Random Forest and GBDT

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)
```

PREPROCESSING

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
```

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-pro
cessing 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 [6]:
```
filtered_data['CleanedText']=final_string #adding a column of CleanedTe
xt which displays the data after pre-processing of the review
filtered_data['CleanedText']=filtered_data['CleanedText'].str.decode("u
tf-8")
```

In [7]:
```
sorted_data=filtered_data.sort_values(by=['Time'])
sampledata = sorted_data.head(100000)

S = sorted_data['Score']
Score = S.head(100000)
```

Splitting

In [8]:
```
# HERE WE ARE SPLITTING THE DATA POINTS IN TO 70% TRAIN AND 30% FOR TES
T
X_1, X_test, y_1, y_test = cross_validation.train_test_split(sampledata
, Score, test_size=0.3, random_state=0)
#HERE WE ARE AGAIN SPLITTING THE TRAIN DATA IN EARLIER LINE X_1 IN TO 7
0% TRAINING AND 30% CROSS VALIDATION DATA
X_tr, X_cv, y_tr, y_cv = cross_validation.train_test_split(X_1, y_1, te
st_size=0.3)
```

RANDOM FOREST AND GBDT FOR BAG OF WORDS

In [9]:
```
count_vect = CountVectorizer(min_df=10) #in scikit-learn
vec = count_vect.fit(X_tr['CleanedText'].values)
```

In [10]:
```
X_trvec = vec.transform(X_tr['CleanedText'].values)
X_cvvec = vec.transform(X_cv['CleanedText'].values)
```

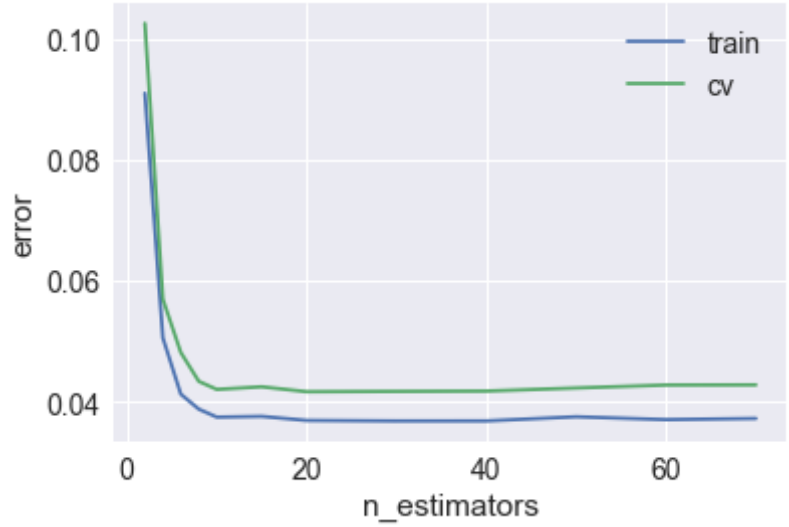
```
X_testvec = vec.transform(X_test['CleanedText'].values)
```

In [24]:
```
from sklearn import preprocessing
from sklearn import ensemble


hyperparameter = dict(n_estimators=[2,4,6,8,10,15,20,30,40,50,60,70])

#Using GridSearchCV
model = GridSearchCV(ensemble.RandomForestClassifier(class_weight='balanced'), hyperparameter, scoring = 'f1', cv=5)
model.fit(X_trvec, y_tr)
a=model.grid_scores_
print(model.best_estimator_)
print(model.score(X_testvec, y_test))

model.fit(X_cvvec, y_cv)
b=model.grid_scores_
```

```
RandomForestClassifier(bootstrap=True, class_weight='balanced',
            criterion='gini', max_depth=None, max_features='auto',
            max_leaf_nodes=None, min_impurity_decrease=0.0,
            min_impurity_split=None, min_samples_leaf=1,
            min_samples_split=2, min_weight_fraction_leaf=0.0,
            n_estimators=30, n_jobs=1, oob_score=False, random_state=None,
            verbose=0, warm_start=False)
0.9678453078911912
```

In [25]:
```
scores = [x[1] for x in a]
cv_scores = [x[1] for x in b]
scores = np.array(scores).reshape(len(hyperparameter['n_estimators']),1)
cv_scores = np.array(cv_scores).reshape(len(hyperparameter['n_estimators']),1)
#for i in enumerate(hyperparameter['max_depth']):
plt.plot(hyperparameter['n_estimators'],1- scores,label='train')
plt.plot(hyperparameter['n_estimators'],1- cv_scores,label='cv')
#plt.legend()
```

```
plt.xlabel('n_estimators')
plt.ylabel('error')
plt.legend()
plt.show()
```

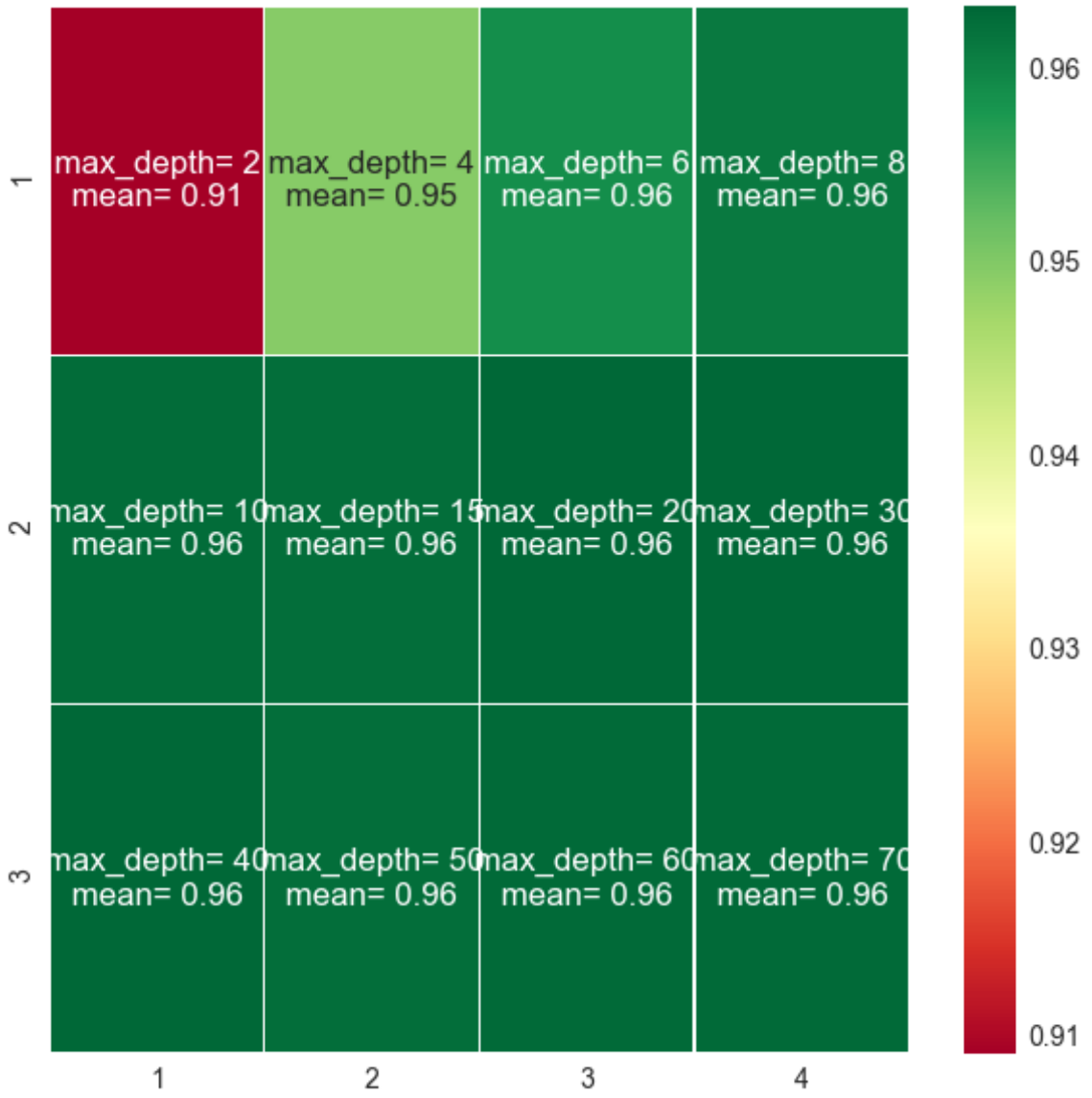


In [26]:
```
n_estimators=[]
mean=[]
for a in a:
    n_estimators.append(a[0]['n_estimators'])
    mean.append(a[1])
n_estimators=np.asarray(n_estimators)
mean=np.asarray(mean)
n_estimators = n_estimators.reshape(3,4)

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

label =np.asarray([" max_depth= {0} \n mean= {1:.2f} ".format(n_estimat
ors,mean) for n_estimators,mean in zip(n_estimators.flatten(),mean.flat
ten())]).reshape(3,4)
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 )
```

```
[[' max_depth= 2 \n mean= 0.91 ' ' max_depth= 4 \n mean= 0.95 '
  ' max_depth= 6 \n mean= 0.96 ' ' max_depth= 8 \n mean= 0.96 ']
 [' max_depth= 10 \n mean= 0.96 ' ' max_depth= 15 \n mean= 0.96 '
  ' max_depth= 20 \n mean= 0.96 ' ' max_depth= 30 \n mean= 0.96 ']
 [' max_depth= 40 \n mean= 0.96 ' ' max_depth= 50 \n mean= 0.96 '
  ' max_depth= 60 \n mean= 0.96 ' ' max_depth= 70 \n mean= 0.96 ']]
```

Out[26]: <matplotlib.axes._subplots.AxesSubplot at 0x1e699db0400>



In [156]:
```
clf = ensemble.RandomForestClassifier(bootstrap=True, class_weight='bal
anced',
            criterion='gini', max_depth=None, max_features='auto',
            max_leaf_nodes=None, min_impurity_decrease=0.0,
```

```
            min_impurity_split=None, min_samples_leaf=1,
            min_samples_split=2, min_weight_fraction_leaf=0.0,
            n_estimators=30, n_jobs=1, oob_score=False, random_state=No
ne,
            verbose=0, warm_start=False)
clf=clf.fit(X_trvec, y_tr)
print("train-error = ",1-clf.score(X_trvec, y_tr))
print("test-error = ",1-clf.score(X_testvec, y_test))
```

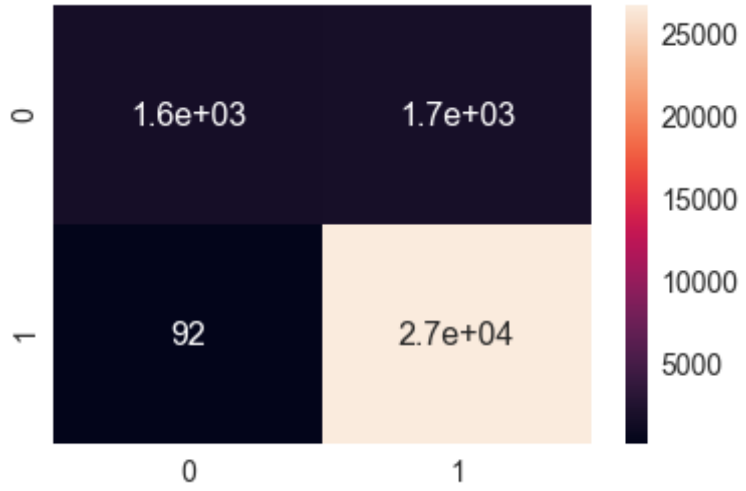
```
train-error =  0.0002448979591836986
test-error =  0.05953333333333333
```

In [158]:

```
importances=clf.feature_importances_
feat_names=count_vect.get_feature_names()
# Sort feature importances in descending order
indices = np.argsort(importances)[::-1][:25]
a=np.take(feat_names,indices)
def againcleaning(X):

    comment_words=' '
    for words in X:

        comment_words = comment_words + words + ' '
    return comment_words
a=againcleaning(a)
from wordcloud import WordCloud, STOPWORDS
import matplotlib.pyplot as plt
word_cloud=WordCloud(background_color='black',stopwords=stop,
                          width=1200,
                          height=1000).generate(a)
plt.imshow(word_cloud)
plt.axis("off")
plt.show()
```

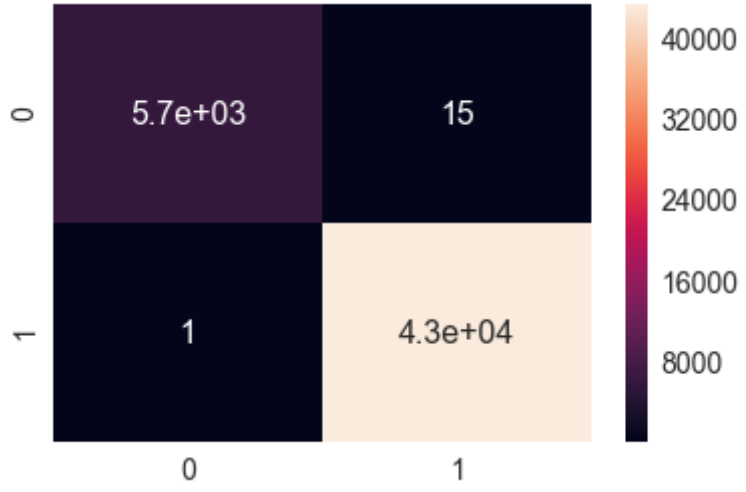
In [29]:
```
pred = clf.predict(X_testvec)
pred1 = clf.predict(X_trvec)
from sklearn.metrics import confusion_matrix
import seaborn as sn
CFM = confusion_matrix(y_test, pred)
CFMtr = confusion_matrix(y_tr, pred1)
df_cm = pd.DataFrame(CFM, range(2), range(2))
#plt.figure(figsize = (10,7))
sn.set(font_scale=1.4)#for label size
sn.heatmap(df_cm, annot=True,annot_kws={"size": 16})
```

Out[29]:
```
<matplotlib.axes._subplots.AxesSubplot at 0x1e6996bd0b8>
```



In [30]:
```
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[30]: <matplotlib.axes._subplots.AxesSubplot at 0x1e686fa2630>



In [31]:
```
from sklearn.metrics import accuracy_score, f1_score, precision_score,
```

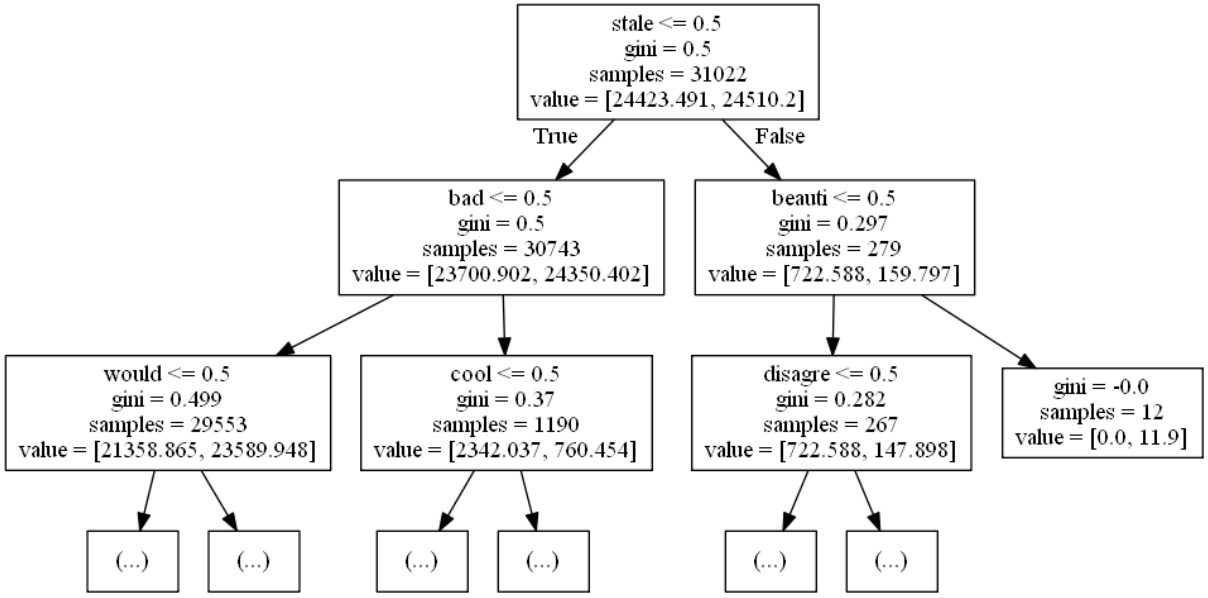
```
recall_score
print(accuracy_score(y_test, pred))
print(f1_score(y_test, pred, average="macro"))
print(precision_score(y_test, pred, average="macro"))
print(recall_score(y_test, pred, average="macro"))
```

```
0.9398
0.8051053898539693
0.9429372256597388
0.7417644769818345
```

In [159]:
```
from sklearn import tree
import os
import graphviz
from sklearn.externals.six import StringIO
os.environ["PATH"] += os.pathsep + r'C:\Users\krush\Anaconda3\Lib\site-packages\graphviz'
#os.path.abspath('C:\\\Users\\\krush\\\Anaconda3\\\Lib\\\site-packages\\\graphviz')
dot_data1 = StringIO()
dot_data = tree.export_graphviz(clf.estimators_[0],feature_names=feat_names, out_file=dot_data1,
                                max_depth = 2)
#graph = graphviz.Source(dot_data)
```

In [160]:
```
from IPython.display import Image
import pydot
graph = pydot.graph_from_dot_data(dot_data1.getvalue())[0]
Image(graph.create_png())
```

Out[160]:



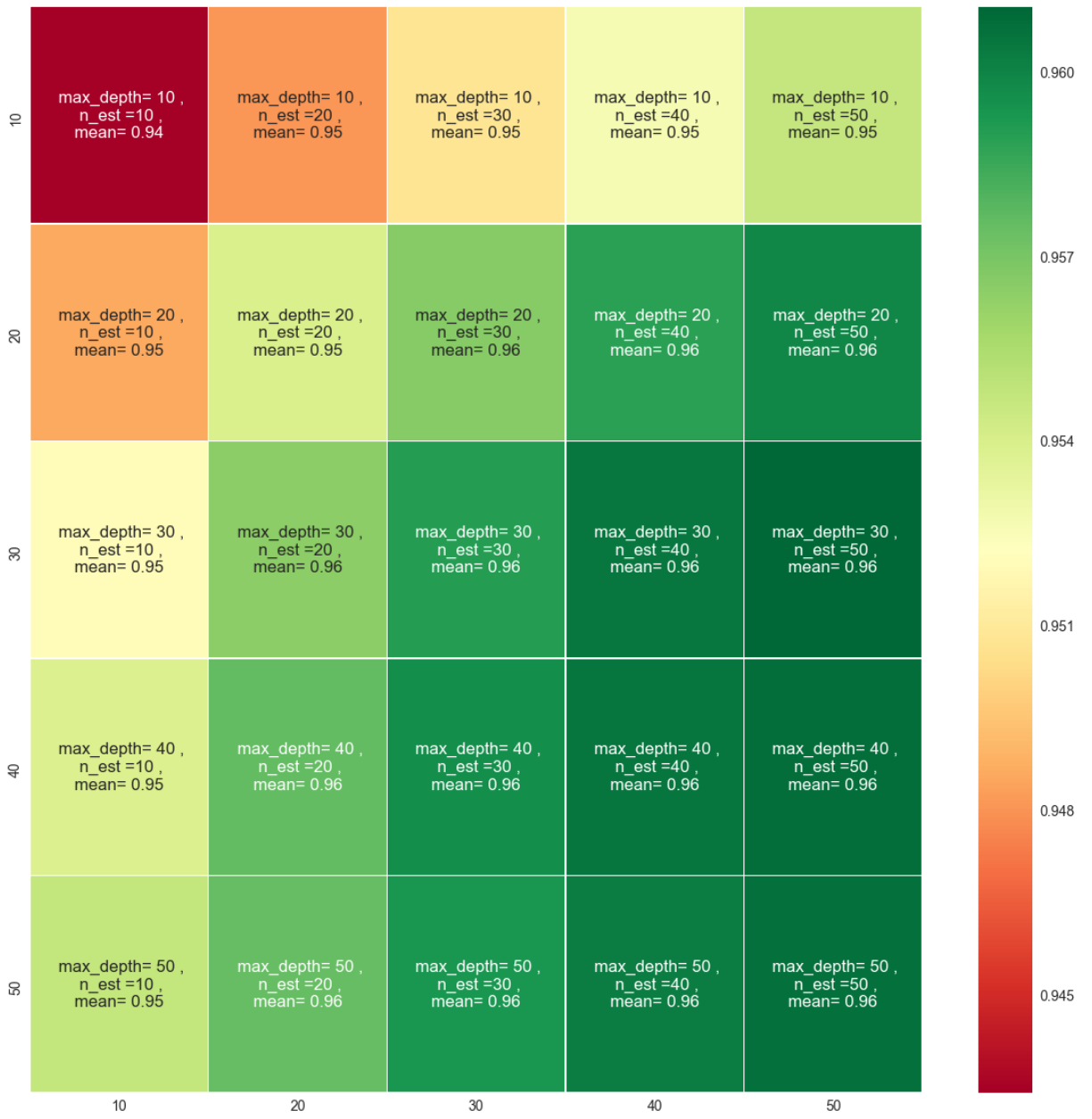
In [49]:
```
from sklearn import preprocessing
from sklearn import ensemble


hyperparameter = dict(n_estimators=[10,20,30,40,50],max_depth=[10,20,30
,40,50])

#Using GridSearchCV
model1 = GridSearchCV(ensemble.GradientBoostingClassifier(), hyperparam
eter, scoring = 'f1', cv=5)
model1.fit(X_trvec, y_tr)
a=model1.grid_scores_
print(model1.best_estimator_)
print(model1.score(X_testvec, y_test))
```

```
GradientBoostingClassifier(criterion='friedman_mse', init=None,
              learning_rate=0.1, loss='deviance', max_depth=30,
              max_features=None, max_leaf_nodes=None,
              min_impurity_decrease=0.0, min_impurity_split=None,
              min_samples_leaf=1, min_samples_split=2,
              min_weight_fraction_leaf=0.0, n_estimators=50,
```

```
                       presort='auto', random_state=None, subsample=1.0, verbose
=0,
                       warm_start=False)
0.9657302652020543
```

In [84]:
```
max_depth=[]
n_estimators = []
mean=[]
for a1 in a:
    max_depth.append(a1[0]['max_depth'])
    n_estimators.append(a1[0]['n_estimators'])
    mean.append(a1[1])
max_depth=np.asarray(max_depth)
n_estimators=np.asarray(n_estimators)
mean=np.asarray(mean)
max_depth = max_depth.reshape(5,5)
n_estimators = n_estimators.reshape(5,5)
mean = mean.reshape(5,5)
result = pd.DataFrame(mean,index=[10,20,30,40,50],columns = [10,20,30,4
0,50])
label =np.asarray([" max_depth= {0} ,\n n_est ={1} ,\n mean= {2:.2f} ".
format(max_depth,n_estimators,mean) for max_depth,n_estimators,mean in
zip(max_depth.flatten(),n_estimators.flatten(),mean.flatten())]).reshap
e(5,5)
import seaborn as sns
fig , ax = plt.subplots(figsize = (20,20))
ax.set_xticks([])
ax.set_yticks([])
sns.heatmap(result,annot=label,fmt="" ,cmap ='RdYlGn',linewidths = 0.30
 , ax = ax )
```

Out[84]:
```
<matplotlib.axes._subplots.AxesSubplot at 0x1e69abf8400>
```



In [161]:
```
clf = ensemble.GradientBoostingClassifier(criterion='friedman_mse', ini
```

```
t=None,
              learning_rate=0.1, loss='deviance', max_depth=30,
              max_features=None, max_leaf_nodes=None,
              min_impurity_decrease=0.0, min_impurity_split=None,
              min_samples_leaf=1, min_samples_split=2,
              min_weight_fraction_leaf=0.0, n_estimators=50,
              presort='auto', random_state=None, subsample=1.0, verbose
=0,
              warm_start=False)
clf=clf.fit(X_trvec, y_tr)
print("train-error = ",1-clf.score(X_trvec, y_tr))
print("test-error = ",1-clf.score(X_testvec, y_test))
```

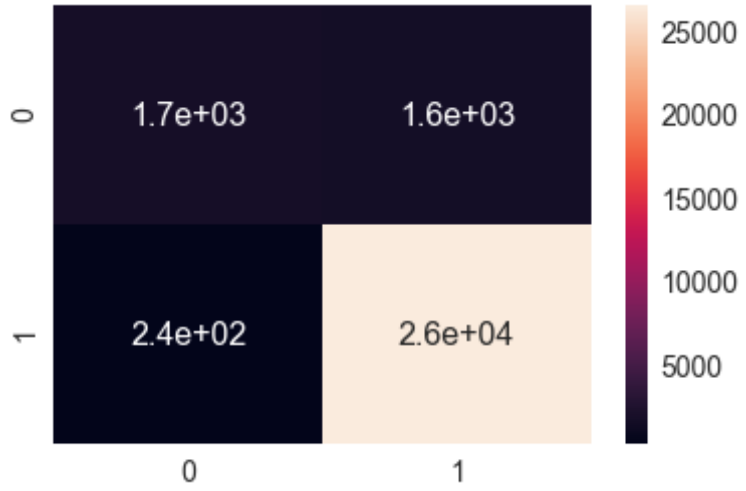
```
train-error =  0.011755102040816312
test-error =  0.06293333333333329
```

In [162]:

```
importances=clf.feature_importances_
feat_names=count_vect.get_feature_names()
# Sort feature importances in descending order
indices = np.argsort(importances)[::-1][:25]
a=np.take(feat_names,indices)
def againcleaning(X):

    comment_words=' '
    for words in X:

        comment_words = comment_words + words + ' '
    return comment_words
a=againcleaning(a)
from wordcloud import WordCloud, STOPWORDS
import matplotlib.pyplot as plt
word_cloud=WordCloud(background_color='black',stopwords=stop,
                          width=1200,
                          height=1000).generate(a)
plt.imshow(word_cloud)
plt.axis("off")
plt.show()
```

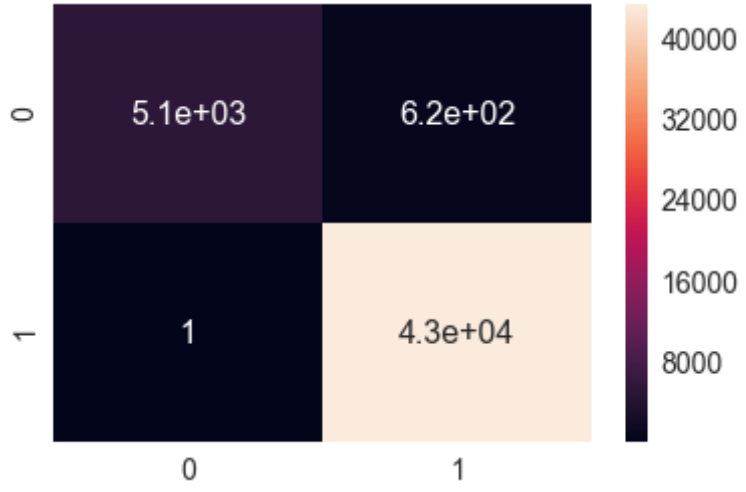
In [87]:
```
pred = clf.predict(X_testvec)
pred1 = clf.predict(X_trvec)
from sklearn.metrics import confusion_matrix
import seaborn as sn
CFM = confusion_matrix(y_test, pred)
CFMtr = confusion_matrix(y_tr, pred1)
df_cm = pd.DataFrame(CFM, range(2), range(2))
#plt.figure(figsize = (10,7))
sn.set(font_scale=1.4)#for label size
sn.heatmap(df_cm, annot=True,annot_kws={"size": 16})
```

Out[87]:
```
<matplotlib.axes._subplots.AxesSubplot at 0x1e69a9b3dd8>
```



In [88]:
```
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[88]: <matplotlib.axes._subplots.AxesSubplot at 0x1e69bb5d1d0>



In [89]:
```
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"))
```

```
0.9378333333333333
0.8067785676963858
0.9100330347039736
0.7520462436003329
```

RANDOM FOREST AND GBDT FOR TF-IDF

In [97]:
```
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_tr['CleanedText'].values)
final_tf_idf1 = tf_idf_vect1.fit(X_tr['CleanedText'].values)
```

In [98]:
```
X_tr_tf_idf = final_tf_idf.transform(X_tr['CleanedText'].values)
X_test_tf_idf = final_tf_idf.transform(X_test['CleanedText'].values)
X_cv_tf_idf = final_tf_idf.transform(X_cv['CleanedText'].values)
```

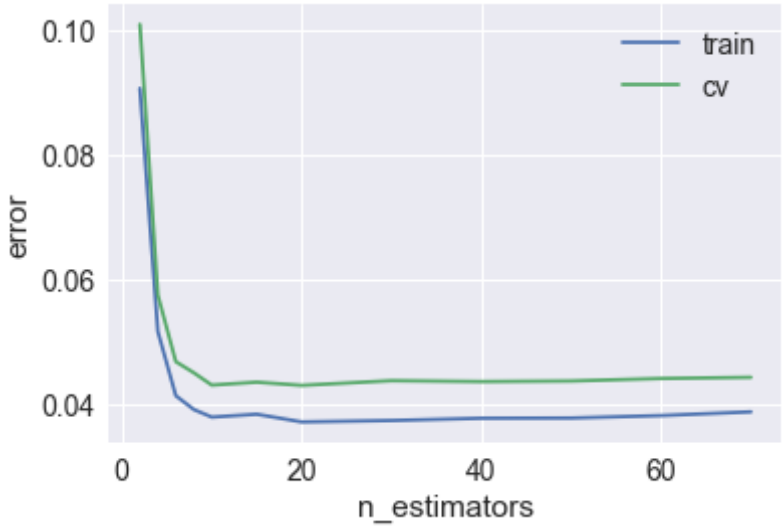
In [99]:
```
from sklearn import preprocessing
from sklearn import ensemble


hyperparameter = dict(n_estimators=[2,4,6,8,10,15,20,30,40,50,60,70])

#Using GridSearchCV
model = GridSearchCV(ensemble.RandomForestClassifier(class_weight='bala
nced'), hyperparameter, scoring = 'f1', cv=5)
model.fit(X_tr_tf_idf, y_tr)
a=model.grid_scores_
print(model.best_estimator_)
print(model.score(X_test_tf_idf, y_test))
```

```
model.fit(X_cv_tf_idf, y_cv)
b=model.grid_scores_
```

```
RandomForestClassifier(bootstrap=True, class_weight='balanced',
            criterion='gini', max_depth=None, max_features='auto',
            max_leaf_nodes=None, min_impurity_decrease=0.0,
            min_impurity_split=None, min_samples_leaf=1,
            min_samples_split=2, min_weight_fraction_leaf=0.0,
            n_estimators=20, n_jobs=1, oob_score=False, random_state=No
ne,
            verbose=0, warm_start=False)
0.9665144873497153
```

In [100]:

```
scores = [x[1] for x in a]
cv_scores = [x[1] for x in b]
scores = np.array(scores).reshape(len(hyperparameter['n_estimators']),1
)
cv_scores = np.array(cv_scores).reshape(len(hyperparameter['n_estimator
s']),1)
#for i in enumerate(hyperparameter['max_depth']):
plt.plot(hyperparameter['n_estimators'],1- scores,label='train')
plt.plot(hyperparameter['n_estimators'],1- cv_scores,label='cv')
#plt.legend()
plt.xlabel('n_estimators')
plt.ylabel('error')
plt.legend()
plt.show()
```

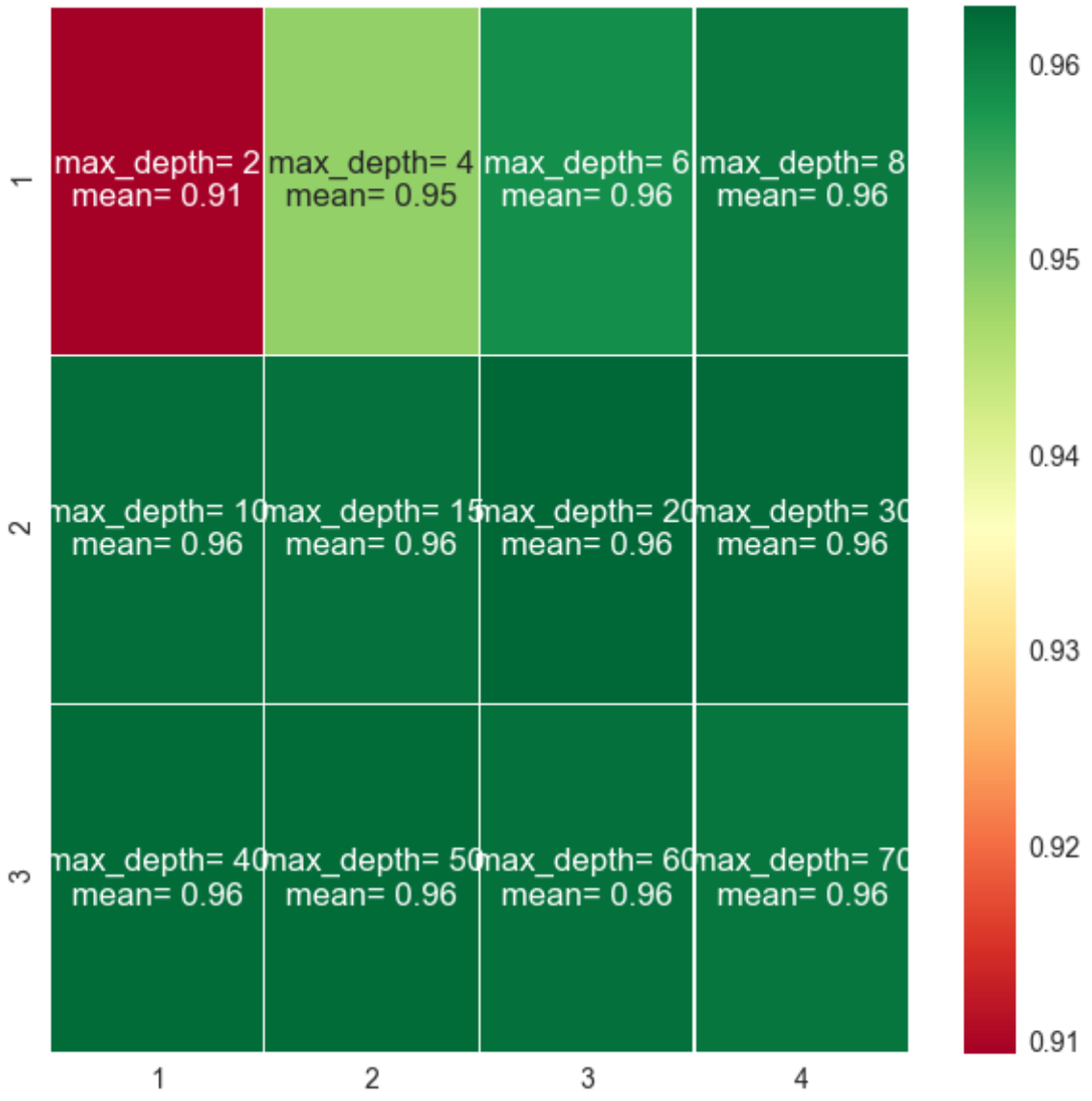


In [101]:
```
n_estimators=[]
mean=[]
for a in a:
    n_estimators.append(a[0]['n_estimators'])
    mean.append(a[1])
n_estimators=np.asarray(n_estimators)
mean=np.asarray(mean)
n_estimators = n_estimators.reshape(3,4)

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

label =np.asarray([" max_depth= {0} \n mean= {1:.2f} ".format(n_estimat
ors,mean) for n_estimators,mean in zip(n_estimators.flatten(),mean.flat
ten())]).reshape(3,4)
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 )
```

```
[[' max_depth= 2 \n mean= 0.91 ' ' max_depth= 4 \n mean= 0.95 '
  ' max_depth= 6 \n mean= 0.96 ' ' max_depth= 8 \n mean= 0.96 ']
 [' max_depth= 10 \n mean= 0.96 ' ' max_depth= 15 \n mean= 0.96 '
  ' max_depth= 20 \n mean= 0.96 ' ' max_depth= 30 \n mean= 0.96 ']
 [' max_depth= 40 \n mean= 0.96 ' ' max_depth= 50 \n mean= 0.96 '
  ' max_depth= 60 \n mean= 0.96 ' ' max_depth= 70 \n mean= 0.96 ']]
```

Out[101]: <matplotlib.axes._subplots.AxesSubplot at 0x1e6a8913358>



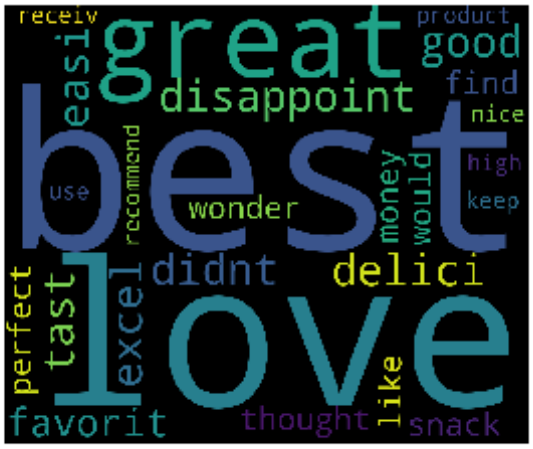
In [163]:
```
clf = ensemble.RandomForestClassifier(bootstrap=True, class_weight='bal
anced',
            criterion='gini', max_depth=None, max_features='auto',
```

```
            max_leaf_nodes=None, min_impurity_decrease=0.0,
            min_impurity_split=None, min_samples_leaf=1,
            min_samples_split=2, min_weight_fraction_leaf=0.0,
            n_estimators=20, n_jobs=1, oob_score=False, random_state=No
ne,
            verbose=0, warm_start=False)
clf=clf.fit(X_tr_tf_idf, y_tr)
print("train-error = ",1-clf.score(X_tr_tf_idf, y_tr))
print("test-error = ",1-clf.score(X_test_tf_idf, y_test))
```

```
train-error =  0.0008571428571428896
test-error =  0.059833333333333294
```

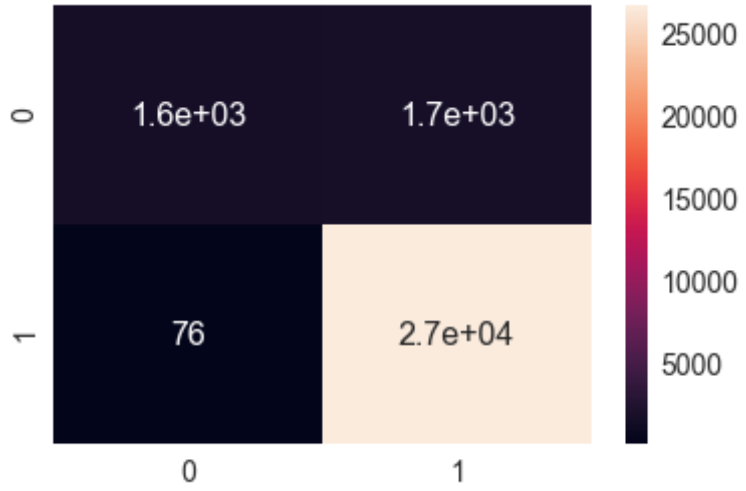
In [164]:

```
importances=clf.feature_importances_
feat_names=tf_idf_vect.get_feature_names()
# Sort feature importances in descending order
indices = np.argsort(importances)[::-1][:25]
a=np.take(feat_names,indices)
def againcleaning(X):

    comment_words=' '
    for words in X:

        comment_words = comment_words + words + ' '
    return comment_words
a=againcleaning(a)
from wordcloud import WordCloud, STOPWORDS
import matplotlib.pyplot as plt
word_cloud=WordCloud(background_color='black',stopwords=stop,
                          width=1200,
                          height=1000).generate(a)
plt.imshow(word_cloud)
plt.axis("off")
plt.show()
```



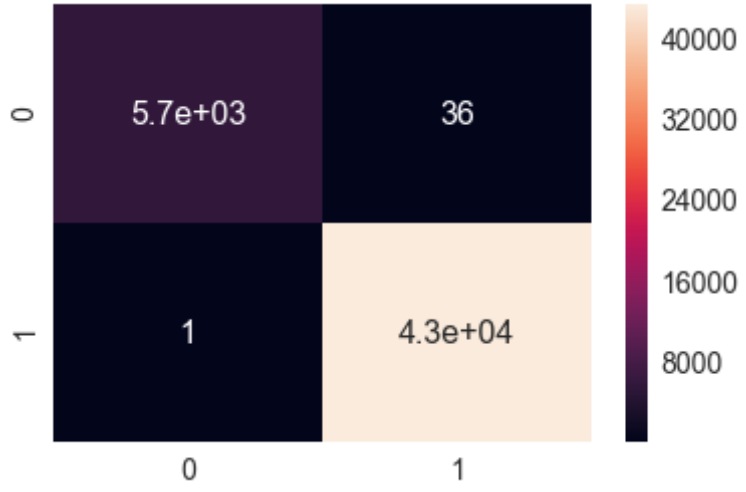
In [105]:
```
pred = clf.predict(X_test_tf_idf)
pred1 = clf.predict(X_tr_tf_idf)
from sklearn.metrics import confusion_matrix
import seaborn as sn
CFM = confusion_matrix(y_test, pred)
CFMtr = confusion_matrix(y_tr, pred1)
df_cm = pd.DataFrame(CFM, range(2), range(2))
#plt.figure(figsize = (10,7))
sn.set(font_scale=1.4)#for label size
sn.heatmap(df_cm, annot=True,annot_kws={"size": 16})
```

Out[105]: <matplotlib.axes._subplots.AxesSubplot at 0x1e6a8aa1208>



In [106]:
```
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[106]: <matplotlib.axes._subplots.AxesSubplot at 0x1e6a98ea898>



In [107]:
```
from sklearn.metrics import accuracy_score, f1_score, precision_score,
```

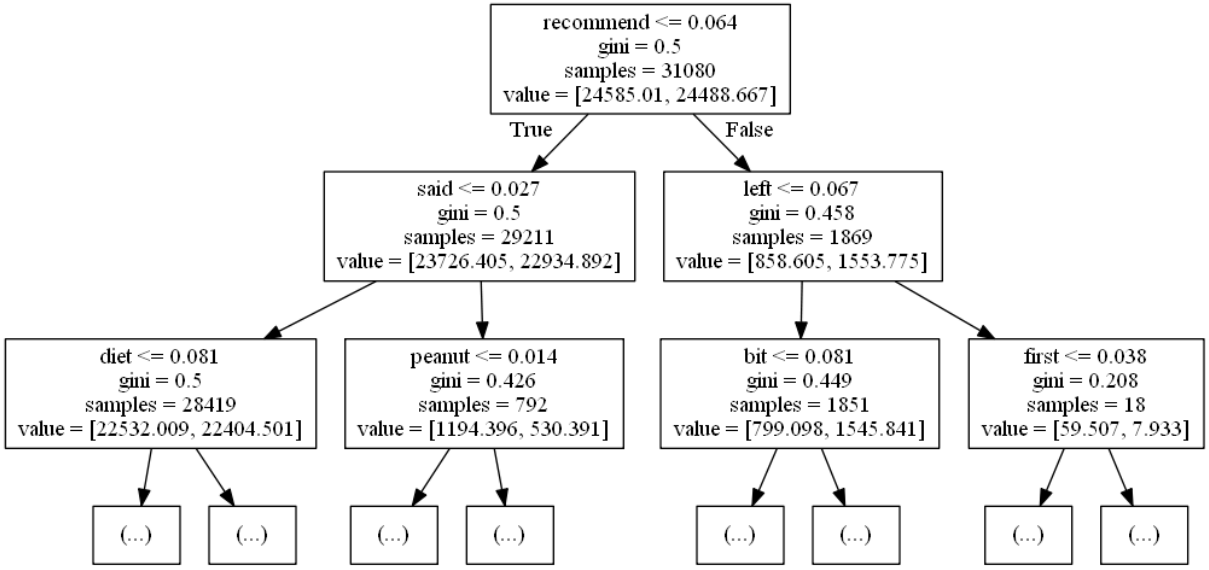
```
recall_score
print(accuracy_score(y_test, pred))
print(f1_score(y_test, pred, average="macro"))
print(precision_score(y_test, pred, average="macro"))
print(recall_score(y_test, pred, average="macro"))
```

```
0.9399333333333333
0.8045545909757814
0.947043085118839
0.7402686937564888
```

In [165]:

```
from sklearn import tree
import os
import graphviz
from sklearn.externals.six import StringIO
os.environ["PATH"] += os.pathsep + r'C:\Users\krush\Anaconda3\Lib\site-packages\graphviz'
#os.path.abspath('C:\\\Users\\\krush\\\Anaconda3\\\Lib\\\site-packages\\\graphviz')
dot_data1 = StringIO()
dot_data = tree.export_graphviz(clf.estimators_[0],feature_names=feat_names, out_file=dot_data1,
                                max_depth = 2)
#graph = graphviz.Source(dot_data)

from IPython.display import Image
import pydot
graph = pydot.graph_from_dot_data(dot_data1.getvalue())[0]
Image(graph.create_png())
```

Out[165]:



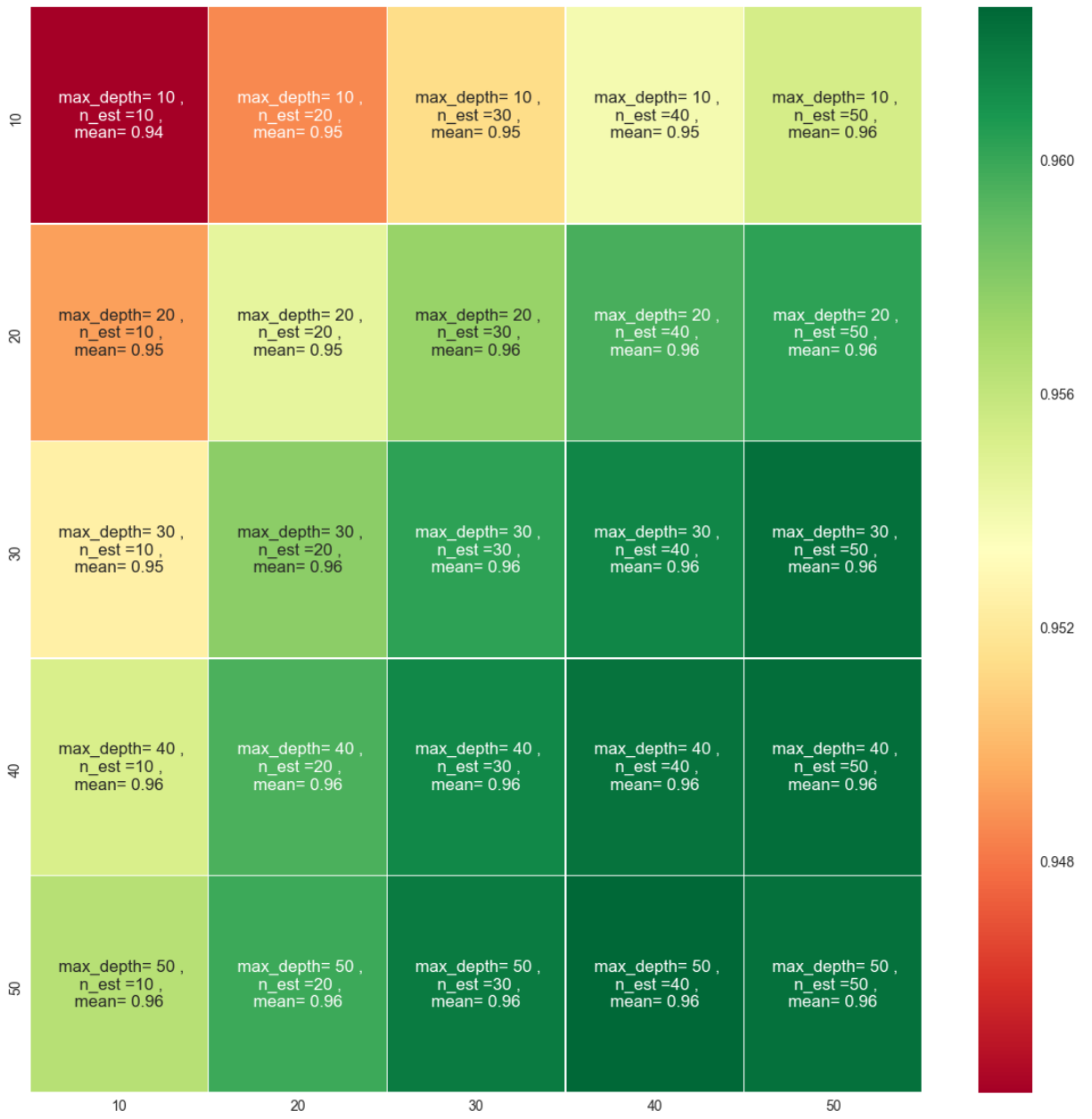
In [110]:
```
from sklearn import preprocessing
from sklearn import ensemble


hyperparameter = dict(n_estimators=[10,20,30,40,50],max_depth=[10,20,30
,40,50])

#Using GridSearchCV
model1 = GridSearchCV(ensemble.GradientBoostingClassifier(), hyperparam
eter, scoring = 'f1', cv=5)
model1.fit(X_tr_tf_idf, y_tr)
a=model1.grid_scores_
print(model1.best_estimator_)
print(model1.score(X_test_tf_idf, y_test))

model1.fit(X_cv_tf_idf, y_cv)
b=model1.grid_scores_
```

```
GradientBoostingClassifier(criterion='friedman_mse', init=None,
              learning_rate=0.1, loss='deviance', max_depth=50,
              max_features=None, max_leaf_nodes=None,
              min_impurity_decrease=0.0, min_impurity_split=None,
```

```
                  min_samples_leaf=1, min_samples_split=2,
                  min_weight_fraction_leaf=0.0, n_estimators=40,
                  presort='auto', random_state=None, subsample=1.0, verbose
=0,
                  warm_start=False)
0.966356608980099
```

In [111]:
```
max_depth=[]
n_estimators = []
mean=[]
for a1 in a:
    max_depth.append(a1[0]['max_depth'])
    n_estimators.append(a1[0]['n_estimators'])
    mean.append(a1[1])
max_depth=np.asarray(max_depth)
n_estimators=np.asarray(n_estimators)
mean=np.asarray(mean)
max_depth = max_depth.reshape(5,5)
n_estimators = n_estimators.reshape(5,5)
mean = mean.reshape(5,5)
result = pd.DataFrame(mean,index=[10,20,30,40,50],columns = [10,20,30,4
0,50])
label =np.asarray([" max_depth= {0} ,\n n_est ={1} ,\n mean= {2:.2f} ".
format(max_depth,n_estimators,mean) for max_depth,n_estimators,mean in
zip(max_depth.flatten(),n_estimators.flatten(),mean.flatten())]).reshap
e(5,5)
import seaborn as sns
fig , ax = plt.subplots(figsize = (20,20))
ax.set_xticks([])
ax.set_yticks([])
sns.heatmap(result,annot=label,fmt="" ,cmap ='RdYlGn',linewidths = 0.30
 , ax = ax )
```

Out[111]: <matplotlib.axes._subplots.AxesSubplot at 0x1e6a9a06a90>



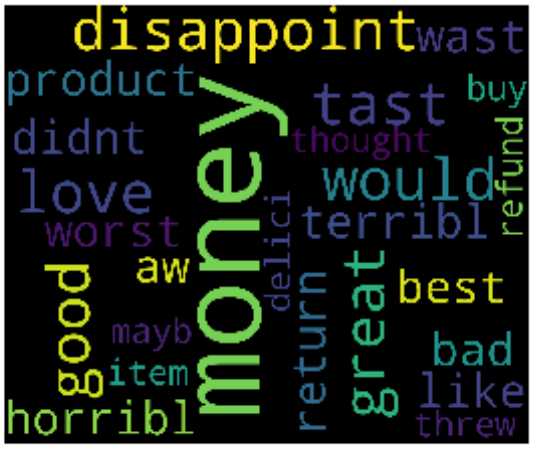
In [166]:
```
clf = ensemble.GradientBoostingClassifier(criterion='friedman_mse', ini
```

```
t=None,
              learning_rate=0.1, loss='deviance', max_depth=50,
              max_features=None, max_leaf_nodes=None,
              min_impurity_decrease=0.0, min_impurity_split=None,
              min_samples_leaf=1, min_samples_split=2,
              min_weight_fraction_leaf=0.0, n_estimators=40,
              presort='auto', random_state=None, subsample=1.0, verbose
=0,
              warm_start=False)
clf=clf.fit(X_tr_tf_idf, y_tr)
print("train-error = ",1-clf.score(X_tr_tf_idf, y_tr))
print("test-error = ",1-clf.score(X_test_tf_idf, y_test))
```

```
train-error =  0.0037346938775509875
test-error =  0.06253333333333333
```

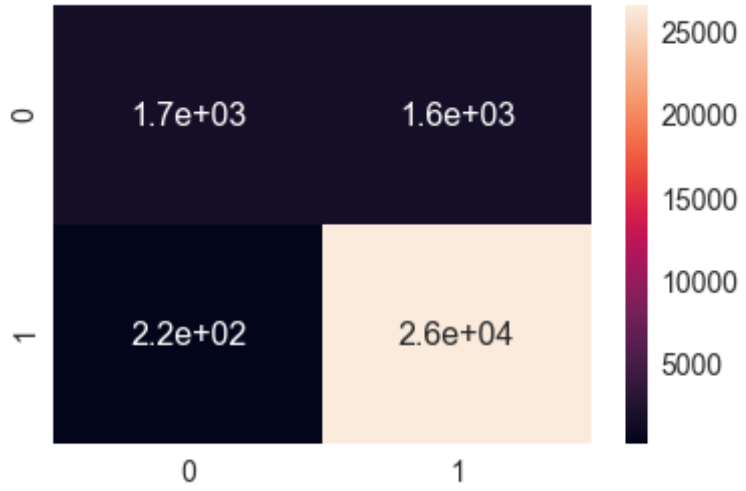
In [167]:

```
importances=clf.feature_importances_
feat_names=tf_idf_vect.get_feature_names()
# Sort feature importances in descending order
indices = np.argsort(importances)[::-1][:25]
a=np.take(feat_names,indices)
def againcleaning(X):

    comment_words=' '
    for words in X:

        comment_words = comment_words + words + ' '
    return comment_words
a=againcleaning(a)
from wordcloud import WordCloud, STOPWORDS
import matplotlib.pyplot as plt
word_cloud=WordCloud(background_color='black',stopwords=stop,
                          width=1200,
                          height=1000).generate(a)
plt.imshow(word_cloud)
plt.axis("off")
plt.show()
```



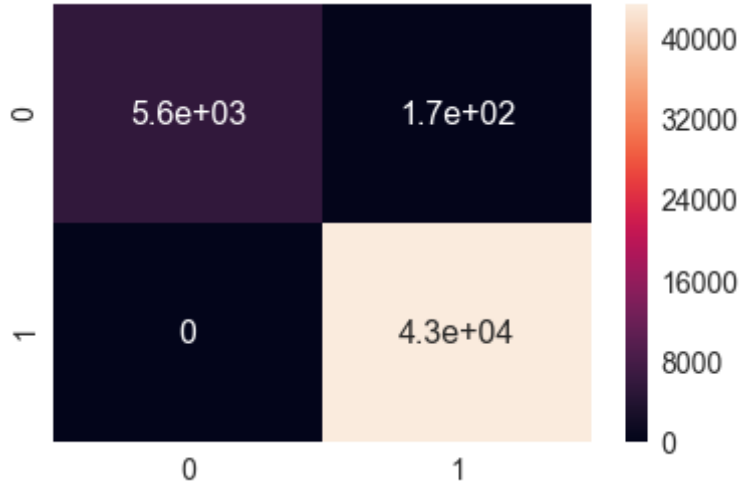
In [114]:
```
pred = clf.predict(X_test_tf_idf)
pred1 = clf.predict(X_tr_tf_idf)
from sklearn.metrics import confusion_matrix
import seaborn as sn
CFM = confusion_matrix(y_test, pred)
CFMtr = confusion_matrix(y_tr, pred1)
df_cm = pd.DataFrame(CFM, range(2), range(2))
#plt.figure(figsize = (10,7))
sn.set(font_scale=1.4)#for label size
sn.heatmap(df_cm, annot=True,annot_kws={"size": 16})
```

Out[114]:
```
<matplotlib.axes._subplots.AxesSubplot at 0x1e6a9ad7da0>
```



In [115]:
```
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[115]: <matplotlib.axes._subplots.AxesSubplot at 0x1e6a89f5cf8>



In [116]:
```
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"))
```

```
0.9385
0.8080641751838463
0.914968725943602
0.7521595506448302
```

RANDOM FOREST AND GBDT FOR WORD 2 VEC

In [117]:
```
from gensim.models import Word2Vec
from gensim.models import KeyedVectors
import pickle
i=0
list_of_sent=[]
for sent in X_tr['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 [118]:
```
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%|██████████| 49000/49000 [01:12<00:00, 676.72it/s]
```

In [119]:

```
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%|██████████| 30000/30000 [00:45<00:00, 659.14it/s]
```

In [120]:

```
list_of_sent2 = []
for sent in X_cv['CleanedText'].values:
    list_of_sent2.append(sent.split())
sent_vectorscv = []; # the avg-w2v for each sentence/review is stored i
n this list
for sent in tqdm(list_of_sent2): # 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_vectorscv.append(sent_vec)
```

```
100%|████████████████████████████████████████████████████████████
████████| 21000/21000 [00:31<00:00, 673.66it/s]
```

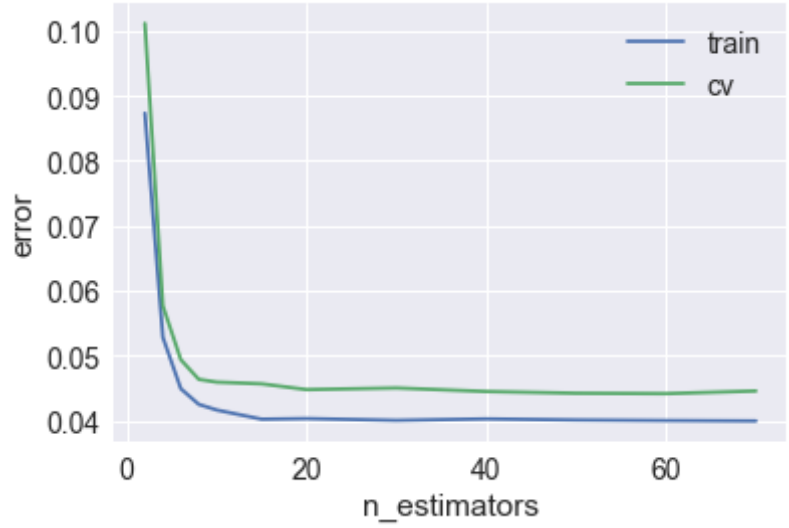
In [121]:

```
from sklearn import preprocessing
from sklearn import ensemble


hyperparameter = dict(n_estimators=[2,4,6,8,10,15,20,30,40,50,60,70])

#Using GridSearchCV
model = GridSearchCV(ensemble.RandomForestClassifier(class_weight='bala
nced'), hyperparameter, scoring = 'f1', cv=5)
model.fit(sent_vectorstr, y_tr)
a=model.grid_scores_
print(model.best_estimator_)
print(model.score(sent_vectorstest, y_test))

model.fit(sent_vectorscv, y_cv)
b=model.grid_scores_
```

```
RandomForestClassifier(bootstrap=True, class_weight='balanced',
            criterion='gini', max_depth=None, max_features='auto',
            max_leaf_nodes=None, min_impurity_decrease=0.0,
            min_impurity_split=None, min_samples_leaf=1,
            min_samples_split=2, min_weight_fraction_leaf=0.0,
            n_estimators=70, n_jobs=1, oob_score=False, random_state=No
ne,
```

```
                       verbose=0, warm_start=False)
0.9638878287261691
```

In [122]:
```
scores = [x[1] for x in a]
cv_scores = [x[1] for x in b]
scores = np.array(scores).reshape(len(hyperparameter['n_estimators']),1
)
cv_scores = np.array(cv_scores).reshape(len(hyperparameter['n_estimator
s']),1)
#for i in enumerate(hyperparameter['max_depth']):
plt.plot(hyperparameter['n_estimators'],1- scores,label='train')
plt.plot(hyperparameter['n_estimators'],1- cv_scores,label='cv')
#plt.legend()
plt.xlabel('n_estimators')
plt.ylabel('error')
plt.legend()
plt.show()
```

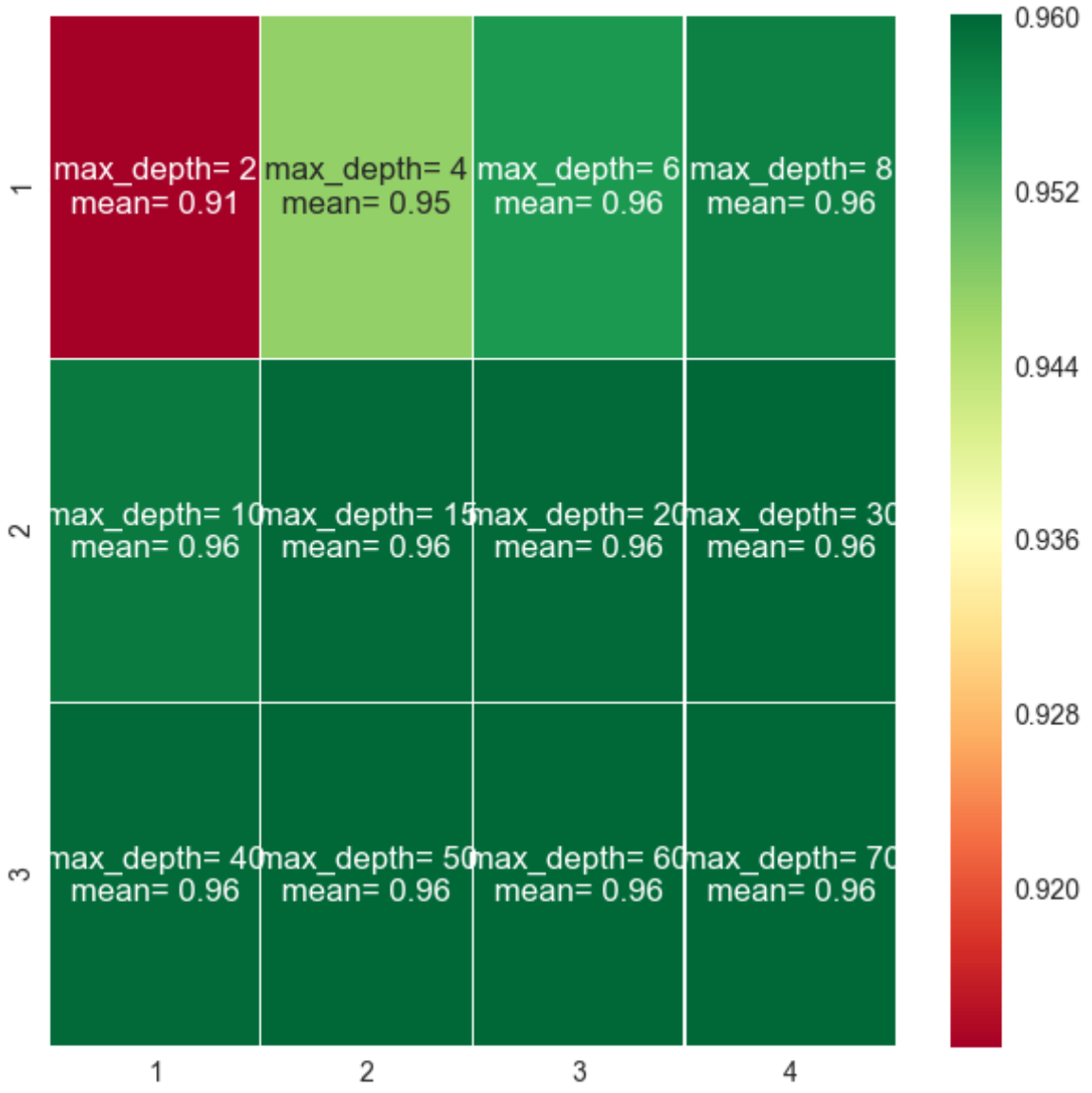


In [123]:
```
n_estimators=[]
mean=[]
for a in a:
    n_estimators.append(a[0]['n_estimators'])
```

```
    mean.append(a[1])
n_estimators=np.asarray(n_estimators)
mean=np.asarray(mean)
n_estimators = n_estimators.reshape(3,4)

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

label =np.asarray([" max_depth= {0} \n mean= {1:.2f} ".format(n_estimat
ors,mean) for n_estimators,mean in zip(n_estimators.flatten(),mean.flat
ten())]).reshape(3,4)
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 )
```

```
[[' max_depth= 2 \n mean= 0.91 ' ' max_depth= 4 \n mean= 0.95 '
  ' max_depth= 6 \n mean= 0.96 ' ' max_depth= 8 \n mean= 0.96 ']
 [' max_depth= 10 \n mean= 0.96 ' ' max_depth= 15 \n mean= 0.96 '
  ' max_depth= 20 \n mean= 0.96 ' ' max_depth= 30 \n mean= 0.96 ']
 [' max_depth= 40 \n mean= 0.96 ' ' max_depth= 50 \n mean= 0.96 '
  ' max_depth= 60 \n mean= 0.96 ' ' max_depth= 70 \n mean= 0.96 ']]
```

Out[123]: <matplotlib.axes._subplots.AxesSubplot at 0x1e6ac168898>



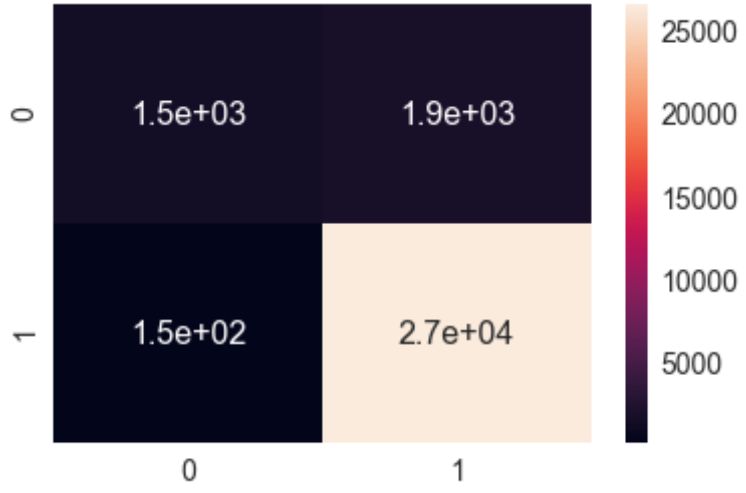
In [177]:
```
clf = ensemble.RandomForestClassifier(bootstrap=True, class_weight='bal
anced',
            criterion='gini', max_depth=None, max_features='auto',
            max_leaf_nodes=None, min_impurity_decrease=0.0,
```

```
            min_impurity_split=None, min_samples_leaf=1,
            min_samples_split=2, min_weight_fraction_leaf=0.0,
            n_estimators=70, n_jobs=1, oob_score=False, random_state=No
ne,
            verbose=0, warm_start=False)
clf=clf.fit(sent_vectorstr, y_tr)
print("train-error = ",format(1-clf.score(sent_vectorstr, y_tr),'.5f'))
print("test-error = ",1-clf.score(sent_vectorstest, y_test))
```

```
train-error =  0.00004
test-error =  0.06643333333333334
```

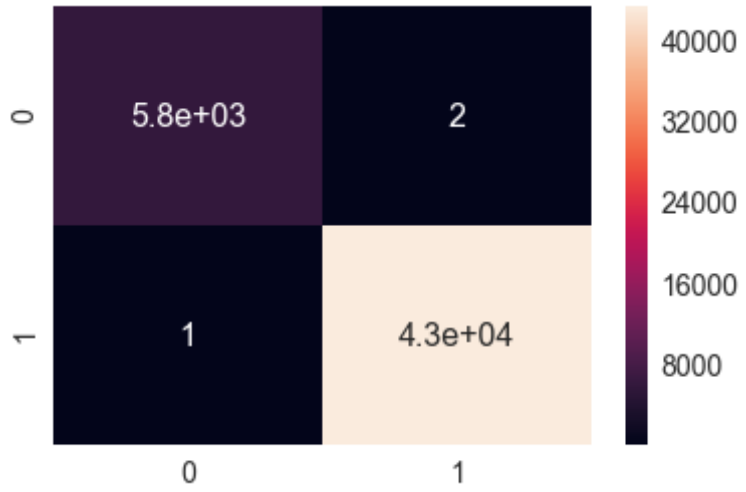
In [125]:
```
pred = clf.predict(sent_vectorstest)
pred1 = clf.predict(sent_vectorstr)
from sklearn.metrics import confusion_matrix
import seaborn as sn
CFM = confusion_matrix(y_test, pred)
CFMtr = confusion_matrix(y_tr, pred1)
df_cm = pd.DataFrame(CFM, range(2), range(2))
#plt.figure(figsize = (10,7))
sn.set(font_scale=1.4)#for label size
sn.heatmap(df_cm, annot=True,annot_kws={"size": 16})
```

Out[125]:
```
<matplotlib.axes._subplots.AxesSubplot at 0x1e6c6911ac8>
```



In [126]:
```
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[126]: <matplotlib.axes._subplots.AxesSubplot at 0x1e6c69a16d8>



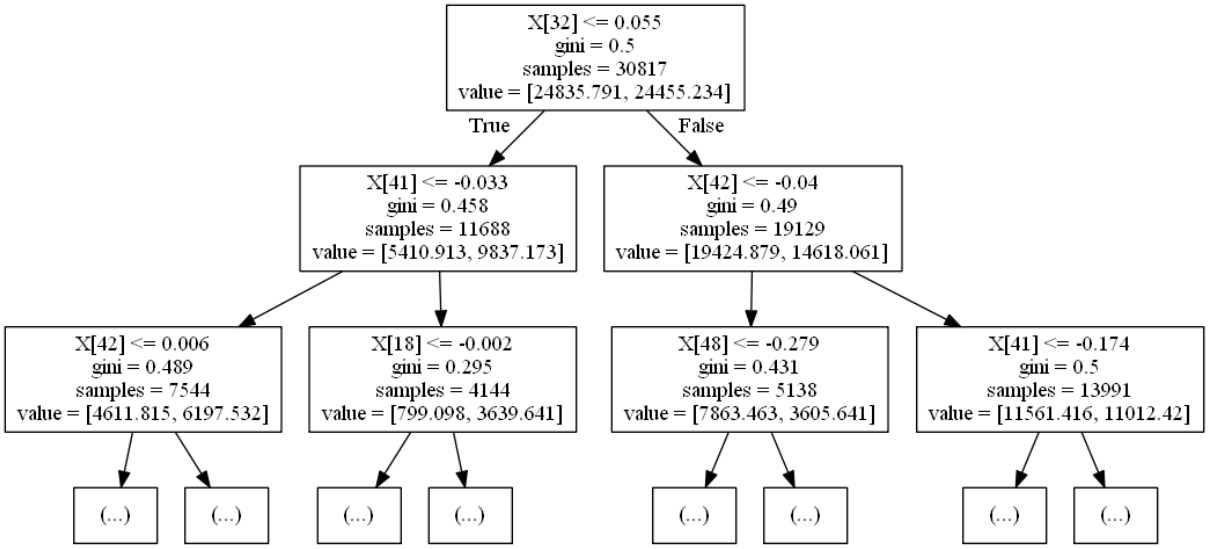
In [127]:
```
from sklearn.metrics import accuracy_score, f1_score, precision_score,
recall_score
print(accuracy_score(y_test, pred))
print(f1_score(y_test, pred, average="macro"))
print(precision_score(y_test, pred, average="macro"))
print(recall_score(y_test, pred, average="macro"))
```

```
0.9329666666666667
0.779696488805706
0.9204868235812538
0.7190699640573484
```

In [128]:
```
from sklearn import tree
import os
import graphviz
from sklearn.externals.six import StringIO
```

```
os.environ["PATH"] += os.pathsep + r'C:\Users\krush\Anaconda3\Lib\site-packages\graphviz'
#os.path.abspath('C:\\\Users\\\krush\\\Anaconda3\\\Lib\\\site-packages\\\graphviz')
dot_data1 = StringIO()
dot_data = tree.export_graphviz(clf.estimators_[0], out_file=dot_data1,
                                max_depth = 2)
#graph = graphviz.Source(dot_data)

from IPython.display import Image
import pydot
graph = pydot.graph_from_dot_data(dot_data1.getvalue())[0]
Image(graph.create_png())
```

Out[128]:



In [129]:

```
from sklearn import preprocessing
from sklearn import ensemble


hyperparameter = dict(n_estimators=[10,20,30,40,50],max_depth=[10,20,30,40,50])

#Using GridSearchCV
```

```
model1 = GridSearchCV(ensemble.GradientBoostingClassifier(), hyperparam
eter, scoring = 'f1', cv=5)
model1.fit(sent_vectorstr, y_tr)
a=model1.grid_scores_
print(model1.best_estimator_)
print(model1.score(sent_vectorstest, y_test))
```

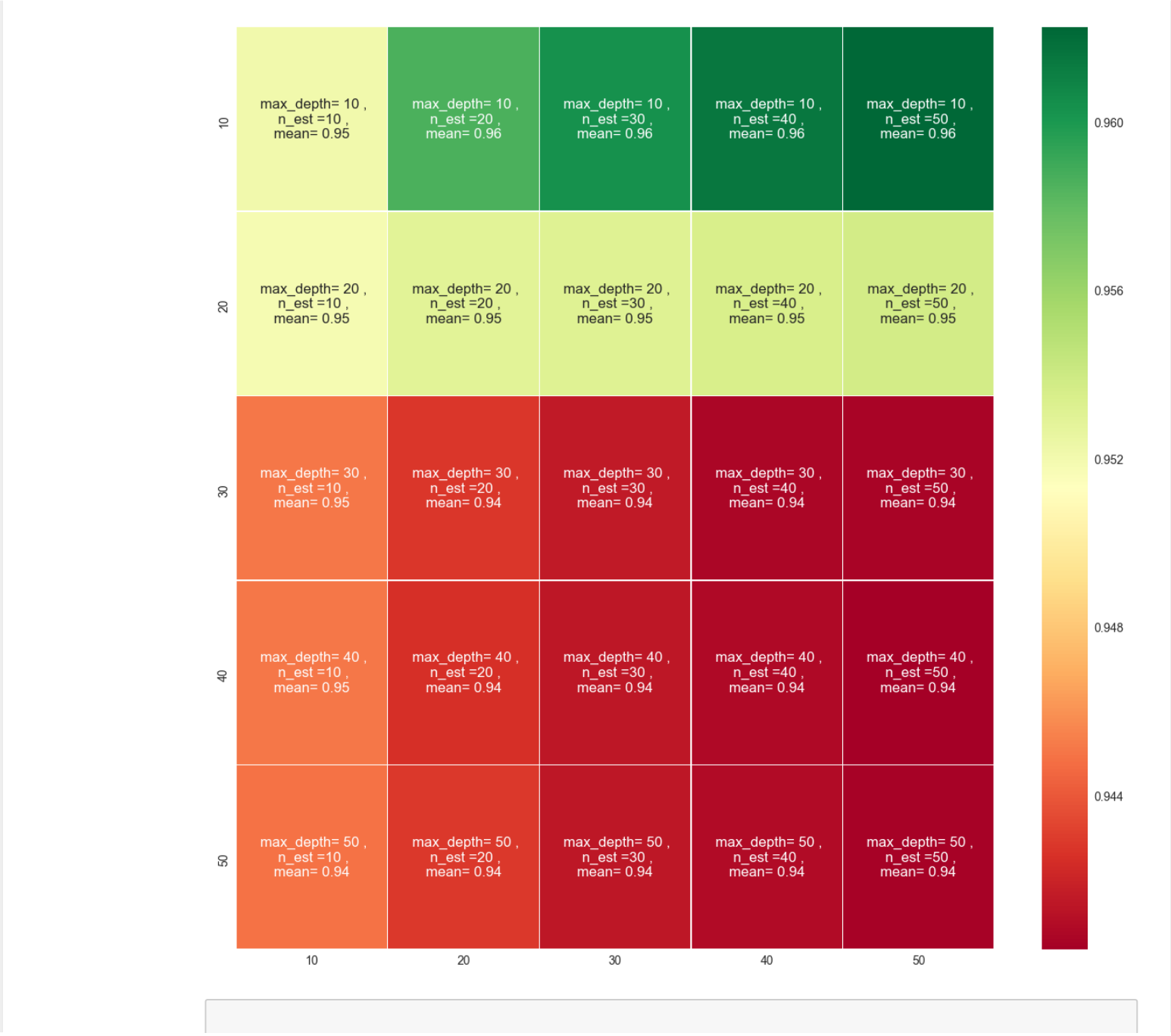
```
GradientBoostingClassifier(criterion='friedman_mse', init=None,
              learning_rate=0.1, loss='deviance', max_depth=10,
              max_features=None, max_leaf_nodes=None,
              min_impurity_decrease=0.0, min_impurity_split=None,
              min_samples_leaf=1, min_samples_split=2,
              min_weight_fraction_leaf=0.0, n_estimators=50,
              presort='auto', random_state=None, subsample=1.0, verbose
=0,
              warm_start=False)
0.9641313109108229
```

In [130]:

```
max_depth=[]
n_estimators = []
mean=[]
for a1 in a:
    max_depth.append(a1[0]['max_depth'])
    n_estimators.append(a1[0]['n_estimators'])
    mean.append(a1[1])
max_depth=np.asarray(max_depth)
n_estimators=np.asarray(n_estimators)
mean=np.asarray(mean)
max_depth = max_depth.reshape(5,5)
n_estimators = n_estimators.reshape(5,5)
mean = mean.reshape(5,5)
result = pd.DataFrame(mean,index=[10,20,30,40,50],columns = [10,20,30,4
0,50])
label =np.asarray([" max_depth= {0} ,\n n_est ={1} ,\n mean= {2:.2f} ".
format(max_depth,n_estimators,mean) for max_depth,n_estimators,mean in
zip(max_depth.flatten(),n_estimators.flatten(),mean.flatten())]).reshap
e(5,5)
import seaborn as sns
fig , ax = plt.subplots(figsize = (20,20))
```

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

Out[130]: <matplotlib.axes._subplots.AxesSubplot at 0x1e6c6c98198>

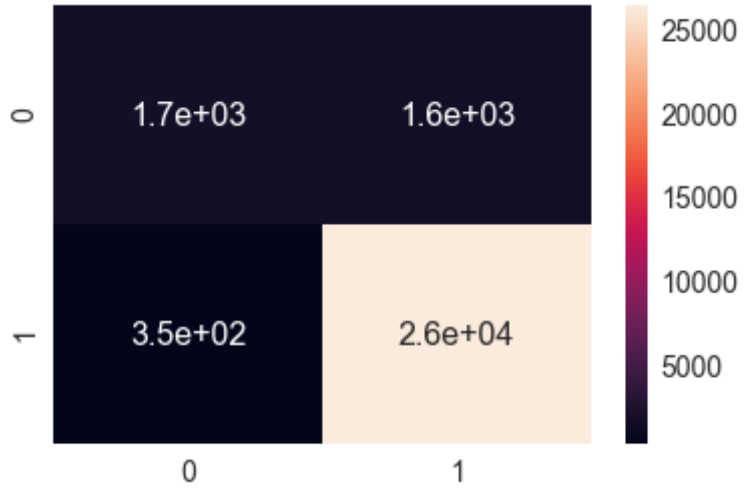


In [134]:
```
clf = ensemble.GradientBoostingClassifier(criterion='friedman_mse', init=None,
                learning_rate=0.1, loss='deviance', max_depth=10,
                max_features=None, max_leaf_nodes=None,
                min_impurity_decrease=0.0, min_impurity_split=None,
                min_samples_leaf=1, min_samples_split=2,
                min_weight_fraction_leaf=0.0, n_estimators=50,
                presort='auto', random_state=None, subsample=1.0, verbose=0,
                warm_start=False)
clf=clf.fit(sent_vectorstr, y_tr)
print("train-error = ",1-clf.score(sent_vectorstr, y_tr))
print("test-error = ",1-clf.score(sent_vectorstest, y_test))
```

```
train-error =  0.007877551020408213
test-error =  0.06523333333333337
```

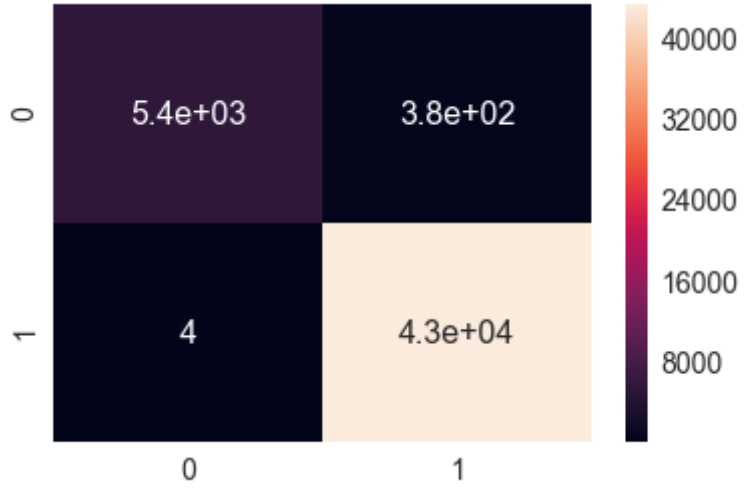
In [135]:
```
pred = clf.predict(sent_vectorstest)
pred1 = clf.predict(sent_vectorstr)
from sklearn.metrics import confusion_matrix
import seaborn as sn
CFM = confusion_matrix(y_test, pred)
CFMtr = confusion_matrix(y_tr, pred1)
df_cm = pd.DataFrame(CFM, range(2), range(2))
#plt.figure(figsize = (10,7))
sn.set(font_scale=1.4)#for label size
sn.heatmap(df_cm, annot=True,annot_kws={"size": 16})
```

Out[135]:
```
<matplotlib.axes._subplots.AxesSubplot at 0x1e6c6f0b978>
```



In [136]:
```
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[136]: <matplotlib.axes._subplots.AxesSubplot at 0x1e6c6f61748>



In [137]:
```
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"))
```

```
0.9347666666666666
0.801636018676974
0.8873877347266274
0.7528078551107837
```

RANDOM FOREST AND GBDT FOR TFIDF W2V

In [138]:

```
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%|██████████| 49000/49000 [01:29<00:00, 548.33it/s]
```

In [139]:

```
# 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%|██████████| 30000/30000 [00:55<00:00, 536.47it/s]
```

In [140]:

```
# 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_cv = []; # the tfidf-w2v for each sentence/review is
 stored in this list
row=0;
for sent in tqdm(list_of_sent2): # 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_cv.append(sent_vec)
    row += 1
```

```
100%|████████████████████████████████████████████████████████████████
████████| 21000/21000 [00:38<00:00, 545.09it/s]
```

In [141]:

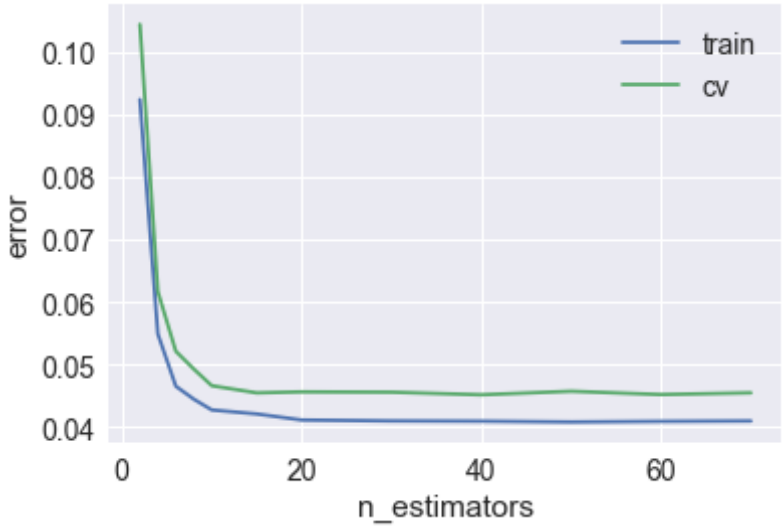
```
from sklearn import preprocessing
from sklearn import ensemble


hyperparameter = dict(n_estimators=[2,4,6,8,10,15,20,30,40,50,60,70])

#Using GridSearchCV
model = GridSearchCV(ensemble.RandomForestClassifier(class_weight='bala
nced'), hyperparameter, scoring = 'f1', cv=5)
model.fit(tfidf_sent_vectors, y_tr)
a=model.grid_scores_
print(model.best_estimator_)
print(model.score(tfidf_sent_vectors_test, y_test))
```

```
model.fit(tfidf_sent_vectors_cv, y_cv)
b=model.grid_scores_
```

```
RandomForestClassifier(bootstrap=True, class_weight='balanced',
            criterion='gini', max_depth=None, max_features='auto',
            max_leaf_nodes=None, min_impurity_decrease=0.0,
            min_impurity_split=None, min_samples_leaf=1,
            min_samples_split=2, min_weight_fraction_leaf=0.0,
            n_estimators=50, n_jobs=1, oob_score=False, random_state=No
ne,
            verbose=0, warm_start=False)
0.9628984876509629
```

In [142]:

```
scores = [x[1] for x in a]
cv_scores = [x[1] for x in b]
scores = np.array(scores).reshape(len(hyperparameter['n_estimators']),1
)
cv_scores = np.array(cv_scores).reshape(len(hyperparameter['n_estimator
s']),1)
#for i in enumerate(hyperparameter['max_depth']):
plt.plot(hyperparameter['n_estimators'],1- scores,label='train')
plt.plot(hyperparameter['n_estimators'],1- cv_scores,label='cv')
#plt.legend()
plt.xlabel('n_estimators')
plt.ylabel('error')
plt.legend()
plt.show()
```

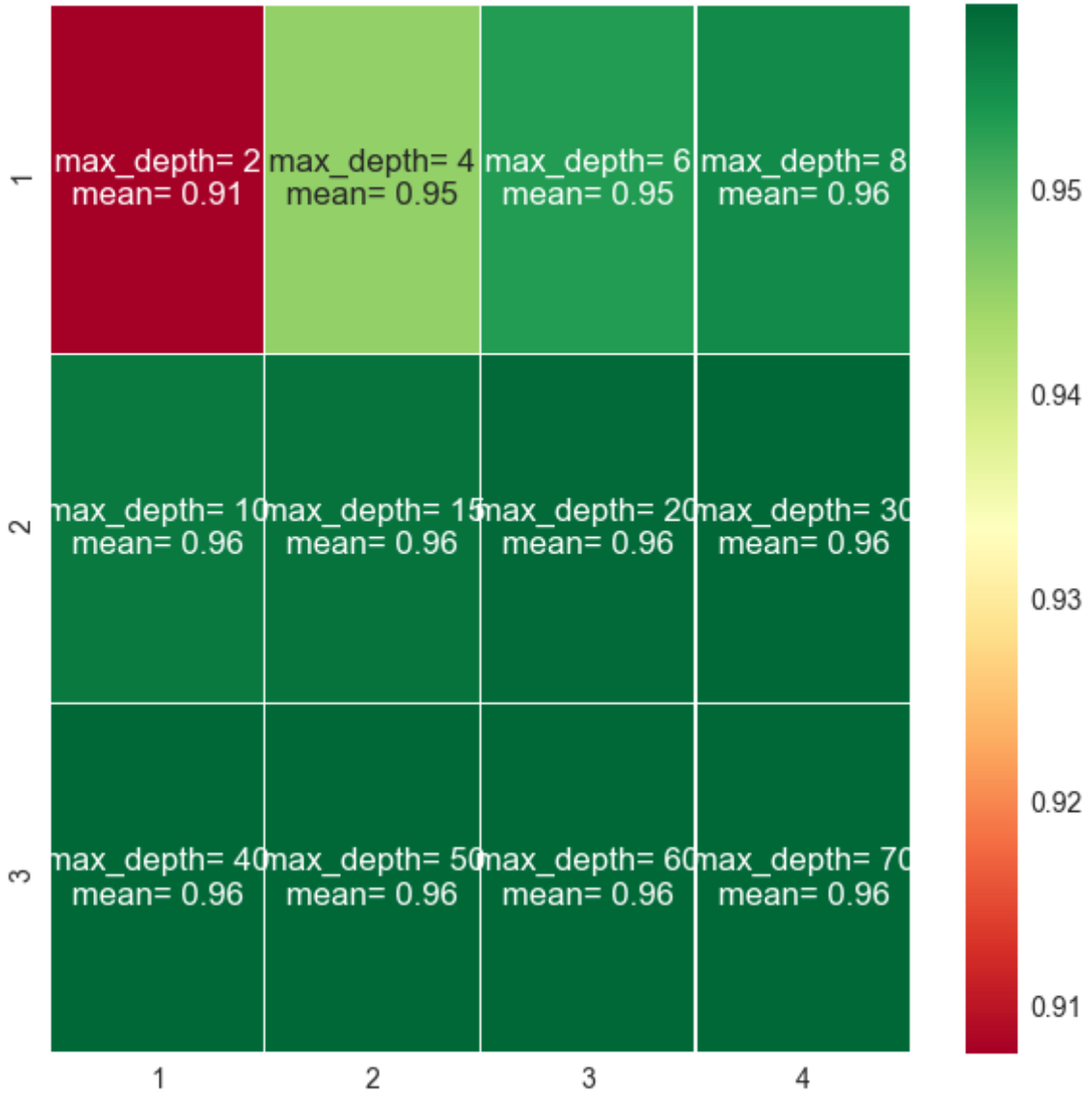


In [143]:
```
n_estimators=[]
mean=[]
for a in a:
    n_estimators.append(a[0]['n_estimators'])
    mean.append(a[1])
n_estimators=np.asarray(n_estimators)
mean=np.asarray(mean)
n_estimators = n_estimators.reshape(3,4)

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

label =np.asarray([" max_depth= {0} \n mean= {1:.2f} ".format(n_estimat
ors,mean) for n_estimators,mean in zip(n_estimators.flatten(),mean.flat
ten())]).reshape(3,4)
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 )
```

```
[[' max_depth= 2 \n mean= 0.91 ' ' max_depth= 4 \n mean= 0.95 '
  ' max_depth= 6 \n mean= 0.95 ' ' max_depth= 8 \n mean= 0.96 ']
 [' max_depth= 10 \n mean= 0.96 ' ' max_depth= 15 \n mean= 0.96 '
  ' max_depth= 20 \n mean= 0.96 ' ' max_depth= 30 \n mean= 0.96 ']
 [' max_depth= 40 \n mean= 0.96 ' ' max_depth= 50 \n mean= 0.96 '
  ' max_depth= 60 \n mean= 0.96 ' ' max_depth= 70 \n mean= 0.96 ']]
```

Out[143]: <matplotlib.axes._subplots.AxesSubplot at 0x1e6c6fe5860>



In [144]:
```
clf = ensemble.RandomForestClassifier(bootstrap=True, class_weight='balanced',
            criterion='gini', max_depth=None, max_features='auto',
```

```
            max_leaf_nodes=None, min_impurity_decrease=0.0,
            min_impurity_split=None, min_samples_leaf=1,
            min_samples_split=2, min_weight_fraction_leaf=0.0,
            n_estimators=50, n_jobs=1, oob_score=False, random_state=No
ne,
            verbose=0, warm_start=False)
clf=clf.fit(tfidf_sent_vectors, y_tr)
print("train-error = ",1-clf.score(tfidf_sent_vectors, y_tr))
print("test-error = ",1-clf.score(tfidf_sent_vectors_test, y_test))
```

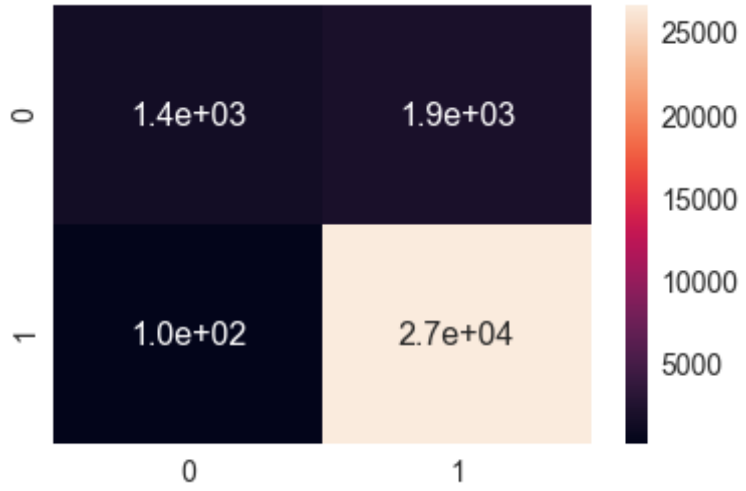
```
train-error =  0.0001224489795917938
test-error =  0.06789999999999996
```

In [145]:

```
pred = clf.predict(tfidf_sent_vectors_test)
pred1 = clf.predict(tfidf_sent_vectors)
from sklearn.metrics import confusion_matrix
import seaborn as sn
CFM = confusion_matrix(y_test, pred)
CFMtr = confusion_matrix(y_tr, 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})
```

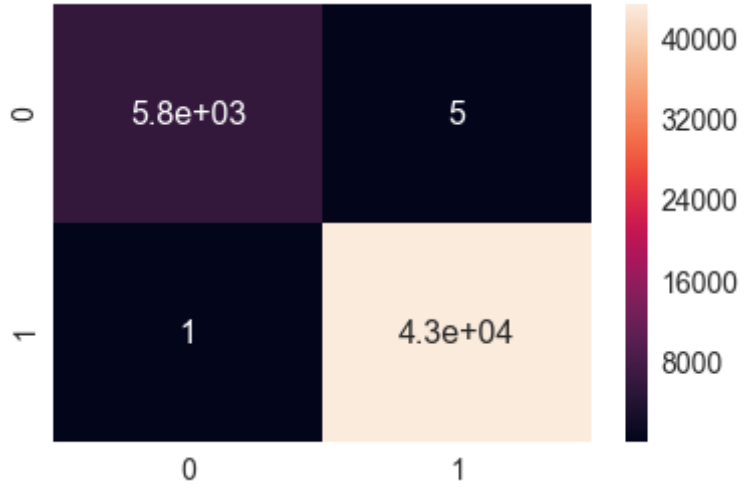
Out[145]:

```
<matplotlib.axes._subplots.AxesSubplot at 0x1e6c6eeac88>
```



In [146]:
```
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[146]: <matplotlib.axes._subplots.AxesSubplot at 0x1e6ce719198>



In [147]:
```
from sklearn.metrics import accuracy_score, f1_score, precision_score,
```

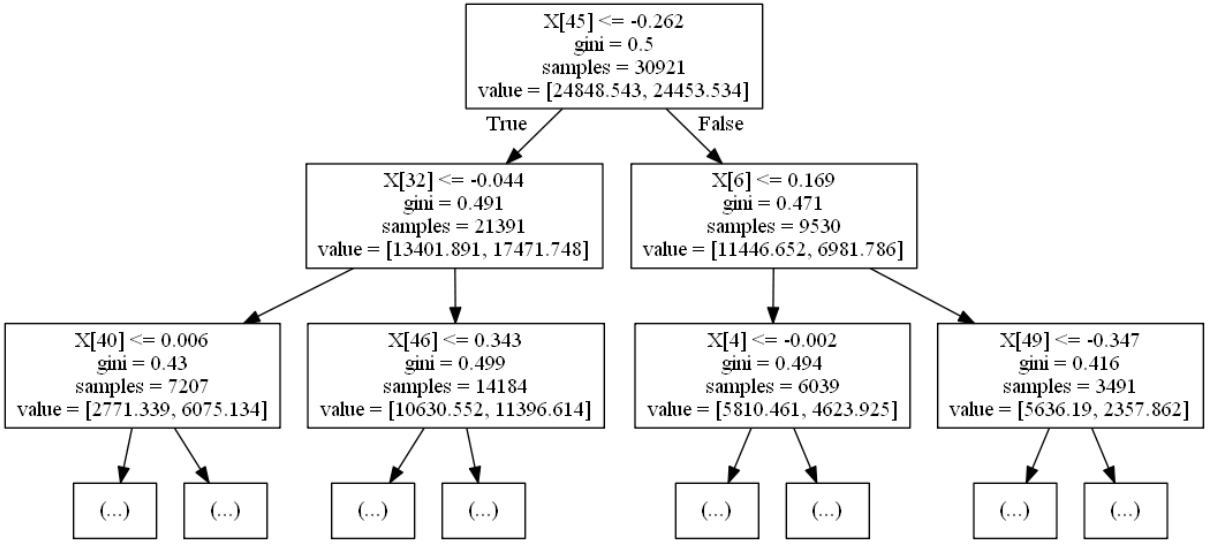
```
recall_score
print(accuracy_score(y_test, pred))
print(f1_score(y_test, pred, average="macro"))
print(precision_score(y_test, pred, average="macro"))
print(recall_score(y_test, pred, average="macro"))
```

```
0.9321
0.771746782357046
0.9314122508996748
0.7088956942975041
```

In [148]:

```
from sklearn import tree
import os
import graphviz
from sklearn.externals.six import StringIO
os.environ["PATH"] += os.pathsep + r'C:\Users\krush\Anaconda3\Lib\site-packages\graphviz'
#os.path.abspath('C:\\\Users\\\krush\\\Anaconda3\\\Lib\\\site-packages\\\graphviz')
dot_data1 = StringIO()
dot_data = tree.export_graphviz(clf.estimators_[0], out_file=dot_data1,
                                max_depth = 2)
#graph = graphviz.Source(dot_data)

from IPython.display import Image
import pydot
graph = pydot.graph_from_dot_data(dot_data1.getvalue())[0]
Image(graph.create_png())
```

Out[148]:



In [149]:

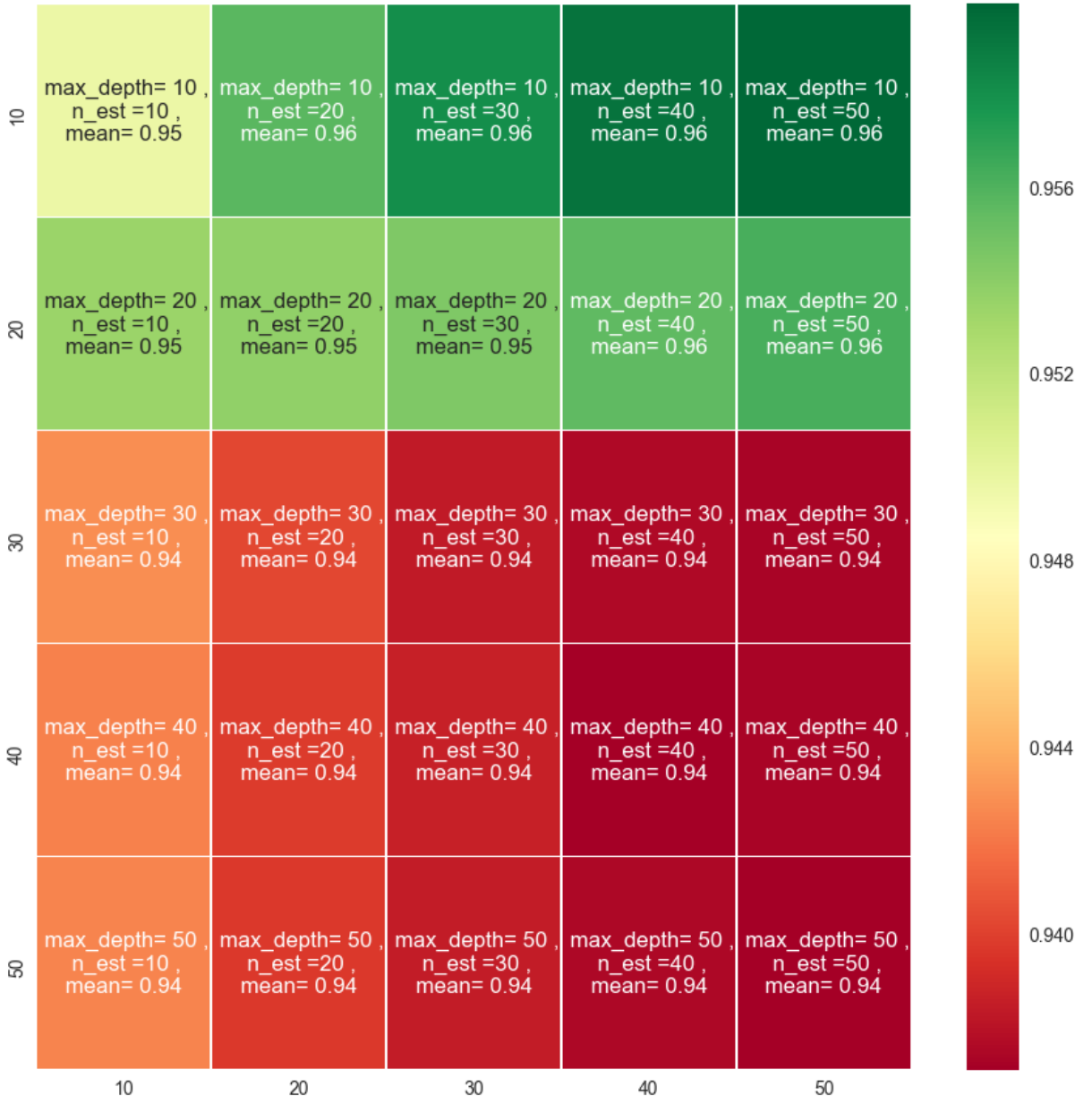
```
from sklearn import preprocessing
from sklearn import ensemble


hyperparameter = dict(n_estimators=[10,20,30,40,50],max_depth=[10,20,30
,40,50])

#Using GridSearchCV
model1 = GridSearchCV(ensemble.GradientBoostingClassifier(), hyperparam
eter, scoring = 'f1', cv=5)



model1.fit(tfidf_sent_vectors, y_tr)
a=model1.grid_scores_
print(model1.best_estimator_)
print(model1.score(tfidf_sent_vectors_test, y_test))
```

```
GradientBoostingClassifier(criterion='friedman_mse', init=None,
              learning_rate=0.1, loss='deviance', max_depth=10,
              max_features=None, max_leaf_nodes=None,
              min_impurity_decrease=0.0, min_impurity_split=None,
```

```
                   min_samples_leaf=1, min_samples_split=2,
                   min_weight_fraction_leaf=0.0, n_estimators=50,
                   presort='auto', random_state=None, subsample=1.0, verbose
=0,
                   warm_start=False)
0.9628453593196831
```

In [151]:
```
max_depth=[]
n_estimators = []
mean=[]
for a1 in a:
    max_depth.append(a1[0]['max_depth'])
    n_estimators.append(a1[0]['n_estimators'])
    mean.append(a1[1])
max_depth=np.asarray(max_depth)
n_estimators=np.asarray(n_estimators)
mean=np.asarray(mean)
max_depth = max_depth.reshape(5,5)
n_estimators = n_estimators.reshape(5,5)
mean = mean.reshape(5,5)
result = pd.DataFrame(mean,index=[10,20,30,40,50],columns = [10,20,30,4
0,50])
label =np.asarray([" max_depth= {0} ,\n n_est ={1} ,\n mean= {2:.2f} ".
format(max_depth,n_estimators,mean) for max_depth,n_estimators,mean in
zip(max_depth.flatten(),n_estimators.flatten(),mean.flatten())]).reshap
e(5,5)
import seaborn as sns
fig , ax = plt.subplots(figsize = (15,15))
ax.set_xticks([])
ax.set_yticks([])
sns.heatmap(result,annot=label,fmt="" ,cmap ='RdYlGn',linewidths = 0.30
 , ax = ax )
```

Out[151]: <matplotlib.axes._subplots.AxesSubplot at 0x1e6ca49da90>



In [152]:
```
clf = ensemble.GradientBoostingClassifier(criterion='friedman_mse', init=None,
```

```
                learning_rate=0.1, loss='deviance', max_depth=10,
                max_features=None, max_leaf_nodes=None,
                min_impurity_decrease=0.0, min_impurity_split=None,
                min_samples_leaf=1, min_samples_split=2,
                min_weight_fraction_leaf=0.0, n_estimators=50,
                presort='auto', random_state=None, subsample=1.0, verbose
=0,
                warm_start=False)
clf=clf.fit(tfidf_sent_vectors, y_tr)
print("train-error = ",1-clf.score(tfidf_sent_vectors, y_tr))
print("test-error = ",1-clf.score(tfidf_sent_vectors_test, y_test))
```

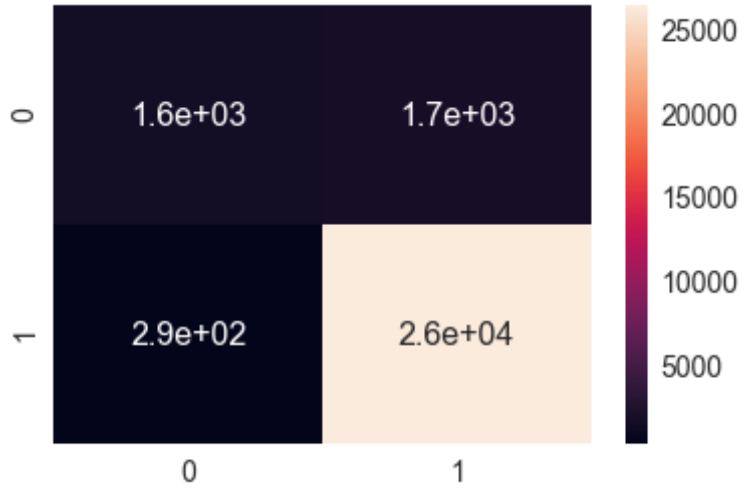
```
train-error =  0.012204081632653074
test-error =  0.06769999999999998
```

In [153]:

```
pred = clf.predict(tfidf_sent_vectors_test)
pred1 = clf.predict(tfidf_sent_vectors)
from sklearn.metrics import confusion_matrix
import seaborn as sn
CFM = confusion_matrix(y_test, pred)
CFMtr = confusion_matrix(y_tr, 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})
```

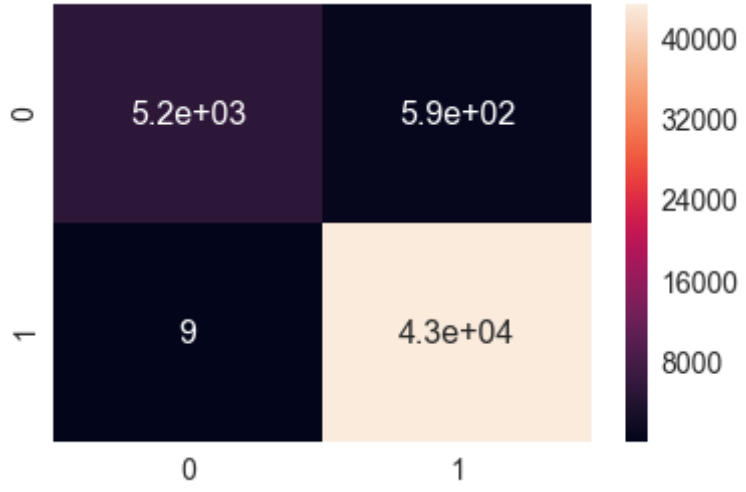
Out[153]:

```
<matplotlib.axes._subplots.AxesSubplot at 0x1e6c89b42b0>
```



In [154]:
```
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[154]: <matplotlib.axes._subplots.AxesSubplot at 0x1e6c6c297b8>



In [155]:
```
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"))
```

```
0.9323
0.7872537733465808
0.8924807730337729
0.7338793025562943
```

In [178]:

```
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
RF Train error
Please add a BOW to the list
.0002
Please enter a TF-IDF for the service
```

```
.0008
Please enter a W2V for the service
.00004
Please enter a TFIDF W2V for the service
0.0001
Continue yes/ no:y
Please add a head to the list
RF Test Error
Please add a BOW to the list
0.0595
Please enter a TF-IDF for the service
0.0598
Please enter a W2V for the service
0.0625
Please enter a TFIDF W2V for the service
0.0678
Continue yes/ no:y
Please add a head to the list
GBDT Train error
Please add a BOW to the list
0.0117
Please enter a TF-IDF for the service
0.003
Please enter a W2V for the service
0.007
Please enter a TFIDF W2V for the service
0.012
Continue yes/ no:y
Please add a head to the list
GBDT Test Error
Please add a BOW to the list
0.0629
Please enter a TF-IDF for the service
0.0625
Please enter a W2V for the service
0.0652
Please enter a TFIDF W2V for the service
0.0676
Continue yes/ no:n
```

In [179]:
```
print(x)
```

```
+------------------+--------+--------+--------+-----------+
|      Table       |  BOW   | TF-IDF |  W2V   | TFIDF W2V |
+------------------+--------+--------+--------+-----------+
|  RF Train error  | 0.0002 | 0.0008 | 4e-05  |   0.0001  |
|  RF Test Error   | 0.0595 | 0.0598 | 0.0625 |   0.0678  |
| GBDT Train error | 0.0117 | 0.003  | 0.007  |   0.012   |
| GBDT Test Error  | 0.0629 | 0.0625 | 0.0652 |   0.0676  |
+------------------+--------+--------+--------+-----------+
```