since the data is already preprocessed and savedonto disc, I would be working on the preprocessed data and wud be implementing DT of different NLP vectorizers

DT - BoW

In [1]:
```
#importing required packages

import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
import seaborn as sns
import warnings
import scipy as sc
import sympy
import datetime
#import date
import os
import sys
import graphviz
%matplotlib inline
warnings.filterwarnings('ignore')
```

importing performance metric libraries

In [2]:
```
#importing Logistic regression libraries:
from sklearn.tree import DecisionTreeClassifier

#train test split libaries:
from sklearn.preprocessing import StandardScaler
from sklearn.model_selection import train_test_split

#importing performance libraries:
```

```
from sklearn.metrics import f1_score
from sklearn.metrics import confusion_matrix
from sklearn.metrics import precision_score
from sklearn.metrics import precision_recall_curve
from sklearn.metrics import roc_curve
from sklearn.metrics import roc_auc_score
from sklearn.metrics import auc
```

Imporing preprocessed cleaned data from database file

In [3]:
```
%%time
import sqlite3
con = sqlite3.connect('/home/reachjalesh/PreprocessingFolder/final_1L.s
qlite')
df = pd.read_sql_query("""select * from reviews""", con)
```

```
CPU times: user 1.21 s, sys: 372 ms, total: 1.58 s
Wall time: 3.21 s
```

In [6]:
```
len(df)
```

Out[6]:
```
100000
```

In [7]:
```
df.columns
```

Out[7]:
```
Index(['index', 'Id', 'ProductId', 'UserId', 'ProfileName',
       'HelpfulnessNumerator', 'HelpfulnessDenominator', 'Score', 'Tim
e',
       'Summary', 'Text', 'CleanedText_Bow', 'ClenedText_W2Vtfdf', 'Bow
_feat',
       'Bow_new_feat', 'w2v_feat', 'w2v_new_feat'],
      dtype='object')
```

In [8]:
```
df.Bow_new_feat.head(2)
```

Out[8]:
```
0    everi book educ witti littl book make son laug...
1    whole seri great way spend time child rememb s...
Name: Bow_new_feat, dtype: object
```

In [9]:
```
df.Bow_feat.head(2)
```

Out[9]:
```
0                           everi book educ
1    whole seri great way spend time child
Name: Bow_feat, dtype: object
```

In [10]:
```
df.CleanedText_Bow.head(2)
```

Out[10]:
```
0    witti littl book make son laugh loud recit car...
1    rememb see show air televis year ago child sis...
Name: CleanedText_Bow, dtype: object
```

In [11]:
```
vectorizer = ['bow']
for i in vectorizer:
    print('unfeatured preprocesed column for vectorizer: {} is: \n {}'.
format(i, df.CleanedText_Bow.head(1)))
    print()
    print('featured preprocesed column for vectorizer: {} is: \n {}'.fo
rmat(i, df.Bow_new_feat.head(1)))
```

```
unfeatured preprocesed column for vectorizer: bow is:
 0    witti littl book make son laugh loud recit car...
Name: CleanedText_Bow, dtype: object

featured preprocesed column for vectorizer: bow is:
 0    everi book educ witti littl book make son laug...
Name: Bow_new_feat, dtype: object
```

In [12]:
```
vectorizer = ['tfidf', 'avg-w2v', 'tfidf-w2v']
for i in vectorizer:
    print('UNFEATURED COLUMN for vectorizer: {} is: \n {}'.format(i, df
.Text.head(1)))
    print()
    print('FEATURED COLUMN for vectorizer: {} is: \n {}'.format(i, df.C
lenedText_W2Vtfdf.head(1)))
```

```
UNFEATURED COLUMN for vectorizer: tfidf is:
 0    this witty little book makes my son laugh at l...
```

```
Name: Text, dtype: object

FEATURED COLUMN for vectorizer: tfidf is:
 0    witty little book makes son laugh loud recite ...
Name: ClenedText_W2Vtfdf, dtype: object
UNFEATURED COLUMN for vectorizer: avg-w2v is:
 0    this witty little book makes my son laugh at l...
Name: Text, dtype: object

FEATURED COLUMN for vectorizer: avg-w2v is:
 0    witty little book makes son laugh loud recite ...
Name: ClenedText_W2Vtfdf, dtype: object
UNFEATURED COLUMN for vectorizer: tfidf-w2v is:
 0    this witty little book makes my son laugh at l...
Name: Text, dtype: object

FEATURED COLUMN for vectorizer: tfidf-w2v is:
 0    witty little book makes son laugh loud recite ...
Name: ClenedText_W2Vtfdf, dtype: object
```

In [13]:

```
for i in df.ClenedText_W2Vtfdf.head(1):
    print(i)
print('\n' * 2)
for i in df.Text.head(1):
    print(i)
```

```
witty little book makes son laugh loud recite car driving along always
sing refrain learned whales india drooping love new words book introduc
es silliness classic book willing bet son still able recite memory coll
ege



this witty little book makes my son laugh at loud  i recite it in the c
ar as we re driving along and he always can sing the refrain  he s lear
ned about whales  india  drooping roses:  i love all the new words this
book  introduces and the silliness of it all   this is a classic book i
am  willing to bet my son will still be able to recite from memory when
he is  in college
```

sorting dataframe based on time

In [14]:
```
#sorting the datframe based on time:
print(len(df))
df = df.sort_values('Time', ascending=True)
print()
df['Time'].head(8)
```

```
100000
```

Out[14]:
```
0    939340800
1    940809600
2    944092800
3    944438400
4    946857600
5    947376000
6    948240000
7    948672000
Name: Time, dtype: int64
```

In [15]:
```
len(df)
```

Out[15]:
```
100000
```

In [16]:
```
df.columns
```

Out[16]:
```
Index(['index', 'Id', 'ProductId', 'UserId', 'ProfileName',
       'HelpfulnessNumerator', 'HelpfulnessDenominator', 'Score', 'Tim
e',
       'Summary', 'Text', 'CleanedText_Bow', 'ClenedText_W2Vtfdf', 'Bow
_feat',
       'Bow_new_feat', 'w2v_feat', 'w2v_new_feat'],
      dtype='object')
```

In [17]:
```
df.Score.value_counts()
```

Out[17]:
```
1    87729
```

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

train test split

In [18]:
```
xtrain, xtest, ytrain, ytest = train_test_split(df.Bow_new_feat, df.Score, test_size=0.2, shuffle=False)
xtr, xcv, ytr, ycv = train_test_split(xtrain, ytrain, test_size=0.2, shuffle=False)
```

BoW object instantiation

In [19]:
```
from sklearn.feature_extraction.text import CountVectorizer

bow_object = CountVectorizer(ngram_range=(1,1))

xtr = bow_object.fit_transform(xtr)
xcv = bow_object.transform(xcv)
xtest = bow_object.transform(xtest)


print(xtr.shape)
print(xcv.shape)
print(xtest.shape)
```

```
(64000, 31265)
(16000, 31265)
(20000, 31265)
```

1. DT on BoW

In [20]:
```
class decisiontree:

    '''building the decision tree classifier based off various parameters'''
```

```
    #instantiating the instance attributes:
    def __init__(self, xtr, ytr, xcv, ycv, minimum_splits=[5,10,100,500
], maximum_depth=[1, 5, 10, 50, 100, 500, 100]):
        self.xtr = xtr
        self.ytr = ytr
        self.xcv = xcv
        self.ycv = ycv
        self.minimum_splits = minimum_splits
        self.maximum_depth = maximum_depth

    #creating a method of calling DT classifier:
    def classfier(self, auc_dict_cv={}, auc_dict_tr={}):
        for splits in self.minimum_splits:
            for depths in self.maximum_depth:
                clf = DecisionTreeClassifier(max_depth=depths, min_samp
les_split=splits)
                print(depths, splits)
                clf.fit(self.xtr, self.ytr)
                y_pred_cv = clf.predict_proba(self.xcv)

                #performance metric on CV data:
                fpr_cv, tpr_cv, thresholds_cv = roc_curve(ycv, y_pred_c
v[:,1])
                auc_val = auc(fpr_cv, tpr_cv)
                auc_dict_cv[zip([splits], [depths])] = auc_val

                #performance metrics for training data:
                y_pred_tr = clf.predict_proba(self.xtr)
                fpr_tr, tpr_tr, thresholds_tr = roc_curve(ytr, y_pred_t
r[:,1])
                auc_val = auc(fpr_tr, tpr_tr)
                auc_dict_tr[zip([splits], [depths])] = auc_val

        return auc_dict_tr, auc_dict_cv
```

BoW decision tree instance creation on training

and CV data

In [21]:
```
%time
BoW_instance = decisiontree(xtr, ytr, xcv, ycv)

dictionary_train, dictionary_cv = BoW_instance.classfier()
```

```
CPU times: user 0 ns, sys: 0 ns, total: 0 ns
Wall time: 5.48 µs
1 5
5 5
10 5
50 5
100 5
500 5
100 5
1 10
5 10
10 10
50 10
100 10
500 10
100 10
1 100
5 100
10 100
50 100
100 100
500 100
100 100
1 500
5 500
10 500
50 500
100 500
500 500
100 500
```

In [22]:
```
train_list = [(list(x), np.round(y,3)) for x, y in dictionary_train.ite
ms()]
cv_list = [(list(x), np.round(y,3)) for x, y in dictionary_cv.items()]
```

In [23]:
```
print(train_list[0])

print()
print(cv_list[0])
```

```
([(500, 10)], 0.816)

([(5, 5)], 0.757)
```

sorting the list based off AUC score

In [24]:
```
tr_list = sorted(train_list, key=lambda x: x[1], reverse=True)
cv_list = sorted(cv_list, key=lambda x: x[1], reverse=True)
print('sorted train list score based off AUC score is:\n', tr_list[0:3
])
print('****************************************')
print('sorted train list score based off AUC score is:\n', cv_list[0:3
])
```

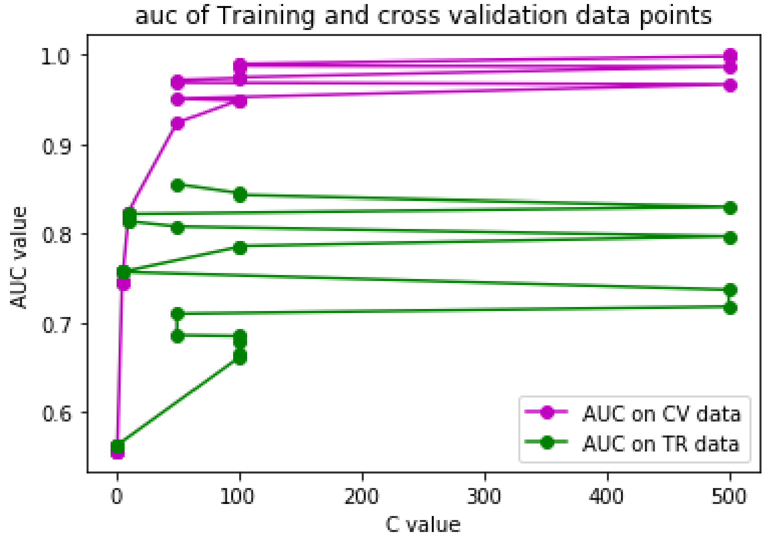
```
sorted train list score based off AUC score is:
 [([(5, 500)], 1.0), ([(10, 500)], 0.998), ([(5, 100)], 0.99)]
****************************************
sorted train list score based off AUC score is:
 [([(500, 50)], 0.855), ([(500, 100)], 0.845), ([(500, 100)], 0.843)]
```

Plotting AUC Curve on training and cv data based off depth

In [25]:
```
plt.plot([x[0][0][1] for x in tr_list], [x[1] for x in tr_list], linest
yle='-', color='m', marker='o',label='AUC on CV data')
plt.plot([x[0][0][1] for x in cv_list], [x[1] for x in cv_list], linest
yle='-', color='g', marker='o', label='AUC on TR data')
#plt.plot([0, 1], [0, 1], linestyle='--')
plt.xlabel("C value")
```

```
plt.ylabel('AUC value')
plt.title('auc of Training and cross validation data points')
plt.legend()
```

Out[25]: <matplotlib.legend.Legend at 0x7f0e3bbdc668>



OptimalBoW - DecesionTree[depth = 50 and splits = 500]

In [26]:

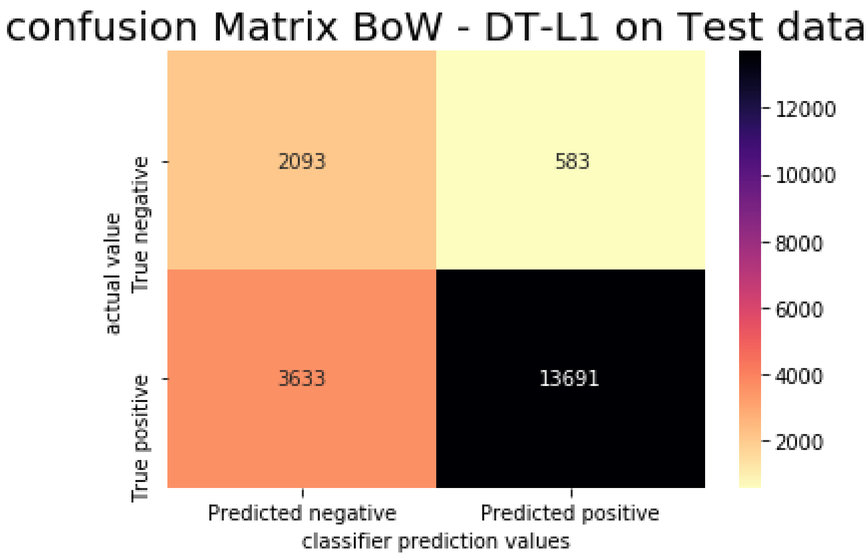
```
dt = DecisionTreeClassifier(max_depth=50, min_samples_split=500, class_
weight='balanced')
dt.fit(xtr, ytr)

ypred = dt.predict_proba(xtest)
y_pred_tr = dt.predict_proba(xtr)

fpr_test, tpr_test, thresholds_test= roc_curve(ytest, ypred[:,1])
auc_test = auc(fpr_test, tpr_test)

print(auc_test)
```

```
0.8465095382508484
```

plotting confusion matrix on test data

In [27]:
```
%time
ypred = np.where(ypred[:,1] < 0.5, 0, 1)
#creating confusion matrix:

cf = confusion_matrix(ytest, ypred)
labels = ['True negative', 'True positive']

df_cf = pd.DataFrame(cf, index=labels, columns=['Predicted negative',
'Predicted positive'])
sns.heatmap(df_cf, annot=True,fmt='3d', cmap='magma_r')

plt.title("confusion Matrix BoW - DT-L1 on Test data", size=20)
plt.xlabel("classifier prediction values")
plt.ylabel("actual value")
plt.show()
```

```
CPU times: user 0 ns, sys: 0 ns, total: 0 ns
Wall time: 5.72 µs
```

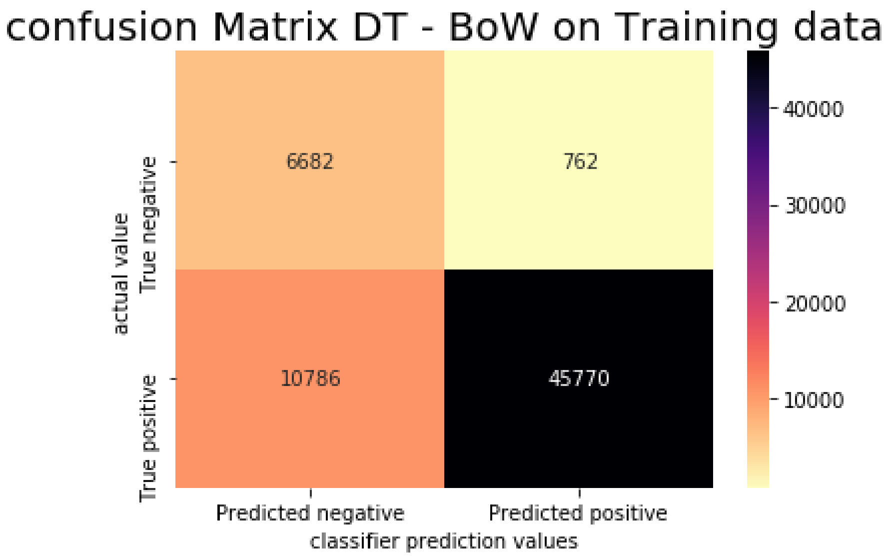


plotting confusion matrix on training data

In [28]:

```
%time
y_pred_tr = np.where(y_pred_tr[:,1] < 0.5, 0, 1)
#creating confusion matrix:

cf = confusion_matrix(ytr, y_pred_tr)
labels = ['True negative', 'True positive']

df_cf = pd.DataFrame(cf, index=labels, columns=['Predicted negative',
'Predicted positive'])
sns.heatmap(df_cf, annot=True,fmt='3d', cmap='magma_r')

plt.title("confusion Matrix DT - BoW on Training data", size=20)
plt.xlabel("classifier prediction values")
plt.ylabel("actual value")
plt.show()
```

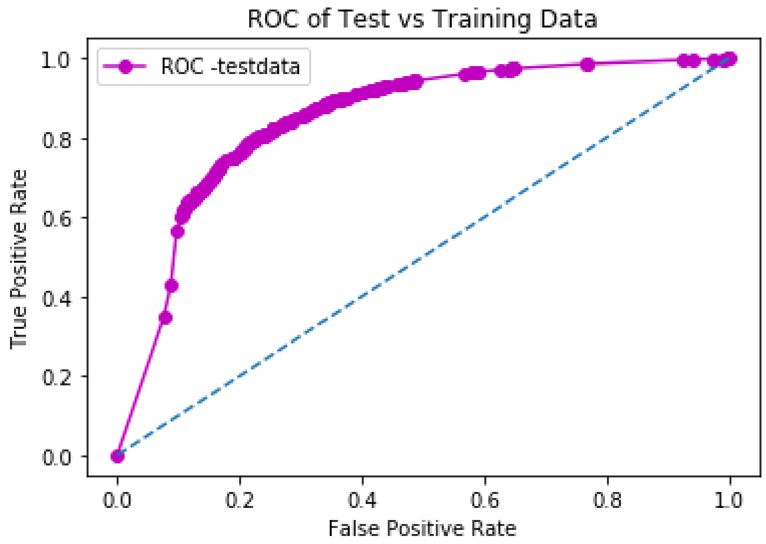
```
CPU times: user 0 ns, sys: 0 ns, total: 0 ns
Wall time: 5.72 µs
```



plotting ROC curve on test data

In [29]:
```
plt.plot(fpr_test, tpr_test, color='m', marker='o',label='ROC -testdat
a')
plt.plot([0, 1], [0, 1], linestyle='--')
#plt.plot(fpr_tr, tpr_tr, linestyle='-', color='g', marker='o', label
='ROC - training data')
plt.xlabel('False Positive Rate')
plt.ylabel('True Positive Rate')
plt.title('ROC of Test vs Training Data')
plt.legend()
```

Out[29]: <matplotlib.legend.Legend at 0x7f0e3cab8dd8>



printing top 25 features of both positive and negative class

In [30]:
```
features = bow_object.get_feature_names()
featuresAndcoeff = sorted(zip(dt.feature_importances_, features))
top_features = zip(featuresAndcoeff[:25],featuresAndcoeff[:-(25+1): -1]
 )
print('\t\t\t\tNegative\t\t\t\t\t\tPositive')
print('-' * 120)

for (wn1, fn1), (wp1, fp1) in top_features:
    print('{:>20} {:>20}                          {:>20} {:>20}'.
format(wn1, fn1, wp1, fp1))
```

```
                                Negative
                                Positive
------------------------------------------------------------------------
------------------------------------------------
                 0.0                  aaa
   0.13948093745846474                great
                 0.0        aaaaaaaaagghh
   0.13001810316688217                  not
```

```
                 0.0           aaaaaagh
  0.0808721205363789               best
                 0.0             aaaaah
0.051088218985481874             delici
                 0.0 aaaaahhhhhhhhhhhhhhh
 0.04603376760119816 4             love
                 0.0              aaaah
 0.03838207466207587         disappoint
                 0.0               aaah
 0.03257144975521582               good
                 0.0             aaamaz
0.029471152783491987              excel
                 0.0             aachen
0.022677666208702856                bad
                 0.0                aad
 0.02224952848379748            perfect
                 0.0               aadp
 0.01593856559068014            favorit
                 0.0              aafco
0.010943421002860288              tasti
                 0.0               aagh
  0.0106602130864715               nice
                 0.0                aah
0.010034615926634318              yummi
                 0.0               aahh
0.009726157264430612            horribl
                 0.0               aand
0.009420355588665231                 aw
                 0.0           aardvark
0.007923216646274354             wonder
                 0.0                 ab
0.0076688564431999316           unfortun
                 0.0              aback
0.007432651745134353            thought
                 0.0             abacor
0.006830754608818924            terribl
                 0.0             abaloo
0.006119704429711919              money
                 0.0            abandon
```

```
0.0060393132437182966        tast
                 0.0     abaolut
0.005586288520877951        howev
                 0.0   abattoir
0.005417981818808328         amaz
                 0.0       abba
 0.00424205980939696       addict
```

since for negative class we are observing many features having feature importance values coming as 0,I m discarding all the features having 0 feature importance values and then printing the output of yop 25 features below:

In [37]:

```
features = bow_object.get_feature_names()
featuresAndcoeff = sorted(zip(dt.feature_importances_, features))
l = []
for a, b in featuresAndcoeff:
    if a == 0:
        continue
    else:
        l.append([a,b])

l = sorted(l, key=lambda x: x[0], reverse=True)
positive = l[:25]
negative = l[-(25 + 1): -1]
a = dict(top25_pos=positive,top25_neg=negative)
dataframe = pd.DataFrame(a)
dataframe
```

Out[37]:

	top25_neg	top25_pos
0	[0.00021228424577740033, sparkl]	[0.13948093745846474, great]
1	[0.00021210079852126024, elder]	[0.13001810316688217, not]
2	[0.00021191758895342224, tax]	[0.0808721205363789, best]
3	[0.0002117346166637638, proven]	[0.051088218985481874, delici]

	top25_neg	top25_pos
4	[0.00021155188124377322, stabl]	[0.046033767601198164, love]
5	[0.00021145872075646526, grill]	[0.03838207466207587, disappoint]
6	[0.00021125268806688243, son]	[0.03257144975521582, good]
7	[0.00021104695634792175, decent]	[0.029471152783491987, excel]
8	[0.00021084152501623084, wheat]	[0.022677666208702856, bad]
9	[0.0002106363934864601, constip]	[0.02224952848379748, perfect]
10	[0.00021043156117435584, sens]	[0.01593856559068014, favorit]
11	[0.00021022702750007175, spicey]	[0.010943421002860288, tasti]
12	[0.0002100227918827855, opportun]	[0.0106602130864715, nice]
13	[0.00020981885374352522, depend]	[0.010034615926634318, yummi]
14	[0.000209411867589231, guilt]	[0.009726157264430612, horribl]
15	[0.00020920881842388246, cure]	[0.009420355588665231, aw]
16	[0.00020542688790375736, grate]	[0.007923216646274354, wonder]
17	[0.00019758220787443478, terrif]	[0.0076688564431999316, unfortun]
18	[0.00019701638881093868, prepar]	[0.007432651745134353, thought]
19	[0.00019623328302354663, friend]	[0.006830754608818924, terribl]
20	[0.00019500042850709542, area]	[0.006119704429711919, money]
21	[0.00017714320052933188, avail]	[0.0060393132437182966, tast]
22	[0.0001613512245253714, mild]	[0.005586288520877951, howev]
23	[0.0001612459332059049, tendenc]	[0.005417981818808328, amaz]
24	[0.00016114074491472156, high]	[0.00424205980939696, addict]

using graphviz, calculating the conditions

In [38]:
```
from sklearn import tree
```

In [39]:
```
clf = tree.DecisionTreeClassifier(max_depth=2, min_samples_split=50)
clf = clf.fit(xtr, ytr)
dot_data = tree.export_graphviz(clf, out_file=None)
graph = graphviz.Source(dot_data)
graph.render("BoW_DecisionTree.png")
```

Out[39]:
```
'BoW_DecisionTree.png.pdf'
```

TF-IDF

In [40]:
```
df.columns
```

Out[40]:
```
Index(['index', 'Id', 'ProductId', 'UserId', 'ProfileName',
       'HelpfulnessNumerator', 'HelpfulnessDenominator', 'Score', 'Tim
e',
       'Summary', 'Text', 'CleanedText_Bow', 'ClenedText_W2Vtfdf', 'Bow
_feat',
       'Bow_new_feat', 'w2v_feat', 'w2v_new_feat'],
      dtype='object')
```

In [41]:
```
#train test split:

xt, xtest, yt, ytest = train_test_split(df.ClenedText_W2Vtfdf, df.Score
, test_size=0.2, shuffle=False)
xtr, xcv, ytr, ycv = train_test_split(xt, yt, test_size=0.2, shuffle=Fa
lse)
```

tf-Idf featurizer

In [42]:
```
from sklearn.feature_extraction.text import TfidfVectorizer

tfidf_object = TfidfVectorizer(ngram_range=(1,1))
xtr = tfidf_object.fit_transform(xtr)
xcv = tfidf_object.transform(xcv)
xtest = tfidf_object.transform(xtest)
```

In [50]:
```
xtr.shape, xcv.shape, xtest.shape
```
Out[50]:
```
((64000, 43852), (16000, 43852), (20000, 43852))
```

since I HAVE MADE A COMMON CLASS FOR ALL THE VECTORIZERS, USING THAT CLASS in here for instantiating tfidf object

In [53]:
```
class decisiontree:

    '''building the decision tree classifier based off various paramete
rs'''

    #instantiating the instance attributes:
    def __init__(self, xtr, ytr, xcv, ycv, minimum_splits=[5,10,100,500
], maximum_depth=[1, 5, 10, 50, 100, 500, 1000]):
        self.xtr = xtr
        self.ytr = ytr
        self.xcv = xcv
        self.ycv = ycv
        self.minimum_splits = minimum_splits
        self.maximum_depth = maximum_depth

    #creating a method of calling DT classifier:
    def classfier(self, auc_dict_cv={}, auc_dict_tr={}):
        for splits in self.minimum_splits:
            for depths in self.maximum_depth:
                clf = DecisionTreeClassifier(max_depth=depths, min_samp
les_split=splits)
                print(depths, splits)
                clf.fit(self.xtr, self.ytr)
                y_pred_cv = clf.predict_proba(self.xcv)

                #performance metric on CV data:
                fpr_cv, tpr_cv, thresholds_cv = roc_curve(ycv, y_pred_c
v[:,1])
                auc_val = auc(fpr_cv, tpr_cv)
                auc_dict_cv[zip([splits], [depths])] = auc_val
```

```
                #performance metrics for training data:
                y_pred_tr = clf.predict_proba(self.xtr)
                fpr_tr, tpr_tr, thresholds_tr = roc_curve(ytr, y_pred_t
r[:,1])
                auc_val = auc(fpr_tr, tpr_tr)
                auc_dict_tr[zip([splits], [depths])] = auc_val

        return auc_dict_tr, auc_dict_cv
```

TFidf decision tree instance creation on training and CV data

In [54]:

```
import time
start = time.time()
Tfidf_instance = decisiontree(xtr, ytr, xcv, ycv)

dictionary_train, dictionary_cv = Tfidf_instance.classfier()
end = time.time()

print('time is(in seconds): ', end - start)
```

```
1 5
5 5
10 5
50 5
100 5
500 5
100 5
1 10
5 10
10 10
50 10
100 10
500 10
100 10
1 100
5 100
```

```
10 100
50 100
100 100
500 100
100 100
1 500
5 500
10 500
50 500
100 500
500 500
100 500
time is(in seconds):  644.251556634903
```

In [55]:

```
#creating dictionary :
train_list = [(list(x), np.round(y,3)) for x, y in dictionary_train.ite
ms()]
cv_list = [(list(x), np.round(y,3)) for x, y in dictionary_cv.items()]


print(train_list[0])
print()
print(cv_list[0])
```

```
([(5, 1)], 0.617)

([(10, 500)], 0.686)
```

sorting the list based off AUC score

In [56]:

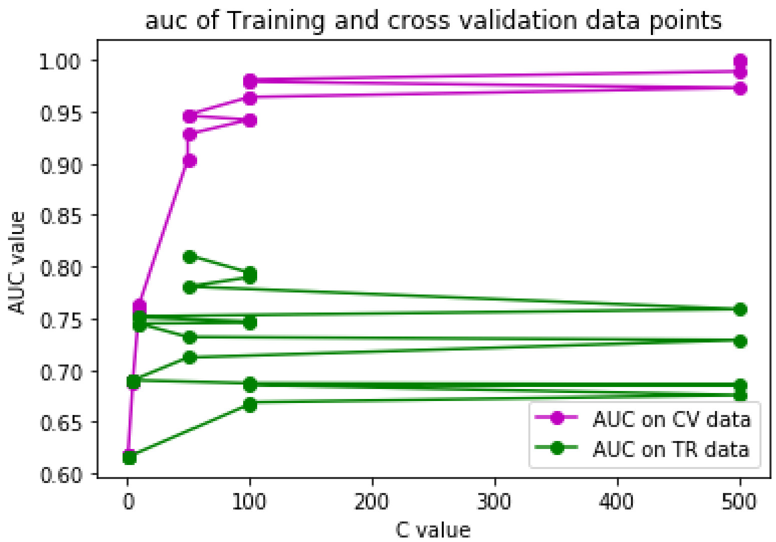
```
tr_list = sorted(train_list, key=lambda x: x[1], reverse=True)
cv_list = sorted(cv_list, key=lambda x: x[1], reverse=True)
print('sorted train list score based off AUC score is:\n', tr_list[0:3
])
print('****************************************')
print()
print('sorted train list score based off AUC score is:\n', cv_list[0:3
])
```

```
sorted train list score based off AUC score is:

 [([(5, 500)], 1.0), ([(10, 500)], 0.999), ([(100, 500)], 0.989)]
****************************************

sorted train list score based off AUC score is:
 [([(500, 50)], 0.811), ([(500, 100)], 0.794), ([(500, 100)], 0.79)]
```

Plotting AUC Curve on training and cv data based off depth

In [57]:
```
plt.plot([x[0][0][1] for x in tr_list], [x[1] for x in tr_list], linest
yle='-', color='m', marker='o',label='AUC on CV data')
plt.plot([x[0][0][1] for x in cv_list], [x[1] for x in cv_list], linest
yle='-', color='g', marker='o', label='AUC on TR data')
#plt.plot([0, 1], [0, 1], linestyle='--')
plt.xlabel("C value")
plt.ylabel('AUC value')
plt.title('auc of Training and cross validation data points')
plt.legend()
```

Out[57]:
```
<matplotlib.legend.Legend at 0x7f0e366ffda0>
```



OptimalTFidf- DecesionTree[depth =50 and splits =500]

In [60]:
```
dt = DecisionTreeClassifier(max_depth=50, min_samples_split=500, class_
weight='balanced')
dt.fit(xtr, ytr)

ypred = dt.predict_proba(xtest)
ypred_tr = dt.predict_proba(xtr)


fpr_tr, tpr_tr, thresholds_tr= roc_curve(ytr, ypred_tr[:,1])
auc_tr = auc(fpr_tr, tpr_tr)


fpr_test, tpr_test, thresholds_test= roc_curve(ytest, ypred[:,1])
auc_test = auc(fpr_test, tpr_test)

print(auc_test)
```

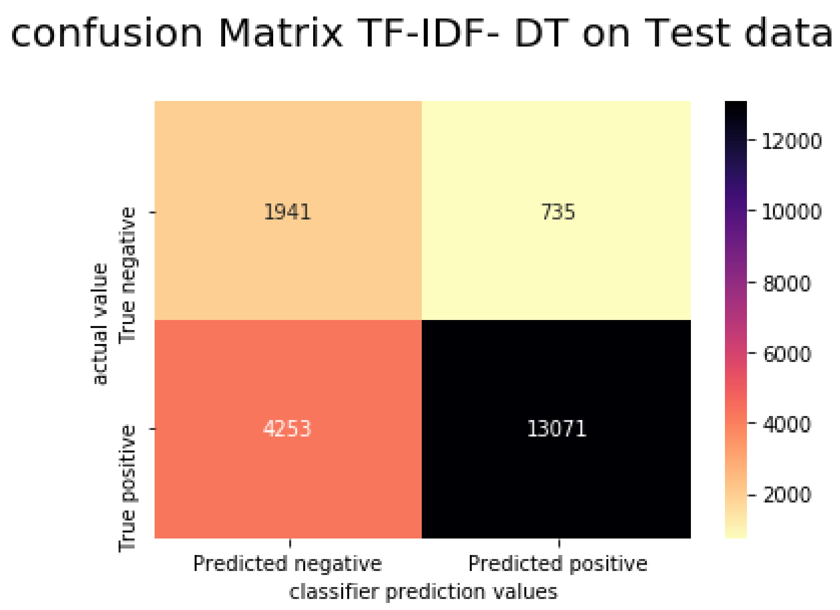
```
0.795184547025839
```

plotting confusion matrix on test data

In [61]:
```
%time
ypred = np.where(ypred[:,1] < 0.5, 0, 1)
#creating confusion matrix:

cf = confusion_matrix(ytest, ypred)
labels = ['True negative', 'True positive']

df_cf = pd.DataFrame(cf, index=labels, columns=['Predicted negative',
'Predicted positive'])
sns.heatmap(df_cf, annot=True,fmt='3d', cmap='magma_r')

plt.title("confusion Matrix TF-IDF- DT on Test data\n", size=20)
```

```
plt.xlabel("classifier prediction values")
plt.ylabel("actual value")
plt.show()
```

```
CPU times: user 0 ns, sys: 0 ns, total: 0 ns
Wall time: 6.44 µs
```



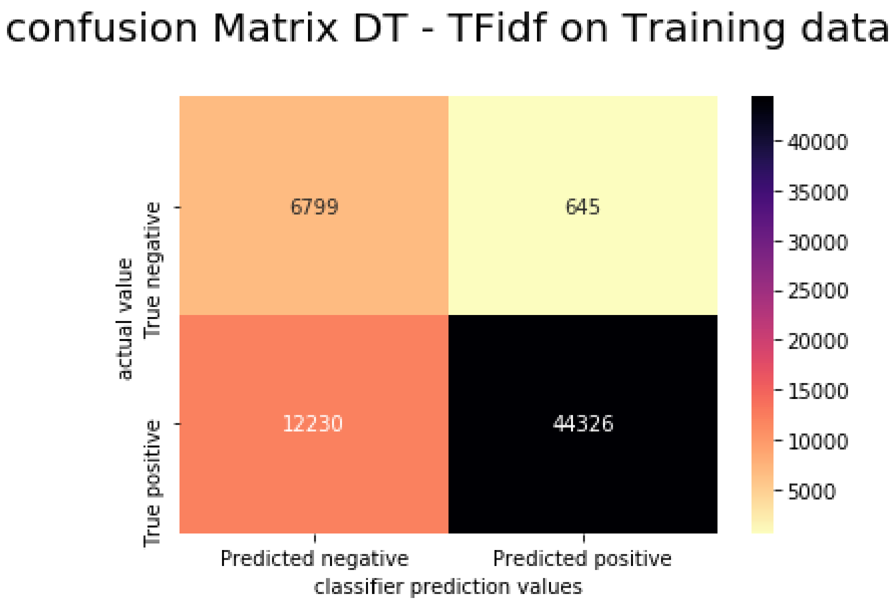
plotting confusion matrix on training data

In [63]:

```
%time
y_pred_tr = np.where(ypred_tr[:,1] < 0.5, 0, 1)
#creating confusion matrix:

cf = confusion_matrix(ytr, y_pred_tr)
labels = ['True negative', 'True positive']

df_cf = pd.DataFrame(cf, index=labels, columns=['Predicted negative',
'Predicted positive'])
sns.heatmap(df_cf, annot=True,fmt='3d', cmap='magma_r')
```

```
plt.title("confusion Matrix DT - TFidf on Training data\n", size=20)
plt.xlabel("classifier prediction values")
plt.ylabel("actual value")
plt.show()
```

```
CPU times: user 0 ns, sys: 0 ns, total: 0 ns
Wall time: 6.44 µs
```

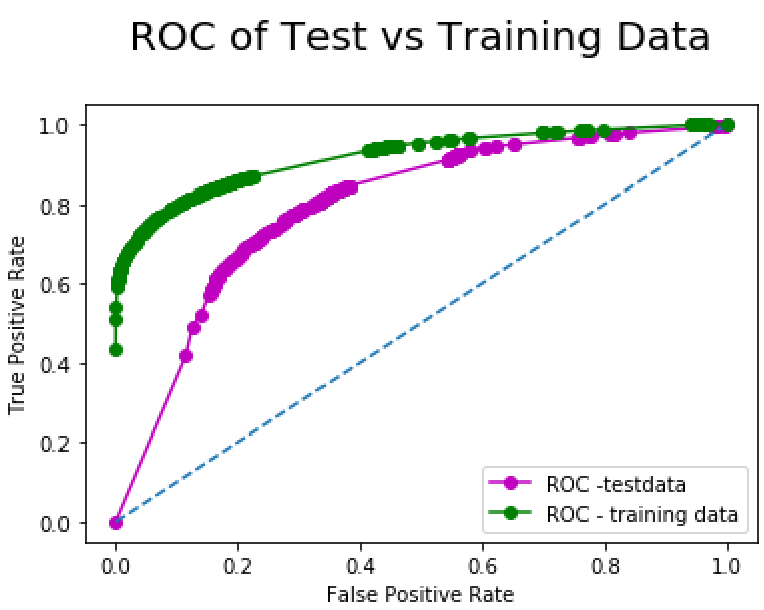


plotting ROC curve on test data and training data

In [65]:
```
plt.plot(fpr_test, tpr_test, color='m', marker='o',label='ROC -testdat
a')
plt.plot([0, 1], [0, 1], linestyle='--')
plt.plot(fpr_tr, tpr_tr, linestyle='-', color='g', marker='o', label='R
OC - training data')
plt.xlabel('False Positive Rate')
plt.ylabel('True Positive Rate')
plt.title('ROC of Test vs Training Data\n', size=20)
plt.legend()
```

Out[65]: `<matplotlib.legend.Legend at 0x7f0e35ce2748>`

```
Out[65]: <matplotlib.legend.Legend at 0x7f0e55ee2748>
```



printing top 25 features of both positive and negative class

In [66]:

```
features = tfidf_object.get_feature_names()
featuresAndcoeff = sorted(zip(dt.feature_importances_, features))
top_features = zip(featuresAndcoeff[:25],featuresAndcoeff[:-(25+1): -1]
 )
print('\t\t\t\tNegative\t\t\t\t\t\tPositive')
print('-' * 120)

for (wn1, fn1), (wp1, fp1) in top_features:
    print('{:>20} {:>20}                                {:>20} {:>20}'.
format(wn1, fn1, wp1, fp1))
```

```
                              Negative
                              Positive
------------------------------------------------------------------------
------------------------------------------------
                0.0                     aaa
```

```
0.10516368328831541                great
           0.0       aaaaaaaaagghh
0.08981493578011883                  not
           0.0              aaaaah
0.05636387210694428                 best
           0.0 aaaaahhhhhhhhhhhhhh
 0.03962773559888438            delicious
           0.0               aaaah
0.026029106587324068                 good
           0.0                aaah
0.02502140986757704                 love
           0.0               aaahs
0.02222678208686428              perfect
           0.0              aachen
0.018976425351451484               loves
           0.0                 aad
0.018892614404691123            favorite
           0.0                aadp
0.018042452974844997           excellent
           0.0               aafco
0.017859814717201716                 bad
           0.0                aagh
0.017834669709603533           wonderful
           0.0              aahhed
0.016094524181307617                nice
           0.0              aahing
0.013603866625851562        disappointed
           0.0                aand
0.013554134169423238              highly
           0.0            aardvark
0.011454931690362291             thought
           0.0               aback
0.011186492407121829               tasty
           0.0             abandon
0.010775294933957291                find
           0.0           abandoned
0.010101852442754114            terrible
           0.0          abandoning
0.009272475511591933       unfortunately
           0.0          abaolutely
```

```
0.009065122709265606           money
               0.0         abattoir
0.007893918145189327            easy
               0.0             abba
 0.00765073106257202           awful
               0.0            abbey
0.007017365346739904         however
               0.0      abbreviated
0.006380040216467915           taste
```

since for negative class we are observing many features having feature importance values coming as 0,I m discarding all the features having 0 feature importance values and then printing the output of yop 25 features below:

In [67]:

```
features = tfidf_object.get_feature_names()
featuresAndcoeff = sorted(zip(dt.feature_importances_, features))
l = []
for a, b in featuresAndcoeff:
    if a == 0:
        continue
    else:
        l.append([a,b])

l = sorted(l, key=lambda x: x[0], reverse=True)
positive = l[:25]
negative = l[-(25 + 1): -1]
a = dict(top25_pos=positive,top25_neg=negative)
dataframe = pd.DataFrame(a)
dataframe
```

Out[67]:

	top25_neg	top25_pos
0	[0.00039993950958781333, prime]	[0.10516368328831541, great]
1	[0.0003990766917521303, play]	[0.08981493578011883, not]
2	[0.0003863104824043948, buy]	[0.05636387210694428, best]

	top25_neg	top25_pos
3	[0.000382929778353476, feel]	[0.03962773559888438, delicious]
4	[0.0003698709445561861, store]	[0.026029106587324068, good]
5	[0.0003677760344292645, trouble]	[0.02502140986757704, love]
6	[0.00036514173476490737, difference]	[0.02222678208686428, perfect]
7	[0.0003645719717059419, reasonable]	[0.018976425351451484, loves]
8	[0.0003640035411819806, run]	[0.018892614404691123, favorite]
9	[0.0003634364390462109, finding]	[0.018042452974844997, excellent]
10	[0.0003628706611624885, plenty]	[0.017859814717201716, bad]
11	[0.00035137151050768523, wheat]	[0.017834669709603533, wonderful]
12	[0.0003375142606820132, yeast]	[0.016094524181307617, nice]
13	[0.00033638280253443537, stays]	[0.013603866625851562, disappointed]
14	[0.0003362727238730474, website]	[0.013554134169423238, highly]
15	[0.0003178528166120719‌4, similar]	[0.011454931690362291, thought]
16	[0.0003174207432259656, biggest]	[0.011186492407121829, tasty]
17	[0.00031698955025107577, live]	[0.010775294933957291, find]
18	[0.0003165592352955709, robust]	[0.010101852442754114, terrible]
19	[0.0003161297959787346, heavy]	[0.009272475511591933, unfortunately]
20	[0.000315701229923574, pleasantly]	[0.009065122709265606, money]
21	[0.00031562967235673485, everything]	[0.007893918145189327, easy]
22	[0.0003112235624949291, recently]	[0.0076507310625720‌2, awful]
23	[0.00030905502572903655, pretty]	[0.007017365346739904, however]
24	[0.000273979699052875, totally]	[0.006380040216467915, taste]

using graphviz for plotting the conditions, saving the same onto files

In [68]:
```
clf = tree.DecisionTreeClassifier(max_depth=3, min_samples_split=50)
clf = clf.fit(xtr, ytr)
dot_data = tree.export_graphviz(clf, out_file=None)
graph = graphviz.Source(dot_data)
graph.render("TFidf_DecisionTree")
```

Out[68]:
```
'TFidf_DecisionTree.pdf'
```

Avg-Word2Vec

In [69]:
```
df.columns
```

Out[69]:
```
Index(['index', 'Id', 'ProductId', 'UserId', 'ProfileName',
       'HelpfulnessNumerator', 'HelpfulnessDenominator', 'Score', 'Tim
e',
       'Summary', 'Text', 'CleanedText_Bow', 'ClenedText_W2Vtfdf', 'Bow
_feat',
       'Bow_new_feat', 'w2v_feat', 'w2v_new_feat'],
      dtype='object')
```

train test cv split

In [70]:
```
xtrain, xtest, ytrain, ytest = train_test_split(df.ClenedText_W2Vtfdf,
df.Score, test_size=0.2, shuffle=False)
xtr, xcv, ytr, ycv = train_test_split(xtrain, ytrain, test_size=0.2, sh
uffle=False)
```

list of lists of train, cv, test data

In [71]:
```
%%time

#training list of words:
train_list = []
for sentence in xtr:
    tmp_list = []
```

```
    for word in sentence.split():
        tmp_list.append(word)
    train_list.append(tmp_list)

#cv list of words
cv_list = []
for sentence in xcv:
    tmp_list = []
    for word in sentence.split():
        tmp_list.append(word)
    cv_list.append(tmp_list)

#test list of words:
test_list = []
for sentence in xtest:
    tmp_list = []
    for word in sentence.split():
        tmp_list.append(word)
    test_list.append(tmp_list)
```

```
CPU times: user 1.19 s, sys: 160 ms, total: 1.35 s
Wall time: 1.35 s
```

instantiating word2vec object for Train, cv, test data

In [72]:

```
%%time

from gensim.models import Word2Vec

#instantiating training,cv, test W2V object:

trainw2v = Word2Vec(train_list, size=1000)
cvw2v = Word2Vec(cv_list, size=1000)
testw2v = Word2Vec(test_list, size=1000)

#training word2vec List:
train_vocab = list(trainw2v.wv.vocab.keys())
```

```
#cv word2vec List:
cv_vocab = list(cvw2v.wv.vocab.keys())

#test word2vec List:
test_vocab = list(testw2v.wv.vocab.keys())
```

```
CPU times: user 1min 43s, sys: 332 ms, total: 1min 43s
Wall time: 1min 43s
```

Avg-W2V for train, cv, test data

In [73]:

```
%%time

#avg-w2v for training data****************************:
train_vector = []
for sentence in train_list:
    vector = np.zeros(1000)
    for word in sentence:
        cnt = 0
        if word in train_vocab:
            vector = vector + trainw2v.wv[word]
            cnt = cnt + 1
    if cnt != 0:
        vector = vector / cnt
    train_vector.append(vector)

train_vector = np.array(train_vector)
print('train vector shape is {}'.format(train_vector.shape))


#avg-w2v for cv data***********************************:
cv_vector = []
for sentence in cv_list:
    vector = np.zeros(1000)
    for word in sentence:
        cnt = 0
        if word in cv_vocab:
            vector = vector + cvw2v.wv[word]
```

```
            cnt = cnt + 1
    if cnt != 0:
        vector = vector / cnt
    cv_vector.append(vector)

cv_vector = np.array(cv_vector)
print('cv vector shape is {}'.format(cv_vector.shape))




#avg-w2v for test data**********************************:
test_vector = []
for sentence in test_list:
    vector = np.zeros(1000)
    for word in sentence:
        cnt = 0
        if word in test_vocab:
            vector = vector + testw2v.wv[word]
            cnt = cnt + 1
    if cnt != 0:
        vector = vector / cnt
    test_vector.append(vector)

test_vector = np.array(test_vector)
print('test vector shape is {}'.format(test_vector.shape))
```

```
train vector shape is (64000, 1000)
cv vector shape is (16000, 1000)
test vector shape is (20000, 1000)
CPU times: user 12min 21s, sys: 1.66 s, total: 12min 23s
Wall time: 12min 24s
```

since I HAVE MADE A COMMON CLASS FOR ALL THE VECTORIZERS, USING THAT CLASS in here for instantiating AvgW2V object

In [74]:
```
class decisiontree:

    '''building the decision tree classifier based off various paramete
rs'''

    #instantiating the instance attributes:
    def __init__(self, xtr, ytr, xcv, ycv, minimum_splits=[5,10,100,500
], maximum_depth=[1, 5, 10, 50, 100, 500, 1000]):
        self.xtr = xtr
        self.ytr = ytr
        self.xcv = xcv
        self.ycv = ycv
        self.minimum_splits = minimum_splits
        self.maximum_depth = maximum_depth

    #creating a method of calling DT classifier:
    def classfier(self, auc_dict_cv={}, auc_dict_tr={}):
        for splits in self.minimum_splits:
            for depths in self.maximum_depth:
                clf = DecisionTreeClassifier(max_depth=depths, min_samp
les_split=splits)
                print(depths, splits)
                clf.fit(self.xtr, self.ytr)
                y_pred_cv = clf.predict_proba(self.xcv)

                #performance metric on CV data:
                fpr_cv, tpr_cv, thresholds_cv = roc_curve(ycv, y_pred_c
v[:,1])
                auc_val = auc(fpr_cv, tpr_cv)
                auc_dict_cv[zip([splits], [depths])] = auc_val

                #performance metrics for training data:
                y_pred_tr = clf.predict_proba(self.xtr)
                fpr_tr, tpr_tr, thresholds_tr = roc_curve(ytr, y_pred_t
r[:,1])
                auc_val = auc(fpr_tr, tpr_tr)
                auc_dict_tr[zip([splits], [depths])] = auc_val

        return auc_dict_tr, auc_dict_cv
```

Avgw2v decision tree instance creation on training and CV data

In [76]:
```
import time
start = time.time()
w2v_instance = decisiontree(train_vector, ytr, cv_vector, ycv)

dictionary_train, dictionary_cv = w2v_instance.classfier()
end = time.time()

print('time is(in seconds): ', end - start)
```

```
1 5
5 5
10 5
50 5
100 5
500 5
1000 5
1 10
5 10
10 10
50 10
100 10
500 10
1000 10
1 100
5 100
10 100
50 100
100 100
500 100
1000 100
1 500
5 500
10 500
50 500
```

```
100 500
500 500
1000 500
time is(in seconds):  3744.681820869446
```

In [77]:
```
#creating dictionary :

train_list = [(list(x), np.round(y,3)) for x, y in dictionary_train.ite
ms()]
cv_list = [(list(x), np.round(y,3)) for x, y in dictionary_cv.items()]


print(train_list[0])
print()
print(cv_list[0])
```

```
([(100, 50)], 0.964)

([(100, 100)], 0.537)
```

sorting the list based off AUC score

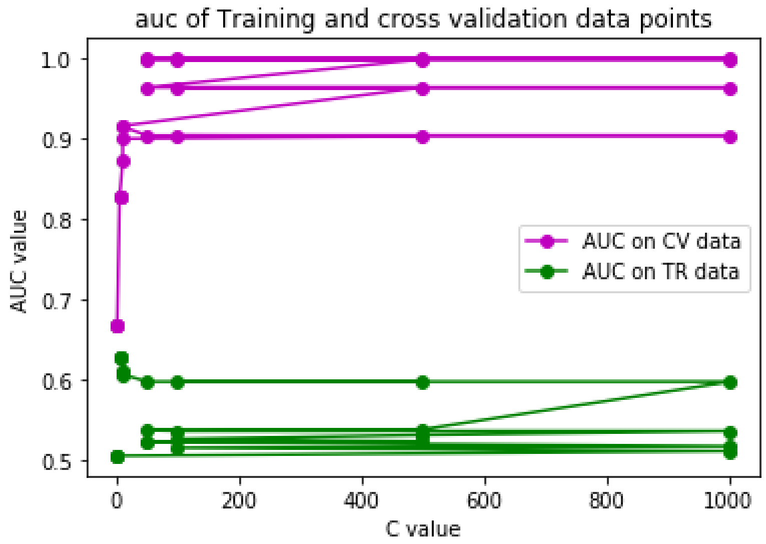
In [78]:
```
tr_list = sorted(train_list, key=lambda x: x[1], reverse=True)
cv_list = sorted(cv_list, key=lambda x: x[1], reverse=True)
print('sorted train list score based off AUC score is:\n', tr_list[0:3
])
print('****************************************')
print()
print('sorted CV list score based off AUC score is:\n', cv_list[0:3])
```

```
sorted train list score based off AUC score is:
 [([(5, 500)], 1.0), ([(5, 50)], 1.0), ([(5, 1000)], 1.0)]
****************************************

sorted CV list score based off AUC score is:
 [([(100, 5)], 0.627), ([(500, 5)], 0.627), ([(10, 5)], 0.627)]
```

Plotting AUC Curve on training and cv data based off depth

In [79]:
```
plt.plot([x[0][0][1] for x in tr_list], [x[1] for x in tr_list], linest
yle='-', color='m', marker='o',label='AUC on CV data')
plt.plot([x[0][0][1] for x in cv_list], [x[1] for x in cv_list], linest
yle='-', color='g', marker='o', label='AUC on TR data')
#plt.plot([0, 1], [0, 1], linestyle='--')
plt.xlabel("C value")
plt.ylabel('AUC value')
plt.title('auc of Training and cross validation data points')
plt.legend()
```

Out[79]: <matplotlib.legend.Legend at 0x7f0e1c1c8d68>



OptimalAvgW2V- DecesionTree[depth = 5 and splits = 100]

In [80]:
```
dt = DecisionTreeClassifier(max_depth=5, min_samples_split=100, class_w
eight='balanced')
dt.fit(train_vector, ytr)
```

```
ypred = dt.predict_proba(test_vector)
ypred_tr = dt.predict_proba(train_vector)


fpr_tr, tpr_tr, thresholds_tr= roc_curve(ytr, ypred_tr[:,1])
auc_tr = auc(fpr_tr, tpr_tr)


fpr_test, tpr_test, thresholds_test= roc_curve(ytest, ypred[:,1])
auc_test = auc(fpr_test, tpr_test)

print(auc_test)
```

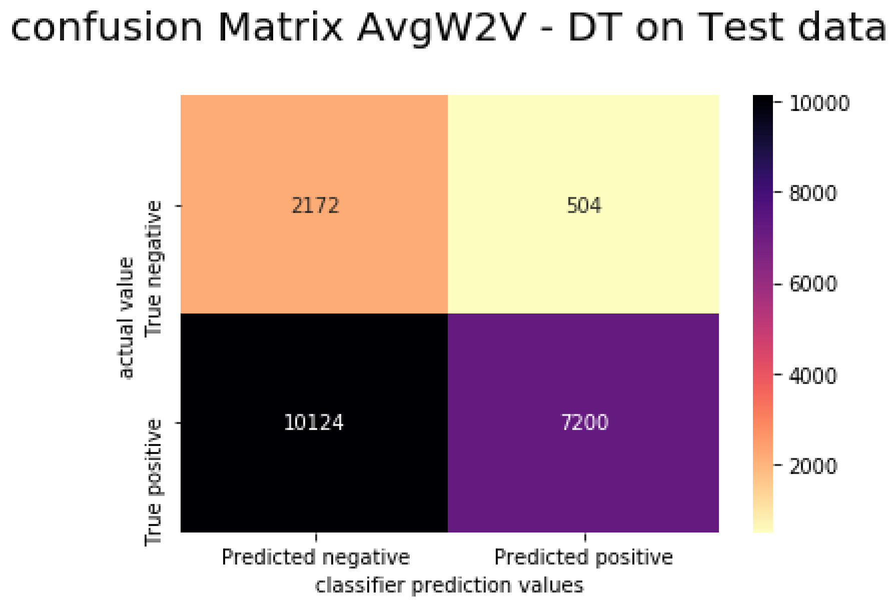
```
0.6617441622584633
```

plotting confusion matrix on test data

In [81]:

```
%time
ypred = np.where(ypred[:,1] < 0.5, 0, 1)
#creating confusion matrix:

cf = confusion_matrix(ytest, ypred)
labels = ['True negative', 'True positive']

df_cf = pd.DataFrame(cf, index=labels, columns=['Predicted negative',
'Predicted positive'])
sns.heatmap(df_cf, annot=True,fmt='3d', cmap='magma_r')

plt.title("confusion Matrix AvgW2V - DT on Test data\n", size=20)
plt.xlabel("classifier prediction values")
plt.ylabel("actual value")
plt.show()
```

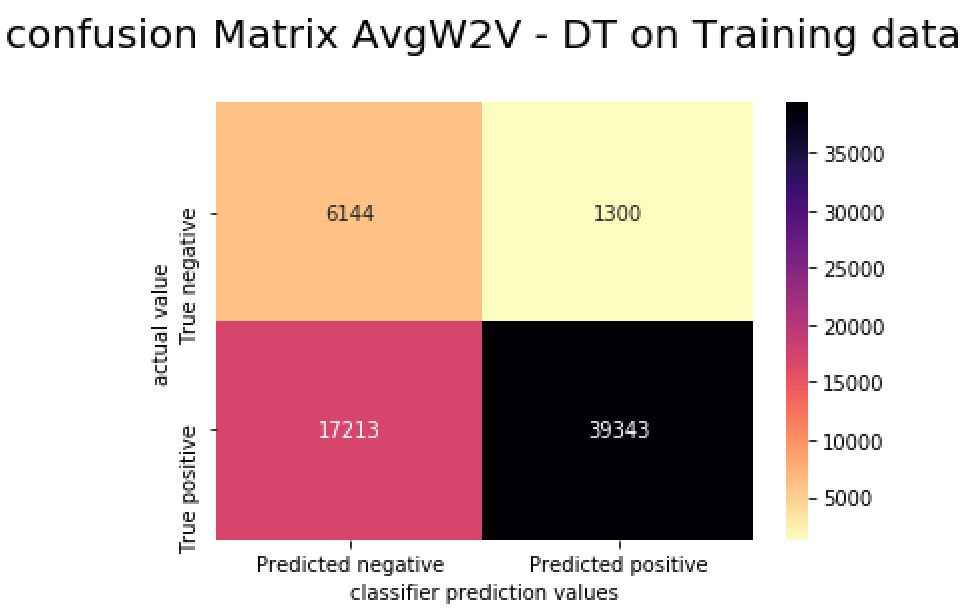
```
CPU times: user 0 ns, sys: 0 ns, total: 0 ns
Wall time: 7.63 µs
```



plotting confusion matrix on training data

In [82]:
```
%time
y_pred_tr = np.where(ypred_tr[:,1] < 0.5, 0, 1)
#creating confusion matrix:

cf = confusion_matrix(ytr, y_pred_tr)
labels = ['True negative', 'True positive']

df_cf = pd.DataFrame(cf, index=labels, columns=['Predicted negative',
'Predicted positive'])
sns.heatmap(df_cf, annot=True,fmt='3d', cmap='magma_r')

plt.title("confusion Matrix AvgW2V - DT on Training data\n", size=20)
plt.xlabel("classifier prediction values")
plt.ylabel("actual value")
plt.show()
```

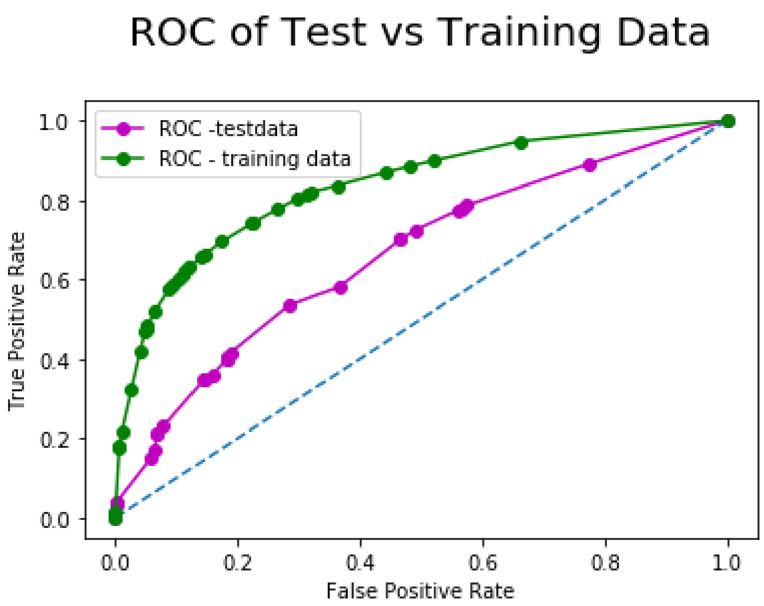
```
CPU times: user 0 ns, sys: 0 ns, total: 0 ns
Wall time: 6.91 µs
```



plotting ROC curve on test data and training data

In [83]:
```
plt.plot(fpr_test, tpr_test, color='m', marker='o',label='ROC -testdat
a')
plt.plot([0, 1], [0, 1], linestyle='--')
plt.plot(fpr_tr, tpr_tr, linestyle='-', color='g', marker='o', label='R
OC - training data')
plt.xlabel('False Positive Rate')
plt.ylabel('True Positive Rate')
plt.title('ROC of Test vs Training Data\n', size=20)
plt.legend()
```

Out[83]: <matplotlib.legend.Legend at 0x7f0e15a40f98>



TFidf-W2V

In [6]:

```
%%time

from gensim.models import Word2Vec
from sklearn.feature_extraction.text import TfidfVectorizer

#train, cv, test split:
xtrain, xtest, ytrain, ytest = train_test_split(df.ClenedText_W2Vtfdf, df.Score, test_size=0.2, shuffle=False)
xtr, xcv, ytr, ycv = train_test_split(xtrain, ytrain, test_size=0.2, shuffle=False)


#training list of words:
train_list = []
```

```
for sentence in xtr:
    tmp_list = []
    for word in sentence.split():
        tmp_list.append(word)
    train_list.append(tmp_list)

#cv list of words:
cv_list = []
for sentence in xcv:
    tmp_list = []
    for word in sentence.split():
        tmp_list.append(word)
    cv_list.append(tmp_list)

#test list of words:
test_list = []
for sentence in xtest:
    tmp_list = []
    for word in sentence.split():
        tmp_list.append(word)
    test_list.append(tmp_list)


#instantiating training,cv, test W2V object:

trainw2v = Word2Vec(train_list, size=1000)
cvw2v = Word2Vec(cv_list, size=1000)
testw2v = Word2Vec(test_list, size=1000)

#training word2vec List:
train_vocab = list(trainw2v.wv.vocab.keys())

#cv word2vec List:
cv_vocab = list(cvw2v.wv.vocab.keys())

#test word2vec List:
test_vocab = list(testw2v.wv.vocab.keys())
```

```
CPU times: user 1min 54s, sys: 444 ms, total: 1min 54s
Wall time: 1min 54s
```

Tfidf vectorizer

In [7]:
```
model = TfidfVectorizer()
xtr = model.fit_transform(xtr)
xcv = model.transform(xcv)
xtest = model.transform(xtest)

# 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 [8]:
```
xtr.shape, xcv.shape, xtest.shape
```

Out[8]:
```
((64000, 43848), (16000, 43848), (20000, 43848))
```

In [9]:
```
len(train_list)
```

Out[9]:
```
64000
```

Creating TFIDF-W2V for training data,

TFIDF-W2V for cv data AND

TFIDF-W2V for test data

In [11]:
```
import time
start = time.time()

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


#tf-idf for train data:
tfidf_train_vectors = []; # the tfidf-w2v for each sentence/review is s
tored in this list
```

```
row=0;
for sent in train_list: # for each review/sentence
    sent_vec = np.zeros(1000) # as word vectors are of zero length
    weight_sum =0; # num of words with a valid vector in the sentence/r
eview
    for word in sent: # for each word in a review/sentence
        if word in train_vocab and word in tfidf_feat:
            vec = trainw2v.wv[word]
            tf_idf = dictionary[word]*(sent.count(word)/len(sent))
            sent_vec += (vec * tf_idf)
            weight_sum += tf_idf
    if weight_sum != 0:
        sent_vec /= weight_sum
    tfidf_train_vectors.append(sent_vec)
    row += 1

#######################################################################
################################################

#tfidf for CV data:

tfidf_cv_vectors = []; # the tfidf-w2v for each sentence/review is stor
ed in this list
row=0;
for sent in cv_list: # for each review/sentence
    sent_vec = np.zeros(1000) # as word vectors are of zero length
    weight_sum =0; # num of words with a valid vector in the sentence/r
eview
    for word in sent: # for each word in a review/sentence
        if word in cv_vocab and word in tfidf_feat:
            vec = cvw2v.wv[word]
            tf_idf = dictionary[word]*(sent.count(word)/len(sent))
            sent_vec += (vec * tf_idf)
            weight_sum += tf_idf
    if weight_sum != 0:
        sent_vec /= weight_sum
    tfidf_cv_vectors.append(sent_vec)
    row += 1
```

```
##########################################################################
##################################################



tfidf_feat = model.get_feature_names() # tfidf words/col-names
# final_tf_idf is the sparse matrix with row= sentence, col=word and ce
ll_val = tfidf


#tfidf for Test Data:

tfidf_test_vectors = []; # the tfidf-w2v for each sentence/review is st
ored in this list
row=0;
for sent in test_list: # for each review/sentence
    sent_vec = np.zeros(1000) # as word vectors are of zero length
    weight_sum =0; # num of words with a valid vector in the sentence/r
eview
    for word in sent: # for each word in a review/sentence
        if word in test_vocab and word in tfidf_feat:
            vec = testw2v.wv[word]
            tf_idf = dictionary[word]*(sent.count(word)/len(sent))
            sent_vec += (vec * tf_idf)
            weight_sum += tf_idf
    if weight_sum != 0:
        sent_vec /= weight_sum
    tfidf_test_vectors.append(sent_vec)
    row += 1


end = time.time()

print('total time taken in seconds is: ', end - start)
```

```
total time taken in seconds is:  2726.192879676819
```

In [ ]:
```
# import pickle
```

```
# file1 = open('tfidfw2v_train.pickle', 'wb')
# pickle.dump(tfidf_train_vectors, file1)
# file1.close()

# file1 = open('tfidfw2v_cv.pickle', 'wb')
# pickle.dump(tfidf_cv_vectors, file1)
# file1.close()

# file1 = open('tfidfw2v_test.pickle', 'wb')
# pickle.dump(tfidf_test_vectors, file1)
# file1.close()
```

conversion of list into array

In [12]:
```
xtr = np.array(tfidf_train_vectors)
xcv = np.array(tfidf_cv_vectors)
xtest = np.array(tfidf_test_vectors)

print(xtr.shape)
print(xcv.shape)
print(xtest.shape)
```

```
(64000, 1000)
(16000, 1000)
(20000, 1000)
```

Tfidf-w2v DT CLASS creation

In [13]:
```
class decisiontree:

    '''building the decision tree classifier based off various paramete
rs'''

    #instantiating the instance attributes:
    def __init__(self, xtr, ytr, xcv, ycv, minimum_splits=[5,10,100,500
], maximum_depth=[1, 5, 10, 50, 100, 500]):
```

```
        self.xtr = xtr
        self.ytr = ytr
        self.xcv = xcv
        self.ycv = ycv
        self.minimum_splits = minimum_splits
        self.maximum_depth = maximum_depth

    #creating a method of calling DT classifier:
    def classfier(self, auc_dict_cv={}, auc_dict_tr={}):
        for splits in self.minimum_splits:
            for depths in self.maximum_depth:
                clf = DecisionTreeClassifier(max_depth=depths, min_samp
les_split=splits, class_weight='balanced')
                print(depths, splits)
                clf.fit(self.xtr, self.ytr)
                y_pred_cv = clf.predict(self.xcv)

                #performance metric on CV data:
                fpr_cv, tpr_cv, thresholds_cv = roc_curve(ycv, y_pred_c
v)
                auc_val = auc(fpr_cv, tpr_cv)
                auc_dict_cv[zip([splits], [depths])] = auc_val

                #performance metrics for training data:
                y_pred_tr = clf.predict(self.xtr)
                fpr_tr, tpr_tr, thresholds_tr = roc_curve(ytr, y_pred_t
r)
                auc_val = auc(fpr_tr, tpr_tr)
                auc_dict_tr[zip([splits], [depths])] = auc_val

        return auc_dict_tr, auc_dict_cv
```

Tfidf-w2v decision tree instance creation on training and CV data

In [14]:
```
import time
start = time.time()
```

```python
tfidf_w2v_instance = decisiontree(xtr, ytr, xcv, ycv)

dictionary_train, dictionary_cv = tfidf_w2v_instance.classfier()
end = time.time()
print()
print('time is(in seconds): ', end - start)
```

```
1 5
5 5
10 5
50 5
100 5
500 5
1 10
5 10
10 10
50 10
100 10
500 10
1 100
5 100
10 100
50 100
100 100
500 100
1 500
5 500
10 500
50 500
100 500
500 500

time is(in seconds):  2606.405438184738
```

In [15]:
```python
#creating dictionary :

train_list = [(list(x), np.round(y,3)) for x, y in dictionary_train.ite
ms()]
cv_list = [(list(x), np.round(y,3)) for x, y in dictionary_cv.items()]
```

```
print(train_list[0])
print()
print(cv_list[0])
```

```
([(5, 100)], 0.998)

([(500, 50)], 0.567)
```

sorting the list based off AUC score

In [16]:

```
tr_list = sorted(train_list, key=lambda x: x[1], reverse=True)
cv_list = sorted(cv_list, key=lambda x: x[1], reverse=True)
print('sorted train list score based off AUC score is:\n', tr_list[0:3
])
print('******************************************')
print()
print('sorted CV list score based off AUC score is:\n', cv_list[0:3])
```

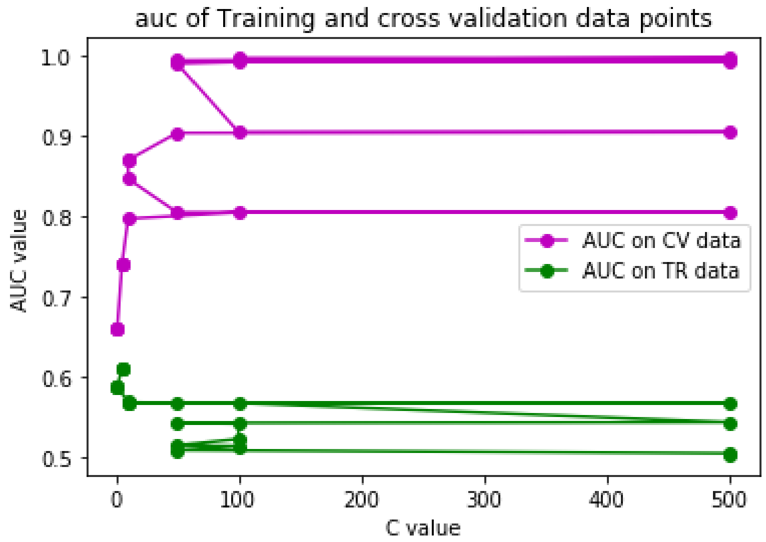
```
sorted train list score based off AUC score is:
 [([(5, 100)], 0.998), ([(5, 500)], 0.998), ([(5, 50)], 0.995)]
****************************************

sorted CV list score based off AUC score is:
 [([(5, 5)], 0.61), ([(500, 5)], 0.61), ([(100, 5)], 0.61)]
```

Plotting AUC Curve on training and cv data based off depth

In [17]:

```
plt.plot([x[0][0][1] for x in tr_list], [x[1] for x in tr_list], linest
yle='-', color='m', marker='o',label='AUC on CV data')
plt.plot([x[0][0][1] for x in cv_list], [x[1] for x in cv_list], linest
yle='-', color='g', marker='o', label='AUC on TR data')
#plt.plot([0, 1], [0, 1], linestyle='--')
plt.xlabel("C value")
plt.ylabel('AUC value')
plt.title('auc of Training and cross validation data points')
plt.legend()
```

Out[17]: <matplotlib.legend.Legend at 0x7fa89b8a9d68>



Optimal Tfidf-W2V- DecesionTree[depth = 5 and splits = 5]

In [18]:

```
dt = DecisionTreeClassifier(max_depth=5, min_samples_split=5, class_wei
ght='balanced')
dt.fit(xtr, ytr)

ypred = dt.predict_proba(xtest)
ypred_tr = dt.predict_proba(xtr)


fpr_tr, tpr_tr, thresholds_tr= roc_curve(ytr, ypred_tr[:,1])
auc_tr = auc(fpr_tr, tpr_tr)


fpr_test, tpr_test, thresholds_test= roc_curve(ytest, ypred[:,1])
auc_test = auc(fpr_test, tpr_test)
```

```
print(auc_test)
```

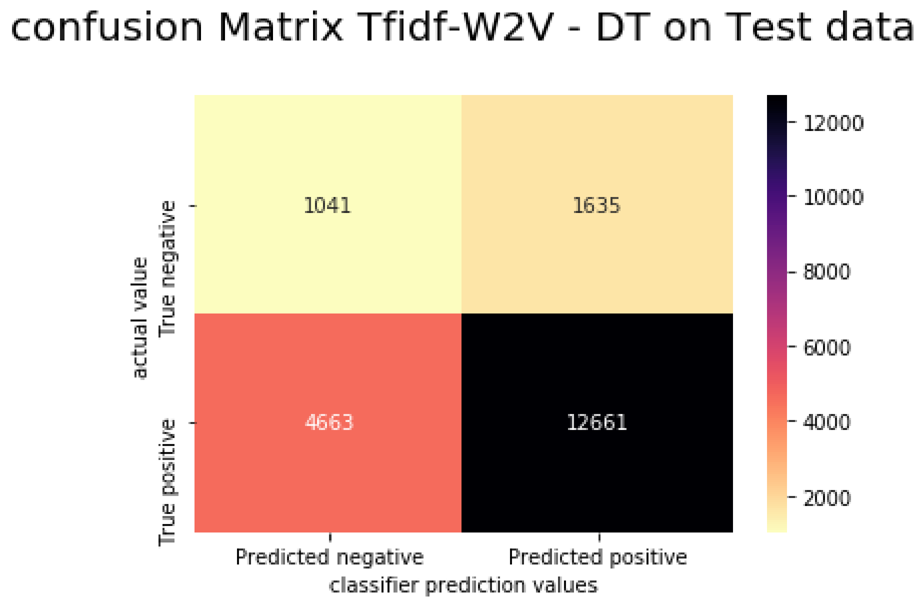
```
0.5587068010750184
```

plotting confusion matrix on test data

In [19]:

```
%time
ypred = np.where(ypred[:,1] < 0.5, 0, 1)
#creating confusion matrix:

cf = confusion_matrix(ytest, ypred)
labels = ['True negative', 'True positive']

df_cf = pd.DataFrame(cf, index=labels, columns=['Predicted negative',
'Predicted positive'])
sns.heatmap(df_cf, annot=True,fmt='3d', cmap='magma_r')

plt.title("confusion Matrix Tfidf-W2V - DT on Test data\n", size=20)
plt.xlabel("classifier prediction values")
plt.ylabel("actual value")
plt.show()
```

```
CPU times: user 0 ns, sys: 0 ns, total: 0 ns
Wall time: 6.44 µs
```



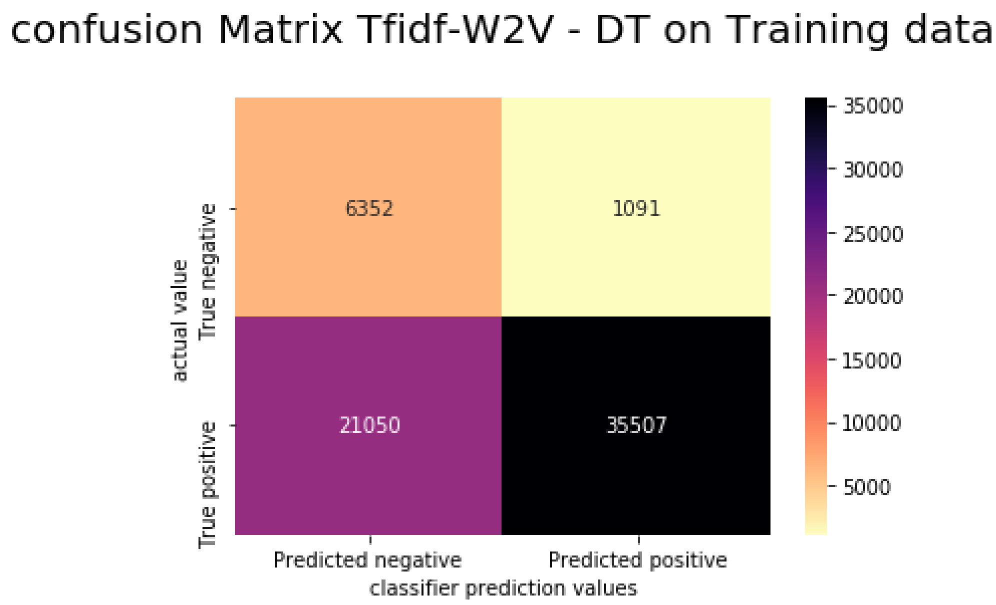
plotting confusion matrix on training data

In [20]:

```
%time
y_pred_tr = np.where(ypred_tr[:,1] < 0.5, 0, 1)
#creating confusion matrix:

cf = confusion_matrix(ytr, y_pred_tr)
labels = ['True negative', 'True positive']

df_cf = pd.DataFrame(cf, index=labels, columns=['Predicted negative',
'Predicted positive'])
sns.heatmap(df_cf, annot=True,fmt='3d', cmap='magma_r')

plt.title("confusion Matrix Tfidf-W2V - DT on Training data\n", size=20
)
plt.xlabel("classifier prediction values")
plt.ylabel("actual value")
plt.show()
```

```
CPU times: user 0 ns, sys: 0 ns, total: 0 ns
```

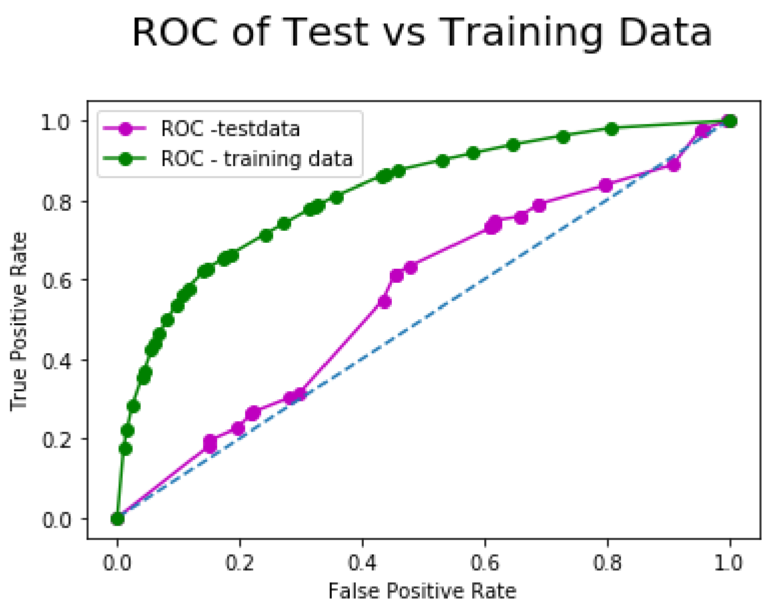
```
Wall time: 6.44 µs
```



plotting ROC curve on test data and training data

In [21]:
```
plt.plot(fpr_test, tpr_test, color='m', marker='o',label='ROC -testdat
a')
plt.plot([0, 1], [0, 1], linestyle='--')
plt.plot(fpr_tr, tpr_tr, linestyle='-', color='g', marker='o', label='R
OC - training data')
plt.xlabel('False Positive Rate')
plt.ylabel('True Positive Rate')
plt.title('ROC of Test vs Training Data\n', size=20)
plt.legend()
```

Out[21]:
```
<matplotlib.legend.Legend at 0x7fa89afdc0f0>
```



Decision Tree classifier performance consolidation of 4 vectorizers

In [22]:
```
perf_dict = dict(algorithm = ['Bow', 'Tfidf', 'W2V', 'Tfidf-W2V'],
                 AUC = [0.846, 0.795, 0.661, 0.558],
                 Depth = [50, 50, 5, 5],
                 Splits =[500, 500, 100, 5])

perf_df = pd.DataFrame(perf_dict)
perf_df
```

Out[22]:

	AUC	Depth	Splits	algorithm
0	0.846	50	500	Bow
1	0.795	50	500	Tfidf

	AUC	Depth	Splits	algorithm
2	0.661	5	100	W2V
3	0.558	5	5	Tfidf-W2V

Conclusions -- out of 1L datapoints:

1. Bow AUC performace was best amongts all the four vectorizer with ~85 % AUC score.
2. TFidf-w2v performace was best amongts all the four vectorizer with ~56 % AUC score.
3. for BoW and TFIDF seperate PDF file was created for graphical visualization of depth 3 only.