|  |  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| <-: assignment  c(): fn to concatenate nums inside () to form vector (1 row vector)  commenting | | | | | | | | | x <- c(1,3,2,5)  x  #comment | | | 1 3 2 5 |
| bring up documentation of fn | | | | | | | | | ?fn\_name: | | | |
| +, - , \*, /, %%, works per normal with vector  <, <=, ==, >=, >  logarithm, log() by default use base e  check if vector elem in vector X | | | | | | | | | x + 5  x < 4  x + c(1,0,0,0)  log2(...) || log10(...) ||  c(a, b, c) %in% X | | | 6 8 7 10  True, T, T, F  2 3 2 5  log(..., base)  FALSE FALSE TRUE |
| s <- c(TRUE, TRUE, FALSE, FALSE), t <- c(TRUE, FALSE, TRUE, FALSE)  s & t: and  s && t (only 1st elem of each vector)  s | t: or  s || t (only 1st elem of each vector)  !s | | | | | | | | | TRUE FALSE FALSE FALSE  TRUE  TRUE TRUE TRUE FALSE  TRUE  FALSE FALSE TRUE TRUE | | | |
| length of vector | | | | | | | | | length(x) | | | 4 |
| square root of each elem  raise elem to power i (fractional/-ve power ok) | | | | | | | | | sqrt(c(4,9,16))  x^2 | | | 2 3 4  1 9 4 25 |
| a:b : create vector in ascending order from a to b including  seq(a,b,step): create vector in ascending order with step from a to b  repeat seq of 1 to i, until vector length = n | | | | | | | | | 1:5  seq(1,10,2)  rep(1:i, length.out = n)) | | | 1 2 3 4 5  1 3 5 7 9 |
| list all objs created  delete unwanted objects (remove all objects: rm(list = ls()) ) | | | | | | | | | ls()  rm(x) | | | "x" |
| mean  getmean <- function(v) { #defining own function  return(sum(v)/length(v)) } | | | | | | | | | mean(x) | | | 2.75 |
| getmean(x) | | | 2.75 |
| middle value if N is odd/ average of middle values if N is even | | | | | | | | | median(x) | | | 2.5 |
| sample variance  standard deviation = sqrt(var) | | | | | | | | | var(x)  sd(x) || sqrt(var(x)) | | | 2.916667  1.707825 |
| max and min of vector | | | | | | | | | max(x) - min(x) | | | 4 |
| sample covariance  if dont use cov(), can do sum((x-mean(x))\*(y-mean(y)))/(length(x)-1) | | | | | | | | | cov(x,c(2,6,0,9)) | | | 6.083333 |
| sample correlation coeff.  -1<= coeff <= 1 always hold and invariant to scale and location change | | | | | | | | | cor(x,c(2,6,0,9)) | | | 0.8836322 |
| Location change: if u = x + a and v = y + b, where a and b are constant mean and median for u will be different from x (similarly for v and y) | | | | | | | | | var, sd, cov and cor are invariant to change in location | | | |
| Scale changes: if u = a\*x and v = b\*y where a and b are constants  mean and median for u will be diff from x (similarly for v and y) | | | | | | | | var, sd, cov will be diff  var(u) = a^2\*var(x), sd(u) = |a|\*sd(x)  cor is invariant if a and b are of the same sign | | | | |
| import csv file from absolute file path  change working directory to one containing csv file  get current working directory  get files in current working directory  "import" r script into console (else highlight script and click run) | | | | | | | | x <- read.csv(absolute file path)  setwd("directory")  getwd()  list.files()  source('....R') | | | | |
| read.table("...")  read.csv("...")  read.csv2("...")  read.delim("...")  read.delim2("...") | Headers  false  true  true  true  true | | | | | | | Separator  " "  ","  ";"  "\t"  "\t" | | | Decimal Point  "."  "."  ","  "."  "," | |
| first few rows of data  summary statistics such as mean median quantiles  total rows and columns of data set(dimensions)  header names/column names | | | | | | | | head(x)  summary(x)  dim(x)  names(x) | | |  | |
| column data from name  column data from index  rows of data  generally: dataframe[row indices, col indices]  sales$gender <- as.factor(sales$gender)  exclude col i of df | | | head(sales$gender)  head(sales[,4])  head(sales[1:2,])  head(sales[,c(1,3,4)])  head(sales[,c("sales\_total", "gender")])  head(sales[sales&$gender=="F",])  head(sales[sales&sales\_total>500,])  change column to vector  df[,-i] | | | | | | | gender column  4th column  1st 2 rows  1,3,4 column  total sales & gender col  rows with gender=="F"  rows with sales>500 | | |
| create png file  plot graph with xlab = x axis label(similar for ylab), col = color of data points, type: type of graph, lty: line type  stop plotting | | | | | png("name.png")  plot(x = sales$col1, y = sales%col2, xlab="x label", ylab = "...", col = "red", type = "1", lty = 1)  dev.off() | | | | | | | |
| create histogram where breaks = num of cells | | | | | hist(x = sales$sales\_total, breaks=500, col = "red") | | | | | | | |
| create table  get array of indices where cond is true  get y where col1 == cond | | | | | table(df$col1)  x1 = which(df$col1 == cond)  df$y[x1] | | | | | | | |
| if (cond1 && cond2) {} else {}  -or ||  - && and || only apply to first 2 elem of corresponding vector  sort() sort vector | | | | | if () {  return()  }  else {}  ifelse(cond, if true, if false) | | | | | | | |
| for (i in 1:5) {print(i)}  for (i in 1:n) {sum <- sum+i}  while (i < 5) { i <- i+2  print(i)} | | | | | repeat{print(i)  if (i==5) {break}  i <- i+1} | | | | | | | |
| to type multiline without script  to install some package  import package | | | | | SHIFT + ENTER  install.packages("...")  library(...) (class for knn) | | | | | | | |
| predict y value  confusion matrix | | | | | knn.pred = knn(train.x, test.x, train.y, k = 1)  confusion.matrix = table(knn.pred, test.y) | | | | | | | |
| create matrix (byrow specify whether data entered in row/col)  create empty vector  then can use for loop to enter num into dist  join 2 matrix like for augmented matrix  sort matrix by a column i  sum diagonal entries of matrix | | | | | | X <- matrix(c(1,2,3,4,5,6,7,8,9), nrow = 3, ncol = 3, byrow = TRUE  dist <- numerice(nrow(X))  dist[i] <- some calculation  cbind(X, dist)  final[order(final[,i], decreasing = FALSE),]  diag(X) | | | | | | |
| get boolean vector of whether data in col1 < 2000  OR 1st k rows (decide how to split data to train and test)  filter data by whether data in col1 < 2000  filter data by whether data in col1 ≥ 2000  get data for X from relevant col for training data  get data for X from relevant col for test data  get data for Y for training data  get data for Y for test data  insert into knn model  get confusion matrix | | | | | | train = df$col1 < 2000  OR train = 1:k  train.data = df[train,]  test.data = df[!train,]  train.x = train.data[,c("x1","x2","x3","x4")]  test.x = test.data[,c("x1","x2","x3","x4")]  train.y = train.data[,c("y1")]  test.y = test.data[,c("y1")]  knn.pred = knn(train.x, test.x, train.y, k = 1)  confusion.matrix = table(knn.pred, test.y) | | | | | | |
| to get same result  standardise X values (all col have mean = 0, var = 1) where col j = col of Y values  standardise X values of dataframe  choose random s rows from 1 to n (randomise choosing of training data) | | | | | | | | | set.seed(...)  standardized.X = scale(df[,-j])  OR df[,-j] <- lapply(df[,-j], scale)  train = sample(1:n, s)  train.x = df[train] #test.x = df[-train] | | | |
| plot graph with 2 X values, and y as color  pch: to fill the points?  plot additional points, x1, x2, x3 | | | | | | colors <- ifelse(y==0, "red", "black")  plot(x = x1, y = x2, col = colors, pch = 16)  points(c(x1.1,x1.2,x1.3), c(x2.1,x2.2,x2.3)) | | | | | | |
| decision trees  return rpart obj (method: indicate type of y ["class"ification], minsplit: min num of y to split, split: type of algo to use [infomation gain])  plot decision tree  predict classification (type: what output to get [prob: table, class: y value] | | | library(rpart)  library(rpart.plot)  fit <- rpart(y ~x1 + x2+ x3+ ..., method = "class", data = df, control =  rpart.control(minsplit = 1), parms = list(split='information'))  rpart.plot(fit, type = 4, extra = 2, clip.right.labs = FALSE, varlen = 0, faclen = 0)  predict(model, newdata = newdata, type = "prob") | | | | | | | | | |
| n-fold cross-validation  put each data into different datasets  get indices array of data that belongs to current testset, j  get training data, not including those in current testset  cp: complexity parameter (decide how large tree will become)  higher cp, smaller depth of tree  get x values of data belonging to testset  predict using model and testdata  add wrong predictions to error | | | | | | n\_folds = 10  folds\_j <- sample (rep(1:n\_folds, length.out = n)  for (j in 1:n\_folds) {  test <- which(folds\_j == j)  train = df[-test, ]  fit <- rpart(y ~ x1 + x2 +..., method = "class", data = train,  control = rpart.control(cp = 0.01),  parms = list(split = 'information'))  test.data = data.frame(df[test, x1+x2+...)  pred = predict(fit, test.data, type = 'class')  error = error + sum(pred != df[test, y]) | | | | | | |
| check if data is in dataframe, if not convert to dataframe  extract Y col in table form  compute prob for P(Y = yj)  do the same for P(Xi = xi|Y = yj)  #repeat for x2, x3,...  compute prob for P(Y = yj|X)  #repeat for prob\_y2, prob­\_y3...  #highest prob = predicted class for testdata  #naives Bayes  laplace: smooths curve to prevent prob tending to 0 (default is 0 which disables laplace smoothing  stores predicted class | | | | | | as.dataframe(df[ , ])  probs <- table(traindata$Y)  probs <- probs/sum(probs)  x1 <- table(traindata[, c("Y", "x1")]);  x1 <- x1/rowSums(x1)  prob­­\_y1 <- x1["y1", testdata[, c("x1")]] \*  x2["y1", testdata[, c("x2")]] \*...  library(e1071)  model <- naivesBayes(Y ~ x1+x2+x3..., traindata, laplace=0.5)  results <- predict(model, testdata) | | | | | | |
| get linear regression gradient and intercept [grad, intercept] | | | | | | lm(y ~ x) (works for more than one feature var as well, y~x1+x2) | | | | | | |
| logistic function (in this e.g., z = x. But can be z = ß0 + ß1x1, then do plot(x, logistic(ß0 + ß1 \* x1)) for some value of ß0, ß1)  logistic regression model (Y~. : Y against all x columns)  see values of ß0, ß1, ... | | | | | | logistic <- function(z) {exp(z) / (1 + exp(z)) }  x <- seq(-10, 10, 0.1)  plot(x, logistic(x), xlab = 'z', ylab = 'f(z)', lty = 1, type = 'l')  model <- glm(Y ~. , data = df, family = binomial(link="logit"))  model | | | | | | |
| Receiving Operating Characteristic (ROC)  making predictions (type='response' for glm models)  creating prediction object  evaluating performance  plot ROC curve  predictions (type='raw' gives vector of prob for diff class), (here we only want second col for pred class)  #add=TRUE add graph to existing graph, col = 2 = red  add legend | | | | library(ROCR)  pred = predict(model, type = 'response') #logistic  predObj = prediction(pred, True\_Y\_values)  roc = performance(predObj, measure = "tpr", x.measure = "fpr")  plot(roc)  nb\_pred <- predict(nb\_model, df, type = 'raw')  score <- nb\_pred[,2]  nb\_predObj <- prediction(score, True\_Y\_values)  nb\_roc <- performance(nb\_predObj, measure = "tpr", x.measure = "fpr")  plot(nb\_roc, add = TRUE, col = 2)  legend("bottomright", c("logistic", "nb"), col=c("black", "red"), lty = 1) | | | | | | | | |
| Fit k-means clustering model (centers: num of clusters)  get vector of which cluster each data point belongs to  return all centroids  return vector of SS of each cluster  return WSS  color data point according to cluster it belongs to | | | | | | kout <- kmeans(df[, c("x1", "x2",...)], centers = k)  kout$cluster  kout$centers  kout$withinss  kout$tot.withinss OR sum(kout$withinss)  plot(df$x1, df$x2, col = kout$cluster) | | | | | | |
| Find appropriate value of k  nstart will run kmeans for each k 25 times as randomly selected centroid may not lead to right convergence  plot WSS against k | | | | | | wss <- numeric(15) #initialise vector  for (k in 1:15) {  wss[k] <- kmeans(df[,c("x1", "x2",...)], centers = k, nstart = 25)  $tot.withinss) }  plot(1:15, wss, type = "b", xlab = "k", ylab = "WSS") | | | | | | |
| Association Rules  preinstalled dataset  Check summary  display grocery labels and their categories (row: labels)  display which item labels appear in every transaction  get actual 100-105 transactions  #paste combine diff labels  #collapse specify how to combine diff labels  use apriori to get frequent itemsets  #min and maxlen specify length of itemsets  #target: type of association mined  display top 10 itemsets sorted by their support  use apriori to create candidate rules  create scatterplot of rules (jitter = off, allow overlap of data points)  display rules  plot rules with highest lift | | | | | | | library('arules'); library('arulesViz')  data(Groceries)  summary(Groceries)  Groceries@itemInfo[1:10,]  Groceries@data[, 100:110] #row: labels, col: transaction  apply(Groceries@data[, 100:105], 2,  function(r) paste(Groceries@itemInfo[r, "labels"],  collapse = ", ")  itemsets <- apriori(Groceries, parameter = list(minlen = 1, maxlen = 10, support = 0.02,  target = "frequent itemsets"))  inspect(head(sort(itemsets, by = "support"), 10)  rules <- apriori(Groceries, parameter = list(support = 0.001, confidence = 0.6, target = "rules")  plot(rules, jitter = off) #x: support, y: lift  inspect(head(sort(rules, by = "lift"), 5)  plot(head(sort(rules, by = "lift"), 5), method = "graph", control = list(alpha = 1)) | | | | | |
| Groceries dataset in arules package is of class transactions | | transaction class contains: 1. transactionInfo (dataframe with vectors of same length as num of transactions), 2. itemInfo (dataframe storing item labels),  3. data (binary incidence matrix indicating which item labels appear in every transaction) | | | | | | | | | | |