Date:23-12-2023

Machine Learning in Drug Design

Semester Course Project

Instructor

MA'AM ISHRAT JABEEN

Submitted By

MEHWISH SHABIR

REGISTRATION NUMBER: 450341

Pre-process file

A. Open the WEKA Explorer and load the file **CDD_2D. arff** into WEKA.

a. Can you identify any attributes that should be removed from the relation prior to the classification process? Which attributes can you identify and why do think it is better to remove them?[1]

Attributes with a high percentage of zero values were removed. High occurrences of zeros indicate limited variability and reduced informativeness for the model, potentially introducing noise rather than contributing valuable insights.

Removing such attributes will streamline the model, improve computational efficiency, and reduce the risk of overfitting.

b. Write down the type of the attribute "class-label". Then click on the "Classify" tab, select "Percentage split" as test option and fill in "85%".[2] Click on "More options" and check "Output predictions" and "Preserve order for % Split".

Type of attribute class label is Nominal.

Percentage spilt means that 85% will be used for training the model and remaining 15% will be used for testing.

c. Choose trees/J48 as classifier (default options) and press "Start". What happens?

```
=== Summary ===

Correctly Classified Instances          81               79.4118 %
Incorrectly Classified Instances        21               20.5882 %
Kappa statistic                          0.4993
Mean absolute error                      0.2249
Root mean squared error                  0.4473
Relative absolute error                 54.4816 %
Root relative squared error             98.1673 %
Total Number of Instances              102

=== Detailed Accuracy By Class ===

                 TP Rate   FP Rate   Precision   Recall  F-Measure   MCC      ROC Area  PRC Area  Class
                 0.861     0.367     0.849       0.861   0.855       0.499    0.693     0.793     0
                 0.633     0.139     0.655       0.633   0.644       0.499    0.693     0.524     1
Weighted Avg.    0.794     0.300     0.792       0.794   0.793       0.499    0.693     0.714

=== Confusion Matrix ===

  a  b   <-- classified as
 62 10 |  a = 0
 11 19 |  b = 1
```

- Using the J48 decision tree classifier with default options on the dataset, the summary indicates that the model achieved an overall accuracy of approximately

79.41%.

- Out of 102 instances, 81 were correctly classified, while 21 were misclassified.
- The detailed accuracy by class shows that Class 0 has a higher True Positive Rate (Recall) and Precision compared to Class 1.
- The confusion matrix reveals that 62 instances of Class 0 were correctly classified, while 10 instances were misclassified.
- As for Class 1, 11 instances were correctly classified, with 19 instances misclassified as Class 0.

Algorithm 1 – *k*-Nearest Neighbor Classifier

A. Practical exercise: Apply the Weka IBk *k*-Nearest Neighbor classifier (weka.classifiers.lazy.IBk)

Run the classifier with different values of *k* (from 1 to 9). Which number of *k* achieves the best overall accuracy?

K	Accuracy
1	87.2549 %
2	81.3725 %
3	87.2549 %
4	84.3137 %
5	84.3137 %
6	82.3529 %
7	82.3529 %
8	84.3137 %
9	82.3529 %

Continue with that number of *k* that could achieve the best result (use the smallest number of *k* if various numbers of *k* achieve the same result).

Both k=1 and k=3 achieve the same overall accuracy of 87.2549%. I'll select k=3.

```
== Summary ===

orrectly Classified Instances          89               87.2549 %
ncorrectly Classified Instances        13               12.7451 %
appa statistic                          0.7018
ean absolute error                      0.1956
oot mean squared error                  0.3053
elative absolute error                 47.3769 %
oot relative squared error             67.0013 %
otal Number of Instances              102

== Detailed Accuracy By Class ===

               TP Rate  FP Rate  Precision  Recall   F-Measure  MCC      ROC Area  PRC Area  Class
               0.889    0.167    0.928      0.889    0.908      0.703    0.933     0.958     0
               0.833    0.111    0.758      0.833    0.794      0.703    0.933     0.801     1
eighted Avg.   0.873    0.150    0.878      0.873    0.874      0.703    0.933     0.912

== Confusion Matrix ===

 a  b   <-- classified as
64  8 |  a = 0
 5 25 |  b = 1
```

The IBk k-Nearest Neighbors classifier, with k=3 on a dataset with instances and 250 attributes, achieved an overall accuracy of 87.25%. The detailed accuracy by class showed high precision and recall for both classes (0 and 1). In the confusion matrix, 64 instances of class 0 were correctly classified, with 8 misclassifications, and 25 instances of class 1 were correctly classified, with 5 misclassifications.

The KNN classifier shows good overall accuracy and is showing good performance in distinguishing between the two classes. High precision and recall are showing that model is effective in making accurate predictions.

a. Change the distanceFunction of the selected "nearestNeighbourSearchAlgorithm". Click on "LinearNNSearch" and select "ManhattenDistance" instead of "EuclideanDistance". Are there differences in the achieved results?

```
=== Summary ===

Correctly Classified Instances          90               88.2353 %
Incorrectly Classified Instances        12               11.7647 %
Kappa statistic                          0.7167
Mean absolute error                      0.2005
Root mean squared error                  0.3128
Relative absolute error                 48.5628 %
Root relative squared error             68.6432 %
Total Number of Instances              102

=== Detailed Accuracy By Class ===

               TP Rate   FP Rate   Precision   Recall  F-Measure  MCC      ROC Area  PRC Area  Class
                 0.917    0.200     0.917      0.917     0.917     0.717     0.928     0.957     0
                 0.800    0.083     0.800      0.800     0.800     0.717     0.928     0.777     1
Weighted Avg.    0.882    0.166     0.882      0.882     0.882     0.717     0.928     0.904

=== Confusion Matrix ===

  a  b   <-- classified as
 66  6 |  a = 0
  6 24 |  b = 1
```

Above results show that the change in model from Euclidean to Manhattan distance in the Knearest had led to improve in model performance.

The model with Manhattan Distance (88.2353%) achieved a higher overall accuracy compared to the model with Euclidean Distance (87.2549%).

In Confusion Matrix the distribution of correctly and incorrectly classified instances changed, reflecting a better performance in distinguishing between classes.

- Euclidean Distance (k=3):

  a b <-- classified as

  64 5 | a = 0

  8 25 | b = 1

- Manhattan Distance (k=3):

  a b <-- classified as

  66 6 | a = 0

  6 24 | b = 1

The Manhattan Distance, in this case, resulted in a slightly better-performing model. Manhattan Distance with k=3 resulted in an overall improvement in model performance compared to

Euclidean Distance with the same k-value. The model demonstrated higher accuracy, a better agreement beyond chance, and improved class distinctions.

B. **Theoretical question:**

You don't need to explore all attributes, but take a look at the top and bottom of the attribute list, or use the "Edit" button to display a table.

The first 85% instances will now be used as training set, the remaining 15% as test set. a.

Describe the basic methodology of k-nearest neighbor algorithm! (in max. 3 sentences)

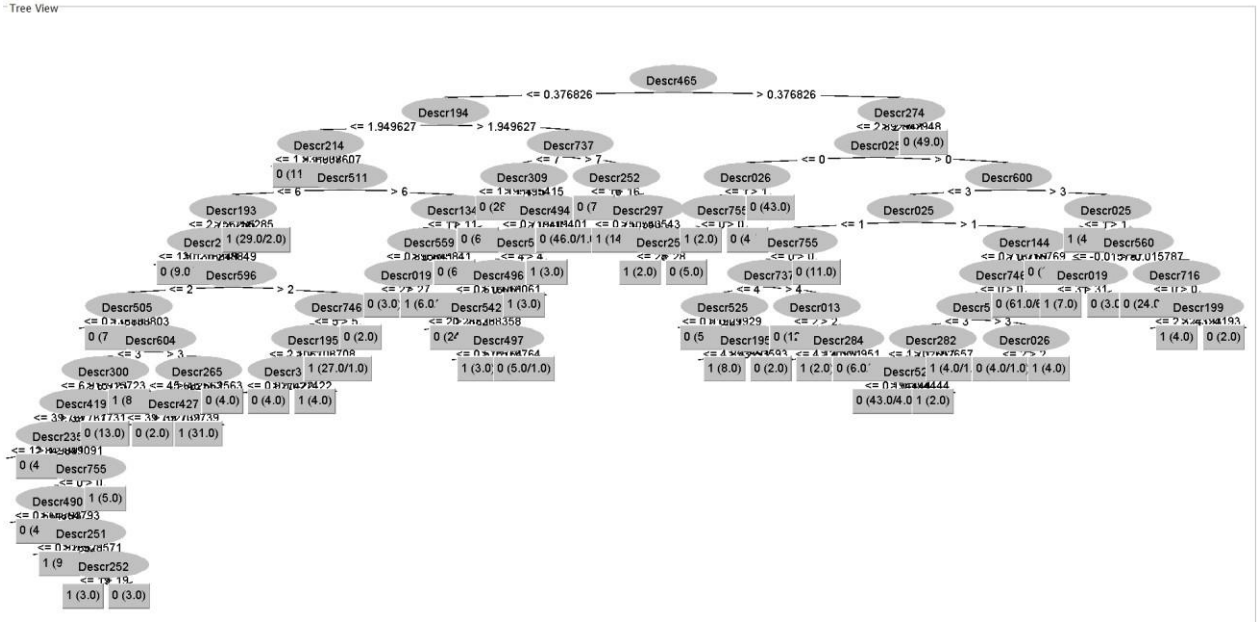
The K-Nearest Neighbors (KNN) algorithm involves selecting a value for K and a distance metric, calculating distances between a new data point and existing data points, identifying the K nearest neighbors, and making predictions based on the majority class (for classification). The algorithm relies on stored training data at the time of testing and involves optional feature normalization. Evaluation metrics assess model performance, and hyperparameters such as K can be tuned for optimization. KNN is straightforward, instance based, and useful for classification and regression tasks.

Algorithm 2 – Decision Tree

A. Practical exercise:

Apply the Weka J48 Decision Tree classifier (weka.classifiers.trees.J48)

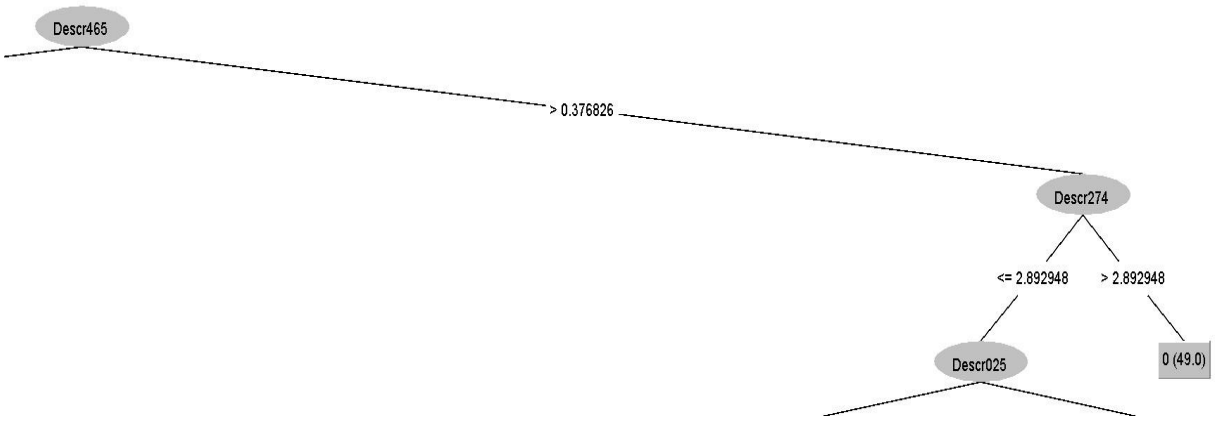
a. Run the classifier with default options and visualize the tree (right click the last entry in the "Result list" window on the left side) and add a screenshot to this file.




```
=== Summary ===

Correctly Classified Instances          81               79.4118 %
Incorrectly Classified Instances        21               20.5882 %
Kappa statistic                          0.4993
Mean absolute error                      0.2249
Root mean squared error                  0.4473
Relative absolute error                 54.4816 %
Root relative squared error             98.1673 %
Total Number of Instances              102

=== Detailed Accuracy By Class ===

                 TP Rate   FP Rate   Precision   Recall  F-Measure  MCC      ROC Area  PRC Area  Class
                 0.861     0.367     0.849       0.861   0.855      0.499    0.693     0.793     0
                 0.633     0.139     0.655       0.633   0.644      0.499    0.693     0.524     1
Weighted Avg.    0.794     0.300     0.792       0.794   0.793      0.499    0.693     0.714

=== Confusion Matrix ===

  a  b   <-- classified as
 62 10 |  a = 0
 11 19 |  b = 1
```

What is the simplest rule (the simplest path) that can be inferred from the depicted tree?

Simplest rule is actually the most precise and direct path from a root node to a leaf node. Simplest rule that can be inferred from the depicted tree is if Descr465 (which is the root node) is greater than 0.376826 and Descr274 is greater than 2.892948, then predict Class 0.



b. What is the number of leaves and what is the size of the tree? (output window)

Number of Leaves: **58**

Size of the tree: **115**

c. Write down the true positive (TP) rate and false positive (FP) rate for both classes.

TP Rate	FP Rate	Class
0.861	**0.367**	**a=0**
0.633	**0.139**	**b=1**

d. Reduce the option *minNumObj* to 1 and classify again. Does the size of the tree change? Do you have an explanation for that?

The parameter **minNumObj** represents the minimum number of instances that must exist in a leaf (end node) of the tree.

```
=== Summary ===

Correctly Classified Instances          79               77.451  %
Incorrectly Classified Instances        23               22.549  %
Kappa statistic                          0.4406
Mean absolute error                      0.2359
Root mean squared error                  0.4669
Relative absolute error                 57.1237 %
Root relative squared error            102.4623 %
Total Number of Instances              102

=== Detailed Accuracy By Class ===

                 TP Rate  FP Rate  Precision  Recall   F-Measure  MCC      ROC Area  PRC Area  Class
                 0.861    0.433    0.827      0.861    0.844      0.442    0.660     0.776     0
                 0.567    0.139    0.630      0.567    0.596      0.442    0.660     0.483     1
Weighted Avg.    0.775    0.347    0.769      0.775    0.771      0.442    0.660     0.690

=== Confusion Matrix ===

  a  b   <-- classified as
 62 10 |  a = 0
 13 17 |  b = 1
```

When the minNumObj is reduced to 1 the size of the tree increases as each leaf can contain just one instance.

Leaves: **62**

Size of the tree: **123**

- Increase in tree size may not lead to better performance.
- Complexity raises the risk of overfitting, where models perform well on the training data but poorly on the new data.
- Accuracy is reduced from 79.4118% to 77.451%.

It's often a good practice to use larger values for minNumObj to encourage the algorithm to create more general and robust models.

Algorithm 3 – Support Vector Machine

A. Practical exercise: Apply the Weka SMO Support Vector Machine classifier (weka.classifiers.functions.SMO)

a. Run the classifier with different kernels (NormalizedPoly, Poly, Puk and RBF) without changing other options. Write down the overall classification accuracy for each kernel.

Kernels	Accuracy percentage
Normalized Poly	72.7273 %
Poly	76.1905 %
Puk	76.1905 %
RBF	72.7273 %

The Poly and Puk Kernels have achieved the highest accuracy among the tested kernels at 76.1905 %. Normalized Poly and RBF have slightly low accuracy at 72.7273%. Selection of kernels plays an important role in SVM, in this case Poly and Puk are showing better performance.

b. Use the kernel that could achieve the best result and change the kernel option "exponent" to 2 and then to 3. Do the results change?

Choosing POLY Kernel:

Exponent =1: Accuracy is 76.1905 %

Exponent =2: Accuracy is 72.2944%.

Exponent=3: Accuracy is 70.5628 %

With increase in exponent value the accuracy decreased. It means that increased in exponent will lead to the formation of flexible and complex boundaries. Too much flexibility will lead to overfitting.

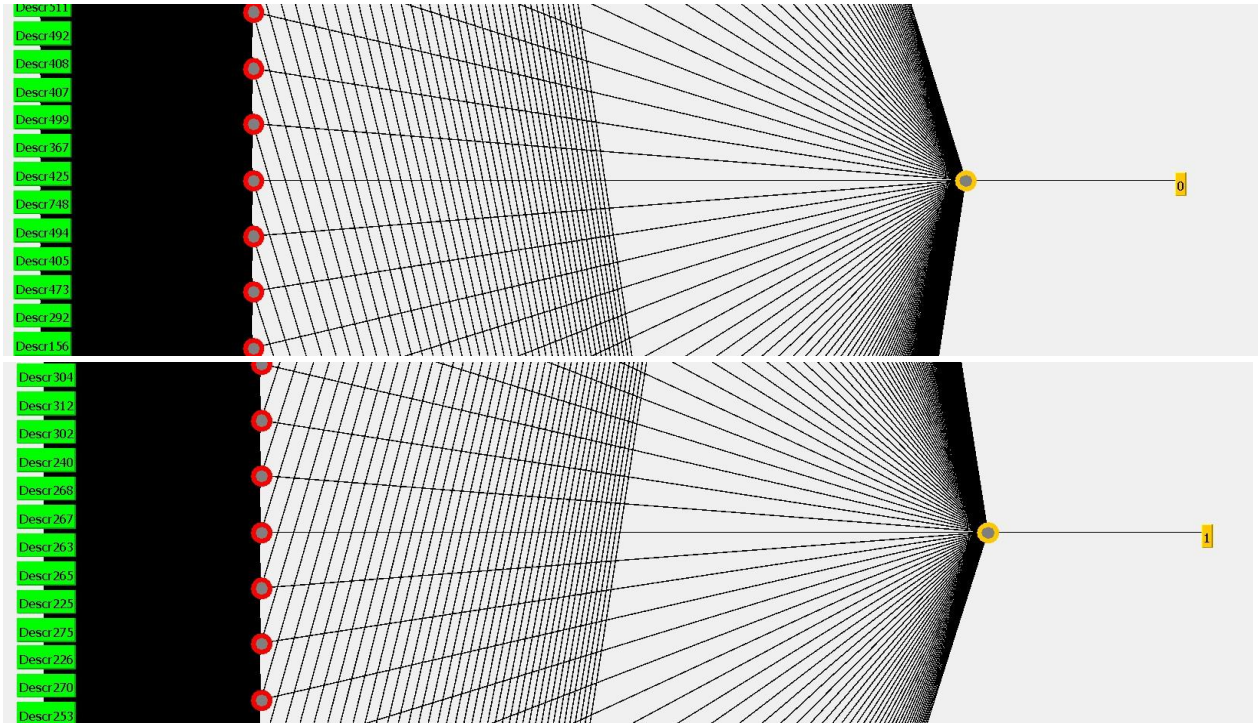
c. * If you have time, play around with the other options of the kernel and write down the best setting.

When exponent is set to 3 adjusting the C to 3 as well, it shows that higher exponent value makes the model more prone to overfitting. The increased value counteracts overfitting to some extent by penalizing margin violations. However, it's crucial to monitor overfitting closely and adjusting C further if needed.

Algorithm 4 – Artificial Neural Network

A. Practical exercise: Apply the Weka MultilayerPerceptron classifier (weka.classifiers.functions.MultilayerPerceptron)

a. Open the File "CDD_2D.arff", choose the multilayer perceptron classifier and change the option GUI to *true. S*elect "Percentage split" as test option and fill in "66**%**" and **un-**check "Preserve order for % Split". Run the classifier and add a screenshot of the GUI.



The GUI shows that by default Num of Epochs are set to 500, learning rate to 0.3 and Momentum to 0.2. Accuracy is 69.69%.

b. Change “Number of Epochs”, “Learning Rate” and “Momentum” in the GUI and write down the best results. You can also try to change the connections via the GUI or add additional hidden layers.

Number of Epochs set to 500, Learning rate to 1.0 and Momentum to 0.1. Results are shown below.

	Before Change in Parameters	**After Changes change in parameters**
Accuracy	69.69%	73.16%.
Correctly Classified	161	169
Incorrectly Classified	70	62
TP rate	0.697	0.732
FP rate	0.401	0.716

An increase in accuracy and TP rate shows that change in parameters have contributed to better model performance.

However, increase in FP rate shows that models are now classifying more instances as positive, leading to higher rate of false positives.

Comparison

A. Practical exercise: Compare the results for the CDD_2D.arff file for the first 3 algorithms

a. Compare the classification results using default options. Which classifier achieves the best classification results (overall accuracy, TP rate, and, FP rate)?

Name	Accuracy	TP rate	FP rate
KNN	**87.2549%.**	**0.873**	**0.150**
Decision Tree	79.411%	0.794	0.300
ANN	69.69%.	0.697	0.401
SVM	76.1905 %	0.762	0.476

Based on the metrics provided, KNN appears to outperform the other classifiers in terms of accuracy, TP rate, and FP rate in this specific scenario.

b. Which classifier was the slowest – which one the fastest?
Slowest: Artificial Neural Network
Fastest: KNN and decision trees are often considered relatively fast, especially when compared to complex algorithms like SVMs and ANN. Here on this data **KNN is the fastest.**

THE END.