Sparse orthogonal factor regression (SOFAR)

Dong Liang and Grace Guinan

In classical statistical learning with a univariate response y and a set of predictors $x_j, j = 1, \dots, p$, the problem can be phrased as selecting a subset of the predictors for the response. Intuitively this set includes all important predictors while excluding any spurious predictors. In big data applications, the number of predictors p is typically large. The large p renders classical learning techniques such as least square regression inappropriate, because the least square coefficients are not uniquely estimable (Hastie et al. 2009).

To achieve selection in big data applications, various penalized least square methods have been proposed. A very popular method is called *lasso*. The lasso method is based on the assumption that only a small subset of the predictors can be sufficiently predictive of the response. In terms of the association vector between response and predictors, this vector is assumed to be *sparse*. The sparsity of the solution vector can be achieved through an L1 penalty term added to the usual least square objective function.

$$\hat{\beta} = \text{argmin}_{\beta} \left\{ (y - X\beta)^T (y - X\beta) + \lambda |\beta|_1 \right\}$$

Robust numerical algorithms in terms of convergence and computational efficiency have been developed for the lasso method in the univariate response case.

The sparsity principle can be applied to multivariate responses as well, specifically to uncover an association network between a response matrix Y and a predictor matrix X. Denote n the number of rows for both matrices, q the columns for Y, and p the columns for X. The matrices generically capture different sets of measurements on the same n samples. Examples include genetic expressions and genetic variants. In this project, the response matrix Y is the component of various fluorescence measurements, and the predictor matrix X is the high-resolution mass-spectrometry (HRMS) intensity variables. In these big-data applications, n is typically small, but either q or p (or both) can be large.

One method to uncover an association network is proposed by Uematsu et al. (2019). This method assumes there are a small number of processes controlling the association between responses and predictors. In their application, this is an expression pathway from genetic variants to certain gene expressions or phenotypes (Uematsu et al. 2019). The small number of processes is called the rank r. The association network is encoded by a $p \times q$ matrix C where a non-zero entry C_{ij} denotes that predictor i is associated with response j, while controlling for the effects of all other predictors. Their method estimates the C matrix as a tuple of matrices (U, D, V) where $C = UDV$, D is a diagonal matrix of dimension r, U is a sparse and orthogonal matrix of dimension $p \times r$, V is a sparse and orthogonal matrix of dimension $q \times r$.

The U matrix selects r sets of predictors, each set is highly predictive of a corresponding set of responses selected by the corresponding column of the V matrix. These sets are called latent predictors and the corresponding response. The $k, k = 1, \dots, r$ latent predictor is the k column of matrix XU, likewise, the k latent response is the k column of matrix YV. The predictors associated with each latent predictor are distinct due to the orthogonality of the matrix U, likewise, the responses associated with each latent response are also distinct. Thus the algorithm generated a very sparse network encoding the association between Y and X through the latent variables. The solution matrix C is very high dimensional but typically sparse.

The R package `rrpack` implements the statistical learning algorithm SOFAR. The theoretical property of the algorithm in terms of its convergence, the computing details, and method derivation can be found in the paper (Uematsu et al. 2019). This handout introduces the method and explains how it was applied to Dissolved Organic Matter datasets.

Packages and Data

First, load `rrpack` package and other useful packages

```
rm(list=ls())
library(rrpack)
library(stringr)
library(ggplot2)
library(dplyr)
```

And load the data.

```
X ← read.csv("data/HRMS_data.csv")
Y ← read.csv("data/FDOM_data.csv")
```

Normalization

Next we normalize the `X` dataset using the quantile method.

```
# center and scale (quant)
X2←sapply(X[,-c(1:3)], function(x) quantile(x,probs = c(.05,.95) ))
X3 ←as.data.frame(t(X2))

# make new column subtracting 95th and 5th percentile
X3$Sj ← X3$‘95%‘ - X3$‘5%‘

# select only nonzero values
X4 ← X3[X3$Sj != 0,]

#delete these compounds from original dataset
X4$ID ← row.names(X4)
XX ← X[,c("Sample", "Location", "Depth",X4$ID)]

# add Sj into X dataset as a row
Sv←X4$Sj
Sv←t(c(NA, NA, NA, Sv))
colnames(Sv)← colnames(XX)
XX← rbind(Sv, XX)

# make Z dataset (divide X by Sj)
XX2← XX[-c(1:3)]
Z ← XX2/XX2[rep(1,nrow(XX2)),]
Z← cbind(XX[,1:3],Z)
Z← Z[-1,]
```

Merge X and Y

Next we merge the X and Y datasets so they contain the same observations.

```
idlist ← list()
for (i in 1:145){
  q←str_sub(Z$Sample[i], -1, -1)
  idlist[[length(idlist)+1]] ←q
}
Z$Profile ←idlist

# merge data together
Z2 ← merge(Z,Y[,c("Profile","Depth","Location")],by=c("Profile","Depth","
    Location"))
y3 ← merge(Y,Z2[,c("Profile","Depth","Location")],by=c("Profile","Depth","
    Location"))
```

SOFAR

Next, we apply the SOFAR learning algorithm to uncover the association between Y and X. The resulting fit is called `fit_quantile_all1234` because it is the quantile SOFAR run on the full `X` dataset using components 1-4 in the `Y` dataset.

```
if(!interactive()){
  z31 ← as.matrix(Z2[,-c(1:4)])
  y31 ← as.matrix(y3[,-c(1:3)])
  fit_quantile_all1234← sofar(y31, z31, nrank = 3,
                                    control=list(nlam = 100,
                                                 lam.min.factor=1e-10,
                                                 lam.max.factor=5),ic.type="BIC")

  save(fit_quantile_all1234, file="sofarFit2ALL_quantile1234.rData")
}
```

This code is very time-consuming. It's best to not run this interactively. Instead, it should be allowed to run on a high-performance computer overnight. The expected timing for running this on the real data is around 4 hours. The algorithm takes the Y and X data frames, which were converted to matrices. The maximum rank is assumed to be 3, this defines the upper limit of the number of latent pairs of variables to uncover. The `control` argument specifies some internal tuning parameters that will allow convergence of the algorithms to non-trivial solutions. The `ic.type` provides the details for estimating the number of ranks. We will settle at BIC as it's indicative of the predictive performance. Lastly, we save the output of the algorithm into a physical file in the current working directory.

Now suppose the hours have passed and we are ready to do statistical inferences. We first collect the results.

```
load(file="fit/sofarFit2ALL_quantile1234.rData")
```

The code loads the saved object `fit_quantile_all1234` into R.

Fitted Values

Like any regression method, we can use the R^2 values to evaluate the fit of the SOFAR method.

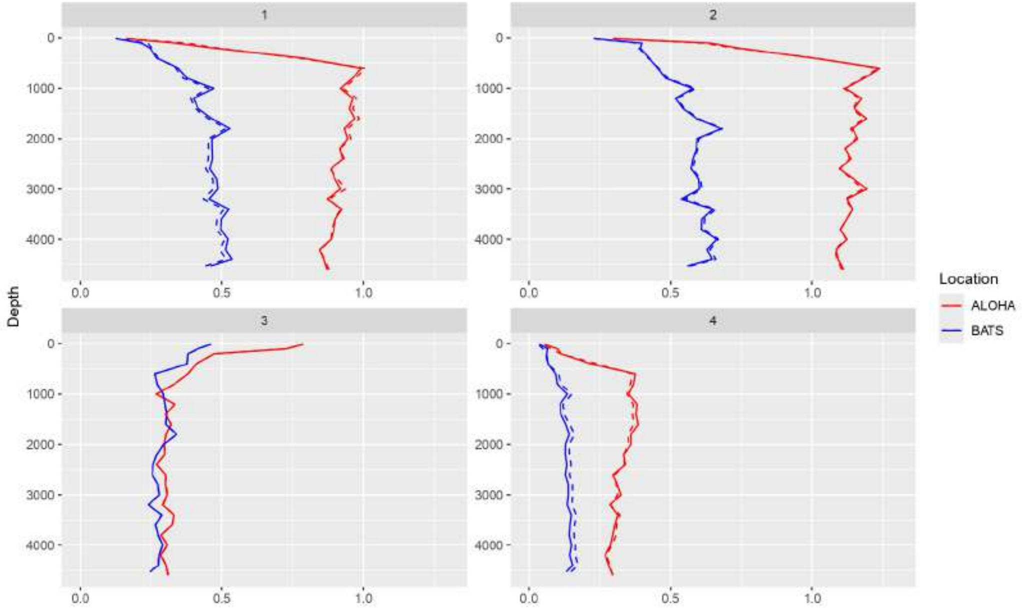
```
## fitted value
fitted_sofar ← function(x){
  with(x,X%*%U%*%diag(D)%*%t(V))
}
yhat ← fitted_sofar(fit_quantile_all1234)

## R-square
totalVar ← function(x){
  sum(diag(crossprod(x)))
}
1-totalVar(fit_quantile_all1234$Y-yhat)/totalVar(fit_quantile_all1234$Y)
```

```
[1] 1
```

The helper function `fitted_sofar` computes the fitted Y, this is a matrix of the same dimension as the response Y. The helper function `totalVar` compute the variance in a matrix, which is the trace of its cross product. The R^2 is computed as the proportion of variance explained by the SOFAR regression.

The fits can also be assessed visually.

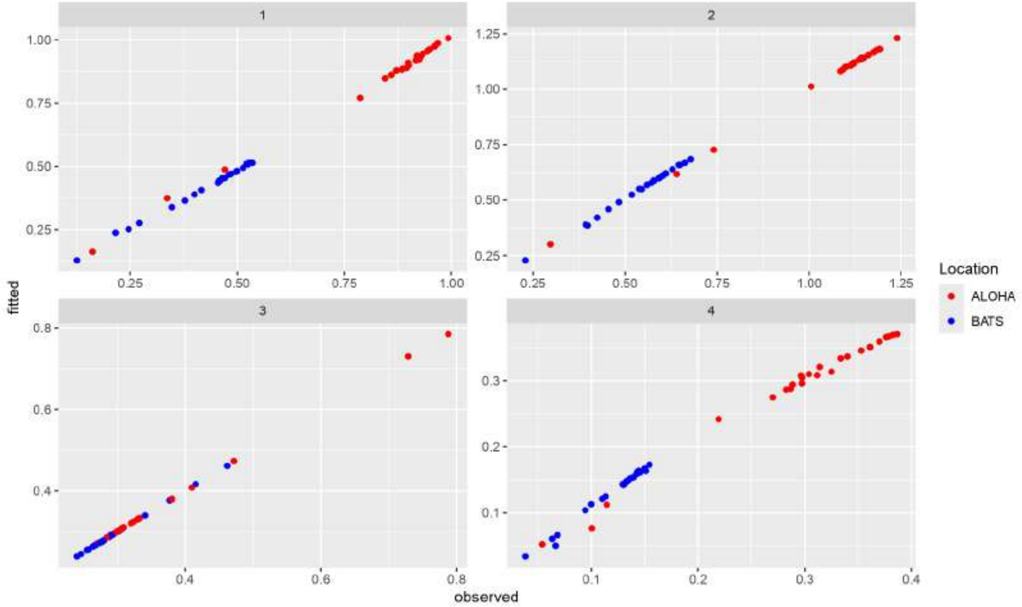
```
## plot fitted versus observed
linkY ← y3[c(1:3)]
linkY$i ← as.numeric(row.names(linkY))

data_ ← data.frame(
  y=c(fit_quantile_all1234$Y),
  fit=c(yhat),
  i=c(row(fit_quantile_all1234$Y)),
  j=c(col(fit_quantile_all1234$Y))
)
data_1 ← full_join(linkY, data_, by = "i")
avgdat ← aggregate(cbind(y, fit)~j +Location +Depth, FUN= mean, data = data_1
    )

#graph with location and depth
ggplot(data =avgdat,aes(y = y, x = Depth, group = Location))+geom_line(
    linetype=1, aes(color = Location))+
  geom_line(aes(x=Depth,y=fit , color=Location),linetype=2)+
  facet_wrap(~j,scales = "free")+coord_flip() + scale_x_reverse()+ylim(0,1.3)+
      scale_color_manual(values = c("red", "blue"))+theme(axis.title.x =
      element_blank())
```



The above codes show the fitted values in the dashed line and the observed data in the solid line. Each panel shows one response variable, and the horizontal axis shows the sample number.

The fits appear to track the data well. We can use another visual to evaluate the performance.

```
## scatter plot
ggplot(avgdat,aes(x=y,y=fit, color = Location))+geom_point()+
  facet_wrap(~j,scales="free")+xlab("observed")+ylab("fitted")+
     scale_color_manual(values = c("red", "blue"))
```

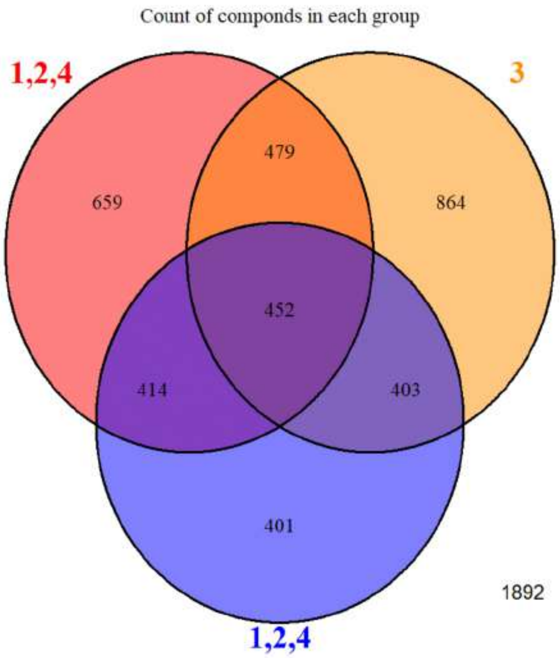


The above plots compare directly the fit and the observed and evaluate the fitting performance of SOFAR.

Venn Diagram

We can also visualize the overlap between the SOFAR groupings

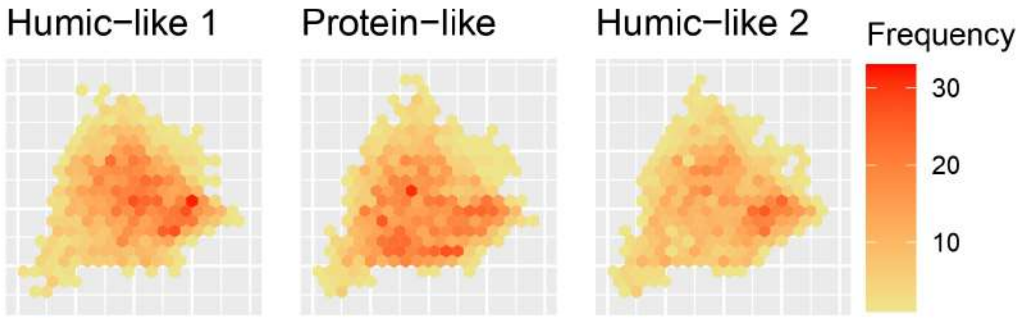
```
#install.packages("VennDiagram")
#install.packages("gplots")
library(VennDiagram)
library(gplots)

# make T/F dataset for if compounds are inclued in each grouping
ULL2 ← as.data.frame(fit_quantile_all1234[["U"]])
ULL2[ULL2 ≥ .01] ←1
ULL2[ULL2 ≤ −.01] ←1
ULL2$V1 ← ULL2$V1==1
ULL2$V2 ← ULL2$V2==1
ULL2$V3 ← ULL2$V3==1

# Create logical vectors
set1 ← ULL2$V1
set2 ← ULL2$V2
set3 ← ULL2$V3

venn_data ← list(
  col1 = which(set1),
  col2 = which(set2),
  col3 = which(set3)
)
unselected_count ← sum(!set1 & !set2 & !set3)

# Generate the Venn diagram
venn.plot ← venn.diagram(
  x = venn_data,
  category.names = c("1,2,4", "3", "1,2,4"),
  output = TRUE,
  main = "Count of componds in each group",
  filename = NULL,
  fill = c("red", "darkorange", "blue"),
  alpha = 0.5,
  cat.col = c("red", "darkorange", "blue"),
  cat.cex = 1.5,
  cat.fontface = "bold"
)

grid.newpage()
grid::grid.draw(venn.plot)
grid.text(paste(unselected_count), x = 0.9, y = 0.1, gp = gpar(fontsize = 12,
    col = "black"))
```



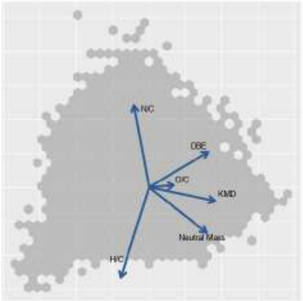
Heatmap

Now we are ready to visualize the association network in the form of a heatmap. Helper functions `heatmap` and `pca_heatmap` from the `heatmap.R` file are used.

```
library(viridis)
library(cowplot)

# load metadata and run PCA
metadat ← read.csv("data/metadat.csv")
meta.pca ←princomp(metadat[-1], cor = TRUE)
ggdata_ ← data.frame(meta.pca$scores, ID = metadat$ID)

# make heat map
source("helper_functions/heatmap.R")
ID_quant ← as.list(read.csv("data/quant_ids.csv")$ID)
heatmap(fit_quantile_all1234, ID_quant, ggdata_)
```



```
# make PCA
loads <- as.data.frame(meta.pca$loadings[,c(1:2)])
pca_heatmap(ggdata_, loads)
```



Here we can see the PCA-arranged compounds from groups created by SOFAR association network of whole dataset. PCA was conducted with the meta variables: H/C ratio, O/C ratio, number of N-containing molecular formulas, double bond equivalence (DBE), Kendrick Mass (KMD), and Neutral Mass. Selected compounds were PCA-arranged with component 1 as the x-axis (Var: 42.1) and component 2 as the y-axis (Var= 19.8). A density-based coloring was applied (scale 0 -30).

References

1. Hastie, T., et al. (2009). The elements of statistical learning: data mining, inference, and prediction, Springer.

2. Uematsu, Y., et al. (2019). "SOFAR: Large-scale association network learning." IEEE transactions on information theory 65(8): 4924-4939.