R Code

♠ library(MASS)
lda(),qda()≈lm()
♠ library(class)
knn(train, test, cl, k = 1, l = 0, prob = FALSE, use.all = TRUE)
♠ library(boot)
cv.glm(data, glmfit, cost, K)
boot(data, statistic, R, sim = "ordinary")
♠ library(leaps)
S3 method for class 'formula'
regsubsets(x=, data=, nvmax=8, force.in=NULL, force.out=NULL, intercept=TRUE,
method=c("exhaustive", "backward", "forward", "seqrep"))
♠ library(glmnet)
glmnet(x, y, family=c("gaussian","binomial", "poisson", "multinomial","cox","mgaussian"), alpha = 1, lambda=NULL))
alpha: The elasticnet mixing parameter, with 0¡=alpha¡= 1. The penalty is defined as
$(1-\alpha)/2||\beta||_2^2+\alpha||\beta||_1$.
'alpha=1' is the lasso penalty, and 'alpha=0' the ridge penalty.
S3 method for class 'glmnet'
predict(object, newx, s = NULL, type=c("link", "response","coefficients","nonzero","class"), exact = FALSE, offset, ...)
newx: Matrix of new values for 'x' at which predictions are to be made. Must be a matrix; can be sparse as in 'Matrix' package. This argument is not used for 'type=c("coefficients","nonzero")'
♠ library(pls)
mvr(formula, ncomp, data, subset, method = pls.options()$mvralg, scale = FALSE, validation = c("none", "CV", "LOO"), model = TRUE, x = FALSE, y = FALSE, ...)
plsr(..., method = pls.options()$plsralg)
pcr(..., method = pls.options()$pcralg)
♠ library(splines)
bs(x, df = NULL, knots = NULL, degree = 3, intercept = FALSE, Boundary.knots = range(x))
ns(x, df = NULL, knots = NULL, intercept = FALSE, Boundary.knots = range(x))
smooth.spline(x, y = NULL, w = NULL, df, spar = NULL, cv = FALSE, all.knots = FALSE, nknots = .nknots.smspl, keep.data = TRUE, df.offset = 0, penalty = 1, control.spar = list(), tol = 1e-6 * IQR(x))
♠ library(gam)
gam(formula, family = gaussian, data, weights, subset, na.action, start, etastart, mustart, control = gam.control(...), model=TRUE, method, x=FALSE, y=TRUE, ...)
lo(..., span=0.5, degree=1)
gam.lo(x, y, w, span, degree, ncols, xeval)
♠ library(gam)
tree(formula, data, weights, subset, na.action = na.pass, control = tree.control(nobs, ...), method = "recursive.partition", split = c("deviance", "gini"), model = FALSE, x = FALSE, y = TRUE, wts = TRUE, ...)
prune.tree(tree, k = NULL, best = NULL, newdata, nwts, method = c("deviance", "misclass"), loss, eps = 1e-3)
cv.tree(object, rand, FUN = prune.tree, K = 10, ...)
♠ library(randomForest)
S3 method for class 'formula'
randomForest(formula, data=NULL, ..., subset, na.action=na.fail)
Default S3 method:
randomForest(x, y=NULL, ntree=500, mtry=if (!is.null(y) && !is.factor(y)) max(floor(ncol(x)/3), 1) else floor(sqrt(ncol(x))), importance=FALSE, proximity, oob.prox=proximity)
S3 method for class 'randomForest'
importance(x, type=NULL, class=NULL, scale=TRUE, ...)
♠ library(gbm)
gbm(formula = formula(data), distribution = "bernoulli", data = list(), n.trees = 100, interaction.depth = 1, n.minobsinnode = 10, shrinkage = 0.001, cv.folds=0, verbose = "CV")
Currently available options are "gaussian" (squared error), "laplace" (absolute loss), "tdist" (t-distribution loss), "bernoulli" (logistic regression for 0-1 outcomes), "huberized" (huberized hinge loss for 0-1 outcomes), "multinomial" (classification when there are more than 2 classes), "adaboost" (the AdaBoost exponential loss for 0-1 outcomes), "poisson" (count outcomes), "coxph" (right censored observations), "quantile", or "pairwise" (ranking measure using the LambdaMart algorithm).

Nouns

cross-entropy(deviance)
feedforward neural network
back propagation algorithm
Markov random field
Restricted Boltzman Machines
energy function
contrastive divergence
generalized cross-validation
Adaboost.M1 algrithm
forward stagewise additive modeling
PRSS, Penalized sum of squares
backfitting
classification Error rate(Misclassification rate)
Gini Index
Cross-Entropy or Deviance
softmax function
Parzen estimate(Kernel)
varying coefficient model
Nadaraya-Watson kernel-weight average
locally weighted linear regression
Epanechnikov/Tri-Cube kernel
Gaussian kernel
Nearest Neighbor kernel
locally polynomial regression
stochastic search variable selection(SSVS)