

> tree.fit = tree(Sales~., train)

> summary(tree.fit)

Regression tree:

tree(formula = Sales ~ ., data = train)

Variables actually used in tree construction:

[1] "Price" "Age" "CompPrice" "Education" "Advertising" "Income"

Number of terminal nodes: 23

Residual mean deviance: 3.296 = 913 / 277

Distribution of residuals:

Min. 1st Qu. Median Mean 3rd Qu. Max.

-4.994 -1.178 0.037 0.000 1.390 4.627

> train.pred = predict(tree.fit, train)

> train.mse = mean((train$Sales-train.pred)^2)

> train.mse

[1] 3.043394

> test.pred = predict(tree.fit, test)

> test.mse = mean((test$Sales-test.pred)^2)

> test.mse

[1] 6.03393

(a) Answer : As has been shown in the figure 1.(a) and the code as above,

There are six variables have actually been used, which are "Price", "Age", "CompPrice", "Education", "Advertising" and "Income", and the number of terminal nodes are 23.

The train error is **3.043394** and the test error is **6.03393**

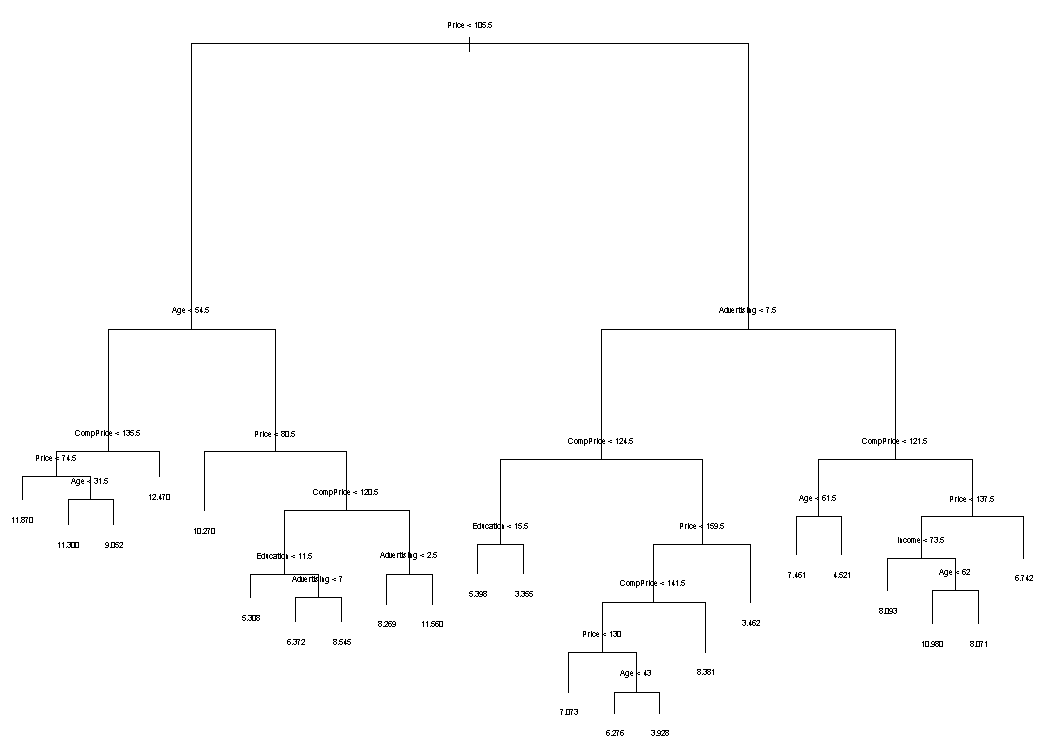


Figure 1.(a) Regression tree

(b)

> set.seed(10)

> cv.fit=cv.tree(tree.fit)

> plot(cv.fit$size, cv.fit$dev, main = "The relationship between CVdev and cvsize",type="b")

**Answer:** From Figure 1 (b), the lowest size is “11”

> prune.fit = prune.tree(tree = tree.fit, best = 11)

> plot(prune.fit)

> text(prune.fit, pretty = 0, cex = 0.5)

> #after pruning

> prune.train.pred = predict(prune.fit, train)

> prune.train.mse = mean((train$Sales-prune.train.pred)^2)

> prune.train.mse

[1] 4.38325

> prune.test.pred = predict(prune.fit, test)

> prune.test.mse = mean((test$Sales-prune.test.pred)^2)

> prune.test.mse

[1] 6.05599

However, from the code above, after pruning, the train error increase to 4.38325 and test error slightly increase to 6.05599, which suggests the pruned tree did not perform better.

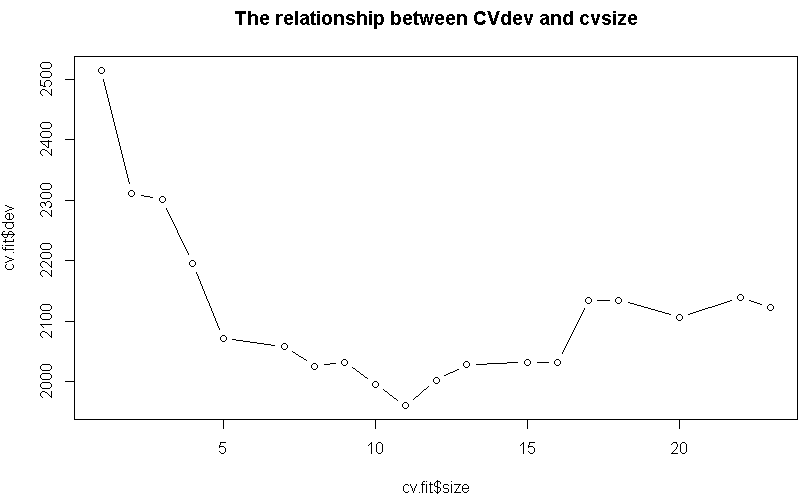


Figure 1.(b) The relationship between CVdev and cvsize

(C)

> #Bagging

> library(randomForest)

> bag.fit = randomForest(Sales~., data = train, mtry = 3 , ntree = 1000 ,importance = TRUE)

> bag.fit

Call:

randomForest(formula = Sales ~ ., data = train, mtry = 3, ntree = 1000, importance = TRUE)

Type of random forest: regression

Number of trees: 1000

No. of variables tried at each split: 3

Mean of squared residuals: 5.27523

% Var explained: 34.9

> #after bagging

> bag.train.pred = predict(bag.fit, train)

> bag.train.mse = mean((train$Sales-bag.train.pred)^2)

> bag.train.mse

[1] 1.059408

> bag.test.pred = predict(bag.fit, test)

> bag.test.mse = mean((test$Sales-bag.test.pred)^2)

> bag.test.mse

[1] 4.396091

> #randomForest

> rf.fit = randomForest(Sales~., data = train, importance = TRUE)

> rf.fit

Call:

randomForest(formula = Sales ~ ., data = train, importance = TRUE)

Type of random forest: regression

Number of trees: 500

No. of variables tried at each split: 3

Mean of squared residuals: 5.269197

% Var explained: 34.97

> #after rf

> rf.train.pred = predict(rf.fit, train)

> rf.train.mse = mean((train$Sales-rf.train.pred)^2)

> rf.train.mse

[1] 1.058377

> rf.test.pred = predict(rf.fit, test)

> rf.test.mse = mean((test$Sales-rf.test.pred)^2)

> rf.test.mse

[1] 4.426227

|  |  |  |
| --- | --- | --- |
|  | Train MSE | Test MSE |
| Bagging | 1.059408 | 4.396091 |
| RF | 1.058377 | 4.426227 |

From the code above, we can get the lower train MSE and lower test MSE as shown in the

Table, which suggests that decorrelating trees was an effective strategy for this problem.

(d)

Using n.trees = 11408, shrinkage = 0.001, we get:

> library(gbm)

> set.seed(10)

> boost.fit=gbm(Sales ~ . ,data=train, distribution="gaussian", n.trees = 20000, shrinkage = 0.001, cv.folds=5)

> best.iter = gbm.perf(boost.fit,method='cv')

> best.iter

[1] 11408

> summary(boost.fit)

var rel.inf

Price Price 39.3538183

CompPrice CompPrice 20.5270515

Age Age 14.2350753

Advertising Advertising 13.0119174

Income Income 6.9486922

Population Population 4.0732032

Education Education 1.5063011

Urban Urban 0.2056938

US US 0.1382473

> boost.fit=gbm(Sales ~ . ,data=train, distribution="gaussian", n.trees = best.iter, shrinkage = 0.001, interaction.depth = 1)

> summary(boost.fit)

var rel.inf

Price Price 43.23500051

CompPrice CompPrice 20.78566958

Advertising Advertising 14.28949436

Age Age 13.86266002

Income Income 5.41494993

Population Population 1.44733798

Education Education 0.89838120

Urban Urban 0.06650642

US US 0.00000000

> boost.train.pred = predict(boost.fit, train, n.trees=best.iter)

> boost.train.mse = mean((train$Sales-boost.train.pred)^2)

> boost.train.mse

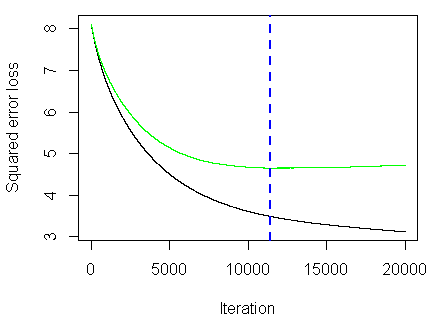
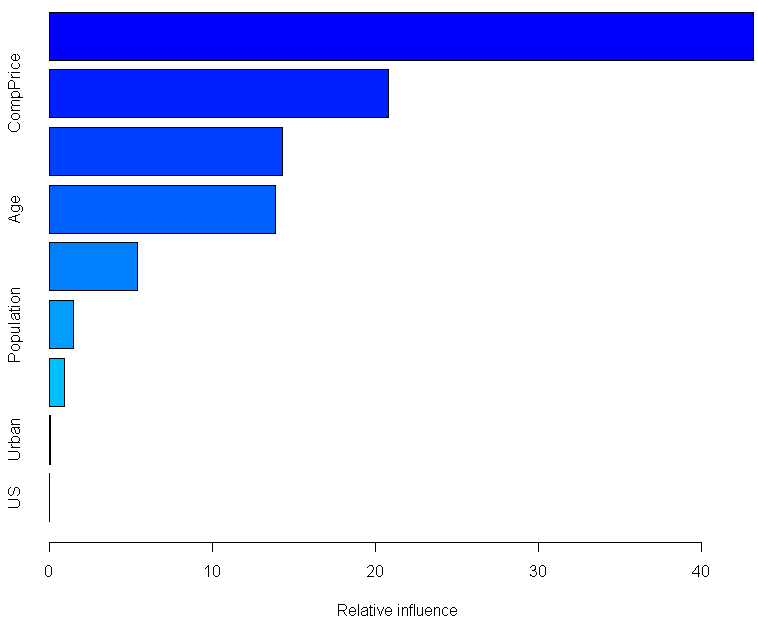
[1] 3.493034

> boost.test.pred = predict(boost.fit, test, n.trees=best.iter)

> boost.test.mse = mean((test$Sales-boost.test.pred)^2)

> boost.test.mse

[1] 3.775372

Answer :

As has shown above, we get test error is3.775372 for n.trees = 11408 and shrinkage = 0.001.

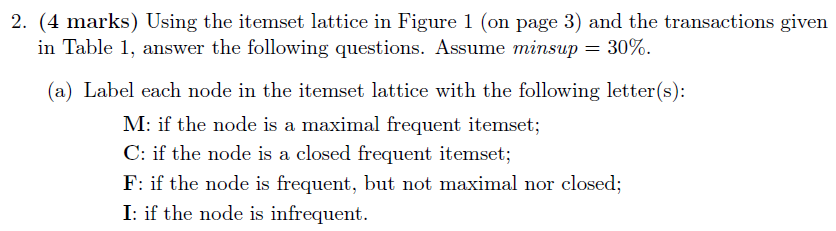
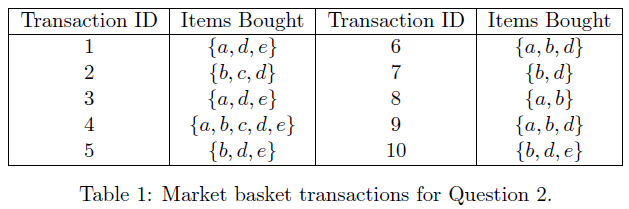
Similarly, when increasing the shrinkage, the n.trees size decreases correspondingly, and we can get different testing error as below table. Obviously, the n=trees is 1585 and shrinkage = 0.01 performs

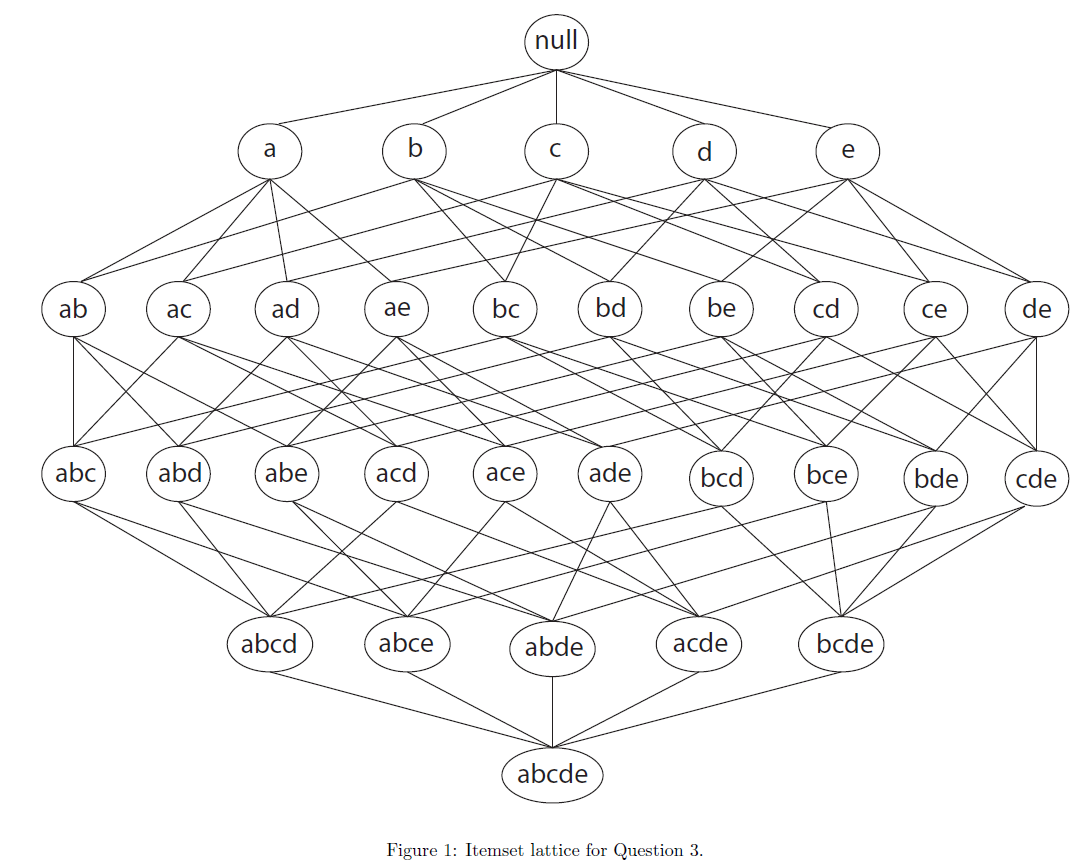
the best with the lowest testing error is 3.726715.

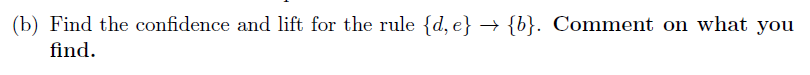
|  |  |  |  |
| --- | --- | --- | --- |
| **n.trees** | **shrinkage** | **train.mse** | **test.mse** |
| 20000(11408) | 0.001 | 3.493034 | 3.775372 |
| 5000(3295) | 0.005 | 3.245314 | 3.790217 |
| 1585 | 0.01 | 3.273168 | 3.726715 |

(e)

In conclusion, the Boosted regression tree model performs the best, and (n.trees = 11408 and shrinkage = 0.001) one with the lowest test error, which is 3.726715. And the most important predictors from the code are “Price”, “CompPrice”, “Age” and “Advertising”.

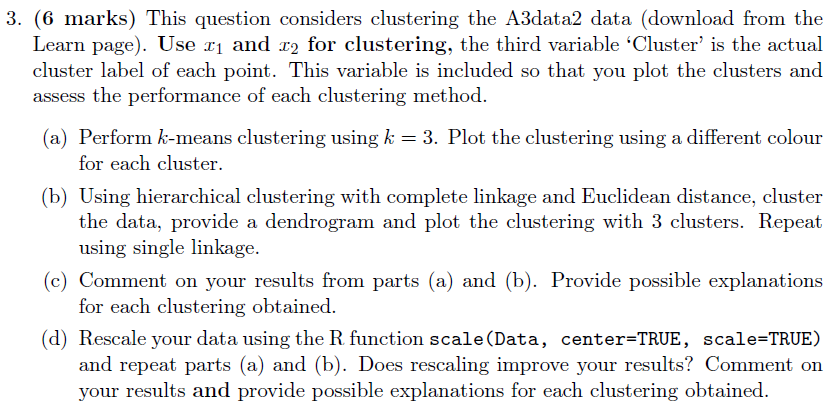


(a)



In this context, confidence {d,e} → {b} = 60% means there are 60% chance that the customer purchased item {d,e} that also purchased item {b}.

In this context, lift {d,e} → {b} = 0.75 <1, item {d,e} and item {b} are negatively correlated, which means the occurrence of item {d,e} reduces the likelihood of item {b}.



(a)

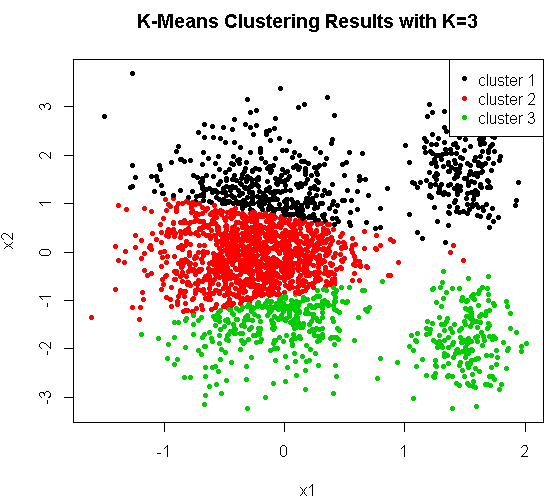
> library(cluster)

> A3data = read.csv(file = 'A3data2.csv', header = TRUE)

> km.out = kmeans(A3data[, 1:2], 3, nstart = 100)

> plot(A3data[, 1:2], col=(km.out$cluster), main="K-Means Clustering Results with K=3", pch =20)

> legend("topright", legend = c("cluster 1", "cluster 2","cluster 3"), col =c(1,2,3), pch = 20)



(b)

> #hc complete

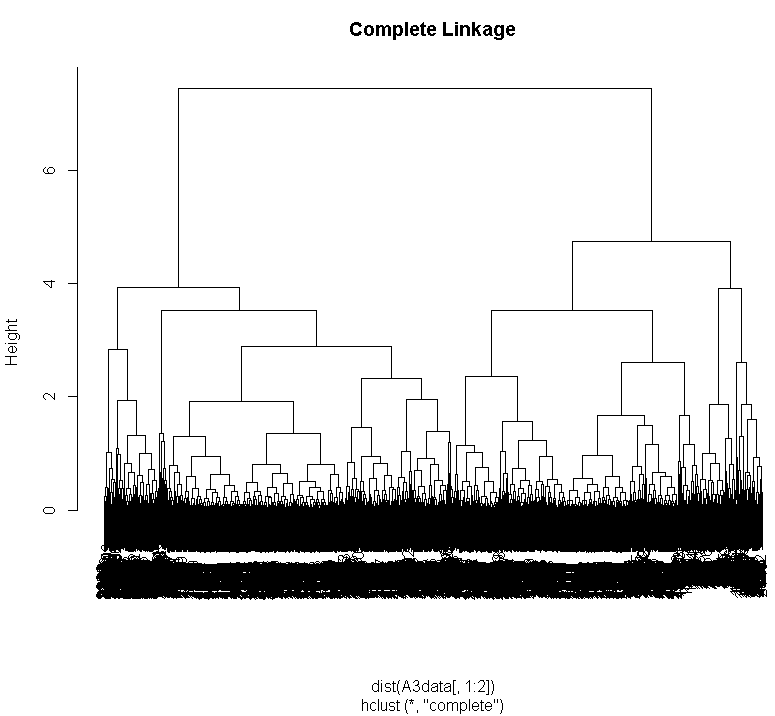
> hc.complete = hclust(dist(A3data[, 1:2]), method="complete")

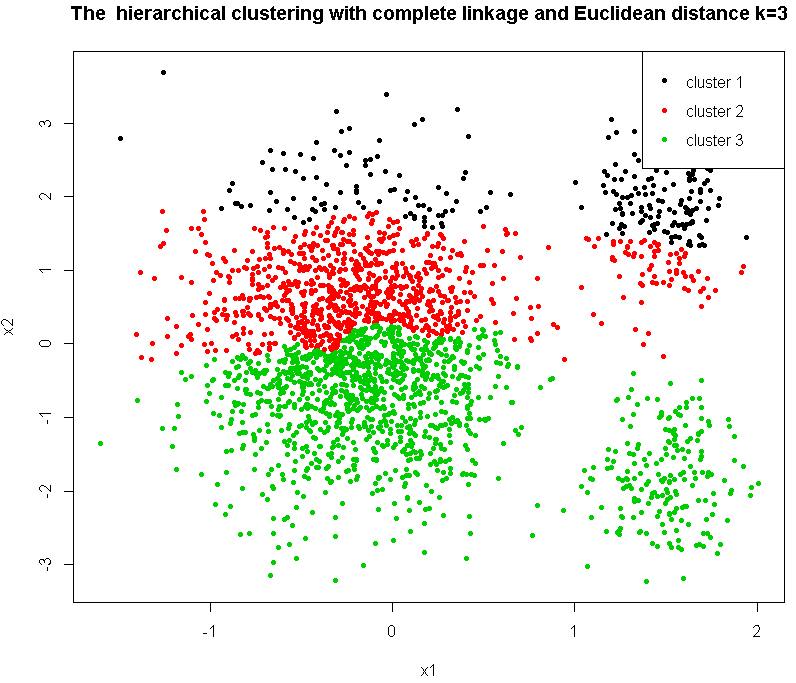
> plot(hc.complete, main = "Complete Linkage")

> plot(A3data[, 1:2], col=(cutree(hc.complete, 3) ), main="The hierarchical clustering with complete linkage and Euclidean distance k=3", pch =20)

> legend("topright", legend = c("cluster 1", "cluster 2","cluster 3"),

+ col = c(1, 2, 3), pch = 20)





> #hc single

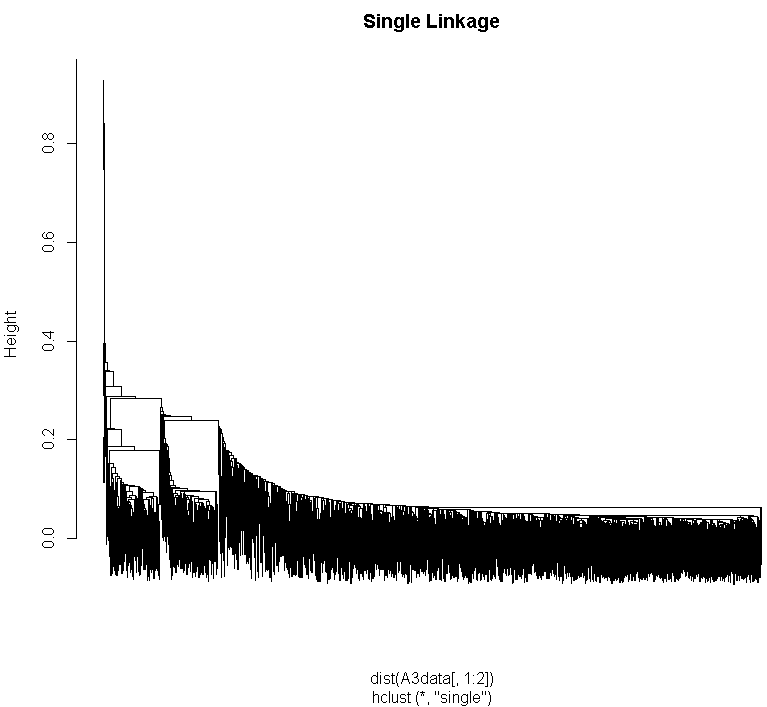
> hc.single=hclust(dist(A3data[, 1:2]),method="single")

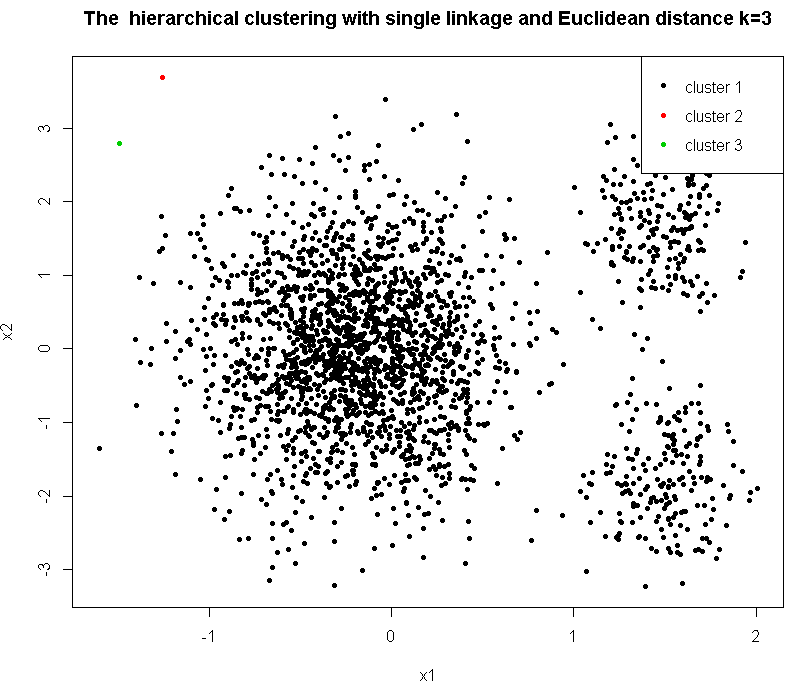
> plot(hc.single, main="Single Linkage", labels=FALSE,cex=0.5)

> plot(A3data[, 1:2], col=(cutree(hc.single, 3) ), main="The hierarchical clustering with single linkage and Euclidean distance k=3", pch =20)

> legend("topright", legend = c("cluster 1", "cluster 2","cluster 3"),

+ col = c(1, 2, 3), pch = 20)





(c)

**Answer**: From figures above, we can tell the clusters are 3 globular shape with different sizes and densities. After applying K-means model, the largest cluster was split into 3 parts wrongly even though the other two cluster seems to be better classified. After applying complete linkage model, not just the largest one was split into 3 parts, the smaller clusters were also failed to be classified. The single linkage model was totally failed to be classified due to the two outliers, it is clear that the two outliers were classified into two classes and the rest shapes become the whole class. In a word, all three models did a very poor job on this classifying.

(d)

> #rescale

> Rescale3=scale(A3data[, 1:2], center=TRUE, scale=TRUE)

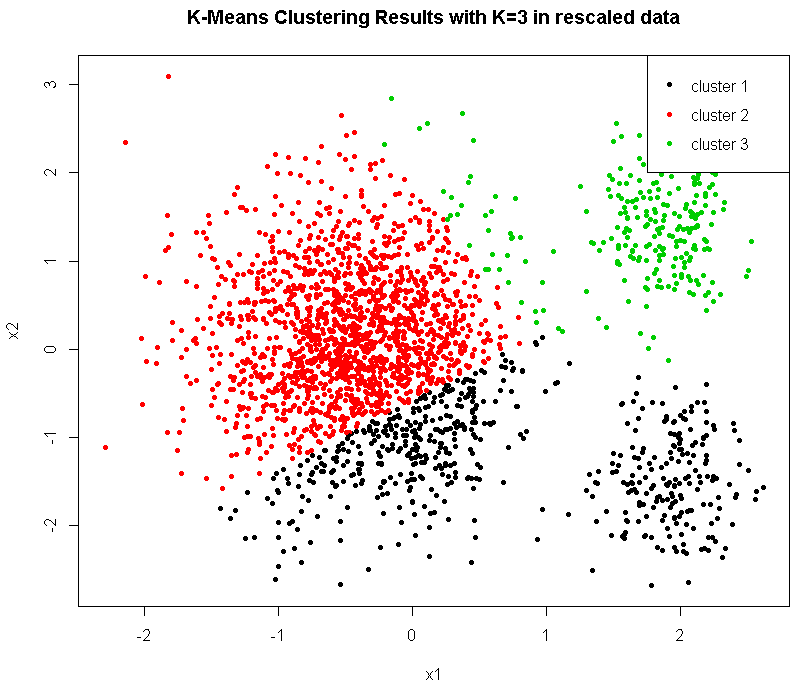
> # apply k-means

> set.seed(10)

> Rescale3.km.out=kmeans(Rescale3,3, nstart=20)

> plot(Rescale3, col = (Rescale3.km.out$cluster), main="K-Means Clustering Results with K=3 in rescaled data", pch =20)

> legend("topright", legend = c("cluster 1", "cluster 2","cluster 3"), col = c(1, 2, 3), pch = 20)



> # rescale HC complete

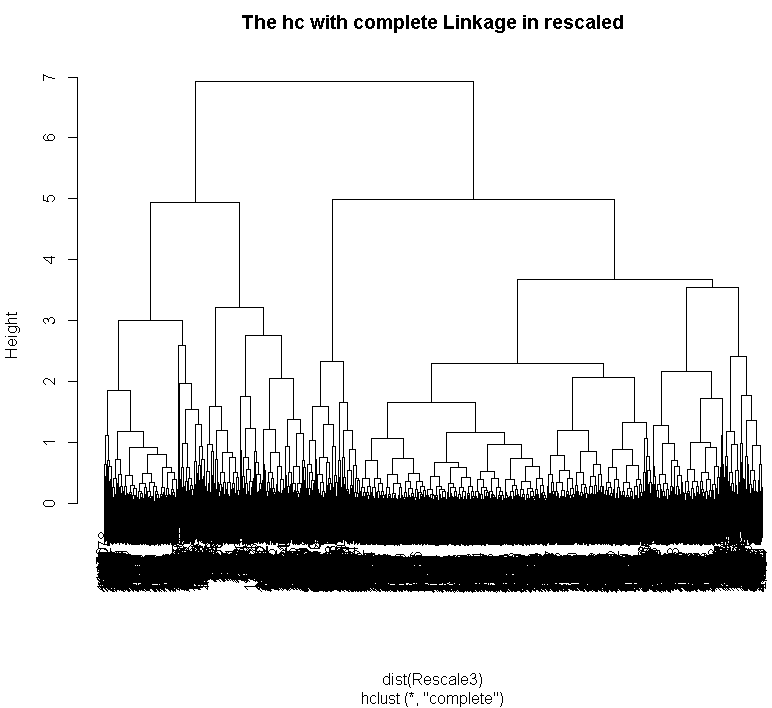
> rescaled.complete = hclust(dist(Rescale3), method="complete")

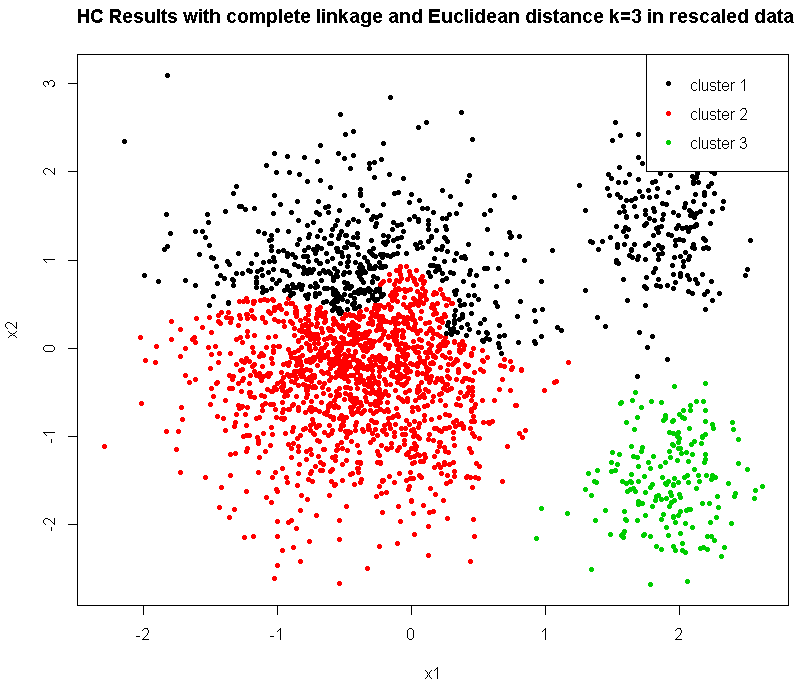
> plot(rescaled.complete, main = "The hc with complete Linkage in rescaled")

> plot(Rescale3, col = (cutree(rescaled.complete, 3) ), main=" HC Results with complete linkage and Euclidean distance k=3 in rescaled data", pch =20)

> legend("topright", legend = c("cluster 1", "cluster 2","cluster 3"),

+ col = c(1, 2, 3), pch = 20)





# rescale HC single

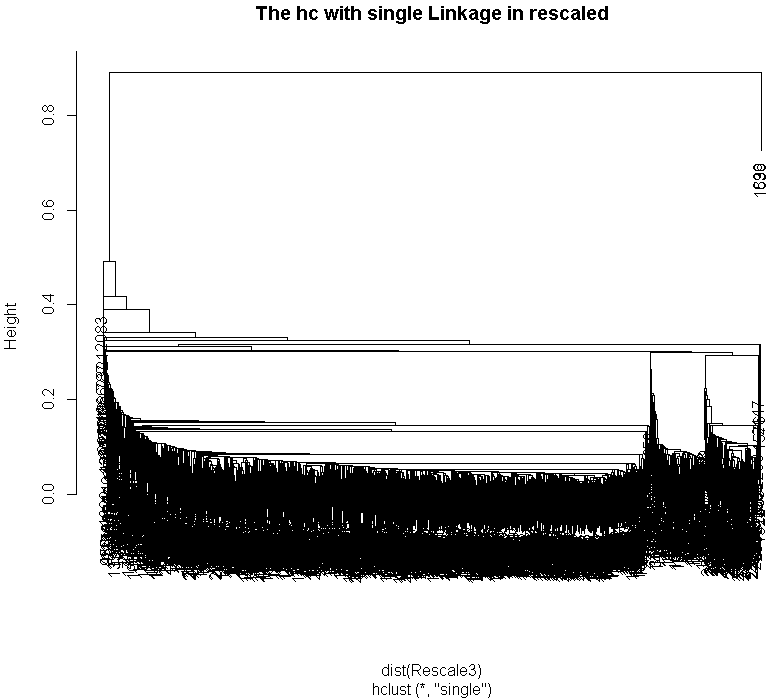
> rescaled.single = hclust(dist(Rescale3), method="single")

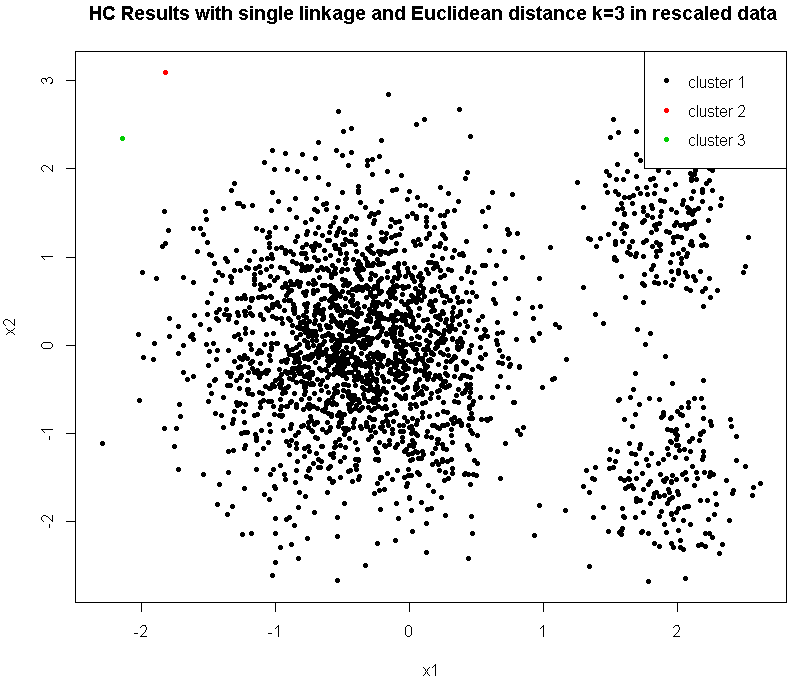
> plot(rescaled.single, main = "The hc with single Linkage in rescaled")

> plot(Rescale3, col = (cutree(rescaled.single, 3) ), main=" HC Results with single linkage and Euclidean distance k=3 in rescaled data", pch =20)

> legend("topright", legend = c("cluster 1", "cluster 2","cluster 3"),

+ col = c(1, 2, 3), pch = 20)





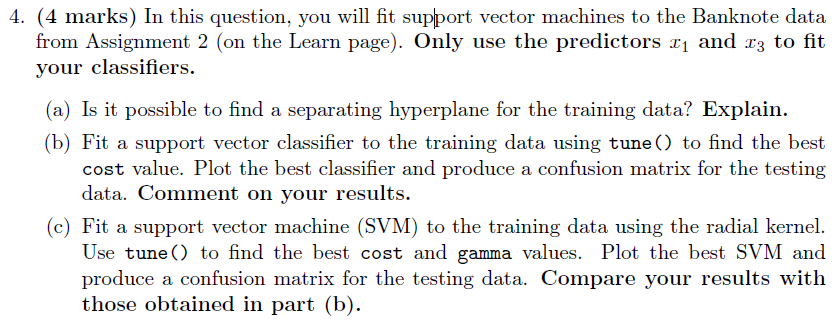
**Answer:** After using recalled data with three models, we can tell the single model was still

influenced by the two outliers and did a poor job on classification, however the complete

linkage model improved a lot, it was be able to classify the two smaller clusters into right

classes whereas the largest cluster was still split into two parts wrongly. The k-means model was better than the un-rescaled one, however the result was still not good in terms of the

largest cluster was still split into three parts.



(a)

train.bank = read.csv("BankTrain.csv", header = TRUE)[,c("x1","x3","y")]

> train.bank$y=as.factor(train.bank$y)

> test.bank = read.csv("Banktest.csv", header = TRUE)[,c("x1","x3","y")]

> test.bank$y=as.factor(test.bank$y)

> plot(train.bank[, c('x1', 'x3')], col = train.bank$y, main = ' The class of train data')

> legend("topright", legend = c("Forged banknote", "Genuine Banknote"), pch = c(1, 1), col = c(2, 1), text.col = "black")



**Answer:** For the figure above, it is clear there are some data of these two classes are

overlapped, therefore, it is impossible to find a separating hyperplane for the training data.

(b)

> library(e1071)

> set.seed(10)

> tune.out = tune(svm, y ~ ., data = train.bank, kernel="linear", ranges=list(cost=c(0.01,0.1,1,10,100,1000)))

> summary(tune.out)

Parameter tuning of ‘svm’:

- sampling method: 10-fold cross validation

- best parameters:

cost

0.1

- best performance: 0.1135417

> tune.out$best.parameters

cost

2 0.1

> bestmodel=tune.out$best.model

> plot(bestmodel, test.bank)

> test.pred = predict(bestmodel, test.bank)

> table(test.pred, test.bank$y)

test.pred 0 1

0 197 11

1 39 165

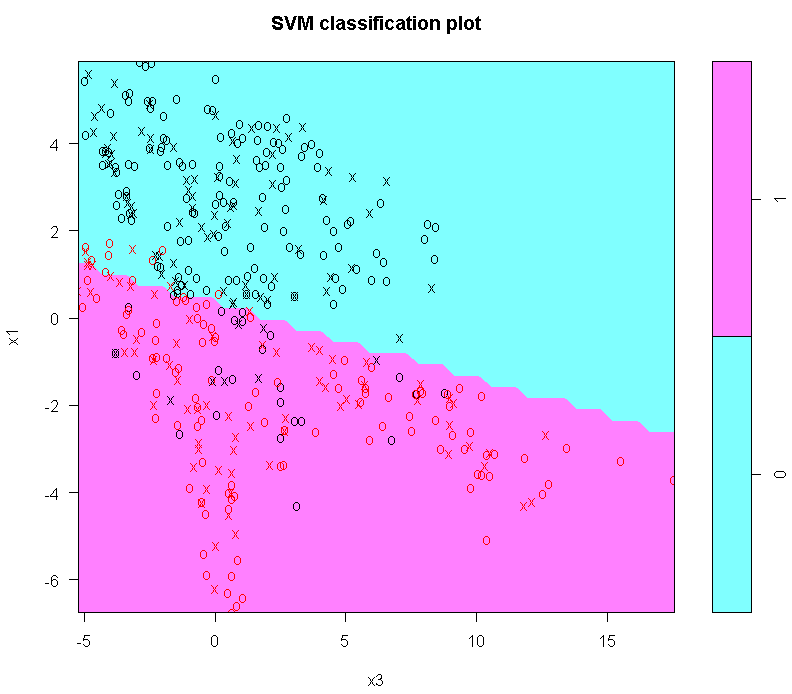


Figure 4.(b)

**Answer:** From the code above, the test error = (39+11) / 412= 12.14%, Precision is 80.88% , Specificity is 83.47% and Sensitivity is 93.75%, the test error is very low and Precision is

relative high, therefore this model is much useful.

(C)

> ##radial

> set.seed(10)

> tune.out = tune(svm, y ~ ., data = train.bank, kernel="radial", ranges=list(cost=c(0.1,1.5,10,50,100,1000), gamma=c(0.5,1,2,3,4)))

> summary(tune.out)

Parameter tuning of ‘svm’:

- sampling method: 10-fold cross validation

- best parameters:

cost gamma

10 4

- best performance: 0.09479167

> tune.out$best.parameters

cost gamma

27 10 4

> plot(tune.out$best.model, test.bank)

> test.pred.radial = predict(tune.out$best.model, test.bank)

> table(predict=test.pred.radial, truth=test.bank$y)

truth

predict 0 1

0 212 12

1 24 164

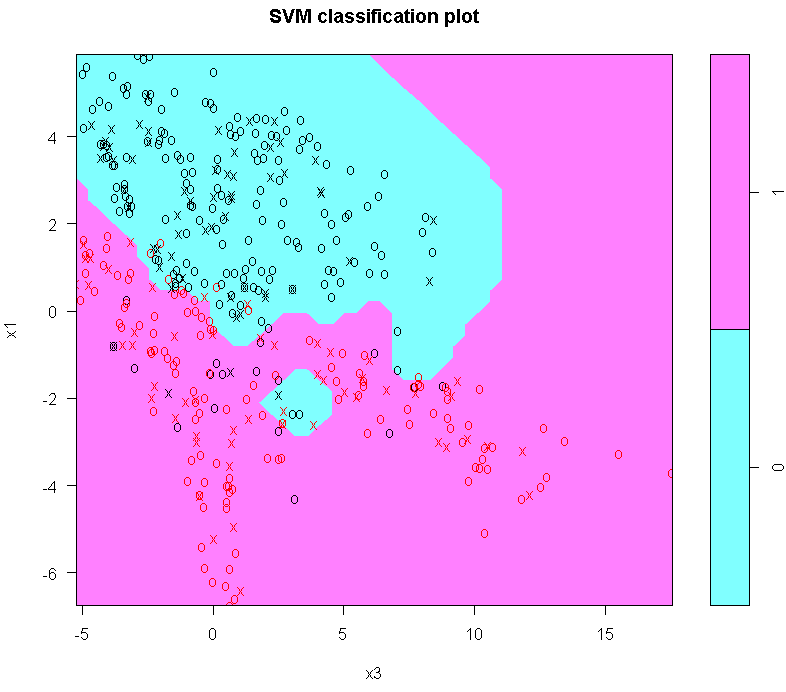


Figure 4.(c)

**Answer:** From the code above, the test error is 8.74%, Precision is 87.23%, Specificity is

89.83% and Sensitivity is 93.18%. Compare to the previous one, the test error is much

lower and Precision is much higher, In a word, the radial kernel model did a better job on

classifying the bank data.