GBDT+ASSIGNMENT

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 - 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.

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

```
In [1]: 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  B000LQOCH0   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 colu
        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 [65]: # checking for Nan values along 'profilename' column
         #data[data['ProfileName'].isnull()].head()
```

```
In [66]: # 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  B000LQOCH0   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 'nega
        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  B000LQOCH0   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=Fals
```

In [64]:
```
#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 obsereved 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  A3MQ68M8KRBUQH     Claudia                      2

        HelpfulnessDenominator     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 cheaperse...
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 GBDT

7 [4.1] GBDT 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 [5]:
```
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_sta
```

In [6]:
```
# intializing for bag of words
model= CountVectorizer(dtype=float,min_df=5,ngram_range=(1,1))
final_counts= model.fit_transform(X_train)
```

In [7]:
```
#standardizing the bag of words
standardizing  = StandardScaler(with_mean = False)
final_std_data = standardizing.fit_transform(final_counts)
final_std_data.shape
```

Out[7]: (70000, 10047)

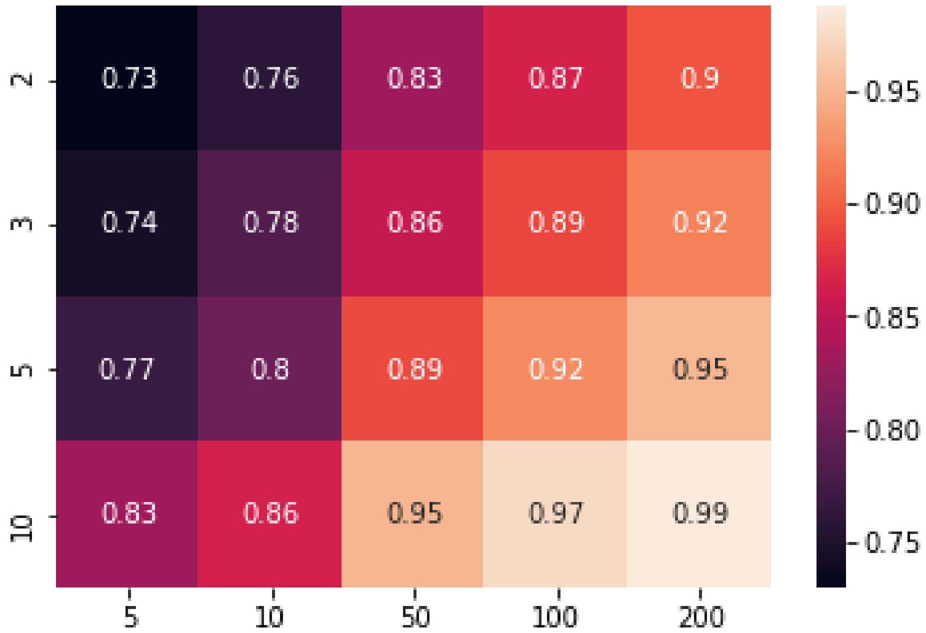
In [7]:
```
from sklearn.metrics import make_scorer
from sklearn.model_selection import cross_val_score
from sklearn.metrics import roc_auc_score
from sklearn.ensemble import GradientBoostingClassifier
# creating list for hyperparameter alpha
depth = [2,3,5,10]
estimators= [5,10,50,100,200]
# empty list 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:
        gbdt=GradientBoostingClassifier(n_estimators=k,max_depth=d)
        auc = make_scorer(roc_auc_score,greater_is_better=True,
                          needs_threshold=True)
        auc_scores = cross_val_score(gbdt, final_std_data, y_train, cv=5, scoring=auc)
        gbdt.fit(final_std_data,y_train)
        y_pred_proba = gbdt.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)
```

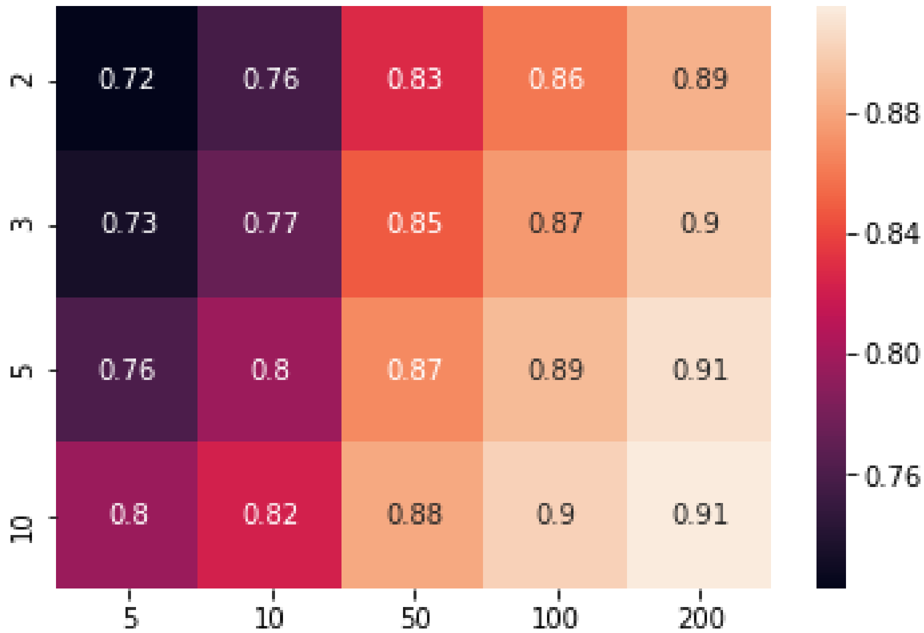
```
2
3
5
10
train data scores
[[0.7299294033013094, 0.7593547801835927, 0.83128320417395907, 0.86553233181121436, 0.89545282
**************************************************
CV scores
[[0.7222949188515625, 0.75739115407934143, 0.82816820495661003, 0.86027580668816928, 0.8856294
```

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

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



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



In [30]:
```python
from sklearn.ensemble import GradientBoostingClassifier
gbdt=GradientBoostingClassifier(n_estimators=200,max_depth=5)
gbdt.fit(final_std_data,y_train)
```

Out[30]:
```
GradientBoostingClassifier(criterion='friedman_mse', init=None,
              learning_rate=0.1, loss='deviance', max_depth=5,
              max_features=None, max_leaf_nodes=None,
              min_impurity_decrease=0.0, min_impurity_split=None,
              min_samples_leaf=1, min_samples_split=2,
              min_weight_fraction_leaf=0.0, n_estimators=200,
              presort='auto', random_state=None, subsample=1.0, verbose=0,
              warm_start=False)
```

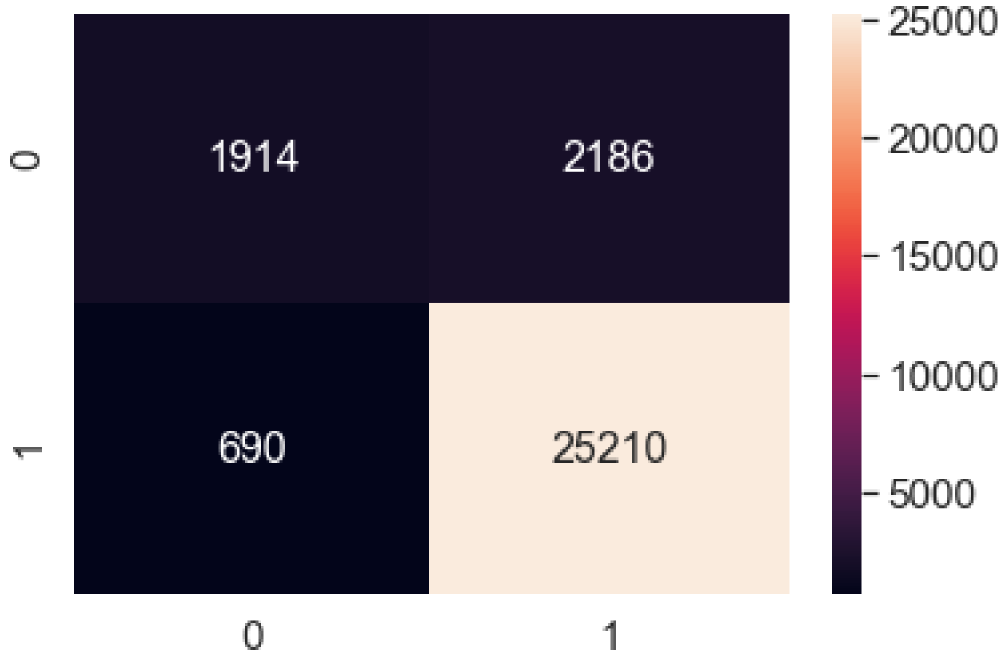
In [31]:
```python
from sklearn.metrics import confusion_matrix
from sklearn.metrics import f1_score
from sklearn.metrics import precision_score
from sklearn.metrics import recall_score

predictions=gbdt.predict(standardizing.transform(model.transform(X_test)))
print('accuracy = {0}'.format(gbdt.score(standardizing.transform(model.transform(X_tes
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))
```

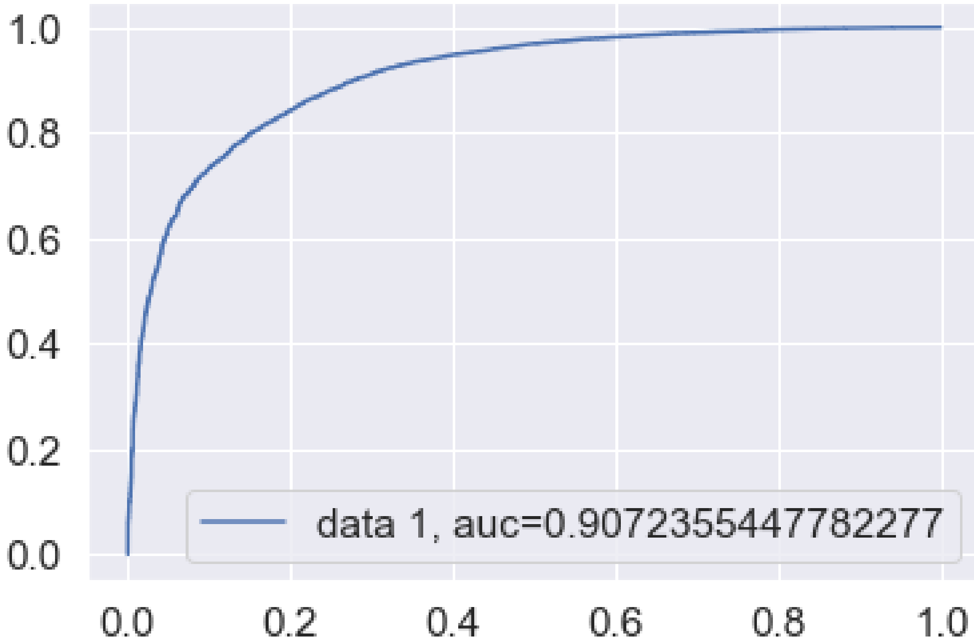
```
accuracy = 90.41333333333334
precision = 0.9202073295371587
Recall=0.9733590733590733
f1_score=0.9460372260582407
```

```
In [32]: %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[32]: <matplotlib.axes._subplots.AxesSubplot at 0x7e8ee1cc0>
```




```
In [33]: y_pred_proba = gbdt.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 [34]: feat_names = model.get_feature_names()
         feat_names = np.array(feat_names)
         # Sort feature importances in descending order
         indices = np.argsort(gbdt.feature_importances_)[::-1][:20]
         print (feat_names[indices])
```

```
['not' 'great' 'best' 'love' 'disappoint' 'good' 'delici' 'worst'
 'terribl' 'aw' 'horribl' 'product' 'nice' 'excel' 'bland' 'perfect'
 'find' 'unfortun' 'bad' 'stale']
```

```
In [35]: from wordcloud import WordCloud
         wordcloud = WordCloud(width = 400, height = 400,
                         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] GBDT on TFIDF, SET 2

```
In [20]: # 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_sta
                                                                      shuffle=False)
```

```
In [21]: vector = TfidfVectorizer(ngram_range = (1,1),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
```

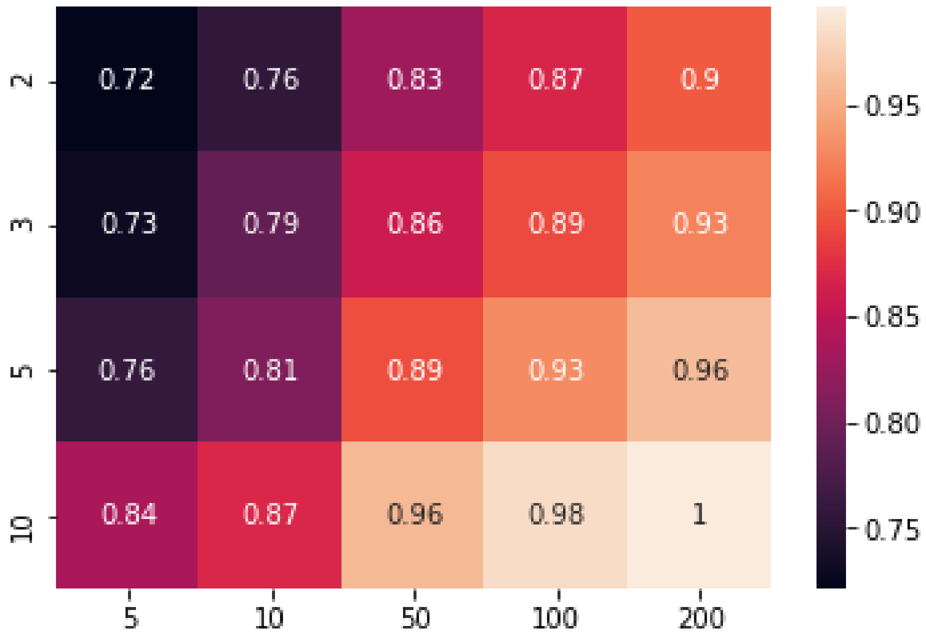
Out[21]: (70000, 10047)

In [13]:
```
from sklearn.metrics import make_scorer
from sklearn.model_selection import cross_val_score
from sklearn.metrics import roc_auc_score
from sklearn.ensemble import GradientBoostingClassifier
# creating list for hyperparameter alpha
depth = [2,3,5,10]
estimators= [5,10,50,100,200]
# 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:
        gbdt=GradientBoostingClassifier(n_estimators=k,max_depth=d)
        auc = make_scorer(roc_auc_score,greater_is_better=True,
                          needs_threshold=True)
        auc_scores = cross_val_score(gbdt, tfidf_std_data, y_train, cv=5, scoring=auc)
        gbdt.fit(tfidf_std_data,y_train)
        y_pred_proba = gbdt.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)
```

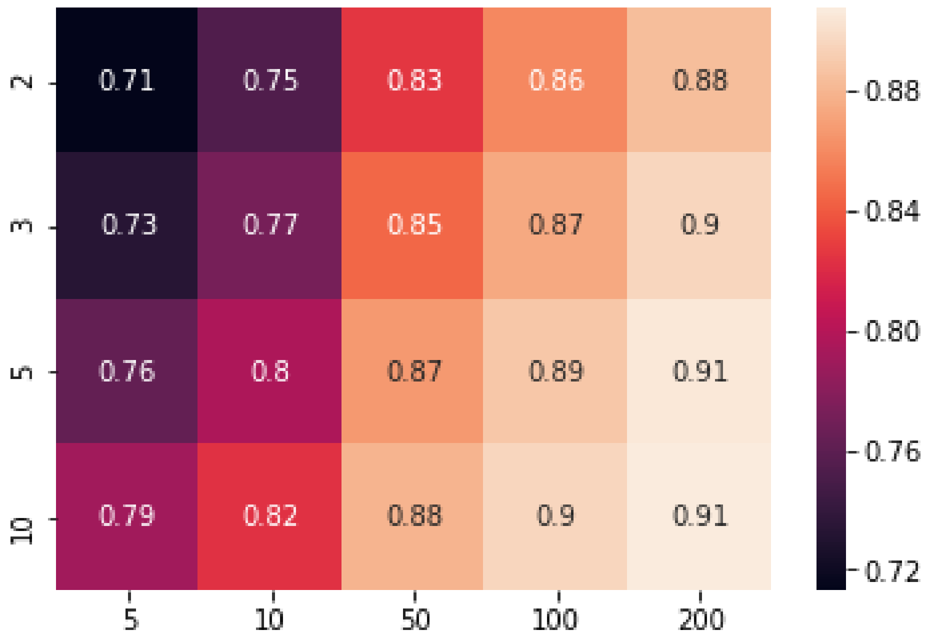
```
2
3
5
10
train data scores
[[0.72168814241916923, 0.75726764427514859, 0.83017858616956386, 0.86950389299360131, 0.900877
**************************************************
CV scores
[[0.71347058381871198, 0.75420279208271834, 0.8260059474653364, 0.85920223039830379, 0.8843726
```

```
In [14]: print ('*'*50+'train heat map'+'*'*50)
         train_values = pd.DataFrame(data=train_auc_values,index=[2,3,5,10],columns=[5,10,50,10
         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],columns=[5,10,50,100,200])
         ax1 = sns.heatmap(CV_values,annot=True)
         plt.show(ax1)
```

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




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




```
In [36]: gbdt=GradientBoostingClassifier(n_estimators=200,max_depth=3)
         gbdt.fit(tfidf_std_data,y_train)
```

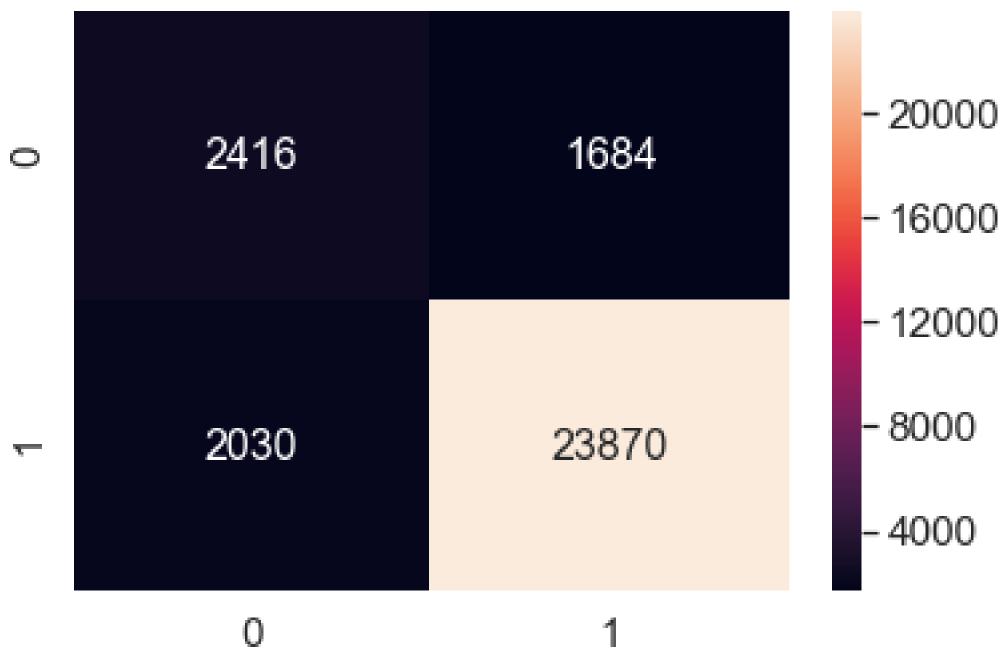
```
Out[36]: GradientBoostingClassifier(criterion='friedman_mse', init=None,
                       learning_rate=0.1, loss='deviance', max_depth=3,
                       max_features=None, max_leaf_nodes=None,
                       min_impurity_decrease=0.0, min_impurity_split=None,
                       min_samples_leaf=1, min_samples_split=2,
                       min_weight_fraction_leaf=0.0, n_estimators=200,
                       presort='auto', random_state=None, subsample=1.0, verbose=0,
                       warm_start=False)
```

```
In [37]: predictions=gbdt.predict(standardizing.transform(model.transform(tfidf_test)))
         print('accuracy = {0}'.format(gbdt.score(standardizing.transform(model.transform(tfid
         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))
```

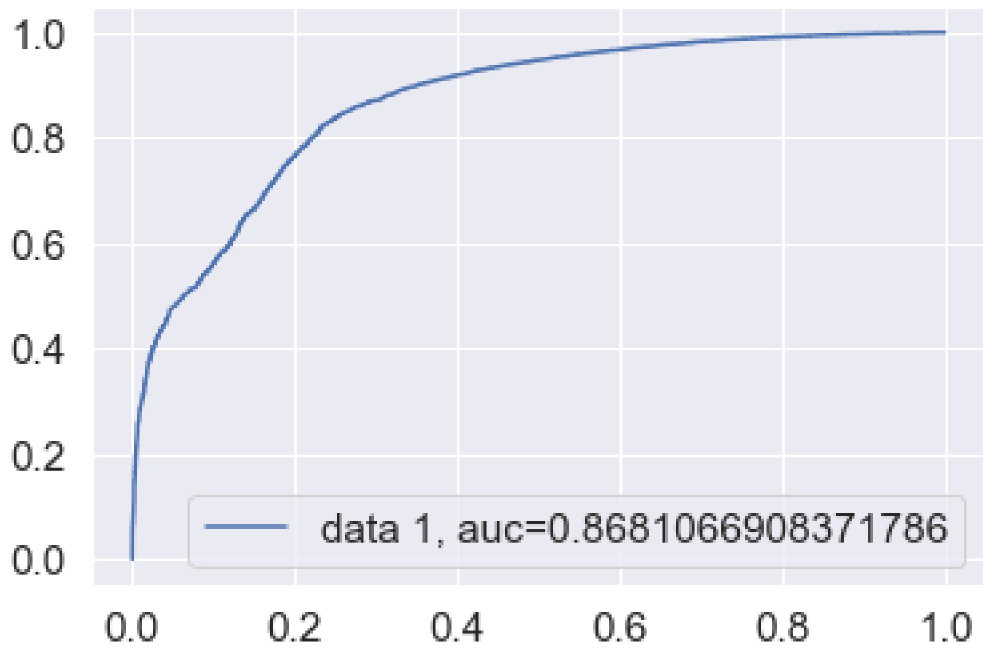
```
accuracy = 87.62
precision = 0.9341003365422244
Recall=0.9216216216216216
f1_score=0.9278190228164963
```

In [38]:
```
%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[38]: <matplotlib.axes._subplots.AxesSubplot at 0x7ec6a4470>



In [39]:
```
y_pred_proba = gbdt.predict_proba(standardizing.transform(model.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()
```



8.0.1 [4.2.1] Feature Importance

```
In [40]: feat_names = model.get_feature_names()
         feat_names = np.array(feat_names)
         # Sort feature importances in descending order
         indices = np.argsort(gbdt.feature_importances_)[::-1][:25]
         print (feat_names[indices])
```

```
['not' 'great' 'love' 'best' 'disappoint' 'good' 'worst' 'terribl' 'aw'
 'delici' 'horribl' 'perfect' 'threw' 'bland' 'stale' 'unfortun' 'return'
 'nice' 'wast' 'mayb' 'money' 'wors' 'tast' 'find' 'product']
```

```
In [41]: from wordcloud import WordCloud
         wordcloud = WordCloud(width = 400, height = 400,
                         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] GBDT on Avg W2V, SET 3

```
In [42]: 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)

In [44]: standardizing  = StandardScaler(with_mean = False)
         avg_w2v_std_train_data = standardizing.fit_transform(avg_w2v_train_data)
         avg_w2v_std_train_data.shape

Out[44]: (70000, 300)
```

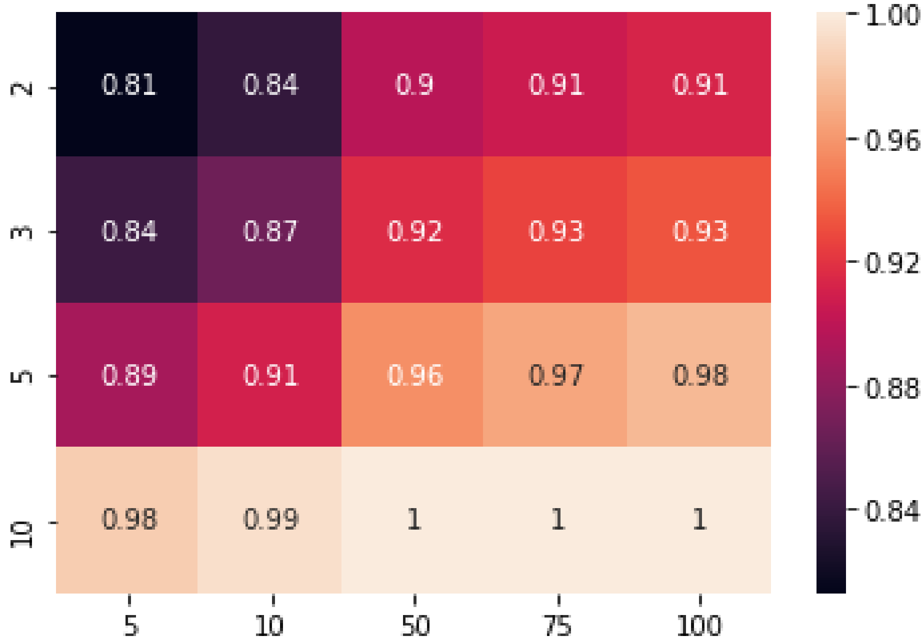
In [37]:
```
from sklearn.metrics import make_scorer
from sklearn.model_selection import cross_val_score
from sklearn.metrics import roc_auc_score
from sklearn.ensemble import GradientBoostingClassifier
# creating list for hyperparameter alpha
depth = [2,3,5,10]
estimators= [5,10,50,75,100]
# empty list 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:
        gbdt=GradientBoostingClassifier(n_estimators=k,max_depth=d)
        auc = make_scorer(roc_auc_score,greater_is_better=True,
                          needs_threshold=True)
        auc_scores = cross_val_score(gbdt, avg_w2v_std_train_data, y_train, cv=5, sco
        gbdt.fit(avg_w2v_std_train_data,y_train)
        y_pred_proba = gbdt.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
train data scores
[[0.8127245277364512l, 0.83794433359019327, 0.89766453947795055, 0.90693249505453333, 0.912863
**************************************************
CV scores
[[0.80466785866859025, 0.83179600866237235, 0.88032351835974521, 0.88782687608556887, 0.891814
```

In [40]:
```
print ('*'*50+'train heat map'+'*'*50)
train_values = pd.DataFrame(data=train_auc_values,index=[2,3,5,10],columns=[5,10,50,75
```

```
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],columns=[5,10,50,75,100])
ax1 = sns.heatmap(CV_values,annot=True)
plt.show(ax1)
```

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

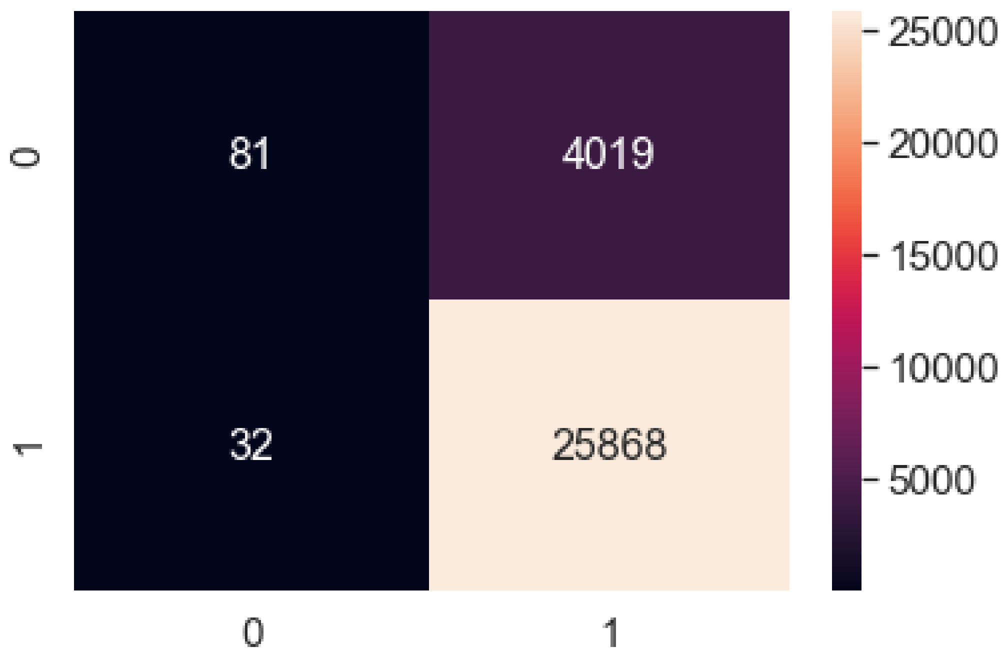



```
**************************************************CV heat map**********************************
```

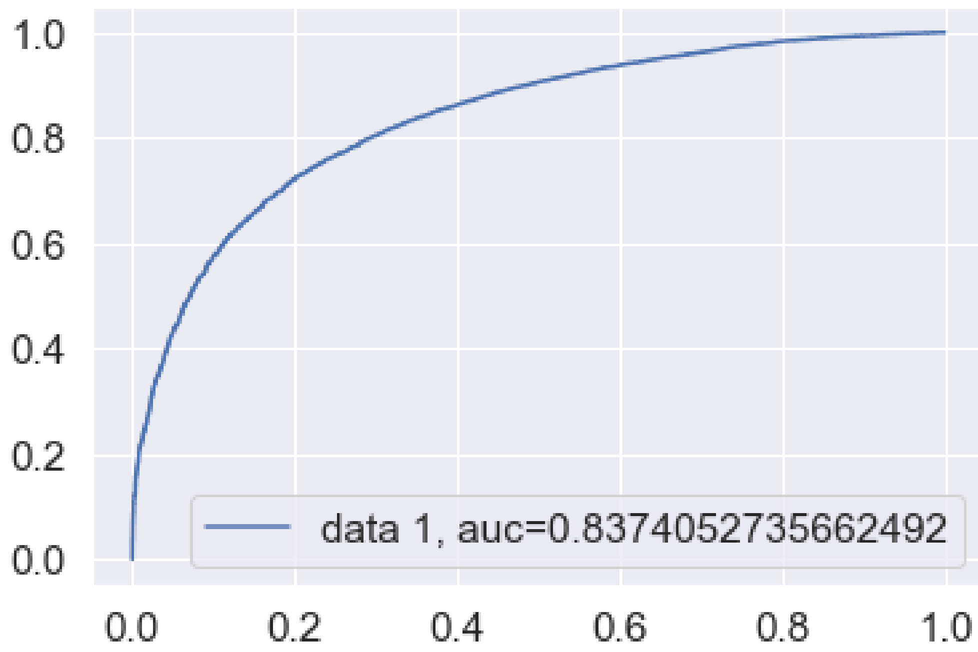

```
In [52]: gbdt=GradientBoostingClassifier(n_estimators=100,max_depth=3)
         gbdt.fit(avg_w2v_std_train_data,y_train)

Out[52]: GradientBoostingClassifier(criterion='friedman_mse', init=None,
                       learning_rate=0.1, loss='deviance', max_depth=3,
                       max_features=None, max_leaf_nodes=None,
                       min_impurity_decrease=0.0, min_impurity_split=None,
                       min_samples_leaf=1, min_samples_split=2,
                       min_weight_fraction_leaf=0.0, n_estimators=100,
                       presort='auto', random_state=None, subsample=1.0, verbose=0,
                       warm_start=False)

In [53]: predictions=gbdt.predict(standardizing.transform(avg_w2v_test_data))
         print('accuracy = {0}'.format(gbdt.score(standardizing.transform(avg_w2v_test_data), y
         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 = 86.49666666666667
precision = 0.8655268176799277
Recall=0.9987644787644787
f1_score=0.9273845161059028
```

In [54]:
```
%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[54]: <matplotlib.axes._subplots.AxesSubplot at 0x7e91dbb00>



In [55]:
```
y_pred_proba = gbdt.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] GBDT on TFIDF W2V, SET 4

In [56]:
```
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 [57]:
```
standardizing  = StandardScaler(with_mean = False)
tfidf_w2v_std_train_data = standardizing.fit_transform(tfidf_w2v_train_data)
tfidf_w2v_std_train_data.shape
```

Out[57]:
```
(70000, 300)
```

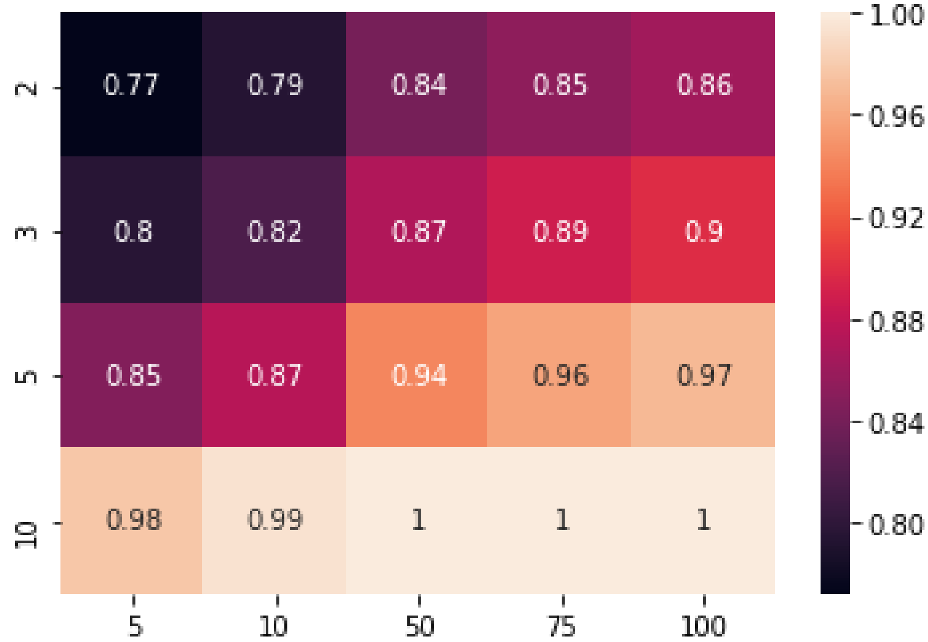
In [46]:
```
from sklearn.metrics import make_scorer
from sklearn.model_selection import cross_val_score
from sklearn.metrics import roc_auc_score
from sklearn.ensemble import GradientBoostingClassifier
# creating list for hyperparameter alpha
depth = [2,3,5,10]
estimators= [5,10,50,75,100]
# empty list 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:
            gbdt=GradientBoostingClassifier(n_estimators=k,max_depth=d)
            auc = make_scorer(roc_auc_score,greater_is_better=True,
                              needs_threshold=True)
            auc_scores = cross_val_score(gbdt, tfidf_w2v_std_train_data, y_train, cv=5, sc
            gbdt.fit(tfidf_w2v_std_train_data,y_train)
            y_pred_proba = gbdt.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)
```

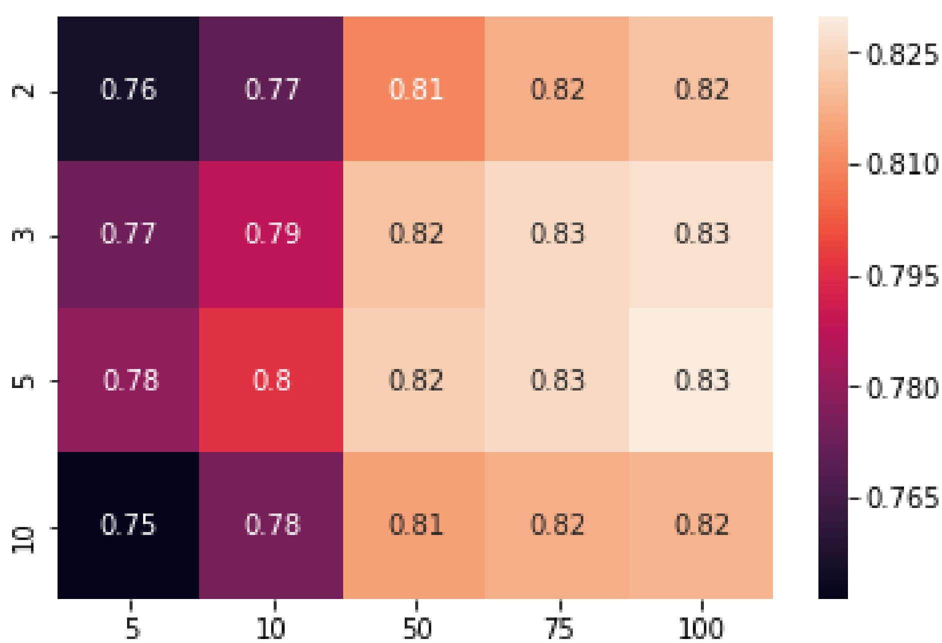
```
2
3
5
10
train data scores
[[0.77171853119908618, 0.79451127223053442, 0.84101441397664856, 0.85364216426991524, 0.863300
**************************************************
CV scores
[[0.75597395744779317, 0.77071682281610909, 0.80996906887959719, 0.8172257990953552, 0.8212145
```

In [47]:
```
print ('*'*50+'train heat map'+'*'*50)
train_values = pd.DataFrame(data=train_auc_values,index=[2,3,5,10],columns=[5,10,50,75
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],columns=[5,10,50,75,100])
ax1 = sns.heatmap(CV_values,annot=True)
plt.show(ax1)
```

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


0.77
0.79
0.84
0.85
0.86
0.8
0.82
0.87
0.89
0.9
0.85
0.87
0.94
0.96
0.97
0.98
0.99
1
1
1
2
3
5
10
5
10
50
75
100
1.00
0.96
0.92
0.88
0.84
0.80

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


0.76
0.77
0.81
0.82
0.82
0.77
0.79
0.82
0.83
0.83
0.78
0.8
0.82
0.83
0.83
0.75
0.78
0.81
0.82
0.82
2
3
5
10
5
10
50
75
100
0.825
0.810
0.795
0.780
0.765

In [59]:
```
gbdt=GradientBoostingClassifier(n_estimators=75,max_depth=3)
gbdt.fit(tfidf_w2v_std_train_data,y_train)
```

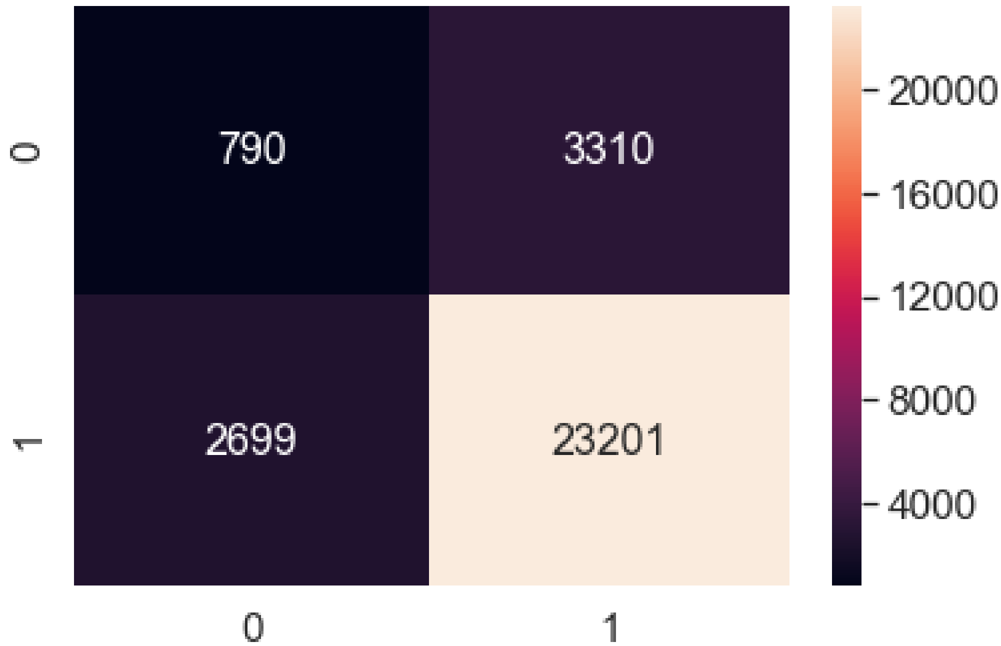
Out[59]:
```
GradientBoostingClassifier(criterion='friedman_mse', init=None,
              learning_rate=0.1, loss='deviance', max_depth=3,
              max_features=None, max_leaf_nodes=None,
              min_impurity_decrease=0.0, min_impurity_split=None,
              min_samples_leaf=1, min_samples_split=2,
              min_weight_fraction_leaf=0.0, n_estimators=75,
              presort='auto', random_state=None, subsample=1.0, verbose=0,
              warm_start=False)
```

In [60]:
```
predictions=gbdt.predict(standardizing.transform(tfidf_w2v_test_data))
print('accuracy = {0}'.format(gbdt.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))
```

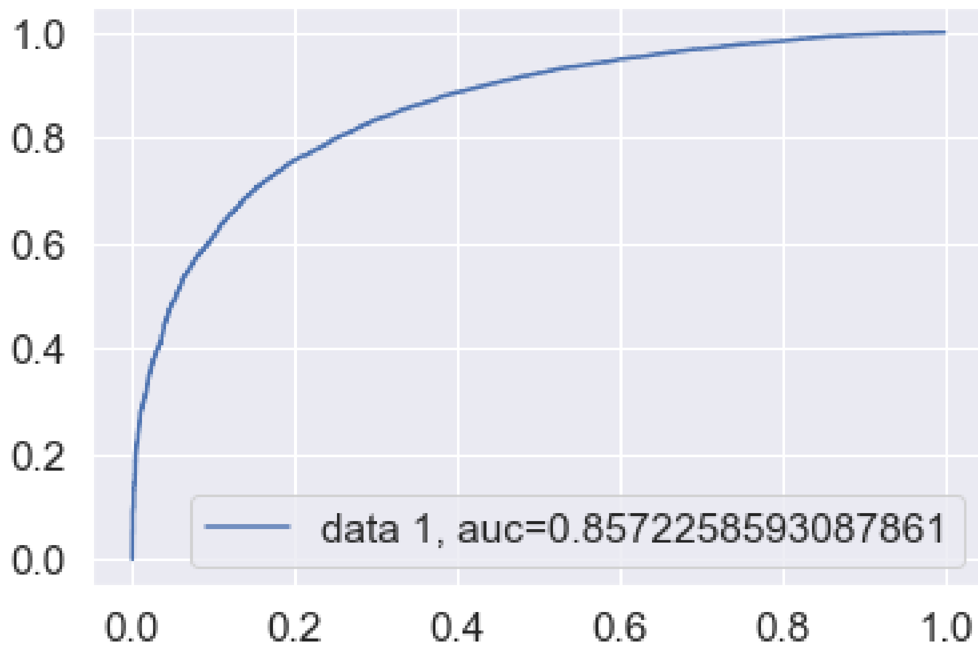
```
accuracy = 79.97
precision = 0.875146165742522
Recall=0.8957915057915058
f1_score=0.8853484955448284
```

In [61]:
```
%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[61]:
```
<matplotlib.axes._subplots.AxesSubplot at 0x7e8e6db70>
```



In [62]:
```
y_pred_proba = gbdt.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()
```



11 5) Tabular form of Results

```
In [63]: from prettytable import PrettyTable

         x = PrettyTable()

         x.field_names = ['Featurization','depth','n_estimators','accuracy','AUC',
                          'precision','recall','f1_score']
         x.add_row(['BOW','5','200','90.413','0.907','0.920','0.9733','0.946'])
         x.add_row(['TFIDF','5','200','87.62','0.868','0.934','0.9216','0.9278'])
         x.add_row(['Avg W2V','3','100','86.45','0.8374','0.8655','0.9987','0.927'])
         x.add_row(['TFIDF W2V','3','75','79.97','0.857','0.8751','0.8958','0.8853'])
         print (x)
```

```
+---------------+-------+--------------+----------+--------+-----------+--------+----------+
| Featurization | depth | n_estimators | accuracy |  AUC   | precision | recall | f1_score |
+---------------+-------+--------------+----------+--------+-----------+--------+----------+
|      BOW      |   5   |     200      |  90.413  | 0.907  |   0.920   | 0.9733 |  0.946   |
|     TFIDF     |   5   |     200      |  87.62   | 0.868  |   0.934   | 0.9216 |  0.9278  |
|    Avg W2V    |   3   |     100      |  86.45   | 0.8374 |   0.8655  | 0.9987 |  0.927   |
|   TFIDF W2V   |   3   |      75      |  79.97   | 0.857  |   0.8751  | 0.8958 |  0.8853  |
+---------------+-------+--------------+----------+--------+-----------+--------+----------+
```

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) compared to BOW has highest AUC and Recall score. So,BOW is the best featurization.