HW 3

Sam Pell

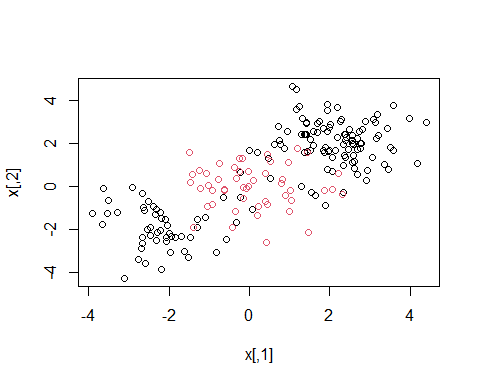
11/27/2023

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)

## Warning: package 'e1071' was built under R version 4.3.2

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)

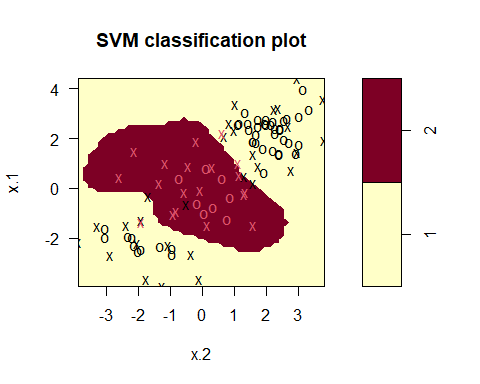


Quite clearly, the above data is not linearly separable. Create a training-testing partition with 100 random observations in the training partition. Fit an svm on this training data using the radial kernel, and tuning parameters , cost . Plot the svm on the training data.

random\_subset <- sample(1:nrow(dat), 100)  
dat\_train <- dat[random\_subset,]  
dat\_test <- dat[-random\_subset,]  
  
svmfit1 = svm(y ~ ., data = dat\_train, kernel = "radial", cost = 1, gamma = 1, scale = FALSE)  
print(svmfit1)

##   
## Call:  
## svm(formula = y ~ ., data = dat\_train, kernel = "radial", cost = 1,   
## gamma = 1, scale = FALSE)  
##   
##   
## Parameters:  
## SVM-Type: C-classification   
## SVM-Kernel: radial   
## cost: 1   
##   
## Number of Support Vectors: 48

plot(svmfit1, dat\_train)

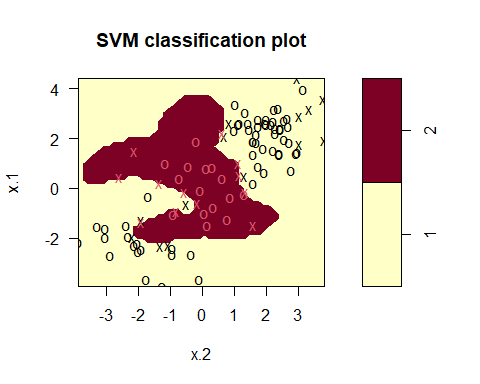


Notice that the above decision boundary is decidedly non-linear. It seems to perform reasonably well, but there are indeed some misclassifications. Let’s see if increasing the cost [[1]](#footnote-27) helps our classification error rate. Refit the svm with the radial kernel, , and a cost of 10000. Plot this svm on the training data.

svmfit2 = svm(y ~ ., data = dat\_train, kernel = "radial", cost = 10000, gamma = 1, scale = FALSE)  
print(svmfit2)

##   
## Call:  
## svm(formula = y ~ ., data = dat\_train, kernel = "radial", cost = 10000,   
## gamma = 1, scale = FALSE)  
##   
##   
## Parameters:  
## SVM-Type: C-classification   
## SVM-Kernel: radial   
## cost: 10000   
##   
## Number of Support Vectors: 26

plot(svmfit2, dat\_train)



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

*Student Answer*

The clear danger of increasing the cost, especially by so significant a margin, is overfitting of the svm. This can easily lead to a situation where the svm is optimally adapted to the training dataset, but is so adapted that the differences between the training and test datasets are too great, and the model no longer models the testing data well.

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  
tab <- table(true=dat[-random\_subset,"y"], pred=predict(svmfit2, newdata=dat[-random\_subset,]))  
tab

## pred  
## true 1 2  
## 1 59 18  
## 2 5 18

accuracy <- function(x){  
 sum(diag(x))/sum(rowSums(x))\*100  
}  
  
accuracy(tab)

## [1] 77

The confusion matrix, as well as the accuracy rate, is not perfectly reflective of the data. Furthermore, the svm is misclassifying datapoints of class 2 at a significantly higher rate than those of class 1, which shows concerning disparity.

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(dplyr)

##   
## Attaching package: 'dplyr'

## The following objects are masked from 'package:stats':  
##   
## filter, lag

## The following objects are masked from 'package:base':  
##   
## intersect, setdiff, setequal, union

nrow(dat\_train[dat\_train$y == 2, ]) / nrow(dat\_train)

## [1] 0.27

nrow(dat[dat$y == 2, ]) / nrow(dat)

## [1] 0.25

*Student Response* It does appear that the proportion of class 2 in the training dataset is representative of the underlying number, which leads me to believe that representativeness is not the issue here. Perhaps the test dataset is the unrepresentative one?

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 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, 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[-random\_subset,"y"], pred=predict(tune.out$best.model, newdata=dat[-random\_subset,]))

## pred  
## true 1 2  
## 1 64 13  
## 2 2 21

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.

*Student Response*

This confusion matrix is improved, especially with its bias against predicting class 2, where it now has a 21-13 correct-missed split, compared to the 18-18 seen prior. The improved model still needs to maintain proper representation of the full dataset by the test and training dataset, and the training and testing datasets must match up well and be representative of each other as well.

Let’s turn now to decision trees.

library(kmed)

## Warning: package 'kmed' was built under R version 4.3.2

data(heart)  
library(tree)

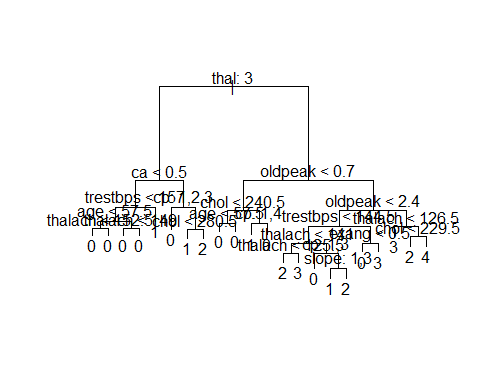
## Warning: package 'tree' was built under R version 4.3.2

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$class <- as.factor(heart$class)

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~.-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.

tree.pred = predict(tree.heart, heart[-train,], type="class")  
tab2 <- with(heart[-train,], table(tree.pred, class))  
tab2

## class  
## tree.pred 0 1 2 3 4  
## 0 33 2 3 1 1  
## 1 1 2 0 1 1  
## 2 1 0 0 2 1  
## 3 0 2 1 0 0  
## 4 1 1 2 1 0

accuracy(tab2)

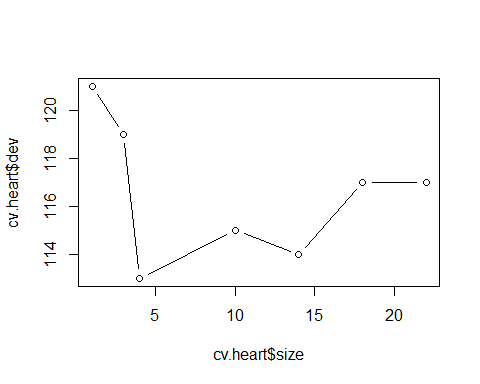
## [1] 61.40351

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.

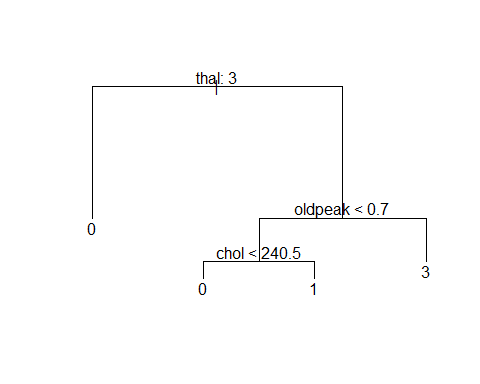
set.seed(814)  
cv.heart <- cv.tree(tree.heart, FUN = prune.misclass)  
cv.heart

## $size  
## [1] 22 18 14 10 4 3 1  
##   
## $dev  
## [1] 117 117 114 115 113 119 121  
##   
## $k  
## [1] -Inf 0.000000 1.000000 2.000000 2.833333 4.000000 7.500000  
##   
## $method  
## [1] "misclass"  
##   
## attr(,"class")  
## [1] "prune" "tree.sequence"

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



prune.heart <- prune.misclass(tree.heart, best = 4)  
  
plot(prune.heart)  
text(prune.heart, pretty=0)



tree.pred = predict(prune.heart, heart[-train,], type="class")  
tab3 <- with(heart[-train,], table(tree.pred, class))  
tab3

## class  
## tree.pred 0 1 2 3 4  
## 0 31 3 3 3 1  
## 1 1 1 0 0 0  
## 2 0 0 0 0 0  
## 3 4 3 3 2 2  
## 4 0 0 0 0 0

accuracy(tab3)

## [1] 59.64912

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

*Student Input*

While the tree above did lose some accuracy (going from 61.4% to 59.6% accuracy), it is far more interpretable, having just 4 outcomes. This results in a tree which retains much of the accuracy with very insignificant losses. This often follows with other pruning measures, and in general, it is good practice to attempt to prune every tree.

Discuss the ways a decision tree could manifest algorithmic bias.

*Student Answer*

A decision tree could show algorithmic bias with misrepresentative data, especially with poor training-testing splits. This would manifest in the tree by having one group be predicted at much higher rates than it actually occurs, simply because the training dataset does not accurately show what the classes should be. This could be seen in a dataset that may have used the first n rows, rather than a random sample of n rows, which might lead the tree to predict based on a different characteristic, and, while it may still be improved with pruning, this could lead to an overly simplistic tree which misclassifies at extremely high rates.

1. Remember this is a parameter that decides how smooth your decision boundary should be [↑](#footnote-ref-27)