# HW 3

Dylan Doby

9/24/2024

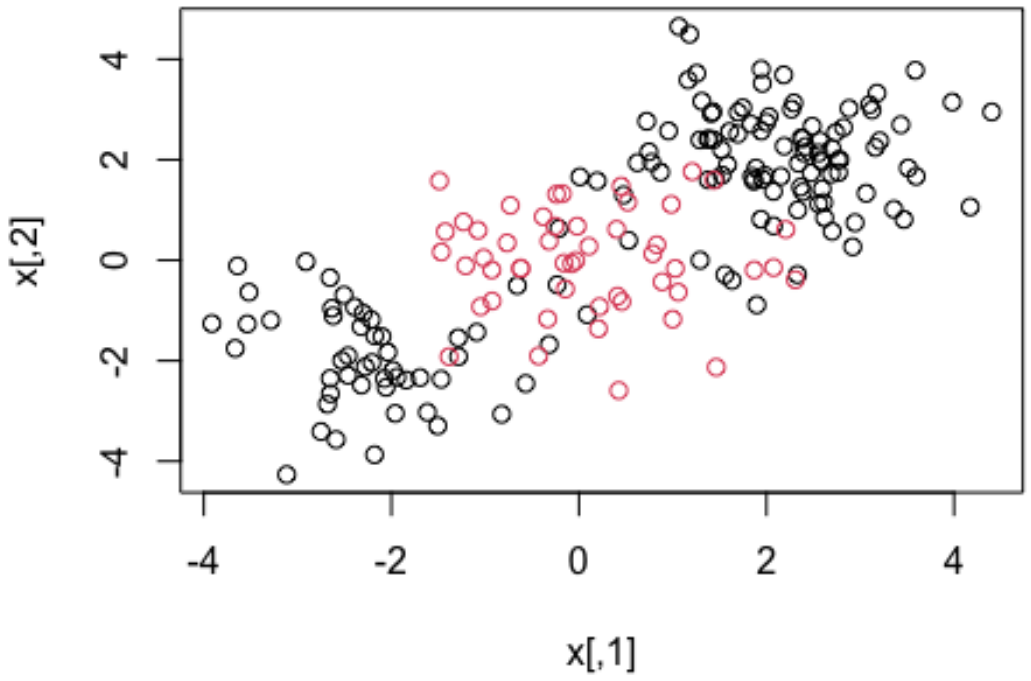
Let $E[X] = \mu$. Show that $Var[X] := E[(X - E[X])^2] = E[X^2] - (E[X])^2$. Note, all you have to do is show the second equality (the first is our definition from class).

$Var(X) = E[(X - E[X])^2]$ <— By class definition

$E[(X - E[X])^2] = E[X^2 - 2X \cdot E[X] + (E[X])^2]$ <— By expanding the squared term
$E[X^2 - 2X \cdot E[X] + (E[X])^2] = E[X^2] - 2E[X] \cdot E[X] + E[(E[X])^2]$ <— By the linearity of expectations $E[X^2] - 2E[X] \cdot E[X] + E[(E[X])^2] = E[X^2] - 2E[X] \cdot E[X] + (E[X])^2$ <— By $E[X]$ being a constant $E[X^2] - 2E[X] \cdot E[X] + (E[X])^2 = E[X^2] - (E[X])^2$ <— By simplification

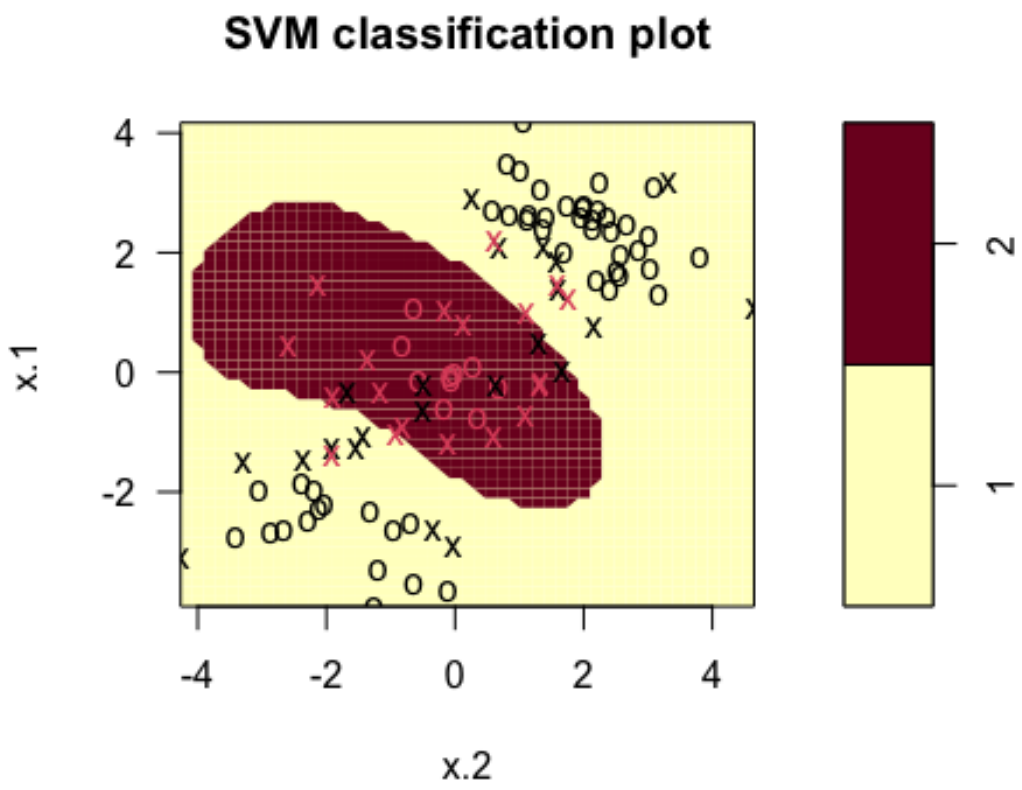
In the computational section of this homework, we will discuss support vector machines and tree-based methods. I will begin by simulating some data for you to use with SVM.

```
library(e1071)
set.seed(1)
x=matrix(rnorm(200*2),ncol=2)
x[1:100,]=x[1:100,]+2
x[101:150,]=x[101:150,]-2
y=c(rep(1,150),rep(2,50))
dat=data.frame(x=x,y=as.factor(y))
plot(x, col=y)
```




Quite clearly, the above data is not linearly separable. Create a training-testing partition with 100 random observations in the training partition. Fit an svm on this training data using the radial kernel, and tuning parameters $\gamma = 1$, cost $= 1$. Plot the svm on the training data.

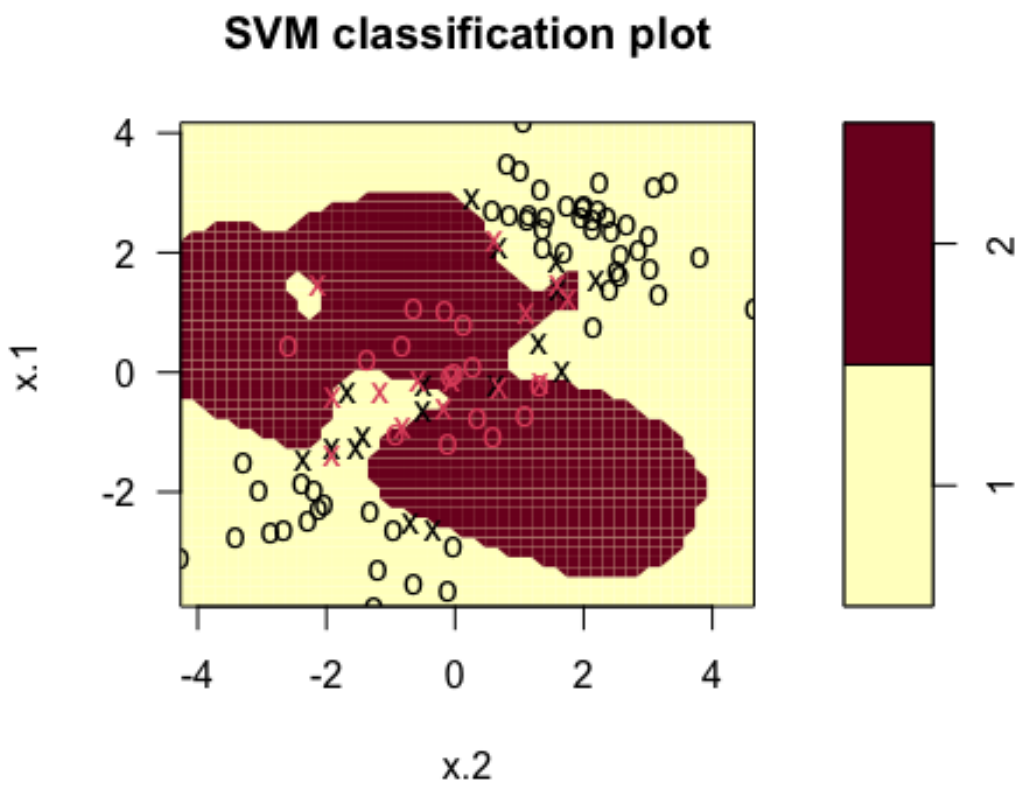
```
set.seed(1)
# Split the data into training and testing sets
train_indices = sample(1:200, 100)  # Randomly select 100 observations for
the training set
train_data = dat[train_indices, ]    # Training data
test_data = dat[-train_indices, ]    # Testing data

# Fit the SVM model on the training data using radial kernel with gamma = 1
and cost = 1
svmfit = svm(y ~ ., data = train_data, kernel = "radial", gamma = 1, cost =
1)

# Plot the SVM results on the training data
plot(svmfit, train_data)
```




Notice that the above decision boundary is decidedly non-linear. It seems to perform reasonably well, but there are indeed some misclassifications. Let's see if increasing the cost [1] helps our classification error rate. Refit the svm with the radial kernel, $\gamma = 1$, and a cost of 10000. Plot this svm on the training data.

```
# Refit the SVM model with a higher cost (10000)
svmfit = svm(y ~ ., data = train_data, kernel = "radial", gamma = 1, cost = 10000)

# Plot the SVM with the higher cost on the training data
plot(svmfit, train_data)
```

[1] Remember this is a parameter that decides how smooth your decision boundary should be




It would appear that we are better capturing the training data, but comment on the dangers (if any exist), of such a model.

*Student Response*

While increasing the cost parameter appears to help in better capturing the training data by reducing the number of misclassifications, it introduces the risk of overfitting, which occurs when a model becomes too closely tailored to the specifics of the training data, capturing noise and outliers rather than general patterns. This can lead to a very complex decision boundary that fits the training data perfectly but may perform poorly on unseen test data.

In this case, by using a very high cost, we are prioritizing correctly classifying the training data at the expense of creating a smoother and more generalizable decision boundary. This could result in the model having high variance, meaning it will struggle to generalize to new data, leading to poor performance on the test set or in real-world applications.

The major danger is that the model might perform worse when predicting on new data, as it fails to capture the broader structure of the data and instead overfits to the idiosyncrasies of the training set. Therefore, while increasing the cost can reduce training error, it may increase the overall test error due to overfitting.

Create a confusion matrix by using this svm to predict on the current testing partition. Comment on the confusion matrix. Is there any disparity in our classification results?

```
#remove eval = FALSE in above
table(true=dat[-train_indices,"y"], pred=predict(svmfit, newdata=dat[-
train_indices,]))
```

```
##     pred
## true  1  2
##    1 67 12
##    2  2 19
```

```
#accuracy calculation
(27+11)/(27+11+1+2)
```

```
## [1] 0.9268293
```

The model correctly classified 67 instances as class 1, but misclassified 12 instances as class 2. Additionally, the model correctly classified 19 instances as class 2, but misclassified 2 instances as class 1. There appears to be a disparity in the misclassification rates between the two classes with class 1 having more misclassifications (12 instances) than class 2 (2 instances). However, there is such a small number of overall misclassifications, instance-wise and percentage-wise, that it is unlikely to have a substantial impact on the model's overall performance. The high accuracy rate of 92.68% demonstrates that the model is generally performing well in distinguishing between the two classes.

Is this disparity because of imbalance in the training/testing partition? Find the proportion of class 2 in your training partition and see if it is broadly representative of the underlying 25% of class 2 in the data as a whole.

```
# Calculate the proportion of class 2 in the entire dataset
total_proportion_class_2 = sum(dat$y == 2) / nrow(dat)

# Calculate the proportion of class 2 in the training partition
train_proportion_class_2 = sum(train_data$y == 2) / nrow(train_data)

# Print the results
cat("Proportion of class 2 in the entire dataset:", total_proportion_class_2,
"\n")
```

```
## Proportion of class 2 in the entire dataset: 0.25
```

```
cat("Proportion of class 2 in the training partition:",
train_proportion_class_2, "\n")
```

```
## Proportion of class 2 in the training partition: 0.29
```

*Student Response*

The proportion of class 2 in the entire dataset is 25%, which matches the expected proportion, as 50 out of 200 observations belong to class 2. However, in the training partition, the proportion of class 2 is slightly higher at 29%. While this difference is not extreme, it does indicate a mild imbalance in the training partition, with a slightly higher representation of class 2 than in the overall dataset. This could contribute to the model being more attuned to class 2, leading to fewer misclassifications for class 2 compared to class 1. Although this small imbalance might not fully explain the disparity in misclassifications, it could be a contributing factor.

Let's try and balance the above to solutions via cross-validation. Using the `tune` function, pass in the training data, and a list of the following cost and $\gamma$ values: {0.1, 1, 10, 100, 1000} and {0.5, 1,2,3,4}. Save the output of this function in a variable called `tune.out`.

```
set.seed(1)

# Perform cross-validation using the tune function with the specified cost
and gamma values
tune.out <- tune(
  svm,
  y ~ .,
  data = train_data,
  kernel = "radial",
  ranges = list(cost = c(0.1, 1, 10, 100, 1000), gamma = c(0.5, 1, 2, 3, 4))
)

# Print the results of the cross-validation
summary(tune.out)
```

```
##
## Parameter tuning of 'svm':
##
## - sampling method: 10-fold cross validation
##
## - best parameters:
##  cost gamma
##     1   0.5
##
## - best performance: 0.12
##
## - Detailed performance results:
##     cost gamma error dispersion
## 1  1e-01   0.5  0.28 0.15491933
## 2  1e+00   0.5  0.12 0.07888106
## 3  1e+01   0.5  0.15 0.10801234
## 4  1e+02   0.5  0.17 0.11595018
## 5  1e+03   0.5  0.23 0.14944341
```

```
## 6  1e-01   1.0  0.25 0.13540064
## 7  1e+00   1.0  0.14 0.09660918
## 8  1e+01   1.0  0.16 0.10749677
## 9  1e+02   1.0  0.21 0.15238839
## 10 1e+03   1.0  0.20 0.14142136
## 11 1e-01   2.0  0.28 0.14757296
## 12 1e+00   2.0  0.15 0.10801234
## 13 1e+01   2.0  0.19 0.15238839
## 14 1e+02   2.0  0.18 0.14757296
## 15 1e+03   2.0  0.23 0.12516656
## 16 1e-01   3.0  0.28 0.15491933
## 17 1e+00   3.0  0.15 0.10801234
## 18 1e+01   3.0  0.20 0.16329932
## 19 1e+02   3.0  0.20 0.13333333
## 20 1e+03   3.0  0.27 0.11595018
## 21 1e-01   4.0  0.29 0.14491377
## 22 1e+00   4.0  0.16 0.09660918
## 23 1e+01   4.0  0.18 0.13984118
## 24 1e+02   4.0  0.21 0.11972190
## 25 1e+03   4.0  0.31 0.15951315
```

I will take `tune.out` and use the best model according to error rate to test on our data. I will report a confusion matrix corresponding to the 100 predictions.

```
table(true=dat[-train_indices,"y"], pred=predict(tune.out$best.model,
newdata=dat[-train_indices,]))
```

```
##     pred
## true  1  2
##    1 72  7
##    2  1 20
```

```
#accuracy calculation
(27+12)/(27+12+0+2)
```

```
## [1] 0.9512195
```

Comment on the confusion matrix. How have we improved upon the model in question 2 and what qualifications are still necessary for this improved model.

*Student Response*

The model correctly classified 72 instances as class 1 and misclassified 7 instances as class 2. The model also correctly classified 20 instances as class 2 and misclassified 1 instance as class 1. The accuracy has also increased from 92.68% to 95.12%, indicating better overall classification performance.

Although there are improvements in the accuracy and classification of class 2, the model still misclassified 7 instances of class 1,suggesting that there may still be some boundary

overlap or noise that the model is struggling to manage for class 1. Additionally, the model's improved accuracy could indicate that it is more closely fitting the training data, but this may come at the cost of overfitting. This could lead to decreased performance when applied to new, unseen data. To address this, it would be important to assess the model's performance on a separate validation or test set to ensure it generalizes well beyond the training partition.

Also, even though the model performed perfectly for class 2, the training partition still contained a slightly higher proportion of class 2 than the overall dataset (29% in the training set compared to 25% in the full dataset). This imbalance could have influenced the model to focus more on class 2, potentially leading to the misclassification of some class 1 instances. Stratified sampling or class weighting could be employed to confirm that the model is learning in a more balanced way.

Let's turn now to decision trees.

```
library(kmed)
data(heart)
library(tree)
```

The response variable is currently a categorical variable with four levels. Convert heart disease into binary categorical variable. Then, ensure that it is properly stored as a factor.

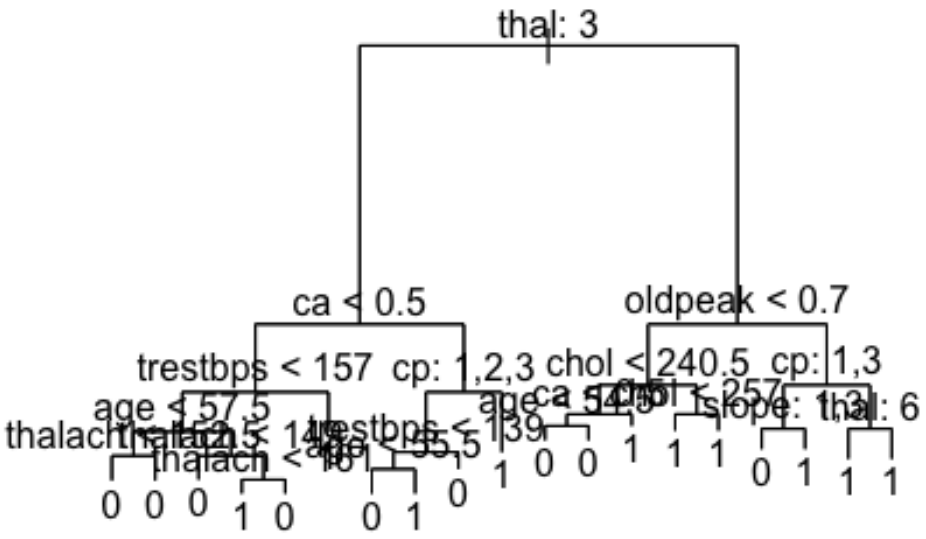
```
# Convert the 'class' variable into a binary categorical variable
heart$heart_disease <- ifelse(heart$class == 0, 0, 1)

# Ensure the binary variable is stored as a factor
heart$heart_disease <- as.factor(heart$heart_disease)

# Check the structure to confirm the changes
str(heart)
```

```
## 'data.frame':    297 obs. of  15 variables:
##  $ age          : num  63 67 67 37 41 56 62 57 63 53 ...
##  $ sex          : logi  TRUE TRUE TRUE TRUE FALSE TRUE ...
##  $ cp           : Factor w/ 4 levels "1","2","3","4": 1 4 4 3 2 2 4 4 4 4
...
##  $ trestbps     : num  145 160 120 130 130 120 140 120 130 140 ...
##  $ chol         : num  233 286 229 250 204 236 268 354 254 203 ...
##  $ fbs          : logi  TRUE FALSE FALSE FALSE FALSE FALSE ...
##  $ restecg      : Factor w/ 3 levels "0","1","2": 3 3 3 1 3 1 3 1 3 3 ...
##  $ thalach      : num  150 108 129 187 172 178 160 163 147 155 ...
##  $ exang        : logi  FALSE TRUE TRUE FALSE FALSE FALSE ...
##  $ oldpeak      : num  2.3 1.5 2.6 3.5 1.4 0.8 3.6 0.6 1.4 3.1 ...
```

```
##  $ slope        : Factor w/ 3 levels "1","2","3": 3 2 2 3 1 1 3 1 2 3 ...
##  $ ca           : num  0 3 2 0 0 0 2 0 1 0 ...
##  $ thal         : Factor w/ 3 levels "3","6","7": 2 1 3 1 1 1 1 1 3 3 ...
##  $ class        : int  0 2 1 0 0 0 3 0 2 1 ...
##  $ heart_disease: Factor w/ 2 levels "0","1": 1 2 2 1 1 1 2 1 2 2 ...
##  - attr(*, "na.action")= 'omit' Named int [1:6] 88 167 193 267 288 303
##   ..- attr(*, "names")= chr [1:6] "88" "167" "193" "267" ...
```

Train a classification tree on a 240 observation training subset (using the seed I have set for you). Plot the tree.

```
set.seed(101)

# Split the dataset into training and testing sets
train_indices <- sample(1:nrow(heart), 240)  # randomly select 240 rows for
training
train_data <- heart[train_indices, ]
test_data <- heart[-train_indices, ]

# Train the classification tree on the training data
tree_model <- tree(heart_disease~.-class, data = train_data)

# Plot the classification tree
plot(tree_model)
text(tree_model, pretty = 0)
```




Use the trained model to classify the remaining testing points. Create a confusion matrix to evaluate performance. Report the classification error rate.

```
# Use the trained model to predict on the testing data
tree_predictions <- predict(tree_model, test_data, type = "class")
with(test_data, table(predicted = tree_predictions, actual = heart_disease))
```

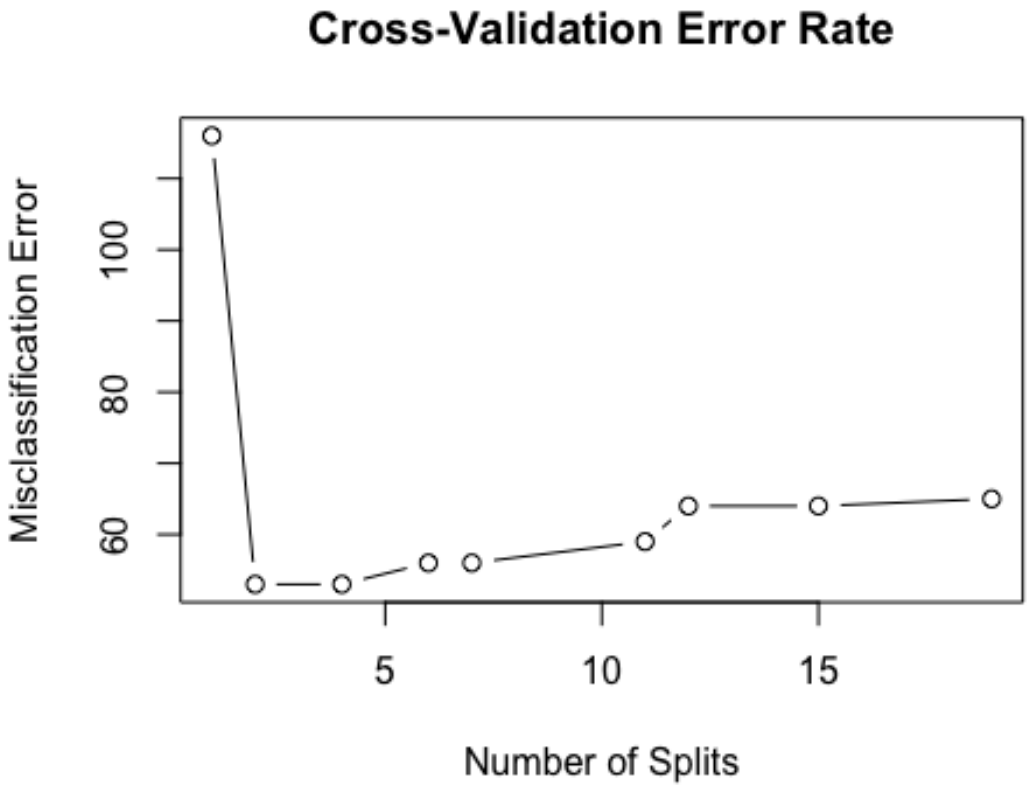
```
##          actual
## predicted  0  1
##         0 28  3
##         1  8 18
```

```
# Calculate the classification accuracy
accuracy <- sum(tree_predictions == test_data$heart_disease) /
nrow(test_data)

# Calculate the classification error rate
error_rate <- 1 - accuracy

# Print the classification error rate
cat("Classification Error Rate:", error_rate, "\n")
```

```
## Classification Error Rate: 0.1929825
```

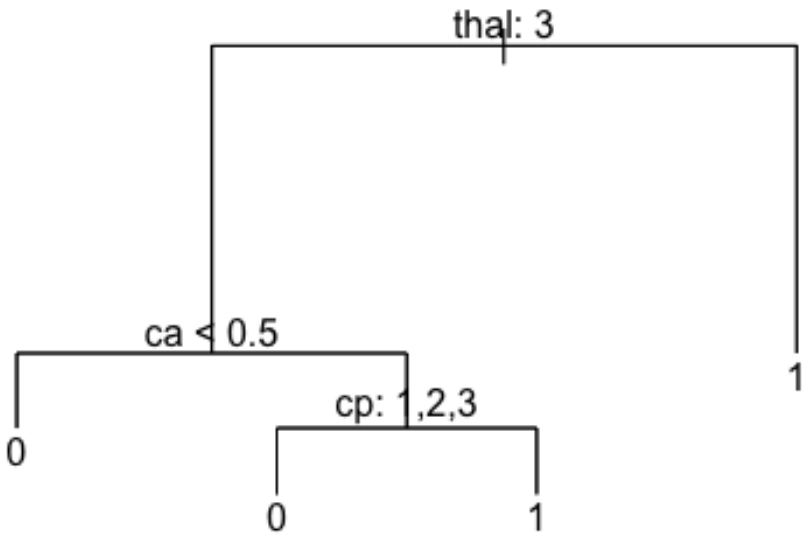
Above we have a fully grown (bushy) tree. Now, cross validate it using the `cv.tree` command. Specify cross validation to be done according to the misclassification rate. Choose an ideal number of splits, and plot this tree. Finally, use this pruned tree to test on the testing set. Report a confusion matrix and the misclassification rate.

```
set.seed(101)

# Perform cross-validation using the misclassification rate as the criterion
cv_tree <- cv.tree(tree_model, FUN = prune.misclass)

# Plot the cross-validation results to identify the ideal number of splits
plot(cv_tree$size, cv_tree$dev, type = "b", xlab = "Number of Splits", ylab =
"Misclassification Error", main = "Cross-Validation Error Rate")
```




```
# Identify the optimal number of splits (where deviance is minimized)
optimal_size <- cv_tree$size[which.min(cv_tree$dev)]

# Prune the tree to the optimal size
pruned_tree <- prune.misclass(tree_model, best = optimal_size)
```

```r
# Plot the pruned tree
plot(pruned_tree)
text(pruned_tree, pretty = 0)
```




```r
# Use the pruned tree to predict on the testing data
pruned_tree_predictions <- predict(pruned_tree, test_data, type = "class")

# Create a confusion matrix for the pruned tree
pruned_confusion_matrix <- table(True = test_data$heart_disease, Predicted =
pruned_tree_predictions)

# Print the confusion matrix
print(pruned_confusion_matrix)
```

```
##     Predicted
## True  0  1
##    0 26 10
##    1  4 17
```

```r
# Calculate the misclassification rate for the pruned tree
pruned_error_rate <- 1 - sum(diag(pruned_confusion_matrix)) /
sum(pruned_confusion_matrix)
```

```r
# Print the misclassification rate
cat("Pruned Tree Misclassification Rate:", pruned_error_rate, "\n")
```

```
## Pruned Tree Misclassification Rate: 0.245614
```

Discuss the trade-off in accuracy and interpretability in pruning the above tree.

*Student Input*

A fully grown (unpruned) tree captures every possible relationship in the data, including noise. This can lead to overfitting, where the model performs extremely well on the training data but struggles to generalize to unseen data (test set). The result is usually a higher complexity but higher accuracy on the training set.

By pruning the tree, the model is simplified by cutting branches that contribute little to the predictive power. While this reduces the model's complexity and makes it less prone to overfitting, it might also slightly reduce the model's accuracy. As seen in my case, the pruned tree's misclassification rate is 0.2456 vs. 0.1929825 with the unpruned tree, which indicates a compromise between fit and generalization. However, pruning generally helps avoid overfitting and improves the model's performance on the test set.

Discuss the ways a decision tree could manifest algorithmic bias.

*Student Answer*

A decision tree can introduce algorithmic bias in several ways, largely due to the data it learns from and the processes used during its construction. One major source of bias comes from biased training data. If the data used to train the decision tree contains underlying biases (such as racial, gender, or socio-economic disparities), the model will likely learn and replicate these biases in its predictions.

Another way bias can occur is through feature selection bias. Decision trees tend to prioritize features that create the largest splits first. If these features are correlated with sensitive attributes (like race or gender), the tree might implicitly base its decisions on those attributes, even if they are not explicitly included in the model. This can lead to discriminatory outcomes if, for instance, certain demographic groups are unfairly predicted to have worse outcomes based on historical data patterns.

Additionally, sampling bias can affect the fairness of a decision tree. If the training set does not adequately represent the full diversity of the population or is skewed toward certain groups, the tree will not perform equally well across all subgroups. As a result, certain segments of the population might receive less accurate or less favorable predictions, which could reinforce existing inequalities.

Overfitting to a biased dataset can also exacerbate these issues. If the tree becomes overly complex and memorizes specific biases in the training data, it will reinforce those biases in its predictions, leading to poor generalization to new, unbiased data.