# HW 3

Luca Buchoux

9/24/2024

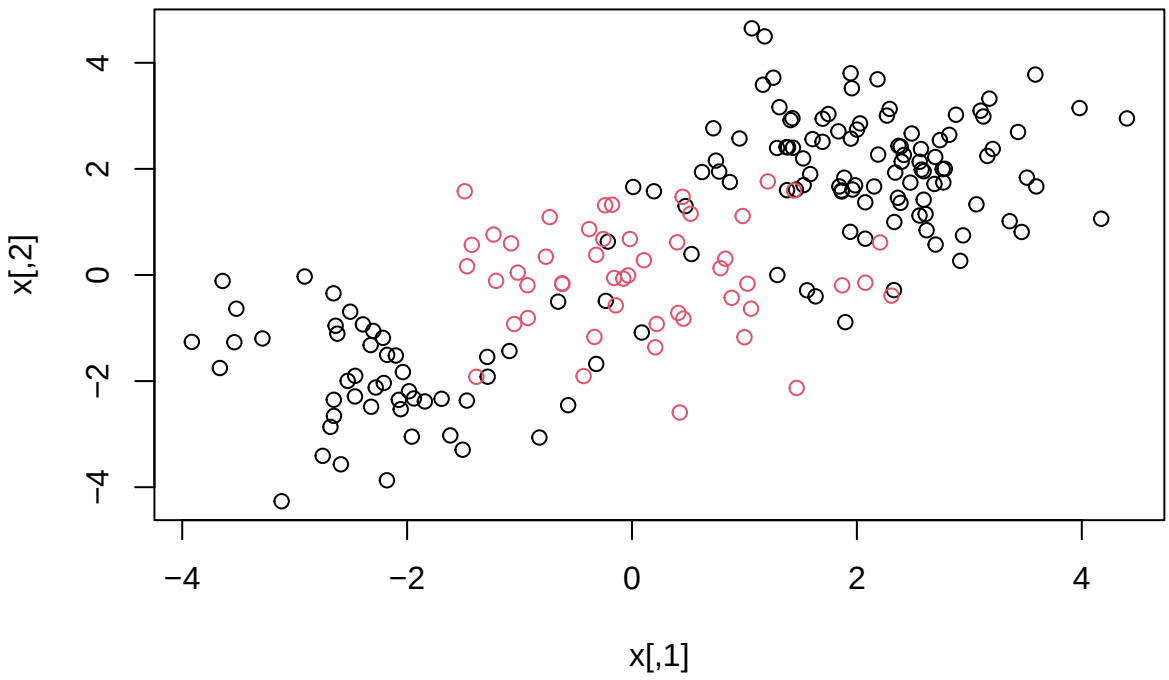
Let $E[X] = \mu$. Show that $Var[X] := E[(X - E[X])^2] = E[X^2] - (E[X])^2$. Note, all you have to do is show the second equality (the first is our definition from class).
$E[(X - E[X])^2] = E[X^2 - 2XE[X] + E[X]^2]$
$=> E[X^2] - E[2XE[X]] + E[E[X]^2]$
$=> E[X^2] - 2E[X]E[X] + E[X]^2$
$=> E[X^2] - E[X]^2$

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)
```

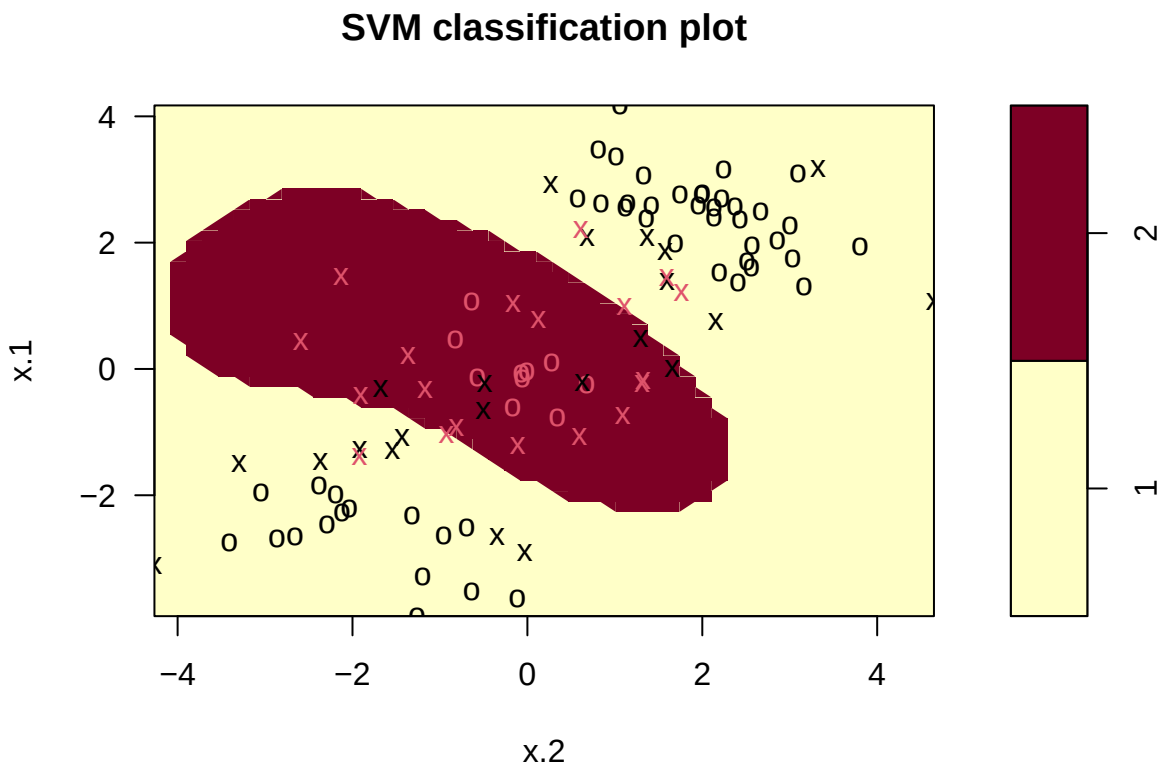
```
## Warning: package 'e1071' was built under R version 4.2.3
```

```
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.

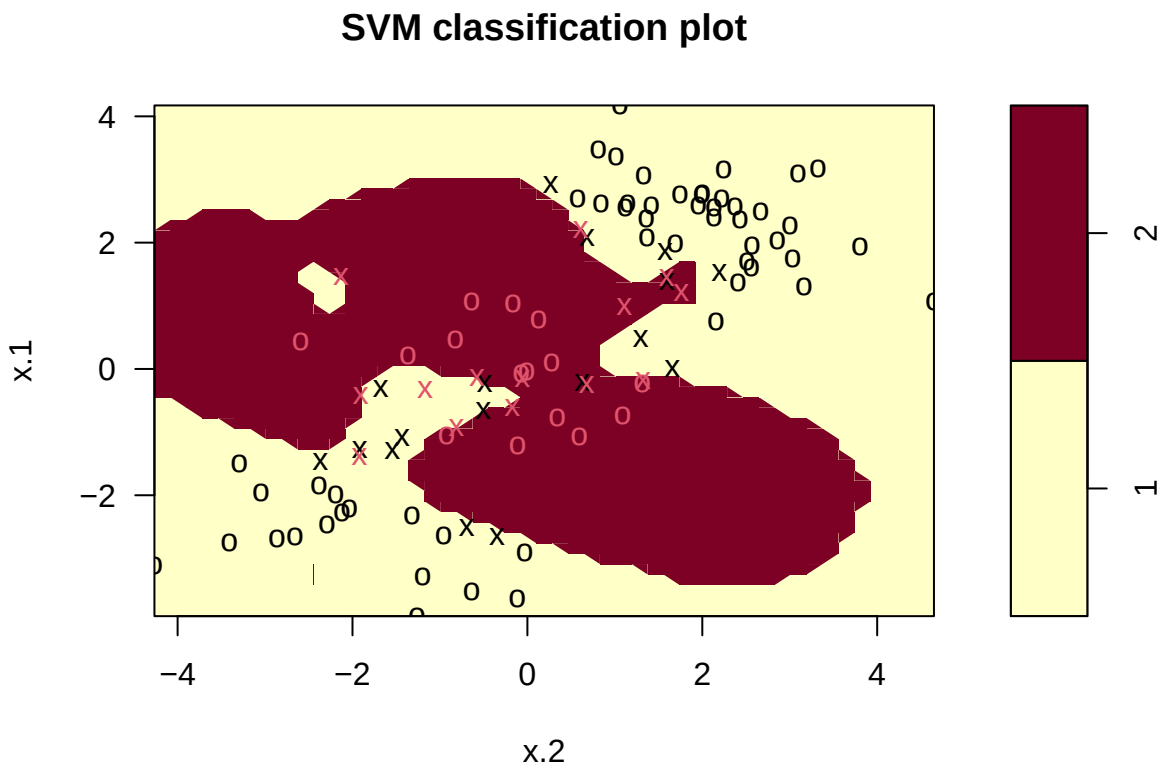
```
set.seed(1)
train<-sample(200,100)
svm_mod<-svm(y~.,data=dat[train,],kernel='radial',gamma=1,cost=1)
plot(svm_mod,dat[train,])
```




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.

```
svm_mod1<-svm(y~.,data=dat[train,],kernel='radial',gamma=1,cost=10000)
plot(svm_mod1,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.

*One issue could be that this svm is overfitting our training data, meaning that it works very good on the training set but would perform poorly on other sets of data because it is too specfically trained.*

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?

```
#remove eval = FALSE in above
table(true=dat[-train,"y"], pred=predict(svm_mod1, newdata=dat[-train,]))
```

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

```
sum(dat[train,'y']==2)/100
```

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

*Our training data has a proportion of 0.29 of class 2 which is pretty close to the underlying proportion, so the issue is less likely to be due to poor representation and more likely to be because our model was overfitted.*

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)))
summary(tune.out)
```

```
##
## Parameter tuning of 'svm':
##
## - sampling method: 10-fold cross validation
##
## - best parameters:
##  cost gamma
##     1   0.5
##
## - best performance: 0.12
##
## - Detailed performance results:
##     cost gamma error dispersion
## 1  1e-01   0.5  0.28 0.15491933
## 2  1e+00   0.5  0.12 0.07888106
## 3  1e+01   0.5  0.15 0.10801234
## 4  1e+02   0.5  0.17 0.11595018
## 5  1e+03   0.5  0.23 0.14944341
## 6  1e-01   1.0  0.25 0.13540064
## 7  1e+00   1.0  0.14 0.09660918
## 8  1e+01   1.0  0.16 0.10749677
## 9  1e+02   1.0  0.21 0.15238839
## 10 1e+03   1.0  0.20 0.14142136
## 11 1e-01   2.0  0.28 0.14757296
## 12 1e+00   2.0  0.15 0.10801234
## 13 1e+01   2.0  0.19 0.15238839
## 14 1e+02   2.0  0.18 0.14757296
## 15 1e+03   2.0  0.23 0.12516656
```

```
## 16 1e-01   3.0  0.28 0.15491933
## 17 1e+00   3.0  0.15 0.10801234
## 18 1e+01   3.0  0.20 0.16329932
## 19 1e+02   3.0  0.20 0.13333333
## 20 1e+03   3.0  0.27 0.11595018
## 21 1e-01   4.0  0.29 0.14491377
## 22 1e+00   4.0  0.16 0.09660918
## 23 1e+01   4.0  0.18 0.13984118
## 24 1e+02   4.0  0.21 0.11972190
## 25 1e+03   4.0  0.31 0.15951315
```

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.

*This model does appear to classify the data better as the number of misclassifications for both class 1 and 2 have decreased. However, just as before, we still seem to misclassify class 2 much more than class 1. But since we know the true proportion of class 2 is significantly lower than class 1, it makes sense that our model tends to predict class 1 more often.*

Let's turn now to decision trees.

```
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
```

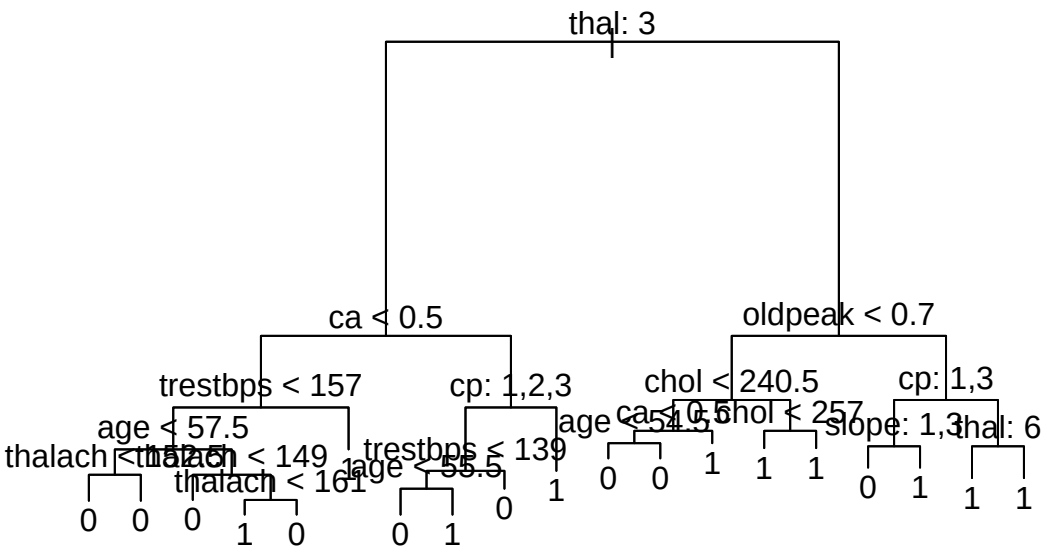
The response variable is currently a categorical variable with four levels. Convert heart disease into binary categorical variable. Then, ensure that it is properly stored as a factor.

```
heart[heart[,'class']!=0,'class']<-1
heart$class<-as.factor(heart$class)
levels(heart$class)
```

```
## [1] "0" "1"
```

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

```
set.seed(101)
train1<-sample(297,240)
tree_mod<-tree(class~.,data=heart[train1,])
plot(tree_mod)
text(tree_mod,pretty=0)
```




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

```
table(true=heart[-train1,'class'],
      pred=predict(tree_mod,newdata=heart[-train1,],type='class'))
```

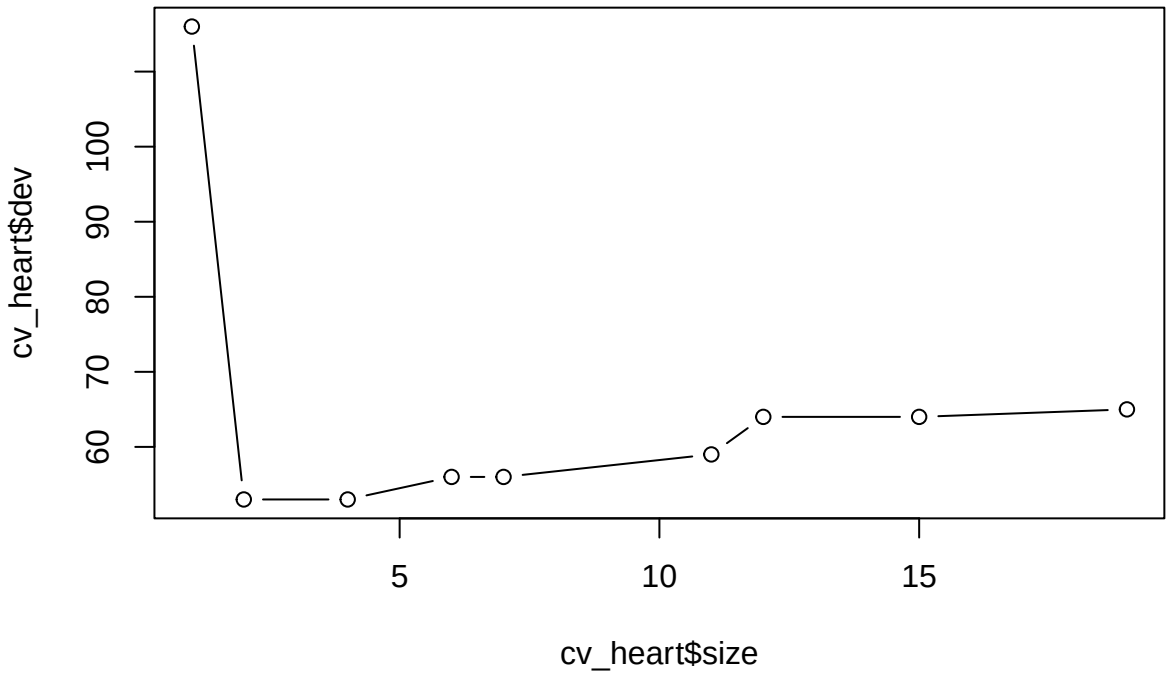
```
##     pred
## true  0  1
##    0 28  8
##    1  3 18
```

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

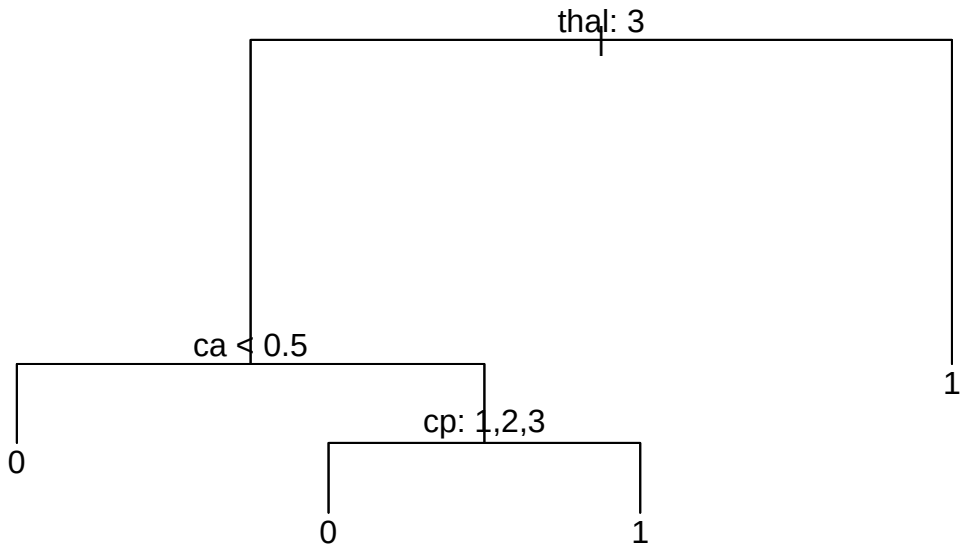
```
## [1] 0.1929825
```

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.

```
set.seed(101)
cv_heart<-cv.tree(tree_mod,FUN=prune.misclass)
plot(cv_heart$size,cv_heart$dev,type='b')
```



```
prune_mod<-prune.misclass(tree_mod,best=3)
plot(prune_mod)
text(prune_mod,pretty=0)
```




```
table(true=heart[-train1,'class'],
      pred=predict(prune_mod,newdata=heart[-train1,],type='class'))
```

```
##     pred
## true  0  1
##    0 26 10
##    1  4 17
```

```
(10+4)/(26+10+4+17)
```

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

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

*We can see that our pruned tree has a slightly higher misclassification rate, but it is only higher by about 5%. Despite this, by comparing the plots of each tree we can see that our pruned tree is much easier to read and interpret than the full tree, which is near impossible to read at the terminal nodes.*

Discuss the ways a decision tree could manifest algorithmic bias.

*Having very strict requirements for purity in a decision tree can easily lead to overfitting, making the model very biased on what it was trained on if we do not cross vailidate.*