Stat 432 Homework 11

Assigned: Nov 8, 2021; Due: 11:59 PM CT, Nov 16, 2021

Contents

Question 1: Fitting and Tuning Trees

We will use the Social Network Ads data from HW 10. This time, we will use all three covariates: age, gender and salary to predict the purchase outcome. Fit a 5-fold cross-validation CART model and answer the following question:

```r
# Loading the Social Network Ads data
data <- read.csv("Social_Network_Ads.csv")

# Converting 'Gender' and 'Purchased' to factor variables
data$Purchased = as.factor(data$Purchased)
data$Gender = as.factor(data$Gender)
```

```r
set.seed(667346304)
# Fitting a 5-fold cross-validation CART model
library(rpart)
rpart.fit = rpart(
  as.factor(Purchased) ~ Age + EstimatedSalary + Gender,
  data = data[, -1],
  control = rpart.control(xval = 5)
)
```

- [5 Points] How large (# of terminal nodes) is the tree that gives the smallest cross-validation error?

```r
# Index corresponding to min xerror
min_ind = which.min(rpart.fit$cptable[, 4])
# CP corresponding to min xerror
cp_min = rpart.fit$cptable[min_ind, 1]

# Tree with smallest cross-validation error
prunedtree = prune(rpart.fit, cp = cp_min)
prunedtree
```

```
## n= 400
##
## node), split, n, loss, yval, (yprob)
##       * denotes terminal node
##
## 1) root 400 143 0 (0.6425000 0.3575000)
##   2) Age< 42.5 285  46 0 (0.8385965 0.1614035)
##     4) EstimatedSalary< 90500 241   9 0 (0.9626556 0.0373444) *
##     5) EstimatedSalary>=90500 44   7 1 (0.1590909 0.8409091) *
##   3) Age>=42.5 115  18 1 (0.1565217 0.8434783) *
```

From above result, we can see that the number of terminal nodes (denoted by *) in the tree that gives the smallest cross-validation error is 3.

- [10 Points] If we use the 1sd rule, obtain and plot the tree corresponding to that `cp` value.

```
rpart.fit$cptable
```

```
##          CP nsplit rel error    xerror       xstd
## 1 0.5524476      0 1.0000000 1.0000000 0.06702990
## 2 0.2097902      1 0.4475524 0.4685315 0.05222681
## 3 0.0100000      2 0.2377622 0.2587413 0.04052175
```

```
# Min xerror value
xerror_min = rpart.fit$cptable[min_ind, 4]
# Adding 1-SD to the smallest cross-validation error
xerror_1sd=xerror_min+ rpart.fit$cptable[3,5]
```

```
# Added value of 1-SD + smallest cross-validation error
xerror_1sd
```

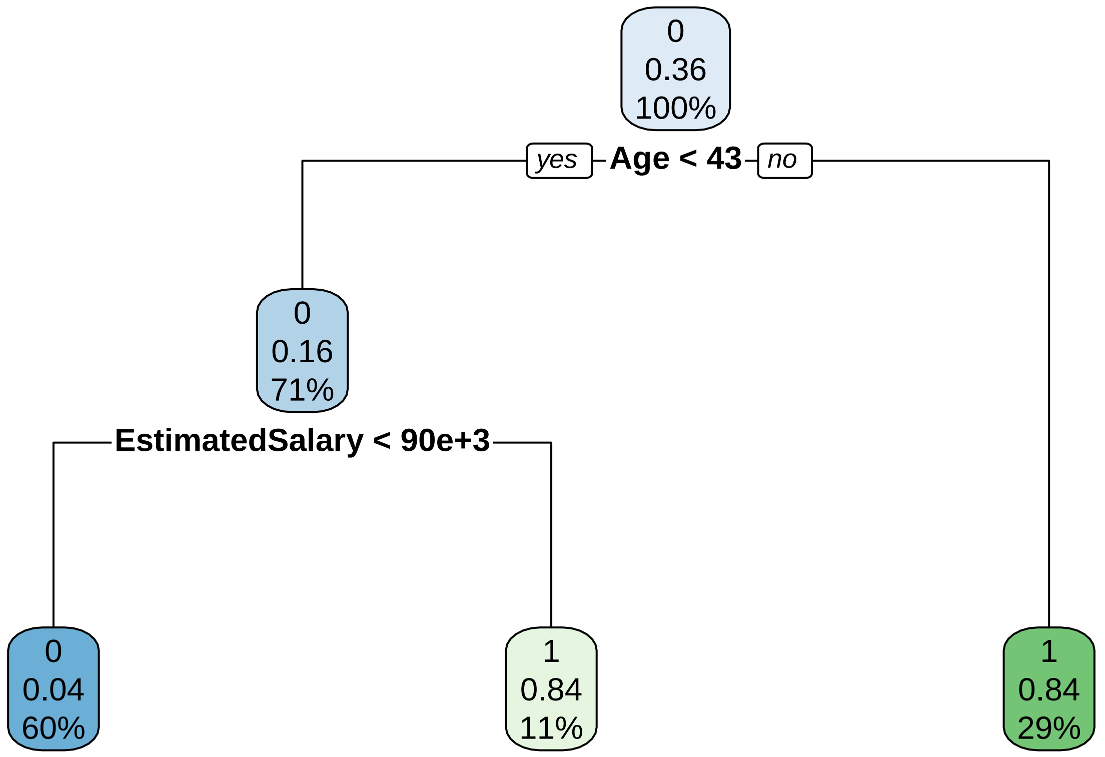
```
## [1] 0.299263
```

```
# Get the middle value between the one below and above xerror_1sd

cptarg = sqrt(rpart.fit$cptable[3,1]*rpart.fit$cptable[2,1])
cptarg
```

```
## [1] 0.04580286
```

The cp value corresponding to the 1sd rule is 0.04580286.

```
# Tree corresponding to `cptarg`
prunedtree2 = prune(rpart.fit,cp=cptarg)

library(rpart.plot)
rpart.plot(prunedtree2)
```



Above is the plot the tree corresponding to the `cp` value obtained from 1sd rule.

- [5 Points] What is the first variable that is being used to split? and what is the splitting rule?

```
# Pruned tree
prunedtree2
```

```
## n= 400
##
## node), split, n, loss, yval, (yprob)
##       * denotes terminal node
##
## 1) root 400 143 0 (0.6425000 0.3575000)
##   2) Age< 42.5 285  46 0 (0.8385965 0.1614035)
##     4) EstimatedSalary< 90500 241   9 0 (0.9626556 0.0373444) *
##     5) EstimatedSalary>=90500 44   7 1 (0.1590909 0.8409091) *
##   3) Age>=42.5 115  18 1 (0.1565217 0.8434783) *
```

Age is the first variable that is being used to split. The splitting rule is Age <42.5 Yes (left-subtree) or No (right-subtree)

- [5 Points] Based on this plot, for a new subject with age 35, and salary 10,000, what is the predicted outcome?

The Age value of 35< 42.5, hence we move down the left subtree. Comparing salary of 10,000 with 90e+3,as 10000< 90e+3 we move along the left subtree leading to class=0. Hence, the predicted outcome using the pruned tree is class 0.

Question 2: Fitting and Tuning Random Forests

The goal of this question is to learn how to read documentations of a new package yourself and be able to successfully implement it. The original `randomForest` package is a little bit slow computationally. There is a faster package `ranger`, however, it names the parameters slightly differently. Carefully read the `ranger()` function (starting from page 15) in this documentation, and **figure out what are the corresponding parameter names for `mtry` and `nodesize`**, and also read the `caret` package random forest related documentation here to find out how to specify the `train()` function and perform tuning using the `ranger` package. Then, complete the following tasks:

- [5 Points] Load the Cleveland Clinic Heart Disease Data from HW8. Process the outcome variable `num` so that `num` >0 is labeled as 1, and 0 otherwise (this is the same as HW8).

```
# Loading Cleveland Clinic Heart Disease Data
heart = read.csv("processed_cleveland.csv")
# Processing `num` so that `num` > 0 is labeled as 1, and 0 otherwise
heart$num[which(heart$num > 0)] = 1
```

- [5 Points] For `ca` and `thal`, remove any observation that contains `"?"`. Then convert both variables as factors. You want to consider using the `as.factor()` function.

```
# Removing observation that contains "?" for `ca` and `thal` variables
heart=heart[heart$ca != '?', ]
heart=heart[heart$thal != '?', ]

# Converting `ca` and `thal` variables as factors
heart$ca=as.factor(heart$ca)
heart$thal=as.factor(heart$thal)
```

- [5 Points] Construct a grid of tuning parameters using `expand.grid()` with some `mtry` and `nodesize` values. Pick 2 `mtry` values and 3 `nodesize` values at your choice. In addition, the package requires you to specify a variable `splitrule = "gini"` in this grid.

```
# Loading the libraries
library(ranger)
library(e1071)
library(caret)
```

```
## Loading required package: ggplot2

## Loading required package: lattice
```

```
# Constructing grid of tuning parameters
grid <- expand.grid(
    mtry = c(3, 4),
    splitrule = "gini",
    min.node.size = c(5, 10, 20)
  )
```

- [5 Points] Construct `trControl()` to be 10 fold cross-validation

```r
# Constructing `trControl()` to be 10 fold cross-validation
control <- trainControl(method = "cv",  number = 10)
```

- [5 Points] In your `train()` function, specify two arguments `num.trees = 300` and `respect.unordered.factors = "partition"`.

```r
set.seed(667346304)

# Constructing `train()` function
ranger.forest <- train(
  y ~ .,
  method = "ranger",
  data = data.frame("x" = heart[, -14], "y" = as.factor(heart[, 14])),
  tuneGrid = grid,
  trControl = control,
  num.trees = 300,
  respect.unordered.factors = "partition"
)
```

- [5 Points] What is the best tuning parameter? You may want to iterate your process to narrow down to a good range of tuning. The `ranger` package utilize multiple cores of your CPU. Hence you may do this process at your computer's capacity.

```r
# Best tuning parameters
ranger.forest$bestTune
```

```
##   mtry splitrule min.node.size
## 2    3      gini            10
```

The best tuning parameters obtained are: mtry=3 and min.node.size=5 with “gini” as splitrule.

- [5 Points] Provide a statement to explain why we want to consider `respect.unordered.factors = "partition"`, and how is it different from its default value.

Unordered factor covariates are handled in 3 ways: ‘ignore’(default value), ’order’ and ’partition’. For ’ignore’ all factors are regarded ordered. For ’order’ and 2-class classification the factor levels are ordered by their proportion falling in the second class. The ordered treatment of variable is good if all of the categorical variables were known to have ordered relations with the outcome. This is not the case here. In ’partition’ option, all possible 2-partitions are considered for splitting (without ordering) hence it becomes the most appropriate choice for our case.

The default ’ignore’ in fact weakens the expressiveness of the ranger model (which is why it is faster). The default is making things easy for ranger at the expense of model quality.

Question 3: A Simulation Study

We are going to use the following code to generate data and perform simulation studies to analyze the effect of `nodesize`. Note that this is a **regression** question and we are going to use the `randomForest` package to complete this question. Complete the following task:

- [15 Points] Setup a simulation study to analyze the performance of random forest. For the idea of simulation, please review HW4, HW6 and HW7. Use the following setting:
  - Your simulation should repeat `nsim = 100` times
  - Within each simulation, you should generate training and testing data using the following code, and evaluate the prediction mean squared error of random forests
  - Set `mtry` = 1 and `ntree` = 300 for all simulations
  - Use a grid of `nodesize` values: `c(1, 5, 10, 20, 30, 40)`
  - Leave all other tuning parameters as default

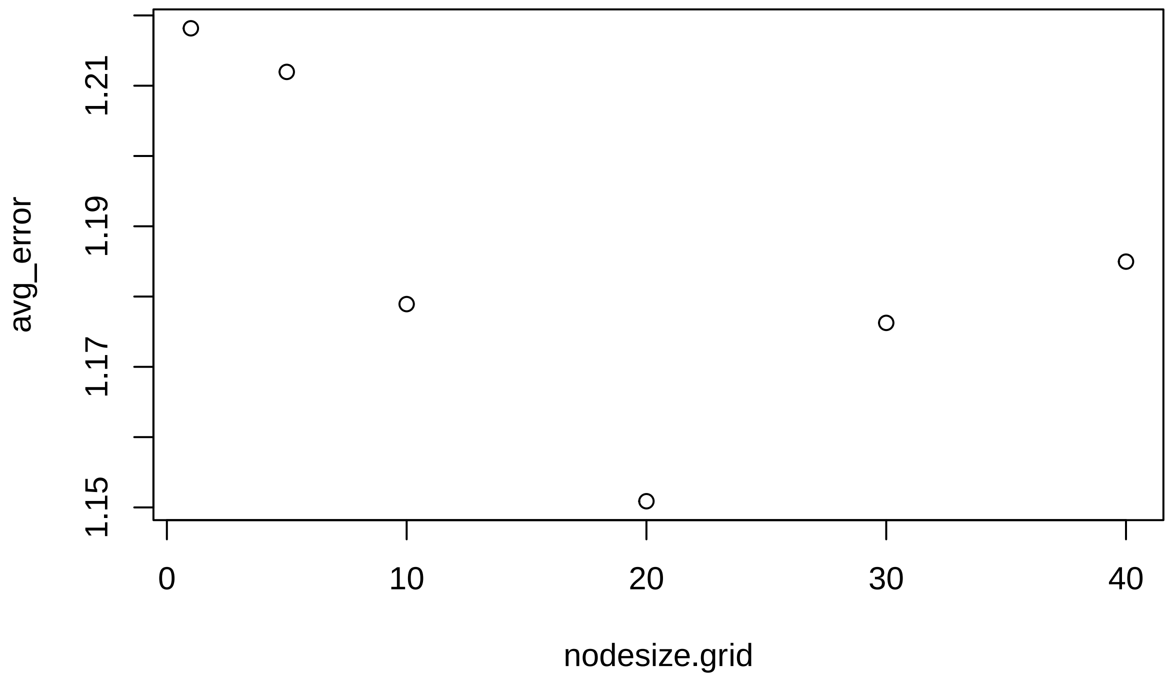
```r
# Setting the seed
set.seed(667346304)

# Loading libraries
library(MASS)
library(randomForest)
```

```
## randomForest 4.6-14

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

##
## Attaching package: 'randomForest'

## The following object is masked from 'package:ggplot2':
##
##     margin

## The following object is masked from 'package:ranger':
##
##     importance
```

```r
# Setting parameter values
nsim = 100
mtry.val = 1
ntree.val = 300
nodesize.grid = c(1, 5, 10, 20, 30, 40)

# Creating matrix to store mse values for different node size
mse =  matrix(NA, nsim, length(nodesize.grid))

for (index in 1:length(nodesize.grid)) {
  for (i in 1:nsim) {
    n = 200
    X = mvrnorm(n, c(0, 0), matrix(c(1, 0.5, 0.5, 1), 2, 2))
    y = rnorm(n, mean = X[, 1] + X[, 2])
    XTest = mvrnorm(n, c(0, 0), matrix(c(1, 0.5, 0.5, 1), 2, 2))
    yTest = rnorm(n, mean = XTest[, 1] + XTest[, 2])

    rf.fit = randomForest(X,
                          y,
                          ntree = ntree.val,
                            mtry = mtry.val,
                            nodesize = nodesize.grid[index])
    mse[i, index] = mean((yTest - predict(rf.fit, XTest)) ^ 2)
  }
}
```

- [5 Points] Make a plot of your grid of `nodesize` values against the averaged error across all simulations.

```
# Mean of mse estimates
avg_error = colMeans(mse)
# Creating data frame of avg MSE errors and corresponding node size values
df = data.frame(cbind(nodesize.grid, avg_error))
# Plot
plot(nodesize.grid, avg_error)
```



- [10 Points] What do you observe on this plot? Can you explain why this is happening?

We observe that the minimum average MSE is obtained for the node size=20.The minimum average MSE first decreases with increase in nodesize and then starts to increase with increase in nodesize.

Increasing node size initially decreases the avg MSE as reducing complexity of the trees helps to get a good model with only relevant features being considered while making the classification decision. However, when we keep on increasing the node size, the tree/model becomes far too simpler to be any useful in performing classification task. Such a simpler model, with a few predictors,is not able to capture the trends to make correct classification decisions.