HW 3

Anna Fetter

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).

step 1:expand.

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

step 2: distribute E.

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

step 3: simplify terms.

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

step 4: subtract

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

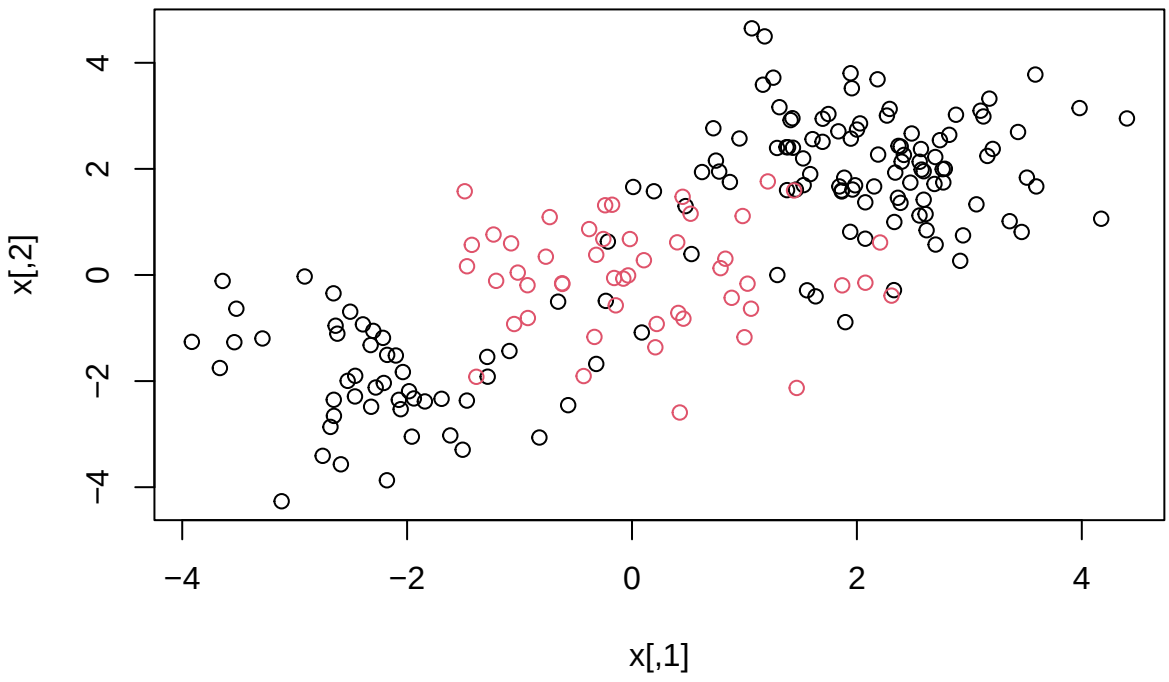
therefore, $Var[X] := E[(X - E[X])^2] = E[X^2] - (E[X])^2$

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 4.3.3
```

```
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)
```



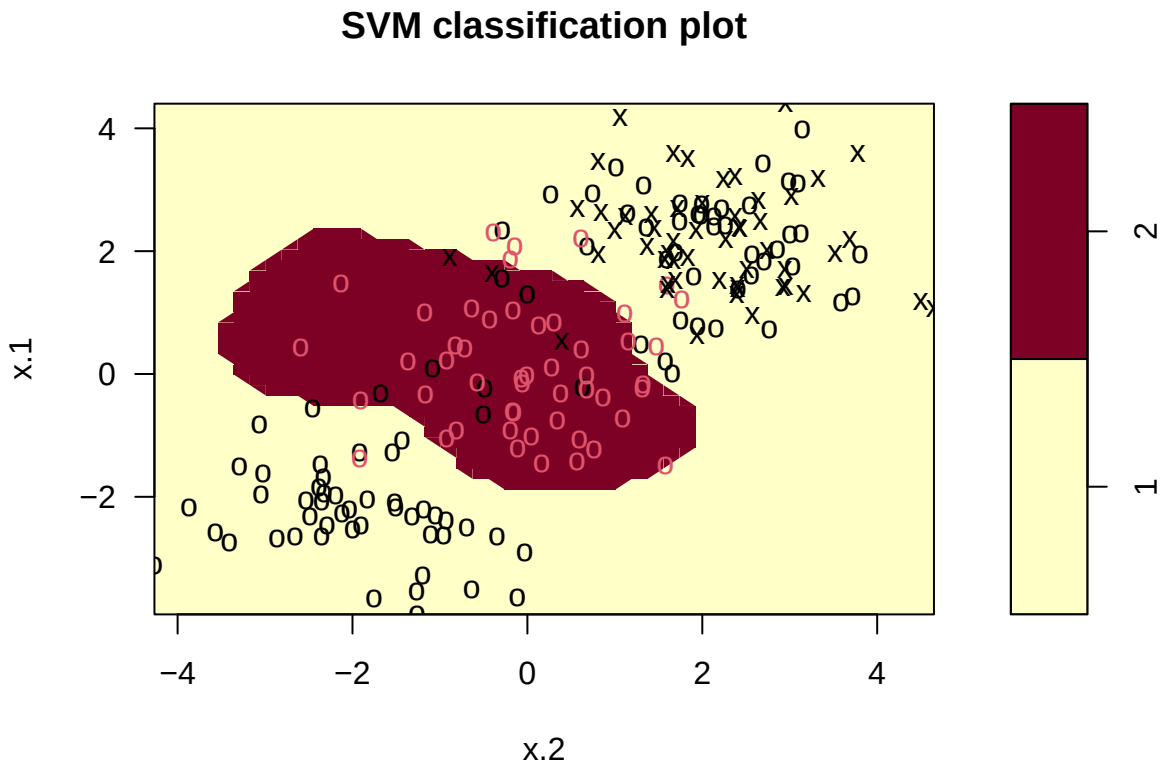
Quite clearly, the above data is not linearly separable. Create a training-testing partition with 100 random observations in the training partition. Fit an svm on this training data using the radial kernel, and tuning parameters $\gamma = 1$, cost $= 1$. Plot the svm on the training data.

```
set.seed(1)

#create random indexes for sample data
train <- sample(1:nrow(dat), 100)
#ask Andy if this needs to be 100 to get 100 random observations

#partition data into training and testing
train_data <- dat[train,]
test_data <- dat[-train,]

#fitting an SVM
svmfit = svm(y~., data = train_data, kernel = "radial", cost = 1, gamma =1, scale = FALSE)
print(svmfit)
```

```
##
## Call:
## svm(formula = y ~ ., data = train_data, 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)
```



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.

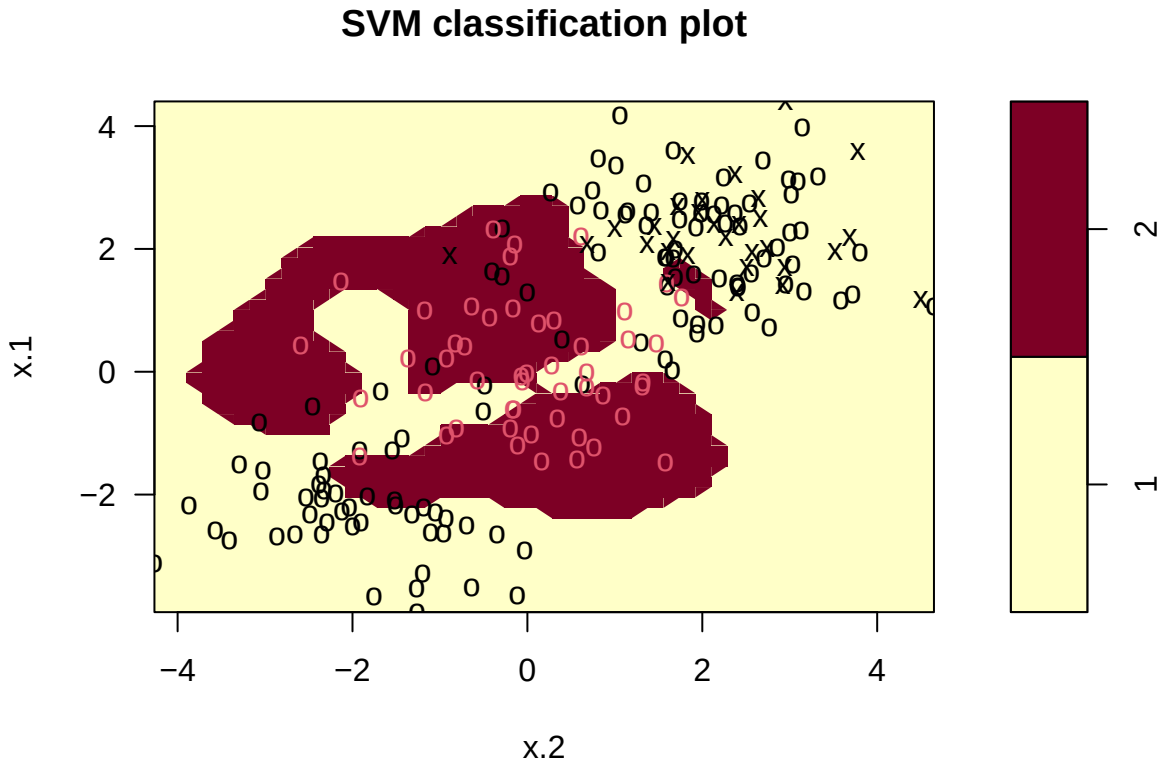
```
svmfit = svm(y~., data = train_data, kernel = "radial", cost = 10000, gamma =1, scale = FALSE)
print(svmfit)
```

```
##
## Call:
```

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

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

```
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.

Student Answer

The danger of increasing the cost of the svm fit is that there may be overfitting of the data. When data is overfitted, then the model may not work as well on new data that isn't the training set. The classifications will not be super accurate because the classification will be too close to the training data and not generalize 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
table(true=dat[-train,"y"], pred=predict(svmfit, newdata=dat[-train,]))
```

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

```
percent_correct = (62+18)/(62+17+3+18)
print(percent_correct)
```

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

The confusion matrix reveals that there is an 80% accuracy rate. However, the confusion matrix reveals that our model over predicts 2s when the true value is 1. It's much more rare that the model predicts 1 when the true value is 2. This could be because there are less class 2s in the data in general.

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 of y table of train_data to find number of class 2
#summary(train_data$y)
train_class_counts <- table(train_data$y)

# Calculate proportions
train_class_props <- prop.table(train_class_counts)

# Output the proportions
train_class_props
```

```
##
##    1    2
## 0.71 0.29
```

```
print(train_class_props[2])
```

```
##    2
## 0.29
```

```
#Class 2 is .29 of the training data
```

Student Response

.29 (the training partition proportion of 2s) is fairly close to .25 (whole data proportion of 2s) as a proportion for a data set of this size. However, within a bigger data set this .04 set difference between the training data and whole data could become a bigger concern. For a data set this size, the random sampling performed okay. Since .29 is greater than .25, the training model may over predict the proportion of 2s in the data. This over-fitting is apparent in our classification results in the 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`.

```
set.seed(1)
tune.out <- tune(svm, y~., data = train_data, 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.

```
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.

Student Response

The confusion matrix shows far fewer errors (8% error rate) compared to our previous calculated error (20%). So by tuning the model we are getting a better error rate. To improve this model further, we should use a larger training set to get closer to the population proportions. We should also do a better job of managing the class imbalance between 1 and 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.

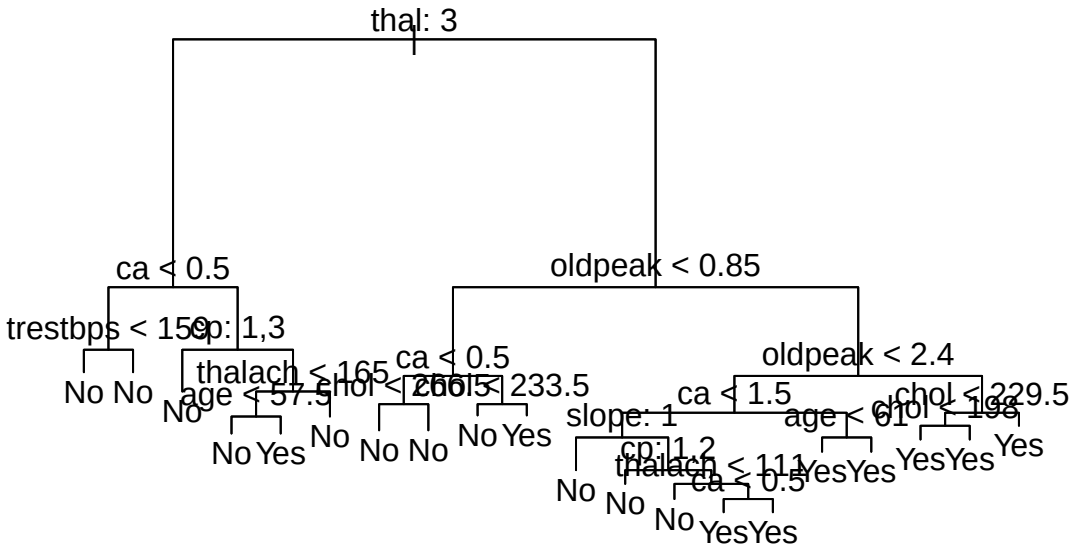
```
heart_disease = ifelse(heart$class <= 1, "No", "Yes")
heart = data.frame(heart, heart_disease)
heart_disease_fac <- as.factor(heart_disease)
heart <- data.frame(heart, heart_disease_fac)
#assuming class is the variable to say if someone has heart disease. Based off the AHA website, there a
names(heart)
```

```
##  [1] "age"               "sex"               "cp"
##  [4] "trestbps"          "chol"              "fbs"
##  [7] "restecg"           "thalach"           "exang"
## [10] "oldpeak"           "slope"             "ca"
## [13] "thal"              "class"             "heart_disease"
## [16] "heart_disease_fac"
```

```
heart <- heart[, -15]
```

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_HD=sample(1:nrow(heart), 240)

tree.heart = tree(heart_disease_fac~. -class, heart, subset = train_HD)
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_HD,], type="class")
#predict command used for trained data, data to predict, type since the type can be regression or class
with(heart[-train_HD,], table(tree.pred, heart_disease_fac))
```

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

```
error <- (5+9)/(34+5+9+9)
print(error)
```

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

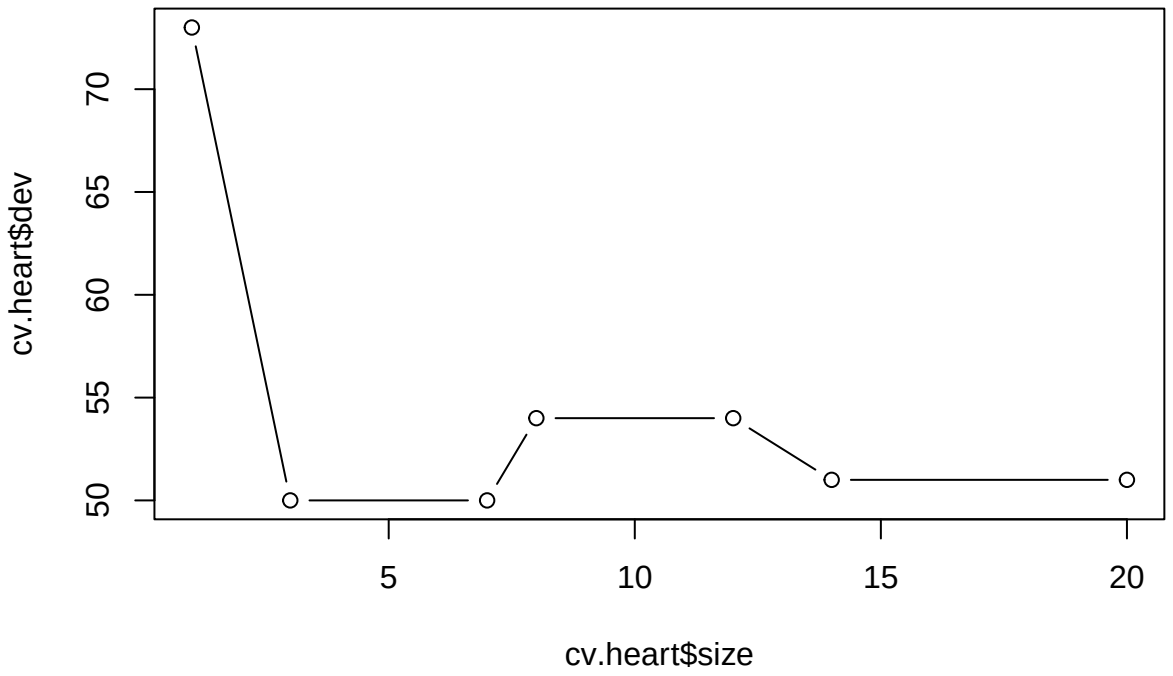
Above we have a fully grown (bushy) tree. Now, cross validate it using the `cv.tree` command. Specify cross validation to be done according to the misclassification rate. Choose an ideal number of splits, and

plot this tree. Finally, use this pruned tree to test on the testing set. Report a confusion matrix and the misclassification rate.

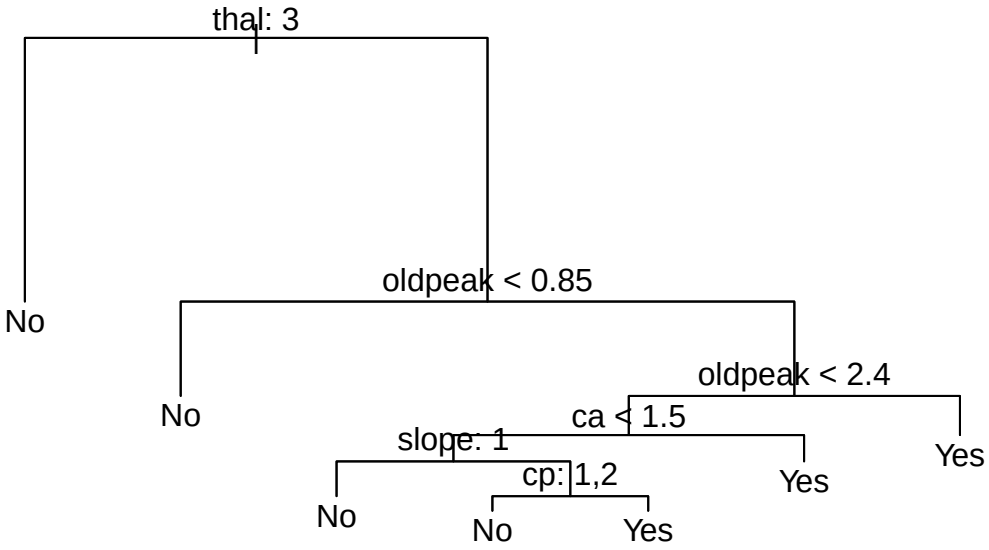
```
set.seed(101)

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"
```

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



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




```
tree.pred = predict(prune.heart, heart[-train_HD,], type="class")
with(heart[-train_HD,], table(tree.pred, heart_disease_fac))
```

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

```
prune_tree_error <- (7+5)/(38+7+5+7)
print(prune_tree_error)
```

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

Prune tree error = 0.2105263

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

Student Input

The error went down about 4% when we pruned the tree. The predictability is still above the 50% which would occur if we just flipped coins to classify. Pruning a tree helps increase the interpretability and readability of the tree. Not pruning a tree can lead to overfitting, where the model is too closely aligned to the training data and won't predict well for the data as a whole. Over-pruning the tree can also lead to some sacrifices in the predictive performance because there may not be perfectly pure nodes.

Discuss the ways a decision tree could manifest algorithmic bias.

Student Answer

Overfitting to the training data (too bushy of a tree) could manifest into algorithmic bias because it predicts too closely to the training data which may or may not be fully representative of the whole data set or the testing data. This could especially be a problem if the data is skewed in the training data.