Best orthogonalized subset selection (BOSS)

Sen Tian

2019-10-17

Installation

We maintain a Github page for package and keep the most updated version there. To install, simply run the following command in the console:

```r
library(devtools)
install_github(repo="sentian/boss", subdir="r-package")
```

A stable version can be installed from CRAN using

```r
install.packages(repo="boss", repos = "http://cran.us.r-project.org")
```

Introduction

BOSS is a least squares based subset selection method. It briefly takes the following steps:

- order the predictors based on their partial correlations with the response,
- perform best subset regression upon the orthogonal basis of the ordered predictors,
- transform the coefficients back to the original space,
- choose the optimal solution using the selection rule AICc-hdf.

The hdf is a heuristic degrees of freedom for BOSS that can be plugged into a selection rule such as AICc-hdf, which can further be used as a selection rule for BOSS. AICc-hdf is defined as

$$\text{AICc-hdf} = n\log\left(\frac{\text{RSS}}{n}\right) + n\frac{n+\text{hdf}}{n-hdf-2},$$

More details can be referred to our paper

Tian, Hurvich and Simonoff (2019), On the use of information criterion in least squares based subset selection problems.

This vignette is structured as follows. We start by simulating a dataset. We then introduces the components, functionalities and basic usage of the package. It is followed by a discussion between BOSS and forward stepwise regression (FS). Finally, we study real data examples and compare BOSS with some popular regularization methods.

A small simulated dataset

The model generating mechanism is $y = X\beta + \epsilon$. We consider a sparse model where only a few predictors matter, and a high signal-to-noise ratio. The detailed parameters are given as:

```r
n = 200 # number of observations
p = 14 # number of predictors
p0 = 6 # number of active predictors (beta_j=0)
rho = 0.9 # correlation between predictors
```

```
nrep = 1000 # number of replications of y to be generated
SNR = 7 # signal-to-noise ratio
```

We make the predictors with $\beta_j \neq 0$ pairwise correlated with opposite effects. Replications of the response y are generated by fixing X. We assume the columns of X and y have 0 mean, so we can exclude the intercept term from model fitting.

```
library(MASS)
# function to generate the data
# columns of X have mean 0 and norm 1, y has mean 0
simu.data <- function(n, p, p0, rho, nrep, SNR){
  # true beta
  beta = rep(0,p)
  beta = c(rep(c(1,-1),p0/2),rep(0,p-p0))
  names(beta) = paste0('X', seq(1,p))
  # covariance matrix
  covmatrix  = matrix(0,nrow=p,ncol=p)
  diag(covmatrix) = 1
  for(i in 1:(p0/2)){
    covmatrix[2*i-1,2*i] = covmatrix[2*i,2*i-1] = rho
  }
  # generate the predictors given the correlation structure
  set.seed(65)
  x = mvrnorm(n,mu=rep(0,p),Sigma=covmatrix)
  x = scale(x,center=TRUE,scale=FALSE)
  colnorm = apply(x,2,function(m){sqrt(sum(m^2))})
  x = scale(x,center=FALSE,scale=colnorm) # standardization
  # sigma calculated based on SNR
  sd = sqrt(t(beta/colnorm)%*%covmatrix%*%(beta/colnorm) / SNR)
  mu = x%*%beta
  # generate replications of y by fixing X
  y = matrix(rep(mu,each=nrep),ncol=nrep,byrow=TRUE) +
    scale(matrix(rnorm(n*nrep,mean=0,sd=sd),nrow=n,ncol=nrep),center=TRUE,scale=FALSE)
  return(list(x=x, y=y, beta=beta, sigma=sd))
}
dataset = simu.data(n, p, p0, rho, nrep, SNR)
x = dataset$x
y = dataset$y
beta = dataset$beta
mu = x%*%beta
sigma = dataset$sigma
```

The first $p_0 = 6$ predictors are active with $\beta_j \neq 0$.

```
print(beta)
#>  X1  X2  X3  X4  X5  X6  X7  X8  X9 X10 X11 X12 X13 X14
#>   1  -1   1  -1   1  -1   0   0   0   0   0   0   0   0
```

Quick start

Fitting the model is simple.

```
library(boss)
# choose a single replication as illustration
```

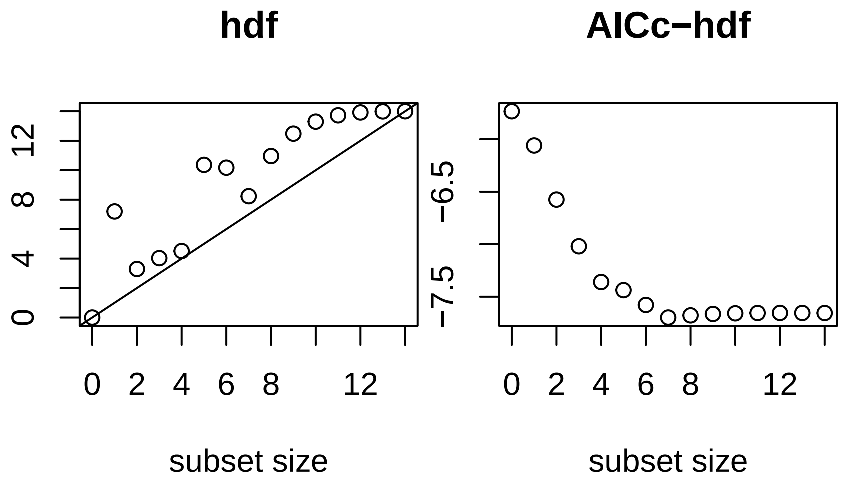
```
rep = 1
# fit the model
boss_model = boss(x, y[,rep], intercept = FALSE)
```

The 'boss' object contains estimated coefficient vectors for entire solution paths of both BOSS and FS.

```
betahat = boss_model$beta_boss
print(dim(betahat))
#> [1] 14 15
```

By default, it also calculates the hdf for BOSS and multiple information criteria.

```
# the heuristic degrees of freedom
plot(0:p, boss_model$hdf, main='hdf', ylab='', xlab='subset size')
abline(0,1)
# AICc-hdf
plot(0:p, boss_model$IC_boss$aicc, main='AICc-hdf', ylab='', xlab='subset size')
```



The optimal estimated coefficient vector and fitted mean vector can be obtained as follows.

```
# the default is chosen by AICc
betahat_aicc = coef(boss_model)
muhat_aicc = predict(boss_model, newx=x)
# use Cp rather than AICc
betahat_cp = coef(boss_model, ic='cp')
muhat_cp = predict(boss_model, newx=x)
```

Besides information criterion, cross-validation (CV) can be used as a selection rule.

```
# the default is 10-fold CV with 1 replication
boss_cv_model = cv.boss(x, y[,rep], intercept=FALSE)
# coefficient vector selected by minimizing CV error
betahat_cv = coef(boss_cv_model)
# fitted values
muhat_cv = predict(boss_cv_model, newx=x)
```

Calling 'cv.boss' runs CV for FS as well.

```
# coefficient vector for FS selected by CV
betahat_fs_cv = coef(boss_cv_model, method='fs')
# fitted values
muhat_fs_cv = predict(boss_cv_model, newx=x, method='fs')
```

Let's compare the coefficient vectors selected using different selection rules. The first three columns are for BOSS while the last column is for FS.

```
tmp = cbind(betahat_aicc, betahat_cp, betahat_cv, betahat_fs_cv)
dimnames(tmp) = list(dimnames(tmp)[[1]], c('boss_aicc', 'boss_cp', 'boss_cv', 'boss_fs_cv'))
print(tmp)
#> 14 x 4 sparse Matrix of class "dgCMatrix"
#>       boss_aicc     boss_cp     boss_cv  boss_fs_cv
#> X1   0.98882419  0.98882419  0.98882419  0.98765570
#> X2  -0.98234925 -0.98234925 -0.98234925 -0.97959482
#> X3   0.99651792  0.99651792  0.99651792  0.99624608
#> X4  -0.97551605 -0.97551605 -0.97551605 -0.97576686
#> X5   0.97769511  0.97769511  0.97769511  0.96142155
#> X6  -0.98043979 -0.98043979 -0.98043979 -0.96509794
#> X7   .           .           .           .
#> X8   .           .           .           .
#> X9  -0.05364203 -0.05364203 -0.05364203 -0.05000839
#> X10  .           .           .           .
#> X11  .           .           .          -0.02973859
#> X12  .           .           .           .
#> X13  .           .           .           .
#> X14  .           .           .          -0.01829151
```

Compare the solution paths of BOSS and FS

We see that FS gives a denser solution than BOSS. In fact, under the specific design of the true model, the true active predictors $(X_1, \cdots, X_6)$ are pairwise correlated with opposite effects. Predictors say (X_1, X_2) together lead to a high R^2 but each single one of them contributes little. Therefore, FS has trouble stepping in all the true active predictors in the early stage. For example, as indicated below, the inactive predictor X_9 joins in the first step.

```
print(boss_model$steps_fs)
#>  X9  X6  X5  X1  X2  X3  X4 X11 X14 X13  X8  X7 X12 X10
#>   9   6   5   1   2   3   4  11  14  13   8   7  12  10
```

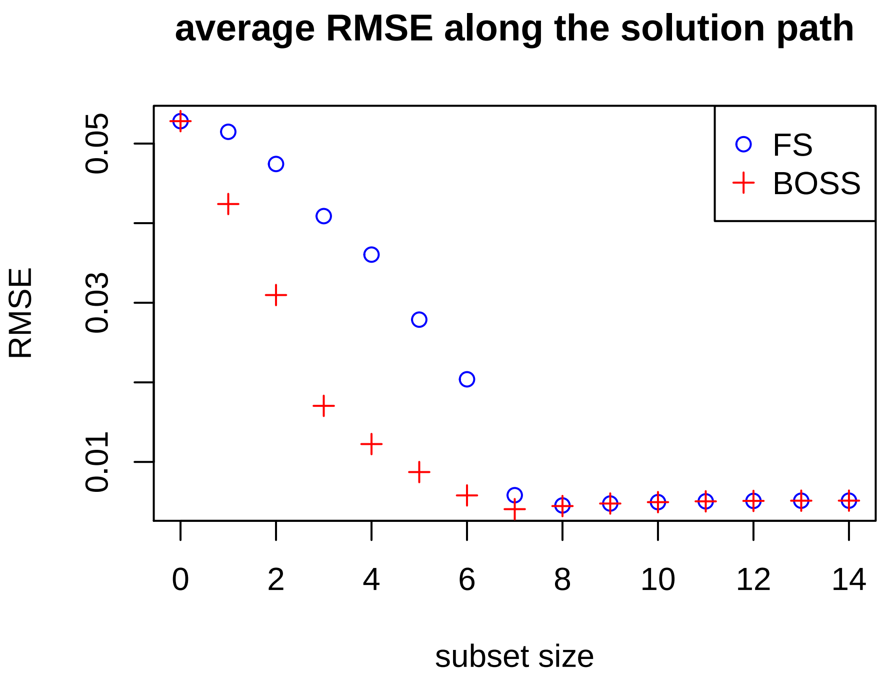
Let's set aside the selection rule for now, and compare the solution paths of the two methods. We calculate the average RMSE at each subset size based on 1000 replications. The RMSE is defined as

$$\text{RMSE} = \sqrt{\frac{1}{n}\|\hat{\mu} - X\beta\|_2^2}.$$

```
# function to calculate RMSE
calc.rmse <- function(muhat){
  sqrt( Matrix::colSums(sweep(muhat, 1, mu)^2) / n )
}
rmse_solutionpath = list(boss=list(), fs=list())
rmse = nvar = list(boss=c(), fs=c())
for(rep in 1:nrep){
  boss_cv_model = cv.boss(x, y[,rep], intercept=FALSE)
  # RMSE along the solution path
  rmse_solutionpath[['boss']][[rep]] = calc.rmse(x %*% boss_cv_model$boss$beta_boss)
  rmse_solutionpath[['fs']][[rep]] = calc.rmse(x %*% boss_cv_model$boss$beta_fs)
  # RMSE for the optimal subset selected via a selection rule
  rmse[['boss']][rep] = calc.rmse(predict(boss_cv_model$boss, newx = x)) # AICc
  rmse[['fs']][rep] = calc.rmse(predict(boss_cv_model, newx = x, method = 'fs')) # CV
  # number of variables
  nvar[['boss']][rep] = sum(coef(boss_cv_model$boss)!=0)
  nvar[['fs']][rep] = sum(coef(boss_cv_model, method='fs')!=0)
}
```

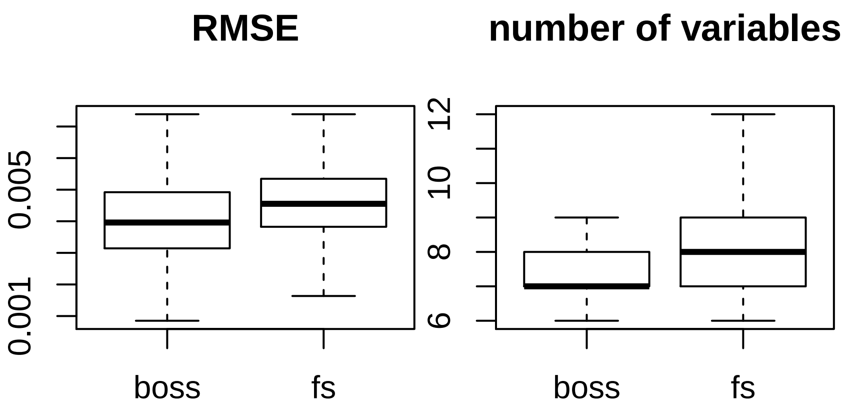
BOSS clearly provides a better solution path than FS in steps less than 8.

```
# average RMSE over replications
rmse_avg = lapply(rmse_solutionpath, function(xx){colMeans(do.call(rbind, xx))})
plot(0:p, rmse_avg$fs, col='blue', pch=1, main='average RMSE along the solution path',
     ylab='RMSE', xlab='subset size')
points(0:p, rmse_avg$boss, col='red', pch=3)
legend('topright', legend = c('FS', 'BOSS'), col=c('blue', 'red'), pch=c(1,3))
```



Next, we bring back the selection rule and compare their performances. The selection rule for BOSS is AICc-hdf and it's 10-fold CV for FS. BOSS shows better predictive performance, and it provides more sparse solutions than FS.

```
boxplot(rmse, outline=FALSE, main='RMSE')
boxplot(nvar, outline=FALSE, main='number of variables')
```



Real data examples

We compare the performance of BOSS with FS and some popular regularization methods, through some real datasets. We consider four datasets from the StatLib library, that are the 'boston housing', 'hitters', 'auto' and 'college'. An intercept term is included in all the procedures. We present the result first and show the simulation code in the Appendix at the end of this document.

The selection rule is AICc for BOSS and LASSO, and 10-fold CV for FS and SparseNet. We see that BOSS is the best for 'hitters' and 'auto' datasets, while LASSO is the best for 'boston housing' and 'college'. LASSO based on the coordinate descent algorithm, is the fastest method among all. BOSS is also considerably computationally efficient by comparing with FS and SparseNet.

Dataset	n..p	Metrics	BOSS	FS	LASSO	SparseNet
boston	506, 13	RMSE	3.372	3.371	3.363	3.370
		# predictors	12.004	12.016	12.012	12.000
		running time (s)	0.031	0.232	0.006	0.317
hitters	263, 19	RMSE	233.853	235.014	234.064	239.427
		# predictors	11.152	10.627	14.205	12.905
		running time (s)	0.018	0.124	0.006	0.414
college	777, 17	RMSE	1565.476	1567.612	1564.807	1570.550
		# predictors	17.991	16.108	16.008	15.425
		running time (s)	0.097	0.766	0.009	0.574
auto	392, 6	RMSE	2.628	2.628	2.643	2.628
		# predictors	3.000	3.000	5.008	3.008
		running time (s)	0.011	0.096	0.006	0.198

Appendix: simulation code for the real data examples

Code to process the datasets. We remove all the entries with 'NA' values. And we recast binary category variables into $\{0, 1\}$ and remove category variableswith more than 2 categories.

```r
library(ISLR)
dataset = list()
# Boston Housing data
tmp = Boston
tmp = na.omit(tmp)
tmp$chas = as.factor(tmp$chas)
dataset$boston$x = data.matrix(tmp[,!names(tmp) %in% 'medv'])
dataset$boston$y = tmp$medv

# MLB hitters salary
tmp = Hitters
tmp = na.omit(tmp)
tmp[,c('League', 'Division', 'NewLeague')] =
  lapply(tmp[,c('League', 'Division', 'NewLeague')], as.factor)
dataset$hitters$x = data.matrix(tmp[,!(names(tmp) %in% c('Salary'))])
dataset$hitters$y = tmp$Salary

# # College data
tmp = College
tmp$Private = as.factor(tmp$Private)
dataset$college$x = data.matrix(tmp[,!(names(tmp) %in% c('Outstate'))])
dataset$college$y = tmp$Outstate
```

```r
# Auto data
tmp = Auto
dataset$auto$x = data.matrix(tmp[,!(names(tmp) %in% c('mpg','name','origin'))])
dataset$auto$y = tmp$mpg
```

Code to calculate LOO error, number of variables and timing for each fitting procedure.

```r
# The following will take about 30 minutes to run
library(glmnet)
library(sparsenet)
rmse <- function(y_hat, y){
  sqrt(sum( (y_hat - y)^2 / length(y)) )
}
rdresult <- function(x, y, nrep){
  p = dim(x)[2]

  allmethods = c('lasso','sparsenet','boss','fs')
  error = numvar = time = replicate(length(allmethods), rep(NA,nrep), simplify=F)
  names(error) = names(numvar) = names(time) = allmethods

  for(i in 1:nrep){
    index = 1:nrow(x)
    index = index[-i]

    x.train = x[index, , drop=FALSE]
    y.train = y[index]
    x.test = x[-index, , drop=FALSE]
    x.test.withint = cbind(rep(1,nrow(x.test)), x.test)
    y.test = y[-index]

    # BOSS
    ptm = proc.time()
    boss_model = boss(x.train, y.train, intercept = TRUE)
    time_tmp = proc.time() - ptm
    boss_pred = as.numeric( predict(boss_model, newx=x.test) )
    error$boss[i] = rmse(boss_pred, y.test)
    numvar$boss[i] = sum(coef(boss_model)!=0)
    time$boss[i] = time_tmp[3]

    # FS
    ptm = proc.time()
    boss_cv_model = cv.boss(x.train, y.train)
    time_tmp = proc.time() - ptm
    fs_pred = as.numeric( predict(boss_cv_model, newx=x.test, method='fs') )
    error$fs[i] = rmse(fs_pred, y.test)
    numvar$fs[i] = sum(coef(boss_cv_model, method='fs')!=0)
    time$fs[i] = time_tmp[3]

    # LASSO
    ptm = proc.time()
    lasso_model = glmnet(x.train, y.train, intercept=TRUE)
    lasso_aicc = as.numeric(calc.ic(predict(lasso_model, newx=x.train), y.train,
                                    ic='aicc', df=lasso_model$df+1))
    lasso_pred = predict(lasso_model, newx=x.test, s=lasso_model$lambda[which.min(lasso_aicc)])
    time_tmp = proc.time() - ptm
    error$lasso[i] = rmse(lasso_pred, y.test)
    numvar$lasso[i] = sum(coef(lasso_model, s=lasso_model$lambda[which.min(lasso_aicc)])!=0)
    time$lasso[i] = time_tmp[3]

    # Sparsenet
    ptm = proc.time()
    sparsenet_cv_model = cv.sparsenet(x.train, y.train)
    time_tmp = proc.time() - ptm
    sparsenet_pred = predict(sparsenet_cv_model, newx=x.test, which='parms.min')
    error$sparsenet[i] = rmse(sparsenet_pred, y.test)
    numvar$sparsenet[i] = sum(coef(sparsenet_cv_model, which='parms.min')!=0)
    time$sparsenet[i] = time_tmp[3]
  }
  return(list(error=error, numvar=numvar, time=time))
}
result = lapply(dataset, function(xx){rdresult(xx$x, xx$y, nrow(xx$x))})
```

Code to make the table

```r
library(knitr)
library(kableExtra)
tmp = data.frame(Dataset = rep(names(result), each=3),
  "n, p" = rep(unlist(lapply(dataset, function(xx){paste(dim(xx$x), collapse = ', ')})) , each=3),
  Metrics = rep(c('RMSE', '# predictors', 'running time (s)'), length(result)),
  BOSS = unlist(lapply(result, function(xx){unlist(lapply(xx, function(yy){round(mean(yy$boss), 3)}))}))
  FS = unlist(lapply(result, function(xx){unlist(lapply(xx, function(yy){round(mean(yy$fs), 3)}))})),
  LASSO = unlist(lapply(result, function(xx){unlist(lapply(xx, function(yy){round(mean(yy$lasso), 3)}))
  SparseNet = unlist(lapply(result, function(xx){unlist(lapply(xx, function(yy){round(mean(yy$sparsenet
rownames(tmp) = NULL
kable(tmp, align = "c") %>%
  kable_styling(full_width = F) %>%
  column_spec(1, bold = T) %>%
  collapse_rows(columns = 1:2, valign = "middle")
```