vignette

Derek Hoare

5/24/2021

Homework 2 Vignette

The R package *HoareHW2* implements an iterative ordinary least squares solver, a weighted leveraging regression solver, and an elastic net algorithm.

To download and install the package, use `devtools`:

```
library(devtools)
devtools::install_github("hoared/STSCI_6520_HW2")
```

You can subsequently load the package with the usual R commands:

```
library(HoareHW2)
```

We now go over how to use the three functions described above.

1. solve_ols

The solve_ols function computes the Ordinary Least Squares solution using two different coordinate descent methods. The Gauss-Seidel method, and the Jacobi Method. The Jacobi Method has a parallel implementation which can be used if the user specifies a number of cores greater than 1.

```
alpha = 3
v = as.vector(c(0,1) %*% t(rep(1,50))) # Construct v
A = diag(rep(alpha, 100))
for(i in 1:(100-1)){
  A[i,i+1] <- A[i+1,i] <- -1
}

b = A %*% v

# Gauss-Seidel
HoareHW2::solve_ols(A, b , method = "Gauss", numcores = 1, niter = 1000)
```

```
##   [1] 0 1 0 1 0 1 0 1 0 1 0 1 0 1 0 1 0 1 0 1 0 1 0 1 0 1 0 1 0 1 0 1 0 1 0 1 0
##  [38] 1 0 1 0 1 0 1 0 1 0 1 0 1 0 1 0 1 0 1 0 1 0 1 0 1 0 1 0 1 0 1 0 1 0 1 0 1
##  [75] 0 1 0 1 0 1 0 1 0 1 0 1 0 1 0 1 0 1 0 1 0 1 0 1 0 1
```

```
# Sequential Jacobi
HoareHW2::solve_ols(A, b , method = "Jacobi", numcores = 1, niter = 1000)
```

```
##   [1] 0 1 0 1 0 1 0 1 0 1 0 1 0 1 0 1 0 1 0 1 0 1 0 1 0 1 0 1 0 1 0 1 0 1 0 1 0
##  [38] 1 0 1 0 1 0 1 0 1 0 1 0 1 0 1 0 1 0 1 0 1 0 1 0 1 0 1 0 1 0 1 0 1 0 1 0 1
##  [75] 0 1 0 1 0 1 0 1 0 1 0 1 0 1 0 1 0 1 0 1 0 1 0 1 0 1
```

```r
# Parallel Jacobi
HoareHW2::solve_ols(A, b , method = "Jacobi", numcores = 2, niter = 1000)
```

```
##   [1] 0 1 0 1 0 1 0 1 0 1 0 1 0 1 0 1 0 1 0 1 0 1 0 1 0 1 0 1 0 1 0 1 0 1 0 1 0
##  [38] 1 0 1 0 1 0 1 0 1 0 1 0 1 0 1 0 1 0 1 0 1 0 1 0 1 0 1 0 1 0 1 0 1 0 1 0 1
##  [75] 0 1 0 1 0 1 0 1 0 1 0 1 0 1 0 1 0 1 0 1 0 1 0 1 0 1
```

2. algo_leverage

We can also get least squares estimates using subsampling. The \texttt{algo_leverage} function allows the user to do this for both uniform subsampling and leverage-weighted subsampling. The leverage-weighted subsampling has better statistical properties and is recommended.

```r
# Univariate Case
n<- 500
X = rt(n, 6)
Y = -X + rnorm(n, mean = 0, sd = 1)

algo_leverage(X, Y, r = 50, method = "uniform")
```

```
##   X[idx, ]
## -0.8525174
```

```r
algo_leverage(X, Y, r = 50, method = "leverage")
```

```
##  X[idx, ]
## -1.067108
```

```r
# Multivariate Case

X = cbind(rt(n, 6), rt(n, 6))
Y = X %*% c(1,1)+ rnorm(n, mean = 0, sd = 1)

algo_leverage(X, Y, r = 50, method = "uniform")
```

```
## X[idx, ]1 X[idx, ]2
## 0.9950187 1.1298648
```

```r
algo_leverage(X, Y, r = 50, method = "leverage")
```

```
## X[idx, ]1 X[idx, ]2
## 1.0678753 0.9672587
```

3. elnet_coord

Finally we implement the elastic net algorithm using coordinate descent. This is a penalized least squares regression function that has both $L1$ and $L2$ penalties. α controls the relative weighting of these penalties. A value of $\alpha = 1$ results in a lasso (L1) penalty, and a value of $\alpha = 0$ results in a ridge regression (L2) penalty. λ is an overall weight for the penalty.

```r
beta = c(2,0,-2,0,1,0,-2,0,0,0,0,0,0,0,0,0,0,0,0,0)
p <- 20 #Number of predictors is fixed

# Covariance Matrix
Sigma <- diag(rep(1,20))
Sigma[1,2] <- Sigma[2,1] <- 0.8
```

```
Sigma[5,6] <- Sigma[6,5] <- 0.8

# A sequence of lambdas
lambdas <- seq(0.01, 3, length.out = 25)

# Set n, alpha, and lambdas
n <- 20
alpha <- 0.5
lambdas <- seq(0.01, 3, length.out = 25)


# Generate Data
X <-MASS::mvrnorm(n, rep(0,20), Sigma, tol = 1e-6, empirical = FALSE, EISPACK = FALSE)
Y <- X %*% beta + rnorm(n)

beta_hat = elnet_coord(X, Y, lambdas, alpha)

matplot(log(lambdas), t(beta_hat$betas), type = "l", main = paste("Elastic Net, alpha=", alpha, ", n=",n
```