

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 - unique 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 considered as a positive review. A rating of 1 or 2 can be considered as negative one. A review of rating 3 is considered neutral 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 will 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
```

```

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 [3]: # 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 """, 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[3]:

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

	Id	ProductId	UserId	ProfileName	HelpfulnessNumerator	HelpfulnessDenominator
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	COUNT(*)
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	COUNT
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	COUNT
80638	AZY10LLTJ71NX	B006P7E5ZI	undertheshrine "undertheshrine"	1334707200	5	I was recommended to try green tea extract to ...	1

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 delete 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
sample_data = filtered_data.sample(n=50000,random_state=0)
sorted_data=sample_data.sort_values('ProductId', axis=0, ascending=True
, inplace=False, kind='quicksort', na_position='last')
```

```
In [6]: #Deduplication of entries
final=sorted_data.drop_duplicates(subset={"UserId","ProfileName","Time"
,"Text"}, keep='first', inplace=False)
final.shape
```

```
Out[6]: (46100, 10)
```

```
In [7]: #Checking to see how much % of data still remains
(final['Id'].size*1.0)/(filtered_data['Id'].size*1.0)*100
```

```
Out[7]: 8.767358799879805
```

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 calculations

```
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()

(46100, 10)
```

```
Out[0]: 1    38959
0     7141
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 observed 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-BATT-REEL/dp/B00004RRDY><

br /><http://www.amazon.com/victor-flt-flagnet-bait-kettle/up/B00004RBD1>
br />
The Victor M380 and M502 traps are unreal, of course -- total fly genocide. Pretty stinky, but only right nearby.

=====

I recently tried this flavor/brand and was surprised at how delicious these chips are. The best thing was that there were a lot of "brown" chips 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 flavor because they do not seem to be as salty, and the onion flavor is better. 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 they were ordering; the other wants crispy cookies. Hey, I'm sorry; but these reviews do nobody any good beyond reminding us to look before ordering.

These are chocolate-oatmeal cookies. If you don't like that combination, don't order this type of cookie. I find the combo quite nice, really. The oatmeal sort of "calms" the rich chocolate flavor and gives the cookie sort of a coconut-type consistency. Now let's also remember 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 cookie dough; however, I don't see where these taste like raw cookie dough. Both are soft, however, so is this the confusion? And, yes, they stick 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 suggest 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 try. I'm here to place my second order.

=====

I love to order my coffee on amazon. easy and shows up quickly.
This k cup is great coffee. dcaf is very good as well

=====

In [0]: `# 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)

```

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

=====

I recently tried this flavor/brand and was surprised at how delicious these 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 flavor because they do not seem to be as salty, and the onion flavor is better. 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 they were ordering; the other wants crispy cookies. Hey, I'm sorry; but these reviews do nobody any good beyond reminding us to look before ordering. These are chocolate-oatmeal cookies. If you don't like that combination, don't order this type of cookie. I find the combo quite nice, really. The oatmeal sort of "calms" the rich chocolate flavor and gives the cookie sort of a coconut-type consistency. Now let's also remember 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 cookie dough; however, I don't see where these taste like raw cookie dough. Both are soft, however, so is this the confusion? And, yes, they stick 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 suggest Nabisco's Ginger Snaps. If you want a cookie that's soft, chewy and tastes like a combination of chocolate 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. This k cup is great coffee. dcaf is very good as well

```
In [0]: # https://stackoverflow.com/a/47091490/4084039
import re
from bs4 import BeautifulSoup

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

Wow. So far, two two-star reviews. One obviously had no idea what they were ordering; the other wants crispy cookies. Hey, I am sorry; but these reviews do nobody any good beyond reminding us to look before ordering. These are chocolate-oatmeal cookies. If you do not like that combination, 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 cookie sort of a coconut-type consistency. Now let it also remember that tastes differ; so, I have 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 cookie dough; however, I do not see where these taste like raw cookie dough. Both are soft, however, so is this the confusion? And, yes, they stick together. Soft cookies tend to do that. They are not 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 suggest Nabisco's Ginger Snaps. If you want a cookie that is soft, chewy and tastes like a combination of chocolate and oatmeal, give these a try. I am here to place my second order.

=====

```
In [0]: #remove words with numbers python: https://stackoverflow.com/a/18082370/4084039
```

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

Why is this \$[...] when the same product is available for \$[...] here?
 />
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 were ordering the other wants crispy cookies Hey I am sorry but these reviews do nobody any good beyond reminding us to look before ordering
br
r These are chocolate oatmeal cookies If you do not like that combination 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 cookie sort of a coconut type consistency Now let is also remember that tastes differ so I have given my opinion
br
r 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 cookie dough however I do not see where these taste like raw cookie dough Both are soft however so is this the confusion And yes they stick together Soft cookies tend to do that They are not individually wrapped which would add to the cost Oh yeah chocolate chip cookies tend to be somewhat sweet
br
r So if you want something hard and crisp I suggest Nabisco is Ginger Snaps If you want a cookie that is soft chewy and tastes like a combination of chocolate 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', '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', "you're", "you've",\
```



```

        "you'll", "you'd", 'your', 'yours', 'yourself', 'yourselfe
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 [10]: # Combining all the above students
from tqdm import tqdm
preprocessed_reviews = []
# tqdm is for printing the status bar
for sentence in tqdm(final['Text'].values):
    sentence = re.sub(r"http\S+", "", sentence)
    sentence = BeautifulSoup(sentence, 'lxml').get_text()
    sentence = decontracted(sentence)
    sentence = re.sub("\S*d\S*", "", sentence).strip()
    sentence = re.sub('[^A-Za-z]+', ' ', sentence)
    # https://gist.github.com/sebleier/554280

```

```
sentence = ' '.join(e.lower() for e in sentence.split() if e.lower()
() not in stopwords)
preprocessed_reviews.append(sentence.strip())
```

```
100%|██████████| 46100/46100 [00:21<00:00, 2128.77it/s]
```

```
In [11]: preprocessed_reviews[1500]
```

```
Out[11]: 'great coffee received gift folks traveled costa rica bought plantation
found ordered several times given gifts others wonderful coffee prompt
service company would highly recommend'
```

[3.2] Preprocessing Review Summary

```
In [0]: ## Similarly 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', 'aahhs', '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-grams
# count_vect = CountVectorizer(ngram_range=(1,2))
# please do read the CountVectorizer documentation http://scikit-learn.org/stable/modules/generated/sklearn.feature\_extraction.text.CountVectorizer.html

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

[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.get_feature_names()[0:10])
```

```
print('='*50)

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

```
some sample features(unique words in the corpus) ['ability', 'able', '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_sentence=[]
for sentence in preprocessed_reviews:
    list_of_sentence.append(sentence.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 variable 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 occurred at least 5 times
    w2v_model=Word2Vec(list_of_sentence,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 google's word2vec file, keep want_to_train_w2v = True, to train your own w2v ")

[('snack', 0.9951335191726685), ('calorie', 0.9946465492248535), ('wonderful', 0.9946032166481018), ('excellent', 0.9944332838058472), ('especially', 0.9941144585609436), ('baked', 0.9940600395202637), ('salted', 0.994047224521637), ('alternative', 0.9937226176261902), ('tasty', 0.9936816692352295), ('healthy', 0.9936649799346924)]

=====
[('varieties', 0.9994194507598877), ('become', 0.9992934465408325), ('popcorn', 0.9992750883102417), ('de', 0.9992610216140747), ('miss', 0.9992451071739197), ('melitta', 0.999218761920929), ('choice', 0.9992102384567261), ('american', 0.9991837739944458), ('beef', 0.9991780519485474), ('finish', 0.9991567134857178)]

```

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

number of words that occurred 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_sentence): # 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
```

```
100%|███████████████████████████████████████████████████████████████████████████  
██████████ | 4986/4986 [00:03<00:00, 1330.47it/s]
```

[4.4.1.2] TFIDF weighted W2v

```
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 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_sentence): # 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 corpus
# sent.count(word) = tf value of word in this review
```

[illegible]

- 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 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

```
In [0]: final["Clean_text"] = preprocessed_reviews
```

```
In [0]: final = final.sort_values('Time', axis=0, ascending=True, inplace=False,
    , kind='quicksort', na_position='last')
    #split data in train, test and cv before using it to avoid data leakage
    from sklearn.model_selection import train_test_split

    X = final['Clean_text']
    y = final['Score']

    X_train,X_test,y_train_,y_test_ = train_test_split(X,y,test_size=.5,random_state=0,shuffle=False)
    X_cv,X_test,y_cv_,y_test_ = train_test_split(X_test,y_test_,test_size=.5,random_state=0,shuffle=False)
```

```
In [0]: from sklearn.tree import DecisionTreeRegressor, DecisionTreeClassifier
    from sklearn.model_selection import GridSearchCV
    from sklearn.metrics import roc_auc_score
```

```

from sklearn.metrics import confusion_matrix
import sklearn
from tqdm import tqdm
import matplotlib.pyplot as plt
import numpy as np
#function to find an optimal value of k,AUC,ROC,confusion matrix
def best_para(x_train,x_cv,x_test,y_train,y_cv,y_test):
    depth = [1, 5, 10, 50, 100, 500, 1000]
    min_samples_split = [5, 10, 100, 500]

    param_grid = {'max_depth': [1, 5, 10, 50, 100, 500, 1000],
                  'min_samples_split': [5, 10, 100, 500],
                  'class_weight': ['balanced']}
    grid_search = GridSearchCV(DecisionTreeClassifier(),param_grid,
scoring = 'roc_auc',n_jobs=-1,return_train_score = True)
    grid_search.fit(x_train,y_train)

    auc_cv = grid_search.cv_results_['mean_test_score'].reshape(4,7
)
    auc = grid_search.cv_results_['mean_train_score'].reshape(4,7)
    print('best depth = ',grid_search.best_estimator_.max_depth)
    print('best min_samples_split = ',grid_search.best_estimator_.
min_samples_split)

    # matrix for auc
    confusion_matr = pd.DataFrame(auc,index=min_samples_split,colum
ns=depth)
    sns.heatmap(confusion_matr,annot=True,cmap='viridis')
    plt.title("AUC matrix for train")
    plt.xlabel("depth")
    plt.ylabel("min_samples_split")
    plt.show()

    confusion_matr_cv = pd.DataFrame(auc_cv,index=min_samples_split
,columns=depth)
    sns.heatmap(confusion_matr_cv,annot=True,cmap='viridis')
    plt.title("AUC matrix for cv")

```

```

plt.xlabel("depth")
plt.ylabel("min_samples_split")
plt.show()

def test(x_train,x_cv,x_test,y_train,y_cv,y_test,depth,min_samples_split):
    clf = DecisionTreeClassifier(max_depth = depth, min_samples_split = min_samples_split, class_weight = 'balanced')
    clf.fit(x_train,y_train)
    prob_test = clf.predict_proba(x_test)
    prob_test = prob_test[:,1]

    prob_train = clf.predict_proba(x_train)
    prob_train = prob_train[:,1]
    print("AUC Score: {}".format(roc_auc_score(y_test,prob_test)))

    #ROC curve
    fpr_tr, tpr_tr, thres_tr = roc_curve(y_train, prob_train)
    fpr, tpr, thres = roc_curve(y_test, prob_test)
    plt.plot([0,0],[1,1],linestyle='--')
    plt.plot(fpr, tpr, 'r', marker='.', label='test')
    plt.plot(fpr_tr, tpr_tr, 'b', marker='.', label='train')
    plt.legend(loc='upper right')
    plt.title("ROC curve")
    plt.show()

    #confusion matrix for train and test
    print("Confusion matrix for train data")
    predict_tr = clf.predict(x_train)
    confu_metrix_(y_train, predict_tr)

    print("Confusion matrix for test data")
    predict_te = clf.predict(x_test)
    confu_metrix_(y_test, predict_te)

def confu_metrix_(y, predict):

```

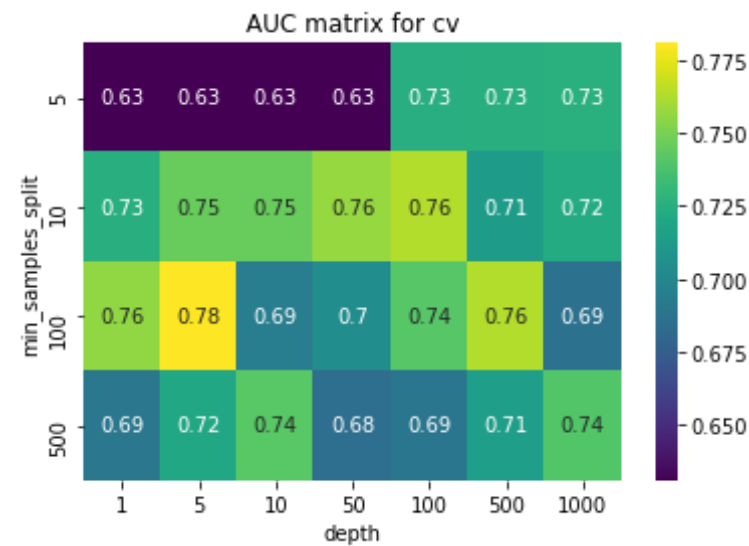
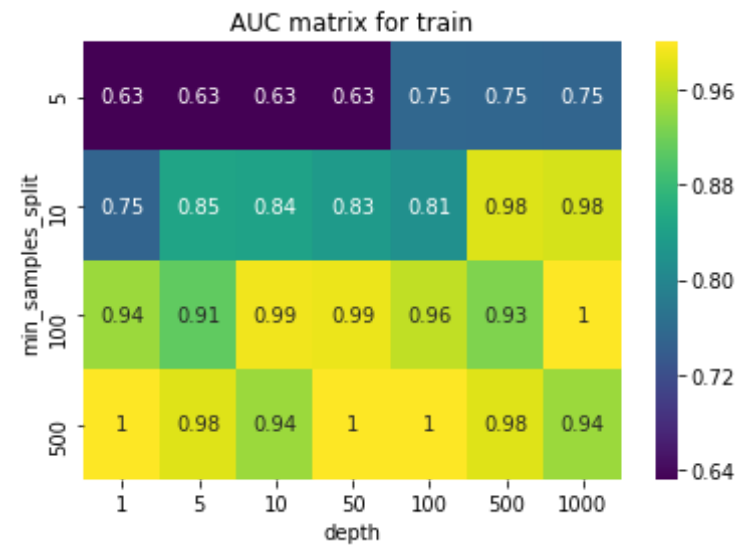
```
confu_metrix = confusion_matrix(y,predict)
confu_df = pd.DataFrame(confu_metrix,index=["-ve","+ve"],columns=
"-ve","+ve"])
sns.heatmap(confu_df,annot=True,fmt='d',cmap='viridis')
plt.title("Confusion matrix")
plt.xlabel("predicted label")
plt.ylabel("True label")
plt.show()
```

[5.1] Applying Decision Trees on BOW, SET 1

```
In [0]: # Please write all the code with proper documentation
import pickle
bow_cv,bow_test,bow_train = pickle.load(open("bow.pkl",'rb'))
y_cv,y_test,y_train = pickle.load(open("label.pkl",'rb'))
```

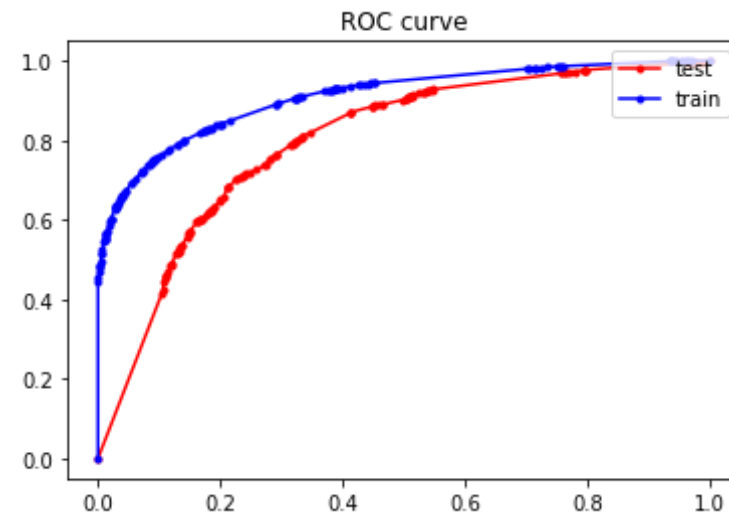
```
In [49]: best_para(bow_train,bow_cv,bow_test,y_train,y_cv,y_test)

dict_keys(['mean_fit_time', 'std_fit_time', 'mean_score_time', 'std_score_time', 'param_class_weight', 'param_max_depth', 'param_min_samples_split', 'params', 'split0_test_score', 'split1_test_score', 'split2_test_score', 'mean_test_score', 'std_test_score', 'rank_test_score', 'split0_train_score', 'split1_train_score', 'split2_train_score', 'mean_train_score', 'std_train_score'])
best depth = 50
best min_samples_split = 500
```

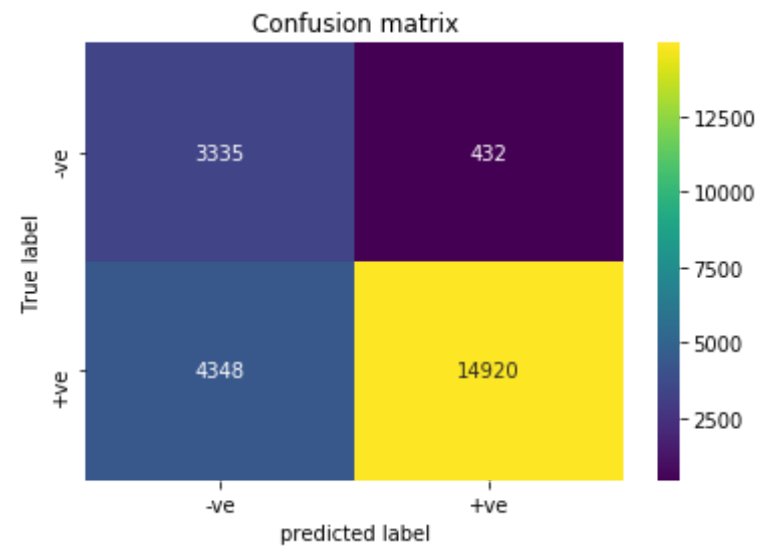


```
In [0]: test(bow_train,bow_cv,bow_test,y_train,y_cv,y_test,50,500)
```

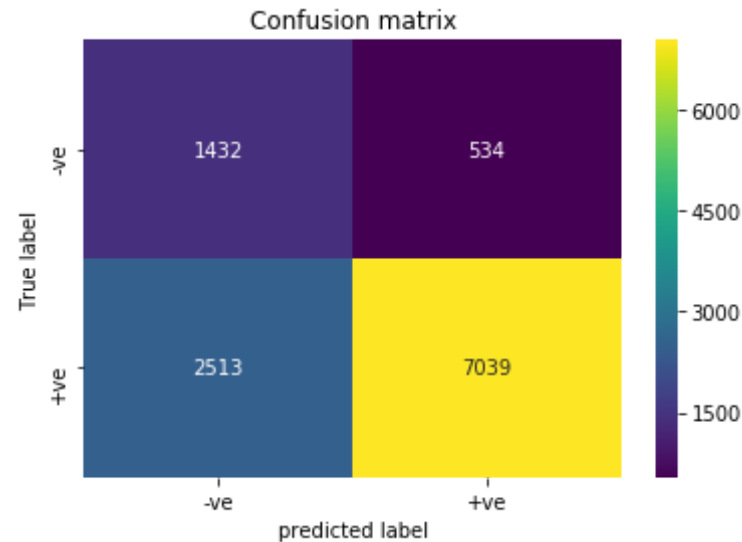
AUC Score: 0.7991635121180676



Confusion matrix for train data



Confusion matrix for test data



[5.1.1] Top 20 important features from SET 1

```
In [0]: # Please write all the code with proper documentation
cnt_vec = CountVectorizer()
p = cnt_vec.fit_transform(X_train)

clf = DecisionTreeClassifier(max_depth = 50, min_samples_split = 500, class_weight='balanced')
clf.fit(p,y_train_)
feat_log_prob = clf.feature_importances_

p = pd.DataFrame(feat_log_prob.T,columns=['+ve'])
p["Feature"] = cnt_vec.get_feature_names()
p = p.sort_values(by = '+ve',kind = 'quicksort',ascending= False)
```

```
In [0]: print("Top 20 features")
print(p[:20])
```

Top 20 features
+ve Feature

17326	0.141480	not
11247	0.086953	great
2304	0.054901	best
6774	0.045274	delicious
18913	0.032885	perfect
15150	0.030806	love
15160	0.025260	loves
10999	0.023742	good
9375	0.016062	favorite
8929	0.014758	excellent
28938	0.014432	wonderful
17160	0.013442	nice
7347	0.013179	disappointed
27403	0.012428	unfortunately
1803	0.010936	bad
16508	0.008253	money
11679	0.007426	happy
25831	0.007119	tasty
26051	0.006575	terrible
844	0.006333	amazing

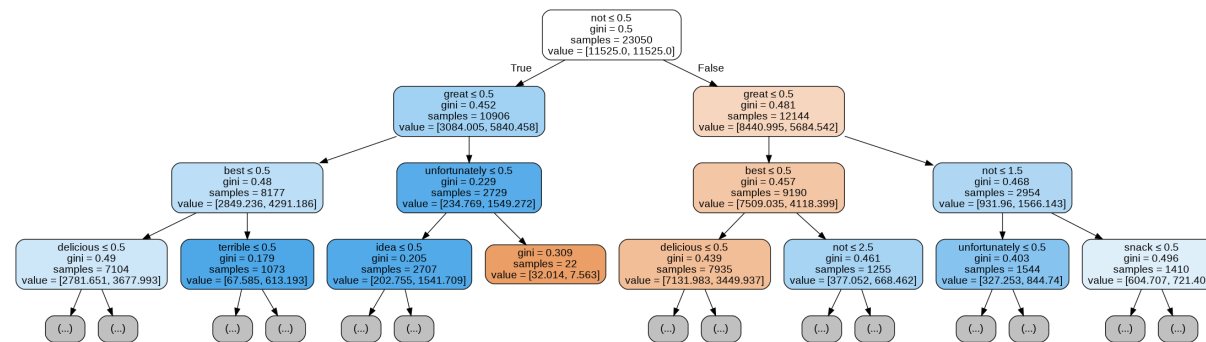
[5.1.2] Graphviz visualization of Decision Tree on BOW, SET 1

```
In [0]: # Please write all the code with proper documentation
from sklearn import tree
import pydotplus
from IPython.display import Image

visulize_data = tree.export_graphviz(clf,out_file = None,max_depth = 3,
feature_names = p["Feature"],filled=True, rounded=True,
special_characters=True)
graph = pydotplus.graph_from_dot_data(visulize_data)
```

```
In [0]: Image(graph.create_png())
```

```
Out[0]:
```

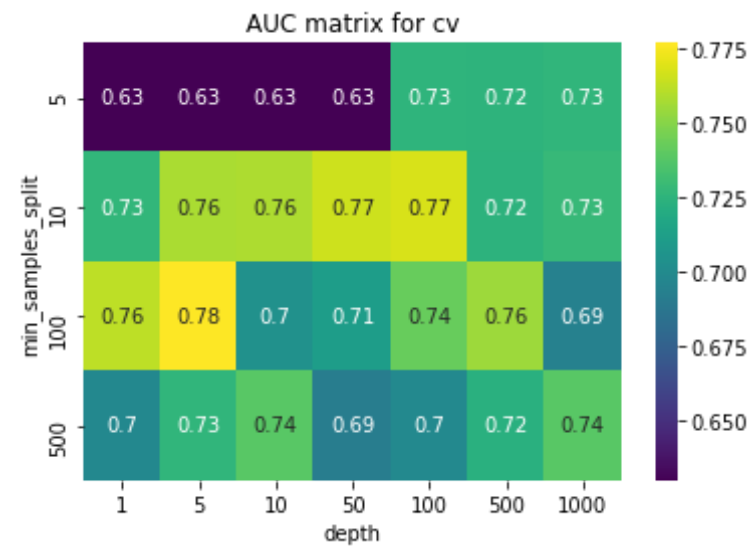
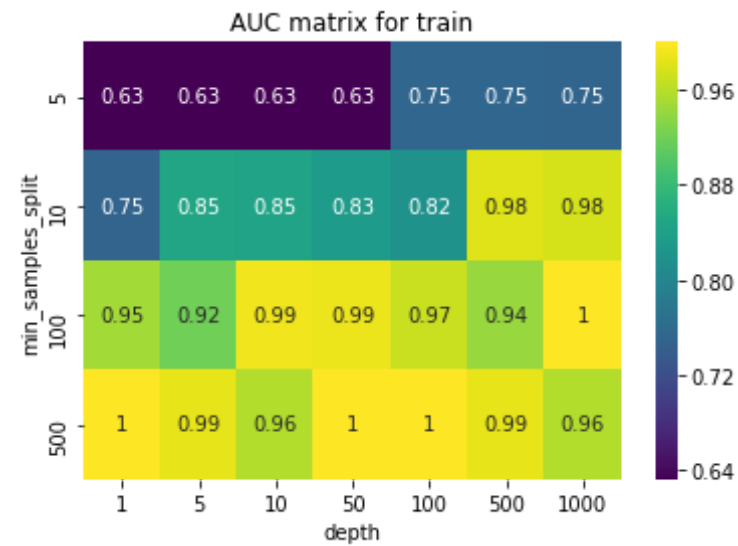


[5.2] Applying Decision Trees on TFIDF, SET 2

```
In [0]: # Please write all the code with proper documentation
tfidf_cv,tfidf_test,tfidf_train = pickle.load(open("tfidf.pkl",'rb'))
```

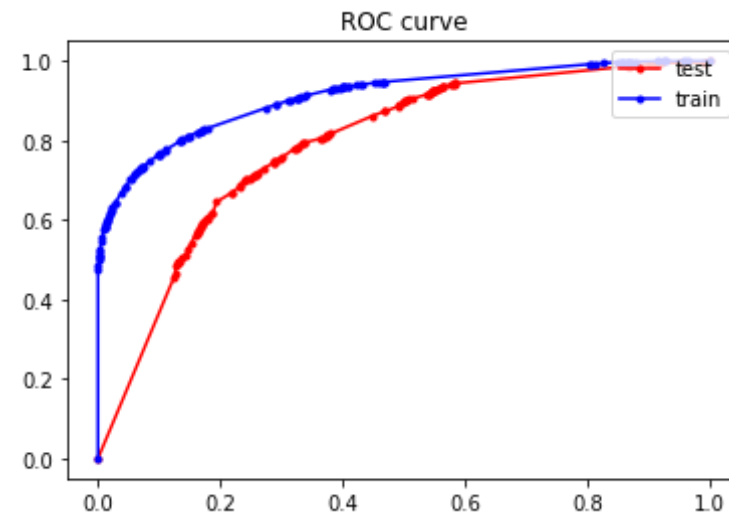
```
In [51]: best_para(tfidf_train,tfidf_cv,tfidf_test,y_train,y_cv,y_test)

dict_keys(['mean_fit_time', 'std_fit_time', 'mean_score_time', 'std_score_time', 'param_class_weight', 'param_max_depth', 'param_min_samples_split', 'params', 'split0_test_score', 'split1_test_score', 'split2_test_score', 'mean_test_score', 'std_test_score', 'rank_test_score', 'split0_train_score', 'split1_train_score', 'split2_train_score', 'mean_train_score', 'std_train_score'])
best depth = 50
best min_samples_split = 500
```

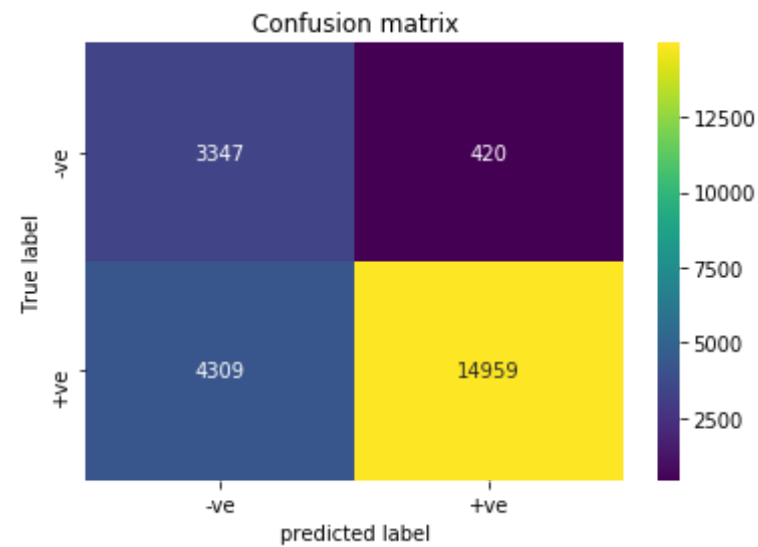


```
In [0]: test(tfidf_train,tfidf_cv,tfidf_test,y_train,y_cv,y_test,50,500)
```

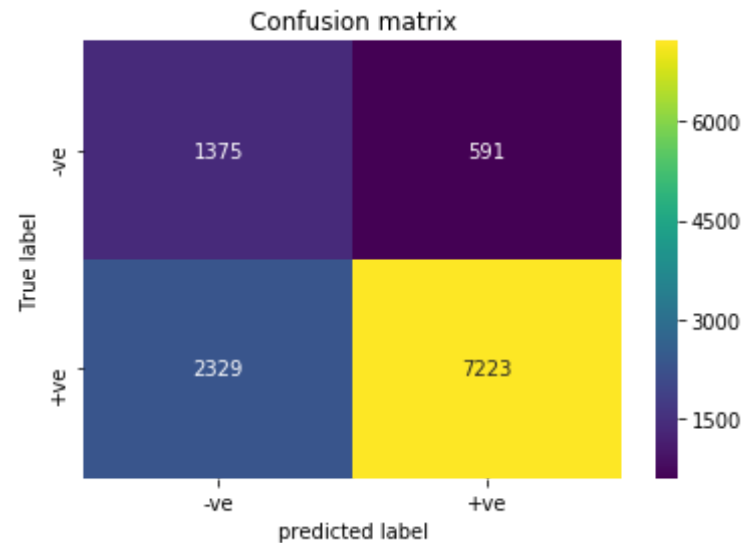
AUC Score: 0.7897589475437549



Confusion matrix for train data



Confusion matrix for test data



[5.2.1] Top 20 important features from SET 2

```
In [0]: # Please write all the code with proper documentation
cnt_vec = TfidfVectorizer()
p = cnt_vec.fit_transform(X_train)

clf = DecisionTreeClassifier(max_depth = 50, min_samples_split = 500, class_weight='balanced')
clf.fit(p, y_train_)
feat_log_prob = clf.feature_importances_

p = pd.DataFrame(feat_log_prob.T, columns=['+ve'])
p["Feature"] = cnt_vec.get_feature_names()
p = p.sort_values(by = '+ve', kind = 'quicksort', ascending= False)
```

```
In [0]: print("Top 20 features")
print(p[:20])
```

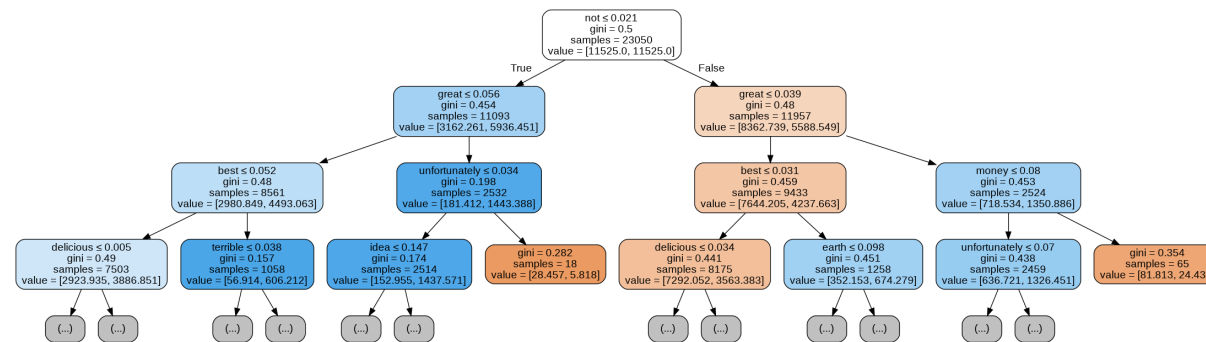
Top 20 features

	+ve	Feature
17326	0.115319	not
11247	0.084625	great
2304	0.055190	best
6774	0.044075	delicious
15150	0.031722	love
18913	0.031495	perfect
10999	0.029550	good
15160	0.023972	loves
16508	0.020617	money
28938	0.016729	wonderful
12073	0.015365	highly
8929	0.015068	excellent
17160	0.014956	nice
9375	0.013630	favorite
27403	0.012961	unfortunately
812	0.011029	always
7347	0.009909	disappointed
844	0.009355	amazing
26051	0.008999	terrible
20137	0.008574	product

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

```
In [0]: # Please write all the code with proper documentation
visulize_data = tree.export_graphviz(clf,out_file = None,max_depth = 3,
feature_names = p["Feature"],filled=True, rounded=True,
special_characters=True)
graph = pydotplus.graph_from_dot_data(visulize_data)
Image(graph.create_png())
```

Out[0]:



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

```
In [0]: # Please write all the code with proper documentation
avg_w2v_cv, avg_w2v_test, avg_w2v_train = pickle.load(open("avg_w2v.pkl",
'rb'))
```

```
In [0]: i=0
list_of_sentence=[]
for sentence in X_train:
    list_of_sentence.append(sentence.split())
```

```
In [53]: 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 occurred atleast 5 times
    w2v_model=Word2Vec(list_of_sentence,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_train_w2v = True, to train your own w2v ")

```

```

[('excellent', 0.8301419019699097), ('good', 0.8039566874504089), ('fantastic', 0.7925791144371033), ('wonderful', 0.7814489603042603), ('awesome', 0.7783530950546265), ('terrific', 0.7645547389984131), ('amazing', 0.7476153373718262), ('decent', 0.7386139035224915), ('delicious', 0.7101123929023743), ('perfect', 0.6913671493530273)]
=====
[('weakest', 0.837216854095459), ('experienced', 0.8155016303062439), ('greatest', 0.7978144288063049), ('tastiest', 0.7795753479003906), ('best', 0.7636765241622925), ('surpasses', 0.7628974914550781), ('disgusting', 0.7605960965156555), ('closest', 0.754327118396759), ('ive', 0.7434050440788269), ('q', 0.7397200465202332)]

```

```

In [54]: w2v_words = list(w2v_model.wv.vocab)
          print("number of words that occurred minimum 5 times ", len(w2v_words))
          print("sample words ", w2v_words[0:50])

```

```

number of words that occurred minimum 5 times 9345
sample words ['happens', 'say', 'name', 'three', 'times', 'michael', 'stars', 'comedy', 'two', 'live', 'old', 'story', 'house', 'coming', 'back', 'supply', 'store', 'couple', 'suddenly', 'get', 'caught', 'inside', 'broken', 'start', 'lake', 'board', 'got', 'hopes', 'small', 'dog', 'steps', 'car', 'starts', 'slide', 'waters', 'minutes', 'later', 'find', 'home', 'somehow', 'light', 'fireplace', 'done', 'magic', 'weird', 'looking', 'dead', 'guy', 'known', 'way']

```

```

In [0]: def vectorize_W2V(data):
          sent_vectors = []; # the avg-w2v for each sentence/review is stored in this list
          for sent in tqdm(data): # 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.split(): # 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)
    return sent_vectors

```

In [56]: *# vectorize all train, test and cv data*

```

avg_w2v_train = vectorize_W2V(X_train)
avg_w2v_cv = vectorize_W2V(X_cv)
avg_w2v_test = vectorize_W2V(X_test)

```

```

100%|██████████| 23050/23050 [00:36<00:00, 629.69it/s]
100%|██████████| 11525/11525 [00:19<00:00, 602.60it/s]
100%|██████████| 11525/11525 [00:19<00:00, 593.79it/s]

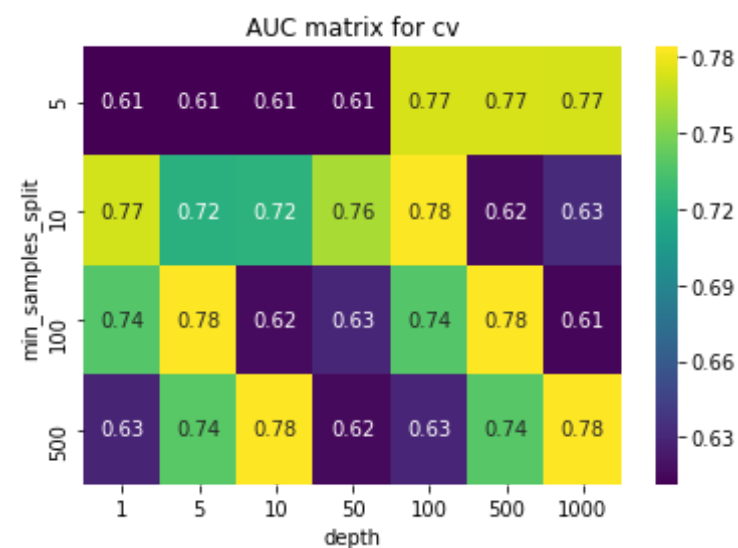
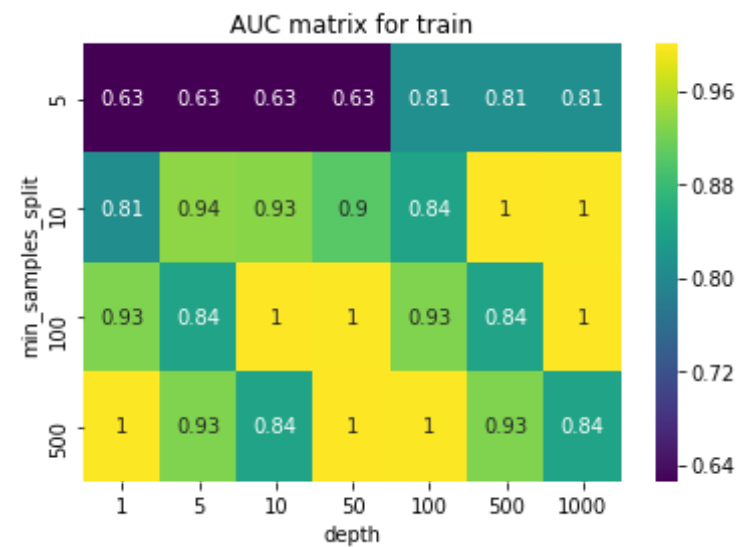
```

In [57]: `best_para(avg_w2v_train, avg_w2v_cv, avg_w2v_test, y_train_, y_cv_, y_test_)`

```

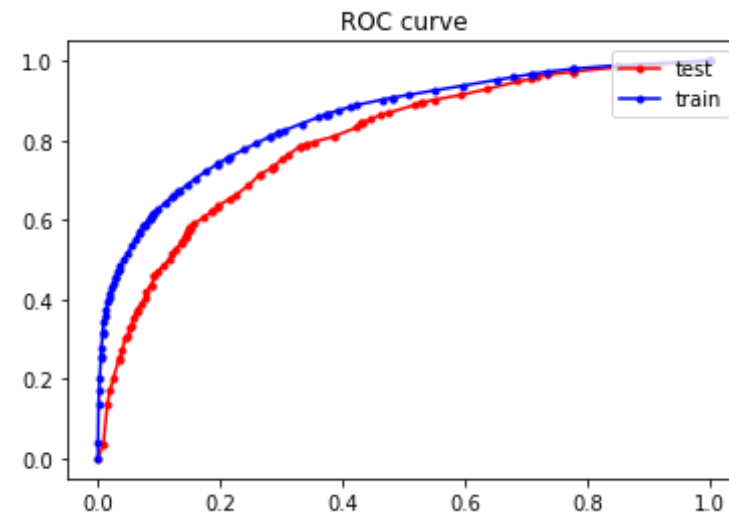
dict_keys(['mean_fit_time', 'std_fit_time', 'mean_score_time', 'std_score_time', 'param_class_weight', 'param_max_depth', 'param_min_samples_split', 'params', 'split0_test_score', 'split1_test_score', 'split2_test_score', 'mean_test_score', 'std_test_score', 'rank_test_score', 'split0_train_score', 'split1_train_score', 'split2_train_score', 'mean_train_score', 'std_train_score'])
best depth = 50
best min_samples_split = 500

```

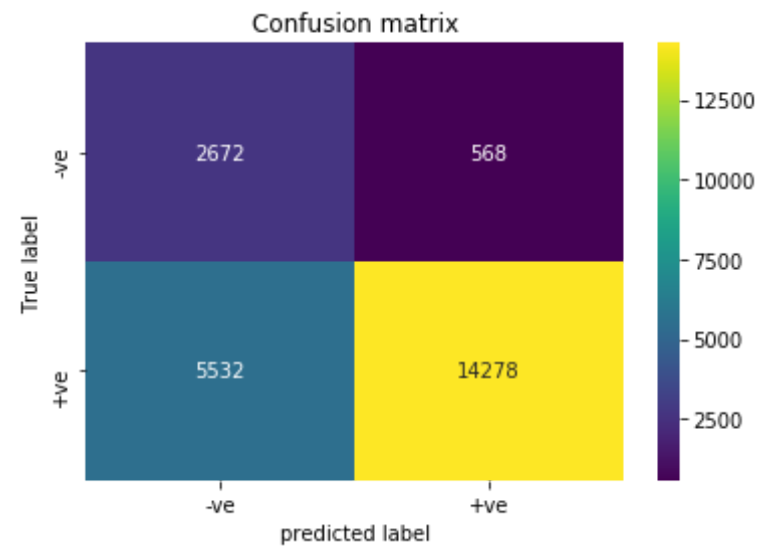


```
In [0]: test(avg_w2v_train,avg_w2v_cv,avg_w2v_test,y_train_,y_cv_,y_test_,10,50
0)
```

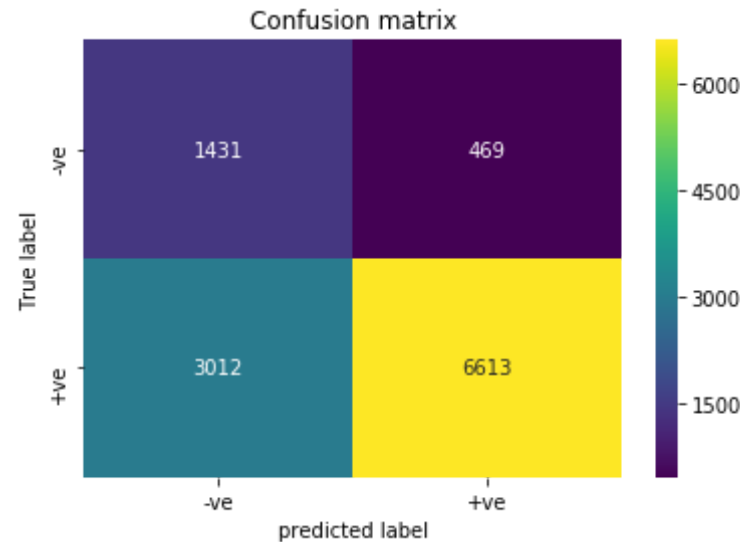
AUC Score: 0.7970249624060151



Confusion matrix for train data



Confusion matrix for test data



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

```
In [0]: # Please write all the code with proper documentation
tfidf_w2v_cv,tfidf_w2v_test,tfidf_w2v_train = pickle.load(open("tfidf_w
2v.pkl", 'rb'))
```

```
In [0]: model_ = TfidfVectorizer(ngram_range=(1,2), min_df=10,max_features=500)
tf_idf_matrix = model_.fit_transform(X_train)
# 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
def vectorizer_W2V_tfidf(data):
    tfidf_sent_vectors = []; # the tfidf-w2v for each sentence/review i
```

```

s stored in this list
row=0;
for sent in tqdm(data): # 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.split(): # 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 corpus
            # sent.count(word) = tf value 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
return tfidf_sent_vectors

```

```

In [60]: tfidf_w2v_train = vectorizer_W2V_tfidf(X_train)
         tfidf_w2v_cv = vectorizer_W2V_tfidf(X_cv)
         tfidf_w2v_test = vectorizer_W2V_tfidf(X_test)

```

```

100%|██████████| 23050/23050 [00:46<00:00, 492.86it/s]
100%|██████████| 11525/11525 [00:23<00:00, 487.81it/s]
100%|██████████| 11525/11525 [00:24<00:00, 473.34it/s]

```

```

In [61]: best_para(tfidf_w2v_train,tfidf_w2v_cv,tfidf_w2v_test,y_train_,y_cv_,y_test_)

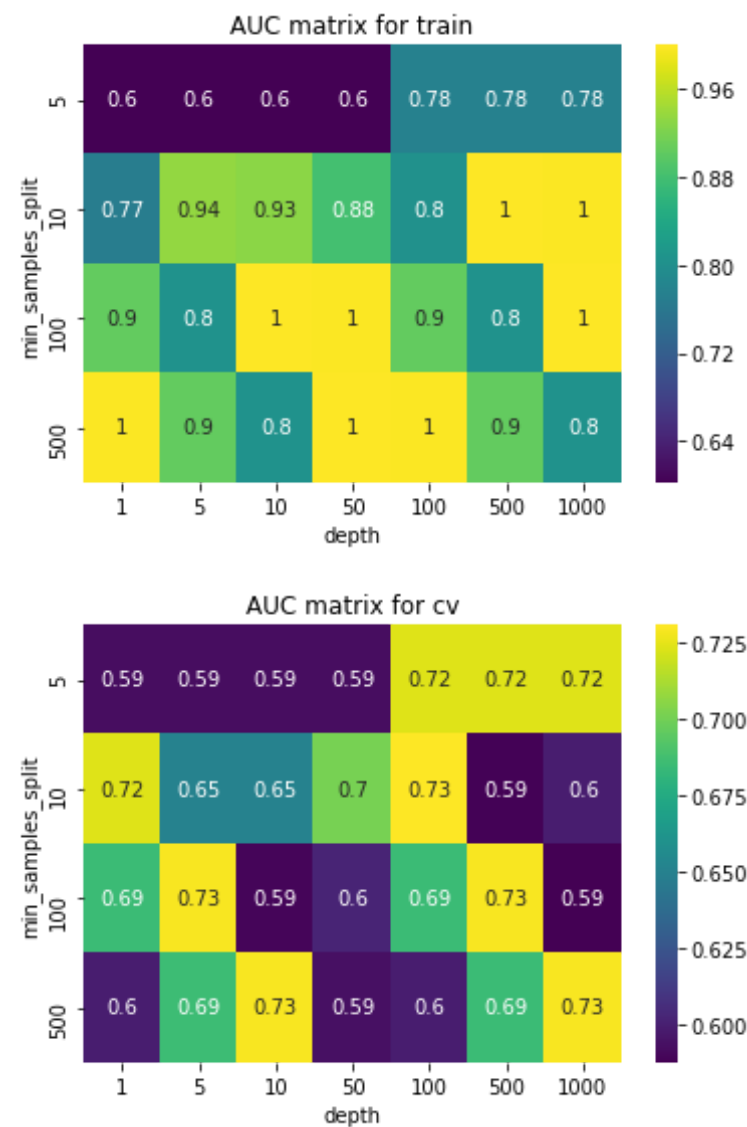
```

```

dict_keys(['mean_fit_time', 'std_fit_time', 'mean_score_time', 'std_score_time', 'param_class_weight', 'param_max_depth', 'param_min_samples_split', 'params', 'split0_test_score', 'split1_test_score', 'split2_test_score', 'mean_test_score', 'std_test_score', 'rank_test_score', 'split0_train_score', 'split1_train_score', 'split2_train_score', 'mean_train_score', 'std_train_score'])

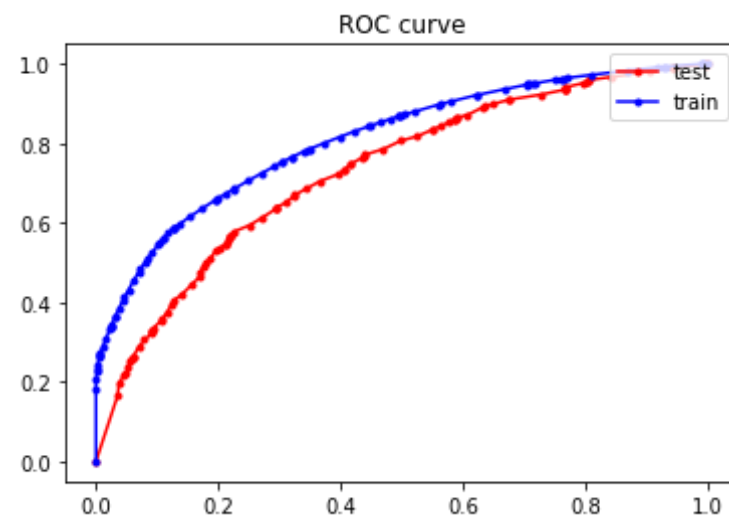
```

```
best depth = 10  
best min_samples_split = 500
```

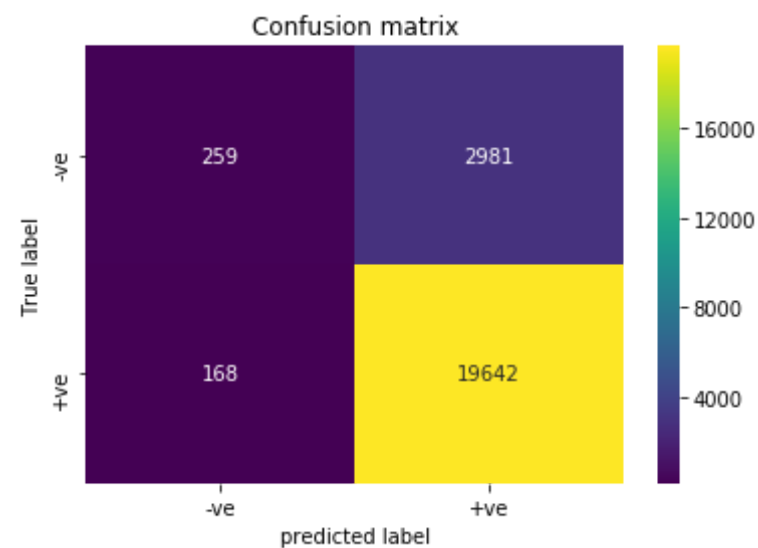


```
In [0]: test(tfidf_w2v_train,tfidf_w2v_cv,tfidf_w2v_test,y_train_,y_cv_,y_test_,50,500)
```

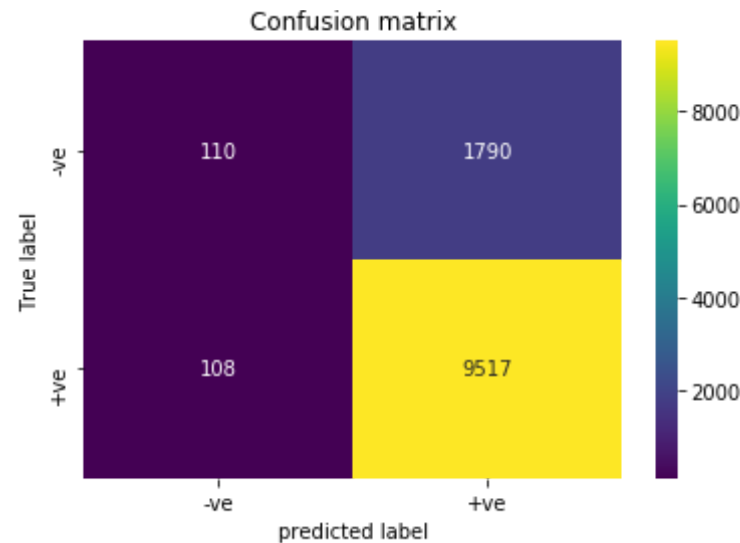
AUC Score: 0.7319252768284348



Confusion matrix for train data



Confusion matrix for test data



[6] Conclusions

```
In [0]: # Please compare all your models using Prettytable library
df = pd.DataFrame({"Model":[" Decision Trees on BOW","Decision Trees on
TFIDF","Decision Trees on AVG W2V","Decision Trees on TFIDF W2V"],
                  "Hyper parameter(Depth)":[50, 50, 10, 50],
                  "Hyper parameter(min_samples_split)":[500, 500, 500, 500],
                  "AUC":[0.7991635121180676, 0.7897589475437549, 0.79
70249624060151, 0.7319252768284348]}
                  ,columns=["Model","Hyper parameter(Depth)","Hyper par
ameter(min_samples_split)","AUC"])
df.sort_values(by="AUC",ascending=False)
```

Out[0]:

	Model	Hyper parameter(Depth)	Hyper parameter(min_samples_split)	AUC
0	Decision Trees on BOW	50	500	0.799164

	Model	Hyper parameter(Depth)	Hyper parameter(min_samples_split)	AUC
2	Decision Trees on AVG W2V	10	500	0.797025
1	Decision Trees on TFIDF	50	500	0.789759
3	Decision Trees on TFIDF W2V	50	500	0.731925