R programming: syntax

*For strengths and weaknesses of each algo & their syntax consult The Book.*

1. Software management

|  |  |
| --- | --- |
| **Command example** | **Operation explanation** |
| install.packages(“RWeka”) | Installs the RWeka package and all its dependencies |
| ?install.packages | Help file for package function |
| Library(RODBC) | Loads a package into R (RODBC used for importing data from SQL databases - ODBC is a standard protocol for connecting to databases regardless of OS or Database Management System, aka DBMS) |
| save(x, y, z, file = “mydata.RData”) | Saves objects x, y, z regardless of whether they are vectors, factors, lists or data frames into a file of given name. |
| load(“mydata.RData”) | Recreates the x, y, z data structures |
| save.image() | Saves current session to a file called .RData (R will look for this file automatically next time you start R) |
| pt\_data <- read.csv("/path/to/data.csv", stringsAsFactors = FALSE) | Reads Comma Seperated Value (CSV) files into an R object. |
| mydata <- read.csv("mydata.csv", stringsAsFactors = FALSE,  header = FALSE) | By default R assumes that CSV files include headers as the first row of the file thus header = FALSE must be used for headless CSV files. |
| write.csv(pt\_data, file = "pt\_data.csv") | Used to create a CSV file from an R object |
|  |  |
| mydb <- odbcConnect("my\_dsn") | DSN = data source name (required for using RODBC to import data from Open Database Connectivity Structured Query Language, ODB SQL, databases) |
| mydb <- odbcConnect("my\_dsn", uid = "my\_username"  pwd = "my\_password") | If a password is needed |
| sqlQuery() | Function used to query SQL databases |
| > patient\_query <- "select \* from patient\_data where alive = 1"  > patient\_data <- sqlQuery(channel = mydb, query = patient\_query,  stringsAsFactors = FALSE) | Typical method for using SQL in R  Resulting patient\_data variable will be a data frame containing all rows selected using the SQL query stored in patient\_query |
| odbcClose(mydb) | Closes the mydb connection (automatically done when R session is ended) |

1. Data structures

|  |  |
| --- | --- |
| object\_name | Prints the information stored in an R object |
| remove(object\_name) | Removes an R object |
| c() | Combine function which creates a vector |
| CVector\_name <- c(“John”) | Writes a character vector |
| NVector\_name <- c(9.81) | Writes a numeric vector |
| IVector\_name <- c(12, 13) | Writes an integer vector (two entries) |
| LVector\_name <- c(TRUE, FALSE) | Writes a logical vector (two entries) |
| NULL | Special vector type used in machine learning used to indicate absence of a value |
| NA | Special vector type used in machine learning used to indicate missing value (used for uninitialized values in vectors) |
| Vector\_name[2] | Prints 2nd value of a vector in the form:  [1] 13 |
| Vector\_name[1:4] | Prints elements of the vector from 1st to 4th in the form: [1] 12, 13, NA, NA |
| Vector\_name[-2] | Prints all elements of the vector except 2nd element, in usual format. |
| Vector\_name[c(TRUE, FALSE)] | Prints vector according to the logical vector specified |
| factor() | Factor function used for storing nominal values (small, medium, large) while taking up less memory than c() |
| gender <- factor(c(“MALE”, “FEMALE”, “MALE”)) | Creating a factor of 3 genders &storing this in the ‘gender’ var in the form:  [1] MALE FEMALE MALE  Levels: FEMALE MALE |
| levels | Keyword for explicitly adding levels to factors even if they do not appear in data |
| blood <- factor(c("O", "AB", "A"), levels = c("A", "B", "AB", "O")) | Creating a factor with 3 blood types and adding a level that did not appear in the data before writing to the var ‘blood’ in the form:  [1] O AB A  Levels: A B AB O |
| list() | List function which creates a list a fast way of assigning/displaying data of an object |
| subject1 <- list(fullname = subject\_name[1],  temperature = temperature[1],  flu\_status = flu\_status[1],  gender = gender[1],  blood = blood[1]) | Stores the following information for the ‘subject1’ object:  $fullname  [1] "John Doe"  $temperature  [1] 98.1  $flu\_status  [1] FALSE  $gender  [1] MALE  Levels: FEMALE MALE  $blood  [1] O  Levels: A B AB O |
| subject1[2] | Since values are labelled with the names specified in the list command this prints the values of the second feature of the subject1 object i.e.:  $temperature  [1] 98.1 |
| subject1$temperature | An easier way of accessing subject1’s temperature feature. Also ensures that if you add or remove values from the list that you do not accidently retrieve the wrong list item |
| subject1[c(“temperature”, “flu\_status”)] | Accessing several items in a list by specifying a vector of names (note lists can be used to build datasets but this is better done with a specialised data structure: the data frame = a list of vectors) |
| Data.frame() | Using data vectors previously created, this function combines them into a data frame (columns are features/attributes and rows are examples) |
| pt\_data <- data.frame(subject\_name, temperature, flu\_status, gender, blood, stringsAsFactors = FALSE) | An example of a dataframe. stringsAsFactors = FALSE required to prevent R from automatically converting every character vector to a factor, to output:  subject\_name temp flu\_status gender blood  1 John Doe 98.1 FALSE MALE O  2 Jane Doe 98.6 FALSE FEMALE AB  3 Steve Graves 101.4 TRUE MALE A |
| pt\_data$subject\_name | Extracts the subject names from the data frame above to output:  [1] "John Doe" "Jane Doe" "Steve Graves" |
| pt\_data[c("temperature", "flu\_status")] | Similarly to lists, you can extract several features using a vector of names, to output:  temperature flu\_status  1 98.1 FALSE  2 98.6 FALSE  3 101.4 TRUE |
| pt\_data[1, 2] | Extracts data from first row, second column to output:  [1] 98.1 |
| pt\_data[c(1, 3), c(2, 4)] | Extracts more than one row and clumn of data from a data frame to output:  temperature gender  1 98.1 MALE  3 101.4 MALE |
| pt\_data[, 1] | Extracts all of one column where left blank |
| pt\_data[c(1, 3), c("temperature", "gender")] | Since all methods may be used for data tables this is equivalent to:  pt\_data[-2, c(-1, -3, -5)] |
| matrix() | Function used to create a matrix |
| m <- matrix(c('a', 'b', 'c', 'd'), nrow = 2) | Matrix creation results in the following:  [,1] [,2]  [1,] "a" "c"  [2,] "b" "d" |
| m <- matrix(c('a', 'b', 'c', 'd'), ncol = 2) | Equivalent matrix creation (specify either nrow or ncol) |
| m <- matrix(c('a', 'b', 'c', 'd', 'e', 'f'), nrow = 2) | Note that column-major order is implemented this columns are filled first. Output:  [,1] [,2] [,3]  [1,] "a" "c" "e"  [2,] "b" "d" "f" |
| m[2, 3] | Extraction of data within a matrix. Can also be done with arrays (outside scope of sheet)  Will return values f and m, like for data structures. |
| View(function) | Used to view a function |

1. Univariate statistics

|  |  |
| --- | --- |
| str(data\_frame\_name) | Useful exploring of data frame files. Output:  'data.frame': 150 obs. of 6 variables:  $ year : int 2011 2011 2011 2011 2012 2010 2011 2010 2011 2010 ...  $ model : chr "SEL" "SEL" "SEL" "SEL" ...  $ price : int 21992 20995 19995 17809 17500 17495 17000 16995 16995 16995 ...  $ mileage : int 7413 10926 7351 11613 8367 25125 27393 21026 32655 36116 ...  $ color : chr "Yellow" "Gray" "Silver" "Gray" ...  $ transmission: chr "AUTO" "AUTO" "AUTO" "AUTO" ... |
| summary(usedcars$year) | Useful for investigating numeric variables (displays several common summary statistics). Output:  Min. 1st Qu. Median Mean 3rd Qu. Max.  2000 2008 2009 2009 2010 2012 |
| mean() | Function used to find the mean of data |
| median() | Function used to find the median of data |
| mode() | WRONG. Look at the table output for the category with the greatest number of values. |
| range() | Returns minimum and maximum values of data |
| IQR() | Used to find the inter-quartile range of data |
| quantile() | Identifies quantiles for a set of values |
| quantile(usedcars$price, probs = c(0.01, 0.99)) | Returns arbitrary quantiles such as 1st and 99th percentiles. Output:  1% 99%  5428.69 20505.00 |
| quantile(usedcars$price, seq(from = 0, to = 1, by = 0.20)) | Returns:  0% 20% 40% 60% 80% 100%  800.0 10759.4 12993.8 13992.0 14999.0 21992.0 |
| main, xlab, ylab | Parameters used to label title & axis of plots |
| boxplot(usedcars$price, main="Boxplot of Used Car Prices", ylab="Price ($)") |  |
| hist(usedcars$price, main = "Histogram of Used Car Prices",  xlab = "Price ($)") |  |
| var() | Outputs the variance of a dataset |
| sd() | Outputs the standard deviation of a dataset |
| table(usedcars$year) | Explores categorical variables by showing frequency of occurance of a dataset in a table. Output:  2000 2001 2002 2003 2004 2005 2006 2007 2008 $  3 1 1 1 3 2 6 11 14 $ |
| model\_table <- table(usedcars$model)  prop.table(model\_table) | Builts a proportional table. Output:  2000 2001 2002 $  0.020000000 0.006666667 0.006666667 $ |
| > color\_table <- table(usedcars$color)  > color\_pct <- prop.table(color\_table) \* 100  > round(color\_pct, digits = 1) | More clean way of showing proportional tables. Output:  Black Blue Gold Gray Green Red $  23.3 11.3 0.7 10.7 3.3 16.7 $ |

1. Multivariate statistics

|  |  |
| --- | --- |
| plot() | Plot function requires two inputs: y&x |
| plot(x = usedcars$mileage, y = usedcars$price,  main = "Scatterplot of Price vs. Mileage",  xlab = "Used Car Odometer (mi.)",  ylab = "Used Car Price ($)") | Scatterplot command |
| usedcars$conservative <- usedcars$color %in% c("Black", "Gray", "Silver", "White") | Splits the colours of used cars into conservative and non conservative categories.  %in% returns TRUE or FALSE for each value in the vector on the LHS of the operator depending on whether the value is found in the vector on the RHS |
| CrossTable(x = usedcars$model, y = usedcars$conservative) | Function from the gmodels package to look at cross-tabulation. Output: |
| CrossTable(x = usedcars$model, y = usedcars$conservative, chisq = TRUE) | This includes Chi Squarred test (probability that cell counts are due to chance alone: if low then likely that two vars are associated)  Output:    Since probability is nearly 93% it is highly likely that variations are due to chance alone and not a true association between model and colour. |

1. k-Nearest-Neighbour

|  |  |
| --- | --- |
| wbcd <- wbcd[-1] | Used to remove ID variables as a model that includes an ID will most likely suffer from overfitting. |
| wbcd$diagnosis <- factor(wbcd$diagnosis, levels = c("B", "M"), labels = c("Benign", "Malignant")) | Feature must be coded as a factor so we need to recode the diagnosis variable. |
| normalize <- function(x) {  return ((x - min(x)) / (max(x) - min(x)))  } | Creates the normalize function. Now normalize() can be used as a regular function. |
| wbcd\_n <- as.data.frame(lapply(wbcd[2:31], normalize)) | This command applies the normalize() function to columns 2  through 31 in the wbcd data frame, converts the resulting list to a data frame, and  assigns it the name wbcd\_n. |
| wbcd\_train <- wbcd\_n[1:469, ]  wbcd\_test <- wbcd\_n[470:569, ] | Splits the full dataset into two datasets (1-469 for training and the remaining 100 for testing) |
| wbcd\_train\_labels <- wbcd[1:469, 1] | Extracts the diagnosis label from the previous data (for use later) |
| install.packages("class")  library(class) | Useful package for k-NN |
| wbcd\_test\_pred <- knn(train = wbcd\_train, test = wbcd\_test, cl = wbcd\_train\_labels, k=21) | Runs a basic kNN algo to the data. Trains and tests on the relevant inputs. |
| CrossTable(x = wbcd\_test\_labels, y = wbcd\_test\_pred, prop.chisq=FALSE) | Method of checking how well the predicted values match up with known values: |
| wbcd\_z <- as.data.frame(scale(wbcd[-1])) | Z-normalises the data using the built in scale function |

1. Probabilistic Learning - Classification using Naïve Bayes

|  |  |
| --- | --- |
| install.packages(“tm”) | A useful text mining package for NB |
| install.packages(“wordcloud”) | A useful text frequency visualising tool |
| install.packages(“e1071”) | A nice package from Vienna University with ML (Naïve Baines & others) |
| Install.packages(“gmodels”) | Useful for evaluating model perf (crosstable function) |
| sms\_corpus <- Corpus(VectorSource(sms\_raw$text)) | Two functions used here. First Corpus creates an R object to store text documents with a parameter specifying the format of text documents to be loaded. Here we use VectorSource() which tells corpus to use the messages in the vector sms\_train$text. |
| print(sms\_corpus) | Used to show information about a corpus |
| inspect(sms\_corpus[1:3]) | Used to view the first, second and third SMS messages |
| corpus\_clean <- tm\_map(sms\_corpus, tolower)  corpus\_clean <- tm\_map(corpus\_clean, removeNumbers) | tm\_map() is used to transform or map a corpus - here converts all to lowercase & removes numbers |
| corpus\_clean <- tm\_map(corpus\_clean, removeWords, stopwords()) | Removes useless stop words and replaces them by a space |
| corpus\_clean <- tm\_map(corpus\_clean, removePunctuation) | Removes all punctuation and replaces them by a space |
| corpus\_clean <- tm\_map(corpus\_clean, stripWhitespace) | Removes all excess space thus seperates words by a single space |
| sms\_dtm <- DocumentTermMatrix(corpus\_clean) | This will tokenize the corpus and return the sparse matrix with the name sms\_dtm. |
| sms\_raw\_train <- sms\_raw[1:4169, ]  sms\_raw\_test <- sms\_raw[4170:5559, ] | Splits the raw data frame for training and testing |
| sms\_dtm\_train <- sms\_dtm[1:4169, ]  sms\_dtm\_test <- sms\_dtm[4170:5559, ] | Splits the Document Term Matrix |
| sms\_corpus\_train <- corpus\_clean[1:4169]  sms\_corpus\_test <- corpus\_clean[4170:5559] | Splits the Corpus |
| prop.table(table(sms\_raw\_train$type))  prop.table(table(sms\_raw\_test$type)) | Checks the proportion of spam in the training and test frames are similar (ie subsets are representative) |
| wordcloud(sms\_corpus\_train, min.freq = 40, random.order = FALSE) | Visual representation of word frequency within a tm corpus object. Set min.freq to 10% of number of documents in the corpus. |
| spam <- subset(sms\_raw\_train, type == "spam") | Creates a subset of SMS messages of spam type |
| ham <- subset(sms\_raw\_train, type == "ham") | Creates a subset of SMS messages of ham type |
| wordcloud(spam$text, max.words = 40, scale = c(3, 0.5)) | Creates a visual representation with the 40 most used words. |
| wordcloud(ham$text, max.words = 40, scale = c(3, 0.5)) | Creates a visual representation with the 40 most used words. |
| findFreqTerms(sms\_dtm\_train, 5) | Finds all terms that are mentionned at least 5 times |
| sms\_freq\_words <- findFreqTerms(sms\_dtm\_train, 5) | Saves frequent words into an object for later use |
| sms\_dtm\_freq\_train <- sms\_dtm\_train[ , sms\_freq\_words] | Prepares training dataset |
| sms\_dtm\_freq\_test <- sms\_dtm\_test[ , sms\_freq\_words] | Prepares testing dataset |
| convert\_counts <- function(x) {  x <- ifelse(x > 0, "Yes", "No")  } | Custom function that converts counts to Yes/No strings. |
| apply() | Works like lapply() but instead takes 3 inputs: the dataset to apply the function to, either a row (MARGIN=1) or a column (MARGIN=2) and the function to be applied. |
| sms\_train <- apply(sms\_dtm\_freq\_train, MARGIN = 2, convert\_counts) | Converts sms\_dtm\_freq\_train to Yes/No |
| sms\_test <- apply(sms\_dtm\_freq\_test, MARGIN = 2, convert\_counts) | Converts sms\_dtm\_freq\_test to Yes/No |
| sms\_train\_labels <- sms\_raw[1:4169, ]$type sms\_test\_labels <- sms\_raw[4170:5559, ]$type | Creates the par of vectors with labels for each of the rows in the training and testing matrices. |
| sms\_classifier <- naiveBayes(sms\_train, sms\_train\_labels)  sms\_test\_pred <- predict(sms\_classifier, sms\_test)  CrossTable(sms\_test\_pred, sms\_test\_labels, prop.chisq = FALSE, prop.t = FALSE, dnn = c('predicted', 'actual')) | Runs the naiveBayes algorithm to train the machine  Makes a prediction based on training  Evaluates model performance |
| sms\_classifier2 <- naiveBayes(sms\_train, sms\_train\_labels,  laplace = 1)  sms\_test\_pred2 <- predict(sms\_classifier2, sms\_test)  CrossTable(sms\_test\_pred2, sms\_test\_labels,  prop.chisq = FALSE, prop.t = FALSE, prop.r = FALSE,  dnn = c('predicted', 'actual')) | Better method of using naiveBayes by adding a Laplace estimator to avoid every message with the word ringtone being interpretted as spam.  Model evaluation |

1. Divide and Conquer – Classification Using Decision Trees and Rules

|  |  |
| --- | --- |
| set.seed(123)  train\_sample <- sample(1000, 900) | Use to pick 900 random samples from 1000 ordered sample data |
| credit\_train <- credit[train\_sample, ]  credit\_test <- credit[-train\_sample, ] | Splits dataset into training and testing objects |
| install.packages("C50")  library(C50) | Useful divide & conquer algo package |
| credit\_train$default<-as.factor(credit\_train$default) | Used to convert non-factor stuff to factor |
| credit\_model <- C5.0(credit\_train[-17], credit\_train$default) | Turning an R object into a C5.0 decision tree with the following info:  Call:  C5.0.default(x = credit\_train[-17], y = credit\_train$default)  Classification Tree  Number of samples: 900  Number of predictors: 20  Tree size: 54  Non-standard options: attempt to group attributes |
| summary(credit\_model) | Contains the following:  checking\_balance in {> 200 DM,unknown}: 1 (412/50)  checking\_balance in {< 0 DM,1 - 200 DM}:  :...other\_debtors = guarantor:  :...months\_loan\_duration > 36: 2 (4/1)  …  = *If checking account balance is unknown or greater than 200 DM, then classify as "not likely to default."*  *Otherwise, if the checking account balance is less than zero DM or between*  *one and 200 DM.*  *And …*  Evaluation on training data (900 cases):  Decision Tree  ----------------  Size Errors  54 135(15.0%) <<  (a) (b) <-classified as  ---- ----  589 44 (a): class 1  91 176 (b): class 2  O*utput indicates an error rate of 15%. 44 false positives, 91 false negatives.* |
| credit\_pred <- predict(credit\_model, credit\_test) | Predicts the future decisions based on model (used to evaluate model performance). |
| CrossTable(credit\_test$default, credit\_pred, prop.chisq = FALSE, prop.c = FALSE, prop.r = FALSE, dnn = c('actual default', 'predicted default')) | Correctly predicted 59 no default and 14 default out of 100 thus 73% accuracy. |
| credit\_boost10 <- C5.0(credit\_train[-17], credit\_train$default, trials = 10) | Boosting model performance by using multiple weak performaning learners with different strengths together. (always start with 10 trials) |
| > summary(credit\_boost10)  > credit\_boost\_pred10 <- predict(credit\_boost10, credit\_test)  > CrossTable(credit\_test$default, credit\_boost\_pred10, prop.chisq = FALSE, prop.c = FALSE, prop.r = FALSE, dnn = c('actual default', 'predicted default')) | Used to show algorithm improved its performance by having only 29 mistakes on 900 training examples however its performance only went up to 76%.  The lack of an even greater improvement may be a function of our relatively small  training dataset, or it may just be a very difficult problem to solve. |
| > matrix\_dimensions <- list(c("no", "yes"), c("no", "yes"))  > names(matrix\_dimensions) <- c("predicted", "actual") | Creates a cost matrix. Predicted and actual values take two values yes or no so 2x2 matrix composed of two vectors each with two values. Also important to name matrix dimensions to avoid later confusion. matrix\_dimensions now holds:  $predicted  [1] "no" "yes"  $actual  [1] "no" "yes" |
| error\_cost <- matrix(c(0, 1, 4, 0), nrow = 2, dimnames = matrix\_dimensions) | Creates the following matrix :  **actual**  **predicted** no yes  no 0 4  yes 1 0 |
| > credit\_cost <- C5.0(credit\_train[-17], credit\_train$default, costs = error\_cost)  > credit\_cost\_pred <- predict(credit\_cost, credit\_test)  > CrossTable(credit\_test$default, credit\_cost\_pred, prop.chisq = FALSE, prop.c = FALSE, prop.r = FALSE, dnn = c('actual default', 'predicted default')) | Same algorithm call but with a cost function added as a parameter.  This artifically filters out damaging false positives.  Does not necessarily lead to better performance, this is a trade off. |
| mushrooms <- read.csv("mushrooms.csv", stringsAsFactors = TRUE) | Import the data into mushrooms object |
| mushrooms$veil\_type <- NULL | Since veil type is always the same value for all samples it cannot be used for prediction thus it must be dropped as shown. |
| mushroom\_1R <- OneR(type ~ ., data = mushrooms) | Creates rules using 1R algo |
| > mushroom\_1R  > summary(mushroom\_1R) | Reveals accuracy of ~99% |
| mushroom\_JRip <- JRip(type ~ ., data = mushrooms) | Creates rules using RIPPER algo |
| > mushroom\_JRip | Following rules: |

1. Forecasting Numeric Data - Regression Methods

|  |  |
| --- | --- |
| a <- y\_bar - b\*x\_bar | Calculation for linear regression via Ordinary Least Squares (OLS) a? |
| b <- cov(launch$temperature, launch$distress\_ct) / var(launch$temperature) | Calculation for linear regression OLS b. |
| a <- mean(launch$distress\_ct) - b \* mean(launch$temperature)  > a | Estimation for linear regression OLS a. |
| r <- cov(launch$temperature, launch$distress\_ct) / (sd(launch$temperature) \* sd(launch$distress\_ct)) | Explicit calculation in R for Pearson’s correlation |
| cor(launch$temperature, launch$distress\_ct) | Simplest way to calculate Pearson’s correlation |
| reg <- function(y, x) {  x <- as.matrix(x)  x <- cbind(Intercept = 1, x)  b<-solve(t(x) %\*% x) %\*% t(x) %\*% y  colnames(b) <- "estimate"  print(b)  } | Simple multivariable linear regression function creation.  - as.matrix() function is used to convert the data frame into matrix form  - cbind() function is used to bind an additional column onto the x matrix  - Intercept = 1 instructs R to name the new column Intercept and  to fill the column with repeating 1 values.  - solve() takes the inverse of a matrix  - t() is used to transpose a matrix  - %\*% multiplies two matrices |
| reg(y = launch$distress\_ct, x = launch[2]) | Should do the univariate simple regression for us as detailed above. |
|  |  |