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. Note that here $y_i\in\{0,1\}$.

```
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>
#include <assert.h>
// [[Rcpp::depends(RcppArmadillo)]]
using namespace Rcpp;
using namespace std;
using namespace arma;

int myassert(bool flag,char * info)
  {
  if (flag == 0){
    throw std::invalid_argument(info);
  }
  return 0;
};

// [[Rcpp::export]]
SEXP cpp_l(Mat<double> X,
              vec y,
              double beta_0,
              vec beta,
              double alpha,
              double lambda) {

  double n = X.n_rows;
  double p = X.n_cols;
  double n_y = y.n_rows;
  double n_beta = beta.n_rows;
  // valid dimension
```

```
  myassert(n==n_y,"# of obs in X and y do not match!\n");
  myassert(p==n_beta,"# of features in X and beta do not match!\n")

  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));

  return wrap(l+Omega);
}
```

```
# R implementation of the logistic loss function
# here y \in {0,1}
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 <- 1;
y <- c(1,2)
X <- matrix(c(1,2,3,4),2,2)
beta_0 <- 2
beta <- c(2,3)
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, 3),
                    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)    0.15
## 3 cpp_l(X, y, beta = beta, beta_0 = beta_0, alpha, lambda)    0.19
## 5 cpp_l(X, y, beta = beta, beta_0 = beta_0, alpha, lambda)    0.21
## 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)    0.06
```

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