Large Scale Logistic Regression in High Dimension

Yixiang Yuan

Introduction

The package provides functions for large scale, high-dimensional logistic regression analysis both with the entire data set or with data split across multiple cores for faster computation.

The vignette will go through installation of the package, creating simulation data sets for the functions, and using the functions with and without splitting.

Installation

The following code chunk will install the package through devtools. It will first try to install devtools, then install the package located on Github.

```
# Optional: install devtools
# install.packages("devtools")

# Install the package from github
devtools::install_github("yixiang-yuan/logisticDCtools")
```

Then, check if the package is installed by loading it:

```
library(logisticDCtools)
```

Methods

In this section, we will introduce the two methods included in the package: `full.logistic` and `dc.logistic`.

`full.logistic`

The function runs high-dimension logistic regression without splitting the data set. It allows for penalty functions including LASSO, SCAD and MCP in estimation to tackle with the high-dimensionality, and also allows for debiasing operations in the final estimate. The function will return the estimated active set and the final estimate.

Parameters

- x: the design matrix
- y: the response vector
- penalty: penalty method. Supported methods include 'lasso', 'SCAD' and 'MCP'
- debias: boolean parameter indicating whether to desparsify the estimate
- ridge_lambda: lambda parameter used in the regularized inverse of the debiasing step

Output The function returns a list with the active set, the final estimate and running time of the function.

```
p <- 1000
n <- 1000
s <- 30
```

```r
X <- matrix(rnorm(n * p), nrow = n, ncol = p)
beta <- sample(c(rep(10 / sqrt(s), s), rep(0, p - s)))
beta.active <- ifelse(beta != 0, 1, 0)
y <- ifelse(1 / (1 + exp(-X %*% beta)) > 0.5, 1, 0)

result <- full.logistic(X, y, penalty = "lasso")
sens <- sum(as.numeric((result$active == 1) & (beta.active == 1))) / s
spec <- sum(as.numeric((result$active == 0) & (beta.active == 0))) / (p - s)
bias <- mean((beta - result$final)^2)

data.frame(
  sensitivity = sens,
  specificity = spec,
  bias = bias,
  time = result$time
)
```

```
##   sensitivity specificity       bias time
## 1           1    0.814433 0.03842268 2.49
```

dc.logistic

This function runs large scale high-dimension logistic regression with the divide-and-conquer approach. It still allows for penalty functions including LASSO, SCAD and MCP in estimation, and also allows for debiasing operations for each local estimate. The active set is combined using majority voting, while the final estimate is determined from the chosen aggregation method, which includes 'average', 'sparse', and 'weighted'.

Parameters

- X: the design matrix
- y: the response vector
- k: the number of nodes to randomly and equally split the data
- tau: the thresholding parameter proportion to k deciding the number of votes needed to be in the final active set
- penalty: penalty method. Supported methods include 'lasso', 'SCAD' and 'MCP'
- aggregate.method: aggregation method. Supported methods include 'average', 'sparse' and 'weighted'
- debias: boolean parameter indicating whether to desparsify the estimate
- ridge_lambda: lambda parameter used in the regularized inverse of the debiasing step

Output The function returns a list with the active set, all the local and the combined final estimators, and the running time.

```r
p <- 1000
n <- 1000
s <- 30
k <- 5
tau <- 0.2
X <- matrix(rnorm(n * p), nrow = n, ncol = p)
beta <- sample(c(rep(10 / sqrt(s), s), rep(0, p - s)))
beta.active <- ifelse(beta != 0, 1, 0)
y <- ifelse(1 / (1 + exp(-X %*% beta)) > 0.5, 1, 0)

result <- dc.logistic(X, y, k, tau, penalty = "lasso")
sens <- sum(as.numeric((result$active == 1) & (beta.active == 1))) / s
spec <- sum(as.numeric((result$active == 0) & (beta.active == 0))) / (p - s)
```

```r
bias <- mean((beta - result$final)^2)

data.frame(
  sensitivity = sens,
  specificity = spec,
  bias = bias,
  time = result$time
)
```

```
##   sensitivity specificity       bias time
## 1           1   0.7804124 0.09048778  0.8
```

Conclusions

We can see a significant improvement on running time with the divide-and-conquer algorithm. Meanwhile, the variable selection abilities of the divide-and-conquer algorithm does not underperform the results from evaluating the entire data set. This short demonstration showed the advantages of using divide-and-conquer for very large data sets in practice.