Medium-hard

Luofeng Liao

Introduction

`glmnet(family="binomial", alpha=1)` uses cyclic coordinate descent to solve L1-regularized logistic regression. Note that the default parameter setting implies data are standardized. The algorithm uses a quadratic approximation to the log-likelihood. Loss function 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 $y_i\in\{0,1\}$.

`sklearn` provides different gradient descent algorithm to solve L1-regularized logistic regresison, including `liblinear` and `saga`. Loss function is

$$C\sum\log\exp(-y_i(x_i^\top w+c)+1)+\|w\|_1.$$

Here $y_i\in\{-1,1\}$.

Solution

In this task I use the `spam` dataset. First, to compare the two different algorithms, we have to ensure the hyperparameter settings are identical. In `glmnet` we have hyperparameters α and λ, in `sklearn` we have C. By comparing the objective functions we see the follow relation should hold if we want two algorithms to optimize the same thing,

$$N\alpha\lambda=\frac{1}{C}.$$

Other trivial settings include whether to standardize the data set, whether to include an intercept term, and stopping criteria.

```
library(glmnet)
library(microbenchmark)
library(plotly)
library(RcppCNPy)

# compute 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)
}
```

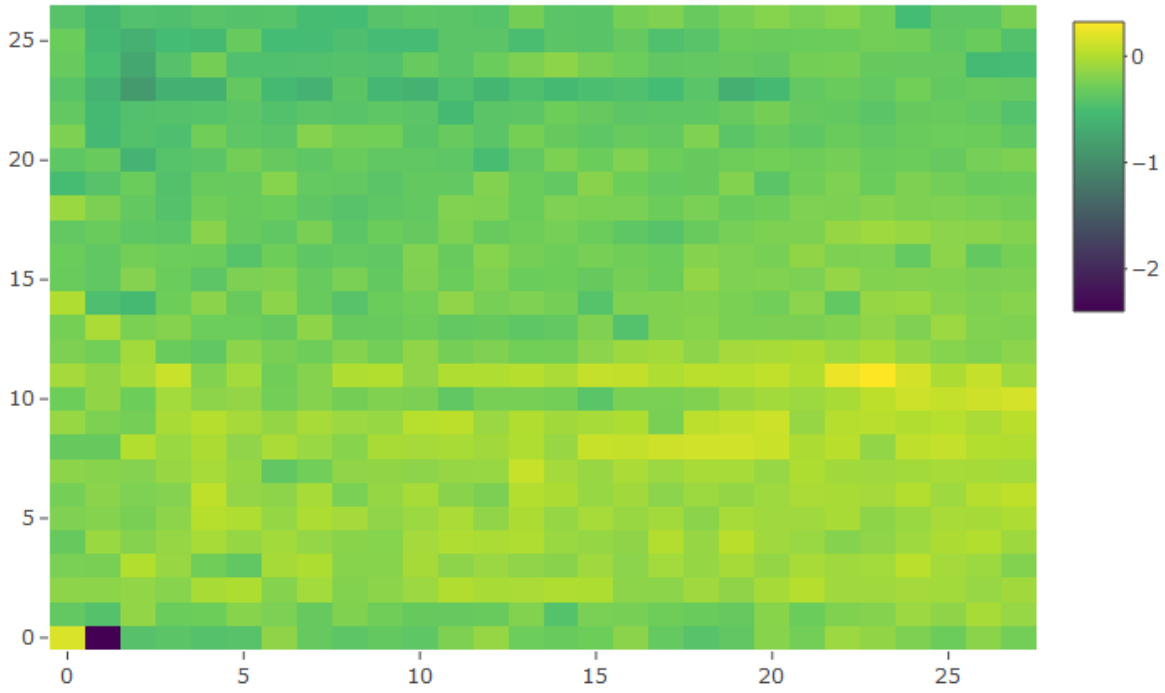
```
# load shuffeled data set spam
tag <- npyLoad('y.npy')
feat_mat <- npyLoad('X.npy')

# para that divide the data set
p_tot <- 57 # Each email is represented by 57 features
n_tot <- length(spam$spam)
```

```r
p_itv <- 2 # Interval for p
n_itv <- as.integer(n_tot/p_tot*p_itv)
p_index <- seq(3,p_tot,p_itv)
n_index <- seq(100,n_tot,n_itv)

# hyperpara of the model
lambda <- 1/n_tot
maxit <- 1e+5
tol <- 1e-04
alpha <- 1 # only L1 regularization
```

```r
# matrix storing the timing
glmnet_timing <- matrix(0,nrow=length(p_index),ncol=length(n_index))
glmnet_cost <- matrix(0,nrow=length(p_index),ncol=length(n_index))

# plug in the model data sets of different size
for(cur_p in 1:length(p_index)){
  for(cur_n in 1:length(n_index)){

    # extract data set
    train_class <- tag[1:n_index[cur_n]]
    train_feat <- feat_mat[1:n_index[cur_n],1:p_index[cur_p]]

    # Obtain coefficients and find loss function value
    mod <- glmnet(train_feat,
                              train_class,
                              lambda=lambda,
                              thresh=tol,
                              family="binomial",
                              alpha=alpha,
                              standardize=F)
    co <- coef(mod)
    glmnet_cost[cur_p,cur_n] <- r_l(train_feat,
                                    train_class,
                                    co[2:length(co)],
                                    co[1],
                                    alpha,
                                    lambda)

    # repeat algo for each dataset 10 times
    tim<-microbenchmark(glmnet(train_feat,
                              train_class,
                              lambda=lambda,
                              thresh=tol,
                              family="binomial",
                              alpha=alpha,
                              standardize=F),
                        times=10)

    glmnet_timing[cur_p,cur_n] <- mean(tim$time)

  }
}
```

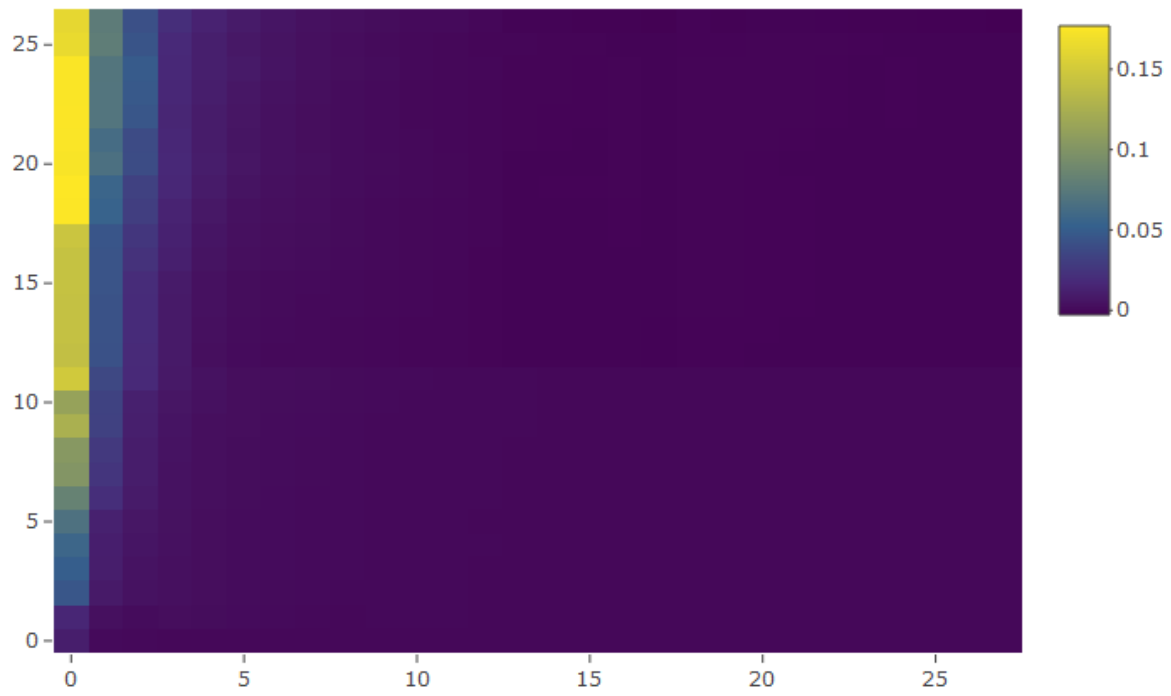
```
# loading pre-computed results
sklearn_cost <- npyLoad('los_python.npy')
glmnet_cost <- npyLoad('glmnet_cost')

sklearn_timing <- npyLoad('tim_python.npy')
glmnet_timing <- npyLoad('glmnet_timing')
```

```
# compare timing
plot_ly(z = (log10(sklearn_timing) - log10(glmnet_timing)) ,type='heatmap')
```



We can see `sklearn` outperforms `glmnet` of 1-2 order of magnitude, especially in larger scale data set. Note that darker areas prefer `sklearn`.

```
# compare accuracy
plot_ly(z = sklearn_cost - glmnet_cost,type='heatmap')
```



Here we can see as for accuracy (yellow menas favoring `glmnet`), we can see both algorithms reach very close accuracy when number of observations and features are both large. Gaps occur when we have few features (i.e., on the left side of diagram clusters of yellow appear).

The training time and objective function value are pre-computed in Python. Code is as follow.

```
from sklearn import linear_model
import numpy as np
from timeit import default_timer as timer

# load data se
feat_mat = np.load('X.npy')
tag = np.load('y.npy')
p_tot = 57 # Each email is represented by 57 features
n_tot = len(tag)
p_itv = 2
n_itv = int(n_tot/p_tot*p_itv)
rep = 3

# cost function
def cost(X,y,beta,beta_0,alpha,llambda):
    n = X.shape[0]
    Xb = np.reshape(X.dot(beta) + beta_0, (-1,))
    l = -np.sum(np.multiply(y,Xb) - np.log(1+np.exp(Xb)))/n
    Omega = llambda*np.sum((1-alpha)/2* np.multiply(beta,beta) + alpha*abs(beta))
    return(l+Omega)

# hyperparameter
llambda = 1/n_tot
alpha = 1
maxit = 1e+3
```

```
C = 1 / (n_tot * llambda *alpha)
tol = 1e-04

# spliting data set
p_index = range(3,p_tot,p_itv)
n_index = range(100,n_tot,n_itv)

# matrices that stores results
timing_res = np.zeros((len(p_index),len(n_index)))
cost_res = np.zeros((len(p_index),len(n_index)))

# create logistic regression model, same setting as it is in R
LR = linear_model.LogisticRegression(penalty='l1',
                                     tol = tol,
                                     solver='liblinear',
                                     C=C,
                                     max_iter=maxit,
                                     n_jobs = 8
                                     )

# train the model with data set of different sizes
for p_idx, cur_p in enumerate(p_index):
    for n_idx, cur_n in enumerate(n_index):
        print(cur_n)
        train_class = tag[0:cur_n]
        train_feat = feat_mat[0:cur_n,0:cur_p]

        start = timer()
        LR.fit(train_feat,train_class)
        end = timer()

        beta = LR.coef_
        beta_0 = LR.intercept_

        timing_res[p_idx,n_idx] = (end-start)*1e+9 # from second to nanosecond
        cost_res[p_idx,n_idx] = cost(train_feat,
                                     train_class,
                                     beta.transpose(),
                                     beta_0,
                                     alpha,
                                     llambda)
        print(timing_res[p_idx,n_idx])

# save results
np.save('tim_python.npy',timing_res)
np.save('los_python.npy',cost_res)
```