# HW 3

Neel Singh

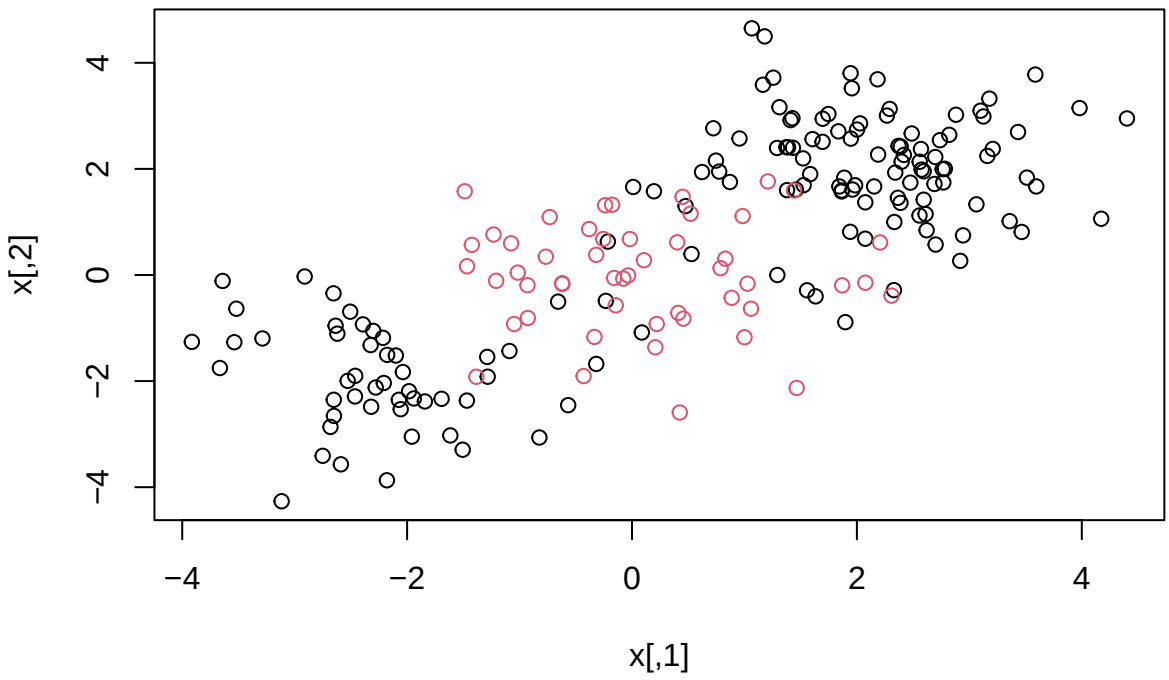
10/10/2024

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).

1. $$Var[X] = E[(X - E[X])^2] = E[X^2 - 2XE[X] + (E[X])^2]$$
2. Apply linearity of expectations
$$E[X^2 - 2XE[X] + (E[X])^2] = E[X^2] - 2(E[X])(E[X]) + E[(E[X])^2]$$
3. $E[X]$ is a constant, so $E[(E[X])^2] = (E[X])^2$
$$E[X^2] - 2(E[X])(E[X]) + (E[X])^2$$
4. $$E[X^2] - 2(E[X])^2 + (E[X])^2 = E[X^2] - (E[X])^2$$

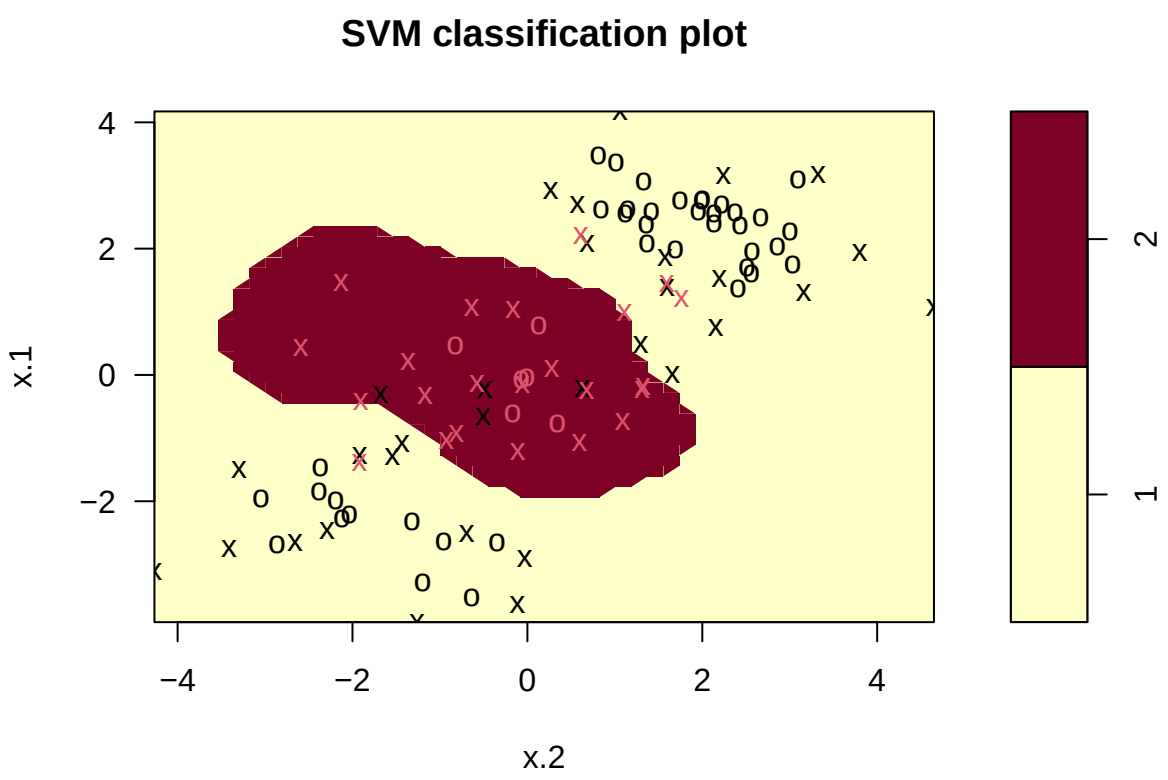
In the computational section of this homework, we will discuss support vector machines and tree-based methods. I will begin by simulating some data for you to use with SVM.

```
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.

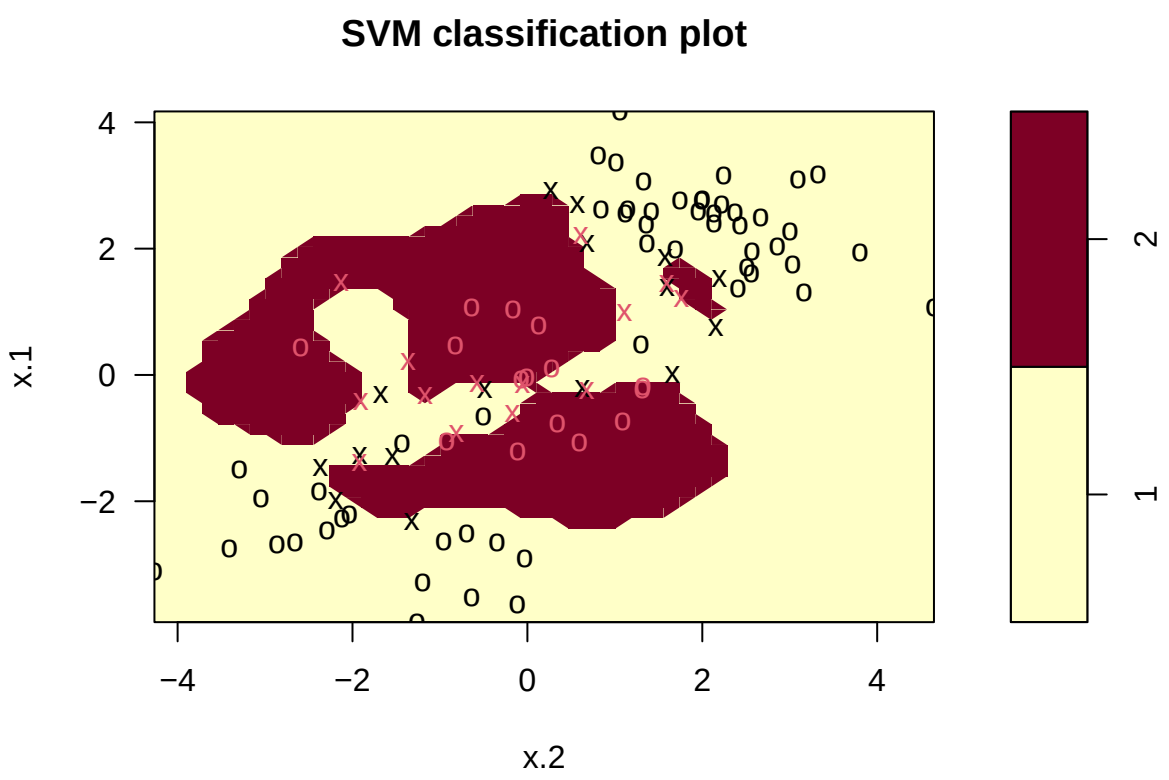
```
set.seed(1)
train <- sample(1:nrow(dat), 100)
svm1 <- svm(y ~., data = dat[train,], kernel = "radial", cost = 1, gamma = 1, scale = FALSE)
plot(svm1, data = dat[train,])
```




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.

```
set.seed(1)
svm2 <- svm(y ~., data = dat[train,], kernel = "radial", cost = 10000, gamma = 1, scale = FALSE)
plot(svm2, data = dat[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.

*By increasing the cost parameter to a **very** high value, we risk overfitting the model to the data. Clearly, there is a very complex decision boundary with a cost of 10000. The model may be learning the noise present in our dataset rather than the "true" relationship present. Thus, this model may not generalize well to unseen data (and could perform very badly).*

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,"y"], pred=predict(svm2, newdata=dat[-train,]))
```

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

*The matrix shows that we have pretty decent classification accuracy, with an error rate of 20%. It does, however, seem like there are more observations of class 1 that are being classified than those of class 2. The*

*classification error rate specifically for class 1 is 21.52%, versus a error rate specifically for class 2 of 14.29%. This suggests that the model is better at identifying class 2.*

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.

```
summary(dat[train,]$y)
```

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

*It seems like the training partition is broadly representative of the underlying 25% of class 2 in the dataset as a whole. Class 2 is 29% of the training partition.*

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 $\gamma$ 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 = dat[train,], kernel = "radial",
                 ranges = list(cost = c(0.1, 1, 10, 100, 1000),
                               gamma = 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[-train,"y"], pred=predict(tune.out$best.model, newdata=dat[-train,]))
```

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

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 new confusion matrix shows a much improved classification accuracy. It has a error rate of 8%. The model still incorrectly classifies observations from class 2 far less frequently than it does for class 1. Class 1 has a error rate of 8.86%, versus an error rate of 4.76% for class 2.*

Let's turn now to decision trees.

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

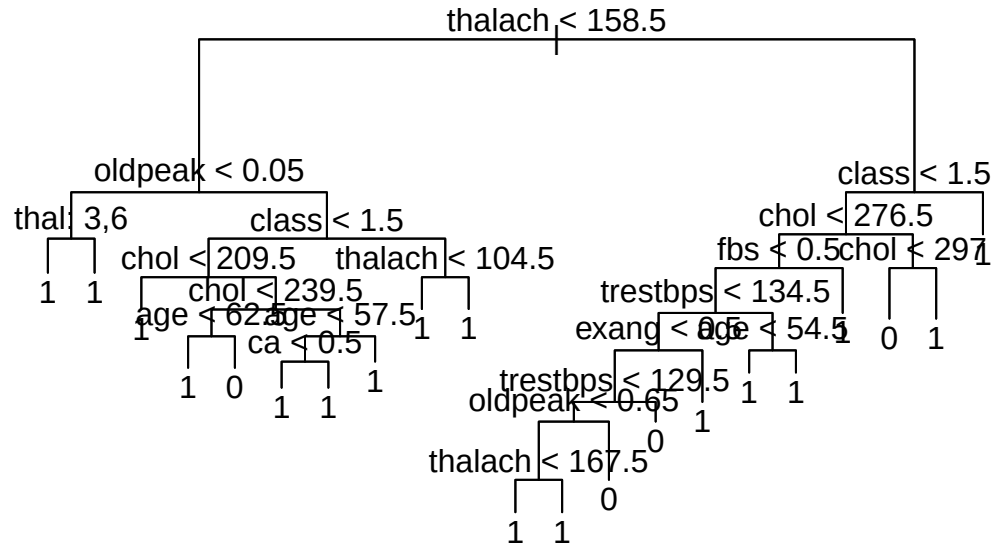
The response variable is currently a categorical variable with four levels. Convert heart disease into binary categorical variable. Then, ensure that it is properly stored as a factor.

```
heart$binary <- ifelse(heart$cp %in% c(3, 4), 1, 0)
heart$binary <- as.factor(heart$binary)
str(heart$binary)
```

```
##  Factor w/ 2 levels "0","1": 1 2 2 2 1 1 2 2 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)
heart_train <- sample(1:nrow(heart), 240)
heart_dtree <- tree(binary~.-cp, heart, subset = heart_train)
plot(heart_dtree)
text(heart_dtree, 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_test <- predict(heart_dtree, heart[-heart_train,], type = "class")

heart_matrix <- table(heart$binary[-heart_train], heart_test)
heart_matrix
```

```
##    heart_test
##      0  1
##   0  1 16
##   1  3 37
```

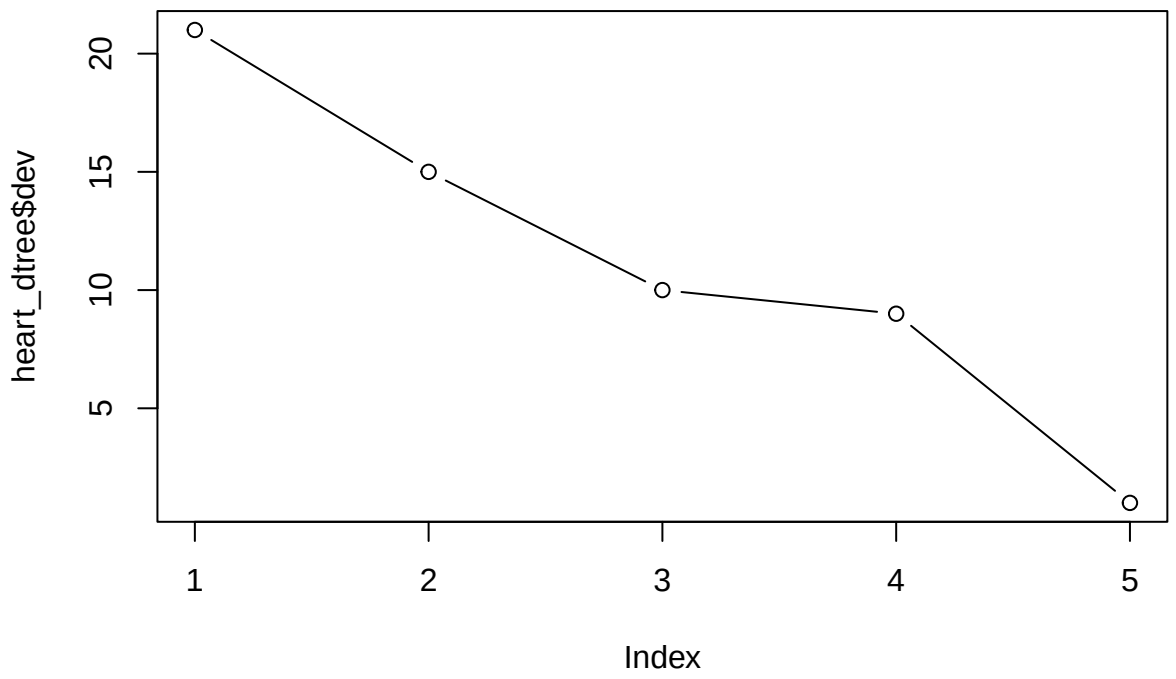
*The classification error rate is 33.33%*

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)
heart_dtree_cv <- cv.tree(heart_dtree, FUN = prune.misclass)
heart_dtree_cv
```

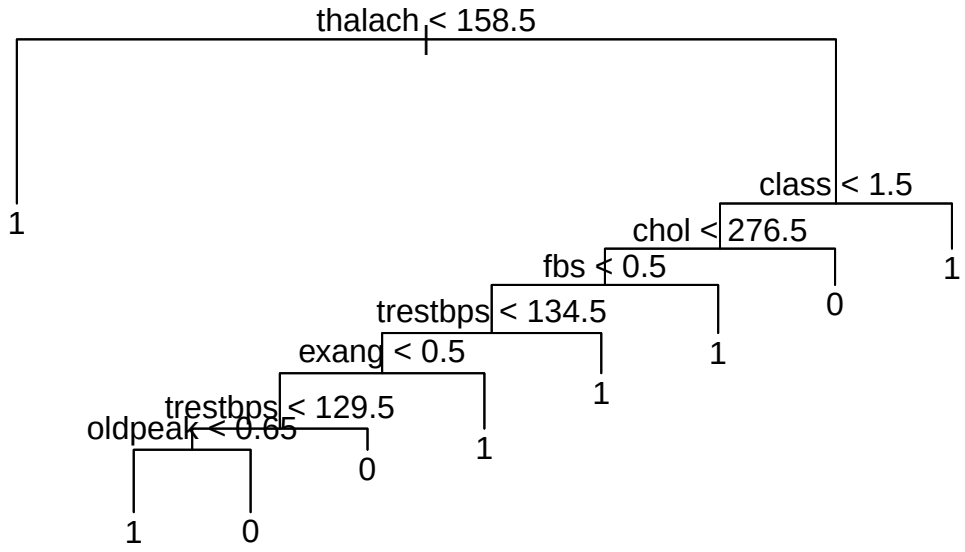
```
## $size
## [1] 21 15 10  9  1
##
## $dev
## [1] 74 73 73 75 68
##
## $k
## [1] -Inf 0.00 0.20 1.00 2.25
##
## $method
## [1] "misclass"
##
## attr(,"class")
## [1] "prune"         "tree.sequence"
```

```r
plot(heart_dtree_cv$size, heart_dtree$dev, type = "b")
```




```r
heart_dtree_pruned <- prune.misclass(heart_dtree, best = 5)

plot(heart_dtree_pruned)
text(heart_dtree_pruned, pretty = 0)
```




```
pruned_test_predictions <- predict(heart_dtree_pruned, heart[-heart_train,], type = "class")
pruned_confusion_matrix <- table(heart$binary[-heart_train], pruned_test_predictions)
pruned_confusion_matrix
```

```
##    pruned_test_predictions
##      0  1
##   0  1 16
##   1  9 31
```

*The misclassification rate is 43.86%*

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

*By pruning the tree, we reduce the tree's complexity and remove some branches. This helps reduce overfitting and improves the ability of the tree to generalize to new data. Also, it makes the tree less complex. With less branches, it is easy to interpret how the model makes classification deciesions. However, we do make a trade off in accuracy for this increase in interpretability. The classification rate after pruning is 43.86%, which is higher than the non-pruned classification rate of 33.33%.*

Discuss the ways a decision tree could manifest algorithmic bias.

- *A very bushy decision tree with many branches could overfit to noise in the training data. This may result in the model not generalizing well to new data. Thus, predictions can be biased if the model learns patterns that are specific to the training data, but do not exist in the general population.*
- *If training data is imbalanced, the model will not generalize well to populations not represented well in the training data. For example, if the dataset used to train the model lacks many observations from Black participants, the model may not accurately capture the race-specific determinants of cardiovascular disease that differ from white participants. Thus, predictions for Black participants from a model trained mostly on white participants may lead to predictions that are not accurate.*