HW 3

Nikita McClure

10/03/2024

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).

$E[(X - E[X])^2] = E[X^2] - (E[X])^2$

$E[(X - E[X])^2]$

$E[(X - E[X])(X - E[X])]$

$E[X^2 - (2 * XE[X]) + (E[X])^2]$

$E[X^2] - E[(2 * XE[X])] + E[(E[X])^2]$

$E[X^2] - (2 * (E[X])^2) + (E[X])^2$

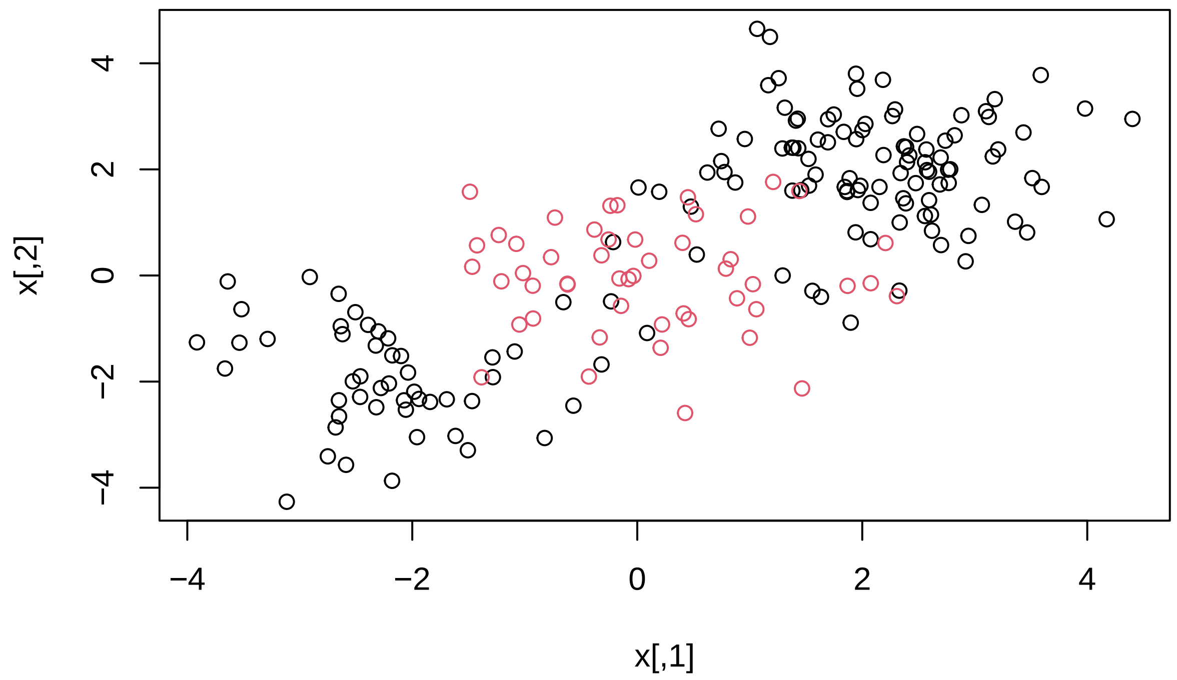
$E[X^2] - (E[X])^2$

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.

```r
library(e1071)
```

```
## Warning: package 'e1071' was built under R version 4.3.3
```

```r
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.

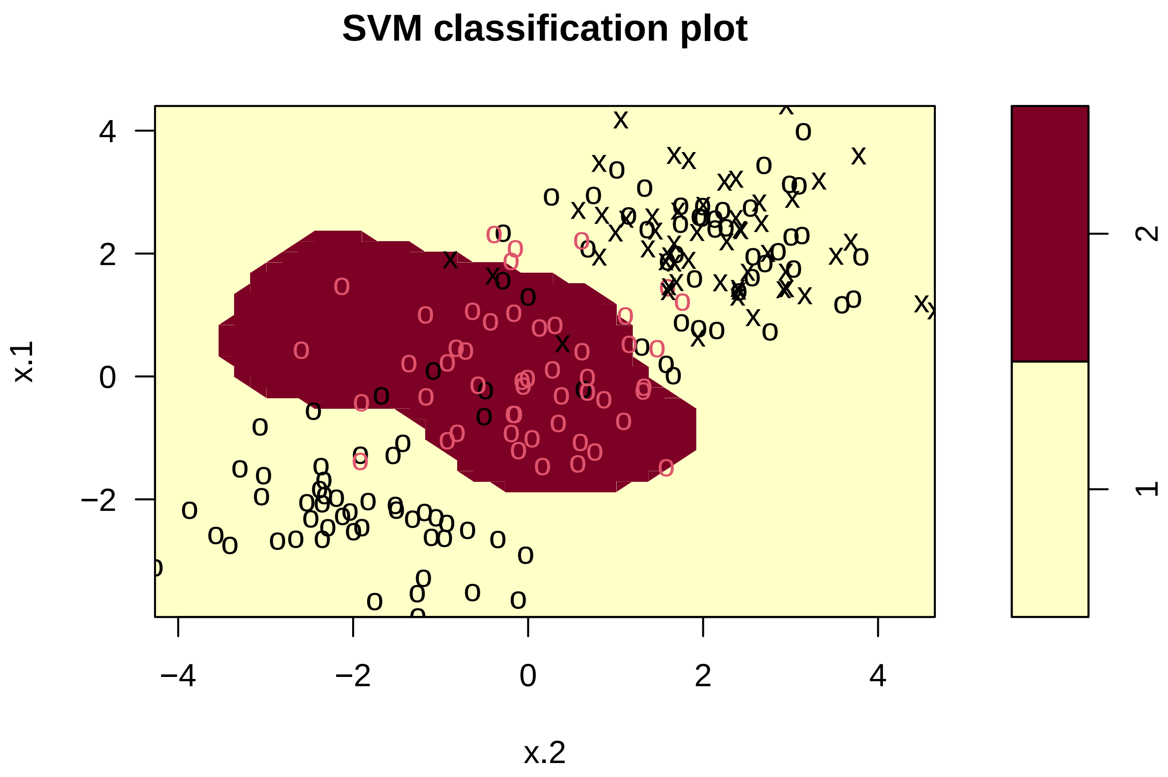
```
##in progress rn - 10/3
set.seed(1)

train = sample(1:nrow(dat), 100)
train_dat = dat[train, ]

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

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

```
plot(svmfit, dat)
```



```
test_dat = 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.

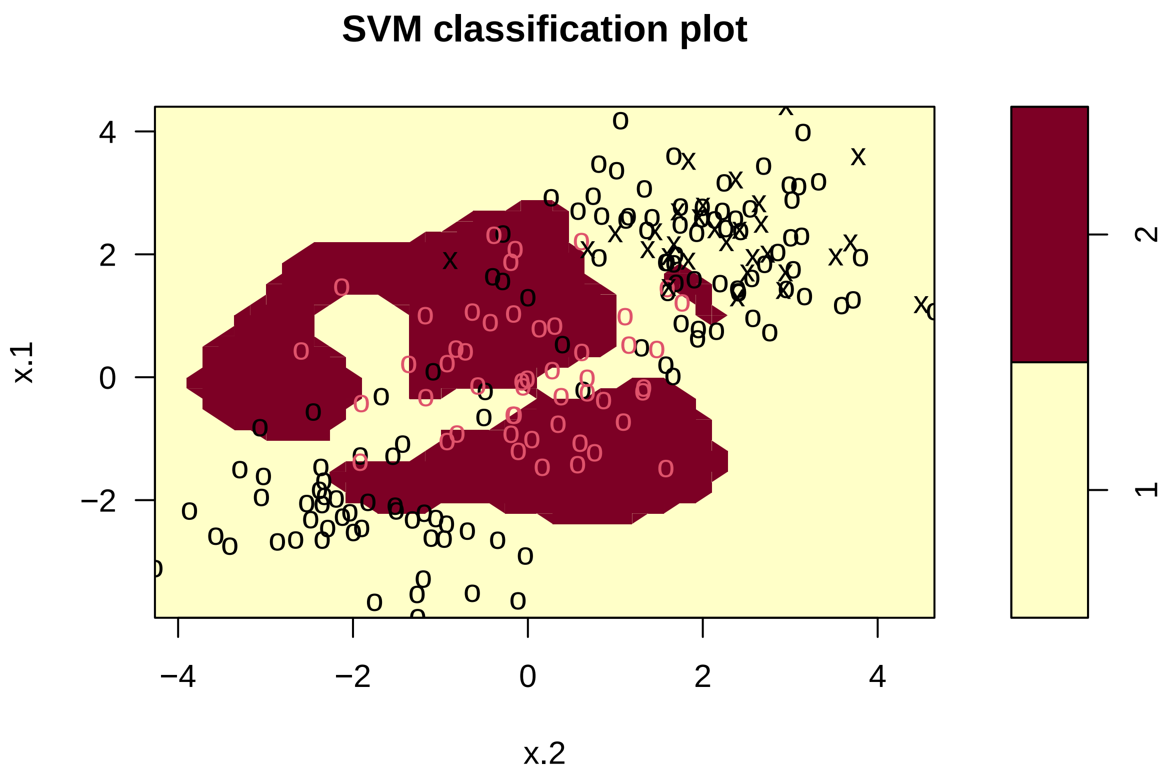
```
set.seed(1)
svmfit = svm(y~., data=train_dat, kernel = "radial", cost = 10000, gamma=1, scale = FALSE)
print(svmfit)
```

```
##
## Call:
## svm(formula = y ~ ., data = train_dat, kernel = "radial", cost = 10000,
##     gamma = 1, scale = FALSE)
```

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

```
##
##
## Parameters:
##    SVM-Type:  C-classification
##  SVM-Kernel:  radial
##        cost:  10000
##
## Number of Support Vectors:  30
```

```r
plot(svmfit, dat)
```



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

((class - “overfitting”- the cost/kernel arguments))

Student Answer

The high cost, 10,000, can lead to overfitting. The classification will be set super strictly to the training data so if the testing does not follow the training data very very closely then the classifications will not be super accurate.

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?

```r
table(true=dat[-train,"y"], pred=predict(svmfit, newdata=dat[-train,]))
```

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

```r
correctness <- (62+18)/(62+18+20)
print(correctness)
```

```
## [1] 0.8
```

Answer

Our classification results have an accuracy of 80%, there were 17 ones predicted as twos and 3 twos predicted as one, meaning it is overfitting twos (which is discussed in the next answer :).*

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.

```r
summary(train_dat$y == 2)
```

```
##    Mode   FALSE    TRUE 
## logical      71      29
```

```r
twos <- 29/(71+29)
print(twos)
```

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

Student Answer

The training partition has 29% of the data as class 2. This is not fully representative of the 25% partition.

A 4% disparity is not huge for a training set so small; it is not surprising that the data is not perfectly representative of the entire data set. However, this disparity, though small, can negatively influence the algorithm.

The training set having a larger proportion of twos than the entire data set can lead to the model predicting more variables as 2 than it should because it's expecting 29% instead of 25% of the data to be 2.

This overfitting is apparent in our classification results of the testing data above.

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`.

```r
set.seed(1)
tune.out <- tune(svm, y ~., data = train_dat, kernel = "radial", ranges = list(cost = c(0.1, 1, 10, 100
```

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.

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

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.

Answer The misclassification rate went down to 8%, a substantial improvement from the 20% it was before. There is still the over classification of twos (ones being classified as twos) so that should still be addressed, maybe by trying more or different parameters.

Let's turn now to decision trees.

```r
library(kmed)
```

```
## Warning: package 'kmed' was built under R version 4.3.3
```

```r
data(heart)
library(tree)
```

```
## Warning: package 'tree' was built under R version 4.3.3
```

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.

```r
# View(heart)
##class is the four category variable, I assume 0 & 1 means no disease and 2 & 3 means yes

disease = ifelse(heart$class <= 1, "No", "Yes")
heart = data.frame(heart, disease)

diseasefac <- as.factor(disease)
heart <- data.frame(heart, diseasefac)
names(heart)
```

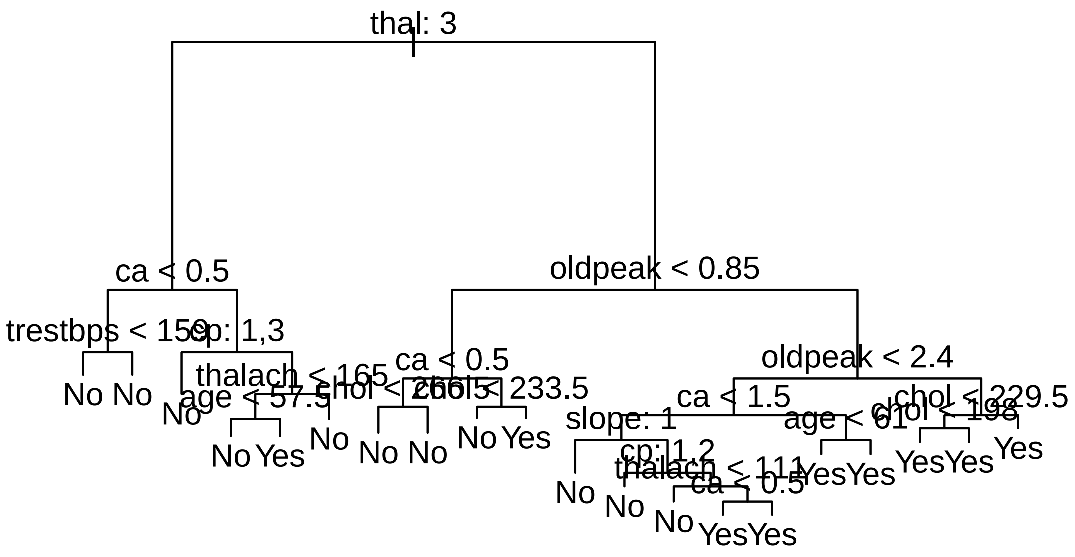
```
##  [1] "age"         "sex"         "cp"          "trestbps"    "chol"
##  [6] "fbs"         "restecg"     "thalach"     "exang"       "oldpeak"
## [11] "slope"       "ca"          "thal"        "class"       "disease"
## [16] "diseasefac"
```

```r
heart <- heart[, -15]
# str(heart)
```

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

```r
set.seed(101)
train=sample(1:nrow(heart), 240)

tree.heart = tree(diseasefac ~. -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.

```r
tree.pred = predict(tree.heart, heart[-train,], type="class")
with(heart[-train,], table(tree.pred, diseasefac))
```

```
##          diseasefac
## tree.pred No Yes
##       No  34   5
##       Yes  9   9
```

```r
error = (5+9)/(34+5+9+9)
print(error)
```

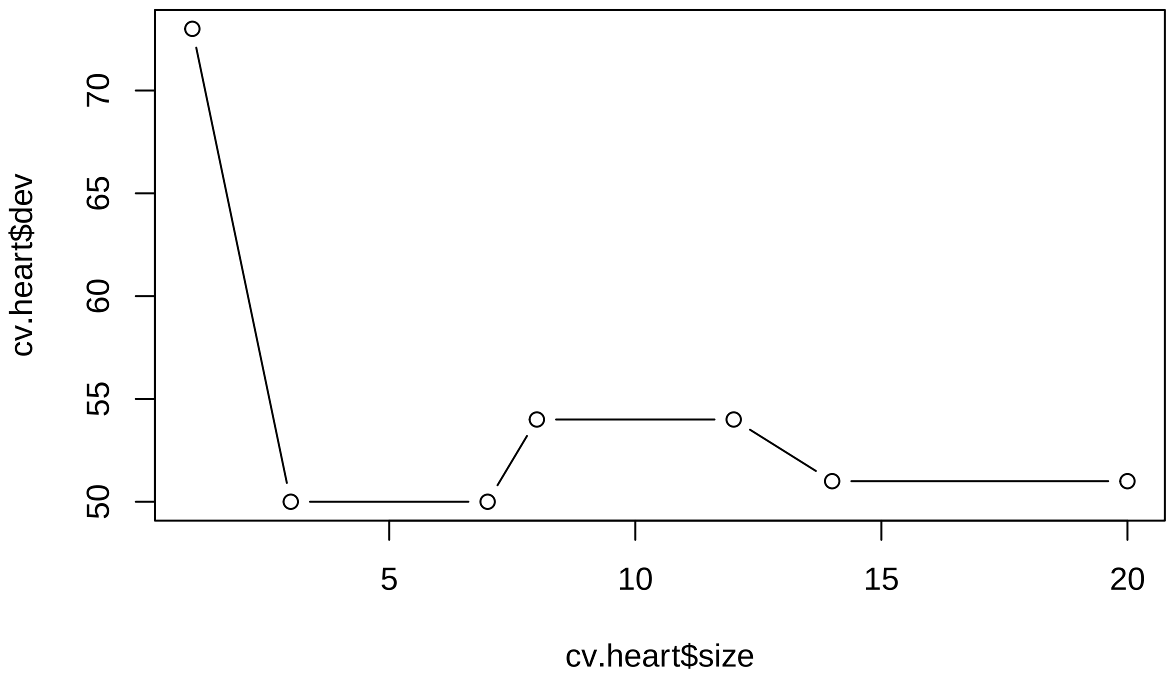
```
## [1] 0.245614
```

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.

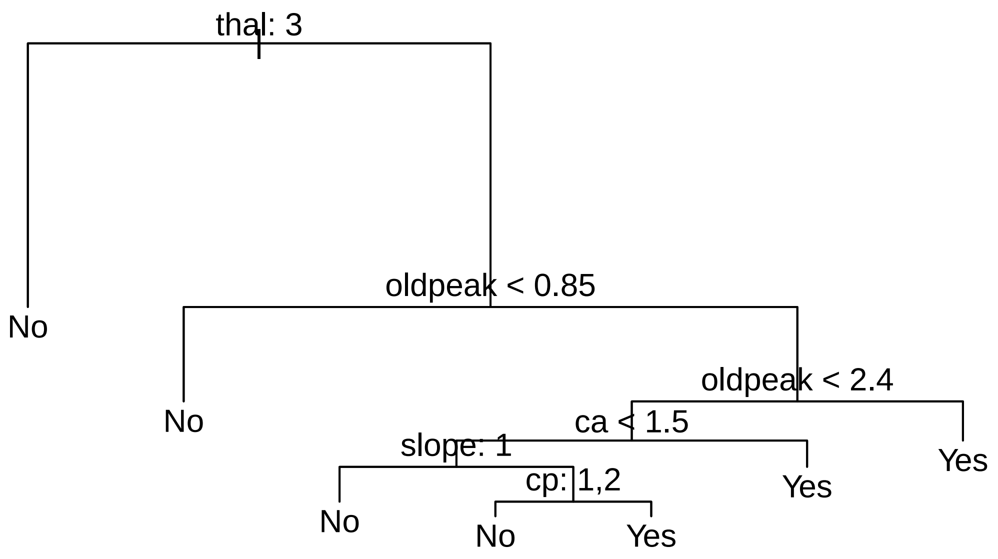
```r
set.seed(101)
cv.heart = cv.tree(tree.heart, FUN = prune.misclass)
cv.heart
```

```
## $size
## [1] 20 14 12  8  7  3  1
##
## $dev
## [1] 51 51 54 54 50 50 73
##
## $k
## [1]  -Inf  0.00  1.00  1.25  2.00  2.25 14.50
##
## $method
## [1] "misclass"
##
## attr(,"class")
## [1] "prune"         "tree.sequence"
```

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



```r
prune.heart <- prune.misclass(tree.heart, best =5)
plot(prune.heart)
text(prune.heart, pretty=0)
```



```r
tree.pred = predict(prune.heart, heart[-train,], type="class")
with(heart[-train,], table(tree.pred, diseasefac))
```

```
##          diseasefac
## tree.pred No Yes
##       No  38   7
##       Yes  5   7
```

```r
misclass <- ((5+7)/(38+7+5+7))
print(misclass)
```

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

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

Answer

The error went down, but not significantly, meaning it could potentially be pruned more. Pruning the tree makes it ignore more specific/less informative branches, these branches may lead to overfitting, so pruning helps reduce this risk. Pruning also makes the tree easier to interpret due to it having less branches.

Either way, the correctness is well above 50% so it is a fine predictor for binary classification with or without the pruning.

Discuss the ways a decision tree could manifest algorithmic bias.

Answer

If a tree is too “bushy”, or not pruned down well, then the decision algorithm will be over-fit to the testing data. This is because some of the branches are too specific so the algorithm is fit to expect these specific patterns, when the testing data does not follow the training data very strictly it will not be able to accurately predict the testing data.

Alternatively, if the tree is pruned down too far it may provide insufficient data and the algorithm may be underfit.

Additionally, as with all decision algorithms, if the testing-training partition is not representative, the algorithm will be inaccurate.

Bias can also be added by the nodes/featured that are kept. If there is bias in these features the the tree might reinforce it, and vise-versa with excluding it.