A collection of algorithms for large data and regression analysis

Yasin Elmaci

Load libraries

```
library(glmnet)
library(foreach)
library(care)
library(crossval)
library(fdrtool)
library(randomForest)
library(mboost)
library(e1071)
```

Part I

I - Linear analysis

```
set.seed(694208008135)
library(lars) # diabetes dataset
data(diabetes) # load diabetes dataset
x = as.data.frame.matrix(diabetes$x) # extracts diabetes covariates
y = diabetes$y # extracts diabetes score


linear_model = lm(y~sex+age+bmi+glu+map+ltg,
                  data=x) # linear model

summary(linear_model) # summary of all components of a linear model

x1 =  cbind(rep(1,nrow(diabetes$x)), x$sex, x$age, x$bmi, x$glu, x$map, x$ltg)

# x1 - matrix of sex, age, bmi, glu, map and ltg with an intercept 1
# cbind() - bind vector/matrix by row or column
# rep() - replicates value inside function
# nrow() - number of rows in structure


OLS = solve(t(x1)%*%x1)%*%t(x1)%*%y # OLS regression

# t() - transposes the matrix
# solve() - invers the matrix
# %*% - matrix multiplication

x2 =  cbind(x$sex, x$age, x$bmi, x$glu, x$map, x$ltg)
```

```r
COV = solve(cov(x2))%*%cov(x2,y) # sample covariance based estimate

# x2 - a given matrix as x1 without intercept
# solve() - invers the matrix
# cov() - covariance
# %*% - matrix multiplication
```

II - Linear Prediction

```r
x_train = data.frame(x[1:300,]) # x training data
y_train = y[1:300] # y training data

x_new = data.frame(x[301:442,]) # x predict data
y_new = y[301:442] # y predict data

linear_training = lm(y_train~sex+age+bmi+glu+map+ltg,
                     data=x_train)
# training linear model

linear_prediction = predict.lm(linear_training, x_new)
# linear prediction model

squared_difference = (linear_prediction-y_new)^2
# squared difference

## Part I - Logistic analysis


y_binary = as.numeric(y>200) # initiating variable as binary (i.e y>x)

logistic_regression = glm(y_binary~sex+age+bmi+glu+map+ltg,
                          data=x,
                          family=binomial)
# logistic regression

# family = binomial transforms generalised function to be logistic

## Part I - Logistic prediction

ybin_train = y_binary[1:300] # initiating training variable as binary (i.e y>x)

logistic_training = glm(ybin_train~sex+age+bmi+glu+map+ltg,
                        data = x_train,
                        family=binomial)

eta = predict.glm(logistic_training,x_new) # logistic linear predictor
```

Inverse logit function ($logit^{-1}(eta) = exp(eta)/(exp(eta)+1)$) to transform the linear predictor ($eta = x\beta$) back to a probability which ranges between 0 and 1.

```r
logistic_prediction = (exp(eta)/(exp(eta)+1)) # inverse logit link
```

III - Linear mixed model

```r
load("E:/Imperial College London/Term 2/AR/Week1/exam.London")
dim(exam) #returns the dimensions of the dataset

# row x columns
# 3935 observations
# 10 covariates


# The standard linear model treats all observations independently and disregards
# potential group structures


fixed_effects = lm(normexam~standLRT+as.factor(school),
                   data=exam)
# fixed effects linear regression
# as.factor() - introducing covariates, (i.e grouped covariates such as number of schools)

library(nlme) # random effect package

random_intercept = lme(normexam~standLRT,
                       random = ~1|school,
                       data=exam)
# random = ~1|school - states that each school has a random intercept

summary(random_intercept) # extract StdDev for intercept and residual

std_intercept = 0.3071927
std_residual = 0.7535887
correlation_coefficient = std_intercept^2 / (std_intercept^2+std_residual^2)
# intraclass correlation coefficient

random_slope = lme(fixed=normexam~standLRT,
                   random = ~ 1 + standLRT | school,
                   data = exam)
# random = ~ 1 + standLRT | school - states that each slope is random depending on school

random_intercept_covariates= lme(normexam~
                                   standLRT + schavg + schgend,
                                 random = ~ 1 | school,
                                 data = exam)

# addition of covariates to model

anova(random_intercept, random_slope)
anova(random_intercept, random_intercept_covariates)

# anova() - anova test comparing models
```

Part II

I - Large data analysis

```r
set.seed(694208008135)
load("E:/Imperial College London/Term 2/AR/Week2/data_epigenetic_clock_control")
#load data

y = control_mice$y_control # extract control (age)

hist(y, breaks=50)

# hist() - plots histogram
# breaks - sets x-scale

x = control_mice$x_control # extract number of mice and methylation site

dim(x) #returns the dimensions of the dataset

# row x columns
# 409 observations
# 3663 covariates

linear_regression = lm(y~x[,1])
summary(linear_regression)$coefficients
# summary can be used for indexing p-values of coefficients

p_vec = rep(NA, dim(x)[2])
# create empty vector of NA of size dim(x)[2] to store covariates
# dim(x)[2] OR ncol(x) can be used

for(i in 1:ncol(x)){
  p_vec[i]=summary(lm(y~x[,i]))$coefficients[2,4]
}

# for-loop which iterates through every covariate and extracts the respective
# coefficient, iteration only through univariate regression

sites_ranked = cbind(colnames(x), p_vec) # combining names and p-values

sites_ordered = sites_ranked[order(p_vec, decreasing = F),]
# ordering ranked sites by p-values in decreasing order

head(sites_ordered, 10) # showing top 10 sites ordered by decreasing values
```

II - Multiple testing adjustment

```r
library(qvalue)

hist(p_vec, breaks=50) # plots distribution of p-values

# bonferroni correction with p-values below 0.05
# most conservative
```

```r
bonferroni = p.adjust(p_vec, method="bonferroni")
table(bonferroni<0.05)
output[which(bonferroni<0.05),]

# Benjamini-Hochberg FDR correction with p-values below 0.05
# most conservative FDR
bh = p.adjust(p_vec, method="BH")
table(bh<0.05)
output[which(bh<0.05),]

# q-value FDR correction with p-values below 0.05
qobj = qvalue(p=p_vec)
qvalues = qobj$qvalues
table(qvalues<0.05)
output[which(qvalues<0.05),]

qobj$pi0 # extract Null variable

# local FDR correction with p-values below 0.05
# leas conservative FDR
lfdr_out = fdrtool(x=p_vec, statistic="pvalue", verbose=FALSE)
table(lfdr_out$lfdr<0.2)
output[which(lfdr_out$lfdr<0.05),]

lfdr_out$param # extracts censored and eta values
```

Part III

I - Penalised regression

```r
set.seed(694208008135)
load("E:/Imperial College London/Term 2/AR/Week2/data_epigenetic_clock_control")

y = control_mice$y_control # extract control (age)

hist(y, breaks=50)

# hist() - plots histogram
# breaks - sets x-scale

x = control_mice$x_control # extract number of mice and methylation site
is.matrix(x)
# ensure x is a matrix

dim(x) #returns the dimensions of the dataset

# row x columns
# 409 observations
# 3663 covariates

# lasso cross validation optimising the mean squared error
```

```
lasso_cv = cv.glmnet(x,y,family="gaussian",alpha=1, type.measure="mse")
lasso_cv$lambda.min # extracts the lambda that minimises the mse
lasso_cv$lambda.1se # extracts the lambda that is the largest lambda
# (smallest model) that has a mse
# that is within 1 standard error of the minimum mse

# lasso regression with lambda min

lasso_min = glmnet(x,y,family="gaussian",alpha=1,lambda=lasso_cv$lambda.min)
sum(abs(lasso_min$beta)>0) # variables included in model

# lasso regression with lambda 1se

lasso_1se = glmnet(x,y,family="gaussian",alpha=1,lambda=lasso_cv$lambda.1se)
sum(abs(lasso_1se$beta)>0) # variables included in model


# regularisation parameter, cross-validation error and model size plots

par(mfrow=c(1,3))
plot(1:100, lasso_cv$lambda, main="1. Regularization parameter",
     xlab="Regularisation model", ylab="Lambda")
abline(v=which(lasso_cv$lambda == lasso_cv$lambda.min),col="red",lwd=2)
abline(v=which(lasso_cv$lambda == lasso_cv$lambda.1se),col="red",lty=2,lwd=2)
plot(1:100, lasso_cv$cvm, main="2. Cross-validation error",
     xlab="Regularisation model", ylab="Cross-validation error")
abline(v=which(lasso_cv$lambda == lasso_cv$lambda.min),col="red",lwd=2)
abline(v=which(lasso_cv$lambda == lasso_cv$lambda.1se),col="red",lty=2,lwd=2)
plot(1:100, as.vector(lasso_cv$nzero), main= "3. Model size",
     xlab="Regularisation model", ylab="Model size")
abline(v=which(lasso_cv$lambda == lasso_cv$lambda.min),col="red",lwd=2)
abline(v=which(lasso_cv$lambda == lasso_cv$lambda.1se),col="red",lty=2,lwd=2)
```

Figure 1 shows the regularisation parameter which decreases from around 1.4 to nearly zero. The first models have a strong regularisation while the last models only have very little regularisation. The vertical red line indicated the lambda with the minimum cv-error, and the dashed red line indicates the largest lambda (smallest model) within one standard error of the minimum cv-error.

Figure 2 shows the bias-variance trade-off of the test error. At first the cross-validation error reduces when reducing the regularisation parameter, but at some point there is a saturation. After reaching the minimum error it increases again when further reducing the regularisation parameter. There is a clear minimum of the cross-validation error at the 67th position of the lambda vector which allows us to define the optimal regularisation parameter (lambda.min). The largest lambda (sparsest model) within 1 standard error is at position 41.

Figure 3 shows the model complexity. With strong regularisation few variables are included into the model. When reducing the regularisation, the model size increases. The model with the lambda that miminises the MSE includes 301 variables, while the sparsest model within 1 standard error includes 158 variables.*

```
# ridge cross validation optimising the mean squared error

ridge_cv = cv.glmnet(x,y,family="gaussian",alpha==, type.measure="mse")
ridge_cv$lambda.min # extracts the lambda that minimises the mse
ridge_cv$lambda.1se # extracts the lambda that is the largest lambda
# (smallest model) that has a mse
```

```r
# that is within 1 standard error of the minimum mse

# lasso regression with lambda min

lasso_min = glmnet(x,y,family="gaussian",alpha=0,lambda=ridge_cv$lambda.min)
sum(abs(lasso_min$beta)>0) # variables included in model

# lasso regression with lambda 1se

lasso_1se = glmnet(x,y,family="gaussian",alpha=0,lambda=ridge_cv$lambda.1se)
sum(abs(lasso_1se$beta)>0) # variables included in model

# Ridge regression does not perform variable selection,
# all regression coefficients are unequal to zero and thus included into the model.

# elastic regression - dataframe structure to analyse min and 1se lambad at alpha fit

a = seq(0.05, 0.95, 0.05)
search = foreach(i = a, .combine = rbind)%do%{
  elasticnet_cv = cv.glmnet(x,y,family = "gaussian", type.measure = "mse", alpha = i)
  data.frame(cvm = elasticnet_cv$cvm[elasticnet_cv$lambda == elasticnet_cv$lambda.1se],
             lambda.1se = elasticnet_cv$lambda.1se,
             alpha = i)
}
elasticnet = search[search$cvm == min(search$cvm), ]
```

II - prediction rule

```r
# prediction rule function for training and test data

prediction_rule = function(train.x, train.y, test.x, test.y,
                           lambda = lambda, alpha=alpha){

  glmnet.fit = glmnet(x=train.x, y=train.y, lambda = lambda, alpha=alpha)
  ynew = predict(glmnet.fit , test.x)

  # compute squared error risk (MSE)
  out = mean( (ynew - test.y)^2 )
  return(out)
}

# rdige with K-fol cross-validation with B segments (i.e for lambda = 1se)

ridge_cv =  crossval(prediction_rule, x, y, lambda=ridge.cv$lambda.1se, alpha=0,
                         K=5, B=20, verbose=FALSE)
lasso_cv =  crossval(prediction_rule, x, y, lambda=lasso.cv$lambda.1se, alpha=1,
                         K=5, B=20, verbose=FALSE)
elasticnet_cv  =  crossval(prediction_rule, x, y, lambda=elasticnet.cv$lambda.1se,
                         alpha=elasticnet.cv$alpha, K=5, B=20, verbose=FALSE)

#table for comparison of mse and se

table.out = rbind(c(ridge_cv$stat, ridge_cv$stat.se),
```

```
                  c(lasso_cv$stat, lasso_cv$stat.se),
                  c(elasticnet_cv$stat, elasticnet_cv$stat.se))
rownames(table.out) = c("ridge", "lasso", "elastic net")
colnames(table.out) = c("mse", "se")
table.out
```

III - Shrinkage t-score

```
load("E:/Imperial College London/Term 2/AR/Week3/JAMA2011_breast_cancer")

y = data_bc$rcb
table(y)
x = data_bc$x
dim(x)

library(corpcor)
library(st)

sample.var = var.shrink(x,lambda=0) # sample variance with no shrinkage
shrink.var = var.shrink(x) # shrinkage variance with unspecified lambda

# boxplot template for comparing variances
boxplot(cbind(sort(sample.var)[1:1000],
              sort(shrink.var)[1:1000]),
        ylim=c(0,0.25),
names=c("sample variance", "shrinkage variance"))


shrink_t = shrinkt.stat(X=as.matrix(x), L=as.factor(y)) # shrinkage t-score

# for speed X and L may be defined outside of function


# multiple testing correction on t-statistic

FDR_shrinkage = fdrtool(shrinkt, statistic = "normal", verbose =FALSE)
sum(FDR_shrinkage$lfdr<0.2)

# statistic = "normal" - fit a Normal-distribution to the t-score
```

Part IV

I - Predicting treatment (classification)

```
# diagonal discriminant analysis

# ranking features using sda function
ranking.DDA = sda.ranking(x, y, diagonal=TRUE)

# factors passing <0.2 threshold and IDs
numVarsDDA = sum(ranking.DDA[,"lfdr"]<0.2)
numVarsDDA
```

```
selVars = ranking.DDA[,"idx"][1:numVarsDDA]
selVars

# build prediction rule for dda
dda.out = sda(x[, selVars, drop=FALSE], y, diagonal=TRUE)

dda.pred = predict(dda.out, x[, selVars, drop=FALSE], verbose=FALSE)

# evaluate: sensitivity (TPR)/specificity (TNR) based on confusion matrix

cM=confusionMatrix(as.character(y), as.character(dda.pred$class), negative="0")

TPR = cM[2]/(cM[2]+cM[4])
TNR = cM[3]/(cM[1]+cM[3])

# # area under the curve of the ROC curve

head(dda.pred$posterior) # case/ no case
roc.out=roc(y, dda.pred$posterior[,2])
plot(roc.out)
roc.out$auc

# linear discriminant analysis

# ranking
ranking.LDA = sda.ranking(x, y, diagonal=FALSE)
numVarsLDA = sum(ranking.LDA[,"lfdr"]<0.2)
numVarsLDA

selVars = ranking.LDA[,"idx"][1:numVarsLDA]

# build prediction rule for lda
lda.out = sda(x[, selVars, drop=FALSE], y, diagonal=FALSE)
lda.pred = predict(lda.out, x[, selVars, drop=FALSE], verbose=FALSE)

# evaluate: sensitivity  and specificity based on confusion matrix
cM=confusionMatrix(as.character(y), as.character(lda.pred$class), negative="0")
TPR = cM[2]/(cM[2]+cM[4])
TNR = cM[3]/(cM[1]+cM[3])

# Area under the curve (AUC) of the ROC curve
roc.out=roc(y, lda.pred$posterior[,1])
plot(roc.out)
roc.out$auc


# prediction rule
svm.out=svm(x=x,y=y)
svm.pred=predict(svm.out,newdata=x,decision.values=TRUE)
svm.pred.dv=attr(svm.pred, "decision.values")
svm.pred=predict(svm.out,newdata=x)

# evaluation based on confusion matrix
```

```
cM=confusionMatrix(as.character(y), as.character(svm.pred), negative="0")
TPR = cM[2]/(cM[2]+cM[4])
TPR
TNR = cM[3]/(cM[1]+cM[3])
TNR

# Area under the curve (AUC) of the ROC curve
roc.out=roc(y, as.vector(svm.pred.dv))
plot(roc.out)
roc.out$auc

# # # random forest to build a prediction rule

# prediction rule
rf.out = randomForest(y=y, x=x)
rf.pred = predict(rf.out, newdata=x, type="response")
rf.pred.prob = predict(rf.out, newdata=x, type="prob")

# evaluation based on confusion matrix and sensitivity and specificity
cM=confusionMatrix(as.character(y), as.character(rf.pred), negative="0")
TPR = cM[2]/(cM[2]+cM[4])
TPR
TNR = cM[3]/(cM[1]+cM[3])
TNR

# area under the curve and ROC plot
roc.out=roc(y, as.vector(rf.pred.prob[,2]))
plot(roc.out)
roc.out$auc
```

II - Evaluating prediction performance using cross-validation

```
# prediction function using crossvalidation
predfun.da = function(Xtrain, Ytrain, Xtest, Ytest, diagonal=FALSE)
{
  # estimate ranking and determine the best numVars variables
  ranking.DA = sda.ranking(Xtrain, Ytrain, verbose=FALSE, diagonal=diagonal, fdr=TRUE)
  numVars = sum(ranking.DA[,"lfdr"]<0.2)
  selVars = ranking.DA[,"idx"][1:numVars]
  # fit and predict
  sda.out = sda(Xtrain[, selVars, drop=FALSE], Ytrain, diagonal=diagonal, verbose=FALSE)
  sda.class = predict(sda.out, Xtest[, selVars, drop=FALSE], verbose=FALSE)$class
  # count false and true positives/negatives
  negative = levels(Ytrain)[1] # negatives or baseline is the good response class
  cm = confusionMatrix(Ytest, sda.class, negative=negative)
  return(cm)
}
set.seed(2)

# prediction function support vector machines
predfun.svm = function(Xtrain, Ytrain, Xtest, Ytest)
{
  # fit
```

```
  svm.out=svm(x=Xtrain,y=Ytrain)
  # predict
  svm.pred=predict(svm.out,newdata=Xtest)
  # count false and true positives/negatives
  negative = levels(Ytest)[1] # negatives are the good response class
  cm = confusionMatrix(Ytest, svm.pred, negative=negative)
  return(cm)
}

# prediction function random forests
predfun.rf = function(Xtrain, Ytrain, Xtest, Ytest)
{
  # fit
  rf.out = randomForest(y=Ytrain, x=Xtrain)
  # predict
  rf.pred = predict(rf.out, newdata=Xtest, type="response")
  # count false and true positives/negatives
  negative = levels(Ytest)[1] # negatives are the good response class
  cm = confusionMatrix(Ytest, rf.pred, negative=negative)
  return(cm)
}


# fivefold crossvalidation with two repetitions to evaluate prediction performance
TPR = rep(0,4)
TNR = rep(0.4) # ,?

# dda
cv.dda = crossval(predfun.da, X=x, Y=y, K=5, B=2, diagonal=TRUE, verbose=FALSE)
cv.dda$stat
TPR[1] = cv.dda$stat[2]/(cv.dda$stat[2]+cv.dda$stat[4])
TNR[1] = cv.dda$stat[3]/(cv.dda$stat[1]+cv.dda$stat[3])

# lda
cv.lda = crossval(predfun.da, X=x, Y=y, K=5, B=2, diagonal=FALSE, verbose=FALSE)
cv.lda$stat
TPR[2] = cv.lda$stat[2]/(cv.lda$stat[2]+cv.lda$stat[4])
TNR[2] = cv.lda$stat[3]/(cv.lda$stat[1]+cv.lda$stat[3])

# svm
cv.svm = crossval(predfun.svm, X=x, Y=y, K=5, B=2, verbose=FALSE)
cv.svm$stat
TPR[3] = cv.svm$stat[2]/(cv.svm$stat[2]+cv.svm$stat[4])
TNR[3] = cv.svm$stat[3]/(cv.svm$stat[1]+cv.svm$stat[3])

# randomForest
cv.rf = crossval(predfun.rf, X=x, Y=y, K=5, B=2, verbose=FALSE)
cv.rf$stat
TPR[4] = cv.rf$stat[2]/(cv.rf$stat[2]+cv.rf$stat[4])
TNR[4] = cv.rf$stat[3]/(cv.rf$stat[1]+cv.rf$stat[3])

# final results
# true positive rate (sensitivity)
```

```
TPR
# true negative rate (specificity)
TNR
```

Part V

I - Non-parametric prediction using random forests

```
rf.out = randomForest(x=x, y=y, ntree=100, importance=TRUE)
#importance=TRUE assesses which variables are important


# rank variables according to their importance and plot top 10
importance=rf.out$importance
importance[order(importance[,2], decreasing =TRUE),] [1:10,]
varImpPlot(rf.out)

# bagging with randomForest function
# mtry= how many predictors of x should be considered to split the tree, which is a bagging approach
bagging.out = randomForest(x=x, y=y, ntree=100, importance =TRUE, mtry=ncol(x))

# # prediction function for random forest
predfun.rf = function(train.x, train.y, test.x, test.y, ntree){
  # fit the model and build a prediction rule
  rf.fit = randomForest(x=train.x, y=train.y, ntree=ntree)
  # predict the new observation based on the test data and the prediction rule
  ynew = predict(rf.fit , newdata=test.x)
  # compute squared error risk (MSE)
  out = mean( (ynew - test.y)^2 )
  return( out )
}
# ntree as open paramter: compare performance for different training length of rf algorithm

# compare prediction performance ntree=10, ntree=100
library(crossval)
set.seed(14)
# 10 trees
cv.out.rf10 = crossval(predfun.rf, x, y, ntree=10, K=5, B=10, verbose=FALSE)
cv.out.rf10$stat
cv.out.rg10$stat.se

# 100trees
cv.out.rf100=crossval(predfun.rf, x, y, ntree=100, K=5, B=10, verbose=FALSE)
cv.out.rf100$stat
cv.out.rf$stat.se

# compare random forest with regularised regression prediciton performance

library(glmnet)
predfun.glmnet = function(train.x, train.y, test.x, test.y, lambda = lambda, alpha=alpha){
  # fit glmnet prediction rule
  glmnet.fit = glmnet(x=train.x, y=train.y, lambda = lambda, alpha=alpha)
  # predict the new observation based on the test data and the prediction rule
```

```
  ynew = predict(glmnet.fit , newx=test.x)
  # compute squared error risk (MSE)
  out = mean( (ynew - test.y)^2 )
  return( out )
}

# crossvalidation to tune the regularisation parameter
library(foreach)
set.seed(12)
lasso.cv = cv.glmnet(x,y,family="gaussian",alpha=1, type.measure="mse")
set.seed(123)
ridge.cv = cv.glmnet(x,y,family="gaussian",alpha=0, type.measure="mse")
set.seed(1234)
a = seq(0.05, 0.95, 0.05)
search = foreach(i = a, .combine = rbind)%do%{
  cv = cv.glmnet(x,y,family = "gaussian", type.measure = "mse", alpha = i)
  data.frame(cvm = cv$cvm[cv$lambda == cv$lambda.1se],
             lambda.1se = cv$lambda.1se, alpha = i)
}
elasticnet.cv = search[search$cvm == min(search$cvm), ]

# compare if random forest is better than lasso and elastic net
set.seed(15)
cv.out.ridge = crossval(predfun.glmnet, x, y, lambda=ridge.cv$lambda.1se,
                        alpha=0, K=5, B=10, verbose=FALSE)
cv.out.lasso = crossval(predfun.glmnet, x, y, lambda=lasso.cv$lambda.1se,
                        alpha=1, K=5, B=10, verbose=FALSE)
cv.out.enet = crossval(predfun.glmnet, x, y, lambda = elasticnet.cv$lambda.1se,
                       alpha = elasticnet.cv$alpha, K=5, B=10, verbose=FALSE)
table.out = rbind(c(cv.out.ridge$stat, cv.out.ridge$stat.se), c(cv.out.lasso$stat, cv.out.lasso$stat.se)
                  c(cv.out.enet$stat, cv.out.enet$stat.se), c(cv.out.rf10$stat, cv.out.rf10$stat.se),
                  c(cv.out.rf100$stat, rownames(table.out) = c("ridge", "lasso", "elastic net", "rf 10"
colnames(table.out)=c("mse", "se")))

# predict age of new mice using lasso model based on nitrogendioxide exposure

# building the prediction rule
lasso.cv.out = glmnet(x,y,family="gaussian",alpha=1,lambda=lasso.cv$lambda.1se)
# evaluating on the new data
lasso.hat = predict(lasso.cv.out,newx=x.test, s=lasso.cv$lambda.1se)

# predict age of new mice using ridge model
# building the prediction rule
ridge.cv.out = glmnet(x,y,family="gaussian",alpha=0,lambda=ridge.cv$lambda.1se)
# evaluating on the new data
ridge.hat = predict(ridge.cv.out,newx=x.test, s=ridge.cv$lambda.1se)

# predict age of new mice using elastic net
# building the prediction rule
enet.out = glmnet(x, y, family = "gaussian",
                  lambda = elasticnet.cv$lambda.1se, alpha = elasticnet.cv$alpha)
# evaluating on the new data
enet.hat = predict(enet.out,newx=x.test,
```

```
                    lambda = elasticnet.cv$lambda.1se, alpha = elasticnet.cv$alpha)

# predict age of new mice using random forest model with ntree=500 for a reliable prediction
# building the prediction rule
rf.out = randomForest(x=x, y=y, importance =TRUE, ntree=500)
# evaluating on the new data
rf.hat = predict(rf.out,newdata=x.test)

# hypothesis testing that NO2 exposure reduces the biological age of mice: t-test
# random forest
t.test(rf.hat~as.factor(exposed))

# ridge
t.test(ridge.hat~as.factor(exposed))


# # fit decision tree to a predictor matrix with y.input, x.input
# tree package
tree.out = tree(y.input ~ x.input)
plot(tree.out)
text(tree.out)

# prune tree using cross validation (FUN=prune.misclass for misclassification as criterion)
set.seed(33)
cv.out = cv.tree(tree.out, FUN=prune.misclass)
cv.out

# continue with model with smallest $size (here: best=5)
pruned.tree = prune.tree(tree.out, best=5)
plot(pruned.tree)
text(pruned.tree)

# fit a random forest to the data
rf.out = randomForest(y=y.input, x=x.input)
varImpPlot(rf.out, main="")
```