Assignment: Logistic regression algorithm on Amazon Fine Food Reviews for :

1. BoW

2. Tf-Idf

3. AvgW2V

4. TF-idf-W2V Vectorizers

In [1]:
```
#imporing general purpose libraries:
import time
import datetime
import os
import sys
import pickle
import random
import psutil

#importing EDA libraries:
import math
import pandas as pd
import numpy as np
import scipy as sc
import matplotlib.pyplot as plt
import seaborn as sns
%matplotlib inline
import warnings
warnings.filterwarnings('ignore')
```

importing performance metric libraries

In [2]:
```
#importing Logistic regression libraries:
from sklearn.linear_model import LogisticRegression

#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/jalesh_j/Data_Preprocessing/cleaned.sqlite')
df = pd.read_sql_query("""select * from cleandf""", con)
```

```
CPU times: user 2.44 s, sys: 1.25 s, total: 3.69 s
Wall time: 6.83 s
```

feature engineered / non engineered columns for BoW and {tf-idf, avg-w2v, tfidf-w2v}

In [4]:
```
df.columns
```

Out[4]:
```
Index(['index', 'Score', 'Time', 'Text', 'Summary', 'cleanedtext',
       'numeric_score', 'bow_feat', 'bow_new_feat', 'tfw2v_feat'],
      dtype='object')
```

In [5]:
```
vectorizer = ['bow']
for i in vectorizer:
```

```
    print('unfeatured preprocesed column for vectorizer: {} is: \n {}'.
format(i, df.cleanedtext.head(1)))
    print()
    print('featured preprocesed column for vectorizer: {} is: \n {}'.fo
rmat(i, df.bow_new_feat.head(1)))
```

```
unfeatured preprocesed column for vectorizer: bow is:
 0    bought sever vital can dog food product found ...
Name: cleanedtext, dtype: object

featured preprocesed column for vectorizer: bow is:
 0    good qualiti dog food bought sever vital can d...
Name: bow_new_feat, dtype: object
```

In [6]:
```
vectorizer = ['tfidf', 'avg-w2v', 'tfidf-w2v']
for i in vectorizer:
    print('UNFEATURED COLUMN for vectorizer: {} is: \n {}'.format(i, df
.Text.head(1)))
    print()
    print('FEATURED COLUMN for vectorizer: {} is: \n {}'.format(i, df.t
fw2v_feat.head(1)))
```

```
UNFEATURED COLUMN for vectorizer: tfidf is:
 0    i have bought several of the vitality canned d...
Name: Text, dtype: object

FEATURED COLUMN for vectorizer: tfidf is:
 0    i have bought several of the vitality canned d...
Name: tfw2v_feat, dtype: object
UNFEATURED COLUMN for vectorizer: avg-w2v is:
 0    i have bought several of the vitality canned d...
Name: Text, dtype: object

FEATURED COLUMN for vectorizer: avg-w2v is:
 0    i have bought several of the vitality canned d...
Name: tfw2v_feat, dtype: object
UNFEATURED COLUMN for vectorizer: tfidf-w2v is:
 0    i have bought several of the vitality canned d...
Name: Text, dtype: object
```

```
FEATURED COLUMN for vectorizer: tfidf-w2v is:

 0    i have bought several of the vitality canned d...
Name: tfw2v_feat, dtype: object
```

In [7]:
```
for i in df.tfw2v_feat.head(1):
    print(i)
print('\n' * 2)
for i in df.Text.head(1):
    print(i)
```

```
i have bought several of the vitality canned dog food products and have
found them all to be of good quality  the product looks more like a ste
w than a processed meat and it smells better  my labrador is finicky an
d she appreciates this product better than  most  good quality dog food



i have bought several of the vitality canned dog food products and have
found them all to be of good quality  the product looks more like a ste
w than a processed meat and it smells better  my labrador is finicky an
d she appreciates this product better than  most
```

sorting dataframe based on time

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

```
364171
```

Out[8]:
```
117924    939340800
117901    940809600
298792    944092800
169281    944438400
```

```
298791    946857600
169342    947376000
169267    948240000
63317     948672000
Name: Time, dtype: int64
```

taking out 1L datapoints, saving it into vectorizers seperately

In [9]:
```
%%time
d = df.head(100000)
#bow
bow = d['cleanedtext']
bow_featured = d.bow_new_feat

#tfidf
tfidf = d.Text
tfidf_featured = d.tfw2v_feat

#w2v
w2v = d['Text']
w2v_featured = d['tfw2v_feat']

#class labels:
y = d['numeric_score'].apply(lambda x: 0 if int(x) < 3 else 1)
print(y.value_counts())
```

```
1    87729
0    12271
Name: numeric_score, dtype: int64
CPU times: user 60 ms, sys: 4 ms, total: 64 ms
Wall time: 58.8 ms
```

1. LR on BoW [L1 reg.]

train, cv, test split

In [10]:
```
xtrain, xtest, ytrain, ytest = train_test_split(bow, y, 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]:
```
%%time
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, 31507)
(16000, 31507)
(20000, 31507)
CPU times: user 3.82 s, sys: 32 ms, total: 3.85 s
Wall time: 4.09 s
```

column standardization

In [12]:
```
sc = StandardScaler(with_mean=False)
xtr = sc.fit_transform(xtr)
xcv = sc.transform(xcv)
xtest = sc.transform(xtest)
```

Logistic regression on Bow

In [13]:
```
%%time
from sklearn.linear_model import LogisticRegression

auc_cv_dict = {}
```

```
auc_tr_dict = {}

c_val = [10 ** -4, 10 ** -3, 10 ** -2, 10 ** -1, 10**1, 10 ** 2, 10 **
3, 10** 4]

for i in c_val:
    lr = LogisticRegression(penalty='l1', C=i, class_weight='balanced',
 random_state=56, n_jobs=-1,max_iter=4, solver='liblinear')
    lr.fit(xtr, ytr)
    y_pred_cv = lr.predict_proba(xcv)
    fpr_cv, tpr_cv, thresholds_cv = roc_curve(ycv, y_pred_cv[:,1])
    auc_cv_dict[i] = auc(fpr_cv, tpr_cv)
    #performance metrics for training data:
    y_pred_tr = lr.predict_proba(xtr)
    fpr_tr, tpr_tr, thresholds_tr = roc_curve(ytr, y_pred_tr[:,1])
    auc_tr_dict[i] = auc(fpr_tr, tpr_tr)
```

```
CPU times: user 2.84 s, sys: 20 ms, total: 2.86 s
Wall time: 2.91 s
```

optimal C value for AUC score on training and CV data

In [14]:

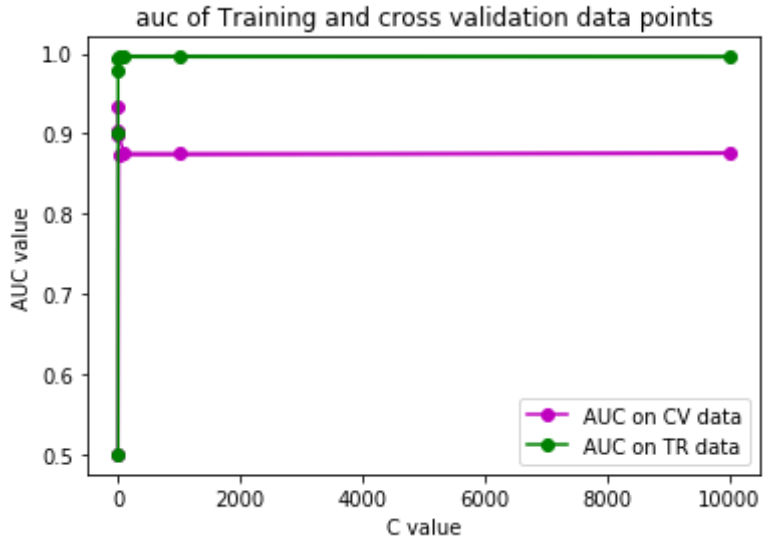
```
#sorting dictionary wrt higest AuC Score of both training and cv data:

cv_tup = sorted(auc_cv_dict.items(), key= lambda x: x[1],reverse=True)
tr_tup = sorted(auc_tr_dict.items(), key= lambda x: x[1],reverse=True)
print(cv_tup)
print('*' * 70)
print(tr_tup)
```

```
[(0.01, 0.9344722722430896), (0.1, 0.9036619905656527), (0.001, 0.89948
6394637262), (100, 0.8754335943456197), (1000, 0.8753183629109675), (10
000, 0.8752821223898771), (10, 0.8734602140821538), (0.0001, 0.5)]
**********************************************************************
[(10, 0.9958451742124992), (10000, 0.9956293167702406), (1000, 0.995629
2763857568), (100, 0.9956199665744278), (0.1, 0.9944117245792161), (0.0
1, 0.9796718107875941), (0.001, 0.9003041832973238), (0.0001, 0.5)]
```

Plotting AUC Curve on training and cv data

In [15]:
```
plt.plot([x[0] for x in cv_tup], [x[1] for x in cv_tup], linestyle='-',
 color='m', marker='o',label='AUC on CV data')
plt.plot([x[0] for x in tr_tup], [x[1] for x in tr_tup], linestyle='-',
 color='g', marker='o', label='AUC on TR data')
#plt.plot([0, 1], [0, 1], linestyle='--')
plt.xlabel("C value")
plt.ylabel('AUC value')
plt.title('auc of Training and cross validation data points')
plt.legend()
```

Out[15]:
```
<matplotlib.legend.Legend at 0x7f61dd4f1390>
```



optimal LR-BoW(L1) for C=0.01

In [16]:
```
%%time

lr = LogisticRegression(penalty='l1', C=0.01, class_weight='balanced',
random_state=56, n_jobs=-1,max_iter=4, solver='liblinear')
lr.fit(xtr, ytr)
```

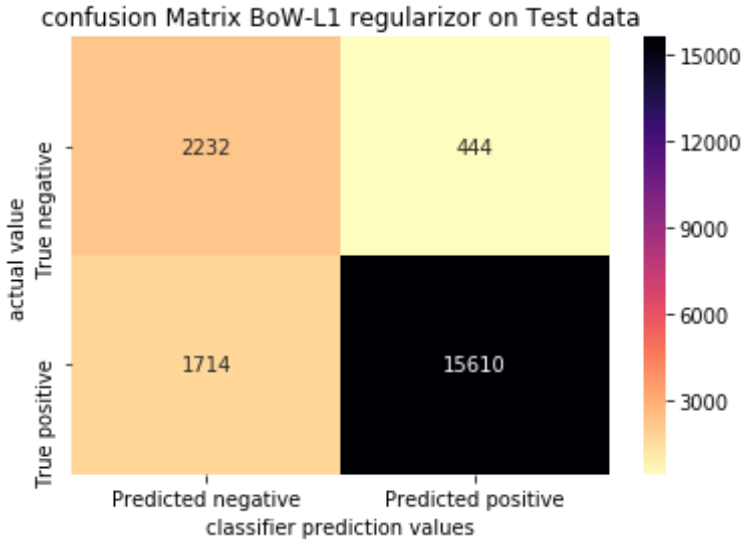
```
y_pred_test = lr.predict_proba(xtest)
y_pred = lr.predict(xtest)
fpr_test, tpr_test, thresholds_test= roc_curve(ytest, y_pred_test[:,1])
auc_test = auc(fpr_test, tpr_test)
optimal_weight = lr.coef_
print(auc_test)
```

```
0.9355066340482061
CPU times: user 336 ms, sys: 0 ns, total: 336 ms
Wall time: 336 ms
```

plotting confusion matrix on test data

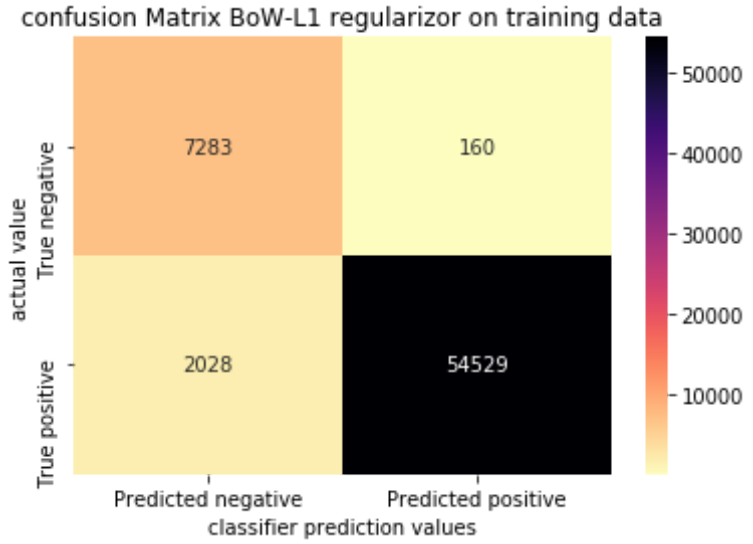
In [17]:
```
%time
y_pred_test = np.where(y_pred_test[:,1] < 0.5, 0, 1)
#creating confusion matrix:

cf = confusion_matrix(ytest, y_pred_test)
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-L1 regularizor on Test data")
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 confusion matrix on train data

In [18]:
```
y_pred_tr = np.where(y_pred_tr[:,1] > 0.5, 1, 0)
#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-L1 regularizor on training data")
plt.xlabel("classifier prediction values")
plt.ylabel("actual value")
plt.show()
```



plotting ROC curve on test vs 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('ROC of Test vs Training Data')
plt.legend()
```

Out[19]:
```
<matplotlib.legend.Legend at 0x7f61d2e12eb8>
```

printing top 25 features of both positive and negative class

In [20]:
```
features = bow_object.get_feature_names()
featuresAndcoeff = sorted(zip(np.ravel(lr.coef_), 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
------------------------------------------------------------------------
------------------------------------------------
-0.44459299956765147                    not
    0.6263077277746858                  great
-0.29936861567813094             disappoint
```

```
    0.4858661827285232          best
-0.20442379509905848         worst
    0.4224379017284834          love
-0.16926739877470445        terribl
   0.41922429303318237         delici
 -0.1620133425170866           aw
     0.406558410874198       perfect
-0.15254359086801572         bland
     0.319615661853075          good
 -0.1492549400049215       unfortun
    0.3094363026669777           excel
 -0.1479858984661613        thought
    0.2806475152469429           nice
 -0.1358089713058439        horribl
   0.25018003929096766         favorit
 -0.1347963227723577           didn
    0.2383404078596342           wonder
-0.13184256169491568          hope
   0.21245829472645633           tasti
-0.13024584914194598         stale
    0.1957801981051137           amaz
 -0.1301248582960169        return
   0.18672092578254104         smooth
-0.12841267392632158         money
     0.185516233531009           keep
  -0.127941874145075             bad
   0.17937000978521725           year
-0.11621134786585013        product
   0.17805179011524236         addict
-0.11555954697958949          tast
    0.1739518463225993           alway
-0.11262203573631353         threw
    0.1726206209166541          find
-0.11100253618316525         would
   0.16849552308994062           happi
-0.11016094303456905          mayb
   0.16552773809233923           thank
-0.10708722390933566          weak
    0.1597557317837326          awesom
```

```
-0.10670303709031166                stick
   0.14021516342894103                 easi
-0.10195329646092088                away
    0.1385230533229843                 yummi
-0.09844665936235886                chang
   0.13744071844451508              refresh
-0.09473271766468103            tasteless
   0.13690124233854306                 hook
```

Multicollinearity on BoW-L1

In [21]:
```
# adding noise on train data
x = xtr
print(type(x))
print(x.shape)
#x[x.nonzero()] = x[x.nonzero()] + np.random.normal(0,1)
x[x.nonzero()] = x[x.nonzero()] + 0.1
x = x.toarray()
print(x.shape)
```

```
<class 'scipy.sparse.csr.csr_matrix'>
(64000, 31507)
(64000, 31507)
```

training logistic regression on perturbed data:

In [22]:
```
%%time
from sklearn.linear_model import LogisticRegression

auc_cv_dict = {}
auc_tr_dict = {}

c_val = [10 ** -4, 10 ** -3, 10 ** -2, 10 ** -1, 10**1, 10 ** 2, 10 **
3, 10** 4]
```

```
for i in c_val:
    lr = LogisticRegression(penalty='l1', C=i, class_weight='balanced',
 random_state=56, n_jobs=-1,max_iter=4, solver='liblinear')
    lr.fit(x, ytr)
    y_pred_cv = lr.predict_proba(xcv)
    fpr_cv, tpr_cv, thresholds_cv = roc_curve(ycv, y_pred_cv[:,1])
    auc_cv_dict[i] = auc(fpr_cv, tpr_cv)
    #performance metrics for training data:
    y_pred_tr = lr.predict_proba(x)
    fpr_tr, tpr_tr, thresholds_tr = roc_curve(ytr, y_pred_tr[:,1])
    auc_tr_dict[i] = auc(fpr_tr, tpr_tr)
```

```
CPU times: user 56 s, sys: 5.44 s, total: 1min 1s
Wall time: 42 s
```

optimal C on perturbed data

In [23]:
```
#sorting dictionary wrt higest AuC Score of both training and cv data:

cv_tup = sorted(auc_cv_dict.items(), key= lambda x: x[1],reverse=True)
tr_tup = sorted(auc_tr_dict.items(), key= lambda x: x[1],reverse=True)
print(cv_tup)
print('\n' * 3)
print(tr_tup)
```

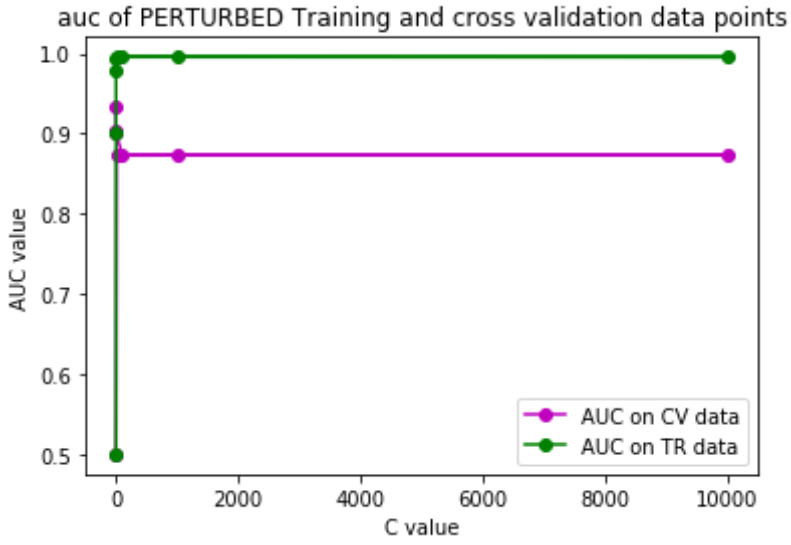
```
[(0.01, 0.9345069020743537), (0.1, 0.9044302896127688), (0.001, 0.90041
11654897893), (100, 0.8741683471530519), (1000, 0.8740628301914144), (1
0000, 0.8740242071916227), (10, 0.8728285216659257), (0.0001, 0.5)]
```

```
[(10, 0.995873984978459), (10000, 0.9956439858401454), (1000, 0.9956439
715867978), (100, 0.9956347472955528), (0.1, 0.9949043036321585), (0.0
1, 0.979605516093857), (0.001, 0.9009483146285113), (0.0001, 0.5)]
```

plotting cv vs perturbed training data

In [24]:
```
plt.plot([x[0] for x in cv_tup], [x[1] for x in cv_tup], linestyle='-',
 color='m', marker='o',label='AUC on CV data')
plt.plot([x[0] for x in tr_tup], [x[1] for x in tr_tup], linestyle='-',
 color='g', marker='o', label='AUC on TR data')
#plt.plot([0, 1], [0, 1], linestyle='--')
plt.xlabel("C value")
plt.ylabel('AUC value')
plt.title('auc of PERTURBED Training and cross validation data points')
plt.legend()
```

Out[24]:
```
<matplotlib.legend.Legend at 0x7f61cc0d6cf8>
```



optimal BoW-LR on PERTURBED train and test data

In [25]:
```
%%time

lr = LogisticRegression(penalty='l1', C=0.01, class_weight='balanced',
random_state=56, n_jobs=-1,max_iter=4, solver='liblinear')
lr.fit(x, ytr)
y_pred_test = lr.predict_proba(xtest)
y_pred = lr.predict(xtest)
```

```
fpr_test, tpr_test, thresholds_test= roc_curve(ytest, y_pred_test[:,1])
auc_test = auc(fpr_test, tpr_test)
optimal_weight_perturbed = lr.coef_
print(auc_test)
```

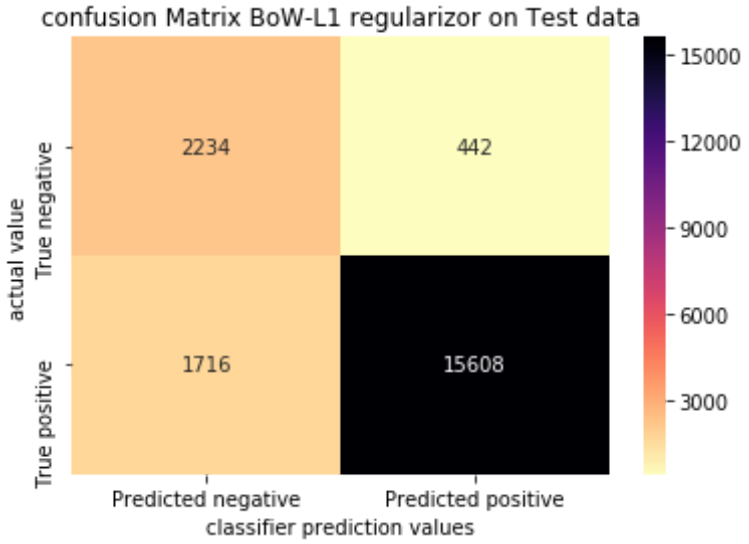
```
0.9353710401668508
CPU times: user 3.55 s, sys: 4 ms, total: 3.56 s
Wall time: 3.55 s
```

confusion matrix on test data

In [26]:

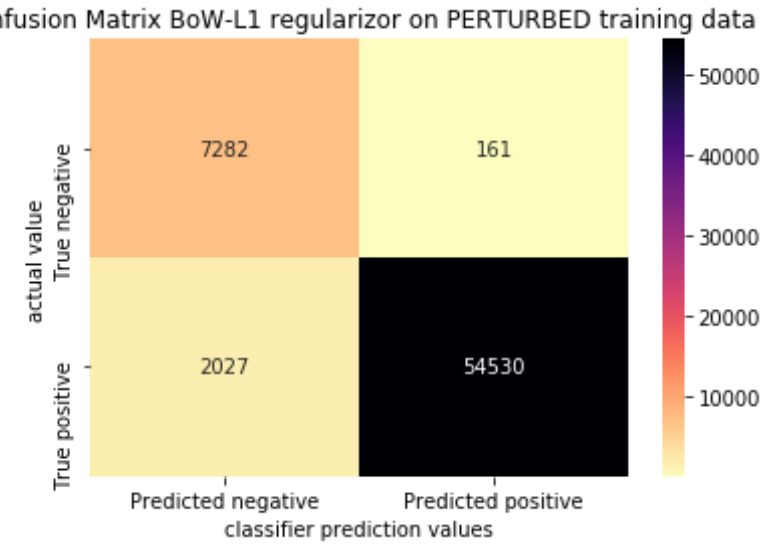
```
%time
y_pred_test = np.where(y_pred_test[:,1] < 0.5, 0, 1)
#creating confusion matrix:

cf = confusion_matrix(ytest, y_pred_test)
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-L1 regularizor on Test data")
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
```



confusion matrix on PERTURBED training data

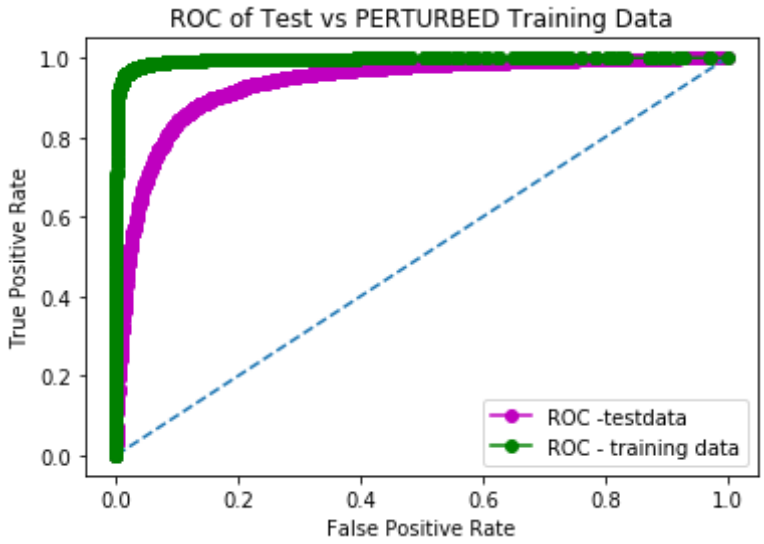
In [27]:
```
y_pred_tr = np.where(y_pred_tr[:,1] > 0.5, 1, 0)
#creating confusion matrix:

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-L1 regularizor on PERTURBED training da
ta")
plt.xlabel("classifier prediction values")
plt.ylabel("actual value")
plt.show()
```



ROC-AUC curve for PERTURBED train and 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('ROC of Test vs PERTURBED Training Data')
plt.legend()
```

Out[28]:
```
<matplotlib.legend.Legend at 0x7f61df7f8ef0>
```



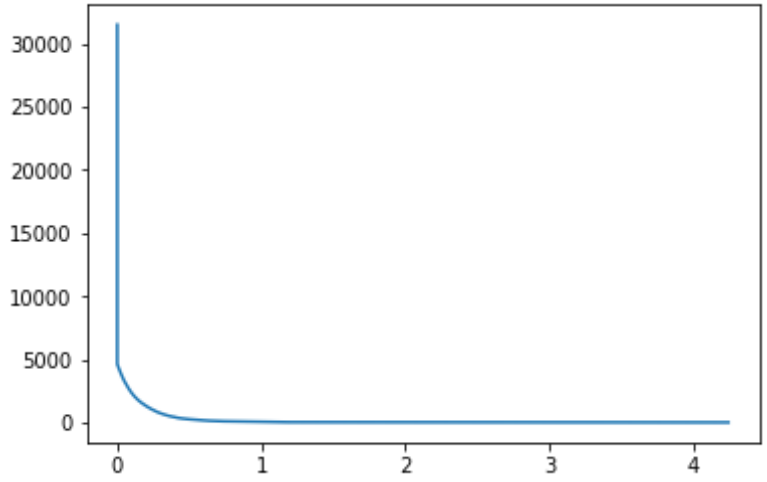
Multicollinearity test through Pertubation

In [29]:
```
w = optimal_weight
w1 = optimal_weight_perturbed
w = w + 1
w1 = w1 + 1

print(w.shape)
print(w1.shape)
print('************************************')

#flattening:
w = np.ravel(w)
w1 = np.ravel(w1)

print(len(w))
print(len(w1))
```

```
weight_diff = np.abs(np.divide(w-w1, w)) * 100
```

```
(1, 31507)
(1, 31507)
**********
31507
31507
```

In [30]:
```
#plotting

plt.plot(np.ravel(sorted(weight_diff, reverse=True)), list(range(1, 315
08)))
plt.show()
```



We could see from the plot above after a value , there's steep change in the value, the one which is to be found

In [31]:
```
print(max(weight_diff))

#computing percentile value
p0, p10, p20, p30, p40, p50, p60, p70, p80, p90, p100 = [np.percentile(
```

```
weight_diff, i) for i in range(0, 101, 10)]
print(p80)
print(p90)
print(p100)
```

```
4.242508849548514
0.0
0.05458863213579915
4.242508849548514
```

we could see there is significant change in the value between 90 percentile and 100 percentile value above hence further segregating between 90 percentile to 100 percentile below

In [32]:
```
p91, p92, p93, p94, p95, p96, p97, p98, p99, p100 = [np.percentile(weig
ht_diff, i) for i in range(91, 101, 1)]

for i in [p91, p92, p93, p94, p95, p96, p97, p98, p99, p100]:
    print(i)
```

```
0.06959069937593673
0.08804774665833284
0.10878275038757088
0.1348072225761367
0.16683568095780857
0.20567477077580504
0.25379184133560884
0.32390403471513984
0.44842580578541363
4.242508849548514
```

we could see between 99 and 100 percentile there's steep difference so further segregating from 99.1 percentile and so on

In [33]:
```
#checking percentile value of 99.1 and so on

print(np.percentile(weight_diff, 99.1))
```

```
print(np.percentile(weight_diff, 99.2))
print(np.percentile(weight_diff, 99.3))
print(np.percentile(weight_diff, 99.4))
print(np.percentile(weight_diff, 99.5))
print(np.percentile(weight_diff, 99.6))
print(np.percentile(weight_diff, 99.7))
print(np.percentile(weight_diff, 99.8))
print(np.percentile(weight_diff, 99.9))
print(np.percentile(weight_diff, 100))
```

```
0.47393346464872643
0.5005209237288906
0.5292974305674735
0.5679671406289156
0.6045056455844512
0.6529999970942804
0.7336527280390962
0.8471311392398194
1.014208921755671
4.242508849548514
```

Printing out features whose % change > threshold {1.0 in my case}

In [34]:
```
new_features = zip(bow_object.get_feature_names(), np.abs(np.divide(w-w1, w1)* 100))
for a, b in new_features:
    if b >= 1:
        print(a, b)
```

```
asin 1.0695381381782767
bhajan 1.2017160292128632
birthright 1.2021857099032134
blend 1.005174271122919
cat 1.7447991799765397
clunk 2.1451604671947733
compass 1.2324366932737782
cook 1.1091498037685488
date 1.6780995045540563
detector 1.0035656135003341
```

```
edif 1.0243160260251083
effect 1.0131831059248635
either 1.0862893580466662
escap 1.5428318116627082
ever 1.046850731809992
expir 1.4030737548283596
felid 1.415722995167102
flavor 1.2200344972102084
food 1.1659317235996816
is 1.1322093482583688
keep 1.0069398680416846
kleenex 1.1614247848718522
mayb 1.359769420530777
messag 1.0104151004478656
miscarriag 1.0425444071328875
not 4.069845302429988
nuh 1.2168187771840164
refund 1.376444745020049
reggiano 1.20193226705168
relax 1.0248919093281355
sensibl 1.3462638963220752
sevencup 1.017555120124254
ship 1.0815174014378486
store 1.1288427903894593
time 1.3141680760187717
tommorow 1.721130038655505
uniqu 1.1238934028842469
unplug 1.2199239384014087
worldfin 2.5812426895706024
```

BoW-L1 -- checking sparcity with different decreased value of C

In [31]:
```
lr = LogisticRegression(C= 10, class_weight='balanced',penalty= 'l1')
lr.fit(xtr,ytr)
ypred = lr.predict(xtest)
print('F1-Score on test set is : {:<.3f}'.format(f1_score(ytest, ypred
)))
```

```
print('Non Zero element count is {}'.format(np.count_nonzero(lr.coef_
)))
```

```
F1-Score on test set is : 0.912
Non Zero element count is 15236
```

```
/home/jalesh_j/.local/lib/python3.5/site-packages/sklearn/svm/base.py:9
31: ConvergenceWarning: Liblinear failed to converge, increase the numb
er of iterations.
  "the number of iterations.", ConvergenceWarning)
```

In [32]:
```
%%time
lr = LogisticRegression(C= 1, class_weight='balanced',penalty= 'l1')
lr.fit(xtr,ytr)
ypred = lr.predict(xtest)
print('F1-Score on test set is : {:<.3f}'.format(f1_score(ytest, ypred
)))
print('Non Zero element count is {}'.format(np.count_nonzero(lr.coef_
)))
```

```
F1-Score on test set is : 0.917
Non Zero element count is 11169
CPU times: user 34 s, sys: 4 ms, total: 34 s
Wall time: 34 s
```

In [33]:
```
%%time
lr = LogisticRegression(C= 0.1, class_weight='balanced',penalty= 'l1')
lr.fit(xtr,ytr)
ypred = lr.predict(xtest)
print('F1-Score on test set is : {:<.3f}'.format(f1_score(ytest, ypred
)))
print('Non Zero element count is {}'.format(np.count_nonzero(lr.coef_
)))
```

```
F1-Score on test set is : 0.929
Non Zero element count is 9779
CPU times: user 2.84 s, sys: 0 ns, total: 2.84 s
Wall time: 2.84 s
```

In [34]:
```
%%time
lr = LogisticRegression(C= 0.01, class_weight='balanced',penalty= 'l1')
lr.fit(xtr,ytr)
ypred = lr.predict(xtest)
print('F1-Score on test set is : {:<.3f}'.format(f1_score(ytest, ypred
)))
print('Non Zero element count is {}'.format(np.count_nonzero(lr.coef_
)))
```

```
F1-Score on test set is : 0.930
Non Zero element count is 4062
CPU times: user 1.37 s, sys: 0 ns, total: 1.37 s
Wall time: 1.37 s
```

LR on BoW(L2)

In [35]:
```
%%time
from sklearn.linear_model import LogisticRegression

auc_cv_dict = {}
auc_tr_dict = {}

c_val = [10 ** -4, 10 ** -3, 10 ** -2, 10 ** -1, 10**1, 10 ** 2, 10 **
3, 10** 4]

for i in c_val:
    lr = LogisticRegression(penalty='l2', C=i, class_weight='balanced',
 random_state=56, n_jobs=-1,max_iter=4, solver='sag')
    lr.fit(xtr, ytr)
    y_pred_cv = lr.predict_proba(xcv)
    fpr_cv, tpr_cv, thresholds_cv = roc_curve(ycv, y_pred_cv[:,1])
    auc_cv_dict[i] = auc(fpr_cv, tpr_cv)
    #performance metrics for training data:
    y_pred_tr = lr.predict_proba(xtr)
    fpr_tr, tpr_tr, thresholds_tr = roc_curve(ytr, y_pred_tr[:,1])
    auc_tr_dict[i] = auc(fpr_tr, tpr_tr)
```

```
/home/jalesh_j/.local/lib/python3.5/site-packages/sklearn/linear_model/
```

```
/home/jalesh_j/.local/lib/python3.5/site-packages/sklearn/linear_model/
sag.py:334: ConvergenceWarning: The max_iter was reached which means th
e coef_ did not converge
  "the coef_ did not converge", ConvergenceWarning)
/home/jalesh_j/.local/lib/python3.5/site-packages/sklearn/linear_model/
sag.py:334: ConvergenceWarning: The max_iter was reached which means th
e coef_ did not converge
  "the coef_ did not converge", ConvergenceWarning)
/home/jalesh_j/.local/lib/python3.5/site-packages/sklearn/linear_model/
sag.py:334: ConvergenceWarning: The max_iter was reached which means th
e coef_ did not converge
  "the coef_ did not converge", ConvergenceWarning)
/home/jalesh_j/.local/lib/python3.5/site-packages/sklearn/linear_model/
sag.py:334: ConvergenceWarning: The max_iter was reached which means th
e coef_ did not converge
  "the coef_ did not converge", ConvergenceWarning)
/home/jalesh_j/.local/lib/python3.5/site-packages/sklearn/linear_model/
sag.py:334: ConvergenceWarning: The max_iter was reached which means th
e coef_ did not converge
  "the coef_ did not converge", ConvergenceWarning)
/home/jalesh_j/.local/lib/python3.5/site-packages/sklearn/linear_model/
sag.py:334: ConvergenceWarning: The max_iter was reached which means th
e coef_ did not converge
  "the coef_ did not converge", ConvergenceWarning)
/home/jalesh_j/.local/lib/python3.5/site-packages/sklearn/linear_model/
sag.py:334: ConvergenceWarning: The max_iter was reached which means th
e coef_ did not converge
  "the coef_ did not converge", ConvergenceWarning)
```

```
CPU times: user 2.17 s, sys: 28 ms, total: 2.2 s
Wall time: 2.69 s
```

```
/home/jalesh_j/.local/lib/python3.5/site-packages/sklearn/linear_model/
sag.py:334: ConvergenceWarning: The max_iter was reached which means th
e coef_ did not converge
  "the coef_ did not converge", ConvergenceWarning)
```

optimal C

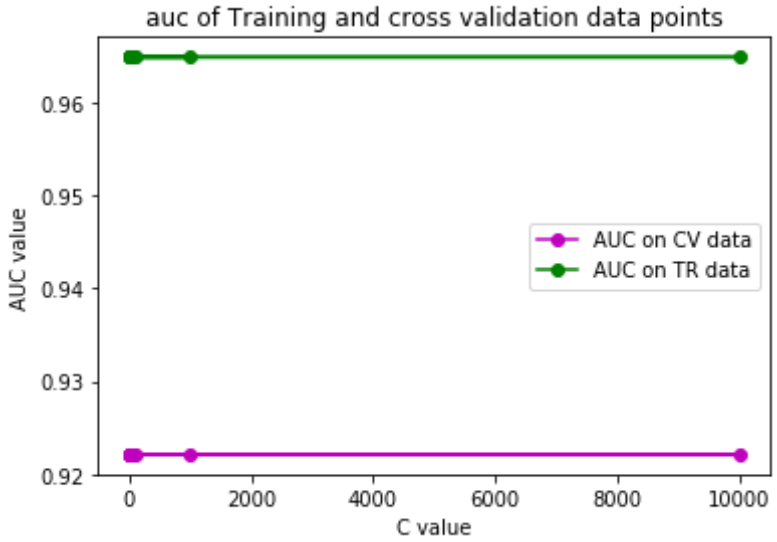
In [36]:
```
#sorting dictionary wrt higest AuC Score of both training and cv data:

cv_tup = sorted(auc_cv_dict.items(), key= lambda x: x[1],reverse=True)
tr_tup = sorted(auc_tr_dict.items(), key= lambda x: x[1],reverse=True)
print(cv_tup)
print('*' * 70)
print(tr_tup)
```

```
[(0.0001, 0.9222189829460161), (0.001, 0.9221585820775322), (0.01, 0.92
21518708699228), (0.1, 0.9221513675293522), (10, 0.9221513339733142),
(10000, 0.9221513004172761), (100, 0.9221513004172761), (1000, 0.922151
3004172761)]
**********************************************************************
[(0.0001, 0.9648523063047845), (0.001, 0.9648224146599895), (0.01, 0.96
48194024525987), (0.1, 0.96481894159437), (10000, 0.9648189130876756),
(100, 0.9648189130876756), (1000, 0.9648189130876756), (10, 0.964818913
0876756)]
```

plotting ROC-AUC on train vs cv data

In [37]:
```
plt.plot([x[0] for x in cv_tup], [x[1] for x in cv_tup], linestyle='-',
 color='m', marker='o',label='AUC on CV data')
plt.plot([x[0] for x in tr_tup], [x[1] for x in tr_tup], linestyle='-',
 color='g', marker='o', label='AUC on TR data')
#plt.plot([0, 1], [0, 1], linestyle='--')
plt.xlabel("C value")
plt.ylabel('AUC value')
plt.title('auc of Training and cross validation data points')
plt.legend()
```

Out[37]:
```
<matplotlib.legend.Legend at 0x7fd148e60e10>
```



Optimal L2-BoW

In [38]:
```
lr = LogisticRegression(penalty='l2', C=0.0001, class_weight='balanced'
, random_state=56, n_jobs=-1,max_iter=4, solver='sag')
lr.fit(xtr, ytr)
y_pred_test = lr.predict_proba(xtest)
y_pred = lr.predict(xtest)
fpr_test, tpr_test, thresholds_test= roc_curve(ytest, y_pred_test[:,1])
auc_test = auc(fpr_test, tpr_test)
print(auc_test)
```

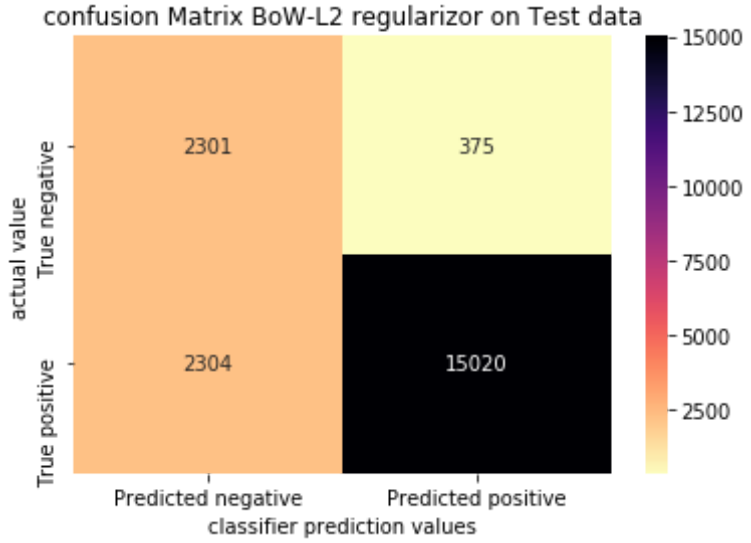
```
0.9263315055985648
```

```
/home/jalesh_j/.local/lib/python3.5/site-packages/sklearn/linear_model/
sag.py:334: ConvergenceWarning: The max_iter was reached which means th
e coef_ did not converge
  "the coef_ did not converge", ConvergenceWarning)
```

confusion matrix on test data

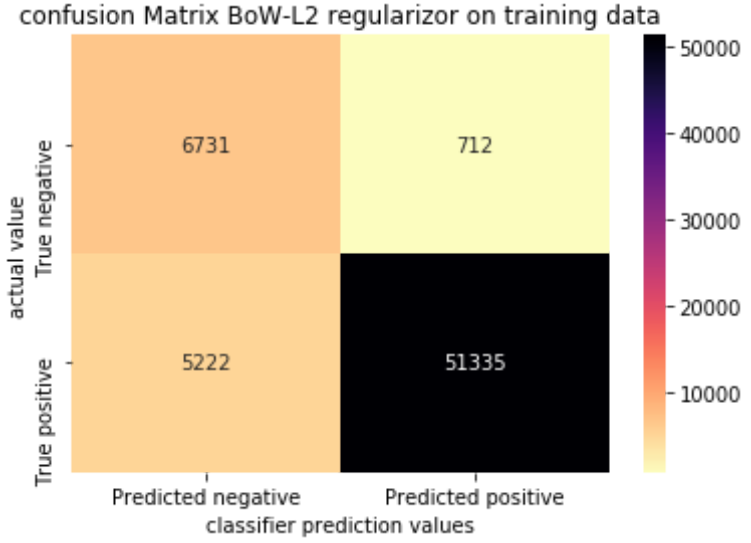
In [39]:
```
%time
y_pred_test = np.where(y_pred_test[:,1] < 0.5, 0, 1)
#creating confusion matrix:

cf = confusion_matrix(ytest, y_pred_test)
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-L2 regularizor on Test data")
plt.xlabel("classifier prediction values")
plt.ylabel("actual value")
plt.show()
```

```
CPU times: user 0 ns, sys: 0 ns, total: 0 ns
Wall time: 5.96 µs
```



confusion matrix on training data

In [40]:
```
y_pred_tr = np.where(y_pred_tr[:,1] > 0.5, 1, 0)
#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-L2 regularizor on training data")
plt.xlabel("classifier prediction values")
plt.ylabel("actual value")
plt.show()
```



ROC-AUC on test vs training 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('ROC of Test vs Training Data')
plt.legend()
```

Out[41]: <matplotlib.legend.Legend at 0x7fd1469750b8>

top 25 features for both positive and negative class

In [42]:
```
features = bow_object.get_feature_names()
featuresAndcoeff = sorted(zip(lr.coef_[0], 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
-------------------------------------------------------------------------
-------------------------------------------------
-0.03636758861347394                      not
  0.043531071634431276                    great
-0.03515925598576446                disappoint
   0.03400131907910012                     love
-0.02250723580278787                    worst
   0.02899878560414292                     best
-0.022285978723451973                   money
   0.022707208514560547                  delici
-0.022130438063942905                     bad
    0.01915329833162593                     good
-0.021732960422216036                  horribl
   0.018795667664374214                 perfect
-0.021493750158529883                       aw
    0.01858567358658857                 favorit
 -0.0214920046296824                   terribl
  0.016133476207883916                     find
-0.020301823657937708                   threw
   0.015939894475238598                  excel
-0.02023304386374258                   return
   0.01582709638524083                     make
-0.019711368134454347                    wast
    0.01572720436624412                     use
-0.018348022292538308                   would
   0.015653898065226365                    nice
-0.017084451172109928                 thought
   0.015558377793843746                  wonder
-0.016753028405749323                  refund
   0.015166434801747203                    easi
 -0.0164771071608918                  unfortun
  0.013860220199422117                    year
-0.01627571562716384                    stale
   0.01381015464716299                   snack
-0.015782117746430617                    didn
    0.01363986752924283                   alway
-0.015057633550327855                   bland
   0.013520932317813051                    high
-0.015004540088593544                    mayb
```

```
   0.013088300078464465                keep
 -0.0148328849976455                 receiv
   0.01242368757475468                  tasti
-0.014454694773901445                   away
   0.012264575158826103                 thank
-0.014107038415686415                product
   0.012253772329065089                 enjoy
-0.013296392255244408              tasteless
    0.01195062468671477                   add
-0.012799010213631628                  wors
   0.011599949103107375                 store
-0.01259499898897743                  sorri
  0.011309026845681087                  also
```

TfIDF(L1)

In [43]:
```
xt, xtest, yt, ytest = train_test_split(tfidf, y, test_size=0.2, shuffl
e=False)
xtr, xcv, ytr, ycv = train_test_split(xt, yt, test_size=0.2, shuffle=Fa
lse)
```

tf-Idf featurizer

In [44]:
```
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)
```

data standardization

In [45]:
```
xtr = sc.fit_transform(xtr)
xcv = sc.transform(xcv)
xtest = sc.transform(xtest)
```

```
print(xtr.shape, ytr.shape)
print(xcv.shape, ycv.shape)
print(xtest.shape, ytest.shape)
```

```
(64000, 45293) (64000,)
(16000, 45293) (16000,)
(20000, 45293) (20000,)
```

Tfidf-BoW

In [46]:

```
%%time

auc_cv_dict = {}
auc_tr_dict = {}

c_val = [10 ** -4, 10 ** -3, 10 ** -2, 10 ** -1, 10**1, 10 ** 2, 10 **
3, 10** 4]

for i in c_val:
    lr = LogisticRegression(penalty='l1', C=i, class_weight='balanced',
 random_state=56, n_jobs=-1,max_iter=4, solver='saga')
    lr.fit(xtr, ytr)
    y_pred_cv = lr.predict_proba(xcv)
    fpr_cv, tpr_cv, thresholds_cv = roc_curve(ycv, y_pred_cv[:,1])
    auc_cv_dict[i] = auc(fpr_cv, tpr_cv)
    #performance metrics for training data:
    y_pred_tr = lr.predict_proba(xtr)
    fpr_tr, tpr_tr, thresholds_tr = roc_curve(ytr, y_pred_tr[:,1])
    auc_tr_dict[i] = auc(fpr_tr, tpr_tr)
```

```
/home/jalesh_j/.local/lib/python3.5/site-packages/sklearn/linear_model/
sag.py:334: ConvergenceWarning: The max_iter was reached which means th
e coef_ did not converge
  "the coef_ did not converge", ConvergenceWarning)
/home/jalesh_j/.local/lib/python3.5/site-packages/sklearn/linear_model/
sag.py:334: ConvergenceWarning: The max_iter was reached which means th
e coef_ did not converge
  "the coef_ did not converge", ConvergenceWarning)
/home/jalesh_j/.local/lib/python3.5/site-packages/sklearn/linear_model/
```

```
sag.py:334: ConvergenceWarning: The max_iter was reached which means th
e coef_ did not converge
  "the coef_ did not converge", ConvergenceWarning)
/home/jalesh_j/.local/lib/python3.5/site-packages/sklearn/linear_model/
sag.py:334: ConvergenceWarning: The max_iter was reached which means th
e coef_ did not converge
  "the coef_ did not converge", ConvergenceWarning)
/home/jalesh_j/.local/lib/python3.5/site-packages/sklearn/linear_model/
sag.py:334: ConvergenceWarning: The max_iter was reached which means th
e coef_ did not converge
  "the coef_ did not converge", ConvergenceWarning)
/home/jalesh_j/.local/lib/python3.5/site-packages/sklearn/linear_model/
sag.py:334: ConvergenceWarning: The max_iter was reached which means th
e coef_ did not converge
  "the coef_ did not converge", ConvergenceWarning)
/home/jalesh_j/.local/lib/python3.5/site-packages/sklearn/linear_model/
sag.py:334: ConvergenceWarning: The max_iter was reached which means th
e coef_ did not converge
  "the coef_ did not converge", ConvergenceWarning)
```

```
CPU times: user 4min 28s, sys: 60 ms, total: 4min 28s
Wall time: 4min 28s
```

```
/home/jalesh_j/.local/lib/python3.5/site-packages/sklearn/linear_model/
sag.py:334: ConvergenceWarning: The max_iter was reached which means th
e coef_ did not converge
  "the coef_ did not converge", ConvergenceWarning)
```

optimal C

In [47]:

```
#sorting dictionary wrt higest AuC Score of both training and cv data:

cv_tup = sorted(auc_cv_dict.items(), key= lambda x: x[1],reverse=True)
tr_tup = sorted(auc_tr_dict.items(), key= lambda x: x[1],reverse=True)
print(cv_tup)
print('*' * 70)
print(tr_tup)
```

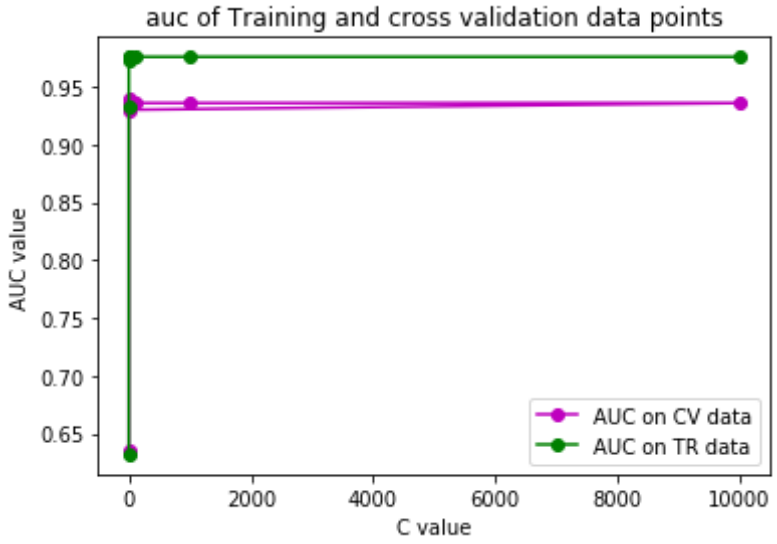
```
[(0.01, 0.939793655868602), (0.1, 0.936522311275473), (10, 0.9360538018
```

```
72266), (100, 0.9360504462684613), (1000, 0.9360499093718523), (10000,
0.9360498758158142), (0.001, 0.9301817636624079), (0.0001, 0.6347100603
954995)]
**********
[(10000, 0.976002704629659), (1000, 0.9760026737474066), (100, 0.976002
4742005446), (10, 0.9760008065589134), (0.1, 0.9757923608572382), (0.0
1, 0.9726462123864053), (0.001, 0.9333498954853118), (0.0001, 0.6319512
282953859)]
```

AUC of train vs CV datapoints

In [48]:
```
plt.plot([x[0] for x in cv_tup], [x[1] for x in cv_tup], linestyle='-',
 color='m', marker='o',label='AUC on CV data')
plt.plot([x[0] for x in tr_tup], [x[1] for x in tr_tup], linestyle='-',
 color='g', marker='o', label='AUC on TR data')
#plt.plot([0, 1], [0, 1], linestyle='--')
plt.xlabel("C value")
plt.ylabel('AUC value')
plt.title('auc of Training and cross validation data points')
plt.legend()
```

Out[48]:
```
<matplotlib.legend.Legend at 0x7fd13364be80>
```



tfidf model on optimal C = 0.01

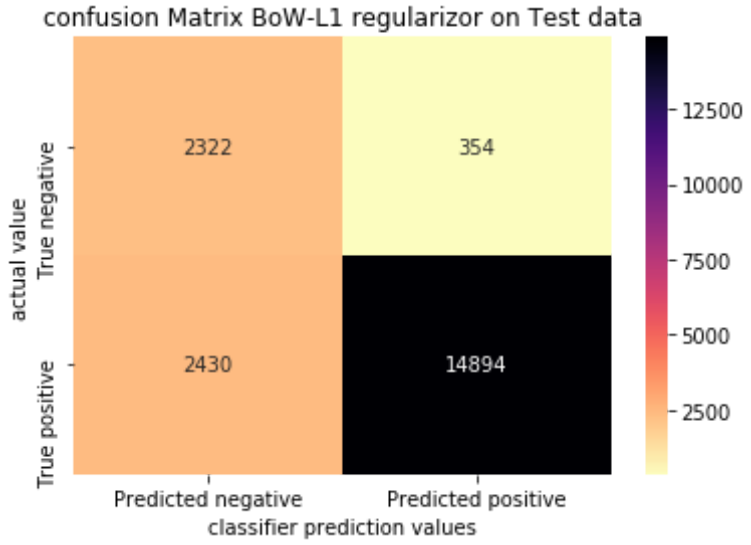
In [49]:
```
%%time

lr = LogisticRegression(penalty='l1', C=0.01, class_weight='balanced',
random_state=56, n_jobs=-1,max_iter=4, solver='saga')
lr.fit(xtr, ytr)
y_pred_test = lr.predict_proba(xtest)
y_pred = lr.predict(xtest)
fpr_test, tpr_test, thresholds_test= roc_curve(ytest, y_pred_test[:,1])
auc_test = auc(fpr_test, tpr_test)
print(auc_test)
```

```
0.9374070299668087
CPU times: user 31.7 s, sys: 84 ms, total: 31.7 s
Wall time: 31.6 s
```

```
/home/jalesh_j/.local/lib/python3.5/site-packages/sklearn/linear_model/
sag.py:334: ConvergenceWarning: The max_iter was reached which means th
e coef_ did not converge
  "the coef_ did not converge", ConvergenceWarning)
```

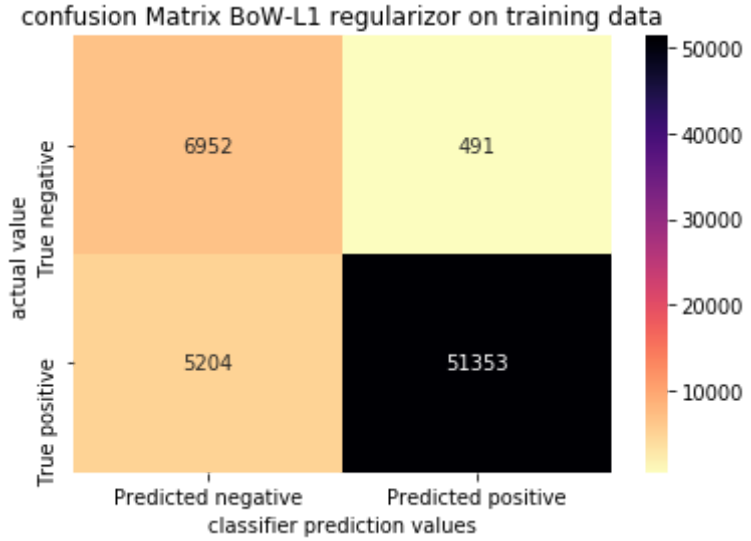
In [50]:
```
%time
y_pred_test = np.where(y_pred_test[:,1] < 0.5, 0, 1)
#creating confusion matrix:

cf = confusion_matrix(ytest, y_pred_test)
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-L1 regularizor on Test data")
plt.xlabel("classifier prediction values")
plt.ylabel("actual value")
plt.show()
```

```
CPU times: user 0 ns, sys: 0 ns, total: 0 ns
Wall time: 5.96 µs
```



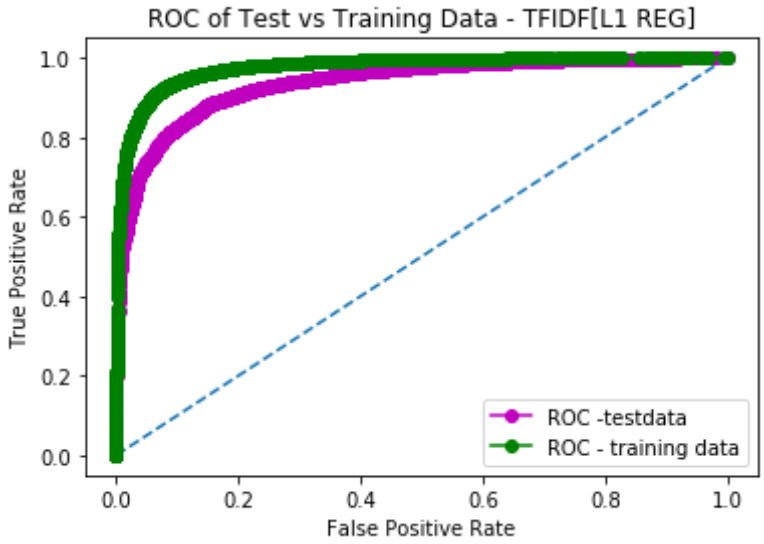
In [51]:
```
y_pred_tr = np.where(y_pred_tr[:,1] > 0.5, 1, 0)
#creating confusion matrix:
```

```
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-L1 regularizor on training data")
plt.xlabel("classifier prediction values")
plt.ylabel("actual value")
plt.show()
```



In [52]:
```
plt.plot(fpr_test, tpr_test, color='m', marker='o',label='ROC -testdat
a')
plt.plot([0, 1], [0, 1], linestyle='--')
plt.plot(fpr_tr, tpr_tr, linestyle='-', color='g', marker='o', label='R
OC - training data')
plt.xlabel('False Positive Rate')
plt.ylabel('True Positive Rate')
plt.title('ROC of Test vs Training Data - TFIDF[L1 REG]')
plt.legend()
```

Out[52]: <matplotlib.legend.Legend at 0x7fd13b455470>



In [53]:
```
features = tfidf_object.get_feature_names()
featuresAndcoeff = sorted(zip(lr.coef_[0], 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
------------------------------------------------------------------------
------------------------------------------------
-0.026316998451721432                  not
   0.024685912449091768                  and
-0.016277134113167013                  was
   0.022330237182991004                great
-0.015639578457239404                worst
```

```
   0.016984941308149067                  is
-0.015299290993678188          disappointed
   0.015026057297874665                  best
-0.01485286094252556              terrible
  0.014748187414641279                   for
-0.014724488056589463                 awful
   0.01335684876421827                  love
-0.014327472210215221              horrible
   0.013046344188651717                    my
-0.01352757901359064                waste
  0.011827215186505827              delicious
-0.013282732861537049                   bad
   0.010979073985741827                  good
-0.013282713467550458                 threw
   0.010936882321652009                    are
-0.013119708790591805                 money
   0.009430736899993256               perfect
-0.01253162277666273                   did
  0.009366663775193573               favorite
-0.012508329656551631                 return
   0.009225621730943684                  with
-0.011727757342320837                 stale
   0.009185215643839822             wonderful
-0.011452438806917493       disappointment
   0.008980959926877113                    in
-0.011241167850519113          disappointing
   0.008856180571410105                highly
-0.010745274439909473                 didn
     0.0087951404960201                loves
-0.01067147615290245                refund
  0.008730969031211528                  find
-0.010547432174682791                 would
   0.008604846395087826                    you
-0.010539002315397419               thought
   0.008405205638374196             excellent
-0.010421000923193521                 maybe
   0.008212293205100654                  easy
-0.010236450179067946                 bland
   0.007828859527665977                  nice
-0.010113336684640081                  away
```

```
   0.007212369931757082                snack
-0.010045006302338247          unfortunately
   0.007152869577531311                always
-0.009886625132302197                were
   0.007126965597516702                   use
```

TFidf-L2 regulrizer

In [54]:
```
%%time
from sklearn.linear_model import LogisticRegression

auc_cv_dict = {}
auc_tr_dict = {}

c_val = [10 ** -4, 10 ** -3, 10 ** -2, 10 ** -1, 10**1, 10 ** 2, 10 **
3, 10** 4]

for i in c_val:
    lr = LogisticRegression(penalty='l2', C=i, class_weight='balanced',
 random_state=56, n_jobs=-1,max_iter=4, solver='sag')
    lr.fit(xtr, ytr)
    y_pred_cv = lr.predict_proba(xcv)
    fpr_cv, tpr_cv, thresholds_cv = roc_curve(ycv, y_pred_cv[:,1])
    auc_cv_dict[i] = auc(fpr_cv, tpr_cv)
    #performance metrics for training data:
    y_pred_tr = lr.predict_proba(xtr)
    fpr_tr, tpr_tr, thresholds_tr = roc_curve(ytr, y_pred_tr[:,1])
    auc_tr_dict[i] = auc(fpr_tr, tpr_tr)
```

```
/home/jalesh_j/.local/lib/python3.5/site-packages/sklearn/linear_model/
sag.py:334: ConvergenceWarning: The max_iter was reached which means th
e coef_ did not converge
  "the coef_ did not converge", ConvergenceWarning)
/home/jalesh_j/.local/lib/python3.5/site-packages/sklearn/linear_model/
sag.py:334: ConvergenceWarning: The max_iter was reached which means th
e coef_ did not converge
  "the coef_ did not converge", ConvergenceWarning)
```

```
/home/jalesh_j/.local/lib/python3.5/site-packages/sklearn/linear_model/
sag.py:334: ConvergenceWarning: The max_iter was reached which means th
e coef_ did not converge
  "the coef_ did not converge", ConvergenceWarning)
/home/jalesh_j/.local/lib/python3.5/site-packages/sklearn/linear_model/
sag.py:334: ConvergenceWarning: The max_iter was reached which means th
e coef_ did not converge
  "the coef_ did not converge", ConvergenceWarning)
/home/jalesh_j/.local/lib/python3.5/site-packages/sklearn/linear_model/
sag.py:334: ConvergenceWarning: The max_iter was reached which means th
e coef_ did not converge
  "the coef_ did not converge", ConvergenceWarning)
/home/jalesh_j/.local/lib/python3.5/site-packages/sklearn/linear_model/
sag.py:334: ConvergenceWarning: The max_iter was reached which means th
e coef_ did not converge
  "the coef_ did not converge", ConvergenceWarning)
/home/jalesh_j/.local/lib/python3.5/site-packages/sklearn/linear_model/
sag.py:334: ConvergenceWarning: The max_iter was reached which means th
e coef_ did not converge
  "the coef_ did not converge", ConvergenceWarning)
```

```
CPU times: user 3.06 s, sys: 28 ms, total: 3.09 s
Wall time: 3.58 s
```

```
/home/jalesh_j/.local/lib/python3.5/site-packages/sklearn/linear_model/
sag.py:334: ConvergenceWarning: The max_iter was reached which means th
e coef_ did not converge
  "the coef_ did not converge", ConvergenceWarning)
```

optimal C

In [55]:

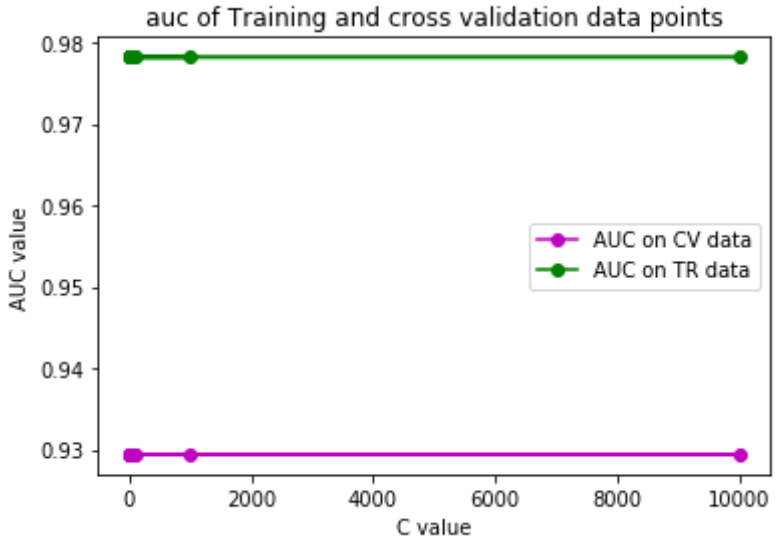
```
#sorting dictionary wrt higest AuC Score of both training and cv data:

cv_tup = sorted(auc_cv_dict.items(), key= lambda x: x[1],reverse=True)
tr_tup = sorted(auc_tr_dict.items(), key= lambda x: x[1],reverse=True)
print(cv_tup)
print('*' * 70)
print(tr_tup)
```

```
[(0.0001, 0.9294887979207069), (0.001, 0.9294252427846466), (0.01, 0.92
94175248958958), (0.1, 0.9294164175466403), (10000, 0.929416182654373
9), (100, 0.9294161826543739), (1000, 0.9294161826543739), (10, 0.92941
61826543739)]
**********
[(0.0001, 0.9782271247655423), (0.001, 0.9781899900447734), (0.01, 0.97
81862794233659), (0.1, 0.9781859397185892), (10000, 0.978185894582989
5), (100, 0.9781858945829895), (1000, 0.9781858945829895), (10, 0.97818
58945829895)]
```

AUC of train vs cv data

In [56]:
```
plt.plot([x[0] for x in cv_tup], [x[1] for x in cv_tup], linestyle='-',
 color='m', marker='o',label='AUC on CV data')
plt.plot([x[0] for x in tr_tup], [x[1] for x in tr_tup], linestyle='-',
 color='g', marker='o', label='AUC on TR data')
#plt.plot([0, 1], [0, 1], linestyle='--')
plt.xlabel("C value")
plt.ylabel('AUC value')
plt.title('auc of Training and cross validation data points')
plt.legend()
```

Out[56]:
```
<matplotlib.legend.Legend at 0x7fd13f1228d0>
```



optimal tfidf(L2)

In [57]:
```
lr = LogisticRegression(penalty='l2', C=0.0001, class_weight='balanced'
, random_state=56, n_jobs=-1,max_iter=4, solver='sag')
lr.fit(xtr, ytr)
y_pred_test = lr.predict_proba(xtest)
y_pred = lr.predict(xtest)
fpr_test, tpr_test, thresholds_test= roc_curve(ytest, y_pred_test[:,1])
auc_test = auc(fpr_test, tpr_test)
print(auc_test)
```

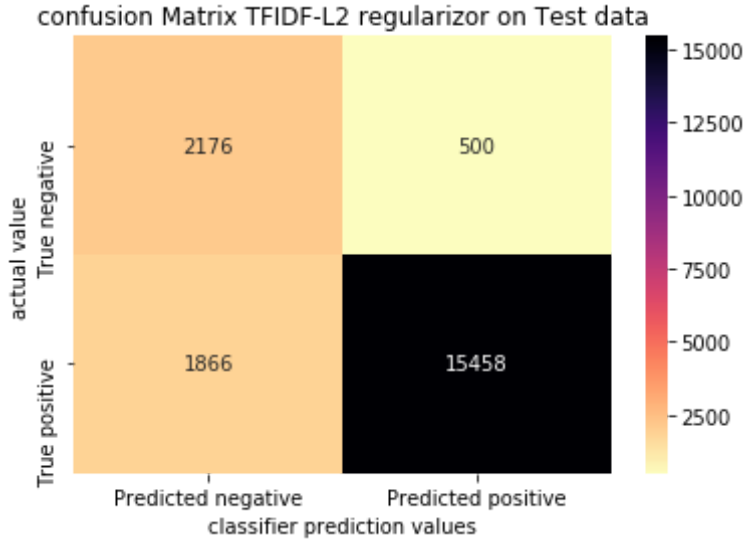
```
0.9288325397014399
```

```
/home/jalesh_j/.local/lib/python3.5/site-packages/sklearn/linear_model/
sag.py:334: ConvergenceWarning: The max_iter was reached which means th
e coef_ did not converge
  "the coef_ did not converge", ConvergenceWarning)
```

confusion matrix on test data

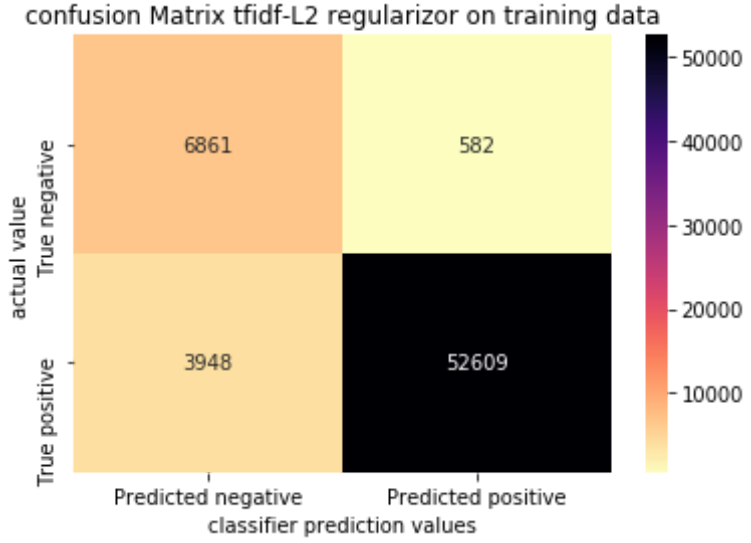
In [58]:
```
%time
y_pred_test = np.where(y_pred_test[:,1] < 0.5, 0, 1)
#creating confusion matrix:

cf = confusion_matrix(ytest, y_pred_test)
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-L2 regularizor on Test data")
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
```



confusion matrix on train data

In [59]:
```
y_pred_tr = np.where(y_pred_tr[:,1] > 0.5, 1, 0)
#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-L2 regularizor on training data")
plt.xlabel("classifier prediction values")
plt.ylabel("actual value")
plt.show()
```

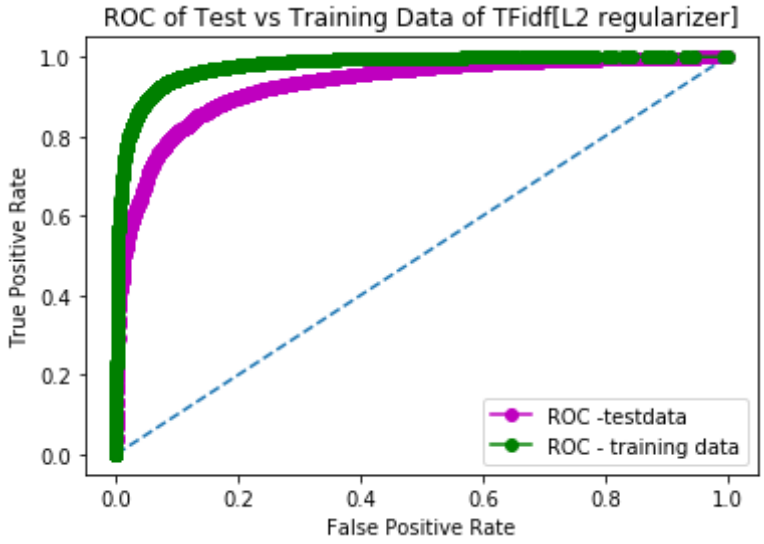


ROC of train vs test data

In [60]:
```
plt.plot(fpr_test, tpr_test, color='m', marker='o',label='ROC -testdat
a')
plt.plot([0, 1], [0, 1], linestyle='--')
plt.plot(fpr_tr, tpr_tr, linestyle='-', color='g', marker='o', label='R
```

```
OC - training data')
plt.xlabel('False Positive Rate')
plt.ylabel('True Positive Rate')
plt.title('ROC of Test vs Training Data of TFidf[L2 regularizer]')
plt.legend()
```

Out[60]: <matplotlib.legend.Legend at 0x7fd146db7ef0>



top 25 positive and negative reviews

In [61]:
```
features = tfidf_object.get_feature_names()
featuresAndcoeff = sorted(zip(lr.coef_[0], 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
-------------------------------------------------------------------------
-------------------------------------------------
  -0.030586216068246                      not
  0.029215636799053597                    great
-0.023303750268383928                   worst
   0.025919424212112435                     and
-0.02271190079113811              disappointed
  0.021369708002574664                     best
-0.02222171328634551                  terrible
  0.019379759234883866                      is
-0.02204217089522468                    awful
   0.01876832356616267                     love
-0.021765058044845784                 horrible
    0.01769458919240648                delicious
-0.02022203096979263                    waste
  0.017586523470395025                      for
-0.019974131605316477                   threw
    0.01640021917512217                     my
-0.019331407182338403                     bad
    0.01492005935336908                    good
-0.01920175616827927                    money
  0.014407798570096721                      are
-0.019160678746868994                     was
   0.014342948854815544                 perfect
-0.01866251017749516                    return
  0.014154637508417396                  favorite
-0.018029897360677664                     did
    0.01405125595643186                wonderful
-0.017854276988693404                     stale
   0.013568031888898241                    highly
-0.017413655246236337           disappointment
    0.01353818499846033                   loves
-0.016929105787569596            disappointing
   0.012958142575059228                excellent
-0.015936246755006402                   refund
    0.01289675656000294                    find
-0.015856578392937984                     didn
    0.01252013408410544                    easy
-0.01567757709867981                    maybe
```

```
  0.012388043721818814                with
-0.01549468334832811                bland
  0.011965425168044086                 nice
-0.015463240916173251              thought
   0.011615500051856166                  in
-0.015325973446231324         unfortunately
   0.011604074669511605                  you
-0.014880722450283127                away
   0.011121972916887086                snack
-0.01404907405530725             tasteless
   0.01102446406782148                always
-0.013973353165231954                would
   0.010752454462403754                makes
```

Avg-W2V --L1

In [6]:
```
#train, cv, test split:
print(type(w2v))
xtrain, xtest, ytrain, ytest = train_test_split(w2v, y, test_size=0.2,
shuffle=False)
xtr, xcv, ytr, ycv = train_test_split(xtrain, ytrain, test_size=0.2, sh
uffle=False)
```

```
<class 'pandas.core.series.Series'>
```

list of lists of train, cv, test data

In [7]:
```
%%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 1.7 s, sys: 316 ms, total: 2.02 s
Wall time: 2.02 s
```

instantiating word2vec object for Train, cv, test data

In [8]:

```
%%time

from gensim.models import Word2Vec

#instantiating training,cv, test word to vector object:
trainw2v = Word2Vec(train_list, size=1000, workers=8)
cvw2v = Word2Vec(cv_list, size=1000,workers=8)
testw2v = Word2Vec(test_list, size=1000, workers=8)

#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 5min 29s, sys: 1.62 s, total: 5min 31s
Wall time: 48.4 s
```

Avg-W2V for train, cv, test data

In [70]:
```
%%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 40min 24s, sys: 6.15 s, total: 40min 30s
Wall time: 40min 20s
```

column standardization

In [71]:
```
sc = StandardScaler(with_mean=False)
xtr = sc.fit_transform(train_vector)
xcv = sc.transform(cv_vector)
xtest = sc.transform(test_vector)
```

L1 regularizer

In [72]:

```
%%time

auc_cv_dict = {}
auc_tr_dict = {}

c_val = [10 ** -4, 10 ** -3, 10 ** -2, 10 ** -1, 10**1, 10 ** 2, 10 **
3, 10** 4]

for i in c_val:
    lr = LogisticRegression(penalty='l1', C=i, class_weight='balanced',
 random_state=56, n_jobs=-1,max_iter=4, solver='saga')
    lr.fit(xtr, ytr)
    y_pred_cv = lr.predict_proba(xcv)
    fpr_cv, tpr_cv, thresholds_cv = roc_curve(ycv, y_pred_cv[:,1])
    auc_cv_dict[i] = auc(fpr_cv, tpr_cv)
    #performance metrics for training data:
    y_pred_tr = lr.predict_proba(xtr)
    fpr_tr, tpr_tr, thresholds_tr = roc_curve(ytr, y_pred_tr[:,1])
    auc_tr_dict[i] = auc(fpr_tr, tpr_tr)
```

```
/home/jalesh_j/.local/lib/python3.5/site-packages/sklearn/linear_model/
sag.py:334: ConvergenceWarning: The max_iter was reached which means th
e coef_ did not converge
  "the coef_ did not converge", ConvergenceWarning)
/home/jalesh_j/.local/lib/python3.5/site-packages/sklearn/linear_model/
sag.py:334: ConvergenceWarning: The max_iter was reached which means th
e coef_ did not converge
  "the coef_ did not converge", ConvergenceWarning)
/home/jalesh_j/.local/lib/python3.5/site-packages/sklearn/linear_model/
sag.py:334: ConvergenceWarning: The max_iter was reached which means th
e coef_ did not converge
  "the coef_ did not converge", ConvergenceWarning)
/home/jalesh_j/.local/lib/python3.5/site-packages/sklearn/linear_model/
sag.py:334: ConvergenceWarning: The max_iter was reached which means th
e coef_ did not converge
  "the coef_ did not converge", ConvergenceWarning)
/home/jalesh_j/.local/lib/python3.5/site-packages/sklearn/linear_model/
sag.py:334: ConvergenceWarning: The max_iter was reached which means th
e coef_ did not converge
```

```
  "the coef_ did not converge", ConvergenceWarning)
/home/jalesh_j/.local/lib/python3.5/site-packages/sklearn/linear_model/
sag.py:334: ConvergenceWarning: The max_iter was reached which means th
e coef_ did not converge
  "the coef_ did not converge", ConvergenceWarning)
/home/jalesh_j/.local/lib/python3.5/site-packages/sklearn/linear_model/
sag.py:334: ConvergenceWarning: The max_iter was reached which means th
e coef_ did not converge
  "the coef_ did not converge", ConvergenceWarning)
```

```
CPU times: user 52.2 s, sys: 10.2 s, total: 1min 2s
Wall time: 47.1 s
```

```
/home/jalesh_j/.local/lib/python3.5/site-packages/sklearn/linear_model/
sag.py:334: ConvergenceWarning: The max_iter was reached which means th
e coef_ did not converge
  "the coef_ did not converge", ConvergenceWarning)
```

optimal C value for AUC score on training and CV data

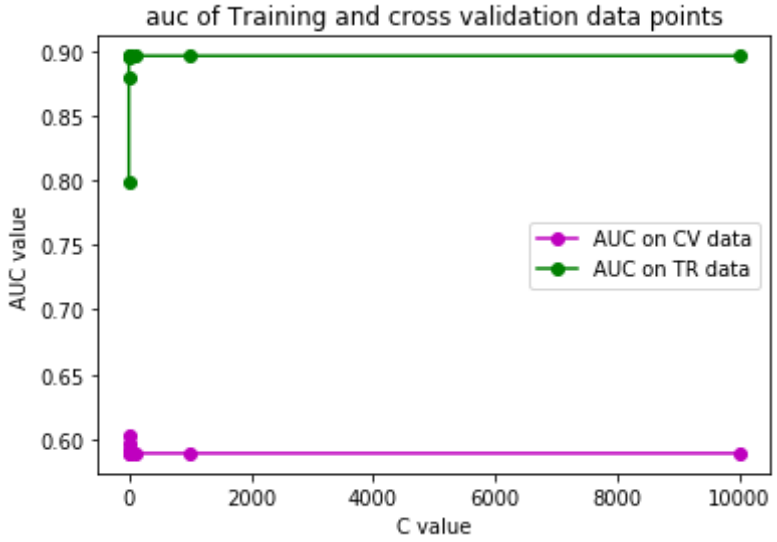
In [73]:
```
#sorting dictionary wrt higest AuC Score of both training and cv data:

cv_tup = sorted(auc_cv_dict.items(), key= lambda x: x[1],reverse=True)
tr_tup = sorted(auc_tr_dict.items(), key= lambda x: x[1],reverse=True)
print(cv_tup)
print('*' * 70)
print(tr_tup)
```

```
[(0.001, 0.6034703788771988), (0.0001, 0.596988224783577), (0.01, 0.591
7525432792357), (0.1, 0.58922463270903), (10, 0.5890893347636259), (10
0, 0.5890884958626748), (1000, 0.5890882945264465), (10000, 0.589088294
5264464)]
**********************************************************************
[(10000, 0.8960481516174921), (1000, 0.8960481516174921), (100, 0.89604
81468663763), (10, 0.8960474341990126), (0.1, 0.8959720363674821), (0.0
1, 0.8946331363133524), (0.001, 0.8789666124153388), (0.0001, 0.7993185
752607772)]
```

Plotting AUC Curve on training and cv data

In [74]:
```
plt.plot([x[0] for x in cv_tup], [x[1] for x in cv_tup], linestyle='-',
 color='m', marker='o',label='AUC on CV data')
plt.plot([x[0] for x in tr_tup], [x[1] for x in tr_tup], linestyle='-',
 color='g', marker='o', label='AUC on TR data')
#plt.plot([0, 1], [0, 1], linestyle='--')
plt.xlabel("C value")
plt.ylabel('AUC value')
plt.title('auc of Training and cross validation data points')
plt.legend()
```

Out[74]:
```
<matplotlib.legend.Legend at 0x7fd0e6e9e198>
```



optimal LR-avg-W2V(L1) for C=0.001

In [75]:
```
lr = LogisticRegression(penalty='l1', C=0.001, class_weight='balanced',
 random_state=56, n_jobs=-1,max_iter=4, solver='saga')
lr.fit(xtr, ytr)
y_pred_test = lr.predict_proba(xtest)
y_pred = lr.predict(xtest)
```

```
fpr_test, tpr_test, thresholds_test= roc_curve(ytest, y_pred_test[:,1])
auc_test = auc(fpr_test, tpr_test)
optimal_weight = lr.coef_
print(auc_test)
```

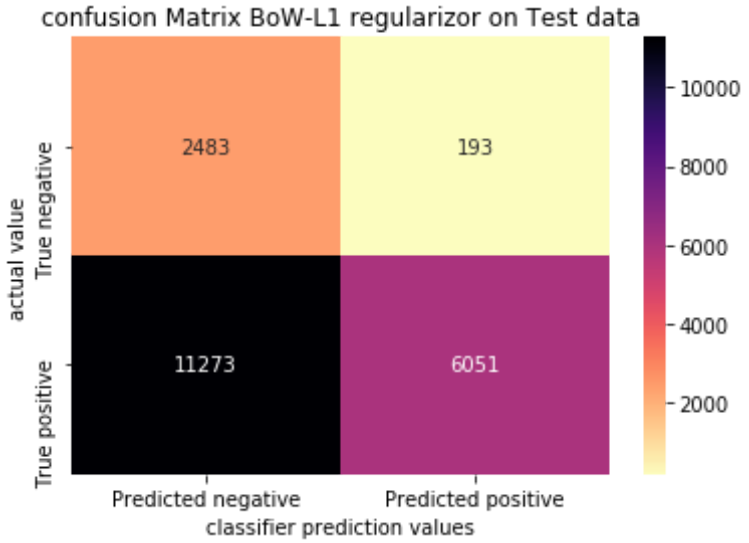
```
0.7309565231571743
```

```
/home/jalesh_j/.local/lib/python3.5/site-packages/sklearn/linear_model/
sag.py:334: ConvergenceWarning: The max_iter was reached which means th
e coef_ did not converge
  "the coef_ did not converge", ConvergenceWarning)
```

plotting confusion matrix on test data

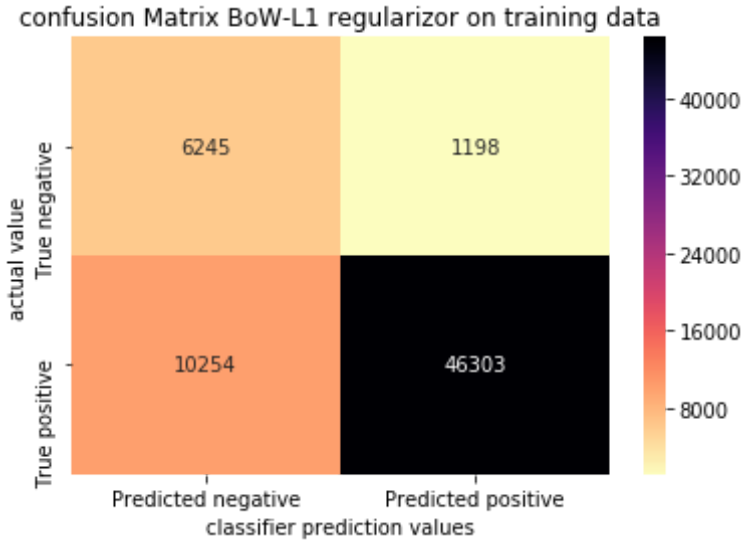
In [76]:
```
%time
y_pred_test = np.where(y_pred_test[:,1] < 0.5, 0, 1)
#creating confusion matrix:

cf = confusion_matrix(ytest, y_pred_test)
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-L1 regularizor on Test data")
plt.xlabel("classifier prediction values")
plt.ylabel("actual value")
plt.show()
```

```
CPU times: user 0 ns, sys: 0 ns, total: 0 ns
Wall time: 8.11 µs
```



plotting confusion matrix on train data

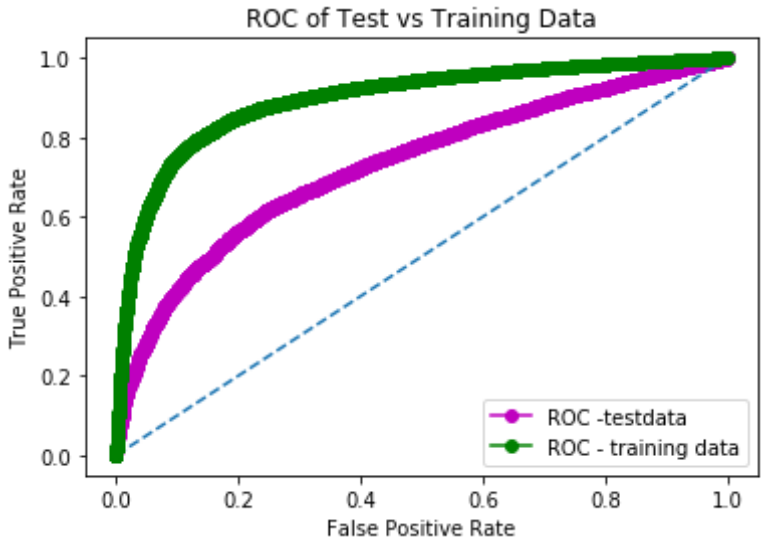
In [77]:
```
y_pred_tr = np.where(y_pred_tr[:,1] > 0.5, 1, 0)
#creating confusion matrix:

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-L1 regularizor on training data")
plt.xlabel("classifier prediction values")
plt.ylabel("actual value")
plt.show()
```



plotting ROC curve on test vs training data

In [78]:
```
plt.plot(fpr_test, tpr_test, color='m', marker='o',label='ROC -testdat
a')
plt.plot([0, 1], [0, 1], linestyle='--')
plt.plot(fpr_tr, tpr_tr, linestyle='-', color='g', marker='o', label='R
OC - training data')
plt.xlabel('False Positive Rate')
plt.ylabel('True Positive Rate')
plt.title('ROC of Test vs Training Data')
plt.legend()
```

Out[78]:
```
<matplotlib.legend.Legend at 0x7fd0e65e3860>
```



printing top 25 features of both positive and negative class

In [ ]:
```
features = bow_object.get_feature_names()
featuresAndcoeff = sorted(zip(lr.coef_[0], 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))
```

Avg-W2v L2 regularizer

In [79]:
```
%%time
from sklearn.linear_model import LogisticRegression
```

```
auc_cv_dict = {}
auc_tr_dict = {}

c_val = [10 ** -4, 10 ** -3, 10 ** -2, 10 ** -1, 10**1, 10 ** 2, 10 **
3, 10** 4]

for i in c_val:
    lr = LogisticRegression(penalty='l2', C=i, class_weight='balanced',
 random_state=56, n_jobs=-1,max_iter=4, solver='sag')
    lr.fit(xtr, ytr)
    y_pred_cv = lr.predict_proba(xcv)
    fpr_cv, tpr_cv, thresholds_cv = roc_curve(ycv, y_pred_cv[:,1])
    auc_cv_dict[i] = auc(fpr_cv, tpr_cv)
    #performance metrics for training data:
    y_pred_tr = lr.predict_proba(xtr)
    fpr_tr, tpr_tr, thresholds_tr = roc_curve(ytr, y_pred_tr[:,1])
    auc_tr_dict[i] = auc(fpr_tr, tpr_tr)
```

```
/home/jalesh_j/.local/lib/python3.5/site-packages/sklearn/linear_model/
sag.py:334: ConvergenceWarning: The max_iter was reached which means th
e coef_ did not converge
  "the coef_ did not converge", ConvergenceWarning)
/home/jalesh_j/.local/lib/python3.5/site-packages/sklearn/linear_model/
sag.py:334: ConvergenceWarning: The max_iter was reached which means th
e coef_ did not converge
  "the coef_ did not converge", ConvergenceWarning)
/home/jalesh_j/.local/lib/python3.5/site-packages/sklearn/linear_model/
sag.py:334: ConvergenceWarning: The max_iter was reached which means th
e coef_ did not converge
  "the coef_ did not converge", ConvergenceWarning)
/home/jalesh_j/.local/lib/python3.5/site-packages/sklearn/linear_model/
sag.py:334: ConvergenceWarning: The max_iter was reached which means th
e coef_ did not converge
  "the coef_ did not converge", ConvergenceWarning)
/home/jalesh_j/.local/lib/python3.5/site-packages/sklearn/linear_model/
sag.py:334: ConvergenceWarning: The max_iter was reached which means th
e coef_ did not converge
  "the coef_ did not converge", ConvergenceWarning)
/home/jalesh_j/.local/lib/python3.5/site-packages/sklearn/linear_model/
```

```
sag.py:334: ConvergenceWarning: The max_iter was reached which means th
e coef_ did not converge
  "the coef_ did not converge", ConvergenceWarning)
/home/jalesh_j/.local/lib/python3.5/site-packages/sklearn/linear_model/
sag.py:334: ConvergenceWarning: The max_iter was reached which means th
e coef_ did not converge
  "the coef_ did not converge", ConvergenceWarning)
```

```
CPU times: user 24.8 s, sys: 9.92 s, total: 34.7 s
Wall time: 19.3 s
```

```
/home/jalesh_j/.local/lib/python3.5/site-packages/sklearn/linear_model/
sag.py:334: ConvergenceWarning: The max_iter was reached which means th
e coef_ did not converge
  "the coef_ did not converge", ConvergenceWarning)
```

optimal C

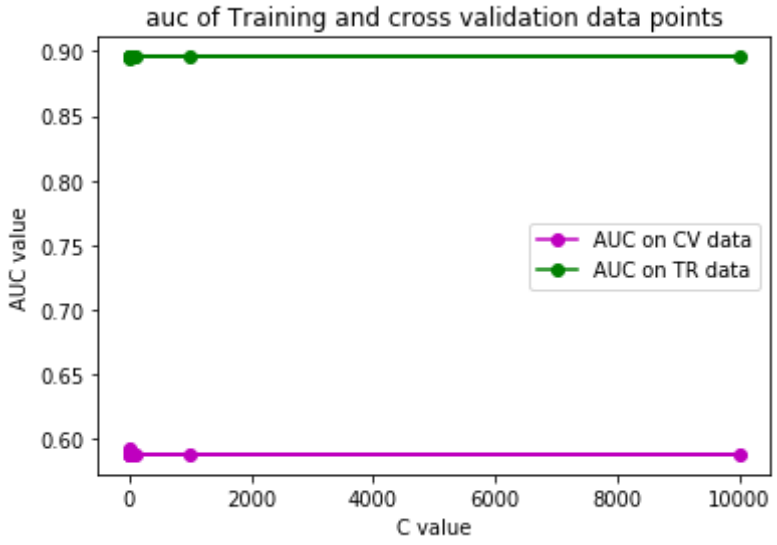
In [80]:
```
#sorting dictionary wrt higest AuC Score of both training and cv data:

cv_tup = sorted(auc_cv_dict.items(), key= lambda x: x[1],reverse=True)
tr_tup = sorted(auc_tr_dict.items(), key= lambda x: x[1],reverse=True)
print(cv_tup)
print('*' * 70)
print(tr_tup)
```

```
[(0.0001, 0.5918883781212484), (0.001, 0.5883229484106787), (0.01, 0.58
79353090591639), (0.1, 0.5878969209516385), (10, 0.5878938337961382),
(10000, 0.5878938002401002), (100, 0.5878938002401002), (1000, 0.587893
8002401002)]
**********************************************************************
[(100, 0.8956667177910478), (10000, 0.8956667154154899), (1000, 0.89566
67154154899), (10, 0.8956667035377006), (0.1, 0.8956661167749044), (0.0
1, 0.8956603655492786), (0.001, 0.8955974275188251), (0.0001, 0.8945316
002184764)]
```

AUC of train vs cv data

In [81]:
```
plt.plot([x[0] for x in cv_tup], [x[1] for x in cv_tup], linestyle='-',
 color='m', marker='o',label='AUC on CV data')
plt.plot([x[0] for x in tr_tup], [x[1] for x in tr_tup], linestyle='-',
 color='g', marker='o', label='AUC on TR data')
#plt.plot([0, 1], [0, 1], linestyle='--')
plt.xlabel("C value")
plt.ylabel('AUC value')
plt.title('auc of Training and cross validation data points')
plt.legend()
```

Out[81]: <matplotlib.legend.Legend at 0x7fd0e6596d30>



optimal avg-W2V(L2)

In [82]:
```
lr = LogisticRegression(penalty='l2', C=0.0001, class_weight='balanced'
, random_state=56, n_jobs=-1,max_iter=4, solver='sag')
lr.fit(xtr, ytr)
y_pred_test = lr.predict_proba(xtest)
y_pred = lr.predict(xtest)
fpr_test, tpr_test, thresholds_test= roc_curve(ytest, y_pred_test[:,1])
```

```
auc_test = auc(fpr_test, tpr_test)
print(auc_test)
```

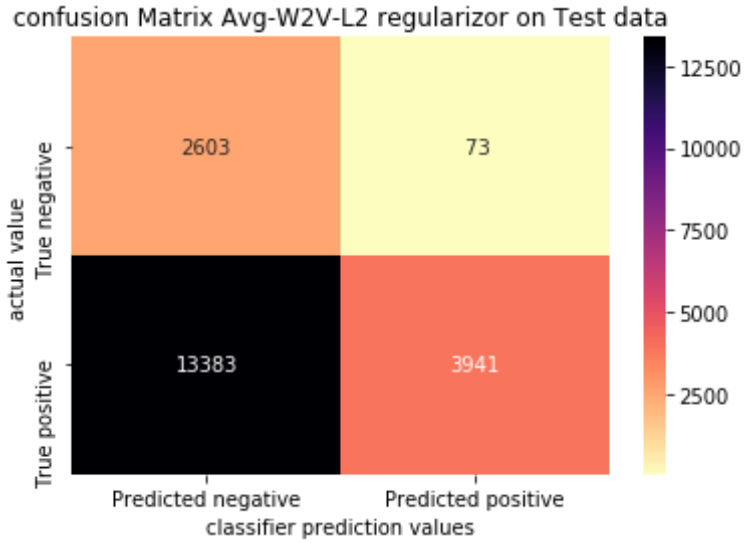
```
0.7076581896978676
```

```
/home/jalesh_j/.local/lib/python3.5/site-packages/sklearn/linear_model/
sag.py:334: ConvergenceWarning: The max_iter was reached which means th
e coef_ did not converge
  "the coef_ did not converge", ConvergenceWarning)
```

confusion matrix on test data

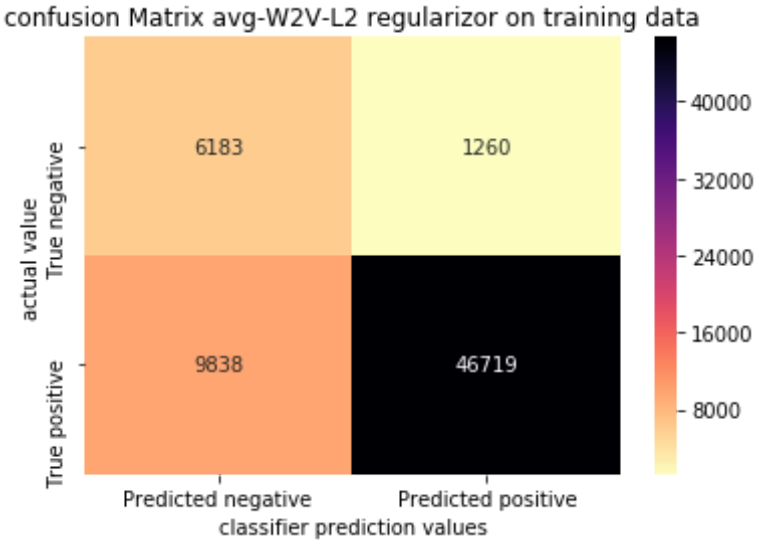
In [83]:
```
%time
y_pred_test = np.where(y_pred_test[:,1] < 0.5, 0, 1)
#creating confusion matrix:

cf = confusion_matrix(ytest, y_pred_test)
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-L2 regularizor on Test data")
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 µs
```



confusion matrix on train data

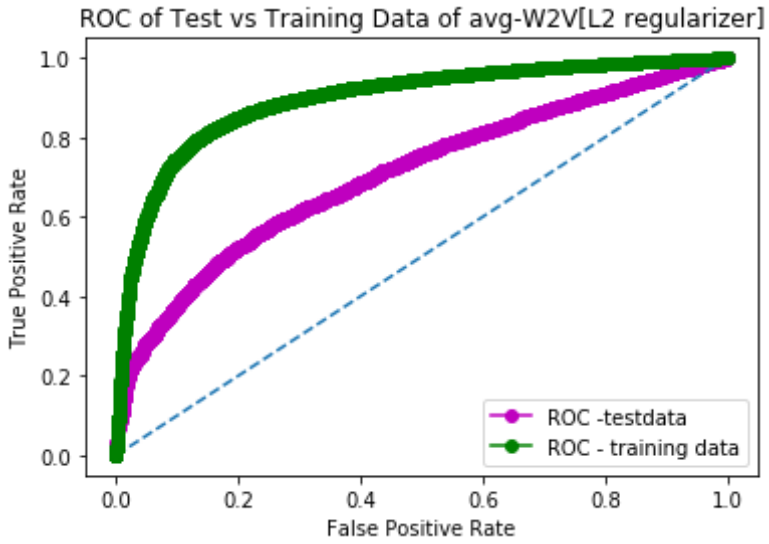
In [84]:
```
y_pred_tr = np.where(y_pred_tr[:,1] > 0.5, 1, 0)
#creating confusion matrix:

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-L2 regularizor on training data")
plt.xlabel("classifier prediction values")
plt.ylabel("actual value")
plt.show()
```



ROC of train vs test data

In [85]:
```
plt.plot(fpr_test, tpr_test, color='m', marker='o',label='ROC -testdat
a')
plt.plot([0, 1], [0, 1], linestyle='--')
plt.plot(fpr_tr, tpr_tr, linestyle='-', color='g', marker='o', label='R
OC - training data')
plt.xlabel('False Positive Rate')
plt.ylabel('True Positive Rate')
plt.title('ROC of Test vs Training Data of avg-W2V[L2 regularizer]')
plt.legend()
```

Out[85]:
```
<matplotlib.legend.Legend at 0x7fd0e6f69f28>
```



Feature enginnering on AVG-W2V

In [86]:
```
type(w2v_featured)
```

Out[86]:
```
pandas.core.series.Series
```

In [87]:
```
#train, cv, test split:
print(type(w2v))
xtrain, xtest, ytrain, ytest = train_test_split(w2v_featured, y, test_s
ize=0.2, shuffle=False)
xtr, xcv, ytr, ycv = train_test_split(xtrain, ytrain, test_size=0.2, sh
uffle=False)
```

```
<class 'pandas.core.series.Series'>
```

list of lists of train, cv, test data

In [88]:
```
%%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 3.26 s, sys: 72 ms, total: 3.34 s
Wall time: 3.33 s
```

instantiating word2vec object for Train, cv, test data

In [90]:
```
%%time

from gensim.models import Word2Vec

#instantiating training,cv, test word to vector object:
trainw2v = Word2Vec(train_list, size=1000, workers=8)
cvw2v = Word2Vec(cv_list, size=1000,workers=8)
```

```
testw2v = Word2Vec(test_list, size=1000, workers=8)

#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 4min 41s, sys: 1.53 s, total: 4min 43s
Wall time: 41.9 s
```

Avg-W2V for train, cv, test data

In [91]:
```
%%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 47min 29s, sys: 7.46 s, total: 47min 37s
Wall time: 47min 27s
```

column standardization

In [92]:
```
sc = StandardScaler(with_mean=False)
xtr = sc.fit_transform(train_vector)
xcv = sc.transform(cv_vector)
xtest = sc.transform(test_vector)
```

Avg-W2V[L1] regularizer on featurized text

In [93]:
```
%%time

auc_cv_dict = {}
auc_tr_dict = {}

c_val = [10 ** -4, 10 ** -3, 10 ** -2, 10 ** -1, 10**1, 10 ** 2, 10 **
3, 10** 4]

for i in c_val:
    lr = LogisticRegression(penalty='l1', C=i, class_weight='balanced',
 random_state=56, n_jobs=-1,max_iter=4, solver='saga')
    lr.fit(xtr, ytr)
    y_pred_cv = lr.predict_proba(xcv)
    fpr_cv, tpr_cv, thresholds_cv = roc_curve(ycv, y_pred_cv[:,1])
    auc_cv_dict[i] = auc(fpr_cv, tpr_cv)
    #performance metrics for training data:
    y_pred_tr = lr.predict_proba(xtr)
    fpr_tr, tpr_tr, thresholds_tr = roc_curve(ytr, y_pred_tr[:,1])
    auc_tr_dict[i] = auc(fpr_tr, tpr_tr)
```

```
/home/jalesh_j/.local/lib/python3.5/site-packages/sklearn/linear_model/
sag.py:334: ConvergenceWarning: The max_iter was reached which means th
e coef_ did not converge
  "the coef_ did not converge", ConvergenceWarning)
/home/jalesh_j/.local/lib/python3.5/site-packages/sklearn/linear_model/
sag.py:334: ConvergenceWarning: The max_iter was reached which means th
e coef_ did not converge
  "the coef_ did not converge", ConvergenceWarning)
/home/jalesh_j/.local/lib/python3.5/site-packages/sklearn/linear_model/
sag.py:334: ConvergenceWarning: The max_iter was reached which means th
e coef_ did not converge
```

```
  "the coef_ did not converge", ConvergenceWarning)
/home/jalesh_j/.local/lib/python3.5/site-packages/sklearn/linear_model/
sag.py:334: ConvergenceWarning: The max_iter was reached which means th
e coef_ did not converge
  "the coef_ did not converge", ConvergenceWarning)
/home/jalesh_j/.local/lib/python3.5/site-packages/sklearn/linear_model/
sag.py:334: ConvergenceWarning: The max_iter was reached which means th
e coef_ did not converge
  "the coef_ did not converge", ConvergenceWarning)
/home/jalesh_j/.local/lib/python3.5/site-packages/sklearn/linear_model/
sag.py:334: ConvergenceWarning: The max_iter was reached which means th
e coef_ did not converge
  "the coef_ did not converge", ConvergenceWarning)
/home/jalesh_j/.local/lib/python3.5/site-packages/sklearn/linear_model/
sag.py:334: ConvergenceWarning: The max_iter was reached which means th
e coef_ did not converge
  "the coef_ did not converge", ConvergenceWarning)
/home/jalesh_j/.local/lib/python3.5/site-packages/sklearn/linear_model/
sag.py:334: ConvergenceWarning: The max_iter was reached which means th
e coef_ did not converge
  "the coef_ did not converge", ConvergenceWarning)
```

```
CPU times: user 52.8 s, sys: 10.1 s, total: 1min 2s
Wall time: 47.6 s
```

optimal C

In [94]:

```
#sorting dictionary wrt higest AuC Score of both training and cv data:

cv_tup = sorted(auc_cv_dict.items(), key= lambda x: x[1],reverse=True)
tr_tup = sorted(auc_tr_dict.items(), key= lambda x: x[1],reverse=True)
print(cv_tup)
print('*' * 70)
print(tr_tup)
```

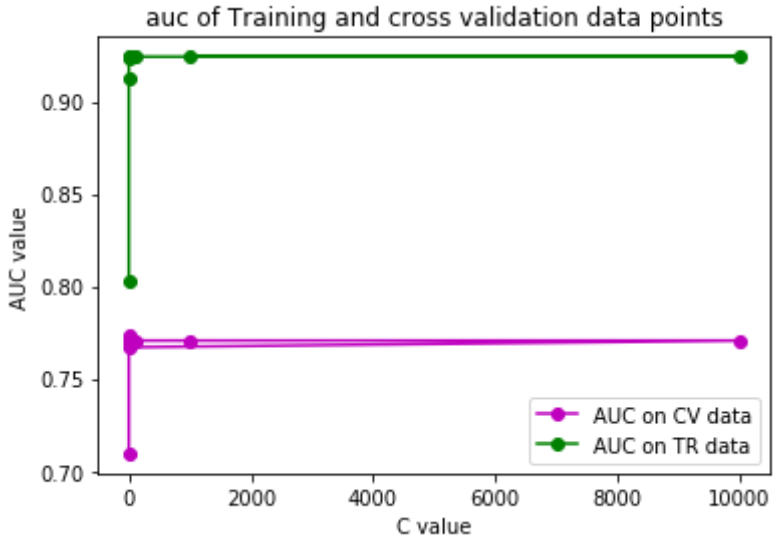
```
[(0.01, 0.7734136584349679), (0.1, 0.7707660199210118), (10, 0.77059985
04206049), (100, 0.7705976357220937), (1000, 0.7705973672737894), (1000
0, 0.7705972666056752), (0.001, 0.7669821739587965), (0.0001, 0.7094977
```

```
614095898)]
**********************************************************************
[(1000, 0.9241721509211591), (10000, 0.924172150921159), (100, 0.924172
072527749), (10, 0.9241716924384882), (0.1, 0.924121045544502), (0.01,
0.9232553732488299), (0.001, 0.9121025993660762), (0.0001, 0.8024658972
572025)]
```

AUC of train vs cv data

In [95]:
```
plt.plot([x[0] for x in cv_tup], [x[1] for x in cv_tup], linestyle='-',
 color='m', marker='o',label='AUC on CV data')
plt.plot([x[0] for x in tr_tup], [x[1] for x in tr_tup], linestyle='-',
 color='g', marker='o', label='AUC on TR data')
#plt.plot([0, 1], [0, 1], linestyle='--')
plt.xlabel("C value")
plt.ylabel('AUC value')
plt.title('auc of Training and cross validation data points')
plt.legend()
```

Out[95]:
```
<matplotlib.legend.Legend at 0x7fcd1573f400>
```



optimal avg-W2V(L1) for C = 0.01

In [97]:
```
%%time

lr = LogisticRegression(penalty='l1', C=0.01, class_weight='balanced',
random_state=56, n_jobs=-1,max_iter=4, solver='saga')
lr.fit(xtr, ytr)
y_pred_test = lr.predict_proba(xtest)
y_pred = lr.predict(xtest)
fpr_test, tpr_test, thresholds_test= roc_curve(ytest, y_pred_test[:,1])
auc_test = auc(fpr_test, tpr_test)
print(auc_test)
```

```
0.7485325618589382
CPU times: user 6.51 s, sys: 200 ms, total: 6.71 s
Wall time: 6.43 s
```

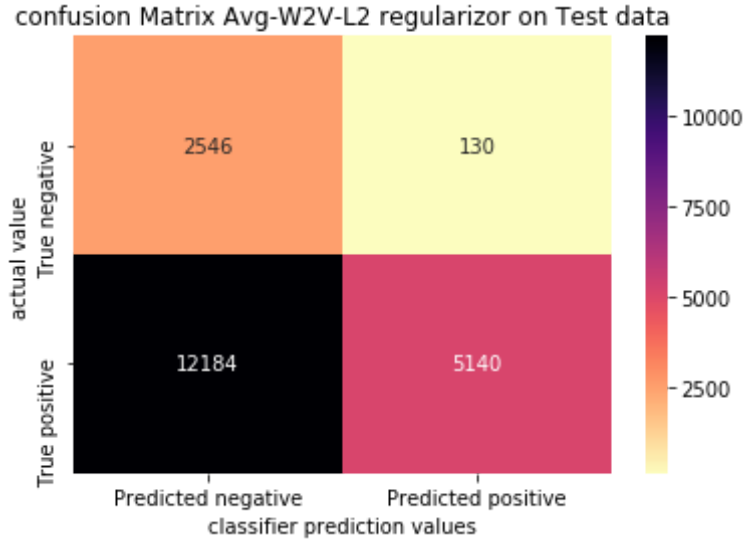
```
/home/jalesh_j/.local/lib/python3.5/site-packages/sklearn/linear_model/
sag.py:334: ConvergenceWarning: The max_iter was reached which means th
e coef_ did not converge
  "the coef_ did not converge", ConvergenceWarning)
```

confusion matrix on test data

In [98]:
```
%time
y_pred_test = np.where(y_pred_test[:,1] < 0.5, 0, 1)
#creating confusion matrix:

cf = confusion_matrix(ytest, y_pred_test)
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-L2 regularizor on Test data")
plt.xlabel("classifier prediction values")
```

```
plt.ylabel("actual value")
plt.show()
```

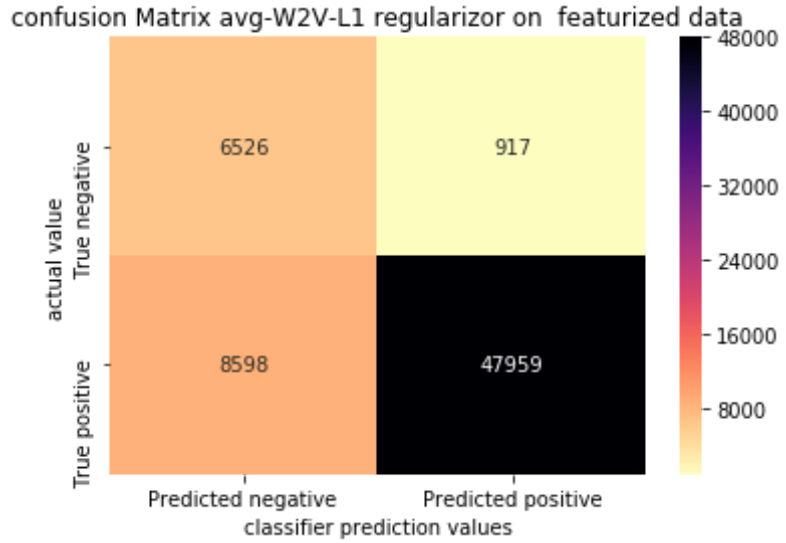
```
CPU times: user 0 ns, sys: 0 ns, total: 0 ns
Wall time: 5.96 µs
```



confusion matrix on train data

In [102]:
```
#y_pred_tr = np.where(y_pred_tr[:,1] > 0.5, 1, 0)
#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-L1 regularizor on  featurized data"
)
plt.xlabel("classifier prediction values")
```

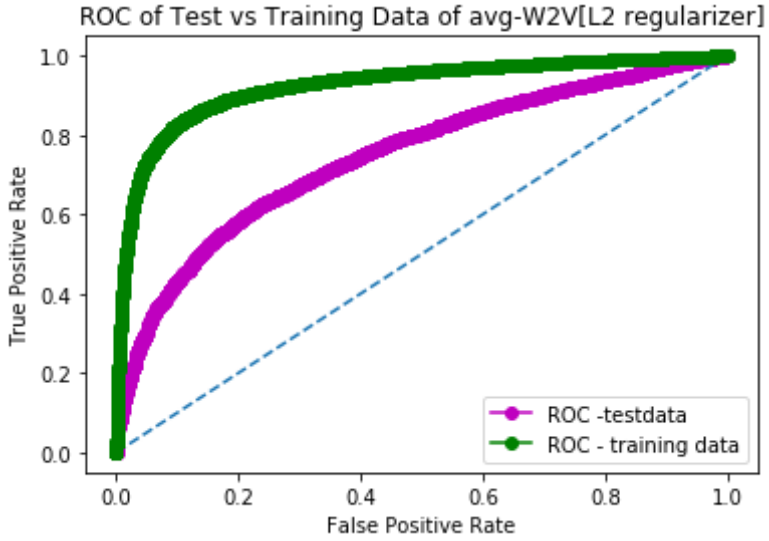
```
plt.ylabel("actual value")
plt.show()
```



ROC of train vs test data

In [100]:
```
plt.plot(fpr_test, tpr_test, color='m', marker='o',label='ROC -testdat
a')
plt.plot([0, 1], [0, 1], linestyle='--')
plt.plot(fpr_tr, tpr_tr, linestyle='-', color='g', marker='o', label='R
OC - training data')
plt.xlabel('False Positive Rate')
plt.ylabel('True Positive Rate')
plt.title('ROC of Test vs Training Data of avg-W2V[L1 regularizer with
 featurized text]')
plt.legend()
```

Out[100]:
```
<matplotlib.legend.Legend at 0x7fcd162da710>
```



Conclusion on using featurized column : we could see the performance of model on unseen data point is increaed by almost 15%(from 60.1 % to 74.6%) when we used featurized column.

TFidf-W2V[L1 regulazier]

In [9]:
```
#train, cv, test split:
print(type(w2v))
xtrain, xtest, ytrain, ytest = train_test_split(w2v, y, test_size=0.2,
shuffle=False)
xtr, xcv, ytr, ycv = train_test_split(xtrain, ytrain, test_size=0.2, sh
uffle=False)
```

```
<class 'pandas.core.series.Series'>
```

In [10]:
```
from sklearn.feature_extraction.text import TfidfVectorizer
```

```
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_)))
```

TFIDF-W2V for training data

In [11]:
```
%%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

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
```

```
CPU times: user 1h 13min 22s, sys: 7.94 s, total: 1h 13min 29s
Wall time: 1h 13min 20s
```

TFIDF-W2V for CV data

In [12]:
```
%%time

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
```

```
CPU times: user 14min 41s, sys: 1.53 s, total: 14min 42s
Wall time: 14min 40s
```

TFIDF-W2V for test data

In [13]:
```
%%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

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
```

```
CPU times: user 19min 4s, sys: 2.44 s, total: 19min 7s
Wall time: 19min 4s
```

In [15]:
```
import pickle

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

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

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

conversion of list into array

In [16]:
```
xtr = np.array(tfidf_train_vectors)
xcv = np.array(tfidf_cv_vectors)
xtest = np.array(tfidf_test_vectors)
```

standardizing data

In [17]:
```
#standardizing the data:
sc = StandardScaler(with_mean=False)
xtr = sc.fit_transform(xtr)
xcv = sc.transform(xcv)
xtest = sc.transform(xtest)
print(xtr.shape)
print(xcv.shape)
print(xtest.shape)
```

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

TFidf-W2V [L1 regularizer]

In [18]:
```
%%time

auc_cv_dict = {}
auc_tr_dict = {}

c_val = [10 ** -4, 10 ** -3, 10 ** -2, 10 ** -1, 10**1, 10 ** 2, 10 **
3, 10** 4]

for i in c_val:
    lr = LogisticRegression(penalty='l1', C=i, class_weight='balanced',
 random_state=56, n_jobs=-1,max_iter=4, solver='saga')
    lr.fit(xtr, ytr)
    y_pred_cv = lr.predict_proba(xcv)
    fpr_cv, tpr_cv, thresholds_cv = roc_curve(ycv, y_pred_cv[:,1])
    auc_cv_dict[i] = auc(fpr_cv, tpr_cv)
    #performance metrics for training data:
    y_pred_tr = lr.predict_proba(xtr)
    fpr_tr, tpr_tr, thresholds_tr = roc_curve(ytr, y_pred_tr[:,1])
    auc_tr_dict[i] = auc(fpr_tr, tpr_tr)
```

```
/home/jalesh_j/.local/lib/python3.5/site-packages/sklearn/linear_model/
sag.py:334: ConvergenceWarning: The max_iter was reached which means th
```

```
sag.py:334: ConvergenceWarning: The max_iter was reached which means th
e coef_ did not converge
  "the coef_ did not converge", ConvergenceWarning)
/home/jalesh_j/.local/lib/python3.5/site-packages/sklearn/linear_model/
sag.py:334: ConvergenceWarning: The max_iter was reached which means th
e coef_ did not converge
  "the coef_ did not converge", ConvergenceWarning)
/home/jalesh_j/.local/lib/python3.5/site-packages/sklearn/linear_model/
sag.py:334: ConvergenceWarning: The max_iter was reached which means th
e coef_ did not converge
  "the coef_ did not converge", ConvergenceWarning)
/home/jalesh_j/.local/lib/python3.5/site-packages/sklearn/linear_model/
sag.py:334: ConvergenceWarning: The max_iter was reached which means th
e coef_ did not converge
  "the coef_ did not converge", ConvergenceWarning)
/home/jalesh_j/.local/lib/python3.5/site-packages/sklearn/linear_model/
sag.py:334: ConvergenceWarning: The max_iter was reached which means th
e coef_ did not converge
  "the coef_ did not converge", ConvergenceWarning)
/home/jalesh_j/.local/lib/python3.5/site-packages/sklearn/linear_model/
sag.py:334: ConvergenceWarning: The max_iter was reached which means th
e coef_ did not converge
  "the coef_ did not converge", ConvergenceWarning)
/home/jalesh_j/.local/lib/python3.5/site-packages/sklearn/linear_model/
sag.py:334: ConvergenceWarning: The max_iter was reached which means th
e coef_ did not converge
  "the coef_ did not converge", ConvergenceWarning)
```

```
CPU times: user 40.5 s, sys: 6.83 s, total: 47.3 s
Wall time: 37 s
```

```
/home/jalesh_j/.local/lib/python3.5/site-packages/sklearn/linear_model/
sag.py:334: ConvergenceWarning: The max_iter was reached which means th
e coef_ did not converge
  "the coef_ did not converge", ConvergenceWarning)
```

optimal C value for AUC score on training and CV data

In [19]:
```
#sorting dictionary wrt higest AuC Score of both training and cv data:
```

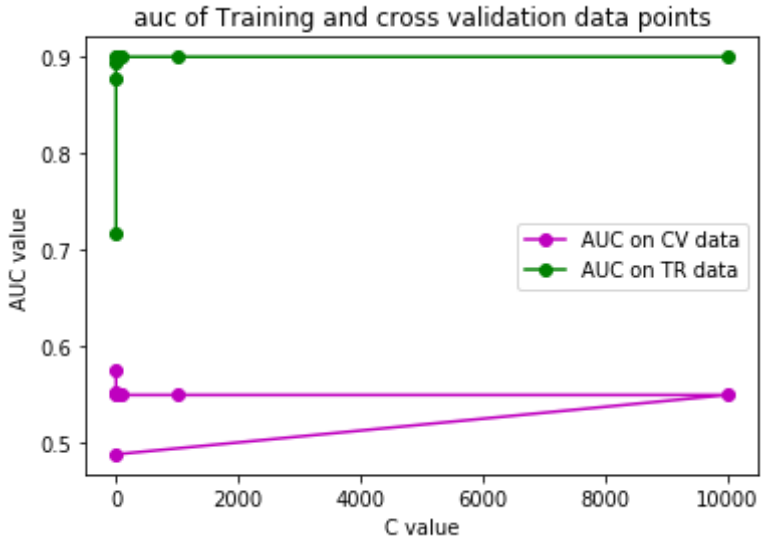
```
cv_tup = sorted(auc_cv_dict.items(), key= lambda x: x[1],reverse=True)
tr_tup = sorted(auc_tr_dict.items(), key= lambda x: x[1],reverse=True)
print(cv_tup)
print('*' * 70)
print(tr_tup)
```

```
[(0.001, 0.5737799964135306), (0.01, 0.5513072492853907), (0.1, 0.55003
87303791134), (10, 0.5489669505239037), (100, 0.5489482598107117), (100
0, 0.5489470853493802), (10000, 0.5489469175691899), (0.0001, 0.4873940
0318701825)]
**********************************************************************
[(10000, 0.8995013029352956), (1000, 0.8995013005597376), (100, 0.89950
10701306234), (10, 0.8994978274941182), (0.1, 0.8990692792282542), (0.0
1, 0.8944466692731763), (0.001, 0.8780469341868389), (0.0001, 0.7156053
83689763)]
```

Plotting AUC Curve on training and cv data

In [20]:
```
plt.plot([x[0] for x in cv_tup], [x[1] for x in cv_tup], linestyle='-',
 color='m', marker='o',label='AUC on CV data')
plt.plot([x[0] for x in tr_tup], [x[1] for x in tr_tup], linestyle='-',
 color='g', marker='o', label='AUC on TR data')
#plt.plot([0, 1], [0, 1], linestyle='--')
plt.xlabel("C value")
plt.ylabel('AUC value')
plt.title('auc of Training and cross validation data points')
plt.legend()
```

Out[20]:
```
<matplotlib.legend.Legend at 0x7f628d0b8860>
```



optimal LR-TFidf-W2V(L1) for C=0.001

In [21]:
```
lr = LogisticRegression(penalty='l1', C=0.001, class_weight='balanced',
 random_state=56, n_jobs=-1,max_iter=4, solver='saga')
lr.fit(xtr, ytr)
y_pred_test = lr.predict_proba(xtest)
y_pred = lr.predict(xtest)
fpr_test, tpr_test, thresholds_test= roc_curve(ytest, y_pred_test[:,1])
auc_test = auc(fpr_test, tpr_test)
optimal_weight = lr.coef_
print(auc_test)
```

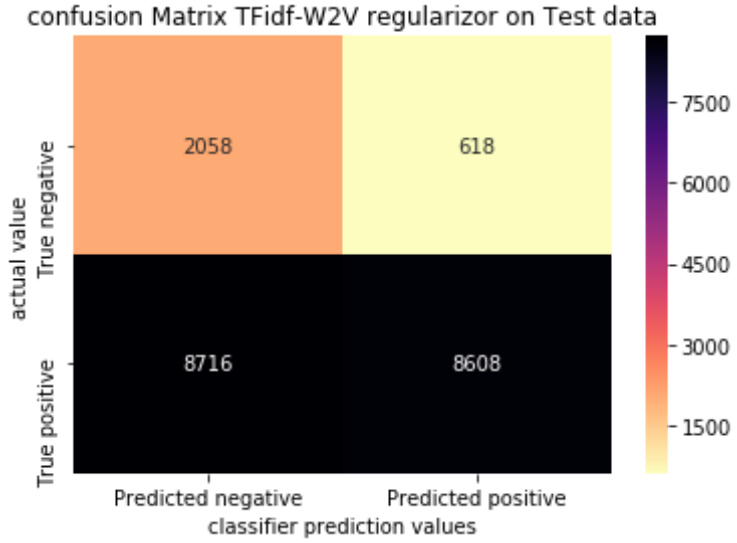
```
0.6787495785070885
```

```
/home/jalesh_j/.local/lib/python3.5/site-packages/sklearn/linear_model/
sag.py:334: ConvergenceWarning: The max_iter was reached which means th
e coef_ did not converge
  "the coef_ did not converge", ConvergenceWarning)
```

plotting confusion matrix on test data

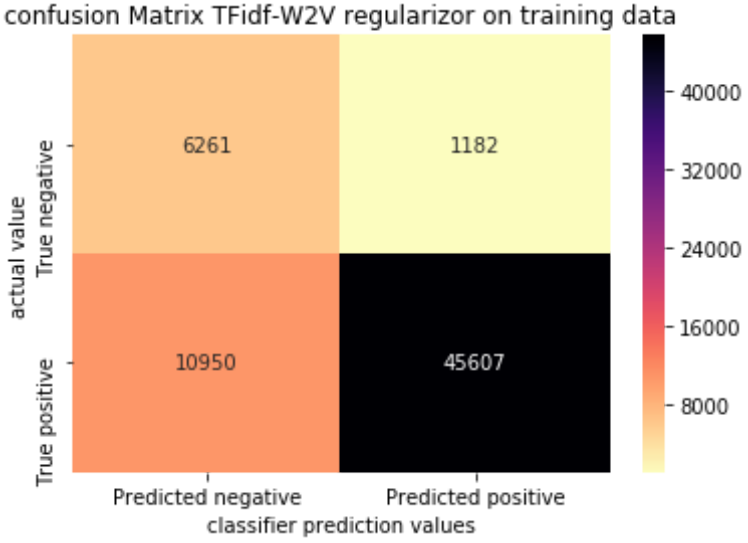
In [22]:
```
%time
y_pred_test = np.where(y_pred_test[:,1] < 0.5, 0, 1)
#creating confusion matrix:

cf = confusion_matrix(ytest, y_pred_test)
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 regularizor on Test data")
plt.xlabel("classifier prediction values")
plt.ylabel("actual value")
plt.show()
```

```
CPU times: user 0 ns, sys: 0 ns, total: 0 ns
Wall time: 5.96 µs
```



plotting confusion matrix on train data

In [23]:
```
y_pred_tr = np.where(y_pred_tr[:,1] > 0.5, 1, 0)
#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 regularizor on training data")
plt.xlabel("classifier prediction values")
plt.ylabel("actual value")
plt.show()
```



plotting ROC curve on test vs 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('ROC of Test vs Training Data')
plt.legend()
```

Out[24]: <matplotlib.legend.Legend at 0x7f628cff7080>

TF-idf-W2v [L2 regularizer]

In [25]:
```
%%time
from sklearn.linear_model import LogisticRegression

auc_cv_dict = {}
auc_tr_dict = {}

c_val = [10 ** -4, 10 ** -3, 10 ** -2, 10 ** -1, 10**1, 10 ** 2, 10 **
3, 10** 4]

for i in c_val:
```

```
    lr = LogisticRegression(penalty='l2', C=i, class_weight='balanced',
random_state=56, n_jobs=-1,max_iter=4, solver='sag')
    lr.fit(xtr, ytr)
    y_pred_cv = lr.predict_proba(xcv)
    fpr_cv, tpr_cv, thresholds_cv = roc_curve(ycv, y_pred_cv[:,1])
    auc_cv_dict[i] = auc(fpr_cv, tpr_cv)
    #performance metrics for training data:
    y_pred_tr = lr.predict_proba(xtr)
    fpr_tr, tpr_tr, thresholds_tr = roc_curve(ytr, y_pred_tr[:,1])
    auc_tr_dict[i] = auc(fpr_tr, tpr_tr)
```

```
/home/jalesh_j/.local/lib/python3.5/site-packages/sklearn/linear_model/
sag.py:334: ConvergenceWarning: The max_iter was reached which means th
e coef_ did not converge
  "the coef_ did not converge", ConvergenceWarning)
/home/jalesh_j/.local/lib/python3.5/site-packages/sklearn/linear_model/
sag.py:334: ConvergenceWarning: The max_iter was reached which means th
e coef_ did not converge
  "the coef_ did not converge", ConvergenceWarning)
/home/jalesh_j/.local/lib/python3.5/site-packages/sklearn/linear_model/
sag.py:334: ConvergenceWarning: The max_iter was reached which means th
e coef_ did not converge
  "the coef_ did not converge", ConvergenceWarning)
/home/jalesh_j/.local/lib/python3.5/site-packages/sklearn/linear_model/
sag.py:334: ConvergenceWarning: The max_iter was reached which means th
e coef_ did not converge
  "the coef_ did not converge", ConvergenceWarning)
/home/jalesh_j/.local/lib/python3.5/site-packages/sklearn/linear_model/
sag.py:334: ConvergenceWarning: The max_iter was reached which means th
e coef_ did not converge
  "the coef_ did not converge", ConvergenceWarning)
/home/jalesh_j/.local/lib/python3.5/site-packages/sklearn/linear_model/
sag.py:334: ConvergenceWarning: The max_iter was reached which means th
e coef_ did not converge
  "the coef_ did not converge", ConvergenceWarning)
/home/jalesh_j/.local/lib/python3.5/site-packages/sklearn/linear_model/
sag.py:334: ConvergenceWarning: The max_iter was reached which means th
e coef_ did not converge
  "the coef_ did not converge", ConvergenceWarning)
/home/jalesh_j/.local/lib/python3.5/site-packages/sklearn/linear_model/
```

```
sag.py:334: ConvergenceWarning: The max_iter was reached which means th

e coef_ did not converge
  "the coef_ did not converge", ConvergenceWarning)
```

```
CPU times: user 24.2 s, sys: 6.89 s, total: 31.1 s
Wall time: 20.7 s
```

optimal C

In [26]:
```
#sorting dictionary wrt higest AuC Score of both training and cv data:

cv_tup = sorted(auc_cv_dict.items(), key= lambda x: x[1],reverse=True)
tr_tup = sorted(auc_tr_dict.items(), key= lambda x: x[1],reverse=True)
print(cv_tup)
print('*' * 70)
print(tr_tup)
```

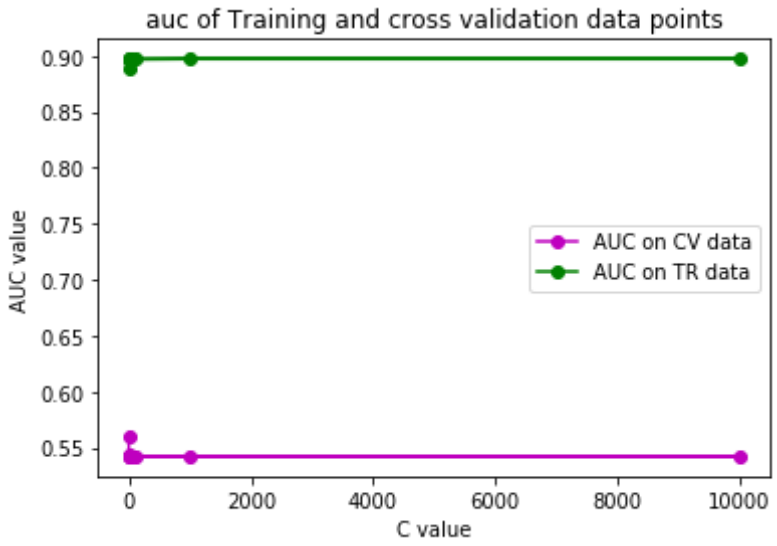
```
[(0.0001, 0.5608166949074282), (0.001, 0.5442380994182188), (0.01, 0.54
26059337276303), (0.1, 0.5422269853899695), (10, 0.5421829598680523),
(10000, 0.5421825907516338), (1000, 0.5421825907516338), (100, 0.542182
4565274815)]
**********************************************************************
[(0.01, 0.8972943407742671), (0.1, 0.8972262952943731), (10, 0.89721591
88575564), (100, 0.8972159164819985), (10000, 0.8972158832241882), (100
0, 0.8972158832241882), (0.001, 0.8967812701590585), (0.0001, 0.8894408
509499183)]
```

AUC of train vs cv data

In [27]:
```
plt.plot([x[0] for x in cv_tup], [x[1] for x in cv_tup], linestyle='-',
 color='m', marker='o',label='AUC on CV data')
plt.plot([x[0] for x in tr_tup], [x[1] for x in tr_tup], linestyle='-',
 color='g', marker='o', label='AUC on TR data')
#plt.plot([0, 1], [0, 1], linestyle='--')
plt.xlabel("C value")
plt.ylabel('AUC value')
```

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

Out[27]: <matplotlib.legend.Legend at 0x7f628c2daf98>



optimal TFidf-W2V(L2)

In [28]:
```
lr = LogisticRegression(penalty='l2', C=0.0001, class_weight='balanced'
, random_state=56, n_jobs=-1,max_iter=4, solver='sag')
lr.fit(xtr, ytr)
y_pred_test = lr.predict_proba(xtest)
y_pred = lr.predict(xtest)
fpr_test, tpr_test, thresholds_test= roc_curve(ytest, y_pred_test[:,1])
auc_test = auc(fpr_test, tpr_test)
print(auc_test)
```

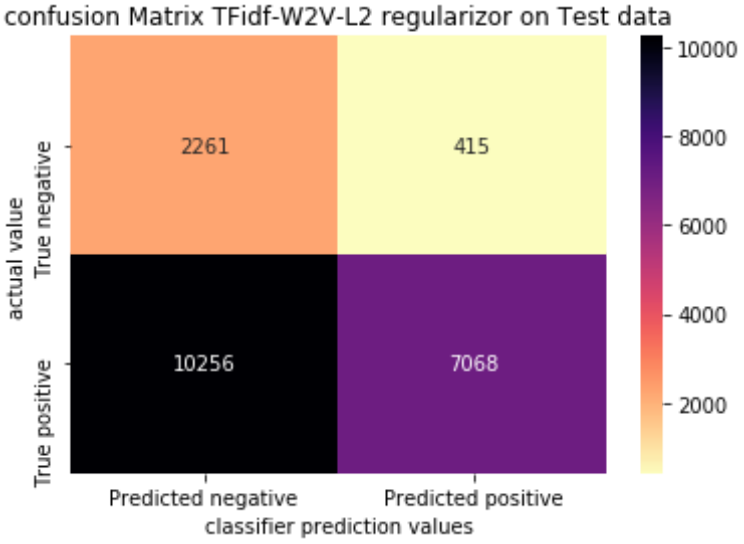
```
0.698095887437147
```

```
/home/jalesh_j/.local/lib/python3.5/site-packages/sklearn/linear_model/
sag.py:334: ConvergenceWarning: The max_iter was reached which means th
e coef_ did not converge
  "the coef_ did not converge", ConvergenceWarning)
```

confusion matrix on test data

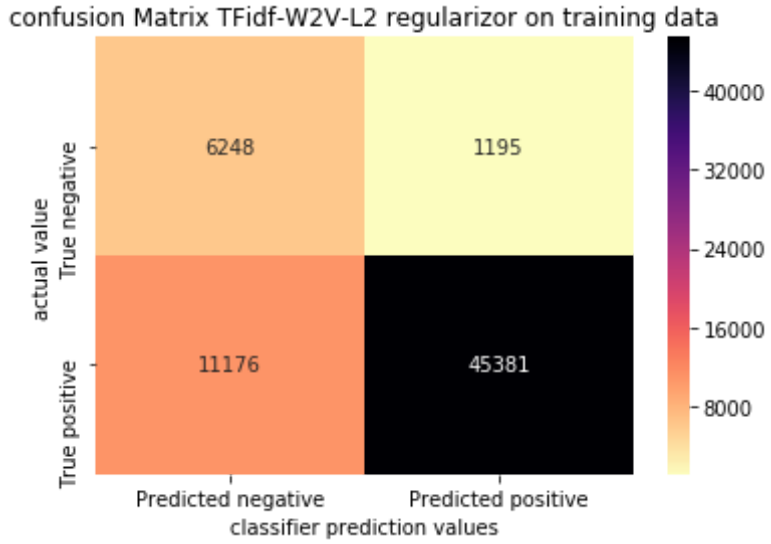
In [29]:
```
%time
y_pred_test = np.where(y_pred_test[:,1] < 0.5, 0, 1)
#creating confusion matrix:

cf = confusion_matrix(ytest, y_pred_test)
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-L2 regularizor on Test data")
plt.xlabel("classifier prediction values")
plt.ylabel("actual value")
plt.show()
```

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



confusion matrix on train data

In [30]:
```
y_pred_tr = np.where(y_pred_tr[:,1] > 0.5, 1, 0)
#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-L2 regularizor on training data")
plt.xlabel("classifier prediction values")
plt.ylabel("actual value")
plt.show()
```

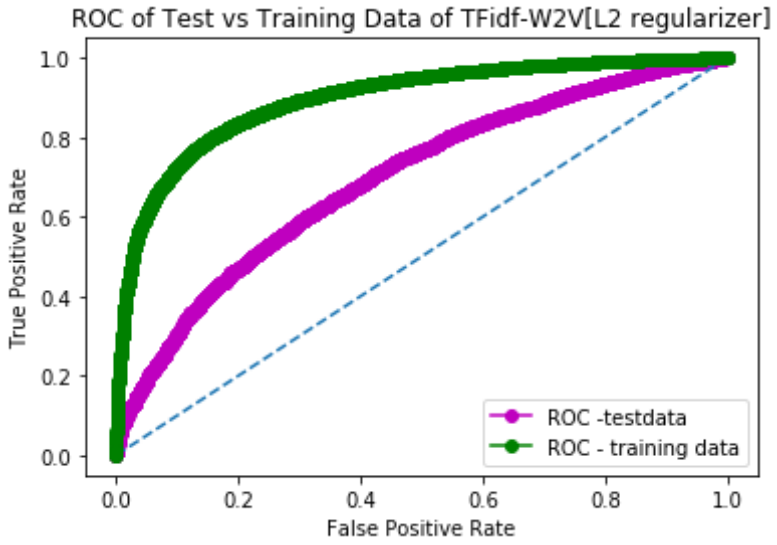


ROC of train vs test data

In [31]:
```
plt.plot(fpr_test, tpr_test, color='m', marker='o',label='ROC -testdat
```

```
a')
plt.plot([0, 1], [0, 1], linestyle='--')
plt.plot(fpr_tr, tpr_tr, linestyle='-', color='g', marker='o', label='R
OC - training data')
plt.xlabel('False Positive Rate')
plt.ylabel('True Positive Rate')
plt.title('ROC of Test vs Training Data of TFidf-W2V[L2 regularizer]')
plt.legend()
```

Out[31]: <matplotlib.legend.Legend at 0x7f62f20bd4e0>



TFidf-W2V -> FeatureEngineering

In [32]:
```
df.columns
```

Out[32]:
```
Index(['index', 'Score', 'Time', 'Text', 'Summary', 'cleanedtext',
       'numeric_score', 'bow_feat', 'bow_new_feat', 'tfw2v_feat'],
      dtype='object')
```

In [33]:
```
#train, cv, test split of featurized column:
```

```
xtrain, xtest, ytrain, ytest = train_test_split(w2v_featured, y, test_s
ize=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 [34]:
```
%%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 2.49 s, sys: 80 ms, total: 2.57 s
Wall time: 2.57 s
```

instantiating word2vec object for Train, cv, test data

In [35]:
```
%%time

from gensim.models import Word2Vec

#instantiating training,cv, test word to vector object:
trainw2v = Word2Vec(train_list, size=1000, workers=8)
cvw2v = Word2Vec(cv_list, size=1000,workers=8)
testw2v = Word2Vec(test_list, size=1000, workers=8)

#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 5min 46s, sys: 1.48 s, total: 5min 48s
Wall time: 48 s
```

Featurized TFidf-W2V on train, cv and test data

In [36]:
```
%%time

#featurized train data:

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

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


#featurized 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



#featurized test data

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

```
ll_val = tfidf

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
```

```
CPU times: user 2h 1min 4s, sys: 13.9 s, total: 2h 1min 18s
Wall time: 2h 58s
```

In [37]:
```
#pickeling and saving it onto disc:
file1 = open('tfidfw2v_featured_train.pickle', 'wb')
pickle.dump(tfidf_train_vectors, file1)
file1.close()

file1 = open('tfidfw2v_featured_cv.pickle', 'wb')
pickle.dump(tfidf_cv_vectors, file1)
file1.close()

file1 = open('tfidfw2v_featured_test.pickle', 'wb')
pickle.dump(tfidf_test_vectors, file1)
file1.close()
```

conversion of list into array

In [38]:
```
xtr = np.array(tfidf_train_vectors)
xcv = np.array(tfidf_cv_vectors)
xtest = np.array(tfidf_test_vectors)
```

column standardization

In [41]:
```
sc = StandardScaler(with_mean=False)
xtr = sc.fit_transform(xtr)
xcv = sc.transform(xcv)
xtest = sc.transform(xtest)
```

TFidf-W2V[L1] regularizer on featurized text

In [42]:
```
%%time

auc_cv_dict = {}
auc_tr_dict = {}

c_val = [10 ** -4, 10 ** -3, 10 ** -2, 10 ** -1, 10**1, 10 ** 2, 10 **
3, 10** 4]

for i in c_val:
    lr = LogisticRegression(penalty='l1', C=i, class_weight='balanced',
 random_state=56, n_jobs=-1,max_iter=4, solver='saga')
    lr.fit(xtr, ytr)
    y_pred_cv = lr.predict_proba(xcv)
    fpr_cv, tpr_cv, thresholds_cv = roc_curve(ycv, y_pred_cv[:,1])
    auc_cv_dict[i] = auc(fpr_cv, tpr_cv)
    #performance metrics for training data:
    y_pred_tr = lr.predict_proba(xtr)
    fpr_tr, tpr_tr, thresholds_tr = roc_curve(ytr, y_pred_tr[:,1])
    auc_tr_dict[i] = auc(fpr_tr, tpr_tr)
```

```
/home/jalesh_j/.local/lib/python3.5/site-packages/sklearn/linear_model/
sag.py:334: ConvergenceWarning: The max_iter was reached which means th
e coef_ did not converge
  "the coef_ did not converge", ConvergenceWarning)
```

```
/home/jalesh_j/.local/lib/python3.5/site-packages/sklearn/linear_model/
sag.py:334: ConvergenceWarning: The max_iter was reached which means th
e coef_ did not converge
  "the coef_ did not converge", ConvergenceWarning)
/home/jalesh_j/.local/lib/python3.5/site-packages/sklearn/linear_model/
sag.py:334: ConvergenceWarning: The max_iter was reached which means th
e coef_ did not converge
  "the coef_ did not converge", ConvergenceWarning)
/home/jalesh_j/.local/lib/python3.5/site-packages/sklearn/linear_model/
sag.py:334: ConvergenceWarning: The max_iter was reached which means th
e coef_ did not converge
  "the coef_ did not converge", ConvergenceWarning)
/home/jalesh_j/.local/lib/python3.5/site-packages/sklearn/linear_model/
sag.py:334: ConvergenceWarning: The max_iter was reached which means th
e coef_ did not converge
  "the coef_ did not converge", ConvergenceWarning)
/home/jalesh_j/.local/lib/python3.5/site-packages/sklearn/linear_model/
sag.py:334: ConvergenceWarning: The max_iter was reached which means th
e coef_ did not converge
  "the coef_ did not converge", ConvergenceWarning)
/home/jalesh_j/.local/lib/python3.5/site-packages/sklearn/linear_model/
sag.py:334: ConvergenceWarning: The max_iter was reached which means th
e coef_ did not converge
  "the coef_ did not converge", ConvergenceWarning)
```

```
CPU times: user 42 s, sys: 7.24 s, total: 49.3 s
Wall time: 38 s
```

```
/home/jalesh_j/.local/lib/python3.5/site-packages/sklearn/linear_model/
sag.py:334: ConvergenceWarning: The max_iter was reached which means th
e coef_ did not converge
  "the coef_ did not converge", ConvergenceWarning)
```

optimal C

In [43]:

```
#sorting dictionary wrt higest AuC Score of both training and cv data:

cv_tup = sorted(auc_cv_dict.items(), key= lambda x: x[1],reverse=True)
tr_tup = sorted(auc_tr_dict.items(), key= lambda x: x[1],reverse=True)
```

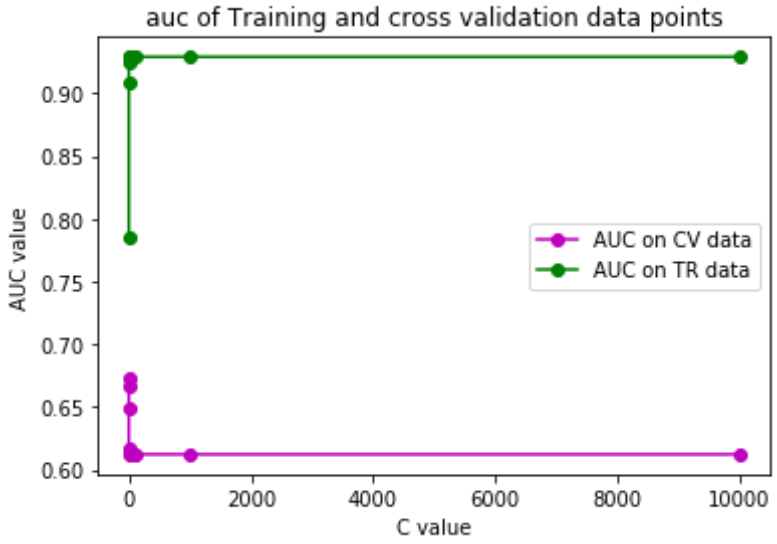
```
print(cv_tup)
print('*' * 70)
print(tr_tup)
```

```
[(0.001, 0.6728175219966541), (0.0001, 0.6658190747016465), (0.01, 0.64
92498749030902), (0.1, 0.6168433660518127), (10, 0.6122215587074966),
(100, 0.6121868617641564), (1000, 0.6121825665912863), (10000, 0.612182
2981429821)]
**********************************************************************
[(10000, 0.9290203794858215), (1000, 0.9290203747347057), (100, 0.92902
01894411911), (10, 0.929017084587043), (0.1, 0.9286499729990528), (0.0
1, 0.9250308652078028), (0.001, 0.9084884731196992), (0.0001, 0.7847024
612924758)]
```

AUC of train vs cv data

In [44]:
```
plt.plot([x[0] for x in cv_tup], [x[1] for x in cv_tup], linestyle='-',
 color='m', marker='o',label='AUC on CV data')
plt.plot([x[0] for x in tr_tup], [x[1] for x in tr_tup], linestyle='-',
 color='g', marker='o', label='AUC on TR data')
#plt.plot([0, 1], [0, 1], linestyle='--')
plt.xlabel("C value")
plt.ylabel('AUC value')
plt.title('auc of Training and cross validation data points')
plt.legend()
```

Out[44]:
```
<matplotlib.legend.Legend at 0x7f629f80e780>
```



optimal LR-TFidf-W2V model for C=0.001

In [45]:
```
%%time

lr = LogisticRegression(penalty='l1', C=0.001, class_weight='balanced',
 random_state=56, n_jobs=-1,max_iter=4, solver='saga')
lr.fit(xtr, ytr)
y_pred_test = lr.predict_proba(xtest)
y_pred = lr.predict(xtest)
fpr_test, tpr_test, thresholds_test= roc_curve(ytest, y_pred_test[:,1])
auc_test = auc(fpr_test, tpr_test)
optimal_weight = lr.coef_
print(auc_test)
```

```
0.6674664246598462
CPU times: user 3.81 s, sys: 192 ms, total: 4 s
Wall time: 3.7 s
```

```
/home/jalesh_j/.local/lib/python3.5/site-packages/sklearn/linear_model/
sag.py:334: ConvergenceWarning: The max_iter was reached which means th
```

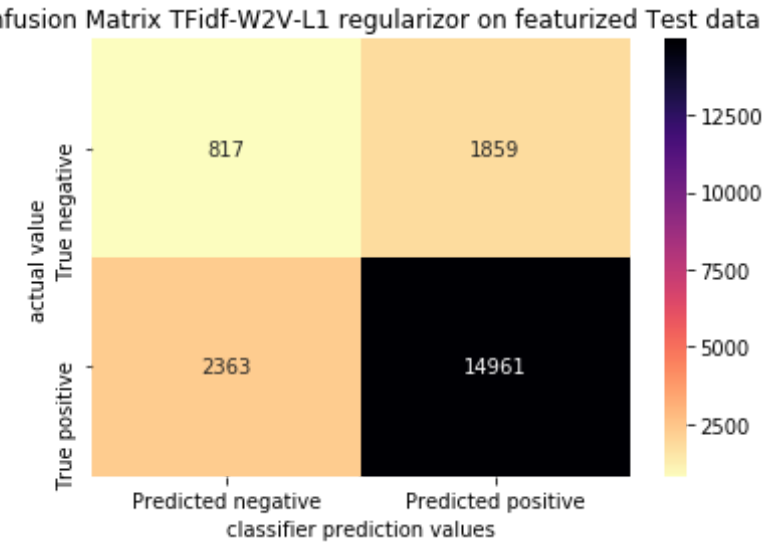
```
e coef_ did not converge
  "the coef_ did not converge", ConvergenceWarning)
```

confusion matrix on test data

In [46]:

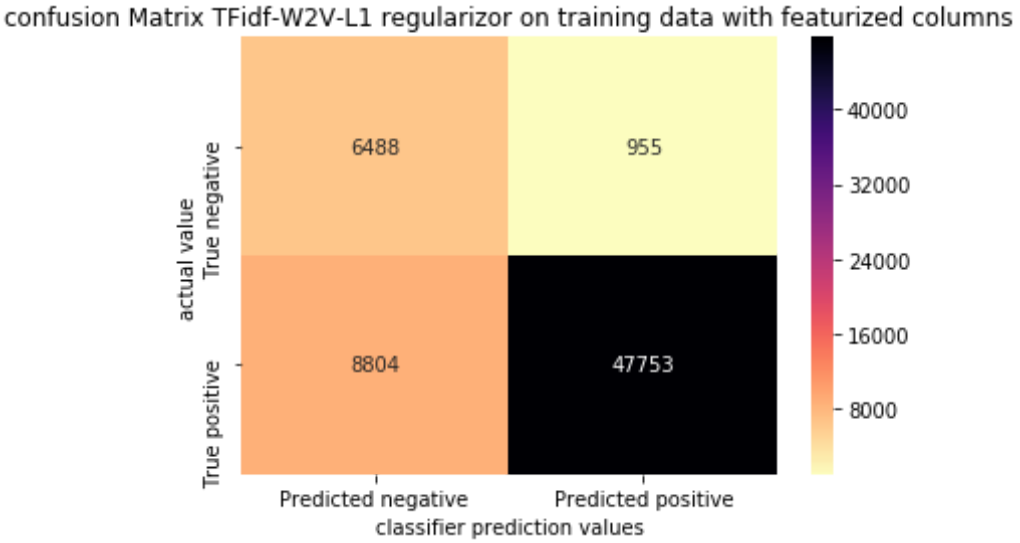
```
%time
y_pred_test = np.where(y_pred_test[:,1] < 0.5, 0, 1)
#creating confusion matrix:

cf = confusion_matrix(ytest, y_pred_test)
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-L1 regularizor on featurized Test
 data")
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

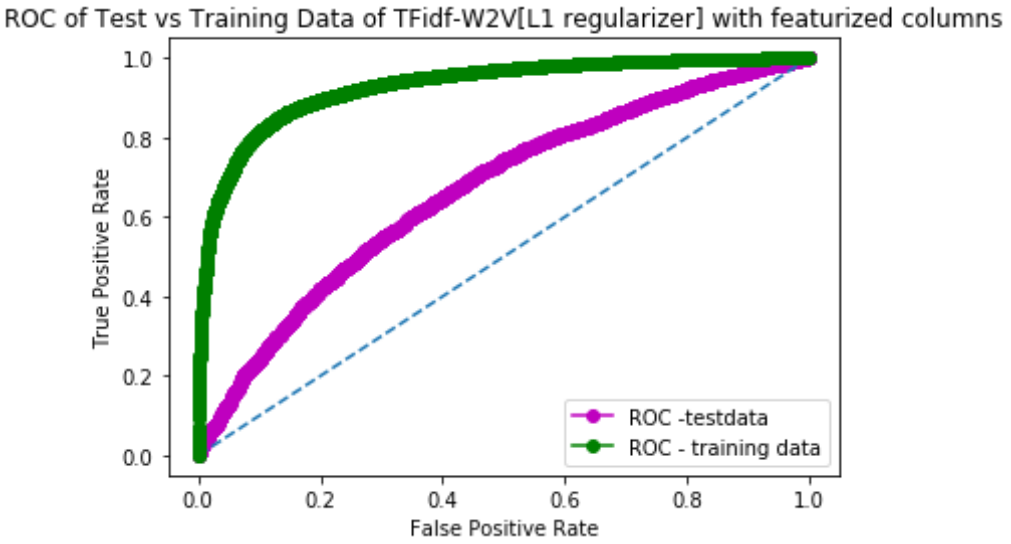
In [47]:
```
y_pred_tr = np.where(y_pred_tr[:,1] > 0.5, 1, 0)
#creating confusion matrix:

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-L1 regularizor on training data w
ith featurized columns")
plt.xlabel("classifier prediction values")
plt.ylabel("actual value")
plt.show()
```



ROC-AUC on train vs test data

In [48]:
```
plt.plot(fpr_test, tpr_test, color='m', marker='o',label='ROC -testdat
a')
plt.plot([0, 1], [0, 1], linestyle='--')
plt.plot(fpr_tr, tpr_tr, linestyle='-', color='g', marker='o', label='R
OC - training data')
plt.xlabel('False Positive Rate')
plt.ylabel('True Positive Rate')
plt.title('ROC of Test vs Training Data of TFidf-W2V[L1 regularizer] wi
th featurized columns')
plt.legend()
```

Out[48]:
```
<matplotlib.legend.Legend at 0x7f62bc6100b8>
```



we could see that after feature engineering applied we could see the performance of the model is increased by almost 9% from 59 % to 68 %

Logistic Regression performance consolidation of 4 vectorizers

In [49]:
```
perf_dict = dict(algorithm = ['Bow-L1', 'Tfidf-L1', 'W2V-L1', 'Tfidf-W2
V-L1',
                              'Bow-L2', 'Tfidf-L2', 'W2V-L2', 'Tfidf-W2V
-L2'],
                 C_optimal = [0.01, 0.01, 0.001, 0.001, 0.0001, 0.0001,
0.0001, 0.0001],
                 AUC = [0.928, 0.937, 0.730, 0.678, 0.926, 0.929,0.707,
0.698])

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

Out[49]:

	AUC	C_optimal	algorithm

0	0.928	0.0100	Bow-L1
1	0.937	0.0100	Tfidf-L1

	AUC	C_optimal	algorithm
2	0.730	0.0010	W2V-L1
3	0.678	0.0010	Tfidf-W2V-L1
4	0.926	0.0001	Bow-L2
5	0.929	0.0001	Tfidf-L2
6	0.707	0.0001	W2V-L2
7	0.698	0.0001	Tfidf-W2V-L2

Conclusions -- out of 1L datapoints:

1. TFidf with L1 performed best on unseen test data point with 93% AUC score.
2. TFidf-w2v with L1 performed least on unseen test datapoints with 67% AUC score.
3. perturbation test was performed on BoW with L1 and it was observed multicollinearity amongst features.
4. feature engineering was performed on W2V and TFidf-W2V with L1 regularizer
5. with feature engineered column, we observed dip in model performance of TFidf-w2v(L1) by 2 % (AUC ~ 67%)
6. with feature engineered column, we observed slightly increase in model performance of Avg-W2V by 2 % (AUC ~ 75)

In [ ]: