R명령어들의 속도 평가

이진아[1], 허문열[2]

요 약

최근에 R은 여러 분야에서 많이 사용되고 있다. 특히 모의실험(simulation)이나 통계학 관련 연구에 많이 사용되고 있다. 모의실험을 하는 경우에는 많은 반복으로 인해 R 프로그램의 수행 속도가 매우 중요하다. 또한 데이터마이닝 분야에서도 R을 많이 사용하고 있다. 우리는 데이터 마이닝에서 데이터의 전처리 과정 중 Fayyad & Irani 방법을 사용하여 연속형 변수를 이산화하는 실험을 하였으며, 이를 위해 R을 사용하였다. 이 프로그램은 재귀 함수를 이용하고 이런 과정에서 빈도표 작성, information 계산, 빈도표의 분할, 정지 규칙등의 여러 함수를 사용하게 되어있다. 우리가 작성한 R 코드를 사용하여 UCI DB의 Iono 자료를 (속성이 35개, 사례수가 약 1000개정도) 이산화 하였을 때 7초 이상의 상당한 시간이 소요된다. 반면에 JAVA로 만들어진 Weka에서 똑같은 Fayyad & Irani 방법을 수행했을 때 위와 같은 큰 자료를 이산화하는 속도가 매우 빨라 수행시간은 거의 무시할 만하였다. 이런 차이점을 보고 R 프로그램의 수행 속도를 높이는 방법을 찾게 되었다. 이 본 발표에서는 R 코드 중 시간이 많이 소요되는 것들을 몇 가지 선정하고 이들을 더 효율적으로 만들 수 있는 코드를 작성하여 이들 코드의 수행속도를 비교하였다. 또한 몇 가지 명령에 대해서는 SAS와도 비교하였다.

주요용어 : R, SAS, 계산속도

실험과정과 결과

모든 과정은 10000번 반복실험 하였고 R의 수행 속도를 측정하기 위해 system.time 함수를 사용하였다. 이때 명령문의 기본 형태는 system.time(for(i in 1:10000) 명령문)이 된다. system.time을 수행하면 사용자 cpu, system cpu, elapsed, subproc1, subproc2 의 시간이 측정된다. Windows NT4/2000/XP에서는 subproc1, subproc2 의 시간은 결측값(NA)으로 출력 된다. 아래의 반복실험에서는 cpu속도인 user cpu time을 비교 기준으로 하겠다. 이때 사용된 컴퓨터는 Pentium(4) 1.80GHz, 256MB RAM 이다. 속도비교 명령은 다음과 같다. 또한 몇 가지 명령에 대해서는 SAS명령어도 비교한다.

(1) 균일난수 생성
(2) 벡터 생성
(3) 벡터의 길이 측정
(4) 데이터 분할 및 병합
(5) 행렬의 열 또는 행별 합과 정렬
(6) 행렬의 계산

1 성균관대학교 대학원 통계학과, 석사과정
2 성균관대학교 통계학과, 교수

(1) 균일 난수 생성

R과 SAS에서의 균일 난수 생성 속도를 비교한다. 균일난수 10,000,000개를 생성한다. 이 때 SAS에서도 같은 실험을 하는데 SAS의 균일 난수 생성 함수인 RANUNI 함수와 uniform 함수를 비교한 것이 <표 1> 과 같다. 이 결과에 의하면 R에서의 수행속도는 4.22초인 반면에 SAS에서의 수행속도는 8.51초, 8.54초로 R에 비해 매우 속도가 느리다

<표 1> 균일 난수 생성

프로그램	명령어	결과
R	system.time(runif(10000000))	4.22 초
SAS (RANUNI)	x=RANUNI(1943)	8.51 초
SAS (uniform)	x=uniform(1)	8.54 초

(2) 벡터의 길이 측정

균일 난수 10,000개를 생성하여 100 × 100 행렬에서 행의 개수를 구한다. 이 때 행의 개수를 구하는 명령어는 dim 함수와 length 함수를 이용하여 구할 수 있다. 결과는 <표 2>와 같다. 이 결과를 보면 dim 함수를 사용할 경우는 0.05초, length 함수를 사용 할 경우는 0.08초가 소요된다. dim 함수는 2차원 이상의 경우를 고려하는 함수이고 length는 1차원 벡터에 사용하는 함수라 length의 수행속도가 더 빠를 것이라고 예상하지만 결과를 보면 dim 함수의 수행 속도가 더 빠르다. 1차원 벡터의 경우를 제외하면 dim 함수를 사용하는 것이 더 수행 속도를 빠르게 할 수 있다.

<표 2> 행렬의 행의 개수를 계산 (10,000 번 반복)

명령어	결과
dim(x)[1]	0.05 초
length(x[,1])	0.08 초

```
x<-matrix(runif(10000),nrow=c(100))
```

(3) 벡터 생성

일정한 간격의 수를 갖는 벡터를 생성하는 함수로는 c함수와 seq함수가 있다. 1부터 100까지의 수를 갖는 벡터 a를 생성하는 명령을 10,000번 반복 수행한 결과가 <표 3> 에 나타나 있다. 이 표의 4가지 함수는 모두 1부터 100까지의 요소를 갖는 벡터 a를 생성하는 명령어이다. 1:100 와 c(1:100) 의 수행속도는 별 차이가 나지 않지만 c(1:100)과 seq(1,100)은 0.02초와 0.47초로 상당한 시간 차이를 보인다. 또한 seq(1,100)과 seq(1,100,1)을 보면 같은 seq함수이지만 수행속도는 각각 0.47초와 1.47초가 소요된다. seq(1,100,1)은 1부터 100까지 생성을 할 때 증가량을 1로 정해주는 부분에서 나타난다. seq함수는 기본으로 증가량이 1로 정해져 있다. 이 결과에 의하면 증가량을 1로 하는 벡터의 생성 할 때 seq함수보다는 c함수를 쓰는 것이 더 효율적이다.

<표 3> 벡터 생성 (10,000 번 반복)

명령어	결과
a<-1:100	0.03 초
a<-c(1:100)	0.02 초
a<-seq(1,100)	0.47 초
a<-seq(1,100,1)	1.47 초

(4) 데이터 병합

두 벡터를 합칠 때 사용되는 함수로는 c함수, append 함수, list 함수, array함수가 있다. a와 b 는 각 각 다음과 같다.

a<-1:1000 ; b<-1000:2000

<표 4>는 2개의 벡터를 병합한 것이고 <표 5>는 10개의 벡터를 병합한 것이다. <표 4>를 보면 list 함수의 수행 시간이 가장 짧고 array 함수의 수행 시간이 가장 길다. <표 5>는 10개의 벡터를 병합할 때의 속도 비교이다. 여기서는 R의 c 함수, list 함수, array 함수와 SAS의 수직 병합을 비교 하였다. R의 경우는 <표 4>의 경우와 마찬가지로 list 함수의 수행속도가 가장 빠르며 array 함수의 속도가 가장 느리다. SAS의 경우를 살펴보면 R의 시간에 비해 8.48초라는 상당한 시간을 소요한다. 이 결과에 의하면 벡터를 합치는 경우에는 list 함수를 사용하는 것이 더 효율적인 것을 알 수 있다.

<표 4> 2개의 벡터 병합 (10,000 번 반복)

명령어	결과
x<-c(a,b)	0.23 초
x<-append(a,b)	0.36 초
x<-list(a,b)	0.25 초
x<-array(c(a,b)	0.76 초

<표 5> 10개의 벡터 병합 (10,000 번 반복)

명령어	결과
x<-c(c(a,a,a,a,a,a,a,a,a,a))	1.70 초
x<-list(c(a,a,a,a,a,a,a,a,a,a))	0.91 초
x<-array(c(a,a,a,a,a,a,a,a,a,a)	2.08 초
c=a//a//a//a//a//a//a//a//a//a (SAS)	8.48 초

(5) 자료의 행 또는 열 별 합과 정렬

행렬에서 열 별로 합을 구하거나 정렬을 하는데 apply함수를 이용할 수 있다. apply함수를 사용하는 경우와 따로 sum 과 sort를 이용한 함수를 만들어 행렬의 합과 정렬을 구하는 경우를 비교하여본다. <표 6> 에 프로그램 코드가 나타나 있고, <표 7> 에 수행 결과가 나타나있다. 이 결과를 보면 apply함수를 이용한 합 계산은 6.42초가 소요되는 반면 colSum이라는 새로운 함수를 만들어 사용하는 경우 2.17초가 소요되었다. apply함수를 이용한 정렬은 24.89초, 새로운 함수를 사용한 경우 20.25초의 시간이 소요되었다. 이를 볼 때 apply함수는 코드를 간단히 해주는 장점이 있지만 많은 반복을 할 경우 계산속도가 느려져, 관련 함수를 만들어 사용하는 것이 더 효율적인 것을 알 수 있다.

<표 6> apply 함수를 사용한 합과 정렬과 새로운 프로그램 코드

```
data<-array(sample(100),dim=c(10,10))
xs<-array(0, dim=c(10))
xc<-array(0, dim=c(10,10))

colSum<-function(x){  # sum함수를 이용한 열 별 합 구하는 함수
        for(i in 1:10) xs[i]<-sum(x[,i])
        xs
}

colSort<-function(x){  # sort함수를 이용한 열 별 정렬을 구하는 함수
        for(i in 1:10) xc[,i]<-sort(x[,i])
        xc
}
```

<표 7> 자료의 열 별 합과 정렬 (10,000 번 반복)

명령어	결과
apply(data,2,sum)	6.42 초
x<-colSum(data)	2.17 초
apply(data,2,sort)	24.89 초
colSort(data)	20.25 초

(6) 데이터의 선택

행렬이나 배열에서 행 또는 열의 길이를 알고자 할 때, 해당 항목을 벡터로 만들어 length 함수를 사용하는 경우와 2 차원 배열 자체를 사용하는 경우를 비교하여 보았다. <표 8>에 나타나있는 바와 같이 length를 구하고자 하는 항목을 먼저 벡터로 만들어 사용하는 것이 매우 효율적인 것을 알 수 있다.

(7) 행렬의 연산 속도

비교 연산으로는 덧셈과 행렬의 곱셈이다. 이 경우에는 $n \times 1$의 행렬과 $n \times n$행렬을 벡터, 배열, 행렬의 속성으로 비교한다. <표 9>, <표 10>과 같이 벡터와 행렬의 덧셈과 곱셈에서 벡터성분, 배열성분, 행렬성분 모두 합과 곱셈에서 시간의 차이가 많이 나지 않는다.

<표 8> 데이터의 선택 (10,000 번 반복)

명령어	결과
length(data)	0.00 초
length(x[1,])	0.06 초

x<-matrix(c(1:100),ncol=c(10),byrow = FALSE) ; data<-x[1,]

<표 9> 행렬의 덧셈과 곱셈 (n*1 행렬) (10,000 번 반복)

명령어	결 과	명령문	결과
seq+seq	0.03 초	seq*seq	0.03 초
벡터+벡터	0.03 초	벡터%*%벡터	0.06 초
배열+배열	0.03 초	배열%*%배열	0.05 초
행렬+행렬	0.05 초	행렬%*%행렬	0.06 초

x<-seq(10) ; xv<-as.vector(x) ; xr<-as.array(x) ; xm<-as.matrix(x)

<표 10> 행렬의 덧셈과 곱셈 (n*n 행렬) (10,000 번 반복)

명령어	결 과	명령문	결과
배열+배열	0.07 초	배열%*%배열	0.16 초
행렬+행렬	0.06 초	행렬%*%행렬	0.17 초

x<-seq(100) ; 배열<-array(x,dim=c(10,10)) ; 행렬<-matrix(x,ncol=10)

참고문헌

[1] Fayyad, U. M and Irani, K. B.(1992). On the Handling of Continuous-valued Attributes in Decision Tree Generation, *Machine Learning*, Vol. 8, 87-192

[2] Ihaka, R. and Gentleman, R.(1996). R: A language for data analysis and graphics, *Journal of Computational and Graphical statistics*, 5(3), 299-314, (http://www.r-project.org)

[3] Merz, C. J. and Murphuy, P. M.(1996). UCI Repository of Machine Learning Databases, Department of Information and Computer Science, University of California, Irvine, CA, (http://www.ics.uci.edu/~mlearn/MLRepository.html)

[4] SAS Institute Inc. SAS Version 8.02