K-NN Amazon Fine Food Review

- Objective: To train a K-NN model using following vectorizers
  - Bag of Words
  - TF-IDF
  - Average Word2Vec
  - TF-IDF Weighted Word2Vec
- To check the performance and for finding any difference, for each vectorizer, k-NN will run in
  - Brute-Force approach
  - KD-Tree approach
- To select the best hyper parameter for k-NN, Simple Cross Validation approach has been used
- Totally 150K reviews are available
  - 90K has been used for Training
  - 30K used for Cross-Validation
  - 30K used for Testing the model
- It is a imbalanced dataset and the classification is a polarity classfication.
  - Since it is binary classification, both MSE and F1-Score has been calculated for checking model performance
  - Final hyper parameter selection has been made based on highest F1-score
- Euclidean Distance metric will be used to find the nearest neighbours in all the test cases
- Data Cleaning and DataSet has been done in a seperate Notebook - Datqa_Wrangling_AFF_Review.ipynb

In [1]:
```
from pathlib import Path # for file management
import numpy as np # for array handling
import pandas as pd # for handling tables
import os # for file handling
```

```
import sqlite3 # for database handling
from sklearn.neighbors import KNeighborsClassifier # for valiating my i
mplementation
import scipy
from sklearn.metrics import f1_score, mean_squared_error
from prettytable import PrettyTable # for pretty table
import time # for time measurement
```

In [2]:
```
# All the outputs generated by this notebook will be placed in a separa
te folder
data_dir = 'Output'
if not os.path.exists(data_dir):
    os.mkdir(data_dir)

def getFullFileNamePath(file_name):
    return str(Path.cwd() / data_dir / file_name)
```

In [3]:
```
# Max number of k that need to be tried
max_k = 52
k_range = list(range(3,max_k,2))
```

In [4]:
```
verbose = False
max_test_cases = 1000
limit_test_cases = False #True
```

In [5]:
```
# Names of files having vectors
file_names = {
    'BoW':
    {
        'train_label': 'Train_bow_label.npz', # Training Data
        'train_data' : 'Train_bow_svd.npz', # Training Label
        'cv_label'   : 'CV_bow_label.npz', # Cross-Validation Data
        'cv_data'    : 'CV_bow_svd.npz', # Cross-Validation Label
        'test_label' : 'Test_bow_label.npz', # Test Data
        'test_data'  : 'Test_bow_svd.npz', # Test Label
    },
    'tfidf':
    {
```

```
        'train_label': 'Train_tfidf_label.npz', # Training Data
        'train_data' : 'Train_tfid_svd.npz', # Training Label
        'cv_label'   : 'CV_tfidf_label.npz', # Cross-Validation Data
        'cv_data'    : 'CV_tfid_svd.npz', # Cross-Validation Label
        'test_label' : 'Test_tfidf_label.npz', # Test Data
        'test_data'  : 'Test_tfid_svd.npz', # Test Label
    },
    'avg_w2v':
    {
        'train_label': 'Train_avg_w2v_label.npz', # Training Data
        'train_data' : 'Train_avg_w2v.npy', # Training Label
        'cv_label'   : 'CV_avg_w2v_label.npz', # Cross-Validation Data
        'cv_data'    : 'CV_avg_w2v.npy', # Cross-Validation Label
        'test_label' : 'Test_avg_w2v_label.npz', # Test Data
        'test_data'  : 'Test_avg_w2v.npy', # Test Label
    },
    'tfidf_w_w2v':
    {
        'train_label': 'Train_tfidf_w_w2v_label.npz', # Training Data
        'train_data' : 'Train_tfidf_w_w2v.npy', # Training Label
        'cv_label'   : 'CV_tfidf_w_w2v_label.npz', # Cross-Validation D
ata
        'cv_data'    : 'CV_tfidf_w_w2v.npy', # Cross-Validation Label
        'test_label' : 'Test_tfidf_w_w2v_label.npz', # Test Data
        'test_data'  : 'Test_tfidf_w_w2v.npy', # Test Label
    }
}
```

In [6]:
```
# Append file path
for vec_type, files in file_names.items():
    for file_type, file_name in files.items():
        files[file_type] = getFullFileNamePath(file_name)
file_names
```

Out[6]:
```
{'BoW': {'train_label': 'D:\\Jupyter_Notebooks\\TTT\\Output\\Train_bow_
label.npz',
  'train_data': 'D:\\Jupyter_Notebooks\\TTT\\Output\\Train_bow_svd.np
z',
  'cv_label': 'D:\\Jupyter_Notebooks\\TTT\\Output\\CV_bow_label.npz',
```

```
  'cv_data': 'D:\\Jupyter_Notebooks\\TTT\\Output\\CV_bow_svd.npz',
  'test_label': 'D:\\Jupyter_Notebooks\\TTT\\Output\\Test_bow_label.np
z',
  'test_data': 'D:\\Jupyter_Notebooks\\TTT\\Output\\Test_bow_svd.npz'},
 'tfidf': {'train_label': 'D:\\Jupyter_Notebooks\\TTT\\Output\\Train_tf
idf_label.npz',
  'train_data': 'D:\\Jupyter_Notebooks\\TTT\\Output\\Train_tfid_svd.np
z',
  'cv_label': 'D:\\Jupyter_Notebooks\\TTT\\Output\\CV_tfidf_label.npz',
  'cv_data': 'D:\\Jupyter_Notebooks\\TTT\\Output\\CV_tfid_svd.npz',
  'test_label': 'D:\\Jupyter_Notebooks\\TTT\\Output\\Test_tfidf_label.n
pz',
  'test_data': 'D:\\Jupyter_Notebooks\\TTT\\Output\\Test_tfid_svd.np
z'},
 'avg_w2v': {'train_label': 'D:\\Jupyter_Notebooks\\TTT\\Output\\Train_
avg_w2v_label.npz',
  'train_data': 'D:\\Jupyter_Notebooks\\TTT\\Output\\Train_avg_w2v.np
y',
  'cv_label': 'D:\\Jupyter_Notebooks\\TTT\\Output\\CV_avg_w2v_label.np
z',
  'cv_data': 'D:\\Jupyter_Notebooks\\TTT\\Output\\CV_avg_w2v.npy',
  'test_label': 'D:\\Jupyter_Notebooks\\TTT\\Output\\Test_avg_w2v_labe
l.npz',
  'test_data': 'D:\\Jupyter_Notebooks\\TTT\\Output\\Test_avg_w2v.npy'},
 'tfidf_w_w2v': {'train_label': 'D:\\Jupyter_Notebooks\\TTT\\Output\\Tr
ain_tfidf_w_w2v_label.npz',
  'train_data': 'D:\\Jupyter_Notebooks\\TTT\\Output\\Train_tfidf_w_w2v.
npy',
  'cv_label': 'D:\\Jupyter_Notebooks\\TTT\\Output\\CV_tfidf_w_w2v_labe
l.npz',
  'cv_data': 'D:\\Jupyter_Notebooks\\TTT\\Output\\CV_tfidf_w_w2v.npy',
  'test_label': 'D:\\Jupyter_Notebooks\\TTT\\Output\\Test_tfidf_w_w2v_l
abel.npz',
  'test_data': 'D:\\Jupyter_Notebooks\\TTT\\Output\\Test_tfidf_w_w2v.np
y'}}
```

In [7]:
```
def getKNNPredictions(k_range,train_data,train_label,test_data,test_lab
el,alg_method):
    '''
```

```
    Function to run k-NN using required algorithm over the given test d
ata set
    Input:
        k_rnage - required list of k values which need to be tried
        train_data - Training Data Set
        train_label - Training Data Set's label
        test_data - Test Data Set, for which predictions has to be made
        test_label - Test Data SEt's label
    Ouptut:
        returns two lists having MSE and F1 scores for each k_value res
pectively
    '''
    f1_scores = []
    mse_scores = []

    '''
    For each requested k value
        1. Create a k-NN model using requested algorithm
        2. Fit Training Data to the model
        3. Make Predictions for Test Data
        4. Calculate MSE Score
        5. Calculate F1 Score
    '''
    for k_val in k_range:
        time_start = time.time()

        # Build the Model
        neigh = KNeighborsClassifier(n_neighbors=k_val,algorithm=alg_me
thod)
        neigh.fit(train_data, train_label)

        # Make Predictions over test dataset
        prediction = neigh.predict(test_data)

        # Evaluate the accuracy of the prediction
        # Using F1-Score
        mse_accuracy = mean_squared_error(test_label,prediction)
        if verbose == True: print('mse_accuracy: ', mse_accuracy)
        current_f1_score = f1_score(test_label, prediction)
```

```
        if verbose == True: print('f1_score: ', current_f1_score)

        mse_scores.append(mse_accuracy)
        f1_scores.append(current_f1_score)

        time_elapsed = time.time() - time_start
        print('k = {0} ==> Time elapsed: {1} seconds'.format(k_val, tim
e.time() - time_start))

    return mse_scores, f1_scores
```

In [8]:

```
def printResults(k_range, mse_scores,f1_scores):
    '''
    Helper function to print MSE and F1 score in a pretty way
    '''
    results = PrettyTable()
    results.field_names = ["K", "MSE", "F1"]

    max_f1_score = float('-inf')
    max_f1_k = 0

    min_mse_score = float('+inf')
    min_mse_k = 0

    for k_val, mse_val, f1_val in zip(k_range, mse_scores, f1_scores):
        results.add_row([k_val, mse_val, f1_val])

        if f1_val > max_f1_score:
            max_f1_score = f1_val
            max_f1_k = k_val

        if mse_val < min_mse_score:
            min_mse_score = mse_val
            min_mse_k = k_val

    print(results)
    print('F1: k - {0}, f1-score - {1} '.format(max_f1_k, max_f1_score
))
```

```
    print('MSE: k - {0}, mse-score - {1} '.format(min_mse_k, min_mse_sc
ore))
```

Hyper Parameter Selection

Vectorizer - Bag of Words

In [9]:
```
# Load BoW Training and CV Dataset for Hyper parameter selection
org_train_label = scipy.sparse.load_npz(file_names['BoW']['train_label'
]).todense()
org_train_data = scipy.sparse.load_npz(file_names['BoW']['train_data'])
.todense()
#org_train_data = np.load(file_names['BoW']['train_data'])

org_cv_label = scipy.sparse.load_npz(file_names['BoW']['cv_label']).tod
ense()
org_cv_data = scipy.sparse.load_npz(file_names['BoW']['cv_data']).toden
se()
#org_cv_data = np.load(file_names['BoW']['cv_data'])

train_data = org_train_data
train_label = np.array(org_train_label.T).ravel()

cv_data = org_cv_data
cv_label = np.array(org_cv_label.T).ravel()

if limit_test_cases == True:
    train_data = train_data[:max_test_cases]
    train_label = train_label[:max_test_cases]

    cv_data = cv_data[:max_test_cases]
    cv_label = cv_label[:max_test_cases]

print(type(train_label), train_label.shape, type(train_data), train_dat
a.shape)
print(type(cv_label), cv_label.shape, type(cv_data), cv_data.shape)
```

```
<class 'numpy.ndarray'> (90000,) <class 'numpy.matrixlib.defmatrix.matr
ix'> (90000, 1537)
<class 'numpy.ndarray'> (30000,) <class 'numpy.matrixlib.defmatrix.matr
ix'> (30000, 1537)
```

In [10]:
```
# Brute-Force approach
k_range = list(range(3,max_k,2))
bow_brute_mse_scores, bow_brute_f1_scores = \
    getKNNPredictions(k_range,train_data, train_label, cv_data, cv_labe
l, 'brute')
printResults(k_range,bow_brute_mse_scores, bow_brute_f1_scores)
```

```
k = 3 ==> Time elapsed: 115.0858805179596 seconds
k = 5 ==> Time elapsed: 133.31354236602783 seconds
k = 7 ==> Time elapsed: 123.99845385551453 seconds
k = 9 ==> Time elapsed: 122.85356736183167 seconds
k = 11 ==> Time elapsed: 118.81225943565369 seconds
k = 13 ==> Time elapsed: 120.17365598678589 seconds
k = 15 ==> Time elapsed: 118.996826171875 seconds
k = 17 ==> Time elapsed: 118.2817747592926 seconds
k = 19 ==> Time elapsed: 117.05101704597473 seconds
k = 21 ==> Time elapsed: 118.58693099021912 seconds
k = 23 ==> Time elapsed: 119.97916865348816 seconds
k = 25 ==> Time elapsed: 123.96663236618042 seconds
k = 27 ==> Time elapsed: 116.06859254837036 seconds
k = 29 ==> Time elapsed: 117.25644946098328 seconds
k = 31 ==> Time elapsed: 119.48455452919006 seconds
k = 33 ==> Time elapsed: 117.00515270233154 seconds
k = 35 ==> Time elapsed: 117.17868995666504 seconds
k = 37 ==> Time elapsed: 124.01041984558105 seconds
k = 39 ==> Time elapsed: 124.61580228805542 seconds
k = 41 ==> Time elapsed: 125.04964470863342 seconds
k = 43 ==> Time elapsed: 121.98384118080139 seconds
k = 45 ==> Time elapsed: 125.68394446372986 seconds
k = 47 ==> Time elapsed: 116.3668258190155 seconds
k = 49 ==> Time elapsed: 120.05802202224731 seconds
k = 51 ==> Time elapsed: 119.44163513183594 seconds
+----+---------------------+--------------------+
| K  |         MSE         |         F1         |
```

```
+----+---------------------+--------------------+
| 3  | 0.16673333333333334 | 0.905661800761948  |
| 5  | 0.15716666666666668 | 0.9123491904151098 |
| 7  | 0.15503333333333333 | 0.9141137148449763 |
| 9  |        0.1558       | 0.9140366365040828 |
| 11 | 0.15643333333333334 | 0.9139009668482947 |
| 13 | 0.15703333333333333 | 0.9137448047311276 |
| 15 | 0.15763333333333332 | 0.9135039233259561 |
| 17 | 0.15796666666666667 | 0.9134002156314529 |
| 19 | 0.15863333333333332 | 0.9131141255728188 |
| 21 |        0.159        | 0.9129625574775564 |
| 23 |        0.1595       | 0.9127509436026475 |
| 25 | 0.15996666666666667 | 0.9125275686710533 |
| 27 | 0.16023333333333334 | 0.912400911161731  |
| 29 | 0.16056666666666666 | 0.9122346724970393 |
| 31 |        0.1612       | 0.9119254024914402 |
| 33 | 0.16133333333333333 | 0.9118621845066831 |
| 35 | 0.16163333333333332 | 0.9117320469645945 |
| 37 |        0.1617       | 0.9116988550521505 |
| 39 | 0.16216666666666665 | 0.9114633569309724 |
| 41 |        0.1622       | 0.9114757677194004 |
| 43 |        0.1624       | 0.9113762869720231 |
| 45 |        0.1625       | 0.9113265547410736 |
| 47 | 0.16276666666666667 | 0.9111939619896335 |
| 49 | 0.16273333333333334 | 0.9112073042086501 |
| 51 | 0.16263333333333332 | 0.9112602535421328 |
+----+---------------------+--------------------+
F1: k - 7, f1-score - 0.9141137148449763
MSE: k - 7, mse-score - 0.15503333333333333
```

In [11]:

```
# KD-Tree approach
k_range = list(range(3,11,2))
bow_kdtree_mse_scores, bow_kdtree_f1_scores = \
    getKNNPredictions(k_range,train_data, train_label, cv_data, cv_labe
l, 'kd_tree')
printResults(k_range,bow_kdtree_mse_scores, bow_kdtree_f1_scores)
```

```
k = 3 ==> Time elapsed: 6538.84246134758 seconds
k = 5 ==> Time elapsed: 6548.847714185715 seconds
```

```
k = 7 ==> Time elapsed: 6556.653838157654 seconds
k = 9 ==> Time elapsed: 6565.559769392014 seconds
+---+--------------------+--------------------+
| K |        MSE         |         F1         |
+---+--------------------+--------------------+
| 3 | 0.16673333333333334 | 0.905661800761948  |
| 5 | 0.15716666666666668 | 0.9123491904151098 |
| 7 | 0.15503333333333333 | 0.9141137148449763 |
| 9 |       0.1558       | 0.9140366365040828 |
+---+--------------------+--------------------+
F1: k - 7, f1-score - 0.9141137148449763
MSE: k - 7, mse-score - 0.15503333333333333
```

Observation

- When using Bag of Words Vectorizer, we see that model performs well with k=7

Vectorizer - TFIDF

In [9]:
```
# Load TF-IDF Training and CV Dataset for Hyper parameter selection
org_train_label = scipy.sparse.load_npz(file_names['tfidf']['train_label']).todense()
org_train_data = scipy.sparse.load_npz(file_names['tfidf']['train_data']).todense()

org_cv_label = scipy.sparse.load_npz(file_names['tfidf']['cv_label']).todense()
org_cv_data = scipy.sparse.load_npz(file_names['tfidf']['cv_data']).todense()

train_data = org_train_data
train_label = np.array(org_train_label.T).ravel()

cv_data = org_cv_data
cv_label = np.array(org_cv_label.T).ravel()
```

```
if limit_test_cases == True:
    train_data = train_data[:max_test_cases]
    train_label = train_label[:max_test_cases]

    cv_data = cv_data[:max_test_cases]
    cv_label = cv_label[:max_test_cases]

print(type(train_label), train_label.shape, type(train_data), train_dat
a.shape)
print(type(cv_label), cv_label.shape, type(cv_data), cv_data.shape)
```

```
<class 'numpy.ndarray'> (90000,) <class 'numpy.matrixlib.defmatrix.matr
ix'> (90000, 1606)
<class 'numpy.ndarray'> (30000,) <class 'numpy.matrixlib.defmatrix.matr
ix'> (30000, 1606)
```

In [13]:
```
# Brute-Force approach
k_range = list(range(3,max_k,2))
tfidf_brute_mse_scores, tfidf_brute_f1_scores = \
    getKNNPredictions(k_range,train_data, train_label, cv_data, cv_labe
l, 'brute')
printResults(k_range,tfidf_brute_mse_scores, tfidf_brute_f1_scores)
```

```
k = 3 ==> Time elapsed: 102.56496548652649 seconds
k = 5 ==> Time elapsed: 117.7151312828064 seconds
k = 7 ==> Time elapsed: 119.3837685585022 seconds
k = 9 ==> Time elapsed: 120.3312497138977 seconds
k = 11 ==> Time elapsed: 117.68431258201599 seconds
k = 13 ==> Time elapsed: 114.6634361743927 seconds
k = 15 ==> Time elapsed: 115.44330430030823 seconds
k = 17 ==> Time elapsed: 122.4944748878479 seconds
k = 19 ==> Time elapsed: 120.31233954429626 seconds
k = 21 ==> Time elapsed: 114.75117254257202 seconds
k = 23 ==> Time elapsed: 118.76444053649902 seconds
k = 25 ==> Time elapsed: 120.68232297897339 seconds
k = 27 ==> Time elapsed: 116.83361291885376 seconds
k = 29 ==> Time elapsed: 116.87449789047241 seconds
k = 31 ==> Time elapsed: 126.26536154747009 seconds
k = 33 ==> Time elapsed: 123.93066716194153 seconds
k = 35 ==> Time elapsed: 119.1324622631073 seconds
```

```
k = 37 ==> Time elapsed: 119.9612500667572 seconds
k = 39 ==> Time elapsed: 119.50564241409302 seconds
k = 41 ==> Time elapsed: 117.94164848327637 seconds
k = 43 ==> Time elapsed: 117.86984300613403 seconds
k = 45 ==> Time elapsed: 121.96884346008301 seconds
k = 47 ==> Time elapsed: 120.19166588783264 seconds
k = 49 ==> Time elapsed: 122.24867677688599 seconds
k = 51 ==> Time elapsed: 119.00386810302734 seconds
+----+---------------------+--------------------+
| K  |         MSE         |         F1         |
+----+---------------------+--------------------+
| 3  | 0.16426666666666667 | 0.909760117194653  |
| 5  | 0.16206666666666666 | 0.9110859149263012 |
| 7  |        0.1577       | 0.9130921983209949 |
| 9  |        0.1544       | 0.9147651994700426 |
| 11 | 0.15243333333333334 | 0.9158307411974748 |
| 13 | 0.15243333333333334 | 0.9158400353350388 |
| 15 |        0.1517       | 0.9162479986749849 |
| 17 | 0.15136666666666668 | 0.9164597016023697 |
| 19 | 0.15163333333333334 | 0.9163771392856487 |
| 21 |        0.1516       | 0.9164308551688656 |
| 23 | 0.15156666666666666 | 0.9164722523283796 |
| 25 | 0.15153333333333333 | 0.9164952240999266 |
| 27 | 0.15143333333333334 | 0.9165733174180516 |
| 29 |        0.1518       |  0.91640355385858  |
| 31 | 0.15166666666666667 | 0.9165045693103828 |
| 33 | 0.15186666666666668 | 0.916397533763946  |
| 35 |        0.1518       | 0.9164771476780867 |
| 37 |        0.1519       | 0.9164389841386267 |
| 39 | 0.15223333333333333 | 0.9162678987221092 |
| 41 |        0.1526       | 0.916092375366569  |
| 43 |        0.1527       | 0.9160450838449556 |
| 45 | 0.15276666666666666 | 0.916026897777452  |
| 47 | 0.15296666666666667 | 0.9159169613573483 |
| 49 | 0.15316666666666667 | 0.9158131950678808 |
| 51 |        0.1533       | 0.9157522577808717 |
+----+---------------------+--------------------+
F1: k - 27, f1-score - 0.9165733174180516
MSE: k - 17, mse-score - 0.15136666666666668
```

In [10]:
```
# KD-Tree approach
k_range = list(range(3,30,2))
tfidf_kdtree_mse_scores, tfidf_kdtree_f1_scores = \
    getKNNPredictions(k_range,train_data, train_label, cv_data, cv_labe
l, 'kd_tree')
printResults(k_range,tfidf_kdtree_mse_scores, tfidf_kdtree_f1_scores)
```

```
k = 3 ==> Time elapsed: 6837.099238395691 seconds
k = 5 ==> Time elapsed: 6837.875531196594 seconds
k = 7 ==> Time elapsed: 6837.056284666061 seconds
k = 9 ==> Time elapsed: 6840.406259536743 seconds
k = 11 ==> Time elapsed: 6837.030170440674 seconds
k = 13 ==> Time elapsed: 6835.343750953674 seconds
k = 15 ==> Time elapsed: 6874.895222187042 seconds
k = 17 ==> Time elapsed: 6888.608699798584 seconds
k = 19 ==> Time elapsed: 6857.466369628906 seconds
k = 21 ==> Time elapsed: 6835.5521235466 seconds
k = 23 ==> Time elapsed: 6834.806116342545 seconds
k = 25 ==> Time elapsed: 6834.130923509598 seconds
k = 27 ==> Time elapsed: 6838.871550321579 seconds
k = 29 ==> Time elapsed: 6837.834372997284 seconds
+----+---------------------+--------------------+
| K  |         MSE         |         F1         |
+----+---------------------+--------------------+
| 3  | 0.16426666666666667 | 0.909760117194653  |
| 5  | 0.16206666666666666 | 0.9110859149263012 |
| 7  |        0.1577       | 0.9130921983209949 |
| 9  |        0.1544       | 0.9147651994700426 |
| 11 | 0.15243333333333334 | 0.9158307411974748 |
| 13 | 0.15243333333333334 | 0.9158400353350388 |
| 15 |        0.1517       | 0.9162479986749849 |
| 17 | 0.15136666666666668 | 0.9164597016023697 |
| 19 | 0.15163333333333334 | 0.9163771392856487 |
| 21 |        0.1516       | 0.9164308551688656 |
| 23 | 0.15156666666666666 | 0.9164722523283796 |
| 25 | 0.15153333333333333 | 0.9164952240999266 |
| 27 | 0.15143333333333334 | 0.9165733174180516 |
| 29 |        0.1518       |  0.91640355385858  |
+----+---------------------+--------------------+
```

```
F1: k - 27, f1-score - 0.9165733174180516
MSE: k - 17, mse-score - 0.15136666666666668
```

Observation

- When using TF-IDF Vectorizer, we see that model performs well with k=27

Vectorizer - Average Word2Vec

In [15]:
```
# Load Average Wrod2Vec Training and CV Dataset for Hyper parameter sel
ection

org_train_label = scipy.sparse.load_npz(file_names['avg_w2v']['train_la
bel']).todense()
#org_train_data = scipy.sparse.load_npz(file_names['avg_w2v']['train_da
ta']).todense()
org_train_data = np.load(file_names['avg_w2v']['train_data'])

org_cv_label = scipy.sparse.load_npz(file_names['avg_w2v']['cv_label'])
.todense()
#org_cv_data = scipy.sparse.load_npz(file_names['avg_w2v']['cv_data']).
todense()
org_cv_data = np.load(file_names['avg_w2v']['cv_data'])

train_data = org_train_data
train_label = np.array(org_train_label.T).ravel()

cv_data = org_cv_data
cv_label = np.array(org_cv_label.T).ravel()

if limit_test_cases == True:
    train_data = train_data[:max_test_cases]
    train_label = train_label[:max_test_cases]

    cv_data = cv_data[:max_test_cases]
    cv_label = cv_label[:max_test_cases]
```

```
print(type(train_label), train_label.shape, type(train_data), train_dat
a.shape)
print(type(cv_label), cv_label.shape, type(cv_data), cv_data.shape)
```

```
<class 'numpy.ndarray'> (90000,) <class 'numpy.ndarray'> (90000, 50)
<class 'numpy.ndarray'> (30000,) <class 'numpy.ndarray'> (30000, 50)
```

In [16]:
```
# Brute-Force approach
k_range = list(range(3,max_k,2))
avgw2v_brute_mse_scores, avgw2v_brute_f1_scores = \
    getKNNPredictions(k_range,train_data, train_label, cv_data, cv_labe
l, 'brute')
printResults(k_range,avgw2v_brute_mse_scores, avgw2v_brute_f1_scores)
```

```
k = 3 ==> Time elapsed: 42.81718158721924 seconds
k = 5 ==> Time elapsed: 57.568732500076294 seconds
k = 7 ==> Time elapsed: 57.53534007072449 seconds
k = 9 ==> Time elapsed: 57.62413692474365 seconds
k = 11 ==> Time elapsed: 57.644469261169434 seconds
k = 13 ==> Time elapsed: 57.536977767944336 seconds
k = 15 ==> Time elapsed: 57.618945837020874 seconds
k = 17 ==> Time elapsed: 57.53389024734497 seconds
k = 19 ==> Time elapsed: 57.70058727264404 seconds
k = 21 ==> Time elapsed: 57.67755365371704 seconds
k = 23 ==> Time elapsed: 57.67277956008911 seconds
k = 25 ==> Time elapsed: 57.61006426811218 seconds
k = 27 ==> Time elapsed: 57.806564807891846 seconds
k = 29 ==> Time elapsed: 57.804850339889526 seconds
k = 31 ==> Time elapsed: 57.7475700378418 seconds
k = 33 ==> Time elapsed: 57.73784589767456 seconds
k = 35 ==> Time elapsed: 57.73495149612427 seconds
k = 37 ==> Time elapsed: 57.45272493362427 seconds
k = 39 ==> Time elapsed: 57.60468792915344 seconds
k = 41 ==> Time elapsed: 57.66063356399536 seconds
k = 43 ==> Time elapsed: 57.64335322380066 seconds
k = 45 ==> Time elapsed: 59.16372013092041 seconds
k = 47 ==> Time elapsed: 57.75542664527893 seconds
k = 49 ==> Time elapsed: 60.13322114944458 seconds
k = 51 ==> Time elapsed: 57.981051445007324 seconds
```

```
+----+---------------------+--------------------+
| K  |         MSE         |         F1         |
+----+---------------------+--------------------+
| 3  | 0.14133333333333334 | 0.9189571466799189 |
| 5  | 0.13453333333333334 | 0.9234881516587676 |
| 7  | 0.13286666666666666 | 0.9247754208500037 |
| 9  | 0.13303333333333334 | 0.9248611503341806 |
| 11 | 0.13306666666666667 | 0.9250244158966269 |
| 13 | 0.13493333333333332 | 0.9241123317460912 |
| 15 | 0.13486666666666666 | 0.9242123403139401 |
| 17 | 0.13513333333333333 | 0.9241590900587422 |
| 19 |        0.1349       | 0.9243480699130759 |
| 21 |        0.1353       | 0.9242002651776877 |
| 23 | 0.13626666666666667 | 0.9236971778408243 |
| 25 | 0.13646666666666665 | 0.9236251026042832 |
| 27 |        0.137        | 0.9233695044188387 |
| 29 |        0.1368       | 0.9235184494968319 |
| 31 |        0.137        | 0.9234237591295275 |
| 33 | 0.13753333333333334 | 0.9231743194428927 |
| 35 |        0.1373       | 0.9233289280196564 |
| 37 |        0.1374       | 0.9232916480571683 |
| 39 | 0.13836666666666667 | 0.9227907668842884 |
| 41 | 0.13846666666666665 | 0.9227478985345533 |
| 43 | 0.13873333333333332 | 0.9226279000594885 |
| 45 | 0.13896666666666666 | 0.9225251342662282 |
| 47 |        0.1395       | 0.9222393578476001 |
| 49 | 0.13936666666666667 | 0.922322340919647  |
| 51 |        0.1393       | 0.9223681521799707 |
+----+---------------------+--------------------+
F1: k - 11, f1-score - 0.9250244158966269
MSE: k - 7, mse-score - 0.13286666666666666
```

In [17]:
```
# KD-Tree approach
avgw2v_kdtree_mse_scores, avgw2v_kdtree_f1_scores = \
    getKNNPredictions(k_range,train_data, train_label, cv_data, cv_labe
l, 'kd_tree')
printResults(k_range,avgw2v_kdtree_mse_scores, avgw2v_kdtree_f1_scores)
```

```
k = 3 ==> Time elapsed: 331.57648229599 seconds
```

```
k = 5 ==> Time elapsed: 338.0353455543518 seconds
k = 7 ==> Time elapsed: 347.8797388076782 seconds
k = 9 ==> Time elapsed: 350.1976001262665 seconds
k = 11 ==> Time elapsed: 353.5085380077362 seconds
k = 13 ==> Time elapsed: 357.42634987831116 seconds
k = 15 ==> Time elapsed: 357.1015679836273 seconds
k = 17 ==> Time elapsed: 358.84724402427673 seconds
k = 19 ==> Time elapsed: 359.8031542301178 seconds
k = 21 ==> Time elapsed: 356.2374334335327 seconds
k = 23 ==> Time elapsed: 359.9635663032532 seconds
k = 25 ==> Time elapsed: 366.4614279270172 seconds
k = 27 ==> Time elapsed: 365.66833448410034 seconds
k = 29 ==> Time elapsed: 367.2583975791931 seconds
k = 31 ==> Time elapsed: 368.0406000614166 seconds
k = 33 ==> Time elapsed: 369.22495555877686 seconds
k = 35 ==> Time elapsed: 368.1530952453613 seconds
k = 37 ==> Time elapsed: 370.61842703819275 seconds
k = 39 ==> Time elapsed: 370.97141003608704 seconds
k = 41 ==> Time elapsed: 374.75942730903625 seconds
k = 43 ==> Time elapsed: 367.9760568141937 seconds
k = 45 ==> Time elapsed: 373.26580238342285 seconds
k = 47 ==> Time elapsed: 372.45383644104004 seconds
k = 49 ==> Time elapsed: 373.2021975517273 seconds
k = 51 ==> Time elapsed: 374.9793543815613 seconds
+----+---------------------+--------------------+
| K  |         MSE         |         F1         |
+----+---------------------+--------------------+
| 3  | 0.14133333333333334 | 0.9189571466799189 |
| 5  | 0.13453333333333334 | 0.9234881516587676 |
| 7  | 0.13286666666666666 | 0.9247754208500037 |
| 9  | 0.13303333333333334 | 0.9248611503341806 |
| 11 | 0.13306666666666667 | 0.9250244158966269 |
| 13 | 0.13493333333333332 | 0.9241123317460912 |
| 15 | 0.13486666666666666 | 0.9242123403139401 |
| 17 | 0.13513333333333333 | 0.9241590900587422 |
| 19 |        0.1349       | 0.9243480699130759 |
| 21 |        0.1353       | 0.9242002651776877 |
| 23 | 0.13626666666666667 | 0.9236971778408243 |
| 25 | 0.13646666666666665 | 0.9236251026042832 |
```

```
| 27 |         0.137         | 0.9233695044188387 |
| 29 |         0.1368        | 0.9235184494968319 |
| 31 |         0.137         | 0.9234237591295275 |
| 33 | 0.13753333333333334 | 0.9231743194428927 |
| 35 |         0.1373        | 0.9233289280196564 |
| 37 |         0.1374        | 0.9232916480571683 |
| 39 | 0.13836666666666667 | 0.9227907668842884 |
| 41 | 0.13846666666666665 | 0.9227478985345533 |
| 43 | 0.13873333333333332 | 0.9226279000594885 |
| 45 | 0.13896666666666666 | 0.9225251342662282 |
| 47 |         0.1395        | 0.9222393578476001 |
| 49 | 0.13936666666666667 | 0.922322340919647  |
| 51 |         0.1393        | 0.9223681521799707 |
+----+-----------------------+--------------------+
F1: k - 11, f1-score - 0.9250244158966269
MSE: k - 7, mse-score - 0.13286666666666666
```

Observation

- When using Average Word2Vec Vectorizer, we see that model performs well with k=11

Vectorizer - TF-IDF Weighted Word2Vec

In [18]:
```
# Load TF-IDF Weighted Word2Vec Training and CV Dataset for Hyper param
eter selection
org_train_label = scipy.sparse.load_npz(file_names['tfidf_w_w2v']['trai
n_label']).todense()
#org_train_data = scipy.sparse.load_npz(file_names['tfidf_w_w2v']['trai
n_data']).todense()
org_train_data = np.load(file_names['tfidf_w_w2v']['train_data'])

org_cv_label = scipy.sparse.load_npz(file_names['tfidf_w_w2v']['cv_labe
l']).todense()
#org_cv_data = scipy.sparse.load_npz(file_names['tfidf_w_w2v']['cv_dat
a']).todense()
org_cv_data = np.load(file_names['tfidf_w_w2v']['cv_data'])
```

```
train_data = org_train_data
train_label = np.array(org_train_label.T).ravel()

cv_data = org_cv_data
cv_label = np.array(org_cv_label.T).ravel()

if limit_test_cases == True:
    train_data = train_data[:max_test_cases]
    train_label = train_label[:max_test_cases]

    cv_data = cv_data[:max_test_cases]
    cv_label = cv_label[:max_test_cases]

print(type(train_label), train_label.shape, type(train_data), train_dat
a.shape)
print(type(cv_label), cv_label.shape, type(cv_data), cv_data.shape)
```

```
<class 'numpy.ndarray'> (90000,) <class 'numpy.ndarray'> (90000, 50)
<class 'numpy.ndarray'> (30000,) <class 'numpy.ndarray'> (30000, 50)
```

In [19]:

```
# Brute-Force approach
k_range = list(range(3,max_k,2))
tfidfw2v_brute_mse_scores, tfidfw2v_brute_f1_scores = \
    getKNNPredictions(k_range,train_data, train_label, cv_data, cv_labe
l, 'brute')
printResults(k_range,tfidfw2v_brute_mse_scores, tfidfw2v_brute_f1_score
s)
```

```
k = 3 ==> Time elapsed: 42.61489796638489 seconds
k = 5 ==> Time elapsed: 57.6293420791626 seconds
k = 7 ==> Time elapsed: 57.89067506790161 seconds
k = 9 ==> Time elapsed: 59.90505647659302 seconds
k = 11 ==> Time elapsed: 59.500378370285034 seconds
k = 13 ==> Time elapsed: 57.4723060131073 seconds
k = 15 ==> Time elapsed: 58.43940019607544 seconds
k = 17 ==> Time elapsed: 59.93528771400452 seconds
k = 19 ==> Time elapsed: 59.94027042388916 seconds
k = 21 ==> Time elapsed: 58.03796339035034 seconds
k = 23 ==> Time elapsed: 57.571579933166504 seconds
```

```
k = 25 ==> Time elapsed: 58.343034744262695 seconds
k = 27 ==> Time elapsed: 58.4063720703125 seconds
k = 29 ==> Time elapsed: 57.97798299789429 seconds
k = 31 ==> Time elapsed: 57.78242897987366 seconds
k = 33 ==> Time elapsed: 57.71919798851013 seconds
k = 35 ==> Time elapsed: 57.44141507148743 seconds
k = 37 ==> Time elapsed: 57.58303213119507 seconds
k = 39 ==> Time elapsed: 57.31874179840088 seconds
k = 41 ==> Time elapsed: 57.377586126327515 seconds
k = 43 ==> Time elapsed: 57.36558222770691 seconds
k = 45 ==> Time elapsed: 57.40853834152222 seconds
k = 47 ==> Time elapsed: 57.25893473625183 seconds
k = 49 ==> Time elapsed: 57.264939069747925 seconds
k = 51 ==> Time elapsed: 57.87918829917908 seconds
+----+--------------------+--------------------+
| K  |         MSE        |         F1         |
+----+--------------------+--------------------+
| 3  |        0.1551      | 0.9113562324969995 |
| 5  |        0.1468      | 0.916971456581577  |
| 7  | 0.14323333333333332 | 0.9193793504568566 |
| 9  | 0.14293333333333333 | 0.9198234920160054 |
| 11 |        0.1428      | 0.9200268817204301 |
| 13 | 0.14316666666666666 | 0.9199754057125823 |
| 15 | 0.14366666666666666 | 0.919784105713754  |
| 17 | 0.14303333333333335 | 0.9202164252644888 |
| 19 |        0.1434      | 0.9200876769327934 |
| 21 | 0.14456666666666668 | 0.9195078042352591 |
| 23 |        0.145       | 0.9193249258160238 |
| 25 |        0.1453      | 0.9192014680531613 |
| 27 |        0.1455      | 0.9191142407115722 |
| 29 | 0.14593333333333333 | 0.9189079054604727 |
| 31 | 0.14593333333333333 | 0.9189379351207229 |
| 33 | 0.14586666666666667 | 0.9189959646070119 |
| 35 |        0.1459      | 0.9190179281762845 |
| 37 | 0.14616666666666667 | 0.9188789196189067 |
| 39 | 0.14596666666666666 | 0.9190019051846919 |
| 41 | 0.14636666666666667 | 0.9188129795691967 |
| 43 | 0.14606666666666668 | 0.918977886251017  |
| 45 | 0.14693333333333333 | 0.9185212569316081 |
```

```
| 47 |        0.1468       | 0.9186012124796687 |
| 49 | 0.14676666666666666 | 0.918633230462181  |
| 51 | 0.14723333333333333 | 0.9184227537168713 |
+----+---------------------+--------------------+
F1: k - 17, f1-score - 0.9202164252644888
MSE: k - 11, mse-score - 0.1428
```

In [20]:
```
# KD-Tree approach
tfidfw2v_kdtree_mse_scores, tfidfw2v_kdtree_f1_scores = \
    getKNNPredictions(k_range,train_data, train_label, cv_data, cv_label, 'kd_tree')
printResults(k_range,tfidfw2v_kdtree_mse_scores, tfidfw2v_kdtree_f1_scores)
```

```
k = 3 ==> Time elapsed: 264.00709652900696 seconds
k = 5 ==> Time elapsed: 274.75236773490906 seconds
k = 7 ==> Time elapsed: 281.3347682952881 seconds
k = 9 ==> Time elapsed: 286.7343304157257 seconds
k = 11 ==> Time elapsed: 291.325058221817 seconds
k = 13 ==> Time elapsed: 294.4866008758545 seconds
k = 15 ==> Time elapsed: 297.75087451934814 seconds
k = 17 ==> Time elapsed: 302.7136056423187 seconds
k = 19 ==> Time elapsed: 302.7565290927887 seconds
k = 21 ==> Time elapsed: 307.4818539619446 seconds
k = 23 ==> Time elapsed: 306.5892496109009 seconds
k = 25 ==> Time elapsed: 308.7075824737549 seconds
k = 27 ==> Time elapsed: 310.52973914146423 seconds
k = 29 ==> Time elapsed: 311.8262085914612 seconds
k = 31 ==> Time elapsed: 313.5366668701172 seconds
k = 33 ==> Time elapsed: 315.07156252861023 seconds
k = 35 ==> Time elapsed: 316.0010769367218 seconds
k = 37 ==> Time elapsed: 317.3734085559845 seconds
k = 39 ==> Time elapsed: 318.6041166782379 seconds
k = 41 ==> Time elapsed: 319.5336320400238 seconds
k = 43 ==> Time elapsed: 320.32352232933044 seconds
k = 45 ==> Time elapsed: 321.4285657405853 seconds
k = 47 ==> Time elapsed: 322.56851720809937 seconds
k = 49 ==> Time elapsed: 323.85906624794006 seconds
k = 51 ==> Time elapsed: 324.1103949546814 seconds
```

```
+----+---------------------+--------------------+
| K  |         MSE         |         F1         |
+----+---------------------+--------------------+
| 3  |        0.1551       | 0.9113562324969995 |
| 5  |        0.1468       | 0.916971456581577  |
| 7  | 0.14323333333333332 | 0.9193793504568566 |
| 9  | 0.14293333333333333 | 0.9198234920160054 |
| 11 |        0.1428       | 0.9200268817204301 |
| 13 | 0.14316666666666666 | 0.9199754057125823 |
| 15 | 0.14366666666666666 | 0.919784105713754  |
| 17 | 0.14303333333333335 | 0.9202164252644888 |
| 19 |        0.1434       | 0.9200876769327934 |
| 21 | 0.14456666666666668 | 0.9195078042352591 |
| 23 |        0.145        | 0.9193249258160238 |
| 25 |        0.1453       | 0.9192014680531613 |
| 27 |        0.1455       | 0.9191142407115722 |
| 29 | 0.14593333333333333 | 0.9189079054604727 |
| 31 | 0.14593333333333333 | 0.9189379351207229 |
| 33 | 0.14586666666666667 | 0.9189959646070119 |
| 35 |        0.1459       | 0.9190179281762845 |
| 37 | 0.14616666666666667 | 0.9188789196189067 |
| 39 | 0.14596666666666666 | 0.9190019051846919 |
| 41 | 0.14636666666666667 | 0.9188129795691967 |
| 43 | 0.14606666666666668 | 0.918977886251017  |
| 45 | 0.14693333333333333 | 0.9185212569316081 |
| 47 |        0.1468       | 0.9186012124796687 |
| 49 | 0.14676666666666666 | 0.918633230462181  |
| 51 | 0.14723333333333333 | 0.9184227537168713 |
+----+---------------------+--------------------+
F1: k - 17, f1-score - 0.9202164252644888
MSE: k - 11, mse-score - 0.1428
```

Observation

- When using TF-IDF Weighted Word2Vec Vectorizer, we see that model performs well with k=17

Test Result using selected Hyper-Parameter

In [33]:
```
final_results = PrettyTable()
final_results.field_names = ["Vectorizer", "Method", "Optimal K", "MSE"
, "F1"]
```

Bag of Words

In [9]:
```
org_train_label = scipy.sparse.load_npz(file_names['BoW']['train_label'
]).todense()
org_train_data = scipy.sparse.load_npz(file_names['BoW']['train_data'])
.todense()
#org_train_data = np.load(file_names['BoW']['train_data'])

org_test_label = scipy.sparse.load_npz(file_names['BoW']['test_label'])
.todense()
org_test_data = scipy.sparse.load_npz(file_names['BoW']['test_data']).t
odense()
#org_test_data = np.load(file_names['BoW']['test_data'])

print(type(org_train_label), org_train_label.shape, type(org_train_data
), org_train_data.shape)
print(type(org_test_label), org_test_label.shape, type(org_test_data),
org_test_data.shape)

train_data = org_train_data
train_label = np.array(org_train_label.T).ravel()

test_data = org_test_data
test_label = np.array(org_test_label.T).ravel()

if limit_test_cases == True:
    train_data = train_data[:max_test_cases]
    train_label = train_label[:max_test_cases]
```

```
    test_data = test_data[:max_test_cases]
    test_label = test_label[:max_test_cases]
```

```
<class 'numpy.matrixlib.defmatrix.matrix'> (1, 90000) <class 'numpy.mat
rixlib.defmatrix.matrix'> (90000, 1537)
<class 'numpy.matrixlib.defmatrix.matrix'> (1, 30000) <class 'numpy.mat
rixlib.defmatrix.matrix'> (30000, 1537)
```

In [10]:
```
k_range = [7]
bow_mse_scores, bow_f1_scores = \
    getKNNPredictions(k_range,train_data, train_label, test_data, test_
label, 'brute')
printResults(k_range,bow_mse_scores, bow_f1_scores)
```

```
k = 7 ==> Time elapsed: 112.56130075454712 seconds
+---+--------+--------------------+
| K |  MSE   |         F1         |
+---+--------+--------------------+
| 7 | 0.1586 | 0.9121037463976945 |
+---+--------+--------------------+
F1: k - 7, f1-score - 0.9121037463976945
MSE: k - 7, mse-score - 0.1586
```

In [34]:
```
final_results.add_row(['Bag Of Words', 'Brute', 7, bow_mse_scores[0], b
ow_f1_scores[0]])
```

TFIDF

In [11]:
```
org_train_label = scipy.sparse.load_npz(file_names['tfidf']['train_labe
l']).todense()
org_train_data = scipy.sparse.load_npz(file_names['tfidf']['train_data'
]).todense()

org_test_label = scipy.sparse.load_npz(file_names['tfidf']['test_label'
]).todense()
org_test_data = scipy.sparse.load_npz(file_names['tfidf']['test_data'])
.todense()
```

```
print(type(org_train_label), org_train_label.shape, type(org_train_data
), org_train_data.shape)
print(type(org_test_label), org_test_label.shape, type(org_test_data),
org_test_data.shape)

train_data = org_train_data
train_label = np.array(org_train_label.T).ravel()

test_data = org_test_data
test_label = np.array(org_test_label.T).ravel()

if limit_test_cases == True:
    train_data = train_data[:max_test_cases]
    train_label = train_label[:max_test_cases]

    test_data = test_data[:max_test_cases]
    test_label = test_label[:max_test_cases]
```

```
<class 'numpy.matrixlib.defmatrix.matrix'> (1, 90000) <class 'numpy.mat
rixlib.defmatrix.matrix'> (90000, 1606)
<class 'numpy.matrixlib.defmatrix.matrix'> (1, 30000) <class 'numpy.mat
rixlib.defmatrix.matrix'> (30000, 1606)
```

In [12]:

```
k_range = [27]
tfidf_mse_scores, tfidf_f1_scores = \
    getKNNPredictions(k_range,train_data, train_label, test_data, test_
label, 'brute')
printResults(k_range,tfidf_mse_scores, tfidf_f1_scores)
```

```
k = 27 ==> Time elapsed: 123.78212785720825 seconds
+----+--------+--------------------+
| K  |  MSE   |         F1         |
+----+--------+--------------------+
| 27 | 0.1535 | 0.9153912580153234 |
+----+--------+--------------------+
F1: k - 27, f1-score - 0.9153912580153234
MSE: k - 27, mse-score - 0.1535
```

In [35]:
```
final_results.add_row(['TF-IDF', 'Brute', 27, bow_mse_scores[0], bow_f1
_scores[0]])
```

Average Word2Vec

In [13]:
```
org_train_label = scipy.sparse.load_npz(file_names['avg_w2v']['train_la
bel']).todense()
#org_train_data = scipy.sparse.load_npz(file_names['avg_w2v']['train_da
ta']).todense()
org_train_data = np.load(file_names['avg_w2v']['train_data'])

org_test_label = scipy.sparse.load_npz(file_names['avg_w2v']['test_labe
l']).todense()
#org_test_data = scipy.sparse.load_npz(file_names['avg_w2v']['test_dat
a']).todense()
org_test_data = np.load(file_names['avg_w2v']['test_data'])

print(type(org_train_label), org_train_label.shape, type(org_train_data
), org_train_data.shape)
print(type(org_test_label), org_test_label.shape, type(org_test_data),
org_test_data.shape)

train_data = org_train_data
train_label = np.array(org_train_label.T).ravel()

test_data = org_test_data
test_label = np.array(org_test_label.T).ravel()

if limit_test_cases == True:
    train_data = train_data[:max_test_cases]
    train_label = train_label[:max_test_cases]

    test_data = test_data[:max_test_cases]
    test_label = test_label[:max_test_cases]
```

```
<class 'numpy.matrixlib.defmatrix.matrix'> (1, 90000) <class 'numpy.nda
rray'> (90000, 50)
```

```
<class 'numpy.matrixlib.defmatrix.matrix'> (1, 30000) <class 'numpy.nda
rray'> (30000, 50)
```

In [14]:
```
k_range = [11]
avgw2v_mse_scores, avgw2v_f1_scores = \
    getKNNPredictions(k_range,train_data, train_label, test_data, test_
label, 'brute')
printResults(k_range,avgw2v_mse_scores, avgw2v_f1_scores)
```

```
k = 11 ==> Time elapsed: 58.05729603767395 seconds
+----+---------------------+--------------------+
| K  |         MSE         |         F1         |
+----+---------------------+--------------------+
| 11 | 0.13583333333333333 | 0.9235129605645964 |
+----+---------------------+--------------------+
F1: k - 11, f1-score - 0.9235129605645964
MSE: k - 11, mse-score - 0.13583333333333333
```

In [36]:
```
final_results.add_row(['Average Word2Vec', 'Brute', 11, bow_mse_scores[
0], bow_f1_scores[0]])
```

TF-IDF Weighted Word2Vec

In [15]:
```
org_train_label = scipy.sparse.load_npz(file_names['tfidf_w_w2v']['trai
n_label']).todense()
#org_train_data = scipy.sparse.load_npz(file_names['tfidf_w_w2v']['trai
n_data']).todense()
org_train_data = np.load(file_names['tfidf_w_w2v']['train_data'])

org_test_label = scipy.sparse.load_npz(file_names['tfidf_w_w2v']['test_
label']).todense()
#org_test_data = scipy.sparse.load_npz(file_names['tfidf_w_w2v']['test_
data']).todense()
org_test_data = np.load(file_names['tfidf_w_w2v']['test_data'])

print(type(org_train_label), org_train_label.shape, type(org_train_data
), org_train_data.shape)
```

```
print(type(org_test_label), org_test_label.shape, type(org_test_data),
org_test_data.shape)

train_data = org_train_data
train_label = np.array(org_train_label.T).ravel()

test_data = org_test_data
test_label = np.array(org_test_label.T).ravel()

if limit_test_cases == True:
    train_data = train_data[:max_test_cases]
    train_label = train_label[:max_test_cases]

    test_data = test_data[:max_test_cases]
    test_label = test_label[:max_test_cases]
```

```
<class 'numpy.matrixlib.defmatrix.matrix'> (1, 90000) <class 'numpy.nda
rray'> (90000, 50)
<class 'numpy.matrixlib.defmatrix.matrix'> (1, 30000) <class 'numpy.nda
rray'> (30000, 50)
```

In [16]:
```
k_range = [17]
tfidfw2v_mse_scores, tfidfw2v_f1_scores = \
    getKNNPredictions(k_range,train_data, train_label, test_data, test_
label, 'brute')
printResults(k_range,tfidfw2v_mse_scores, tfidfw2v_f1_scores)
```

```
k = 17 ==> Time elapsed: 57.719196796417236 seconds
+----+---------------------+--------------------+
| K  |         MSE         |         F1         |
+----+---------------------+--------------------+
| 17 | 0.14563333333333334 | 0.918723839642824 |
+----+---------------------+--------------------+
F1: k - 17, f1-score - 0.918723839642824
MSE: k - 17, mse-score - 0.14563333333333334
```

In [37]:
```
final_results.add_row(['TF-IDF Weighted Word2Vec', 'Brute', 17, bow_mse
_scores[0], bow_f1_scores[0]])
```

Final Observation

In [38]:
```
print(final_results)
```

```
+--------------------------+--------+-----------+--------+-------------
-------+
|        Vectorizer        | Method | Optimal K |  MSE   |         F1
       |
+--------------------------+--------+-----------+--------+-------------
-------+
|       Bag Of Words       | Brute  |     7     | 0.1586 | 0.9121037463
976945 |
|          TF-IDF          | Brute  |     27    | 0.1586 | 0.9121037463
976945 |
|     Average Word2Vec     | Brute  |     11    | 0.1586 | 0.9121037463
976945 |
| TF-IDF Weighted Word2Vec | Brute  |     17    | 0.1586 | 0.9121037463
976945 |
+--------------------------+--------+-----------+--------+-------------
-------+
```

- Both Brute-Force and KD-Tree gives exactly same results
- Brute-Force approach is much faster than KD-Tree
  - When the dimension of the data is high, it is better to use Brute-Force approach