Assignment - To apply RF and XGBOOST claasifer algo on all the four vectorizers

Since the data is already preprocessed and saved onto disc, I would be working on the preprocessed data for the given assignment

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.sqlite')
df = pd.read_sql_query("""select * from reviews""", con)
```

```
CPU times: user 1.21 s, sys: 852 ms, total: 2.06 s
Wall time: 2.06 s
```

In [4]:
```
len(df)
```

Out[4]:
```
100000
```

In [5]:
```
df.columns
```

Out[5]:
```
Index(['index', 'Id', 'ProductId', 'UserId', 'ProfileName',
       'HelpfulnessNumerator', 'HelpfulnessDenominator', 'Score', 'Time',
       'Summary', 'Text', 'CleanedText_Bow', 'ClenedText_W2Vtfdf', 'Bow_feat',
       'Bow_new_feat', 'w2v_feat', 'w2v_new_feat'],
      dtype='object')
```

In [6]:
```
#op of BoW cleaned text after preprocessing:
df.CleanedText_Bow.head(2)
```

Out[6]:
```
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 [7]:
```
#op of BoW feature engineered text column after preprocessing:
df.Bow_new_feat.head(2)
```

Out[7]:
```
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
```

sorting dataframe based on time

In [4]:
```
#sorting the datframe based on time:
print(len(df))
df = df.sort_values('Time', ascending=True)
print()
df['Time'].head(8)
```

```
100000
```

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

In [9]:
```
df.Score.value_counts()
```

Out[9]:
```
1    87729
0    12271
Name: Score, dtype: int64
```

1. RF - BoW

train test split

In [10]:
```
xtrain, xtest, ytrain, ytest = train_test_split(df.CleanedText_Bow, df.
Score, test_size=0.2, shuffle=False)
xtr, xcv, ytr, ycv = train_test_split(xtrain, ytrain, test_size=0.2, sh
uffle=False)
```

BoW object instantiation

In [11]:
```
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, 29812)
(16000, 29812)
(20000, 29812)
```

In [12]:
```
os.cpu_count()
```

Out[12]: 1

BoW - RF instance creation on training and CV data

In [13]:
```
class RandomForest:

    '''building the Random Forest classifier based off hypeparameters n
o_of_base_models and max_depth of base_models'''

    #instantiating the instance attributes:
    def __init__(self, xtr, ytr, xcv, ycv, maximum_depth=[1, 3, 5, 7, 9
, 11, 13, 15], estimators=[5,7,13,9,11,13,15]):
        self.xtr = xtr
        self.ytr = ytr
        self.xcv = xcv
        self.ycv = ycv
        self.estimators = estimators
        self.maximum_depth = maximum_depth

    #creating a method of calling RF classifier:
    def classfier(self, auc_dict_cv={}, auc_dict_tr={}):
        for estimator in self.estimators:
            for depths in self.maximum_depth:
                clf = RandomForestClassifier(max_depth=depths, n_estima
tors=estimator, oob_score=True)
                print(depths, estimator)
                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[auc_val] = [estimator, depths]

                #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[auc_val] = [estimator, depths]
```

```
        return auc_dict_tr, auc_dict_cv
```

In [14]:
```
%time
#importing random forest classifier:
from sklearn.ensemble import RandomForestClassifier

BoW_instance = RandomForest(xtr, ytr, xcv, ycv)

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

```
CPU times: user 0 ns, sys: 0 ns, total: 0 ns
Wall time: 6.2 µs
1 5
/usr/local/lib/python3.5/site-packages/sklearn/ensemble/weight_boostin
g.py:29: DeprecationWarning: numpy.core.umath_tests is an internal NumP
y module and should not be imported. It will be removed in a future Num
Py release.
  from numpy.core.umath_tests import inner1d
3 5
5 5
7 5
9 5
11 5
13 5
15 5
1 7
3 7
5 7
7 7
9 7
11 7
13 7
15 7
1 13
3 13
5 13
7 13
```

```
9 13
11 13
13 13
15 13
1 9
3 9
5 9
7 9
9 9
11 9
13 9
15 9
1 11
3 11
5 11
7 11
9 11
11 11
13 11
15 11
1 13
3 13
5 13
7 13
9 13
11 13
13 13
15 13
1 15
3 15
5 15
7 15
9 15
11 15
13 15
15 15
```

In [15]:
```
train_list = [[np.round(x, 5), y] for x, y in dictionary_train.items()]
```

```
cv_list = [[np.round(x, 5), y] for x, y in dictionary_cv.items()]

print(train_list[0])

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

```
[0.54274, [5, 1]]

[0.55308, [5, 1]]
```

sorting the data based off AUC score

In [16]:
```
tr_list = sorted(train_list, key=lambda x: x[0], reverse=True)
cv_list = sorted(cv_list, key=lambda x: x[0], 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:
 [[0.88212, [15, 13]], [0.87603, [15, 15]], [0.87196, [13, 15]]]
****************************************

sorted CV list score based off AUC score is:
 [[0.85593, [15, 13]], [0.84809, [15, 15]], [0.84443, [13, 15]]]
```

In [17]:
```
len(tr_list)
```

Out[17]:
```
56
```

creating dataframe for plotting different CV value of AUC , depth and estimator models

In [18]:
```
auc_scores = []
no_base_modesl = []
no_depths = []
```

```
for i in range(len(cv_list)):
    auc_scores.append(cv_list[i][0])
    no_base_modesl.append(cv_list[i][1][0])
    no_depths.append(cv_list[i][1][1])

df_dict = dict(base_models=no_base_modesl,
               depths=no_depths,
               auc_val=auc_scores)
d_dframe = pd.DataFrame(df_dict)
d_dframe.head(4)
```

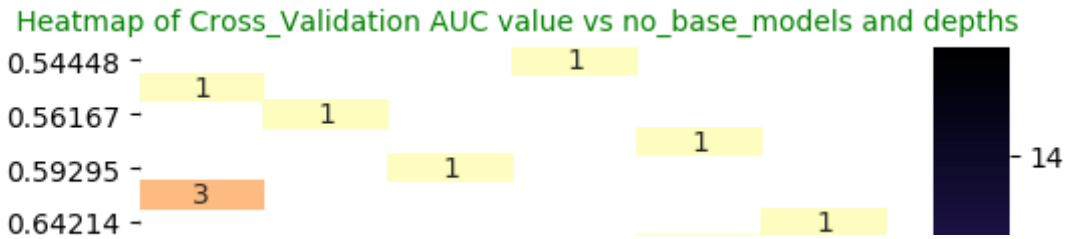
Out[18]:

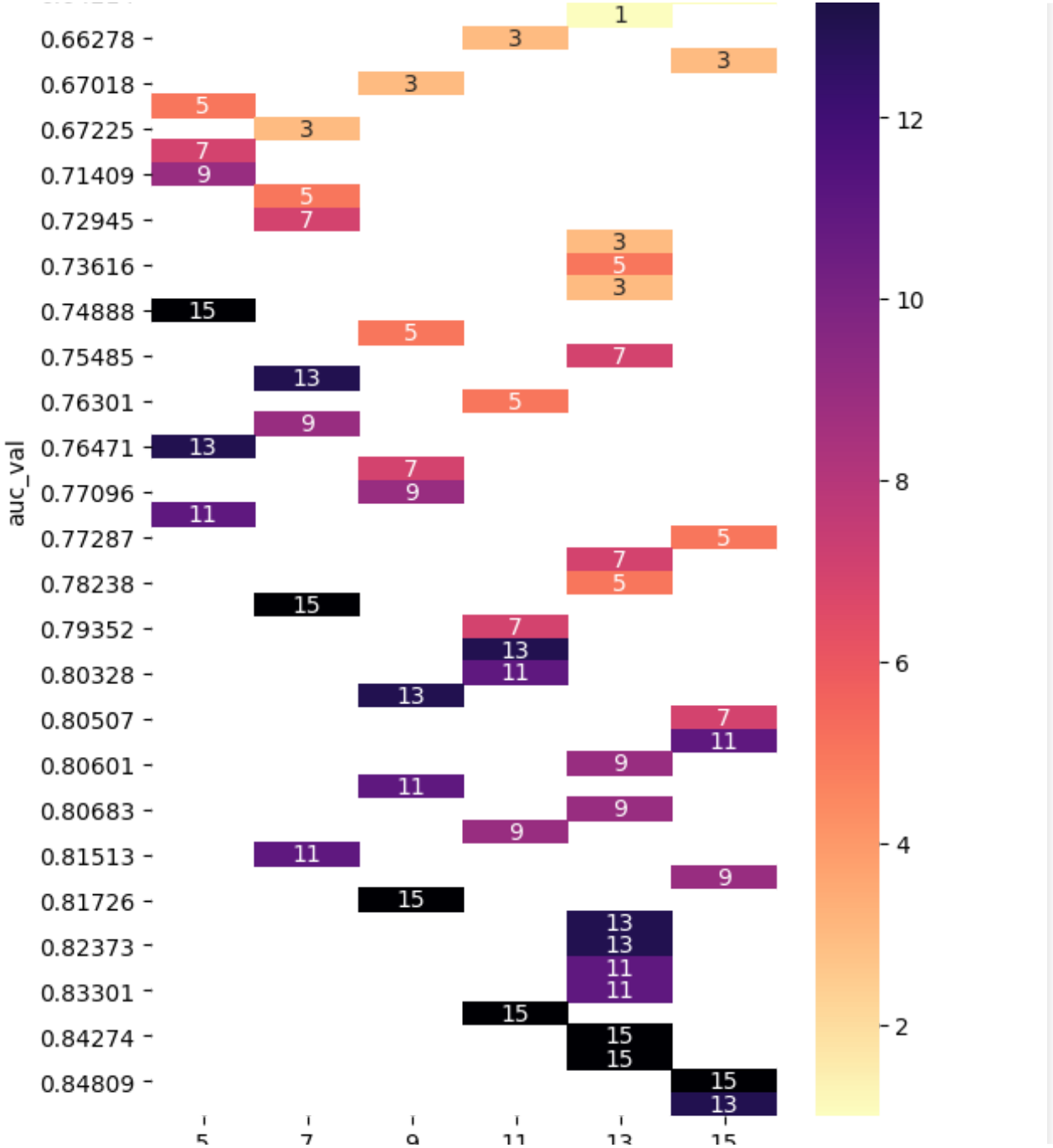
	auc_val	base_models	depths
0	0.85593	15	13
1	0.84809	15	15
2	0.84443	13	15
3	0.84274	13	15

Plotting heatmap of CV data of AUC vs no of base models and depths

In [19]:
```
fig = plt.figure(figsize=(6, 10), dpi=100)
d_pivot = d_dframe.pivot('auc_val','base_models', 'depths')
sns.heatmap(d_pivot, annot=True, cmap='magma_r')
plt.title(' Heatmap of Cross_Validation AUC value vs no_base_models and
 depths', size=10, color='green')
```

Out[19]:
```
Text(0.5, 1.0, ' Heatmap of Cross_Validation AUC value vs no_base_model
s and depths')
```



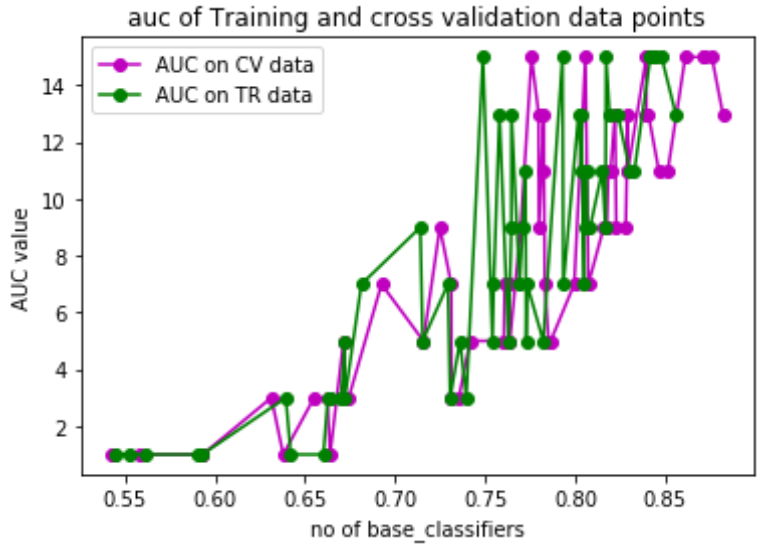


base_models

Plotting AUC Curve on training and cv data based off depth

In [20]:
```
plt.plot([x[0] for x in tr_list], [x[1][1] for x in tr_list], linestyle
='-', color='m', marker='o',label='AUC on CV data')
plt.plot([x[0] for x in cv_list], [x[1][1] for x in cv_list], linestyle
='-', color='g', marker='o', label='AUC on TR data')
#plt.plot([0, 1], [0, 1], linestyle='--')
plt.xlabel("no of base_classifiers")
plt.ylabel('AUC value')
plt.title('auc of Training and cross validation data points')
plt.legend()
```

Out[20]: <matplotlib.legend.Legend at 0x7f9f782c94a8>



1. OptimalBoW - RF[depth = 13 and basemodels = 15]

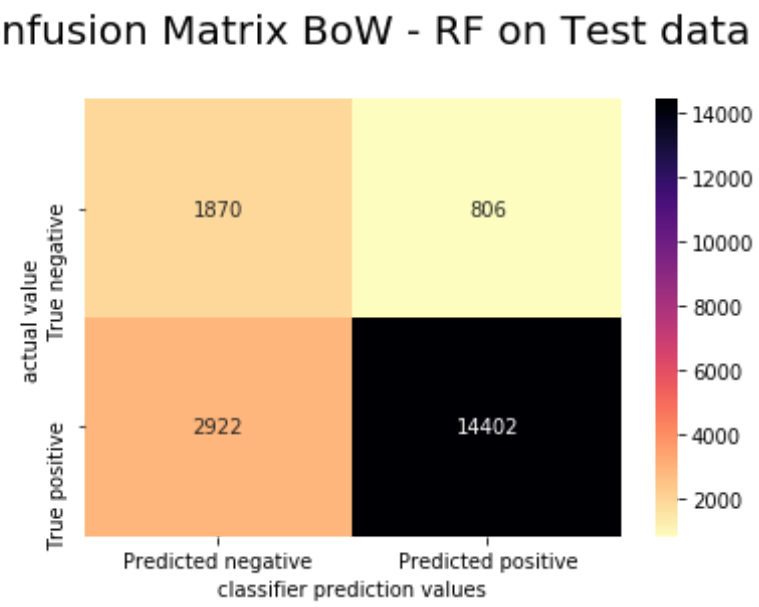
In [21]:
```
rf = RandomForestClassifier(max_depth=13, n_estimators=15, class_weight
='balanced')
rf.fit(xtr, ytr)

#prediction on training data:
y_pred_tr = rf.predict_proba(xtr)
fpr_tr, tpr_tr, thresholds_tr= roc_curve(ytr, y_pred_tr[:,1])
auc_tr = auc(fpr_tr, tpr_tr)

#prediction on test data:
ypred = rf.predict_proba(xtest)
fpr_test, tpr_test, thresholds_test= roc_curve(ytest, ypred[:,1])
auc_test = auc(fpr_test, tpr_test)

print(auc_test)
```

```
0.8435879172089559
```

plotting confusion matrix on test data

In [22]:
```
%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 - RF on Test data\n", size=20)
plt.xlabel("classifier prediction values")
plt.ylabel("actual value")
plt.show()
```

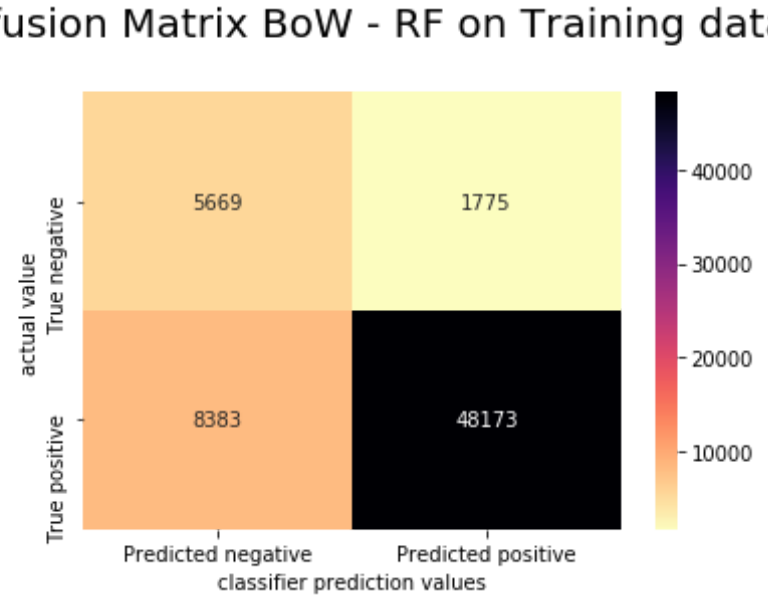
```
CPU times: user 0 ns, sys: 0 ns, total: 0 ns
Wall time: 6.44 µs
```



plotting confusion matrix on training data

In [23]:
```
%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 BoW - RF on Training data\n", size=20)
plt.xlabel("classifier prediction values")
plt.ylabel("actual value")
plt.show()
```

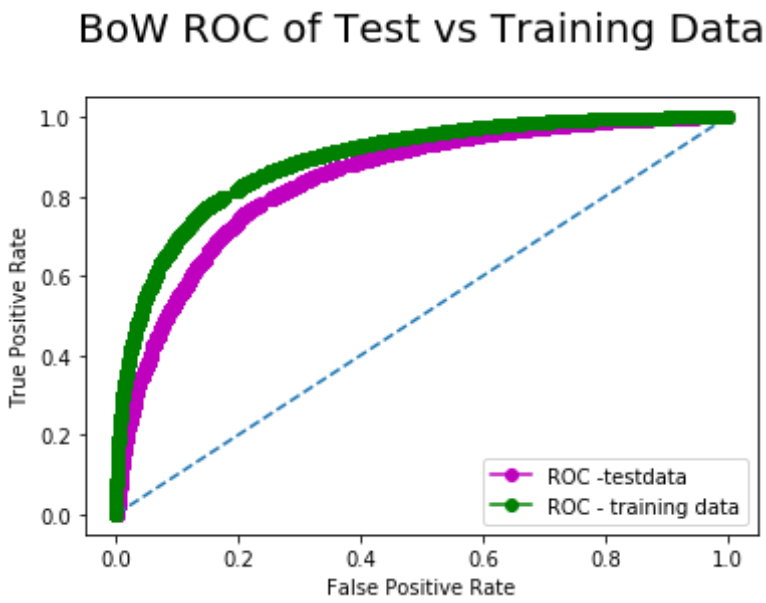
```
CPU times: user 0 ns, sys: 0 ns, total: 0 ns
Wall time: 6.44 µs
```



plotting ROC curve on test data and training data

In [24]:
```
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('BoW ROC of Test vs Training Data\n', size=20)
plt.legend()
```

Out[24]:
```
<matplotlib.legend.Legend at 0x7f9f75290f60>
```



important features for positive and negative reviews

In [25]:
```
features = bow_object.get_feature_names()
featuresAndcoeff = sorted(zip(rf.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
```

```
 0.03720678049914402                not
                 0.0       aaaaaaaaagghh
 0.03112867415343571               would
                 0.0              aaaaah
0.030172031917479716               great
                 0.0 aaaaahhhhhhhhhhhhhhh
 0.029582446256334893               best
                 0.0              aaaah
0.029495845013799534              delici
                 0.0              aaah
0.029134259497958122              return
                 0.0             aachen
0.022993920719890093             perfect
                 0.0               aad
 0.02176874081766172               love
                 0.0              aadp
0.021675569005422016               nice
                 0.0             aafco
0.016945143540645833                bad
                 0.0              aagh
0.015392929128202207               wast
                 0.0               aah
0.013350576737147544              bland
                 0.0              aahh
 0.01308493304703448               away
                 0.0              aand
0.011571253063845707               keep
                 0.0           aardvark
0.011350439393212779              compani
                 0.0                 ab
0.010795778347092416              stale
                 0.0              aback
0.010739876822661778               item
                 0.0            abandon
0.010591538568098522               poor
                 0.0            abaolut
0.010173271675200827               find
                 0.0           abattoir
0.010151966634456703             thought
                 0.0               abba
```

```
0.009490147545727004              email
                0.0             abbey
0.009024688528102546              guess
                0.0              abbi
0.008851123173408516              thank
                0.0           abbrevi
 0.00868018915856111            disgust
                0.0               abc
0.007625183844275655            terribl
```

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 [26]:
```
features = bow_object.get_feature_names()
featuresAndcoeff = sorted(zip(rf.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[26]:

	top25_neg	top25_pos
0	[1.1754936182607552e-06, purina]	[0.03720678049914402, not]
1	[1.119503755430242e-06, colorado]	[0.03112867415343571, would]
2	[1.1167861569477255e-06, believ]	[0.030172031917479716, great]

	top25_neg	top25_pos
3	[1.0772362169897143e-06, cartlidg]	[0.029582446256334893, best]
4	[1.0772362169890502e-06, joke]	[0.029495845013799534, delici]
5	[1.0551623497611525e-06, pastri]	[0.029134259497958122, return]
6	[1.0523245516374902e-06, overwhelm]	[0.029293920719890093, perfect]
7	[1.0477965815791225e-06, fragrant]	[0.02176874081766172, love]
8	[1.045308670497818e-06, improp]	[0.021675569005422016, nice]
9	[1.0043092868371666e-06, weird]	[0.016945143540645833, bad]
10	[9.419596307973705e-07, touch]	[0.015392929128202207, wast]
11	[8.293828986316791e-07, sheet]	[0.013350576737147544, bland]
12	[7.389826606651589e-07, tripl]	[0.01308493304703448, away]
13	[7.315333909514978e-07, blackberri]	[0.011571253063845707, keep]
14	[7.282259153996223e-07, vine]	[0.011350439393212779, compani]
15	[5.839064116007434e-07, yellowish]	[0.010795778347092416, stale]
16	[4.749445100890555e-07, popular]	[0.010739876822661778, item]
17	[4.3791089427800616e-07, dose]	[0.010591538568098522, poor]
18	[4.138034720529871e-07, crack]	[0.010173271675200827, find]
19	[2.7156491466996766e-07, experi]	[0.010151966634456703, thought]
20	[1.8082748814359065e-07, individualist]	[0.009490147545727004, email]
21	[1.3922680059819814e-07, whoever]	[0.009024688528102546, guess]
22	[1.3109020518206562e-07, shall]	[0.008851123173408516, thank]
23	[1.2936355796734093e-07, ive]	[0.00868018915856111, disgust]
24	[3.884747368190756e-08, toler]	[0.007625183844275655, terribl]

Feature Engineering on BoW

In [27]:
```
df.columns
```

Out[27]:
```
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')
```

I will be taking "Bow_new_feat" column as this is the combination of two columns: "Reviews" and 'Summary'

In [28]:
```
df.Bow_new_feat.head(2)
```

Out[28]:
```
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
```

Train CV test split

In [29]:
```
xtrain, xtest, ytrain, ytest = train_test_split(df.Bow_new_feat, df.Sco
re, test_size=0.2, shuffle=False)
xtr, xcv, ytr, ycv = train_test_split(xtrain, ytrain, test_size=0.2, sh
uffle=False)
```

BoW object instantiation

In [30]:
```
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)
```

BoW - RF instance creation on training and CV data

In [31]:

```
class RandomForest:

    '''building the Random Forest classifier based off hypeparameters n
o_of_base_models and max_depth of base_models'''

    #instantiating the instance attributes:
    def __init__(self, xtr, ytr, xcv, ycv, maximum_depth=[1, 3, 5, 7, 9
, 11, 13, 15], estimators=[5,7,13,9,11,13,15]):
        self.xtr = xtr
        self.ytr = ytr
        self.xcv = xcv
        self.ycv = ycv
        self.estimators = estimators
        self.maximum_depth = maximum_depth

    #creating a method of calling RF classifier:
    def classfier(self, auc_dict_cv={}, auc_dict_tr={}):
        for estimator in self.estimators:
            for depths in self.maximum_depth:
                clf = RandomForestClassifier(max_depth=depths, n_estima
tors=estimator, oob_score=True)
                print(depths, estimator)
                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[auc_val] = [estimator, depths]

                #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[auc_val] = [estimator, depths]

        return auc_dict_tr, auc_dict_cv
```

In [32]:

```
%time
#importing random forest classifier:
from sklearn.ensemble import RandomForestClassifier

BoW_instance = RandomForest(xtr, ytr, xcv, ycv)

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

```
CPU times: user 0 ns, sys: 0 ns, total: 0 ns
Wall time: 6.68 µs
1 5
3 5
5 5
7 5
9 5
11 5
13 5
15 5
1 7
3 7
5 7
7 7
9 7
11 7
13 7
15 7
1 13
```

```
3 13
5 13
7 13
9 13
11 13
13 13
15 13
1 9
3 9
5 9
7 9
9 9
11 9
13 9
15 9
1 11
3 11
5 11
7 11
9 11
11 11
13 11
15 11
1 13
3 13
5 13
7 13
9 13
11 13
13 13
15 13
1 15
3 15
5 15
7 15
9 15
11 15
13 15
15 15
```

sorting the data based off AUC score

In [33]:
```
train_list = [[np.round(x, 5), y] for x, y in dictionary_train.items()]
cv_list = [[np.round(x, 5), y] for x, y in dictionary_cv.items()]

print(train_list[0])

print()
print(cv_list[0])

tr_list = sorted(train_list, key=lambda x: x[0], reverse=True)
cv_list = sorted(cv_list, key=lambda x: x[0], 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])
```

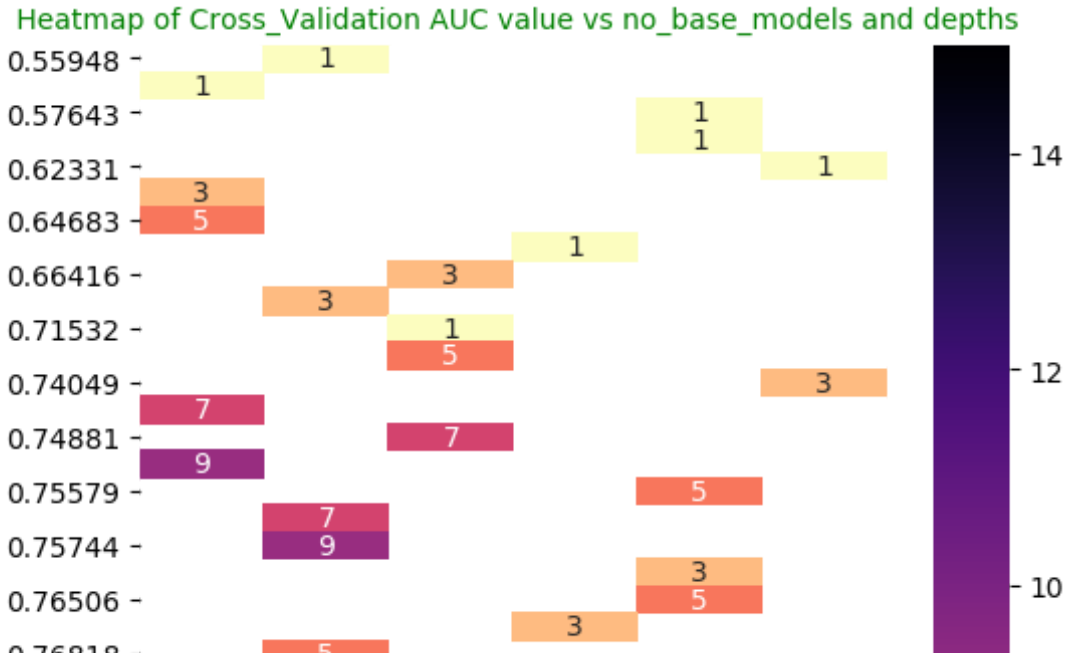
```
[0.57621, [5, 1]]

[0.57413, [5, 1]]
sorted train list score based off AUC score is:
 [[0.89061, [15, 13]], [0.89049, [11, 15]], [0.88402, [13, 15]]]
****************************************

sorted CV list score based off AUC score is:
 [[0.87443, [15, 13]], [0.86533, [13, 11]], [0.86511, [11, 15]]]
```

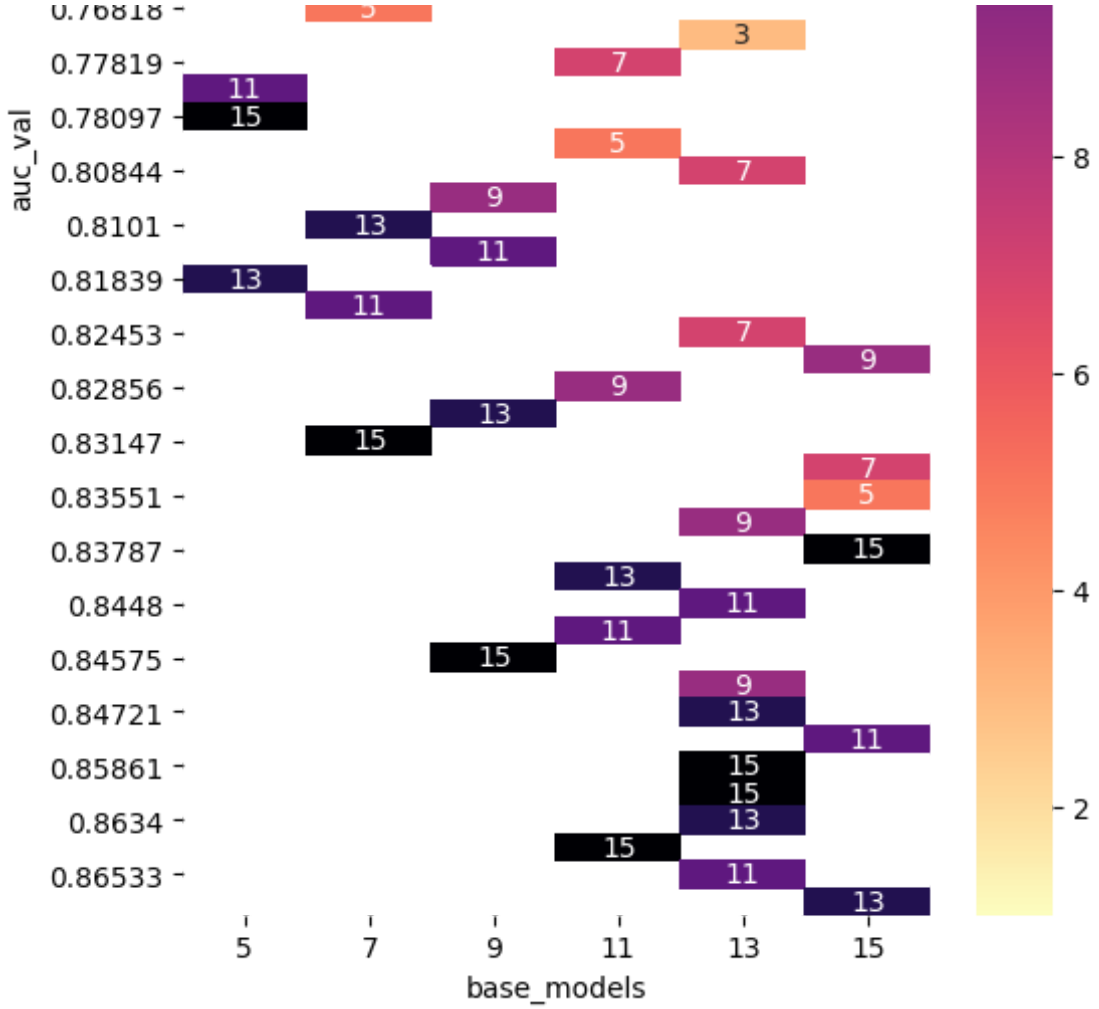
Plotting heatmap of CV data of AUC vs no of base models and depths

In [34]:
```
#creating dataframe for pivot table:
auc_scores = []
no_base_modesl = []
no_depths = []

for i in range(len(cv_list)):
    auc_scores.append(cv_list[i][0])
```

```
    no_base_modesl.append(cv_list[i][1][0])
    no_depths.append(cv_list[i][1][1])

df_dict = dict(base_models=no_base_modesl,
               depths=no_depths,
               auc_val=auc_scores)
d_dframe = pd.DataFrame(df_dict)
d_dframe.head(4)



#plotting heatmap:
fig = plt.figure(figsize=(6, 10), dpi=100)
d_pivot = d_dframe.pivot('auc_val','base_models', 'depths')
sns.heatmap(d_pivot, annot=True, cmap='magma_r')
plt.title(' Heatmap of Cross_Validation AUC value vs no_base_models and
 depths', size=10, color='green')
```

Out[34]: Text(0.5, 1.0, ' Heatmap of Cross_Validation AUC value vs no_base_model
s and depths')



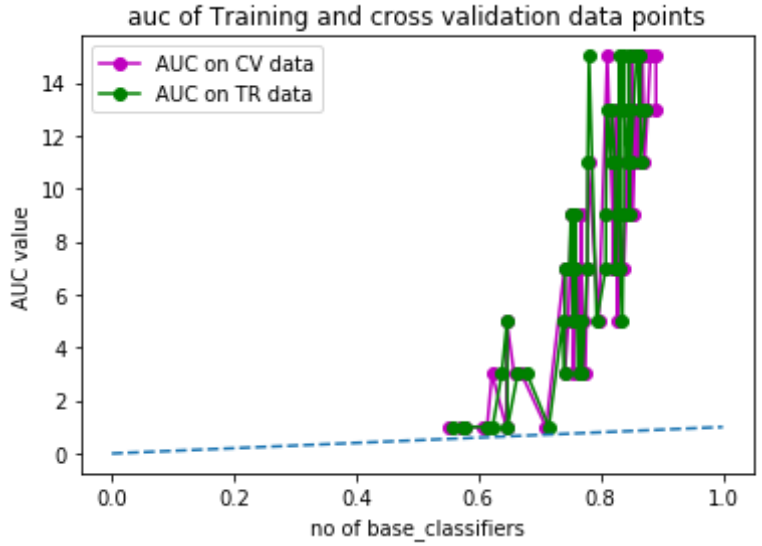


Plotting AUC Curve on training and cv data based off depth

In [35]:
```
plt.plot([x[0] for x in tr_list], [x[1][1] for x in tr_list], linestyle
='-', color='m', marker='o',label='AUC on CV data')
plt.plot([x[0] for x in cv_list], [x[1][1] for x in cv_list], linestyle
='-', color='g', marker='o', label='AUC on TR data')
plt.plot([0, 1], [0, 1], linestyle='--')
plt.xlabel("no of base_classifiers")
```

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

Out[35]: <matplotlib.legend.Legend at 0x7f9f71ec57f0>



1. OptimalBoW - RF[depth = 13 and basemodels = 15] on Featured enginnering column

In [36]:
```
rf = RandomForestClassifier(max_depth=13, n_estimators=15, class_weight
='balanced')
rf.fit(xtr, ytr)

#prediction on training data:
y_pred_tr = rf.predict_proba(xtr)
fpr_tr, tpr_tr, thresholds_tr= roc_curve(ytr, y_pred_tr[:,1])
auc_tr = auc(fpr_tr, tpr_tr)

#prediction on test data:
ypred = rf.predict_proba(xtest)
```

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

print(auc_test)
```

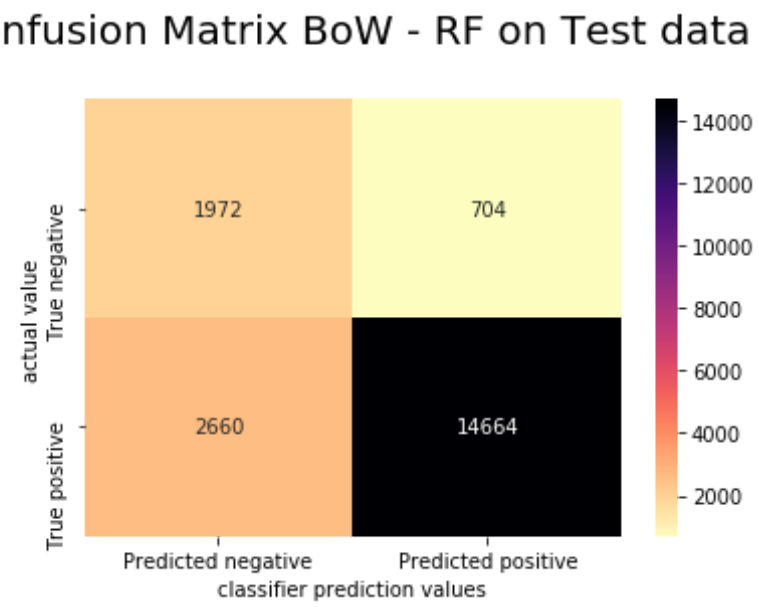
```
0.8758911749306889
```

plotting confusion matrix on test data

In [37]:

```
%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 - RF on Test data\n", size=20)
plt.xlabel("classifier prediction values")
plt.ylabel("actual value")
plt.show()
```

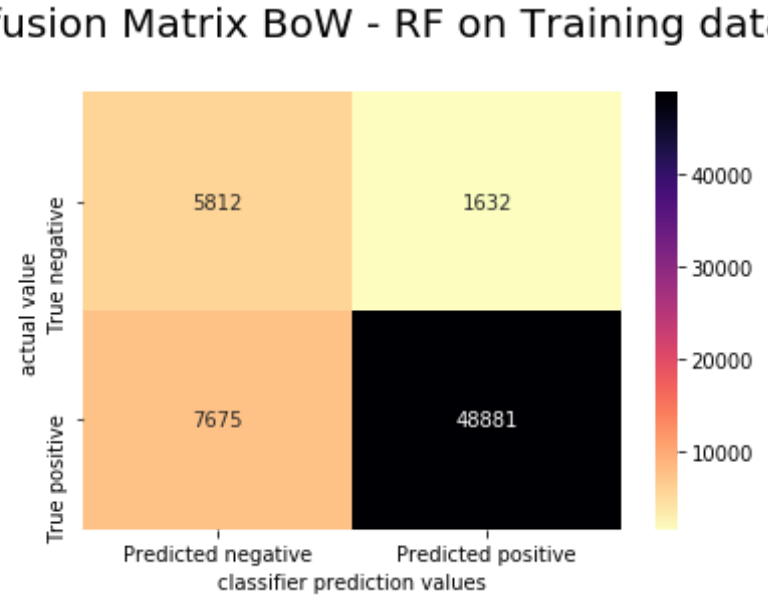
```
CPU times: user 0 ns, sys: 0 ns, total: 0 ns
Wall time: 6.44 µs
```



plotting confusion matrix on training data

In [38]:
```
%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 BoW - RF on Training data\n", size=20)
plt.xlabel("classifier prediction values")
plt.ylabel("actual value")
plt.show()
```

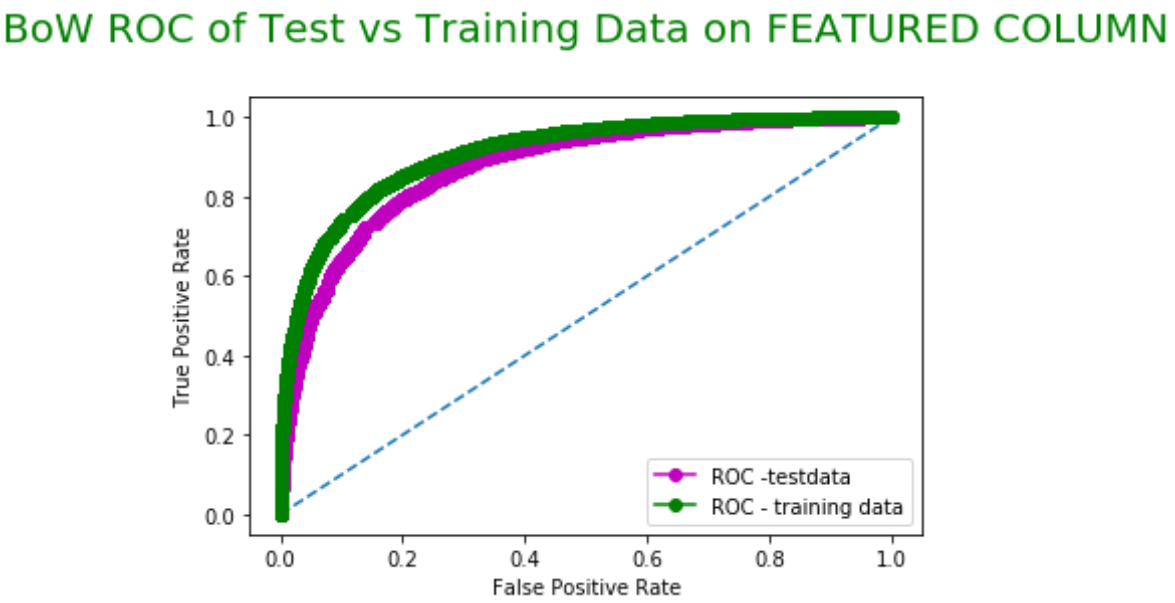
```
CPU times: user 0 ns, sys: 0 ns, total: 0 ns
Wall time: 6.2 µs
```



plotting ROC curve on test data and training data

In [39]:
```
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('BoW ROC of Test vs Training Data on FEATURED COLUMN\n', size
=20, color='green')
plt.legend()
```

Out[39]:
```
<matplotlib.legend.Legend at 0x7f9f705170b8>
```



important features for positive and negative reviews

In [40]:
```
features = bow_object.get_feature_names()
featuresAndcoeff = sorted(zip(rf.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
 0.06171946913594278                  not
                0.0         aaaaaaaaagghh
 0.03569732959491379                 love
                0.0              aaaaaagh
 0.03316434400192405                 best
                0.0               aaaaah
0.027997494494580014                excel
                0.0 aaaaahhhhhhhhhhhhhhh
 0.025484348847760237               delici
                0.0                aaaah
0.022668263977738138                yummi
                0.0                 aaah
 0.02058609958370873                worst
                0.0               aaamaz
0.018834190018347272                receiv
                0.0               aachen
0.018667256998553945                return
                0.0                 aad
0.016198999327574216               terribl
                0.0                 aadp
0.014889903786834715               favorit
                0.0                aafco
0.014874838059549752                  poor
                0.0                 aagh
0.013812739261322572               perfect
                0.0                  aah
0.013513446972501652                refund
                0.0                 aahh
0.013080160603422014              unfortun
                0.0                 aand
0.012393302785014909                  amaz
                0.0             aardvark
0.009691136096974832                money
                0.0                   ab
 0.00895500410419194               horribl
                0.0                aback
0.008002937753678673                  easi
                0.0               abacor
```

```
0.007932562005754202                rich
                 0.0              abaloo
0.007909578643068628            tasteless
                 0.0             abandon
0.007434017807440536                 nice
                 0.0             abaolut
 0.00728045775722417               product
                 0.0            abattoir
0.007001213598140437                 mayb
                 0.0               abba
0.006894815255388098                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 [41]:
```
features = bow_object.get_feature_names()
featuresAndcoeff = sorted(zip(rf.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[41]:

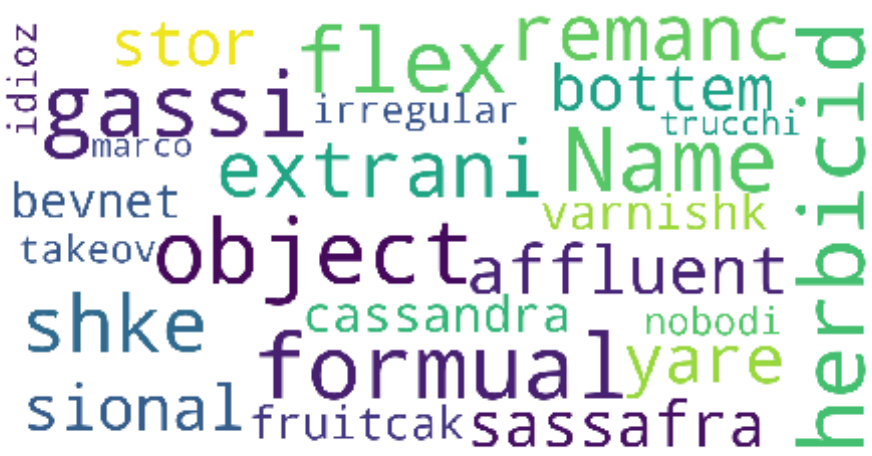
	top25_neg	top25_pos
0	[9.697001613524187e-07, toast]	[0.06171946913594278, not]
1	[9.29276549007025e-07, sympathet]	[0.03569732959491379, love]

	top25_neg	top25_pos
2	[8.810536747917336e-07, belief]	[0.03316434400192405, best]
3	[8.460658424392607e-07, hibiscus]	[0.027997494494580014, excel]
4	[8.342724579631245e-07, entertain]	[0.025484348847760237, delici]
5	[7.656045656912377e-07, juevo]	[0.022668263977738138, yummi]
6	[6.552504081143158e-07, emerg]	[0.02058609958370873, worst]
7	[6.027780678997406e-07, somehow]	[0.018834190018347272, receiv]
8	[5.671840499126144e-07, wean]	[0.018667256998553945, return]
9	[5.655393775697265e-07, reason]	[0.016198999327574216, terribl]
10	[5.563539746762475e-07, result]	[0.014889903786834715, favorit]
11	[5.421145810710712e-07, metal]	[0.014874838059549752, poor]
12	[5.392422477925466e-07, bunch]	[0.013812739261322572, perfect]
13	[4.851531238317594e-07, issu]	[0.013513446972501652, refund]
14	[4.3385601923541095e-07, fruit]	[0.013080160603422014, unfortun]
15	[3.868353513650867e-07, culprit]	[0.012393302785014909, amaz]
16	[3.791970029871709e-07, roizen]	[0.009691136096974832, money]
17	[2.7225760654093073e-07, lift]	[0.00895500410419194, horribl]
18	[2.4578432225365336e-07, tough]	[0.008002937753678673, easi]
19	[2.2112117016823037e-07, allow]	[0.007932562005754202, rich]
20	[2.1108405685238384e-07, dinner]	[0.007909578643068628, tasteless]
21	[6.261921991748851e-08, scarlet]	[0.007434017807440536, nice]
22	[2.7851871849940903e-08, cartlig]	[0.00728045775722417, product]
23	[2.711510471379777e-08, sincer]	[0.007001213598140437, mayb]
24	[9.321469455178233e-09, meat]	[0.006894815255388098, addict]

WORDCLOUD on BoW

In [42]:
```
from wordcloud import WordCloud, STOPWORDS
stop = set(STOPWORDS)
```

In [65]:
```
%%time

dataframe['25_Positive_features'] = dataframe.top25_pos.apply(lambda x:
 x[1])
plt.rcParams['figure.figsize']=(8.0,6.0)
plt.Figure(figsize=(12, 10), dpi=80, facecolor='w', edgecolor='k')
plt.rcParams['font.size']=12                #10
plt.rcParams['savefig.dpi']=100             #72
plt.rcParams['figure.subplot.bottom']=.1

def show_wordcloud(data, title=None):
    wordcloud = WordCloud(background_color='white',stopwords=stop,max_w
ords=25,max_font_size=40,scale=3,
                          random_state=42).generate(str(data))

    fig = plt.figure(1, figsize=(8, 8))
    plt.axis('off')
    if title:
        fig.suptitle(title, fontsize=20)
        fig.subplots_adjust(top=2.3)

    plt.imshow(wordcloud)
    plt.show()

show_wordcloud(dataframe['25_Positive_features'])
#final.loc[final['Score'] >3]['CleanedText_Bow']
```




```
CPU times: user 308 ms, sys: 0 ns, total: 308 ms
Wall time: 312 ms
```

TFIDF - Random Forest

In [44]:
```
#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 [45]:
```
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 [46]:
```
xtr.shape, xcv.shape, xtest.shape
```

Out[46]:
```
((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 [47]:
```
class RandomForest:

    '''building the Random Forest classifier based off hypeparameters n
o_of_base_models and max_depth of base_models'''

    #instantiating the instance attributes:
    def __init__(self, xtr, ytr, xcv, ycv, maximum_depth=[1, 3, 5, 7, 9
, 11, 13, 15], estimators=[5,7,13,9,11,13,15]):
        self.xtr = xtr
        self.ytr = ytr
        self.xcv = xcv
        self.ycv = ycv
        self.estimators = estimators
        self.maximum_depth = maximum_depth

    #creating a method of calling RF classifier:
    def classfier(self, auc_dict_cv={}, auc_dict_tr={}):
        for estimator in self.estimators:
            for depths in self.maximum_depth:
                clf = RandomForestClassifier(max_depth=depths, n_estima
tors=estimator, oob_score=True)
                print(depths, estimator)
                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[auc_val] = [estimator, depths]

                #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[auc_val] = [estimator, depths]

        return auc_dict_tr, auc_dict_cv
```

TFidf -RF instance creation on training and CV data

In [48]:
```
import time
start = time.time()
Tfidf_instance = RandomForest(xtr, ytr, xcv, ycv)

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

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

```
1 5
3 5
5 5
7 5
9 5
11 5
13 5
15 5
1 7
3 7
5 7
7 7
9 7
```

```
11 7
13 7
15 7
1 13
3 13
5 13
7 13
9 13
11 13
13 13
15 13
1 9
3 9
5 9
7 9
9 9
11 9
13 9
15 9
1 11
3 11
5 11
7 11
9 11
11 11
13 11
15 11
1 13
3 13
5 13
7 13
9 13
11 13
13 13
15 13
1 15
3 15
5 15
7 15
```

```
9 15
11 15
13 15
15 15
time is(in seconds):  59.57049107551575
```

sorting the data based off AUC score

In [49]:
```
train_list = [[np.round(x, 5), y] for x, y in dictionary_train.items()]
cv_list = [[np.round(x, 5), y] for x, y in dictionary_cv.items()]

print(train_list[0])

print()
print(cv_list[0])

tr_list = sorted(train_list, key=lambda x: x[0], reverse=True)
cv_list = sorted(cv_list, key=lambda x: x[0], 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])
```

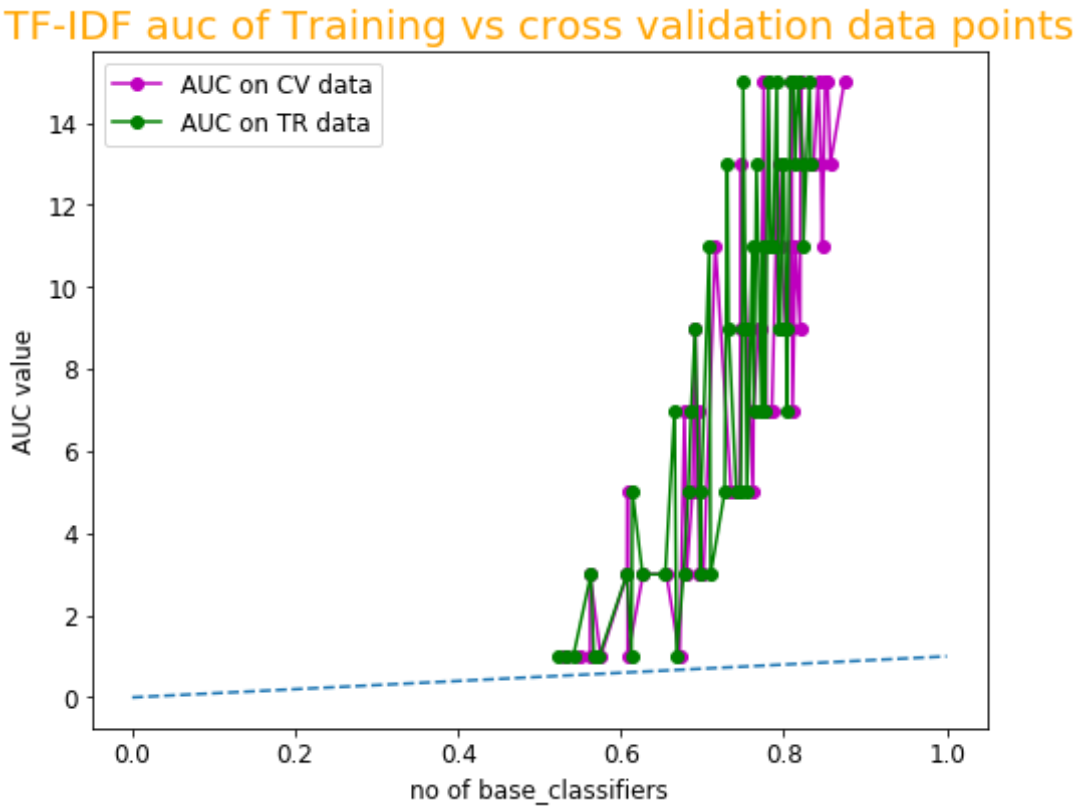
```
[0.53364, [5, 1]]

[0.53378, [5, 1]]
sorted train list score based off AUC score is:
 [[0.87425, [13, 15]], [0.85715, [13, 13]], [0.85364, [11, 15]]]
****************************************

sorted CV list score based off AUC score is:
 [[0.83311, [13, 13]], [0.83158, [13, 15]], [0.82396, [15, 11]]]
```

Plotting AUC Curve on training and cv data based off depth

In [50]:
```
plt.plot([x[0] for x in tr_list], [x[1][1] for x in tr_list], linestyle
```

```
='-', color='m', marker='o',label='AUC on CV data')
plt.plot([x[0] for x in cv_list], [x[1][1] for x in cv_list], linestyle
='-', color='g', marker='o', label='AUC on TR data')
plt.plot([0, 1], [0, 1], linestyle='--')
plt.xlabel("no of base_classifiers")
plt.ylabel('AUC value')
plt.title('TF-IDF auc of Training vs cross validation data points', siz
e=20, color='orange')
plt.legend()
```

Out[50]: <matplotlib.legend.Legend at 0x7f9f6df5efd0>



2. OptimalTfidf - RF[depth = 13 and basemodels

= 15]

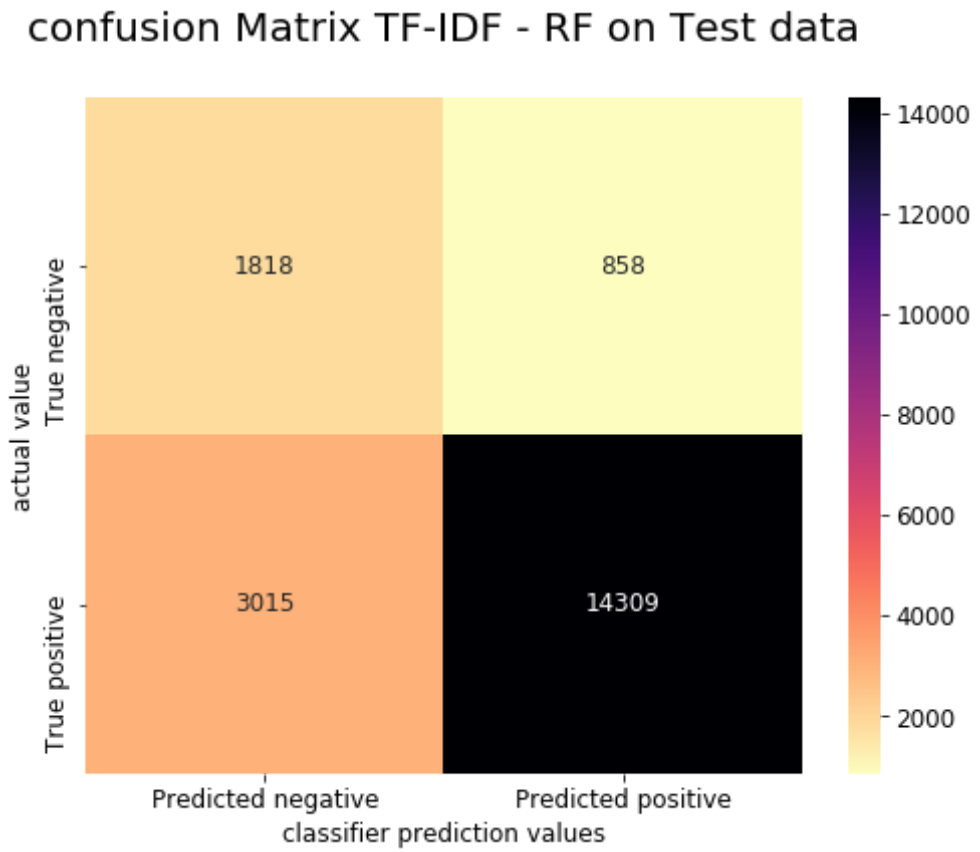
In [51]:
```
rf = RandomForestClassifier(max_depth=13, n_estimators=15, class_weight
='balanced')
rf.fit(xtr, ytr)

#prediction on training data:
y_pred_tr = rf.predict_proba(xtr)
fpr_tr, tpr_tr, thresholds_tr= roc_curve(ytr, y_pred_tr[:,1])
auc_tr = auc(fpr_tr, tpr_tr)

#prediction on test data:
ypred = rf.predict_proba(xtest)
fpr_test, tpr_test, thresholds_test= roc_curve(ytest, ypred[:,1])
auc_test = auc(fpr_test, tpr_test)

print(auc_test)
```

```
0.8345600955706056
```

plotting confusion matrix on test data

In [52]:
```
%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 - RF 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: 7.15 µs
```

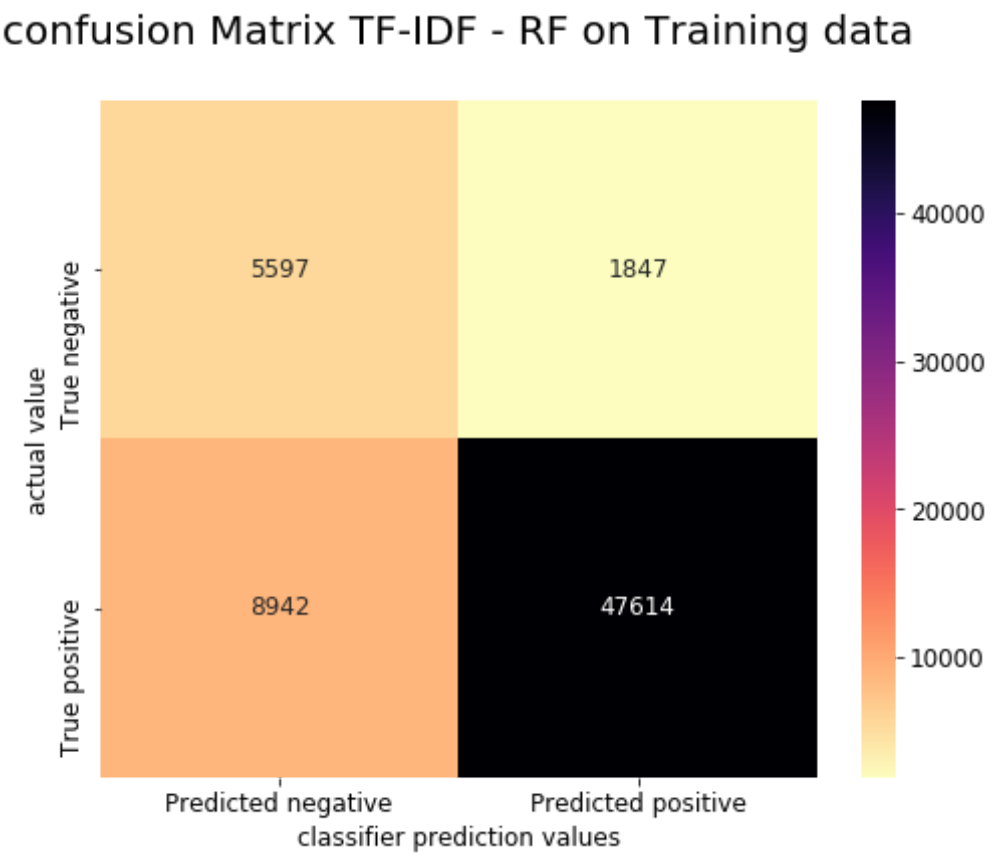


plotting confusion matrix on training data

In [53]:
```
%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 TF-IDF - RF on Training data\n", size=20)
plt.xlabel("classifier prediction values")
plt.ylabel("actual value")
plt.show()
```

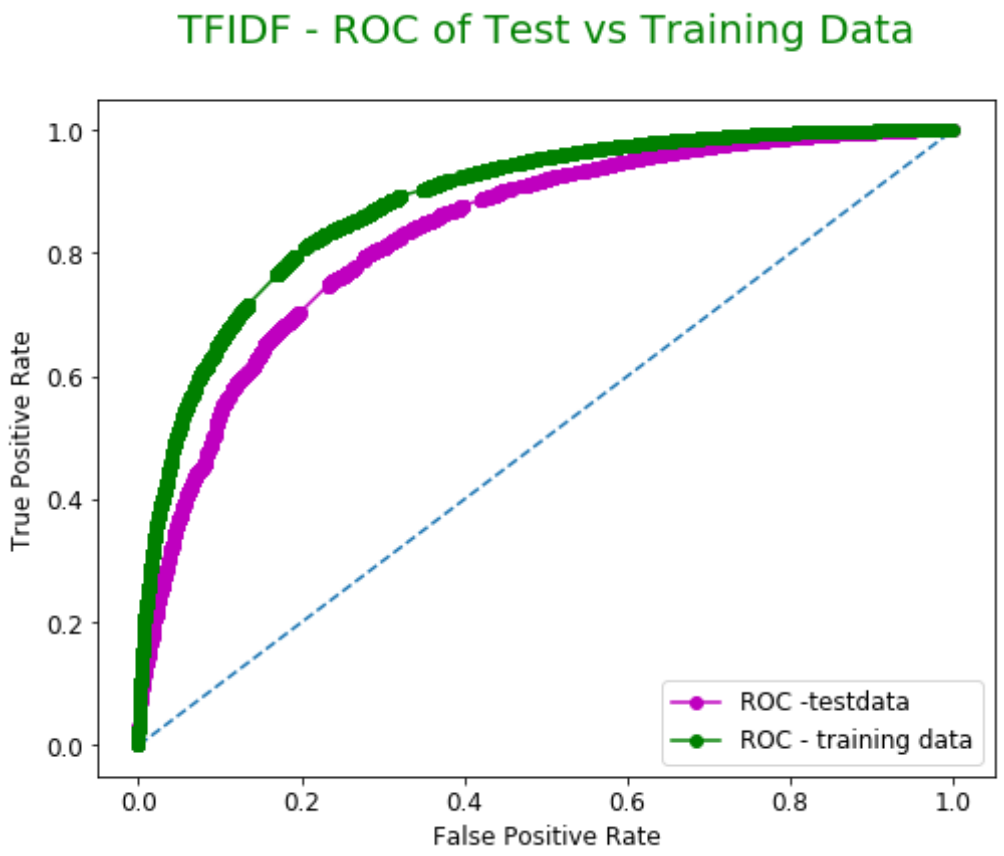
```
CPU times: user 0 ns, sys: 0 ns, total: 0 ns
Wall time: 6.68 µs
```



plotting ROC curve on test data and training data

In [54]:
```
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('TFIDF - ROC of Test vs Training Data\n', size=20, color='gre
en')
plt.legend()
```

Out[54]: <matplotlib.legend.Legend at 0x7f9f706a2128>



important features for positive and negative reviews

In [55]:
```
features = bow_object.get_feature_names()
featuresAndcoeff = sorted(zip(rf.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
 0.033056366993710465               bevnet
                 0.0        aaaaaaaaagghh
  0.03209340815880886                 stor
                 0.0            aaaaaagh
  0.02916151102557499                gassi
                 0.0              aaaaah
 0.020132750898760286            irregular
                 0.0 aaaaahhhhhhhhhhhhhhh
  0.018256027565178405                flex
                 0.0              aaaah
 0.017088233135053456             trucchi
                 0.0               aaah
 0.015848813580509196               brown
                 0.0             aaamaz
  0.01422569420826981            fruitcak
                 0.0             aachen
 0.014163842033324362              remanc
                 0.0                aad
 0.012987368190423948           cassandra
                 0.0               aadp
  0.01136559120706073               yare
```

```
                0.0                aafco
0.009603437154564213                sional
                0.0                aagh
0.009358924828700151                idioz
                0.0                aah
0.009313841814198916                marco
                0.0                aahh
 0.00914809152321237              sassafra
                0.0                aand
0.008220649801127794                takeov
                0.0             aardvark
 0.00744221027755049                nobodi
                0.0                  ab
0.007325687029078247              herbicid
                0.0                aback
0.007026299222115536                extrani
                0.0               abacor
0.006740164009083288                  shke
                0.0               abaloo
0.006397932332640246                gastic
                0.0              abandon
0.006307924009288898               varnishk
                0.0              abaolut
0.005882428266264026               affluent
                0.0             abattoir
0.005817928668787988                formual
                0.0                abba
0.005732004718676944                bottem
```

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 [56]:
```
features = bow_object.get_feature_names()
featuresAndcoeff = sorted(zip(rf.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[56]:

	top25_neg	top25_pos
0	[6.526216959862608e-07, biryani]	[0.033056366993710465, bevnet]
1	[6.414818596889261e-07, intro]	[0.03209340815880886, stor]
2	[6.280058246106124e-07, alpo]	[0.02916151102557499, gassi]
3	[6.240134074921158e-07, refreger]	[0.020132750898760286, irregular]
4	[5.636457607559917e-07, btn]	[0.018256027565178405, flex]
5	[5.480274700057098e-07, foutchi]	[0.017088233135053456, trucchi]
6	[5.091668568471323e-07, yin]	[0.015848813580509196, brown]
7	[4.981945015147794e-07, suprem]	[0.01422569420826981, fruitcak]
8	[4.7667356893687913e-07, cosmet]	[0.014163842033324362, remanc]
9	[4.536273552580307e-07, xanthan]	[0.012987368190423948, cassandra]
10	[4.521985886352647e-07, playdat]	[0.01136559120706073, yare]
11	[4.417072695378616e-07, smapler]	[0.009603437154564213, sional]
12	[3.774945574227972e-07, tha]	[0.009358924828700151, idioz]
13	[3.7577654044856037e-07, hype]	[0.009313841814198916, marco]
14	[2.8753481469002456e-07, kneecap]	[0.00914809152321237, sassafra]
15	[2.792254660620498e-07, drysdal]	[0.008220649801127794, takeov]
16	[2.3660047509947226e-07, yemini]	[0.00744221027755049, nobodi]

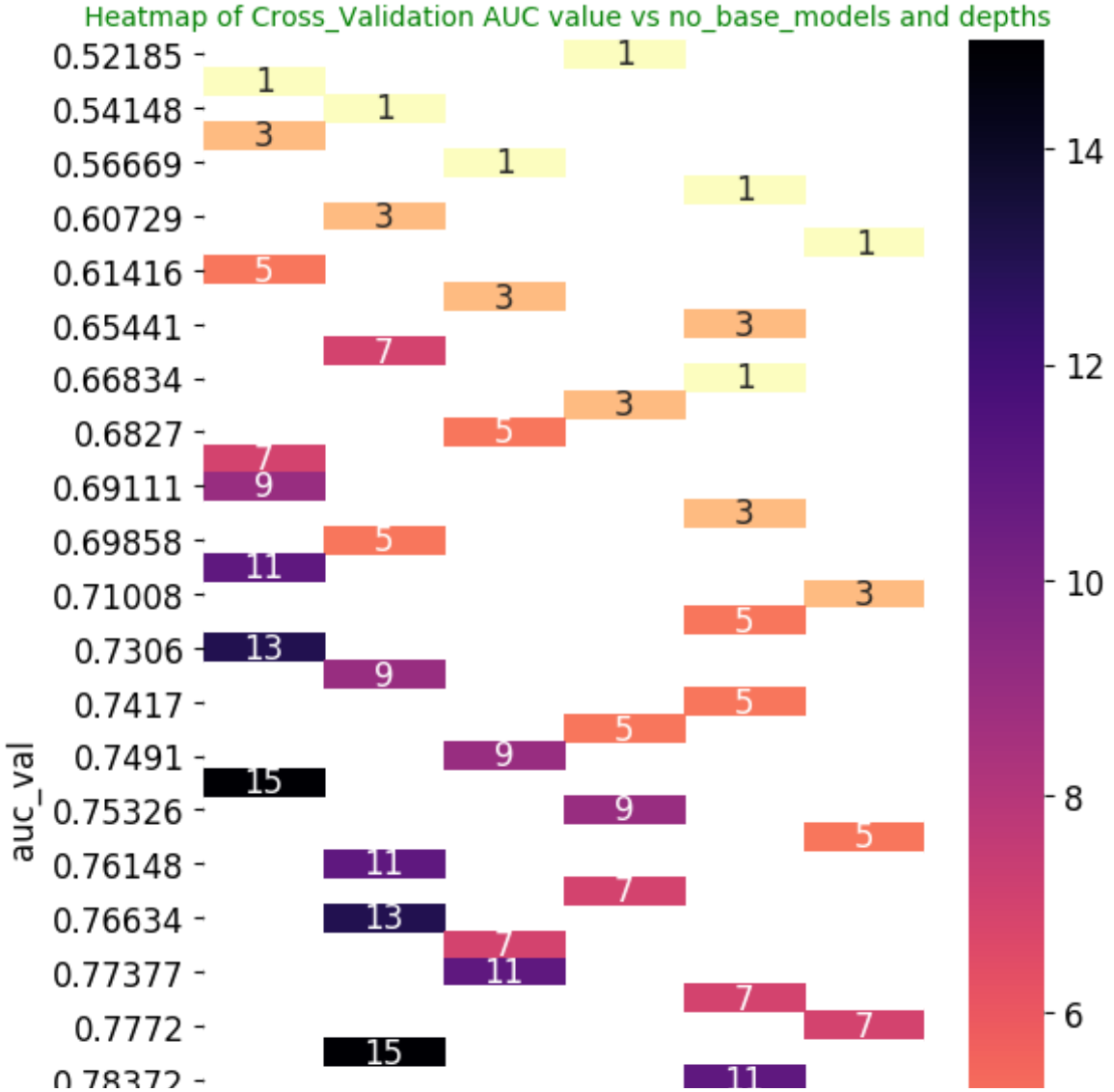
	top25_neg	top25_pos
17	[2.152593094602331e-07, siet]	[0.007325687029078247, herbicid]
18	[2.080758601659428e-07, manufactor]	[0.007026299222115536, extrani]
19	[2.0382189065141183e-07, natuarl]	[0.006740164009083288, shke]
20	[1.8159444482171362e-07, manuka]	[0.006397932332640246, gastic]
21	[1.5468239091729117e-07, street]	[0.006307924009288898, varnishk]
22	[1.3920195696581262e-07, dweller]	[0.005882428266264026, affluent]
23	[1.3788310827296744e-07, traci]	[0.005817986687879885, formual]
24	[1.264661233164313e-07, urband]	[0.005732004718676944, bottem]

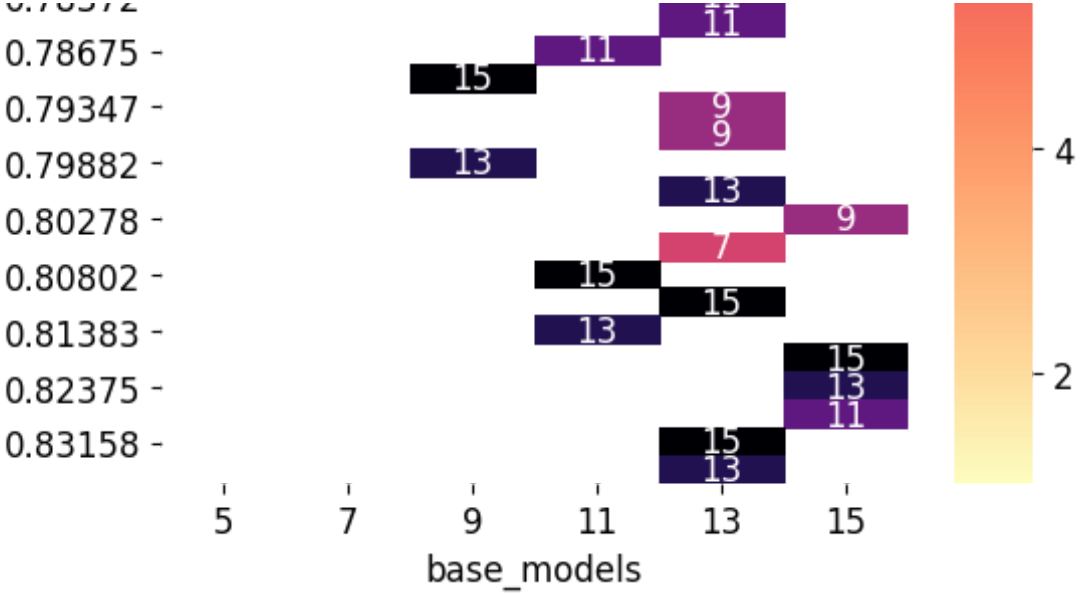
Plotting heatmap of CV data of AUC vs no of base models and depths

In [57]:
```
#creating dataframe for pivot table:
auc_scores = []
no_base_modesl = []
no_depths = []

for i in range(len(cv_list)):
    auc_scores.append(cv_list[i][0])
    no_base_modesl.append(cv_list[i][1][0])
    no_depths.append(cv_list[i][1][1])

df_dict = dict(base_models=no_base_modesl,
               depths=no_depths,
               auc_val=auc_scores)
d_dframe = pd.DataFrame(df_dict)
d_dframe.head(4)



#plotting heatmap:
fig = plt.figure(figsize=(6, 10), dpi=100)
d_pivot = d_dframe.pivot('auc_val','base_models', 'depths')
```

```
sns.heatmap(d_pivot, annot=True, cmap='magma_r')
plt.title(' Heatmap of Cross_Validation AUC value vs no_base_models and
 depths', size=10, color='green')
```

Out[57]: Text(0.5, 1.0, ' Heatmap of Cross_Validation AUC value vs no_base_model
s and depths')





In [58]:

```
%%time

#WOrdcloud - Tfidf

dataframe['25_Positive_features'] = dataframe.top25_pos.apply(lambda x:
 x[1])
plt.rcParams['figure.figsize']=(8.0,6.0)
plt.Figure(figsize=(12, 10), dpi=80, facecolor='w', edgecolor='k')
plt.rcParams['font.size']=12                #10
plt.rcParams['savefig.dpi']=100             #72
plt.rcParams['figure.subplot.bottom']=.1

def show_wordcloud(data, title=None):
    wordcloud = WordCloud(background_color='white',stopwords=stop,max_w
ords=25,max_font_size=40,scale=3,
                          random_state=42).generate(str(data))

    fig = plt.figure(1, figsize=(8, 8))
    plt.axis('off')
    if title:
```

```
        fig.suptitle(title, fontsize=20)
        fig.subplots_adjust(top=2.3)

    plt.imshow(wordcloud)
    plt.show()

show_wordcloud(dataframe['25_Positive_features'])
```


```
CPU times: user 300 ms, sys: 8 ms, total: 308 ms
Wall time: 431 ms
```

W2V - RF

In [59]:
```
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 [60]:
```
%%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 1.06 s, sys: 204 ms, total: 1.26 s
Wall time: 1.27 s
```

instantiating word2vec object for Train, cv, test data

In [61]:
```
%%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 2min 8s, sys: 520 ms, total: 2min 8s
Wall time: 2min 11s
```

Avg-W2V for train, cv, test data

In [62]:

```
%%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 13min 2s, sys: 1.85 s, total: 13min 4s
Wall time: 13min 5s
```

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

In [63]:
```
class RandomForest:

    '''building the Random Forest classifier based off hypeparameters n
o_of_base_models and max_depth of base_models'''

    #instantiating the instance attributes:
    def __init__(self, xtr, ytr, xcv, ycv, maximum_depth=[1, 3, 5, 7, 9
, 11], estimators=[5,7,13,9,11,13]):
        self.xtr = xtr
        self.ytr = ytr
        self.xcv = xcv
        self.ycv = ycv
        self.estimators = estimators
        self.maximum_depth = maximum_depth

    #creating a method of calling RF classifier:
    def classfier(self, auc_dict_cv={}, auc_dict_tr={}):
        for estimator in self.estimators:
            for depths in self.maximum_depth:
                clf = RandomForestClassifier(max_depth=depths, n_estima
tors=estimator, oob_score=True)
                print(depths, estimator)
                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[auc_val] = [estimator, depths]

                #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[auc_val] = [estimator, depths]

        return auc_dict_tr, auc_dict_cv
```

W2V instance creation

In [64]:
```
import time
start = time.time()
w2v_instance = RandomForest(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
3 5
5 5
7 5
9 5
11 5
1 7
3 7
5 7
7 7
9 7
11 7
1 13
3 13
5 13
7 13
9 13
11 13
1 9
3 9
5 9
7 9
```

```
9 9
11 9
1 11
3 11
5 11
7 11
9 11
11 11
1 13
3 13
5 13
7 13
9 13
11 13
time is(in seconds):  463.107351064682
```

sorting the list based off AUC score

In [67]:
```
#creating dictionary :

train_list = [[np.round(x, 5), y] for x, y in dictionary_train.items()]
cv_list = [[np.round(x, 5), y] for x, y in dictionary_cv.items()]

print(train_list[0])

print()
print(cv_list[0])

tr_list = sorted(train_list, key=lambda x: x[0], reverse=True)
cv_list = sorted(cv_list, key=lambda x: x[0], 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])
```

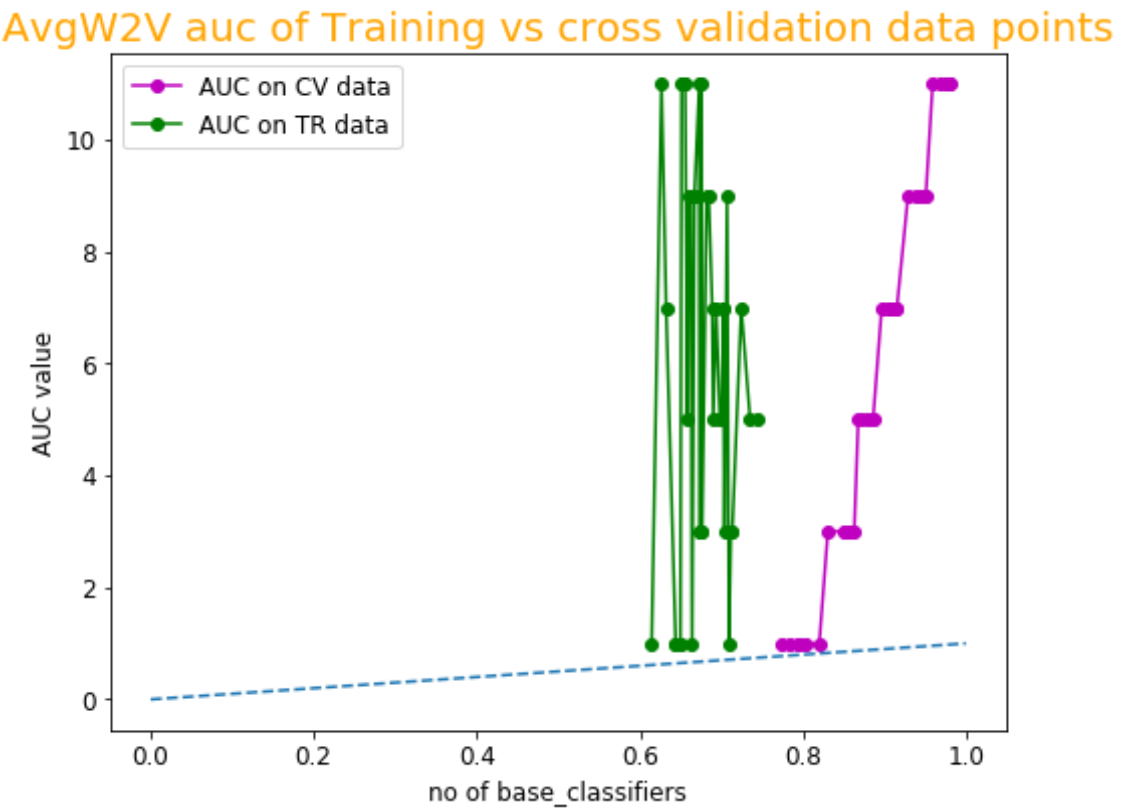
```
[0.77494, [5, 1]]

[0.64429, [5, 1]]
```

```
sorted train list score based off AUC score is:
 [[0.97987, [13, 11]], [0.97896, [13, 11]], [0.97755, [11, 11]]]
****************************************

sorted CV list score based off AUC score is:
 [[0.74465, [9, 5]], [0.73523, [13, 5]], [0.72435, [13, 7]]]
```

Plotting AUC Curve on training and cv data based off depth

In [68]:
```
plt.plot([x[0] for x in tr_list], [x[1][1] for x in tr_list], linestyle
='-', color='m', marker='o',label='AUC on CV data')
plt.plot([x[0] for x in cv_list], [x[1][1] for x in cv_list], linestyle
='-', color='g', marker='o', label='AUC on TR data')
plt.plot([0, 1], [0, 1], linestyle='--')
plt.xlabel("no of base_classifiers")
plt.ylabel('AUC value')
plt.title('AvgW2V auc of Training vs cross validation data points', siz
e=20, color='orange')
plt.legend()
```

Out[68]:
```
<matplotlib.legend.Legend at 0x7f9f61afa438>
```



3.Optimal Avg W2V - RF[depth = 9 and basemodels = 5]

In [69]:
```
rf = RandomForestClassifier(max_depth=9, n_estimators=5, class_weight=
'balanced')
rf.fit(train_vector, ytr)

#prediction on training data:
y_pred_tr = rf.predict_proba(train_vector)
fpr_tr, tpr_tr, thresholds_tr= roc_curve(ytr, y_pred_tr[:,1])
auc_tr = auc(fpr_tr, tpr_tr)
```

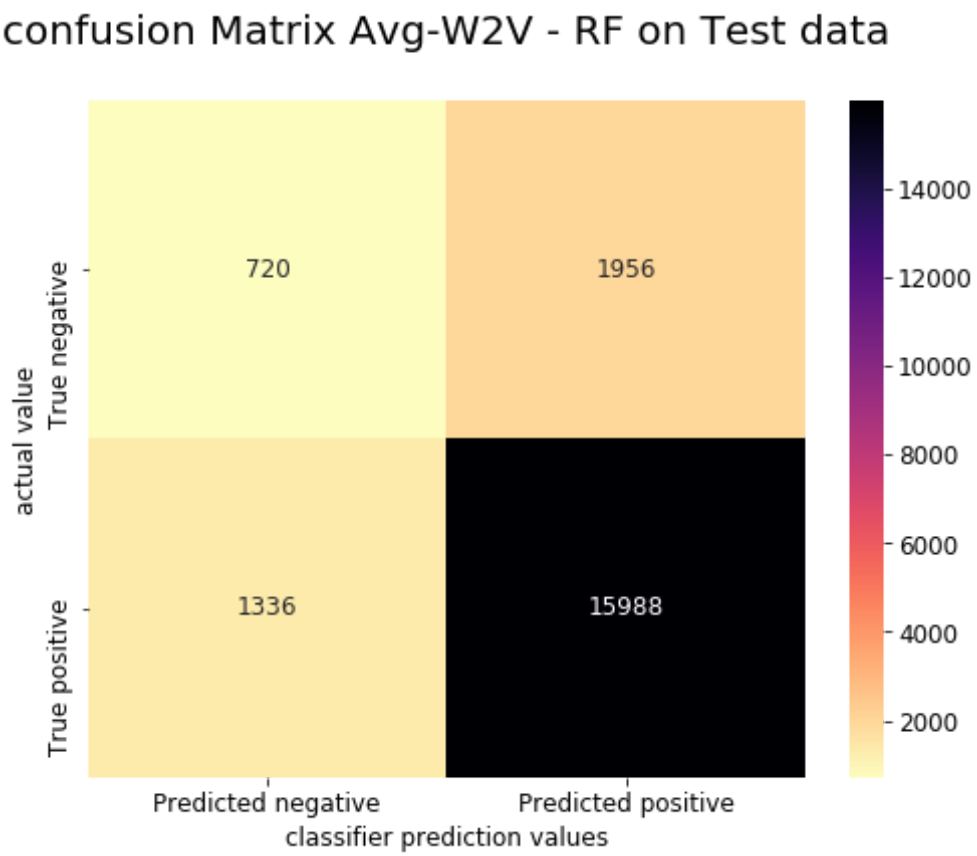
```
#prediction on test data:
ypred = rf.predict_proba(test_vector)
fpr_test, tpr_test, thresholds_test= roc_curve(ytest, ypred[:,1])
auc_test = auc(fpr_test, tpr_test)

print(auc_test)
```

```
0.7130055844143741
```

plotting confusion matrix on test data

In [70]:
```
%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 Avg-W2V - RF 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: 11 µs
```

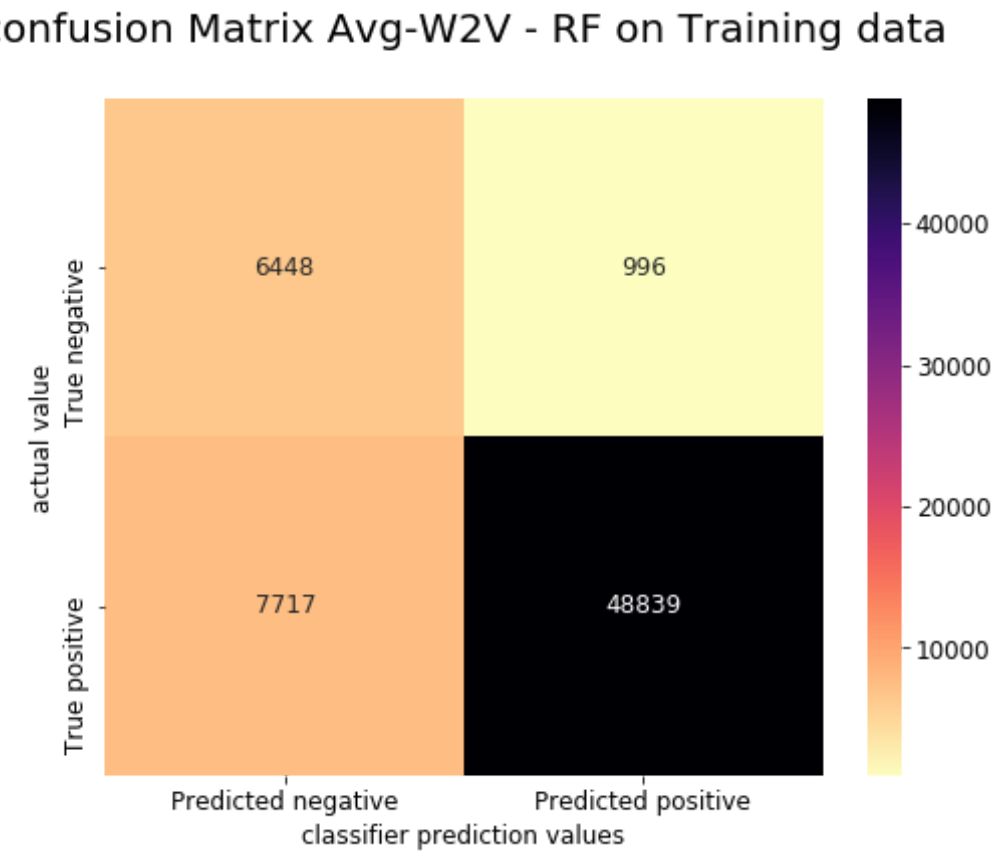


plotting confusion matrix on training data

In [71]:
```
%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 Avg-W2V - RF 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: 7.63 µs
```

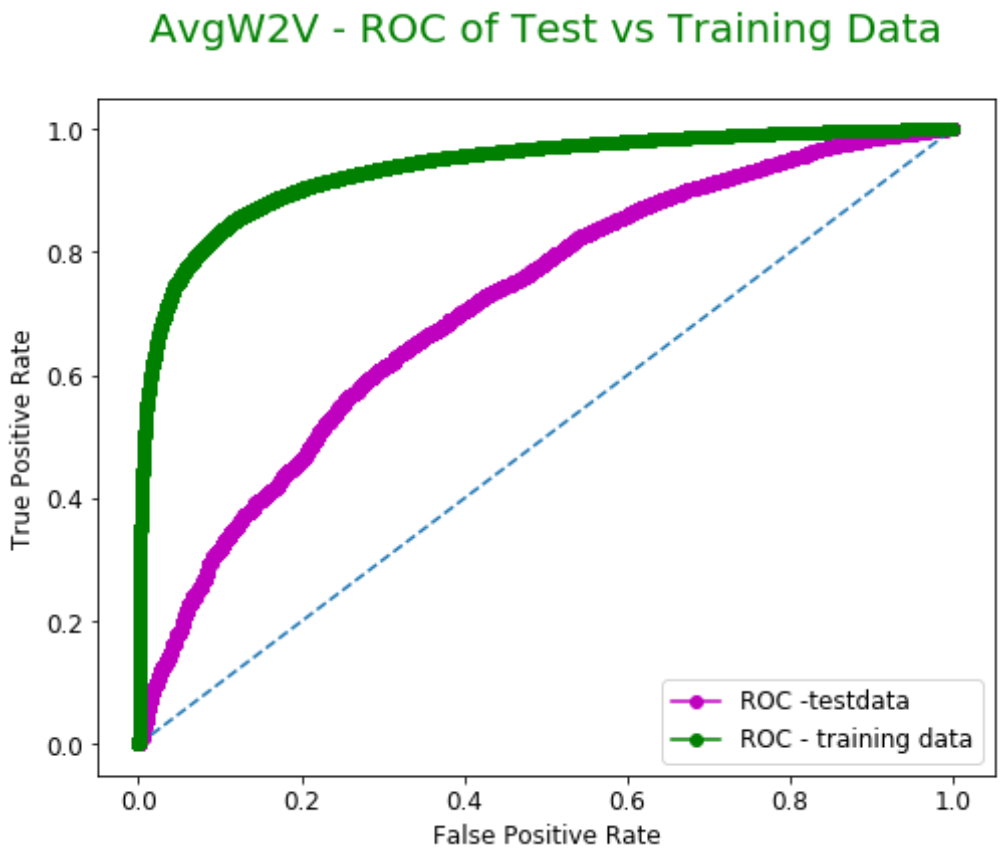


plotting ROC curve on test data and training data

In [72]:
```
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('AvgW2V - ROC of Test vs Training Data\n', size=20, color='gr
een')
plt.legend()
```

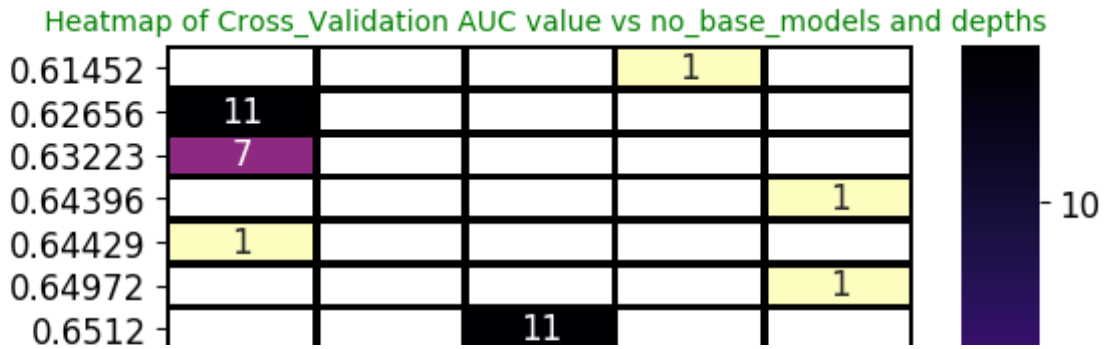
Out[72]: <matplotlib.legend.Legend at 0x7f9f61abdd68>

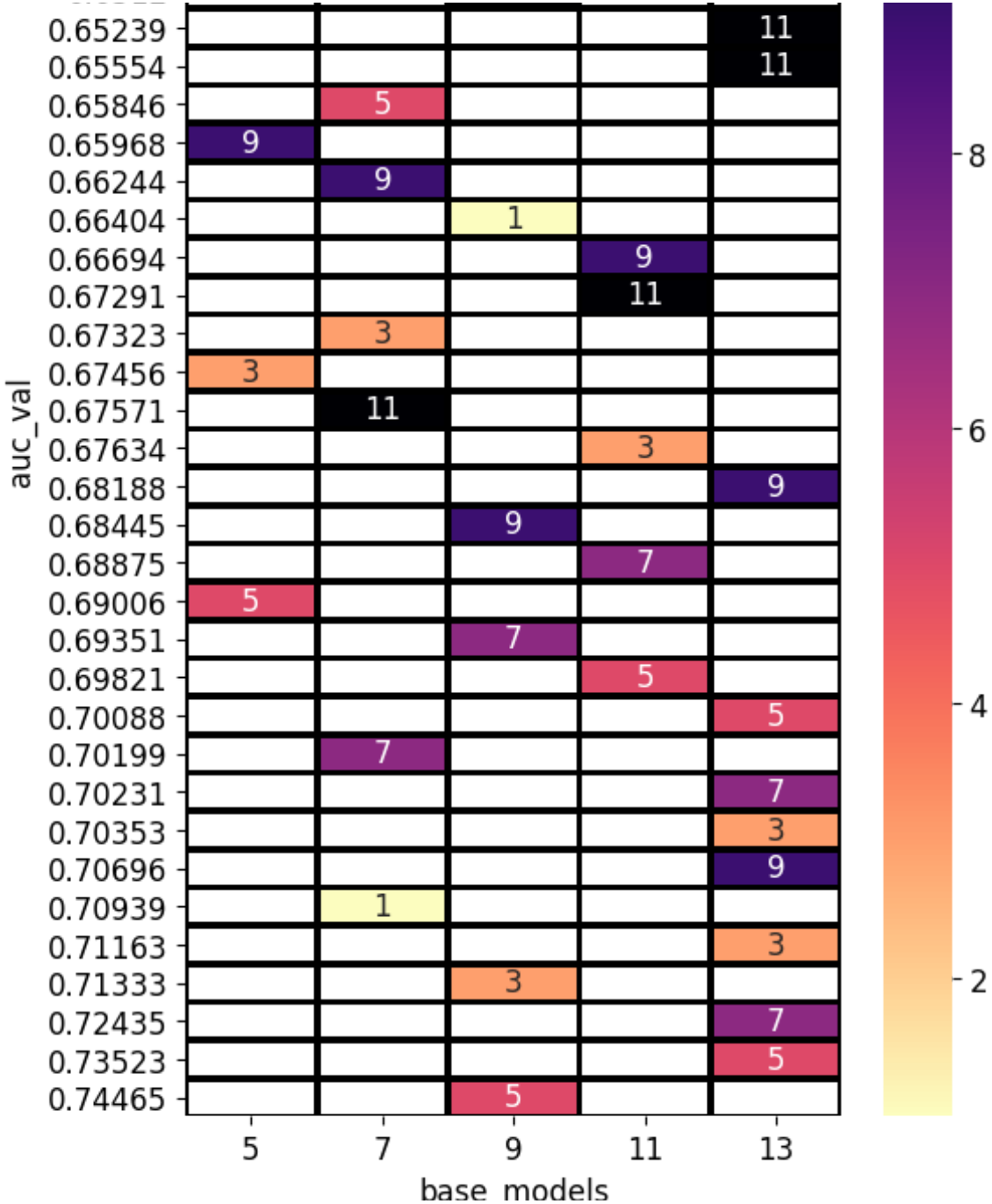


Plotting heatmap of CV data of AUC vs no of base models and depths

In [74]:
```
#creating dataframe for pivot table:
auc_scores = []
no_base_modesl = []
no_depths = []

for i in range(len(cv_list)):
    auc_scores.append(cv_list[i][0])
    no_base_modesl.append(cv_list[i][1][0])
    no_depths.append(cv_list[i][1][1])

df_dict = dict(base_models=no_base_modesl,
               depths=no_depths,
               auc_val=auc_scores)
d_dframe = pd.DataFrame(df_dict)
d_dframe.head(4)



#plotting heatmap:
fig = plt.figure(figsize=(6, 10), dpi=100)
d_pivot = d_dframe.pivot('auc_val','base_models', 'depths')
sns.heatmap(d_pivot, annot=True, cmap='magma_r', linecolor='black', lin
ewidth=2)
plt.title(' Heatmap of Cross_Validation AUC value vs no_base_models and
 depths', size=10, color='green')
```

Out[74]:
```
Text(0.5, 1.0, ' Heatmap of Cross_Validation AUC value vs no_base_model
s and depths')
```





TFidf-W2V

In [5]:
```
%%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, sh
uffle=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 57s, sys: 732 ms, total: 1min 57s
Wall time: 2min
```

TFidf vectorizer

In [6]:
```
%%time
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_)))

print(xtr.shape, xcv.shape, xtest.shape)
print(len(train_list))
```

```
(64000, 43852) (16000, 43852) (20000, 43852)
64000
```

```
CPU times: user 4.14 s, sys: 52 ms, total: 4.19 s
Wall time: 4.26 s
```

Creating TFIDF-W2V for training data,

TFIDF-W2V for cv data AND

TFIDF-W2V for test data

In [7]:
```
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:  2721.835436820984
```

In [10]:
```
import psutil
```

In [11]:
```
dict(psutil.virtual_memory()._asdict())
```

Out[11]:
```
{'active': 2489413632,
 'available': 28764880896,
 'buffers': 45551616,
 'cached': 756719616,
 'free': 28354658304,
 'inactive': 582782976,
 'percent': 9.1,
 'shared': 11411456,
 'slab': 62783488,
 'total': 31628300288,
 'used': 2471370752}
```

In [12]:
```
2471370752 / 31628300288 * 100
```

Out[12]: 7.813795649770201

conversion of list into array

In [11]:
```
%%time

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()
```

```
CPU times: user 1.4 s, sys: 2.46 s, total: 3.86 s
Wall time: 7.69 s
```

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 RF CLASS creation

In [13]:
```
class RandomForest:

    '''building the Random Forest classifier based off hypeparameters n
o_of_base_models and max_depth of base_models'''

    #instantiating the instance attributes:
    def __init__(self, xtr, ytr, xcv, ycv, maximum_depth=[1, 3, 5, 7, 9
, 11], estimators=[5,7,9,11,13]):
        self.xtr = xtr
        self.ytr = ytr
        self.xcv = xcv
        self.ycv = ycv
        self.estimators = estimators
        self.maximum_depth = maximum_depth

    #creating a method of calling RF classifier:
    def classfier(self, auc_dict_cv={}, auc_dict_tr={}):
        for estimator in self.estimators:
            for depths in self.maximum_depth:
                clf = RandomForestClassifier(max_depth=depths, n_estima
tors=estimator, oob_score=True)
                print(depths, estimator)
                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[auc_val] = [estimator, depths]

                #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[auc_val] = [estimator, depths]

        return auc_dict_tr, auc_dict_cv
```

TFIDF - W2V instance creation

In [16]:
```
import time
from sklearn.ensemble import RandomForestClassifier

start = time.time()
tfidfw2v_instance = RandomForest(xtr, ytr, xcv, ycv)

dictionary_train, dictionary_cv = tfidfw2v_instance.classfier()
end = time.time()

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

```
1 5
3 5
5 5
7 5
9 5
11 5
1 7
3 7
5 7
7 7
9 7
11 7
1 9
3 9
5 9
7 9
9 9
11 9
1 11
3 11
5 11
7 11
9 11
11 11
1 13
```

```
3 13
5 13
7 13
9 13
11 13
time is(in seconds):  402.73194789886475
```

sorting the list based off AUC score

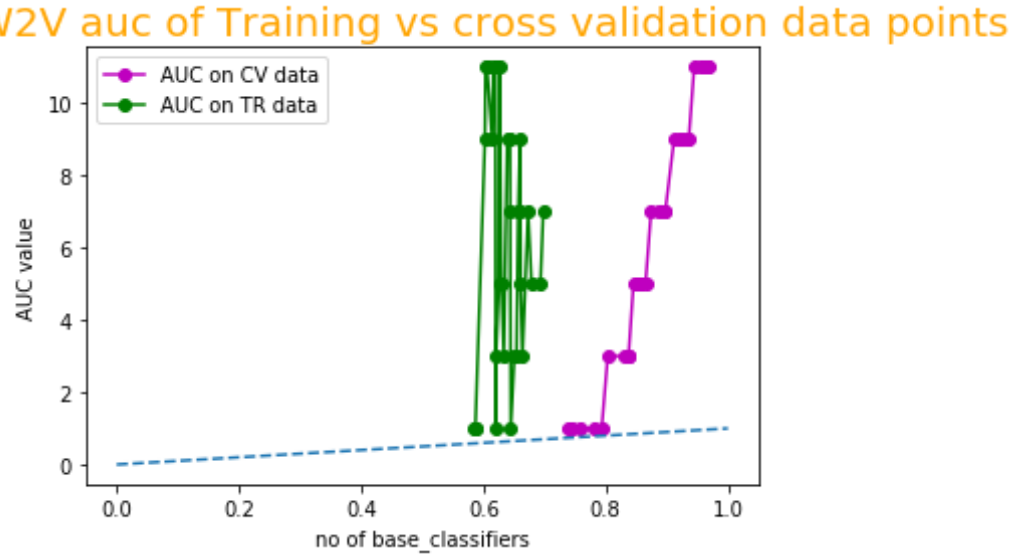
In [17]:
```
#creating dictionary :

train_list = [[np.round(x, 5), y] for x, y in dictionary_train.items()]
cv_list = [[np.round(x, 5), y] for x, y in dictionary_cv.items()]

print(train_list[0])

print()
print(cv_list[0])

tr_list = sorted(train_list, key=lambda x: x[0], reverse=True)
cv_list = sorted(cv_list, key=lambda x: x[0], 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])
```

```
[0.74557, [5, 1]]

[0.58669, [5, 1]]
sorted train list score based off AUC score is:
 [[0.97001, [13, 11]], [0.96571, [11, 11]], [0.96286, [9, 11]]]
****************************************

sorted CV list score based off AUC score is:
 [[0.69779, [13, 7]], [0.69324, [13, 5]], [0.67869, [9, 5]]]
```

Plotting AUC Curve on training and cv data based off depth

In [18]:
```
plt.plot([x[0] for x in tr_list], [x[1][1] for x in tr_list], linestyle
='-', color='m', marker='o',label='AUC on CV data')
plt.plot([x[0] for x in cv_list], [x[1][1] for x in cv_list], linestyle
='-', color='g', marker='o', label='AUC on TR data')
plt.plot([0, 1], [0, 1], linestyle='--')
plt.xlabel("no of base_classifiers")
plt.ylabel('AUC value')
plt.title('TFidf - W2V auc of Training vs cross validation data points'
, size=20, color='orange')
plt.legend()
```

Out[18]: <matplotlib.legend.Legend at 0x7fabaafbf4a8>



4.Optimal TFidf - W2V - RF[depth = 7 and basemodels = 13]

In [20]:
```
rf = RandomForestClassifier(max_depth=7, n_estimators=13, class_weight=
```

```
'balanced')
rf.fit(xtr, ytr)

#prediction on training data:
y_pred_tr = rf.predict_proba(xtr)
fpr_tr, tpr_tr, thresholds_tr= roc_curve(ytr, y_pred_tr[:,1])
auc_tr = auc(fpr_tr, tpr_tr)

#prediction on test data:
ypred = rf.predict_proba(xtest)
fpr_test, tpr_test, thresholds_test= roc_curve(ytest, ypred[:,1])
auc_test = auc(fpr_test, tpr_test)

print(auc_test)
```

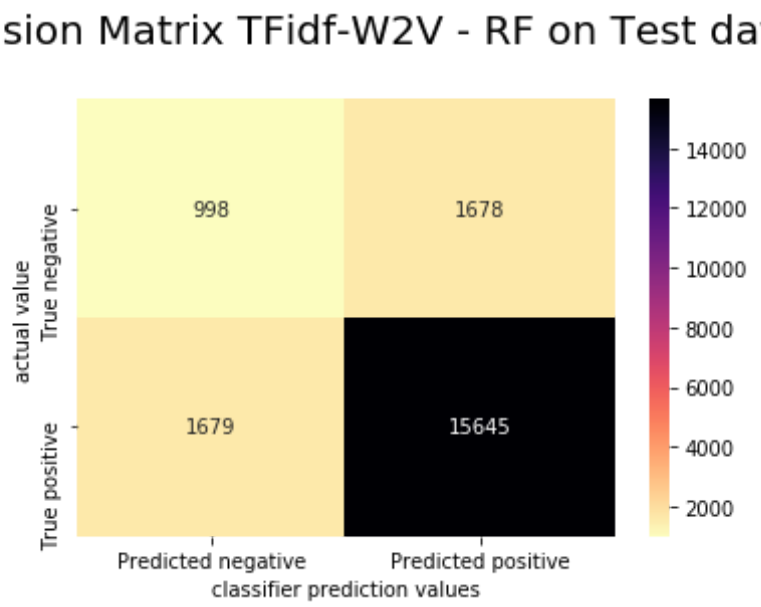
```
0.7680564953222484
```

plotting confusion matrix on test data

In [21]:

```
%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 - RF 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: 6.91 µs
```

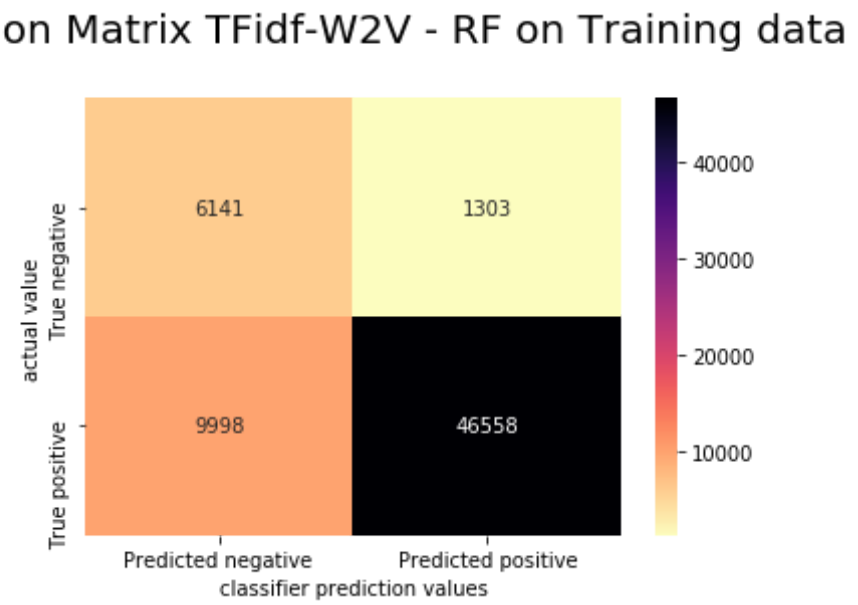


plotting confusion matrix on training data

In [22]:
```
%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 TFidf-W2V - RF 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
```

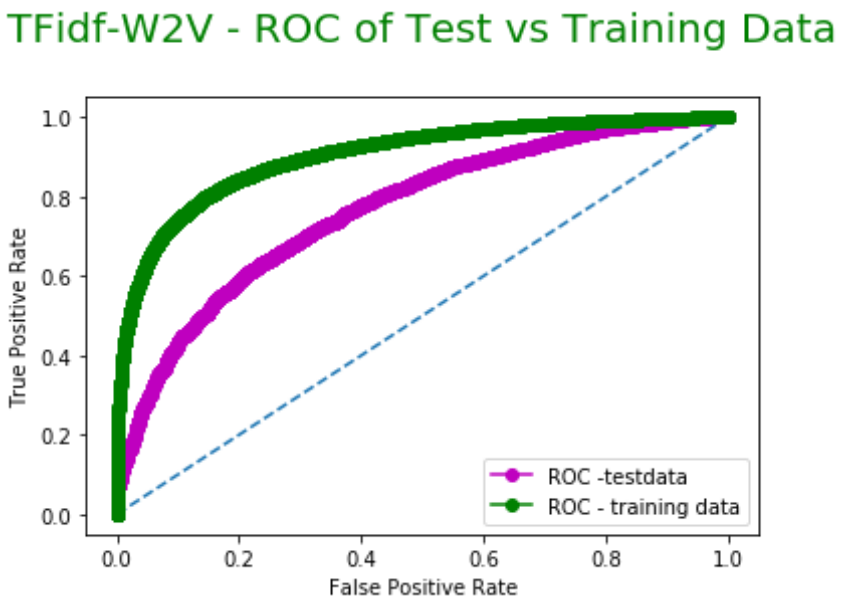
```
Wall time: 6.2 µs
```



plotting ROC curve on test data and training data

In [23]:
```
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('TFidf-W2V - ROC of Test vs Training Data\n', size=20, color=
'green')
plt.legend()
```

Out[23]:
```
<matplotlib.legend.Legend at 0x7fabaabea518>
```

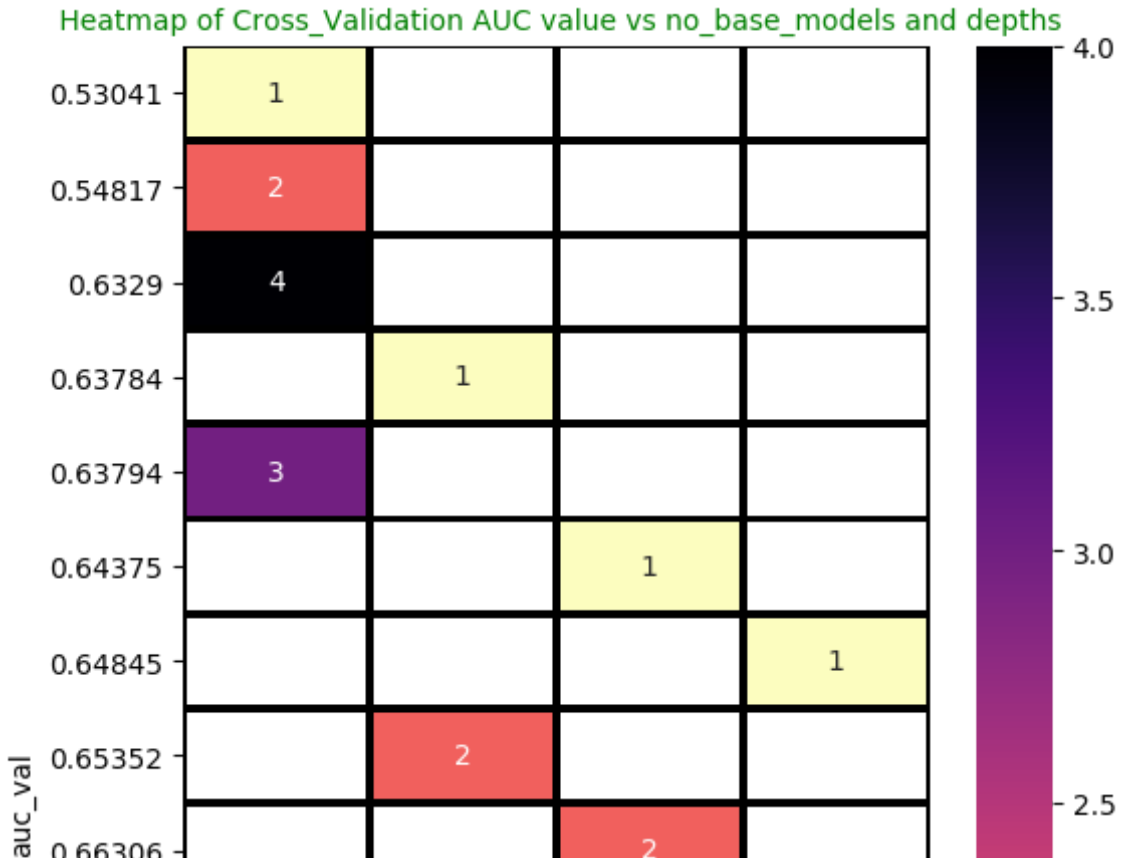


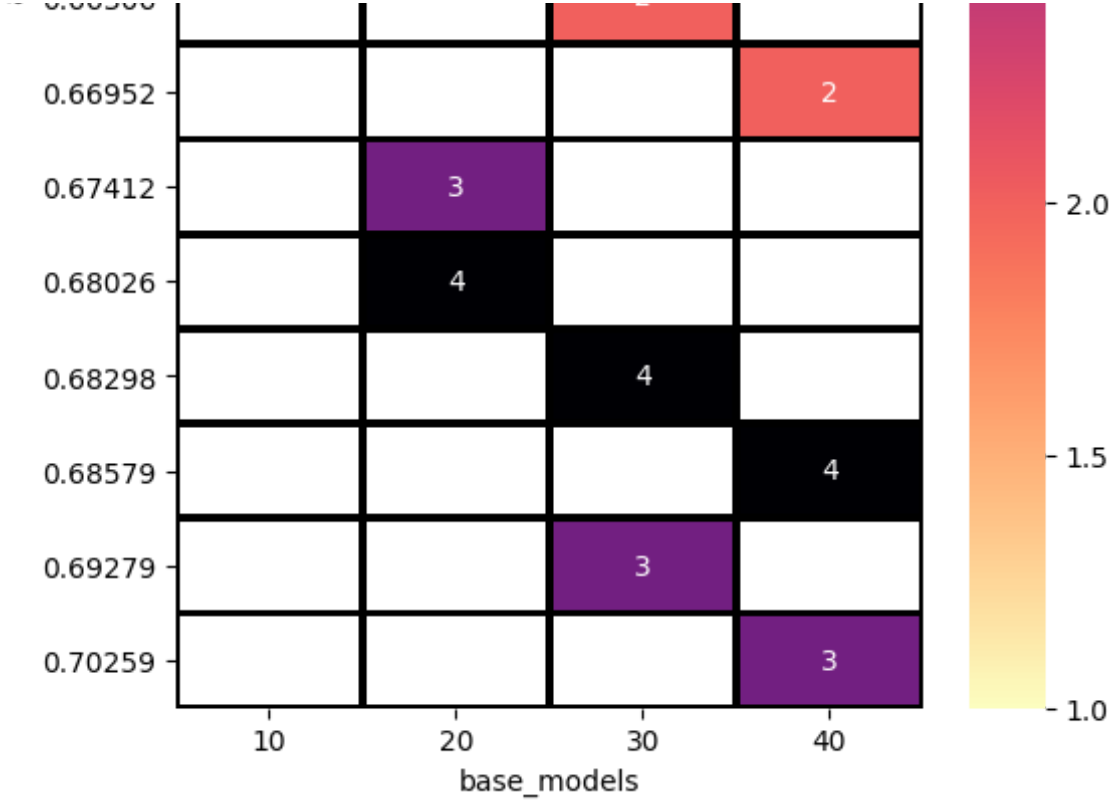
Plotting heatmap of CV data of AUC vs no of base models and depths

In [21]:
```
#creating dataframe for pivot table:
auc_scores = []
no_base_modesl = []
no_depths = []

for i in range(len(cv_list)):
    auc_scores.append(cv_list[i][0])
    no_base_modesl.append(cv_list[i][1][0])
    no_depths.append(cv_list[i][1][1])

df_dict = dict(base_models=no_base_modesl,
               depths=no_depths,
               auc_val=auc_scores)
d_dframe = pd.DataFrame(df_dict)
```

```
d_dframe.head(4)



#plotting heatmap:
fig = plt.figure(figsize=(6, 10), dpi=100)
d_pivot = d_dframe.pivot('auc_val','base_models', 'depths')
sns.heatmap(d_pivot, annot=True, cmap='magma_r', linecolor='black', lin
ewidth=2)
plt.title(' Heatmap of Cross_Validation AUC value vs no_base_models and
 depths', size=10, color='green')
```

Out[21]:
```
Text(0.5, 1.0, ' Heatmap of Cross_Validation AUC value vs no_base_model
s and depths')
```





XGBOOST implementaion on Four vectorizers...

In [13]:
```
from xgboost.sklearn import XGBClassifier
```

Train-Test-CV split...

In [7]:
```
xtrain, xtest, ytrain, ytest = train_test_split(df.CleanedText_Bow, df.
Score, test_size=0.2, shuffle=False)
xtr, xcv, ytr, ycv = train_test_split(xtrain, ytrain, test_size=0.2, sh
uffle=False)
```

BoW instantiation...

In [8]:
```
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, 29808)
(16000, 29808)
(20000, 29808)
```

Building common XGB Class

In [6]:
```
class XGB:

    '''building the XGboostRandom Forest classifier based off hypeparam
eters no_of_base_models and max_depth of base_models'''

    #instantiating the instance attributes:
    def __init__(self, xtr, ytr, xcv, ycv, maximum_depth=[1, 2, 3, 4],
estimators=[10, 20, 30, 40, 50, 60]):
        self.xtr = xtr
        self.ytr = ytr
        self.xcv = xcv
        self.ycv = ycv
        self.estimators = estimators
        self.maximum_depth = maximum_depth

    #creating a method of calling RF classifier:
```

```
    def classfier(self, auc_dict_cv={}, auc_dict_tr={}):
        for estimator in self.estimators:
            for depths in self.maximum_depth:
                clf = XGBClassifier(booster='gbtree',silent=0,max_depth
=depths, n_estimators=estimator, learning_rate=0.1)
                print(depths, estimator)
                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[auc_val] = [estimator, depths]

                #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[auc_val] = [estimator, depths]

        return auc_dict_tr, auc_dict_cv
```

In [22]:

```
%%time

BoW_instance = XGB(xtr, ytr, xcv, ycv)

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

```
1 10
2 10
3 10
4 10
1 20
2 20
3 20
4 20
1 30
```

```
2 30
3 30
4 30
1 40
2 40
3 40
4 40
1 50
2 50
3 50
4 50
1 60
2 60
3 60
4 60
1 70
2 70
3 70
4 70
1 80
2 80
3 80
4 80
CPU times: user 3min 41s, sys: 2.64 s, total: 3min 44s
Wall time: 3min 45s
```

Sorting the data based off AUC score

In [23]:
```
train_list = [[np.round(x, 5), y] for x, y in dictionary_train.items()]
cv_list = [[np.round(x, 5), y] for x, y in dictionary_cv.items()]

print(train_list[0])

print()
print(cv_list[0])

tr_list = sorted(train_list, key=lambda x: x[0], reverse=True)
cv_list = sorted(cv_list, key=lambda x: x[0], reverse=True)
```

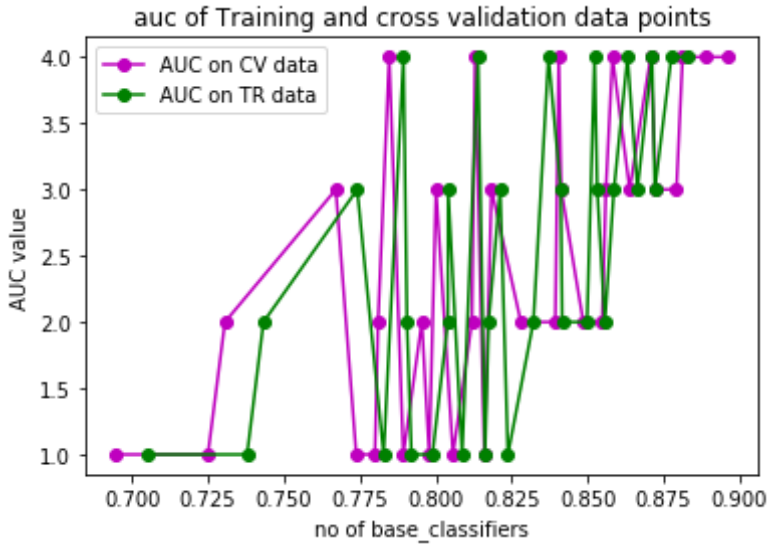
```
print('sorted train list score based off AUC score is:\n', tr_list[0:3
])
print('****************************************')
print()
print('sorted CV list score based off AUC score is:\n', cv_list[0:3])
```

```
[0.69475, [10, 1]]

[0.70502, [10, 1]]
sorted train list score based off AUC score is:
 [[0.89601, [80, 4]], [0.88904, [70, 4]], [0.88127, [60, 4]]]
****************************************

sorted CV list score based off AUC score is:
 [[0.88303, [80, 4]], [0.87755, [70, 4]], [0.87212, [80, 3]]]
```

AUC Curve on training and cv data based off depth

In [24]:
```
plt.plot([x[0] for x in tr_list], [x[1][1] for x in tr_list], linestyle
='-', color='m', marker='o',label='AUC on CV data')
plt.plot([x[0] for x in cv_list], [x[1][1] for x in cv_list], linestyle
='-', color='g', marker='o', label='AUC on TR data')
#plt.plot([0, 1], [0, 1], linestyle='--')
plt.xlabel("no of base_classifiers")
plt.ylabel('AUC value')
plt.title('auc of Training and cross validation data points')
plt.legend()
```

Out[24]:
```
<matplotlib.legend.Legend at 0x7f1b205dd5f8>
```



OptimalBoW - XGBoost[depth = 4 and basemodels = 80]

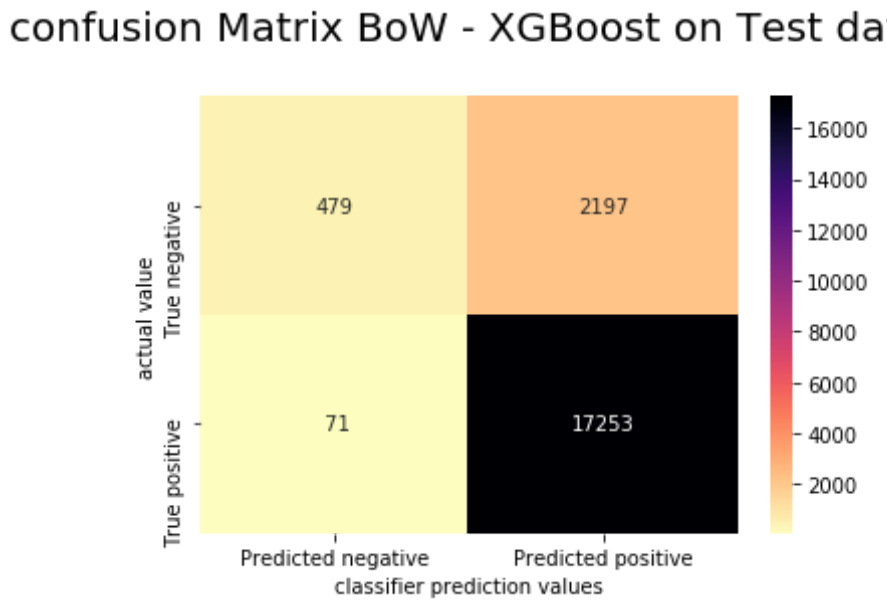
In [25]:
```
clfxgb = XGBClassifier(booster='gbtree',silent=0,max_depth=4, n_estimat
ors=80, learning_rate=0.1, class_weight='balanced')
clfxgb.fit(xtr, ytr)

#prediction on training data:
y_pred_tr = clfxgb.predict_proba(xtr)
fpr_tr, tpr_tr, thresholds_tr= roc_curve(ytr, y_pred_tr[:,1])
auc_tr = auc(fpr_tr, tpr_tr)

#prediction on test data:
ypred = clfxgb.predict_proba(xtest)
fpr_test, tpr_test, thresholds_test= roc_curve(ytest, ypred[:,1])
auc_test = auc(fpr_test, tpr_test)

print(auc_test)
```

```
0.8852802962374704
```

Confusion matrix on test data

In [26]:
```
%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 - XGBoost on Test data\n", size=20)
plt.xlabel("classifier prediction values")
plt.ylabel("actual value")
plt.show()
```

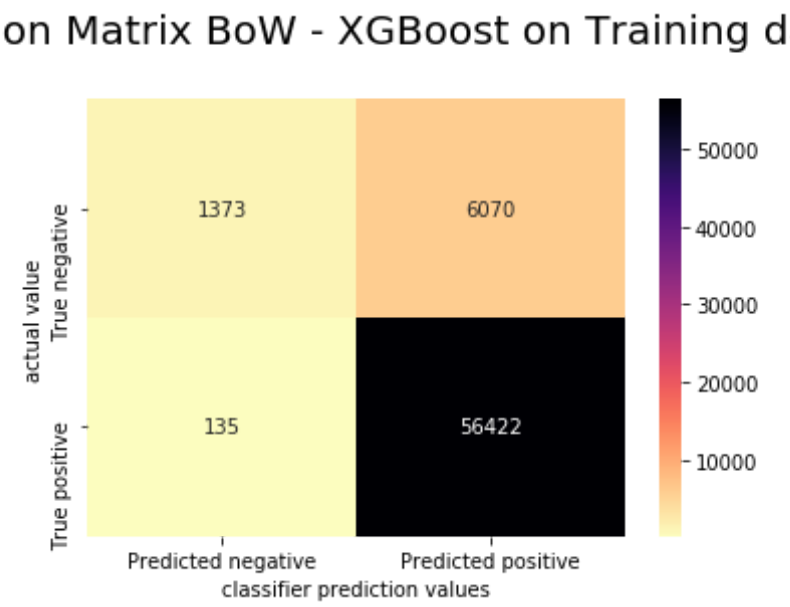
```
CPU times: user 0 ns, sys: 0 ns, total: 0 ns
Wall time: 6.2 µs
```



Confusion matrix on training data

In [27]:
```
%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 BoW - XGBoost on Training data\n", size=20)
plt.xlabel("classifier prediction values")
plt.ylabel("actual value")
plt.show()
```

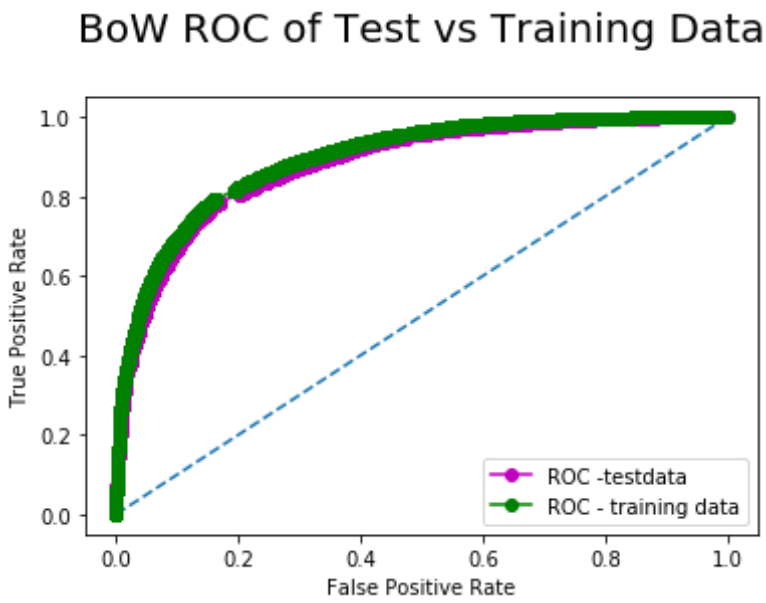
```
CPU times: user 0 ns, sys: 0 ns, total: 0 ns
Wall time: 6.44 µs
```



ROC curve - training vs test data

In [28]:
```
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('BoW ROC of Test vs Training Data\n', size=20)
plt.legend()
```

Out[28]:
```
<matplotlib.legend.Legend at 0x7f1b20000b38>
```



Feature Importance

In [29]:
```
features = bow_object.get_feature_names()
featuresAndcoeff = sorted(zip(clfxgb.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
```

```
0.027376839891076088           disappoint
                 0.0        aaaaaaaaagghh
 0.02200981229543686               return
                 0.0               aaaaah
 0.02034367248415947                  not
                 0.0 aaaaahhhhhhhhhhhhhhh
 0.01972492039203643                great
                 0.0               aaaah
 0.01964561827480793                money
                 0.0                aaah
 0.01670743338763714              horribl
                 0.0              aachen
0.016628563404083252                worst
                 0.0                aad
0.016302969306707382              favorit
                 0.0                aadp
 0.01630045846104622                best
                 0.0              aafco
 0.01586350053548813                love
                 0.0                aagh
0.015563997440040112              terribl
                 0.0                 aah
0.015115649439394474                 bad
                 0.0                aahh
0.015099555253982544              delici
                 0.0                aand
0.015033259056508541                mayb
                 0.0            aardvark
0.014950428158044815                easi
                 0.0                  ab
0.014194663614034653             perfect
                 0.0               aback
0.014138751663267612               bewar
                 0.0             abandon
 0.01347731240093708                  aw
                 0.0             abaolut
0.013415071181952953                didn
                 0.0            abattoir
0.013010956346988678                gross
```

```
                0.0                abba
0.012943311594426632               threw
                0.0              abbey
0.012790671549737453               excel
                0.0               abbi
0.012744717299938202               sorri
                0.0             abbrevi
0.012597453780472279             descript
                0.0                abc
0.012104352004826069               good
```

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 [30]:
```
features = bow_object.get_feature_names()
featuresAndcoeff = sorted(zip(clfxgb.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[30]:

	top25_neg	top25_pos
0	[0.0006088018, sweeten]	[0.02737684, disappoint]
1	[0.0006074863, manufactur]	[0.022009812, return]

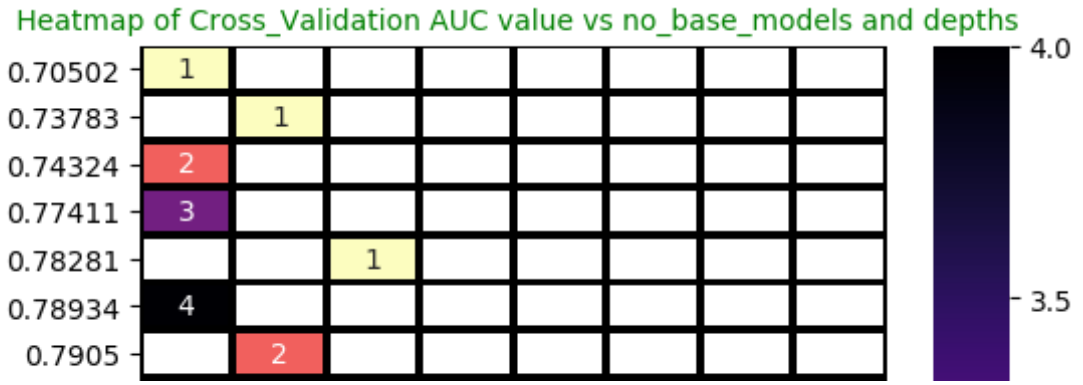
	top25_neg	top25_pos

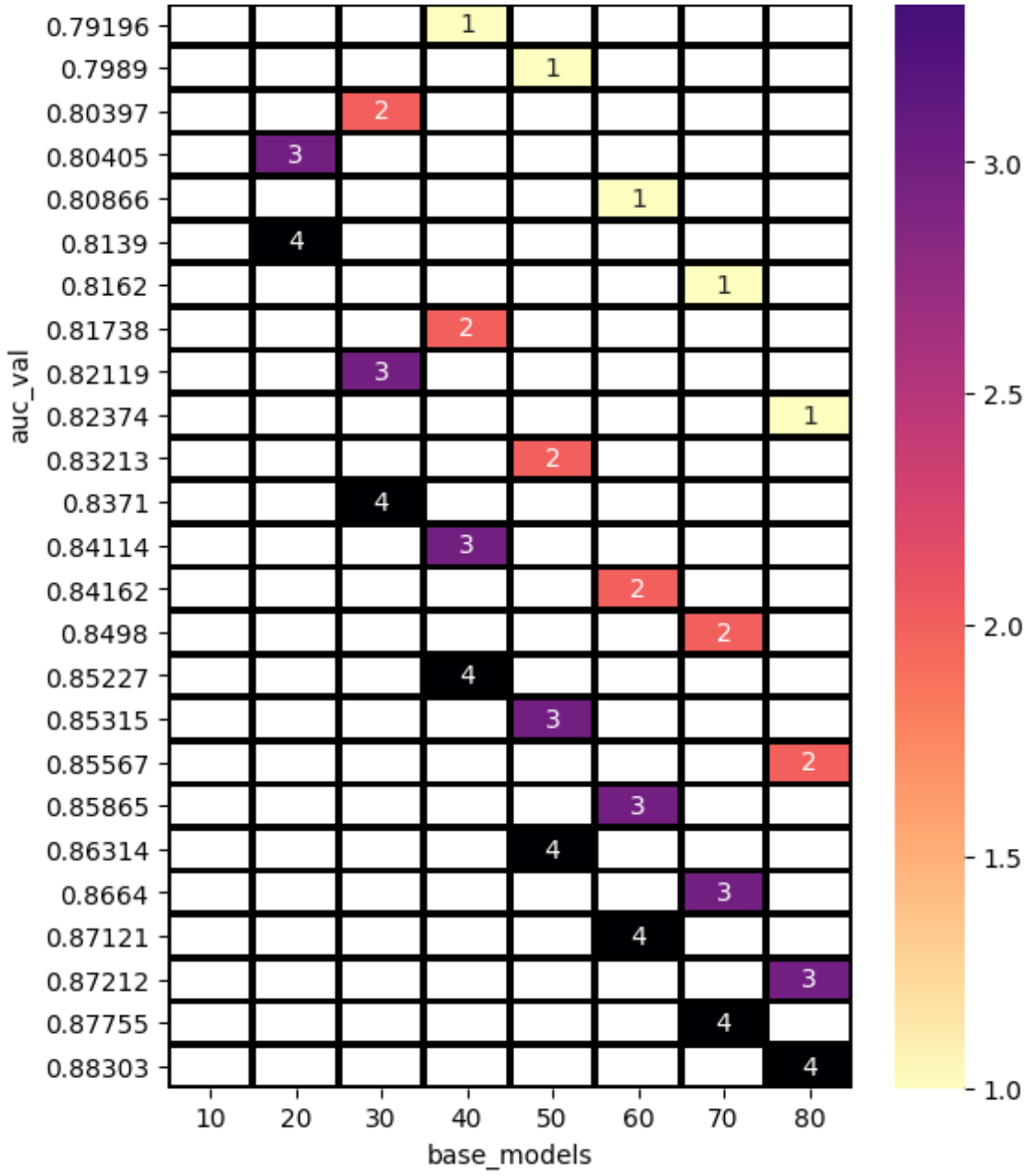
2	[0.00060181285, ever]	[0.020343672, not]
3	[0.0005959539, sever]	[0.01972492, great]
4	[0.00058725785, don]	[0.019645618, money]
5	[0.0005866793, well]	[0.016707433, horribl]
6	[0.00058602006, experi]	[0.016628563, worst]
7	[0.00055399636, call]	[0.01630297, favorit]
8	[0.00055040594, contain]	[0.016300458, best]
9	[0.0005502545, like]	[0.0158635, love]
10	[0.0005353016, order]	[0.015563997, terribl]
11	[0.00053460075, better]	[0.015115649, bad]
12	[0.0005254598, juic]	[0.015099555, delici]
13	[0.0005103482, differ]	[0.015033259, mayb]
14	[0.00049740163, even]	[0.014950428, easi]
15	[0.0004904082, stop]	[0.014194664, perfect]
16	[0.00048564223, pound]	[0.014138752, bewar]
17	[0.00045989756, see]	[0.013477312, aw]
18	[0.00044233247, found]	[0.013415071, didn]
19	[0.00037186596, give]	[0.013010956, gross]
20	[0.00031900912, doesn]	[0.012943312, threw]
21	[0.00031846092, two]	[0.012790672, excel]
22	[0.0003037696, natur]	[0.012744717, sorri]
23	[0.00021034965, one]	[0.012597454, descript]
24	[0.00019601843, tea]	[0.012104352, good]

Heatmap

In [32]:
```
#creating dataframe for pivot table:
auc_scores = []
no_base_models = []
no_depths = []

for i in range(len(cv_list)):
    auc_scores.append(cv_list[i][0])
    no_base_models.append(cv_list[i][1][0])
    no_depths.append(cv_list[i][1][1])

df_dict = dict(base_models=no_base_models,
               depths=no_depths,
               auc_val=auc_scores)
d_dframe = pd.DataFrame(df_dict)
#d_dframe.head(4)



#plotting heatmap:
fig = plt.figure(figsize=(6, 10), dpi=100)
d_pivot = d_dframe.pivot('auc_val','base_models', 'depths')
sns.heatmap(d_pivot, annot=True, cmap='magma_r', linecolor='black', lin
ewidth=2)
plt.title(' Heatmap of Cross_Validation AUC value vs no_base_models and
 depths', size=10, color='green')
```

Out[32]:
```
Text(0.5, 1.0, ' Heatmap of Cross_Validation AUC value vs no_base_model
s and depths')
```





TFIDF

train test split

In [33]:
```
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 [34]:
```
%%time
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)
```

```
CPU times: user 4.04 s, sys: 176 ms, total: 4.22 s
Wall time: 4.25 s
```

TFIDF instance creation

In [35]:
```
import time
start = time.time()
Tfidf_instance = XGB(xtr, ytr, xcv, ycv)

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

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

```
1 10
2 10
```

```
3 10
4 10
1 20
2 20
3 20
4 20
1 30
2 30
3 30
4 30
1 40
2 40
3 40
4 40
1 50
2 50
3 50
4 50
1 60
2 60
3 60
4 60
1 70
2 70
3 70
4 70
1 80
2 80
3 80
4 80
time is(in seconds):  493.9323706626892
```

Sorting the data based off AUC score

In [36]:
```
train_list = [[np.round(x, 5), y] for x, y in dictionary_train.items()]
cv_list = [[np.round(x, 5), y] for x, y in dictionary_cv.items()]

print(train_list[0])
```

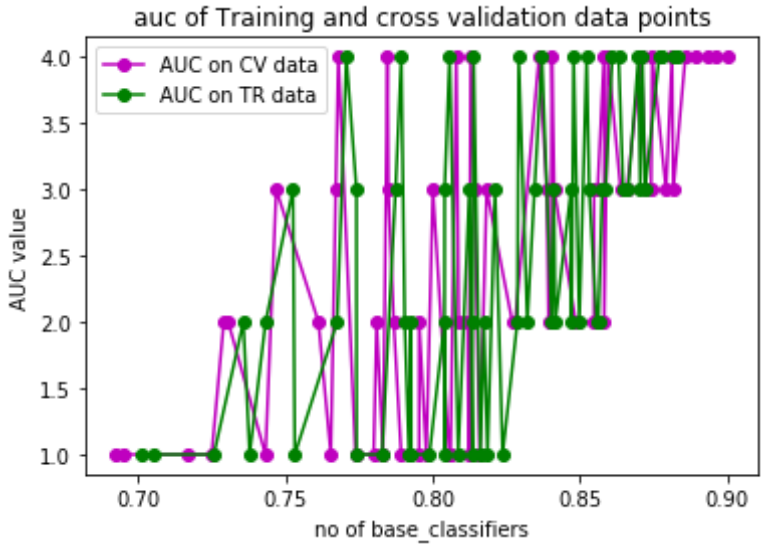
```
print()
print(cv_list[0])

tr_list = sorted(train_list, key=lambda x: x[0], reverse=True)
cv_list = sorted(cv_list, key=lambda x: x[0], 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])
```

```
[0.69475, [10, 1]]

[0.70502, [10, 1]]
sorted train list score based off AUC score is:
 [[0.90007, [80, 4]], [0.89601, [80, 4]], [0.89338, [70, 4]]]
****************************************

sorted CV list score based off AUC score is:
 [[0.88303, [80, 4]], [0.88201, [80, 4]], [0.87755, [70, 4]]]
```

AUC Curve on training and cv data based off depth

In [37]:
```
plt.plot([x[0] for x in tr_list], [x[1][1] for x in tr_list], linestyle
='-', color='m', marker='o',label='AUC on CV data')
plt.plot([x[0] for x in cv_list], [x[1][1] for x in cv_list], linestyle
='-', color='g', marker='o', label='AUC on TR data')
#plt.plot([0, 1], [0, 1], linestyle='--')
plt.xlabel("no of base_classifiers")
plt.ylabel('AUC value')
plt.title('auc of Training and cross validation data points')
plt.legend()
```

Out[37]:
```
<matplotlib.legend.Legend at 0x7f1b1de9ab00>
```



OptimalTFidf - XGBoost[depth = 4 and basemodels = 80]

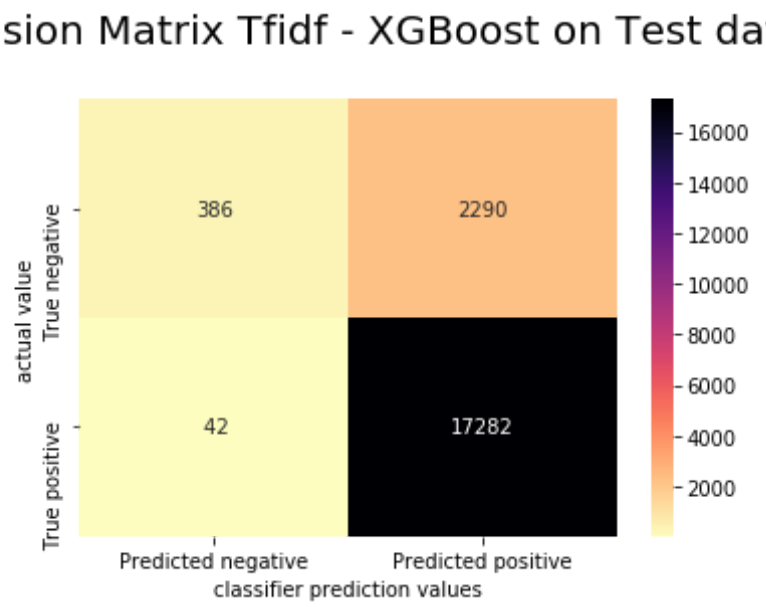
In [38]:
```
clfxgb = XGBClassifier(booster='gbtree',silent=0,max_depth=4, n_estimat
ors=80, learning_rate=0.1, class_weight='balanced')
clfxgb.fit(xtr, ytr)

#prediction on training data:
y_pred_tr = clfxgb.predict_proba(xtr)
fpr_tr, tpr_tr, thresholds_tr= roc_curve(ytr, y_pred_tr[:,1])
auc_tr = auc(fpr_tr, tpr_tr)

#prediction on test data:
ypred = clfxgb.predict_proba(xtest)
fpr_test, tpr_test, thresholds_test= roc_curve(ytest, ypred[:,1])
auc_test = auc(fpr_test, tpr_test)

print(auc_test)
```

```
0.8832488039437586
```

Confusion matrix on test data

In [39]:
```
%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 - XGBoost 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: 7.39 µs
```

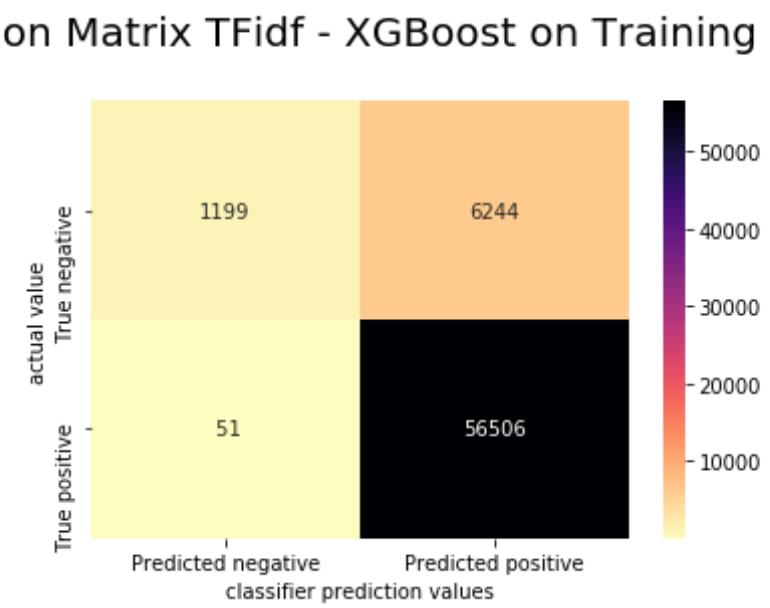


Confusion matrix on training data

In [40]:
```
%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 TFidf - XGBoost on Training data\n", size=2
0)
plt.xlabel("classifier prediction values")
plt.ylabel("actual value")
plt.show()
```

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

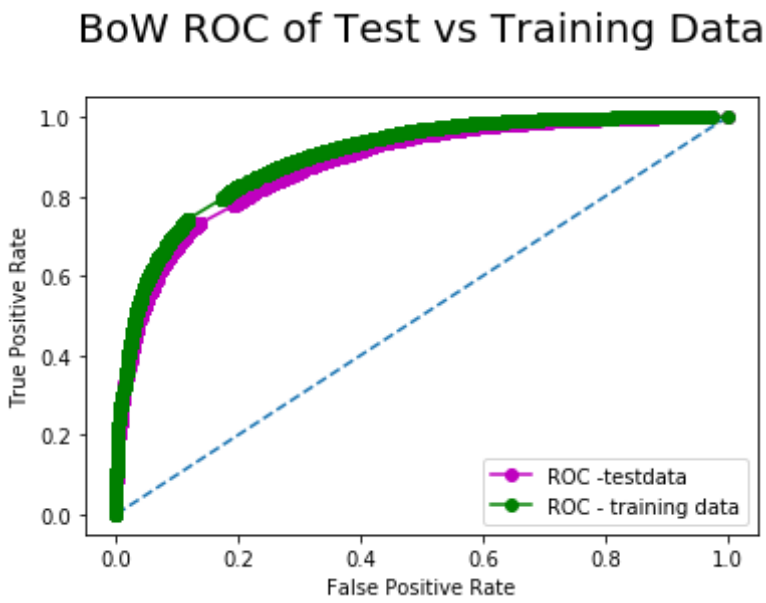
```
Wall time: 7.39 µs
```



ROC curve - training vs test data

In [41]:
```
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('BoW ROC of Test vs Training Data\n', size=20)
plt.legend()
```

Out[41]:
```
<matplotlib.legend.Legend at 0x7f1b20035ac8>
```



Feature Importance

In [42]:
```
features = tfidf_object.get_feature_names()
featuresAndcoeff = sorted(zip(clfxgb.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
```

```
 0.01900656148791313                best
                 0.0        aaaaaaaaagghh
 0.01840704120695591              horrible
                 0.0               aaaaah
 0.017405742779374123                 not
                 0.0 aaaaahhhhhhhhhhhhhhh
  0.01619257591664791                money
                 0.0                aaaah
 0.016128158196806908                great
                 0.0                aaah
 0.015471220016479492             received
                 0.0                aaahs
 0.015331736765801907         disappointed
                 0.0                aachen
  0.01469665952026844                 bad
                 0.0                 aad
  0.0146265632065773                threw
                 0.0                aadp
 0.014441183768212795                gross
                 0.0               aafco
 0.014279133640229702            delicious
                 0.0                aagh
 0.014184785075485706                 didn
                 0.0              aahhed
 0.014178333804011345           disgusting
                 0.0               aahing
  0.01414472982287407           wonderful
                 0.0                  aand
 0.014046834781765938                sorry
                 0.0             aardvark
 0.013720561750233173                return
                 0.0               aback
 0.013659702613949776           description
                 0.0               abandon
 0.013600846752524376                worst
                 0.0            abandoned
 0.013287031091749668                highly
                 0.0           abandoning
  0.01327624823898077             excellent
                 0.0           abaolutely
```

```
0.013225429691374302              beware
                 0.0            abattoir
0.013210281729698181              refund
                 0.0               abba
0.013103865087032318               worse
                 0.0              abbey
0.013080248609185219             favorite
                 0.0        abbreviated
 0.01298781018704176            tasteless
```

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 [43]:
```
features = bow_object.get_feature_names()
featuresAndcoeff = sorted(zip(clfxgb.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[43]:

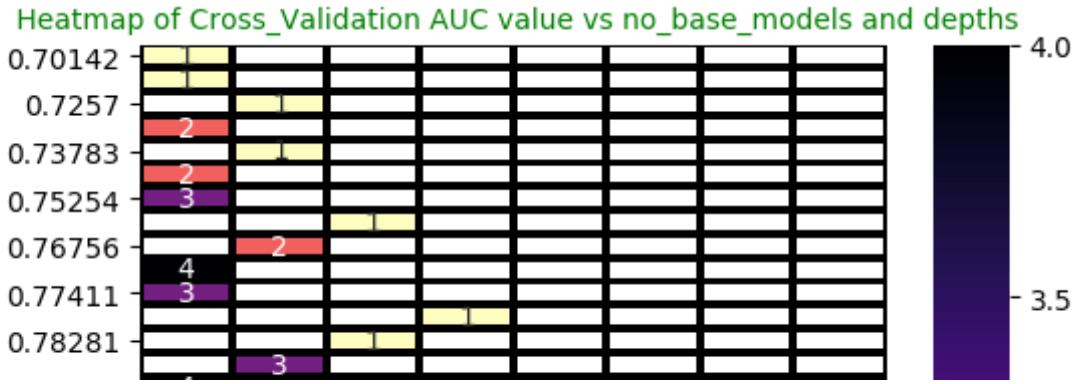
	top25_neg	top25_pos
0	[0.0007298733, bartlet]	[0.019006561, bullet]
1	[0.0007202348, retest]	[0.018407041, osu]
2	[0.00068993523, vice]	[0.017405743, terriffi]

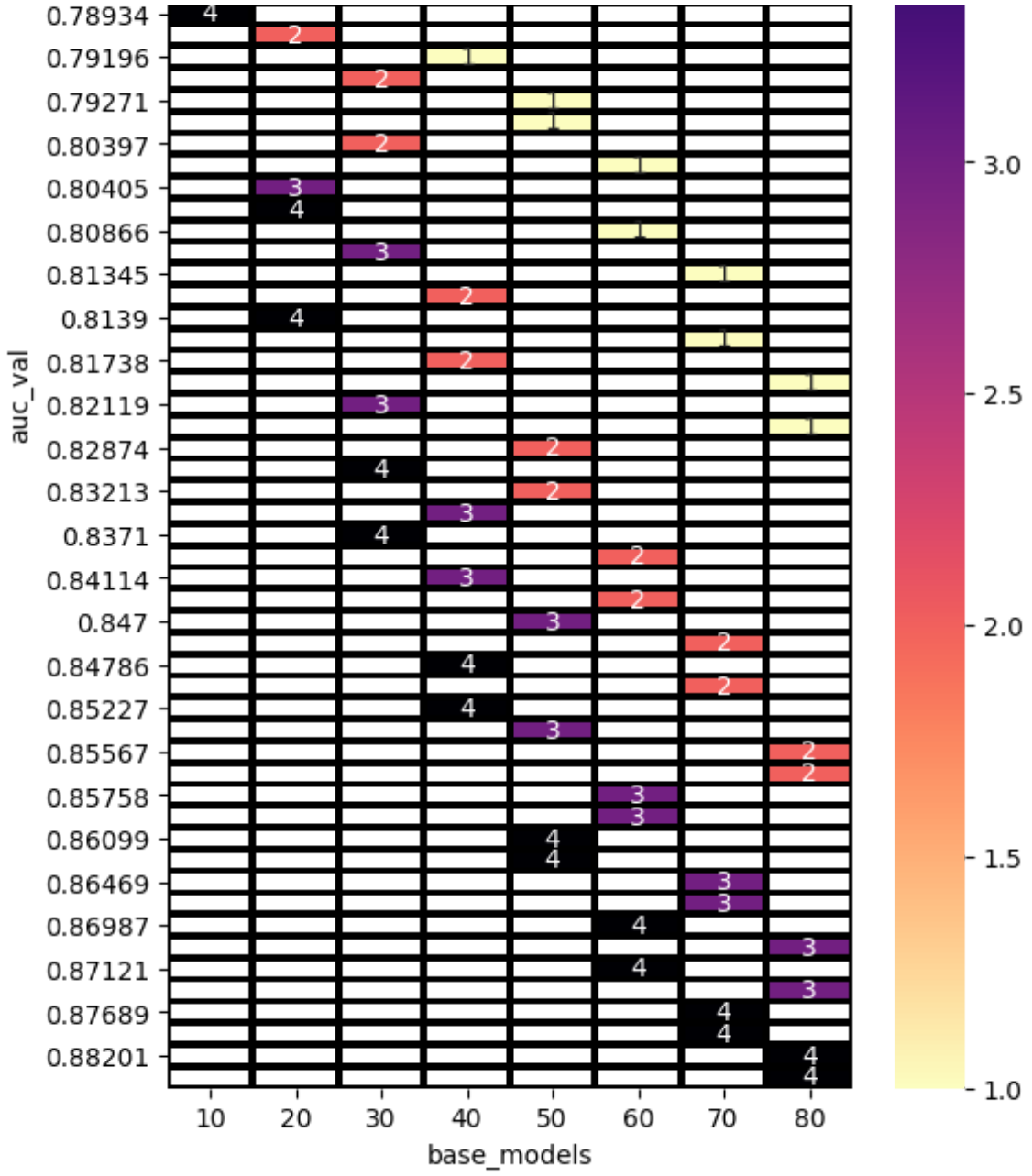
	top25_neg	top25_pos
3	[0.0006737732, lavassa]	[0.016192576, steamer]
4	[0.0006645846, loganberri]	[0.016128158, mooccan]
5	[0.00066255784, notat]	[0.015331737, goldfish]
6	[0.00064665114, gruel]	[0.0146966595, blanco]
7	[0.00063473836, scrumtuuti]	[0.014441184, mts]
8	[0.00062988885, emin]	[0.014279134, fridg]
9	[0.0005922614, gir]	[0.014184785, giftabl]
10	[0.00057439297, outsid]	[0.014178334, goya]
11	[0.0005629583, pug]	[0.013659703, gain]
12	[0.0005547379, seldom]	[0.013287031, odwalla]
13	[0.00053238694, troubl]	[0.013276248, inulin]
14	[0.00052860816, chaim]	[0.01322543, burma]
15	[0.00051889895, tripper]	[0.013080249, kendrel]
16	[0.00050525484, arabian]	[0.01298228, simplysmooth]
17	[0.00041220707, insist]	[0.012599087, hersey]
18	[0.00039616018, towel]	[0.012185354, predominatey]
19	[0.00029157574, nex]	[0.012160459, wudgi]
20	[0.00028238425, butter]	[0.011986482, tase]
21	[0.0002600626, mealsav]	[0.011838709, bimpl]
22	[0.00021107327, dad]	[0.011832709, satisi]
23	[0.00015861144, bhut]	[0.011683925, vinyl]
24	[0.00013958059, gum]	[0.011578206, spra]

Heatmap

In [44]:
```
#creating dataframe for pivot table:
auc_scores = []
no_base_models = []
no_depths = []

for i in range(len(cv_list)):
    auc_scores.append(cv_list[i][0])
    no_base_models.append(cv_list[i][1][0])
    no_depths.append(cv_list[i][1][1])

df_dict = dict(base_models=no_base_models,
               depths=no_depths,
               auc_val=auc_scores)
d_dframe = pd.DataFrame(df_dict)
#d_dframe.head(4)



#plotting heatmap:
fig = plt.figure(figsize=(6, 10), dpi=100)
d_pivot = d_dframe.pivot('auc_val','base_models', 'depths')
sns.heatmap(d_pivot, annot=True, cmap='magma_r', linecolor='black', lin
ewidth=2)
plt.title(' Heatmap of Cross_Validation AUC value vs no_base_models and
 depths', size=10, color='green')
```

Out[44]:
```
Text(0.5, 1.0, ' Heatmap of Cross_Validation AUC value vs no_base_model
s and depths')
```





AvgW2V - XGboost

In [13]:
```
# %%time

# import pickle

# file1 = open('AvgW2V_train.pickle', 'wb')
# pickle.dump(train_vector, file1)
# file1.close()

# file1 = open('AvgW2V_cv.pickle', 'wb')
# pickle.dump(cv_vector, file1)
# file1.close()

# file1 = open('AvgW2V_test.pickle', 'wb')
# pickle.dump(test_vector, file1)
# file1.close()
```

```
CPU times: user 560 ms, sys: 1.87 s, total: 2.43 s
Wall time: 2.42 s
```

W2V instance creation

In [36]:
```
# import pickle

# train_vector = pickle.load(open('AvgW2V_train.pickle', 'rb'))
# cv_vector = pickle.load(open('AvgW2V_cv.pickle', 'rb'))
# test_vector = pickle.load(open('AvgW2V_test.pickle', 'rb'))

# print(train_vector.shape)
# print(cv_vector.shape)
# print(test_vector.shape)
```

In [19]:
```
df.columns
```

Out[19]:
```
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 [20]:
```
df.w2v_new_feat.head(3)
```

Out[20]:
```
0    every book educational witty little book makes...
1    whole series great way spend time child rememb...
2    entertainingl funny beetlejuice well written m...
Name: w2v_new_feat, dtype: object
```

In [21]:
```
#train-test-cv split:
xtrain, xtest, ytrain, ytest = train_test_split(df.w2v_new_feat, df.Sco
re, 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 [22]:
```
%%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 1.66 s, sys: 44 ms, total: 1.71 s
Wall time: 1.7 s
```

instantiating word2vec object for Train, cv, test data

In [23]:

```
%%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 2min 18s, sys: 1.21 s, total: 2min 19s
Wall time: 48.9 s
```

Avg-W2V for train, cv, test data

In [24]:

```
%%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 15min 58s, sys: 4.45 s, total: 16min 2s
Wall time: 16min 1s
```

common class creation

In [26]:

```
class XGB:

    '''building the XGboost classifier based off hypeparameters no_of_base_models and max_depth of base_models'''

    #instantiating the instance attributes:
    def __init__(self, xtr, ytr, xcv, ycv, maximum_depth=[1, 2, 3, 4], estimators=[10, 20, 30, 40]):
        self.xtr = xtr
        self.ytr = ytr
        self.xcv = xcv
        self.ycv = ycv
        self.estimators = estimators
        self.maximum_depth = maximum_depth
```

```
    #creating a method of calling RF classifier:
    def classfier(self, auc_dict_cv={}, auc_dict_tr={}):
        for estimator in self.estimators:
            for depths in self.maximum_depth:
                clf = XGBClassifier(booster='gbtree',silent=0,max_depth
=depths, n_estimators=estimator, learning_rate=0.1)
                print(depths, estimator)
                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[auc_val] = [estimator, depths]

                #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[auc_val] = [estimator, depths]

        return auc_dict_tr, auc_dict_cv
```

Avg W2V instance classifier

In [27]:
```
import time
from xgboost.sklearn import XGBClassifier
start = time.time()
w2v_instance = XGB(train_vector, ytr, cv_vector, ycv)

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

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

```
1 10
```

```
2 10
3 10
4 10
1 20
2 20
3 20
4 20
1 30
2 30
3 30
4 30
1 40
2 40
3 40
4 40
time is(in seconds):  436.2052698135376
```

sorting the list based off AUC score

In [28]:
```
#creating dictionary :

train_list = [[np.round(x, 5), y] for x, y in dictionary_train.items()]
cv_list = [[np.round(x, 5), y] for x, y in dictionary_cv.items()]

print(train_list[0])

print()
print(cv_list[0])

tr_list = sorted(train_list, key=lambda x: x[0], reverse=True)
cv_list = sorted(cv_list, key=lambda x: x[0], 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])
```

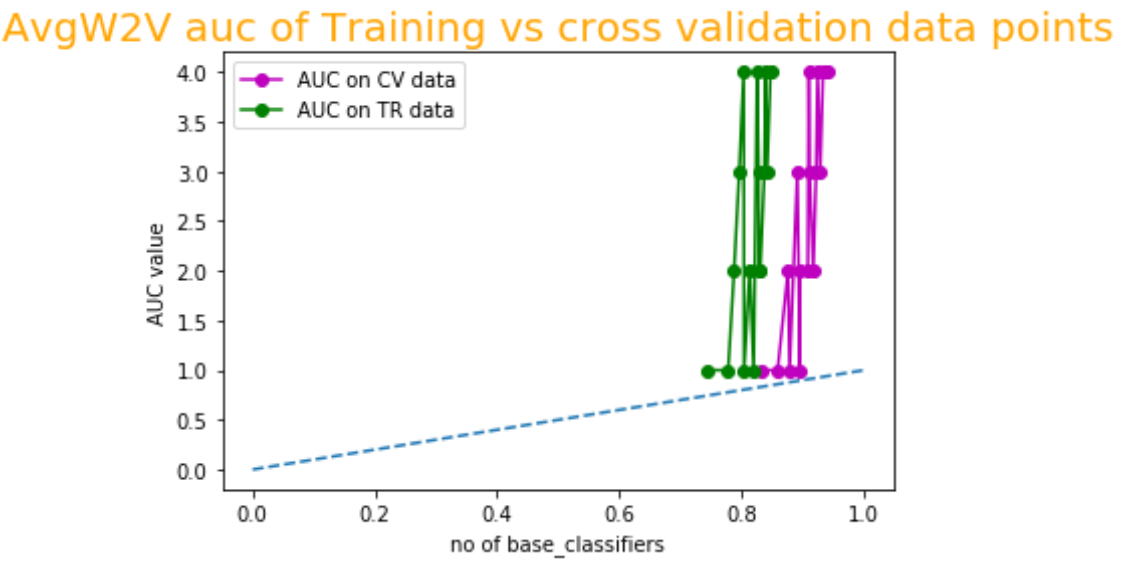
```
[0.83373, [10, 1]]
```

```
[0.74279, [10, 1]]
sorted train list score based off AUC score is:
 [[0.93905, [40, 4]], [0.93296, [30, 4]], [0.92832, [40, 3]]]
****************************************

sorted CV list score based off AUC score is:
 [[0.847, [40, 4]], [0.84146, [40, 3]], [0.83885, [30, 4]]]
```

Plotting AUC Curve on training and cv data based off depth

In [29]:
```
plt.plot([x[0] for x in tr_list], [x[1][1] for x in tr_list], linestyle
='-', color='m', marker='o',label='AUC on CV data')
plt.plot([x[0] for x in cv_list], [x[1][1] for x in cv_list], linestyle
='-', color='g', marker='o', label='AUC on TR data')
plt.plot([0, 1], [0, 1], linestyle='--')
plt.xlabel("no of base_classifiers")
plt.ylabel('AUC value')
plt.title('AvgW2V auc of Training vs cross validation data points', siz
e=20, color='orange')
plt.legend()
```

Out[29]:
```
<matplotlib.legend.Legend at 0x7f98e48b0e48>
```



Optimal Avg-W2V-XGBoost[depth = 4 and basemodels = 40]

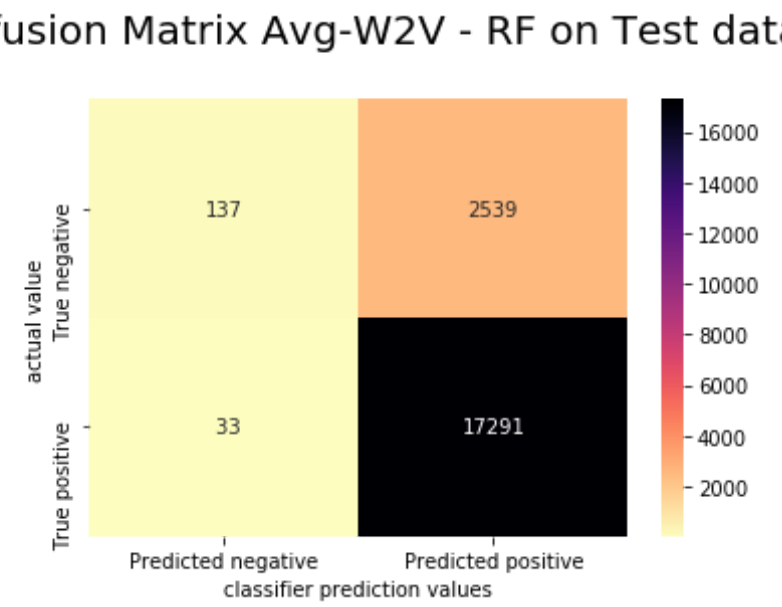
In [31]:
```
clfxgb = XGBClassifier(booster='gbtree',max_depth=4, n_estimators=40, c
lass_weight='balanced')
clfxgb.fit(train_vector, ytr)

#prediction on training data:
y_pred_tr = clfxgb.predict_proba(train_vector)
fpr_tr, tpr_tr, thresholds_tr= roc_curve(ytr, y_pred_tr[:,1])
auc_tr = auc(fpr_tr, tpr_tr)

#prediction on test data:
ypred = clfxgb.predict_proba(test_vector)

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

print(auc_test)
```

```
0.8496831167109989
```

plotting confusion matrix on test data

In [32]:
```
%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 Avg-W2V - RF on Test data\n", size=20)
plt.xlabel("classifier prediction values")
plt.ylabel("actual value")
plt.show()
```

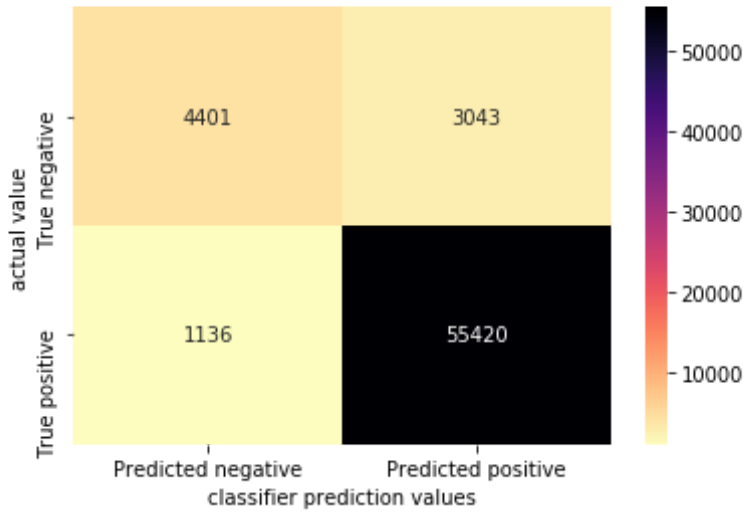
```
CPU times: user 0 ns, sys: 0 ns, total: 0 ns
Wall time: 6.44 µs
```



plotting confusion matrix on training data

In [33]:
```
%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 Avg-W2V - RF on Training data\n", size=20)
plt.xlabel("classifier prediction values")
plt.ylabel("actual value")
plt.show()
```

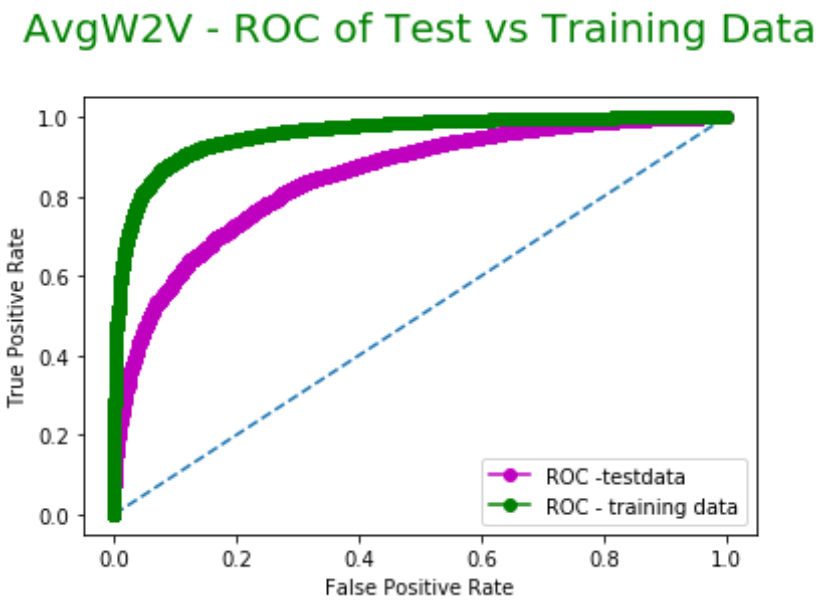
```
CPU times: user 0 ns, sys: 0 ns, total: 0 ns
Wall time: 10.5 µs
```



ROC curve training vs test data

In [34]:
```
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('AvgW2V - ROC of Test vs Training Data\n', size=20, color='gr
een')
plt.legend()
```

Out[34]:
```
<matplotlib.legend.Legend at 0x7f98e8509748>
```

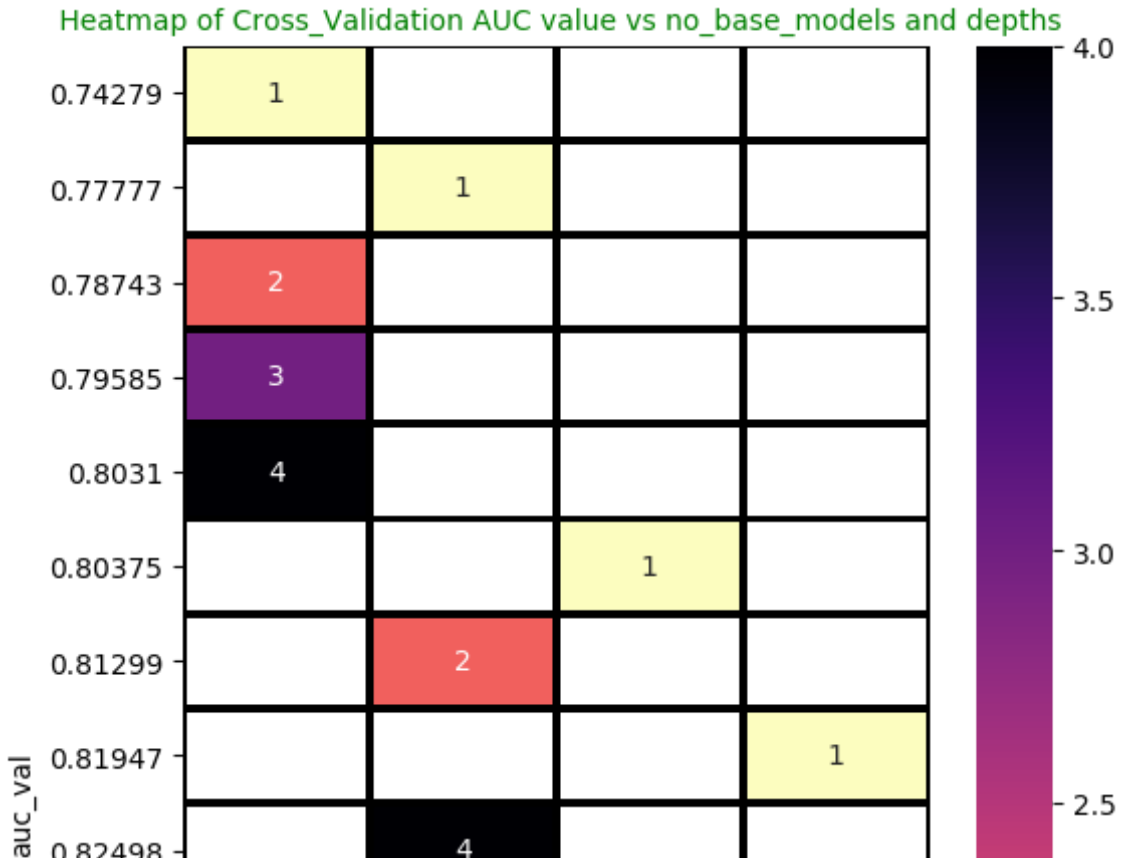


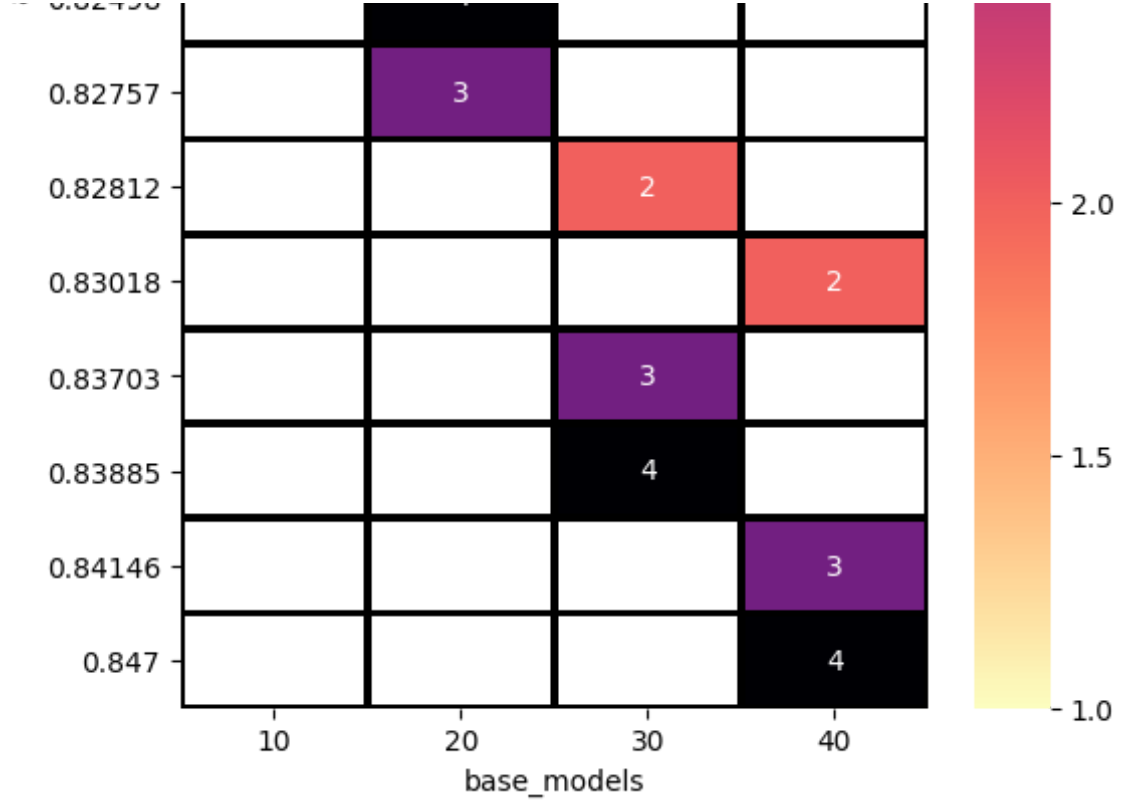
Heatmap

In [35]:
```
#creating dataframe for pivot table:
auc_scores = []
no_base_modesl = []
no_depths = []

for i in range(len(cv_list)):
    auc_scores.append(cv_list[i][0])
    no_base_modesl.append(cv_list[i][1][0])
    no_depths.append(cv_list[i][1][1])

df_dict = dict(base_models=no_base_modesl,
               depths=no_depths,
               auc_val=auc_scores)
d_dframe = pd.DataFrame(df_dict)
```

```
#d_dframe.head(4)



#plotting heatmap:
fig = plt.figure(figsize=(6, 10), dpi=100)
d_pivot = d_dframe.pivot('auc_val','base_models', 'depths')
sns.heatmap(d_pivot, annot=True, cmap='magma_r', linecolor='black', lin
ewidth=2)
plt.title(' Heatmap of Cross_Validation AUC value vs no_base_models and
 depths', size=10, color='green')
```

Out[35]:
```
Text(0.5, 1.0, ' Heatmap of Cross_Validation AUC value vs no_base_model
s and depths')
```





TFidf-W2V

importing pickeled tfidf vector files for train test nd cv data

In [5]:
```
%%time

import pickle

train_vector = pickle.load(open('tfidfw2v_train.pickle', 'rb'))
cv_vector = pickle.load(open('tfidfw2v_cv.pickle', 'rb'))
test_vector = pickle.load(open('tfidfw2v_test.pickle', 'rb'))
```

```
CPU times: user 1.01 s, sys: 1.81 s, total: 2.82 s
```

```
Wall time: 2.81 s
```

conversion of list into vectors

In [6]:
```
%%time
train_vector = np.array(train_vector)
cv_vector = np.array(cv_vector)
test_vector = np.array(test_vector)
```

```
CPU times: user 336 ms, sys: 348 ms, total: 684 ms
Wall time: 685 ms
```

In [7]:
```
train_vector.shape, test_vector.shape, cv_vector.shape
```

Out[7]:
```
((64000, 1000), (20000, 1000), (16000, 1000))
```

list of words creation fo rTFidf vectorizers

In [8]:
```
%%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, sh
uffle=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)
```

```
CPU times: user 1.43 s, sys: 324 ms, total: 1.75 s
Wall time: 1.77 s
```

TFidf dictionary creation

In [9]:
```
%%time
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_)))

print(xtr.shape, xcv.shape, xtest.shape)
print(len(train_list))
```

```
(64000, 43852) (16000, 43852) (20000, 43852)
64000
CPU times: user 4.3 s, sys: 40 ms, total: 4.34 s
Wall time: 4.04 s
```

XGBoost class creation

In [10]:

```
class XGB:

    '''building the XGboostRandom Forest classifier based off hypeparam
eters no_of_base_models and max_depth of base_models'''

    #instantiating the instance attributes:
    def __init__(self, xtr, ytr, xcv, ycv, maximum_depth=[1, 2, 3, 4],
estimators=[10, 20, 30, 40]):
        self.xtr = xtr
        self.ytr = ytr
        self.xcv = xcv
        self.ycv = ycv
        self.estimators = estimators
        self.maximum_depth = maximum_depth

    #creating a method of calling RF classifier:
    def classfier(self, auc_dict_cv={}, auc_dict_tr={}):
        for estimator in self.estimators:
            for depths in self.maximum_depth:
                clf = XGBClassifier(booster='gbtree',max_depth=depths,
n_estimators=estimator)
                print(depths, estimator)
                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[auc_val] = [estimator, depths]

                #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[auc_val] = [estimator, depths]

        return auc_dict_tr, auc_dict_cv
```

TFIDF - W2V instance creation

In [12]:
```
print(train_vector.shape, test_vector.shape, cv_vector.shape)
```

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

In [13]:
```
import time
from xgboost.sklearn import XGBClassifier
start = time.time()
tfidfw2v_instance = XGB(train_vector, ytr, cv_vector, ycv)

dictionary_train, dictionary_cv = tfidfw2v_instance.classfier()
end = time.time()

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

```
1 10
2 10
3 10
4 10
1 20
2 20
3 20
4 20
1 30
2 30
3 30
4 30
1 40
2 40
3 40
4 40
time is(in seconds):  441.2281515598297
```

sorting the list based off AUC score

In [14]:
```
#creating dictionary :

train_list = [[np.round(x, 5), y] for x, y in dictionary_train.items()]
cv_list = [[np.round(x, 5), y] for x, y in dictionary_cv.items()]

print(train_list[0])

print()
print(cv_list[0])

tr_list = sorted(train_list, key=lambda x: x[0], reverse=True)
cv_list = sorted(cv_list, key=lambda x: x[0], 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])
```

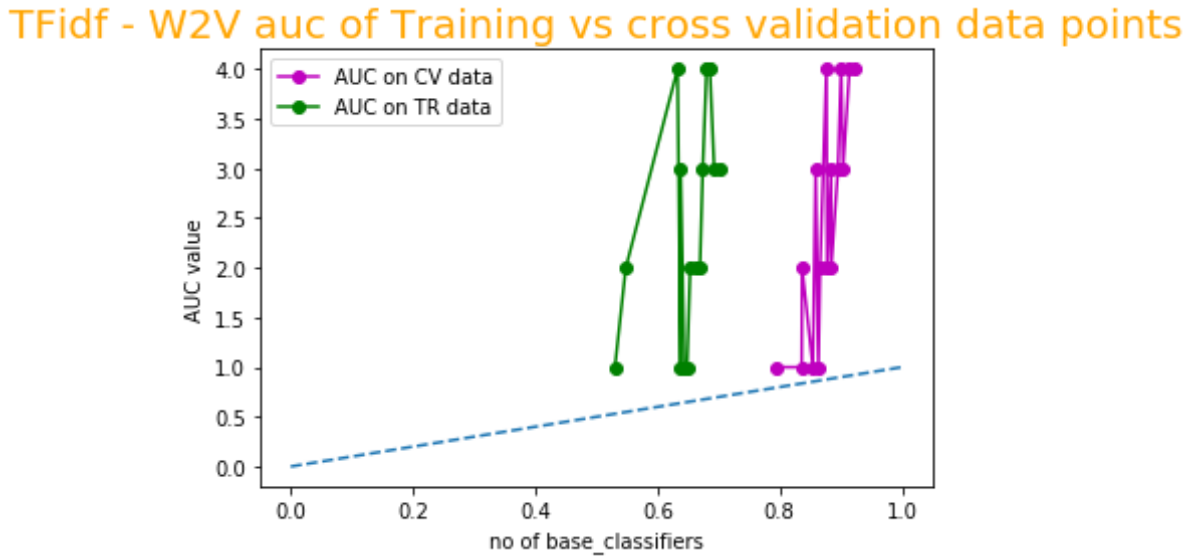
```
[0.79524, [10, 1]]

[0.53041, [10, 1]]
sorted train list score based off AUC score is:
 [[0.92383, [40, 4]], [0.91417, [30, 4]], [0.90366, [40, 3]]]
****************************************

sorted CV list score based off AUC score is:
 [[0.70259, [40, 3]], [0.69279, [30, 3]], [0.68579, [40, 4]]]
```

Plotting AUC Curve on training and cv data based off depth

In [15]:
```
plt.plot([x[0] for x in tr_list], [x[1][1] for x in tr_list], linestyle
='-', color='m', marker='o',label='AUC on CV data')
plt.plot([x[0] for x in cv_list], [x[1][1] for x in cv_list], linestyle
='-', color='g', marker='o', label='AUC on TR data')
plt.plot([0, 1], [0, 1], linestyle='--')
```

```
plt.xlabel("no of base_classifiers")
plt.ylabel('AUC value')
plt.title('TFidf - W2V auc of Training vs cross validation data points'
, size=20, color='orange')
plt.legend()
```

Out[15]: <matplotlib.legend.Legend at 0x7fdd973293c8>



Optimal TFidf - XGBoost[depth = 3 and basemodels = 40]

In [16]:
```
%%time
clfxgb = XGBClassifier(booster='gbtree',max_depth=3, n_estimators=40, c
lass_weight='balanced')
clfxgb.fit(train_vector, ytr)

#prediction on training data:
y_pred_tr = clfxgb.predict_proba(train_vector)
fpr_tr, tpr_tr, thresholds_tr= roc_curve(ytr, y_pred_tr[:,1])
auc_tr = auc(fpr_tr, tpr_tr)
```

```
#prediction on test data:
ypred = clfxgb.predict_proba(test_vector)

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

print(auc_test)
```

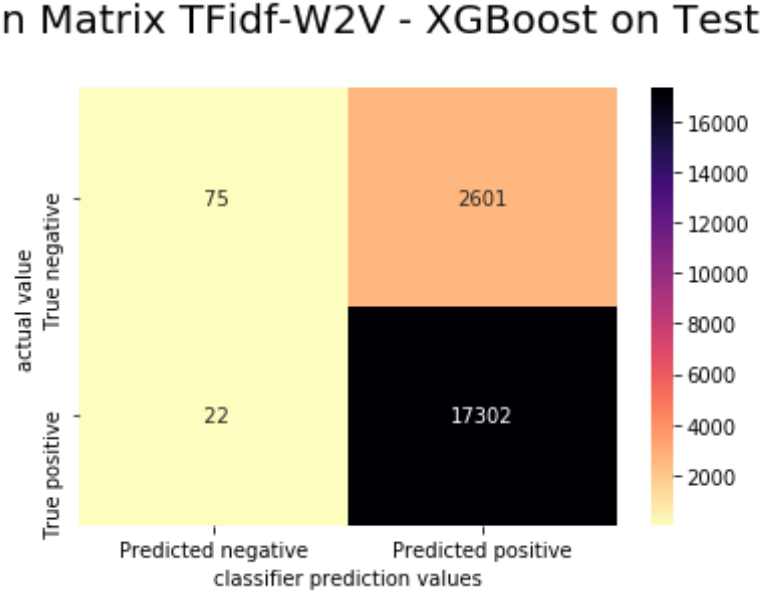
```
0.7852330864428897
CPU times: user 6min 6s, sys: 1.22 s, total: 6min 7s
Wall time: 48.8 s
```

confusion matrix on test data

In [17]:

```
%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 - XGBoost on Test data\n", size=2
0)
plt.xlabel("classifier prediction values")
plt.ylabel("actual value")
plt.show()
```

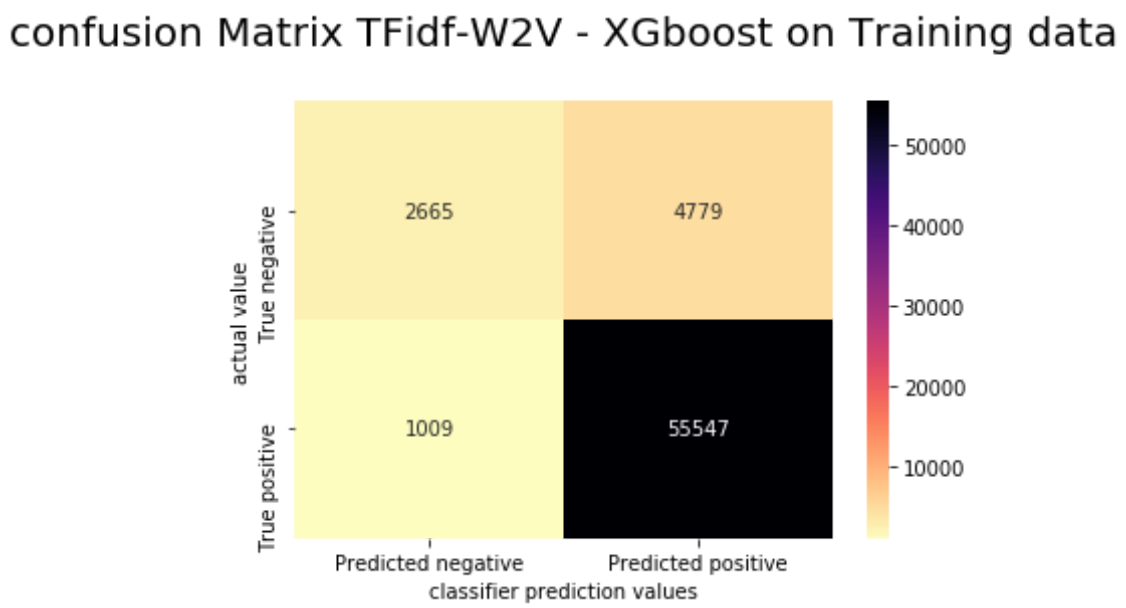
```
CPU times: user 0 ns, sys: 0 ns, total: 0 ns
Wall time: 7.15 µs
```



confusion matrix on training data

In [18]:
```
%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 TFidf-W2V - XGboost on Training data\n", si
ze=20)
plt.xlabel("classifier prediction values")
plt.ylabel("actual value")
plt.show()
```

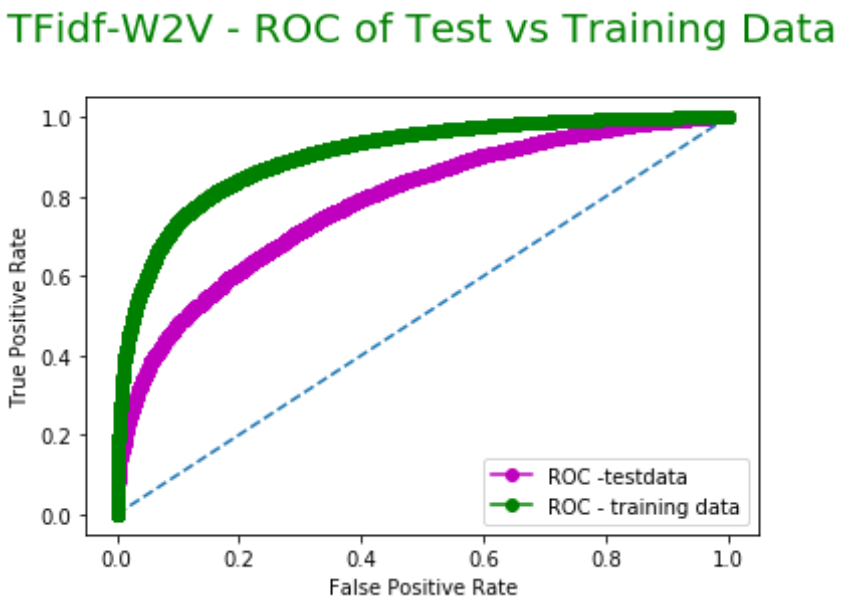
```
CPU times: user 0 ns, sys: 0 ns, total: 0 ns
Wall time: 6.91 µs
```



ROC curve on test data and training data

In [19]:
```
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('TFidf-W2V - ROC of Test vs Training Data\n', size=20, color=
'green')
plt.legend()
```

Out[19]:
```
<matplotlib.legend.Legend at 0x7fdd941cdd30>
```

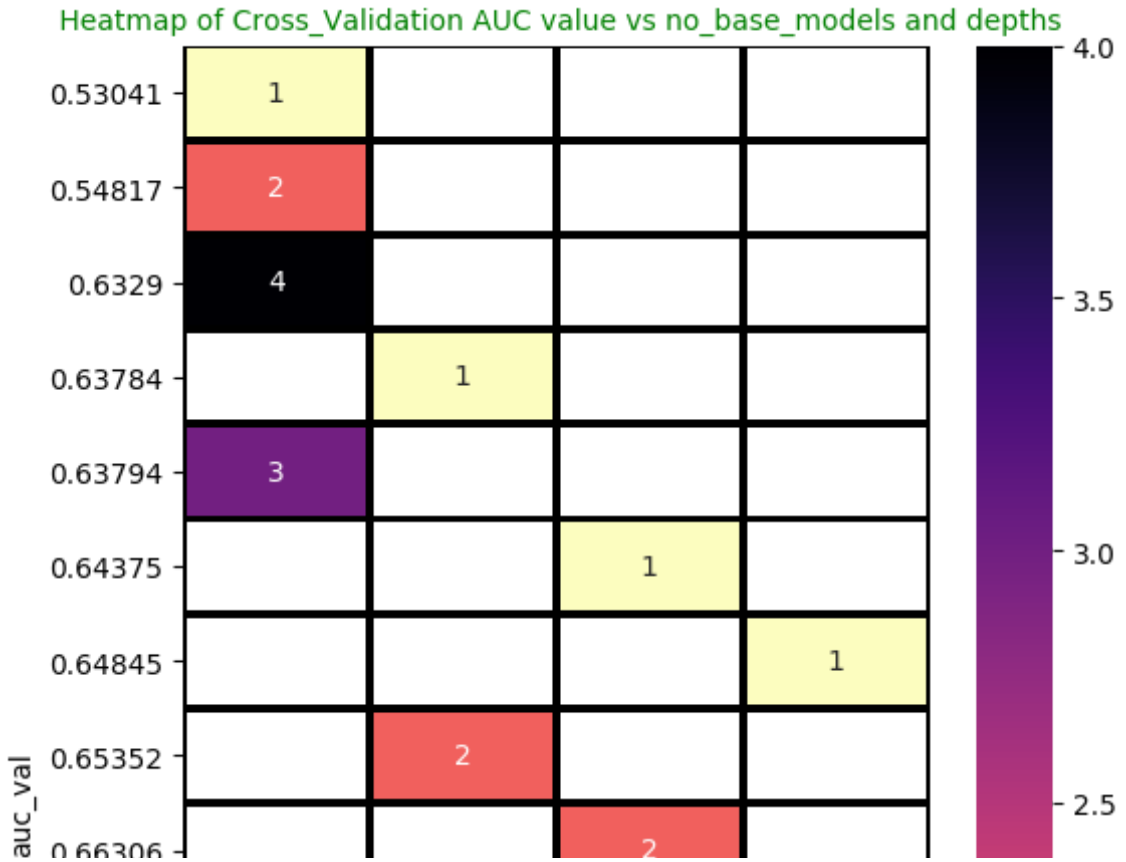


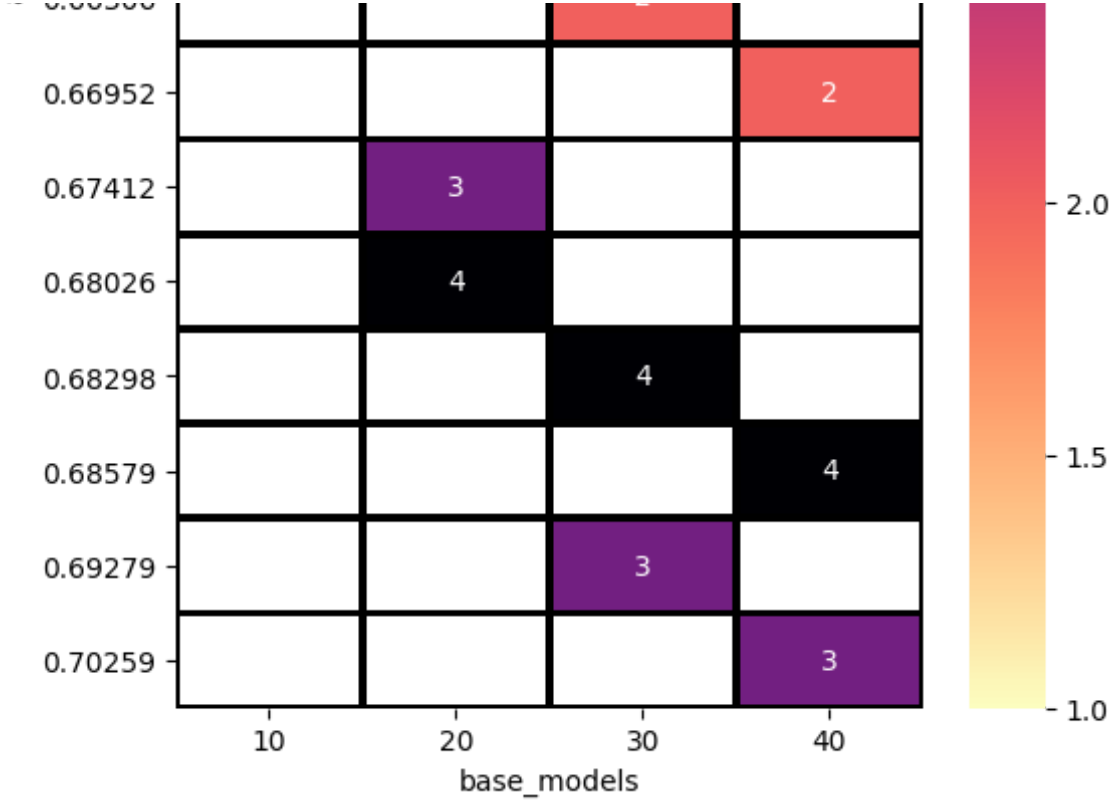
Heatmap of CV data of AUC vs no of base models and depths

In [20]:
```
#creating dataframe for pivot table:
auc_scores = []
no_base_modesl = []
no_depths = []

for i in range(len(cv_list)):
    auc_scores.append(cv_list[i][0])
    no_base_modesl.append(cv_list[i][1][0])
    no_depths.append(cv_list[i][1][1])

df_dict = dict(base_models=no_base_modesl,
               depths=no_depths,
               auc_val=auc_scores)
d_dframe = pd.DataFrame(df_dict)
```

```
#d_dframe.head(4)



#plotting heatmap:
fig = plt.figure(figsize=(6, 10), dpi=100)
d_pivot = d_dframe.pivot('auc_val','base_models', 'depths')
sns.heatmap(d_pivot, annot=True, cmap='magma_r', linecolor='black', lin
ewidth=2)
plt.title(' Heatmap of Cross_Validation AUC value vs no_base_models and
 depths', size=10, color='green')
```

Out[20]:
```
Text(0.5, 1.0, ' Heatmap of Cross_Validation AUC value vs no_base_model
s and depths')
```





Performance-Measurement

In [24]:
```
perf_dict = dict(Algorithm = ['Bow-RF', 'Tfidf-RF', 'AvgW2V-RF', 'Tfidf
W2V-RF','Bow-XGB', 'Tfidf-XGB', 'AvgW2V-XGB', 'TfidfW2V-XGB' ],
                 AUC = [0.875,0.834,0.713,0.768, 0.885,0.88,0.849, 0.785
 ],
                 Classifiers = [15,15,5,13, 80,80,40, 40],
                 Depth =[13,13,9,13,4,4,4, 3])

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

Out[24]:

	AUC	Algorithm	Classifiers	Depth

	AUC	Algorithm	Classifiers	Depth
0	0.875	Bow-RF	15	13
1	0.834	Tfidf-RF	15	13
2	0.713	AvgW2V-RF	5	9
3	0.768	TfidfW2V-RF	13	13
4	0.885	Bow-XGB	80	4
5	0.880	Tfidf-XGB	80	4
6	0.849	AvgW2V-XGB	40	4
7	0.785	TfidfW2V-XGB	40	3

Conclusion - Out of 1L datapoints

1. BoW AUC performace was best amongts all the four vectorizer with ~87 % AUC score.

2. Applied Feature engineering on BoW vectorizers and observed AUC score was increased by 1%(~ 88.5%)

3. AvgW2V-RF performace was worst amongts all the four vectorizer with ~71% AUC score.

4. Wordcloud creation done for BoW and TF-IDF vectorizers

5. Heatmap is created of AUC_Score wrt Depth and no_of_classifiers for all the 8 Vectorizers

In [ ]: