Apply Decision Tree on Amazon Fine Food Reviews

Data Source: https://www.kaggle.com/snap/amazon-fine-food-reviews (https://www.kaggle.com/snap/amazon-fine-food-reviews)
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\HIMANSHU NEGI\Anaconda3\lib\site-packages\gensim\utils.py:1212: Use
rWarning: 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('database.sqlite')

filtered_data = pd.read_sql_query(""" SELECT * FROM Reviews WHERE Score != 3 LIMIT 100000""

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

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

```
Number of data points in our data (100000, 10)
```

Out[2]:

	Id	ProductId	UserId	ProfileName	HelpfulnessNumerator	HelpfulnessDenom
0	1	B001E4KFG0	A3SGXH7AUHU8GW	delmartian	1	
1	2	B00813GRG4	A1D87F6ZCVE5NK	dll pa	0	
2	3	B000LQOCH0	ABXLMWJIXXAIN	Natalia Corres "Natalia Corres"	1	

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 [7]:

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

(80668, 7)

Out[7]:

	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
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 [5]:

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

Out[5]:

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

In [8]:

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

Out[8]:

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 [9]:

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

Out[9]:

	Id	ProductId	UserId	ProfileName	HelpfulnessNumerator	HelpfulnessDen
0	78445	B000HDL1RQ	AR5J8UI46CURR	Geetha Krishnan	2	
1	138317	B000HDOPYC	AR5J8UI46CURR	Geetha Krishnan	2	
2	138277	B000HDOPYM	AR5J8UI46CURR	Geetha Krishnan	2	
3	73791	B000HDOPZG	AR5J8UI46CURR	Geetha Krishnan	2	
4	155049	B000PAQ75C	AR5J8UI46CURR	Geetha Krishnan	2	

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

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

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

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

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

In [10]:

```
#Sorting data according to ProductId in ascending order
sorted_data=filtered_data.sort_values('ProductId', axis=0, ascending=True, inplace=False, k
```

In [11]:

```
#Deduplication of entries
final=sorted_data.drop_duplicates(subset={"UserId","ProfileName","Time","Text"}, keep='firs
final.shape
```

Out[11]:

```
(87775, 10)
```

In [12]:

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

Out[12]:

```
87.775
```

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

In [13]:

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

display.head()
```

Out[13]:

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

In [14]:

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

In [15]:

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

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

```
(87773, 10)
```

Out[15]:

```
1    73592
0    14181
Name: Score, dtype: int64
```

[3] Preprocessing

[3.1]. Preprocessing Review Text

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

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

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

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

In [16]:

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

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

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

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

```
My dogs loves this chicken but its a product from China, so we wont be buyin
g it anymore.  Its very hard to find any chicken products made in the USA bu
t they are out there, but this one isnt.  Its too bad too because its a good
product but I wont take any chances till they know what is going on with the
china imports.
==================================================
The Candy Blocks were a nice visual for the Lego Birthday party but the cand
y has little taste to it.  Very little of the 2 lbs that I bought were eaten
and I threw the rest away.  I would not buy the candy again.
==================================================
was way to hot for my blood, took a bite and did a jig  lol
==================================================
My dog LOVES these treats. They tend to have a very strong fish oil smell. S
o if you are afraid of the fishy smell, don't get it. But I think my dog lik
es it because of the smell. These treats are really small in size. They are
great for training. You can give your dog several of these without worrying
about him over eating. Amazon's price was much more reasonable than any othe
r retailer. You can buy a 1 pound bag on Amazon for almost the same price as
a 6 ounce bag at other retailers. It's definitely worth it to buy a big bag
if your dog eats them a lot.
==================================================
```

In [17]:

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

print(sent_0)
```

```
My dogs loves this chicken but its a product from China, so we wont be buyin
g it anymore.  Its very hard to find any chicken products made in the USA bu
t they are out there, but this one isnt.  Its too bad too because its a good
product but I wont take any chances till they know what is going on with the
china imports.
```

In [18]:

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

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

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

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

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

```
My dogs loves this chicken but its a product from China, so we wont be buyin
g it anymore.  Its very hard to find any chicken products made in the USA bu
t they are out there, but this one isnt.  Its too bad too because its a good
product but I wont take any chances till they know what is going on with the
china imports.
==================================================
The Candy Blocks were a nice visual for the Lego Birthday party but the cand
y has little taste to it.  Very little of the 2 lbs that I bought were eaten
and I threw the rest away.  I would not buy the candy again.
==================================================
was way to hot for my blood, took a bite and did a jig  lol
==================================================
My dog LOVES these treats. They tend to have a very strong fish oil smell. S
o if you are afraid of the fishy smell, don't get it. But I think my dog lik
es it because of the smell. These treats are really small in size. They are
great for training. You can give your dog several of these without worrying
about him over eating. Amazon's price was much more reasonable than any othe
r retailer. You can buy a 1 pound bag on Amazon for almost the same price as
a 6 ounce bag at other retailers. It's definitely worth it to buy a big bag
if your dog eats them a lot.
```

In [19]:

```
# 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 [20]:

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

```
was way to hot for my blood, took a bite and did a jig  lol
==================================================
```

In [21]:

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

```
My dogs loves this chicken but its a product from China, so we wont be buyin
g it anymore.  Its very hard to find any chicken products made in the USA bu
t they are out there, but this one isnt.  Its too bad too because its a good
product but I wont take any chances till they know what is going on with the
china imports.
```

In [22]:

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

```
was way to hot for my blood took a bite and did a jig lol
```

In [23]:

```
# 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'll", "you'd", 'your', 'yours', 'yourself', 'yourselves', 'he', 'him', 'his
            'she', "she's", 'her', 'hers', 'herself', 'it', "it's", 'its', 'itself', 'they'
            'theirs', 'themselves', 'what', 'which', 'who', 'whom', 'this', 'that', "that'l
            'am', 'is', 'are', 'was', 'were', 'be', 'been', 'being', 'have', 'has', 'had',
            'did', 'doing', 'a', 'an', 'the', 'and', 'but', 'if', 'or', 'because', 'as', 'u
            'at', 'by', 'for', 'with', 'about', 'against', 'between', 'into', 'through', 'd
            'above', 'below', 'to', 'from', 'up', 'down', 'in', 'out', 'on', 'off', 'over',
            'then', 'once', 'here', 'there', 'when', 'where', 'why', 'how', 'all', 'any', '
            'most', 'other', 'some', 'such', 'only', 'own', 'same', 'so', 'than', 'too', 'v
            's', 't', 'can', 'will', 'just', 'don', "don't", 'should', "should've", 'now',
            've', 'y', 'ain', 'aren', "aren't", 'couldn', "couldn't", 'didn', "didn't", 'do
            "hadn't", 'hasn', "hasn't", 'haven', "haven't", 'isn', "isn't", 'ma', 'mightn',
            "mustn't", 'needn', "needn't", 'shan', "shan't", 'shouldn', "shouldn't", 'wasn'
            'won', "won't", 'wouldn', "wouldn't"])
```

In [24]:

```
# Combining all the above stundents
from tqdm import tqdm
preprocessed_reviews = []
# tqdm is for printing the status bar
for sentance in tqdm(final['Text'].values):
    sentance = re.sub(r"http\S+", "", sentance)
    sentance = BeautifulSoup(sentance, 'lxml').get_text()
    sentance = decontracted(sentance)
    sentance = re.sub("\S*\d\S*", "", sentance).strip()
    sentance = re.sub('[^A-Za-z]+', ' ', sentance)
    # https://gist.github.com/sebleier/554280
    sentance = ' '.join(e.lower() for e in sentance.split() if e.lower() not in stopwords)
    preprocessed_reviews.append(sentance.strip())
```

```
100%|██████████████████████████████████████████████████████████████
███| 87773/87773 [00:47<00:00, 1835.14it/s]
```

In [25]:

```
preprocessed_reviews[1500]
```

Out[25]:

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

[5] Assignment 8: Decision Trees

Applying Decision Trees

In [26]:

```
from sklearn.metrics import accuracy_score,confusion_matrix,f1_score,precision_score,recall
from sklearn.model_selection import GridSearchCV
from sklearn.tree import DecisionTreeClassifier
from sklearn.metrics import roc_curve,auc


from sklearn.model_selection import cross_val_score
from collections import Counter
from sklearn.metrics import accuracy_score
from sklearn.model_selection import cross_validate
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
```

In [27]:

```
#spilliting data
from sklearn.model_selection import train_test_split

# split the data set into train and test
X_1, X_test, y_1, y_test = train_test_split(preprocessed_reviews, final['Score'], test_size

# split the train data set into cross validation train and cross validation test
X_tr, X_cv, y_tr, y_cv = train_test_split(X_1, y_1, test_size=0.3)
```

[5.1] Applying Decision Trees on BOW, SET 1

In [28]:

```
# Please write all the code with proper documentation
#code for VECTORIZER
count_vect = CountVectorizer(min_df = 10)
Xbow_tr = count_vect.fit_transform(X_tr)
Xbow_test = count_vect.transform(X_test)
Xbow_cv = count_vect.transform(X_cv)
print("the type of count vectorizer :",type(X_tr))
print("the shape of out text BOW vectorizer : ",Xbow_tr.get_shape())
print("the number of unique words :", Xbow_tr.get_shape()[1])
```

```
the type of count vectorizer : <class 'list'>
the shape of out text BOW vectorizer :  (43008, 8194)
the number of unique words : 8194
```

In [29]:

```
# Data-preprocessing: Standardizing the data
import warnings
warnings.filterwarnings('ignore')


from sklearn.preprocessing import StandardScaler
sc = StandardScaler(with_mean=False)
Xbow_tr_std = sc.fit_transform(Xbow_tr)
Xbow_test_std = sc.transform(Xbow_test)
Xbow_cv_std = sc.fit_transform(Xbow_cv)
```

Code for hyperparameter tuning Depth

In [30]:

```
#code for hyperparameter tuning
import numpy as np
hyper =  [1, 5, 10, 50, 100, 500, 1000]

auc1=[]
auc2=[]

for j in hyper:

    model = DecisionTreeClassifier(max_depth=j)
    model.fit(Xbow_tr_std, y_tr)

    probs = model.predict_proba(Xbow_tr_std)
    preds = probs[:,1]
    roc_auc1=metrics.roc_auc_score(y_tr, preds)
    auc1.append(roc_auc1)


    probs = model.predict_proba(Xbow_cv_std)
    preds = probs[:,1]
    roc_auc2=metrics.roc_auc_score(y_cv, preds)
    auc2.append(roc_auc2)
```

In [31]:

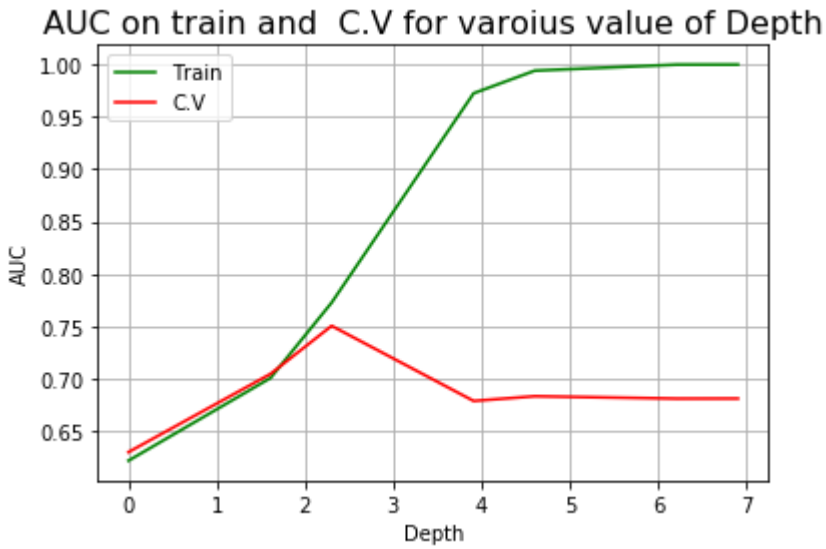
```
r=[]
import math
for p in (hyper):
    q=math.log(p)
    r.append(q)
print(r)
```

```
[0.0, 1.6094379124341003, 2.302585092994046, 3.912023005428146, 4.6051701859
88092, 6.214608098422191, 6.907755278982137]
```

In [32]:

```
#code for plotting graph
print(hyper)

print('----------------------------------------------------------------------------------------------------')
print(auc1)

print('----------------------------------------------------------------------------------------------------')
print('----------------------------------------------------------------------------------------------------')
print(auc2)

plt.title('AUC on train and  C.V for varoius value of Depth',size=16)
plt.plot(r, auc1,'g',label ='Train')
plt.plot(r, auc2,'r',label ='C.V')

plt.ylabel('AUC',size=10)
plt.xlabel('Depth',size=10)
plt.grid()
plt.legend()
plt.show()
```

```
[1, 5, 10, 50, 100, 500, 1000]
------------------------------------------------------------------------------
---------------
[0.6216320193344282, 0.700566588177966, 0.772607873628504, 0.972589854566916
2, 0.994222918842723, 0.9999971066840552, 0.9999971066840552]
------------------------------------------------------------------------------
---------------
------------------------------------------------------------------------------
--------------
[0.6297757147527574, 0.704474046802959, 0.7504032188939301, 0.6786671609580
3, 0.6831574287255924, 0.6808854005950544, 0.6809482584562871]
```



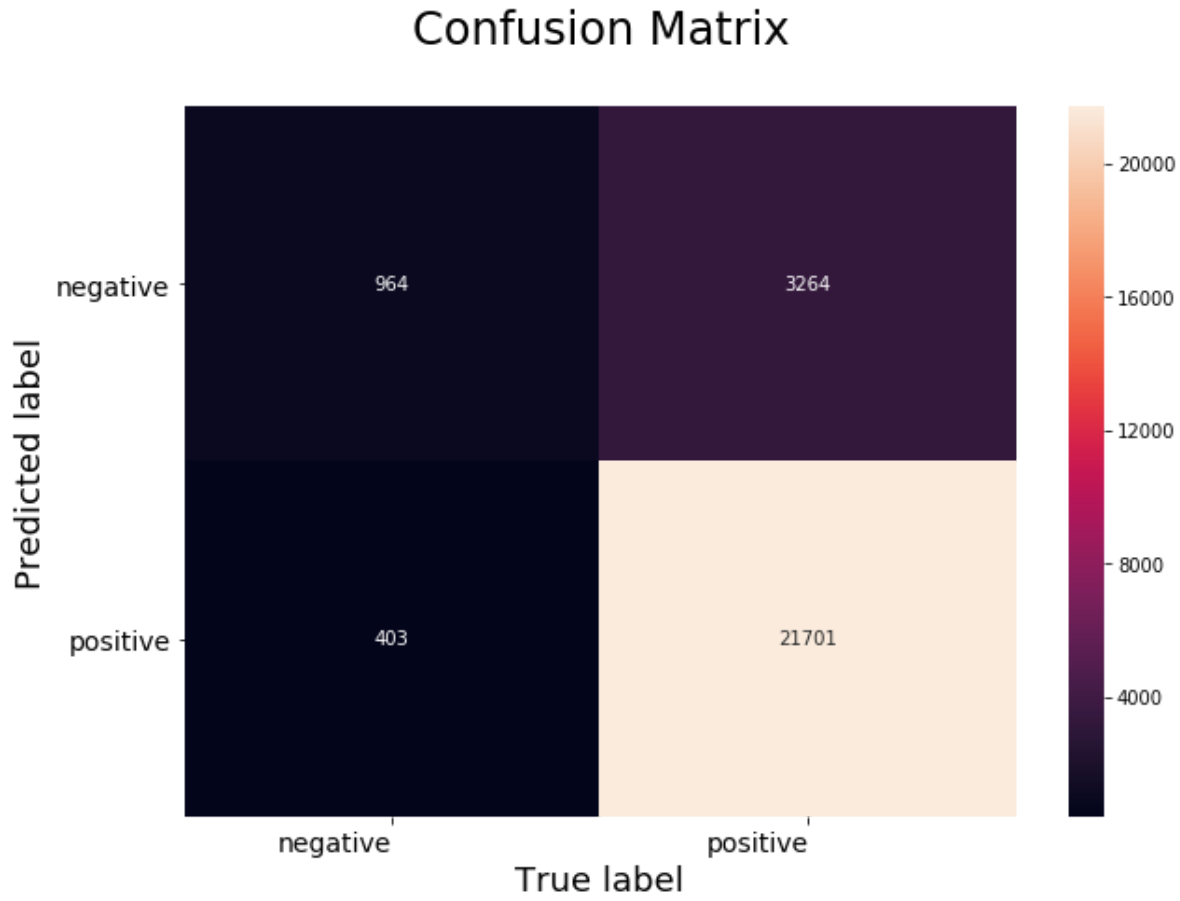
In [33]:

```
dtc = DecisionTreeClassifier(max_depth=10)

# fitting the model
dtc.fit(Xbow_tr_std, y_tr)

# predict the response
pred = dtc.predict(Xbow_test_std)

# evaluate accuracy
acc = accuracy_score(y_test, pred) * 100
print('\nThe accuracy of the Decision tree classifier for depth = %f is %f%%' % (10, acc))
```

```
The accuracy of the Decision tree classifier for depth = 10.000000 is 86.073
978%
```

In [34]:

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

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

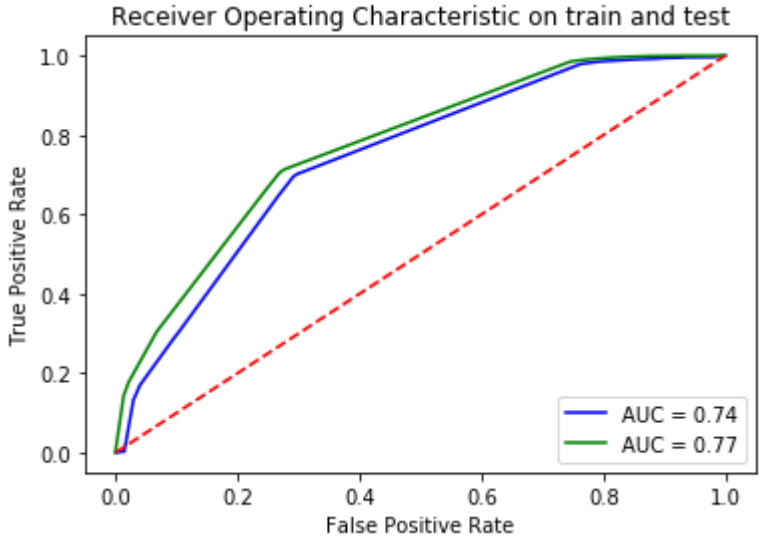


In [36]:

```
dtc.fit(Xbow_tr_std, y_tr)
probs2 = dtc.predict_proba(Xbow_tr_std)
preds2 = probs2[:,1]
fpr2, tpr2, threshold2 = metrics.roc_curve(y_tr, preds2)
roc_auc2 = metrics.auc(fpr2, tpr2)


probs1 = dtc.predict_proba(Xbow_test_std)
preds1 = probs1[:,1]
fpr1, tpr1, threshold1 = metrics.roc_curve(y_test, preds1)
roc_auc1 = metrics.auc(fpr1, tpr1)
```

In [37]:

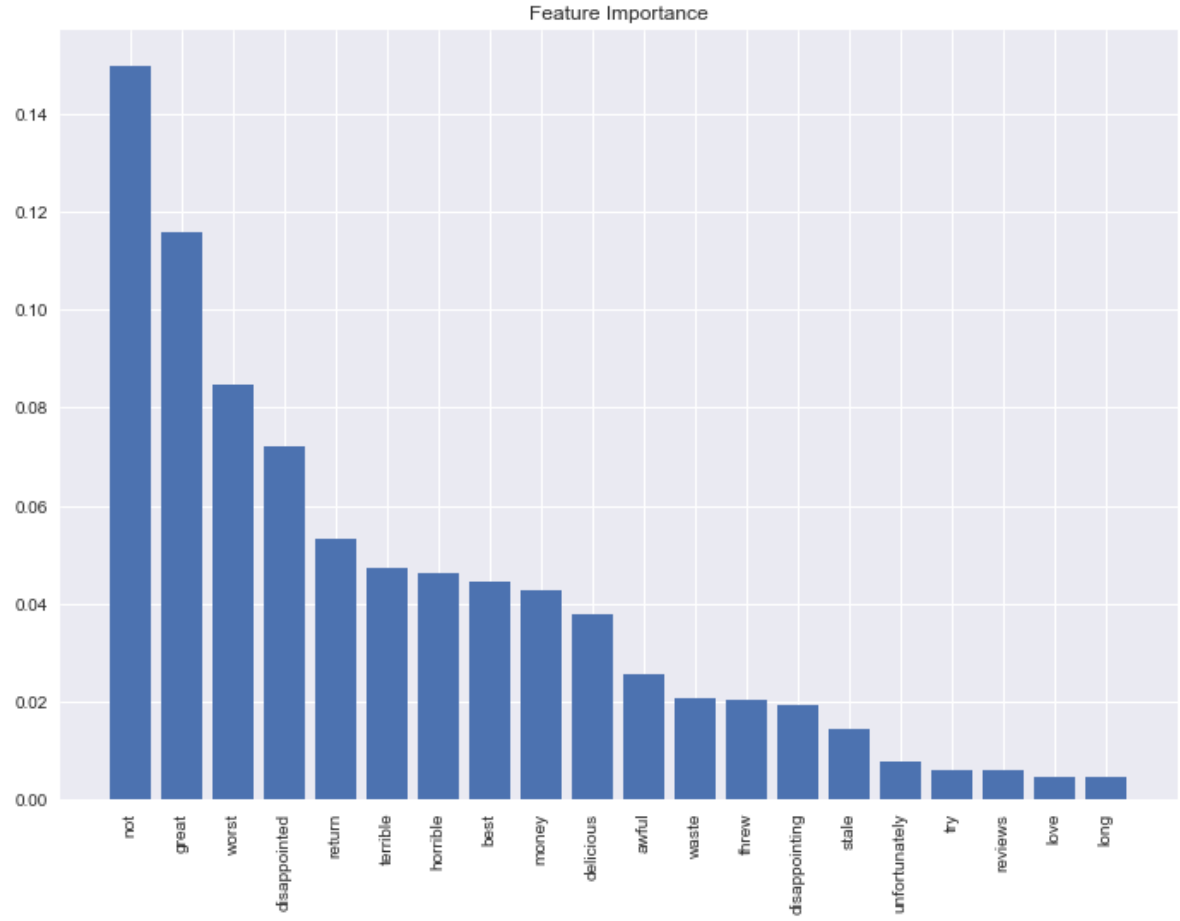
```
plt.title('Receiver Operating Characteristic on train and test')
plt.plot(fpr1, tpr1, 'b', label = 'AUC = %0.2f' % roc_auc1)
plt.plot(fpr2, tpr2, 'g', label = 'AUC = %0.2f' % roc_auc2)
#plt.plot(neighbors, auc1,'g')
#plt.plot(neighbors, auc2,'r')
plt.legend(loc = 'lower right')
plt.plot([0, 1], [0, 1],'r--')
plt.ylabel('True Positive Rate')
plt.xlabel('False Positive Rate')
plt.show()
```



[5.1.1] Top 20 important features from SET 1

In [38]:

```
#refer code snippet for top 20 feature extraction
#https://github.com/cyanamous/Amazon-Food-Reviews-Analysis-and-Modelling/blob/master/6%20Am
# Calculate feature importances from decision trees
importances = dtc.feature_importances_

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

# Rearrange feature names so they match the sorted feature importances
names = count_vect.get_feature_names()

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

# Create plot
plt.figure()

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

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

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

# Show plot
plt.show()
# uni_gram.get_feature_names()from sklearn import tree
```



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

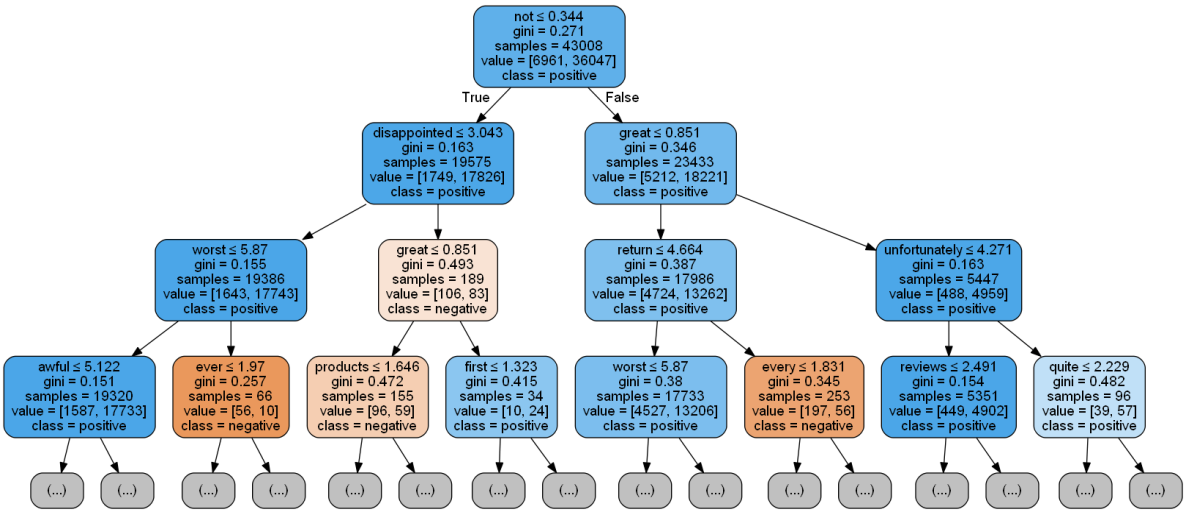
In [39]:

```
import pydotplus
from IPython.display import Image
from IPython.display import SVG
from graphviz import Source
from IPython.display import display
```

In [52]:

```
target = ['negative','positive']
# Create DOT data
data = tree.export_graphviz(dtc,out_file=None,class_names=target,feature_names=count_vect.g

# Draw graph
graph = pydotplus.graph_from_dot_data(data)

# Show graph
Image(graph.create_png())
```

Out[52]:



Code for hyperparameter tuning min_samples_split

In [54]:

```
#code for hyperparameter tuning
import numpy as np
hyper =  [5, 10, 100, 500]

auc1=[]
auc2=[]

for j in hyper:

    model = DecisionTreeClassifier(min_samples_split=j)
    model.fit(Xbow_tr_std, y_tr)

    probs = model.predict_proba(Xbow_tr_std)
    preds = probs[:,1]
    roc_auc1=metrics.roc_auc_score(y_tr, preds)
    auc1.append(roc_auc1)


    probs = model.predict_proba(Xbow_cv_std)
    preds = probs[:,1]
    roc_auc2=metrics.roc_auc_score(y_cv, preds)
    auc2.append(roc_auc2)
```

In [56]:

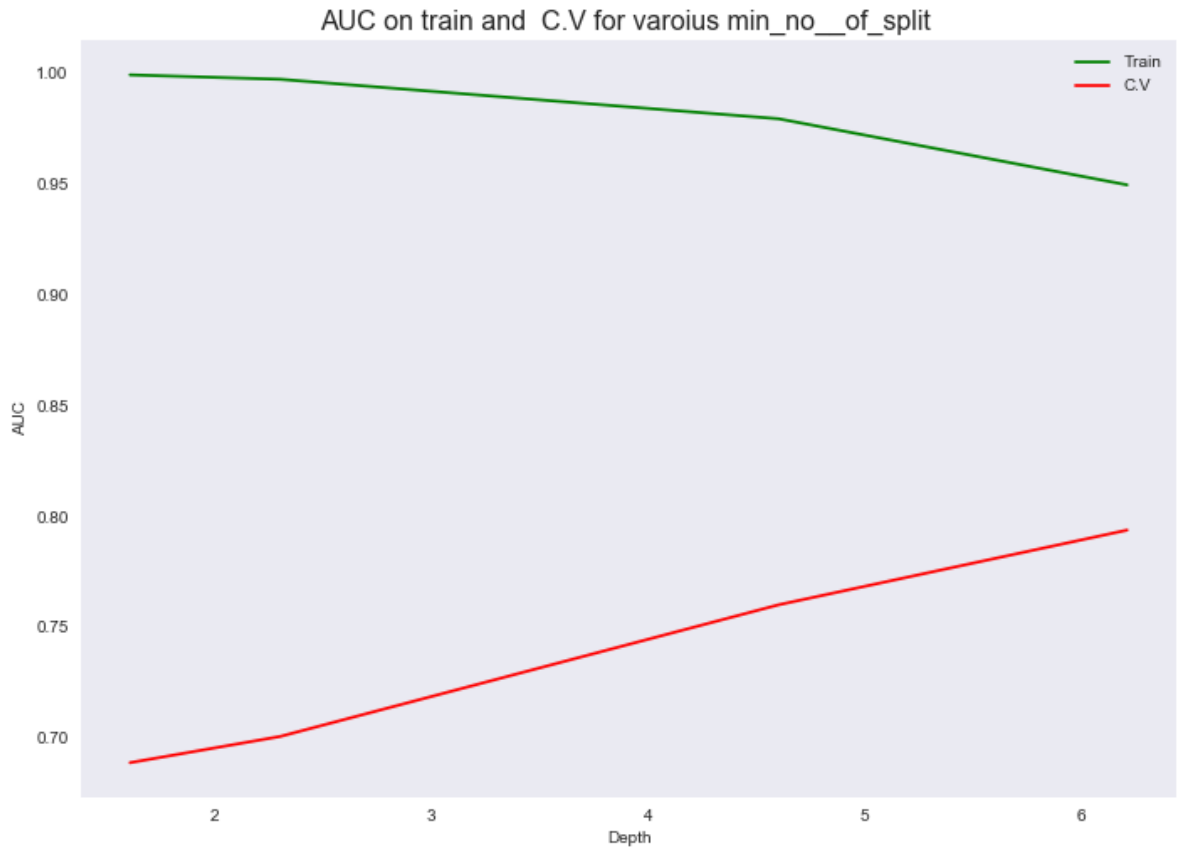
```
r2=[]
import math
for p in (hyper):
    q=math.log(p)
    r2.append(q)
print(r)
```

```
[0.0, 1.6094379124341003, 2.302585092994046, 3.912023005428146, 4.6051701859
88092, 6.214608098422191, 6.907755278982137]
```

In [57]:

```
#code for plotting graph
print(hyper)

print('------------------------------------------------------------------------------------------------------------------------')
print(auc1)

print('------------------------------------------------------------------------------------------------------------------------')
print('------------------------------------------------------------------------------------------------------------------------')
print(auc2)

plt.title('AUC on train and  C.V for varoius min_no__of_split ',size=16)
plt.plot(r2, auc1,'g',label ='Train')
plt.plot(r2, auc2,'r',label ='C.V')

plt.ylabel('AUC',size=10)
plt.xlabel('Depth',size=10)
plt.grid()
plt.legend()
plt.show()
```

```
[5, 10, 100, 500]
------------------------------------------------------------------------------------------
----------------
[0.9992677021329003, 0.997273848771405, 0.9794494662981835, 0.94961811357976
38]
------------------------------------------------------------------------------------------
----------------
------------------------------------------------------------------------------------------
--------------
[0.6888239112343102, 0.7006522390558922, 0.7600902560098522, 0.7937941044001
541]
```



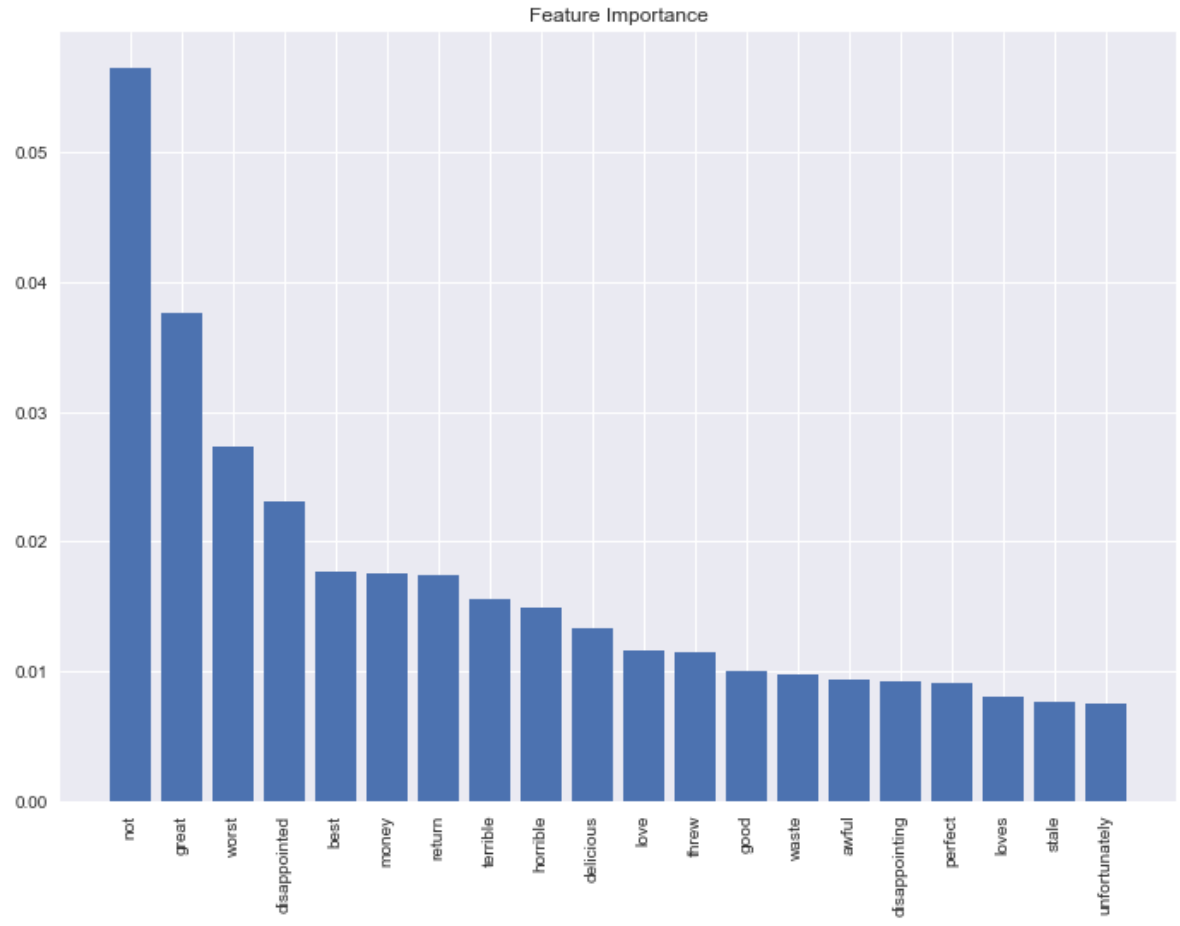
In [61]:

```
dtc = DecisionTreeClassifier(min_samples_split=100)

# fitting the model
dtc.fit(Xbow_tr_std, y_tr)

# predict the response
pred = dtc.predict(Xbow_test_std)

# evaluate accuracy
acc = accuracy_score(y_test, pred) * 100
print('\nThe accuracy of the Decision tree classifier for split = %f is %f%%' % (100, acc))
```

```
The accuracy of the Decision tree classifier for split = 100.000000 is 84.20
5529%
```

In [59]:

```
#refer code snippet for top 20 feature extraction
#https://github.com/cyanamous/Amazon-Food-Reviews-Analysis-and-Modelling/blob/master/6%20Am
# Calculate feature importances from decision trees
importances = dtc.feature_importances_

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

# Rearrange feature names so they match the sorted feature importances
names = count_vect.get_feature_names()

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

# Create plot
plt.figure()

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

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

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

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



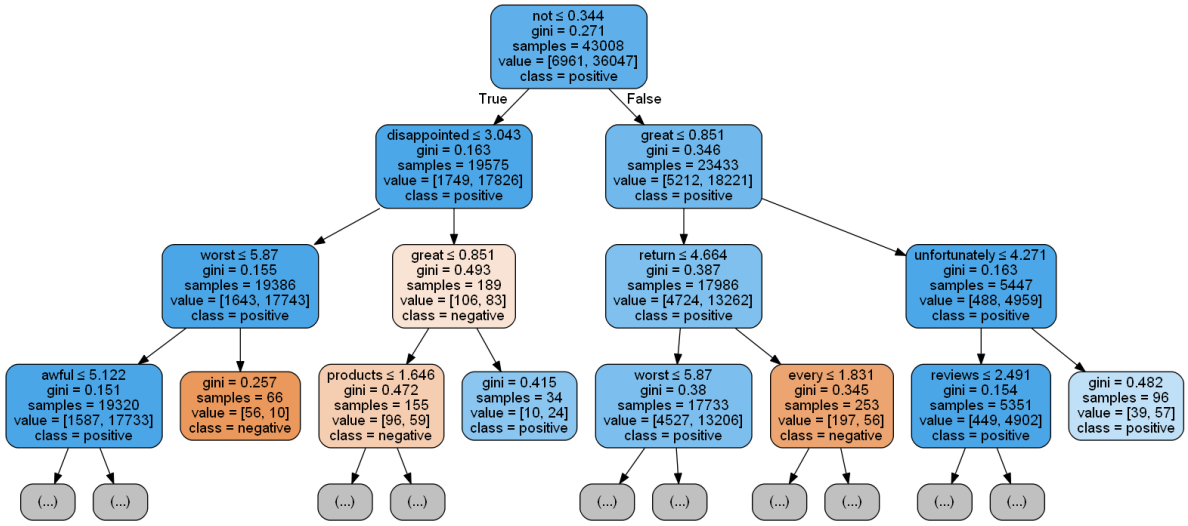
In [59]:

```
from sklearn import tree
import pydotplus
import networkx as nx
from graphviz import render
from IPython.display import Image
from IPython.display import SVG
from graphviz import Source
from IPython.display import display
```

In [62]:

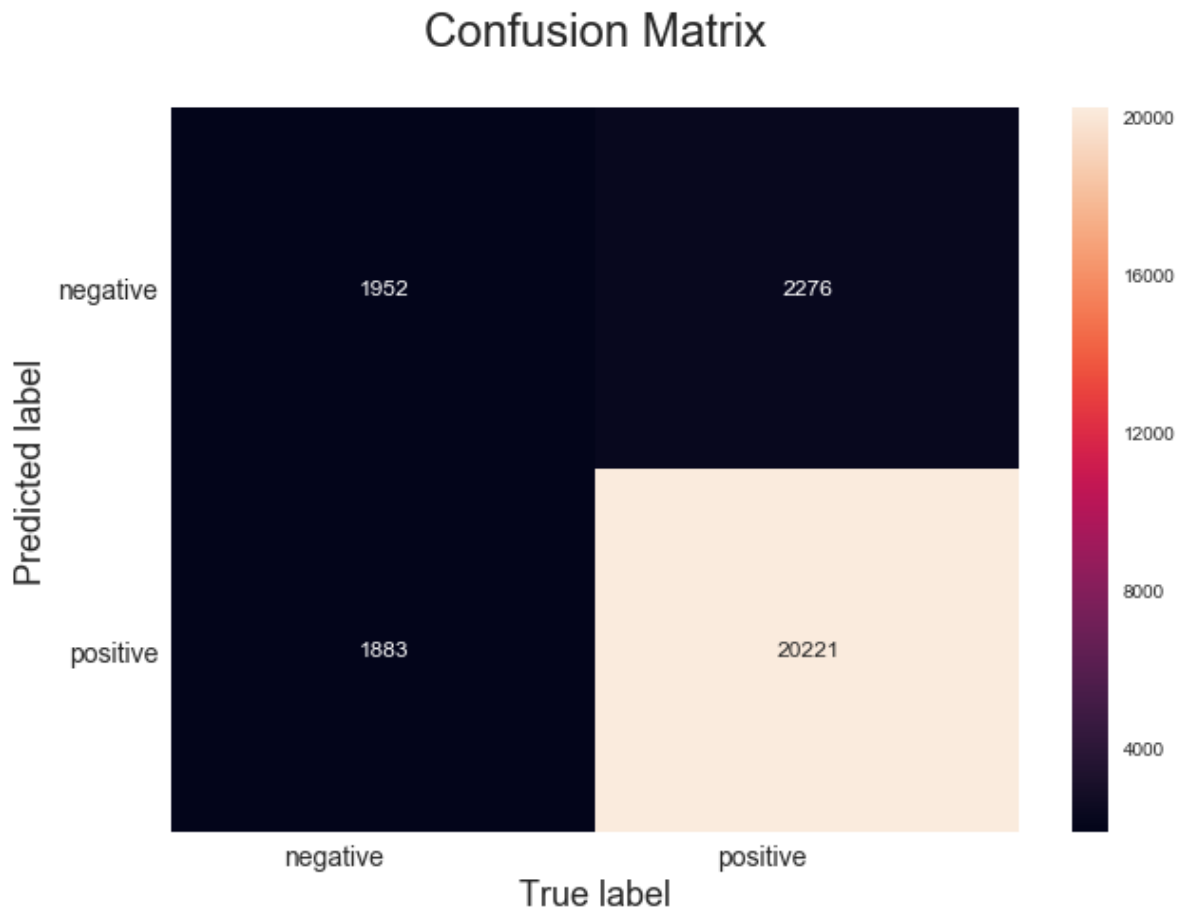
```
target = ['negative','positive']
# Create DOT data
data = tree.export_graphviz(dtc,out_file=None,class_names=target,feature_names=count_vect.g

# Draw graph
graph = pydotplus.graph_from_dot_data(data)

# Show graph
Image(graph.create_png())
```

Out[62]:



In [63]:

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

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

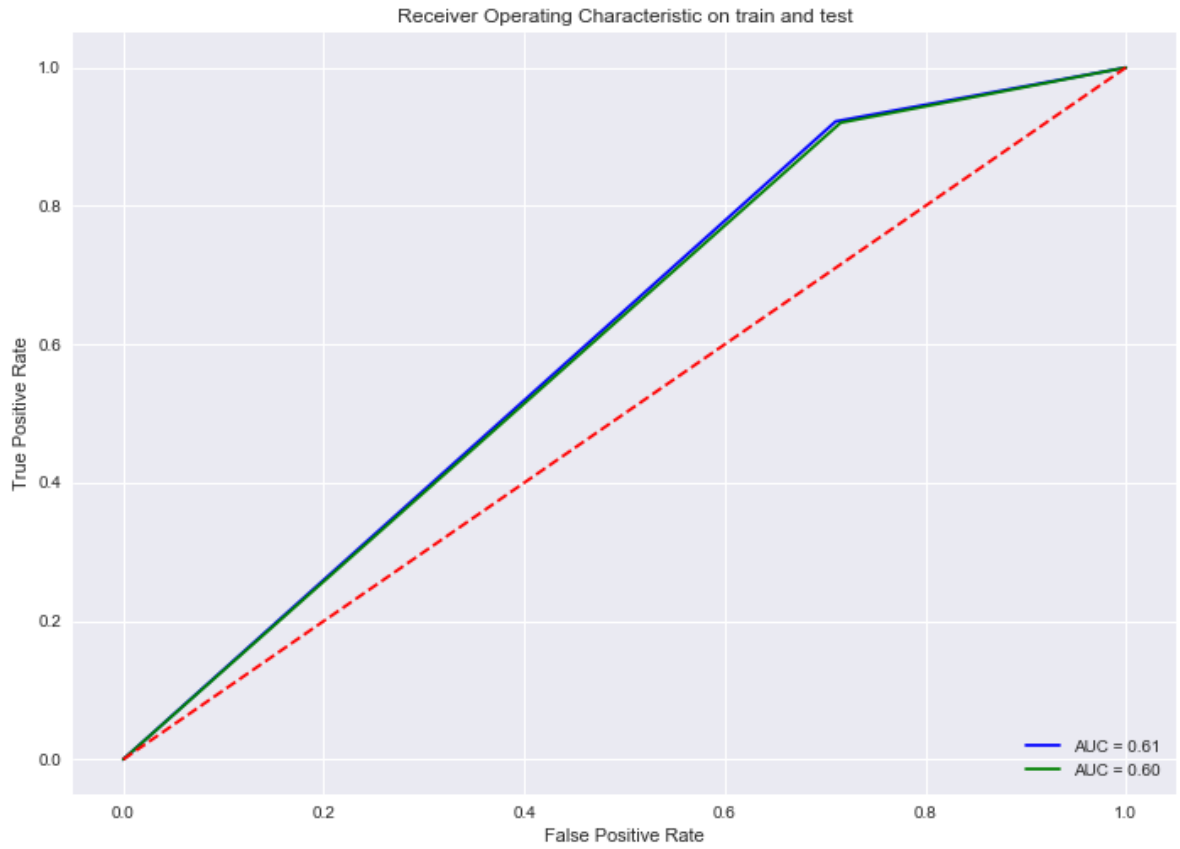


In [65]:

```
dtc = DecisionTreeClassifier(min_samples_split=100)

# fitting the model
dtc.fit(Xbow_tr_std, y_tr)
probs2 = dtc.predict(Xbow_tr)
preds2 = probs2
fpr2, tpr2, threshold2 = metrics.roc_curve(y_tr, preds2)
roc_auc2 = metrics.auc(fpr2, tpr2)


probs1 = dtc.predict(Xbow_test)
preds1 = probs1
fpr1, tpr1, threshold1 = metrics.roc_curve(y_test, preds1)
roc_auc1 = metrics.auc(fpr1, tpr1)
```

In [66]:

```
plt.title('Receiver Operating Characteristic on train and test')
plt.plot(fpr1, tpr1, 'b', label = 'AUC = %0.2f' % roc_auc1)
plt.plot(fpr2, tpr2, 'g', label = 'AUC = %0.2f' % roc_auc2)
#plt.plot(neighbors, auc1,'g')
#plt.plot(neighbors, auc2,'r')
plt.legend(loc = 'lower right')
plt.plot([0, 1], [0, 1],'r--')
plt.ylabel('True Positive Rate')
plt.xlabel('False Positive Rate')
plt.show()
```



[5.2] Applying Decision Trees on TFIDF, SET 2

In [67]:

```
# Please write all the code with proper documentation
#code for VECTORIZER
count_vect  = TfidfVectorizer(ngram_range=(1,2), min_df=10)

Xbow_tr = count_vect.fit_transform(X_tr)
Xbow_test = count_vect.transform(X_test)
Xbow_cv = count_vect.transform(X_cv)
print("the type of count vectorizer :",type(X_tr))
print("the shape of out text BOW vectorizer : ",Xbow_tr.get_shape())
print("the number of unique words :", Xbow_tr.get_shape()[1])
```

```
the type of count vectorizer : <class 'list'>
the shape of out text BOW vectorizer :  (43008, 25697)
the number of unique words : 25697
```

In [68]:

```
# Data-preprocessing: Standardizing the data
import warnings
warnings.filterwarnings('ignore')


from sklearn.preprocessing import StandardScaler
sc = StandardScaler(with_mean=False)
Xbow_tr_std = sc.fit_transform(Xbow_tr)
Xbow_test_std = sc.transform(Xbow_test)
Xbow_cv_std = sc.fit_transform(Xbow_cv)
```

Code for hyperparameter tuning Depth

In [ ]:

```
import numpy as np
hyper =  [1, 5, 10, 50, 100, 500, 1000]

auc1=[]
auc2=[]

for j in hyper:

    model = DecisionTreeClassifier(max_depth=j)
    model.fit(Xbow_tr_std, y_tr)

    probs = model.predict_proba(Xbow_tr_std)
    preds = probs[:,1]
    roc_auc1=metrics.roc_auc_score(y_tr, preds)
    auc1.append(roc_auc1)


    probs = model.predict_proba(Xbow_cv_std)
    preds = probs[:,1]
    roc_auc2=metrics.roc_auc_score(y_cv, preds)
    auc2.append(roc_auc2)
```

In [71]:

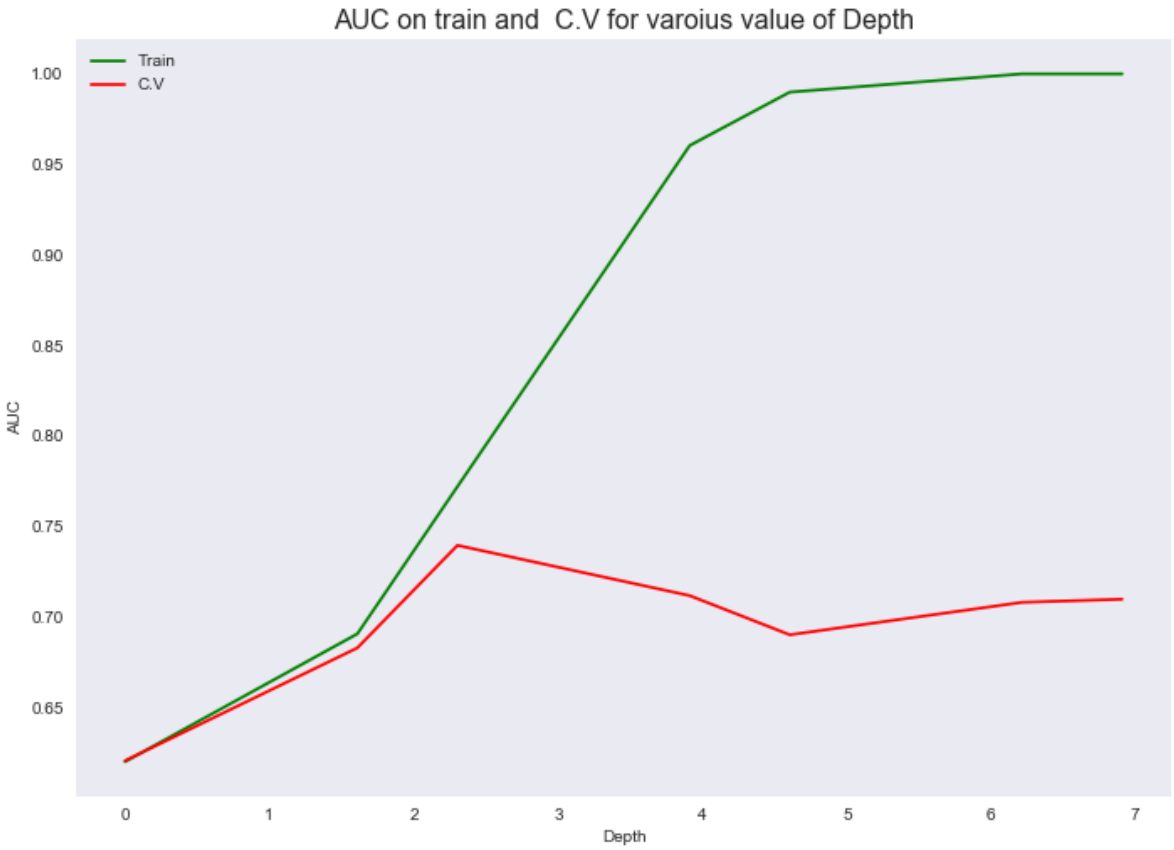
```
r=[]
import math
for p in (hyper):
    q=math.log(p)
    r.append(q)
print(r)
```

```
[0.0, 1.6094379124341003, 2.302585092994046, 3.912023005428146, 4.6051701859
88092, 6.214608098422191, 6.907755278982137]
```

In [72]:

```
#code for plotting graph
print(hyper)

print('----------------------------------------------------------------------------------------------------')
print(auc1)

print('----------------------------------------------------------------------------------------------------')
print('----------------------------------------------------------------------------------------------------')
print(auc2)

plt.title('AUC on train and  C.V for varoius value of Depth',size=16)
plt.plot(r, auc1,'g',label ='Train')
plt.plot(r, auc2,'r',label ='C.V')

plt.ylabel('AUC',size=10)
plt.xlabel('Depth',size=10)
plt.grid()
plt.legend()
plt.show()
```

```
[1, 5, 10, 50, 100, 500, 1000]
----------------------------------------------------------------------------
---------------
[0.6202093846519958, 0.6908179626156241, 0.7721076507853896, 0.9604385453177
386, 0.9899095646278049, 0.9999971066840552, 0.9999971066840552]
----------------------------------------------------------------------------
---------------
----------------------------------------------------------------------------
-------------
[0.6207518453890556, 0.6830452629415331, 0.7396634857645128, 0.7118924108050
412, 0.6902758434122364, 0.7081937862839643, 0.7099174856370652]
```



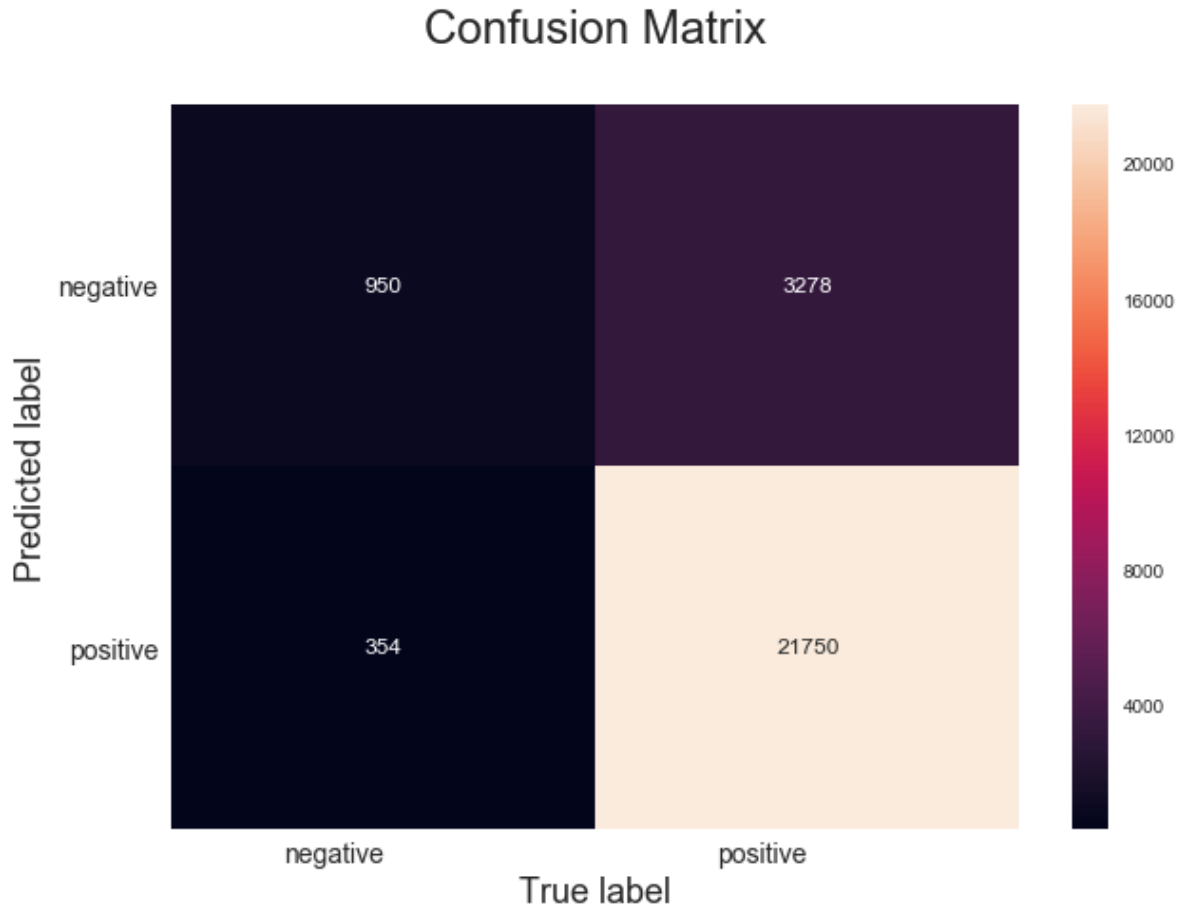
In [73]:

```
dtc = DecisionTreeClassifier(max_depth=10)

# fitting the model
dtc.fit(Xbow_tr_std, y_tr)

# predict the response
pred = dtc.predict(Xbow_test_std)

# evaluate accuracy
acc = accuracy_score(y_test, pred) * 100
print('\nThe accuracy of the Decision tree classifier for depth = %f is %f%%' % (10, acc))
```

```
The accuracy of the Decision tree classifier for depth = 10.000000 is 86.206
897%
```

In [74]:

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

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

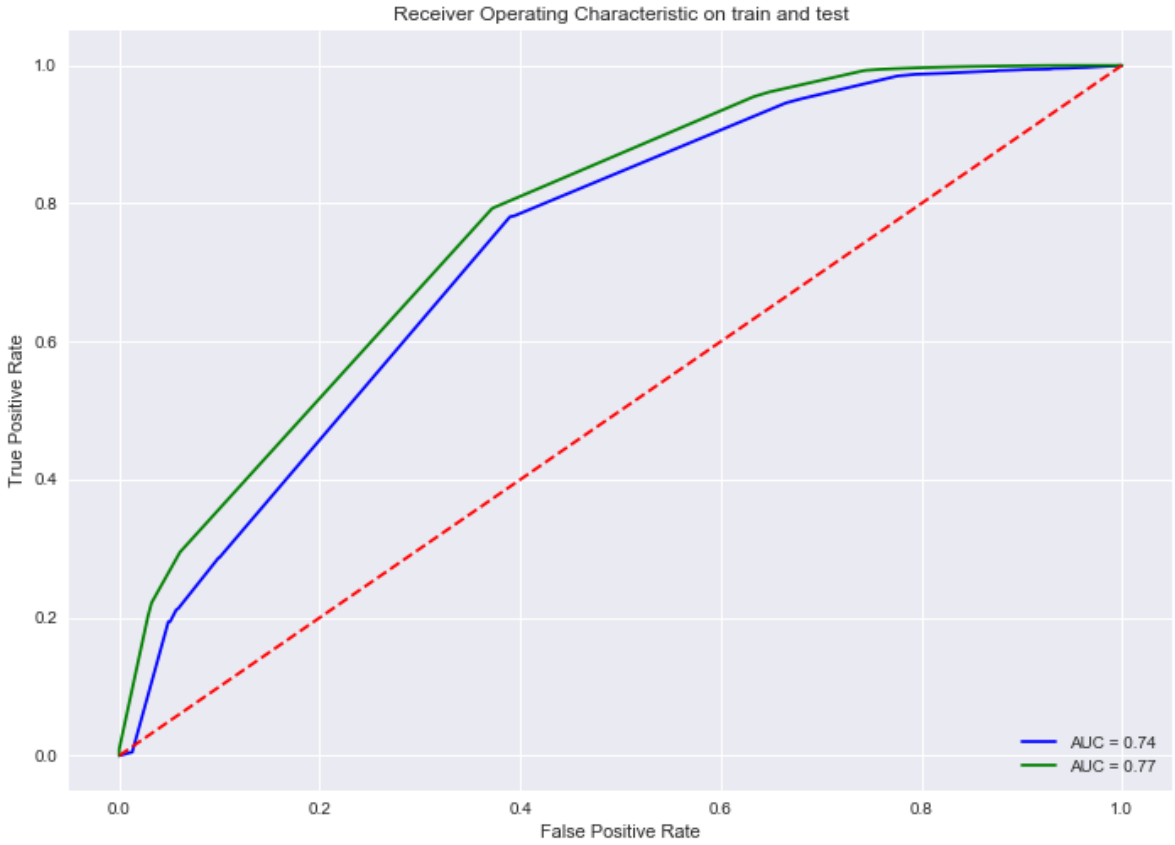


In [75]:

```
dtc.fit(Xbow_tr_std, y_tr)
probs2 = dtc.predict_proba(Xbow_tr_std)
preds2 = probs2[:,1]
fpr2, tpr2, threshold2 = metrics.roc_curve(y_tr, preds2)
roc_auc2 = metrics.auc(fpr2, tpr2)


probs1 = dtc.predict_proba(Xbow_test_std)
preds1 = probs1[:,1]
fpr1, tpr1, threshold1 = metrics.roc_curve(y_test, preds1)
roc_auc1 = metrics.auc(fpr1, tpr1)
```

In [76]:

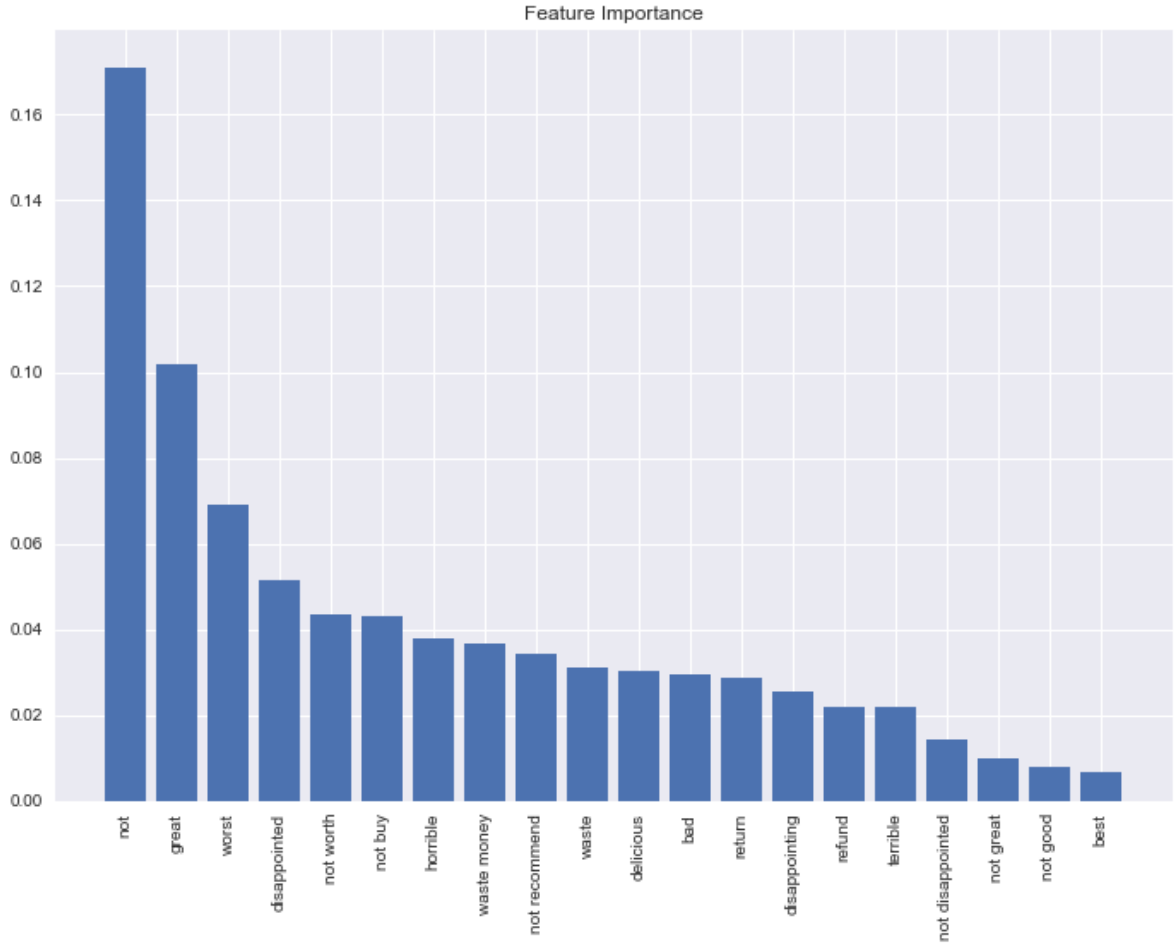
```
plt.title('Receiver Operating Characteristic on train and test')
plt.plot(fpr1, tpr1, 'b', label = 'AUC = %0.2f' % roc_auc1)
plt.plot(fpr2, tpr2, 'g', label = 'AUC = %0.2f' % roc_auc2)
#plt.plot(neighbors, auc1,'g')
#plt.plot(neighbors, auc2,'r')
plt.legend(loc = 'lower right')
plt.plot([0, 1], [0, 1],'r--')
plt.ylabel('True Positive Rate')
plt.xlabel('False Positive Rate')
plt.show()
```



[5.2.1] Top 20 important features from SET 2

In [77]:

```
#refer code snippet for top 20 feature extraction
#https://github.com/cyanamous/Amazon-Food-Reviews-Analysis-and-Modelling/blob/master/6%20Am
# Calculate feature importances from decision trees
importances = dtc.feature_importances_

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

# Rearrange feature names so they match the sorted feature importances
names = count_vect.get_feature_names()

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

# Create plot
plt.figure()

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

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

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

# Show plot
plt.show()
# uni_gram.get_feature_names()from sklearn import tree
```



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

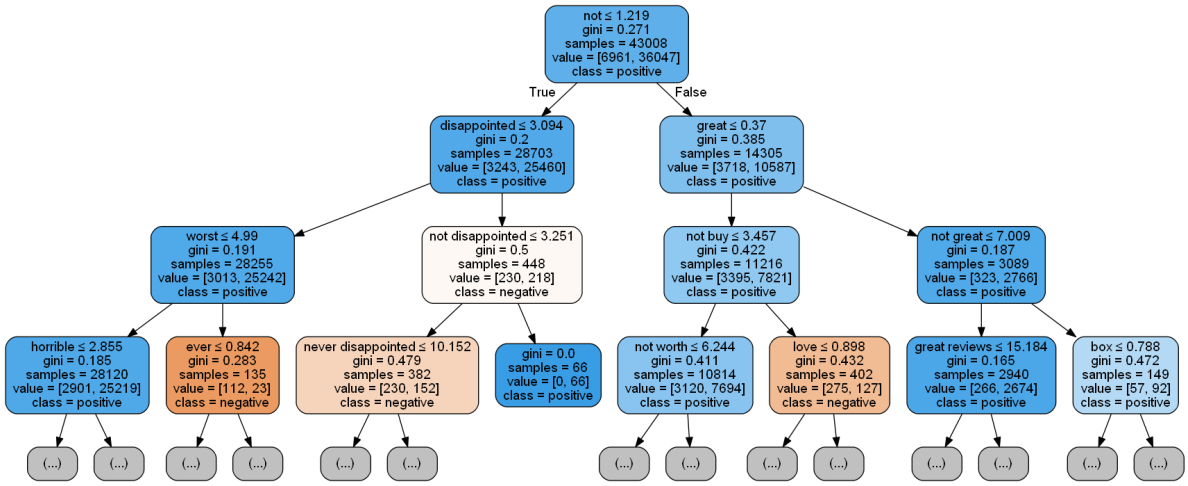
In [78]:

```
import pydotplus
from IPython.display import Image
from IPython.display import SVG
from graphviz import Source
from IPython.display import display
```

In [80]:

```
target = ['negative','positive']
# Create DOT data
data = tree.export_graphviz(dtc,out_file=None,class_names=target,feature_names=count_vect.g

# Draw graph
graph = pydotplus.graph_from_dot_data(data)

# Show graph
Image(graph.create_png())
```

Out[80]:



Code for hyperparameter tuning min_samples_split

In [82]:

```
import numpy as np
hyper =  [5, 10, 100, 500]

auc1=[]
auc2=[]

for j in hyper:

    model = DecisionTreeClassifier(min_samples_split=j)
    model.fit(Xbow_tr_std, y_tr)

    probs = model.predict_proba(Xbow_tr_std)
    preds = probs[:,1]
    roc_auc1=metrics.roc_auc_score(y_tr, preds)
    auc1.append(roc_auc1)


    probs = model.predict_proba(Xbow_cv_std)
    preds = probs[:,1]
    roc_auc2=metrics.roc_auc_score(y_cv, preds)
    auc2.append(roc_auc2)
```

In [83]:

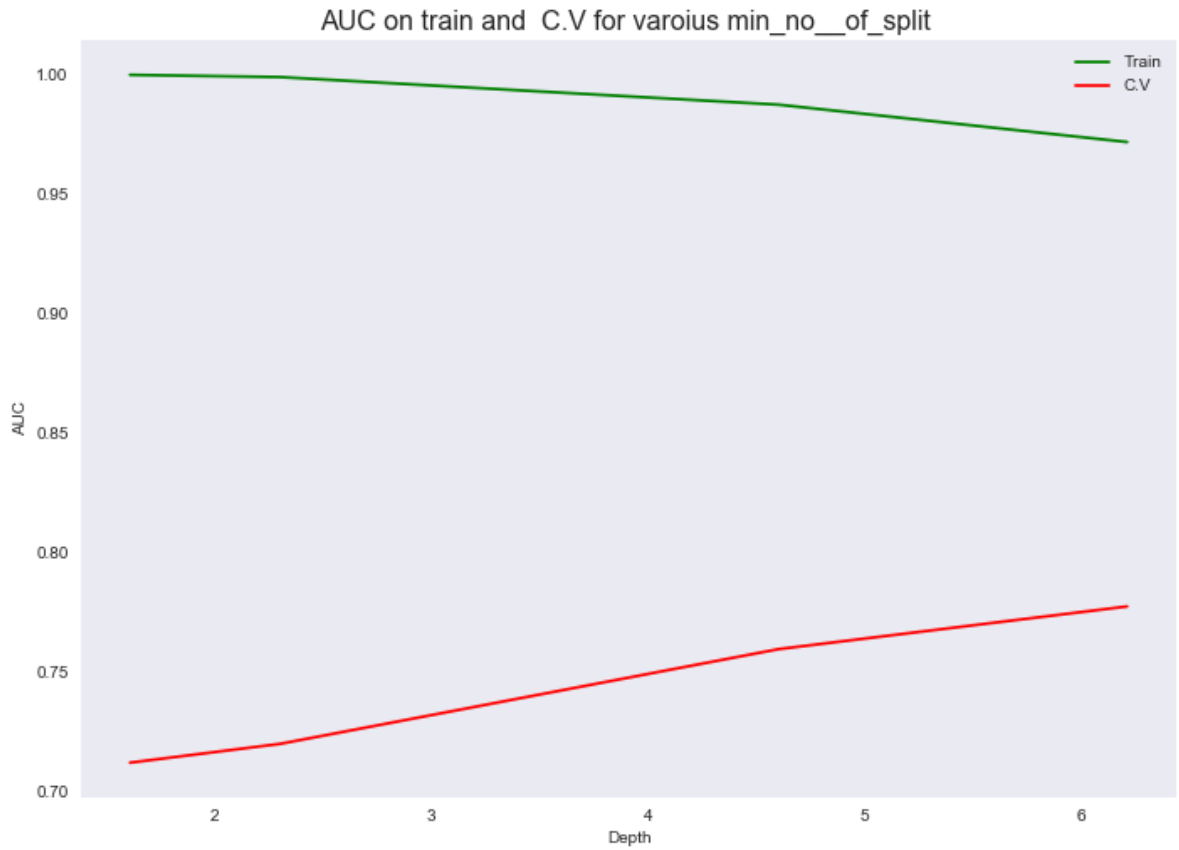
```
r2=[]
import math
for p in (hyper):
    q=math.log(p)
    r2.append(q)
print(r)
```

```
[0.0, 1.6094379124341003, 2.302585092994046, 3.912023005428146, 4.6051701859
88092, 6.214608098422191, 6.907755278982137]
```

In [84]:

```
#code for plotting graph
print(hyper)

print('------------------------------------------------------------------------------------------------------------------------')
print(auc1)

print('------------------------------------------------------------------------------------------------------------------------')
print('------------------------------------------------------------------------------------------------------------------------')
print(auc2)

plt.title('AUC on train and  C.V for varoius min_no__of_split ',size=16)
plt.plot(r2, auc1,'g',label ='Train')
plt.plot(r2, auc2,'r',label ='C.V')

plt.ylabel('AUC',size=10)
plt.xlabel('Depth',size=10)
plt.grid()
plt.legend()
plt.show()
```

```
[5, 10, 100, 500]
------------------------------------------------------------------------------------------
----------------
[0.9997740961877785, 0.9988489922893409, 0.9872868414736691, 0.9716509795207
551]
------------------------------------------------------------------------------------------
----------------
------------------------------------------------------------------------------------------
--------------
[0.7117056878918443, 0.7195930182925034, 0.7592736233002837, 0.7771401586580
904]
```



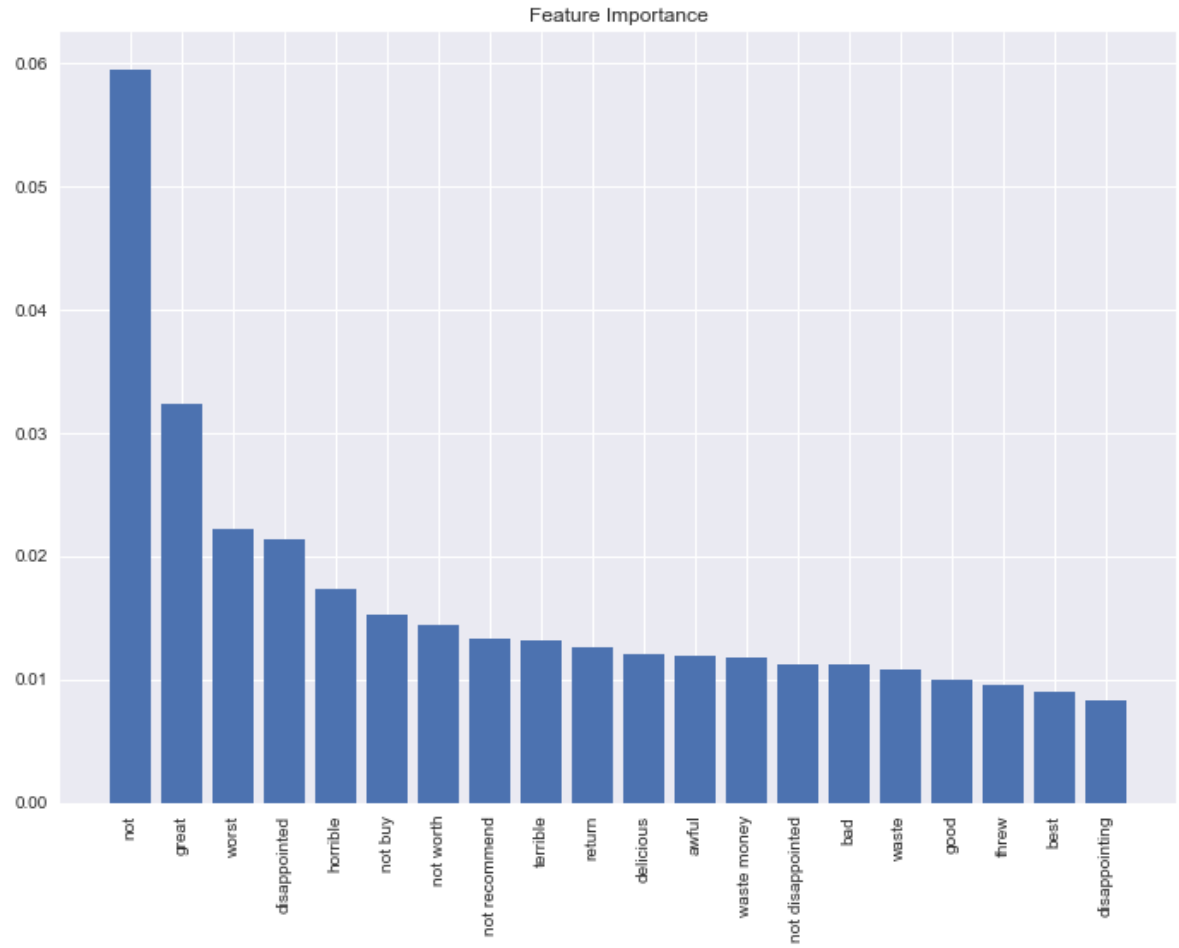
In [85]:

```
dtc = DecisionTreeClassifier(min_samples_split=100)

# fitting the model
dtc.fit(Xbow_tr_std, y_tr)

# predict the response
pred = dtc.predict(Xbow_test_std)

# evaluate accuracy
acc = accuracy_score(y_test, pred) * 100
print('\nThe accuracy of the Decision tree classifier for split = %f is %f%%' % (100, acc))
```

```
The accuracy of the Decision tree classifier for split = 100.000000 is 84.74
8595%
```

In [86]:

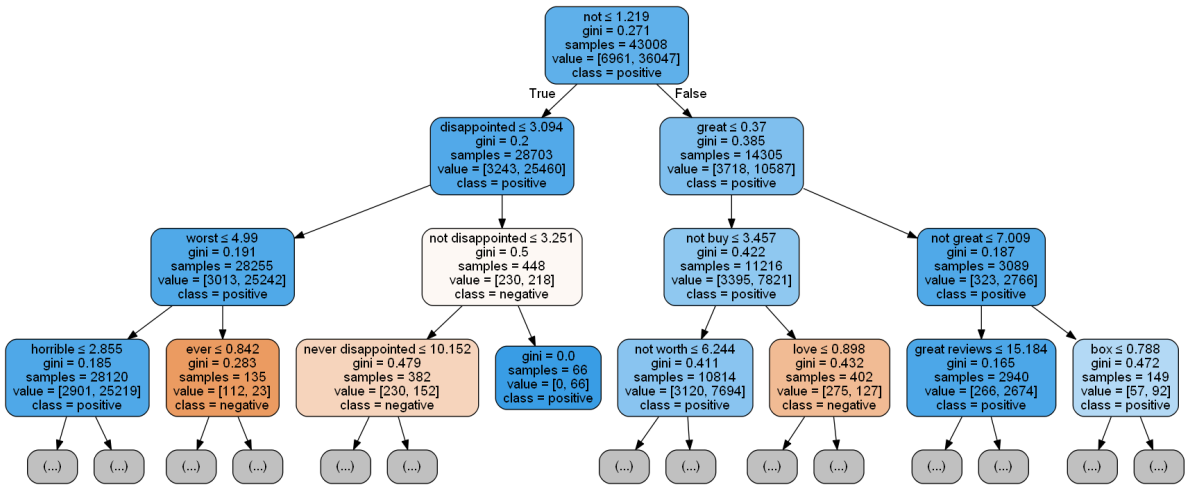
```
#refer code snippet for top 20 feature extraction
#https://github.com/cyanamous/Amazon-Food-Reviews-Analysis-and-Modelling/blob/master/6%20Am
# Calculate feature importances from decision trees
importances = dtc.feature_importances_

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

# Rearrange feature names so they match the sorted feature importances
names = count_vect.get_feature_names()

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

# Create plot
plt.figure()

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

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

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

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



In [87]:

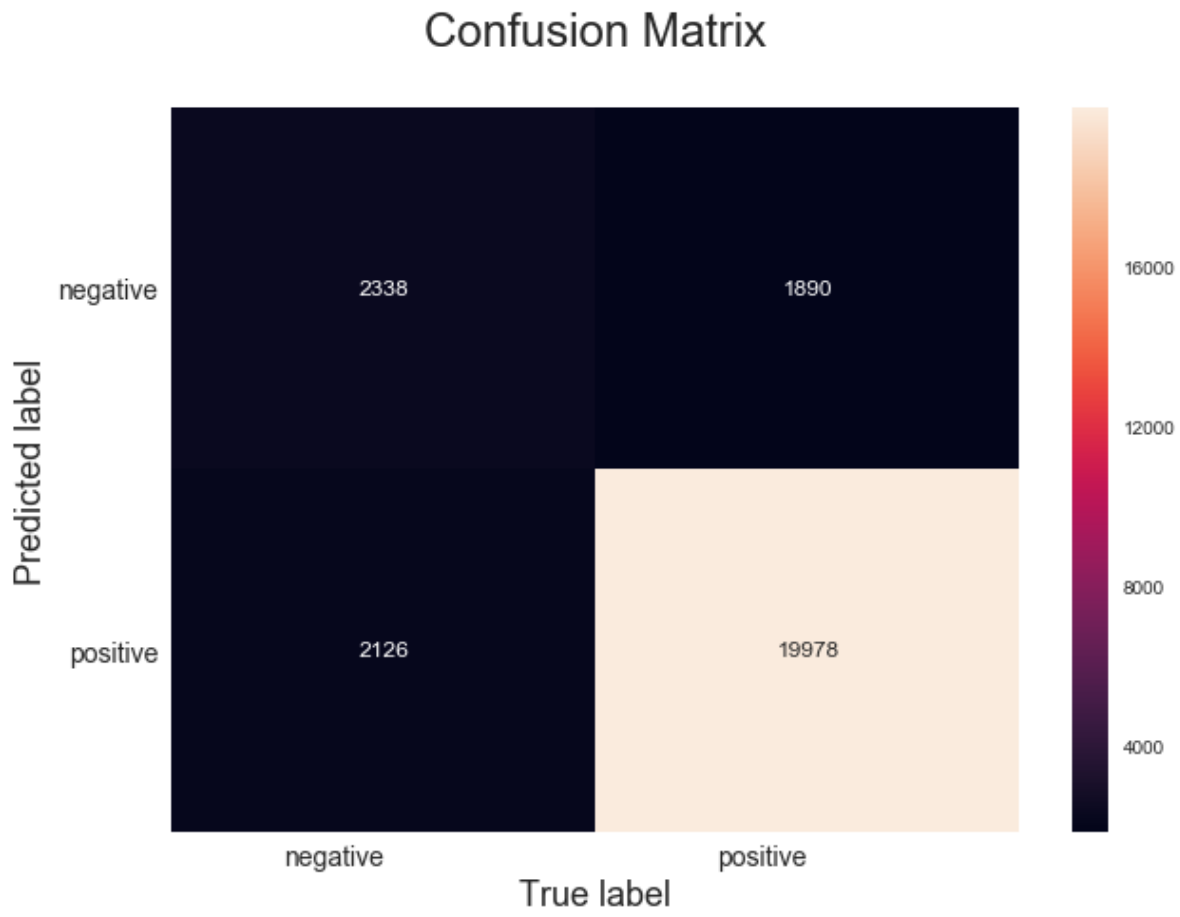
```
target = ['negative','positive']
# Create DOT data
data = tree.export_graphviz(dtc,out_file=None,class_names=target,feature_names=count_vect.g

# Draw graph
graph = pydotplus.graph_from_dot_data(data)

# Show graph
Image(graph.create_png())
```

Out[87]:



In [88]:

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

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

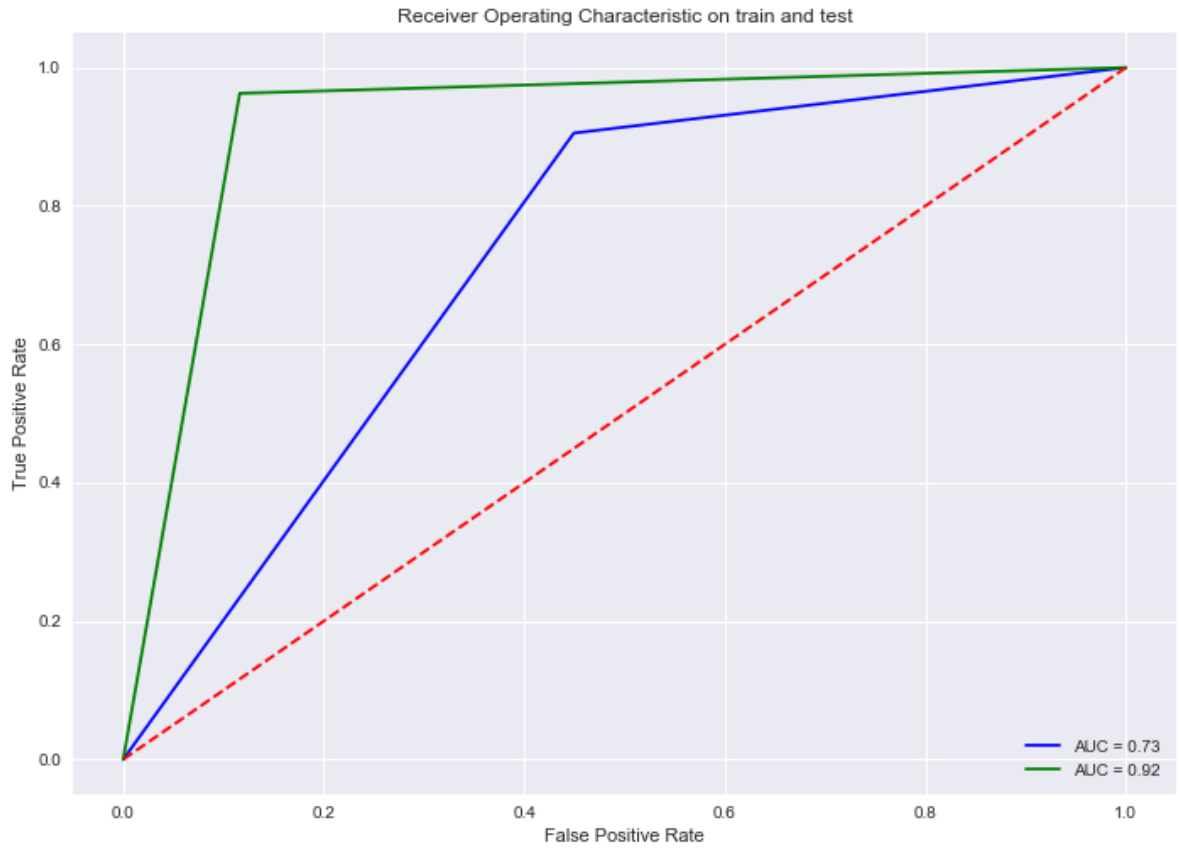


In [95]:

```
dtc = DecisionTreeClassifier(min_samples_split=100)

# fitting the model
dtc.fit(Xbow_tr, y_tr)
probs2 = dtc.predict(Xbow_tr)
preds2 = probs2
fpr2, tpr2, threshold2 = metrics.roc_curve(y_tr, preds2)
roc_auc2 = metrics.auc(fpr2, tpr2)


probs1 = dtc.predict(Xbow_test)
preds1 = probs1
fpr1, tpr1, threshold1 = metrics.roc_curve(y_test, preds1)
roc_auc1 = metrics.auc(fpr1, tpr1)
```

In [96]:

```
plt.title('Receiver Operating Characteristic on train and test')
plt.plot(fpr1, tpr1, 'b', label = 'AUC = %0.2f' % roc_auc1)
plt.plot(fpr2, tpr2, 'g', label = 'AUC = %0.2f' % roc_auc2)
#plt.plot(neighbors, auc1,'g')
#plt.plot(neighbors, auc2,'r')
plt.legend(loc = 'lower right')
plt.plot([0, 1], [0, 1],'r--')
plt.ylabel('True Positive Rate')
plt.xlabel('False Positive Rate')
plt.show()
```



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

In [97]:

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

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

    sent_of_cv=[]
for sent in X_cv:
    sent_of_cv.append(sent.split())

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

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

```
number of words that occured minimum 5 times  12647
```

In [98]:

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

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


    Xbow_cv = [];
for sent in sent_of_cv:
    sent_vec = np.zeros(50)
    cnt_words =0;
    for word in sent: #
        if word in w2v_words:
            vec = w2v_model.wv[word]
            sent_vec += vec
            cnt_words += 1
    if cnt_words != 0:
        sent_vec /= cnt_words
    Xbow_cv.append(sent_vec)
```

In [99]:

```
import warnings
warnings.filterwarnings('ignore')
# Data-preprocessing: Standardizing the data

from sklearn.preprocessing import StandardScaler
sc = StandardScaler(with_mean=False)
Xbow_tr_std = sc.fit_transform(Xbow_tr)
Xbow_test_std = sc.transform(Xbow_test)
Xbow_cv_std = sc.fit_transform(Xbow_cv)
```

Code for hyperparameter tuning Depth

In [100]:

```
#code for hyperparameter tuning
import numpy as np
hyper =  [1, 5, 10, 50, 100, 500, 1000]

auc1=[]
auc2=[]

for j in hyper:

    model = DecisionTreeClassifier(max_depth=j)
    model.fit(Xbow_tr_std, y_tr)

    probs = model.predict_proba(Xbow_tr_std)
    preds = probs[:,1]
    roc_auc1=metrics.roc_auc_score(y_tr, preds)
    auc1.append(roc_auc1)


    probs = model.predict_proba(Xbow_cv_std)
    preds = probs[:,1]
    roc_auc2=metrics.roc_auc_score(y_cv, preds)
    auc2.append(roc_auc2)
```

In [101]:

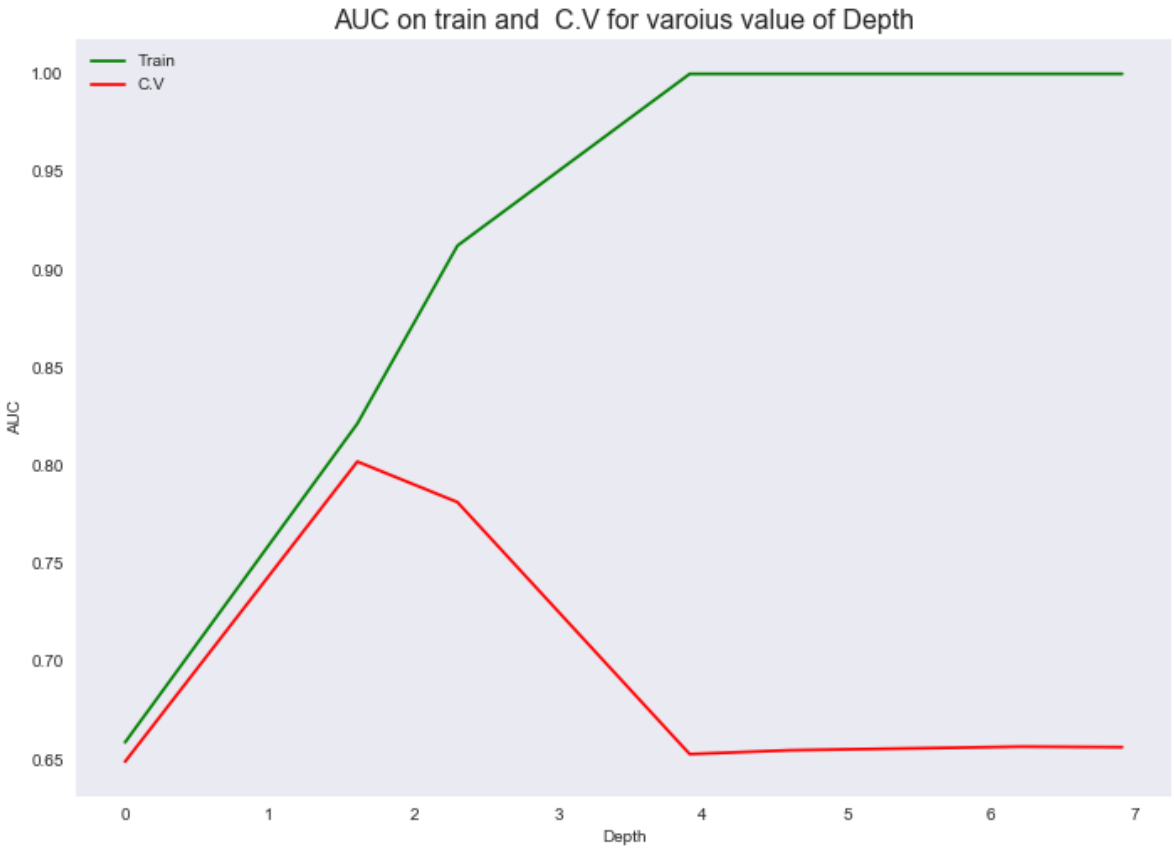
```
r=[]
import math
for p in (hyper):
    q=math.log(p)
    r.append(q)
print(r)
```

```
[0.0, 1.6094379124341003, 2.302585092994046, 3.912023005428146, 4.6051701859
88092, 6.214608098422191, 6.907755278982137]
```

In [102]:

```
#code for plotting graph
print(hyper)

print('-------------------------------------------------------------------------------------------------')
print(auc1)

print('-------------------------------------------------------------------------------------------------')
print('-------------------------------------------------------------------------------------------------')
print(auc2)

plt.title('AUC on train and  C.V for varoius value of Depth',size=16)
plt.plot(r, auc1,'g',label ='Train')
plt.plot(r, auc2,'r',label ='C.V')

plt.ylabel('AUC',size=10)
plt.xlabel('Depth',size=10)
plt.grid()
plt.legend()
plt.show()
```

```
[1, 5, 10, 50, 100, 500, 1000]
-------------------------------------------------------------------------------
---------------
[0.6586823746728815, 0.8214450102967177, 0.912258904734771, 0.99999713258839
91, 0.9999971325883991, 0.9999971325883991, 0.9999971325883991]
-------------------------------------------------------------------------------
---------------
-------------------------------------------------------------------------------
-------------
[0.6488042980231463, 0.8020315145593007, 0.7813157366820123, 0.6526632598744
851, 0.6546330442910688, 0.656482177978138, 0.6562132571558393]
```



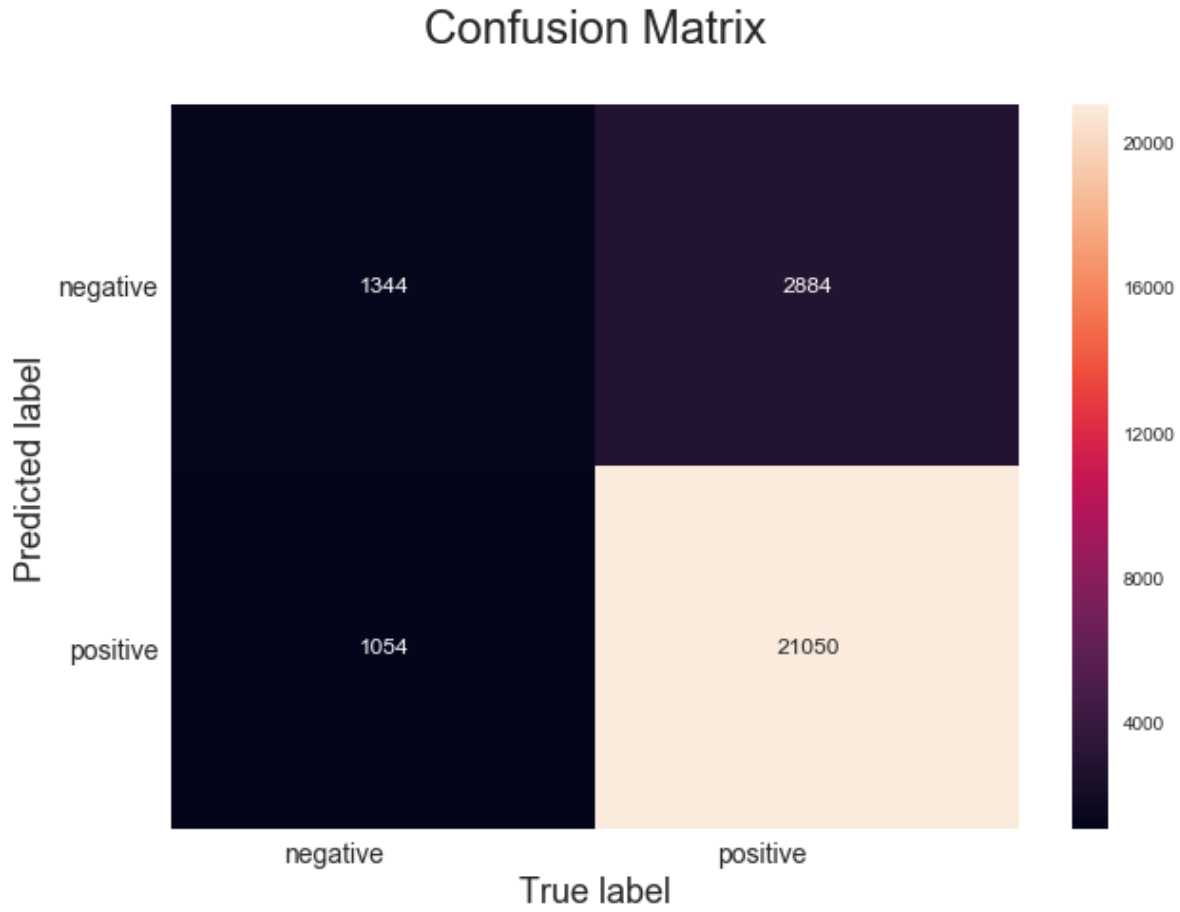
In [103]:

```
dtc = DecisionTreeClassifier(max_depth=5)

# fitting the model
dtc.fit(Xbow_tr_std, y_tr)

# predict the response
pred = dtc.predict(Xbow_test_std)

# evaluate accuracy
acc = accuracy_score(y_test, pred) * 100
print('\nThe accuracy of the Decision tree classifier for depth = %f is %f%%' % (5, acc))
```

```
The accuracy of the Decision tree classifier for depth = 5.000000 is 85.0448
12%
```

In [104]:

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

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

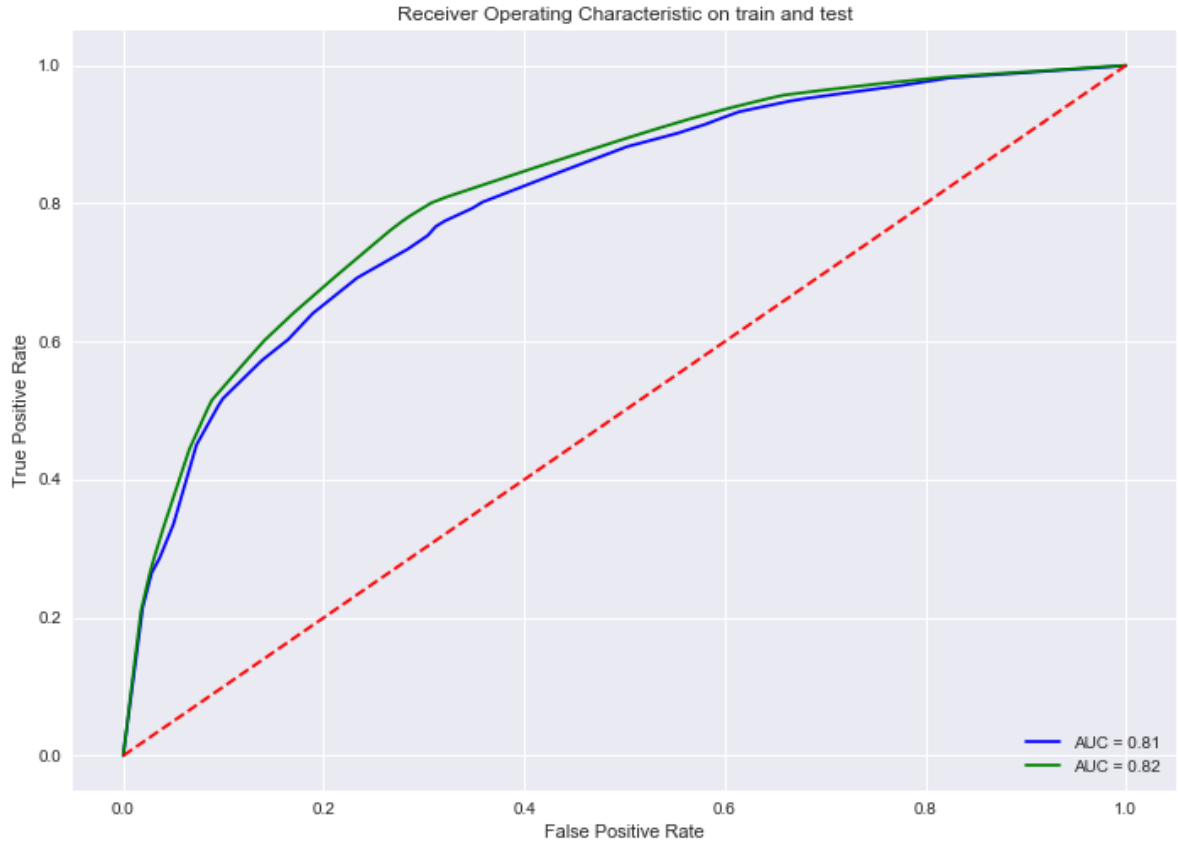


In [105]:

```
dtc.fit(Xbow_tr_std, y_tr)
probs2 = dtc.predict_proba(Xbow_tr_std)
preds2 = probs2[:,1]
fpr2, tpr2, threshold2 = metrics.roc_curve(y_tr, preds2)
roc_auc2 = metrics.auc(fpr2, tpr2)


probs1 = dtc.predict_proba(Xbow_test_std)
preds1 = probs1[:,1]
fpr1, tpr1, threshold1 = metrics.roc_curve(y_test, preds1)
roc_auc1 = metrics.auc(fpr1, tpr1)
```

In [106]:

```
plt.title('Receiver Operating Characteristic on train and test')
plt.plot(fpr1, tpr1, 'b', label = 'AUC = %0.2f' % roc_auc1)
plt.plot(fpr2, tpr2, 'g', label = 'AUC = %0.2f' % roc_auc2)
#plt.plot(neighbors, auc1,'g')
#plt.plot(neighbors, auc2,'r')
plt.legend(loc = 'lower right')
plt.plot([0, 1], [0, 1],'r--')
plt.ylabel('True Positive Rate')
plt.xlabel('False Positive Rate')
plt.show()
```



Code for hyperparameter tuning min_samples_split

In [109]:

```
#code for hyperparameter tuning
import numpy as np
hyper =  [5, 10, 100, 500]

auc1=[]
auc2=[]

for j in hyper:

    model = DecisionTreeClassifier(min_samples_split=j)
    model.fit(Xbow_tr_std, y_tr)

    probs = model.predict_proba(Xbow_tr_std)
    preds = probs[:,1]
    roc_auc1=metrics.roc_auc_score(y_tr, preds)
    auc1.append(roc_auc1)


    probs = model.predict_proba(Xbow_cv_std)
    preds = probs[:,1]
    roc_auc2=metrics.roc_auc_score(y_cv, preds)
    auc2.append(roc_auc2)
```

In [110]:

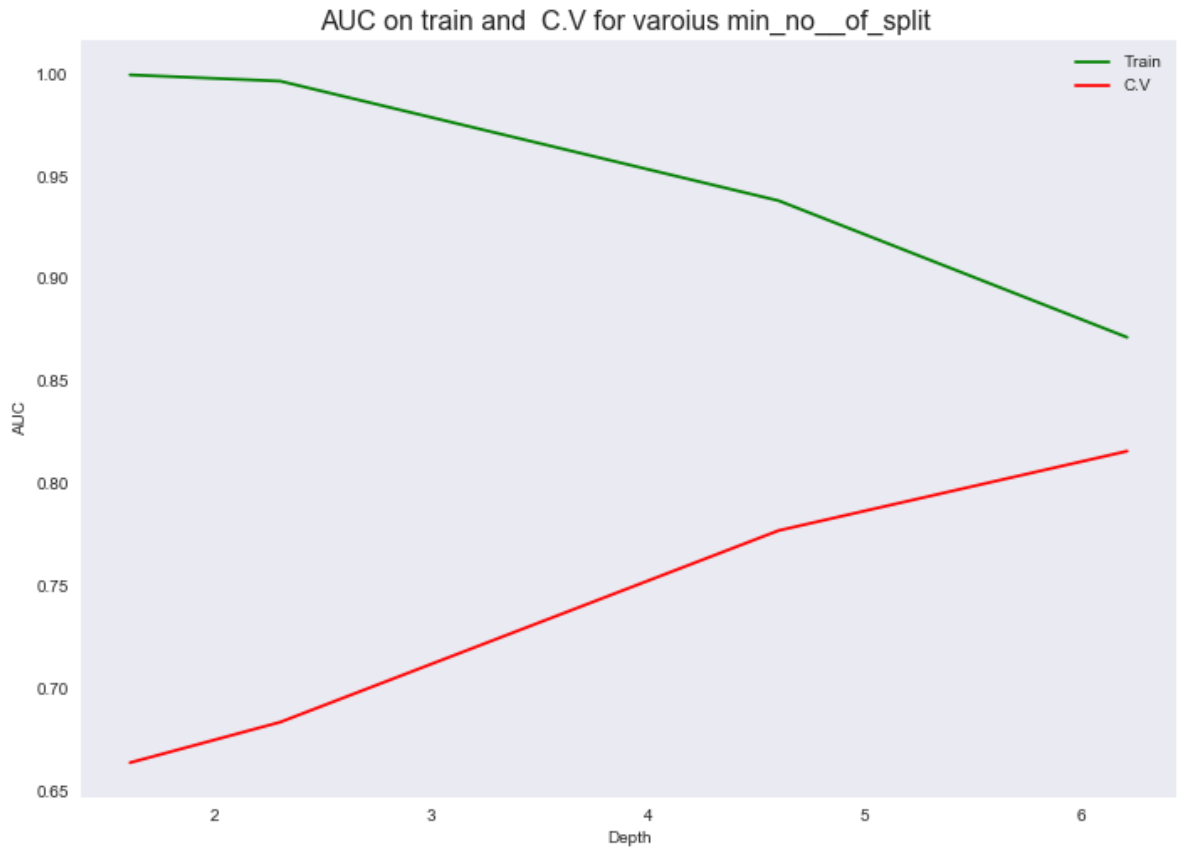
```
r2=[]
import math
for p in (hyper):
    q=math.log(p)
    r2.append(q)
print(r)
```

```
[0.0, 1.6094379124341003, 2.302585092994046, 3.912023005428146, 4.6051701859
88092, 6.214608098422191, 6.907755278982137]
```

In [111]:

```
#code for plotting graph
print(hyper)

print('------------------------------------------------------------------------------------------------------------------------')
print(auc1)

print('------------------------------------------------------------------------------------------------------------------------')
print('------------------------------------------------------------------------------------------------------------------------')
print(auc2)

plt.title('AUC on train and  C.V for varoius min_no__of_split ',size=16)
plt.plot(r2, auc1,'g',label ='Train')
plt.plot(r2, auc2,'r',label ='C.V')

plt.ylabel('AUC',size=10)
plt.xlabel('Depth',size=10)
plt.grid()
plt.legend()
plt.show()
```

```
[5, 10, 100, 500]
------------------------------------------------------------------------------------------------------------------------
[0.999497607169927, 0.9964515133829791, 0.9381290986973714, 0.8714990194588113]
------------------------------------------------------------------------------------------------------------------------
------------------------------------------------------------------------------------------------------------------------
[0.6640489960577904, 0.6838132262637114, 0.7772742943902042, 0.8159783189729961]
```



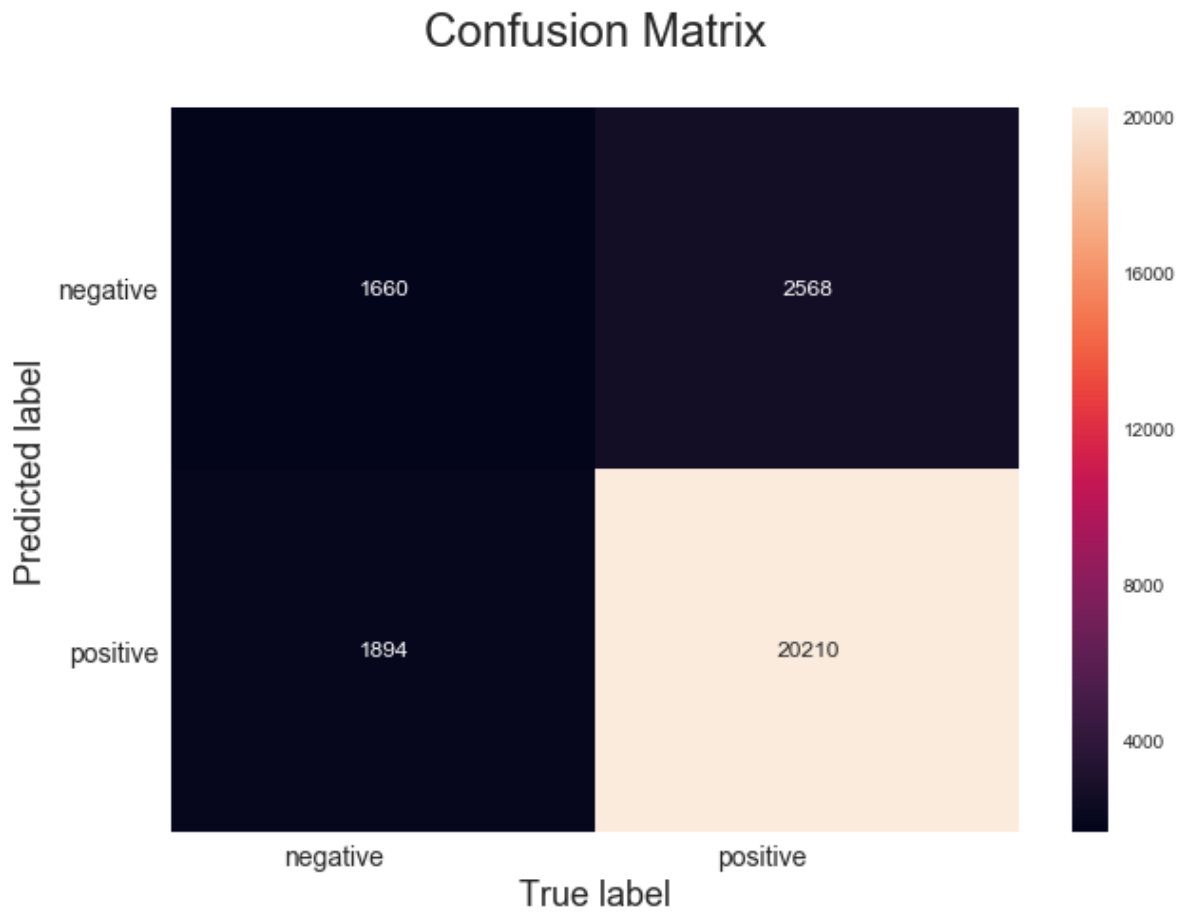
In [112]:

```
dtc = DecisionTreeClassifier(min_samples_split=100)

# fitting the model
dtc.fit(Xbow_tr_std, y_tr)

# predict the response
pred = dtc.predict(Xbow_test_std)

# evaluate accuracy
acc = accuracy_score(y_test, pred) * 100
print('\nThe accuracy of the Decision tree classifier for split = %f is %f%%' % (100, acc))
```

```
The accuracy of the Decision tree classifier for split = 100.000000 is 83.05
4838%
```

In [113]:

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

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

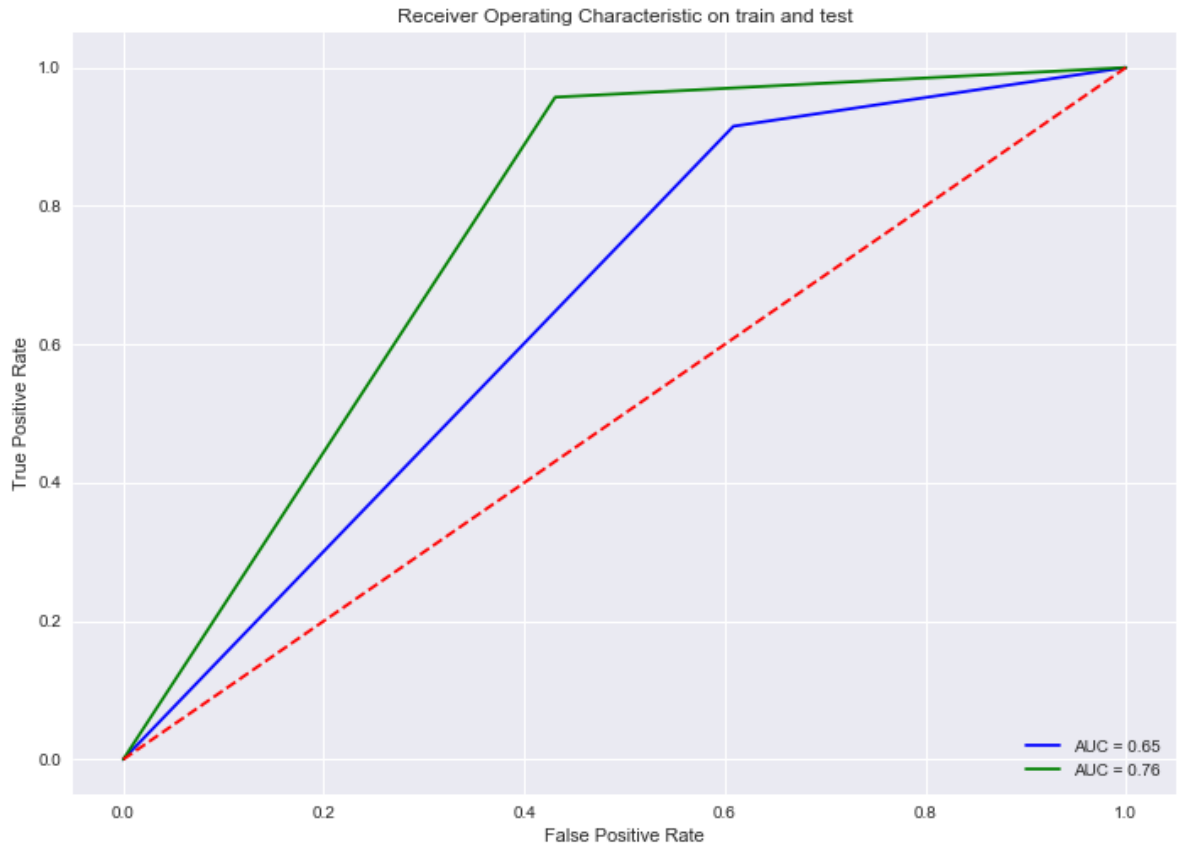


In [115]:

```
dtc = DecisionTreeClassifier(min_samples_split=100)

# fitting the model
dtc.fit(Xbow_tr, y_tr)
probs2 = dtc.predict(Xbow_tr)
preds2 = probs2
fpr2, tpr2, threshold2 = metrics.roc_curve(y_tr, preds2)
roc_auc2 = metrics.auc(fpr2, tpr2)


probs1 = dtc.predict(Xbow_test)
preds1 = probs1
fpr1, tpr1, threshold1 = metrics.roc_curve(y_test, preds1)
roc_auc1 = metrics.auc(fpr1, tpr1)
```

In [116]:

```
plt.title('Receiver Operating Characteristic on train and test')
plt.plot(fpr1, tpr1, 'b', label = 'AUC = %0.2f' % roc_auc1)
plt.plot(fpr2, tpr2, 'g', label = 'AUC = %0.2f' % roc_auc2)
#plt.plot(neighbors, auc1,'g')
#plt.plot(neighbors, auc2,'r')
plt.legend(loc = 'lower right')
plt.plot([0, 1], [0, 1],'r--')
plt.ylabel('True Positive Rate')
plt.xlabel('False Positive Rate')
plt.show()
```



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

In [117]:

```
# S = ["abc def pqr", "def def def abc", "pqr pqr def"]
model = TfidfVectorizer()
model.fit(X_tr)
# 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 [118]:

```
# 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

Xbow_tr = []; # the tfidf-w2v for each sentence/review is stored in this list
row=0;
for sent in sent_of_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
    Xbow_tr.append(sent_vec)
    row += 1
```

In [120]:

```
# S = ["abc def pqr", "def def def abc", "pqr pqr def"]
model = TfidfVectorizer()
model.fit(X_cv)
# 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 [121]:

```
# 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

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

In [122]:

```
# S = ["abc def pqr", "def def def abc", "pqr pqr def"]
model = TfidfVectorizer()
model.fit(X_test)
# 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 [123]:

```
# 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

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

In [124]:

```
import warnings
warnings.filterwarnings('ignore')
# Data-preprocessing: Standardizing the data

from sklearn.preprocessing import StandardScaler
sc = StandardScaler(with_mean=False)
Xbow_tr_std = sc.fit_transform(Xbow_tr)
Xbow_test_std = sc.transform(Xbow_test)
Xbow_cv_std = sc.fit_transform(Xbow_cv)
```

Code for hyperparameter tuning Depth

In [125]:

```
#code for hyperparameter tuning
import numpy as np
hyper =  [1, 5, 10, 50, 100, 500, 1000]

auc1=[]
auc2=[]

for j in hyper:

    model = DecisionTreeClassifier(max_depth=j)
    model.fit(Xbow_tr_std, y_tr)

    probs = model.predict_proba(Xbow_tr_std)
    preds = probs[:,1]
    roc_auc1=metrics.roc_auc_score(y_tr, preds)
    auc1.append(roc_auc1)


    probs = model.predict_proba(Xbow_cv_std)
    preds = probs[:,1]
    roc_auc2=metrics.roc_auc_score(y_cv, preds)
    auc2.append(roc_auc2)
```

In [126]:

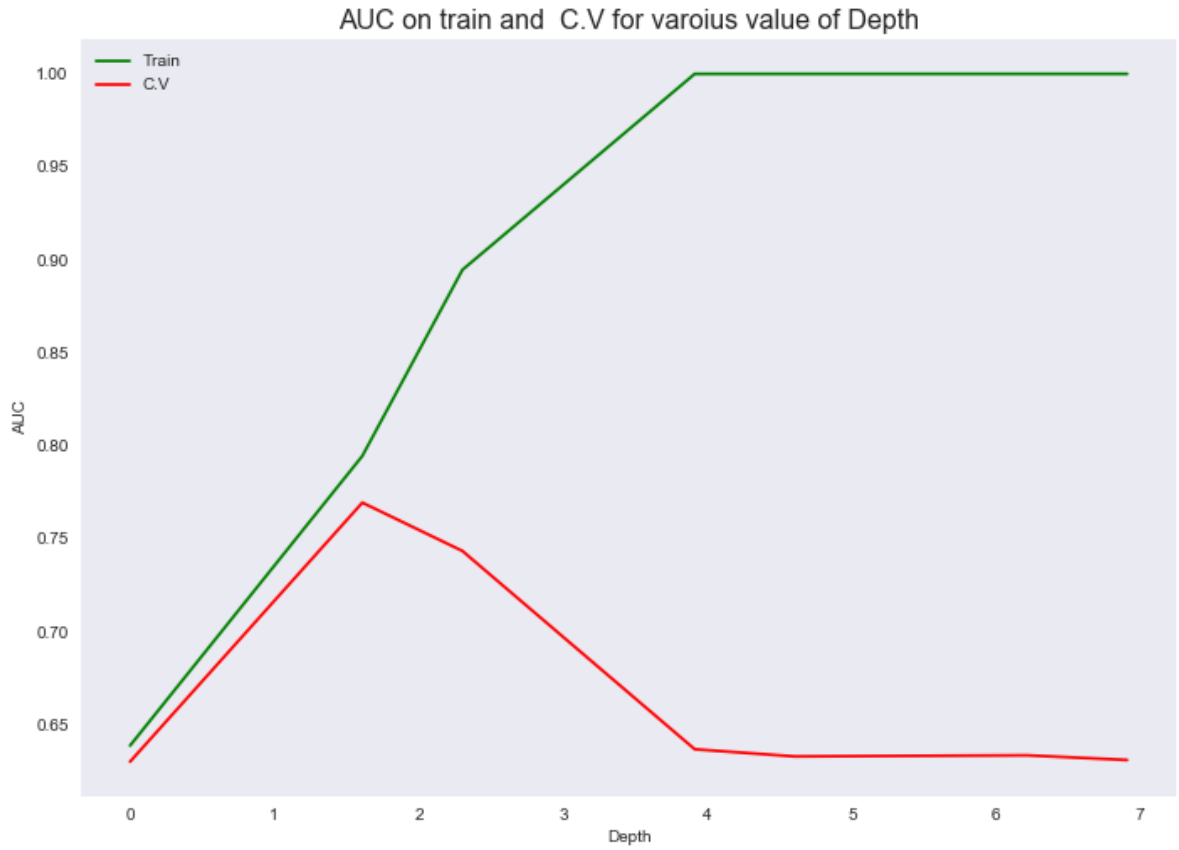
```
r=[]
import math
for p in (hyper):
    q=math.log(p)
    r.append(q)
print(r)
```

```
[0.0, 1.6094379124341003, 2.302585092994046, 3.912023005428146, 4.6051701859
88092, 6.214608098422191, 6.907755278982137]
```

In [127]:

```
#code for plotting graph
print(hyper)

print('----------------------------------------------------------------------------------------------------')
print(auc1)

print('----------------------------------------------------------------------------------------------------')
print('----------------------------------------------------------------------------------------------------')
print(auc2)

plt.title('AUC on train and  C.V for varoius value of Depth',size=16)
plt.plot(r, auc1,'g',label ='Train')
plt.plot(r, auc2,'r',label ='C.V')

plt.ylabel('AUC',size=10)
plt.xlabel('Depth',size=10)
plt.grid()
plt.legend()
plt.show()
```

```
[1, 5, 10, 50, 100, 500, 1000]
----------------------------------------------------------------------------
----------------
[0.6386995306814377, 0.7945263958030627, 0.8946010752367077, 0.9999971325883
991, 0.9999971325883991, 0.9999971325883991, 0.9999971325883991]
----------------------------------------------------------------------------
----------------
----------------------------------------------------------------------------
--------------
[0.6300543651234802, 0.7693806868615294, 0.7433891344039603, 0.6367446147436
491, 0.632879559749081, 0.6334814822126106, 0.6309637911013355]
```



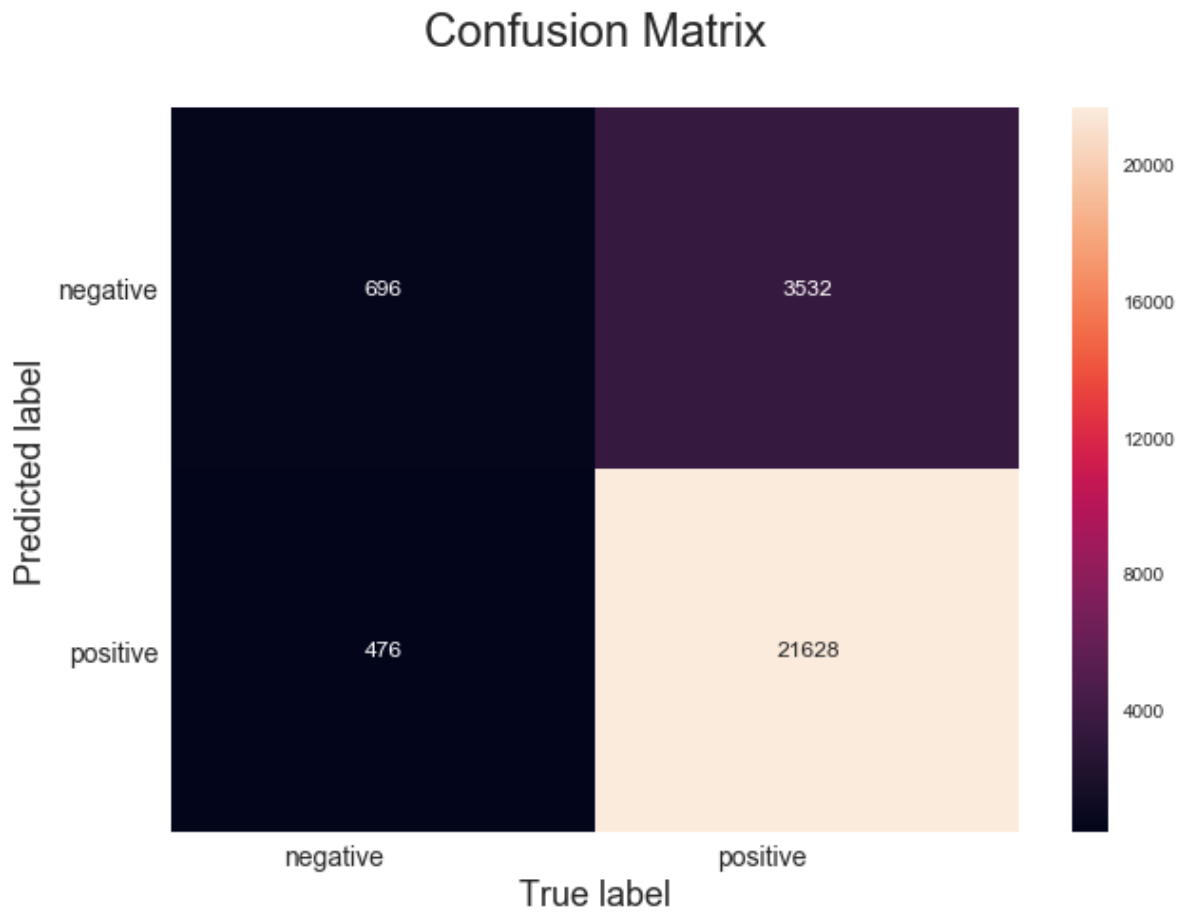
In [128]:

```
dtc = DecisionTreeClassifier(max_depth=5)

# fitting the model
dtc.fit(Xbow_tr_std, y_tr)

# predict the response
pred = dtc.predict(Xbow_test_std)

# evaluate accuracy
acc = accuracy_score(y_test, pred) * 100
print('\nThe accuracy of the Decision tree classifier for depth = %f is %f%%' % (5, acc))
```

```
The accuracy of the Decision tree classifier for depth = 5.000000 is 84.7789
76%
```

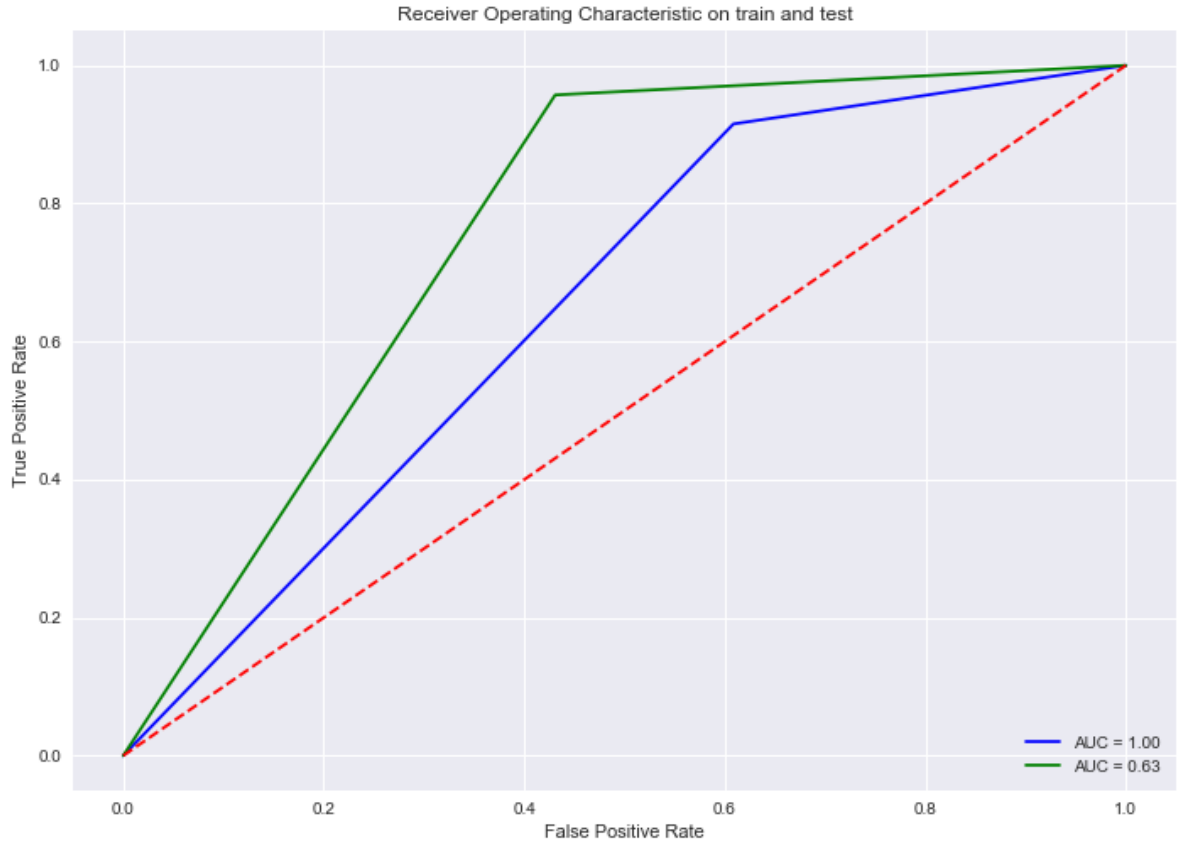
In [129]:

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

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



In [130]:

```
plt.title('Receiver Operating Characteristic on train and test')
plt.plot(fpr1, tpr1, 'b', label = 'AUC = %0.2f' % roc_auc1)
plt.plot(fpr2, tpr2, 'g', label = 'AUC = %0.2f' % roc_auc2)
#plt.plot(neighbors, auc1,'g')
#plt.plot(neighbors, auc2,'r')
plt.legend(loc = 'lower right')
plt.plot([0, 1], [0, 1],'r--')
plt.ylabel('True Positive Rate')
plt.xlabel('False Positive Rate')
plt.show()
```



Code for hyperparameter tuning min_samples_split

In [131]:

```
#code for hyperparameter tuning
import numpy as np
hyper =  [5, 10, 100, 500]

auc1=[]
auc2=[]

for j in hyper:

    model = DecisionTreeClassifier(min_samples_split=j)
    model.fit(Xbow_tr_std, y_tr)

    probs = model.predict_proba(Xbow_tr_std)
    preds = probs[:,1]
    roc_auc1=metrics.roc_auc_score(y_tr, preds)
    auc1.append(roc_auc1)


    probs = model.predict_proba(Xbow_cv_std)
    preds = probs[:,1]
    roc_auc2=metrics.roc_auc_score(y_cv, preds)
    auc2.append(roc_auc2)
```

In [132]:

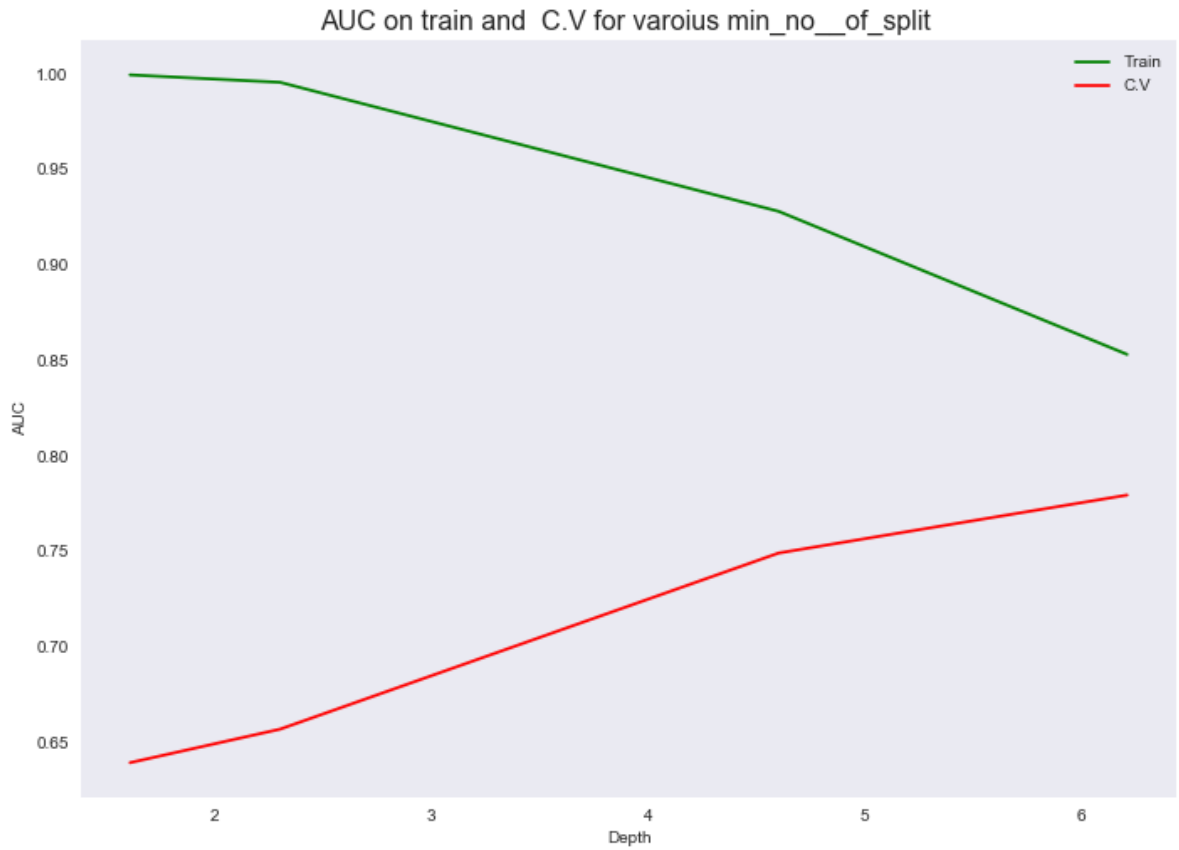
```
r2=[]
import math
for p in (hyper):
    q=math.log(p)
    r2.append(q)
print(r)
```

```
[0.0, 1.6094379124341003, 2.302585092994046, 3.912023005428146, 4.6051701859
88092, 6.214608098422191, 6.907755278982137]
```

In [133]:

```
#code for plotting graph
print(hyper)

print('------------------------------------------------------------------------------------------------------------------------')
print(auc1)

print('------------------------------------------------------------------------------------------------------------------------')
print('------------------------------------------------------------------------------------------------------------------------')
print(auc2)

plt.title('AUC on train and  C.V for varoius min_no__of_split ',size=16)
plt.plot(r2, auc1,'g',label ='Train')
plt.plot(r2, auc2,'r',label ='C.V')

plt.ylabel('AUC',size=10)
plt.xlabel('Depth',size=10)
plt.grid()
plt.legend()
plt.show()
```

```
[5, 10, 100, 500]
------------------------------------------------------------------------------------------
----------------
[0.9994420702493366, 0.9955411032254347, 0.9280016001870407, 0.8531200887481
227]
------------------------------------------------------------------------------------------
----------------
------------------------------------------------------------------------------------------
--------------
[0.6394125023766506, 0.6569640990702232, 0.7491440486592573, 0.7794846659719
402]
```



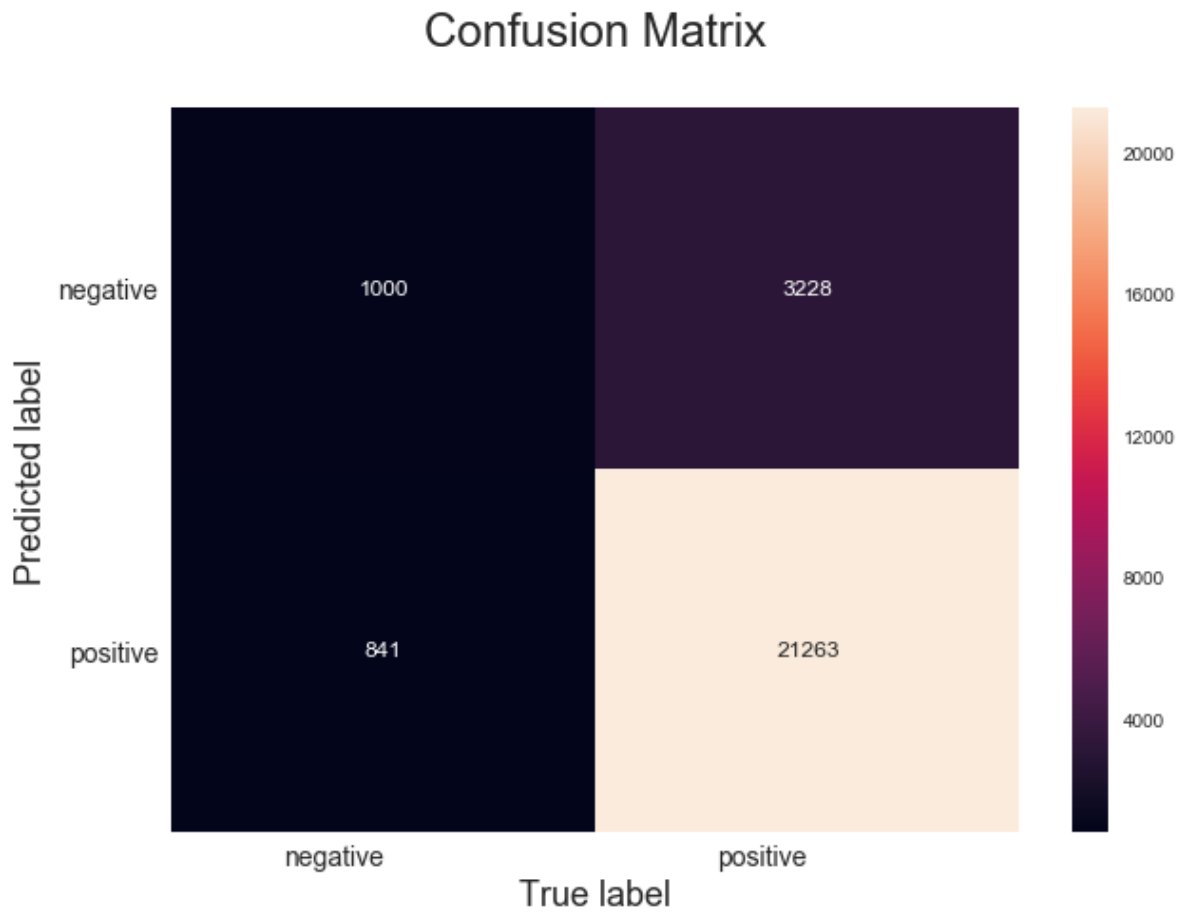
In [134]:

```
dtc = DecisionTreeClassifier(min_samples_split=500)

# fitting the model
dtc.fit(Xbow_tr_std, y_tr)

# predict the response
pred = dtc.predict(Xbow_test_std)

# evaluate accuracy
acc = accuracy_score(y_test, pred) * 100
print('\nThe accuracy of the Decision tree classifier for split = %f is %f%%' % (500, acc))
```

```
The accuracy of the Decision tree classifier for split = 500.000000 is 84.54
7319%
```

In [136]:

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

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

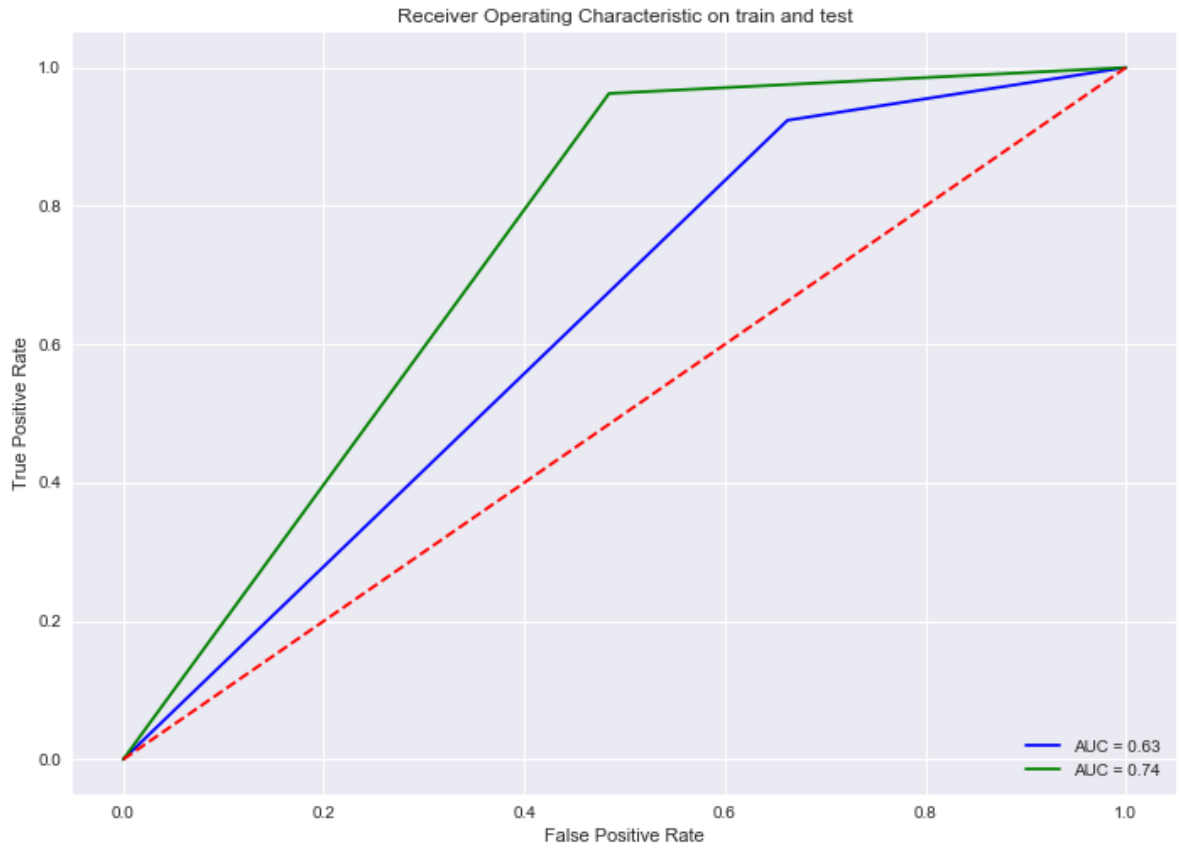


In [137]:

```
dtc = DecisionTreeClassifier(min_samples_split=100)

# fitting the model
dtc.fit(Xbow_tr, y_tr)
probs2 = dtc.predict(Xbow_tr)
preds2 = probs2
fpr2, tpr2, threshold2 = metrics.roc_curve(y_tr, preds2)
roc_auc2 = metrics.auc(fpr2, tpr2)


probs1 = dtc.predict(Xbow_test)
preds1 = probs1
fpr1, tpr1, threshold1 = metrics.roc_curve(y_test, preds1)
roc_auc1 = metrics.auc(fpr1, tpr1)
```

In [138]:

```
plt.title('Receiver Operating Characteristic on train and test')
plt.plot(fpr1, tpr1, 'b', label = 'AUC = %0.2f' % roc_auc1)
plt.plot(fpr2, tpr2, 'g', label = 'AUC = %0.2f' % roc_auc2)
#plt.plot(neighbors, auc1,'g')
#plt.plot(neighbors, auc2,'r')
plt.legend(loc = 'lower right')
plt.plot([0, 1], [0, 1],'r--')
plt.ylabel('True Positive Rate')
plt.xlabel('False Positive Rate')
plt.show()
```



[6] Conclusions

In [135]:

```
from prettytable import PrettyTable

x = PrettyTable()

x.field_names = ["Vactorizer", "Model","Hyperparameter_name","Hyperparameter_value", "AUC%"

x.add_row(["BOW", "Decision Tree","Depth",10,86.073978])

x.add_row(["BOW", "Decision Tree","Min _sample_Split",100,84.205529])

x.add_row(["TFIDF", "Decision Tree","Depth",10,86.206897])

x.add_row(["TFIDF", "Decision Tree","Min _sample_Split",100,84.748595])

x.add_row(["AVGW2V", "Decision Tree","Depth",5,85.044812])

x.add_row(["AVGW2V", "Decision Tree","Min _sample_Split",100,83.054838])

x.add_row(["TFIDFW2V","Decision Tree","Depth",5,84.778976])

x.add_row(["TFIDFW2V","Decision Tree", "Min _sample_Split",500,84.547319])

print(x)
```

```
+------------+---------------+---------------------+----------------------+-
----------+
| Vactorizer |     Model     | Hyperparameter_name | Hyperparameter_value |
   AUC%   |
+------------+---------------+---------------------+----------------------+-
----------+
|    BOW     | Decision Tree |        Depth        |          10          |
86.073978 |
|    BOW     | Decision Tree |  Min _sample_Split  |         100          |
84.205529 |
|   TFIDF    | Decision Tree |        Depth        |          10          |
86.206897 |
|   TFIDF    | Decision Tree |  Min _sample_Split  |         100          |
84.748595 |
|   AVGW2V   | Decision Tree |        Depth        |          5           |
85.044812 |
|   AVGW2V   | Decision Tree |  Min _sample_Split  |         100          |
83.054838 |
|  TFIDFW2V  | Decision Tree |        Depth        |          5           |
84.778976 |
|  TFIDFW2V  | Decision Tree |  Min _sample_Split  |         500          |
84.547319 |
+------------+---------------+---------------------+----------------------+-
----------+
```

------->> Step followered:-

STEP 1 :- Data cleaning (removing duplication).

STEP 2 :- Text Preprocessing.

STEP 3:- Featurization on text reviews i.e BOW,TFIDF,avgW2V,TFIDF-W2V.

STEP 4:- Standardization on vectors i.e BOW,TFIDF,avgW2V,TFIDF-W2V.

STEP 5:- Using AUC as a metric and plot curve for Train(predected value on itself) and C.V predected value on train.

STEP 6:- Plot "AUC VS Depth" to analise overfitting and underfitting.

STEP 7:- Once , we analise optimal value of Depth then we train the MODEL again with this analised optimal Depth and make predictions on test_data.

STEP 8:- Compute test accuracy using predicted values of test_data.

STEP 9:- Plot Seaborn Heatmap for representation of Confusion Matrix on different SVM model.

STEP 10:- Plot ROC curve for train and test on different Decision tree model.

STEP 11:- Plot "AUC VS Min_sample_split" to analise overfitting and underfitting.

STEP 12:- Once , we analise optimal value ofno_of_splits then we train the MODEL again with this analised optimal no_of_splits make and predictions on test_data.

STEP 13:- Compute test accuracy using predicted values of test_data.

STEP 14:- Plot Seaborn Heatmap for representation of Confusion Matrix on different SVM model.

STEP 15:- Plot ROC curve for train and test on different Decision tree model.

----->>>> Repeat from STEP 4 to STEP 15 for each of these four vectorizers : Bag Of Words(BoW), TFIDF , Avg Word2Vec and TFIDF Word2Vec saparately on Decision tree.

------>>>> We extract the top 20 important features from SET1 and SET2 i.e BOW AND TFIDF vectorizer in Decision tree.

------>>>> We Visualize our decision tree upto depth 3 using Graphviz from SET1 and SET2 i.e BOW AND TFIDF vectorizer in Decision tree.

---------->>>>> AT THE END WE MAKE A TABLE TO COMPAIR OUR RESULTS OF Decision tree with different vectorizers and hyperparameter(depth and no_of splits).

In [ ]: