SDS385 Fall '16: Statistical Models For Big Data
Exercises 01 - Linear regression

Matteo Vestrucci

August 29th 2016

A)

The objective function can be rewritten like this:

$$\hat{\beta} = \arg\min_{\beta \in \mathcal{R}^P} (X\beta - Y)^T \frac{W}{2} (X\beta - Y).$$

The function is smooth and convex, being a sum of squares. Thus we can set the first derivative equal to zero to find the minimum:

$$\begin{aligned}
\nabla_\beta \left[(X\beta - Y)^T \frac{W}{2} (X\beta - Y) \right] &= 0 \\
2X^T \frac{W}{2} (X\beta - Y) &= 0 \\
(X^T W X)\beta - X^T W Y &= 0 \\
(X^T W X)\beta &= X^T W Y.
\end{aligned}$$

B)

Inverting a matrix is a very slow and expansive operation in the order of $O(P^3)$. The "inversion method" is also not very stable because it introduces precision problems when using floating point variables, especially when the determinant is close to zero. As long as within the N rows we can find a number of linearly independent ones greater than P and with positive weight assigned, then the matrix resulting from the product $X^T W X$ will be symmetric and positive definite, which compels us to use the Choleski decomposition and the Gaussian elimination, an algorithm that is much faster than the "inversion method".

The algorithm can be summarized as this:

- *Calculate* $A = X^T W X$
- *Calculate* $B = X^T W Y$
- *Apply Choleski decomposition on* A*, keep the upper triangular matrix* C
- *Apply Gaussian elimination on the system* $C^T D = B$*, keep the vector* D
- *Apply Gaussian elimination on the system* $C\beta = D$*, keep the solution* β

The Choleski decomposition is also the fastest but also the most unstable among the other decompositions available. A good middle ground would be the QR decomposition that outputs an orthonormal matrix Q and an upper triangular matrix R. In that case we would work on $W^{\frac{1}{2}}X = QR$ and proceed with the following simplifications:

$$\begin{aligned} X^T W Y &= X^T W X \beta \\ X^T W^{\frac{1}{2}} W^{\frac{1}{2}} Y &= X^T W^{\frac{1}{2}} W^{\frac{1}{2}} X \beta \\ (QR)^T W^{\frac{1}{2}} Y &= (QR)^T Q R \beta \\ R^T Q^T W^{\frac{1}{2}} Y &= R^T Q^T Q R \beta \\ Q^T W^{\frac{1}{2}} Y &= R\beta. \end{aligned}$$

C)

Running the code in appendix, we can obtain the following benchmarks:

silly_data(N=21,P=7) Unit:microseconds

expr	min	lq	mean	median	uq	max	neval
inversion	25.44	26.4	32.61	27.03	34.35	61.48	10
factorization	31.82	33.36	41.39	34.74	38.82	91.56	10

silly_data(N=150,P=50) Unit:microseconds

expr	min	lq	mean	median	uq	max	neval
inversion	409.76	411.72	416.63	412.99	416.85	445.64	10
factorization	77.9	79.63	90.17	80.19	85.52	170.83	10

silly_data(N=900,P=300) Unit:milliseconds

expr	min	lq	mean	median	uq	max	neval
inversion	71.45	71.59	71.79	71.64	72.01	72.42	10
factorization	6.39	6.41	6.45	6.45	6.48	6.53	10

silly_data(N=3000,P=1000) Unit:milliseconds

expr	min	lq	mean	median	uq	max	neval
inversion	2650.49	2654.23	2659.47	2656.83	2666.07	2673.65	10
factorization	227.22	227.78	228.13	228.02	228.56	229.47	10

D)

Running the code in appendix, we can obtain the following benchmarks:

sparse_data(N=900,P=300,prop=0.05) Unit:microseconds

expr	min	lq	mean	median	uq	max	neval
inversion_m	72864.47	72937.55	73102.84	72959.86	73162.11	73717.14	10
factorization_m	8035.45	8058.48	8162.63	8079.58	8180.62	8700.15	10
sparse_m	758.47	839.55	865.19	852.88	873.94	890.28	10

sparse_data(N=3000,P=1000,prop=0.05) Unit:milliseconds

expr	min	lq	mean	median	uq	max	neval
inversion_m	2675.76	2683.88	2696.26	2686.89	2698.53	2775.58	10
factorization_m	245.69	247.65	248.43	247.85	249.18	256.64	10
sparse_m	8.04	8.25	8.80	8.53	8.81	9.62	10

sparse_data(N=3000,P=1000,prop=0.01) Unit:milliseconds

expr	min	lq	mean	median	uq	max	neval
inversion_m	2664.54	2669.19	2673.19	2671.9	2673.31	2688.53	10
factorization_m	235.53	235.25	236.04	236.15	236.78	237.61	10
sparse_m	7.18	7.20	7.29	7.23	7.27	7.91	10

CODE)

```
library(Matrix)
library(microbenchmark)

inversion_m<-function(matA,matB){
  #matA*beta=matB
  beta<-solve.default(matA)%*%matB
  return(beta)
}
factorization_m<-function(matA,matB){
  #matA*beta=matB
  #matL'*matL*beta=matB
  #matL'*matC=matB
  #matL*beta=matC
  matL<-chol.default(matA)
  matC<-forwardsolve(t(matL),matB)
  beta<-backsolve(matL,matC)
  return(beta)
}
silly_data<-function(N,P){
  X<-matrix(rnorm(N*P,sd=10), nrow=N, ncol=P)
  truebeta<-rnorm(P,sd=10)
  W<-diag(sample(c(1,2),N,replace=T))
  error<-rnorm(N)
  Y = X%*%truebeta+error
```

```
  matA<-t(X)%*%W%*%X
  matB<-t(X)%*%W%*%Y
  return(list(matA=matA,matB=matB))
}
data<-silly_data(N=21,P=7)
matA<-data$matA;matB<-data$matB
p7<-microbenchmark(inversion_m(matA,matB),factorization_m(matA,matB),times=10);p7
data<-silly_data(N=150,P=50)
matA<-data$matA;matB<-data$matB
p50<-microbenchmark(inversion_m(matA,matB),factorization_m(matA,matB),times=10);p50
data<-silly_data(N=900,P=300)
matA<-data$matA;matB<-data$matB
p300<-microbenchmark(inversion_m(matA,matB),factorization_m(matA,matB),times=10);p300
data<-silly_data(N=3000,P=1000)
matA<-data$matA;matB<-data$matB
p1000<-microbenchmark(inversion_m(matA,matB),factorization_m(matA,matB),times=10);p1000

sparse_data<-function(N,P,prop=0.05){
  X<-matrix(rnorm(N*P,sd=10), nrow=N, ncol=P)
  mask = matrix(rbinom(N*P,1,prop), nrow=N)
  X = mask*X
  truebeta<-rnorm(P,sd=10)
  W<-diag(sample(c(1,2),N,replace=T))
  error<-rnorm(N)
  Y = X%*%truebeta+error
  matA<-Matrix(t(X)%*%W%*%X,sparse=T)
  matB<-Matrix(t(X)%*%W%*%Y,sparse=F)
  return(list(matA=matA,matB=matB))
}
sparse_m<-function(matA,matB){
  #matA*beta=matB
  beta<-solve(matA,matB)
  return(beta)
}
data<-sparse_data(N=900,P=300,prop=0.05)
matA<-data$matA;matB<-data$matB
p300_0.05<-microbenchmark(inversion_m(matA,matB),factorization_m(matA,matB),
                       sparse_m(matA,matB),times=10);p300_0.05
data<-sparse_data(N=3000,P=1000,prop=0.05)
matA<-data$matA;matB<-data$matB
p1000_0.05<-microbenchmark(inversion_m(matA,matB),factorization_m(matA,matB),
                        sparse_m(matA,matB),times=10);p1000_0.05
data<-sparse_data(N=3000,P=1000,prop=0.01)
matA<-data$matA;matB<-data$matB
p1000_0.01<-microbenchmark(inversion_m(matA,matB),factorization_m(matA,matB),
                        sparse_m(matA,matB),times=10);p1000_0.01
```