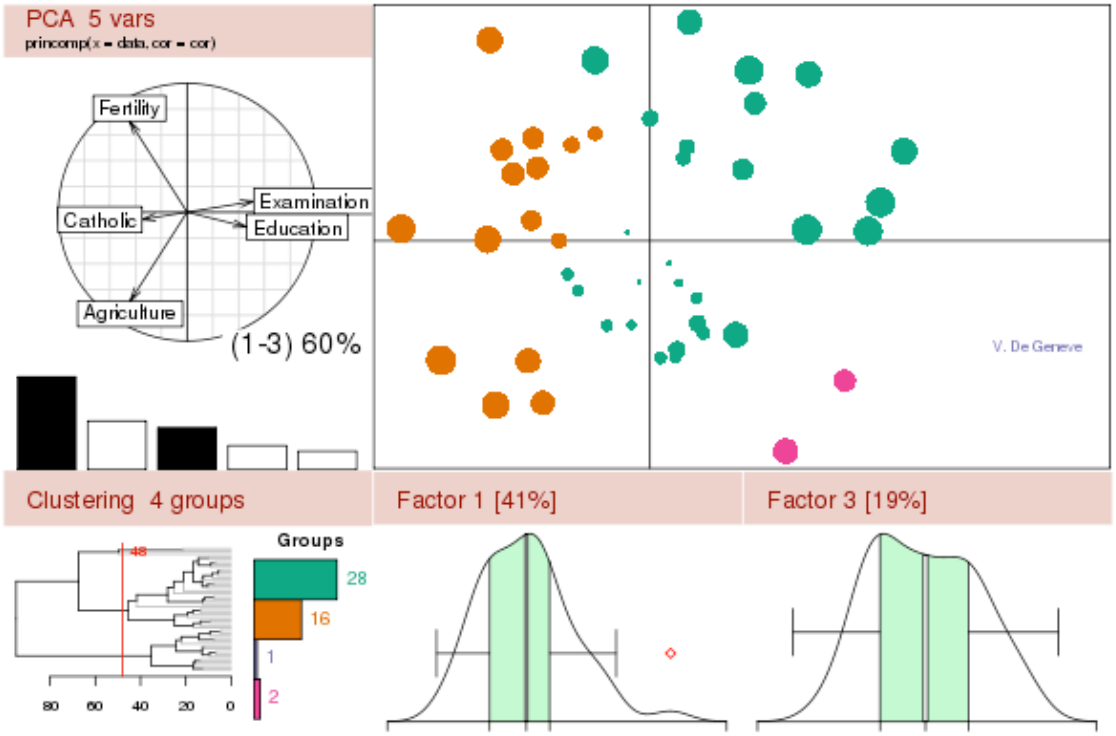
집합과 논리연산

Useful Functions

유 충현
블로그 모음 12탄(http://blog.naver.com/bdboys) • (주)오픈베이스 • 2010년 11월 5일



집합연산

R에서의 집합연산에 대해서 알아보자.

```
> (x=sort(sample(1:10,5)))
[1] 2 3 4 6 8
> (y=sort(sample(1:10,5)))
[1]  2  5  8  9 10
```

1. 합집합 (x∪y)

합집합은 union 함수를 사용한다.

```
> union(x,y)
[1]  2  3  4  6  8  5  9 10
```

그리고 unique 함수를 응용해서 구할 수도 있다. unique함수는 벡터에 대해서 unique한 값을 구한다.

```
> unique(c(x,y))
[1]  2  3  4  6  8  5  9 10
```

2. 교집합 (x∩y)

교집합은 intersect 함수를 사용한다.

```
> intersect(x, y)
[1] 2 8
```

그리고 unique 함수와 sort함수를 응용해서 구할 수도 있다. sort함수는 정렬함다.

```
> sort(unique(c(x,y)))[table(c(x,y))==2]
[1] 2 8
```

또한 다음과 같이 구할 수도 있다.

```
> unique(y[match(x, y, 0)])
[1] 2 8
```

사실 R에서 intersect 함수가 이와 같이 정의되어 있다.

```
> intersect <- function (x, y)
unique(y[match(x, y, 0)])
```

3. 차집합 (x-y)

차집합은 setdiff 함수를 사용한다.

```
> setdiff(x, y)  # x-y
[1] 3 4 6
> setdiff(y,x)  # y-x
[1]  5  9 10
```

그리고 intersect를 응용해서 구할 수도 있다.

```
> intersect(sort(unique(c(x,y)))[table(c(x,y))==1],x)  # x-y
[1] 3 4 6
> intersect(sort(unique(c(x,y)))[table(c(x,y))==1],y)  # y-x
[1]  5  9 10
```

R의 setdiff 함수는 다음과 같이 정의되어 있다.

```
> setdiff <- function (x, y)
unique(if (length(x) || length(y)) x[match(x, y, 0) == 0] else x)
```

4. 원소의 집합포함관계 (a∈x, a∈y)

is.element 함수를 이용해서 특정 원소가 집합에 포함되는지를 검증할 수 있다.

단, 이 함수의 반환값은 논리 벡터인데 그 원소의 개수가 함수의 첫번째 인수의 개수와 동일하다.

```
> a=9
> is.element(a,x)  # a∈x
[1] FALSE
> is.element(a,y)  # x∈y
```

[1] TRUE

그리고 as.logical함수와 sum함수를 응용해서 구할 수도 있다.

```
> as.logical(sum(x==a)) # a∈x
[1] FALSE
> as.logical(sum(y==a))  # a∈y
[1] TRUE
```

다음과 같은 방법을 사용할 수도 있다.

```
> all(!is.na(match(a,x))) # a∈x
[1] FALSE
> all(!is.na(match(a,y)))  # a∈y
[1] TRUE
> all(match(a,x,0)>0) # a∈x
[1] FALSE
> all(match(a,y,0)>0)  # a∈y
[1] TRUE
```

또한 다음처럼 구할 수도 있다.

```
> match(a, x, 0) > 0
[1] FALSE
> match(a, y, 0) > 0
[1] TRUE
```

R에서 is.element 함수가 이와 같이 정의되어 있다.

```
> is.element <- function (el, set)
match(el, set, 0) > 0
```

5. 집합의 집합포함관계 (z⊂x, z⊂y)

is.element 함수를 이용해서 특정 집합이 집합에 포함되는지를 검증할 수 있다.

단, is.element 함수의 반환값은 논리 벡터인데 그 원소의 개수가 함수의

첫번째 인수의 개수와 동일하기 때문에 prod함수를 이용하였다.

```
〉 z=c(2,8)
〉 as.logical(prod(is.element(z,x)))  # z⊂x
[1] TRUE
〉 as.logical(prod(is.element(z,y)))  # z⊂y
[1] TRUE
〉 as.logical(prod(is.element(x,z)))  # x⊂z
[1] FALSE
```

그리고 as.logical함수와 prod함수, intersect함수를 응용해서 구할 수도 있다.

```
〉 as.logical(prod(intersect(x, z)==z))  # z⊂x
[1] TRUE
〉 as.logical(prod(intersect(y, z)==z))  # z⊂y
[1] TRUE
〉 as.logical(prod(intersect(x, z)==x)) # 원소의 개수가 배가 아니면 경고가 발생한다.
Warning message:
longer object length
 is not a multiple of shorter object length in: intersect(x, z) == x
[1] FALSE
```

다음을 응용할 수도 있다.

```
〉 match(z,x,0)〉0
[1] TRUE TRUE
〉 match(z,y,0)〉0
[1] TRUE TRUE
```

```
> match(x,z,0)>0
[1]  TRUE FALSE FALSE FALSE  TRUE

> all(match(z,x,0)>0)  # z⊂x
[1] TRUE
> all(match(z,y,0)>0)  # z⊂y
[1] TRUE
> all(match(x,z,0)>0)  # x⊂z
[1] FALSE

> all(is.element(z,x))  # z⊂x
[1] TRUE
> all(is.element(z,y))  # z⊂y
[1] TRUE
> all(is.element(x,z))  # x⊂z
[1] FALSE
```

all 함수는 인수의 값이 모두 TRUE일 경우에만 TRUE를 반환하고 아니면 FALSE를 반환한다.

6. is.element을 응용한 교집합

is.element을 응용하면 sort(unique(c(x,y)))[table(c(x,y))==2]을

다음과 같이 간략화 시킬 수 있다.

```
> x[is.element(x,y)]  # x∩y
[1] 2 8
> y[is.element(y,x)]  # y∩x
[1] 2 8
```

7. is.element을 응용한 차집합

is.element을 응용하면 intersect(sort(unique(c(x,y)))[table(c(x,y))==1],x)을 다음과 같이 간략화 시킬 수 있다.

```
> x[!is.element(x,y)]  # x-y
[1] 3 4 6
> y[!is.element(y,x)]  # y-x
[1]  5  9 10
```

8. 집합의 상등(x=y)

setequal을 이용하여 집합의 상등을 알아볼 수 있다.

```
> setequal(x,y)
[1] FALSE
> setequal(x,x)
[1] TRUE
```

setequal 함수는 다음과 같이 정의되어 있다.

```
> setequal <- function (x, y)
all(c(match(x, y, 0) > 0, match(y, x, 0) > 0))
```

이 함수를 수식으로 표현하자면 다음과 같다.

if x⊂y and y⊂x then x=y

othwise x≠y

다음처럼 구할 수도 있다.

```
> all(sort(x)==sort(y))
[1] FALSE
> all(sort(x)==sort(x))
[1] TRUE
```

> all(sort(x)==sort(z)) # 원소의 개수가 배가 아니면 경고가 발생한다.

[1] FALSE

Warning message:

longer object length

 is not a multiple of shorter object length in: sort(x) == sort(z)

warning message가 번거롭다면 options의 warn 값을 음수로 바꾸면 나오지 않는다. 그러나 이 방법은 권장하지 않는다. 차라리 다음과 같이 ifelse를 이용해서 예외처리를 해주면 된다.

> ifelse(length(x)!=length(z),FALSE,all(sort(x)==sort(z)))

[1] FALSE

앞서 집합포함관계에서도 warning message가 출력되었는데 이와 같이 처리할 수 있겠다.

논리연산자의 최적화에 대하여

논리연사자는 일반적으로 조건문 안에서 사용된다. 그러므로 if문이나 ifelse 함수등과 자주사용되거나 벡터등의 Subset을 구하기 위해서 [,] 안에서 사용되기도 한다.

이번에는 논리연산자 &&,||,&,|에 대해서 알아보자.

&&, & 는 논리 AND, ||, | 는 논리 OR를 의미한다. &&,||와 &,|의 차이점은 앞의 것은 피연산자가 벡터도 가능한데 뒤의 것은 스칼라 형태의값만 가능하다. 즉, 원소의 개수가 1인 피연산자만 사용할 수 있다는 점이다. 물론 이 경우는 피연산자가 데이터 객체일 경우이고, 피연산자가 수식일 경우에는

그 결과값에 대해서 동일하게 적용된다.

이번에 이야기할 주제는 이것이 아니라, 조건문에 있어서 논리 AND와 논리 OR를 R 인터프리터가 번역할 때의 최적화에 대한 것이다.

이 연산자들은 연산자를 기준으로 좌측에서 우측으로 번역되어진다. 그러면 논리 AND와 논리 OR의 특성에 대해서 짚어 보자. 논리 AND는 두 값이 모두 TRUE이어야만 결과가 TRUE가 된다. 논리 OR는 두 중 하나만 TRUE이어도 결과가 TRUE가 된다.

이 특징을 적절히 활용해서 번역의 최적화를 적용시킨 것이다. 예를 들어 좌변의 값이 FALSE인 논리 AND 연산은 우변의 값에 상관없이 FALSE가 된다. 그러므로 R은 오른쪽의 값은 번역하지 않는다. 그만큼 연산의 양이 줄게된다. 또 좌변의 값이 TRUE인 논리 OR 연산은 우변의 값에 상관없이 TRUE가 된다. 그러므로 R은 오른쪽의 값은 번역하지 않는다.

다음 예를 보자.
문자형을 갖는 벡터(스칼라) x를 만든다.

```
〉 (x="1")
[1] "1"
```

좌변의 값이 FALSE이므로 우변의 수식을 수행하지 않으며, 조건의 결과가 FALSE이므로 print(x)도 수행되지 않았다. 여전히 x는 문자형이다.

```
〉 if (is.numeric(x) && (x=as.numeric(x))) print(x)
〉 x
[1] "1"
```

좌변의 값이 TRUE이므로 우변의 수식을 수행하여 x가 수치형이 되었으며 조건의 결과가 TRUE여서 print(x)도 수행되었다.

```
> if (is.character(x) && (x=as.numeric(x))) print(x)
[1] 1
```

다시 문자형을 갖는 벡터(스칼라) x를 만든다.

```
> x="1"
```

좌변의 값이 TRUE이므로 우변의 수식을 수행하지 않으며, 되었으며 조건의 결과가 TRUE여서 print(x)도 수행되었다. 여전히 x는 문자형이다.

```
> if (is.character(x) || (x=as.numeric(x))) print(x)
[1] "1"
```

좌변의 값이 FALSE이므로 우변의 수식을 수행하여 x가 수치형이 되었으며 조건의 결과가 TRUE여서 print(x)도 수행되었다.

```
> if (is.numeric(x) || (x=as.numeric(x))) print(x)
[1] 1
```

다시 문자형을 갖는 벡터(스칼라) x를 만든다.

```
> x="1"
```

좌변의 값이 FALSE이므로 우변의 수식을 수행하지 않으며, 조건의 결과가 FALSE이므로 print(x)도 수행되지 않았다. 여전히 x는 문자형이다. &&의 연산과 동일하다.

```
> if (is.numeric(x) & (x=as.numeric(x))) print(x)
> x
[1] "1"
```

좌변의 값이 TRUE이므로 우변의 수식을 수행하지 않으며, 되었으며 조건의 결과가 TRUE여서 print(x)도 수행되었다. 여전히 x는 문자형이다. ||의 연산과 동일하다.

```
> if (is.character(x) | (x=as.numeric(x))) print(x)
[1] "1"
```

다시 문자형을 갖는 벡터 y를 만든다.

```
> (y=c("1","2"))
[1] "1" "2"
```

좌변의 값이 FALSE이므로 우변의 수식을 수행하지 않으며, 조건의 결과가 FALSE 이므로 print(y)도 수행되지 않았다. 여전히 y는 문자형이다.

```
> if (is.numeric(y) && (y=as.numeric(y))) print(y)
> y
[1] "1" "2"
```

좌변의 값이 TRUE이므로 우변의 수식을 수행하지 않으며, 되었으며 조건의 결과가 TRUE여서 print(y)도 수행되었다. 여전히 y는 문자형이다.

```
> if (is.character(y) || (y=as.numeric(y))) print(y)
[1] "1" "2"
```

좌변의 값이 FALSE이므로 우변의 수식을 수행하지 않으며, 조건의 결과가 FALSE 이므로 print(y)도 수행되지 않았다. 여전히 y는 문자형이다. 그러나 Warning message가 발생하였다. & 연산자는 벡터의 연산에 사용할 수 없다.

```
> if (is.numeric(y) & (y=as.numeric(y))) print(y)
Warning message:
the condition has length > 1 and only the first element will be used in:
if (is.numeric(y) & as.numeric(y)) print(y)
```

좌변의 값이 TRUE이므로 우변의 수식을 수행하지 않으며, 되었으며 조건의 결과가 TRUE여서 print(y)도 수행되었다. 여전히 y는 문자형이다. 그러나 Warning message가 발생하였다. | 연산자는 벡터의 연산에 사용할 수 없다.

```
> if (is.character(y) | (y=as.numeric(y))) print(y)
[1] "1" "2"
Warning message:
the condition has length > 1 and only the first element will be used in:
if (is.character(y) | as.numeric(y)) print(y)
```