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 polarity (positivity/negativity) of a review.

[1]. Reading Data

[1.1] Loading the data

The dataset is available in two forms

1. .csv file
2. SQLite Database

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

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

In [1]:

```
%matplotlib inline
import warnings
warnings.filterwarnings("ignore")


import sqlite3
import pandas as pd
import numpy as np
import nltk
import string
import matplotlib.pyplot as plt
import seaborn as sns
from sklearn.feature_extraction.text import TfidfTransformer
from sklearn.feature_extraction.text import TfidfVectorizer
from sklearn.model_selection import cross_val_score
from sklearn.neighbors import KNeighborsClassifier
from sklearn.metrics import accuracy_score

from sklearn.model_selection import train_test_split

from scipy.sparse import csr_matrix, issparse
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
from sklearn.metrics import classification_report
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 sklearn.neighbors import KNeighborsClassifier
from gensim.models import Word2Vec
from gensim.models import KeyedVectors
import pickle

from tqdm import tqdm
import os
```

In [2]:

```
# using SQLite Table to read data.
con = sqlite3.connect('../input/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 po
# 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 LIMI
# for tsne assignment you can take 5k data points

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

# Give reviews with Score>3 a positive rating(1), and reviews with a score<3 a nega
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 (30000, 10)
```

Out[2]:

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

In [3]:

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

In [4]:

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

(80668, 7)

Out[4]:

	UserId	ProductId	ProfileName	Time	Score	Text	COUNT(*)
0	#oc-R115TNMSPFT9I7	B005ZBZLT4	Breyton	1331510400	2	Overall its just OK when considering the price...	2
1	#oc-R11D9D7SHXIJB9	B005HG9ESG	Louis E. Emory "hoppy"	1342396800	5	My wife has recurring extreme muscle spasms, u...	3
2	#oc-R11DNU2NBKQ23Z	B005ZBZLT4	Kim Cieszykowski	1348531200	1	This coffee is horrible and unfortunately not ...	2
3	#oc-R11O5J5ZVQE25C	B005HG9ESG	Penguin Chick	1346889600	5	This will be the bottle that you grab from the...	3
4	#oc-R12KPBODL2B5ZD	B007OSBEV0	Christopher P. Presta	1348617600	1	I didnt like this coffee. Instead of telling y...	2

In [5]:

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

Out[5]:

	UserId	ProductId	ProfileName	Time	Score	Text	COUNT(*)
80638	AZY10LLTJ71NX	B001ATMQK2	undertheshrine "undertheshrine"	1296691200	5	I bought this 6 pack because for the price tha...	5

In [6]:

```
display['COUNT(*)'].sum()
```

Out[6]:

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 [7]:

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

Out[7]:

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

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

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

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

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

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

In [8]:

```
#Sorting data according to ProductId in ascending order
sorted_data=filtered_data.sort_values('ProductId', axis=0, ascending=True, inplace=
```

In [9]:

```
#Deduplication of entries
final=sorted_data.drop_duplicates(subset={"UserId","ProfileName","Time","Text"}, ke
final.shape
```

Out[9]:

```
(28072, 10)
```

In [10]:

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

Out[10]:

```
93.57333333333332
```

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 [11]:

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

display.head()
```

Out[11]:

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

In [12]:

```
final=final[final.HelpfulnessNumerator<=final.HelpfulnessDenominator]
```

In [13]:

```
#Before starting the next phase of preprocessing lets see the number of entries lef
print(final.shape)

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

```
(28072, 10)
```

Out[13]:

```
1    23606
0     4466
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 [14]:

```
# 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)
```

```
Our dogs just love them.  I saw them in a pet store and a tag was atta
ched regarding them being made in China and it satisfied me that they
were safe.
==================================================
When I ordered these, I thought they were a bit pricey, but I decided
to give them a try anyway.  I'm glad I did!  My dogs absolutely love t
hese dried liver treats.  And, since my dogs are all small, I can cut
the treats in half and still have large enough pieces to satisfy them.
They're great for training; I'll definitely order them again, and woul
d recommend them to anyone.
==================================================
This was my favorite stevia product and I had it on subscribe and save
until I queried customer service about NuNaturals GMO use.  Yes, NuNat
urals uses GMO products.  SO, I've canceled my subscribe and save orde
r and am now using <a href="http://www.amazon.com/gp/product/B001ELL3U
0">Stevita Stevia Clear Liquid Extract, 3.3-Ounce Container (Pack of
3)</a>.
==================================================
TOTALLY ORGASMIC.  these chips are the best spicy chip i have ever tas
ted.  signed up for the subscribe and save option.  the case contained
15(FIFTEEN, FULL SIZED BAGS) OF CHIPS.  the price per unit equals $1.7
3  per package.  that is not even the cost of plain chips.  if you add
the free shipping and the fast delivery, this deal is a steal.  so run
like you stole something over to your computer and order the SPICY THA
I CHIPS.  p.s.  even if you paid the going price of $30.00, you are st
ill ahead of the curve.  ENJOY
==================================================
```

In [15]:

```
# 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)
```

```
Our dogs just love them.  I saw them in a pet store and a tag was atta
ched regarding them being made in China and it satisfied me that they
were safe.
```

In [16]:

```
# https://stackoverflow.com/questions/16206380/python-beautifulsoup-how-to-remove-a
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)
```

```
Our dogs just love them.  I saw them in a pet store and a tag was atta
ched regarding them being made in China and it satisfied me that they
were safe.
==================================================
When I ordered these, I thought they were a bit pricey, but I decided
to give them a try anyway.  I'm glad I did!  My dogs absolutely love t
hese dried liver treats.  And, since my dogs are all small, I can cut
the treats in half and still have large enough pieces to satisfy them.
They're great for training; I'll definitely order them again, and woul
d recommend them to anyone.
==================================================
This was my favorite stevia product and I had it on subscribe and save
until I queried customer service about NuNaturals GMO use.  Yes, NuNat
urals uses GMO products.  SO, I've canceled my subscribe and save orde
r and am now using Stevita Stevia Clear Liquid Extract, 3.3-Ounce Cont
ainer (Pack of 3).
==================================================
TOTALLY ORGASMIC.  these chips are the best spicy chip i have ever tas
ted.  signed up for the subscribe and save option.  the case contained
15(FIFTEEN, FULL SIZED BAGS) OF CHIPS.  the price per unit equals $1.7
3  per package.  that is not even the cost of plain chips.  if you add
the free shipping and the fast delivery, this deal is a steal.  so run
like you stole something over to your computer and order the SPICY THA
I CHIPS.  p.s.  even if you paid the going price of $30.00, you are st
ill ahead of the curve.  ENJOY
```

In [17]:

```
# 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 [18]:

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

```
This was my favorite stevia product and I had it on subscribe and save
until I queried customer service about NuNaturals GMO use.  Yes, NuNat
urals uses GMO products.  SO, I have canceled my subscribe and save or
der and am now using <a href="http://www.amazon.com/gp/product/B001ELL
3U0">Stevita Stevia Clear Liquid Extract, 3.3-Ounce Container (Pack of
3)</a>.
==================================================
```

In [19]:

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

```
Our dogs just love them.  I saw them in a pet store and a tag was atta
ched regarding them being made in China and it satisfied me that they
were safe.
```

In [20]:

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

```
This was my favorite stevia product and I had it on subscribe and save
until I queried customer service about NuNaturals GMO use Yes NuNatura
ls uses GMO products SO I have canceled my subscribe and save order an
d am now using a href http www amazon com gp product B001ELL3U0 Stevit
a Stevia Clear Liquid Extract 3 3 Ounce Container Pack of 3 a
```

In [21]:

```python
# 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', 'ourse
            "you'll", "you'd", 'your', 'yours', 'yourself', 'yourselves', 'he', 'hi
            'she', "she's", 'her', 'hers', 'herself', 'it', "it's", 'its', 'itself'
            'theirs', 'themselves', 'what', 'which', 'who', 'whom', 'this', 'that',
            'am', 'is', 'are', 'was', 'were', 'be', 'been', 'being', 'have', 'has',
            'did', 'doing', 'a', 'an', 'the', 'and', 'but', 'if', 'or', 'because',
            'at', 'by', 'for', 'with', 'about', 'against', 'between', 'into', 'thro
            'above', 'below', 'to', 'from', 'up', 'down', 'in', 'out', 'on', 'off',
            'then', 'once', 'here', 'there', 'when', 'where', 'why', 'how', 'all',
            'most', 'other', 'some', 'such', 'only', 'own', 'same', 'so', 'than', '
            's', 't', 'can', 'will', 'just', 'don', "don't", 'should', "should've",
            've', 'y', 'ain', 'aren', "aren't", 'couldn', "couldn't", 'didn', "didn
            "hadn't", 'hasn', "hasn't", 'haven', "haven't", 'isn', "isn't", 'ma', '
            "mustn't", 'needn', "needn't", 'shan', "shan't", 'shouldn', "shouldn't"
            'won', "won't", 'wouldn', "wouldn't"])
```

In [22]:

```python
# 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 sto
    preprocessed_reviews.append(sentance.strip())
```

```
100%|██████████| 28072/28072 [00:11<00:00, 2392.03it/s]
```

In [23]:

```python
preprocessed_reviews[1500]
```

Out[23]:

```
'favorite stevia product subscribe save queried customer service nunat
urals gmo use yes nunaturals uses gmo products canceled subscribe save
order using'
```

[3.2] Preprocessing Review Summary

In [24]:

```python
## Similartly you can do preprocessing for review summary also.
```

[4] Featurization

[4.1] BAG OF WORDS

#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])

[4.2] Bi-Grams and n-Grams.

4g#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.feature_extra [(http://scikit-learn.org/stable/modules/generated/sklearn.feature_extra](http://scikit-learn.org/stable/modules/generated/sklearn.feature_extra)

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])

Type *Markdown* and LaTeX: α^2

[4.3] TF-IDF

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])

[4.4] Word2Vec

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())

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/0B7XkCwpI5KDYNlNUTTlS (https://drive.google.com/file/d/0B7XkCwpI5KDYNlNUTTl

it's 1.9GB in size.

http://kavita-ganesan.com/gensim-word2vec-tutorial-starter-code/#.W17SRFAzZPY (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 ")

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])

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

[4.4.1.1] Avg W2v

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]))

[4.4.1.2] TFIDF weighted W2v

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_)))

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
```

[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=50
    0)
                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/) 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/)

[[](http://)](http://)

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 [25]:

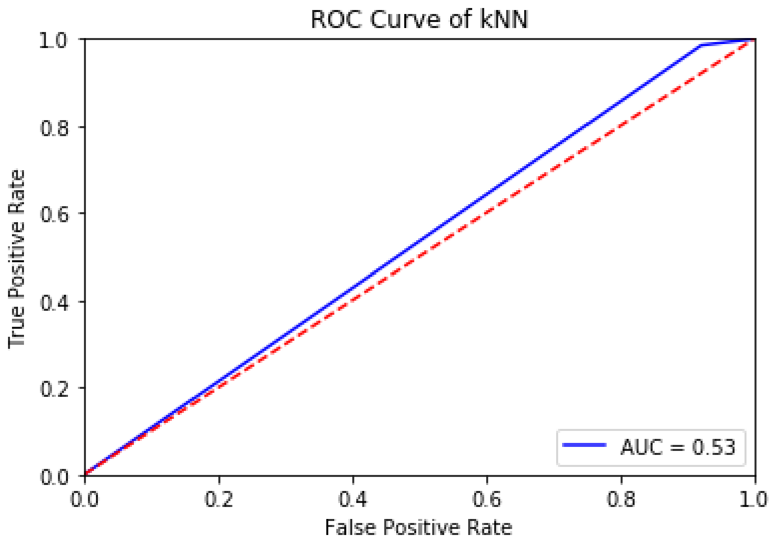
```
ing KNN brute force

x_tr,x_cv, y_tr, y_cv):

[]
ange(1,30,2):

ighborsClassifier(n_neighbors=i,algorithm='brute',n_jobs=-1)
ross_val_score(knn,x_tr, y_tr, cv=6, scoring='accuracy',n_jobs=-1)
s.mean()
s.append(d)

2*cv_scores.index(max(cv_scores)) +1
 optimum value of k is :",optimum_K)
hborsClassifier(n_neighbors=optimum_K,algorithm='brute',n_jobs=-1)
tr,y_tr)
.predict(x_cv)
uracy of the classifier:",accuracy_score(y_cv,y_pred))
tackoverflow.com/questions/52910061/implementing-roc-curves-for-k-nn-machine-learnin
threshold = roc_curve(y_cv, y_pred)
auc(fpr, tpr)
'Receiver Operating Characteristic')
pr, tpr, 'b', label = 'AUC = %0.2f' % roc_auc)
(loc = 'lower right')
0, 1], [0, 1],'r--')
0, 1])
0, 1])
('True Positive Rate')
('False Positive Rate')
'ROC Curve of kNN')

rn.metrics import confusion_matrix
ion_matrix(y_cv,y_pred)

l = ["negative", "positive"]
.DataFrame(cm, index = class_label, columns = class_label)
p(df_cm, annot = True, fmt = "d")
'Confusion Matrix")
("Predicted Label")
("True Label")

sification_report(y_cv, y_pred))
```

[5.1.1] Applying KNN brute force on BOW, SET 1

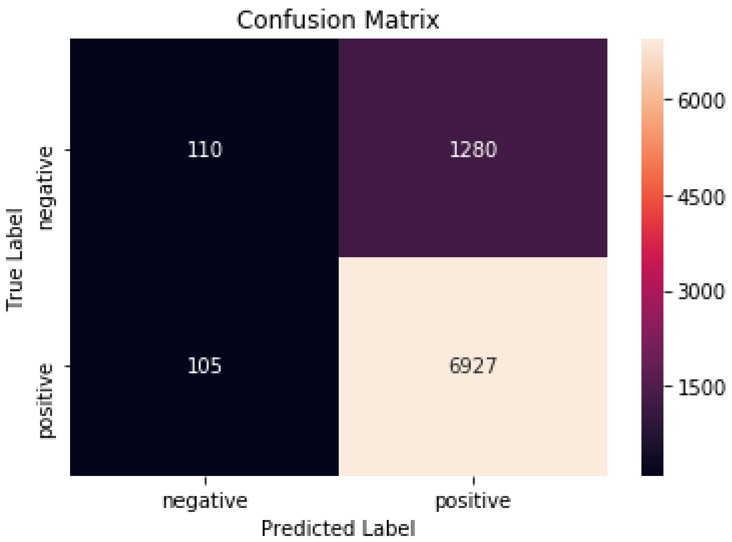
In [26]:

```
# Please write all the code with proper documentation
x_tr,x_cv, y_tr, y_cv= train_test_split(preprocessed_reviews,final['Score'],test_si
vect = CountVectorizer()
vect= vect.fit(preprocessed_reviews)
x_tr=vect.transform(x_tr)
x_cv= vect.transform(x_cv)
brute_Knn(x_tr,x_cv, y_tr, y_cv)
```

```
1
3
5
7
9
11
13
15
17
19
21
23
25
27
29
the optimum value of k is : 29
accuracy of the classifier: 0.8355497506530515
```




```
[[ 110 1280]
 [ 105 6927]]
```



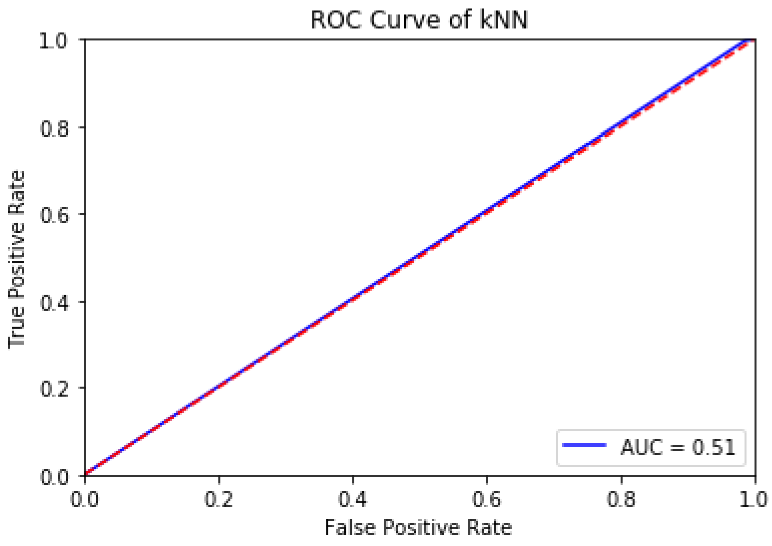
	precision	recall	f1-score	support
0	0.51	0.08	0.14	1390
1	0.84	0.99	0.91	7032
micro avg	0.84	0.84	0.84	8422
macro avg	0.68	0.53	0.52	8422
weighted avg	0.79	0.84	0.78	8422

[5.1.2] Applying KNN brute force on TFIDF, SET 2

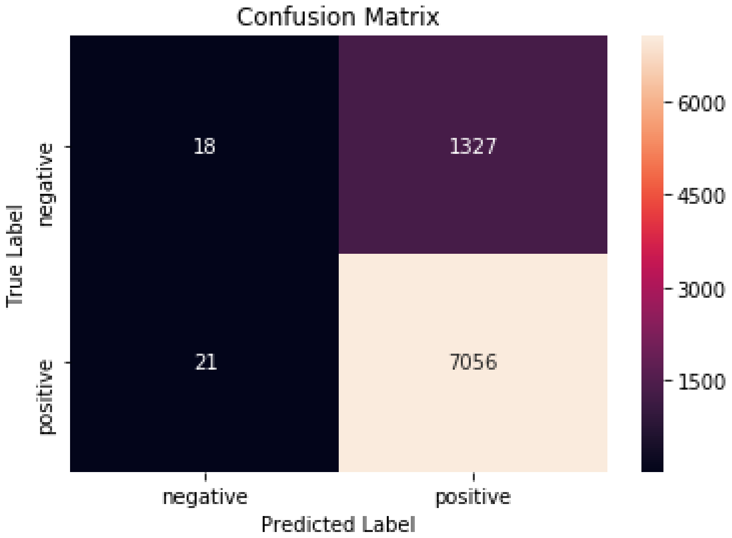
In [27]:

```
# Please write all the code with proper documentation
tf_idf_vect = TfidfVectorizer(ngram_range=(1,2), min_df=10)
tf_idf_vect.fit(preprocessed_reviews)
final_tf_idf = tf_idf_vect.transform(preprocessed_reviews)
x_tr,x_cv, y_tr, y_cv= train_test_split(final_tf_idf,final['Score'],test_size=0.3)
brute_Knn(x_tr,x_cv, y_tr, y_cv)
```

```
1
3
5
7
9
11
13
15
17
19
21
23
25
27
29
the optimum value of k is : 1
accuracy of the classifier: 0.8399430064117787
```




```
[[  18 1327]
 [  21 7056]]
```



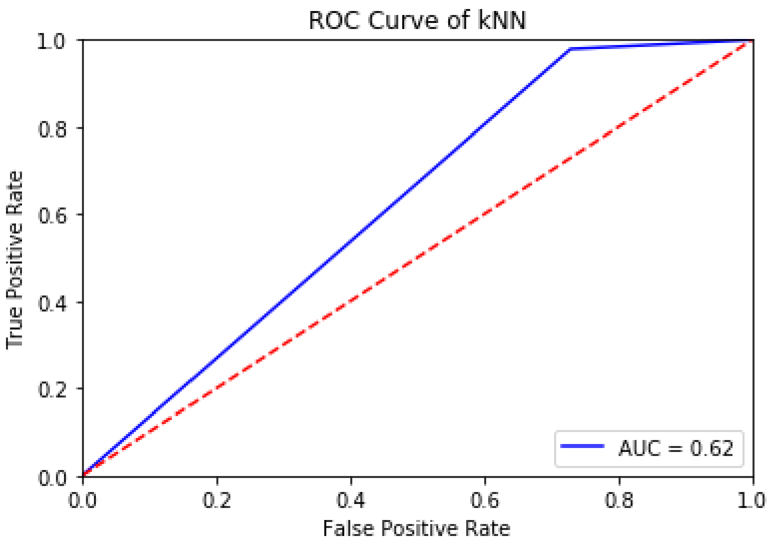

```
              precision    recall  f1-score   support

           0       0.46      0.01      0.03      1345
           1       0.84      1.00      0.91      7077

   micro avg       0.84      0.84      0.84      8422
   macro avg       0.65      0.51      0.47      8422
weighted avg       0.78      0.84      0.77      8422
```

[5.1.3] Applying KNN brute force on AVG W2V, SET 3

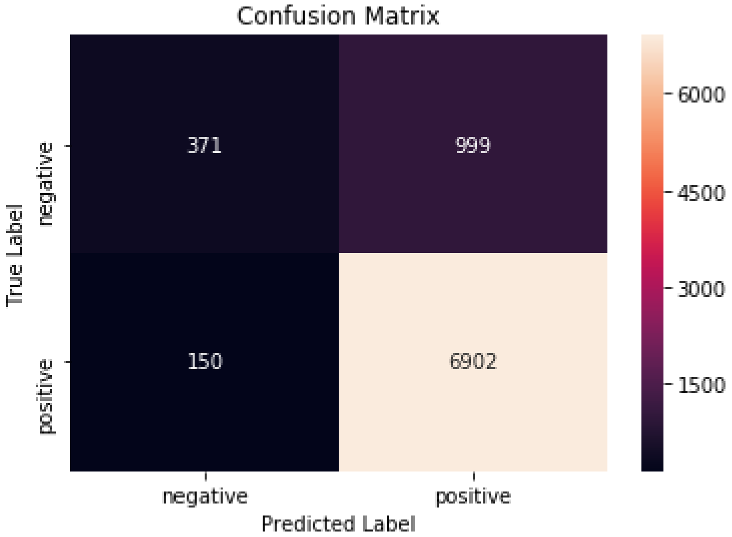
In [28]:

```
e all the code with proper documentation
nce=[]
in preprocessed_reviews:
entance.append(sentance.split())
d2Vec(list_of_sentance,min_count=5,size=50, workers=4)
ist(w2v_model.wv.vocab)
= []; # the avg-w2v for each sentence/review is stored in this list
qdm(list_of_sentance): # for each review/sentence
= np.zeros(50) # as word vectors are of zero length 50, you might need to change thi
 =0; # num of words with a valid vector in the sentence/review
in sent: # for each word in a review/sentence
rd in w2v_words:
ec = w2v_model.wv[word]
ent_vec += vec
nt_words += 1
rds != 0:
vec /= cnt_words
ors.append(sent_vec)
tr, y_cv= train_test_split(sent_vectors,final['Score'],test_size=0.3)
r,x_cv, y_tr, y_cv)
```

```
100%|██████████| 28072/28072 [01:03<00:00, 439.64it/s]

1
3
5
7
9
11
13
15
17
19
21
23
25
27
29
the optimum value of k is : 15
accuracy of the classifier: 0.863571598195203
```



```
[[ 371  999]
 [ 150 6902]]
```



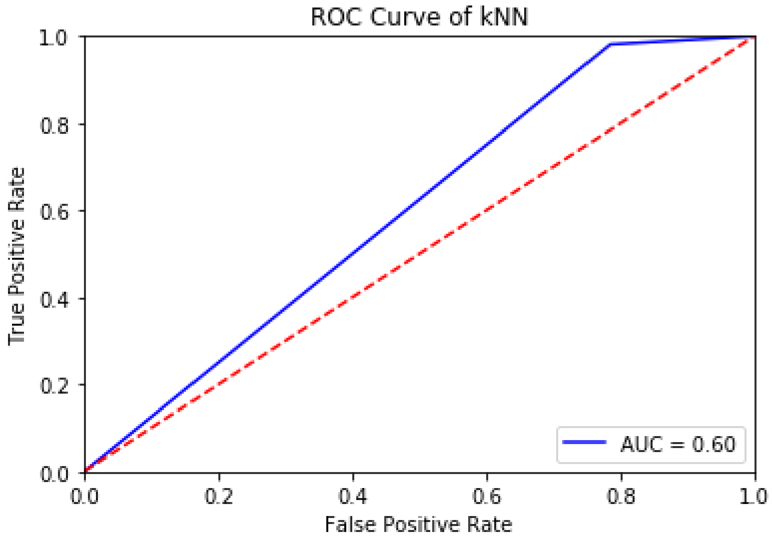

```
              precision    recall  f1-score   support

           0       0.71      0.27      0.39      1370
           1       0.87      0.98      0.92      7052

   micro avg       0.86      0.86      0.86      8422
   macro avg       0.79      0.62      0.66      8422
weighted avg       0.85      0.86      0.84      8422
```

[5.1.4] Applying KNN brute force on TFIDF W2V, SET 4

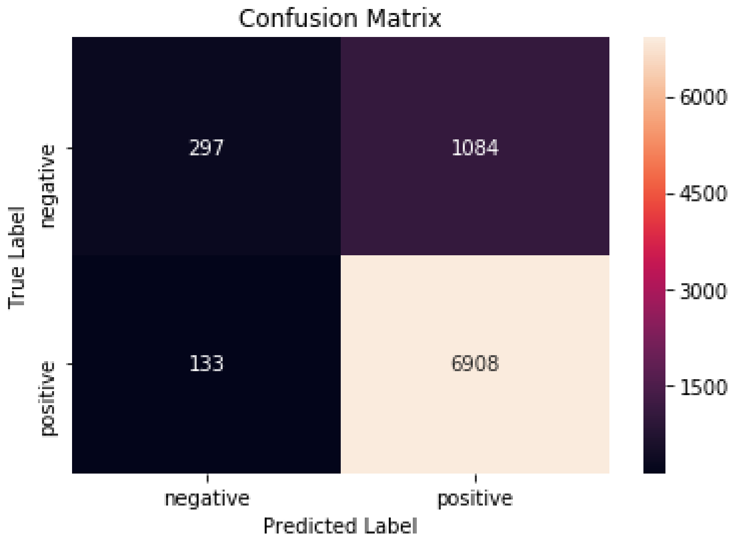
In [29]:

```
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_)))
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 = tfi

tfidf_sent_vectors = []; # the tfidf-w2v for each sentence/review is stored in this
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
x_tr,x_cv, y_tr, y_cv= train_test_split(tfidf_sent_vectors,final['Score'],test_size
brute_Knn(x_tr,x_cv, y_tr, y_cv)
```

```
100%|██████████| 28072/28072 [10:42<00:00, 43.72it/s]

1
3
5
7
9
11
13
15
17
19
21
23
25
27
29
the optimum value of k is : 15
accuracy of the classifier: 0.8554975065305154
```




```
[[ 297 1084]
 [ 133 6908]]
```




```
              precision    recall  f1-score   support

           0       0.69      0.22      0.33      1381
           1       0.86      0.98      0.92      7041

   micro avg       0.86      0.86      0.86      8422
   macro avg       0.78      0.60      0.62      8422
weighted avg       0.84      0.86      0.82      8422
```

In [ ]:

In [30]:

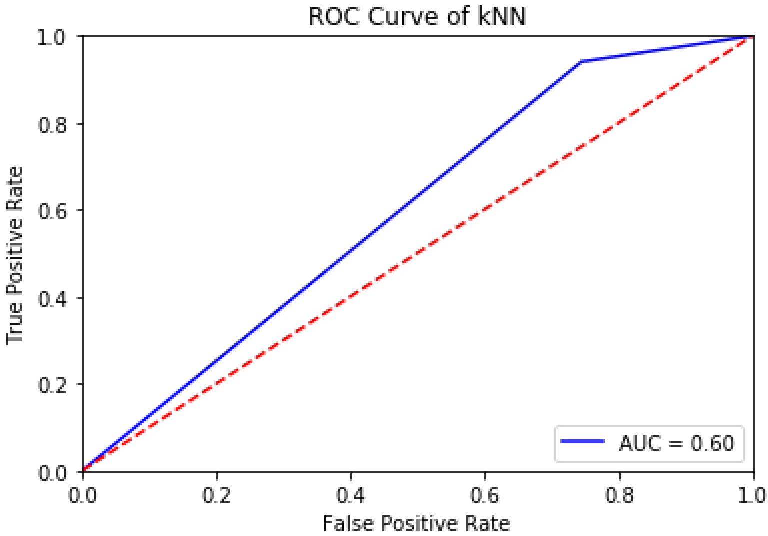
```
## [5.2] Applying KNN kd-tree
def kdtree_Knn(x_tr,x_cv, y_tr, y_cv):

    cv_scores=[]
    for i in range(1,30,2):

      knn= KNeighborsClassifier(n_neighbors=i,algorithm='kd_tree',n_jobs=-1)
      scores=cross_val_score(knn,x_tr, y_tr, cv=6, scoring='accuracy',n_jobs=-1)
      d= round(scores.mean(),2)
      cv_scores.append(d)
      print(i)

    optimum_K=2*cv_scores.index(max(cv_scores)) +1
    print("the optimum value of K is:",optimum_K)
    knn= KNeighborsClassifier(n_neighbors=optimum_K,algorithm='kd_tree',n_jobs=-1)
    knn.fit(x_tr,y_tr)
    y_pred=knn.predict(x_cv)
    print("accuracy of the classifier:",accuracy_score(y_cv,y_pred))
   # https://stackoverflow.com/questions/52910061/implementing-roc-curves-for-k-nn-
    fpr, tpr, threshold = roc_curve(y_cv, y_pred)
    roc_auc = auc(fpr, tpr)
    plt.title('Receiver Operating Characteristic')
    plt.plot(fpr, tpr, 'b', label = 'AUC = %0.2f' % roc_auc)
    plt.legend(loc = 'lower right')
    plt.plot([0, 1], [0, 1],'r--')
    plt.xlim([0, 1])
    plt.ylim([0, 1])
    plt.ylabel('True Positive Rate')
    plt.xlabel('False Positive Rate')
    plt.title('ROC Curve of kNN')
    plt.show()
    from sklearn.metrics import confusion_matrix
    cm= confusion_matrix(y_cv,y_pred)
    print(cm)
    class_label = ["negative", "positive"]
    df_cm = pd.DataFrame(cm, index = class_label, columns = class_label)
    sns.heatmap(df_cm, annot = True, fmt = "d")
    plt.title("Confusion Matrix")
    plt.xlabel("Predicted Label")
    plt.ylabel("True Label")
    plt.show()
    from sklearn.metrics import confusion_matrix
    print(classification_report(y_cv, y_pred))
```

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

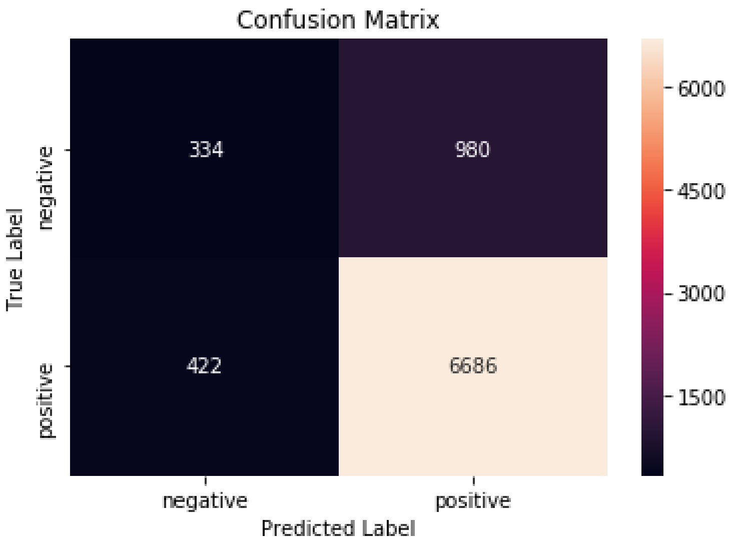
In [31]:

```
# Please write all the code with proper documentation
x_tr,x_cv, y_tr, y_cv= train_test_split(preprocessed_reviews,final['Score'],test_si
vect = CountVectorizer(min_df=10, max_features=500)
x_tr = vect.fit_transform(x_tr)
x_cv= vect.transform(x_cv)
from scipy.sparse import csr_matrix, issparse
x_tr=x_tr.todense()
x_cv=x_cv.todense()

kdtree_Knn(x_tr,x_cv, y_tr, y_cv)
```

```
1
3
5
7
9
11
13
15
17
19
21
23
25
27
29
the optimum value of K is: 9
accuracy of the classifier: 0.8335312277368796
```




```
[[ 334  980]
 [ 422 6686]]
```



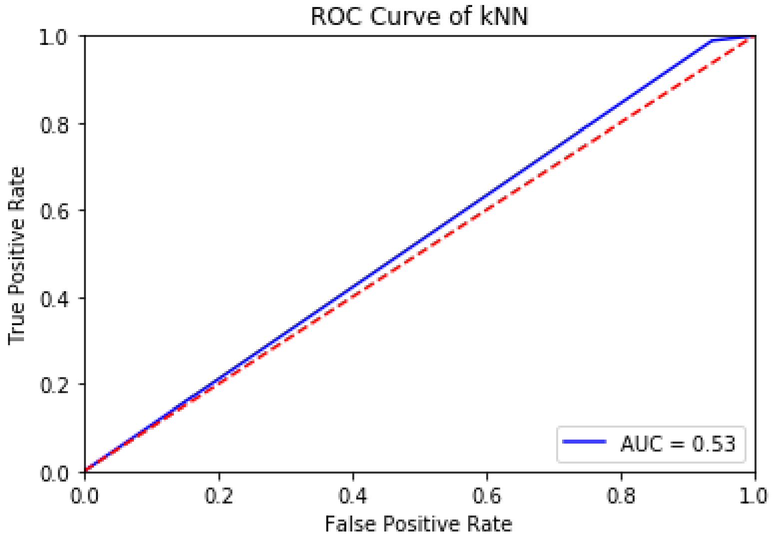

```
              precision    recall  f1-score   support

           0       0.44      0.25      0.32      1314
           1       0.87      0.94      0.91      7108

   micro avg       0.83      0.83      0.83      8422
   macro avg       0.66      0.60      0.61      8422
weighted avg       0.81      0.83      0.81      8422
```

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

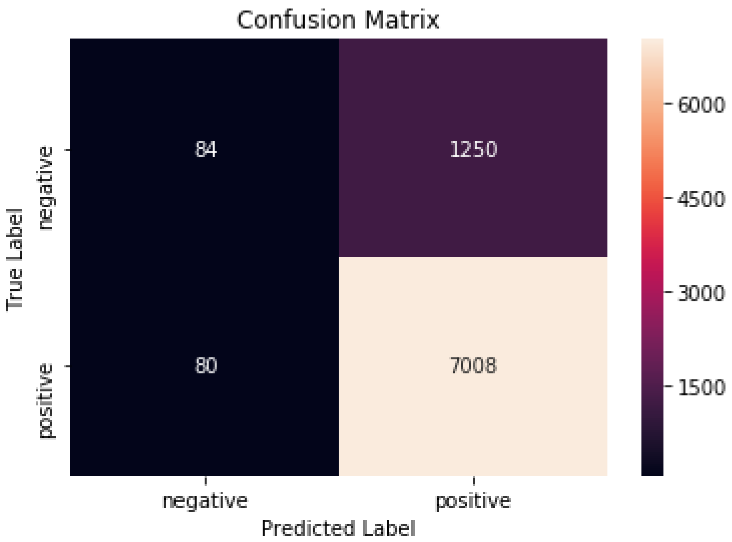
In [35]:

```
# Please write all the code with proper documentation
tf_idf_vect = TfidfVectorizer(ngram_range=(1,2),min_df=10, max_features=500 )
tf_idf_vect.fit(preprocessed_reviews)
final_tf_idf = tf_idf_vect.transform(preprocessed_reviews)
x_tr,x_cv, y_tr, y_cv= train_test_split(final_tf_idf,final['Score'],test_size=0.3)
x_tr=x_tr.todense()
x_cv=x_cv.todense()
kdtree_Knn(x_tr,x_cv, y_tr, y_cv)
```

```
1
3
5
7
9
11
13
15
17
19
21
23
25
27
29
the optimum value of K is: 5
accuracy of the classifier: 0.8420802659700783
```




```
[[  84 1250]
 [  80 7008]]
```



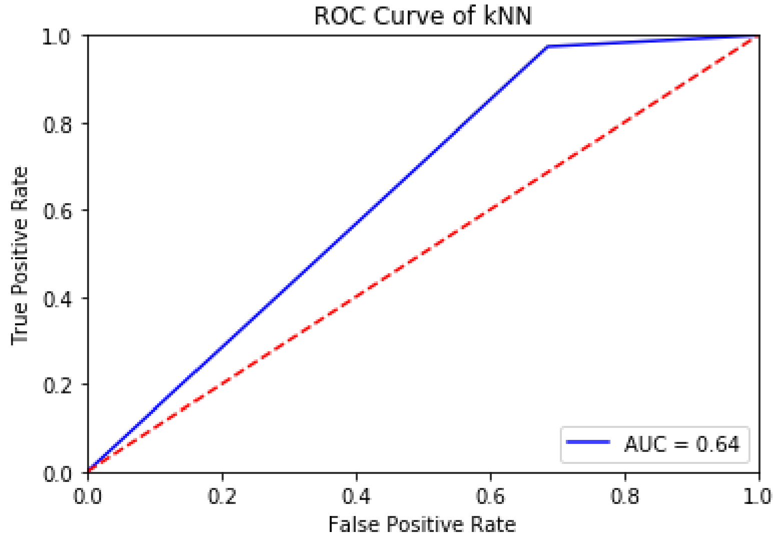
	precision	recall	f1-score	support
0	0.51	0.06	0.11	1334
1	0.85	0.99	0.91	7088
micro avg	0.84	0.84	0.84	8422
macro avg	0.68	0.53	0.51	8422
weighted avg	0.80	0.84	0.79	8422

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

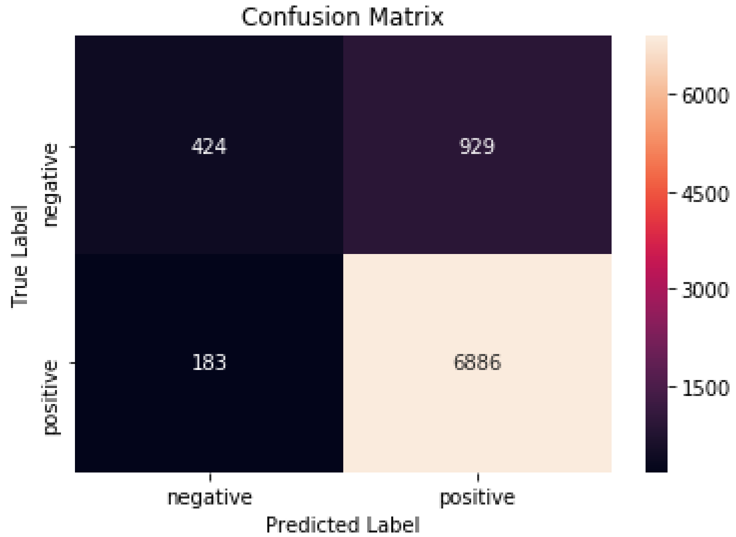
In [33]:

```
# Please write all the code with proper documentation

x_tr,x_cv, y_tr, y_cv= train_test_split(sent_vectors,final['Score'],test_size=0.3)
kdtree_Knn(x_tr,x_cv, y_tr, y_cv)
```

```
1
3
5
7
9
11
13
15
17
19
21
23
25
27
29
the optimum value of K is: 11
accuracy of the classifier: 0.8679648539539302
```




```
[[ 424  929]
 [ 183 6886]]
```



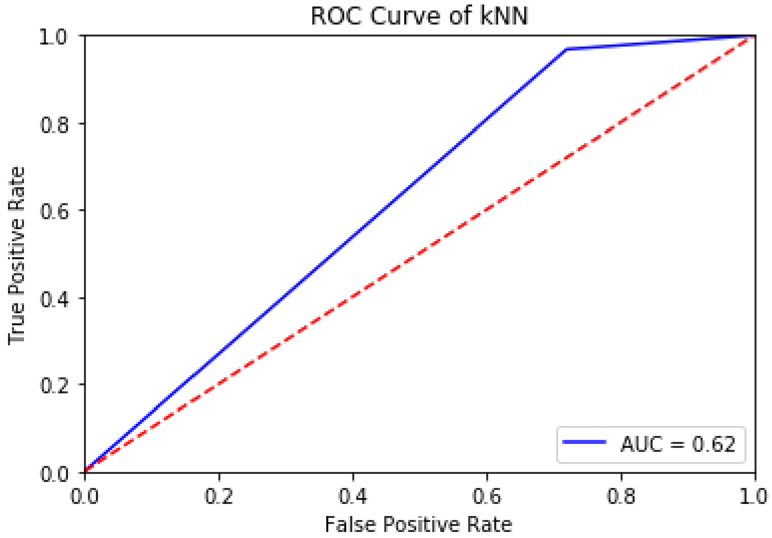
```
              precision    recall  f1-score   support

           0       0.70      0.31      0.43      1353
           1       0.88      0.97      0.93      7069

   micro avg       0.87      0.87      0.87      8422
   macro avg       0.79      0.64      0.68      8422
weighted avg       0.85      0.87      0.85      8422
```

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

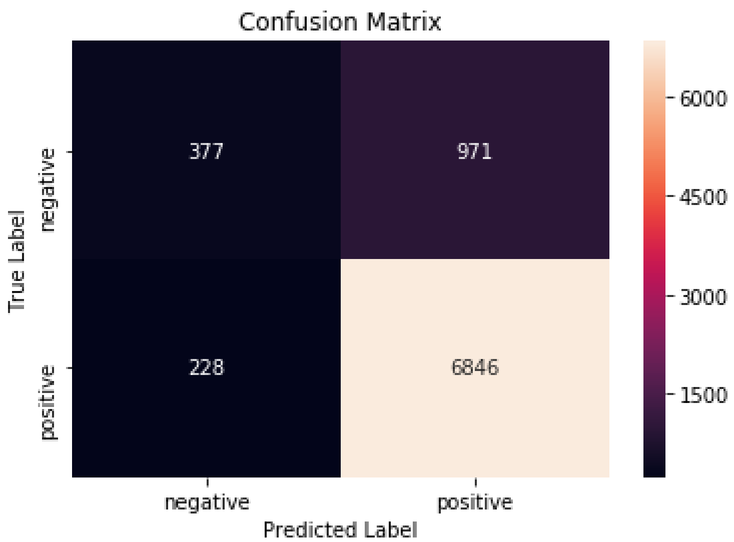
In [34]:

```
# Please write all the code with proper documentation
x_tr,x_cv, y_tr, y_cv= train_test_split(tfidf_sent_vectors,final['Score'],test_size
kdtree_Knn(x_tr,x_cv, y_tr, y_cv)
```

```
1
3
5
7
9
11
13
15
17
19
21
23
25
27
29
the optimum value of K is: 7
accuracy of the classifier: 0.8576347660888151
```




```
[[ 377  971]
 [ 228 6846]]
```



```
              precision    recall  f1-score   support

           0       0.62      0.28      0.39      1348
           1       0.88      0.97      0.92      7074

   micro avg       0.86      0.86      0.86      8422
   macro avg       0.75      0.62      0.65      8422
weighted avg       0.84      0.86      0.83      8422
```

[6] Conclusions

In [8]:

```python
# Please compare all your models using Prettytable library
#https://stackoverflow.com/questions/36423259/how-to-use-pretty-table-in-python-to-
from prettytable import PrettyTable
x = PrettyTable(["Vectorizer", "Model", "Hyper parameter","AUC","ACCURACY"])
x.add_row(["BOW","Brute",29,0.53,83])
x.add_row(["TFIDF","Brute",1,0.51,83])
x.add_row(["AVG W2V","Brute",15,0.62,86])
x.add_row(["TFIDF W2V","Brute",15,0.60,85])
x.add_row(["BOW","KDtree",9,0.60,83])
x.add_row(["TFIDF","KDtree",5,0.53,84])
x.add_row(["AVG W2V","KDtree",11,0.64,86])
x.add_row(["TFIDF W2V","KDtree",7,0.62,85])
print(x)
```

```
+------------+--------+-----------------+------+----------+
| Vectorizer | Model  | Hyper parameter | AUC  | ACCURACY |
+------------+--------+-----------------+------+----------+
|    BOW     | Brute  |        29       | 0.53 |    83    |
|   TFIDF    | Brute  |        1        | 0.51 |    83    |
|  AVG W2V   | Brute  |        15       | 0.62 |    86    |
| TFIDF W2V  | Brute  |        15       | 0.6  |    85    |
|    BOW     | KDtree |        9        | 0.6  |    83    |
|   TFIDF    | KDtree |        5        | 0.53 |    84    |
|  AVG W2V   | KDtree |        11       | 0.64 |    86    |
| TFIDF W2V  | KDtree |        7        | 0.62 |    85    |
+------------+--------+-----------------+------+----------+
```

1.After running all the vectorizers on brute and KD tree models they have accuracy around 85% on test data.
2.If we look into confusion matrix and classifiaction report the classifier performs poor in predicting negative reviews.
3.Even though the classifier has 85% it is not able to predict negative reviews this seems to be overfitting problem
4.The classifier might work well if we take balanced data.
5.AVG W2V vetorizer have seems to work well for both Kdtree and brute models compared to accuracy of other vectorizers.