Random Forest On Amazon Fine Food Reviews Analysis

Data Source: https://www.kaggle.com/snap/amazon-fine-food-reviews

The Amazon Fine Food Reviews dataset consists of reviews of fine foods from Amazon.

Number of reviews: 568,454
Number of users: 256,059
Number of products: 74,258
Timespan: Oct 1999 - Oct 2012
Number of Attributes/Columns in data: 10

Attribute Information:

1. Id
2. ProductId - unique identifier for the product
3. UserId - unqiue identifier for the user
4. ProfileName
5. HelpfulnessNumerator - number of users who found the review helpful
6. HelpfulnessDenominator - number of users who indicated whether they found the review helpful or not
7. Score - rating between 1 and 5
8. Time - timestamp for the review
9. Summary - brief summary of the review
10. Text - text of the review

Objective:

Given a review, determine whether the review is positive (rating of 4 or 5) or negative (rating of 1 or 2).

[Q] How to determine if a review is positive or negative?

[Ans] We could use Score/Rating. A rating of 4 or 5 can be cosnidered as a positive review. A rating of 1 or 2 can be considered as negative one. A review of rating 3 is considered nuetral and such reviews are ignored from our analysis. This is an approximate and proxy way of determining the polarity (positivity/negativity) of a review.

[1]. Reading Data

[1.1] Loading the data

The dataset is available in two forms

1. .csv file
2. SQLite Database

Here as we only want to get the global sentiment of the recommendations (positive or negative), we will purposefully ignore all Scores equal to 3. If the score is above 3, then the recommendation wil be set to "positive". Otherwise, it will be set to "negative".

In [0]:

```
!pip install tqdm
import warnings
warnings.filterwarnings("ignore")
import sqlite3
import pandas as pd
import numpy as np
import nltk
import string
import matplotlib.pyplot as plt
import seaborn as sns
from sklearn.metrics import confusion_matrix,roc_curve,auc,roc_auc_score
from sklearn.model_selection import train_test_split,RandomizedSearchCV
!pip install vaderSentiment
from vaderSentiment.vaderSentiment import SentimentIntensityAnalyzer
import re
import pickle
from tqdm import tqdm
import os
from bs4 import BeautifulSoup
from sklearn.ensemble import RandomForestClassifier
from scipy import sparse
import random
%env JOBLIB_TEMP_FOLDER=/tmp
```

```
Requirement already satisfied: tqdm in /usr/local/envs/py3env/lib/python3.5/site-packages (4.29.0)
Requirement already satisfied: vaderSentiment in /usr/local/envs/py3env/lib/python3.5/site-
packages (3.2.1)
env: JOBLIB_TEMP_FOLDER=/tmp
```

In [0]:

```
!pip install wordcloud
from wordcloud import WordCloud
```

```
Requirement already satisfied: wordcloud in /usr/local/envs/py3env/lib/python3.5/site-packages
(1.5.0)
Requirement already satisfied: pillow in /usr/local/envs/py3env/lib/python3.5/site-packages (from
wordcloud) (3.4.1)
Requirement already satisfied: numpy>=1.6.1 in /usr/local/envs/py3env/lib/python3.5/site-packages
(from wordcloud) (1.14.0)
```

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')
```

[2] Exploratory Data Analysis

[2.1] Data Cleaning: Deduplication

It is observed (as shown in the table below) that the reviews data had many duplicate entries. Hence it was necessary to remove duplicates in order to get unbiased results for the analysis of the data. Following is an example:

In [0]:

```
# Removing all the neutral reviews
input_data = input_data[input_data.Score != 3]
# Sorting the data with respect to the ProductID
sorted_data=input_data.sort_values('ProductId', axis=0, ascending=True, inplace=False, kind='quicksort', na_position='last')
# Removing the duplicates of UserId,ProfileName,Time,Text
input_data = sorted_data.drop_duplicates(subset={"UserId","ProfileName","Time","Text"}, keep='first', inplace=False)
# Removing all the rows having the usefulness more or equal than the total number of votes
input_data = input_data[input_data.HelpfulnessNumerator<=input_data.HelpfulnessDenominator]
# Sorting the data by time in ascending order
data = input_data.iloc[input_data.Time.argsort()]
# Assigning score value 1 for reviews > 4 else 0 i.e 1 represents 'Positive' and 0 represents 'Negative' Review
data.Score = data.Score.map(lambda x : 1 if (x > 3) else 0)
```

In [0]:

```
data.head(5)
```

Out[0]:

	Id	ProductId	UserId	ProfileName	HelpfulnessNumerator	HelpfulnessDenominator	Score	Time	Su
150523	150524	0006641040	ACITT7DI6IDDL	shari zychinski	0	0	1	939340800	EVEI edu
150500	150501	0006641040	AJ46FKXOVC7NR	Nicholas A Mesiano	2	2	1	940809600	Th grea spe
451855	451856	B00004CXX9	AIUWLEQ1ADEG5	Elizabeth Medina	0	0	1	944092800	Ente
374358	374359	B00004CI84	A344SMIA5JECGM	Vincent P. Ross	1	2	1	944438400	A day t
451854	451855	B00004CXX9	AJH6LUC1UT1ON	The Phantom of the Opera	0	0	1	946857600	FAN'

In [0]:

```
data.Score.value_counts()
```

Out[0]:

```
1    307061
0     57110
Name: Score, dtype: int64
```

[3] Preprocessing

[3.1]. Preprocessing Review Text

Now that we have finished deduplication our data requires some preprocessing before we go on further with analysis and making the prediction model.

Hence in the Preprocessing phase we do the following in the order below:-

1. Begin by removing the html tags
2. Remove any punctuations or limited set of special characters like , or . or # etc.
3. Check if the word is made up of english letters and is not alpha-numeric
4. Check to see if the length of the word is greater than 2 (as it was researched that there is no adjective in 2-letters)
5. Convert the word to lowercase
6. Remove Stopwords

After which we collect the words used to describe positive and negative reviews

Methods For Pre-Processing

In [0]:

```
def remove_urls(sentence):
  """A method to remove URLS from the text data
     Credits to : https://stackoverflow.com/a/40823105/4084039
  """
  return re.sub(r"http\S+", "", sentence)

def remove_tags(sentence):
  """A method to remove tag elemts from the text data
     Credits to : https://stackoverflow.com/questions/16206380/python-beautifulsoup-how-to-remove-
all-tags-from-an-element
  """
  return BeautifulSoup(sentence, 'lxml').get_text()
  return BeautifulSoup(sentence, 'lxml').get_text()

def deconstructing_sentence(sentence):
  """ A method to expand certain words like can't to cannot
      Credits to : https://stackoverflow.com/a/47091490/4084039
  """
  # specific
  phrase = re.sub(r"won't", "will not", sentence)
  phrase = re.sub(r"can\'t", "can not", phrase)
  # general
  phrase = re.sub(r"n\'t", " not", phrase)
  phrase = re.sub(r"\'re", " are", phrase)
  phrase = re.sub(r"\'s", " is", phrase)
  phrase = re.sub(r"\'d", " would", phrase)
  phrase = re.sub(r"\'ll", " will", phrase)
  phrase = re.sub(r"\'t", " not", phrase)
  phrase = re.sub(r"\'ve", " have", phrase)
  phrase = re.sub(r"\'m", " am", phrase)
  return phrase

def remove_words_with_numbers(sentence):
  """ A method to remove words with numbers
      Credits to : https://stackoverflow.com/a/18082370/4084039
  """
  return re.sub("\S*\d\S*", "", sentence).strip()

def remove_special_chars(sentence):
  """ A method to remove special characters in text sentence/data
      Credits to : https://stackoverflow.com/a/18082370/4084039
  """
  return re.sub('[^A-Za-z0-9]+', ' ', sentence)

# https://gist.github.com/sebleier/554280
# we are removing the words from the stop words list: 'no', 'nor', 'not'
# <br /><br /> ==> after the above steps, we are getting "br br"
# we are including them into stop words list
# instead of <br /> if we have <br/> these tags would have revmoved in the 1st step

stopwords= set(['br', 'the', 'i', 'me', 'my', 'myself', 'we', 'our', 'ours', 'ourselves', 'you', "y
ou're", "you've",\
            "you'll", "you'd", 'your', 'yours', 'yourself', 'yourselves', 'he', 'him', 'his',
'himself', \
            'she', "she's", 'her', 'hers', 'herself', 'it', "it's", 'its', 'itself', 'they', 'them',
'their',\
            'theirs', 'themselves', 'what', 'which', 'who', 'whom', 'this', 'that', "that'll",
'these', 'those', \
            'am', 'is', 'are', 'was', 'were', 'be', 'been', 'being', 'have', 'has', 'had', 'having',
'do', 'does', \
            'did', 'doing', 'a', 'an', 'the', 'and', 'but', 'if', 'or', 'because', 'as', 'until', '
while', 'of', \
            'at', 'by', 'for', 'with', 'about', 'against', 'between', 'into', 'through', 'during',
'before', 'after',\
            'above', 'below', 'to', 'from', 'up', 'down', 'in', 'out', 'on', 'off', 'over', 'under'
, 'again', 'further',\
            'then', 'once', 'here', 'there', 'when', 'where', 'why', 'how', 'all', 'any', 'both', 'e
ach', 'few', 'more',\
            'most', 'other', 'some', 'such', 'only', 'own', 'same', 'so', 'than', 'too', 'very', \
            's', 't', 'can', 'will', 'just', 'don', "don't", 'should', "should've", 'now', 'd', 'll'
, 'm', 'o', 're', \
            've', 'y', 'ain', 'aren', "aren't", 'couldn', "couldn't", 'didn', "didn't", 'doesn', "do
esn't", 'hadn',\
            "hadn't", 'hasn', "hasn't", 'haven', "haven't", 'isn', "isn't", 'ma', 'mightn',
"mightn't", 'mustn',\
            "mustn't", 'needn', "needn't", 'shan', "shan't", 'shouldn', "shouldn't", 'wasn',
"wasn't", 'weren', "weren't", \
            'won', "won't", 'wouldn', "wouldn't"])


def clean_sentence(list_of_sent):
  preprocessed_text = []
  for i in tqdm(list_of_sent):
    sent = remove_urls(i)
    sent = remove_tags(sent)
    sent = deconstructing_sentence(sent)
    sent = remove_words_with_numbers(sent)
    sent = remove_special_chars(sent)
    sentence = ' '.join(e.lower() for e in sent.split() if e.lower() not in stopwords)
    preprocessed_text.append(sentence.strip())
    preprocessed_text.append(sentence.strip())
  return preprocessed_text

# 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'])
```

In [0]:

```
data['Cleaned'] = clean_sentence(data.Text.values)
```

```
100%|██████████| 364171/364171 [02:45<00:00, 2193.92it/s]
```

[3.2] Preprocessing Review Summary

In [0]:

```
data['Cleaned_Summary'] = clean_sentence(data.Summary.astype(str).values)
```

```
100%|██████████| 364171/364171 [01:49<00:00, 3311.82it/s]
```

[4] Featurization

In [0]:

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


data['sum_pos_score'] = [vaderAnalysis(i) for i in data.Cleaned_Summary.values]
```

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)
```

In [0]:

```
#x represents n_estimators y represents max_depth
# Credits to : https://stackoverflow.com/questions/1262955/how-do-i-pick-2-random-items-from-a-pyt
hon-set
params_combo = set((x,y) for x in range(10,210,10) for y in range(10,51,5))
```

[4.1] BAG OF WORDS

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)
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')
```

```
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))
```

[4.3] TF-IDF

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')

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))
```

[4.4] Word2Vec

In [0]:

```
# Train your own Word2Vec model using your own text corpus
list_of_sentence=[]
for sent in tqdm(x_train.Cleaned):
    list_of_sentence.append(sent.split())
```

```
100%|██████████| 254919/254919 [00:03<00:00, 74339.80it/s]
```

In [0]:

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

w2v_model=gensim.models.Word2Vec(list_of_sentence,min_count=5,size=50, workers=8)
w2v_words = list(w2v_model.wv.vocab)
```

[4.4.1] Converting text into vectors using Avg W2V, TFIDF-W2V

[4.4.1.1] Avg W2v

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_vec /= cnt_words
    sent_vectors_train.append(sent_vec)

sent_vectors_train = np.nan_to_num(sent_vectors_train)

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))
```

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)

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))
```

[4.4.1.2] TFIDF weighted W2v

In [0]:

```
!kaggle datasets download -d sanjeev5/tfidfw2vtrain
!unzip tfidfw2vtrain.zip
tfidf_w2v_train = pickle.load( open( "Tfidf_W2V_Train.txt", "rb" ) )
!kaggle datasets download -d sanjeev5/tfidfw2vtest
!unzip tfidfw2vtest.zip
tfidf_w2v_test = pickle.load( open( "Tfidf_W2V_Test.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))

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))
```

[5] Assignment 9: Random Forests

1. **Apply Random Forests & GBDT on these feature sets**

    - SET 1:Review text, preprocessed one converted into vectors using (BOW)
    - SET 2:Review text, preprocessed one converted into vectors using (TFIDF)
    - SET 3:Review text, preprocessed one converted into vectors using (AVG W2v)
    - SET 4:Review text, preprocessed one converted into vectors using (TFIDF W2v)

2. **The hyper paramter tuning (Consider any two hyper parameters)**

    - Find the best hyper parameter which will give the maximum AUC value
    - Find the best hyper paramter using k-fold cross validation or simple cross validation data
    - Use gridsearch cv or randomsearch cv or you can also write your own for loops to do this task of hyperparameter tuning

3. **Feature importance**

- Get top 20 important features and represent them in a word cloud. Do this for BOW & TFIDF.

4. **Feature engineering**

- To increase the performance of your model, you can also experiment with with feature engineering like :
  - Taking length of reviews as another feature.
  - Considering some features from review summary as well.

5. **Representation of results**

- You need to plot the performance of model both on train data and cross validation data for each hyper parameter, like shown in the figure

  with X-axis as **n_estimators**, Y-axis as **max_depth**, and Z-axis as **AUC Score** , we have given the notebook which explains how to plot this 3d plot, you can find it in the same drive *3d_scatter_plot.ipynb*

(or)

- You need to plot the performance of model both on train data and cross validation data for each hyper parameter, like shown in the figure

  seaborn heat maps with rows as **n_estimators**, columns as **max_depth**, and values inside the cell representing **AUC Score**
- You choose either of the plotting techniques out of 3d plot or heat map
- Once after you found the best hyper parameter, you need to train your model with it, and find the AUC on test data and plot the ROC curve on both train and test.
- Along with plotting ROC curve, you need to print the confusion matrix with predicted and original labels of test data points. Please visualize your confusion matrices using seaborn heatmaps.

6. **Conclusion**

- You need to summarize the results at the end of the notebook, summarize it in the table format. To print out a table please refer to this prettytable library link

Note: Data Leakage

1. There will be an issue of data-leakage if you vectorize the entire data and then split it into train/cv/test.
2. To avoid the issue of data-leakag, make sure to split your data first and then vectorize it.
3. While vectorizing your data, apply the method fit_transform() on you train data, and apply the method transform() on cv/test data.
4. For more details please go through this link.

[5.1] Applying RF

In [0]:

```
#OOB Scoring method Credits to :
https://datascience.stackexchange.com/questions/13151/randomforestclassifier-oob-scoring-method

def best_param_search(train_data,train_label,params):
  """Method to find the best hyper params for the RFC"""
  train_score = []
  oob_score = []
  par_cols = []
  for i,j in params:
    rfc = RandomForestClassifier(criterion = 'gini',class_weight = 'balanced_subsample',random_stat
e = 0,n_estimators = i,max_depth= j,n_jobs = -1,oob_score = True)
    rfc.fit(train_data,train_label)
    train_score.append(roc_auc_score(train_label,rfc.predict_proba(train_data)[:,1]))
    pred_train = np.argmax(rfc.oob_decision_function_,axis=1)
    oob_score.append(roc_auc_score(train_label, pred_train))
    par_cols.append(str(i)+','+str(j))
  df_cm = pd.DataFrame(data = [train_score,oob_score],index=['Train','OOB'], columns=par_cols)
  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('(n_estimators,max_depth) Params')
  plt.title('OOB Results Heat Map')
  plt.show()

# Confusion Matrix
def confusion_matrix_display(conf_mtrx,tst_labels,Title):
  """To print the confusion matrix
     Reused from the previous assignments
  """
  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,n_estimators):
  """RF on test data"""
  rfc = RandomForestClassifier(criterion = 'gini',class_weight = 'balanced_subsample',random_state
= 0,n_estimators = n_estimators,max_depth= md,n_jobs = -1,oob_score = True)
  rfc.fit(train_data,train_label)
  print('The ROC AUC score for the params max_depth' ,md, ' and n_estimators ',n_estimators, 'is '
, roc_auc_score(test_label,rfc.predict_proba(test_data)[:,1]))
  confusion_matrix_display(confusion_matrix(train_label,rfc.predict(train_data)),train_label,'Train
Data Confusion Matrix')
  confusion_matrix_display(confusion_matrix(test_label,rfc.predict(test_data)),test_label,'Test
Data Confusion Matrix')
  roc_curve_draw(train_label,test_label,rfc.predict_proba(train_data)[:,1],rfc.predict_proba(test_d
ata)[:,1])
  return rfc

def Wordcl(title,val):
  """To print the wordcloud of top important features"""
  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()

def roc_curve_draw(y_train,y_test,train_predict_proba,test_predict_proba):
    """Method to draw the roc curve for the train and the test date
    This code was reused from KNN assignment
    """
    fpr_train, tpr_train, thresholds_train = roc_curve(y_train,train_predict_proba)
    auc_train =  auc(fpr_train, tpr_train)
    print('\n The auc of train is : ',auc_train)
    fpr_test, tpr_test, thresholds_test = roc_curve(y_test,test_predict_proba)
    auc_test =  auc(fpr_test, tpr_test)
    print('\n The auc of train is : ',auc_test)
    plt.title('Receiver Operating Characteristic')
    plt.plot(fpr_train, tpr_train, 'b', label = 'Train AUC = %0.2f' % auc_train)
    plt.plot(fpr_test, tpr_test, 'g', label = 'Test AUC = %0.2f' % auc_test)
    plt.legend(loc = 'lower right')
    plt.plot([0, 1], [0, 1],'r--')
    plt.xlim([0, 1])
    plt.ylim([0, 1])
    plt.ylabel('True Positive Rate')
    plt.xlabel('False Positive Rate')
    plt.show()
```

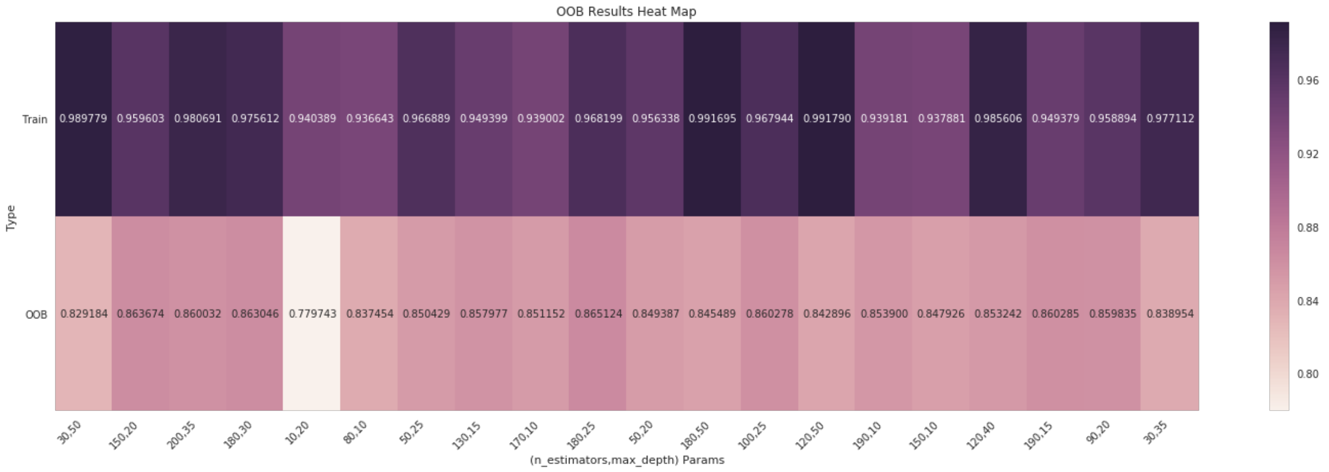
[5.1.1] Applying Random Forests on BOW, SET 1

[5.1.1] Applying Random Forests on BOW, SET 1

In [0]:

```
random_params = random.sample(params_combo,20)
best_param_search(training_data,y_train,random_params)
```

Out[0]:

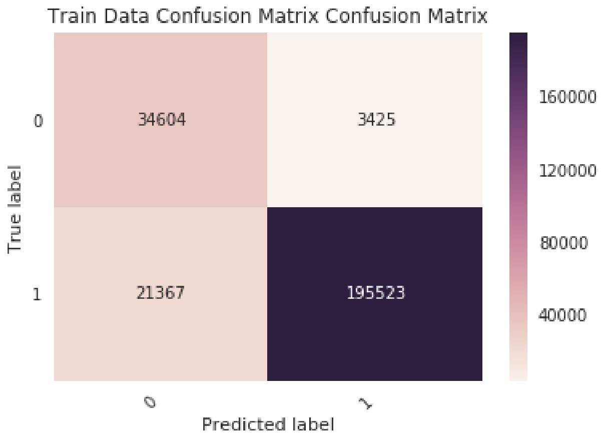


In [0]:

```
rfc = on_test(training_data,y_train,test_data,y_test,25,180)
```

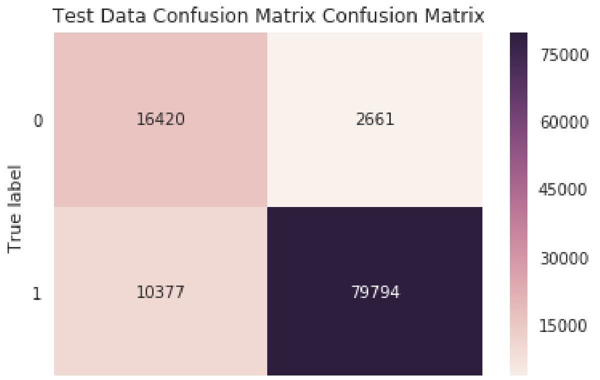
```
The ROC AUC score for the params max_depth 25  and n_estimators  180 is  0.9457652367692366
```

Out[0]:



```
The TPR is :  0.9014846235418876
The TNR is :  0.9099371532251702
The FPR is :  0.09006284677482973
The FNR is :  0.09851537645811241
```

Out[0]:

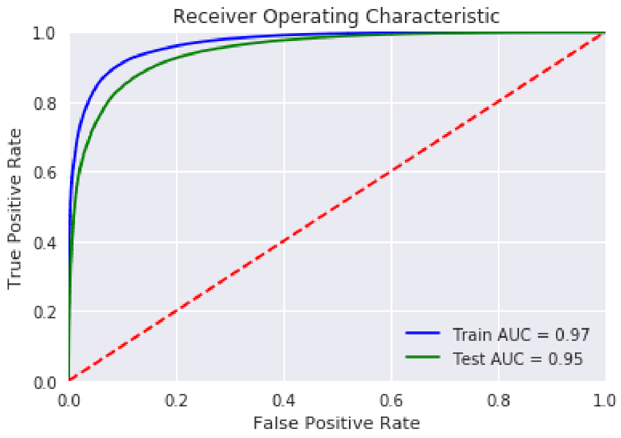


```
The TPR is :  0.884918654556343
The TNR is :  0.8605419003196897
The FPR is :  0.13945809968031025
The FNR is :  0.11508134544365706


 The auc of train is :  0.9681987630448388

 The auc of train is :  0.9457652367692366
```

Out[0]:

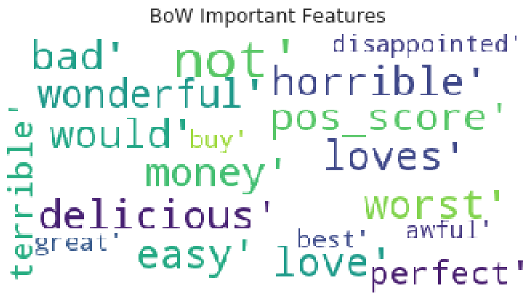


[5.1.2] Wordcloud of top 20 important features from SET 1

In [0]:

```
top_20 = np.argsort(rfc.feature_importances_)[-20:].tolist()
vals  = [feat_names[i] for i in top_20]
Wordcl('BoW Important Features',vals)
```

Out[0]:

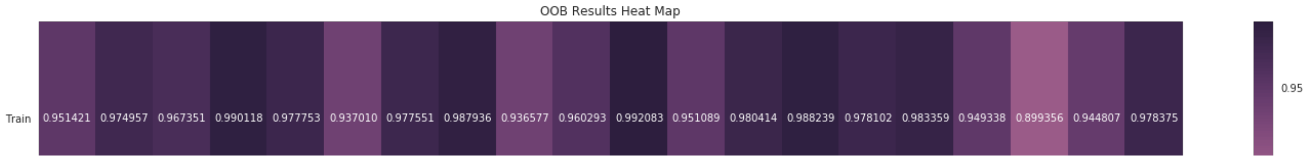


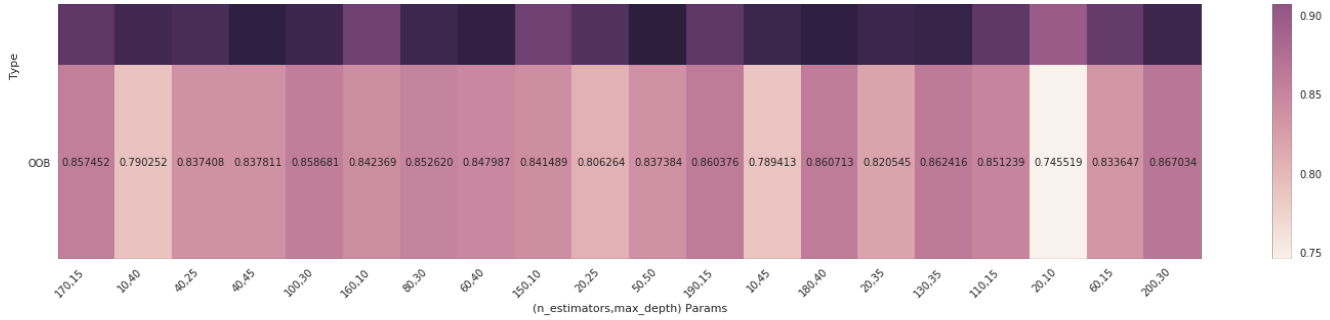
[5.1.3] Applying Random Forests on TFIDF, SET 2

In [0]:

```
random_params = random.sample(params_combo,20)
best_param_search(training_data,y_train,random_params)
```

Out[0]:



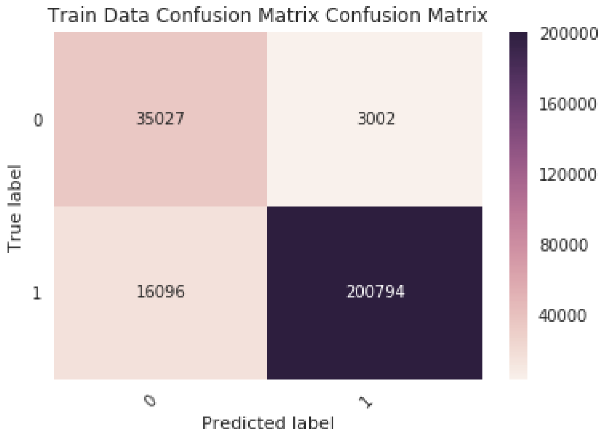


In [0]:

```
rfc = on_test(training_data,y_train,test_data,y_test,30,200)
```

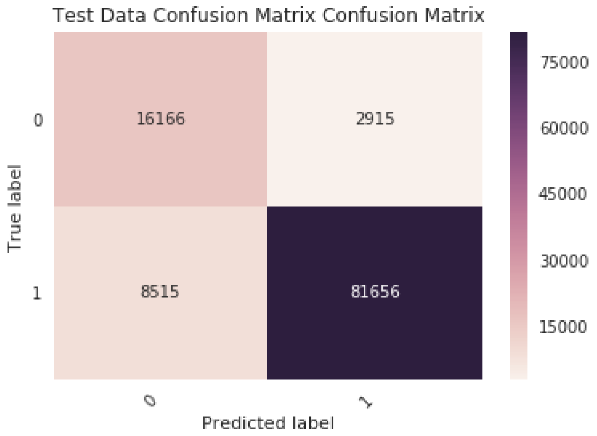
```
The ROC AUC score for the params max_depth 30  and n_estimators  200 is  0.9503323190855009
```

Out[0]:



```
The TPR is :  0.9257872654340911
The TNR is :  0.9210602434983828
The FPR is :  0.07893975650161719
The FNR is :  0.07421273456590899
```

Out[0]:

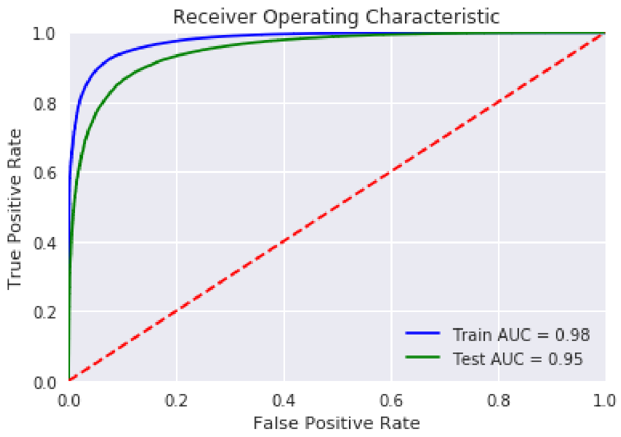


```
The TPR is :  0.9055683091015958
The TNR is :  0.8472302290236361
The FPR is :  0.15276977097636393
The FNR is :  0.09443169089840414


 The auc of train is :  0.9783747162551417

 The auc of train is :  0.9503323190855009
```

Out[0]:

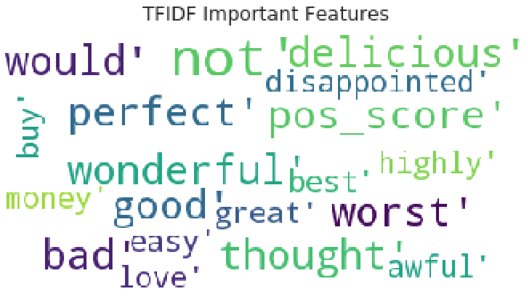


[5.1.4] Wordcloud of top 20 important features from SET 2

In [0]:

```
top_20 = np.argsort(rfc.feature_importances_)[-20:].tolist()
vals  = [feat_names[i] for i in top_20]
Wordcl('TFIDF Important Features',vals)
```

Out[0]:

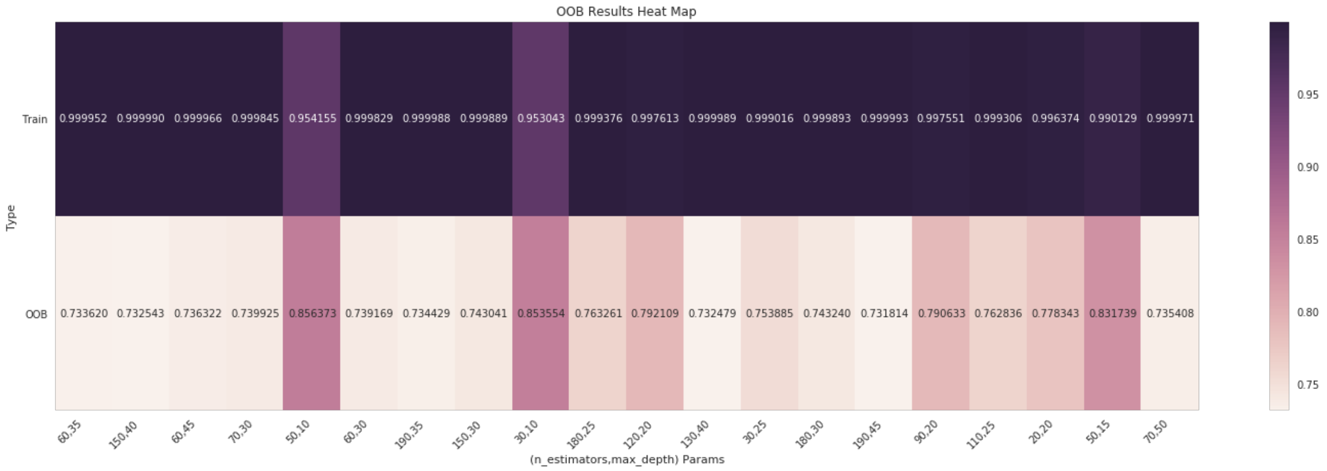


[5.1.5] Applying Random Forests on AVG W2V, SET 3

In [0]:

```
random_params = random.sample(params_combo,20)
best_param_search(training_data,y_train,random_params)
```

Out[0]:



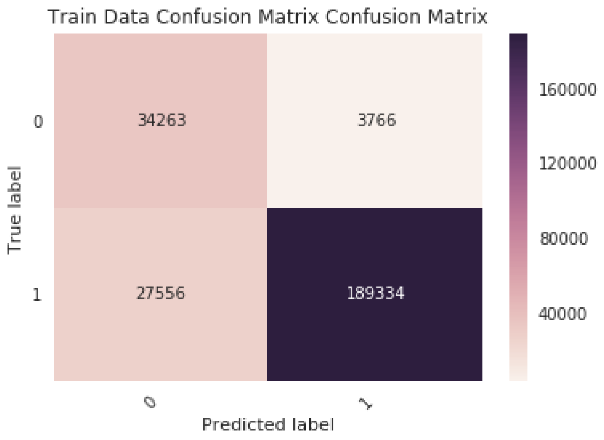
In [0]:

In [0]:

```
rfc = on_test(training_data,y_train,test_data,y_test,10,50)
```

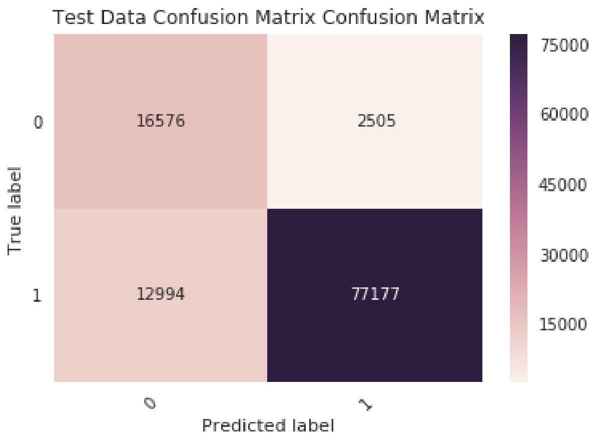
```
The ROC AUC score for the params max_depth 10  and n_estimators  50 is  0.9334759086688468
```

Out[0]:


Train Data Confusion Matrix Confusion Matrix

```
The TPR is :  0.8729494213656692
The TNR is :  0.9009703121302164
The FPR is :  0.09902968786978358
The FNR is :  0.12705057863433078
```

Out[0]:

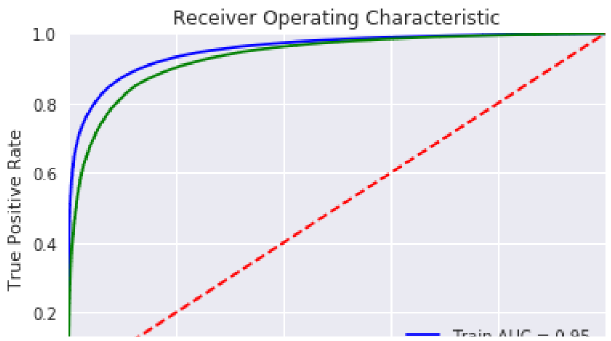

Test Data Confusion Matrix Confusion Matrix

```
The TPR is :  0.8558960197846314
The TNR is :  0.8687175724542738
The FPR is :  0.1312824275457261
The FNR is :  0.14410398021536858
```

```
 The auc of train is :  0.954154750092979

 The auc of train is :  0.9334759086688468
```

Out[0]:

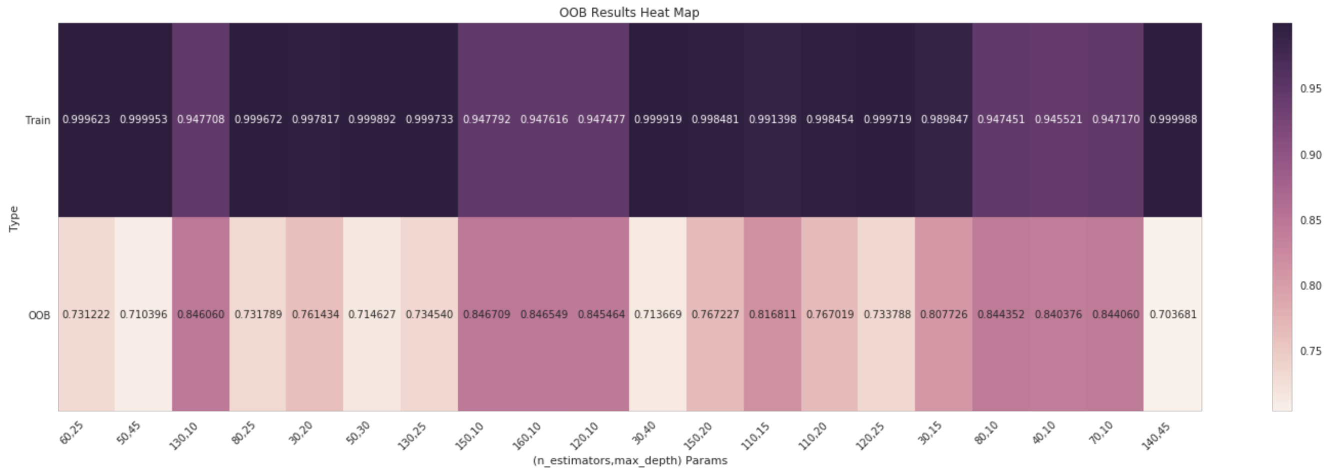

Receiver Operating Characteristic

[5.1.6] Applying Random Forests on TFIDF W2V, SET 4

In [0]:

```
random_params = random.sample(params_combo,20)
best_param_search(training_data,y_train,random_params)
```

Out[0]:

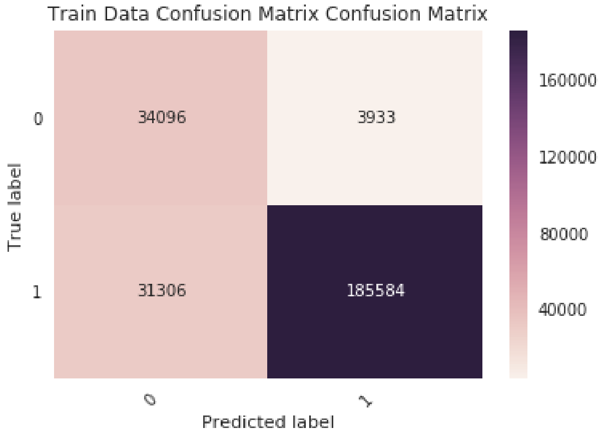


In [0]:

```
rfc = on_test(training_data,y_train,test_data,y_test,10,150)
```

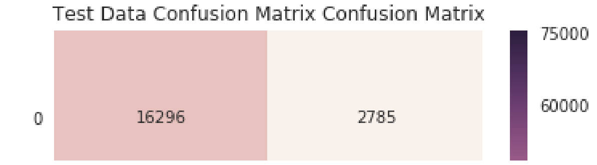
```
The ROC AUC score for the params max_depth 10  and n_estimators  150 is  0.9217006519609665
```

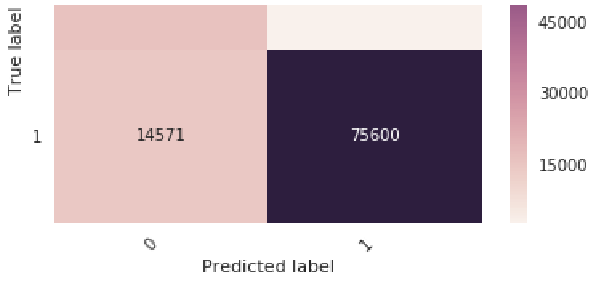
Out[0]:



```
The TPR is :  0.8556595509244317
The TNR is :  0.8965789266086408
The FPR is :  0.10342107339135923
The FNR is :  0.14434044907556826
```

Out[0]:



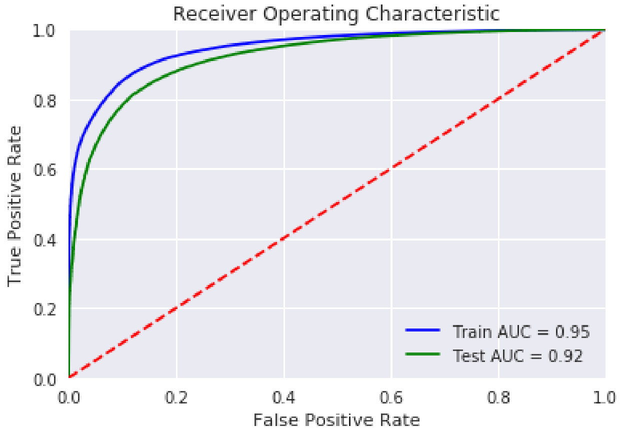


```
The TPR is :  0.8384070266493662
The TNR is :  0.8540432891357895
The FPR is :  0.14595671086421047
The FNR is :  0.1615929733506338
```

```
 The auc of train is :  0.9477922742398601

 The auc of train is :  0.9217006519609665
```

Out[0]:



[6] Conclusions

Model	Max Depth - n_estimators	Train FPR	Train FNR	Test FPR	Test FNR
Bag Of Words	25 - 100	0.0900	0.0985	0.1394	0.1150
TFIDF	30 -200	0.0789	0.0742	0.1527	0.0944
Avg W2V	10 - 50	0.0990	0.1270	0.1312	0.1441
TF-IDF W2V	10 - 150	0.1034	0.1443	0.1459	0.1615

Since our prime focus is to reduce FPR Avg W2V model has produced the best result