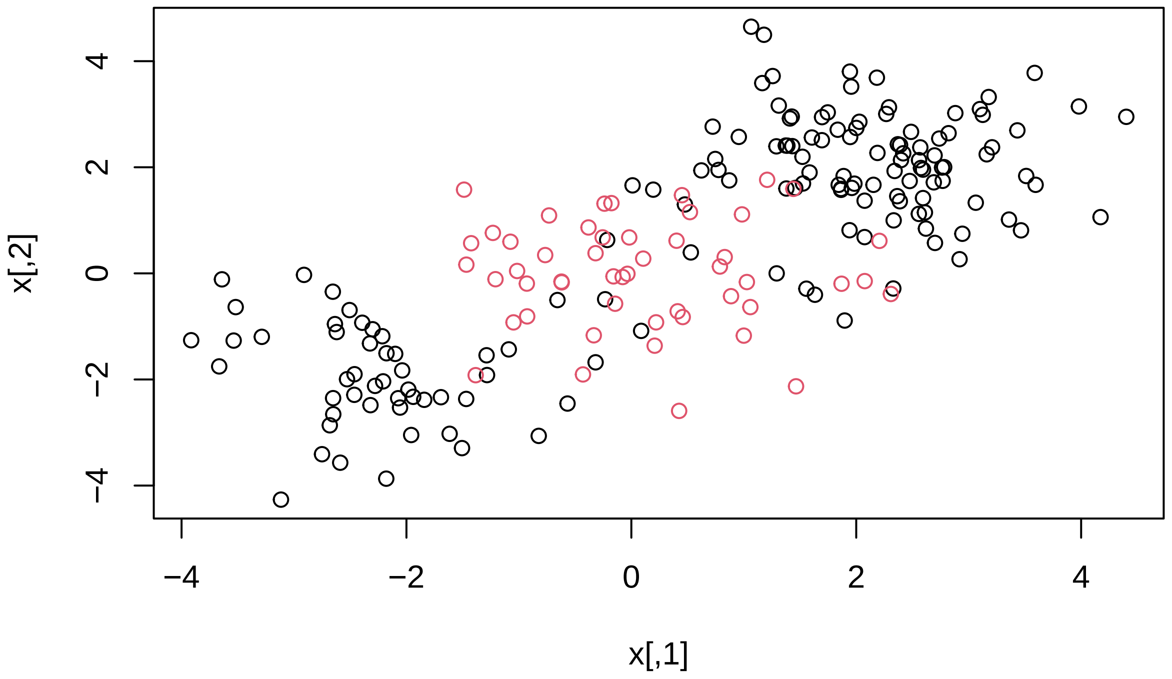
HW 3

Logan Schmitt

02/29/2024

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.

```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)
```



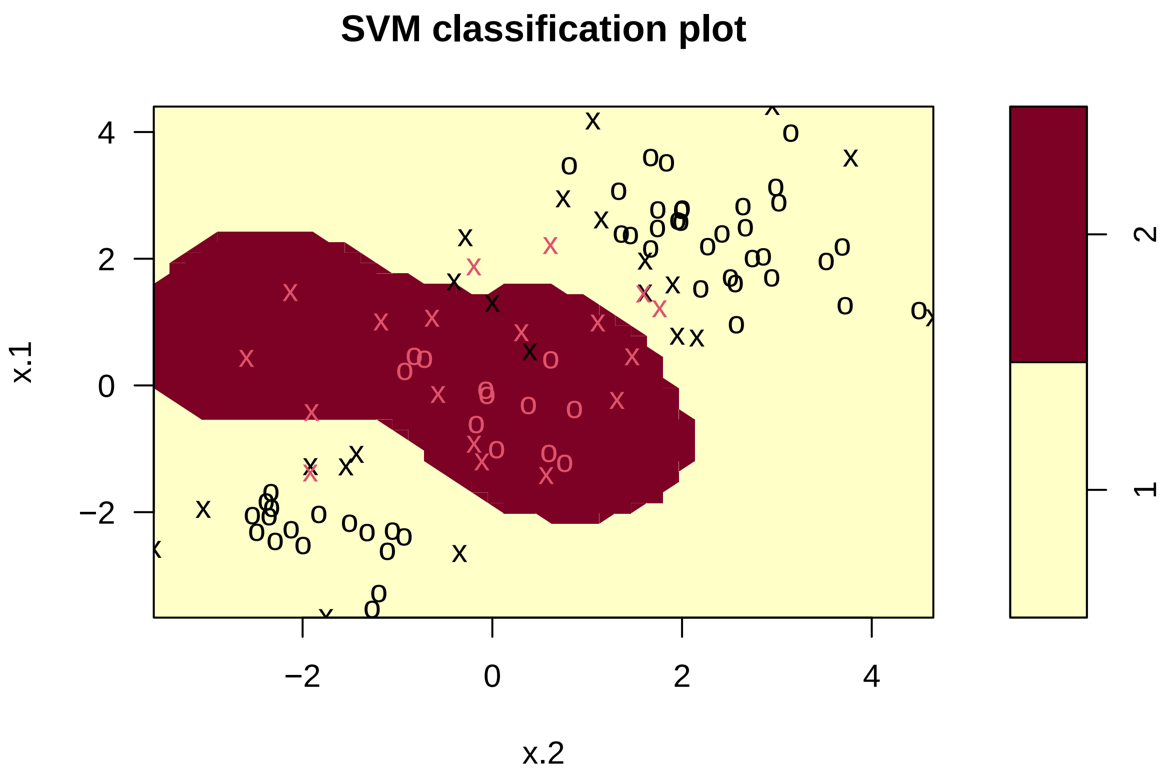
Quite clearly, the above data is not linearly separable. Create a training-testing partition with 100 random observations in the training partition. Fit an svm on this training data using the radial kernel, and tuning parameters $\gamma = 1$, cost $= 1$. Plot the svm on the training data.

```r
set.seed(3)
indices=sample(1:nrow(dat), 100)
train = dat[indices, ]
```

```r
svmfit = svm(y ~ .,
                data = train,
                kernel = "radial",
                cost = 1,
                scale = FALSE)
print(svmfit)
```

```
##
## Call:
## svm(formula = y ~ ., data = train, kernel = "radial", cost = 1, scale = FALSE)
##
##
## Parameters:
##    SVM-Type:  C-classification
##  SVM-Kernel:  radial
##        cost:  1
##
## Number of Support Vectors:  40
```

```r
plot(svmfit, 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.

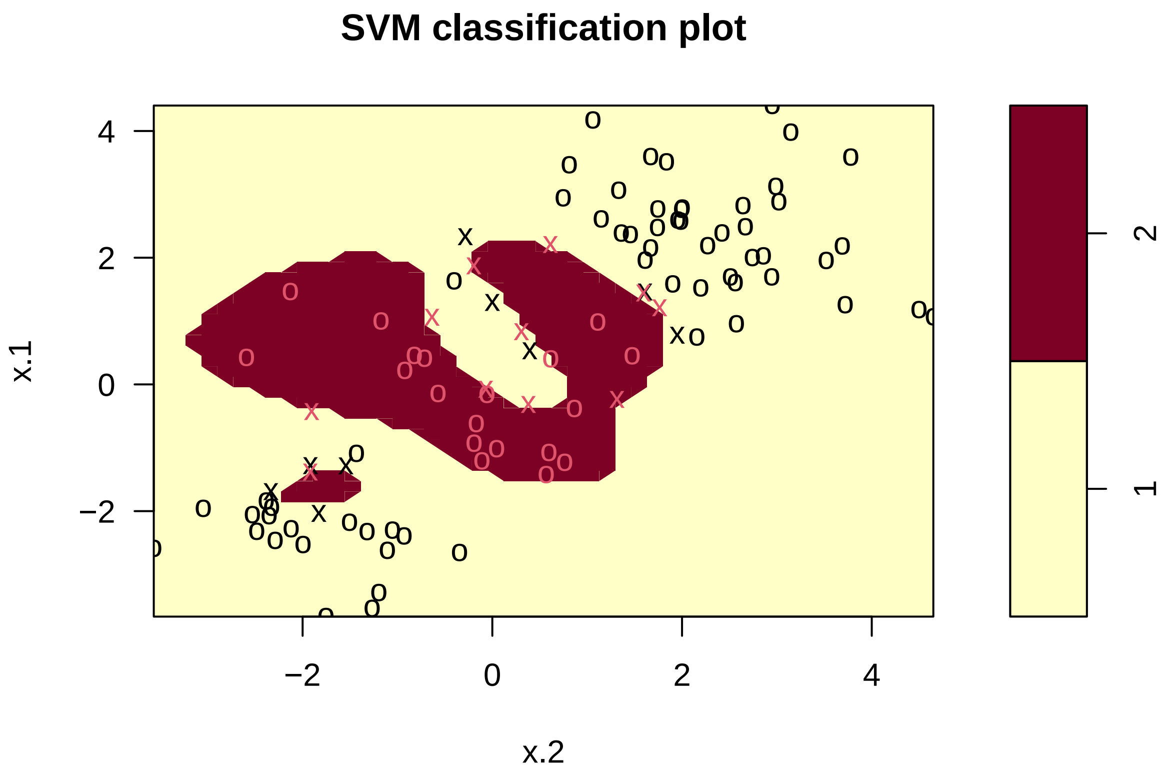
[1]Remember this is a parameter that decides how smooth your decision boundary should be

Refit the svm with the radial kernel, $\gamma = 1$, and a cost of 10000. Plot this svm on the training data.

```
svmfit2 = svm(y ~ .,
                data = train,
                kernel = "radial",
                cost = 10000,
                scale = FALSE)
print(svmfit2)
```

```
##
## Call:
## svm(formula = y ~ ., data = train, kernel = "radial", cost = 10000,
##     scale = FALSE)
##
##
## Parameters:
##    SVM-Type:  C-classification
##  SVM-Kernel:  radial
##        cost:  10000
##
## Number of Support Vectors:  20
```

```
plot(svmfit2, train)
```



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

Logan Response: By increasing the cost of an SVM model, there will be fewer misclassifications, however, the model will not be as generalizable for future data points.

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[-indices,"y"], pred=predict(svmfit2, newdata=dat[-indices,]))
```

```
##     pred
## true  1  2
##    1 68 12
##    2  9 11
```

```
class1.misclassification = 12/(68+12)
class2.misclassification = 9/(11+9)
class1.misclassification
```

```
## [1] 0.15
```

```
class2.misclassification
```

```
## [1] 0.45
```

Logan Response: There appears to be a much higher misclassification rate for Class 2 than for Class 1. There is potential evidence of overfitting

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.

```
prop = (9+11) / 100
prop
```

```
## [1] 0.2
```

Logan Response: The proportion of Class in the training partition is 20%, which is broadly representative of the underlying 25% of Class 2 in the data as a whole.

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)
tune.out = tune(svm,
                y~.,
                data = train,
                ranges = list(gamma = c(0.1, 1, 10, 100, 1000),
                              cost = c(0.5, 1,2,3,4)))
```

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

```
##     pred
## true  1  2
##    1 70 10
##    2  2 18
```

```r
newclass1.misclassification = 10/(70+10)
newclass2.misclassification = 2/(2+18)
newclass1.misclassification
```

```
## [1] 0.125
```

```r
newclass2.misclassification
```

```
## [1] 0.1
```

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.

Logan Response: Compared to the model used in Question 2, we have fewer misclassifications for both 1 and 2. For this improved model, there still cannot be an imbalance of training and testing data (the training set must be representative of the testing set).

Let's turn now to decision trees.

```r
library(kmed)
data(heart)
library(tree)
head(heart,1)
```

```
##   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
##   class
## 1     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.

```r
# Creating 2 classes by setting 0 for 0 values and 1 for non-0 values.
HeartDisease = ifelse(heart$class == 0, "0", "1")
heart = data.frame(heart, HeartDisease)

# Storing it as a factor
HeartDiseaseFac = as.factor(HeartDisease)
heart = data.frame(heart, HeartDiseaseFac)
head(heart)
```

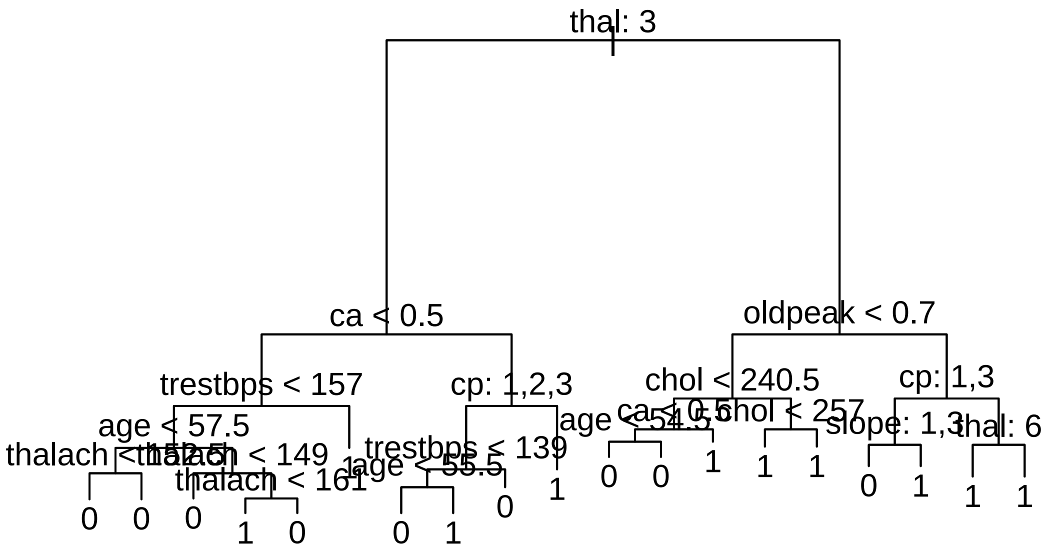
```
##   age   sex cp trestbps chol   fbs restecg thalach exang oldpeak slope ca thal
## 1  63  TRUE  1      145  233  TRUE       2     150 FALSE     2.3     3  0    6
## 2  67  TRUE  4      160  286 FALSE       2     108  TRUE     1.5     2  3    3
## 3  67  TRUE  4      120  229 FALSE       2     129  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 HeartDisease HeartDiseaseFac
## 1     0            0               0
## 2     2            1               1
## 3     1             1                1
## 4     0             0                0
## 5     0             0                0
## 6     0             0                0
```

```
# Dropping some columns:

to_drop = c("HeartDisease")
heart.new = heart[, !names(heart) %in% to_drop]

head(heart.new)
```

```
##   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 HeartDiseaseFac
## 1     0               0
## 2     2               1
## 3     1               1
## 4     0               0
## 5     0               0
## 6     0               0
```

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

```
set.seed(101)
train2=sample(1:nrow(heart.new), 240)

tree.heart = tree(HeartDiseaseFac~. -class, heart.new, subset=train2)
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.new[-train2,], type="class")
with(heart.new[-train2,], table(heart.pred, HeartDiseaseFac)) # Creates confusion matrix
```

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

```
class.error.rate = (3+8) / (28+8+3+18)
class.error.rate
```

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

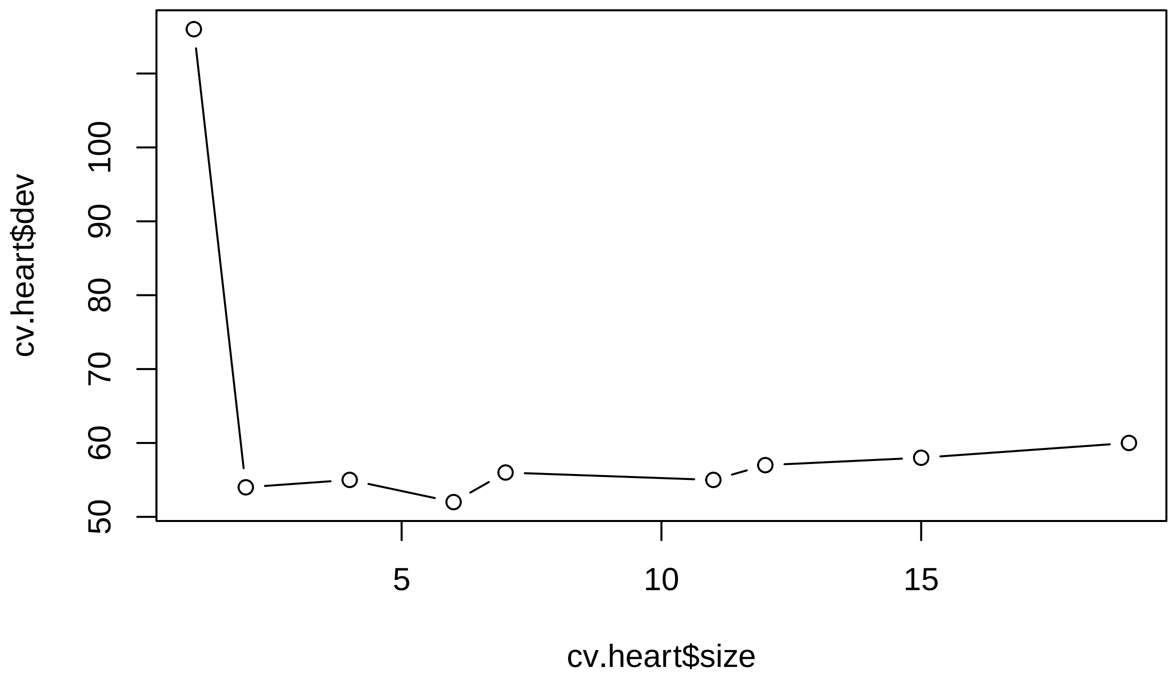
The classification error rate is ~19%.

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.

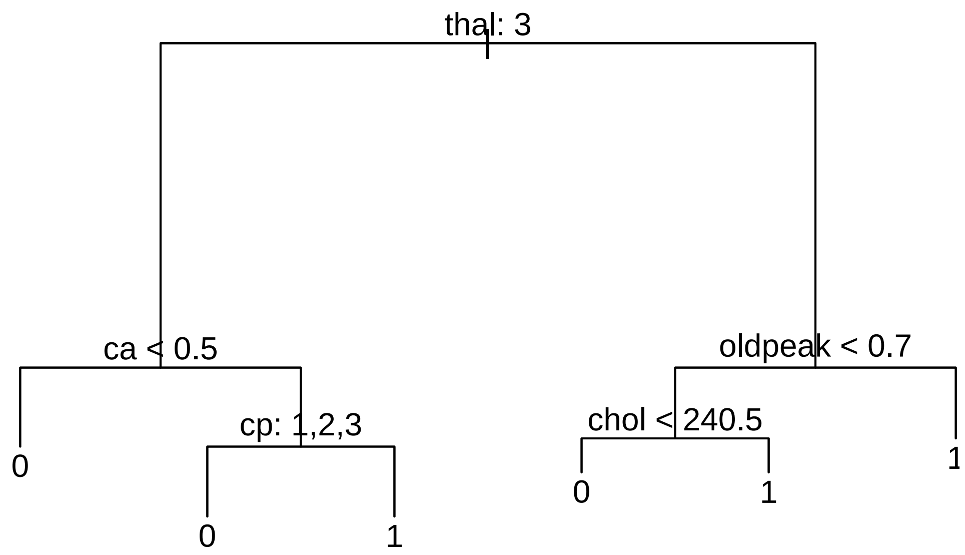
```
set.seed(13)
cv.heart = cv.tree(tree.heart, FUN = prune.misclass)
cv.heart
```

```
## $size
## [1] 19 15 12 11  7  6  4  2  1
##
## $dev
## [1]  60  58  57  55  56  52  55  54 116
##
## $k
## [1]       -Inf  0.0000000  0.6666667  1.0000000  1.5000000  2.0000000  3.5000000
## [8]  5.5000000 63.0000000
##
## $method
## [1] "misclass"
##
## attr(,"class")
## [1] "prune"         "tree.sequence"
```

```
plot(cv.heart$size, cv.heart$dev, type = "b")
```



```
# Use 6
prune.heart = prune.misclass(tree.heart, best = 6)

plot(prune.heart)
text(prune.heart, pretty=0)
```



```
heart.pred2 = predict(prune.heart, heart.new[-train2,], type="class")
with(heart.new[-train2,], table(heart.pred2, HeartDiseaseFac))
```

```
##              HeartDiseaseFac
## heart.pred2  0  1
##            0 29  5
##            1  7 16
```

```
# Misclassification Rate:
(5+7) / (29+5+7+16)
```

```
## [1] 0.2105263
```

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

Logan Input: Pruning allows for simpler models to be obtained, resulting in increase interpretability. However, this oftentimes can lead to less accuracy of the model, resulting in higher rates of misclassification.

Discuss the ways a decision tree could manifest algorithmic bias.

Logan Answer: A decision tree could manifest algorithmic bias in a variety of ways. One way this can be seen is with the variables present in the decision tree. In this problem, age is not present in the final decision tree, yet it can be a useful metric in determining whether a person has heart disease. However, this model may have been trained on data in which age did not have a major impact on determining who had heart disease. Here, we see poor sampling data (not representative of testing data) causing feature selection bias, thereby leading to algorithmic bias by causing higher misclassificaton rates by (un)favoring certain groups.