B529: Homework 3

Nathan Byers

Wednesday, April 7, 2015

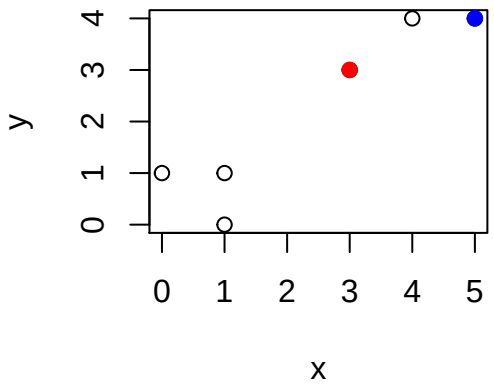
Question 1

In the clustering problems, the input data set contains the following six data points (1,1), (0,1), (1,0), (3,3) (4,4), (5,4). Manhattan distance is chosen to compute the distance between two data points. The k-means clustering method is used to find the clusters with k = 2 and initial centroids (3,3) and (5,4). Please describe the steps of the k-means clustering. (In each step, provide the information about cluster assignments of data points, centroids; and when the algorithm stops). (25 points)

Answer 1

First I read the points into R and plot them.

```
x <- c(1, 0, 1, 3, 4, 5)
y <- c(1, 1, 0, 3, 4, 4)
seed1 <- c(3, 3)
seed2 <- c(5, 4)

plot(x = x, y = y)
points(c(seed1[1], seed2[1]),
       c(seed1[2], seed2[2]),
       col = c("red", "blue"),
       pch = 19)
```



Then I create a data frame that gives the initial state of the points.

```
group.df <- data.frame(x, y, group = NA,
                       centr_x = NA, centr_y = NA)
group.df[group.df$x == 3 &
           group.df$y == 3,
         c("group", "centr_x", "centr_y")] <- c(1, 3, 3)

group.df[group.df$x == 5 &
           group.df$y == 4,
         c("group", "centr_x", "centr_y")] <- c(2, 5, 4)

manDist <- function(x, y, centr_x, centr_y){
  abs(centr_x - x) +
    abs(centr_y - y)
}

group.df$dist1 <- unlist(mapply(manDist, group.df$x,
                       group.df$y,
                       MoreArgs = list(
                         centr_x = seed1[1],
                         centr_y = seed1[2])))

group.df$dist2 <- unlist(mapply(manDist, group.df$x,
                                group.df$y,
                                MoreArgs = list(
                                  centr_x = seed2[1],
                                  centr_y = seed2[2])))

group.df$group <- apply(group.df, 1, function(row){
  if(row["dist1"] ==
       min(row[c("dist1", "dist2")])){
    1
  }else{2}
})

group.df$change <- NA

group.df
```

```
##   x y group centr_x centr_y dist1 dist2 change
## 1 1 1     1      NA      NA     4     7     NA
## 2 0 1     1      NA      NA     5     8     NA
## 3 1 0     1      NA      NA     5     8     NA
## 4 3 3     1       3       3     0     3     NA
## 5 4 4     2      NA      NA     2     1     NA
## 6 5 4     2       5       4     3     0     NA
```

Now we find the new centroids, recalculate the distances from each point to the new centroids, and regroup the points according to the shortest Manhatten distance to the new centroids.

```
reGroup <- function(df){
  center_x1 <- mean(df[df$group == 1, "x"])
  center_y1 <- mean(df[df$group == 1, "y"])
  center_x2 <- mean(df[df$group == 2, "x"])
```

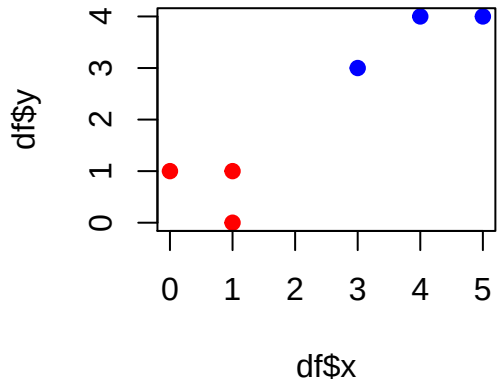
```
  center_y2 <- mean(df[df$group == 2, "y"])
  df[df$group == 1, "centr_x"] <- center_x1
  df[df$group == 1, "centr_y"] <- center_y1
  df[df$group == 2, "centr_x"] <- center_x2
  df[df$group == 2, "centr_y"] <- center_y2
  df$dist1 <- unlist(mapply(manDist, df$x,
                            df$y,
                            MoreArgs = list(
                              centr_x = center_x1,
                              centr_y = center_y1)))
  df$dist2 <- unlist(mapply(manDist, df$x,
                            df$y,
                            MoreArgs = list(
                              centr_x = center_x2,
                              centr_y = center_y2)))
  g.initial <- df$group
  g <- apply(df, 1, function(row){
    if(row["dist1"] != row["dist2"]){
      if(row["dist1"] ==
           min(row[c("dist1", "dist2")])){
        1
      }else{2}
    }else{0}
  })
  df$group <- g
  df$change <- sapply(1:length(g), function(i){
    !(g[i] == g.initial[i])
  })
  df
}

group.df2 <- reGroup(group.df)

group.df2
```

```
##   x y group centr_x centr_y dist1 dist2 change
## 1 1 1     1    1.25    1.25   0.5   6.5  FALSE
## 2 0 1     1    1.25    1.25   1.5   7.5  FALSE
## 3 1 0     1    1.25    1.25   1.5   7.5  FALSE
## 4 3 3     2    1.25    1.25   3.5   2.5   TRUE
## 5 4 4     2    4.50    4.00   5.5   0.5  FALSE
## 6 5 4     2    4.50    4.00   6.5   0.5  FALSE
```

We can see that at least one point changed. Below is a plot of the new groups.

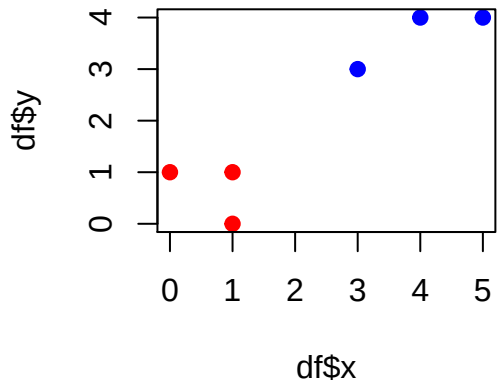
```
plotGroups <- function(df){
  group.cols <- df$group
  group.cols[group.cols == 1] <- "red"
  group.cols[group.cols == 2] <- "blue"
  group.cols[group.cols == 0] <- "black"
  plot(df$x, df$y, pch = 19,
       col = group.cols)
}
```

```
plotGroups(group.df2)
```



Now we regroup again, because at least one point changed groups in the last iteration.

```
group.df3 <- reGroup(group.df2)

plotGroups(group.df3)
```




```
group.df3
```

```
##   x y group   centr_x   centr_y     dist1     dist2 change
## 1 1 1     1 0.6666667 0.6666667 0.6666667 5.6666667  FALSE
## 2 0 1     1 0.6666667 0.6666667 1.0000000 6.6666667  FALSE
## 3 1 0     1 0.6666667 0.6666667 1.0000000 6.6666667  FALSE
```

```
## 4 3 3     2 4.0000000 3.6666667 4.6666667 1.6666667  FALSE
## 5 4 4     2 4.0000000 3.6666667 6.6666667 0.3333333  FALSE
## 6 5 4     2 4.0000000 3.6666667 7.6666667 1.3333333  FALSE
```

We can see in the table that no points have changed groups, and the plot looks the same as the plot in the last iteration. So we stop here, because the algorithm has converged.

Question 2

Suppose DNA bases in a protein-coding region follow the distribution:

DNA base	Probability
A	θ
C	$\frac{1}{4}$
G	$\frac{1}{2}$
T	$\frac{1}{4} - \theta$

In an experiment, the number of observed "A" or "C" bases at the position is x, and the number of observed "G" or "T" bases at the position is y. The EM algorithm can be used to find parameter θ. Describe the Expectation step and Maximization step in the EM algorithm (25 points)

Answer 2

If the number of A, C, G, and T bases are a, c, g, and t respectively, then the likelihood is

$$P(a, c, g, t|\theta) = C(\theta)^a \left(\frac{1}{4}\right)^c \left(\frac{1}{2}\right)^g \left(\frac{1}{4} - \theta\right)^t$$

We take the log, take the derivative with respect to θ, and set it equal to 0 to solve for θ.

$$\ln P(a, c, g, t|\theta) = \ln C + a \ln \theta + c \ln \frac{1}{4} + g \ln \frac{1}{2} + t \ln \left(\frac{1}{4} - \theta\right)$$

$$\frac{\partial}{\partial \theta} \ln P(a, c, g, t|\theta) = \frac{a}{\theta} - \frac{t}{\frac{1}{4} - \theta} = 0$$

$$\theta = \frac{a}{4(t + a)}$$

We know $\frac{a}{c} = \frac{\theta}{\frac{1}{4}}$ and $a + c = x$, so $a = \frac{c\theta}{\frac{1}{4}} = \frac{(x-a)\theta}{\frac{1}{4}} = \frac{\theta}{\frac{1}{4}+\theta}x$. We also know $\frac{t}{g} = \frac{\frac{1}{4}-\theta}{\frac{1}{2}}$ and $g + t = y$, so $t = \frac{(\frac{1}{4}-\theta)g}{\frac{1}{2}} = \frac{(\frac{1}{4}-\theta)(y-t)}{\frac{1}{2}} = \frac{\frac{1}{4}-\theta}{\frac{3}{4}-\theta}y$. For the algorithm, we give an initial value to θ, and if we know the value of x and y, we solve for a and t. Using these initial values we solve for a new θ,

$$\theta_{new} = \frac{a}{4(t + a)}$$

We then use θ_{new} to find a_{new} and t_{new}. The steps are repeated until convergence.

Suppose x = 12,000 and y = 13,000 and the initial θ is 0.1. Below is the EM algorithm.

```
theta <- 0.1
calcA <- function(theta, x = 12000){
  (x * theta)/((1/4) + theta)
}
calcT <- function(theta, y = 13000){
  (((1/4) - theta) * y)/((3/4)-theta)
}
calcTheta <- function(a, t){
  a/(4 * ( t+ a))
}

em.df <- data.frame(n = 0, a = NA, t = NA, theta = theta)
theta.diff <- theta
while(theta.diff> 0.00001){
  n.last = em.df[nrow(em.df), "n"]
  theta.old = em.df[em.df$n == n.last, "theta"]
  a.new = calcA(theta.old)
  t.new = calcT(theta.old)
  theta.new = calcTheta(a.new, t.new)
  n.new = n.last + 1
  em.new.df = data.frame(n = n.new, a = a.new,
                         t = t.new, theta = theta.new)
  em.df <<- rbind(em.df, em.new.df)
  theta.diff <<- abs(theta.new - theta.old)
  if(n.new > 1000) break
}
em.df
```

```
##     n        a         t     theta
## 1   0       NA        NA 0.1000000
## 2   1 3428.571 3000.0000 0.1333333
## 3   2 4173.913 2459.4595 0.1573074
## 4   3 4634.555 2033.1017 0.1737700
## 5   4 4920.688 1719.7820 0.1852538
## 6   5 5107.469 1490.4059 0.1935270
## 7   6 5236.038 1319.2885 0.1996864
## 8   7 5328.684 1188.5530 0.2044073
## 9   8 5397.994 1086.3503 0.2081164
## 10  9 5451.446 1004.8036 0.2110918
## 11 10 5493.704  938.5760 0.2135209
## 12 11 5527.799  883.9650 0.2155335
## 13 12 5555.780  838.3403 0.2172222
## 14 13 5579.073  799.7915 0.2186546
## 15 14 5598.698  766.9020 0.2198810
## 16 15 5615.405  738.6021 0.2209395
## 17 16 5629.755  714.0702 0.2218596
## 18 17 5642.177  692.6657 0.2226644
## 19 18 5653.003  673.8825 0.2233723
## 20 19 5662.493  657.3155 0.2239978
## 21 20 5670.857  642.6370 0.2245530
```

```
## 22 21 5678.262  629.5793 0.2250478
## 23 22 5684.845  617.9214 0.2254901
## 24 23 5690.720  607.4796 0.2258868
## 25 24 5695.980  598.0996 0.2262436
## 26 25 5700.702  589.6513 0.2265653
## 27 26 5704.954  582.0242 0.2268560
## 28 27 5708.792  575.1235 0.2271192
## 29 28 5712.263  568.8680 0.2273581
## 30 29 5715.409  563.1871 0.2275751
## 31 30 5718.266  558.0198 0.2277727
## 32 31 5720.863  553.3128 0.2279528
## 33 32 5723.229  549.0193 0.2281171
## 34 33 5725.387  545.0981 0.2282673
## 35 34 5727.357  541.5129 0.2284047
## 36 35 5729.158  538.2317 0.2285305
## 37 36 5730.807  535.2257 0.2286458
## 38 37 5732.317  532.4695 0.2287515
## 39 38 5733.701  529.9404 0.2288485
## 40 39 5734.971  527.6179 0.2289377
## 41 40 5736.137  525.4838 0.2290197
## 42 41 5737.209  523.5216 0.2290950
## 43 42 5738.194  521.7163 0.2291644
## 44 43 5739.101  520.0547 0.2292282
## 45 44 5739.935  518.5244 0.2292871
## 46 45 5740.703  517.1145 0.2293413
## 47 46 5741.411  515.8151 0.2293912
## 48 47 5742.063  514.6170 0.2294373
## 49 48 5742.665  513.5118 0.2294798
## 50 49 5743.219  512.4922 0.2295190
## 51 50 5743.731  511.5511 0.2295552
## 52 51 5744.203  510.6823 0.2295887
## 53 52 5744.639  509.8801 0.2296195
## 54 53 5745.042  509.1392 0.2296480
## 55 54 5745.414  508.4546 0.2296744
## 56 55 5745.757  507.8221 0.2296987
## 57 56 5746.075  507.2376 0.2297212
## 58 57 5746.368  506.6973 0.2297420
## 59 58 5746.639  506.1978 0.2297613
## 60 59 5746.890  505.7359 0.2297791
## 61 60 5747.122  505.3088 0.2297955
## 62 61 5747.336  504.9138 0.2298107
## 63 62 5747.534  504.5484 0.2298248
## 64 63 5747.718  504.2105 0.2298378
## 65 64 5747.887  503.8978 0.2298498
## 66 65 5748.044  503.6085 0.2298610
## 67 66 5748.189  503.3408 0.2298713
## 68 67 5748.324  503.0931 0.2298808
```

Question 3

Use the ID3 method to construct a decision tree using the following data set for credit card application. (25 points)

Age	Income	Gender	Risk
<25	>50K	M	High
<25	>50K	F	High
≥ 25	<50K	F	High
≥ 25	>50K	F	Low
≥ 25	>50K	M	Low
<25	<50K	M	High

Answer 3

Question 4

A linear SVM is used to analyze a 2-dimensional data set, and the solution α to the quadratic programming program contains only three non-zero elements $\alpha_1 = 0.05$, $\alpha_2 = 0.05$, and $\alpha_3 = 0.1$. The corresponding input data points are $\mathbf{x}_1 = [0, 3]^T$, $y_1 = 1$, $\mathbf{x}_2 = [2, 4]^T$, $y_2 = 1$, $\mathbf{x}_3 = [3, -0.5]^T$, $y_3 = -1$. Compute the weight vector $\mathbf{w}$ and parameter b for the separating line with the maximum margin (15 points). Compute the margin of the separating line (10 points).