Data Mining - Assignment 3: Part 1

Ryan Durfey

Sunday, February 22, 2015

```
library(caret)
library(knitr)
```

Data Preparation

```
set.seed(8675309)
data(GermanCredit)
GC<-GermanCredit

## Creation of training indices to encompass 700 observations of the data.
## These will be used to subset the data into train and holdout samples.
train_indices<-sample(1:nrow(GC),as.integer(nrow(GC)*.7),replace=FALSE)
length(train_indices)
```

```
## [1] 700
```

```
head(train_indices)
```

```
## [1] 160 478 764 768 268 670
```

```
###########################################
## function to create factor variables ##
###########################################
combine.vars<-function(data=GC, cols){
        as.factor(apply((as.matrix(data[,cols],ncol=length(cols),byrow=TRUE)
               %*% matrix(seq_along(cols))),1,max))
}

######################################################################
## Create single-vector, combined categorical variables with levels ##
######################################################################
Telephone<-combine.vars(data=GC,cols=8)
ForeignWorker<-combine.vars(data=GC,cols=9)
CheckingAccountStatus<-combine.vars(data=GC,cols=11:14)
CreditHistory<-combine.vars(data=GC,cols=15:19)
Purpose<-combine.vars(data=GC,cols=20:30)
SavingsAccountsBonds<-combine.vars(data=GC,cols=31:35)
EmploymentDuration<-combine.vars(data=GC,cols=36:40)
PersonalStatus<-combine.vars(data=GC,cols=41:45)
OtherDebtorsGuarantors<-combine.vars(data=GC,cols=46:48)
Property<-combine.vars(data=GC,cols=49:52)
OtherInstallmentPlans<-combine.vars(data=GC,cols=53:55)
Housing<-combine.vars(data=GC,cols=56:58)
Job<-combine.vars(data=GC,cols=59:62)
```

```
##########################################################################
## Recreate Dataset with factor variables and Class in the first column ##
##########################################################################
GC_2<-cbind(Class=GC[,10],GC[,1:7],Telephone,ForeignWorker,CheckingAccountStatus,CreditHistory,Purpose,S

############################################################################
## Creation and saving of training and test/holdout samples using indices ##
############################################################################
GC_tr<-GC_2[train_indices,]
GC_te<-GC_2[-train_indices,]

## Save training and holdout samples for future use
saveRDS(GC_tr,"C:/Misc Docs/UChicago/DataMining/Data_Mining/GC_train.rds")
saveRDS(GC_te,"C:/Misc Docs/UChicago/DataMining/Data_Mining/GC_test.rds")
```

Logistic Regression Model with Training Dataset

First, we can fit the null and full models, using none and all of the variable, respectively, to gain some insight into what we are working with and to get a baseline. We'll output the AIC for both models so that we can then compare them to subsequent models.

```
## Null model; observe AIC
model.null<-model.full<-glm(GC_tr$Class~1,family=binomial(link=logit))
summary(model.null)$aic
```

```
## [1] 865.5137
```

```
## Full model; observe AIC
model.full<-glm(GC_tr,family=binomial(link=logit))
summary(model.full)$aic
```

```
## [1] 709.978
```

According the AIC, the full model is largely better than the null model, though this does not come as a surprise. Next, using the drunction drop1(), we can discover which variables are good candidates to drop to help lower the AIC.

```
## Use drop1 to determine variables to drop that would lower AIC
drop.1<-drop1(model.full,test="Chisq")
drop.1
```

```
## Single term deletions
##
## Model:
## Class ~ Duration + Amount + InstallmentRatePercentage + ResidenceDuration +
##     Age + NumberExistingCredits + NumberPeopleMaintenance + Telephone +
##     ForeignWorker + CheckingAccountStatus + CreditHistory + Purpose +
##     SavingsAccountsBonds + EmploymentDuration + PersonalStatus +
##     OtherDebtorsGuarantors + Property + OtherInstallmentPlans +
##     Housing + Job
##                            Df Deviance    AIC    LRT  Pr(>Chi)
## <none>                          611.98 709.98
## Duration                    1   614.29 710.29  2.317 0.1279679
## Amount                      1   619.37 715.37  7.389 0.0065610 **
## InstallmentRatePercentage   1   624.75 720.75 12.777 0.0003509 ***
## ResidenceDuration           1   612.14 708.14  0.158 0.6906787
## Age                         1   613.92 709.92  1.942 0.1634541
## NumberExistingCredits       1   614.08 710.08  2.107 0.1466328
## NumberPeopleMaintenance     1   613.90 709.90  1.921 0.1657666
## Telephone                   1   613.65 709.65  1.668 0.1965262
## ForeignWorker               1   613.93 709.93  1.951 0.1624417
## CheckingAccountStatus       3   662.88 754.88 50.905 5.125e-11 ***
## CreditHistory               4   635.20 725.20 23.217 0.0001146 ***
## Purpose                     9   637.33 717.33 25.351 0.0026068 **
## SavingsAccountsBonds        4   618.85 708.85  6.873 0.1427680
## EmploymentDuration          4   620.73 710.73  8.756 0.0674945 .
## PersonalStatus              3   620.63 712.63  8.653 0.0342723 *
## OtherDebtorsGuarantors      2   619.97 713.97  7.989 0.0184162 *
## Property                    3   614.61 706.61  2.629 0.4524205
## OtherInstallmentPlans       2   613.90 707.90  1.917 0.3834276
## Housing                     2   614.29 708.29  2.316 0.3140979
## Job                         3   614.70 706.70  2.721 0.4367171
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
```

Here, we see a number of variables for which removing them would lower the AIC below the 709.978 we got from the full model. So, we can rerun glm() after removing these variables to get a model that should have a better AIC and thus be a better fit.

```
## Candidate variables to remove
sig.names<-c("ResidenceDuration","Age","NumberPeopleMaintenance","Telephone",
             "ForeignWorker","SavingsAccountsBonds","Property",
             "OtherInstallmentPlans","Housing","Job")

## Create new model without those variables
model.drop<-glm(GC_tr[,!names(GC_tr)%in%sig.names],family=binomial(link=logit))
summary(model.drop)$aic
```

```
## [1] 699.9285
```

Ah-HA! This AIC is indeed significantly lower than what we observed with the full model. It is now our leading candidate for best model.

To double check that there are no additional variables that we can drop to lower the AIC even further, we can run drop() on this new model.

```
## run drop1 again to see if there are any other variables we can drop to get a lower AIC
drop1(model.drop,test="Chisq")
```

```
## Single term deletions
##
## Model:
## Class ~ Duration + Amount + InstallmentRatePercentage + NumberExistingCredits +
##     CheckingAccountStatus + CreditHistory + Purpose + EmploymentDuration +
##     PersonalStatus + OtherDebtorsGuarantors
##                          Df Deviance    AIC    LRT  Pr(>Chi)
## <none>                       639.93 699.93
## Duration                  1   646.27 704.27  6.339 0.0118082 *
## Amount                    1   645.75 703.75  5.821 0.0158334 *
## InstallmentRatePercentage  1   651.23 709.23 11.304 0.0007732 ***
## NumberExistingCredits     1   642.76 700.76  2.829 0.0925681 .
## CheckingAccountStatus     3   708.59 762.59 68.658 8.270e-15 ***
## CreditHistory             4   671.12 723.12 31.192 2.797e-06 ***
## Purpose                   9   662.96 704.96 23.036 0.0061151 **
## EmploymentDuration        4   651.08 703.08 11.153 0.0248986 *
## PersonalStatus            3   648.07 702.07  8.144 0.0431226 *
## OtherDebtorsGuarantors    2   647.99 703.99  8.059 0.0177820 *
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
```

Nope, none of the remaining variables are candidates to be dropped. Thus, this is the model we are content with. We'll save it for any future use and output the summary.

```
## output summary of model
summary(model.drop)
```

```
##
## Call:
## glm(formula = GC_tr[, !names(GC_tr) %in% sig.names], family = binomial(link = logit))
##
## Deviance Residuals:
##     Min       1Q   Median       3Q      Max
## -2.7440  -0.7488   0.3938   0.7132   2.1063
##
## Coefficients:
##                             Estimate Std. Error z value Pr(>|z|)
## (Intercept)               -5.033e-02  8.468e-01  -0.059 0.952608
## Duration                  -2.878e-02  1.146e-02  -2.512 0.011998 *
## Amount                    -1.299e-04  5.412e-05  -2.401 0.016354 *
## InstallmentRatePercentage -3.367e-01  1.022e-01  -3.296 0.000979 ***
## NumberExistingCredits     -3.489e-01  2.075e-01  -1.681 0.092743 .
## CheckingAccountStatus2     5.071e-01  2.494e-01   2.033 0.042040 *
## CheckingAccountStatus3     1.185e+00  4.317e-01   2.746 0.006028 **
## CheckingAccountStatus4     1.991e+00  2.657e-01   7.495 6.65e-14 ***
## CreditHistory2             6.345e-02  6.504e-01   0.098 0.922288
## CreditHistory3             9.984e-01  5.254e-01   1.900 0.057403 .
## CreditHistory4             1.022e+00  5.913e-01   1.729 0.083886 .
## CreditHistory5             2.150e+00  5.412e-01   3.972 7.12e-05 ***
## Purpose2                   1.442e+00  4.447e-01   3.243 0.001184 **
## Purpose3                   5.082e-01  3.015e-01   1.685 0.091900 .
## Purpose4                   5.563e-01  2.847e-01   1.954 0.050729 .
## Purpose5                   2.610e-01  8.577e-01   0.304 0.760855
## Purpose6                   1.929e-01  6.360e-01   0.303 0.761678
## Purpose7                  -4.173e-01  4.666e-01  -0.894 0.371220
## Purpose9                   1.529e+01  4.929e+02   0.031 0.975258
## Purpose10                  7.444e-01  3.909e-01   1.904 0.056872 .
## Purpose11                  2.450e-01  9.636e-01   0.254 0.799301
## EmploymentDuration2        1.662e-01  2.794e-01   0.595 0.552019
## EmploymentDuration3        9.686e-01  3.479e-01   2.784 0.005375 **
## EmploymentDuration4        4.318e-01  3.128e-01   1.381 0.167379
## EmploymentDuration5       -1.798e-01  4.502e-01  -0.399 0.689648
## PersonalStatus2            3.804e-01  4.491e-01   0.847 0.396975
## PersonalStatus3            9.321e-01  4.435e-01   2.102 0.035561 *
## PersonalStatus4            5.037e-01  5.182e-01   0.972 0.331067
## OtherDebtorsGuarantors2   -8.757e-01  4.247e-01  -2.062 0.039211 *
## OtherDebtorsGuarantors3    8.542e-01  4.928e-01   1.733 0.083063 .
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## (Dispersion parameter for binomial family taken to be 1)
##
##     Null deviance: 863.51  on 699  degrees of freedom
## Residual deviance: 639.93  on 670  degrees of freedom
## AIC: 699.93
##
## Number of Fisher Scoring iterations: 14
```

```
## save model for future use
saveRDS(model.drop,"C:/Misc Docs/UChicago/DataMining/Data_Mining/model_drop.rds")
```

Confusion Matrices and Holdout

Lastly, we look at the confusion matrices and complete the holdout validation to determine how well our model predicts the Good/Bad status of the dependent variable, Class. We'll first use the demarcation of 0.5 to interpret the fitted values as Bad (also noted as 0) and Good (also noted as 1).

```
## Confusion Matrix for Training dataset
mdp<-model.drop$fitted.values
mdp[mdp>=0.5]<-1
mdp[mdp<0.5]<-0
table(GC_tr$Class,mdp)
```

```
##       mdp
##          0   1
##   Bad  116  99
##   Good  50 435
```

```
round(prop.table(table(GC_tr$Class,mdp),1),2)
```

```
##       mdp
##           0    1
##   Bad  0.54 0.46
##   Good 0.10 0.90
```

```
## Confusion Matrix for Holdout dataset
mdp2<-predict(model.drop,newdata=GC_te[,-1],type="response")
mdp2[mdp2>=0.5]<-1
mdp2[mdp2<0.5]<-0
table(GC_te$Class,mdp2)
```

```
##       mdp2
##          0   1
##   Bad   45  40
##   Good  38 177
```

```
round(prop.table(table(GC_te$Class,mdp2),1),2)
```

```
##       mdp2
##           0    1
##   Bad  0.53 0.47
##   Good 0.18 0.82
```

From the confusion matrices for the training set, we see that our model correctly predicts "Bad" outputs 54% of the time and "Good" outputs 90% of the time. Similarly, in the holdout set, it predicted "Bad" 53% and "Good" 82%. Ideally, we would like our model to predict both outcomes 100% of the time, but that isn't realistic. Our model can be considered decent at predicting "Good" outcomes, but poor with predicting the "Bad". If correctly predicting the Good is all we care about, then we could move forward with this model.

However, if we are more concerned with predicting "Bad" (which may be the case since in this example we are juding credit worthiness), we may not be satisfied with this. To help fix this issue, we can adjust the probability threshold at which we consider an output "Good" or "Bad". Previously, we used 0.5, but seen below, if we change that to 0.85, our model becomes better at predicting which entries will labeled as "Bad". Although, it is worth noting that in becoming better at predicting the "Bad", we become worse at predicting the "Good".

```
## Confusion Matrix for Training dataset with updated threshold
mdp3<-model.drop$fitted.values
mdp3[mdp3>=0.85]<-1
mdp3[mdp3<0.85]<-0
table(GC_tr$Class,mdp3)
```

```
##       mdp3
##          0   1
##   Bad  196  19
##   Good 243 242
```

```
round(prop.table(table(GC_tr$Class,mdp3),1),2)
```

```
##       mdp3
##           0    1
##   Bad  0.91 0.09
##   Good 0.50 0.50
```

```
## Confusion Matrix for Holdout dataset with updated threshold
mdp2<-predict(model.drop,newdata=GC_te[,-1],type="response")
mdp2[mdp2>=0.85]<-1
mdp2[mdp2<0.85]<-0
table(GC_te$Class,mdp2)
```

```
##       mdp2
##          0   1
##   Bad   74  11
##   Good 105 110
```

```r
round(prop.table(table(GC_te$Class,mdp2),1),2)
```

```
##       mdp2
##           0    1
##   Bad  0.87 0.13
##   Good 0.49 0.51
```

Data Mining - Assignment 3: Part 2

Ryan Durfey

Sunday, February 22, 2015

Loading in the Previously Created Training and Holdout Datasets

```
library(rpart)
set.seed(8675309)

## load back in the Training and Test/Holdout datasets from Part 1
GC_train<-readRDS("C:/Misc Docs/UChicago/DataMining/Data_Mining/GC_train.rds")
GC_test<-readRDS("C:/Misc Docs/UChicago/DataMining/Data_Mining/GC_test.rds")
```

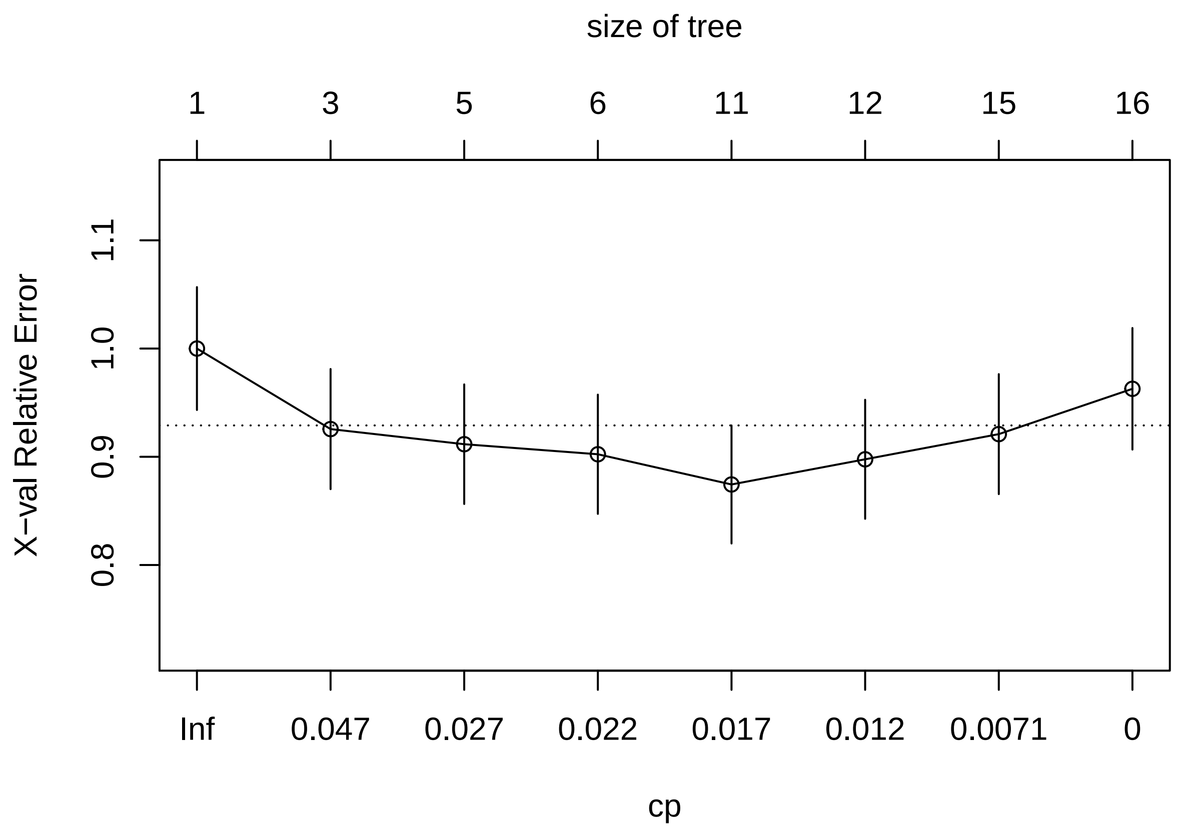
Classification Tree Creation

The first classification tree we create is with setting Cp=0. This will give us the largest tree possible. We can then determine how much we should prune the tree to get a better model.

```
## create first classification tree
tree.1<-rpart(GC_train,control=rpart.control(cp=0,minsplit=30,xval=10,maxsurrogate=0))
printcp(tree.1)
```

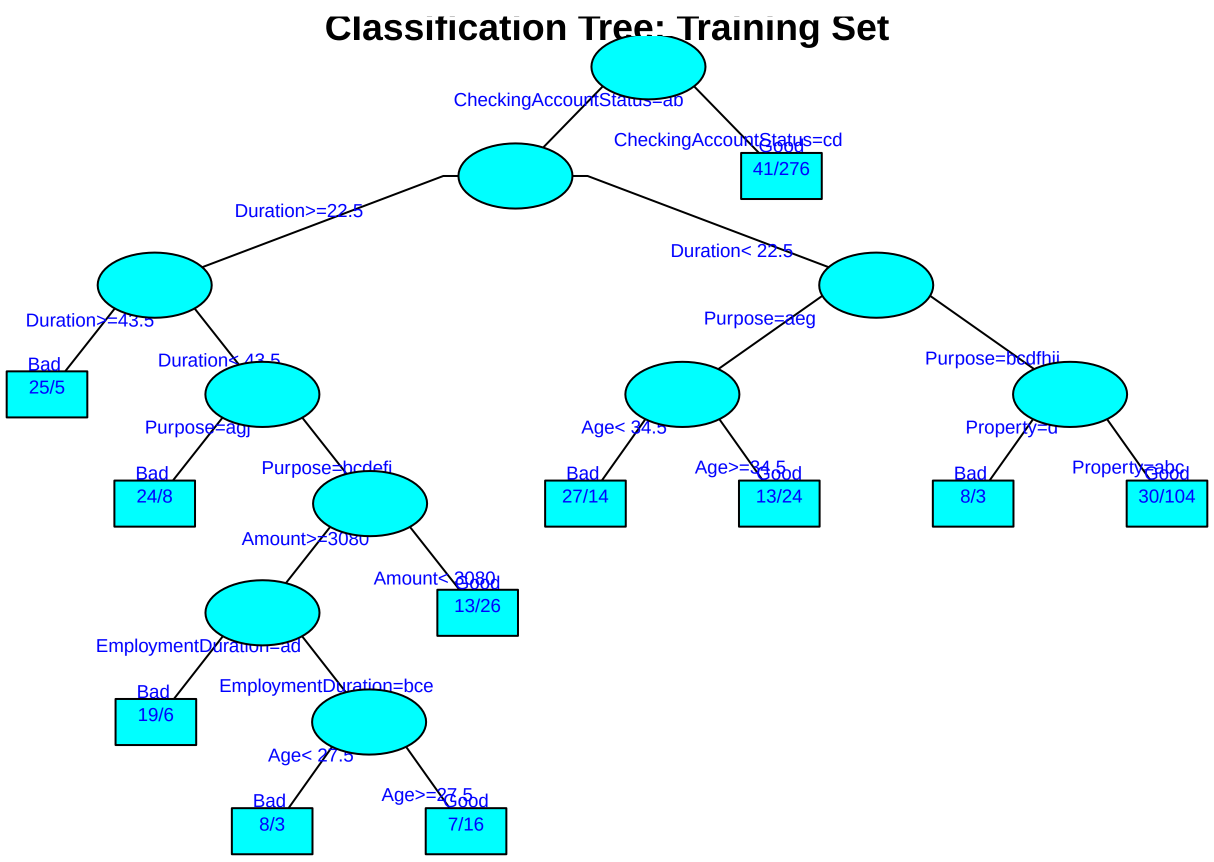
```
##
## Classification tree:
## rpart(formula = GC_train, control = rpart.control(cp = 0, minsplit = 30,
##     xval = 10, maxsurrogate = 0))
##
## Variables actually used in tree construction:
## [1] Age                   Amount                CheckingAccountStatus
## [4] Duration              EmploymentDuration    Property
## [7] Purpose
##
## Root node error: 215/700 = 0.30714
##
## n= 700
##
##          CP nsplit rel error  xerror     xstd
## 1 0.0744186      0   1.00000 1.00000 0.056768
## 2 0.0302326      2   0.85116 0.92558 0.055508
## 3 0.0232558      4   0.79070 0.91163 0.055253
## 4 0.0201550      5   0.76744 0.90233 0.055079
## 5 0.0139535     10   0.66512 0.87442 0.054541
## 6 0.0108527     11   0.65116 0.89767 0.054991
## 7 0.0046512     14   0.61860 0.92093 0.055424
## 8 0.0000000     15   0.61395 0.96279 0.056159
```

```
## The tree size that corresponds to the lowest cross-validation error is #5.
## This gives us Cp=0.0139535 and nsplit=10

plotcp(tree.1)
```



The lowest cross-validation error that we observe is 0.87442. This corresponds to a tree with a Cp of 0.0139535. We can then plug that Cp value into rpart() to build our pruned tree.

```
## Rebuild classification tree with parameters received above
tree.2<-rpart(GC_train,control=rpart.control(cp=0.0139535,minsplit=30,xval=10,maxsurrogate=0))

## plot the tree
opar<-par(no.readonly=TRUE)
par(mai=c(.1,.1,.1,.1))
plot(tree.2,main="Classification Tree: Training Set",col=3,compress=TRUE,branch=0.2,uniform=TRUE)
text(tree.2,cex=.6,col=4,use.n=TRUE,fancy=TRUE,fwidth=.4,fheight=.4,bg=c(5))
```



```r
par(opar)
```

Our pruned tree has 11 ending nodes from 10 splits. We see that right off the bat, if an entry has a favorable CheckingAccountStatus value, then it is classified as “Good”. If it doesn’t, then it goes through several other decision splits to determine its resulting value.

Confusion Matrices Using Pruned Tree

```r
## confusion matrices on training set with pruned tree
table(GC_train$Class,predict(tree.2,type="class"))
```

```
##
##         Bad Good
##   Bad   111  104
##   Good   39  446
```

```r
## prop.table by row (i.e., by Actual)
round(prop.table(table(GC_train$Class,predict(tree.2,type="class")),1),2)
```

```
##
##          Bad Good
##   Bad   0.52 0.48
##   Good 0.08 0.92
## prop.table by column (i.e., by Predicted)
round(prop.table(table(GC_train$Class,predict(tree.2,type="class")),2),2)
```

```
##
##          Bad Good
##   Bad  0.74 0.19
##   Good 0.26 0.81
```

Similar to our findings with Logistic Regression, from the above tables we see that our model seems to be better at classifying "Good" outputs than "Bad" ones. 92% of actual "Good" values were classified correctly, but only 52% of actual "Bad" values were correctly predicted. Unfortunately, unlike with the logistic regression model, we can't easily adjust our probability threshold to tweak this classification.

Holdout Validation

```
## confusion matrices on holdout set
table(GC_test$Class,predict(tree.2,newdata=GC_test[,-1],type="class"))
```

```
##
##         Bad Good
##   Bad    34   51
##   Good   41  174
```

```
round(prop.table(table(GC_test$Class,predict(tree.2,newdata=GC_test[,-1],type="class")),1),2)
```

```
##
##          Bad Good
##   Bad  0.40 0.60
##   Good 0.19 0.81
```

```
round(prop.table(table(GC_test$Class,predict(tree.2,newdata=GC_test[,-1],type="class")),2),2)
```

```
##
##          Bad Good
##   Bad  0.45 0.23
##   Good 0.55 0.77
```

```
## save tree for future use
saveRDS(tree.2,"C:/Misc Docs/UChicago/DataMining/Data_Mining/tree_2.rds")
```

For the holdout dataset, again we see our model working well to predict "Good" values, but poorly for "Bad" ones. Thus, the usefulness of this model may depend on other outside factors. For instance, if the the business is not very averse to "Bad" credit loans, then this model could be of use. However, if avoiding such "Bad" loans is very important, it may not be desireable.

Summary of Results & Comparison of Classification Tree vs. Logistic Regression

The classification tree model that we created provided a way to classify and predict “Good” and “Bad” credit loans based on certain criteria. This model, as illustrated in the tree plot, is easy to interpret and thus may be good at communicating the results to a business to transform into actionable ideas.

The tree model gave us similar results as the logistic regression model. Both were better at classifying “Good” outcomes than “Bad”. However, the logistic regression model did slightly better classifying “Bad” in the Holdout dataset (53% for logistic regression vs. 40% for the tree model). Neither number is great, admittedly, but this difference plus the ability to adjust the probability threshold to achieve a better classification rate according to business needs makes the logistic regression model the best choice.