Introduction to Machine Learning - Lab 2

Ugurcan Lacin

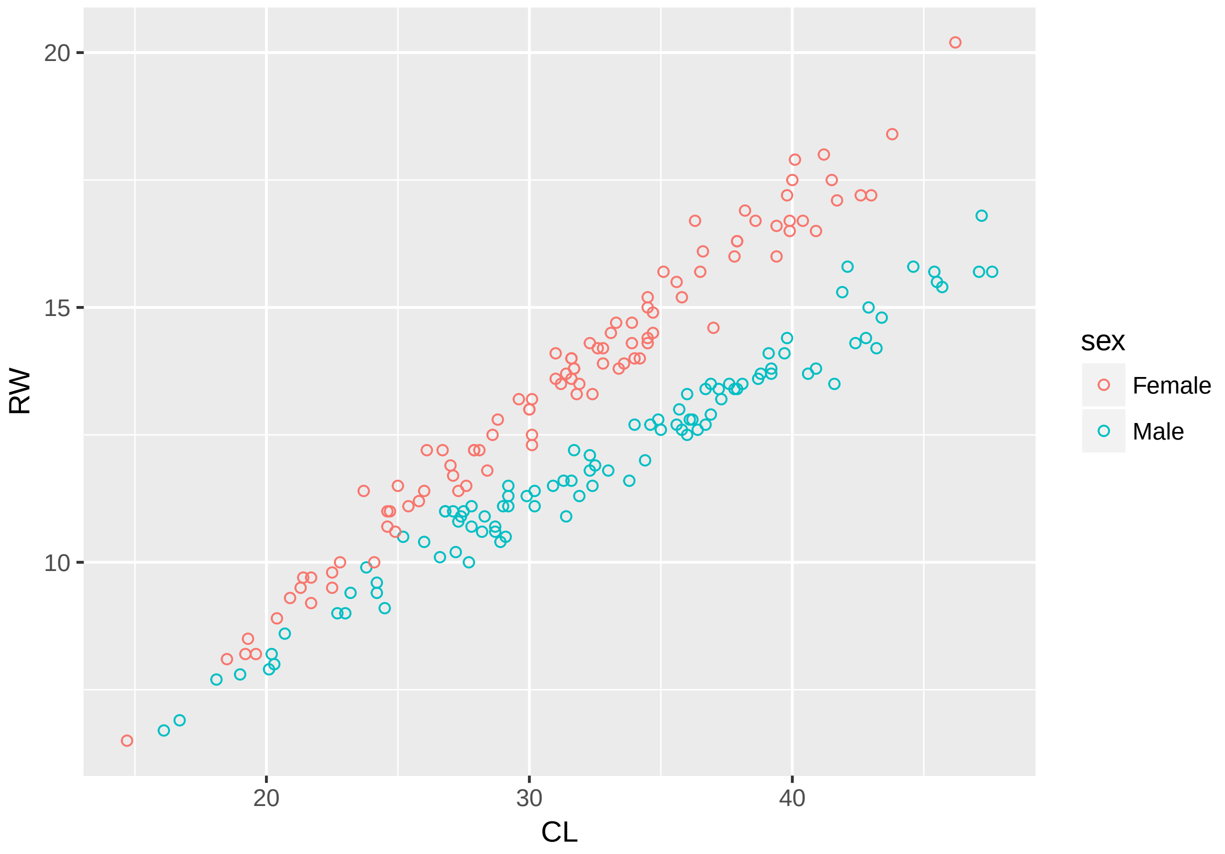
11/16/2017

Assignment 1

In this assignment, *australian-crabs.csv* file will be used which contains measurements of various crabs. The main purpose is trying to predict sex by RW and CL as inputs. Before starting build a model, the data will be analyzed by using scatterplot to understand data shape.

Part 1

```r
df <- read.csv("/home/ugur/git/ML_Lab2_Group/australian-crabs.csv",
               sep = ",")
library(ggplot2)

ggplot(df, aes(x=CL, y=RW, color=sex)) + geom_point(shape=1)
```



According to scatterplot of carapace length (CL) versus rear width (RW) where observations are colored by Sex, Data is easy to classify by Linear Discriminant Analysis (LDA). This is because RW and CL are increasing linearly, and they are not going in the same direction, which means seperated from each other. So, decision boundaries are linear and LDA is applicable.

Part 2

Finding w_{0i} and w_i is necessary to compute discrimant function. They are calculated in disc_func() by applying formulas on the below. After this, classification is possible with these calculated w_{0i} and w_i.

$$w_{0i} = -\frac{1}{2}\mu_i^T \sum^{-1} \mu_i + \log \pi_i$$

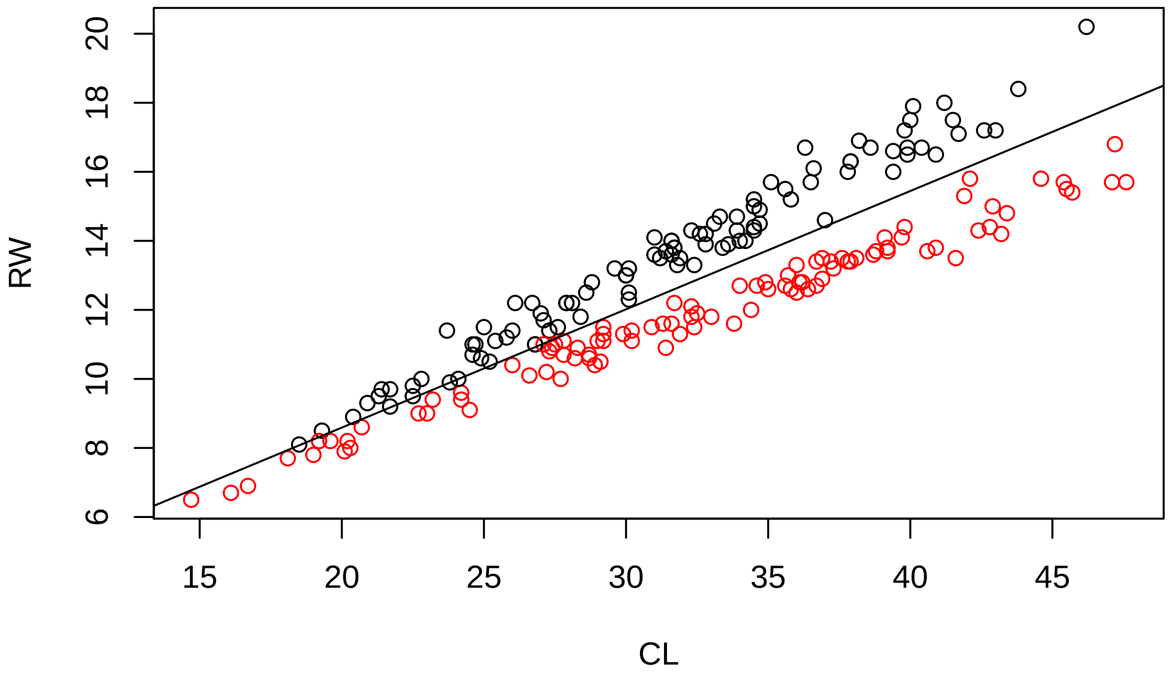
$$w_i = \sum^{-1} \mu_i$$

```r
Y <- df$sex
X <- data.frame(RW = df$RW,CL = df$CL)
X <- as.matrix(X)

disc_fun=function(label, S){
  X1=X[Y==label,]
  means <- as.matrix(colMeans(X1))
  b1 <- -0.5 * t(means) %*% solve(S) %*% means + log(0.5)
  w1 <- solve(S) %*% means
  return(c(w1[1], w1[2], b1[1,1]))
}

X1=X[Y=="Male",]
X2=X[Y=="Female",]

S=cov(X1)*dim(X1)[1]+cov(X2)*dim(X2)[1]
S=S/dim(X)[1]

#discriminant function coefficients
res1=disc_fun("Male",S)
res2=disc_fun("Female",S)
res <- res1 - res2

# classification
d=res[1]*X[,1]+res[2]*X[,2]+res[3]
Yfit=(d>0)
```

Part 3

Data is classified by using w_{0i} and w_i found. Now this classified data is plotted as x-axis CL and y-axis RW. Intercept and slope values are calculated by helping of w_{0i} and w_i. Thus there is a decision boundary between the two classes. When the decision boundary is drawn, it can be observed that the two classes are uniformly separated from the center. Thus, the data was classified first, and then the decision line is used to visualize the classification.

```r
plot(X[,2], X[,1], col=Yfit+1, xlab="CL", ylab="RW")
intercept <- -res[3]/res[1]
slope <- -res[2]/res[1]
abline(intercept,slope)
```



The confusion matrix result appears below because it is difficult to graphically interpret the consistency of the classified data. According to the confusion matrix result, the data seems to be classified with a high consistency.

```r
table(Yfit,df$sex)
```

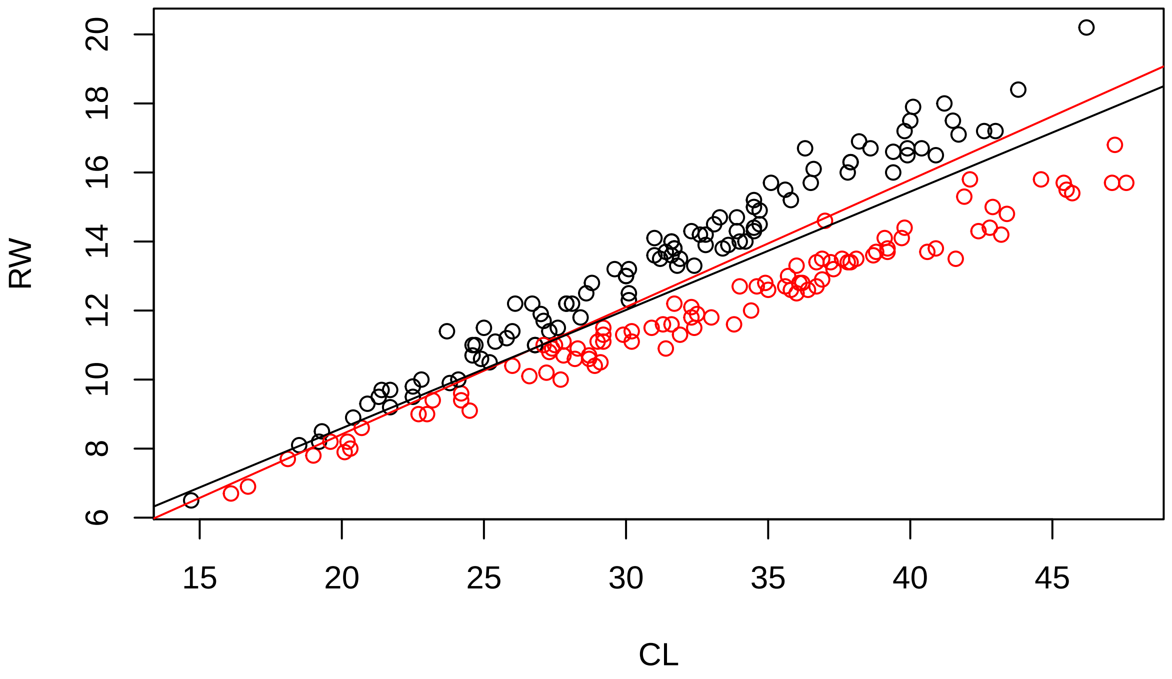
```
##
## Yfit    Female Male
##   FALSE     97    4
##   TRUE       3   96
```

Part 4

Now another method, logistic regression, is used. The model is trained using the glm () function. Then we take the coefficients and calculate weights for both RW and CL. Thus, the decision boundary is calculated here as calculated in the same liner discrement analysis section. Both decision boundaries are drawn for a comparative graph. The line drawn with red is the line for logistic regression. The other is the decision boundary belonging to the method of discrement analysis. It is unlikely to make a clear statement from the graph. For this reason, we are looking at confusion matrix results to compare numerical accuracy.

```r
glmResult <- glm(data = df,formula = as.factor(sex) ~ CL + RW , family = binomial())
coef <- as.matrix(glmResult$coefficients)
```

```
rwWeight <- coef[3]
clWeight <- coef[2]
inter <- coef[1] - 0.5
d=coef[3]*X[,1]+coef[2]*X[,2]+coef[1]
YfitGLM=(d>0.5)

plot(X[,2], X[,1], col=YfitGLM+1, xlab="CL", ylab="RW")
interceptGLM <- -inter/rwWeight
slopeGLM <- -(clWeight)/rwWeight
abline(interceptGLM,slopeGLM,col = "red")
abline(intercept,slope)
```



The confusion matrix results of both models can be seen below. According to the results, the glm () function looks like a better model by making one more accurate guess. But such a small difference certainly does not prove that the glm () function is better than the LDA. More data can be consulted to get a clearer result.

```
# comparison
# GLM
print(table(YfitGLM,df$sex))
```

```
##
## YfitGLM Female Male
##   FALSE     98    4
##   TRUE       2   96
```

```
# LDA
print(table(Yfit,df$sex))
```

```
##
## Yfit    Female Male
##   FALSE     97    4
##   TRUE       3   96
```

Assignment 2

Part 1

Data is imported to R and divided into training/validation/test as 50/25/25 by using set.seed(12345).

```
library(gdata)
data <- read.xls("/home/ugur/git/ML_Lab2_Group/creditscoring.xls")

n=dim(data)[1]
set.seed(12345)
id=sample(1:n, floor(n*0.5))
train=data[id,]

rest=data[-id,]
n=dim(rest)[1]
id=sample(1:n, floor(n*0.5))
test=rest[id,]
valid=rest[-id,]
```

Part 2

```
library(tree)

confusion_matrix <- function(true_res, pred){
  tab <- table(true_res, pred)
  tab_plot <- addmargins(table(true_res, pred))
  miss <- 1 - sum(diag(tab))/sum(tab)
  l <- list(confusion_matrix = tab_plot,
  Missclassification_error = miss)
  return(l)
}

fitByGini=tree(good_bad~., data=train,method = "gini")
giniTrainPred <- predict(fitByGini, newdata=train,type="class")
giniTestPred <- predict(fitByGini, newdata=test,type="class")

misclassificationRate <- confusion_matrix(train$good_bad,giniTrainPred)
print(misclassificationRate)
```

```
## $confusion_matrix
##         pred
## true_res bad good Sum
##     bad   61   86 147
##     good  20  333 353
##     Sum   81  419 500
##
```

```
## $Missclassification_error
## [1] 0.212
```

```
misclassificationRate <- confusion_matrix(test$good_bad,giniTestPred)
print(misclassificationRate)
```

```
## $confusion_matrix
##         pred
## true_res bad good Sum
##     bad   34   41  75
##     good  21  154 175
##     Sum   55  195 250
##
## $Missclassification_error
## [1] 0.248
```

Gini, which is a measure of impurity, is used to fit a tree. Then, predictions are produced with both train and test data. With these predictions, the misclassification rates for both train and test data are calculated separately. When these rates are examined, the following interpretation can be made. The misclassification rate of the train data is lower than that of the test data. That's why the model is trained with the train data set. Therefore, the train learns more about the template in the that dataset.

```
fitByDeviance=tree(good_bad~., data=train,method = "deviance")
devianceTrainPred <- predict(fitByDeviance, newdata=train,type="class")
devianceTestPred <- predict(fitByDeviance, newdata=test,type="class")

misclassificationRate <- confusion_matrix(train$good_bad,devianceTrainPred)
print(misclassificationRate)
```

```
## $confusion_matrix
##         pred
## true_res bad good Sum
##     bad   61   86 147
##     good  20  333 353
##     Sum   81  419 500
##
## $Missclassification_error
## [1] 0.212
```

```
misclassificationRate <- confusion_matrix(test$good_bad,devianceTestPred)
print(misclassificationRate)
```

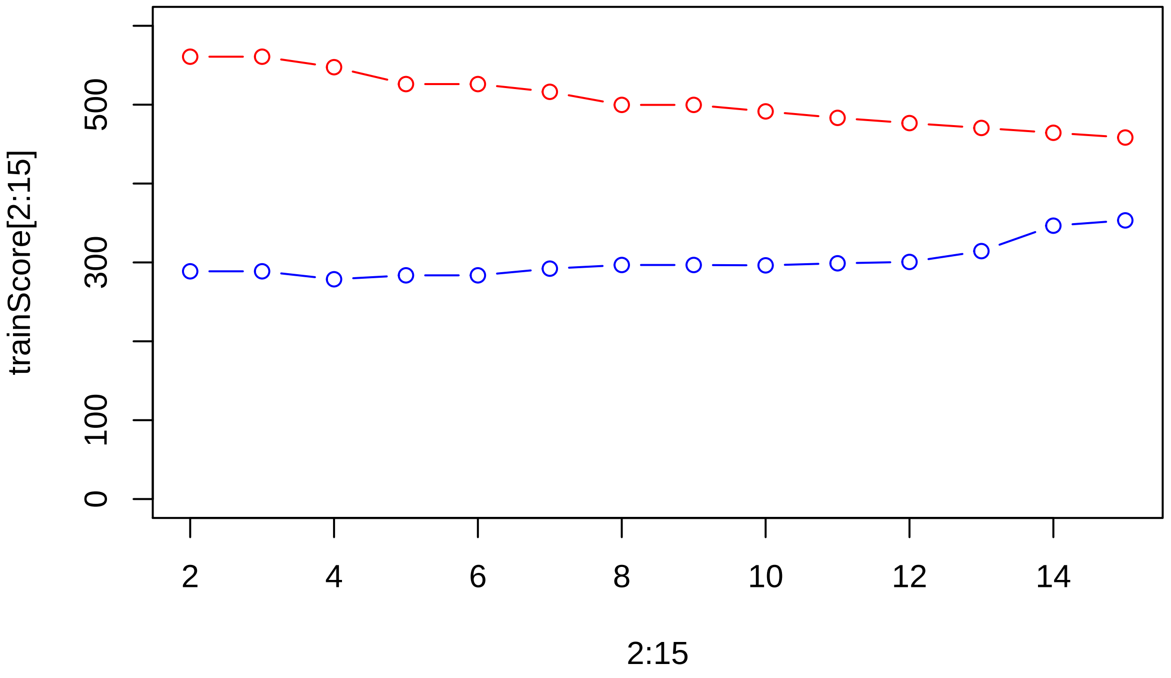
```
## $confusion_matrix
##         pred
## true_res bad good Sum
##     bad   34   41  75
##     good  21  154 175
##     Sum   55  195 250
##
## $Missclassification_error
## [1] 0.248
```

This time Deviance, which is another measure of impurity, is used to fit a tree. Then, predictions are produced with both train and test data again, and misclassification rates are calculated as previously calculated. Train misclassification rate is still higher than test misclassification rate for the same reason.

Part 3

The train and validation data sets will now be used to find the optimal tree depth. Depths between 2 and 15 are tested individually. Each train and validation result is saved as a vector and finally plot is drawn. The red line reflects the results of the train set, and the blue line reflects the results of the validation set.

```r
fit=tree(good_bad~., data=train)
trainScore=rep(0,15)
testScore=rep(0,15)
for(i in 2:15) {
  prunedTree=prune.tree(fit,best=i) # i is tree depth here.
  pred=predict(prunedTree, newdata=valid,
               type="tree")
  trainScore[i]=deviance(prunedTree)
  testScore[i]=deviance(pred)
}

plot(2:15, trainScore[2:15], type="b", col="red",
     ylim=c(0,600))
points(2:15, testScore[2:15], type="b", col="blue")
```

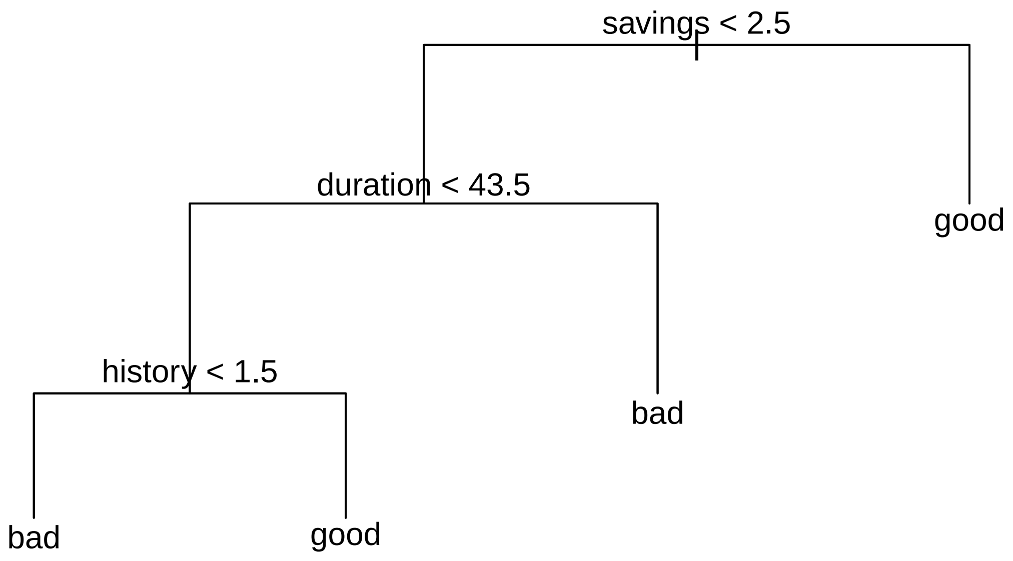


To find the best depth, the difference between the train and test results was calculated and the lowest index value was selected. This is the reason to minimize the problem of overfitting. If the difference is small, the risk of overfitting will be lowest.

```
minimumScore <- min(testScore[2:15])
bestDepth <- which(minimumScore == testScore)
print(bestDepth)
```

```
## [1] 4
```

```
bestTree <- prune.tree(fit,best=bestDepth)
plot(bestTree)
text(bestTree)
```



```
YpredTest <- predict(bestTree, newdata=test,type="class")
misclassificationRate <- confusion_matrix(test$good_bad,YpredTest)
print(misclassificationRate)
```

```
## $confusion_matrix
##         pred
## true_res bad good Sum
##     bad   22   53  75
##     good  12  163 175
##     Sum   34  216 250
##
## $Missclassification_error
## [1] 0.26
```

The tree is printed which fitted according to the optimum tree depth. It can be seen that each variable has a different decision parameter. Each decision goes on to another decision branch.

Finally, the misclassification rate is printed using the test data.

Part 4

```
library(MASS)
library(e1071)
fit=naiveBayes(as.factor(good_bad)~., data=train)
```

```r
Yfit=predict(fit, newdata=train)
YfitTest=predict(fit, newdata=test)

misclassificationRate <- confusion_matrix(train$good_bad,Yfit)
print(misclassificationRate)
```

```
## $confusion_matrix
##         pred
## true_res bad good Sum
##     bad   95   52 147
##     good  98  255 353
##     Sum  193  307 500
##
## $Missclassification_error
## [1] 0.3
```

```r
misclassificationRate <- confusion_matrix(test$good_bad,YfitTest)
print(misclassificationRate)
```

```
## $confusion_matrix
##         pred
## true_res bad good Sum
##     bad   50   25  75
##     good  61  114 175
##     Sum  111  139 250
##
## $Missclassification_error
## [1] 0.344
```

In this step a model is being trained using Naive Bayes. Then the model is tested with both train and test data. Confusion matrix results and misclassification rates can be seen above. According to result, misclassification error is higher than step 3.

Part 5

```r
# Fit Bayes Naive algorithm
fit_naive2=naiveBayes(as.factor(good_bad)~., data=train)
# Make raw predictions
YfitTrain=predict(fit_naive2, newdata=train, type= "raw")
# Classify data manually
Ypredict_final <- YfitTrain[,1]/YfitTrain[,2] <= 10

misclassificationRate <- confusion_matrix(train$good_bad,Ypredict_final)
print(misclassificationRate)
```

```
## $confusion_matrix
##         pred
## true_res FALSE TRUE Sum
##     bad     27  120 147
##     good    17  336 353
##     Sum     44  456 500
```

```
##
## $Missclassification_error
## [1] 0.274
```

```
# Make raw predictions for test
YfitTest=predict(fit_naive2, newdata=test, type= "raw")
# Classify data manually
Ypredict_final <- YfitTest[,1]/YfitTest[,2] <= 10

misclassificationRate <- confusion_matrix(test$good_bad,Ypredict_final)
print(misclassificationRate)
```

```
## $confusion_matrix
##          pred
## true_res FALSE TRUE Sum
##     bad     17   58  75
##     good    13  162 175
##     Sum     30  220 250
##
## $Missclassification_error
## [1] 0.284
```

There is better results than previous step with custom loss matrix.This is due to the fact that now only datapoints that are 10 times more likely to be "bad" than "good" are actually classified as being "bad". This means that the misclassification of "bad" points are decreased (in this part only $7/172 * 100 = 4.069767\%$ of datapoints classified as "good" were in fact "bad" compared to $98/353 * 100 = 27.76204\%$ (train) and $49/250 * 100 = 19.6\%$ (test data) in part 4). However, the trade off is that some more "good" datapoints are misclassified as "bad": $65/172 * 100 = 37.7907\%$ in part 5 compared to $52/353 * 100 = 14.73088\%$ (training data) and $31/172 * 100 = 18.02326\%$ (testing data) in part 4.

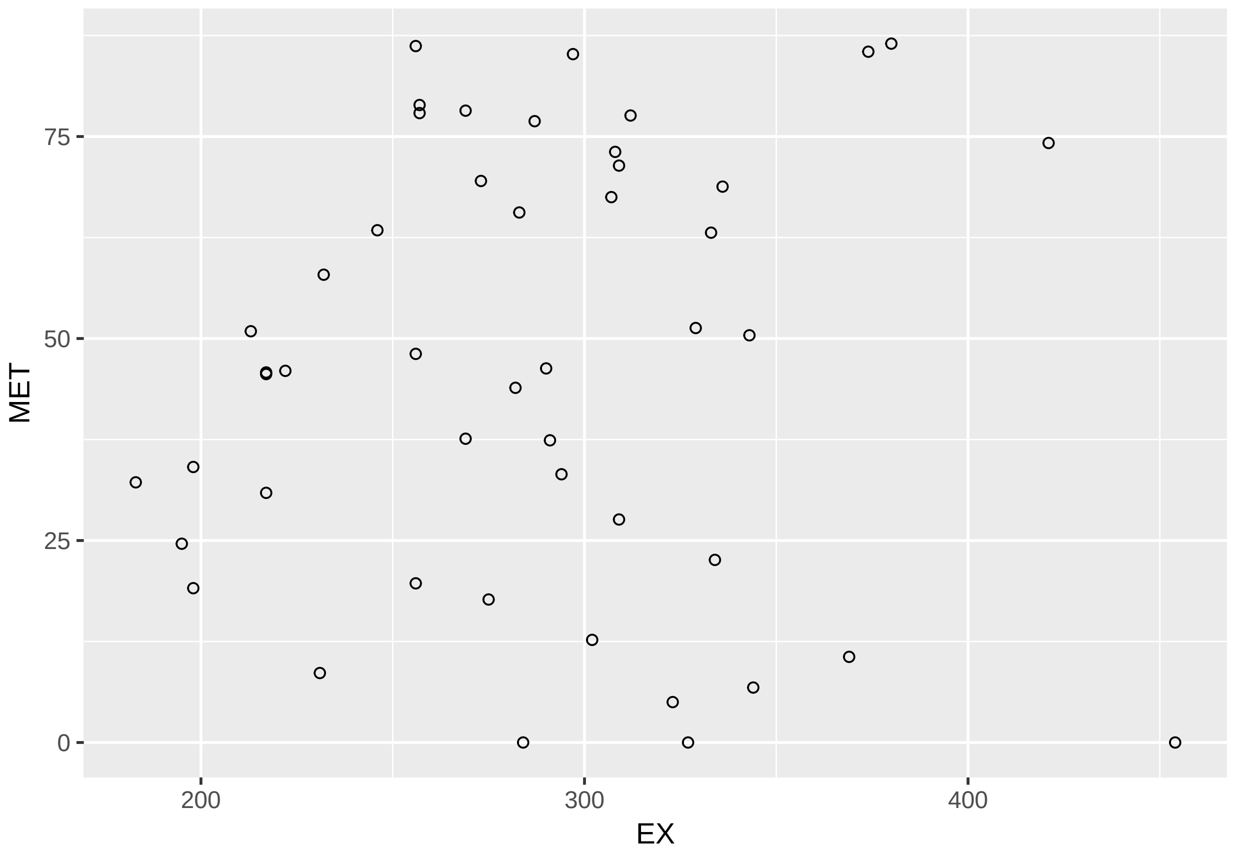
Assignment 3

Part 1

Data is imported in R. Then, it is reordered with respect to the increase of MET, and plot is drawn with EX versus MET.

```
df <- read.csv("/home/ugur/git/ML_Lab2_Group/State.csv",
               sep = ";")
library(dplyr)
options(digits=2)

# Replace , as . for R
df$MET <- gsub("\\,",".",df$MET)
# Change type as numeric from factor
df$MET <- as.numeric(df$MET)
df$EX <- as.numeric(df$EX)
# Reorder it
df <- arrange(df, MET)

tree_data <- data.frame(EX = df$EX, MET = df$MET)
data <- tree_data
```

```r
library(ggplot2)
ggplot(data, aes(x=EX, y=MET)) + geom_point(shape=1)
```

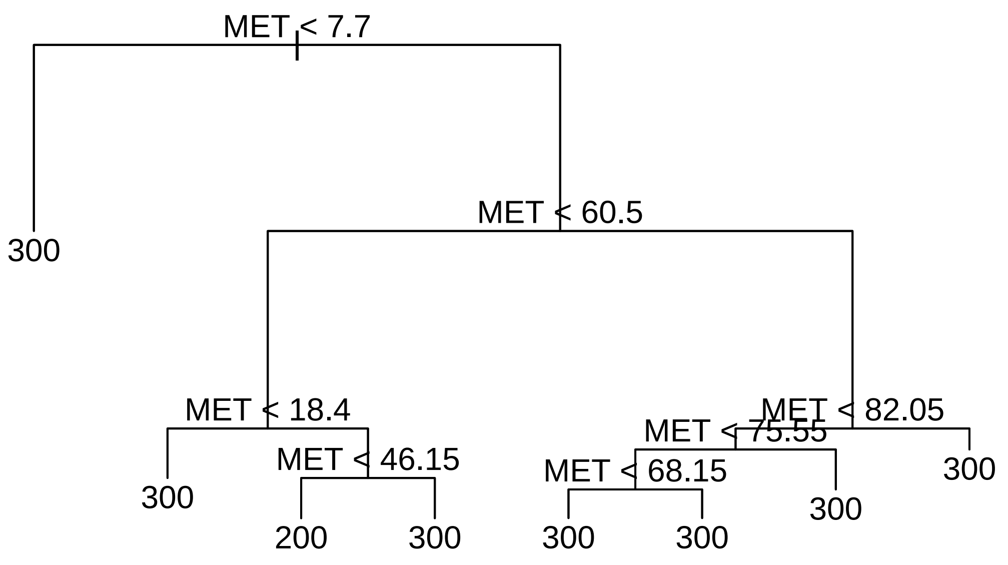


When we examine the data, it seems that there is no significant distribution. For this reason, the use of the tree algorithm can be an effective method for classifying the data.

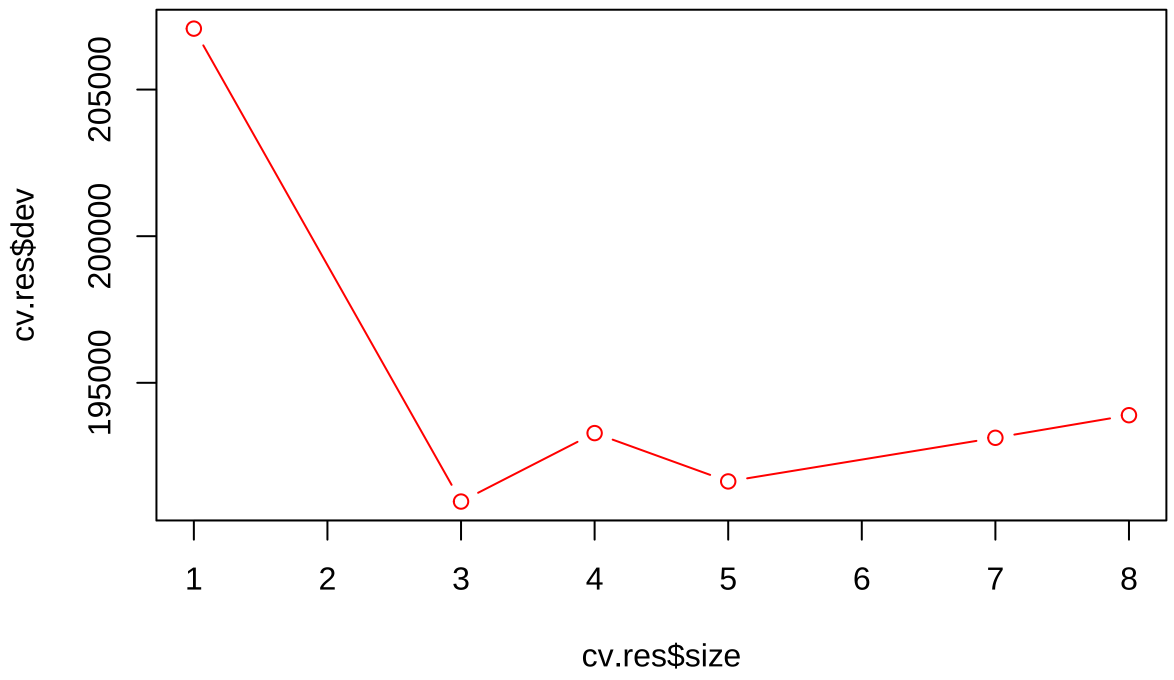
Part 2

In this part, regression tree is fitted with target EX and feature MET. Cross validation is used to increase model quality. Also, used the entire data set and set minimum number of observations in a leaf equal to 8.

```r
library(tree)

fit=tree(EX~MET, data=data,control = tree.control(nobs = 48,minsize = 8))
plot(fit)
text(fit)
```

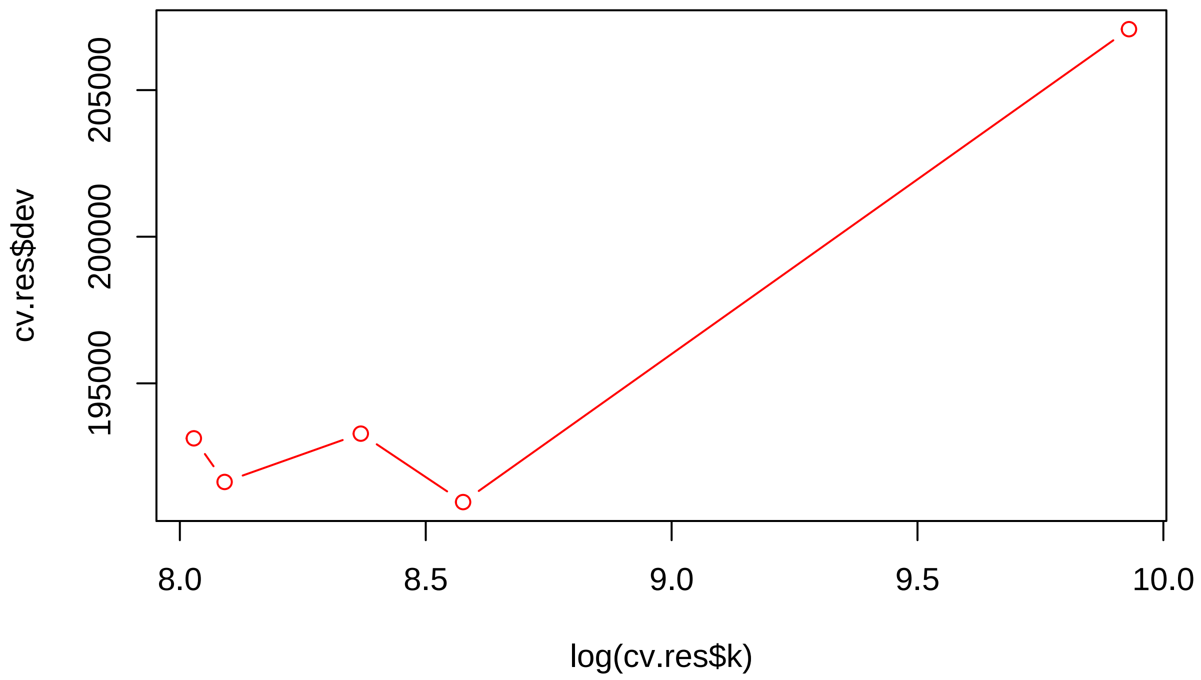


```r
set.seed(12345)
cv.res=cv.tree(fit)
plot(cv.res$size, cv.res$dev, type="b",
     col="red")
```



```r
plot(log(cv.res$k), cv.res$dev,
     type="b", col="red")
```

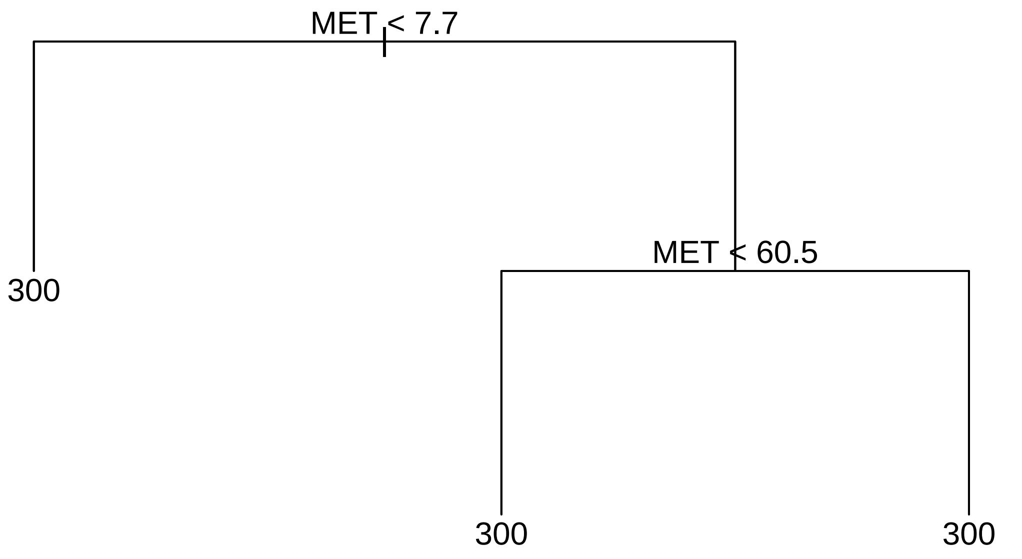
```
## Warning in log(cv.res$k): NaNs produced
```



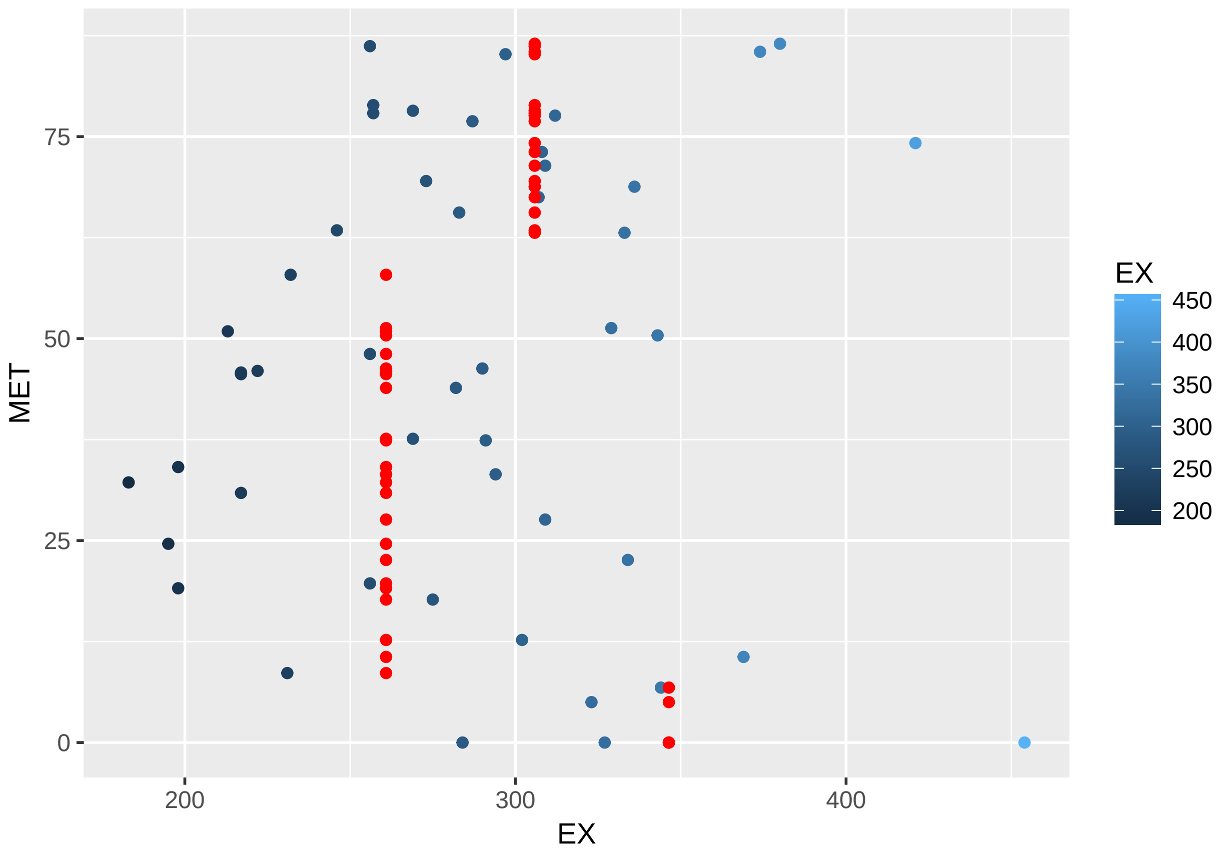
```r
minDev <- min(cv.res$dev)
index <- which(minDev == cv.res$dev)
fit <- tree(EX~MET, data=data)
best <- cv.res$size[index]
print(best)
```

```
## [1] 3
```

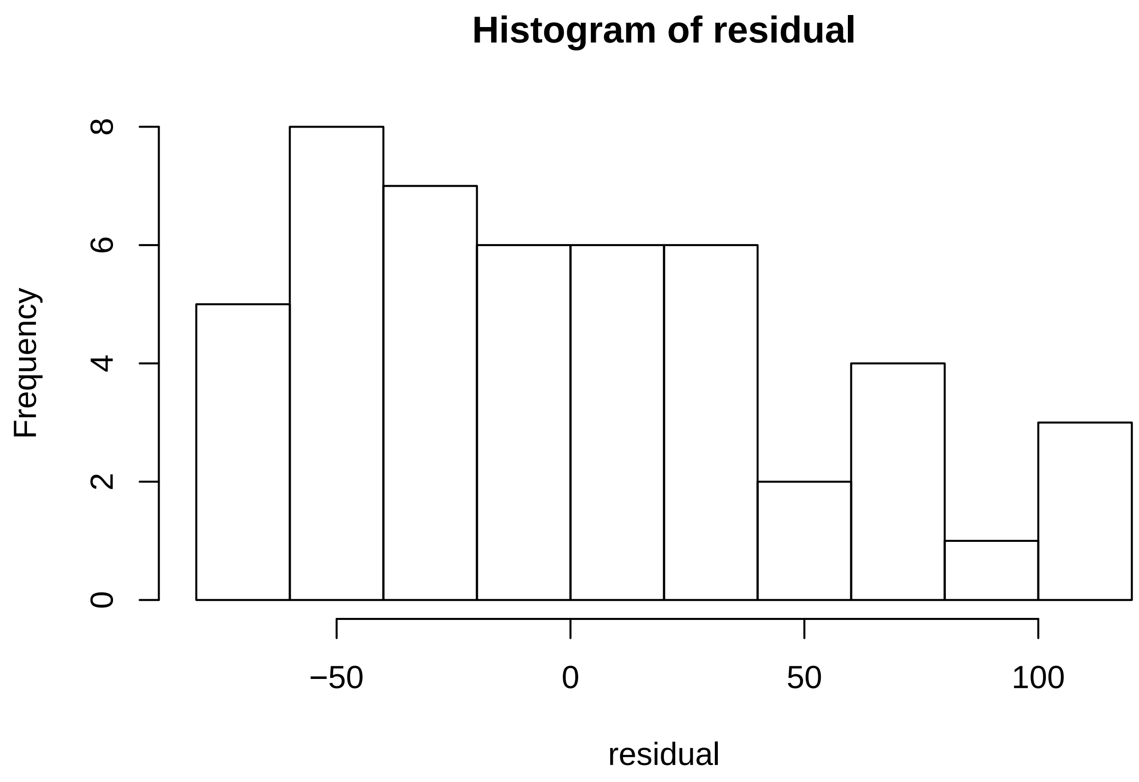
```r
prunedTree=prune.tree(fit,best=best)
plot(prunedTree)
text(prunedTree)
```



```r
pred=predict(prunedTree, newdata=data)
newData <- data.frame(EX = pred, MET = data$MET)

ggplot() + geom_point(data = data, aes(x=EX, y=MET,color = EX)) + geom_point(data = newData, colour = "
```



```
residual <- df$EX - pred
hist(residual)
```



The graph of the values obtained as a result of cross validation is drawn above. In addition, 3 was chosen as the best model size. We use this value to generate predictions We draw the classification predictions we have obtained and the data itself together graphically. As a result, the data set by the red dot is classified. We have seen that the data are classified in the direction of these points.

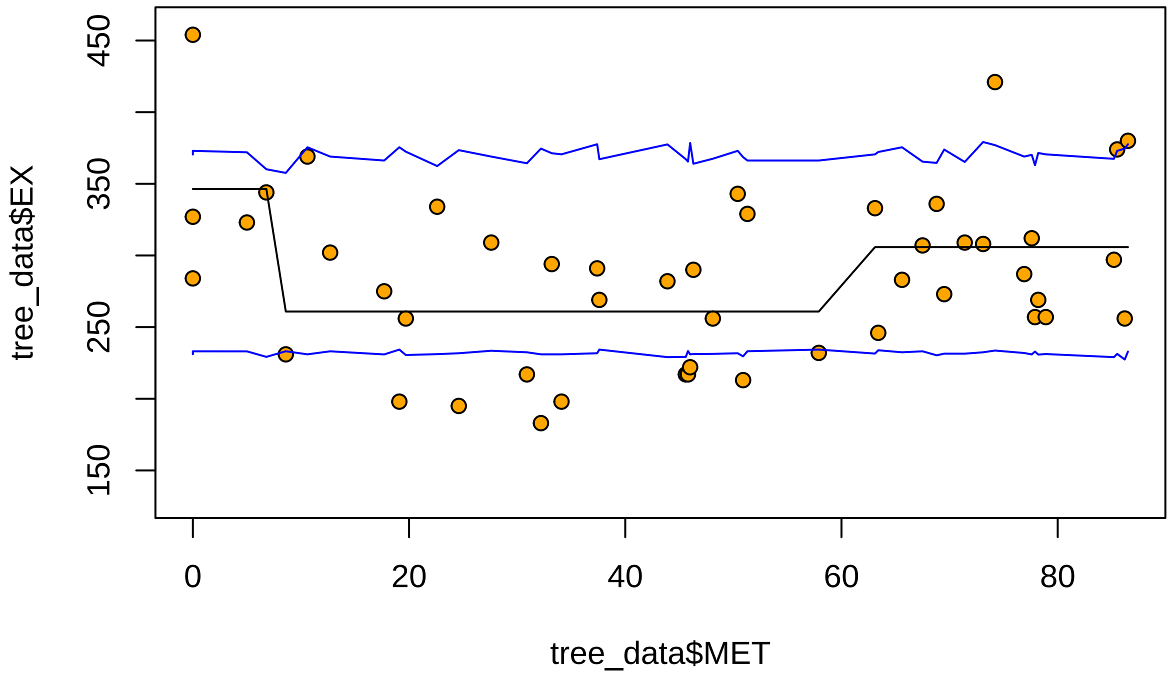
On the other hand, the residuals histogram plotted below seems to be centered around 0. That indicates the mean of residuals of this tree is around 0.

Part 3

```r
library(boot)

# computing bootstrap samples
f=function(data, ind){
  data1 <- data[ind,]# extract bootstrap sample
  fit=tree(EX~MET, data=data1,control = tree.control(nobs = 48,minsize = 8))
  res= prune.tree(fit, best = 3)
  pred=predict(res, newdata=data1)
  return(pred)
}
nonParametricBootstrap=boot(data, f, R=1000) #make bootstrap
```

```
## Warning in prune.tree(fit, best = 3): best is bigger than tree size
```

```r
e <- envelope(nonParametricBootstrap)


plot(tree_data$MET, tree_data$EX, pch=21, bg="orange",ylim=c(130, 460))
points(tree_data$MET,pred,type="l") #plot fitted line
#plot cofidence bands
points(tree_data$MET,e$point[2,], type="l", col="blue")
points(tree_data$MET,e$point[1,], type="l", col="blue")
```

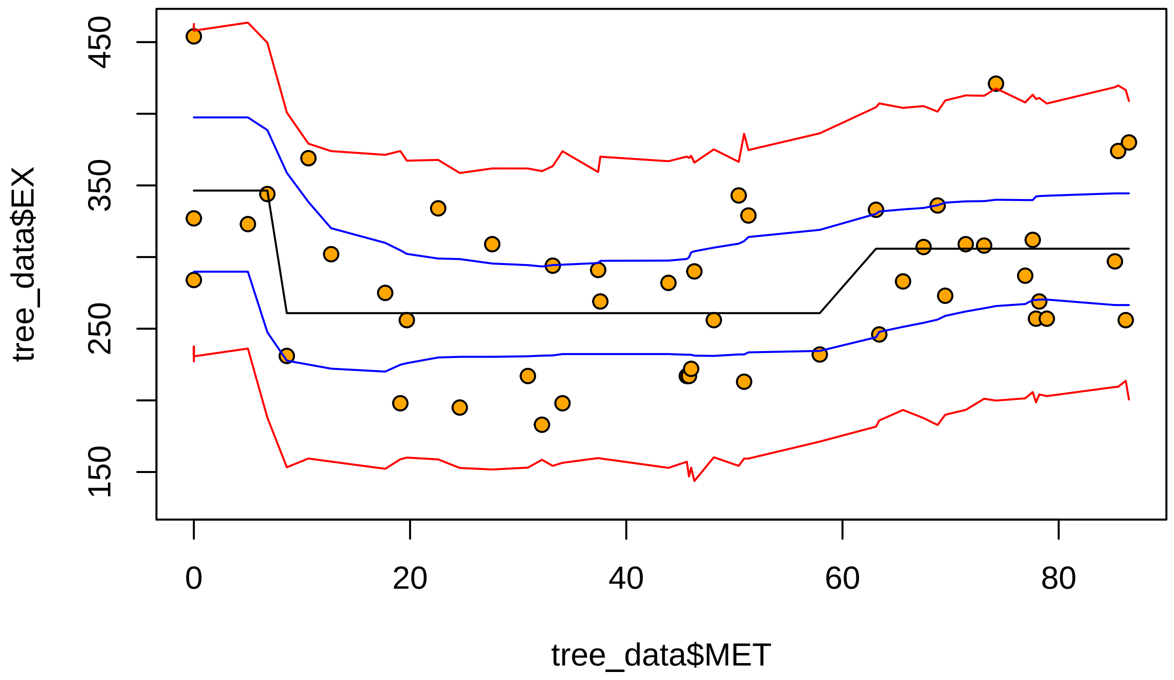


In the middle, decision boundary may be seen. Also, 95% confidence interval is drawn as upper and lower lines. The u pper line is more rocky than left one, and as seen high values for EX observations are not included. However, the lower line is quite smooth and does not include many observations for EX where EX observations are low. So, it is obvious that confidence bands include less than 95% of the observations according to graph.

Part 4

```
mle=tree(EX~MET, data=data,control = tree.control(nobs = 48,minsize = 8))
mle = prune.tree(mle, best = 3)
pred_mle = predict(mle, newdata=tree_data)
residuals_mle <- tree_data$EX - pred_mle

rng=function(data, mle) {

    data1=data.frame(MET=data$MET, EX=data$EX)
    n=length(data$MET)
    #generate new Price
    data1$EX=rnorm(n,predict(mle, newdata=data1),sd(residuals_mle))
    return(data1)
}


f=function(data){
    res_tree=tree(EX~MET, data=data, control = tree.control(nobs = length(data$MET),minsize = 8))
    # compute the tree
    res= prune.tree(res_tree, best = 3)
    #predict values for data
    predv=predict(res,newdata=tree_data)
    return(predv)
}
```

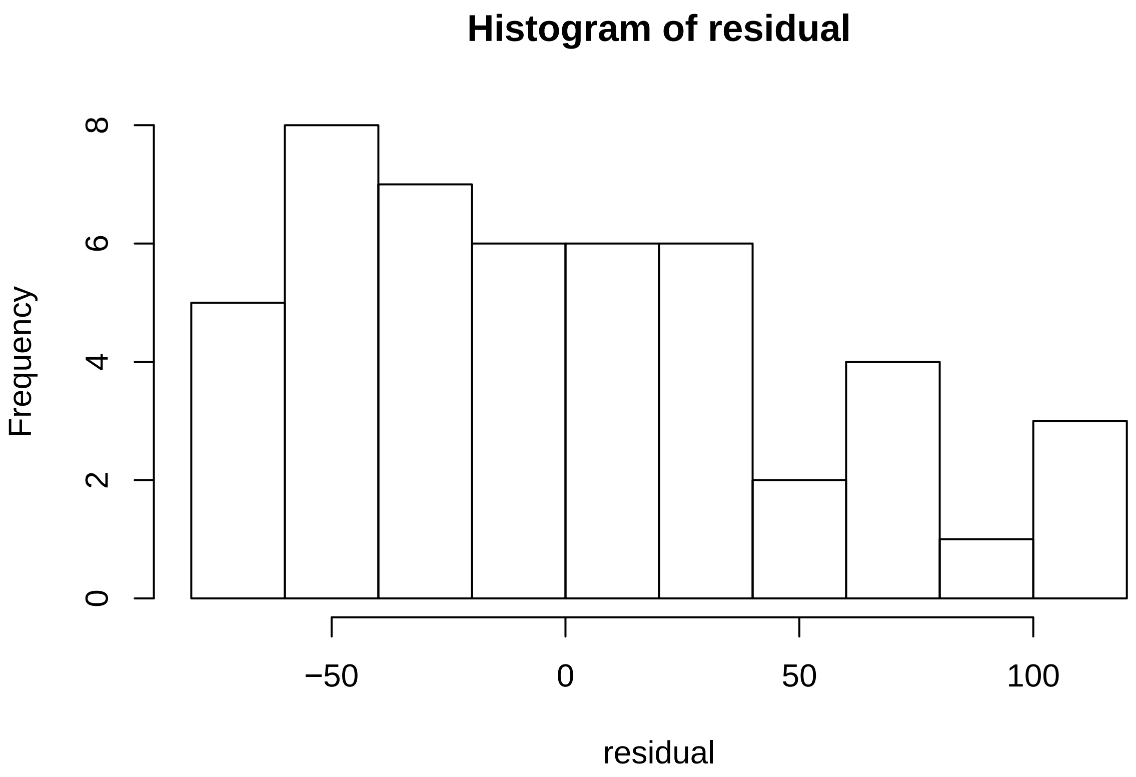
```r
paramboot1=boot(tree_data, statistic=f, R=1000,
         mle=mle,ran.gen=rng, sim="parametric")

e_confidence <- envelope(paramboot1)

f1=function(data){
    #res=tree(EX~MET, data=tree_data, control = tree.control(nobs = 48,minsize = 8)) #fit linear
    res_tree=tree(EX~MET, data=data, control = tree.control(nobs = length(data$MET),minsize = 8))
    # compute the tree
    res= prune.tree(res_tree, best = 3)
    #predict values for all Area values from the riginal data
    pred=predict(res,newdata=tree_data)
    n=length(tree_data$MET)
    predicted=rnorm(n,pred, sd(residuals_mle))
    return(predicted)
}

paramboot2=boot(tree_data, statistic=f1, R=1000,
         mle=mle,ran.gen=rng, sim="parametric")

e_prediction <- envelope(paramboot2)


plot(tree_data$MET, tree_data$EX, pch=21, bg="orange", ylim=c(130, 460))
points(tree_data$MET,e_confidence$point[2,], type="l", col="blue")
points(tree_data$MET,e_confidence$point[1,], type="l", col="blue")
points(tree_data$MET,e_prediction$point[2,], type="l", col="red")
points(tree_data$MET,e_prediction$point[1,], type="l", col="red")
points(tree_data$MET,pred_mle,type="l")
```



Wikipedia says: A confidence band is used in statistical analysis to represent the uncertainty in an estimate of a curve or function based on limited or noisy data. Similarly, a prediction band is used to represent the uncertainty about the value of a new data-point on the curve, but subject to noise.

The 95% prediction band (given by the red line) is wider than the 95% confidence band (given by the blue line). Unlike non-parametric bootstrap, parametric bootstrap result is wider and more smooth. As seen, confidence interval includes most of the points on the graph.

Part 5



The residuals is centered around 0 but it's hard to interpret due to so few observations. It wouldn't be wrong to assume that the residuals is approximate normal distributed with mean 0 and variance S^2 It is not recommended to perform an non-parametric bootstrap with sample size below 40, and in our case, the sample size is 48, so if the distribution could be assumed and a parametric bootstrap utilized, the results would improve.

Appendix

```
df <- read.csv("/home/ugur/git/ML_Lab2_Group/australian-crabs.csv",
               sep = ",")
library(ggplot2)

ggplot(df, aes(x=CL, y=RW, color=sex)) + geom_point(shape=1)
Y <- df$sex
X <- data.frame(RW = df$RW,CL = df$CL)
X <- as.matrix(X)

disc_fun=function(label, S){
  X1=X[Y==label,]
  means <- as.matrix(colMeans(X1))
  b1 <- -0.5 * t(means) %*% solve(S) %*% means + log(0.5)
  w1 <- solve(S) %*% means
  return(c(w1[1], w1[2], b1[1,1]))
}

X1=X[Y=="Male",]
X2=X[Y=="Female",]

S=cov(X1)*dim(X1)[1]+cov(X2)*dim(X2)[1]
S=S/dim(X)[1]

#discriminant function coefficients
res1=disc_fun("Male",S)
res2=disc_fun("Female",S)
res <- res1 - res2

# classification
d=res[1]*X[,1]+res[2]*X[,2]+res[3]
Yfit=(d>0)
plot(X[,2], X[,1], col=Yfit+1, xlab="CL", ylab="RW")
intercept <- -res[3]/res[1]
slope <- -res[2]/res[1]
abline(intercept,slope)
table(Yfit,df$sex)
glmResult <- glm(data = df,formula = as.factor(sex) ~ CL + RW , family = binomial())
coef <- as.matrix(glmResult$coefficients)
rwWeight <- coef[3]
clWeight <- coef[2]
inter <- coef[1] - 0.5
d=coef[3]*X[,1]+coef[2]*X[,2]+coef[1]
YfitGLM=(d>0.5)

plot(X[,2], X[,1], col=YfitGLM+1, xlab="CL", ylab="RW")
interceptGLM <- -inter/rwWeight
slopeGLM <- -(clWeight)/rwWeight
abline(interceptGLM,slopeGLM,col = "red")
abline(intercept,slope)
# comparison
# GLM
print(table(YfitGLM,df$sex))
# LDA
print(table(Yfit,df$sex))
library(gdata)
data <- read.xls("/home/ugur/git/ML_Lab2_Group/creditscoring.xls")

n=dim(data)[1]
set.seed(12345)
id=sample(1:n, floor(n*0.5))
train=data[id,]

rest=data[-id,]
```

```r
n=dim(rest)[1]
id=sample(1:n, floor(n*0.5))
test=rest[id,]
valid=rest[-id,]
library(tree)

confusion_matrix <- function(true_res, pred){
  tab <- table(true_res, pred)
  tab_plot <- addmargins(table(true_res, pred))
  miss <- 1 - sum(diag(tab))/sum(tab)
  l <- list(confusion_matrix = tab_plot,
  Missclassification_error = miss)
  return(l)
}

fitByGini=tree(good_bad~., data=train,method = "gini")
giniTrainPred <- predict(fitByGini, newdata=train,type="class")
giniTestPred <- predict(fitByGini, newdata=test,type="class")

misclassificationRate <- confusion_matrix(train$good_bad,giniTrainPred)
print(misclassificationRate)

misclassificationRate <- confusion_matrix(test$good_bad,giniTestPred)
print(misclassificationRate)
fitByDeviance=tree(good_bad~., data=train,method = "deviance")
devianceTrainPred <- predict(fitByDeviance, newdata=train,type="class")
devianceTestPred <- predict(fitByDeviance, newdata=test,type="class")

misclassificationRate <- confusion_matrix(train$good_bad,devianceTrainPred)
print(misclassificationRate)

misclassificationRate <- confusion_matrix(test$good_bad,devianceTestPred)
print(misclassificationRate)
fit=tree(good_bad~., data=train)
trainScore=rep(0,15)
testScore=rep(0,15)
for(i in 2:15) {
  prunedTree=prune.tree(fit,best=i) # i is tree depth here.
  pred=predict(prunedTree, newdata=valid,
               type="tree")
  trainScore[i]=deviance(prunedTree)
  testScore[i]=deviance(pred)
}

plot(2:15, trainScore[2:15], type="b", col="red",
     ylim=c(0,600))
points(2:15, testScore[2:15], type="b", col="blue")
minimumScore <- min(testScore[2:15])
bestDepth <- which(minimumScore == testScore)
print(bestDepth)

bestTree <- prune.tree(fit,best=bestDepth)
plot(bestTree)
```

```r
text(bestTree)

YpredTest <- predict(bestTree, newdata=test,type="class")
misclassificationRate <- confusion_matrix(test$good_bad,YpredTest)
print(misclassificationRate)
library(MASS)
library(e1071)
fit=naiveBayes(as.factor(good_bad)~., data=train)

Yfit=predict(fit, newdata=train)
YfitTest=predict(fit, newdata=test)

misclassificationRate <- confusion_matrix(train$good_bad,Yfit)
print(misclassificationRate)

misclassificationRate <- confusion_matrix(test$good_bad,YfitTest)
print(misclassificationRate)
# Fit Bayes Naive algorithm
fit_naive2=naiveBayes(as.factor(good_bad)~., data=train)
# Make raw predictions
YfitTrain=predict(fit_naive2, newdata=train, type= "raw")
# Classify data manually
Ypredict_final <- YfitTrain[,1]/YfitTrain[,2] <= 10

misclassificationRate <- confusion_matrix(train$good_bad,Ypredict_final)
print(misclassificationRate)


# Make raw predictions for test
YfitTest=predict(fit_naive2, newdata=test, type= "raw")
# Classify data manually
Ypredict_final <- YfitTest[,1]/YfitTest[,2] <= 10

misclassificationRate <- confusion_matrix(test$good_bad,Ypredict_final)
print(misclassificationRate)
df <- read.csv("/home/ugur/git/ML_Lab2_Group/State.csv",
               sep = ";")
library(dplyr)
options(digits=2)

# Replace , as . for R
df$MET <- gsub("\\,",".",df$MET)
# Change type as numeric from factor
df$MET <- as.numeric(df$MET)
df$EX <- as.numeric(df$EX)
# Reorder it
df <- arrange(df, MET)

tree_data <- data.frame(EX = df$EX, MET = df$MET)
data <- tree_data

library(ggplot2)
ggplot(data, aes(x=EX, y=MET)) + geom_point(shape=1)
```

```r
library(tree)

fit=tree(EX~MET, data=data,control = tree.control(nobs = 48,minsize = 8))
plot(fit)
text(fit)
set.seed(12345)
cv.res=cv.tree(fit)
plot(cv.res$size, cv.res$dev, type="b",
     col="red")
plot(log(cv.res$k), cv.res$dev,
     type="b", col="red")

minDev <- min(cv.res$dev)
index <- which(minDev == cv.res$dev)
fit <- tree(EX~MET, data=data)
best <- cv.res$size[index]
print(best)
prunedTree=prune.tree(fit,best=best)
plot(prunedTree)
text(prunedTree)

pred=predict(prunedTree, newdata=data)
newData <- data.frame(EX = pred, MET = data$MET)

ggplot() + geom_point(data = data, aes(x=EX, y=MET,color = EX)) + geom_point(data = newData, colour = "

residual <- df$EX - pred
hist(residual)

library(boot)

# computing bootstrap samples
f=function(data, ind){
  data1 <- data[ind,]# extract bootstrap sample
  fit=tree(EX~MET, data=data1,control = tree.control(nobs = 48,minsize = 8))
  res= prune.tree(fit, best = 3)
  pred=predict(res, newdata=data1)
  return(pred)
}
nonParametricBootstrap=boot(data, f, R=1000) #make bootstrap

e <- envelope(nonParametricBootstrap)


plot(tree_data$MET, tree_data$EX, pch=21, bg="orange",ylim=c(130, 460))
points(tree_data$MET,pred,type="l") #plot fitted line
#plot cofidence bands
points(tree_data$MET,e$point[2,], type="l", col="blue")
points(tree_data$MET,e$point[1,], type="l", col="blue")
mle=tree(EX~MET, data=data,control = tree.control(nobs = 48,minsize = 8))
mle = prune.tree(mle, best = 3)
pred_mle = predict(mle, newdata=tree_data)
residuals_mle <- tree_data$EX - pred_mle
```

```r
rng=function(data, mle) {

    data1=data.frame(MET=data$MET, EX=data$EX)
    n=length(data$MET)
    #generate new Price
    data1$EX=rnorm(n,predict(mle, newdata=data1),sd(residuals_mle))
    return(data1)
}


f=function(data){
    res_tree=tree(EX~MET, data=data, control = tree.control(nobs = length(data$MET),minsize = 8))
    # compute the tree
    res= prune.tree(res_tree, best = 3)
    #predict values for data
    predv=predict(res,newdata=tree_data)
    return(predv)
}

paramboot1=boot(tree_data, statistic=f, R=1000,
         mle=mle,ran.gen=rng, sim="parametric")

e_confidence <- envelope(paramboot1)

f1=function(data){
    #res=tree(EX~MET, data=tree_data, control = tree.control(nobs = 48,minsize = 8)) #fit linear
    res_tree=tree(EX~MET, data=data, control = tree.control(nobs = length(data$MET),minsize = 8))
    # compute the tree
    res= prune.tree(res_tree, best = 3)
    #predict values for all Area values from the riginal data
    pred=predict(res,newdata=tree_data)
    n=length(tree_data$MET)
    predicted=rnorm(n,pred, sd(residuals_mle))
    return(predicted)
}

paramboot2=boot(tree_data, statistic=f1, R=1000,
         mle=mle,ran.gen=rng, sim="parametric")

e_prediction <- envelope(paramboot2)


plot(tree_data$MET, tree_data$EX, pch=21, bg="orange", ylim=c(130, 460))
points(tree_data$MET,e_confidence$point[2,], type="l", col="blue")
points(tree_data$MET,e_confidence$point[1,], type="l", col="blue")
points(tree_data$MET,e_prediction$point[2,], type="l", col="red")
points(tree_data$MET,e_prediction$point[1,], type="l", col="red")
points(tree_data$MET,pred_mle,type="l")


hist(residual)
```