HW7-Q10.1

2025-02-25

Question 10.1:

Using the same crime data set uscrime.txt as in Questions 8.2 and 9.1, find the best model you can using

(a) a regression tree model, and

(b) a random forest model.

In R, you can use the tree package or the rpart package, and the randomForest package. For each model, describe one or two qualitative takeaways you get from analyzing the results (i.e., don't just stop when you have a good model, but interpret it too).

Answer:

Codes are referenced from office hour Feb 24th, AIs, and https://www.geeksforgeeks.org/oob-errors-for-random-forests-in-scikit-learn/.

a) regression tree model:

Load the data into R:

```r
rm(list=ls())
set.seed(42)
uscrime <- read.table("uscrime.txt", header=TRUE)
```

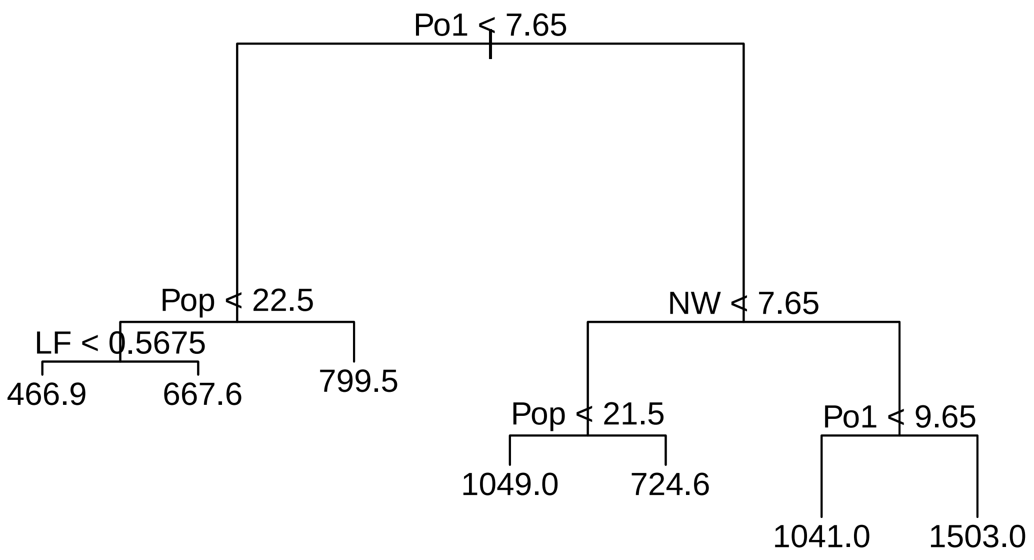
Fit the regression tree function to the crime data:

```r
library(tree)
uscrime_tree <- tree(Crime~., data=uscrime)
summary(uscrime_tree)
```

```
##
## Regression tree:
## tree(formula = Crime ~ ., data = uscrime)
## Variables actually used in tree construction:
## [1] "Po1" "Pop" "LF"  "NW"
## Number of terminal nodes:  7
## Residual mean deviance:  47390 = 1896000 / 40
## Distribution of residuals:
##     Min.  1st Qu.   Median     Mean  3rd Qu.     Max.
## -573.900  -98.300   -1.545    0.000  110.600  490.100
```

Visualize the regression tree, see how many points are in each leaf, and see which leaf each point is in:

```
plot(uscrime_tree)
text(uscrime_tree)
```



The plot shows that there are 4 variables are used: Po1, Pop, LF, and NW; and 7 nodes.

Let's see how the tree was split:

```
uscrime_tree$frame
```

```
##        var  n        dev      yval splits.cutleft splits.cutright
## 1      Po1 47 6880927.66  905.0851          <7.65           >7.65
## 2      Pop 23  779243.48  669.6087          <22.5           >22.5
## 4       LF 12  243811.00  550.5000        <0.5675         >0.5675
## 8   <leaf>  7   48518.86  466.8571
## 9   <leaf>  5   77757.20  667.6000
## 5   <leaf> 11  179470.73  799.5455
## 3       NW 24 3604162.50 1130.7500          <7.65           >7.65
## 6      Pop 10  557574.90  886.9000          <21.5           >21.5
## 12  <leaf>  5  146390.80 1049.2000
## 13  <leaf>  5  147771.20  724.6000
## 7      Po1 14 2027224.93 1304.9286          <9.65           >9.65
## 14  <leaf>  6  170828.00 1041.0000
## 15  <leaf>  8 1124984.88 1502.8750
```

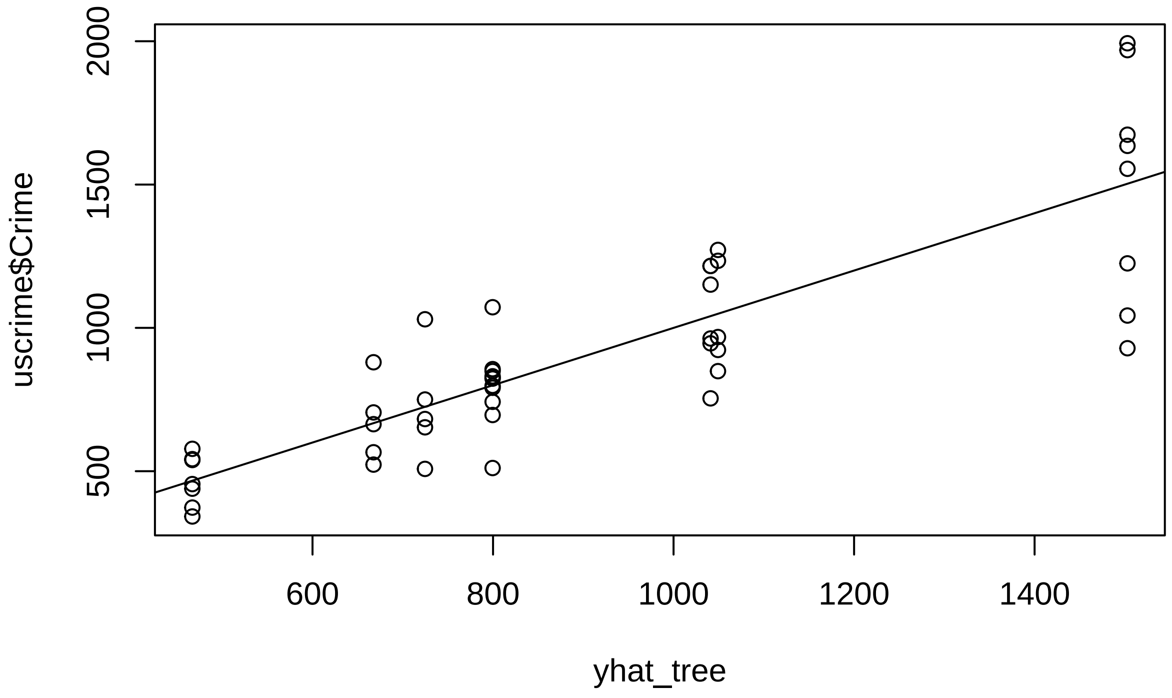
Explanation:

- **var**: indicates the variables used for splitting at each node.
- **leaf**: indicates the node is the terminal node (leaf).
- **n**: the number of observations (data points) that fall into each node.
- **dev**: represents the deviance of the node, which is a measure of the node's impurity or heterogeneity. In regression trees, deviance is typically the sum of squared residuals for the data points within the node. Lower deviance values indicate that the data points within the node are more similar in terms of their response variable ("Crime" in this case).
- **yval**: is the predicted value for the node. For regression trees, it's usually the average value of the response variable for the data points within the node.
- **splits.cutleft**: indicates the condition for observations going to the left branch.
- **splits.cutright**: indicates the condition for observations going to the right branch.

Next, let's plot the tree to see how it looks like:

```r
yhat_tree <- predict(uscrime_tree)

#Plot to visualize the relationship between the predicted crime rates (yhat_tree) from your regression
plot(yhat_tree, uscrime$Crime)
abline(a = 0, b = 1)
```



This chart shows how well we're predicting crime. The closer the dots are to the line, the better we're doing. It looks pretty good overall, but not perfect. We're a little off in some places, especially when the crime numbers are high.

Next, I am going to examine training and CV deviance for different tree sizes.

```
#This shows the different tree sizes (number of terminal nodes or leaves) that were considered during p
prune.tree(uscrime_tree)$size
```

```
## [1] 7 6 5 4 3 2 1
```

```
#This shows the corresponding deviance values for each tree size.
#Deviance is a measure of the tree's prediction error on the training data.
#Lower deviance is generally better:
prune.tree(uscrime_tree)$dev
```

```
## [1] 1895722 2013257 2276670 2632631 3364043 4383406 6880928
```

```
set.seed(42)


#Show the tree sizes considered during cross-validation
cv.tree(uscrime_tree)$size
```

```
## [1] 7 6 5 4 3 2 1
```

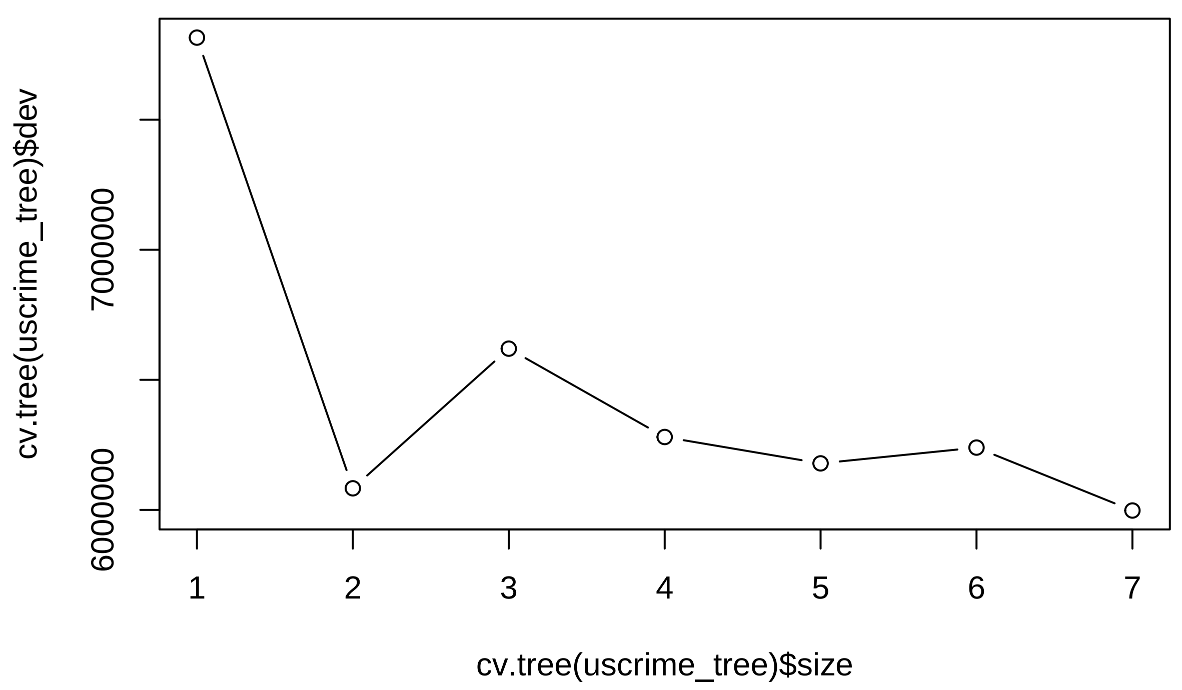
```
#Show the cross-validation deviance values for each tree size.
#These values represent the estimated prediction error on unseen data.
cv.tree(uscrime_tree)$dev
```

```
## [1] 6672664 6667827 7119020 7006961 7327015 7484260 8472709
```

We can see that the prune tree and cross validation both suggested 7 nodes.

Let's plot tosee the relationship between tree size (complexity) and deviance (error) as the tree is pruned by cross vaidation. **This plot helps to choose the optimal size for decision tree by visualizing the trade-off between complexity and error. Pruning a tree can improve its performance on unseen data by reducing overfitting.**:

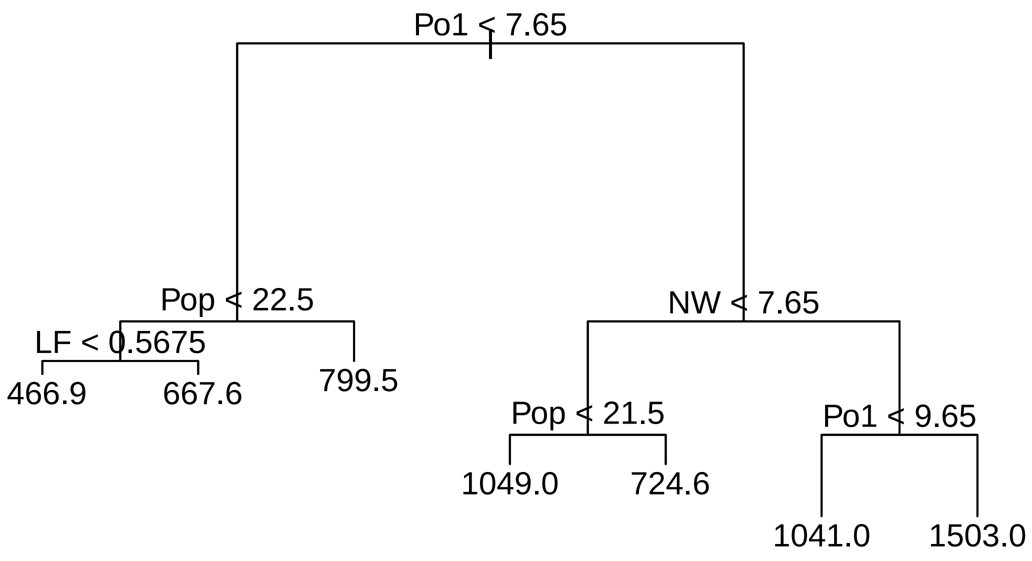
```r
plot(cv.tree(uscrime_tree)$size, cv.tree(uscrime_tree)$dev, type="b")
```



At 7 nodes, the error seems the lowest.

Next, I am going to mannually prune the tree into 7 nodes:

```r
uscrime_tree_pruned <- prune.tree(uscrime_tree, best=7)
plot(uscrime_tree_pruned)
text(uscrime_tree_pruned)
```



```r
summary(uscrime_tree_pruned)
```

```
##
## Regression tree:
## tree(formula = Crime ~ ., data = uscrime)
## Variables actually used in tree construction:
## [1] "Po1" "Pop" "LF"  "NW"
## Number of terminal nodes:  7
## Residual mean deviance:  47390 = 1896000 / 40
## Distribution of residuals:
##     Min.  1st Qu.   Median     Mean  3rd Qu.     Max.
## -573.900  -98.300   -1.545    0.000  110.600  490.100
```

Getting prediction from regression tree:

```r
uscrime_tree_prediction <- predict(uscrime_tree_pruned, data=uscrime$Crime)
uscrime_tree_prediction
```

```
##         1         2         3         4         5         6         7         8
##  799.5455 1502.8750  466.8571 1502.8750 1049.2000  724.6000 1041.0000 1502.8750
##         9        10        11        12        13        14        15        16
##  799.5455  667.6000 1502.8750  799.5455  799.5455  667.6000  799.5455 1041.0000
##        17        18        19        20        21        22        23        24
##  466.8571 1502.8750  724.6000 1502.8750  799.5455  466.8571 1041.0000 1049.2000
##        25        26        27        28        29        30        31        32
##  667.6000 1502.8750  466.8571 1041.0000 1502.8750  799.5455  466.8571 1041.0000
##        33        34        35        36        37        38        39        40
##  799.5455 1049.2000  724.6000 1049.2000  799.5455  667.6000  799.5455 1041.0000
##        41        42        43        44        45        46        47
##  667.6000  466.8571  799.5455  724.6000  466.8571  724.6000 1049.2000
```

Manuually calculating R squared:

```r
#R squared = 1 - (Residual Sum of Squares / Total Sum of Squares)

#RSS represents the sum of the squared differences between the actual values of the dependent variable
#and the values predicted by the model.

#TSS is the sum of the squared differences between the actual values of the dependent variable
#and the mean of the dependent variable.

#Residual Sum of Squares:
RSS <- sum((uscrime_tree_prediction - uscrime$Crime)^2)

#Total Sum of Squares:
TSS <- sum((uscrime$Crime - mean(uscrime$Crime))^2)

#R-squared:
R <- 1-(RSS/TSS)
R
```

```
## [1] 0.7244962
```

R squared shows that the model fits 72.45%. which is not bad

b) random forest model:

Load the randomForest package:

```r
library(randomForest)
```

```
## randomForest 4.7-1.2
```

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

```r
set.seed(123)
```

Grow the random trees in a loop from 1 to 10 to see how many branches are best.

This is chosen by Out-of-Bag error method.

Out-of-Bag (OOB) error is an estimate of the model's prediction error in random forests. It is calculated using data that was not included in the bootstrap sample for training each tree.

How It Works:

- Random forest builds multiple decision trees using bootstrap sampling (random sampling with replacement).
- Each tree is trained on about 63% of the data. The remaining 37% (OOB data) is not used for training that specific tree.
- After training, each tree makes predictions on its own OOB data.
- The final OOB error is the average error from all these predictions.

```r
library(randomForest)

#Create a list to store models:
models <- list()

#Create a vector for errors (out-of-bag errors):
oob_errors <- numeric(10)

#Loop mtry values from 1 to 10:
for (m in 1:10) {
  set.seed(123)
  rf_model <- randomForest(Crime ~ .,
                           data = uscrime,
                           mtry = m,
                           importance = TRUE,
                           ntree = 500)
  #Store the model:
  models[[m]] <- rf_model

  # Store the Out-of-Bag (OOB) error:
  oob_errors[m] <- rf_model$mse[500]
}

#Create a data frame of mtry values and corresponding OOB errors
results <- data.frame(mtry = 1:10, OOB_Error = oob_errors)

#Print results
print(results)
```

```
##    mtry OOB_Error
## 1     1  90283.75
## 2     2  83081.13
## 3     3  81150.31
## 4     4  84238.14
## 5     5  84022.61
## 6     6  85246.58
## 7     7  87683.58
## 8     8  89547.22
## 9     9  90934.01
## 10   10  93904.75
```

```r
#Find the best mtry value
best_mtry <- results$mtry[which.min(results$OOB_Error)]
cat("Best mtry:", best_mtry, "\n")
```

```
## Best mtry: 3
```

With best mtry=3, let's run the model:

```
uscrime_random_forest <- randomForest(Crime~.,
                                      data=uscrime,
                                      mtry=3,
                                      importance=TRUE,
                                      ntree=500)
uscrime_random_forest
```

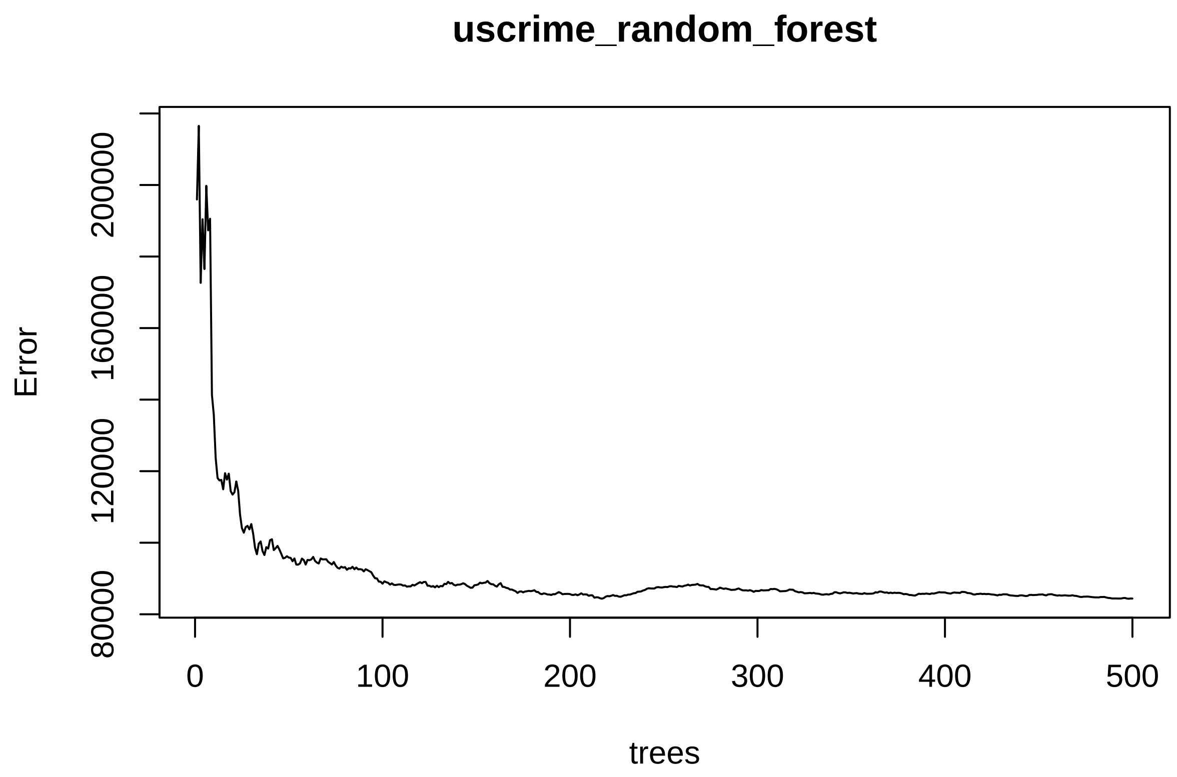
```
##
## Call:
##  randomForest(formula = Crime ~ ., data = uscrime, mtry = 3, importance = TRUE,      ntree = 500)
##                Type of random forest: regression
##                      Number of trees: 500
## No. of variables tried at each split: 3
##
##           Mean of squared residuals: 84389.38
##                     % Var explained: 42.36
```

```
summary(uscrime_random_forest)
```

```
##                 Length Class  Mode
## call              6    -none- call
## type              1    -none- character
## predicted        47    -none- numeric
## mse             500    -none- numeric
## rsq             500    -none- numeric
## oob.times        47    -none- numeric
## importance       30    -none- numeric
## importanceSD     15    -none- numeric
## localImportance   0    -none- NULL
## proximity         0    -none- NULL
## ntree             1    -none- numeric
## mtry              1    -none- numeric
## forest           11    -none- list
## coefs             0    -none- NULL
## y                47    -none- numeric
## test              0    -none- NULL
## inbag             0    -none- NULL
## terms             3    terms  call
```

Plot the model:

```
plot(uscrime_random_forest)
```



It seems like the error rate drops sharply initially and then stabilizes around 100 trees. This suggests that adding more trees beyond that point might not significantly improve the model's performance.

Calculate the R squared for the random forest:

```
rf_pred <- predict(uscrime_random_forest, data=uscrime$Crime)

RSS_rf <- sum((rf_pred-uscrime$Crime)^2)

R_rf <- 1-(RSS_rf/TSS)
R_rf
```

```
## [1] 0.4235805
```

With the best mtry of random forest, R squared is much lower than the previous one (42.36% < 72.45%). This indicates the randomforest is not a good fit for this set of data.