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))
```

```r
  print(l)
  print(Omega)
  return(l+Omega)
}
```

```r
# Start calculation
n <- length(spam$spam)
alpha <- rnorm(1)
lambda <- rnorm(1)
y <- c(1,2)
X <- matrix(c(4,5,6,7),2,2)
beta_0 <- 1
beta <- c(1,2)
all.equal(cpp_l(X,y,beta=beta,beta_0=beta_0,alpha,lambda),
          r_l(X,y,beta=beta,beta_0=beta_0,alpha,lambda))
```

```
## [1] -10
## [1] -3.513331
```

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

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

```r
print(result)
```

```
##                                                        test elapsed
## 1 cpp_l(X, y, beta = beta, beta_0 = beta_0, alpha, lambda)    0.02
## 3 cpp_l(X, y, beta = beta, beta_0 = beta_0, alpha, lambda)    0.00
## 5 cpp_l(X, y, beta = beta, beta_0 = beta_0, alpha, lambda)    0.00
## 2   r_l(X, y, beta = beta, beta_0 = beta_0, alpha, lambda)    0.14
## 4   r_l(X, y, beta = beta, beta_0 = beta_0, alpha, lambda)    0.25
## 6   r_l(X, y, beta = beta, beta_0 = beta_0, alpha, lambda)    0.41
##   replications
## 1         2000
## 3         2000
## 5         2000
## 2         2000
## 4         2000
## 6         2000
```