

Amazon Fine Food Reviews Analysis

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

EDA: <https://nycdatascience.com/blog/student-works/amazon-fine-foods-visualization/> (<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: 74,268
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.

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

```
In [247]: %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 matplotlib.pyplot as sns
from sklearn.feature_extraction.text import TfidfTransformer
from sklearn.feature_extraction.text import TfidfVectorizer

from sklearn.feature_extraction.text import CountVecorizer
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
```

```
In [248]: # using SQLite Table to read data.
con = sqlite3.connect('database.sqlite')

# filtering only positive and negative reviews i.e.
# not taking into consideration those reviews with Score=3
# SELECT * FROM Reviews WHERE Score != 3 LIMIT 500000, will give top 500000 data points
# you can change the number to any other number based on your computing power

# filtered_data = pd.read_sql_query(""" SELECT * FROM Reviews WHERE Score != 3 LIMIT 500000""", con)
# for tsne assignment you can take 5k data points

filtered_data = pd.read_sql_query(""" SELECT * FROM Reviews WHERE Score != 3 LIMIT 20000""", con)

# Give reviews with Score>3 a positive rating(1), and reviews with a score<3 a negative rating(0).
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 (20000, 10)
```

Out[248]:

	Id	ProductId	UserId	ProfileName	HelpfulnessNumerator	HelpfulnessDenominator	Score	Time	Summary	Text
0	1	B001E4KFG0	A3SGXH7AUHU8GW	deimartian	1	1	1	1303862400	Good Quality Dog Food	I have bought several of the Vitality earned d...
1	2	B00813GRG4	A1D87F6ZCVE5NK	dli pa	0	0	0	1346976000	Not as Advertised	Product arrived labeled as Jumbo Salted Peanut...
2	3	B000LQOCH0	ABXLMWJUXKAIN	Natalia Corres 'Natalia Corres'	1	1	1	1219017600	"Delight" says it all	This is a confection that has been around a fe...

```
In [249]: display = pd.read_sql_query("""
SELECT UserId, ProductId, ProfileName, Time, Score, Text, COUNT(*)
FROM Reviews
GROUP BY UserId
HAVING COUNT(*)>1
""", con)
```

```
In [250]: print(display.shape)
display.head()

(80668, 7)
```

Out[250]:

		UserId	ProductId	ProfileName	Time	Score	Text	COUNT(*)
0	#oc-R1151TNMSPFT9I7	B007Y59HVM	Breyton	1331510400	2		Overall its just OK when considering the price...	2
1	#oc-R11D9D7SHXUB9	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 [251]: display[display['UserId']=='AZY18LLTJ71NX']
```

Out[251]:

	UserId	ProductId	ProfileName	Time	Score	Text	COUNT(*)
80638	AZY10LLTJ71NX	B006P7E5Z1	undertheshrine "undertheshrine"	1334707200	5	I was recommended to try green tea extract to ...	5

```
In [252]: display['COUNT(*)'].sum()

Out[252]: 393963
```

[2] Exploratory Data Analysis

[2.1] Data Cleaning: Deduplication

It is observed (as shown in the table below) that the reviews have 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 [253]: display= pd.read_sql_query("""
SELECT *
FROM Reviews
WHERE Score != 3 AND UserId="AR5J8UI46CURR"
ORDER BY ProductID
""", con)
display.head()
```

Out[253]:

	Id	ProductId	UserId	ProfileName	HelpfulnessNumerator	HelpfulnessDenominator	Score	Time	Summary	Text
0	78445	B000HDL1RQ	ARSJ8UI46CURR	Geetha Krishnan	2	2	5	1199577600	LOACKER QUADRATINI VANILLA WAFERS	DELICIOUS WAFERS. I FIND THAT EUROPEAN WAFERS ...
1	138317	B000HDOPLYC	ARSJ8UI46CURR	Geetha Krishnan	2	2	5	1199577600	LOACKER QUADRATINI VANILLA WAFERS	DELICIOUS WAFERS. I FIND THAT EUROPEAN WAFERS ...
2	138277	B000HDOPLYM	ARSJ8UI46CURR	Geetha Krishnan	2	2	5	1199577600	LOACKER QUADRATINI VANILLA WAFERS	DELICIOUS WAFERS. I FIND THAT EUROPEAN WAFERS ...
3	73791	B000HDOPEG	ARSJ8UI46CURR	Geetha Krishnan	2	2	5	1199577600	LOACKER QUADRATINI VANILLA WAFERS	DELICIOUS WAFERS. I FIND THAT EUROPEAN WAFERS ...
4	155049	B000PAQ75C	ARSJ8UI46CURR	Geetha Krishnan	2	2	5	1199577600	LOACKER QUADRATINI VANILLA WAFERS	DELICIOUS WAFERS. I FIND THAT EUROPEAN WAFERS ...

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=B000HDOPEG 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 delete 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 [254]: #Sorting data according to ProductId in ascending order
sorted_data=filtered_data.sort_values('ProductId', axis=0, ascending=True, inplace=False, kind='quicksort', na_position='last')
```

```
In [255]: #Deduplication of entries
final=sorted_data.drop_duplicates(subset=["UserId","ProfileName","Time","Text"], keep='first', inplace=False)
final.shape

Out[255]: (19354, 10)
```

```
In [256]: #Checking to see how much % of data still remains
(final['Id'].size*1.0)/(filtered_data['Id'].size*1.0)*100

Out[256]: 96.77
```

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 calculations

```
In [257]: display= pd.read_sql_query("""
SELECT *
FROM Reviews
WHERE Score != 3 AND Id=44737 OR Id=64422
ORDER BY ProductID
""", con)
display.head()

Out[257]:
```

	Id	ProductId	UserId	ProfileName	HelpfulnessNumerator	HelpfulnessDenominator	Score	Time	Summary	Text
0	64422	B000MIDROQ	A161DK06JJCXYF	J. E. Stephens 'Jeanne'	3	1	5	1224892800	Bought This for My Son at College	My son loves spaghetti so I didn't hesitate or...
1	44737	B001EQ55RW	A2V0I904FH7ABY	Ram	3	2	4	1212883200	Pure cocoa taste with crunchy almonds inside	It was almost a 'love at first bite' - the per...

```
In [258]: final=final[(final.HelpfulnessNumerator<=final.HelpfulnessDenominator)]
```

```
In [259]: #Before starting the next phase of preprocessing Lets see the number of entries left
          print(final.shape)

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

          (19354, 10)

Out[259]: 1    16339
          0     3015
          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 - # 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

```
[260]: # printing some random reviews
sent_0 = final['Text'].values[0]
print(sent_0)
print("-"*50)

sent_1000 = final['Text'].values[1000]
print(sent_1000)
print("-"*50)

sent_1500 = final['Text'].values[1500]
print(sent_1500)
print("-"*50)

sent_4900 = final['Text'].values[4900]
print(sent_4900)
print("-"*50)

We have used the Victor fly bait for 3 seasons. Can't beat it. Great product!
=====
I received this box with great anticipation since they don't sell these on the west coast. I got the package, opened the box and was EXTREMELY disappointed. The cookies looked like a gorilla shook the box to death and left most of the box filled with crumbs. AND THERE WAS A RODENT SIZED HOLE ON THE SIDE OF THE BOX!!!!!! So, needless to say I will not NOT be reordering these again.
=====
I have two cats. My big boy has eaten these and never had a problem...as a matter of fact he has never vomited or had a hair ball since I adopted him at 2 months. My girl cat throws up every time she eats this particular flavor. Since I treat them equally these are no longer purchased. I hate to see my girl sick so I just recommend you watch your cats after you give them these treats. If not a problem...carry on.
=====
I was always a fan of Dave's, so I bought this at a local store to try Blair's and I'm glad I did. The jalapeno sauce is very mild (for me) but one of the most delicious condiments I've ever tasted. The Afterdeath is a bit painful, but still very tasty on rice & beans, burritos, or any chicken dish I've tried it on. The Sudden Death kicked my ass when I underestimated it, but now a few drops in a dish or pot are just right if I want heat without changing flavor much.
=====
```

```
[261]: # 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)

We have used the Victor fly bait for 3 seasons. Can't beat it. Great product!
```

```
[262]: # https://stackoverflow.com/questions/16206380/python-beautifulsoup-how-to-remove-all-tags-from-an-element
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)

We have used the Victor Fly bait for 3 seasons. Can't beat it. Great product!
```

I received this box with great anticipation since they don't sell these on the west coast. I got the package, opened the box and was EXTREMELY disappointed. The cookies looked like a gorilla shook the box to death and left most of the box filled with crumbs. AND THERE WAS A RODENT SIZED HOLE ON THE SIDE OF THE BOX!!!!!!! So, needless to say I will not NOT be reordering these again.

=====

I have two cats. My big boy has eaten these and never had a problem...as a matter of fact he has never vomited or had a hair ball since I adopted him at 2 months. My girl cat throws up every time she eats this particular flavor. Since I treat them equally these are no longer purchased. I hate to see my girl sick so I just recommend you watch your cats after you give them these treats. If not a problem...carry on.

=====

I was always a fan of Dave's, so I bought this at a local store to try Blair's and I'm glad I did. The jalapeno sauce is very mild (for me) but one of the most delicious condiments I've ever tasted. The Afterdeath is a bit painful, but still very tasty on rice & beans, burritos, or any chicken dish I've tried it on. The Sudden Death kicked my ass when I underestimated it, but now a few drops in a dish or pot are just right if I want heat without changing flavor much.

```
[1263]: # https://stackoverflow.com/a/47891498/4884839
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
```

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

I have two cats. My big boy has eaten these and never had a problem...as a matter of fact he has never vomited or had a hair ball since I adopted him at 2 months. My girl cat throws up every time she eats this particular flavor. Since I treat them equally these are no longer purchased. I hate to see my girl sick so I just recommend you watch your cats after you give them these treat
ts. If not a problem...carry on.
=====
```

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

We have used the Victor fly bait for seasons. Can't beat it. Great product!
```

```
[266]: #remove special character: https://stackoverflow.com/a/5843547/4084039
sent_1500 = re.sub('[\A-Za-z0-9]*', ' ', sent_1500)
print(sent_1500)
```

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

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

[illegible]

Out[269]: 'two cats big boy eaten never problem matter fact never vomited hair ball since adopted months girl cat throws every time eats particular flavor since treat equally no longer purchased hate see girl sick recommend watch cats give treats not problem carry'

[3.2] Preprocessing Review Summary

```
In [271]: from sklearn.cross_validation import train_test_split
X_1, Y_test, X_1_cv, Y_cv = train_test_split(X_1, Y_1, test_size=0.3, random_state=0)
X_tr, X_cv, Y_tr, Y_cv = train_test_split(X_1, Y_1, test_size=0.3, random_state=0)
```

```
In [272]: ## Similarly you can do preprocessing for review summary also.
```

[4] Featurization

[4.1] BAG OF WORDS

```
[273]: ##Bow
count_vect = CountVectorizer( min_df=20, max_df=50) #in scikit-Learn
count_vect.fit(X_tr)
print("some feature names ", count_vect.get_feature_names()[:10])
print("-"*50)

X_Bow_Tr = count_vect.transform(X_tr)
X_Bow_Cv = count_vect.transform(X_cv)
X_Bow_Test = count_vect.transform(X_test)

print("the type of count vectorizer ",type(X_Bow_Tr))
print("the shape of out text Bow vectorizer ",X_Bow_Tr.get_shape())
print("the number of unique words ", X_Bow_Tr.get_shape()[1])

some feature names  ['absolute', 'according', 'acidic', 'active', 'actual', 'addictive', 'additives', 'adult', 'adults', 'afford']
=====
the type of count vectorizer <class 'scipy.sparse.csr.csr_matrix'>
the shape of out text Bow vectorizer (5482, 1119)
the number of unique words 1119
```

```
In [274]: Row_Feature = count_vect.get_feature_names()
          X_Row_Tr = X_Row_Tr.toarray()
          X_Row_Cv = X_Row_Cv.toarray()
          X_Row_Test = X_Row_Test.toarray()
```

[4.3] TF-IDF


```
In [275]: tf_idf_vect = TfidfVectorizer(ngram_range=(1,2), min_df=20, max_df=50)
tf_idf_vect.fit(X_tr)
print("some sample features(unique words in the corpus)",tf_idf_vect.get_feature_names()[0:10])
print("-"*50)

X_Tfidf_Tr = tf_idf_vect.transform(X_tr)
X_Tfidf_Cv = tf_idf_vect.transform(X_cv)
X_Tfidf_Test = tf_idf_vect.transform(X_test)

print("the type of count vectorizer ",type(X_Tfidf_Tr))
print("the shape of out text TFIDF vectorizer ",X_Tfidf_Tr.get_shape())
print("the number of unique words including both unigrams and bigrams ", X_Tfidf_Tr.get_shape()[1])

some sample features(unique words in the corpus) ['able find', 'able get', 'absolute', 'absolutely delicious', 'absolutely loves', 'according', 'acidic', 'active', 'actual', 'add little']
=====
the type of count vectorizer <class 'scipy.sparse.csr.csr_matrix'>
the shape of out text TFIDF vectorizer (9482, 1760)
the number of unique words including both unigrams and bigrams 1760

In [276]: X_Tfidf_Tr = X_Tfidf_Tr.toarray()
X_Tfidf_Cv = X_Tfidf_Cv.toarray()
X_Tfidf_Test = X_Tfidf_Test.toarray()
tf_idf_feature = tf_idf_vect.get_feature_names()
```

[4.4] Word2Vec

```
In [277]: # Train your own Word2Vec model using your own text corpus
i=0
list_of_sentence=[]
for sentence in preprocessed_reviews:
    list_of_sentence.append(sentence.split())

In [278]: # Using Google News Word2Vectors

# in this project we are using a pretrained model by google
# its 3.3G file, once you load this into your memory
# it occupies ~90b, so please do this step only if you have >12G of ram
# we will provide a pickle file which contains a dict,
# and it contains all our corpus words as keys and model[word] as values
# To use this code-snippet, download "GoogleNews-vectors-negative300.bin"
# from https://drive.google.com/file/d/0B7XKcP15KdYNUt7L5S2jpm/edit
# it's 1.9GB in size.

# http://kavita-ganesan.com/gensim-word2vec-tutorial-starter-code/#.W175RFAz2PY
# you can comment this whole cell
# or change these variable according to your need

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 occurred atleast 5 times
    w2v_model=Word2Vec(list_of_sentence,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-negative300.bin', binary=True)
        print(w2v_model.wv.most_similar('great'))
        print(w2v_model.wv.most_similar('worst'))
    else:
        print("you don't have google's word2vec file, keep want_to_train_w2v = True, to train your own w2v ")

[('awesome', 0.8423455953598022), ('good', 0.8371583223342896), ('fantastic', 0.8326079308685303), ('excellent', 0.814829230895327), ('wonderful', 0.8053280115127563), ('amazing', 0.7542715668678284), ('decent', 0.734554589747083), ('delicious', 0.7085295753479004), ('perfect', 0.694447934627533), ('especially', 0.6708776950836182)]
=====
[('closest', 0.8194966316223145), ('personal', 0.8029736280441284), ('disappointing', 0.798120379447937), ('st', 0.7885026173591614), ('greatest', 0.77597980159568787), ('surpasses', 0.7696995139122009), ('best', 0.7687111496925354), ('fav', 0.7664926052093506), ('bye', 0.7653446197509766), ('quenching', 0.7647767663002014)]

In [279]: w2v_words = list(w2v_model.wv.vocab)
print("number of words that occurred minimum 5 times ",len(w2v_words))
print("sample words ", w2v_words[0:50])

number of words that occurred minimum 5 times 8370
sample words ['used', 'fly', 'bait', 'seasons', 'ca', 'not', 'beat', 'great', 'product', 'available', 'traps', 'course', 'total', 'pretty', 'stinky', 'right', 'nearby', 'really', 'good', 'idea', 'final', 'outstanding', 'use', 'car', 'window', 'everybody', 'asks', 'bought', 'made', 'two', 'thumbs', 'received', 'shipment', 'could', 'hardly', 'wait', 'try', 'love', 'call', 'instead', 'stickers', 'removed', 'easily', 'daughter', 'designed', 'signs', 'printed', 'reverse', 'windows', 'beautifully']
```

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

[4.4.1.1] Avg W2v

```
In [353]: # average Word2Vec
# compute average word2vec for each review.

def getAvgWordToVector(list_of_sentence):
    sent_vectors = []; # the avg-w2v for each sentence/review is stored in this list
    for sentence in list_of_sentence: # for each review/sentence
        sent = sentence.split()
        sent_vec = np.zeros(50) # as word vectors are of zero length 50, you might need to change this to 300 if you use google's w2v
        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 sent_vectors

In [354]: X_AvgW2V_Tr = getAvgWordToVector(X_tr)
X_AvgW2V_Cv = getAvgWordToVector(X_cv)
X_AvgW2V_Test = getAvgWordToVector(X_test)

In [360]: X_AvgW2V_Tr = np.array(X_AvgW2V_Tr)
X_AvgW2V_Cv = np.array(X_AvgW2V_Cv)
X_AvgW2V_Test = np.array(X_AvgW2V_Test)
```

[4.4.1.2] TFIDF weighted W2v

```
In [282]: # 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 [283]: # 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 = tfidf

def getAvgW2VtfidfToVector(list_of_sentence):
    tfidf_sent_vectors = []; # the tfidf-w2v for each sentence/review is stored in this list
    row =0;
    for sentence in list_of_sentence: # for each review/sentence
        sent = []
        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
        sent = sentence.split()
        for word in sent: # for each word in a review/sentence3
            if word in w2v_words and word in tfidf_feat:
                vec = w2v_model.wv[word]
                #tf_idf = tf_idf_matrix[row, tfidf_feat.index(word)]
                # to reduce the computation we are
                # dictionary[word] = idf value of word in whole corpus
                # sent.count(word) = tf value of word in this review
                tf_idf = dictionary[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 tfidf_sent_vectors

In [284]: X_AvgW2Vtfidf_Tr = getAvgW2VtfidfToVector(X_tr)
X_AvgW2Vtfidf_Cv = getAvgW2VtfidfToVector(X_cv)
X_AvgW2Vtfidf_Test = getAvgW2VtfidfToVector(X_test)

In [367]: X_AvgW2Vtfidf_Tr = np.array(X_AvgW2Vtfidf_Tr)
X_AvgW2Vtfidf_Cv = np.array(X_AvgW2Vtfidf_Cv)
X_AvgW2Vtfidf_Test = np.array(X_AvgW2Vtfidf_Test)
```

[5] Assignment 9: Random Forests

1. Apply Random Forests & GBDT on these feature sets

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

2. The hyper paramter tuning (Consider two hyperparameters: n_estimators & max_depth)

- Find the best hyper parameter which will give the maximum AUC (https://www.aplledaicalcourse.com/course/applied-ai-course-online/lessons/receiver-operating-characteristic-curve-roc-curve-and-auc-1/) value
- Find the best hyper paramter using k-fold cross validation or simple cross validation data
- Use gridsearch cv or randomsearch cv or you can also write your own for loops to do this task of hyperparameter tuning


3. Feature importance

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

4. Feature engineering

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

5. Representation of results

- You need to plot the performance of model both on train data and cross validation data for each hyper parameter, like shown in the figure with X-axis as **n_estimators**, Y-axis as **max_depth**, and Z-axis as **AUC Score** , we have given the notebook which explains how to plot this 3d plot, you can find it in the same drive 3d_scatter_plot 

(or)

- You need to plot the performance of model both on train data and cross validation data for each hyper parameter, like shown in the figure **seaborn heat maps** (https://seaborn.pydata.org/generated/seaborn.heatmap.html) with rows as **n_estimators**, columns as **max_depth**, and values inside the cell representing **AUC Score**
- You choose either of the plotting techniques out of 3d plot or heat map
- Once after you found the best hyper parameter, you need to train your model with it, and find the AUC on test data and plot the ROC curve on both train and test.

confusion matrices using **seaborn heatmaps**,

(https://seaborn.pydata.org/generated/seaborn.heatmap.html)
(https://seaborn.pydata.org/generated/seaborn.heatmap.html)

(https://seaborn.pydata.org/generated/seaborn.heatmap.html)

6. Conclusion (https://seaborn.pydata.org/generated/seaborn.heatmap.html)

- (https://seaborn.pydata.org/generated/seaborn.heatmap.html)
- You need to summarize the results at the end of the notebook. summarize it in the table format. To print out a table please refer to this prettytable library (https://seaborn.pydata.org/generated/seaborn.heatmap.html) link (http://zetcode.com/python/prettytable/)



Note: Data Leakage

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

[5.1] Applying RF

[5.1.1] Applying Random Forests on BOW, SET 1

```
In [294]: from sklearn.ensemble import RandomForestClassifier
from sklearn.model_selection import GridSearchCV
from wordcloud import WordCloud
from sklearn.metrics import roc_curve, auc, roc_auc_score ,accuracy_score

In [286]: params = {
    'n_estimators': [50,200,300,400],
    'max_depth' : [3,5,7,10,12]
}
```



```
In [287]: #https://www.data-science.com/resources/notebooks/random-forest-intro
rf = RandomForestClassifier(criterion='gini', class_weight = "balanced")

cv_rf = GridSearchCV(rf, cv = 10,param_grid=params,n_jobs = -1,scoring='roc_auc')
cv_rf.fit(X_Bow_Cv, Y_cv)

print('Best Parameters using grid search: \n',cv_rf.best_params_,"\n\n")
Set1_Cv_Results = pd.DataFrame(cv_rf.cv_results_)[['mean_test_score', 'std_test_score', 'params']]
print(Set1_Cv_Results)

Best Parameters using grid search:
{'max_depth': 10, 'n_estimators': 400}

mean_test_score  std_test_score  params
0      0.643441      0.025287  {'max_depth': 3, 'n_estimators': 50}
1      0.609084      0.045558  {'max_depth': 3, 'n_estimators': 200}
2      0.697472      0.039055  {'max_depth': 3, 'n_estimators': 300}
3      0.697018      0.044071  {'max_depth': 3, 'n_estimators': 400}
4      0.654929      0.047565  {'max_depth': 5, 'n_estimators': 50}
5      0.696370      0.034670  {'max_depth': 5, 'n_estimators': 200}
6      0.712885      0.031875  {'max_depth': 5, 'n_estimators': 300}
7      0.715732      0.027767  {'max_depth': 5, 'n_estimators': 400}
8      0.671349      0.042850  {'max_depth': 7, 'n_estimators': 50}
9      0.701873      0.037003  {'max_depth': 7, 'n_estimators': 200}
10     0.707175      0.047620  {'max_depth': 7, 'n_estimators': 300}
11     0.707066      0.048585  {'max_depth': 7, 'n_estimators': 400}
12     0.678914      0.039886  {'max_depth': 10, 'n_estimators': 50}
13     0.716249      0.039936  {'max_depth': 10, 'n_estimators': 200}
14     0.713080      0.049178  {'max_depth': 10, 'n_estimators': 300}
15     0.717288      0.043703  {'max_depth': 10, 'n_estimators': 400}
16     0.668089      0.039014  {'max_depth': 12, 'n_estimators': 50}
17     0.715042      0.030559  {'max_depth': 12, 'n_estimators': 200}
18     0.712791      0.040021  {'max_depth': 12, 'n_estimators': 300}
19     0.717282      0.031615  {'max_depth': 12, 'n_estimators': 400}
```

```
In [288]: #examine the best model
print("\t best_score_ :",cv_rf.best_score_)
print("\t best_params_ :",cv_rf.best_params_)
#print("\t best_estimator_ :",cv_rf.best_estimator_)
Set1_best = cv_rf.best_params_
Set1_best_max_depth = cv_rf.best_params_['max_depth']
Set1_best_estimator = cv_rf.best_params_['n_estimators']
Set1_Cv_AUC = cv_rf.best_score_

best_score_      : 0.71728791847956
best_params_     : {'max_depth': 10, 'n_estimators': 400}
```

```
In [289]: Set1_Weights = []
rf = RandomForestClassifier(criterion='gini',max_depth = Set1_best_max_depth,n_estimators = Set1_best_estimator,class_weight = "balanced")
rf.fit(X_Bow_Tr,Y_tr)
Set1_Weights = rf.feature_importances_.tolist()
```

```
In [290]: #https://qitta.com/bmj0114/items/460424c110a8ce22d945
Set1_Tr_prob = rf.predict_proba(X_Bow_Tr) # Probability of TRAIN-Validation
Set1_Tst_prob = rf.predict_proba(X_Bow_Test) # Probability of Cross-Validation

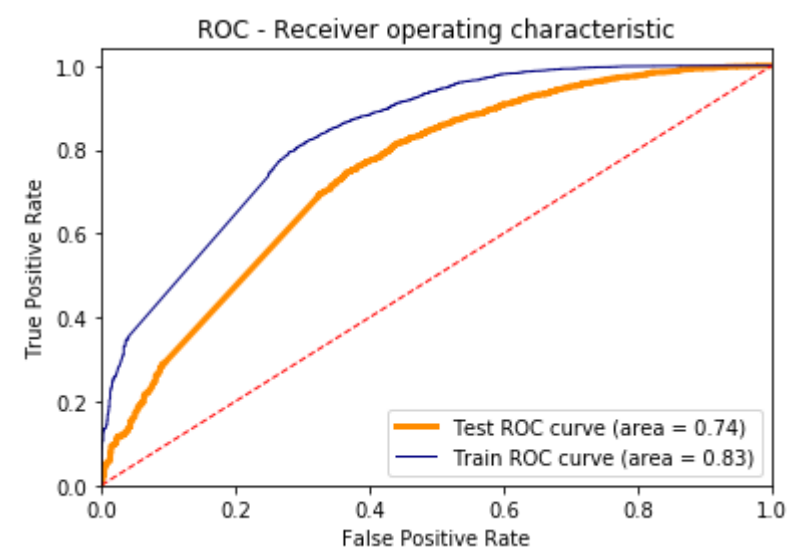
set1_tst_fpr, set1_tst_tpr, thresholds = roc_curve(Y_test,Set1_Tst_prob[:,1])
set1_tst_roc_auc = auc(set1_tst_fpr, set1_tst_tpr)

set1_train_fpr, set1_train_tpr, thresholds = roc_curve(Y_tr,Set1_Tr_prob[:,1])
set1_train_roc_auc = auc(set1_train_fpr, set1_train_tpr)

print(" Train Data      AUC for the Best Landa is ", set1_train_roc_auc)
print(" Test Validation  AUC for the Best Landa is ", set1_tst_roc_auc)

lw=1
plt.figure()
plt.plot(set1_tst_fpr, set1_tst_tpr, color='darkorange', lw=3, label='Test ROC curve (area = %0.2f)' % set1_tst_roc_auc)
plt.plot(set1_train_fpr, set1_train_tpr, color='navy', lw=1, label='Train ROC curve (area = %0.2f)' % set1_train_roc_auc)
plt.plot([0, 1], [0,1], color='red', lw=1w, linestyle='--')
plt.xlim([0.0, 1.0])
plt.ylim([0.0, 1.04])
plt.xlabel('False Positive Rate')
plt.ylabel('True Positive Rate')
plt.ylabel('True Positive Rate')
plt.title('ROC - Receiver operating characteristic')
plt.legend(loc='lower right')
plt.show()
```

Train Data AUC for the Best Landa is 0.834293209760876
Test Validation AUC for the Best Landa is 0.7409527746577063



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

```
In [291]: # Top Important features
set1_imp_Features=pd.DataFrame([Bow_Feature,Set1_Weights],index=['feature','Decision_imp']).T
#set1_imp_Features= set1_imp_Features[set1_imp_Features['Decision_imp']>0]
set1_imp_Features.sortd = set1_imp_Features.sort_values(by='Decision_imp')[::-1]
```

```
In [292]: #wordCloud = WordCloud(width = 800, height = 800,background_color = 'white',min_font_size = 10).generate(set1_imp_Features_sortd)
#text = .tolist()
import matplotlib.pyplot as plt
wordcloud = WordCloud(width = 2000, height = 800).generate(' '.join(set1_imp_Features_sortd['feature']))
#plot the WordCloud image  figsize = (8,8), facecolor = None )
plt.figure(figsize = (20, 5), facecolor = None)
plt.imshow(wordcloud, interpolation='bilinear')
plt.axis("off")
plt.tight_layout(pad = 0)
plt.show() # Please write all the code with proper documentation
```



```
In [295]: set1_Tst_Pred = rf.predict(X_Bow_Test)
set1_Tr_Pred = rf.predict(X_Bow_Tr)

set1_Tr_Acc = accuracy_score(Y_tr,set1_Tr_Pred,normalize=True)
set1_Tst_Acc = accuracy_score(Y_test,set1_Tst_Pred,normalize=True)

print("\n\nAccuracy for Train Data : ",set1_Tr_Acc)
print("\n\nAccuracy for Test Data : ",set1_Tst_Acc)

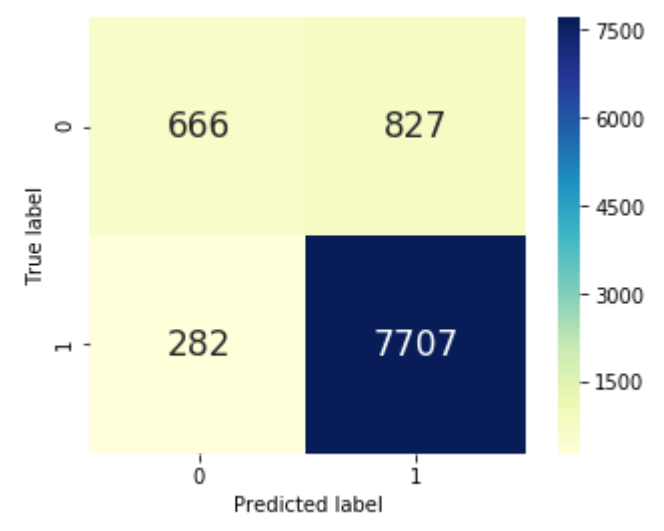
Accuracy for Train Data : 0.8830415524151023
Accuracy for Test Data : 0.8367487515068022
```

Train Confusion Matrix

```
In [296]: print("\n\nTrain Accuracy ::",set1_Tr_Acc)
Train_CM= confusion_matrix(Y_tr, set1_Tr_Pred, labels=None, sample_weight=None)
print("Confusion Matrix::\n",Train_CM,"\n\n")
plt.imshow(Train_CM, cmap='binary')
sns.heatmap(Train_CM, cmap='YlGnBu', fmt="d" ,annot=True,annot_kws={"size": 17})
plt.xlabel('Predicted label')
plt.ylabel('True label')

Train Accuracy :: 0.8830415524151023
Confusion Matrix::
[[ 666  827]
 [ 282 7707]]
```

Out[296]: Text(83.4,0.5,'True label')



Test Confusion Matrix

```
In [297]: print("\n\nTest Accuracy ::",set1_Tst_Acc)
Test_CM= confusion_matrix(Y_test, set1_Tst_Pred, labels=None, sample_weight=None)
print("\n\nConfusion Matrix::\n",Test_CM,"\n\n")
sns.heatmap(Test_CM, cmap='YlGnBu', fmt="d" ,annot=True,annot_kws={"size": 17})
plt.xlabel('Predicted label')
plt.ylabel('True label')

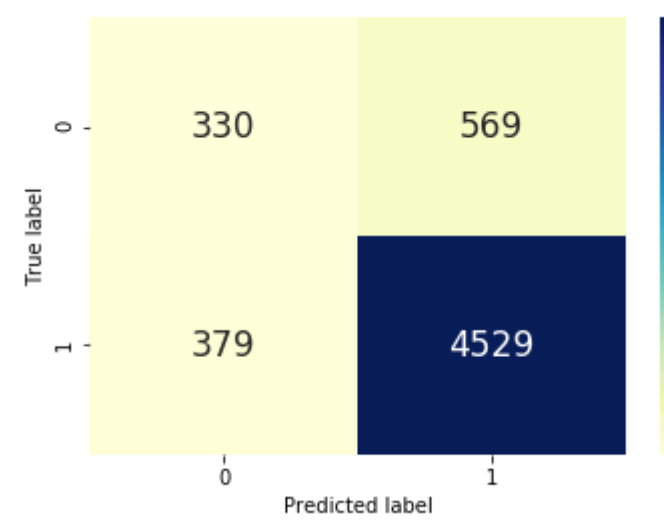
Test Accuracy :: 0.8367487515068022
```

Confusion Matrix::

[[330 569]

[379 4529]]

Out[297]: Text(33,0.5,'True label')



```
In [299]: Total_AUC = {}
```

```
In [301]: Total_AUC['set1']=Set1_best , set1_tst_roc_auc , set1_Tst_Acc]
```

[5.1.3] Applying Random Forests on TFIDF, SET 2


```
In [382]: # Please write all the code with proper documentation
https://www.data-science.com/resources/notebooks/random-forest-intro
rf = RandomForestClassifier(criterion='gini', class_weight = "balanced")

cv_rf = GridSearchCV(rf, cv = 10,param_grid=params,n_jobs = -1,scoring='roc_auc')
cv_rf.fit(X_Tfidf_cv, Y_cv)

print('Best Parameters using grid search: \n',cv_rf.best_params_,"\n\n")
set2_cv_Results = pd.DataFrame(cv_rf.cv_results_[["mean_test_score", 'std_test_score', 'params']]
print(set2_cv_Results)

Best Parameters using grid search:
{'max_depth': 7, 'n_estimators': 400}

mean_test_score std_test_score params
0 0.662213 0.047307 {'max_depth': 3, 'n_estimators': 50}
1 0.735209 0.045516 {'max_depth': 3, 'n_estimators': 200}
2 0.735797 0.040532 {'max_depth': 3, 'n_estimators': 300}
3 0.744562 0.028849 {'max_depth': 3, 'n_estimators': 400}
4 0.691272 0.040558 {'max_depth': 5, 'n_estimators': 50}
5 0.748971 0.031082 {'max_depth': 5, 'n_estimators': 200}
6 0.746537 0.041614 {'max_depth': 5, 'n_estimators': 300}
7 0.753043 0.043247 {'max_depth': 5, 'n_estimators': 400}
8 0.694922 0.039421 {'max_depth': 7, 'n_estimators': 50}
9 0.744150 0.041287 {'max_depth': 7, 'n_estimators': 200}
10 0.746680 0.026723 {'max_depth': 7, 'n_estimators': 300}
11 0.757011 0.036303 {'max_depth': 7, 'n_estimators': 400}
12 0.709344 0.034387 {'max_depth': 10, 'n_estimators': 50}
13 0.736842 0.029193 {'max_depth': 10, 'n_estimators': 200}
14 0.750815 0.026400 {'max_depth': 10, 'n_estimators': 300}
15 0.754055 0.031486 {'max_depth': 10, 'n_estimators': 400}
16 0.707134 0.035376 {'max_depth': 12, 'n_estimators': 50}
17 0.750006 0.037306 {'max_depth': 12, 'n_estimators': 200}
18 0.753781 0.024363 {'max_depth': 12, 'n_estimators': 300}
19 0.756736 0.035124 {'max_depth': 12, 'n_estimators': 400}
```

```
In [383]: #examine the best model
print("\t best_score_ :",cv_rf.best_score_)
print("\t best_params_ :",cv_rf.best_params_)
#print("\t best_estimator_ :",cv_rf.best_estimator_)
set2_best = cv_rf.best_params_
set2_best_max_depth = cv_rf.best_params_['max_depth']
set2_best_estimator = cv_rf.best_params_['n_estimators']
set2_cv_AUC = cv_rf.best_score_

best_score_ : 0.7570108882707602
best_params_ : {'max_depth': 7, 'n_estimators': 400}
```

```
In [384]: set2_Weights = []
rf = RandomForestClassifier(criterion='gini',max_depth = set2_best_max_depth,n_estimators = set2_best_estimator,class_weight = "balanced")
rf.fit(X_Tfidf_Tr,Y_Tr)
set2_Weights = rf.feature_importances_.tolist()
```

```
In [385]: https://q.ito.com/hw/9114/items/468424c11008cc220405
set2_Tr_prob = rf.predict_proba(X_Tfidf_Tr) # Probability of TRAIN-Validation
set2_Tst_prob = rf.predict_proba(X_Tfidf_Test) # Probability of Cross-Validation

set2_tst_fpr, set2_tst_tpr, thresholds = roc_curve(Y_test,set2_Tst_prob[:,1])
set2_tst_roc_auc = auc(set2_tst_fpr, set2_tst_tpr)

set2_train_fpr, set2_train_tpr, thresholds = roc_curve(Y_Tr,set2_Tr_prob[:,1])
set2_train_roc_auc = auc(set2_train_fpr, set2_train_tpr)

print(" Train Data AUC for the Best Lambda is ", set2_train_roc_auc)
print(" Test Validation AUC for the BEst Lambda is ", set2_tst_roc_auc)

lw=1
plt.figure()
plt.plot(set2_tst_fpr, set2_tst_tpr, color='darkorange', lw=3, label='Test ROC curve (area = %0.2f)' % set2_tst_roc_auc)
plt.plot(set2_train_fpr, set2_train_tpr, color='navy', lw=1, label='Train ROC curve (area = %0.2f)' % set2_train_roc_auc)
plt.plot([0, 1], [0,1], color='red', lw=1, linestyle='--')
plt.xlim([0.0, 1.0])
plt.ylim([0.0, 1.0])
plt.xlabel('False Positive Rate')
plt.ylabel('True Positive Rate')
plt.title('ROC - Receiver operating characteristic')
plt.legend(loc='lower right')
plt.show()

Train Data AUC for the Best Lambda is 0.8669396139718906
Test Validation AUC for the BEst Lambda is 0.7799772771158391

ROC - Receiver operating characteristic
True Positive Rate
False Positive Rate
Test ROC curve (area = 0.78)
Train ROC curve (area = 0.87)
```

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

```
In [386]: # Top Important features
set2_Imp_Features=pd.DataFrame([tf_idf_feature,set2_Weights],index=['feature','Decision_Imp']).T
#set2_Imp_Features= set2_Imp_Features[[set2_Imp_Features['Decision_Imp']>0]]
set2_Imp_Features_sortd = set2_Imp_Features.sort_values(by='Decision_Imp')[:-20][::-1]
set2_Imp_Features_sortd
```

Out[386]:

	feature	Decision_Imp
1679	waste money	0.0428325
398	disappointing	0.0319496
1282	refund	0.0319321
135	beware	0.0288037
1666	threw	0.0287607
1075	nothing like	0.024663
403	disgusting	0.0189419
1056	not purchase	0.0187356
367	definitely not	0.0180357
1190	poor	0.0178736
1069	not waste	0.0177539
399	disappointment	0.0174419
1658	vomiting	0.0164105
990	never buy	0.0155283
389	died	0.0141417
892	made china	0.0136379
1600	trash	0.0127151
1185	positive	0.012324
1324	sadly	0.0118478
1461	stay away	0.0117262

```
In [387]: #wordCloud = WordCloud(width = 800, height = 800,background_color = 'white',min_font_size = 10).generate(set2_Imp_Features_sortd)
#text = .tolist()
import matplotlib.pyplot as plt
wordCloud = WordCloud(width = 2000, height = 800,generate(' '.join(set2_Imp_Features_sortd['feature'])))
#plot the WordCloud image figsize = (8,8), facecolor = None
plt.figure(figsize = (20, 5), facecolor = None)
plt.imshow(wordCloud, interpolation='bilinear')
plt.axis('off')
plt.tight_layout(pad = 0)
plt.show() # Please write all the code with proper documentation
```



```
In [388]: # Please write all the code with proper documentation
```

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

```
In [389]: https://www.data-science.com/resources/notebooks/random-forest-intro
rf = RandomForestClassifier(criterion='gini', class_weight = "balanced")
cv_rf = GridSearchCV(rf, cv = 10,param_grid=params,n_jobs = -1,scoring='roc_auc')
cv_rf.fit(X_AvgW2V_cv, Y_cv)

print('Best Parameters using grid search: \n',cv_rf.best_params_,"\n\n")
set3_cv_Results = pd.DataFrame(cv_rf.cv_results_[["mean_test_score", 'std_test_score', 'params']]
print(set3_cv_Results)

Best Parameters using grid search:
{'max_depth': 12, 'n_estimators': 400}

mean_test_score std_test_score params
0 0.836100 0.023008 {'max_depth': 3, 'n_estimators': 50}
1 0.840973 0.023082 {'max_depth': 3, 'n_estimators': 200}
2 0.841936 0.023063 {'max_depth': 3, 'n_estimators': 300}
3 0.840683 0.026147 {'max_depth': 3, 'n_estimators': 400}
4 0.850090 0.025923 {'max_depth': 5, 'n_estimators': 50}
5 0.853525 0.024603 {'max_depth': 5, 'n_estimators': 200}
6 0.852137 0.025501 {'max_depth': 5, 'n_estimators': 300}
7 0.855531 0.024954 {'max_depth': 5, 'n_estimators': 400}
8 0.855585 0.025901 {'max_depth': 7, 'n_estimators': 50}
9 0.858596 0.025968 {'max_depth': 7, 'n_estimators': 200}
10 0.858525 0.024750 {'max_depth': 7, 'n_estimators': 300}
11 0.857500 0.027270 {'max_depth': 7, 'n_estimators': 400}
12 0.85358 0.024602 {'max_depth': 10, 'n_estimators': 50}
13 0.859845 0.025576 {'max_depth': 10, 'n_estimators': 200}
14 0.860273 0.026159 {'max_depth': 10, 'n_estimators': 300}
15 0.860995 0.027224 {'max_depth': 10, 'n_estimators': 400}
16 0.853554 0.025247 {'max_depth': 12, 'n_estimators': 50}
17 0.860726 0.027989 {'max_depth': 12, 'n_estimators': 200}
18 0.860837 0.027059 {'max_depth': 12, 'n_estimators': 300}
19 0.861734 0.025802 {'max_depth': 12, 'n_estimators': 400}
```

```
In [310]: #examine the best model
print("\t best_score_ :",cv_rf.best_score_)
print("\t best_params_ :",cv_rf.best_params_)
#print("\t best_estimator_ :",cv_rf.best_estimator_)
set3_best = cv_rf.best_params_
set3_best_max_depth = cv_rf.best_params_['max_depth']
set3_best_estimator = cv_rf.best_params_['n_estimators']
set3_cv_AUC = cv_rf.best_score_

best_score_ : 0.861733977932179
best_params_ : {'max_depth': 12, 'n_estimators': 400}
```

```
In [311]: set3_Weights = []
rf = RandomForestClassifier(criterion='gini',max_depth = set3_best_max_depth,n_estimators = set3_best_estimator,class_weight = "balanced")
rf.fit(X_AvgW2V_Tr,Y_Tr)
set3_Weights = rf.feature_importances_.tolist()
```



```
In [312]: #https://qitita.com/bmj0114/items/460424c110a8ce22d945
set3_Tr_prob = rf.predict_proba(X_Avg02V_Tr) # Probability of TRAIN-Validation
set3_Tst_prob = rf.predict_proba(X_Avg02V_Test) # Probability of Cross-Validation

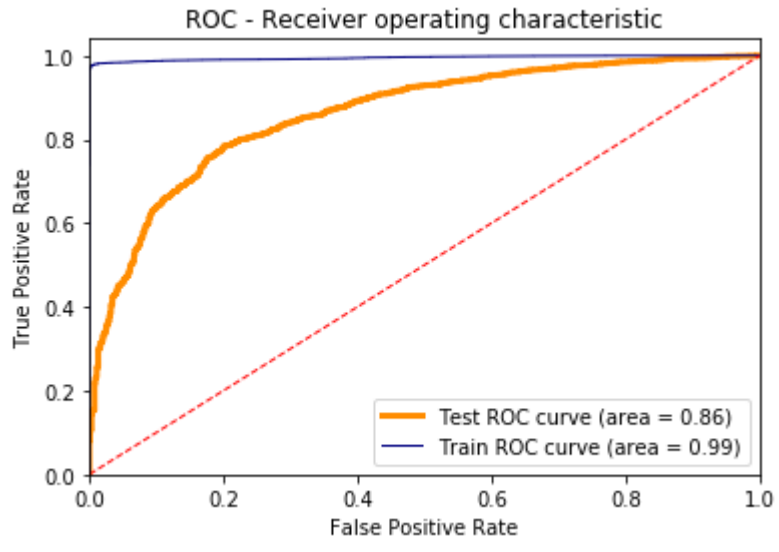
set3_tst_fpr, set3_tst_tpr, thresholds = roc_curve(Y_test,set3_Tst_prob[:,1])
set3_tst_roc_auc = auc(set3_tst_fpr, set3_tst_tpr)

set3_train_fpr, set3_train_tpr, thresholds = roc_curve(Y_tr,set3_Tr_prob[:,1])
set3_train_roc_auc = auc(set3_train_fpr, set3_train_tpr)

print(" Train Data      AUC for the Best Landa is ", set3_train_roc_auc)
print(" Test Validation AUC for the Best Landa is ", set3_tst_roc_auc)

lw=1
plt.figure()
plt.plot(set3_tst_fpr, set3_tst_tpr, color='darkorange', lw=3, label='Test ROC curve (area = %0.2f)' % set3_tst_roc_auc)
plt.plot(set3_train_fpr, set3_train_tpr, color='navy', lw=1, label='Train ROC curve (area = %0.2f)' % set3_train_roc_auc)
plt.plot([0, 1], [0, 1], color='red', lw=1, linestyle='--')
plt.xlim([0.0, 1.0])
plt.ylim([0.0, 1.04])
plt.xlabel('False Positive Rate')
plt.ylabel('True Positive Rate')
plt.title('ROC - Receiver operating characteristic')
plt.legend(loc='lower right')
plt.show()
```

Train Data AUC for the Best Landa is 0.9945529171599563
Test Validation AUC for the Best Landa is 0.8600261496745908



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

```
In [313]: #https://www.data-science.com/resources/notebooks/random-forest-intro
rf = RandomForestClassifier(criterion='gini', class_weight = "balanced")

cv_rf = GridSearchCV(rf, cv = 10,param_grid=params,n_jobs = -1,scoring='roc_auc')
cv_rf.fit(X_Avg02Vtfidf_Cv, Y_Cv)

print('Best Parameters using grid search: \n',cv_rf.best_params_,"\n\n")
set4_Cv_Results = pd.DataFrame(cv_rf.cv_results_)[['mean_test_score', 'std_test_score', 'params']]
print(set4_Cv_Results)
```

Best Parameters using grid search:
{'max_depth': 12, 'n_estimators': 400}

	mean_test_score	std_test_score	params
0	0.807602	0.022725	{'max_depth': 3, 'n_estimators': 50}
1	0.813968	0.024227	{'max_depth': 3, 'n_estimators': 200}
2	0.815458	0.021999	{'max_depth': 3, 'n_estimators': 300}
3	0.814543	0.022441	{'max_depth': 3, 'n_estimators': 400}
4	0.820643	0.023053	{'max_depth': 5, 'n_estimators': 50}
5	0.826128	0.024330	{'max_depth': 5, 'n_estimators': 200}
6	0.825642	0.022612	{'max_depth': 5, 'n_estimators': 300}
7	0.825571	0.024993	{'max_depth': 5, 'n_estimators': 400}
8	0.825886	0.026777	{'max_depth': 7, 'n_estimators': 50}
9	0.830532	0.024474	{'max_depth': 7, 'n_estimators': 200}
10	0.827999	0.024073	{'max_depth': 7, 'n_estimators': 300}
11	0.829130	0.025798	{'max_depth': 7, 'n_estimators': 400}
12	0.830379	0.025085	{'max_depth': 10, 'n_estimators': 50}
13	0.827539	0.024124	{'max_depth': 10, 'n_estimators': 200}
14	0.832652	0.023917	{'max_depth': 10, 'n_estimators': 300}
15	0.831062	0.023785	{'max_depth': 10, 'n_estimators': 400}
16	0.825817	0.028909	{'max_depth': 12, 'n_estimators': 50}
17	0.829274	0.025613	{'max_depth': 12, 'n_estimators': 200}
18	0.831568	0.024501	{'max_depth': 12, 'n_estimators': 300}
19	0.833066	0.024335	{'max_depth': 12, 'n_estimators': 400}

```
In [314]: #examine the best model
print("\t best_score_      :",cv_rf.best_score_)
print("\t best_params_     :",cv_rf.best_params_)
#print("\t best_estimator_ :",cv_rf.best_estimator_)
set4_best = cv_rf.best_params_
set4_best_max_depth = cv_rf.best_params_['max_depth']
set4_best_estimator = cv_rf.best_params_['n_estimators']
set4_Cv_AUC = cv_rf.best_score_

best_score_      : 0.8330661854986233
best_params_     : {'max_depth': 12, 'n_estimators': 400}
```

```
In [315]: set4_Weights = []
rf = RandomForestClassifier(criterion='gini',max_depth = set4_best_max_depth,n_estimators = set4_best_estimator,class_weight = "balanced")
rf.fit(X_Avg02Vtfidf_Tr,Y_Tr)
set4_Weights = rf.feature_importances_.tolist()
```

```
In [316]: #https://qitita.com/bmj0114/items/460424c110a8ce22d945
set4_Tr_prob = rf.predict_proba(X_Avg02Vtfidf_Tr) # Probability of TRAIN-Validation
set4_Tst_prob = rf.predict_proba(X_Avg02Vtfidf_Test) # Probability of Cross-Validation

set4_tst_fpr, set4_tst_tpr, thresholds = roc_curve(Y_test,set4_Tst_prob[:,1])
set4_tst_roc_auc = auc(set4_tst_fpr, set4_tst_tpr)

set4_train_fpr, set4_train_tpr, thresholds = roc_curve(Y_tr,set4_Tr_prob[:,1])
set4_train_roc_auc = auc(set4_train_fpr, set4_train_tpr)

print(" Train Data      AUC for the Best Landa is ", set4_train_roc_auc)
print(" Test Validation AUC for the Best Landa is ", set4_tst_roc_auc)

lw=1
plt.figure()
plt.plot(set4_tst_fpr, set4_tst_tpr, color='darkorange', lw=3, label='Test ROC curve (area = %0.2f)' % set4_tst_roc_auc)
plt.plot(set4_train_fpr, set4_train_tpr, color='navy', lw=1, label='Train ROC curve (area = %0.2f)' % set4_train_roc_auc)
plt.plot([0, 1], [0, 1], color='red', lw=1, linestyle='--')
plt.xlim([0.0, 1.0])
plt.ylim([0.0, 1.04])
plt.xlabel('False Positive Rate')
plt.ylabel('True Positive Rate')
plt.title('ROC - Receiver operating characteristic')
plt.legend(loc='lower right')
plt.show()
```

Train Data AUC for the Best Landa is 0.9967385245134028
Test Validation AUC for the Best Landa is 0.8402028922836476

