Decision Tree Classifier

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)
```

Text 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 [5]:
```
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 [6]:
```
sorted_data=filtered_data.sort_values(by=['Time'])
sampledata = sorted_data.head(100000)

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

Splitting data

In [7]:
```
# 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)
```

Feature importance

In [21]:
```
comment_words = ' '
for val in X_tr['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 [22]:

```
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()
```

gridsearch and random search for decison tree, All Metrics for Decision tree classifier with

BOW.

In [8]:
```
count_vect = CountVectorizer(min_df=10) #in scikit-learn
vec = count_vect.fit(X_tr['CleanedText'].values)
```

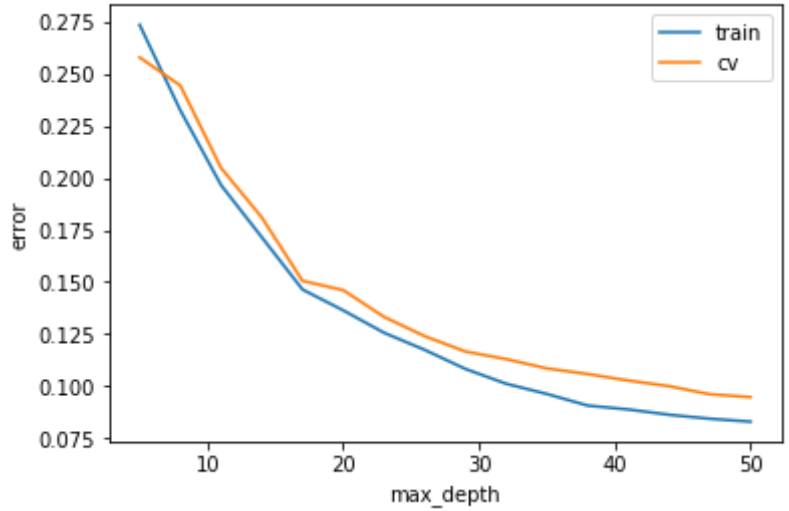
In [9]:
```
X_trvec = vec.transform(X_tr['CleanedText'].values)
X_cvvec = vec.transform(X_cv['CleanedText'].values)
X_testvec = vec.transform(X_test['CleanedText'].values)
```

In [14]:
```
from sklearn import preprocessing
from sklearn import tree


hyperparameter = dict(max_depth=[5,8,11,14,17,20,23,26,29,32,35,38,41,44,47,50])

#Using GridSearchCV
model = GridSearchCV(tree.DecisionTreeClassifier(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_
```

```
DecisionTreeClassifier(class_weight='balanced', criterion='gini',
            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, presort=False, random_state=None,
            splitter='best')
0.9225027356573392
```

In [17]:
```
scores = [x[1] for x in a]
```

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

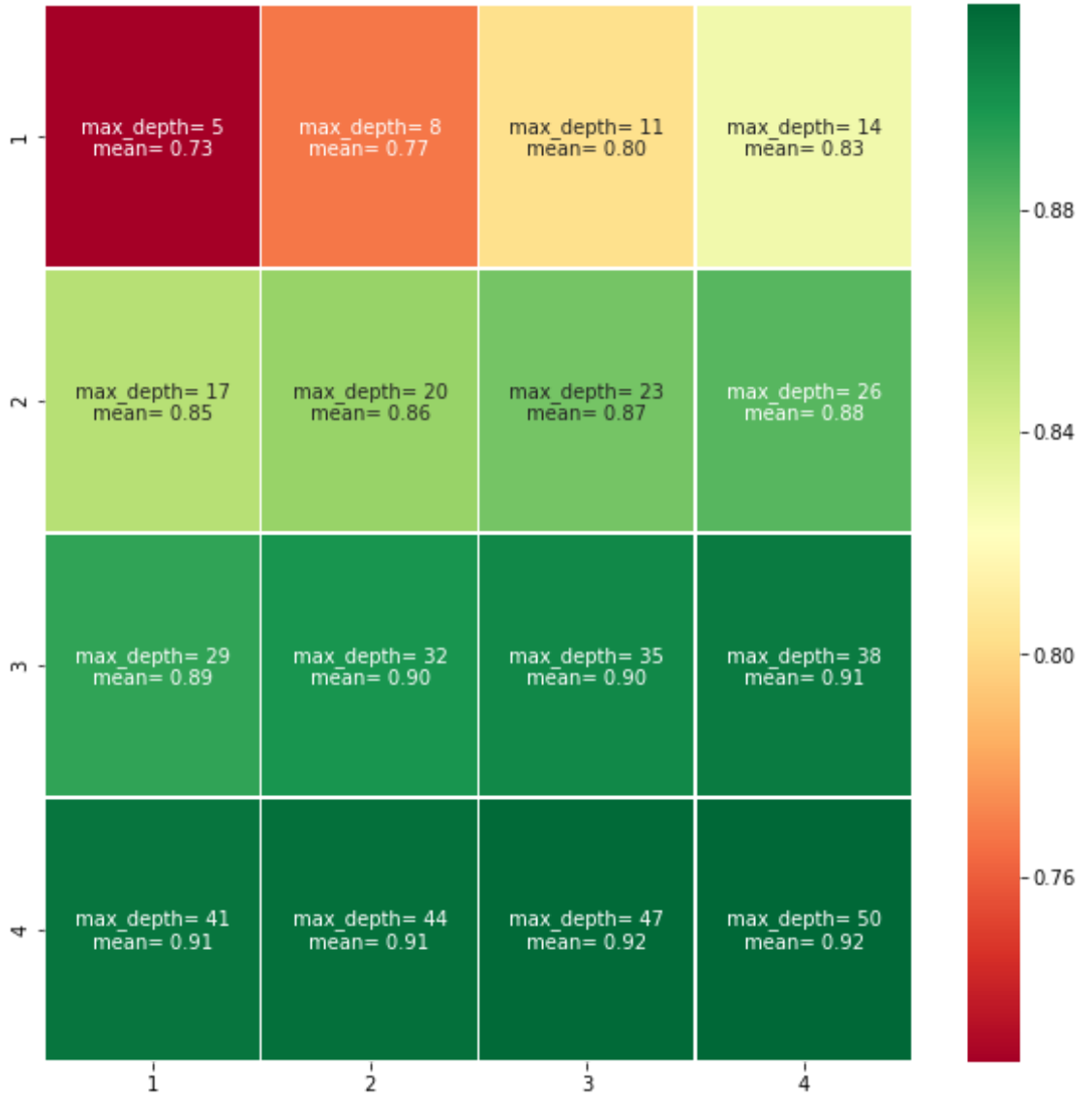


In [18]:
```
max_depth=[]

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

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

label =np.asarray([" max_depth= {0} \n mean= {1:.2f} ".format(max_depth
,mean) for max_depth,mean in zip(max_depth.flatten(),mean.flatten())]).
reshape(4,4)
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= 5 \n mean= 0.73 ' ' max_depth= 8 \n mean= 0.77 '
  ' max_depth= 11 \n mean= 0.80 ' ' max_depth= 14 \n mean= 0.83 ']
 [' max_depth= 17 \n mean= 0.85 ' ' max_depth= 20 \n mean= 0.86 '
  ' max_depth= 23 \n mean= 0.87 ' ' max_depth= 26 \n mean= 0.88 ']
 [' max_depth= 29 \n mean= 0.89 ' ' max_depth= 32 \n mean= 0.90 '
  ' max_depth= 35 \n mean= 0.90 ' ' max_depth= 38 \n mean= 0.91 ']
 [' max_depth= 41 \n mean= 0.91 ' ' max_depth= 44 \n mean= 0.91 '
  ' max_depth= 47 \n mean= 0.92 ' ' max_depth= 50 \n mean= 0.92 ']]
```

Out[18]: <matplotlib.axes._subplots.AxesSubplot at 0x215c0aa0fd0>



In [20]:
```
clf = tree.DecisionTreeClassifier(class_weight='balanced', criterion='g
ini',
            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, presort=False, random_state=N
one,
            splitter='best')
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 =  0.06146938775510202
test-error =  0.1338666666666667
```

In [30]:

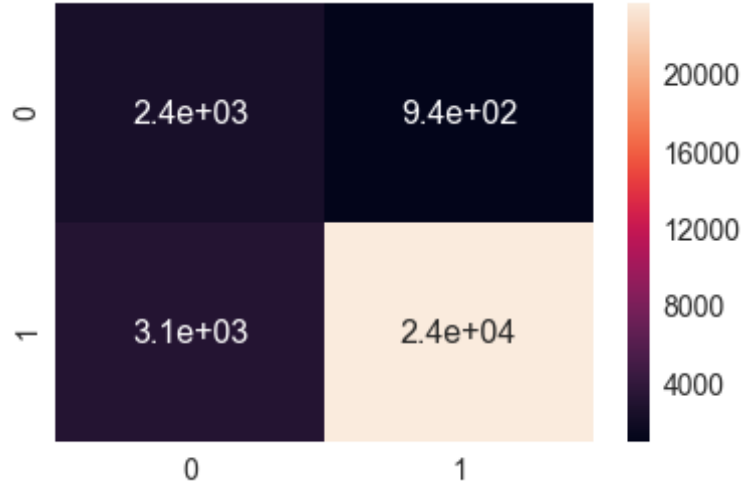
```
def important_features(vectorizer,classifier,n):
    class_labels = classifier.classes_
    print(class_labels)
    feature_names =vectorizer.get_feature_names()

    topn_class1 = sorted(zip(classifier.feature_importances_, feature_n
ames),reverse=True)[0:n]


    print("Important words in  reviews")
    for coef,feat in topn_class1:

        print(feat)



important_features(vec,clf,n=20)
```

```
[0 1]
Important words in  reviews
great
disappoint
best
love
delici
good
perfect
tast
excel
```

```
favorit
howev
bad
aw
nice
would
thought
product
hope
wont
tasti
```

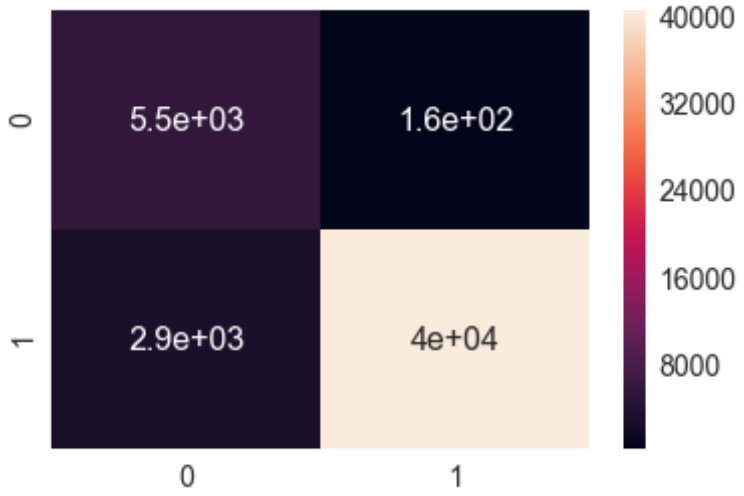
In [22]:
```
pred = clf.predict(X_testvec)
pred1 = clf.predict(X_trvec)
from sklearn.metrics import confusion_matrix
import seaborn as sn
CFM = confusion_matrix(y_test, pred)
CFMtr = confusion_matrix(y_tr, pred1)
df_cm = pd.DataFrame(CFM, range(2), range(2))
#plt.figure(figsize = (10,7))
sn.set(font_scale=1.4)#for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

Out[22]: <matplotlib.axes._subplots.AxesSubplot at 0x215cfda13c8>



In [23]:
```
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[23]: <matplotlib.axes._subplots.AxesSubplot at 0x215be169908>



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"))
```

```
0.8661333333333333
0.7331015250961659
0.700053730330962
0.8017628698900441
```

In [25]:
```
feature_names =vec.get_feature_names()
```

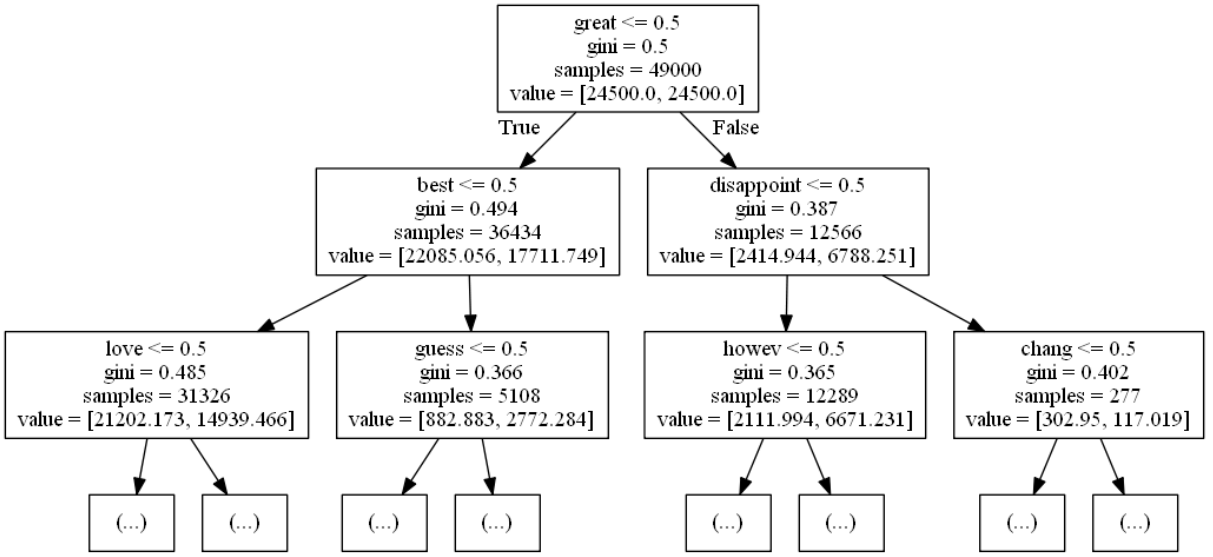
In [26]:
```
import os
```

```
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,feature_names=feature_names, out_file=dot_data1,
                                max_depth = 2)
#graph = graphviz.Source(dot_data)
```

In [27]:
```
from IPython.display import Image
import pydot
graph = pydot.graph_from_dot_data(dot_data1.getvalue())[0]
```

In [28]:
```
Image(graph.create_png())
```

Out[28]:



gridsearch and random search for decison tree,

All Metrics for Decision tree classifier with TF-IDF

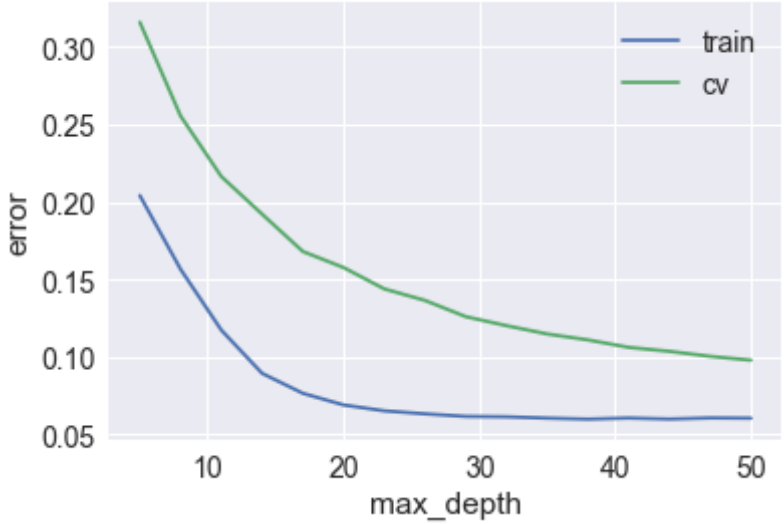
In [10]:
```
tf_idf_vect = TfidfVectorizer(ngram_range=(1,2),min_df=10)
tf_idf_vect1 = TfidfVectorizer(ngram_range=(1,2))
final_tf_idf = tf_idf_vect.fit(X_tr['CleanedText'].values)
final_tf_idf1 = tf_idf_vect1.fit(X_tr['CleanedText'].values)
```

In [11]:
```
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 [37]:
```
from sklearn import preprocessing
from sklearn import tree


hyperparameter = dict(max_depth=[5,8,11,14,17,20,23,26,29,32,35,38,41,44,47,50])

#Using GridSearchCV
modeltf = GridSearchCV(tree.DecisionTreeClassifier(class_weight='balanced'), hyperparameter, scoring = 'f1', cv=5)
modeltf.fit(X_tr_tf_idf, y_tr)
a=modeltf.grid_scores_
print(modeltf.best_estimator_)
print(modeltf.score(X_test_tf_idf, y_test))
modeltf.fit(X_cv_tf_idf, y_cv)
b=modeltf.grid_scores_
```

```
DecisionTreeClassifier(class_weight='balanced', criterion='gini',
            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, presort=False, random_state=None,
            splitter='best')
0.9137381448978787
```

In [61]:
```
scores = [x[1] for x in a]
cv_scores = [x[1] for x in b]
scores = np.array(scores).reshape(len(hyperparameter['max_depth']),1)
cv_scores = np.array(cv_scores).reshape(len(hyperparameter['max_depth'
]),1)
#for i in enumerate(hyperparameter['max_depth']):
plt.plot(hyperparameter['max_depth'],1- scores,label='train')
plt.plot(hyperparameter['max_depth'],1- cv_scores,label='cv')
plt.legend()
plt.xlabel('max_depth')
plt.ylabel('error')
plt.show()
```

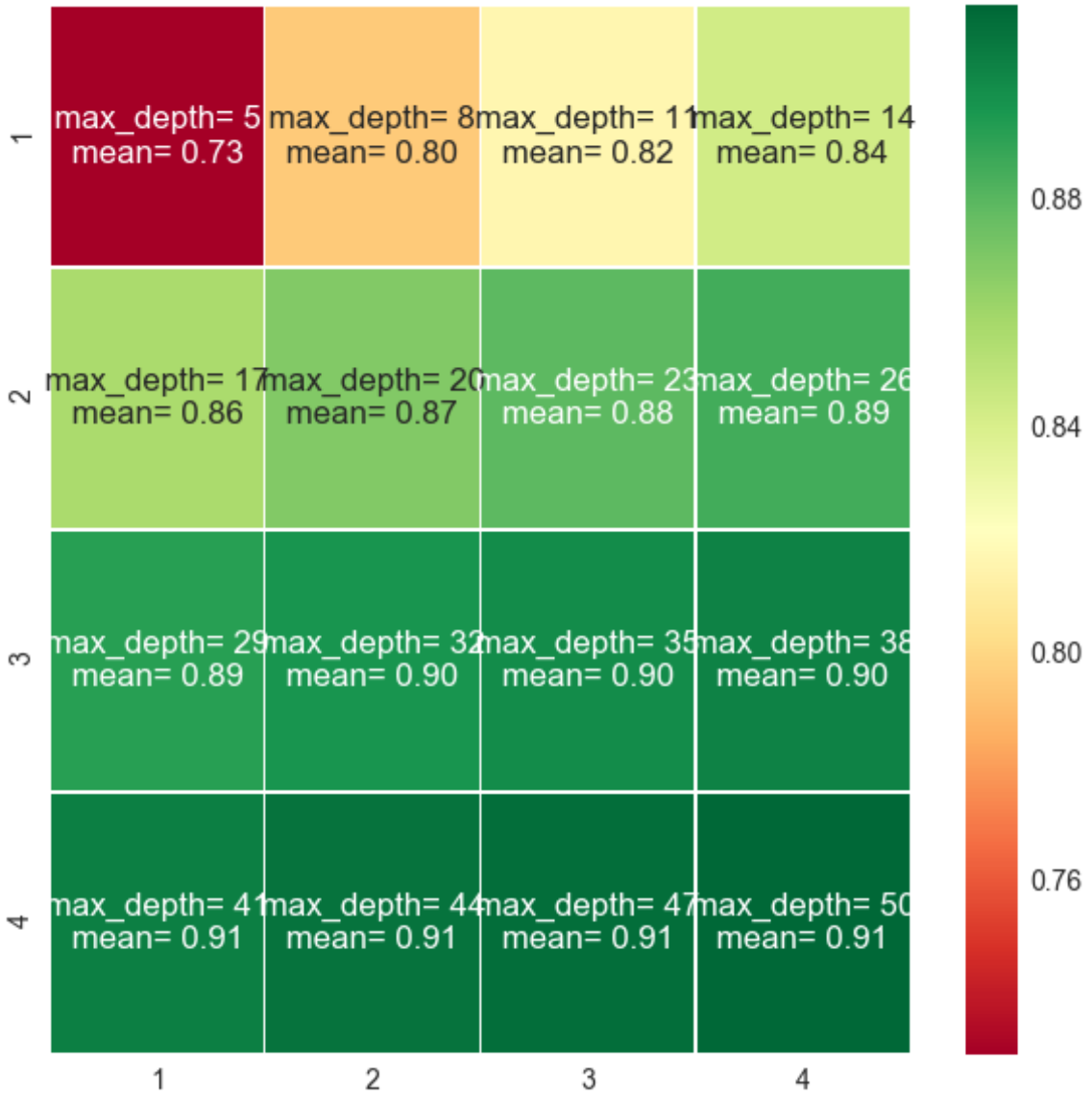


In [40]:
```
max_depth=[]

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

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

label =np.asarray([" max_depth= {0} \n mean= {1:.2f} ".format(max_depth
,mean) for max_depth,mean in zip(max_depth.flatten(),mean.flatten())]).
reshape(4,4)
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= 5 \n mean= 0.73 ' ' max_depth= 8 \n mean= 0.80 '
  ' max_depth= 11 \n mean= 0.82 ' ' max_depth= 14 \n mean= 0.84 ']
 [' max_depth= 17 \n mean= 0.86 ' ' max_depth= 20 \n mean= 0.87 '
  ' max_depth= 23 \n mean= 0.88 ' ' max_depth= 26 \n mean= 0.89 ']
 [' max_depth= 29 \n mean= 0.89 ' ' max_depth= 32 \n mean= 0.90 '
  ' max_depth= 35 \n mean= 0.90 ' ' max_depth= 38 \n mean= 0.90 ']
 [' max_depth= 41 \n mean= 0.91 ' ' max_depth= 44 \n mean= 0.91 '
  ' max_depth= 47 \n mean= 0.91 ' ' max_depth= 50 \n mean= 0.91 ']]
```

Out[40]: <matplotlib.axes._subplots.AxesSubplot at 0x215c07797f0>



In [42]:
```
clf = tree.DecisionTreeClassifier(class_weight='balanced', criterion='gini',
            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, presort=False, random_state=N
one,
            splitter='best')
clf=clf.fit(X_tr_tf_idf, y_tr)
```

In [44]:
```
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 =  0.07324489795918365
test-error =  0.14529999999999998
```

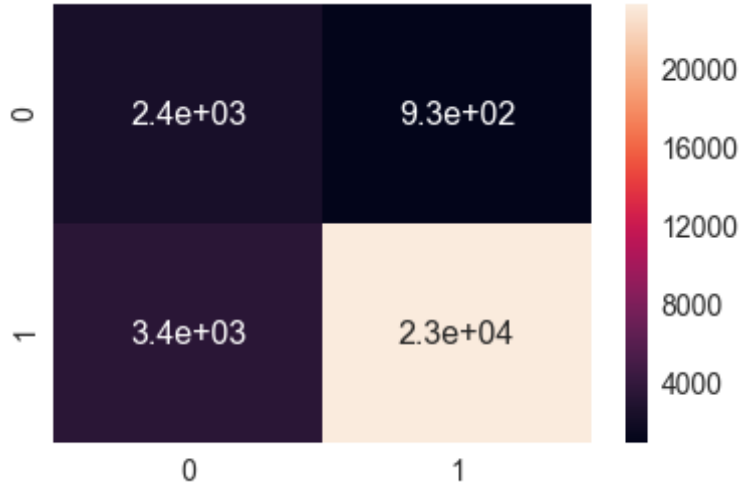
In [45]:
```
important_features(tf_idf_vect,clf,n=20)
```

```
[0 1]
Important words in  reviews
great
love
best
disappoint
delici
good
perfect
excel
tast
nice
favorit
like
would
high recommend
bad
find
wonder
tri
bought
use
```

In [48]:
```
pred = clf.predict(X_test_tf_idf)
pred1 = clf.predict(X_tr_tf_idf)
```

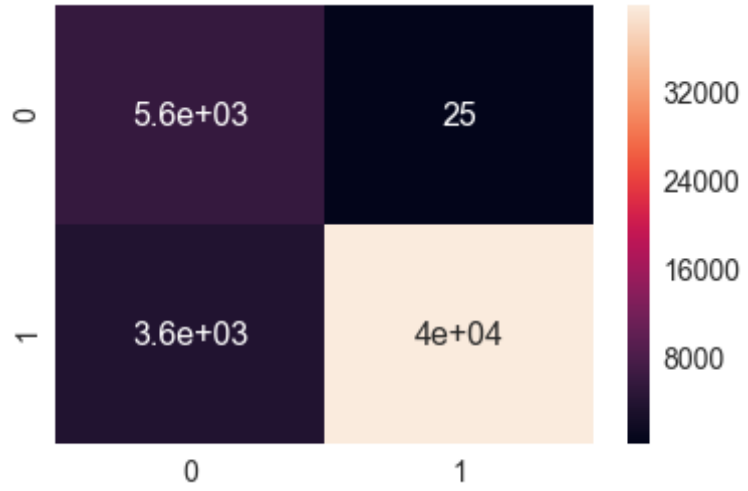
```
from sklearn.metrics import confusion_matrix
import seaborn as sn
CFM = confusion_matrix(y_test, pred)
CFMtr = confusion_matrix(y_tr, pred1)
df_cm = pd.DataFrame(CFM, range(2), range(2))
#plt.figure(figsize = (10,7))
sn.set(font_scale=1.4)#for label size
sn.heatmap(df_cm, annot=True,annot_kws={"size":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

Out[48]: <matplotlib.axes._subplots.AxesSubplot at 0x215d2074908>



In [49]:
```
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[49]: <matplotlib.axes._subplots.AxesSubplot at 0x215d3e247f0>



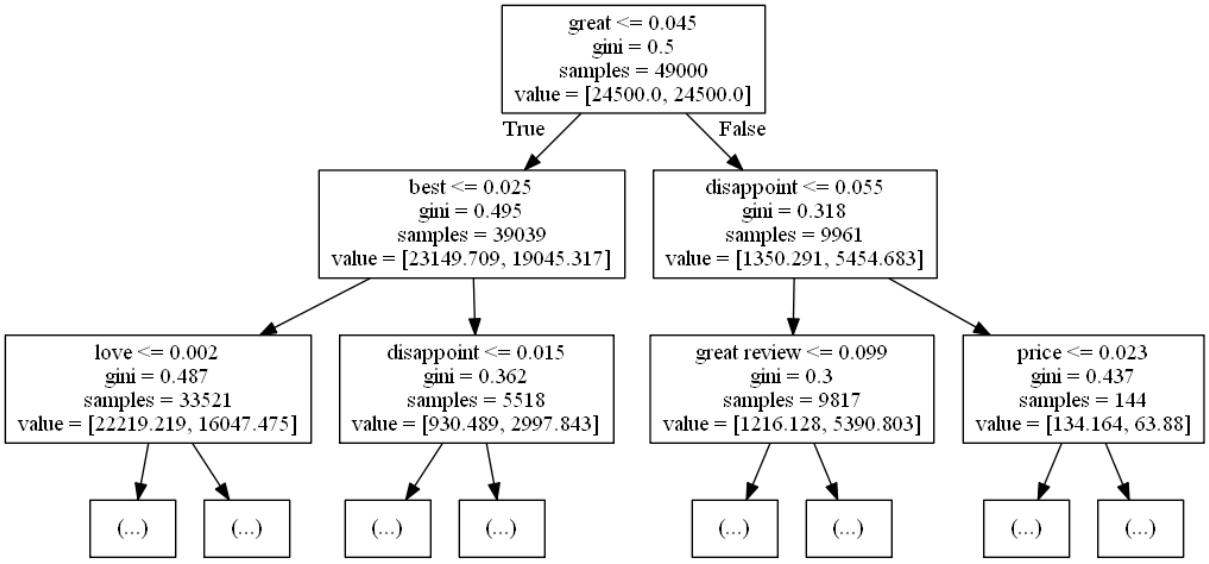
In [50]:
```
from sklearn.metrics import accuracy_score, f1_score, precision_score,
recall_score
print(accuracy_score(y_test, pred))
print(f1_score(y_test, pred, average="macro"))
print(precision_score(y_test, pred, average="macro"))
print(recall_score(y_test, pred, average="macro"))
```

```
0.8547
0.7196700190498124
0.6871405142727494
0.7963769735699642
```

In [52]:
```
feature_names =tf_idf_vect.get_feature_names()
import os
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_data2 = StringIO()
dot_data = tree.export_graphviz(clf, out_file=dot_data2,feature_names=f
eature_names,max_depth = 2)
#graph = graphviz.Source(dot_data)
```

In [53]:
```
from IPython.display import Image
import pydot
graph = pydot.graph_from_dot_data(dot_data2.getvalue())[0]
Image(graph.create_png())
```

Out[53]:



gridsearch and random search for decison tree, All Metrics for Decision tree classifier with W2V

In [12]:
```
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)
```

In [13]:
```
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 [00:49<00:00, 985.74it/s]
```

In [14]:
```
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 [00:32<00:00, 937.35it/s]
```

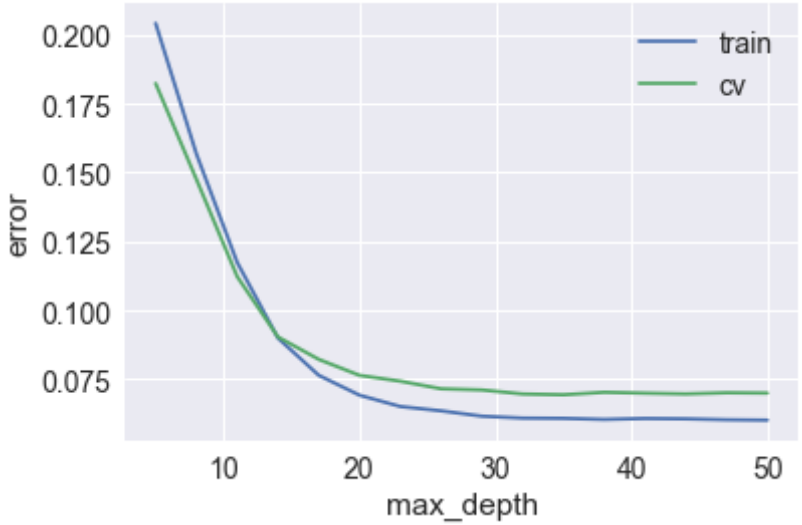
In [15]:
```
list_of_sent2 = []
for sent in X_cv['CleanedText'].values:
    list_of_sent2.append(sent.split())
sent_vectorscv = []; # the avg-w2v for each sentence/review is stored i
n this list
for sent in tqdm(list_of_sent2): # for each review/sentence
    sent_vec = np.zeros(50) # as word vectors 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 [00:22<00:00, 951.65it/s]
```

In [69]:
```
from sklearn import preprocessing
from sklearn import tree


hyperparameter = dict(max_depth=[5,8,11,14,17,20,23,26,29,32,35,38,41,4
4,47,50])

#Using GridSearchCV
modelw2v = GridSearchCV(tree.DecisionTreeClassifier(class_weight='balan
ced'), hyperparameter, scoring = 'f1', cv=5)
modelw2v.fit(sent_vectorstr, y_tr)
a=modelw2v.grid_scores_
print(modelw2v.best_estimator_)
print(modelw2v.score(sent_vectorstest, y_test))
modelw2v.fit(sent_vectorscv, y_cv)
b=modelw2v.grid_scores_
```

```
DecisionTreeClassifier(class_weight='balanced', criterion='gini',
            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, presort=False, random_state=N
one,
            splitter='best')
0.942390305213199
```

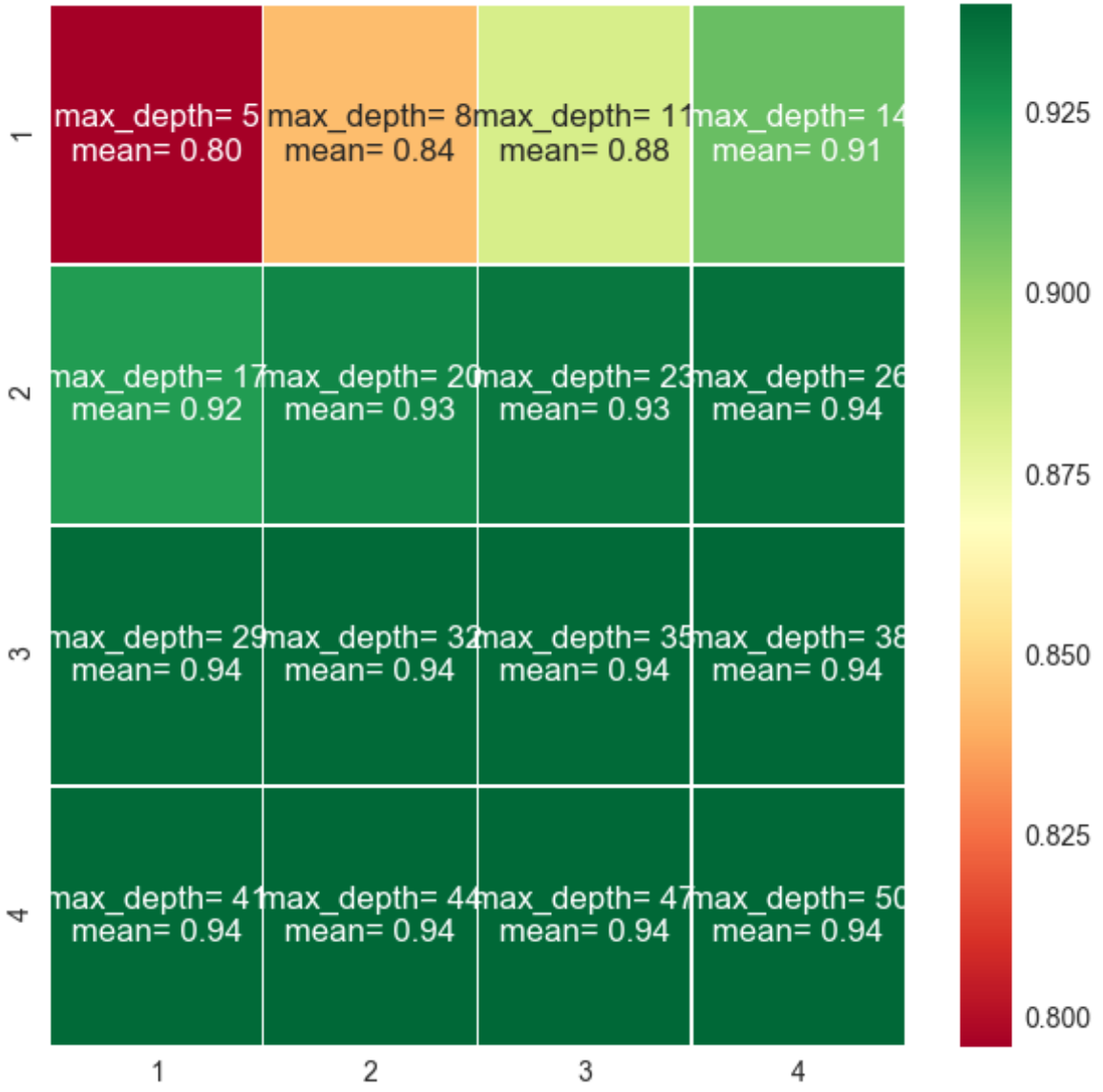
In [71]:
```
scores = [x[1] for x in a]
cv_scores = [x[1] for x in b]
scores = np.array(scores).reshape(len(hyperparameter['max_depth']),1)
cv_scores = np.array(cv_scores).reshape(len(hyperparameter['max_depth'
]),1)
#for i in enumerate(hyperparameter['max_depth']):
plt.plot(hyperparameter['max_depth'],1- scores,label='train')
plt.plot(hyperparameter['max_depth'],1- cv_scores,label='cv')
plt.legend()
plt.xlabel('max_depth')
plt.ylabel('error')
plt.show()
```



In [72]:
```
max_depth=[]

mean=[]
for a in a:
    max_depth.append(a[0]['max_depth'])
    mean.append(a[1])
max_depth=np.asarray(max_depth)
mean=np.asarray(mean)
max_depth = max_depth.reshape(4,4)

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

label =np.asarray([" max_depth= {0} \n mean= {1:.2f} ".format(max_depth
,mean) for max_depth,mean in zip(max_depth.flatten(),mean.flatten())]).
reshape(4,4)
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= 5 \n mean= 0.80 ' ' max_depth= 8 \n mean= 0.84 '
  ' max_depth= 11 \n mean= 0.88 ' ' max_depth= 14 \n mean= 0.91 ']
 [' max_depth= 17 \n mean= 0.92 ' ' max_depth= 20 \n mean= 0.93 '
  ' max_depth= 23 \n mean= 0.93 ' ' max_depth= 26 \n mean= 0.94 ']
 [' max_depth= 29 \n mean= 0.94 ' ' max_depth= 32 \n mean= 0.94 '
  ' max_depth= 35 \n mean= 0.94 ' ' max_depth= 38 \n mean= 0.94 ']
 [' max_depth= 41 \n mean= 0.94 ' ' max_depth= 44 \n mean= 0.94 '
  ' max_depth= 47 \n mean= 0.94 ' ' max_depth= 50 \n mean= 0.94 ']]
```

Out[72]:
```
<matplotlib.axes._subplots.AxesSubplot at 0x215f0fe9f60>
```



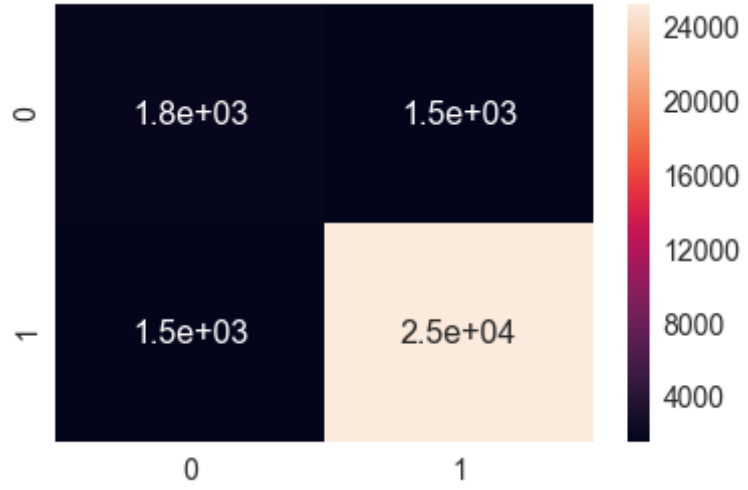
In [74]:
```
clf = tree.DecisionTreeClassifier(class_weight='balanced', criterion='gini',
            max_depth=50, max_features=None, max_leaf_nodes=None,
            min_impurity_decrease=0.0, min_impurity_split=None,
```

```
            min_samples_leaf=1, min_samples_split=2,
            min_weight_fraction_leaf=0.0, presort=False, random_state=N
one,
            splitter='best')
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 =  0.0002448979591836986
test-error =  0.10183333333333333
```

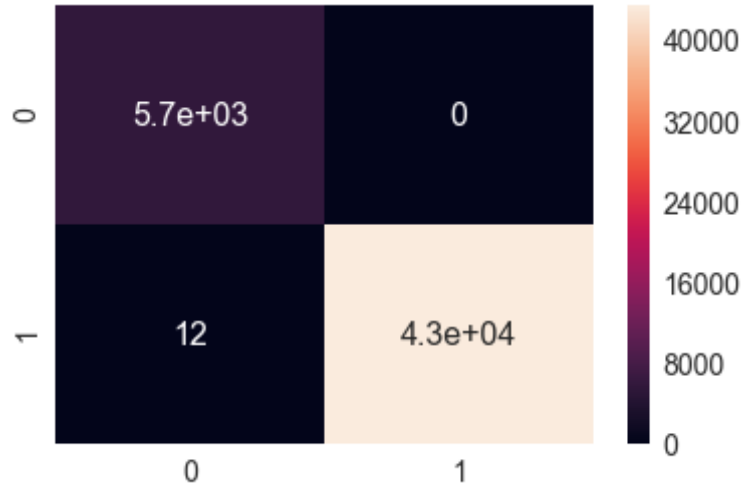
In [76]:
```
pred = clf.predict(sent_vectorstest)
pred1=clf.predict(sent_vectorstr)
from sklearn.metrics import confusion_matrix
import seaborn as sn
CFM = confusion_matrix(y_test, pred)
CFMtr = confusion_matrix(y_tr, pred1)
df_cm = pd.DataFrame(CFM, range(2), range(2))
#plt.figure(figsize = (10,7))
sn.set(font_scale=1.4)#for label size
sn.heatmap(df_cm, annot=True,annot_kws={"size": 16})
```

Out[76]: <matplotlib.axes._subplots.AxesSubplot at 0x215f10999e8>



In [77]:
```
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[77]:
```
<matplotlib.axes._subplots.AxesSubplot at 0x215f1087cc0>
```



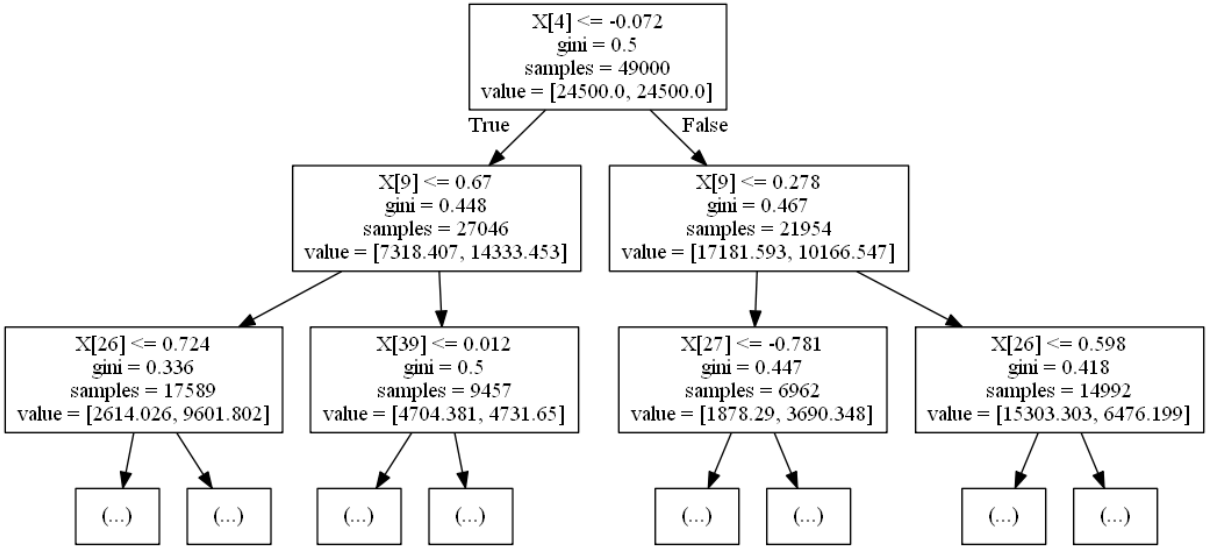
In [78]:
```
from sklearn.metrics import accuracy_score, f1_score, precision_score,
recall_score
print(accuracy_score(y_test, pred))
print(f1_score(y_test, pred, average="macro"))
print(precision_score(y_test, pred, average="macro"))
print(recall_score(y_test, pred, average="macro"))
```

```
0.8981666666666667
0.7428528921439033
0.7427576293675364
0.7429482672616776
```

In [79]:
```
import os
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_data4 = StringIO()
dot_data = tree.export_graphviz(clf, out_file=dot_data4,
                                max_depth = 2)
#graph = graphviz.Source(dot_data)
from IPython.display import Image
import pydot
graph = pydot.graph_from_dot_data(dot_data4.getvalue())[0]
Image(graph.create_png())
```

Out[79]:



gridsearch and random search for decison tree, All Metrics for Decision tree classifier with TF-IDF W2V

In [97]:
```
tf_idf_vect1 = TfidfVectorizer(ngram_range=(1,2))
final_tf_idf1 = tf_idf_vect.fit(X_tr['CleanedText'].values)
```

In [16]:
```
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 [01:05<00:00, 749.20it/s]
```

In [17]:

```
# 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 [00:37<00:00, 792.57it/s]
```

In [18]:

```
# 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 [00:25<00:00, 813.55it/s]
```

In [119]:

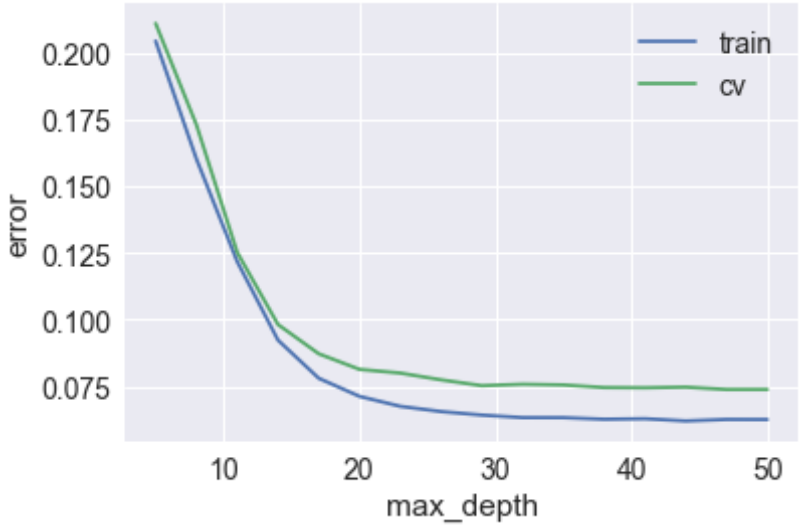
```
from sklearn import preprocessing
from sklearn import tree


hyperparameter = dict(max_depth=[5,8,11,14,17,20,23,26,29,32,35,38,41,44,47,50])

#Using GridSearchCV
modeltfw2v = GridSearchCV(tree.DecisionTreeClassifier(class_weight='balanced'), hyperparameter, scoring = 'f1', cv=5)
modeltfw2v.fit(tfidf_sent_vectors, y_tr)
a=modeltfw2v.grid_scores_
print(modeltfw2v.best_estimator_)
print(modeltfw2v.score(tfidf_sent_vectors_test, y_test))
modeltfw2v.fit(tfidf_sent_vectors_cv, y_cv)
b=modeltfw2v.grid_scores_
```

```
DecisionTreeClassifier(class_weight='balanced', criterion='gini',
            max_depth=44, 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, presort=False, random_state=None,
            splitter='best')
0.9383907614308515
```

In [120]:

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

```
plt.plot(hyperparameter['max_depth'],1- cv_scores,label='cv')
plt.legend()
plt.xlabel('max_depth')
plt.ylabel('error')
plt.show()
```

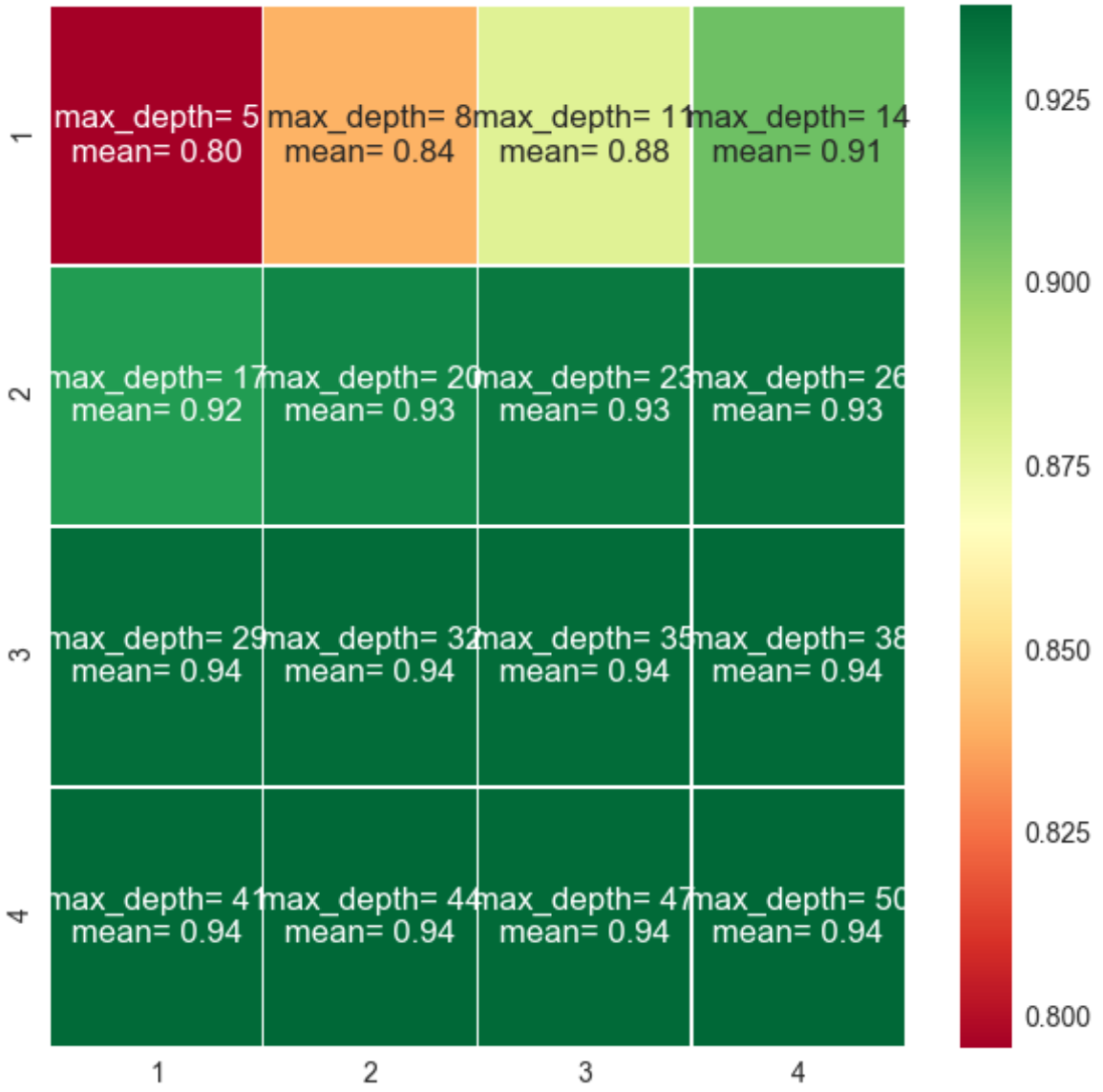


In [121]:
```
max_depth=[]

mean=[]
for a in a:
    max_depth.append(a[0]['max_depth'])
    mean.append(a[1])
max_depth=np.asarray(max_depth)
mean=np.asarray(mean)
max_depth = max_depth.reshape(4,4)

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

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

```
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= 5 \n mean= 0.80 ' ' max_depth= 8 \n mean= 0.84 '
  ' max_depth= 11 \n mean= 0.88 ' ' max_depth= 14 \n mean= 0.91 ']
 [' max_depth= 17 \n mean= 0.92 ' ' max_depth= 20 \n mean= 0.93 '
  ' max_depth= 23 \n mean= 0.93 ' ' max_depth= 26 \n mean= 0.93 ']
 [' max_depth= 29 \n mean= 0.94 ' ' max_depth= 32 \n mean= 0.94 '
  ' max_depth= 35 \n mean= 0.94 ' ' max_depth= 38 \n mean= 0.94 ']
 [' max_depth= 41 \n mean= 0.94 ' ' max_depth= 44 \n mean= 0.94 '
  ' max_depth= 47 \n mean= 0.94 ' ' max_depth= 50 \n mean= 0.94 ']]
```

Out[121]: <matplotlib.axes._subplots.AxesSubplot at 0x215f4aa5898>



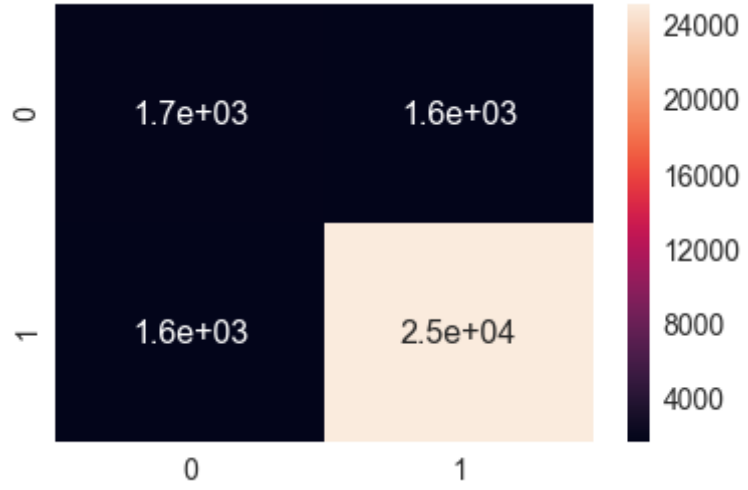
In [123]:
```
clf = tree.DecisionTreeClassifier(class_weight='balanced', criterion='gini',
            max_depth=44, max_features=None, max_leaf_nodes=None,
            min_impurity_decrease=0.0, min_impurity_split=None,
```

```
            min_samples_leaf=1, min_samples_split=2,
            min_weight_fraction_leaf=0.0, presort=False, random_state=N
one,
            splitter='best')
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 =  0.00030612244897954
test-error =  0.10840000000000005
```

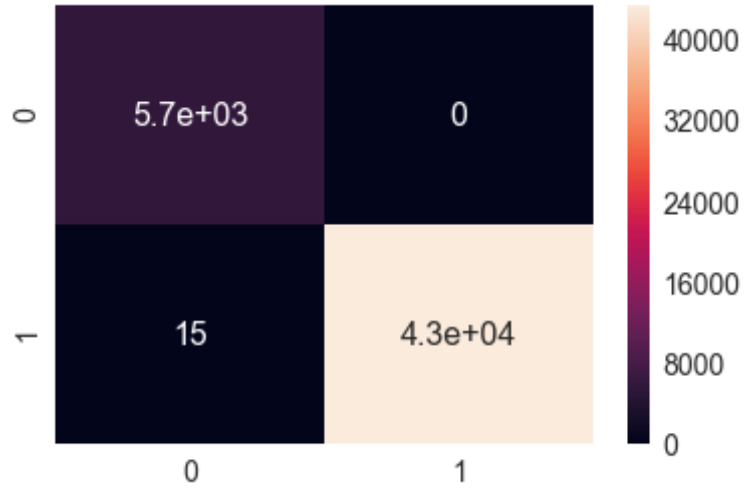
In [127]:
```
pred = clf.predict(tfidf_sent_vectors_test)
pred1 = clf.predict(tfidf_sent_vectors)
from sklearn.metrics import confusion_matrix
import seaborn as sn
CFM = confusion_matrix(y_test, pred)
CFMtr = confusion_matrix(y_tr, pred1)
df_cm = pd.DataFrame(CFM, range(2), range(2))
#plt.figure(figsize = (10,7))
sn.set(font_scale=1.4)#for label size
sn.heatmap(df_cm, annot=True,annot_kws={"size": 16})
```

Out[127]: <matplotlib.axes._subplots.AxesSubplot at 0x215f203cb38>



In [128]:
```
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[128]: <matplotlib.axes._subplots.AxesSubplot at 0x215f203c400>



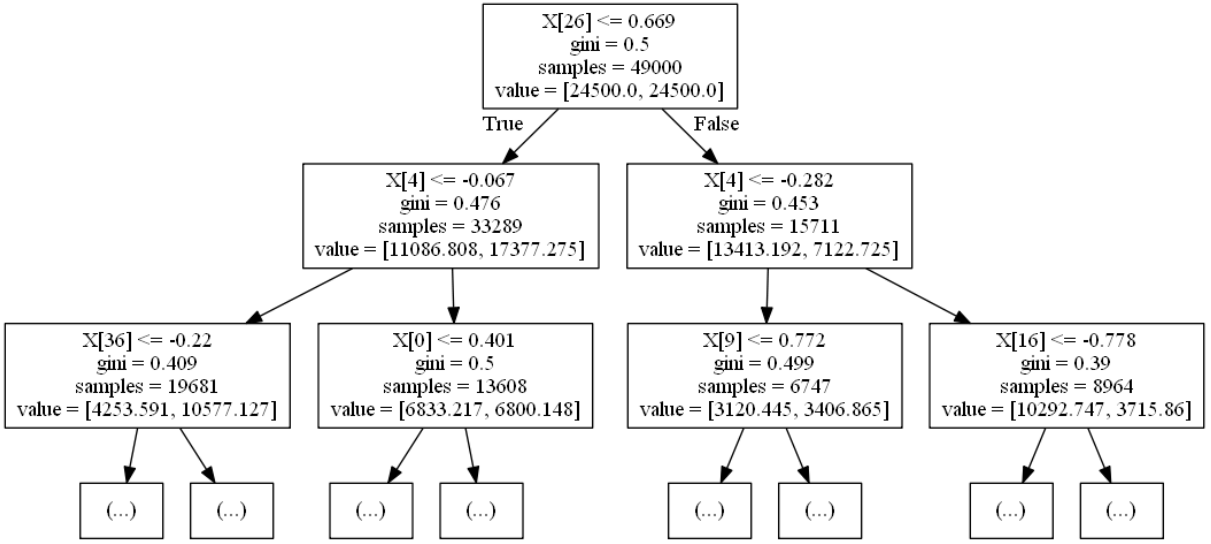
In [129]:
```
from sklearn.metrics import accuracy_score, f1_score, precision_score,
recall_score
print(accuracy_score(y_test, pred))
print(f1_score(y_test, pred, average="macro"))
print(precision_score(y_test, pred, average="macro"))
print(recall_score(y_test, pred, average="macro"))
```

```
0.8916
0.726878143609891
0.7262885846565068
0.7274724190478035
```

In [130]:
```
import os
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_data6 = StringIO()
dot_data = tree.export_graphviz(clf, out_file=dot_data6,
                                max_depth = 2)
#graph = graphviz.Source(dot_data)
from IPython.display import Image
import pydot
graph = pydot.graph_from_dot_data(dot_data6.getvalue())[0]
Image(graph.create_png())
```

Out[130]:



In [132]:

```
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
Train error
Please add a BOW to the list
0.0614
Please enter a TF-IDF for the service
0.0732
Please enter a W2V for the service
0.0002
Please enter a TFIDF W2V for the service
0.0003
Continue yes/ no:y
Please add a head to the list
Test error
Please add a BOW to the list
0.133
Please enter a TF-IDF for the service
0.145
Please enter a W2V for the service
0.101
Please enter a TFIDF W2V for the service
0.108
Continue yes/ no:n
```

In [133]:
```
print(x)
```

```
+-------------+--------+--------+--------+-----------+
|    Table    |  BOW   | TF-IDF |  W2V   | TFIDF W2V |
+-------------+--------+--------+--------+-----------+
```

```
| Train error | 0.0614 | 0.0732 | 0.0002 |   0.0003  |
|  Test error | 0.133  | 0.145  | 0.101  |   0.108   |
+-------------+--------+--------+--------+-----------+
```