

# Amazon Fine Food Reviews Analysis

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

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

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

Number of reviews: 568,454

Number of users: 256,059

Number of products: 74,258

Timespan: Oct 1999 - Oct 2012

Number of Attributes/Columns in data: 10

Attribute Information:

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

## Objective:

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

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

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

## [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 [3]: %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:\ProgramData\Anaconda3\lib\site-packages\gensim\utils.py:1197: UserWarning: detected Windows; aliasing chunkize to chunkize_serial
  warnings.warn("detected Windows; aliasing chunkize to chunkize_serial")

```

```

In [4]: # using SQLite Table to read data.
con = sqlite3.connect('C:/Users/Excel/Desktop/vins/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 50
0000 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
# re != 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 100000""", 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 (100000, 10)

Out[4]:

	Id	ProductId	UserId	ProfileName	HelpfulnessNumerator	HelpfulnessDenominator
0	1	B001E4KFG0	A3SGXH7AUHU8GW	delmartian	1	1

	Id	ProductId	UserId	ProfileName	HelpfulnessNumerator	HelpfulnessDenominator
1	2	B00813GRG4	A1D87F6ZCVE5NK	dll pa	0	0
2	3	B000LQOCH0	ABXLMWJIXXAIN	Natalia Corres "Natalia Corres"	1	1

```
In [5]: display = pd.read_sql_query("""
SELECT UserId, ProductId, ProfileName, Time, Score, Text, COUNT(*)
FROM Reviews
GROUP BY UserId
HAVING COUNT(*)>1
""", con)
```

```
In [6]: print(display.shape)
display.head()
```

```
(80668, 7)
```

```
Out[6]:
```

	UserId	ProductId	ProfileName	Time	Score	Text	COUNT(*)
--	--------	-----------	-------------	------	-------	------	----------

	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 [7]: `display[display['UserId']=='AZY10LLTJ71NX']`

Out[7]:

	UserId	ProductId	ProfileName	Time	Score	Text	COUNT
--	--------	-----------	-------------	------	-------	------	-------

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

In [8]: `display['COUNT(*)'].sum()`

Out[8]: 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 [9]: `display= pd.read_sql_query("""  
SELECT *  
FROM Reviews  
WHERE Score != 3 AND UserId="AR5J8UI46CURR"  
ORDER BY ProductID  
""", con)  
display.head()`

Out[9]:

	Id	ProductId	UserId	ProfileName	HelpfulnessNumerator	Helpfuln
--	----	-----------	--------	-------------	----------------------	----------

	Id	ProductId	UserId	ProfileName	HelpfulnessNumerator	Helpfuln
0	78445	B000HDL1RQ	AR5J8UI46CURR	Geetha Krishnan	2	2
1	138317	B000HDOPYC	AR5J8UI46CURR	Geetha Krishnan	2	2
2	138277	B000HDOPYM	AR5J8UI46CURR	Geetha Krishnan	2	2
3	73791	B000HDOPZG	AR5J8UI46CURR	Geetha Krishnan	2	2
4	155049	B000PAQ75C	AR5J8UI46CURR	Geetha Krishnan	2	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 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 [10]: #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 [11]: #Deduplication of entries
final=sorted_data.drop_duplicates(subset={"UserId","ProfileName","Time",
"Text"}, keep='first', inplace=False)
final.shape
```

```
Out[11]: (87775, 10)
```

```
In [12]: #Checking to see how much % of data still remains
(final['Id'].size*1.0)/(filtered_data['Id'].size*1.0)*100
```

```
Out[12]: 87.775
```

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

display.head()
```

Out[13]:

	Id	ProductId	UserId	ProfileName	HelpfulnessNumerator	Helpfuln
0	64422	B000MIDROQ	A161DK06JJMCYF	J. E. Stephens "Jeanne"	3	1
1	44737	B001EQ55RW	A2V0I904FH7ABY	Ram	3	2

```
In [14]: final=final[final.HelpfulnessNumerator<=final.HelpfulnessDenominator]
```

```
In [15]: #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()
```

```
(87773, 10)
```

```
Out[15]: 1    73592  
        0    14181  
        Name: Score, dtype: int64
```

```
In [16]: final['Time']=pd.to_datetime(final['Time'],unit='s')  
        final=final.sort_values(by='Time')
```

## [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 observed to be better than Porter Stemming)

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

```
In [17]: # 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)

```

I bought a few of these after my apartment was infested with fruit flies. After only a few hours, the trap had "attracted" many flies and within a few days they were practically gone. This may not be a long term solution, but if flies are driving you crazy, consider buying this. One caution- the surface is very sticky, so try to avoid touching it.

=====

I have made these brownies for family and for a den of cub scouts and no one would have known they were gluten free and everyone asked for seconds! These brownies have a fudgy texture and have bits of chocolate chips in them which are delicious. I would say the mix is very thick and a little difficult to work with. The cooked brownies are slightly difficult to cut into very neat edges as the edges tend to crumble a little and I would also say that they make a slightly thinner layer of brownies than most of the store brand gluten containing but they taste just as good, if not better. Highly recommended!<br /><br />(For those wondering, this mix requires 2 eggs OR 4 egg whites and 7 tbs melted butter to prepare. They do have suggestions for lactose free and low fat preparations)

=====

This gum is my absolute favorite. By purchasing on amazon I can get the savings of large quantities at a very good price. I highly recommend to all gum chewers. Plus as you enjoy the peppermint flavor and freshening of breath you are whitening your teeth all at the same time.

```
=====
This is an excellent product, both tasty and priced right. It's difficult to find this product in regular local grocery stores, so I was thrilled to find it.
=====
```

```
In [18]: # 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 bought a few of these after my apartment was infested with fruit flies. After only a few hours, the trap had "attracted" many flies and within a few days they were practically gone. This may not be a long term solution, but if flies are driving you crazy, consider buying this. One caution- the surface is very sticky, so try to avoid touching it.

```
In [19]: # 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)
```

I bought a few of these after my apartment was infested with fruit flies. After only a few hours, the trap had "attracted" many flies and within a few days they were practically gone. This may not be a long term solution, but if flies are driving you crazy, consider buying this. One caution- the surface is very sticky, so try to avoid touching it.

=====

I have made these brownies for family and for a den of cub scouts and no one would have known they were gluten free and everyone asked for seconds! These brownies have a fudgy texture and have bits of chocolate chips in them which are delicious. I would say the mix is very thick and a little difficult to work with. The cooked brownies are slightly difficult to cut into very neat edges as the edges tend to crumble a little and I would also say that they make a slightly thinner layer of brownies than most of the store brand gluten containing but they taste just as good, if not better. Highly recommended!(For those wondering, this mix requires 2 eggs OR 4 egg whites and 7 tbs melted butter to prepare. They do have suggestions for lactose free and low fat preparations)

=====

This gum is my absolute favorite. By purchasing on amazon I can get the savings of large quantities at a very good price. I highly recommend to all gum chewers. Plus as you enjoy the peppermint flavor and freshening of breath you are whitening your teeth all at the same time.

=====

This is an excellent product, both tasty and priced right. It's difficult to find this product in regular local grocery stores, so I was thrilled to find it.

```
In [20]: # 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 [21]: sent_1500 = decontracted(sent_1500)
print(sent_1500)
print("="*50)

```

This gum is my absolute favorite. By purchasing on amazon I can get the savings of large quantities at a very good price. I highly recommend to all gum chewers. Plus as you enjoy the peppermint flavor and freshening of breath you are whitening your teeth all at the same time.

=====

```

In [22]: #remove words with numbers python: https://stackoverflow.com/a/18082370/4084039
sent_0 = re.sub(r"\S*\d\S*", "", sent_0).strip()
print(sent_0)

```

I bought a few of these after my apartment was infested with fruit flies. After only a few hours, the trap had "attracted" many flies and within a few days they were practically gone. This may not be a long term solution, but if flies are driving you crazy, consider buying this. One caution- the surface is very sticky, so try to avoid touching it.

```

In [23]: #remove spacial character: https://stackoverflow.com/a/5843547/4084039
sent_1500 = re.sub(r'[^A-Za-z0-9]+', ' ', sent_1500)
print(sent_1500)

```

This gum is my absolute favorite By purchasing on amazon I can get the savings of large quantities at a very good price I highly recommend to all gum chewers Plus as you enjoy the peppermint flavor and freshening of breath you are whitening your teeth all at the same time

```
In [24]: # 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", 'your', 'yours', 'yourself', 'yourselves', 'he', 'him', 'his', 'himself', \
               'she', "she's", 'her', 'hers', 'herself', 'it', "it's", 'its', 'itself', 'they', 'them', 'their', \
               'theirs', 'themselves', 'what', 'which', 'who', 'whom', 'this', 'that', "that'll", 'these', 'those', \
               'am', 'is', 'are', 'was', 'were', 'be', 'been', 'being', 'have', 'has', 'had', 'having', 'do', 'does', \
               'did', 'doing', 'a', 'an', 'the', 'and', 'but', 'if', 'or', 'because', 'as', 'until', 'while', 'of', \
               'at', 'by', 'for', 'with', 'about', 'against', 'between', 'into', 'through', 'during', 'before', 'after', \
               'above', 'below', 'to', 'from', 'up', 'down', 'in', 'out', 'on', 'off', 'over', 'under', 'again', 'further', \
               'then', 'once', 'here', 'there', 'when', 'where', 'why', 'how', 'all', 'any', 'both', '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", 'didn', "didn't", 'doesn', "doesn't", 'hadn', \
               "hadn't", 'hasn', "hasn't", 'haven', "haven't", 'isn', "isn't", 'ma', 'mightn', "mightn't", 'mustn', \
```



```
"mustn't", 'needn', "needn't", 'shan', "shan't", 'shouldn',  
"shouldn't", 'wasn', "wasn't", 'weren', "weren't", \  
'won', "won't", 'wouldn', "wouldn't"])
```

```
In [25]: # Combining all the above students  
from tqdm import tqdm  
preprocessed_reviews = []  
# tqdm is for printing the status bar  
for sentence in tqdm(final['Text'].values):  
    sentence = re.sub(r"http\S+", "", sentence)  
    sentence = BeautifulSoup(sentence, 'lxml').get_text()  
    sentence = decontracted(sentence)  
    sentence = re.sub("\S*\d\S*", "", sentence).strip()  
    sentence = re.sub('[^A-Za-z]+', ' ', sentence)  
    # https://gist.github.com/sebleier/554280  
    sentence = ' '.join(e.lower() for e in sentence.split() if e.lower  
( ) not in stopwords)  
    preprocessed_reviews.append(sentence.strip())
```

```
100%|████████████████████████████████████████| 87773/87773 [00:38<00:00, 226  
3.93it/s]
```

```
In [26]: preprocessed_reviews[1500]
```

```
Out[26]: 'gum absolute favorite purchasing amazon get savings large quantities go  
od price highly recommend gum chewers plus enjoy peppermint flavor fres  
hing breath whitening teeth time'
```

## [3.2] Preprocessing Review Summary

### BOW XGBOOST

```
In [35]: X=preprocessed_reviews  
Y=final['Score']
```

```
In [36]: from sklearn.model_selection import train_test_split
X_train,X_test,y_train,y_test=train_test_split(X,Y,test_size=0.33,shuffle=False)
X_train,X_cv,y_train,y_cv=train_test_split(X_train,y_train,test_size=0.33,shuffle=False)
```

```
In [37]: from sklearn.feature_extraction.text import CountVectorizer
vectorizer=CountVectorizer(min_df=10,max_features=10000,ngram_range=(1,2))
vectorizer.fit(X_train)

X_train_bow=vectorizer.transform(X_train)
X_cv_bow=vectorizer.transform(X_cv)
X_test_bow=vectorizer.transform(X_test)

print(np.shape(X_train_bow),np.shape(X_cv_bow),np.shape(X_test_bow))

(39400, 10000) (19407, 10000) (28966, 10000)
```

```
In [38]: from sklearn.model_selection import GridSearchCV,RandomizedSearchCV
import xgboost as xgb
n_estimator=[20,40,60,80,90,100]
depth=[10,50,100,500,1000]
parameter={"n_estimators":n_estimator,"max_depth":depth}
RFX=GridSearchCV(xgb.XGBClassifier(),parameter,verbose=1,scoring='roc_auc',n_jobs=-1)
RFX.fit(X_train_bow,y_train)

opt_estimator,opt_depth=RFX.best_params_.get('n_estimators'),RFX.best_params_.get('max_depth')
print("The best optimized estimator:",opt_estimator)
print("The best optimized depth: ",opt_depth)

train_score=RFX.cv_results_.get('mean_train_score')
cv_score=RFX.cv_results_.get('mean_test_score')

df_heatmap=pd.DataFrame(train_score.reshape(6,5),index=n_estimator,columns=depth)
fig=plt.figure(figsize=(5,3))
```

```

heatmap=sns.heatmap(df_heatmap,annot=True)
plt.ylabel("n_estimators",size=20)
plt.xlabel("depth",size=20)
plt.title("CV data for train data")
plt.show()

df_heatmap=pd.DataFrame(cv_score.reshape(6,5),index=n_estimator,columns=depth)
fig=plt.figure(figsize=(5,4))
heatmap=sns.heatmap(df_heatmap,annot=True)
plt.ylabel("n_estimators",size=20)
plt.xlabel("depth",size=20)
plt.title("cv data for test data")
plt.show()

```

Fitting 3 folds for each of 30 candidates, totalling 90 fits

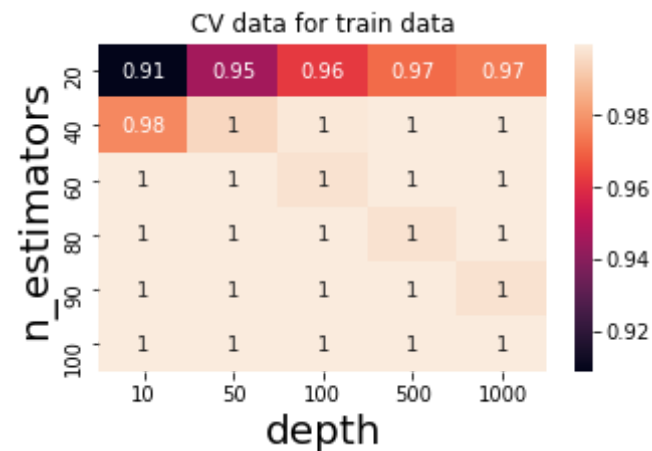
```

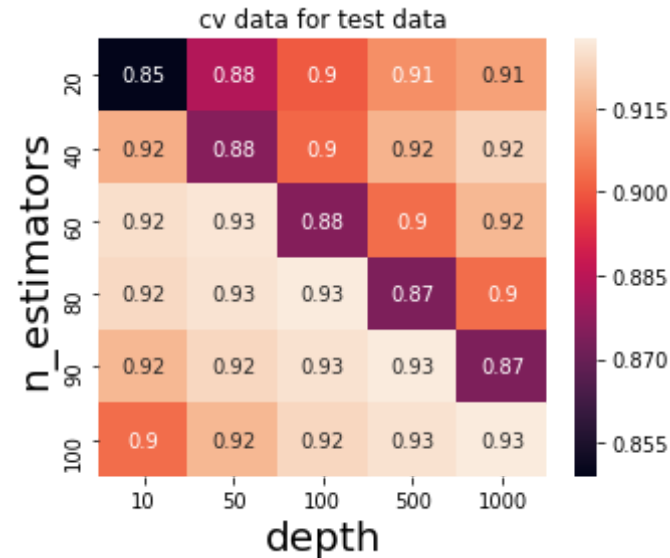
[Parallel(n_jobs=-1)]: Done 42 tasks      | elapsed: 16.1min
[Parallel(n_jobs=-1)]: Done 90 out of 90 | elapsed: 58.9min finished

```

The best optimized estimator: 100

The best optimized depth: 100





```
In [39]: from sklearn.metrics import roc_auc_score
RFX=xgb.XGBClassifier(n_estimators=100,max_depth=100)
RFX.fit(X_train_bow,y_train)

y_train_pred_proba=RFX.predict_proba(X_train_bow)
y_test_pred_proba=RFX.predict_proba(X_test_bow)

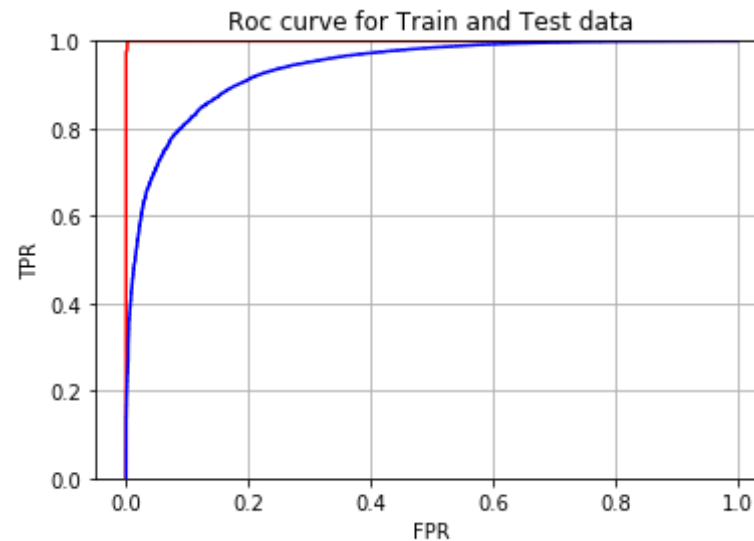
fpr, tpr, threshold=roc_curve(y_train,y_train_pred_proba[:,1])
fpr1, tpr1, threshold1=roc_curve(y_test,y_test_pred_proba[:,1])

print("The AUC value for Test data:", roc_auc_score(y_test,y_test_pred_p
roba[:,1]))

plt.plot(fpr,tpr,'r',label="Train data")
plt.plot(fpr1,tpr1,'b',label='Test data')
plt.ylim(0,1)
plt.grid(True)
plt.title("Roc curve for Train and Test data")
plt.xlabel("FPR")
plt.ylabel("TPR")
plt.show()
```

The AUC value for Test data: 0.9333333333333333

The AUC value for Test data: 0.9392231740758202



```
In [40]: def confusion_matrix(y_test, test, y_train, train):  
    from sklearn.metrics import confusion_matrix  
    y_test_pred=RFX.predict(test)  
    y_train_pred=RFX.predict(train)  
    cm_test=confusion_matrix(y_test,y_test_pred)  
    cm_train=confusion_matrix(y_train,y_train_pred)  
    print("confusion matrix on test data")  
    print(cm_test)  
    print("***200")  
    print("confusion matrix on train data")  
    print(cm_train)  
    import seaborn as sns  
    class_label=["0","1"]  
    df_cm=pd.DataFrame(cm_test,index=class_label,columns=class_label)  
    sns.heatmap(df_cm,annot=True,fmt='d')  
    plt.xlabel("predicted label",size=15)  
    plt.ylabel("actual label",size=15)  
    plt.title("confusion matrix",size=18)  
    plt.show()
```

```
In [41]: confusion_matrix(y_test,X_test_bow,y_train,X_train_bow)
```

```
C:\ProgramData\Anaconda3\lib\site-packages\sklearn\preprocessing\label.  
py:151: DeprecationWarning: The truth value of an empty array is ambigu  
ous. Returning False, but in future this will result in an error. Use `  
array.size > 0` to check that an array is not empty.
```

```
if diff:
```

```
C:\ProgramData\Anaconda3\lib\site-packages\sklearn\preprocessing\label.  
py:151: DeprecationWarning: The truth value of an empty array is ambigu  
ous. Returning False, but in future this will result in an error. Use `  
array.size > 0` to check that an array is not empty.
```

```
if diff:
```

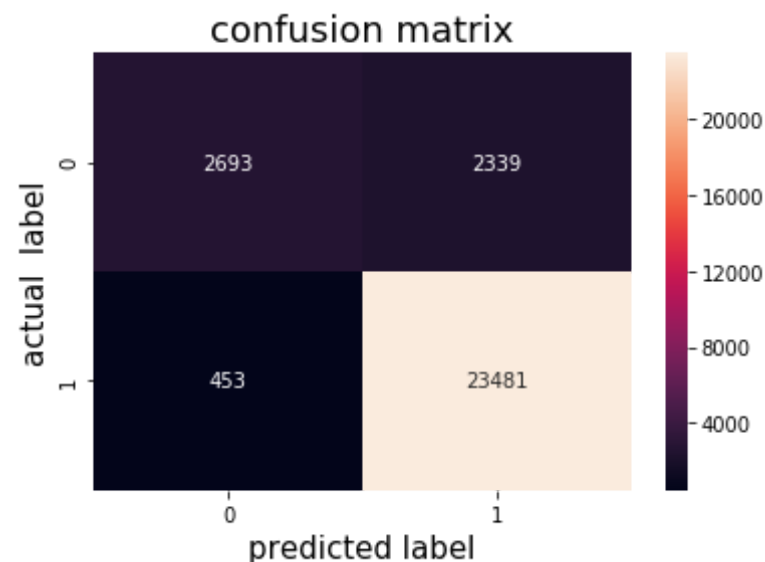
```
confusion matrix on test data
```

```
[[ 2693  2339]  
 [   453 23481]]
```

```
*****  
*****  
*****
```

```
confusion matrix on train data
```

```
[[ 5706    32]  
 [    1 33661]]
```



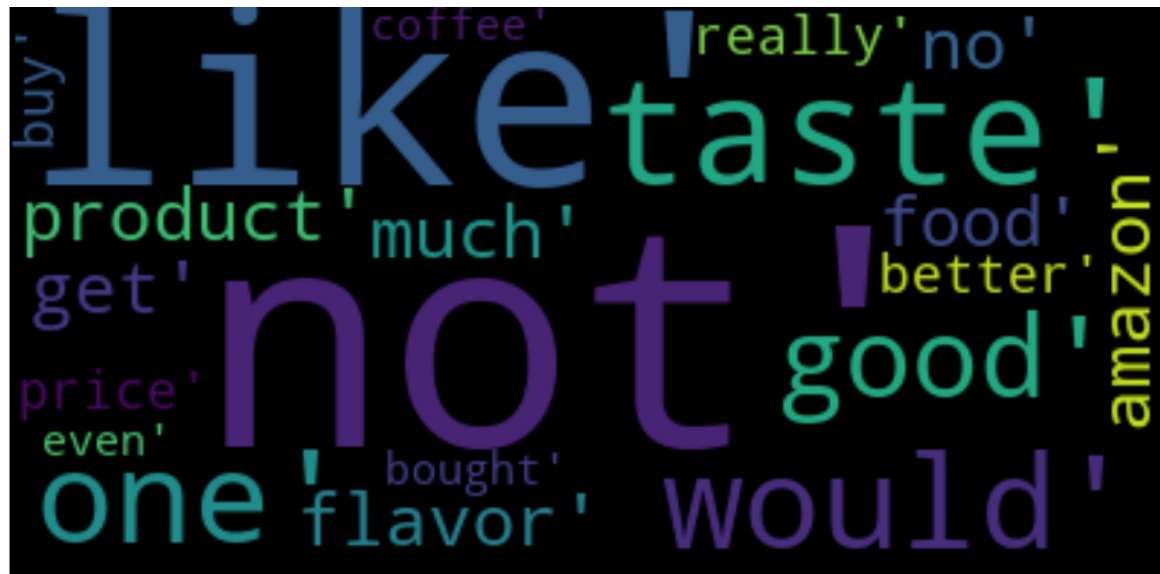
## Top 20 feature for BAG OF WORDS

```
In [42]: features = vectorizer.get_feature_names()
coef = RFX.feature_importances_
coef_df = pd.DataFrame({'word': features, 'coefficient': coef}, index =
None)
df = coef_df.sort_values("coefficient", ascending = False)[:20]
v=np.array(df['word'])
v
```

```
Out[42]: array(['not', 'like', 'taste', 'would', 'one', 'good', 'product',
'flavor', 'no', 'get', 'amazon', 'much', 'food', 'really', 'pric
e',
'buy', 'better', 'bought', 'coffee', 'even'], dtype=object)
```

```
In [43]: from wordcloud import WordCloud, STOPWORDS
stopwords = set(STOPWORDS)

wordcloud = WordCloud(max_words=10000).generate(str(v))
plt.figure(figsize = (15, 15), facecolor = None)
plt.imshow(wordcloud, interpolation="bilinear")
plt.axis("off")
plt.tight_layout(pad = 0)
plt.show()
```



## TF-IDF XGBOOST

```
In [44]: X=preprocessed_reviews  
Y=final['Score']
```

```
In [45]: from sklearn.model_selection import train_test_split  
X_train,X_test,y_train,y_test=train_test_split(X,Y,test_size=0.33,shuffle=False)  
X_train,X_cv,y_train,y_cv=train_test_split(X_train,y_train,test_size=0.33,shuffle=False)
```

```
In [46]: from sklearn.feature_extraction.text import TfidfVectorizer  
vectorizer_tf=TfidfVectorizer(min_df=10,max_features=10000,ngram_range=(1,2))  
vectorizer_tf.fit(X_train)  
  
X_train_tf=vectorizer_tf.transform(X_train)  
X_cv_tf=vectorizer_tf.transform(X_cv)
```



```
X_test_tf=vectorizer_tf.transform(X_test)

print("after TF-idf vectorizer")
print(np.shape(X_train_tf),np.shape(X_cv_tf),np.shape(X_test_tf))
print(y_train.shape,y_cv.shape,y_test.shape)
```

```
after TF-idf vectorizer
(39400, 10000) (19407, 10000) (28966, 10000)
(39400,) (19407,) (28966,)
```

```
In [47]: n_estimator=[20,40,60,80,90,100]
depth=[10,50,100,500,1000]
parameter={"n_estimators":n_estimator,"max_depth":depth}
RFX=GridSearchCV(xgb.XGBClassifier(),parameter,verbose=1,scoring='roc_auc',n_jobs=-1)
RFX.fit(X_train_tf,y_train)

opt_estimator,opt_depth=RFX.best_params_.get('n_estimators'),RFX.best_params_.get('max_depth')
print("The best optimized estimator:",opt_estimator)
print("The best optimized depth: ",opt_depth)

train_score=RFX.cv_results_.get('mean_train_score')
cv_score=RFX.cv_results_.get('mean_test_score')

df_heatmap=pd.DataFrame(train_score.reshape(6,5),index=n_estimator,columns=depth)
fig=plt.figure(figsize=(5,3))
heatmap=sns.heatmap(df_heatmap,annot=True)
plt.ylabel("n_estimators",size=20)
plt.xlabel("depth",size=20)
plt.title("CV data for train data")
plt.show()

df_heatmap=pd.DataFrame(cv_score.reshape(6,5),index=n_estimator,columns=depth)
fig=plt.figure(figsize=(5,4))
heatmap=sns.heatmap(df_heatmap,annot=True)
plt.ylabel("n_estimators",size=20)
```

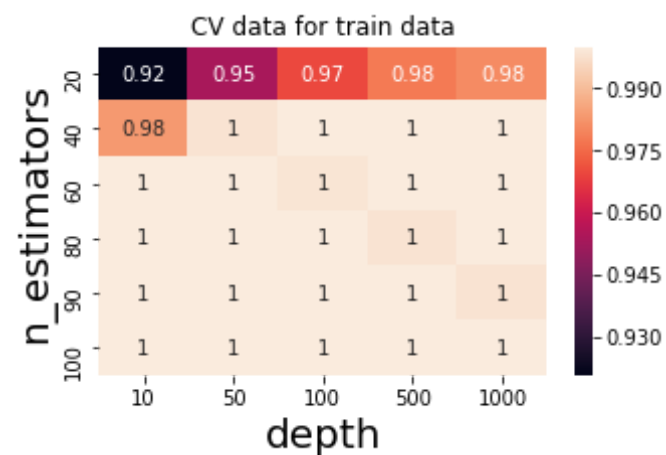
```
plt.xlabel("depth",size=20)
plt.title("cv data for test data")
plt.show()
```

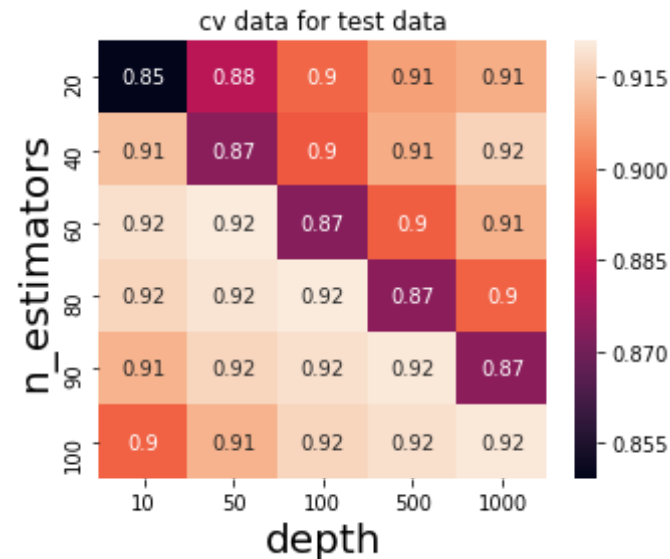
Fitting 3 folds for each of 30 candidates, totalling 90 fits

```
[Parallel(n_jobs=-1)]: Done 42 tasks | elapsed: 22.6min
[Parallel(n_jobs=-1)]: Done 90 out of 90 | elapsed: 88.4min finished
```

The best optimized estimator: 100

The best optimized depth: 100





```
In [48]: from sklearn.metrics import roc_auc_score
RFX=xgb.XGBClassifier(n_estimators=100,max_depth=100)
RFX.fit(X_train_tf,y_train)

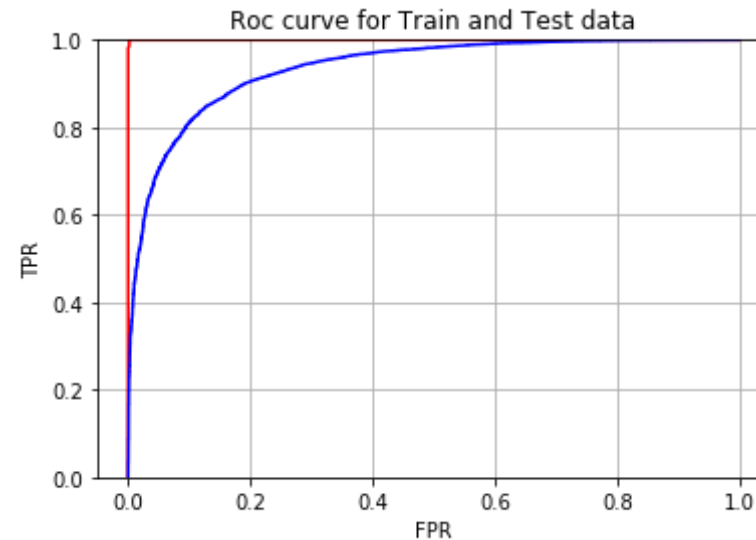
y_train_pred_proba=RFX.predict_proba(X_train_tf)
y_test_pred_proba=RFX.predict_proba(X_test_tf)

fpr,tpr,threshold=roc_curve(y_train,y_train_pred_proba[:,1])
fpr1,tpr1,threshold1=roc_curve(y_test,y_test_pred_proba[:,1])

print("The AUC value for Test data:",roc_auc_score(y_test,y_test_pred_p
roba[:,1]))

plt.plot(fpr,tpr,'r',label="Train data")
plt.plot(fpr1,tpr1,'b',label='Test data')
plt.ylim(0,1)
plt.grid(True)
plt.title("Roc curve for Train and Test data")
plt.xlabel("FPR")
plt.ylabel("TPR")
plt.show()
```

The AUC value for Test data: 0.935629290996717



```
In [49]: confusion_matrix(y_test,X_test_tf,y_train,X_train_tf)
```

```
C:\ProgramData\Anaconda3\lib\site-packages\sklearn\preprocessing\label.py:151: DeprecationWarning: The truth value of an empty array is ambiguous. Returning False, but in future this will result in an error. Use `array.size > 0` to check that an array is not empty.
```

```
if diff:
```

```
C:\ProgramData\Anaconda3\lib\site-packages\sklearn\preprocessing\label.py:151: DeprecationWarning: The truth value of an empty array is ambiguous. Returning False, but in future this will result in an error. Use `array.size > 0` to check that an array is not empty.
```

```
if diff:
```

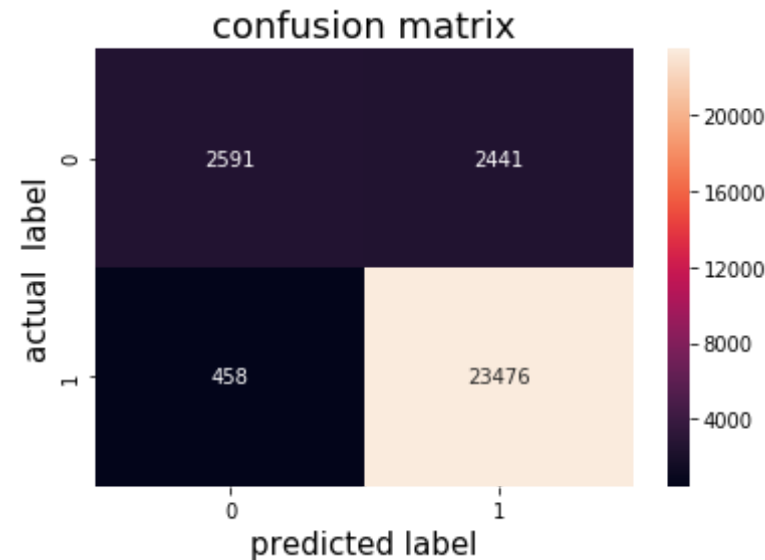
```
confusion matrix on test data
```

```
[[ 2591  2441]
 [   458 23476]]
```

```
*****
*****
*****
```

```
confusion matrix on train data
```

```
[[ 5724    14]
 [    0 33662]]
```



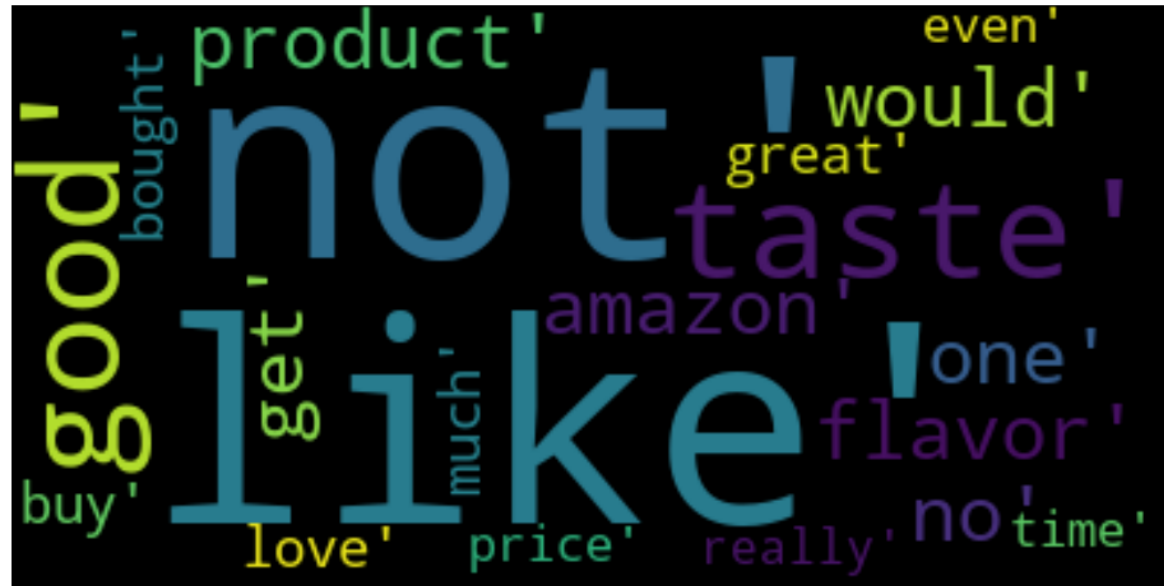
## TOP 20 FEATURES FOR TF-IDF

```
In [50]: features = vectorizer_tf.get_feature_names()
coef = RFX.feature_importances_
coef_df = pd.DataFrame({'word': features, 'coefficient': coef}, index =
None)
df = coef_df.sort_values("coefficient", ascending = False)[:20]
v=np.array(df['word'])
v
```

```
Out[50]: array(['not', 'like', 'taste', 'good', 'product', 'would', 'one',
'flavor', 'no', 'get', 'amazon', 'great', 'much', 'love', 'buy',
'bought', 'price', 'even', 'time', 'really'], dtype=object)
```

```
In [51]: from wordcloud import WordCloud, STOPWORDS
stopwords = set(STOPWORDS)
```

```
wordcloud = WordCloud(max_words=10000).generate(str(v))
plt.figure(figsize = (15, 15), facecolor = None)
plt.imshow(wordcloud, interpolation="bilinear")
plt.axis("off")
plt.tight_layout(pad = 0)
plt.show()
```



## AVG W2V XGBOOST

```
In [27]: X=preprocessed_reviews
        Y=final['Score']
```

```
In [28]: from sklearn.model_selection import train_test_split
        X_train,X_test,y_train,y_test=train_test_split(X,Y,test_size=0.33,shuffle=False)
        X_train,X_cv,y_train,y_cv=train_test_split(X_train,y_train,test_size=0.33,shuffle=False)
```

```
In [29]: i=0
list_of_sentence=[]
for sentence in preprocessed_reviews:
    list_of_sentence.append(sentence.split())
```

```
In [30]: sent_of_train=[]
for sent in X_train:
    sent_of_train.append(sent.split())
```

```
In [31]: sent_of_cv=[]
for sent in X_cv:
    sent_of_cv.append(sent.split())

sent_of_test=[]
for sent in X_test:
    sent_of_test.append(sent.split())

# Train your own Word2Vec model using your own train text corpus
# min_count = 5 considers only words that occurred at least 5 times
w2v_model=Word2Vec(sent_of_train,min_count=5,size=50, workers=4)

w2v_words = list(w2v_model.wv.vocab)
```

```
In [32]: train_vectors = [];
for sent in sent_of_train:
    sent_vec = np.zeros(50)
    cnt_words =0;
    for word in sent: #
        if word in w2v_words:
            vec = w2v_model.wv[word]
            sent_vec += vec
            cnt_words += 1
    if cnt_words != 0:
        sent_vec /= cnt_words
    train_vectors.append(sent_vec)

cv_vectors = [];
```

```

for sent in sent_of_cv:
    sent_vec = np.zeros(50)
    cnt_words = 0;
    for word in sent: #
        if word in w2v_words:
            vec = w2v_model.wv[word]
            sent_vec += vec
            cnt_words += 1
    if cnt_words != 0:
        sent_vec /= cnt_words
    cv_vectors.append(sent_vec)

# compute average word2vec for each review for X_test .
test_vectors = [];
for sent in sent_of_test:
    sent_vec = np.zeros(50)
    cnt_words = 0;
    for word in sent: #
        if word in w2v_words:
            vec = w2v_model.wv[word]
            sent_vec += vec
            cnt_words += 1
    if cnt_words != 0:
        sent_vec /= cnt_words
    test_vectors.append(sent_vec)

```

```

In [33]: X_train_wv=train_vectors
        X_cv_wv=cv_vectors
        X_test_wv=test_vectors

```

```

In [34]: from sklearn.preprocessing import StandardScaler
        standardised=StandardScaler(with_mean=False)
        X_train_wv=standardised.fit_transform(X_train_wv)
        X_cv_wv=standardised.transform(X_cv_wv)
        X_test_wv=standardised.transform(X_test_wv)

```



```

In [35]: from sklearn.model_selection import GridSearchCV, RandomizedSearchCV
import xgboost as xgb
n_estimator=[20,40,60,80,90,100]
depth=[10,50,100,500,1000]
parameter={"n_estimators":n_estimator,"max_depth":depth}
RFX=GridSearchCV(xgb.XGBClassifier(),parameter,verbose=1,scoring='roc_auc',n_jobs=-1)
RFX.fit(X_train_wv,y_train)

opt_estimator,opt_depth=RFX.best_params_.get('n_estimators'),RFX.best_params_.get('max_depth')
print("The best optimized estimator:",opt_estimator)
print("The best optimized depth: ",opt_depth)

train_score=RFX.cv_results_.get('mean_train_score')
cv_score=RFX.cv_results_.get('mean_test_score')

df_heatmap=pd.DataFrame(train_score.reshape(6,5),index=n_estimator,columns=depth)
fig=plt.figure(figsize=(5,3))
heatmap=sns.heatmap(df_heatmap,annot=True)
plt.ylabel("n_estimators",size=20)
plt.xlabel("depth",size=20)
plt.title("CV data for train data")
plt.show()

df_heatmap=pd.DataFrame(cv_score.reshape(6,5),index=n_estimator,columns=depth)
fig=plt.figure(figsize=(5,4))
heatmap=sns.heatmap(df_heatmap,annot=True)
plt.ylabel("n_estimators",size=20)
plt.xlabel("depth",size=20)
plt.title("cv data for test data")
plt.show()

```

Fitting 3 folds for each of 30 candidates, totalling 90 fits

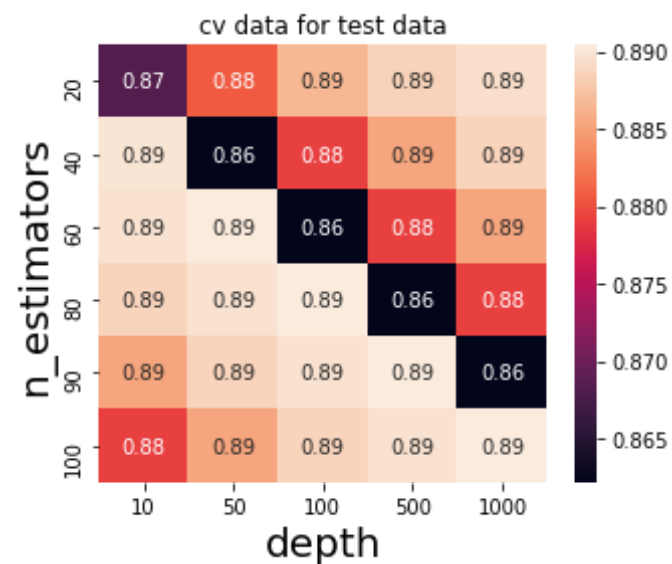
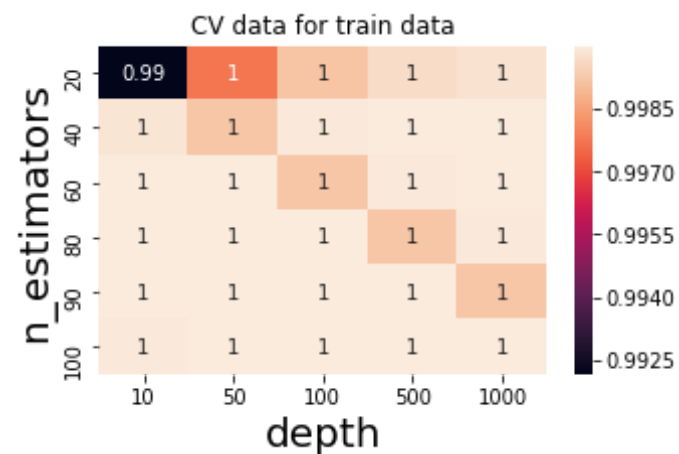
```

[Parallel(n_jobs=-1)]: Done 42 tasks      | elapsed: 8.4min
[Parallel(n_jobs=-1)]: Done 90 out of 90 | elapsed: 20.5min finished

```

The best optimized estimator: 100

The best optimized depth: 50



```
In [36]: from sklearn.metrics import roc_auc_score
RFX=rgb.XGBClassifier(n_estimators=100,max_depth=50)
RFX.fit(X_train_wv,y_train)
```

```

y_train_pred_proba=RFX.predict_proba(X_train_wv)
y_test_pred_proba=RFX.predict_proba(X_test_wv)

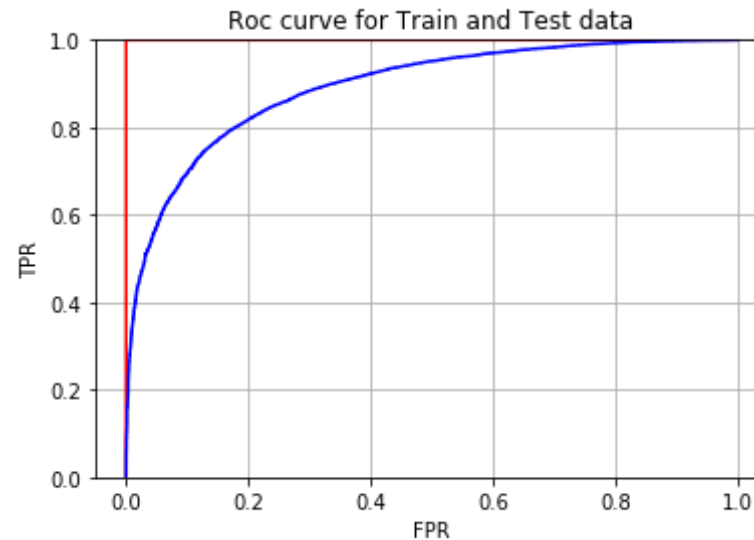
fpr,tpr,threshold=roc_curve(y_train,y_train_pred_proba[:,1])
fpr1,tpr1,threshold1=roc_curve(y_test,y_test_pred_proba[:,1])

print("The AUC value for Test data:",roc_auc_score(y_test,y_test_pred_p
roba[:,1]))

plt.plot(fpr,tpr,'r',label="Train data")
plt.plot(fpr1,tpr1,'b',label='Test data')
plt.ylim(0,1)
plt.grid(True)
plt.title("Roc curve for Train and Test data")
plt.xlabel("FPR")
plt.ylabel("TPR")
plt.show()

```

The AUC value for Test data: 0.8922104223618129



```

In [38]: from sklearn.metrics import confusion_matrix
y_test_pred=RFX.predict(X_test_wv)

```

```

y_train_pred=RFX.predict(X_train_wv)
cm_test=confusion_matrix(y_test,y_test_pred)
cm_train=confusion_matrix(y_train,y_train_pred)
print("confusion matrix on test data")
print(cm_test)
print("*"*200)
print("confusion matrix on train data")
print(cm_train)
import seaborn as sns
class_label=["0","1"]
df_cm=pd.DataFrame(cm_test,index=class_label,columns=class_label)
sns.heatmap(df_cm,annot=True,fmt='d')
plt.xlabel("predicted label",size=15)
plt.ylabel("actual label",size=15)
plt.title("confusion matrix",size=18)
plt.show()

```

confusion matrix on test data

```

[[ 2029  3003]
 [   724 23210]]

```

```

*****
*****
*****

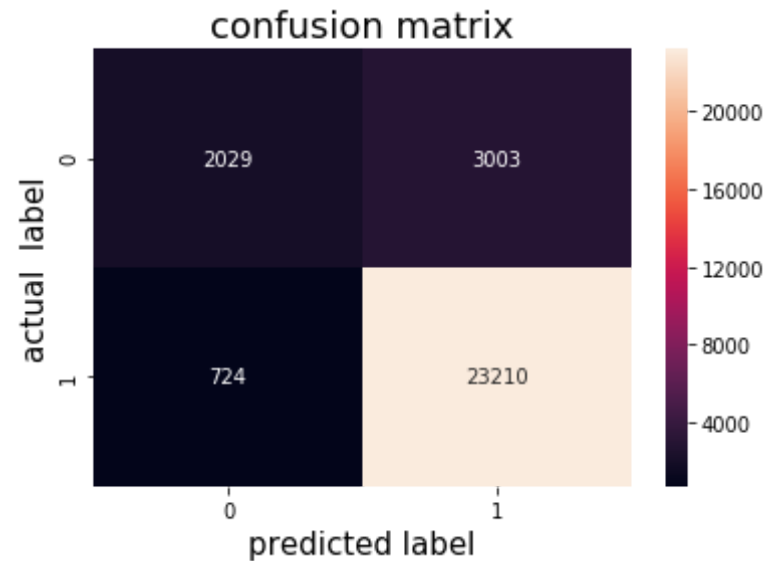
```

confusion matrix on train data

```

[[ 5727    11]
 [     0 33662]]

```



## TF-IDF W2V

```
In [39]: X=preprocessed_reviews  
Y=final["Score"]
```

```
In [40]: from sklearn.model_selection import train_test_split  
X_train, X_test, y_train, y_test = train_test_split(X, Y, test_size=0.33, shuffle=False) # this is random splitting  
X_train, X_cv, y_train, y_cv = train_test_split(X_train, y_train, test_size=0.33, shuffle=False)
```

```
In [41]: model = TfidfVectorizer()  
tf_idf_matrix = model.fit_transform(X_train)  
# 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]: 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

tfidf_train_vectors = []; # the tfidf-w2v for each sentence/review is stored in this list
row=0;
for sent in tqdm(sent_of_train): # 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 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_train_vectors.append(sent_vec)
    row += 1
```

100%|██| 39400/39400 [12:31<00:00, 52.45it/s]

```
In [43]: tfidf_cv_vectors = []; # the tfidf-w2v for each sentence/review is stored in this list
row=0;
for sent in tqdm(sent_of_cv): # 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 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_cv_vectors.append(sent_vec)
row += 1
```

```
100%|████████████████████████████████████████| 19407/19407 [19:01<00:00, 1
7.00it/s]
```

In [44]:

```
tfidf_test_vectors = []; # the tfidf-w2v for each sentence/review is st
ored in this list
row=0;
for sent in tqdm(sent_of_test): # 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/r
review
    for word in sent: # for each word in a review/sentence
        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_test_vectors.append(sent_vec)
row += 1
```

```
100%|████████████████████████████████████████| 28966/28966 [09:17<00:00, 5
1.93it/s]
```

```
In [45]: X_train_tw=tfidf_train_vectors
X_cv_tw=tfidf_cv_vectors
X_test_tw=tfidf_test_vectors
```

```
In [46]: from sklearn.preprocessing import StandardScaler
standardised=StandardScaler(with_mean=False)
X_train_tw=standardised.fit_transform(X_train_tw)
X_cv_tw=standardised.transform(X_cv_tw)
X_test_tw=standardised.transform(X_test_tw)
```

```
In [47]: from sklearn.model_selection import GridSearchCV,RandomizedSearchCV
import xgboost as xgb
n_estimator=[20,40,60,80,90,100]
depth=[10,50,100,500,1000]
parameter={"n_estimators":n_estimator,"max_depth":depth}
RFX=GridSearchCV(xgb.XGBClassifier(),parameter,verbose=1,scoring='roc_auc',n_jobs=-1)
RFX.fit(X_train_tw,y_train)

opt_estimator,opt_depth=RFX.best_params_.get('n_estimators'),RFX.best_params_.get('max_depth')
print("The best optimized estimator:",opt_estimator)
print("The best optimized depth: ",opt_depth)

train_score=RFX.cv_results_.get('mean_train_score')
cv_score=RFX.cv_results_.get('mean_test_score')

df_heatmap=pd.DataFrame(train_score.reshape(6,5),index=n_estimator,columns=depth)
fig=plt.figure(figsize=(5,3))
heatmap=sns.heatmap(df_heatmap,annot=True)
plt.ylabel("n_estimators",size=20)
plt.xlabel("depth",size=20)
plt.title("CV data for train data")
plt.show()

df_heatmap=pd.DataFrame(cv_score.reshape(6,5),index=n_estimator,columns=depth)
```



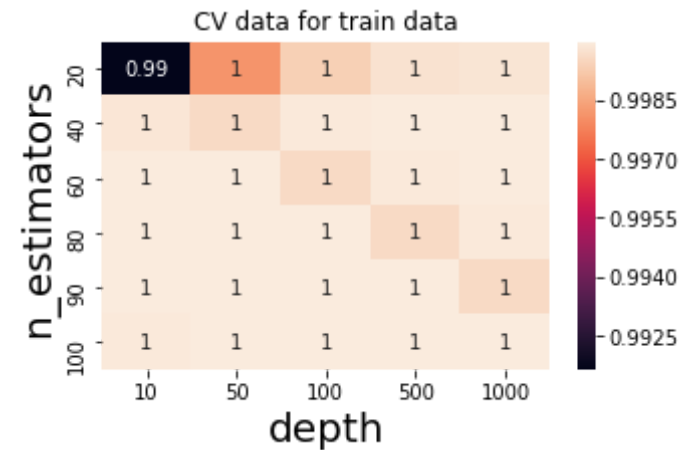
```
fig=plt.figure(figsize=(5,4))
heatmap=sns.heatmap(df_heatmap,annot=True)
plt.ylabel("n_estimators",size=20)
plt.xlabel("depth",size=20)
plt.title("cv data for test data")
plt.show()
```

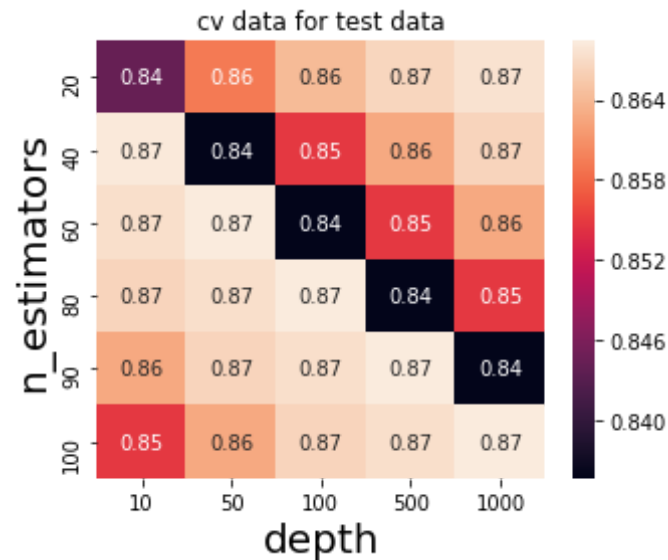
Fitting 3 folds for each of 30 candidates, totalling 90 fits

```
[Parallel(n_jobs=-1)]: Done 42 tasks      | elapsed: 10.2min
[Parallel(n_jobs=-1)]: Done 90 out of 90 | elapsed: 70.6min finished
```

The best optimized estimator: 100

The best optimized depth: 50





```
In [49]: from sklearn.metrics import roc_auc_score
RFX=xgb.XGBClassifier(n_estimators=100,max_depth=50)
RFX.fit(X_train_tw,y_train)

y_train_pred_proba=RFX.predict_proba(X_train_tw)
y_test_pred_proba=RFX.predict_proba(X_test_tw)

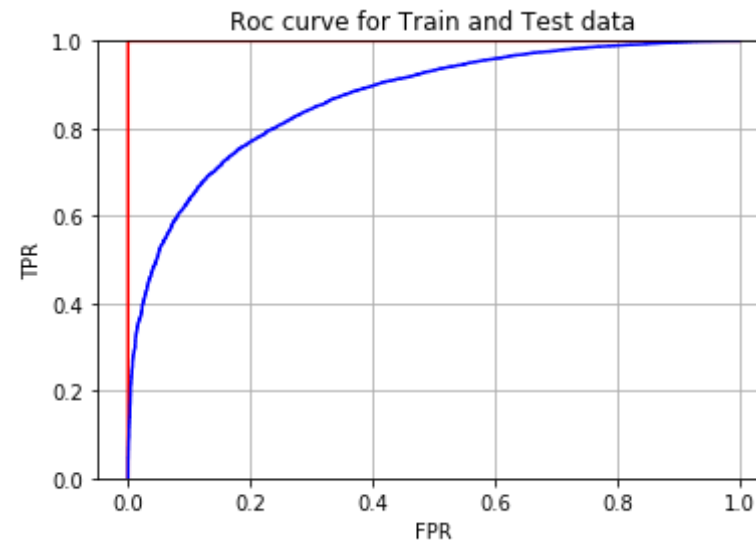
fpr, tpr, threshold=roc_curve(y_train,y_train_pred_proba[:,1])
fpr1, tpr1, threshold1=roc_curve(y_test,y_test_pred_proba[:,1])

print("The AUC value for Test data:", roc_auc_score(y_test,y_test_pred_p
roba[:,1]))

plt.plot(fpr,tpr,'r',label="Train data")
plt.plot(fpr1,tpr1,'b',label='Test data')
plt.ylim(0,1)
plt.grid(True)
plt.title("Roc curve for Train and Test data")
plt.xlabel("FPR")
plt.ylabel("TPR")
plt.show()
```

The AUC value for Test data: 0.8633333333333333

The AUC value for Test data: 0.8682067840110914



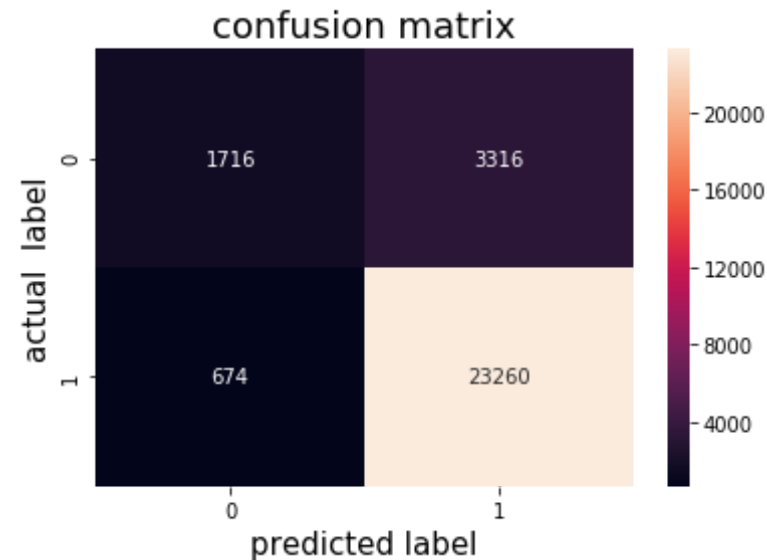
```
In [48]: from sklearn.metrics import confusion_matrix
y_test_pred=RFX.predict(X_test_tw)
y_train_pred=RFX.predict(X_train_tw)
cm_test=confusion_matrix(y_test,y_test_pred)
cm_train=confusion_matrix(y_train,y_train_pred)
print("confusion matrix on test data")
print(cm_test)
print("*"*200)
print("confusion matrix on train data")
print(cm_train)
import seaborn as sns
class_label=["0","1"]
df_cm=pd.DataFrame(cm_test,index=class_label,columns=class_label)
sns.heatmap(df_cm,annot=True,fmt='d')
plt.xlabel("predicted label",size=15)
plt.ylabel("actual label",size=15)
plt.title("confusion matrix",size=18)
plt.show()
```

confusion matrix on test data

```

[[ 1716  3316]
 [   674 23260]]
*****
*****
*****
confusion matrix on train data
[[ 5727    11]
 [     0 33662]]

```



## CONCLUSION

```

In [51]: print("XGB00ST")
from tabulate import tabulate
print(tabulate ([['BOW(100)', 100, 94],['TF-IDF(100)',100,93.5],['AVG-W
2V(50)',100,89] , ['TFIDF-W2V(50)',100,86]], headers=['Vectorizer(be
st_depth)', 'best_estimator', 'AUC_test']))

```

XGB00ST

Vectorizer(best\_depth)      best\_estimator      AUC\_test

BOW(100)	100	94
TF-IDF(100)	100	93.5
AVG-W2V(50)	100	89
TFIDF-W2V(50)	100	86

1. Xgboost is very very slow as compared all other model.
2. Tf-IDF is very best among the vectorizer which give the better accuracy.
3. we can improve accuracy and confusion matrix by taking more data points and feature engineering.

In [0]: *## Similarly you can do preprocessing for review summary also.*

#### Note: Data Leakage

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

## [5.1] Applying RF

### [5.1.1] Applying Random Forests on BOW, SET 1

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

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

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

### **[5.1.3] Applying Random Forests on TFIDF, SET 2**

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

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

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

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

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

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

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

## **[5.2] Applying GBDT using XGBOOST**

### **[5.2.1] Applying XGBOOST on BOW, SET 1**

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

### **[5.2.2] Applying XGBOOST on TFIDF, SET 2**

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

### **[5.2.3] Applying XGBOOST on AVG W2V, SET 3**

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

### **[5.2.4] Applying XGBOOST on TFIDF W2V, SET 4**

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

## **[6] Conclusions**

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