ML01-project

# Breast Cancer data analysis

## Introduction

The phenotypes for characterisation are:

Sample ID (code number)  
Clump thickness  
Uniformity of cell size  
Uniformity of cell shape  
Marginal adhesion  
Single epithelial cell size  
Number of bare nuclei  
Bland chromatin  
Number of normal nuclei  
Mitosis  
Classes, i.e. diagnosis

## clean the data

bc\_data <- read.table("breast-cancer-wisconsin.data", header = FALSE, sep = ",")  
head(bc\_data)

## V1 V2 V3 V4 V5 V6 V7 V8 V9 V10 V11  
## 1 1000025 5 1 1 1 2 1 3 1 1 2  
## 2 1002945 5 4 4 5 7 10 3 2 1 2  
## 3 1015425 3 1 1 1 2 2 3 1 1 2  
## 4 1016277 6 8 8 1 3 4 3 7 1 2  
## 5 1017023 4 1 1 3 2 1 3 1 1 2  
## 6 1017122 8 10 10 8 7 10 9 7 1 4

As we can see, the dataset has no header, so we add the collum names in order to have an easier manipulation of the data.

colnames(bc\_data) <- c("sample\_code\_number", "clump\_thickness", "uniformity\_of\_cell\_size", "uniformity\_of\_cell\_shape", "marginal\_adhesion", "single\_epithelial\_cell\_size", "bare\_nuclei", "bland\_chromatin", "normal\_nucleoli", "mitosis", "classes")

We also want to change the name for the response variable

bc\_data$classes.num=bc\_data$classes  
bc\_data$classes.num[bc\_data$classes.num=="2"] <- 0  
bc\_data$classes.num[bc\_data$classes.num=="4"] <- 1  
bc\_data$classes[bc\_data$classes=="2"] <- "benign"  
bc\_data$classes[bc\_data$classes=="4"] <- "malignant"  
head(bc\_data)

## sample\_code\_number clump\_thickness uniformity\_of\_cell\_size  
## 1 1000025 5 1  
## 2 1002945 5 4  
## 3 1015425 3 1  
## 4 1016277 6 8  
## 5 1017023 4 1  
## 6 1017122 8 10  
## uniformity\_of\_cell\_shape marginal\_adhesion single\_epithelial\_cell\_size  
## 1 1 1 2  
## 2 4 5 7  
## 3 1 1 2  
## 4 8 1 3  
## 5 1 3 2  
## 6 10 8 7  
## bare\_nuclei bland\_chromatin normal\_nucleoli mitosis classes  
## 1 1 3 1 1 benign  
## 2 10 3 2 1 benign  
## 3 2 3 1 1 benign  
## 4 4 3 7 1 benign  
## 5 1 3 1 1 benign  
## 6 10 9 7 1 malignant  
## classes.num  
## 1 0  
## 2 0  
## 3 0  
## 4 0  
## 5 0  
## 6 1

bc\_data[bc\_data=="?"]<-NA  
#length(bc\_data[bc\_data$bare\_nuclei==NA])  
nrow(bc\_data)-length(complete.cases(bc\_data)[complete.cases(bc\_data)==TRUE])

## [1] 16

length(bc\_data$bare\_nuclei[is.na(bc\_data$bare\_nuclei)])

## [1] 16

We have 16 missing data that are all in the bc\_data$bare\_nuclei collunm. As the number of observation with missing data is low compared to the total number of observation, we could just ignore these 16 observations for the rest of the study and loose small amount of data. However, we could also replace the missing values by the mean of the collum. It does not change the global mean but reduces the variance. We can also try to apply some algorith that will guess the value with the MICE library. We could also use the library Amelia but we have to make the asomption that all variables follow a multivariate law.

bc\_data\_missing<-bc\_data  
  
bc\_data<-na.omit(bc\_data)  
bc\_data\_knn<-bc\_data#used in comparison  
  
bc\_data[,2:10] <- apply(bc\_data[, 2:10], 2, function(x) as.numeric(as.character(x)))  
#bc\_data$classes <- as.factor(bc\_data$classes)  
#bc\_data$classes <- as.numeric(bc\_data$classes)  
nrow(bc\_data)

## [1] 683

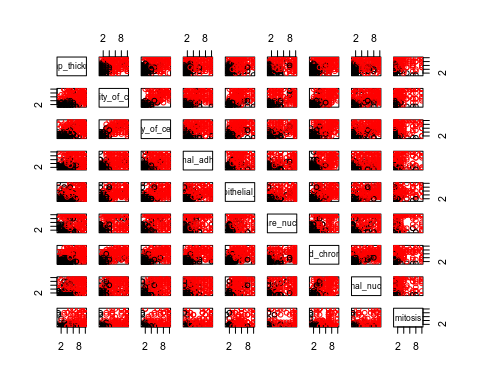
Now we have a clean dataset.

## Visualisation

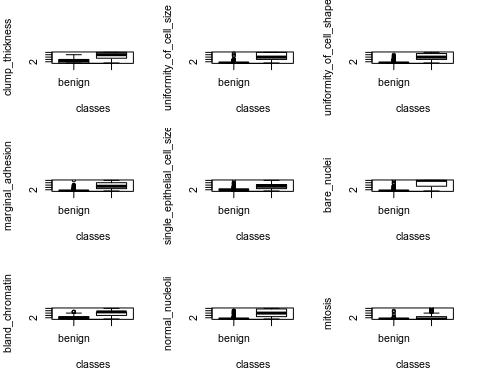
summary(bc\_data)

## sample\_code\_number clump\_thickness uniformity\_of\_cell\_size  
## Min. : 63375 Min. : 1.000 Min. : 1.000   
## 1st Qu.: 877617 1st Qu.: 2.000 1st Qu.: 1.000   
## Median : 1171795 Median : 4.000 Median : 1.000   
## Mean : 1076720 Mean : 4.442 Mean : 3.151   
## 3rd Qu.: 1238705 3rd Qu.: 6.000 3rd Qu.: 5.000   
## Max. :13454352 Max. :10.000 Max. :10.000   
## uniformity\_of\_cell\_shape marginal\_adhesion single\_epithelial\_cell\_size  
## Min. : 1.000 Min. : 1.00 Min. : 1.000   
## 1st Qu.: 1.000 1st Qu.: 1.00 1st Qu.: 2.000   
## Median : 1.000 Median : 1.00 Median : 2.000   
## Mean : 3.215 Mean : 2.83 Mean : 3.234   
## 3rd Qu.: 5.000 3rd Qu.: 4.00 3rd Qu.: 4.000   
## Max. :10.000 Max. :10.00 Max. :10.000   
## bare\_nuclei bland\_chromatin normal\_nucleoli mitosis   
## Min. : 1.000 Min. : 1.000 Min. : 1.00 Min. : 1.000   
## 1st Qu.: 1.000 1st Qu.: 2.000 1st Qu.: 1.00 1st Qu.: 1.000   
## Median : 1.000 Median : 3.000 Median : 1.00 Median : 1.000   
## Mean : 3.545 Mean : 3.445 Mean : 2.87 Mean : 1.603   
## 3rd Qu.: 6.000 3rd Qu.: 5.000 3rd Qu.: 4.00 3rd Qu.: 1.000   
## Max. :10.000 Max. :10.000 Max. :10.00 Max. :10.000   
## classes classes.num   
## Length:683 Min. :0.0000   
## Class :character 1st Qu.:0.0000   
## Mode :character Median :0.0000   
## Mean :0.3499   
## 3rd Qu.:1.0000   
## Max. :1.0000

plot(bc\_data[,2:10],col=(bc\_data$classes.num+1))

 We dont add the response variable into the scatter plot because box plots are better to display binnary variables

attach(mtcars)  
par(mfrow=c(3,3))  
for (c in c(2:10)){  
 boxplot(bc\_data[,c]~classes,data=bc\_data,xlab="classes",ylab=colnames(bc\_data)[c])  
}

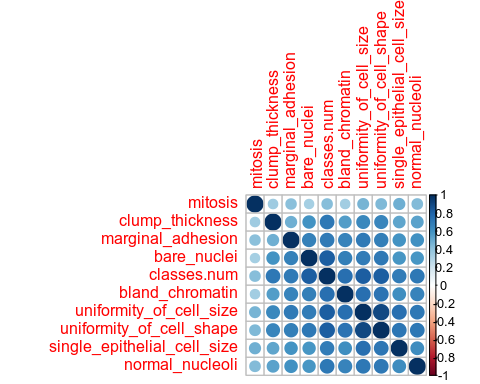


## Correlation

library(corrplot)

## corrplot 0.84 loaded

# calculate correlation matrix  
corMatMy <- cor(bc\_data[, c(-1,-11)])  
#corMatMy <- cor(bc\_data[, -1])  
corrplot(corMatMy, order = "hclust")



## PCA

### what is PCA?

## Classification

### why do we do classification?

Choose a classifier C(X) that assigns a class label from {benign; malignant} to a future unlabeled observation X. We want to assess the uncertainty in each classification we want to understand the roles of the different predictors.

### Create train subset

We divide the data into test and train dataset. We don’t need to use the function scale() because the data uses already the same range ( from 1 to 10)

n<-nrow(bc\_data)  
ntrain<-round(2\*n/3)  
ntest<-n-ntrain  
train<-sample(n,ntrain)  
bc\_data.train<-bc\_data[train,]  
bc\_data.test<-bc\_data[-train,]

### Bayes Classifier

This probability is called the Bayes error rate. It is the lowest errorprobability that can be achieved by a classifier. It characterizes thedifficulty of the classification task

### K nearest neighbor

#### facts

We can say that half of the information provided by the training set is contained in the nearest neighbor (asymptotically). Howerver, the KNN classifier breaks down for big dimention of datasets. We have 10 predictors and that is already a lot for KNN. #### implementation

library(FNN)

pred.bc\_data.knn<-knn(bc\_data.train[,2:10],bc\_data.test[,2:10],factor(bc\_data.train$classes),k=5)  
table(bc\_data.test$classes,pred.bc\_data.knn)#adjacent error matrix

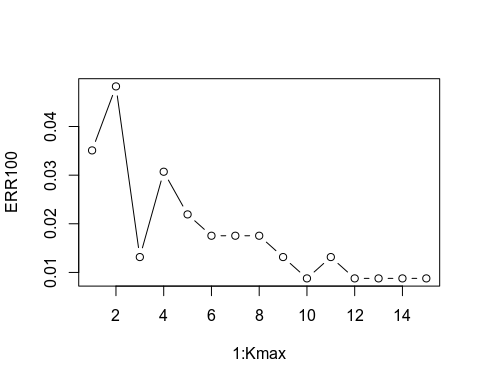
## pred.bc\_data.knn  
## benign malignant  
## benign 136 1  
## malignant 4 87

mean(bc\_data.test$classes!=pred.bc\_data.knn)

## [1] 0.02192982

#### find the optimal k

Kmax<-15  
ERR100<-c(0,Kmax)  
for(k in 1:Kmax){  
 classespred<-knn(bc\_data.train[,2:10],bc\_data.test[,2:10],factor(bc\_data.train$classes),k=k)  
 ERR100[k]<-mean(bc\_data.test$classes!=classespred)  
}  
plot(1:Kmax,ERR100,type = "b")

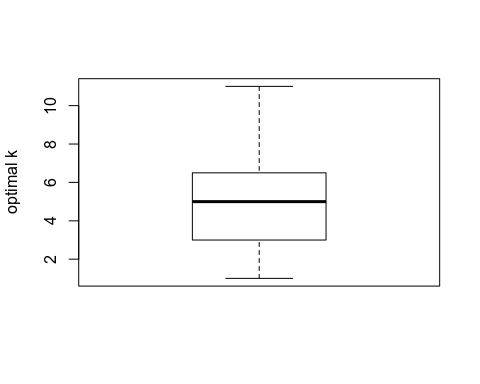


k\_best <- which.min(ERR100)  
k\_best

## [1] 10

#### box plot of optimal k

Kmax<-15  
max\_sim<-20  
K\_best<-c(0,max\_sim)  
for(s in 1:max\_sim){  
 ERR100<-c(0,Kmax)  
 train<-sample(n,ntrain)  
 bc\_data.train<-bc\_data[train,]  
 bc\_data.test<-bc\_data[-train,]  
 for(k in 1:Kmax){  
 classespred<-knn(bc\_data.train[,2:10],bc\_data.test[,2:10],factor(bc\_data.train$classes),k=k)  
 ERR100[k]<-mean(bc\_data.test$classes!=classespred)  
 }  
 k\_best[s] <- which.min(ERR100)  
}  
boxplot(k\_best,ylab="optimal k")



### Linear Discriminant Analysis (LDA)

#### Theory

The parameters to estimates are πk,̂ μk,and ̂Σ

LDA uses the full likelihood based on the joint distribution of X and Y (generative model).

LDA assumes that the data are *Gaussian*. More specifically, it assumes that all classes share the same covariance matrix.

LDA finds linear decision boundaries in a K−1 dimensional subspace. As such, it is not suited if there are higher-order interactions between the independent variables. Although LDA also has a number of parameters proportion top2, it isusually much more stable than QDA. This method is recommended when n is small.

#### Implementation

#bc\_data.train.scaled<-scale(bc\_data[train,c(-1,-11)])  
#bc\_data.test.scaled<-scale(bc\_data[-train,c(-1,-11)])

library(MASS)  
lda.bc\_data <- lda(classes~. ,data=bc\_data.train[,c(-1,-12)])  
pred.bc\_data.lda<-predict(lda.bc\_data,newdata=bc\_data.test[,c(-1,-12)])  
perf <-table(bc\_data.test$classes,pred.bc\_data.lda$class)  
print(perf)#confusion matrix

##   
## benign malignant  
## benign 138 1  
## malignant 9 80

1-sum(diag(perf))/nrow(bc\_data.test)#mean error rate

## [1] 0.04385965

#### Receiver Operating Characteristic (ROC)

In our case we have two classes therefore LDA assigns x to class 2 if xT̂Σ−1(̂μ2−̂μ1 ) >s,where the threshold *s* depends on the estimated prior probabibitie. If the prior probabilities cannot be estimated, or if the model assumption are not verified, a different threshold may give betterresult. The Receiver Operating Characteristic (ROC) curve describes theperformance of the classifier for any value of s. By changing the value of s we can change the confusion matrix values. This allow us to prioritize the true positive rate (sensitivity) and false positive rate (1-specificity).

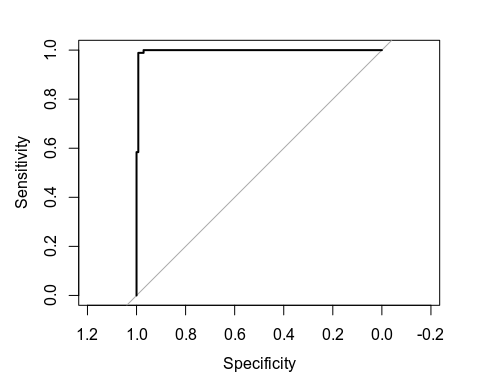
library(pROC)

## Type 'citation("pROC")' for a citation.

##   
## Attaching package: 'pROC'

## The following objects are masked from 'package:stats':  
##   
## cov, smooth, var

roc\_lda<-roc(bc\_data.test$classes,as.vector(pred.bc\_data.lda$x))  
plot(roc\_lda)

 ### Quadratic Discriminant Analysis (QDA) LDA don’t assumes that all classes share the same covariance matrix. So The parameters to estimates are πk,̂ μk,and ̂Σk. Mean between classes, mean inside classe and empirical variance matrix in class k.

#### Implementation

qda.bc\_data <- qda(classes~. ,data=bc\_data.train[,c(-1,-12)])  
pred.bc\_data.qda<-predict(qda.bc\_data,newdata=bc\_data.test[,c(-1,-12)])  
perf <-table(bc\_data.test$classes,pred.bc\_data.qda$class)  
print(perf)#confusion matrix

##   
## benign malignant  
## benign 135 4  
## malignant 1 88

1-sum(diag(perf))/nrow(bc\_data.test)#mean error rate

## [1] 0.02192982

### Naive Bayes

#### Theory

For Naive Bayes classifers, we set the covariance matrix to diagonal matrix. This assumption means that the predictors are conditionally independent given the class variable Y. A further simplification is achieved by assuming that the covariancematrices are diagonal and equal:Σ1=···=Σc=Σ=diag(σ21,…,σ2p). Naive Bayes classifiers have a number of parameters proportional to p.They usually outperform other methods when p is very large. #### Implementation

library(naivebayes)  
nb.bc\_data <- naive\_bayes(as.factor(classes)~. ,data=bc\_data.train[,c(-1,-12)])  
pred.bc\_data.nb<-predict(nb.bc\_data,newdata=bc\_data.test[,c(-1,-12)],type="class")  
pred.bc\_data.nb.prob<-predict(nb.bc\_data,newdata=bc\_data.test[,c(-1,-12)],type="prob")  
perf <-table(bc\_data.test$classes,pred.bc\_data.nb)  
print(perf)#confusion matrix

## pred.bc\_data.nb  
## benign malignant  
## benign 137 2  
## malignant 2 87

1-sum(diag(perf))/nrow(bc\_data.test)#mean error rate

## [1] 0.01754386

### Logistic regression

Logistic regression uses the conditional likelihood based on the conditional probabilities Pk(x) (discriminative model). logReg models fit by maximizing the conditional likelihood, which is the likelihood function, assuming thexiare fixed. unction`(β) =logL(β)is maximized using an iterative optimizationalgorithm: the Newton-Raphson algorithm. #### Binomial logistic regression As we have only two classes to predict, we can use this classifier

glm.bc\_data <- glm(as.factor(classes)~. ,data=bc\_data.train[,c(-1,-12)],family=binomial)  
pred.bc\_data.glm<-predict(glm.bc\_data,newdata=bc\_data.test[,c(-1,-12)],type="response")  
perf <-table(bc\_data.test$classes,pred.bc\_data.glm>0.5)  
print(perf)#confusion matrix

##   
## FALSE TRUE  
## benign 138 1  
## malignant 6 83

1-sum(diag(perf))/nrow(bc\_data.test)#mean error rate

## [1] 0.03070175

### Tree classifier

#### Impurity measures Qt

Unlike the tree for the regression, we will not use Qt= MSE within the region. We have three other options: Misclassification error, Gini Index, Entropy. In our casse we have two classes: if p is the proportion of malign in the region Rt then: Misclassification error = 1−max(p,1−p), Gini Index = 2p(1−p), Entropy = −plogp−(1−p)log(1−p). All three are similar, but entropy and the Gini index are differentiable,and hence more amenable to numerical optimization.

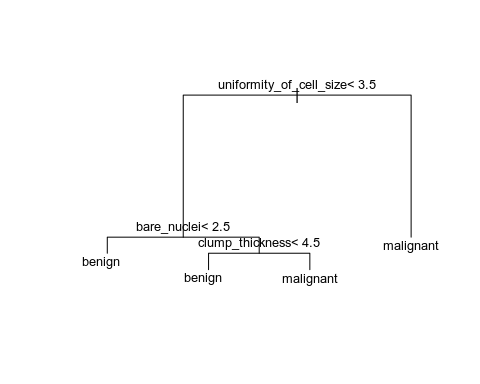
Consider a nodetwith sizentwith impurit yQt. For some variable j and split point s, we split t in two nodes, tL and tR, with sizes ntL and ntR, and with impurities QtL and QtR. The average decrease of impurity is ∆(j,s) =Qt−(ntLntQtL+ntRntQtR) If Qt is the entropy, then ∆(j,s)is interpreted as an information gain.We select at each step the splitting variablejand the split pointsthatmaximizes∆(j,s)or, equivalently, that minimizes the average impurity.

When splitting a predictor having q possible unordered values, thereare 2q−1−1 possible partitions of the q values into two groups. All the dichotomies can be explored for small q, but the computations become prohibitive for large q. In the 2-class case, this computation simplifies. We order the predictorlevels according to the proportion falling in outcome class 1. Then wesplit this predictor as if it were an ordered predictor. One can show this gives the optimal split, in terms of entropy or Gini index, among all possible 2q−1−1 splits.

Trees can easily handle qualitative predictors without the need tocreate dummy variables. But the algorith of partitionning tends to favor predictor with many levels: they should be avoided. In our case this is not a problem because every predicto has the same number of level.

#### Implementation

library(rpart)  
tree.bc\_data<-rpart(classes~.,data=bc\_data.train[2:11],method="class",control=rpart.control(xval = 10,minbucket = 5,cp=0))  
#x val is number of cross-validations  
#minbucket is the minimum number of observations in any terminal <leaf> node.  
# cp is complexity parameter. Any split that does not decrease the overall lack of fit by a factor of cp is not attempted.  
plot(tree.bc\_data,margin = 0.05)  
text(tree.bc\_data,pretty=0,cex=0.8)



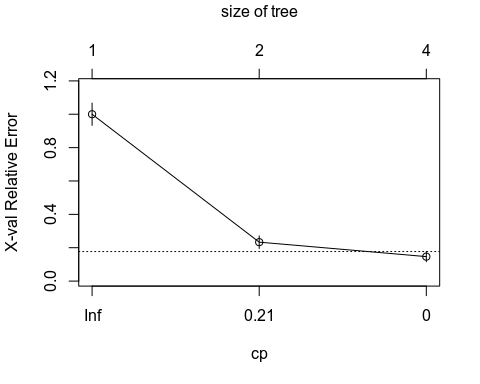
pred.bc\_data.tree=predict(tree.bc\_data,newdata=bc\_data.test,type='class')  
table(bc\_data.test[,"classes"],pred.bc\_data.tree)

## pred.bc\_data.tree  
## benign malignant  
## benign 136 3  
## malignant 7 82

err<-mean(bc\_data.test[,"classes"]!=pred.bc\_data.tree)  
print(err)

## [1] 0.04385965

plotcp(tree.bc\_data)

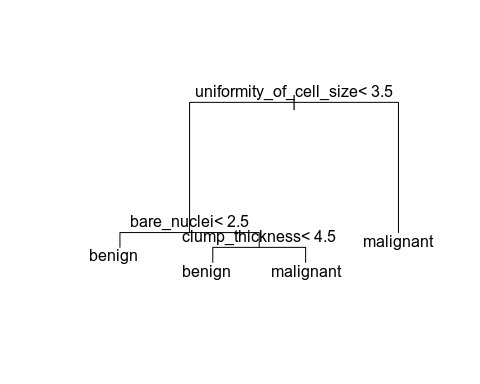


printcp(tree.bc\_data)

##   
## Classification tree:  
## rpart(formula = classes ~ ., data = bc\_data.train[2:11], method = "class",   
## control = rpart.control(xval = 10, minbucket = 5, cp = 0))  
##   
## Variables actually used in tree construction:  
## [1] bare\_nuclei clump\_thickness uniformity\_of\_cell\_size  
##   
## Root node error: 150/455 = 0.32967  
##   
## n= 455   
##   
## CP nsplit rel error xerror xstd  
## 1 0.806667 0 1.000000 1.00000 0.066850  
## 2 0.053333 1 0.193333 0.23333 0.037893  
## 3 0.000000 3 0.086667 0.14667 0.030504

We see that for a complexity of 0.015, we have the smallest cross validation relative error #### Pruning

pruned\_tree<-prune(tree = tree.bc\_data,cp=0.036810)  
plot(pruned\_tree,margin = 0.1)  
text(pruned\_tree,pretty=0)



pred.pruned\_tree=predict(pruned\_tree,newdata=bc\_data.test[2:11],type='class')  
table(bc\_data.test[,"classes"],pred.pruned\_tree)

## pred.pruned\_tree  
## benign malignant  
## benign 136 3  
## malignant 7 82

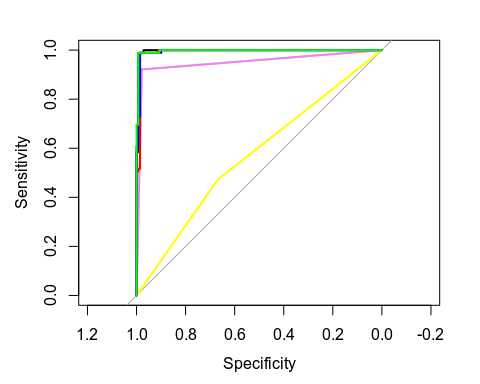
err1<-mean(bc\_data.test[,"classes"]!=pred.pruned\_tree)  
print(err1)

## [1] 0.04385965

As we can see, by prunning, we have a different confusion matrix, a lower complexity but the same error rate. #### Bagging As we can see, trees generally do not have the same level of predictiveaccuracy as some of the other modern regression and classificationapproaches. To improve the quality of the prediction, we can use bagging method.

### Compare the roc curve

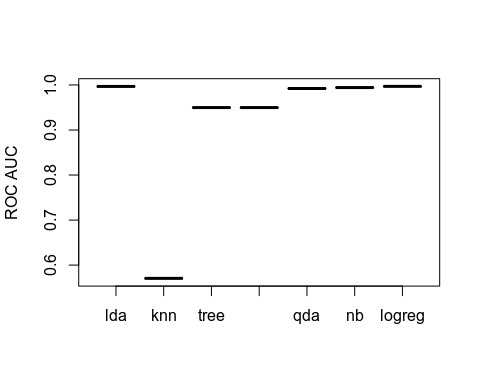
roc\_lda<-roc(bc\_data.test$classes,as.vector(pred.bc\_data.lda$x))  
plot(roc\_lda)  
roc\_knn<-roc(bc\_data.test$classes,as.vector(as.numeric(pred.bc\_data.knn)))  
plot(roc\_knn,add=TRUE,col='yellow')  
roc\_tree<-roc(bc\_data.test$classes,as.vector(as.numeric(pred.bc\_data.tree)))  
plot(roc\_tree,add=TRUE,col='pink')  
roc\_pruned\_tree<-roc(bc\_data.test$classes,as.vector(as.numeric(pred.pruned\_tree)))  
plot(roc\_pruned\_tree,add=TRUE,col='violet')  
roc\_qda<-roc(bc\_data.test$classes,as.vector(pred.bc\_data.qda$posterior[,1]))  
plot(roc\_qda,add=TRUE,col='red')  
roc\_nb<-roc(bc\_data.test$classes,as.vector(pred.bc\_data.nb.prob[,1]))  
plot(roc\_nb,add=TRUE,col='blue')  
roc\_logreg<-roc(bc\_data.test$classes,as.vector(pred.bc\_data.glm))  
plot(roc\_logreg,add=TRUE,col='green')



vec\_auc<-c(auc(roc\_lda),auc(roc\_knn),auc(roc\_tree),auc(roc\_pruned\_tree),auc(roc\_qda),auc(roc\_nb),auc(roc\_logreg))  
  
AUC=matrix(0,1,7)  
  
for (i in (1:7)){  
 print(vec\_auc[i])  
 AUC[1,i]=vec\_auc[i]  
}

## [1] 0.9967666  
## [1] 0.5704874  
## [1] 0.9498828  
## [1] 0.9498828  
## [1] 0.9921591  
## [1] 0.9942608  
## [1] 0.9968475

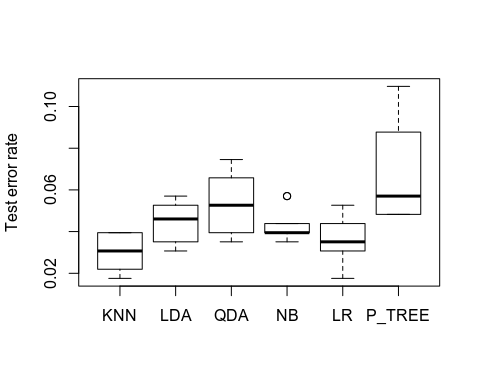
boxplot(AUC,ylab = "ROC AUC", names=c("lda","knn","tree","pruned\_tree","qda","nb","logreg"))

 ###Compare on repeated tests

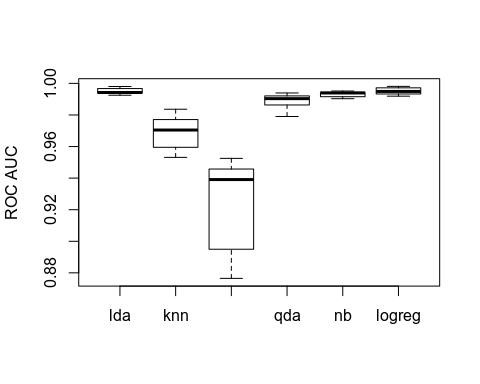
M<-10  
ERR<-matrix(0,M,6)  
AUC=matrix(0,M,6)  
for(i in 1:M){  
 #sampling  
 train<-sample(n,ntrain)  
 bc\_data.train<-bc\_data[train,]  
 bc\_data.test<-bc\_data[-train,]  
 #KNN  
 pred.bc\_data.knn<-knn(bc\_data.train[,2:10],bc\_data.test[,2:10],factor(bc\_data.train$classes),k=5)  
 ERR[i,1]<-mean(bc\_data.test$classes!=pred.bc\_data.knn)  
 #LDA  
 lda.bc\_data <- lda(classes~. ,data=bc\_data.train[,c(-1,-12)])  
 pred.bc\_data.lda<-predict(lda.bc\_data,newdata=bc\_data.test[,c(-1,-12)])  
 perf <-table(bc\_data.test$classes,pred.bc\_data.lda$class)  
 ERR[i,2]<-1-sum(diag(perf))/nrow(bc\_data.test)#mean error rat  
 #QDA  
 qda.bc\_data <- qda(classes~. ,data=bc\_data.train[,c(-1,-12)])  
 pred.bc\_data.qda<-predict(qda.bc\_data,newdata=bc\_data.test[,c(-1,-12)])  
 perf <-table(bc\_data.test$classes,pred.bc\_data.qda$class)  
 ERR[i,3]<-1-sum(diag(perf))/nrow(bc\_data.test)#mean error rate  
 #NB  
 nb.bc\_data <- naive\_bayes(as.factor(classes)~. ,data=bc\_data.train[,c(-1,-12)])  
 pred.bc\_data.nb<-predict(nb.bc\_data,newdata=bc\_data.test[,c(-1,-12)],type="class")  
 pred.bc\_data.nb.prob<-predict(nb.bc\_data,newdata=bc\_data.test[,c(-1,-12)],type="prob")  
 perf <-table(bc\_data.test$classes,pred.bc\_data.nb)  
 ERR[i,4]<-1-sum(diag(perf))/nrow(bc\_data.test)#mean error rate  
 #LOGREG  
 glm.bc\_data <- glm(as.factor(classes)~. ,data=bc\_data.train[,c(-1,-12)],family=binomial)  
 pred.bc\_data.glm<-predict(glm.bc\_data,newdata=bc\_data.test[,c(-1,-12)],type="response")  
 perf <-table(bc\_data.test$classes,pred.bc\_data.glm>0.5)  
 ERR[i,5]<-1-sum(diag(perf))/nrow(bc\_data.test)#mean error rate  
 #PRUNED\_TREE  
 tree.bc\_data<-rpart(classes~.,data=bc\_data.train[2:11],method="class",control=rpart.control(xval = 10,minbucket = 5,cp=0))  
 pruned\_tree<-prune(tree = tree.bc\_data,cp=0.036810)  
 pred.pruned\_tree=predict(pruned\_tree,newdata=bc\_data.test[2:11],type='class')  
 table(bc\_data.test[,"classes"],pred.pruned\_tree)  
 ERR[i,6]<-mean(bc\_data.test[,"classes"]!=pred.pruned\_tree)  
   
 #AUC  
 roc\_lda<-roc(bc\_data.test$classes,as.vector(pred.bc\_data.lda$x))  
 roc\_knn<-roc(bc\_data.test$classes,as.vector(as.numeric(pred.bc\_data.knn)))  
 roc\_pruned\_tree<-roc(bc\_data.test$classes,as.vector(as.numeric(pred.pruned\_tree)))  
 roc\_qda<-roc(bc\_data.test$classes,as.vector(pred.bc\_data.qda$posterior[,1]))  
 roc\_nb<-roc(bc\_data.test$classes,as.vector(pred.bc\_data.nb.prob[,1]))  
 roc\_logreg<-roc(bc\_data.test$classes,as.vector(pred.bc\_data.glm))  
   
 vec\_auc<-c(auc(roc\_lda),auc(roc\_knn),auc(roc\_pruned\_tree),auc(roc\_qda),auc(roc\_nb),auc(roc\_logreg))  
 for (j in (1:6)){  
 print(vec\_auc[j])  
 AUC[i,j]=vec\_auc[j]  
 }  
  
}

## [1] 0.9967317  
## [1] 0.9770792  
## [1] 0.9444827  
## [1] 0.9863249  
## [1] 0.9916573  
## [1] 0.9978498  
## [1] 0.9933775  
## [1] 0.9705857  
## [1] 0.9345489  
## [1] 0.9854649  
## [1] 0.9915713  
## [1] 0.9940655  
## [1] 0.994153  
## [1] 0.9531407  
## [1] 0.9436184  
## [1] 0.9790344  
## [1] 0.9903107  
## [1] 0.9926495  
## [1] 0.9978505  
## [1] 0.9786706  
## [1] 0.8764881  
## [1] 0.9909061  
## [1] 0.9939649  
## [1] 0.9971892  
## [1] 0.9925174  
## [1] 0.9703277  
## [1] 0.8949428  
## [1] 0.9917434  
## [1] 0.9912273  
## [1] 0.9932915  
## [1] 0.9980159  
## [1] 0.9717262  
## [1] 0.9518849  
## [1] 0.9898313  
## [1] 0.9946263  
## [1] 0.9980985  
## [1] 0.99543  
## [1] 0.9595347  
## [1] 0.9457416  
## [1] 0.9939344  
## [1] 0.9951807  
## [1] 0.9957624  
## [1] 0.9942962  
## [1] 0.9672692  
## [1] 0.9337487  
## [1] 0.9888557  
## [1] 0.994384  
## [1] 0.9966655  
## [1] 0.9937996  
## [1] 0.983631  
## [1] 0.8933532  
## [1] 0.9920635  
## [1] 0.9943783  
## [1] 0.9918981  
## [1] 0.9938459  
## [1] 0.9563619  
## [1] 0.9525256  
## [1] 0.993766  
## [1] 0.9939258  
## [1] 0.9939258

boxplot(ERR,ylab="Test error rate",names=c("KNN","LDA","QDA","NB","LR","P\_TREE"))



boxplot(AUC,ylab = "ROC AUC", names=c("lda","knn","pruned\_tree","qda","nb","logreg"))



### Compare using 5 cross validation

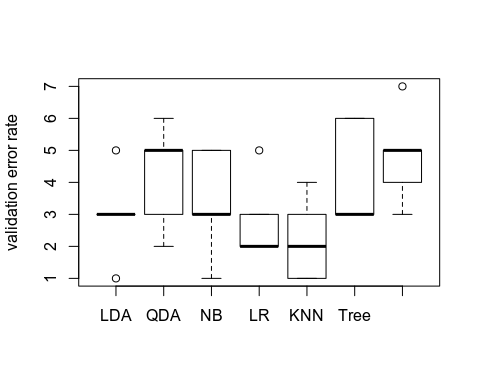
#initialize before the loop  
K<-5  
folds=sample(1:K,ntrain,replace=TRUE)  
CV<-matrix(0,K,7)  
for(k in (1:K)){  
 #Decision Tree  
 tree.bc\_data<-rpart(classes~.,data=bc\_data.train[folds!=k,2:11],method="class",control=rpart.control(xval = 10,minbucket = 5,cp=0))  
   
 pred.bc\_data.tree=predict(tree.bc\_data,newdata=bc\_data.train[folds==k,2:11],type='class')  
 CV[k,6]<-sum(bc\_data.train[folds==k,'classes']!=pred.bc\_data.tree)  
 #Decision Tree Pruned  
 pruned\_tree.bc\_data<-prune(tree = tree.bc\_data,cp=0.036810)  
 pred.bc\_data.pruned\_tree=predict(pruned\_tree.bc\_data,newdata=bc\_data.train[folds==k,2:11],type='class')  
 CV[k,7]<-sum(bc\_data.train[folds==k,'classes']!=pred.bc\_data.pruned\_tree)  
 #KNN  
 knn.bc\_data<-knn(bc\_data.train[folds!=k,2:10],bc\_data.train[folds==k,2:10],factor(bc\_data.train$classes[folds!=k]),k=5)#k=5 this is the best, we choose it before  
 CV[k,5]<-sum(bc\_data.train$classes[folds==k]!=knn.bc\_data)  
 #LDA  
 lda.bc\_data <- lda(classes~. ,data=bc\_data.train[folds!=k,c(-1,-12)])#fit  
 pred.bc\_data.lda<-predict(lda.bc\_data,newdata=bc\_data.train[folds==k,c(-1,-12)])#pred  
 CV[k,1]<-sum(pred.bc\_data.lda$class!=bc\_data.train$classes[folds==k])  
 #QDA  
 qda.bc\_data <- qda(classes~. ,data=bc\_data.train[folds!=k,c(-1,-12)])#fit  
 pred.bc\_data.qda<-predict(qda.bc\_data,newdata=bc\_data.train[folds==k,c(-1,-12)])#pred  
 CV[k,2]<-sum(pred.bc\_data.qda$class!=bc\_data.train$classes[folds==k])  
 #NB  
 nb.bc\_data <- naive\_bayes(as.factor(classes)~. , data = bc\_data.train[folds!=k,c(-1,-12)])#fit  
 pred.bc\_data.nb<-predict(nb.bc\_data,newdata=bc\_data.train[folds==k,c(-1,-12)],type="class")#pred  
 CV[k,3]<-sum(pred.bc\_data.nb!=bc\_data.train$classes[folds==k])  
 #Logreg  
 #glm.bc\_data <- glm(as.factor(classes)~. ,data=bc\_data.train[,c(-1,-12)],family=binomial)  
 #  
 logreg.bc\_data <- glm(as.factor(classes)~. , data = bc\_data.train[folds!=k,c(-1,-12)],family = binomial)#fit  
 pred.bc\_data.logreg<-predict(logreg.bc\_data,newdata=bc\_data.train[folds==k,c(-1,-12)],type="response")#pred  
 perf <-table(bc\_data.train$classes[folds==k],pred.bc\_data.logreg<0.5)#just a matrix to have the errors on the diagonals:  
 CV[k,4]<- sum(diag(perf))#mean error rate  
  
}

## Warning: glm.fit: fitted probabilities numerically 0 or 1 occurred

print(CV)

## [,1] [,2] [,3] [,4] [,5] [,6] [,7]  
## [1,] 5 5 5 3 3 3 4  
## [2,] 3 5 5 5 4 6 7  
## [3,] 3 2 1 2 1 3 5  
## [4,] 1 3 3 2 1 3 3  
## [5,] 3 6 3 2 2 6 5

boxplot(CV,ylab='validation error rate',names=c("LDA","QDA","NB","LR","KNN","Tree","Pruned\_T"))



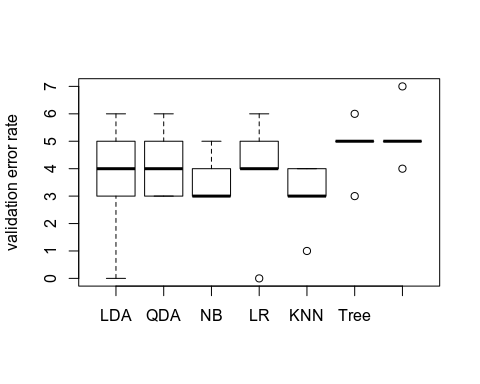
err\_cv=colSums(CV)/ntrain  
print(err\_cv)

## [1] 0.03296703 0.04615385 0.03736264 0.03076923 0.02417582 0.04615385  
## [7] 0.05274725

function.cross\_validation<-function(dataset=bc\_data){  
   
 n<-nrow(dataset)  
 ntrain<-round(2\*n/3)  
 ntest<-n-ntrain  
 train<-sample(n,ntrain)  
 bc\_data.train<-dataset[train,]  
 bc\_data.test<-dataset[-train,]  
 #initialize before the loop  
 K<-5  
 folds=sample(1:K,ntrain,replace=TRUE)  
 CV<-matrix(0,K,7)  
 for(k in (1:K)){  
 #Decision Tree  
 tree.bc\_data<-rpart(classes~.,data=bc\_data.train[folds!=k,2:11],method="class",control=rpart.control(xval = 10,minbucket = 5,cp=0))  
   
 pred.bc\_data.tree=predict(tree.bc\_data,newdata=bc\_data.train[folds==k,2:11],type='class')  
 CV[k,6]<-sum(bc\_data.train[folds==k,'classes']!=pred.bc\_data.tree)  
 #Decision Tree Pruned  
 pruned\_tree.bc\_data<-prune(tree = tree.bc\_data,cp=0.036810)  
 pred.bc\_data.pruned\_tree=predict(pruned\_tree.bc\_data,newdata=bc\_data.train[folds==k,2:11],type='class')  
 CV[k,7]<-sum(bc\_data.train[folds==k,'classes']!=pred.bc\_data.pruned\_tree)  
 #KNN  
 knn.bc\_data<-knn(bc\_data.train[folds!=k,2:10],bc\_data.train[folds==k,2:10],factor(bc\_data.train$classes[folds!=k]),k=5)#k=5 this is the best, we choose it before  
 CV[k,5]<-sum(bc\_data.train$classes[folds==k]!=knn.bc\_data)  
 #LDA  
 lda.bc\_data <- lda(classes~. ,data=bc\_data.train[folds!=k,c(-1,-12)])#fit  
 pred.bc\_data.lda<-predict(lda.bc\_data,newdata=bc\_data.train[folds==k,c(-1,-12)])#pred  
 CV[k,1]<-sum(pred.bc\_data.lda$class!=bc\_data.train$classes[folds==k])  
 #QDA  
 qda.bc\_data <- qda(classes~. ,data=bc\_data.train[folds!=k,c(-1,-12)])#fit  
 pred.bc\_data.qda<-predict(qda.bc\_data,newdata=bc\_data.train[folds==k,c(-1,-12)])#pred  
 CV[k,2]<-sum(pred.bc\_data.qda$class!=bc\_data.train$classes[folds==k])  
 #NB  
 nb.bc\_data <- naive\_bayes(as.factor(classes)~. , data = bc\_data.train[folds!=k,c(-1,-12)])#fit  
 pred.bc\_data.nb<-predict(nb.bc\_data,newdata=bc\_data.train[folds==k,c(-1,-12)],type="class")#pred  
 CV[k,3]<-sum(pred.bc\_data.nb!=bc\_data.train$classes[folds==k])  
 #Logreg  
 #glm.bc\_data <- glm(as.factor(classes)~. ,data=bc\_data.train[,c(-1,-12)],family=binomial)  
 #  
 logreg.bc\_data <- glm(as.factor(classes)~. , data = bc\_data.train[folds!=k,c(-1,-12)],family = binomial)#fit  
 pred.bc\_data.logreg<-predict(logreg.bc\_data,newdata=bc\_data.train[folds==k,c(-1,-12)],type="response")#pred  
 perf <-table(bc\_data.train$classes[folds==k],pred.bc\_data.logreg<0.5)#just a matrix to have the errors on the diagonals:  
 CV[k,4]<- sum(diag(perf))#mean error rate  
   
 }  
 print(CV)  
 boxplot(CV,ylab='validation error rate',names=c("LDA","QDA","NB","LR","KNN","Tree","Pruned\_T"))  
 err\_cv=colSums(CV)/ntrain  
 print(err\_cv)  
}

function.cross\_validation()

## [,1] [,2] [,3] [,4] [,5] [,6] [,7]  
## [1,] 6 6 5 6 3 5 4  
## [2,] 4 3 3 4 3 5 5  
## [3,] 0 3 3 0 1 3 5  
## [4,] 3 4 4 5 4 5 7  
## [5,] 5 5 3 4 4 6 5



## [1] 0.03956044 0.04615385 0.03956044 0.04175824 0.03296703 0.05274725  
## [7] 0.05714286

### Decision on the best model,

We will chose the model with thelogistic regression because it offers the best perfomences on the trainning set.

## Regression for a dataset without missing values

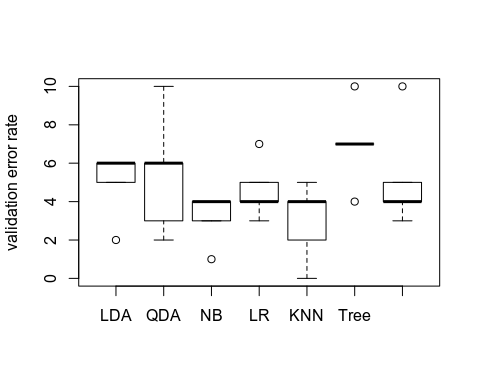
reg<- lm(bare\_nuclei ~. , data=bc\_data[,2:10])  
pred<-predict(reg,newdata=bc\_data\_missing[is.na(bc\_data\_missing$bare\_nuclei),2:10])  
print(length(pred))

## [1] 16

bc\_data\_missing\_lm<-bc\_data\_missing  
bc\_data\_missing\_lm[,"bare\_nuclei"]<-as.numeric(bc\_data\_missing[,"bare\_nuclei"])  
bc\_data\_missing\_lm[is.na(bc\_data\_missing$bare\_nuclei),"bare\_nuclei"]<-pred  
#print(bc\_data\_missing[is.na(bc\_data\_missing$bare\_nuclei),2:10])  
  
#for( i in bc\_data\_missing[is.na(bc\_data\_missing$bare\_nuclei),2:10]){  
# print(length(i[1]))  
# print(i[1])  
# print(i)  
 #i<-predict(reg,newdata=bc\_data\_missing[is.na(bc\_data\_missing$bare\_nuclei),2:10])  
#}  
function.cross\_validation(bc\_data\_missing\_lm)

## Warning: glm.fit: fitted probabilities numerically 0 or 1 occurred

## [,1] [,2] [,3] [,4] [,5] [,6] [,7]  
## [1,] 6 6 4 4 5 10 10  
## [2,] 5 10 4 7 4 4 4  
## [3,] 2 3 1 3 0 7 5  
## [4,] 6 2 3 5 2 7 4  
## [5,] 6 6 4 4 4 7 3



## [1] 0.05364807 0.05793991 0.03433476 0.04935622 0.03218884 0.07510730  
## [7] 0.05579399

bc\_data\_missing\_knn<-bc\_data\_missing  
  
for (i in 1:ncol(bc\_data\_missing\_knn)){  
 if (class(bc\_data\_missing\_knn[5,i])=='factor'){  
 bc\_data\_missing\_knn[,i] = as.numeric(bc\_data\_missing\_knn[,i])  
 }  
}  
  
for (i in 1:ncol(bc\_data\_knn)){  
 if (class(bc\_data\_knn[5,i])=='factor'){  
 bc\_data\_knn[,i] = as.numeric(bc\_data\_knn[,i])  
 }  
}  
bc\_data.train2<-scale(bc\_data\_knn[,-11])  
y.train<-bc\_data\_knn$bare\_nuclei  
#print(y.train)  
#print(bc\_data.train2)  
reg<-knn.reg(train=bc\_data.train2, y=y.train, k = 5)# on applique le model knn en se basant sur le training valeurs on fait les test sur la partie réservée. y est la réponse des valeur train  
  
#mean((y.test-reg$pred)^2) # = Mean squared error   
#plot(y.test,reg$pred,xlab='y',ylab='prediction')   
#si les point sont sur la ligne x=y c'est que le aleur prédite est la même que la vraie valuer   
#abline(0,1)  
#knn.predict(train=bc\_data.train2, bc\_data\_missing\_knn, y=y.train, k=5)  
#pred<-predict(reg,newdata=bc\_data\_missing[is.na(bc\_data\_missing$bare\_nuclei),2:10])

# REGRESSION

#Clean data  
  
bc\_data <- read.table("breast-cancer-wisconsin.data", header = FALSE, sep = ",")  
head(bc\_data)

## V1 V2 V3 V4 V5 V6 V7 V8 V9 V10 V11  
## 1 1000025 5 1 1 1 2 1 3 1 1 2  
## 2 1002945 5 4 4 5 7 10 3 2 1 2  
## 3 1015425 3 1 1 1 2 2 3 1 1 2  
## 4 1016277 6 8 8 1 3 4 3 7 1 2  
## 5 1017023 4 1 1 3 2 1 3 1 1 2  
## 6 1017122 8 10 10 8 7 10 9 7 1 4

colnames(bc\_data) <- c("sample\_code\_number", "clump\_thickness", "uniformity\_of\_cell\_size", "uniformity\_of\_cell\_shape", "marginal\_adhesion", "single\_epithelial\_cell\_size", "bare\_nuclei", "bland\_chromatin", "normal\_nucleoli", "mitosis", "classes")  
  
bc\_data$classes.num=bc\_data$classes # we copie the class column and name it classes.num  
bc\_data$classes.num[bc\_data$classes.num=="2"] <- 0  
bc\_data$classes.num[bc\_data$classes.num=="4"] <- 1  
bc\_data$classes[bc\_data$classes=="2"] <- "benign"  
bc\_data$classes[bc\_data$classes=="4"] <- "malignant"  
head(bc\_data)

## sample\_code\_number clump\_thickness uniformity\_of\_cell\_size  
## 1 1000025 5 1  
## 2 1002945 5 4  
## 3 1015425 3 1  
## 4 1016277 6 8  
## 5 1017023 4 1  
## 6 1017122 8 10  
## uniformity\_of\_cell\_shape marginal\_adhesion single\_epithelial\_cell\_size  
## 1 1 1 2  
## 2 4 5 7  
## 3 1 1 2  
## 4 8 1 3  
## 5 1 3 2  
## 6 10 8 7  
## bare\_nuclei bland\_chromatin normal\_nucleoli mitosis classes  
## 1 1 3 1 1 benign  
## 2 10 3 2 1 benign  
## 3 2 3 1 1 benign  
## 4 4 3 7 1 benign  
## 5 1 3 1 1 benign  
## 6 10 9 7 1 malignant  
## classes.num  
## 1 0  
## 2 0  
## 3 0  
## 4 0  
## 5 0  
## 6 1

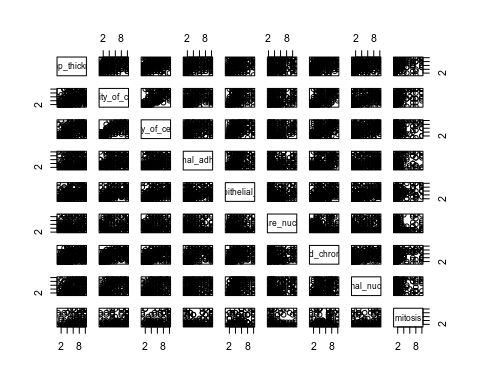
bc\_data[bc\_data=="?"]<-NA  
nrow(bc\_data)-length(complete.cases(bc\_data)[complete.cases(bc\_data)==TRUE])

## [1] 16

length(bc\_data$bare\_nuclei[is.na(bc\_data$bare\_nuclei)])

## [1] 16

bc\_data\_sansna<-na.omit(bc\_data)  
  
# We want to predict the bare nuclei  
  
data<-bc\_data\_sansna[,2:10]  
plot(data)



#on va essayer de prédire toute la colonne bare nuclei comme ça   
  
library('FNN')  
  
for (i in 1:ncol(data)){  
 if (class(data[5,i])=='factor'){  
 data[,i] = as.numeric(data[,i])  
 print(c(i, "DONE"))  
   
 }  
 print(c(i,class(data[,i])))  
}

## [1] "1" "integer"  
## [1] "2" "integer"  
## [1] "3" "integer"  
## [1] "4" "integer"  
## [1] "5" "integer"  
## [1] "6" "DONE"  
## [1] "6" "numeric"  
## [1] "7" "integer"  
## [1] "8" "integer"  
## [1] "9" "integer"

n<-nrow(data)  
ntrain<-round(2\*n/3)  
ntest<-n-ntrain  
train<-sample(n,ntrain)  
bc\_data.train<-data[train,]  
bc\_data.train2<-scale(bc\_data.train)  
y.train<-bc\_data.train$bare\_nuclei  
bc\_data.test<-data[-train,]  
bc\_data.test2<-scale(bc\_data.test)  
y.test<-bc\_data.test$bare\_nuclei  
  
  
"x.train<-scale(data[data$train==T,1:4]) #on garde que les colonnes 1,2,3 et4 où train = true  
summary(x.train)  
y.train<-data[data$train==T,5]# que la colonne lpsa où train = true   
y.train  
x.tst<-scale(data[data$train==F,1:4])#les 4 premieres colonnes où train =F  
y.tst<-data[data$train==F,5]# que la colonne lpsa où train = f"

## [1] "x.train<-scale(data[data$train==T,1:4]) #on garde que les colonnes 1,2,3 et4 où train = true\nsummary(x.train)\ny.train<-data[data$train==T,5]# que la colonne lpsa où train = true \ny.train\nx.tst<-scale(data[data$train==F,1:4])#les 4 premieres colonnes où train =F\ny.tst<-data[data$train==F,5]# que la colonne lpsa où train = f"

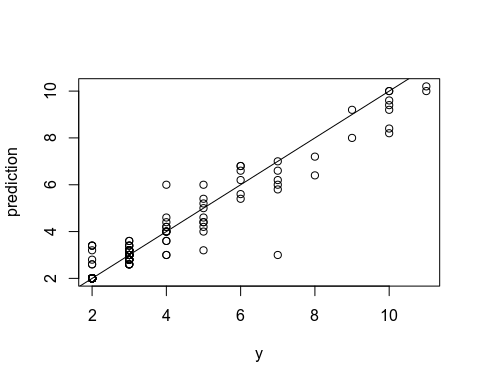
class(bc\_data[,2])

## [1] "integer"

reg<-knn.reg(train=bc\_data.train2, test = bc\_data.test2, y=y.train, k = 5)# on applique le model knn en se basant sur le training valeurs on fait les test sur la partie réservée. y est la réponse des valeur train  
  
mean((y.test-reg$pred)^2) # = Mean squared error

## [1] 0.2614035

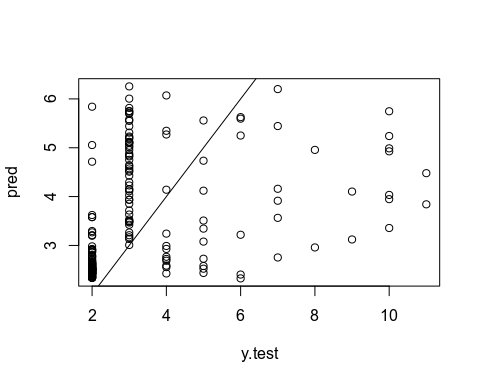
plot(y.test,reg$pred,xlab='y',ylab='prediction')   
#si les point sont sur la ligne x=y c'est que le aleur prédite est la même que la vraie valuer   
abline(0,1)



reg<- lm(bare\_nuclei ~. , data=bc\_data.train)  
reg

##   
## Call:  
## lm(formula = bare\_nuclei ~ ., data = bc\_data.train)  
##   
## Coefficients:  
## (Intercept) clump\_thickness   
## 1.93071 0.03252   
## uniformity\_of\_cell\_size uniformity\_of\_cell\_shape   
## 0.21960 0.06051   
## marginal\_adhesion single\_epithelial\_cell\_size   
## -0.05690 -0.02192   
## bland\_chromatin normal\_nucleoli   
## 0.08742 0.06171   
## mitosis   
## 0.04729

pred<-predict(reg,newdata=bc\_data.test)  
plot(y.test,pred)  
abline(0,1)



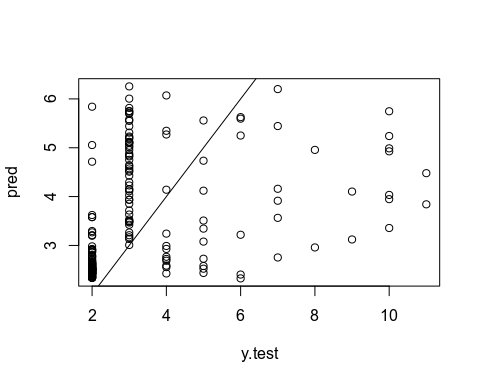
mse<-mean((y.test-pred)^2)  
mse

## [1] 3.443953

reg<- lm(bare\_nuclei ~. , data=bc\_data.train)  
reg

##   
## Call:  
## lm(formula = bare\_nuclei ~ ., data = bc\_data.train)  
##   
## Coefficients:  
## (Intercept) clump\_thickness   
## 1.93071 0.03252   
## uniformity\_of\_cell\_size uniformity\_of\_cell\_shape   
## 0.21960 0.06051   
## marginal\_adhesion single\_epithelial\_cell\_size   
## -0.05690 -0.02192   
## bland\_chromatin normal\_nucleoli   
## 0.08742 0.06171   
## mitosis   
## 0.04729

pred<-predict(reg,newdata=bc\_data.test)  
plot(y.test,pred)  
abline(0,1)



mse<-mean((y.test-pred)^2)  
mse

## [1] 3.443953