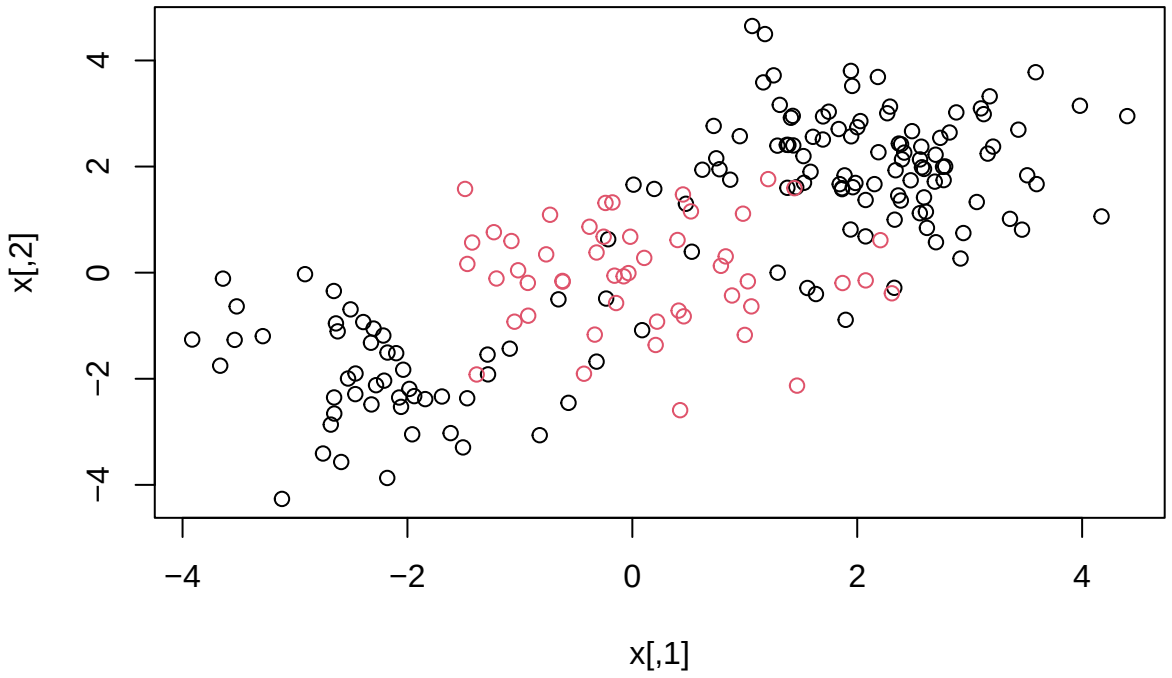
STOR 390: HOMEWORK 3

Riley Harper

February 22, 2024

In 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)
```

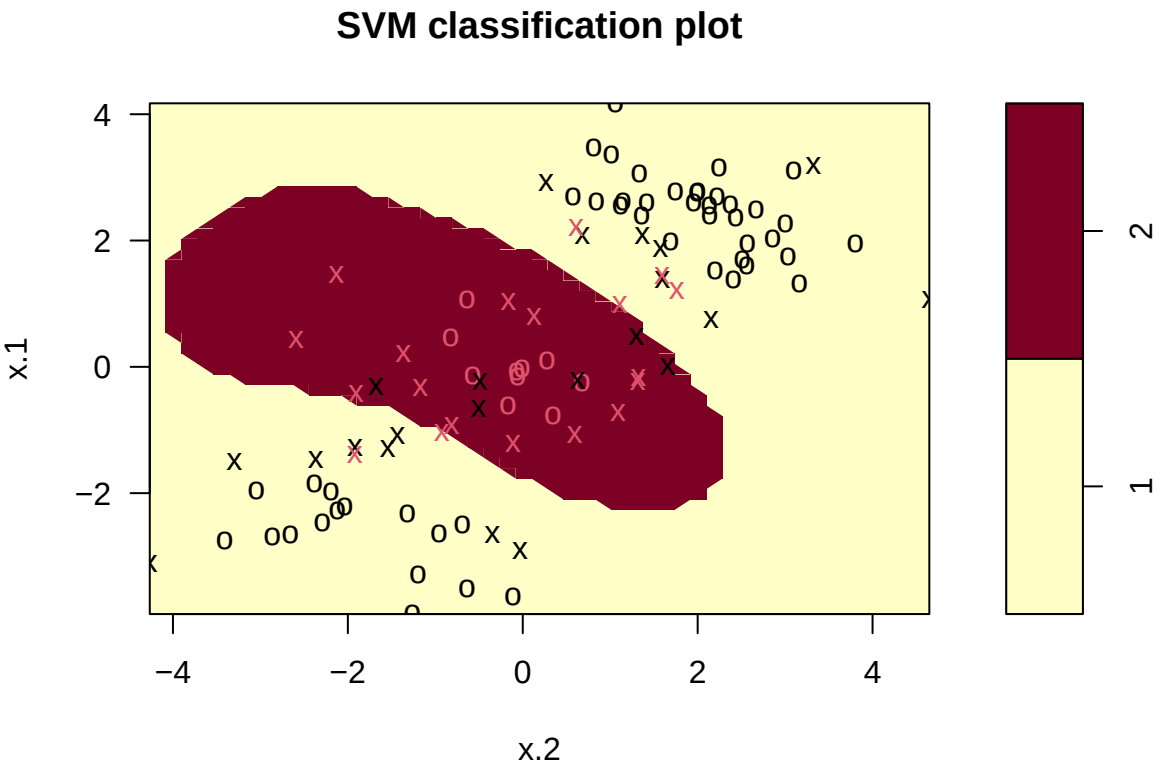


0.1

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.

```r
set.seed(1)
trainIndices = sample(1:nrow(dat), 100)
trainData = dat[trainIndices, ]
testData = dat[-trainIndices, ]

svmmod = svm(y ~ x.1 + x.2, data = trainData, kernel = "radial", gamma = 1, cost = 1)
plot(svmmod, trainData)
```

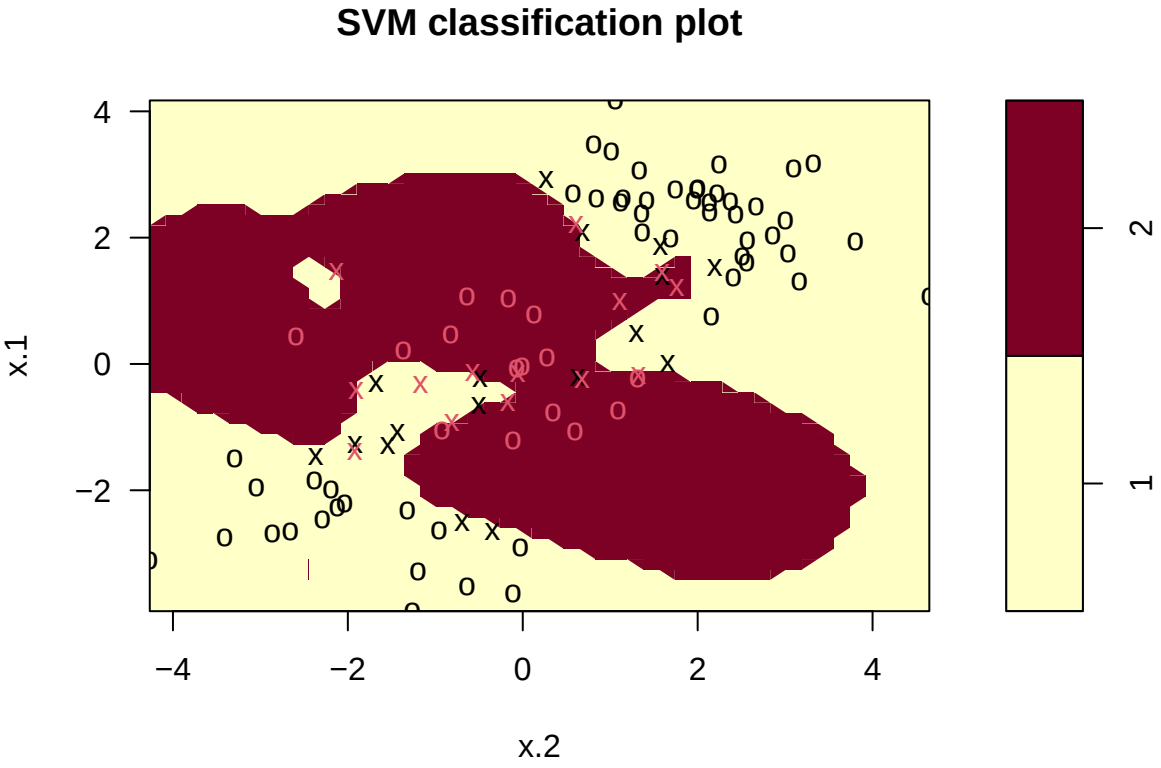


0.2

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.

```r
svmmod1 = svm(y ~ x.1 + x.2, data = trainData, kernel = "radial", gamma = 1, cost = 10000)
plot(svmmod1, trainData)
```

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



0.3

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

SVM does a great job of capturing the extremely complex decision boundary present in the features, however, the dangers of utilizing an extremely high cost come from overfitting on the training data. Overfitting is a common problem in data science and results from creating a model which is overly complex to the point of being Lagrangian and fitting all of the training data. These models tend to possess a characteristic of poorly performing on unseen test data.

0.4

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?

```
table(true=dat[-trainIndices,"y"], pred=predict(svmmod1, newdata=dat[-trainIndices,]))
```

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

The confusion matrix does reveal a disparity in the classification results, particularly in the model's ability to correctly classify instances of class 2. The number of false positives is significantly higher than the number of false negatives, indicating that the model is more likely to misclassify class 2 instances as class 1. This could be a sign of the model's bias towards class 1, possibly due to overfitting on the training data where complex

decision boundaries might have been emphasized. The relatively high number of false positives compared to false negatives suggests that while the model is quite sensitive-high true positive rate, it may not be very specific-lower true negative rate.

Addressing overfitting requires careful tuning of the model's parameters, such as the cost and γ, and the introduction of regularization techniques. Additionally, using cross-validation, as mentioned in my previous homework, to select the parameters can help ensure that the model generalizes well to new data. Balancing the dataset or adjusting the decision threshold based on the costs of different types of misclassifications could also help mitigate the disparity in classification accuracy between the classes.

0.5

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.

```
table(true=dat[-trainIndices,"y"], pred=predict(svmmod1, newdata=dat[-trainIndices,]))
```

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

```
classProportions = table(dat[trainIndices, "y"]) / length(trainIndices)
proportionOfClass2 = classProportions["2"]
print(paste0("The proportion of class 2 is ", proportionOfClass2 * 100, "%."))
```

```
## [1] "The proportion of class 2 is 29%."
```

A proportion of 29% for class 2 indicates a slight imbalance but is relatively close to the underlying 25% of class 2 in the data as a whole.

0.6

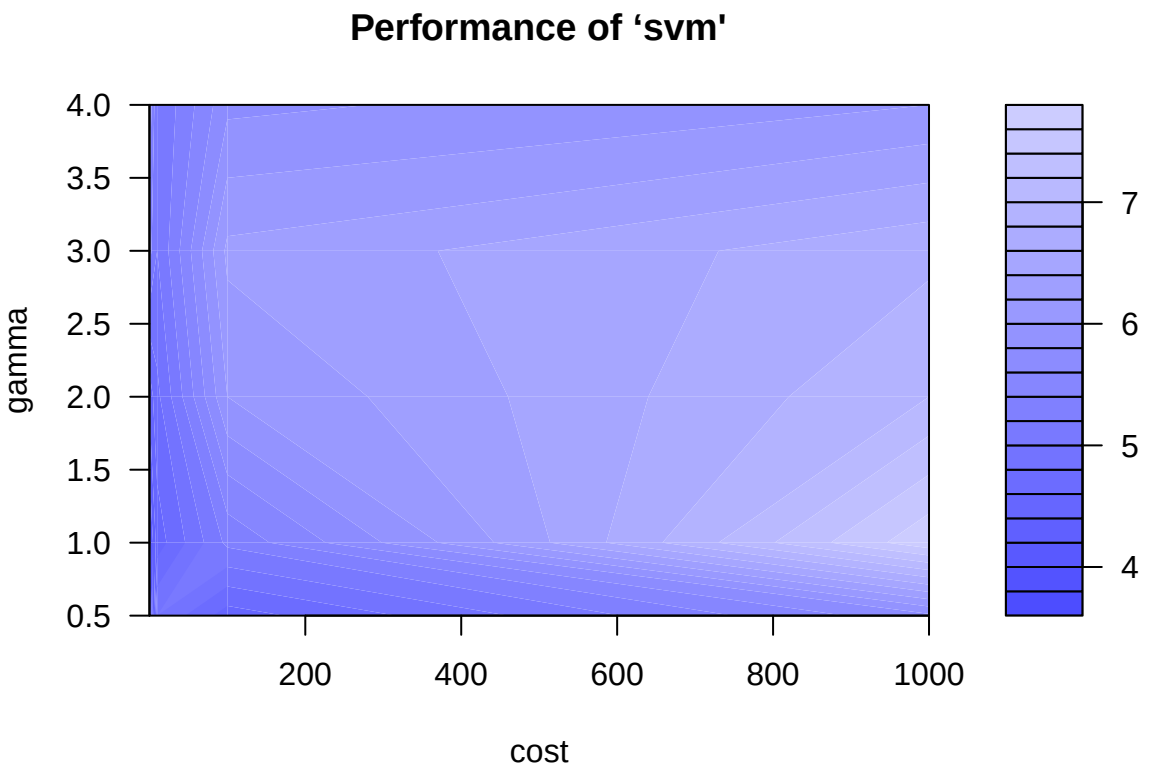
Let's try and balance the above to solutions via cross-validation. Using the `tune` function, pass in the training data, and a list of the following cost and γ values: {0.1, 1, 10, 100, 1000} and {0.5, 1,2,3,4}. Save the output of this function in a variable called `tune.out`.

```
set.seed(1)
tuneGrid <- expand.grid(cost = c(0.1, 1, 10, 100, 1000), gamma = c(0.5, 1, 2, 3, 4))
tune.out <- tune(svm, y ~ x.1 + x.2, data = trainData,
                 kernel = "radial",
                 ranges = tuneGrid,
                 scale = FALSE)
print(tune.out)
```

```
##
## Parameter tuning of 'svm':
##
## - sampling method: 10-fold cross validation
```

```
##
## - best parameters:
##  cost gamma
##     1   0.5
##
## - best performance: 0.15
```

```
plot(tune.out)
```



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

0.7

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.

True Positives (TP): 73 (correctly predicted as class 1) False Negatives (FN): 6 (class 1 incorrectly predicted as class 2) False Positives (FP): 3 (class 2 incorrectly predicted as class 1) True Negatives (TN): 18 (correctly predicted as class 2) The confusion matrix indicates a higher accuracy, with the model correctly classifying the majority of the data. However, there are still a few misclassifications, as seen in the presence of both false positives and false negatives, though they are relatively low in number. This outcome demonstrates that the tuned model, with the best combination of cost and γ parameters identified through cross-validation, performs well on the test set. This is because the model effectively balances the trade-off between sensitivity (true positive rate) and specificity (true negative rate).

1

Let's turn now to decision trees.

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

1.1

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.

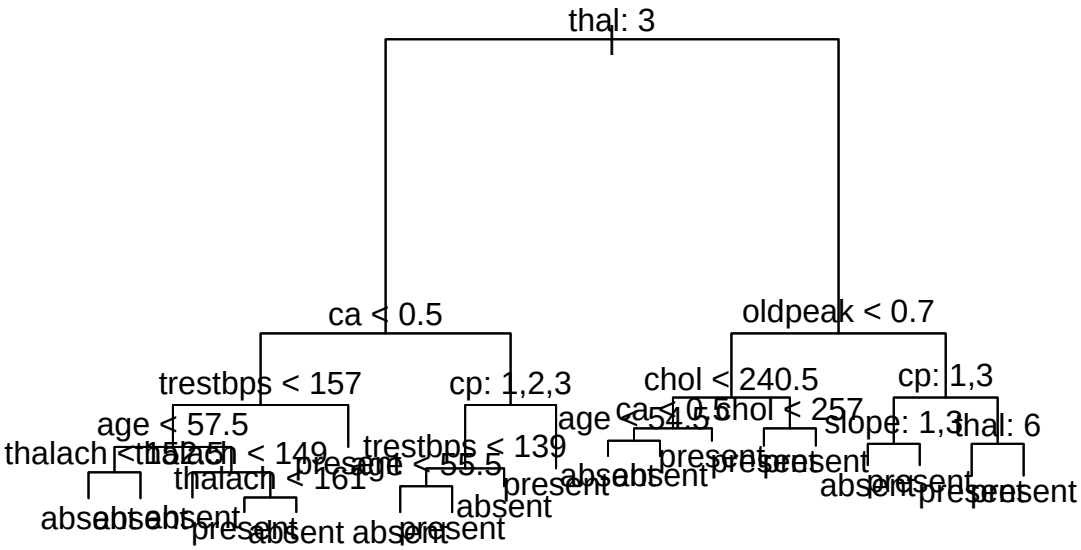
```
heart$class_binary <- ifelse(heart$class > 0, 1, 0)
heart$class_binary <- as.factor(heart$class_binary)
levels(heart$class_binary) <- c("absent", "present")
str(heart$class_binary)
```

```
##  Factor w/ 2 levels "absent","present": 1 2 2 1 1 1 2 1 2 2 ...
```

1.2

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

```
set.seed(101)
trainIndices <- sample(1:nrow(heart), 240)
trainData <- heart[trainIndices,]
treeMod <- tree(class_binary ~ . - class, data = trainData)
plot(treeMod)
text(treeMod, pretty = 0)
```



1.3

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

```
testData <- heart[-trainIndices, ]
predictions <- predict(treeMod, newdata = testData, type = "class")
confMatrix <- table(Predicted = predictions, Actual = testData$class_binary)
print(confMatrix)
```

```
##          Actual
## Predicted absent present
##   absent      28       3
##   present      8      18
```

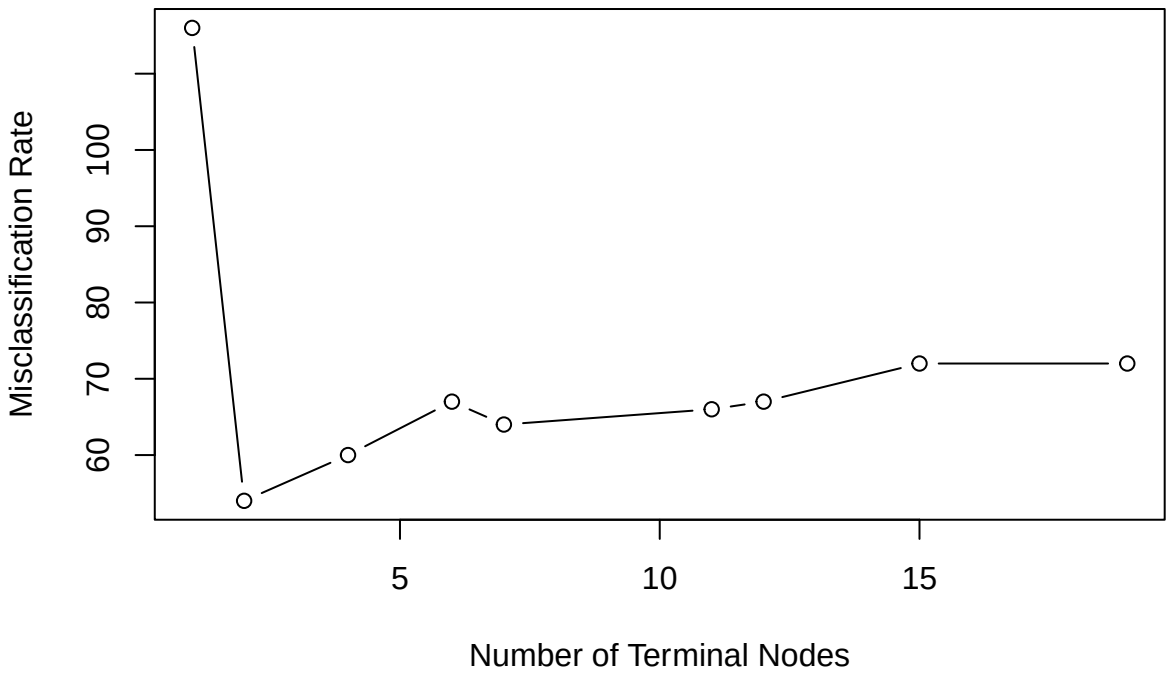
```
errorRate <- 1 - sum(diag(confMatrix)) / nrow(testData)
print(paste0("The classification error rate is ", round(errorRate, 4)* 100, "%"))
```

```
## [1] "The classification error rate is 19.3%"
```

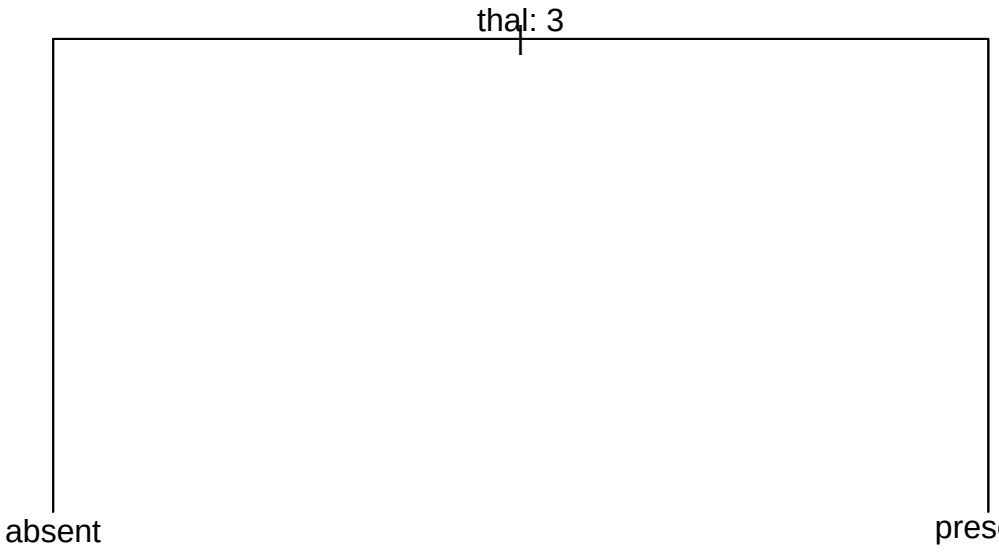
1.4

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.

```
cvTree <- cv.tree(treeMod, FUN=prune.misclass)
plot(cvTree$size, cvTree$dev, type='b', xlab="Number of Terminal Nodes", ylab="Misclassification Rate")
```



```
optimalSize <- cvTree$size[which.min(cvTree$dev)]
prunedTree <- prune.misclass(treeMod, best=optimalSize)
plot(prunedTree)
text(prunedTree, pretty=0)
```




```
prunedPredictions <- predict(prunedTree, newdata = testData, type = "class")
confMatrixPruned <- table(Predicted = prunedPredictions, Actual = testData$class_binary)
print(confMatrixPruned)
```

```
##           Actual
## Predicted absent present
##   absent      28       9
##   present      8      12
```

```
errorRatePruned <- 1 - sum(diag(confMatrixPruned)) / nrow(testData)
print(paste0("The classification error rate of the pruned tree is ", round(errorRatePruned, 4) * 100, "
```

```
## [1] "The classification error rate of the pruned tree is 29.82%"
```

1.5

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

Initially, the fully grown decision tree achieved a classification error rate of 19.3%, reflecting a reasonably high level of accuracy in distinguishing between the presence and absence of heart disease. The confusion matrix associated with this model showed a balanced ability to predict both classes, with relatively few misclassifications. However, the complexity of a fully grown tree, with its numerous branches and leaves, can make it challenging for users to interpret the model's decision-making process. The detailed splits may capture noise in the data rather than underlying patterns, making the model's predictions less reliable when applied to new unseen data.

In contrast, the pruned tree, optimized through cross-validation to minimize the misclassification rate, demonstrated a slightly higher error rate of 29.82%. This increase in the error rate following pruning indicates a reduction in accuracy. The confusion matrix revealed that while the pruned tree continued to

correctly identify the absence of disease, its ability to correctly predict the presence of heart disease diminished slightly. This result suggests that pruning, by simplifying the model, might have removed some of the nuance and detail necessary for capturing more complex patterns associated with the disease's presence in the data.

Despite this reduction in accuracy, the pruned tree offers a significant advantage in terms of interpretability. By reducing the tree to a smaller number of terminal nodes (2 nodes), pruning makes the model's logic more straightforward and easier for users to understand. This simplification allows non-data scientists to grasp the key factors influencing the model's predictions, allowing for easier validation of the model's decision rules in clinical settings.

The trade-off between accuracy and interpretability in pruning decision trees is a fundamental challenge in machine learning. The goal often being to optimize model performance while ensuring that the models remain understandable to those who use them. In this case, the slight decrease in accuracy with pruning should be considered against the backdrop of a more interpretable model.The decision to prune a tree and accept a higher error rate may be justified if the resulting model aligns better with the needs for intrepretability and application in real-world scenarios.

Overall, the choice between a more accurate but complex model and a more simple but interpretable model depends on the specific context and objectives of the analysis. In domains where understanding the model's rationale is as important as its predictive power, such as in healthcare, the benefits of pruning for interpretability may outweigh the drawbacks of a modest decrease in accuracy.

1.6

Discuss the ways a decision tree could manifest algorithmic bias.

If the training data contains historical biases, stereotypes, or imbalances, the decision tree will likely learn and perpetuate these biases in its predictions. For example, a dataset with an underrepresentation of certain demographic groups can lead the tree to make less accurate predictions for those groups, as it has not been adequately trained on data reflective of their characteristics or outcomes. Similarly, if the data reflects historical prejudices or discriminatory practices, the decision tree may inadvertently learn these patterns, leading to biased decisions that favor one group over another.

Another way bias manifests in decision trees is through the selection of features used to split the data. If the features include, or are closely correlated with, sensitive attributes such as race, gender, or age, the tree may make splits that result in discriminatory decision paths. Even when sensitive attributes are not directly used, proxy variables that are associated with these attributes can introduce bias. The model's reliance on such features can exacerbate disparities, making it essential to critically assess and validate the choice of features in model development. Evidence of this was demonstrated to me in the work done on my prior STOR 455 class which involved demographic data collected by TELAMON for the Social Innovation and Entrepreneurship Lab at UNC. It was necessary for me to remove categories which possessed sensitive attributes to avoid modeling on data which is not relevant to the research question I was hoping to answer and to avoid biased results.