|  |
| --- |
| UMUC |
| Classification via Multilayer Perceptron Function |
| Week 4 Group Exercise |
|  |
| **DBST 667** |
|  |

|  |
| --- |
| For this exercise, you will use Multilayer Perceptron classification function in WEKA Explorer interface. You will run the algorithm with different parameters’ values and will compare the results from the best run with the J48 algorithm results from week 3 exercise. The analyses include the classification accuracy, confusion matrix, detailed accuracy by class. In addition to setting the algorithm parameters in generic object editor, the model enables changing the learning rate and momentum interactively at run time. |

**Classification via Multilayer Perceptron Function**

[1.0 The Data File 2](#_Toc380329302)

[2.0 Loading the Data File 4](#_Toc380329303)

[3.0 Selecting a Classifier 6](#_Toc380329304)

[3.1 Setting Test Options 7](#_Toc380329305)

[3.2 Setting Evaluation Options 8](#_Toc380329306)

[4.0 Analyzing Results 12](#_Toc380329307)

[4.1 Run Information 12](#_Toc380329308)

[4.2 Evaluation on Test Split 13](#_Toc380329309)

[4.3 Detailed Accuracy of Class 14](#_Toc380329310)

[4.4 Confusion Matrix 17](#_Toc380329311)

[5.0 Results Visualization 17](#_Toc380329312)

[6.0 Comparison with J48 Algorithm Results from Weed 3 Exercise 23](#_Toc380329313)

**Classification via Multilayer Perceptron Function**

This exercise illustrates the use of Multilayer Perceptron classifier function in Weka. The function implements a back propagation algorithm to build a Neural Network model to classify the instances.

# 1.0 The Data File

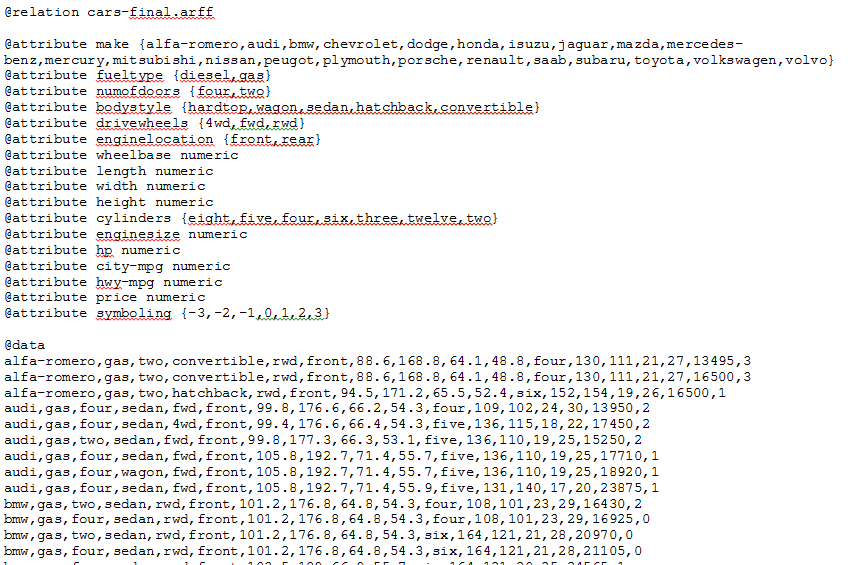
We will use the attached not discretized **cars-final.arff** data file for the first run of an exercise below. Let’s first examine the file content.

Figure 1 shows the partial content of the **cars-final.arff** file. The file **header** contains

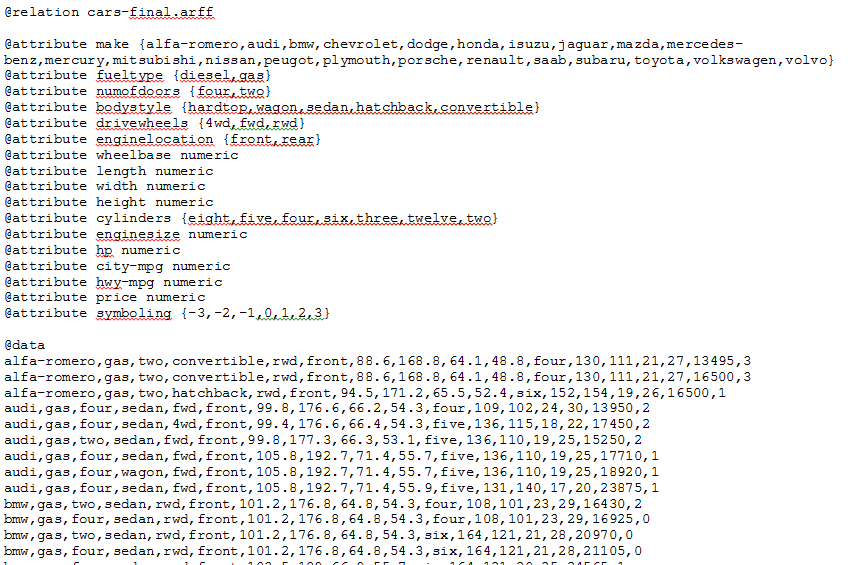
* **relation** name- the data file name is specified after the token **@relation**.
* **attributes** list - each attribute definition follows the token **@attribute**.

The data has 17 attributes. The attribute types are

* **Numeric** – real or integer numbers. The attribute definition includes the attribute name and keyword numeric. The keyword numeric is case insensitive.



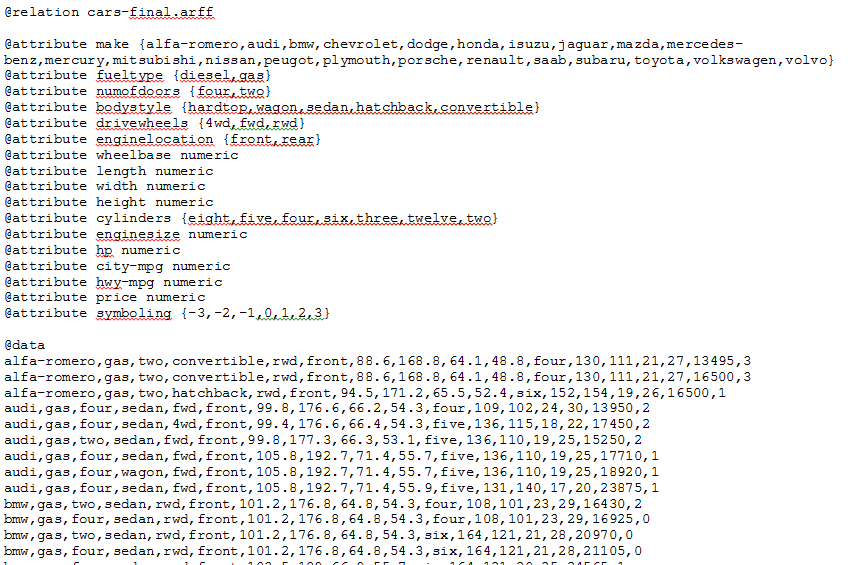
* **Nominal** – the attribute definition includes the attribute name and a list of valid values.

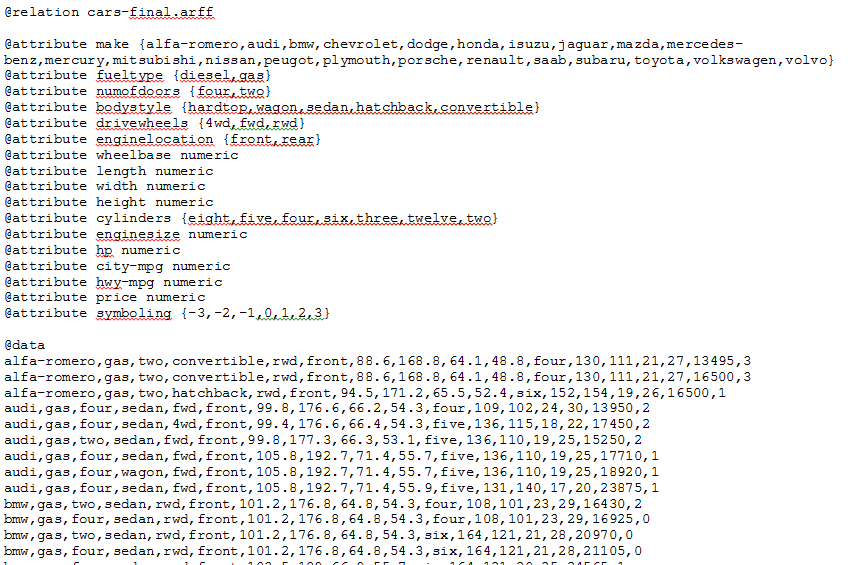


Each row in the data section (an instance) corresponds to a specific car record, and there are 197 car records. The order in which the attributes are declared indicates the column position in the data section. For example, if a fueltype is the second attribute on a list, the fueltype value for each car record is in the second column of the data row.

Example - First data row

Make fuel type numofdoors bodystyle drivewheels enginelocation wheelbase





Relation

Header

Attributes

Data

(Instances)

Figure **:** Content of the **cars-final.arff** file

Table 1 summarizes the possible values for each of the 17 attributes in the **cars-final.arff** data file.

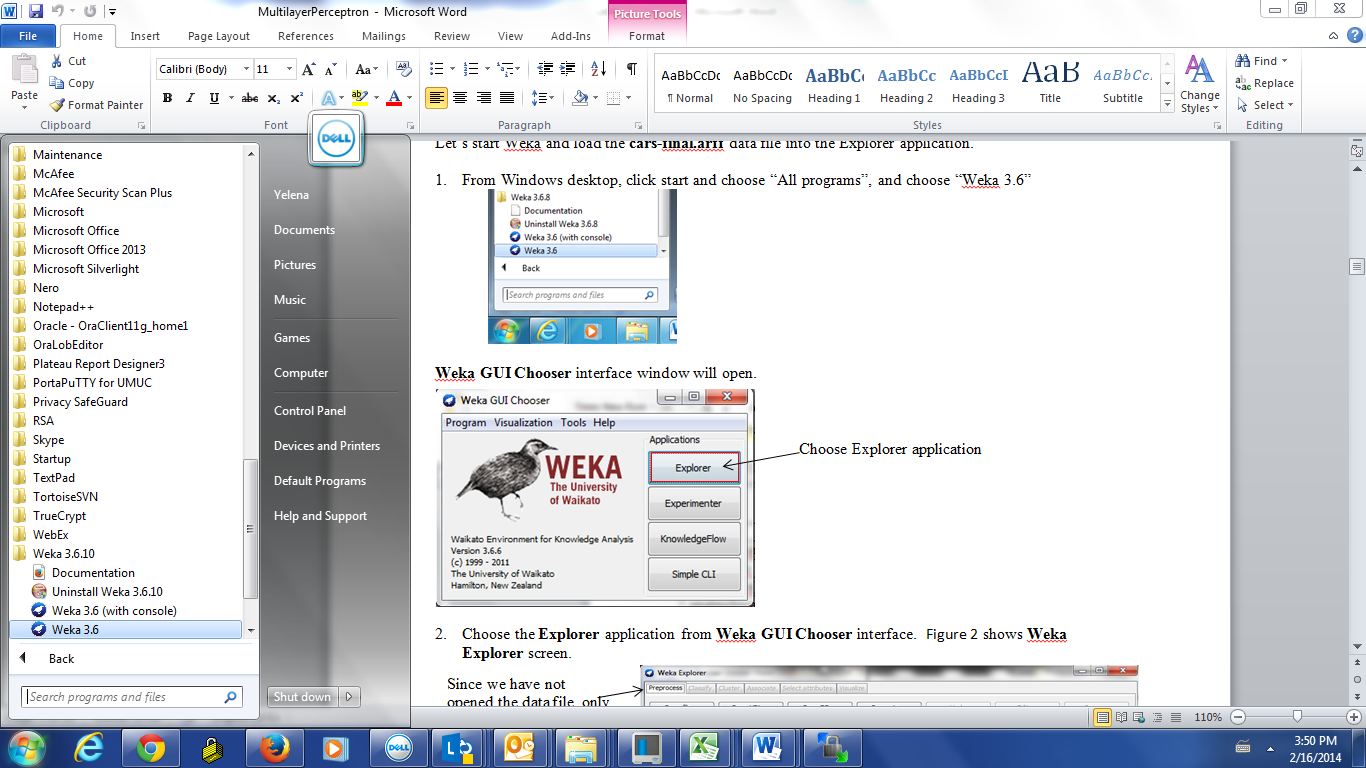
**Table 1**

|  |  |
| --- | --- |
| make | {Alfa-romero, Audi, BMW, Chevrolet, Dodge, Honda, Isuzu, Jaguar, Mazda, Mercedes-Benz, Mercury, Mitsubishi, Nissan, Peugot, Plymouth, Porsche, Renault, Saab, Subaru, Toyota, Volkswagen, Volvo} |
| fueltype | {diesel, gas} |
| numofdoors | {four, two} |
| bodystyle | {hardtop, wagon, sedan, hatchback, convertible} |
| drivewheels | {4wd, fwd, rwd} |
| enginelocation | {front, rear} |
| wheelbase | numeric (values from 86.6 to 120.9) |
| length | numeric (values from 141.1 to 208.1) |
| width | numeric (values from 60.3 to 72.3) |
| height | numeric (values from 47.8 to 59.8) |
| cylinders | {eight, five, four, six, three, twelve, two} |
| enginesize | numeric (values from 61 to 326) |
| hp | numeric (values from 48 to 288) |
| city-mpg | numeric (values from 13 to 49 |
| hwy-mpg | numeric (values from 16 to 54) |
| price | numeric |
| symboling | symboling { -3, -2, -1, 0, 1, 2, 3}  A value of +3 indicates that the auto is risky, -3 that it is probably pretty safe. |

# 2.0 Loading the Data File

Let’s start Weka and load the **cars-final.arff** data file into the Explorer application.

1. From Windows desktop, click start and choose “All programs”, and choose “Weka 3.6”



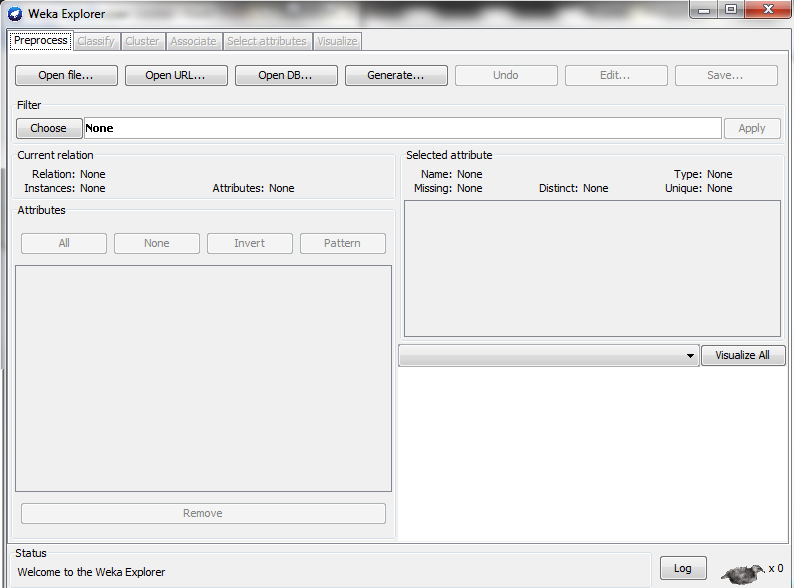
**Weka GUI Chooser** interface window on Figure 2 will open.



Choose Explorer application

Figure : Weka GUI Chooser Window

1. Choose the **Explorer** application from **Weka GUI Chooser** interface. Figure 3 shows **Weka** **Explorer** screen.



Since we have not opened the data file, only **Preprocess** tab is active.

The rest of the tabs are greyed out.

The attributes list is empty

Welcome message on a status bar

Number 0 next to an X means that no processes are currently running.

Figure : Preprocess tab before opening the file

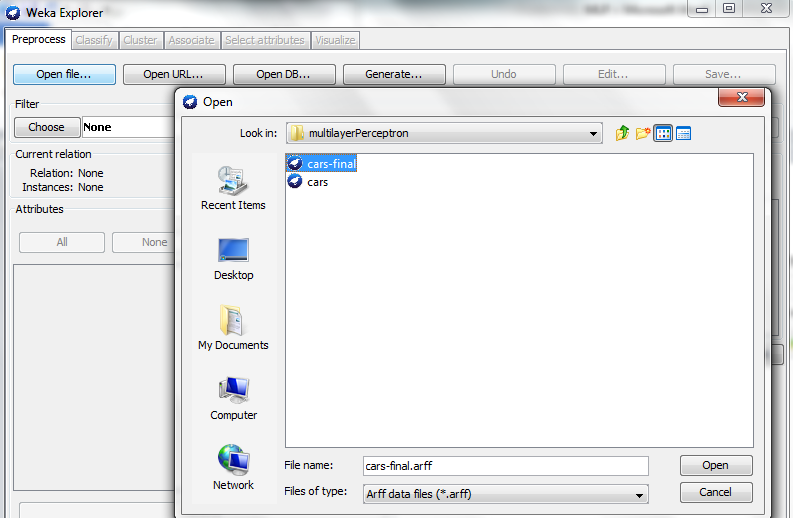
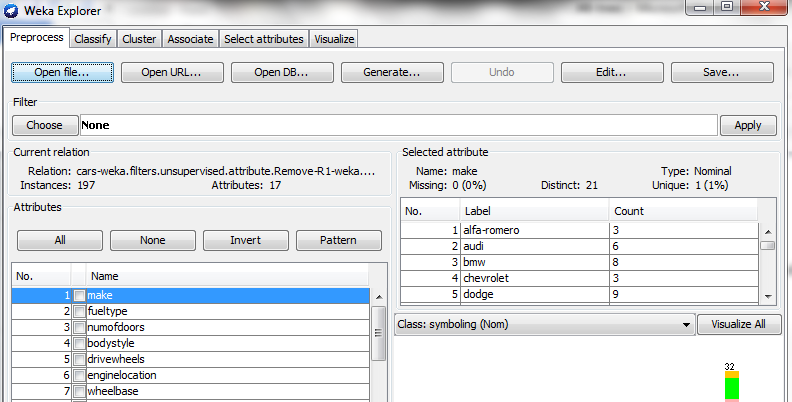
1. Click **Open file**… button on Figure 4 to open the data file, **cars-final.arff**.

Figure : Open cars-final data file

Figure 5 shows that the dataset has 197 instances and 17 attributes. All tabs are enabled.

All tabs are available. Preprocess tab is active

Attribute type of make is nominal

Current relation panel shows that cars dataset has 197 instances with 17 attributes

Valid values of a make attribute and the number of instances that have each value

Attributes list

Make is the selected attribute

Last attribute is selected for a class by default (dependent variable)

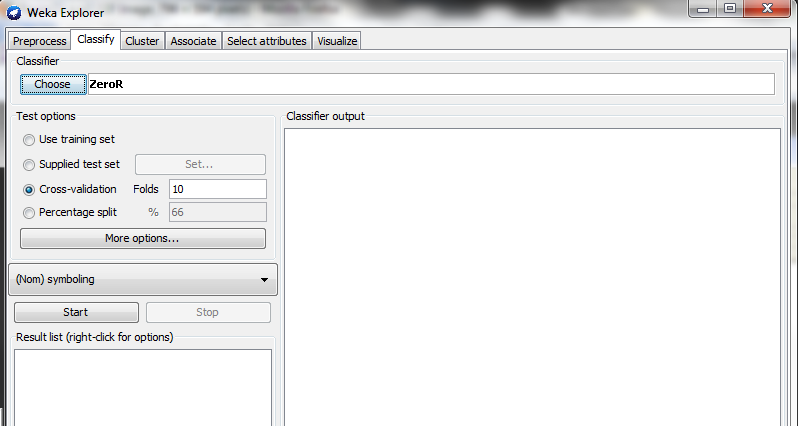
Figure : Preprocessing tab after te data file is opened

# 3.0 Selecting a Classifier

Once the dataset is loaded, all the tabs are available. Follow the steps below to select **Multilayer Perceptron** classifier (**weka.classifier. functions.MultilayerPerceptron**)

1. Click the on a **Classify** tab. Figure 6 shows the classify tab interface.

Selected classifier name and specified parameters (ZeroR is the default)



Stop the algorithm execution

An attribute to predict (dependent variable) ; last attribute is selected by default

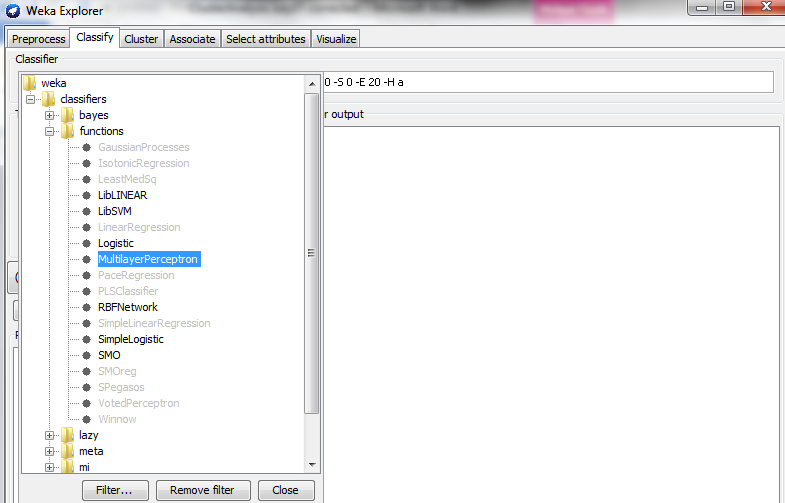
The classification output will be displayed in this area.

Clicking choose button opens up the hierarchical menu with data mining algorithms

Figure : Weka Classify tab

Start the algorithm execution

1. Click on a **Choose** button at the top-left to expand the hierarchical menu, and follow Figure 7 to find and select **MultilayerPerceptron** function.



Select multylayer perceptron

Expand functions folder

Expand classifiers folder

Figure : Select MultilayerPerceptron algorithm function

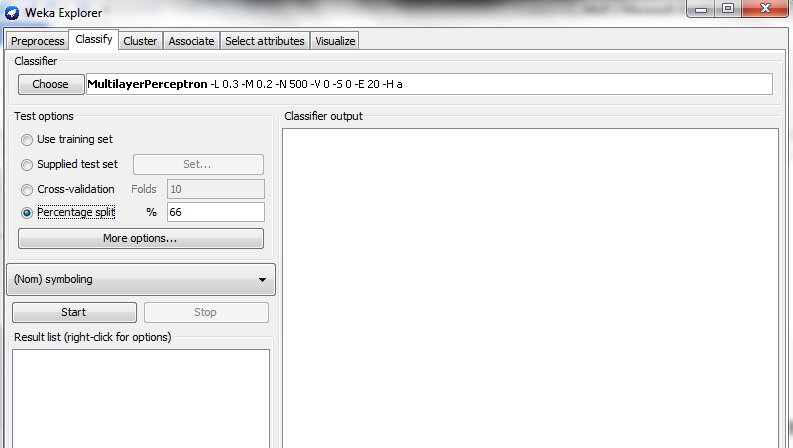
## 3.1 Setting Test Options

Select **Percentage split** in the **Test options** panel.

**Percentage split:** The value in the ‘%’ field specifies the percentage of data to be used for building an initial model (training data). After the data model is built, the remaining data (test data) are used to test the accuracy of the model. By default, 66% of data are used for training.

1. Check **Percentage split** radio-button on Figure 8 and keep **66%** for the percentage value.

Clicking on this text box brings up the **GenericObjectEditor** dialog box to set the additional algorithm options.



66% of dataset will be used as training data, and the remaining 34% will be used as test data

The classification output will be displayed in this area.

Options for the algorithm output content

Stop the algorithm execution

An attribute to predict (dependent variable)

Start the algorithm execution

New result entry is added after each algorithm run

Figure : Select percentage split test option

## 3.2 Setting Evaluation Options

1. Click on a textbox on the right of the **Choose** button to open a **GenericObjectEditor** dialog box on Figure 9. Keep the default values for all parameters.
2. Click **OK**.

The following parameters can be set to control the initial network structure.

**Hiddenlayers** - The number of nodes each hidden layer has. We determine the optimal number of nodes in a layer by trial-and-error. The parameter value is a comma-separated list of integers and/or wildcards.

Examples:

* 3, 2 – one hidden layer with 3 nodes, and one hidden layer with 2 nodes
* 5 – one hidden layer with 5 nodes
* 0 – no hidden layer

Supported wildcards (predefined values):

* ‘a’= (number of attributes + number of classes)/2. – the default value.

Our dataset has 17 attributes and 7 classes, a=(17+7)/2=12

* ‘i’= number of attributes
* ‘o’= number of classes
* ‘t’=number of attributes +number of classes

In our case, t=17+7 =24

**Learningrate** - the amount the weights are updated on each iteration

**Momentum** - momentum applied to the weights during an update

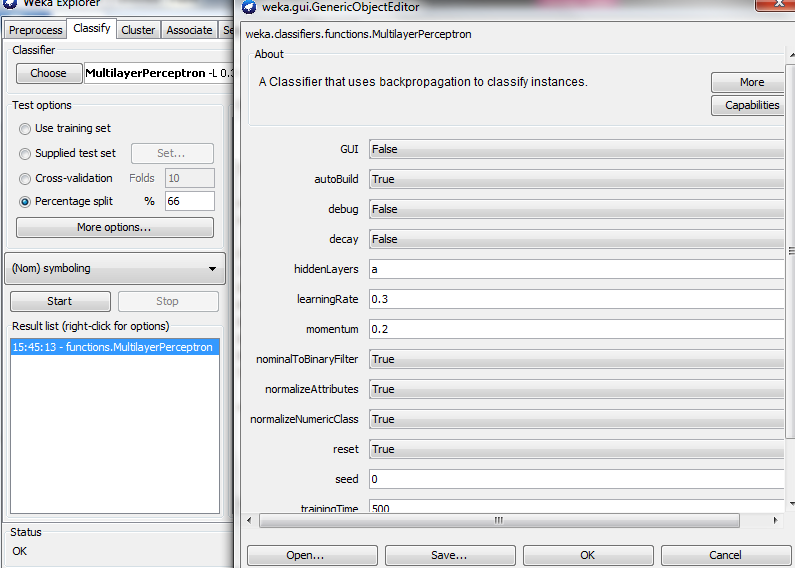
**trainingTime** - The number of passes through the data (epochs) to train the network.

**Seed -** A random number for setting the initial network weights

**Decay -** When set to true, the learning rate will decrease with time during the training

Clicking on this text box brings up the **GenericObjectEditor** dialog box to set the additional algorithm options.

Click “more” button to get more explanation for parameters



66% of dataset will be used as training data, and the remaining 34% will be used as test data

An attribute being predicted is **symboling**

Convert nominal input values into binary

Wildcard a = Number of nodes in a hidden layer= (number of attributes + number of classes)/2

Clicking start button to run the algorithm (step 3)

Number of training epochs (passes through the data)

Click to continue

Figure : GenericObjectEditor Options

Random number used for initial weights assignment

1. Click **Start** to run the algorithm.

The network training stops when the specified number of epochs is reached. We can accept the result or increase the desired number of epochs and press **Start**again to continue training.

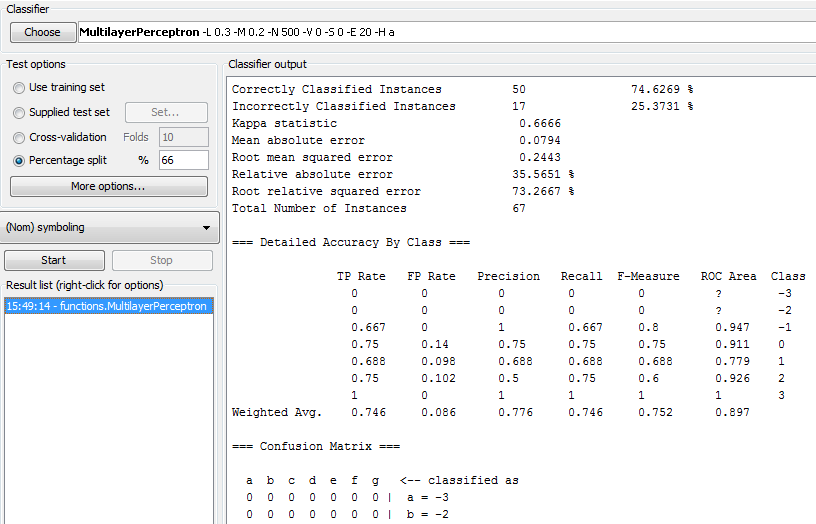
The **Classifier output** area in the right panel on Figure 10 displays the algorithm run results.

The classification accuracy of our model for the test data is **74.6269%**.

The test data consists of 67 instances (34% out of 197 instances).

50 instances were classified correctly, and 17 instances were misclassified.

Algorithm name



Number of instances in the test data (34% out of 197 instances in the dataset)

66% of dataset is used as training data, and the remaining 34% is used as test data

New entry on the results list

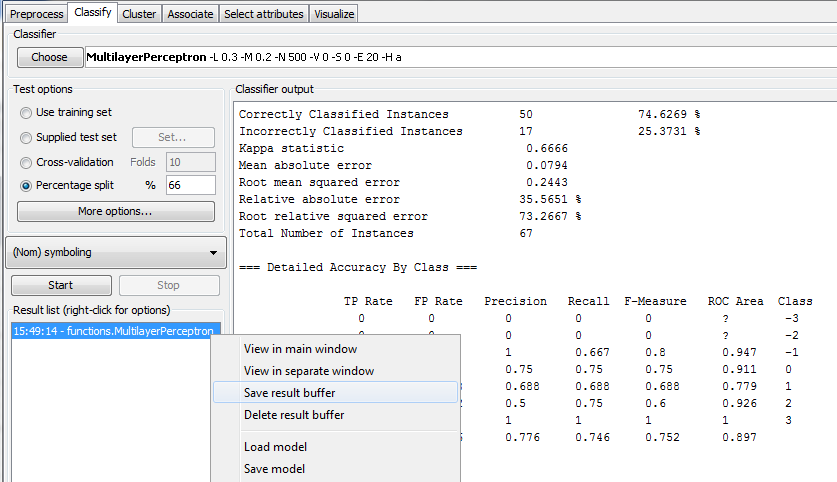
An attribute being predicted is **symboling**

Classification accuracy

Figure : Partial Algorithm Output

A new result entry has been added to the **Result list** panel.

1. Right-click on an entry to open a pop-up menu on Figure 11.
2. Select **Save result buffer** to save the algorithm run result as **result.txt,** as shown on Figure 12.

****

Select to save the results file

An attribute being predicted **symboling**

Right mouse click on results

Figure : Save results buffer

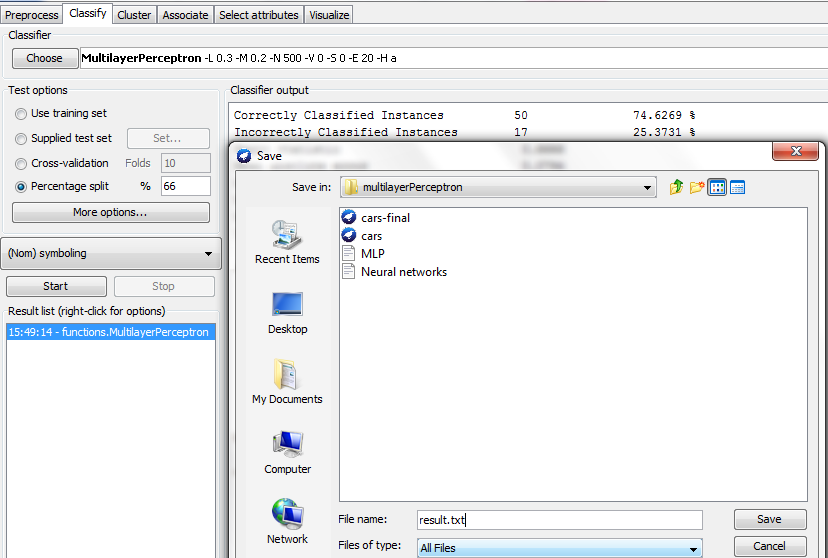
****

Figure : Save the result file

# 4.0 Analyzing Results

Let’s open the result file we just saved (**result.txt**) to analyze the results.

## 4.1 Run Information

**Run Information** on Figure 13 includes:

Classification scheme: **MultilayerPerceptron**

Relation name: **cars-final**

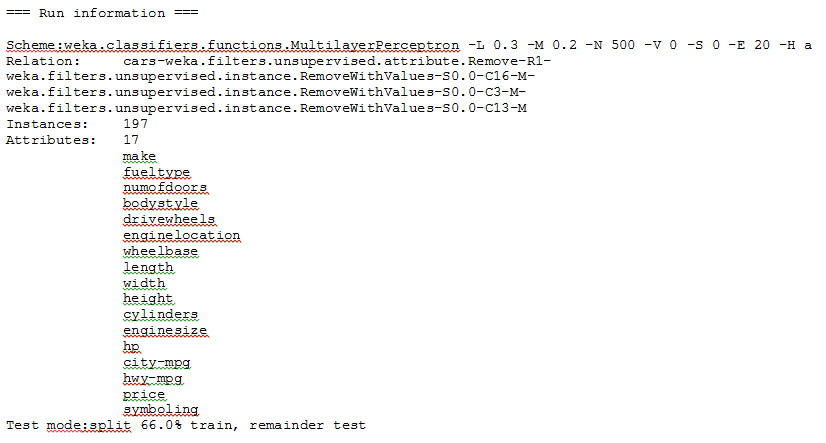
Number of instances: **197**

Number of attributes: **17**

Attributes list used for classification

Algorithm parameters

Classification scheme (algorithm name)

******

v

Data pre-processing filters before running Multilayer Perceptron algorithm

Training time

66% of dataset is used as training data, and the remaining 34% is used as test data

Attributes used for classification

Number of attributes

Number of instances

Relation name

v

Figure : Run Information

The output includes the time it took to build the model, which is 7.46 seconds.

We should rerun the algorithm with different **trainingTime, momentum, and** **learningrate** values, and compare the results of each run. Table 2 summarizes the five algorithm runs.

**Table 2**

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **trainingTime**  **(epochs)** | **Momentum** | **Learning Rate** | **Correctly classified Instances** (%) | **Time to build model(s)** |
| 500 | 0.2 | 0.3 | 74.6269 | 10.55 |
| 500 | 0.4 | 0.3 | 73.1343 | 15.32 |
| 500 | 0.8 | 0.3 | 74.6267 | 12.77 |
| 500 | 0.2 | 1.0 | 79.1045 | 11.37 |
| 500 | 0.8 | 1.0 | 35.8209 | 15.71 |

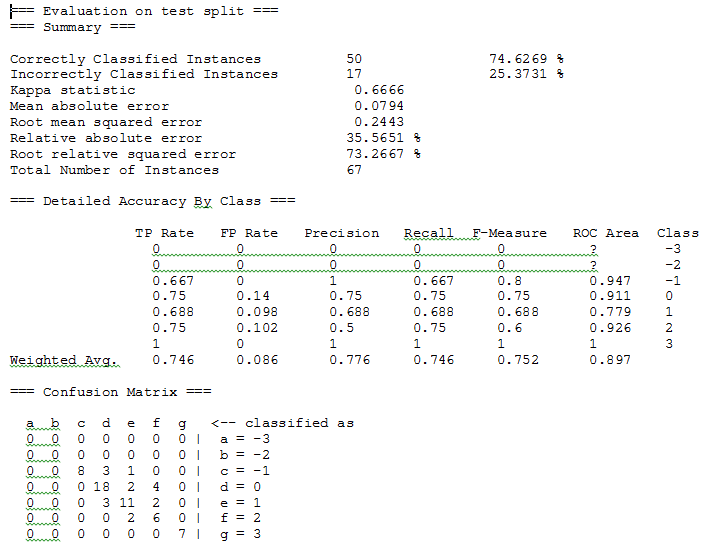
Which parameter options give the best performance?

## 4.2 Evaluation on Test Split

**Evaluation on test split** - an output section on Figure 14 shows an estimated predictive performance of a function. The statistics summarize how accurately the classifier predicted each class of the test instances.

In this case, 74.6269% of instances in the **test** datawere classified correctly. The **Root Mean squared error** is 0.2443.

**Total Number of Instances** in the test data = 34% of 197 (total number of instances in the data set) = 67



**Test option**

**Number of instances in the test data (34% of the dataset)**

**Prediction Accuracy**

**Actual class**

**Number of correctly classified instances is the sum of numbers on a diagonal**

Figure : Evaluation on test split

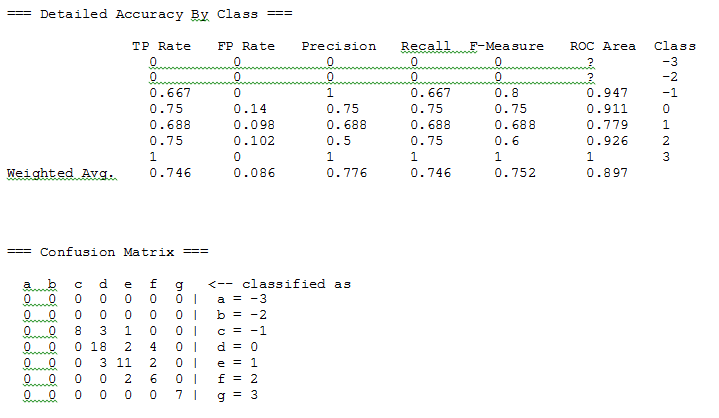
## 4.3 Detailed Accuracy of Class

**Detailed Accuracy By Class** demonstrates break down of the classifier’s prediction accuracy. The measures (Precision, Recall, F- Measure, and ROC area) on Figure 15 are useful for comparing classifiers.

**True Positive (TP)** rate is equivalent to **Recall** or **Sensitivity**. In the confusion matrix, this is the diagonal element divided by the sum over the relevant row.

TP / (TP + FN)

True positive rate for class c= 8 /(8+3+1) = 0.667

****

**True positives (TP) for class c** – instances that belong to class c that were classified as class c (diagonal element).

**False negatives (FN) for class c (Type II error)** – instances that belong to class c but classified as either a, b, d, f, f, or g

Row sum for class c =8+3+1=12

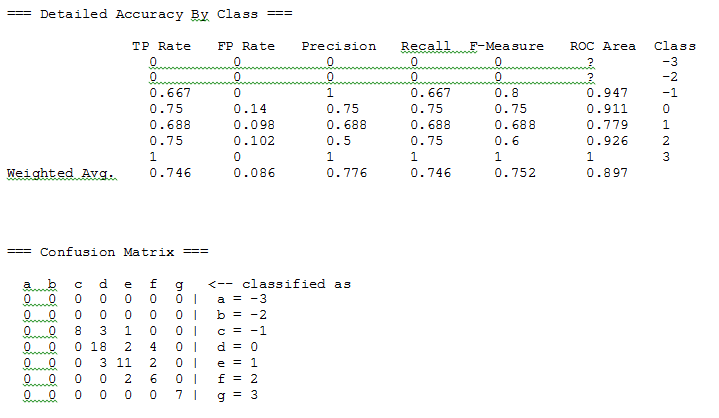
**Example:**

Actual class

Diagonal element in class c

**False Positive (FP)** rate is the column sum of class x minus the diagonal element, divided by the rows sums of all other classes.  
 FP / (FP + TN)

The False Positive rate for class d= (3+18+3-18) /(8+3+1+3+11+2+2+6+7) = 0.14

****

**Example:**

Elements in all other classes, and their Sum=8+3+1+3+11+2+2+6+7

Column sum – diagonal element=(3+18+3)-18

Actual class

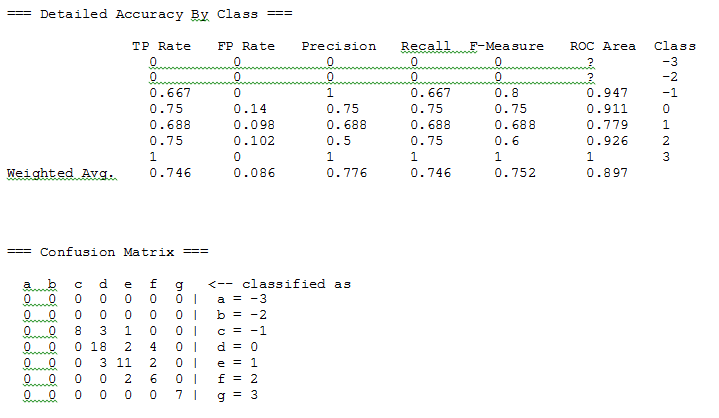
Diagonal element in class d

**False positives (FP) for class d (Type I error)** – instances that don’t belong to class d, that where classified as class d

**True negatives (TN) for class d** **(Type II error)**– instances that don’t belong to class d, and classified as either a, b, c, e, f, or g

**Precision** is the diagonal element divided by the sum over the relevant column.

TP / (TP + FP)

Precision for class f= 6/(6+4+2) = 0.5  
****

Diagonal element in class f

**True positives (TP) for class f** – instances that belong to class f that were classified as class f (diagonal element).

**False positives (FP) for class f** **(Type I error)** – instances that don’t belong to class f, that where classified as class f

**Example:**

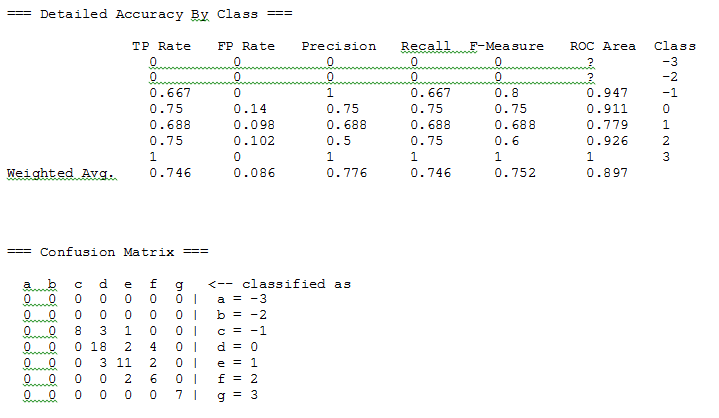
Actual class

Column sum=4+2+6

**F-Measure** is a combined measure for recall and precision

2TP / (2TP + FP + FN) or 2\*Precision\*Recall /(Precision + Recall)

F- Measure for class f= 2\*6/(2\*6+6+2)=0.6

****

**True positives (TP) for class f** – instances that belong to class f that were classified as class f (diagonal element).

**False positives (FP) for class f** **(Type I error)**– instances that don’t belong to class f, that where classified as class f

**False negatives (FN) for class f (Type II error)** – instances that belong to class f, and classified as either a, b, c, d, e, or g

Actual class

True positives for class f=6

False positives for class f=4+2=6

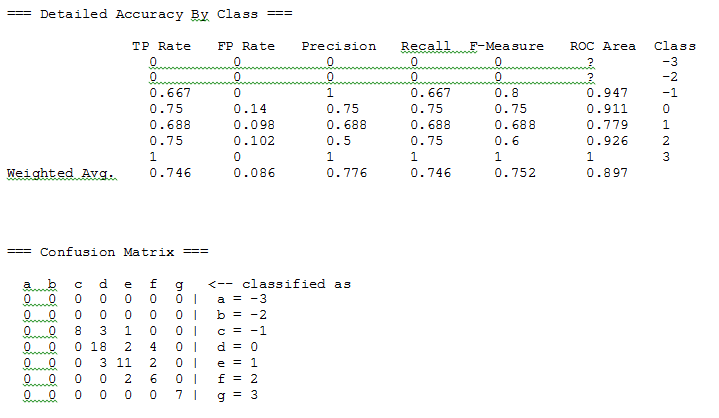
False negatives for class f=2

The area under the ROC curve is determined by the true positive and false positive rate. ROC area=1 indicates a perfect prediction, while ROC area=0.5 indicates a random prediction.

The ROC area values in the detailed accuracy by class section of an output indicate that class -1 has the best prediction because it has the highest ROC area value. Class 1 has the weakest prediction because it has the lowest ROC value.

The question mark in the ROC Area column indicates that an algorithm has not been able to predict the class. In our example, class -3 and class -2 have a question mark in the ROC Area column, and the values for all measures are 0.

An algorithm has not been able to predict the class

****

Class - 1 has the highest ROC area- the best prediction

Class 1 has the lowest ROC area- the weakest prediction

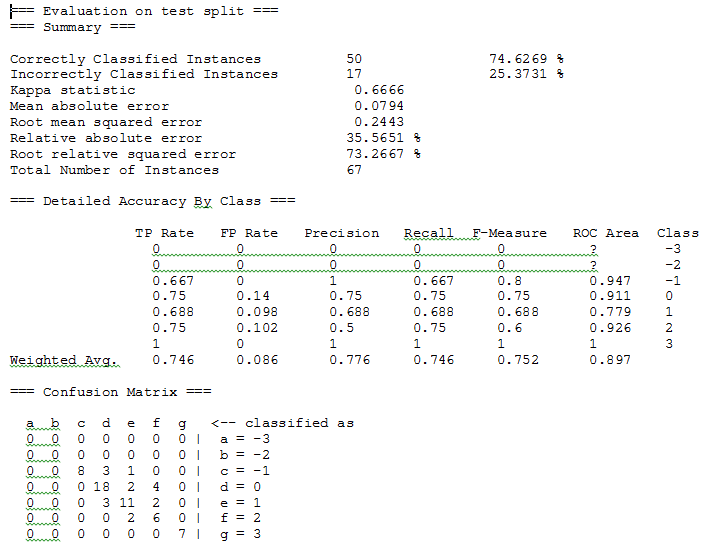
**Actual class**

**Number of correctly classified instances is the sum of numbers on a diagonal**

Figure : Detailed acuracy by class

## 4.4 Confusion Matrix

In our dataset, we have five classes. Therefore the size of the **Confusion matrix** is 5x5.



4 instances classified as f have an actual class d

**Actual class**

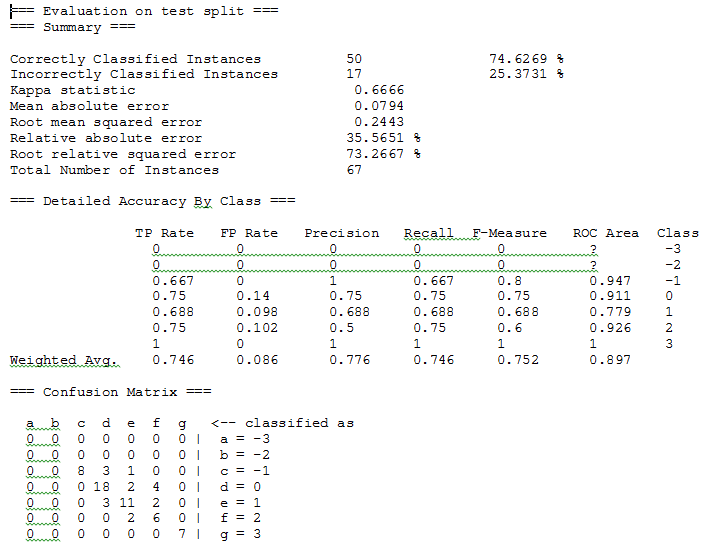
1 instance classified as e has an actual class c

A **Confusion matrix** shows how many instances in a test data have been assigned to each class. For each matrix element, the row label is an actual class, and the column label is a predicted class. For example

* 4 instances classified as f have an actual class d
* 1 instance classified as e has an actual class c

Based on the above confusion matrix:

* The number of correctly classified instances is the sum of numbers on a matrix diagonal from top right to bottom left. The model made **50** correct predictions (8+18+11+6+7)
* The number of incorrectly classified instances is **17** (3+1+2+4+3+2+2). It’s the sum of numbers not on the diagonal from top right to bottom left.



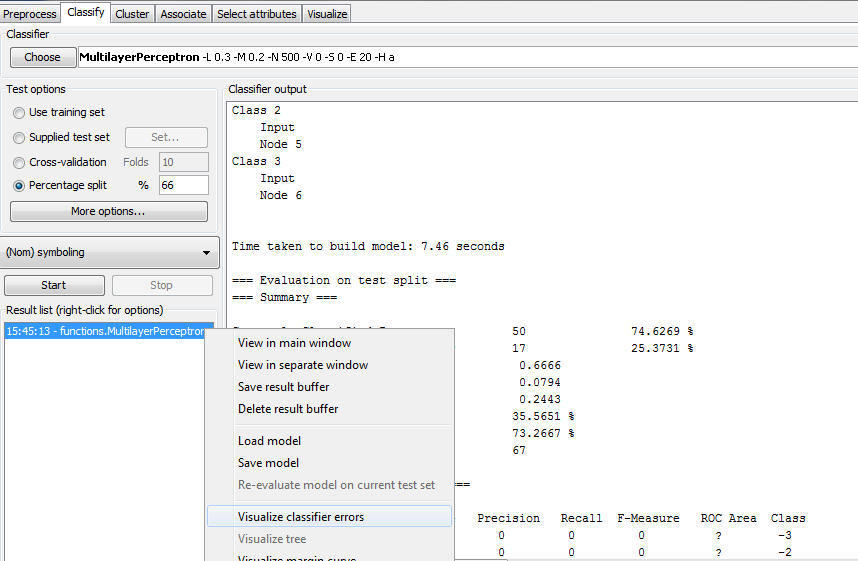
Number of incorrectly classified instances= sum of numbers not on diagonal

* The model scored **67** cases (50 +17) – total number of instances in the test data
* The accuracy rate is 50/67 = **0.7462 -** number of correct predictions/number of incorrect predictions

# 5.0 Results Visualization

Weka lets us **visualize classification errors**.

1. Right-click on the entry in **Result list**  and select **Visualize classifier errors** from the menu, as shown on Figure 16.



Select to open a two-dimensional plot

Right click on result to open a pop-up menu

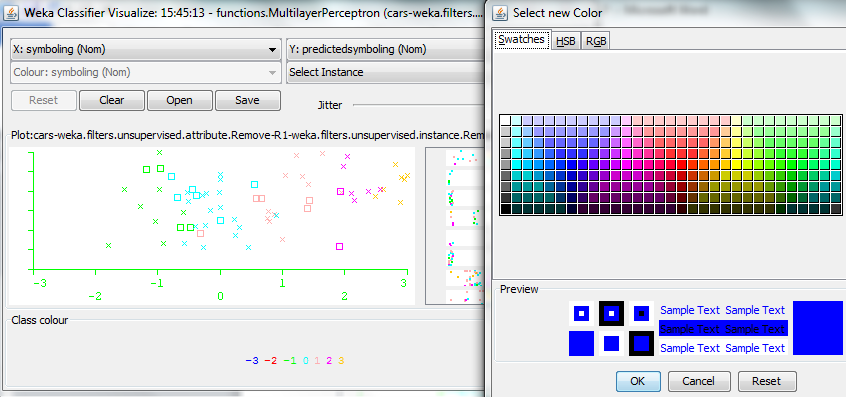
Classifying attribute is **symboling**

Figure : Select visualize classifier errors

1. Select **Visualize classifier errors** from the popup menu to open a two-dimensional plot on Figure 17.
2. Choose which attributes to use for X-axis and Y-axis using the selection boxes at the top of the **Weka Classifier Visualize** screen.
3. The **Class Color** key at the bottom of the screen on Figure 17 specifies what color code represents each class value. To change the color, click on a class value, and select the color from the palette.

In this example, red color represents the instances with symboling class **-2**. Green color represents the instances with symboling class **-1**, and so on.

Left-click on **-1** to open the **color** palette window. The top section of the pallet has the color choices, and the bottom section has sample text preview. Select the color and click ok.



Different color code is used for each class. Click on a value to open the color pallet

Different color codes are used for the actual symboling values

Sample text preview

Correctly classified instances are represented as x

Incorrectly classified instances are represented as boxes

Figure : Classification errors

Correctly classified instances are represented as crosses, and incorrectly classified instances are represented as boxes (see Figure 18).

Left-click on the **green box** data point to view the **instance info,** including the instance number, attributes values, predicted class, and actual class.

Instance number

X-axis attribute

Y-axis attribute



Instance 33 is misclassified. Predictedsymboling =0 while actual symboling= -1

Attribute values for the selected instance

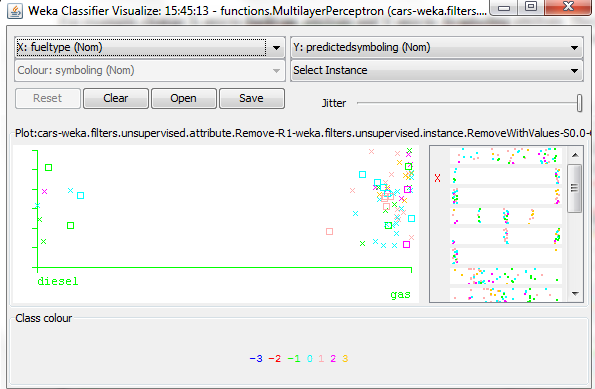
Different color code is used for each class

Click on the following box representing incorrectly classified instance to bring up more information

Figure : Instance Information

1. Each horizontal strip in the right panel shows the distribution of values for an attribute. Left-click on a strip to select an attribute for **X** -axis, and right-click on a strip to select an attribute for **Y-axis**.

Follow Figure 19 to change X-axis to **fueltype** attribute and Y-axis to **Symboling** attribute. The instances will spread out in the plot area and concentration points will be invisible. Keep sliding **Jitter** to the right, until the concentration points are visible.



Use different color code for each actual symboling class

Predicted symboling is selected for the Y-axis

Class color code legend

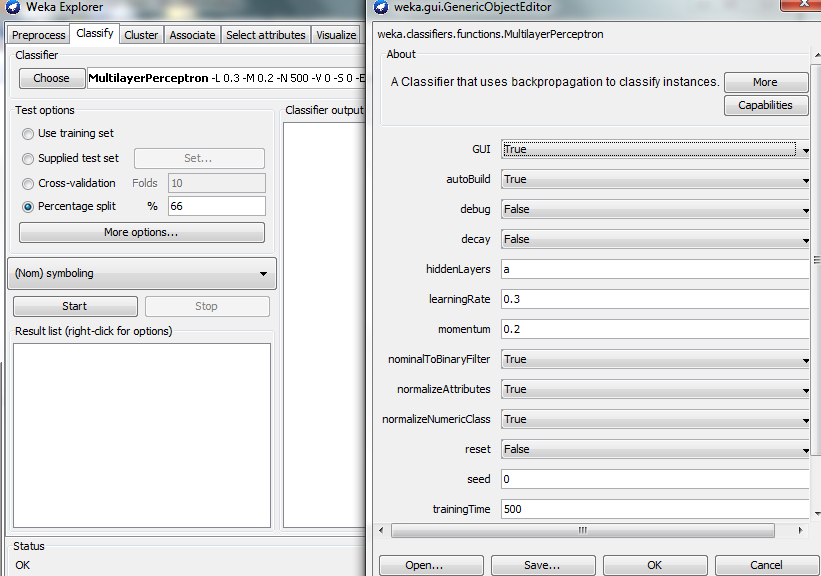
Each strip shows the distribution of values for an attribute

Values for the fuel type attribute (X-axis).

Figure : Change x and y attributes

Weka lets us build a graphical representation of the Neural Network and enables modifying initial values for learning rate and for momentum interactively before training the network. In addition, it is possible to add or to remove the network nodes.

1. Click on a textbox next to the **Choose** button to open the **GenericObjectEditor** dialog box on Figure 20. Set **GUI**to **True** and Click **OK**.
2. Click on **Start** to rerun the algorithm. The window with a network graphical representation on Figure 21 will open.



Use percentage split training option

Initial training time value

Initial learning Rate and momentum values that can be edited through graphical interface before network training

Click to open the object editor

Set GUI option to True

Click ok to apply the settings changes

Figure : Set GUI=true

This network consists of three layers:

* one input layer on the left with one rectangular box for each attribute (colored green);
  + no calculations are performed at this layer
  + the nodes output=the nodes input
* zero ore more hidden layers (the nodes in red) to which all the input nodes are connected;
* one output layer on the right (orange);

The nodes at each layer are not connected to each other, and there are no direct connections between input and output layer. The number of outputs may or may not be equal to the number of inputs.

The output from each hidden layer node is a sigmoid/activation function of the weighted sum of node inputs. The weights are adjusted for each iteration based on the error function for the last run.

\*

Input 1

Hidden Layer Node

Weight 1

Output

Sigmoid function of the sum of weighted inputs

Weight 2

\*

Input 2

Weight 3

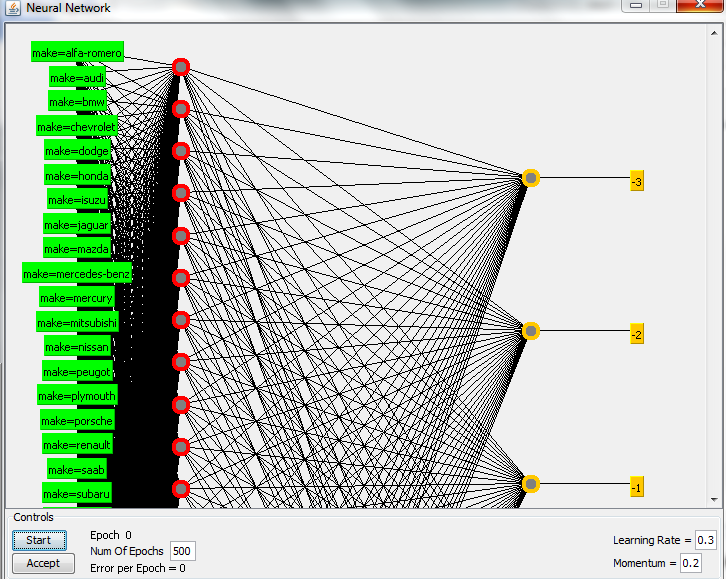
\*

Input 3

The labels at the far right show the symboling class that each output node represents.

This **MultilayerPerceptron** function displays:

* input nodes (in green)
* output nodes (in yellow) nodes broken down into layers
* hidden nodes (in red)
* learning rate
* momentum
* training time



Click to start accept the training result

**Current epoch during training**

Click to start training the network

**Learning rate and momentum can be modified before clicking start to train the network**

**Training time**

**Hidden layer nodes**

**Class labels**

**Input nodes**

**Output nodes – an output is a weighted sum of inputs.**

An error for the current epoch during the network training, and the value changes when the current epoch changes

Figure : Network Model

To delete a network node before starting the network training, right click on the node. All connections to the node will be deleted when the node is deleted.

To add a node before starting the network training, left click on an empty space.

# 6.0 Comparison with J48 Algorithm Results from Weed 3 Exercise

|  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| Algorithm/Test option | Correctly classified Instances (%) | Root Mean Squared Error | Kappa statistic | Roc Area by Class | | | | | | | Discretized numeric attributes |
| J48 - Percentage split | 74.6269% | 0.2588 | 0.6599 | -3 | -2 | -1 | 0 | 1 | 2 | 3 | Yes |
| n/a | n/a | 0.958 | 0.882 | 0.791 | 0.813 | 0.981 |
| J48- Cross-validation | 76.6497 % | 0.2241 | 0.6982 | -3 | -2 | -1 | 0 | 1 | 2 | 3 | Yes |
| n/a | 0.96 | 0.936 | 0.94 | 0.872 | 0.912 | 0.975 |
| J48 -Use training set | 87.3096 % | 0.1633 | 0.835 | -3 | -2 | -1 | 0 | 1 | 2 | 3 | Yes |
| n/a | 0.979 | 0.993 | 0.992 | 0.956 | 0.975 | 0.993 |
| Multilayer Perception - Percentage split | 74.6269 % | 0.2443 | 0.6666 | -3 | -2 | -1 | 0 | 1 | 2 | 3 | No |
| n/a | n/a | 0.947 | 0.911 | 0.779 | 0.926 | 1 |
| Multilayer Perception - Percentage split | ? | ? | ? | -3 | -2 | -1 | 0 | 1 | 2 | 3 | Yes |
|  |  |  |  |  |  |  |
| Multilayer Perception – Cross validation | ? | ? | ? | -3 | -2 | -1 | 0 | 1 | 2 | 3 | No |
|  |  |  |  |  |  |  |
| Multilayer Perception – Cross validation | ? | ? | ? | -3 | -2 | -1 | 0 | 1 | 2 | 3 | Yes |
|  |  |  |  |  |  |  |

Which classification algorithm is superior for the dataset? If we run the multilayer perceptron exercise on a discretized file, do we expect the same results? What happens if we use a different test option? What classes each algorithm does and does not predict?