Amazon Fine Food Reviews Analysis

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

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: 74,258
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

[1]. Reading Data

[1.1] Loading the data

The dataset is available in two forms

1. .csv file
2. SQLite Database

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

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

In [1]:
```
%matplotlib inline
import warnings
warnings.filterwarnings("ignore")


import sqlite3
import pandas as pd
import numpy as np
import nltk
import string
import matplotlib.pyplot as plt
import seaborn as sns
from sklearn.model_selection import train_test_split
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
import prettytable
```

```
C:\Users\Hardik\Anaconda3\lib\site-packages\gensim\utils.py:1212: UserWarning: detected Windows; aliasing chu
nkize to chunkize_serial
  warnings.warn("detected Windows; aliasing chunkize to chunkize_serial")
```

In [0]:
```
#USER DEFINED FUNCTIONS

#changing reviews with score less than 3 to be positive and vice-versa
def data_filter(filtered_data):
    actualScore = filtered_data['Score']
    positiveNegative = actualScore.map(partition)
    filtered_data['Score'] = positiveNegative
    return filtered_data["Score"]


# 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
```

In [0]:
```
# Code to read csv file into colaboratory:
!pip install -U -q PyDrive
from pydrive.auth import GoogleAuth
from pydrive.drive import GoogleDrive
from google.colab import auth
from oauth2client.client import GoogleCredentials

# 1. Authenticate and create the PyDrive client.
auth.authenticate_user()
gauth = GoogleAuth()
gauth.credentials = GoogleCredentials.get_application_default()
drive = GoogleDrive(gauth)

#2. Get the file #Training Variants
downloaded = drive.CreateFile({'id':'1PPCURbwWREuTo7ZgIt9MZv26tpsAp9Bs'}) # replace the id with id of file yo
u want to access
downloaded.GetContentFile('Reviews.csv')

downloaded1 = drive.CreateFile({'id':'1PPC'}) # replace the id with id of file you want to access
downloaded.GetContentFile('Reviews.csv')

downloaded = drive.CreateFile({'id':'1PPCURbwWREuTo7ZgIt9MZv26tpsAp9Bs'}) # replace the id with id of file yo
u want to access
downloaded.GetContentFile('Reviews.csv')

downloaded = drive.CreateFile({'id':'1PPCURbwWREuTo7ZgIt9MZv26tpsAp9Bs'}) # replace the id with id of file yo
u want to access
downloaded.GetContentFile('Reviews.csv')

#3. Read file as panda dataframe
import pandas as pd
data = pd.read_csv('Reviews.csv')
```

In [0]:
```
data.head(20)
```

Out[0]:

	Id	ProductId	UserId	ProfileName	HelpfulnessNumerator	HelpfulnessDenominator	Score	Time	Summ
0	1	B001E4KFG0	A3SGXH7AUHU8GW	delmartian	1	1	5	1303862400	G Qua Dog F
1	2	B00813GRG4	A1D87F6ZCVE5NK	dll pa	0	0	1	1346976000	No Adverti
2	3	B000LQOCH0	ABXLMWJIXXAIN	Natalia Corres "Natalia Corres"	1	1	4	1219017600	"Deli says i
3	4	B000UA0QIQ	A395BORC6FGVXV	Karl	3	3	2	1307923200	Co Medic
4	5	B006K2ZZ7K	A1UQRSCLF8GW1T	Michael D. Bigham "M. Wassir"	0	0	5	1350777600	Great t
5	6	B006K2ZZ7K	ADT0SRK1MGOEU	Twoapennything	0	0	4	1342051200	Nice T
6	7	B006K2ZZ7K	A1SP2KVKFXXRU1	David C. Sullivan	0	0	5	1340150400	Great! as good expens brar
7	8	B006K2ZZ7K	A3JRGQVEQN31IQ	Pamela G. Williams	0	0	5	1336003200	Wonde tasty t

	Id	ProductId	UserId	ProfileName	HelpfulnessNumerator	HelpfulnessDenominator	Score	Time	Summ
8	9	B000E7L2R4	A1MZYO9TZK0BBI	R. James	1	1	5	1322006400	Yay Ba
9	10	B00171APVA	A21BT40VZCCYT4	Carol A. Reed	0	0	5	1351209600	Hea Dog F
10	11	B0001PB9FE	A3HDKO7OW0QNK4	Canadian Fan	1	1	5	1107820800	The E Hot Sa in W
11	12	B0009XLVG0	A2725IB4YY9JEB	A Poeng "SparkyGoHome"	4	4	5	1282867200	My LOVE "diet" f better t th
12	13	B0009XLVG0	A327PCT23YH90	LT	1	1	1	1339545600	My C Are Fan the N F
13	14	B001GVISJM	A18ECVX2RJ7HUE	willie "roadie"	2	2	4	1288915200	fresh grea
14	15	B001GVISJM	A2MUGFV2TDQ47K	Lynrie "Oh HELL no"	4	5	5	1268352000	Strawbe Twizzle Yum

	Id	ProductId	UserId	ProfileName	HelpfulnessNumerator	HelpfulnessDenominator	Score	Time	Summ
15	16	B001GVISJM	A1CZX3CP8IKQIJ	Brian A. Lee	4	5	5	1262044800	Lot twizzl just w exp
16	17	B001GVISJM	A3KLWF6WQ5BNYO	Erica Neathery	0	0	2	1348099200	poor ta
17	18	B001GVISJM	AFKW14U97Z6QO	Becca	0	0	5	1345075200	Lov
18	19	B001GVISJM	A2A9X58G2GTBLP	Wolfee1	0	0	5	1324598400	GRE SWE CANI
19	20	B001GVISJM	A3IV7CL2C13K2U	Greg	0	0	5	1318032000	Hc delive twiz

In [0]:

```
#Converting from timestamp to datetime object.

from datetime import datetime
Time = []
for i in list(data["Time"]):
  Time.append(datetime.fromtimestamp(i).strftime('%d-%m-%Y'))

data["Time"] = pd.to_datetime(Time)
data = data.sort_values("Time")
```

In [0]:
```
data.head(20)
```

Out[0]:

	Id	ProductId	UserId	ProfileName	HelpfulnessNumerator	HelpfulnessDenominator	Score	Time	
451855	451856	B00004CXX9	AIUWLEQ1ADEG5	Elizabeth Medina	0	0	5	1999-02-12	
230284	230285	B00004RYGX	A344SMIA5JECGM	Vincent P. Ross	1	2	5	1999-06-12	
451877	451878	B00004CXX9	A344SMIA5JECGM	Vincent P. Ross	1	2	5	1999-06-12	
374358	374359	B00004CI84	A344SMIA5JECGM	Vincent P. Ross	1	2	5	1999-06-12	
150523	150524	0006641040	ACITT7DI6IDDL	shari zychinski	0	0	5	1999-08-10	
150500	150501	0006641040	AJ46FKXOVC7NR	Nicholas A Mesiano	2	2	5	1999-10-25	This who
230268	230269	B00004RYGX	A1B2IZU1JLZA6	Wes	19	23	1	2000-01-19	WARNI
374342	374343	B00004CI84	A1B2IZU1JLZA6	Wes	19	23	1	2000-01-19	WARNI
451863	451864	B00004CXX9	A1B2IZU1JLZA6	Wes	19	23	1	2000-01-19	WARNI

	Id	ProductId	UserId	ProfileName	HelpfulnessNumerator	HelpfulnessDenominator	Score	Time	
76881	76882	B00002N8SM	A32DW342WBJ6BX	Buttersugar	0	0	5	2000-01-24	
451976	451977	B00004CXX9	ACJR7EQF9S6FP	Jeremy Robertson	2	3	4	2000-02-26	Bettlejuice
374449	374450	B00004CI84	ACJR7EQF9S6FP	Jeremy Robertson	2	3	4	2000-02-26	Bettlejuice
230375	230376	B00004RYGX	ACJR7EQF9S6FP	Jeremy Robertson	2	3	4	2000-02-26	Bettlejuice
451854	451855	B00004CXX9	AJH6LUC1UT1ON	The Phantom of the Opera	0	0	5	2000-03-01	
230325	230326	B00004RYGX	A2DEE7F9XKP3ZR	jerome	0	3	5	2000-03-06	Resear
451902	451903	B00004CXX9	A2DEE7F9XKP3ZR	jerome	0	1	5	2000-03-06	
374399	374400	B00004CI84	A2DEE7F9XKP3ZR	jerome	0	3	5	2000-03-06	Resear

	Id	ProductId	UserId	ProfileName	HelpfulnessNumerator	HelpfulnessDenominator	Score	Time	
230333	230334	B00004RYGX	A1GB1Q193DNFGR	Bruce Lee Pullen	5	5	5	2000-03-10	Fabulous
374407	374408	B00004CI84	A1GB1Q193DNFGR	Bruce Lee Pullen	5	5	5	2000-03-10	Fabulous
451934	451935	B00004CXX9	A1GB1Q193DNFGR	Bruce Lee Pullen	5	5	5	2000-03-10	Fabulous

In [0]:
```
# 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 = data

#Reading equal amount of positive and negative data.
#filtered_data_1 = data[data["Score"]>3][0:10000]
#filtered_data_2 = data[data["Score"]<=2][0:10000]

#filtered_data_1["Score"] = data_filter(filtered_data_1)
#filtered_data_2["Score"] = data_filter(filtered_data_2)

'''
# 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

def data_filter(filtered_data):
    actualScore = filtered_data['Score']
    positiveNegative = actualScore.map(partition)
    filtered_data['Score'] = positiveNegative
    return filtered_data["Score"]


#print("Number of data points in our data", filtered_data.shape)
#filtered_data.head(3)
'''
```

Out[0]:
```
'\n# Give reviews with Score>3 a positive rating(1), and reviews with a score<3 a negative rating(0).\ndef pa
rtition(x):\n    if x < 3:\n        return 0\n    return 1\n\n#changing reviews with score less than 3 to be
positive and vice-versa\n\ndef data_filter(filtered_data):\n    actualScore = filtered_data[\'Score\']\n    p
ositiveNegative = actualScore.map(partition) \n    filtered_data[\'Score\'] = positiveNegative\n    return fi
ltered_data["Score"]\n\n\n#print("Number of data points in our data", filtered_data.shape)\n#filtered_data.he
ad(3)\n'
```

In [0]:
```

```

In [0]:
```

```

In [0]:
```
'''

#Stacking both positive and negative data
filtered_data = filtered_data_1.append(filtered_data_2, ignore_index = True)

#Shuffling the data points to mix the data
from sklearn.utils import shuffle
filtered_data = shuffle(filtered_data)

'''
```

Out[0]:
```
'\n\n#Stacking both positive and negative data\nfiltered_data = filtered_data_1.append(filtered_data_2, ignor
e_index = True)\n\n#Shuffling the data points to mix the data\nfrom sklearn.utils import shuffle\nfiltered_da
ta = shuffle(filtered_data)\n\n'
```

In [0]:
```
#Calling function partition
filtered_data["Score"] = filtered_data["Score"].map(partition)
```

In [0]:
```
print("Number of data points in our data", filtered_data.shape)
filtered_data.head(3)
```

```
Number of data points in our data (568454, 10)
```

Out[0]:

	Id	ProductId	UserId	ProfileName	HelpfulnessNumerator	HelpfulnessDenominator	Score	Time	Summai
451855	451856	B00004CXX9	AIUWLEQ1ADEG5	Elizabeth Medina	0	0	1	1999-02-12	Entertainin Funn
230284	230285	B00004RYGX	A344SMIA5JECGM	Vincent P. Ross	1	2	1	1999-06-12	A modei day fai ta
451877	451878	B00004CXX9	A344SMIA5JECGM	Vincent P. Ross	1	2	1	1999-06-12	A modei day fai ta

In [0]:
```
filtered_data.reset_index(drop = True)
```

Out[0]:

	Id	ProductId	UserId	ProfileName	HelpfulnessNumerator	HelpfulnessDenominator	Score	Time	
0	451856	B00004CXX9	AIUWLEQ1ADEG5	Elizabeth Medina	0	0	1	1999-02-12	
1	230285	B00004RYGX	A344SMIA5JECGM	Vincent P. Ross	1	2	1	1999-06-12	
2	451878	B00004CXX9	A344SMIA5JECGM	Vincent P. Ross	1	2	1	1999-06-12	
3	374359	B00004CI84	A344SMIA5JECGM	Vincent P. Ross	1	2	1	1999-06-12	
4	150524	0006641040	ACITT7DI6IDDL	shari zychinski	0	0	1	1999-08-10	
5	150501	0006641040	AJ46FKXOVC7NR	Nicholas A Mesiano	2	2	1	1999-10-25	This wh
6	230269	B00004RYGX	A1B2IZU1JLZA6	Wes	19	23	0	2000-01-19	WARN
7	374343	B00004CI84	A1B2IZU1JLZA6	Wes	19	23	0	2000-01-19	WARN
8	451864	B00004CXX9	A1B2IZU1JLZA6	Wes	19	23	0	2000-01-19	WARN

	Id	ProductId	UserId	ProfileName	HelpfulnessNumerator	HelpfulnessDenominator	Score	Time	
9	76882	B00002N8SM	A32DW342WBJ6BX	Buttersugar	0	0	1	2000-01-24	
10	451977	B00004CXX9	ACJR7EQF9S6FP	Jeremy Robertson	2	3	1	2000-02-26	Bettlejui
11	374450	B00004CI84	ACJR7EQF9S6FP	Jeremy Robertson	2	3	1	2000-02-26	Bettlejui
12	230376	B00004RYGX	ACJR7EQF9S6FP	Jeremy Robertson	2	3	1	2000-02-26	Bettlejui
13	451855	B00004CXX9	AJH6LUC1UT1ON	The Phantom of the Opera	0	0	1	2000-03-01	
14	230326	B00004RYGX	A2DEE7F9XKP3ZR	jerome	0	3	1	2000-03-06	Resea
15	451903	B00004CXX9	A2DEE7F9XKP3ZR	jerome	0	1	1	2000-03-06	
16	374400	B00004CI84	A2DEE7F9XKP3ZR	jerome	0	3	1	2000-03-06	Resea

	Id	ProductId	UserId	ProfileName	HelpfulnessNumerator	HelpfulnessDenominator	Score	Time	
17	230334	B00004RYGX	A1GB1Q193DNFGR	Bruce Lee Pullen	5	5	1	2000-03-10	Fabulo
18	374408	B00004CI84	A1GB1Q193DNFGR	Bruce Lee Pullen	5	5	1	2000-03-10	Fabulo
19	451935	B00004CXX9	A1GB1Q193DNFGR	Bruce Lee Pullen	5	5	1	2000-03-10	Fabulo
20	149768	B00004S1C5	A7P76IGRZZBFJ	E. Thompson "Soooooper Genius"	18	18	1	2000-05-12	
21	1245	B00002Z754	A29Z5PI9BW2PU3	Robbie	7	7	1	2000-06-23	
22	1244	B00002Z754	A3B8RCEI0FXFI6	B G Chase	10	10	1	2000-06-29	
23	131217	B00004RAMX	A5NQLNC6QPGSI	Kim Nason	7	8	1	2000-07-31	
24	451948	B00004CXX9	A1FJOY14X3MUHE	Justin Howard	2	2	1	2000-08-15	
25	374421	B00004CI84	A1FJOY14X3MUHE	Justin Howard	2	2	1	2000-08-15	

	Id	ProductId	UserId	ProfileName	HelpfulnessNumerator	HelpfulnessDenominator	Score	Time
26	230347	B00004RYGX	A1FJOY14X3MUHE	Justin Howard	2	2	1	2000-08-15
27	374422	B00004CI84	A1048CYU0OV4O8	Judy L. Eans	2	2	1	2000-09-01
28	230348	B00004RYGX	A1048CYU0OV4O8	Judy L. Eans	2	2	1	2000-09-01
29	451949	B00004CXX9	A1048CYU0OV4O8	Judy L. Eans	2	2	1	2000-09-01
...	...	...	...	...	...	...	...	...
568424	17883	B001EO653M	A3IGARBJ4SE9EQ	Arielle M.	0	0	1	2012-12-10
568425	200504	B007PXI6CO	A1007OFJTJRYII	Jan	0	0	1	2012-12-10
568426	275087	B000FF3V06	A3EEJG97L9YIAB	Linda	0	0	1	2012-12-10
568427	566827	B001PQTYN2	AR4GVRPKO4MBL	ShahinX1	0	0	1	2012-12-10

	Id	ProductId	UserId	ProfileName	HelpfulnessNumerator	HelpfulnessDenominator	Score	Time	
568428	201918	B009062AQW	ARUJ8B1HTISK0	Blondie	0	0	1	2012-12-10	
568429	201995	B000LQL9M6	AI2R6R7UP4NYB	Sparky	0	0	1	2012-12-10	
568430	549437	B004TGZRJA	A2GDTF5XZFVW2G	Kirk Henry "old school"	0	0	1	2012-12-10	
568431	513699	B000VK33C6	A3LRLNH9WJDQY6	Diane Milan	0	0	1	2012-12-10	
568432	203834	B000VK4CTO	A1V5J0VTEL8DS8	Peter	0	0	0	2012-12-10	
568433	377295	B004VLWASE	A36DVRTEHDJKNP	Steve	0	0	1	2012-12-10	
568434	330098	B001OHX1ZY	A88XJQH33JG01	maryann	0	0	1	2012-12-10	b
568435	377319	B001IAQ8KC	AQO4BNFU7T4EH	Andrew P Freese	0	0	1	2012-12-10	
568436	203701	B004AW1Z94	A1D6FDBK9FJI8C	Jason Mark	0	0	0	2012-12-10	

	Id	ProductId	UserId	ProfileName	HelpfulnessNumerator	HelpfulnessDenominator	Score	Time	
568437	432397	B003A199AI	A2VI7ZC4CWDUPW	Vanessa Close	2	2	1	2012-12-10	
568438	432396	B003A199AI	A3F0WUGQIS488X	Daniel L	2	2	1	2012-12-10	Perfe
568439	80520	B008ADQYYU	A2DKVL26ZX0WGS	Sil	0	0	1	2012-12-10	
568440	50682	B007PQTIRI	A2XWFSMXJ1RR0R	Mel M.	0	0	1	2012-12-10	
568441	398671	B001D09KAM	AUS545VE0P2J1	Paul	0	0	1	2012-12-10	
568442	80622	B000WFKHN8	A29FD8FJONPAJ	Baxter "Uru"	0	0	1	2012-12-10	
568443	254881	B007PA32L2	A2ZOKXJOQULWXS	marie johnston	0	0	1	2012-12-10	
568444	398672	B001D09KAM	A2TKFP9YMPASAS	sammy3856	0	0	1	2012-12-10	
568445	254880	B007PA32L2	A3JK142YXC2RGK	judaicagirl "Dalia L."	0	0	0	2012-12-10	Tas

	Id	ProductId	UserId	ProfileName	HelpfulnessNumerator	HelpfulnessDenominator	Score	Time	
568446	254879	B007PA32L2	A2EJ1GFUFYUEBM	michael crow	0	0	0	2012-12-10	
568447	377320	B001IAQ8KC	A1JXRHA7PD04ZF	John	0	0	1	2012-12-10	
568448	274819	B005VOOOM0	AZS3RRB62EJVP	Dg7023	0	0	1	2012-12-10	
568449	305646	B001ELL4E0	AZ3EHLAKPMUU6	bigtiger	0	0	1	2012-12-10	
568450	202269	B000P0QHOI	AN4FZJKFHMAS0	Dawn E. Curran	0	0	1	2012-12-10	
568451	106712	B001HTG6E2	A29Y8V009MI4G5	Cate	0	0	1	2012-12-10	
568452	465911	B005OU6UD2	A2550XGZEFDH2Y	Melanie G. Nihart "Grammy"	0	0	0	2012-12-10	
568453	424141	B000EMK53G	A2B7BUH8834Y6M	Shelley Gammon "Geek"	0	0	1	2012-12-10	Shocl

568454 rows × 10 columns

In [0]:

In [0]:
```
'''display = pd.read_sql_query("""
SELECT UserId, ProductId, ProfileName, Time, Score, Text, COUNT(*)
FROM Reviews
GROUP BY UserId
HAVING COUNT(*)>1
""", con)'''
```

Out[0]:
```
'display = pd.read_sql_query("""\nSELECT UserId, ProductId, ProfileName, Time, Score, Text, COUNT(*)\nFROM Re
views\nGROUP BY UserId\nHAVING COUNT(*)>1\n""", con)'
```

In [0]:
```
'''print(display.shape)
display.head()'''
```

Out[0]:
```
'print(display.shape)\ndisplay.head()'
```

In [0]:
```
#display[display['UserId']=='AZY10LLTJ71NX']
```

In [0]:
```
#display['COUNT(*)'].sum()
```

[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]:
```
'display= pd.read_sql_query("""\nSELECT *\nFROM Reviews\nWHERE Score != 3 AND UserId="AR5J8UI46CURR"\nORDER B
Y ProductID\n""", con)\ndisplay.head()'
```

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

In [0]:
```
#Deduplication of entries
final=sorted_data.drop_duplicates(subset={"UserId","ProfileName","Time","Text"}, keep='first', inplace=False)
final.shape
```

Out[0]:
```
(393933, 10)
```

In [0]:
```
#Checking to see how much % of data still remains
(final['Id'].size*1.0)/(filtered_data['Id'].size*1.0)*100
```

Out[0]:
```
69.29901100176971
```

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]:
```
'display= pd.read_sql_query("""\nSELECT *\nFROM Reviews\nWHERE Score != 3 AND Id=44737 OR Id=64422\nORDER BY ProductID\n""", con)\n\ndisplay.head()'
```

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()
```

```
(393931, 10)
```

Out[0]:
```
1    336824
0     57107
Name: Score, dtype: int64
```

In [0]:
```
final.shape
```

Out[0]:
```
(393931, 10)
```

[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[4300]
print(sent_4900)
print("="*50)
```

```
It's a great book with adorable illustrations.  A true classic.  Kids love the poem and there is music that g
oes with it, if you can find it.  I think it's sung by Carol King.
==================================================
Well if you have had this product before you know it is amazing.  I am not going to describe the taste but i
will tell you the product was shipped neatly and fresh.  Everything tasted great and the expiration date was
much further into the future than this would have ever lasted.  Top notch.
==================================================
Cat thought the bubbles were interesting, but didn't go crazy over them. They smell funny, leave a residue, a
nd don't maintain form when they touch down as advertised. I'll go pick up a big bottle of regular bubbles fo
r him to chase. Not worth the price.
==================================================
The previous reviewer's experience is lamentable but after reading the reviews for other Brussel's Bonsai & a
fter reception of my own tree it's clear that his experience was the exception, not the rule.<br /><br />The
tree arrived well-packaged, lush & green. The soil was moist & wrapped in plastic to ensure it remained that
way. The tree is planted in a dark blue glazed ceramic pot. Included was a small pamphlet outlining the basic
s of Indoor Bonsai care. Overall, quite nice & well done. I wouldn't hesitate to order another tree from Brus
sel's. See my images above.<br /><br />UPDATE 08/11/2011:<br /><br />I ordered a second ficus from Brussel's
& it arrived in the same excellent condition as before & attractively trained. Very, very pleased.
==================================================
```

In [0]:

```
# 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)
```

```
It's a great book with adorable illustrations.  A true classic.  Kids love the poem and there is music that g
oes with it, if you can find it.  I think it's sung by Carol King.
```

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)
```

```
It's a great book with adorable illustrations.  A true classic.  Kids love the poem and there is music that g
oes with it, if you can find it.  I think it's sung by Carol King.
==================================================
Well if you have had this product before you know it is amazing.  I am not going to describe the taste but i
will tell you the product was shipped neatly and fresh.  Everything tasted great and the expiration date was
much further into the future than this would have ever lasted.  Top notch.
==================================================
Cat thought the bubbles were interesting, but didn't go crazy over them. They smell funny, leave a residue, a
nd don't maintain form when they touch down as advertised. I'll go pick up a big bottle of regular bubbles fo
r him to chase. Not worth the price.
==================================================
The previous reviewer's experience is lamentable but after reading the reviews for other Brussel's Bonsai & a
fter reception of my own tree it's clear that his experience was the exception, not the rule.The tree arrived
well-packaged, lush & green. The soil was moist & wrapped in plastic to ensure it remained that way. The tree
is planted in a dark blue glazed ceramic pot. Included was a small pamphlet outlining the basics of Indoor Bo
nsai care. Overall, quite nice & well done. I wouldn't hesitate to order another tree from Brussel's. See my
images above.UPDATE 08/11/2011:I ordered a second ficus from Brussel's & it arrived in the same excellent con
dition as before & attractively trained. Very, very pleased.
```

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)
```

```
Cat thought the bubbles were interesting, but did not go crazy over them. They smell funny, leave a residue,
and do not maintain form when they touch down as advertised. I will go pick up a big bottle of regular bubble
s for him to chase. Not worth the price.
==================================================
```

In [0]:
```
#remove words with numbers python: https://stackoverflow.com/a/18082370/4084039
sent_0 = re.sub("\S*\d\S*", "", sent_0).strip()
print(sent_0)
```

```
It's a great book with adorable illustrations.  A true classic.  Kids love the poem and there is music that g
oes with it, if you can find it.  I think it's sung by Carol King.
```

In [0]:
```
#remove spacial character: https://stackoverflow.com/a/5843547/4084039
sent_1500 = re.sub('[^A-Za-z0-9]+', ' ', sent_1500)
print(sent_1500)
```

```
Cat thought the bubbles were interesting but did not go crazy over them They smell funny leave a residue and
do not maintain form when they touch down as advertised I will go pick up a big bottle of regular bubbles for
him to chase Not worth the price
```

In [0]:
```
# 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 revmoved in the 1st step

stopwords= set(['br', 'the', 'i', 'me', 'my', 'myself', 'we', 'our', 'ours', 'ourselves', 'you', "you're", "y
ou'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', 'thos
e', \
            'am', 'is', 'are', 'was', 'were', 'be', 'been', 'being', 'have', 'has', 'had', 'having', 'do', 'd
oes', \
            'did', 'doing', 'a', 'an', 'the', 'and', 'but', 'if', 'or', 'because', 'as', 'until', 'while', 'o
f', \
            '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', 'fe
w', '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", 'must
n',\
            "mustn't", 'needn', "needn't", 'shan', "shan't", 'shouldn', "shouldn't", 'wasn', "wasn't", 'were
n', "weren't", \
            'won', "won't", 'wouldn', "wouldn't"])
```

In [0]:
```
# 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%|██████████| 393931/393931 [02:58<00:00, 2206.16it/s]
```

In [0]:
```
#Used when 50% of positive and 50% of negative reviews are needed.
'''
final["New_Text"] = preprocessed_reviews

a = final[final["Score"]==0][0:500]
b = final[final["Score"]==1][0:500]

#Stacking both positive and negative data
a = a.append(b, ignore_index = True)

#Shuffling the data points to mix the data
from sklearn.utils import shuffle
a = shuffle(a)'''
```

Out[0]:
```
'\nfinal["New_Text"] = preprocessed_reviews\n\na = final[final["Score"]==0][0:500]\nb = final[final["Score"]=
=1][0:500]\n\n#Stacking both positive and negative data\na = a.append(b, ignore_index = True)\n\n#Shuffling t
he data points to mix the data\nfrom sklearn.utils import shuffle\na = shuffle(a)'
```

In [0]:
```

```

In [0]:
```
#Train-Test Split

X_train,X_test,y_train, y_test = train_test_split(list(preprocessed_reviews[0:20000]), list(final["Score"][0:
20000]), random_state = 42, test_size = 0.3, stratify = list(final["Score"][0:20000]))
```

In [0]:

```

```

[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(max_features = 200) #in scikit-learn
count_vect.fit(X_train)
print("some feature names ", count_vect.get_feature_names()[:10])
print('='*50)

final_counts = count_vect.transform(X_train)
final_counts_test = count_vect.transform(X_test)
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])
print("="*50)
print("for test data")
print("the type of count vectorizer ",type(final_counts_test))
print("the shape of out text BOW vectorizer ",final_counts_test.get_shape())
print("the number of unique words ", final_counts_test.get_shape()[1])
```

```
some feature names  ['actually', 'add', 'almost', 'also', 'always', 'amazon', 'amount', 'another', 'anythin
g', 'around']
==================================================
the type of count vectorizer  <class 'scipy.sparse.csr.csr_matrix'>
the shape of out text BOW vectorizer  (14000, 200)
the number of unique words  200
==================================================
for test data
the type of count vectorizer  <class 'scipy.sparse.csr.csr_matrix'>
the shape of out text BOW vectorizer  (6000, 200)
the number of unique words  200
```

[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-grams
# count_vect = CountVectorizer(ngram_range=(1,2))
# please do read the CountVectorizer documentation http://scikit-learn.org/stable/modules/generated/sklearn.f
eature_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(X_train)
final_bigram_counts_test = count_vect.transform(X_test)
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])

print("="*50)
print("for test data")

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

```
the type of count vectorizer  <class 'scipy.sparse.csr.csr_matrix'>
the shape of out text BOW vectorizer  (14000, 5000)
the number of unique words including both unigrams and bigrams  5000
==================================================
for test data
the type of count vectorizer  <class 'scipy.sparse.csr.csr_matrix'>
the shape of out text BOW vectorizer  (6000, 5000)
the number of unique words including both unigrams and bigrams  5000
```

[4.3] TF-IDF

In [0]:
```
tf_idf_vect = TfidfVectorizer(ngram_range=(1,2), min_df=10, max_features = 200)
tf_idf_vect.fit(X_train)
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(X_train)
final_tf_idf_test = tf_idf_vect.transform(X_test)

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])

print("="*50)
print("for test data")

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

```
some sample features(unique words in the corpus) ['actually', 'add', 'almost', 'also', 'always', 'amazon', 'a
mount', 'another', 'anything', 'around']
==================================================
the type of count vectorizer  <class 'scipy.sparse.csr.csr_matrix'>
the shape of out text TFIDF vectorizer  (14000, 200)
the number of unique words including both unigrams and bigrams  200
==================================================
for test data
the type of count vectorizer  <class 'scipy.sparse.csr.csr_matrix'>
the shape of out text TFIDF vectorizer  (6000, 200)
the number of unique words including both unigrams and bigrams  200
```

In [0]:

[4.4] Word2Vec

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

In [0]:
```
# Train your own Word2Vec model using your own text corpus for test data
i=0
list_of_sentance_test=[]
for sentance in X_test:
    list_of_sentance_test.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 values
# To use this code-snippet, download "GoogleNews-vectors-negative300.bin"
# from https://drive.google.com/file/d/0B7XkCwpI5KDYNlNUTTlSS21pQmM/edit
# it's 1.9GB in size.


# http://kavita-ganesan.com/gensim-word2vec-tutorial-starter-code/#.W17SRFAzZPY
# 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_train_w2v = True, to train your own w2v ")
```

```
[('good', 0.8626651763916016), ('wonderful', 0.7947291731834412), ('excellent', 0.7797785997390747), ('fantas
tic', 0.7426627278327942), ('amazing', 0.7361242175102234), ('awesome', 0.7321888208389282), ('perfect', 0.73
15688729286194), ('love', 0.7119004726409912), ('fine', 0.6943828463554382), ('delicious', 0.688669204711914
1)]
==================================================
[('unfiltered', 0.960004448890686), ('abroad', 0.9342438578605652), ('favorites', 0.9341275095939636), ('iv
e', 0.933422327041626), ('rank', 0.9327754974365234), ('stacks', 0.9294769763946533), ('coca', 0.928681612014
7705), ('heard', 0.9250746965408325), ('name', 0.9249507784843445), ('sweetest', 0.9236856698989868)]
```

In [0]:
```
#Creating a word vocabulary based on training data.
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  8052
sample words  ['every', 'time', 'pick', 'peanut', 'butter', 'goo', 'completely', 'loses', 'mind', 'sits', 'fl
oor', 'eats', 'instead', 'chewing', 'everything', 'else', 'house', 'happy', 'couple', 'bucks', 'seller', 'res
ponded', 'concerns', 'beyond', 'expectations', 'would', 'highly', 'recommend', 'buying', 'product', 'fresh',
'service', 'responsive', 'made', 'french', 'onion', 'soup', 'added', 'protein', 'tastes', 'great', 'especiall
y', 'glad', 'high', 'sent', 'gift', 'review', 'sister', 'good', 'family']
```

In [0]:
```

```

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

[4.4.1.1] Avg W2v

In [0]:
```
#Converting each word of a review from training data into vector of len 50, adding them up and the finding an
 average. Hence converting training data into vector form.

# 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%|██████████| 14000/14000 [00:27<00:00, 513.48it/s]

14000
50
```

In [0]:
```
#Resulting vector representation of Training data
X_tr_w2v = sent_vectors
```

In [0]:
```
#Converting each word of a review from test data into vector of len 50, adding them up and the finding an ave
rage. Hence converting test data into vector form.

# 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_test): # 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%|██████████| 6000/6000 [00:11<00:00, 506.01it/s]

6000
50
```

In [0]:
```
#Resulting vector representation of reviews of Test data
X_ts_w2v = sent_vectors
```

[4.4.1.2] TFIDF weighted W2v

In [0]:
```
#Training the TfidfVectorizer on Train data

# S = ["abc def pqr", "def def def abc", "pqr pqr def"]
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 [0]:
```
#Converting each word of a review from Train data into Tfidf W2v vector representation of len 50. Hence conve
rting Train data into vector form.

# 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%|██████████| 14000/14000 [03:42<00:00, 62.92it/s]
```

In [0]:
```
#Resulting vector representation of reviews of Train data
X_tr_tf = tfidf_sent_vectors
```

In [0]:
```
#Converting each word of a review from Test data into Tfidf W2v vector representation of len 50. Hence conver
ting Test data into vector form.
#Test data is converted to vectors using w2v_model and TfidfVectorizer which were built on Training data.

# 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_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/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%|██████████| 6000/6000 [01:32<00:00, 65.12it/s]
```

In [0]:
```
#Resulting vector representation of reviews of Test data
X_ts_tf = tfidf_sent_vectors
```

[5] Assignment 3: KNN

1. **Apply Knn(brute force version) 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. **Apply Knn(kd tree version) on these feature sets**
   NOTE: sklearn implementation of kd-tree accepts only dense matrices, you need to convert the sparse matrices of CountVectorizer/TfidfVectorizer into dense matices. You can convert sparse matrices to dense using .toarray() attribute. For more information please visit this [link (https://docs.scipy.org/doc/scipy-0.18.1/reference/generated/scipy.sparse.csr_matrix.toarray.html)](https://docs.scipy.org/doc/scipy-0.18.1/reference/generated/scipy.sparse.csr_matrix.toarray.html)

    - SET 5:Review text, preprocessed one converted into vectors using (BOW) but with restriction on maximum features generated.

```
count_vect = CountVectorizer(min_df=10, max_features=500)
count_vect.fit(preprocessed_reviews)
```

    - SET 6:Review text, preprocessed one converted into vectors using (TFIDF) but with restriction on maximum features generated.

```
tf_idf_vect = TfidfVectorizer(min_df=10, max_features=500)
tf_idf_vect.fit(preprocessed_reviews)
```

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

3. **The hyper paramter tuning(find best K)**

    - Find the best hyper parameter which will give the maximum [AUC (https://www.appliedaicourse.com/course/applied-ai-course-online/lessons/receiver-operating-characteristic-curve-roc-curve-and-auc-1/)](https://www.appliedaicourse.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

4. **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

- 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 (https://www.appliedaicourse.com/course/applied-ai-course-online/lessons/confusion-matrix-tpr-fpr-fnr-tnr-1/) with predicted and original labels of test data points

5. **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 (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/applied-ai-course/leakage-bow-and-tfidf)

In [0]:
```
# Common User defined function used.

#Plots AUC Score vs Neighbors
def plot_best_hyperparameter(gridcv):
    cv_result = pd.DataFrame(gridcv.cv_results_)      #gridcv.cv_results_ outputs dict object of cross valida
tion score and train score.
    '''cv = {
    "neighbors" : cv_result["param_n_neighbors"],
    "train score" : cv_result["mean_train_score"],
    "validation score" : cv_result["mean_test_score"]}'''

    #CV_score = pd.DataFrame(cv)

    #Plot for Train and Test data
    plt.figure()
    plt.title("Best hyperparameter for Train and Test data")
    line1 = plt.plot(cv_result["param_n_neighbors"], cv_result["mean_train_score"], label = "Train_AUC_Score"
)
    line2 = plt.plot(cv_result["param_n_neighbors"], cv_result["mean_test_score"], label = "Cross_Validation_
AUC_Score")
    plt.xticks(rotation=90)
    plt.xlabel("Neighbors/ K")
    plt.ylabel("AUC Score")
    plt.legend()
    plt.show()


    '''plt.figure()
    plt.title("Best hyperparameter for validation data")
    plt.plot(cv_result["param_n_neighbors"], cv_result["mean_test_score"])
    plt.xticks(rotation=90)
    plt.xlabel("Neighbors")
    plt.ylabel("AUC Score")
    plt.show()'''


def plot_roc_curve(test_y, predict_proba_y):     #Plots ROC Curve

    fpr, tpr, threshold = roc_curve(test_y, predict_proba_y)
    auc_area = metrics.auc(fpr, tpr)

    plt.figure()
```

```
    plt.plot(fpr, tpr, color = 'darkorange', linewidth = 2,label = "AUC: %0.2f" %auc_area)
    plt.plot([0,1],[0,1], linewidth = 2, linestyle="--")
    plt.xlim([0,1])
    plt.ylim([0,1])
    plt.xlabel("FPR")
    plt.ylabel("TPR")
    plt.title("ROC Curve")
    plt.legend(loc = "lower right")
    plt.show()



def plot_confusion_mat(test_y, predict_proba_y):    #Plots Confusion Matrix

    cnf_mat = confusion_matrix(test_y, predict_proba_y)
    cnf_df = pd.DataFrame(cnf_mat, index = ["Actual: 0", "Actual: 1"], columns = ["Predicted: 0", "Predicted:
 1"], dtype= float)

    plt.figure(figsize=(5,3))
    plt.title("Confusion Matrix")
    sns.heatmap(cnf_df, annot = True, fmt = "g")
```

In [0]:

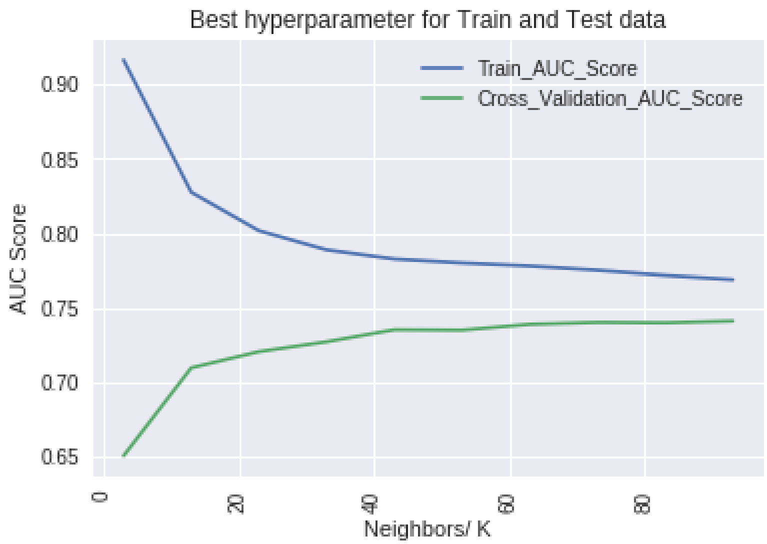
```

```

[5.2] Applying KNN kd-tree

[5.2.1] Applying KNN kd-tree on BOW, SET 5

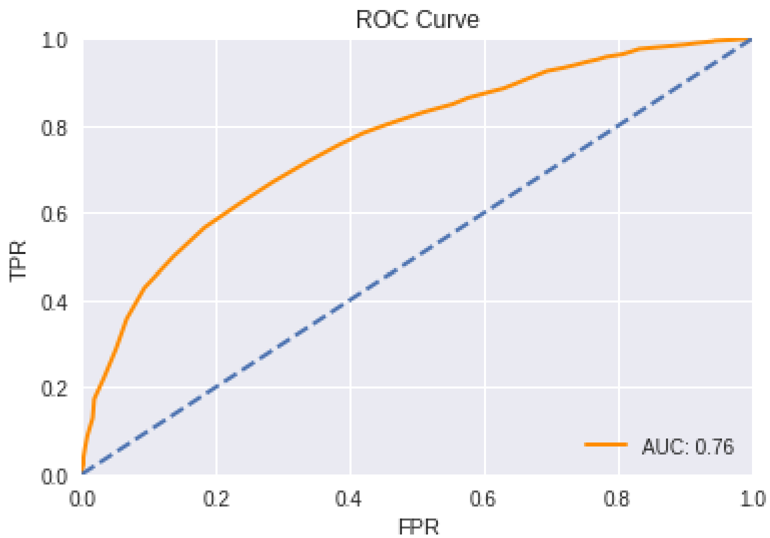
In [0]:
```
# Please write all the code with proper documentation
#final_counts = final_counts.toarray()

#count_vect = CountVectorizer(max_features = 200)
#count_vect.fit(X_train)

#final_counts = count_vect.transform(X_train)
#final_counts_test = count_vect.transform(X_test)

#Converting the sparse matrix to dense
final_counts = final_counts.toarray()
final_counts_test = final_counts_test.toarray()

from sklearn.model_selection import GridSearchCV
from sklearn.neighbors import KNeighborsClassifier

#Parameters

'''params = {"n_neighbors":[3,5,7,9,11,13,15,17,19,23,25,27,29,31,35,41,45,51,75,99]
         }'''

params = {"n_neighbors":[3,13,23,33,43,53,63,73,83,93]
         }

knn_bow = KNeighborsClassifier(algorithm = "kd_tree")

gridcv = GridSearchCV(knn_bow, params, scoring = 'roc_auc')
gridcv.fit(final_counts, y_train)
print(gridcv.best_params_)

bst_paramtr = gridcv.best_params_["n_neighbors"]

#Plots the graph for all  parameters to find best hyperparameter.
plot_best_hyperparameter(gridcv)
```

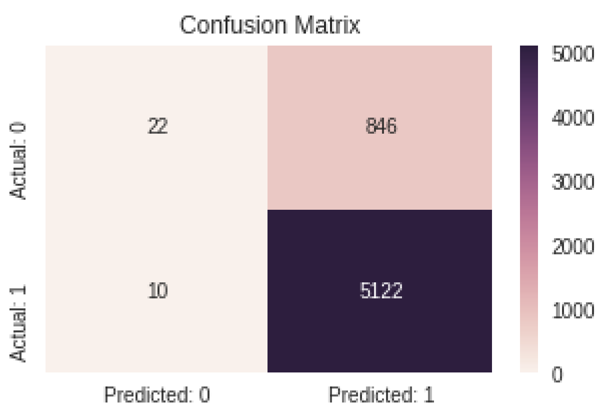
```
{'n_neighbors': 93}
```



In [0]:
```
from sklearn.metrics import roc_auc_score
#final_counts_test = final_counts_test.toarray()

knn_bow_1 = KNeighborsClassifier(n_neighbors = bst_paramtr , algorithm = "kd_tree")
knn_bow_1.fit(final_counts, y_train)

y_predict_proba_bow_kd = knn_bow_1.predict_proba(final_counts_test)[:,1]
y_predict_bow_kd = knn_bow_1.predict(final_counts_test)

roc_auc = roc_auc_score(y_test, y_predict_proba_bow_kd)

print("AUC for KNN with BOW for KD Tree method: ", roc_auc)

#Plots ROC curve
plot_roc_curve(y_test, y_predict_proba_bow_kd)

#Plots Confusion matrix
plot_confusion_mat(y_test, y_predict_bow_kd)
```

```
AUC for KNN with BOW for KD Tree method:  0.7619648873428133
```





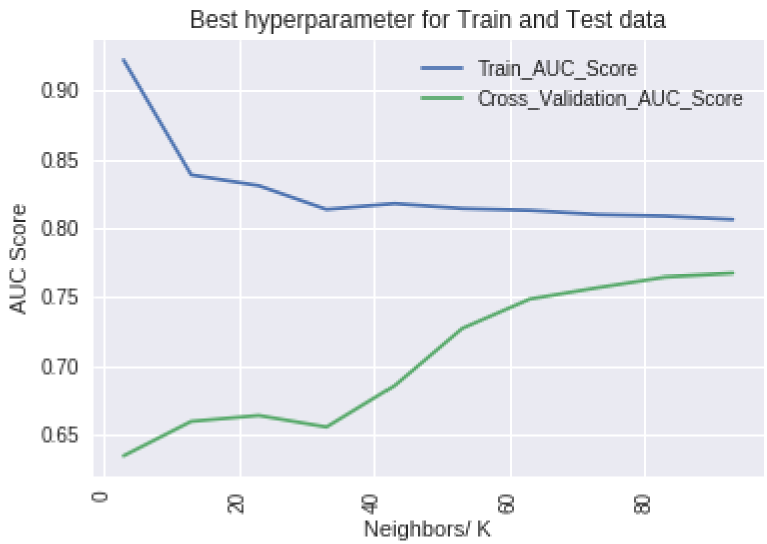
In [0]:

```

```

[5.2.2] Applying KNN kd-tree on TFIDF, SET 6

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

#tf_idf_vect = TfidfVectorizer(ngram_range=(1,2), min_df=10, max_features = 200)
#tf_idf_vect.fit(X_train)

#final_counts = tf_idf_vect.transform(X_train)
#final_counts_test = tf_idf_vect.transform(X_test)

#Converts sparse matix to dense
final_tf_idf = final_tf_idf.toarray()
final_tf_idf_test = final_tf_idf_test.toarray()

#Parameters
'''params = {"n_neighbors":[25,27,29,31,35,41,45,51,75,99]
         }'''

params = {"n_neighbors":[3,13,23,33,43,53,63,73,83,93]
         }

knn_bow = KNeighborsClassifier(algorithm = "kd_tree")

gridcv = GridSearchCV(knn_bow, params, scoring = 'roc_auc', cv = 5)
gridcv.fit(final_tf_idf, y_train)
print(gridcv.best_params_)

bst_paramtr = gridcv.best_params_["n_neighbors"]

plot_best_hyperparameter(gridcv)
```

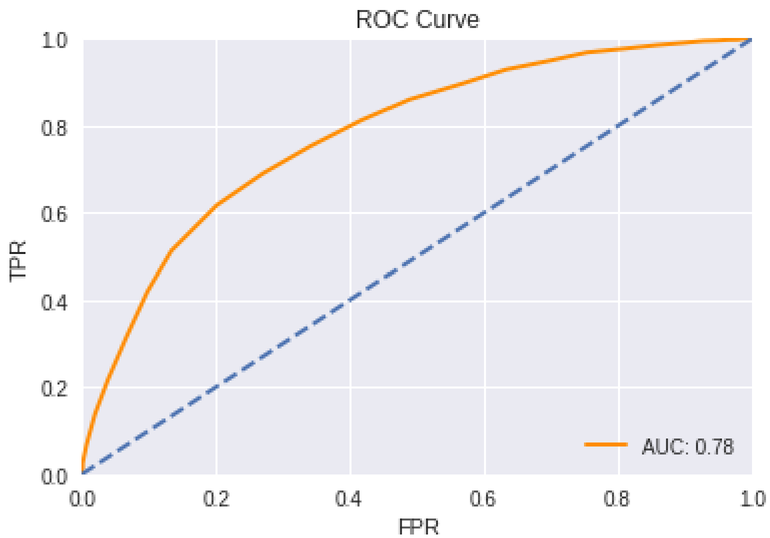
```
{'n_neighbors': 93}
```

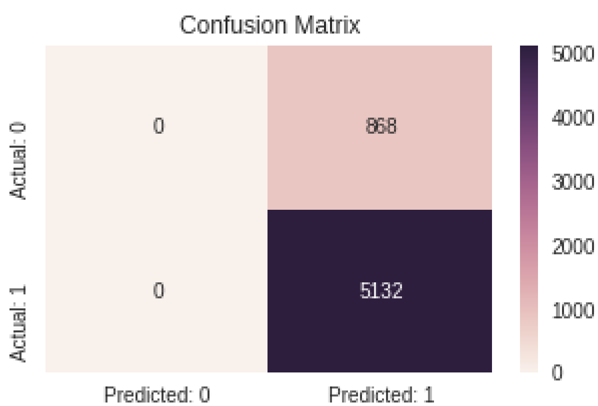


In [0]:

```
knn_bow_1 = KNeighborsClassifier(n_neighbors = bst_paramtr , algorithm = "kd_tree")
knn_bow_1.fit(final_tf_idf, y_train)

y_predict_proba_tfidf_kd = knn_bow_1.predict_proba(final_tf_idf_test)[:,1]
y_predict_tfidf_kd = knn_bow_1.predict(final_tf_idf_test)

roc_auc = roc_auc_score(y_test, y_predict_proba_tfidf_kd)

print("AUC for KNN with TFIDF for KD Tree method: ", roc_auc)

#Plots ROC curve
plot_roc_curve(y_test, y_predict_proba_tfidf_kd)

#Plots Confusion matrix
plot_confusion_mat(y_test, y_predict_tfidf_kd)
```

```
AUC for KNN with TFIDF for KD Tree method:  0.7808253804626971
```



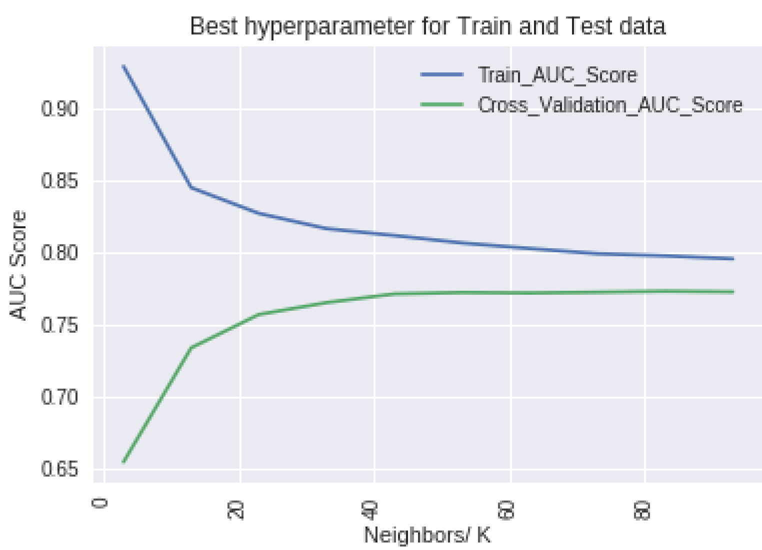


In [0]:

[5.2.3] Applying KNN kd-tree on AVG W2V, SET 3

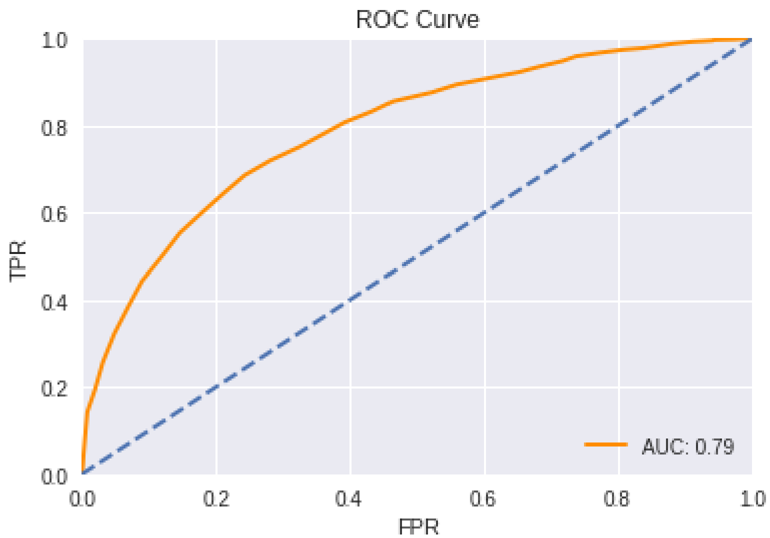
In [0]:
```
# Please write all the code with proper documentation

#Parameters
'''params = {"n_neighbors":[25,27,29,31,35,41,45,51,75,99]
         }'''
params = {"n_neighbors":[3,13,23,33,43,53,63,73,83,93]
         }

knn_bow = KNeighborsClassifier(algorithm = "kd_tree")

gridcv = GridSearchCV(knn_bow, params, scoring = 'roc_auc', cv = 5)
gridcv.fit(X_tr_w2v, y_train)
print(gridcv.best_params_)

bst_paramtr = gridcv.best_params_["n_neighbors"]

#Plots the graph for all  parameters to find best hyperparameter.
plot_best_hyperparameter(gridcv)
```

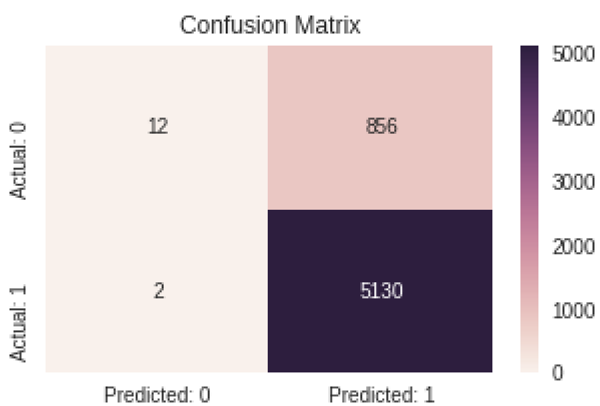
```
{'n_neighbors': 83}
```



In [0]:
```
from sklearn.model_selection import GridSearchCV
from sklearn.neighbors import KNeighborsClassifier
from sklearn.metrics import roc_auc_score

knn_bow_1 = KNeighborsClassifier(n_neighbors = bst_paramtr , algorithm = "kd_tree")
knn_bow_1.fit(X_tr_w2v, y_train)

y_predict_proba_w2v_kd = knn_bow_1.predict_proba(X_ts_w2v)[:,1]
y_predict_w2v_kd = knn_bow_1.predict(X_ts_w2v)

roc_auc = roc_auc_score(y_test, y_predict_proba_w2v_kd)

print("AUC for KNN with AVG W2V for KD Tree method: ", roc_auc)

#Plots ROC curve
plot_roc_curve(y_test, y_predict_proba_w2v_kd)

#Plots Confusion matrix
plot_confusion_mat(y_test, y_predict_w2v_kd)
```

```
AUC for KNN with AVG W2V for KD Tree method:  0.7915083276163657
```

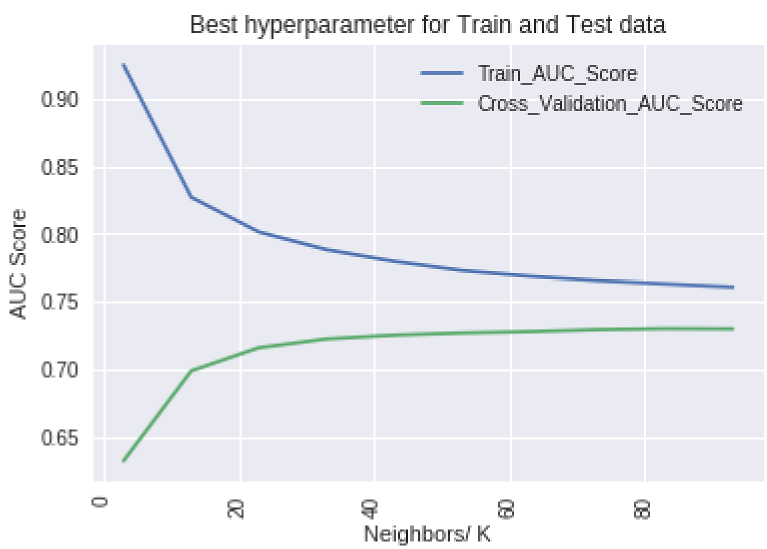




[5.2.4] Applying KNN kd-tree on TFIDF W2V, SET 4

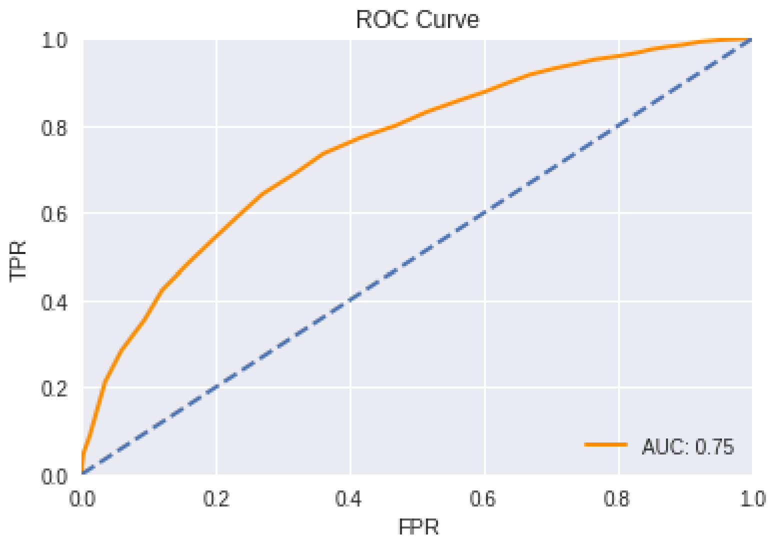
In [0]:
```
# Please write all the code with proper documentation

#Parameters
'''params = {"n_neighbors":[25,27,29,31,35,41,45,51,75,99]
         }'''
params = {"n_neighbors":[3,13,23,33,43,53,63,73,83,93]
         }
knn_bow = KNeighborsClassifier(algorithm = "kd_tree")

gridcv = GridSearchCV(knn_bow, params, scoring = 'roc_auc', cv = 5)
gridcv.fit(X_tr_tf, y_train)
print(gridcv.best_params_)

bst_paramtr = gridcv.best_params_["n_neighbors"]

#Plots the graph for all  parameters to find best hyperparameter.
plot_best_hyperparameter(gridcv)
```

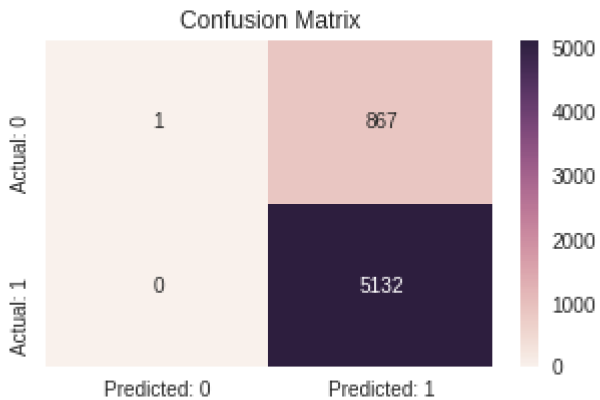
```
{'n_neighbors': 83}
```



In [0]:
```
knn_bow_1 = KNeighborsClassifier(n_neighbors = bst_paramtr , algorithm = "kd_tree")
knn_bow_1.fit(X_tr_tf, y_train)

y_predict_proba_tfidf_w2v_kd = knn_bow_1.predict_proba(X_ts_tf)[:,1]
y_predict_tfidf_w2v_kd = knn_bow_1.predict(X_ts_tf)

roc_auc = roc_auc_score(y_test, y_predict_proba_tfidf_w2v_kd)

print("AUC for KNN with TFIDF W2V for KD Tree method: ", roc_auc)

#Plots ROC curve
plot_roc_curve(y_test, y_predict_proba_tfidf_w2v_kd)

#Plots Confusion matrix
plot_confusion_mat(y_test, y_predict_tfidf_w2v_kd)
```

```
AUC for KNN with TFIDF W2V for KD Tree method:  0.7502317841249089
```





[6] Conclusions

In [2]:
```
# Please compare all your models using Prettytable library

print("\nNote: For this assignment I have used 20K datapoints for KD Tree algo and 50k points for Brute Force
 method as mentioned.\n")

print("\nTable: Summary of all the Vectorization techniques and Algorithms used.")

table = prettytable.PrettyTable()
table.field_names=["Algorithm", "Vectorization Technique","Best Test hyperparameter", "AUC Test Score"]

table.add_row(["Brute force","BOW",93,0.74126])
table.add_row(["Brute force","TFIDF",93,0.57777])
table.add_row(["Brute force","AVG W2V",83,0.84825])
table.add_row(["Brute force","TFIDF W2V",83,0.8155])
table.add_row(["KD Tree","BOW",93,0.76196])
table.add_row(["KD Tree","TFIDF",93,0.78082])
table.add_row(["KD Tree","AVG W2V",83,0.7915])
table.add_row(["KD Tree","TFIDF W2V",83,0.7502])

print(table)


print("\nLooking at the above table, we can conclude that for the given dataset, KNN model with AVG W2V vecto
rization and Brute force method seems to give the better AUC Score of 0.84825")
```

Note: For this assignment I have used 20K datapoints for KD Tree algo and 50k points for Brute Force method as mentioned.

Table: Summary of all the Vectorization techniques and Algorithms used.

Algorithm	Vectorization Technique	Best Test hyperparameter	AUC Test Score
Brute force	BOW	93	0.74126
Brute force	TFIDF	93	0.57777
Brute force	AVG W2V	83	0.84825
Brute force	TFIDF W2V	83	0.8155
KD Tree	BOW	93	0.76196
KD Tree	TFIDF	93	0.78082
KD Tree	AVG W2V	83	0.7915
KD Tree	TFIDF W2V	83	0.7502

Looking at the above table, we can conclude that for the given dataset, KNN model with AVG W2V vectorization and Brute force method seems to give the better AUC Score of 0.84825