6102 ... 12187 12188 12189]
[    0     1     2 ... 12187 12188 12189] [12190 12191 12192 ... 18277 18278 18279]
[    0     1     2 ... 18277 18278 18279] [18280 18281 18282 ... 24367 24368 24369]
[    0     1     2 ... 24367 24368 24369] [24370 24371 24372 ... 30457 30458 30459]
[    0     1     2 ... 30457 30458 30459] [30460 30461 30462 ... 36547 36548 36549]
[    0     1     2 ... 36547 36548 36549] [36550 36551 36552 ... 42637 42638 42639]
[    0     1     2 ... 42637 42638 42639] [42640 42641 42642 ... 48727 48728 48729]
[    0     1     2 ... 48727 48728 48729] [48730 48731 48732 ... 54817 54818 54819]
[    0     1     2 ... 54817 54818 54819] [54820 54821 54822 ... 60907 60908 60909]
[    0     1     2 ... 60907 60908 60909] [60910 60911 60912 ... 66997 66998 66999]
```

In [111]:

```
# Please write all the code with proper documentation
from sklearn.linear_model import LogisticRegression
from sklearn.metrics import f1_score
from sklearn.metrics import roc_auc_score
from sklearn.metrics import accuracy_score
from math import log
from sklearn.model_selection import RandomizedSearchCV
from sklearn.model_selection import GridSearchCV
from sklearn.tree import DecisionTreeClassifier


alpha_values = np.arange(7)
#C = np.array([0.0001,0.001,0.01,0.1,1,10,100,500,1000,10000])
max_depth = np.array([1, 5, 10, 50, 100, 500, 1000])
min_samples_split = np.array([5, 10, 100, 500])
cv_auc2 = np.empty(len(alpha_values))
train_auc2 = np.empty(len(alpha_values))


neigh = DecisionTreeClassifier()
#params we need to try on classifier
param_grid = {'max_depth':[1, 5, 10, 50, 100, 500, 1000],
             'min_samples_split':[5, 10, 100, 500] ,'class_weight':['balanced']}
tscv2 = TimeSeriesSplit(n_splits=10) #For time based splitting
clf = RandomizedSearchCV(neigh,param_grid,cv=tscv2,verbose=1)
clf.fit(train_avgw2v,y_train)
```

```
train_auc3= clf.cv_results_['mean_train_score']
train_auc_std= clf.cv_results_['std_train_score']
cv_auc3 = clf.cv_results_['mean_test_score']
cv_auc_std= clf.cv_results_['std_test_score']


print("Best max_depth is:-",clf.best_estimator_.max_depth)
print("Best min_samples_split is:-",clf.best_estimator_.min_samples_split)


"""
d = max(cv_auc)

i = np.where(cv_auc == d)

i = i[0][0]
max_depth_value = float(max_depth[i])
print("Best max_depth is:-",max_depth_value)


max_depth = np.log(max_depth)

plt.plot(max_depth, train_auc, label='Train AUC')
plt.plot(max_depth, cv_auc, label='CV AUC')
plt.legend()
plt.xlabel("max_depth: hyperparameter")
plt.ylabel("AUC")
plt.title("Performance PLOT")
plt.show()
"""
```

```
Fitting 10 folds for each of 10 candidates, totalling 100 fits
```

```
[Parallel(n_jobs=1)]: Using backend SequentialBackend with 1 concurrent workers.
[Parallel(n_jobs=1)]: Done 100 out of 100 | elapsed:  4.5min finished
```

```
Best max_depth is:- 50
Best min_samples_split is:- 5
```

Out[111]:

```
'\nd = max(cv_auc)\n\ni = np.where(cv_auc == d)\n\ni = i[0][0]\nmax_depth_value =
float(max_depth[i])\nprint("Best max_depth is:-",max_depth_value)      \n    \n     \nmax_depth = np.l
og(max_depth)     \n     \nplt.plot(max_depth, train_auc, label=\'Train AUC\')\nplt.plot(max_depth, c
v_auc, label=\'CV AUC\')\nplt.legend()\nplt.xlabel("max_depth:
hyperparameter")\nplt.ylabel("AUC")\nplt.title("Performance PLOT")\nplt.show()\n'
```

In [112]:

```
import plotly.offline as offline
import plotly.graph_objs as go
offline.init_notebook_mode()
import numpy as np

trace1 = go.Scatter3d(x=max_depth,y=min_samples_split,z=train_auc3, name = 'train')
trace2 = go.Scatter3d(x=max_depth,y=min_samples_split,z=cv_auc3, name = 'Cross validation')
data = [trace1, trace2]

layout = go.Layout(scene = dict(
        xaxis = dict(title='n_estimators'),
        yaxis = dict(title='max_depth'),
        zaxis = dict(title='AUC'),))

fig = go.Figure(data=data, layout=layout)
offline.iplot(fig, filename='avgw2v.png')
```

In [113]:

```
best_max_depth2 = clf.best_estimator_.max_depth
best_min_samples_split2 = clf.best_estimator_.min_samples_split
```

In [114]:

```
# LogisticRegression with best best "C" for l1 penalty of bow
model = DecisionTreeClassifier(max_depth = best_max_depth2 ,min_samples_split =
best_min_samples_split2,class_weight='balanced')
model.fit(train_avgw2v,y_train)
#pred = model.predict_proba(x_ts_final_counts_bigram)
pred=model.predict(test_avgw2v)
    # evaluate CV AUC
auc_score_avgw2v = roc_auc_score(y_true=np.array(y_ts), y_score=model.predict_proba(test_avgw2v)[:,
1])*100
auc_score_avgw2v_lambda_l1 = best_max_depth2
#print('\nThe AUCScore of the DecisionTreeClassifier of best_max_depth = %f is %f%%' %
(best_max_depth2, auc_score_avgw2v))
print('\nThe AUCScore of the DecisionTreeClassifier of best_max_depth = %f and min_samples_split =
%f is %f%%' % (best_max_depth2,best_min_samples_split2, auc_score_avgw2v))
```

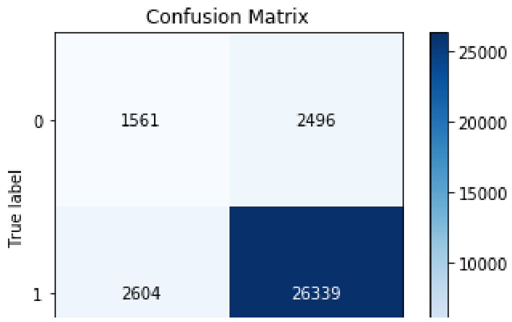
```
The AUCScore of the DecisionTreeClassifier of best_max_depth = 50.000000 and min_samples_split = 5
.000000 is 64.786225%
```

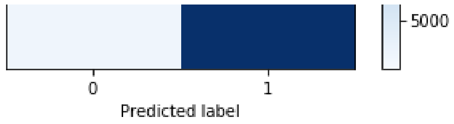
In [115]:

```
import scikitplot.metrics as skplt
skplt.plot_confusion_matrix(y_ts ,pred)
```

Out[115]:

```
<matplotlib.axes._subplots.AxesSubplot at 0xef80327588>
```





False Positive rate --> when it is actually -ve, how often does it predicted +ve = fp/actual-ve = 2496/4057 = .61

In [148]:

```
# FPR for avgw2v_l1
avgw2v_FPR_l1 = .61
```

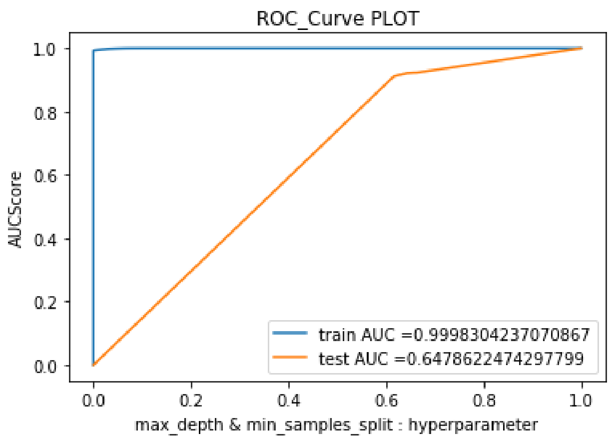
In [117]:

```
print(classification_report(y_ts, pred))
```

```
              precision    recall  f1-score   support

           0       0.37      0.38      0.38      4057
           1       0.91      0.91      0.91     28943

   micro avg       0.85      0.85      0.85     33000
   macro avg       0.64      0.65      0.65     33000
weighted avg       0.85      0.85      0.85     33000
```

In [118]:

```
from sklearn.metrics import roc_curve, auc

train_fpr, train_tpr, thresholds = roc_curve(y_train, model.predict_proba(train_avgw2v)[:,1])
test_fpr, test_tpr, thresholds = roc_curve(y_ts, model.predict_proba(test_avgw2v)[:,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("max_depth & min_samples_split : hyperparameter")
plt.ylabel("AUCScore")
plt.title("ROC_Curve PLOT")
plt.show()
```



[5.4] Applying Decision Trees on TFIDF W2V, SET 4

In [119]:

```
#To show how Time Series Split splits the data
from sklearn.model_selection import TimeSeriesSplit
tscv3 = TimeSeriesSplit(n_splits=10)
for train, cv in tscv3.split(tfidf_sent_vectors):
    print("%s %s" % (train, cv))
#    print(x_tr_final_counts_bigram[train].shape,x_tr_final_counts_bigram[cv].shape)
```

```
[    0     1     2 ...  6097  6098  6099] [ 6100  6101  6102 ... 12187 12188 12189]
[    0     1     2 ... 12187 12188 12189] [12190 12191 12192 ... 18277 18278 18279]
[    0     1     2 ... 18277 18278 18279] [18280 18281 18282 ... 24367 24368 24369]
[    0     1     2 ... 24367 24368 24369] [24370 24371 24372 ... 30457 30458 30459]
[    0     1     2 ... 30457 30458 30459] [30460 30461 30462 ... 36547 36548 36549]
[    0     1     2 ... 36547 36548 36549] [36550 36551 36552 ... 42637 42638 42639]
[    0     1     2 ... 42637 42638 42639] [42640 42641 42642 ... 48727 48728 48729]
[    0     1     2 ... 48727 48728 48729] [48730 48731 48732 ... 54817 54818 54819]
[    0     1     2 ... 54817 54818 54819] [54820 54821 54822 ... 60907 60908 60909]
[    0     1     2 ... 60907 60908 60909] [60910 60911 60912 ... 66997 66998 66999]
```

In [120]:

```
# Please write all the code with proper documentation
from sklearn.linear_model import LogisticRegression
from sklearn.metrics import f1_score
from sklearn.metrics import roc_auc_score
from sklearn.metrics import accuracy_score
from math import log
from sklearn.model_selection import RandomizedSearchCV
from sklearn.model_selection import GridSearchCV
from sklearn.tree import DecisionTreeClassifier


alpha_values = np.arange(7)
#C = np.array([0.0001,0.001,0.01,0.1,1,10,100,500,1000,10000])
max_depth = np.array([1, 5, 10, 50, 100, 500, 1000])
min_samples_split = np.array([5, 10, 100, 500])
cv_auc3 = np.empty(len(alpha_values))
train_auc3 = np.empty(len(alpha_values))


neigh = DecisionTreeClassifier()
#params we need to try on classifier
param_grid = {'max_depth':[1, 5, 10, 50, 100, 500, 1000],
             'min_samples_split':[5, 10, 100, 500] ,'class_weight':['balanced']}
tscv3 = TimeSeriesSplit(n_splits=10) #For time based splitting
clf = RandomizedSearchCV(neigh,param_grid,cv=tscv3,verbose=1)
clf.fit(tfidf_sent_vectors,y_train)


train_auc3= clf.cv_results_['mean_train_score']
train_auc_std= clf.cv_results_['std_train_score']
cv_auc3 = clf.cv_results_['mean_test_score']
cv_auc_std= clf.cv_results_['std_test_score']


print("Best max_depth is:-",clf.best_estimator_.max_depth)
print("Best min_samples_split is:-",clf.best_estimator_.min_samples_split)


"""
d = max(cv_auc)

i = np.where(cv_auc == d)

i = i[0][0]
max_depth_value = float(max_depth[i])
print("Best max_depth is:-",max_depth_value)


max_depth = np.log(max_depth)

plt.plot(max_depth, train_auc, label='Train AUC')
plt.plot(max_depth, cv_auc, label='CV AUC')
plt.legend()
plt.xlabel("max_depth: hyperparameter")
plt.ylabel("AUC")
plt.title("Performance PLOT")
plt.show()
"""
```

```
Fitting 10 folds for each of 10 candidates, totalling 100 fits
```

```
[Parallel(n_jobs=1)]: Using backend SequentialBackend with 1 concurrent workers.
[Parallel(n_jobs=1)]: Done 100 out of 100 | elapsed:  4.7min finished
```

```
Best max_depth is:- 100
Best min_samples_split is:- 10
```

Out[120]:

```
'\nd = max(cv_auc)\n\ni = np.where(cv_auc == d)\n\ni = i[0][0]\nmax_depth_value =
float(max_depth[i])\nprint("Best max_depth is:-",max_depth_value)      \n    \n     \nmax_depth = np.l
og(max_depth)     \n     \nplt.plot(max_depth, train_auc, label=\'Train AUC\')\nplt.plot(max_depth, c
v_auc, label=\'CV AUC\')\nplt.legend()\nplt.xlabel("max_depth:
hyperparameter")\nplt.ylabel("AUC")\nplt.title("Performance PLOT")\nplt.show()\n'
```

In [121]:

```
import plotly.offline as offline
import plotly.graph_objs as go
offline.init_notebook_mode()
import numpy as np

trace1 = go.Scatter3d(x=max_depth,y=min_samples_split,z=train_auc3, name = 'train')
trace2 = go.Scatter3d(x=max_depth,y=min_samples_split,z=cv_auc3, name = 'Cross validation')
data = [trace1, trace2]

layout = go.Layout(scene = dict(
        xaxis = dict(title='n_estimators'),
        yaxis = dict(title='max_depth'),
        zaxis = dict(title='AUC'),))

fig = go.Figure(data=data, layout=layout)
offline.iplot(fig, filename='TFidf-avgw2v.png')
```

In [122]:

```
best_max_depth3 = clf.best_estimator_.max_depth
best_min_samples_split3 = clf.best_estimator_.min_samples_split
```

In [123]:

```
# LogisticRegression with best best "C" for l1 penalty of bow
```

```
# LogisticRegression with best best 'C' for l1 penalty of bow
model = DecisionTreeClassifier(max_depth = best_max_depth3 ,min_samples_split =
best_min_samples_split3,class_weight='balanced')
model.fit(tfidf_sent_vectors,y_train)
#pred = model.predict_proba(x_ts_final_counts_bigram)
pred=model.predict(tfidf_sent_vectors_ts)
    # evaluate CV AUC
auc_score_sent_vectors = roc_auc_score(y_true=np.array(y_ts),
y_score=model.predict_proba(tfidf_sent_vectors_ts)[:,1])*100
auc_score_sent_vectors_lambda_l1 = best_max_depth3
#print('\nThe AUCScore of the DecisionTreeClassifier of best_max_depth = %f is %f%%' %
(best_max_depth3, auc_score_sent_vectors))
print('\nThe AUCScore of the DecisionTreeClassifier of best_max_depth = %f and min_samples_split =
%f is %f%%' % (best_max_depth3,best_min_samples_split3, auc_score_sent_vectors))
```

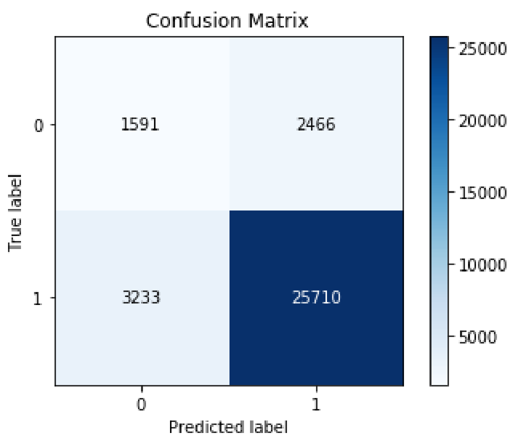
```
The AUCScore of the DecisionTreeClassifier of best_max_depth = 100.000000 and min_samples_split =
10.000000 is 64.534046%
```

In [124]:

```
import scikitplot.metrics as skplt
skplt.plot_confusion_matrix(y_ts ,pred)
```

Out[124]:

```
<matplotlib.axes._subplots.AxesSubplot at 0xef80327f60>
```



False Positive rate --> when it is actually -ve, how often does it predicted +ve = fp/actual-ve = 2466/4057 = .60

In [149]:

```
# FPR for sent_vectors_l1
sent_vectors_FPR_l1 =.60
```

In [126]:

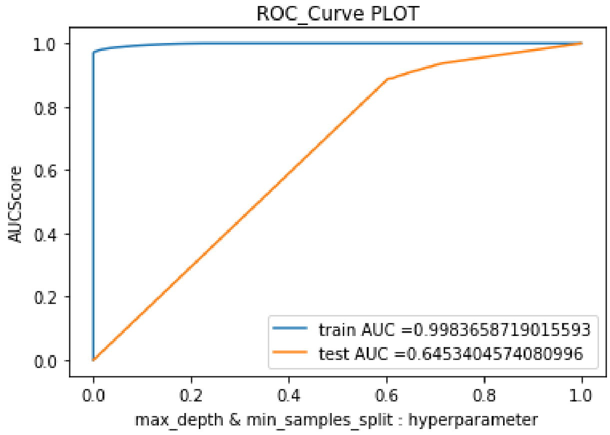
```
print(classification_report(y_ts, pred))
```

```
              precision    recall  f1-score   support

           0       0.33      0.39      0.36      4057
           1       0.91      0.89      0.90     28943

   micro avg       0.83      0.83      0.83     33000
   macro avg       0.62      0.64      0.63     33000
weighted avg       0.84      0.83      0.83     33000
```

In [127]:

```
from sklearn.metrics import roc_curve, auc

train_fpr, train_tpr, thresholds = roc_curve(y_train, model.predict_proba(tfidf_sent_vectors)
[:,1])
```

```
[:,1])
test_fpr, test_tpr, thresholds = roc_curve(y_ts, model.predict_proba(tfidf_sent_vectors_ts)[:,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("max_depth & min_samples_split : hyperparameter")
plt.ylabel("AUCScore")
plt.title("ROC_Curve PLOT")
plt.show()
```



[6] Conclusions

In [152]:

```
# Please compare all your models using Prettytable library
from prettytable import PrettyTable
x = PrettyTable()
x.field_names = ["Vectorizer","roc_auc_score","FPR"]
x.add_row(["BOW",auc_score_bowT_l1,bowt_FPR_l1])
x.add_row(["TFIDF",auc_score_tfidf_l1,tfidf_FPR_l1])
x.add_row(["AVG -W2V",auc_score_avgw2v,avgw2v_FPR_l1])
x.add_row(["TFIDF W2V",auc_score_sent_vectors,sent_vectors_FPR_l1])
print(x)
```

```
+------------+-------------------+-------+
| Vectorizer |   roc_auc_score   |  FPR  |
+------------+-------------------+-------+
|    BOW     | 69.07976998230933 | 0.51  |
|   TFIDF    | 70.08227930445356 | 0.49  |
|  AVG -W2V  | 64.78622474297799 | 0.61  |
| TFIDF W2V  | 64.53404574080996 |  0.6  |
+------------+-------------------+-------+
```

as per the table, we can consider TFIDF vectorizer because it has less false positive rate and more roc_auc_score

In [ ]: