In [1]:
```
from google.colab import drive
import os

drive.mount('drive')
os.chdir('drive/My Drive/Assignments_AFR_2018')
```

```
Go to this URL in a browser: https://accounts.google.com/o/oauth2/auth?
client_id=947318989803-6bn6qk8qdgf4n4g3pfee6491hc0brc4i.apps.googleuser
content.com&redirect_uri=urn%3Aietf%3Awg%3Aoauth%3A2.0%3Aoob&scope=emai
l%20https%3A%2F%2Fwww.googleapis.com%2Fauth%2Fdocs.test%20https%3A%2F%2
Fwww.googleapis.com%2Fauth%2Fdrive%20https%3A%2F%2Fwww.googleapis.com%2
Fauth%2Fdrive.photos.readonly%20https%3A%2F%2Fwww.googleapis.com%2Faut
h%2Fpeopleapi.readonly&response_type=code

Enter your authorization code:
··········
Mounted at drive
```

Amazon Fine Food Reviews Analysis

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

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

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

Number of reviews: 568,454
Number of users: 256,059
Number of products: 74,258
Timespan: Oct 1999 - Oct 2012
Number of Attributes/Columns in data: 10

Attribute Information:

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

Objective:

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

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

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

[1]. Reading Data

[1.1] Loading the data

The dataset is available in two forms

1. .csv file

2. SQLite Database

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

Here as we only want to get the global sentiment of the recommendations (positive or negative), we will purposefully ignore all Scores equal to 3. If the score is above 3, then the recommendation wil be set to "positive". Otherwise, it will be set to "negative".

In [2]:
```
%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
```

```
paramiko missing, opening SSH/SCP/SFTP paths will be disabled.  `pip in
stall paramiko` to suppress
```

In [3]:

```
# 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 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 Sco
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 """, con)

# Give reviews with Score>3 a positive rating(1), and reviews with a sc
ore<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 (525814, 10)
```

Out[3]:

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

In [0]:

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

In [0]:

```
print(display.shape)
```

```
display.head()
```

(80668, 7)

Out[0]:

	UserId	ProductId	ProfileName	Time	Score	Text	COU
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 [0]:
```
display[display['UserId']=='AZY10LLTJ71NX']
```
Out[0]:

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

In [0]:
```
display['COUNT(*)'].sum()
```
Out[0]: 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 [0]:
```
display= pd.read_sql_query("""
SELECT *
FROM Reviews
WHERE Score != 3 AND UserId="AR5J8UI46CURR"
ORDER BY ProductID
""", con)
display.head()
```
Out[0]:

	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 delelte the others. for eg. in the above just the review for ProductId=B000HDL1RQ remains. This method ensures that there is only one representative for each product and deduplication without sorting would lead to possibility of different representatives still existing for the same product.

In [0]:
```
sample_data = filtered_data.head(50000)
```

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

In [6]:
```
#Deduplication of entries
final=sorted_data.drop_duplicates(subset={"UserId","ProfileName","Time"
,"Text"}, keep='first', inplace=False)
final.shape
```

Out[6]:
```
(46072, 10)
```

In [7]:
```
#Checking to see how much % of data still remains
(final['Id'].size*1.0)/(filtered_data['Id'].size*1.0)*100
```

Out[7]:
```
8.762033722951463
```

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

In [0]:
```
display= pd.read_sql_query("""
SELECT *
FROM Reviews
WHERE Score != 3 AND Id=44737 OR Id=64422
ORDER BY ProductID
""", con)

display.head()
```

Out[0]:

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

In [0]:
```
final=final[final.HelpfulnessNumerator<=final.HelpfulnessDenominator]
```

In [0]:
```
#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()
```

```
(46072, 10)
```

Out[0]:
```
1    38480
0     7592
Name: Score, dtype: int64
```

[3] Preprocessing

[3.1]. Preprocessing Review Text

Now that we have finished deduplication our data requires some preprocessing before we go on further with analysis and making the prediction model.

Hence in the Preprocessing phase we do the following in the order below:-

1. Begin by removing the html tags
2. Remove any punctuations or limited set of special characters like , or . or # etc.
3. Check if the word is made up of english letters and is not alpha-numeric
4. Check to see if the length of the word is greater than 2 (as it was researched that there is no adjective in 2-letters)
5. Convert the word to lowercase
6. Remove Stopwords
7. Finally Snowball Stemming the word (it was obsereved to be better than Porter Stemming)

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

In [0]:
```
# 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)
```

```
This was a really good idea and the final product is outstanding. I use
the decals on my car window and everybody asks where i bought the decal
s i made.  Two thumbs up!
==================================================
These are thin,crisp, fragrant cookies and are very delicious and tast
y. They are excellent with a glass of cold almond milk or hot herbal te
a. (my choices) If you like ginger snaps you will love Lars ginger snap
s.
==================================================
Green Mountain "Nantucket Blend" K-Cups make a very good cup of coffee
in my <a href="http://www.amazon.com/gp/product/B000AQPMHA">Keurig B-40
B40 Elite Gourmet Single-Cup Home-Brewing System</a>. This is a very sm
ooth tasting brew that my wife prefers over the <a href="http://www.ama
zon.com/gp/product/B0029XDZIK">Coffee People, Donut Shop K-Cups for Keu
rig Brewers (Pack of 50) [Amazon Frustration-Free Packaging</a>] I gene
rally drink in the morning.<br /><br />These are good on both "Small" a
nd "Large" cup settings as well.<br /><br />Highly Recommended!<br /><b
r />CFH
==================================================
Besides being smaller than runts, they look the same and have the same
consistency.  Unfortunately, they taste nothing like banana runts...nor
do they even taste good.  Yucky stuff.  Trying to return with vendor.
==================================================
```

In [0]:

```
# remove urls from text python: https://stackoverflow.com/a/40823105/40
```

```
84039
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)
```

```
This was a really good idea and the final product is outstanding. I use
the decals on my car window and everybody asks where i bought the decal
s i made.  Two thumbs up!
```

In [0]:

```
# 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)
```

In [0]:

```
# 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 [0]:

```
sent_1500 = decontracted(sent_1500)
print(sent_1500)
print("="*50)
```

In [0]:

```
#remove words with numbers python: https://stackoverflow.com/a/1808237
0/4084039
sent_0 = re.sub("\S*\d\S*", "", sent_0).strip()
print(sent_0)
```

```
This was a really good idea and the final product is outstanding. I use
the decals on my car window and everybody asks where i bought the decal
s i made.  Two thumbs up!
```

In [0]:

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

```
Green Mountain Nantucket Blend K Cups make a very good cup of coffee in
my a href http www amazon com gp product B000AQPMHA Keurig B 40 B40 Eli
te Gourmet Single Cup Home Brewing System a This is a very smooth tasti
ng brew that my wife prefers over the a href http www amazon com gp pro
duct B0029XDZIK Coffee People Donut Shop K Cups for Keurig Brewers Pack
of 50 Amazon Frustration Free Packaging a I generally drink in the morn
```

ing br br These are good on both Small and Large cup settings as well b
r br Highly Recommended br br CFH

In [0]:

```
# https://gist.github.com/sebleier/554280
# we are removing the words from the stop words list: 'no', 'nor', 'no
t'
# <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 revmoved in
 the 1st step

stopwords= set(['br', 'the', 'i', 'me', 'my', 'myself', 'we', 'our', 'o
urs', 'ourselves', 'you', "you're", "you've",\
            "you'll", "you'd", 'your', 'yours', 'yourself', 'yourselve
s', 'he', 'him', 'his', 'himself', \
            'she', "she's", 'her', 'hers', 'herself', 'it', "it's", 'it
s', 'itself', 'they', 'them', 'their',\
            'theirs', 'themselves', 'what', 'which', 'who', 'whom', 'th
is', 'that', "that'll", 'these', 'those', \
            'am', 'is', 'are', 'was', 'were', 'be', 'been', 'being', 'h
ave', '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', 'h
ow', 'all', 'any', 'both', 'each', 'few', 'more',\
            'most', 'other', 'some', 'such', 'only', 'own', 'same', 's
o', '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', "is
n'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 [10]:

```
from bs4 import BeautifulSoup
# Combining all the above stundents
from tqdm import tqdm
preprocessed_reviews = []
# tqdm is for printing the status bar
for sentance in tqdm(final['Text'].values):
    sentance = re.sub(r"http\S+", "", sentance)
    sentance = BeautifulSoup(sentance, 'lxml').get_text()
    sentance = decontracted(sentance)
    sentance = re.sub("\S*\d\S*", "", sentance).strip()
    sentance = re.sub('[^A-Za-z]+', ' ', sentance)
    # https://gist.github.com/sebleier/554280
    sentance = ' '.join(e.lower() for e in sentance.split() if e.lower
() not in stopwords)
    preprocessed_reviews.append(sentance.strip())
```

```
100%|██████████| 46072/46072 [00:20<00:00, 2233.63it/s]
```

In [0]:

```
preprocessed_reviews[1500]
```

Out[0]:

```
'wow far two two star reviews one obviously no idea ordering wants cris
py cookies hey sorry reviews nobody good beyond reminding us look order
ing chocolate oatmeal cookies not like combination not order type cooki
e find combo quite nice really oatmeal sort calms rich chocolate flavor
gives cookie sort coconut type consistency let also remember tastes dif
fer given opinion soft chewy cookies advertised not crispy cookies blur
b would say crispy rather chewy happen like raw cookie dough however no
t see taste like raw cookie dough soft however confusion yes stick toge
ther soft cookies tend not individually wrapped would add cost oh yeah
chocolate chip cookies tend somewhat sweet want something hard crisp su
ggest nabiso ginger snaps want cookie soft chewy tastes like combinatio
n chocolate oatmeal give try place second order'
```

In [0]:

```
final["Clean_text"] = preprocessed_reviews
```

[3.2] Preprocessing Review Summary

In [0]:
```
## Similartly you can do preprocessing for review summary also.
```

[4] Featurization

[4.1] BAG OF WORDS

In [0]:
```
#BoW
count_vect = CountVectorizer() #in scikit-learn
count_vect.fit(preprocessed_reviews)
print("some feature names ", count_vect.get_feature_names()[:10])
print('='*50)

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

```
some feature names  ['aa', 'aahhhs', 'aback', 'abandon', 'abates', 'abb
ott', 'abby', 'abdominal', 'abiding', 'ability']
==================================================
the type of count vectorizer  <class 'scipy.sparse.csr.csr_matrix'>
the shape of out text BOW vectorizer  (4986, 12997)
the number of unique words  12997
```

[4.2] Bi-Grams and n-Grams.

In [0]:
```
#bi-gram, tri-gram and n-gram

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

```
# please do read the CountVectorizer documentation http://scikit-learn.org/stable/modules/generated/sklearn.feature_extraction.text.CountVectorizer.html

# you can choose these numebrs min_df=10, max_features=5000, of your choice
count_vect = CountVectorizer(ngram_range=(1,2), min_df=10, max_features=5000)
final_bigram_counts = count_vect.fit_transform(preprocessed_reviews)
print("the type of count vectorizer ",type(final_bigram_counts))
print("the shape of out text BOW vectorizer ",final_bigram_counts.get_shape())
print("the number of unique words including both unigrams and bigrams ", final_bigram_counts.get_shape()[1])
```

```
the type of count vectorizer  <class 'scipy.sparse.csr.csr_matrix'>
the shape of out text BOW vectorizer  (4986, 3144)
the number of unique words including both unigrams and bigrams  3144
```

[4.3] TF-IDF

In [0]:

```
tf_idf_vect = TfidfVectorizer(ngram_range=(1,2), min_df=10)
tf_idf_vect.fit(preprocessed_reviews)
print("some sample features(unique words in the corpus)",tf_idf_vect.get_feature_names()[0:10])
print('='*50)

final_tf_idf = tf_idf_vect.transform(preprocessed_reviews)
print("the type of count vectorizer ",type(final_tf_idf))
print("the shape of out text TFIDF vectorizer ",final_tf_idf.get_shape())
print("the number of unique words including both unigrams and bigrams ", final_tf_idf.get_shape()[1])
```

```
some sample features(unique words in the corpus) ['ability', 'able', 'able find', 'able get', 'absolute', 'absolutely', 'absolutely delicious', 'absolutely love', 'absolutely no', 'according']
==================================================
```

```
the type of count vectorizer  <class 'scipy.sparse.csr.csr_matrix'>
the shape of out text TFIDF vectorizer  (4986, 3144)
the number of unique words including both unigrams and bigrams  3144
```

[4.4] Word2Vec

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

In [0]:
```
# 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 ~9Gb, so please do this step only if you have >12G of ram
# we will provide a pickle file wich contains a dict ,
# and it contains all our courpus words as keys and  model[word] as val
ues
# To use this code-snippet, download "GoogleNews-vectors-negative300.bi
n"
# from https://drive.google.com/file/d/0B7XkCwpI5KDYNlNUTTlSS21pQmM/edi
t
# it's 1.9GB in size.


# http://kavita-ganesan.com/gensim-word2vec-tutorial-starter-code/#.W17
SRFAzZPY
# you can comment this whole cell
# or change these varible 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 occured atleast 5 times
    w2v_model=Word2Vec(list_of_sentance,min_count=5,size=50, workers=4)
    print(w2v_model.wv.most_similar('great'))
    print('='*50)
    print(w2v_model.wv.most_similar('worst'))

elif want_to_use_google_w2v and is_your_ram_gt_16g:
    if os.path.isfile('GoogleNews-vectors-negative300.bin'):
        w2v_model=KeyedVectors.load_word2vec_format('GoogleNews-vectors
-negative300.bin', binary=True)
        print(w2v_model.wv.most_similar('great'))
        print(w2v_model.wv.most_similar('worst'))
    else:
        print("you don't have gogole's word2vec file, keep want_to_trai
n_w2v = True, to train your own w2v ")
```

```
[('snack', 0.9951335191726685), ('calorie', 0.9946465492248535), ('wond
erful', 0.9946032166481018), ('excellent', 0.9944332838058472), ('espec
ially', 0.9941144585609436), ('baked', 0.9940600395202637), ('salted',
0.994047224521637), ('alternative', 0.9937226176261902), ('tasty', 0.99
36816692352295), ('healthy', 0.9936649799346924)]
==================================================
[('varieties', 0.9994194507598877), ('become', 0.9992934465408325), ('p
opcorn', 0.9992750883102417), ('de', 0.9992610216140747), ('miss', 0.99
92451071739197), ('melitta', 0.999218761920929), ('choice', 0.999210238
4567261), ('american', 0.9991837739944458), ('beef', 0.999178051948547
4), ('finish', 0.9991567134857178)]
```

In [0]:

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

```
number of words that occured minimum 5 times  3817
sample words  ['product', 'available', 'course', 'total', 'pretty', 'st
inky', 'right', 'nearby', 'used', 'ca', 'not', 'beat', 'great', 'receiv
ed', 'shipment', 'could', 'hardly', 'wait', 'try', 'love', 'call', 'ins
tead', 'removed', 'easily', 'daughter', 'designed', 'printed', 'use',
'car', 'windows', 'beautifully', 'shop', 'program', 'going', 'lot', 'fu
n', 'everywhere', 'like', 'tv', 'computer', 'really', 'good', 'idea',
```

```
'final', 'outstanding', 'window', 'everybody', 'asks', 'bought', 'made']
```

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

[4.4.1.1] Avg W2v

In [0]:
```python
# average Word2Vec
# compute average word2vec for each review.
sent_vectors = []; # the avg-w2v for each sentence/review is stored in this list
for sent in tqdm(list_of_sentance): # for each review/sentence
    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)
print(len(sent_vectors))
print(len(sent_vectors[0]))
```

```
100%|██████████████████████████████████████████| 4986/4986 [00:03<00:00, 1330.47it/s]
```

```
4986
50
```

[4.4.1.2] TFIDF weighted W2v

In [0]:
```python
# 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 [0]:
```python
# 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

tfidf_sent_vectors = []; # the tfidf-w2v for each sentence/review is stored in this list
row=0;
for sent in tqdm(list_of_sentance): # 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 courpus
            # sent.count(word) = tf valeus 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
```

```
100%|██████████████████████████████████████████████████
████████| 4986/4986 [00:20<00:00, 245.63it/s]
```

[5] Assignment 4: Apply Naive Bayes

1. **Apply Multinomial NaiveBayes 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)

2. **The hyper paramter tuning(find best Alpha)**
    - Find the best hyper parameter which will give the maximum AUC value
    - Consider a wide range of alpha values for hyperparameter tuning, start as low as 0.00001
    - 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**
    - Find the top 10 features of positive class and top 10 features of negative class for both feature sets Set 1 and Set 2 using absolute values of `coef_` parameter of MultinomialNB and print their corresponding feature names

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. Here on X-

axis you will have alpha values, since they have a wide range, just to represent those alpha values on the graph, apply log function on those alpha values.

- 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.
- Along with plotting ROC curve, you need to print the confusion matrix with predicted and original labels of test data points. Please visualize your confusion matrices using seaborn heatmaps.

6. **Conclusion**

- 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 link

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.

Applying Multinomial Naive Bayes

In [0]:
```
# short final data based on time
final = final.sort_values('Time', axis=0, ascending=True, inplace=False
, kind='quicksort', na_position='last')
#split data in train, test and cv before using it to avoid data leakage
from sklearn.model_selection import train_test_split
```

```
X = final['Clean_text']
y = final['Score']

X_train,X_test,y_train,y_test = train_test_split(X,y,test_size=.5,rando
m_state=0)
X_cv,X_test,y_cv,y_test = train_test_split(X_test,y_test,test_size=.5,r
andom_state=0)
```

[5.1] Applying Naive Bayes on BOW, SET 1

In [0]:

```
# Please write all the code with proper documentation
from sklearn.naive_bayes import MultinomialNB
from sklearn.model_selection import cross_val_score
from sklearn.metrics import roc_auc_score
from sklearn.metrics import confusion_matrix

#function to find an optimal value of k,AUC,ROC,confusion matrix
def NB_model(x_train,x_cv,x_test,y_train,y_cv,y_test):

    alpha = np.arange(0.00001,2,0.001)
    cv_score = [] #list to store values of k
    pred_cv = []
    pred_train = []
    opt_a = 0
    max_auc = -1
    for a in tqdm(alpha):
        nb = MultinomialNB(alpha=a)
        nb.fit(x_train,y_train)

        prob_cv = nb.predict_proba(x_cv)
        prob_tr = nb.predict_proba(x_train)
        prob_cv = prob_cv[:,1]
        prob_tr = prob_tr[:,1]
        auc_cv = roc_auc_score(y_cv,prob_cv)
        auc_tr = roc_auc_score(y_train,prob_tr)
        pred_cv.append(auc_cv)
        pred_train.append(auc_tr)
```

```
        if max_auc < auc_cv:
            max_auc = auc_cv
            opt_a = a

    #AUC curve
    plt.plot(alpha,pred_cv,'r',label='CV')
    plt.plot(alpha,pred_train,'g',label='train')
    plt.legend(loc='upper right')
    plt.title('Alpha vs AUC Score')
    plt.ylabel('AUC Score')
    plt.xlabel('Alpha')
    plt.show()

    print("optimal a: ",opt_a)

def test(x_train,x_cv,x_test,y_train,y_cv,y_test,opt_a):
    nb = MultinomialNB(alpha=opt_a)
    nb.fit(x_train,y_train)
    prob_test = nb.predict_proba(x_test)
    prob_test = prob_test[:,1]

    prob_train = nb.predict_proba(x_train)
    prob_train = prob_train[:,1]
    print("AUC Score: {}".format(roc_auc_score(y_test,prob_test)))

    #ROC curve
    fpr_tr,tpr_tr,thres_tr = roc_curve(y_train,prob_train)
    fpr,tpr,thres = roc_curve(y_test,prob_test)
    plt.plot([0,0],[1,1],linestyle='--')
    plt.plot(fpr,tpr,'r',marker='.',label='test')
    plt.plot(fpr_tr,tpr_tr,'b',marker='.',label='train')
    plt.legend(loc='upper right')
    plt.title("ROC curve")
    plt.show()
    print("The optimal value of k is {}".format(opt_a))

    #confusion matrix fortrain and test
    print("Confusion matrix for train data")
    predict_tr = nb.predict(x_train)
```

```
    confu_metrix_(predict_tr,y_train)

    print("Confusion matrix for test data")
    predict_te = nb.predict(x_test)
    confu_metrix_(predict_te,y_test)


def confu_metrix_(predict,y):
    confu_metrix = confusion_matrix(predict,y)
    confu_df = pd.DataFrame(confu_metrix,index=["-ve","+ve"],columns=[
"-ve","+ve"])
    sns.heatmap(confu_df,annot=True,fmt='d',cmap='viridis')
    plt.title("Confusion matrix")
    plt.xlabel("predicted label")
    plt.ylabel("True label")
    plt.show()
```

In [0]:

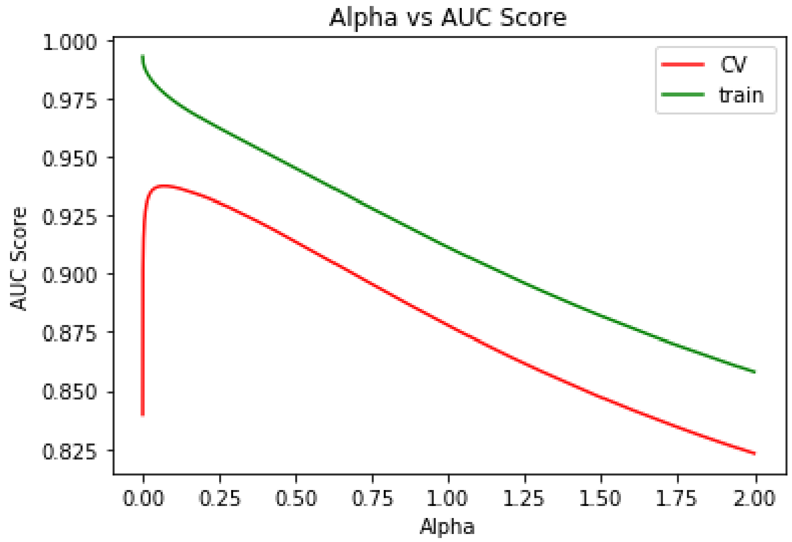
```
#BOW
count_vec = CountVectorizer()
bow_train = count_vec.fit_transform(X_train)
bow_test = count_vec.transform(X_test)
bow_cv = count_vec.transform(X_cv)

#normalize
from sklearn import preprocessing
bow_train=preprocessing.normalize(bow_train)
bow_cv=preprocessing.normalize(bow_cv)
bow_test=preprocessing.normalize(bow_test)
```

In [15]:

```
NB_model(bow_train,bow_cv,bow_test,y_train,y_cv,y_test)
```

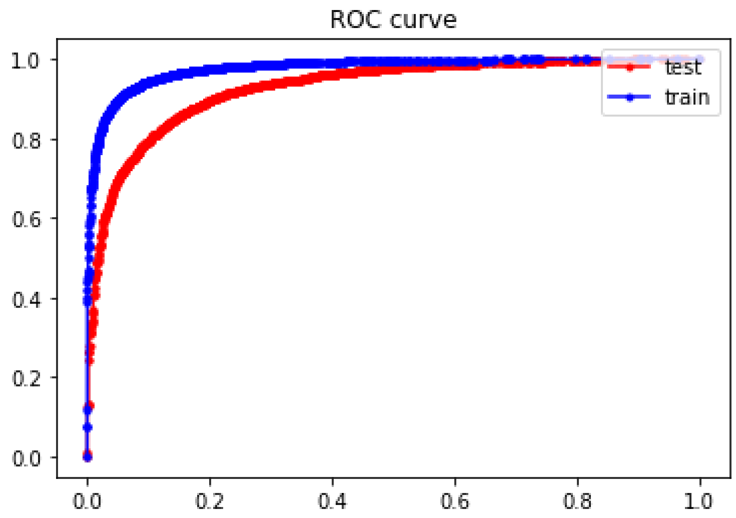
```
100%|██████████| 2000/2000 [01:10<00:00, 28.41it/s]
```



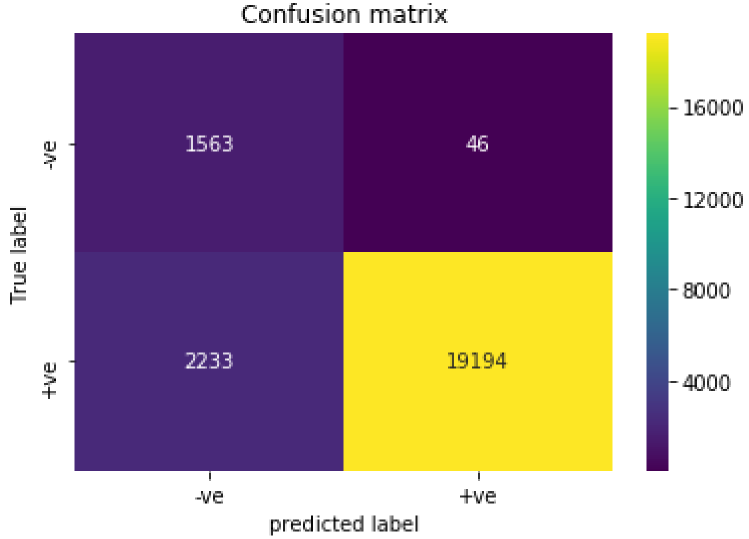

```
optimal a:  0.06601
```

In [18]:
```
test(bow_train,bow_cv,bow_test,y_train,y_cv,y_test,0.06601)
```

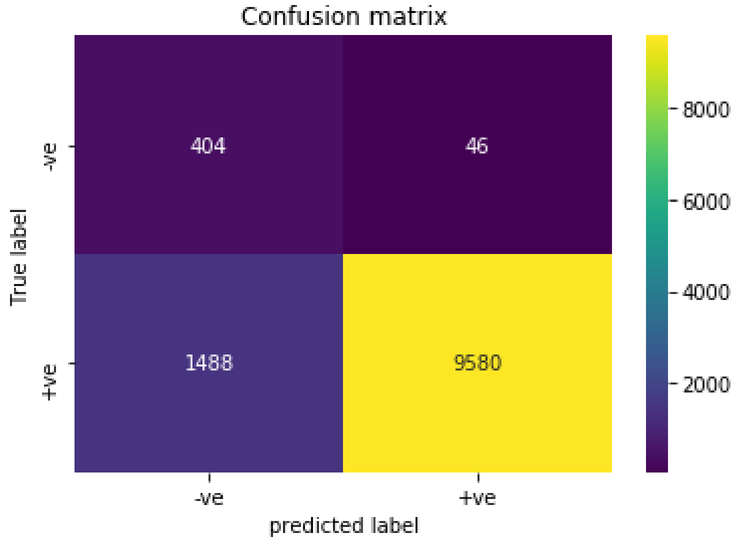
```
AUC Score: 0.9286461382996808
```



```
The optimal value of k is 0.06601
Confusion matrix for train data
```




```
Confusion matrix for test data
```



[5.1.1] Top 10 important features of positive class from SET 1

In [0]:
```
# Please write all the code with proper documentation
Nb = MultinomialNB(alpha=0.06601)
Nb.fit(bow_train,y_train)




pos_features = Nb.feature_log_prob_[1,:].argsort()
neg_features = Nb.feature_log_prob_[0,:].argsort()
```

In [22]:
```
print(np.take(count_vec.get_feature_names(),pos_features[:10]))
```

```
['piled' 'calamares' 'redesigned' 'redenbacker' 'redeeming' 'calculate'
 'calculation' 'gurgle' 'caledonian' 'calendar']
```

[5.1.2] Top 10 important features of negative class from SET 1

In [23]:
```
# Please write all the code with proper documentation
print(np.take(count_vec.get_feature_names(),neg_features[:10]))
```

```
['licoricey' 'nad' 'nacl' 'nachos' 'nabob' 'nabiso' 'nabiscos' 'nabisc
o'
 'nabbed' 'na']
```

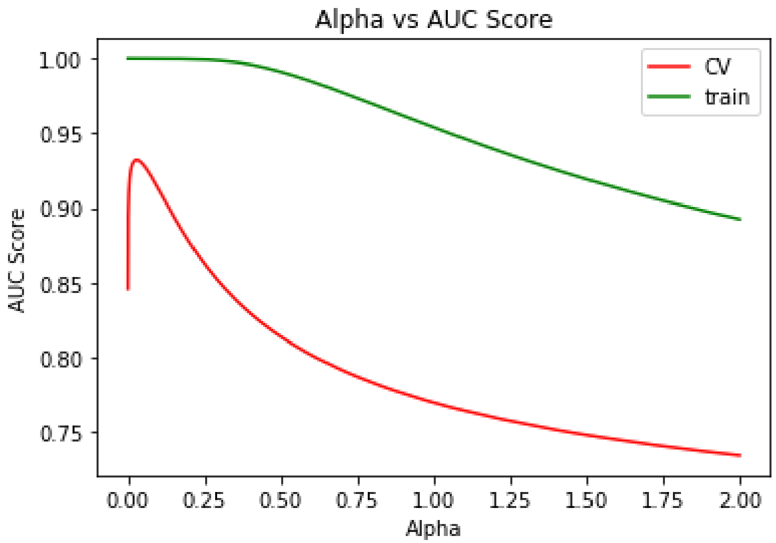
[5.2] Applying Naive Bayes on TFIDF, SET 2

In [0]:
```
#TFIDF
tf_idf_vect = TfidfVectorizer(ngram_range=(1,2))
tf_idf_train = preprocessing.normalize(tf_idf_vect.fit_transform(X_trai
n))
tf_idf_test = preprocessing.normalize(tf_idf_vect.transform(X_test))
tf_idf_cv = preprocessing.normalize(tf_idf_vect.transform(X_cv))
```

In [25]:
```
# Please write all the code with proper documentation
```

```
NB_model(tf_idf_train,tf_idf_cv,tf_idf_test,y_train,y_cv,y_test)
```

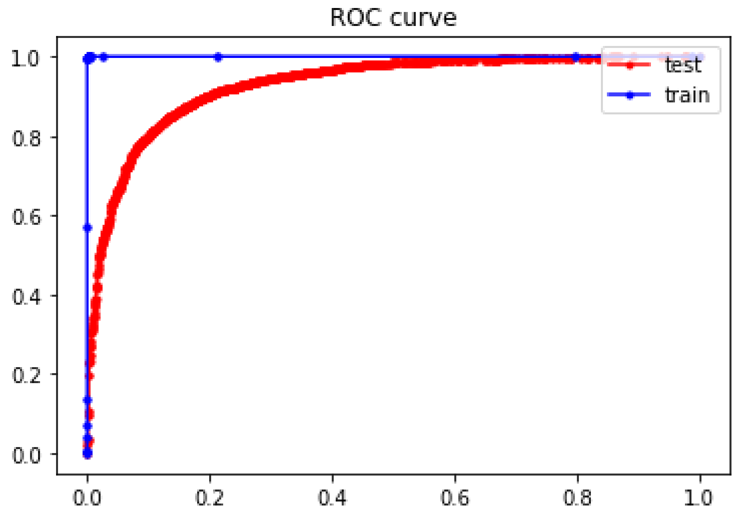
```
100%|██████████| 2000/2000 [03:33<00:00,  9.10it/s]
```



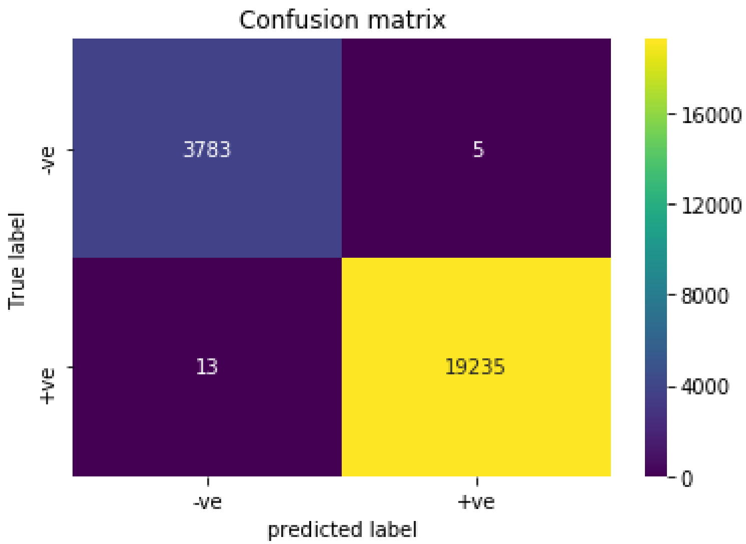

```
optimal a:  0.02901
```

In [26]:
```
test(tf_idf_train,tf_idf_cv,tf_idf_test,y_train,y_cv,y_test,0.02901)
```

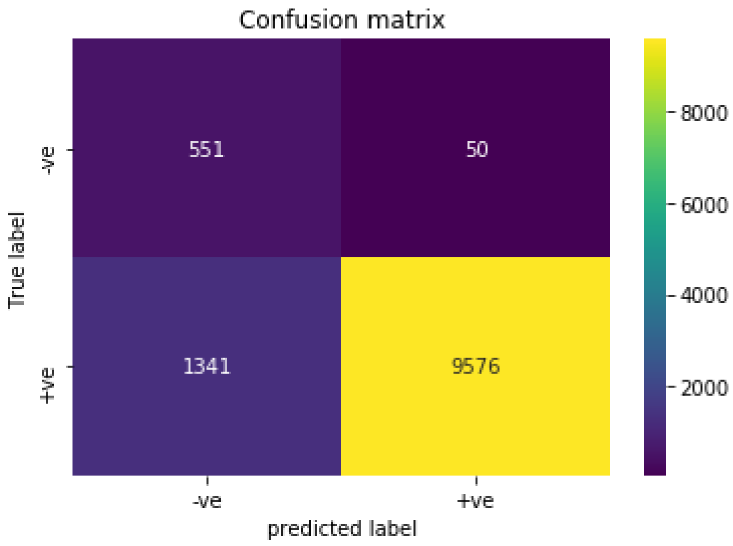
```
AUC Score: 0.9305983255796383
```




```
The optimal value of k is 0.02901
Confusion matrix for train data
```




```
Confusion matrix for test data
```



[5.2.1] Top 10 important features of positive class from SET 2

In [0]:
```
# Please write all the code with proper documentation
Nb = MultinomialNB(alpha=0.03001)
Nb.fit(tf_idf_train,y_train)



pos_features_tfidf = Nb.feature_log_prob_[1,:].argsort()
neg_features_tfidf = Nb.feature_log_prob_[0,:].argsort()
```

In [28]:
```
print(np.take(tf_idf_vect.get_feature_names(),pos_features_tfidf[:10]))
```

```
['root technically' 'buy hamburger' 'buy halves' 'question freshness'
 'question exactly' 'buy gushers' 'even pouring' 'question bother'
 'question health' 'even preparing']
```

[5.2.2] Top 10 important features of negative class from SET 2

In [29]:
```
# Please write all the code with proper documentation
print(np.take(tf_idf_vect.get_feature_names(),neg_features_tfidf[:10]))
```

```
['longer insulin' 'ounce usual' 'ounce unsweetened' 'ounce typical'
 'ounce two' 'ounce travel' 'ounce trader' 'ounce tootsie'
 'ounce tomatoes' 'ounce tea']
```

[6] Conclusions

In [30]:
```
# Please compare all your models using Prettytable library
df = pd.DataFrame({"Model":["Naive bayes with BOW","Naive bayes with TF
IDF"],
                   "Hyper parameter(K)":[0.06601,0.02901],
                   "AUC":[0.9286461382996808,0.9305983255796383]}
                  ,columns=["Model","Hyper parameter(K)","AUC"])
df.sort_values(by="AUC",ascending=False)
```

Out[30]:

	Model	Hyper parameter(K)	AUC
1	Naive bayes with TFIDF	0.02901	0.930598
0	Naive bayes with BOW	0.06601	0.928646

From above execution we can say that TFIDF model performs batter than the Bag Of Word model.