DATA MINING
TERM PROJECT

MINING OF DEXTER DATASET

M.Yasin SAĞLAM -15011804

WEKA PART

Dataset Analysis

Dexter is a text classification problem in a bag-of-word representation. This is a two-class classification problem with sparse continuous input variables. This dataset is one of five datasets of the NIPS 2003 feature selection challenge. The original data were formatted by Thorsten Joachims in the "bag-of-words" representation. There were 9947 features (of which 2562 are always zeros for all the examples) representing frequencies of occurrence of word stems in text. The task is to learn which Reuters articles are about 'corporate acquisitions' or not. Dataset labelled the related articles as 1 and non-related articles as -1. The dataset doesn't contain missing values and can be considered as sparse matrix form.

DEXTER DATASET	POSITIVE EX.	NEGATIVE EX.	TOTAL
TRAINING	150	150	300
VALIDATION	150	150	300
TEST	1000	1000	2000
TOTAL	1300	1300	2600

Table-1. Distribution of samples in Dexter Dataset

Dataset doesn't contain any outlier values. After feature Selection step on WEKA the new attribute number is 49 with class label. The screenshot of dataset condition after attribute selection step is showed in figure below.

```
Scheme:     weka.classifiers.bayes.NaiveBayes
Relation:   dexter-weka.filters.supervised.attribute.AttributeSelection-Eweka.attributeSelection.CfsSubsetEval -P 1 -E
1-Sweka.attributeSelection.BestFirst -D 1 -N 5
Instances:  600
Attributes: 49
V100, V246, V267, V625, V1051, V1243, V1564, V2061, V2633, V2780, V3432, V3880, V4307, V4382, V4575, V4636, V5127,
V5774, V6661, V6864, V6926, V7054, V7840, V8214, V8942, V10243, V10456, V10778, V11223, V12160, V12169, V12609,
V12915, V12934, V13684, V13928, V15293, V15797, V16487, V16583, V16816, V17016, V17470, V18159, V19171, V19326,
V19385, V19684
          class
Test mode:  split 50.0% train, remainder test
```

Classification

In given project, Dexter dataset is already splitted train and validation datas. Train datas used for training classifier and validation datas used to calculate classifier model accuracy. Then, most successful classifier models are determined. Most successfull 3 classifier models are liste below with accuracy metrics.

- Multilayer Perceptron - Accuracy : **94 %**
- SGD - Accuracy : **94 %**
- Random Forest - Accuracy **% 93**

Screenshots of Most 3 Successful Classifiers' WEKA Results

```
Time taken to build model: 0.06 seconds

=== Evaluation on test split ===

Time taken to test model on test split: 0 seconds

=== Summary ===

Correctly Classified Instances         282               94      %
Incorrectly Classified Instances        18                6      %
Kappa statistic                          0.88
Mean absolute error                      0.1618
Root mean squared error                  0.2316
Relative absolute error                 32.3485 %
Root relative squared error             46.2998 %
Total Number of Instances              300

=== Detailed Accuracy By Class ===

                 TP Rate  FP Rate  Precision  Recall   F-Measure  MCC      ROC Area  PRC Area  Class
                 0.921    0.041    0.959      0.921    0.940      0.881    0.982     0.987     -1
                 0.959    0.079    0.922      0.959    0.940      0.881    0.982     0.977     1
Weighted Avg.    0.940    0.059    0.941      0.940    0.940      0.881    0.982     0.982

=== Confusion Matrix ===

   a   b   <-- classified as
 140  12 |   a = -1
   6 142 |   b = 1
```

Figure-1. MultiLayer Perceptron Results

```
Time taken to build model: 0.08 seconds

=== Evaluation on test split ===

Time taken to test model on test split: 0 seconds

=== Summary ===

Correctly Classified Instances         282               94      %
Incorrectly Classified Instances        18                6      %
Kappa statistic                          0.8799
Mean absolute error                      0.06
Root mean squared error                  0.2449
Relative absolute error                 11.9979 %
Root relative squared error             48.9769 %
Total Number of Instances              300

=== Detailed Accuracy By Class ===

                 TP Rate  FP Rate  Precision  Recall   F-Measure  MCC      ROC Area  PRC Area  Class
                 0.954    0.074    0.929      0.954    0.942      0.880    0.940     0.910     -1
                 0.926    0.046    0.951      0.926    0.938      0.880    0.940     0.917     1
Weighted Avg.    0.940    0.060    0.940      0.940    0.940      0.880    0.940     0.914

=== Confusion Matrix ===

   a   b   <-- classified as
 145   7 |   a = -1
  11 137 |   b = 1
```

Figure-2. SGD Results

```
RandomForest

Bagging with 100 iterations and base learner

weka.classifiers.trees.RandomTree -K 0 -M 1.0 -V 0.001 -S 1 -do-not-check-capabilities

Time taken to build model: 0.32 seconds

=== Evaluation on test split ===

Time taken to test model on test split: 0.02 seconds

=== Summary ===

Correctly Classified Instances         279               93      %
Incorrectly Classified Instances        21                7      %
Kappa statistic                          0.8601
Mean absolute error                      0.1596
Root mean squared error                  0.2492
Relative absolute error                 31.9209 %
Root relative squared error             49.8216 %
Total Number of Instances              300

=== Detailed Accuracy By Class ===

                 TP Rate  FP Rate  Precision  Recall   F-Measure  MCC      ROC Area  PRC Area  Class
                 0.908    0.047    0.952      0.908    0.929      0.861    0.972     0.976     -1
                 0.953    0.092    0.910      0.953    0.931      0.861    0.972     0.963     1
Weighted Avg.    0.930    0.069    0.931      0.930    0.930      0.861    0.972     0.970

=== Confusion Matrix ===

   a   b   <-- classified as
 138  14 |   a = -1
   7 141 |   b = 1
```

Figure-3. Random Forest Results

Clustering

Clustering operation applied on all of train + validation datas. Then, most successful clusterer models are determined. Most successful 3 clusterer models are liste below with accuracy metrics.

- DensityBased(Canopy) - Incorrect rate : **12,1667 %**
- Canopy - Incorrect rate : **17 %**
- Simple EM (expectation maximisation) - Incorrect rate : **25 %**

Screenshots of Most 3 Successful Clusterers' WEKA Results

```
Time taken to build model (full training data) : 0.01 seconds

=== Model and evaluation on training set ===

Clustered Instances

0      261 ( 44%)
1      339 ( 56%)


Log likelihood: -210.54953


Class attribute: class
Classes to Clusters:

   0   1  <-- assigned to cluster
 244  56 | -1
  17 283 | 1

Cluster 0 <-- -1
Cluster 1 <-- 1

Incorrectly clustered instances :       73.0     12.1667 %
```

Figure-4. DensityBased Canopy Clusterer Results

```
Time taken to build model (full training data) : 0.01 seconds

=== Model and evaluation on training set ===

Clustered Instances

0      358 ( 60%)
1      242 ( 40%)


Class attribute: class
Classes to Clusters:

   0   1  <-- assigned to cluster
 278  22 | -1
  80 220 | 1

Cluster 0 <-- -1
Cluster 1 <-- 1

Incorrectly clustered instances :       102.0    17      %
```

Figure-5. Canopy Clusterer Results

```
Time taken to build model (full training data) : 0.13 seconds

=== Model and evaluation on training set ===

Clustered Instances

0      376 ( 63%)
1      224 ( 37%)


Log likelihood: -177.92456


Class attribute: class
Classes to Clusters:

   0   1  <-- assigned to cluster
 113 187 | -1
 263  37 | 1

Cluster 0 <-- 1
Cluster 1 <-- -1

Incorrectly clustered instances :       150.0    25      %
```

Figure-6. EM Clusterer Results

PYTHON IMPLEMENTATIONS

The second part of the given project K-NN classifier and K-Means clusterer implemented by using python 3 programming language. In addition, the Fully Connected Neural Network Classifier model implemented by using Keras has tensorflow backend on python 3 base.

Classification

The given dataset is already splitted train and validation datas. Train datas used for training classifier and validation datas used to calculate classifier model accuracy. In implemented Neural Network model normalized dataset user for network optimization and convergence.

The accuracy of k-NN algorithm depends with k-value figure shown below.

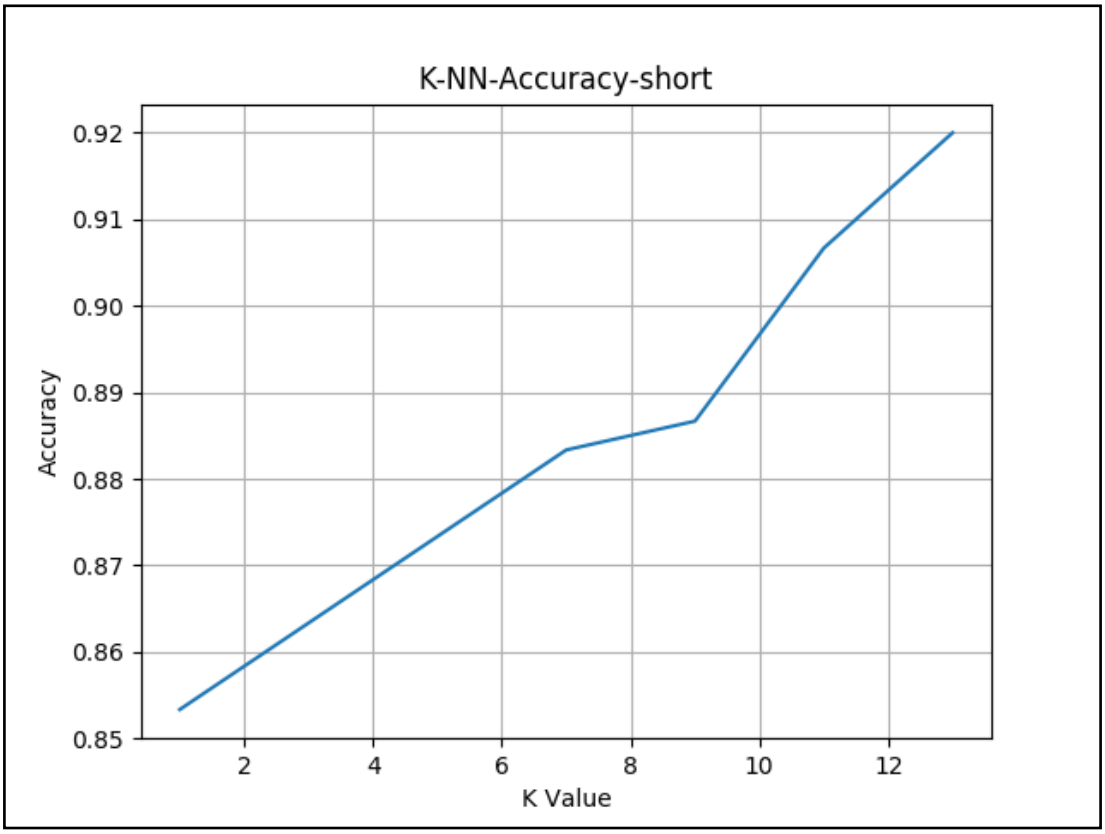

Figure-7. k-NN Python Implementation Results

The Designed Fully Connected Classifier Neural Network Model

```
model = Sequential()
# model.add(Input(shape=(20000),dtype=float))
model.add(Dense(32, input_dim=x_train.shape[1]))
model.add(Dense(64))
model.add(Dense(128))
model.add(Dense(y_train.shape[1], activation='softmax'))

# try using different optimizers and different optimizer configs
model.compile('adam', 'categorical_crossentropy', metrics=['accuracy'])

print('Train...')
model.fit(x_train,
          y_train,
          batch_size=16,
          epochs=8,
          validation_data=[x_test, y_test])

model.save("dexter_mlp_model.h5")
```

Figure-8. Designed FC Classifier Neural Network Model and Hyperparameters

```
Train...
Train on 300 samples, validate on 300 samples
Epoch 1/8
2018-12-16 14:54:59.127477: I tensorflow/core/platform/cpu_feature_guard.cc:140] Your CPU supports instructions that this Tenso

 16/300 [>.............................] - ETA: 3s - loss: 0.6933 - acc: 0.4375
176/300 [================>.............] - ETA: 0s - loss: 0.6965 - acc: 0.4716
300/300 [==============================] - 0s 1ms/step - loss: 0.6933 - acc: 0.5433 - val_loss: 0.6852 - val_acc: 0.8767
Epoch 2/8

 16/300 [>.............................] - ETA: 0s - loss: 0.6752 - acc: 1.0000
112/300 [==========>...................] - ETA: 0s - loss: 0.6695 - acc: 0.8571
224/300 [=====================>........] - ETA: 0s - loss: 0.6564 - acc: 0.8705
300/300 [==============================] - 0s 598us/step - loss: 0.6393 - acc: 0.8933 - val_loss: 0.6057 - val_acc: 0.9100
Epoch 3/8

 16/300 [>.............................] - ETA: 0s - loss: 0.5102 - acc: 1.0000
192/300 [==================>...........] - ETA: 0s - loss: 0.4111 - acc: 1.0000
300/300 [==============================] - 0s 436us/step - loss: 0.3443 - acc: 0.9900 - val_loss: 0.3284 - val_acc: 0.9500
Epoch 4/8

 16/300 [>.............................] - ETA: 0s - loss: 0.0920 - acc: 1.0000
192/300 [==================>...........] - ETA: 0s - loss: 0.0555 - acc: 1.0000
300/300 [==============================] - 0s 403us/step - loss: 0.0400 - acc: 1.0000 - val_loss: 0.1864 - val_acc: 0.9300
Epoch 5/8

 16/300 [>.............................] - ETA: 0s - loss: 0.0072 - acc: 1.0000
144/300 [=============>................] - ETA: 0s - loss: 0.0032 - acc: 1.0000
256/300 [========================>.....] - ETA: 0s - loss: 0.0026 - acc: 1.0000
300/300 [==============================] - 0s 570us/step - loss: 0.0027 - acc: 1.0000 - val_loss: 0.1611 - val_acc: 0.9567
Epoch 6/8

 16/300 [>.............................] - ETA: 0s - loss: 0.0017 - acc: 1.0000
192/300 [==================>...........] - ETA: 0s - loss: 0.0012 - acc: 1.0000
300/300 [==============================] - 0s 418us/step - loss: 0.0011 - acc: 1.0000 - val_loss: 0.1509 - val_acc: 0.9567
Epoch 7/8

 16/300 [>.............................] - ETA: 0s - loss: 7.8764e-04 - acc: 1.0000
144/300 [=============>................] - ETA: 0s - loss: 6.9755e-04 - acc: 1.0000
288/300 [===========================>..] - ETA: 0s - loss: 6.6743e-04 - acc: 1.0000
300/300 [==============================] - 0s 502us/step - loss: 6.8117e-04 - acc: 1.0000 - val_loss: 0.1489 - val_acc: 0.9567
Epoch 8/8

 16/300 [>.............................] - ETA: 0s - loss: 6.7997e-04 - acc: 1.0000
192/300 [==================>...........] - ETA: 0s - loss: 5.6081e-04 - acc: 1.0000
300/300 [==============================] - 0s 480us/step - loss: 5.4682e-04 - acc: 1.0000 - val_loss: 0.1476 - val_acc: 0.9567
```

Figure-9. Training Output of Designed FC Classifier Neural Network Model

```
****
TP :  145   FP :  8
FN :  5   TN :  142
****

Accuracy :  95.67  %
Precision :  94.77  %
Recall :  96.67  %
F1-Score :  95.71  %
```

Figure-10. Test Results of Designed FC Classifier Neural Network Model

Clustering

In this step of the project k-Means++ algorithm implemented. The results of implemented algorithm shown in the figure below.

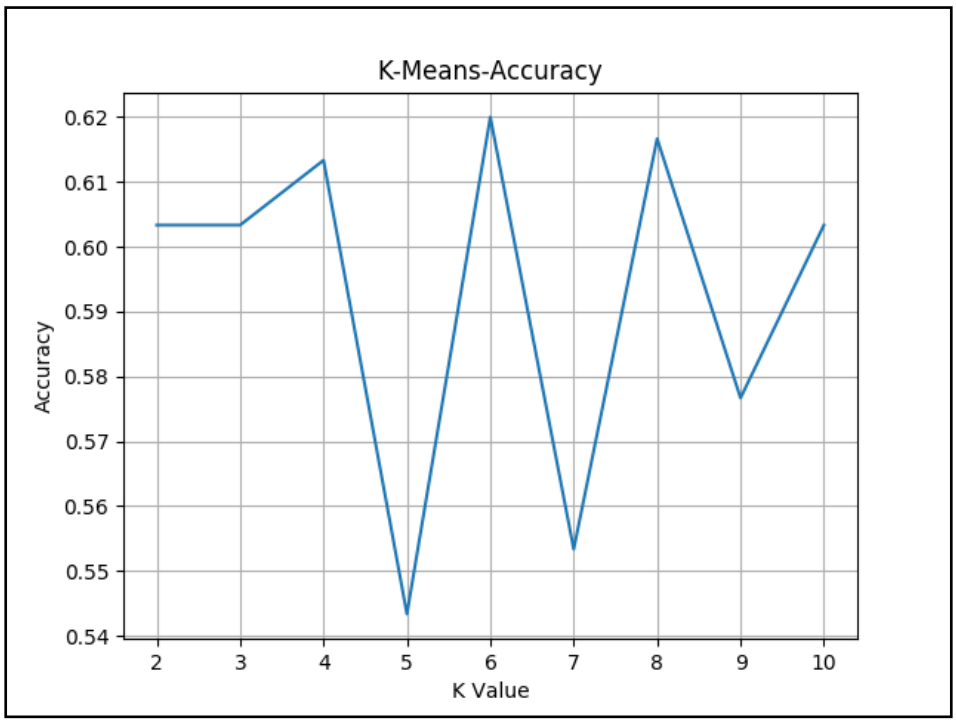

Figure-11. Results of Implemented K-Means++ Algorithm