HW 3

Student Name

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).

Expanding $(X - E[X])^2$ *yields* $X^2 - 2X * E[X] + E[X]^2$

So, $E[(X - E[X])^2] = E[X^2 - 2X * E[X] + E[X]^2]$

Since E *is linear, applying this property yields* $E[X^2 - 2X * E[X] + E[X]^2] = E[X^2] + E[-2X * E[X]] + E[E[X]^2]$

Now, we may focus on the second term of our above expression.

Again, since E *is linear, and* $E[X]$ *and* -2 *are both constants, we may "pull out" the* $E[X]$ *and* -2 *from* $E[-2X * E[X]]$ *to get* $-2 * E[X] * E[X]$.

So our entire expression becomes $E[X^2] - 2 * E[X] * E[X] + E[E[X]^2]$.

Since $E[X]^2$ *is a constant, and the expectation of a constant is just that constant,* $E[E[X]^2] = E[X]^2$. *So our above expression becomes* $E[X^2] - 2 * E[X] * E[X] + E[X]^2$

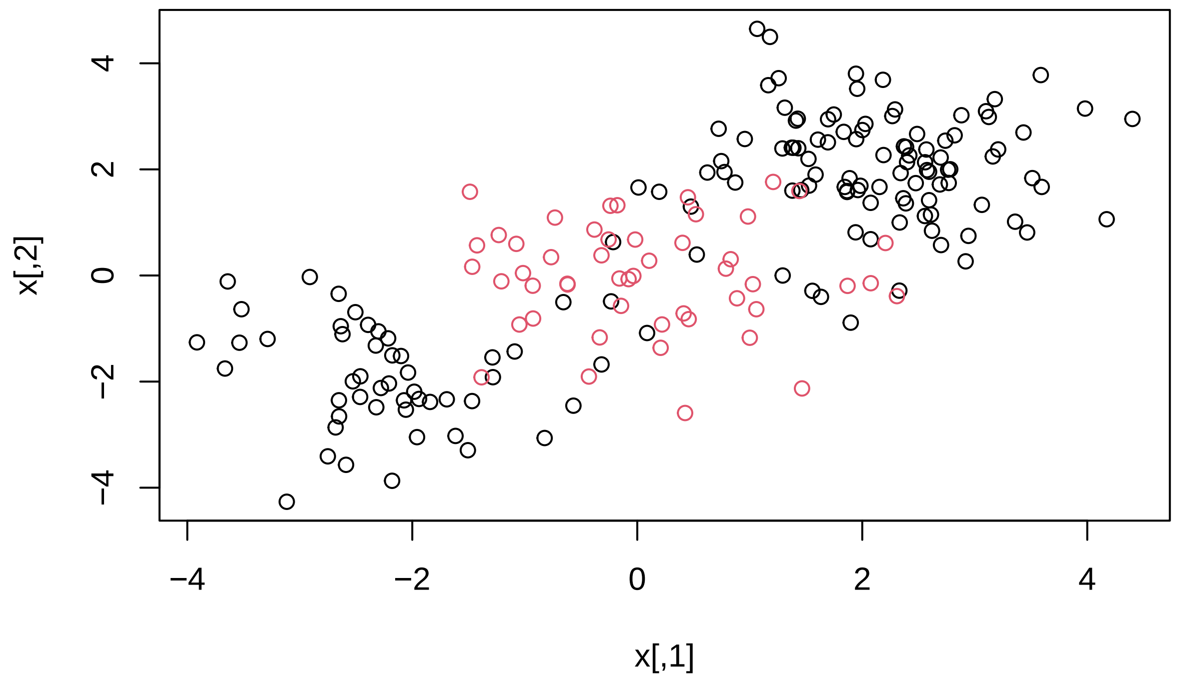
This, in turn, is equivalent to

$$E[X]^2 - 2E[X]^2 + E[X]^2$$

which, when combining like terms, simplifies to $E[X^2] - (E[X])^2$.

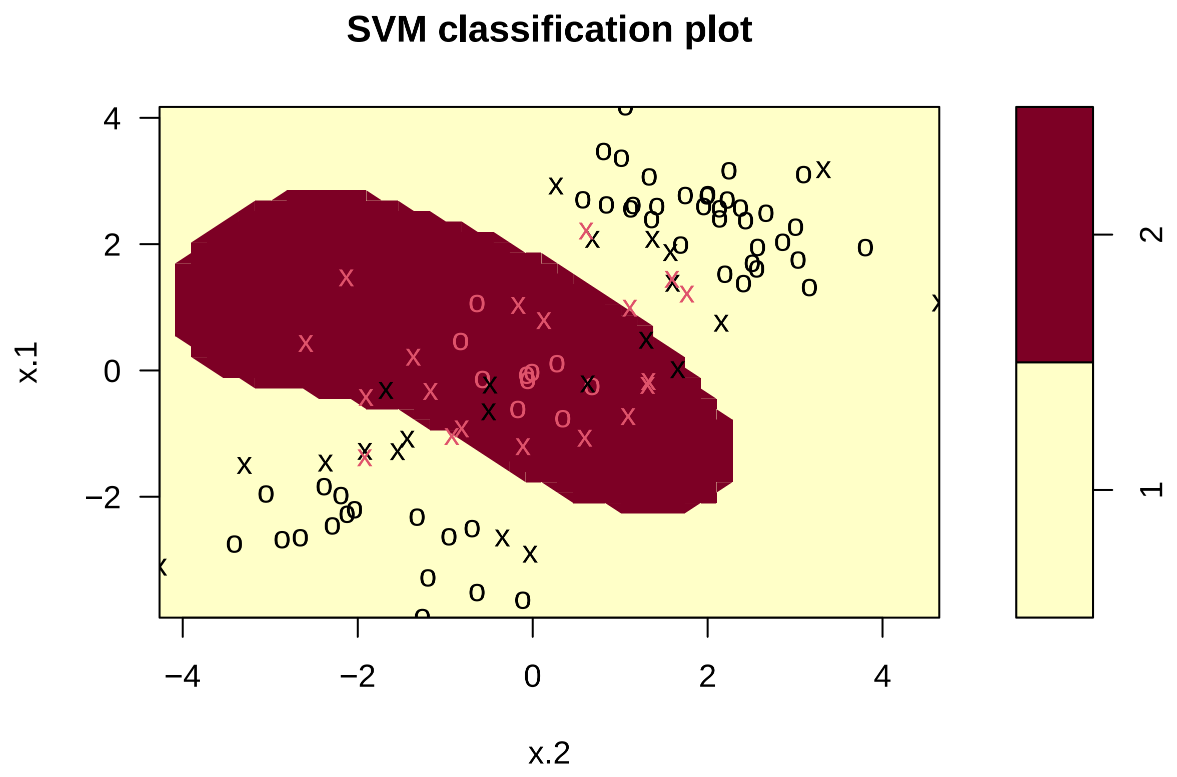
In the computational section of this homework, we will discuss support vector machines and tree-based methods. I will begin by simulating some data for you to use with SVM.

```r
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)
```



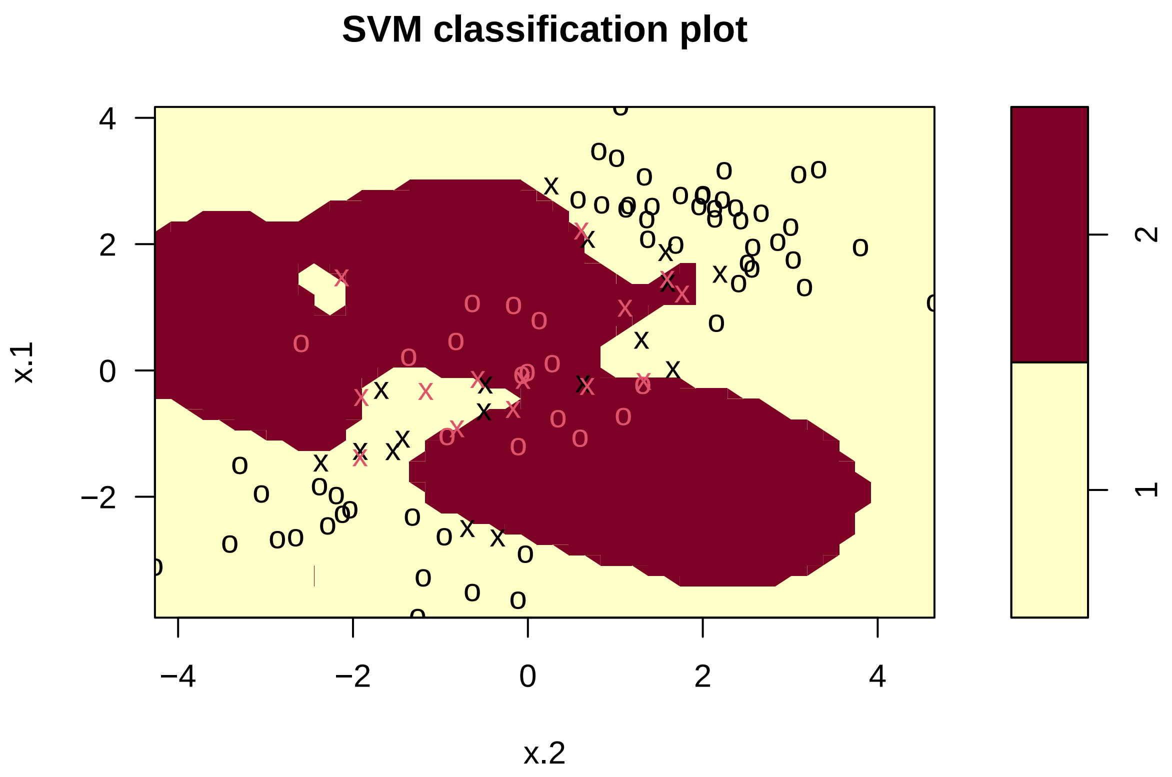
Quite clearly, the above data is not linearly separable. Create a training-testing partition with 100 random observations in the training partition. Fit an svm on this training data using the radial kernel, and tuning parameters $\gamma = 1$, cost $= 1$. Plot the svm on the training data.

```r
library(e1071)
set.seed(1)

train = sample(200,100)

traindat = dat[train,]

svmfit = svm(y~., data = traindat, kernel = "radial", gamma = 1, cost = 1)

plot (svmfit, traindat)
```



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

```
library(e1071)
svmfit = svm(y~., data = traindat, kernel = "radial", gamma = 1, cost = 10000)

plot (svmfit, traindat)
```



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

Increasing the cost makes the model more susceptible to overfitting the training data, and thus failing to generalize on other data that exists outside the training set. This is because a higher cost imbues a greater penalty to points that are misclassified. This causes the model to accomodate patterns in the training data that may not actually be representative of the population/overall dataset. This in turn can cause the model to perform poorly on testing data.

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?

```r
accuracy <- function(x){
  sum(diag(x)/(sum(rowSums(x)))) * 100
}
#remove eval = FALSE in above

#testing data
tab1 = table(true=dat[-train,"y"], pred=predict(svmfit, newdata=dat[-train,]))
print("Confusion matrix for testing data")
```

```
## [1] "Confusion matrix for testing data"
```

```
tab1
```

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

```
accuracy (tab1)
```

```
## [1] 86
```

```
#training data
tab2 = table(true=dat[train,"y"], pred=predict(svmfit, newdata=dat[train,]))
print("Confusion matrix for training data")
```

```
## [1] "Confusion matrix for training data"
```

```
tab2
```

```
##     pred
## true  1  2
##    1 66  5
##    2  2 27
```

```
accuracy (tab2)
```

```
## [1] 93
```

The accuracy of this SVM on the testing data, 86%, is lower than the accuracy of this SVM on the training data, 93%. When run on the testing data, the algorithm misclassifies 15.2% of observations in which $Y = 1$ and 9.5% of the observations in which $Y = 2$. When run on the training data, the algorithm misclassifies 7.04% of observations in which $Y = 1$ and 6.9% of observations in which $Y = 2$. Thus, on the training data, the algorithm performs similarly for both classes of Y; but when run on the testing data, the algorithm more frequently misclassifies observations in which $Y = 1$. It also has a decreased accuracy overall when run on the testing data. ##

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(as.factor(y[train]))
```

```
##
##  1  2
## 71 29
```

```r
table(as.factor(y))
```

```
##
##   1   2
## 150  50
```

The proportion of class 2 in my training dataset is 29/(71+29) = 29%, and the proportion of class 2 in the data as a whole is 50/150 = 33.3%. Therefore, this disparity is not a result of imbalance in the training/testing partition, because they contain similar proportions of class 2. It is likely due to overfitting the decision boundary due to the high cost parameter we implemented.

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`.

```r
set.seed(1)
tune.out = tune(svm, y~., data = traindat, kernel = "radial", ranges = list(cost = c(0.1, 1, 10, 100, 1
```

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.

```r
tab3 = table(true=dat[-train,"y"], pred=predict(tune.out$best.model, newdata=dat[-train,]))
print("Tuned SVM on testing data")
```

```
## [1] "Tuned SVM on testing data"
```

```r
tab3
```

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

```r
accuracy (tab3)
```

```
## [1] 92
```

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.

The accuracy is higher than before for our testing data (92% compared to a prior 86%). This is likely because the model is not overfitting as severely. However, our original data was imbalanced (it only had 33.3% of data points belonging to class 2), our tuned SVM still misclassifies class 1 as class 2 more often than it misclassifies class 2 as class 1. It misclassified 7/79 = 8.86% of true class 1's as class 2, and 1/21 = 4.76% of true class 2's as class 1's.

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.

```
heartdisease = ifelse(heart$class==0, 0, 1)

#ensuring variable has been factorized correctly
is.factor(heartdisease)
```

```
## [1] FALSE
```

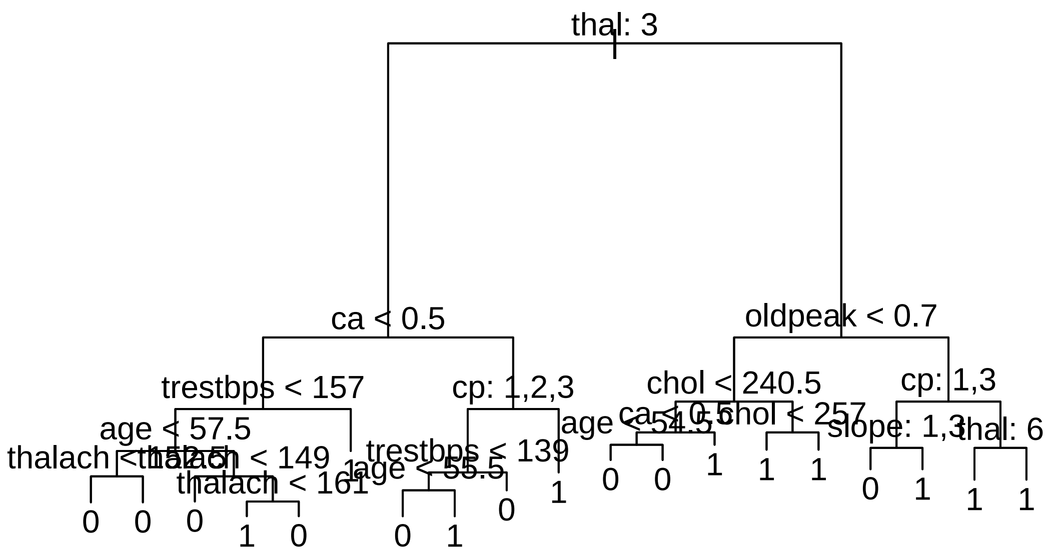
```
#It is not a factor, so we will turn it onto one
heartdisease = as.factor(heartdisease)
is.factor(heartdisease)
```

```
## [1] TRUE
```

```
#Replacing original column with factorized version to avoid confusion later on
heart$class = heartdisease
```

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

```
set.seed(101)
train = sample(1:nrow(heart), 240)

tree.heart = tree(class~., heart, subset=train)
plot(tree.heart)
text(tree.heart, pretty=0)
```



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

```
heart.pred = predict (tree.heart, heart[-train,], type = "class")
with (heart[-train,], table(heart.pred, class))
```

```
##           class
## heart.pred  0  1
##          0 28  3
##          1  8 18
```

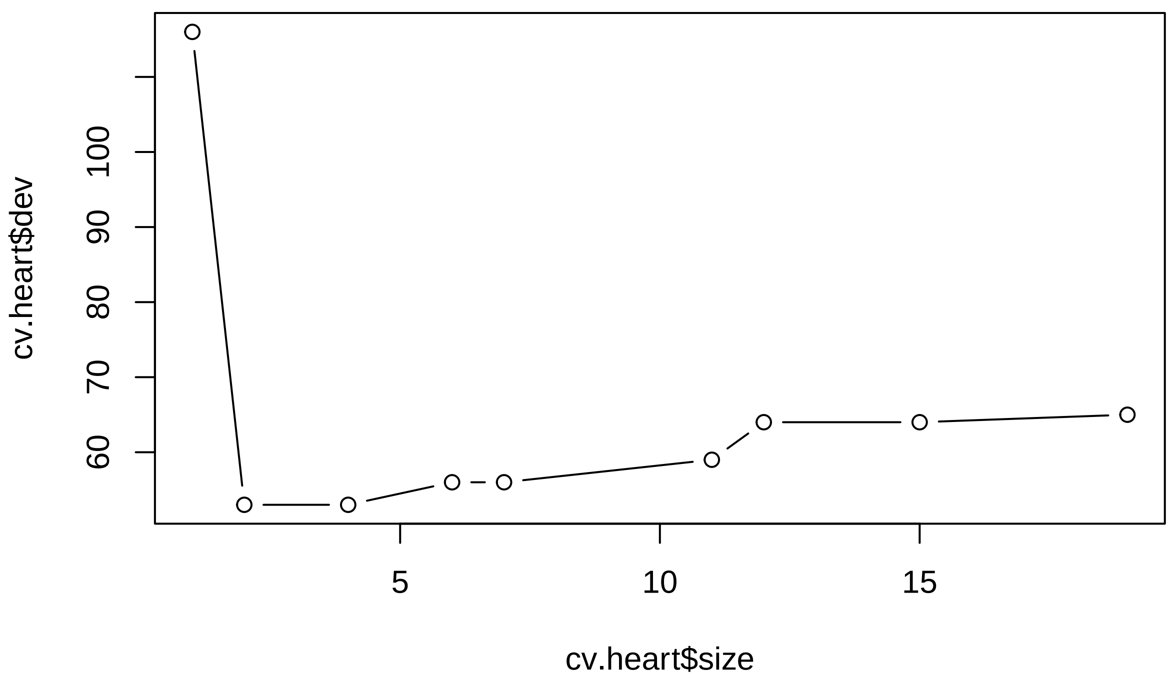
```
1 - ((28 + 18)/ (28 + 3 + 8 + 18))
```

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

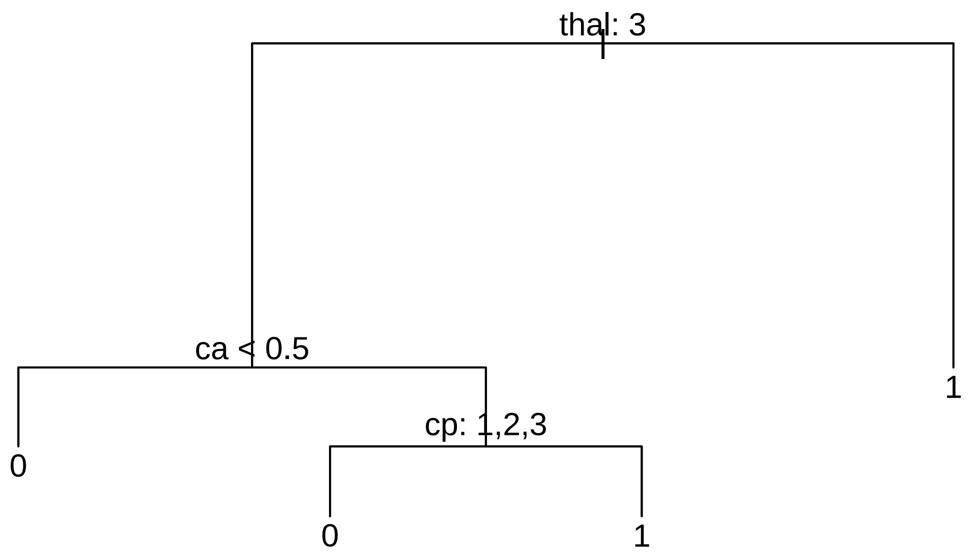
```
#Classification error is around 19.29%.
```

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.heart = cv.tree (tree.heart, FUN = prune.misclass)
plot (cv.heart$size, cv.heart$dev, type = "b")
```



```r
#ideal number of splits appears to be 3
prune.heart = prune.misclass(tree.heart, best = 3)
plot (prune.heart)
text(prune.heart, pretty = 0)
```



```r
tree.pred = predict(prune.heart, heart[-train,], type = "class")
with (heart[-train,], table (tree.pred,class))
```

```
##          class
## tree.pred  0  1
##         0 26  4
##         1 10 17
```

```r
1 - (26 + 17) / (26 + 4 + 10 + 17)
```

```
## [1] 0.245614
```

```r
#misclassification rate of around 24.56%
```

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

The above tree yields a lower accuracy and larger misclassification rate than our initial unpruned tree (compare around a 24% misclassification rate to around a 19% misclassification rate for our earlier tree.) However, this trade-off is worth it because of the increased interpretability it provides. The unpruned tree has so many branches that it is difficult to even read them, let alone determine which variables are responsible for classifying a patient's heart disease status. Our pruned tree retains only the most "important" predictors (the last ones to be removed when the tree is pruned). These are "thal," "ca" (specifically distinguishing between less than vs greater than 0.5), and "cp" (specifically distinguishing between values 1, 2, and 3, and all other values).

Discuss the ways a decision tree could manifest algorithmic bias.

Decision trees are extremely prone to overfitting, which is why it is important to prune them. If a tree is too "bushy," it may pick up on patterns in the training data that are not actually related to the outcome variable in the overall data set. This is because the model is attempting to make sure the nodes are as pure as possible, but in doing so, it may accidentally capture noise in the training set and fail to generalize to the testing data.

A non-representative data set can also manifest algorithmic bias. If the tree is trained on non-representative data, it may identify patterns that are not present in the overall data. This may cause the tree to fail to generalize well to the overall data set, therefore not accurately classifying the target variable.