Tree-based Methods

Tree examples

Here we use the cancer registry data, and the infarct data that we considered in the past.

We first read in the cancer registry data

```
dat <- read.csv("Cancer_Registry.csv")
dat <- dat %>% select(-PctSomeCol18_24, -PctEmployed16_Over, -PctPrivateCoverageAlone)

train.ind <- sample(1:nrow(dat), floor(nrow(dat)/2))

dat.train <- dat[train.ind,]
dat.test <- dat[-train.ind,]
```

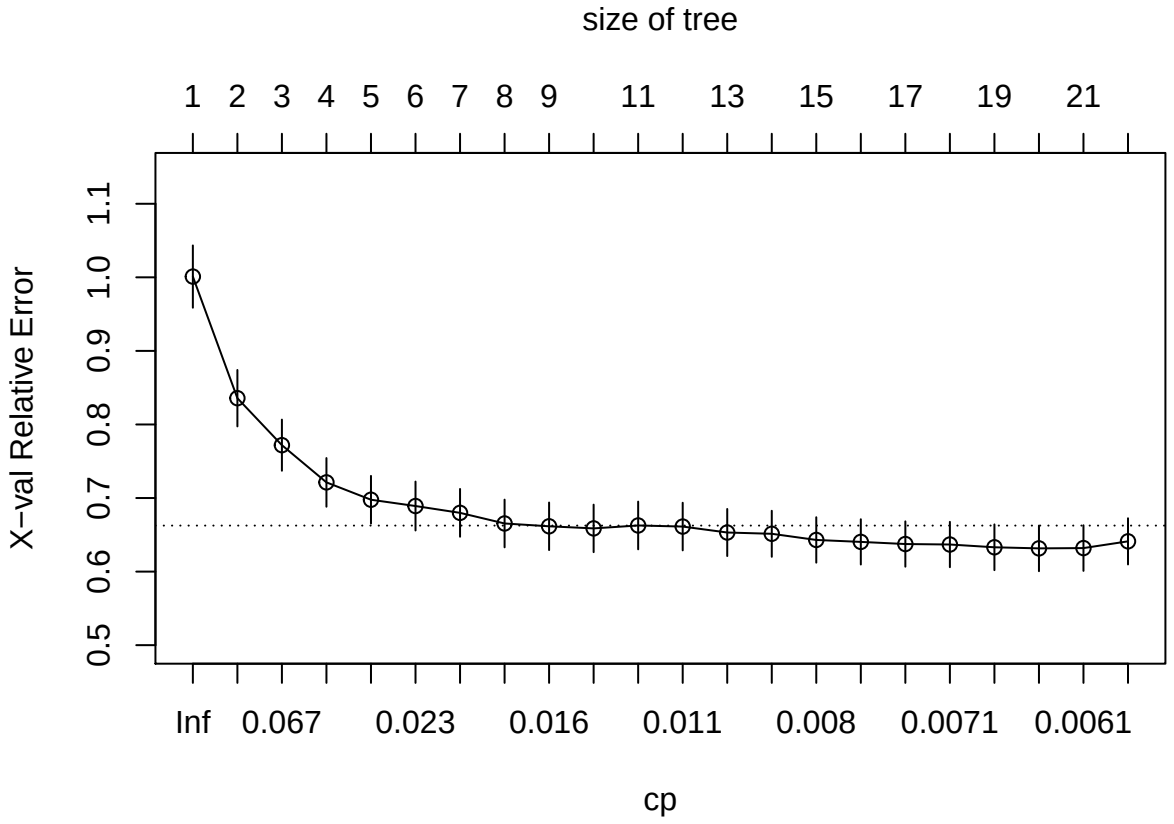
We then fit a single tree

```
### Fitting a single tree

fit <- rpart(formula = TARGET_deathRate ~ .,
             data = dat.train %>% select(-Geography),
             control = rpart.control(cp = 0.005))

cpTab <- printcp(fit)
```

```
##
## Regression tree:
## rpart(formula = TARGET_deathRate ~ ., data = dat.train %>% select(-Geography),
##     control = rpart.control(cp = 0.005))
##
## Variables actually used in tree construction:
##  [1] avgDeathsPerYear       incidenceRate          medIncome
##  [4] PctBachDeg25_Over      PctHS18_24             PctHS25_Over
##  [7] PctPrivateCoverage     PctPublicCoverageAlone PctUnemployed16_Over
## [10] PctWhite               PercentMarried
##
## Root node error: 1150627/1523 = 755.5
##
## n= 1523
##
##           CP nsplit rel error  xerror     xstd
## 1  0.1829277      0   1.00000 1.00103 0.042350
## 2  0.0759538      1   0.81707 0.83572 0.038363
## 3  0.0584003      2   0.74112 0.77184 0.034771
## 4  0.0329896      3   0.68272 0.72127 0.033086
## 5  0.0244961      4   0.64973 0.69768 0.032399
## 6  0.0210615      5   0.62523 0.68912 0.033247
## 7  0.0195071      6   0.60417 0.67985 0.032499
## 8  0.0165005      7   0.58466 0.66540 0.032432
## 9  0.0153196      8   0.56816 0.66158 0.032324
## 10 0.0130370      9   0.55284 0.65871 0.032285
## 11 0.0118490     10   0.53981 0.66269 0.032446
## 12 0.0109417     11   0.52796 0.66122 0.032362
```

```
## 13 0.0093089     12   0.51702 0.65312 0.032004
## 14 0.0083006     13   0.50771 0.65138 0.031284
## 15 0.0077651     14   0.49941 0.64302 0.030896
## 16 0.0073570     15   0.49164 0.64038 0.030809
## 17 0.0071564     16   0.48428 0.63748 0.030813
## 18 0.0069707     17   0.47713 0.63681 0.030833
## 19 0.0063052     18   0.47016 0.63305 0.031088
## 20 0.0061858     19   0.46385 0.63155 0.031023
## 21 0.0060354     20   0.45767 0.63197 0.031012
## 22 0.0050000     21   0.45163 0.64113 0.031411
```

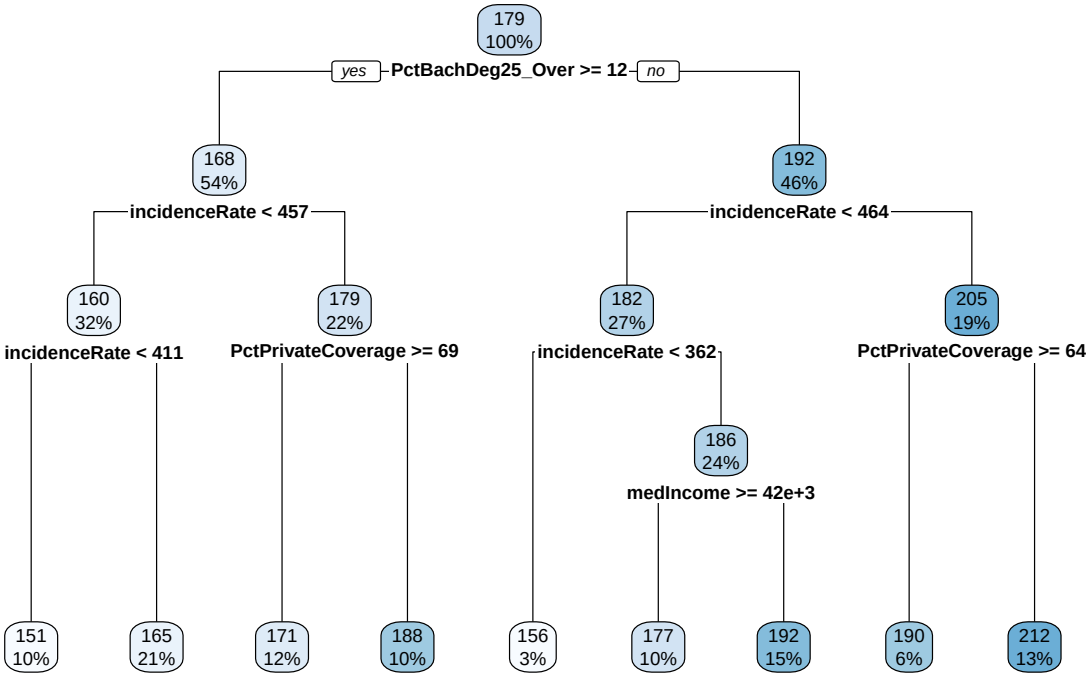
```
plotcp(fit)
```



We can now look at the tree with minimum cv error (and the best tree using the 1se rule)

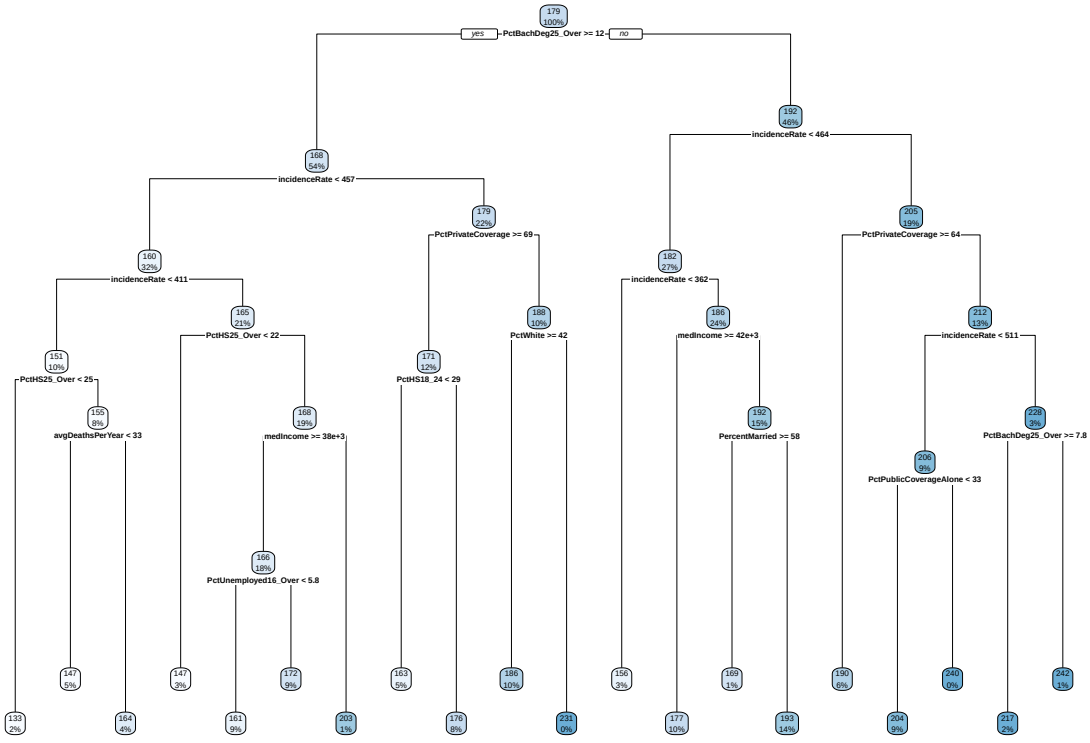
```
minErr <- which.min(cpTab[,4])
tree.best <- prune(fit, cp = cpTab[minErr,1])

good_inds <- which(cpTab[,4]<cpTab[minErr,4]+cpTab[minErr,5])
min_complexity_ind <- good_inds[1]
tree.1se <- prune(fit, cp = cpTab[min_complexity_ind,1])

rpart.plot(tree.1se)
```

```
## Warning: Bad 'data' field in model 'call' (expected a data.frame or a matrix).
## To silence this warning:
##     Call rpart.plot with roundint=FALSE,
##     or rebuild the rpart model with model=TRUE.
```




```
rpart.plot(tree.best)
```

```
## Warning: Bad 'data' field in model 'call' (expected a data.frame or a matrix).
## To silence this warning:
##     Call rpart.plot with roundint=FALSE,
##     or rebuild the rpart model with model=TRUE.
```




```
preds.1se <- predict(tree.1se, dat.test)
preds.best <- predict(tree.best, dat.test)
```

```
(MSE.1tree.1se <- mean((preds.1se - dat.test$TARGET_deathRate)^2))
```

```
## [1] 520.8619
```

```
(MSE.1tree.1best <- mean((preds.best - dat.test$TARGET_deathRate)^2))
```

```
## [1] 490.3799
```

In practice we can probably do better by aggregating trees. We try using a random forest with 5 variables per split

```
#### Fitting a random forest

fit.rf <- randomForest(TARGET_deathRate ~ .,
                       data = dat.train %>% select(-Geography),
                       mtry = 5)
fit.ranger <- ranger(TARGET_deathRate ~ .,
                       data = dat.train %>% select(-Geography),
                       mtry = 5)

preds.rf <- predict(fit.rf, dat.test)
preds.ranger<- predict(fit.ranger, dat.test)$predictions

(MSE.rf <- mean((preds.rf - dat.test$TARGET_deathRate)^2))
```

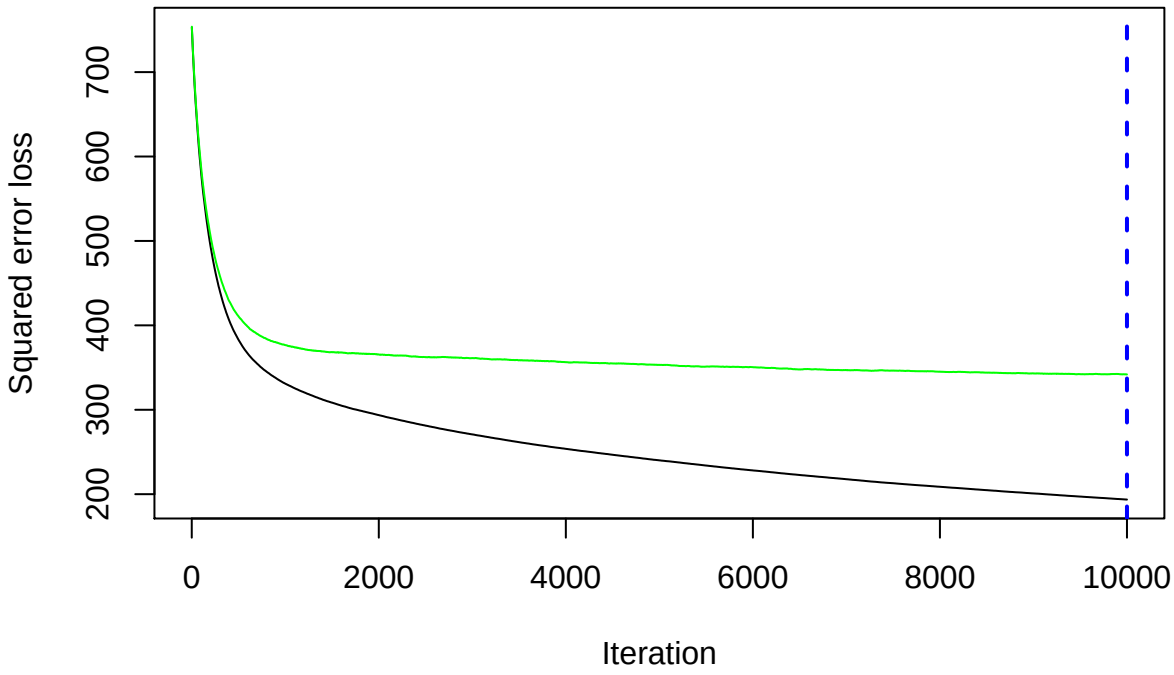
```
## [1] 392.2658
```

```
(MSE.ranger <- mean((preds.ranger - dat.test$TARGET_deathRate)^2))
```

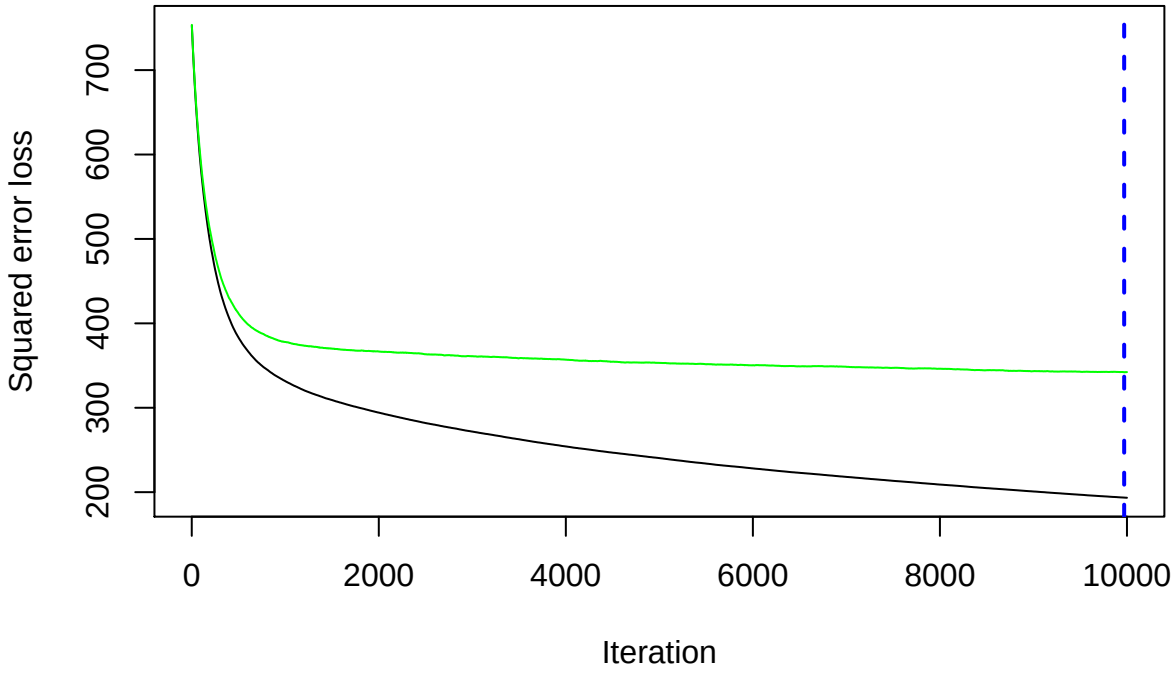
```
## [1] 392.0533
```

We can also try a boosted tree model (in my experience these tend to perform the best, often with trees with only a single split)

```
##### Boosting
set.seed(1)
fit.gbm.1 <- gbm(TARGET_deathRate ~ .,
                 data = dat.train %>% select(-Geography),
                 distribution = "gaussian",
                 n.trees = 10000,
                 interaction.depth = 1,
                 shrinkage = 0.01,
                 cv.folds = 10)

fit.gbm.3 <- gbm(TARGET_deathRate ~ .,
                   data = dat.train %>% select(-Geography),
                   distribution = "gaussian",
                   n.trees = 10000,
                   interaction.depth = 1,
                   shrinkage = 0.01,
                   cv.folds = 10)

best.fit.gbm.1 <- gbm.perf(fit.gbm.1, method = "cv")
```




```
best.fit.gbm.3 <- gbm.perf(fit.gbm.3, method = "cv")
```




```
preds.gbm.1 <- predict(fit.gbm.1, dat.test, ntree = best.fit.gbm.1)
```

```
## Using 10000 trees...
```

```
preds.gbm.3 <- predict(fit.gbm.3, dat.test, ntree = best.fit.gbm.3)
```

```
## Using 9970 trees...
```

```
(MSE.gbm.1 <- mean((preds.gbm.1 - dat.test$TARGET_deathRate)^2))
```

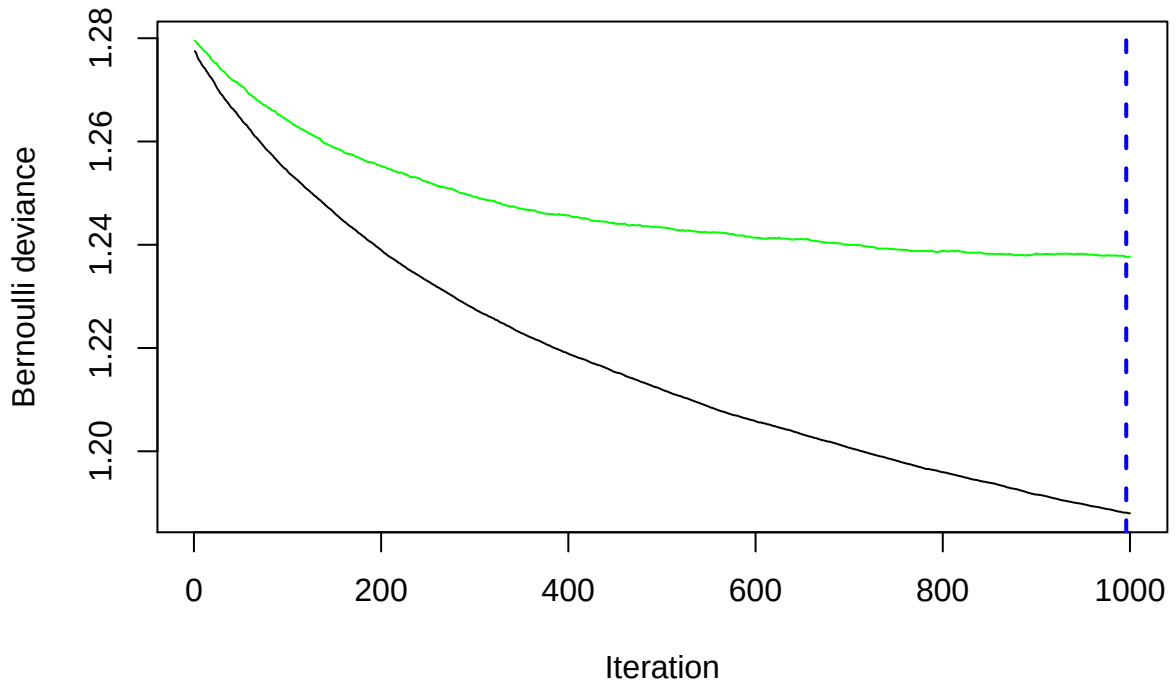
```
## [1] 353.7634
```

```
(MSE.gbm.3 <- mean((preds.gbm.3 - dat.test$TARGET_deathRate)^2))
```

```
## [1] 356.3851
```

We now try applying boosted classification trees to the infarct example

```
##### Trying boosting on the infarct data from before!
set.seed(10)

dat.infarc <- read.table("infarc-data/infarcts.txt", header = T)

dat.infarc.use <- dat.infarc %>%
  select(infarcts, age, educ, income, weight, height, packyrs, alcoh,
         chd, claud, htn, diabetes, ldl, crt) %>%
  na.omit()

train.ind <- sample(1:nrow(dat.infarc.use), floor(nrow(dat.infarc.use)/2))

dat.train <- dat.infarc.use[train.ind,]
dat.test <- dat.infarc.use[-train.ind,]

fit.gbm.1 <- gbm(infarcts ~ .,
                 data = dat.train,
                 distribution = "bernoulli",
                 n.trees = 1000,
                 interaction.depth = 1,
                 shrinkage = 0.01,
                 cv.folds = 10)

best.fit.gbm.1 <- gbm.perf(fit.gbm.1, method = "cv")
```



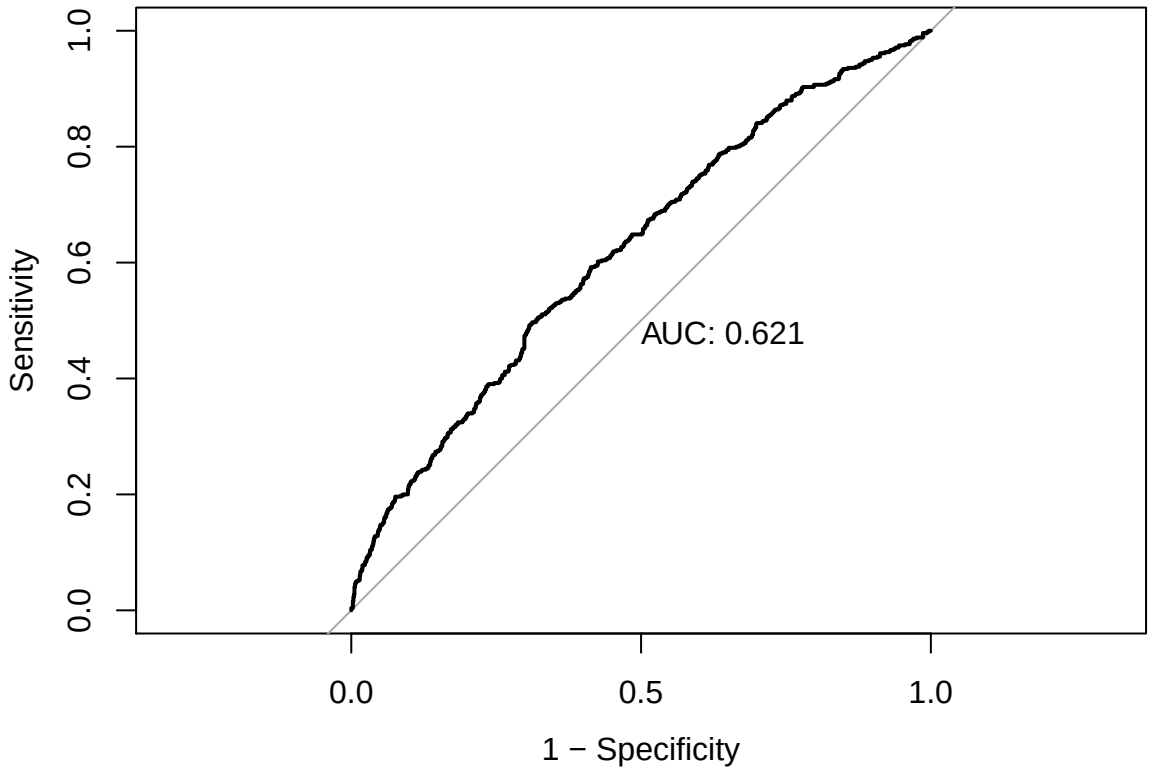

```
preds.gbm.1 <- predict(fit.gbm.1, dat.test, ntree = best.fit.gbm.1)
```

```
## Using 996 trees...
```

```
roc.gbm <- roc(dat.test$infarcts, preds.gbm.1)
```

```
## Setting levels: control = 0, case = 1
## Setting direction: controls < cases
```

```
plot(roc.gbm, legacy.axes = TRUE, print.auc = TRUE)
```



This is roughly the same performance we had with logistic regression