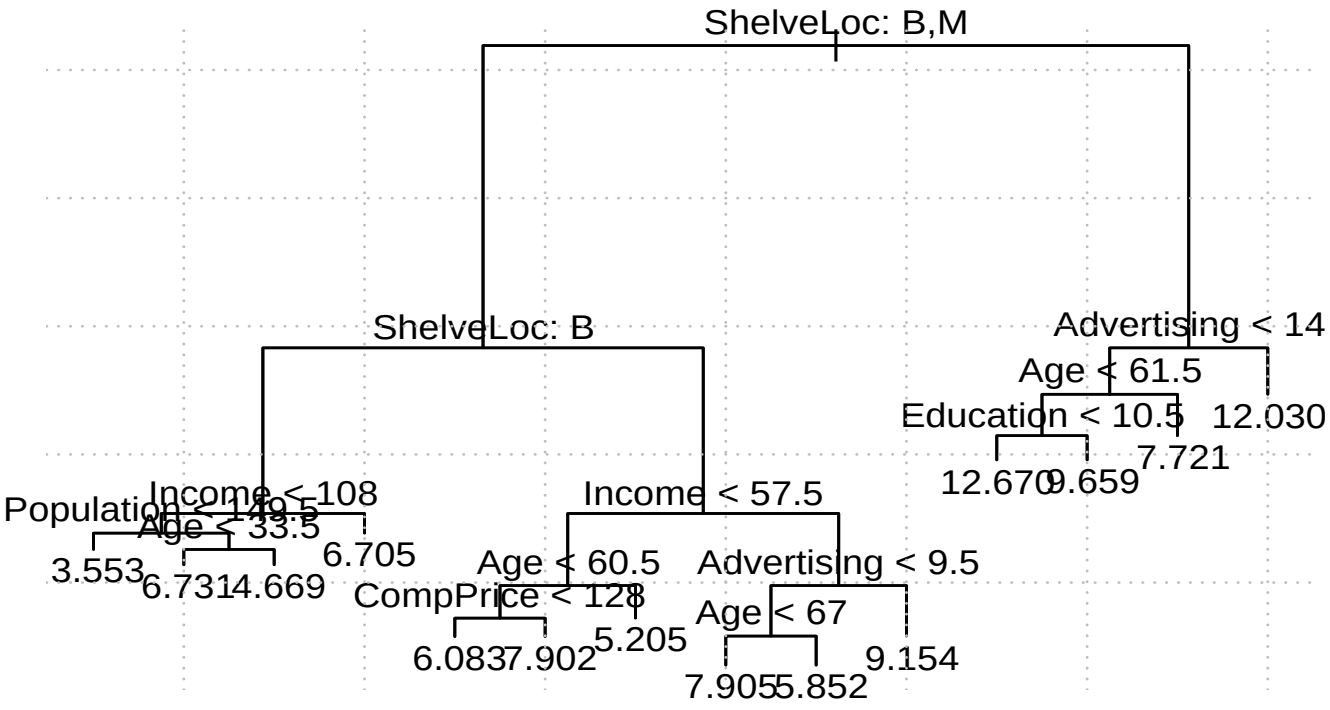
Daniel Rance
49144736
dra124

STAT462: Data Mining
Assignment 3

1.

a. Fit a regression tree to the training set (do not prune the tree). Plot the tree and interpret the results. What are the test and training MSEs for your tree?

```
library(tree)
library(ISLR)
library(rpart)
library(MASS)
library(devEMF)
carseats_train = read.csv("carseatsTrain.csv",header=T,na.strings="?")
carseats_test = read.csv("carseatsTest.csv",header =T,na.strings="?")
xtrain <- as.factor(carseats_train$ShelveLoc)
carseats_train$ShelveLoc <- xtrain
xtest <- as.factor(carseats_test$ShelveLoc)
carseats_test$ShelveLoc <- xtest
attach(carseats_train)
train_tree = tree(Sales~., carseats_train)
summary(train_tree)

## Regression tree:
## tree(formula = Sales ~ ., data = carseats_train)
## Variables actually used in tree construction:
## [1] "ShelveLoc"   "Income"      "Population"  "Age"         "CompPrice"
## [6] "Advertising" "Education"
## Number of terminal nodes:  14
## Residual mean deviance:  3.717 = 936.6 / 252
## Distribution of residuals:
##    Min. 1st Qu.  Median    Mean 3rd Qu.    Max.
## -5.2050 -1.3610 -0.2297  0.0000  1.3120  5.4550
plot(train_tree)
text(train_tree, pretty = 1)
```



Here we can see that R has elected to use seven out of a possible nine predictor variables to best predict our Y variable, *Sales*. These variables were Shelveloc (factorized into B = Bad, M = Medium, G = Good), Income, Population, Age, Comp Price, Advertising and Education. The predictor variables not used were Urban and US.

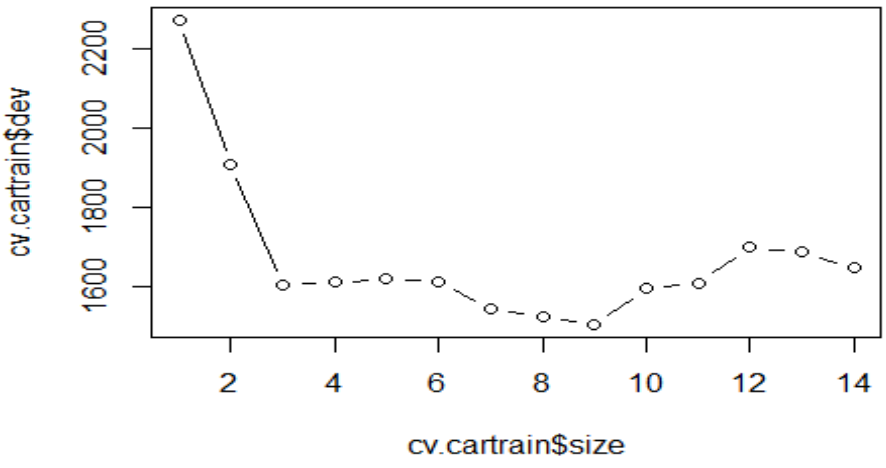
The training and testing MSE for our initial, unpruned tree can be found using the following R Code:

```
yhat_test=predict(train_tree, newdata = carseats_test)
mean((yhat_test-carseats_test$Sales)^2)

## [1] 5.163211

yhat_train=predict(train_tree, data = carseats_train)
mean((yhat_train-carseats_train$Sales)^2)

## [1] 3.521092

Where 5.163 and 3.521 are our testing and training MSE, respectively.
```

b. Use the cv.tree() R function to prune your tree. Does the pruned tree perform better?

The optimal (or near optimal) level of pruning can be determined by using the cv.tree() function:

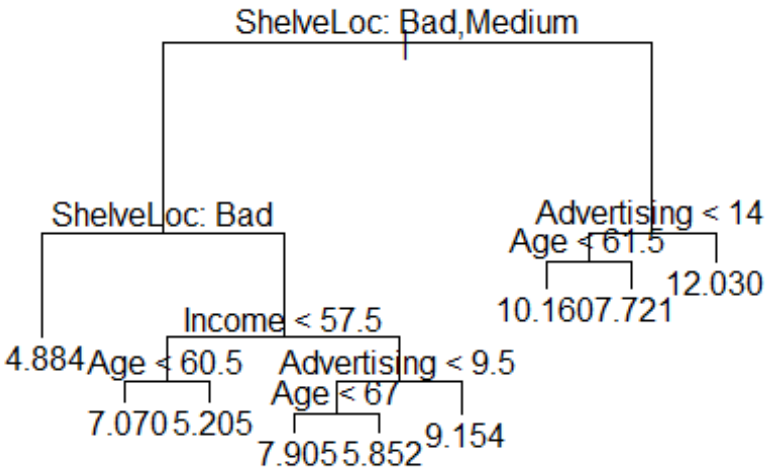
```
cv.cartrain = cv.tree(train_tree)
plot(cv.cartrain$size, cv.cartrain$dev, type = 'b')
```



From this graph we can ascertain that the most suitable level of tree complexity would be 9, as it sits at the bottom of our characteristic U shape, i.e the level of complexity that minimizes tree error. We could also elect to use a tree complexity of 3 as this also reduces error to a comparable level whilst being much less complex. Using a complexity of 9, we can prune our tree with the following code:

```
prune_cartrain <- prune.tree(train_tree, best = 9)
plot(prune_cartrain)
text(prune_cartrain, pretty = 0)
```

Daniel Rance
49144736
dra124



We can calculate the testing and training MSE for our new, pruned tree model using the following R code:

```
yhat_test_prune=predict(prune_cartrain, newdata = carseats_test)
mean((yhat_test_prune-carseats_test$Sales)^2)
```

```
## [1] 5.227227
```

```
yhat_train_prune=predict(prune_cartrain, data = carseats_train)
mean((yhat_train_prune-carseats_train$Sales)^2)
```

```
## [1] 4.072688
```

Where **5.228** and **4.073** are our testing and training MSEs respectively.

If we were to solely analyse and compare the testing MSEs for our original and pruned graph, we would probably determine it to not necessarily be an improvement. However, we must also consider the increased level of generality that our pruned graph provides. With a slight (≈0.1) increase to the testing MSE, the pruned tree sacrifices a small amount of accuracy on the training data, in order to gain more generality and reduce the level of overfitting present in a fully formed tree. This increased level of generality allows our model to perform better against unseen datasets.

c. Fit a bagged regression tree and a random forest to the training set. What are the test and training MSEs for each model? Was decorrelating trees an effective strategy for this problem?

```
#fitting a bagged regression tree
library(randomForest)
set.seed(1)
bagged_cartrain = randomForest(Sales~.,data=carseats_train, mtry = 9, impor
tance=TRUE, ntree = 700)
yhat_train_bag = predict(bagged_cartrain,data=carseats_train)
yhat_test_bag = predict(bagged_cartrain, newdata=carseats_test)

mean((yhat_train_bag-carseats_train$Sales)^2)
```

```
## [1] 5.446159
```

```
mean((yhat_test_bag-carseats_test$Sales)^2)

## [1] 4.993332

#fitting a random forest to the training set
set.seed(1)
rf_cartrain = randomForest(Sales~.,data=carseats_train, mtry = 3, importanc
e=TRUE, ntree = 700)
yhat_test_rf = predict(rf_cartrain,newdata=carseats_test)
yhat_train_rf = predict(rf_cartrain,data=carseats_train)
mean((yhat_train_rf-carseats_train$Sales)^2)

## [1] 5.522785

mean((yhat_test_rf-carseats_test$Sales)^2)

## [1] 5.025095
```

The above R code first fits a bagged regression tree to the training set and calculates the MSE for both the training and testing set. These errors are **5.446** for the training set and **4.993** for the testing set. The code then fits a random forest model to the training set and calculates the trainings and testing MSE. The errors are **5.522** for the training set and **5.025** f or the testing set.

Both the bagged regression and random forest models provided us with a decrease in the testing MSE present in our previously pruned regression tree (**5.228**). Because of this, we would determine that decorrelating trees was an effective strategy for this problem as it lowers the residual error when dealing with unseen data sets.

d. Fit a boosted regression tree to the training set. Experiment with different tree depths, shrinkage parameters and the number of trees. What are the test and training MSEs for for your best tree? Comment on your results.

```
library(gbm)
set.seed(1)
boost_cartrain=gbm(Sales~.,data=carseats_train,distribution="gaussian",n.tr
ees=5000,interaction.depth=4)
yhat_train_boost = predict(boost_cartrain,data=carseats_train, n.trees=5000
)
mean((yhat_train_boost-carseats_train$Sales)^2)

## [1] 1.453213e-06

yhat_test_boost=predict(boost_cartrain,newdata=carseats_test,n.trees=5000)
mean((yhat_test_boost-carseats_test$Sales)^2)

## [1] 6.993629
```

With the initial settings of n = 5000 and d = 4, there appears to be significant overfitting to our training data (very small MSE value). Also, the testing MSE is high and not competitive against some of the other testing MSEs we have seen so far in this assignment.

```
set.seed(1)
boost_cartrain=gbm(Sales~.,data=carseats_train,distribution="gaussian",n.tr
ees=500,interaction.depth=4)
yhat_train_boost = predict(boost_cartrain,data=carseats_train, n.trees=500)
mean((yhat_train_boost-carseats_train$Sales)^2)

## [1] 0.3800129
```

```
yhat_test_boost=predict(boost_cartrain,newdata=carseats_test,n.trees=500)
mean((yhat_test_boost-carseats_test$Sales)^2)

## [1] 6.224819
```

In an attempt to combat the significant overfitting found in the previous model, here I reduced *n* to 10% of its original size, whilst keeping *d* = 4. This appears to have been effective at both reducing the overfitting present on the training set (slightly larger training MSE) and reducing the overall testing error (reduction in testing MSE). Training MSEs for boosted regression trees are typically quite low, but I still think there is some improvement to be made in the testing MSE.

```
set.seed(1)
boost_cartrain=gbm(Sales~.,data=carseats_train,distribution="gaussian",n.trees=500,interaction.depth=6)
yhat_train_boost = predict(boost_cartrain,data=carseats_train, n.trees=500)
mean((yhat_train_boost-carseats_train$Sales)^2)

## [1] 0.1089998

yhat_test_boost=predict(boost_cartrain,newdata=carseats_test,n.trees=500)
mean((yhat_test_boost-carseats_test$Sales)^2)

## [1] 5.841883
```

By keeping *n* = 500 and changing *d = 6*, we again see quite a significant reduction to our testing MSE (which is good) but we have also experienced a drop to our training MSE (which is not necessarily good considering its already exceptionally low level).

```
set.seed(1)
boost_cartrain=gbm(Sales~.,data=carseats_train,distribution="gaussian",n.trees=1000,interaction.depth=6)
yhat_train_boost = predict(boost_cartrain,data=carseats_train, n.trees=1000)
mean((yhat_train_boost-carseats_train$Sales)^2)

## [1] 0.007055444

yhat_test_boost=predict(boost_cartrain,newdata=carseats_test,n.trees=1000)
mean((yhat_test_boost-carseats_test$Sales)^2)

## [1] 6.15064
```

I was potentially that a little overzealous in my initial culling of n = 5000 to n = 500. To check this, I fit a new model with *n* = 1000, while keeping *d* = 6. This model performed considerably worse on both measured metrics (more overfitting on the training data + more error on the testing data) so I have elected to keep *n* = 500 and *d = 6* as the parameters for my best boosted regression tree.

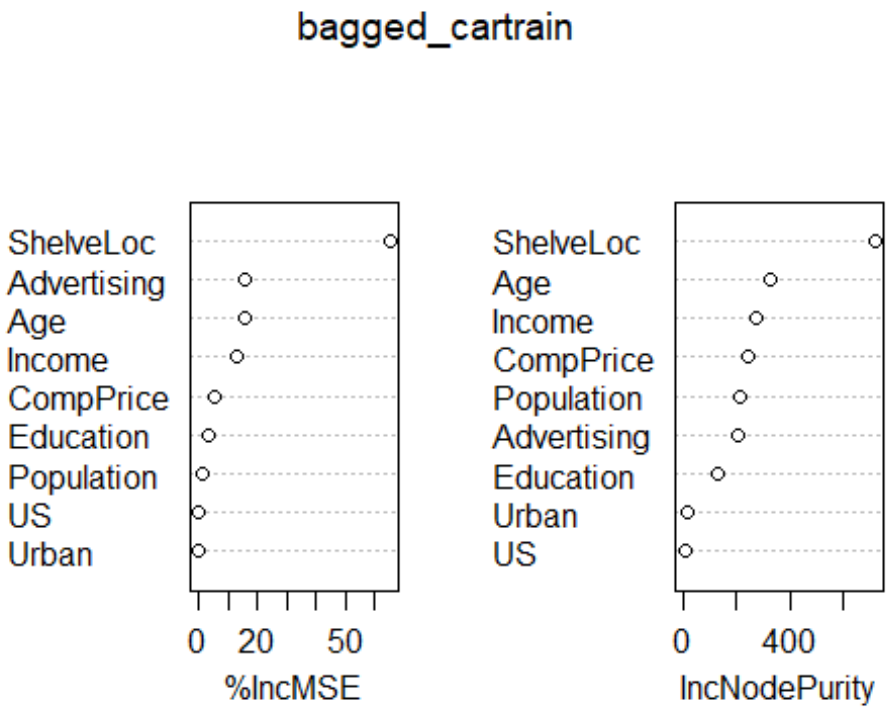
The training and testing MSEs for my best boosted tree were **0.109** and **5.842** respectively.

e. Which model performed best and which predictors were the most important in this model?

Out of all the fitted models that were tested in this question, the **bagged regression tree** performed the best as it is the lowest testing MSE (**4.993**) while its training error (**5.446**) remained at a reasonable level, suggesting we are not overfitting our training data to an

excessive extent. We can find which predictors were the most important in this model using the following R code:

```
importance(bagged_cartrain)

##                %IncMSE IncNodePurity
## CompPrice    4.9497030     245.52092
## Income      13.2115925     274.41888
## Advertising 15.8070100     210.51739
## Population   0.8617321     216.03029
## ShelveLoc   65.7330237     724.45335
## Age         15.6669862     324.87620
## Education    3.3303730     132.05246
## Urban       -0.5837369      18.69156
## US          -0.5114458      14.14830

varImpPlot(bagged_cartrain)
```



Here we can determine that ShelveLoc was our most important predictor variable, with an associated IncMSE value of 65.733 and a IncNodePurtiy value of 724.45335. We can look at a more visual representation of predictor importance using the VarImpPlot() function (pictured above). While ShelveLoc was far and away our most important predictor variable, the variables of Advertising, Age and Income were also relatively important in building the bagged regression model.

Daniel Rance
49144736
dra124

2.

a. Label each node in the itemset with the following letter(s): M, C, F, I for maximal frequent, closed frequent, frequent, and independent item sets, respectively.

QUESTION 2a WILL BE PROVIDED AS A SEPARATE PAGE AT THE BOTTOM OF THIS DOCUMENT.

Please let me know if the quality/legibility is not sufficient, I am happy to reupload in a digital format for readability.

b. Find the confidence and lift for the rule {d, e} -> {b}. Comment on what you find.

The **confidence** for a rule can be found with the following equation:

$$\mathrm{C}(\{a,b\} \rightarrow \{c\}) = \frac{\sigma(\{a,b,c\})}{\sigma(\{a,b\})}$$

Therefore,

$$\mathrm{C}(\{d,e\} \rightarrow \{b\} = \frac{\sigma(\{b,d,e\})}{\sigma(\{d,e\})}$$

and with support values of 0.3 and 0.5,

$$\mathrm{C}(\{d,e\} \rightarrow \{b\} = \frac{0.3}{0.5} = \mathbf{0.6}$$

By receiving a confidence value of 0.6 (and by using a retail store context) we can say that when items {d, e} are purchased, 60% of the time item {b} was also purchased.

The **lift** for a rule can be found with the following equation:

$$L(X,Y) = \frac{C(X \rightarrow Y)}{S(Y)}$$

Therefore,

$$L(\{d,e\},\{b\}) = \frac{C(\{d,e\} \rightarrow \{b\})}{S(\{b\})}$$

with a confidence value of 0.6 and a support of 0.6,

$$L(\{d,e\},\{b\}) = \frac{0.6}{0.6} = \mathbf{1}$$

Daniel Rance
49144736
dra124

Lift has a range of -1:1, so 1 is the highest positive correlation possible. This means that any changes in the state of itemset {d, e} will be reflected in itemset {b} i.e if we have an increase of sales of items {d, e} together, we would **strongly** suspect to see an increase of items {b} as well.

3. From A3data2, use x1 and x2 for clustering.

a. Perform k-means clustering using k = 3. Plot the clustering using a different colour for each cluster.

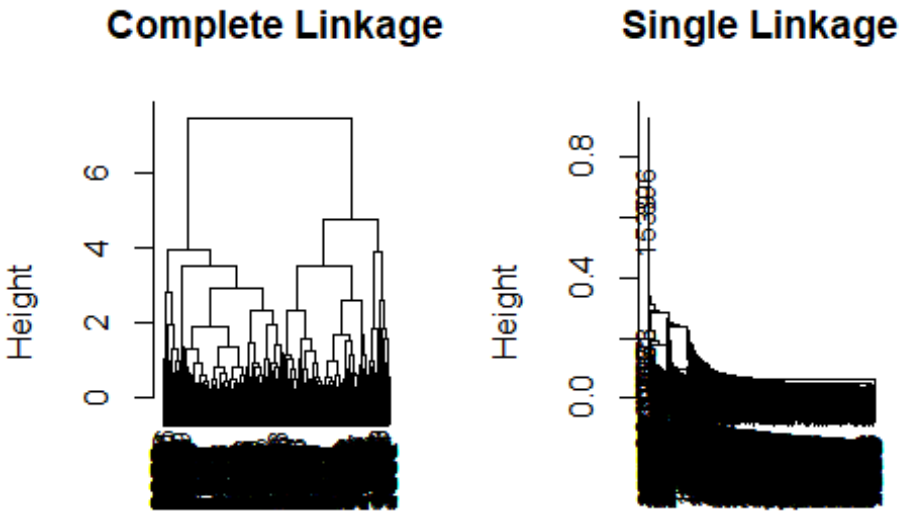
```
cluster_set = read.csv("A3data2.csv",header =T,na.strings="?")
cluster_x1x2 = cluster_set[,c("x1","x2")]

set.seed(1)
km.out = kmeans(cluster_x1x2, 3, nstart = 20)
#km.out$cluster
plot(cluster_x1x2, col = (km.out$cluster  +1),
     main = "K-Means Clustering Results with K=3",
     xlab = "", ylab = "", pch = 20, cex = 2, asp = 1)
```

b. Using hierarchical clustering with complete linkage and Euclidean distance, cluster the data, provide a dendrogram and plot the clustering with 3 clusters. Repeat using single linkage.

```
hc.complete = hclust(dist(cluster_x1x2), method = "complete")
hc.single   = hclust(dist(cluster_x1x2), method = "single")
par(mfrow = c(1, 2))
```

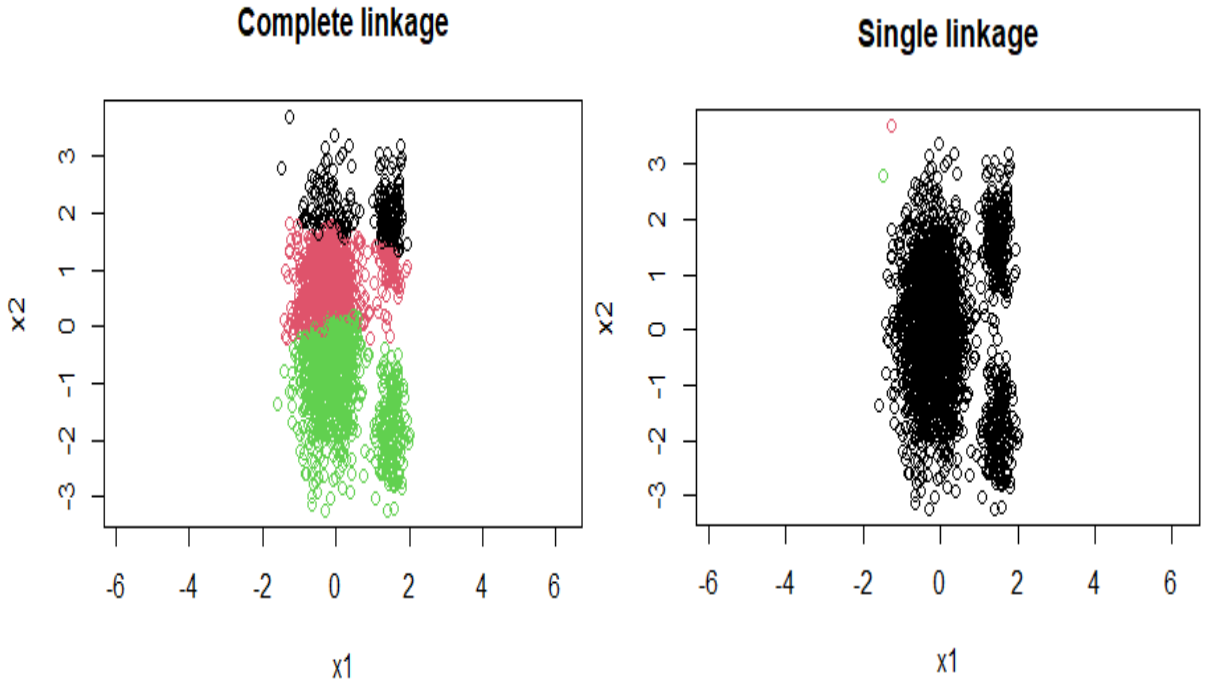
Daniel Rance
49144736
dra124

```
plot(hc.complete, main = "Complete Linkage", xlab = "", sub =
"", cex = .9)
plot(hc.single,   main = "Single Linkage",   xlab = "", sub =
"", cex = .9)
par(mfrow = c(1, 1))
```




```
plot(cluster_x1x2, main = "Complete linkage", col =
cutree(hc.complete, 3), asp = 1)
```

```
plot(cluster_x1x2, main = "Single linkage", col = cutree(hc.single,
3), asp = 1)
```



Daniel Rance
49144736
dra124

c. Comment on your results from parts (a) and (b). Provide possible explanations for each clustering obtained.

The printouts for both dendrograms are incredibly dense, to the point of illegibility. This is due to the scale of our initial dataset, which has 2400 objects. Potentially, if we scale our data more appropriately, we may be able to obtain some more legibility in our dendrograms. The clustering (k = 3) differed greatly for our dataset depending on the linkage used.

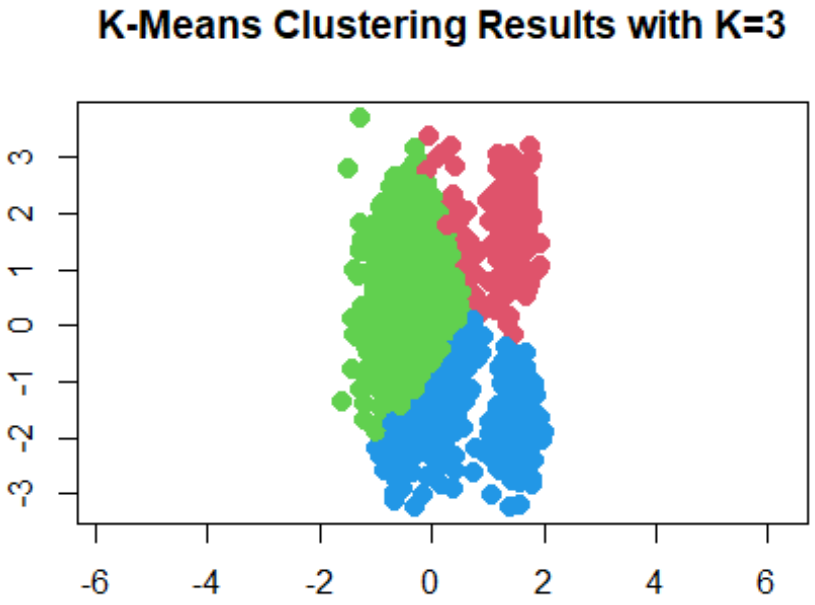
For **complete** linkage, we can see three fairly evenly spaced clusters. Intuitively, this does not appear to be an optimal clustering of our data. Most likely we would prefer the clusters to be assigned to the bottom right grouping, top right grouping and centre left grouping. Complete linkage tends to produce compact clusters with relatively small maximum distances between points in the same cluster. Again, looking intuitively, we can see that the conditions of the complete linkage model would not be satisfied if the aforementioned optimal clustering were implemented as objects at the bottom and top of the centre left cluster would be very far apart from each other - therefore not satisfying complete linkage's condition of minimizing distance within the same cluster. Because of this, we have three clusters that are quite close to each other, with some objects being closer to points in other clusters than their own.

For **single** linkage, the clustering is quite obviously non-optimal. The two outliers in the top left corner of the graph have been assigned individual clusters, and all other objects in the dataset have been assigned to the third cluster. Single linkage aims to find the minimum distance between any two observations in two different clusters. With single linkage's low threshold for merging (where points can be added one-by-one), it is clear that the two outliers in the top left corner have been added individually, and not other potential clustering satisfied the condition for distance between clusters to be minimized. Looking at this graph, we can understand why single linkage has chosen this clustering, as no other points in the graph could maximise the distance between clusters like the two objects in the two left corner can.

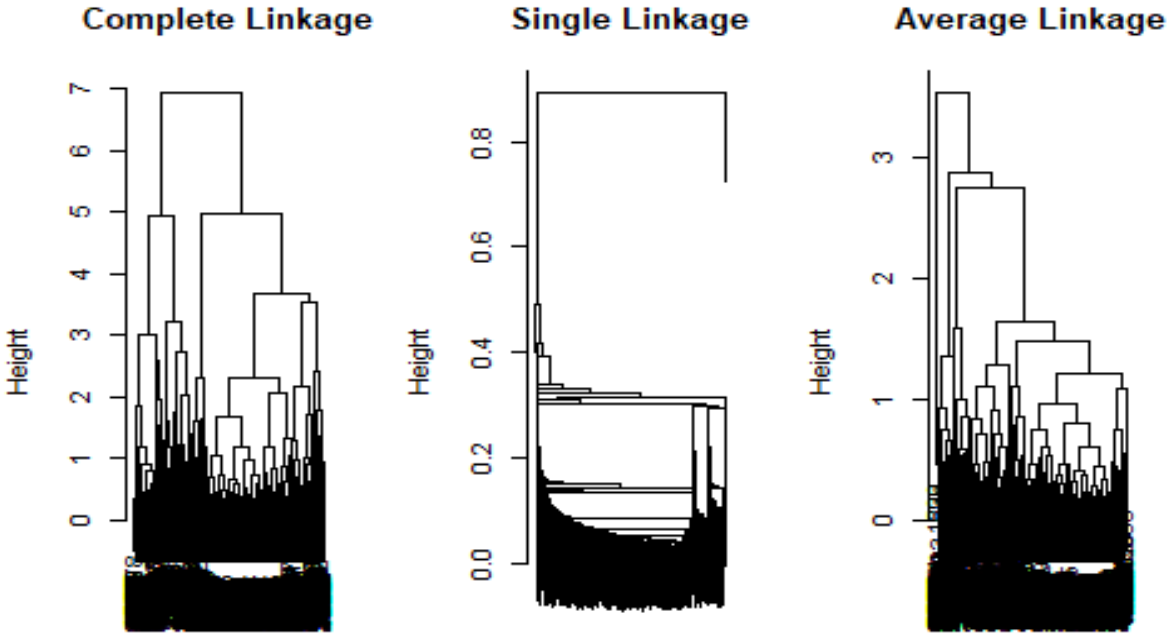
d. Rescale your data using the R function scale(Data, centre=True, scale=True) and repeat parts (a) and (b). Does rescaling improve your results? Comment on your results and provide possible explanations for each clustering obtained.

```
cluster_set = read.csv("A3data2.csv",header =T,na.strings="?")
cluster_x1x2 = cluster_set[,c("x1","x2")]

scaled_data <- scale(cluster_x1x2, center=TRUE, scale=TRUE);

set.seed(1)
km.out = kmeans(scaled_data, 3, nstart = 20)
plot(cluster_x1x2, col = (km.out$cluster  +1),
     main = "K-Means Clustering Results with K=3",
     xlab = "", ylab = "", pch = 20, cex = 2, asp = 1)
```

Daniel Rance
49144736
dra124

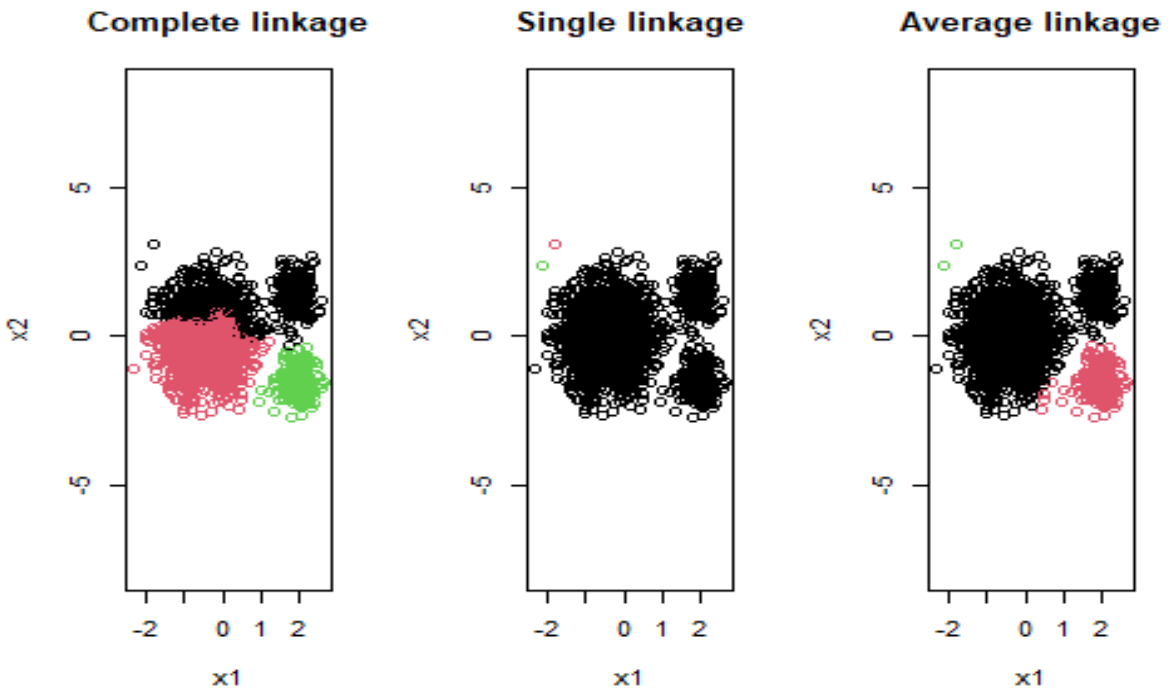



```
scaled.complete <- hclust(dist(scaled_data), method = "complet
e")
scaled.single <- hclust(dist(scaled_data), method = "single")
scaled.average <- hclust(dist(scaled_data), method = "average"
)

par(mfrow = c(1,3))
plot(scaled.complete, main = "Complete Linkage", xlab = "", su
b = "", cex = .9, lables = FALSE)

plot(scaled.single, main = "Single Linkage", xlab = "", sub =
"", cex = .9, labels = FALSE)
plot(scaled.average, main = "Average Linkage", xlab = "", sub
= "", cex = .9, lables = FALSE)
```



Daniel Rance
49144736
dra124

```
par(mfrow = c(1,3))
plot(scaled_data, main = "Complete linkage", col = cutree(scal
ed.complete, 3), asp = 1)
plot(scaled_data, main = "Single linkage", col = cutree(scaled
.single, 3), asp = 1)
plot(scaled_data, main = "Average linkage", col = cutree(scale
d.average, 3), asp = 1)
```



It appears that scaling our data has taken us closer to a more optimal clustering of our dataset. In particular, the k-means function was improved significantly. I suspect this is due to kmeans sensitivity to outliers in a data set, when we scaled the data set, the effects of the outliers on the function were weakened, leading to a more optimal clustering. We see a minor improvement in the complete linkage function, where our clustering is getting closer to identifying the 3 clusters that we intuitively see in our graph. There has no discernable impact of scaling on our clusterings when using single linkage, we still see the two single-object clusters in the top left portion of our graph while the remaining objects constitute the third cluster. For good measure, I fit a average linkage cluster as well, and this in turn seemed to perform better than the single linkage, while not performing as well as the k means clustering or the complete linkage clustering.

I believe scaling **has improved** our results, particularly for the k means and complete linkage clustering. Overall, I believe the **scaled k means function** performed the best.

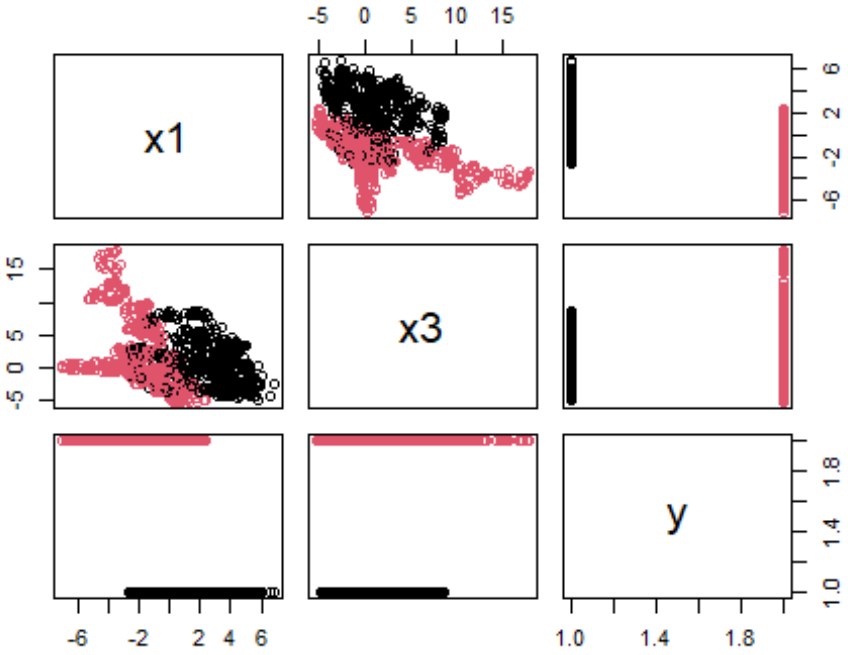
Daniel Rance
49144736
dra124

4. In this question, you will fit support vector machines to the Banknote data from Assignment 2. Only use the predictors x1 and x3 to fit your classifiers.

a. Is it possible to find a separating hyperplane for the training data? Explain.

We can answer this question by loading our dataset, setting our response variable y as a factor, and plotting the data to see if we can visually identify a potential separating hyperplane.

```
library("e1071")
library("ggplot2")

bank.train = read.csv("BankTrain.csv",header=T,na.strings="?")
bank.test = read.csv("BankTest.csv",header =T,na.strings="?")
bank.train = bank.train[,c("x1","x3", "y")]
bank.test = bank.test[,c("x1","x3","y")]
bank.train$y = factor(bank.train$y, levels = c(0,1))
bank.test$y = factor(bank.test$y, levels = c(0,1))
plot(bank.train, col = bank.train$y)
```



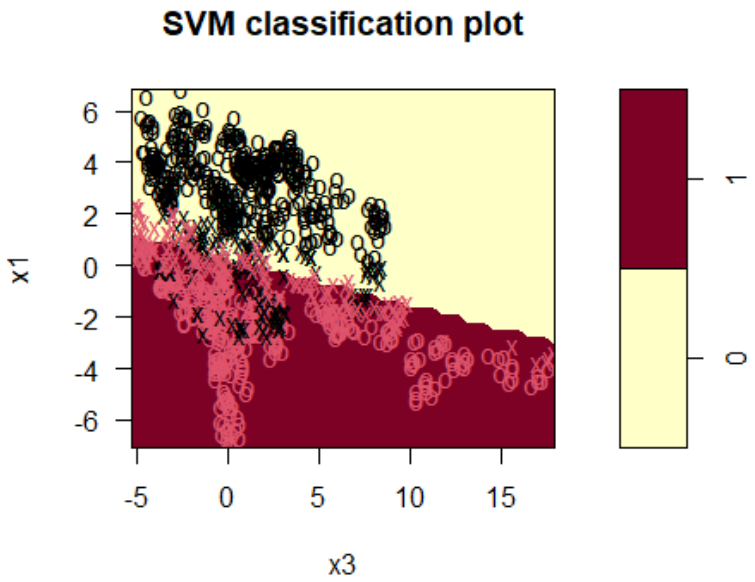
By looking at these various printouts, we can quite easily ascertain that it is **not possible** to find a separating hyperplane for the training data as there exists significant overlap between response variables 0 and 1.

Daniel Rance
49144736
dra124

b. Fit a support vector classifier to the training data using tune() to find the best cost value. Plot the best classifier and produce a confusion matrix for the testing data. Comment on your results.

Without knowing the ideal cost value for our support vector classifier, we can fit an initial model using the following code and a placeholder cost variable of 1.

```
svmfit.linear = svm(formula = y ~ .,
                    data = bank.train,
                    type = 'C-classification',
                    kernel = 'linear', cost = 0.1, scale = FALSE)
plot(svmfit.linear, bank.train)
```



We can then use the tune() function to iterate over a selection of different cost variables to find the most optimal.

```
set.seed(1)
tune.out=tune(svm ,y~.,data=bank.train ,kernel ="linear", rang
es=list(cost=c(0.001, 0.01, 0.1, 1,5,10,100)))
summary(tune.out)

##
## Parameter tuning of 'svm':
##
## - sampling method: 10-fold cross validation
##
## - best parameters:
##  cost
##   0.1
##
## - best performance: 0.1135417
##
## - Detailed performance results:
##    cost     error dispersion
## 1 1e-03 0.2229167 0.06481068
## 2 1e-02 0.1260417 0.04644219
## 3 1e-01 0.1135417 0.03487812
```

```
## 4 1e+00 0.1197917 0.04426232
## 5 5e+00 0.1208333 0.04397539
## 6 1e+01 0.1187500 0.04342361
## 7 1e+02 0.1187500 0.04342361
```

Here we can see that our best performance was associated with a cost value of 1e-01. If we do not want to browse potentially large tables to find our ideal cost value though, we can simple use the built in $best.model feature.

```
bestmod=tune.out$best.model
summary(bestmod)

##
## Call:
## best.tune(method = svm, train.x = y ~ ., data = bank.train,
ranges = list(cost = c(0.001,
##     0.01, 0.1, 1, 5, 10, 100)), kernel = "linear")
##
##
## Parameters:
##    SVM-Type:  C-classification
##  SVM-Kernel:  linear
##        cost:  0.1
##
## Number of Support Vectors:  350
##
##  ( 175 175 )
##
##
## Number of Classes:  2
##
## Levels:
##  0 1

ypred=predict (bestmod, bank.test)
table(predict =ypred , truth=bank.test$y)

##        truth
## predict   0   1
##       0 197  11
##       1  39 165
```

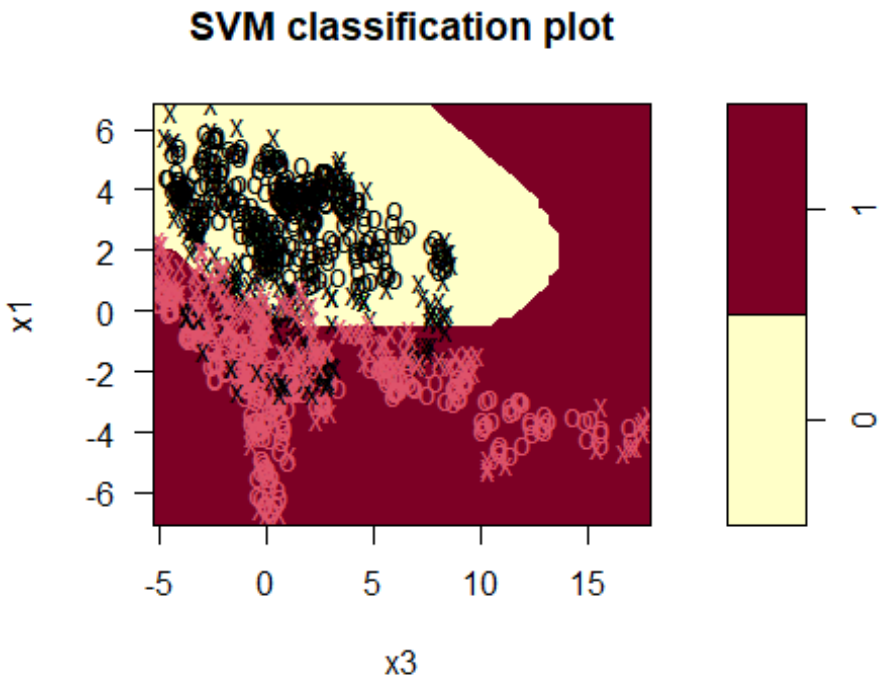
It turns out that our initial SVM Classification Plot with a cost = 0.1 was the most optimal cost we could use. We can see from the confusion matrix that, using the single kernel, the overall testing error is (39+11)/412 or **12.13%.** For true non-forged banknotes, we have an error rate of 39/204 or **19.12%.** For true forged banknotes we have an error rate of 11/176 or **6.25%.** All three of these figures are a significant improvement from the multiple logistic regression functions we used in the previous assignment.

Daniel Rance
49144736
dra124

c. Fit a support vector machine (SVM) to the training data using the radial kernel. Use tune() to find the best cost and gamma values. Plot the best SVM and produce a confusion matrix for the testing data. Compare your results with those obtained in part (b).

Again, we estimate the best cost and gamma values (0.1 and 1 respectively) for this initial fit of our radial kernel SVM.

```
svmfit.radial=svm(y~., data=bank.train, kernel ="radial", gamm
a=1,
cost=0.1)
plot(svmfit.radial , bank.train)
```




```
summary(svmfit.radial)

##
## Call:
## svm(formula = y ~ ., data = bank.train, kernel = "radial",
gamma = 1,
##     cost = 0.1)
##
##
## Parameters:
##    SVM-Type:  C-classification
##  SVM-Kernel:  radial
##        cost:  0.1
##
## Number of Support Vectors:  384
##
##  ( 190 194 )
##
```

```
##
## Number of Classes:  2
##
## Levels:
##  0 1
```

Here we use the tune() function to find the ideal cost and gamma values.

```
set.seed(1)
tune.out.radial=tune(svm , y~., data = bank.train, kernel ="ra
dial",
ranges=list(cost=c(0.1,1,10,100,1000),
gamma=c(0.5,1,2,3,4) ))
summary(tune.out.radial)
```

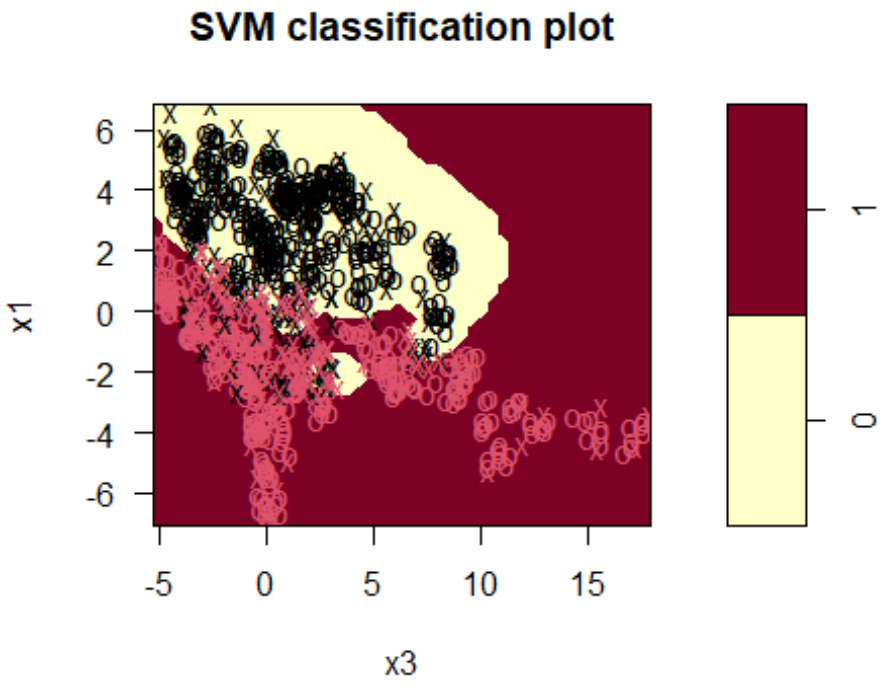
```
##
## Parameter tuning of 'svm':
##
## - sampling method: 10-fold cross validation
##
## - best parameters:
##  cost gamma
##    10     4
##
## - best performance: 0.09166667
##
## - Detailed performance results:
##     cost gamma      error dispersion
## 1  1e-01   0.5 0.10625000 0.04013384
## 2  1e+00   0.5 0.10000000 0.03746140
## 3  1e+01   0.5 0.09791667 0.03995319
## 4  1e+02   0.5 0.09895833 0.04398910
## 5  1e+03   0.5 0.10416667 0.04280843
## 6  1e-01   1.0 0.10000000 0.04143454
## 7  1e+00   1.0 0.10000000 0.04055223
## 8  1e+01   1.0 0.09791667 0.04143454
## 9  1e+02   1.0 0.09895833 0.04086320
## 10 1e+03   1.0 0.10104167 0.04555100
## 11 1e-01   2.0 0.10104167 0.04282251
## 12 1e+00   2.0 0.10000000 0.04201245
## 13 1e+01   2.0 0.09791667 0.04172450
## 14 1e+02   2.0 0.09270833 0.04485756
## 15 1e+03   2.0 0.09375000 0.04606423
## 16 1e-01   3.0 0.10000000 0.04055223
## 17 1e+00   3.0 0.09791667 0.04342361
## 18 1e+01   3.0 0.09895833 0.04586752
## 19 1e+02   3.0 0.09375000 0.04836246
## 20 1e+03   3.0 0.09479167 0.04404388
## 21 1e-01   4.0 0.10000000 0.04055223
## 22 1e+00   4.0 0.09687500 0.04254004
## 23 1e+01   4.0 0.09166667 0.05002893
## 24 1e+02   4.0 0.09270833 0.04002856
## 25 1e+03   4.0 0.09791667 0.04479032
```

```
bestmod.radial=tune.out.radial$best.model
summary(bestmod.radial)
```

```
##
## Call:
## best.tune(method = svm, train.x = y ~ ., data = bank.train,
ranges = list(cost = c(0.1,
##     1, 10, 100, 1000), gamma = c(0.5, 1, 2, 3, 4)), kernel
= "radial")
##
##
## Parameters:
##    SVM-Type:  C-classification
##  SVM-Kernel:  radial
##        cost:  10
##
## Number of Support Vectors:  257
##
##  ( 131 126 )
##
##
## Number of Classes:  2
##
## Levels:
##  0 1
```

From our tune() function, we discover that the optimal cost and gamma values are 10 and 4, respectively. Using that data, we can plot our optimal radial kernel SVM.

```
svmfit.radial=svm(y~., data=bank.train, kernel ="radial", gamm
a=4,
cost=1e+01)
plot(svmfit.radial , bank.train)
```



Daniel Rance
49144736
dra124

And then calculate the error rates with the following:

```
ypred=predict (svmfit.radial, bank.test)
table(predict =ypred , truth=bank.test$y)
```

```
##         truth
## predict   0   1
##       0 212  12
##       1  24 164
```

From this confusion matrix we can determine that our radial kernel SVM has an overall testing error of (24+12)/412 or **8.7%.** For true non-forged banknotes, we have an error rate of 24/188 or **12.76%.** For true forged banknotes we have an error rate of 12/176 or **6.81%.** Again, all three of these figures are a significant improvement from the multiple logistic regression functions we used in the previous assignment. Furthermore, barring a minor increase to the true forged banknote error, the radial kernel SVM performed better on all metrics. This is to be expected with a more complex function. Again, the exact context of our work will determine which method we would implement, depending on which error rate we intend to minimize.

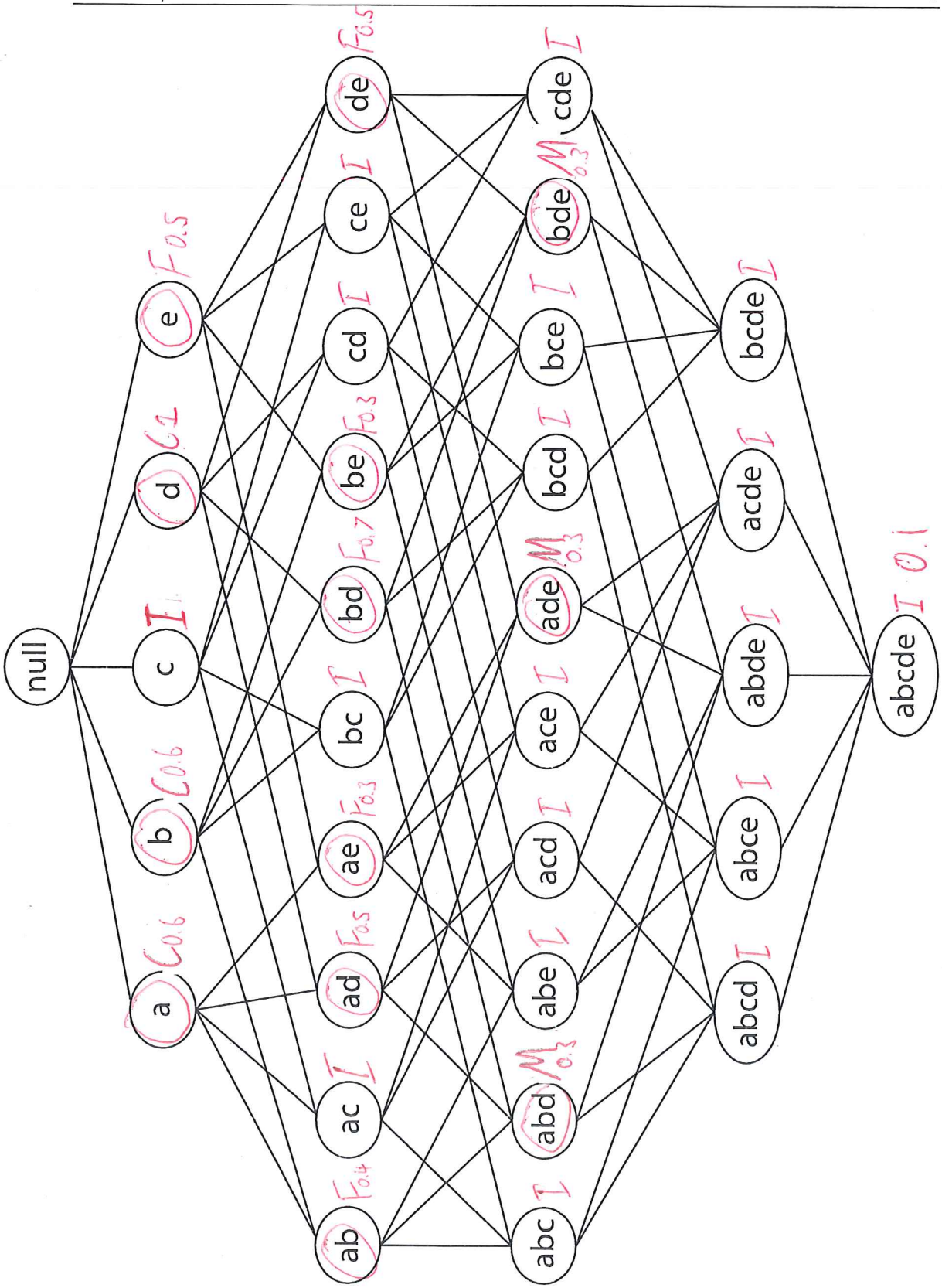


Figure 1: Itemset lattice for Question 3.