

RANDOM FOREST

February 2, 2019

1 Amazon Fine Food Reviews Analysis

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

EDA: <https://nycdatascience.com/blog/student-works/amazon-fine-foods-visualization/>

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 - unique 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 considered as a positive review. A rating of 1 or 2 can be considered as negative one. A review of rating 3 is considered neutral and such reviews are ignored from our analysis. This is an approximate and proxy way of determining the polarity (positivity/negativity) of a review.

2 [1]. Reading Data

3 [1.1] Loading the data

The dataset is available in two forms 1. .csv file 2. SQLite Database

In order to load the data, We have used the SQLite dataset as it is easier to query the data and visualise the data efficiently.

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 will be set to "positive". Otherwise, it will be set to "negative".

```
In [2]: 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.feature_extraction.text import TfidfTransformer
from sklearn.feature_extraction.text import TfidfVectorizer

from sklearn.model_selection import train_test_split
from sklearn.naive_bayes import MultinomialNB
from sklearn.metrics import accuracy_score
from sklearn.model_selection import cross_val_score
from collections import Counter
from sklearn.preprocessing import StandardScaler

from sklearn.feature_extraction.text import CountVectorizer
from sklearn.metrics import confusion_matrix
from sklearn import metrics
from sklearn.metrics import roc_curve, auc

import re
# Tutorial about Python regular expressions: https://pymotw.com/2/re/
import string
from nltk.corpus import stopwords
from nltk.stem import SnowballStemmer
from nltk.stem.wordnet import WordNetLemmatizer

import pickle
```

```
In [1]: data = pd.read_csv('Reviews.csv')
print (data.head())
print(data.shape)
```

	Id	ProductId	UserId	ProfileName	\
0	1	B001E4KFG0	A3SGXH7AUHU8GW	delmartian	
1	2	B00813GRG4	A1D87F6ZCVE5NK	dll pa	
2	3	B000LQOCHO	ABXLMWJIXXAIN	Natalia Corres	"Natalia Corres"
3	4	B000UA0QIQ	A395BORC6FGVXV	Karl	
4	5	B006K2ZZ7K	A1UQRSCLF8GW1T	Michael D. Bigham	"M. Wassir"

	HelpfulnessNumerator	HelpfulnessDenominator	Score	Time	\
--	----------------------	------------------------	-------	------	---

0	1	1	5	1303862400
1	0	0	1	1346976000
2	1	1	4	1219017600
3	3	3	2	1307923200
4	0	0	5	1350777600

	Summary	Text
0	Good Quality Dog Food	I have bought several of the Vitality canned d...
1	Not as Advertised	Product arrived labeled as Jumbo Salted Peanut...
2	"Delight" says it all	This is a confection that has been around a fe...
3	Cough Medicine	If you are looking for the secret ingredient i...
4	Great taffy	Great taffy at a great price. There was a wid...

(568454, 10)

3.1 [2] Data Cleaning: Deduplication and Nan features

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 [2]: #checking for Nan values in data. True indicates Nan values are present along the column
data.isnull().any()
```

```
Out[2]: Id                False
        ProductId         False
        UserId            False
        ProfileName        True
        HelpfulnessNumerator False
        HelpfulnessDenominator False
        Score              False
        Time               False
        Summary            True
        Text               False
        dtype: bool
```

```
In [18]: # checking for Nan values along 'profilename' column
#data[data['ProfileName'].isnull()].head()
```

```
In [19]: # checking for Nan values along 'summary' column
#data[data['Summary'].isnull()]
```

```
In [5]: #Dropping Nan values
data = data.dropna()
```

```
In [6]: #printing shape of data after dropping Nan values
print (data.shape)
```

(568411, 10)

```
In [7]: #Review score should lie between 1 to 5
        #Returns True if all the scores lie between 1 to 5(inclusive)
        list1 = data['Score'].map(lambda x: True if x in [1,2,3,4,5] else False)
        list1.all()
```

Out[7]: True

```
In [8]: filtered_data = data.loc[data['Score']!=3]
        print (filtered_data.head())
        print (filtered_data.shape)
```

	Id	ProductId	UserId	ProfileName	\
0	1	B001E4KFG0	A3SGXH7AUHU8GW	delmartian	
1	2	B00813GRG4	A1D87F6ZCVE5NK	dll pa	
2	3	B000LQOCHO	ABXLMWJIXXAIN	Natalia Corres	"Natalia Corres"
3	4	B000UA0QIQ	A395BORC6FGVXV	Karl	
4	5	B006K2ZZ7K	A1UQRSCLF8GW1T	Michael D. Bigham	"M. Wassir"

	HelpfulnessNumerator	HelpfulnessDenominator	Score	Time	\
0	1	1	5	1303862400	
1	0	0	1	1346976000	
2	1	1	4	1219017600	
3	3	3	2	1307923200	
4	0	0	5	1350777600	

	Summary	Text
0	Good Quality Dog Food	I have bought several of the Vitality canned d...
1	Not as Advertised	Product arrived labeled as Jumbo Salted Peanut...
2	"Delight" says it all	This is a confection that has been around a fe...
3	Cough Medicine	If you are looking for the secret ingredient i...
4	Great taffy	Great taffy at a great price. There was a wid...

(525773, 10)

```
In [9]: #mapping positive(>3) and negative(<3) reviews based on scores of the data.
        pos_negative = filtered_data['Score'].map(lambda x: 'positive' if int (x) >3 else 'negative')
        filtered_data['Score'] = pos_negative
        print ('shape of filtered_data')
        print (filtered_data.shape)
        print (filtered_data.head())
```

shape of filtered_data
(525773, 10)

	Id	ProductId	UserId	ProfileName	\
0	1	B001E4KFG0	A3SGXH7AUHU8GW	delmartian	
1	2	B00813GRG4	A1D87F6ZCVE5NK	dll pa	
2	3	B000LQOCHO	ABXLMWJIXXAIN	Natalia Corres	"Natalia Corres"
3	4	B000UA0QIQ	A395BORC6FGVXV	Karl	
4	5	B006K2ZZ7K	A1UQRSCLF8GW1T	Michael D. Bigham	"M. Wassir"

	HelpfulnessNumerator	HelpfulnessDenominator	Score	Time \
0	1	1	positive	1303862400
1	0	0	negative	1346976000
2	1	1	positive	1219017600
3	3	3	negative	1307923200
4	0	0	positive	1350777600

	Summary	Text
0	Good Quality Dog Food	I have bought several of the Vitality canned d...
1	Not as Advertised	Product arrived labeled as Jumbo Salted Peanut...
2	"Delight" says it all	This is a confection that has been around a fe...
3	Cough Medicine	If you are looking for the secret ingredient i...
4	Great taffy	Great taffy at a great price. There was a wid...

C:\Users\Lenovo\Anaconda3\lib\site-packages\ipykernel_launcher.py:3: SettingWithCopyWarning:
A value is trying to be set on a copy of a slice from a DataFrame.
Try using .loc[row_indexer,col_indexer] = value instead

See the caveats in the documentation: <http://pandas.pydata.org/pandas-docs/stable/indexing.html>
This is separate from the ipykernel package so we can avoid doing imports until

```
In [10]: #arranging data with increasing productid
sorted_data = filtered_data.sort_values('ProductId',axis=0,ascending=True,inplace=False)
```

```
In [20]: #finding the duplicates in our data
#If the same person gives for the same product at the same time we call it as suplica
#sorted_data.loc[sorted_data.duplicated(["UserId","ProfileName","Time","Text"],keep =
```

```
In [12]: #counting number of duplicates present in our data
sorted_data.duplicated(["UserId","ProfileName","Time","Text"]).sum()
```

```
Out[12]: 161612
```

```
In [13]: #dropping all duplicates keeping the first one
final = sorted_data.drop_duplicates(subset={"UserId","ProfileName","Time","Text"},keep =
final.shape
```

```
Out[13]: (364161, 10)
```

```
In [14]: #helpfulness numerator denotes number of people who found the review helpful
#helpfulness denominator denotes number of people who indicated whether or not the re
#so, helpfulness numerator should be less than denominator
final = final[final.HelpfulnessNumerator<=final.HelpfulnessDenominator]
```

```
In [15]: #final shape of data after preprocessing
final.shape
```

```
Out[15]: (364159, 10)
```

```
In [16]: #counting number of positive and negative reviews present
         final['Score'].value_counts()
```

```
Out[16]: positive    307054
         negative     57105
         Name: Score, dtype: int64
```

4 [3] Preprocessing

5 [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
7. Finally Snowball Stemming the word (it was observed to be better than Porter Stemming)

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

```
In [18]: stop = set(stopwords.words('english')) #set of stopwords
         sno = nltk.stem.SnowballStemmer('english') #initialising the snowball stemmer
         not_words = re.findall(r'\w*n[\'|o]t',str(stop)) #finding NOT words in stop words
         not_words.append('n\'t')
         not_words.append('no')
         print(not_words)
         stop_words = stop - set(not_words) #removing NOT words from stop words
         # https://stackoverflow.com/a/47091490/4084039
         import re

         def decontracted(phrase):
             # specific
             phrase = re.sub(r'\w*n[\'|o]t', "not", phrase)
             # general
             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 cleanhtmlpunc(sentence): #function to clean the word of any html-tags
    clean = re.compile('<.*?>')
    clean = re.sub(clean, ' ', sentence)
    clean = re.sub(r"(http|www)\S+", "", clean)
    clean = re.sub(r"\S+com", "", clean)
    #clean = re.sub(r"(\w+)", "", clean)
    clean = re.sub(r"\.", " ", clean)
    cleaned = re.sub(r'[?+|!+\\"'+|"+|#+|:|+]', r' ', clean)
    cleantext = re.sub(r'[\.,+|+|+|+|+|/+]', r' ', cleaned)
    return cleantext
#def cleanpunc(sentence): #function to clean the word of any punctuation or special c
    # return cleaned
print('*****')
print(stop_words)

```

```

["won't", 'not', "needn't", "don't", "mustn't", "haven't", "aren't", "shan't", "wouldn't", "mi
*****
{'his', 'who', "you've", 'don', 'o', 'your', 'mustn', 'my', 'other', 'both', 'out', 'wasn', "s

```

```

In [19]: def cleanedtext(reviews):
    str1=' '
    final_string=[]
    s=' '
    for sent in reviews:
        filtered_sentence=[]
        sent=cleanhtmlpunc(decontracted(sent)) # remove HTML tags
        for w in sent.split():
            for cleaned_words in w.split():
                if((cleaned_words.isalpha()) & (len(cleaned_words)>2)):
                    if((cleaned_words.lower() not in stop_words)):
                        s=(sno.stem(cleaned_words.lower())).encode('utf8')
                        filtered_sentence.append(s)
                    else:
                        continue
                else:
                    continue
            str1 = b" ".join(filtered_sentence) #final string of cleaned words for review
            final_string.append(str1)
    return final_string

```

```

In [20]: final_data['CleanedText']=final_string #adding a column of CleanedText which displays
final_data.head() #below the processed review can be seen in the CleanedText Column

```

```

Out[20]:
      Id  ProductId  UserId ProfileName  HelpfulnessNumerator  \
370049  370050  B001G8W7SI  A3TT44LPX0926Z      J "J"          0
268423  268424  B0000GIVAU  A302B2MHY8R5PD      izzy          0

```

449524	449525	B003P9WU6A	ADD2FROEIZCPK	Matthr	1
512616	512617	B003RM9SA6	A37700CWJ4XD69	Macoma	6
368187	368188	B000SAPXPY	A3MQ68M8KRBQUH	Claudia	2

	Helpfulness	Denominator	Score	Time	\
370049		0	positive	1330905600	
268423		0	positive	1341360000	
449524		1	positive	1296604800	
512616		6	positive	1290988800	
368187		2	positive	1340928000	

	Summary	\
370049	excellent product--wish it was cheaper!	
268423	Nothing compares	
449524	Love it! Leaves cat's coat beautiful!	
512616	Best Vanilla We Could Find	
368187	Excellent flavor	

	Text	\
370049	great product but wish it could be found a bit...	
268423	Honestly, I used to hate mineral water until I...	
449524	I have been using this product for a while now...	
512616	We have friends and family that bring back van...	
368187	I'm a big fan of good, strong British tea like...	

	CleanedText
370049	b'great product wish could found bit cheaper...
268423	b'honest use hate miner water tri everi singl ...
449524	b'use product cat coat shini healthi greatest ...
512616	b'friend famili bring back vanilla mexico far ...
368187	b'big fan good strong british tea like tip tol...

6 Applying Random Forest

7 [4.1] Random Forest on BOW, SET 1

```
In [2]: import pickle
        with open("final_data.pkl", "rb") as f:
            final_data = pickle.load(f)

In [3]: final_data = final_data.iloc[0:100000,:]
```

```
In [4]: from sklearn.model_selection import train_test_split
        from sklearn.model_selection import cross_val_score
        from sklearn.metrics import accuracy_score
        from sklearn.ensemble import RandomForestClassifier
```



```

# split the data set into train and test
X_train, X_test, y_train, y_test = train_test_split(final_data['CleanedText'].values,
                                                    test_size=0.30, random_state=42)

```

C:\Users\Lenovo\Anaconda3\lib\site-packages\sklearn\ensemble\weight_boosting.py:29: DeprecationWarning: inner1d

```

from numpy.core.umath_tests import inner1d

```

```

In [7]: # intializing for bag of words
model= CountVectorizer(dtype=float,min_df=5,ngram_range=(1,2))
final_counts= model.fit_transform(X_train)

```

```

In [8]: #standardizing the bag of words
from sklearn.ensemble import GradientBoostingClassifier
standardizing = StandardScaler(with_mean = False)
final_std_data = standardizing.fit_transform(final_counts)
final_std_data.shape

```

Out[8]: (70000, 89447)

```

In [7]: from sklearn.metrics import make_scorer
from sklearn.model_selection import cross_val_score
from sklearn.metrics import roc_auc_score

# creating list for hyperparameter alpha
depth = [2,3,5,10,50]
estimators= [5,10,50,100,200,500]
# empty list that will hold cv scores

cv_scores = []
train_auc_values = []

# perform 10-fold cross validation
for d in depth:
    print (d)
    train = []
    cv = []
    for k in estimators:
        rfc=RandomForestClassifier(n_estimators=k,max_depth=d,class_weight='balanced',
        auc = make_scorer(roc_auc_score,greater_is_better=True,
                           needs_threshold=True)
        auc_scores = cross_val_score(rfc, final_std_data, y_train, cv=10, scoring=auc)
        rfc.fit(final_std_data,y_train)
        y_pred_proba = rfc.predict_proba(final_std_data)[::,1]
        train_auc = roc_auc_score(y_train, y_pred_proba)
        train.append(train_auc)
        cv.append(auc_scores.mean())
    train_auc_values.append(train)
    cv_scores.append(cv)

```

```

print ('train data scores')
print (train_auc_values)
print (''*50)
print ('CV scores')
print (cv_scores)

2
3
5
10
50
train data scores
[[0.58760141272581612, 0.67004897940438024, 0.7939194005516369, 0.8570231570316289, 0.88733754
*****
CV scores
[[0.60135298131588755, 0.63051601788059064, 0.78941796820792831, 0.84096902887076763, 0.870531

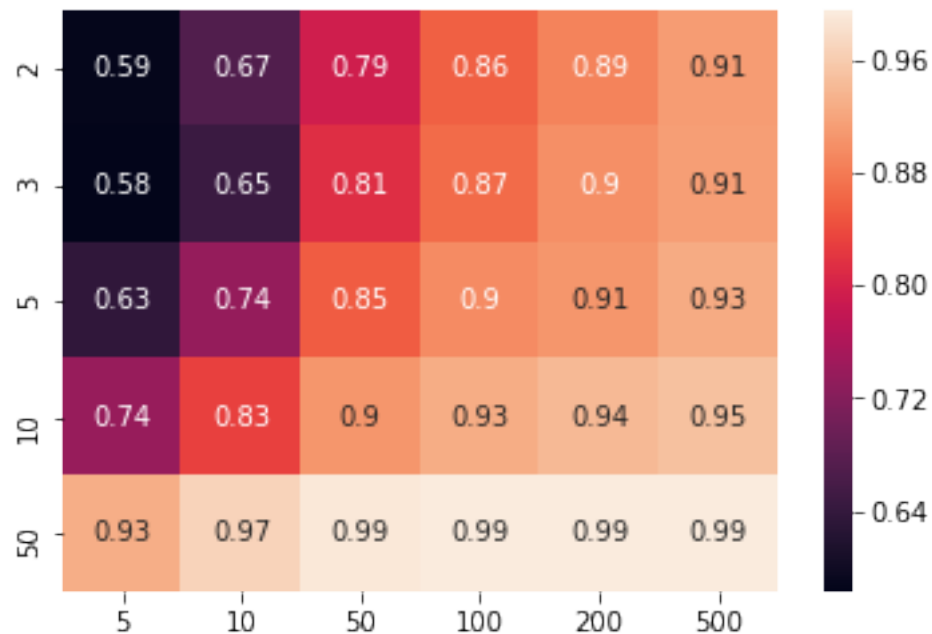
```

```

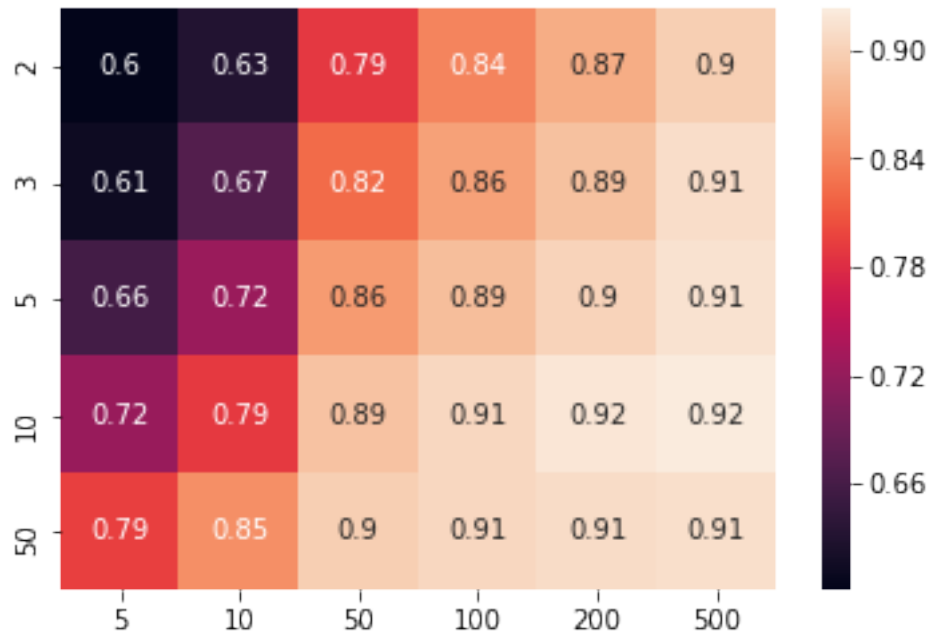
In [8]: print (''*50+'train heat map'*50)
        train_values = pd.DataFrame(data=train_auc_values,index=[2,3,5,10,50],columns=[5,10,50
        ax = sns.heatmap(train_values,annot=True)
        plt.show(ax)
        print (''*50+'CV heat map'*50)
        CV_values = pd.DataFrame(data=cv_scores,index=[2,3,5,10,50],columns=[5,10,50,100,200,50
        ax1 = sns.heatmap(CV_values,annot=True)
        plt.show(ax1)

```

*****train heat map*****



*****CV heat map*****



```
In [9]: rfc=RandomForestClassifier(n_estimators=200,max_depth=10,class_weight='balanced',n_jobs=-1)
rfc.fit(final_std_data,y_train)
```

```
Out[9]: RandomForestClassifier(bootstrap=True, class_weight='balanced',
                                criterion='gini', max_depth=10, 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=200, n_jobs=-1, oob_score=False,
                                random_state=None, verbose=0, warm_start=False)
```

```
In [10]: from sklearn.metrics import confusion_matrix
from sklearn.metrics import f1_score
from sklearn.metrics import precision_score
from sklearn.metrics import recall_score

predictions=rfc.predict(standardizing.transform(model.transform(X_test)))
print('accuracy = {0}'.format(rfc.score(standardizing.transform(model.transform(X_test))))
precision = precision_score(y_test, predictions,pos_label=1)
Recall = recall_score(y_test, predictions,pos_label=1)
```

```

f1 = f1_score(y_test, predictions,pos_label=1)
print ('precision = {0}'.format(precision))
print ('Recall={0}'.format(Recall))
print ('f1_score={0}'.format(f1))

```

```

accuracy = 87.60333333333334
precision = 0.9607794258174415
Recall=0.8928571428571429
f1_score=0.9255738557905901

```

```

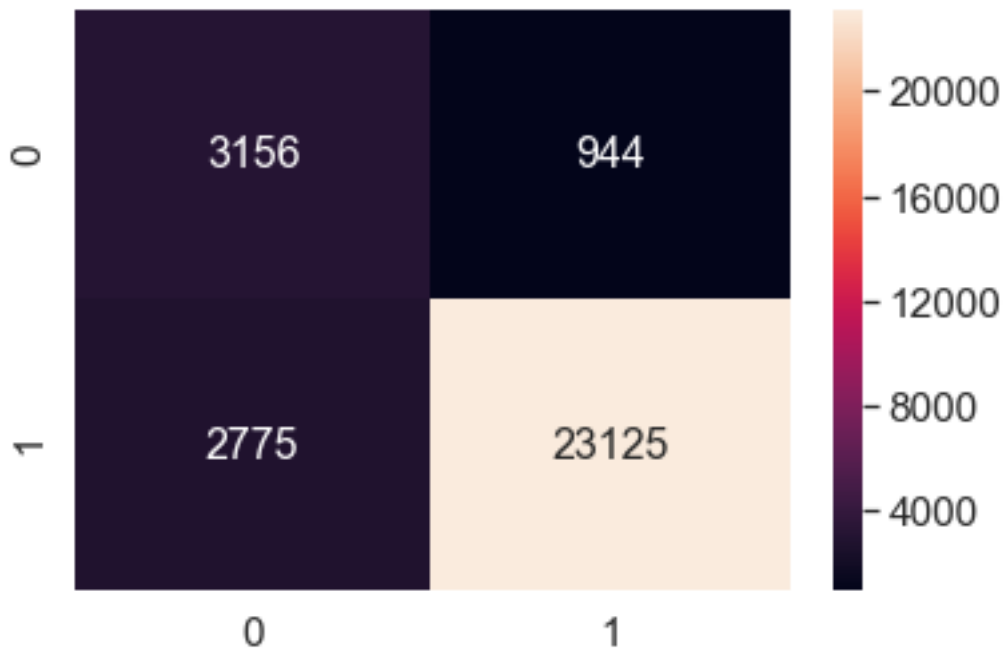
In [11]: %matplotlib inline
from sklearn.metrics import confusion_matrix
import seaborn as sns
result = confusion_matrix(y_test,predictions)
#print(result)
sns.set(font_scale=1.4)#for label size
sns.heatmap(result, annot=True,annot_kws={"size": 16}, fmt='g')

```

```

Out[11]: <matplotlib.axes._subplots.AxesSubplot at 0x133e77048>

```

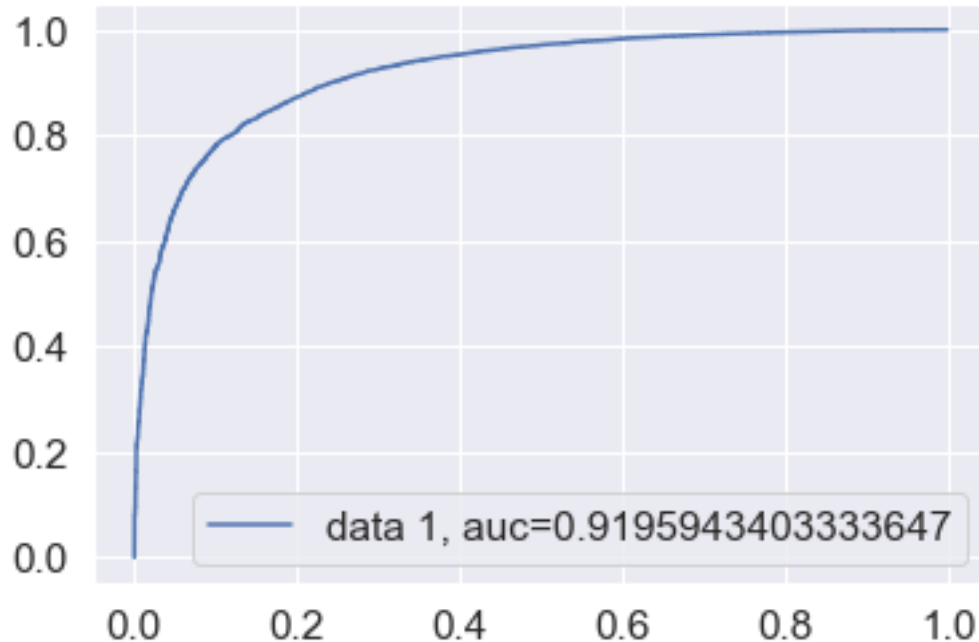


```

In [12]: y_pred_proba = rfc.predict_proba(standardizing.transform(model.transform(X_test)))[:,
fpr, tpr, _ = roc_curve(y_test, y_pred_proba,pos_label=1 )
#auc = roc_auc_score(y_test, y_pred_proba)
auc = np.trapz(tpr,fpr)

```

```
plt.plot(fpr, tpr, label="data 1, auc="+str(auc))
plt.legend(loc=4)
plt.show()
```



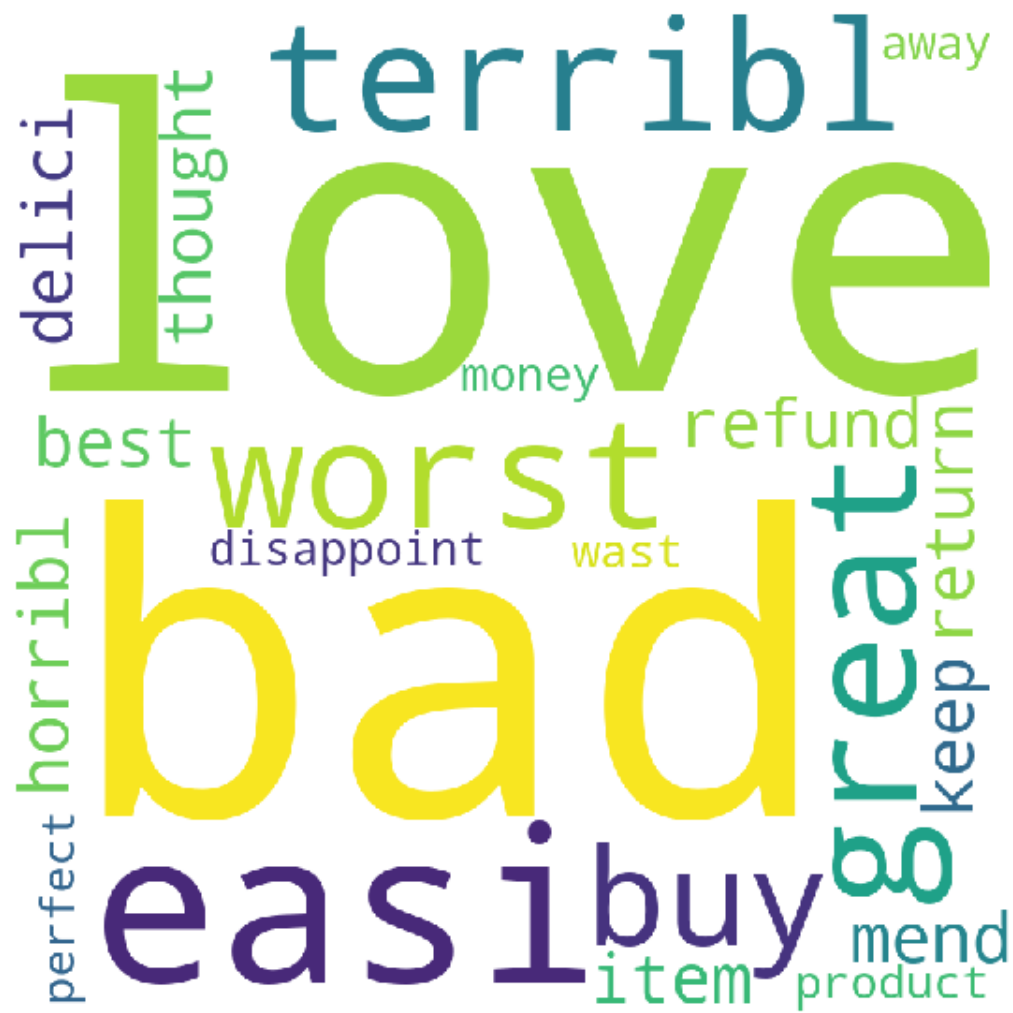
7.0.1 [4.1.1] Feature Importance

```
In [14]: feat_names = model.get_feature_names()
feat_names = np.array(feat_names)
# Sort feature importances in descending order
indices = np.argsort(rfc.feature_importances_)[::-1][:25]
print (feat_names[indices])
```

```
['not' 'best' 'great' 'would' 'disappoint' 'wast' 'thought' 'love'
 'not buy' 'return' 'product' 'not mend' 'money' 'refund' 'bad' 'away'
 'terribl' 'delici' 'item' 'horribl' 'easi' 'worst' 'perfect' 'would not'
 'keep']
```

```
In [15]: from wordcloud import WordCloud
wordcloud = WordCloud(width = 800, height = 800,
                      background_color = 'white').generate(' '.join(feat_names[indices]))
plt.figure(figsize = (8, 8), facecolor = None)
plt.imshow(wordcloud)
plt.axis("off")
plt.tight_layout(pad = 0)
```

```
plt.show()
```



8 [4.2] Random Forest on TFIDF, SET 2

```
In [16]: # split the data set into train and test
         tfidf_train, tfidf_test, y_train, y_test = train_test_split(final_data['CleanedText'],
                                                                     final_data['Score'].values,
                                                                     test_size=0.3, random_state=42,
                                                                     shuffle=False)

In [17]: vector = TfidfVectorizer(ngram_range = (1,2),min_df=5)
         tf_idf_vector = vector.fit_transform(tfidf_train)
```

```

standardizing = StandardScaler(with_mean = False)
tfidf_std_data = standardizing.fit_transform(tf_idf_vector)
tfidf_std_data.shape

```

Out[17]: (70000, 89447)

```

In [11]: from sklearn.metrics import make_scorer
from sklearn.model_selection import cross_val_score
from sklearn.metrics import roc_auc_score

# creating list for hyperparameter alpha
depth = [2,3,5,10,50]
estimators= [5,10,50,100,200,500]
# empty list that will hold cv scores

cv_scores = []
train_auc_values = []

# perform 10-fold cross validation
for d in depth:
    train = []
    cv = []
    print (d)
    for k in estimators:
        rfc=RandomForestClassifier(n_estimators=k,max_depth=d,class_weight='balanced'
        auc = make_scorer(roc_auc_score,greater_is_better=True,
                           needs_threshold=True)
        auc_scores = cross_val_score(rfc, tfidf_std_data, y_train, cv=10, scoring=auc)
        rfc.fit(tfidf_std_data,y_train)
        y_pred_proba = rfc.predict_proba(tfidf_std_data)[:,:1]
        train_auc = roc_auc_score(y_train, y_pred_proba)
        train.append(train_auc)
        cv.append(auc_scores.mean())
    train_auc_values.append(train)
    cv_scores.append(cv)

print ('train data scores')
print (train_auc_values)
print ('*'*50)
print ('CV scores')
print (cv_scores)

```

```

2
3
5
10
50
train data scores

```

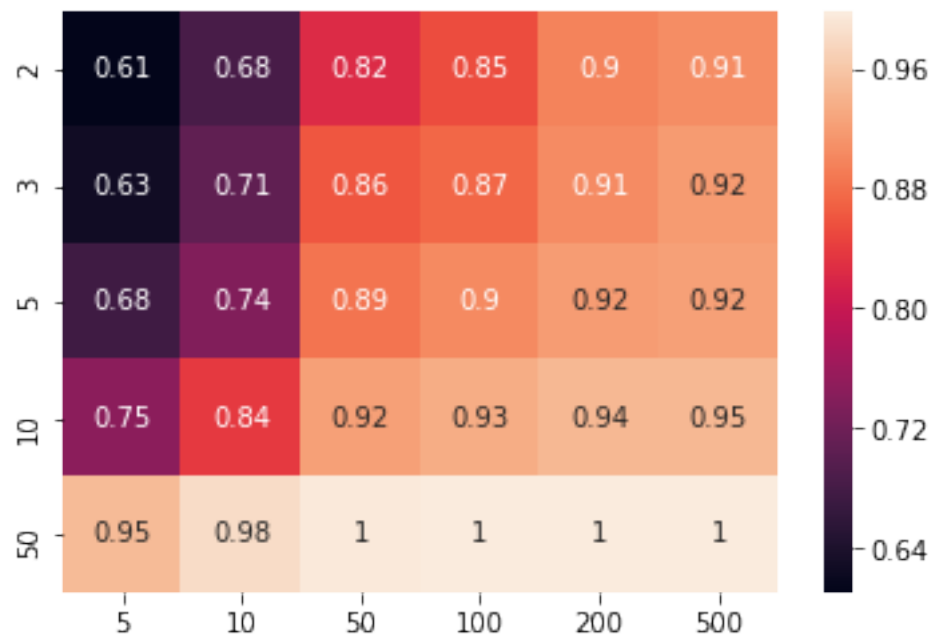
```
[[0.61086528070511981, 0.68403660409726919, 0.82442561344516507, 0.8528345914717399, 0.8972298
*****
```

CV scores

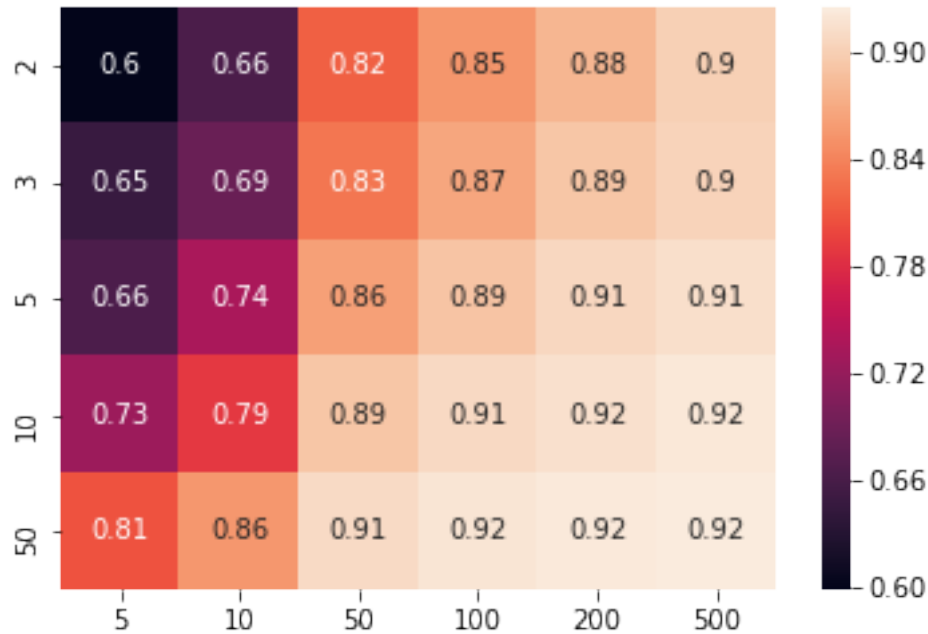
```
[[0.5982760900649331, 0.66255171472456453, 0.81557578019725185, 0.85407751759819983, 0.8797078
```

```
In [12]: print (' '*50+'train heat map'+' '*50)
         train_values = pd.DataFrame(data=train_auc_values,index=[2,3,5,10,50],columns=[5,10,50,100,200,500])
         ax = sns.heatmap(train_values,annot=True)
         plt.show(ax)
         print (' '*50+'CV heat map'+' '*50)
         CV_values = pd.DataFrame(data=cv_scores,index=[2,3,5,10,50],columns=[5,10,50,100,200,500])
         ax1 = sns.heatmap(CV_values,annot=True)
         plt.show(ax1)
```

```
*****train heat map*****
```



```
*****CV heat map*****
```

```
In [18]: rfc=RandomForestClassifier(n_estimators=200,max_depth=10,class_weight='balanced',n_jobs=-1)
rfc.fit(tfidf_std_data,y_train)
```

```
Out[18]: RandomForestClassifier(bootstrap=True, class_weight='balanced',
                                criterion='gini', max_depth=10, 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=200, n_jobs=-1, oob_score=False,
                                random_state=None, verbose=0, warm_start=False)
```

```
In [19]: from sklearn.metrics import confusion_matrix
from sklearn.metrics import f1_score
from sklearn.metrics import precision_score
from sklearn.metrics import recall_score

predictions=rfc.predict(standardizing.transform(vector.transform(tfidf_test)))
print('accuracy = {0}'.format(rfc.score(standardizing.transform(vector.transform(tfidf_test)),y_test)))
precision = precision_score(y_test, predictions,pos_label=1)
Recall = recall_score(y_test, predictions,pos_label=1)
f1 = f1_score(y_test, predictions,pos_label=1)
print ('precision = {0}'.format(precision))
print ('Recall={0}'.format(Recall))
print ('f1_score={0}'.format(f1))
```

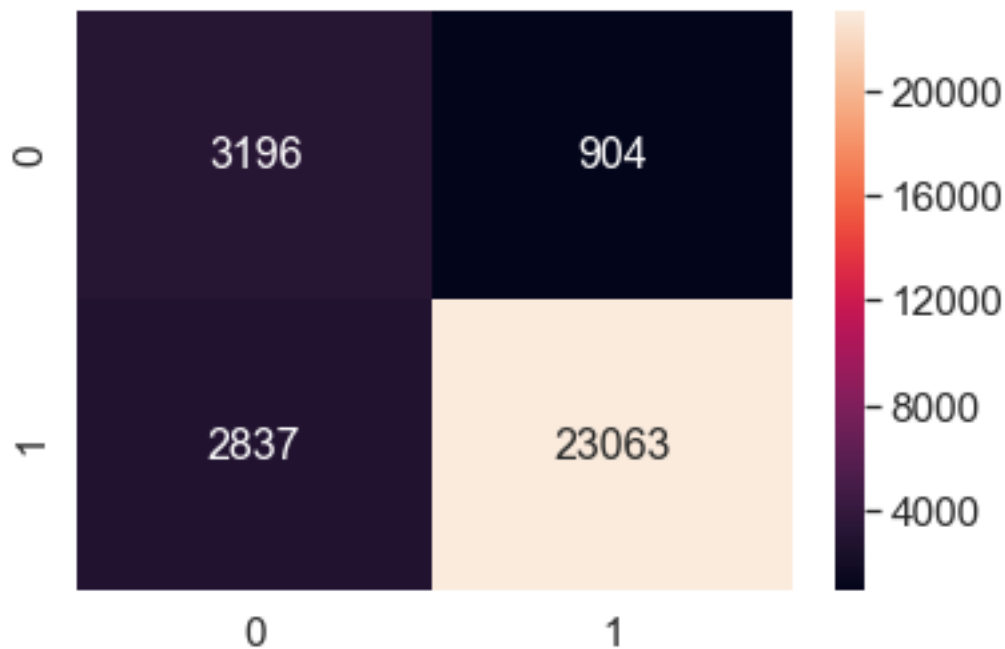
accuracy = 87.53

precision = 0.9622814703550716

Recall=0.8904633204633204
f1_score=0.9249804479916578

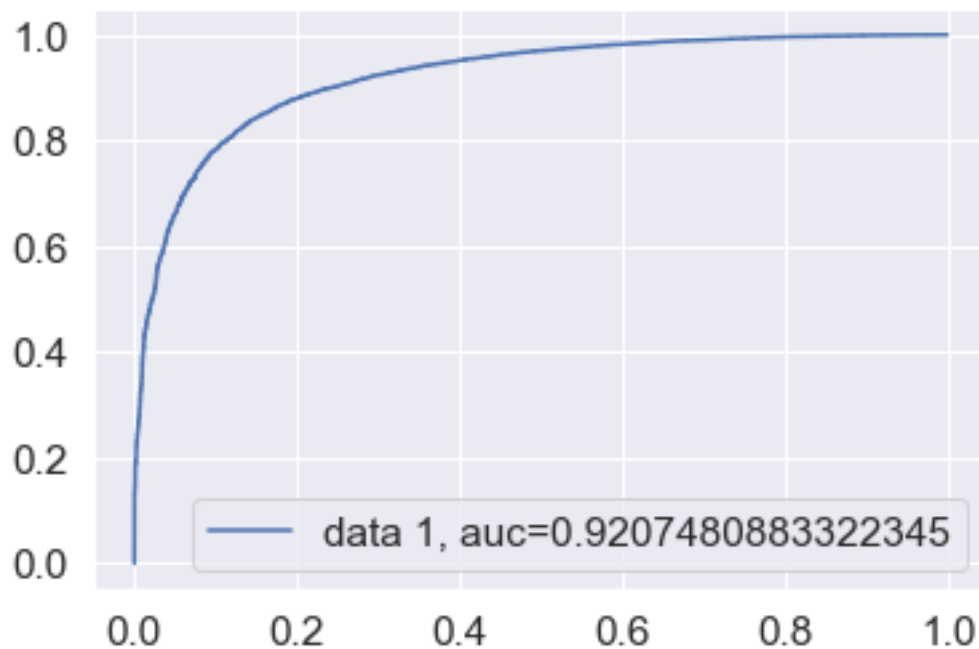
```
In [20]: %matplotlib inline
from sklearn.metrics import confusion_matrix
import seaborn as sns
result = confusion_matrix(y_test,predictions)
#print(result)
sns.set(font_scale=1.4)#for label size
sns.heatmap(result, annot=True,annot_kws={"size": 16}, fmt='g')
```

Out[20]: <matplotlib.axes._subplots.AxesSubplot at 0x12a25c080>



8.0.1 [4.2.1] Feature Importance

```
In [21]: y_pred_proba = rfc.predict_proba(standardizing.transform(vector.transform(tfidf_test)))
fpr, tpr, _ = roc_curve(y_test, y_pred_proba, pos_label=1 )
#auc = roc_auc_score(y_test, y_pred_proba)
auc = np.trapz(tpr,fpr)
plt.plot(fpr,tpr,label="data 1, auc="+str(auc))
plt.legend(loc=4)
plt.show()
```



```
In [22]: feat_names = vector.get_feature_names()
         feat_names = np.array(feat_names)
         # Sort feature importances in descending order
         indices = np.argsort(rfc.feature_importances_)[::-1][:25]
         print (feat_names[indices])
```

```
['not' 'would' 'great' 'best' 'high mend' 'perfect' 'love' 'delici' 'bad'
 'disappoint' 'thought' 'money' 'return' 'worst' 'product' 'wast' 'threw'
 'not mend' 'find' 'easi' 'not buy' 'keep' 'wast money' 'look' 'bought']
```

```
In [23]: from wordcloud import WordCloud
         wordcloud = WordCloud(width = 800, height = 800,
                               background_color = 'white').generate(' '.join(feat_names[indices]))
         plt.figure(figsize = (8, 8), facecolor = None)
         plt.imshow(wordcloud)
         plt.axis("off")
         plt.tight_layout(pad = 0)

         plt.show()
```



9 [4.3] Random Forest on Avg W2V, SET 3

```
In [12]: import pickle
         with open("avg_w2v_train_data.pkl", "rb") as f:
             avg_w2v_train_data = pickle.load(f)

         with open("avg_w2v_test_data.pkl", "rb") as f:
             avg_w2v_test_data = pickle.load(f)

         with open("y_train.pkl", "rb") as f:
             y_train = pickle.load(f)

         with open("y_test.pkl", "rb") as f:
```

```
y_test = pickle.load(f)
```

```
In [9]: standardizing = StandardScaler(with_mean = False)
avg_w2v_std_train_data = standardizing.fit_transform(avg_w2v_train_data)
avg_w2v_std_train_data.shape
```

```
Out[9]: (70000, 300)
```

```
In [15]: from sklearn.metrics import make_scorer
from sklearn.model_selection import cross_val_score
from sklearn.metrics import roc_auc_score
```

```
# creating list for hyperparameter alpha
depth = [2,3,5,10,50]
estimators= [5,10,50,100,200,500]
# empty list that will hold cv scores
```

```
cv_scores = []
train_auc_values = []
```

```
# perform 10-fold cross validation
```

```
for d in depth:
```

```
    train = []
```

```
    cv = []
```

```
    print (d)
```

```
    for k in estimators:
```

```
        rfc=RandomForestClassifier(n_estimators=k,max_depth=d,class_weight='balanced')
        auc = make_scorer(roc_auc_score,greater_is_better=True,
                           needs_threshold=True)
```

```
        auc_scores = cross_val_score(rfc, avg_w2v_std_train_data, y_train, cv=10, scoring='roc_auc')
```

```
        rfc.fit(avg_w2v_std_train_data,y_train)
```

```
        y_pred_proba = rfc.predict_proba(avg_w2v_std_train_data)[::,1]
```

```
        train_auc = roc_auc_score(y_train, y_pred_proba)
```

```
        train.append(train_auc)
```

```
        cv.append(auc_scores.mean())
```

```
    train_auc_values.append(train)
```

```
    cv_scores.append(cv)
```

```
print ('train data scores')
```

```
print (train_auc_values)
```

```
print (' '*50)
```

```
print ('CV scores')
```

```
print (cv_scores)
```

```
2
3
5
10
50
```

```

train data scores
[[0.8041602908515888, 0.81961270993965962, 0.85740548422046392, 0.86294148226144607, 0.8567914
*****
CV scores
[[0.79290402911128965, 0.81669218378421837, 0.84951378904252606, 0.85079316861826104, 0.851640

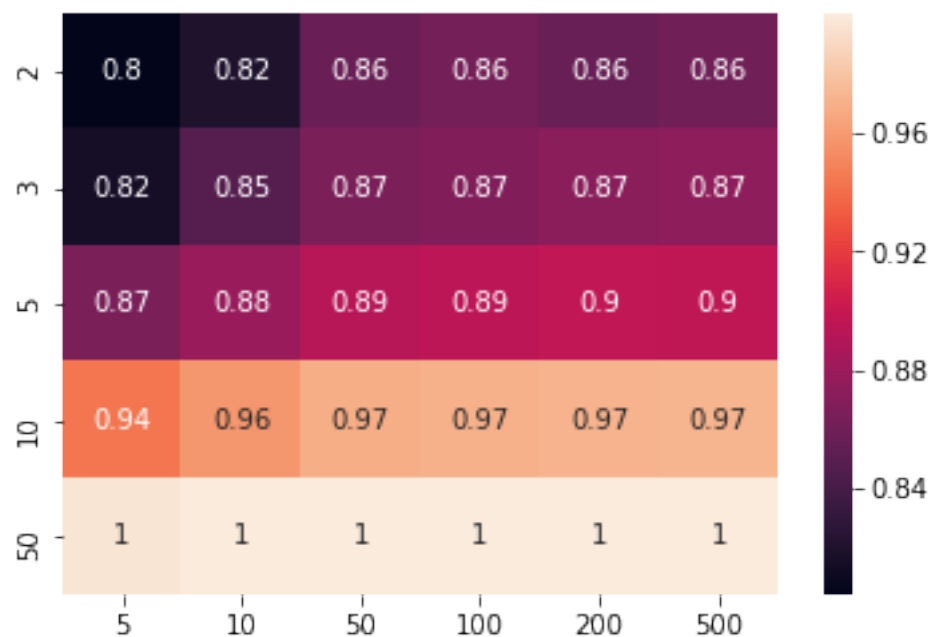
```

```

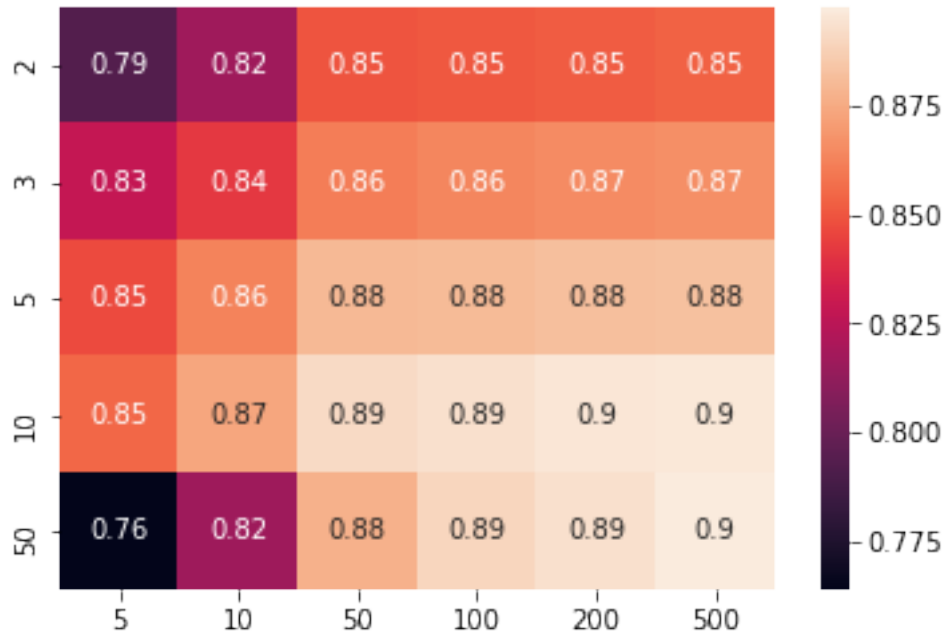
In [16]: print (''*50+'train heat map'*50)
         train_values = pd.DataFrame(data=train_auc_values,index=[2,3,5,10,50],columns=[5,10,50,100,200,500])
         ax = sns.heatmap(train_values,annot=True)
         plt.show(ax)
         print (''*50+'CV heat map'*50)
         CV_values = pd.DataFrame(data=cv_scores,index=[2,3,5,10,50],columns=[5,10,50,100,200,500])
         ax1 = sns.heatmap(CV_values,annot=True)
         plt.show(ax1)

```

*****train heat map*****



*****CV heat map*****



```
In [10]: from sklearn.ensemble import RandomForestClassifier
rfc=RandomForestClassifier(n_estimators=200,max_depth=5,class_weight='balanced',n_jobs=-1)
rfc.fit(avg_w2v_std_train_data,y_train)
```

```
Out[10]: RandomForestClassifier(bootstrap=True, class_weight='balanced',
                                criterion='gini', max_depth=5, 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=200, n_jobs=-1, oob_score=False,
                                random_state=None, verbose=0, warm_start=False)
```

```
In [13]: from sklearn.metrics import confusion_matrix
from sklearn.metrics import f1_score
from sklearn.metrics import precision_score
from sklearn.metrics import recall_score

predictions=rfc.predict(standardizing.transform(avg_w2v_test_data))
print('accuracy = {0}'.format(rfc.score(standardizing.transform(avg_w2v_test_data), y_test)))
precision = precision_score(y_test, predictions,pos_label=1)
Recall = recall_score(y_test, predictions,pos_label=1)
f1 = f1_score(y_test, predictions,pos_label=1)
print ('precision = {0}'.format(precision))
print ('Recall={0}'.format(Recall))
print ('f1_score={0}'.format(f1))
```

```

accuracy = 84.43666666666667
precision = 0.9220035778175313
Recall=0.8954826254826255
f1_score=0.9085496033689158

```

```

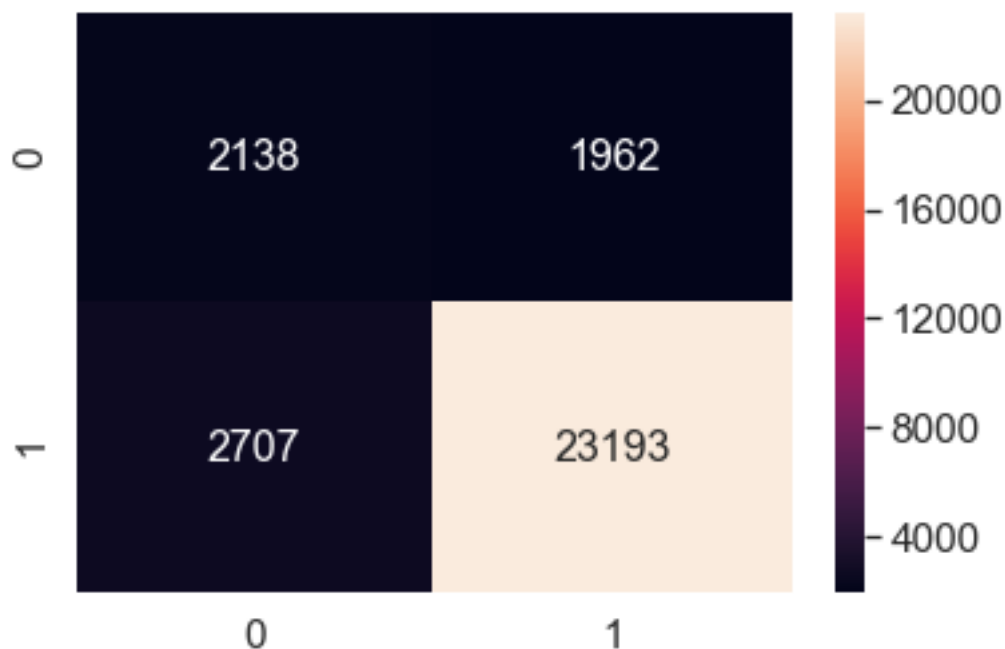
In [14]: %matplotlib inline
         from sklearn.metrics import confusion_matrix
         import seaborn as sns
         result = confusion_matrix(y_test,predictions)
         #print(result)
         sns.set(font_scale=1.4)#for label size
         sns.heatmap(result, annot=True,annot_kws={"size": 16}, fmt='g')

```

```

Out[14]: <matplotlib.axes._subplots.AxesSubplot at 0x2b811aee48>

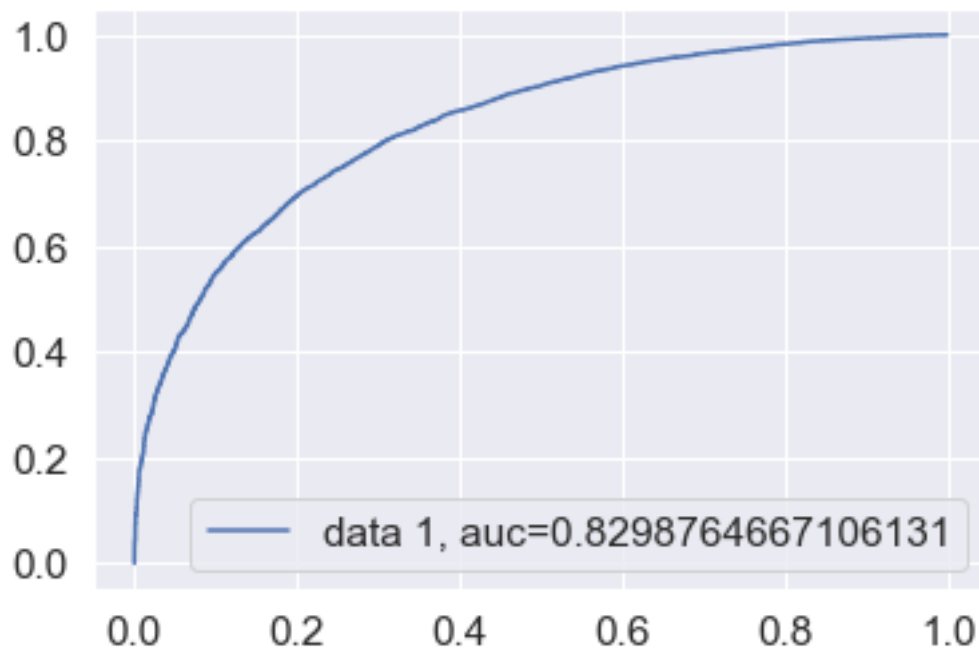
```



```

In [16]: y_pred_proba = rfc.predict_proba(standardizing.transform(avg_w2v_test_data))[:,1]
         fpr, tpr, _ = roc_curve(y_test, y_pred_proba, pos_label=1 )
         #auc = roc_auc_score(y_test, y_pred_proba)
         auc = np.trapz(tpr,fpr)
         plt.plot(fpr,tpr,label="data 1, auc="+str(auc))
         plt.legend(loc=4)
         plt.show()

```

10 [4.4] Random Forest on TFIDF W2V, SET 4

```
In [29]: import pickle
         with open("tfidf_w2v_train_data.pkl", "rb") as f:
             tfidf_w2v_train_data = pickle.load(f)

         with open("tfidf_w2v_test_data.pkl", "rb") as f:
             tfidf_w2v_test_data = pickle.load(f)

In [30]: standardizing = StandardScaler(with_mean = False)
         tfidf_w2v_std_train_data = standardizing.fit_transform(tfidf_w2v_train_data)
         tfidf_w2v_std_train_data.shape

Out[30]: (70000, 300)

In [19]: from sklearn.metrics import make_scorer
         from sklearn.model_selection import cross_val_score
         from sklearn.metrics import roc_auc_score

         # creating list for hyperparameter alpha
         depth = [2,3,5,10,50]
         estimators= [5,10,50,100,200,500]
         # empty list that will hold cv scores

         cv_scores = []
```

```

train_auc_values = []

# perform 10-fold cross validation
for d in depth:
    train = []
    cv = []
    print (d)
    for k in estimators:
        rfc=RandomForestClassifier(n_estimators=k,max_depth=d,class_weight='balanced')
        auc = make_scorer(roc_auc_score,greater_is_better=True,
                           needs_threshold=True)
        auc_scores = cross_val_score(rfc, tfidf_w2v_std_train_data, y_train, cv=10, scoring=auc)
        rfc.fit(tfidf_w2v_std_train_data,y_train)
        y_pred_proba = rfc.predict_proba(tfidf_w2v_std_train_data)[::,1]
        train_auc = roc_auc_score(y_train, y_pred_proba)
        train.append(train_auc)
        cv.append(auc_scores.mean())
    train_auc_values.append(train)
    cv_scores.append(cv)

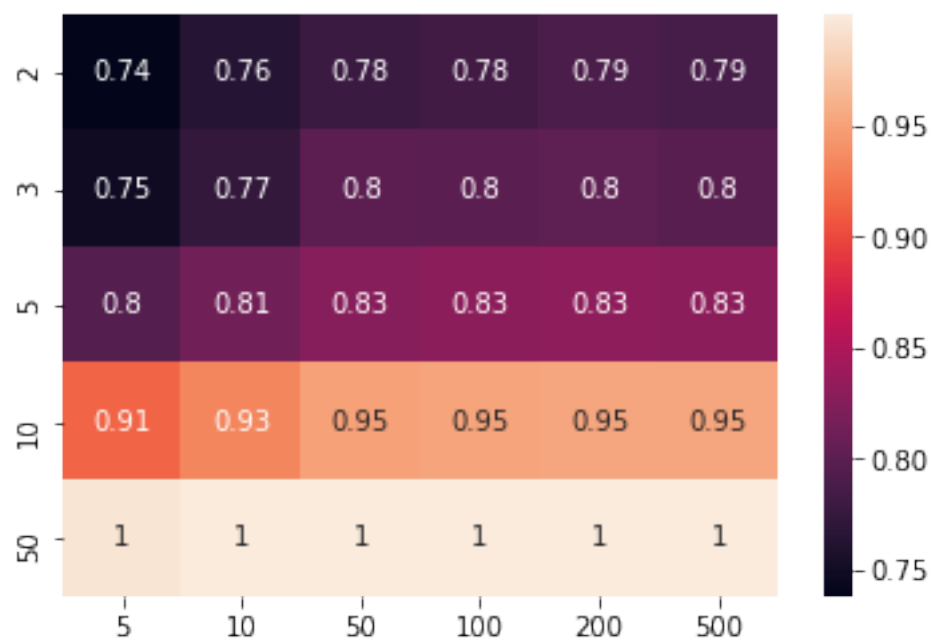
print ('train data scores')
print (train_auc_values)
print (''*50)
print ('CV scores')
print (cv_scores)

2
3
5
10
50
train data scores
[[0.73793846831175858, 0.76381056578912732, 0.77846501967431125, 0.78395994485610343, 0.7893211000000001]
*****
CV scores
[[0.73854256676863628, 0.75462865962283354, 0.77554283757869669, 0.77718230743237027, 0.7796240000000001]

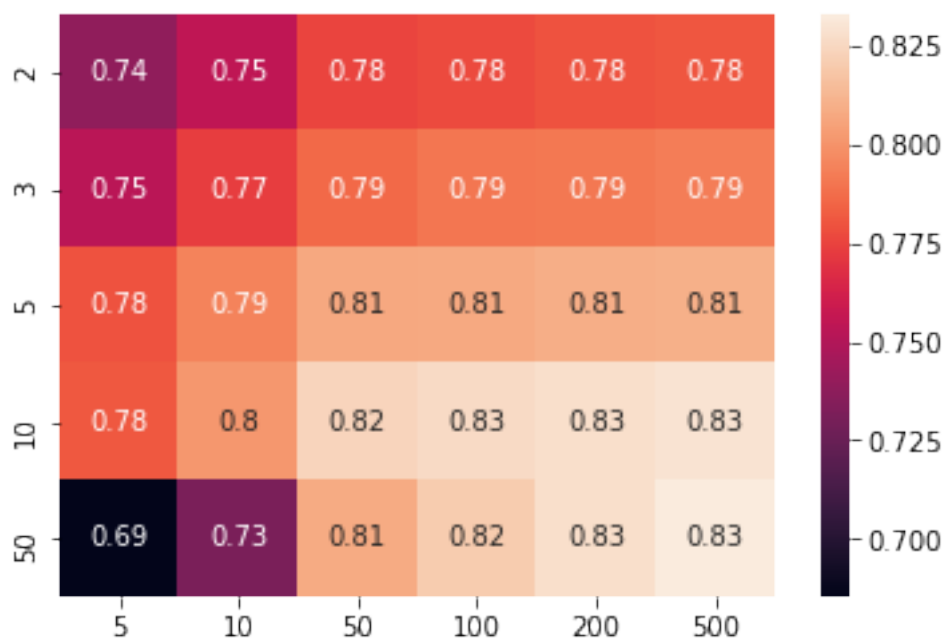
In [20]: print (''*50+'train heat map'+''*50)
         train_values = pd.DataFrame(data=train_auc_values,index=[2,3,5,10,50],columns=[5,10,50,100,200,500])
         ax = sns.heatmap(train_values,annot=True)
         plt.show(ax)
         print (''*50+'CV heat map'+''*50)
         CV_values = pd.DataFrame(data=cv_scores,index=[2,3,5,10,50],columns=[5,10,50,100,200,500])
         ax1 = sns.heatmap(CV_values,annot=True)
         plt.show(ax1)

*****train heat map*****

```



*****CV heat map*****



```
In [50]: rfc=RandomForestClassifier(n_estimators=500,max_depth=5,class_weight='balanced',n_jobs=-1)
rfc.fit(tfidf_w2v_std_train_data,y_train)
```

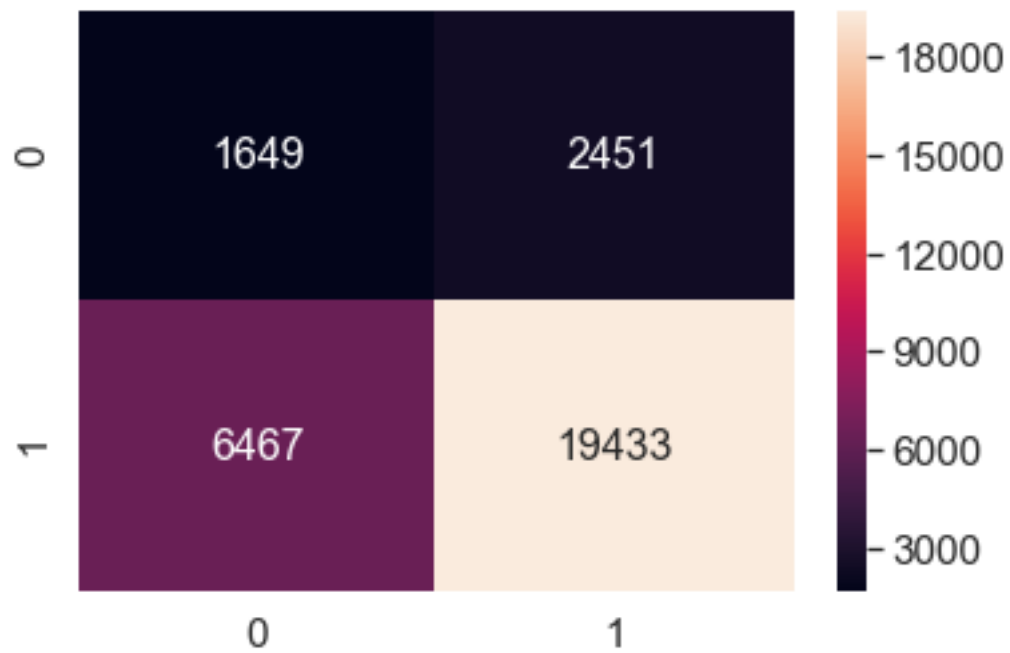
```
Out[50]: RandomForestClassifier(bootstrap=True, class_weight='balanced',
                                criterion='gini', max_depth=5, max_features='auto',
                                max_leaf_nodes=None, min_impurity_decrease=0.0,
                                min_impurity_split=None, min_samples_leaf=1,
                                min_samples_split=2, min_weight_fraction_leaf=0.0,
                                n_estimators=500, n_jobs=-1, oob_score=False,
                                random_state=None, verbose=0, warm_start=False)
```

```
In [51]: predictions=rfc.predict(standardizing.transform(tfidf_w2v_test_data))
print('accuracy = {0}'.format(rfc.score(standardizing.transform(tfidf_w2v_test_data),
precision = precision_score(y_test, predictions,pos_label=1)
Recall = recall_score(y_test, predictions,pos_label=1)
f1 = f1_score(y_test, predictions,pos_label=1)
print ('precision = {0}'.format(precision))
print ('Recall={0}'.format(Recall))
print ('f1_score={0}'.format(f1))
```

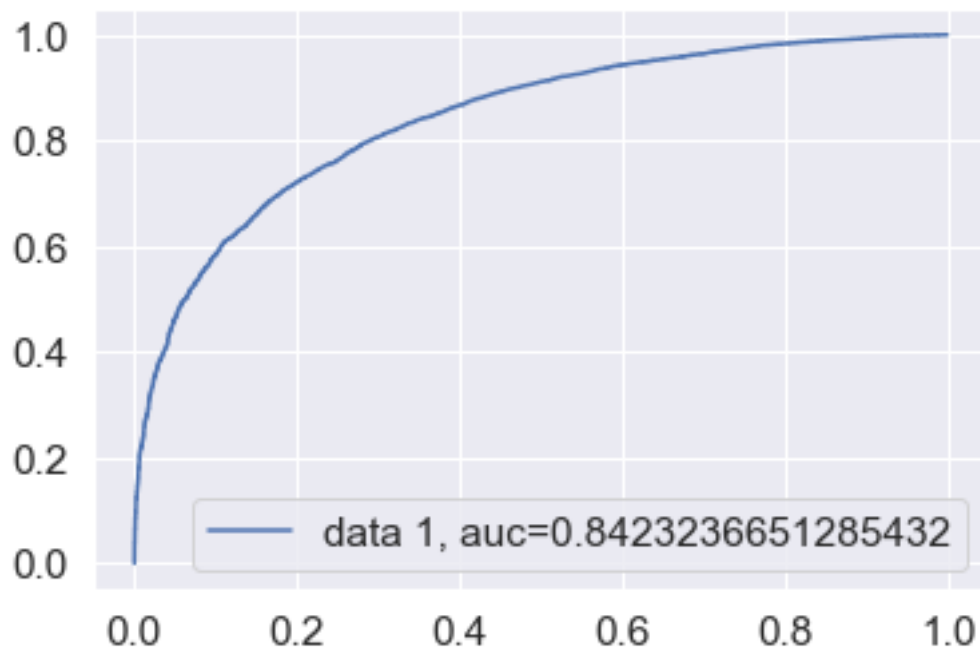
```
accuracy = 70.27333333333333
precision = 0.8880003655638823
Recall=0.7503088803088803
f1_score=0.8133684915452872
```

```
In [52]: %matplotlib inline
from sklearn.metrics import confusion_matrix
import seaborn as sns
result = confusion_matrix(y_test,predictions)
#print(result)
sns.set(font_scale=1.4)#for label size
sns.heatmap(result, annot=True,annot_kws={"size": 16}, fmt='g')
```

```
Out[52]: <matplotlib.axes._subplots.AxesSubplot at 0x127edcb00>
```



```
In [53]: y_pred_proba = rfc.predict_proba(standardizing.transform(tfidf_w2v_test_data))[:,1]
fpr, tpr, _ = roc_curve(y_test, y_pred_proba, pos_label=1 )
#auc = roc_auc_score(y_test, y_pred_proba)
auc = np.trapz(tpr,fpr)
plt.plot(fpr,tpr,label="data 1, auc="+str(auc))
plt.legend(loc=4)
plt.show()
```



11 5) Tabular form of Results

In [17]: `from prettytable import PrettyTable`

```
x = PrettyTable()

x.field_names = ['Featurization', 'depth', 'n_estimators', 'accuracy', 'AUC',
                 'precision', 'recall', 'f1_score']
x.add_row(['BOW', '10', '200', '87.6', '0.9159', '0.960', '0.8928', '0.9255'])
x.add_row(['TFIDF', '10', '200', '87.53', '0.9201', '0.9622', '0.8904', '0.925'])
x.add_row(['Avg W2V', '5', '200', '84.45', '0.823', '0.922', '0.8954', '0.9085'])
x.add_row(['TFIDF W2V', '5', '200', '70.27', '0.8423', '0.880', '0.7503', '0.813'])
print (x)
```

Featurization	depth	n_estimators	accuracy	AUC	precision	recall	f1_score
BOW	10	200	87.6	0.9159	0.960	0.8928	0.9255
TFIDF	10	200	87.53	0.9201	0.9622	0.8904	0.925
Avg W2V	5	200	84.45	0.823	0.922	0.8954	0.9085
TFIDF W2V	5	200	70.27	0.8423	0.880	0.7503	0.813

12 6) Conclusion

- 1) BOW and TfIdf values are very close
- 2) Avg w2v and tfidf w2v are performing poorly, and they are highly skewed towards positive class.
- 3) Both BOW and TFIDF are best, the difference between the various metrics is very small, we can use both techniques.