BLM4800

INTRODUCTION TO DATA MINING

DOÇ. DR. SONGÜL VARLI ALBAYRAK

USER KNOWLEDGE MODELING
PROJECT REPORT

FURKAN GERÇEK

15011059

1-)WEKA

a-) Data Analysis

The User Knowledge Modelling Dataset was taken from UCI machine learning repository and was created by Hamdi Tolga Kahraman. Faculty of Technology ,Department of Software Engineering, Karadeniz Technical University, Trabzon, Turkiye.This Dataset contains 6 attributes(5 continuous input attributes and 1 class) and 249 samples.

The dataset is about students' knowledge status about the subject of Electrical DC Machines. The attributes of dataset are shown below.

STG (The degree of study time for goal object materails), (input value)
SCG (The degree of repetition number of user for goal object materails) (input value)
STR (The degree of study time of user for related objects with goal object) (input value)
LPR (The exam performance of user for related objects with goal object) (input value)
PEG (The exam performance of user for goal objects) (input value)
UNS (The knowledge level of user) (target value)

Sum of weights: 249 | Missing: 0 (0%)

No.	Label	Count
1	High	60
2	Low	82
3	Middle	83
4	very_low	24

There are no missing data in this dataset. The entropy of this dataset is calculated with "Information Gain Attribute Evaluation" method and the most important attribute is –as expected- found to be PEG. (Exam performance)

```
=== Attribute Selection on all input data ===

Search Method:
        Attribute ranking.

Attribute Evaluator (supervised, Class (nominal): 6 Class):
        Information Gain Ranking Filter

Ranked attributes:
 1.3369   5 PEG
 0.0846   4 LPR
 0.0596   1 STG
 0        2 SCG
 0        3 STR

Selected attributes: 5,4,1,2,3 : 5
```

b-) Classification

Classification is a data mining technique that helps analyze data and predict outcomes. The most sucessfull 3 classification methods in this dataset was SimpleLogistic, LMT and RandomForest.

1- Simple Logistic Regression

This was the best working algorithm for this dataset with a 94.37% success rate.

Binary logistic regression is used to predict the odds of being a case based on the values of the independent variables (predictors). The odds are defined as the probability that a particular outcome is a case divided by the probability that it is a noncase.

```
=== Summary ===

Correctly Classified Instances         235               94.3775 %
Incorrectly Classified Instances        14                5.6225 %
Kappa statistic                          0.9209
Mean absolute error                      0.0611
Root mean squared error                  0.1568
Relative absolute error                 17.1108 %
Root relative squared error             37.1265 %
Total Number of Instances              249

=== Detailed Accuracy By Class ===

                 TP Rate  FP Rate  Precision  Recall   F-Measure  MCC      ROC Area  PRC Area  Class
                 0.967    0.005    0.983      0.967    0.975      0.967    0.993     0.965     High
                 0.963    0.048    0.908      0.963    0.935      0.902    0.987     0.971     Low
                 0.940    0.018    0.963      0.940    0.951      0.927    0.981     0.970     Middle
                 0.833    0.009    0.909      0.833    0.870      0.857    0.994     0.950     very_low
Weighted Avg.    0.944    0.024    0.945      0.944    0.944      0.922    0.987     0.967

=== Confusion Matrix ===

  a  b  c  d   <-- classified as
 58  0  2  0 |  a = High
  0 79  1  2 |  b = Low
  1  4 78  0 |  c = Middle
  0  4  0 20 |  d = very_low
```

2- Logistic Model Tree

This method actually had the same success rate with Simple Logistic algorithm. LMT is a classification model with an associated suprervised training algorithm that combines logistic regression and decision tree learning.

```
=== Summary ===

Correctly Classified Instances         235               94.3775 %
Incorrectly Classified Instances        14                5.6225 %
Kappa statistic                          0.9209
Mean absolute error                      0.0611
Root mean squared error                  0.1568
Relative absolute error                 17.1108 %
Root relative squared error             37.1265 %
Total Number of Instances              249

=== Detailed Accuracy By Class ===

                 TP Rate  FP Rate  Precision  Recall   F-Measure  MCC      ROC Area  PRC Area  Class
                 0.967    0.005    0.983      0.967    0.975      0.967    0.993     0.965     High
                 0.963    0.048    0.908      0.963    0.935      0.902    0.987     0.971     Low
                 0.940    0.018    0.963      0.940    0.951      0.927    0.981     0.970     Middle
                 0.833    0.009    0.909      0.833    0.870      0.857    0.994     0.950     very_low
Weighted Avg.    0.944    0.024    0.945      0.944    0.944      0.922    0.987     0.967

=== Confusion Matrix ===

  a  b  c  d   <-- classified as
 58  0  2  0 |  a = High
  0 79  1  2 |  b = Low
  1  4 78  0 |  c = Middle
  0  4  0 20 |  d = very_low
```

3- Random Forest

The third best working algorithm for this dataset is Random Forest method.

Random Forests grows many classification trees. To classify a new object from an input vector, put the input vector down each of the trees in the forest. Each tree gives a classification, and the tree "votes" for that class. The forest chooses the classification having the most votes (over all the trees in the forest)

```
=== Summary ===

Correctly Classified Instances         233               93.5743 %
Incorrectly Classified Instances        16                6.4257 %
Kappa statistic                          0.9091
Mean absolute error                      0.0722
Root mean squared error                  0.1649
Relative absolute error                 20.2199 %
Root relative squared error             39.0591 %
Total Number of Instances              249

=== Detailed Accuracy By Class ===

                 TP Rate  FP Rate  Precision  Recall   F-Measure  MCC      ROC Area  PRC Area  Class
                 0.983    0.011    0.967      0.983    0.975      0.967    0.983     0.944     High
                 0.976    0.066    0.879      0.976    0.925      0.888    0.985     0.962     Low
                 0.928    0.018    0.963      0.928    0.945      0.918    0.978     0.948     Middle
                 0.708    0.000    1.000      0.708    0.829      0.829    0.993     0.957     very_low
Weighted Avg.    0.936    0.030    0.940      0.936    0.934      0.911    0.983     0.952

=== Confusion Matrix ===

  a  b  c  d   <-- classified as
 59  0  1  0 |  a = High
  0 80  2  0 |  b = Low
  2  4 77  0 |  c = Middle
  0  7  0 17 |  d = very_low
```

LMT

```
=== Confusion Matrix ===

  a  b  c  d   <-- classified as
 58  0  2  0 |  a = High
  0 79  1  2 |  b = Low
  1  4 78  0 |  c = Middle
  0  4  0 20 |  d = very_low
```

SimpleLogistic

```
=== Confusion Matrix ===

  a  b  c  d   <-- classified as
 58  0  2  0 |  a = High
  0 79  1  2 |  b = Low
  1  4 78  0 |  c = Middle
  0  4  0 20 |  d = very_low
```

RandomForest

```
=== Confusion Matrix ===

  a  b  c  d   <-- classified as
 59  0  1  0 |  a = High
  0 80  2  0 |  b = Low
  2  4 77  0 |  c = Middle
  0  7  0 17 |  d = very_low
```

c-) Clustering

Clustering is the grouping of a particular set of objects based on their characteristics, aggregating them according to their similarities.

1-) Average Hierarchical Clustering

The best results for this dataset were obtained with Average Hierarchical Clustering method. With 38.55% unsuccess rate.

In average linkage hierarchical clustering, the distance between two clusters is defined as the average distance between each point in one cluster to every point in the other cluster. For example, the distance between clusters “r” and “s” to the left is equal to the average length each arrow between connecting the points of one cluster to the other.

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

Clustered Instances

0         1 (  0%)
1       109 ( 44%)
2       109 ( 44%)
3        25 ( 10%)
4         5 (  2%)


Class attribute: Class
Classes to Clusters:

  0  1  2  3  4  <-- assigned to cluster
  0 33  1 21  5 | High
  0 11 70  1  0 | Low
  0 61 19  3  0 | Middle
  1  4 19  0  0 | very_low

Cluster 0 <-- very_low
Cluster 1 <-- Middle
Cluster 2 <-- Low
Cluster 3 <-- High
Cluster 4 <-- No class

Incorrectly clustered instances :       96.0     38.5542 %
```

2-) Expectation Maximisation

Next comes EM clustering with 44% error rate.

This method is an iterative method to find maximum likelihood estimates of parameters in statistical models. From what I've understood, it's very similiar to K-means but it doesn't do hard-assignment. EM computes the probability for clusters.

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

Clustered Instances

0       27 ( 11%)
1       75 ( 30%)
2        7 (  3%)
3       92 ( 37%)
4       48 ( 19%)


Log likelihood: 0.7988


Class attribute: Class
Classes to Clusters:

  0  1  2  3  4  <-- assigned to cluster
 25  0  0 35  0 | High
  0 50  0  0 32 | Low
  2  8  7 57  9 | Middle
  0 17  0  0  7 | very_low

Cluster 0 <-- High
Cluster 1 <-- Low
Cluster 2 <-- No class
Cluster 3 <-- Middle
Cluster 4 <-- very_low

Incorrectly clustered instances :       110.0    44.1767 %
```

3-) Cobweb

Then comes Cobweb Clustering with %55.1402 of the data being incorrectly clustered. Cobweb clustering is an incremental system for hierarchical conceptual clustering

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

Clustered Instances

0       62 ( 25%)
1       85 ( 34%)
2       61 ( 24%)
3       24 ( 10%)
4       17 (  7%)


Class attribute: Class
Classes to Clusters:

  0  1  2  3  4  <-- assigned to cluster
 34  7 10  0  9 | High
  4 48 15 15  0 | Low
 24 15 33  3  8 | Middle
  0 15  3  6  0 | very_low

Cluster 0 <-- High
Cluster 1 <-- Low
Cluster 2 <-- Middle
Cluster 3 <-- very_low
Cluster 4 <-- No class

Incorrectly clustered instances :       128.0    51.4056 %
```

2) Coding

I've used C for writing my implementation of methods which are K-NN and K-Means algorithms.

a) Classification: K-Nearest Neighbor

The code is written in Code::Blocks 17.12 IDE with GCC 5.1.0 Compiler.
None of predefined functions were used, the auxillary functions are written by me, which are :

- **calculateDistance**(TRAINING,TEST) : Calculates the Euclidian distance between 2 samples.

- **returnHighest**(int,int,int,int): Returns the variable with highest value. This is for counting the nearest neighbors' classes.

- **applyKNN**(TEST*,TRAINING[]): Applies KNN to a given test sample. Contains 3 parts: finding the nearest neighbors, counting up the nearest neighbors' classes, classifying the given test sample.

The algorithm I wrote uses 10-fold Cross Validation and Euclidian Distance but it only calculates for the first fold. Which are first 10% of the dataset. My KNN function does not iterate for entirety of the dataset.

Here's my results for 5NN :

```
Correctly classified percentage : 83.333336
```

Here's WEKA's results with same options:

```
Correctly Classified Instances         209               83.9357 %
```

As we can see, the results are very similiar.

b) Clustering : K-Means

The code is written in Code::Blocks 17.12 IDE with GCC 5.1.0 Compiler.
None of predefined functions were used, the auxillary functions are written by me, which are :

- **calculateDistance**(SAMPLE,SAMPLE) : Calculates the Euclidian distance between 2 samples.
- **findMinimumDistanceIndex**(SAMPLE) : Finds the closest point
- **newMeanPoints**(SAMPLE[], int, SAMPLE*,SAMPLE*,SAMPLE*,SAMPLE*): Finds the mean values after every iteration, chooses these new mean values as new points.

My algorithm chooses 4 random points over the dataset and all the points have distinct classes. The algorithm runs for the given iteration number.

Here's my results for K-Means with 100 Iterations

```
Correctly classified percentage : 48.594376
```

Here's WEKA's results for the same options

```
Incorrectly clustered instances :       125.0     50.2008 %
```

Although the results are similiar, my algorithm has some inconsistencies due to choosing points randomly. Sometimes, the randomized datas are chosen badly which often increases the accuracy.