Linear SVC and SVC with RBF Kernal.

In [1]:

```
#to ignore warnings
import warnings
warnings.filterwarnings("ignore")
#to use sqlite3 database
import sqlite3
import numpy as np
import pandas as pd
import string
import nltk
import matplotlib.pyplot as plt

from nltk.stem.porter import PorterStemmer
from nltk.corpus import stopwords
from nltk.stem import PorterStemmer
from nltk.stem.wordnet import WordNetLemmatizer
import re
from sklearn.feature_extraction.text import CountVectorizer
from sklearn import cross_validation
from sklearn.metrics import accuracy_score
from sklearn.grid_search import GridSearchCV
from sklearn.grid_search import RandomizedSearchCV
from sklearn.linear_model import LogisticRegression
from sklearn.cross_validation import cross_val_score
from sklearn.feature_extraction.text import TfidfTransformer
from sklearn.feature_extraction.text import TfidfVectorizer
```

```
C:\Users\krush\Anaconda3\lib\site-packages\sklearn\cross_validation.py:
41: DeprecationWarning: This module was deprecated in version 0.18 in f
avor of the model_selection module into which all the refactored classe
s and functions are moved. Also note that the interface of the new CV i
terators are different from that of this module. This module will be re
moved in 0.20.
  "This module will be removed in 0.20.", DeprecationWarning)
```

```
C:\Users\krush\Anaconda3\lib\site-packages\sklearn\grid_search.py:42: D
eprecationWarning: This module was deprecated in version 0.18 in favor
of the model_selection module into which all the refactored classes and
functions are moved. This module will be removed in 0.20.
  DeprecationWarning)
```

In [2]:

```
con = sqlite3.connect('database.sqlite')

filtered_data = pd.read_sql_query(""" SELECT * FROM Reviews WHERE Score
 != 3 """, con)


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

#changing reviews with score less than 3 to be positive and vice-versa
actualScore = filtered_data['Score']
positiveNegative = actualScore.map(partition)
filtered_data['Score'] = positiveNegative
```

Text Preprocessing on all data points

In [3]:

```
stop = set(stopwords.words('english')) #set of stopwords
sno = nltk.stem.SnowballStemmer('english') #initialising the snowball s
temmer

def cleanhtml(sentence): #function to clean the word of any html-tags
    cleanr = re.compile('<.*?>')
    cleantext = re.sub(cleanr, ' ', sentence)
    return cleantext
def cleanpunc(sentence): #function to clean the word of any punctuation
 or special characters
    cleaned = re.sub(r'[?|!|\'|"|#]',r'',sentence)
    cleaned = re.sub(r'[.|,|)|(|\|/]',r' ',cleaned)
    return  cleaned
```

In [4]:

```
#Code for implementing step-by-step the checks mentioned in the pre-processing phase
# this code takes a while to run as it needs to run on 500k sentences.
i=0
str1=' '
final_string=[]
all_positive_words=[] # store words from +ve reviews here
all_negative_words=[] # store words from -ve reviews here.
s=''
for sent in filtered_data['Text'].values:
    filtered_sentence=[]
    #print(sent);
    sent=cleanhtml(sent) # remove HTMl tags
    for w in sent.split():
        for cleaned_words in cleanpunc(w).split():
            if((cleaned_words.isalpha()) & (len(cleaned_words)>2)):
                if(cleaned_words.lower() not in stop):
                    s=(sno.stem(cleaned_words.lower())).encode('utf8')
                    filtered_sentence.append(s)
                    if (filtered_data['Score'].values)[i] == 'positive':
                        all_positive_words.append(s) #list of all words used to describe positive reviews
                    if(filtered_data['Score'].values)[i] == 'negative':
                        all_negative_words.append(s) #list of all words used to describe negative reviews reviews
                else:
                    continue
            else:
                continue
    #print(filtered_sentence)
    str1 = b" ".join(filtered_sentence) #final string of cleaned words
    #print("************************************************************************")
    final_string.append(str1)
    i+=1
```

In [5]:

```
filtered_data['CleanedText']=final_string #adding a column of CleanedText which displays the data after pre-processing of the review
filtered_data['CleanedText']=filtered_data['CleanedText'].str.decode("utf-8")
```

In [6]:

```
sorted_data=filtered_data.sort_values(by=['Time'])
sampledata = sorted_data.head(50000)

S = sorted_data['Score']
Score = S.head(50000)
```

In [7]:

```
sampledata['Score'].value_counts()
```

Out[7]:

```
1    44521
0     5479
Name: Score, dtype: int64
```

Splitting the data in to train and test data

In [86]:

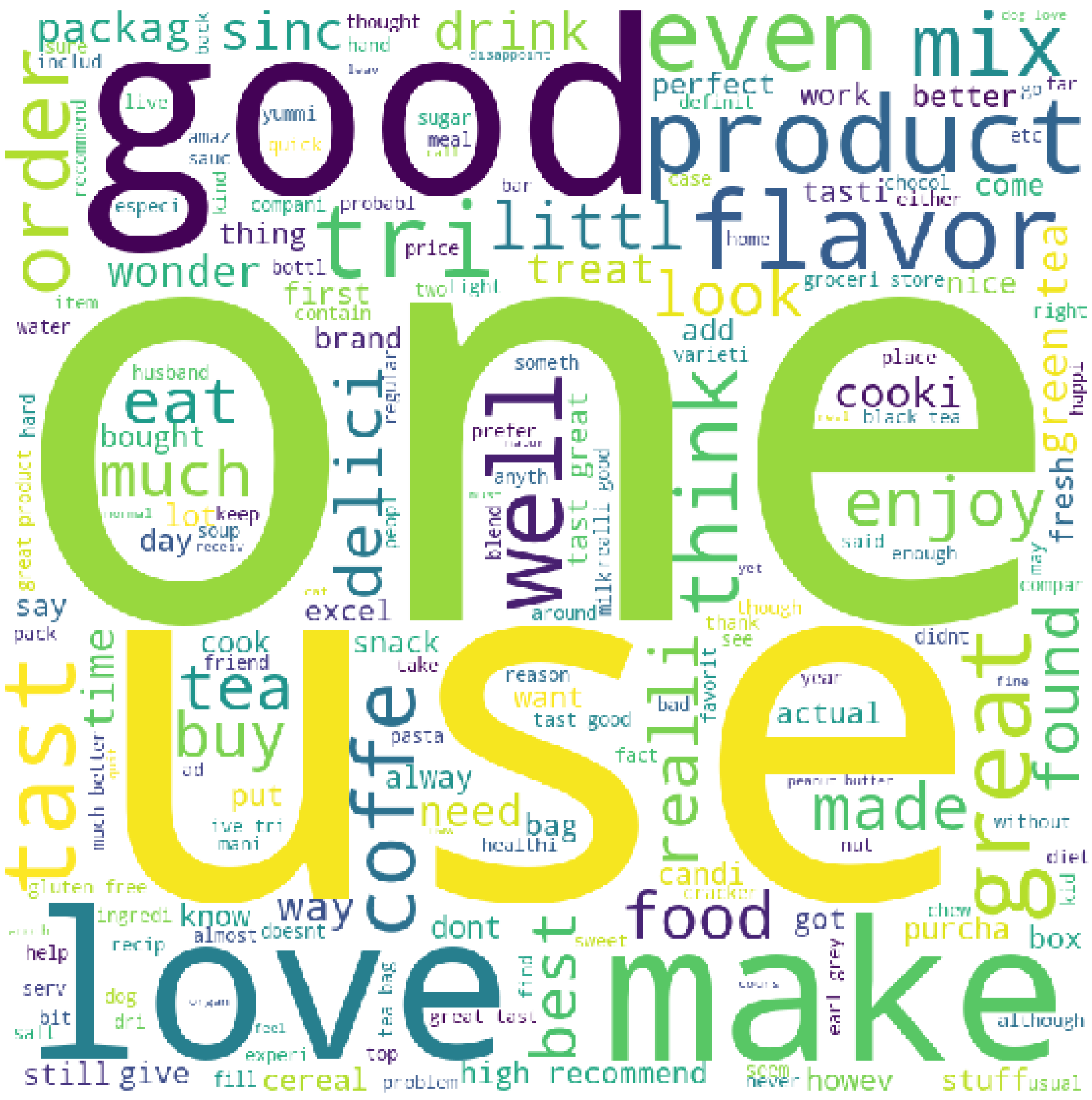
```
X_train, X_test, y_train, y_test = cross_validation.train_test_split(sampledata, Score, test_size=0.2, random_state=0)
```

In [131]:

```
comment_words = ' '
for val in X_train['CleanedText'].values:

    # typecaste each val to string
    val = str(val)

    # split the value
    tokens = val.split()

    # Converts each token into lowercase
    for i in range(len(tokens)):
        tokens[i] = tokens[i].lower()

    for words in tokens:
        comment_words = comment_words + words + ' '
```

In [132]:

```
from wordcloud import WordCloud
wordcloud = WordCloud(width = 800, height = 800,
                background_color ='white',
                min_font_size = 10).generate(comment_words)

# plot the WordCloud image
plt.figure(figsize = (8, 8), facecolor = None)
plt.imshow(wordcloud)
plt.axis("off")
plt.tight_layout(pad = 0)

plt.show()
```


one
use
good
love
make
product
flavor
tri
tast
great
coffe
order
even
mix
drink
packag
sinc
littl
well
think
realli
food
tea
best
buy
much
eat
delici
enjoy
made
found
time
look
treat
cooki
green tea
fresh
wonder
first
brand
excel
snack
need
alway
bag
candi
dont
purcha
box
got
high recommend
cereal
still
give
stuff
howev
actual
add
perfect
work
better
tasti
come
nice
thing
price
bought
day
lot
say
cook
way
know
help
dog
serv
bit
top
husband
water
item
yummi
sugar
meal
live
amaz
sauc
esperi
compani
probabl
bottl
contain
groceri store
right
varieti
someth
prefer
anyth
place
black tea
said
enough
though
thank
see
didnt
year
reason
want
tast good
bad
fact
pasta
peanut butter
without
diet
chew
although
earl grey
find
tea bag
great tast
problem
fill
experi
usual
never
gluten free
ingredi
recip
almost
doesnt
sweet
cracker
healthi
favorit
kid
happi
compar
may
blend
milk
around
yet
mani
ive tri
put
friend
much better
great product
hard
soup
keep
regular
peopl
tast great
definit
thought
hand
disappoint
sure
includ
recommend
kind
quick
bar
case
chocol
either
home
etc
far
dog love
two
light
pack
salt
feel
seem

BOW

In [87]:

```
count_vect = CountVectorizer(min_df=10) #in scikit-learn
vec = count_vect.fit(X_train['CleanedText'].values)
```

In [88]:

```
X_trvec = vec.transform(X_train['CleanedText'].values)
X_testvec = vec.transform(X_test['CleanedText'].values)
```

In [89]:

```
from sklearn.preprocessing import StandardScaler

std_svm=StandardScaler(with_mean=False).fit(X_trvec)
X_trvecs=std_svm.transform(X_trvec)
X_testvecs=std_svm.transform(X_testvec)
```

```
C:\Users\krush\Anaconda3\lib\site-packages\sklearn\utils\validation.py:
475: DataConversionWarning: Data with input dtype int64 was converted t
o float64 by StandardScaler.
  warnings.warn(msg, DataConversionWarning)
C:\Users\krush\Anaconda3\lib\site-packages\sklearn\utils\validation.py:
475: DataConversionWarning: Data with input dtype int64 was converted t
o float64 by StandardScaler.
  warnings.warn(msg, DataConversionWarning)
C:\Users\krush\Anaconda3\lib\site-packages\sklearn\utils\validation.py:
475: DataConversionWarning: Data with input dtype int64 was converted t
o float64 by StandardScaler.
  warnings.warn(msg, DataConversionWarning)
```

Linear SVC

Here we are using SGD Classifier as it is similar to Linear SVC

In [90]:

```
from sklearn.metrics import roc_auc_score
from sklearn.model_selection import GridSearchCV, cross_val_score
from sklearn import preprocessing
from sklearn import linear_model
tuned_parameters = [{'C': [10**-4, 10**-2, 10**0, 10**2, 10**4]}]

hyperparameter = dict(alpha=[10**-4, 10**-2, 10**0, 10**2, 10**4],penal
ty=['l1','l2'])
```

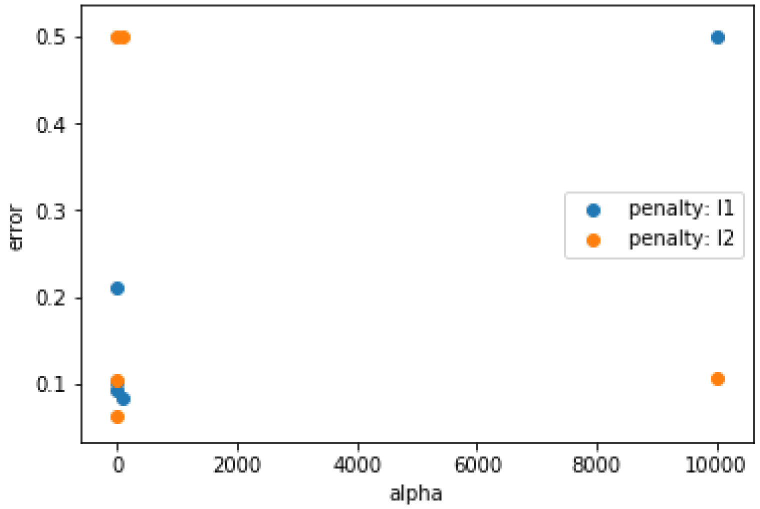
```
#Using GridSearchCV
model = GridSearchCV(linear_model.SGDClassifier(class_weight='balanced'
), hyperparameter, scoring = 'roc_auc', cv=5)
model.fit(X_trvecs, y_train)

print(model.best_estimator_)
print(model.score(X_testvecs, y_test))
cross_val_score(model,X_trvecs,y_train,cv=5 )
```

```
SGDClassifier(alpha=1, average=False, class_weight='balanced', epsilon=
0.1,
       eta0=0.0, fit_intercept=True, l1_ratio=0.15,
       learning_rate='optimal', loss='hinge', max_iter=None, n_iter=Non
e,
       n_jobs=1, penalty='l2', power_t=0.5, random_state=None,
       shuffle=True, tol=None, verbose=0, warm_start=False)
0.9409624910429286
```

```
Out[90]: array([0.93505258, 0.9410865 , 0.94077335, 0.93335846, 0.93794759])
```

```
In [91]: scores = [x[1] for x in model.grid_scores_]
scores = np.array(scores).reshape(len(hyperparameter['penalty']), len(h
yperparameter['alpha']))
for ind, i in enumerate(hyperparameter['penalty']):
    plt.scatter(hyperparameter['alpha'],1- scores[ind], label='penalty:
 ' + str(i))
plt.legend()
plt.xlabel('alpha')
plt.ylabel('error')
plt.show()
```

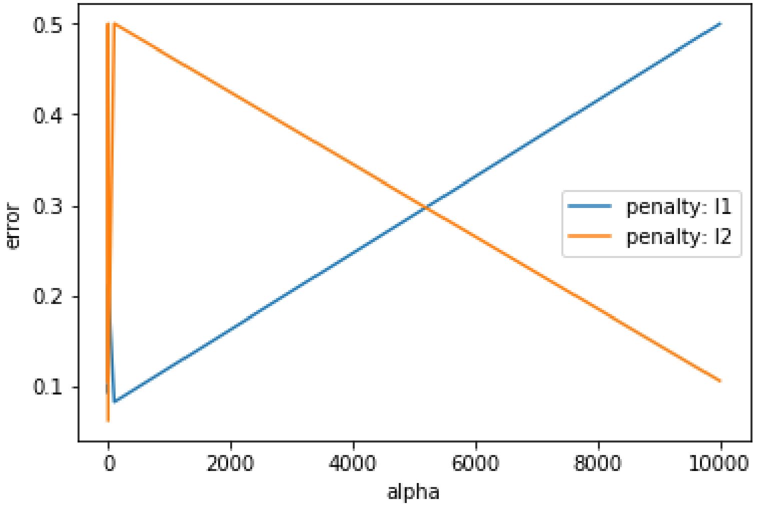
```
C:\Users\krush\Anaconda3\lib\site-packages\sklearn\model_selection\_sea
rch.py:761: DeprecationWarning: The grid_scores_ attribute was deprecat
ed in version 0.18 in favor of the more elaborate cv_results_ attribut
e. The grid_scores_ attribute will not be available from 0.20
  DeprecationWarning)
```



In [92]:

```
scores = [x[1] for x in model.grid_scores_]
scores = np.array(scores).reshape(len(hyperparameter['penalty']), len(h
yperparameter['alpha']))
for ind, i in enumerate(hyperparameter['penalty']):
    plt.plot(hyperparameter['alpha'],1- scores[ind], label='penalty: '
+ str(i))
plt.legend()
plt.xlabel('alpha')
plt.ylabel('error')
plt.show()
```

```
C:\Users\krush\Anaconda3\lib\site-packages\sklearn\model_selection\_sea
rch.py:761: DeprecationWarning: The grid_scores_ attribute was deprecat
ed in version 0.18 in favor of the more elaborate cv_results_ attribut
e. The grid_scores_ attribute will not be available from 0.20
  DeprecationWarning)
```



```
In [93]: print(float(1-model.score(X_testvecs, y_test)))
```

```
0.05903750895707138
```

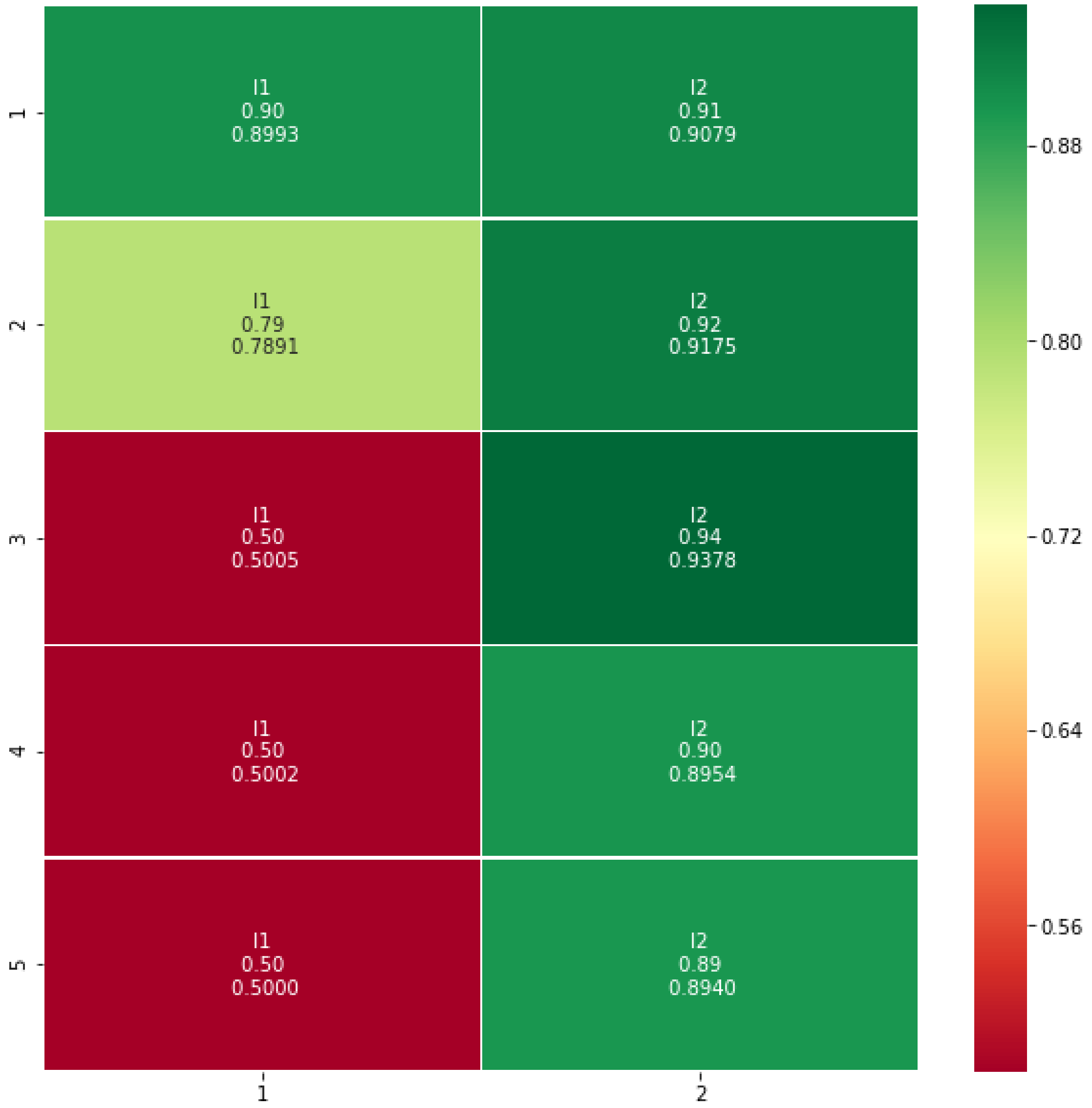
```
In [94]: alpha=[]
         pen=[]
         mean=[]
         for a in model.grid_scores_:
             alpha.append(a[0]['alpha'])
             pen.append(a[0]['penalty'])
             mean.append(a[1])
         alpha=np.asarray(alpha)
         pen=np.asarray(pen)
         mean=np.asarray(mean)
         alpha = alpha.reshape(5,2)
         pen = pen.reshape(5,2)
         mean = mean.reshape(5,2)
         result = pd.DataFrame(mean,index=[1,2,3,4,5],columns = [1,2])

         label =np.asarray([" {0} \n {1:.2f} \n {1:.4f}".format(pen,mean,alpha)
         for pen,mean,alpha in zip(pen.flatten(),mean.flatten(),alpha.flatten
         ())]).reshape(5,2)
```

```
print(label)

import seaborn as sns
fig , ax = plt.subplots(figsize = (10,10))
ax.set_xticks([])
ax.set_yticks([])
sns.heatmap(result,annot=label,fmt="" ,cmap ='RdYlGn',linewidths = 0.30
 , ax = ax )
```

```
[[' l1 \n 0.90 \n 0.8993' ' l2 \n 0.91 \n 0.9079']
 [' l1 \n 0.79 \n 0.7891' ' l2 \n 0.92 \n 0.9175']
 [' l1 \n 0.50 \n 0.5005' ' l2 \n 0.94 \n 0.9378']
 [' l1 \n 0.50 \n 0.5002' ' l2 \n 0.90 \n 0.8954']
 [' l1 \n 0.50 \n 0.5000' ' l2 \n 0.89 \n 0.8940']]
```

```
C:\Users\krush\Anaconda3\lib\site-packages\sklearn\model_selection\_sea
rch.py:761: DeprecationWarning: The grid_scores_ attribute was deprecat
ed in version 0.18 in favor of the more elaborate cv_results_ attribut
e. The grid_scores_ attribute will not be available from 0.20
  DeprecationWarning)
```

```
Out[94]: <matplotlib.axes._subplots.AxesSubplot at 0x2d517e211d0>
```

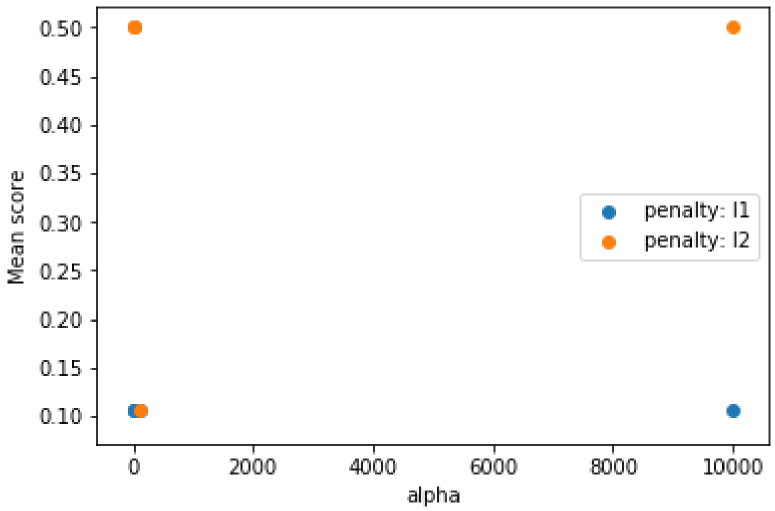


In [100]:
```
from sklearn import preprocessing
import scipy.stats as stats

tuned_parameters = dict(alpha= np.random.uniform(low=2**-9, high=2**9,
size=10),penalty=['l1','l2'])
#lb = preprocessing.LabelBinarizer()
#y_train = np.array([number[0] for number in lb.fit_transform(y_trai
n)])
#Using RandomizedSearchCV
modelr = RandomizedSearchCV(linear_model.SGDClassifier(class_weight='ba
lanced'),tuned_parameters,n_iter=10 ,scoring = 'roc_auc', cv=5)
modelr.fit(X_trvecs, y_train)
#y_test = np.array([number[0] for number in lb.fit_transform(y_test)])
print(modelr.best_estimator_)
print(modelr.score(X_testvecs, y_test))
```

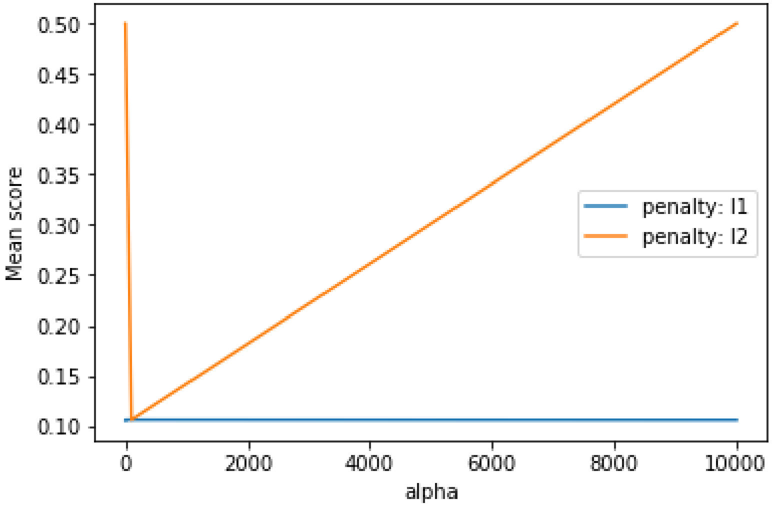
```
SGDClassifier(alpha=116.44286016427347, average=False,
       class_weight='balanced', epsilon=0.1, eta0=0.0, fit_intercept=Tr
ue,
       l1_ratio=0.15, learning_rate='optimal', loss='hinge', max_iter=N
one,
       n_iter=None, n_jobs=1, penalty='l2', power_t=0.5, random_state=N
one,
       shuffle=True, tol=None, verbose=0, warm_start=False)
0.89476406748746
```

In [101]:

```
scores = [x[1] for x in modelr.grid_scores_]
scores = np.array(scores).reshape(len(hyperparameter['penalty']), len(h
yperparameter['alpha']))
for ind, i in enumerate(hyperparameter['penalty']):
    plt.scatter(hyperparameter['alpha'],1-scores[ind], label='penalty:
 ' + str(i))
plt.legend()
plt.xlabel('alpha')
plt.ylabel('Mean score')
plt.show()
```



In [102]:
```
scores = [x[1] for x in modelr.grid_scores_]
scores = np.array(scores).reshape(len(hyperparameter['penalty']), len(hyperparameter['alpha']))
for ind, i in enumerate(hyperparameter['penalty']):
    plt.plot(hyperparameter['alpha'],1-scores[ind], label='penalty: ' + str(i))
plt.legend()
plt.xlabel('alpha')
plt.ylabel('Mean score')
plt.show()
```



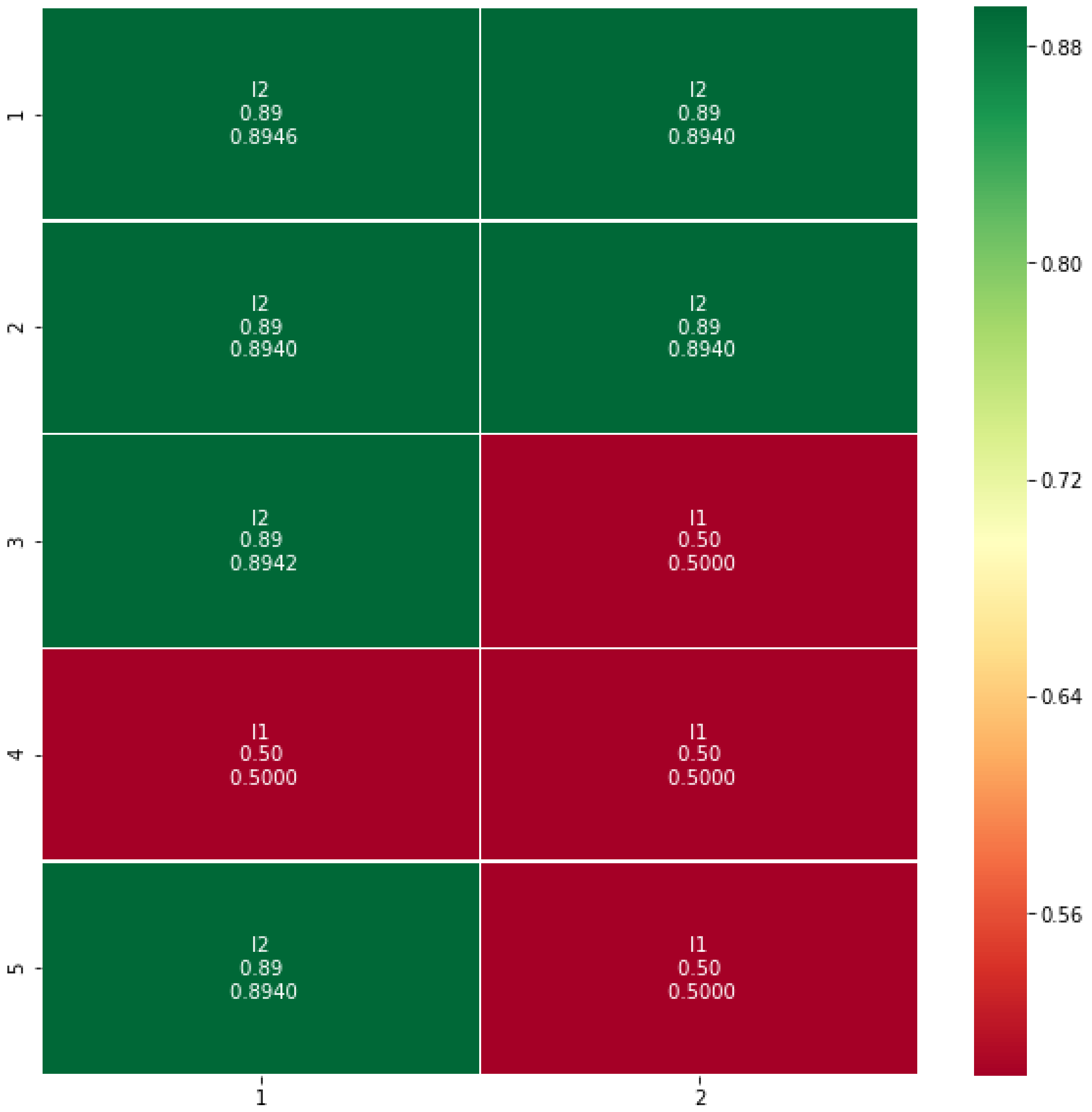
In [103]:

```
alpha=[]
pen=[]
mean=[]
for a in modelr.grid_scores_:
    alpha.append(a[0]['alpha'])
    pen.append(a[0]['penalty'])
    mean.append(a[1])
alpha=np.asarray(alpha)
pen=np.asarray(pen)
mean=np.asarray(mean)
alpha = alpha.reshape(5,2)
pen = pen.reshape(5,2)
mean = mean.reshape(5,2)
result = pd.DataFrame(mean,index=[1,2,3,4,5],columns = [1,2])

label =np.asarray([" {0} \n {1:.2f} \n {1:.4f}".format(pen,mean,alpha)
for pen,mean,alpha in zip(pen.flatten(),mean.flatten(),alpha.flatten
())]).reshape(5,2)
print(label)

import seaborn as sns
fig , ax = plt.subplots(figsize = (10,10))
```

```python
ax.set_xticks([])
ax.set_yticks([])
sns.heatmap(result,annot=label,fmt="" ,cmap ='RdYlGn',linewidths = 0.30
 , ax = ax )
```

```
[[' l2 \n 0.89 \n 0.8946' ' l2 \n 0.89 \n 0.8940']
 [' l2 \n 0.89 \n 0.8940' ' l2 \n 0.89 \n 0.8940']
 [' l2 \n 0.89 \n 0.8942' ' l1 \n 0.50 \n 0.5000']
 [' l1 \n 0.50 \n 0.5000' ' l1 \n 0.50 \n 0.5000']
 [' l2 \n 0.89 \n 0.8940' ' l1 \n 0.50 \n 0.5000']]
```

```
Out[103]: <matplotlib.axes._subplots.AxesSubplot at 0x2d51a1b3b00>
```



Feature Importance

In [20]:

```
modeln1 =linear_model.SGDClassifier(alpha=1, average=False, class_weight='balanced', epsilon=0.1,
       eta0=0.0, fit_intercept=True, l1_ratio=0.15,
       learning_rate='optimal', loss='hinge', max_iter=None, n_iter=None,
       n_jobs=1, penalty='l2', power_t=0.5, random_state=None,
       shuffle=True, tol=None, verbose=0, warm_start=False)
modeln1.fit(X_trvecs, y_train)
pred = modeln1.predict(X_trvecs)
def important_features(vectorizer,classifier,n=10):
    class_labels = classifier.classes_
    print(class_labels)
    feature_names =vectorizer.get_feature_names()
    classifier.coef_
    topn_class1 = sorted(zip(pred,classifier.coef_[0], feature_names),reverse=True)
    coef1 = classifier.coef_
    #print(coef1.shape)
    #topn_class2 = sorted(zip(classifier.coef_[1], feature_names),reverse=True)[:n]
    #coef2 = classifier.coef_[1]
    #print(topn_class1)

    count=0
    count1=0
    print("Important words in positive reviews")
    for class1,coef,feat in topn_class1:
        if class1 == 1 and count<=n:
            print( class1,coef, feat)
            count=count+1
        if count == 20:
            break
    print("Important words in negative reviews")
    for class1,coef,feat in topn_class1:
        if class1 == 0 and count1<=n:
            print( class1,coef, feat)
            count1=count1+1
        if count1 == 20:
            break
    #return coef1
    #for coef, feat in topn_class2:
     #   print( coef, feat)
    #return coef2
important_features(vec,modeln1,n=20)
```

```
[0 1]
Important words in positive reviews
1 0.10800672215073107 great
1 0.08629165319237286 love
1 0.08622789191911029 best
1 0.06314915954614025 delici
1 0.04980419314012348 perfect
1 0.0472781602437701 good
1 0.046754249517012784 nice
1 0.04412185632958524 find
1 0.04252720921330362 wonder
1 0.0391837885469243 easi
1 0.03825218476214582 tasti
1 0.03627535600155874 keep
1 0.035965017969561974 snack
1 0.033658962577258957 alway
1 0.03313875223840117 use
1 0.0328914985618877 amaz
1 0.031963948556065214 addict
1 0.031955955625261906 right
1 0.030944761529493645 glad
1 0.02944263801631547 year
Important words in negative reviews
0 0.05075102591260754 favorit
0 0.048278620097382235 excel
0 0.039027258924891305 thank
0 0.03433773476182155 fast
0 0.029474838059943224 high
0 0.026363547359044212 quit
0 0.025794128303325472 satisfi
0 0.023055804589395316 littl
0 0.022756743803946544 light
0 0.01928827196981785 spici
0 0.01890179316650672 sometim
```

```
0 0.018235822681456015 balanc
0 0.01818153740630166 chip
0 0.01752426643903926 start
0 0.01723522596572356 goe
0 0.01603657704386417 lay
0 0.01582785910557442 easili
0 0.015488107062358798 wow
0 0.015101981133438403 havent
0 0.015004102082235153 found
```

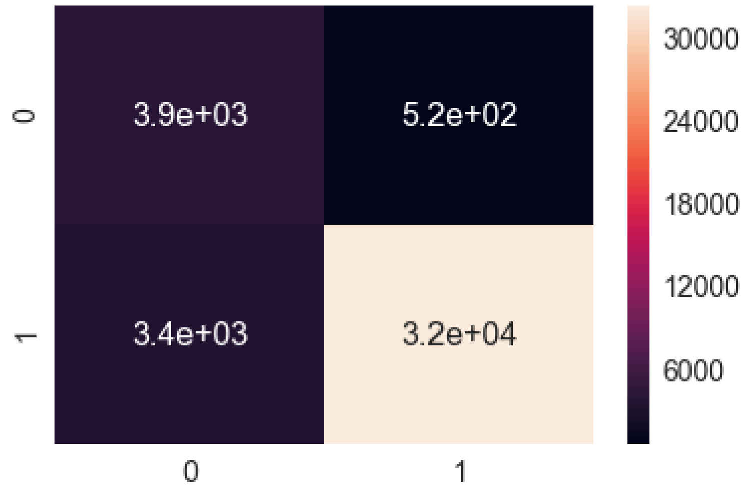
```
In [21]: modell2 = linear_model.SGDClassifier(alpha=1, average=False, class_weight='balanced', epsilon=0.1,
                eta0=0.0, fit_intercept=True, l1_ratio=0.15,
                learning_rate='optimal', loss='hinge', max_iter=None, n_iter=None,
                n_jobs=1, penalty='l2', power_t=0.5, random_state=None,
                shuffle=True, tol=None, verbose=0, warm_start=False)
         modell2.fit(X_trvecs, y_train)
         print(modell2.score(X_testvecs, y_test))
         print("error",1-modell2.score(X_testvecs, y_test))
```

```
0.8923
error 0.10770000000000002
```

Confusion Matrix

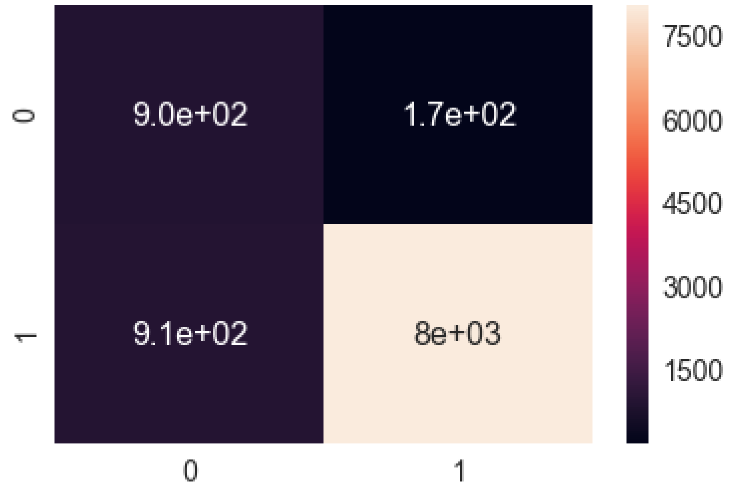
```
In [22]: pred = modell2.predict(X_testvecs)
         pred1 = modell2.predict(X_trvecs)
         from sklearn.metrics import confusion_matrix
         import seaborn as sn
         CFMt = confusion_matrix(y_train, pred1)
         CFM = confusion_matrix(y_test, pred)
         df_cm = pd.DataFrame(CFMt, range(2), range(2))
         #plt.figure(figsize = (10,7))
         sn.set(font_scale=1.4)#for label size
         sn.heatmap(df_cm, annot=True,annot_kws={"size": 16})
```

```
Out[22]: <matplotlib.axes._subplots.AxesSubplot at 0x1b0035c6828>
```



```
In [23]: df_cm = pd.DataFrame(CFM, range(2), range(2))
         #plt.figure(figsize = (10,7))
         sn.set(font_scale=1.4)#for label size
         sn.heatmap(df_cm, annot=True,annot_kws={"size": 16})
```

```
Out[23]: <matplotlib.axes._subplots.AxesSubplot at 0x1b003638320>
```



ALL Other metrics

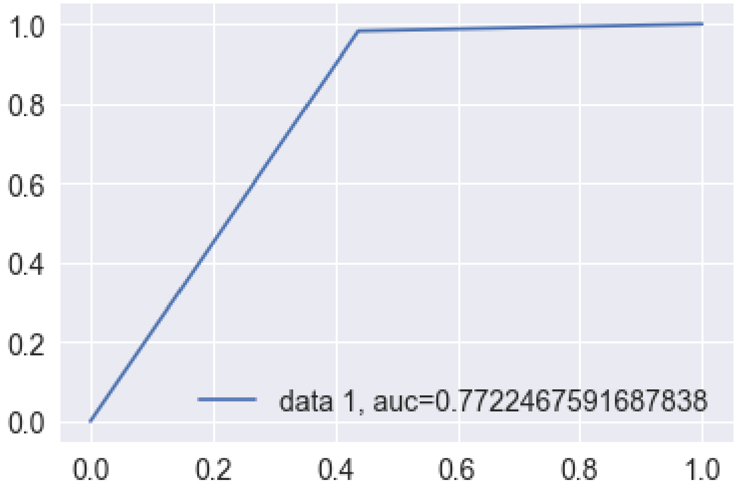
In [24]:

```
from sklearn.metrics import accuracy_score, f1_score, precision_score,
recall_score
print(accuracy_score(y_test, pred))
print(f1_score(y_test, pred, average="macro"))
print(precision_score(y_test, pred, average="macro"))
print(recall_score(y_test, pred, average="macro"))
```

```
0.8923
0.7820068891085642
0.7393430163778145
0.8701179076281675
```

In [127]:

```
from sklearn import metrics
from sklearn import calibration
model1 = calibration.CalibratedClassifierCV(base_estimator=model, metho
d='sigmoid')
model1.fit(X_trvecs, y_train)
pred = pred = model1.predict(X_testvecs)
fpr, tpr, _ = metrics.roc_curve(y_test,pred)
auc = metrics.roc_auc_score(y_test, pred)
plt.plot(fpr,tpr,label="data 1, auc="+str(auc))
plt.legend(loc=4)
plt.show()
```



Dimensionality reductioan as it is said to do as much as you can for SVC as it dont work effectively for hugh dimentions and we are using randomized search as grid search is brute force and take many parameters , so that it is very complicated to run.

In [128]:

```
from sklearn import preprocessing
from sklearn import svm
import scipy.stats as stats
tuned_parameters ={'C': np.random.uniform(low=10**-4, high=10**4, size=
5),'gamma': np.random.uniform(low=10**-4, high=10**4, size=5)}
#lb = preprocessing.LabelBinarizer()
#y_train = np.array([number[0] for number in lb.fit_transform(y_trai
n)])
#Using RandomizedSearchCV
modelr1 = RandomizedSearchCV(svm.SVC(kernel='rbf',class_weight='balance
d'),tuned_parameters,n_iter=5 ,scoring = 'roc_auc', cv=5)
modelr1.fit(X_trvecs, y_train)
```

```
#y_test = np.array([number[0] for number in lb.fit_transform(y_test)])
print(modelr1.best_estimator_)
print(modelr1.score(X_testvecs, y_test))
```

```
SVC(C=7475.511493356914, cache_size=200, class_weight='balanced', coef0
=0.0,
  decision_function_shape='ovr', degree=3, gamma=8456.395185711095,
  kernel='rbf', max_iter=-1, probability=False, random_state=None,
  shrinking=True, tol=0.001, verbose=False)
0.8080191779037197
```

In [26]:

```
from sklearn import svm
modelsvc=svm.SVC(C=7475.511493356914, cache_size=200, class_weight='bal
anced', coef0=0.0,
  decision_function_shape='ovr', degree=3, gamma=8456.395185711095,
  kernel='rbf', max_iter=-1, probability=False, random_state=None,
  shrinking=True, tol=0.001, verbose=False)
modelsvc.fit(X_trvecs, y_train)
print(modelsvc.score(X_testvecs, y_test))
print("error=",float(1-modelsvc.score(X_testvecs, y_test)))
```

```
0.9336
error= 0.06640000000000001
```

In [139]:

```
tuned_parameters = [10**-4,10**-2, 10**0, 10**2,10**4]
ERROR=[]
for c in tuned_parameters:
    for gamma in tuned_parameters:
        print("for c= ", c)
        print("for gamma= ", gamma)
        modelsvc1 = svm.SVC(C=c, cache_size=200, class_weight='balance
d', coef0=0.0,
        decision_function_shape='ovr', degree=3, gamma=gamma,
        kernel='rbf', max_iter=-1, probability=False, random_state=None
,
        shrinking=True, tol=0.001, verbose=False)
        modelsvc1.fit(X_trvecs, y_train)
        f1_score=modelsvc1.score(X_testvecs, y_test)
        error = float(1-f1_score)
        print("error=",error)
        ERROR.append(error)
```

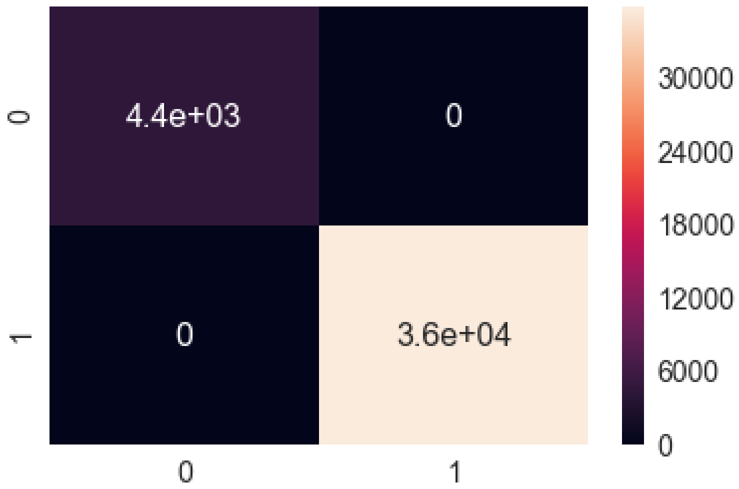
```
for c=  0.0001
for gamma=  0.0001
error= 0.8925
for c=  0.0001
for gamma=  0.01
error= 0.8925
for c=  0.0001
for gamma=  1
error= 0.8925
for c=  0.0001
for gamma=  100
error= 0.8925
for c=  0.0001
for gamma=  10000
error= 0.8925
for c=  0.01
for gamma=  0.0001
error= 0.15669999999999995
for c=  0.01
for gamma=  0.01
error= 0.8221
for c=  0.01
for gamma=  1
error= 0.5548
for c=  0.01
for gamma=  100
error= 0.5548
for c=  0.01
for gamma=  10000
error= 0.5548
for c=  1
for gamma=  0.0001
error= 0.0857
for c=  1
for gamma=  0.01
error= 0.06610000000000005
```

```
for c=  1
for gamma=  1
error= 0.06640000000000001
for c=  1
for gamma=  100
error= 0.06640000000000001
for c=  1
for gamma=  10000
error= 0.06640000000000001
for c=  100
for gamma=  0.0001
error= 0.06410000000000005
for c=  100
for gamma=  0.01
error= 0.06610000000000005
for c=  100
for gamma=  1
error= 0.06640000000000001
for c=  100
for gamma=  100
error= 0.06640000000000001
for c=  100
for gamma=  10000
error= 0.06640000000000001
for c=  10000
for gamma=  0.0001
error= 0.06969999999999998
for c=  10000
for gamma=  0.01
error= 0.06610000000000005
for c=  10000
for gamma=  1
error= 0.06640000000000001
for c=  10000
for gamma=  100
error= 0.06640000000000001
for c=  10000
for gamma=  10000
error= 0.06640000000000001
```

Feature importance is not done for svc because coef_ is only applicable for linear svc

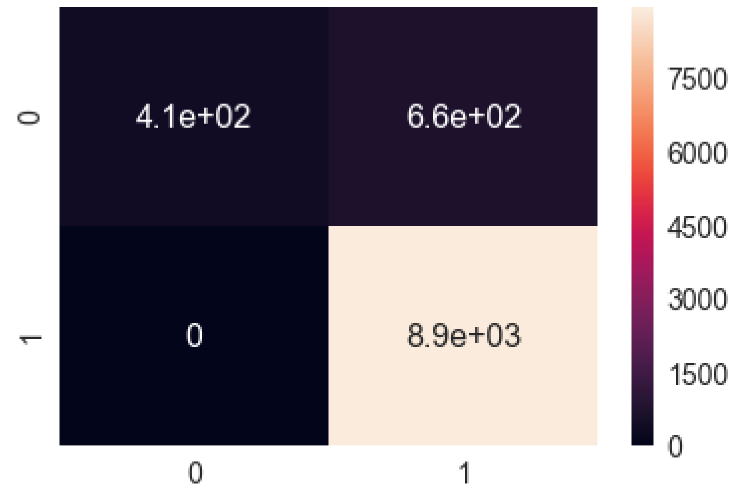
In [27]:
```
pred = modelsvc.predict(X_testvecs)
pred1 = modelsvc.predict(X_trvecs)
from sklearn.metrics import confusion_matrix
import seaborn as sn
CFMt = confusion_matrix(y_train, pred1)
CFM = confusion_matrix(y_test, pred)
df_cm = pd.DataFrame(CFMt, range(2), range(2))
#plt.figure(figsize = (10,7))
sn.set(font_scale=1.4)#for label size
sn.heatmap(df_cm, annot=True,annot_kws={"size": 16})
```

Out[27]:
```
<matplotlib.axes._subplots.AxesSubplot at 0x1b00375b550>
```



In [142]:
```
df_cm = pd.DataFrame(CFM, range(2), range(2))
#plt.figure(figsize = (10,7))
sn.set(font_scale=1.4)#for label size
sn.heatmap(df_cm, annot=True,annot_kws={"size": 16})
```

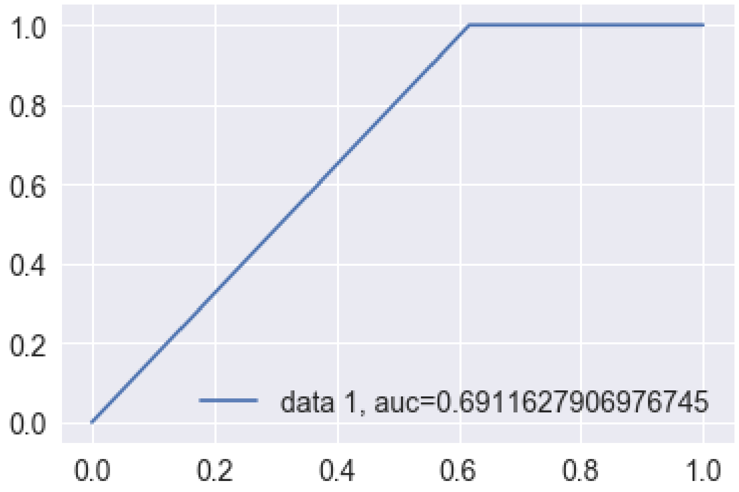
Out[142]:
```
<matplotlib.axes._subplots.AxesSubplot at 0x1fa182a4a20>
```



In [143]:
```
from sklearn.metrics import accuracy_score, f1_score, precision_score,
recall_score
print(accuracy_score(y_test, pred))
print(f1_score(y_test, pred, average="macro"))
print(precision_score(y_test, pred, average="macro"))
print(recall_score(y_test, pred, average="macro"))
```

```
0.9336
0.7586490511490995
0.9653769944728334
0.6911627906976744
```

In [144]:
```
from sklearn import metrics
from sklearn import calibration
model1 = calibration.CalibratedClassifierCV(base_estimator=modelsvc, me
thod='sigmoid')
model1.fit(X_trvecs, y_train)
pred = pred = model1.predict(X_testvecs)
fpr, tpr, _ = metrics.roc_curve(y_test,pred)
auc = metrics.roc_auc_score(y_test, pred)
plt.plot(fpr,tpr,label="data 1, auc="+str(auc))
plt.legend(loc=4)
plt.show()
```



TFIDF

In [133]:
```
tf_idf_vect = TfidfVectorizer(ngram_range=(1,2),min_df=10)
tf_idf_vect1 = TfidfVectorizer(ngram_range=(1,2))
final_tf_idf = tf_idf_vect.fit(X_train['CleanedText'].values)
final_tf_idf1 = tf_idf_vect1.fit(X_train['CleanedText'].values)
```

In [134]:
```
X_tr_tf_idf = final_tf_idf.transform(X_train['CleanedText'].values)
X_test_tf_idf = final_tf_idf.transform(X_test['CleanedText'].values)
```

In [135]:
```
X_tr_tf_idf.shape
```

Out[135]:
```
(40000, 26821)
```

In [136]:
```
from sklearn.preprocessing import StandardScaler

std_svm=StandardScaler(with_mean=False).fit(X_tr_tf_idf)
```

```
X_tr_tf_idfs=std_svm.transform(X_tr_tf_idf)
X_test_tf_idfs=std_svm.transform(X_test_tf_idf)
```

In [137]:

```
X_tr_tf_idfs.shape
```

Out[137]:

```
(40000, 26821)
```

In [138]:

```
from sklearn import preprocessing
from sklearn import linear_model
tuned_parameters = [{'C': [10**-4, 10**-2, 10**0, 10**2, 10**4]}]

hyperparameter = dict(alpha=[10**-4, 10**-2, 10**0, 10**2, 10**4],penal
ty=['l1','l2'])

#Using GridSearchCV
modeltf = GridSearchCV(linear_model.SGDClassifier(class_weight='balance
d'), hyperparameter, scoring = 'roc_auc', cv=5)
modeltf.fit(X_tr_tf_idfs, y_train)

print(modeltf.best_estimator_)
print(modeltf.score(X_test_tf_idfs, y_test))
```

```
SGDClassifier(alpha=1, average=False, class_weight='balanced', epsilon=
0.1,
       eta0=0.0, fit_intercept=True, l1_ratio=0.15,
       learning_rate='optimal', loss='hinge', max_iter=None, n_iter=Non
e,
       n_jobs=1, penalty='l2', power_t=0.5, random_state=None,
       shuffle=True, tol=None, verbose=0, warm_start=False)
0.9601608233991272
```

In [139]:

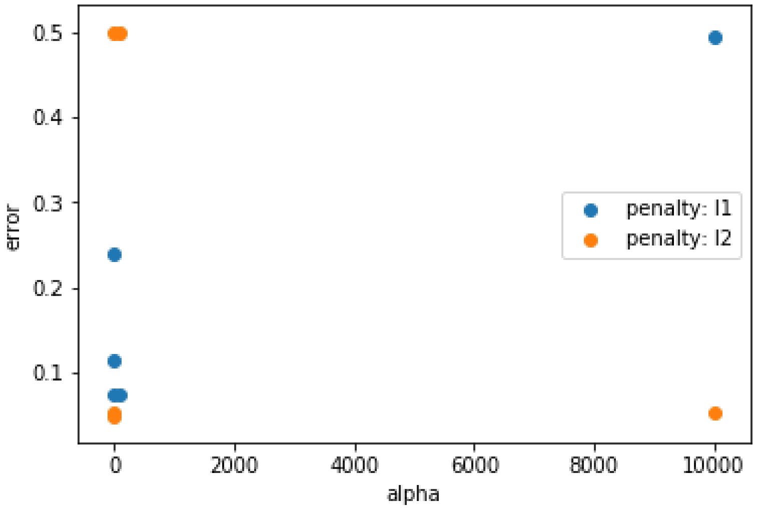
```
print(float(1-modeltf.score(X_test_tf_idfs, y_test)))
```

```
0.03983917660087277
```

In [140]:

```
scores = [x[1] for x in modeltf.grid_scores_]
scores = np.array(scores).reshape(len(hyperparameter['penalty']), len(h
yperparameter['alpha']))
for ind, i in enumerate(hyperparameter['penalty']):
    plt.scatter(hyperparameter['alpha'],1- scores[ind], label='penalty:
  ' + str(i))
plt.legend()
plt.xlabel('alpha')
plt.ylabel('error')
plt.show()
```

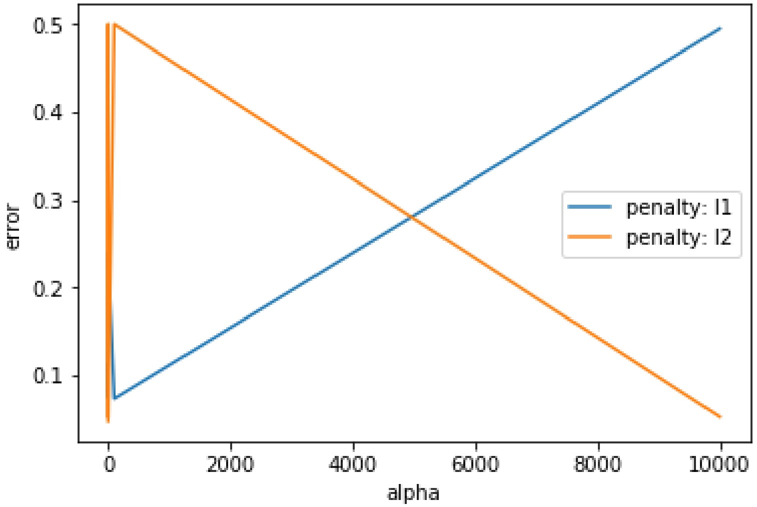
```
C:\Users\krush\Anaconda3\lib\site-packages\sklearn\model_selection\_sea
rch.py:761: DeprecationWarning: The grid_scores_ attribute was deprecat
ed in version 0.18 in favor of the more elaborate cv_results_ attribut
e. The grid_scores_ attribute will not be available from 0.20
  DeprecationWarning)
```



In [141]:

```
scores = [x[1] for x in modeltf.grid_scores_]
scores = np.array(scores).reshape(len(hyperparameter['penalty']), len(h
yperparameter['alpha']))
for ind, i in enumerate(hyperparameter['penalty']):
    plt.plot(hyperparameter['alpha'],1- scores[ind], label='penalty: '
+ str(i))
plt.legend()
plt.xlabel('alpha')
plt.ylabel('error')
plt.show()
```

```
C:\Users\krush\Anaconda3\lib\site-packages\sklearn\model_selection\_sea
rch.py:761: DeprecationWarning: The grid_scores_ attribute was deprecat
ed in version 0.18 in favor of the more elaborate cv_results_ attribut
e. The grid_scores_ attribute will not be available from 0.20
  DeprecationWarning)
```



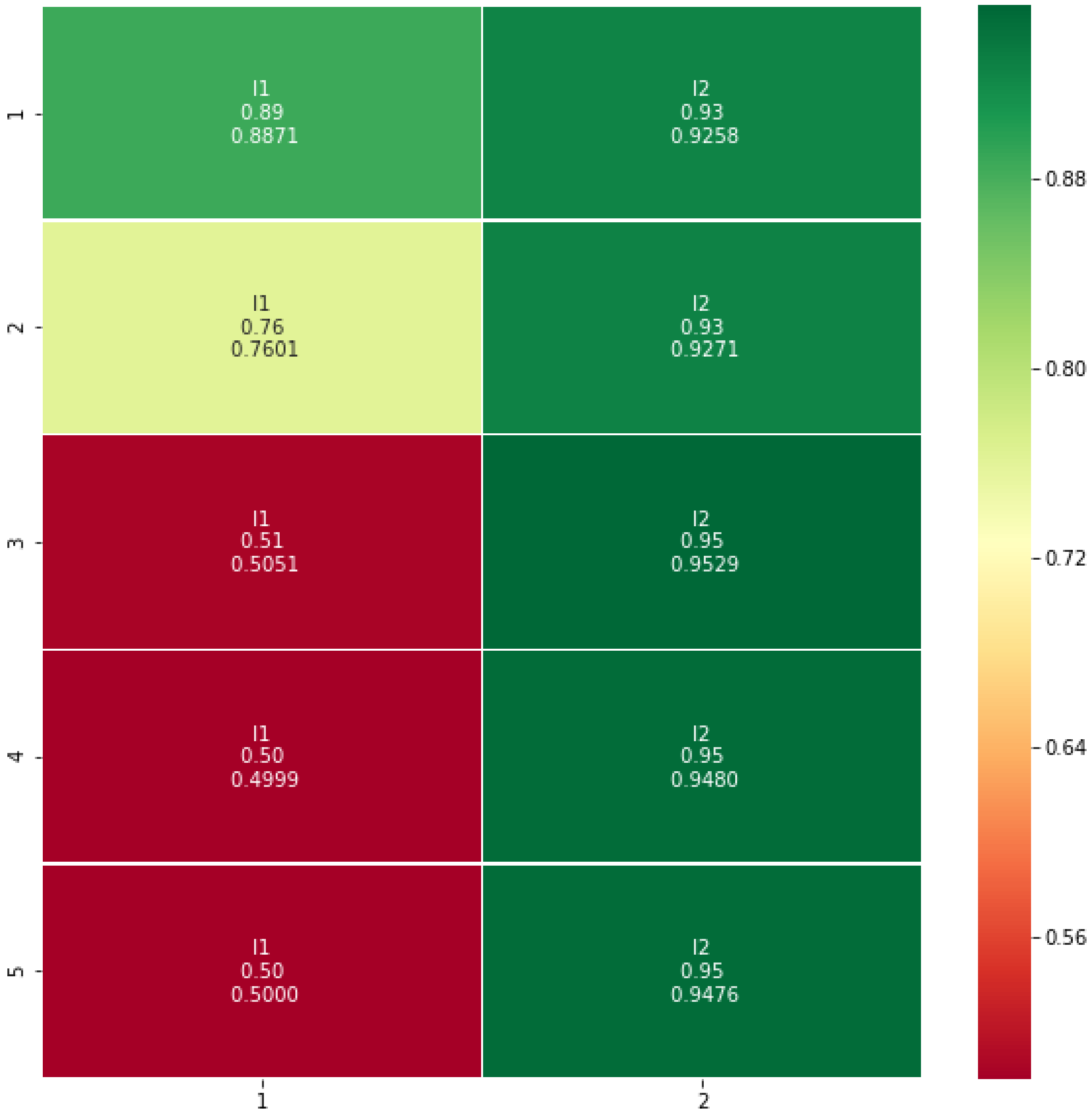
In [142]:

```
alpha=[]
pen=[]
mean=[]
for a in modeltf.grid_scores_:
    alpha.append(a[0]['alpha'])
    pen.append(a[0]['penalty'])
    mean.append(a[1])
alpha=np.asarray(alpha)
pen=np.asarray(pen)
mean=np.asarray(mean)
alpha = alpha.reshape(5,2)
pen = pen.reshape(5,2)
mean = mean.reshape(5,2)
result = pd.DataFrame(mean,index=[1,2,3,4,5],columns = [1,2])

label =np.asarray([" {0} \n {1:.2f} \n {1:.4f}".format(pen,mean,alpha)
for pen,mean,alpha in zip(pen.flatten(),mean.flatten(),alpha.flatten
())]).reshape(5,2)
print(label)

import seaborn as sns
fig , ax = plt.subplots(figsize = (10,10))
ax.set_xticks([])
ax.set_yticks([])
sns.heatmap(result,annot=label,fmt="" ,cmap ='RdYlGn',linewidths = 0.30
 , ax = ax )
```

```
C:\Users\krush\Anaconda3\lib\site-packages\sklearn\model_selection\_sea
rch.py:761: DeprecationWarning: The grid_scores_ attribute was deprecat
ed in version 0.18 in favor of the more elaborate cv_results_ attribut
e. The grid_scores_ attribute will not be available from 0.20
  DeprecationWarning)
```

```
[[' l1 \n 0.89 \n 0.8871' ' l2 \n 0.93 \n 0.9258']
 [' l1 \n 0.76 \n 0.7601' ' l2 \n 0.93 \n 0.9271']
 [' l1 \n 0.51 \n 0.5051' ' l2 \n 0.95 \n 0.9529']
 [' l1 \n 0.50 \n 0.4999' ' l2 \n 0.95 \n 0.9480']
 [' l1 \n 0.50 \n 0.5000' ' l2 \n 0.95 \n 0.9476']]
```

```
Out[142]: <matplotlib.axes._subplots.AxesSubplot at 0x2d5484c3048>
```



In [143]:

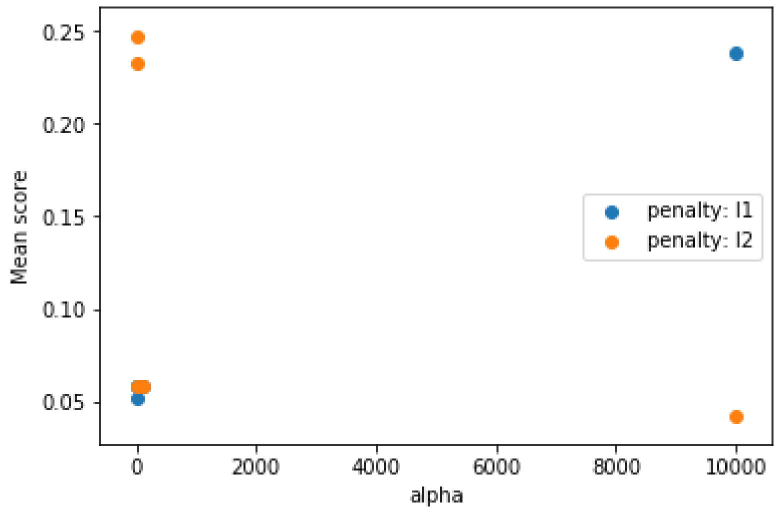
```
from sklearn import preprocessing
import scipy.stats as stats

tuned_parameters = dict(alpha = np.random.uniform(low=10**-4, high=10**
4, size=10),penalty=['l1','l2'])
#lb = preprocessing.LabelBinarizer()
#y_train = np.array([number[0] for number in lb.fit_transform(y_trai
n)])
#Using RandomizedSearchCV
modeltfr = RandomizedSearchCV(linear_model.SGDClassifier(class_weight=
'balanced'),tuned_parameters,n_iter=10 ,scoring = 'f1', cv=5)
modeltfr.fit(X_tr_tf_idfs, y_train)
#y_test = np.array([number[0] for number in lb.fit_transform(y_test)])
print(modeltfr.best_estimator_)
print(modeltfr.score(X_test_tf_idfs, y_test))
```

```
C:\Users\krush\Anaconda3\lib\site-packages\sklearn\metrics\classificati
on.py:1135: UndefinedMetricWarning: F-score is ill-defined and being se
t to 0.0 due to no predicted samples.
  'precision', 'predicted', average, warn_for)
```

```
SGDClassifier(alpha=8501.517532469872, average=False, class_weight='bal
anced',
       epsilon=0.1, eta0=0.0, fit_intercept=True, l1_ratio=0.15,
       learning_rate='optimal', loss='hinge', max_iter=None, n_iter=Non
e,
       n_jobs=1, penalty='l2', power_t=0.5, random_state=None,
       shuffle=True, tol=None, verbose=0, warm_start=False)
0.9574722459436379
```

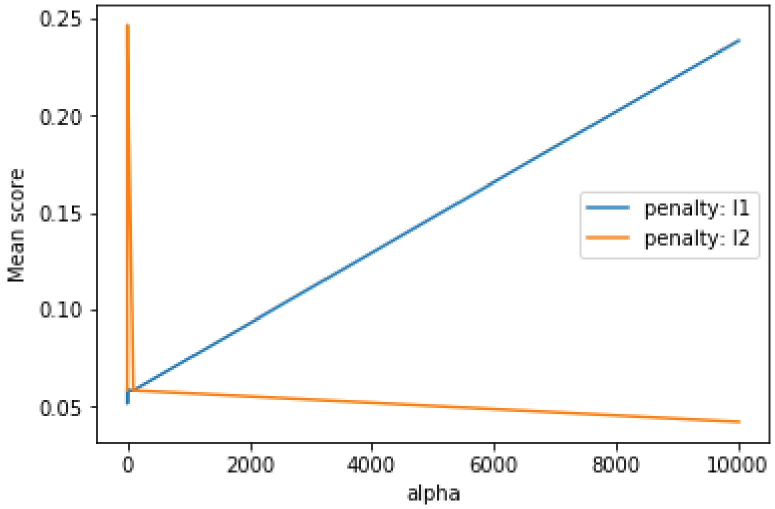
In [150]:

```
scores = [x[1] for x in modeltfr.grid_scores_]
scores = np.array(scores).reshape(len(hyperparameter['penalty']), len(h
yperparameter['alpha']))
for ind, i in enumerate(hyperparameter['penalty']):
    plt.scatter(hyperparameter['alpha'],1-scores[ind], label='penalty:
 ' + str(i))
plt.legend()
plt.xlabel('alpha')
plt.ylabel('Mean score')
plt.show()
```



In [151]:

```
scores = [x[1] for x in modeltfr.grid_scores_]
scores = np.array(scores).reshape(len(hyperparameter['penalty']), len(hyperparameter['alpha']))
for ind, i in enumerate(hyperparameter['penalty']):
    plt.plot(hyperparameter['alpha'],1-scores[ind], label='penalty: ' + str(i))
plt.legend()
plt.xlabel('alpha')
plt.ylabel('Mean score')
plt.show()
```



In [152]:
```
modeltfr.grid_scores_
```

Out[152]:
```
[mean: 0.94174, std: 0.00002, params: {'penalty': 'l1', 'alpha': 8453.1
57608092422},
 mean: 0.94852, std: 0.00301, params: {'penalty': 'l2', 'alpha': 6255.5
16337490452},
 mean: 0.94174, std: 0.00002, params: {'penalty': 'l1', 'alpha': 5601.7
720045479045},
 mean: 0.94174, std: 0.00002, params: {'penalty': 'l1', 'alpha': 8501.5
17532469872},
 mean: 0.76153, std: 0.38034, params: {'penalty': 'l2', 'alpha': 6980.4
70359332149},
 mean: 0.94174, std: 0.00002, params: {'penalty': 'l1', 'alpha': 8078.1
81137587298},
 mean: 0.75340, std: 0.37670, params: {'penalty': 'l1', 'alpha': 1116.3
893678264146},
 mean: 0.76710, std: 0.38272, params: {'penalty': 'l2', 'alpha': 8453.1
57608092422},
 mean: 0.94174, std: 0.00002, params: {'penalty': 'l1', 'alpha': 6980.4
70359332149},
 mean: 0.95801, std: 0.00223, params: {'penalty': 'l2', 'alpha': 8501.5
17532469872}]
```

In [153]:

```
alpha=[]
pen=[]
mean=[]
for a in modeltfr.grid_scores_:
    alpha.append(a[0]['alpha'])
    pen.append(a[0]['penalty'])
    mean.append(a[1])
alpha=np.asarray(alpha)
pen=np.asarray(pen)
mean=np.asarray(mean)
alpha = alpha.reshape(5,2)
pen = pen.reshape(5,2)
mean = mean.reshape(5,2)
result = pd.DataFrame(mean,index=[1,2,3,4,5],columns = [1,2])
```

In [154]:

```
label =np.asarray([" {0} \n {1:.2f} \n {1:.4f}".format(pen,mean,alpha)
for pen,mean,alpha in zip(pen.flatten(),mean.flatten(),alpha.flatten
())]).reshape(5,2)
print(label)
```

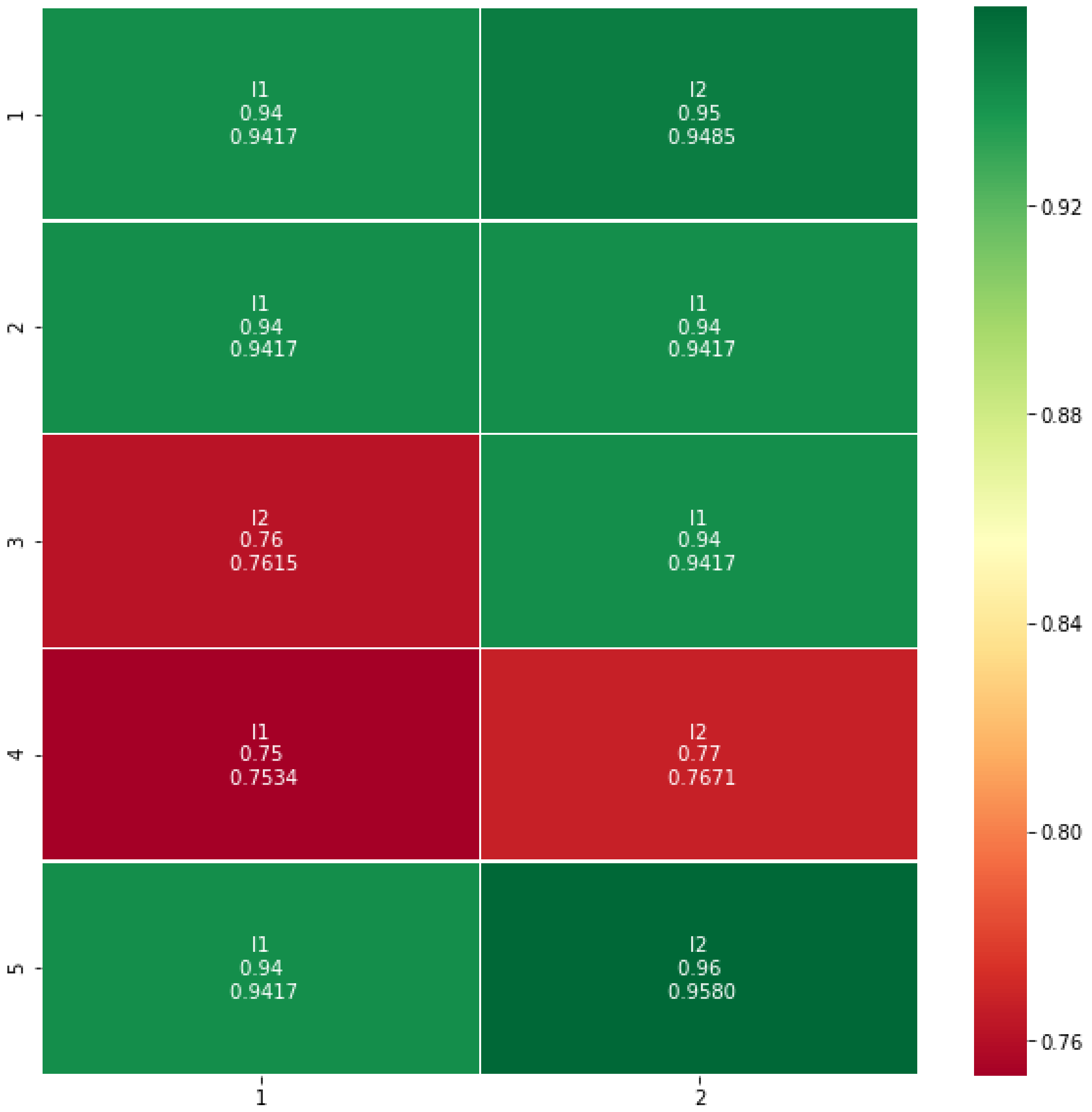
```
[[' l1 \n 0.94 \n 0.9417' ' l2 \n 0.95 \n 0.9485']
 [' l1 \n 0.94 \n 0.9417' ' l1 \n 0.94 \n 0.9417']
 [' l2 \n 0.76 \n 0.7615' ' l1 \n 0.94 \n 0.9417']
 [' l1 \n 0.75 \n 0.7534' ' l2 \n 0.77 \n 0.7671']
 [' l1 \n 0.94 \n 0.9417' ' l2 \n 0.96 \n 0.9580']]
```

In [155]:

```
import seaborn as sns
fig , ax = plt.subplots(figsize = (10,10))
ax.set_xticks([])
ax.set_yticks([])
sns.heatmap(result,annot=label,fmt="" ,cmap ='RdYlGn',linewidths = 0.30
 , ax = ax )
```

Out[155]:

```
<matplotlib.axes._subplots.AxesSubplot at 0x2d52c4575f8>
```



In [41]:

```
modeln5 = linear_model.SGDClassifier(alpha=1, average=False, class_weig
ht='balanced', epsilon=0.1,
       eta0=0.0, fit_intercept=True, l1_ratio=0.15,
       learning_rate='optimal', loss='hinge', max_iter=None, n_iter=Non
e,
       n_jobs=1, penalty='l2', power_t=0.5, random_state=None,
       shuffle=True, tol=None, verbose=0, warm_start=False)
modeln5.fit(X_tr_tf_idfs, y_train)
pred = modeln5.predict(X_tr_tf_idfs)
def important_features(vectorizer,classifier,n=20):
    class_labels = classifier.classes_
    print(class_labels)
    feature_names =vectorizer.get_feature_names()
    classifier.coef_
    topn_class1 = sorted(zip(pred,classifier.coef_[0], feature_names),r
everse=True)
    coef1 = classifier.coef_
    #print(coef1.shape)
    #topn_class2 = sorted(zip(classifier.coef_[1], feature_names),rever
se=True)[:n]
    #coef2 = classifier.coef_[1]
    #print(topn_class1)

    count=0
    count1=0
    print("Important words in positive reviews")
    for class1,coef,feat in topn_class1:
        if class1 == 1 and count<=n:
            print( class1,coef, feat)
            count=count+1
        if count == 20:
            break
    print("Important words in negative reviews")
    for class1,coef,feat in topn_class1:
        if class1 == 0 and count1<=n:
            print( class1,coef, feat)
            count1=count1+1
        if count1 == 20:
            break
    #return coef1
    #for coef, feat in topn_class2:
     #   print( coef, feat)
    #return coef2
important_features(final_tf_idf,modeln5,n=20)
```

```
[0 1]
Important words in positive reviews
1 0.0778945791282877 great
1 0.07315479097367088 best
1 0.06947596643256357 love
1 0.05352364259889439 delici
1 0.04420789114706366 excel
1 0.043319827407037416 perfect
1 0.04268367618940267 favorit
1 0.040895874232618756 good
1 0.037741282609970755 wonder
1 0.03675620931707281 use
1 0.035788253781169045 find
1 0.03388129253836104 keep
1 0.03349587896362717 thank
1 0.031570654338943946 amaz
1 0.030607417060253318 addict
1 0.03016493298527238 quick
1 0.02842460854561192 easi
1 0.027865016499877236 alway
1 0.02736717186107623 fast
1 0.02622095541001253 great product
Important words in negative reviews
0 0.04285489050664508 nice
0 0.034343466102948264 high recommend
0 0.03387764317205727 tasti
0 0.0318018444460152 snack
0 0.027406749666413863 right
0 0.02460950448803967 year
0 0.02351470978596911 friend
0 0.023295931106548935 light
0 0.02244690014187829 along
0 0.022165025213249978 great tast
0 0.02096394876665235 daughter
0 0.02062435097146422 need
0 0.020317802907540945 conveni
0 0.019813517562983194 hard find
```

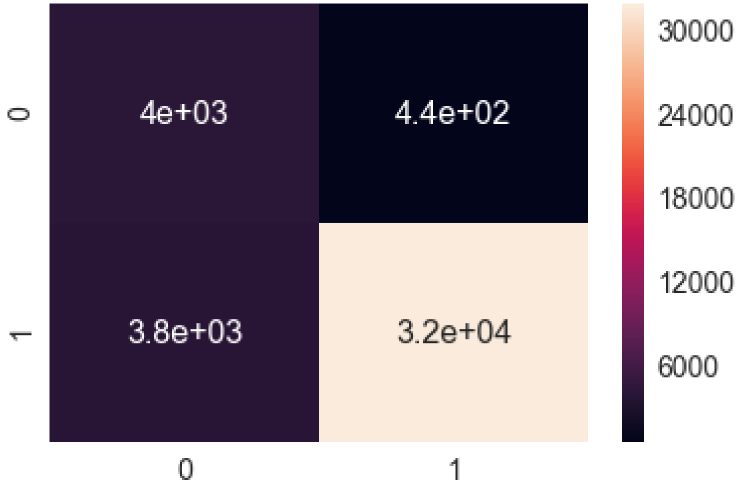
```
0 0.019778002865314205 tea
0 0.018765921135238294 yum
0 0.018354471005168273 supermarket
0 0.018151218487194466 add
0 0.017632601249007996 four star
0 0.01757726065738091 realli enjoy
```

In [42]:
```
modell2 = linear_model.SGDClassifier(alpha=1, average=False, class_weig
ht='balanced', epsilon=0.1,
       eta0=0.0, fit_intercept=True, l1_ratio=0.15,
       learning_rate='optimal', loss='hinge', max_iter=None, n_iter=Non
e,
       n_jobs=1, penalty='l2', power_t=0.5, random_state=None,
       shuffle=True, tol=None, verbose=0, warm_start=False)
modell2.fit(X_tr_tf_idfs, y_train)
print(modell2.score(X_test_tf_idfs, y_test))
print("error=",float(1-modell2.score(X_test_tf_idfs, y_test)))
```

```
0.88
error= 0.12
```

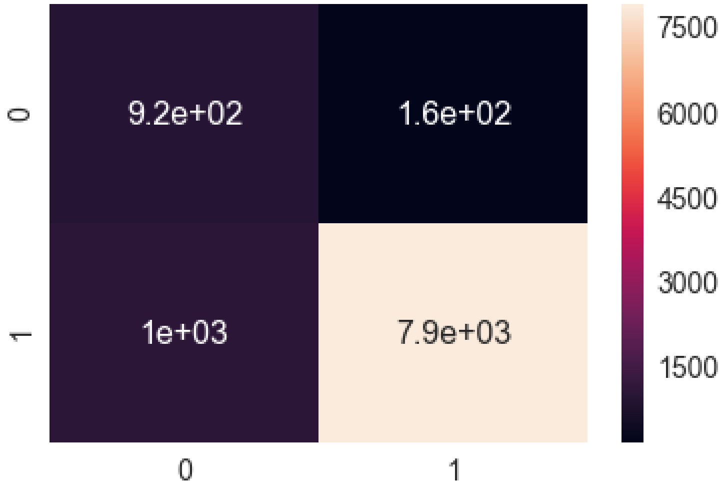
In [43]:
```
pred = modell2.predict(X_test_tf_idfs)
pred1 = modell2.predict(X_tr_tf_idfs)
from sklearn.metrics import confusion_matrix
import seaborn as sn
CFMt = confusion_matrix(y_train, pred1)
CFM = confusion_matrix(y_test, pred)
df_cm = pd.DataFrame(CFMt, range(2), range(2))
#plt.figure(figsize = (10,7))
sn.set(font_scale=1.4)#for label size
sn.heatmap(df_cm, annot=True,annot_kws={"size": 16})
```

Out[43]:
```
<matplotlib.axes._subplots.AxesSubplot at 0x1b013d38550>
```



In [44]:
```
df_cm = pd.DataFrame(CFM, range(2), range(2))
#plt.figure(figsize = (10,7))
sn.set(font_scale=1.4)#for label size
sn.heatmap(df_cm, annot=True,annot_kws={"size": 16})
```

Out[44]:
```
<matplotlib.axes._subplots.AxesSubplot at 0x1b012946160>
```



In [169]:
```
from sklearn import preprocessing
from sklearn import svm
import scipy.stats as stats
tuned_parameters ={'C': np.random.uniform(low=10**-4, high=10**4, size=
5),'gamma': np.random.uniform(low=10**-4, high=10**4, size=5)}
#lb = preprocessing.LabelBinarizer()
#y_train = np.array([number[0] for number in lb.fit_transform(y_trai
n)])
#Using RandomizedSearchCV
modeltfr1 = RandomizedSearchCV(svm.SVC(kernel='rbf',class_weight='balan
ced'),tuned_parameters,n_iter=5 ,scoring = 'roc_auc', cv=5)
modeltfr1.fit(X_tr_tf_idfs, y_train)
#y_test = np.array([number[0] for number in lb.fit_transform(y_test)])
print(modeltfr1.best_estimator_)
print(modeltfr1.score(X_test_tf_idfs, y_test))
```

```
SVC(C=7665.536150957552, cache_size=200, class_weight='balanced', coef0
=0.0,
  decision_function_shape='ovr', degree=3, gamma=1524.8403549731308,
  kernel='rbf', max_iter=-1, probability=False, random_state=None,
  shrinking=True, tol=0.001, verbose=False)
0.8080537815126051
```

In [171]:

```
modeltfr1.grid_scores_
```

Out[171]:

```
[mean: 0.79141, std: 0.00678, params: {'gamma': 1524.8403549731308,
'C': 7665.536150957552},
 mean: 0.79141, std: 0.00678, params: {'gamma': 5429.508574916167, 'C':
9286.40011075788},
 mean: 0.79141, std: 0.00678, params: {'gamma': 2252.4574315827113,
'C': 3355.307653614047},
 mean: 0.79141, std: 0.00678, params: {'gamma': 1524.8403549731308,
'C': 9286.40011075788},
 mean: 0.79141, std: 0.00678, params: {'gamma': 5429.508574916167, 'C':
7665.536150957552}]
```

In [45]:

```
from sklearn import svm
modelsvc = svm.SVC(C=7665.536150957552, cache_size=200, class_weight='b
alanced', coef0=0.0,
  decision_function_shape='ovr', degree=3, gamma=1524.8403549731308,
  kernel='rbf', max_iter=-1, probability=False, random_state=None,
  shrinking=True, tol=0.001, verbose=False)
modelsvc.fit(X_tr_tf_idfs, y_train)
print(modelsvc.score(X_test_tf_idfs, y_test))
```

```
0.9336
```

In [23]:

```
tuned_parameters = [10**-4,10**-2, 10**0, 10**2,10**4]
ERROR=[]
for c in tuned_parameters:
    for gamma in tuned_parameters:
        print("for c= ", c)
        print("for gamma= ", gamma)
        modelsvc2 = svm.SVC(C=7665.536150957552, cache_size=200, class_
weight='balanced', coef0=0.0,
        decision_function_shape='ovr', degree=3, gamma=1524.84035497313
08,
        kernel='rbf', max_iter=-1, probability=False, random_state=None
,
        shrinking=True, tol=0.001, verbose=False)
        modelsvc2.fit(X_tr_tf_idfs, y_train)
        f1_score=modelsvc2.score(X_test_tf_idfs, y_test)
        error = float(1-f1_score)
        print("error=",error)
        ERROR.append(error)
```

```
for c=  0.0001
for gamma=  0.0001
error= 0.06640000000000001
for c=  0.0001
for gamma=  0.01
error= 0.06640000000000001
for c=  0.0001
for gamma=  1
error= 0.06640000000000001
for c=  0.0001
for gamma=  100
error= 0.06640000000000001
for c=  0.0001
for gamma=  10000
```

```
error= 0.06640000000000001
for c=  0.01
for gamma=  0.0001
error= 0.06640000000000001
for c=  0.01
for gamma=  0.01
error= 0.06640000000000001
for c=  0.01
for gamma=  1
error= 0.06640000000000001
for c=  0.01
for gamma=  100
error= 0.06640000000000001
for c=  0.01
for gamma=  10000
error= 0.06640000000000001
for c=  1
for gamma=  0.0001
error= 0.06640000000000001
for c=  1
for gamma=  0.01
error= 0.06640000000000001
for c=  1
for gamma=  1
error= 0.06640000000000001
for c=  1
for gamma=  100
error= 0.06640000000000001
for c=  1
for gamma=  10000
error= 0.06640000000000001
for c=  100
for gamma=  0.0001
error= 0.06640000000000001
for c=  100
for gamma=  0.01
error= 0.06640000000000001
for c=  100
for gamma=  1
```

```
error= 0.06640000000000001
for c=  100
for gamma=  100
error= 0.06640000000000001
for c=  100
for gamma=  10000
error= 0.06640000000000001
for c=  10000
for gamma=  0.0001
error= 0.06640000000000001
for c=  10000
for gamma=  0.01
error= 0.06640000000000001
for c=  10000
for gamma=  1
error= 0.06640000000000001
for c=  10000
for gamma=  100
error= 0.06640000000000001
for c=  10000
for gamma=  10000
error= 0.06640000000000001
```

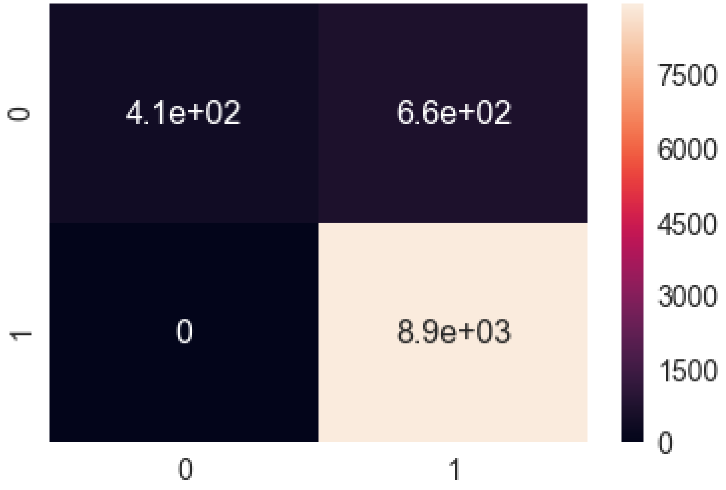
In [46]:

```
pred = modelsvc.predict(X_test_tf_idfs)
pred1 = modelsvc.predict(X_tr_tf_idfs)
from sklearn.metrics import confusion_matrix
import seaborn as sn
print(confusion_matrix(y_test, pred))
CFM = confusion_matrix(y_test, pred)
CFMtr = confusion_matrix(y_train, pred1)
df_cm = pd.DataFrame(CFM, range(2), range(2))
#plt.figure(figsize = (10,7))
sn.set(font_scale=1.4)#for label size
sn.heatmap(df_cm, annot=True,annot_kws={"size": 16})
```

```
[[ 411  664]
 [   0 8925]]
```

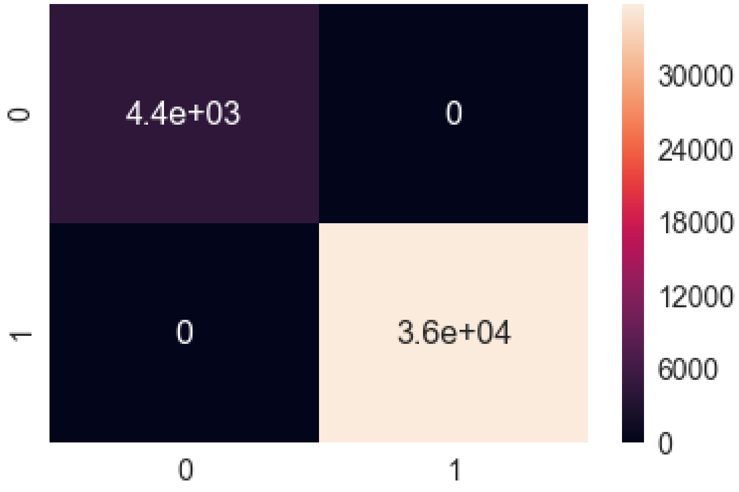
Out[46]:

```
<matplotlib.axes._subplots.AxesSubplot at 0x1b0706ab7f0>
```



In [47]:
```
df_cm = pd.DataFrame(CFMtr, range(2), range(2))
#plt.figure(figsize = (10,7))
sn.set(font_scale=1.4)#for label size
sn.heatmap(df_cm, annot=True,annot_kws={"size": 16})
```

Out[47]:
```
<matplotlib.axes._subplots.AxesSubplot at 0x1b0705630b8>
```



In [48]:
```
TP = CFM[0][0]
```

```
FP = CFM[1][0]
FN = CFM[0][1]
TN = CFM[1][1]
P = TP+FP
N = TN+FN
print('TP Value is',TP )
print('FP Value is',FP )
print('TN Value is',TN )
print('FN Value is',FN )

TPR = float(TP/P)
FPR = float(FP/P)
TNR = float(TN/N)
FNR = float(FN/N)
print('TPR Value is',TPR )
print('FPR Value is',FPR )
print('TNR Value is',TNR )
print('FNR Value is',FNR )
```

```
TP Value is 411
FP Value is 0
TN Value is 8925
FN Value is 664
TPR Value is 1.0
FPR Value is 0.0
TNR Value is 0.9307539889456669
FNR Value is 0.06924601105433309
```

In [49]:

```
from sklearn.metrics import accuracy_score, f1_score, precision_score,
recall_score
print(accuracy_score(y_test, pred))
print(f1_score(y_test, pred, average="macro"))
print(float(1-f1_score(y_test, pred, average="macro")))
print(precision_score(y_test, pred, average="macro"))
print(recall_score(y_test, pred, average="macro"))
```

```
0.9336
0.7586490511490995
0.24135094885090047
```

```
0.9653769944728334
0.6911627906976744
```

WORD2VEC

In [109]:

```
from gensim.models import Word2Vec
from gensim.models import KeyedVectors
import pickle
i=0
list_of_sent=[]
for sent in X_train['CleanedText'].values:
    list_of_sent.append(sent.split())
w2v_model=Word2Vec(list_of_sent,min_count=5,size=50, workers=4)
w2v_words = list(w2v_model.wv.vocab)
```

```
C:\Users\krush\Anaconda3\lib\site-packages\gensim\utils.py:1197: UserWa
rning: detected Windows; aliasing chunkize to chunkize_serial
  warnings.warn("detected Windows; aliasing chunkize to chunkize_seria
l")
```

In [110]:

```
from tqdm import tqdm
import os
sent_vectorstr = []; # the avg-w2v for each sentence/review is stored i
n this list
for sent in tqdm(list_of_sent): # for each review/sentence
    sent_vec = np.zeros(50) # as word vectors are of zero length
    cnt_words =0; # num of words with a valid vector in the sentence/re
view
    for word in sent: # for each word in a review/sentence
        if word in w2v_words:
            vec = w2v_model.wv[word]
            sent_vec += vec
            cnt_words += 1
    if cnt_words != 0:
        sent_vec /= cnt_words
    sent_vectorstr.append(sent_vec)
```

```
100%|██████████████████████████████████████████████████████████████████
████████| 40000/40000 [00:37<00:00, 1060.43it/s]
```

```
In [111]: list_of_sent1 = []
          for sent in X_test['CleanedText'].values:
              list_of_sent1.append(sent.split())
          sent_vectorstest = []; # the avg-w2v for each sentence/review is stored
           in this list
          for sent in tqdm(list_of_sent1): # for each review/sentence
              sent_vec = np.zeros(50) # as word vectors are of zero length
              cnt_words =0; # num of words with a valid vector in the sentence/re
          view
              for word in sent: # for each word in a review/sentence
                  if word in w2v_words:
                      vec = w2v_model.wv[word]
                      sent_vec += vec
                      cnt_words += 1
              if cnt_words != 0:
                  sent_vec /= cnt_words
              sent_vectorstest.append(sent_vec)
```

```
100%|██████████████████████████████████████████████████████████████████
████████| 10000/10000 [00:09<00:00, 1035.89it/s]
```

```
In [112]: from sklearn.preprocessing import StandardScaler

          std_svm=StandardScaler(with_mean=False).fit(sent_vectorstr)
          sent_vectorstrs=std_svm.transform(sent_vectorstr)
          sent_vectorstests=std_svm.transform(sent_vectorstest)
```

```
In [113]: from sklearn import preprocessing
          from sklearn import linear_model
          tuned_parameters = [{'C': [10**-4, 10**-2, 10**0, 10**2, 10**4]}]

          hyperparameter = dict(alpha=[10**-4, 10**-2, 10**0, 10**2, 10**4],penal
          ty=['l1','l2'])

          #Using GridSearchCV
```

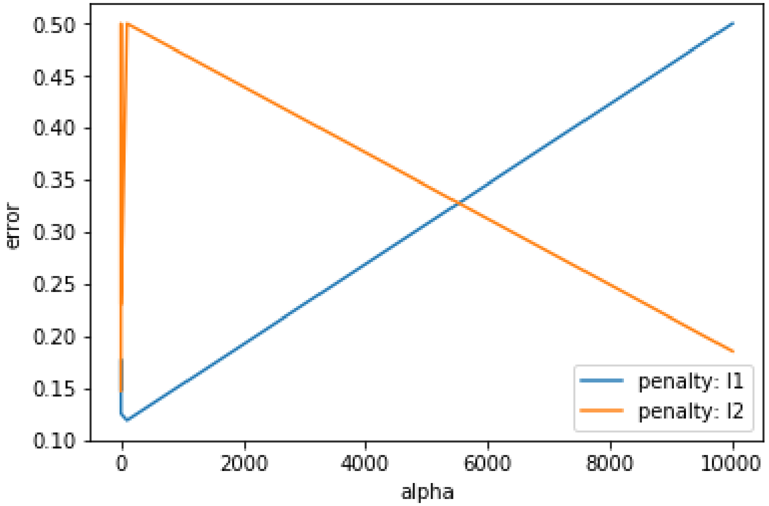
```
modelw2v = GridSearchCV(linear_model.SGDClassifier(class_weight='balanc
ed'), hyperparameter, scoring = 'roc_auc', cv=5)
modelw2v.fit(sent_vectorstrs, y_train)

print(modelw2v.best_estimator_)
print(modelw2v.score(sent_vectorstests, y_test))
```

```
SGDClassifier(alpha=0.01, average=False, class_weight='balanced', epsil
on=0.1,
       eta0=0.0, fit_intercept=True, l1_ratio=0.15,
       learning_rate='optimal', loss='hinge', max_iter=None, n_iter=Non
e,
       n_jobs=1, penalty='l2', power_t=0.5, random_state=None,
       shuffle=True, tol=None, verbose=0, warm_start=False)
0.8831390267735001
```

In [114]:

```
scores = [x[1] for x in modelw2v.grid_scores_]
scores = np.array(scores).reshape(len(hyperparameter['penalty']), len(h
yperparameter['alpha']))
for ind, i in enumerate(hyperparameter['penalty']):
    plt.plot(hyperparameter['alpha'],1- scores[ind], label='penalty: '
+ str(i))
plt.legend()
plt.xlabel('alpha')
plt.ylabel('error')
plt.show()
```

```
C:\Users\krush\Anaconda3\lib\site-packages\sklearn\model_selection\_sea
rch.py:761: DeprecationWarning: The grid_scores_ attribute was deprecat
ed in version 0.18 in favor of the more elaborate cv_results_ attribut
e. The grid_scores_ attribute will not be available from 0.20
  DeprecationWarning)
```



In [115]:

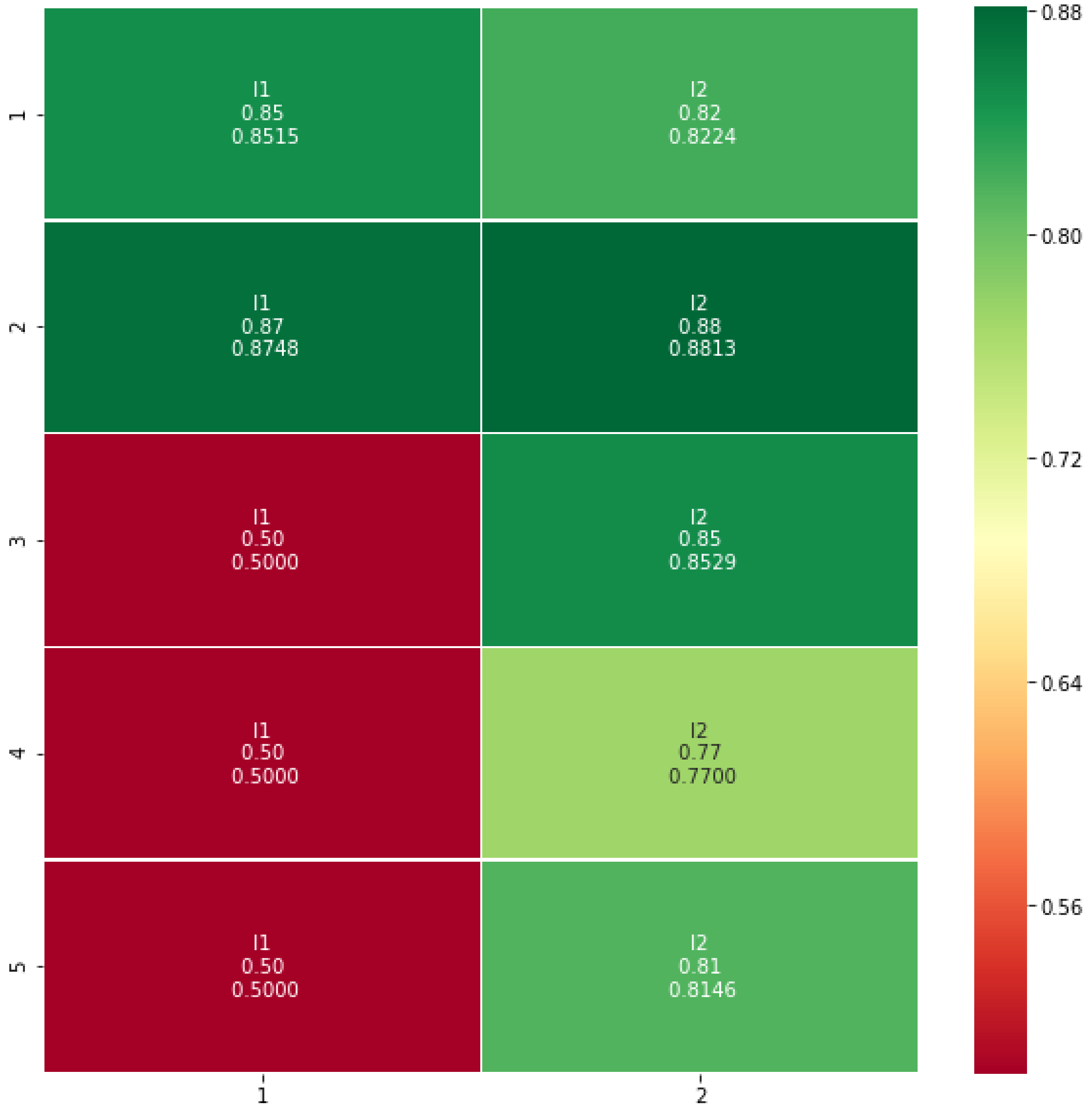
```
alpha=[]
pen=[]
mean=[]
for a in modelw2v.grid_scores_:
    alpha.append(a[0]['alpha'])
    pen.append(a[0]['penalty'])
    mean.append(a[1])
alpha=np.asarray(alpha)
pen=np.asarray(pen)
mean=np.asarray(mean)
alpha = alpha.reshape(5,2)
pen = pen.reshape(5,2)
mean = mean.reshape(5,2)
result = pd.DataFrame(mean,index=[1,2,3,4,5],columns = [1,2])

label =np.asarray([" {0} \n {1:.2f} \n {1:.4f}".format(pen,mean,alpha)
for pen,mean,alpha in zip(pen.flatten(),mean.flatten(),alpha.flatten
())]).reshape(5,2)
print(label)

import seaborn as sns
fig , ax = plt.subplots(figsize = (10,10))
```

```
ax.set_xticks([])
ax.set_yticks([])
sns.heatmap(result,annot=label,fmt="" ,cmap ='RdYlGn',linewidths = 0.30
 , ax = ax )
```

```
C:\Users\krush\Anaconda3\lib\site-packages\sklearn\model_selection\_sea
rch.py:761: DeprecationWarning: The grid_scores_ attribute was deprecat
ed in version 0.18 in favor of the more elaborate cv_results_ attribut
e. The grid_scores_ attribute will not be available from 0.20
  DeprecationWarning)
```

```
[[' l1 \n 0.85 \n 0.8515' ' l2 \n 0.82 \n 0.8224']
 [' l1 \n 0.87 \n 0.8748' ' l2 \n 0.88 \n 0.8813']
 [' l1 \n 0.50 \n 0.5000' ' l2 \n 0.85 \n 0.8529']
 [' l1 \n 0.50 \n 0.5000' ' l2 \n 0.77 \n 0.7700']
 [' l1 \n 0.50 \n 0.5000' ' l2 \n 0.81 \n 0.8146']]
```

```
Out[115]: <matplotlib.axes._subplots.AxesSubplot at 0x2d52c358518>
```



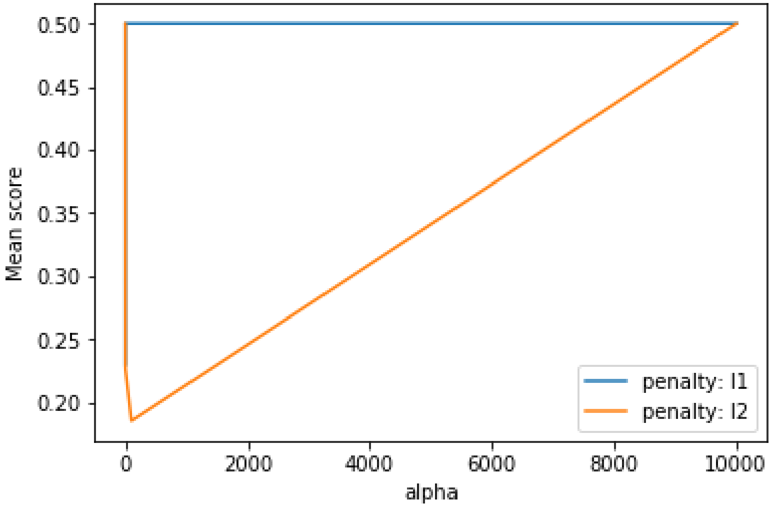
In [119]:

```
from sklearn import preprocessing
import scipy.stats as stats

tuned_parameters = dict(alpha=np.random.uniform(low=2**-9, high=2**9, s
ize=10),penalty=['l1','l2'])
#lb = preprocessing.LabelBinarizer()
#y_train = np.array([number[0] for number in lb.fit_transform(y_trai
n)])
#Using RandomizedSearchCV
modelw2vr = RandomizedSearchCV(linear_model.SGDClassifier(class_weight=
'balanced'),tuned_parameters,n_iter=10 ,scoring = 'roc_auc', cv=5)
modelw2vr.fit(sent_vectorstrs, y_train)
#y_test = np.array([number[0] for number in lb.fit_transform(y_test)])
print(modelw2vr.best_estimator_)
print(modelw2vr.score(sent_vectorstests, y_test))
```

```
SGDClassifier(alpha=507.15311139310444, average=False,
       class_weight='balanced', epsilon=0.1, eta0=0.0, fit_intercept=Tr
ue,
       l1_ratio=0.15, learning_rate='optimal', loss='hinge', max_iter=N
one,
       n_iter=None, n_jobs=1, penalty='l2', power_t=0.5, random_state=N
one,
       shuffle=True, tol=None, verbose=0, warm_start=False)
0.8141191062471501
```

In [120]:

```
scores = [x[1] for x in modelw2vr.grid_scores_]
scores = np.array(scores).reshape(len(hyperparameter['penalty']), len(h
yperparameter['alpha']))
for ind, i in enumerate(hyperparameter['penalty']):
    plt.plot(hyperparameter['alpha'],1-scores[ind], label='penalty: ' +
 str(i))
plt.legend()
plt.xlabel('alpha')
plt.ylabel('Mean score')
plt.show()
```



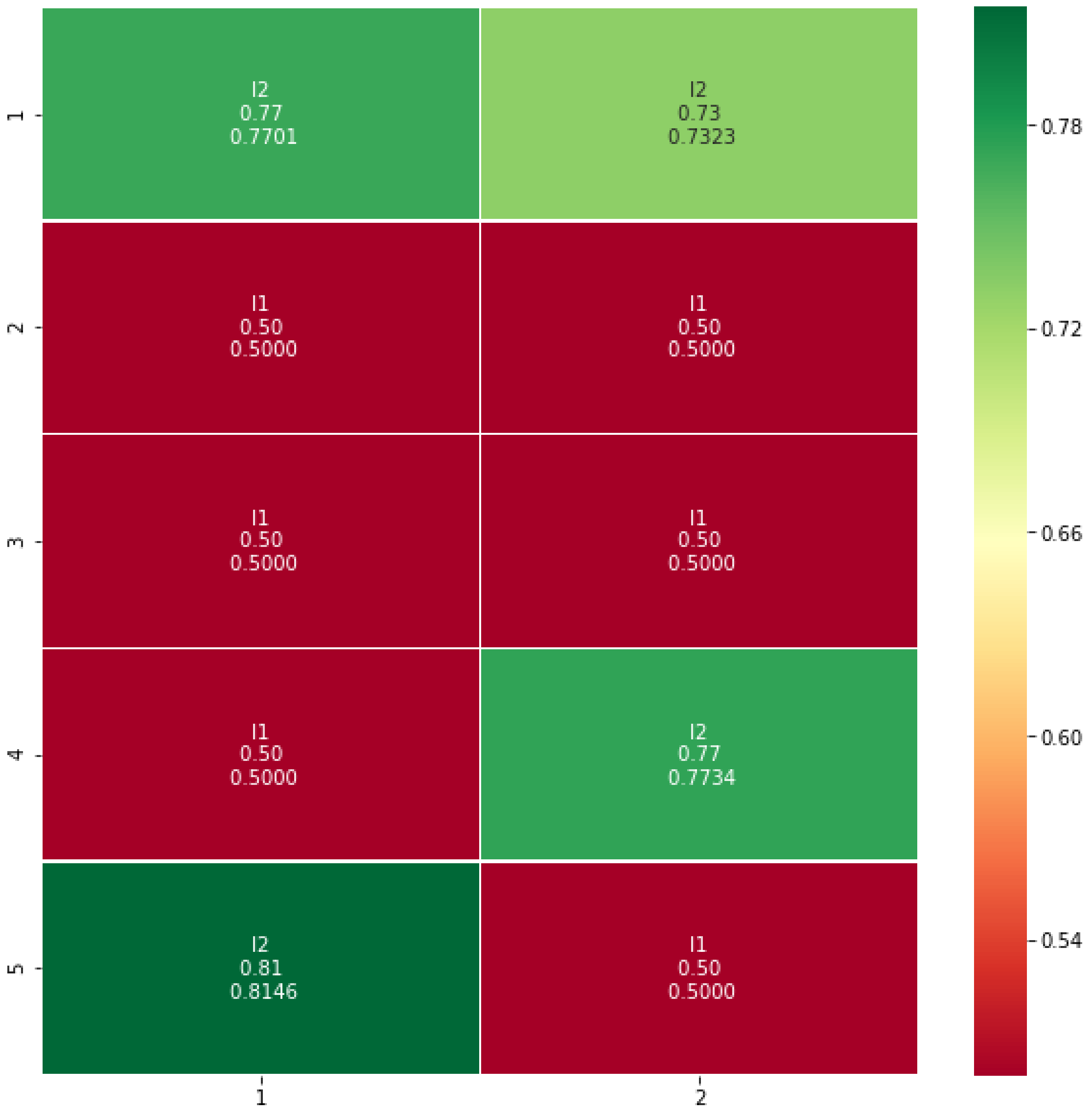
In [121]:

```
alpha=[]
pen=[]
mean=[]
for a in modelw2vr.grid_scores_:
    alpha.append(a[0]['alpha'])
    pen.append(a[0]['penalty'])
    mean.append(a[1])
alpha=np.asarray(alpha)
pen=np.asarray(pen)
mean=np.asarray(mean)
alpha = alpha.reshape(5,2)
pen = pen.reshape(5,2)
mean = mean.reshape(5,2)
result = pd.DataFrame(mean,index=[1,2,3,4,5],columns = [1,2])

label =np.asarray([" {0} \n {1:.2f} \n {1:.4f}".format(pen,mean,alpha)
for pen,mean,alpha in zip(pen.flatten(),mean.flatten(),alpha.flatten
())]).reshape(5,2)
print(label)

import seaborn as sns
fig , ax = plt.subplots(figsize = (10,10))
```

```
ax.set_xticks([])
ax.set_yticks([])
sns.heatmap(result,annot=label,fmt="" ,cmap ='RdYlGn',linewidths = 0.30
 , ax = ax )
```

```
[[' l2 \n 0.77 \n 0.7701' ' l2 \n 0.73 \n 0.7323']
 [' l1 \n 0.50 \n 0.5000' ' l1 \n 0.50 \n 0.5000']
 [' l1 \n 0.50 \n 0.5000' ' l1 \n 0.50 \n 0.5000']
 [' l1 \n 0.50 \n 0.5000' ' l2 \n 0.77 \n 0.7734']
 [' l2 \n 0.81 \n 0.8146' ' l1 \n 0.50 \n 0.5000']]
```

```
Out[121]: <matplotlib.axes._subplots.AxesSubplot at 0x2d52c4ca6a0>
```



In [19]:

```
from sklearn import linear_model
#model l2 is not working properly because ....  when we see last cell
 best one is with penality='l1'
modell2 = linear_model.SGDClassifier(alpha=0.01, average=False, class_w
eight='balanced', epsilon=0.1,
       eta0=0.0, fit_intercept=True, l1_ratio=0.15,
       learning_rate='optimal', loss='hinge', max_iter=None, n_iter=Non
e,
       n_jobs=1, penalty='l2', power_t=0.5, random_state=None,
       shuffle=True, tol=None, verbose=0, warm_start=False)
modell2.fit(sent_vectorstrs, y_train)
print(modell2.score(sent_vectorstests, y_test))
```

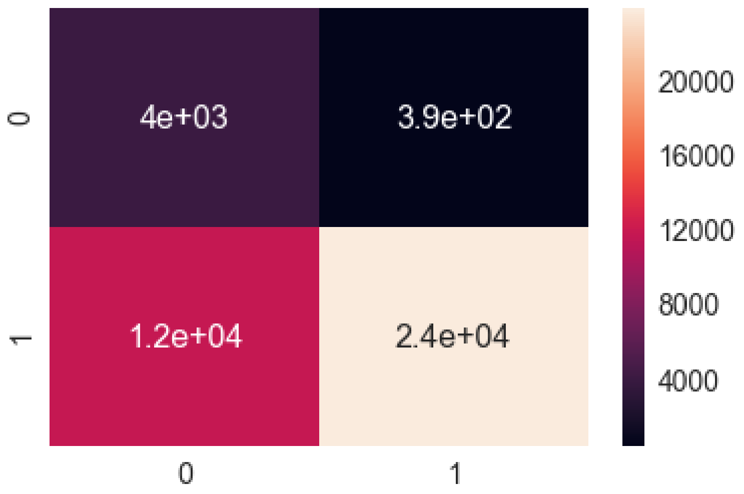
```
0.825
```

In [88]:

```
pred = modell2.predict(sent_vectorstests)
pred1 = modell2.predict(sent_vectorstrs)
from sklearn.metrics import confusion_matrix
import seaborn as sn
CFMt = confusion_matrix(y_train, pred1)
CFM = confusion_matrix(y_test, pred)
df_cm = pd.DataFrame(CFMt, range(2), range(2))
#plt.figure(figsize = (10,7))
sn.set(font_scale=1.4)#for label size
sn.heatmap(df_cm, annot=True,annot_kws={"size": 16})
```

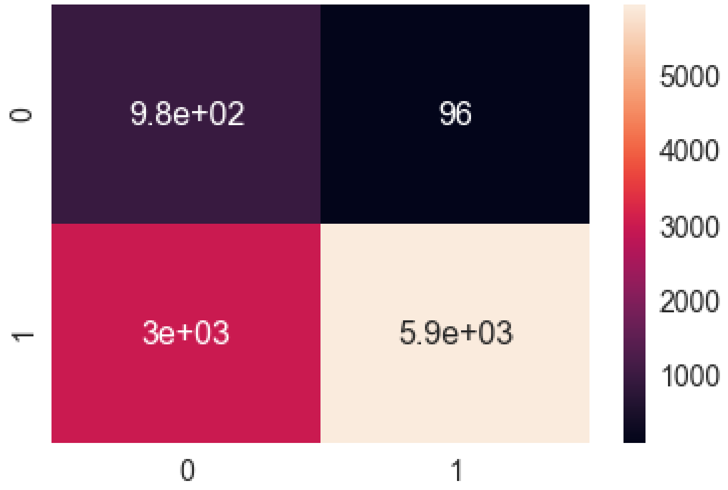
Out[88]:

```
<matplotlib.axes._subplots.AxesSubplot at 0x27421a33ef0>
```



In [89]:
```python
df_cm = pd.DataFrame(CFM, range(2), range(2))
#plt.figure(figsize = (10,7))
sn.set(font_scale=1.4)#for label size
sn.heatmap(df_cm, annot=True,annot_kws={"size": 16})
```

Out[89]:
```
<matplotlib.axes._subplots.AxesSubplot at 0x2741b41acf8>
```



In [90]:
```python
from sklearn.metrics import accuracy_score, f1_score, precision_score, recall_score
print(accuracy_score(y_test, pred))
print(f1_score(y_test, pred, average="macro"))
print(float(1-f1_score(y_test, pred, average="macro")))
print(precision_score(y_test, pred, average="macro"))
print(recall_score(y_test, pred, average="macro"))
```

```
0.6898
0.5896629998253885
0.4103370001746115
0.6148555837507862
0.7869454758647645
```

In [48]:
```python
from sklearn import preprocessing
from sklearn import svm
import scipy.stats as stats
```

```
tuned_parameters ={'C': np.random.uniform(low=10**-4, high=10**4, size=5),'gamma': np.random.uniform(low=10**-4, high=10**4, size=5)}
#lb = preprocessing.LabelBinarizer()
#y_train = np.array([number[0] for number in lb.fit_transform(y_train)])
#Using RandomizedSearchCV
modelw2vr1 = RandomizedSearchCV(svm.SVC(kernel='rbf',class_weight='balanced'),tuned_parameters,n_iter=5 ,scoring = 'roc_auc', cv=5)
modelw2vr1.fit(sent_vectorstrs, y_train)
#y_test = np.array([number[0] for number in lb.fit_transform(y_test)])
print(modelw2vr1.best_estimator_)
print(modelw2vr1.score(sent_vectorstests, y_test))
```

```
SVC(C=578.6781512458239, cache_size=200, class_weight='balanced', coef0=0.0,
  decision_function_shape='ovr', degree=3, gamma=1808.766276202022,
  kernel='rbf', max_iter=-1, probability=False, random_state=None,
  shrinking=True, tol=0.001, verbose=False)
0.9307539889456669
```

In [55]:

```
from sklearn import svm
modelsvc2 = svm.SVC(C=578.6781512458239, cache_size=200, class_weight='balanced', coef0=0.0,
  decision_function_shape='ovr', degree=3, gamma=1808.766276202022,
  kernel='rbf', max_iter=-1, probability=False, random_state=None,
  shrinking=True, tol=0.001, verbose=False)
modelsvc2.fit(sent_vectorstrs, y_train)
print(modelsvc2.score(sent_vectorstests, y_test))
```

```
0.9336
```

In [58]:

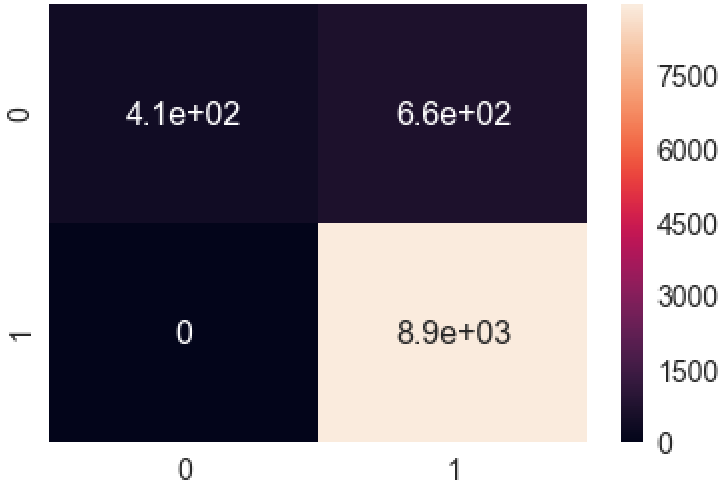
```
tuned_parameters = [10**-2, 10**0, 10**2]
ERROR=[]
for c in tuned_parameters:
    for gamma in tuned_parameters:
        print("for c= ", c)
        print("for gamma= ", gamma)
        modelsvc1 = svm.SVC(C=0.01, cache_size=200, class_weight='balanced', coef0=0.0,
        decision_function_shape='ovr', degree=3, gamma=2041.51393776413
74,
        kernel='rbf', max_iter=-1, probability=False, random_state=None
,
        shrinking=True, tol=0.001, verbose=False)
        modelsvc1.fit(sent_vectorstr1, y_train)
        f1_score=modelsvc1.score(sent_vectorstests, y_test)
        error = float(1-f1_score)
        print("error=",error)
        ERROR.append(error)
```

```
for c=  0.01
for gamma=  0.01
error= 0.10750000000000004
for c=  0.01
for gamma=  1
error= 0.10750000000000004
for c=  0.01
for gamma=  100
error= 0.10750000000000004
for c=  1
for gamma=  0.01
error= 0.10750000000000004
for c=  1
for gamma=  1
error= 0.10750000000000004
for c=  1
for gamma=  100
error= 0.10750000000000004
for c=  100
for gamma=  0.01
error= 0.10750000000000004
for c=  100
for gamma=  1
error= 0.10750000000000004
for c=  100
for gamma=  100
error= 0.10750000000000004
```

In [49]:
```
pred = modelw2vr1.predict(sent_vectorstests)
pred1 = modelw2vr1.predict(sent_vectorstrs)
from sklearn.metrics import confusion_matrix
import seaborn as sn
print(confusion_matrix(y_test, pred))
CFM = confusion_matrix(y_test, pred)
CFMtr = confusion_matrix(y_train, pred1)
df_cm = pd.DataFrame(CFM, range(2), range(2))
#plt.figure(figsize = (10,7))
sn.set(font_scale=1.4)#for label size
sn.heatmap(df_cm, annot=True,annot_kws={"size": 16})
```

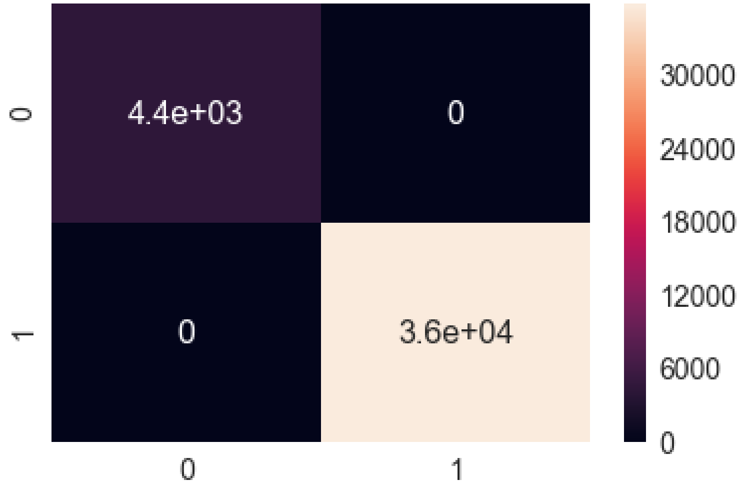
```
[[ 411  664]
 [   0 8925]]
```

Out[49]:
```
<matplotlib.axes._subplots.AxesSubplot at 0x2741a633dd8>
```



In [50]:
```
df_cm = pd.DataFrame(CFMtr, range(2), range(2))
#plt.figure(figsize = (10,7))
sn.set(font_scale=1.4)#for label size
sn.heatmap(df_cm, annot=True,annot_kws={"size": 16})
```

Out[50]:
```
<matplotlib.axes._subplots.AxesSubplot at 0x2741a760780>
```



In [51]:

```
TP = CFM[0][0]
FP = CFM[1][0]
FN = CFM[0][1]
TN = CFM[1][1]
P = TP+FP
N = TN+FN
print('TP Value is',TP )
print('FP Value is',FP )
print('TN Value is',TN )
print('FN Value is',FN )

TPR = float(TP/P)
FPR = float(FP/P)
TNR = float(TN/N)
FNR = float(FN/N)
print('TPR Value is',TPR )
print('FPR Value is',FPR )
print('TNR Value is',TNR )
print('FNR Value is',FNR )
```

```
TP Value is 411
FP Value is 0
TN Value is 8925
FN Value is 664
```

```
TPR Value is 1.0
FPR Value is 0.0
TNR Value is 0.9307539889456669
FNR Value is 0.06924601105433309
```

In [52]:

```
from sklearn.metrics import accuracy_score, f1_score, precision_score,
recall_score
print(accuracy_score(y_test, pred))
print(f1_score(y_test, pred, average="macro"))
print(float(1-f1_score(y_test, pred, average="macro")))
print(precision_score(y_test, pred, average="macro"))
print(recall_score(y_test, pred, average="macro"))
```

```
0.9336
0.7586490511490995
0.24135094885090047
0.9653769944728334
0.6911627906976744
```

TF-IDF W2V

In [122]:

```
from tqdm import tqdm
import os
# TF-IDF weighted Word2Vec

tfidf_feat = tf_idf_vect.get_feature_names()
dictionary = dict(zip(tf_idf_vect1.get_feature_names(), list(tf_idf_vec
t1.idf_)))# tfidf words/col-names
# final_tf_idf is the sparse matrix with row= sentence, col=word and ce
ll_val = tfidf

tfidf_sent_vectors = []; # the tfidf-w2v for each sentence/review is st
ored in this list
row=0;
for sent in tqdm(list_of_sent): # for each review/sentence
    sent_vec = np.zeros(50) # as word vectors are of zero length
    weight_sum =0; # num of words with a valid vector in the sentence/r
```

```
eview
    for word in sent: # for each word in a review/sentence
        if word in w2v_words:
            vec = w2v_model.wv[word]
            # obtain the tf_idfidf of a word in a sentence/review
            tf_idf = dictionary[word]*(sent.count(word)/len(sent))
            sent_vec += (vec * tf_idf)
            weight_sum += tf_idf
    if weight_sum != 0:
        sent_vec /= weight_sum
    tfidf_sent_vectors.append(sent_vec)
    row += 1
```

```
100%|█████████████████████████████████
█████| 40000/40000 [00:49<00:00, 811.16it/s]
```

In [123]:

```
# TF-IDF weighted Word2Vec
tfidf_feat = tf_idf_vect1.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_sent_vectors_test = []; # the tfidf-w2v for each sentence/review
 is stored in this list
row=0;
for sent in tqdm(list_of_sent1): # for each review/sentence
    sent_vec = np.zeros(50) # as word vectors are of zero length
    weight_sum =0; # num of words with a valid vector in the sentence/r
eview
    for word in sent: # for each word in a review/sentence
        if word in w2v_words:
            vec = w2v_model.wv[word]
            # obtain the tf_idfidf of a word in a sentence/review
            tf_idf = dictionary[word]*(sent.count(word)/len(sent))
            sent_vec += (vec * tf_idf)
            weight_sum += tf_idf
    if weight_sum != 0:
        sent_vec /= weight_sum
    tfidf_sent_vectors_test.append(sent_vec)
    row += 1
```

```
100%|████████████████████████████████████████████████████████████
█████████| 10000/10000 [00:12<00:00, 831.50it/s]
```

In [124]:

```python
from sklearn.preprocessing import StandardScaler

std_svm=StandardScaler(with_mean=False).fit(tfidf_sent_vectors)
tfidf_sent_vectorss=std_svm.transform(tfidf_sent_vectors)
tfidf_sent_vectors_tests=std_svm.transform(tfidf_sent_vectors_test)
```

In [125]:

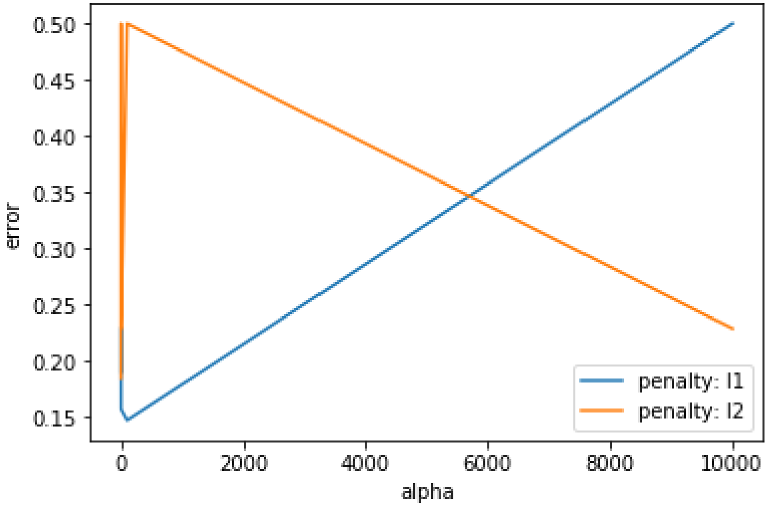
```python
from sklearn import preprocessing
from sklearn import linear_model
tuned_parameters = [{'C': [10**-4, 10**-2, 10**0, 10**2, 10**4]}]

hyperparameter = dict(alpha=[10**-4, 10**-2, 10**0, 10**2, 10**4],penal
ty=['l1','l2'])

#Using GridSearchCV
modeltfw2v = GridSearchCV(linear_model.SGDClassifier(class_weight='bala
nced'), hyperparameter, scoring = 'roc_auc', cv=5)
modeltfw2v.fit(tfidf_sent_vectorss, y_train)

print(modeltfw2v.best_estimator_)
print(modeltfw2v.score(tfidf_sent_vectors_tests, y_test))
```

```
SGDClassifier(alpha=0.01, average=False, class_weight='balanced', epsil
on=0.1,
       eta0=0.0, fit_intercept=True, l1_ratio=0.15,
       learning_rate='optimal', loss='hinge', max_iter=None, n_iter=Non
e,
       n_jobs=1, penalty='l2', power_t=0.5, random_state=None,
       shuffle=True, tol=None, verbose=0, warm_start=False)
0.8549359390267736
```

In [126]:

```python
scores = [x[1] for x in modeltfw2v.grid_scores_]
scores = np.array(scores).reshape(len(hyperparameter['penalty']), len(h
yperparameter['alpha']))
for ind, i in enumerate(hyperparameter['penalty']):
    plt.plot(hyperparameter['alpha'],1- scores[ind], label='penalty: '
+ str(i))
plt.legend()
plt.xlabel('alpha')
plt.ylabel('error')
plt.show()
```

```
C:\Users\krush\Anaconda3\lib\site-packages\sklearn\model_selection\_sea
rch.py:761: DeprecationWarning: The grid_scores_ attribute was deprecat
ed in version 0.18 in favor of the more elaborate cv_results_ attribut
e. The grid_scores_ attribute will not be available from 0.20
  DeprecationWarning)
```



In [127]:

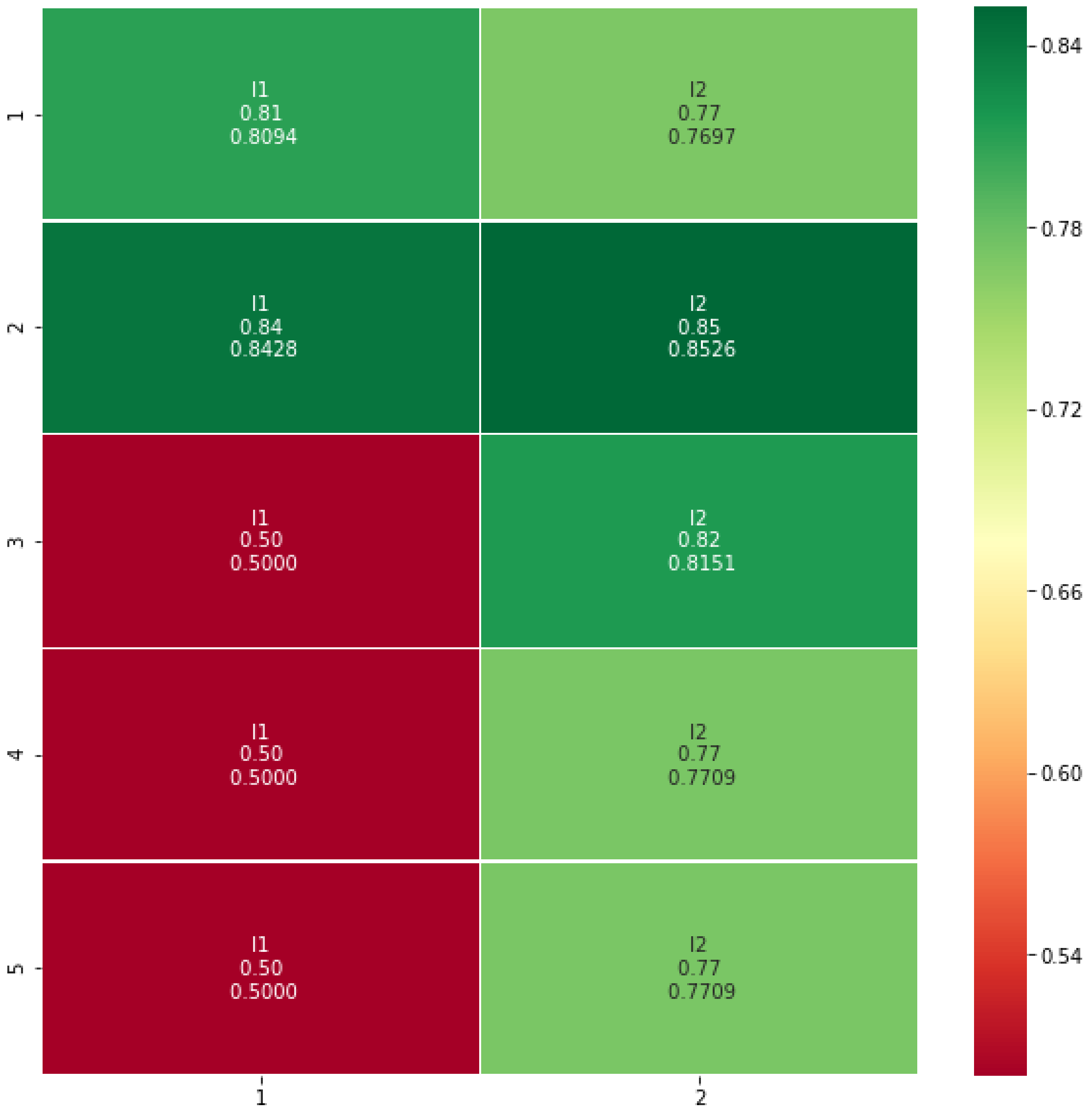
```
alpha=[]
pen=[]
mean=[]
for a in modeltfw2v.grid_scores_:
    alpha.append(a[0]['alpha'])
    pen.append(a[0]['penalty'])
    mean.append(a[1])
alpha=np.asarray(alpha)
pen=np.asarray(pen)
mean=np.asarray(mean)
alpha = alpha.reshape(5,2)
```

```
pen = pen.reshape(5,2)
mean = mean.reshape(5,2)
result = pd.DataFrame(mean,index=[1,2,3,4,5],columns = [1,2])

label =np.asarray([" {0} \n {1:.2f} \n {1:.4f}".format(pen,mean,alpha)
for pen,mean,alpha in zip(pen.flatten(),mean.flatten(),alpha.flatten
())]).reshape(5,2)
print(label)

import seaborn as sns
fig , ax = plt.subplots(figsize = (10,10))
ax.set_xticks([])
ax.set_yticks([])
sns.heatmap(result,annot=label,fmt="" ,cmap ='RdYlGn',linewidths = 0.30
 , ax = ax )
```

```
C:\Users\krush\Anaconda3\lib\site-packages\sklearn\model_selection\_sea
rch.py:761: DeprecationWarning: The grid_scores_ attribute was deprecat
ed in version 0.18 in favor of the more elaborate cv_results_ attribut
e. The grid_scores_ attribute will not be available from 0.20
  DeprecationWarning)
```

```
[[' l1 \n 0.81 \n 0.8094' ' l2 \n 0.77 \n 0.7697']
 [' l1 \n 0.84 \n 0.8428' ' l2 \n 0.85 \n 0.8526']
 [' l1 \n 0.50 \n 0.5000' ' l2 \n 0.82 \n 0.8151']
 [' l1 \n 0.50 \n 0.5000' ' l2 \n 0.77 \n 0.7709']
 [' l1 \n 0.50 \n 0.5000' ' l2 \n 0.77 \n 0.7709']]
```

```
Out[127]: <matplotlib.axes._subplots.AxesSubplot at 0x2d530649940>
```



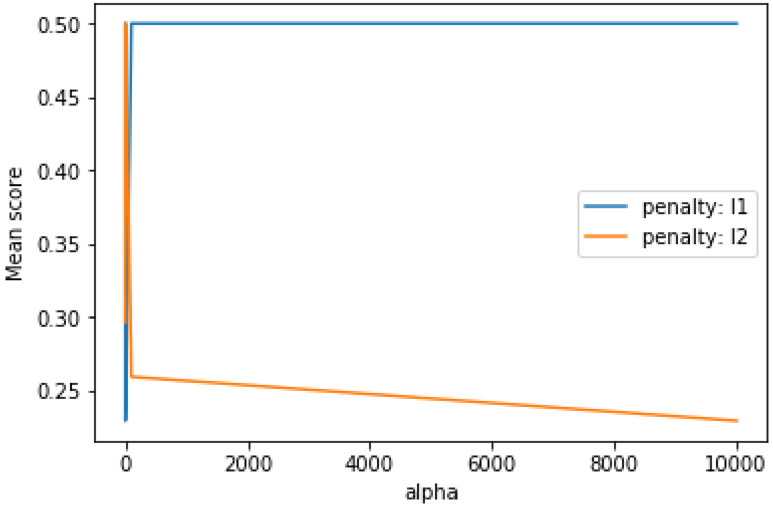
In [128]:

```
from sklearn import preprocessing
import scipy.stats as stats

tuned_parameters = dict(alpha=np.random.uniform(low=2**-9, high=2**9, s
ize=10),penalty=['l1','l2'])
#lb = preprocessing.LabelBinarizer()
#y_train = np.array([number[0] for number in lb.fit_transform(y_trai
n)])
#Using RandomizedSearchCV
modeltfw2vr = RandomizedSearchCV(linear_model.SGDClassifier(class_weigh
t='balanced'),tuned_parameters,n_iter=10 ,scoring = 'roc_auc', cv=5)
modeltfw2vr.fit(tfidf_sent_vectorss, y_train)
#y_test = np.array([number[0] for number in lb.fit_transform(y_test)])
print(modeltfw2vr.best_estimator_)
print(modeltfw2vr.score(tfidf_sent_vectors_tests, y_test))
```

```
SGDClassifier(alpha=458.8377426817654, average=False, class_weight='bal
anced',
       epsilon=0.1, eta0=0.0, fit_intercept=True, l1_ratio=0.15,
       learning_rate='optimal', loss='hinge', max_iter=None, n_iter=Non
e,
       n_jobs=1, penalty='l2', power_t=0.5, random_state=None,
       shuffle=True, tol=None, verbose=0, warm_start=False)
0.7684676698586412
```

In [129]:

```
scores = [x[1] for x in modeltfw2vr.grid_scores_]
scores = np.array(scores).reshape(len(hyperparameter['penalty']), len(h
yperparameter['alpha']))
for ind, i in enumerate(hyperparameter['penalty']):
    plt.plot(hyperparameter['alpha'],1-scores[ind], label='penalty: ' +
 str(i))
plt.legend()
plt.xlabel('alpha')
plt.ylabel('Mean score')
plt.show()
```



In [130]:

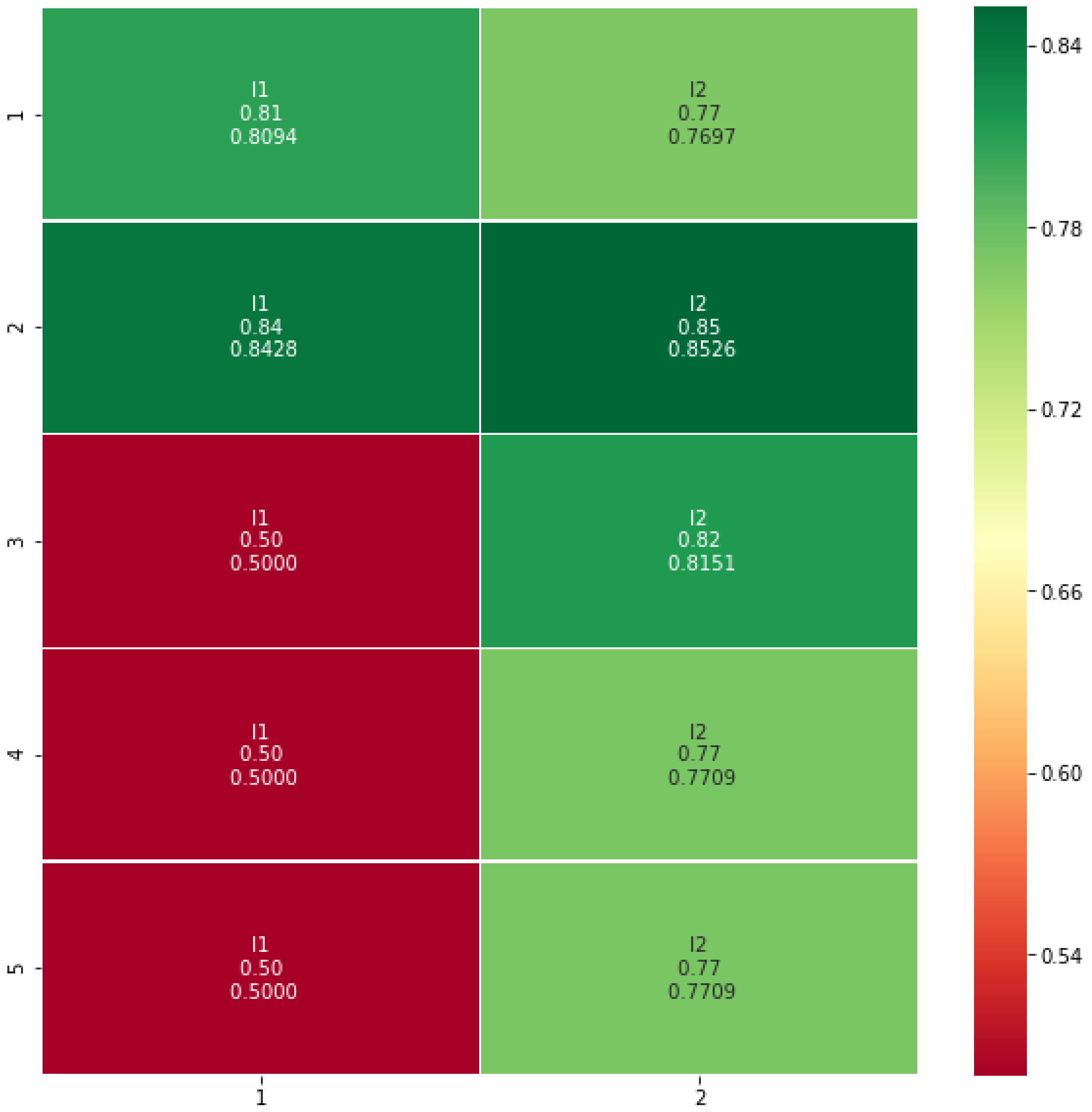
```
alpha=[]
pen=[]
mean=[]
for a in modeltfw2v.grid_scores_:
    alpha.append(a[0]['alpha'])
    pen.append(a[0]['penalty'])
    mean.append(a[1])
alpha=np.asarray(alpha)
pen=np.asarray(pen)
mean=np.asarray(mean)
alpha = alpha.reshape(5,2)
pen = pen.reshape(5,2)
mean = mean.reshape(5,2)
result = pd.DataFrame(mean,index=[1,2,3,4,5],columns = [1,2])

label =np.asarray([" {0} \n {1:.2f} \n {1:.4f}".format(pen,mean,alpha)
for pen,mean,alpha in zip(pen.flatten(),mean.flatten(),alpha.flatten
())]).reshape(5,2)
print(label)

import seaborn as sns
fig , ax = plt.subplots(figsize = (10,10))
```

```
ax.set_xticks([])
ax.set_yticks([])
sns.heatmap(result,annot=label,fmt="" ,cmap ='RdYlGn',linewidths = 0.30
 , ax = ax )
```

```
C:\Users\krush\Anaconda3\lib\site-packages\sklearn\model_selection\_sea
rch.py:761: DeprecationWarning: The grid_scores_ attribute was deprecat
ed in version 0.18 in favor of the more elaborate cv_results_ attribut
e. The grid_scores_ attribute will not be available from 0.20
  DeprecationWarning)
```

```
[[' l1 \n 0.81 \n 0.8094' ' l2 \n 0.77 \n 0.7697']
 [' l1 \n 0.84 \n 0.8428' ' l2 \n 0.85 \n 0.8526']
 [' l1 \n 0.50 \n 0.5000' ' l2 \n 0.82 \n 0.8151']
 [' l1 \n 0.50 \n 0.5000' ' l2 \n 0.77 \n 0.7709']
 [' l1 \n 0.50 \n 0.5000' ' l2 \n 0.77 \n 0.7709']]
```

```
Out[130]: <matplotlib.axes._subplots.AxesSubplot at 0x2d52ff82198>
```



In [76]:
```
#model l2 is not working properly because ....  when we see last cell
 best one is with penality='l1'
modell2 = linear_model.SGDClassifier(alpha=0.01, average=False, class_w
eight='balanced', epsilon=0.1,
       eta0=0.0, fit_intercept=True, l1_ratio=0.15,
       learning_rate='optimal', loss='hinge', max_iter=None, n_iter=Non
e,
       n_jobs=1, penalty='l2', power_t=0.5, random_state=None,
       shuffle=True, tol=None, verbose=0, warm_start=False)
modell2.fit(tfidf_sent_vectorss, y_train)
print(modell2.score(tfidf_sent_vectors_tests, y_test))
```

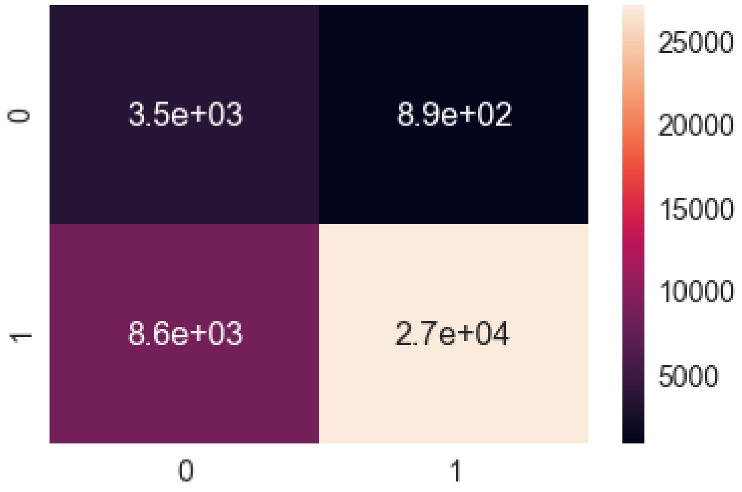
```
0.7584
```

In [77]:

```
pred = modell2.predict(tfidf_sent_vectors_tests)
pred1 = modell2.predict(tfidf_sent_vectorss)
from sklearn.metrics import confusion_matrix
import seaborn as sn
CFMt = confusion_matrix(y_train, pred1)
CFM = confusion_matrix(y_test, pred)
df_cm = pd.DataFrame(CFMt, range(2), range(2))
#plt.figure(figsize = (10,7))
sn.set(font_scale=1.4)#for label size
sn.heatmap(df_cm, annot=True,annot_kws={"size": 16})
```

Out[77]:

```
<matplotlib.axes._subplots.AxesSubplot at 0x27420255be0>
```

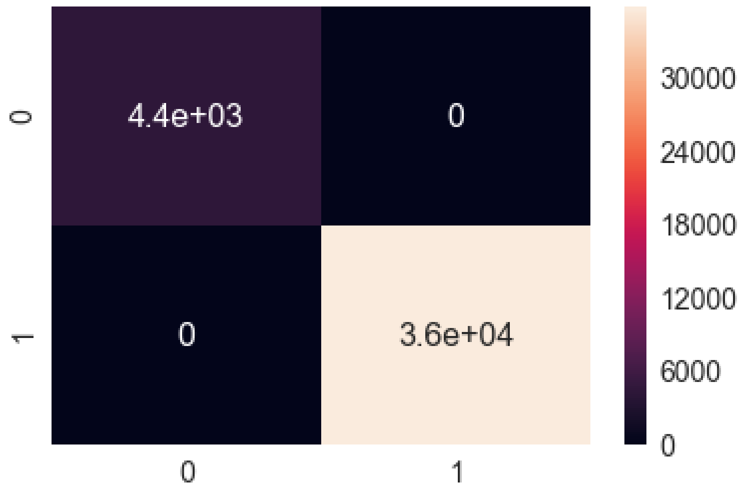


In [78]:

```
df_cm = pd.DataFrame(CFMtr, range(2), range(2))
```

```
#plt.figure(figsize = (10,7))
sn.set(font_scale=1.4)#for label size
sn.heatmap(df_cm, annot=True,annot_kws={"size": 16})
```

Out[78]:
```
<matplotlib.axes._subplots.AxesSubplot at 0x2741fd45198>
```



In [103]:
```
from sklearn.metrics import accuracy_score, f1_score, precision_score,
recall_score
print(accuracy_score(y_test, pred))
print(f1_score(y_test, pred, average="macro"))
print(float(1-f1_score(y_test, pred, average="macro")))
print(precision_score(y_test, pred, average="macro"))
print(recall_score(y_test, pred, average="macro"))
```

```
0.7189
0.6040944046821365
0.3959055953178635
0.6140110370642459
0.7717477688749919
```

In [59]:
```
from sklearn import preprocessing
from sklearn import svm
import scipy.stats as stats
tuned_parameters ={'C': np.random.uniform(low=10**-4, high=10**4, size=
5),'gamma': np.random.uniform(low=10**-4, high=10**4, size=5)}
#lb = preprocessing.LabelBinarizer()
#y_train = np.array([number[0] for number in lb.fit_transform(y_trai
n)])
#Using RandomizedSearchCV
modelw2vr2 = RandomizedSearchCV(svm.SVC(kernel='rbf',class_weight='bala
nced'),tuned_parameters,n_iter=5 ,scoring = 'roc_auc', cv=5)
modelw2vr2.fit(tfidf_sent_vectorss, y_train)
#y_test = np.array([number[0] for number in lb.fit_transform(y_test)])
print(modelw2vr2.best_estimator_)
print(modelw2vr2.score(tfidf_sent_vectors_tests, y_test))
```

```
SVC(C=1704.01534350303, cache_size=200, class_weight='balanced', coef0=
0.0,
  decision_function_shape='ovr', degree=3, gamma=3610.027190026463,
  kernel='rbf', max_iter=-1, probability=False, random_state=None,
  shrinking=True, tol=0.001, verbose=False)
0.8080191779037196
```

In [61]:

```
from sklearn import svm
modelsvc3 = svm.SVC(C=1704.01534350303, cache_size=200, class_weight='b
alanced', coef0=0.0,
  decision_function_shape='ovr', degree=3, gamma=3610.027190026463,
  kernel='rbf', max_iter=-1, probability=False, random_state=None,
  shrinking=True, tol=0.001, verbose=False)
modelsvc3.fit(tfidf_sent_vectorss, y_train)
print(modelsvc3.score(tfidf_sent_vectors_tests, y_test))
```

```
0.9336
```

In [68]:

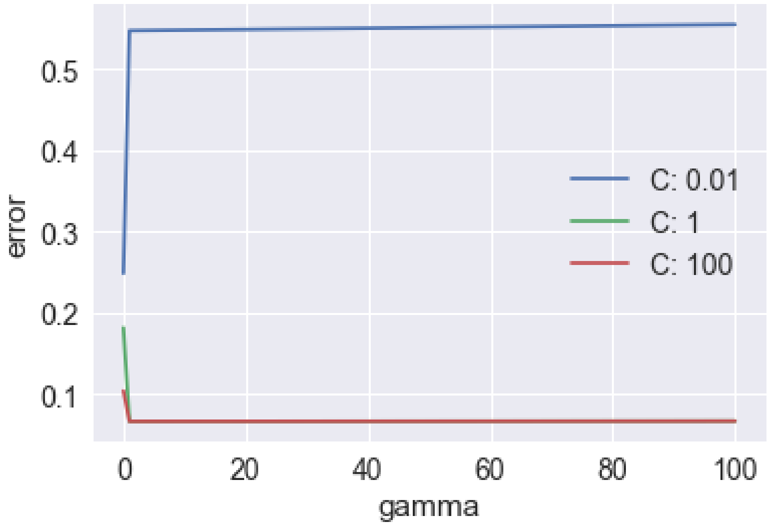
```
tuned_parameters = [10**-2, 10**0, 10**2]
ERROR=[]
for c in tuned_parameters:
    for gamma in tuned_parameters:
        print("for c= ", c)
        print("for gamma= ", gamma)
        modelsvc1 = svm.SVC(C=c, cache_size=200, class_weight='balance
d', coef0=0.0,
        decision_function_shape='ovr', degree=3, gamma=gamma,
        kernel='rbf', max_iter=-1, probability=False, random_state=None
,
        shrinking=True, tol=0.001, verbose=False)
        modelsvc1.fit(tfidf_sent_vectorss, y_train)
        f1_score=modelsvc1.score(tfidf_sent_vectors_tests, y_test)
        error = float(1-f1_score)
        print("error=",error)
        ERROR.append(error)
```

```
for c=  0.01
for gamma=  0.01
error= 0.24829999999999997
for c=  0.01
for gamma=  1
error= 0.5470999999999999
for c=  0.01
for gamma=  100
error= 0.5547
for c=  1
for gamma=  0.01
error= 0.18110000000000004
for c=  1
for gamma=  1
error= 0.06599999999999995
for c=  1
for gamma=  100
error= 0.06640000000000001
for c=  100
for gamma=  0.01
error= 0.10370000000000001
for c=  100
for gamma=  1
error= 0.06599999999999995
for c=  100
for gamma=  100
error= 0.06640000000000001
```

In [69]:

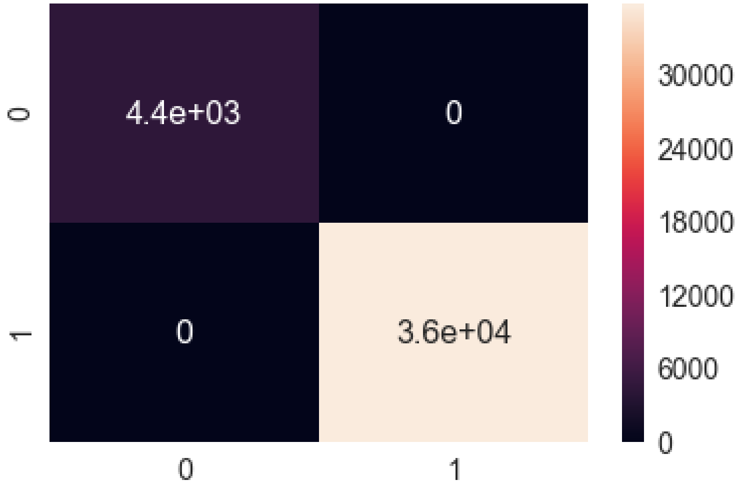
```
scores = np.array(ERROR).reshape(len(tuned_parameters), len(tuned_param
eters))
```

```
for ind, i in enumerate(tuned_parameters):
    plt.plot(tuned_parameters,scores[ind], label='C: ' + str(i))
plt.legend()
plt.xlabel('gamma')
plt.ylabel('error')
plt.show()
```



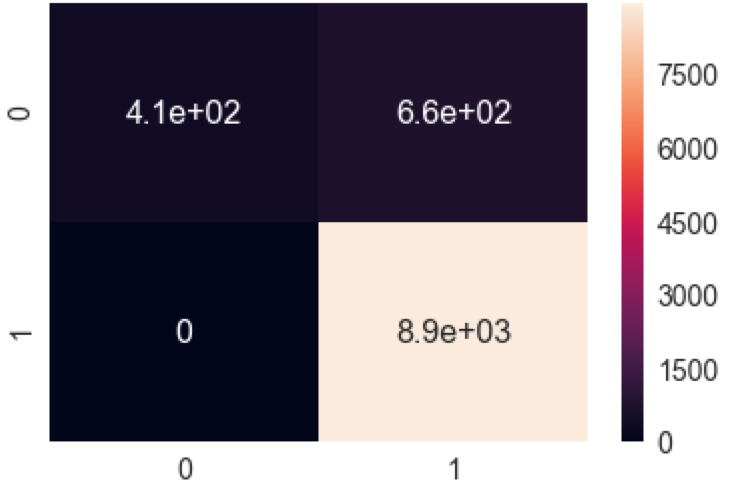
In [79]:
```
pred = modelsvc3.predict(tfidf_sent_vectors_tests)
pred1 = modelsvc3.predict(tfidf_sent_vectorss)
from sklearn.metrics import confusion_matrix
import seaborn as sn
CFMt = confusion_matrix(y_train, pred1)
CFM = confusion_matrix(y_test, pred)
df_cm = pd.DataFrame(CFMt, range(2), range(2))
#plt.figure(figsize = (10,7))
sn.set(font_scale=1.4)#for label size
sn.heatmap(df_cm, annot=True,annot_kws={"size": 16})
```

Out[79]:
```
<matplotlib.axes._subplots.AxesSubplot at 0x27420f57208>
```



```
In [81]: df_cm = pd.DataFrame(CFM, range(2), range(2))
         #plt.figure(figsize = (10,7))
         sn.set(font_scale=1.4)#for label size
         sn.heatmap(df_cm, annot=True,annot_kws={"size": 16})
```

```
Out[81]: <matplotlib.axes._subplots.AxesSubplot at 0x27420b730b8>
```



```
In [63]: TP = CFM[0][0]
```

```
FP = CFM[1][0]
FN = CFM[0][1]
TN = CFM[1][1]
P = TP+FP
N = TN+FN
print('TP Value is',TP )
print('FP Value is',FP )
print('TN Value is',TN )
print('FN Value is',FN )

TPR = float(TP/P)
FPR = float(FP/P)
TNR = float(TN/N)
FNR = float(FN/N)
print('TPR Value is',TPR )
print('FPR Value is',FPR )
print('TNR Value is',TNR )
print('FNR Value is',FNR )
```

```
TP Value is 411
FP Value is 0
TN Value is 8925
FN Value is 664
TPR Value is 1.0
FPR Value is 0.0
TNR Value is 0.9307539889456669
FNR Value is 0.06924601105433309
```

In [65]:

```
pred = modelsvc3.predict(tfidf_sent_vectors_tests)
from sklearn.metrics import accuracy_score, f1_score, precision_score, recall_score
print(accuracy_score(y_test, pred))
print(f1_score(y_test, pred, average="macro"))
print(float(1-f1_score(y_test, pred, average="macro")))
print(precision_score(y_test, pred, average="macro"))
print(recall_score(y_test, pred, average="macro"))
```

```
0.9336
0.7586490511490995
0.24135094885090047
```

```
0.9653769944728334
0.6911627906976744
```

In [3]:
```
from prettytable import PrettyTable
x = PrettyTable(["Table", "BOW", "TF-IDF","W2V","TFIDF W2V"])
while True:
    #- Get value
    prompt = input("Please add a head to the list\n")

    try:
         #- Type Casting.
        prompt1 = float(input("Please add a BOW to the list\n"))
        prompt2 = float(input("Please enter a TF-IDF for the service\n"
))
        prompt3 = float(input("Please enter a W2V for the service\n"))
        prompt4 = float(input("Please enter a TFIDF W2V for the service
\n"))
    except ValueError:
        print("Please enter valid type")
        continue
    #- Add row
    x.add_row([ prompt,prompt1, prompt2,prompt3,prompt4])
    #- Ask user to Continue or not.
    choice = input("Continue yes/ no:").lower()
    if not(choice=="yes" or choice=="y"):
        break
```

```
Please add a head to the list
SGD Train error
Please add a BOW to the list
0.099
Please enter a TF-IDF for the service
0.057
Please enter a W2V for the service
0.11
Please enter a TFIDF W2V for the service
0.14
Continue yes/ no:y
Please add a head to the list
SGD Test error
SGD Test error
Please add a BOW to the list
0.106
Please enter a TF-IDF for the service
0.07
Please enter a W2V for the service
0.175
Please enter a TFIDF W2V for the service
0.25
Continue yes/ no:y
Please add a head to the list
SVC Train error
Please add a BOW to the list
0.2
Please enter a TF-IDF for the service
0.08
Please enter a W2V for the service
0.07
Please enter a TFIDF W2V for the service
0.2
Continue yes/ no:y
Please add a head to the list
SVC Test error
Please add a BOW to the list
0.06
Please enter a TF-IDF for the service
0.07
Please enter a W2V for the service
0.07
Please enter a TFIDF W2V for the service
0.07
Continue yes/ no:n
```

In [4]:
```
print(x)
```

```
+-----------------+-------+--------+-------+-----------+
|      Table      |  BOW  | TF-IDF |  W2V  | TFIDF W2V |
+-----------------+-------+--------+-------+-----------+
| SGD Train error | 0.099 | 0.057  |  0.11 |    0.14   |
|  SGD Test error | 0.106 |  0.07  | 0.175 |    0.25   |
| SVC Train error |  0.2  |  0.08  |  0.07 |    0.2    |
|  SVC Test error |  0.06 |  0.07  |  0.07 |    0.07   |
+-----------------+-------+--------+-------+-----------+
```