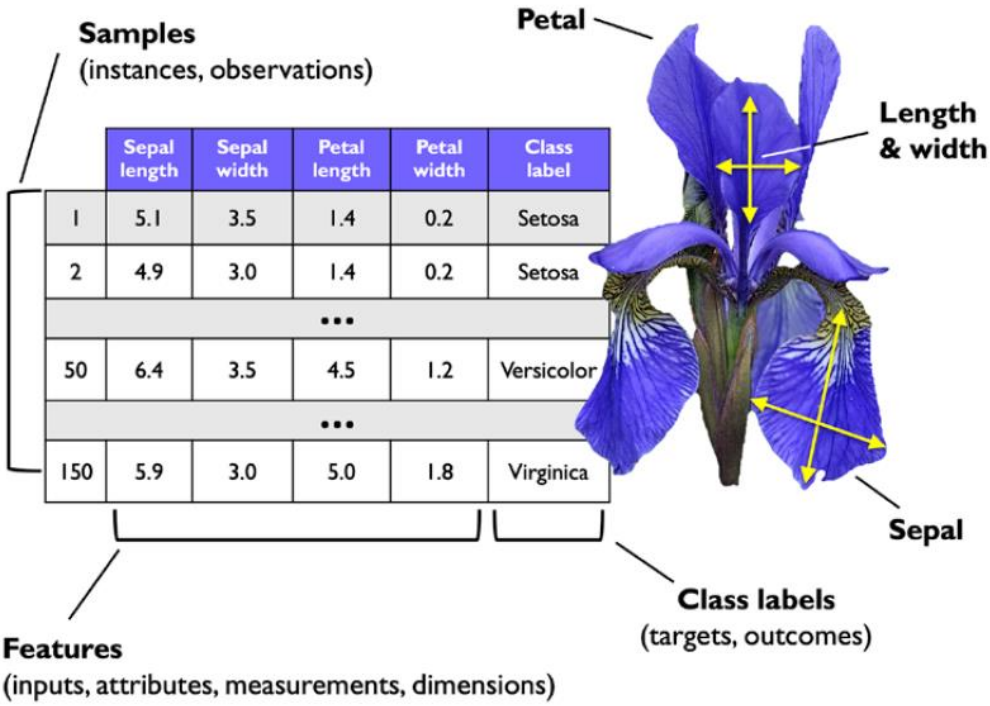
Classification in Weka

Contents

Decision Trees.

1. Load the “iris.arff” dataset.



	Sepal length	Sepal width	Petal length	Petal width	Class label
1	5.1	3.5	1.4	0.2	Setosa
2	4.9	3.0	1.4	0.2	Setosa
...					
50	6.4	3.5	4.5	1.2	Versicolor
...					
150	5.9	3.0	5.0	1.8	Virginica

2. Open the *iris.arff* dataset.
3. Go to classify tab.
4. Choose J48 classifier under trees.
5. Set the test option to percentage split 80%
6. Select the target “(Nom) class”
7. Start training the model.
8. The first section in the output is running information.

```
=== Run information ===

Scheme:       weka.classifiers.trees.J48 -C 0.25 -M 2
Relation:     iris
Instances:    150
Attributes:   5
              sepallength
              sepalwidth
              petallength
              petalwidth
              class
Test mode:    split 80.0% train, remainder test
```

a. “Scheme” is the classifier used with its options
b. “Relation” is the dataset used
c. “Instances” the number of samples in the dataset
d. “Attributes” the number and the name of each feature
e. “Test mode” the method used for testing the model

9. The second section is the training summary.
    a. The hierarchy of the tree.
    b. Feature to split at.
    c. The threshold value to split.
    d. The class label assigned to a particular leaf.
    e. The number of elements reached the leaf followed by the number of which are classified incorrectly.
    f. Number of leaves in the tree.
    g. “Size of the tree” is the total number of nodes
    h. Time taken to construct the tree model

```
=== Classifier model (full training set) ===

J48 pruned tree
------------------

petalwidth <= 0.6: Iris-setosa (50.0)
petalwidth > 0.6
|   petalwidth <= 1.7
|   |   petallength <= 4.9: Iris-versicolor (48.0/1.0)
|   |   petallength > 4.9
|   |   |   petalwidth <= 1.5: Iris-virginica (3.0)
|   |   |   petalwidth > 1.5: Iris-versicolor (3.0/1.0)
|   petalwidth > 1.7: Iris-virginica (46.0/1.0)

Number of Leaves  :     5

Size of the tree :      9


Time taken to build model: 0.09 seconds
```

10. The third section is a summary about the performance of the model.

```
=== Summary ===

Correctly Classified Instances          30               100       %
Incorrectly Classified Instances         0                 0       %
Kappa statistic                          1
Mean absolute error                      0.0105
Root mean squared error                  0.0166
Relative absolute error                  2.3665 %
Root relative squared error              3.5274 %
Total Number of Instances               30
```

11. The fourth section is the detailed accuracy by class.

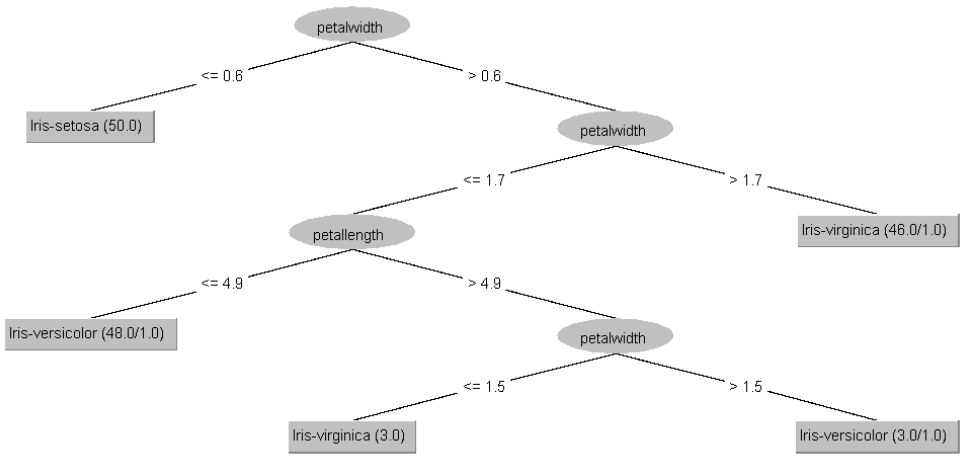
```
=== Detailed Accuracy By Class ===

               TP Rate   FP Rate   Precision   Recall  F-Measure   MCC      ROC Area  PRC Area  Class
               1.000    0.000    1.000      1.000    1.000      1.000    1.000     1.000     Iris-setosa
               1.000    0.000    1.000      1.000    1.000      1.000    1.000     1.000     Iris-versicolor
               1.000    0.000    1.000      1.000    1.000      1.000    1.000     1.000     Iris-virginica
Weighted Avg.  1.000    0.000    1.000      1.000    1.000      1.000    1.000     1.000

=== Confusion Matrix ===

  a  b  c   <-- classified as
 11  0  0 |  a = Iris-setosa
  0 10  0 |  b = Iris-versicolor
  0  0  9 |  c = Iris-virginica
```

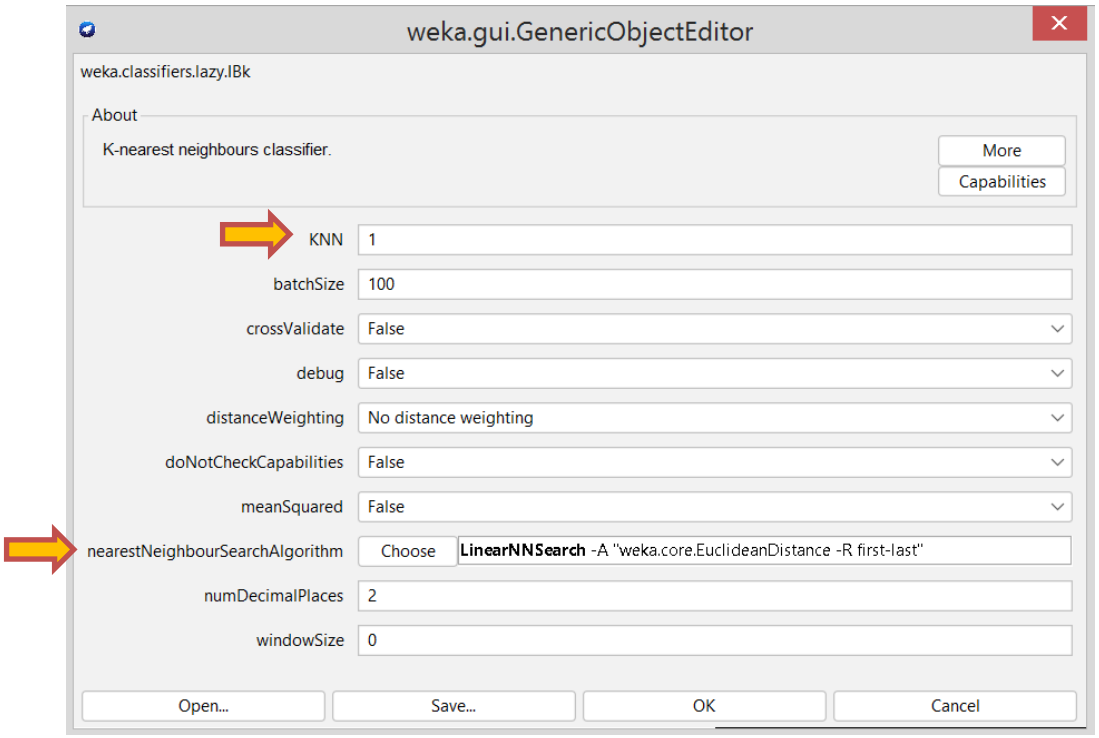
12. You can visualize the tree by right-click on the classifier in the “Result list” and select “Visualize tree”.



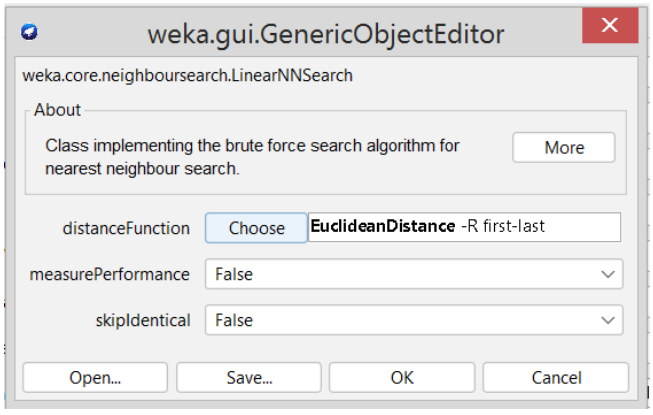
13. You can export your model for deployment by right-clicking on the classifier in the “Result list” and select “Save model”.

KNN

1. Load the “iris.arff” dataset.
2. Go to “classify” tab.
3. Choose “IBk” classifier under “Lazy” category.
4. Click on the classifier options.



   a. “KNN” sets the number of neighbors for voting.
   b. “nearestNeighborSearchAlgorithm” the search method for the nearest instances.
5. Set the “KNN” to 3.
6. Click on the search algorithm to modify the options.



a. Choose the distance funtion and press OK.

7. Set the test options to percentage split of 80%.
8. Run the model and observe the results.

```
=== Summary ===

Correctly Classified Instances          29               96.6667 %
Incorrectly Classified Instances         1                3.3333 %
Kappa statistic                          0.9497
Mean absolute error                      0.0322
Root mean squared error                  0.1477
Relative absolute error                  7.2309 %
Root relative squared error             31.2966 %
Total Number of Instances               30

=== Detailed Accuracy By Class ===

                 TP Rate  FP Rate  Precision  Recall   F-Measure  MCC      ROC Area  PRC Area  Class
                 1.000    0.000    1.000      1.000    1.000      1.000    1.000     1.000     Iris-setosa
                 1.000    0.050    0.909      1.000    0.952      0.929    0.975     0.909     Iris-versicolor
                 0.889    0.000    1.000      0.889    0.941      0.921    0.947     0.923     Iris-virginica
Weighted Avg.    0.967    0.017    0.970      0.967    0.966      0.953    0.976     0.947

=== Confusion Matrix ===

  a  b  c   <-- classified as
 11  0  0 |  a = Iris-setosa
  0 10  0 |  b = Iris-versicolor
  0  1  8 |  c = Iris-virginica
```

9. The model is no better than the decision trees. But it is faster than the J48 classifier.
10. Modify the classifier options (number of neighbors and the distance metric) and try again.

Logistic Regression

1. Load the “diabetes.arff” dataset.
2. Go to “classify” tab.
3. Choose “SimpleLogistic” classifier under “functions” category.
4. Set test options to percentage split of 80%.
5. Set the target value to “(Nom) class”
6. Start training the model.
7. Notice the classifier output.
    a. The weights for “tested_negative” class label is negative of the weights of the “tested_positive” class label.

```
=== Classifier model (full training set) ===

SimpleLogistic:

Class tested_negative :
4.18 +
[preg] * -0.06 +
[plas] * -0.02 +
[pres] * 0.01 +
[insu] * 0    +
[mass] * -0.04 +
[pedi] * -0.47 +
[age] * -0.01

Class tested_positive :
-4.18 +
[preg] * 0.06 +
[plas] * 0.02 +
[pres] * -0.01 +
[insu] * -0 +
[mass] * 0.04 +
[pedi] * 0.47 +
[age] * 0.01
```

The net input (linear regression formula) for predicting “tested_negative” class label.

The net input (linear regression formula) for predicting “tested_positive” class label.

b. The model summary shows that it classified 126 instances out of 154 instances correctly – that is the accuracy is 81.81%

```
=== Summary ===

Correctly Classified Instances         126               81.8182 %
Incorrectly Classified Instances        28               18.1818 %
Kappa statistic                          0.5568
Mean absolute error                      0.2938
Root mean squared error                  0.3756
Relative absolute error                 65.5728 %
Root relative squared error             80.355  %
Total Number of Instances              154
```

c. The “Detailed Accuracy By Class” and the “Confusion Matrix” sections show that most of the misclassification is at the “tested_positive” class label.

```
=== Detailed Accuracy By Class ===

                 TP Rate  FP Rate  Precision  Recall   F-Measure  MCC      ROC Area  PRC Area  Class
                 0.914    0.388    0.835      0.914    0.873      0.564    0.840     0.886     tested_negative
                 0.612    0.086    0.769      0.612    0.682      0.564    0.840     0.780     tested_positive
Weighted Avg.    0.818    0.292    0.814      0.818    0.812      0.564    0.840     0.852

=== Confusion Matrix ===

  a  b   <-- classified as
 96  9 |  a = tested_negative
 19 30 |  b = tested_positive
```

8. Experiment the model again but with cross validation of 10 Folds. Does it perform better than the initial model?

Exercise

Evaluate the KNN, Decision Tree, and Logistic Regression model on the "car_evaluation.csv" dataset.

Consider the following fixed training and testing options:

1. Set the test option to percentage split of 80%.
2. The target label is "(Nom) decision".
3. Leave the default options of each classifier, except for KNN, set k = 3.
4. For logistic regression model, we will choose "Logistic" model instead of "Simple logistic" as this is a multi-class problem.

Experiment 1 – removing inconvenient attributes

1. Load the "car_evaluation.csv" file into Weka.
2. Notice that the attributes "number of doors" and "number of persons" contain string values (both nominal and numeric values).
3. Go the "classify" tab, you find that almost all the classifiers cannot be used!
4. Return to the "preprocess" tab.
5. Select the "number of doors" and "number of persons" and remove them.
6. Go to the "classify" tab and run the mentioned classifiers.
7. Compare the results of each classifier.

Experiment 2 – fixing inconvenient attributes

1. Reload the "car_evaluation.csv" file into Weka.
2. In filters, choose *Unsupervised* → *attribute* → *StringToNominal*.
3. Click on the filter options.
4. Set the attribute range to "3-4" to indicate that we will pre-process the third and fourth attributes.
5. Click Ok, then click "Apply" on the filter.
6. Notice how the attributes "number of doors" and "number of persons" are changed.
7. Go to the "classify" tab and run the mentioned classifiers.
8. Compare the results of each classifier.

Experiment 3 – changing a nominal attribute to numerical

1. Reload the “car_evaluation.csv” file into Weka.
2. In filters, choose *Unsupervised → attribute → StringToNominal*.
3. Click on the filter options.
4. Set the attribute range to “3-4” to indicate that we will pre-process the third and fourth attributes.
5. Click Ok, then click “Apply” on the filter.
6. Next, choose another filter called “OrdinalToNumeric”. Choose *Unsupervised → attribute → OrdinalToNumeric*.
7. Click Ok, then click Apply.
8. Go to the “classify” tab and run the mentioned classifiers.
9. Compare the results of each classifier with respect to the previous experiment.