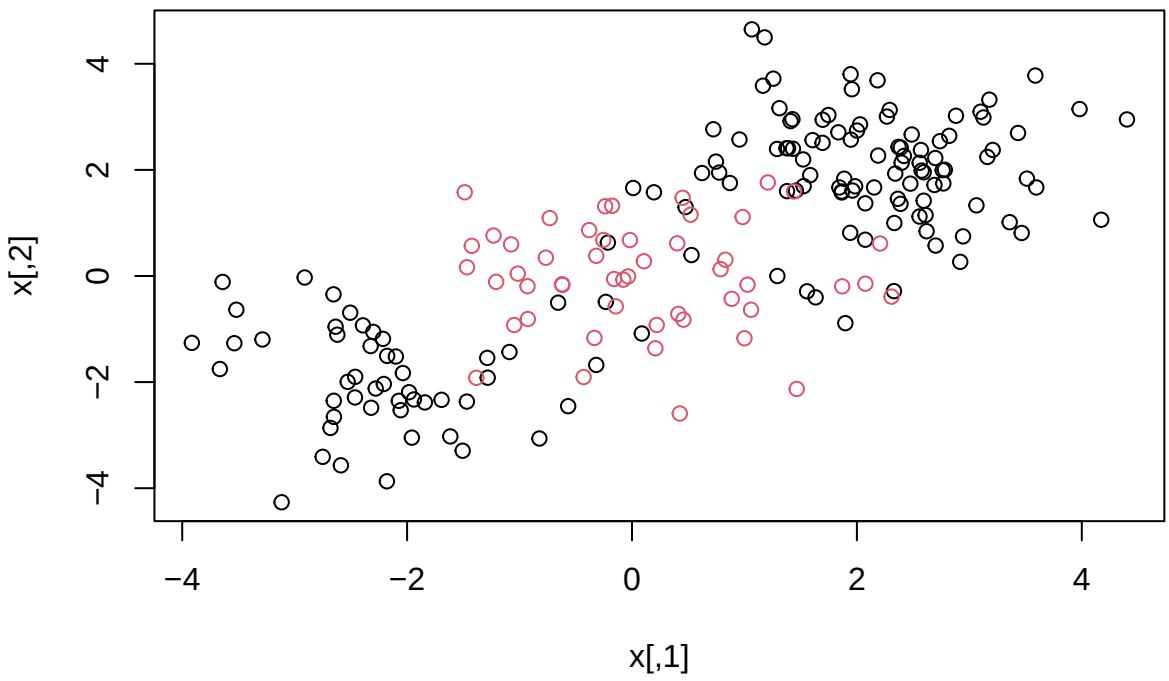
# HW 3

Lalida Kungval

11/28/2023

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)
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)
```



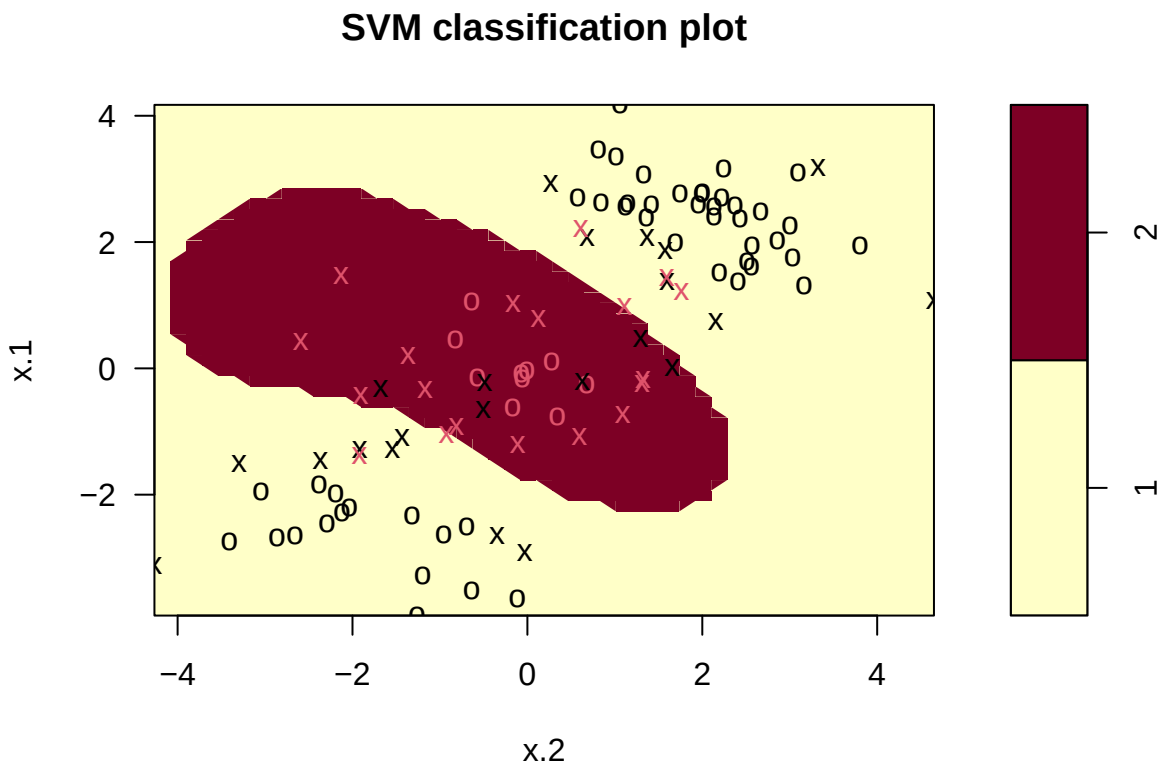
Quite clearly, the above data is not linearly separable. Create a training-testing partition with 100 random observations in the training partition. Fit an svm on this training data using the radial kernel, and tuning parameters $\gamma = 1$, cost $= 1$. Plot the svm on the training data.

```
set.seed(1)

total_rows <- nrow(dat)
train <- sample(total_rows, 100)
train_data <- dat[train, ]
test_data <- dat[-train, ]

svm_model <- svm(y ~ ., data=train_data, kernel="radial", gamma=1, cost=1)

plot(svm_model, train_data)
```

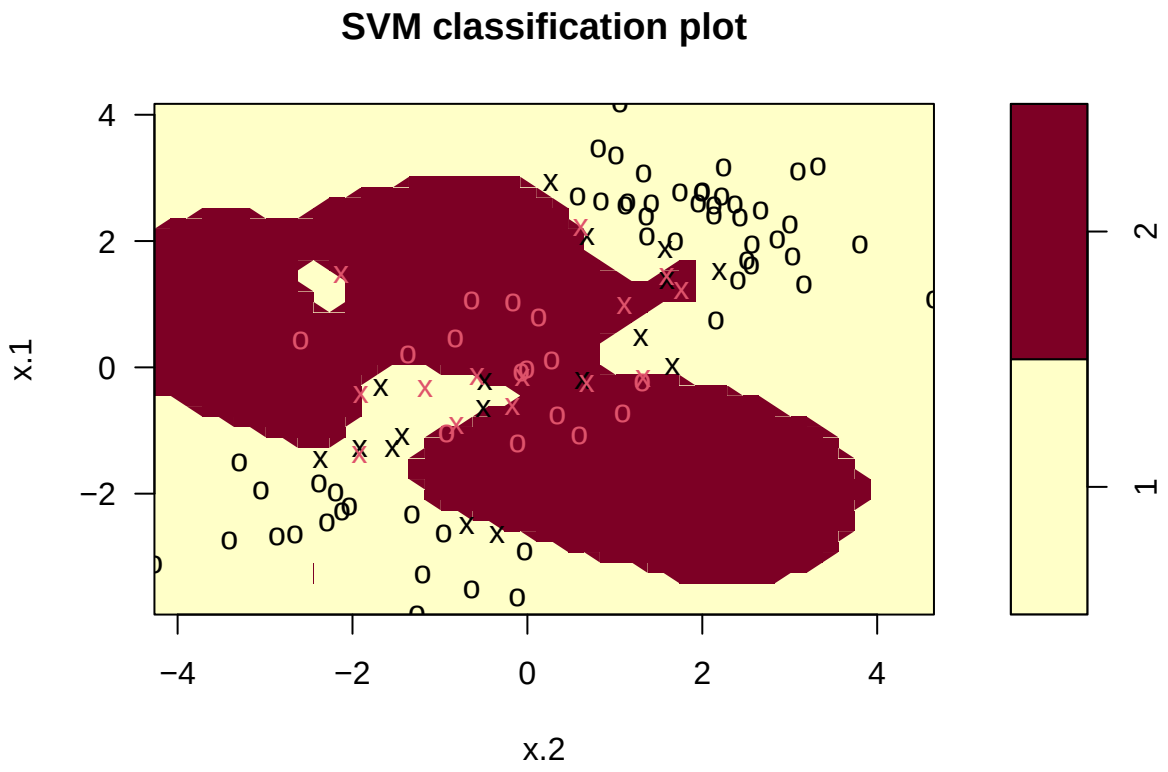



Notice that the above decision boundary is decidedly non-linear. It seems to perform reasonably well, but there are indeed some misclassifications. Let's see if increasing the cost [1] helps our classification error rate. Refit the svm with the radial kernel, $\gamma = 1$, and a cost of 10000. Plot this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

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

```
svm_model_2 <- svm(y ~ ., data=train_data, kernel="radial", gamma=1, cost=10000)

# Plot the SVM with the training data
plot(svm_model_2, train_data)
```




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

*Although increasing the cost might decrease classification error, the model might be overfitting, leading to the problem of not being general enough to be used on other data and gives inaccurate result.*

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_model_2, newdata=dat[-train,]))
```

67 of class 1 is correctly classified, and 19 of class 2 is correctly classified. 12 of class 1 is misclassified as class 2, while 2 of class 2 is misclassified as class 1. There seems to be more misclassifications when the model is used to predict the training set.

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.

```
proportion_train <- sum(dat[train, "y"] == 2) / length(train)

proportion_train
```

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

*0.29 being the proportion of class 2 in the training set, class 2 is more represented in the training set than the total dataset. However, this difference is only 4%, so the training partition is broadly representative of the underlying 25% of class 2 in the data as a whole.*

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)
cost_values <- c(0.1, 1, 10, 100, 1000)
gamma_values <- c(0.5, 1, 2, 3, 4)

tune.out <- tune(svm, y ~ ., data=train_data,
                 kernel="radial",
                 ranges=list(cost=cost_values, gamma=gamma_values),
                 scale=FALSE)
```

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,]))
```

*The model is able to identy class 1 more accurately with 73 of class 1 being classified correctly. However, the accuracy for class 2 decreased, as now there are only 18 observations of class 2 that are classified correctly. While misclassification of class 1 as class 2 dropped to 6 observations, misclassification of class 2 as class 1 increased to 3.*

Let's turn now to decision trees.

```r
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.

```r
heart$class_binary <- ifelse(heart$class > 0, 1, 0)
heart$class_binary <- as.factor(heart$class_binary)
```

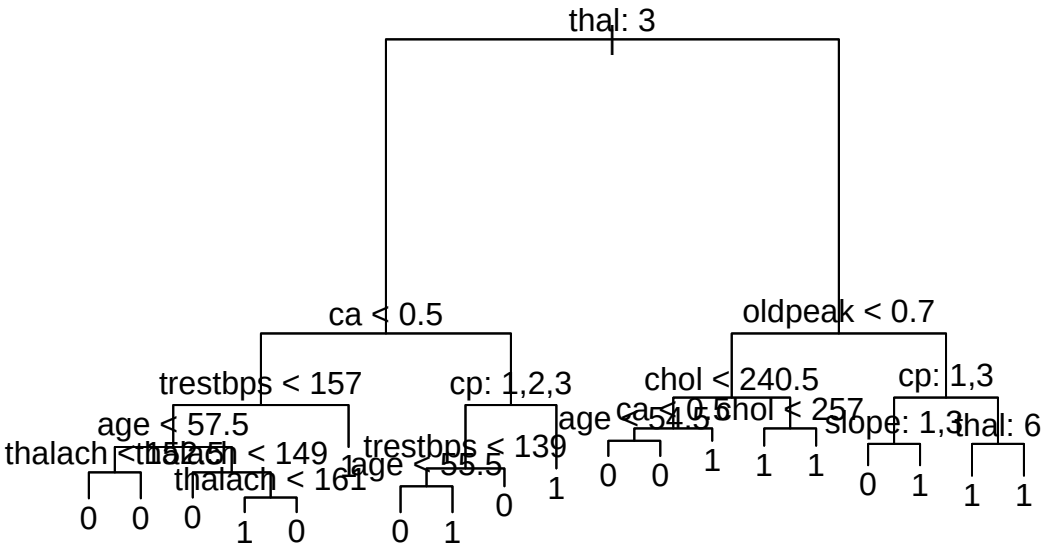
Train a classification tree on a 240 observation training subset (using the seed I have set for you). Plot the tree.

```r
set.seed(101)
train_indices <- sample(1:nrow(heart), 240)

train_data <- heart[train_indices, ]

heart_tree <- tree(class_binary ~ . - class, data = train_data)

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




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

```
test_data <- heart[-train_indices, ]

test_pred <- predict(heart_tree, newdata=test_data, type="class")

conf_matrix <- table(True = test_data$class_binary, Predicted = test_pred)
print(conf_matrix)
```

```
##     Predicted
## True  0  1
##    0 28  8
##    1  3 18
```

```
error_rate <- sum(diag(conf_matrix)) / nrow(test_data)
classification_error_rate <- 1 - error_rate
print(classification_error_rate)
```

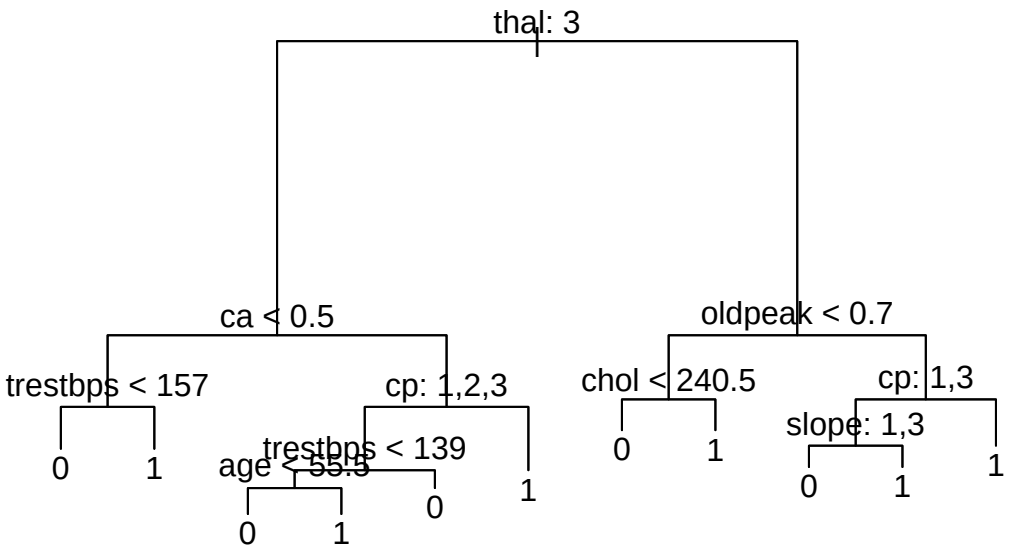
```
## [1] 0.1929825
```

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.

```
cv_heart_tree <- cv.tree(heart_tree, FUN=prune.misclass)
optimal_size <- which.min(cv_heart_tree$dev)

pruned_heart_tree <- prune.misclass(heart_tree, best=optimal_size)

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




```
test_pred_pruned <- predict(pruned_heart_tree, newdata=test_data, type="class")

conf_matrix_pruned <- table(True = test_data$class_binary, Predicted = test_pred_pruned)
print(conf_matrix_pruned)
```

```
##     Predicted
## True  0  1
##    0 29  7
##    1  4 17
```

```r
misclassification_rate_pruned <- sum(conf_matrix_pruned) - sum(diag(conf_matrix_pruned))
misclassification_rate_pruned <- misclassification_rate_pruned / sum(conf_matrix_pruned)

print(misclassification_rate_pruned)
```

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

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

*Pruning reduces the size of the tree to prevent overfitting and increase interpretability which could lead to decrease in accuracy from loss of relevant information, increase in bias and higher error rate for testing data. However, in this case, the misclassification error rate remains the same at approximately 19.298%. This suggests that although the tree was generalized, increasing interpretability and removing overfitting, pruning did not affect the accuracy of the model. The removed splits may not have much contribution to the predictive power of the model or are not as significant.*

Discuss the ways a decision tree could manifest algorithmic bias.

*Bias can be from a biased training data where different classes are not represented equally. Bias could also arise from trying to prevent overfitting. As the tree is generalized, it may no longer accurately represent the minority classes or complex patterns within the data, leading to biased predictions against these groups. Algortihmic choices like splitting criteria and stopping condition can also lead to favoring features that are more prevalent in the majority class and not capturing critical patterns in smaller or minority classes.*