Data Mining with R

วิชา การค้นพบองค์ความรู้และการทำเหมืองข้อมูลชั้นสูง

Veerasak Kritsanapraphan

Chulalongkorn University

Email : veerasak.kr568@cbs.chula.ac.th

@veerasakk

Agenda

- Overview and Data Visualization
- Data Preparation
- Predictive Data Mining
    - Decision Tree
    - K-Nearest Neighbor
    - Naive Bayes Classifier
    - Neural Network

Slide and Source Codes

https://github.com/vkrit/chula_datamining

Overview

- Predictive Data Mining
  - Two Phases of Processing
    - Training Phase : Learn a model from training data
    - Predicting Phase : Deploy the model to production and use that to predict the future outcome

Data

- Iris Data Set from UCI Machine Learning Repository (https://archive.ics.uci.edu/ml/datasets/Iris)

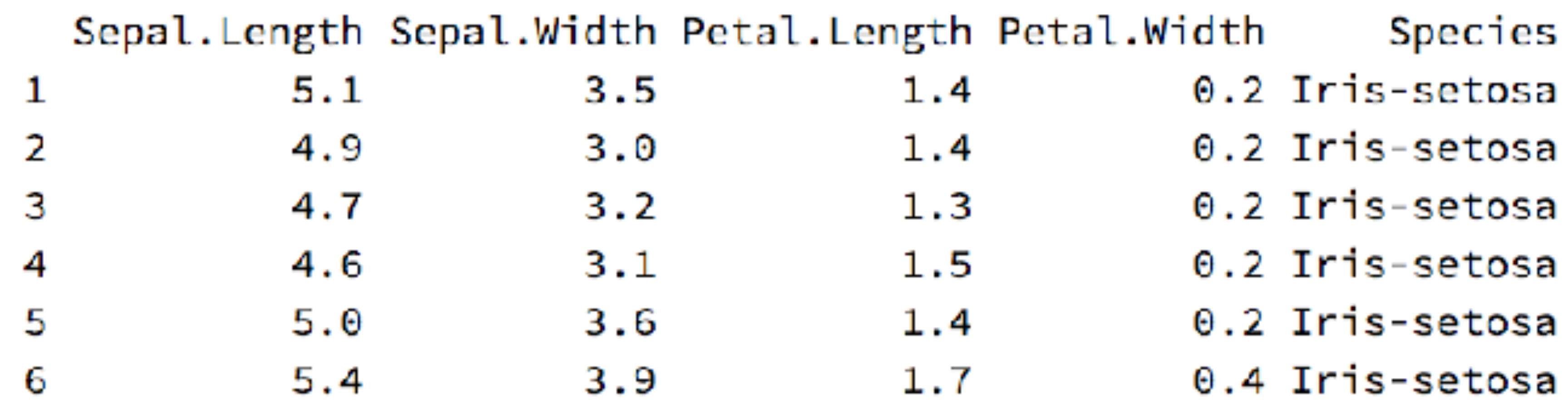
Attribute Information:

1. Sepal Length in cm
2. Sepal width in cm
3. Petal length in cm
4. Petal length in cm
5. Classes:
   - Iris Setosa
   - Iris Versicolour
   - Iris Virginica

UCI Machine Learning Repository
Center for Machine Learning and Intelligent Systems

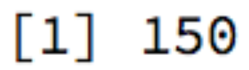
Getting Data

```
> iris <- read.csv("iris.data.csv", header=TRUE)

> head(iris)
```



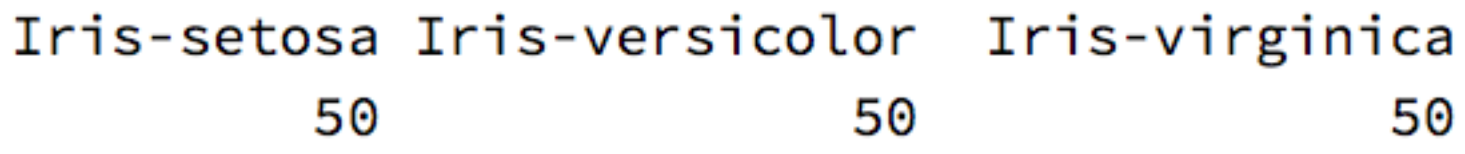
```
  Sepal.Length Sepal.Width Petal.Length Petal.Width     Species
1          5.1         3.5          1.4         0.2 Iris-setosa
2          4.9         3.0          1.4         0.2 Iris-setosa
3          4.7         3.2          1.3         0.2 Iris-setosa
4          4.6         3.1          1.5         0.2 Iris-setosa
5          5.0         3.6          1.4         0.2 Iris-setosa
6          5.4         3.9          1.7         0.4 Iris-setosa
```

```
> nrow(iris)
```



```
[1] 150
```

```
> table(iris$Species)
```



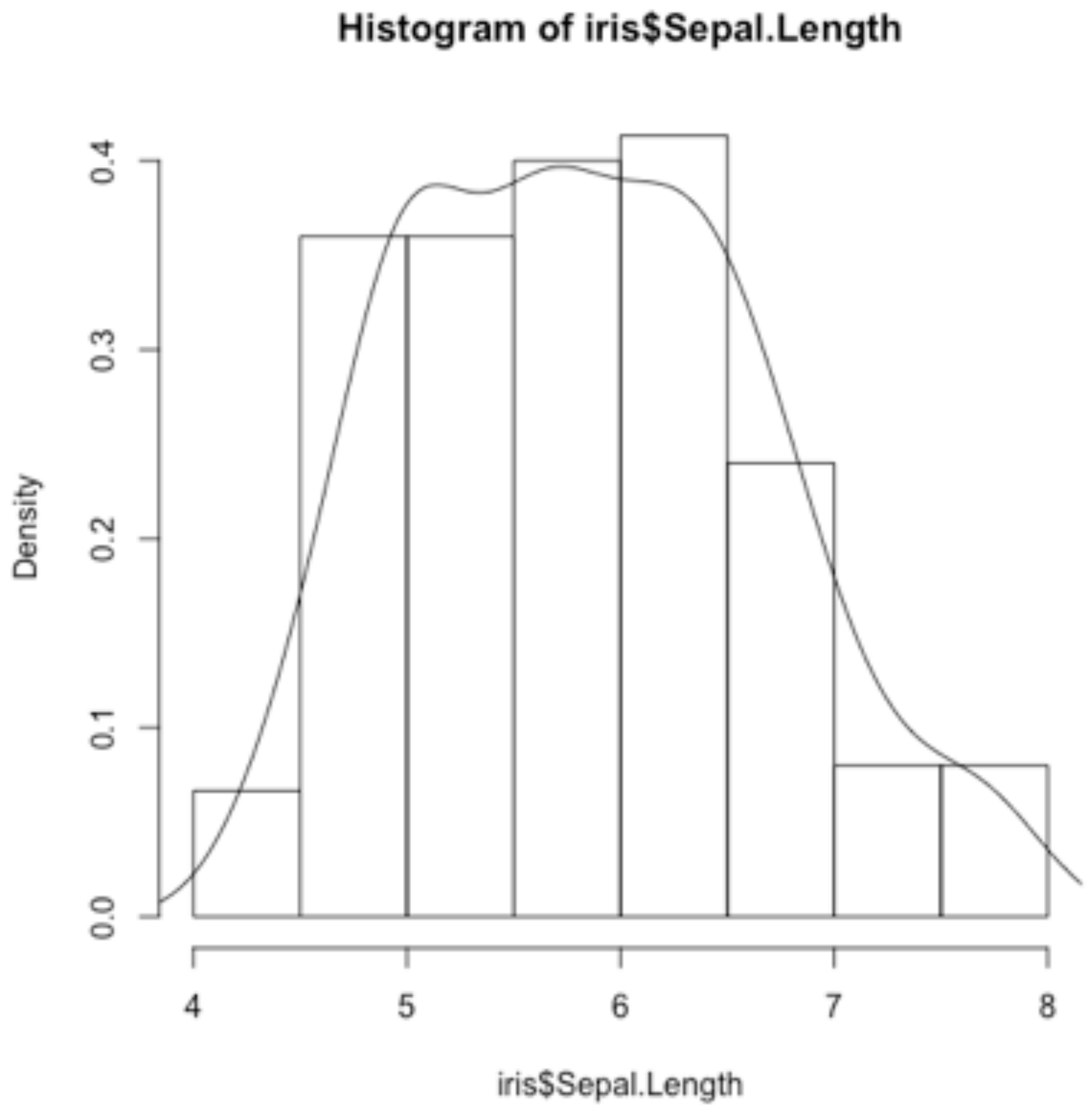
```
 Iris-setosa Iris-versicolor  Iris-virginica
          50              50              50
```

Data Visualization

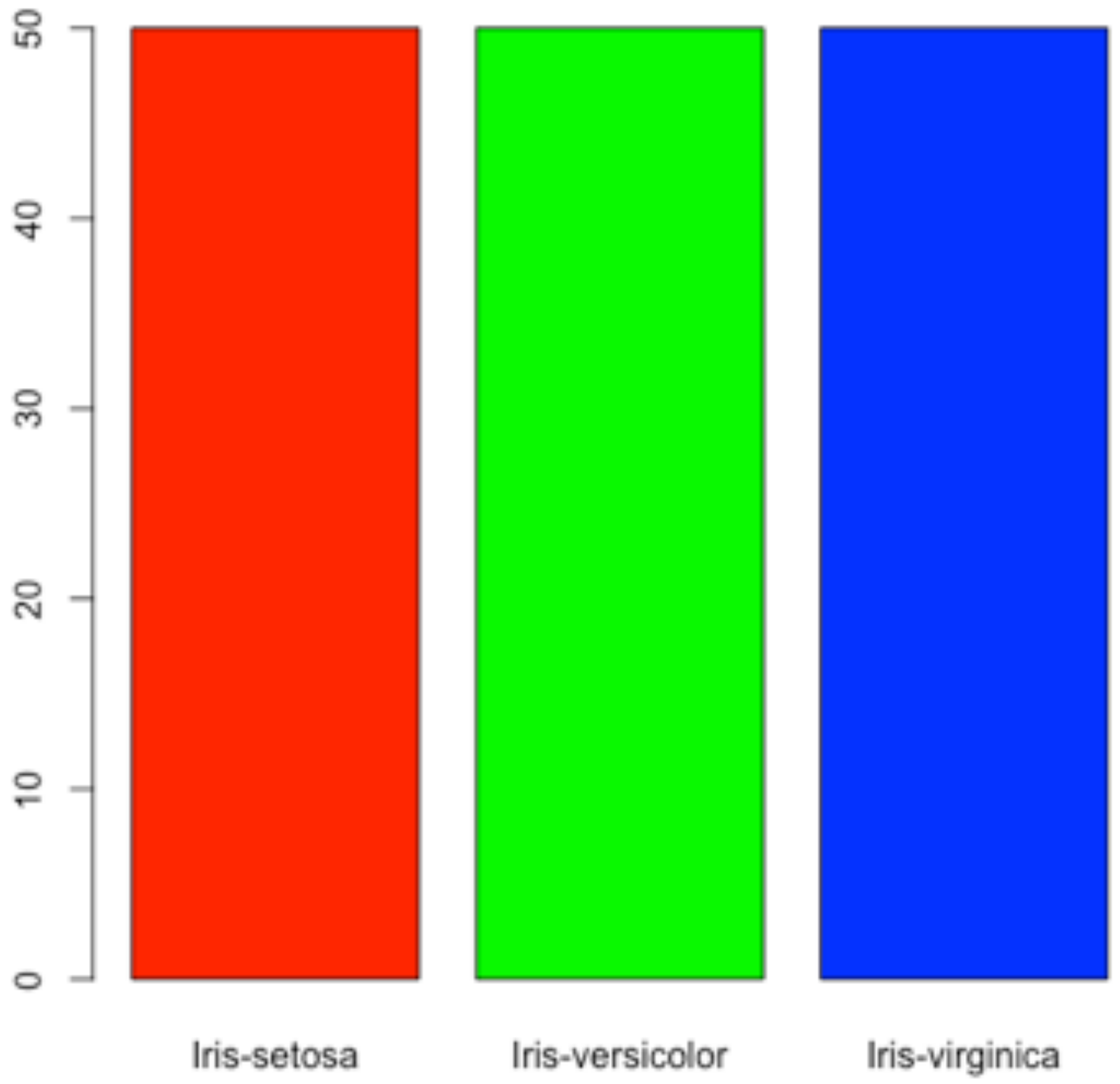
- Visualizing existing data is a very useful way to come up with ideas about what features should be included.

- "Dataframe" in R is a common way where data samples are organized in a tabular structure.

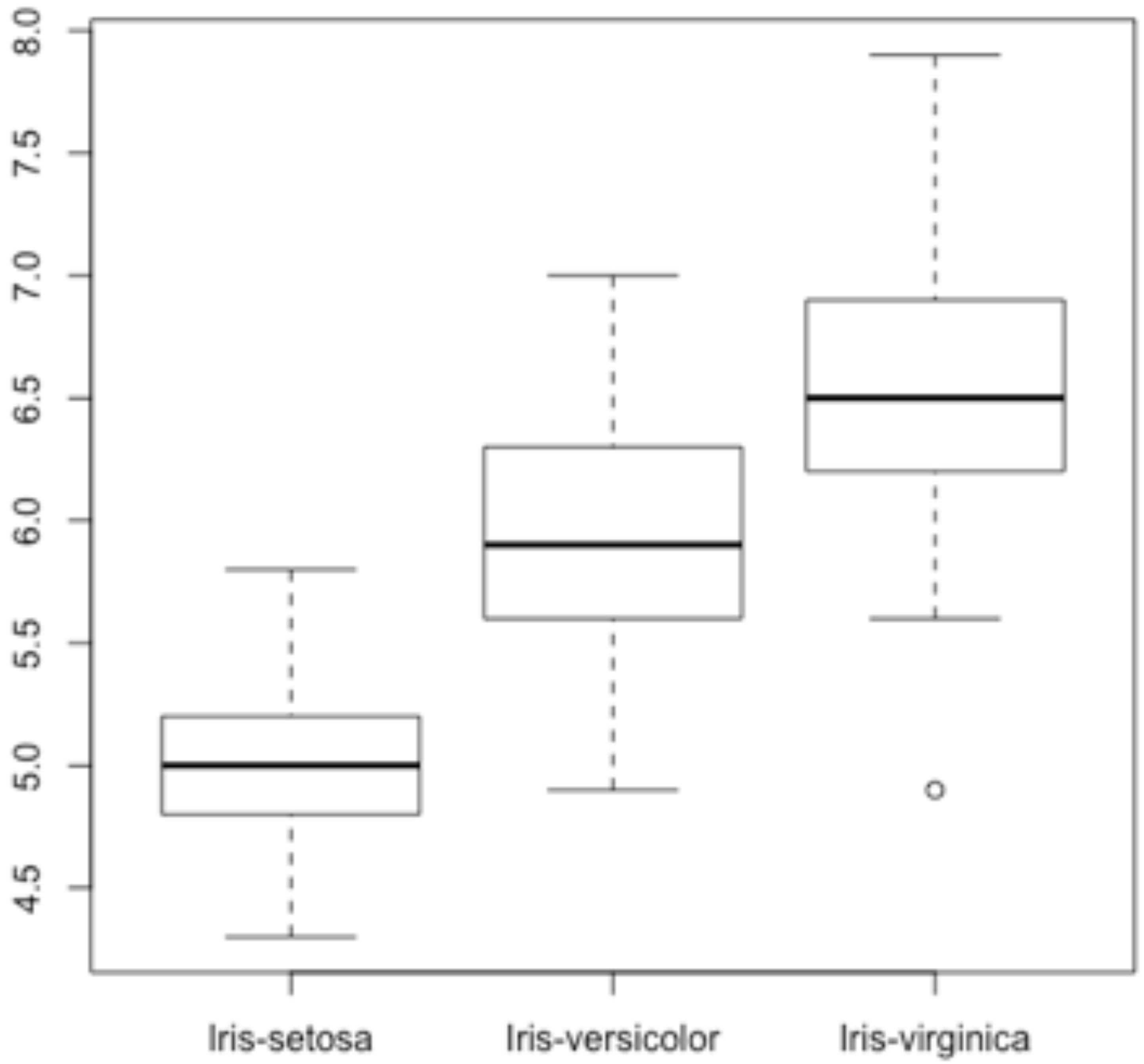
```
> hist(iris$Sepal.Length, breaks = 10, prob=T)

> lines(density(iris$Sepal.Length))  # density curve
```



```
> categories <- table(iris$Species)

> barplot(categories, col=c('red', 'green', ‘blue'))
```

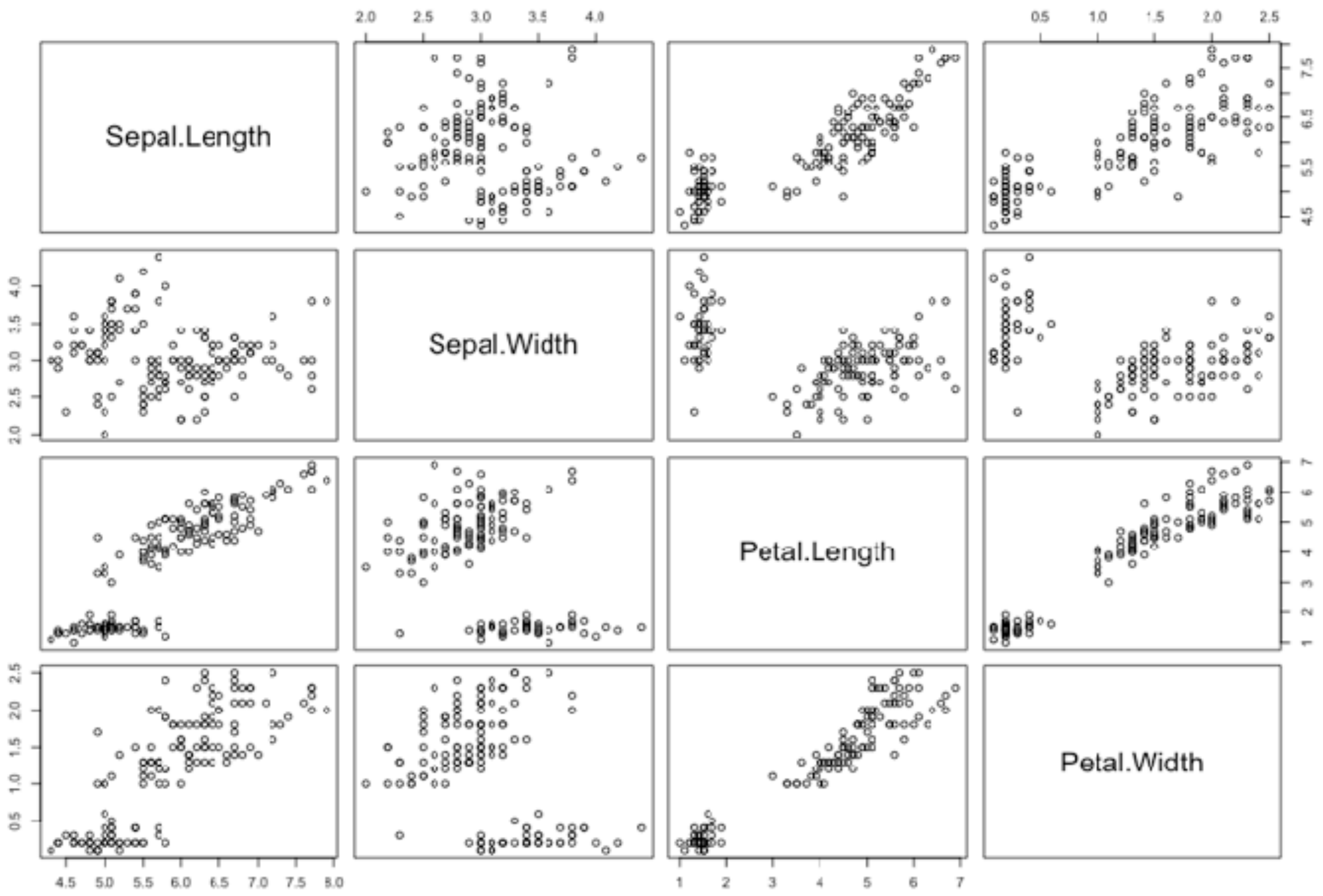


```
> boxplot(Sepal.Length~Species, data = iris)
```



```
> pairs(iris[, c(1,2,3,4)]
> cor(iris[, c(1,2,3,4)]
```

```
             Sepal.Length Sepal.Width Petal.Length Petal.Width
Sepal.Length    1.0000000  -0.1093692    0.8717542   0.8179536
Sepal.Width    -0.1093692   1.0000000   -0.4205161  -0.3565441
Petal.Length    0.8717542  -0.4205161    1.0000000   0.9627571
Petal.Width     0.8179536  -0.3565441    0.9627571   1.0000000
```

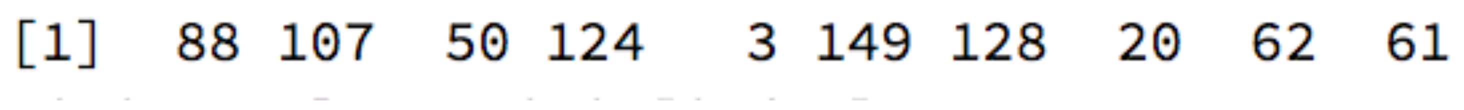


Preparing Training Data

- At this step, the purpose is to transform the raw data in a form that can fit into the data mining model.
  - Data sampling
  - Data validation and handle missing data
  - Normalize numeric value into a uniform range
  - Binarize categorical field to binary fields
  - Discretize numeric value into categories
  - Create derived fields from existing fields
  - Reduce dimensionality
  - Log transformation

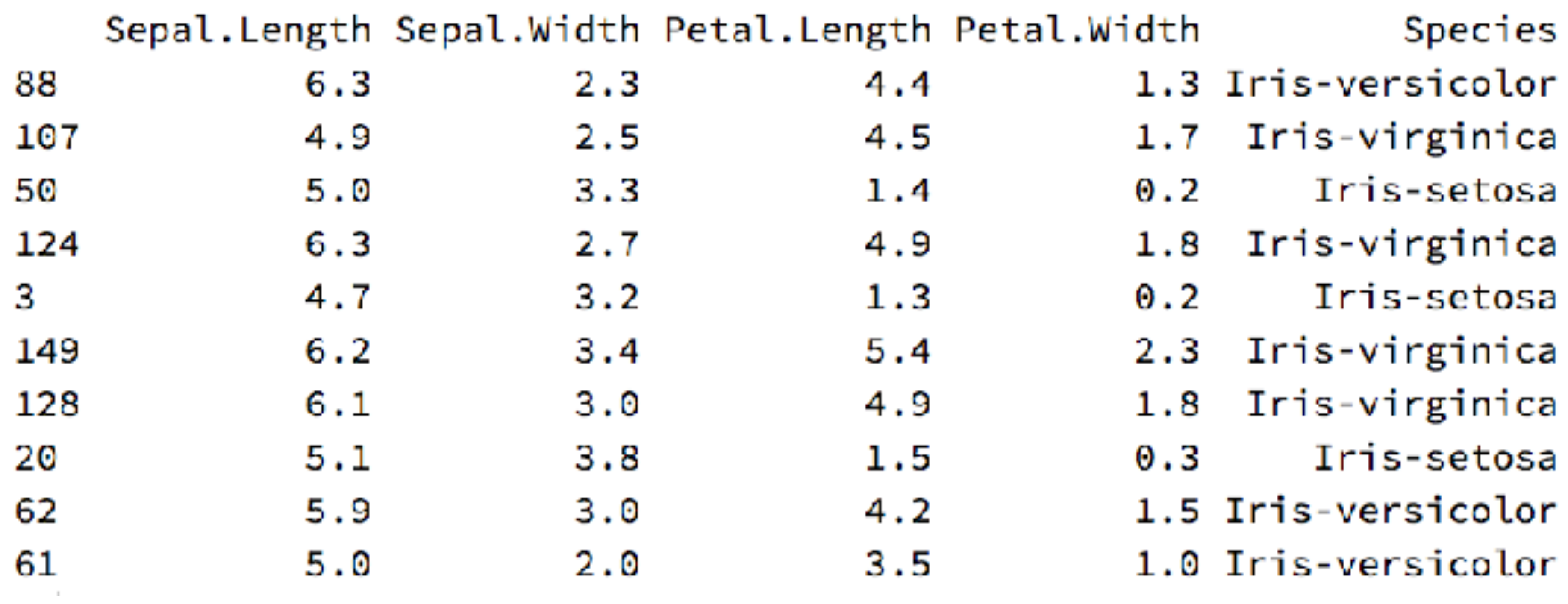
Data Sampling

```
> index <- sample(1:nrow(iris), 10, replace=T)
> index
```



```
[1]  88 107  50 124   3 149 128  20  62  61
```

```
> irissample <- iris[index, ]
> irissample
```



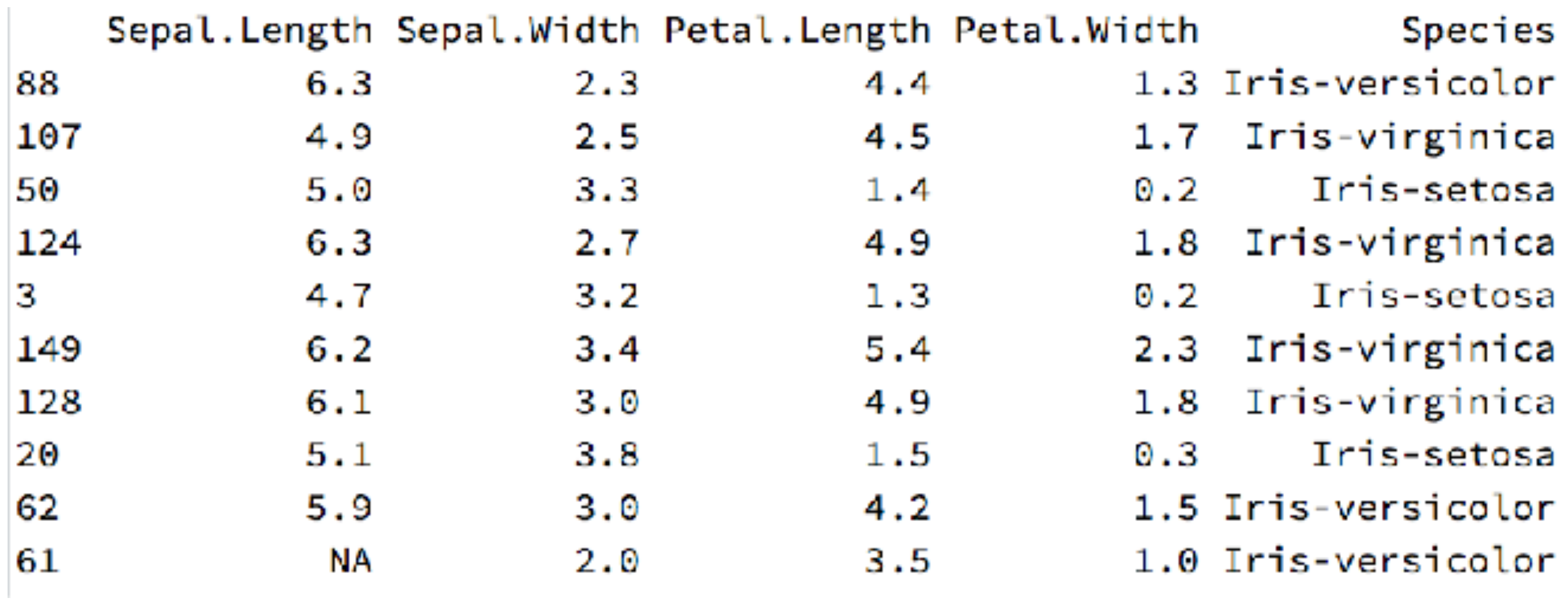
```
    Sepal.Length Sepal.Width Petal.Length Petal.Width         Species
88           6.3         2.3          4.4         1.3 Iris-versicolor
107          4.9         2.5          4.5         1.7  Iris-virginica
50           5.0         3.3          1.4         0.2     Iris-setosa
124          6.3         2.7          4.9         1.8  Iris-virginica
3            4.7         3.2          1.3         0.2     Iris-setosa
149          6.2         3.4          5.4         2.3  Iris-virginica
128          6.1         3.0          4.9         1.8  Iris-virginica
20           5.1         3.8          1.5         0.3     Iris-setosa
62           5.9         3.0          4.2         1.5 Iris-versicolor
61           5.0         2.0          3.5         1.0 Iris-versicolor
```

Impute missing data

- Discard the whole record

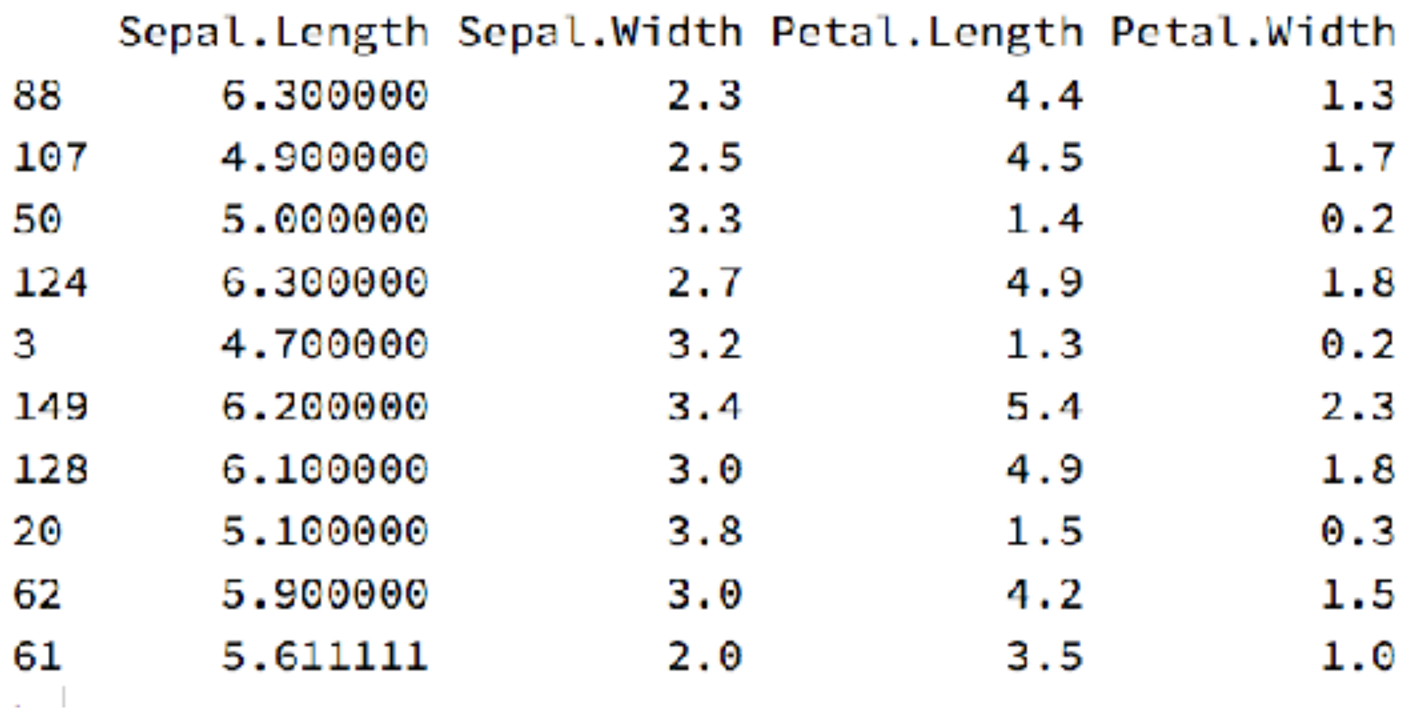
- Infer missing value based on the data of other record. Approach is to fill the missing data with the average or the median.

```
> irissample[10, 1] <- NA
> irissample
```



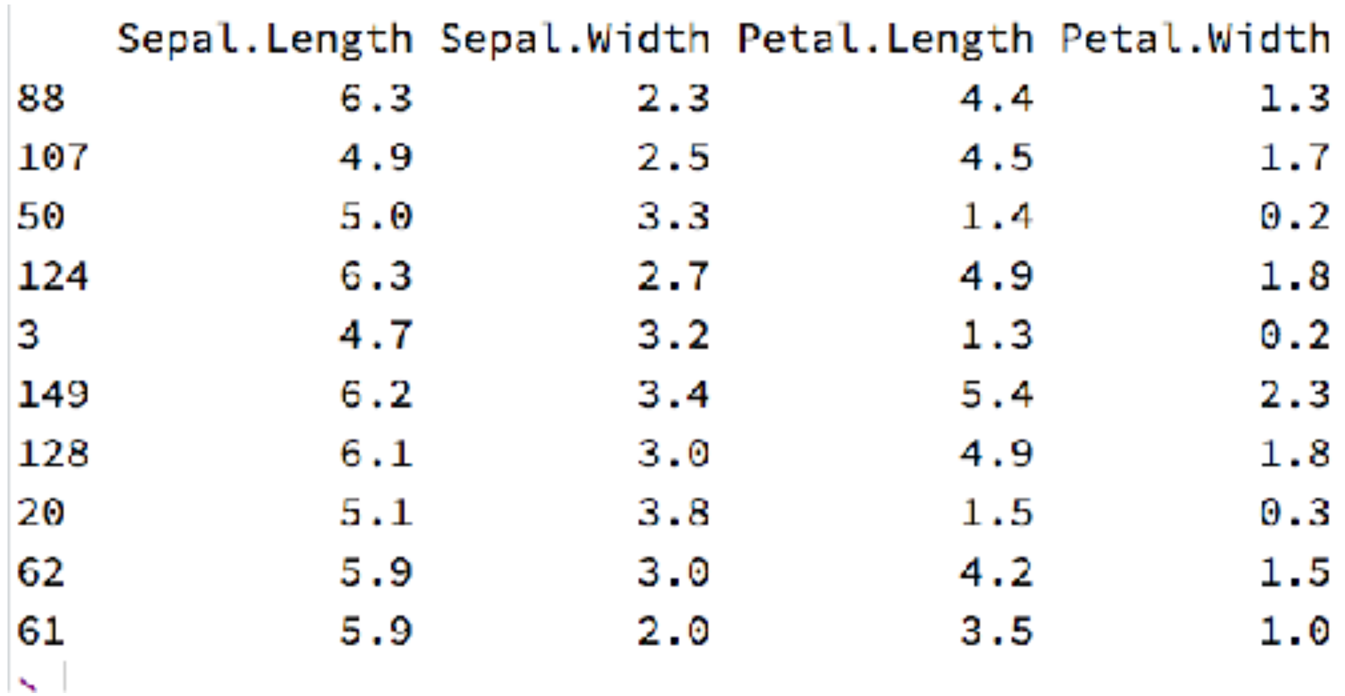
```
    Sepal.Length Sepal.Width Petal.Length Petal.Width         Species
88           6.3         2.3          4.4         1.3 Iris-versicolor
107          4.9         2.5          4.5         1.7  Iris-virginica
50           5.0         3.3          1.4         0.2     Iris-setosa
124          6.3         2.7          4.9         1.8  Iris-virginica
3            4.7         3.2          1.3         0.2     Iris-setosa
149          6.2         3.4          5.4         2.3  Iris-virginica
128          6.1         3.0          4.9         1.8  Iris-virginica
20           5.1         3.8          1.5         0.3     Iris-setosa
62           5.9         3.0          4.2         1.5 Iris-versicolor
61            NA         2.0          3.5         1.0 Iris-versicolor
```

```
> library(e1071)
> fixIris1 <- impute(irissample[, 1:4], what = ‘mean')
> fixIris1
```



```
    Sepal.Length Sepal.Width Petal.Length Petal.Width
88      6.300000         2.3          4.4         1.3
107     4.900000         2.5          4.5         1.7
50      5.000000         3.3          1.4         0.2
124     6.300000         2.7          4.9         1.8
3       4.700000         3.2          1.3         0.2
149     6.200000         3.4          5.4         2.3
128     6.100000         3.0          4.9         1.8
20      5.100000         3.8          1.5         0.3
62      5.900000         3.0          4.2         1.5
61      5.611111         2.0          3.5         1.0
```

```
> fixIris2 <- impute(irissample[, 1:4], what = ‘median’)
> fixIris2
```

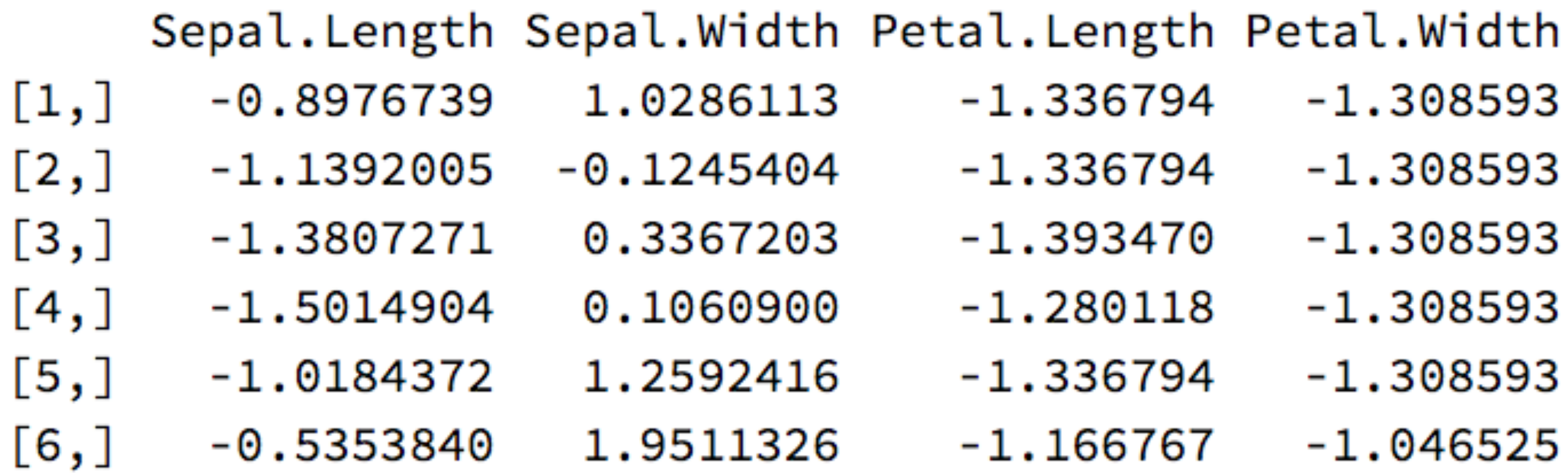


```
    Sepal.Length Sepal.Width Petal.Length Petal.Width
88           6.3         2.3          4.4         1.3
107          4.9         2.5          4.5         1.7
50           5.0         3.3          1.4         0.2
124          6.3         2.7          4.9         1.8
3            4.7         3.2          1.3         0.2
149          6.2         3.4          5.4         2.3
128          6.1         3.0          4.9         1.8
20           5.1         3.8          1.5         0.3
62           5.9         3.0          4.2         1.5
61           5.9         2.0          3.5         1.0
```

Normalize numeric value

- scale the column using x - mean(x) / std

```
> scaleiris <- scale(iris[, 1:4])
> head(scaleiris)
```



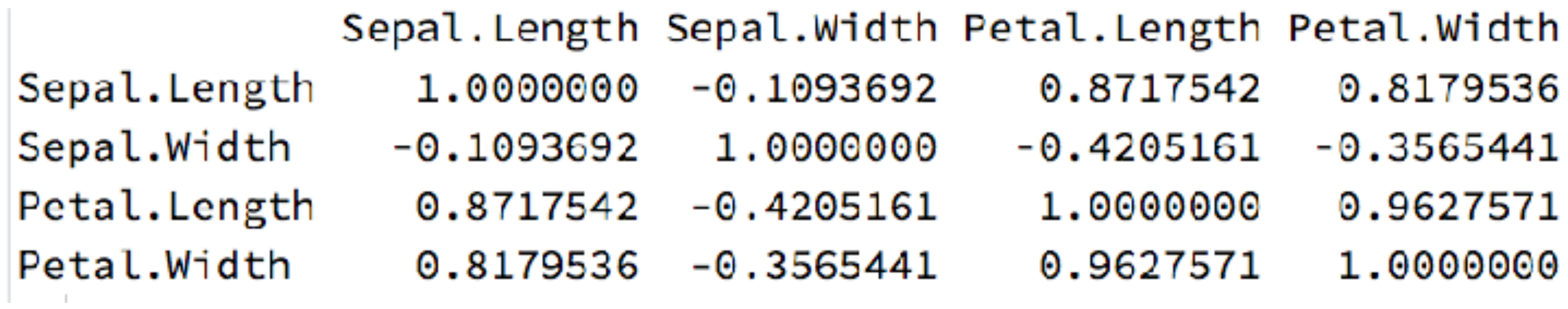
```
     Sepal.Length Sepal.Width Petal.Length Petal.Width
[1,]   -0.8976739   1.0286113    -1.336794   -1.308593
[2,]   -1.1392005  -0.1245404    -1.336794   -1.308593
[3,]   -1.3807271   0.3367203    -1.393470   -1.308593
[4,]   -1.5014904   0.1060900    -1.280118   -1.308593
[5,]   -1.0184372   1.2592416    -1.336794   -1.308593
[6,]   -0.5353840   1.9511326    -1.166767   -1.046525
```

Reduce dimensionality

There are two ways to reduce the number of input attributes.

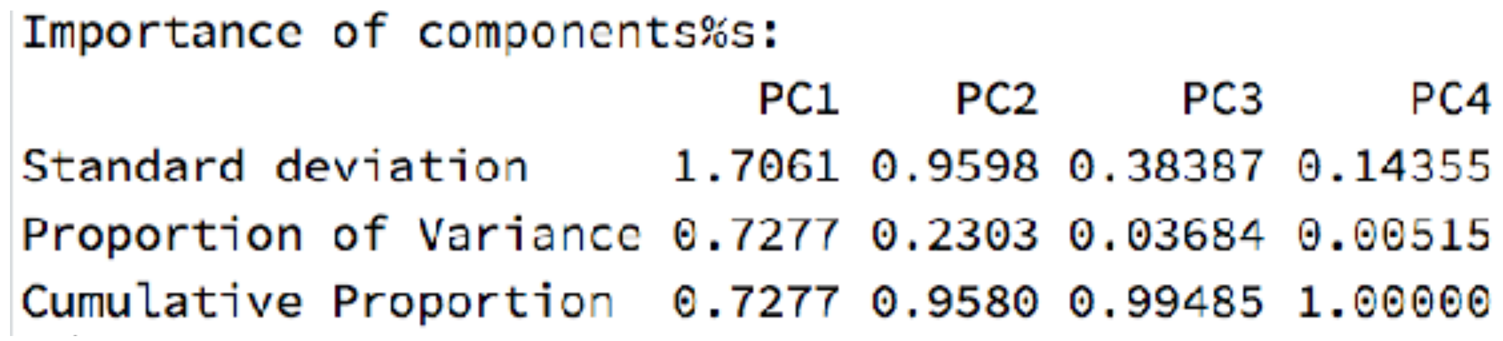
1. Removing irrelevant input variables.
2. Removing redundant input variables.

```
> cor(iris[,-5])
```



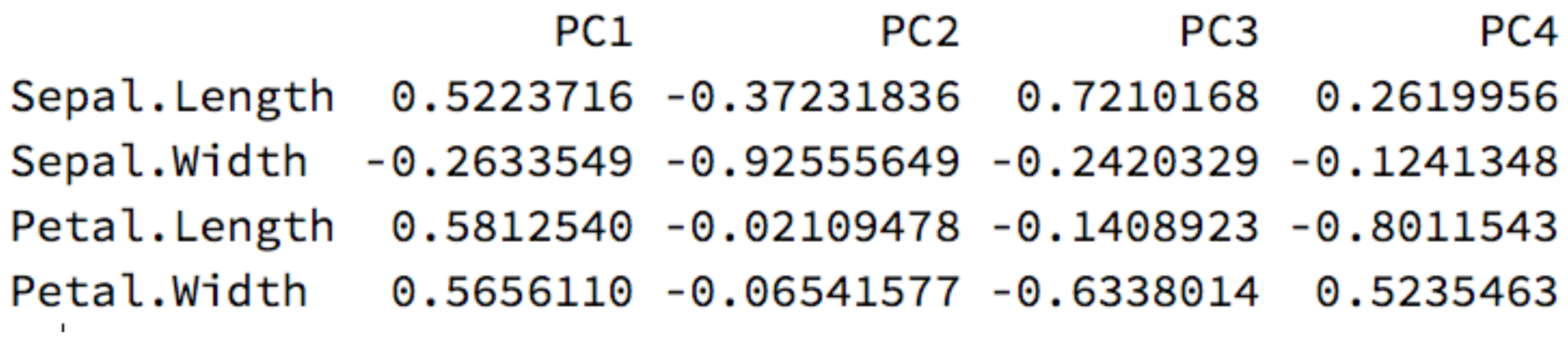
```
             Sepal.Length Sepal.Width Petal.Length Petal.Width
Sepal.Length    1.0000000  -0.1093692    0.8717542   0.8179536
Sepal.Width    -0.1093692   1.0000000   -0.4205161  -0.3565441
Petal.Length    0.8717542  -0.4205161    1.0000000   0.9627571
Petal.Width     0.8179536  -0.3565441    0.9627571   1.0000000
```

```
> pca <- prcomp(iris[, -5], scale=T)
> summary(pca)
```

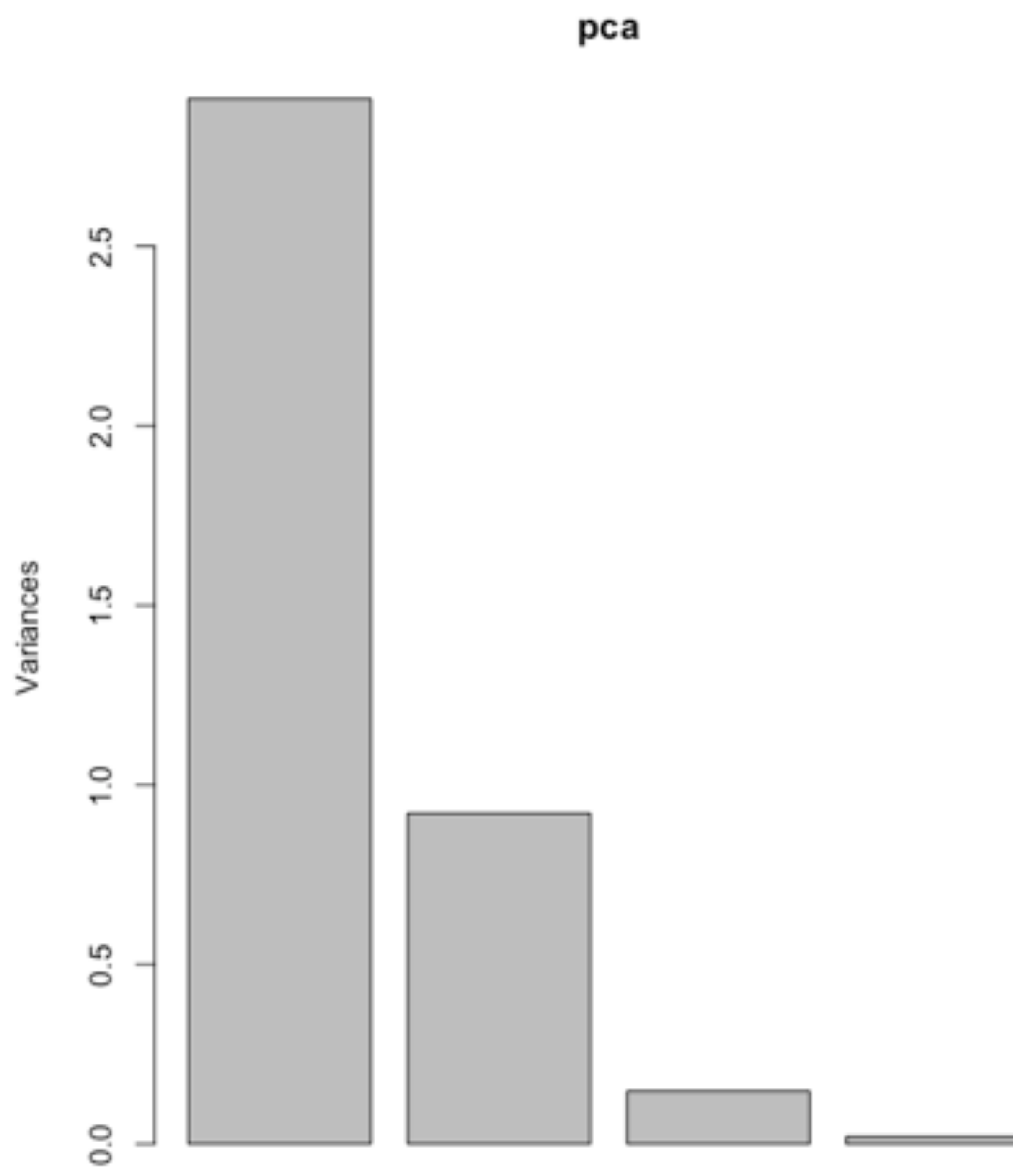


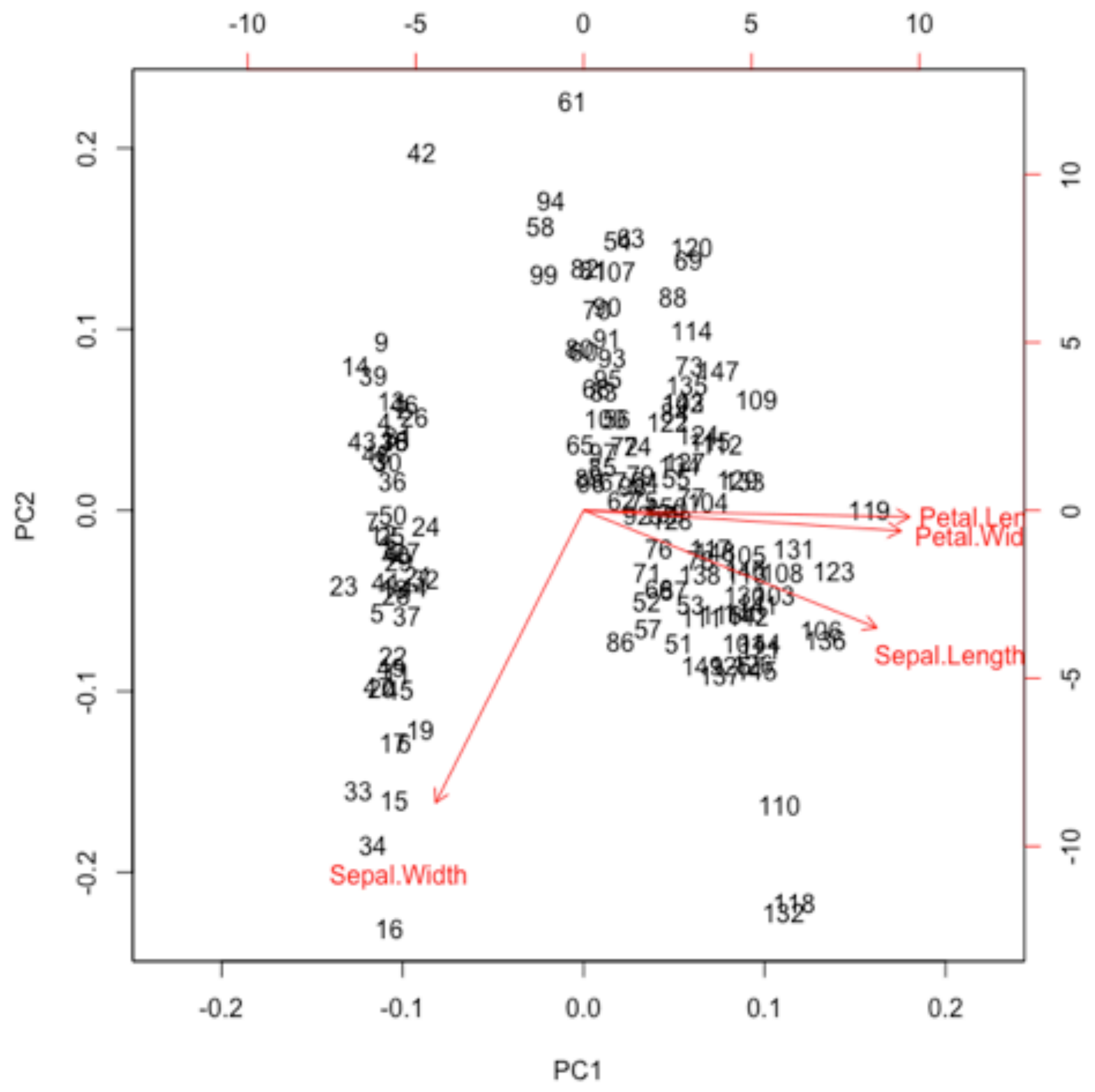
```
Importance of components%s:
                          PC1    PC2     PC3     PC4
Standard deviation     1.7061 0.9598 0.38387 0.14355
Proportion of Variance 0.7277 0.2303 0.03684 0.00515
Cumulative Proportion  0.7277 0.9580 0.99485 1.00000
```

```
> plot(pca)
> pca$rotation
```



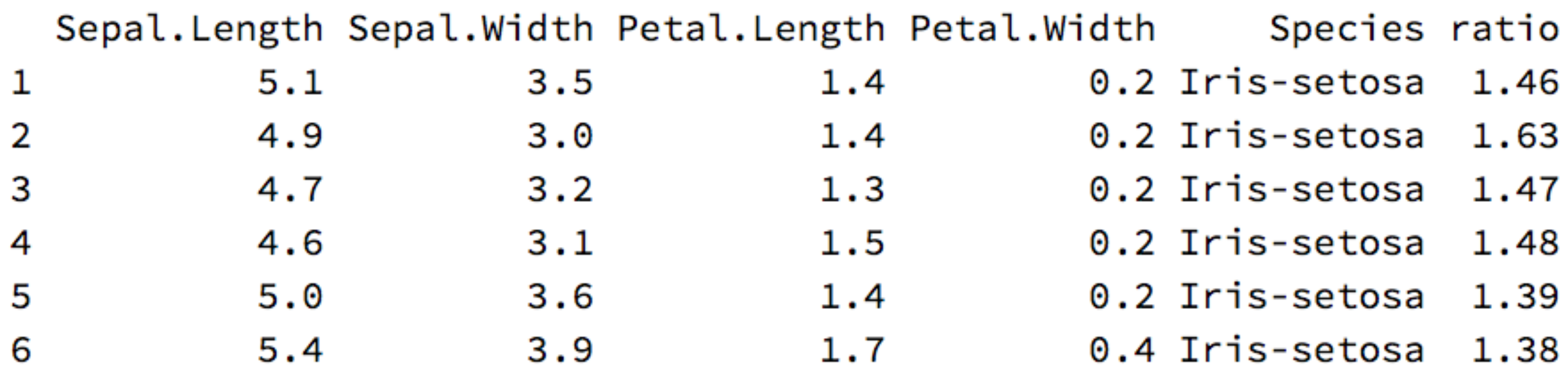
```
                    PC1         PC2        PC3        PC4
Sepal.Length  0.5223716 -0.37231836  0.7210168  0.2619956
Sepal.Width  -0.2633549 -0.92555649 -0.2420329 -0.1241348
Petal.Length  0.5812540 -0.02109478 -0.1408923 -0.8011543
Petal.Width   0.5656110 -0.06541577 -0.6338014  0.5235463
```


pca
Variances
0.0
0.5
1.0
1.5
2.0
2.5


PC1
PC2
Petal.Len
Petal.Wid
Sepal.Length
Sepal.Width
-0.2
-0.1
0.0
0.1
0.2
-10
-5
0
5
10

Add derived attributes

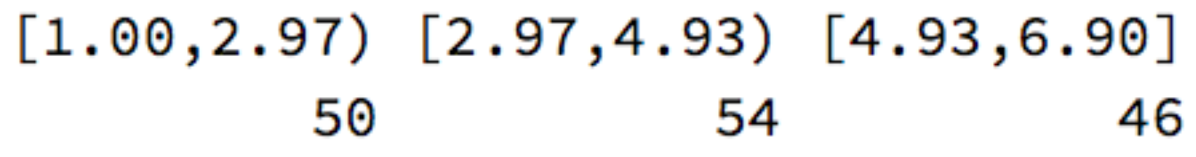
```
> iris2 <- transform(iris, ratio=round(Sepal.Length/Sepal.Width,2))
> head(iris2)
```



```
  Sepal.Length Sepal.Width Petal.Length Petal.Width     Species ratio
1          5.1         3.5          1.4         0.2 Iris-setosa  1.46
2          4.9         3.0          1.4         0.2 Iris-setosa  1.63
3          4.7         3.2          1.3         0.2 Iris-setosa  1.47
4          4.6         3.1          1.5         0.2 Iris-setosa  1.48
5          5.0         3.6          1.4         0.2 Iris-setosa  1.39
6          5.4         3.9          1.7         0.4 Iris-setosa  1.38
```

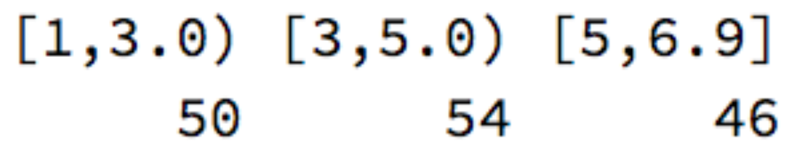
Discretize numeric value into categories

> table(discretize(iris$Petal.Length, categories=3)) # interval width



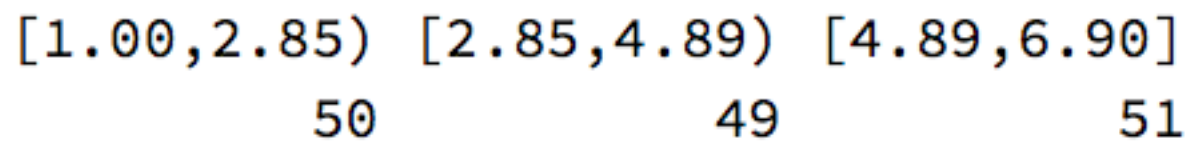
```
[1.00,2.97) [2.97,4.93) [4.93,6.90]
         50          54          46
```

> table(discretize(iris$Petal.Length, "frequency", categories=3)) # equal frequency



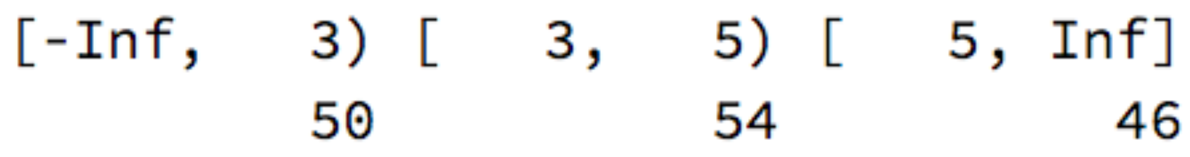
```
[1,3.0) [3,5.0) [5,6.9]
     50      54      46
```

> table(discretize(iris$Petal.Length, "cluster", categories=3)) # k-means



```
[1.00,2.85) [2.85,4.89) [4.89,6.90]
         50          49          51
```

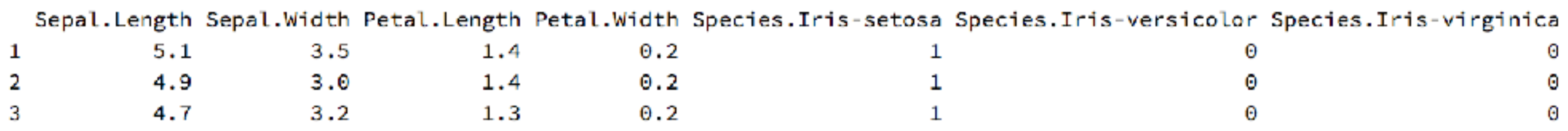
> table(discretize(iris$Petal.Length, "fixed", categories=c(-Inf,3,5,Inf)))



```
[-Inf,    3) [   3,    5) [   5, Inf]
          50           54          46
```

Binarize categorical attributes

```
> install.packages(“dummies")

> library(dummies)

> iris.dummy <- dummy.data.frame(iris, sep = “.")

> head(iris.dummy)
```



```
  Sepal.Length Sepal.Width Petal.Length Petal.Width Species.Iris-setosa Species.Iris-versicolor Species.Iris-virginica
1          5.1         3.5          1.4         0.2                   1                       0                      0
2          4.9         3.0          1.4         0.2                   1                       0                      0
3          4.7         3.2          1.3         0.2                   1                       0                      0
```

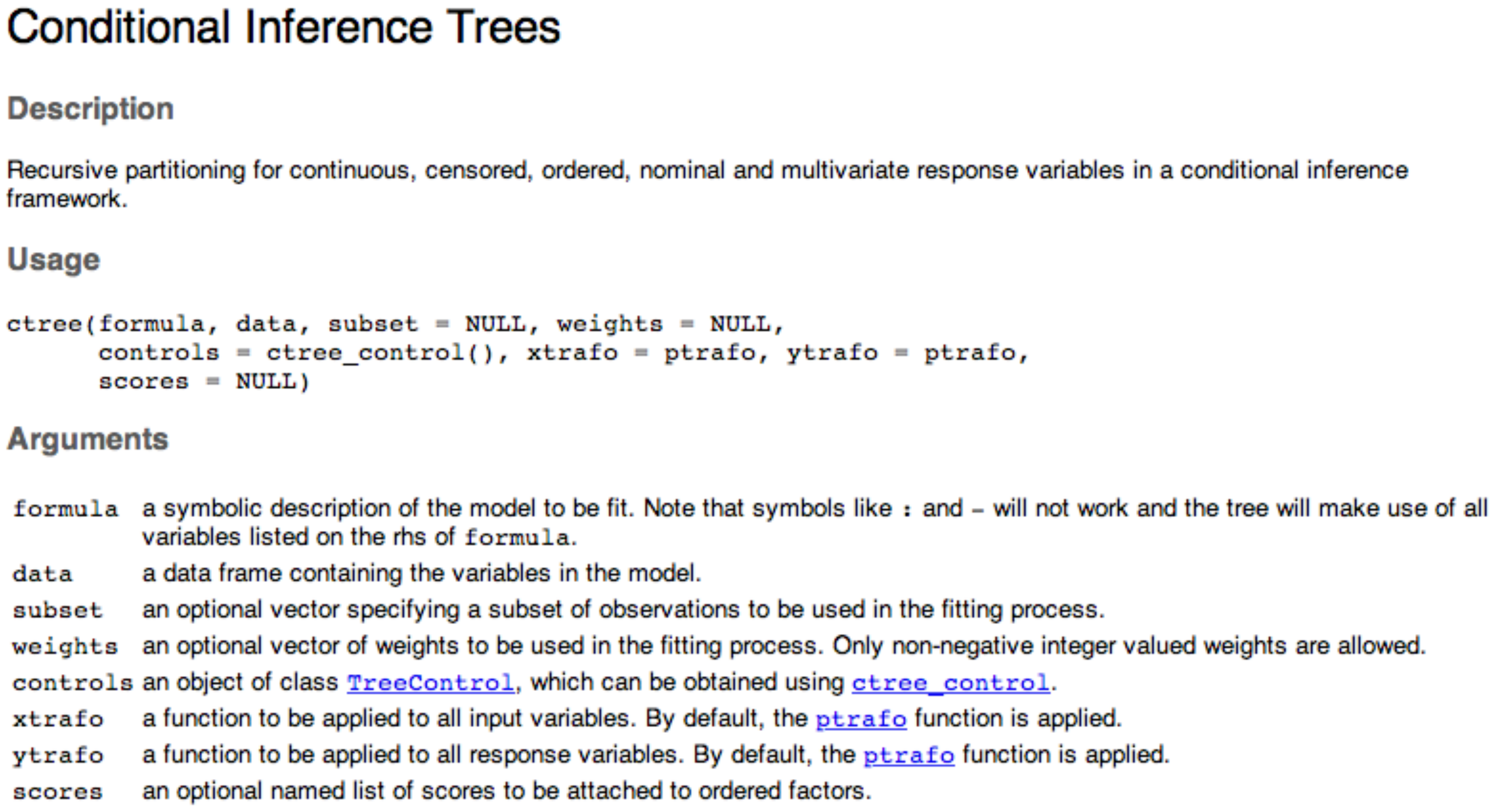
Data Mining

Techniques

Iris Data Preparation

```
> set.seed(1234)

> ind <- sample(2, nrow(iris), replace=TRUE, prob=c(0.7, 0.3))

> trainData <- iris[ind==1,]

> testData <- iris[ind==2,]
```

Decision Tree



Conditional Inference Trees

Description

Recursive partitioning for continuous, censored, ordered, nominal and multivariate response variables in a conditional inference framework.

Usage

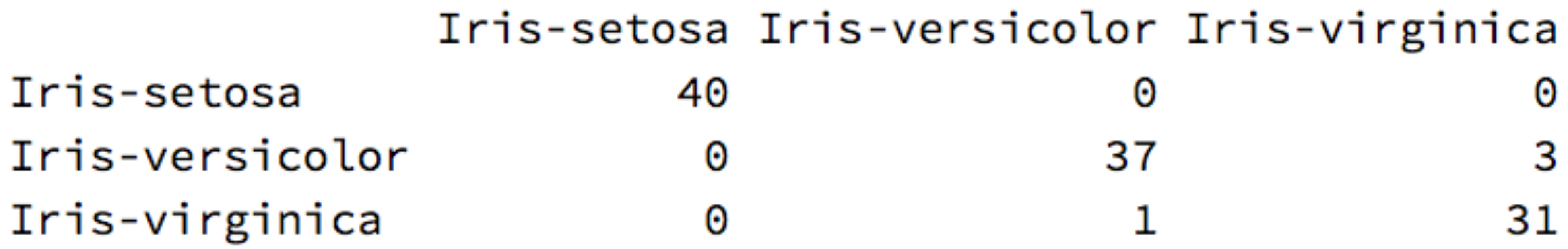
```
ctree(formula, data, subset = NULL, weights = NULL,
      controls = ctree_control(), xtrafo = ptrafo, ytrafo = ptrafo,
      scores = NULL)
```

Arguments

`formula`	a symbolic description of the model to be fit. Note that symbols like `:` and `-` will not work and the tree will make use of all variables listed on the rhs of `formula`.
`data`	a data frame containing the variables in the model.
`subset`	an optional vector specifying a subset of observations to be used in the fitting process.
`weights`	an optional vector of weights to be used in the fitting process. Only non-negative integer valued weights are allowed.
`controls`	an object of class `TreeControl`, which can be obtained using `ctree_control`.
`xtrafo`	a function to be applied to all input variables. By default, the `ptrafo` function is applied.
`ytrafo`	a function to be applied to all response variables. By default, the `ptrafo` function is applied.
`scores`	an optional named list of scores to be attached to ordered factors.

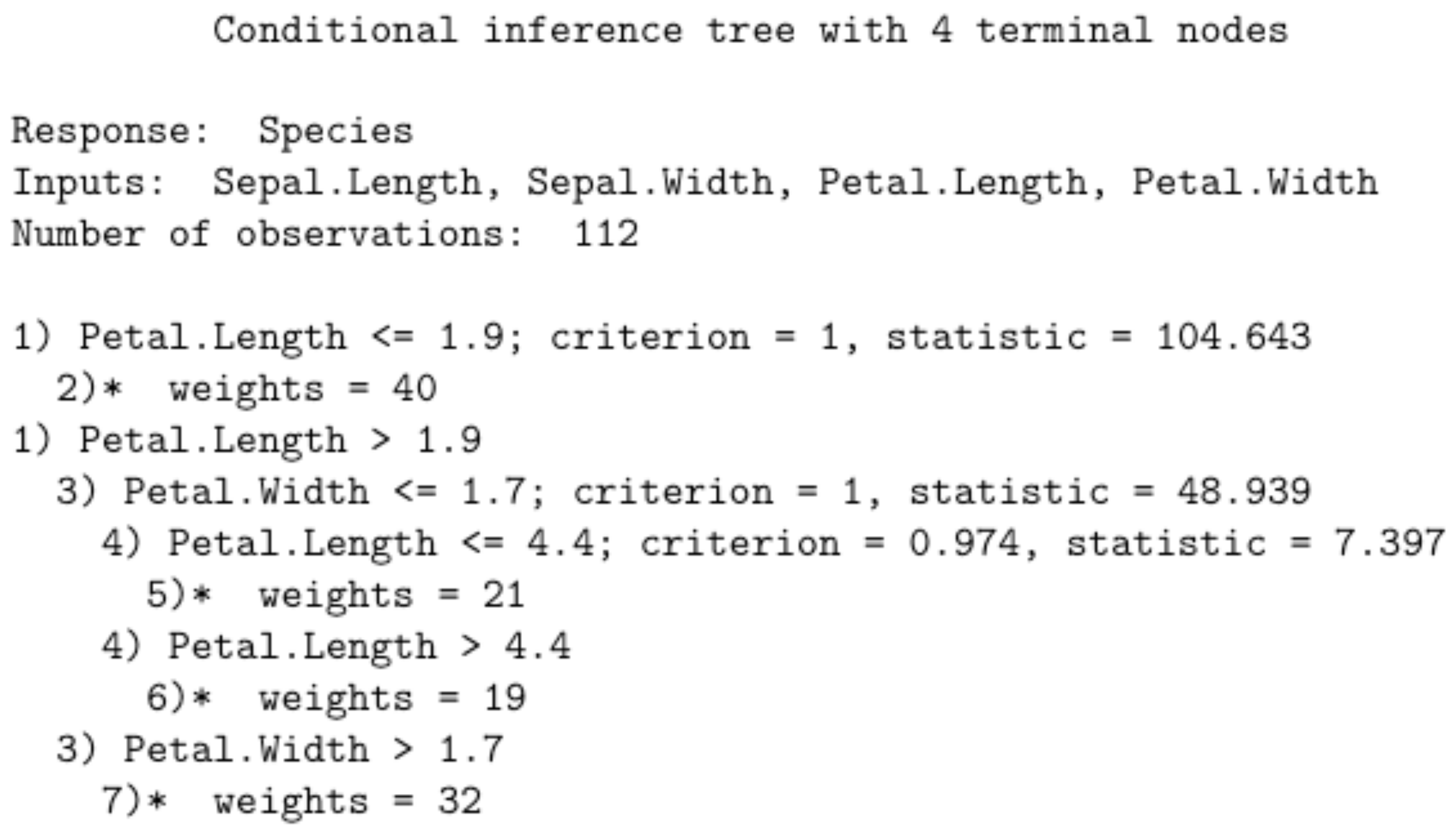
Decision Tree - Create Model

```
> library(party)
> # Create Formula
> my.formula <- Species ~ Sepal.Length + Sepal.Width + Petal.Length + Petal.Width
> # Create Decision Tree Model using ctree function
> iris.ctree <- ctree(my.formula, data = trainData)
> # Predict using Train Data
> table(predict(iris.ctree), trainData$Species)
```



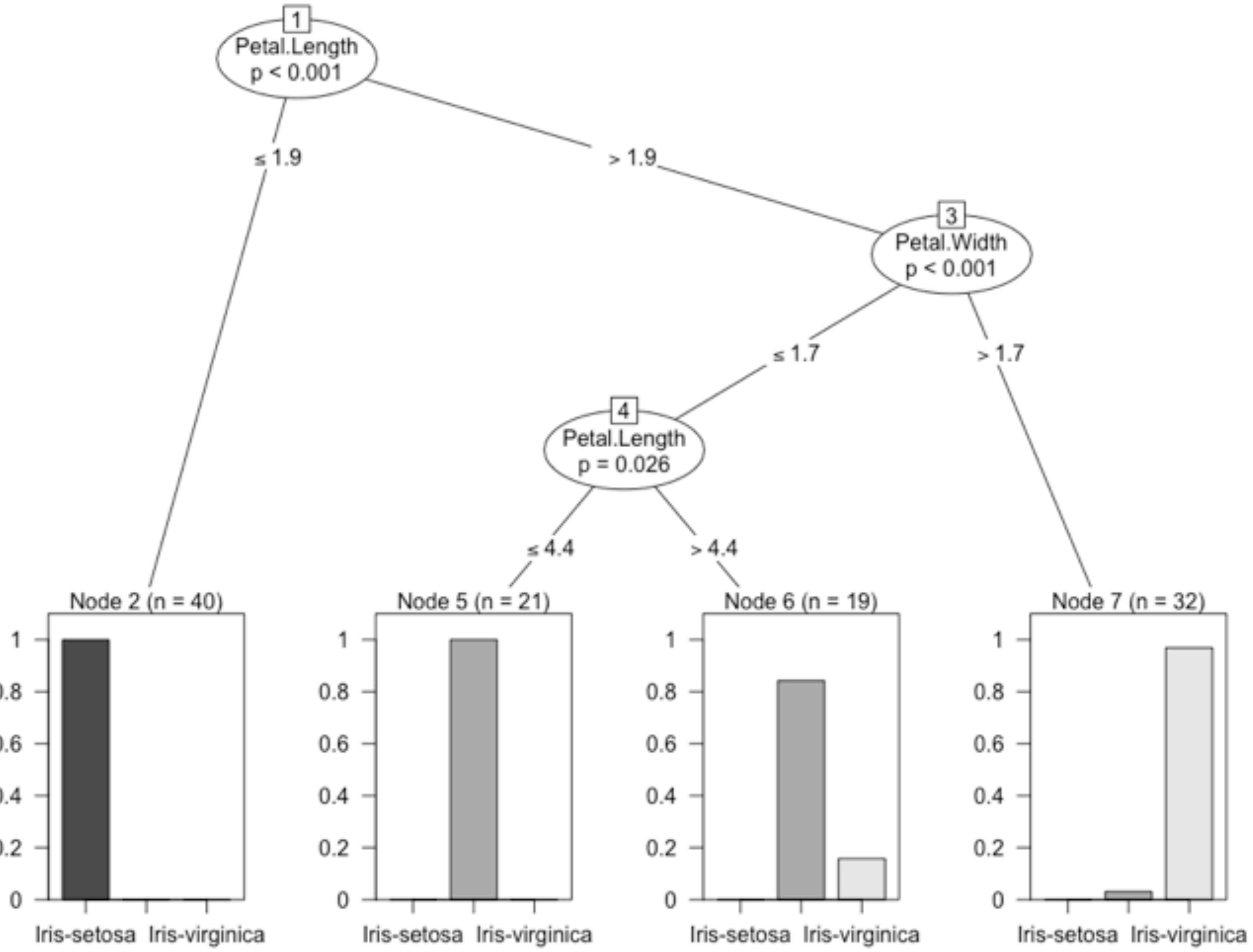
```
                  Iris-setosa Iris-versicolor Iris-virginica
  Iris-setosa              40               0              0
  Iris-versicolor           0              37              3
  Iris-virginica            0               1             31
```

> print(iris.ctree)

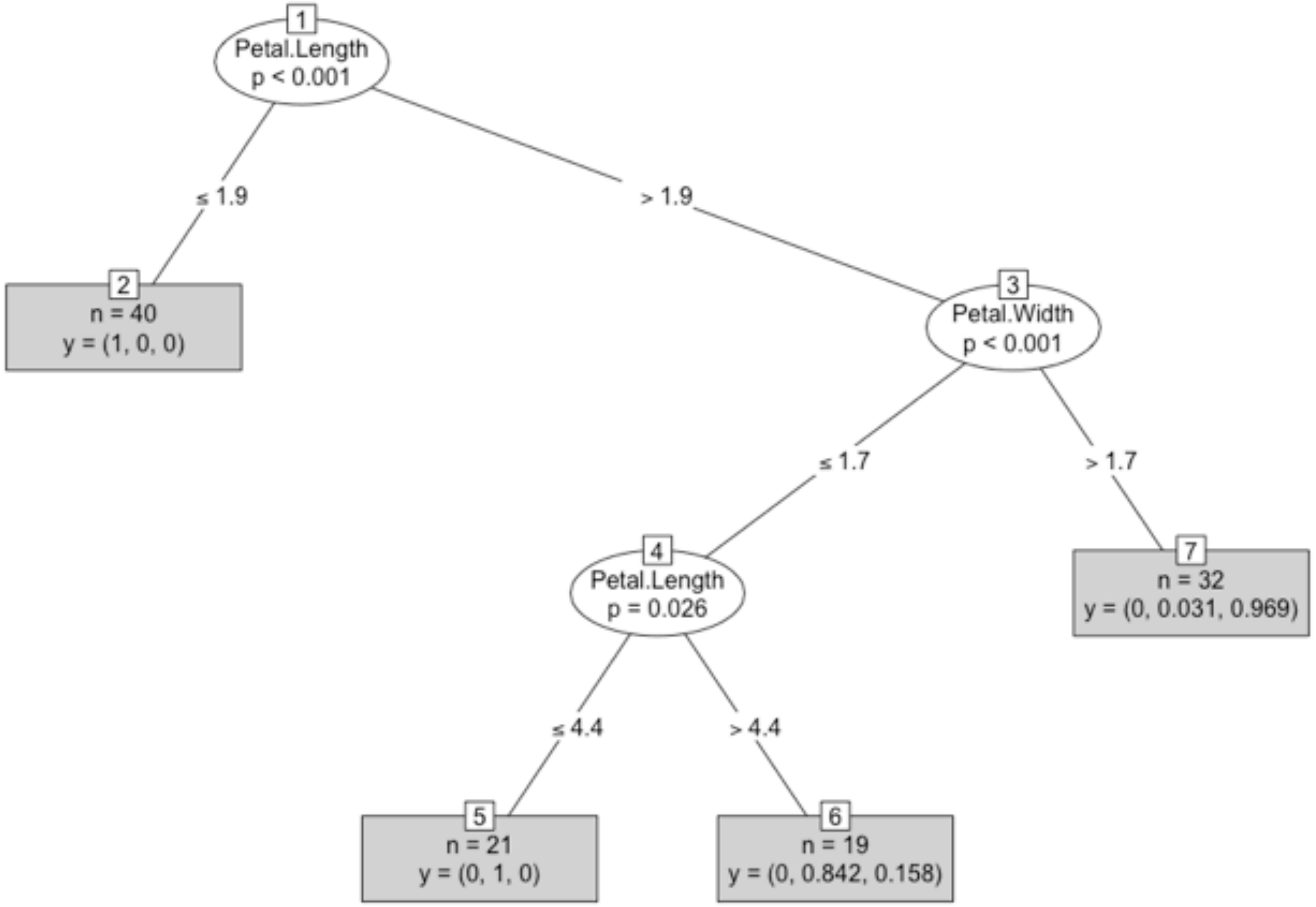


```
         Conditional inference tree with 4 terminal nodes

Response:  Species
Inputs:  Sepal.Length, Sepal.Width, Petal.Length, Petal.Width
Number of observations:  112

1) Petal.Length <= 1.9; criterion = 1, statistic = 104.643
  2)*  weights = 40
1) Petal.Length > 1.9
  3) Petal.Width <= 1.7; criterion = 1, statistic = 48.939
    4) Petal.Length <= 4.4; criterion = 0.974, statistic = 7.397
      5)*  weights = 21
    4) Petal.Length > 4.4
      6)*  weights = 19
  3) Petal.Width > 1.7
    7)*  weights = 32
```

```
> plot(iris.ctree)
```

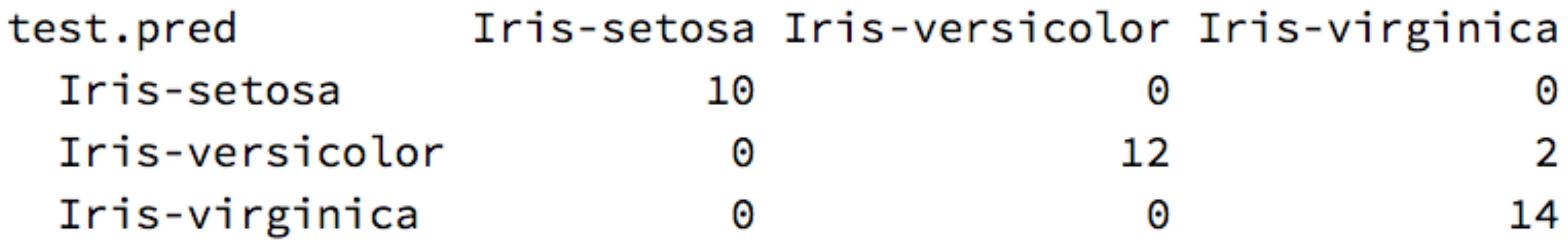


```
> plot(iris.ctree, type='simple')
```



Decision Tree - Prediction

```
> test.pred <- predict(iris.ctree, newdata=testData)  # Predict test data

> table(test.pred, testData$Species)
```



```
test.pred         Iris-setosa Iris-versicolor Iris-virginica
  Iris-setosa              10               0              0
  Iris-versicolor           0              12              2
  Iris-virginica            0               0             14
```

Check how accurate our model

- How to check the accuracy of our model?
- We can use accuracy

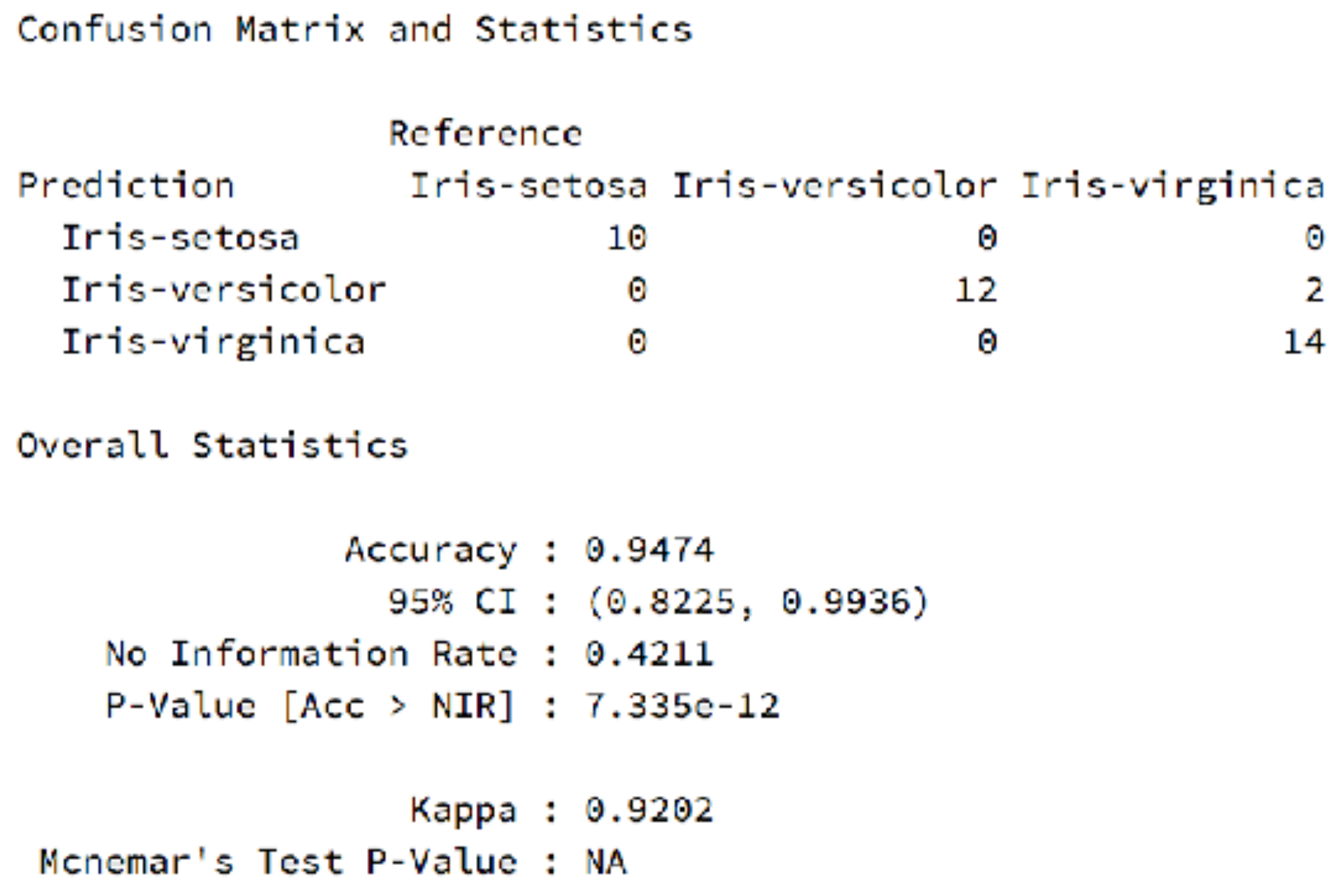
$$Accuracy = \frac{\text{Number of correct classifications}}{\text{Total number of test cases}}$$

Check performance using Caret Package

> install.packages("caret")

> library(caret)

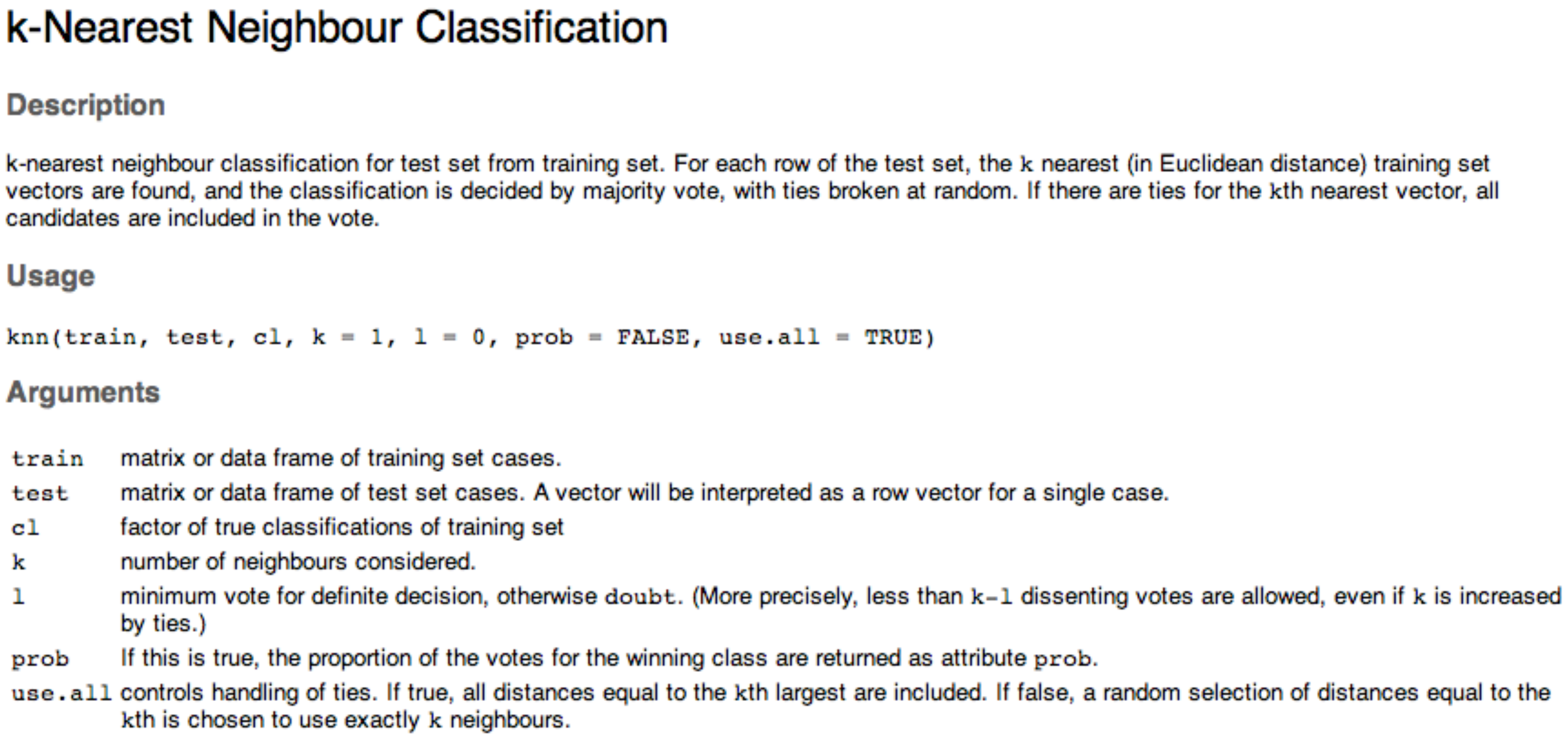
> cf <- confusionMatrix(test.pred, testData$Species)



```
Confusion Matrix and Statistics

                 Reference
Prediction        Iris-setosa Iris-versicolor Iris-virginica
  Iris-setosa              10               0              0
  Iris-versicolor           0              12              2
  Iris-virginica            0               0             14

Overall Statistics

               Accuracy : 0.9474
                 95% CI : (0.8225, 0.9936)
    No Information Rate : 0.4211
    P-Value [Acc > NIR] : 7.335e-12

                  Kappa : 0.9202
 Mcnemar's Test P-Value : NA
```

```
Statistics by Class:

                     Class: Iris-setosa Class: Iris-versicolor Class: Iris-virginica
Sensitivity                      1.0000                 1.0000                0.8750
Specificity                      1.0000                 0.9231                1.0000
Pos Pred Value                   1.0000                 0.8571                1.0000
Neg Pred Value                   1.0000                 1.0000                0.9167
Prevalence                       0.2632                 0.3158                0.4211
Detection Rate                   0.2632                 0.3158                0.3684
Detection Prevalence             0.2632                 0.3684                0.3684
Balanced Accuracy                1.0000                 0.9615                0.9375
```

K-NN



k-Nearest Neighbour Classification

Description

k-nearest neighbour classification for test set from training set. For each row of the test set, the `k` nearest (in Euclidean distance) training set vectors are found, and the classification is decided by majority vote, with ties broken at random. If there are ties for the `k`th nearest vector, all candidates are included in the vote.

Usage

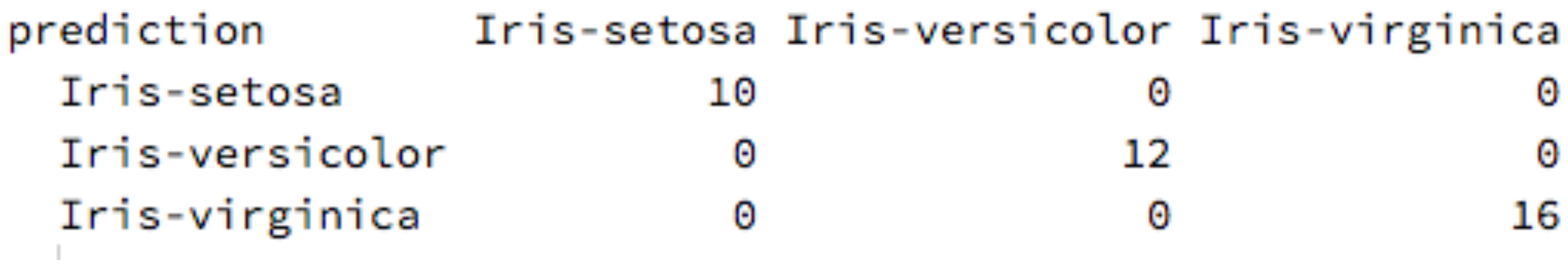
```
knn(train, test, cl, k = 1, l = 0, prob = FALSE, use.all = TRUE)
```

Arguments

`train`	matrix or data frame of training set cases.
`test`	matrix or data frame of test set cases. A vector will be interpreted as a row vector for a single case.
`cl`	factor of true classifications of training set
`k`	number of neighbours considered.
`l`	minimum vote for definite decision, otherwise `doubt`. (More precisely, less than `k-l` dissenting votes are allowed, even if `k` is increased by ties.)
`prob`	If this is true, the proportion of the votes for the winning class are returned as attribute `prob`.
`use.all`	controls handling of ties. If true, all distances equal to the `k`th largest are included. If false, a random selection of distances equal to the `k`th is chosen to use exactly `k` neighbours.

K-NN

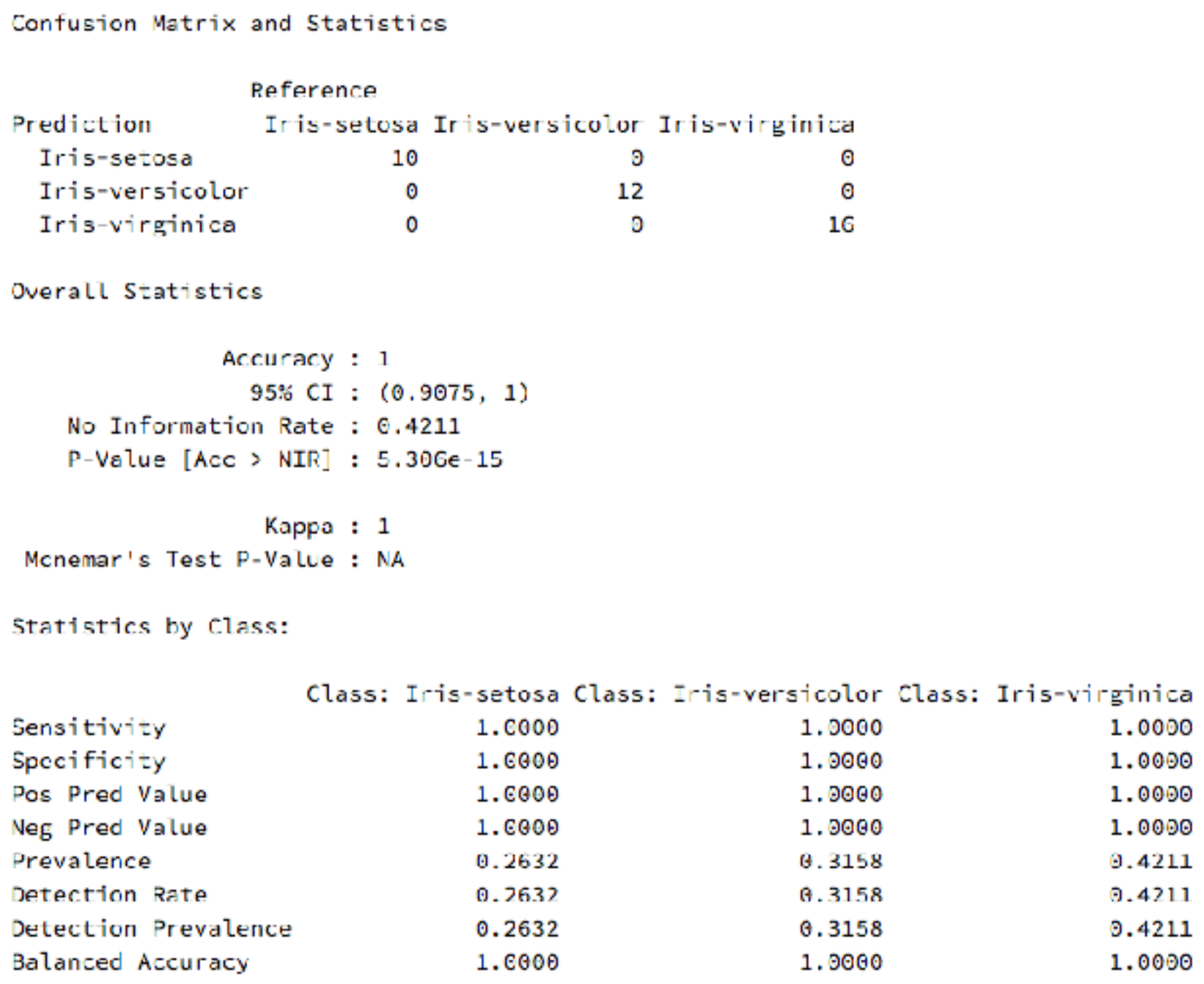
```
> library(class)
> train_input <- as.matrix(trainData[, -5])
> train_output <- as.vector(trainData[, 5])
> test_input <- as.matrix(testData[, -5])
> prediction <- knn(train_input, test_input, train_output, k=5)
> table(prediction, testData$Species)
```



```
prediction        Iris-setosa Iris-versicolor Iris-virginica
  Iris-setosa              10               0              0
  Iris-versicolor           0              12              0
  Iris-virginica            0               0             16
```

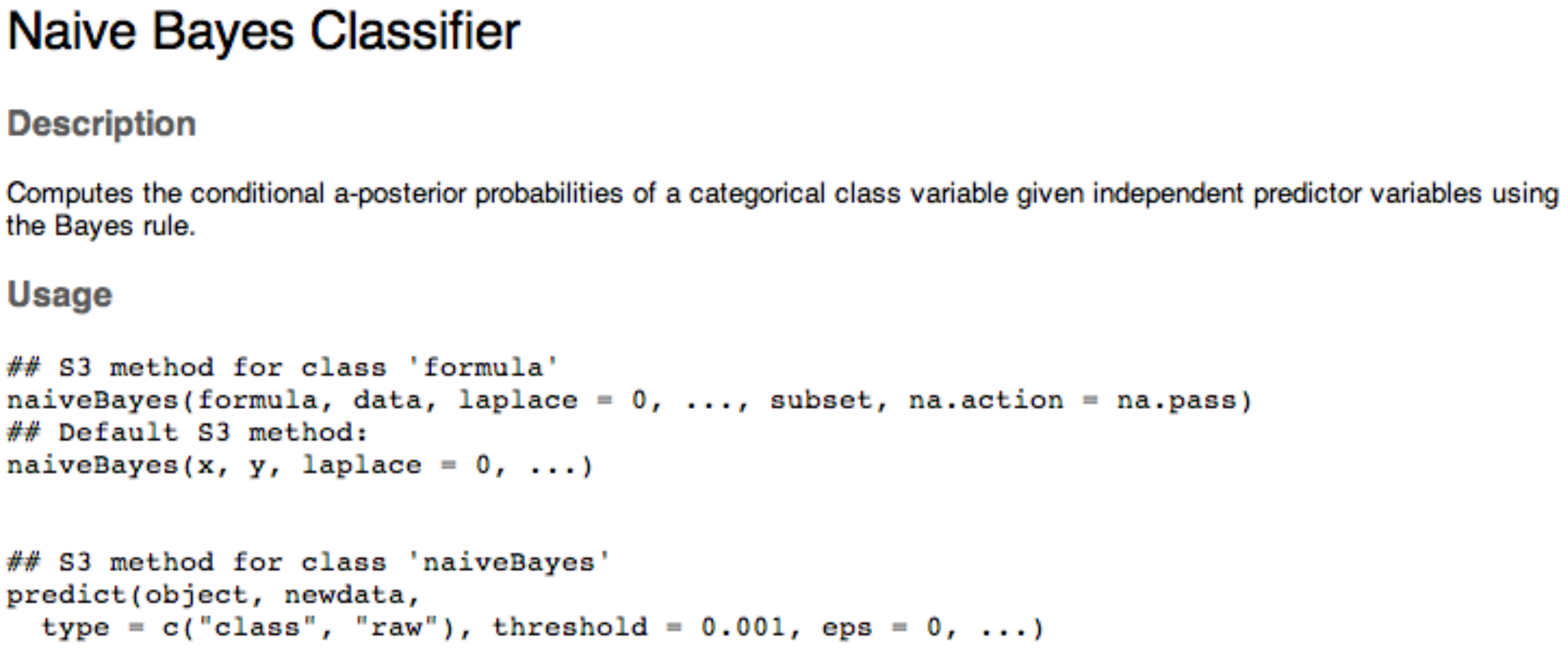
K-NN Performance

> confusionMatrix(prediction, testData$Species)



```
Confusion Matrix and Statistics

                 Reference
Prediction        Iris-setosa Iris-versicolor Iris-virginica
  Iris-setosa              10               0              0
  Iris-versicolor           0              12              0
  Iris-virginica            0               0             16

Overall Statistics

               Accuracy : 1
                 95% CI : (0.9075, 1)
    No Information Rate : 0.4211
    P-Value [Acc > NIR] : 5.306e-15

                  Kappa : 1
 Mcnemar's Test P-Value : NA

Statistics by Class:

                     Class: Iris-setosa Class: Iris-versicolor Class: Iris-virginica
Sensitivity                      1.0000                 1.0000                1.0000
Specificity                      1.0000                 1.0000                1.0000
Pos Pred Value                   1.0000                 1.0000                1.0000
Neg Pred Value                   1.0000                 1.0000                1.0000
Prevalence                       0.2632                 0.3158                0.4211
Detection Rate                   0.2632                 0.3158                0.4211
Detection Prevalence             0.2632                 0.3158                0.4211
Balanced Accuracy                1.0000                 1.0000                1.0000
```

Naive Bayes



Naive Bayes Classifier

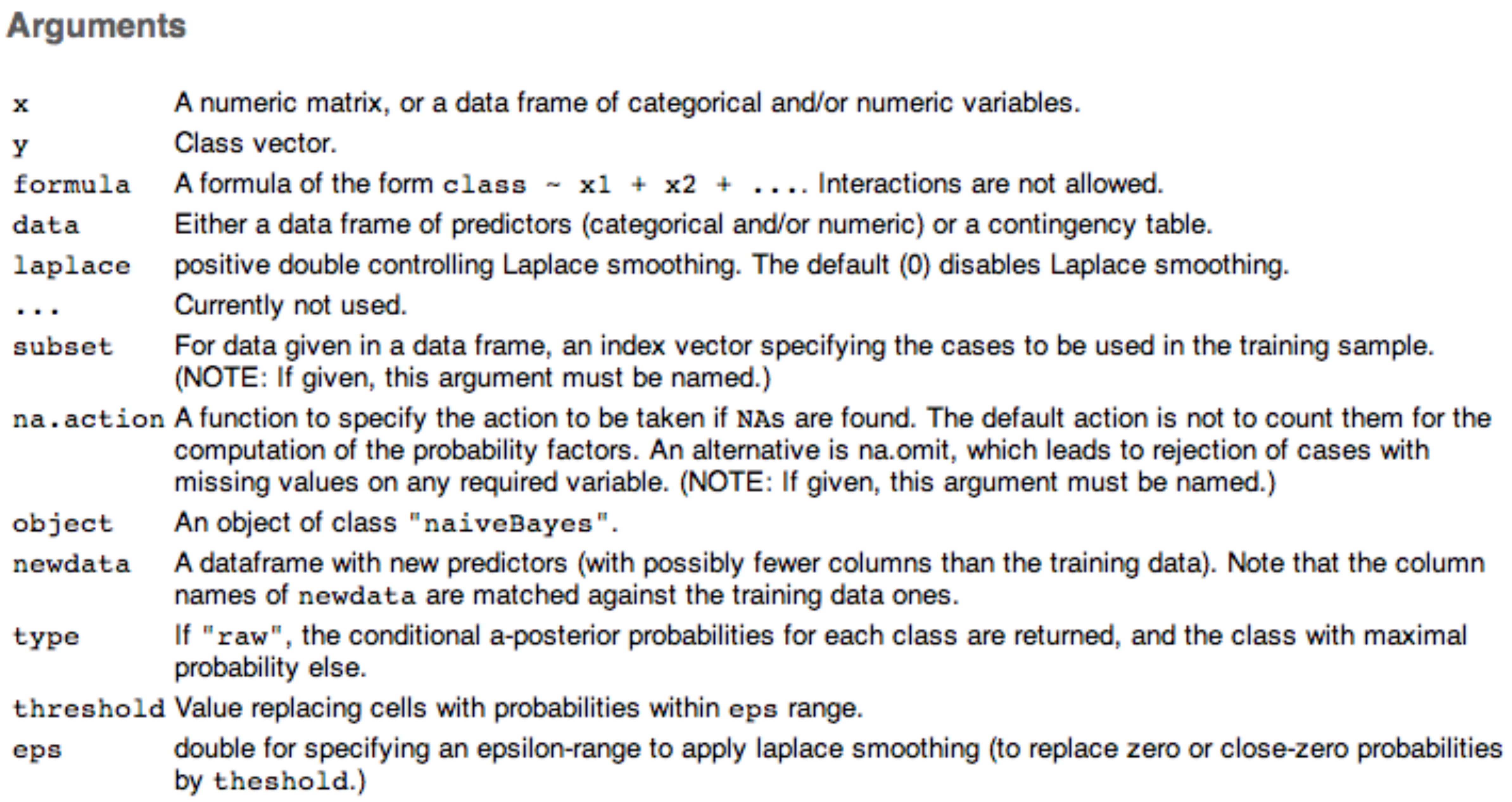
Description

Computes the conditional a-posterior probabilities of a categorical class variable given independent predictor variables using the Bayes rule.

Usage

```
## S3 method for class 'formula'
naiveBayes(formula, data, laplace = 0, ..., subset, na.action = na.pass)
## Default S3 method:
naiveBayes(x, y, laplace = 0, ...)


## S3 method for class 'naiveBayes'
predict(object, newdata,
  type = c("class", "raw"), threshold = 0.001, eps = 0, ...)
```

Naive Bayes

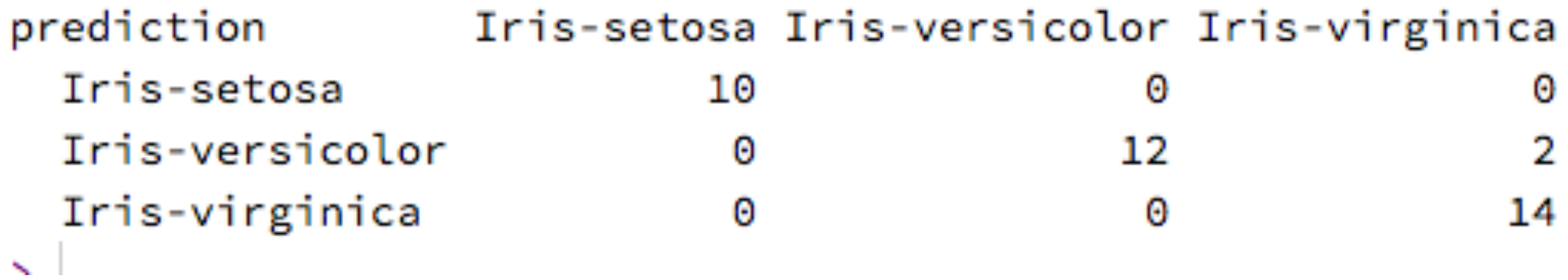


Arguments

`x`	A numeric matrix, or a data frame of categorical and/or numeric variables.
`y`	Class vector.
`formula`	A formula of the form `class ~ x1 + x2 + ...`. Interactions are not allowed.
`data`	Either a data frame of predictors (categorical and/or numeric) or a contingency table.
`laplace`	positive double controlling Laplace smoothing. The default (0) disables Laplace smoothing.
`...`	Currently not used.
`subset`	For data given in a data frame, an index vector specifying the cases to be used in the training sample. (NOTE: If given, this argument must be named.)
`na.action`	A function to specify the action to be taken if `NA`s are found. The default action is not to count them for the computation of the probability factors. An alternative is na.omit, which leads to rejection of cases with missing values on any required variable. (NOTE: If given, this argument must be named.)
`object`	An object of class `"naiveBayes"`.
`newdata`	A dataframe with new predictors (with possibly fewer columns than the training data). Note that the column names of `newdata` are matched against the training data ones.
`type`	If `"raw"`, the conditional a-posterior probabilities for each class are returned, and the class with maximal probability else.
`threshold`	Value replacing cells with probabilities within `eps` range.
`eps`	double for specifying an epsilon-range to apply laplace smoothing (to replace zero or close-zero probabilities by `theshold`.)

Naive Bayes

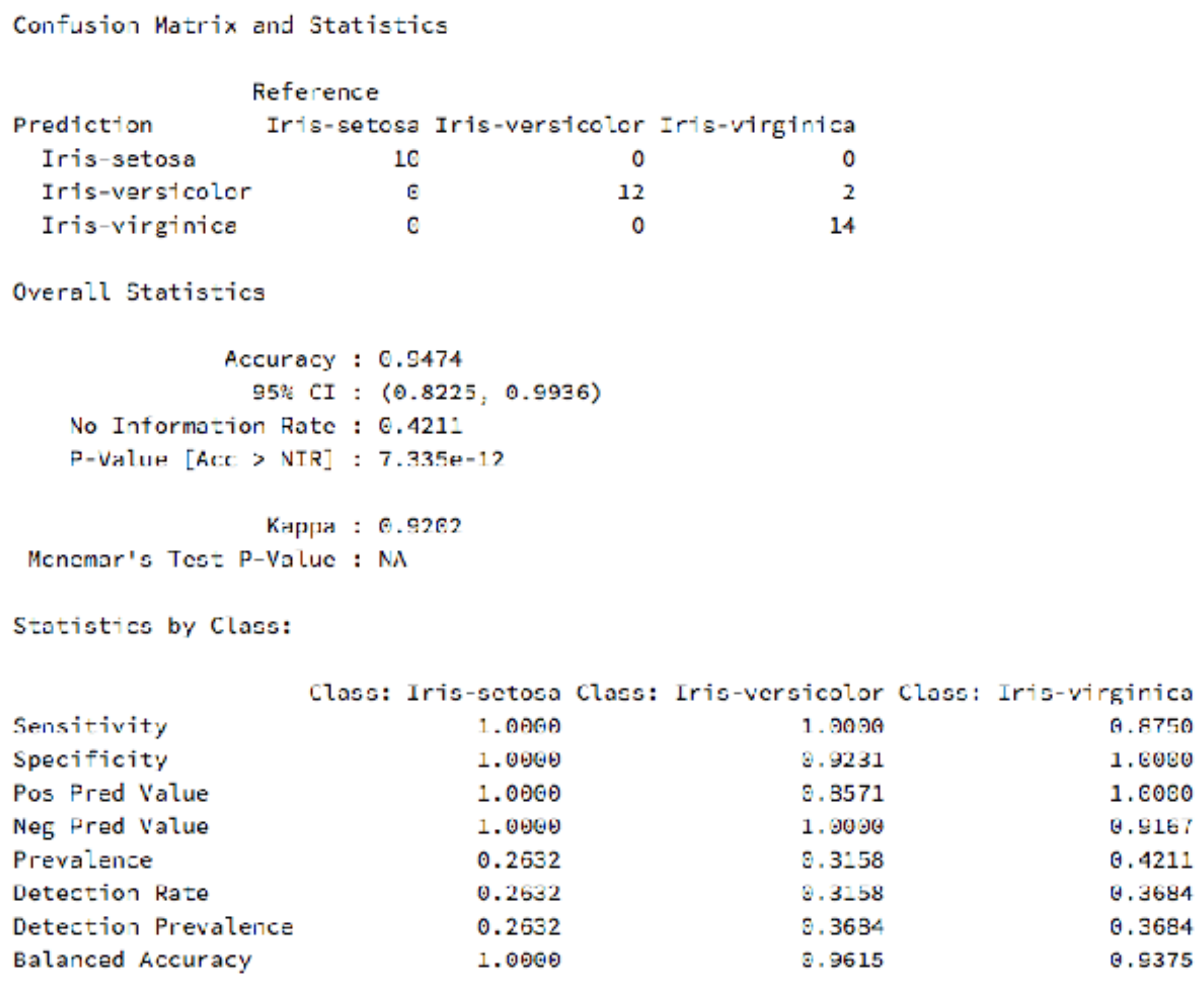
```
> library(e1071)
> nb.model <- naiveBayes(Species~. , data= trainData)
> prediction <- predict(nb.model, testData[,-5])
> table(prediction, testData[,5])
```



```
prediction        Iris-setosa Iris-versicolor Iris-virginica
  Iris-setosa              10               0              0
  Iris-versicolor           0              12              2
  Iris-virginica            0               0             14
>
```

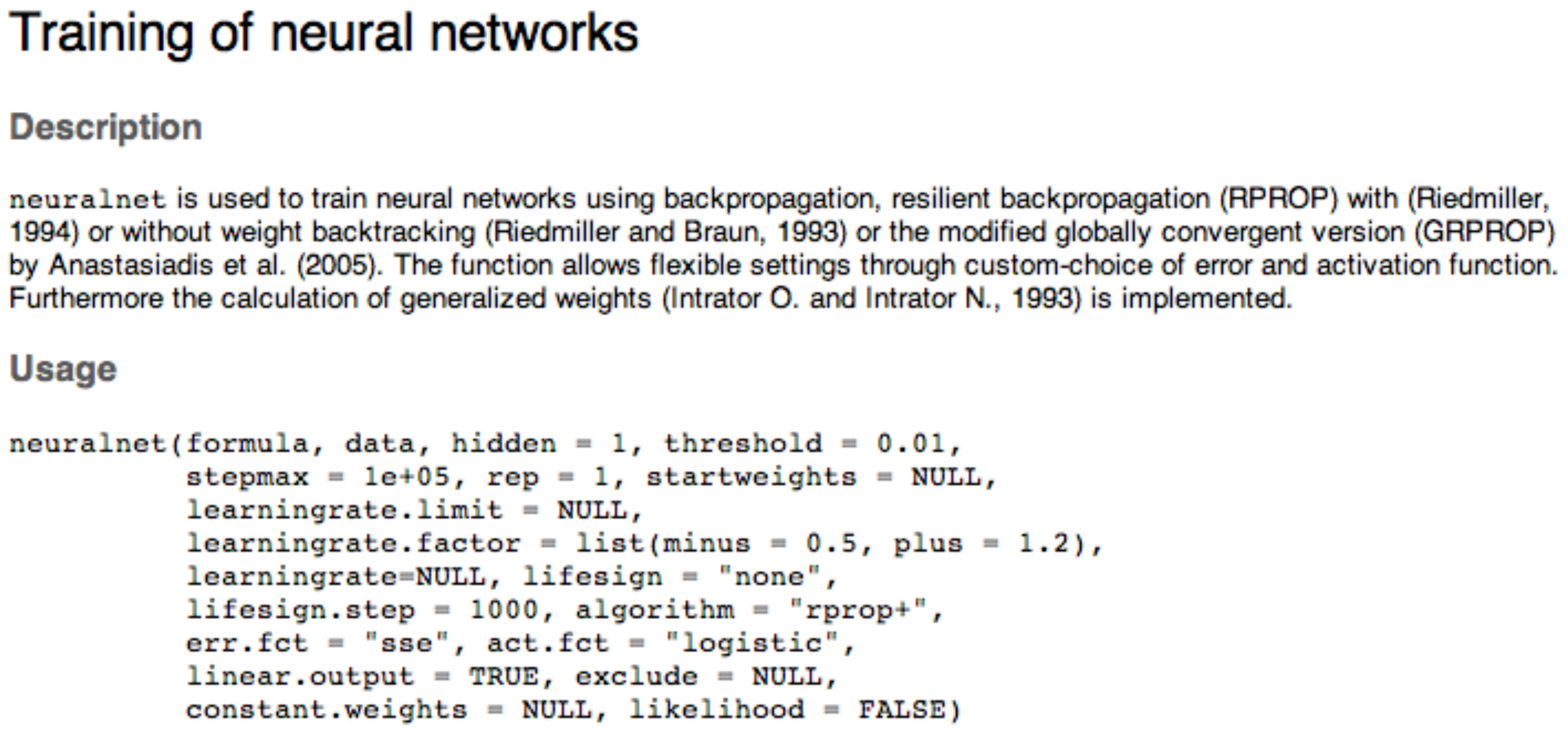
Performance of Naive Bayes

> confusionMatrix(prediction, testData$Species)



```
Confusion Matrix and Statistics

                 Reference
Prediction        Iris-setosa Iris-versicolor Iris-virginica
  Iris-setosa              10               0              0
  Iris-versicolor           0              12              2
  Iris-virginica            0               0             14

Overall Statistics

               Accuracy : 0.9474
                 95% CI : (0.8225, 0.9936)
    No Information Rate : 0.4211
    P-Value [Acc > NIR] : 7.335e-12

                  Kappa : 0.9202
 Mcnemar's Test P-Value : NA

Statistics by Class:

                     Class: Iris-setosa Class: Iris-versicolor Class: Iris-virginica
Sensitivity                      1.0000                 1.0000                0.8750
Specificity                      1.0000                 0.9231                1.0000
Pos Pred Value                   1.0000                 0.8571                1.0000
Neg Pred Value                   1.0000                 1.0000                0.9167
Prevalence                       0.2632                 0.3158                0.4211
Detection Rate                   0.2632                 0.3158                0.3684
Detection Prevalence             0.2632                 0.3684                0.3684
Balanced Accuracy                1.0000                 0.9615                0.9375
```

Neural Network



Training of neural networks

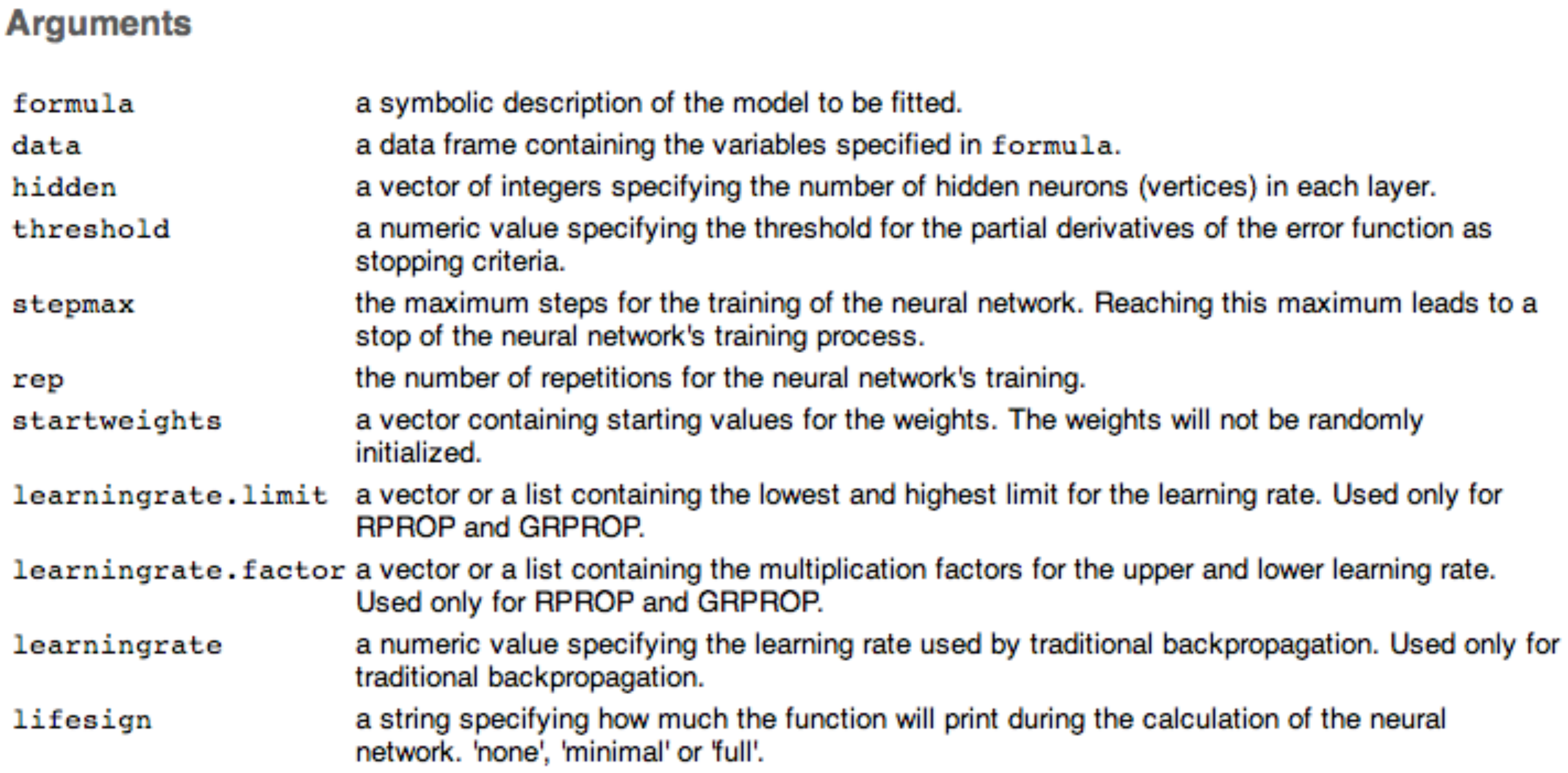
Description

`neuralnet` is used to train neural networks using backpropagation, resilient backpropagation (RPROP) with (Riedmiller, 1994) or without weight backtracking (Riedmiller and Braun, 1993) or the modified globally convergent version (GRPROP) by Anastasiadis et al. (2005). The function allows flexible settings through custom-choice of error and activation function. Furthermore the calculation of generalized weights (Intrator O. and Intrator N., 1993) is implemented.

Usage

```
neuralnet(formula, data, hidden = 1, threshold = 0.01,
          stepmax = 1e+05, rep = 1, startweights = NULL,
          learningrate.limit = NULL,
          learningrate.factor = list(minus = 0.5, plus = 1.2),
          learningrate=NULL, lifesign = "none",
          lifesign.step = 1000, algorithm = "rprop+",
          err.fct = "sse", act.fct = "logistic",
          linear.output = TRUE, exclude = NULL,
          constant.weights = NULL, likelihood = FALSE)
```

Neural Network



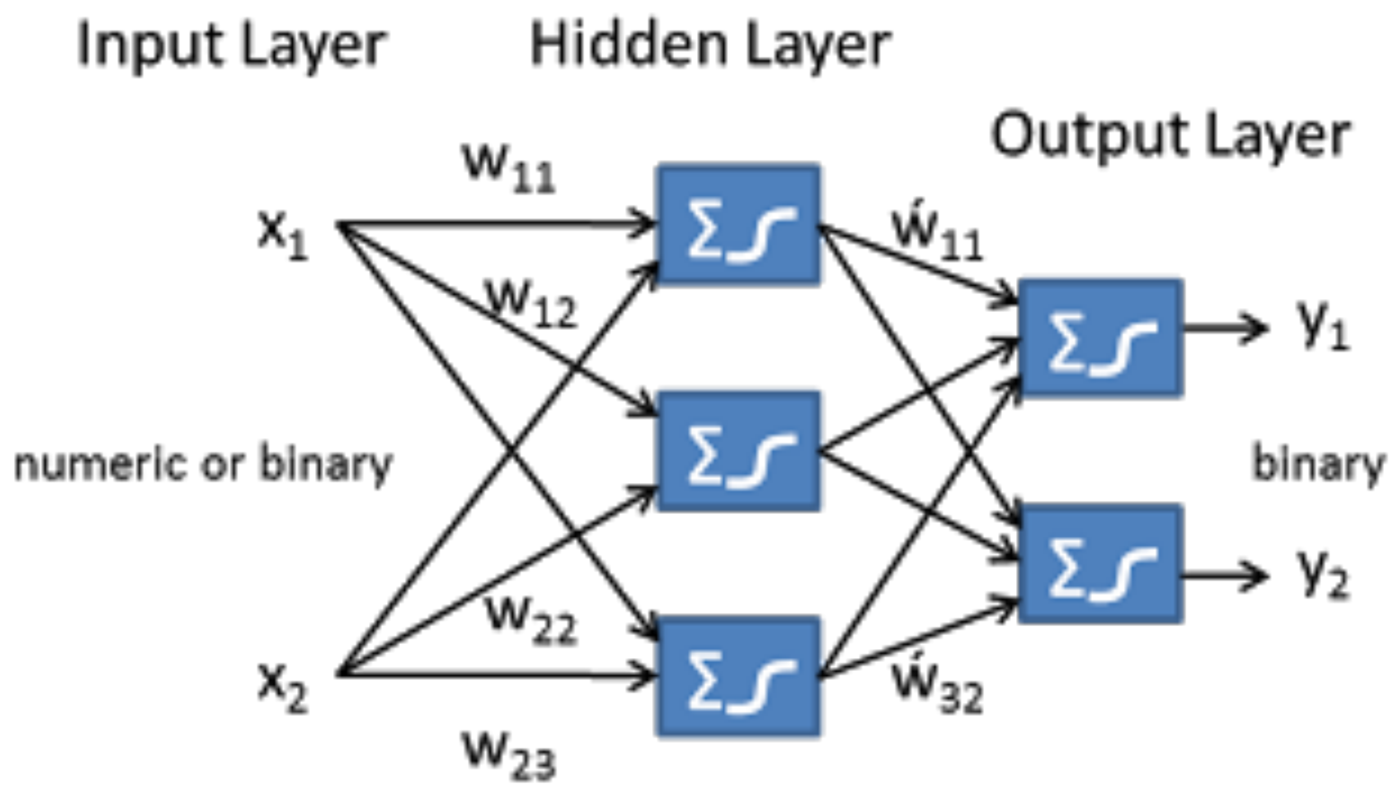
Arguments

`formula`	a symbolic description of the model to be fitted.
`data`	a data frame containing the variables specified in `formula`.
`hidden`	a vector of integers specifying the number of hidden neurons (vertices) in each layer.
`threshold`	a numeric value specifying the threshold for the partial derivatives of the error function as stopping criteria.
`stepmax`	the maximum steps for the training of the neural network. Reaching this maximum leads to a stop of the neural network's training process.
`rep`	the number of repetitions for the neural network's training.
`startweights`	a vector containing starting values for the weights. The weights will not be randomly initialized.
`learningrate.limit`	a vector or a list containing the lowest and highest limit for the learning rate. Used only for RPROP and GRPROP.
`learningrate.factor`	a vector or a list containing the multiplication factors for the upper and lower learning rate. Used only for RPROP and GRPROP.
`learningrate`	a numeric value specifying the learning rate used by traditional backpropagation. Used only for traditional backpropagation.
`lifesign`	a string specifying how much the function will print during the calculation of the neural network. 'none', 'minimal' or 'full'.

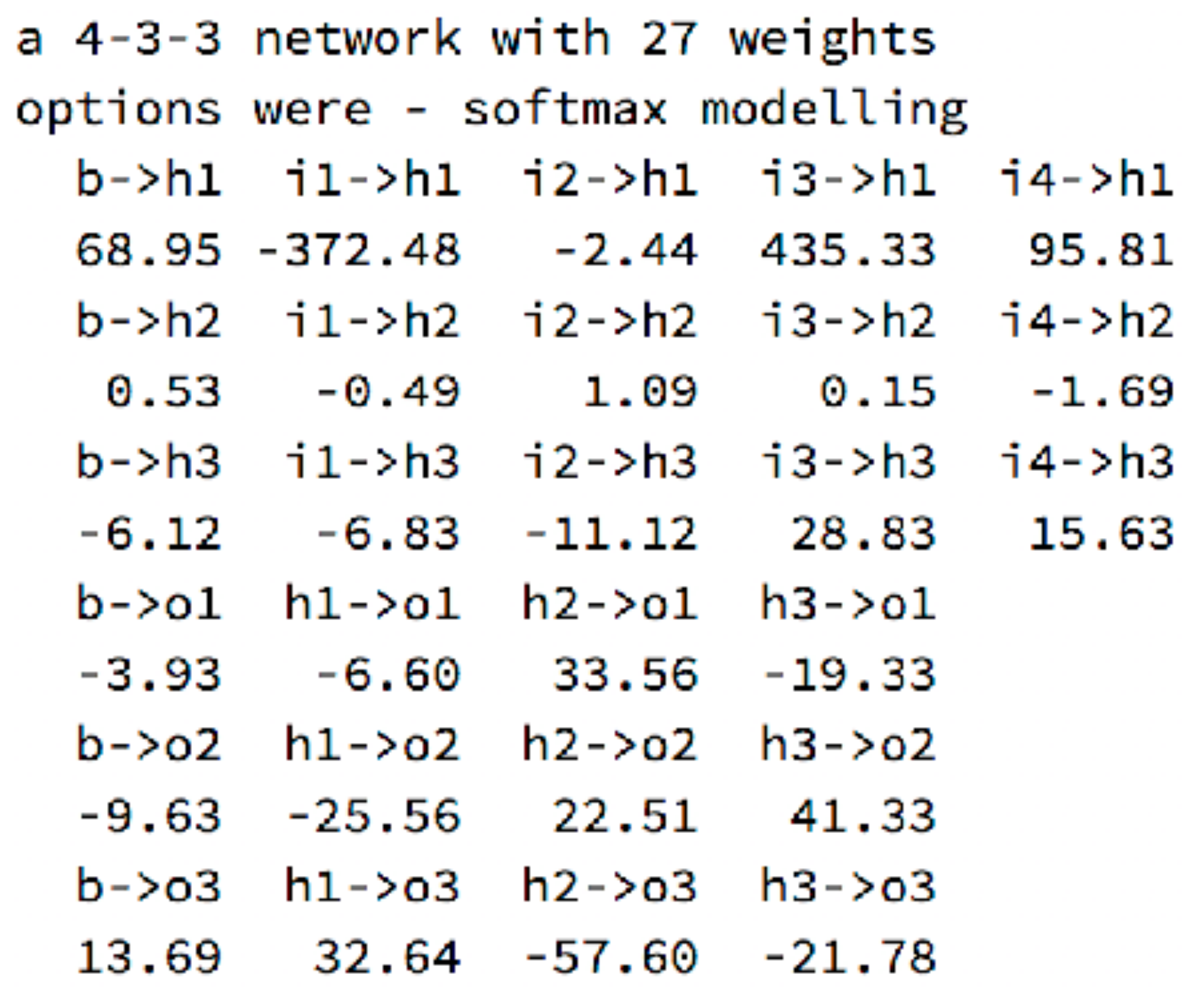
Neural Network

`lifesign.step`	an integer specifying the stepsize to print the minimal threshold in full lifesign mode.
`algorithm`	a string containing the algorithm type to calculate the neural network. The following types are possible: 'backprop', 'rprop+', 'rprop-', 'sag', or 'slr'. 'backprop' refers to backpropagation, 'rprop+' and 'rprop-' refer to the resilient backpropagation with and without weight backtracking, while 'sag' and 'slr' induce the usage of the modified globally convergent algorithm (grprop). See Details for more information.
`err.fct`	a differentiable function that is used for the calculation of the error. Alternatively, the strings 'sse' and 'ce' which stand for the sum of squared errors and the cross-entropy can be used.
`act.fct`	a differentiable function that is used for smoothing the result of the cross product of the covariate or neurons and the weights. Additionally the strings, 'logistic' and 'tanh' are possible for the logistic function and tangent hyperbolicus.
`linear.output`	logical. If act.fct should not be applied to the output neurons set linear output to TRUE, otherwise to FALSE.
`exclude`	a vector or a matrix specifying the weights, that are excluded from the calculation. If given as a vector, the exact positions of the weights must be known. A matrix with n-rows and 3 columns will exclude n weights, where the first column stands for the layer, the second column for the input neuron and the third column for the output neuron of the weight.
`constant.weights`	a vector specifying the values of the weights that are excluded from the training process and treated as fix.
`likelihood`	logical. If the error function is equal to the negative log-likelihood function, the information criteria AIC and BIC will be calculated. Furthermore the usage of confidence.interval is meaningfull.

Neural Network

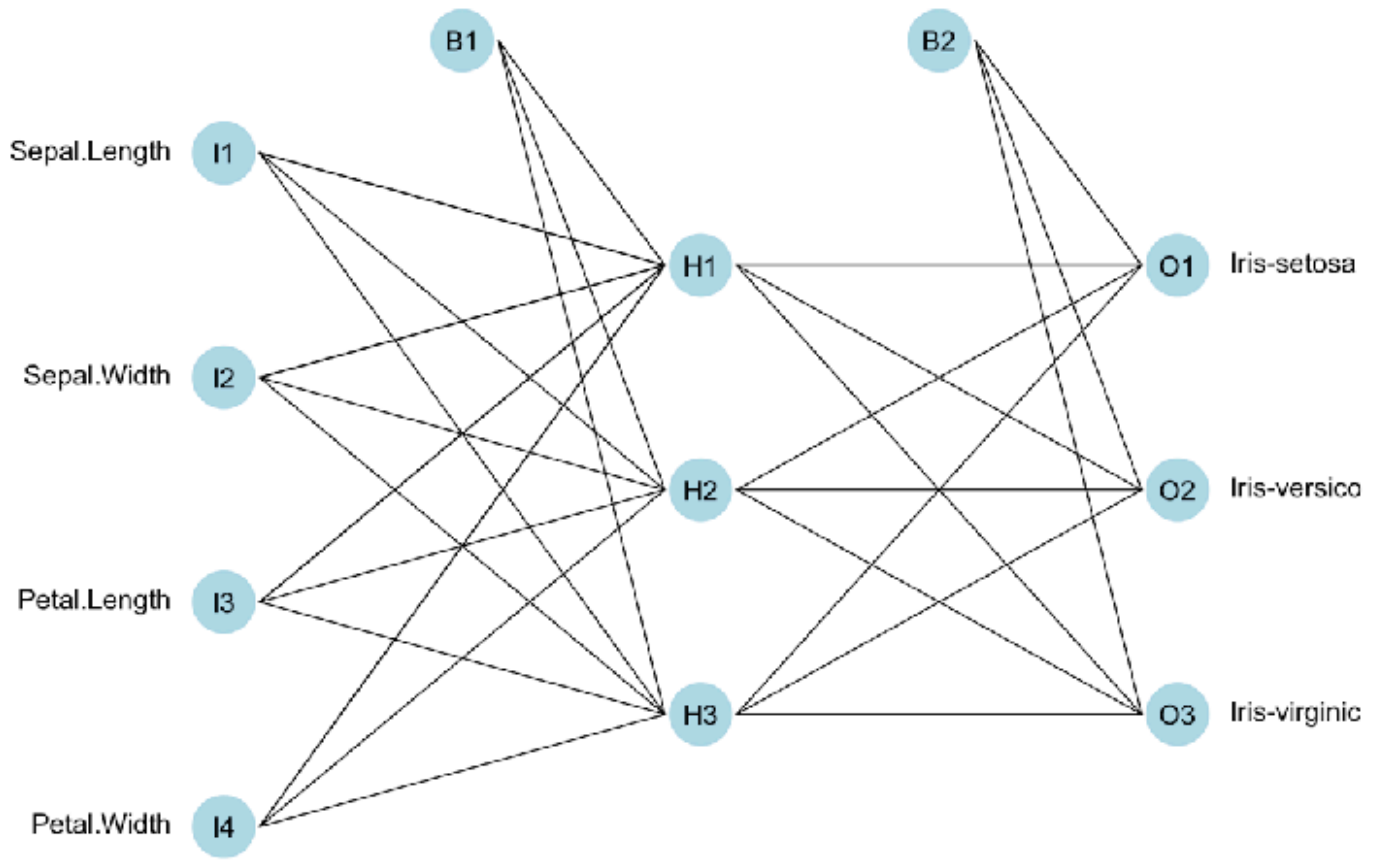


```
> library(nnet)
> library(NeuralNetTools)

> # Create Neural Network Model
> nn <- nnet(Species~., data=trainData, size=3, maxit=500)

# summarize the fit
> summary(nn)
```



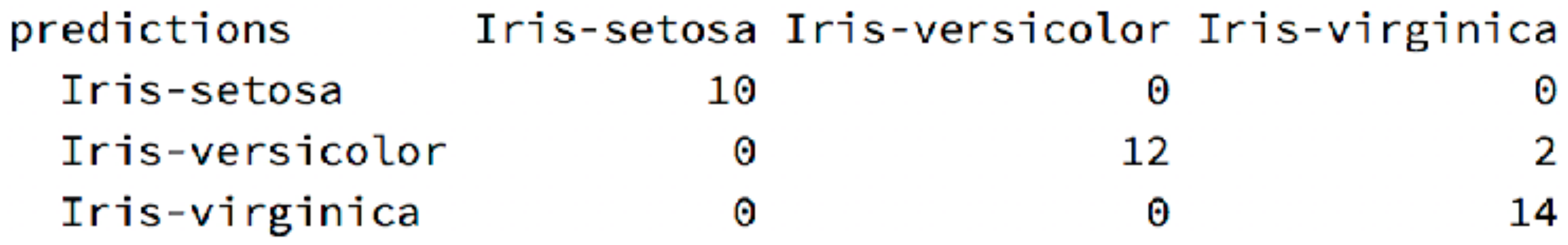
```
a 4-3-3 network with 27 weights
options were - softmax modelling
  b->h1  i1->h1  i2->h1  i3->h1  i4->h1
  68.95 -372.48   -2.44  435.33   95.81
  b->h2  i1->h2  i2->h2  i3->h2  i4->h2
   0.53   -0.49    1.09    0.15   -1.69
  b->h3  i1->h3  i2->h3  i3->h3  i4->h3
  -6.12   -6.83  -11.12   28.83   15.63
  b->o1  h1->o1  h2->o1  h3->o1
  -3.93   -6.60   33.56  -19.33
  b->o2  h1->o2  h2->o2  h3->o2
  -9.63  -25.56   22.51   41.33
  b->o3  h1->o3  h2->o3  h3->o3
  13.69   32.64  -57.60  -21.78
```

```
> library(devtools)
> source_url('https://goo.gl/qB3rHg')
> plot.nnet(nn, pid=F)
```



Neural Network Prediction

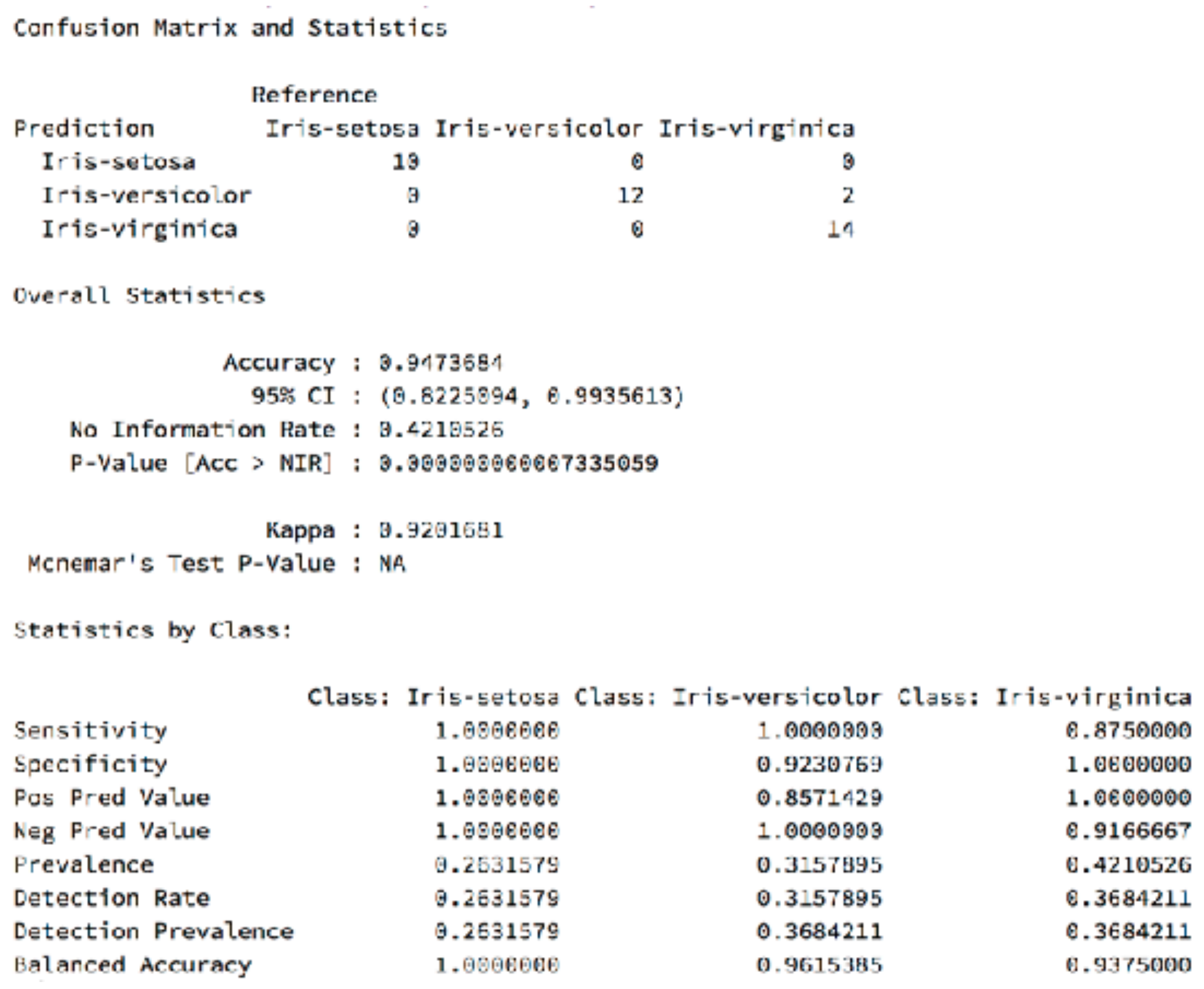
```
> # make predictions
> predictions <- predict(nn, testData[,1:4], type="class")
> # summarize accuracy
> table(predictions, testData$Species)
```



```
predictions       Iris-setosa Iris-versicolor Iris-virginica
  Iris-setosa              10               0              0
  Iris-versicolor           0              12              2
  Iris-virginica            0               0             14
```

Performance of Neural Network

> confusionMatrix(predictions, testData$Species)



```
Confusion Matrix and Statistics

                 Reference
Prediction        Iris-setosa Iris-versicolor Iris-virginica
  Iris-setosa              10               0              0
  Iris-versicolor           0              12              2
  Iris-virginica            0               0             14

Overall Statistics

               Accuracy : 0.9473684
                 95% CI : (0.8225094, 0.9935613)
    No Information Rate : 0.4210526
    P-Value [Acc > NIR] : 0.0000000000067335059

                  Kappa : 0.9201681
 Mcnemar's Test P-Value : NA

Statistics by Class:

                     Class: Iris-setosa Class: Iris-versicolor Class: Iris-virginica
Sensitivity                   1.0000000              1.0000000             0.8750000
Specificity                   1.0000000              0.9230769             1.0000000
Pos Pred Value                1.0000000              0.8571429             1.0000000
Neg Pred Value                1.0000000              1.0000000             0.9166667
Prevalence                    0.2631579              0.3157895             0.4210526
Detection Rate                0.2631579              0.3157895             0.3684211
Detection Prevalence          0.2631579              0.3684211             0.3684211
Balanced Accuracy             1.0000000              0.9615385             0.9375000
```

End of Part 1