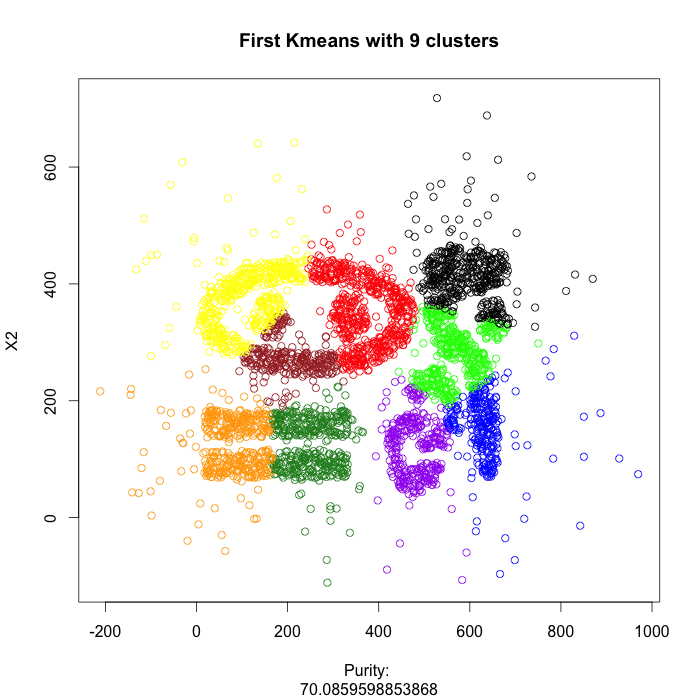
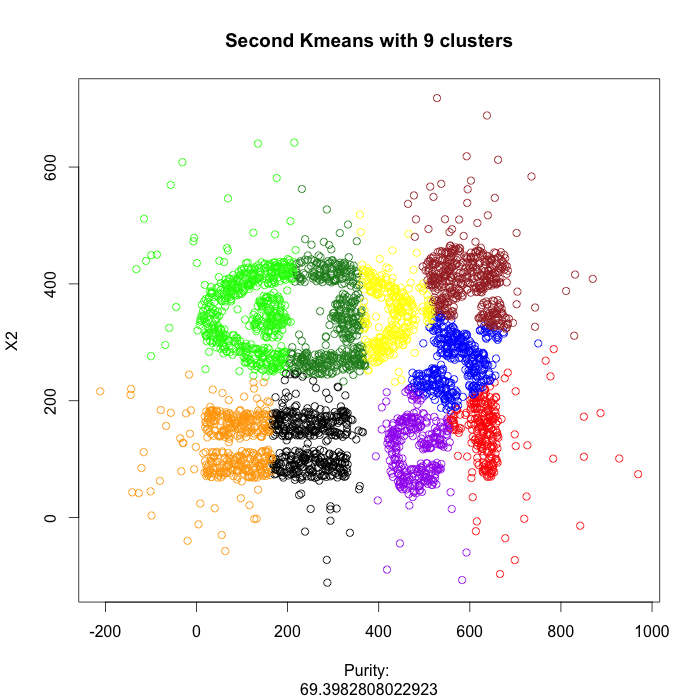
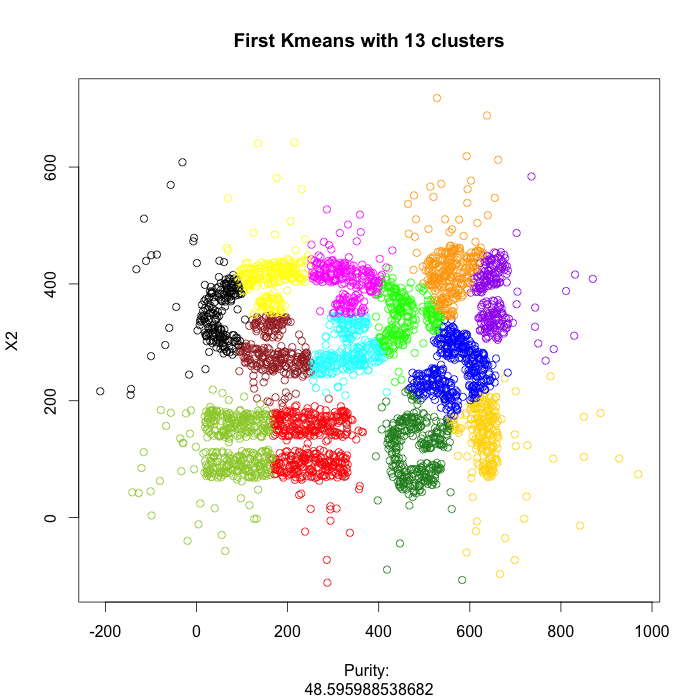
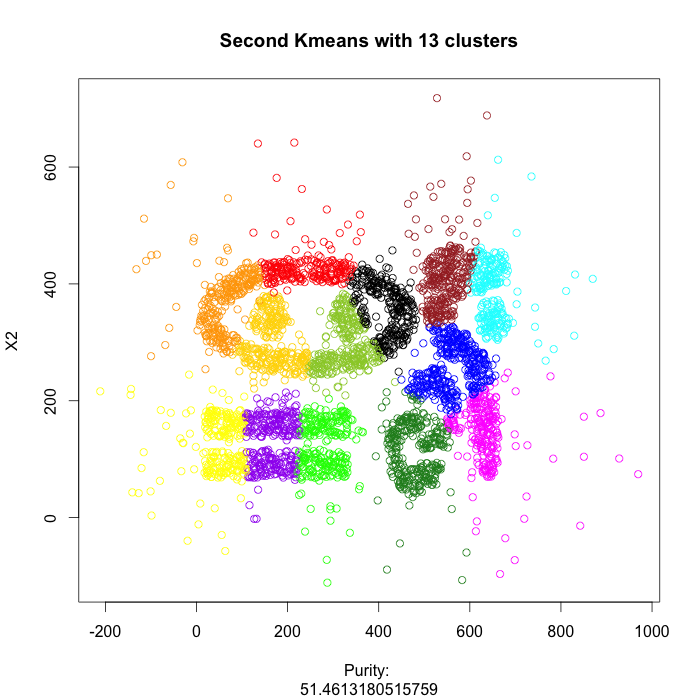
**4.**



With kmeans and k=9, the best purity we had was about 70%. We can see that kmeans did do a good job of clustering most the irregularly shaped objects, and it did not even cluster the rectangular shaped objects well.

Based on this first clustering, we can see that kmeans is not going to cluster this data well, with all of the convex shapes.





Kmeans with k=13 returns some awful results, as kmeans is sensitive to initialization. 13 clusters is way too many, as seen by the over partitioning of the data. There are multiple clusters where there should be a single cluster, such as the multicolored oval that surrounds the yellow and green circles.

As told by the first plot, kmeans on the Complex9\_GN16 dataset does not work well because it has convex shapes.

We can also see from all the plots that outliers are in separate clusters, which is another disadvantage of kmeans.

**5**. Running the program yielded these results:

> print(lowestSSE)

[1] 129.2636

9th attribute:

> purity\_kmeans(lowestSSECluster$cl, HAbalone$rings, numclusters)

[1] 26.5895954 0.5985157

> ordinal\_agreement(lowestSSECluster$cl, HAbalone$rings)

[1] 2.570056

> variance(lowestSSECluster$cl, HAbalone$rings)

[1] 6.701898323 0.005949548

10th attribute:

> purity\_kmeans(lowestSSECluster$cl, transformedClass, numclusters)

[1] 58.6464355 0.5985157

> ordinal\_agreement(lowestSSECluster$cl, transformedClass)

[1] 0.6670896

> variance(lowestSSECluster$cl, transformedClass)

[1] 0.462076436 0.005949548

**6**.

# Not enough clusters

db <- dbscan(complex9[1:2], 5, 8)

# Too many clusters

db <- dbscan(complex9[1:2], 5, 6)

# Too many clusters. Obviously not going to get good clustering from eps=5

db <- dbscan(complex9[1:2], 5, 7)

# Too many clusters

db <- dbscan(complex9[1:2], 8, 10)

# Number of clusters is ok but clustering itself is terrible. Not going to get good clustering from eps=8

db <- dbscan(complex9[1:2], 8, 12)

plot(complex9[,1:2], xlab = c("Purity: ", purity(db$cl, complex9[,3], 7)) , col = c("yellow","green","blue", "red","forestgreen", "brown", "black", "purple", "orange", "gold", "magenta", "yellowgreen", "cyan")[db$cl], main = "DBSCAN with 7 clusters")

# Again, number of clusters is fine but clustering itself is awful. Not going to get good clustering from high minpts

db <- dbscan(complex9[1:2], 10, 15)

plot(complex9[,1:2], xlab = c("Purity: ", purity(db$cl, complex9[,3], 12)) , col = c("yellow","green","blue", "red","forestgreen", "brown", "black", "purple", "orange", "gold", "magenta", "yellowgreen", "cyan")[db$cl], main = "DBSCAN with 12 clusters")

# Very good clustering results except for one circle object.

db <- dbscan(complex9[1:2], 12, 4)

plot(complex9[,1:2], xlab = c("Purity: ", purity(db$cl, complex9[,3], 10)) , col = c("yellow","green","blue", "red","forestgreen", "brown", "black", "purple", "orange", "gold", "magenta", "yellowgreen", "cyan")[db$cl], main = "DBSCAN with 10 clusters")

# Ok results but has lower purity than previous clustering and does not cluster the oval well

db <- dbscan(complex9[1:2], 14, 10)

plot(complex9[,1:2], xlab = c("Purity: ", purity(db$cl, complex9[,3], 10)) , col = c("yellow","green","blue", "red","forestgreen", "brown", "black", "purple", "orange", "gold", "magenta", "yellowgreen", "cyan")[db$cl], main = "DBSCAN with 10 clusters")

By slowly increasing the epsilon parameter and decreasing the minpts parameter, a clustering with purity of 92.5% and 6.5% outliers was obtained. Using a small epsilon with high minpts resulted in about 10 or so clusters, which is good, but the actual clustering was always terrible.

> ordinal\_agreement(db$cl, transformedClass)

[1] 0.793058

> db <- dbscan(HAbaloneNumeric, 12, 4)

> ordinal\_agreement(db$cl, transformedClass)

[1] 0.9503939

> db <- dbscan(HAbaloneNumeric, 0.7, 4)

> ordinal\_agreement(db$cl, transformedClass)

[1] 0.793058

> db <- dbscan(HAbaloneNumeric, 0.7, 8)

> ordinal\_agreement(db$cl, transformedClass)

[1] 0.7881357

> db <- dbscan(HAbaloneNumeric, 0.7, 12)

> ordinal\_agreement(db$cl, transformedClass)

[1] 0.7867164

> db <- dbscan(HAbaloneNumeric, 0.7, 3)

> ordinal\_agreement(db$cl, transformedClass)

[1] 0.7919858

As for minimizing the ordinal variance of HAbalone using DBSCAN, the best result obtained was 0.786 with epsilon=0.7 minpts=12. Any epsilon over 1 would not yield desirable results; only epsilon below 1 would give anything useful. Minpts didn’t really seem to make too much of an impact when the right epsilon was found.

**7**. My program works by using simulated annealing on each attribute in HAbalone. It generates random weights and applies those weights to a column in HAbalone, trying to optimize the ring variance by looking for the weights that result in the lowest variance. We always take a set of weights that results in a lower variance, but sometimes we accept worse variances with some probability to avoid getting stuck in local maxima.

First run:

[1] "Seed: " "9957"

[1] "Weight vector: " "0.644783909665421" "0.554246273124591" "1"

[5] "1" "0.12805168190971" "0.501830240478739" "0.0410262197256088"

[9] "1"

[1] "Variance: " "6.50702091563589"

> purity\_kmeans(cluster1$cl, HAbalone$rings, 6)

[1] 26.5414258 0.5985157

Second run:

[1] "Seed: " "1460"

[1] "Weight vector: " "0.899949649814516" "0.225848915288225" "1"

[5] "0.912668748991564" "0.29599526594393" "0.309574884129688" "0.00680477335117757"

[9] "1"

[1] "Variance: " "6.47936671754954"

> purity\_kmeans(cluster2$cl, HAbalone$rings, 6)

[1] 26.9026975 0.5985157

Third run:

[1] "Seed: " "849"

[1] "Weight vector: " "0.98669466888532" "0.94885303475894" "1" "1"

[6] "0.431450423784554" "0.468232655432075" "0.738544460851699" "1"

[1] "Variance: " "6.5031002225252"

> purity\_kmeans(cluster3$cl, HAbalone$rings, 6)

[1] 25.0240848 0.5985157

From the weights used by the algorithm, it appears that the 3rd attribute, diameter, always has a high weight and is therefore important in minimizing the variance of the rings attribute.

We can also see the same pattern with other attributes such as the 4th and 8th attributes, height and shell weight.

The remaining attributes are deemed as not so important by the algorithm.

**8**.

First run using lm:

Coefficients:

Estimate Std. Error t value Pr(>|t|)

(Intercept) 9.50811 0.05957 159.624 < 2e-16 \*\*\*

HAbalone$sex 0.40743 0.04672 8.720 < 2e-16 \*\*\*

HAbalone$length 0.94960 0.09679 9.811 < 2e-16 \*\*\*

HAbalone$height 0.50737 0.06412 7.913 3.18e-15 \*\*\*

HAbalone$wholeWeight 4.49104 0.35763 12.558 < 2e-16 \*\*\*

HAbalone$shuckedWeight -4.47858 0.18187 -24.625 < 2e-16 \*\*\*

HAbalone$visceraWeight -1.14211 0.14222 -8.031 1.25e-15 \*\*\*

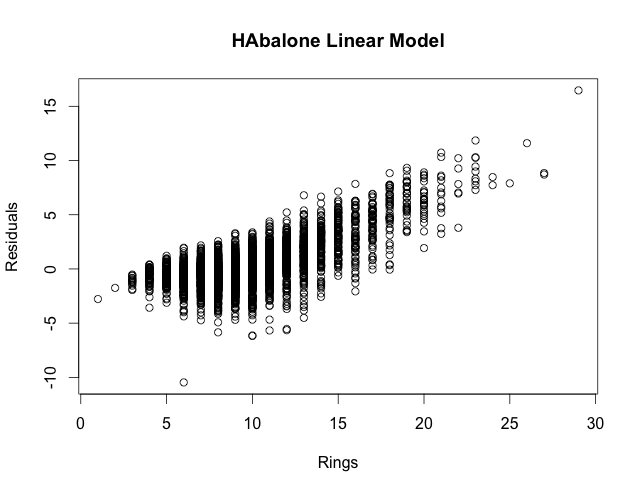
HAbalone$shellWeight 1.30495 0.15652 8.337 < 2e-16 \*\*\*

> print(summary(fit)$r.squared)

[1] 0.5321223

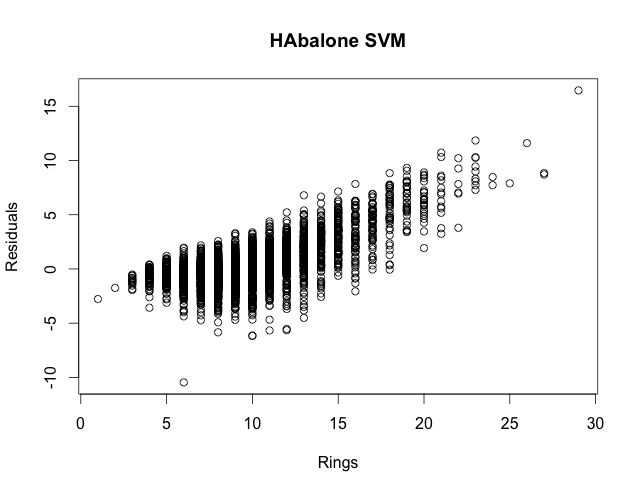
> print(mean(fit$residuals^2))

[1] 4.862549



Second run using svm:

> print(mean(fit2$residuals^2))

[1] 4.336401

By looking at the weights, the linear model algorithm decided that attributes such as whole weight and shucked weight were very important. However, both predictions models failed to fit the data well.

We can see that the linear model has an R-squared value of 53%, which is awful. We also see that both residual plots have clear linear patterns in them, which indicates a bad fit. Prediction models should not have a clear pattern if they fit the data well. The plots should appear random with points scattered all over the place.

**9**.

For Complex9\_GN16 dataset:

Kmeans: did a terrible job rediscovering classes because the data contains convex and irregularly shaped clusters. Also, there are many outliers in this data which is something that kmeans cannot handle. Kmeans is sensitive to outliers and does not group outliers in a separate cluster.

DBSCAN: did an overall great job rediscovering classes for Complex9. DBSCAN was able to handle the convex and irregular shapes in the data as well as cluster the outliers. There was one circle shaped cluster that DBSCAN was not able to recognize well. However, with some extra tuning of the DBSCAN parameters, DBSCAN should be able to recognize all clusters.

For HAbalone dataset:

Kmeans: Running kmeans on HAbalone yielded fairly good results with the lowest variances of rings coming in at around 6.5 or so. The data for HAbalone also has many outliers, such as Complex9, which kmeans cannot classify, but HAbalone has less irregular/convex shapes than Convex9 which enables kmeans to return better results. The problem here is that there seems to be a lower bound on the variance that one can achieve with kmeans on HAbalone, because the results from #7 never went below 6.5. There may be better ways to optimize variance over simulated annealing on each attribute, but it is doubtful that would yield a much better result, judging by the limitations of kmeans in general and also kmeans with 6 clusters.

DBSCAN: It’s difficult to visualize the clustering results for HAbalone since there are so many attributes, but DBSCAN seems to do a good job rediscovering classes on HAbalone. Ordinal variances as low as 0.78 were achieved, meaning the clusters did not have much variation and are seemingly good clustering results. In this case, DBSCAN didn’t seem to be extremely sensitive to initialization, as the range of variances achieved was about 0.16.

**Appendix A:**

How to run procedure for #7

Run Problem0.r, selecting the Abalone raw data file

Run the function in Problem7.r, find\_best\_clustering and pass it the numeric only version of HAbalone, called HAbaloneNumeric, which is generated in Problem0.r

So:

1. Run Problem0.r
2. Run cluster\_variable\_name 🡨 find\_best\_clustering(HAbaloneNumeric)

**Appendix B**

R code for problems 0, 1, 2, 3, 7, 8

Problem 0:

# Read in original data

library(readr)

data <- read\_csv(file.choose(), col\_names = FALSE)

# Create empty data frame for transformed data

sex = c(1)

length = c(1)

diameter = c(1)

height = c(1)

wholeWeight = c(1)

shuckedWeight = c(1)

visceraWeight = c(1)

shellWeight = c(1)

rings = data$X9

class = c('A')

HAbalone <- data.frame(sex, length, diameter, height, wholeWeight, shuckedWeight, visceraWeight, shellWeight, rings, class, stringsAsFactors=FALSE)

transformedClass = c()

# Transform the original data

for(i in 1:nrow(data))

{

# Convert sex attribute to integer

if(data[i,1] == 'M')

HAbalone[i,1] = 2

else if(data[i,1] == 'F')

HAbalone[i,1] = 1

else if(data[i,1] == 'I')

HAbalone[i,1] = 0

# Calculate z-scores for attributes 2-8

for(j in 2:8)

{

HAbalone[i,j] = ((data[i,j] - mean(data[,j])) /sd(data[,j]))

}

# Create class attribute based on ring attribute

# Also create a numeric based class attribute for use with clustering functions

if(data[i,9] <= 5)

{

HAbalone[i,10] = 'A'

transformedClass[i] = 4

}

else if(data[i,9] <= 8)

{

HAbalone[i,10] = 'B'

transformedClass[i] = 3

}

else if(data[i,9] <= 11)

{

HAbalone[i,10] = 'C'

transformedClass[i] = 2

}

else if(data[i,9] <= 17)

{

HAbalone[i,10] = 'D'

transformedClass[i] = 1

}

else

{

HAbalone[i,10] = 'E'

transformedClass[i] = 0

}

}

# Create numeric only version of HAbalone to be used with clustering algorithms

HAbaloneNumeric = data.frame(HAbalone[1:8], stringsAsFactors=FALSE)

Problem 1:

purity <- function(clusterassignment, classes, numclusters)

{

require(matrixStats)

# Create table of cluster assignment versus classes

t <- table(clusterassignment, classes)

pure = 0

total = 0

outliers = 0

# Count total number of objects

for(x in t)

total = total+x

# Count outliers in cluster 0

for(x in t[1,])

outliers = outliers+x

# Calculate purity on all clusters except 0

for(x in apply(t[2:numclusters,], 1, max))

pure = pure+x

pure = pure\*100/(total-outliers)

outliers = outliers/total\*100

results <- c(pure, outliers)

return(results)

}

Problem 1a (purity for kmeans):

purity\_kmeans <- function(clusterassignment, classes, numclusters)

{

require(matrixStats)

# Create table of cluster assignment versus classes

t <- table(clusterassignment, classes)

pure = 0

total = 0

# Pick the top objects with largest distance from centroid

outliers = 25

# Count total number of objects

for(x in t)

total = total+x

# Calculate purity on all clusters

for(x in apply(t[1:numclusters,], 1, max))

pure = pure+x

pure = pure\*100/(total-outliers)

outliers = outliers/total\*100

results <- c(pure, outliers)

return(results)

}

Problem 2:

ordinal\_agreement <- function(clusterassignment, classes)

{

# Get levels of the clusters

clusterlevels <- levels(factor(clusterassignment))

n <- length(clusterlevels)

b <- 1

totalOrdinalAgreement <- 0

populationSize = length(classes)

# Check if we have a 0 cluster

if (clusterlevels[1] == "0")

{

b <- 2

populationSize = populationSize - length(classes[clusterassignment==clusterlevels[1]])

}

# Calculate ordinal agreement

for (i in b:n)

{

cluster <- classes[clusterassignment==clusterlevels[i]]

clusterSize <- length(cluster)

clusterOrdinalAgreement <- 0

if(clusterSize != 1)

clusterOrdinalAgreement <- 0

for(j in 1:clusterSize)

{

for(k in 1:clusterSize)

{

if(j != k)

{

individualAgreement = abs(cluster[j] - cluster[k])

clusterOrdinalAgreement = clusterOrdinalAgreement + individualAgreement

}

}

clusterOrdinalAgreement = clusterOrdinalAgreement / (clusterSize^2 - clusterSize)

clusterOrdinalAgreement = clusterOrdinalAgreement \* (clusterSize / populationSize)

totalOrdinalAgreement = totalOrdinalAgreement + clusterOrdinalAgreement

}

}

return (totalOrdinalAgreement)

}

Problem 3:

variance <- function(clusterassignment, numerical)

{

# Get levels of the clusters

clusterlevels <- levels(factor(clusterassignment))

n <- length(clusterlevels)

var\_weight <- array(dim=c(n, 2))

population <- 0

b <- 1

outliers <- 0

# Check if we have a 0 cluster

if (clusterlevels[1] == "0")

{

b <- 2

var\_weight[1,] <- c(0, 0)

outliers <- length(numerical[clusterassignment == "0"])

}

else

{

outliers = 25

}

# Calculate variance

for (i in b:n)

{

cluster <- numerical[clusterassignment==clusterlevels[i]]

population <- population + length(cluster)

if (length(cluster) == 1)

{

var\_weight[i,] <- c(0, length(cluster))

} else

{

var\_weight[i,] <- c(var(cluster), length(cluster))

}

}

percentage <- outliers / (outliers + population)

totalv <- 0

for (i in 1:n)

totalv <- totalv + var\_weight[i, 1] \* var\_weight[i, 2] / population

return (c(totalv, percentage))

}

Problem 7:

find\_best\_clustering <- function(data)

{

acceptanceProbability <- function(neighbor, newNeighbor, temp, startTemp)

{

result = 0.0

# If the new neighbor is better than the old one, accept it

if(newNeighbor < neighbor)

result = 1.0

# Otherwise we accept the new neighbor with some probability

else

result = ((neighbor - newNeighbor) + (temp/startTemp^2))\*(temp/startTemp)

return (result)

}

# Transform the data by modifying a column with a random weight

transformData <- function(data, column, weight)

{

data[, i] = data[, i] \* weight

return (data)

}

seed = sample(1:10000, 1, TRUE)

set.seed(seed)

currentData <- data

transformedData <- data

k <- kmeans(data, 6)

kNew <- k

cooldownRate = 0.7

weights = c(1, 1, 1, 1, 1, 1, 1, 1)

variances = c()

# Simulated annealing on each attribute in the dataset

for(i in 1:ncol(data))

{

temp = 50

startTemp = temp

while(temp > 1)

{

w = runif(1, 0.0, 1.0)

transformedData <- transformData(currentData, i, w)

# Have to use MacQueen algorithm here or else too much memory is used and program fails to complete

# Also have to increase max iterations and sometimes with only 10 iterations kmeans does not converge

kNew <- kmeans(transformedData, 6, algorithm="MacQueen", iter.max=100)

neighbor = variance(k$cl, HAbalone$rings)

newNeighbor = variance(kNew$cl, HAbalone$rings)

if(acceptanceProbability(neighbor[1], newNeighbor[1], temp, startTemp) >= runif(1, 0.0, 1.0))

{

k <- kNew

currentData <- transformedData

weights[i] = w

variances = c(variances, newNeighbor[1])

}

temp = temp - cooldownRate

gc()

}

}

print(c("Seed: ", seed))

print(c("Weight vector: ", weights))

print(c("Variance: ", variance(k$cl, HAbalone$rings)[1]))

plot(variances)

return (k)

}

Problem 8:

library("e1071")

# Fit linear model

fit <- lm(HAbalone$rings ~ HAbalone$sex + HAbalone$length + HAbalone$height + HAbalone$wholeWeight + HAbalone$shuckedWeight + HAbalone$visceraWeight + HAbalone$shellWeight, data=HAbalone)

print(summary(fit)$r.squared)

print(mean(fit$residuals^2))

plot(HAbalone$rings, fit$residuals, ylab="Residuals", xlab="Rings", main="HAbalone Linear Model")

fit2 <- svm(HAbalone$rings ~ HAbalone$sex + HAbalone$length + HAbalone$height + HAbalone$wholeWeight + HAbalone$shuckedWeight + HAbalone$visceraWeight + HAbalone$shellWeight, data=HAbalone)

print(mean(fit2$residuals^2))

plot(HAbalone$rings, fit2$residuals, ylab="Residuals", xlab="Rings", main="HAbalone SVM")