Assignment 4, Part II Work

Matthew Dunne

August 1, 2018

Data

First we load the dataset (the same one we used in Part I), convert the necessary variables into factors and split then split into train and test data.

```
setwd("C:/Users/mjdun/Desktop/Data Mining/Assignments")
#using csv which has it in factor variables
MyData <- read.csv(file="German.Credit.csv", header=TRUE, sep=",")
columns<-c(1,2,4,5,7,8,9,10,11,12,13,15,16,17,18,19,20,21)
MyData[columns] <- lapply(MyData[columns], factor)
```

Split into train and test exactly as we did in Part I (same seed).

```
library(caret)
```

```
## Loading required package: lattice
```

```
## Loading required package: ggplot2
```

```
#split data into train and test
set.seed(1234)
s1<-sample(1:nrow(MyData), nrow(MyData)*.7, replace=FALSE)
```

Build a Classification Tree on the Training Data

Build a full classification tree on the training data to predict the Creditability (Class) variable. Set the minimum node size=30, use 10-fold cross validation (Xval=10), and use the full tree (cp=0).

```
library(rpart)
#will use first variable as response variable automatically
whole_tree=rpart(MyData[s1, ],control=rpart.control(cp=0,minsplit=30,xval=10, maxsurrogate=0))
```

Evaluate the Complexity Parameter Plots and Prints

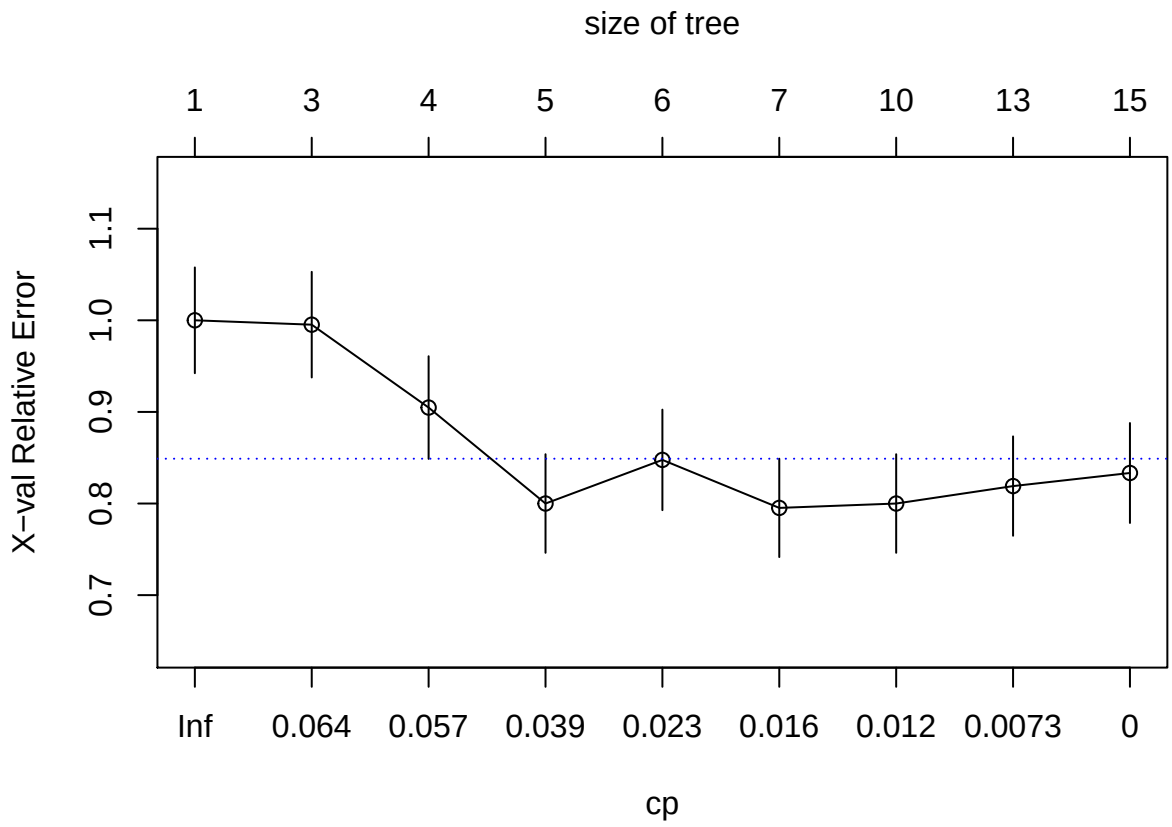
Here are the cp values from our model and the plot thereof.

```
printcp(whole_tree)
```

```
##
## Classification tree:
## rpart(formula = MyData[s1, ], control = rpart.control(cp = 0,
##     minsplit = 30, xval = 10, maxsurrogate = 0))
##
## Variables actually used in tree construction:
## [1] Account.Balance                   Concurrent.Credits
## [3] Credit.Amount                     Duration.in.Current.address
## [5] Duration.of.Credit..month.        Occupation
## [7] Payment.Status.of.Previous.Credit Purpose
```

```
## [9] Value.Savings.Stocks
##
## Root node error: 210/700 = 0.3
##
## n= 700
##
##          CP nsplit rel error  xerror     xstd
## 1 0.0666667      0   1.00000 1.00000 0.057735
## 2 0.0619048      2   0.86667 0.99524 0.057656
## 3 0.0523810      3   0.80476 0.90476 0.056027
## 4 0.0285714      4   0.75238 0.80000 0.053807
## 5 0.0190476      5   0.72381 0.84762 0.054863
## 6 0.0126984      6   0.70476 0.79524 0.053697
## 7 0.0111111      9   0.66667 0.80000 0.053807
## 8 0.0047619     12   0.63333 0.81905 0.054239
## 9 0.0000000     14   0.62381 0.83333 0.054554
```

```
plotcp(whole_tree,minline=TRUE, col=4)
```



The Cross Validation Error (xerror) is lowest on 6 splits (or 7 nodes). We see this validated in the cp plot. We will use the corresponding CP value, 0.0126984, to build a pruned tree on the training data.

```
#will use first variable as response variable automatically
pruned_tree=rpart(MyData[s1, ],control=rpart.control(cp=0.0126984,minsplit=30,xval=10, maxsurrogate=0))
```

Confusion Matrix of Predictions

Now we will generate a confusion matrix using this pruned tree of actual Bad and Good vs predicted Bad and Good. This is still for the training data.

```
table(MyData[s1, 1],predict(pruned_tree,type="class"))
```

```
##
##       0   1
##   0 103 107
##   1  26 464
```

We see we are predicting a little less than half of the Bad's correctly, but the vast majority of the Good's correctly. Here are the percentages.

```
round(prop.table(table(MyData[s1, 1],predict(pruned_tree,type="class")),1),2)
```

```
##
##        0    1
##   0 0.49 0.51
##   1 0.05 0.95
```

We might think of this another way: When we guess 0 or 1 how often are we right? The table below shows that when we guess 0, we are right 80% of the time, and when we guess 1, we are right 81% of the time.

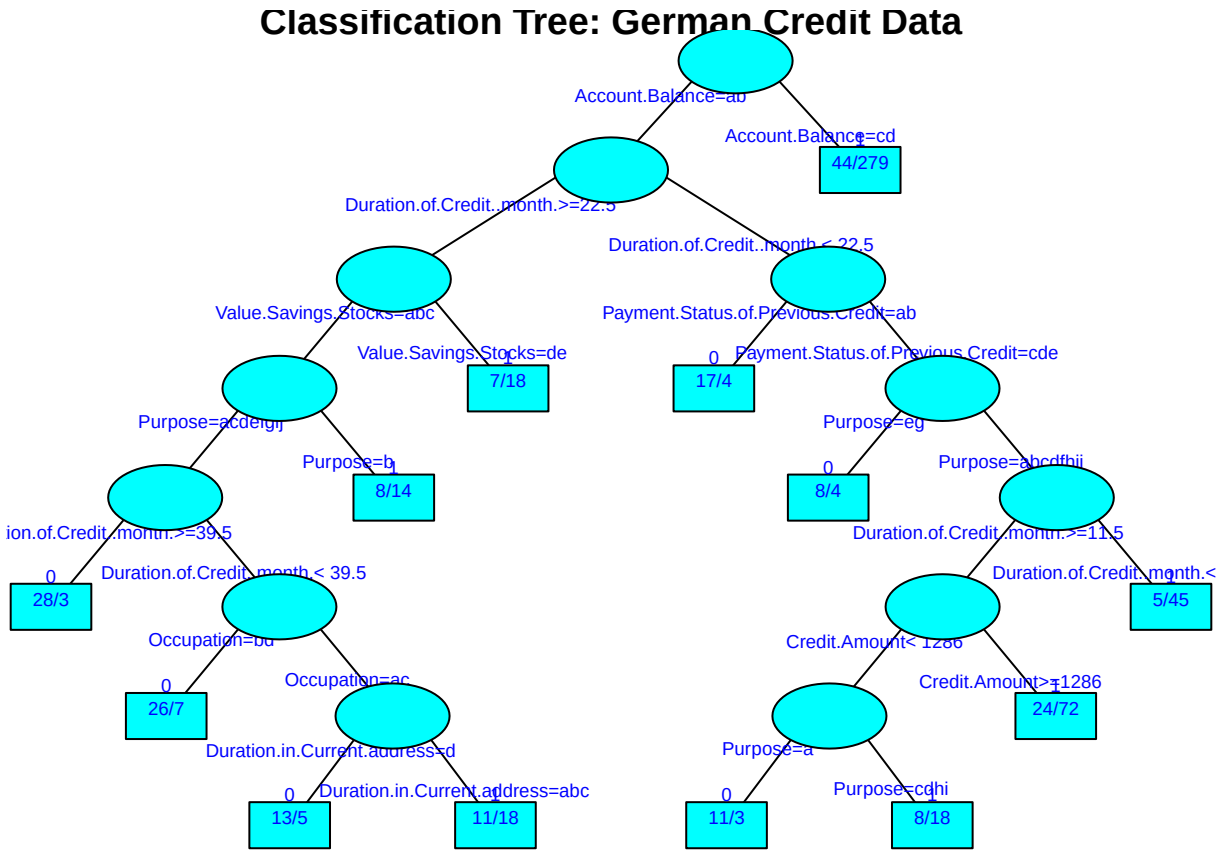
```
round(prop.table(table(MyData[s1, 1],predict(pruned_tree,type="class")),2),2)
```

```
##
##        0    1
##   0 0.80 0.19
##   1 0.20 0.81
```

Interpreting a Graph of the Tree

To further interpret the tree it may help to actually look at it.

```
par(mai=c(0.1,0.1,0.1,0.1))
plot(pruned_tree,main="Classification Tree: German Credit Data",col=3, compress=TRUE, branch=0.2,unifor
text(pruned_tree,cex=0.6,col=4,use.n=TRUE,fancy=TRUE,fwidth=0.4,fheight=0.4,bg=c(5))
```



The tree does provide some insight into the relative importance of variables and the interaction between them. The most important variable is Account Balance. When it is over a certain level, 279 Goods would be correctly rated as Good and 44 Bads incorrectly rated as Good. After that the model looks at Duration of Credit. Interestingly, after that split we see the variables in different order of importance. Furthermore we see that Duration of Credit reoccurs as a variable on which to split.

Also of note is that for Credit Amounts above a certain balance there are no interactions - the split results in a terminal node. It is only for the Amounts below a certain balance that we see the complicated interactions. We also see that each split (after the second) results in a terminal node.

Holdout Validation Testing

How does the pruned tree model work on our test data. Let us build a confusion matrix to check.

```
#make sure to use model on training data
table(MyData[-s1, 1],predict(pruned_tree, newdata=MyData[-s1, -1], type="class"))
```

```
##
##       0   1
##   0  32  58
##   1  25 185
```

```
round(prop.table(table(MyData[-s1, 1],predict(pruned_tree, newdata=MyData[-s1, -1],type="class")),1),2)
```

```
##
##        0    1
##   0 0.36 0.64
```

```
##   1 0.12 0.88
```

Clearly it does not do as well, relative to the training data, as it correctly predicting only 88% of Good ratings and a dismal 36% of Bad ratings.

Summary of Results

How does this model compare with our logistic regression model from Part I? Not very well. The tree model only catches 36% of the Bad ratings on our test data. This is compared to 61% for the logistic regression model. The tree model might be preferable if our goal was to maximize the percentage of detections of Good. In that respect it performs more favorably on the test data than logistic regression.