2018 Fall

Computational Statistics
HW#3

182STG18 이하경

[Problem 1] K-means Clustering

Description

Clustering은 반응변수 Y가 존재하지 않는 Unsupervised Learning의 일종으로, Supervised Learning과 달리 결과의 평가와 비교가 쉽지 않다. 따라서 적절한 Objective Function을 설정하는 것이 매우 중요하다. K-means 알고리즘은 이러한 Objective Function을 같은 군집 내 관측치들 사이 거리의 합(SWS; Sum of Within Scatters)으로 하여, Local Search 를 이용해 SWS 을 최소화하는 군집의 구분을 찾는다. 함수의 값이 정의되기 위해서 관측치들은 모두 수치형이어야 하는 특징이 있다.

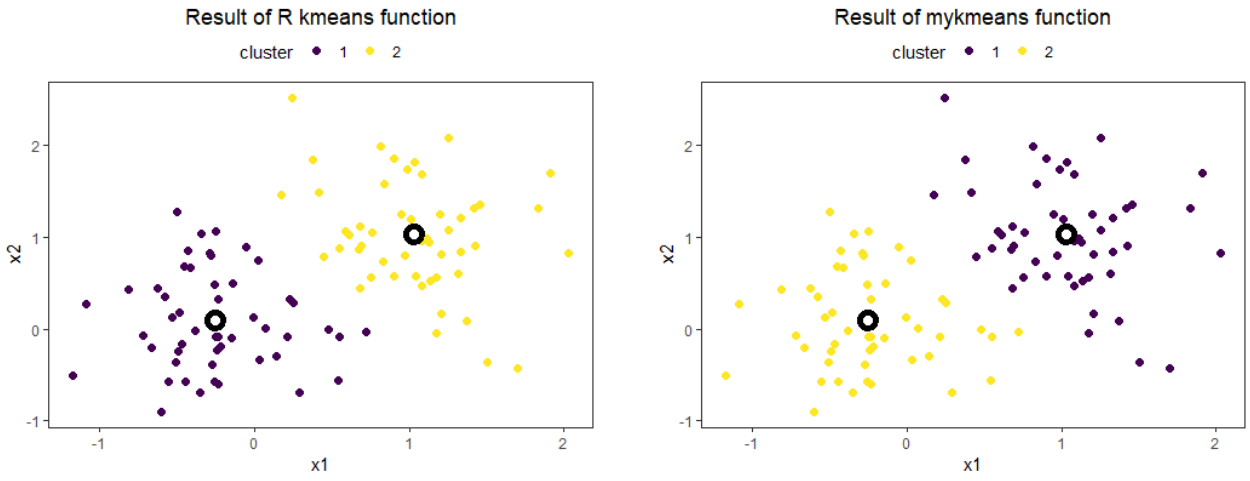
Goal n 개의 관측치를 k 개의 군집으로 구분하는 경우, SWS 를 최소화하여 같은 군집 내 동질성과 서로 다른 군집 사이의 이질성을 최대화하는 k-means 알고리즘을 직접 구현해보고 R 의 kmeans 함수와 결과를 비교하려고 한다.

우선 비교에 사용하기 위해 두 가지 정규분포 $N(0, 0.5^2)$과 $N(1, 0.5^2)$을 따르는 random sample 을 100 개씩 생성하여 변수 x_1과 x_2의 관측치를 50 개씩 생성하였다. 전체 관측치 100 개를 2 개의 군집으로 구분하고자 한다.

R 의 kmeans 함수와 동일하게 관측치 matrix X 와 서로 다른 군집의 수 k, 최대 반복 수 (default maxiter=10), 초기 시행횟수(defalult nstart=1)을 input 으로 하여 다음의 알고리즘에 따라 함수를 작성하였다.

1) 관측치 중 k 개의 점을 랜덤하게 선택하여 초기 평균 값(initial mean)으로 지정한다.
2) 각각의 관측치 별로 가장 가까운 평균 값을 기준으로 cluster 를 분류한다.
3) 각 cluster 별로 평균 값을 update 한다.
4) 새로운 평균 값을 기준으로 cluster 를 재분류하고, cluster mean 을 다시 update 한다.
5) 분류된 cluster 와 cluster 별 mean 값이 변하지 않을 때까지 위의 과정을 반복한다.

위에서 생성한 100 개의 normal sample matrix 를 k=2 개의 군집으로 분류하기 위해 maxiter=10, nstart=10 으로 지정하고 결과를 확인하였다. 왼쪽은 R 의 kmeans 함수, 오른쪽은 직접 작성한 mykmeans 함수의 결과를 요약한 것이다.



R kmeans		cluster 1	cluster 2
centers	x1	-0.2502	1.0336
	x2	0.0813	1.0197
size		50	50
Within SS		21.1924	26.0569
Total WSS		47.2493	
Between SS		63.2179	
TSS		110.4673	
niter		1	
comp.time		0.00057	

mykmeans		cluster 1	cluster 2
centers	x1	1.0336	-0.2502
	x2	1.0197	0.0813
size		50	50
Within SS		26.0569	21.1924
Total WSS		47.2493	
Between SS		63.2179	
TSS		110.4673	
niter		2	
comp.time		1.320	

Discussion

그래프에서 각 관측치 별로 분류된 군집을 확인해보면 순서에 상관없이 100 개의 점들이 각각 50 개씩 군집을 형성하여 같은 결과를 보이고 있다. 크게 표시된 두개의 점은 각 군집의 최종 cluster mean 을 나타내는 점이다. 표에서 Total WSS 값과 Between SS 값을 더하면 Total SS 값과 일치하는 것을 확인할 수 있다. 같은 함수를 여러 번 반복 해보았을 때 Total WSS 의 값이 동일하게 계산되었으므로 해당 값이 min SWS 이며 나눠진 군집이 타당하다고 할 수 있다. 이 경우 관측치들의 구분이 눈으로 보기에도 명확하기 때문에, 계산 속도가 비교적 상당히 빠르고 결과 또한 예상과 일치하였다.

cf. 초기 시행횟수 nstart 를 1 으로 하여 몇 번 시행했을 때는 분류된 군집의 결과가 몇 개의 점에서 다르게 나타났고 total WSS 또한 위의 47.2493 보다 조금 크게 계산되었다. 따라서 초기 시행횟수를 여러 번으로 하여 그중 가장 total WSS 를 가장 작게 하는 초기 평균 지점을 찾는 것이 계산의 안정성을 높이고 결과의 타당성 또한 높일 수 있다는 것을 알게 되었다.

과제의 경우와 달리 실제 관측치들은 설명변수의 개수가 3 차원 이상으로 그래프와 같이 시각적으로 확인하기 어렵고, 군집의 분류 또한 답이 정해져 있지 않다. 관측치가 모두 수치형일 경우 K-means 알고리즘을 통해 SWS 을 최소화하는 군집을 찾아내는 것이 매우 합리적인 방법이 될 수 있다.

[Problem 2] Variable Selection in Regression: using Simulated Annealing & Genetic Algorithm

Description

HW#2 에서 Baseball data 를 이용해 야구 선수의 연봉을 예측하는 회귀 모형의 잠재적인 설명변수는 27 개로, 총 고려 가능한 변수의 조합은 2^{27} = 134,217,728 개이다. 모형의 예측력을 높이기 위해 AIC 를 가장 작게 하는 모형을 찾으려고 하나, 설명변수 선택에 따른 경우의 수가 매우 많으므로 global 최적 값을 찾는 것은 상당히 많은 시간이 소요된다. HW#2 중 Local Search 방법 중 하나로 Stepwise Regression 을 이용해 (local) min AIC 를 찾았다. Stepwise Regression 은 이전과 다음 단계의 변수 선택(또는 제거)에 따른 AIC 만을 비교하는 Greedy 방식으로 빠른 시간 내에 local minimum 을 찾아낼 수 있지만 경쟁력 없는 local 값에 빠질 수 있다는 단점이 있다. 과제에서 찾고자 하는 최적 값은 설명변수의 선택에 따른 모형의 AIC 로, 상당히 많은 local minima 를 가지고 있기 때문에 이 값들에 trap 되어 global minima 를 찾기 쉽지 않다. 본 과제에서 사용하는 Simulated Annealing 과 Genetic Algorithm 은 이러한 local search 의 단점을 보완할 수 있도록 만들어진 알고리즘으로, global minimum 값을 찾아낼 가능성이 높아진다.

Goal Simulated Annealing 과 Genetic Algorithm 을 R 의 'GenSA'와 'GA' package 를 이용해 Baseball Data 에서 AIC 를 가장 작게 하는 변수의 조합을 찾아보려고 한다. 찾은 결과와 HW#2 에서 Stepwise Regression 의 결과를 비교한다.

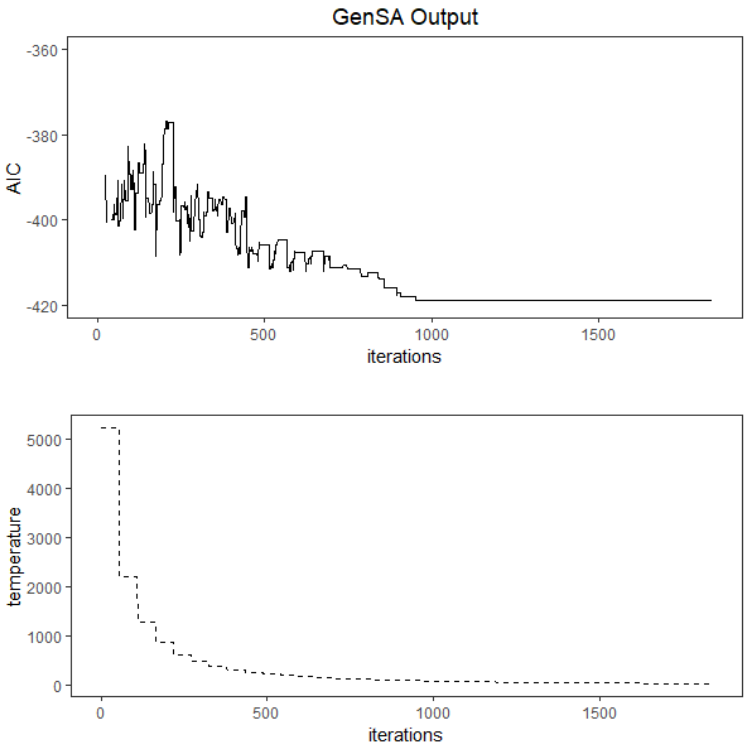
1. Simulated Annealing (R GenSA package)

Annealing 알고리즘은 Local optima 에 빠져 종료되는 위험을 줄이고자 일정한 확률에 비례하여 해당 opitimum 값을 선택하지 않는 확률적인 방법을 도입하는 방법으로 metropolis-hasting 과 비슷하다. 이때 이 확률은 '온도'에 의해 결정되며, 높은 온도에서 시작하여 온도를 점차 낮춰 나가는 양상을 띄고 있다. 온도가 높을 경우에는 현재까지의 최적 해보다 랜덤 생성된 다른 해의 '에너지'가 더 높더라도 일정 확률로 다른 해를 채택하여 local optima 에서 빠져나올 장치를 마련하고 있다. 최적화가 계속 지속되어 온도가 낮아질수록 에너지가 높은 해를 채택할 확률은 줄어들고 점차 하나의 최적 해로 수렴한다.

R 에서 GenSA package 를 이용하기 위해 입력 값과 출력 값은 모두 수치형으로 갖는 하나의 함수가 필요하였다. 따라서 설명변수의 선택을 length=27 의 vector 형 input 으로 하여 해당 모형의 AIC 를 numeric 값으로 반환하는 함수를 생성하고 GenSA 함수에 사용하였다.

	fn	lower	upper	max.time
GenSA	myaic	rep(0, 27)	rep(1, 27)	10 (seconds)

해를 찾을 범위는 최소 rep(0, 27)에서 최대 rep(1, 27)으로 설명변수가 존재하지 않는 null model 부터 27 개 변수를 모두 사용하는 full model 까지를 고려한다. 최대 소요시간을 10 초로 설정하고 GenSA 를 이용해 min AIC 를 찾은 결과는 아래와 같다.



value
-418.95
par (variable index)
2 3 6 8 10 13 14 15 16 24 25 26

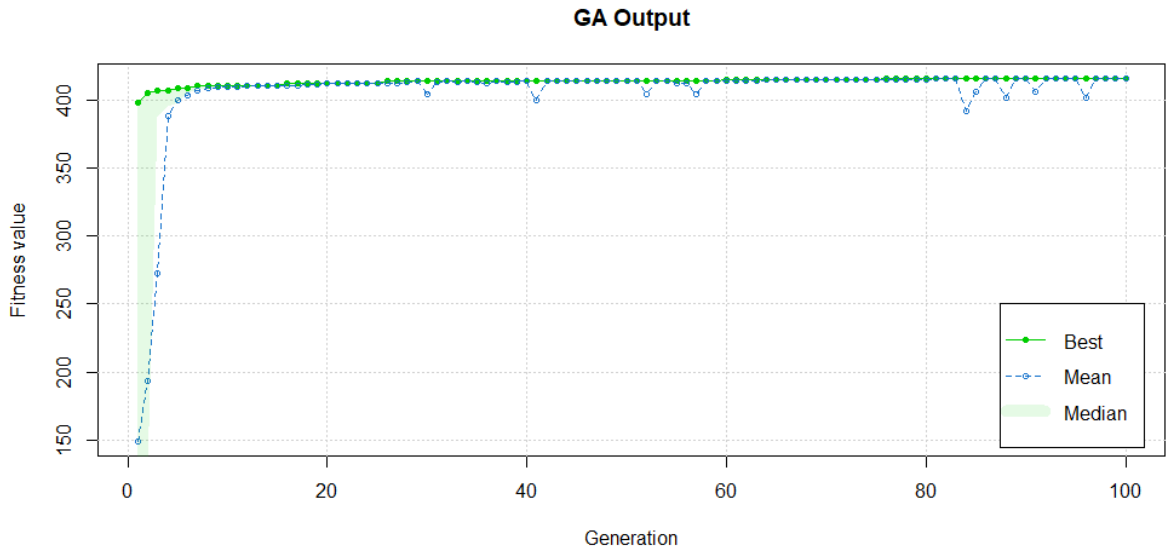
Annealing 알고리즘에 따라 AIC 값은 초기 -100.27 에서 시작하여 변동을 거쳐 점차 작은 값으로 이동하였다. 왼쪽의 그래프는 -420 부터 -360 사이의 AIC 에 대해서 그린 그래프이다. iteration 수의 증가에 따라 AIC 값이 변동하지만 점차 작아지는 경향을 보인다. 온도의 변화를 보면 초기의 높은 온도에서는 값의 변동이 심하지만 낮은 온도로 변화함에 따라 새로운 해를 선택할 확률이 줄어드므로 반복 수 1000 회를 넘어서부터는 AIC 가 거의 하나의 값에 수렴하는 것을 확인할 수 있다.

2. Genetic Algorithm (R GA package)

유전 알고리즘은 자연의 진화 과정을 모방하여 자연선택, 적자생존 이론을 바탕으로 하는 최적화 알고리즘이다. 가능한 해의 자료구조를 유전자로 나타내고 이들을 선택, 교차, 변이 등의 연산을 거쳐 변형함으로써 점차 더 좋은 해를 탐색해가는 과정이라고 할 수 있다. 일반적으로 우수한 해(함수의 적합도가 높은 해)일수록 다음 세대로 전해질 확률을 높게 부여하여 점차 좋은 optima 를 찾아나간다. '변이(mutation)' 연산을 이용해 임의로 값을 변형하는 기법을 사용하는데, 이 역시 local optima 에 빠질 가능성을 줄이고 다른 곳에 존재하는 global optima 를 찾을 가능성을 높이는 방법이다.

	fn	lower	upper	pcrossover	pmutation	popsize	maxiter
GA	-myaic	rep(0, 27)	rep(1, 27)	0.8 (default)	0.1 (default)	20	100

R 의 GA package 를 이용하여, 다음의 속성들을 설정하여 결과 값을 찾아보았다. maximize 하기 위한 fitness 함수는 위에서 작성한 myaic 함수의 반대 부호 값이다.



100 iters	1000 iters
value	**value**
416.22	418.94
par (variable index)	**par (variable index)**
3 8 10 13 14 15 16 24	1 3 8 10 13 14 15 16 24 25 26

위의 그래프에서 Best 는 세대마다 형성된 20 개의 유전자 중 가장 우수한 개체의 함수 값을 나타내며, Mean 은 해당 세대의 평균 함수 값을 나타낸다. 그래프의 곡선이 최적해에 가까워질수록 평평해지는 것을 확인할 수 있다. 반복 수를 100 회로 설정하였을 때 찾은 값은 local maxima 에 해당하였으며 선택된 변수들 또한 다른 방법들과 상이하였다. 따라서 최대 반복 수를 1000 으로 재설정하고 같은 함수를 다시 계산하였을 때는 계산 소요 시간이 조금 더 걸렸지만 조금 더 값이 커져 최적 값에 가까워졌다. 하지만 이 역시 global optima 에는 해당하지 않았다.

Discussion - summary -

Method	Selected Variables													# of Variables	AIC
	1	2	3	6	8	10	13	14	15	16	24	25	26		
GenSA		•	•	•	•	•	•	•	•	•	•	•	•	12	**-418.95**
GA (100)			•		•	•	•	•	•	•	•			8	-416.22
GA (1000)	•		•	•	•	•	•	•	•	•	•	•	•	12	-418.94

															(HW#2)
Stepwise	•		•	•	•	•	•	•	•	•	•	•	•	12	-418.94
All Possible		•	•	•	•	•	•	•	•	•	•	•	•	12	**-418.95**

위에서 Simulated Annealing 과 Genetic Algorithm 의 두 가지 방법을 이용해 min AIC 를 찾아보았다. 이 두 알고리즘은 Greedy Search 의 단점을 보완하여 좀 더 경쟁력 있는 minimum 값을 찾을 것으로 기대하지만 실제로 global minimum 을 찾기 위해서는 시간이 많이 소요된다는 단점이 있다. 실제로 R 을 사용해 구현해 보았을 때, 최대 소요시간이나 반복 수를 작게 설정하였음에도 불구하고 stepwise 를 이용했을 때보다 계산 시간이 눈에 띄게 길었다.

GenSA 의 경우 찾아낸 min AIC 의 값은 -418.95 로 global min AIC 에 해당한다. HW#2 에서 leaps package 를 이용해 All Possible Regression 을 해보았을 때와 같은 결과를 보였으며 선택된 변수들 또한 같았다. GA 의 경우, 반복 수를 100 으로 설정하였을 때는 local minimum 인 -416.22 에서 벗어나지 못하였지만 반복 수를 좀 더 크게 설정하였을 때는 Stepwise 의 결과와 동일하게 좀 더 경쟁력 있는 값을 찾았다. 알고리즘 내의 교차 확률(pcrossover), 변이 확률(pmutation) 등을 임의로 조정하여 시도해보았으나 global 값을 찾아내는 데에 영향을 미치지는 못하였다. 반복 수를 더욱 크게 늘려 오랜 시간 동안 세대 구성을 반복한다면 최적 값을 찾아낼 수 있을 것이라고 예상하였다.

두 가지 알고리즘과 R package 내 설정 값에 대한 배경지식이 부족하여 관련 속성들을 설정하는 데에 어려움이 있었지만 직접 구현해보면서 다음 단계의 값이 현재의 값보다 좋지 않다면 계산을 종료하는 greedy search 의 단점이 보완되는 것을 그래프의 변동에서 시각적으로 확인할 수 있었으며, 결과 값 또한 나아지는 것을 확인하였다. 최적에 가까운 값을 찾기 위한 좀 더 효과적인 방법임을 알 수 있었다.

[Appendix] R Code

```
# (1) kmeans
  set.seed(1234)
  x1 = matrix(rnorm(100, sd = 0.5), ncol = 2)
  x2 = matrix(rnorm(100, 1, sd = 0.5), ncol = 2)
  x = rbind(x1, x2) ; colnames(x) = c("x1", "x2") ; x = as.data.frame(x)

  kmean = kmeans(x, 2, nstart = 10)

  ggplot() + theme_test() + theme(legend.position = "top", plot.title = element_text(hjust = 0.5)) + labs(title = "Result of R kmeans function") +
    geom_point(data = data.frame(x, cluster = factor(kmean$cluster)), aes(x1, x2, color = cluster), size = 2) + scale_color_viridis_d() +
    geom_point(data = data.frame(kmean$centers), aes(x1, x2), size = 3, shape = 21, color = "black", fill = "white", stroke = 3)

# function
  mykmeans <- function(x, k, maxiter=10, nstart=1) {

    n = nrow(x)
    tss = sum(dist(x)^2) / n
    niter = 0

    init.mean = list() ; output = list()
    for (ns in 1:nstart) {

      init.mean[[ns]] <- x[sample(n,k),]

      cl.mean = init.mean[[ns]]
      cl = numeric(k)
      ss = matrix(numeric(n*k), nrow = n)
      change = F

      while (change == F & niter <= maxiter) {
        cl.mean0 <- cl.mean
        cl0 <- cl

        for (j in 1:k) {
          for (i in 1:n) {
            ss[i,j] = sum((x[i,]-cl.mean[j,])^2)
            cl[i] = which.min(ss[i,])
          }
        }
        spl = split(x, cl)
        for (j in 1:k) { cl.mean[j,] = apply(spl[[j]], mean, MARGIN = 2) }

        change = identical(cl, cl0)
        niter <- niter + 1
      }

      size = c() ; wss = c()
      for (j in 1:k) {
        size[j] = nrow(spl[[j]])
        wss[j] = sum(dist(spl[[j]])^2) / size[j]
      }
      tot.wss = sum(wss)
      bss = tss - tot.wss

      output[[ns]] <- list(cluster=cl, centers=cl.mean, tss=tss, wss=wss, tot.wss=tot.wss, bss=bss, size=size, niter=niter)
```

```
    }

    final <- output[[which.min(tot.wss)]]
    return(final)
  }

  mykmean = mykmeans(x, k=2, nstart=10) ; mykmean

  data = data.frame(x, cluster = factor(mykmean$cluster))
  ggplot() + theme_test() +
    geom_point(data = data, aes(x1, x2, color = cluster), size = 2) + scale_color_viridis_d() +
    geom_point(data = mykmean$centers, aes(x1, x2), size = 3, shape = 21, color = "black", fill = "white", stroke = 3) +
    theme(legend.position = "top", plot.title = element_text(hjust = 0.5)) + labs(title = "Result of mykmeans function")


# (2) baseball
  baseball = read.csv("C:/Users/HG/Documents/R/data/baseball.txt", header = T, sep = " ")

  myaic <- function(par) {
    index = as.logical(par >= 0.5)
    y = baseball[1]
    x = baseball[-1][index]

    model <- lm(log(salary) ~., data.frame(y, x))
    aic <- extractAIC(model)[2]

    return(aic)
  }

# (2-1) GenSA
  gensa.out <- GenSA(fn = myaic, lower = rep(0, 27), upper = rep(1, 27),
                     control = list(max.time = 10, seed = 1))
  gensa.out$value
  as.logical(gensa.out$par >= 0.5) %>% which()

  gensa.info <- as.data.frame(gensa.out$trace.mat)

  g1 <- ggplot(gensa.info) + labs(x = "iterations", y = "AIC", title = "GenSA Output") + ylim(-420, -360) + theme_test() +
    geom_line(aes(1:length(function.value), current.minimum), color = "blue", linetype = "dashed") +
    geom_line(aes(1:length(function.value), function.value)) + theme(plot.title = element_text(hjust = 0.5))

  g2 <- ggplot(gensa.info) + geom_line(aes(1:length(temperature), temperature), linetype = "dashed") +
    labs(x = "iterations", title = "")  + theme_test() + theme(plot.subtitle = element_text(hjust = 0.5))

  grid.arrange(g1, g2, nrow = 2)


# (2-2) GA
  ga.out <- ga(type = "real-valued", fitness = function(x) -myaic(x),
               lower = rep(0, 27), upper = rep(1, 27),
               popSize = 20, maxiter = 100, seed = 1)

  ga.info <- summary(ga.out)
  ga.info$fitness
  which(ga.info$solution[1,] >= 0.5)

  plot(ga.out, main = "GA Output")
```