Hard

Luofeng Liao

March 13, 2018

Hard

In [[https://web.stanford.edu/~hastie/glmnet/glmnet_alpha.html][here]] the author explains the detail of the loss function. The loss function of logistic regression is

$$-\Big\{\frac{1}{N}\sum\big[y_i(\beta_0+x_i^\top\beta)-\log(1+e^{(\beta_0+x_i^\top\beta)})\big]\Big\}+\lambda\Big\{\frac{(1-\alpha)}{2}\|\beta\|_2^2+\alpha\|\beta\|_1\Big\}.$$

When $\alpha = 1$ we get what is requied.

```
library(Rcpp)
library(RcppArmadillo)
library(inline)
library(ElemStatLearn)
library(testthat)
library(rbenchmark)
```

```
# Rcpp function
sourceCpp('C.cpp')
# The C code is as follow
```

```
#include <RcppArmadillo.h>
// [[Rcpp::depends(RcppArmadillo)]]
using namespace Rcpp;
using namespace std;
using namespace arma;
// [[Rcpp::export]]
SEXP cpp_l(Mat<double> X,
           vec y,
           double beta_0,
           vec beta,
           double alpha,
           double lambda) {

  double n = X.n_rows;
  vec Xb = beta_0 + X*beta;
  double l = -accu(y%Xb - log(1+exp(Xb)))/n;
  double Omega = lambda *accu((1-alpha)/2*beta%beta + alpha*abs(beta));
//cout<<l<<endl;
//cout<<Omega<<endl;
  return wrap(l+Omega);
}
```

```
# R implementation of the logistic loss function
r_l <- function(X,y,beta,beta_0,alpha,lambda){
  n <- dim(X)[1]
  Xb <- X%*%beta + beta_0
  l <- -sum(y*Xb - log(1+exp(Xb)))/n
  Omega <- lambda*sum((1-alpha)/2*beta*beta + alpha*abs(beta))
  return(l+Omega)
}
```

```
# Start calculation
n <-10
p <- 10
alpha <- 1;
lambda <- rnorm(1);
y <- rnorm(n)
X <- matrix(rnorm(n*p),n,p)
beta_0 <- rnorm(1)
beta <- rnorm(p)
a <- cpp_l(X,y,beta=beta,beta_0=beta_0,alpha,lambda)
b <- r_l(X,y,beta=beta,beta_0=beta_0,alpha,lambda)
all.equal(a,b)
```

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

I also want to see how efficient C++ implemtation is.

```
result <- benchmark(replications=rep(10000, 10),
                  cpp_l(X,y,beta=beta,beta_0=beta_0,alpha,lambda),
                    r_l(X,y,beta=beta,beta_0=beta_0,alpha,lambda),
                    columns=c('test', 'elapsed'))
print(result)
```

```
##                                                         test elapsed
## 1  cpp_l(X, y, beta = beta, beta_0 = beta_0, alpha, lambda)    0.08
## 3  cpp_l(X, y, beta = beta, beta_0 = beta_0, alpha, lambda)    0.13
## 5  cpp_l(X, y, beta = beta, beta_0 = beta_0, alpha, lambda)    0.06
## 7  cpp_l(X, y, beta = beta, beta_0 = beta_0, alpha, lambda)    0.08
## 9  cpp_l(X, y, beta = beta, beta_0 = beta_0, alpha, lambda)    0.08
## 11 cpp_l(X, y, beta = beta, beta_0 = beta_0, alpha, lambda)    0.07
## 13 cpp_l(X, y, beta = beta, beta_0 = beta_0, alpha, lambda)    0.07
## 15 cpp_l(X, y, beta = beta, beta_0 = beta_0, alpha, lambda)    0.08
## 17 cpp_l(X, y, beta = beta, beta_0 = beta_0, alpha, lambda)    0.08
## 19 cpp_l(X, y, beta = beta, beta_0 = beta_0, alpha, lambda)    0.10
## 2    r_l(X, y, beta = beta, beta_0 = beta_0, alpha, lambda)    0.08
## 4    r_l(X, y, beta = beta, beta_0 = beta_0, alpha, lambda)    0.08
## 6    r_l(X, y, beta = beta, beta_0 = beta_0, alpha, lambda)    0.08
## 8    r_l(X, y, beta = beta, beta_0 = beta_0, alpha, lambda)    0.09
## 10   r_l(X, y, beta = beta, beta_0 = beta_0, alpha, lambda)    0.08
## 12   r_l(X, y, beta = beta, beta_0 = beta_0, alpha, lambda)    0.08
## 14   r_l(X, y, beta = beta, beta_0 = beta_0, alpha, lambda)    0.09
## 16   r_l(X, y, beta = beta, beta_0 = beta_0, alpha, lambda)    0.08
## 18   r_l(X, y, beta = beta, beta_0 = beta_0, alpha, lambda)    0.08
## 20   r_l(X, y, beta = beta, beta_0 = beta_0, alpha, lambda)    0.12
```

Not very significant actually. Probably vectorization in R is already highly optimized. I should dive into the source code and then update this report.