Decision Trees

Ross Jacobucci

March 27, 2015

This script will go over Decision Trees in R. It will not cover generalization such as SEM Trees or Rasch Trees

Resources:
Basic intro to Decision Trees: http://www.statmethods.net/advstats/cart.html
Full list of data mining packages in R: http://cran.r-project.org/web/views/MachineLearning.html

For longitudinal data:
REEMtree

Two packages will be used and their caret equivalents:
rpart (tree accomplishes very similar thing):http://cran.r-project.org/web/packages/rpart/vignettes/longintro.pdf

party: http://cran.r-project.org/web/packages/party/vignettes/party.pdf

In caret, method =
"rpart" – tuning = cp (complexity parameter)
"rpart2" – tuning = maxdepth
"rpartCost" – tuning = cp and cost
"ctree" – tuning = mincriterion (p value thresholds)
"ctree2" – tuning = maxdepth
(see "train_model_list" in caret reference manual)

Bonus:
non-greedy tree algorithm:
evtree: http://cran.r-project.org/web/packages/evtree/vignettes/evtree.pdf

Lets load the main packages

```
library(caret)
library(rpart)
library(pROC)
library(party)
library(MASS) # for boston data
data(Boston)
```

Regression (continous outcome)

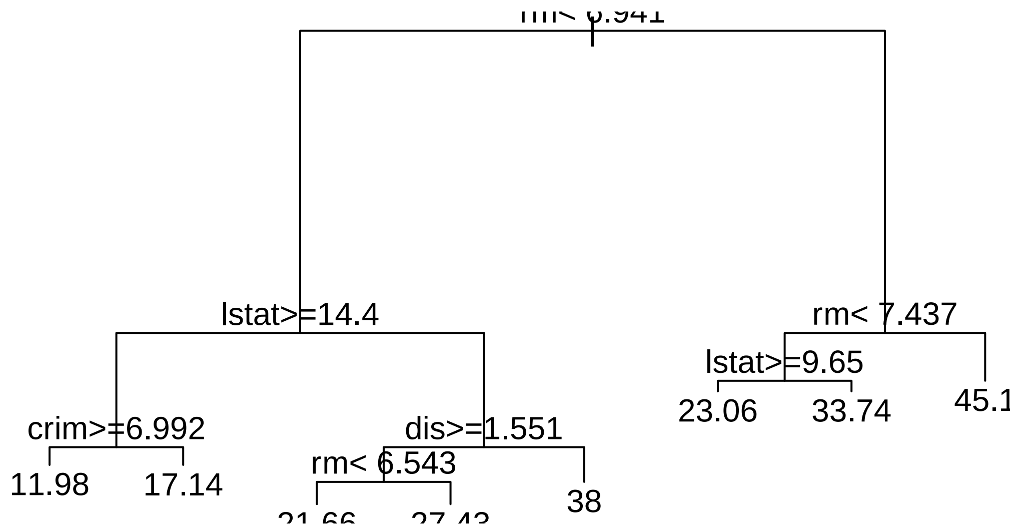
Use rpart first with the Boston data use regression first – predicting median value of homes

```
#str(Boston)

# lets get a baseline with linear regression
lm.Boston <- lm(medv ~., data=Boston)
#summary(lm.Boston)
```

We do pretty well with linear regression R-squared of .74

How about if we just blindly apply Decision Trees

```
rpart.Boston <- rpart(medv ~., data=Boston)
#summary(rpart.Boston)
plot(rpart.Boston);text(rpart.Boston)
```



```
pred1 <- predict(rpart.Boston)
cor(pred1,Boston$medv)**2
```

```
## [1] 0.8075721
```

Doing really well – Rsquared = 0.81

What if we tried regularized (penalized) regression instead?
Note: for glmnet, both the x's and y have to be in separate matrices
– and all class = numeric
– don't worry about response, doesn't have to be factor for logistic
—– just specify "binomial"

```
y.B <- Boston$medv
x.B <- sapply(Boston[,-14],as.numeric)

# alpha =1 for lasso, 0 for ridge
library(glmnet)
```

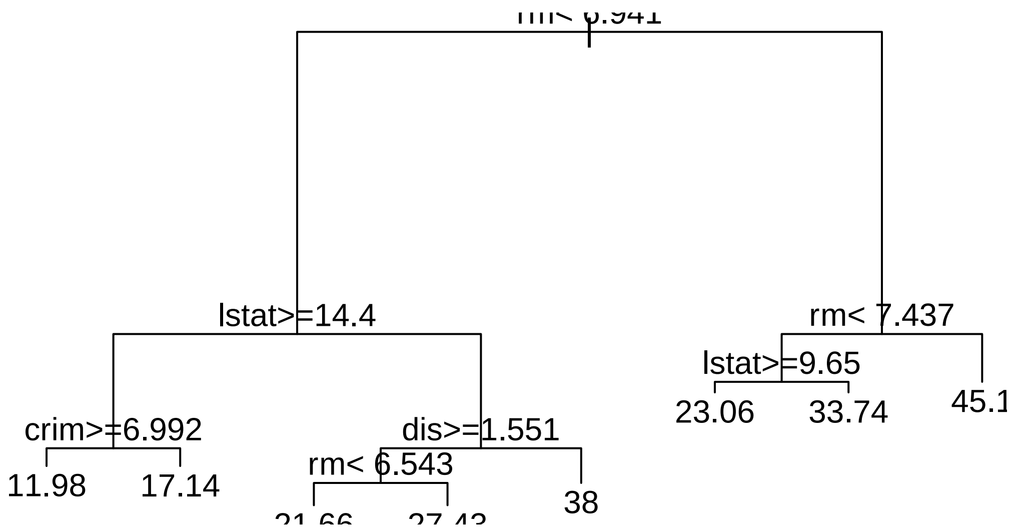
```
## Loading required package: Matrix
## Loaded glmnet 1.9-8
##
##
## Attaching package: 'glmnet'
##
## The following object is masked from 'package:pROC':
##
##     auc
```

```
cv <- cv.glmnet(x.B,y.B,alpha=1)
lasso.reg <- glmnet(x.B,y.B,alpha=1,family="gaussian",lambda=cv$lambda.min)

lasso.resp <- predict(lasso.reg,newx=x.B)
cor(y.B,lasso.resp)**2
```

```
##              s0
## [1,] 0.7402516
```

Taking into account cross-validation, we do worse compared to linear regression with no tuning.
So what does the tree look like?

```
plot(rpart.Boston);text(rpart.Boston)
```



Doesn't come out that well!
Good news, there are better options for plotting!!!!

http://blog.revolutionanalytics.com/2013/06/plotting-classification-and-regression-trees-with-plotrpart.html
Let's load some new packages:

```
library(rattle)
```

```
## Rattle: A free graphical interface for data mining with R.
## Version 3.3.0 Copyright (c) 2006-2014 Togaware Pty Ltd.
## Type 'rattle()' to shake, rattle, and roll your data.
```

```
library(rpart.plot)
library(RColorBrewer)
library(partykit)
```

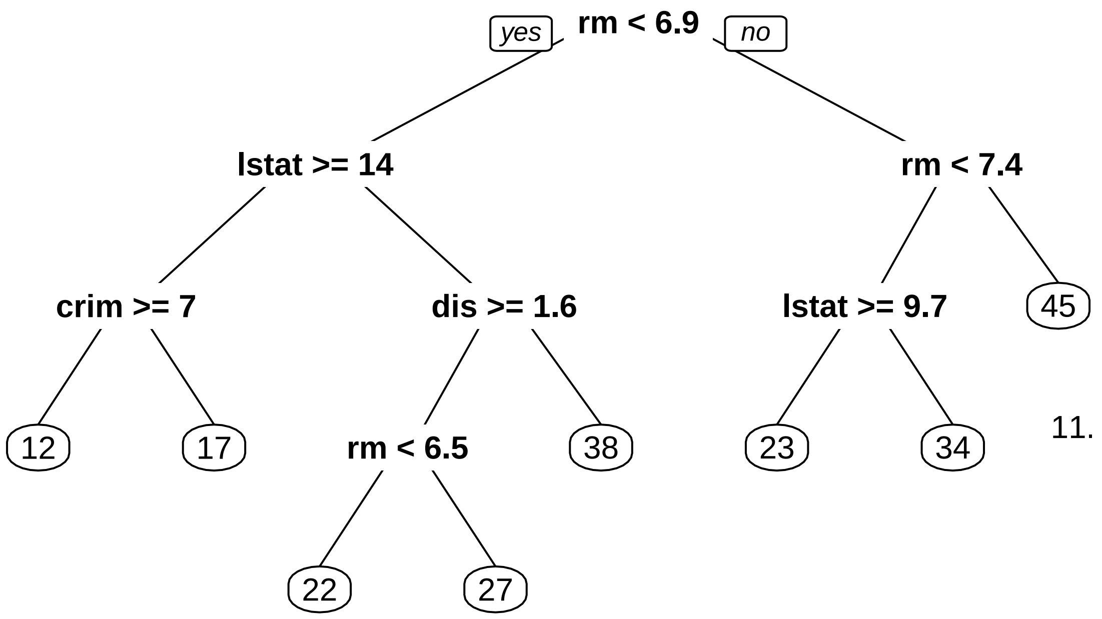
```
##
## Attaching package: 'partykit'
##
```

```
## The following objects are masked from 'package:party':
##
##     ctree, ctree_control, edge_simple, mob, mob_control,
##     node_barplot, node_bivplot, node_boxplot, node_inner,
##     node_surv, node_terminal
```

Note: rattle is package that uses a GUI (think SPSS) for data mining applications check out book: http://www.amazon.com/Data-Mining-Rattle-Excavating-Knowledge/dp/1441998896

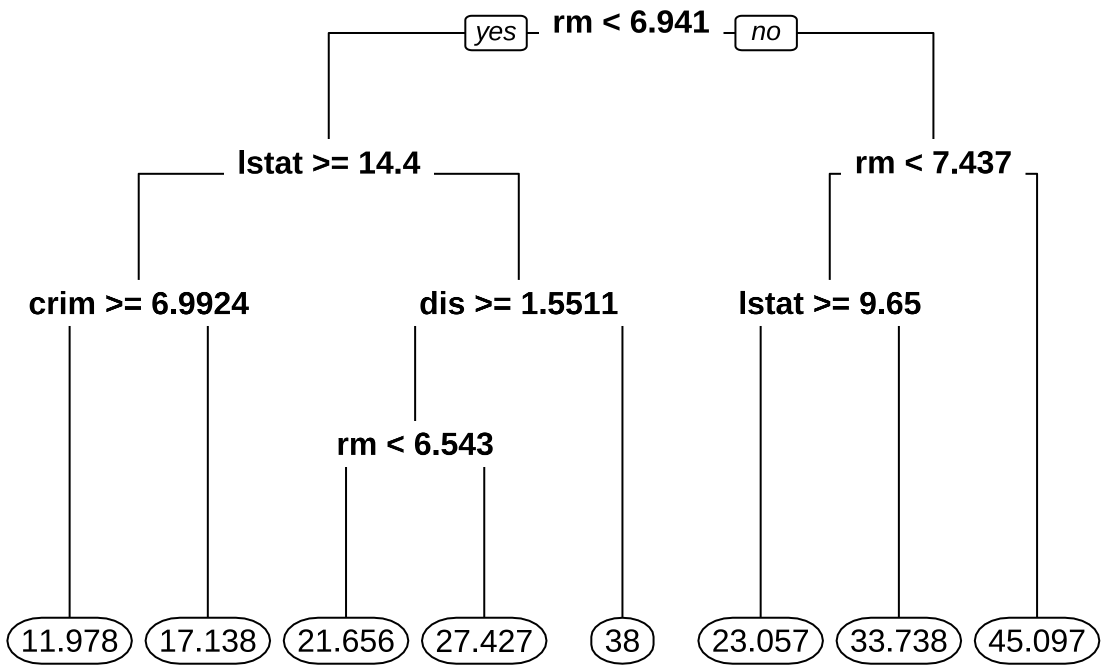
Anyways, lets try some new, prettier plots:

```r
# prp(); from rpart.plot
prp(rpart.Boston);text(rpart.Boston)
```

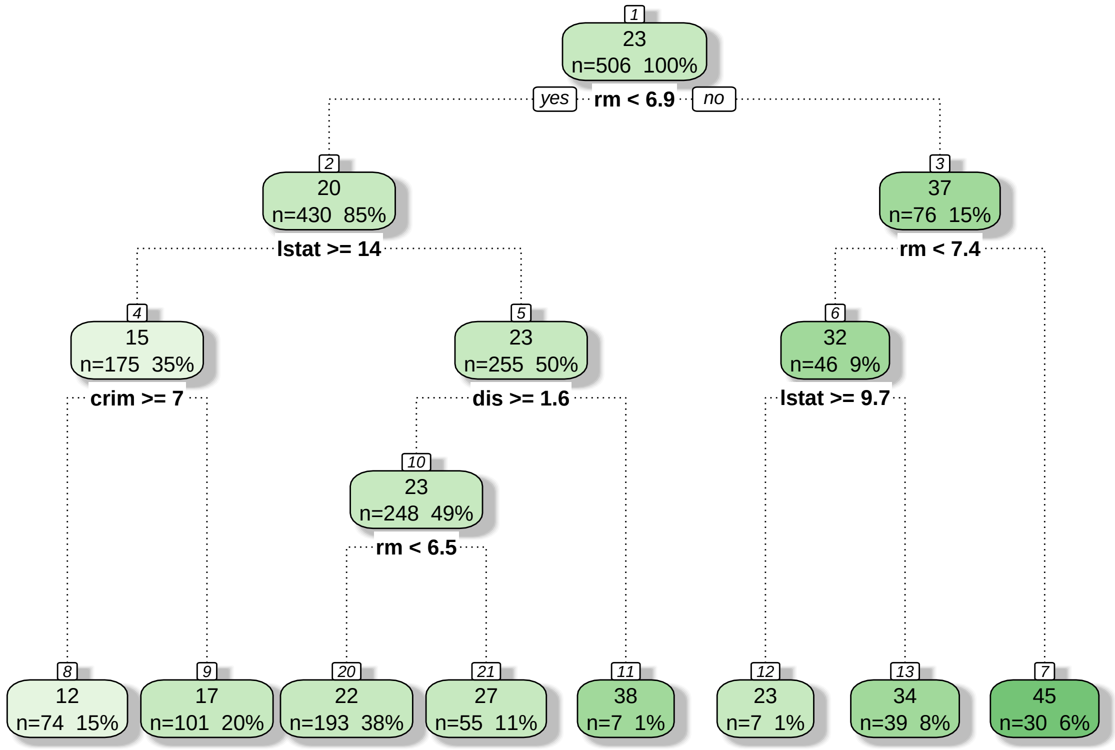


Note, prp() offers many additional capabilities for tweaking the plot For instance:

```r
# ?prp
prp(rpart.Boston,varlen=10,digits=5,fallen.leaves=T)
```



```
#fancyRpartPlot(); from rattle
fancyRpartPlot(rpart.Boston)
```



Rattle 2015−Apr−30 09:56:29 RJacobucci

So what about with conditional inference trees?

```r
ctree.Boston <- ctree(medv ~., data=Boston)

pred2 <- predict(ctree.Boston)
cor(pred2,Boston$medv)**2
```

```
## [1] 0.8746338
```

We do better than rpart, Rsquared = 0.87

Note: the models are not optimizing based on Rsquared, most likely MSE

So what do we think now? Are we happy with results? Remember, decision trees are generally quite robust, so it may not be necessary to check assumptions. – See Table 10.1 ESL

But what about generalizability?

Although not as serious as with SVM for instance, Decision Trees have a propensity to overfit, meaning the tree structure won't generalize well

So let's try just creating a simple Training and Test datasets

```r
train = sample(dim(Boston)[1], dim(Boston)[1]/2) # half of sample
Boston.train = Boston[train, ]
Boston.test = Boston[-train, ]
```

Try linear regression first

```r
lm.train <- lm(medv ~., data=Boston.train)

pred.lmTest <- predict(lm.train,Boston.test)
cor(pred.lmTest,Boston.test$medv)**2
```

```
## [1] 0.7125475
```

Note: we are taking our lm object trained on the train dataset, and using these fixed coefficients to predict values on the test dataset.

In SEM, this is referred to as a tight replication strategy – Bentler
No difference in using a test dataset – both Rsq are 0.74

How about with rpart?

```r
rpart.train <- rpart(medv ~., data=Boston.train)

pred.rpartTest <- predict(rpart.train,Boston.test)
cor(pred.rpartTest,Boston.test$medv)**2
```

```
## [1] 0.6826779
```

Not as good – drops from 0.81 to 0.76 – still better than lm()
caret:

```
ctree.train <- ctree(medv ~., data=Boston.train)

pred.ctreeTest <- predict(ctree.train,Boston.test)
cor(pred.ctreeTest,Boston.test$medv)**2
```

```
## [1] 0.7018021
```

Drops from 0.87 to 0.77

Better than rpart (barely) and lm()

It is worth noting how much more of an effect there was for using a test dataset with the tree methods as compared to lm(), this is pretty typical, and much more important with more "flexible" methods such as random forests, gbm, svm etc…

Classification (categorical outcome)

Two Biggest Things To Remember:
1. Make sure functions outcome variable is categorical; as.factor(outcome)
2. Using predict() changes. Variable across packages

As a baseline, we will use logistic regression.

```
library(ISLR)
data(Default)
head(Default)
```

```
##   default student   balance    income
## 1      No      No  729.5265 44361.625
## 2      No     Yes  817.1804 12106.135
## 3      No      No 1073.5492 31767.139
## 4      No      No  529.2506 35704.494
## 5      No      No  785.6559 38463.496
## 6      No     Yes  919.5885  7491.559
```

```
str(Default)
```

```
## 'data.frame':    10000 obs. of  4 variables:
##  $ default: Factor w/ 2 levels "No","Yes": 1 1 1 1 1 1 1 1 1 1 ...
##  $ student: Factor w/ 2 levels "No","Yes": 1 2 1 1 1 2 1 2 1 1 ...
##  $ balance: num  730 817 1074 529 786 ...
##  $ income : num  44362 12106 31767 35704 38463 ...
```

My favorite function in R is str(), as it gives the class of each variable and other summary characteristics. Most important thing to note is that the "default" variable is already coded as a factor variable, meaning that R now knows it is categorical, and will change the cost function (thus estimator) accordingly.

This is really important because rpart,randomForest and other packages do not automatically detect whether it is a regression or classification problem. If you don't change the outcome variable to its proper class, you could get a suboptimal answer (use the wrong estimator i.e. regression instead of logistic

regression)
Now let's do logistic regression

```
lr.out <- glm(default~student+balance+income,family="binomial",data=Default)
summary(lr.out)
```

```
##
## Call:
## glm(formula = default ~ student + balance + income, family = "binomial",
##     data = Default)
##
## Deviance Residuals:
##     Min       1Q   Median       3Q      Max
## -2.4691  -0.1418  -0.0557  -0.0203   3.7383
##
## Coefficients:
##               Estimate Std. Error z value Pr(>|z|)
## (Intercept) -1.087e+01  4.923e-01 -22.080  < 2e-16 ***
## studentYes  -6.468e-01  2.363e-01  -2.738  0.00619 **
## balance      5.737e-03  2.319e-04  24.738  < 2e-16 ***
## income       3.033e-06  8.203e-06   0.370  0.71152
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## (Dispersion parameter for binomial family taken to be 1)
##
##     Null deviance: 2920.6  on 9999  degrees of freedom
## Residual deviance: 1571.5  on 9996  degrees of freedom
## AIC: 1579.5
##
## Number of Fisher Scoring iterations: 8
```

I always find it much harder to figure out how well I am doing with logistic regression. One of the best ways to assess results in my opinion is the use of receiver operating characteristic curves (ROC curves).

These plots are a balanace of sensitivity and specificity. Ideally the curve gets as close as possible to the upper left corner.

To get this plot, we need to get our predictions from our logistic model.

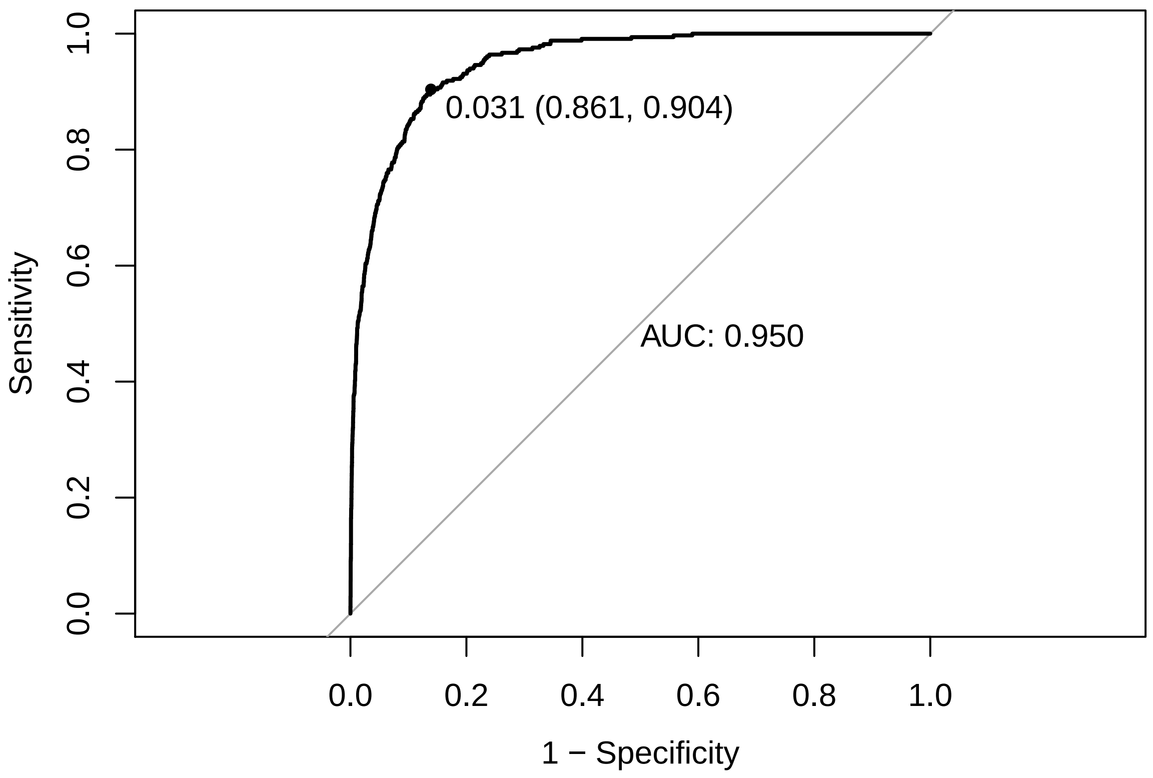
```
glm.probs=predict(lr.out,type="response")
#glm.pred00=ifelse(glm.probs>0.5,1,0)

rocCurve <- roc(Default$default,glm.probs)
pROC::auc(rocCurve)
```

```
## Area under the curve: 0.9496
```

```
pROC::ci.roc(rocCurve)
```

```
## 95% CI: 0.9402-0.959 (DeLong)
```

```r
# quartz()
plot(rocCurve, legacy.axes = TRUE,print.thres=T,print.auc=T)
```



```
##
## Call:
## roc.default(response = Default$default, predictor = glm.probs)
##
## Data: glm.probs in 9667 controls (Default$default No) < 333 cases (Default$default Yes).
## Area under the curve: 0.9496
```

For AUC (area under the curve), values of 0.8 and 0.9 are good (the higher the better)

predict() with missing variables

How about lasso logistic regression?

```r
library(glmnet)
yy = as.numeric(Default$default)
xx = sapply(Default[,2:4],as.numeric)
lasso.out <- cv.glmnet(xx,yy,family="binomial",alpha=1,nfolds=10) #alpha=1 ==  lasso; 0 =
# find best lambda
ll <- lasso.out$lambda.min

lasso.probs <- predict(lasso.out,newx=xx,s=ll,type="response")
```

Results from lasso using CV

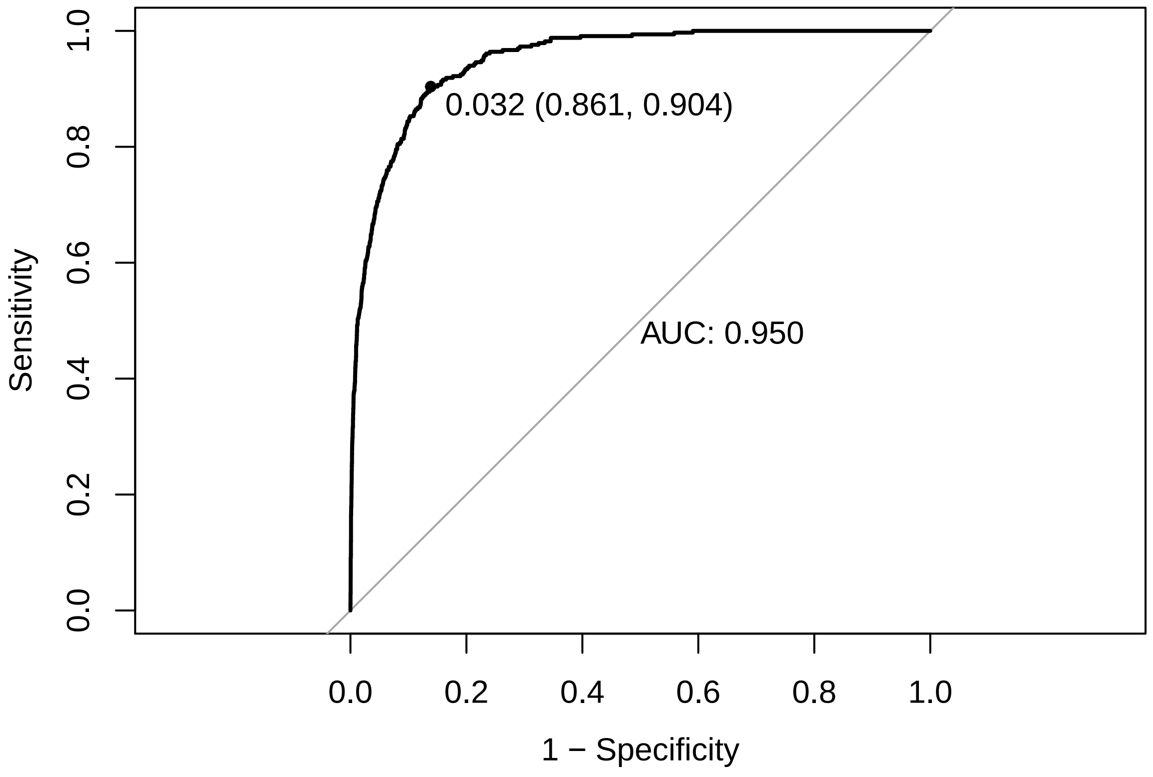
```r
rocCurve.lasso <- roc(Default$default,lasso.probs)
pROC::auc(rocCurve.lasso)
```

```
## Area under the curve: 0.9495
```

```r
pROC::ci.roc(rocCurve.lasso)
```

```
## 95% CI: 0.9401-0.959 (DeLong)
```

```r
# quartz()
plot(rocCurve.lasso, legacy.axes = TRUE,print.thres=T,print.auc=T)
```



```
##
## Call:
## roc.default(response = Default$default, predictor = lasso.probs)
##
## Data: lasso.probs in 9667 controls (Default$default No) < 333 cases (Default$default Yes).
## Area under the curve: 0.9495
```

Almost identical results to logistic regression with no penalization.

What is an Interaction?

aka Strobl 2009 problems