Decision Trees On Amazon Food Reviews

Importing Essential Packages

In [0]:

```
!pip install pydotplus
from sklearn.model_selection import train_test_split,GridSearchCV,RandomizedSearchCV
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
from sklearn.metrics import roc_auc_score,confusion_matrix
import numpy as np
import warnings
warnings.filterwarnings("ignore")
import seaborn as sns
from sklearn.tree import DecisionTreeClassifier
!pip install wordcloud
from wordcloud import WordCloud
from sklearn.pipeline import Pipeline, FeatureUnion
from sklearn.preprocessing import FunctionTransformer
from sklearn.externals.six import StringIO
from IPython.display import Image
from sklearn.tree import export_graphviz
import pydotplus
import nltk
nltk.download('stopwords')
from nltk.corpus import stopwords
!pip install vaderSentiment
from vaderSentiment.vaderSentiment import SentimentIntensityAnalyzer
import os
%env JOBLIB_TEMP_FOLDER=/tmp
from IPython.core.display import display, HTML
```

Input Data Processing

Fetching and Preprocessing

In [0]:

```
!pip install -q kaggle
!mkdir -p ~/.kaggle
!cp kaggle.json ~/.kaggle/
!kaggle datasets download -d snap/amazon-fine-food-reviews
!unzip amazon-fine-food-reviews.zip

input_data = pd.read_csv('Reviews.csv')

# Removing all the neutral reviews
input_data = input_data[input_data.Score != 3]
sorted_data=input_data.sort_values('ProductId', axis=0, ascending=True, inplace=False, kind='quicksort', na_position='last')
input_data = sorted_data.drop_duplicates(subset={"UserId","ProfileName","Time","Text"}, keep='first', inplace=False)
input_data = input_data[input_data.HelpfulnessNumerator<=input_data.HelpfulnessDenominator]

data = input_data.iloc[input_data.Time.argsort()]
```

In [0]:

```
stop = set(stopwords.words('english'))

def cleanhtml(sentence):
    cleanr = re.compile('<.*?>')
    cleantext = re.sub(cleanr, ' ', sentence)
    return cleantext
```

```
def cleanpunc(sentence):
    cleaned = re.sub(r'[?|!|\'|"|#]',r'',sentence)
    cleaned = re.sub(r'[.|,|)|(|\|/]',r' ',cleaned)
    return  cleaned


#Cleaning Review Text
import re
i=0
str1=' '
final_string=[]
s=''
for sent in data['Text'].values:
    filtered_sentence=[]
    sent=cleanhtml(sent)
    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):
                    filtered_sentence.append(cleaned_words.lower())
                else:
                    continue
            else:
                continue
    str1 = ' '.join(filtered_sentence)
    final_string.append(str1)
    i+=1


data['Cleaned']=final_string


#Cleaning Summary Text
i=0
str1=' '
final_string=[]
s=''
for sent in data['Summary'].astype(str):
    filtered_sentence=[]
    sent=cleanhtml(sent)
    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):
                    filtered_sentence.append(cleaned_words.lower())
                else:
                    continue
            else:
                continue
    str1 = ' '.join(filtered_sentence)
    final_string.append(str1)
    i+=1


data['Clean_summary']=final_string
```

In [0]:

```
# Omitting null values rows in the speicified columns
data = data[data[['Cleaned','Clean_summary','Score']].notnull()]
# Counting the number of words in the reviews
data["rev_len"] = data['Cleaned'].str.split().str.len()
data['Cleaned'] = data.Cleaned.astype(str)

# Using Vader Pre Trained Corpus Of Strings
analyser = SentimentIntensityAnalyzer()
def vaderAnalysis(sentence):
  """Method to return the positive score of the summary review"""
  return (analyser.polarity_scores(sentence)['pos'])
# Creating a new column to include the positive scores
data['sum_pos_score'] = [vaderAnalysis(i) for i in data.Clean_summary.values]
# Any review with a score greater than 3 is considered to be a positive review
# 1 is positive and 0 is negative
data.Score = data.Score.map(lambda x : 1 if (x > 3) else 0)
```

Split Into Train and Test

In [0]:

```
x_train,x_test,y_train,y_test = train_test_split(data[['Cleaned','sum_pos_score','rev_len']],data.
Score.values,shuffle=False,test_size=0.3)
```

Method Declarations

In [0]:

```
from sklearn.model_selection import TimeSeriesSplit
param_grid = {'max_depth': [2,3,5,8,11,15,20,26,32] ,'min_samples_split': list(range(2,11,2))}
tss = TimeSeriesSplit(n_splits=10)
def best_param_search(train_data,train_label,params,tss):
  """ To choose the best hyperparamters for the Tree Classifier"""
  dtc = DecisionTreeClassifier(criterion = 'gini',splitter = 'best',class_weight = 'balanced')
  rscv = RandomizedSearchCV(dtc,params, scoring = 'roc_auc', cv=tss, n_jobs = -1,verbose = 1,n_iter
= 15)
  rscv.fit(train_data, train_label)
  params = rscv.cv_results_['params']
  train_scores = rscv.cv_results_['mean_train_score']
  cv_scores = rscv.cv_results_['mean_test_score']
  pr = []
  for i in params:
    depth_val = str(i['max_depth'])
    sample_split_val = str(i['min_samples_split'])
    pr.append(depth_val+','+sample_split_val)
  df_cm = pd.DataFrame(data = [train_scores,cv_scores],index=['Train','CV'], columns=pr)
  plt.figure(figsize=(25, 7))
  heatmap = sns.heatmap(df_cm, annot=True, fmt="f")
  heatmap.yaxis.set_ticklabels(heatmap.yaxis.get_ticklabels(), rotation=0, ha='right')
  heatmap.xaxis.set_ticklabels(heatmap.xaxis.get_ticklabels(), rotation=45, ha='right')
  plt.ylabel('Type')
  plt.xlabel('(max_depth,min_samples_split) Params')
  plt.title('Grid Results Heat Map')
  plt.show()

def genGraph(classifier,feature_names):
  """To viz the tree structure"""
  export_graphviz(classifier, out_file='tree.dot',filled=True, max_depth = 3,rounded=True,special_c
haracters=True,feature_names = feature_names)
  os.system('dot -Tpng tree.dot -o tree.png')
  display(Image('tree.png',width = 1000))
  #display(HTML("<style>.container { width:100% !important; }</style>"))
  os.system('rm tree.png')

# Confusion Matrix
def confusion_matrix_display(conf_mtrx,tst_labels,Title):
  class_names = [0,1]
  df_cm = pd.DataFrame(conf_mtrx, index=class_names, columns=class_names)
  TN, FP, FN, TP = conf_mtrx.ravel()
  heatmap = sns.heatmap(df_cm, annot=True, fmt="d")
  heatmap.yaxis.set_ticklabels(heatmap.yaxis.get_ticklabels(), rotation=0, ha='right')
  heatmap.xaxis.set_ticklabels(heatmap.xaxis.get_ticklabels(), rotation=45, ha='right')
  plt.ylabel('True label')
  plt.xlabel('Predicted label')
  plt.title(Title + ' Confusion Matrix')
  plt.show()
  print('\nThe TPR is : ',TP/(TP+FN))
  print('The TNR is : ',TN/(TN+FP))
  print('The FPR is : ',FP/(FP+TN))
  print('The FNR is : ',FN/(TP+FN),'\n')

def on_test(train_data,train_label,test_data,test_label,md,ms,feature_names):
  dtc = DecisionTreeClassifier(criterion = 'gini',splitter = 'best',class_weight = 'balanced',max_d
epth = md,min_samples_split = ms)
  dtc.fit(train_data,train_label)
  print('The ROC AUC score for the params max_depth' ,md, ' and min_samples_split ',ms, 'is ', roc
_auc_score(test_label,dtc.predict_proba(test_data)[:,1]))
  confusion_matrix_display(confusion_matrix(train_label,dtc.predict(train_data)),train_label,'Train
Data Confusion Matrix')
```

```
Data Confusion Matrix')
  confusion_matrix_display(confusion_matrix(test_label,dtc.predict(test_data)),test_label,'Test
Data Confusion Matrix')
  return dtc

def Wordcl(title,val):
  wordcloud = WordCloud(
                          background_color='white',
                          max_words=200,
                          max_font_size=40,
                          random_state=42
                         ).generate(str(val))
  fig = plt.figure(1)
  plt.imshow(wordcloud)
  plt.axis('off')
  plt.title(title)
  plt.show()
```

Bag Of Words Model

In [0]:

```
from sklearn.feature_extraction.text import CountVectorizer
BoW_dict = CountVectorizer(min_df=8,max_features = 5000,ngram_range = (1,2)).fit(x_train.Cleaned)
BoW_train = BoW_dict.transform(x_train.Cleaned)
BoW_test = BoW_dict.transform(x_test.Cleaned)

rev_lens_train = x_train.rev_len.values.reshape(-1,1)
sum_pos_score_train = x_train.sum_pos_score.values.reshape(-1,1)
from scipy import sparse
training_data = sparse.hstack((BoW_train, rev_lens_train,sum_pos_score_train))
feat_names = []
feat_names = BoW_dict.get_feature_names()
feat_names.append('review_length')
feat_names.append('pos_score')
```

Grid Results

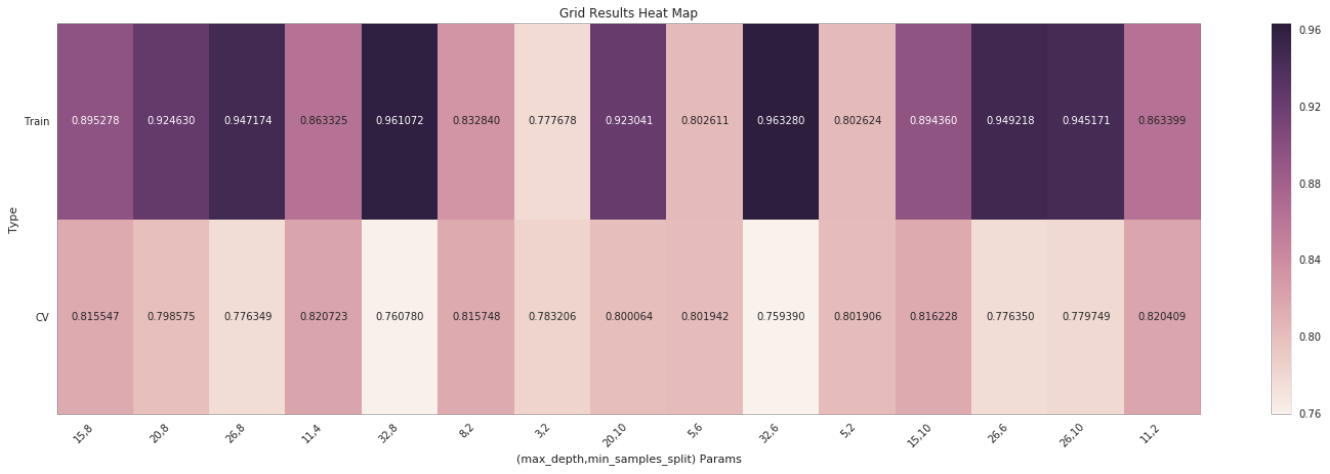
In [0]:

```
best_param_search(training_data,y_train,param_grid,tss)
```

```
Fitting 10 folds for each of 15 candidates, totalling 150 fits
```

```
[Parallel(n_jobs=-1)]: Done  34 tasks      | elapsed:  3.5min
[Parallel(n_jobs=-1)]: Done 150 out of 150 | elapsed: 15.8min finished
```

Out[0]:



Increasing the features from a range in 1000 to 5000 has not improved the model further

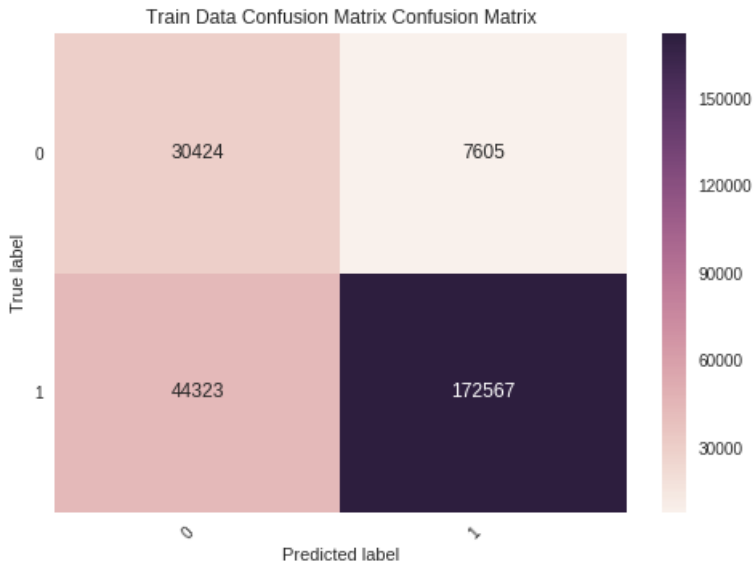
On Test Data

In [9]:

```
rev_lens_test = x_test.rev_len.values.reshape(-1,1)
sum_pos_score_test = x_test.sum_pos_score.values.reshape(-1,1)
test_data = sparse.hstack((BoW_test, rev_lens_test,sum_pos_score_test))
dtc = on_test(training_data,y_train,test_data,y_test,11,4,feat_names)
```

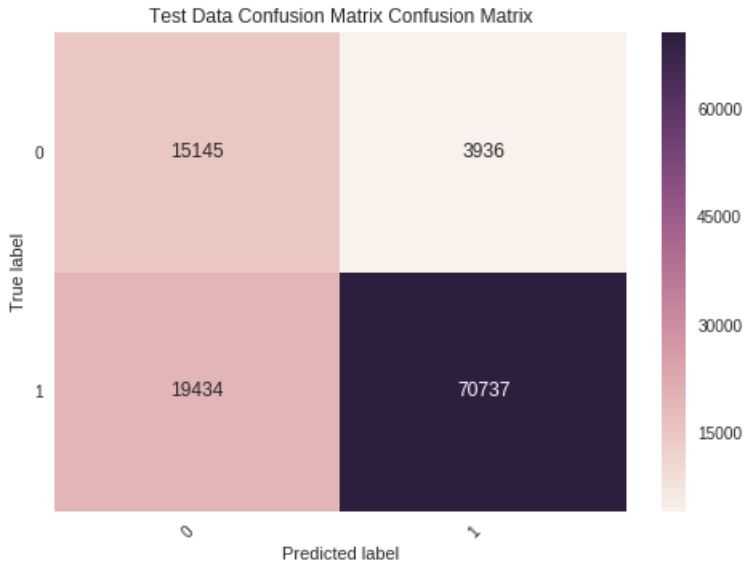
```
The ROC AUC score for the params max_depth 11  and min_samples_split  4 is  0.843667202176517
```

Out[9]:




```
The TPR is :  0.7956429526488081
The TNR is :  0.8000210365773489
The FPR is :  0.19997896342265115
The FNR is :  0.20435704735119184
```

Out[9]:



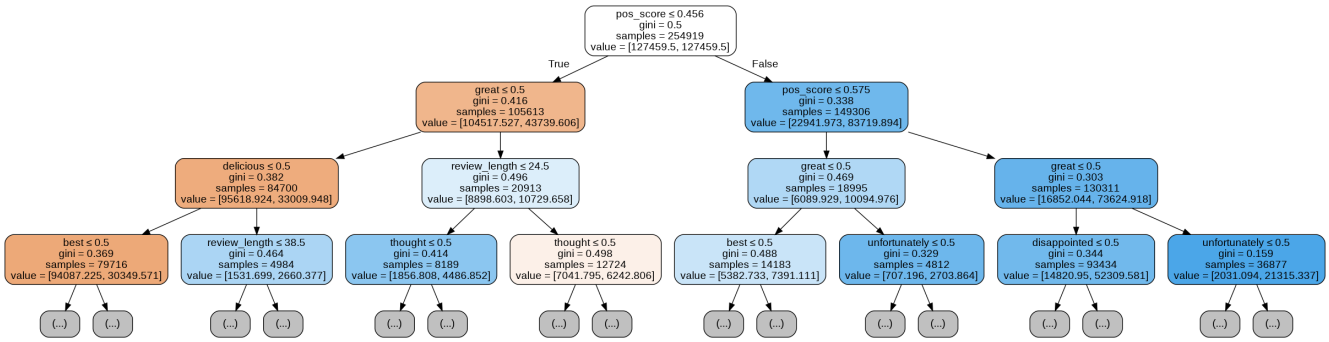

```
The TPR is :  0.7844761619589447
The TNR is :  0.7937215030658771
The FPR is :  0.20627849693412295
The FNR is :  0.21552383804105532
```

Tree Visualization

In [19]:

```
genGraph(dtc,feat_names)
```

Out[19]:



Important Features WordCloud

In [0]:

```
top_200 = np.argsort(dtc.feature_importances_)[-200:].tolist()
vals  = [feat_names[i] for i in top_200]
Wordcl('BoW Important Features',vals)
```

Out[0]:

TFIDF

In [0]:

```
from sklearn.feature_extraction.text import TfidfVectorizer
TFIDF_dict = TfidfVectorizer(min_df=10,max_features = 5000,ngram_range = (1,2)).fit(x_train.Cleaned
)
TFIDF_train = TFIDF_dict.transform(x_train.Cleaned)
TFIDF_test = TFIDF_dict.transform(x_test.Cleaned)

rev_lens_train = x_train.rev_len.values.reshape(-1,1)
sum_pos_score_train = x_train.sum_pos_score.values.reshape(-1,1)
training_data = sparse.hstack((TFIDF_train, rev_lens_train,sum_pos_score_train))
feat_names = []
feat_names = TFIDF_dict.get_feature_names()
feat_names.append('review_length')
feat_names.append('pos_score')
```

Grid Results

In [0]:

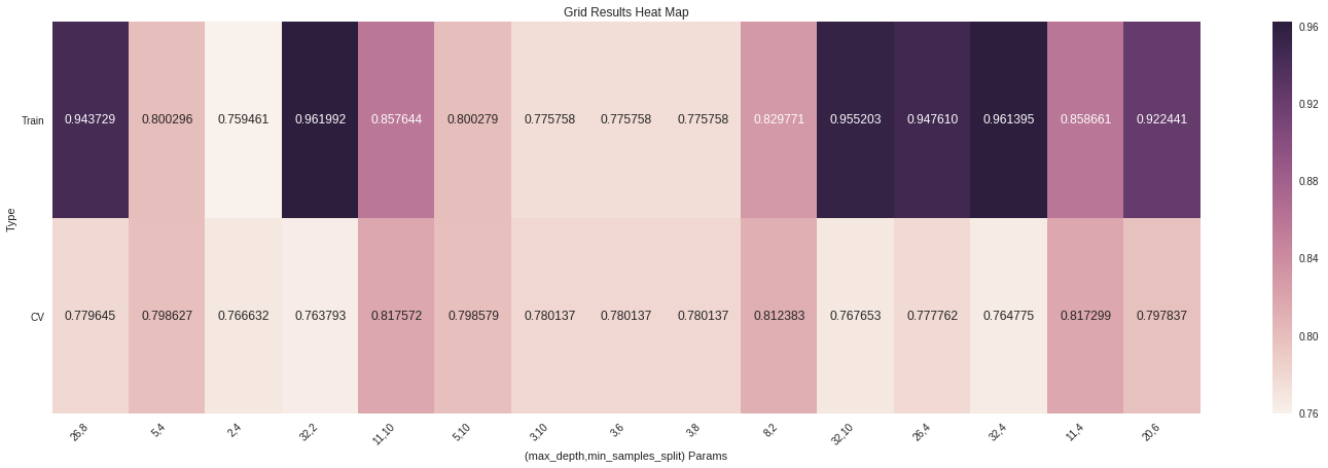
In [0]:

```
best_param_search(training_data,y_train,param_grid,tss)
```

```
Fitting 10 folds for each of 15 candidates, totalling 150 fits
```

```
[Parallel(n_jobs=-1)]: Using backend LokyBackend with 2 concurrent workers.
[Parallel(n_jobs=-1)]: Done  46 tasks      | elapsed: 19.4min
[Parallel(n_jobs=-1)]: Done 150 out of 150 | elapsed: 58.8min finished
```

Out[0]:



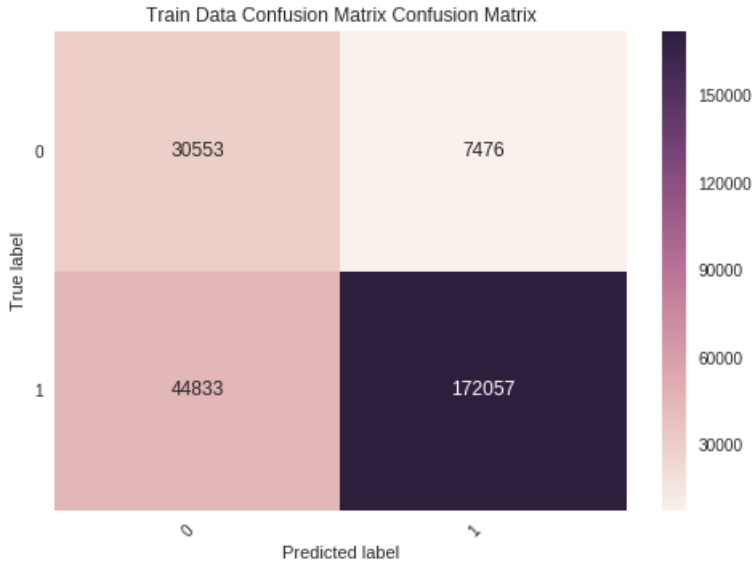
On Test Data

In [21]:

```
rev_lens_test = x_test.rev_len.values.reshape(-1,1)
sum_pos_score_test = x_test.sum_pos_score.values.reshape(-1,1)
test_data = sparse.hstack((TFIDF_test, rev_lens_test,sum_pos_score_test))
dtc = on_test(training_data,y_train,test_data,y_test,11,10,feat_names)
```

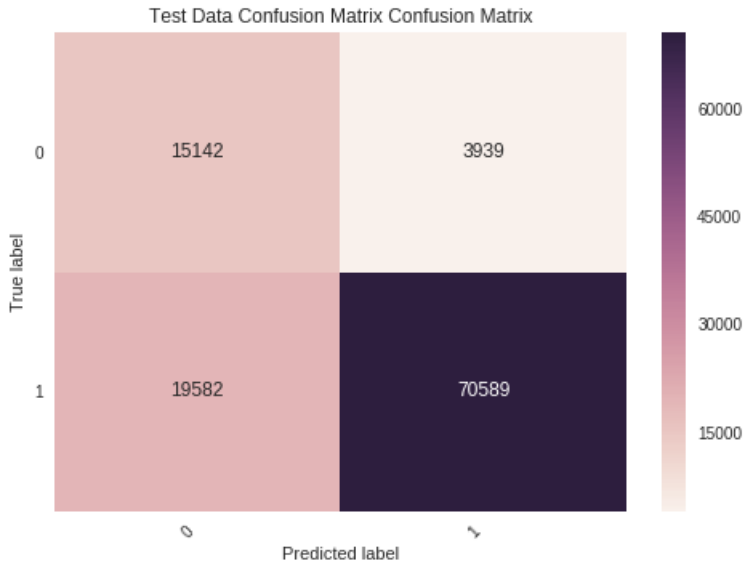
```
The ROC AUC score for the params max_depth 11  and min_samples_split  10 is  0.8381241847129981
```

Out[21]:




```
The TPR is :  0.7932915302687998
The TNR is :  0.8034131846748535
The FPR is :  0.1965868153251466
The FNR is :  0.20670846973120013
```

Out[21]:



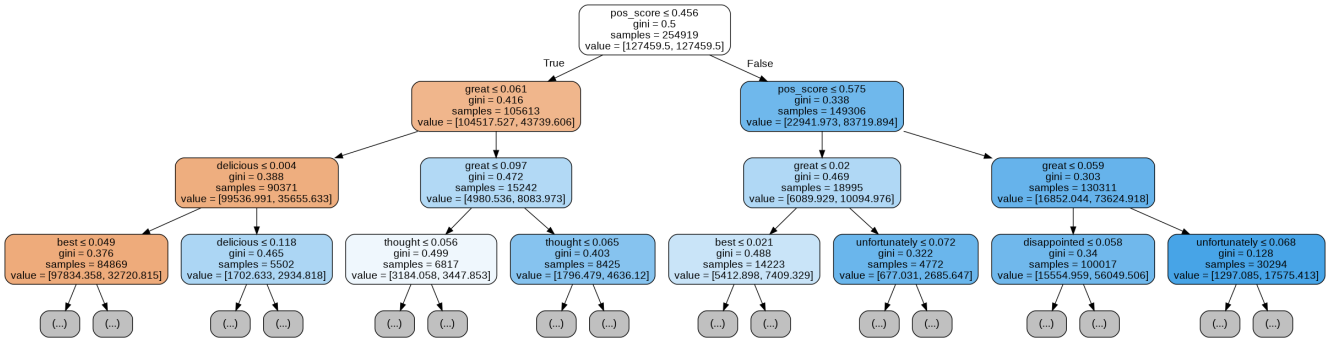

```
The TPR is :  0.782834836033758
The TNR is :  0.7935642786017504
The FPR is :  0.20643572139824956
The FNR is :  0.21716516396624191
```

Tree viz

In [22]:

```
genGraph(dtc,feat_names)
```

Out[22]:



Important Features WordCloud

In [0]:

```
top_200 = np.argsort(dtc.feature_importances_)[-200:].tolist()
vals  = [feat_names[i] for i in top_200]
Wordcl('TFIDF Important Features',vals)
```

Out[0]:

Average W2V

In [0]:

```
import re
def cleanhtml(sentence):
    cleanr = re.compile('<.*?>')
    cleantext = re.sub(cleanr, ' ', sentence)
    return cleantext
def cleanpunc(sentence):
    cleaned = re.sub(r'[?|!|\'|"|#]',r'',sentence)
    cleaned = re.sub(r'[.|,|)|(|\|/]',r' ',cleaned)
    return  cleaned


i=0
list_of_sent=[]
for sent in x_train.Cleaned.values:
    filtered_sentence=[]
    sent=cleanhtml(sent)
    for w in sent.split():
        for cleaned_words in cleanpunc(w).split():
            if(cleaned_words.isalpha()):
                filtered_sentence.append(cleaned_words.lower())
            else:
                continue
    list_of_sent.append(filtered_sentence)
```

In [0]:

```
!pip install gensim
import gensim
from gensim.models import Word2Vec
from gensim.models import KeyedVectors
import pickle
import numpy as np

w2v_model=gensim.models.Word2Vec(list_of_sent,min_count=5,size=50, workers=8)
```

In [0]:

```
sent_vectors_train = []
for sent in x_train.Cleaned.values:
    sent_vec = np.zeros(50)
    cnt_words =0
    for word in sent.split():
        try:
            vec = w2v_model.wv[word]
            sent_vec += vec
            cnt_words += 1
        except:
            pass
    sent_vec /= cnt_words
    sent_vectors_train.append(sent_vec)

sent_vectors_train = np.nan_to_num(sent_vectors_train)
```

In [0]:

```
rev_lens_train = x_train.rev_len.values.reshape(-1,1)
sum_pos_score_train = x_train.sum_pos_score.values.reshape(-1,1)
training_data = np.hstack((sent_vectors_train, rev_lens_train,sum_pos_score_train))
feat_names = []
feat = "F1-F50"
l = feat[1:].split('-')
feat_names = ['F'+str(x) for x in range(int(l[0]), int(l[1][1:])+1)]
```

```
feat_names.append('review_length')
feat_names.append('pos_score')
```

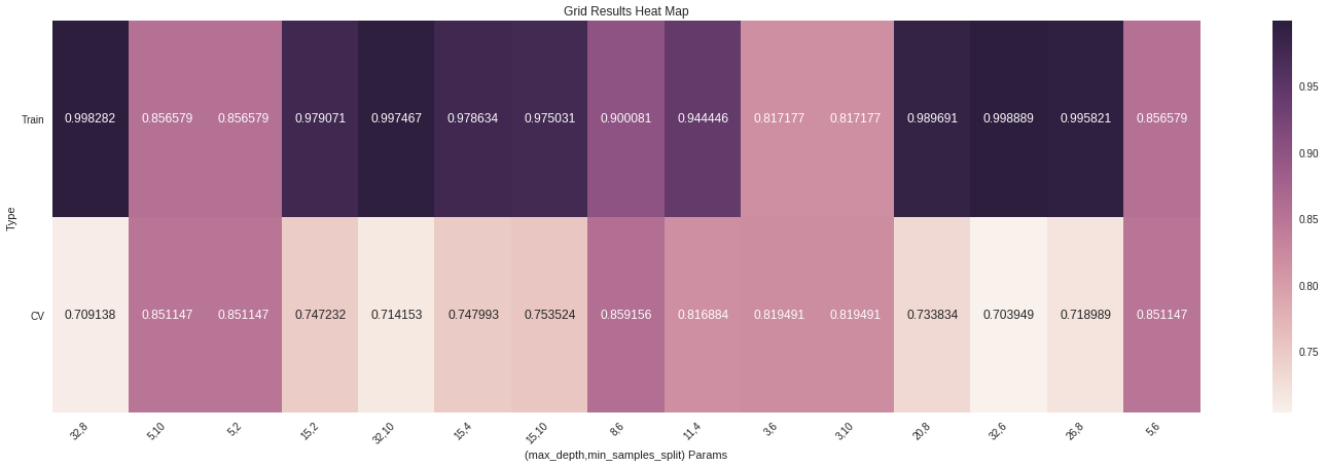
In [0]:

```
best_param_search(training_data,y_train,param_grid,tss)
```

```
Fitting 10 folds for each of 15 candidates, totalling 150 fits
```

```
[Parallel(n_jobs=-1)]: Using backend LokyBackend with 2 concurrent workers.
[Parallel(n_jobs=-1)]: Done  46 tasks      | elapsed:  5.8min
[Parallel(n_jobs=-1)]: Done 150 out of 150 | elapsed: 19.4min finished
```

Out[0]:



In [0]:

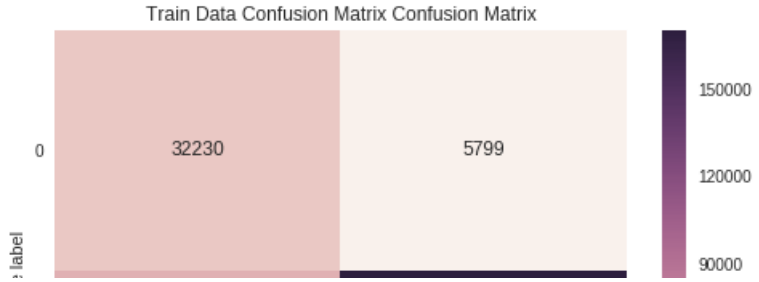
```
sent_vectors_test = []
for sent in x_test.Cleaned.values:
    sent_vec = np.zeros(50)
    cnt_words =0
    for word in sent.split():
        try:
            vec = w2v_model.wv[word]
            sent_vec += vec
            cnt_words += 1
        except:
            pass
    sent_vec /= cnt_words
    sent_vectors_test.append(sent_vec)
sent_vectors_test = np.nan_to_num(sent_vectors_test)
```

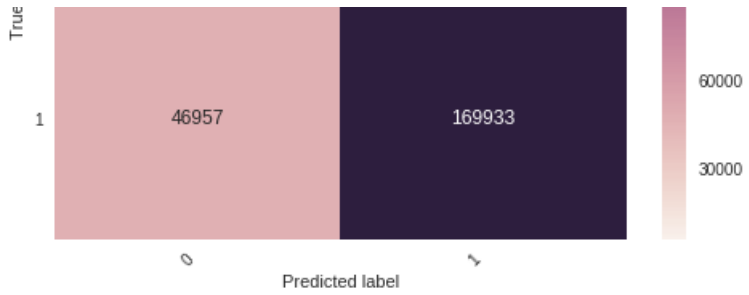
In [0]:

```
rev_lens_test = x_test.rev_len.values.reshape(-1,1)
sum_pos_score_test = x_test.sum_pos_score.values.reshape(-1,1)
test_data = np.hstack((sent_vectors_test, rev_lens_test,sum_pos_score_test))
dtc = on_test(training_data,y_train,test_data,y_test,8,6,feat_names)
```

```
The ROC AUC score for the params max_depth 8  and min_samples_split  6 is  0.8781650052899188
```

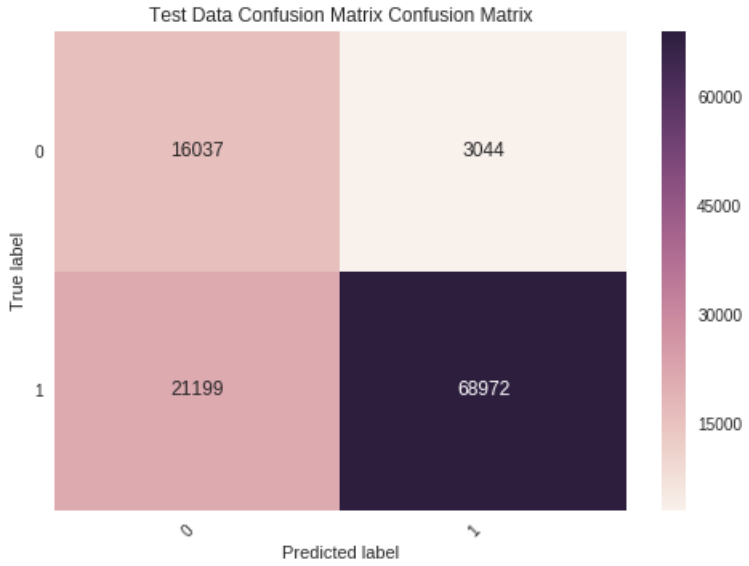
Out[0]:






```
The TPR is :  0.7834985476508829
The TNR is :  0.8475111099424124
The FPR is :  0.15248889005758762
The FNR is :  0.21650145234911705
```

Out[0]:




```
The TPR is :  0.7649022412970911
The TNR is :  0.8404695770661915
The FPR is :  0.1595304229338085
The FNR is :  0.23509775870290892
```

TFIDF W2V

In [0]:

```
!kaggle datasets download -d sanjeev5/w2vtfidf-train
!kaggle datasets download -d sanjeev5/w2vtfidftest
!unzip w2vtfidf-train.zip
!unzip w2vtfidftest.zip
```

In [0]:

```
import pickle
tfidf_w2v_train = pickle.load( open( "AgTFIDF_Train.txt", "rb" ) )
tfidf_w2v_test = pickle.load( open( "test_AgTFIDF.txt", "rb" ) )
```

In [0]:

```
rev_lens_train = x_train.rev_len.values.reshape(-1,1)
sum_pos_score_train = x_train.sum_pos_score.values.reshape(-1,1)
training_data = np.hstack((tfidf_w2v_train, rev_lens_train,sum_pos_score_train))
feat_names = []
feat = "F1-F50"
l = feat[1:].split('-')
```

```
feat_names = ['F'+str(x) for x in range(int(l[0]), int(l[1][1:])+1)]
feat_names.append('review_length')
feat_names.append('pos_score')
```

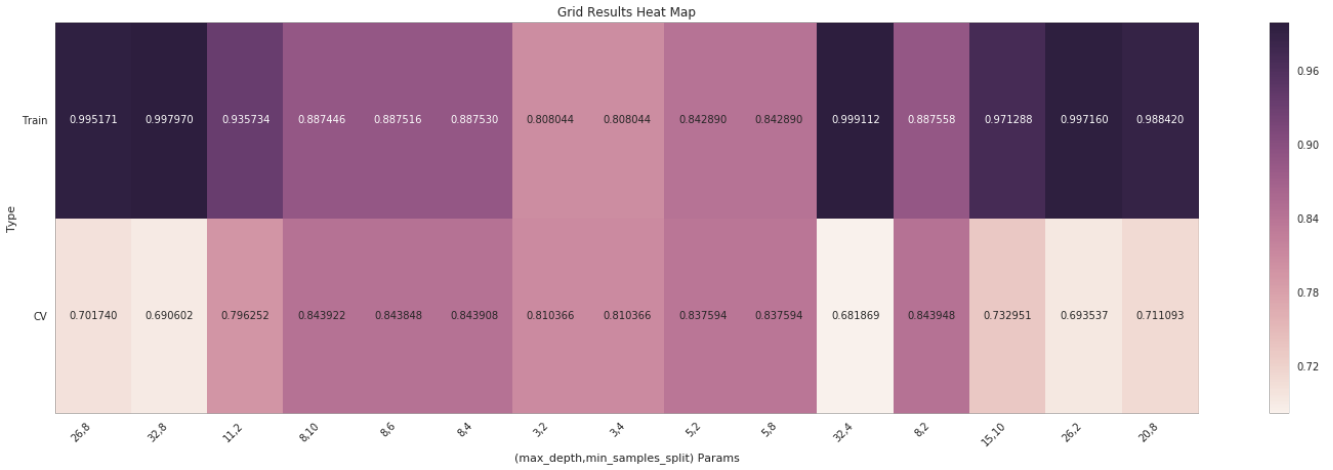
In [0]:

```
best_param_search(training_data,y_train,param_grid,tss)
```

```
Fitting 10 folds for each of 15 candidates, totalling 150 fits
```

```
[Parallel(n_jobs=-1)]: Done  34 tasks      | elapsed:  1.5min
[Parallel(n_jobs=-1)]: Done 150 out of 150 | elapsed:  5.4min finished
```

Out[0]:



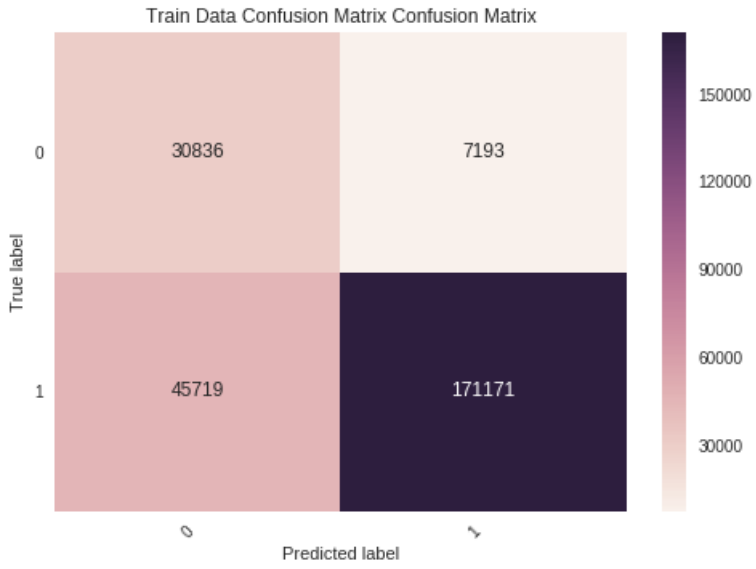
On Test Data

In [0]:

```
tfidf_w2v_test = np.nan_to_num(tfidf_w2v_test)
rev_lens_test = x_test.rev_len.values.reshape(-1,1)
sum_pos_score_test = x_test.sum_pos_score.values.reshape(-1,1)
test_data = np.hstack((tfidf_w2v_test, rev_lens_test,sum_pos_score_test))
dtc = on_test(training_data,y_train,test_data,y_test,8,2,feat_names)
```

```
The ROC AUC score for the params max_depth 8  and min_samples_split  2 is  0.7074896849535952
```

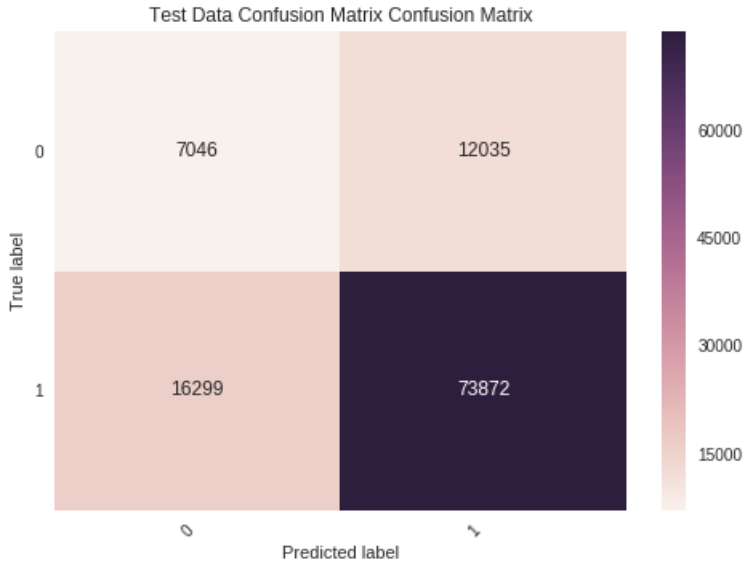
Out[0]:




```
The TPR is :  0.7892065102125502
```

```
The TNR is :  0.8108548739120145
The FPR is :  0.18914512608798548
The FNR is :  0.21079348978744986
```

Out[0]:




```
The TPR is :  0.8192434374688092
The TNR is :  0.36926785807871704
The FPR is :  0.6307321419212829
The FNR is :  0.18075656253119074
```

Conclusion

Model	Max Depth	Min Sample Split	Train FPR	Test FPR	Train FNR	Test FNR
Bag Of Words	11	4	0.20	0.206	0.204	0.215
TFIDF	11	10	0.20	0.207	0.206	0.217
Avg W2V	8	6	0.152	0.160	0.217	0.235
TF-IDF W2V	8	2	0.19	0.63	0.210	0.180

Average W2V has performed very well in reducing the FPR