The R package {bigstatsr}:
memory- and computation-efficient tools for big matrices stored on disk

Florian Privé (@privefl)

Slides: `https://privefl.github.io/R-presentation/bigstatsr.html`

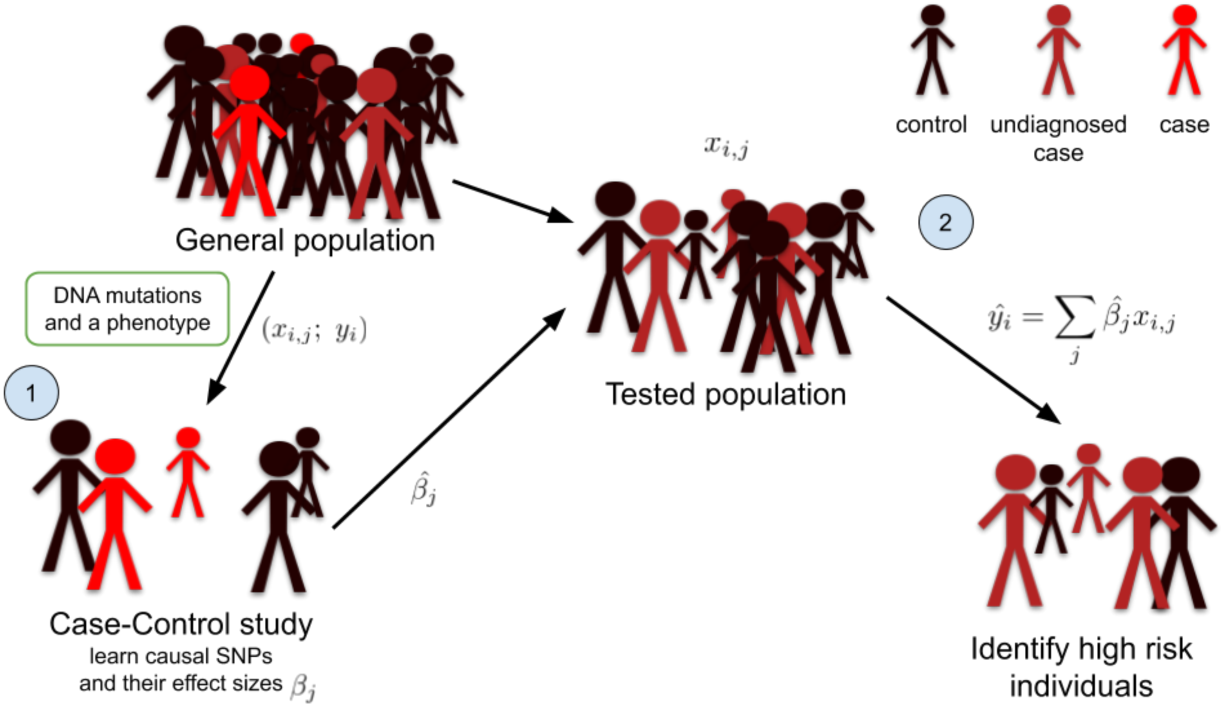
Installation: `remotes::install_github("privefl/bigstatsr")`

Motivation

My thesis work

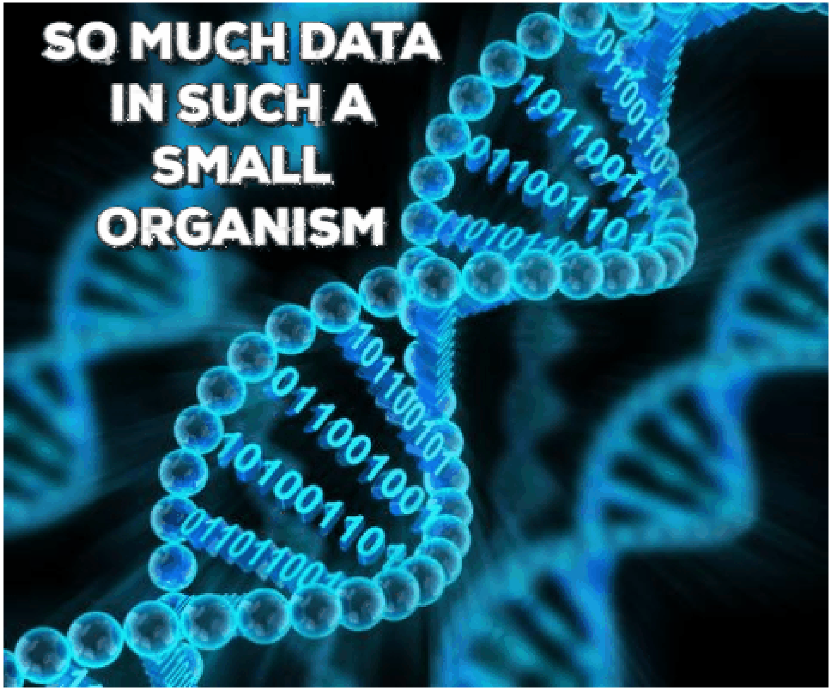
I'm a postdoc in **Predictive Human Genetics**.

$$\boxed{\text{Disease} \sim \text{DNA mutations} + \cdots}$$



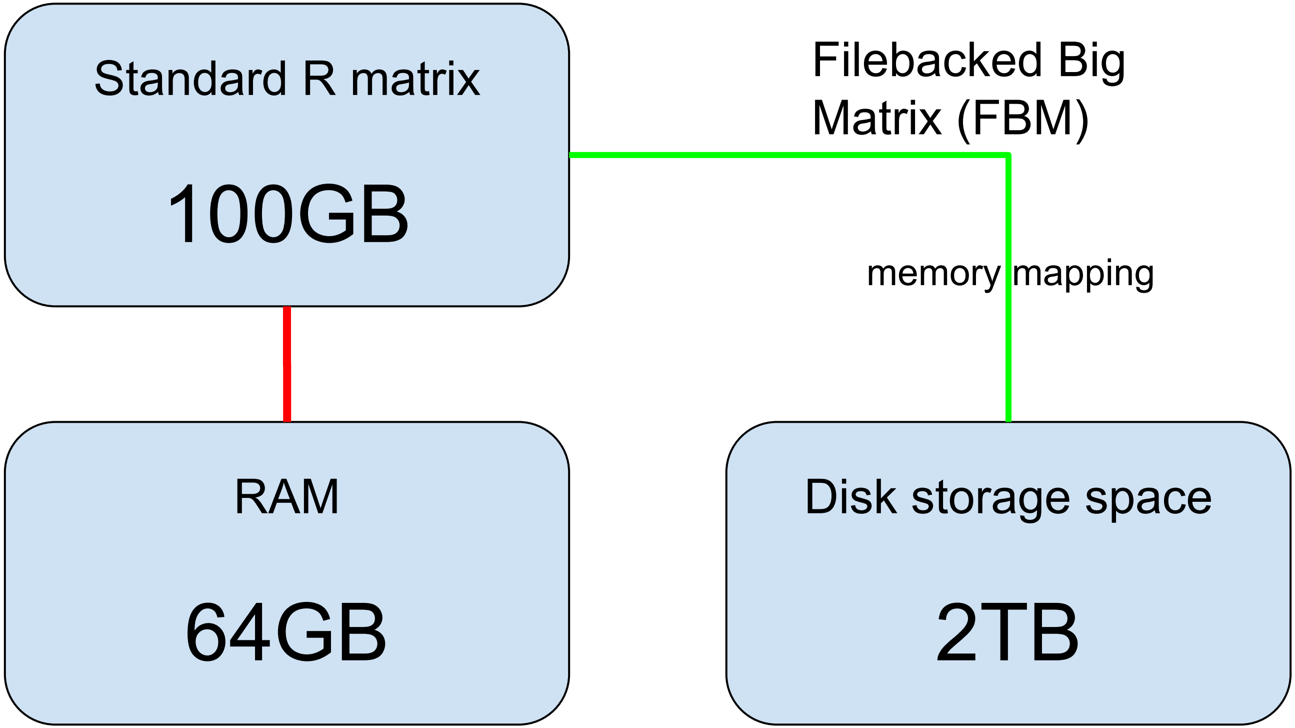
Very large genotype matrices

- previously: 15K x 280K, celiac disease (~30GB)
- currently: 500K x 500K, UK Biobank (~2TB)



But I still want to use R..

The solution I found



Format `FBM` is very similar to format `filebacked.big.matrix` from package {bigmemory} (details in this vignette).

Simple accessors

Similar accessor as R matrices

```
X <- FBM(2, 5, init = 1:10, backingfile = "test")
```

```
X$backingfile
```

```
## [1] "/home/privef/Bureau/R-presentation/test.bk"
```

```
X[, 1]  ## ok
```

```
## [1] 1 2
```

```
X[1, ]  ## bad
```

```
## [1] 1 3 5 7 9
```

```
X[]     ## super bad
```

```
##      [,1] [,2] [,3] [,4] [,5]
## [1,]    1    3    5    7    9
## [2,]    2    4    6    8   10
```

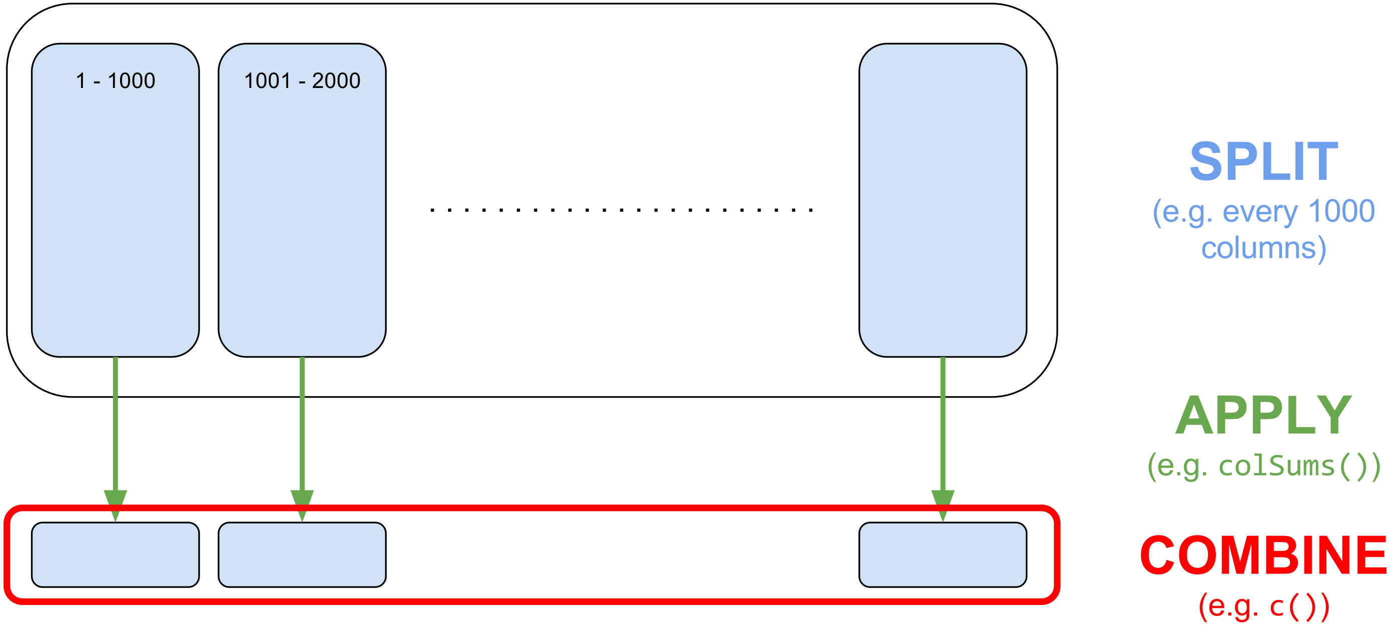
Similar accessor as R matrices

```
colSums(X[])  ## super bad
```

```
## [1]  3  7 11 15 19
```

Split-(par)Apply-Combine Strategy

Apply standard R functions to big matrices (in parallel)



SPLIT
(e.g. every 1000 columns)

APPLY
(e.g. `colSums()`)

COMBINE
(e.g. `c()`)

Implemented in `big_apply()`.

Similar accessor as Rcpp matrices

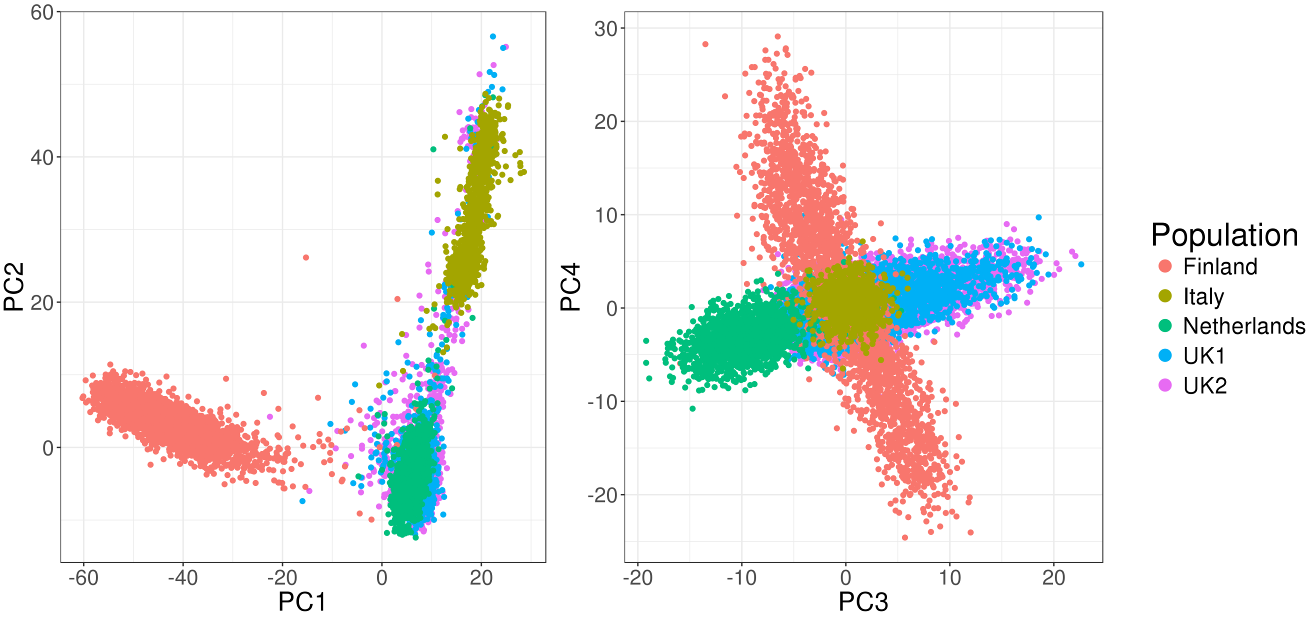
```
// [[Rcpp::depends(rmio, RcppArmadillo, bigstatsr)]]
#include <bigstatsr/BMAcc.h>

// [[Rcpp::export]]
NumericVector big_colsums(Environment BM) {

  XPtr<FBM> xpBM = BM["address"];
  BMAcc<double> macc(xpBM);

  size_t n = macc.nrow();
  size_t m = macc.ncol();

  NumericVector res(m);

  for (size_t j = 0; j < m; j++)
    for (size_t i = 0; i < n; i++)
      res[j] += macc(i, j);

  return res;
}
```

Some examples

from my work

Partial Singular Value Decomposition

15K × 100K -- 10 first PCs -- 6 cores -- **1 min** (vs 2h in base R)



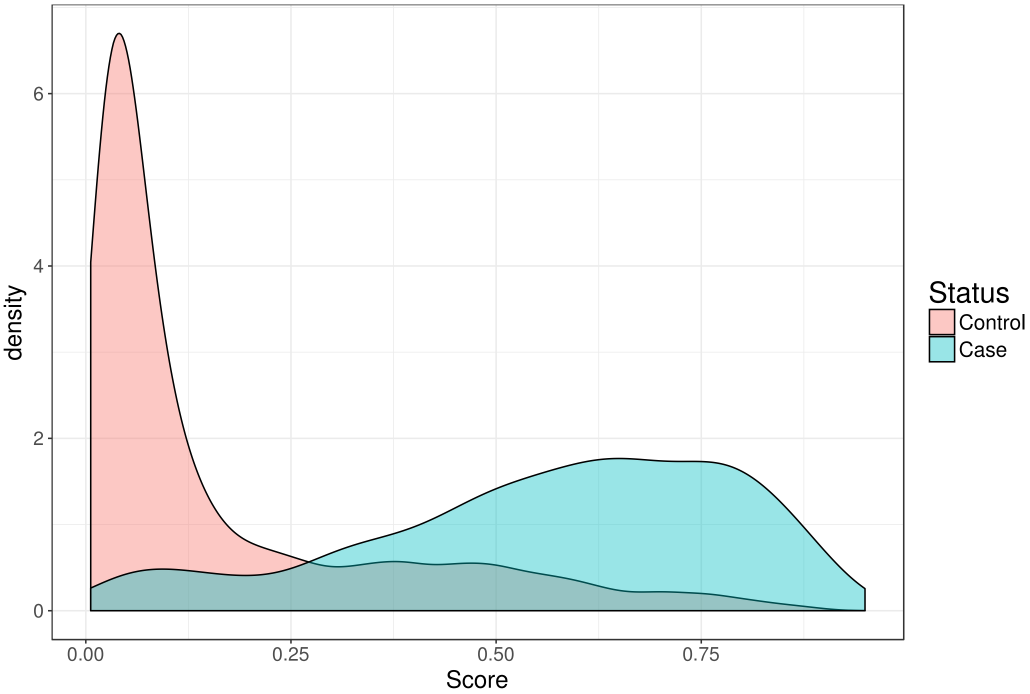
Implemented in `big_randomSVD()`, powered by R packages {RSpectra} and {Rcpp}.

Sparse linear models

Predicting complex diseases with a penalized logistic regression

15K × 280K -- 6 cores -- **2 min** (10x faster than {glmnet})

Automatic (parallel) grid-search for the two hyper-parameters of elastic-net.



Let us try

some functions

Create an FBM object

```
X <- FBM(10e3, 1000, backingfile = "test2")
object.size(X)
```

```
## 680 bytes
```

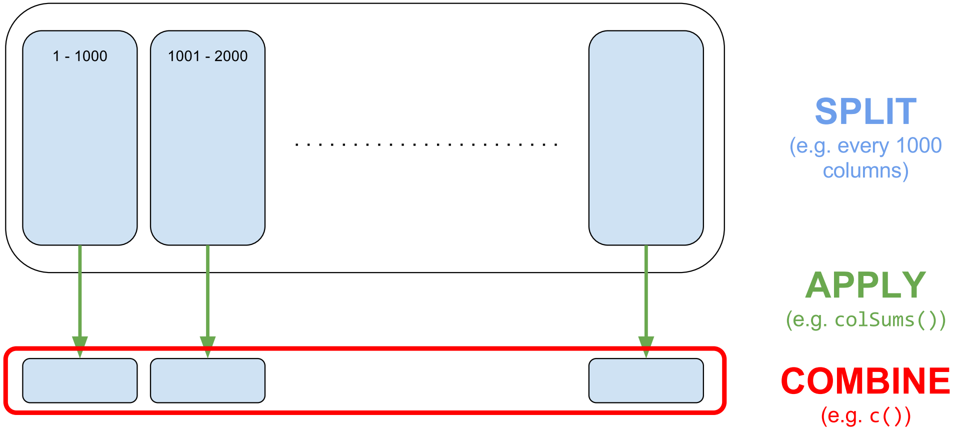
```
file.size(X$backingfile)  ## 8 x 1e4 x 1e3
```

```
## [1] 8e+07
```

```
typeof(X)
```

```
## [1] "double"
```

Fill it with random values



```
big_apply(X, a.FUN = function(X, ind) {
  X[, ind] <- rnorm(nrow(X) * length(ind))
  NULL  ## Here, you don't want to return anything
}, a.combine = 'c')
```

```
## NULL
```

```
X[1:5, 1]
```

```
## [1]  0.9049859  0.4069235  0.2709667 -1.7053191  1.0157806
```

Correlation matrix

```
mat <- X[]
system.time(corr1 <- cor(mat))
```

```
##    user  system elapsed 
##   7.226   0.008   7.243
```

```
system.time(corr2 <- big_cor(X))
```

```
##    user  system elapsed 
##   0.452   0.062   0.514
```

```
all.equal(corr1, corr2[])
```

```
## [1] TRUE
```

Partial Singular Value Decomposition

```
system.time(svd1 <- svd(scale(mat), nu = 10, nv = 10))
```

```
##    user  system elapsed
##   3.802   0.330   4.142
```

```
# Quadratic in the smallest dimension, linear in the other one
system.time(svd2 <- big_SVD(X, fun.scaling = big_scale(), k = 10))
```

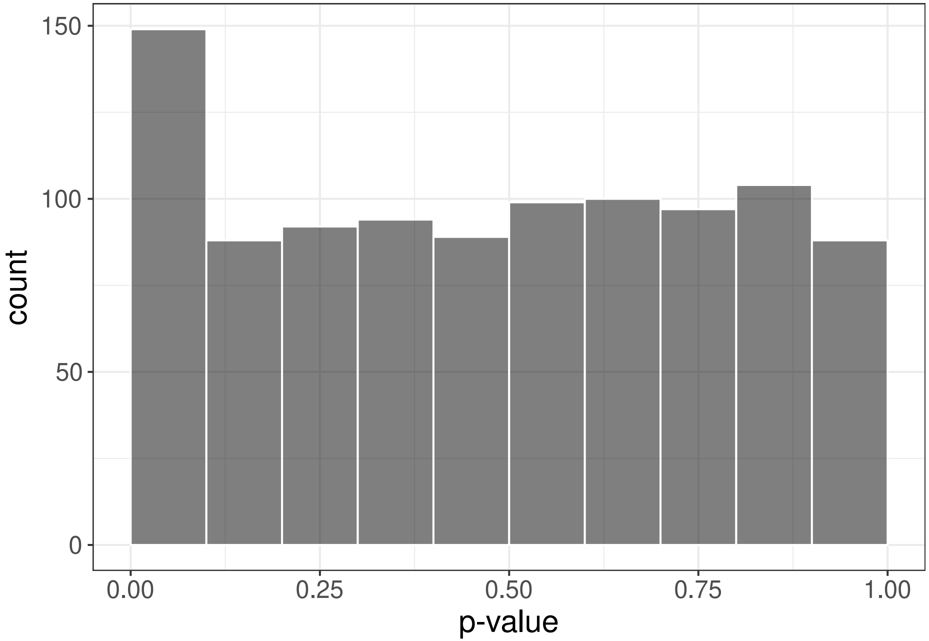
```
##    user  system elapsed
##   1.464   0.112   1.576
```

```
# Linear in both dimensions
# Extremely useful if both dimensions are very large
system.time(svd3 <- big_randomSVD(X, fun.scaling = big_scale(), k = 1
```

```
##    user  system elapsed
##   1.933   0.014   1.948
```

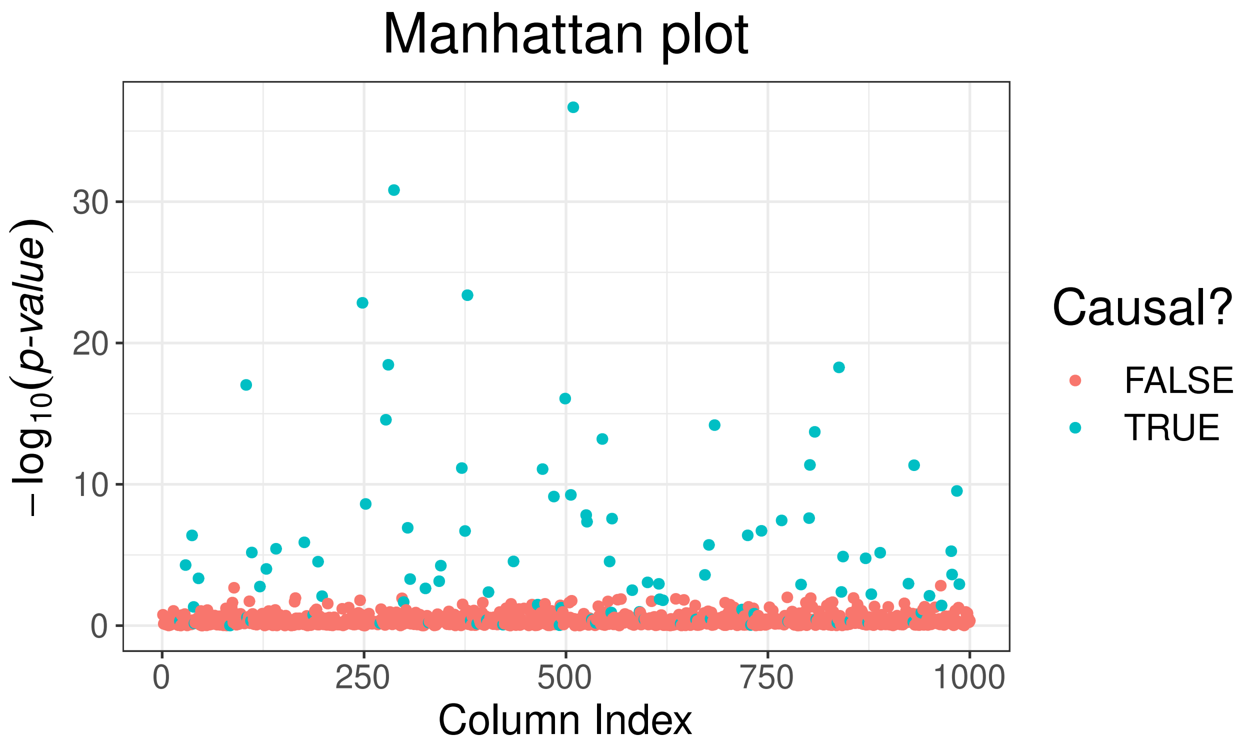
Multiple association

```
M <- 100 # number of causal variables
set <- sample(ncol(X), M)
y <- scale(X[, set]) %*% rnorm(M)
y <- y + rnorm(length(y), sd = 2 * sd(y))

mult_test <- big_univLinReg(X, y, covar.train = svd2$u)
plot(mult_test)
```



Multiple association

```r
library(ggplot2)
plot(mult_test, type = "Manhattan") +
  aes(color = cols_along(X) %in% set) +
  labs(color = "Causal?")
```



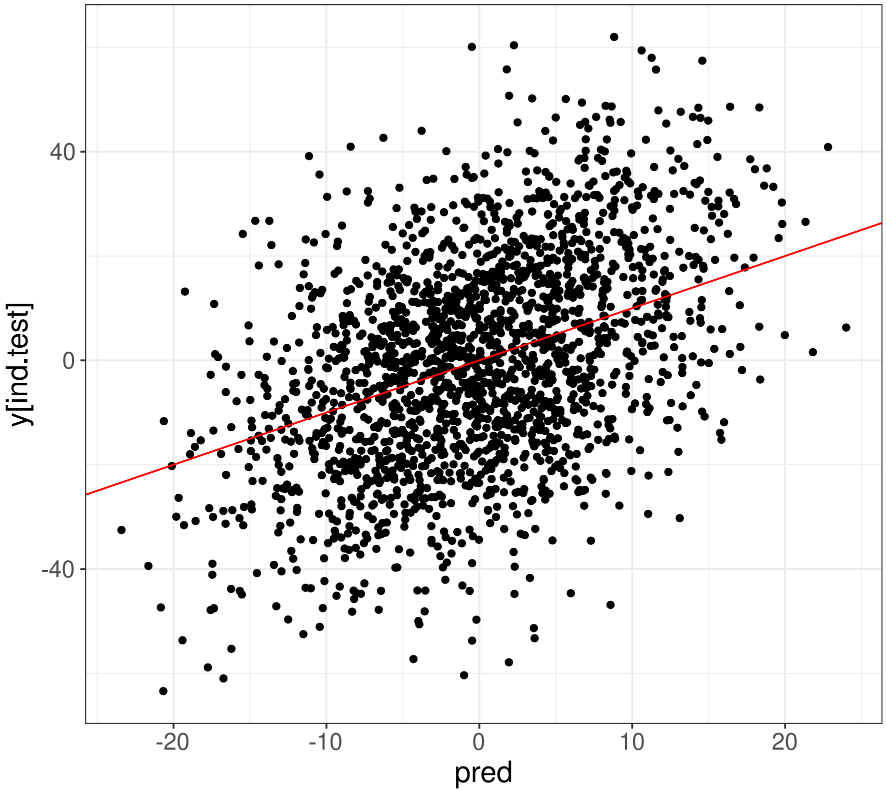
Prediction

```
# Split the indices in train/test sets
ind.train <- sort(sample(nrow(X), size = 0.8 * nrow(X)))
ind.test <- setdiff(rows_along(X), ind.train)

# Train a linear model with elastic-net regularization
# and automatic choice of hyper-parameter lambda
train <- big_spLinReg(X, y[ind.train], ind.train = ind.train,
                      covar.train = svd2$u[ind.train, ],
                      alphas = c(1, 0.1, 0.01))
```

```
# Get predictions for the test set
pred <- predict(train, X = X, ind.row = ind.test,
                covar.row = svd2$u[ind.test, ])
```

Prediction

```
# Plot true value vs prediction
qplot(pred, y[ind.test]) +
  geom_abline(intercept = 0, slope = 1, color = "red") +
  theme_bigstatsr()
```



Toy case:

Compute the sum for each column

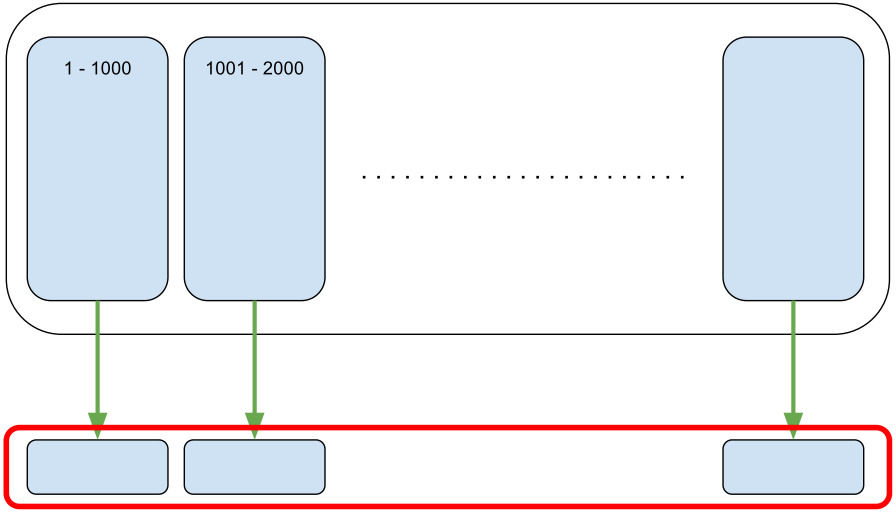
Brute force solution

```r
sums1 <- colSums(X[])  ## /!\ access all the data in memory
```

Do it by blocks

```r
sums2 <- big_apply(X, a.FUN = function(X, ind) colSums(X[, ind]),
                   a.combine = 'c')

all.equal(sums2, sums1)
```

```
## [1] TRUE
```



SPLIT
(e.g. every 1000 columns)

APPLY
(e.g. `colSums()`)

COMBINE
(e.g. `c()`)

Using Rcpp (1/3)

```cpp
// [[Rcpp::depends(bigstatsr, rmio, RcppArmadillo)]]
#include <bigstatsr/BMAcc.h>

// [[Rcpp::export]]
NumericVector bigcolsums(Environment BM) {

  XPtr<FBM> xpBM = BM["address"]; // get the external pointer
  BMAcc<double> macc(xpBM);       // create an accessor to the data

  size_t i, j, n = macc.nrow(), m = macc.ncol();
  NumericVector res(m);  // vector of m zeros

  for (j = 0; j < m; j++)
    for (i = 0; i < n; i++)
      res[j] += macc(i, j);

  return res;
}
```

Using Rcpp (1/3)

```
sums3 <- bigcolsums(X)

all.equal(sums3, sums1)
```

```
## [1] TRUE
```

Using Rcpp (2/3): the bigstatsr way

```
// [[Rcpp::depends(bigstatsr, rmio, RcppArmadillo)]]
#include <bigstatsr/BMAcc.h>

// [[Rcpp::export]]
NumericVector bigcolsums2(Environment BM,
                          const IntegerVector& rowInd,
                          const IntegerVector& colInd) {

  XPtr<FBM> xpBM = BM["address"];
  SubBMAcc<double> macc(xpBM, rowInd - 1, colInd - 1);

  size_t i, j, n = macc.nrow(), m = macc.ncol();
  NumericVector res(m);  // vector of m zeros

  for (j = 0; j < m; j++)
    for (i = 0; i < n; i++)
      res[j] += macc(i, j);

  return res;
}
```

Using Rcpp (2/3): the bigstatsr way

```
sums4 <- bigcolsums2(X, rows_along(mat), cols_along(mat))

all.equal(sums4, sums1)
```

```
## [1] TRUE
```

```
sums5 <- bigcolsums2(X, rows_along(mat), 1:10)

all.equal(sums5, sums1[1:10])
```

```
## [1] TRUE
```

Using Rcpp (3/3): already implemented

```
sums6 <- big_colstats(X)
str(sums6)
```

```
## 'data.frame':    1000 obs. of  2 variables:
##  $ sum: num  184.5 -55.8 77.4 -110.5 -45.1 ...
##  $ var: num  0.997 1.006 0.987 1.005 1.014 ...
```

```
all.equal(sums6$sum, sums1)
```

```
## [1] TRUE
```

Parallelism

Most of the functions are parallelized

```
ind.rep <- rep(cols_along(X), each = 100)  ## size: 100,000
(NCORES <- nb_cores())
```

```
## [1] 2
```

```
system.time(
  mult_test2 <- big_univLinReg(X, y, covar.train = svd2$u,
                               ind.col = ind.rep)
)
```

```
##    user  system elapsed
##   6.186   0.014   6.269
```

```
system.time(
  mult_test3 <- big_univLinReg(X, y, covar.train = svd2$u,
                               ind.col = ind.rep, ncores = NCORES)
)
```

```
##    user  system elapsed
##   0.061   0.054   4.389
```

Parallelize your own functions

```
system.time(
  mult_test4 <- big_parallelize(
    X, p.FUN = function(X, ind, y, covar) {
      bigstatsr::big_univLinReg(X, y, covar.train = covar,
                                ind.col = ind)
    }, p.combine = "rbind", ind = ind.rep,
    ncores = NCORES, y = y, covar = svd2$u)
)
```

```
##    user  system elapsed 
##   0.055   0.057   5.046
```

```
all.equal(mult_test4, mult_test3)
```

```
## [1] TRUE
```

Conclusion

I'm able to run algorithms

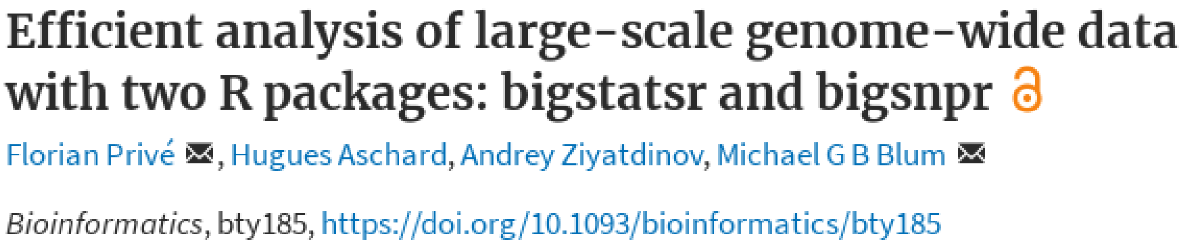
on 100GB of data

in R on my computer

Advantages of using FBM objects

- you can apply algorithms on **data larger than your RAM**,
- you can easily **parallelize** your algorithms because the data on disk is shared,
- you write **more efficient algorithms** (you do less copies and think more about what you're doing),
- you can use **different types of data**, for example, in my field, I'm storing my data with only 1 byte per element (rather than 8 bytes for a standard R matrix). See the documentation of the FBM class for details.

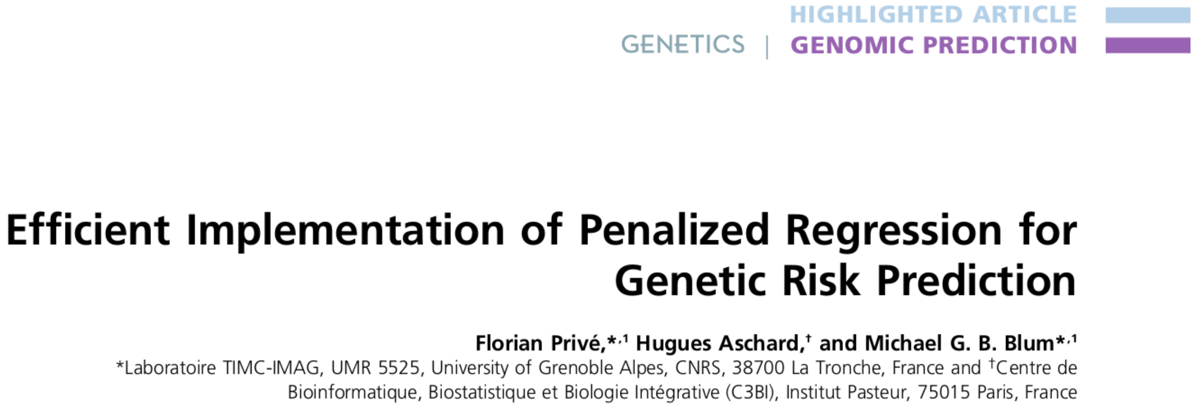
Check publications for details



Efficient analysis of large-scale genome-wide data with two R packages: bigstatsr and bigsnpr

Florian Privé ✉, Hugues Aschard, Andrey Ziyatdinov, Michael G B Blum ✉

Bioinformatics, bty185, https://doi.org/10.1093/bioinformatics/bty185



GENETICS | HIGHLIGHTED ARTICLE
GENOMIC PREDICTION

Efficient Implementation of Penalized Regression for Genetic Risk Prediction

Florian Privé,*[,1] Hugues Aschard,[†] and Michael G. B. Blum*[,1]
*Laboratoire TIMC-IMAG, UMR 5525, University of Grenoble Alpes, CNRS, 38700 La Tronche, France and [†]Centre de Bioinformatique, Biostatistique et Biologie Intégrative (C3BI), Institut Pasteur, 75015 Paris, France

Contributors are welcome!

Make sure to grab an hex sticker

Thanks!

Presentation: https://privefl.github.io/R-presentation/bigstatsr.html

Package's website: https://privefl.github.io/bigstatsr/

DOIs: 10.1093/bioinformatics/bty185
and 10.1534/genetics.119.302019

privefl privefl F. Privé

Slides created via the R package **xaringan**.