Amazon Fine Food Reviews Analysis

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

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]:
```
%matplotlib inline
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.feature_extraction.text import TfidfTransformer
from sklearn.feature_extraction.text import TfidfVectorizer

from sklearn.feature_extraction.text import CountVectorizer
from sklearn.metrics import confusion_matrix
from sklearn import metrics
from sklearn.metrics import roc_curve, auc
from nltk.stem.porter import PorterStemmer

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

from gensim.models import Word2Vec
from gensim.models import KeyedVectors
import pickle

from tqdm import tqdm
import os
```

```
C:\Users\himateja\Anaconda3\lib\site-packages\gensim\utils.py:1209: UserWarnin
g: detected Windows; aliasing chunkize to chunkize_serial
  warnings.warn("detected Windows; aliasing chunkize to chunkize_serial")
```

In [2]:
```
from sklearn.metrics import recall_score
from sklearn.metrics import precision_score
from sklearn.metrics import confusion_matrix
from sklearn.metrics import accuracy_score
from sklearn.metrics import f1_score
from sklearn.metrics import roc_auc_score as roc
import random
from sklearn.model_selection import RandomizedSearchCV
import scipy
import plotly.offline as offline
import plotly.graph_objs as go
offline.init_notebook_mode()
import numpy as np
```

In [3]:
```
# using SQLite Table to read data.
con = sqlite3.connect('database.sqlite')

filtered_data = pd.read_sql_query(""" SELECT * FROM Reviews WHERE Score != 3 LIMI

# Give reviews with Score>3 a positive rating(1), and reviews with a score<3 a ne
def partition(x):
    if x < 3:
        return 0
    return 1

#changing reviews with score less than 3 to be positive and vice-versa
actualScore = filtered_data['Score']
positiveNegative = actualScore.map(partition)
filtered_data['Score'] = positiveNegative
print("Number of data points in our data", filtered_data.shape)
filtered_data.head(3)
```

```
Number of data points in our data (200000, 10)
```

Out[3]:

	Id	ProductId	UserId	ProfileName	HelpfulnessNumerator	HelpfulnessDenominat
0	1	B001E4KFG0	A3SGXH7AUHU8GW	delmartian	1	
1	2	B00813GRG4	A1D87F6ZCVE5NK	dll pa	0	
2	3	B000LQOCH0	ABXLMWJIXXAIN	Natalia Corres "Natalia Corres"	1	

In [4]:
```
display = pd.read_sql_query("""
SELECT UserId, ProductId, ProfileName, Time, Score, Text, COUNT(*)
FROM Reviews
GROUP BY UserId
HAVING COUNT(*)>1
""", con)
```

In [5]:
```
print(display.shape)
display.head()
```

```
(80668, 7)
```

Out[5]:

	UserId	ProductId	ProfileName	Time	Score	Text	COUNT(*)
0	#oc-R115TNMSPFT9I7	B007Y59HVM	Breyton	1331510400	2	Overall its just OK when considering the price...	2
1	#oc-R11D9D7SHXIJB9	B005HG9ET0	Louis E. Emory "hoppy"	1342396800	5	My wife has recurring extreme muscle spasms, u...	3
2	#oc-R11DNU2NBKQ23Z	B007Y59HVM	Kim Cieszykowski	1348531200	1	This coffee is horrible and unfortunately not ...	2
3	#oc-R11O5J5ZVQE25C	B005HG9ET0	Penguin Chick	1346889600	5	This will be the bottle that you grab from the...	3
4	#oc-R12KPBODL2B5ZD	B007OSBE1U	Christopher P. Presta	1348617600	1	I didnt like this coffee. Instead of telling y...	2

In [6]:
```
display[display['UserId']=='AZY10LLTJ71NX']
```

Out[6]:

	UserId	ProductId	ProfileName	Time	Score	Text	COUNT(*)
80638	AZY10LLTJ71NX	B006P7E5ZI	undertheshrine "undertheshrine"	1334707200	5	I was recommended to try green tea extract to ...	5

In [7]:
```
display['COUNT(*)'].sum()
```

Out[7]:
```
393063
```

[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 [8]:
```
display= pd.read_sql_query("""
SELECT *
FROM Reviews
WHERE Score != 3 AND UserId="AR5J8UI46CURR"
ORDER BY ProductID
""", con)
display.head()
```

Out[8]:

	Id	ProductId	UserId	ProfileName	HelpfulnessNumerator	HelpfulnessDenomir
0	78445	B000HDL1RQ	AR5J8UI46CURR	Geetha Krishnan	2	
1	138317	B000HDOPYC	AR5J8UI46CURR	Geetha Krishnan	2	
2	138277	B000HDOPYM	AR5J8UI46CURR	Geetha Krishnan	2	
3	73791	B000HDOPZG	AR5J8UI46CURR	Geetha Krishnan	2	
4	155049	B000PAQ75C	AR5J8UI46CURR	Geetha Krishnan	2	

As it can be seen above that same user has multiple reviews with same values for HelpfulnessNumerator, HelpfulnessDenominator, Score, Time, Summary and Text and on doing analysis it was found that

ProductId=B000HDOPZG was Loacker Quadratini Vanilla Wafer Cookies, 8.82-Ounce Packages (Pack of 8)

ProductId=B000HDL1RQ was Loacker Quadratini Lemon Wafer Cookies, 8.82-Ounce Packages (Pack of 8) and so on

It was inferred after analysis that reviews with same parameters other than ProductId belonged to the same product just having different flavour or quantity. Hence in order to reduce redundancy it was decided to eliminate the rows having same parameters.

The method used for the same was that we first sort the data according to ProductId and then just keep the first similar product review and delelte the others. for eg. in the above just the review for ProductId=B000HDL1RQ remains. This method ensures that there is only one representative for each product and deduplication without sorting would lead to possibility of different representatives still existing for the same product.

In [9]:
```
#Sorting data according to ProductId in ascending order
sorted_data=filtered_data.sort_values('ProductId', axis=0, ascending=True, inplac
```

In [10]:
```
#Deduplication of entries
final=sorted_data.drop_duplicates(subset={"UserId","ProfileName","Time","Text"},
final.shape
```

Out[10]: (160178, 10)

In [11]:
```
#Checking to see how much % of data still remains
(final['Id'].size*1.0)/(filtered_data['Id'].size*1.0)*100
```

Out[11]: 80.089

Observation:- It was also seen that in two rows given below the value of HelpfulnessNumerator is greater than HelpfulnessDenominator which is not practically possible hence these two rows too are removed from calcualtions

In [12]:
```
display= pd.read_sql_query("""
SELECT *
FROM Reviews
WHERE Score != 3 AND Id=44737 OR Id=64422
ORDER BY ProductID
""", con)

display.head()
```

Out[12]:

	Id	ProductId	UserId	ProfileName	HelpfulnessNumerator	HelpfulnessDenomir
0	64422	B000MIDROQ	A161DK06JJMCYF	J. E. Stephens "Jeanne"	3	
1	44737	B001EQ55RW	A2V0I904FH7ABY	Ram	3	

In [13]:
```
final=final[final.HelpfulnessNumerator<=final.HelpfulnessDenominator]
```

In [14]:
```
#Before starting the next phase of preprocessing lets see the number of entries l
print(final.shape)

#How many positive and negative reviews are present in our dataset?
final['Score'].value_counts()
```

```
(160176, 10)
```

Out[14]:
```
1    134799
0     25377
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
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 [16]:
```
# remove urls from text python: https://stackoverflow.com/a/40823105/4084039
sent_0 = re.sub(r"http\S+", "", sent_0)
sent_1000 = re.sub(r"http\S+", "", sent_1000)
sent_150 = re.sub(r"http\S+", "", sent_1500)
sent_4900 = re.sub(r"http\S+", "", sent_4900)

print(sent_0)
```

```
I remembered this book from my childhood and got it for my kids.  It's just as
good as I remembered and my kids love it too.  My older daughter now reads it t
o her sister.  Good rhymes and nice pictures.
```

In [17]:
```
# https://stackoverflow.com/questions/16206380/python-beautifulsoup-how-to-remove
from bs4 import BeautifulSoup

soup = BeautifulSoup(sent_0, 'lxml')
text = soup.get_text()
print(text)
print("="*50)

soup = BeautifulSoup(sent_1000, 'lxml')
text = soup.get_text()
print(text)
print("="*50)

soup = BeautifulSoup(sent_1500, 'lxml')
text = soup.get_text()
print(text)
print("="*50)

soup = BeautifulSoup(sent_4900, 'lxml')
text = soup.get_text()
print(text)
```

```
I remembered this book from my childhood and got it for my kids.  It's just as
good as I remembered and my kids love it too.  My older daughter now reads it t
o her sister.  Good rhymes and nice pictures.
==================================================
The qualitys not as good as the lamb and rice but it didn't seem to bother his
stomach, you get 10 more pounds and it is cheaper wich is a plus for me. You ca
n always ad your own rice and veggies. Its fresher that way and better for him
in my opinion. Plus if you you can get it deliverd to your house for free its e
ven better.  Gotta love pitbulls
==================================================
This is the Japanese version of breadcrumb (pan=bread, a Portuguese loan-word,
and"ko-"  is "child of" or of "derived from".) Panko are used for katsudon, ton
katsu or cutlets served on rice or in soups. The cutlets, pounded chicken or po
rk, are coated with these light and crispy crumbs and fried. They are not gritt
y and dense like regular crumbs. They are very nice on deep fried shrimps and d
ecorative for a more gourmet touch.
==================================================
What can I say... If Douwe Egberts was good enough for my dutch grandmother, i
t's perfect for me.  I like this flavor best with my Senseo... It has a nice da
rk full body flavor without the burt bean taste I tend sense with starbucks.  I
t's a shame most americans haven't bought into single serve coffe makers as our
Dutch counter parts have.  Every cup is fresh brewed and doesn't sit long enoug
h on my desk to get that old taste either.
```

In [18]:
```
# https://stackoverflow.com/a/47091490/4084039
import re

def decontracted(phrase):
    # specific
    phrase = re.sub(r"won't", "will not", phrase)
    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
```

In [19]:
```
sent_1500 = decontracted(sent_1500)
print(sent_1500)
print("="*50)
```

```
This is the Japanese version of breadcrumb (pan=bread, a Portuguese loan-word,
and&quot;ko-&quot;  is &quot;child of&quot; or of &quot;derived from&quot;.) Pa
nko are used for katsudon, tonkatsu or cutlets served on rice or in soups. The
cutlets, pounded chicken or pork, are coated with these light and crispy crumbs
and fried. They are not gritty and dense like regular crumbs. They are very nic
e on deep fried shrimps and decorative for a more gourmet touch.
==================================================
```

In [20]:
```
#remove words with numbers python: https://stackoverflow.com/a/18082370/4084039
sent_0 = re.sub("\S*\d\S*", "", sent_0).strip()
print(sent_0)
```

```
I remembered this book from my childhood and got it for my kids.  It's just as
good as I remembered and my kids love it too.  My older daughter now reads it t
o her sister.  Good rhymes and nice pictures.
```

In [21]:
```
#remove spacial character: https://stackoverflow.com/a/5843547/4084039
sent_1500 = re.sub('[^A-Za-z0-9]+', ' ', sent_1500)
print(sent_1500)
```

```
This is the Japanese version of breadcrumb pan bread a Portuguese loan word and
quot ko quot is quot child of quot or of quot derived from quot Panko are used
for katsudon tonkatsu or cutlets served on rice or in soups The cutlets pounded
chicken or pork are coated with these light and crispy crumbs and fried They ar
e not gritty and dense like regular crumbs They are very nice on deep fried shr
imps and decorative for a more gourmet touch
```

In [23]:
```
# Combining all the above stundents
from tqdm import tqdm
preprocessed_reviews = []
# tqdm is for printing the status bar
for sentance in tqdm(final['Text'].values):
    sentance = re.sub(r"http\S+", "", sentance)
    sentance = BeautifulSoup(sentance, 'lxml').get_text()
    sentance = decontracted(sentance)
    sentance = re.sub("\S*\d\S*", "", sentance).strip()
    sentance = re.sub('[^A-Za-z]+', ' ', sentance)
    # https://gist.github.com/sebleier/554280
    sentance = ' '.join(e.lower() for e in sentance.split() if e.lower() not in s
    preprocessed_reviews.append(sentance.strip())
```

```
100%|██████████████████████████████████████████████████████████████████|
160176/160176 [00:55<00:00, 2869.16it/s]
```

In [24]:
```
preprocessed_reviews[1500]
```

Out[24]:
```
'japanese version breadcrumb pan bread portuguese loan word ko child derived pa
nko used katsudon tonkatsu cutlets served rice soups cutlets pounded chicken po
rk coated light crispy crumbs fried not gritty dense like regular crumbs nice d
eep fried shrimps decorative gourmet touch'
```

In [25]:
```
score=final.Score
```

[4] Featurization

In [27]:
```
#splitting
from sklearn.model_selection import train_test_split
x_train,x_test,y_train,y_test=train_test_split(preprocessed_reviews, score, test_
```

[4.1] BAG OF WORDS

In [28]:
```
#BoW
count_vect = CountVectorizer() #in scikit-learn
count_vect.fit(preprocessed_reviews)
print("some feature names ", count_vect.get_feature_names()[:10])
print('='*50)

final_counts = count_vect.transform(preprocessed_reviews)
print("the type of count vectorizer ",type(final_counts))
print("the shape of out text BOW vectorizer ",final_counts.get_shape())
print("the number of unique words ", final_counts.get_shape()[1])
```

```
some feature names  ['aa', 'aaa', 'aaaa', 'aaaaa', 'aaaaaaaaaaa', 'aaaaaaaaaaa
aaaa', 'aaaaaaahhhhhh', 'aaaaaaarrrrrggghhh', 'aaaaaahhhhhyaaaaaa', 'aaaaaawwww
wwwwww']
==================================================
the type of count vectorizer  <class 'scipy.sparse.csr.csr_matrix'>
the shape of out text BOW vectorizer  (160176, 75601)
the number of unique words  75601
```

In [28]:
```
from sklearn import preprocessing
```

[4.2] Bi-Grams and n-Grams.

In [29]:
```
#bi-gram, tri-gram and n-gram

#removing stop words like "not" should be avoided before building n-grams
# count_vect = CountVectorizer(ngram_range=(1,2))

count_vect = CountVectorizer(ngram_range=(1,2), min_df=10, max_features=5000)
final_bigram_counts = count_vect.fit_transform(x_train)
print("the type of count vectorizer ",type(final_bigram_counts))
print("the shape of out text BOW vectorizer ",final_bigram_counts.get_shape())
print("the number of unique words including both unigrams and bigrams ", final_bi
final_bigram_counts=preprocessing.normalize(final_bigram_counts)
bdata=count_vect.transform(x_test)
bdata=preprocessing.normalize(bdata)
```

```
the type of count vectorizer  <class 'scipy.sparse.csr.csr_matrix'>
the shape of out text BOW vectorizer  (112123, 5000)
the number of unique words including both unigrams and bigrams  5000
```

[4.3] TF-IDF

In [30]:
```
tf_idf_vect = TfidfVectorizer(ngram_range=(1,2), min_df=10)
tdata=tf_idf_vect.fit_transform(x_train)
print("some sample features(unique words in the corpus)",tf_idf_vect.get_feature_
print('='*50)
tdata=preprocessing.normalize(tdata)

tfdata = tf_idf_vect.transform(x_test)
tfdata=preprocessing.normalize(tfdata)
print("the type of count vectorizer ",type(tdata))
print("the shape of out text TFIDF vectorizer ",tdata.get_shape())
print("the number of unique words including both unigrams and bigrams ", tdata.ge
```

```
some sample features(unique words in the corpus) ['aa', 'aafco', 'aback', 'aban
don', 'abandoned', 'abc', 'abdominal', 'abdominal pain', 'abilities', 'abilit
y']
==================================================
the type of count vectorizer  <class 'scipy.sparse.csr.csr_matrix'>
the shape of out text TFIDF vectorizer  (112123, 64337)
the number of unique words including both unigrams and bigrams  64337
```

[4.4] Word2Vec

In [31]:
```
# Train your own Word2Vec model using your own text corpus
i=0
list_of_sentance=[]
for sentance in preprocessed_reviews:
    list_of_sentance.append(sentance.split())
```

In [32]:
```
is_your_ram_gt_16g=False
want_to_use_google_w2v = False
want_to_train_w2v = True

if want_to_train_w2v:
    # min_count = 5 considers only words that occured atleast 5 times
    w2v_model=Word2Vec(list_of_sentance,min_count=5,size=50, workers=4)
    print(w2v_model.wv.most_similar('great'))
    print('='*50)
    print(w2v_model.wv.most_similar('worst'))

elif want_to_use_google_w2v and is_your_ram_gt_16g:
    if os.path.isfile('GoogleNews-vectors-negative300.bin'):
        w2v_model=KeyedVectors.load_word2vec_format('GoogleNews-vectors-negative3
        print(w2v_model.wv.most_similar('great'))
        print(w2v_model.wv.most_similar('worst'))
    else:
        print("you don't have gogole's word2vec file, keep want_to_train_w2v = Tr
```

```
[('terrific', 0.8592232465744019), ('fantastic', 0.8483913540840149), ('excelle
nt', 0.8359922766685486), ('good', 0.8350746631622314), ('awesome', 0.829334735
8703613), ('wonderful', 0.7914259433746338), ('fabulous', 0.7482174634933472),
('perfect', 0.741561770439148), ('nice', 0.7256689667701721), ('amazing', 0.691
5088891983032)]
==================================================
[('greatest', 0.8197488784790039), ('nastiest', 0.7891687154769897), ('best',
0.7311190366744995), ('tastiest', 0.6881915330886841), ('terrible', 0.686642110
3477478), ('horrible', 0.666738748550415), ('disgusting', 0.6628986597061157),
('awful', 0.6502893567085266), ('closest', 0.6485698223114014), ('smoothest',
0.631527304649353)]
```

In [33]:
```
w2v_words = list(w2v_model.wv.vocab)
print("number of words that occured minimum 5 times ",len(w2v_words))
print("sample words ", w2v_words[0:50])
```

```
number of words that occured minimum 5 times  23202
sample words  ['remembered', 'book', 'childhood', 'got', 'kids', 'good', 'lov
e', 'older', 'daughter', 'reads', 'sister', 'rhymes', 'nice', 'pictures', 'love
s', 'really', 'rosie', 'books', 'introduced', 'cd', 'performed', 'king', 'als
o', 'available', 'amazon', 'birthday', 'year', 'later', 'knows', 'songs', 'fa
r', 'go', 'one', 'johnny', 'around', 'chicken', 'soup', 'w', 'rice', 'well', 'w
ritten', 'clever', 'art', 'work', 'maurice', 'sendak', 'plus', 'cheap', 'highl
y', 'recommended']
```

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

[4.4.1.1] Avg W2v

In [34]:
```
# average Word2Vec
# compute average word2vec for each review.
def avg(l):

    sent_vectors = []; # the avg-w2v for each sentence/review is stored in this l
    for sent in tqdm(l): # for each review/sentence
        sent_vec = np.zeros(50) # as word vectors are of zero length 50, you migh
        cnt_words =0; # num of words with a valid vector in the sentence/review
        for word in sent: # for each word in a review/sentence
            if word in w2v_words:
                vec = w2v_model.wv[word]
                sent_vec += vec
                cnt_words += 1
        if cnt_words != 0:
            sent_vec /= cnt_words
        sent_vectors.append(sent_vec)
    return np.matrix(sent_vectors)
```

In [35]:
```
x_1, x_test, y_1, y_test = train_test_split(list_of_sentance, score, test_size=0.
```

In [36]:
```
x_1=avg(x_1)
```

```
100%|
112123/112123 [03:54<00:00, 477.38it/s]
```

In [37]:
```
np.asarray(x_1).shape
```

Out[37]:
```
(112123, 50)
```

In [38]:
```
x_test=avg(x_test)
```

```
100%|
| 48053/48053 [01:49<00:00, 438.35it/s]
```

In [39]:
```
np.asarray(x_test).shape
```

Out[39]:
```
(48053, 50)
```

In [40]:
```
#changing nan values if any
atrain=np.nan_to_num(x_1)
atest=np.nan_to_num(x_test)
```

[4.4.1.2] TFIDF weighted W2v

In [41]:
```
# S = ["abc def pqr", "def def def abc", "pqr pqr def"]
model = TfidfVectorizer()
tf_idf_matrix = model.fit_transform(preprocessed_reviews)
# we are converting a dictionary with word as a key, and the idf as a value
dictionary = dict(zip(model.get_feature_names(), list(model.idf_)))
```

In [42]:
```
# TF-IDF weighted Word2Vec
tfidf_feat = model.get_feature_names() # tfidf words/col-names
# final_tf_idf is the sparse matrix with row= sentence, col=word and cell_val = t

def tw2v(l,d,t=tfidf_feat):
    tfidf_sent_vectors = [] # the tfidf-w2v for each sentence/review is stored in
    row=0
    for sent in tqdm(l): # for each review/sentence
        sent_vec = np.zeros(50) # as word vectors are of zero length
        weight_sum =0; # num of words with a valid vector in the sentence/review
        for word in sent: # for each word in a review/sentence
            if word in w2v_words and word in t:
                vec = w2v_model.wv[word]
                tf_idf = tf_idf_matrix[row, t.index(word)]
                # to reduce the computation we are
                # dictionary[word] = idf value of word in whole courpus
                # sent.count(word) = tf valeus of word in this review
                tf_idf = d[word]*(sent.count(word)/len(sent))
                sent_vec += (vec * tf_idf)
                weight_sum += tf_idf
        if weight_sum != 0:
            sent_vec /= weight_sum
        tfidf_sent_vectors.append(sent_vec)
        row += 1
    return np.matrix(tfidf_sent_vectors)
```

In [43]:
```
x_1, x_test, y_1, y_test = train_test_split(list_of_sentance, score, test_size=0.
```

In [44]:
```
x_1=tw2v(x_1,dictionary)
```

```
100%|
112123/112123 [2:34:44<00:00, 12.08it/s]
```

In [45]:
```
x_test=tw2v(x_test,dictionary)
```

```
100%|
| 48053/48053 [1:19:06<00:00, 10.12it/s]
```

In [46]:
```
# changing 'NaN' to numeric value
train=np.nan_to_num(x_1)
test=np.nan_to_num(x_test)
```

In [47]:
```
from sklearn.tree import DecisionTreeClassifier as dt
from sklearn.model_selection import TimeSeriesSplit
from sklearn.model_selection import GridSearchCV
```

Applying Decision Trees

[5.1] Applying Decision Trees on BOW

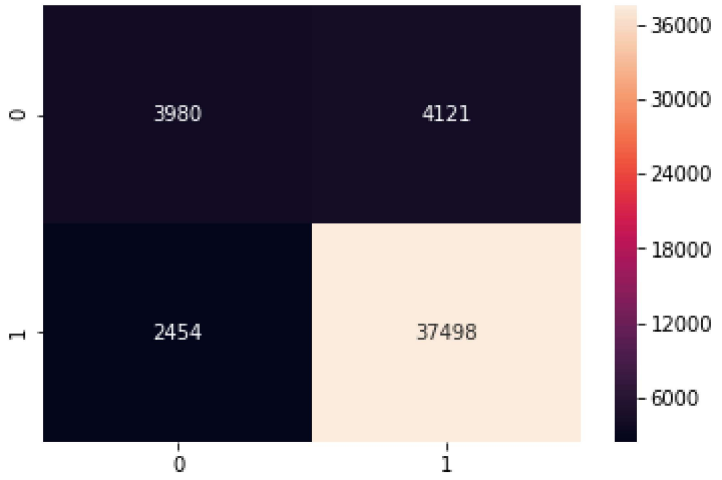
In [48]:
```
# Please write all the code with proper documentation
```

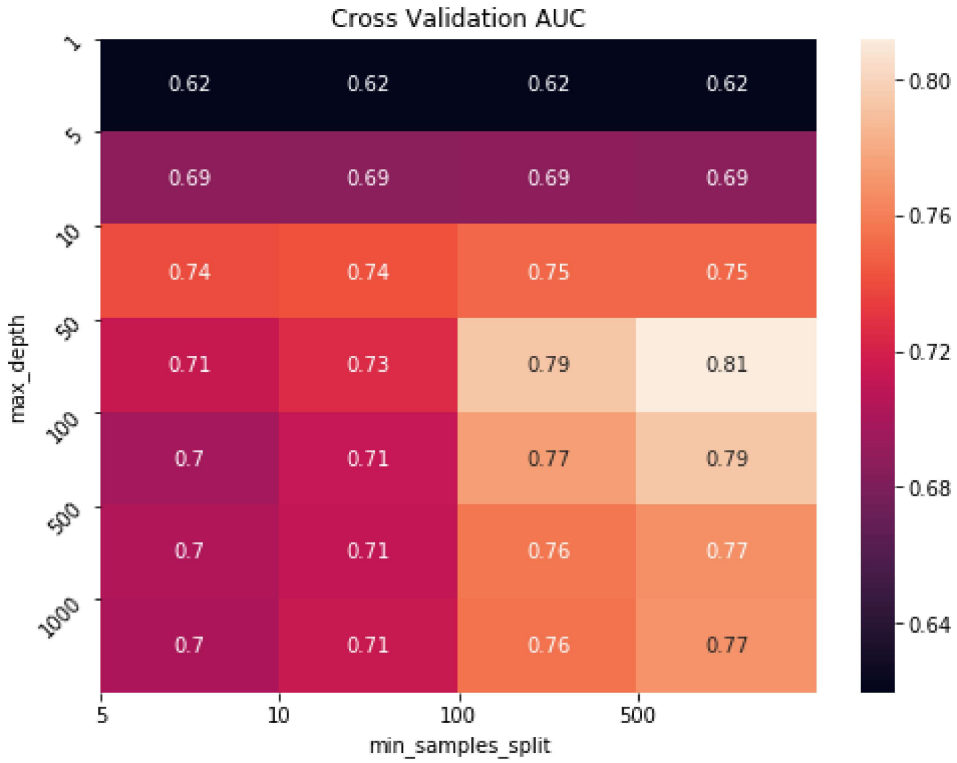
In [49]:
```
def clfg(xtrain,ytrain,xtest,ytest):

    #using the gridsearchcv for finding the best c
    clf = dt()
    max_depth=[1, 5, 10, 50, 100, 500, 1000]
    min_samples_split=[5, 10, 100, 500]
    param_grid = {'max_depth':[1, 5, 10, 50, 100, 500, 1000],'min_samples_split':
    #For time based splitting
    t = TimeSeriesSplit(n_splits=3)

    gsv = GridSearchCV(clf,param_grid,cv=t,n_jobs=-1,verbose=1,scoring="roc_auc")
    gsv.fit(xtrain,ytrain)
    print("Best HyperParameter: ",gsv.best_params_)
    #assinging best alpha to optimal print("Best Accuracy: %.2f%%"%(gsv.best_scor
    print("best estimator: ",gsv.estimator)

    od=gsv.best_params_['max_depth']
    os=gsv.best_params_['min_samples_split']

    dtclf=dt(max_depth=od,min_samples_split=os)
    dtclf.fit(xtrain,ytrain)
    pred=dtclf.predict(xtest)
    #different metrics
    acc=accuracy_score(y_test,pred)*100
    print("\nthe AUC is %.2f%%"%acc)

    re=recall_score(y_test,pred,) * 100
    print("\nthe recall is %.2f%%"%re)

    pre=precision_score(y_test,pred) * 100
    print("the precision is %.2f%%"%pre)

    f1=f1_score(y_test,pred) * 100
    print("the f1 score is %.2f%%"%f1)

    df_cm=pd.DataFrame(confusion_matrix(y_test,pred))
    #sns.set(font_scale=1.4)
    sns.heatmap(df_cm,annot=True,fmt="d")

    print('\n\n')
    plt.figure(figsize=(8, 6))
    scores = gsv.cv_results_['mean_test_score'].reshape(len(max_depth),len(min_sa
    testing = sns.heatmap(scores, annot=True)

    plt.ylabel('max_depth')
    plt.xlabel('min_samples_split')

    plt.yticks(np.arange(len(max_depth)), max_depth, rotation=45)
    plt.xticks(np.arange(len(min_samples_split)), min_samples_split)
    plt.title('Cross Validation AUC')
    plt.show()

    print('\n\n')
    plt.figure(figsize=(8, 6))
    scores = gsv.cv_results_['mean_train_score'].reshape(len(max_depth),len(min_s
    testing = sns.heatmap(scores, annot=True)
```

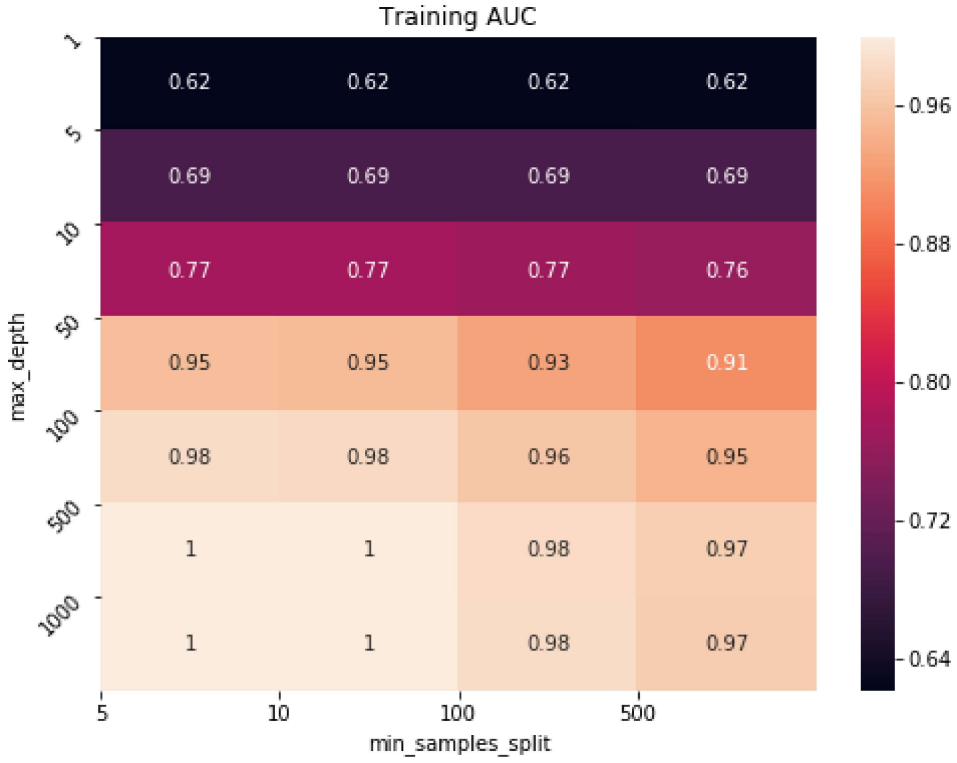
```
plt.ylabel('max_depth')
plt.xlabel('min_samples_split')

plt.yticks(np.arange(len(max_depth)), max_depth, rotation=45)
plt.xticks(np.arange(len(min_samples_split)), min_samples_split)
plt.title('Training AUC')
plt.show()

print('\n\n')
xx1=[1,1,1,1,5,5,5,5,10,10,10,10,50,50,50,50,100,100,100,100,500,500,500,500,
yy1=[5,10,100,500,5,10,100,500,5,10,100,500,5,10,100,500,5,10,100,500,5,10,10
zz1=list(gsv.cv_results_['mean_train_score'])
zz2=list(gsv.cv_results_['mean_test_score'])
# https://plot.ly/python/3d-axes/
trace1 = go.Scatter3d(x=xx1,y=yy1,z=zz1, name = 'train')
trace2 = go.Scatter3d(x=xx1,y=yy1,z=zz2, name = 'Cross validation')
data = [trace1, trace2]

layout = go.Layout(scene = dict(
    xaxis = dict(title='max_depth'),
    yaxis = dict(title='min_samples_split'),
    zaxis = dict(title='AUC'),))

fig = go.Figure(data=data, layout=layout)
offline.iplot(fig, filename='3d-scatter-colorscale')
```

In [50]:
```
%%time
clfg(final_bigram_counts,y_train,bdata,y_test)
```

```
Fitting 3 folds for each of 28 candidates, totalling 84 fits

[Parallel(n_jobs=-1)]: Done  34 tasks      | elapsed:   19.6s
[Parallel(n_jobs=-1)]: Done  84 out of  84 | elapsed:  6.9min finished

Best HyperParameter:  {'max_depth': 50, 'min_samples_split': 500}
best estimator:  DecisionTreeClassifier(class_weight=None, criterion='gini', ma
x_depth=None,
            max_features=None, max_leaf_nodes=None,
            min_impurity_decrease=0.0, min_impurity_split=None,
            min_samples_leaf=1, min_samples_split=2,
            min_weight_fraction_leaf=0.0, presort=False, random_state=None,
            splitter='best')

the AUC is 86.32%

the recall is 93.86%
the precision is 90.10%
the f1 score is 91.94%
```




Cross Validation AUC
max_depth
1
5
10
50
100
500
1000
0.62 0.62 0.62 0.62
0.69 0.69 0.69 0.69
0.74 0.74 0.75 0.75
0.71 0.73 0.79 0.81
0.7 0.71 0.77 0.79
0.7 0.71 0.76 0.77
0.7 0.71 0.76 0.77
5
10
100
500
min_samples_split
0.80
0.76
0.72
0.68
0.64


Training AUC
max_depth
1
5
10
50
100
500
1000
0.62 0.62 0.62 0.62
0.69 0.69 0.69 0.69
0.77 0.77 0.77 0.76
0.95 0.95 0.93 0.91
0.98 0.98 0.96 0.95
1 1 0.98 0.97
1 1 0.98 0.97
5
10
100
500
min_samples_split
0.96
0.88
0.80
0.72
0.64

```
Wall time: 7min 50s
```

The classfier seems to perform good as we have high TN and TP

[5.1.1] Top 20 important features from BOW

In [1]:
```
# Please write all the code with proper documentation
```

In [31]:
```
all_feat = count_vect.get_feature_names()
```

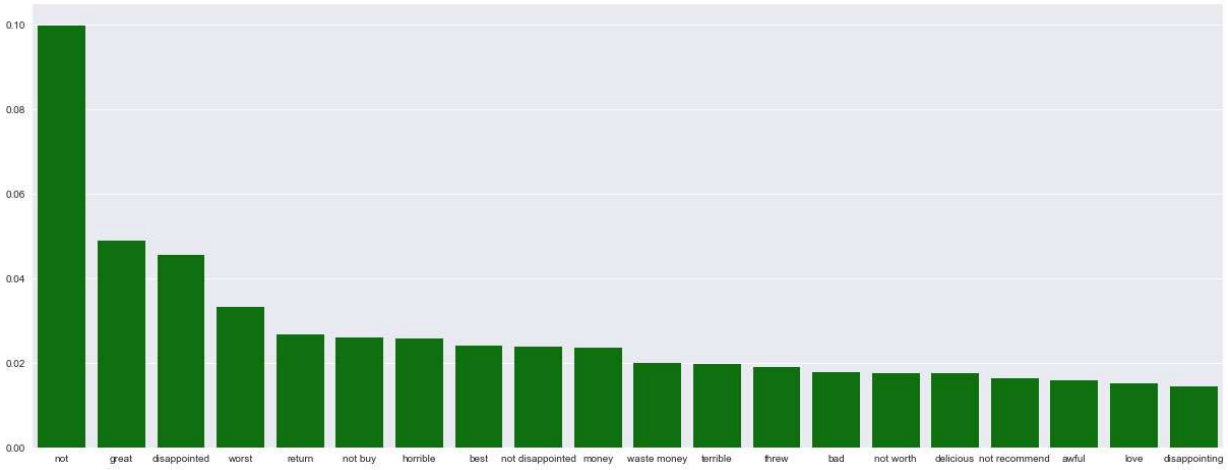
In [34]:
```
clfdt=dt(max_depth=50,min_samples_split=500)
clfdt.fit(final_bigram_counts,y_train)
```

Out[34]:
```
DecisionTreeClassifier(class_weight=None, criterion='gini', max_depth=50,
            max_features=None, max_leaf_nodes=None,
            min_impurity_decrease=0.0, min_impurity_split=None,
            min_samples_leaf=1, min_samples_split=500,
            min_weight_fraction_leaf=0.0, presort=False, random_state=None,
            splitter='best')
```

In [37]:
```
#sorting of features
importance=clfdt.feature_importances_
indices=np.argsort(importance)[::-1]
```

In [40]:
```
#taking 15 features

imp_feat = np.take(all_feat, indices[:20])
value = np.take(importance, indices[:20])
```

In [41]:
```
#plotting
sns.set()
plt.figure(figsize=(21, 8))
imp_plot = sns.barplot(imp_feat, value, color="green")
```



In [ ]:
```

```

[5.1.2] Graphviz visualization of Decision Tree on BOW

In [84]:
```
# Please write all the code with proper documentation
```

In [61]:
```
from sklearn import tree
import graphviz
```

In [64]:
```
clfdt=dt(max_depth=3)
clfdt.fit(final_bigram_counts,y_train)
all_feat=count_vect.get_feature_names()
dot_data = tree.export_graphviz(clfdt, out_file='bowtree.dot',max_depth=3,filled=
graph = graphviz.Source(dot_data)
```

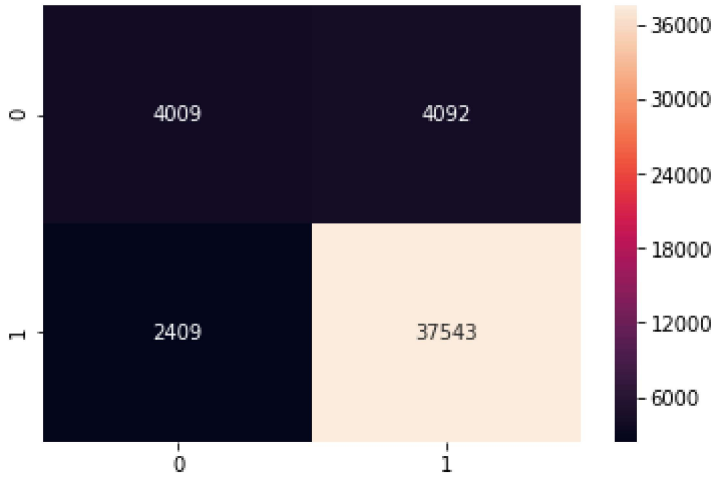
The picture is uploaded in repository.

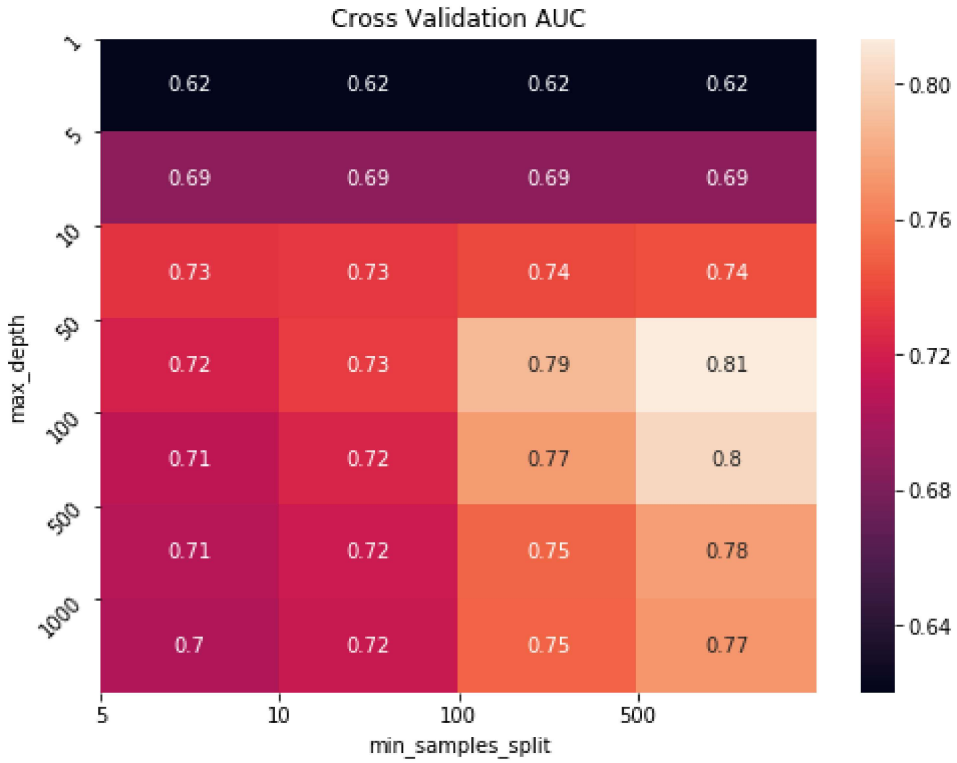
[5.2] Applying Decision Trees on TFIDF

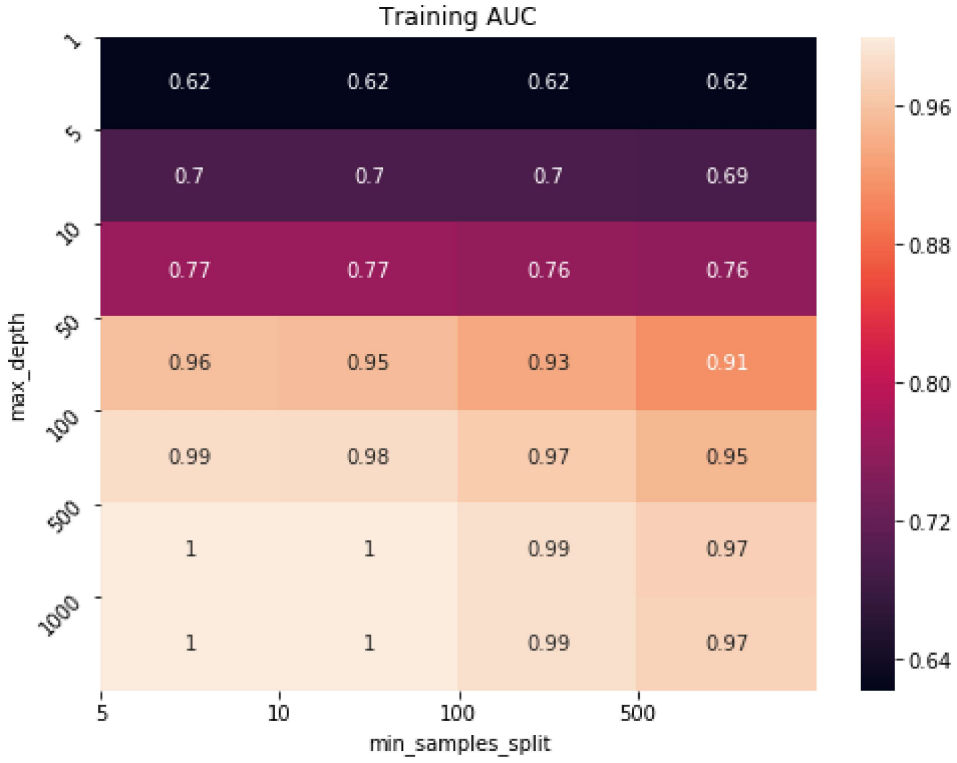
In [0]:
```
# Please write all the code with proper documentation
```

In [51]:
```
%%time
clfg(tdata,y_train,tfdata,y_test)
```

```
Fitting 3 folds for each of 28 candidates, totalling 84 fits

[Parallel(n_jobs=-1)]: Done  34 tasks      | elapsed:   36.4s
[Parallel(n_jobs=-1)]: Done  84 out of  84 | elapsed: 13.8min finished

Best HyperParameter:  {'max_depth': 50, 'min_samples_split': 500}
best estimator:  DecisionTreeClassifier(class_weight=None, criterion='gini', ma
x_depth=None,
            max_features=None, max_leaf_nodes=None,
            min_impurity_decrease=0.0, min_impurity_split=None,
            min_samples_leaf=1, min_samples_split=2,
            min_weight_fraction_leaf=0.0, presort=False, random_state=None,
            splitter='best')

the AUC is 86.47%

the recall is 93.97%
the precision is 90.17%
the f1 score is 92.03%
```




Cross Validation AUC
max_depth
1
5
10
50
100
500
1000
0.62 0.62 0.62 0.62
0.69 0.69 0.69 0.69
0.73 0.73 0.74 0.74
0.72 0.73 0.79 0.81
0.71 0.72 0.77 0.8
0.71 0.72 0.75 0.78
0.7 0.72 0.75 0.77
5
10
100
500
min_samples_split
0.80
0.76
0.72
0.68
0.64


Training AUC
max_depth
1
5
10
50
100
500
1000
0.62 0.62 0.62 0.62
0.7 0.7 0.7 0.69
0.77 0.77 0.76 0.76
0.96 0.95 0.93 0.91
0.99 0.98 0.97 0.95
1 1 0.99 0.97
1 1 0.99 0.97
5
10
100
500
min_samples_split
0.96
0.88
0.80
0.72
0.64

```
Wall time: 15min 26s
```

The classifier performs well as the TN and TP are high

[5.2.1] Top 20 important features from TFIDF

In [0]:
```
# Please write all the code with proper documentation
```

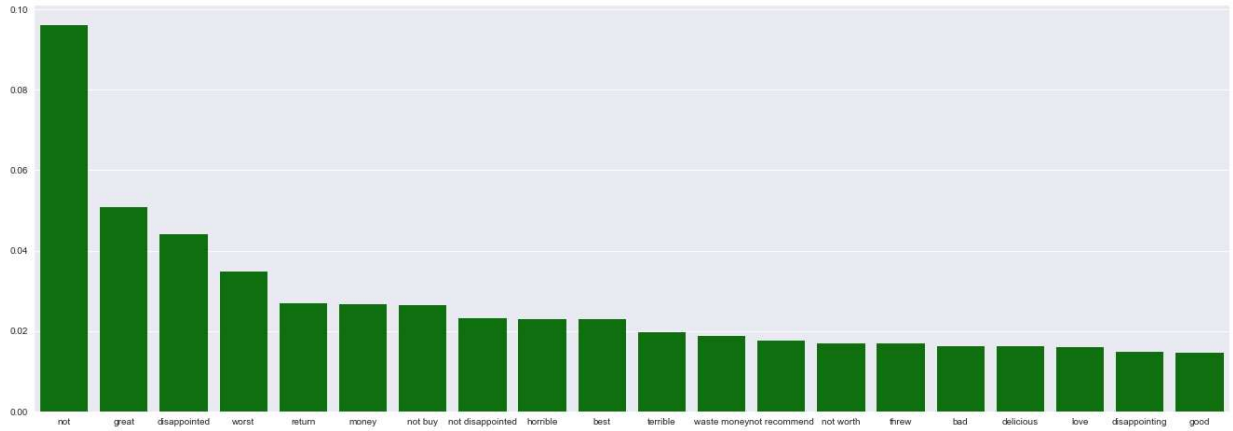
In [61]:
```
clfdt=dt(max_depth=50,min_samples_split=500)
clfdt.fit(tdata,y_train)
```

Out[61]:
```
DecisionTreeClassifier(class_weight=None, criterion='gini', max_depth=50,
            max_features=None, max_leaf_nodes=None,
            min_impurity_decrease=0.0, min_impurity_split=None,
            min_samples_leaf=1, min_samples_split=500,
            min_weight_fraction_leaf=0.0, presort=False, random_state=None,
            splitter='best')
```

In [46]:
```
#sorting the features
importance=clfdt.feature_importances_
indices=np.argsort(importance)[::-1]
all_feat=tf_idf_vect.get_feature_names()
```

In [47]:
```
#selecting 14 features
all_feat=tf_idf_vect.get_feature_names()
imp_feat = np.take(all_feat, indices[:20])
value = np.take(importance, indices[:20])
```

In [48]:
```
#plotting
sns.set()
plt.figure(figsize=(23, 8))
imp_plot = sns.barplot(imp_feat, value, color="green")
```



[5.2.2] Graphviz visualization of Decision Tree on TFIDF

In [0]:
```
# Please write all the code with proper documentation
```

In [65]:
```
clfdt=dt(max_depth=3)
clfdt.fit(tdata,y_train)
all_feat=tf_idf_vect.get_feature_names()
dot_data = tree.export_graphviz(clfdt, out_file='tfidftree.dot',max_depth=3,fille
graph = graphviz.Source(dot_data)
```

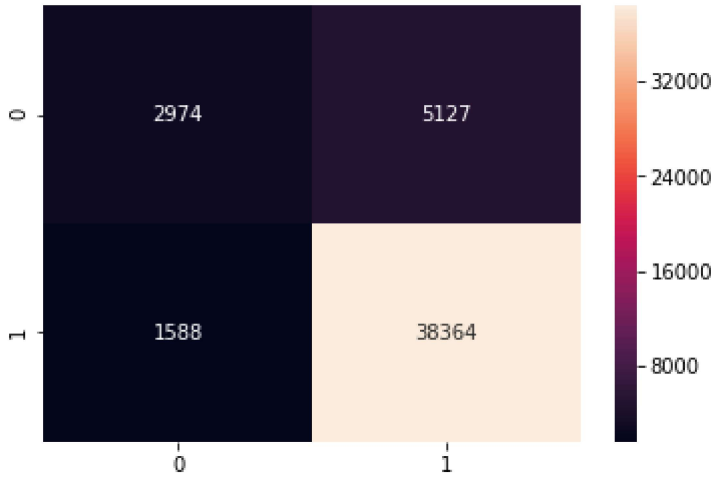
The picture is uploaded in repository.

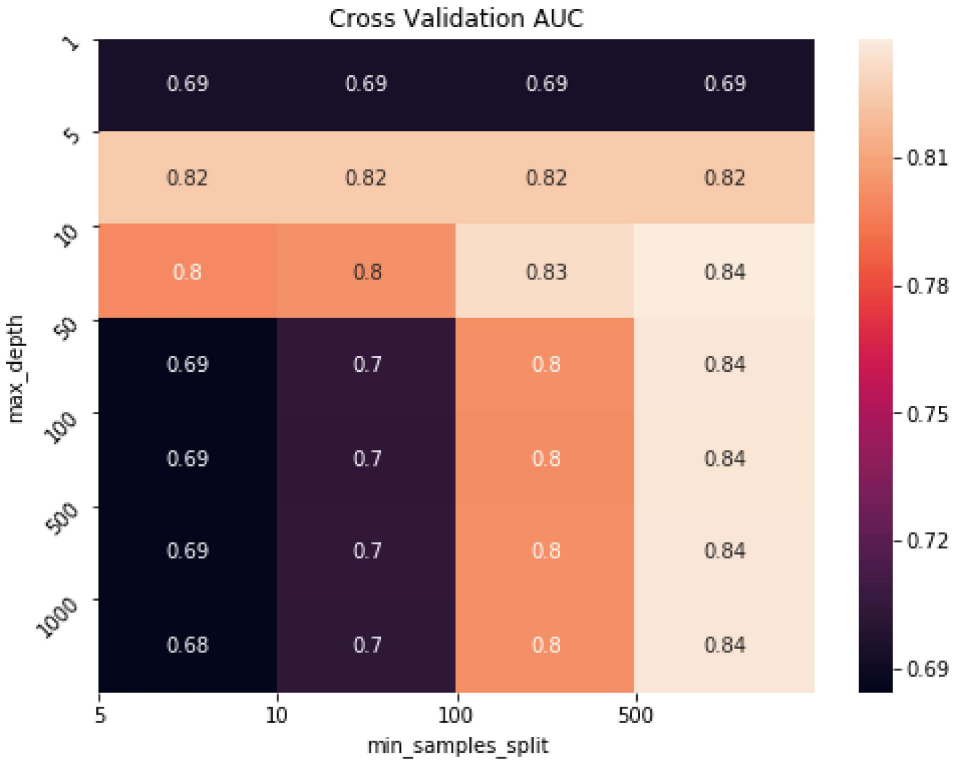
[5.3] Applying Decision Trees on AVG W2V

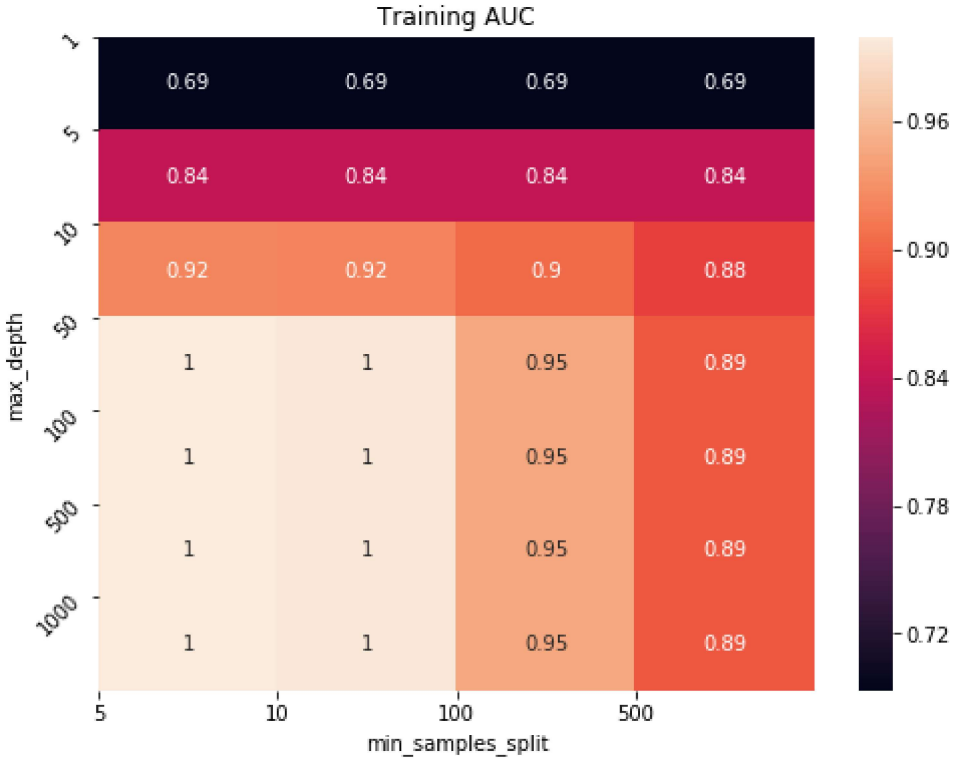
In [0]:
```
# Please write all the code with proper documentation
```

In [52]:
```
%%time
clfg(atrain,y_1,atest,y_test)
```

```
Fitting 3 folds for each of 28 candidates, totalling 84 fits

[Parallel(n_jobs=-1)]: Done  34 tasks      | elapsed:   21.2s
[Parallel(n_jobs=-1)]: Done  84 out of  84 | elapsed:  2.2min finished

Best HyperParameter:  {'max_depth': 10, 'min_samples_split': 500}
best estimator:  DecisionTreeClassifier(class_weight=None, criterion='gini', ma
x_depth=None,
            max_features=None, max_leaf_nodes=None,
            min_impurity_decrease=0.0, min_impurity_split=None,
            min_samples_leaf=1, min_samples_split=2,
            min_weight_fraction_leaf=0.0, presort=False, random_state=None,
            splitter='best')

the AUC is 86.03%

the recall is 96.03%
the precision is 88.21%
the f1 score is 91.95%
```




Cross Validation AUC
max_depth
1
5
10
50
100
500
1000
0.69 0.69 0.69 0.69
0.82 0.82 0.82 0.82
0.8 0.8 0.83 0.84
0.69 0.7 0.8 0.84
0.69 0.7 0.8 0.84
0.69 0.7 0.8 0.84
0.68 0.7 0.8 0.84
5
10
100
500
min_samples_split
0.81
0.78
0.75
0.72
0.69


Training AUC
max_depth
1
5
10
50
100
500
1000
0.69 0.69 0.69 0.69
0.84 0.84 0.84 0.84
0.92 0.92 0.9 0.88
1 1 0.95 0.89
1 1 0.95 0.89
1 1 0.95 0.89
1 1 0.95 0.89
5
10
100
500
min_samples_split
0.96
0.90
0.84
0.78
0.72

```
Wall time: 2min 26s
```

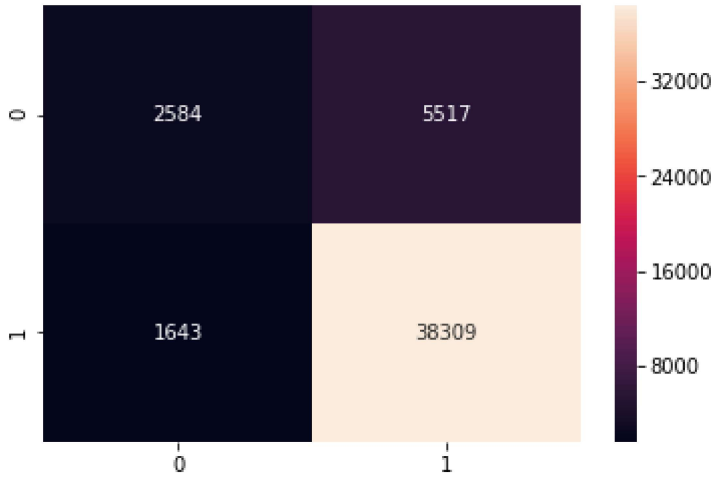
model performs relatively lower then BOW and TFIDF

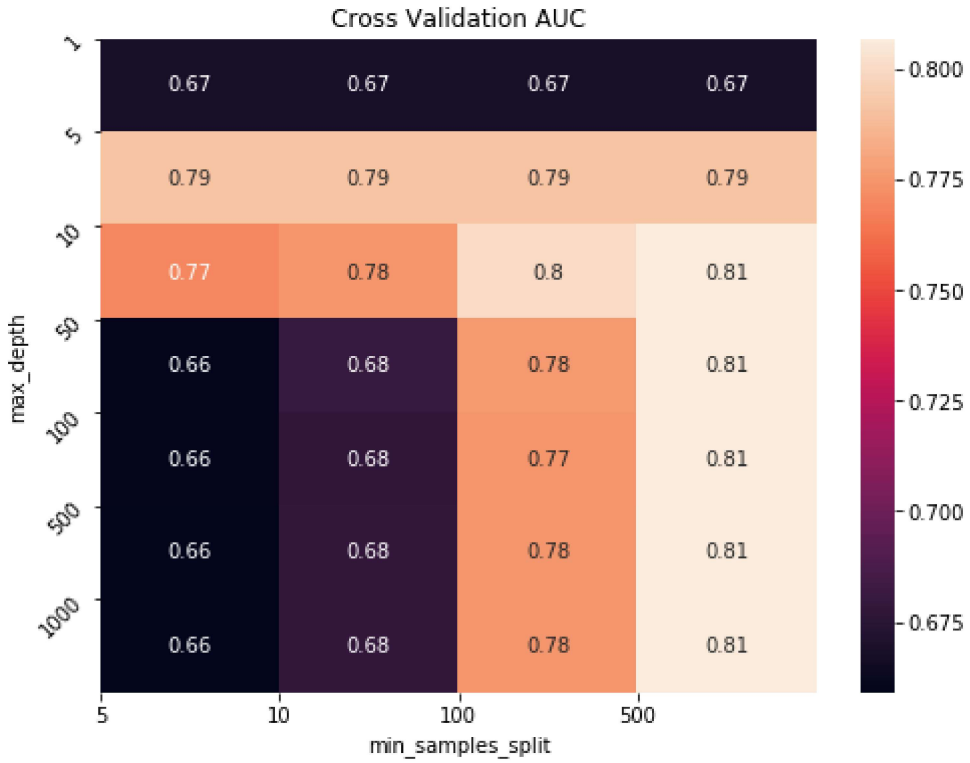
[5.4] Applying Decision Trees on TFIDF W2V

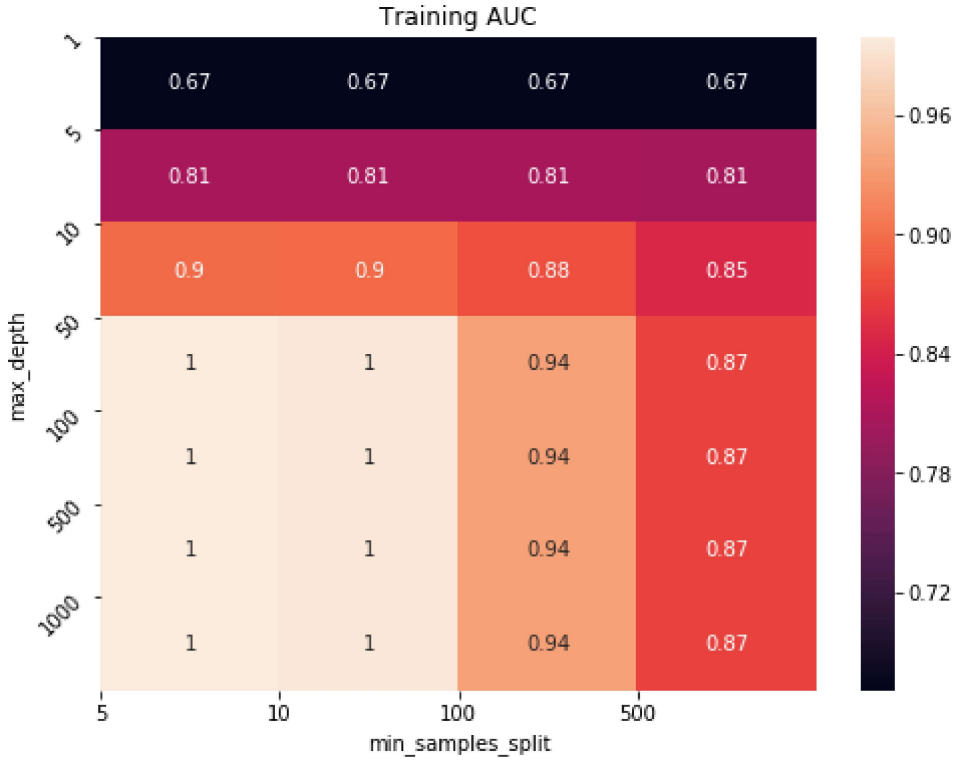
In [59]:
```
# Please write all the code with proper documentation
```

In [53]:
```
%%time
clfg(train,y_1,test,y_test)
```

```
Fitting 3 folds for each of 28 candidates, totalling 84 fits

[Parallel(n_jobs=-1)]: Done  34 tasks      | elapsed:   21.2s
[Parallel(n_jobs=-1)]: Done  84 out of  84 | elapsed:  2.3min finished

Best HyperParameter:  {'max_depth': 10, 'min_samples_split': 500}
best estimator:  DecisionTreeClassifier(class_weight=None, criterion='gini', ma
x_depth=None,
            max_features=None, max_leaf_nodes=None,
            min_impurity_decrease=0.0, min_impurity_split=None,
            min_samples_leaf=1, min_samples_split=2,
            min_weight_fraction_leaf=0.0, presort=False, random_state=None,
            splitter='best')

the AUC is 85.10%

the recall is 95.89%
the precision is 87.41%
the f1 score is 91.45%
```




Cross Validation AUC
max_depth
1
5
10
50
100
500
1000
0.67 0.67 0.67 0.67
0.79 0.79 0.79 0.79
0.77 0.78 0.8 0.81
0.66 0.68 0.78 0.81
0.66 0.68 0.77 0.81
0.66 0.68 0.78 0.81
0.66 0.68 0.78 0.81
5
10
100
500
min_samples_split
0.800
0.775
0.750
0.725
0.700
0.675


Training AUC
max_depth
1
5
10
50
100
500
1000
0.67 0.67 0.67 0.67
0.81 0.81 0.81 0.81
0.9 0.9 0.88 0.85
1 1 0.94 0.87
1 1 0.94 0.87
1 1 0.94 0.87
1 1 0.94 0.87
5
10
100
500
min_samples_split
0.96
0.90
0.84
0.78
0.72

```
Wall time: 2min 32s
```

model performs relatively lower than the BOW and TFIDF

[6] Conclusions

In [0]:
```
# Please compare all your models using Prettytable library
```

In [54]:
```
from prettytable import PrettyTable
```

In [55]:
```
x=PrettyTable()
x.field_names = ['feature_set','max_depth','min_samples_split','AUC']
```

In [56]:
```
x.add_row(["BOW",'50','500','86.32%'])
x.add_row(["TFIDF",'50','500','86.47%'])
x.add_row(["AVG_W2v",'10','500','86.03%'])
x.add_row(["TFIDF_w2V",'10','500','85.10%'])
```

In [57]:
```
print(x)
```

```
+-------------+-----------+-------------------+--------+
| feature_set | max_depth | min_samples_split |  AUC   |
+-------------+-----------+-------------------+--------+
|     BOW     |     50    |        500        | 86.32% |
|    TFIDF    |     50    |        500        | 86.47% |
|   AVG_W2v   |     10    |        500        | 86.03% |
|  TFIDF_w2V  |     10    |        500        | 85.10% |
+-------------+-----------+-------------------+--------+
```

Decision tree performs well on the dataset and it is comparitively much faster than some models.

In [ ]:
```

```