Parallel Computing Project

Statistics for Smart Data

Hugo Brehier

8/01/2020

Contents

1 Data generation

The code is situated in datagen.R (function rlogit()).

We generate the independant variables according to two normal distribution: $N(0,1)$ and $N(1,2)$, such that half the variables follow either one of the two distribution. The response variable is then generated through the logit model.

2 Auxiliary functions

The code is situated in aux_fcts_2b.R . These function are the building blocks of the Newton-Rhapson algorithm, which we will see afterwards. My source for the following functions is : [Gormley, 2016].

2.1 Negative Log-likelihood

It is the function loglikl() which computes the log-likelihood $\ell(\theta)$ of the model. In particular, we use its negative, $J(\theta) = -\ell(\theta)$.

2.2 Gradient

It is the function gradient() which computes the gradient of the negative log-likelihood in relation to θ, $\nabla J(\theta)$.

2.3 Hessian

It is the function hessian() which computes the Hessian of the negative log-likelihood in relation to θ, $H_{J(\theta)}$

3 Newton–Raphson algorithm

The algorithm is implemented in the last function of aux_fcts_2b.R (function rhapson_newton()). This algorithm allows us to find a minimum of the negative log-likelihood $J(\theta)$ by the following repeated schema, until convergence:
$\theta_{n+1} = \theta_n - H^{-1}_{J(\theta)} \nabla J(\theta)$
Thus, we indeed make use of the auxiliary functions. See [Bertsimas, 2009] for further reference.

4 Basic functions

These functions are situated in basics.R . They use of the Newton–Raphson algorithm and expand it further : cross-validation, model comparison and model selection. Each is a building block for the next.

4.1 Cross-Validation

It is the function basic.cv(), which uses rhapson_newton().

4.2 Model comparison

It is the function basic.modelcomparison(), which uses basic.cv().

4.3 Model selection

It is the function basic.modelselection(), which uses basic.modelcomparison().

5 Exceptions

Here is a list of some expections created :

- check the number of folds in basic.cv()
- warn of LOOCV in basic.cv()
- check the type entered in basic.nodelcomparison() variable *models*.

6 Code profiling

Newton–Raphson algorithm is the core of further cross-validation and model selection. Thus, this central piece is a priority to profile.

First result is in run1a.html. I detected the transpose operation in the Hessian function was the main consumer of time. Thus, I stored it *outside the loop*. Result in run1b.html

Then I tried to replace the %*% operator with a custom one.

```
library(Rcpp)
library(microbenchmark)
library(inline)
```

There are two ways to incorporate C++ functions.

```
cppFunction('NumericMatrix mmult(const NumericMatrix& m1, const NumericMatrix& m2){
if (m1.ncol() != m2.nrow()) stop ("Incompatible matrix dimensions");
NumericMatrix out(m1.nrow(),m2.ncol());
NumericVector rm1, cm2;
for (size_t i = 0; i < m1.nrow(); ++i) {
    rm1 = m1(i,_);
    for (size_t j = 0; j < m2.ncol(); ++j) {
      cm2 = m2(_,j);
      out(i,j) = std::inner_product(rm1.begin(), rm1.end(), cm2.begin(), 0.);
    }
  }
return out;
}')
```

```
matmut <- cxxfunction(signature(tm="NumericMatrix",
                                tm2="NumericMatrix"),
                      plugin="RcppEigen",
                      body="
NumericMatrix tm22(tm2);
NumericMatrix tmm(tm);
```

```
const Eigen::Map<Eigen::MatrixXd> ttm(as<Eigen::Map<Eigen::MatrixXd> >(tmm));
const Eigen::Map<Eigen::MatrixXd> ttm2(as<Eigen::Map<Eigen::MatrixXd> >(tm22));

Eigen::MatrixXd prod = ttm*ttm2;
return(wrap(prod));
                  ")
```

We can benchmark them.

```
set.seed(123)
M1 <- matrix(sample(1e3),ncol=50)
M2 <- matrix(sample(1e3),nrow=50)
```

```
identical(matmut(M1,M2), M1 %*% M2)
```

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

```
identical(mmult(M1,M2), M1 %*% M2)
```

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

```
res <- microbenchmark(mmult(M1,M2),matmut(M1,M2), M1 %*% M2,times = 100000)
print(res)
```

```
## Unit: microseconds
##            expr    min     lq      mean  median     uq       max neval
##   mmult(M1, M2) 40.943 42.711 49.785839 43.2640 45.302 10094.541 1e+05
##  matmut(M1, M2)  5.759  6.463  9.074971  6.7490  7.256  4672.739 1e+05
##       M1 %*% M2 10.365 10.866 14.700165 11.0835 11.653 68718.150 1e+05
```

matmut is faster. But for vectors, we need to cast them into matrix of length 1. It cancels out . Rather, we impose the beta vector to be a 1D matrix from the beginning. We also store the $x^T x$ outside the loop in the hessian function. We also use chol2inv(chol()) instead of solve() to inverse the hessian. Results are in run3a.html

After this, we vectorize computations in the auxiliary functions, which loop over each observation. In run3a.html, I use sapply. In run4a.html, I use lapply with standart dot product. Unfortunately, it does not fasten the algorithm... In run5a.html, I use cpp_lapply, another inline C++ function to mimic lapply. It doesn't run faster.

Our best candidate is run2a.html , with loops, variables stored outside of loops and standart matrix multiplication.

7 Parallel computing

In run6a.html, I use mclapply in the hessian function. run6b.html should contain results where I use mclapply in the auxiliary functions. Unfortunately, it was slow or an error occured since it did not finish.

8 Consistency of the procedure of model selection

I finally changed the stopping criterion to the number of iterations rather than the norm of the direction used in [Bertsimas, 2009].

run7.R contains the code for the study with the profiled functions (datagen.R, aux_fcts_2b.R and basics.R).

First, I checked that the algorithm got the same MLE as glm. It was the case for 2 covariates, then it was the same results but scaled differently for 4 covariates. (by a factor of 10 for example).

In my trial,both backward and forward selection recover the same model. Moreover, they recovered the covariates with non-zero coefficients in the generation of data.

9 Conclusion

In this project, I implemented stepwise selection of a logit model estimated though MLE and CV. Code profilling has shown that loops can be faster than vectorization. Parallel computing sadly did not prove an improvement over vectorization *and* loops as mclapply in the hessian function was slow. The **Iteratively reweighted least squares** method or the **BFGS method** may prove to be faster. But further code profilling should be considered.

Thank you for your attention, happy new year !

References

[Bertsimas, 2009] Bertsimas, D. (2009). Lecture 19: Line searches and newton's method. https://ocw.mit.edu/courses/sloan-school-of-management/15-093j-optimization-methods-fall-2009/lecture-notes/MIT15_093J_F09_lec19.pdf. [Accessed December 10, 2019].

[Gormley, 2016] Gormley, M. (2016). Logistic regression. https://www.cs.cmu.edu/~mgormley/courses/10701-f16/slides/lecture5.pdf. [Accessed December 10, 2019].