

Quelques Jeux de Données Massifs

Pierre Lafaye de Micheaux

25 September 2020

Contents

1	Le Jeu de Données EMNIST	3
1.1	Télécharger les Données EMNIST	3
1.2	Lire et Afficher les Données EMNIST	3
1.3	Matrices des Données EMNIST	9
1.4	Les Données EMNIST au Format <code>big.matrix</code>	10
2	Le Jeu de Données Airlines	11
2.1	Accéder aux Données Airlines	11
2.2	Le Jeu de Données Airlines: Première Inspection	11
2.3	Le Jeu de Données Airlines: Premières Manipulations	12
2.4	Le Jeu de Données Airlines: Un Seul Fichier	12
2.5	Le Jeu de Données Airlines: Avec Linux I	13
2.6	Le Jeu de Données Airlines: Avec Linux II	13
2.7	Le Jeu de Données Airlines: Avec Linux III	13
2.8	Le Jeu de Données Airlines	14
2.9	Le Jeu de Données Airlines	14
2.10	Le Jeu de Données Airlines	14
2.11	Le Jeu de Données Airlines	15
2.12	Le Jeu de Données Airlines	15
2.13	Le Jeu de Données Airlines	15
2.14	Le Jeu de Données Airlines	16
2.15	Le Jeu de Données Airlines	16
2.16	Le Jeu de Données Airlines	16
2.17	Le Jeu de Données Airlines	17
2.18	Le Jeu de Données Airlines	17
2.19	Le Jeu de Données Airlines	17
2.20	Le Jeu de Données Airlines	17
2.21	Le Jeu de Données Airlines	18
2.22	Le Jeu de Données Airlines	18

Quelques Jeux de Données Massifs

2.23	Le Jeu de Données Airlines	18
2.24	Le Jeu de Données Airlines	18
2.25	Le Jeu de Données Airlines	19
2.26	Le Jeu de Données Airlines	19
2.27	Le Jeu de Données Airlines	20
2.28	Le Jeu de Données Airlines	20
2.29	Le Jeu de Données Airlines	20
3	Le Jeu de Données The GWAS	20
4	Le Prix Netflix 2006–2009 (source: Hastie)	21
4.1	Le Prix Netflix	21
4.2	Le Prix Netflix	22
4.3	Le Prix Netflix	24
5	Le Jeu de Données Bosch Production Line Performance	25
6	Quelques Autres Jeux de Données	25

1 Le Jeu de Données EMNIST

MNIST (LeCun & Cortes, 2010) est un jeu de données qui est devenu une référence pour les systèmes de classification. Sa version étendue (EMNIST) (<https://www.nist.gov/itl/iad/image-group/emnist-dataset>), est un autre jeu de données (536MB) librement accessible (à http://www.itl.nist.gov/iaui/vip/cs_links/EMNIST/gzip.zip). Il est décrit par (Cohen et al., 2017) come un jeu de données qui “constitutes a more challenging classification tasks involving letters and digits, and that shares the same image structure and parameters as the original MNIST task”. À partir de ce jeu de données, on considère le sous-ensemble de toutes les $n = 280\,000$ images de chiffres écrits à la main, qui contiennent un nombre égal d'échantillons pour chaque classe de chiffre (0 à 9). Ils sont déjà partitionnés en un “training set” de 240 000 cas et un “test set” de 40 000 cas. Ce sont des images de 28 x 28 pixels, en niveaux de gris, avec des valeurs dans l'intervalle [0, 255].

1.1 Télécharger les Données EMNIST

```
# Changer pour votre répertoire de travail:
WORKDIR <- "/home/backup/OrganisationDesFichiers/Universite/Enseignement/CoursDispenses/PaulValery/2020/V3EMNIST"
EMNIST.URL <- "http://www.itl.nist.gov/iaui/vip/cs_links/EMNIST/gzip.zip"
setwd(WORKDIR)
if(!dir.exists(paste(WORKDIR, "/Data", sep = ""))) dir.create(paste(WORKDIR, "/Data", sep = ""))
if(!file.exists(paste(WORKDIR, "/Data/emnist.zip", sep = "")))
download.file(EMNIST.URL, destfile = paste(WORKDIR, "/Data/emnist.zip", sep = ""))
unzip("Data/emnist.zip", exdir = "Data")
if (!"R.utils" %in% installed.packages()) install.packages("R.utils")
if (!file.exists("Data/emnist-digits-train-images-idx3-ubyte")) {
  for (file in c("emnist-digits-train-images-idx3-ubyte",
                "emnist-digits-train-labels-idx1-ubyte",
                "emnist-digits-test-images-idx3-ubyte",
                "emnist-digits-test-labels-idx1-ubyte")) {
    R.utils::gunzip(paste("Data/gzip/", file, ".gz", sep = ""),
                   destname = paste("Data/", file, sep = ""))
  }
}
unlink("Data/gzip", recursive = TRUE) # Pour supprimer le dossier 'gzip'.
```

1.2 Lire et Afficher les Données EMNIST

Lire la description des données ici: <https://www.nist.gov/itl/products-and-services/emnist-dataset>

et surtout en bas de page ici: <http://yann.lecun.com/exdb/mnist/>

Puis consulter l'aide en ligne de la fonction R `readBin()` qui permet de lire des données binaires (de 1 byte chaque) à l'aide de l'argument `what = "raw"`.

Exercice: Utiliser vos connaissances de la première leçon et utiliser la fonction `readBin()` pour lire le magic number du fichier “emnist-digits-train-images-idx3-ubyte”.

Exercice: Créer un code R permettant de lire le nombre d'images stockées dans le fichier “emnist-digits-train-images-idx3-ubyte”.

Quelques Jeux de Données Massifs

Exercice: Extraire la première image du fichier "emnist-digits-train-images-idx3-ubyte". Afficher la dans R au moyen de la fonction `image()`.

Exercice: Calculer la taille totale de toutes les images (en GB) du training set. Vérifier votre calcul au moyen de la fonction `object.size()`.

```
WORKDIR.Data <- paste(WORKDIR, "/Data", sep = "")
setwd(WORKDIR.Data)
file_training_set_image <- "emnist-digits-train-images-idx3-ubyte"
file_training_set_label <- "emnist-digits-train-labels-idx1-ubyte"
file_test_set_image <- "emnist-digits-test-images-idx3-ubyte"
file_test_set_label <- "emnist-digits-test-labels-idx1-ubyte"

readBindata <- function(file, from, n) {
  as.numeric(paste("0x", paste(readBin(file, "raw", n = n)[from:n],
                                collapse = ""), sep = ""))
}

extract_images <- function(file, nbimages = NULL) {
  if (is.null(nbimages)) { # On extrait toutes les images
    nbimages <- readBindata(file, 5, 8)
  }
  nbrows <- readBindata(file, 9, 12)
  nbcols <- readBindata(file, 13, 16)
  raw <- readBin(file, "raw", n = nbimages * nbrows * nbcols + 16)[- (1:16)]
  return(array(as.numeric(paste("0x", raw, sep="")), dim = c(nbcols, nbrows, nbimages)))
}

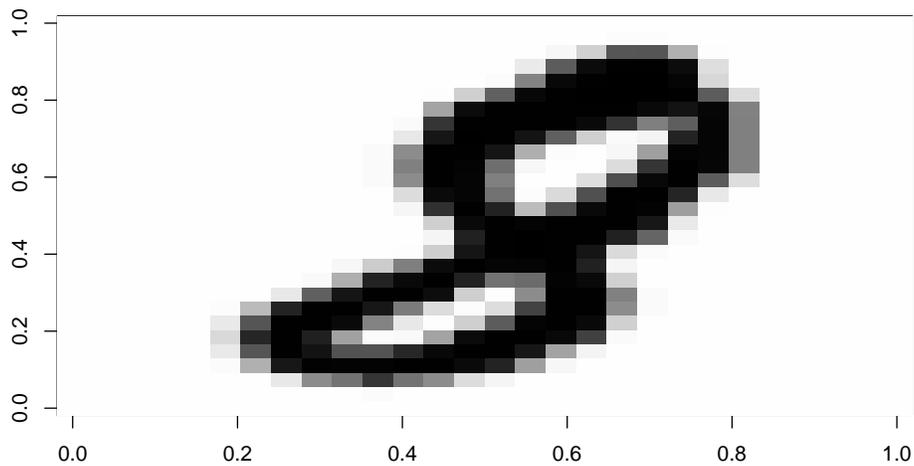
extract_labels <- function(file) {
  nbitem <- readBindata(file, 5, 8)
  raw <- readBin(file, "raw", n = nbitem + 8)[- (1:8)]
  return(as.numeric(paste("0x", raw, sep="")))
}

nbimages_to_extract <- 10
images_training_set <- extract_images(file_training_set_image, nbimages_to_extract)
images_test_set <- extract_images(file_test_set_image, nbimages_to_extract)
labels_training_set <- extract_labels(file_training_set_label)
labels_test_set <- extract_labels(file_test_set_label)

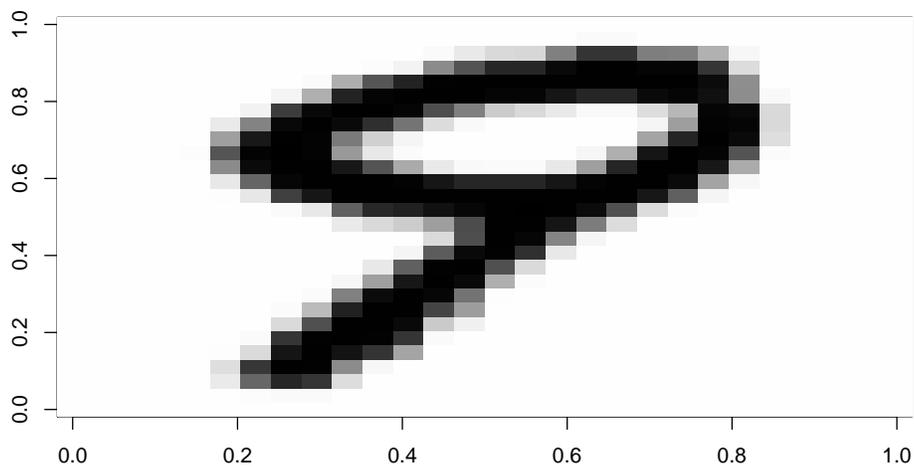
par(ask = FALSE)
for (i in 1:nbimages_to_extract) {
  img <- images_training_set[, , i]
  image(t(img)[, nrow(img):1], col = gray((255:0)/256))
  title(paste("Label:", labels_training_set[i]))
}
```

Quelques Jeux de Données Massifs

Label: 8

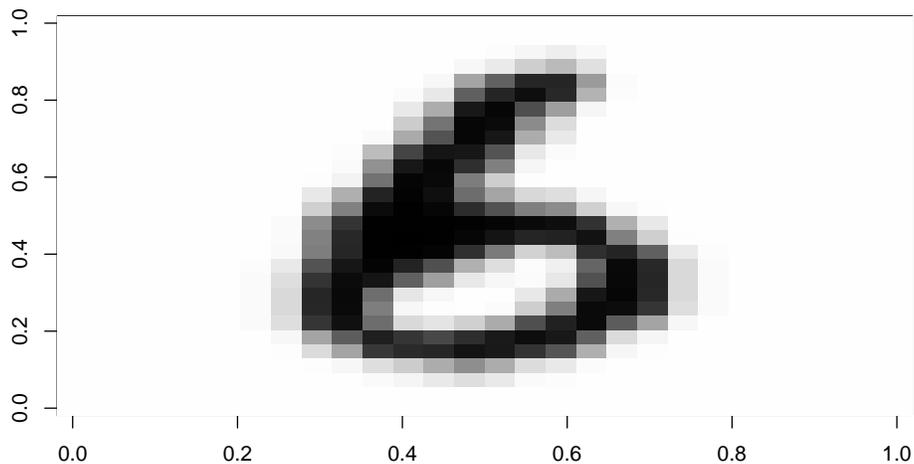


Label: 9

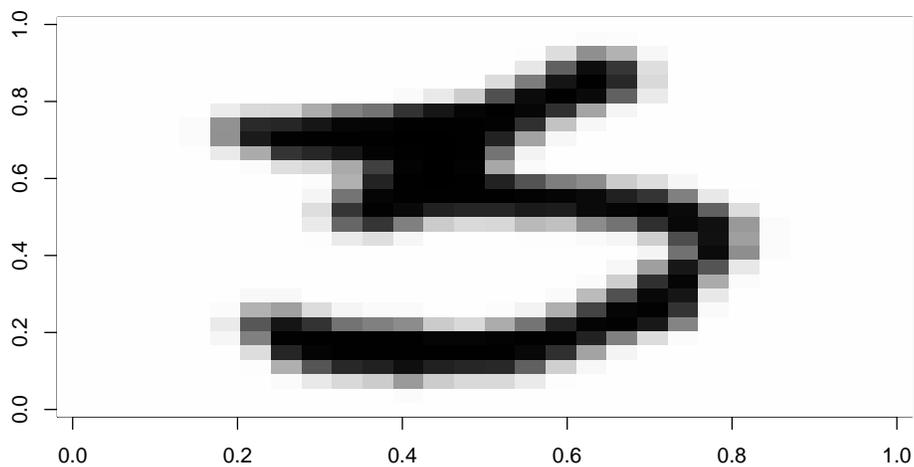


Quelques Jeux de Données Massifs

Label: 6

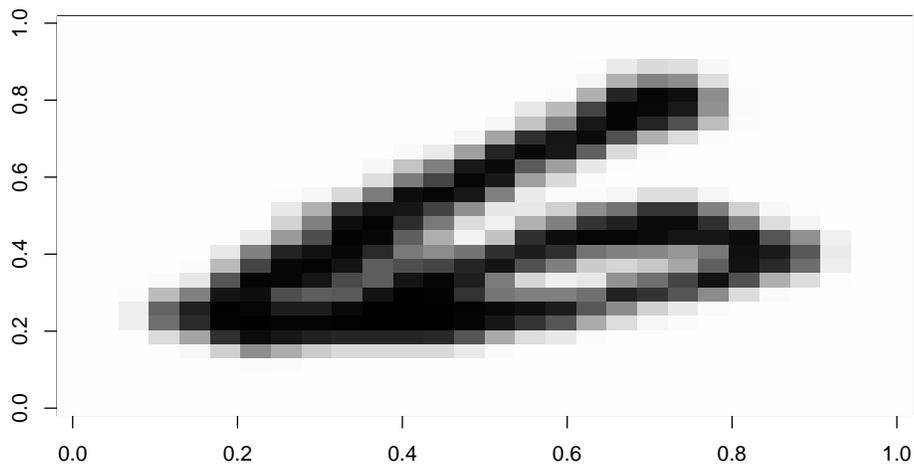


Label: 3

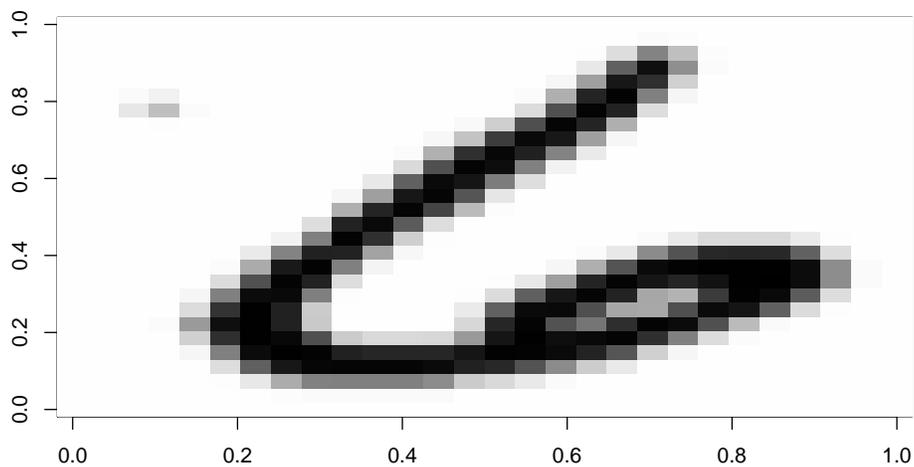


Quelques Jeux de Données Massifs

Label: 6

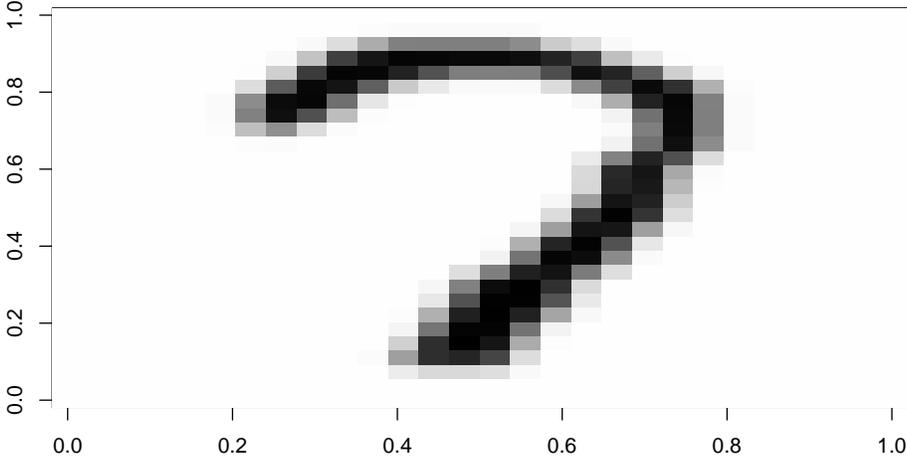


Label: 6

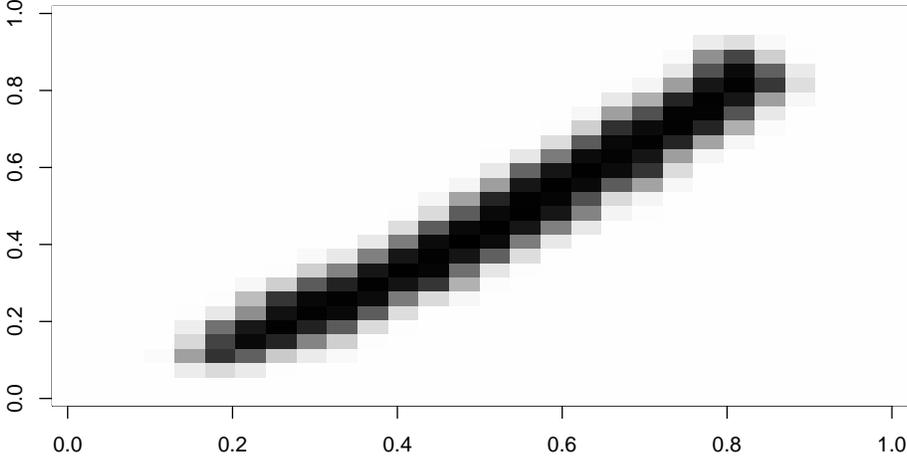


Quelques Jeux de Données Massifs

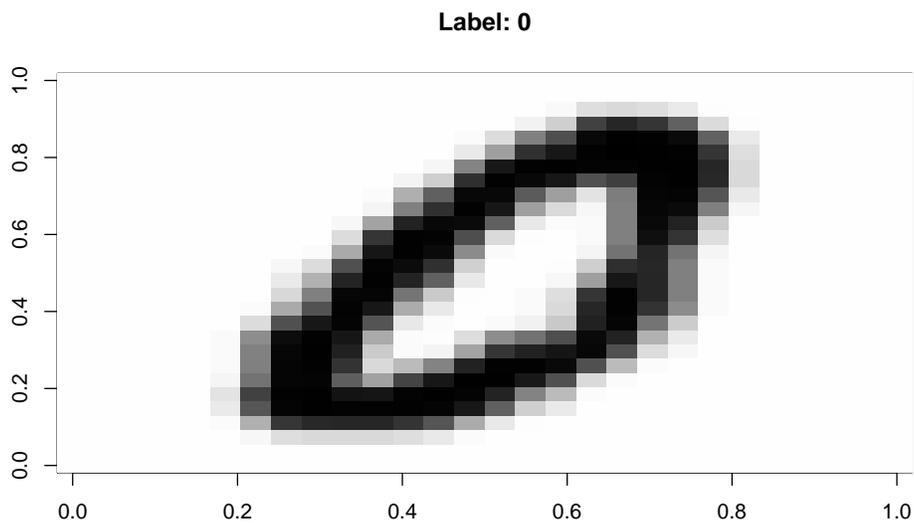
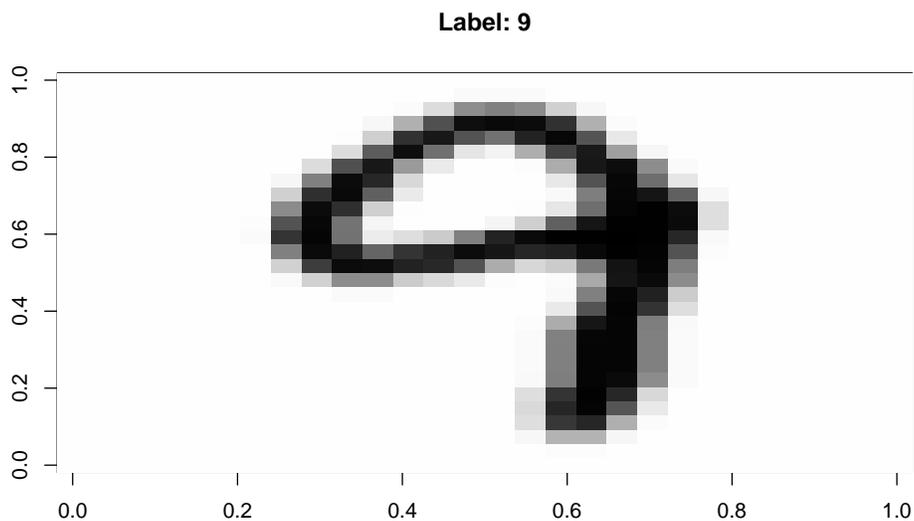
Label: 7



Label: 1



Quelques Jeux de Données Massifs



```
# Taille totale de toutes les images (en GB):  
(as.numeric(object.size(images_training_set)) / 10) * 280000 / 1024 / 1024 / 1024  
## [1] 1.641393
```

1.3 Matrices des Données EMNIST

Exercice: Créer un fichier CSV contenant les données des images.

```
setwd(WORKDIR.Data)  
extract_images_to_matrix <- function(file, nbimages = NULL) {  
  if (is.null(nbimages)) { # On extrait toutes les images  
    nbimages <- readBinData(file, 5, 8)  
  }  
  nbrows <- readBinData(file, 9, 12)  
  nbcols <- readBinData(file, 13, 16)  
  raw <- readBin(file, "raw", n = nbimages * nbrows * nbcols + 16)[-1:16]
```

Quelques Jeux de Données Massifs

```
return(matrix(as.numeric(paste("0x", raw, sep="")), ncol = nbcols * nbrows,
             nrow = nbimages, byrow = TRUE))
}
images_training_set_matrix <- extract_images_to_matrix(file_training_set_image)
write.table(images_training_set_matrix, file = "emnist_images_training_set.csv",
            row.names = FALSE, col.names = FALSE, sep = ",")
rm(images_training_set_matrix)
images_test_set_matrix <- extract_images_to_matrix(file_test_set_image)
write.table(images_test_set_matrix, file = "emnist_images_test_set.csv",
            row.names = FALSE, col.names = FALSE, sep = ",")
rm(images_test_set_matrix)
labels_training_set <- extract_labels(file_training_set_label)
write.table(labels_training_set, file = "emnist_labels_training_set.csv",
            row.names = FALSE, col.names = FALSE, sep = ",")
labels_test_set <- extract_labels(file_test_set_label)
write.table(labels_test_set, file = "emnist_labels_test_set.csv",
            row.names = FALSE, col.names = FALSE, sep = ",")
```

Exercice: Combien de temps cela prend t-il pour lire le fichier "emnist_images_training_set.csv" ?

1.4 Les Données EMNIST au Format `big.matrix`

Télécharger tous les six fichiers "emnist*" ici: https://www.dropbox.com/sh/de1x682sdlbub0r/AAA_8aCp4vs-pBEzWX9PP2ga/Lecture2?dl=0&subfolder_nav_tracking=1

puis exécuter les commandes ci-dessous.

```
R> library("bigmemory")
R> # Takes around 23s:
R> system.time({
X <- read.big.matrix("emnist_images_training_set.csv",
header = FALSE, type = "integer",
backingfile = "emnist_images_training_set.bin",
descriptorfile = "emnist_images_training_set.desc")
})
R> dim(X)
R> # Takes around 4s:
R> system.time({
X <- read.big.matrix("emnist_images_test_set.csv",
header = FALSE, type = "integer",
backingfile = "emnist_images_test_set.bin",
descriptorfile = "emnist_images_test_set.desc")
})
R> dim(X)
R> q("no")
```

Exercice: Comparer le temps de lecture à celui de la section précédente.

2 Le Jeu de Données Airlines

Arrivées et départs pour tous les vols commerciaux aux USA entre octobre 1997 et avril 2008 (en fait seulement pour les compagnies ayant au moins 1% de vols domestiques pour une année donnée).

Environ 120 million d'enregistrements, 29 variables (la plupart à valeurs entières).

On a pré-traité les données afin de créer un seul fichier CSV, en recodant le code transporteur (carrier code), le numéro d'aile d'avion (plane tail number), et les codes des aéroports (airport codes) comme des entiers.

Environ 3.5 milliards d'éléments. Susceptible de dépasser la RAM disponible.

Nous allons construire un gros fichier (12GB) et utiliser le package R `bigmemory` (matrices massives dans des fichiers mappés en mémoire):

- Algorithmes en mémoire externe;
- Calculs distribués;
- Opérer sur les données un "chunk" (gros morceau) à la fois.

2.1 Accéder aux Données Airlines

Les données brutes peuvent être téléchargées depuis:

<https://www.bts.gov/browse-statistical-products-and-data/bts-publications/airline-service-quality-performance-234-time>

ou depuis ici:

https://www.transtats.bts.gov/DL_SelectFields.asp?Table_ID=236&DB_Short_Name=On-Time

Mais on utilisera plutôt les données disponibles à

<http://stat-computing.org/dataexpo/2009/>

car elles ont été organisées.

Cela prend environ 25 mins.

```
$ cd Data
$ wget http://stat-computing.org/dataexpo/2009/{1987..2008}.csv.bz2 # 7.1GB, environ 15mns
$ bzip2 -d *.bz2 # Attendre ... pas mal de temps! (environ 10 mins)
$ wget http://stat-computing.org/dataexpo/2009/airports.csv
$ wget http://stat-computing.org/dataexpo/2009/carriers.csv
$ wget http://stat-computing.org/dataexpo/2009/plane-data.csv
```

Utilisateurs Windows: télécharger les fichiers depuis <http://stat-computing.org/dataexpo/2009/the-data.html> et les décompresser en utilisant <http://www.7-zip.org/>

Avez-vous une stratégie (un outil) pour éviter d'avoir à télécharger les fichiers un par un (en cliquant sur leur lien)?

2.2 Le Jeu de Données Airlines: Première Inspection

Inspecter les fichiers que vous avez téléchargés:

Quelques Jeux de Données Massifs

```
head 1987.csv
head -n 1 1988.csv
```

Lire la page <http://stat-computing.org/dataexpo/2009/the-data.html> pour une description des champs dans le jeu de données.

Garder une copie de 1987.csv pour une utilisation ultérieure:

```
$ cp 1987.csv keep-1987.csv
```

2.3 Le Jeu de Données Airlines: Premières Manipulations

Ordonner suivant la 10ème colonne (flightnum):

```
$ sort -t, -k 10,10 2008.csv
```

Filtrer des lignes: `awk` ou `sed`

Montrer les vols de Des Moines à Chicago O'hare:

```
$ awk -F, '$17 == "DSM" && $18 == "ORD"' 2008.csv
```

Extraire la ligne 7

```
$ sed '7q;d' 2008.csv
```

Filtrer des colonnes: `cut`

Sélectionner seulement les colonnes 9 (carrier) et 10 (flight num):

```
$ cut -f9,10 -d, 2008.csv
```

Compter le nombre de vols pour chaque numéro de vol (flight number) et sauver dans 2008-flights.csv:

```
$ cut -f9,10 -d, 2008.csv | sort | uniq -c > 2008-flights.csv
```

La dernière colonne semble contenir beaucoup de valeurs manquantes (NA). Combien?

```
$ cut -f29 -d, 2008.csv | less
```

```
$ cut -f29 -d, 2008.csv | grep NA | wc -l
```

Combien de valeurs non manquantes (pas NA)?:

```
$ cut -f29 -d, 2008.csv | grep -v NA | wc -l
```

Nombre d'entrées uniques dans la colonne 11 (deux façons):

```
$ cut -f11 -d, 2008.csv | tail -n +2 | sort | uniq | wc -l
```

```
$ cut -f11 -d, 2008.csv | tail -n +2 | awk '!seen[$0]++' | wc -l
```

Exercice: Pouvez-vous facilement faire la même chose en R (ou Python)?

2.4 Le Jeu de Données Airlines: Un Seul Fichier

Disons que l'on veut concaténer tous les fichiers. Nous devons d'abord enlever les premières lignes (en-tête = header) de plusieurs fichiers. Essayons cela avec R sur un seul fichier:

Quelques Jeux de Données Massifs

```
R> system.time({
X <- read.csv("1987.csv", header = TRUE)
write.table(X[-1, ], file = "1987-copy.csv",
quote = FALSE, sep = ",", row.names = FALSE,
col.names = FALSE)
})
q("no")
```

Cela prend environ 17s sur mon ordinateur portable.

```
$ less 1987-copy.csv
```

L'en-tête a bien disparu.

2.5 Le Jeu de Données Airlines: Avec Linux I

Maintenant, mesurons le temps d'exécution pour la même chose avec divers outils Linux. (Chaque fois, nous créons une copie de `1987.csv` afin d'éviter de corrompre le fichier original.)

```
$ rm -f 1987-copy.csv
$ cp 1987.csv 1987-copy.csv
$ time -p sh -c "awk -F, '\$NR != 1' 1987-copy.csv | sponge 1987-copy.csv"
$ rm -f 1987-copy.csv
$ cp -f 1987.csv 1987-copy.csv
$ time -p sh -c 'sed -i '1d' 1987-copy.csv'
$ rm -f 1987-copy.csv
$ cp -f 1987.csv 1987-copy.csv
$ time -p sh -c "ex -c ':1d' -c ':wq' 1987-copy.csv"
```

2.6 Le Jeu de Données Airlines: Avec Linux II

Maintenant, mesurons le temps d'exécution pour la même chose avec d'autres outils Linux.

```
$ rm -f 1987-copy.csv
$ cp -f 1987.csv 1987-copy.csv
$ time -p sh -c 'echo -e "$(sed '1d' 1987-copy.csv)\n" > 1987-copy.csv'
$ rm -f 1987-copy.csv
$ cp -f 1987.csv 1987-copy.csv
$ time -p sh -c 'echo "$(tail -n +2 1987-copy.csv)" > 1987-copy.csv'
$ rm -f 1987-copy.csv
$ cp -f 1987.csv 1987-copy.csv
$ time -p sh -c "awk 'NR>2 {print l} {l=\$0}' 1987-copy.csv | sponge 1987-copy.csv"
```

Pas si mal, hein?

2.7 Le Jeu de Données Airlines: Avec Linux III

Peut-on faire mieux?

```
$ rm -f 1987-copy.csv
$ cp -f 1987.csv 1987-copy.csv
```

Quelques Jeux de Données Massifs

```
$ time -p sh -c "awk 'NR > 1 { print }' 1987-copy.csv | sponge 1987-copy.csv"
$ rm -f 1987-copy.csv
$ cp -f 1987.csv 1987-copy.csv
$ time -p sh -c 'sed "1d" 1987-copy.csv | sponge 1987-copy.csv'
$ rm -f 1987-copy.csv
$ cp -f 1987.csv 1987-copy.csv
$ time -p sh -c 'more +2 1987-copy.csv | sponge 1987-copy.csv'
$ rm -f 1987-copy.csv
$ cp -f 1987.csv 1987-copy.csv
$ time -p sh -c 'tail -n +2 1987-copy.csv | sponge 1987-copy.csv'
```

2.8 Le Jeu de Données Airlines

Comme nous pouvons le vérifier, une ligne (la première) a été supprimée:

```
$ wc -l 1987.csv
$ wc -l 1987-copy.csv
$ less 1987-copy.csv
```

2.9 Le Jeu de Données Airlines

Maintenant, combinons deux fichiers (encore une fois, on travaille sur des copies):

```
$ rm -f 1987-copy.csv
$ cp -f 1987.csv 1987-copy.csv
$ wc -l 1987-copy.csv
$ cp -f 1988.csv 1988-copy.csv
$ wc -l 1988-copy.csv
$ find *copy* -type f -exec cat {} + | wc -l
$ time -p sh -c 'cat 1987-copy.csv 1988-copy.csv | sponge 1988-copy.csv'
$ wc -l 1988-copy.csv
```

2.10 Le Jeu de Données Airlines

Peut-on faire cela plus rapidement?

```
$ rm -f 1987-copy.csv
$ cp -f 1987.csv 1987-copy.csv
$ rm -f 1988-copy.csv
$ cp -f 1988.csv 1988-copy.csv
$ find *copy* -type f -exec cat {} + | wc -l
$ time -p sh -c 'cat 1988-copy.csv >> 1987-copy.csv'
$ wc -l 1987-copy.csv
$ rm -f 1987-copy.csv
$ rm -f 1988-copy.csv
```

Quelques Jeux de Données Massifs

2.11 Le Jeu de Données Airlines

Maintenant, combinons tous les fichiers. Cela prend environ 2 mins avec des outils en ligne de commandes (Vous pouvez essayer de le faire avec R si vous voulez):

```
$ nbfiles="$(ls {1987..2008}.csv|wc -l)"
$ nblinesbefore="$(find {1987..2008}.csv -type f -exec cat {} + | wc -l)"
$ ls ????.csv | du -sh
$ # Remove all headers (except for 1987.csv).
$ for res in {1988..2008}; do
  tail -n +2 ${res}.csv | sponge ${res}.csv
  cat ${res}.csv >> 1987.csv
  rm ${res}.csv
done; mv 1987.csv airline.csv
$ echo "It is done!"
$ nblinesafter="$(find airline.csv -type f -exec cat {} + | wc -l)"
$ ls -alh airline.csv
$ # Check that the sum of number of lines (-l) of all
$ # files is equal to number of lines of the merged file.
$ echo "$nblinesbefore - $nbfiles + 1"|bc
$ echo $nblinesafter
```

Maintenant, on a un seul gros fichier CSV (comma separated values) de 12 GB appelé `airline.csv`.

2.12 Le Jeu de Données Airlines

D'abord, lançons le moniteur de mémoire (e.g., `gnome-system-monitor` &) pour voir comment la mémoire va être consommée.

```
$ free -h
```

```
R> X <- read.csv("airline.csv", header = TRUE)
```

Comme vous pouvez le voir, cela va probablement échouer, ou prendre un temps infini, en épuisant la Mémoire ou la Swap.

2.13 Le Jeu de Données Airlines

Maintenant, nous utilisons le package R `bigmemory` pour lire le fichier.

```
R> library("bigmemory")
R> # Prend environ 13 mins:
R> system.time({
X <- read.big.matrix("airline.csv",
header = TRUE, type = "integer",
backingfile = "airline.bin",
descriptorfile = "airline.desc")
})
R> dim(X)
R> nrow(X)
R> sum(is.na(X[, 17]))
R> sum(is.na(X[, 18]))
```

Quelques Jeux de Données Massifs

```
R> sum(is.na(X[, 23]))
R> q("no")
```

2.14 Le Jeu de Données Airlines

Malheureusement, comme vous pouvez vous en douter, tous les codes de caractères des fichiers (i.e., colonnes 9, 17, 18, 23) ont été remplacés par des valeurs NA. C'est car nous avons utilisé le type "integer" dans `read.big.matrix()`. Utiliser le type "char" n'aidera pas (car c'est seulement une manière que C a de stocker des valeurs sur un seul octet, donc avec seulement 256 valeurs différentes). Cela pourrait changer dans le futur (on espère!) mais pour le moment on doit gérer cette situation (i.e., remplacer toutes les entrées de caractères par un code numérique entier).

```
$ cut -f1 -d, airline.csv|tail -n +2|awk '!seen[$0]++'
$ cut -f2 -d, airline.csv|tail -n +2|awk '!seen[$0]++'
$ cut -f23 -d, airline.csv|tail -n +2|awk '!seen[$0]++'
```

2.15 Le Jeu de Données Airlines

Créer une table de valeurs uniques avec des codes entiers (comme `factor()` dans R):

```
$ sudo apt-get install john
$ sudo ln -s /usr/sbin/unique /usr/bin/unique
$ time -p sh -c "cut -f9 -d, keep-1987.csv | tail -n +2 | sort \
  | uniq | cat -n | sed 's/ \+ //g' | sed 's/\t/,/g'"
```

L'instruction ci-dessous est plus rapide (mais sans ordonner les valeurs cependant):

```
$ time -p sh -c 'cut -f9 -d, keep-1987.csv | tail -n +2 \
  | awk "!seen[\$0]++" | cat -n | sed "s/ \+ //g" \
  | sed "s/\t/,/g"'
```

2.16 Le Jeu de Données Airlines

Créer une table de valeurs uniques (classées) avec des codes entiers (comme `factor()` en R) pour toutes les colonnes de `airline.csv` (prend environ 18 mins):

```
$ date ; for i in {1..29}; do
cut -f${i} -d, airline.csv | tail -n +2 | \
awk '!seen[$0]++' | grep -v NA | sort | cat -n | \
sed 's/ \+ //g' | sed 's/\t/,/g' > \
tmp${i}.txt
done; date
$ cat tmp23.txt
```

Inspecter rapidement tous les autres fichiers `tmp*.txt` (e.g., en utilisant `less`). Le faire sur le fichier `tmp23.txt` me montre que "" est une valeur possible!

2.17 Le Jeu de Données Airlines

Alors maintenant, on doit remplacer „ par ,NA, dans le fichier `airline.csv`. On remplace aussi toutes les entrées de caractères par leurs codes correspondants (tels que donnés dans les fichiers `tmp*.txt` créés plus haut).

NR: Le nombre total d'enregistrements en entrée jusqu'ici.

FNR Le nombre d'enregistrements en entrée dans le fichier d'entrée courant.

Prend environ 10 mins. Vous avez besoin de suffisamment d'espace sur `/tmp`.

```
$ date ; awk -F, ' (FILENAME == ARGV[1]){a[$2]=$1;next} \
(FILENAME == ARGV[2]){b[$2]=$1;next}(FILENAME == ARGV[3]) \
{c[$2]=$1;next}(FILENAME == ARGV[4]){d[$2]=$1;next} \
((FNR > 1) && FILENAME == ARGV[5]){OFS=",";$9=a[$9]; \
$17=b[$17];$18=c[$18];$23=d[$23];gsub (",",",NA,", $0); \
}1' tmp9.txt tmp17.txt tmp18.txt tmp23.txt airline.csv | \
sponge airline.csv ; date
$ rm tmp*.txt
```

2.18 Le Jeu de Données Airlines

Maintenant, on utilise le package R `bigmemory` pour lire le fichier `airline.csv` (environ 11 mins) et créer les fichiers de backend `airline.bin` et `airline.desc`. On ajoute une colonne supplémentaire `age` qui sera remplie plus tard.

```
R> library("bigmemory")
R> # Prend environ 16 mins:
R> system.time({
X <- read.big.matrix("airline.csv",
header = TRUE, type = "integer",
backingfile = "airline.bin",
descriptorfile = "airline.desc",
extraCols = "age")
})
R> dim(X)
R> q("no")
```

2.19 Le Jeu de Données Airlines

La plupart de ce qui suit provient des slides d'un talk donné par John W. Emerson "Jay" and Michael J Jane de l'université de Yale, à la conférence UseR! 2009. Ils ont utilisé le jeu de données `airline.csv` qui malheureusement n'est pas bien formé (nous y reviendrons).

<https://www.r-project.org/conferences/useR-2009/slides/Emerson+Kane.pdf>

2.20 Le Jeu de Données Airlines

Des sessions subséquentes peuvent se connecter instantanément au backend, et on peut interagir avec celui-ci (e.g., calculer quelques statistiques):

Quelques Jeux de Données Massifs

```
R> library("bigmemory")
R> xdesc <- dget("airline.desc")
R> xdesc
R> system.time(x <- attach.big.matrix(xdesc))
R> colnames(x)
R> head(x)
R> system.time(a <- x[,1])
R> max(x[,1]) # Cela ne marche pas!
R> range(x[,1], na.rm = TRUE)
R> tail(x, 1)
```

2.21 Le Jeu de Données Airlines

Pouvons nous avoir tous les vols de JFK à SFO? Certainement!

```
a <- read.table("tmp17.txt", sep = ",")
JFK <- a$aV1[a$aV2 == "JFK"]
SFO <- a$aV1[a$aV2 == "SFO"]
gc(reset=TRUE)
system.time(
y <- x[x[, "Origin"] == JFK & x[, "Dest"] == SFO,]
)
dim(y)
gc()
rm(y)
```

2.22 Le Jeu de Données Airlines

```
library("biganalytics")
# L'étendue des valeurs pour la première colonne
colmean(x, 1, na.rm = TRUE)
# La première colonne est mise en cache.
# Une deuxième opération sur la colonne est rapide.
colrange(x, 1, na.rm = TRUE)
```

2.23 Le Jeu de Données Airlines

Quelle est la meilleure heure de la journée pour voler afin de minimiser les retards? Un calcul simple fait en parallèle sur 3 coeurs.

```
$ gnome-system-monitor
R> library("foreach")
R> library("doMC")
R> registerDoMC(cores = 3)
R> probs <- c(0.9, 0.99, 0.999, 0.9999)
R> desc <- describe(x)
```

2.24 Le Jeu de Données Airlines

Quelques Jeux de Données Massifs

```
R> # Retards par heure du jour.
R> anshourofday <-
foreach (i = seq(0, colmax(x, "CRSDepTime") - 1, by=60),
.combine = cbind)%dopar%
{
library("bigmemory")
x <- attach.big.matrix(desc)
ind <- mwhich(x, "CRSDepTime", c(i, i + 60),
comps = c('ge', 'lt'))
m <- cbind(probs, quantile(x[ind, "DepDelay"],
probs = probs, na.rm = TRUE))
colnames(m) <- c("Probabilites", "Quantiles")
t(m)[2,]
}
```

2.25 Le Jeu de Données Airlines

`CRSDepTime` : heure de départ prévu (local, hhmm).

`DepDelay` : retard du départ, en minutes.

Ce qui a été fait juste au-dessous est **incorrect!**. En effet, la variable `CRSDepTime` devrait être au format hhmm dans le fichier `airline.csv` mais ce n'est pas le cas. De plus, ils supposent que cette variable est en minutes, ce qui est faux!

Même des chercheurs d'une des meilleures universités du monde (Yale) peuvent se tromper! Ne pas négliger l'étape d'exploration des données!

Que faire?

Revenir au site web d'origine pour télécharger les données:

http://www.transtats.bts.gov/DL_SelectFields.asp?Table_ID=236&DB_Short_Name=On-Time

On ne va pas le faire car cela prendrait trop de temps. Aussi, ce qui pourrait être intéressant serait d'écrire des scripts pour télécharger les fichiers automatiquement (faire du *web scrapping*), sans avoir besoin de cliquer sur la page web.

2.26 Le Jeu de Données Airlines

Est-ce que les vieux avions ont plus de retard? Peut être. Un calcul intensif fait en parallèle.

```
uniqTailNum <- na.omit(unique(x[, 11]))
uniqTailNum.len <- length(uniqTailNum)
# 166 avions différents dont TailNum est connu
planeStart <- big.matrix(nrow = uniqTailNum.len,
ncol = 1, shared = TRUE)
psDesc <- describe(planeStart)
foreach(i=1:uniqTailNum.len) %dopar%
{
library("bigmemory")
x <- attach.big.matrix(desc)
```

```
planeStart <- attach.big.matrix(psDesc)
```

2.27 Le Jeu de Données Airlines

Première année où l'avion i peut être trouvé:

```
yearInds <- mwhich(x, "TailNum", uniqTailNum[i],
  comps = 'eq')
minYear <- min( x[yearInds, "Year"], na.rm = TRUE )
# First month in minYear where the plane can be found:
minMonth <- min( x[yearInds, "Month"], na.rm = TRUE )
planeStart[i, 1] <- 12 * minYear + minMonth
return(TRUE)
}
```

2.28 Le Jeu de Données Airlines

```
BadTailNum <- mwhich(x, 11, NA, 'eq')
x[BadTailNum, 30] <- NA
MoreRecentDate <- max(x[,1]) * 12 + max(x[,2])
system.time(foreach(i=1:uniqTailNum.len) %dopar%
{
  library("bigmemory")
  x <- attach.big.matrix(desc)
  planeStart <- attach.big.matrix(psDesc)
  tmpInds <- mwhich(x, 11, uniqTailNum[i], 'eq')
  x[tmpInds, 30] <- as.integer(MoreRecentDate -
    planeStart[i, 1])
})
```

Colonne "age" (i.e., 30) de x est maintenant remplie.

2.29 Le Jeu de Données Airlines

```
blm1 <- biglm.big.matrix(ArrDelay ~ age, data = x)
( out <- summary(blm1) )
names(out)
out$rsq
blm2 <- biglm.big.matrix(ArrDelay ~ age + Year, data = x)
summary(blm2)
```

3 Le Jeu de Données The GWAS

Ce jeu de données a été créé par Yaohui Zeng et Patrick Breheny pour montrer la scalabilité de leur approche développée dans "The biglasso Package: A Memory-and Computation-Efficient Solver for Lasso Model Fitting with Big Data in R"

Les données incluent:

Quelques Jeux de Données Massifs

- Features X: X_3000_1340000_200_logistic.txt (n=3000 sujets et $p=1340000$ gènes)
- Réponse y : y_3000_1340000_200_gwas.txt (étude cas/contrôle)

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

4 Le Prix Netflix 2006–2009 (source: Hastie)

Rank	Team Name	Best Test Score	% Improvement	Best Submit Time
Grand Prize - RMSE = 0.8567 - Winning Team: BellKor's Pragmatic Chaos				
1	BellKor's Pragmatic Chaos	0.8567	10.06	2009-07-26 18:18:28
2	The Ensemble	0.8567	10.06	2009-07-26 18:38:22
3	Grand Prize Team	0.8582	9.90	2009-07-10 21:24:40
4	Opera Solutions and Vandelay United	0.8586	9.84	2009-07-10 01:12:31
5	Vandelay Industries!	0.8591	9.81	2009-07-10 00:32:20
6	PragmaticTheory	0.8594	9.77	2009-06-24 12:06:56
7	BellKor in BigChaos	0.8601	9.70	2009-05-13 08:14:09
8	Dace_	0.8612	9.59	2009-07-24 17:18:43
9	Feeds2	0.8622	9.48	2009-07-12 13:11:51
10	BigChaos	0.8623	9.47	2009-04-07 12:33:59
11	Opera Solutions	0.8623	9.47	2009-07-24 00:34:07
12	BellKor	0.8624	9.46	2009-07-26 17:19:11

4.1 Le Prix Netflix

- compétition démarrée en octobre 2006. Training data sont les scores (ratings) pour 18K films faits par 480K clients Netflix, chaque score entre 1 et 5.
- training data est très sparse (environ 99% de valeurs manquantes.)
- l'objectif est de prédire le score pour un ensemble de 1 million de paires (client, film) qui sont manquantes dans le training data.
- L'algorithme original de Netflix a une root MSE de 0.953. La première équipe qui atteint une amélioration de 10% gagne un million de dollars.

Quelques Jeux de Données Massifs

	<i>movie I</i>	<i>movie II</i>	<i>movie III</i>	<i>movie IV</i>	...
User A	1	?	5	4	...
User B	?	2	3	?	...
User C	4	1	2	?	...
User D	?	5	1	3	...
User E	1	2	?	?	...
⋮	⋮	⋮	⋮	⋮	⋮

Quelques références:

<https://sifter.org/~simon/journal/20061211.html>

<https://datajobs.com/data-science-repo/Recommender-Systems-%5BNetflix%5D.pdf>

http://nicolas-hug.com/blog/matrix_facto_1

http://nicolas-hug.com/blog/matrix_facto_2

http://nicolas-hug.com/blog/matrix_facto_4 http://nicolas-hug.com/blog/matrix_facto_3

4.2 Le Prix Netflix

Comment obtenir le fichier depuis Kaggle:

1. Register (e.g., d'abord créer un compte manuellement) sur Kaggle: <https://www.kaggle.com/>
2. Installer l'API Kaggle (nécessite Python 3): <https://github.com/Kaggle/kaggle-api>
Pour un utilisateur Linux: `pip install kaggle`
3. Aller sur l'onglet 'Account' de votre profil utilisateur (<https://www.kaggle.com/username/account>) et sélectionner 'Create API Token'. Cela va déclencher le téléchargement de `kaggle.json`, un fichier contenant vos identifiants API. Placer ce fichier ici: `~/.kaggle/kaggle.json`
4. Taper ceci dans un terminal: `cd Data; ~/.local/bin/kaggle datasets download -d netflix-inc/netflix-prize-data` pour télécharger le jeu de données du Netflix Prize depuis <https://www.kaggle.com/netflix-inc/netflix-prize-data>
5. Décompresser le fichier `netflix-prize-data.zip`
6. Installer GNU awk (gawk): Linux (`sudo apt-get install gawk`), Windows (<http://gnuwin32.sourceforge.net/packages/gawk.htm>), MacOS

Quelques Jeux de Données Massifs

7. Créer le fichier `process.txt` contenant les lignes suivantes:

```
BEGIN {
    OFS = ","
    FS = ","
}
{
if (match($0, /\.*\:/)) {
    id_movie = substr($0, 1, length($0) - 1)
} else {
    year = substr($3, 1, 4)
    month = substr($3, 6, 2)
    print id_movie, $1, $2, year, month
}
}
```

8. Taper les commandes suivantes dans un terminal Linux pour créer le fichier de 2.2GB nommé `netflix.csv`:

```
awk -f process.txt combined_data_1.txt > netflix_1.txt
awk -f process.txt combined_data_2.txt > netflix_2.txt
awk -f process.txt combined_data_3.txt > netflix_3.txt
awk -f process.txt combined_data_4.txt > netflix_4.txt
head netflix_1.txt
tail netflix_4.txt
mv netflix_1.txt netflix.csv
cat netflix_2.txt >> netflix.csv
rm netflix_2.txt
cat netflix_3.txt >> netflix.csv
rm netflix_3.txt
cat netflix_4.txt >> netflix.csv
rm netflix_4.txt
wc -l netflix.csv
ls -alh netflix.csv
```

9. Taper les commandes suivantes pour créer les fichiers R de backend:

```
R> library("bigmemory")
R> # Prend environ 2 mins:
R> system.time({
X <- read.big.matrix("netflix.csv",
    header = FALSE, type = "integer",
    backingfile = "netflix.bin",
    descriptorfile = "netflix.desc",
    col.names = c("movieID", "customerID", "rating",
        "rentalYear", "rentalMonth"))
})
R> dim(X)
R> q("no")
```

4.3 Le Prix Netflix

Créer la table tri à plat de `movieID` (prend environ 50s):

```
$ time -p sh -c "cut -f1 -d, netflix.csv | sort | uniq -c > table-movieID.txt"
```

Créer une table des valeurs uniques (ordonnées) avec un code entier pour toutes les 17~770 entrées `movieID` (environ 50s):

```
$ time -p sh -c "cut -f1 -d, netflix.csv | sort \  
    | uniq | cat -n | sed 's/ \+ //g' | sed 's/\t/,/g' > unique-movieID.txt"  
$ wc -l unique-movieID.txt
```

Créer une table des valeurs uniques (ordonnées) à code entier pour toutes les 480~189 entrées `customerID` (environ 3 mins):

```
$ time -p sh -c "cut -f2 -d, netflix.csv | sort \  
    | uniq | cat -n | sed 's/ \+ //g' | sed 's/\t/,/g' > unique-customerID.txt"  
$ wc -l unique-customerID.txt
```

```
movieCodes <- sort(read.csv("unique-movieID.txt", header = FALSE)[, 2])  
length(movieCodes)  
length(unique(movieCodes))  
range(movieCodes)  
tab.MovieID <- read.table("table-movieID.txt", header = FALSE, col.names = c("freq", "ID"))  
head(tab.MovieID)  
sum(tab.MovieID$freq)  
customerCodes <- sort(read.csv("unique-customerID.txt", header = FALSE)[, 2])  
length(customerCodes)  
length(unique(customerCodes))  
range(customerCodes)  
idx.customers <- matrix(NA, nrow = max(customerCodes), ncol = 2)  
idx.customers[, 1] <- 1:max(customerCodes)  
idx.customers[customerCodes, 2] <- 1:length(customerCodes)  
library("Matrix")  
M <- Matrix(0, nrow = length(customerCodes), ncol = length(movieCodes),  
    dimnames = list(as.character(customerCodes), NULL), sparse = TRUE)  
dim(M)  
object.size(M)  
con <- file("netflix.csv", open = 'r')  
nblines <- as.numeric(strsplit(system("wc -l netflix.csv",  
    intern = TRUE), " ")[[1]][1]) # Number of lines in the file  
# Beaucoup trop lent, prendra des jours!  
# system.time({  
# for (i in 1:nblines) {  
# if ((i %% 10000) == 0) print(i)  
# just.read <- as.numeric(read.csv(con, colClasses = integer(), nrows = 1,  
# header = FALSE))  
# # Ou de façon équivalente ici:  
# # just.read <- scan(con, what = integer(), n = 5, sep = ",", quiet = TRUE)  
# M[idx.customers[just.read[2], 2], just.read[1]] <- as.integer(just.read[3])  
# }
```

Quelques Jeux de Données Massifs

```
# })
# Essayons une autre approche (prend environ 9h)
library("bigmemory")
X <- dget("netflix.desc")
X <- attach.big.matrix(X)
dim(X)
nbmovies <- max(X[, 1])
system.time({
  for (i in 1:nbmovies) {
    print(nbmovies - i)
    tmp.idx <- X[, 1] == i
    M[idx.customers[X[tmp.idx, 2], 2], i] <- X[tmp.idx, 3]
  }
})
object.size(M)
writeMM(M, file = 'netflix.sparse')
M <- readMM(file = 'netflix.sparse')
```

Voir les données ici:

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

du Workshop:

<https://www.dropbox.com/sh/de1x682sdlbub0r/AAB-C0DG78sDefX3Pe7xZFVGa?dl=0>

5 Le Jeu de Données Bosch Production Line Performance

Ce jeu de données (Raw: 14.3GB) n'est pas traité dans ce cours. Mais vous pouvez l'utiliser pour explorer ce que vous aurez appris.

<https://www.kaggle.com/c/bosch-production-line-performance/data>

6 Quelques Autres Jeux de Données

https://en.wikipedia.org/wiki/List_of_datasets_for_machine_learning_research

<https://www.springboard.com/blog/free-public-data-sets-data-science-project/>