Computing\_HW2\_엄상준

Eom SangJun

2020 10 27

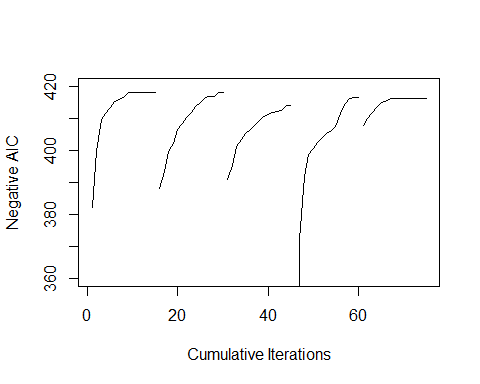
#########################################################################  
### 3.1 RANDOM STARTS LOCAL SEARCH  
#########################################################################  
## INITIAL VALUES  
baseball.dat = read.table('baseball.dat',header=TRUE)  
baseball.dat$freeagent = factor(baseball.dat$freeagent)  
baseball.dat$arbitration = factor(baseball.dat$arbitration)  
baseball.sub = baseball.dat[,-1]  
salary.log = log(baseball.dat$salary)  
n = length(salary.log) #case 개수  
m = length(baseball.sub[1,]) #독립변수 개수  
num.starts = 5 #random start 개수  
runs = matrix(0,num.starts,m)  
itr = 15  
runs.aic = matrix(0,num.starts,itr)  
  
# INITIALIZES STARTING RUNS  
set.seed(1234)   
for(i in 1:num.starts){runs[i,] = rbinom(m,1,.5)}   
#random으로 열을 뽑는 것을 다섯 번 시행  
  
  
## MAIN  
for(k in 1:num.starts){  
 run.current = runs[k,]  
  
 # ITERATES EACH RANDOM START  
 for(j in 1:itr){  
 run.vars = baseball.sub[,run.current==1] #1로 선택된 변수들 뽑아내기.  
 g = lm(salary.log~.,run.vars)  
 run.aic = extractAIC(g)[2] #[1]은 equivalent d.f, [2]가 AIC  
 run.next = run.current  
  
 # TESTS ALL MODELS IN THE 1-NEIGHBORHOOD AND SELECTS THE  
 # MODEL WITH THE LOWEST AIC  
 for(i in 1:m){  
 run.step = run.current  
 run.step[i] = !run.current[i] #0이라면 1로 1이라면 0으로 바꿈.  
 run.vars = baseball.sub[,run.step==1] #바뀐 것으로 variable을 새로 뽑음.  
 g = lm(salary.log~.,run.vars) #model 적용  
 run.step.aic = extractAIC(g)[2] #AIC 구하기  
 if(run.step.aic < run.aic){  
 run.next = run.step  
 run.aic = run.step.aic  
 } #만약 AIC가 더 작다면 run.next로 할당해줌. 그리고 run.aic도 바꿔줌  
 }  
 run.current = run.next #run.next를 current에 적용.  
 runs.aic[k,j]=run.aic   
 }  
 runs[k,] = run.current #최종적으로 제일 작은 aic를 가진 subset이 골라짐.  
}  
  
## OUTPUT  
runs # LISTS OF PREDICTORS

## [,1] [,2] [,3] [,4] [,5] [,6] [,7] [,8] [,9] [,10] [,11] [,12] [,13] [,14]  
## [1,] 0 1 1 0 0 1 0 1 0 1 0 1 1 1  
## [2,] 0 1 1 0 0 1 0 1 0 1 0 1 1 1  
## [3,] 0 0 0 0 0 0 0 1 1 1 0 0 1 1  
## [4,] 1 0 1 0 0 0 0 1 0 1 0 0 1 1  
## [5,] 0 0 1 0 0 0 0 1 0 1 0 1 1 1  
## [,15] [,16] [,17] [,18] [,19] [,20] [,21] [,22] [,23] [,24] [,25] [,26]  
## [1,] 1 1 0 0 0 0 0 0 0 0 1 1  
## [2,] 1 1 0 0 0 0 0 0 0 0 1 1  
## [3,] 0 1 1 1 0 1 1 1 0 0 1 0  
## [4,] 1 1 0 0 0 0 0 0 0 1 0 0  
## [5,] 1 1 0 0 0 1 0 1 0 0 0 0  
## [,27]  
## [1,] 0  
## [2,] 0  
## [3,] 0  
## [4,] 0  
## [5,] 0

runs.aic # AIC VALUES

## [,1] [,2] [,3] [,4] [,5] [,6] [,7]  
## [1,] -382.1396 -399.5996 -409.5874 -411.5453 -413.5192 -415.3651 -415.9898  
## [2,] -388.0134 -393.5020 -399.7536 -402.2507 -406.3813 -408.3073 -410.2499  
## [3,] -391.0411 -395.3438 -401.3549 -403.3081 -405.1843 -406.4663 -407.8866  
## [4,] -199.6572 -372.1001 -391.9549 -398.5628 -400.4917 -402.3174 -404.0648  
## [5,] -407.8876 -409.8851 -411.7404 -413.4253 -414.8167 -415.3757 -416.1567  
## [,8] [,9] [,10] [,11] [,12] [,13] [,14]  
## [1,] -416.4954 -418.0000 -418.0000 -418.0000 -418.0000 -418.0000 -418.0000  
## [2,] -411.9831 -413.6061 -415.1196 -416.6207 -416.7355 -416.9711 -418.0000  
## [3,] -409.2473 -410.4997 -411.3319 -411.9675 -412.2386 -412.5352 -413.8955  
## [4,] -405.2679 -406.2594 -407.3961 -411.5770 -414.6209 -416.2200 -416.3802  
## [5,] -416.1567 -416.1567 -416.1567 -416.1567 -416.1567 -416.1567 -416.1567  
## [,15]  
## [1,] -418.0000  
## [2,] -418.0000  
## [3,] -413.8955  
## [4,] -416.3802  
## [5,] -416.1567

##PLOT  
plot(1:(itr\*num.starts),-c(t(runs.aic)),xlab="Cumulative Iterations",  
 ylab="Negative AIC",ylim=c(360,420),type="n")  
for(i in 1:num.starts) {  
 lines((i-1)\*itr+(1:itr),-runs.aic[i,]) }



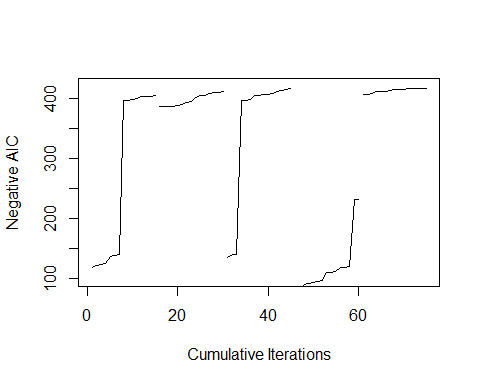
##3-1(a)  
#setting은 위와 동일  
runs2 = matrix(0,num.starts,m)  
runs.aic2 = matrix(0,num.starts,itr)  
  
# INITIALIZES STARTING RUNS  
set.seed(1234)   
for(i in 1:num.starts){runs2[i,] = rbinom(m,1,.5)}   
#random으로 열을 뽑는 것을 다섯 번 시행  
  
for(k in 1:num.starts){  
 run.current = runs2[k,]  
   
 # ITERATES EACH RANDOM START  
 for(j in 1:itr){  
 run.vars = baseball.sub[,run.current==1] #1로 선택된 변수들 뽑아내기.  
 g = lm(salary.log~.,run.vars)  
 run.aic = extractAIC(g)[2] #[1]은 equivalent d.f, [2]가 AIC  
 run.next = run.current  
   
 #aic가 작은게 나오면 바로 채택  
 for(i in 1:m){  
 run.step = run.current  
 run.step[i] = !run.current[i] #0이라면 1로 1이라면 0으로 바꿈.  
 run.vars = baseball.sub[,run.step==1] #바뀐 것으로 variable을 새로 뽑음.  
 g = lm(salary.log~.,run.vars) #model 적용  
 run.step.aic = extractAIC(g)[2] #AIC 구하기  
 if(run.step.aic < run.aic){  
 run.next = run.step  
 run.aic = run.step.aic  
 break  
 } #만약 AIC가 더 작다면 run.next로 할당해줌. 그리고 run.aic도 바꿔줌  
 }  
 run.current = run.next #run.next를 current에 적용.  
 runs.aic2[k,j]=run.aic   
 }  
 runs2[k,] = run.current #최종적으로 제일 작은 aic를 가진 subset이 골라짐.  
}  
  
runs2

## [,1] [,2] [,3] [,4] [,5] [,6] [,7] [,8] [,9] [,10] [,11] [,12] [,13] [,14]  
## [1,] 0 1 0 1 0 0 0 1 1 1 0 1 1 1  
## [2,] 0 0 1 0 0 1 0 1 0 1 1 1 1 1  
## [3,] 0 0 1 0 0 1 0 1 0 1 1 1 1 1  
## [4,] 0 1 0 1 0 0 1 1 1 1 0 0 1 0  
## [5,] 1 0 1 0 0 1 0 1 0 1 1 0 1 1  
## [,15] [,16] [,17] [,18] [,19] [,20] [,21] [,22] [,23] [,24] [,25] [,26]  
## [1,] 1 1 0 0 0 0 0 0 0 0 0 1  
## [2,] 1 1 0 0 1 1 0 0 1 0 0 1  
## [3,] 1 1 0 0 0 1 0 1 0 0 0 0  
## [4,] 1 0 0 0 1 0 1 0 0 0 0 0  
## [5,] 1 1 0 0 0 0 0 0 0 1 0 0  
## [,27]  
## [1,] 1  
## [2,] 1  
## [3,] 1  
## [4,] 0  
## [5,] 1

runs.aic2

## [,1] [,2] [,3] [,4] [,5] [,6]  
## [1,] -118.87903 -120.86363 -122.74945 -124.19637 -137.24499 -137.5626  
## [2,] -385.67324 -386.28517 -386.95993 -387.20933 -388.90768 -390.4413  
## [3,] -135.02096 -139.72876 -140.44932 -396.11099 -396.21333 -398.0474  
## [4,] -79.16357 -80.88432 -90.07468 -92.01331 -93.19905 -95.1881  
## [5,] -406.45354 -406.96372 -408.92228 -411.03872 -411.17277 -412.3083  
## [,7] [,8] [,9] [,10] [,11] [,12] [,13]  
## [1,] -139.26121 -395.8276 -396.1728 -397.7137 -399.6275 -402.6213 -402.7132  
## [2,] -393.34678 -395.3418 -402.2513 -404.6487 -405.4907 -407.6709 -409.3420  
## [3,] -404.31480 -404.7419 -406.6284 -406.7591 -408.6294 -411.0614 -412.7080  
## [4,] -97.30177 -109.3067 -109.7089 -111.6231 -117.9971 -118.0441 -119.8776  
## [5,] -413.94572 -414.3590 -414.6624 -414.8042 -416.8033 -416.8033 -416.8033  
## [,14] [,15]  
## [1,] -402.7818 -404.2657  
## [2,] -409.3859 -410.8496  
## [3,] -414.5081 -415.7830  
## [4,] -230.9679 -232.1195  
## [5,] -416.8033 -416.8033

##PLOT  
plot(1:(itr\*num.starts),-c(t(runs.aic2)),xlab="Cumulative Iterations",  
 ylab="Negative AIC",ylim=c(100,420),type="n")  
for(i in 1:num.starts) {  
 lines((i-1)\*itr+(1:itr),-runs.aic2[i,]) }



* 원래는 plot의 모양이 곡선의 형태를 띄었으나 여기서는 나머지를 고려하지 않고 좋은 것을 바로 택하기 때문에 plot의 모양이 직각에 가까운 형태를 보이고 있다.

##3-1(b)  
#setting은 위와 동일  
#combination function을 위해 gtools package 이용  
runs3 = matrix(0,num.starts,m)  
runs.aic3 = matrix(0,num.starts,itr)  
library(gtools)

## Warning: package 'gtools' was built under R version 3.6.3

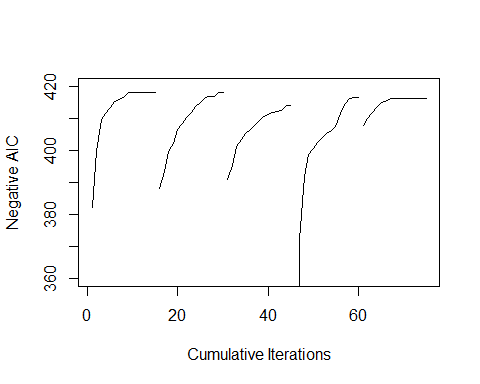
comb <-combinations(m,2) #독립변수에서 2개를 뽑는 경우의 수들  
  
# INITIALIZES STARTING RUNS  
set.seed(1234)   
for(i in 1:num.starts){runs3[i,] = rbinom(m,1,.5)}   
#random으로 열을 뽑는 것을 다섯 번 시행  
  
  
## MAIN  
for(k in 1:num.starts){  
 run.current = runs3[k,]  
   
 # ITERATES EACH RANDOM START  
 for(j in 1:itr){  
 run.vars = baseball.sub[,run.current==1] #1로 선택된 변수들 뽑아내기.  
 g = lm(salary.log~.,run.vars)  
 run.aic = extractAIC(g)[2] #[1]은 equivalent d.f, [2]가 AIC  
 run.next = run.current  
   
 # TESTS ALL MODELS IN THE 2-NEIGHBORHOOD AND SELECTS THE  
 # MODEL WITH THE LOWEST AIC  
 for(i in 1:nrow(comb)){  
 run.step = run.current  
 run.step[comb[i,][1]] = !run.current[comb[i,][1]] #0이라면 1로 1이라면 0으로 바꿈.  
 run.step[comb[i,][2]] = !run.current[comb[i,][2]]  
 run.vars = baseball.sub[,run.step==1] #바뀐 것으로 variable을 새로 뽑음.  
 g = lm(salary.log~.,run.vars) #model 적용  
 run.step.aic = extractAIC(g)[2] #AIC 구하기  
 if(run.step.aic < run.aic){  
 run.next = run.step  
 run.aic = run.step.aic  
 } #만약 AIC가 더 작다면 run.next로 할당해줌. 그리고 run.aic도 바꿔줌  
 }  
 run.current = run.next #run.next를 current에 적용.  
 runs.aic3[k,j]=run.aic   
 }  
 runs3[k,] = run.current #최종적으로 제일 작은 aic를 가진 subset이 골라짐.  
}  
  
## OUTPUT  
runs3 # LISTS OF PREDICTORS

## [,1] [,2] [,3] [,4] [,5] [,6] [,7] [,8] [,9] [,10] [,11] [,12] [,13] [,14]  
## [1,] 1 0 1 0 0 0 0 1 0 1 0 0 1 1  
## [2,] 0 1 1 0 0 1 0 1 0 1 0 0 1 1  
## [3,] 0 1 1 0 0 1 0 1 0 1 0 0 1 1  
## [4,] 0 0 1 0 0 1 0 1 0 1 0 0 1 1  
## [5,] 0 0 1 0 0 1 0 1 0 1 0 1 1 1  
## [,15] [,16] [,17] [,18] [,19] [,20] [,21] [,22] [,23] [,24] [,25] [,26]  
## [1,] 1 1 0 0 0 0 0 0 0 1 1 1  
## [2,] 1 1 0 0 0 0 0 0 0 1 1 1  
## [3,] 1 1 0 0 0 0 0 0 0 1 1 1  
## [4,] 0 0 0 0 0 1 1 1 0 1 1 0  
## [5,] 1 1 0 0 0 1 0 1 0 0 1 1  
## [,27]  
## [1,] 0  
## [2,] 0  
## [3,] 0  
## [4,] 0  
## [5,] 0

runs.aic3 # AIC VALUES

## [,1] [,2] [,3] [,4] [,5] [,6] [,7]  
## [1,] -399.5996 -411.5453 -415.3651 -416.9711 -417.8494 -418.0125 -418.0922  
## [2,] -395.3894 -400.3693 -404.3826 -408.3073 -411.9831 -415.1196 -416.7355  
## [3,] -395.3438 -403.3081 -407.9346 -411.8968 -414.1540 -414.9004 -416.1567  
## [4,] -372.1001 -402.9366 -411.0083 -414.8872 -415.4118 -416.4249 -416.6119  
## [5,] -409.8851 -413.4253 -415.3757 -416.1318 -417.2714 -417.2714 -417.2714  
## [,8] [,9] [,10] [,11] [,12] [,13] [,14]  
## [1,] -418.4511 -418.4511 -418.4511 -418.4511 -418.4511 -418.4511 -418.4511  
## [2,] -418.0000 -418.9472 -418.9472 -418.9472 -418.9472 -418.9472 -418.9472  
## [3,] -416.5907 -416.8036 -418.0818 -418.9421 -418.9472 -418.9472 -418.9472  
## [4,] -416.6119 -416.6119 -416.6119 -416.6119 -416.6119 -416.6119 -416.6119  
## [5,] -417.2714 -417.2714 -417.2714 -417.2714 -417.2714 -417.2714 -417.2714  
## [,15]  
## [1,] -418.4511  
## [2,] -418.9472  
## [3,] -418.9472  
## [4,] -416.6119  
## [5,] -417.2714

##PLOT  
plot(1:(itr\*num.starts),-c(t(runs.aic3)),xlab="Cumulative Iterations",  
 ylab="Negative AIC",ylim=c(360,420),type="n")  
for(i in 1:num.starts) {  
 lines((i-1)\*itr+(1:itr),-runs.aic[i,]) }



* Example 3.3에서의 Plot과 거의 동일함을 알 수 있다.
* Steepest 방법을 사용했기 때문일 수 있다. 만약 steepest로 하지 않고 일정 수만큼만 살펴본다면 결과값이 다를 수 있다.

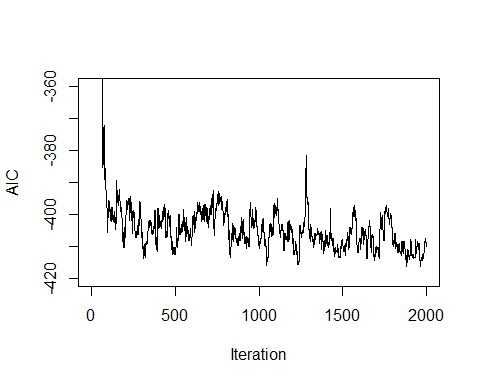
#########################################################################  
### 3.3 SIMULATED ANNEALING  
#########################################################################  
## INITIAL VALUES  
baseball.dat = read.table('baseball.dat',header=TRUE)  
baseball.dat$freeagent = factor(baseball.dat$freeagent)  
baseball.dat$arbitration = factor(baseball.dat$arbitration)  
baseball.sub = baseball.dat[,-1]  
salary.log = log(baseball.dat$salary)  
n = length(salary.log)  
m = length(baseball.sub[1,])  
cooling = c(rep(60,5),rep(120,5),rep(220,5)) #cooling schedule  
tau.start = 10 #시작 온도  
tau = rep(tau.start,15)  
aics = NULL   
  
  
# INITIALIZES STARTING RUN, TEMPERATURE SCHEDULE  
set.seed(1234)  
run = rbinom(m,1,.5)  
run.current = run  
run.vars = baseball.sub[,run.current==1]  
g = lm(salary.log~.,run.vars)  
run.aic = extractAIC(g)[2]  
best.aic = run.aic  
aics = run.aic #여기까지는 3.3과 동일  
for(j in 2:15){tau[j] = 0.9\*tau[j-1]} #온도가 이전의 0.9배로 줄어들도록 설정.  
  
  
## MAIN  
for(j in 1:15){  
  
 # Model에서 더하거나 뺄 predictor를 랜덤으로 선택하고  
 # 더 나은지 확인한다. 더 낫다면 선택.  
 for(i in 1:cooling[j]){  
 pos = sample(1:m,1) #독립변수 중 한 개를 랜덤으로 선택  
 run.step = run.current  
 run.step[pos] = !run.current[pos] #선택된 독립변수를 flip  
 run.vars = baseball.sub[,run.step==1] #flip된 변수 적용  
 g = lm(salary.log~.,run.vars)  
 run.step.aic = extractAIC(g)[2]  
 p = min(1,exp((run.aic-extractAIC(g)[2])/tau[j]))  
 if(run.step.aic < run.aic){  
 run.current = run.step  
 run.aic = run.step.aic}  
 if(rbinom(1,1,p)){ #만약 run.step.aic가 run.aic보다 크다면 p의 확률로 run.step을 채택  
 run.current = run.step  
 run.aic = run.step.aic}  
 if(run.step.aic < best.aic){   
 #지금까지 나온 aic중 제일 작다면 best aic로 선택하고 best predictor subset으로 설정  
 run = run.step  
 best.aic = run.step.aic}  
 aics = c(aics,run.aic)  
 }  
}  
  
## OUTPUT  
run # BEST LIST OF PREDICTORS FOUND

## [1] 0 0 1 0 0 1 0 1 0 1 1 1 1 1 0 0 0 0 0 1 1 1 0 0 0 0 0

best.aic # AIC VALUE

## [1] -416.214

## PLOT OF AIC VALUES  
plot(aics,ylim=c(-420,-360),type="n",ylab="AIC", xlab="Iteration")  
lines(aics)



(1:2001)[aics==min(aics)]

## [1] 1965

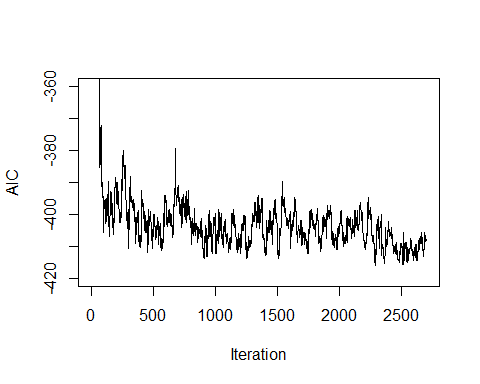
##3-3(a)  
#우선 cooling schedule에서 횟수들을 올려보자.  
baseball.dat = read.table('baseball.dat',header=TRUE)  
baseball.dat$freeagent = factor(baseball.dat$freeagent)  
baseball.dat$arbitration = factor(baseball.dat$arbitration)  
baseball.sub = baseball.dat[,-1]  
salary.log = log(baseball.dat$salary)  
n = length(salary.log)  
m = length(baseball.sub[1,])  
cooling2 = c(rep(100,5),rep(160,5),rep(280,5)) #cooling schedule  
tau.start = 10 #시작 온도  
tau = rep(tau.start,15)  
aics2 = NULL   
  
set.seed(1234)  
run2 = rbinom(m,1,.5)  
run.current = run2  
run.vars = baseball.sub[,run.current==1]  
g = lm(salary.log~.,run.vars)  
run.aic = extractAIC(g)[2]  
best.aic2 = run.aic  
aics2 = run.aic  
for(j in 2:15){tau[j] = 0.9\*tau[j-1]}   
  
for(j in 1:15){  
   
 for(i in 1:cooling2[j]){  
 pos = sample(1:m,1)  
 run.step = run.current  
 run.step[pos] = !run.current[pos]  
 run.vars = baseball.sub[,run.step==1]  
 g = lm(salary.log~.,run.vars)  
 run.step.aic = extractAIC(g)[2]  
 p = min(1,exp((run.aic-extractAIC(g)[2])/tau[j]))  
 if(run.step.aic < run.aic){  
 run.current = run.step  
 run.aic = run.step.aic}  
 if(rbinom(1,1,p)){   
 run.current = run.step  
 run.aic = run.step.aic}  
 if(run.step.aic < best.aic2){   
 run2 = run.step  
 best.aic2 = run.step.aic}  
 aics2 = c(aics2,run.aic)  
 }  
}  
  
## OUTPUT  
run2 # BEST LIST OF PREDICTORS FOUND

## [1] 0 0 1 0 0 1 0 1 0 1 0 0 1 1 1 1 1 0 0 1 0 1 0 1 1 1 0

best.aic2 # AIC VALUE

## [1] -416.0132

## PLOT OF AIC VALUES  
plot(aics2,ylim=c(-420,-360),type="n",ylab="AIC", xlab="Iteration")  
lines(aics2)



* 진동이 훨씬 촘촘해졌음을 알 수 있다.

(1:2701)[aics2==min(aics2)]

## [1] 2287 2288

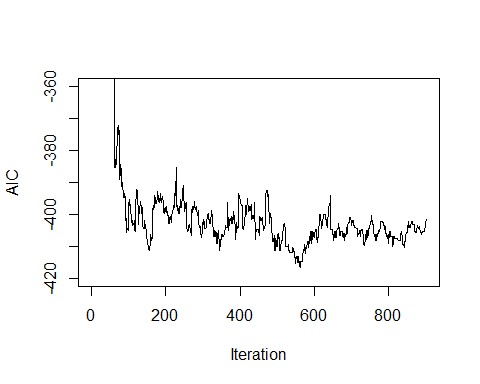
#이번에는 cooling schedule의 횟수를 낮춰보자.  
baseball.dat = read.table('baseball.dat',header=TRUE)  
baseball.dat$freeagent = factor(baseball.dat$freeagent)  
baseball.dat$arbitration = factor(baseball.dat$arbitration)  
baseball.sub = baseball.dat[,-1]  
salary.log = log(baseball.dat$salary)  
n = length(salary.log)  
m = length(baseball.sub[1,])  
cooling3 = c(rep(40,5),rep(60,5),rep(80,5)) #cooling schedule  
tau.start = 10 #시작 온도  
tau = rep(tau.start,15)  
aics3 = NULL   
  
set.seed(1234)  
run3 = rbinom(m,1,.5)  
run.current = run3  
run.vars = baseball.sub[,run.current==1]  
g = lm(salary.log~.,run.vars)  
run.aic = extractAIC(g)[2]  
best.aic3 = run.aic  
aics3 = run.aic  
for(j in 2:15){tau[j] = 0.9\*tau[j-1]}   
  
for(j in 1:15){  
   
 for(i in 1:cooling3[j]){  
 pos = sample(1:m,1)  
 run.step = run.current  
 run.step[pos] = !run.current[pos]  
 run.vars = baseball.sub[,run.step==1]  
 g = lm(salary.log~.,run.vars)  
 run.step.aic = extractAIC(g)[2]  
 p = min(1,exp((run.aic-extractAIC(g)[2])/tau[j]))  
 if(run.step.aic < run.aic){  
 run.current = run.step  
 run.aic = run.step.aic}  
 if(rbinom(1,1,p)){   
 run.current = run.step  
 run.aic = run.step.aic}  
 if(run.step.aic < best.aic3){   
 run3 = run.step  
 best.aic3 = run.step.aic}  
 aics3 = c(aics3,run.aic)  
 }  
}  
  
## OUTPUT  
run3 # BEST LIST OF PREDICTORS FOUND

## [1] 1 0 1 0 0 0 0 1 0 1 1 0 1 1 1 1 1 1 0 0 0 0 0 1 1 1 0

best.aic3 # AIC VALUE

## [1] -416.5059

## PLOT OF AIC VALUES  
plot(aics3,ylim=c(-420,-360),type="n",ylab="AIC", xlab="Iteration")  
lines(aics3)



* 횟수가 줄어든 만큼, 간격이 더욱 듬성듬성하다는 것을 알 수 있다.

(1:901)[aics3==min(aics3)]

## [1] 561 562

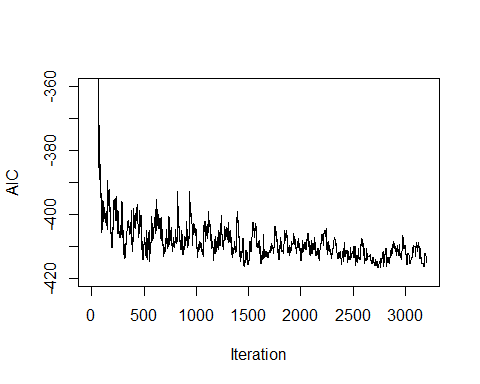
#duration을 높여보자.  
baseball.dat = read.table('baseball.dat',header=TRUE)  
baseball.dat$freeagent = factor(baseball.dat$freeagent)  
baseball.dat$arbitration = factor(baseball.dat$arbitration)  
baseball.sub = baseball.dat[,-1]  
salary.log = log(baseball.dat$salary)  
n = length(salary.log)  
m = length(baseball.sub[1,])  
cooling4 = c(rep(60,8),rep(120,8),rep(220,8)) #cooling schedule  
tau.start = 10   
tau = rep(tau.start,24)  
aics4 = NULL   
  
set.seed(1234)  
run4 = rbinom(m,1,.5)  
run.current = run4  
run.vars = baseball.sub[,run.current==1]  
g = lm(salary.log~.,run.vars)  
run.aic = extractAIC(g)[2]  
best.aic4 = run.aic  
aics4 = run.aic  
for(j in 2:24){tau[j] = 0.9\*tau[j-1]}   
  
for(j in 1:24){  
   
 for(i in 1:cooling4[j]){  
 pos = sample(1:m,1)  
 run.step = run.current  
 run.step[pos] = !run.current[pos]  
 run.vars = baseball.sub[,run.step==1]  
 g = lm(salary.log~.,run.vars)  
 run.step.aic = extractAIC(g)[2]  
 p = min(1,exp((run.aic-extractAIC(g)[2])/tau[j]))  
 if(run.step.aic < run.aic){  
 run.current = run.step  
 run.aic = run.step.aic}  
 if(rbinom(1,1,p)){   
 run.current = run.step  
 run.aic = run.step.aic}  
 if(run.step.aic < best.aic4){   
 run4 = run.step  
 best.aic4 = run.step.aic}  
 aics4 = c(aics4,run.aic)  
 }  
}  
  
## OUTPUT  
run4 # BEST LIST OF PREDICTORS FOUND

## [1] 0 1 1 0 0 0 0 1 0 1 1 1 1 1 1 1 0 0 0 0 0 0 0 0 1 1 0

best.aic4 # AIC VALUE

## [1] -416.4663

## PLOT OF AIC VALUES  
plot(aics4,ylim=c(-420,-360),type="n",ylab="AIC", xlab="Iteration")  
lines(aics4)



* 간격은 촘촘해졌지만 진동의 폭은 비교적 크지 않다는 것을 알 수 있다.

(1:3201)[aics4==min(aics4)]

## [1] 2732 2733 2744 2745 2746 2747 2748 2760 2761

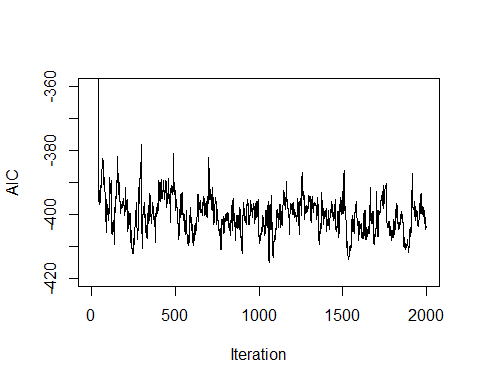
#시작 온도를 높여보자.  
baseball.dat = read.table('baseball.dat',header=TRUE)  
baseball.dat$freeagent = factor(baseball.dat$freeagent)  
baseball.dat$arbitration = factor(baseball.dat$arbitration)  
baseball.sub = baseball.dat[,-1]  
salary.log = log(baseball.dat$salary)  
n = length(salary.log)  
m = length(baseball.sub[1,])  
cooling = c(rep(60,5),rep(120,5),rep(220,5)) #cooling schedule  
tau.start = 20 #10 --> 20  
tau = rep(tau.start,15)  
aics5 = NULL   
  
set.seed(1234)  
run5 = rbinom(m,1,.5)  
run.current = run5  
run.vars = baseball.sub[,run.current==1]  
g = lm(salary.log~.,run.vars)  
run.aic = extractAIC(g)[2]  
best.aic5 = run.aic  
aics5 = run.aic  
for(j in 2:15){tau[j] = 0.9\*tau[j-1]}   
  
for(j in 1:15){  
   
 for(i in 1:cooling[j]){  
 pos = sample(1:m,1)  
 run.step = run.current  
 run.step[pos] = !run.current[pos]  
 run.vars = baseball.sub[,run.step==1]  
 g = lm(salary.log~.,run.vars)  
 run.step.aic = extractAIC(g)[2]  
 p = min(1,exp((run.aic-extractAIC(g)[2])/tau[j]))  
 if(run.step.aic < run.aic){  
 run.current = run.step  
 run.aic = run.step.aic}  
 if(rbinom(1,1,p)){   
 run.current = run.step  
 run.aic = run.step.aic}  
 if(run.step.aic < best.aic5){   
 run5 = run.step  
 best.aic5 = run.step.aic}  
 aics5 = c(aics5,run.aic)  
 }  
}  
  
## OUTPUT  
run5 # BEST LIST OF PREDICTORS FOUND

## [1] 0 1 1 0 0 0 0 1 1 1 0 1 1 1 1 1 0 1 0 0 0 0 0 0 1 1 0

best.aic5 # AIC VALUE

## [1] -415.0361

## PLOT OF AIC VALUES  
plot(aics5,ylim=c(-420,-360),type="n",ylab="AIC", xlab="Iteration")  
lines(aics5)



* 진동의 폭이 크다는 것을 알 수 있다.

(1:2001)[aics5==min(aics5)]

## [1] 1060

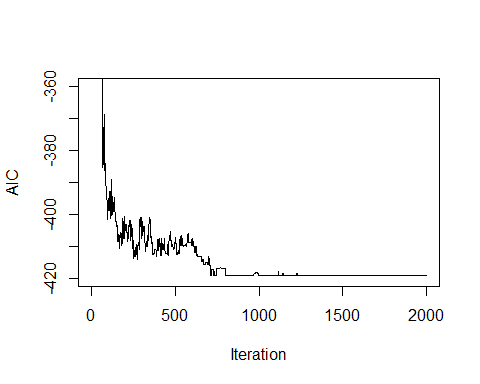
#온도의 강하율을 높여보자.  
baseball.dat = read.table('baseball.dat',header=TRUE)  
baseball.dat$freeagent = factor(baseball.dat$freeagent)  
baseball.dat$arbitration = factor(baseball.dat$arbitration)  
baseball.sub = baseball.dat[,-1]  
salary.log = log(baseball.dat$salary)  
n = length(salary.log)  
m = length(baseball.sub[1,])  
cooling = c(rep(60,5),rep(120,5),rep(220,5)) #cooling schedule  
tau.start = 10   
tau = rep(tau.start,15)  
aics6 = NULL   
  
set.seed(1234)  
run6 = rbinom(m,1,.5)  
run.current = run6  
run.vars = baseball.sub[,run.current==1]  
g = lm(salary.log~.,run.vars)  
run.aic = extractAIC(g)[2]  
best.aic6 = run.aic  
aics6 = run.aic  
for(j in 2:15){tau[j] = 0.7\*tau[j-1]} #0.9에서 0.7로 변경  
  
for(j in 1:15){  
   
 for(i in 1:cooling[j]){  
 pos = sample(1:m,1)  
 run.step = run.current  
 run.step[pos] = !run.current[pos]  
 run.vars = baseball.sub[,run.step==1]  
 g = lm(salary.log~.,run.vars)  
 run.step.aic = extractAIC(g)[2]  
 p = min(1,exp((run.aic-extractAIC(g)[2])/tau[j]))  
 if(run.step.aic < run.aic){  
 run.current = run.step  
 run.aic = run.step.aic}  
 if(rbinom(1,1,p)){   
 run.current = run.step  
 run.aic = run.step.aic}  
 if(run.step.aic < best.aic6){   
 run6 = run.step  
 best.aic6 = run.step.aic}  
 aics6 = c(aics6,run.aic)  
 }  
}  
  
## OUTPUT  
run6 # BEST LIST OF PREDICTORS FOUND

## [1] 0 1 1 0 0 1 0 1 0 1 0 0 1 1 1 1 0 0 0 0 0 0 0 1 1 1 0

best.aic6 # AIC VALUE

## [1] -418.9472

## PLOT OF AIC VALUES  
plot(aics6,ylim=c(-420,-360),type="n",ylab="AIC", xlab="Iteration")  
lines(aics6)



* 기준 온도가 갈수록 급격하게 떨어지기 때문에 추후에는
* exp((run.aic-extractAIC(g)[2])/tau[j])가 매우 크게 나와 무조건 p=1이 선택되게 된다.

(1:2001)[aics6==min(aics6)]

## [1] 712 713 714 715 716 717 726 727

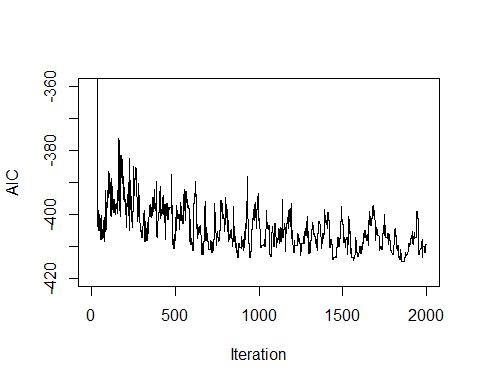
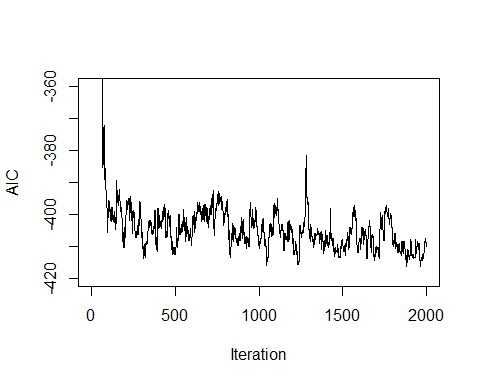
##3-3(b)  
#2-neighborhood  
baseball.dat = read.table('baseball.dat',header=TRUE)  
baseball.dat$freeagent = factor(baseball.dat$freeagent)  
baseball.dat$arbitration = factor(baseball.dat$arbitration)  
baseball.sub = baseball.dat[,-1]  
salary.log = log(baseball.dat$salary)  
n = length(salary.log)  
m = length(baseball.sub[1,])  
cooling = c(rep(60,5),rep(120,5),rep(220,5)) #cooling schedule  
tau.start = 10 #시작 온도  
tau = rep(tau.start,15)  
aics7 = NULL   
  
  
# INITIALIZES STARTING RUN, TEMPERATURE SCHEDULE(이부분은 동일)  
set.seed(1234)  
run7 = rbinom(m,1,.5)  
run.current = run7  
run.vars = baseball.sub[,run.current==1]  
g = lm(salary.log~.,run.vars)  
run.aic = extractAIC(g)[2]  
best.aic7 = run.aic  
aics7 = run.aic  
for(j in 2:15){tau[j] = 0.9\*tau[j-1]}  
  
  
## MAIN  
for(j in 1:15){  
   
 for(i in 1:cooling[j]){  
 pos = sample(1:m,2) #독립변수 중 두 개를 랜덤으로 선택  
 run.step = run.current  
 run.step[pos] = !run.current[pos] #선택된 독립변수를 flip  
 run.vars = baseball.sub[,run.step==1] #flip된 변수 적용  
 g = lm(salary.log~.,run.vars)  
 run.step.aic = extractAIC(g)[2]  
 p = min(1,exp((run.aic-extractAIC(g)[2])/tau[j]))  
 if(run.step.aic < run.aic){  
 run.current = run.step  
 run.aic = run.step.aic}  
 if(rbinom(1,1,p)){  
 run.current = run.step  
 run.aic = run.step.aic}  
 if(run.step.aic < best.aic7){   
 run7 = run.step  
 best.aic7 = run.step.aic}  
 aics7 = c(aics7,run.aic)  
 }  
}  
  
## OUTPUT  
run7 # BEST LIST OF PREDICTORS FOUND

## [1] 0 1 1 0 0 1 0 1 0 1 0 1 1 1 1 1 0 0 0 1 0 0 0 0 1 0 1

best.aic7 # AIC VALUE

## [1] -414.8066

## PLOT OF AIC VALUES  
plot(aics7,ylim=c(-420,-360),type="n",ylab="AIC", xlab="Iteration")  
lines(aics7)



* 왼쪽이 1-neighborhood, 오른쪽이 2-neighborhood
* 2-neighborhood의 경우가 조금 더 진동이 안정적으로 나타나는 모습이다.

(1:2001)[aics7==min(aics7)]

## [1] 1852 1853 1854 1855 1856 1857 1858 1859 1860 1861 1862 1863 1864 1865 1866  
## [16] 1867 1868 1869 1870 1871 1872 1873

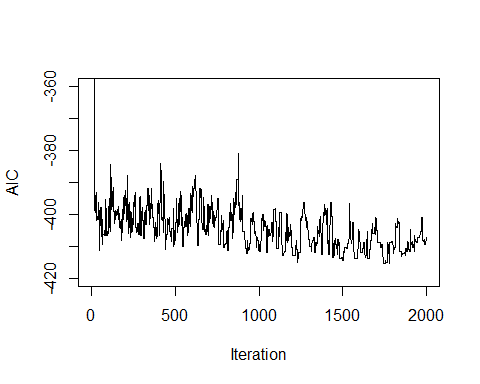
#3-neighborhood  
baseball.dat = read.table('baseball.dat',header=TRUE)  
baseball.dat$freeagent = factor(baseball.dat$freeagent)  
baseball.dat$arbitration = factor(baseball.dat$arbitration)  
baseball.sub = baseball.dat[,-1]  
salary.log = log(baseball.dat$salary)  
n = length(salary.log)  
m = length(baseball.sub[1,])  
cooling = c(rep(60,5),rep(120,5),rep(220,5)) #cooling schedule  
tau.start = 10 #시작 온도  
tau = rep(tau.start,15)  
aics8 = NULL   
  
  
# INITIALIZES STARTING RUN, TEMPERATURE SCHEDULE(이부분은 동일)  
set.seed(1234)  
run8 = rbinom(m,1,.5)  
run.current = run8  
run.vars = baseball.sub[,run.current==1]  
g = lm(salary.log~.,run.vars)  
run.aic = extractAIC(g)[2]  
best.aic8 = run.aic  
aics8 = run.aic  
for(j in 2:15){tau[j] = 0.9\*tau[j-1]}  
  
  
## MAIN  
for(j in 1:15){  
   
 for(i in 1:cooling[j]){  
 pos = sample(1:m,3) #독립변수 중 세 개를 랜덤으로 선택  
 run.step = run.current  
 run.step[pos] = !run.current[pos] #선택된 독립변수를 flip  
 run.vars = baseball.sub[,run.step==1] #flip된 변수 적용  
 g = lm(salary.log~.,run.vars)  
 run.step.aic = extractAIC(g)[2]  
 p = min(1,exp((run.aic-extractAIC(g)[2])/tau[j]))  
 if(run.step.aic < run.aic){  
 run.current = run.step  
 run.aic = run.step.aic}  
 if(rbinom(1,1,p)){  
 run.current = run.step  
 run.aic = run.step.aic}  
 if(run.step.aic < best.aic8){   
 run8 = run.step  
 best.aic8 = run.step.aic}  
 aics8 = c(aics8,run.aic)  
 }  
}  
  
## OUTPUT  
run8 # BEST LIST OF PREDICTORS FOUND

## [1] 0 1 1 0 0 1 0 1 1 1 0 0 1 1 1 1 1 0 0 1 0 0 0 1 1 1 0

best.aic8 # AIC VALUE

## [1] -415.3731

## PLOT OF AIC VALUES  
plot(aics8,ylim=c(-420,-360),type="n",ylab="AIC", xlab="Iteration")  
lines(aics8)



* Best.aic가 2-neighborhood 때보다 약간 더 좋지만 매우 미미한 수준.
* 많은 차이는 모이지 않는다.

(1:2001)[aics8==min(aics8)]

## [1] 1742 1743 1744 1745 1746 1747 1748 1749 1750 1751 1752 1753 1754

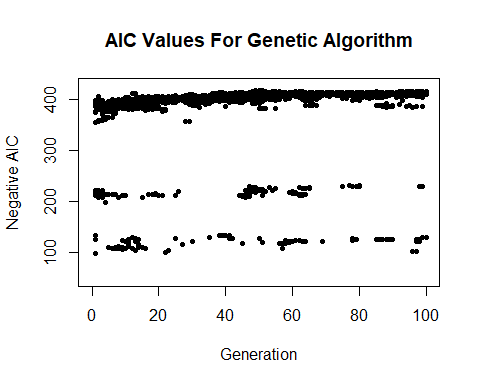
#########################################################################  
### 3.4  
#########################################################################  
## INITIAL VALUES  
baseball.dat = read.table('baseball.dat',header=TRUE)  
baseball.dat$freeagent = factor(baseball.dat$freeagent)  
baseball.dat$arbitration = factor(baseball.dat$arbitration)  
baseball.sub = baseball.dat[,-1]  
salary.log = log(baseball.dat$salary)  
n = length(salary.log)  
m = length(baseball.sub[1,])  
P = 20 #각 generation의 크기  
itr = 100 #generation을 몇 번 돌릴 것인지  
m.rate = .01 #mutation rate  
r = matrix(0,P,1) #Generation의 AIC rank  
phi = matrix(0,P,1)#Generation의 fitness values  
runs = matrix(0,P,m)  
runs.next = matrix(0,P,m)  
runs.aic = matrix(0,P,1)  
aics = matrix(0,P,itr)  
run = NULL  
best.aic = 0  
best.aic.gen = rep(0,itr)  
  
#Starting generation 설정, FITNESS VALUES  
set.seed(1234)   
for(i in 1:P){  
 runs[i,] = rbinom(m,1,.5) #random으로 variable selection  
 run.vars = baseball.sub[,runs[i,]==1]  
 g = lm(salary.log~.,run.vars)  
 runs.aic[i] = extractAIC(g)[2]  
 aics[i,1] = runs.aic[i]  
 if(runs.aic[i] < best.aic){  
 run = runs[i,]  
 best.aic = runs.aic[i]  
 }  
}  
  
r = rank(-runs.aic) #starting genertation의 aic에 rank를 매겨주자.  
phi = 2\*r/(P\*(P+1)) #rank를 이용하여 fitness value 구해줌.  
best.aic.gen[1]=best.aic #starting genertation의 best.aic값.  
  
  
## MAIN  
for(j in 1:itr-1){  
  
 # Generation을 이어가자. 부모 중 첫 번째는 Fitness value를 기준으로 좋은 것을 뽑고  
 # 두 번째는 완전히 랜덤으로 뽑는다.   
 for(i in 1:10){  
 p1 = sample(1:P,1,prob=phi)  
 parent.1 = runs[p1,]   
 parent.2 = runs[sample(c(1:P)[-p1],1),] #중복이 되지 않도록 하자.  
 pos = sample(1:(m-1),1) #분리가 되는 지점을 정해주자.  
 mutate = rbinom(m,1,m.rate) #mutation rate에 기반해서 돌연변이가 일어나는 변수를 선택  
 runs.next[i,] = c(parent.1[1:pos],parent.2[(pos+1):m]) #다음 세대 앞 부분(돌연변이 적용 전)  
 runs.next[i,] = (runs.next[i,]+mutate)%%2 #다음 세대 앞 부분(돌연변이 적용)  
 mutate = rbinom(m,1,m.rate)  
 runs.next[P+1-i,] = c(parent.2[1:pos],parent.1[(pos+1):m]) #다음 세대 뒷 부분(돌연변이 적용 전)  
 runs.next[P+1-i,] = (runs.next[P+1-i,]+mutate)%%2 #다음 세대 뒷 부분(돌연변이 적용)  
 }  
 runs = runs.next  
  
 # New generation에서의 aic와 fitness value 업데이트.  
 for(i in 1:P){  
 run.vars = baseball.sub[,runs[i,]==1]  
 g = lm(salary.log~.,run.vars)  
 runs.aic[i] = extractAIC(g)[2]  
 aics[i,j+1] = runs.aic[i]  
 if(runs.aic[i] < best.aic){  
 run = runs[i,]  
 best.aic = runs.aic[i]  
 }  
 }  
 best.aic.gen[j+1]=best.aic  
 r = rank(-runs.aic)  
 phi = 2\*r/(P\*(P+1))  
}  
  
## OUTPUT  
run # BEST LIST OF PREDICTORS FOUND

## [1] 0 0 1 0 0 1 0 1 0 1 0 0 1 1 1 1 0 0 0 1 0 0 0 1 1 1 0

best.aic # AIC VALUE

## [1] -416.8813

## PLOT OF AIC VALUES  
plot(-aics,xlim=c(0,itr),ylim=c(50,425),type="n",ylab="Negative AIC",  
 xlab="Generation",main="AIC Values For Genetic Algorithm")  
for(i in 1:itr){points(rep(i,P),-aics[,i],pch=20)}



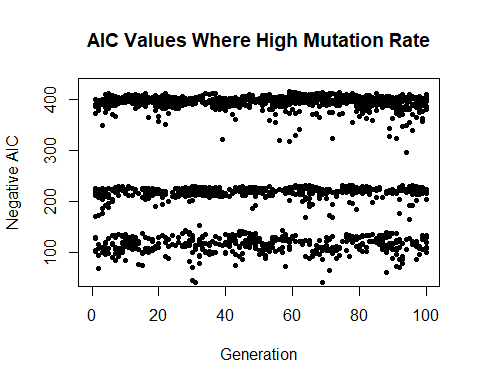
##3-4(a)  
#mutation rates를 조금 높게 설정해보자.  
baseball.dat = read.table('baseball.dat',header=TRUE)  
baseball.dat$freeagent = factor(baseball.dat$freeagent)  
baseball.dat$arbitration = factor(baseball.dat$arbitration)  
baseball.sub = baseball.dat[,-1]  
salary.log = log(baseball.dat$salary)  
n = length(salary.log)  
m = length(baseball.sub[1,])  
P = 20   
itr = 100   
m.rate = .1 #mutation rate 0.01 --> 0.1  
r = matrix(0,P,1)   
phi = matrix(0,P,1)  
runs = matrix(0,P,m)  
runs.next = matrix(0,P,m)  
runs.aic = matrix(0,P,1)  
aics = matrix(0,P,itr)  
run = NULL  
best.aic = 0  
best.aic.gen = rep(0,itr)  
  
set.seed(1234)   
for(i in 1:P){  
 runs[i,] = rbinom(m,1,.5)   
 run.vars = baseball.sub[,runs[i,]==1]  
 g = lm(salary.log~.,run.vars)  
 runs.aic[i] = extractAIC(g)[2]  
 aics[i,1] = runs.aic[i]  
 if(runs.aic[i] < best.aic){  
 run = runs[i,]  
 best.aic = runs.aic[i]  
 }  
}  
  
r = rank(-runs.aic)   
phi = 2\*r/(P\*(P+1))   
best.aic.gen[1]=best.aic   
  
  
for(j in 1:itr-1){  
   
 for(i in 1:10){  
 p1 = sample(1:P,1,prob=phi)  
 parent.1 = runs[p1,]   
 parent.2 = runs[sample(c(1:P)[-p1],1),]  
 pos = sample(1:(m-1),1)  
 mutate = rbinom(m,1,m.rate)  
 runs.next[i,] = c(parent.1[1:pos],parent.2[(pos+1):m])  
 runs.next[i,] = (runs.next[i,]+mutate)%%2  
 mutate = rbinom(m,1,m.rate)  
 runs.next[P+1-i,] = c(parent.2[1:pos],parent.1[(pos+1):m])  
 runs.next[P+1-i,] = (runs.next[P+1-i,]+mutate)%%2  
 }  
 runs = runs.next  
   
 for(i in 1:P){  
 run.vars = baseball.sub[,runs[i,]==1]  
 g = lm(salary.log~.,run.vars)  
 runs.aic[i] = extractAIC(g)[2]  
 aics[i,j+1] = runs.aic[i]  
 if(runs.aic[i] < best.aic){  
 run = runs[i,]  
 best.aic = runs.aic[i]  
 }  
 }  
 best.aic.gen[j+1]=best.aic  
 r = rank(-runs.aic)  
 phi = 2\*r/(P\*(P+1))  
}  
  
## OUTPUT  
run

## [1] 1 1 1 0 0 0 0 1 0 1 0 1 1 1 1 1 0 0 0 1 0 0 0 0 1 1 0

best.aic

## [1] -414.2821

## PLOT OF AIC VALUES  
plot(-aics,xlim=c(0,itr),ylim=c(50,425),type="n",ylab="Negative AIC",  
 xlab="Generation",main="AIC Values Where High Mutation Rate")  
for(i in 1:itr){points(rep(i,P),-aics[,i],pch=20)}



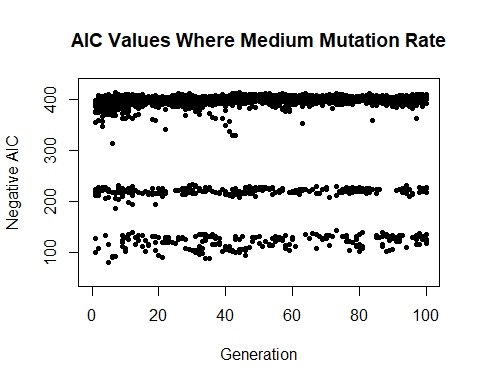
#이번에는 그 중간으로 설정  
baseball.dat = read.table('baseball.dat',header=TRUE)  
baseball.dat$freeagent = factor(baseball.dat$freeagent)  
baseball.dat$arbitration = factor(baseball.dat$arbitration)  
baseball.sub = baseball.dat[,-1]  
salary.log = log(baseball.dat$salary)  
n = length(salary.log)  
m = length(baseball.sub[1,])  
P = 20   
itr = 100   
m.rate = .05 #mutation rate 0.01 --> 0.05  
r = matrix(0,P,1)   
phi = matrix(0,P,1)  
runs = matrix(0,P,m)  
runs.next = matrix(0,P,m)  
runs.aic = matrix(0,P,1)  
aics = matrix(0,P,itr)  
run = NULL  
best.aic = 0  
best.aic.gen = rep(0,itr)  
  
set.seed(1234)   
for(i in 1:P){  
 runs[i,] = rbinom(m,1,.5)   
 run.vars = baseball.sub[,runs[i,]==1]  
 g = lm(salary.log~.,run.vars)  
 runs.aic[i] = extractAIC(g)[2]  
 aics[i,1] = runs.aic[i]  
 if(runs.aic[i] < best.aic){  
 run = runs[i,]  
 best.aic = runs.aic[i]  
 }  
}  
  
r = rank(-runs.aic)   
phi = 2\*r/(P\*(P+1))   
best.aic.gen[1]=best.aic   
  
  
for(j in 1:itr-1){  
   
 for(i in 1:10){  
 p1 = sample(1:P,1,prob=phi)  
 parent.1 = runs[p1,]   
 parent.2 = runs[sample(c(1:P)[-p1],1),]  
 pos = sample(1:(m-1),1)  
 mutate = rbinom(m,1,m.rate)  
 runs.next[i,] = c(parent.1[1:pos],parent.2[(pos+1):m])  
 runs.next[i,] = (runs.next[i,]+mutate)%%2  
 mutate = rbinom(m,1,m.rate)  
 runs.next[P+1-i,] = c(parent.2[1:pos],parent.1[(pos+1):m])  
 runs.next[P+1-i,] = (runs.next[P+1-i,]+mutate)%%2  
 }  
 runs = runs.next  
   
 for(i in 1:P){  
 run.vars = baseball.sub[,runs[i,]==1]  
 g = lm(salary.log~.,run.vars)  
 runs.aic[i] = extractAIC(g)[2]  
 aics[i,j+1] = runs.aic[i]  
 if(runs.aic[i] < best.aic){  
 run = runs[i,]  
 best.aic = runs.aic[i]  
 }  
 }  
 best.aic.gen[j+1]=best.aic  
 r = rank(-runs.aic)  
 phi = 2\*r/(P\*(P+1))  
}  
  
## OUTPUT  
run

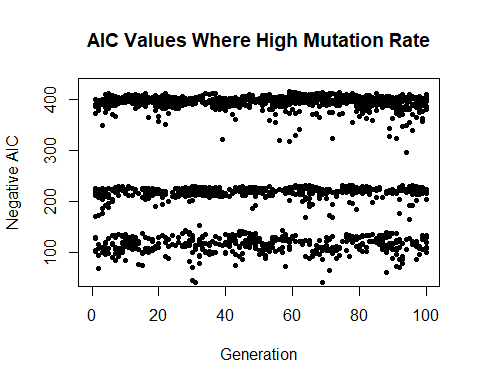
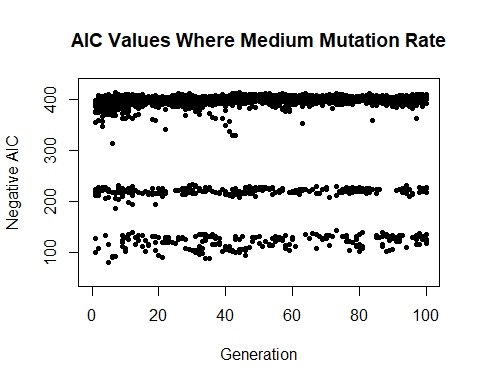
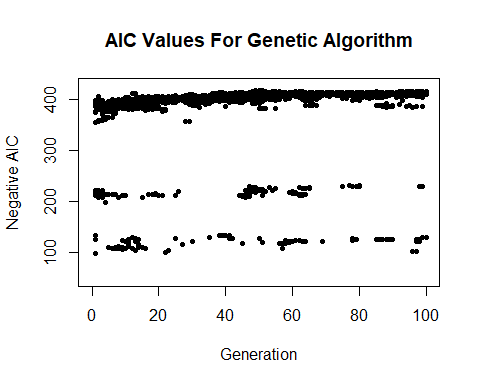
## [1] 0 1 1 1 0 0 0 1 1 1 0 0 1 1 1 1 0 0 0 1 0 0 1 1 1 1 0

best.aic

## [1] -412.5183

## PLOT OF AIC VALUES  
plot(-aics,xlim=c(0,itr),ylim=c(50,425),type="n",ylab="Negative AIC",  
 xlab="Generation",main="AIC Values Where Medium Mutation Rate")  
for(i in 1:itr){points(rep(i,P),-aics[,i],pch=20)}





* Mutation rate를 낮출수록 Converge하고 높일수록 Diverge하는 모습을 알 수 있다.
* 만약 local optimum이 걱정된다면 mutation rate를 조금 높이는 것이 방법이 될 수 있다.

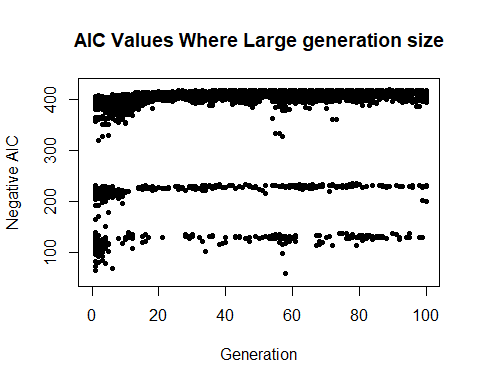
##3-4(b)  
#generation size를 굉장히 크게 해보자.  
baseball.dat = read.table('baseball.dat',header=TRUE)  
baseball.dat$freeagent = factor(baseball.dat$freeagent)  
baseball.dat$arbitration = factor(baseball.dat$arbitration)  
baseball.sub = baseball.dat[,-1]  
salary.log = log(baseball.dat$salary)  
n = length(salary.log)  
m = length(baseball.sub[1,])  
P = 80   
itr = 100   
m.rate = .01  
r = matrix(0,P,1)   
phi = matrix(0,P,1)  
runs = matrix(0,P,m)  
runs.next = matrix(0,P,m)  
runs.aic = matrix(0,P,1)  
aics = matrix(0,P,itr)  
run = NULL  
best.aic = 0  
best.aic.gen = rep(0,itr)  
  
set.seed(1234)   
for(i in 1:P){  
 runs[i,] = rbinom(m,1,.5)   
 run.vars = baseball.sub[,runs[i,]==1]  
 g = lm(salary.log~.,run.vars)  
 runs.aic[i] = extractAIC(g)[2]  
 aics[i,1] = runs.aic[i]  
 if(runs.aic[i] < best.aic){  
 run = runs[i,]  
 best.aic = runs.aic[i]  
 }  
}  
  
r = rank(-runs.aic)   
phi = 2\*r/(P\*(P+1))   
best.aic.gen[1]=best.aic   
  
for(j in 1:itr-1){  
   
 for(i in 1:40){  
 p1 = sample(1:P,1,prob=phi)  
 parent.1 = runs[p1,]   
 parent.2 = runs[sample(c(1:P)[-p1],1),]  
 pos = sample(1:(m-1),1)  
 mutate = rbinom(m,1,m.rate)  
 runs.next[i,] = c(parent.1[1:pos],parent.2[(pos+1):m])  
 runs.next[i,] = (runs.next[i,]+mutate)%%2  
 mutate = rbinom(m,1,m.rate)  
 runs.next[P+1-i,] = c(parent.2[1:pos],parent.1[(pos+1):m])  
 runs.next[P+1-i,] = (runs.next[P+1-i,]+mutate)%%2  
 }  
 runs = runs.next  
   
 for(i in 1:P){  
 run.vars = baseball.sub[,runs[i,]==1]  
 g = lm(salary.log~.,run.vars)  
 runs.aic[i] = extractAIC(g)[2]  
 aics[i,j+1] = runs.aic[i]  
 if(runs.aic[i] < best.aic){  
 run = runs[i,]  
 best.aic = runs.aic[i]  
 }  
 }  
 best.aic.gen[j+1]=best.aic  
 r = rank(-runs.aic)  
 phi = 2\*r/(P\*(P+1))  
}  
  
## OUTPUT  
run

## [1] 1 0 1 0 0 1 0 1 0 1 0 0 1 1 1 1 0 0 0 0 0 0 0 1 1 1 0

best.aic

## [1] -418.9421

## PLOT OF AIC VALUES  
plot(-aics,xlim=c(0,itr),ylim=c(50,425),type="n",ylab="Negative AIC",  
 xlab="Generation",main="AIC Values Where Large generation size")  
for(i in 1:itr){points(rep(i,P),-aics[,i],pch=20)}



* 점의 수가 많아졌지만 전체적인 추이는 비슷하다.

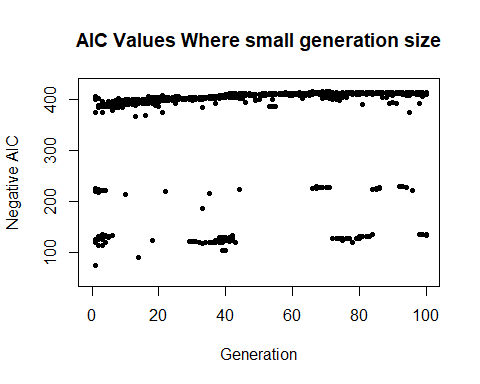
#generation size를 굉장히 작게 해보자.  
baseball.dat = read.table('baseball.dat',header=TRUE)  
baseball.dat$freeagent = factor(baseball.dat$freeagent)  
baseball.dat$arbitration = factor(baseball.dat$arbitration)  
baseball.sub = baseball.dat[,-1]  
salary.log = log(baseball.dat$salary)  
n = length(salary.log)  
m = length(baseball.sub[1,])  
P = 8   
itr = 100   
m.rate = .01  
r = matrix(0,P,1)   
phi = matrix(0,P,1)  
runs = matrix(0,P,m)  
runs.next = matrix(0,P,m)  
runs.aic = matrix(0,P,1)  
aics = matrix(0,P,itr)  
run = NULL  
best.aic = 0  
best.aic.gen = rep(0,itr)  
  
set.seed(1234)   
for(i in 1:P){  
 runs[i,] = rbinom(m,1,.5)   
 run.vars = baseball.sub[,runs[i,]==1]  
 g = lm(salary.log~.,run.vars)  
 runs.aic[i] = extractAIC(g)[2]  
 aics[i,1] = runs.aic[i]  
 if(runs.aic[i] < best.aic){  
 run = runs[i,]  
 best.aic = runs.aic[i]  
 }  
}  
  
r = rank(-runs.aic)   
phi = 2\*r/(P\*(P+1))   
best.aic.gen[1]=best.aic   
  
  
for(j in 1:itr-1){  
   
 for(i in 1:4){  
 p1 = sample(1:P,1,prob=phi)  
 parent.1 = runs[p1,]   
 parent.2 = runs[sample(c(1:P)[-p1],1),]  
 pos = sample(1:(m-1),1)  
 mutate = rbinom(m,1,m.rate)  
 runs.next[i,] = c(parent.1[1:pos],parent.2[(pos+1):m])  
 runs.next[i,] = (runs.next[i,]+mutate)%%2  
 mutate = rbinom(m,1,m.rate)  
 runs.next[P+1-i,] = c(parent.2[1:pos],parent.1[(pos+1):m])  
 runs.next[P+1-i,] = (runs.next[P+1-i,]+mutate)%%2  
 }  
 runs = runs.next  
   
 for(i in 1:P){  
 run.vars = baseball.sub[,runs[i,]==1]  
 g = lm(salary.log~.,run.vars)  
 runs.aic[i] = extractAIC(g)[2]  
 aics[i,j+1] = runs.aic[i]  
 if(runs.aic[i] < best.aic){  
 run = runs[i,]  
 best.aic = runs.aic[i]  
 }  
 }  
 best.aic.gen[j+1]=best.aic  
 r = rank(-runs.aic)  
 phi = 2\*r/(P\*(P+1))  
}  
  
## OUTPUT  
run

## [1] 0 1 1 0 0 0 0 1 0 1 0 0 1 1 1 1 1 0 0 0 0 0 0 1 0 0 0

best.aic

## [1] -414.7214

## PLOT OF AIC VALUES  
plot(-aics,xlim=c(0,itr),ylim=c(50,425),type="n",ylab="Negative AIC",  
 xlab="Generation",main="AIC Values Where small generation size")  
for(i in 1:itr){points(rep(i,P),-aics[,i],pch=20)}



* 역시 수가 적어지기는 했지만, 전체적인 추이는 비슷하다는 것을 알 수 있다. 따라서 generation 수는 너무 많을 필요는 없고 신뢰성을 가질 수 있을 만큼 적절하게 조절하는 것이 좋다.

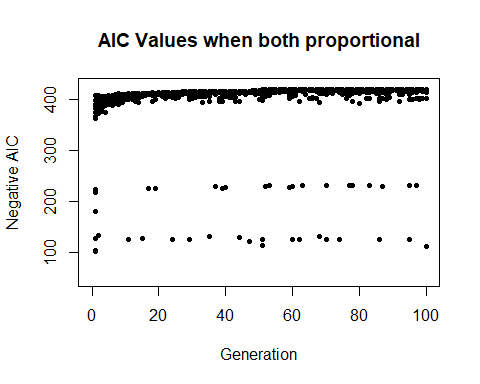
##3-4(c)  
#i는 위에서 했던 방식과 동일하다.  
#ii  
baseball.dat = read.table('baseball.dat',header=TRUE)  
baseball.dat$freeagent = factor(baseball.dat$freeagent)  
baseball.dat$arbitration = factor(baseball.dat$arbitration)  
baseball.sub = baseball.dat[,-1]  
salary.log = log(baseball.dat$salary)  
n = length(salary.log)  
m = length(baseball.sub[1,])  
P = 20   
itr = 100   
m.rate = .01   
r = matrix(0,P,1)   
phi = matrix(0,P,1)  
runs = matrix(0,P,m)  
runs.next = matrix(0,P,m)  
runs.aic = matrix(0,P,1)  
aics = matrix(0,P,itr)  
run = NULL  
best.aic = 0  
best.aic.gen = rep(0,itr)  
  
set.seed(1234)   
for(i in 1:P){  
 runs[i,] = rbinom(m,1,.5)   
 run.vars = baseball.sub[,runs[i,]==1]  
 g = lm(salary.log~.,run.vars)  
 runs.aic[i] = extractAIC(g)[2]  
 aics[i,1] = runs.aic[i]  
 if(runs.aic[i] < best.aic){  
 run = runs[i,]  
 best.aic = runs.aic[i]  
 }  
}  
  
r = rank(-runs.aic)   
phi = 2\*r/(P\*(P+1))   
best.aic.gen[1]=best.aic   
  
  
## MAIN  
for(j in 1:itr-1){  
   
 for(i in 1:10){  
 p1 = sample(1:P,1,prob=phi)  
 parent.1 = runs[p1,]   
 parent.2 = runs[sample(c(1:P)[-p1],1, prob=phi[-p1]),] #prob를 추가해주자.  
 pos = sample(1:(m-1),1)   
 mutate = rbinom(m,1,m.rate)   
 runs.next[i,] = c(parent.1[1:pos],parent.2[(pos+1):m])   
 runs.next[i,] = (runs.next[i,]+mutate)%%2   
 mutate = rbinom(m,1,m.rate)  
 runs.next[P+1-i,] = c(parent.2[1:pos],parent.1[(pos+1):m])   
 runs.next[P+1-i,] = (runs.next[P+1-i,]+mutate)%%2   
 }  
 runs = runs.next  
   
 # New generation에서의 aic와 fitness value 업데이트.  
 for(i in 1:P){  
 run.vars = baseball.sub[,runs[i,]==1]  
 g = lm(salary.log~.,run.vars)  
 runs.aic[i] = extractAIC(g)[2]  
 aics[i,j+1] = runs.aic[i]  
 if(runs.aic[i] < best.aic){  
 run = runs[i,]  
 best.aic = runs.aic[i]  
 }  
 }  
 best.aic.gen[j+1]=best.aic  
 r = rank(-runs.aic)  
 phi = 2\*r/(P\*(P+1))  
}  
  
## OUTPUT  
run # BEST LIST OF PREDICTORS FOUND

## [1] 0 1 1 0 0 1 0 1 0 1 0 1 1 1 1 1 0 0 0 0 0 0 0 0 1 1 0

best.aic # AIC VALUE

## [1] -418

## PLOT OF AIC VALUES  
plot(-aics,xlim=c(0,itr),ylim=c(50,425),type="n",ylab="Negative AIC",  
 xlab="Generation",main="AIC Values when both proportional")  
for(i in 1:itr){points(rep(i,P),-aics[,i],pch=20)}



* 하나를 random으로 설정했을 때보다 더욱 Converge하는 모습을 보인다. 이는, local optimum에는 비교적 더 확실히 다가갈 수 있지만, 반대로 생각하면 local optimum에 빠지기 쉬울 수 있다는 것을 보여주기도 한다.