John Myles White

"Who refuses to do arithmetic is doomed to talk nonsense."

Using Sparse Matrices in R

By John Myles White on 10.31.2011

Introduction

I've recently been working with a couple of large, extremely sparse data sets in R. This has pushed me to spend some time trying to master the CRAN packages that support sparse matrices. This post describes three of them: the Matrix, slam and glmnet packages. The first two packages provide data storage classes for sparse matrices, while the last package can perform GLM analyses on data stored in a sparse matrix.

The Matrix Package

The first package I worked with that provides a sparse matrix implementation is Doug Bates' Matrix package. In fact, matrices of class `Matrix` can be switched between full and sparse representations dynamically, but I'll focus on forcing the use of a sparse representation.

As a first example, it's helpful to generate a 1000×1000 matrix of zeros using the `matrix` class and then another 1000×1000 matrix of zeros using the `Matrix` class:

```
library('Matrix')

m1 <- matrix(0, nrow = 1000, ncol = 1000)
m2 <- Matrix(0, nrow = 1000, ncol = 1000, sparse = TRUE)

object.size(m1)
# 8000200 bytes
object.size(m2)
# 5632 bytes
```

As you can see, the representation of a perfectly sparse matrix using the `Matrix` class uses more than one thousand times as much RAM. And you can check how much more RAM would be required if both matrices had exactly one non-zero entry:

```
m1[500, 500] <- 1
m2[500, 500] <- 1

object.size(m1)
# 8000200 bytes
object.size(m2)
# 5648 bytes
```

The full matrix representation does not change in size because all of the zeros are being represented explicitly, while the sparse matrix is conserving that space by representing only the non-zero entries. With a simple subtraction, we find that adding one additional non-zero entry increases the storage requirement by 16 bytes. We can conclude that setting all of the entries to non-zero values would require `5632 + 16 *`

`1000 * 1000` bytes, which is 16,005,632 bytes or almost exactly twice the amount of storage required to use the full representation implemented by the `matrix` class.

Of course, the take away lesson is that sparse matrices are very efficient if your data is sparse and mildly wasteful if your data is not sparse.

Beyond simple initialization and assignment operations, we can perform quite a few other operations on objects of class `Matrix`, including vector multiplication, matrix addition and subtraction and transposition:

```
m2 %*% rnorm(1000)
m2 + m2
m2 - m2
t(m2)
```

In addition you can do binding operations using objects of class `Matrix` with the cBind and rBind functions:

```
m3 <- cBind(m2, m2)
nrow(m3)
ncol(m3)

m4 <- rBind(m2, m2)
nrow(m4)
ncol(m4)
```

Other operations are probably supported, but I haven't need them so far in my work.

The slam Package

An alternative to the Matrix package is the slam package by Kurt Hornik and others. The sparse matrices generated using this package can be noticeably smaller than those generated by the Matrix package in some cases.

For example, the same perfectly sparse matrix using the slam package requires only 1,032 bytes of space:

```
library('slam')

m1 <- matrix(0, nrow = 1000, ncol = 1000)
m2 <- simple_triplet_zero_matrix(nrow = 1000, ncol = 1000)

object.size(m1)
# 8000200 bytes
object.size(m2)
# 1032 bytes
```

In short, you can make your data fit in 1/5 the amount of RAM under some circumstances. And we can once again check how much additional RAM you need for every new non-zero entry:

```
# BUG HERE
m1[500, 500] <- 1
m2[500, 500] <- 1

object.size(m1)
object.size(m2)
```

Finally, all of the same matrix operations available with the `Matrix` class work on objects implemented by slam:

```
m2 %*% rnorm(1000)
m2 + m2
m2 - m2
t(m2)
```

The glmnet Package

Of course, being able to load sparse data into RAM is only interesting if we can analyze it statistically. For me, this usually means that I fit some sort of GLM to the data: most of the time either linear or logistic regression — preferably with some sort of regularization.

Thankfully, the glmnet package allows full and sparse matrices to be used without any code changes. You simply need to provide arguments that use the Matrix package's `Matrix` class to have glmnet switch over to computing with sparse matrices:

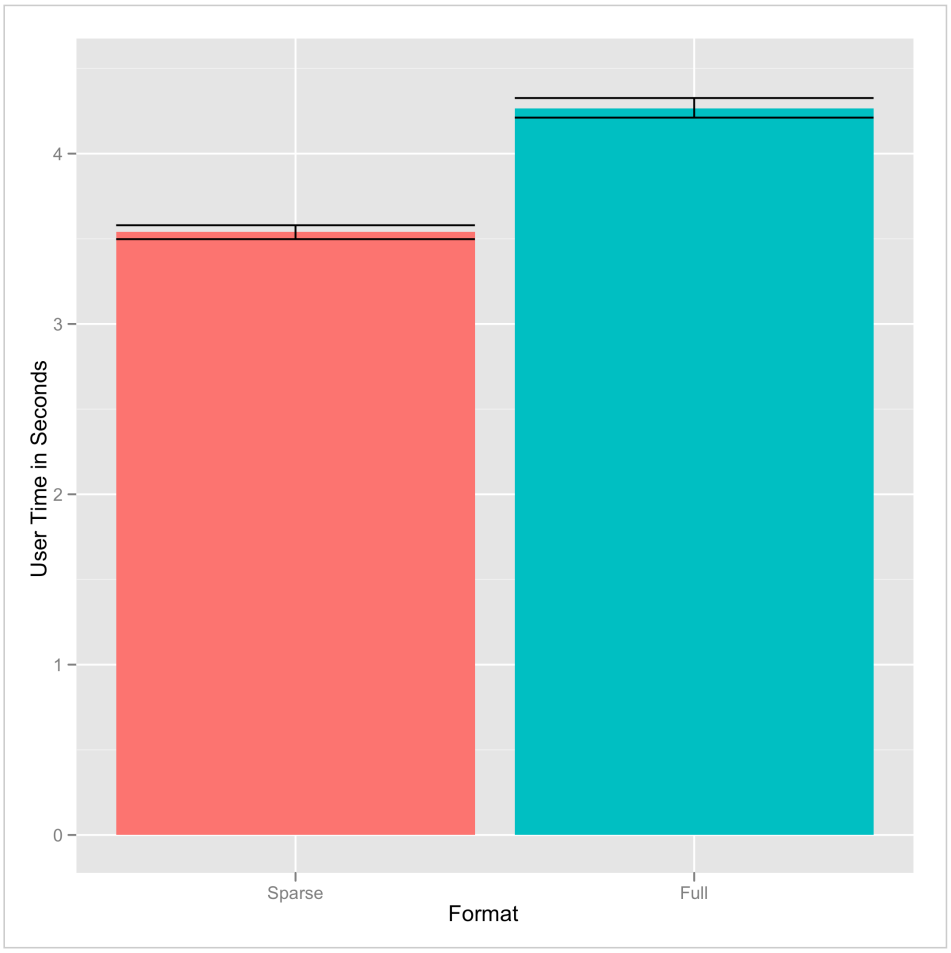
```
library('Matrix')
library('glmnet')

n <- 10000
p <- 500

x <- matrix(rnorm(n * p), n, p)
iz <- sample(1:(n * p),
             size = n * p * 0.85,
             replace = FALSE)
x[iz] <- 0

object.size(x)

sx <- Matrix(x, sparse = TRUE)

object.size(sx)

beta <- rnorm(p)

y <- x %*% beta + rnorm(n)

glmnet.fit <- glmnet(x, y)
```

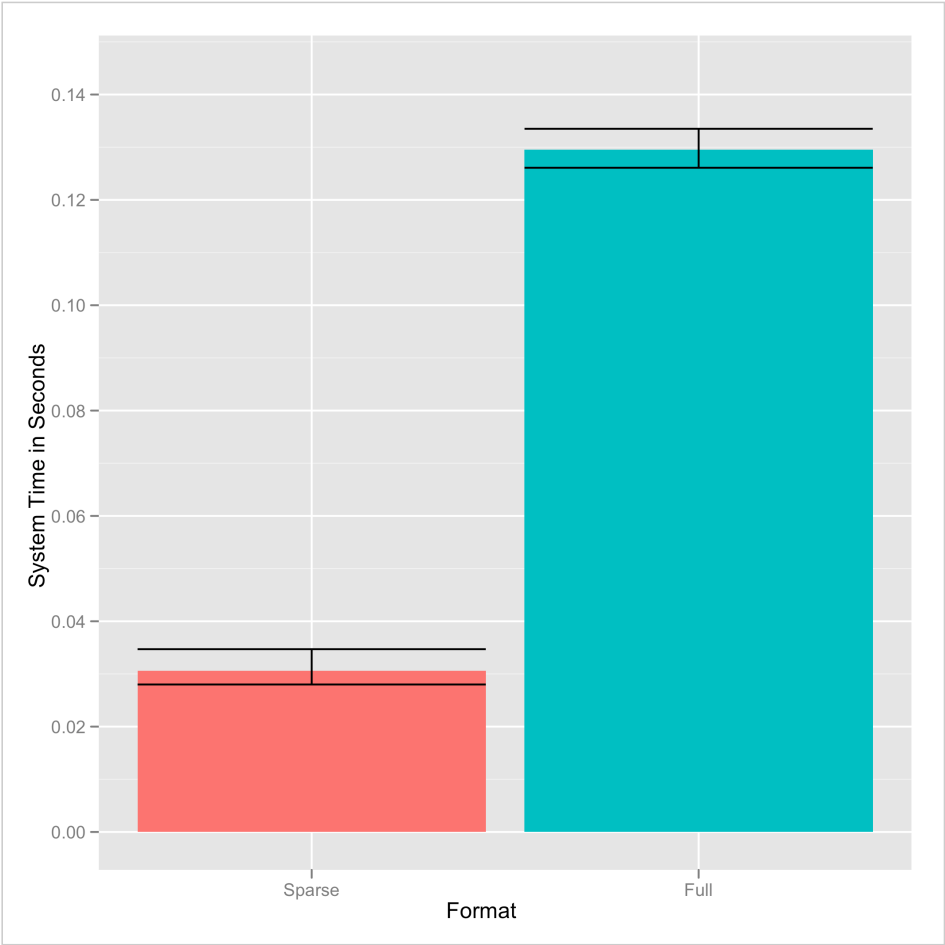
How much more efficient is the use of sparse matrices? Here I've recreated the example given in the glmnet package docs and added some additional data collection to give a sense of the time and speed gains that come with using sparse matrices for sparse data:

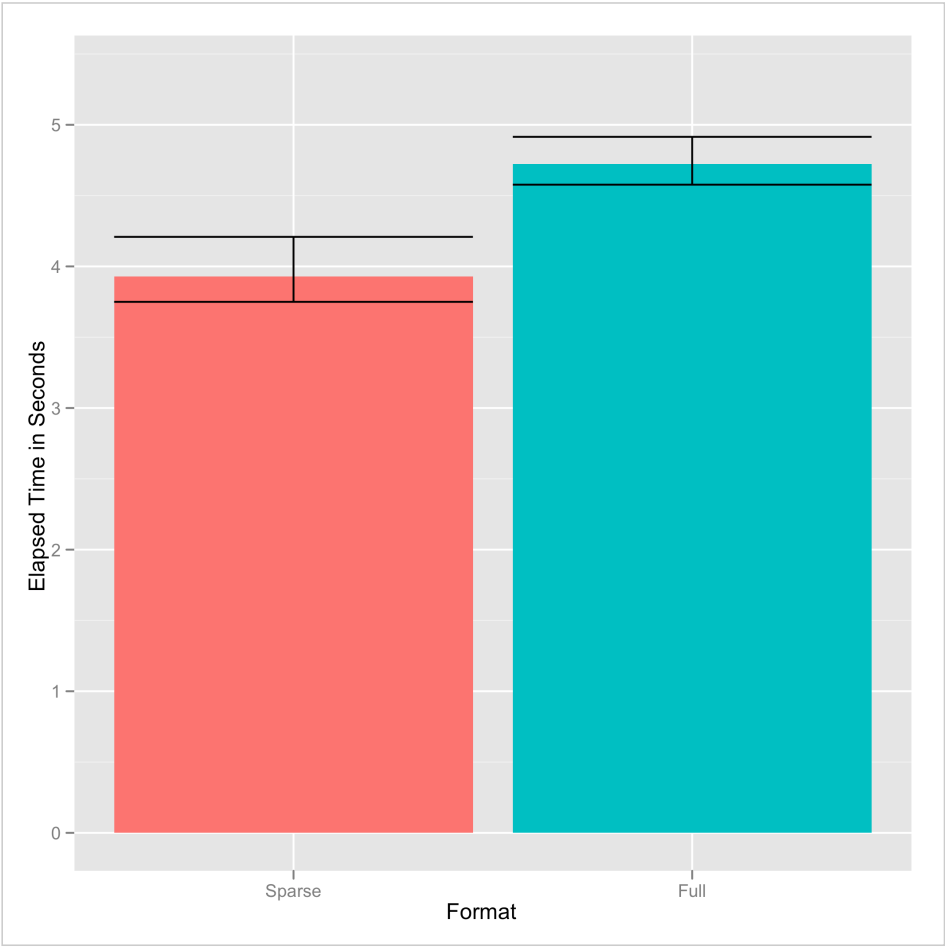
```
library('Matrix')
library('glmnet')

set.seed(1)
performance <- data.frame()

for (sim in 1:10)
{
  n <- 10000
  p <- 500

  nzc <- trunc(p / 10)
```

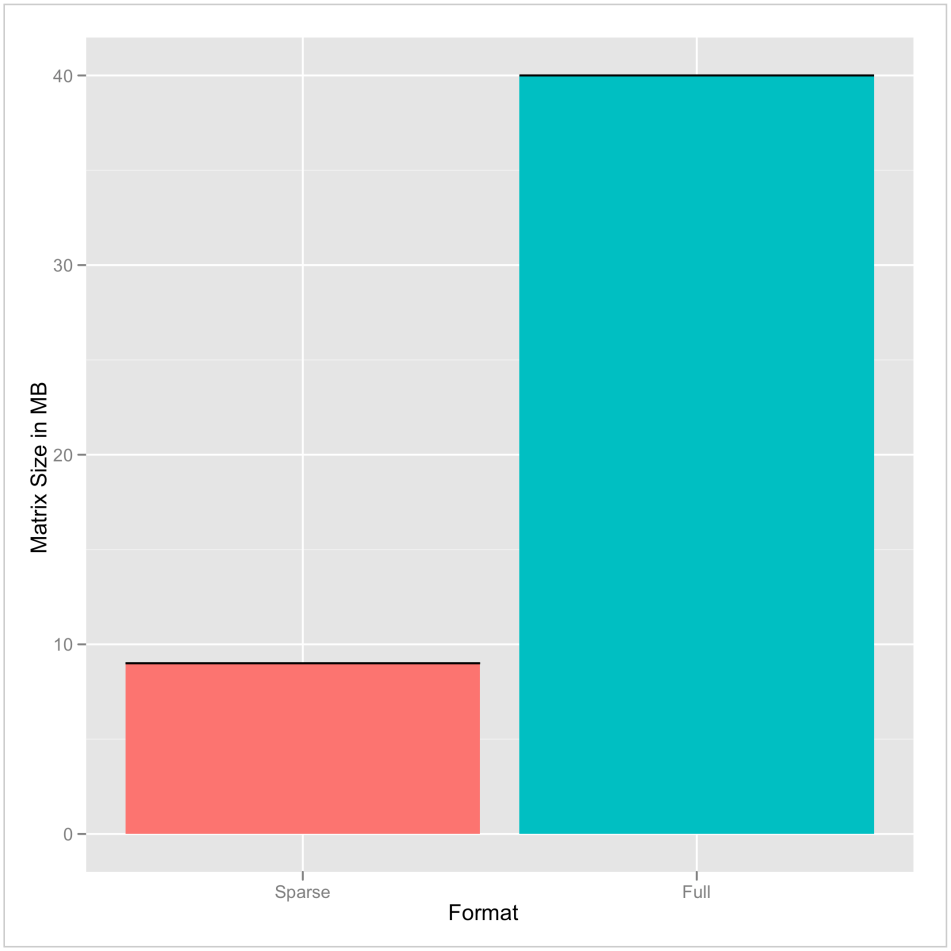
```
  x <- matrix(rnorm(n * p), n, p)
  iz <- sample(1:(n * p),
               size = n * p * 0.85,
               replace = FALSE)
  x[iz] <- 0
  sx <- Matrix(x, sparse = TRUE)

  beta <- rnorm(nzc)
  fx <- x[, seq(nzc)] %*% beta

  eps <- rnorm(n)
  y <- fx + eps

  sparse.times <- system.time(fit1 <- glmnet(sx, y))
  full.times <- system.time(fit2 <- glmnet(x, y))
  sparse.size <- as.numeric(object.size(sx))
  full.size <- as.numeric(object.size(x))

  performance <- rbind(performance, data.frame(Format = 'Sparse',
                                               UserTime = sparse.times[1],
                                               SystemTime = sparse.times[2],
                                               ElapsedTime = sparse.times[3],
                                               Size = sparse.size))
  performance <- rbind(performance, data.frame(Format = 'Full',
                                               UserTime = full.times[1],
                                               SystemTime = full.times[2],
                                               ElapsedTime = full.times[3],
                                               Size = full.size))
}

ggplot(performance, aes(x = Format, y = UserTime, fill = Format)) +
  stat_summary(fun.data = 'mean_cl_boot', geom = 'bar') +
  stat_summary(fun.data = 'mean_cl_boot', geom = 'errorbar') +
  ylab('User Time in Seconds') +
  opts(legend.position = 'none')
ggsave('sparse_vs_full_user_time.pdf')

ggplot(performance, aes(x = Format, y = SystemTime, fill = Format)) +
  stat_summary(fun.data = 'mean_cl_boot', geom = 'bar') +
  stat_summary(fun.data = 'mean_cl_boot', geom = 'errorbar') +
  ylab('System Time in Seconds') +
  opts(legend.position = 'none')
ggsave('sparse_vs_full_system_time.pdf')

ggplot(performance, aes(x = Format, y = ElapsedTime, fill = Format)) +
  stat_summary(fun.data = 'mean_cl_boot', geom = 'bar') +
  stat_summary(fun.data = 'mean_cl_boot', geom = 'errorbar') +
  ylab('Elapsed Time in Seconds') +
  opts(legend.position = 'none')
ggsave('sparse_vs_full_elapsed_time.pdf')

ggplot(performance, aes(x = Format, y = Size / 1000000, fill = Format)) +
  stat_summary(fun.data = 'mean_cl_boot', geom = 'bar') +
  stat_summary(fun.data = 'mean_cl_boot', geom = 'errorbar') +
  ylab('Matrix Size in MB') +
  opts(legend.position = 'none')
ggsave('sparse_vs_full_memory.pdf')
```

The images below show the resulting performance metrics:




0.14
0.12
0.10
0.08
0.06
0.04
0.02
0.00
System Time in Seconds
Sparse
Full
Format


5
4
3
2
1
0
Elapsed Time in Seconds
Sparse
Full
Format



Questions

I have some remaining questions about the use of sparse matrices that others might be able to answer:

- How can I perform assignment in place on a matrix generated by the slam package?
- How can I convert between sparse matrices generated by the Matrix and slam packages?

Posted in Programming, Statistics | 2 Responses

2 responses to "Using Sparse Matrices in R"

Brian Dalesandro 10.5.2012 at 1:24 pm | *Permalink*

Thanks for the examples and analysis. Do you have any experience reading sparse matrices into R? The Matrix package has a few read functions that seem to depend on some very specific sparse matrix representations that I've never heard of (readHB & readMM). Do you know of a more general format and function that can be used? The rest seems straightforward.

John Myles White 10.6.2012 at 3:35 pm | *Permalink*

I'm not aware of any better reading tools in R, although I imagine they exist. In the past I gave up on sparse matrix support in R and switched to Matlab for one project.

« Previous

Next »