Traiter des big data en R avec le package bigmemory

Pierre Lafaye de Micheaux

4 November 2020

Contents

1 Générer des fichiers memory-mapped

Le code suivant permet de générer des fichiers memory-mapped. Comme les données sont volumineuses et ne peuvent être chargées dans R, on utilise **library(bigmemory)** qui crée des fichiers memory-mapped.

```
library(bigmemory)
```

Noter que nous devons créer ces fichiers memory-mapped juste une fois. Dépendamment de la taille des données, cette étape peut être longue. Nous avons fait cela sur un gros fichier de données (32 GB) en utilisant la commande suivante.

Rappel du cours 1: Les instructions pour obtenir ces fichiers sont dans

https://github.com/YaohuiZeng/biglasso_reproduce/blob/master/reproduce_code_revision/Section_6_Big_data_case_README.R

```
# system.time(X <- read.big.matrix("Data/X_3000_1340000_200_logistic.txt",
#    sep=",",backingfile ="dataX.bin",
#    descriptor = "dataX.desc", type="double",shared=TRUE))
#user   system  elapsed
#1699.225  415.044 4460.952
```

Cela crée deux fichiers:

```
- dataX.bin
- dataX.desc
```

La fonction *read.big.matrix()* crée le fichier-backing binaire associé avec *l'objet big.matrix* X et le fichier descripteur partagé (shared descriptor).

2 Charger les données depuis le disque en utilisant le fichier-backing binaire

En utilisant le descripteur partagé, on peut charger les données depuis le disque avec la fonction *attach.big.matrix()*.

Des sessions subséquentes peuvent se connecter au backing de façon instantanée, et on peut interagir avec celui-ci (e.g., calculer certaines statistiques).

Télécharger les fichiers "dataX.desc" et "dataX.bin" ici:

https://www.dropbox.com/sh/de1x682sdlbub0r/AADhFeqLFQTWobf9tkDBcT8Ka/Lecture1?dl=0&subfolder_nav_tracking=1

```
# I had to modify the dirname value in the file 'dataX.desc'
dataXdesc <- dget("Data/dataX.desc")
system.time(dataX <- attach.big.matrix(dataXdesc))
##    user  system elapsed
##   0.001   0.000   0.001
```

```
gc()
##           used (Mb) gc trigger (Mb) max used (Mb)
```

```
## Ncells  553079 29.6    1237667 66.1   675736 36.1
## Vcells 1016466  7.8    8388608 64.0  1753842 13.4
```

2.1 Pointeur vers une matrice C++

La big.matrix contient en fait un pointeur vers une matrice C++ qui est sur le disque

```
dim(dataX)
## [1]    3000 1340000
dataX
## An object of class "big.matrix"
## Slot "address":
## <pointer: 0x5567db2e05e0>
```

```
is.filebacked(dataX)
## [1] TRUE
is.big.matrix(dataX)
## [1] TRUE
is.shared(dataX)
## [1] TRUE
```

2.2 Quelques Statistiques sur Big Matrix

```
summary(dataX[, 1:2])
##        V1                  V2
##  Min.   :-3.53959   Min.   :-3.671300
##  1st Qu.:-0.66469   1st Qu.:-0.672130
##  Median :-0.02253   Median :-0.019991
##  Mean   :-0.00420   Mean   :-0.005014
##  3rd Qu.: 0.71158   3rd Qu.: 0.659005
##  Max.   : 3.81028   Max.   : 3.064524
```

```
range(dataX[,1], na.rm = TRUE)
## [1] -3.539586  3.810277
```

La fonction suivante ne marchera pas sur cette bigmatrix

```
tail(dataX, 1)
max(dataX[,])
```

2.3 Fonction mwhich

```
- Fournit une sélection efficace de lignes pour des objets matrix et big.matrix and matrix
- Basée sur la fonction R `which()` sans débordement de mémoire
```

mwhich(x, cols, vals, comps, op = 'AND')

```
- x: cols: vals:
- un objet big.matrix.
```

```
- un vecteur d'indices de colonne ou de noms.
- une liste de vecteurs de longueurs 1 ou 2; longueur 1: utilisé pour tester l'égalité (ou l'inegalité),
- comps: une liste d'opérateurs incluant 'eq', 'neq', 'le', 'lt', 'ge' et 'gt'. op: soit 'AND' ou 'OR'.
```

Quelle est la signification du code suivant?

```
indices <- mwhich(dataX, 1, -0.66469, "le")
length(indices)
## [1] 750
length(indices)/dim(dataX)[1]
## [1] 0.25
```

2.4 Fonction deepcopy

Cela est nécessaire pour créer une copie d'une big.matrix, avec la nouvelle copie filebacked de façon optionnelle.

```
dataX
## An object of class "big.matrix"
## Slot "address":
## <pointer: 0x5567db2e05e0>
```

```
dataX[1,1]
## [1] -0.6264538
```

```
dataXcopy <- dataX
dataXcopy
## An object of class "big.matrix"
## Slot "address":
## <pointer: 0x5567db2e05e0>
```

dataX et *dataXcopy* pointent vers les mêmes données en memory.

```
dataX[1,1]
## [1] -0.6264538
dataX[1,1]<- as.integer(1)
dataX[1,1]
## [1] 1
```

```
dataXcopy[1,1]
## [1] 1
```

Nous souslignons l'utilisation de `deepcopy()`:

```
w <- deepcopy(dataXcopy,1:10,backingfile = "w.bin",descriptorfile = "w.desc")
dim(w)
## [1] 3000   10
w
## An object of class "big.matrix"
## Slot "address":
## <pointer: 0x5567df6ec690>
```

Nous exportons aussi cette "bigmatrix" dans un fichier texte pour utilisation ultérieure.

```
write.big.matrix(w,"w.txt")
```

2.5 La Fonction `flush()`

flush() force toute information modifiée à être écrite dans le fichier-backing.

```
dataX[1:3,1:3]
##            [,1]      [,2]       [,3]
## [1,]  1.0000000 0.7391149 -0.6188271
## [2,]  0.1836433 0.3866087 -1.1094220
## [3,] -0.8356286 1.2963972 -2.1703352
```

```
dataX[,1] <- 1
dataX[1:3,1:3]
##      [,1]      [,2]       [,3]
## [1,]    1 0.7391149 -0.6188271
## [2,]    1 0.3866087 -1.1094220
## [3,]    1 1.2963972 -2.1703352
```

Le fichier `dataX.bin` n'a pas encore été changé.

```
newdataX <- attach.big.matrix(dget("Data/dataX.desc"))
newdataX[1:3,1:3]
##      [,1]      [,2]       [,3]
## [1,]    1 0.7391149 -0.6188271
## [2,]    1 0.3866087 -1.1094220
## [3,]    1 1.2963972 -2.1703352
```

Vous devez utiliser *flush()*

```
flush(dataX)
newdataX <- attach.big.matrix(dget("Data/dataX.desc"))
newdataX[1:3,1:3]
##      [,1]      [,2]       [,3]
## [1,]    1 0.7391149 -0.6188271
## [2,]    1 0.3866087 -1.1094220
## [3,]    1 1.2963972 -2.1703352
```

```
- Cela change seulement le fichier backed.
- Les données d'origine n'est pas changé
- Le fichier backed ne correspond plus aux données d'origine !!!
```

Nous devrions trouver une façon de regarder à la première colonne du fichier "X_300..."

3 Mémoire Partagée

3.1 Mémoire Partagée Interactive

- Exemple 1

- 1. Deux sessions R sont connectées à la même shared.big.matrix
- 2. l'affectation dans un processus affectera la valeur dans l'autre session.

```
dataX[1:3,4]
## [1] -1.2171201 -0.9462293  0.0914098
ind <-mwhich(dataX,4,0,"le")
mean(dataX[ind,4])
## [1] -0.7828649
sd(dataX[ind,4])
## [1] 0.5866901
```

Dans une deuxième second session, taper les commandes suivantes

```
library(bigmemory)
path <-"MODIFIER-ICI"
path <-"."
setwd(path)
xdes <- dget("dataX.desc")
dataX <- attach.big.matrix(xdes)
dataX[1:3,4]
ind <-mwhich(dataX,4,0,"le")
dataX[ind,4] <- 10
mean(dataX[ind,4])
sd(dataX[ind,4])
```

Et maintenant dans la session courante

```
mean(dataX[ind,4])
## [1] -0.7828649
sd(dataX[ind,4])
## [1] 0.5866901
```

4 Famille bigmemory

La famille de bigmemory fournit une collection de fonctions pour les objets `big.matrix`:

- *biganalytics* pour des fonctions de base en analytics et en statistique,
- *bigtabulate* pour des opérations de tabulation (Emerson and Kane, 2013b),
- *bigalgebra* pour des opérations matricielles avec les librairies BLAS et LAPACK (Kane et al., 2014).

Quelques fonctions additionnelles pour des objets big.matrix sont disponibles dans d'autres packages, tels que

- *bigpca* pour l'analyse en composantes principales et la décomposition en valeurs singulières (Cooper, 2014)
- *bigrf* pour les forêts aléatoires (Lim et al., 2014).
- *biglars* pour la régression least angle et le LASSO (Seligman et al., 2011)

5 Package biglm : régression linéaire à mémoire bornée

biglm crée un objet modèle linéaire qui utilise seulement p^2 emplacements mémoire pour p variables. Il peut être mis à jour avec plus de données en utilisant update. Cela permet de faire des régressions linéaires sur des objets plus gros que la RAM.

5.1 Command biglm

- *biglm* effectue une modélisation `lm`

```
library(biglm)
```

```
big.model = biglm(Temp ~ Wind + Solar.R + Ozone, data = airquality)
summary(big.model)
## Large data regression model: biglm(Temp ~ Wind + Solar.R + Ozone, data = airquality)
## Sample size =  111
##                Coef    (95%     CI)     SE      p
## (Intercept) 72.4186 65.9875 78.8496 3.2155 0.0000
## Wind        -0.3229 -0.7895  0.1436 0.2333 0.1662
## Solar.R      0.0073 -0.0081  0.0226 0.0077 0.3433
## Ozone        0.1720  0.1192  0.2247 0.0264 0.0000
```

- Modélisation *lm*

```
model = lm(Temp ~  Wind + Solar.R + Ozone, data = airquality)
summary(model)[4]
## $coefficients
##                 Estimate  Std. Error    t value     Pr(>|t|)
## (Intercept) 72.418579038 3.215524714 22.5215433 2.107000e-42
## Wind        -0.322944554 0.233264236 -1.3844581 1.690987e-01
## Solar.R      0.007275637 0.007677657  0.9476377 3.454492e-01
## Ozone        0.171966042 0.026389869  6.5163660 2.423506e-09
```

5.2 Commande `bigglm`

Rappel du cours 1: Les instructions pour obtenir ces fichiers sont dans

https://github.com/YaohuiZeng/biglasso_reproduce/blob/master/reproduce_code_revision/Section_6_Big_data_case_README.R

D'abord, on lit le vecteur réponse

```
y <- read.big.matrix("Data/y_3000_1340000_200_logistic.txt", type = "integer",
                     backingfile = "y.bin", descriptorfile = "y.desc")
```

Comme $p > n$, on ne peut pas utiliser un simple modèle linéaire. Dans la prochaine leçon, nous verrons comment contourner ce problème. Par exemple, nous sélectionnons juste 300 prédicteurs aléatoirement à partir de X.

```
set.seed(1)
indp <- sample(1:(dim(dataX)[2]),300)
X.cut <- dataX[,indp]
data.cut <- data.frame(cbind(Y=as.vector(y[,1]),X.cut))
```

Un modèle logistique avec *bigglm*

```
model2 <- bigglm(Y~V2+V3,data=data.cut,family=binomial("logit"))
summary(model2)
## Large data regression model: bigglm(Y ~ V2 + V3, data = data.cut, family = binomial("logit"))
## Sample size =  3000
##                Coef    (95%    CI)     SE      p
## (Intercept)  0.0092 -0.0639 0.0822 0.0365 0.8021
## V2           0.7683 -1.4832 3.0197 1.1257 0.4949
## V3          -0.8874 -2.5196 0.7447 0.8161 0.2768
```

Mais ce n'est pas un exemple avec des données tellement grosses.

Faisons une petite pratique sur des *tall Data* ....

5.3 Exemple Netflix

On va jouer avec les données du Netflix Prize (Netflix, Inc. 2006). Le training set inclue 99,072,112 scores (ratings) et cinq variables à valeurs entières: movie ID, customer ID, rating, rental year et month.

Le fichier `Netflix.txt` (2.15 Go) est disponible ici:

https://www.dropbox.com/sh/de1x682sdlbub0r/AADhFeqLFQTWobf9tkDBcT8Ka/Lecture1?dl=0&preview=Netflix.txt&subfolder_nav_tracking=1

- lire les données

```
# system.time(x <- read.big.matrix("Data/Netflix.txt", sep = ",", type = "integer",
#      backingfile = "Netflix.bin", descriptor = "Netflix.desc",
#     shared = TRUE, col.names = c("movie", "customer", "rating","year", "month")))
#   user  system elapsed
# 89.464   3.471  96.530
# dim(x)
```

```
dim(x)
## [1] 100480507         5
```

- Combien de films ? Combien de clients ?

```
range(x[,1])
## [1]     1 17770
range(x[,2])
## [1]       6 2649429
summary(x[,3])
##    Min. 1st Qu.  Median    Mean 3rd Qu.    Max.
##   1.000   3.000   4.000   3.604   4.000   5.000
range(x[,4])
## [1] 1999 2005
```

```
range(x[,5])
## [1]  1 12
```

Un meilleur résumé est fourni grâce au package *biganalytics*

```
library(biganalytics)
summary(x)
##                  min          max         mean          NAs
## movie    1.000000e+00 1.777000e+04 9.070915e+03 0.000000e+00
## customer 6.000000e+00 2.649429e+06 1.322489e+06 0.000000e+00
## rating   1.000000e+00 5.000000e+00 3.604290e+00 0.000000e+00
## year     1.999000e+03 2.005000e+03 2.004254e+03 0.000000e+00
## month    1.000000e+00 1.200000e+01 6.727243e+00 0.000000e+00
```

- Nous allons regarder les ratings du client 2442

```
cust.indices <- mwhich(x, "customer", 2442, "eq")
head(x[cust.indices, ])
##      movie customer rating year month
## [1,]     1     2442      3 2004     4
## [2,]    30     2442      3 2005     8
## [3,]   188     2442      3 2005     5
## [4,]   191     2442      4 2004     3
## [5,]   283     2442      4 2004     1
## [6,]   457     2442      3 2005     4
```

- Des comparaisons plus complexes: par exemple, nous pourrions être intéressés au films du clien 2442's qui ont été notés 2 ou pire entre février et octobre 2004:

```
these <- mwhich(x, c("customer", "year", "month", "rating"),
    list(2442, 2004, c(2, 10), 2), list("eq", "eq", c("ge", "le"),
        "le"), "AND")
x[these, ]
##      movie customer rating year month
## [1,]  2016     2442      2 2004     6
## [2,]  2211     2442      2 2004     9
## [3,]  2574     2442      2 2004     2
## [4,]  2885     2442      2 2004     4
## [5,]  7291     2442      1 2004     3
## [6,]  7648     2442      1 2004     8
## [7,]  9945     2442      2 2004     6
## [8,] 13069     2442      2 2004     6
```

- Nous fournissons les titres des films pour placer ces notes en contexte:

https://www.dropbox.com/sh/de1x682sdlbub0r/AACwvzZqmMcorQh6PyvoEhZOa/Lecture1/movie_titles.csv?dl=0

```
mnames <- read.csv("Data/movie_titles.csv", header = FALSE)
names(mnames) <- c("movie", "year", "Name of Movie")
mnames[mnames[, 1] %in% unique(x[these, 1]), c(1, 3)]
##       movie                          Name of Movie
## 2050   2016                 The Magdalene Sisters
```

```
## 2248   2211                              Spliced
## 2616   2574                              Abandon
## 2937   2885                             The Girl
## 7417   7291                          Sugar Sweet
## 7777   7648                             Gasoline
## 10123  9945 Melissa Etheridge: Live ... and Alone
## 13307 13069           Learning Guitar for Dummies
```

- Un exemple en utilisant `biglm`: l'année de sortie du film est utilisé (comme un `factor`) pour essayer de prédire les notes d'un client:

```
#library(biganalytics)
lm.0 <- biglm.big.matrix(rating ~ year, data = x, fc = "year")
print(summary(lm.0)$mat)
##                  Coef         (95%        CI)         SE            p
## (Intercept) 3.33700643  3.290699783 3.38331307 0.02315332 0.000000e+00
## year2000    0.02820917 -0.018151996 0.07457033 0.02318058 2.236305e-01
## year2001    0.05372952  0.007394375 0.10006466 0.02316757 2.038586e-02
## year2002    0.04480920 -0.001509059 0.09112745 0.02315913 5.300999e-02
## year2003    0.06927861  0.022966913 0.11559030 0.02315585 2.773009e-03
## year2004    0.25796194  0.211653625 0.30427025 0.02315416 7.916811e-29
## year2005    0.33910078  0.292793185 0.38540837 0.02315380 1.437686e-48
```

Il apparait que les scores donnés pour les films de 2004 et 2005 movies ont été plus élevés (en moyenne) que ceux pour les annés précédentes. Cette régression ne gagnera pas le prix Netflix de $1,000,000. Cependant, elle illustre l'utilisation d'une `big.matrix` pour gérer et étudier plusieurs gigabytes de données.

5.4 Qu'y a-til derrière `biglm` ?

Considérons le modèle linéaire dans le cas standard où $n > p$:

$$y = X\beta + \epsilon$$

L'estimateur des moindres carrés est alors:

$$\hat{\beta} = (X^T X)^{-1} X^T y$$

Quand n est très grand (tall data), l'implémentation de base de R *lm.fit* prend $O(np + p^2)$ en mémoire. Une première option pour surmonter des problèmes de mémoire est de calculer $X^T X$ et $X^T y$ par incréments, et puis d'obtenir l'estimateur des moindres carrés de β, $\hat{\beta} = (X^T X)^{-1} X^T y$. Cette méthode est celle mis en oeuvre dans le package *speedglm* (Enea, 2014).

5.4.1 Implémentation de Biglm

Une autre option est de calculer la décomposition QR incrémentale (Miller, 1992) de $X = QR$ pour obtenir R et $Q^T y$, et ensuite de résoudre β à partir de

$$R\beta = Q^T y.$$

Cette option est implémentée dans le package *biglm*. La fonction *biglm* utilise seulement p^2 de mémoire pour p variables et l'objet ajusté (fitted) peut être mis à jour avec plus de données en utilisant l'option `update`.

Le package fournit aussi un calcul incrémental de l'estimateur de la variance sandwich en accumulant une matrice $(p+1)^2 \times (p+1)^2$ de produits de X et y sans un second passage des données.

6 Combiner bigmemory et foreach

La structure *big.matrix* a plusieurs avantages tels que le support de la mémoire partagée pour l'efficacité des calculs en parallèle, un comportement par référence qui évite des copies temporaires non nécessaires d'objets massifs, et un format en colonne-major qui est compatible avec les packages classiques d'algèbre linéaire (e.g., BLAS, LAPACK).

6.1 Paralléliser est très facile

Faire des calculs parallèles quand vous faites des simulations

```
library(parallel)
cores <- detectCores() ## Combien de coeurs avons nous?
print(cores)
## [1] 4
```

Mon étude de simulations estime l'erreur test d'une regression ridge

```
one.simu <- function(i) {
## tracer les données
n <- 1000; p <- 500
x <- matrix(rnorm(n*p),n,p) ; y <- rnorm(n)
## renvoie les coefficients ridge
  train <- 1:floor(n/2)
  test  <- setdiff(1:n,train)
  ridge <- lm.ridge(y~x+0,lambda=1,subset=train)
  err <- (y[test] - x[test, ] %*% ridge$coef )^2
  return(list(err = mean(err), sd = sd(err)))
}
```

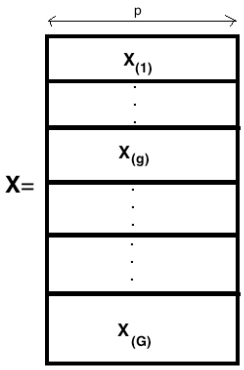
```
library(MASS)
out <- mclapply(1:8, one.simu, mc.cores=cores)
head(do.call(rbind, out))
##      err      sd
## [1,] 9.612864 13.04632
## [2,] 11.97253 17.42302
## [3,] 8.554212 11.49829
## [4,] 10.8209  15.0637
## [5,] 12.83857 16.96813
## [6,] 12.49638 17.4019
```

6.2 Calcul Parallèle avec `foreach`

Pour illustration, nous allons créer la fonction de cross product *cpc()* qui permet le calcul de X^TY par chunks pour n'importe quelles matrices X et Y.

$$(X^TY) = \sum_{g=1}^{G} X_{(g)}^T Y_{(g)}$$

Toues les termes rentrent (successivement) dans la mémoire!



D'abord, défnissons la fonction *readchunk()*

```
readchunk <- function(X, g, size.chunk) {
  rows <- ((g - 1) * size.chunk + 1):(g * size.chunk)
  chunk <- X[rows,]
}
```

```
cpc <- function(X, Y, ng = 1) {
  readchunk <- function(X, g, size.chunk) {
  rows <- ((g - 1) * size.chunk + 1):(g * size.chunk)
  chunk <- X[rows,]
}
  res <- foreach(g = 1:ng, .combine = "+") %dopar% {
    size.chunk <- nrow(X) / ng
    chunk.X <- readchunk(X, g, size.chunk)
    chunk.Y <- readchunk(Y, g, size.chunk)
    term <- t(chunk.X) %*% chunk.Y
  }
  return(res)
}
```

Nous illustrons cela sur un example simple

```
set.seed(1)
n <- 10000
p <- 4
q <- 3

X <- matrix(rnorm(n*p),ncol=4,nrow=n)
```

```
Y <- matrix(rnorm(n*q),ncol=3,nrow=n)

res <- crossprod(X,Y)
res
##            [,1]       [,2]     [,3]
## [1,] -133.31665 -183.74697 29.16314
## [2,] -123.02318 -138.56943 94.13016
## [3,]   44.43988  -60.55680 92.16315
## [4,]  102.23424   26.19045 40.09826
```

```
library(doSNOW)
cl <- makeCluster(4)
registerDoSNOW(cl)
res.cpc <- cpc(X,Y,ng=10)
stopCluster(cl)
res.cpc
##            [,1]       [,2]     [,3]
## [1,] -133.31665 -183.74697 29.16314
## [2,] -123.02318 -138.56943 94.13016
## [3,]   44.43988  -60.55680 92.16315
## [4,]  102.23424   26.19045 40.09826
```

6.3 Calcul Parallèle avec un objet `bigmatrix`

Que faire si X et Y sont des objets `big.matrix` ?

Jouons avec un objet `big.matrix` et calculons $X^T X$.

D'abord, on lit les données MNIST:

https://www.dropbox.com/sh/de1x682sdlbub0r/AADk1bYmP51YSbAll58T0XCXa/Lecture1/WORKSHOP_DATA_SET_LIGHT?dl=0&preview=MNIST10000.csv&subfolder_nav_tracking=1

```
dataX <- read.big.matrix("Data/MNIST10000.csv",descriptorfile = "MNIST.des",backingfile = "MNIST.bin",
                         header = TRUE, sep = ";", type = "integer")
Xdes <- describe(dataX)
# Xdes
```

Ceci ne marchera pas

```
# crossprod(dataX,dataX)
# cpc(dataX,dataX,10)
```

```
XtX.big <- function(X.des, ng = 1) {

  readchunk <- function(X, g, size.chunk) {
  rows <- ((g - 1) * size.chunk + 1):(g * size.chunk)
  chunk <- X[rows,]
}
  res <- foreach(g = 1:ng, .combine = "+") %dopar% {
    require("bigmemory")
```

```
    X <- attach.big.matrix(X.des)
    size.chunk <- nrow(X) / ng
    chunk.X <- readchunk(X, g, size.chunk)
#    chunk.Y <- readchunk(Y, g, size.chunk)
    term <- t(chunk.X) %*% chunk.X
  }
  return(res)
}
```

```
library(doSNOW)
cl <- makeCluster(4)
registerDoSNOW(cl)
cl <- makeCluster(4)
res.big <- XtX.big(Xdes,ng=10)
stopCluster(cl)
summary(res.big[,101])
##     Min.  1st Qu.   Median     Mean  3rd Qu.     Max.
##        0    38264  5369590  6166650 11278473 25399607
```