Convex Biclustering Demo

Eric Chi

2/22/2020

Introduction

We demonstrate a basic processing pipeline for generating a COBRA smoothed solution path. We use the lung cancer data set that comes with the package. The data consists of a 100-by-56 matrix of 100 gene expression levels for 56 tissue samples. More information on this data set can be found in the help documentation for the lung data set. Note the mathematical details of COBRA can be found in the Biometrics manuscript.

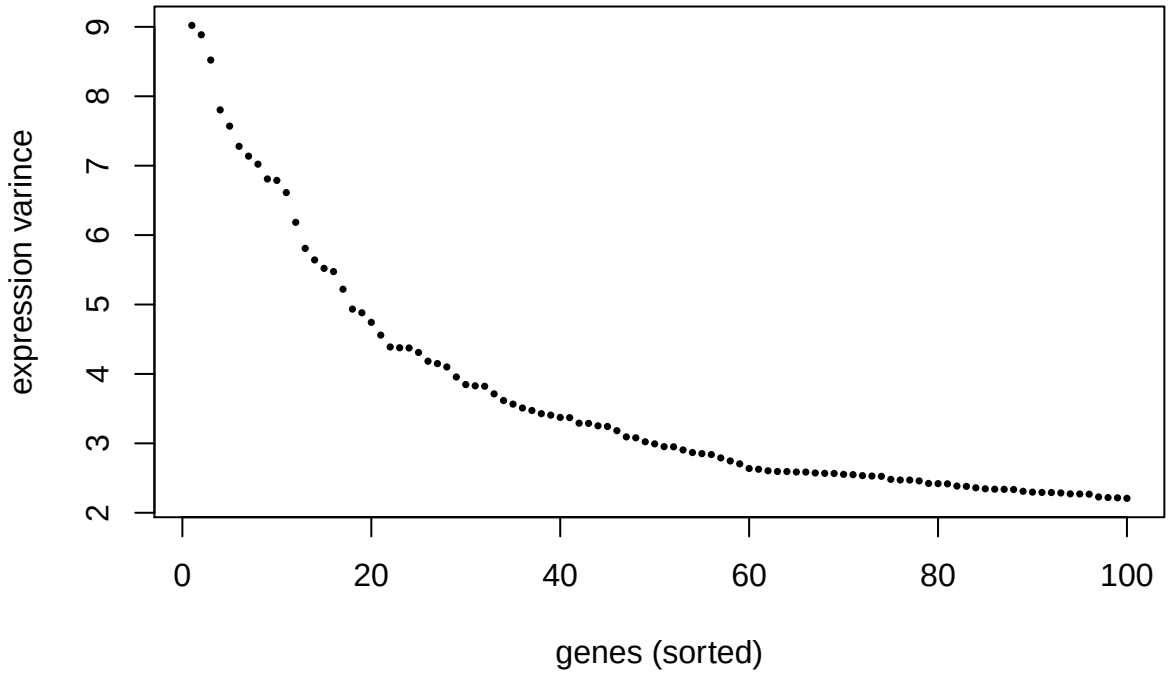
This companion document details how to use the accompanying R package.

Step 1: Filtering features for high dimensional, low sample size data

Some data naturally consist of many more features than samples, e.g.@ genomics data. Here we may have hundreds of subjects (samples) compared to tens of thousands or more genes (features). Many of these features, however, do not aid in revealing underlying subgrouping among the subjects. They are basically noise. Of course the challenge is that we do not know in advance which features are relevant and others noise. Filtering feature is an open problem. Nonetheless, a good rule of thumb is to eliminate features that have little variance across subjects.

The following plot shows the sorted sample variance of the 100 genes in the lung cancer data set.

```
lung_sort <- sort(apply(lung,1,var), decreasing=TRUE, index.return=TRUE)
plot(lung_sort$x, pch=16, ylab="expression varince", xlab="genes (sorted)", cex=0.5)
```



The variance decays slowly, and there is no clear cut segmentation into genes with high variance and low variance. The plot in fact reminds us of a scree plot of eigenvalues for principal component analysis (PCA).

We may opt to choose a cut-off at the inflection or "elbow-point," which 30. Nonetheless, for the sake of illustration we will include more genes and choose a cutoff of 45.

```
X <- lung[lung_sort$ix[1:nF],,drop=FALSE]
```

Step 2: Normalize data and construct weights and edge-incidence matrices

Having prefiltered the feature set, we next center and normalize the data matrix and then construct weight and edge-incidence matrices. It is not necessary to center and normalize the data matrix, but it is important to have a relatively small dynamic range for the default weight graph construction (sparse Gaussian kernel weights) otherwise we will run into numerical underflow. For simplicity we center and normalize the data and then use the resulting transformed matrix as input into the gkn_weights function which computes the sparse Gaussian kernel weights.

```
## Center by subtracting the grand mean of the data matrix
X <- X - mean(c(X))
## Normalize to have unit Frobenius norm
X <- X/norm(X,'f')

## Construct weights and edge-incidence matrices
phi <- 0.5; k_row <- k_col <- 4
wts <- gkn_weights(X,phi=phi,k_row=k_row,k_col=k_col)
w_row <- wts$w_row
w_col <- wts$w_col
E_row <- wts$E_row
E_col <- wts$E_col
```

The output of gkn_weights is a list of row and column weights (w_row and w_col), row and column edge incidence matrices (E_row and E_col), and the number of connected components in the row graph and column graph (nRowComp, nColComp). Column weight between the ith and jth column is calculated as follows. First preweights $\hat{w}_{ij}$ are computed, and then the preweights are rescaled to give the weights w_{ij}.

$$\hat{w}_{ij} = \begin{cases} \exp\left(-\frac{\phi}{p}\|x_{\cdot i} - x_{\cdot j}\|_2^2\right) & \text{if the } i\text{th column is the } k\text{\_col th nearest neighbor of the } j\text{th column or vice versa.} \\ 0 & \text{otherwise.} \end{cases}$$

$$w_{ij} = \frac{1}{\sqrt{p}} \frac{\hat{w}_{ij}}{\sum_{i,j} \hat{w}_{ij}}.$$

Row weights are analogously computed. See the Biometric manuscript for the rationale for the sparse Gaussian kernel and the scalings employed. Note that the row and column edge-incidence matrices are sparse matrices (Matrix::dgCMatrix class).

A natural question is how to choose the scale parameter of the Gaussian kernel, phi, and also the nearest-neighbor numbers k_row and k_col. I have found using phi between 0.5 and 1 to give similar results. I do not advise setting phi to 0 because the resulting uniform weights tends to oversmooth the data. Choosing phi to be larger than 1 may cause numerical underflow. A probably more principled approach is to choose phi to match the local scale of scatter, namely take the distance between each point and its nearest k-th nearest neighbor (rule of thumb: k= 1-10), then take the average or median of these.

As for choosing the number of nearest neighbors k_row and k_col, a good rule of thumb is to choose the smallest number of nearest neighbors so that the row and column graphs are connected.

```
## Connected Components of Row and Column Graphs
wts$nRowComp
```

```
## [1] 1
```

```
wts$nColComp
```

```
## [1] 2
```

Typically you want to generate a complete solution path, i.e. all the centroids eventually fuse, so you want there to be one connected component for the row graph and one connected component for the column graph. See the JCGS paper entitled 'Splitting Methods for Convex Clustering' for more discussion on choosing these parameters.

Step 3: Generate a solution path and perform model selection

The data is now ready to be inputed into the COBRA method. We first construct a sequence of regularization parameters.

```
#### Initialize path parameters and structures
nGamma <- 7
gammaMax <- 1
gammaMin <- 0
gammaSeq <- 10**seq(gammaMin,gammaMax,length.out=nGamma)
```

Note that we use a log-linear scale. This is generally true for any penalized regression method. You will need to apply greater amounts of penalty to get more structured solutions. Choosing the largest amount penalty requires some trial and error. Try a value of gammaMax. Does it result in a solution where there is one bicluster? If not then try doubling it until it does. Note that the smallest value of gammaMax that will give the most biclustered solution is itself the solution to an optimization problem, a second order cone program.

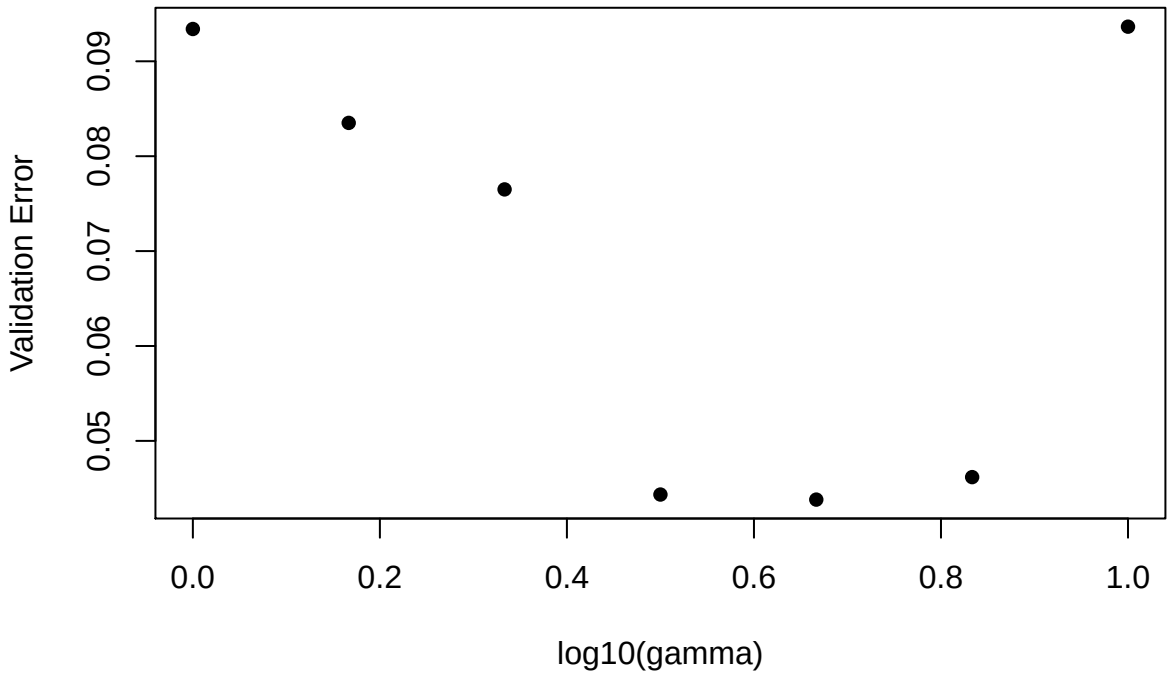
Once we have the sequence of regularization parameters, we are ready to generate the solution path.

```
## Generate solution path
sol <- cobra_validate(X,E_row,E_col,w_row,w_col,gammaSeq,fraction=0.01)
```

```
## [1] "***** Completed gamma = 1 *****"
## [1] "***** Completed gamma = 2 *****"
## [1] "***** Completed gamma = 3 *****"
## [1] "***** Completed gamma = 4 *****"
## [1] "***** Completed gamma = 5 *****"
## [1] "***** Completed gamma = 6 *****"
## [1] "***** Completed gamma = 7 *****"
```

The function cobra_validated aids variable selection as well. The input fraction indicates the fraction of entries in the matrix to hold out to compute a hold out error. The hold out error is recorded. We next plot it.

```
## Plot validation error
verr <- sol$validation_error
plot(log10(gammaSeq), verr, pch=16, ylab="Validation Error", xlab="log10(gamma)")
```

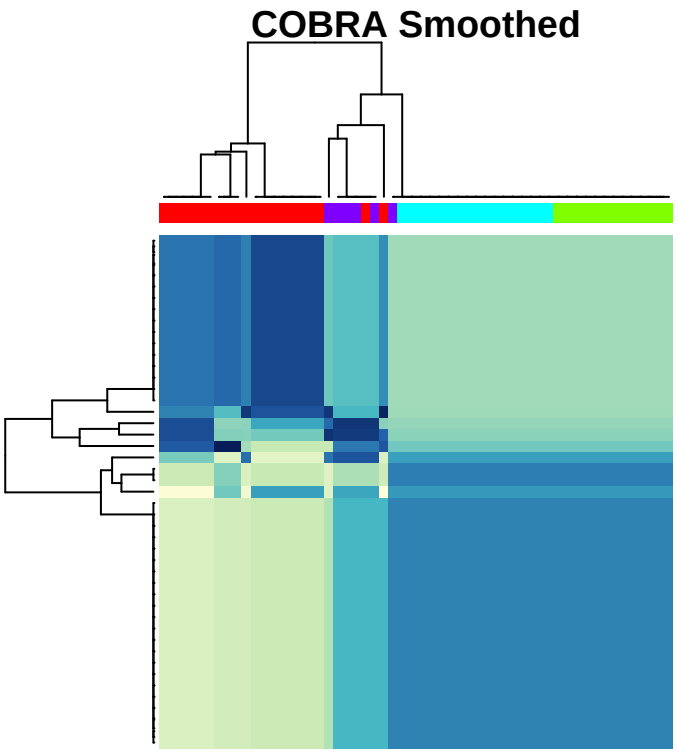


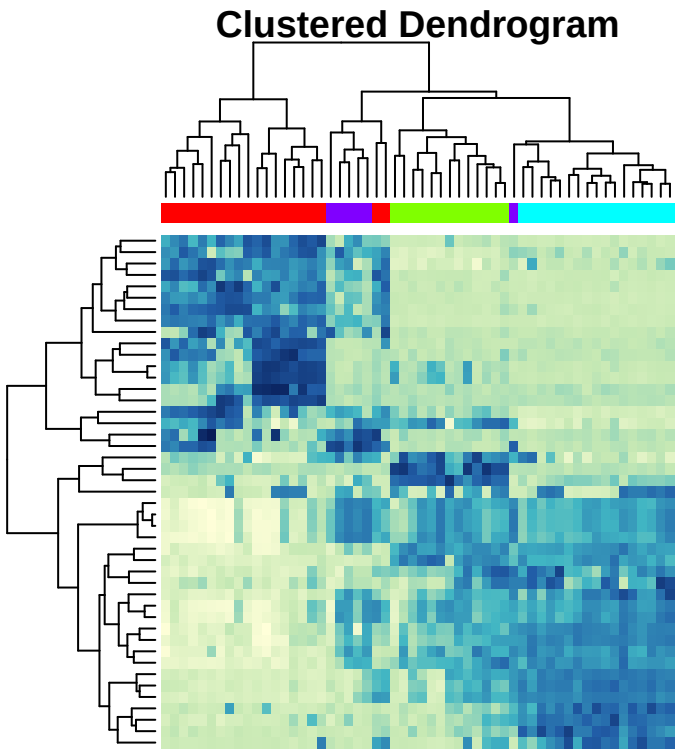
We next show the heatmap of the best COBRA smoothed estimate based on the hold out validation error. For reference we also show the clustered dendrogram of the centered and scaled data. Note that we used the native hierarchical clustering of the heatmap function. You could instead use a different row and column ordering based the seriation package for example. Future releases of cvxbiclustr will use a customized dendrogram based on the solution path of COBRA.

```
ix_min <- which.min(verr)
## Extract the row and column groupings
groups_row <- sol$groups_row[[ix_min]]
groups_col <- sol$groups_col[[ix_min]]
## Smooth the data using the resulting biclustering assignments
M <- biclust_smooth(X,groups_row,groups_col)

## Create annotation for heatmap
types <- colnames(lung)
ty <- as.numeric(factor(types))
cols <- rainbow(4)
YlGnBu5 <- c('#ffffd9','#c7e9b4','#41b6c4','#225ea8','#081d58')
hmcols <- colorRampPalette(YlGnBu5)(256)

## Heatmap of data smoothed at the model selected to minimize validation error
heatmap(M,col=hmcols,labRow=NA,labCol=NA,ColSideCol=cols[ty],main="COBRA Smoothed")
## Clustered dendrogram of the data
heatmap(X, col=hmcols,labRow=NA,labCol=NA,ColSideCol=cols[ty],main="Clustered Dendrogram")
```


COBRA Smoothed


Clustered Dendrogram