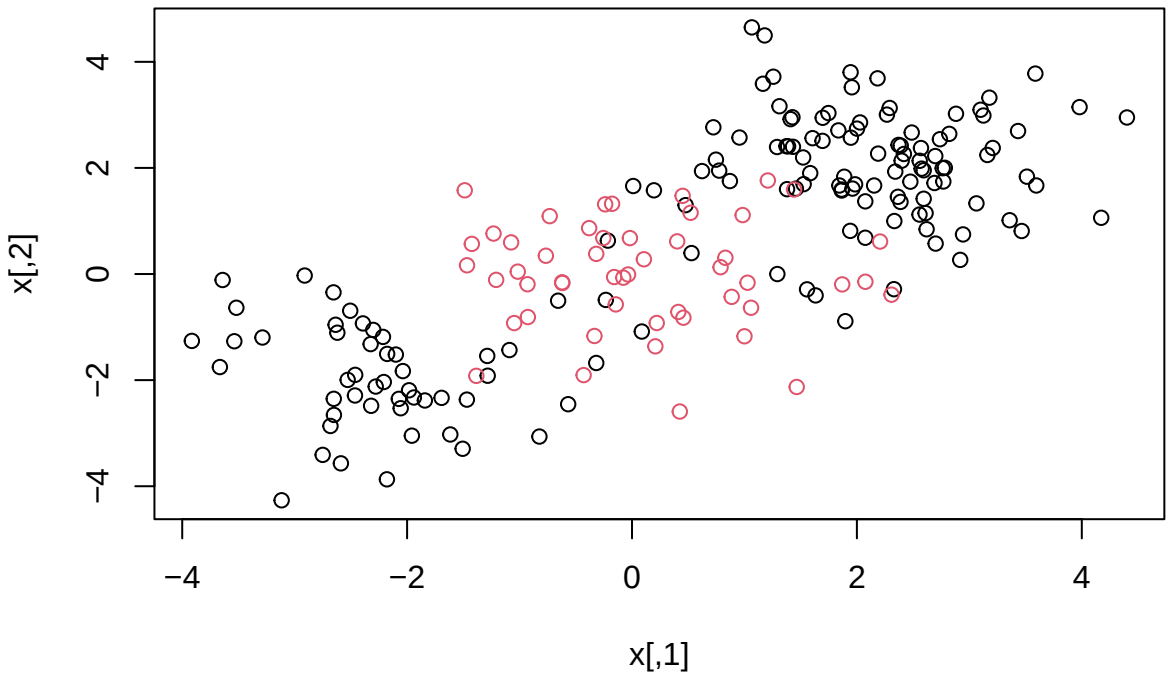
HW 3

IZ Raad

2/22/2024

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)
```

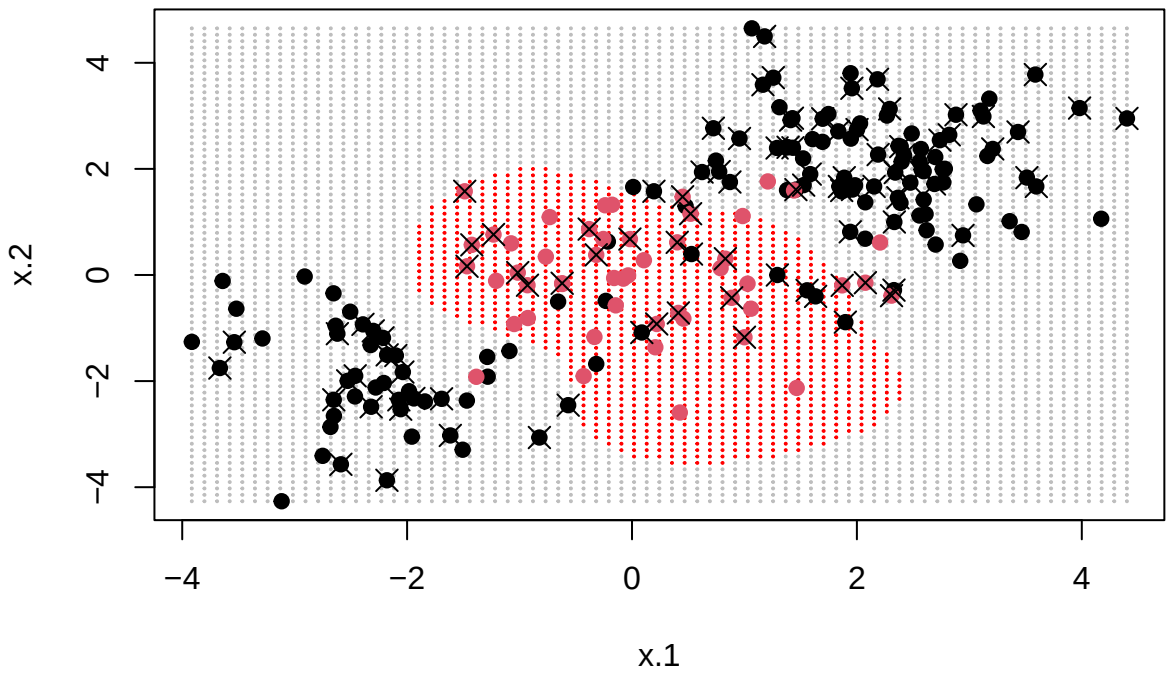


Quite clearly, the above data is not linearly separable. Create a training-testing partition with 100 random observations in the training partition. Fit an svm on this training data using the radial kernel, and tuning parameters $\gamma = 1$, cost $= 1$. Plot the svm on the training data.

```
# Create training-testing partition.
set.seed(1)
train_index = sample(1:nrow(dat), 100, replace=FALSE)
train <- dat[train_index,]
test <- dat[-train_index,]

# Fit svm with radial kernel to train.
svmfit <- svm(y~., data = train, kernel = "radial", gamma = 1, cost = 1,
              scale = FALSE)
print(svmfit)
```

```
##
## Call:
## svm(formula = y ~ ., data = train, kernel = "radial", gamma = 1,
##     cost = 1, scale = FALSE)
##
##
## Parameters:
##    SVM-Type:  C-classification
##  SVM-Kernel:  radial
##        cost:  1
##
## Number of Support Vectors:  54
```

```
# Make a grid.
make.grid = function(x, n = 75) {
  grange = apply(x, 2, range)
  x1 = seq(from = grange[1,1], to = grange[2,1], length = n)
  x2 = seq(from = grange[1,2], to = grange[2,2], length = n)
  expand.grid(x.1 = x1, x.2 = x2)
}
xgrid = make.grid(x)

# Put our prediction on the grid.
ygrid = predict(svmfit, xgrid)
plot(xgrid, col = c("gray","red")[as.numeric(ygrid)], pch = 20, cex = .2)
points(x, col = y, pch = 19)

# Put an x over test data.
points(test, pch = 4, cex = 1.5)
```



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.

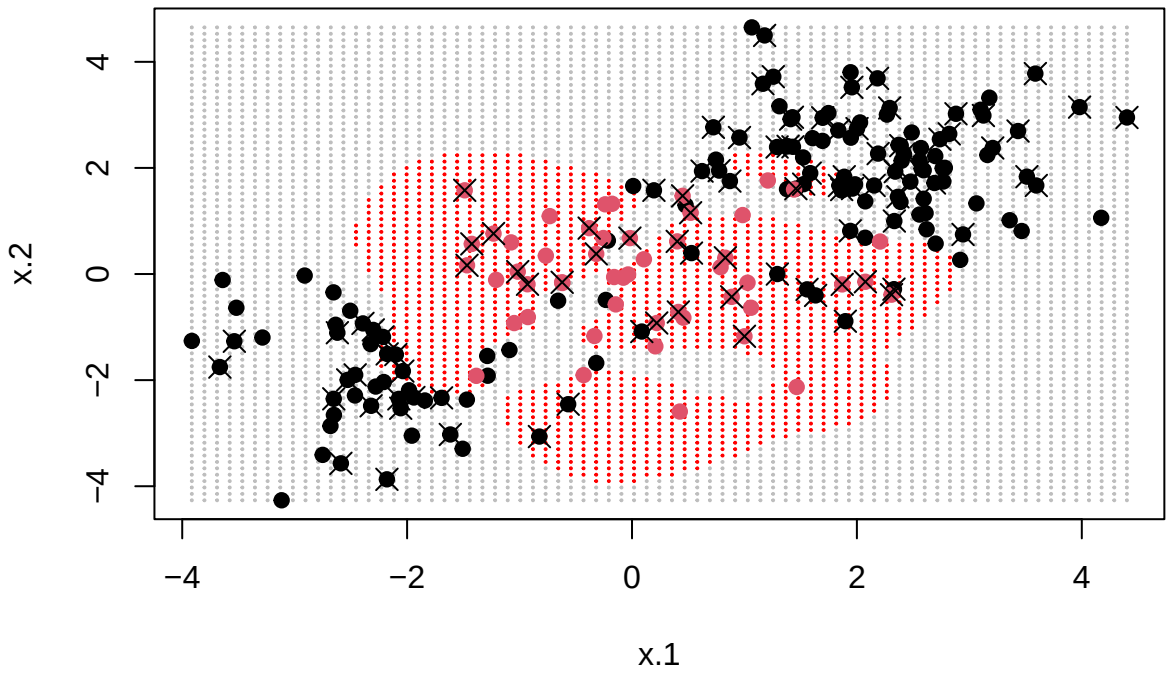
```
# Fit svm with radial kernel to train.
svmfit2 <- svm(y~., data = train, kernel = "radial", gamma = 1, cost = 10000,
               scale = FALSE)
print(svmfit2)
```

```
##
## Call:
## svm(formula = y ~ ., data = train, kernel = "radial", gamma = 1,
##     cost = 10000, scale = FALSE)
##
##
## Parameters:
##    SVM-Type:  C-classification
##  SVM-Kernel:  radial
##        cost:  10000
##
## Number of Support Vectors:  30
```

```
# Put our prediction on the grid.
ygrid2 = predict(svmfit2, xgrid)
```

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

```
plot(xgrid, col = c("gray","red")[as.numeric(ygrid2)], pch = 20, cex = .2)
points(x, col = y, pch = 19)

# Put an x over test data.
points(test, pch = 4, cex = 1.5)
```



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

It is true that increasing cost allows the model to more precisely fit our training data. However, this also leaves room for overfitting. Our model is warped in order to more precisely capture each individual point of our training data, even if this is not reflected in our population as a whole. Indeed, increasingly concave and disconnected areas were created in our region that would predict class two in order to individually accommodate points of class one in our training data set. However, points of class two in our testing data set still lie in these areas, and thus were misclassified.

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

```
##     pred
```

```
## true  1  2
##    1 62 17
##    2  3 18
```

The confusion matrix reveals a disparity in our ability to correctly classify data points. We are more likely to misclassify state one as state two (occurring 17 of 79 instances of state 1) than state two as state one (occurring 3 of 21 instances of state 1). Our model's bias towards falsely predicting state two may be due to overfitting in the model.

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.

```
library(tidyverse)
```

```
## -- Attaching core tidyverse packages ------------------------ tidyverse 2.0.0 --
## v dplyr     1.1.4     v readr     2.1.4
## v forcats   1.0.0     v stringr   1.5.1
## v ggplot2   3.4.4     v tibble    3.2.1
## v lubridate 1.9.3     v tidyr     1.3.0
## v purrr     1.0.2
## -- Conflicts ------------------------------------------ tidyverse_conflicts() --
## x dplyr::filter() masks stats::filter()
## x dplyr::lag()    masks stats::lag()
## i Use the conflicted package (<http://conflicted.r-lib.org/>) to force all conflicts to become errors
```

```
nrow(filter(train, y == 2)) / nrow(train)
```

```
## [1] 0.29
```

29% of the data in our training partition belongs to class two, which is similar enough to the 25% of our original dataset. Therefore, we may not conclude that disparities in the training data are causing the observed bias in our model.

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 ~ x.1 + x.2, data = train, kernel = "radial",
                 ranges = list(cost = c(0.1, 1, 10, 100, 1000),
                               gamma = c(0.5, 1,2,3,4)),
                 scale = FALSE)
```

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

```
##     pred
## true  1  2
##    1 73  6
##    2  3 18
```

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.

6 of 79 members of class one were misclassified as class two, a great improvement from the previous model's 17 misclassifications as class two. The number of class two events misclassified as class one remains the same. This model seems to have decreased bias towards falsely classifying as class two.

Let's turn now to decision trees.

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

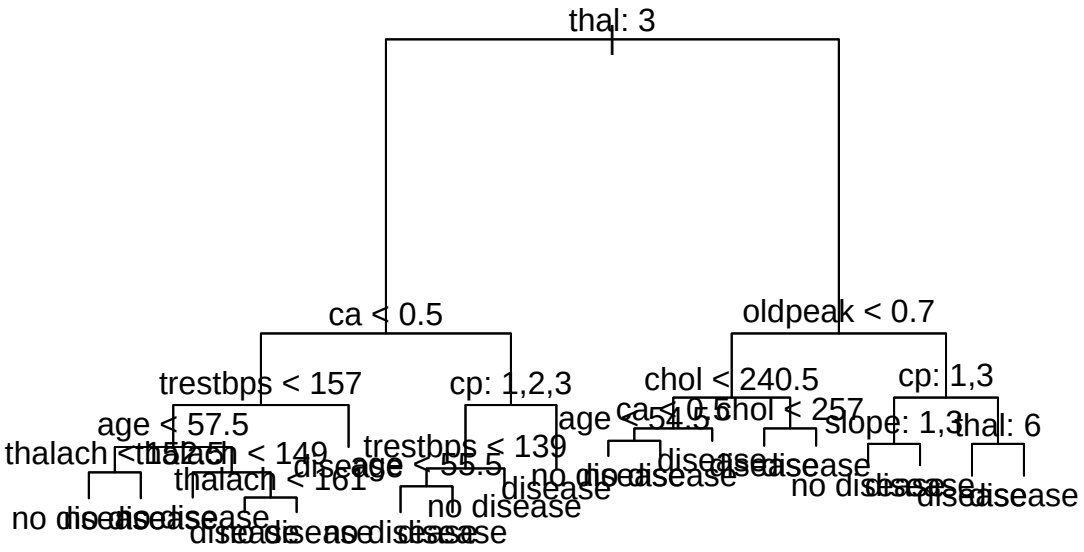
The response variable is currently a categorical variable with four levels. Convert heart disease into binary categorical variable. Then, ensure that it is properly stored as a factor.

```
heartclass = ifelse(heart$class > 0, 1, 0)
heart$class <- as.factor(heartclass)
levels(heart$class) <- c("no disease", "disease")
str(heart$class)
```

```
##  Factor w/ 2 levels "no disease","disease": 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.

```
set.seed(101)

# Partition heart into training and testing data.
train_indexh = sample(1:nrow(heart), 240, replace=FALSE)
trainh <- heart[train_indexh,]
testh <- heart[-train_indexh,]

treeh <- tree(class~., data = trainh)
plot(treeh)
text(treeh, pretty=0)
```



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

```
confh <- table(true=testh[,"class"], pred=predict(treeh, newdata = testh,
                                                  type = "class"))
confh
```

```
##             pred
## true         no disease disease
##   no disease         28       8
##   disease             3      18
```

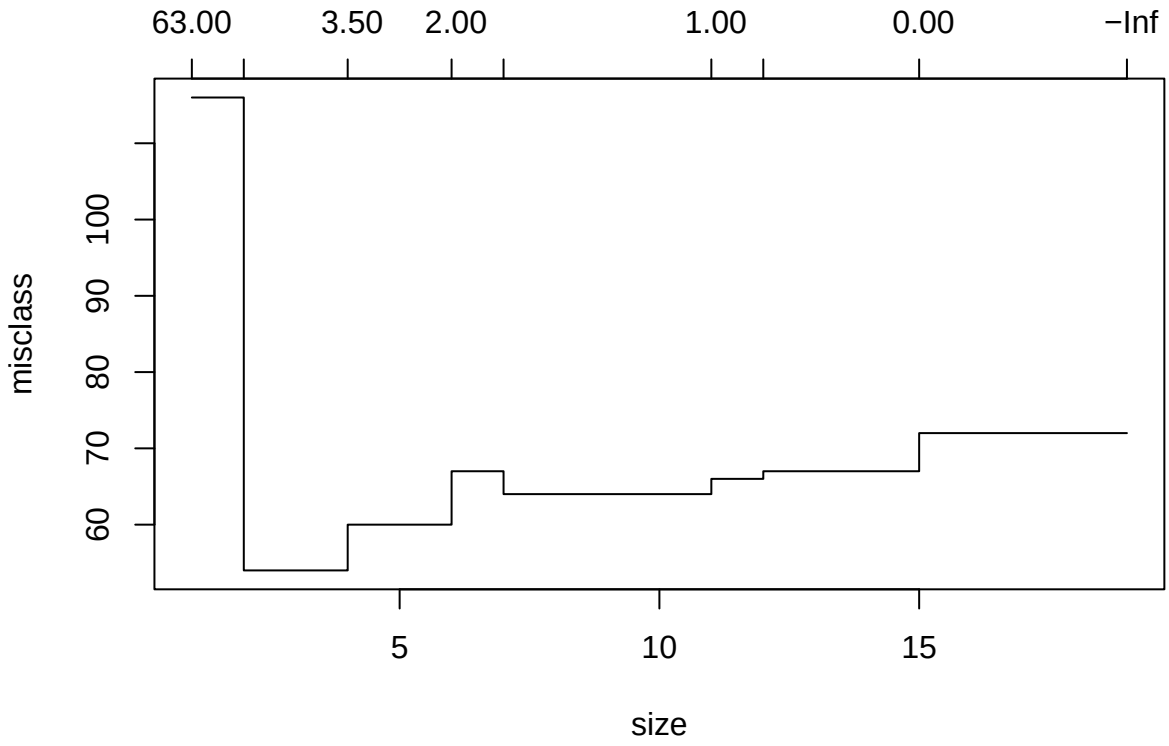
```
1 - (sum(diag(confh))/nrow(testh))
```

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

The classification error rate is 19.29%.

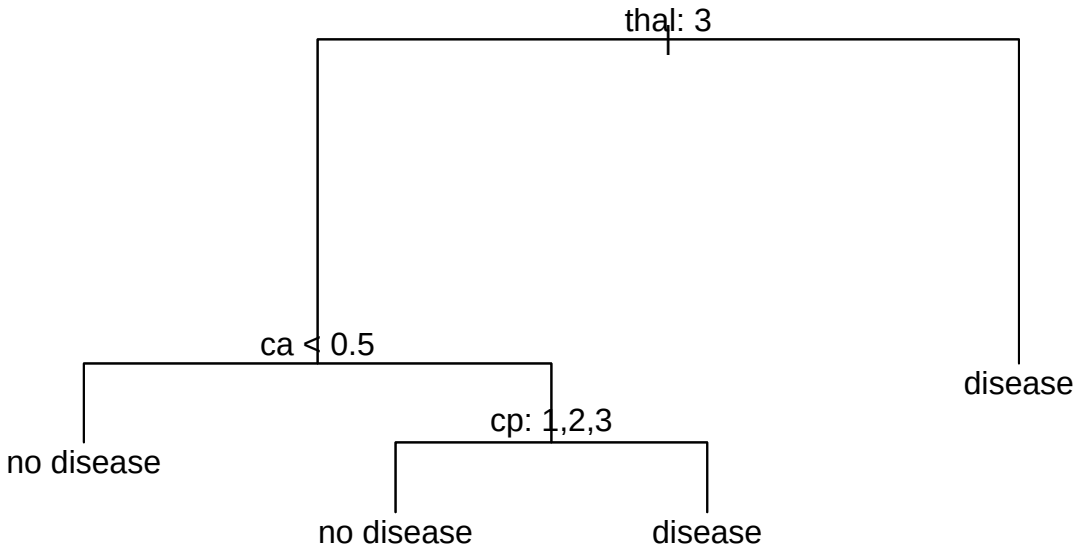
Above we have a fully grown (bushy) tree. Now, cross validate it using the `cv.tree` command. Specify cross validation to be done according to the misclassification rate. Choose an ideal number of splits, and plot this tree. Finally, use this pruned tree to test on the testing set. Report a confusion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

```
cv.treeh <- cv.tree(treeh, FUN= prune.misclass)
plot(cv.treeh)
```



It appears that 4 is the ideal number of splits, as this is the smallest size assigned to the minnimum misclassification rate.

```
pruned <- prune.misclass(treeh, best=4)
plot(pruned)
text(pruned, pretty=0)
```



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

```
confprun <- table(true=testh[,"class"], pred=predict(pruned, newdata = testh,
                                                     type = "class"))
confprun
```

```
##             pred
## true         no disease disease
##   no disease         26      10
##   disease             4      17
```

```
1 - (sum(diag(confprun))/nrow(testh))
```

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

Our pruned tree is more interpretable, with fewer deciding variables being included in this tree. However, our misclassification rate has increased by about 5%. Some may find this tradeoff worthwhile, as it reduces the number of (potentially confusing) variables. However, some situations may require no loss of accuracy. Thus, this trade-off should be evaluated situationally.

Discuss the ways a decision tree could manifest algorithmic bias.

A group underrepresented in the data may always be selected against by a decision tree. It is possible that, regardless of predictor, our data will not have sufficiently many members of this group to select it. This may

continue to perpetuate biases against this group.

Additionally, the decision variables that a tree selects may also perpetuate biases. People of different demographic and socioeconomic groups may be impacted by systemic inequality of opportunity. A decision tree may use these metrics as a way to predict items that may be linked to these inequalities, thus further perpetuating stereotypes. An example of this is predicting that someone of a specific income is going to score lower on tests, while it is known that there are socioeconomic based disparities in access to and quality of education.