R Data Representations

Code

###Concepts of Data Representation

ideas discussed in van der Loo and de Jonge, chapter 3

It's important to remember that computers fundamentally work with binary logic, they work only with two values, 0 and 1.

Any other type of information we want to encode has to be written as strings of 0 and 1 values, as numbers in base 2, with a limited number of digits

It helps to think for a minute of what ordinary base 10 numbers actually mean.

A number such as 7345 really means

```
7*1000 + 3*100 + 4*10 +5
or
7*10^3 + 3^10^2 + 4*10^1 + 5 * 10^0
```

Most of the time, we use base 10, so the consecutive numbers in the value 7345 are refering to multiples of powers of 10

Using the same type of logic, a binary number like 1011 is

```
1*2^3 + 0*2^2 + 1*2^1 + 1^2^0
or
1*8 + 0*4 + 1*2 + 1*1 = 11   (in base 10)
```

So a string of binary digits can encode integer values.

If we use N digits to encode an integer, the maximum value we can encode in base 10 is

2^N-1

I learned on 8 bit computer processors, that manipulated 8 bits (bit-> Binary Digit, 1 Byte is 8 bits) at a time. We then got these amazing 16 bit processors, then 32 and the current standard of 64 bits in most PCs.

Hide

```
2^8-1
```

```
[1] 255
```

Hide

```
2^16-1
```

```
[1] 65535
```

Hide

```
2^32-1
```

```
[1] 4294967295
```

Hide

```
2^64-1
```

```
[1] 1.844674e+19
```

This is 255, 56K, 4.3 Gig, and 1.8 x 10^19 (no idea what 10^19 is)

Every other variable we used in a computer system has to be encoded using integer values.

It is common to use one bit of an integer (the first one) as a “sign bit” so that we can have positive and negative integers

In particular:

1.) Binary variables in R (TRUE and FALSE) are stored as 1 and 0 values.

2.) Integers are stored as a sign bit and N binary bits, so if you have an integer count that exceeds the maximum value, R will typically switch to using floating point number to represent the value.

3.) Floating Point Numbers are represented as a sign bit, an integer mantissa and an integer exponent

```
        x=   (-1)^(Sign bit) x (integer Mantissa) x 2^(integer exponent)
```

4.) Characters are encoded using integers, so each character (a,b, A, 1, $,, etc) has an integer code. There are many different integer encodings currently in use in the world, we’ll talk more about that later.

5.) Sound is usually encoded as a series of integer values, with a specified number of values per second (called the sampling rate)

6.) Pictures are usually a matrix of integer values. A black and white image is a single matrix, a color image is typically a set of three matrices, often one for each color channel of a video monitor (RGB=red,green and blue)

7.) Video is a serices of Pictures, plus sound, so 3 matrices per frame of the video and the audio channel.

Implications of this in calculations

Binary (TRUE/FALSE) values

Hide

```
(1>0)*4
```

```
[1] 4
```

Hide

```
sum(c(1,2,3,4,5)>2.5)
```

```
[1] 3
```

Integers

Hide

```
.Machine$integer.max
```

```
[1] 2147483647
```

Hide

```
t=2147483647+20
print(t)
```

```
[1] 2147483667
```

Hide

```
t-20.5
```

```
[1] 2147483647
```

Hide

```
class(t)
```

```
[1] "numeric"
```

This problem is called “Abstraction Leakage”

As humans we know what we mean when we write 2147483647 + 20

But R tries to represent (abstract) this using only binary bits. Since the result of this sum is greater than the largest integer value the operating system works with, R represents this as a floating point variable t. When we subtract 20.5 from t, R cannot represent the last digit of the calculation in a floating point number and it rounds off.

So we have a math error due to the limits of the computer system.

Comparing floating point values

Hide

```
( (1.0-.9)==0.1)
```

```
[1] FALSE
```

Okay, that’s odd

Hide

```
(1.0-0.9)-0.1
```

```
[1] -2.775558e-17
```

Hide

```
.Machine$double.eps
```

```
[1] 2.220446e-16
```

The double.eps is the claimed precision of floating point (double) values for my computer in R.

Does this mean that the error in a floating point calculation is around 2 x 10^(-16)?

Well, no, errors propagate

The book has an example, solving a linear equation like

Ax=b,

where A, x and b are matrices

so X= (A^-1)b

The error in x over x (error x)/ x is the fractional error in x

(error x)/x <= kappa(A) (error b)/b

so the error in our answer (x) increases by a factor kappa(A) relative to the input (b)

kappa(A) is called the condition number of the matrix A, and ranges from 1 to infinity

the number of valid decimal places in x is typically U-log(kappa(A)) where U is the negative of the exponent of the machine precision (16 in the current case)

Suppose A is variance-covariance matrix of values from the mtcars set

A variance covariance matrix has the variances of the variables along the diagonal, with the covariances along the off-diagonal elements

Hide

```
data(mtcars)
str(mtcars)
```

```
'data.frame':   32 obs. of  11 variables:
 $ mpg : num  21 21 22.8 21.4 18.7 18.1 14.3 24.4 22.8 19.2 ...
 $ cyl : num  6 6 4 6 8 6 8 4 4 6 ...
 $ disp: num  160 160 108 258 360 ...
 $ hp  : num  110 110 93 110 175 105 245 62 95 123 ...
 $ drat: num  3.9 3.9 3.85 3.08 3.15 2.76 3.21 3.69 3.92 3.92 ...
 $ wt  : num  2.62 2.88 2.32 3.21 3.44 ...
 $ qsec: num  16.5 17 18.6 19.4 17 ...
 $ vs  : num  0 0 1 1 0 1 0 1 1 1 ...
 $ am  : num  1 1 1 0 0 0 0 0 0 0 ...
 $ gear: num  4 4 4 3 3 3 3 4 4 4 ...
 $ carb: num  4 4 1 1 2 1 4 2 2 4 ...
```

Hide

```
cor(mtcars[,c('mpg','hp','disp')])
```

```
            mpg         hp       disp
mpg   1.0000000 -0.7761684 -0.8475514
hp   -0.7761684  1.0000000  0.7909486
disp -0.8475514  0.7909486  1.0000000
```

Hide

```
A=cov(mtcars[,c('mpg','hp','disp')])
A
```

```
           mpg        hp       disp
mpg    36.3241 -320.7321  -633.0972
hp   -320.7321 4700.8669  6721.1587
disp -633.0972 6721.1587 15360.7998
```

Hide

```
kappa(A)
```

```
[1] 1886.004
```

Hide

```
16-log10(kappa(A))
```

```
[1] 12.72446
```

Factors

R uses factor variables to indicate categorical or ordinal variables

Ordinal variables are ordered categories.

The state a person lives in (NY, Penn, Mass etc) is a category. It is not ordered, it makes no sense to ask if NY>Penn (although we could order them alphabeically, by population or land area or average income or something.)

Military ranks are ordered though

private<corporal<seargent<lieutant<captain<major<colonel<general

so while these are categories they have a sequence, corporal is closer to private than to colonel

baby<toddler<grade schooler<high schooler<young adult<middle aged<elderly

In R factors are coded as integers, but each integer corresponds to a named category. The use of integer saves space over using texts, but one should not carry out math on factors. They are used as inputs in statistical models, and can be counted (how many generals are there in Cuba?)

Hide

```
names=c('boy','boy','boy','seagull','seagull','boy')
name_f=as.factor(names)
summary(name_f)
```

```
    boy seagull
      4       2
```