B529: Homework 3

Nathan Byers

Wednesday, April 7, 2015

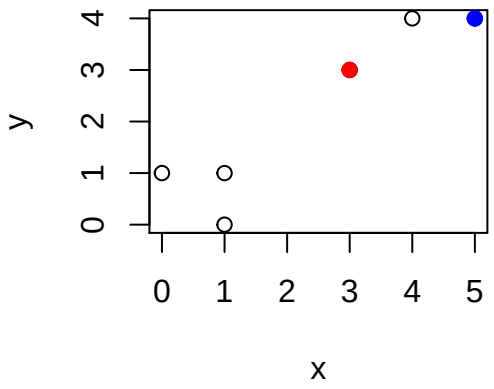
Question 1

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 create the points in 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 Manhattan 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"])
```

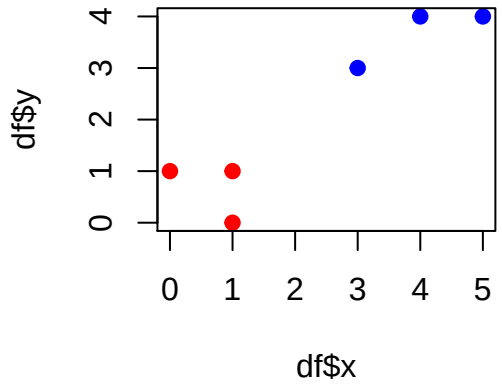
```
  center_y2 <- mean(df[df$group == 2, "y"])
  df[df$group == 1, "centr_x"] <- center_x1
  df[df$group == 1, "centr_y"] <- center_y1
  df[df$group == 2, "centr_x"] <- center_x2
  df[df$group == 2, "centr_y"] <- center_y2
  df$dist1 <- unlist(mapply(manDist, df$x,
                            df$y,
                            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.

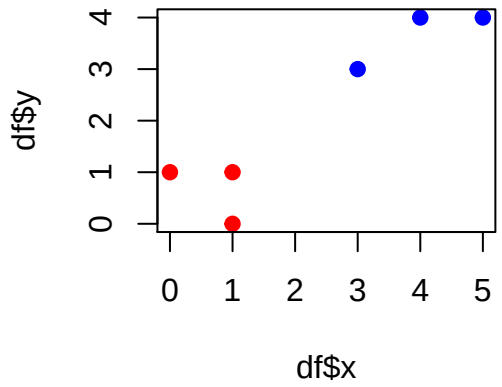
```
plotGroups <- function(df){
  group.cols <- df$group
  group.cols[group.cols == 1] <- "red"
  group.cols[group.cols == 2] <- "blue"
  group.cols[group.cols == 0] <- "black"
  plot(df$x, df$y, pch = 19,
       col = group.cols)
}
```

```
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
}
```

The algorithm converged after 67 iterations

```
head(em.df)
```

```
##   n        a        t     theta
## 1 0       NA       NA 0.1000000
## 2 1 3428.571 3000.000 0.1333333
## 3 2 4173.913 2459.459 0.1573074
## 4 3 4634.555 2033.102 0.1737700
## 5 4 4920.688 1719.782 0.1852538
## 6 5 5107.469 1490.406 0.1935270
```

```
tail(em.df)
```

```
##     n        a        t     theta
## 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

First, I put the factors in a data frame and write a function for finding the entropy of a factor based on how it splits the risk column.

```
age <- factor(c(0, 0, 1, 1, 1, 0), labels = c("<25", ">25")) # 0 = <25, 1 = >25
income <- factor(c(0, 0, 1, 0, 0, 1), labels = c(">50k", "<50k")) # 0 = >50K, 1 = <50K
gender <- factor(c(0, 1, 1, 1, 0, 0), labels = c("M", "F")) # 0 = M, 1 = F
risk <- factor(c(0, 0, 0, 1, 1, 0), labels = c("High", "Low")) # 0 = High, 1 = Low

df <- data.frame(age, income, gender, risk)

df
```

```
##   age income gender risk
## 1 <25   >50k      M High
## 2 <25   >50k      F High
## 3 >25   <50k      F High
## 4 >25   >50k      F  Low
## 5 >25   >50k      M  Low
## 6 <25   <50k      M High
```

```
library(dplyr)

calcEntropy <- function(split.factor, outcome){
  # get proportions
  proportions = sapply(split(split.factor, split.factor),
                       function(level, total){length(level)/total},
                       total = length(split.factor))
  if(identical(split.factor, outcome)){
    sum(sapply(proportions, function(x) -x * log(x, 2)))
  }else{
  # put in data frame, group by factor and outcome,
  # get total counts
```

```
  df = data.frame(split.factor, outcome, I = 1)
  df = group_by(df, split.factor, outcome)
  df.sum = summarize(df, count = sum(I))
  # get the levels of the split.factor and subset
  split.levels = levels(split.factor)
  df.split1 = filter(df.sum, split.factor == split.levels[1])
  # entropy of outcome after split, level 1
  entropy1 = sapply(df.split1$count, function(x, total) {x/total},
                    total = sum(df.split1$count))
  entropy1 = sum(sapply(entropy1, function(x) -x * log(x, 2)))
  # entropy of outcome after split, level 2
  df.split2 = filter(df.sum, split.factor == split.levels[2])
  entropy2 = sapply(df.split2$count, function(x, total) {x/total},
                    total = sum(df.split2$count))
  entropy2 = sum(sapply(entropy2, function(x) -x * log(x, 2)))
  # sum entropies proportionally
  proportions[[1]] * entropy1 + proportions[[2]] * entropy2
  }
}
```

If we calculate the entropy using the Gender column, this is the output of the function.

```
calcEntropy(gender, risk)
```

```
## [1] 0.9182958
```

I loop through all of the columns of the data frame, including the Risk column to get the initial entropy, and I get the change in entropy for each factor.

```
entropy.list <- lapply(df, calcEntropy, outcome = df$risk)

information.gain <- entropy.list[[4]] - unlist(entropy.list[1:3])

information.gain
```

```
##       age    income    gender
## 0.4591479 0.2516292 0.0000000
```

The result indicates that Age is the first factor to split on.

Next we split the table using Age.

```
df.age.split <- lapply(levels(age),
                       function(i) df[df$age == i,
                                      c("income", "gender", "risk")])

df.age.split
```

```
## [[1]]
##   income gender risk
## 1   >50k      M High
```

```
## 2   >50k      F High
## 6   <50k      M High
##
## [[2]]
##   income gender risk
## 3   <50k      F High
## 4   >50k      F  Low
## 5   >50k      M  Low
```

The first data frame in the `df.age.split` object has 0 entropy as a starting point, so there won't be any information gain. So we find the entropy when we split the second data frame using the remaining factors.

```
df2 <- df.age.split[[2]]

entropy.list2 <- lapply(df2, calcEntropy,
                        outcome = df2$risk)

information.gain2 <- entropy.list[[3]] - unlist(entropy.list2[1:2])

information.gain2
```

```
##    income    gender
## 0.9182958 0.2516292
```

We can see that Income has the highest information gain, so we split on that factor next.

Now we split the data by Income and we can see that there will be no information gained from the Gender factor.

```
df.income.split <- lapply(levels(income),
                          function(i) df2[df2$income == i,
                                          c("gender", "risk")])
df.income.split
```

```
## [[1]]
##   gender risk
## 4      F  Low
## 5      M  Low
##
## [[2]]
##   gender risk
## 3      F High
```

So we stop here.

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).

Answer 4

$$\mathbf{w} = \sum \alpha y \boldsymbol{x} = 0.05 \cdot 1 \cdot \begin{bmatrix} 0 \\ 3 \end{bmatrix} + 0.05 \cdot 1 \cdot \begin{bmatrix} 2 \\ 4 \end{bmatrix} 0.1 \cdot -1 \cdot \begin{bmatrix} 3 \\ -0.5 \end{bmatrix} = \begin{bmatrix} -0.2 \\ 0.4 \end{bmatrix}$$

$$y_n(\mathbf{w^T}\mathbf{x}_n + b) = 1$$

$$1(\begin{bmatrix} -0.2 & 0.4 \end{bmatrix} \begin{bmatrix} 0 \\ 3 \end{bmatrix} + b) = 1$$

$$b = -0.2$$

$$margin = \frac{1}{\|\mathbf{w}\|} = \sqrt{-0.2^2 + 0.4^2} = 0.447$$