Predicting advert click-through rates: Tree-based methods

14 September 2014

University of Cape Town

Department of Statistics

Ben Walwyn

WLWBEN001

Predicting Click-Through Rates of Website Advertisements

Predictions for online advertisement click-through rates are an important tool to increase effectiveness, manage costs and increase revenue. A recently shared set of data is available to develop models and gain insight into industry methods that are kept secret. A simple set of methods for analysing big data is decision trees. The objective for this section will be to fit a classification tree and random forest to inform and interpret important predictors for click-through rates.

A random set of 10 million observations is used. The data is not split up into training and test sets, as error rates will be measured and comparison based on cross-validation and out-of-bag techniques.

Each observation has 20 features. The response is binary - click or no click – stored in the *Label* variable. The other variables fall into the following categories:

- Publisher features – features specific to the host and distributer of the advert
- Advertiser – identifying and characteristic information for advertiser
- User - characteristics of internet user and their machine. (e.g. Browser type)
- Interaction of user with advertiser – information regarding history of use within the website.

All features are hidden with indistinguishable names. Integer names are names simply according to their number I<i> for $I = 1,2,\ldots,13$ and similarly for categorical variables are named C<c> for $c = 1,2,\ldots,7$.

Table 6: Advert explanatory variables

Variable Name	Type	(Min, Max)	Mean	StdDev	Missing Obs
I1	Integer	(0, 1900)	3.4969447	9.3561239	0
I2	Integer	(0, 35521)	105.8416596	390.1888956	4577554
I3	Integer	(0, 65535)	26.8076358	392.9192382	0
I4	Integer	(0, 761)	7.3224606	8.7978623	2166385
I5	Integer	(0, 23159456)	18548.8823	69628.3278969	2187370
I6	Integer	(0, 263348)	115.8059468	363.1187447	260350
I7	Integer	(0, 15350)	16.3379225	65.0777414	2255040
I8	Integer	(0, 6047)	12.5169720	16.7898764	435693
I9	Integer	(0, 25618)	106.0689976	221.1097671	5033
I10	Integer	(0, 9)	0.6174538	0.6838847	435693
I11	Integer	(0, 179)	2.7334833	5.2054770	4577554
I12	Integer	(0, 1881)	0.9924048	5.4825946	435693
I13	Integer	(0, 7086)	8.2187556	16.3995672	7718303
Variable Name	Type	Factor Levels			-

C1	Categorical	X<*i*> *i* = 1,..,22	1219055
C2	Categorical	X<*i*> *i* = 1,..,3	0
C3	Categorical	X<*i*> *i* = 1,..,27	0
C4	Categorical	X<*i*> *i* = 1,..,10	0
C5	Categorical	X<*i*> *i* = 1,..,3	4438109
C6	Categorical	X<*i*> *i* = 1,..,17	7691661
C7	Categorical	X<*i*> *i* = 1,..,15	0

From the table it immediately becomes clear that some variables may be less important predictors than others, simply due to the number of *NA* or missing data values. Almost half or more observations have no information for I2, I11, I13, C5 and C6. Although the fact that there is no observation might indicate something about the user or the probability of a click, the algorithms do not use missing values as indicators.

Although big data offers new opportunities for learning and modelling, it requires new computing solutions to perform analysis. Normal algorithms and techniques are not efficient enough in their use of memory and are therefore incapable of handling these large data sets – typically greater than 100 000 rows. The package *RevoScaleR,* developed by *Revolution Analytics* provides these new solutions and manages memory efficiently and implements tasks in parallel.

Some techniques are, however, still more time consuming than others. For example, the random forest took 16 hours to fit, while the classification tree took just 2 hours with similar settings. A balance between prediction power and the time cost is obviously important for any decision to be made within a specific context.

1. Big Data Descriptive analytics

The objective of descriptive statistics is to determine which variables are likely to be most important for any a model predicting clicks on an advertisement. In this case, no information is given for the variables, so the focus is on which predictors are important.

Big data decision trees employ the use of histograms to make split decisions rather than sorting which is computationally onerous. Some variables already have many substantial proportions of missing information, making their histograms less informative and less likely to be used for splits in a decision tree. There are other simple ways to describe the data set, the predictors and their relationship with the response.

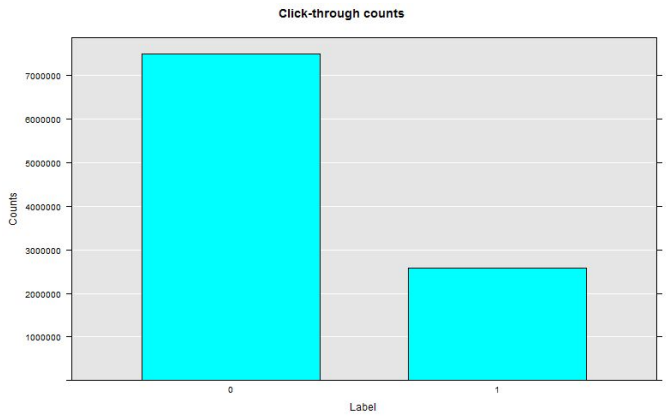

Figure 14: Bar plot of click-through counts

Firstly, the proportion of no clicks in the response is much higher than clicks. Approximately three quarters of all observations were not click-through. The membership will influence how the descriptive summaries are interpreted. For instance, a histogram of an integer predictor is expected to have higher frequencies for clicks. If it has a lower frequency, or even if the frequency is significantly more than 3 times, it indicates the variable might be a strong predictor. This majority proportion also impacts the predictions of the decision trees, and will be considered when setting controls.

Without considering other covariates or confounding, an initial measure of strong integer covariates is the correlation with the click response. Correlation ranges between -1 and 1. A large magnitude of correlation indicates a stronger relationship, positive or negative.

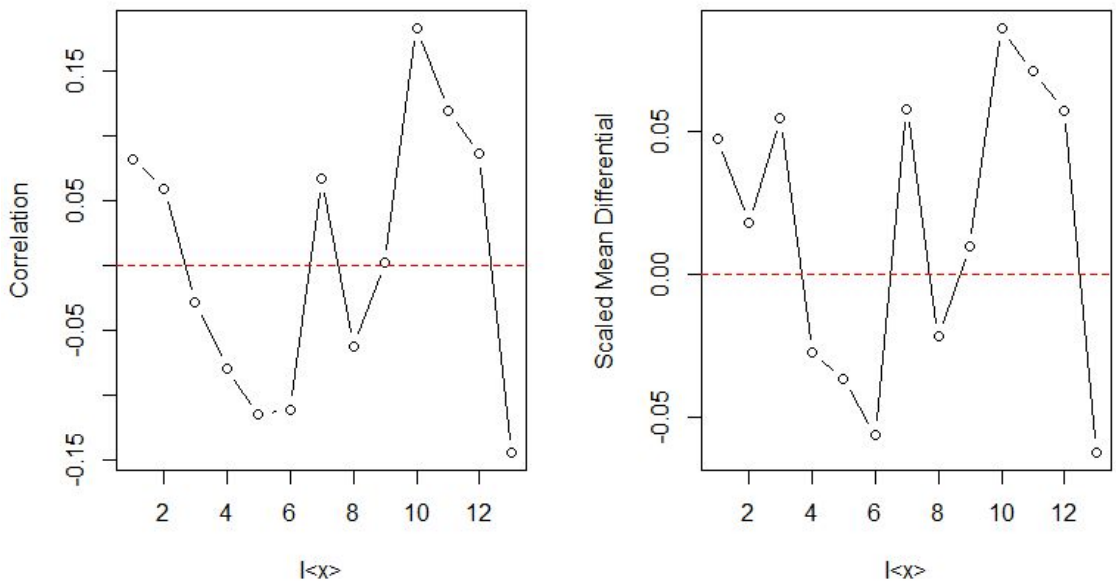

Figure 15: Correlation and mean differential plot identifying important integer variables with large magnitude. Plots correspond almost exactly.

Mean differentials for integer predictors between click and not-click also indicate whether or not there is a difference in the observed distribution for each response level. The mean differential as a fraction of the 99th percentile is reported in figure 15 for comparability between predictors, and reduces the effect of outliers on this comparability. The plots are very similar and it seems that figure 15 identifies I5, I6, I10 and I13 as possibly strong predictors for *label*.

A similar viewpoint was taken when analysing categorical variables. The counts for each categorical factor were cross-tabulated against the response. These counts can be compared by their proportion of the total counts of the response, which are essentially the discrete empirical distributions. The counts and percentages of C2 are shown in table 4 as an example.

Table 7: Categorical variables C2 frequencies and densities for label 0 and 1

	Frequency				Density		
Label	**X1**	**X2**	**X3**	**Row Sum**	**X1**	**X2**	**X3**
0	6608240	891836	1443	7501519	0.8809203	0.1188739	0.0001923
1	2456774	128150	195	2585119	0.9503524	0.0495721	0.0000754

If there are significant differences in the distribution of click and not-clicked counts, then the factor might be important. The more factors within a predictor with these significant differences, the more important the variable. Figure 16 shows the plots of all the factors within each variable. C4 and C5 have identifiable differences.

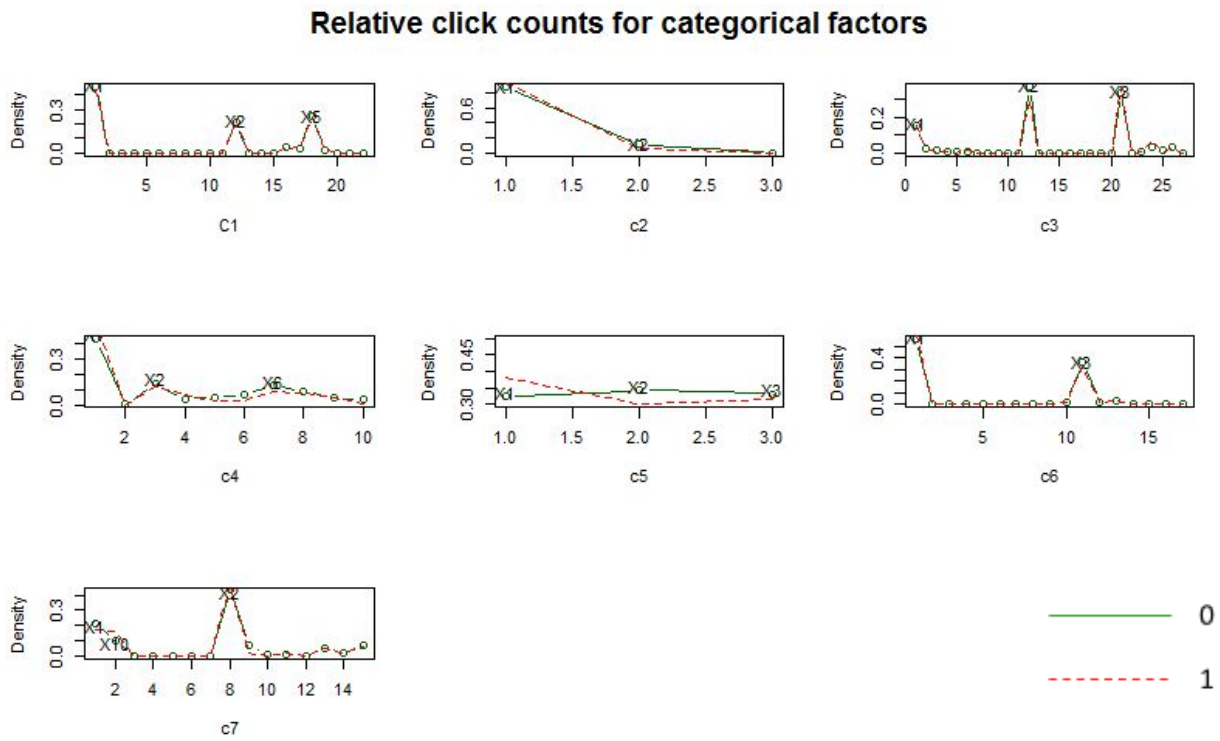

Figure 16: Relative click counts for factors of categorical variables. Factors are labelled if they have significant densities. A difference between densities indicates a possible strong predictor.

Table 6 lists the variables likely to be influential in a plot based on the difference between distributions of variables by *label.* Histograms for the integer variables are included in appendix B.

Table 8: The most likely variables to influence whether a visitor to a website clicks on advertisements

Variable
I5
I6
I10
I11
I13
C4
C5

Note, I11, I13 and C5 have substantial proportions of missing data which may reduce their effect when models are fitted.

2. Big Data Trees

2.1. Classification tree

Two simple classification trees were fitted to the entire sampled data set. The settings controlling the fit for both are outlined in table 6.

Table 9: Settings used to fit classification trees

Argument		Input
Formula	Variables names to be considered	Label ~ I1+I2+I3+I4+I5+I6+I7+I8+I9+I10+I11+I12+I13 +C1+C2+C3+C4+C5+C6+C7
xVal	Number of folds for cross-validation	10
Cp	Min. reduction for split to be implemented	0
maxDepth	Max numbers of splits to terminal node	10
Loss *(only for 2nd tree)*	Loss matrix for misclassification	0 1 3 0

- Firstly, all the explanatory variables are used to fit the tree.
- Cross-validation is performed over 10 folds, roughly of size 1 million each.
- There is no minimum reduction in node impurity for a split to be considered.
- All terminal nodes are within 10 splits from the root node.
- The first tree is fitted without the loss matrix while the second included a loss matrix which encourages more click predictions in terminal nodes. The choice of a 3 times loss is based on the 3:1 ratio of 0 to 1 in the response.

Both trees are pruned to reduce over fitting and select the tree with the lowest cross-validation error. The tuning parameter α is determined by the minimum cross-validation error. This error is still decreasing after more than 150 splits for the first tree indicating a larger tree could be fitted without over fitting. However, considering the scale of the decrease, the tree seems to effectively flatten out around 0.94. Choosing the corresponding α, the tree is pruned back to 55 splits.

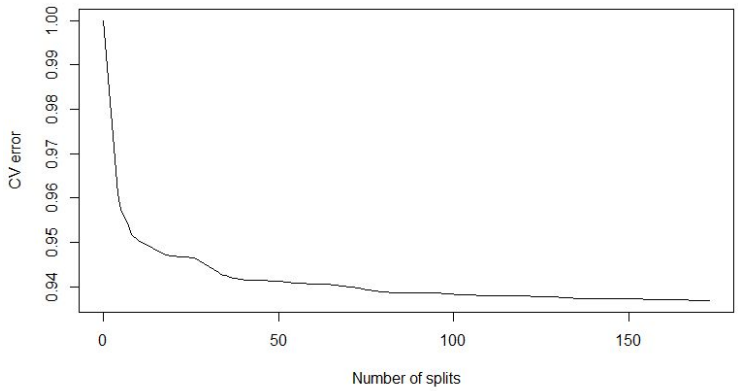

Figure 17 Cross-validation of tree without the loss tuning parameter

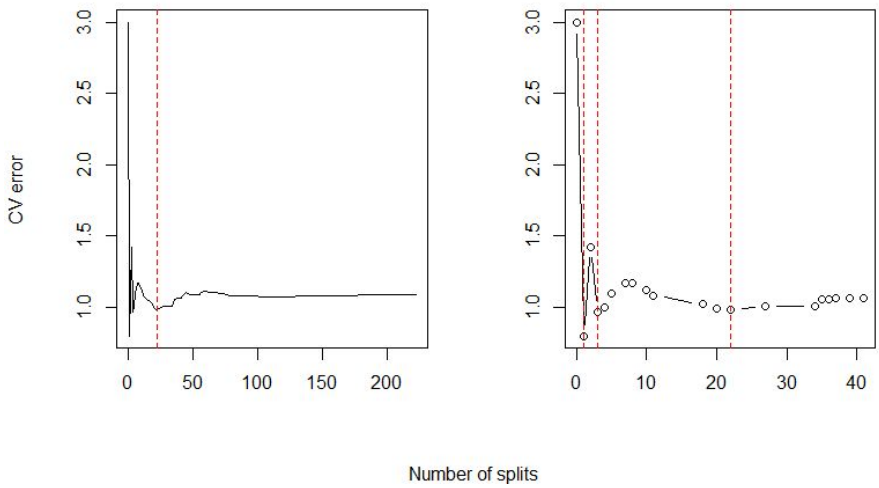

Figure 18: Cross-validation of tree including loss tuning. Chosen number of splits is 22. The number of terminal nodes is equal to the number of splits plus 1.

There is a lower CV error for a tree with 1 or 3 splits. The loss adjustment increases the number of 1's predicted, and the improvement in the error is greater for these fewer nodes. With such few splits, however, there are few variables used to make splits. This does not help to identify which variables are important in advertising, and hence, a tree of this size is not useful. Therefore, 23 terminal nodes are selected. Table 7 shows that the tree without an adjusted loss matrix has a lower relative error.

Table 10: Relative CV error for classification trees

	No loss	*Loss*
α	0.0002	0.0005
CV relative error	0.9408484	0.9836814

However, this does mean outright that it is the better tree. The reason for tuning the adjusted loss matrix is apparent in the misclassification for the first tree in table 8.

Table 11: Unadjusted classification tree confusion matrix

	Predicted	
Observed	0	1
0	0.9675864	0.03241357
1	0.8449920	0.15500795

The first tree predicts 84.5% of clicks incorrectly as no clicks. Although it predicts 97% percent of no clicks correctly, this tree is mostly just predicting zeros to minimise error since the response is so one-sided. Therefore, the loss tuning parameter attempts to improve predictions on the relatively fewer number of observed clicks. The confusion matrix for the second tree, table 9, clearly improves the number of predicted clicks, but at a cost – 28% of no click-through are predicted as click-through, up from 3%.

Table 12: Loss-adjusted tree confusion matrix

	Predicted	
Observed	0	1
0	0.7180120	0.2819880
1	0.4324284	0.5675716

The preferred classification tree depends on the real operation cost of not predicting a click-through when one actually occurs and vice versa. It must also be noted that the predictions are performed on observations used to fit the trees. I.e. The data was not split into a test and training set. Hence, the misclassification might be underestimated for either or both factors in each tree.

The full plot for the loss-adjusted tree is drawn in figure 19, colour-coded by variable. One example path is described by table 10.

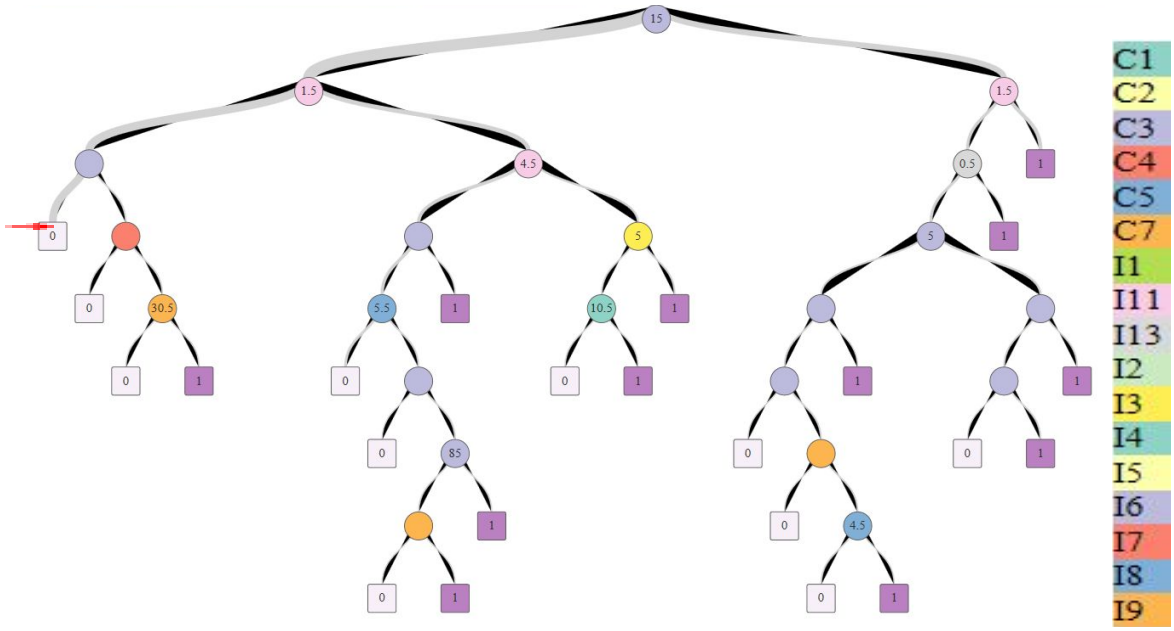

Figure 19: Loss-adjusted classification tree. Number within nodes represents split threshold for integer variables.

Table 13: Sample path to terminal node marked by red arrow with prediction 0

Node	Variable	Partition
1	I6	≥ 15
2	I11	< 1.5
3	C3	X1,X2,X3,X4,X7,X8,X9,X11,X12,X13,X16,X18,X19,X20,X21,X22,X24,X25,SX26,X27

The variables for the first few splits of the tree determine are the important ones. The top three important variables are also the first 3 splits for the sample path in table 10. Variable importance is, therefore, equally as variable as the classification tree. Variables C3 and C7 have very little influence from a high-level descriptive view of the data, but contribute relatively large increases in node purity.

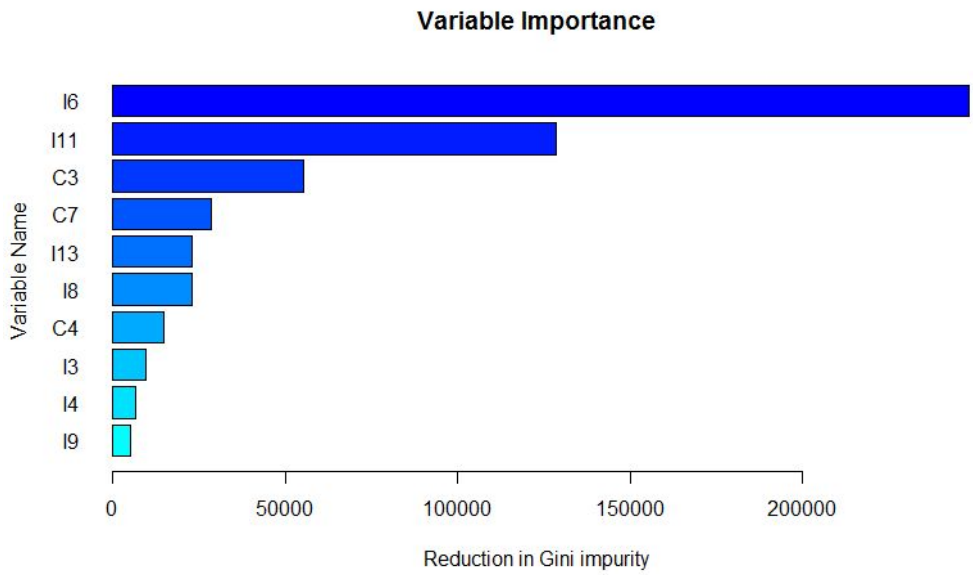

Figure 19: Ten most important variables plotted relative to each other in a bar plot, in decreasing importance.

The first tree does not penalise incorrect click-through predictions. In this context, it can be used to identify what makes people decide not to click-through and improve online advertisements. However, a lot of the people predicted not to click did in fact click, reducing the effectiveness of this approach. The second tree might be a more appropriate model to derive inferences due to the improvement in click-through predictions.

2.2. Random forests

The random forest fits a large number of trees to reduce dependence on strong indicators and reduce variability. The settings, such as the number of trees, are outlined in table 11 below.

Table 14: Settings used to fit random forest

Argument		Input
Formula	Variables names to be considered	Label ~ I1+I2+I3+I4+I5+I6+I7+I8+I9+I10+I11+I12+I13 +C1+C2+C3+C4+C5+C6+C7
nTree	Number of trees to fit to bootstrapped samples	500
mTry	Number of variables to be randomly sampled for split consideration	5 (default = sqrt(20) =4.58)
importance	Store impurity changes to measure var. importance	TRUE
loss	Loss matrix for misclassification	0 1 3 0

- All variables are used for randomisation at each split.

- Trees are grown on 500 bootstrap samples.
- 5 variables are randomised to consider at each split in a tree.
- The reduction in node impurity at each split in a tree is stored to analyse variable importance.
- The random forest penalised incorrectly predicted clicks more heavily than an incorrectly predicted no click-through.

The *mTry* argument ensures that the trees are less influenced by any group of strong predictors, and that predictions are less dependent on each other. This reduces their variability and leads to a decrease in error.

The improvement in the test error for a given random forest is also dependent on the number of trees within the forest. This is apparent in figure 20. The error reaches its minimum after around 80-100 trees. The variation seen after is the variation of each tree when fitted to a different bootstrap sample.

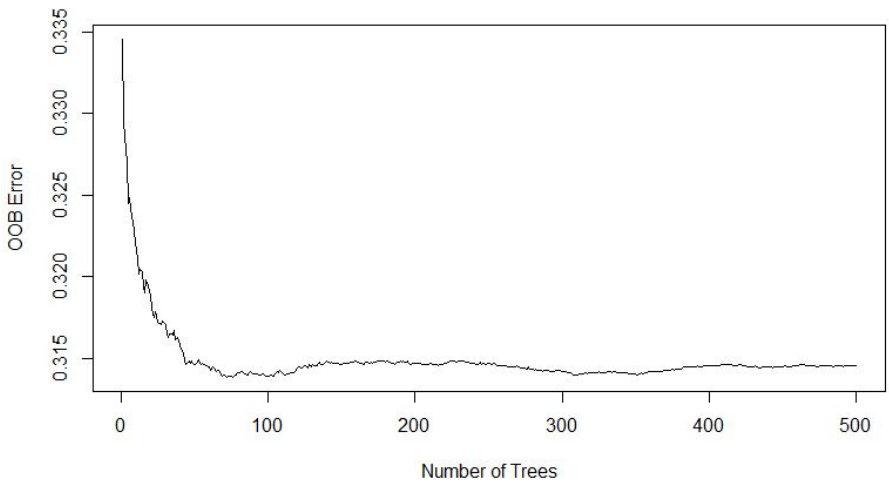

Figure 20: Out-of-bag error for random forest. 100 trees are sufficient to decrease the variability of predictions and decrease the error as much as possible. Random forests do not over fit a data set by including more trees, but there is a minimum error achievable for each random forest. The minimum is around 0.315 for this random forest.

The majority votes of the predictions on OOB sample are selected as the final prediction for each observation. The confusion matrix is seen in table 12. There are now 70% and 63% correctly prediction 0's and 1's. Compared to the simple classification tree, this is a significant improvement, but s much smaller improvement on the loss-adjusted tree. Importantly however, this improvement is made on the number of incorrectly classified clicks, which was influenced the most by the skewed response distribution.

Table 15: Loss-adjusted random forest confusion matrix

	Predicted	
Observed	0	1
0	0.7051327	0.2948673
1	0.3716266	0.6283734

The random forest improves overall ability to predict, but at the cost of interpretation. The exact path for assigning predictions can no longer be identified as in figure 19. Alternatively, variable importance as measured by the average reduction in node impurity across all trees is plotted in figure 21.

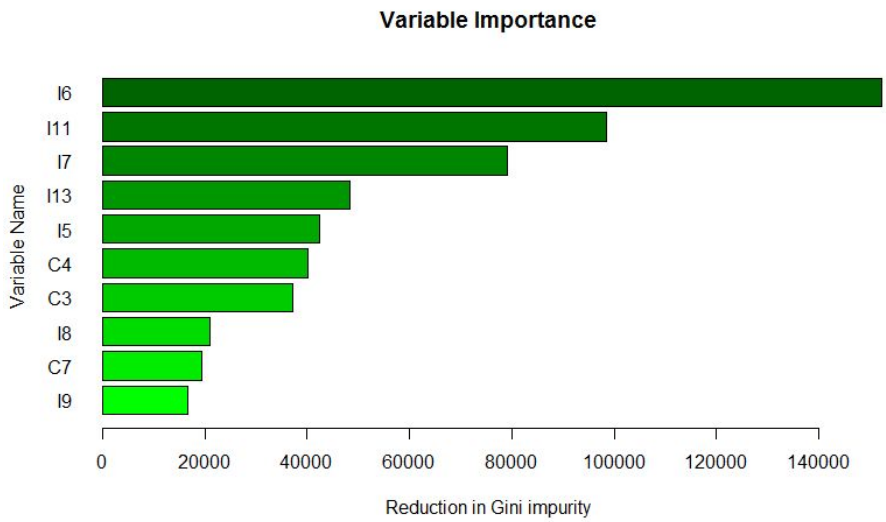

Figure 21: Ten most important variables plotted relative to each other in a bar plot, in decreasing importance.

The *decorrelation* of tree is apparent through the increased importance of other variables besides I6. Note also how C3 is no long so highly ranked, illustrating the variability of a decision tree. The previously identified variables using descriptive statistics variables are more important, namely I13, I5 and C4. I7 is now the third most important variable. Appendix B includes a histogram of I7 by response 0 and 1 which shows that proportionally more click-through counts occur at low values. I6 and I11 remain the most important predictors.

3. Conclusions and recommendations

3.1. The random forest is the better model than the classification tree and should be used to predict click-through rates. Although a random forest without a loss-adjustment was not fitted, there is a clear improvement in predictions compared to the loss-adjusted tree. Random forests in general reduce the variability of these predictions without over fitting.

3.2. Loss adjustment works as a method of rebalancing the misclassification error when the data is heavily skewed. In this example, it increases the number of clicks that are

predicted. This improves the correctly predicted clicks, but at the cost of an increase in misclassified clicks.

3.3. Integer predictors are better than categorical predictors. These are mostly counts, representing features like the number of time a user visits a website and sees the advertisement.

3.4. The proportions of the binary response have a large influence on the decision trees.

3.5. Missing data reduces the effectiveness of some variables in the model that might influence decisions. High volumes of observations have missing data for I11 and C5. If wider collection of this data is possible, the model might be further improved.

3.6. Short ranges of continuous predictors have an impact on the importance and reduction in error. For a predictor with a short length, it effectively becomes a categorical variable with the length as the number of factors when converted to an integer. This reduces the effectiveness of binary splitting. For example, I<> = 0.4 might have a higher probability of a click than 0.3, but both are represented as 0. Perhaps the predictor could be transformed when converting the continuous observation. I<> = 2X could be a proposed transformation. I.e. 0.5 could correspond to 1, 1 to 2. 1.5 to 3 and so on. This would increase the length and increase the importance of the predictor. It is also still relatively easy to interpret.

4. The use of logistic regression to predict click-through rates

The click-through decision space can be thought of as a gradient ranging from aversion to inclination of clicking. The tree-based solution aims to predict the behaviour of individuals in the internet environment, by partitioning their measurable characteristic features at discrete points into homogenous groups that would decide unanimously to click-through or not. However, behaviour does not change at any single point on this gradient, but the probability of clicking might. There are always individual characteristic differences that cannot be accounted for which influence decisions. The user's position of the gradient could alternatively be estimated by the combination of factors that influence the behaviour of the user at the point of interaction. An advertising solution must therefore aim to understand the dynamic within this space and place users at the end with higher probability of clicking. This is essentially the purpose of logistic regression model, and might fit the behavioural response better.

Appendix A

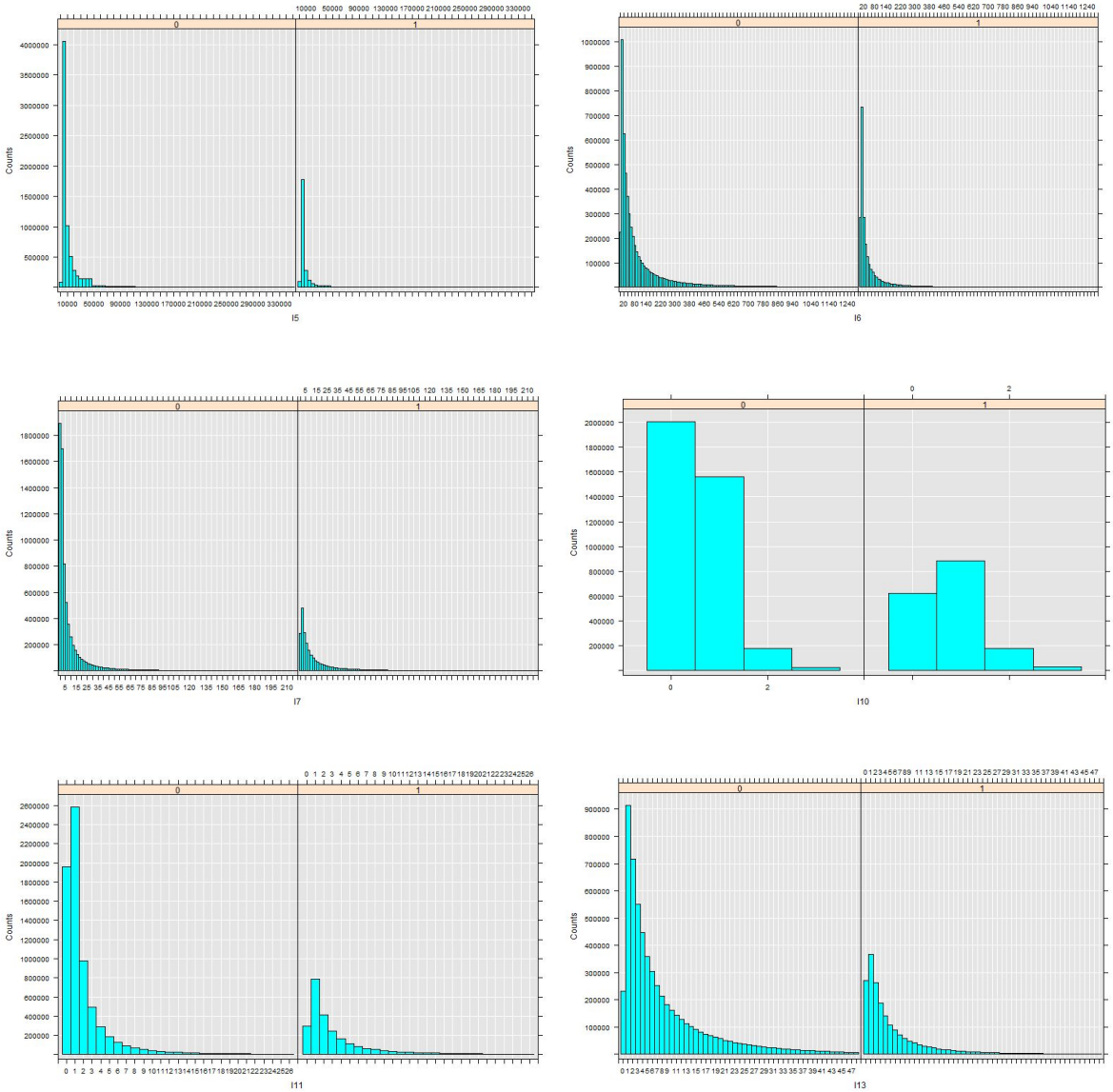

Figure 1: Histograms of strongest integer predictors plotted for response.

Appendix B: R code

```
library(RevoScaleR)
options(scipen=999)

save.image("D:/Analytics/bigtree.RData")

#create xdf
ad <- rxImport(inData="advert25.csv", outFile='ad.xdf')
#import created xdf
ad <- RxXdfData(file="ad.xdf")

#Recode
recode<-rxFactors(inData=ad,
                  factorInfo=c('Label','C1','C2','C3','C4','C5','C6','C7'),
                  sortLevels = T,
                  outFile='recodedAd.xdf',
                  overwrite=T
                  )

recode <- RxXdfData(file="recodedAd.xdf")

# --- Summary statistics --- #

#Exploration
adI <- rxGetInfo(recode, getVarInfo=TRUE, numRows=6)
adI
head(recode)
nrow<-nrow(recode)

###'Label' - analysing the response
rxSummary(Label~., data=recode)
lab.tab <- rxCrossTabs(~Label,data=recode)
lab.prop <- unlist(lab.tab$counts)/nrow(recode)
lab.prop
rxHistogram(~Label, data=recode,main='Click-through counts')

###Predictors

#Integers
cor.matrix <- rxCor(~Label+I1+I2+I3+I4+I5+I6+I7+I8+I9+I10+I11+I12+I13,
data=recode)
cor.matrix[2,3:15]


#99 quantiles
q99<-cbind(
  rxQuantile('I1',data=recode,probs=0.99), #numeric
  rxQuantile('I2',data=recode,probs=0.99),
  rxQuantile('I3',data=recode,probs=0.99),
  rxQuantile('I4',data=recode,probs=0.99),
  rxQuantile('I5',data=recode,probs=0.99),
  rxQuantile('I6',data=recode,probs=0.99),
  rxQuantile('I7',data=recode,probs=0.99),
  rxQuantile('I8',data=recode,probs=0.99),
  rxQuantile('I9',data=recode,probs=0.99),
  rxQuantile('I10',data=recode,probs=0.99),
```

```
  rxQuantile('I11',data=recode,probs=0.99),
  rxQuantile('I12',data=recode,probs=0.99),
  rxQuantile('I13',data=recode,probs=0.99)
)

#scaled means by click/no-click
scaled.means<-cbind(
  means.I1<-rxSummary(~Label:I1, data=recode)$categorical[[1]][,3]/q99[,1],
  means.I2<-rxSummary(~Label:I2, data=recode)$categorical[[1]][,3]/q99[,2],
  means.I3<-rxSummary(~Label:I3, data=recode)$categorical[[1]][,3]/q99[,3],
  means.I4<-rxSummary(~Label:I4, data=recode)$categorical[[1]][,3]/q99[,4],
  means.I5<-rxSummary(~Label:I5, data=recode)$categorical[[1]][,3]/q99[,5],
  means.I6<-rxSummary(~Label:I6, data=recode)$categorical[[1]][,3]/q99[,6],
  means.I7<-rxSummary(~Label:I7, data=recode)$categorical[[1]][,3]/q99[,7],
  means.I8<-rxSummary(~Label:I8, data=recode)$categorical[[1]][,3]/q99[,8],
  means.I9<-rxSummary(~Label:I9, data=recode)$categorical[[1]][,3]/q99[,9],
  means.I10<-rxSummary(~Label:I10,
data=recode)$categorical[[1]][,3]/q99[,10],
  means.I11<-rxSummary(~Label:I11,
data=recode)$categorical[[1]][,3]/q99[,11],
  means.I12<-rxSummary(~Label:I12,
data=recode)$categorical[[1]][,3]/q99[,12],
  means.I13<-rxSummary(~Label:I13, data=recode)$categorical[[1]][,3]/q99[,13]
)
par(mfrow=c(1,2))
#correlation plot
plot(seq(1:13),cor.matrix[2,3:15],type='b',xlab='I<x>',ylab='Correlation',cex
.main=0.8)
abline(a=0,b=0,col='red',lty=2)
#plot scaled means
plot(seq(1:13),-(scaled.means[1,]-scaled.means[2,]),type='b',xlab='I<x>',
     ylab='Scaled Mean Differential')
abline(a=0,b=0,col='red',lty=2)

#Only plot relevant histograms
rxHistogram(~I5|Label, data=recode,startVal=0,endVal=q99[,5]) # more
rxHistogram(~I6|Label, data=recode,startVal=0,endVal=q99[,6]) # more
rxHistogram(~I7|Label, data=recode,startVal=0,endVal=q99[,7]) # more
rxHistogram(~I10|Label, data=recode,startVal=0,endVal=q99[,10]) #
interesting
rxHistogram(~I11|Label, data=recode,startVal=0,endVal=q99[,11]) #more non
clicks for 0
rxHistogram(~I13|Label, data=recode,startVal=0,endVal=q99[,13]) # =0 more
clicks

sum.I5<-rxSummary(~Label:I5,data=recode)
sum.I10<-rxSummary(~Label:F(I10), data=recode)
sum.I10


#Categorical
tab.c2$counts[[1]]
tab.c2.scaled

par(mfrow=c(3,3))
tab.c1<-rxCrossTabs(~Label:C1, data=recode)
tab.c1.scaled<-rbind(tab.c1$counts[[1]][1,]/sum(tab.c1$counts[[1]][1,]),tab.c
1$counts[[1]][2,]/sum(tab.c1$counts[[1]][2,]))
```

```
plot(seq(1:22),tab.c1.scaled[1,],type='b',col='darkgreen',xlab='C1',ylab='Den
sity')
lines(seq(1:22),tab.c1.scaled[2,],col='red',lty=2)
which.c1<-which(tab.c1.scaled[1,]>0.1, arr.ind=TRUE)
text(which.c1,subset(tab.c1.scaled[1,],tab.c1.scaled[1,]>0.1)+0.015,labels=co
lnames(tab.c1.scaled)[which.c1])

tab.c2<-rxCrossTabs(~Label:C2, data=recode)
tab.c2.scaled<-rbind(tab.c2$counts[[1]][1,]/sum(tab.c2$counts[[1]][1,]),tab.c
2$counts[[1]][2,]/sum(tab.c2$counts[[1]][2,]))
plot(seq(1:3),tab.c2.scaled[1,],type='b',col='darkgreen',xlab='c2',ylab='Dens
ity')
lines(seq(1:3),tab.c2.scaled[2,],col='red',lty=2)
which.c2<-which(tab.c2.scaled[1,]>0.1, arr.ind=TRUE)
text(which.c2,subset(tab.c2.scaled[1,],tab.c2.scaled[1,]>0.1)+0.015,labels=co
lnames(tab.c2.scaled)[which.c2])

tab.c3<-rxCrossTabs(~Label:C3, data=recode)
tab.c3.scaled<-rbind(tab.c3$counts[[1]][1,]/sum(tab.c3$counts[[1]][1,]),tab.c
3$counts[[1]][2,]/sum(tab.c3$counts[[1]][2,]))
plot(seq(1:27),tab.c3.scaled[1,],type='b',col='darkgreen',xlab='c3',ylab='Den
sity')
lines(seq(1:27),tab.c3.scaled[2,],col='red',lty=2)
which.c3<-which(tab.c3.scaled[1,]>0.1, arr.ind=TRUE)
text(which.c3,subset(tab.c3.scaled[1,],tab.c3.scaled[1,]>0.1)+0.015,labels=co
lnames(tab.c3.scaled)[which.c3])

tab.c4<-rxCrossTabs(~Label:C4, data=recode)
tab.c4.scaled<-rbind(tab.c4$counts[[1]][1,]/sum(tab.c4$counts[[1]][1,]),tab.c
4$counts[[1]][2,]/sum(tab.c4$counts[[1]][2,]))
plot(seq(1:10),tab.c4.scaled[1,],type='b',col='darkgreen',xlab='c4',ylab='Den
sity')
lines(seq(1:10),tab.c4.scaled[2,],col='red',lty=2)
which.c4<-which(tab.c4.scaled[1,]>0.1, arr.ind=TRUE)
text(which.c4,subset(tab.c4.scaled[1,],tab.c4.scaled[1,]>0.1)+0.03,labels=col
names(tab.c4.scaled)[which.c4])

tab.c5<-rxCrossTabs(~Label:C5, data=recode)
tab.c5.scaled<-rbind(tab.c5$counts[[1]][1,]/sum(tab.c5$counts[[1]][1,]),tab.c
5$counts[[1]][2,]/sum(tab.c5$counts[[1]][2,]))
plot(seq(1:3),tab.c5.scaled[1,],type='b',col='darkgreen',xlab='c5',ylab='Dens
ity',ylim=c(0.3,0.5))
lines(seq(1:3),tab.c5.scaled[2,],col='red',lty=2)
which.c5<-which(tab.c5.scaled[1,]>0.1, arr.ind=TRUE)
text(which.c5,subset(tab.c5.scaled[1,],tab.c5.scaled[1,]>0.1)+0.015,labels=co
lnames(tab.c5.scaled)[which.c5])

tab.c6<-rxCrossTabs(~Label:C6, data=recode)
tab.c6.scaled<-rbind(tab.c6$counts[[1]][1,]/sum(tab.c6$counts[[1]][1,]),tab.c
6$counts[[1]][2,]/sum(tab.c6$counts[[1]][2,]))
plot(seq(1:17),tab.c6.scaled[1,],type='b',col='darkgreen',xlab='c6',ylab='Den
sity')
lines(seq(1:17),tab.c6.scaled[2,],col='red',lty=2)
which.c6<-which(tab.c6.scaled[1,]>0.1, arr.ind=TRUE)
text(which.c6,subset(tab.c6.scaled[1,],tab.c6.scaled[1,]>0.1)+0.015,labels=co
lnames(tab.c6.scaled)[which.c6])

tab.c7<-rxCrossTabs(~Label:C7, data=recode)
```

```
tab.c7.scaled<-rbind(tab.c7$counts[[1]][1,]/sum(tab.c7$counts[[1]][1,]),tab.c
7$counts[[1]][2,]/sum(tab.c7$counts[[1]][2,]))
plot(seq(1:15),tab.c7.scaled[1,],type='b',col='darkgreen',xlab='c7',ylab='Den
sity')
lines(seq(1:15),tab.c7.scaled[2,],col='red',lty=2)
which.c7<-which(tab.c7.scaled[1,]>0.1, arr.ind=TRUE)
text(which.c7,subset(tab.c7.scaled[1,],tab.c7.scaled[1,]>0.1)-0.015,labels=co
lnames(tab.c7.scaled)[which.c7])
par('oma'=c(0,0,2,0))
mtext(expression(bold('Relative click counts for categorical
factors')),side=3,line=0,outer=T,cex=1.2)
par(mfrow=c(1,1))
par('oma'=c(0,0,0,0))

cube.c1 <- rxCube(~Label:C4,data=recode)
rxLinePlot(formula=Counts~Label|C4,data=rxResultsDF(cube.c1))

# --- Classification Tree --- #

adClassTree <- rxDTree(Label ~
I1+I2+I3+I4+I5+I6+I7+I8+I9+I10+I11+I12+I13+C1+C2+C3+C4+C5+C6+C7,
                       data=recode,
                       method="class",
                       xVal=10,
                       cp=0,
                       maxDepth=10,
                       parms=list(loss=c(0,3,1,0))
                      )

adClassTree.noloss <- rxDTree(Label ~
I1+I2+I3+I4+I5+I6+I7+I8+I9+I10+I11+I12+I13+C1+C2+C3+C4+C5+C6+C7,
                       data=recode,
                       method="class",
                       xVal=10,
                       cp=0,
                       maxDepth=10
)

#Cross-validation plots
par(mfrow=c(1,1))
cptable.noloss <- adClassTree.noloss$cptable
plot(cptable.noloss[,'nsplit'],cptable.noloss[,'xerror'],type='l',xlab='Numbe
r of splits',ylab='CV error')

par(mfrow=c(1,2))
par('oma'=c(1,1,0,0))
cptable <- adClassTree$cptable
plot(cptable[,'nsplit'],cptable[,'xerror'],type='l',xlab=,ylab='CV
error',ann=F)
abline(v=c(22),col='red',lty=2)
plot(cptable[1:20,'nsplit'],cptable[1:20,'xerror'],type='b',xlab='Number of
splits',ann=FALSE)
abline(v=c(1,3,22),col='red',lty=2)
mtext('Number of splits',outer=T,side=1)
mtext('CV error',outer=T,side=2)
par('oma'=c(0,0,0,0))
par(mfrow=c(1,1))
```

```
#Tree pruning
cp.no<-22
adClassPruned <-
prune(adClassTree,cp=cptable[which(cptable[,'nsplit']==cp.no,
arr.ind=TRUE),'CP'])
adClassPruned
adClassPrune.noloss <-
prune(adClassTree.noloss,cp=cptable.noloss[which(cptable[,'nsplit']==55,
arr.ind=TRUE),'CP'])
adClassPrune.noloss

# Predictions:
tree.pred <- rxPredict(adClassPruned, data=recode, outData="predictions.xdf",
type="class", writeModelVars=TRUE, overwrite=TRUE)
tree.pred.noloss <- rxPredict(adClassPrune.noloss,
data=recode,outData="predictions_noloss.xdf", type="class",
writeModelVars=TRUE, overwrite=TRUE)

#variable name for prediciton Label_pred
rxGetInfo(tree.pred, getVarInfo=TRUE)
rxGetInfo(tree.pred.noloss, getVarInfo=TRUE)
#Percentages
miss.class <- rxCrossTabs(~Label:Label_Pred, data=tree.pred)
miss.class.mat<- miss.class$counts[[1]]
miss.class.mat/rowSums(miss.class.mat)

miss.class.noloss <- rxCrossTabs(~Label:Label_Pred, data=tree.pred.noloss)
miss.class.mat.noloss<- miss.class.noloss$counts[[1]]
miss.class.mat.noloss/rowSums(miss.class.mat.noloss)

#Tree plot for loss
plot(rxAddInheritance(adClassPruned))
text(rxAddInheritance(adClassPruned), cex=0.3, pretty=0)

library(RevoTreeView)
plot(createTreeView(adClassPruned))

  ###Variable importance
colblue <- colorRampPalette(c('cyan', "blue"))
var.imp <- adClassPruned$variable.importance
barplot(sort(var.imp[which(var.imp>3000)]),horiz=T,las=1,xpd=F,col=colblue(10
),
        main='Variable Importance',xlab='Reduction in Gini
impurity',ylab='Variable Name')

#--- Random Forest ---#

adRf <- rxDForest(Label ~
I1+I2+I3+I4+I5+I6+I7+I8+I9+I10+I11+I12+I13+C1+C2+C3+C4+C5+C6+C7,
                  data=recode,
                  outFile="adRandomForest.xdf",
                  method="class",
                  mTry=5,
                  nTree = 500,
                  parms=list(loss=c(0,3,1,0)),
                  importance=TRUE,
)
```

```
OOB.predrf <- adRf$confusion
rf.confusion <- OOB.predrf[,1:2]/rowSums(OOB.predrf[,1:2])
rf.confusion
#Plots
plot(adRf,main='')
rxVarImpPlot(adRf)

colgreen <- colorRampPalette(c("green", "darkgreen"))
barplot(sort(rxVarImpPlot(adRf)[1:20])[11:20],horiz=T,las=1,xpd=F,col=colgree
n(10),
        main='Variable Importance',xlab='Reduction in Gini
impurity',ylab='Variable Name')

# Predictions:
tree.predrf <- rxPredict(adRf, data=recode, outData="predictionsRF.xdf",
                         type="class", writeModelVars=TRUE, overwrite=TRUE)
rxGetInfo(tree.predrf, getVarInfo=TRUE)

miss.classrf <- rxCrossTabs(~Label:Label_Pred, data=tree.predrf)
miss.classrf.mat/rowSums(miss.classrf.mat)
```