A collection of algorithms for large data and regression analysis

Yasin Elmaci

Load core libraries

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

Section 1

1.1 - Linear analysis

1.1.1 - Linear regression

```
set.seed(1)
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
```

1.1.2 - OLS regression

```
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)
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
```

1.2 - General Linear Prediction

1.2.1 - Linear Prediction

```
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
```

1.2.2 - Logistic Prediction

```
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

## Section 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.

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

1.3 - Linear mixed model

```
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
```

Section 2

2.1 - 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
```

2.2 - Multiple testing adjustment

```r
library(qvalue)

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

2.2.1 - Bonferroni correction

```
# bonferroni correction with p-values below 0.05
# most conservative
bonferroni = p.adjust(p_vec, method="bonferroni")
table(bonferroni<0.05)
output[which(bonferroni<0.05),]
```

2.2.2 - Benjamini-Hochberg FDR correction

```
# 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),]
```

2.2.3 - q-value FDR correction

```
# 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
```

2.2.4 - Local FDR correction

```
# 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
```

Section 3

3.1 - Penalised regression

3.1.1 - Lasso regression

```
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
```

```r
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.*

3.1.2 - Ridge regression

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

ridge_cv = cv.glmnet(x,y,family="gaussian",alpha=0, 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
# that is within 1 standard error of the minimum mse

# lasso regression with lambda min

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

# lasso regression with lambda 1se

ridge_1se = glmnet(x,y,family="gaussian",alpha=0,lambda=ridge_cv$lambda.1se)
sum(abs(ridge_1se$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.
```

3.1.3 - Elastic-net regression

```
# 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_cv = search[search$cvm == min(search$cvm), ]

elasticnet_1se = glmnet(x, y, family = "gaussian",
                        lambda = elasticnet.cv$lambda.1se,
                        alpha = elasticnet.cv$alpha)

sum(abs(elasticnet_1se$beta)>0)
```

3.2 - Generalised Prediction rule (intra-model cross-validation)

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

```r
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_cvk =  crossval(prediction_rule, x, y, lambda=ridge_cv$lambda.1se, alpha=0,
                         K=5, B=20, verbose=FALSE)
lasso_cvk =  crossval(prediction_rule, x, y, lambda=lasso_cv$lambda.1se, alpha=1,
                         K=5, B=20, verbose=FALSE)
elasticnet_cvk  =  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
```

3.3 - Shrinkage t-score

```r
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
```

```r
# 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
```

Section 4

4.1 - Predicting treatment (classification)

4.1.1 - Diagonal discriminant analysis

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

library(sda)
library(pROC)
library(PRROC)

y = as.factor(data_bc$rcb)
table(y) # 0 = excellent response, 1 = lesser response
x = as.matrix(data_bc$x)
dim(x)

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

DDA_variables = sum(DDA_ranking[,"lfdr"]<0.2)
# factors passing <0.2 threshold and IDs

selected_variables = DDA_ranking[,"idx"][1:DDA_variables] # gene indexing

# build prediction rule for dda

DDA_rule = sda(x[, selected_variables, drop = FALSE], y, diagonal = TRUE)

DDA_predicted = predict(DDA_rule, x[, selected_variables, drop = FALSE],
                        verbose = FALSE)

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

cM = confusionMatrix(as.character(y),
                     as.character(DDA_predicted$class),
                     negative="0")
# FP, TP, TN, FN

confusion_dataframe = data.frame(c(cM[[2]],cM[[4]]),c(cM[[1]],cM[[3]]))
colnames(confusion_dataframe)=c("Positive", "Negative")
row.names(confusion_dataframe)=c("Positive", "Negative")
knitr::kable(confusion_dataframe, "pipe")
```

```
TPR = cM[2]/(cM[2]+cM[4]) # true positive rate = TP/TP+FN
TNR = cM[3]/(cM[1]+cM[3]) # true negative rate = TN/FP+TN


head(DDA_predicted$posterior) # case/ no case
ROC = roc(y, DDA_predicted$posterior[,2])
plot(ROC) # ROC
ROC$auc # AUC
# compute the area under the curve (AUC)
# of the receiver operating characteristic (ROC)
```

4.1.2 - Linear discriminant analysis

```
# linear discriminant analysis

LDA_ranking = sda.ranking(x, y, diagonal = FALSE) # ranking features using sda function

LDA_variables = sum(LDA_ranking[,"lfdr"]<0.2)
# factors passing <0.2 threshold and IDs

selected_variables = LDA_ranking[,"idx"][1:LDA_variables] # gene indexing

# build prediction rule for dda

LDA_rule = sda(x[, selected_variables, drop = FALSE], y, diagonal = FALSE)

LDA_predicted = predict(LDA_rule, x[, selected_variables, drop = FALSE],
                        verbose = FALSE)

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

cM = confusionMatrix(as.character(y),
                   as.character(LDA_predicted$class),
                   negative="0")
# FP, TP, TN, FN

confusion_dataframe = data.frame(c(cM[[2]],cM[[4]]),c(cM[[1]],cM[[3]]))
colnames(confusion_dataframe)=c("Positive", "Negative")
row.names(confusion_dataframe)=c("Positive", "Negative")
knitr::kable(confusion_dataframe, "pipe")


TPR = cM[2]/(cM[2]+cM[4]) # true positive rate = TP/TP+FN
TNR = cM[3]/(cM[1]+cM[3]) # true negative rate = TN/FP+TN


head(LDA_predicted$posterior) # case/ no case
ROC = roc(y, LDA_predicted$posterior[,2])
plot(ROC) # ROC
ROC$auc # AUC
# compute the area under the curve (AUC)
# of the receiver operating characteristic (ROC)
```

4.1.3 - Support vector machine regression

```
# support vector machine

SVM_rule = svm(x,y) # build svm train rule

SVM_predicted = predict(SVM_rule, x, decision.values = FALSE) # build prediction rule

SVM_predicted_DV = attr(SVM_predicted, "decision.values")

SVM_predicted = predict(SVM_rule, x)

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

cM = confusionMatrix(as.character(y),
                    as.character(SVM_predicted),
                    negative="0")
# FP, TP, TN, FN

confusion_dataframe = data.frame(c(cM[[2]],cM[[4]]),c(cM[[1]],cM[[3]]))
colnames(confusion_dataframe)=c("Positive", "Negative")
row.names(confusion_dataframe)=c("Positive", "Negative")
knitr::kable(confusion_dataframe, "pipe")


TPR = cM[2]/(cM[2]+cM[4]) # true positive rate = TP/TP+FN
TNR = cM[3]/(cM[1]+cM[3]) # true negative rate = TN/FP+TN


head(SVM_predicted$posterior) # case/ no case
ROC = roc(y, SVM_predicted$posterior[,2])
plot(ROC) # ROC
ROC$auc # AUC
# compute the area under the curve (AUC)
# of the receiver operating characteristic (ROC)
```

4.1.4 - Random forest regression

```
# random forest

random_forest = randomForest(y=y, x=x)
random_forest_predicted = predict(random_forest, x, type="response")
random_forest_predicted_probability = predict(random_forest, x, type="prob")
cM=confusionMatrix(as.character(y),
                    as.character(random_forest_predicted),
                    negative="0")
?confusionMatrix

TPR = cM[2]/(cM[2]+cM[4])

TNR = cM[3]/(cM[1]+cM[3])
```

```r
roc.out=roc(y, as.vector(rf.pred.prob[,2]))
plot(roc.out) # ROC
roc.out$auc # AUC
# compute the area under the curve (AUC)
# of the receiver operating characteristic (ROC)
```

CAUTION: It is NOT good practise to evaluate a prediction rule on the same dataset that was used to establish the prediction rule, results may be misleading due to overfitting.

4.2 - Evaluating prediction performance using cross-validation

```r
prediction_function = function(Xtrain, Ytrain, Xtest, Ytest, method)
{

  if (method=="dda" || method =="lda") {

    if (method=="dda") { # diagonal discriminant analysis
      diagonal=TRUE

    }
    else if (method=="lda") { # linear discriminant analysis
      diagonal==FALSE

    }

    DA_ranking = sda.ranking(Xtrain, Ytrain,
                             verbose=FALSE, diagonal=diagonal, fdr=TRUE)
    ranked_variables = sum(DA_ranking[,"lfdr"]<0.2)
    selected_variables = DA_ranking[,"idx"][1:ranked_variables]

    # fit and predict
    DA_fit = sda(Xtrain[, selected_variables, drop=FALSE],
                 Ytrain, diagonal=diagonal, verbose=FALSE)
    DA_predict = predict(DA_fit, Xtest[, selected_variables, 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, DA_predict, negative=negative)

  }

  else if (method=="svm"){
    # fit
    SVM_fit=svm(Xtrain, Ytrain)

    #predict
    SVM_predict=predict(SVM_fit, Xtest)

    # count false and true positives/negatives
    negative = levels(Ytest)[1] # negatives are the good response class
    cm = confusionMatrix(Ytest, SVM_predict, negative=negative)
  }
  else if (method=="randomforest"){
    # fit
    random_forest_fit = randomForest(y=Ytrain, x=Xtrain)

    #predict
    random_forest_predict = predict(random_forest_fit, Xtest, type="response")

    # count false and true positives/negatives
    negative = levels(Ytest)[1] # negatives are the good response class
    cm = confusionMatrix(Ytest, random_forest_predict, negative=negative)
  }
  else{
    print("Please provide appropriate method, 'dda', 'lda', 'svm', 'randomforest' ")

  }

  return(cm)
}

# k-fold crossvalidation with B repetitions to evaluate prediction performance
TPR = rep(0,4)
TNR = rep(0,4)
K = 5
B = 20
```

4.2.1 - Evaluating prediction performance of DDA cross-validation

```
# dda
DDA_cv = crossval(prediction_function, X = x, Y = y, K, B,
                  method = "dda", verbose = FALSE)
DDA_cv$stat
TPR[1] = DDA_cv$stat[2]/(DDA_cv$stat[2]+DDA_cv$stat[4])
TNR[1] = DDA_cv$stat[3]/(DDA_cv$stat[1]+DDA_cv$stat[3])
```

4.2.2 - Evaluating prediction performance of LDA cross-validation

```
# lda
LDA_cv = crossval(prediction_function, X = x, Y = y, K, B,
                  method = "lda", verbose = FALSE)
LDA_cv$stat
TPR[2] = LDA_cv$stat[2]/(LDA_cv$stat[2]+LDA_cv$stat[4])
TNR[2] = LDA_cv$stat[3]/(LDA_cv$stat[1]+LDA_cv$stat[3])
```

4.2.3 - Evaluating prediction performance of SVM cross-validation

```
# svm
SVM_cv = crossval(prediction_function, X = x, Y = y, K, B,
                  method = "svm", verbose = FALSE)
SVM_cv$stat
TPR[3] = SVM_cv$stat[2]/(SVM_cv$stat[2]+SVM_cv$stat[4])
TNR[3] = SVM_cv$stat[3]/(SVM_cv$stat[1]+SVM_cv$stat[3])
```

4.2.4 - Evaluating prediction performance of random forest cross-validation

```r
# randomForest
randomforest_cv = crossval(prediction_function, X = x, Y = y, K, B,
                           method = "randomforest", verbose = FALSE)
randomforest_cv$stat
TPR[4] = randomforest_cv$stat[2]/(randomforest_cv$stat[2]+randomforest_cv$stat[4])
TNR[4] = randomforest_cv$stat[3]/(randomforest_cv$stat[1]+randomforest_cv$stat[3])

# final results
# true positive rate (sensitivity)
TPR
# true negative rate (specificity)
TNR
```

Section 5

5.1 - Non-parametric prediction using random forests

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

y = control_mice$y_control
x = control_mice$x_control
dim(x)

# build random forest with 100 trees

random_forest = randomForest(x = x, y = y, ntree = 100, importance = TRUE)

importance = random_forest$importance # extract random forest importance

importance_order = importance[order(importance[,2], decreasing = TRUE),] [1:10,]

#  decreasing = TRUE - rank the top 10 most important values

varImpPlot(random_forest) # visualise importance from random forest

# bagging random forest
bagging = randomForest(x, y, ntree = 100, importance = TRUE, mtry = ncol(x))

# mtry = ncol(x) - predictors to split tree, ncol(x) indicates baggin

# function to compare performance for different training lengths of random forest

random_forest_prediction = function(train.x, train.y, test.x, test.y, ntree){

    #fit the model and build a prediction rule
    random_forest_fit = randomForest(train.x, train.y, ntree = ntree)
    #predict the new observation based on the test data and the prediction rule
    random_forest_predicted = predict(random_forest_fit , test.x)
    # compute squared error risk (MSE)
    out = mean( (random_forest_predicted - test.y)^2 )
    return( out )
}

# comparing prediction performance

# k-fold crossvalidation with B repetitions to evaluate prediction performance

K = 5
B = 10 # ideally 100-1000 repetitions

randomforest_cv_nt10 = crossval(random_forest_prediction, x, y, ntree = 10,
                                K, B, verbose = FALSE)

randomforest_cv_nt10$stat
randomforest_cv_nt10$stat.se

randomforest_cv_nt100  =  crossval(random_forest_prediction, x, y, ntree=100,
                                   K=5, B=10, verbose=FALSE)
randomforest_cv_nt100$stat
randomforest_cv_nt100$stat.se

# It is recommended to use at least 500 or more trees.

# Random forest vs regularised regression - prediction performance

table = rbind(c(ridge_cvk$stat, ridge_cvk$stat.se),
                  c(lasso_cvk$stat, lasso_cvk$stat.se),
                  c(elasticnet_cvk$stat, elasticnet_cvk$stat.se),
                  c(randomforest_cv_nt10$stat, randomforest_cv_nt10$stat.se),
                  c(randomforest_cv_nt100$stat, randomforest_cv_nt100$stat.se))

rownames(table) = c("ridge",
                        "lasso",
                        "elastic net",
                        "randomforest_cv_nt10",
                        "randomforest_cv_nt100")

colnames(table) = c("mse", "se")
table
```

5.2 - Non-parametric prediction, multiple approaches

5.3 - Decision trees and random forests

```
titanic = read.csv("E:/Imperial College London/Term 2/AR/Week5/titanic.csv")
library(tree)

dim(titanic)
table(titanic$survived)

x = cbind(titanic$pclass,
          titanic$sex,
          titanic$age,
          titanic$sibsp,
          titanic$parch)

removed = which(is.na(titanic$age)==TRUE)
x = x[-removed,]

hist(as.numeric(x[,3]))

colnames(x) = c("pclass", "sex", "age", "sibsp", "parch")
y = as.factor(titanic$survived[-removed])
table(y)

tree = tree(y ~ x)
plot(tree)
text(tree)
```

Decision trees are prone to over-fitting. This can be prevented by either cross-validation or performing a random forest approach.

```
tree_cv = cv.tree(tree, FUN=prune.misclass)  #cross-validation


tree_pruned = prune.tree(tree, best=5) # pruning tree from best classification
plot(tree_pruned)
text(tree_pruned)
```

Section 6

6.1 - Parameter tuning and GLMNET

```
load("E:/Imperial College London/Term 2/AR/Week5/heart.Rdata")

x = heart.data$x
y = as.factor(heart.data$y)
table(y)
head(x)
summary(x)

head(x)

x.mat = model.matrix( ~ ., x)[,-1] # transform data into matrix
x.mat = as.matrix(x.mat)
head(x.mat, n=3) # summarise data


# cross-validation to tune the regularisation parameter
lasso_cv = cv.glmnet(x.mat, y, family = "binomial", alpha = 1,
                     type.measure = "deviance")
```

For categorical or binary variables the mean squared error is not a good measure for model fit.

```
# lasso model fit

lasso_min = glmnet(x.mat, y, family = "binomial",
                   alpha=1, lambda = lasso_cv$lambda.min)
```

```r
lasso_1se = glmnet(x.mat, y, family = "binomial",
                   alpha=1, lambda = lasso_cv$lambda.1se)

sum(abs(lasso_min$beta)>0)
sum(abs(lasso_1se$beta)>0)


# unspecified lasso

lasso_all = glmnet(x.mat, y, family="binomial" ,alpha=1)
plot(lasso_all, xvar = 'lambda', label=TRUE)
```

6.2 - Prediction performance of regularised regression

```r
# GLM non-regularised

GLM = glm(y ~ x.mat, family = "binomial")
eta = predict.glm(GLM, as.data.frame(x.mat)) # needed for inverse logit transform
GLM_probability =  (exp(eta))/(exp(eta)+1)
par(mfrow=c(1,2))
hist(eta)
hist(GLM_probability)

# lasso regularised regression

lasso_regression = glmnet(x.mat, y,  family = "binomial", alpha = 0.1, lambda = 0.1)
eta = predict(lasso_regression, x.mat) # needed for inverse logit transform
lasso_probability =  (exp(eta))/(exp(eta)+1)
par(mfrow=c(1,2))
hist(eta)
hist(lasso_probability)


# cross-validation to tune regularisation parameter for lasso and ridge
# to minimise the deviance
lasso_cv_dev = cv.glmnet(x.mat, y, family="binomial", alpha=1,
                     type.measure="deviance")
ridge_cv_dev = cv.glmnet(x.mat, y, family="binomial", alpha=0,
                     type.measure="deviance")


prediction_rule = function(train.x, train.y, test.x, test.y,
 lambda = lambda, alpha=alpha, threshold = 0.5){

    #fit glmnet prediction rule
    glmnet.fit = glmnet(x=train.x, y=train.y,family="binomial", lambda = lambda, alpha=alpha)
    #predict the new observation based on the test data and the prediction rule
    eta = predict(glmnet.fit , newx=test.x)
    p.glmnet = (exp(eta)/(exp(eta)+1))
    y.new = as.factor(ifelse(p.glmnet < threshold, 0,1))
    # count false and true positives/negatives
    negative = levels(test.y)[1]
    cm = crossval::confusionMatrix(test.y, y.new, negative=negative)
    #return confusion matrix
    return(cm)

}

TPR = rep(0,3)
TNR = rep(0.3)

#glm
GLM_cv = crossval(prediction_rule, X=x.mat, Y=y, K=10, B=20,
 lambda=0, alpha=0, verbose=FALSE)

GLM_cv$stat
TPR[1] = GLM_cv$stat[2]/(GLM_cv$stat[2]+GLM_cv$stat[4])
TNR[1] = GLM_cv$stat[3]/(GLM_cv$stat[1]+GLM_cv$stat[3])
diagnosticErrors(GLM_cv$stat)

#ridge
ridge_cv = crossval(prediction_rule, X=x.mat, Y=y, K=10, B=20,
 lambda=ridge_cv_dev$lambda.1se, alpha= 0, verbose=FALSE)
ridge_cv$stat
TPR[2] = ridge_cv$stat[2]/(ridge_cv$stat[2]+ridge_cv$stat[4])
TNR[2] = ridge_cv$stat[3]/(ridge_cv$stat[1]+ridge_cv$stat[3])
diagnosticErrors(ridge_cv$stat)

#lasso
lasso_cv = crossval(prediction_rule, X=x.mat, Y=y, K=10, B=20,
 lambda=lasso_cv_dev$lambda.1se, alpha= 1, verbose=FALSE)
lasso_cv$stat
TPR[3] = lasso_cv$stat[2]/(lasso_cv$stat[2]+lasso_cv$stat[4])
TNR[3] = lasso_cv$stat[3]/(lasso_cv$stat[1]+lasso_cv$stat[3])

diagnosticErrors(lasso_cv$stat)

# final comparison
TPR
TNR
```