KNN - Amazon Fine Food Reviews

July 19, 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 [1]: %matplotlib inline
        import warnings
        warnings.filterwarnings("ignore")

        import sqlite3
        import pandas as pd
        import numpy as np
        import string
        import matplotlib.pyplot as plt
        import seaborn as sns
        from sklearn.feature_extraction.text import CountVectorizer, TfidfVectorizer
        from sklearn.metrics import roc_curve, auc,confusion_matrix, f1_score
        from nltk.stem.porter import PorterStemmer

        import re
        from nltk.corpus import stopwords

        from gensim.models import Word2Vec
        from gensim.models import KeyedVectors

        from tqdm import tqdm
        from sklearn.model_selection import train_test_split, TimeSeriesSplit, validation_curve
        from sklearn.neighbors import KNeighborsClassifier
        from sklearn.preprocessing import StandardScaler
        from imblearn.over_sampling import SMOTE
```

```
Using TensorFlow backend.
```

```
In [2]: # using SQLite Table to read data.
        con = sqlite3.connect('./Dataset/database.sqlite')

        # filtering only positive and negative reviews i.e.
        # not taking into consideration those reviews with Score=3
        # SELECT * FROM Reviews WHERE Score != 3 LIMIT 500000, will give top 500000 data points
        # you can change the number to any other number based on your computing power

        # filtered_data = pd.read_sql_query(""" SELECT * FROM Reviews WHERE Score != 3 LIMIT 500
        # for tsne assignment you can take 5k data points

        filtered_data = pd.read_sql_query(""" SELECT * FROM Reviews WHERE Score != 3""", con)

        # Give reviews with Score>3 a positive rating(1), and reviews with a score<3 a negative
        def partition(x):
```

```python
    if x < 3:
        return 0
    return 1

def findMinorClassPoints(df):
    posCount = int(df[df['Score']==1].shape[0]);
    negCount = int(df[df['Score']==0].shape[0]);
    if negCount < posCount:
        return negCount
    return posCount

#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

#Performing Downsampling
samplingCount = findMinorClassPoints(filtered_data)
postive_df = filtered_data[filtered_data['Score'] == 1].sample(n=samplingCount)
negative_df = filtered_data[filtered_data['Score'] == 0].sample(n=samplingCount)

filtered_data = pd.concat([postive_df, negative_df])

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

```
Number of data points in our data (164074, 10)
```

Out[2]:

```
            Id   ProductId          UserId        ProfileName  \
274535  297545  B001410X52  A3KRCYY697J5ZR       Emily Rostel
437684  473329  B0030VJ79Q  A31WB7EE8KED07              Alisa
202798  219732  B001E52YY0  A142PLA1W17R20  Traveling Grandma

        HelpfulnessNumerator  HelpfulnessDenominator  Score        Time  \
274535                     1                       2      1  1231891200
437684                     0                       0      1  1314403200
202798                     1                       1      1  1261785600

                                                Summary  \
274535  Perfect- even for a finicky eater with allergies!
437684                                Easy and Convenient
202798       Healthy snack...ignore the negative review

                                                   Text
274535  My 60 lb dog LOVES these.  She is very picky a...
437684  I really love this product, because, my daught...
202798  I love these because they are no carb, and low...
```

```
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  \
        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 [5]: display[display['UserId']=='AZY10LLTJ71NX']
```

```
Out[5]:                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 [6]: display['COUNT(*)'].sum()
```

```
Out[6]: 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 [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  \
        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 [8]: #Sorting data according to ProductId in ascending order
```

```
sorted_data=filtered_data.sort_values('ProductId', axis=0, ascending=True, inplace=False
```

In [9]:
```
#Deduplication of entries
final=sorted_data.drop_duplicates(subset={"UserId","ProfileName","Time","Text"}, keep='f
final.shape
```

Out[9]:
```
(128566, 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]:
```
78.35854553433207
```

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  \
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 [12]:
```
final=final[final.HelpfulnessNumerator<=final.HelpfulnessDenominator]
```

In [13]:
```
#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()
```

```
(128566, 10)
```

```
Out[13]: 1    71454
         0    57112
         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 [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)
```

```
My daughter loves all the "Really Rosie" books. She was introduced to the Really Rosie CD perfor
==================================================
This tea is very delicious and flavorful.  The aroma is very inviting and doesn't really require
==================================================
The product looked great when received.  I was actually shocked at the packaging - well protecte
==================================================
```

```
Fabulous product line!  No odors!  No stains!  I have two male 8 pound Pomeranians and they love
==================================================
```

```
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)
```

```
My daughter loves all the "Really Rosie" books. She was introduced to the Really Rosie CD perfor
```

```
In [16]: # https://stackoverflow.com/questions/16206380/python-beautifulsoup-how-to-remove-all-t
         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)
```

```
My daughter loves all the "Really Rosie" books. She was introduced to the Really Rosie CD perfor
==================================================
This tea is very delicious and flavorful.  The aroma is very inviting and doesn't really require
==================================================
The product looked great when received.  I was actually shocked at the packaging - well protecte
==================================================
Fabulous product line!  No odors!  No stains!  I have two male 8 pound Pomeranians and they love
```

```
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)
```

```
The product looked great when received.  I was actually shocked at the packaging - well protecte
==================================================
```

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)
```

```
My daughter loves all the "Really Rosie" books. She was introduced to the Really Rosie CD perfor
```

In [20]:
```
#remove spacial character: https://stackoverflow.com/a/5843547/4084039
sent_1500 = re.sub('[^A-Za-z0-9]+', ' ', sent_1500)
print(sent_1500)
```

```
The product looked great when received I was actually shocked at the packaging well protected
```

In [21]:
```
# https://gist.github.com/sebleier/554280
# we are removing the words from the stop words list: 'no', 'nor', 'not'
# <br /><br /> ==> after the above steps, we are getting "br br"
# we are including them into stop words list
# instead of <br /> if we have <br/> these tags would have revmoved in the 1st step

stopwords= set(['br', 'the', 'i', 'me', 'my', 'myself', 'we', 'our', 'ours', 'ourselves
            "you'll", "you'd", 'your', 'yours', 'yourself', 'yourselves', 'he', 'him',
            'she', "she's", 'her', 'hers', 'herself', 'it', "it's", 'its', 'itself', 't
            'theirs', 'themselves', 'what', 'which', 'who', 'whom', 'this', 'that', "th
            'am', 'is', 'are', 'was', 'were', 'be', 'been', 'being', 'have', 'has', 'ha
```

```
'did', 'doing', 'a', 'an', 'the', 'and', 'but', 'if', 'or', 'because', 'as'
'at', 'by', 'for', 'with', 'about', 'against', 'between', 'into', 'through'
'above', 'below', 'to', 'from', 'up', 'down', 'in', 'out', 'on', 'off', 'ov
'then', 'once', 'here', 'there', 'when', 'where', 'why', 'how', 'all', 'any
'most', 'other', 'some', 'such', 'only', 'own', 'same', 'so', 'than', 'too'
's', 't', 'can', 'will', 'just', 'don', "don't", 'should', "should've", 'no
've', 'y', 'ain', 'aren', "aren't", 'couldn', "couldn't", 'didn', "didn't",
"hadn't", 'hasn', "hasn't", 'haven', "haven't", 'isn', "isn't", 'ma', 'migh
"mustn't", 'needn', "needn't", 'shan', "shan't", 'shouldn', "shouldn't", 'w
'won', "won't", 'wouldn', "wouldn't"])
```

In [22]:
```
# 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 stopwor
    preprocessed_reviews.append(sentance.strip())
```

```
100%|| 128566/128566 [00:50<00:00, 2563.09it/s]
```

In [23]:
```
preprocessed_reviews[1500]
```

Out[23]:
```
'product looked great received actually shocked packaging well protected'
```

[3.2] Preprocessing Review Summary

In [24]:
```
## Similartly you can do preprocessing for review summary also.
def concatenateSummaryWithText(str1, str2):
    return str1 + ' ' + str2

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

```
preprocessed_reviews = list(map(concatenateSummaryWithText, preprocessed_reviews, prepr
final['CleanedText'] = preprocessed_reviews
final['CleanedText'] = final['CleanedText'].astype('str')
```

```
100%|| 128566/128566 [00:02<00:00, 48640.76it/s]
```

5 [4] Featurization

5.1 [4.1] BAG OF WORDS

In [25]:
```
# #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])
```

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

In [26]:
```
# #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
```

5.3 [4.3] TF-IDF

In [27]:
```
# 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
```

5.4 [4.4] Word2Vec

In [28]:
```
# # 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 [29]:
```
# # 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.b
#         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,
```

In [30]:
```
# 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])
```

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

[4.4.1.1] Avg W2v

In [31]:
```
# # 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]))
```

[4.4.1.2] TFIDF weighted W2v

In [32]:
```
# # 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 [33]:
```
# # 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
```

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 matr
    <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.appliedaicour
<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 tas
    </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 e
<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 fin
<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.co
<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 form
    <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

In [34]:
```
global result_report
result_report = pd.DataFrame(columns=['VECTORIZER', 'MODEL', 'HYPERPARAMETER', 'F1_SCOR
```

In [35]:
```
#Sorting according to the time for time-based splitting
final['Time'] = pd.to_datetime(final['Time'], unit='s')
final = final.sort_values(by='Time', ascending=True)
```

In [36]:
```
#Using only 60k points for knn-brute because of the lack of resources.
min_final = final.sample(n=60000)
x_train, x_test, y_train, y_test = train_test_split(min_final['CleanedText'], min_final
                                                    test_size=0.30, stratify=min_final[
y_train_tmp = y_train
```

In [37]:
```
k_range = np.array(np.arange(3, 50, 4))
k_range
```

Out[37]:
```
array([ 3,  7, 11, 15, 19, 23, 27, 31, 35, 39, 43, 47])
```

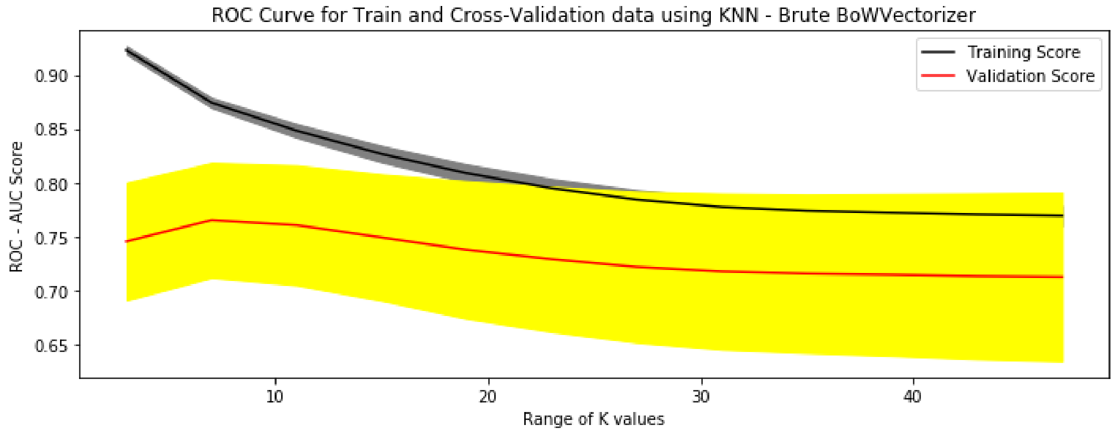
6.1.1 [5.1.1] Applying KNN brute force on BOW, SET 1

In [38]:
```
#Applying BoW Vectorizer on Train and Test Set
bow_model = CountVectorizer(min_df=10)
bow_model.fit(x_train)
```

```
x_train_bow = bow_model.transform(x_train)
x_test_bow = bow_model.transform(x_test)

std_clf = StandardScaler(with_mean=False)
x_train_bow = std_clf.fit_transform(x_train_bow)
x_test_bow = std_clf.transform(x_test_bow)
```

In [39]:
```
sm = SMOTE(random_state=2)
x_train_bow, y_train = sm.fit_sample(x_train_bow, y_train_tmp)
```

In [40]:
```
%%time
# Calculate roc_auc on training set using range of parameter values
train_score, validation_score = validation_curve(estimator=KNeighborsClassifier(algorit
                                                  X=x_train_bow, y=y_train,
                                                  param_name='n_neighbors',
                                                  param_range=k_range,
                                                  cv = 10, scoring='roc_auc', n_jobs=

#Calculate mean and standard deviation for training set scores
mean_train_score = np.mean(train_score, axis = 1)
std_train_score = np.std(train_score, axis = 1)

#Calculate mean and standard deviation for cv set scores
mean_test_score = np.mean(validation_score, axis = 1)
std_test_score = np.std(validation_score, axis = 1)

plt.figure(figsize=(10, 4))
#Plot mean accuracy for train and cv set scores
plt.plot(k_range, mean_train_score, label='Training Score', color='black')
plt.plot(k_range, mean_test_score, label='Validation Score', color='red')

#Plot accuracy bands for training and cv sets
plt.fill_between(k_range, mean_train_score - std_train_score, mean_train_score + std_tr
plt.fill_between(k_range, mean_test_score - std_test_score, mean_test_score + std_test_

# Create plot
plt.title("ROC Curve for Train and Cross-Validation data using KNN - Brute BoWVectorize
plt.xlabel("Range of K values")
plt.ylabel("ROC - AUC Score")
plt.tight_layout()
plt.legend(loc="best")
plt.show()
```

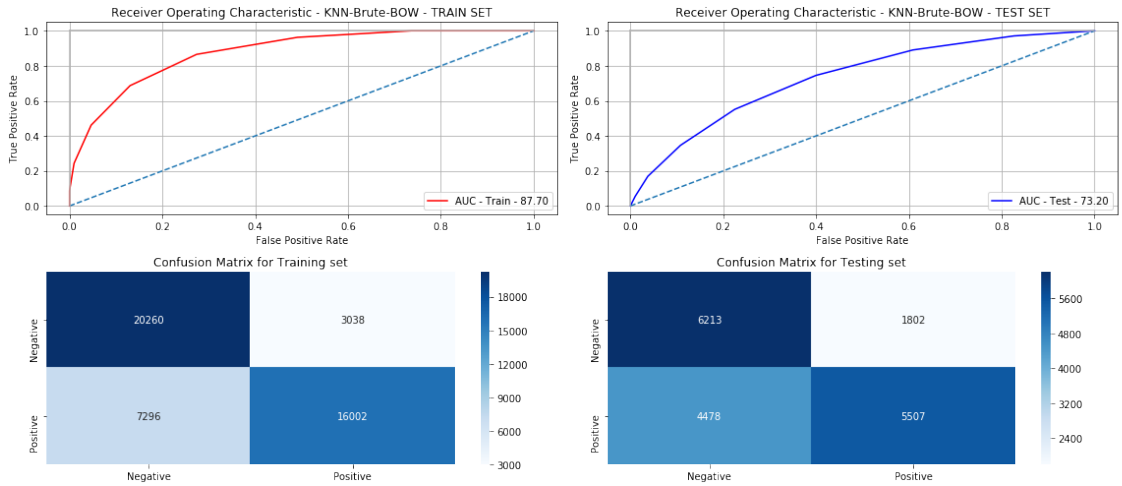



```
CPU times: user 1.58 s, sys: 900 ms, total: 2.48 s
Wall time: 36min 23s
```

In [41]:
```
%%time
#Finding the Optimal K for the final test set
optimal_K = int(k_range[mean_test_score.argmax()])

#Training the KNN model with optimal K
clf = KNeighborsClassifier(n_neighbors=optimal_K, algorithm='brute', n_jobs=-1)

clf.fit(x_train_bow, y_train)
# Get predicted values for test data
pred_test = clf.predict(x_test_bow)
pred_train = clf.predict(x_train_bow)
pred_proba_train = clf.predict_proba(x_train_bow)[:,1]
pred_proba_test = clf.predict_proba(x_test_bow)[:,1]

fpr_train, tpr_train, thresholds_train = roc_curve(y_train, pred_proba_train, pos_label
fpr_test, tpr_test, thresholds_test = roc_curve(y_test, pred_proba_test, pos_label=1)
conf_mat_train = confusion_matrix(y_train, pred_train, labels=[0, 1])
conf_mat_test = confusion_matrix(y_test, pred_test, labels=[0, 1])
f1_sc = f1_score(y_test, pred_test, average='binary', pos_label=1)
auc_sc_train = auc(fpr_train, tpr_train)
auc_sc = auc(fpr_test, tpr_test)

print("Optimal K: {} with AUC: {:.2f}%".format(optimal_K, float(auc_sc*100)))
#Saving the report in a global variable
result_report = result_report.append({'VECTORIZER': 'Bag Of Words(BOW)',
                                       'MODEL': 'Brute',
                                       'HYPERPARAMETER': optimal_K,
                                       'F1_SCORE': f1_sc, 'AUC': auc_sc
```

```
                                    }, ignore_index=True)

        plt.close()
        plt.figure(figsize=(16,7))
        # Plot ROC curve for training set
        plt.subplot(2, 2, 1)
        plt.title('Receiver Operating Characteristic - KNN-Brute-BOW - TRAIN SET')
        plt.plot(fpr_train, tpr_train, color='red', label='AUC - Train - {:.2f}'.format(float(a
        plt.plot([0, 1], ls="--")
        plt.plot([0, 0], [1, 0] , c=".7"), plt.plot([1, 1] , c=".7")
        plt.ylabel('True Positive Rate')
        plt.xlabel('False Positive Rate')
        plt.grid()
        plt.legend(loc='best')

        # Plot ROC curve for test set
        plt.subplot(2, 2, 2)
        plt.title('Receiver Operating Characteristic - KNN-Brute-BOW - TEST SET')
        plt.plot(fpr_test, tpr_test, color='blue', label='AUC - Test - {:.2f}'.format(float(auc
        plt.plot([0, 1], ls="--")
        plt.plot([0, 0], [1, 0] , c=".7"), plt.plot([1, 1] , c=".7")
        plt.ylabel('True Positive Rate')
        plt.xlabel('False Positive Rate')
        plt.grid()
        plt.legend(loc='best')

        #Plotting the confusion matrix for train
        plt.subplot(2, 2, 3)
        plt.title('Confusion Matrix for Training set')
        df_cm = pd.DataFrame(conf_mat_train, index = ["Negative", "Positive"],
                          columns = ["Negative", "Positive"])
        sns.heatmap(df_cm, annot=True,cmap='Blues', fmt='g')

        #Plotting the confusion matrix for test
        plt.subplot(2, 2, 4)
        plt.title('Confusion Matrix for Testing set')
        df_cm = pd.DataFrame(conf_mat_test, index = ["Negative", "Positive"],
                          columns = ["Negative", "Positive"])
        sns.heatmap(df_cm, annot=True,cmap='Blues', fmt='g')

        plt.tight_layout()
        plt.show()
```

```
Optimal K: 7 with AUC: 73.20%
```




```
CPU times: user 9min 45s, sys: 7min 43s, total: 17min 28s
Wall time: 5min 7s
```

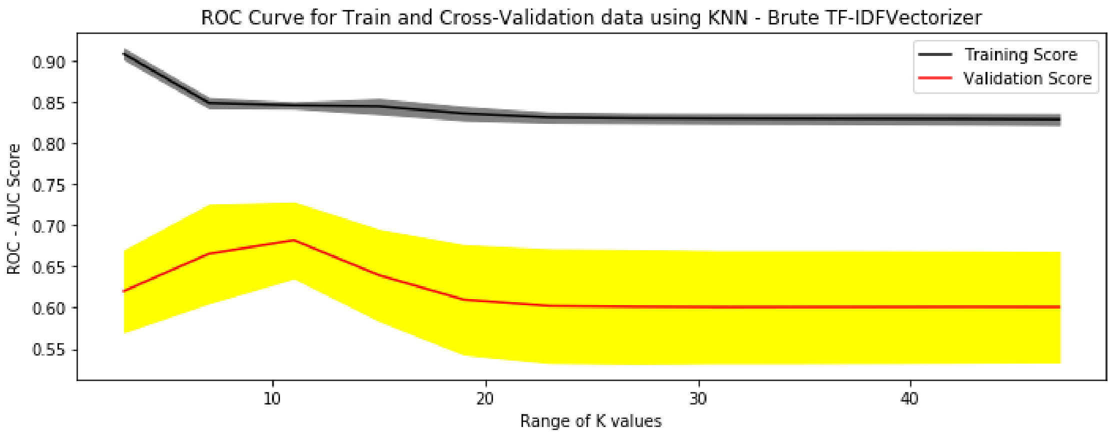
6.1.2 [5.1.2] Applying KNN brute force on TFIDF, SET 2

In [42]:
```
#Applying TFIDF Vectorizer on Train and Test Set
tfidf_model = TfidfVectorizer(min_df=10)
tfidf_model.fit(x_train)

x_train_tfidf = tfidf_model.transform(x_train)
x_test_tfidf = tfidf_model.transform(x_test)

std_clf = StandardScaler(with_mean=False)
x_train_tfidf = std_clf.fit_transform(x_train_tfidf)
x_test_tfidf = std_clf.transform(x_test_tfidf)
```

In [43]:
```
sm = SMOTE(random_state=2)
x_train_tfidf, y_train = sm.fit_sample(x_train_tfidf, y_train_tmp)
```

In [44]:
```
%%time
# Calculate roc_auc on training set using range of parameter values
train_score, validation_score = validation_curve(estimator=KNeighborsClassifier(algorit
                                                  X=x_train_tfidf, y=y_train,
                                                  param_name='n_neighbors',
                                                  param_range=k_range,
                                                  cv = 10, scoring='roc_auc', n_jobs=

#Calculate mean and standard deviation for training set scores
mean_train_score = np.mean(train_score, axis = 1)
std_train_score = np.std(train_score, axis = 1)
```

```
#Calculate mean and standard deviation for cv set scores
mean_test_score = np.mean(validation_score, axis = 1)
std_test_score = np.std(validation_score, axis = 1)

plt.figure(figsize=(10, 4))
#Plot mean accuracy for train and cv set scores
plt.plot(k_range, mean_train_score, label='Training Score', color='black')
plt.plot(k_range, mean_test_score, label='Validation Score', color='red')

#Plot accuracy bands for training and cv sets
plt.fill_between(k_range, mean_train_score - std_train_score, mean_train_score + std_tr
plt.fill_between(k_range, mean_test_score - std_test_score, mean_test_score + std_test_

# Create plot
plt.title("ROC Curve for Train and Cross-Validation data using KNN - Brute TF-IDFVector
plt.xlabel("Range of K values")
plt.ylabel("ROC - AUC Score")
plt.tight_layout()
plt.legend(loc="best")
plt.show()
```



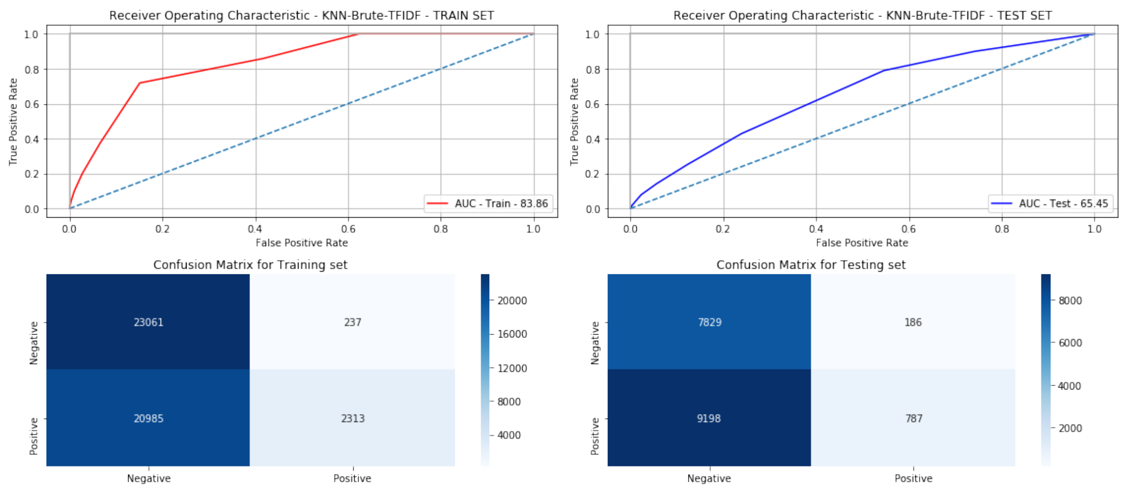

```
CPU times: user 1.47 s, sys: 984 ms, total: 2.45 s
Wall time: 35min 43s
```

In [45]:
```
%%time
#Finding the Optimal K for the final test set
optimal_K = int(k_range[mean_test_score.argmax()])

#Training the KNN model with optimal K
clf = KNeighborsClassifier(n_neighbors=optimal_K, algorithm='brute', n_jobs=-1)
```

```
clf.fit(x_train_tfidf, y_train)
# Get predicted values for test data
pred_test = clf.predict(x_test_tfidf)
pred_train = clf.predict(x_train_tfidf)
pred_proba_train = clf.predict_proba(x_train_tfidf)[:,1]
pred_proba_test = clf.predict_proba(x_test_tfidf)[:,1]

fpr_train, tpr_train, thresholds_train = roc_curve(y_train, pred_proba_train, pos_label
fpr_test, tpr_test, thresholds_test = roc_curve(y_test, pred_proba_test, pos_label=1)
conf_mat_train = confusion_matrix(y_train, pred_train, labels=[0, 1])
conf_mat_test = confusion_matrix(y_test, pred_test, labels=[0, 1])
f1_sc = f1_score(y_test, pred_test, average='binary', pos_label=1)
auc_sc_train = auc(fpr_train, tpr_train)
auc_sc = auc(fpr_test, tpr_test)

print("Optimal K: {} with AUC: {:.2f}%".format(optimal_K, float(auc_sc*100)))
#Saving the report in a global variable
result_report = result_report.append({'VECTORIZER': 'TF-IDF',
                                      'MODEL': 'Brute',
                                      'HYPERPARAMETER': optimal_K,
                                      'F1_SCORE': f1_sc, 'AUC': auc_sc
                                     }, ignore_index=True)


plt.close()
plt.figure(figsize=(16,7))
# Plot ROC curve for training set
plt.subplot(2, 2, 1)
plt.title('Receiver Operating Characteristic - KNN-Brute-TFIDF - TRAIN SET')
plt.plot(fpr_train, tpr_train, color='red', label='AUC - Train - {:.2f}'.format(float(a
plt.plot([0, 1], ls="--")
plt.plot([0, 0], [1, 0] , c=".7"), plt.plot([1, 1] , c=".7")
plt.ylabel('True Positive Rate')
plt.xlabel('False Positive Rate')
plt.grid()
plt.legend(loc='best')

# Plot ROC curve for test set
plt.subplot(2, 2, 2)
plt.title('Receiver Operating Characteristic - KNN-Brute-TFIDF - TEST SET')
plt.plot(fpr_test, tpr_test, color='blue', label='AUC - Test - {:.2f}'.format(float(auc
plt.plot([0, 1], ls="--")
plt.plot([0, 0], [1, 0] , c=".7"), plt.plot([1, 1] , c=".7")
plt.ylabel('True Positive Rate')
plt.xlabel('False Positive Rate')
plt.grid()
plt.legend(loc='best')
```

```
#Plotting the confusion matrix for train
plt.subplot(2, 2, 3)
plt.title('Confusion Matrix for Training set')
df_cm = pd.DataFrame(conf_mat_train, index = ["Negative", "Positive"],
                  columns = ["Negative", "Positive"])
sns.heatmap(df_cm, annot=True,cmap='Blues', fmt='g')

#Plotting the confusion matrix for test
plt.subplot(2, 2, 4)
plt.title('Confusion Matrix for Testing set')
df_cm = pd.DataFrame(conf_mat_test, index = ["Negative", "Positive"],
                  columns = ["Negative", "Positive"])
sns.heatmap(df_cm, annot=True,cmap='Blues', fmt='g')

plt.tight_layout()
plt.show()
```

```
Optimal K: 11 with AUC: 65.45%
```




```
CPU times: user 9min 39s, sys: 7min 33s, total: 17min 13s
Wall time: 4min 58s
```

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

```
In [46]: list_of_sent_train = []
         list_of_sent_test = []

         for sent in x_train:
```

```
        list_of_sent_train.append(sent.split())
    for sent in x_test:
        list_of_sent_test.append(sent.split())

    w2v_model=Word2Vec(list_of_sent_train,min_count=5,size=50, workers=8)
    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])
```

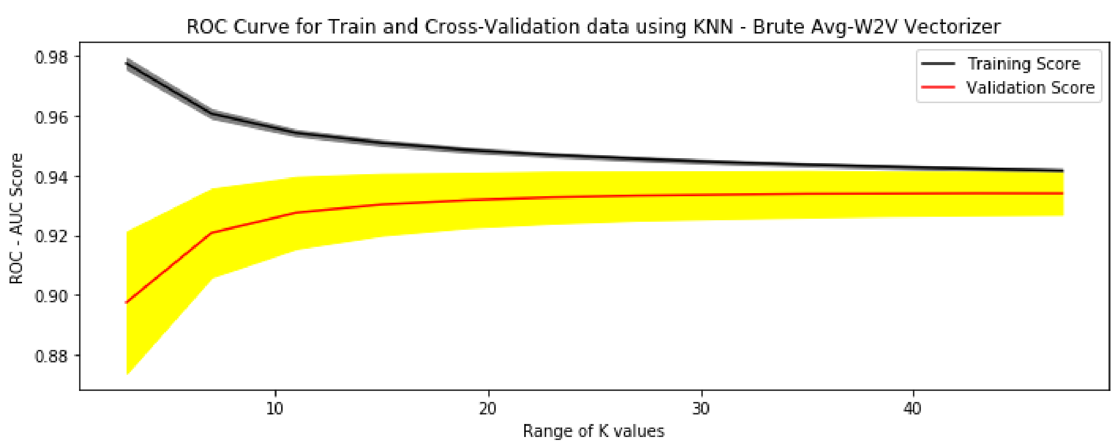
```
number of words that occured minimum 5 times  13449
sample words  ['sons', 'goji', 'scares', 'dive', 'foamy', 'chilis', 'vendor', 'institution', 'fr
```

```
In [47]: # compute average word2vec for each review for train data
    avgw2v_train = [] # the avg-w2v for each sentence/review is stored in this list
    for sent in tqdm(list_of_sent_train): # for each review/sentence
        sent_vec = np.zeros(50)
        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
        avgw2v_train.append(sent_vec)

    # compute average word2vec for each review for test data
    avgw2v_test = [] # the avg-w2v for each sentence/review is stored in this list
    for sent in tqdm(list_of_sent_test): # for each review/sentence
        sent_vec = np.zeros(50)
        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
        avgw2v_test.append(sent_vec)

    std_clf = StandardScaler(with_mean=False)
    avgw2v_train = std_clf.fit_transform(avgw2v_train)
    avgw2v_test = std_clf.transform(avgw2v_test)
```

```
100%|| 42000/42000 [07:48<00:00, 89.74it/s]
100%|| 18000/18000 [03:21<00:00, 89.46it/s]
```

In [48]:
```
sm = SMOTE(random_state=2)
avgw2v_train, y_train = sm.fit_sample(avgw2v_train, y_train_tmp)
```

In [49]:
```
%%time
# Calculate roc_auc on training set using range of parameter values
train_score, validation_score = validation_curve(estimator=KNeighborsClassifier(algorit
                                                   X=avgw2v_train, y=y_train,
                                                   param_name='n_neighbors',
                                                   param_range=k_range,
                                                   cv = 10, scoring='roc_auc', n_jobs=

#Calculate mean and standard deviation for training set scores
mean_train_score = np.mean(train_score, axis = 1)
std_train_score = np.std(train_score, axis = 1)

#Calculate mean and standard deviation for cv set scores
mean_test_score = np.mean(validation_score, axis = 1)
std_test_score = np.std(validation_score, axis = 1)

plt.figure(figsize=(10, 4))
#Plot mean accuracy for train and cv set scores
plt.plot(k_range, mean_train_score, label='Training Score', color='black')
plt.plot(k_range, mean_test_score, label='Validation Score', color='red')

#Plot accuracy bands for training and cv sets
plt.fill_between(k_range, mean_train_score - std_train_score, mean_train_score + std_tr
plt.fill_between(k_range, mean_test_score - std_test_score, mean_test_score + std_test_

# Create plot
plt.title("ROC Curve for Train and Cross-Validation data using KNN - Brute Avg-W2V Vect
plt.xlabel("Range of K values")
plt.ylabel("ROC - AUC Score")
plt.tight_layout()
plt.legend(loc="best")
plt.show()
```

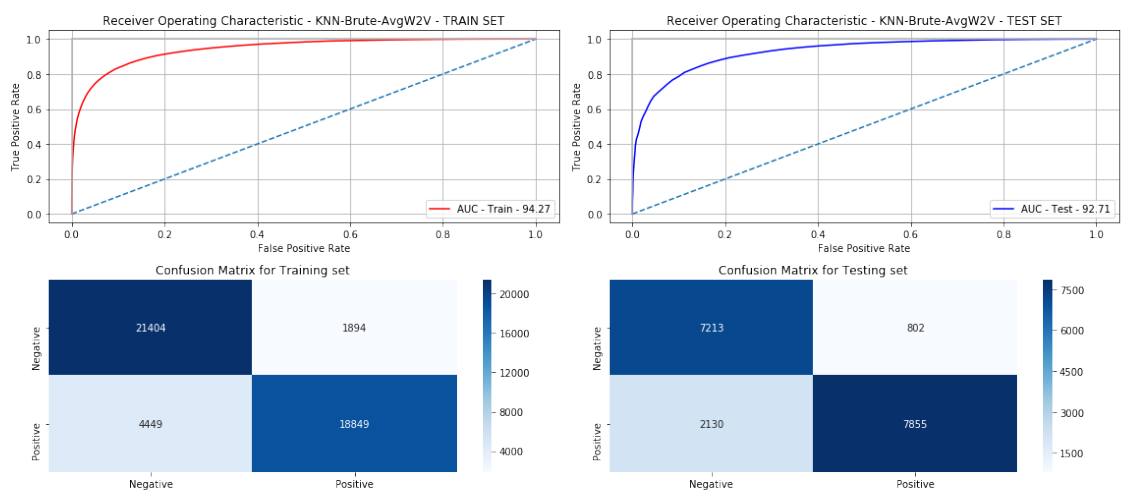


```
CPU times: user 1.26 s, sys: 984 ms, total: 2.24 s
Wall time: 22min 19s
```

```
In [50]: %%time
         #Finding the Optimal K for the final test set
         optimal_K = int(k_range[mean_test_score.argmax()])

         #Training the KNN model with optimal K
         clf = KNeighborsClassifier(n_neighbors=optimal_K, algorithm='brute', n_jobs=-1)

         clf.fit(avgw2v_train, y_train)
         # Get predicted values for test data
         pred_test = clf.predict(avgw2v_test)
         pred_train = clf.predict(avgw2v_train)
         pred_proba_train = clf.predict_proba(avgw2v_train)[:,1]
         pred_proba_test = clf.predict_proba(avgw2v_test)[:,1]

         fpr_train, tpr_train, thresholds_train = roc_curve(y_train, pred_proba_train, pos_label
         fpr_test, tpr_test, thresholds_test = roc_curve(y_test, pred_proba_test, pos_label=1)
         conf_mat_train = confusion_matrix(y_train, pred_train, labels=[0, 1])
         conf_mat_test = confusion_matrix(y_test, pred_test, labels=[0, 1])
         f1_sc = f1_score(y_test, pred_test, average='binary', pos_label=1)
         auc_sc_train = auc(fpr_train, tpr_train)
         auc_sc = auc(fpr_test, tpr_test)

         print("Optimal K: {} with AUC: {:.2f}%".format(optimal_K, float(auc_sc*100)))
         #Saving the report in a global variable
         result_report = result_report.append({'VECTORIZER': 'Avg-W2V',
                                               'MODEL': 'Brute',
                                               'HYPERPARAMETER': optimal_K,
                                               'F1_SCORE': f1_sc, 'AUC': auc_sc
                                              }, ignore_index=True)



         plt.close()
         plt.figure(figsize=(16,7))
         # Plot ROC curve for training set
         plt.subplot(2, 2, 1)
         plt.title('Receiver Operating Characteristic - KNN-Brute-AvgW2V - TRAIN SET')
         plt.plot(fpr_train, tpr_train, color='red', label='AUC - Train - {:.2f}'.format(float(a
         plt.plot([0, 1], ls="--")
         plt.plot([0, 0], [1, 0] , c=".7"), plt.plot([1, 1] , c=".7")
         plt.ylabel('True Positive Rate')
         plt.xlabel('False Positive Rate')
```

```
plt.grid()
plt.legend(loc='best')

# Plot ROC curve for test set
plt.subplot(2, 2, 2)
plt.title('Receiver Operating Characteristic - KNN-Brute-AvgW2V - TEST SET')
plt.plot(fpr_test, tpr_test, color='blue', label='AUC - Test - {:.2f}'.format(float(auc
plt.plot([0, 1], ls="--")
plt.plot([0, 0], [1, 0] , c=".7"), plt.plot([1, 1] , c=".7")
plt.ylabel('True Positive Rate')
plt.xlabel('False Positive Rate')
plt.grid()
plt.legend(loc='best')

#Plotting the confusion matrix for train
plt.subplot(2, 2, 3)
plt.title('Confusion Matrix for Training set')
df_cm = pd.DataFrame(conf_mat_train, index = ["Negative", "Positive"],
                  columns = ["Negative", "Positive"])
sns.heatmap(df_cm, annot=True,cmap='Blues', fmt='g')

#Plotting the confusion matrix for test
plt.subplot(2, 2, 4)
plt.title('Confusion Matrix for Testing set')
df_cm = pd.DataFrame(conf_mat_test, index = ["Negative", "Positive"],
                  columns = ["Negative", "Positive"])
sns.heatmap(df_cm, annot=True,cmap='Blues', fmt='g')

plt.tight_layout()
plt.show()
```

```
Optimal K: 43 with AUC: 92.71%
```



```
CPU times: user 9min 24s, sys: 3min 52s, total: 13min 17s
Wall time: 4min 37s
```

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

```
In [51]: model = TfidfVectorizer()
         model.fit(x_train)

         #Creating the TFIDF W2V Training Set
         # 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

         tfidfw2v_train = []; # the tfidf-w2v for each sentence/review is stored in this list
         row=0;
         for sent in tqdm(list_of_sent_train): # for each review/sentence
             sent_vec = np.zeros(50) # as word vectors are of zero length
             weight_sum =0; # num of words with a valid vector in the sentence/review
             for word in sent: # for each word in a review/sentence
                 if word in w2v_words and word in tfidf_feat:
                     vec = w2v_model.wv[word]
                     tf_idf = dictionary[word]*(sent.count(word)/len(sent))
                     sent_vec += (vec * tf_idf)
                     weight_sum += tf_idf
             if weight_sum != 0:
                 sent_vec /= weight_sum
             tfidfw2v_train.append(sent_vec)
             row += 1

         #Creating the TFIDF W2V Testing Set

         # 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

         tfidfw2v_test = []; # the tfidf-w2v for each sentence/review is stored in this list
         row=0;
         for sent in tqdm(list_of_sent_test): # for each review/sentence
             sent_vec = np.zeros(50) # as word vectors are of zero length
             weight_sum =0; # num of words with a valid vector in the sentence/review
             for word in sent: # for each word in a review/sentence
```

```
            if word in w2v_words and word in tfidf_feat:
                vec = w2v_model.wv[word]
                tf_idf = dictionary[word]*(sent.count(word)/len(sent))
                sent_vec += (vec * tf_idf)
                weight_sum += tf_idf
        if weight_sum != 0:
            sent_vec /= weight_sum
        tfidfw2v_test.append(sent_vec)
        row += 1

    std_clf = StandardScaler(with_mean=False)
    tfidfw2v_train = std_clf.fit_transform(tfidfw2v_train)
    tfidfw2v_test = std_clf.transform(tfidfw2v_test)
```

```
100%|| 42000/42000 [23:55<00:00, 29.25it/s]
100%|| 18000/18000 [10:23<00:00, 28.87it/s]
```

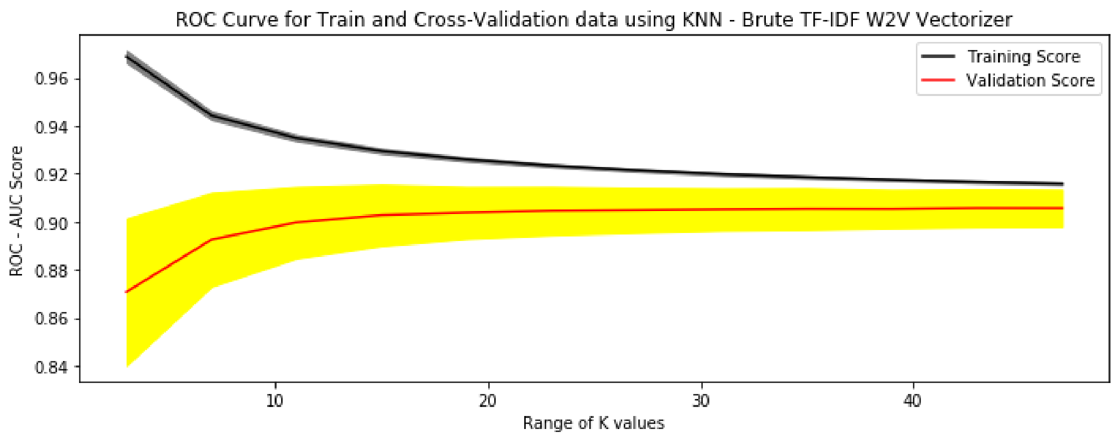
In [52]:

```
sm = SMOTE(random_state=2)
tfidfw2v_train, y_train = sm.fit_sample(tfidfw2v_train, y_train_tmp)
```

In [53]:

```
%%time
# Calculate roc_auc on training set using range of parameter values
train_score, validation_score = validation_curve(estimator=KNeighborsClassifier(algorit
                                                      X=tfidfw2v_train, y=y_train,
                                                      param_name='n_neighbors',
                                                      param_range=k_range,
                                                      cv = 10, scoring='roc_auc', n_jobs=

#Calculate mean and standard deviation for training set scores
mean_train_score = np.mean(train_score, axis = 1)
std_train_score = np.std(train_score, axis = 1)

#Calculate mean and standard deviation for cv set scores
mean_test_score = np.mean(validation_score, axis = 1)
std_test_score = np.std(validation_score, axis = 1)

plt.figure(figsize=(10, 4))
#Plot mean accuracy for train and cv set scores
plt.plot(k_range, mean_train_score, label='Training Score', color='black')
plt.plot(k_range, mean_test_score, label='Validation Score', color='red')

#Plot accuracy bands for training and cv sets
plt.fill_between(k_range, mean_train_score - std_train_score, mean_train_score + std_tr
plt.fill_between(k_range, mean_test_score - std_test_score, mean_test_score + std_test_

# Create plot
plt.title("ROC Curve for Train and Cross-Validation data using KNN - Brute TF-IDF W2V V
```

```
plt.xlabel("Range of K values")
plt.ylabel("ROC - AUC Score")
plt.tight_layout()
plt.legend(loc="best")
plt.show()
```



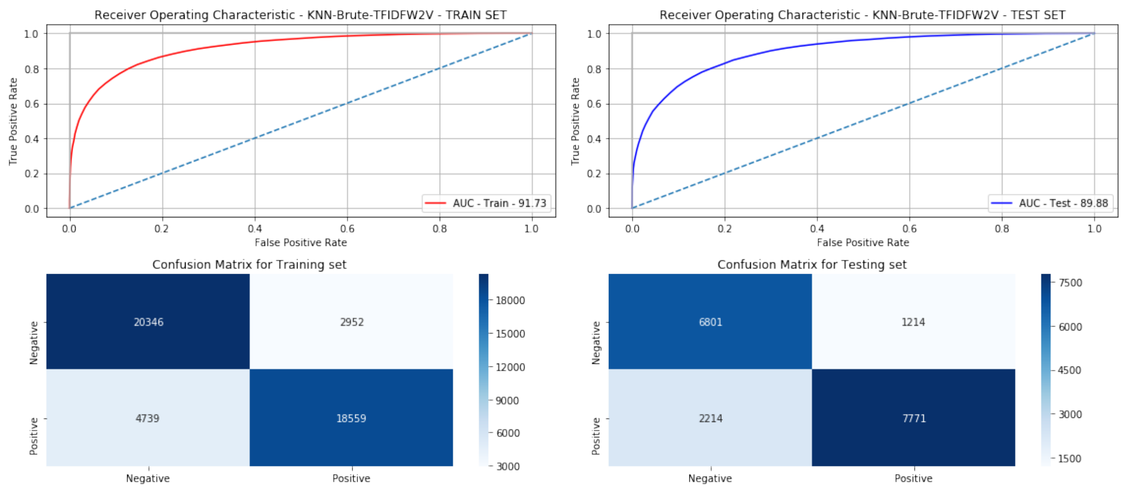

```
CPU times: user 1.36 s, sys: 920 ms, total: 2.28 s
Wall time: 22min 36s
```

In [54]:
```
%%time
#Finding the Optimal K for the final test set
optimal_K = int(k_range[mean_test_score.argmax()])

#Training the KNN model with optimal K
clf = KNeighborsClassifier(n_neighbors=optimal_K, algorithm='brute', n_jobs=-1)

clf.fit(tfidfw2v_train, y_train)
# Get predicted values for test data
pred_test = clf.predict(tfidfw2v_test)
pred_train = clf.predict(tfidfw2v_train)
pred_proba_train = clf.predict_proba(tfidfw2v_train)[:,1]
pred_proba_test = clf.predict_proba(tfidfw2v_test)[:,1]

fpr_train, tpr_train, thresholds_train = roc_curve(y_train, pred_proba_train, pos_label
fpr_test, tpr_test, thresholds_test = roc_curve(y_test, pred_proba_test, pos_label=1)
conf_mat_train = confusion_matrix(y_train, pred_train, labels=[0, 1])
conf_mat_test = confusion_matrix(y_test, pred_test, labels=[0, 1])
f1_sc = f1_score(y_test, pred_test, average='binary', pos_label=1)
auc_sc_train = auc(fpr_train, tpr_train)
auc_sc = auc(fpr_test, tpr_test)
```

```
print("Optimal K: {} with AUC: {:.2f}%".format(optimal_K, float(auc_sc*100)))
#Saving the report in a global variable
result_report = result_report.append({'VECTORIZER': 'TF-IDF W2V',
                                      'MODEL': 'Brute',
                                      'HYPERPARAMETER': optimal_K,
                                      'F1_SCORE': f1_sc, 'AUC': auc_sc
                                     }, ignore_index=True)


plt.close()
plt.figure(figsize=(16,7))
# Plot ROC curve for training set
plt.subplot(2, 2, 1)
plt.title('Receiver Operating Characteristic - KNN-Brute-TFIDFW2V - TRAIN SET')
plt.plot(fpr_train, tpr_train, color='red', label='AUC - Train - {:.2f}'.format(float(a
plt.plot([0, 1], ls="--")
plt.plot([0, 0], [1, 0] , c=".7"), plt.plot([1, 1] , c=".7")
plt.ylabel('True Positive Rate')
plt.xlabel('False Positive Rate')
plt.grid()
plt.legend(loc='best')

# Plot ROC curve for test set
plt.subplot(2, 2, 2)
plt.title('Receiver Operating Characteristic - KNN-Brute-TFIDFW2V - TEST SET')
plt.plot(fpr_test, tpr_test, color='blue', label='AUC - Test - {:.2f}'.format(float(auc
plt.plot([0, 1], ls="--")
plt.plot([0, 0], [1, 0] , c=".7"), plt.plot([1, 1] , c=".7")
plt.ylabel('True Positive Rate')
plt.xlabel('False Positive Rate')
plt.grid()
plt.legend(loc='best')

#Plotting the confusion matrix for train
plt.subplot(2, 2, 3)
plt.title('Confusion Matrix for Training set')
df_cm = pd.DataFrame(conf_mat_train, index = ["Negative", "Positive"],
                  columns = ["Negative", "Positive"])
sns.heatmap(df_cm, annot=True,cmap='Blues', fmt='g')

#Plotting the confusion matrix for test
plt.subplot(2, 2, 4)
plt.title('Confusion Matrix for Testing set')
df_cm = pd.DataFrame(conf_mat_test, index = ["Negative", "Positive"],
                  columns = ["Negative", "Positive"])
sns.heatmap(df_cm, annot=True,cmap='Blues', fmt='g')

plt.tight_layout()
```

```
        plt.show()
```

```
Optimal K: 43 with AUC: 89.88%
```




```
CPU times: user 9min 38s, sys: 3min 55s, total: 13min 34s
Wall time: 4min 53s
```

6.2 [5.2] Applying KNN kd-tree

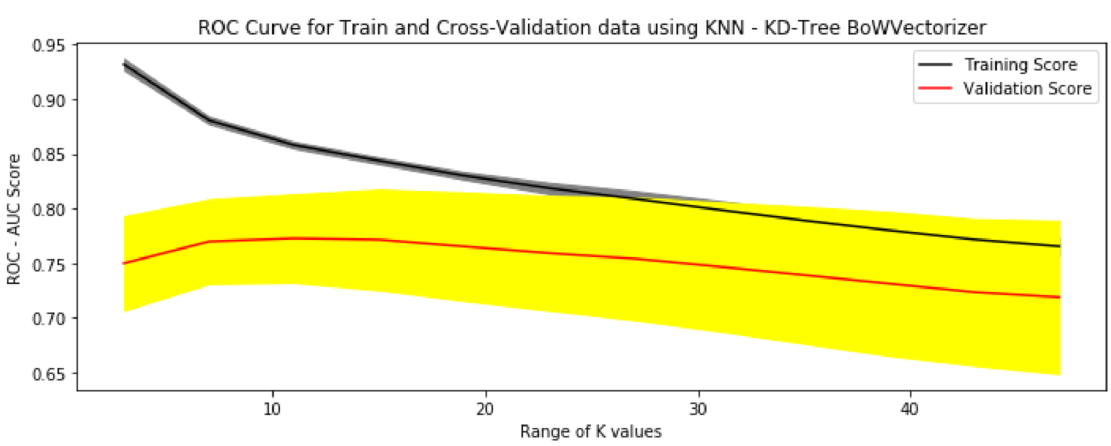
```
In [55]: #Using only 20k points for knn-kdtree because of the lack of resources.
         min_final = final.sample(n=20000)
         x_train, x_test, y_train, y_test = train_test_split(min_final['CleanedText'], min_final
                                                              test_size=0.30, stratify=min_final[
         y_train_tmp = y_train
```

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

```
In [56]: #Applying BoW Vectorizer on Train and Test Set
         bow_model = CountVectorizer(min_df=10, max_features=500)
         bow_model.fit(x_train)

         x_train_bow = bow_model.transform(x_train)
         x_test_bow = bow_model.transform(x_test)

         std_clf = StandardScaler(with_mean=False)
         x_train_bow = std_clf.fit_transform(x_train_bow).toarray()
         x_test_bow = std_clf.transform(x_test_bow).toarray()
```

```
In [57]: sm = SMOTE(random_state=2)
         x_train_bow, y_train = sm.fit_sample(x_train_bow, y_train_tmp)
```

In [58]:
```
%%time
# Calculate roc_auc on training set using range of parameter values
train_score, validation_score = validation_curve(estimator=KNeighborsClassifier(algorit
                                                 X=x_train_bow, y=y_train,
                                                 param_name='n_neighbors',
                                                 param_range=k_range,
                                                 cv = 10, scoring='roc_auc', n_jobs=

#Calculate mean and standard deviation for training set scores
mean_train_score = np.mean(train_score, axis = 1)
std_train_score = np.std(train_score, axis = 1)

#Calculate mean and standard deviation for cv set scores
mean_test_score = np.mean(validation_score, axis = 1)
std_test_score = np.std(validation_score, axis = 1)

plt.figure(figsize=(10, 4))
#Plot mean accuracy for train and cv set scores
plt.plot(k_range, mean_train_score, label='Training Score', color='black')
plt.plot(k_range, mean_test_score, label='Validation Score', color='red')

#Plot accuracy bands for training and cv sets
plt.fill_between(k_range, mean_train_score - std_train_score, mean_train_score + std_tr
plt.fill_between(k_range, mean_test_score - std_test_score, mean_test_score + std_test_

# Create plot
plt.title("ROC Curve for Train and Cross-Validation data using KNN - KD-Tree BoWVectori
plt.xlabel("Range of K values")
plt.ylabel("ROC - AUC Score")
plt.tight_layout()
plt.legend(loc="best")
plt.show()
```

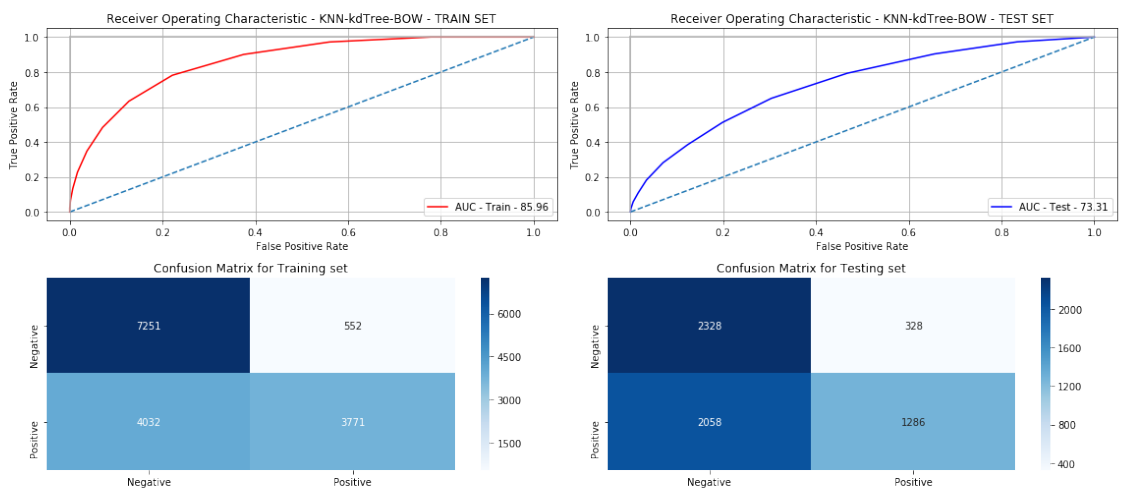


```
CPU times: user 1.34 s, sys: 1.12 s, total: 2.47 s
Wall time: 1h 17min 37s
```

```
In [59]: %%time
         #Finding the Optimal K for the final test set
         optimal_K = int(k_range[mean_test_score.argmax()])

         #Training the KNN model with optimal K
         clf = KNeighborsClassifier(n_neighbors=optimal_K, algorithm='kd_tree', n_jobs=-1)

         clf.fit(x_train_bow, y_train)
         # Get predicted values for test data
         pred_test = clf.predict(x_test_bow)
         pred_train = clf.predict(x_train_bow)
         pred_proba_train = clf.predict_proba(x_train_bow)[:,1]
         pred_proba_test = clf.predict_proba(x_test_bow)[:,1]

         fpr_train, tpr_train, thresholds_train = roc_curve(y_train, pred_proba_train, pos_label
         fpr_test, tpr_test, thresholds_test = roc_curve(y_test, pred_proba_test, pos_label=1)
         conf_mat_train = confusion_matrix(y_train, pred_train, labels=[0, 1])
         conf_mat_test = confusion_matrix(y_test, pred_test, labels=[0, 1])
         f1_sc = f1_score(y_test, pred_test, average='binary', pos_label=1)
         auc_sc_train = auc(fpr_train, tpr_train)
         auc_sc = auc(fpr_test, tpr_test)

         print("Optimal K: {} with AUC: {:.2f}%".format(optimal_K, float(auc_sc*100)))
         #Saving the report in a global variable
         result_report = result_report.append({'VECTORIZER': 'Bag Of Words(BOW)',
                                               'MODEL': 'KD-Tree',
                                               'HYPERPARAMETER': optimal_K,
                                               'F1_SCORE': f1_sc, 'AUC': auc_sc
                                              }, ignore_index=True)


         plt.close()
         plt.figure(figsize=(16,7))
         # Plot ROC curve for training set
         plt.subplot(2, 2, 1)
         plt.title('Receiver Operating Characteristic - KNN-kdTree-BOW - TRAIN SET')
         plt.plot(fpr_train, tpr_train, color='red', label='AUC - Train - {:.2f}'.format(float(a
         plt.plot([0, 1], ls="--")
         plt.plot([0, 0], [1, 0] , c=".7"), plt.plot([1, 1] , c=".7")
         plt.ylabel('True Positive Rate')
         plt.xlabel('False Positive Rate')
         plt.grid()
         plt.legend(loc='best')

         # Plot ROC curve for test set
```

```
    plt.subplot(2, 2, 2)
    plt.title('Receiver Operating Characteristic - KNN-kdTree-BOW - TEST SET')
    plt.plot(fpr_test, tpr_test, color='blue', label='AUC - Test - {:.2f}'.format(float(auc
    plt.plot([0, 1], ls="--")
    plt.plot([0, 0], [1, 0] , c=".7"), plt.plot([1, 1] , c=".7")
    plt.ylabel('True Positive Rate')
    plt.xlabel('False Positive Rate')
    plt.grid()
    plt.legend(loc='best')

    #Plotting the confusion matrix for train
    plt.subplot(2, 2, 3)
    plt.title('Confusion Matrix for Training set')
    df_cm = pd.DataFrame(conf_mat_train, index = ["Negative", "Positive"],
                      columns = ["Negative", "Positive"])
    sns.heatmap(df_cm, annot=True,cmap='Blues', fmt='g')

    #Plotting the confusion matrix for test
    plt.subplot(2, 2, 4)
    plt.title('Confusion Matrix for Testing set')
    df_cm = pd.DataFrame(conf_mat_test, index = ["Negative", "Positive"],
                      columns = ["Negative", "Positive"])
    sns.heatmap(df_cm, annot=True,cmap='Blues', fmt='g')

    plt.tight_layout()
    plt.show()
```

```
Optimal K: 11 with AUC: 73.31%
```




```
CPU times: user 20min 40s, sys: 0 ns, total: 20min 40s
```

Wall time: 2min 39s

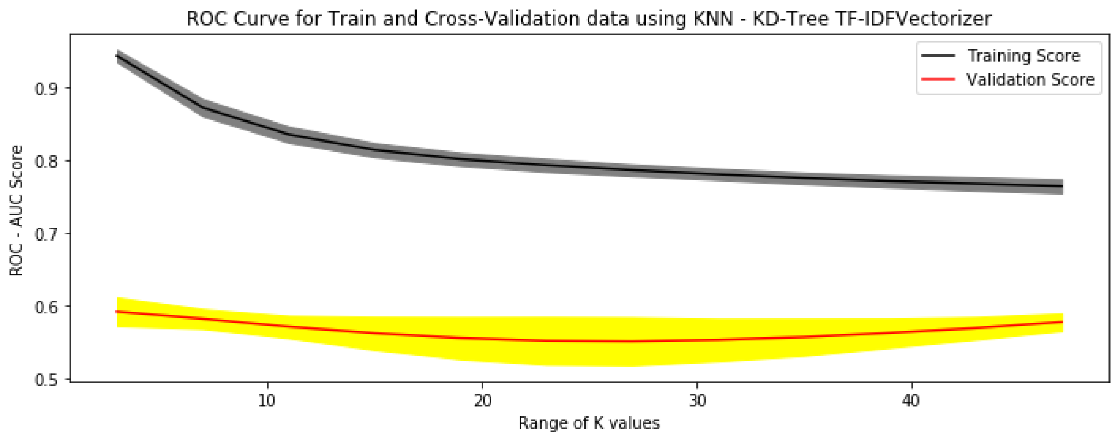
6.2.2 [5.2.2] Applying KNN kd-tree on TFIDF, SET 6

In [60]:
```
#Applying TFIDF Vectorizer on Train and Test Set
tfidf_model = TfidfVectorizer(min_df=10, max_features=500)
tfidf_model.fit(x_train)

x_train_tfidf = tfidf_model.transform(x_train)
x_test_tfidf = tfidf_model.transform(x_test)

std_clf = StandardScaler(with_mean=False)
x_train_tfidf = std_clf.fit_transform(x_train_tfidf).toarray()
x_test_tfidf = std_clf.transform(x_test_tfidf).toarray()
```

In [61]:
```
sm = SMOTE(random_state=2)
x_train_tfidf, y_train = sm.fit_sample(x_train_tfidf, y_train_tmp)
```

In [62]:
```
%%time
# Calculate roc_auc on training set using range of parameter values
train_score, validation_score = validation_curve(estimator=KNeighborsClassifier(algorit
                                                  X=x_train_tfidf, y=y_train,
                                                  param_name='n_neighbors',
                                                  param_range=k_range,
                                                  cv = 10, scoring='roc_auc', n_jobs=

#Calculate mean and standard deviation for training set scores
mean_train_score = np.mean(train_score, axis = 1)
std_train_score = np.std(train_score, axis = 1)

#Calculate mean and standard deviation for cv set scores
mean_test_score = np.mean(validation_score, axis = 1)
std_test_score = np.std(validation_score, axis = 1)

plt.figure(figsize=(10, 4))
#Plot mean accuracy for train and cv set scores
plt.plot(k_range, mean_train_score, label='Training Score', color='black')
plt.plot(k_range, mean_test_score, label='Validation Score', color='red')

#Plot accuracy bands for training and cv sets
plt.fill_between(k_range, mean_train_score - std_train_score, mean_train_score + std_tr
plt.fill_between(k_range, mean_test_score - std_test_score, mean_test_score + std_test_

# Create plot
plt.title("ROC Curve for Train and Cross-Validation data using KNN - KD-Tree TF-IDFVect
plt.xlabel("Range of K values")
plt.ylabel("ROC - AUC Score")
```

```
plt.tight_layout()
plt.legend(loc="best")
plt.show()
```

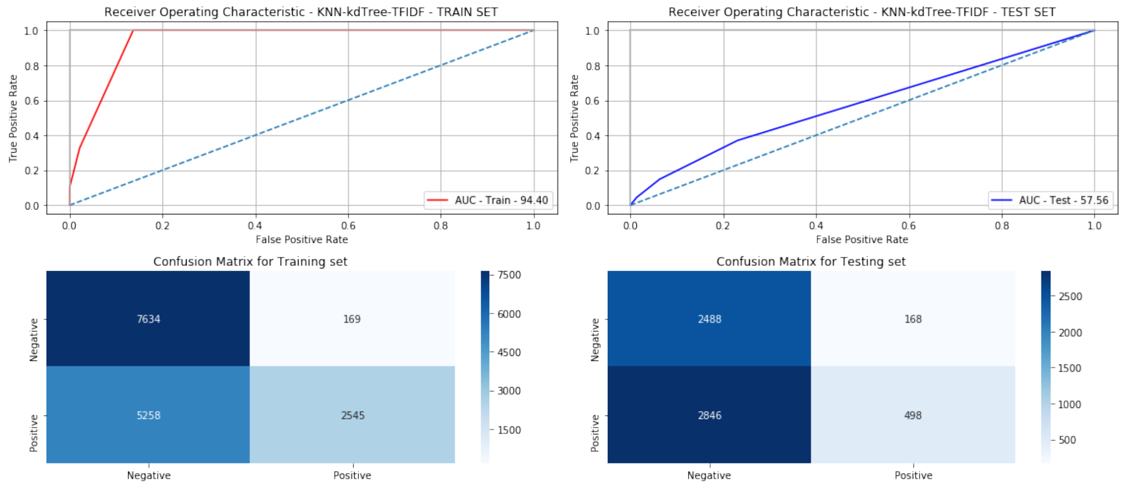



```
CPU times: user 1.45 s, sys: 0 ns, total: 1.45 s
Wall time: 1h 17min 9s
```

In [63]:
```
%%time
#Finding the Optimal K for the final test set
optimal_K = int(k_range[mean_test_score.argmax()])

#Training the KNN model with optimal K
clf = KNeighborsClassifier(n_neighbors=optimal_K, algorithm='kd_tree', n_jobs=-1)

clf.fit(x_train_tfidf, y_train)
# Get predicted values for test data
pred_test = clf.predict(x_test_tfidf)
pred_train = clf.predict(x_train_tfidf)
pred_proba_train = clf.predict_proba(x_train_tfidf)[:,1]
pred_proba_test = clf.predict_proba(x_test_tfidf)[:,1]

fpr_train, tpr_train, thresholds_train = roc_curve(y_train, pred_proba_train, pos_label
fpr_test, tpr_test, thresholds_test = roc_curve(y_test, pred_proba_test, pos_label=1)
conf_mat_train = confusion_matrix(y_train, pred_train, labels=[0, 1])
conf_mat_test = confusion_matrix(y_test, pred_test, labels=[0, 1])
f1_sc = f1_score(y_test, pred_test, average='binary', pos_label=1)
auc_sc_train = auc(fpr_train, tpr_train)
auc_sc = auc(fpr_test, tpr_test)

print("Optimal K: {} with AUC: {:.2f}%".format(optimal_K, float(auc_sc*100)))
#Saving the report in a global variable
```

```python
result_report = result_report.append({'VECTORIZER': 'TF-IDF',
                                      'MODEL': 'KD-Tree',
                                      'HYPERPARAMETER': optimal_K,
                                      'F1_SCORE': f1_sc, 'AUC': auc_sc
                                     }, ignore_index=True)


plt.close()
plt.figure(figsize=(16,7))
# Plot ROC curve for training set
plt.subplot(2, 2, 1)
plt.title('Receiver Operating Characteristic - KNN-kdTree-TFIDF - TRAIN SET')
plt.plot(fpr_train, tpr_train, color='red', label='AUC - Train - {:.2f}'.format(float(a
plt.plot([0, 1], ls="--")
plt.plot([0, 0], [1, 0] , c=".7"), plt.plot([1, 1] , c=".7")
plt.ylabel('True Positive Rate')
plt.xlabel('False Positive Rate')
plt.grid()
plt.legend(loc='best')

# Plot ROC curve for test set
plt.subplot(2, 2, 2)
plt.title('Receiver Operating Characteristic - KNN-kdTree-TFIDF - TEST SET')
plt.plot(fpr_test, tpr_test, color='blue', label='AUC - Test - {:.2f}'.format(float(auc
plt.plot([0, 1], ls="--")
plt.plot([0, 0], [1, 0] , c=".7"), plt.plot([1, 1] , c=".7")
plt.ylabel('True Positive Rate')
plt.xlabel('False Positive Rate')
plt.grid()
plt.legend(loc='best')

#Plotting the confusion matrix for train
plt.subplot(2, 2, 3)
plt.title('Confusion Matrix for Training set')
df_cm = pd.DataFrame(conf_mat_train, index = ["Negative", "Positive"],
                  columns = ["Negative", "Positive"])
sns.heatmap(df_cm, annot=True,cmap='Blues', fmt='g')

#Plotting the confusion matrix for test
plt.subplot(2, 2, 4)
plt.title('Confusion Matrix for Testing set')
df_cm = pd.DataFrame(conf_mat_test, index = ["Negative", "Positive"],
                  columns = ["Negative", "Positive"])
sns.heatmap(df_cm, annot=True,cmap='Blues', fmt='g')

plt.tight_layout()
plt.show()
```

```
Optimal K: 3 with AUC: 57.56%
```




```
CPU times: user 20min 32s, sys: 0 ns, total: 20min 32s
Wall time: 2min 38s
```

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

In [64]:
```
list_of_sent_train = []
list_of_sent_test = []

for sent in x_train:
    list_of_sent_train.append(sent.split())
for sent in x_test:
    list_of_sent_test.append(sent.split())

w2v_model=Word2Vec(list_of_sent_train,min_count=5,size=50, workers=8)
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  7757
sample words  ['cuts', 'sons', 'goji', 'shipped', 'probiotics', 'foamy', 'qualify', 'chex', 'ful
```

In [65]:
```
# compute average word2vec for each review for train data
avgw2v_train = [] # the avg-w2v for each sentence/review is stored in this list
for sent in tqdm(list_of_sent_train): # for each review/sentence
    sent_vec = np.zeros(50)
    cnt_words = 0 # num of words with a valid vector in the sentence/review
```

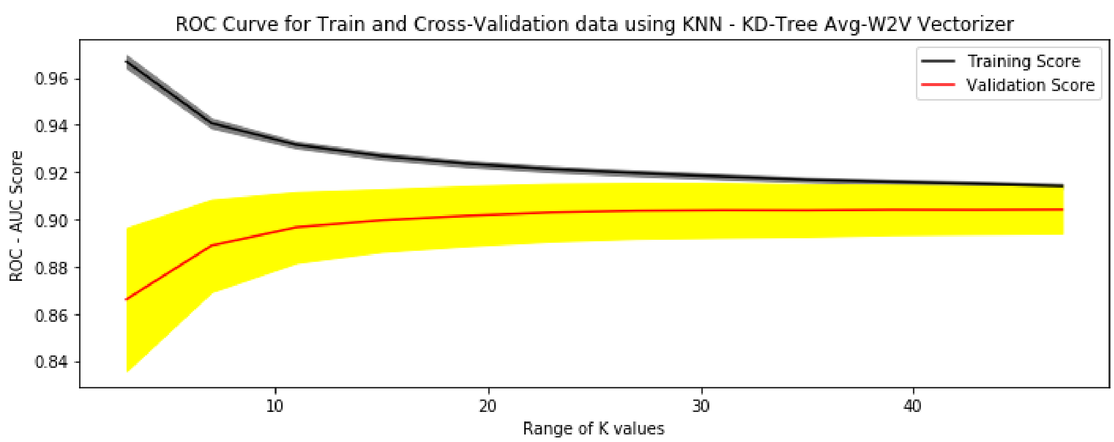
```
            for word in sent: # for each word in a review/sentence
                if word in w2v_words:
                    vec = w2v_model.wv[word]
                    sent_vec += vec
                    cnt_words += 1
            if cnt_words != 0:
                sent_vec /= cnt_words
            avgw2v_train.append(sent_vec)

        # compute average word2vec for each review for test data
        avgw2v_test = [] # the avg-w2v for each sentence/review is stored in this list
        for sent in tqdm(list_of_sent_test): # for each review/sentence
            sent_vec = np.zeros(50)
            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
            avgw2v_test.append(sent_vec)

        std_clf = StandardScaler(with_mean=False)
        avgw2v_train = std_clf.fit_transform(avgw2v_train)
        avgw2v_test = std_clf.transform(avgw2v_test)
```

```
100%|| 14000/14000 [01:21<00:00, 171.38it/s]
100%|| 6000/6000 [00:35<00:00, 170.38it/s]
```

In [66]:
```
sm = SMOTE(random_state=2)
avgw2v_train, y_train = sm.fit_sample(avgw2v_train, y_train_tmp)
```

In [67]:
```
%%time
# Calculate roc_auc on training set using range of parameter values
train_score, validation_score = validation_curve(estimator=KNeighborsClassifier(algorit
                                                    X=avgw2v_train, y=y_train,
                                                    param_name='n_neighbors',
                                                    param_range=k_range,
                                                    cv = 10, scoring='roc_auc', n_jobs=

#Calculate mean and standard deviation for training set scores
mean_train_score = np.mean(train_score, axis = 1)
std_train_score = np.std(train_score, axis = 1)

#Calculate mean and standard deviation for cv set scores
mean_test_score = np.mean(validation_score, axis = 1)
```

```
std_test_score = np.std(validation_score, axis = 1)

plt.figure(figsize=(10, 4))
#Plot mean accuracy for train and cv set scores
plt.plot(k_range, mean_train_score, label='Training Score', color='black')
plt.plot(k_range, mean_test_score, label='Validation Score', color='red')

#Plot accuracy bands for training and cv sets
plt.fill_between(k_range, mean_train_score - std_train_score, mean_train_score + std_tr
plt.fill_between(k_range, mean_test_score - std_test_score, mean_test_score + std_test_

# Create plot
plt.title("ROC Curve for Train and Cross-Validation data using KNN - KD-Tree Avg-W2V Ve
plt.xlabel("Range of K values")
plt.ylabel("ROC - AUC Score")
plt.tight_layout()
plt.legend(loc="best")
plt.show()
```



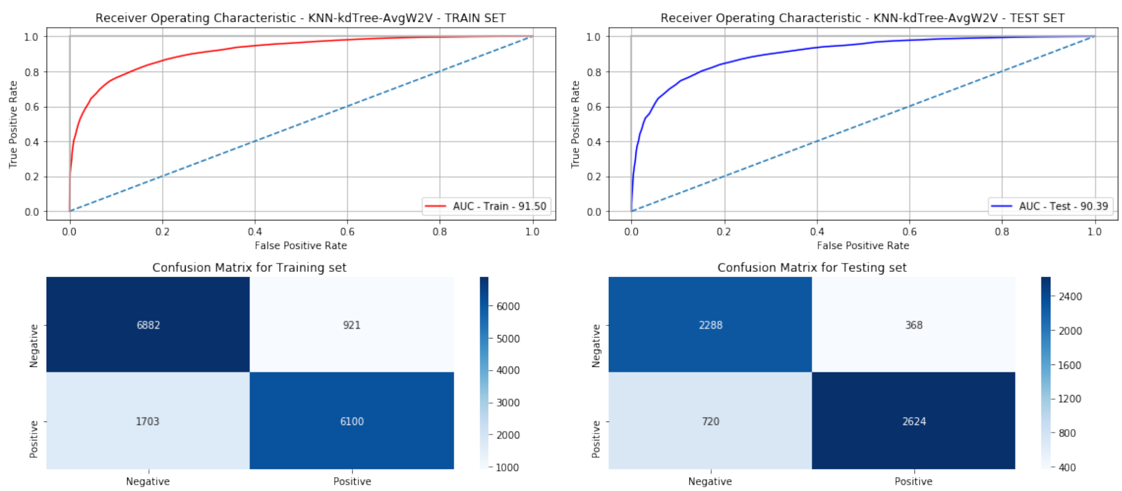

```
CPU times: user 1.17 s, sys: 692 ms, total: 1.86 s
Wall time: 13min
```

In [68]:
```
%%time
#Finding the Optimal K for the final test set
optimal_K = int(k_range[mean_test_score.argmax()])

#Training the KNN model with optimal K
clf = KNeighborsClassifier(n_neighbors=optimal_K, algorithm='kd_tree', n_jobs=-1)

clf.fit(avgw2v_train, y_train)
# Get predicted values for test data
```

```
pred_test = clf.predict(avgw2v_test)
pred_train = clf.predict(avgw2v_train)
pred_proba_train = clf.predict_proba(avgw2v_train)[:,1]
pred_proba_test = clf.predict_proba(avgw2v_test)[:,1]

fpr_train, tpr_train, thresholds_train = roc_curve(y_train, pred_proba_train, pos_label
fpr_test, tpr_test, thresholds_test = roc_curve(y_test, pred_proba_test, pos_label=1)
conf_mat_train = confusion_matrix(y_train, pred_train, labels=[0, 1])
conf_mat_test = confusion_matrix(y_test, pred_test, labels=[0, 1])
f1_sc = f1_score(y_test, pred_test, average='binary', pos_label=1)
auc_sc_train = auc(fpr_train, tpr_train)
auc_sc = auc(fpr_test, tpr_test)

print("Optimal K: {} with AUC: {:.2f}%".format(optimal_K, float(auc_sc*100)))
#Saving the report in a global variable
result_report = result_report.append({'VECTORIZER': 'Avg-W2V',
                                       'MODEL': 'KD-Tree',
                                       'HYPERPARAMETER': optimal_K,
                                       'F1_SCORE': f1_sc, 'AUC': auc_sc
                                      }, ignore_index=True)


plt.close()
plt.figure(figsize=(16,7))
# Plot ROC curve for training set
plt.subplot(2, 2, 1)
plt.title('Receiver Operating Characteristic - KNN-kdTree-AvgW2V - TRAIN SET')
plt.plot(fpr_train, tpr_train, color='red', label='AUC - Train - {:.2f}'.format(float(a
plt.plot([0, 1], ls="--")
plt.plot([0, 0], [1, 0] , c=".7"), plt.plot([1, 1] , c=".7")
plt.ylabel('True Positive Rate')
plt.xlabel('False Positive Rate')
plt.grid()
plt.legend(loc='best')

# Plot ROC curve for test set
plt.subplot(2, 2, 2)
plt.title('Receiver Operating Characteristic - KNN-kdTree-AvgW2V - TEST SET')
plt.plot(fpr_test, tpr_test, color='blue', label='AUC - Test - {:.2f}'.format(float(auc
plt.plot([0, 1], ls="--")
plt.plot([0, 0], [1, 0] , c=".7"), plt.plot([1, 1] , c=".7")
plt.ylabel('True Positive Rate')
plt.xlabel('False Positive Rate')
plt.grid()
plt.legend(loc='best')

#Plotting the confusion matrix for train
plt.subplot(2, 2, 3)
```

```
plt.title('Confusion Matrix for Training set')
df_cm = pd.DataFrame(conf_mat_train, index = ["Negative", "Positive"],
                  columns = ["Negative", "Positive"])
sns.heatmap(df_cm, annot=True,cmap='Blues', fmt='g')

#Plotting the confusion matrix for test
plt.subplot(2, 2, 4)
plt.title('Confusion Matrix for Testing set')
df_cm = pd.DataFrame(conf_mat_test, index = ["Negative", "Positive"],
                  columns = ["Negative", "Positive"])
sns.heatmap(df_cm, annot=True,cmap='Blues', fmt='g')

plt.tight_layout()
plt.show()
```

```
Optimal K: 47 with AUC: 90.39%
```




```
CPU times: user 3min 43s, sys: 304 ms, total: 3min 44s
Wall time: 29.8 s
```

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

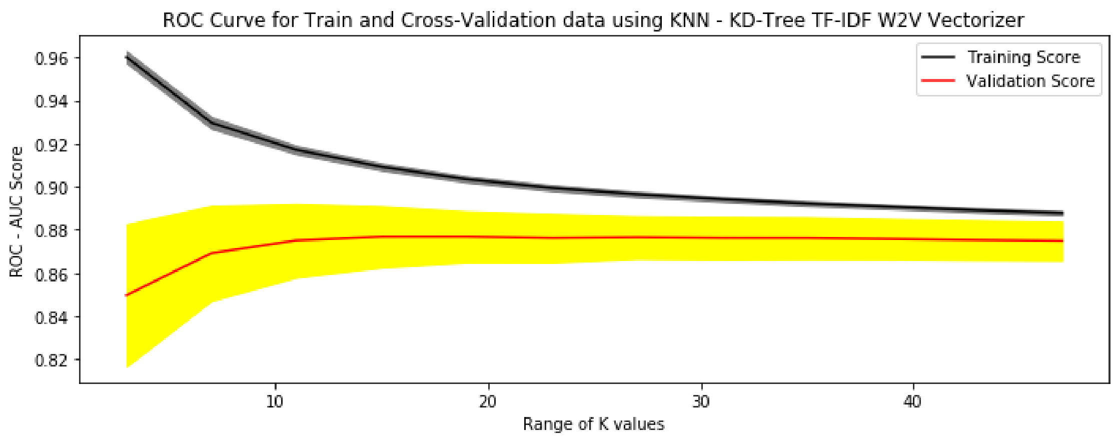
In [69]:
```
model = TfidfVectorizer()
model.fit(x_train)

#Creating the TFIDF W2V Training Set
# 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_)))
```

```python
# TF-IDF weighted Word2Vec
tfidf_feat = model.get_feature_names() # tfidf words/col-names
# final_tf_idf is the sparse matrix with row= sentence, col=word and cell_val = tfidf

tfidfw2v_train = []; # the tfidf-w2v for each sentence/review is stored in this list
row=0;
for sent in tqdm(list_of_sent_train): # for each review/sentence
    sent_vec = np.zeros(50) # as word vectors are of zero length
    weight_sum =0; # num of words with a valid vector in the sentence/review
    for word in sent: # for each word in a review/sentence
        if word in w2v_words and word in tfidf_feat:
            vec = w2v_model.wv[word]
            tf_idf = dictionary[word]*(sent.count(word)/len(sent))
            sent_vec += (vec * tf_idf)
            weight_sum += tf_idf
    if weight_sum != 0:
        sent_vec /= weight_sum
    tfidfw2v_train.append(sent_vec)
    row += 1

#Creating the TFIDF W2V Testing Set

# 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

tfidfw2v_test = []; # the tfidf-w2v for each sentence/review is stored in this list
row=0;
for sent in tqdm(list_of_sent_test): # for each review/sentence
    sent_vec = np.zeros(50) # as word vectors are of zero length
    weight_sum =0; # num of words with a valid vector in the sentence/review
    for word in sent: # for each word in a review/sentence
        if word in w2v_words and word in tfidf_feat:
            vec = w2v_model.wv[word]
            tf_idf = dictionary[word]*(sent.count(word)/len(sent))
            sent_vec += (vec * tf_idf)
            weight_sum += tf_idf
    if weight_sum != 0:
        sent_vec /= weight_sum
    tfidfw2v_test.append(sent_vec)
    row += 1

std_clf = StandardScaler(with_mean=False)
tfidfw2v_train = std_clf.fit_transform(tfidfw2v_train)
tfidfw2v_test = std_clf.transform(tfidfw2v_test)
```

```
100%|| 14000/14000 [04:59<00:00, 46.80it/s]
100%|| 6000/6000 [02:07<00:00, 60.33it/s]
```

In [70]:
```
sm = SMOTE(random_state=2)
tfidfw2v_train, y_train = sm.fit_sample(tfidfw2v_train, y_train_tmp)
```

In [71]:
```
%%time
# Calculate roc_auc on training set using range of parameter values
train_score, validation_score = validation_curve(estimator=KNeighborsClassifier(algorit
                                                    X=tfidfw2v_train, y=y_train,
                                                    param_name='n_neighbors',
                                                    param_range=k_range,
                                                    cv = 10, scoring='roc_auc', n_jobs=

#Calculate mean and standard deviation for training set scores
mean_train_score = np.mean(train_score, axis = 1)
std_train_score = np.std(train_score, axis = 1)

#Calculate mean and standard deviation for cv set scores
mean_test_score = np.mean(validation_score, axis = 1)
std_test_score = np.std(validation_score, axis = 1)

plt.figure(figsize=(10, 4))
#Plot mean accuracy for train and cv set scores
plt.plot(k_range, mean_train_score, label='Training Score', color='black')
plt.plot(k_range, mean_test_score, label='Validation Score', color='red')

#Plot accuracy bands for training and cv sets
plt.fill_between(k_range, mean_train_score - std_train_score, mean_train_score + std_tr
plt.fill_between(k_range, mean_test_score - std_test_score, mean_test_score + std_test_

# Create plot
plt.title("ROC Curve for Train and Cross-Validation data using KNN - KD-Tree TF-IDF W2V
plt.xlabel("Range of K values")
plt.ylabel("ROC - AUC Score")
plt.tight_layout()
plt.legend(loc="best")
plt.show()
```

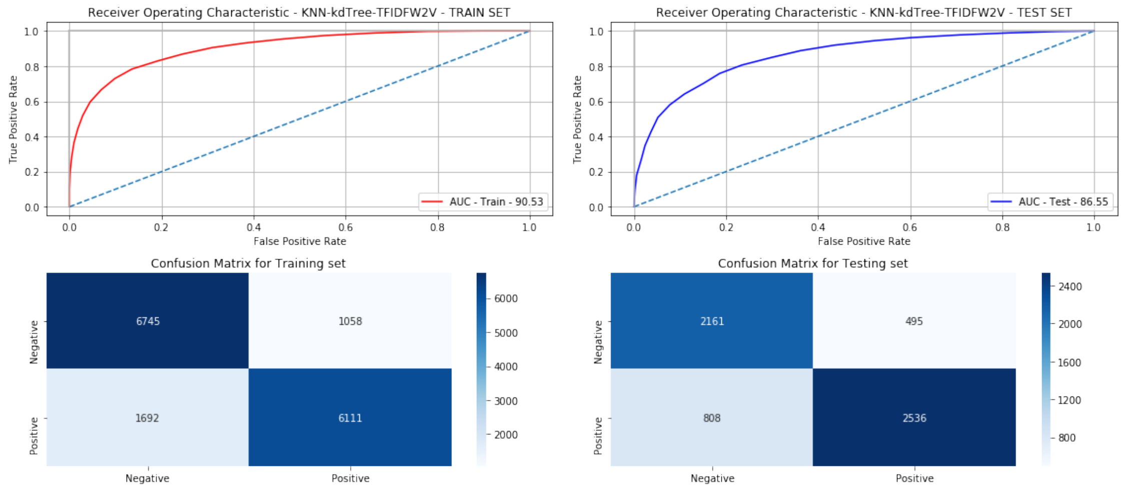



```
CPU times: user 1.16 s, sys: 956 ms, total: 2.12 s
Wall time: 11min 16s
```

In [72]:
```
%%time
#Finding the Optimal K for the final test set
optimal_K = int(k_range[mean_test_score.argmax()])

#Training the KNN model with optimal K
clf = KNeighborsClassifier(n_neighbors=optimal_K, algorithm='kd_tree', n_jobs=-1)

clf.fit(tfidfw2v_train, y_train)
# Get predicted values for test data
pred_test = clf.predict(tfidfw2v_test)
pred_train = clf.predict(tfidfw2v_train)
pred_proba_train = clf.predict_proba(tfidfw2v_train)[:,1]
pred_proba_test = clf.predict_proba(tfidfw2v_test)[:,1]

fpr_train, tpr_train, thresholds_train = roc_curve(y_train, pred_proba_train, pos_label
fpr_test, tpr_test, thresholds_test = roc_curve(y_test, pred_proba_test, pos_label=1)
conf_mat_train = confusion_matrix(y_train, pred_train, labels=[0, 1])
conf_mat_test = confusion_matrix(y_test, pred_test, labels=[0, 1])
f1_sc = f1_score(y_test, pred_test, average='binary', pos_label=1)
auc_sc_train = auc(fpr_train, tpr_train)
auc_sc = auc(fpr_test, tpr_test)

print("Optimal K: {} with AUC: {:.2f}%".format(optimal_K, float(auc_sc*100)))
#Saving the report in a global variable
result_report = result_report.append({'VECTORIZER': 'TF-IDF W2V',
                                      'MODEL': 'KD-Tree',
                                      'HYPERPARAMETER': optimal_K,
                                      'F1_SCORE': f1_sc, 'AUC': auc_sc
```

```
                                                    }, ignore_index=True)


        plt.close()
        plt.figure(figsize=(16,7))
        # Plot ROC curve for training set
        plt.subplot(2, 2, 1)
        plt.title('Receiver Operating Characteristic - KNN-kdTree-TFIDFW2V - TRAIN SET')
        plt.plot(fpr_train, tpr_train, color='red', label='AUC - Train - {:.2f}'.format(float(a
        plt.plot([0, 1], ls="--")
        plt.plot([0, 0], [1, 0] , c=".7"), plt.plot([1, 1] , c=".7")
        plt.ylabel('True Positive Rate')
        plt.xlabel('False Positive Rate')
        plt.grid()
        plt.legend(loc='best')

        # Plot ROC curve for test set
        plt.subplot(2, 2, 2)
        plt.title('Receiver Operating Characteristic - KNN-kdTree-TFIDFW2V - TEST SET')
        plt.plot(fpr_test, tpr_test, color='blue', label='AUC - Test - {:.2f}'.format(float(auc
        plt.plot([0, 1], ls="--")
        plt.plot([0, 0], [1, 0] , c=".7"), plt.plot([1, 1] , c=".7")
        plt.ylabel('True Positive Rate')
        plt.xlabel('False Positive Rate')
        plt.grid()
        plt.legend(loc='best')

        #Plotting the confusion matrix for train
        plt.subplot(2, 2, 3)
        plt.title('Confusion Matrix for Training set')
        df_cm = pd.DataFrame(conf_mat_train, index = ["Negative", "Positive"],
                          columns = ["Negative", "Positive"])
        sns.heatmap(df_cm, annot=True,cmap='Blues', fmt='g')

        #Plotting the confusion matrix for test
        plt.subplot(2, 2, 4)
        plt.title('Confusion Matrix for Testing set')
        df_cm = pd.DataFrame(conf_mat_test, index = ["Negative", "Positive"],
                          columns = ["Negative", "Positive"])
        sns.heatmap(df_cm, annot=True,cmap='Blues', fmt='g')

        plt.tight_layout()
        plt.show()
```

```
Optimal K: 19 with AUC: 86.55%
```




```
CPU times: user 2min 56s, sys: 384 ms, total: 2min 56s
Wall time: 24.1 s
```

7 [6] Conclusions

In [73]:
```
result_report
```

Out[73]:

	VECTORIZER	MODEL	HYPERPARAMETER	F1_SCORE	AUC
0	Bag Of Words(BOW)	Brute	7	0.636868	0.731969
1	TF-IDF	Brute	11	0.143639	0.654477
2	Avg-W2V	Brute	43	0.842721	0.927073
3	TF-IDF W2V	Brute	43	0.819294	0.898771
4	Bag Of Words(BOW)	KD-Tree	11	0.518758	0.733107
5	TF-IDF	KD-Tree	3	0.248379	0.575595
6	Avg-W2V	KD-Tree	47	0.828283	0.903861
7	TF-IDF W2V	KD-Tree	19	0.795608	0.865455