Assignment03 - Amazon Fine Food Reviews Analysis_KNN

January 30, 2019

1 Amazon Fine Food Reviews Analysis

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

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

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

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

Attribute Information:

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

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

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

2 [1]. Reading Data

2.1 [1.1] Loading the data

The dataset is available in two forms 1. .csv file 2. SQLite Database

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

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

In [2]:
```
%matplotlib inline
import warnings
warnings.filterwarnings("ignore")

from prettytable import PrettyTable
import sqlite3
import pandas as pd
import numpy as np
import nltk
import string
import matplotlib.pyplot as plt
import seaborn as sns
from sklearn.feature_extraction.text import TfidfTransformer
from sklearn.feature_extraction.text import TfidfVectorizer

from sklearn.feature_extraction.text import CountVectorizer
from sklearn.metrics import confusion_matrix
from sklearn import metrics
from sklearn.metrics import roc_curve, auc
from nltk.stem.porter import PorterStemmer

import re
# Tutorial about Python regular expressions: https://pymotw.com/2/re/
import string
from nltk.corpus import stopwords
from nltk.stem import PorterStemmer
from nltk.stem.wordnet import WordNetLemmatizer

from gensim.models import Word2Vec
from gensim.models import KeyedVectors
import pickle
from scikitplot import *

from tqdm import tqdm
import os
```

In [3]:
```
# using SQLite Table to read data.

data_path = '/home/monodeepdas112/Datasets/amazon-fine-food-reviews/database.sqlite'
con = sqlite3.connect(data_path)

# 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 point
```

```
# 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 50
# for tsne assignment you can take 5k data points

filtered_data = pd.read_sql_query(""" SELECT * FROM Reviews WHERE Score != 3 LIMIT 5000

# Give reviews with Score>3 a positive rating(1), and reviews with a score<3 a negativ
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 (5000, 10)
```

```
Out[3]:    Id   ProductId          UserId                      ProfileName  \
        0   1  B001E4KFG0  A3SGXH7AUHU8GW                       delmartian
        1   2  B00813GRG4  A1D87F6ZCVE5NK                           dll pa
        2   3  B000LQOCH0   ABXLMWJIXXAIN  Natalia Corres "Natalia Corres"

           HelpfulnessNumerator  HelpfulnessDenominator  Score        Time  \
        0                     1                       1      1  1303862400
        1                     0                       0      0  1346976000
        2                     1                       1      1  1219017600

                          Summary                                               Text
        0  Good Quality Dog Food  I have bought several of the Vitality canned d...
        1      Not as Advertised  Product arrived labeled as Jumbo Salted Peanut...
        2  "Delight" says it all  This is a confection that has been around a fe...
```

```
In [4]: display = pd.read_sql_query("""
        SELECT UserId, ProductId, ProfileName, Time, Score, Text, COUNT(*)
        FROM Reviews
        GROUP BY UserId
        HAVING COUNT(*)>1
        """, con)
```

```
In [5]: print(display.shape)
        display.head()
```

```
(80668, 7)
```

```
Out[5]:                UserId   ProductId              ProfileName        Time  Score  \
        0  #oc-R115TNMSPFT9I7  B007Y59HVM                  Breyton  1331510400      2
        1  #oc-R11D9D7SHXIJB9  B005HG9ET0  Louis E. Emory "hoppy"  1342396800      5
        2  #oc-R11DNU2NBKQ23Z  B007Y59HVM         Kim Cieszykowski  1348531200      1
        3  #oc-R11O5J5ZVQE25C  B005HG9ET0            Penguin Chick  1346889600      5
        4  #oc-R12KPBODL2B5ZD  B007OSBE1U    Christopher P. Presta  1348617600      1

                                                        Text  COUNT(*)
        0  Overall its just OK when considering the price...         2
        1  My wife has recurring extreme muscle spasms, u...         3
        2  This coffee is horrible and unfortunately not ...         2
        3  This will be the bottle that you grab from the...         3
        4  I didnt like this coffee. Instead of telling y...         2
```

```
In [6]: display[display['UserId']=='AZY10LLTJ71NX']
```

```
Out[6]:               UserId   ProductId                    ProfileName        Time  \
        80638  AZY10LLTJ71NX  B006P7E5ZI  undertheshrine "undertheshrine"  1334707200

               Score                                               Text  COUNT(*)
        80638      5  I was recommended to try green tea extract to ...         5
```

```
In [7]: display['COUNT(*)'].sum()
```

```
Out[7]: 393063
```

3 [2] Exploratory Data Analysis

3.1 [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 [8]: display= pd.read_sql_query("""
        SELECT *
        FROM Reviews
        WHERE Score != 3 AND UserId="AR5J8UI46CURR"
        ORDER BY ProductID
        """, con)
        display.head()
```

```
Out[8]:        Id   ProductId         UserId      ProfileName  HelpfulnessNumerator  \
        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
```

```
   HelpfulnessDenominator  Score        Time  \
0                       2      5  1199577600
1                       2      5  1199577600
2                       2      5  1199577600
3                       2      5  1199577600
4                       2      5  1199577600

                             Summary  \
0  LOACKER QUADRATINI VANILLA WAFERS
1  LOACKER QUADRATINI VANILLA WAFERS
2  LOACKER QUADRATINI VANILLA WAFERS
3  LOACKER QUADRATINI VANILLA WAFERS
4  LOACKER QUADRATINI VANILLA WAFERS

                                                Text
0  DELICIOUS WAFERS. I FIND THAT EUROPEAN WAFERS ...
1  DELICIOUS WAFERS. I FIND THAT EUROPEAN WAFERS ...
2  DELICIOUS WAFERS. I FIND THAT EUROPEAN WAFERS ...
3  DELICIOUS WAFERS. I FIND THAT EUROPEAN WAFERS ...
4  DELICIOUS WAFERS. I FIND THAT EUROPEAN WAFERS ...
```

As it can be seen above that same user has multiple reviews with same values for HelpfulnessNumerator, HelpfulnessDenominator, Score, Time, Summary and Text and on doing analysis it was found that ProductId=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 [9]: #Sorting data according to ProductId in ascending order
        sorted_data=filtered_data.sort_values('ProductId', axis=0, ascending=True, inplace=Fals
```

```
In [10]: #Deduplication of entries
         final=sorted_data.drop_duplicates(subset={"UserId","ProfileName","Time","Text"}, keep=
         final.shape
```

```
Out[10]: (4986, 10)
```

```
In [11]: #Checking to see how much % of data still remains
         (final['Id'].size*1.0)/(filtered_data['Id'].size*1.0)*100
```

```
Out[11]: 99.72
```

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

         display.head()
```

```
Out[12]:       Id   ProductId          UserId              ProfileName  \
         0  64422  B000MIDROQ  A161DK06JJMCYF  J. E. Stephens "Jeanne"
         1  44737  B001EQ55RW  A2V0I904FH7ABY                      Ram

            HelpfulnessNumerator  HelpfulnessDenominator  Score        Time  \
         0                     3                       1      5  1224892800
         1                     3                       2      4  1212883200

                                           Summary  \
         0             Bought This for My Son at College
         1  Pure cocoa taste with crunchy almonds inside

                                                         Text
         0  My son loves spaghetti so I didn't hesitate or...
         1  It was almost a 'love at first bite' - the per...
```

```
In [13]: final=final[final.HelpfulnessNumerator<=final.HelpfulnessDenominator]
```

```
In [14]: #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()
```

```
(4986, 10)
```

```
Out[14]: 1    4178
         0     808
         Name: Score, dtype: int64
```

4 [3] Preprocessing

4.1 [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 [15]:
```
# 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)
```

```
Why is this $[...] when the same product is available for $[...] here?<br />http://www.amazon.
==================================================
I recently tried this flavor/brand and was surprised at how delicious these chips are.  The bes
==================================================
Wow.  So far, two two-star reviews.  One obviously had no idea what they were ordering; the ot
==================================================
love to order my coffee on amazon.  easy and shows up quickly.<br />This k cup is great coffee
==================================================
```

In [16]:
```
# 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)
```

```
Why is this $[...] when the same product is available for $[...] here?<br /> /><br />The Victo
```

```
In [17]: # https://stackoverflow.com/questions/16206380/python-beautifulsoup-how-to-remove-all
         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)
```

```
Why is this $[...] when the same product is available for $[...] here? />The Victor M380 and M5
==================================================
I recently tried this flavor/brand and was surprised at how delicious these chips are.  The bes
==================================================
Wow.  So far, two two-star reviews.  One obviously had no idea what they were ordering; the oth
==================================================
love to order my coffee on amazon.  easy and shows up quickly.This k cup is great coffee.  dca
```

```
In [18]: # 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 [19]: sent_1500 = decontracted(sent_1500)
         print(sent_1500)
         print("="*50)
```

```
Wow.  So far, two two-star reviews.  One obviously had no idea what they were ordering; the ot
==================================================
```

```
In [20]: #remove words with numbers python: https://stackoverflow.com/a/18082370/4084039
         sent_0 = re.sub("\S*\d\S*", "", sent_0).strip()
         print(sent_0)
```

```
Why is this $[...] when the same product is available for $[...] here?<br /> /><br />The Victo
```

```
In [21]: #remove spacial character: https://stackoverflow.com/a/5843547/4084039
         sent_1500 = re.sub('[^A-Za-z0-9]+', ' ', sent_1500)
         print(sent_1500)
```

```
Wow So far two two star reviews One obviously had no idea what they were ordering the other wa
```

```
In [22]: # 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', 'ourselve
                     "you'll", "you'd", 'your', 'yours', 'yourself', 'yourselves', 'he', 'him'
                     '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', 'a
                     'at', 'by', 'for', 'with', 'about', 'against', 'between', 'into', 'throug
                     'above', 'below', 'to', 'from', 'up', 'down', 'in', 'out', 'on', 'off', '
                     'then', 'once', 'here', 'there', 'when', 'where', 'why', 'how', 'all', 'a
                     'most', 'other', 'some', 'such', 'only', 'own', 'same', 'so', 'than', 'to
                     's', 't', 'can', 'will', 'just', 'don', "don't", 'should', "should've", '
                     've', 'y', 'ain', 'aren', "aren't", 'couldn', "couldn't", 'didn', "didn't
                     "hadn't", 'hasn', "hasn't", 'haven', "haven't", 'isn', "isn't", 'ma', 'mi
                     "mustn't", 'needn', "needn't", 'shan', "shan't", 'shouldn', "shouldn't",
                     'won', "won't", 'wouldn', "wouldn't"])
```

```
In [23]: # 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 stopwo
        preprocessed_reviews.append(sentance.strip())
```

```
100%|| 4986/4986 [00:03<00:00, 1366.70it/s]
```

```
In [24]: preprocessed_reviews[1500]
```

```
Out[24]: 'wow far two two star reviews one obviously no idea ordering wants crispy cookies hey
```

[3.2] Preprocessing Review Summary

```
In [25]: ## Similartly you can do preprocessing for review summary also.
         # Combining all the above stundents
         from tqdm import tqdm
         preprocessed_review_summarys = []
         # tqdm is for printing the status bar
         for sentance in tqdm(final['Summary'].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 stopwo
             preprocessed_review_summarys.append(sentance.strip())
```

```
100%|| 4986/4986 [00:02<00:00, 2072.53it/s]
```

5 [4] Featurization

5.1 [4.1] BAG OF WORDS

```
In [26]: #BoW
         count_vect = CountVectorizer() #in scikit-learn
         count_vect.fit(preprocessed_reviews)
         print("some feature names ", count_vect.get_feature_names()[:10])
         print('='*50)

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

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

5.2 [4.2] Bi-Grams and n-Grams.

In [27]:
```
#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/mod

# 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
```

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

5.3 [4.3] TF-IDF

In [28]:
```
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_name
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
```

```
some sample features(unique words in the corpus) ['ability', 'able', 'able find', 'able get',
==================================================
the type of count vectorizer  <class 'scipy.sparse.csr.csr_matrix'>
the shape of out text TFIDF vectorizer  (4986, 3144)
the number of unique words including both unigrams and bigrams  3144
```

5.4 [4.4] Word2Vec

In [29]:
```
# Train your own Word2Vec model using your own text corpus
i=0
list_of_sentance=[]
for sentance in preprocessed_reviews:
    list_of_sentance.append(sentance.split())
```

In [30]:
```
# 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.bi
        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, t
```

```
[('bad', 0.9942300319671631), ('tasty', 0.9934707283973694), ('snack', 0.9932047724723816), ('a
==================================================
[('eaten', 0.9993656873703003), ('goes', 0.9993391036987305), ('varieties', 0.9992413520812988)
```

In [31]:
```
w2v_words = list(w2v_model.wv.vocab)
```

```
print("number of words that occured minimum 5 times ",len(w2v_words))
print("sample words ", w2v_words[0:50])
```

```
number of words that occured minimum 5 times  3817
sample words  ['product', 'available', 'course', 'total', 'pretty', 'stinky', 'right', 'nearby
```

5.5 [4.4.1] Converting text into vectors using Avg W2V, TFIDF-W2V

[4.4.1.1] Avg W2v

In [32]:
```
# 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 t
    cnt_words =0; # num of words with a valid vector in the sentence/review
    for word in sent: # for each word in a review/sentence
        if word in w2v_words:
            vec = w2v_model.wv[word]
            sent_vec += vec
            cnt_words += 1
    if cnt_words != 0:
        sent_vec /= cnt_words
    sent_vectors.append(sent_vec)
print(len(sent_vectors))
print(len(sent_vectors[0]))
```

```
100%|| 4986/4986 [00:10<00:00, 466.46it/s]

4986
50
```

[4.4.1.2] TFIDF weighted W2v

In [33]:
```
# S = ["abc def pqr", "def def def abc", "pqr pqr def"]
model = TfidfVectorizer()
tf_idf_matrix = model.fit_transform(preprocessed_reviews)
# we are converting a dictionary with word as a key, and the idf as a value
dictionary = dict(zip(model.get_feature_names(), list(model.idf_)))
```

In [34]:
```
# 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 l
row=0;
for sent in tqdm(list_of_sentance): # for each review/sentence
    sent_vec = np.zeros(50) # as word vectors are of zero length
    weight_sum =0; # num of words with a valid vector in the sentence/review
    for word in sent: # for each word in a review/sentence
        if word in w2v_words and word in tfidf_feat:
            vec = w2v_model.wv[word]
#             tf_idf = tf_idf_matrix[row, tfidf_feat.index(word)]
            # to reduce the computation we are
            # dictionary[word] = idf value of word in whole courpus
            # sent.count(word) = tf valeus of word in this review
            tf_idf = dictionary[word]*(sent.count(word)/len(sent))
            sent_vec += (vec * tf_idf)
            weight_sum += tf_idf
    if weight_sum != 0:
        sent_vec /= weight_sum
    tfidf_sent_vectors.append(sent_vec)
    row += 1
```

```
100%|| 4986/4986 [01:11<00:00, 69.65it/s]
```

6 [5] Assignment 3: KNN

```
<li><strong>Apply Knn(brute force version) on these feature sets</strong>
    <ul>
        <li><font color='red'>SET 1:</font>Review text, preprocessed one converted into vectors
        <li><font color='red'>SET 2:</font>Review text, preprocessed one converted into vectors
        <li><font color='red'>SET 3:</font>Review text, preprocessed one converted into vectors
        <li><font color='red'>SET 4:</font>Review text, preprocessed one converted into vectors
    </ul>
</li>
<br>
<li><strong>Apply Knn(kd tree version) on these feature sets</strong>
    <br><font color='red'>NOTE: </font>sklearn implementation of kd-tree accepts only dense ma
    <ul>
        <li><font color='red'>SET 5:</font>Review text, preprocessed one converted into vectors
        <pre>
        count_vect = CountVectorizer(min_df=10, max_features=500)
        count_vect.fit(preprocessed_reviews)
        </pre>
        </li>
        <li><font color='red'>SET 6:</font>Review text, preprocessed one converted into vectors
        <pre>
            tf_idf_vect = TfidfVectorizer(min_df=10, max_features=500)
            tf_idf_vect.fit(preprocessed_reviews)
```

```
        </pre>
        </li>
        <li><font color='red'>SET 3:</font>Review text, preprocessed one converted into vectors
        <li><font color='red'>SET 4:</font>Review text, preprocessed one converted into vectors
    </ul>
</li>
<br>
<li><strong>The hyper paramter tuning(find best K)</strong>
    <ul>
<li>Find the best hyper parameter which will give the maximum <a href='https://www.appliedaicou
<li>Find the best hyper paramter using k-fold cross validation or simple cross validation data
<li>Use gridsearch cv or randomsearch cv or you can also write your own for loops to do this ta
    </ul>
</li>
<br>
<li>
<strong>Representation of results</strong>
    <ul>
<li>You need to plot the performance of model both on train data and cross validation data for
<img src='train_cv_auc.JPG' width=300px></li>
<li>Once after you found the best hyper parameter, you need to train your model with it, and fi
<img src='train_test_auc.JPG' width=300px></li>
<li>Along with plotting ROC curve, you need to print the <a href='https://www.appliedaicourse.c
<img src='confusion_matrix.png' width=300px></li>
    </ul>
</li>
<br>
<li><strong>Conclusion</strong>
    <ul>
<li>You need to summarize the results at the end of the notebook, summarize it in the table for
    <img src='summary.JPG' width=400px>
</li>
    </ul>
```

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.

6.1 [5.1] Applying KNN brute force

6.1.1 [5.1.1] Applying KNN brute force on BOW, SET 1

In [50]:
```python
#Getting the necessary imports and function definations
from sklearn.neighbors import KNeighborsClassifier
```

```
from sklearn.metrics import roc_auc_score
from sklearn.model_selection import GridSearchCV
from sklearn.model_selection import train_test_split
import scikitplot as skplt
from sklearn.metrics import confusion_matrix


import os.path
import pickle
```

In [51]:
```
'''This function takes input the X, Y data and algorithm to use and
model path (just to avoiding retraining an already trained model)
Performs GridSearchCV and returns the best parameters and cv_results_
data formatted as a pandas DataFrame'''
def perform_grid_search_cv(X, Y, algorithm, model_path):
    if(os.path.exists(model_path)):
        #if present simply load the model
        with open(model_path, 'rb') as input_file:
            clf = pickle.load(input_file)
    else:
        #if model not present then initialize --> perform cross validation --> save i
        knn = KNeighborsClassifier()
        parameters_grid = {
            'weights' : ['uniform', 'distance'],
            'algorithm' : [algorithm],
            'n_neighbors' : [i for i in range(1,301)]
        }

        clf = GridSearchCV(estimator=knn,
                           param_grid=parameters_grid,
                           scoring='roc_auc',
                           verbose=1,
                           error_score='raise',
                           cv = 10,
                           iid=False,
                           pre_dispatch='4*n_jobs',
                           return_train_score=True, n_jobs=-1)
        #Start to fit the model to get the best hyperparameters
        clf.fit(X, Y)

        #Save the model to the supplied model path
        with open(model_path, 'wb') as output_file:
            pickle.dump(clf, output_file)

    #Displaying the best parameters and best score acheived
    print('\nBest Parameters : {0} that led to Max AUC of ROC : {1}'.format(clf.best_

    #Converting the cv_results to a pandas DataFrame
```

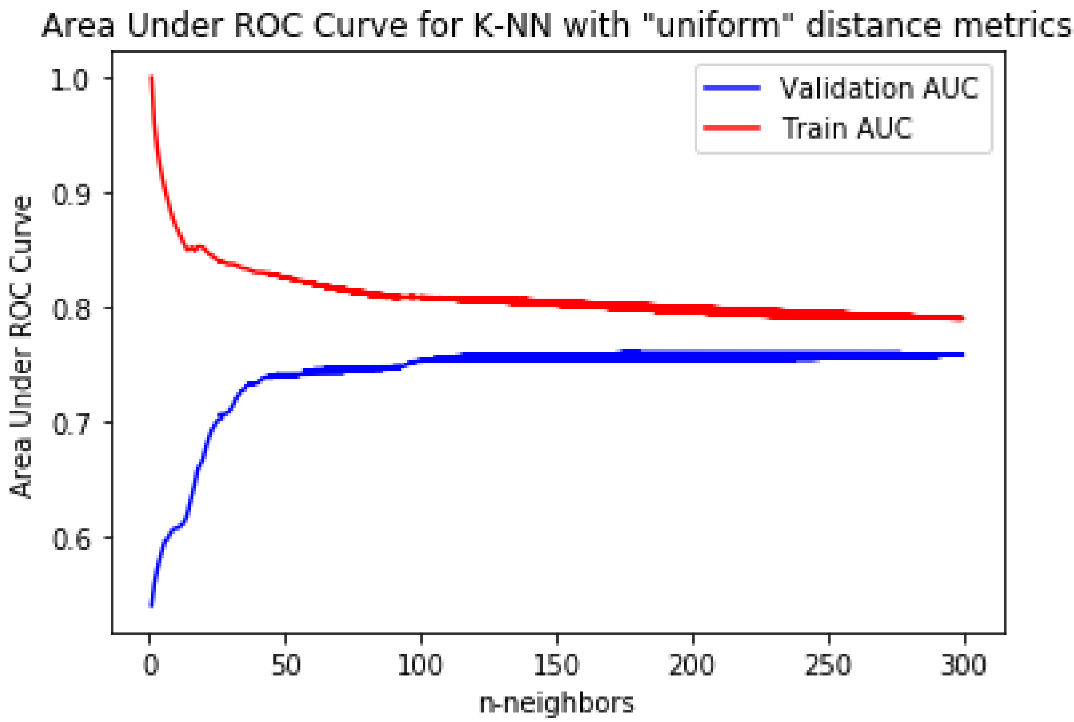
```
    cresults = pd.DataFrame(clf.cv_results_)
    cresults = pd.DataFrame(cresults.loc[:,['param_n_neighbors',
                                            'param_weights',
                                            'rank_test_score',
                                            'mean_train_score',
                                            'mean_test_score',
                                            'std_train_score',
                                            'std_test_score',]])
    return cresults, clf.best_params_, clf.best_score_

def analyse_results(df):
    #Sorting the dataframe as per 1-> best test scores then followed -> number of nei
    cresults = df
    cresults = cresults.sort_values(by=['rank_test_score', 'param_n_neighbors'], ascen

    #seperating the dataframe by the weighing method to maintain uniformity of compar
    uniform_weighted = cresults[cresults['param_weights']=='uniform']
    distance_weighted = cresults[cresults['param_weights']=='distance']

    #plotting the uniform weighted measure K-NN results
    plt.plot(uniform_weighted.param_n_neighbors.tolist(),
             uniform_weighted.mean_test_score.tolist(),
             c='b', label='Validation AUC')
    plt.plot(uniform_weighted.param_n_neighbors.tolist(),
             uniform_weighted.mean_train_score.tolist(),
             c='r', label='Train AUC')
    plt.xlabel('n-neighbors')
    plt.ylabel('Area Under ROC Curve')
    plt.title('Area Under ROC Curve for K-NN with "uniform" distance metrics')
    plt.legend(loc='best')
    plt.show()


    #plotting the inverse-distance weighted measure K-NN results
    plt.plot(distance_weighted.param_n_neighbors.tolist(),
             distance_weighted.mean_test_score.tolist(),
             c='b', label='Validation AUC')
    plt.plot(distance_weighted.param_n_neighbors.tolist(),
             distance_weighted.mean_train_score.tolist(),
             c='r', label='Train AUC')
    plt.xlabel('n-neighbors')
    plt.ylabel('Area Under ROC Curve')
    plt.title('Area Under ROC Curve for K-NN with "inverse-distance" distance metrics
    plt.legend(loc='best')
    plt.show()

def retrain_with_best_hyperparameters(X, Y, best_params_):
    #Initializing the model with the selected best parameters from GridSearchCV
```

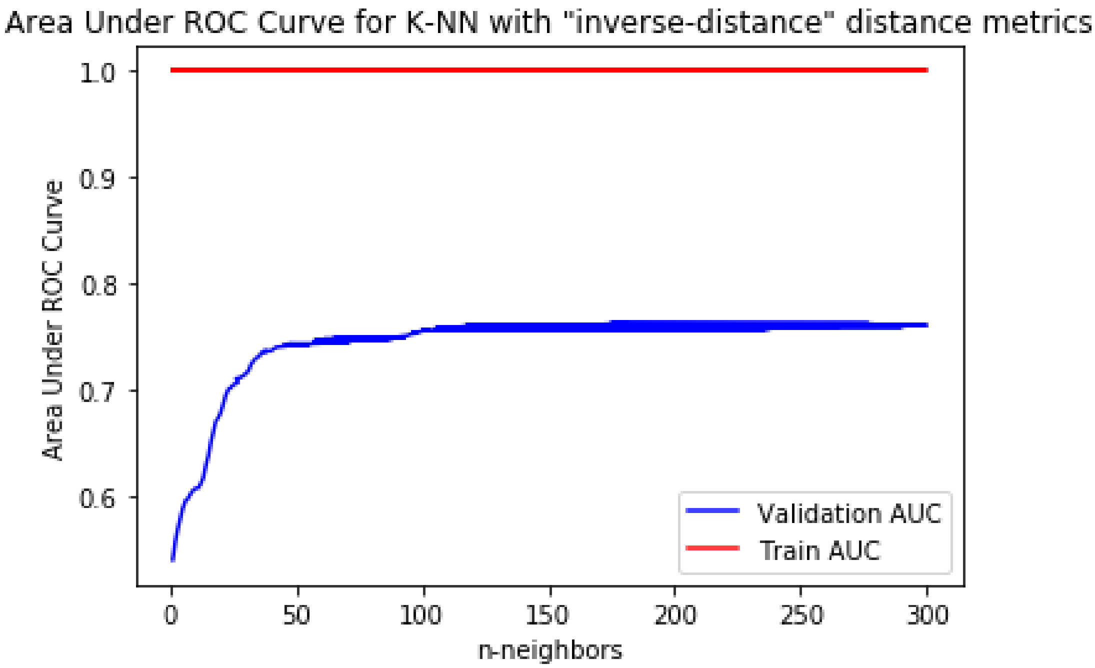
```
    knn = KNeighborsClassifier(algorithm = best_params_['algorithm'],
                               n_neighbors = best_params_['n_neighbors'],
                               weights = best_params_['weights'],
                               n_jobs=-1)

    #Splitting the dataset into train, test splits
    X_train, X_test, y_train, y_test = train_test_split(X, Y, shuffle=True, random_sta
    #Training the model
    knn.fit(X_train, y_train)

    #predicting probability of y_test
    Y_score = knn.predict_proba(X_test)

    #Finding out the ROC_AUC_SCORE
    mroc_auc_score = roc_auc_score(y_test, Y_score[:, 1])
    print('Area Under the Curve : ', mroc_auc_score)

    #Plotting with matplotlib.pyplot
    #ROC Curve for D-train
    tr_fpr, tr_tpr, tr_thresholds = roc_curve(y_train, knn.predict_proba(X_train)[:, 
    plt.plot(tr_fpr, tr_tpr, c='r', label='Train')

    # #ROC Curve for D-test
    te_fpr, te_tpr, te_thresholds = roc_curve(y_test, Y_score[:, 1])
    plt.plot(te_fpr, te_tpr, c='b', label='Test')

    plt.legend(loc='best')
    plt.show()


    #Plotting with scikitplot
    #ROC Curve for D-train
    skplt.metrics.plot_roc(y_train, knn.predict_proba(X_train), title='ROC Curve for 

    #ROC Curve for D-test
    skplt.metrics.plot_roc(y_test, Y_score, title='ROC Curve for Validation')
    plt.show()

    #Confusion Matrix
    con_mat = confusion_matrix(y_test, knn.predict(X_test))

    print('\n\nConfusion Matrix : \n', con_mat)
    print('True Negative : ', con_mat[0][0])
    print('False Positive : ', con_mat[0][1])
    print('True Positive : ', con_mat[1][1])
    print('False Negative : ', con_mat[1][0])
```

In [52]:
```
prettytable_data = []
```

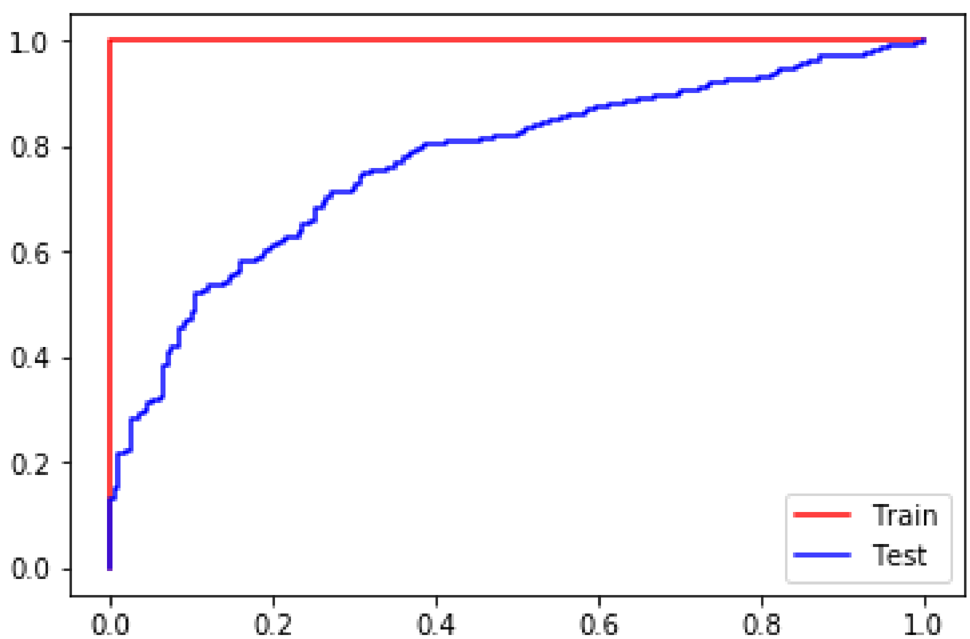
```
In [53]: # Please write all the code with proper documentation
         path = 'saved_models/grid_search_cv_bow.pkl'
         cresults, best_hyperparameters, best_score = perform_grid_search_cv(X=final_bigram_co
                                                                  Y=final['Score'].values,
                                                                  algorithm='brute',
                                                                  model_path=path)


         #PrettyTable Data Collection
         prettytable_data.append(['BOW', best_hyperparameters['algorithm'], best_hyperparamete

         #Analysing the results
         analyse_results(df=cresults)

         #Using the best combination of hyper parameters to retrain the model and plot ROC Cur
         retrain_with_best_hyperparameters(X=final_bigram_counts,
                                           Y=final['Score'].values,
                                           best_params_=best_hyperparameters)
```

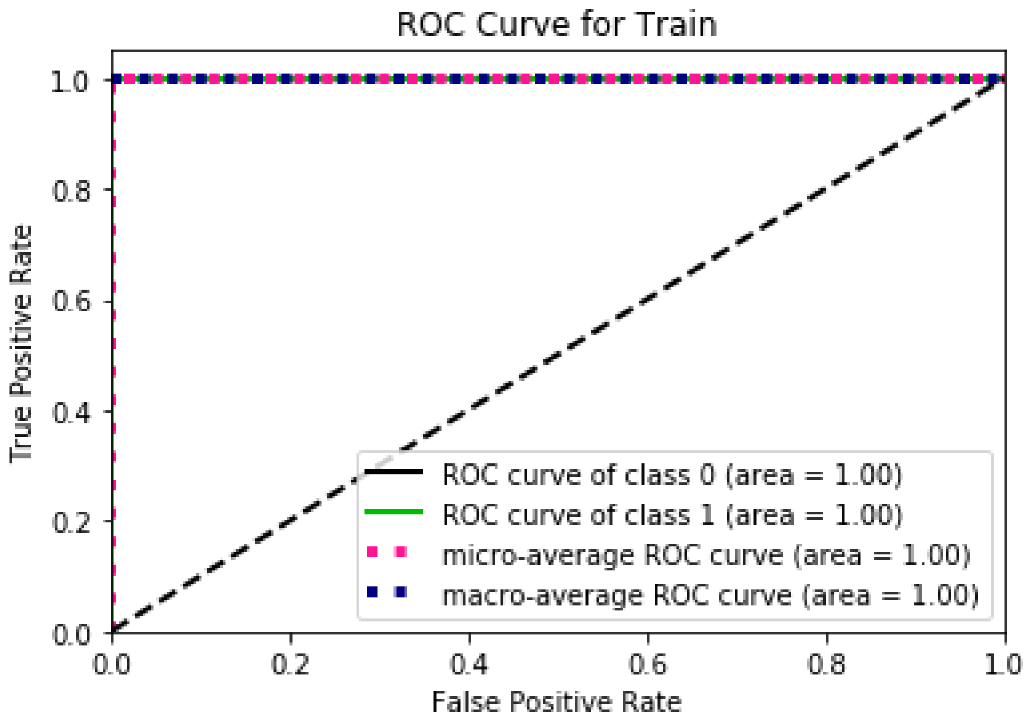
```
Best Parameters : {'algorithm': 'brute', 'n_neighbors': 176, 'weights': 'distance'} that led to
```

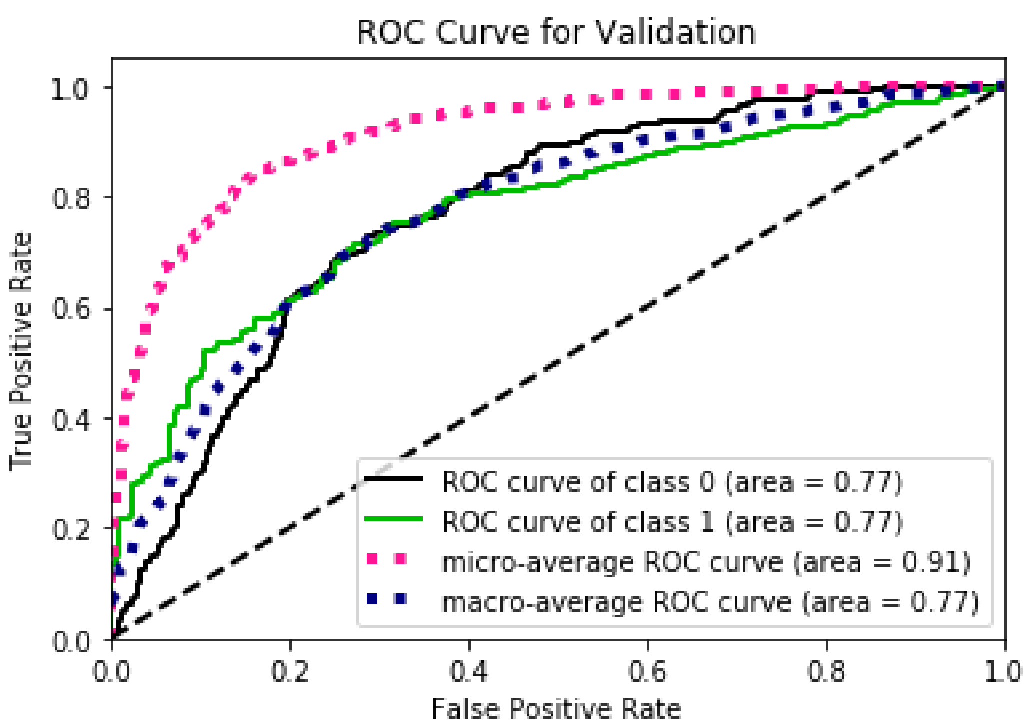





```
Area Under the Curve :  0.7708197475929264
```




ROC Curve for Train
True Positive Rate
False Positive Rate
0.0
0.2
0.4
0.6
0.8
1.0
ROC curve of class 0 (area = 1.00)
ROC curve of class 1 (area = 1.00)
micro-average ROC curve (area = 1.00)
macro-average ROC curve (area = 1.00)

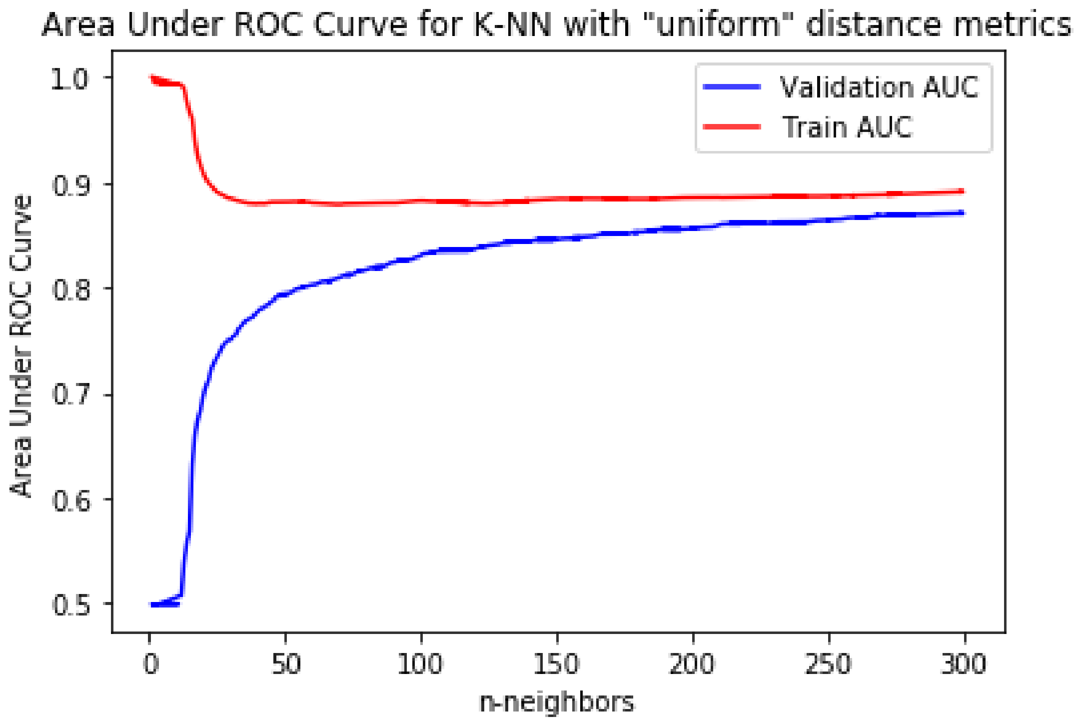

ROC Curve for Validation
True Positive Rate
False Positive Rate
0.0
0.2
0.4
0.6
0.8
1.0
ROC curve of class 0 (area = 0.77)
ROC curve of class 1 (area = 0.77)
micro-average ROC curve (area = 0.91)
macro-average ROC curve (area = 0.77)

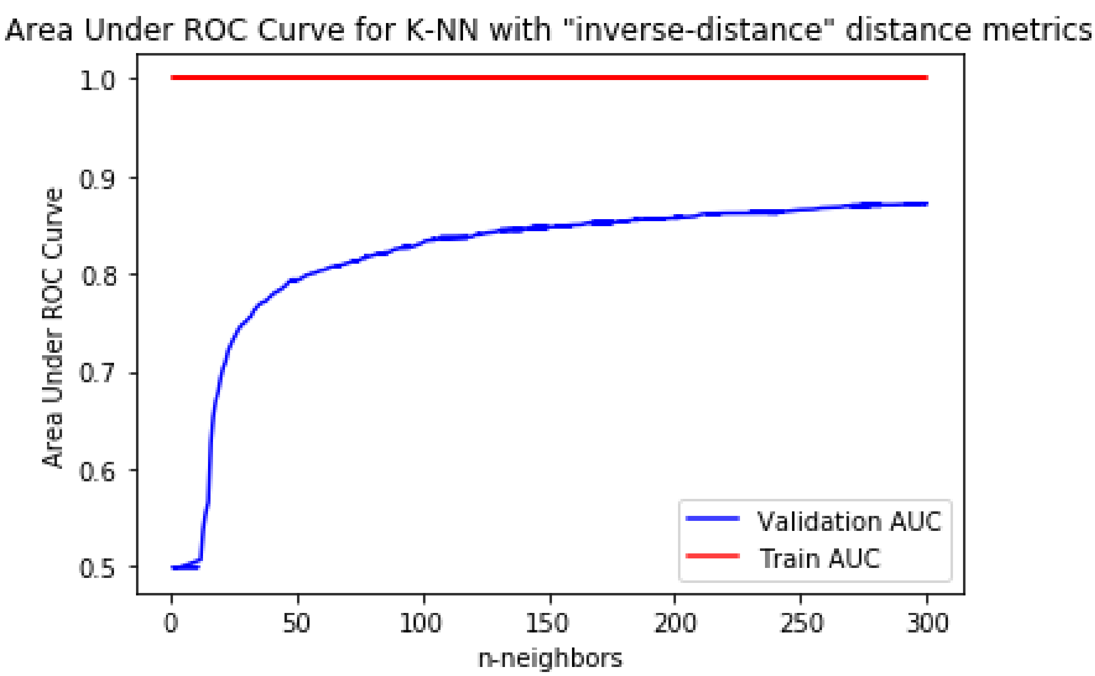
```
Confusion Matrix :
 [[   0  199]
 [   0 1048]]
True Negative :  0
False Positive :  199
True Positive :  1048
False Negative :  0
```

6.1.2 [5.1.2] Applying KNN brute force on TFIDF, SET 2

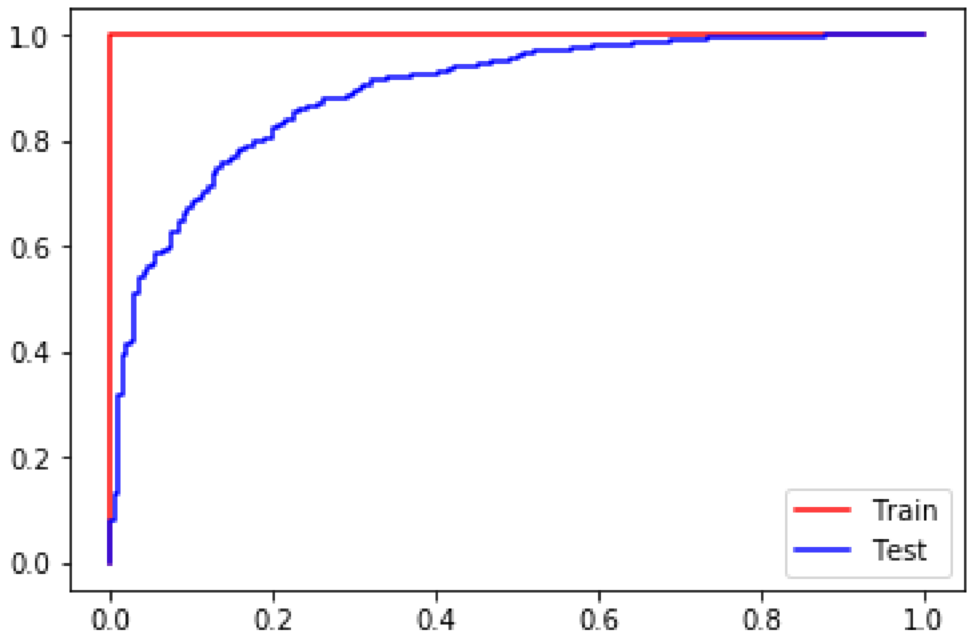
In [54]:
```
# Please write all the code with proper documentation
path = 'saved_models/grid_search_cv_tfidf.pkl'
cresults, best_hyperparameters, best_score = perform_grid_search_cv(X=final_tf_idf,
                                                                    Y=final['Score'].values,
                                                                    algorithm='brute',
                                                                    model_path=path)

#PrettyTable Data Collection
prettytable_data.append(['TF-IDF', best_hyperparameters['algorithm'], best_hyperparame

#analysing the dataframe
analyse_results(cresults)

#Using the best combination of hyper parameters to retrain the model and plot ROC Cur
retrain_with_best_hyperparameters(X=final_tf_idf,
                                  Y=final['Score'].values,
                                  best_params_=best_hyperparameters)
```

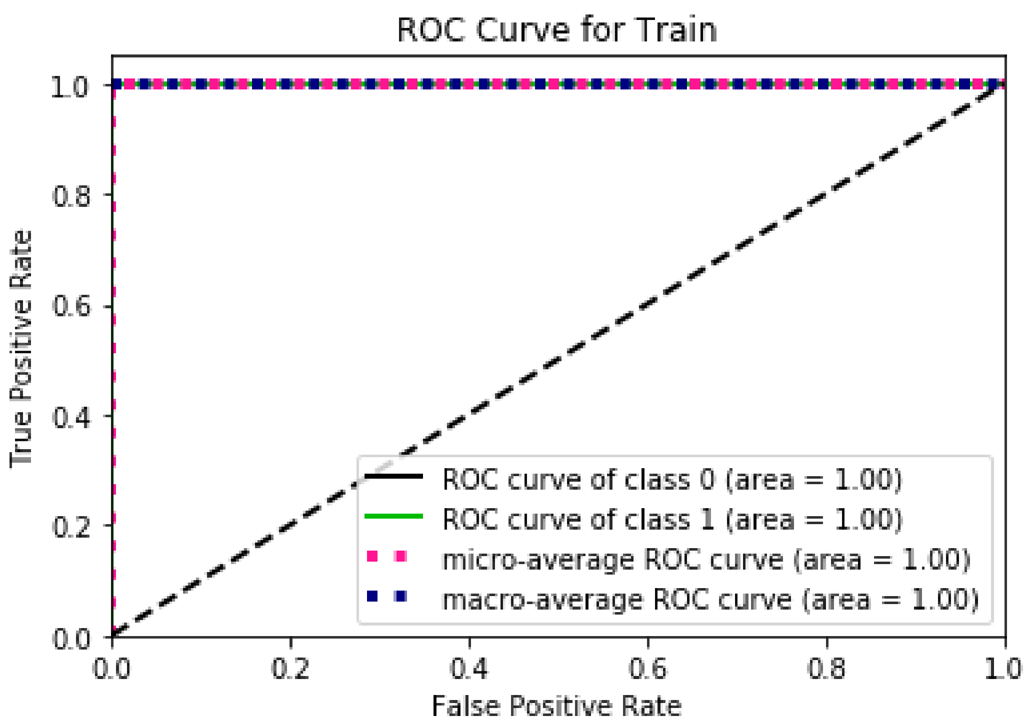
```
Best Parameters : {'algorithm': 'brute', 'n_neighbors': 299, 'weights': 'distance'} that led to
```

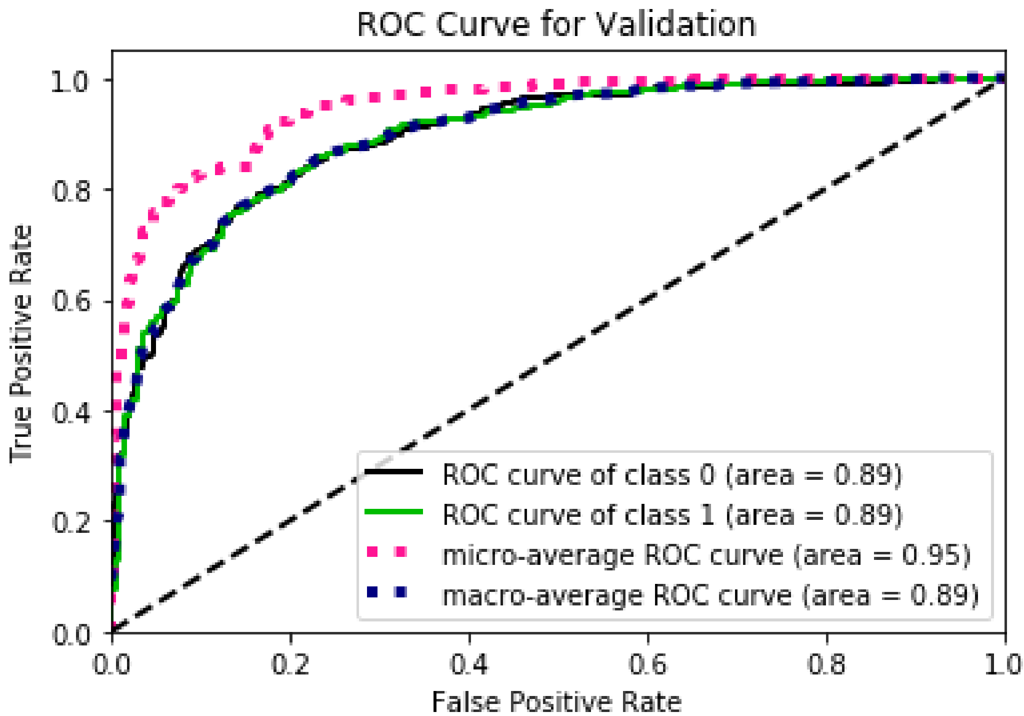





```
Area Under the Curve :  0.8929619471402815
```


Train
Test
0.0
0.2
0.4
0.6
0.8
1.0


ROC Curve for Train
True Positive Rate
False Positive Rate
ROC curve of class 0 (area = 1.00)
ROC curve of class 1 (area = 1.00)
micro-average ROC curve (area = 1.00)
macro-average ROC curve (area = 1.00)



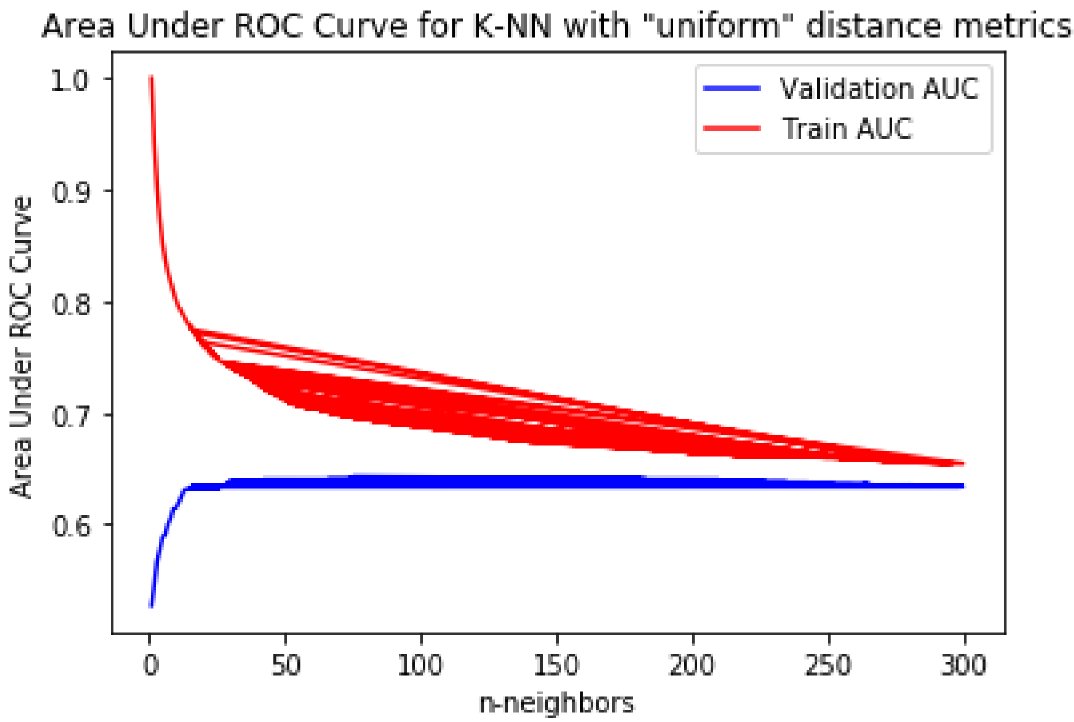

```
Confusion Matrix :
 [[   0  199]
 [   0 1048]]
True Negative :  0
False Positive :  199
True Positive :  1048
False Negative :  0
```

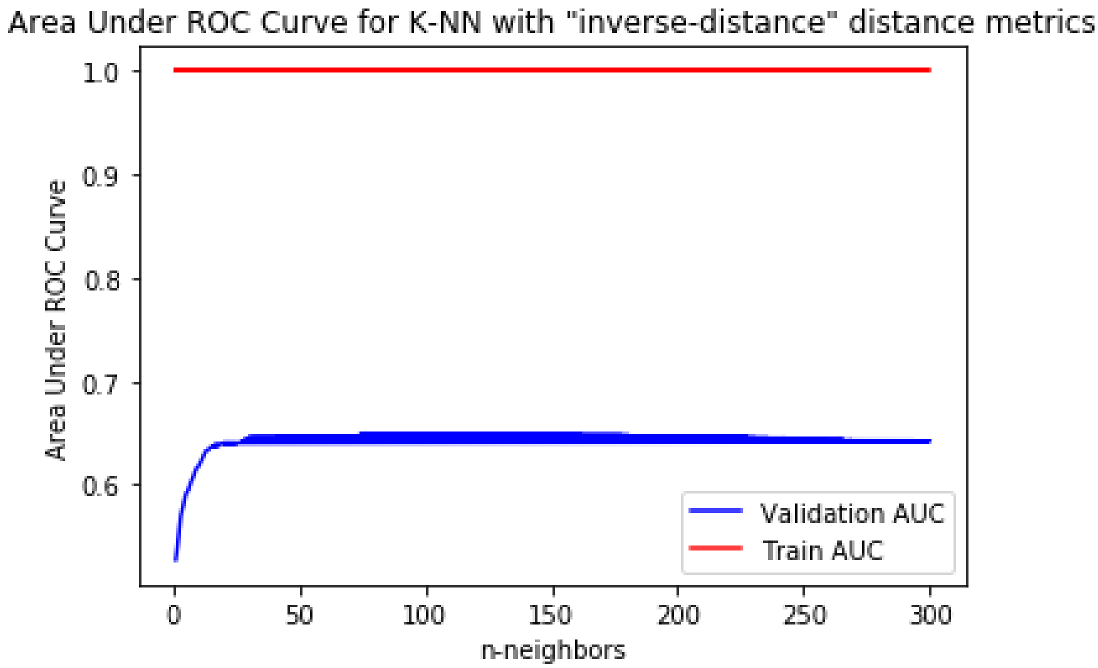
6.1.3 [5.1.3] Applying KNN brute force on AVG W2V, SET 3

In [55]:
```
# Please write all the code with proper documentation
path = 'saved_models/grid_search_cv_avg_w2v.pkl'
cresults, best_hyperparameters, best_score = perform_grid_search_cv(X=sent_vectors,
                                                                    Y=final['Score'].values,
                                                                    algorithm='brute',
                                                                    model_path=path)

#PrettyTable Data Collection
prettytable_data.append(['Avg. W2V', best_hyperparameters['algorithm'], best_hyperpara

#analysing the dataframe
```

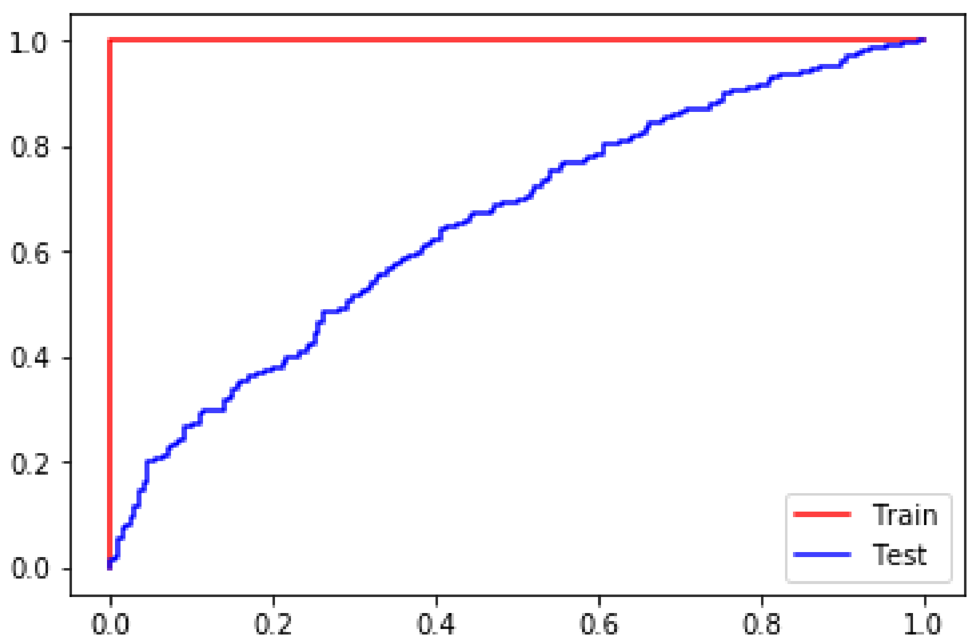
```python
analyse_results(cresults)

#Using the best combination of hyper parameters to retrain the model and plot ROC Cur
retrain_with_best_hyperparameters(X=sent_vectors,
                                  Y=final['Score'].values,
                                  best_params_=best_hyperparameters)
```

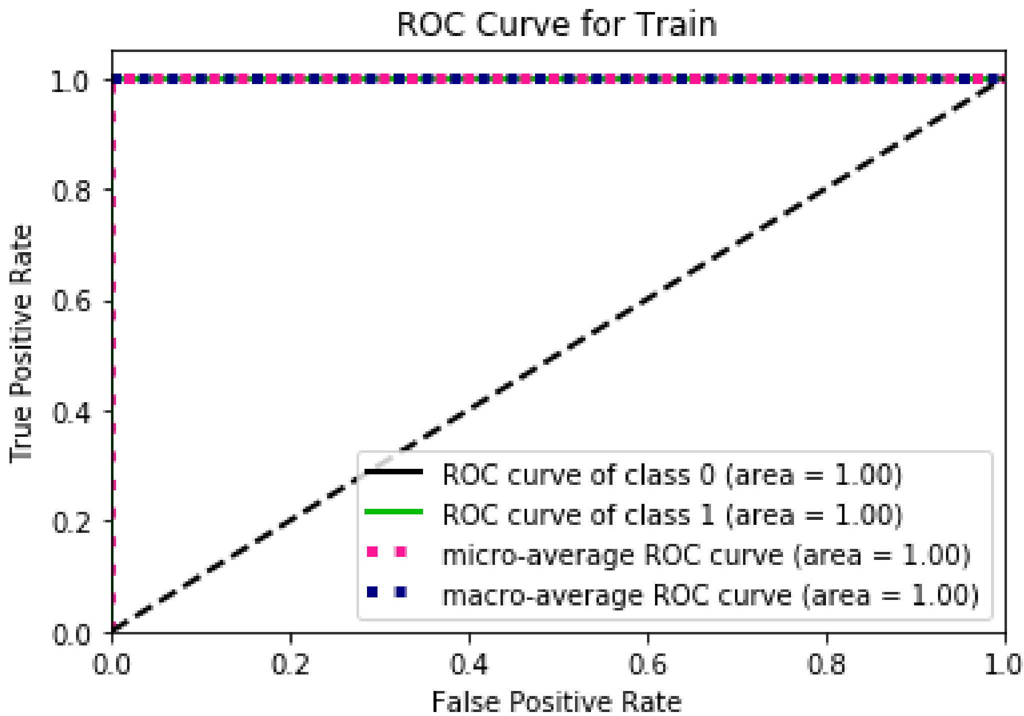
```
Best Parameters : {'algorithm': 'brute', 'n_neighbors': 76, 'weights': 'distance'} that led to
```

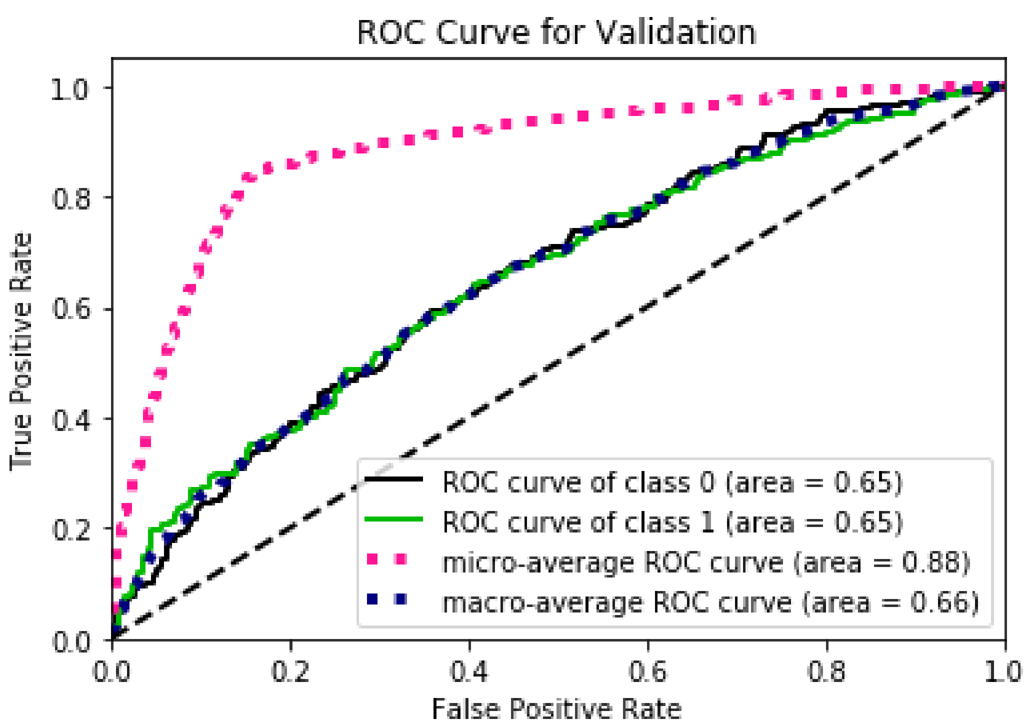





```
Area Under the Curve :  0.6548103110974721
```




ROC Curve for Train
True Positive Rate
False Positive Rate
ROC curve of class 0 (area = 1.00)
ROC curve of class 1 (area = 1.00)
micro-average ROC curve (area = 1.00)
macro-average ROC curve (area = 1.00)
0.0
0.2
0.4
0.6
0.8
1.0

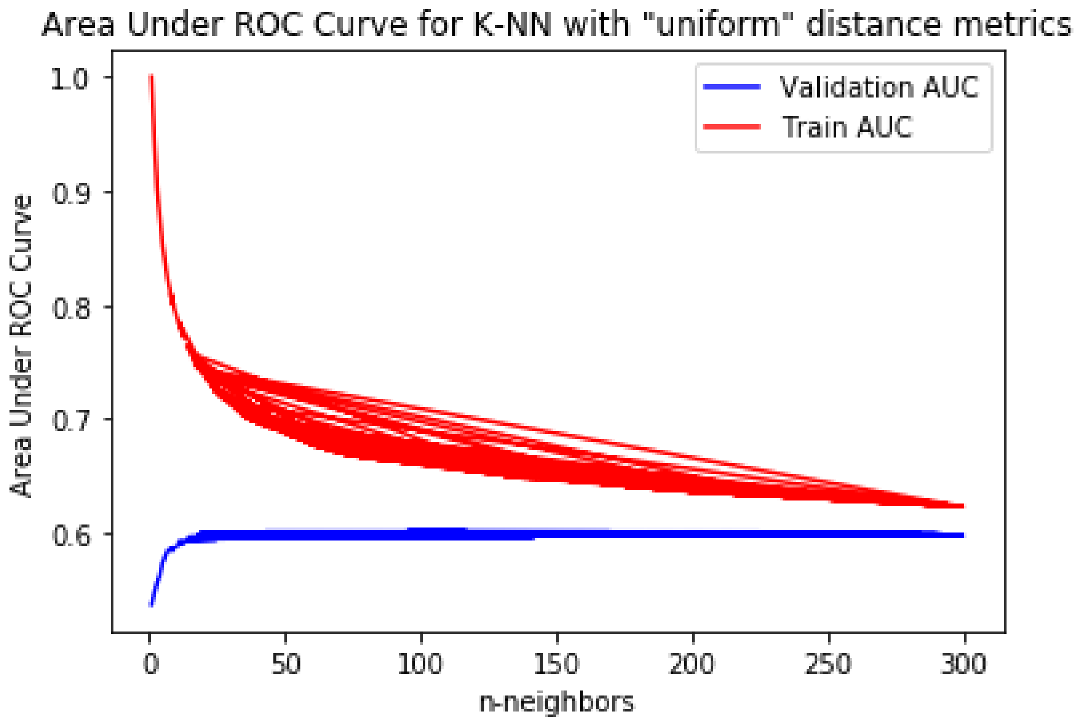

ROC Curve for Validation
True Positive Rate
False Positive Rate
ROC curve of class 0 (area = 0.65)
ROC curve of class 1 (area = 0.65)
micro-average ROC curve (area = 0.88)
macro-average ROC curve (area = 0.66)
0.0
0.2
0.4
0.6
0.8
1.0

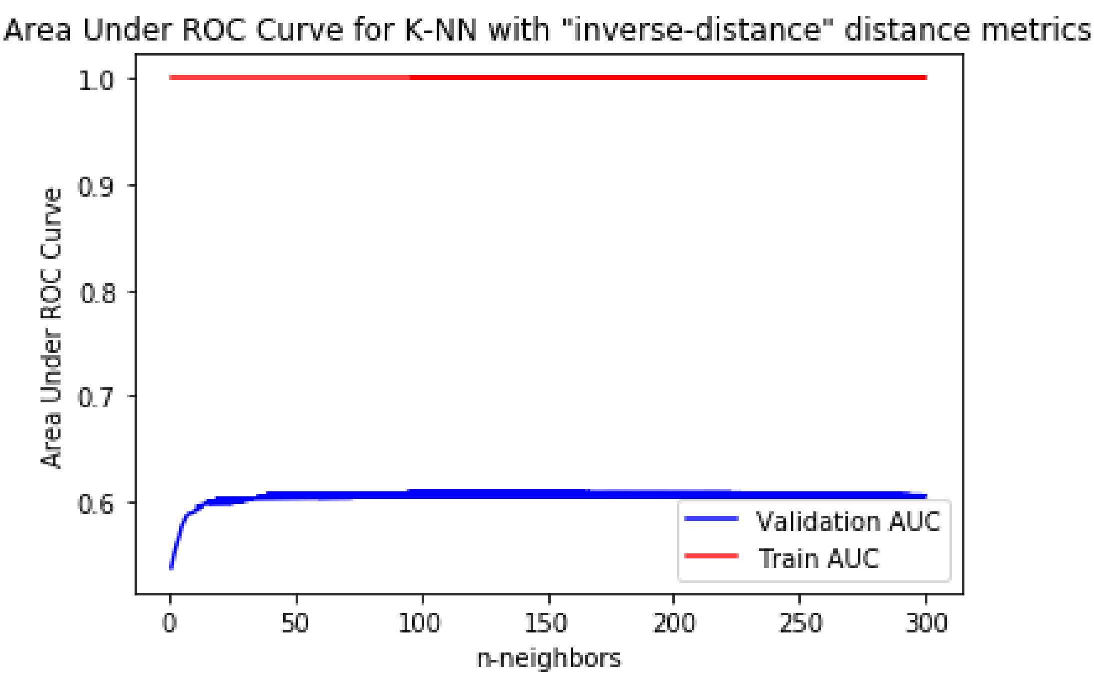
```
Confusion Matrix :
 [[   0  199]
 [   0 1048]]
True Negative :  0
False Positive :  199
True Positive :  1048
False Negative :  0
```

6.1.4 [5.1.4] Applying KNN brute force on TFIDF W2V, SET 4

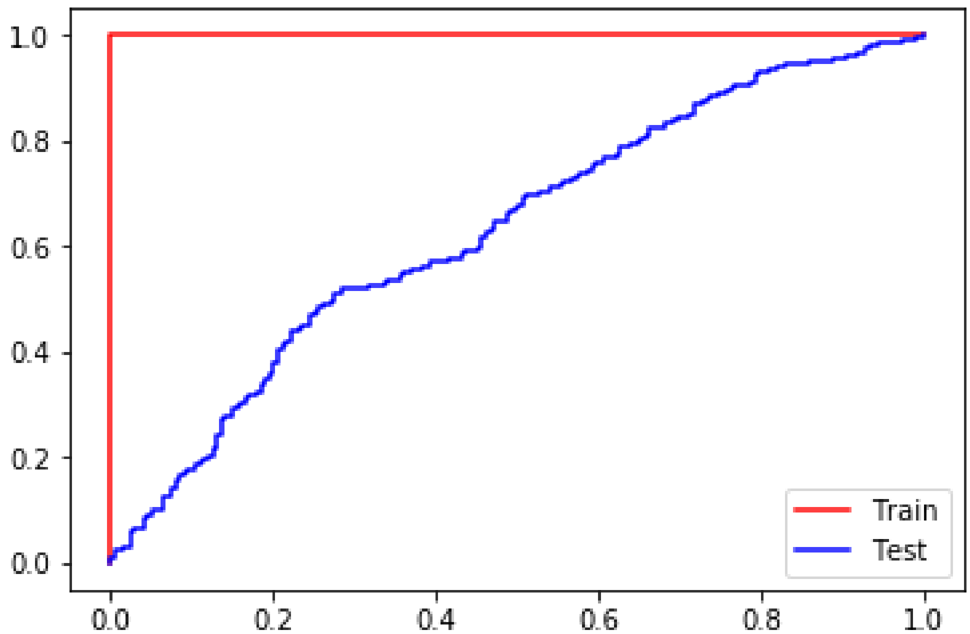
In [56]:
```
# Please write all the code with proper documentation
path = 'saved_models/grid_search_cv_tf_idf_w2v.pkl'
cresults, best_hyperparameters, best_score = perform_grid_search_cv(X=tfidf_sent_vecto
                                                                      Y=final['Score'].values,
                                                                      algorithm='brute',
                                                                      model_path=path)


#PrettyTable Data Collection
prettytable_data.append(['TF-IDF. W2V', best_hyperparameters['algorithm'], best_hyper

#analysing the dataframe
analyse_results(cresults)

#Using the best combination of hyper parameters to retrain the model and plot ROC Cur
retrain_with_best_hyperparameters(X=tfidf_sent_vectors,
                                  Y=final['Score'].values,
                                  best_params_=best_hyperparameters)
```

```
Best Parameters : {'algorithm': 'brute', 'n_neighbors': 96, 'weights': 'distance'} that led to
```

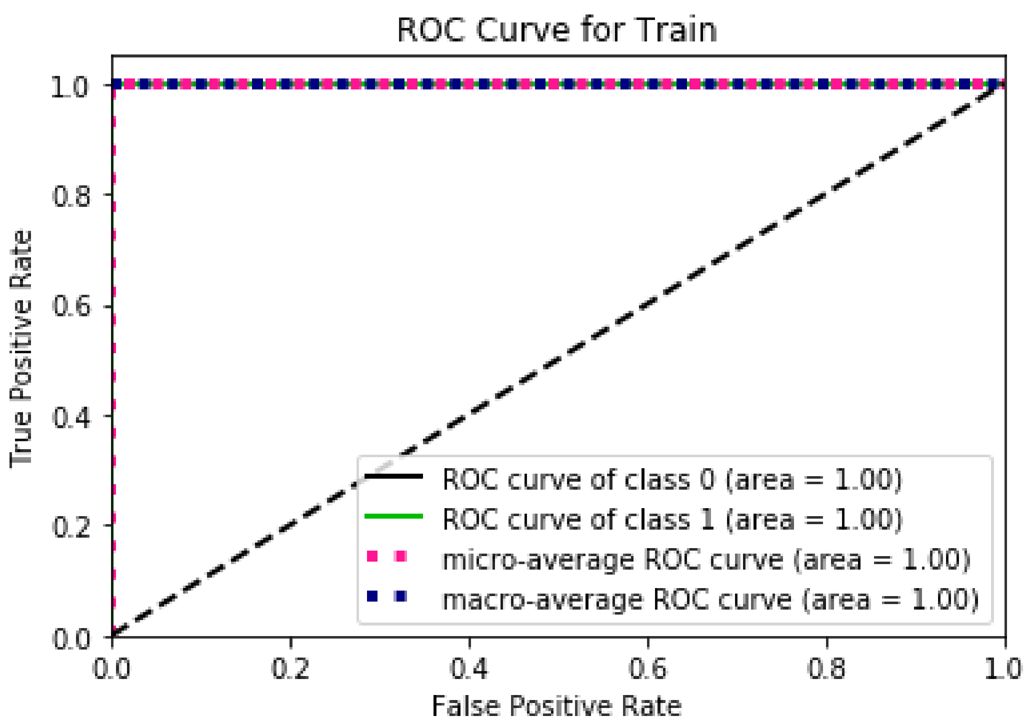




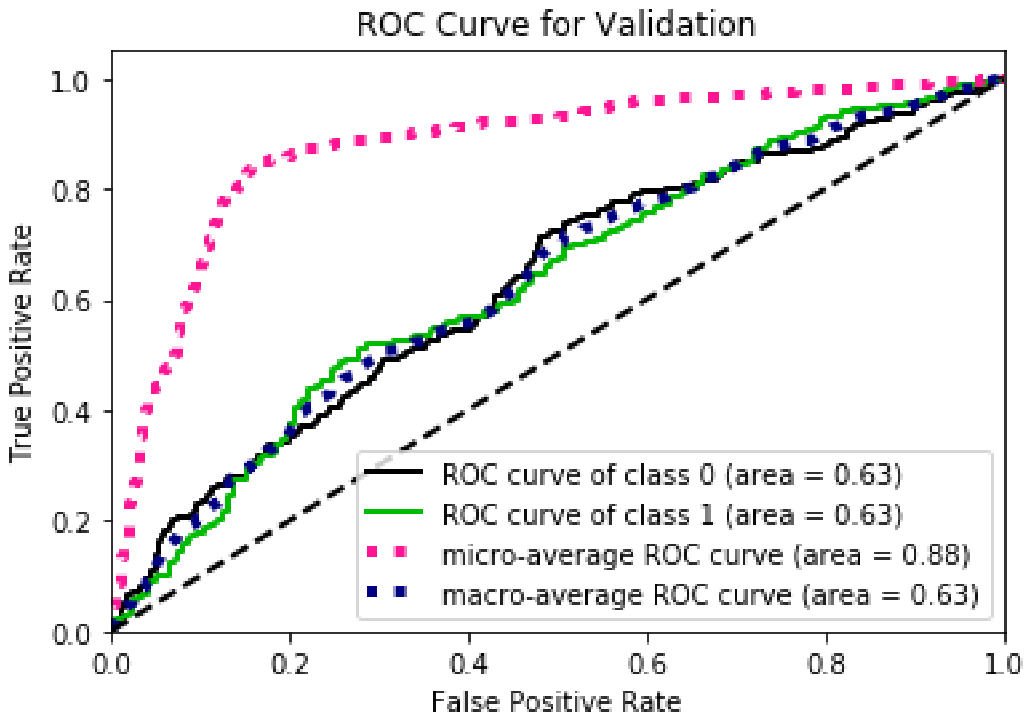

```
Area Under the Curve :  0.6302936437914766
```



1.0
0.8
0.6
0.4
0.2
0.0
0.0
0.2
0.4
0.6
0.8
1.0
Train
Test



ROC Curve for Train
True Positive Rate
False Positive Rate
1.0
0.8
0.6
0.4
0.2
0.0
0.0
0.2
0.4
0.6
0.8
1.0
ROC curve of class 0 (area = 1.00)
ROC curve of class 1 (area = 1.00)
micro-average ROC curve (area = 1.00)
macro-average ROC curve (area = 1.00)




```
Confusion Matrix :
 [[   0  199]
 [   0 1048]]
True Negative :  0
False Positive :  199
True Positive :  1048
False Negative :  0
```

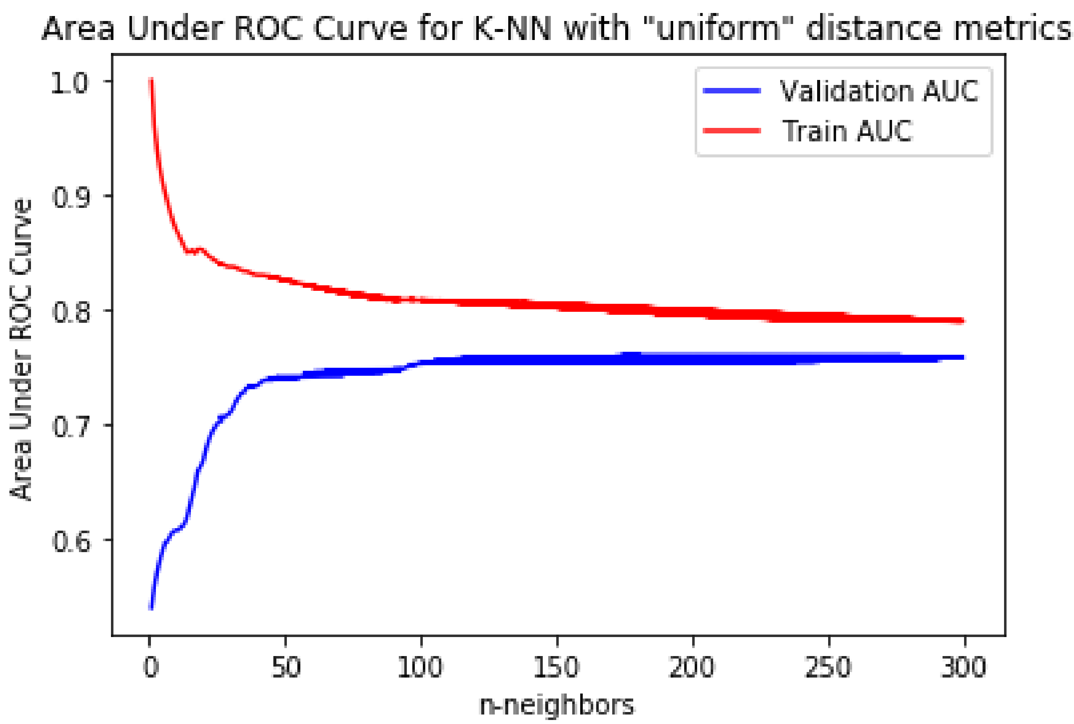
6.2 [5.2] Applying KNN kd-tree

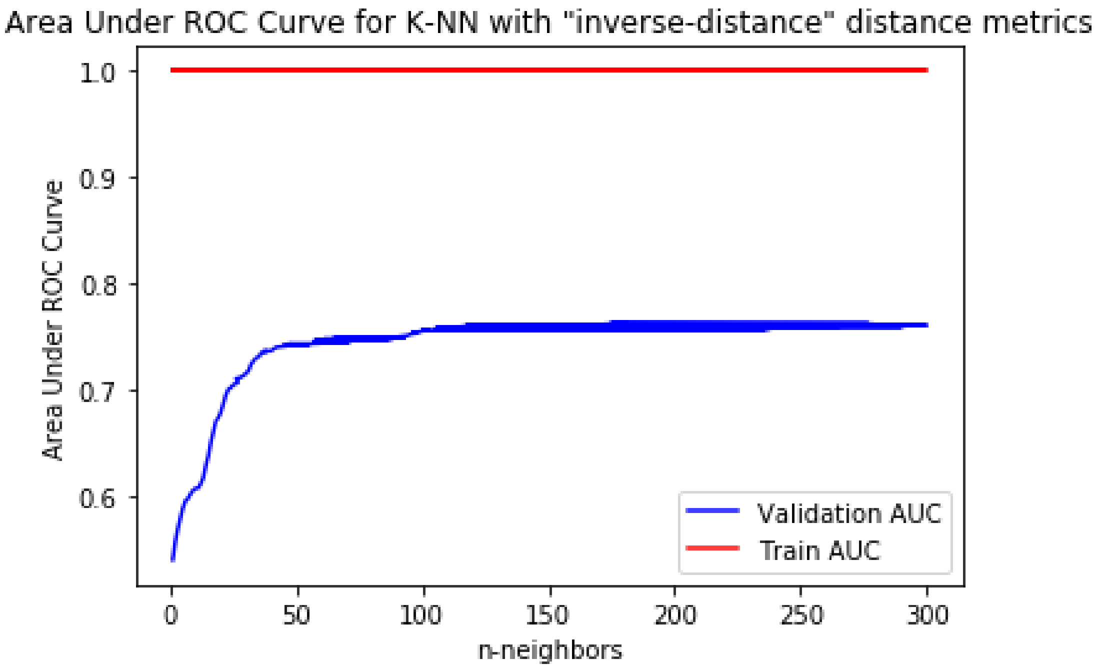
6.2.1 [5.2.1] Applying KNN kd-tree on BOW, SET 5

In [57]:
```
# Please write all the code with proper documentation
path = 'saved_models/grid_search_cv_bow_kd.pkl'
cresults, best_hyperparameters, best_score = perform_grid_search_cv(X=final_bigram_cou
                                                                    Y=final['Score'].values,
                                                                    algorithm='kd_tree',
                                                                    model_path=path)
#PrettyTable Data Collection
prettytable_data.append(['BOW-kd', best_hyperparameters['algorithm'], best_hyperparame
```

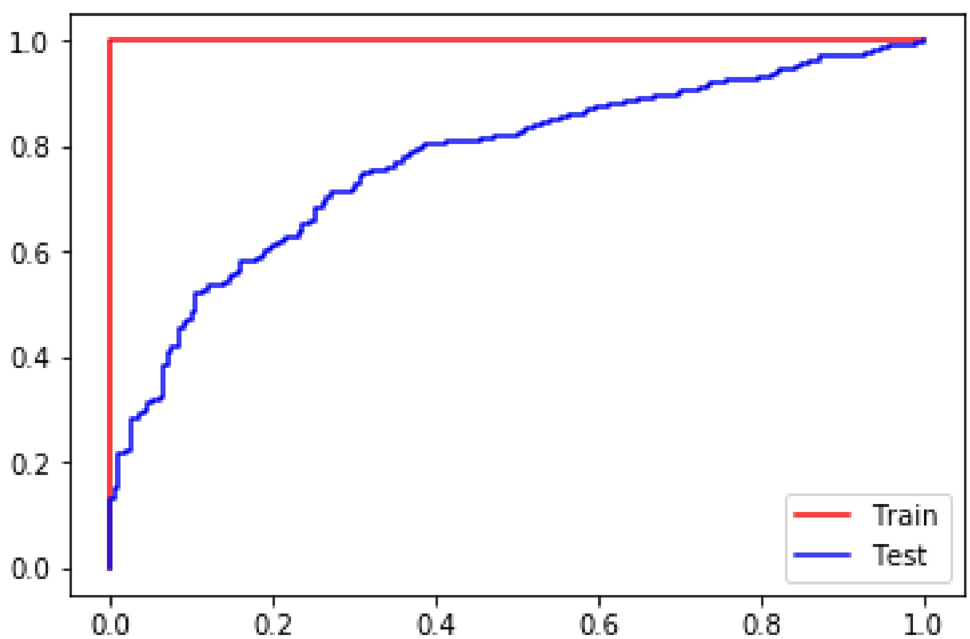
```
#Analysing the results
analyse_results(df=cresults)

#Using the best combination of hyper parameters to retrain the model and plot ROC Cur
retrain_with_best_hyperparameters(X=final_bigram_counts,
                                  Y=final['Score'].values,
                                  best_params_=best_hyperparameters)
```

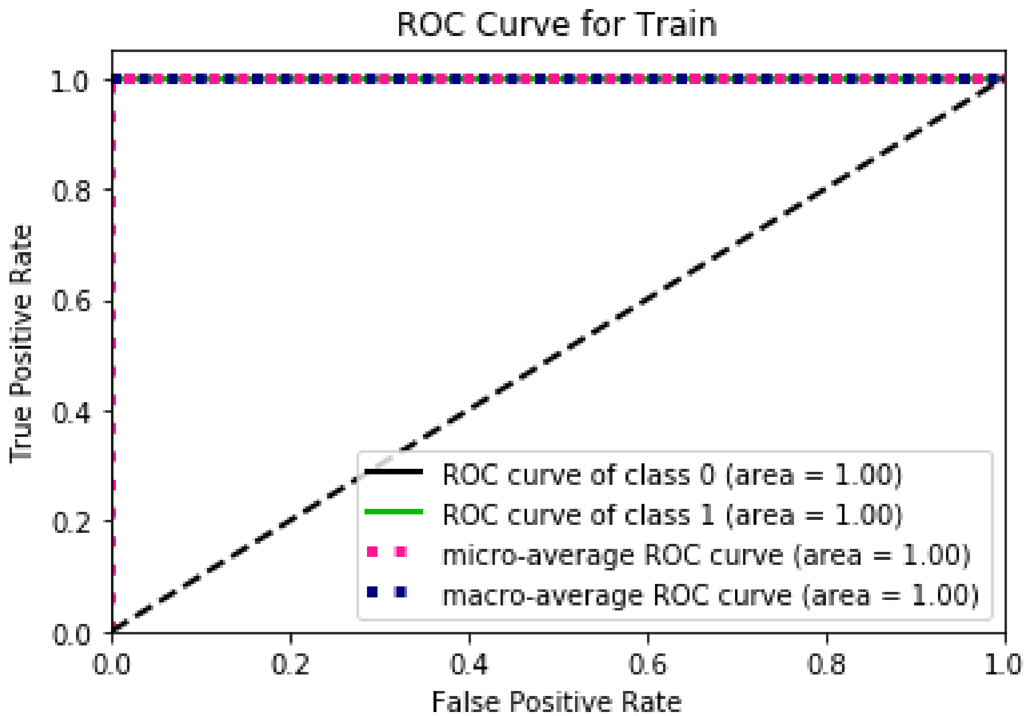
```
Best Parameters : {'algorithm': 'kd_tree', 'n_neighbors': 176, 'weights': 'distance'} that led
```

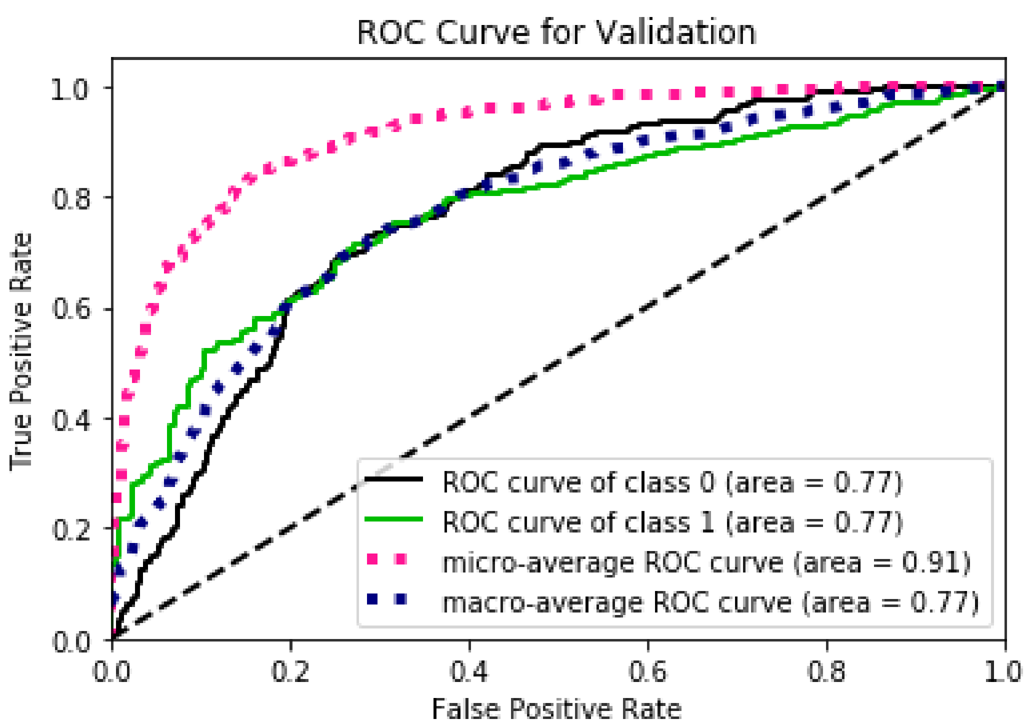





```
Area Under the Curve :  0.7708197475929264
```




ROC Curve for Train
True Positive Rate
False Positive Rate
0.0
0.2
0.4
0.6
0.8
1.0
ROC curve of class 0 (area = 1.00)
ROC curve of class 1 (area = 1.00)
micro-average ROC curve (area = 1.00)
macro-average ROC curve (area = 1.00)

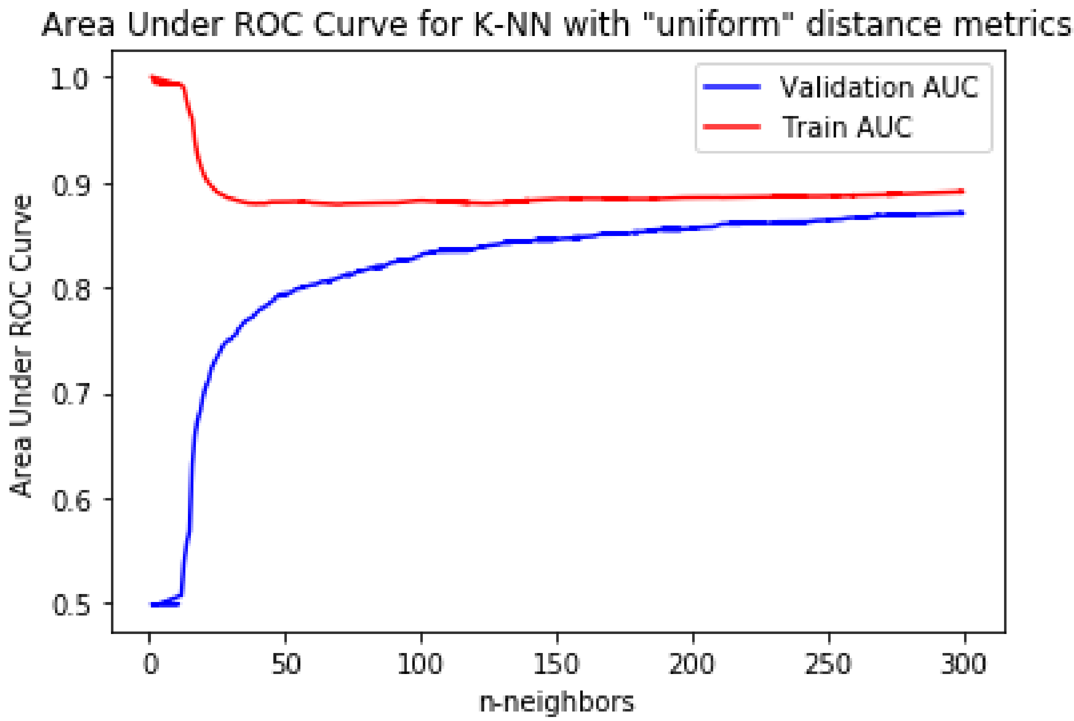

ROC Curve for Validation
True Positive Rate
False Positive Rate
0.0
0.2
0.4
0.6
0.8
1.0
ROC curve of class 0 (area = 0.77)
ROC curve of class 1 (area = 0.77)
micro-average ROC curve (area = 0.91)
macro-average ROC curve (area = 0.77)

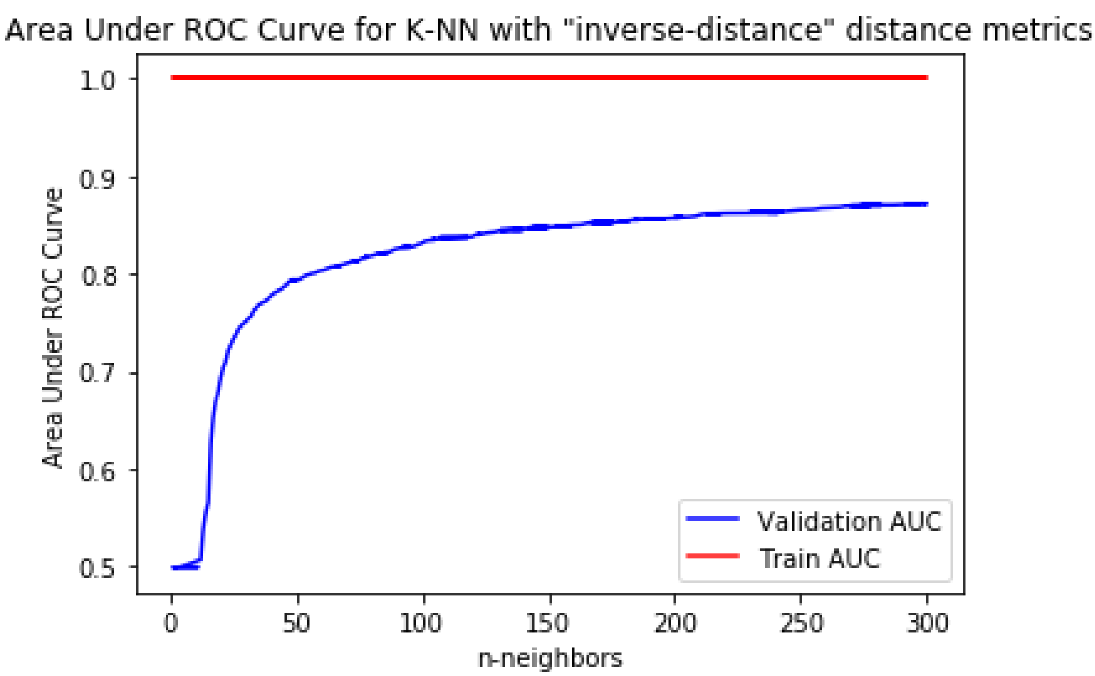
```
Confusion Matrix :
 [[   0  199]
 [   0 1048]]
True Negative :  0
False Positive :  199
True Positive :  1048
False Negative :  0
```

6.2.2 [5.2.2] Applying KNN kd-tree on TFIDF, SET 6

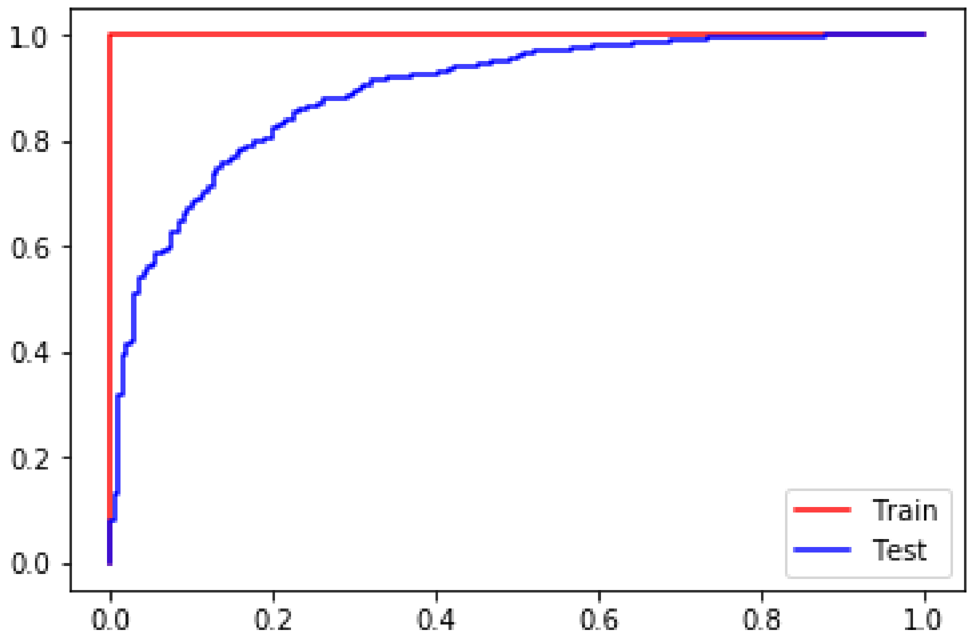
In [58]:
```
# Please write all the code with proper documentation
path = 'saved_models/grid_search_cv_tfidf_kd.pkl'
cresults, best_hyperparameters, best_score = perform_grid_search_cv(X=final_tf_idf,
                                                          Y=final['Score'].values,
                                                          algorithm='kd_tree',
                                                          model_path=path)


#PrettyTable Data Collection
prettytable_data.append(['TF-IDF-kd', best_hyperparameters['algorithm'], best_hyperpa


#analysing the dataframe
analyse_results(cresults)

#Using the best combination of hyper parameters to retrain the model and plot ROC Cur
retrain_with_best_hyperparameters(X=final_tf_idf,
                                  Y=final['Score'].values,
                                  best_params_=best_hyperparameters)
```

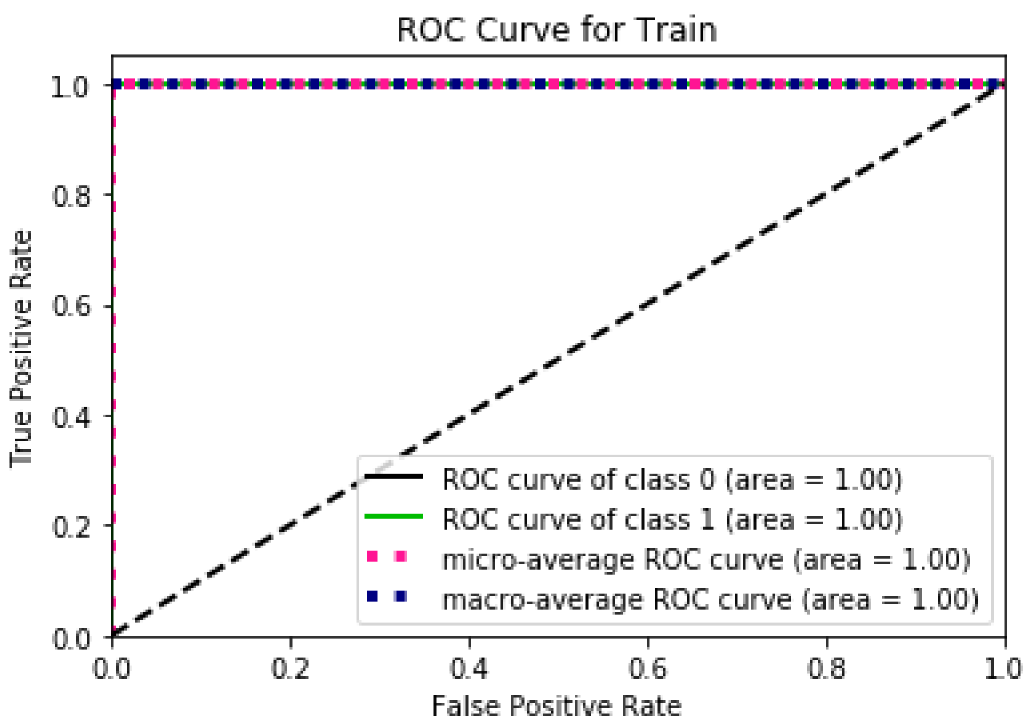
```
Best Parameters : {'algorithm': 'kd_tree', 'n_neighbors': 299, 'weights': 'distance'} that led
```

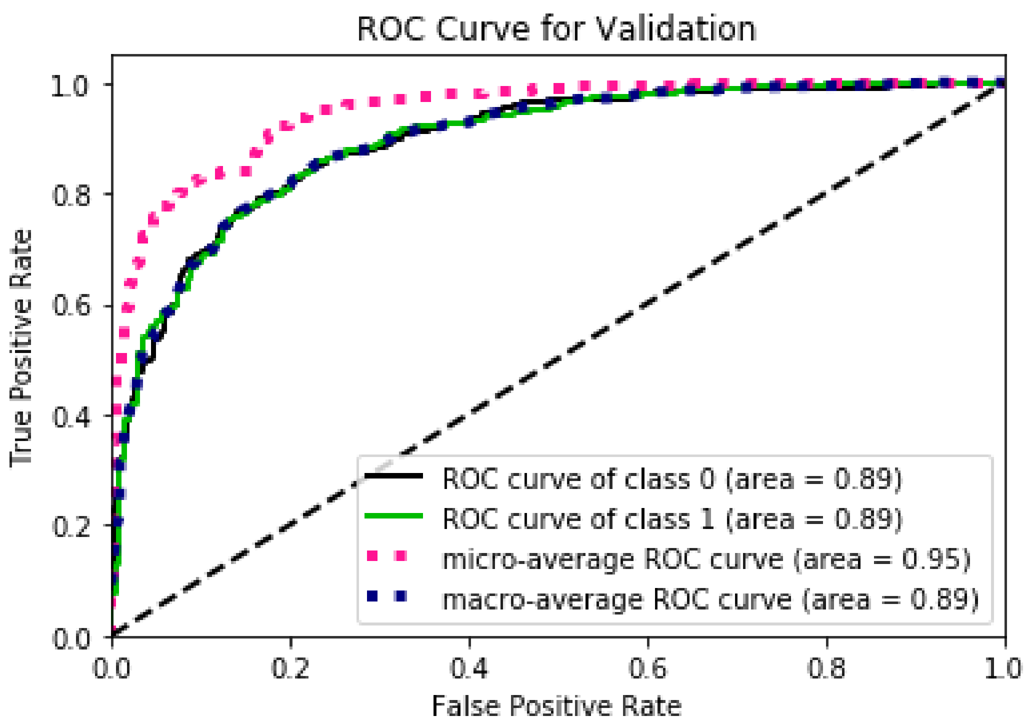





```
Area Under the Curve :  0.8929619471402815
```


1.0
0.8
0.6
0.4
0.2
0.0
0.0
0.2
0.4
0.6
0.8
1.0
Train
Test


ROC Curve for Train
True Positive Rate
False Positive Rate
1.0
0.8
0.6
0.4
0.2
0.0
0.0
0.2
0.4
0.6
0.8
1.0
ROC curve of class 0 (area = 1.00)
ROC curve of class 1 (area = 1.00)
micro-average ROC curve (area = 1.00)
macro-average ROC curve (area = 1.00)



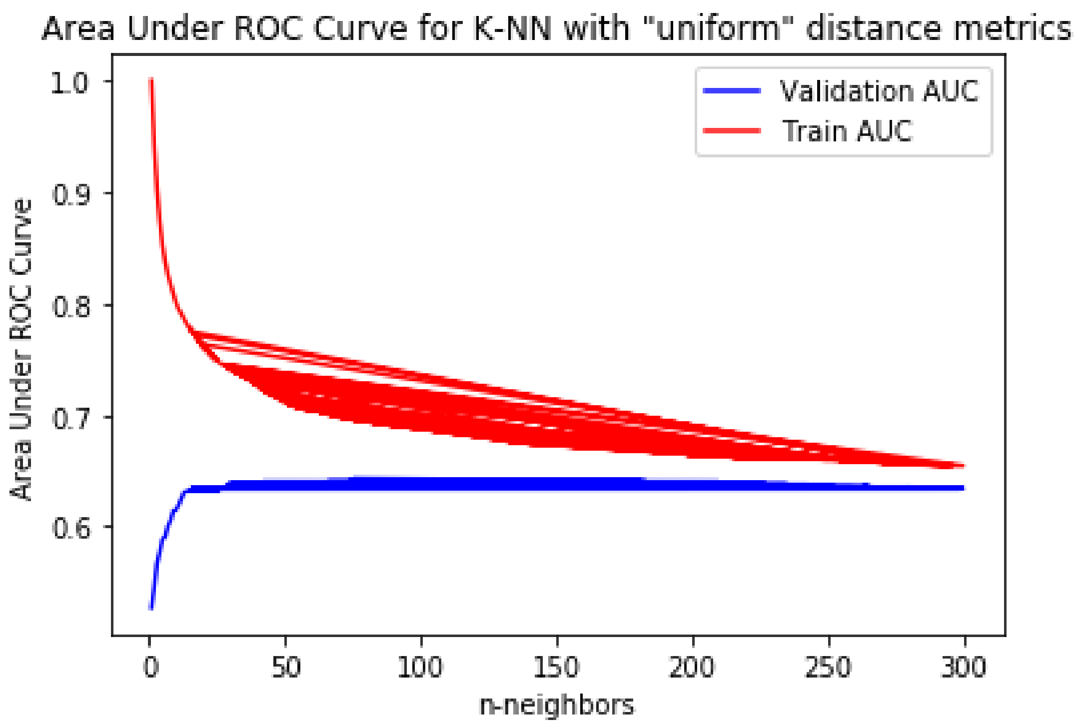

```
Confusion Matrix :
 [[   0  199]
 [   0 1048]]
True Negative :  0
False Positive :  199
True Positive :  1048
False Negative :  0
```

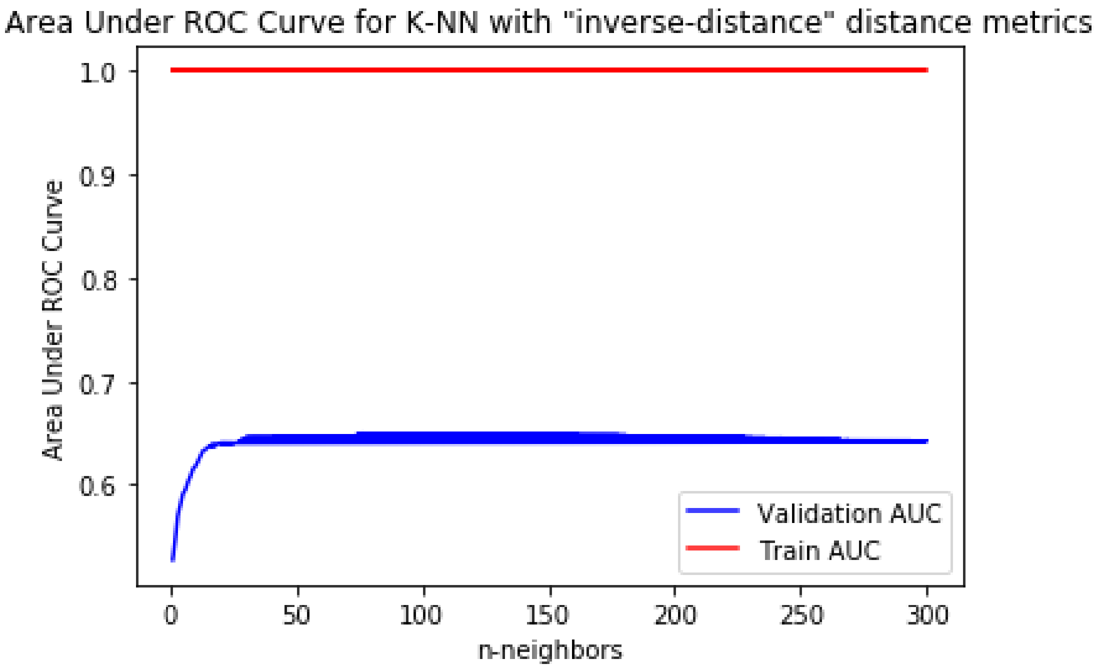
6.2.3 [5.2.3] Applying KNN kd-tree on AVG W2V, SET 3

In [59]:
```
# Please write all the code with proper documentation
# Please write all the code with proper documentation
path = 'saved_models/grid_search_cv_avg_w2v_kd.pkl'
cresults, best_hyperparameters, best_score = perform_grid_search_cv(X=sent_vectors,
                                                                    Y=final['Score'].values,
                                                                    algorithm='kd_tree',
                                                                    model_path=path)


#PrettyTable Data Collection
prettytable_data.append(['Avg. W2V-kd', best_hyperparameters['algorithm'], best_hyper
```

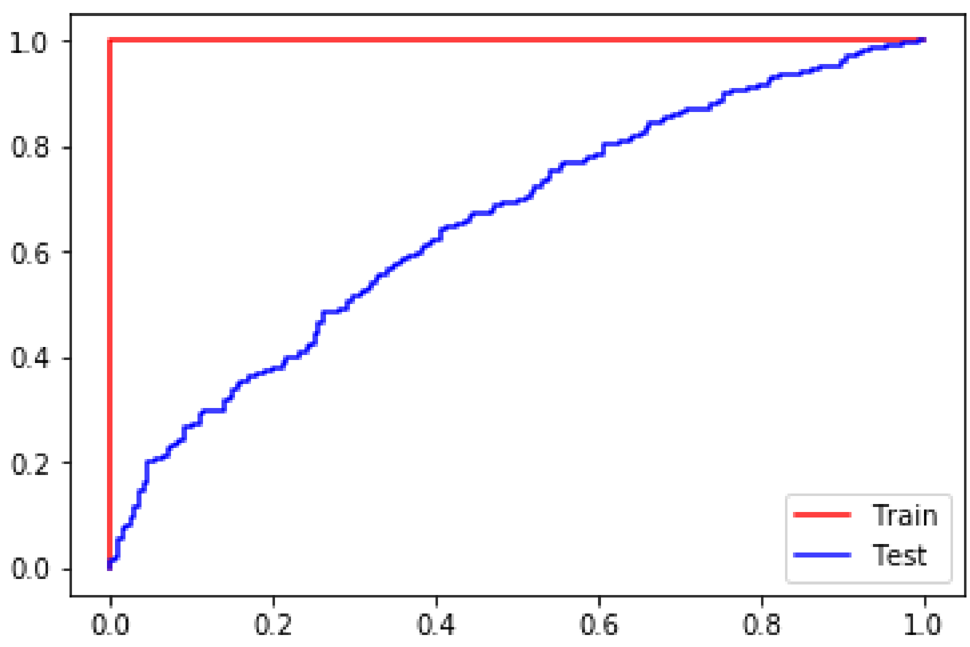
```
#analysing the dataframe
analyse_results(cresults)

#Using the best combination of hyper parameters to retrain the model and plot ROC Cur
retrain_with_best_hyperparameters(X=sent_vectors,
                                  Y=final['Score'].values,
                                  best_params_=best_hyperparameters)
```

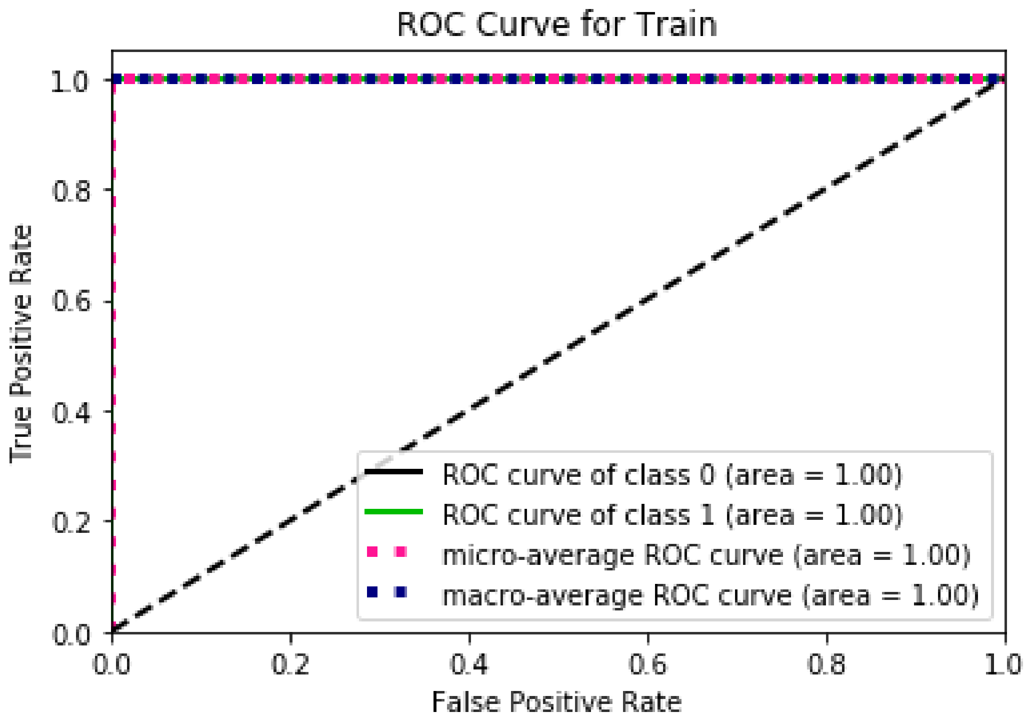
```
Best Parameters : {'algorithm': 'kd_tree', 'n_neighbors': 76, 'weights': 'distance'} that led t
```

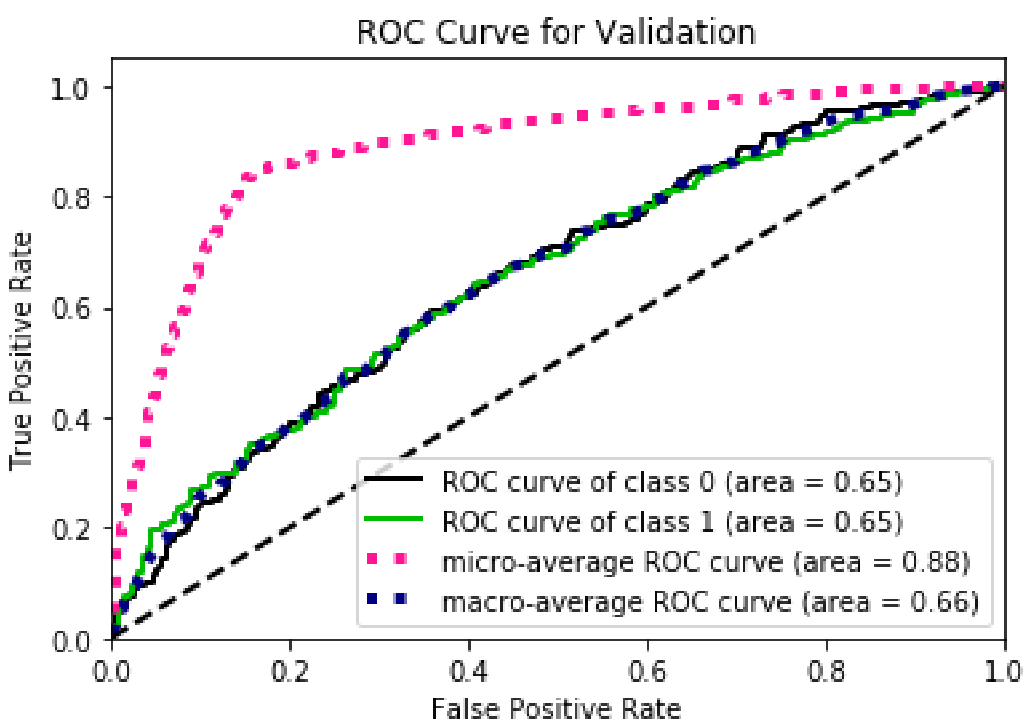





```
Area Under the Curve :  0.6548103110974721
```




ROC Curve for Train
True Positive Rate
False Positive Rate
0.0
0.2
0.4
0.6
0.8
1.0
ROC curve of class 0 (area = 1.00)
ROC curve of class 1 (area = 1.00)
micro-average ROC curve (area = 1.00)
macro-average ROC curve (area = 1.00)

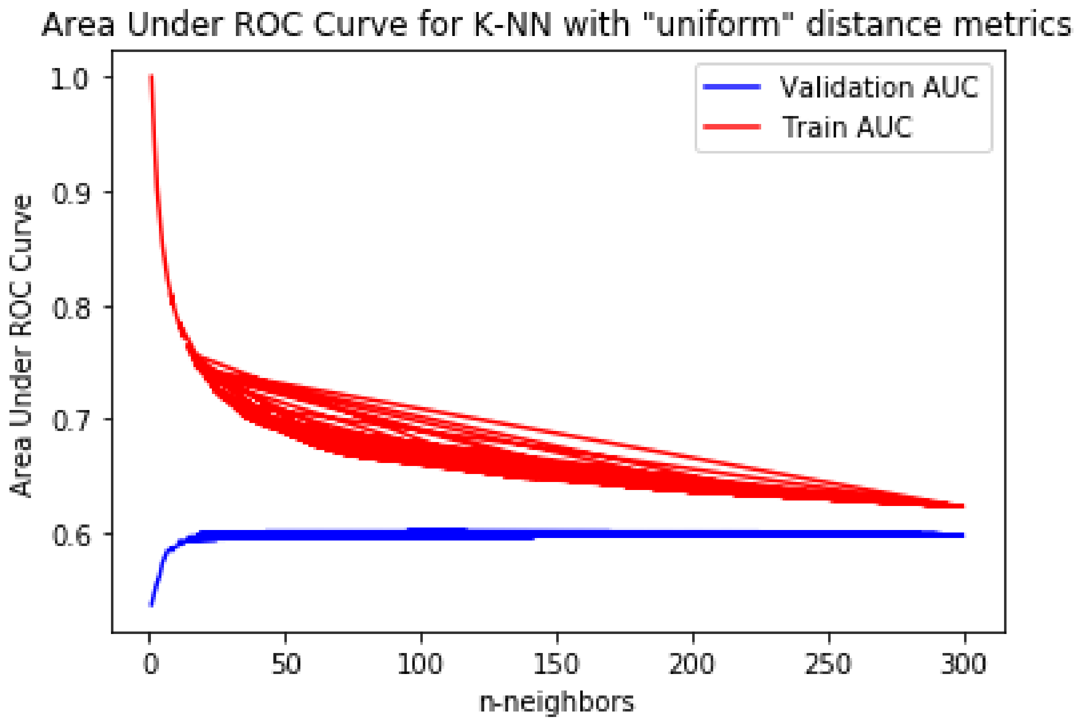

ROC Curve for Validation
True Positive Rate
False Positive Rate
0.0
0.2
0.4
0.6
0.8
1.0
ROC curve of class 0 (area = 0.65)
ROC curve of class 1 (area = 0.65)
micro-average ROC curve (area = 0.88)
macro-average ROC curve (area = 0.66)

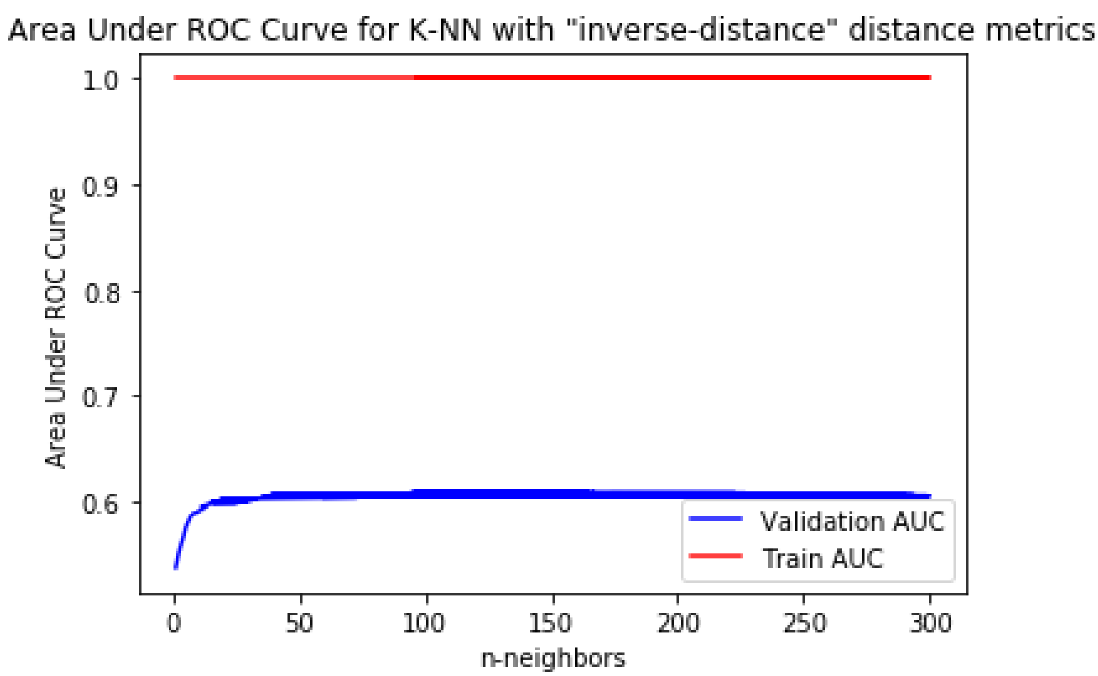
```
Confusion Matrix :
 [[   0  199]
 [   0 1048]]
True Negative :  0
False Positive :  199
True Positive :  1048
False Negative :  0
```

6.2.4 [5.2.4] Applying KNN kd-tree on TFIDF W2V, SET 4

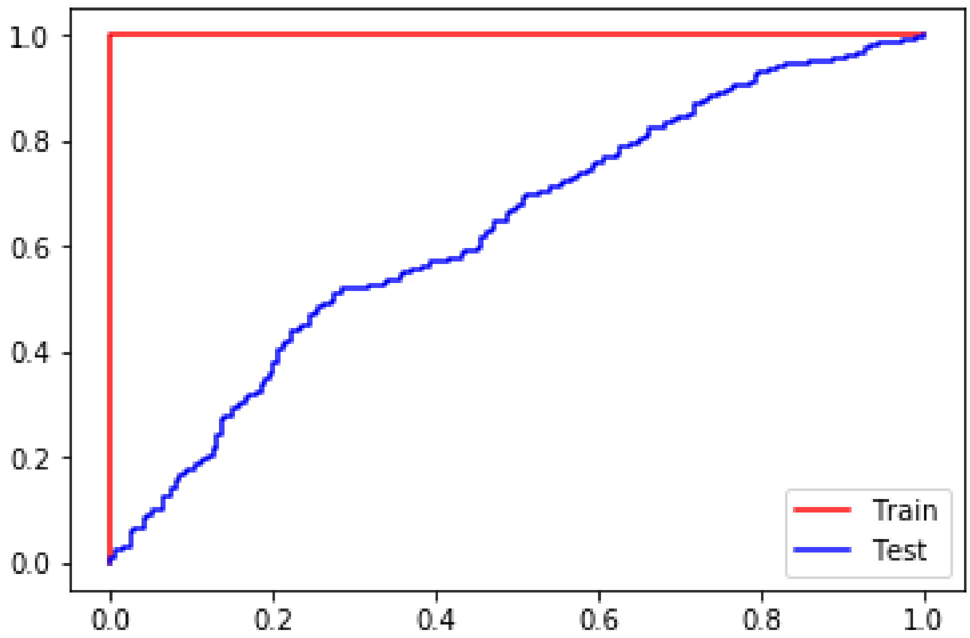
```
In [60]: # Please write all the code with proper documentation
         path = 'saved_models/grid_search_cv_tf_idf_w2v_kd.pkl'
         cresults, best_hyperparameters, best_score = perform_grid_search_cv(X=tfidf_sent_vecto
                                                                            Y=final['Score'].values,
                                                                            algorithm='kd_tree',
                                                                            model_path=path)

         #PrettyTable Data Collection
         prettytable_data.append(['TF-IDF W2V-kd', best_hyperparameters['algorithm'], best_hype

         #analysing the dataframe
         analyse_results(cresults)

         #Using the best combination of hyper parameters to retrain the model and plot ROC Cur
         retrain_with_best_hyperparameters(X=tfidf_sent_vectors,
                                           Y=final['Score'].values,
                                           best_params_=best_hyperparameters)
```

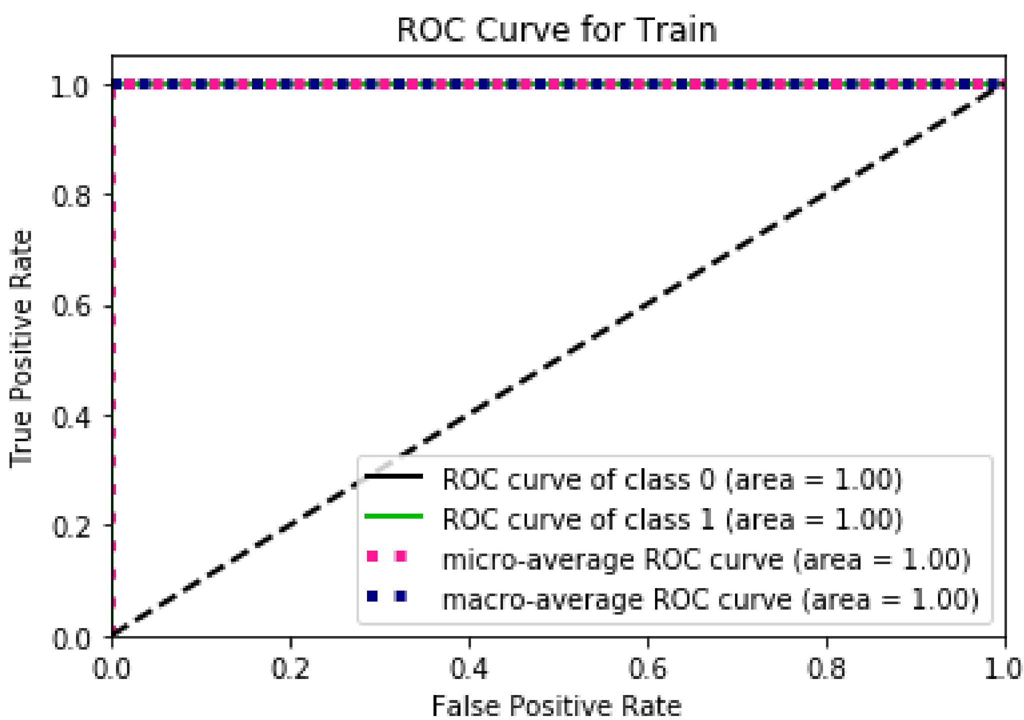
```
Best Parameters : {'algorithm': 'kd_tree', 'n_neighbors': 96, 'weights': 'distance'} that led 
```

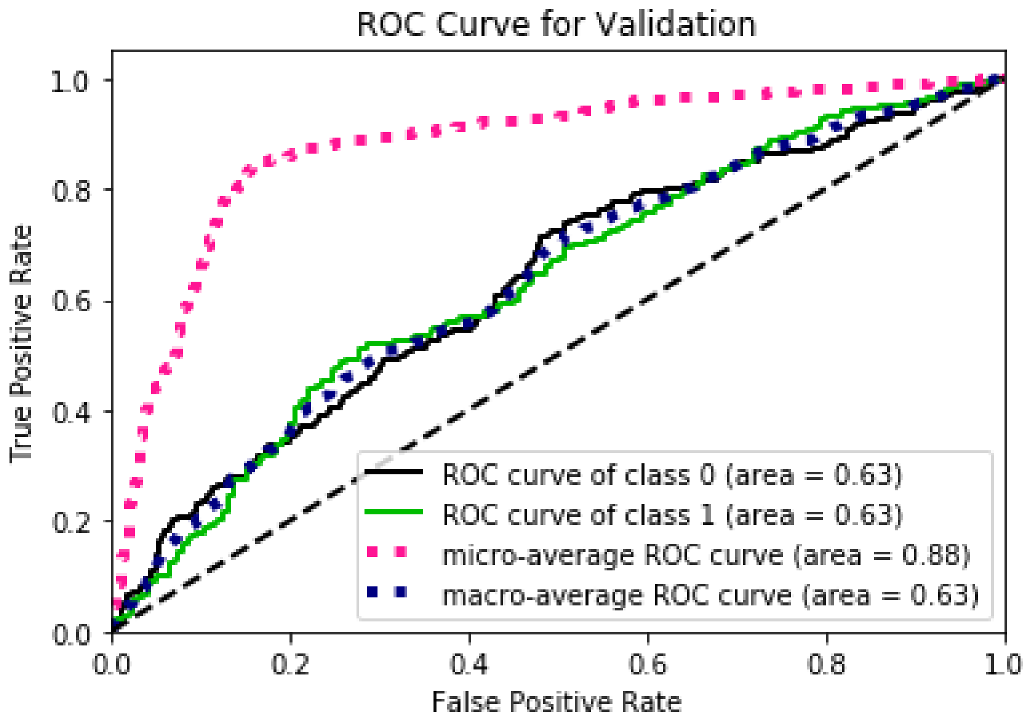





```
Area Under the Curve :  0.6302936437914766
```


1.0
0.8
0.6
0.4
0.2
0.0
0.0
0.2
0.4
0.6
0.8
1.0
Train
Test


ROC Curve for Train
True Positive Rate
False Positive Rate
1.0
0.8
0.6
0.4
0.2
0.0
0.0
0.2
0.4
0.6
0.8
1.0
ROC curve of class 0 (area = 1.00)
ROC curve of class 1 (area = 1.00)
micro-average ROC curve (area = 1.00)
macro-average ROC curve (area = 1.00)




```
Confusion Matrix :
 [[   0  199]
 [   0 1048]]
True Negative :  0
False Positive :  199
True Positive :  1048
False Negative :  0
```

7 [6] Conclusions

In [63]:
```
# Please compare all your models using Prettytable library
```

In [64]:
```
x = PrettyTable()

x.field_names = ["Vectorizer", "Model", "Hyper parameter", "AUC"]
[x.add_row(i) for i in prettytable_data]
print(x)
```

```
+---------------+---------+-----------------+--------------------+
|   Vectorizer  |  Model  | Hyper parameter |        AUC         |
```

BOW	brute	176	0.7619322914873843
TF-IDF	brute	299	0.8718646539440431
Avg. W2V	brute	76	0.6495227390655259
TF-IDF. W2V	brute	96	0.60942321846935
BOW-kd	kd_tree	176	0.7619322914873843
TF-IDF-kd	kd_tree	299	0.8718646539440431
Avg. W2V-kd	kd_tree	76	0.6495227390655259
TF-IDF W2V-kd	kd_tree	96	0.60942321846935

8 TF-IDF Vectorizer seem to perform the best with the max AUC score of 0.87