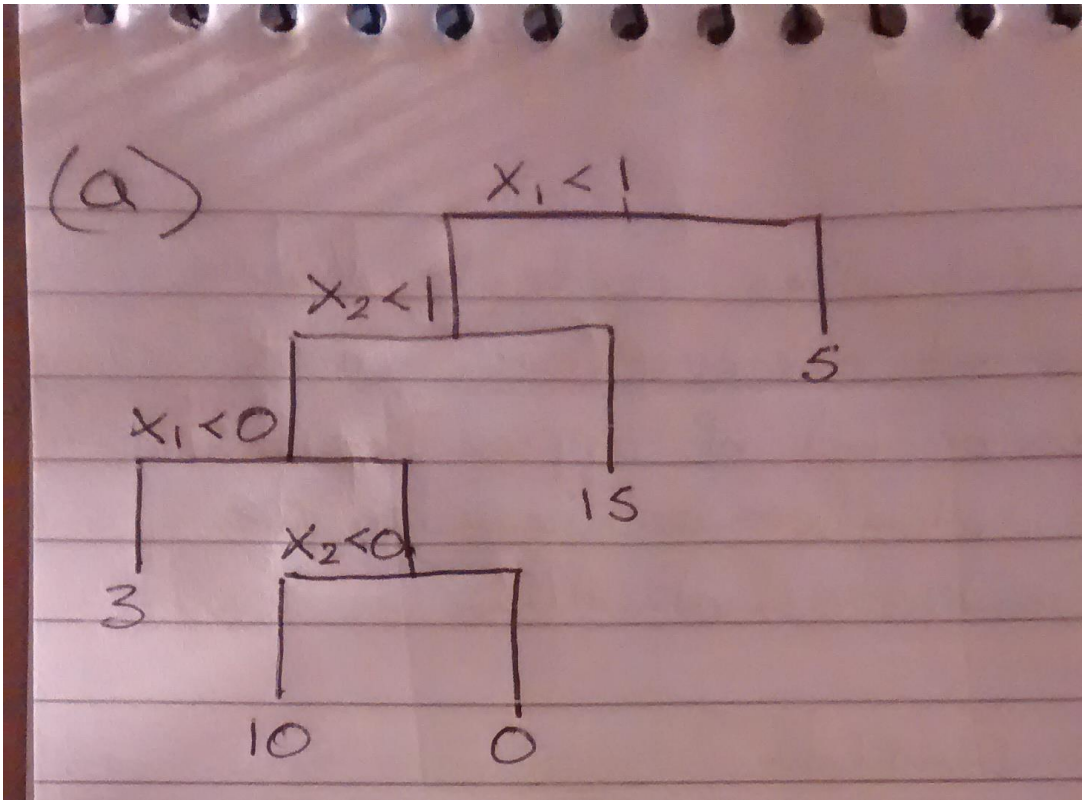
Homework 4

Darragh Hanley

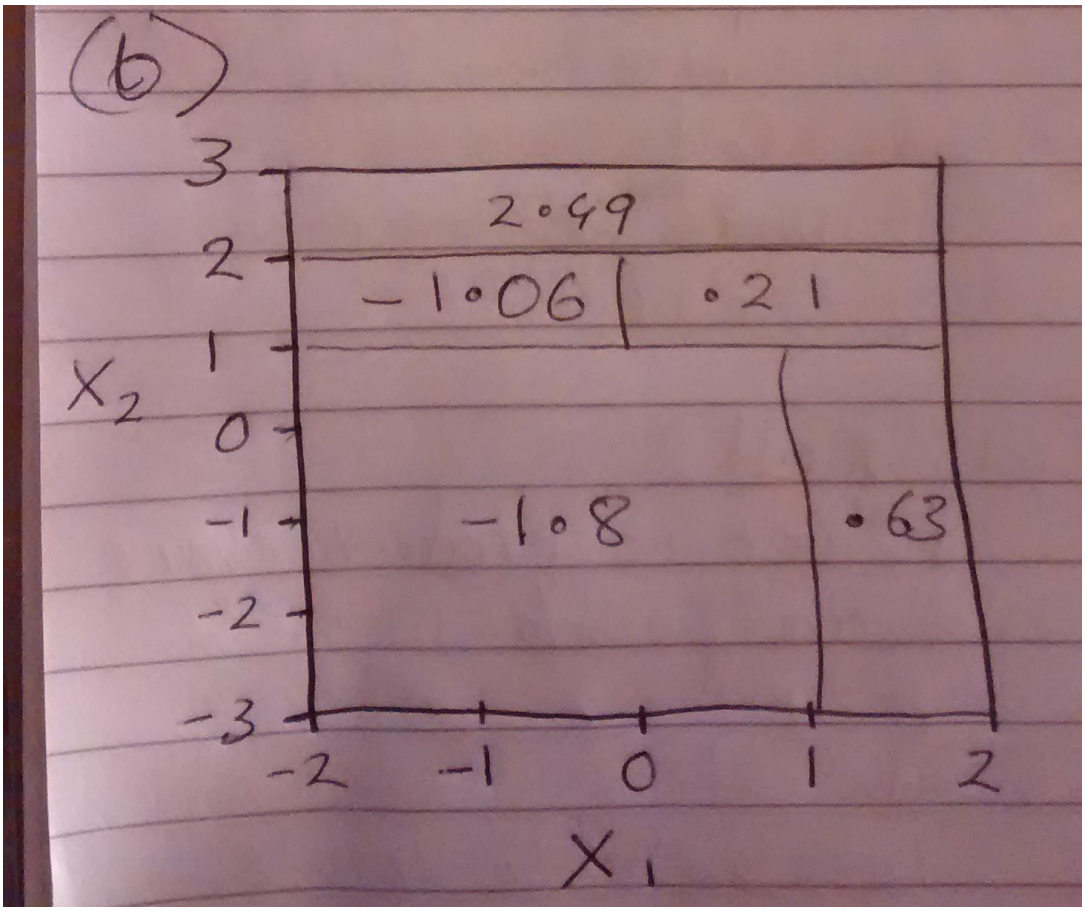
Tuesday, March 03, 2015

1. This question relates to the plots in Figure 8.12 of your textbook An Introduction to Statistical Learning.

(a) Sketch the tree corresponding to the partition of the predictor space illustrated in the left-hand panel of Figure 8.12. The numbers inside the boxes indicate the mean of Y within each region.



(b) Create a diagram similar to the left-hand panel of Figure 8.12, using the tree illustrated in the right-hand panel of the same figure. You should divide up the predictor space into the correct regions, and indicate the mean for each region.

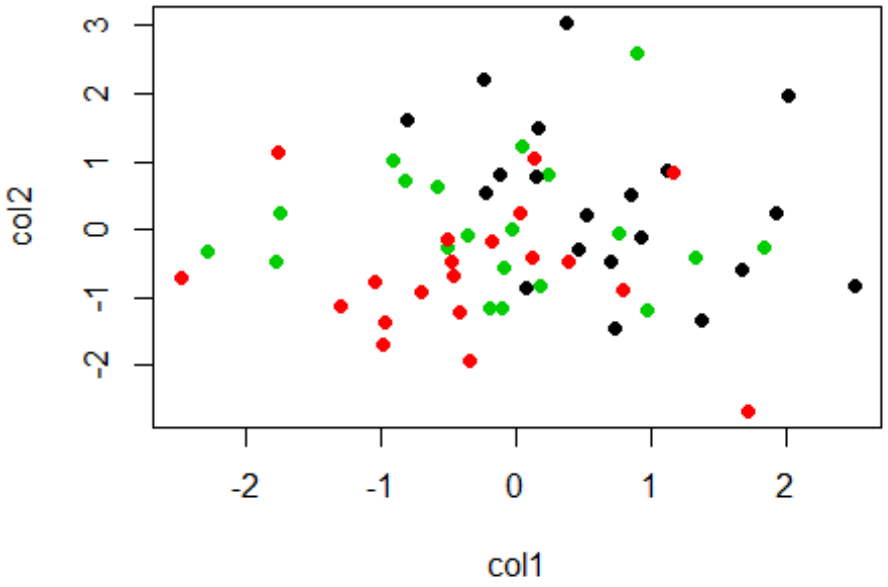


Q2(a) Generate a simulated data set with 20 observations in each of three classes (i.e. 60 observations total), and 50 variables. Hint: There are a number of functions in R that you can use to generate data. One example is the rnorm() function;runif() is another option. Be sure to add a mean shift to the observations in each class so that there are three distinct classes.

For this exercise I worked Andrew Beckerman, Sevvandi Kandanaarachchi and Tony Wu.

```
# Lets set our seed to get reproducable results.
set.seed(100)
# For the response variable y create 20 of each response type
y <- rep(c(1,2,3),20 )
# For the predictions create a matrix with 50 columns * 60rows of random normal observati
ons
x <- matrix(rnorm(60*50), ncol=50) # defaults to mean = 0 and sd = 1
# Shift the classes all .7 unit apart
x[y==2,]= x[y==2,] - .6
x[y==3,]= x[y==3,] + .6
# Give the matrix column and row names
dimnames(x) <- list(rownames(x, do.NULL = FALSE, prefix = "row"),
colnames(x, do.NULL = FALSE, prefix = "col"))
# Plot the first two columns against each other to examine the data.
```

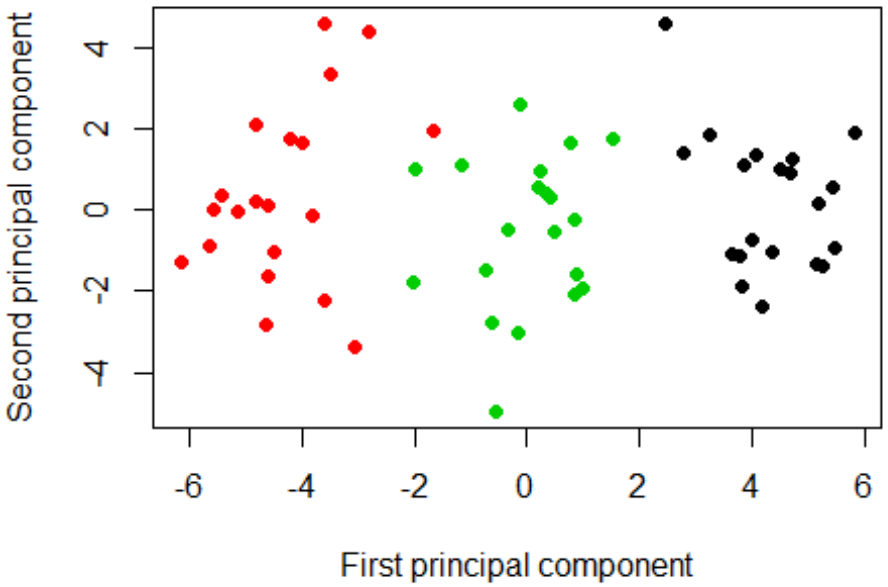
```
par(mfrow=c(1,1))
plot(x[,1:2], col =(4-y), pch=19)
```



(b) Perform PCA on the 60 observations and plot the first two principal component score vectors. Use a different color to indicate the observations in each of the three classes. If the three classes appear separated in this plot, then continue to part (c). If not, then return to part (a) and modify the simulation so that there is greater separation between the three classes. Do not continue to part (c) until the three classes show at least some separation in the first two principal component score vectors.

We now perform principal components analysis using the prcomp(). The variables are already scaled so no need to do this.

```
pr.out =prcomp(x, scale =FALSE)
# Plot the first two principal component score vectors
plot(pr.out$x[,1:2], col=4-y, pch =19, xlab ="First principal component", ylab="Second pr
incipal component")
```



From the above plot we can see the first component does a good job of separating the classes, however the second class does not so good.

(c) Perform K-means clustering of the observations with K = 3. How well do the clusters that you obtained in K-means clustering compare to the true class labels? Hint: You can use the table() function in R to compare the true class labels to the class labels obtained by clustering. Be careful how you interpret the results: K-means clustering will arbitrarily number the clusters, so you cannot simply check whether the true class labels and clustering labels are the same.

The function kmeans() performs K-means clustering in R.

```
# Option nstart attempts 20 initial random centorids
km.out.c =kmeans(x,3, nstart =20)
# create a contigency table of the assigned cluster and the actual class
table(km.out.c$cluster, y, dnn=c("Cluster","Class"))
```

```
##         Class
## Cluster  1  2  3
##       1  0 20  0
##       2  0  0 20
##       3 20  0  0
```

Although K-means clustering will arbitrarily number the clusters, we see that each arbitrary cluster is assigned to one class only. K-means performs perfectly in this case.

(d) Perform K-means clustering with K = 2. Describe your results.

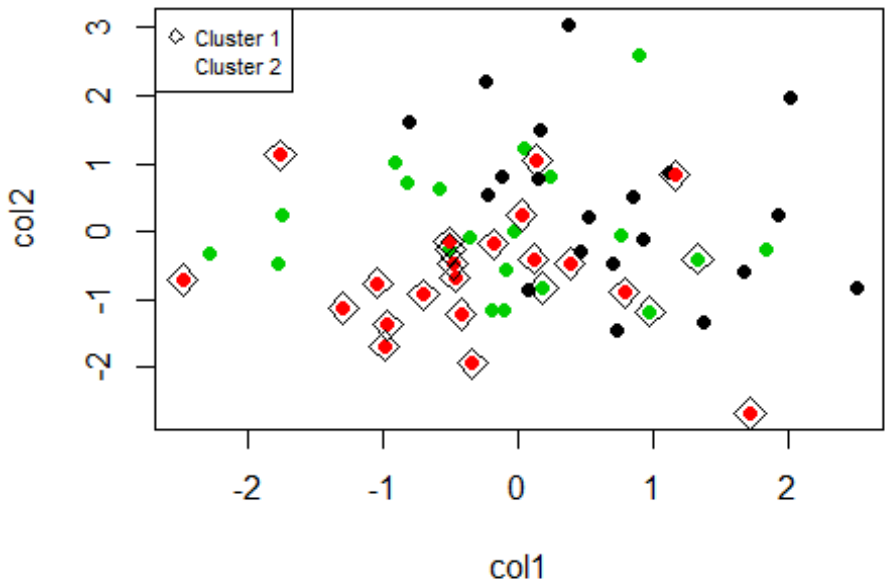
```
# Change to 2 clusters
km.out.d =kmeans(x,2, nstart =20)
# We can see hear that two of the clusters
table(km.out.d$cluster, y, dnn=c("Cluster","Class"))
```

```
##         Class
## Cluster  1  2  3
##       1  4 20  0
##       2 16  0 20
```

In the table above we see that two of the classes are perfectly assigned to two clusters. ie. class 2 to cluster 1 and class 3 to cluster 2. However as we only have two classes K-means has forced the class 1 to be split among the two clusters - however, majority of class1 observations go to cluster 2.

Below we plot the points, response variables (color) and the cluster class (point border or not) for the first two variables.

```
# plot only for the first two variables to see how the points are assigned
plot(x[,1:2], col=(4-y), pch=19)
points(x[km.out.d$cluster==1,1:2], pch=5, cex = 1.5)
legend("topleft", c(paste("Cluster",unique(km.out.d$cluster))), pch=c(5,27), cex=.7)
```



(e) Now perform K-means clustering with K = 4, and describe your results.

```
# Change to 4 clusters
km.out.e =kmeans(x,4, nstart =20)
# Show the contingency table of clusters and class
table(km.out.e$cluster, y, dnn=c("Cluster","Class"))
```

```
##         Class
## Cluster  1  2  3
##       1 12  1  0
##       2  0 17  0
##       3  0  0 18
##       4  8  2  2
```

In the table above, we see that the majority of observations in each class are assigned to one cluster only. Class 1 to cluster 1. Class 2 to cluster 2. Class 3 to cluster 3. However k-means seemed to have forced

some observations from each class into a fourth cluster (cluster 4). We also see an outlier, with class 2 having one value in cluster 1.

(f) Now perform K-means clustering with K = 3 on the first two principal component score vectors, rather than on the raw data. That is, perform K-means clustering on the 60x2 matrix of which the first column is the first principal component score vector, and the second column is the second principal component score vector. Comment on the results.

Performs k-means with K=3, replacing on the first two principal components

```
km.out.f =kmeans(pr.out$x[,1:2],3, nstart =20)
```

Below it can be seen that only the first two principal components almost perfectly separate each class into unique clusters. There is only one outlier in class 2. Here we have reduced the variables from 50 to 2, and achieved almost the same accuracy as seen in part (c) where three clusters were also used.

```
table(km.out.f$cluster, y, dnn=c("Cluster","Class"))
```

```
##         Class
## Cluster  1  2  3
##       1  0  0 20
##       2 20  1  0
##       3  0 19  0
```

(g) Using the scale() function, perform K-means clustering with K = 3 on the data after scaling each variable to have standard deviation one. How do these results compare to those obtained in (c)? Explain.

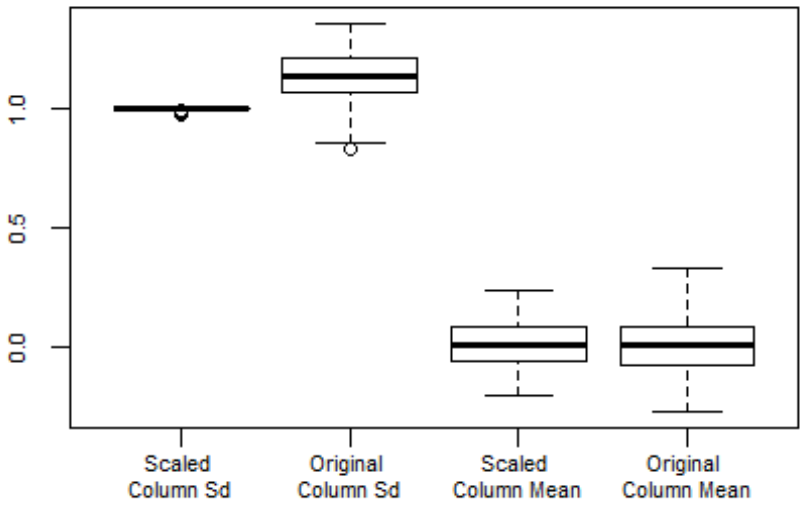
Lets first scale our variables to have standard deviation of 1. We do not want to center them. The default of the scale function is sd=1. This is what the question asks for.

```
x.scale <- scale(x, center = FALSE, scale = TRUE)
# Now lets look at the clustering result. It performs just as well as in part (c). Lets e
xplore why this is.
km.out.g =kmeans(x.scale, 3, nstart =20)
table(km.out.g$cluster, y, dnn=c("Cluster","Class"))
```

```
##         Class
## Cluster  1  2  3
##       1  0  0 20
##       2 20  0  0
##       3  0 20  0
```

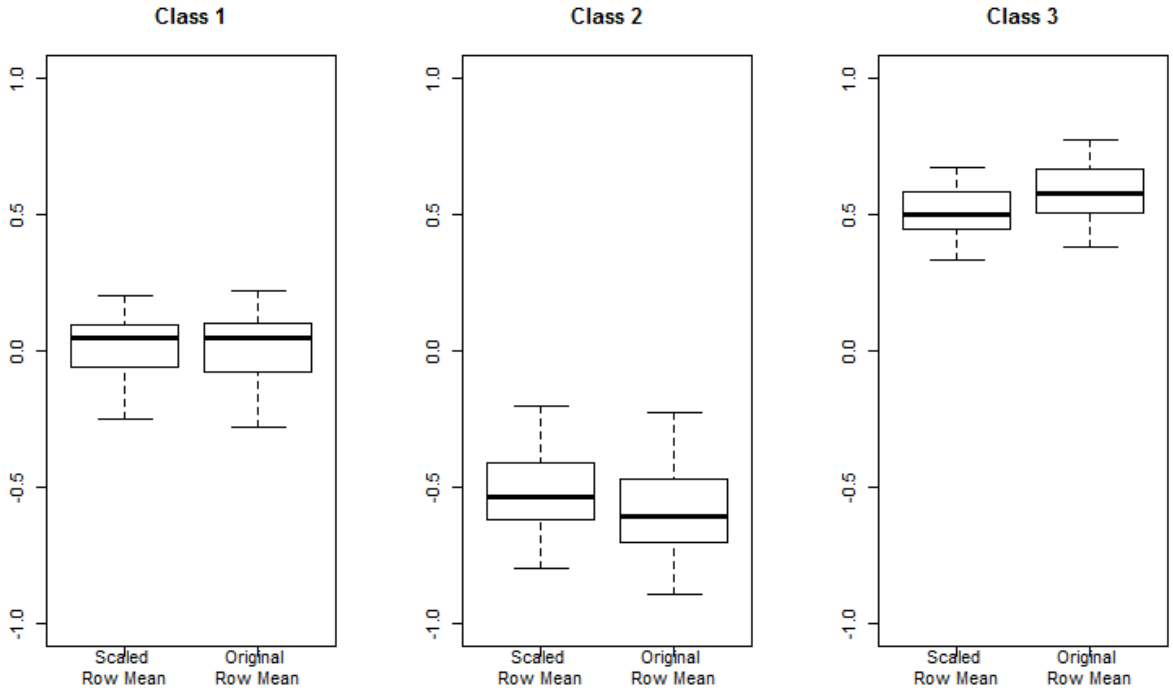
First lets check if the data (x.scale) is scaled to Sd = 1. From below it appears like it is. We do notice the the columnwise means have shifted slightly.

```
# for each variable/column, get the mean and sd before and after the scaling
check.col <- cbind(apply(x.scale,2,sd), apply(x,2,sd), apply(x.scale,2,mean), apply(x,2,m
ean))
colnames(check.col) <- c("Scaled \nColumn Sd","Original \nColumn Sd","Scaled \nColumn Mea
n","Original \nColumn Mean")
# create a boxplot for the data to compare
boxplot(check.col, cex.axis=0.7)
```



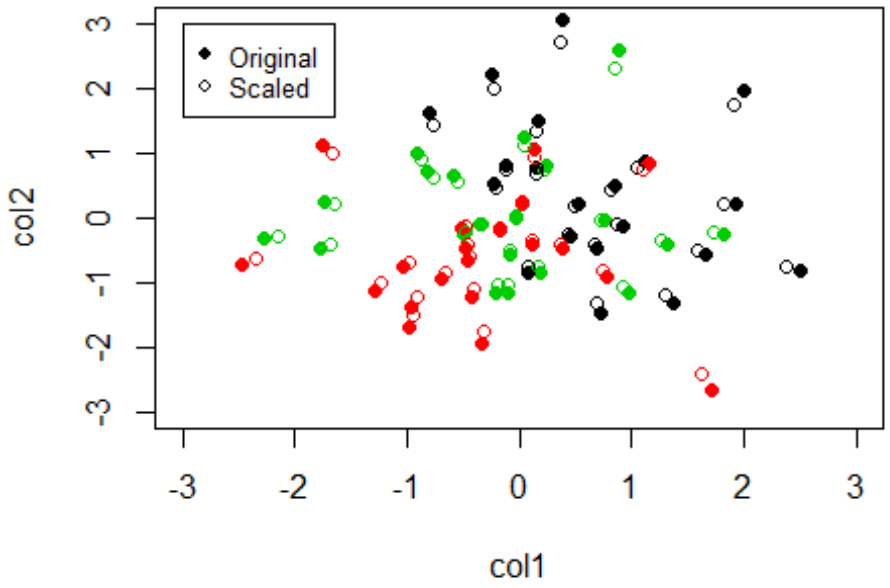
Lets examine the rowwise means to see how they compare after the scaling. The below plot shows a slight change in rowwise mean of each class however the general mean shift performed in part (a) of the question is very much kept intact.

```
# for each row, get the mean before and after the scaling
check.row <- cbind(apply(x.scale,1,mean), apply(x,1,mean))
colnames(check.row) <- c("Scaled \nRow Mean","Original \nRow Mean")
par(mfrow=c(1,3))
boxplot(check.row[y==1,], ylim=c(-1,1), main="Class 1")
boxplot(check.row[y==2,], ylim=c(-1,1), main="Class 2")
boxplot(check.row[y==3,], ylim=c(-1,1), main="Class 3")
```



Now lets look at some data points. We can see that there has been a very small centering of the data points. However each class (or color) has its own center it is shifting into.

```
par(mfrow=c(1,1))
plot(x[,1:2], col =(4-y), pch=19, xlim=c(-3,3), ylim=c(-3,3))
points(x.scale[,1:2], col =(4-y), pch=1)
legend(-3,3, c("Original","Scaled"), pch=c(19,1), cex=.8)
```



Given the above boxplots it can be seen that the data within each cluster does not move much with the scaling, so we would expect that the clustering should be able to get similar results with the columnwise scaling of SD as without the scaling of SD in part (c).

3. You may work in groups up to size 4 on this problem. If you do work in groups, write the names of all your group members on your problem set. Recall the body dataset from problem 4 of Homework 3. In that problem we used PCR and PLSR to predict someone's weight. Here we will re-visit this objective, using bagging and random forests. Start by setting aside 200 observations from your dataset to act as a test set, using the remaining 307 as a training set. Ideally, you would be able to use your code from Homework 3 to select the same test set as you did on that problem.Using the randomForest package in R (hint: see section 8.3.3 in the textbook for guidance), use Bagging and Random Forests to predict the weights in the test set, so that you have two sets of predictions. Then answer the following questions:

For this exercise I worked Andrew Beckerman, Sevvandi Kandanaarachchi and Tony Wu.

```
# Place the data set in this directory
setwd("C:/Users/dahanley/Google Drive/stats216/Homework 4")
load("body.RData")
# Set our seed, load our library and sample the 200 test data point, the rest we put in t
rain.
set.seed(100)
# create the index for our test and train set
test = sort(sample(1:nrow(X), 200))
train = (1:nrow(X))[-test]
# Create a data frame with the predictors and the weight, which is the response.
mydf <- data.frame(Y$Weight, X)
# Create a vector with the test weight observations.
Weight.test=mydf[-train,"Y.Weight"]
# The randomForest() function can be used to perform both random forests and bagging.
library (randomForest)
```

Bagging first : The argument mtry=ncol(X) indicates that all X predictors should be considered for each split of the tree in other words, that bagging should be done. In both models we pass the train and test set

```
bag.Weight =randomForest(Y.Weight ~ .,data=mydf ,subset=train, xtest=mydf[test,-1],
                         ytest=mydf[test,1], mtry=ncol(X), importance =TRUE)
```

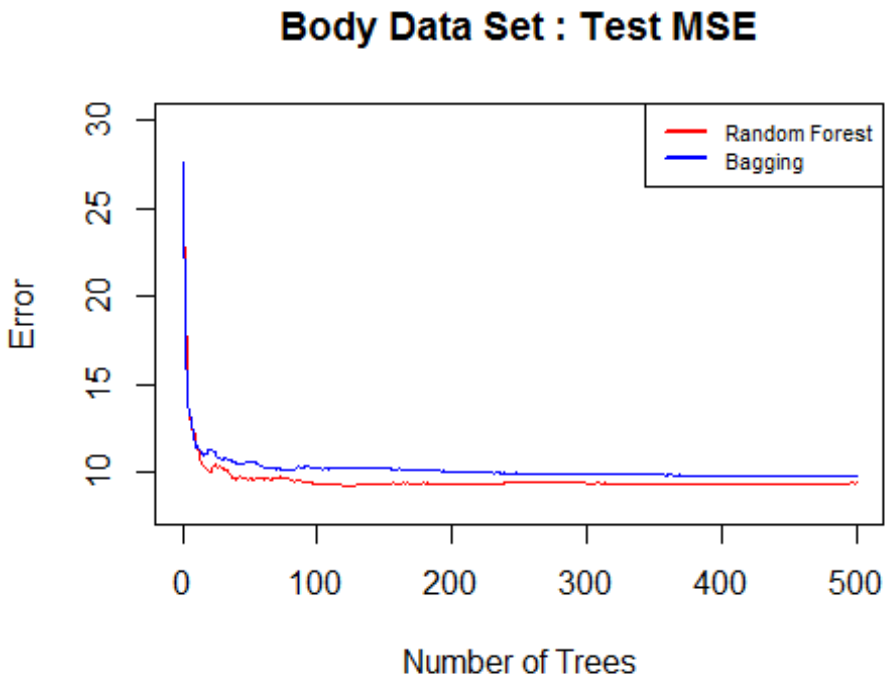
Random Forest : proceeds in exactly the same way, except that we use a smaller value of the mtry argument. By default, randomForest() uses p/3 variables when building a random forest of regression trees.

```
rf.Weight =randomForest(Y.Weight ~ .,data=mydf ,subset=train, xtest=mydf[test,-1],
                        ytest=mydf[test,1], importance =TRUE)
```

(a) Produce a plot of test MSE (as in Figure 8.8 in the text) as a function of number of trees for Bagging and Random Forests. You should produce one plot with two curves, one corresponding to Bagging and the other to Random Forests. Hint: If you read the documentation for the randomForest() function, you can find a way to obtain the data for both curves with only one call each to the randomForest() function.

The randomForest() function allows us to extract the test mse, if a test set is given (through the xtest or additionally ytest arguments) which we did. Below we extract this using the $test list and pulling the second element of the list, which contais the mse. We create an empty plot and for each model add lines of the mse against the number of trees.

```
plot(1, type="n", xlab="Number of Trees", ylab="Error", xlim=c(0, 500), ylim=c(8, 30), ma
in = "Body Data Set : Test MSE")
lines(1:500, rf.Weight$test[[2]], col="red")
lines(1:500, bag.Weight$test[[2]], col="blue")
legend("topright", c("Random Forest", "Bagging"), col=c("red", "blue"), lwd=2, cex=.7)
```

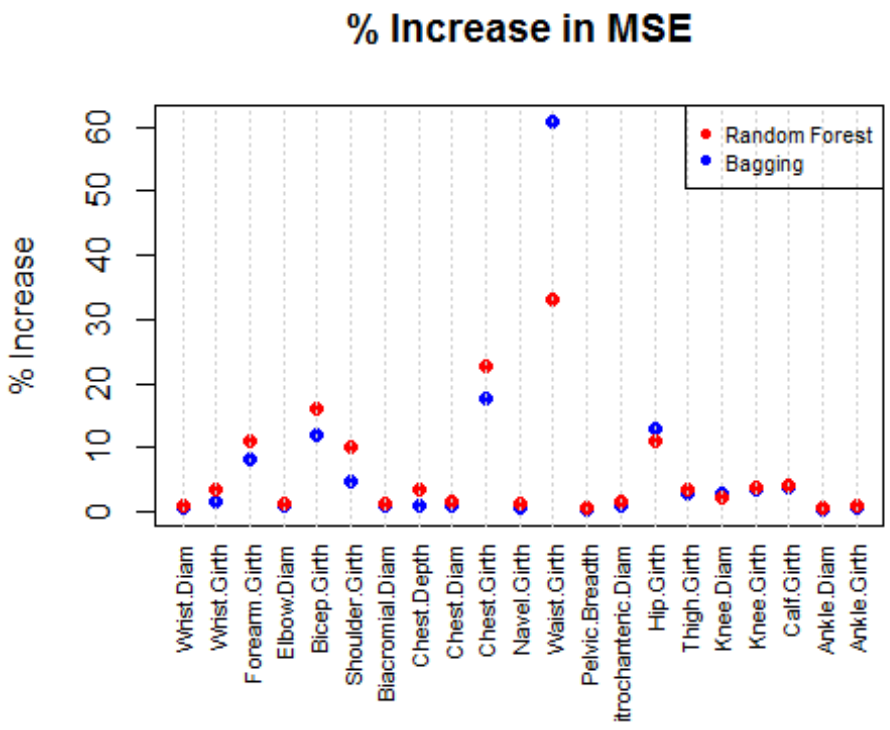


(b) Which variables does your random forest identify as most important? How do they compare with the most important variables as identified by Bagging?

Instead of using function varimpplot() to plot the variable importance we shall pull our own plot in order to compare the result of both models against each other.

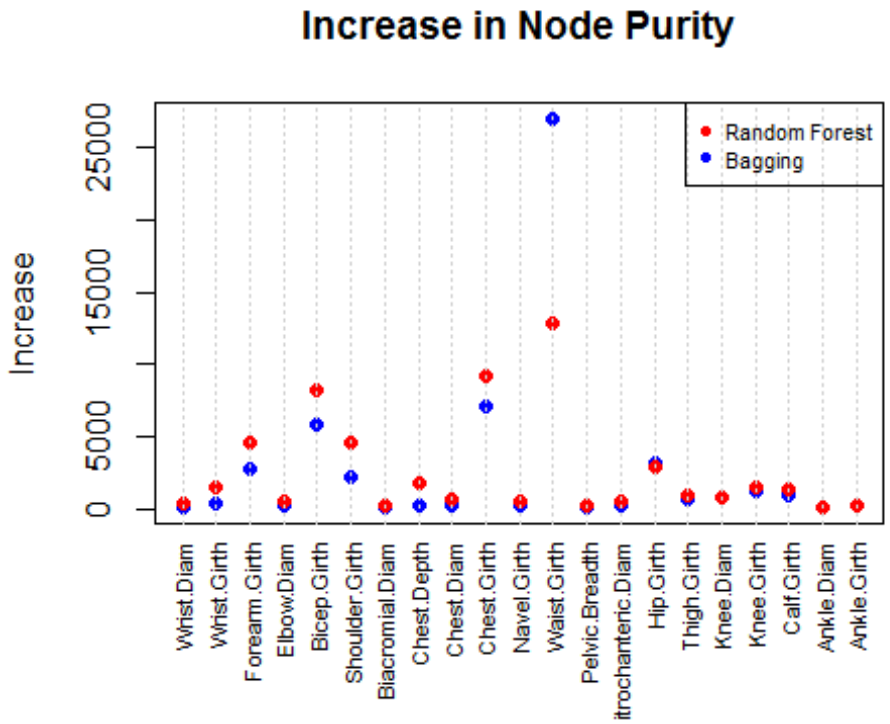
Lets plot the % increase in MSE

```
plot(bag.Weight$importance[,1], xaxt='n', xlab="", ylab="% Increase", pch=19, col="blue",
main="% Increase in MSE")
points(rf.Weight$importance[,1], pch=19, col="red")
abline(v=(seq(0,21,1)), col="lightgray", lty="dotted")
axis(1, at=1:21, labels=names(X), tick=FALSE, las=2, line=-0.5, cex.axis=0.7)
legend("topright", c("Random Forest", "Bagging"), col=c("red", "blue"), pch=19, cex=.7)
```



Now lets look at the the Increase in Node Purity

```
plot(bag.Weight$importance[,2], xaxt='n', xlab="", ylab="Increase", pch=19, col="blue",
     main="Increase in Node Purity")
points(rf.Weight$importance[,2], pch=19, col="red")
abline(v=(seq(0,21,1)), col="lightgray", lty="dotted")
axis(1, at=1:21, labels=names(X), tick=FALSE, las=2, line=-0.5, cex.axis=0.7)
legend("topright", c("Random Forest", "Bagging"), col=c("red", "blue"), pch=19, cex=.7)
```



By looking at both charts above, we can see in both the Random Forest and Bagging, Waist Girth stands out as the most important variable. However for Bagging the associated importance laid on Waist Girth is much higher. This makes sense as we know Bagging looks at every variable for every split and can tend to let important variable overly dominate. Random forest only chooses a random subset (in our case 1/3) of variables at every split, because of this other variables get to influence the model more.

Waist Girth is followed by Chest Girth and then Bicep Girth in each model and measurement. However random forest lays higher importance on these than bagging does. In general the .girth measurements are more important than the diameter readings.

(c) Compare the test error of your random forest (with 500 trees) against the test errors of the three methods you evaluated in problem 4(f) on Homework 3. Does your random forest make better predictions than your predictions from Homework 3? If you did not successfully solve problem 4(f) on Homework 3, you may compare the test error of your random forest against the test errors in the Homework 3 solutions.

Lets recalculate the test errors from 4(f) using the same seed and code.

```
# Fit the linear regression using the best subset selection
set.seed(100)
# Output the features which are chosen, when subsetting to 5 features using best subsets
load the package
library(leaps)
library(pls)
# fit the linear regression based best subset selection, default method is "exhaustive"
regfit.full = regsubsets(Y.Weight ~ ., data = mydf[train,], nvmax = 21)
# output the summary and extract the best subsets features.
reg.summary = summary(regfit.full)
features <- subset(reg.summary$which[9,], reg.summary$which[9,] == TRUE)
# Fit the three models from HW3 - Best subsets, PCR, PLSR.
bsubs.mod <- glm(mydf$Y.Weight[train] ~ ., data = mydf[train,reg.summary$which[9,]])
pcr.mod <- pcr(Y.Weight ~ .,ncomp = 3, data = mydf[train,], scale=TRUE)
plsr.mod <- plsr(Y.Weight ~ .,ncomp = 4, data = mydf[train,], scale=TRUE)
# Calculate the MSE for each model and hold in a vector of results
mse.all <- vector()
mse.all[1] <- mean((Weight.test-predict(bsubs.mod, mydf[test,]))^2)
mse.all[2] <- mean((Weight.test-predict(pcr.mod, mydf[test,]))^2)
mse.all[3] <- mean((Weight.test-predict(plsr.mod, mydf[test,]))^2)
# Now we add the Random forest results using 500 trees
mse.all[4] <- rf.Weight$test[[2]][500]
# add the names of
names(mse.all) <- c("Best subset MSE (9 Features)", "PCR MSE (3 components)",
                    "PLSR MSE (4 components)", "Random Forest (500 trees)")
mse.all
```

```
## Best subset MSE (9 Features)       PCR MSE (3 components)
##                     7.587586                     8.623650
##      PLSR MSE (4 components)    Random Forest (500 trees)
##                     7.846782                     9.422534
```

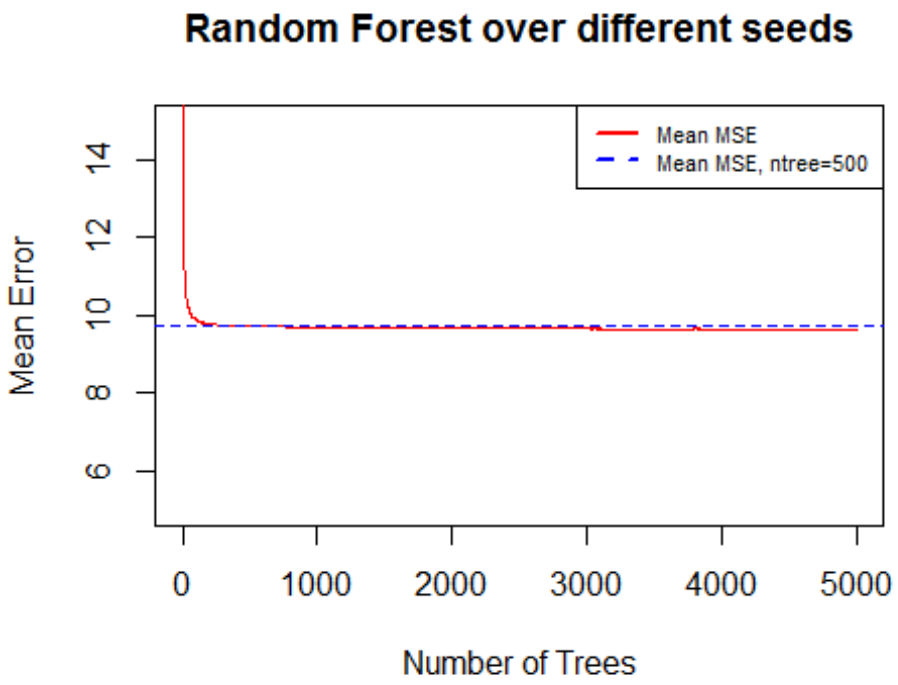
we can see above that the Random Forest using 500 trees did not perform as well as the feature selection models used in part 4(f)

(d) The randomForest() function uses 500 as the default number of trees. For this problem, would it be valuable to include more trees? How can you tell?

To answer this question lets run Random Forest a number of times up to 5000 trees, each time changing the seed. Plot the mean of the mse and see if iterations much larger than 500 produce a better results than 500 iterations.

For Random Forest, the improvement between 500 and 5000 trees appears in the below plot to be negligible so there would not be significant value in going over 500 trees. In fact as can be seen below the plot the improvement between 400 and 500 trees is nearly as great as the improvement between 500 and 5000 trees. Therefore there is not much value in adding more trees beyond 500.

```
# We shall also reshuffle the train and test data with each seed.
set.seed(1)
n = 5000 # number of trees
iter = 50 # number of iterations
mod.mse = matrix(NA,nrow=n, ncol=iter)
# first lets pull the model data for different seeds.
for(i in 1:iter){
    set.seed(100+i)      # change the seed for each itteration
    test = sort(sample(1:nrow(X), 200))
    train = (1:nrow(X))[-test]
    # a random forest of regression trees.
    rf.mod =randomForest(Y.Weight ~ .,data=mydf ,subset=train, xtest=mydf[test,-1],
                         ytest=mydf[test,1], ntree=n, importance =TRUE)
    mod.mse[,i] <- rf.mod$test[[2]]
}

# Create an empty plot and for each model add lines of the mse against the number of tree
s.
plot(1, type="n", xlab="Number of Trees", ylab="Mean Error", xlim=c(0, n), ylim=c(5, 15),
main = "Random Forest over different seeds")
lines(1:n, rowMeans(mod.mse), col="red")
abline(h=rowMeans(mod.mse)[500],col="blue", lty=2)
legend("topright", c("Mean MSE", "Mean MSE, ntree=500"), col=c("red", "blue"), lty=c(1,2)
, lwd=2, cex=.7)
```



Below we see, for Random Forest, the MSE at 500 trees, MSE at 5000 trees, and MSE improvement from 500 to 5000 trees. As can be seen the improvement is relatively negligible.

```
c(mod.mse[500], mod.mse[5000],(mod.mse[500] - mod.mse[5000]))
```

```
## [1] 9.79473150 9.73483165 0.05989985
```

Below we see, for Random Forest, the MSE at 300 trees, MSE at 500 trees, and MSE improvement from 400 to 500 trees. Here we see the improvement between 400 and 500 trees is that same as the improvement from 500 to 5000 trees.

```
c(mod.mse[400], mod.mse[500],(mod.mse[400] - mod.mse[500]))
```

```
## [1] 9.8478764 9.7947315 0.0531449
```

Lets check for bagging also call to the function (here we will use the average of 10 calls for performance) to see if results are the same. And we see the results are quite similar as before. Only a very marginal improvement between 500 and 5000 trees so it is not very valuable.

```
mod.mse.bag = matrix(NA,nrow=n, ncol=10)
# first lets pull the model data for different seeds.
for(i in 1:10){
    set.seed(100+i)      # change the seed for each iteration
    test = sort(sample(1:nrow(X), 200))
    train = (1:nrow(X))[-test]
    # a bagging of regression trees using all variables
     bag.mod =randomForest(Y.Weight ~ .,data=mydf ,subset=train, xtest=mydf[test,-1],
                           ytest=mydf[test,1], ntree=n, mtry=ncol(X), importance =F)
     mod.mse.bag[,i] <- bag.mod$test[[2]]
}
mod.mse <- rowMeans(mod.mse.bag)
bag.mse <- c(mod.mse[500],mod.mse[5000], (mod.mse[500] - mod.mse[5000]))
```

```
names(bag.mse) <- c("mse @ 500", "mse @ 5000", "delta mse 500 - 5000")
bag.mse
```

```
##            mse @ 500           mse @ 5000 delta mse 500 - 5000
##          10.14916175          10.07073904           0.07842271
```

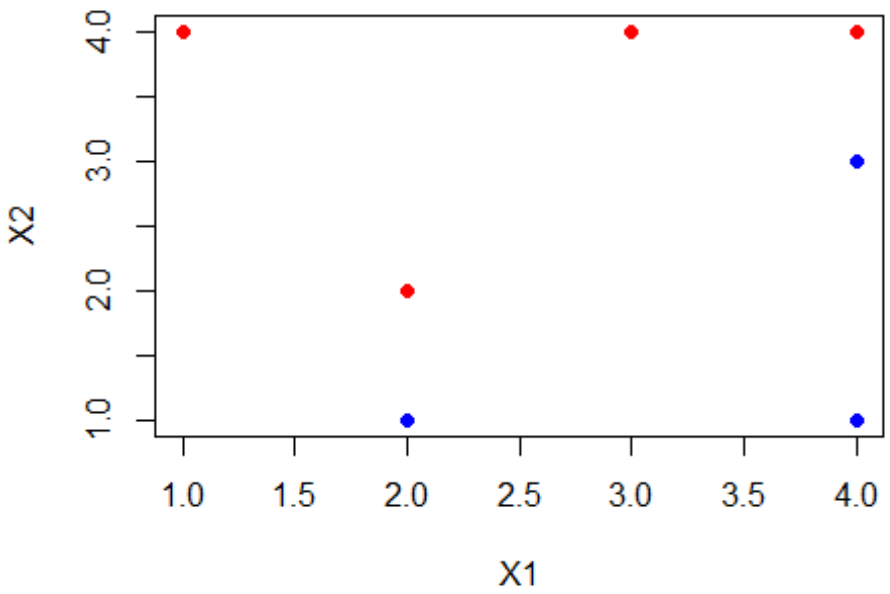
4 (a) We are given n = 7 observations in p = 2 dimensions. For each observation, there is an associated class label. Plot or sketch the observations.

Create a data frame with the observations.

```
X1 <- c(3,2,4,1,2,4,4)
X2 <- c(4,2,4,4,1,3,1)
Y  <- c(rep("RED", 4),rep("BLUE", 3))
mydf <- data.frame(X1, X2, Y)
```

Now lets plot the data

```
plot(X1, X2, col=Y, pch=19, data=mydf)
```



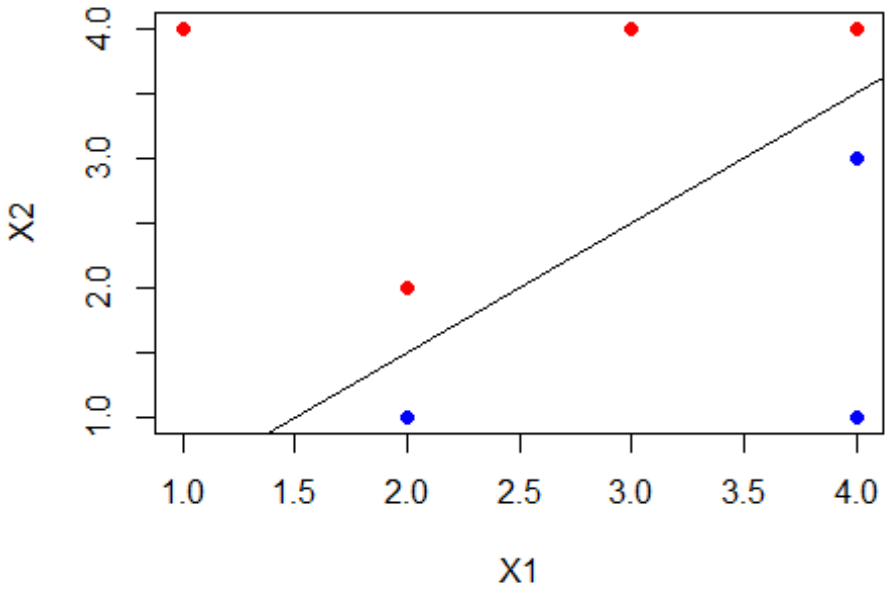
(b) Plot or sketch the separating hyperplane with maximum margin, and provide the equation for this hyperplane.

The maximal margin hyperplane is the separating hyperplane which is farthest from the training observations. We will find this with the package e1071 which contains the svm function we will use. We then compute the fit of a linear separating hyperplane with maximum margin. Notice that we have to specify a cost parameter, which is a tuning parameter.

```
library(e1071)
fit.svm = svm(Y ~ ., data = mydf, kernel = "linear", cost = 10, scale = FALSE)
```

we now leverage the code in the STATS216 Chapter 9 SVM R lab, to extract the coefficients of the hyperplane.

```
# Extract beta_0 and beta_1
beta0 = fit.svm$rho
beta = drop(t(fit.svm$coefs) %*% as.matrix(mydf[fit.svm$index,1:2]))
# Replot, this time with the solid line representing the maximal margin plane.
plot(X1, X2, col=Y, pch=19, data=mydf)
abline(beta0/beta[2], -beta[1]/beta[2])
```



The a, b arguments above in abline() represent the intercept and slope, single values in the plot functions.

```
paste("Intercept: ", round(beta0/beta[2],1), ", Slope: ", round(-beta[1]/beta[2],1), sep=
"")
```

```
## [1] "Intercept: -0.5, Slope: 1"
```

Therefore the formula for this line will take the form : **-0.5 + X1 - X2 = 0**

(c) Describe the classification rule for the maximal margin classifier. It should be something along the lines of "Classify to Red if Beta0 + Beta1 X1 + Beta2 X2 > 0, and classify to Blue otherwise." Provide the values for Beta0, Beta1, and Beta2.

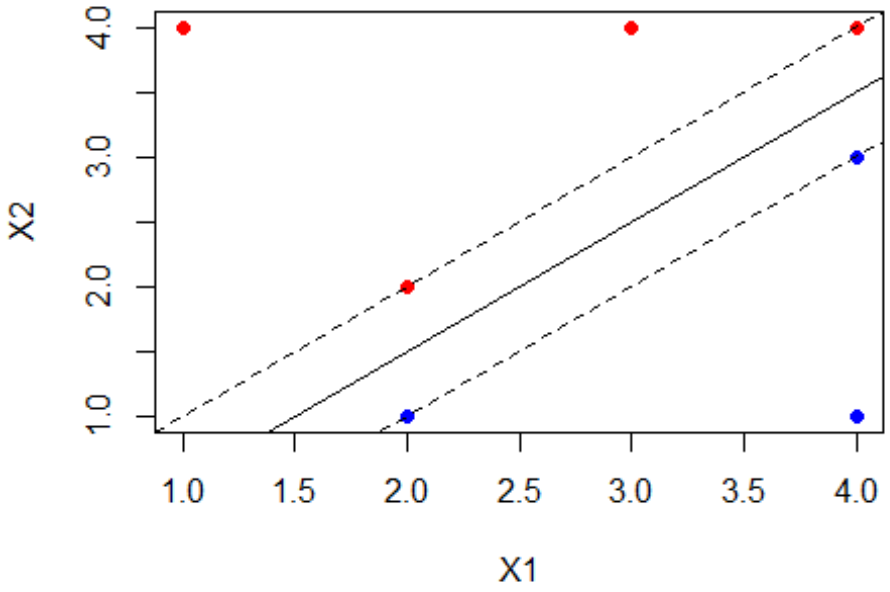
Classify to Red if -0.5 + X1 - X2 < 0 and classify to Blue otherwise.

$$\beta_0 = -0.5, \beta_1 = 1, \beta_2 = -1$$

(d) On your plot or sketch, indicate the margin for the maximal margin hyperplane. How wide is the margin?

Lets plot two dashed lines as the margins of the maximal margin plane.

```
# Replot, this time with the dashed lines representing the margins for the maximal margin hyperplane.
plot(X1, X2, col=Y, pch=19, data=mydf)
abline(beta0/beta[2], -beta[1]/beta[2])
abline((beta0 - 1)/beta[2], -beta[1]/beta[2], lty = 2)
abline((beta0 + 1)/beta[2], -beta[1]/beta[2], lty = 2)
```



To find the margin length we compute the smallest distance from any training observation to the given separating hyperplane. This is the same as computing the distance from the dashed margin line to the solid hyperplane. Here we use parallel geometry to get the required margin width. Note the margin width is from the solid line to either of the dashed lines.
(See equation in the following link under section "Distance between two parallel lines" :
http://en.wikipedia.org/wiki/Parallel_%28geometry%29)

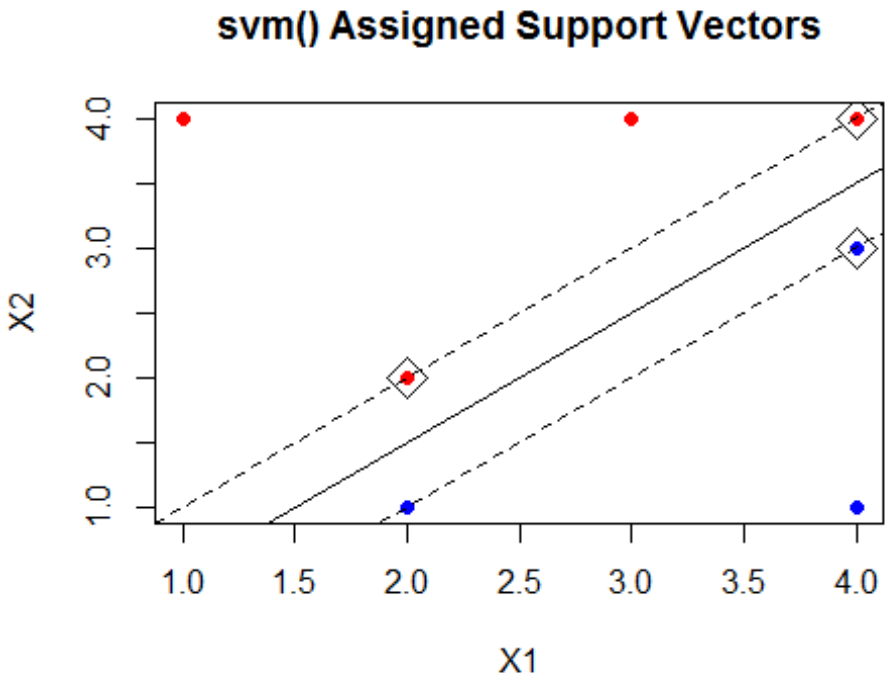
```
d = abs(beta0/beta[2] - (beta0 + 1)/beta[2]) / sqrt((-beta[1]/beta[2])^2+1)
names(d) <- "Margin Width"
d

## Margin Width
##    0.3535941
```

The maximal margin hyperplane is shown as a solid line. The "Margin Width", seen above, is the distance from the solid line to either of the dashed lines.

(e) Indicate the support vectors for the maximal margin classifier.

Highlight the points svm used as support vectors on the data set.

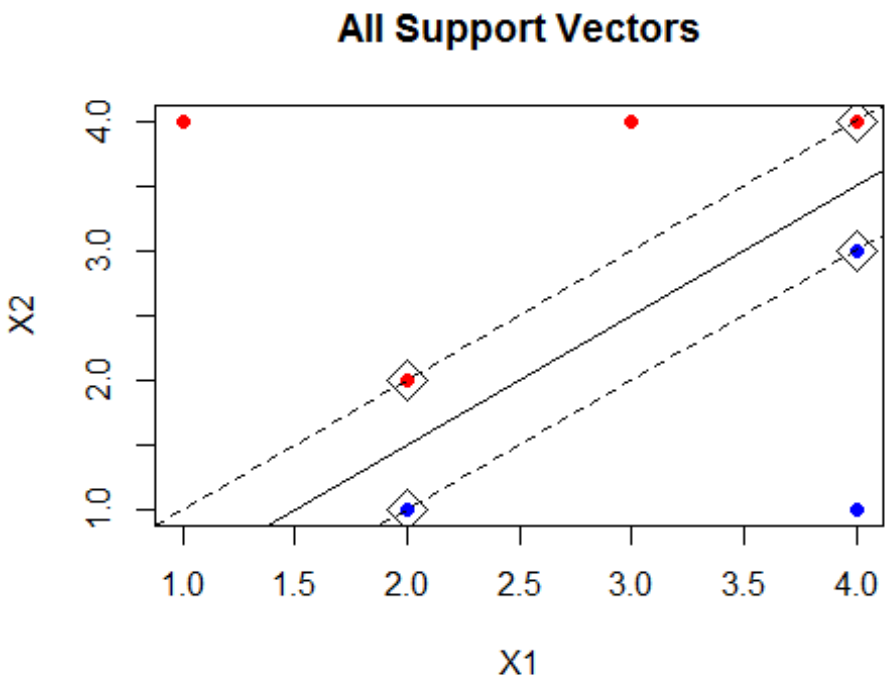
```
# Replot, this time indicating the support vectors for the maximal margin classifier.
plot(X1, X2, col=Y, pch=19, data=mydf, main="svm() Assigned Support Vectors")
abline(beta0/beta[2], -beta[1]/beta[2])
abline((beta0 - 1)/beta[2], -beta[1]/beta[2], lty = 2)
abline((beta0 + 1)/beta[2], -beta[1]/beta[2], lty = 2)
points(mydf[fit.svm$index,], pch = 5, cex = 2)
```



Note the svm fit did not fit the fifth observation (2,1) as a support vector, even though it is on the margin just as the three chosen support vectors. The definition of a support vector is : "A point is a support point if an arbitrarily small perturbation of that point may change the resulting maximal margin hyperplane.". Note that the above definition does not require the hyperplane to change under all small perturbations of that point, only that it changes for certain specific perturbations. Note also that under this definition, the determination of support points by computer software may be subject to floating point accuracy issues, as a point may be a support point under this definition but if we move the point by just a hair (caused e.g. by floating point rounding) then it might no longer be a support point.

Given this I would also mark the fourth point as a support vector as seen below.

```
# Replot, this time indicating all support vectors.
plot(X1, X2, col=Y, pch=19, data=mydf, main="All Support Vectors")
abline(beta0/beta[2], -beta[1]/beta[2])
abline((beta0 - 1)/beta[2], -beta[1]/beta[2], lty = 2)
abline((beta0 + 1)/beta[2], -beta[1]/beta[2], lty = 2)
points(mydf[fit.svm$index,], pch = 5, cex = 2)
points(2,1, pch = 5, cex = 2)
```



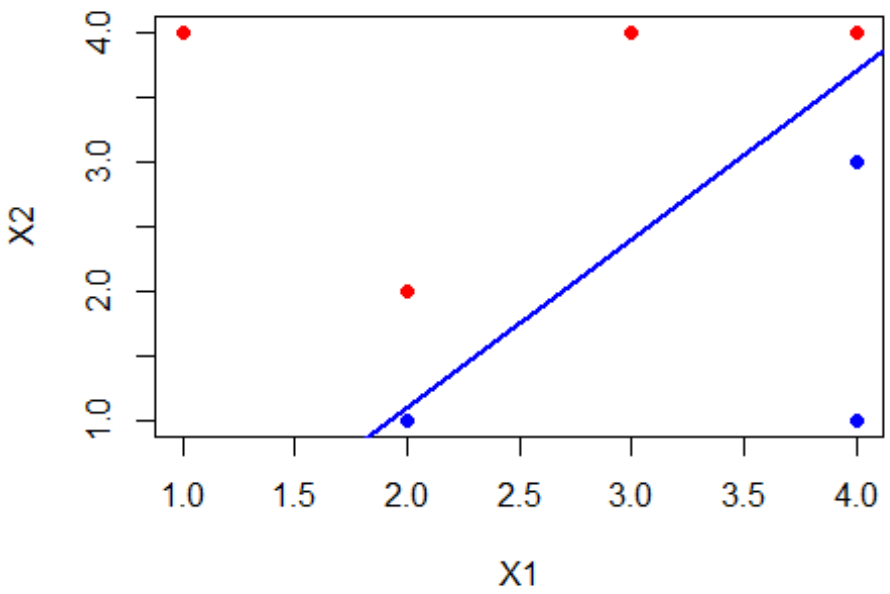
(f) Would a slight movement of the seventh observation affect the maximal margin hyperplane? Why or why not?

The seventh point is (4,1). As this point is not a support vector and is quite far from the margins of the separating hyperplane with maximum margin, a slight movement of this would not affect the maximal margin hyperplane.

(g) Sketch a hyperplane that is not the optimal separating hyperplane, and provide the equation for this hyperplane.

Here we draw separating hyperplane which is not the optimal separating hyberplane.

```
# Replot, this time with random separating hyperplane.
plot(X1, X2, col=Y, pch=19, data=mydf)
abline(-1.5, 1.3, col="blue",lwd=2)
```



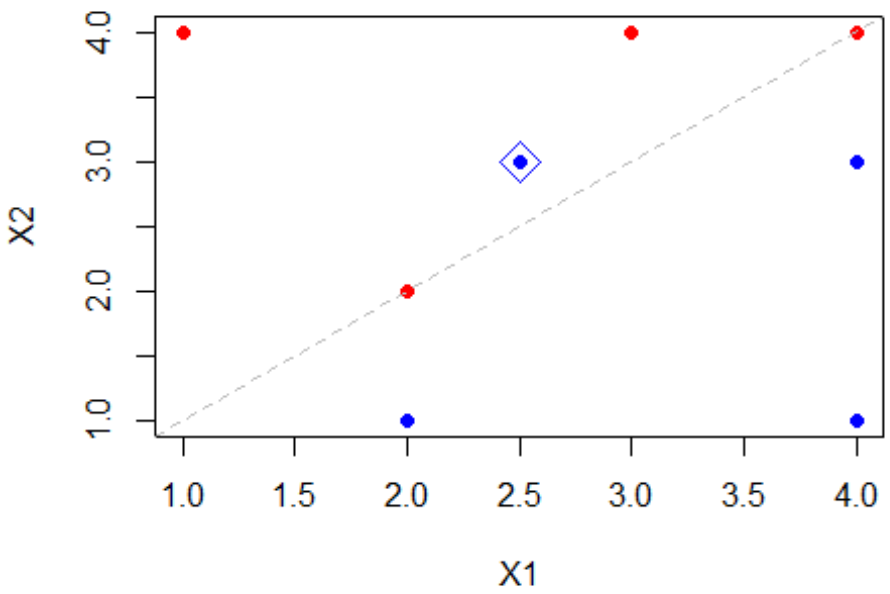
The equation for this is **-1.5 + 1.3 * X1 - X2 = 0**.

$$\beta_0 = -1.5, \beta_1 = 1.3, \beta_2 = -1$$

(h) Draw an additional observation on the plot so that the two classes are no longer separable by a hyperplane.

Plot the original data again, and add an extra point which prevents the two classes from being separable by a hyperplane. We then draw a line which holds all reds to one side of it, using the tightest fit to the red classes. Even with this we cannot avoid this new blue point from being on the red side. Therefore the below points are no longer separable by a linear hyperplane.

```
# Replot, this time indicating the support vectors for the maximal margin classifier.
plot(X1, X2, col=Y, pch=19, data=mydf)
points(2.5, 3, col="blue", pch=19)
points(2.5, 3, col="blue", pch=5, cex=2)
abline(0,1, col="grey", lty=2)
```


X2
1.0
2.0
3.0
4.0
1.0
1.5
2.0
2.5
3.0
3.5
4.0
X1