TDDE01: Machine Learning, LAB2

The dataset for the first assignment contains information from a private enterprise about their customers. The goal is to predict how well a customer will manage their loans based on a number of predictive features such as marital status, job, age etc. We first split the data into subsets of 50/25/25 of train/validation/test.

data = read.csv("assignment2/creditscoring.csv", header = TRUE,   
 sep = ",", quote = "\"", dec = ".",   
 fill = TRUE, comment.char = "")  
# use half data for training  
n = length(data[, 1])  
set.seed(12345)  
id = sample(1:n, floor(n \* 0.5))  
train = data[id, ]  
remainder = data[-id, ]  
  
# split the remaining data into validation/test  
n = length(remainder[, 1])  
id = sample(1:n, floor(n \* 0.5))  
validation = remainder[id, ]  
test = remainder[-id, ]

We then fit two models using either the gini or the deviance measures of impurity. The misclassification rates are reported and the model with the lowest rate is selected.

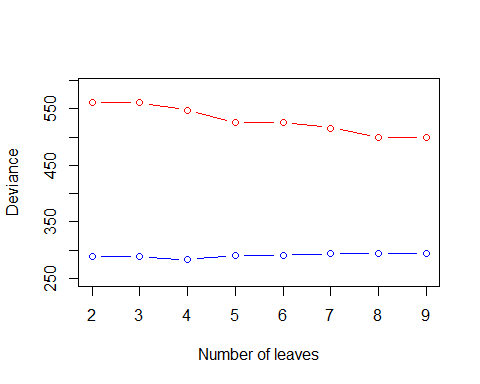
# fit one model using deviance (cross-entropy)  
model.deviance = tree(formula = good\_bad ~ .,   
 data = train,  
 split = "deviance")  
  
# and another using gini-index  
model.gini = tree(formula = good\_bad ~ .,   
 data = train,  
 split = "gini")  
  
get\_misclass\_rate=function(model, data)  
{  
 cm = get\_confusion\_matrix(model, data)  
 misclass\_rate = (cm[1,2] + cm[2,1]) / sum(cm)  
 return (misclass\_rate)  
}  
  
get\_confusion\_matrix=function(model, data)  
{  
 prediction = predict(model, newdata = data, type = "class")  
 cm = table(prediction, data$good\_bad)  
 return (cm)  
}  
  
cat("\nmisclass rate using 'deviance' for train data: ",   
 get\_misclass\_rate(model.deviance, train),  
 "\nmisclass rate using 'deviance' for test data: ",  
 get\_misclass\_rate(model.deviance, test),  
 "\nmisclass rate using 'gini' for train data: ",   
 get\_misclass\_rate(model.gini, train),  
 "\nmisclass rate using 'gini' for test data: ",  
 get\_misclass\_rate(model.gini, test))

##   
## misclass rate using 'deviance' for train data: 0.212   
## misclass rate using 'deviance' for test data: 0.284   
## misclass rate using 'gini' for train data: 0.23   
## misclass rate using 'gini' for test data: 0.34

Since deviance had the lower misclassification rates, this model is chosen for the subsequent steps.

Next we use the training and validation sets to choose the optimal tree depth. This is done by measuring the deviance for when pruning the tree to have a set number of terminal nodes (leaves) in the range 2-9.

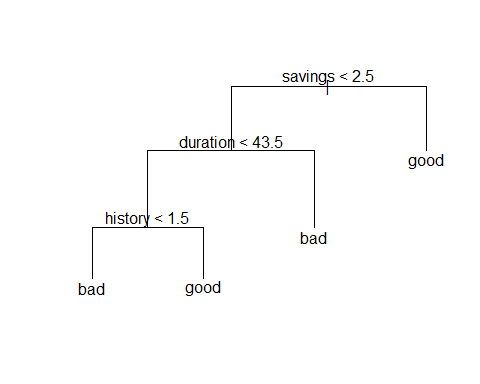
train\_score = numeric(9)  
validation\_score = numeric(9)  
  
# try number of terminal nodes in range 2-9 to find  
# optimal depth, i.e tree with lowest deviance  
for (i in 2:9)  
{  
 pruned\_tree = prune.tree(model.deviance, best = i)  
 prediction = predict(pruned\_tree, newdata = validation, type = "tree")  
 train\_score[i] = deviance(pruned\_tree)  
 validation\_score[i] = deviance(prediction)  
}  
  
plot(2:9, train\_score[2:9],   
 type="b", col="red", ylim=c(250, 590),  
 xlab="Number of leaves", ylab="Deviance")  
  
points(2:9, validation\_score[2:9],   
 type="b", col="blue")



# since i = 2..9, optimal i is equal to index + 1  
optimal.leaves = which.min(validation\_score[2:9]) + 1  
# prune tree to optimal number of leaves  
optimal.tree = prune.tree(model.deviance, best = optimal.leaves)  
# get variables selected of optimal tree  
summary(optimal.tree)

##   
## Classification tree:  
## snip.tree(tree = model.deviance, nodes = c(5L, 3L, 9L))  
## Variables actually used in tree construction:  
## [1] "savings" "duration" "history"   
## Number of terminal nodes: 4   
## Residual mean deviance: 1.117 = 547.5 / 490   
## Misclassification error rate: 0.251 = 124 / 494

# Variables actually used in tree construction:  
# "savings" "duration" "history"   
  
# print the optimal tree  
# depth = 3 (if root depth = 0)  
plot(optimal.tree)  
text(optimal.tree)



# get misclassification rate for optimal tree on test data  
optimal.misclass = get\_misclass\_rate(optimal.tree, test)  
cat("\nmisclass rate of optimal tree:", optimal.misclass)

##   
## misclass rate of optimal tree: 0.26

The optimal tree has 4 leaves, with a depth of 3. The variables selected by the tree was Savings, Duration, and History, with the tree splitting for different values of these variables to do make predictions.

Next we use the training data to perform classification using a Naive Bayes classifier, measuring its misclassification rates and confusion matrices on the test and traninig data.

# fit a naive bayes model to train data  
model.bayes = naiveBayes(formula = good\_bad ~ .,   
 data = train)  
  
# misclass-rate is higher for train than test..  
cat("\nmisclass rate for naive bayes model on train data:",  
 get\_misclass\_rate(model.bayes, train),  
 "\nmisclass rate for naive bayes model on test data:",  
 get\_misclass\_rate(model.bayes, test))

##   
## misclass rate for naive bayes model on train data: 0.3   
## misclass rate for naive bayes model on test data: 0.32

cat("\n\nCM for naiveBayes (train):")

##   
##   
## CM for naiveBayes (train):

print(get\_confusion\_matrix(model.bayes, train))

##   
## prediction bad good  
## bad 95 98  
## good 52 255

cat("\nCM for naiveBayes (test):")

##   
## CM for naiveBayes (test):

print(get\_confusion\_matrix(model.bayes, test))

##   
## prediction bad good  
## bad 47 49  
## good 31 123

From the results we can see that the Naive Bayes classifier has a higher misclassification rate than our optimal tree model.

For the final task we use a modified loss matrix for our naive bayes classifier: ( picture of loss matrix here )

To calculate the new misclassification rate and confusion matrix using this loss matrix, we get the raw probabilities for each class from our classifier, and apply the new classification policy to our probabilities.

# get raw probabilities for both classes from naiveBayes classifier  
raw.train = predict(model.bayes, newdata = train, type = "raw")  
raw.test = predict(model.bayes, newdata = test, type = "raw")  
  
# predicting 'good' must be 10x more probable than bad  
# to make the prediction 'good' with new loss-matrix   
preds.train = (raw.train[, 2] / raw.train[, 1]) > 10  
preds.test = (raw.test[, 2] / raw.test[, 1]) > 10  
  
# convert booleans to good/bad labels  
preds.train[which(preds.train == TRUE)] = "good"  
preds.train[which(preds.train == FALSE)] = "bad"  
preds.test[which(preds.test == TRUE)] = "good"  
preds.test[which(preds.test == FALSE)] = "bad"  
  
# CM for train and test  
cm.train = table(preds.train, train$good\_bad)  
cm.test = table(preds.test, test$good\_bad)  
  
cat("\nCM for naiveBayes + new loss matrix (train):")

##   
## CM for naiveBayes + new loss matrix (train):

print(cm.train)

##   
## preds.train bad good  
## bad 137 263  
## good 10 90

cat("\nCM for naiveBayes + new loss matrix (test):")

##   
## CM for naiveBayes + new loss matrix (test):

print(cm.test)

##   
## preds.test bad good  
## bad 70 131  
## good 8 41

# misclass-rate for train and test  
misclass.train = (cm.train[1,2] + cm.train[2,1]) / sum(cm.train)  
misclass.test = (cm.test[1,2] + cm.test[2,1]) / sum(cm.test)  
  
cat("\nmisclass rate naiveBayes + new loss matrix (train data):",  
 misclass.train,  
 "\nmisclass rate naiveBayes + new loss matrix (test data):",  
 misclass.test)

##   
## misclass rate naiveBayes + new loss matrix (train data): 0.546   
## misclass rate naiveBayes + new loss matrix (test data): 0.556

As we can see our misclassification rates have increased from the previous model, but our rates have lowered for classifying a customers future loan management as bad when the true classification is good, which was the point of the modified loss matrix.