Random Forest Guided Project

Code

Data Preparation

```
adult <- read.csv("adult.data")
# name the columns
names(adult) <- c("age",
                  "workclass",
                  "fnlwgt",
                  "education",
                  "education.num",
                  "marital.status",
                  "occupation",
                  "relationship",
                  "race",
                  "sex",
                  "capital.gain",
                  "capital.loss",
                  "hours.per.week",
                  "native.country",
                  "y")
# make change the high column to a boolean
adult$y <- factor(ifelse(adult$y== " >50K", "yes", "no"))
```

Let's eradicate the " ?" 's

```
any(sapply(adult, function(x) any(x == " ?")))
```

```
[1] TRUE
```

```
which(sapply(adult, function(x) any(x == " ?")))
```

```
     workclass     occupation native.country
             2              7             14
```

```
adult <- adult[!adult$workclass == " ?",]
adult <- adult[!adult$occupation == " ?",]
adult <- adult[!adult$native.country== " ?",]
```

Make a 70/30 split

```
set.seed(5333)
 k = sample(1:30161, size = 9048, rep =F)
 adult.train <- adult[-k,]
 adult.test <- adult[k,]
```

Model Building

use the randomforest function

```
# fit a random forest
library(randomForest)
```

```
randomForest 4.7-1.1
```

```
Type rfNews() to see new features/changes/bug fixes.
```

```
fit.rf <- randomForest(y ~ ., data = adult.train, ntree = 100, mtry = 4, nodesize = 10)
# confusionmatrix for the random forest
library(caret)
```

```
Loading required package: ggplot2
```

```

Attaching package: 'ggplot2'
```

```
The following object is masked from 'package:randomForest':

    margin
```

```
Loading required package: lattice
```

```r
adult.test$rf0_yhat = predict(fit.rf, newdata = adult.test, type = "response")
cmat_0 <- confusionMatrix(adult.test$rf0_yhat, adult.test$y)
```

Given that I have 14 features, I am going to make the mtry = 4. We will start with 100 trees to save RAM.

I like to make a function to display all the accuracy metrics.

```r
# display for accuracy, precision, recall, f1score
table1 <- function(confusion_matrix,model_name){
  confusion_matrix -> tab
  tab[1,1] -> tn
  tab[2,2] -> tp
  tab[1,2] -> fp
  tab[2,1] -> fn
  a = (tp+tn)/(tp+tn+fp+fn)
  p = tp/(tp+fp)
  r = tp/(tp+fn)
  sprintf("%s || Accuracy: %f | Precision: %f | Recall(TPR): %f",model_name,a,p,r)
  return(list("model.name" = model_name,"accuracy" = a, "precision" = p, "recall" = r))
}

table1(cmat_0$table,"initial randForest")
```

```
$model.name
[1] "initial randForest"

$accuracy
[1] 0.8588638

$precision
[1] 0.607489

$recall
[1] 0.7813031
```

Optimization

```r
# let's try different things
v.ns = c(5,10,20) # put into node size
v.nt = c(100,500,1000) # put into ntrees
v.mt = c(3,4,5) # put into mtry
```

```r
set.seed(5333)
# intialize the results
ROW <- data.frame(ntree = numeric(), mtry = numeric(), nodesize = numeric(), accuracy = numer

# run the long loop
for(i in v.ns){
  for(j in v.nt){
    for(k in v.mt){
      # feed these into the forest
      atm.rf <- randomForest(y ~ ., data = adult.train, ntree = j, mtry = k, nodesize = i)
      yhat.atm = predict(atm.rf, newdata = adult.test, type = "response")
      cmat.atm <- confusionMatrix(yhat.atm, adult.test$y)
      table1(cmat.atm$table,"model at the moment")$accuracy -> acc
      ROW <- rbind(ROW, data.frame(ntree = j, mtry = k, nodesize = i, accuracy = acc))
    }
  }
}

# get the best model
opt <- ROW[which.max(ROW$accuracy), ]
print(opt)
```

```
   ntree mtry nodesize  accuracy
13   500    3       10 0.8635057
```

```r
bestfit.rf <- randomForest(y ~ ., data = adult.train, ntree = 500, mtry = 3, nodesize = 10)
yhat.atm = predict(bestfit.rf, newdata = adult.test, type = "response")
cmat.atm <- confusionMatrix(yhat.atm, adult.test$y)
table1(cmat.atm$table,"best fit random forest")
```

```
$model.name
[1] "best fit random forest"

$accuracy
[1] 0.8629531
```

```
$precision
[1] 0.6167401

$recall
[1] 0.7909605
```

```
table1(cmat_0$table,"initial randForest")
```

```
$model.name
[1] "initial randForest"

$accuracy
[1] 0.8588638

$precision
[1] 0.607489

$recall
[1] 0.7813031
```

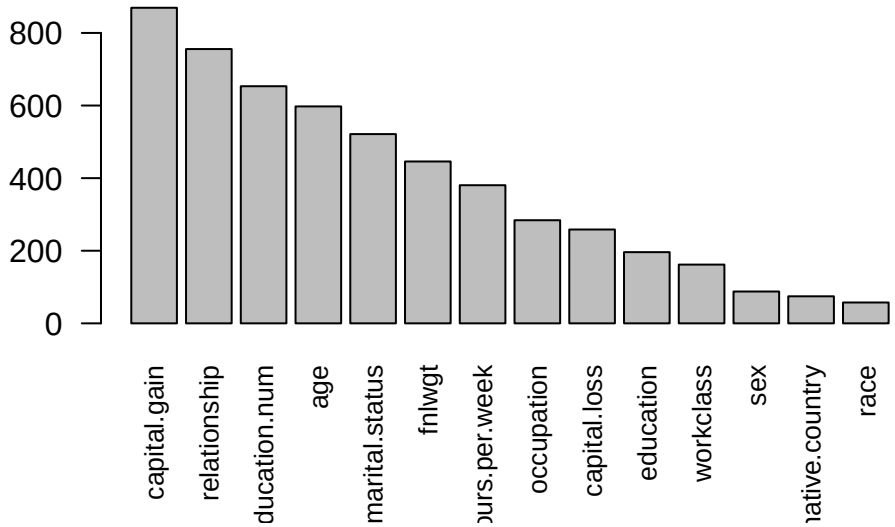
It looks as though the optimal rf model does preform better than our initial rf model. Note that every preformance metric is about 0.01 higher.

Analysis

```
# variables importance test is here
library(vip)
```

```
Attaching package: 'vip'

The following object is masked from 'package:utils':

    vi
```

```
vi(bestfit.rf) -> X
barplot(X$Importance,names.arg = X$Variable, horiz = F,las = 2, cex.names = 0.8)
```



Here we can see the top preforming predictors were "capital gain", relationship status, education, age, "final weight", and hours worked per week.

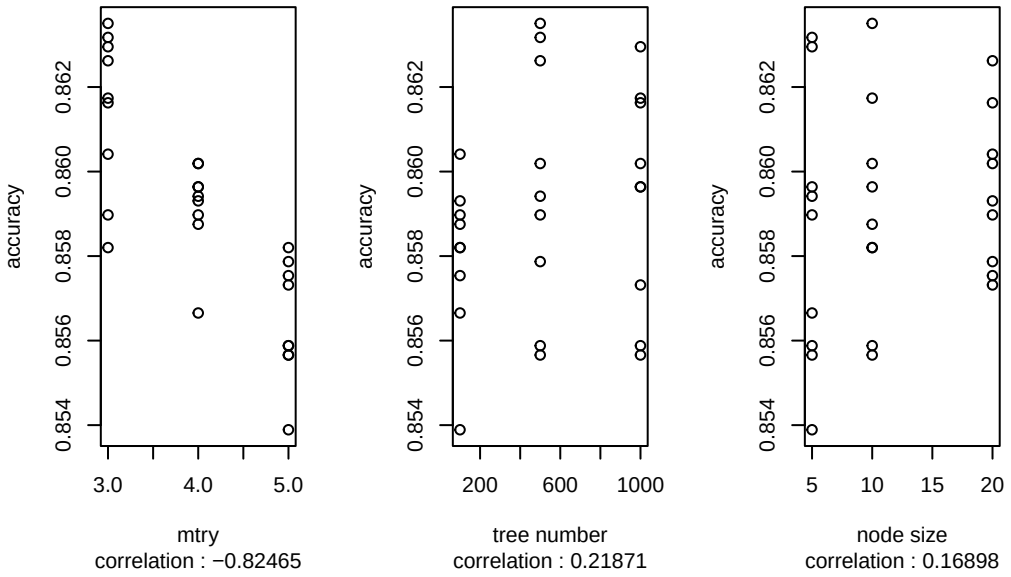
```
# tree training
library(rpart)
library(rpart.plot)
simple.tree <- rpart(y ~., data = adult.train, method = "class")
# get performance
tree.yhat <- predict(simple.tree, newdata = adult.test, type = "class")
confusionMatrix(tree.yhat,adult.test$y) -> cmat.stree
table1(cmat.stree$table, "simple tree")
```

```
$model.name
[1] "simple tree"

$accuracy
[1] 0.8423961

$precision
[1] 0.5088106

$recall
[1] 0.7878581
```

Looks like the random forest gets a better accuracy than the a basic decision tree although by 0.02 factor. I think it is important to remember that because random forest will generate a forest of decision trees while rpart() or tree() produces a single tree; it is only intuitive

to assume that random forest preforms better than a single decision tree. Because of this, overfitting is less likely and this reduces variance.

```
# visualize
par(mfrow = c(1,3))
COR = round(cor(ROW$mtry,ROW$accuracy),5)
plot(ROW$mtry,ROW$accuracy, xlab = "mtry", ylab = "accuracy",
     sub = paste("correlation :",COR))
COR = round(cor(ROW$ntree,ROW$accuracy),5)
plot(ROW$ntree,ROW$accuracy, xlab = "tree number", ylab = "accuracy",
     sub = paste("correlation :",COR))
COR = round(cor(ROW$nodesize,ROW$accuracy),5)
plot(ROW$nodesize,ROW$accuracy, xlab = "node size", ylab = "accuracy",
     sub = paste("correlation :",COR))
```




```
library(kableExtra)
kable(sort_by(ROW,ROW$accuracy),caption = "comparison of decision tree perfomance")
```

Table 1: comparison of decision tree perfomance

	ntree	mtry	nodesize	accuracy
3	100	5	5	0.8538904
6	500	5	5	0.8556587
18	1000	5	10	0.8556587

	ntree	mtry	nodesize	accuracy
9	1000	5	5	0.8558798
15	500	5	10	0.8558798
2	100	4	5	0.8566534
27	1000	5	20	0.8573165
21	100	5	20	0.8575376
24	500	5	20	0.8578691
10	100	3	10	0.8582007
12	100	5	10	0.8582007
11	100	4	10	0.8587533
1	100	3	5	0.8589744
23	500	4	20	0.8589744
20	100	4	20	0.8593059
5	500	4	5	0.8594164
8	1000	4	5	0.8596375
17	1000	4	10	0.8596375
14	500	4	10	0.8601901
26	1000	4	20	0.8601901
19	100	3	20	0.8604111
25	1000	3	20	0.8616269
16	1000	3	10	0.8617374
22	500	3	20	0.8626216
7	1000	3	5	0.8629531
4	500	3	5	0.8631742
13	500	3	10	0.8635057

While the accuracy differences ranged from 0.8538 to 0.8635, it does seem that `mtry` has the biggest impact here when it is at the lowest value. The other tuning metrics seem to have no real affect on accuracy as their correlation is closer to zero juxtapose to `mtry` which is around -0.8.

Reporting

Random forest is a powerful machine learning algorithm that leverages multiple decision trees. This ensemble approach results in more accurate and robust predictions compared to a single decision tree. Additionally, Random Forest enables us to assess the relative importance of different predictors, helping us identify the key drivers of the outcome variable and gain valuable insights from our data analysis.

The biggest drawback of random forest is that is computationally expensive. Because of how long it took to render this document, we would need more time and heftier equipment to find precise tuning parameters. (I think a Bayesian approach utilizing C++ based systems like RStan could be considered if time is an issue). Random forest is a "forest" of trees meaning many decision trees (at most I rendered over 4900 trees in total) here are being render by the program.