STOR390HW3

Jillian Myler

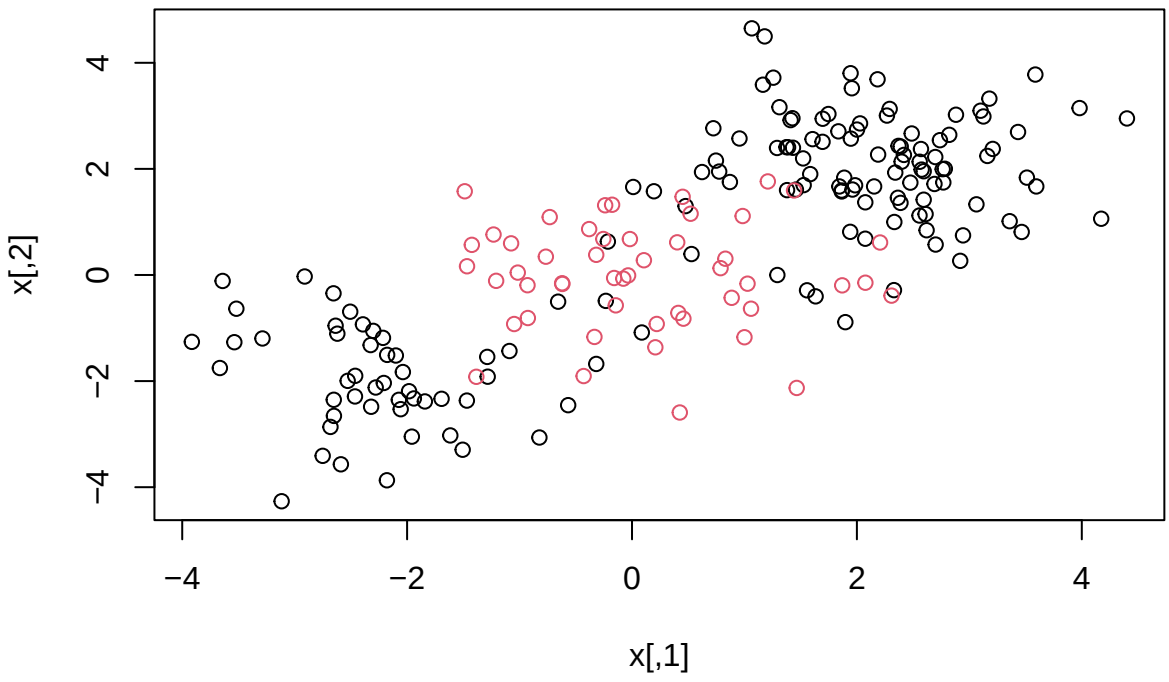
2024-02-24

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)
```

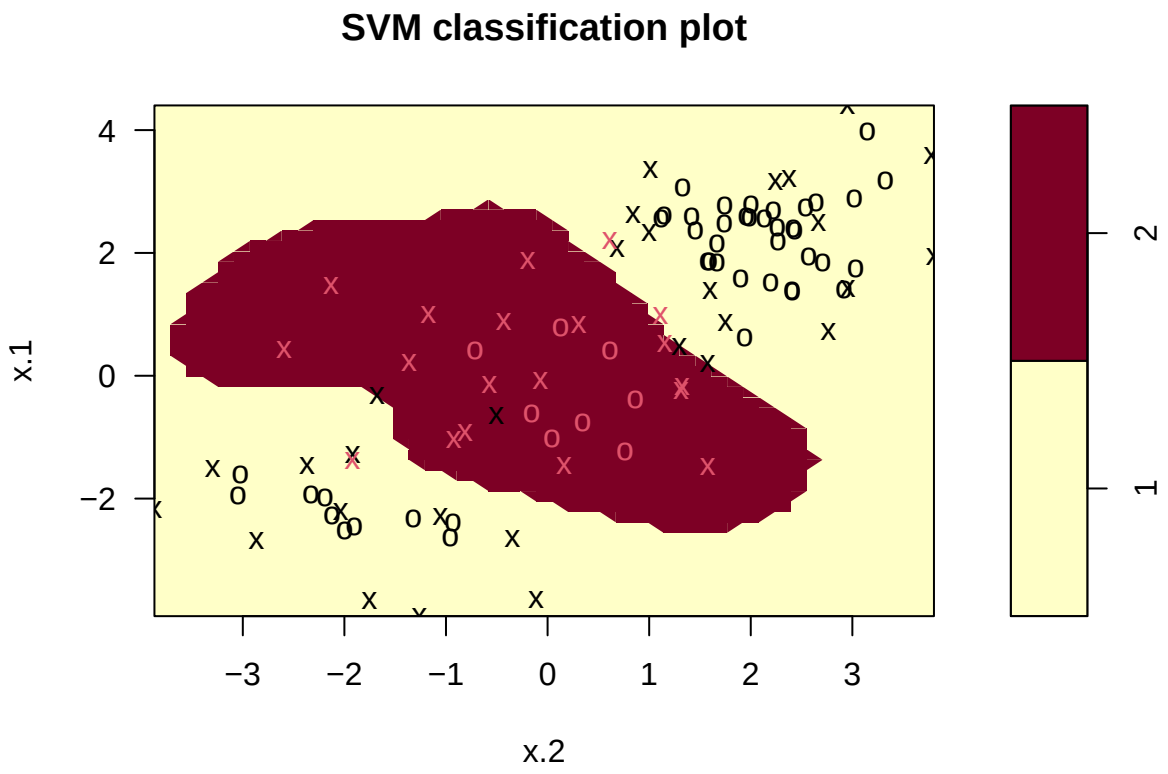
```
## Warning: package 'e1071' was built under R version 4.2.3
```

```
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.

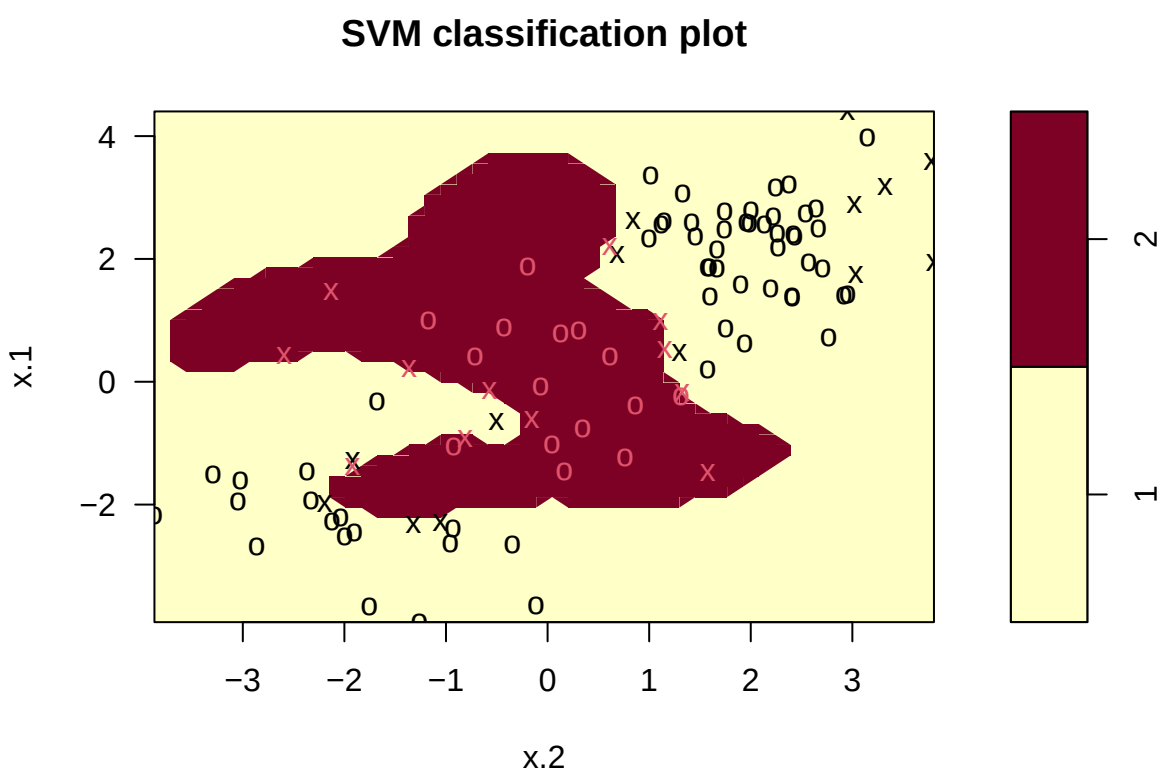
```
train_indices <- sample(1:nrow(dat), 100)
train <- dat[train_indices, ]
test <- dat[-train_indices, ]

svm_model <- svm(y ~ ., data = train, kernel = "radial", gamma = 1, cost = 1, scale= FALSE)

plot(svm_model, data=train)
```



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.

```
svm_model2 <- svm(y ~ ., data = train, kernel = "radial", gamma = 1, cost = 10000, scale=FALSE)

plot(svm_model2, data=train)
```

[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.

This model is better at capturing the training data, it clearly does a great job of separating the reds from the blacks; however, this is instead introducing over-fitting which means that this particular model would not work on data outside of this training set whatsoever.

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?

It does appear that this svm incorrectly (over) predicts class 2 at a higher rate than it incorrectly predicts class one.

```
predictions <- predict(svm_model, newdata = test)

# Create a table of true vs. predicted values
result_table <- table(true = test$y, pred = predictions)

print(result_table)
```

```
##     pred
```

```
## true  1  2
##    1 66 11
##    2  4 19
```

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.

```
total_prop_class_2 <- mean(dat$y == 2)


train_prop_class_2 <- mean(train$y == 2)


total_prop_class_2
```

```
## [1] 0.25
```

```
train_prop_class_2
```

```
## [1] 0.27
```

Student Response This disparity does not appear to be because of an imbalance in training/testing partition. Above, it is shown that the proportion of class 2 in the training set is .28, which is fairly representative of the true proportion of class 2 which is 0.25. Rather, this imbalance likely comes from the overfitting that occurred by ratcheting up the cost.

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)
library(e1071)


cost_values <- c(0.1, 1, 10, 100, 1000)
gamma_values <- c(0.5, 1, 2, 3, 4)


parameter_grid <- expand.grid(cost = cost_values, gamma = gamma_values)


tune.out <- tune(svm, y ~ ., data = train, kernel = "radial", ranges = list(cost = cost_values, gamma =
```

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

predictions <- predict(tune.out$best.model, newdata = test)


confusion_matrix2 <- table(true = test$y, pred = predictions)

# Print the confusion matrix
print(confusion_matrix2)
```

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 After tuning the hyperparameters, we can see that the new confusion matrix closely resembles the previous one, suggesting a modest enhancement in performance. Notably, there's a discernible boost in accuracy, with a noticeable reduction in the tendency to overpredict class 2. However, it's still important to ensure that the training data remains representative of the broader sample. Even with well optimized hyperparameters, a mismatch between biased training and testing data could still impede the model's effectiveness and utility. # Let's turn now to decision trees.

```
#install.packages("kmed")
library(kmed)
```

```
## Warning: package 'kmed' was built under R version 4.2.3
```

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

```
## Warning: package 'tree' was built under R version 4.2.3
```

```
head(heart)
```

```
##   age   sex cp trestbps chol   fbs restecg thalach exang oldpeak slope ca thal
## 1  63  TRUE  1      145  233  TRUE       2     150 FALSE     2.3     3  0    6
## 2  67  TRUE  4      160  286 FALSE       2     108  TRUE     1.5     2  3    3
## 3  67  TRUE  4      120  229 FALSE       2     129  TRUE     2.6     2  2    7
## 4  37  TRUE  3      130  250 FALSE       0     187 FALSE     3.5     3  0    3
## 5  41 FALSE  2      130  204 FALSE       2     172 FALSE     1.4     1  0    3
## 6  56  TRUE  2      120  236 FALSE       0     178 FALSE     0.8     1  0    3
##   class
## 1     0
## 2     2
## 3     1
## 4     0
## 5     0
## 6     0
```

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.

```
library(dplyr)
```

```
## Warning: package 'dplyr' was built under R version 4.2.3
```

```
##
## Attaching package: 'dplyr'
```

```
## The following objects are masked from 'package:stats':
##
##     filter, lag
```

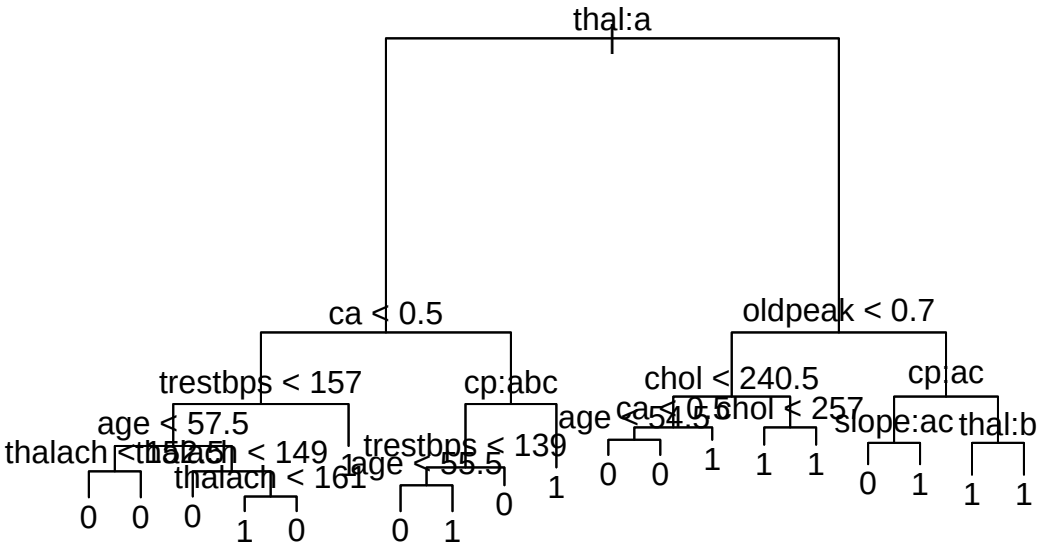
```
## The following objects are masked from 'package:base':
##
##     intersect, setdiff, setequal, union
```

```
heart <- heart%>%mutate(heart_disease=ifelse(class == 0, 0, 1))

heart$heart_disease <- factor(heart$heart_disease, levels=c(0,1))

heart <- subset(heart, select = -class)

# Check the structure of the data
str(heart)
```

```
## 'data.frame':    297 obs. of  14 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 ...
##  $ heart_disease: Factor w/ 2 levels "0","1": 1 2 2 1 1 1 2 1 2 2 ...
```

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

```r
set.seed(101)


train_indices <- sample(1:nrow(heart), 240, replace=FALSE)
train <- heart[train_indices, ]
test_data<-heart[-train_indices,]

tree_model <- tree(heart_disease ~ ., data = train)


plot(tree_model)
text(tree_model)
```



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

```r
#test_indices <- !(1:nrow(heart) %in% train)
#test_indices<- heart[-train,]


tree.pred <- predict(tree_model, newdata = heart[-train_indices, ], type = "class")
```

```r
conf_matrix <- table(tree.pred, heart$heart_disease[-train_indices])

print(conf_matrix)
```

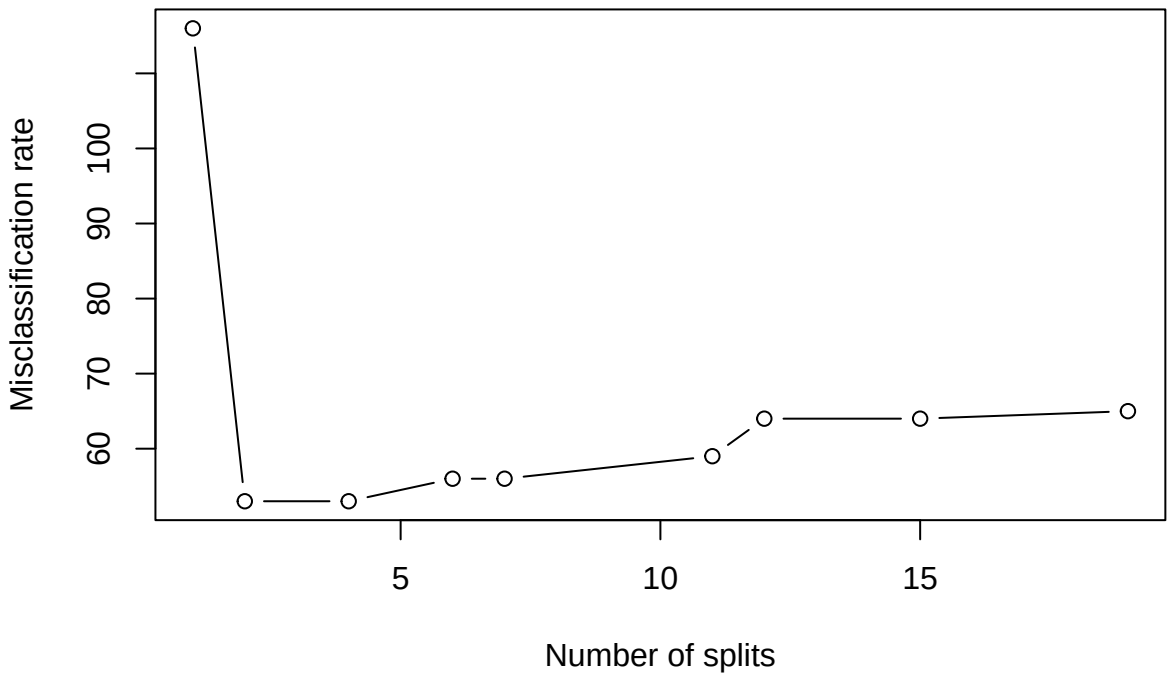
```
##
## tree.pred  0  1
##         0 28  3
##         1  8 18
```

```r
error_rate <- 1 - sum(diag(conf_matrix)) / sum(conf_matrix)
error_rate
```

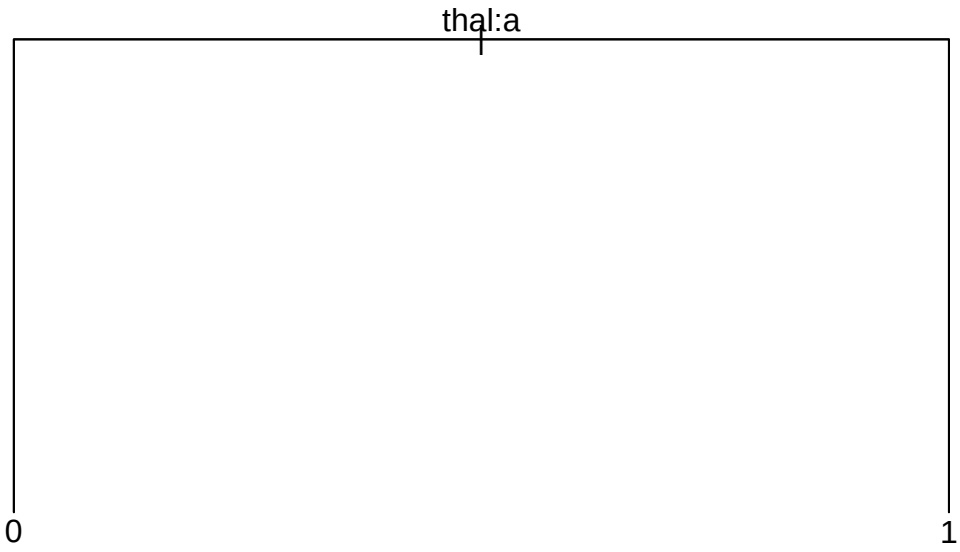
```
## [1] 0.1929825
```

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.

```r
set.seed(101)
cv_result <- cv.tree(tree_model, FUN = prune.misclass)


plot(cv_result$size, cv_result$dev, type = "b", xlab = "Number of splits", ylab = "Misclassification ra
```




```
#ideal_splits <- cv_result$size[which.min(cv_result$dev)]
#ideal_splits


pruned_tree <- prune.misclass(tree_model, best = 2)

# Plot the pruned tree
plot(pruned_tree)
text(pruned_tree)
```




```
pruned_predictions <- predict(pruned_tree, newdata = heart[-train_indices,], type = "class")


conf_matrix_pruned <- table( test_data$heart_disease,pruned_predictions)

conf_matrix_pruned
```

```
##    pruned_predictions
##      0  1
##   0 28  8
##   1  9 12
```

```
# Calculate the misclassification rate
misclassification_rate <- 1 - sum(diag(conf_matrix_pruned)) / sum(conf_matrix_pruned)
misclassification_rate
```

```
## [1] 0.2982456
```

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

The trade off in accuracy and interpretability in pruning the above tree lies in the fact that the accuracy of classification inherently goes down (in this case by 10%) when tuning on fewer parameters, however, the

interpretability becomes much clearer. Rather than looking at a decision tree that has so many possible pathways with a muddled overall interpretation, the pruned tree represents a fairly simplistic system for classification.

Discuss the ways a decision tree could manifest algorithmic bias.

Student Answer

A decision tree could manifest algorithmic bias in multiple ways. First, if the training data set is not representative of the population as a whole. For example, if the training set contains 95% of a hypothetical class 2, but the population is truly only 30% class 2(these numbers most certainly do not work out), then the prediction rate of class 2 would be way higher than it should be. Thus, the misclassification rate would be very high due to bias in the training-testing split. Further, when the models are set to lean towards interpretability over complexity, it is possible that features that are highly correlated with other features are cut out. An example of this being if there is a common trend in lower levels of education by gender, race, or other demographic grouping and the metric chosen by the tree is instead that of wages (high or low), which is typically highly correlated with years of education then there is now a demographic bias introduced to the model.