|  |  |  |
| --- | --- | --- |
| **## SPLIT INTO TEST AND TRAINING DATASETS FIRST**  train <- sample(1:nrow(dat), nrow(dat)\*0.8)  trainset <- dat[train,]  testSet <- dat[-train,]  #bintrain <- rep(T, nrow(dat)) # logical vector  **## LOGIT** logit <- glm(y ~ x, data = , family = binomial, subset = train) #type=response for probabilities logit.pred <- predict(logit, newdata = , type = “response”) table(logit.pred, test$y) #confusion matrix  **## K-NN** library(class) knn <- knn(train = train$x, test = test$x, cl = train$y, k = k)  table(knn, test$test) # confusion matrix  mean(knn == test$y) # accuracy  **## LOOCV**  - approx. unbiased estimate of test error (almost no bias)  - high variance  glm <- glm(y ~ x, data = dat)  # cross-validation  library(boot)  cv <- cv.glm(dat, glm)  cv$delta[1] # LOOCV statistic  **## K-FOLD CV**  - approx. medium level of bias  - medium variance  library(boot)  cv.kfold <- cv.glm(dat, glm, K = 10) # low K, low flexibility  cv.kfold$delta # 1st num is std err est., 2nd is bias correctd  **## BOOTSTRAP**  # create function first  alpha <- function(data, index) {  X <- data$X[index]  Y <- data$Y[index]  return((var(Y)-cov(X,Y))/(var(X)+var(Y)-2\*cov(X,Y)))  }  library(boot)  boot(data, alpha, R = 1000)  **## BEST SUBSET SELECTION**  library(leaps)  best.sub <- regsubsets(y ~ x, data = , nvmax = ncol(dat)-1) best.sum <- summary(best.subset)  names(best.sum) # look at summary of best subset select  best.sum$rsp # show R^2 of models  coef(best.sub, 6) # get coef of model with 6 predictors  **## FORWARD & BACKWARD STEPWISE SELECTION**  library(leaps)  forward <- regsubsets(y ~ x, data = dat,  nvmax = ncol(dat)-1, method = “forward”) backward <- regsubsets(y ~ x, data = dat,  nvmax = ncol(dat)-1, method = “backward”)  coef(forward, 7)  coef(backward, 7)  **## RIDGE**  # make x & y: MUST DO FOR RIDGE  x <- model.matrix(y ~ x, data = dat[,-1] # exclude intercpt  y <- dat$y  library(glmnet)  ridge <- glmnet(x, y, alpha = 0) # alpha = 1 for LASSO  length(ridge$lambda) # used 100 lambdas  ridge$lambda[1]  coef(ridge)[1]  # CV choosing tuning parameter  cv.ridge <- cv.glmnet(x[train,], y[train], alpha = 0) # k=10  cv.lambda <- cv.ridge$lambda.min # get smallest lambda  ridge.final <- glmnet(x, y, alpha = 0)  predict(ridge.final, s = cv.lambda, type = “coefficients”)  **## PRINCIPAL COMPONENTS ANALYSIS**  pca=prcomp(data, scale=TRUE) #make vars mean 0 or 1  names(pca)  pca$rotation #loading vectors  biplot(pca, scale=0) #loading vectors in red, each numb.  represent one obs.  pca$sdev  pvar=pca$sdev^2  pve=pvar/sum(pvar)  pve  plot(pve,xlab=’Principle Component”, ylab=”Proportion of variance explained”, ylim=c(0,1), type=’b’)  **## K-MEANS CLUSTERING**  kmeans=kmeans(x, centers=2, nstart=20) # center is number of clusters  kmeans$cluster  plot(x,col=(kmeans$cluster))  **## HIERARCHIAL CLUSTERING**  hi.clust= hclsut(dist(x), method = “complete”)  plot(hi.clust)  cutree(hi.clust,2)  plot(x,col=cutree(hi.clust, 2))  **## INSTRUMENTAL VARIABLE**  #IV by hand: 2sls, fisrt stage:  fisrt=lm(z~x, data=data)  summary(fisrt)  data$hat = predict(first)  #second stage:  second=lm(y~x+hat, data=data)  summary(second)  #shortcut  library(AER)  ivreg <-ivereg(y~x|exo-x+z, data=data) | **## LASSO**  library(hdm)  # post-lasso to reduce bias of estimator: OLS applied to the data after removing vars not selected by LASSO  lasso <- rlasso(y ~ x, data = , post = T)  # OR manually do OLS  x <- which(coef(lasso)[-1] != 0)  x <- paste(names(x), sep = “”, collapse = “+”)  formula <- paste(c(“y”, x), collapse = “ ~ ”)  ols <- lm(formula, data = )  **## DOUBLE LASSO**  library(hdm)  f <- the focal variables we are interested in  # step 1: LASSO of y on controls  step1 <- rlasso(y ~ . –f, data = , post = F)  cy <- which(coef(step1)[-1] != 0)  # step 2: LASSO of f on c  step2 <- rlasso(f ~ . –y, data = , post = F)  cf <- which(coef(step2)[-1] != 0)  # step 3: union of kept vars  union <- c(names(cy), names(cf))  # step 4: OLS of y on f and kept vars  unionf <- paste(c(“f”, union), collapse = “+”)  formula <- paste(“y”, unionf), collapse = “ ~ “)  ols <- lm(formula, data = )  **## BASIS FUNCTION**  # polynom reg of y on 4th deg polynom in x  poly <- lm(y ~ poly(x, 4), data = )  table(cut(data$x, 4)) # fit a step function  step = lm(y ~ cut(x, 4), data = )  **## SPLINES**  # fit y to x using reg spline (cubic is default)  library(splines)  spline <- lm(y ~ bs(x; knots = c(a,b,c)), dat = )  xlims <- range(data$x)  x.grid <- seq(from = xlims[1], to = xlims[2])  spline.pred <- predict(spline,  newdata = list(x = x.grid), se = T)  # smoothing spline  smth.spln <- smooth.spline(dat$x, dat$y, cv = T)  **## REGRESSION TREE**  library(tree)  reg.tree <- tree(y ~ x, data = , subset = train)  plot(reg.tree)  text(reg.tree)  pred.tree <- predict(reg.tree, newdata = data[-train])  plot(pred.tree, test$y)  abline(0, 1)  mean((pred.tree – test$y)^2) # MSE  **## CLASSIFICATION TREE**  library(tree)  class.tree <- tree(y ~ x, data = , subset = train)  plot(class.tree)  text(class.tree, pretty = 0)  pred.class <- predict(class.tree, newdata = data[-train,],  type = “class”)  table(pred.class, dat$y[-train])  **## BAGGING**  library(randomForest)  bagging <- randomForest(y ~ x, data = , subset = train,  mtry = ncol(dat)-1, importance = T)  pred.bag <- predict(bagging, newdata = dat[-train,])  plot(pred.bag, dat$y[-train])  abline(0, 1)  mean((pred.bag – dat$y[-train])^2) # MSE | **## RANDOM FOREST**  library(randomForest)  rf <- randomForest(y ~ x, data = , subset = train, mtry = 6,  importance = T)  rf.pred <- predict(rf, newdata = data[-train,])  plot(rf.pred, dat$y[-train])  abline(0, 1)  mean((rf.pred – dat$y[-train])^2) # MSE better than bagg  importance(rf)  varImpPlot(rf)  **## BOOSTING**  library(gbm)  # if classification problem: distribution = “bernoulli”  boost <- gbm(y ~ x, data = dat[train,], distribution =  “gaussian”, n.trees = 5000, interaction.depth = 4)  boost.pred <- predict(boost, newdata = dat[-train,],  n.trees = 5000)  mean((boost.pred – dat$y[-train])^2) # MSE  **## INDIVIDUAL CONDITIONAL EXPECTATIONS**  ex. Do random forest first  # ICE curves and c-ICE curves  library(pdp)  ice <- partial(rf, pred.var = “x1”, ice = T)  ice1 <- plotPartial(ice, alpha = 0.5)  ice2 <- plotPartial(ice, center = T, alpha = 0.5)  grid.arrange(ice1, ice2, ncol = 2)  **## PARTIAL DEPENDENCE PLOTS**  ex. Do random forest first  # avg of ICE plots to investigate effect of changing x1  library(ggplot2)  pdp <- partial(rf, pred.var = “x1”)  autoplot(pdp, legend.title = “”)  # looking at interaction between x1 and x2  pdp2 <- partial(rf, pred.var = c(“x1”, “x2”, chull = T)  autoplot(pdp2, contour = T, legend.title = “”)  **## SURROGATE MODELS: a specific example**  # make prediction matrix  n <- 1000  cmean <- colMeans(Boston)  # fix all vars values at their means  pred.set <- matrix(data = cmean, nrow = ncol(Boston),  byrow = T)  colnames(pred.set) <- colnames(Boston)  # let lstat vary  pred.set[,”lstat”] <- rnorm(n, cmean[“lstat”], 1)  pred.set <- data.frame(pred.set)  # LIME (local interpretable model explanation)  yhat <- predict(rf, newdata = pred.set, type = “response”)  plot(pred.set$lstat, yhat)  ols <- lm(yhat ~ pred.set$lstat)  **## SUPPORT VECTOR CLASSIFIER**  # Y HAS TO BE FACTOR FOR SVM  dat <- data.frame(x = x, y = as.factor(y))  library(e1071)  svm1 <- svm(y ~ ., data = dat, kernel = “linear”,  cost = 10, scale = F)  plot(svm1, dat)  svm1$index # check the support vectors  # tuning the cost parameter  tune <- tune(svm, y ~ ., data = dat, kernel = “linear”,  ranges = list(cost = c(0.001, 0.01, 0.1, 1, 5, 10, 100)))  bestmodel <- tune$best.model  testdata <- data.frame(x = xtest, y = as.factor(ytest))  y.pred <- predict(bestmodel, testdata)  table(predict = y.pred, truth = testdata$y)  **## SUPPORT VECTOR MACHINE**  svm2 <- svm(y ~ ., data = dat[train,], kernel = “radial”,  gamma = 1, cost = 1)  library(e1071)  plot(svm2, dat2[train,]) |