projectR Vignette

Gaurav Sharma, Genevieve L. Stein-O'Brien

8 March 2019

Contents

```
knitr::opts_chunk$set(echo = TRUE)
options(scipen = 1, digits = 2)
set.seed(1234)
```

1 Introduction

Technological advances continue to spur the exponential growth of biological data as illustrated by the rise of the omics—genomics, transcriptomics, epigenomics, proteomics, etc.—each with there own high throughput technologies. In order to leverage the full power of these resources, methods to integrate multiple data sets and data types must be developed. The reciprocal nature of the genomic, transcriptomic, epigenomic, and proteomic biology requires that the data provides a complementary view of cellular function and regulatory organization; however, the technical heterogeneity and massive size of high-throughput data even within a particular omic makes integrated analysis challenging. To address these challenges, we developed projectR, an R package for integrated analysis of high dimensional omic data. projectR uses the relationships defined within a given high dimensional data set, to interrogate related biological phenomena in an entirely new data set. By relying on relative comparisons within data type, projectR is able to circumvent many issues arising from technological variation. For a more extensive example of how the tools in the projectRR package can be used for *in silico* experiments, or additional information on the algorithm, see Stein-O'Brien, et al.

2 Getting started with projectR

2.1 Installation Instructions

For automatic Bioconductor package installation, start R, and run:

```
source("https://bioconductor.org/biocLite.R")
biocLite("projectR")
```

For the current development level version, start R, and run:

```
library(devtools)
install_github("projectR", "genesofeve")
```

2.2 Methods

Projection can roughly be defined as a mapping or transformation of points from one space to another often lower dimensional space. Mathematically, this can described as a function $\varphi(x) = y : \Re^D \mapsto \Re^d$ s.t. $d \leq D$ for $x \in \Re^D, y \in \Re^d$ Barbakh, Wu, and Fyfe (2009) . The projectR package uses projection functions defined in a training dataset to interrogate related biological phenomena in an entirely new data set. These functions can be the product of any one of several methods common to "omic" analyses including regression, PCA, NMF, clustering. Individual sections focussing on one specific method are included in the vignette. However, the general design of the projectR function is the same regardless.

2.3 The base projectR function

The generic projectR function is executed as follows:

```
library(projectR)
projectR(data, loadings, dataNames=NULL, loadingsNames=NULL, NP = NULL, full = false)
```

2.3.1 Input Arguments

The inputs that must be set each time are only the data and loadings, with all other inputs having default values. However, incongruities in the feature mapping between the data and loadings, i.e. a different format for the rownames of each object, will throw errors or result in an empty mapping and should be checked before running. To overcoming mismatched feature names in the objects themselve, the /code{dataNames} and /code{loadingNames} arguements can be manually supplied by the user.

The arguments are as follows:

data a dataset to be projected into the pattern space

loadings a matrix of continous values with unique rownames to be projected

dataNames a vector containing unique name, i.e. gene names, for the rows of the target dataset to be used to match features with the loadings, if not provided by `rownames(data)`. Order of names in vector must match order of rows in data.

loadingsNames a vector containing unique names, i.e. gene names, for the rows of loadings to be used to match features with the data, if not provided by `rownames(loadings)`. Order of names in vector must match order of rows in loadings.

NP vector of integers indicating which columns of loadings object to use. The default of NP = NA will use entire matrix.

full logical indicating whether to return the full model solution. By default only the new pattern object is returned.

The `loadings` arguement in the generic projectR function is suitable for use with any genernal feature space, or set of feature spaces, whose rows annotation links them to the data to be projected. Ex: the coeffients associated with individual genes as the result of regression analysis or the amplituded values of individual genes as the result of non-negative matrix factorization (NMF).

2.3.2 Output

The basic output of the base projectR function, i.e. `full=FALSE`, returns `projectionPatterns` representing relative weights for the samples from the new data in this previously defined feature space, or set of feature spaces. The full output of the base projectR function, i.e. `full=TRUE`, returns `projectionFit`, a list containing `projectionPatterns` and `Projection`. The `Projection` object contains additional information from the proceedure used to obtain the `projectionPatterns`. For the the the base projectR function, `Projection` is the full lmFit model from the package *limma*

3 PCA projection

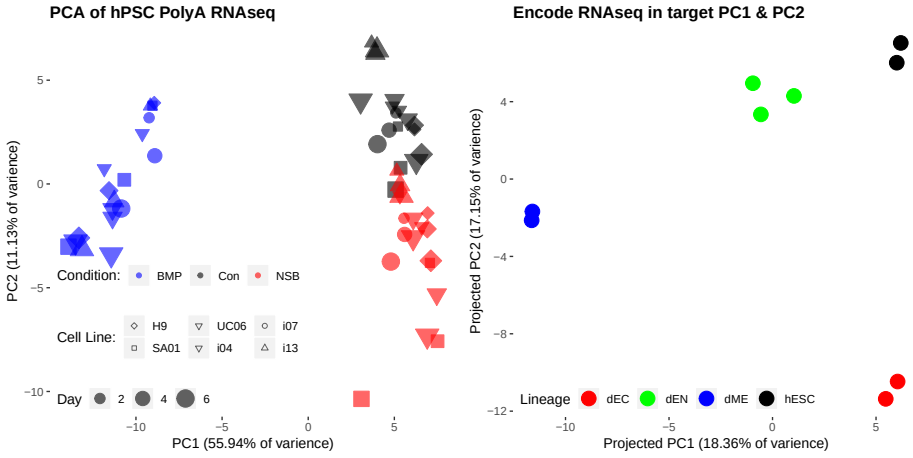
Projection of principle components is achieved by matrix multiplication of a new data set by previously generated eigenvectors, or gene loadings. If the original data were standardized such that each gene is centered to zero average expression level, the principal components are normalized eigenvectors of the covariance matrix of the genes. Each PC is ordered according to how much of the variation present in the data they contain. Projection of the original samples into each PC will maximized the variance of the samples in the direction of that component and uncorrelated to previous components. Projection of new data places the new samples into the PCs defined by the original data. Because the components define an orthonormal basis set, they provide an isomorphism between a vector space, V, and $\Re^n$ which preserves inner products. If V is an inner product space over $\Re$ with orthonormal basis $B = v_1, ..., v_n$ and $v \epsilon V s.t [v]_B = (r_1, ..., r_n)$, then finding the coordinate of v_i in v is precisely the inner product of v with v_i, i.e. $r_i = \langle v, v_i \rangle$. This formulation is implemented for only those genes belonging to both the new data and the PC space. The `projectR` function has S4 method for class `prcomp`.

3.1 Obtaining PCs to project.

```
# data to define PCs
library(projectR)
data(p.RNAseq6l3c3t)

# do PCA on RNAseq6l3c3t expression data
pc.RNAseq6l3c3t<-prcomp(t(p.RNAseq6l3c3t))
pcVAR <- round(((pc.RNAseq6l3c3t$sdev)^2/sum(pc.RNAseq6l3c3t$sdev^2))*100,2)
dPCA <- data.frame(cbind(pc.RNAseq6l3c3t$x,pd.RNAseq6l3c3t))

#plot pca
library(ggplot2)
setCOL <- scale_colour_manual(values = c("blue","black","red"), name="Condition:")
setFILL <- scale_fill_manual(values = c("blue","black","red"),guide = FALSE)
setPCH <- scale_shape_manual(values=c(23,22,25,25,21,24),name="Cell Line:")

pPCA <- ggplot(dPCA, aes(x=PC1, y=PC2, colour=ID.cond, shape=ID.line,
        fill=ID.cond)) +
        geom_point(aes(size=days),alpha=.6)+
        setCOL + setPCH  + setFILL +
        scale_size_area(breaks = c(2,4,6), name="Day") +
        theme(legend.position=c(0,0), legend.justification=c(0,0),
              legend.direction = "horizontal",
              panel.background = element_rect(fill = "white",colour=NA),
              legend.background = element_rect(fill = "transparent",colour=NA),
              plot.title = element_text(vjust = 0,hjust=0,face="bold")) +
        labs(title = "PCA of hPSC PolyA RNAseq",
            x=paste("PC1 (",pcVAR[1],"% of varience)",sep=""),
            y=paste("PC2 (",pcVAR[2],"% of varience)",sep=""))
```

3.2 Projecting prcomp objects

```
# data to project into PCs from RNAseq6   l3c3t expression data
data(p.ESepiGen4c1l)

library(projectR)
PCA2ESepi <- projectR(data = p.ESepiGen4c1l$mRNA.Seq,loadings=pc.RNAseq6l3c3t,
full=TRUE, dataNames=map.ESepiGen4c1l[["GeneSymbols"]])
## [1] "93 row names matched between data and loadings"
## [1] "Updated dimension of data: 93 9"

pd.ESepiGen4c1l<-data.frame(Condition=sapply(colnames(p.ESepiGen4c1l$mRNA.Seq),
  function(x) unlist(strsplit(x,'_'))[1]),stringsAsFactors=FALSE)
pd.ESepiGen4c1l$color<-c(rep("red",2),rep("green",3),rep("blue",2),rep("black",2))
names(pd.ESepiGen4c1l$color)<-pd.ESepiGen4c1l$Cond

dPCA2ESepi<- data.frame(cbind(t(PCA2ESepi[[1]]),pd.ESepiGen4c1l))

#plot pca
library(ggplot2)
setEpiCOL <- scale_colour_manual(values = c("red","green","blue","black"),
  guide = guide_legend(title="Lineage"))

pPC2ESepiGen4c1l <- ggplot(dPCA2ESepi, aes(x=PC1, y=PC2, colour=Condition)) +
  geom_point(size=5) + setEpiCOL +
  theme(legend.position=c(0,0), legend.justification=c(0,0),
  panel.background = element_rect(fill = "white"),
  legend.direction = "horizontal",
  plot.title = element_text(vjust = 0,hjust=0,face="bold")) +
  labs(title = "Encode RNAseq in target PC1 & PC2",
  x=paste("Projected PC1 (",round(PCA2ESepi[[2]][1],2),"% of varience)",sep=""),
  y=paste("Projected PC2 (",round(PCA2ESepi[[2]][2],2),"% of varience)",sep=""))
```



4 NMF projection

NMF decomposes a data matrix of D with N genes as rows and M samples as columns, into two matrices, as D AP. The pattern matrix P has rows associated with BPs in samples and the amplitude matrix A has columns indicating the relative association of a given gene, where the total number of BPs (k) is an input parameter. CoGAPS and GWCoGAPS seek a pattern matrix ($\mathbf{P}$) and the corresponding distribution matrix of weights ($\mathbf{A}$) whose product forms a mock data matrix ($\mathbf{M}$) that represents the gene-wise data $\mathbf{D}$ within noise limits (ε). That is,

$$\mathbf{D} = \mathbf{M} + \varepsilon = \mathbf{AP} + \varepsilon. \quad (1)$$

The number of rows in $\mathbf{P}$ (columns in $\mathbf{A}$) defines the number of biological patterns (k) that CoGAPS/GWCoGAPS will infer from the number of nonorthogonal basis vectors required to span the data space. As in the Bayesian Decomposition algorithm Wang, Kossenkov, and Ochs (2006), the matrices $\mathbf{A}$ and $\mathbf{P}$ in CoGAPS are assumed to have the atomic prior described in Sibisi and Skilling (1997). In the CoGAPS/GWCoGAPS implementation, α_A and α_P are corresponding parameters for the expected number of atoms which map to each matrix element in $\mathbf{A}$ and $\mathbf{P}$, respectively. The corresponding matrices $\mathbf{A}$ and $\mathbf{P}$ are found by MCMC sampling.

Projection of CoGAPS/GWCoGAPS patterns is implemented by solving the factorization in 1 for the new data matrix where $\mathbf{A}$ is the fixed nonorthogonal basis vectors comprising the average of the posterior mean for the CoGAPS/GWCoGAPS simulations performed on the original data. The patterns $\mathbf{P}$ in the new data associated with this amplitude matrix is estimated using the least-squares fit to the new data implemented with the lmFit function in the limma package (cite Limma). The `projectR` function has S4 method for class `Linear Embedding Matrix, LME`.

```
library(projectR)
projectR(data, loadings,dataNames = NULL, loadingsNames = NULL,
     NP = NA, full = FALSE)
```

4.0.1 Input Arguments

The inputs that must be set each time are only the data and patterns, with all other inputs having default values. However, inconguities between gene names–rownames of the loadings object and either rownames of the data object will throw errors and, subsequently, should be checked before running.

The arguments are as follows:

data a target dataset to be projected into the pattern space

loadings a CogapsResult object

dataNames rownames (eg. gene names) of the target dataset, if different from existing rownames of data

loadingsNames loadingsNames rownames (eg. gene names) of the loadings to be matched with dataNames

NP vector of integers indicating which columns of loadings object to use. The default of NP = NA will use entire matrix.

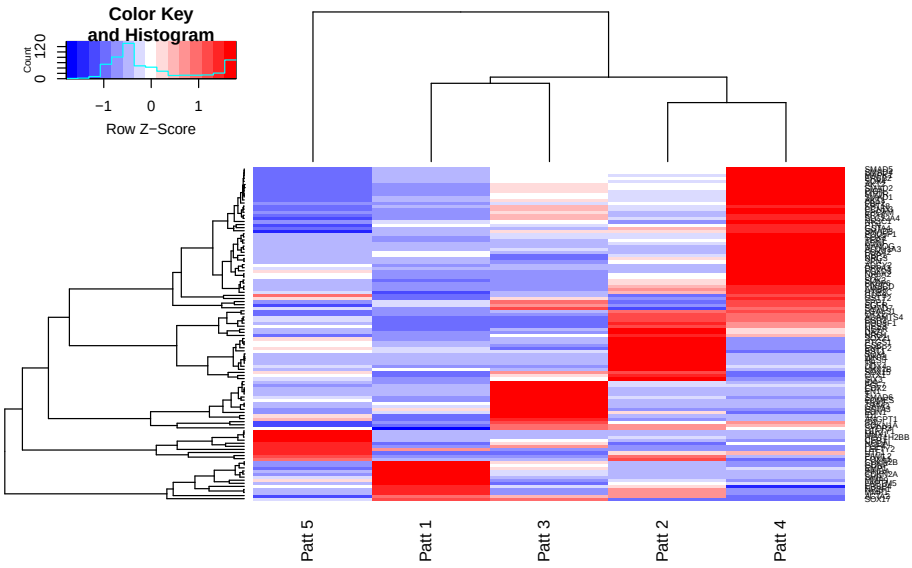
full logical indicating whether to return the full model solution. By default only the new pattern object is returned.

4.0.2 Output

The basic output of the base projectR function, i.e. `full=FALSE`, returns `projectionPatterns` representing relative weights for the samples from the new data in this previously defined feature space, or set of feature spaces. The full output of the base projectR function, i.e. `full=TRUE`, returns `projectionFit`, a list containing `projectionPatterns` and `Projection`. The `Projection` object contains additional information from the proceedure used to obtain the `projectionPatterns`. For the the the base projectR function, `Projection` is the full lmFit model from the package *limma*.

4.1 Obtaining CoGAPS patterns to project.

```
# get data
library(projectR)
AP <- get(data("AP.RNAseq6l3c3t")) #CoGAPS run data
AP <- AP$Amean
# heatmap of gene weights for CoGAPs patterns
library(gplots)
pNMF<-heatmap.2(as.matrix(AP),col=bluered, trace='none',
          distfun=function(c) as.dist(1-cor(t(c))) ,
          cexCol=1,cexRow=.5,scale = "row",
          hclustfun=function(x) hclust(x, method="average")
      )
```



4.2 Projecting CoGAPS objects

```
# data to project into PCs from RNAseq6l3c3t expression data
library(projectR)
```

```
data('p.ESepiGen4c1l4')
## Warning in data("p.ESepiGen4c1l4"): data set 'p.ESepiGen4c1l4' not found
data('p.RNAseq6l3c3t')

NMF2ESepi <- projectR(p.ESepiGen4c1l$mRNA.Seq,loadings=AP,full=TRUE,
    dataNames=map.ESepiGen4c1l[["GeneSymbols"]])
## [1] "93 row names matched between data and loadings"
## [1] "Updated dimension of data: 93 9"

dNMF2ESepi<- data.frame(cbind(t(NMF2ESepi),pd.ESepiGen4c1l))

#plot pca
library(ggplot2)
setEpiCOL <- scale_colour_manual(values = c("red","green","blue","black"),
guide = guide_legend(title="Lineage"))

pNMF2ESepiGen4c1l <- ggplot(dNMF2ESepi, aes(x=X1, y=X2, colour=Condition)) +
  geom_point(size=5) + setEpiCOL +
  theme(legend.position=c(0,0), legend.justification=c(0,0),
  panel.background = element_rect(fill = "white"),
  legend.direction = "horizontal",
  plot.title = element_text(vjust = 0,hjust=0,face="bold"))
  labs(title = "Encode RNAseq in target PC1 & PC2",
  x=paste("Projected PC1 (",round(PCA2ESepi[[2]][1],2),"% of varience)",sep=""),
  y=paste("Projected PC2 (",round(PCA2ESepi[[2]][2],2),"% of varience)",sep=""))
## $x
## [1] "Projected PC1 (18.36% of varience)"
##
## $y
## [1] "Projected PC2 (17.15% of varience)"
##
## $title
## [1] "Encode RNAseq in target PC1 & PC2"
##
## attr(,"class")
## [1] "labels"
```

5 Clustering projection

As cannonical projection is not defined for clustering objects, the projectR package offers two transfer learning inspired methods to achieve the "projection" of clustering objects. These methods are defined by the function used to quantify and transfer the relationships which define each cluster in the original data set to the new dataset. Briefly, `cluster2pattern` uses the corelation of each genes expression to the mean of each cluster to define continuous weights. These weights are output as a pclust object which can serve as input to `projectR`. Alternatively, the `intersectoR` function can be used to test for significant overlap between two clustering objects. Both `cluster2pattern` and `intersectoR` methods are coded for a generic list structure with additional S4 class methods for kmeans and hclust objects. Further details and examples are provided in the followin respecitive sections.

5.1 cluster2pattern

`cluster2pattern` uses the corelation of each genes expression to the mean of each cluster to define continuous weights.

```
library(projectR)
cluster2pattern(clusters = NA, NP = NA, Data = NA)
```

5.1.1 Input Arguments

The inputs that must be set each time are the clusters and data.

The arguments are as follows:

clusters a clustering object

NP either the number of clusters desired or the subset of clusters to use

data data used to make clusters object

5.1.2 Output

The output of the `cluster2pattern` function is a `pclust` class object; specifically, a matrix of genes (rows) by clusters (columns). A gene's value outside of its assigned cluster is zero. For the cluster containing a given gene, the gene's value is the corelation of the gene's expression to the mean of that cluster.

5.2 intersectoR

`intersectoR` function can be used to test for significant overlap between two clustering objects. The base function finds and tests the intersecting values of two sets of lists, presumably the genes associated with patterns in two different datasets. S4 class methods for hclust and kmeans objects are also available.

```
library(projectR)
intersectoR(pSet1 = NA, pSet2 = NA, pval = 0.05, full = FALSE, k = NULL)
```

5.2.1 Input Arguments

The inputs that must be set each time are the clusters and data.

The arguments are as follows:

pSet1 a list for a set of patterns where each entry is a set of genes associated with a single pattern

pSet2 a list for a second set of patterns where each entry is a set of genes associated with a single pattern

pval the maximum p-value considered significant

full logical indicating whether to return full data frame of signigicantly overlapping sets. Default is false will return summary matrix.

k numeric giving cut height for hclust objects, if vector arguments will be applied to pSet1 and pSet2 in that order

5.2.2 Output

The output of the `intersectoR` function is a summary matrix showing the sets with stastically significant overlap under the specified p-val threshold based on a hypergeometric test. If `full==TRUE` the full data frame of signigicantly overlapping sets will also be returned.

6 Correlation based projection

Correlation based projection requires a matrix of gene-wise correlation values to serve as the Pattern input to the `projectR` function. This matrix can be user-generated or the result of the `correlateR` function included in the projectR package. User-generated matrixes with each row corresponding to an individual gene can be input to the generic `projectR` function. The `correlateR` function allows users to create a weight matrix for projection with values quantifying the within dataset correlation of each genes expression to the expression pattern of a particular gene or set of genes as follows.

6.1 correlateR

```
library(projectR)
correlateR(genes = NA, dat = NA, threshtype = "R", threshold = 0.7, absR = FALSE, ...)
```

6.1.1 Input Arguments

The inputs that must be set each time are only the genes and data, with all other inputs having default values.

The arguments are as follows:

genes gene or character vector of genes for reference expression pattern dat

data matrix or data frame with genes to be used for to calculate correlation

threshtype Default "R" indicates thresholding by R value or equivalent. Alternatively, "N" indicates a numerical cut off.

threshold numeric indicating value at which to make threshold

absR logical indicating where to include both positive and negatively correlated genes

... addtion imputes to the cor function

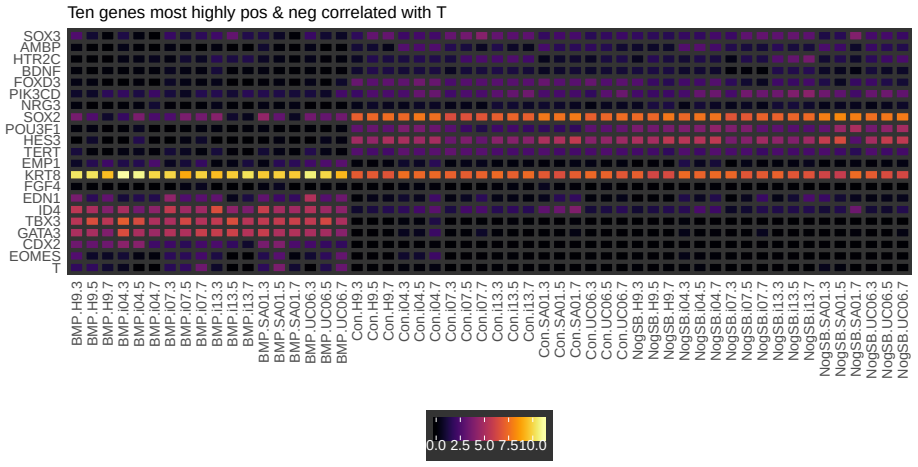
6.1.2 Output

The output of the `correlateR` function is a `correlateR` class object. Specifically, a matrix of correlation values for those genes whose expression pattern pattern in the dataset is correlated (and anti-correlated if absR=TRUE) above the value given in as the threshold arguement. As this information may be useful in its own right, it is recommended that users inspect the `correlateR` object before using it as input to the `projectR` function.

6.2 Obtaining and visualizing `correlateR` objects.

```
# data to
library(projectR)
```

```
data("p.RNAseq6l3c3t")

# get genes correlated to T
cor2T<-correlateR(genes="T", dat=p.RNAseq6l3c3t, threshtype="N", threshold=10, absR=TRUE)
cor2T <- cor2T@corM
### heatmap of genes more correlated to T
indx<-unlist(sapply(cor2T,rownames))
colnames(p.RNAseq6l3c3t)<-pd.RNAseq6l3c3t$sampleX
library(reshape2)
pm.RNAseq6l3c3t<-melt(cbind(p.RNAseq6l3c3t[indx,],indx))
## Using indx as id variables

library(gplots)
library(ggplot2)
library(viridis)
## Loading required package: viridisLite
pCorT<-ggplot(pm.RNAseq6l3c3t, aes(variable, indx, fill = value)) +
  geom_tile(colour="gray20", size=1.5, stat="identity") +
  scale_fill_viridis(option="B") +
  xlab("") +  ylab("") +
  scale_y_discrete(limits=indx) +
  ggtitle("Ten genes most highly pos & neg correlated with T") +
  theme(
    panel.background = element_rect(fill="gray20"),
    panel.border = element_rect(fill=NA,color="gray20", size=0.5, linetype="solid"),
    panel.grid.major = element_blank(),
    panel.grid.minor = element_blank(),
    axis.line = element_blank(),
    axis.ticks = element_blank(),
    axis.text = element_text(size=rel(1),hjust=1),
    axis.text.x = element_text(angle = 90,vjust=.5),
    legend.text = element_text(color="white", size=rel(1)),
    legend.background = element_rect(fill="gray20"),
    legend.position = "bottom",
    legend.title=element_blank()
)
```



6.3 Projecting correlateR objects.

```
# data to project into from RNAseq6l3c3t expression data
data(ESepiGen4c1l4)

library(projectR)
cor2ESepi <- projectR(p.ESepiGen4c1l$mRNA.Seq,loadings=cor2T[[1]],full=FALSE,
    AnnotationObj=map.ESepiGen4c1l, IDcol="GeneSymbols")
```

Barbakh, Wesam Ashour, Ying Wu, and Colin Fyfe. 2009. "Review of Linear Projection Methods." In *Non-Standard Parameter Adaptation for Exploratory Data Analysis*, 29–48. Berlin, Heidelberg: Springer Berlin Heidelberg.

Sibisi, Sibusiso, and John Skilling. 1997. "Prior Distributions on Measure Space." *Journal of the Royal Statistical Society: Series B (Statistical Methodology)* 59 (1): 217–35. doi:10.1111/1467-9868.00065.

Wang, Guoli, Andrew V. Kossenkov, and Michael F. Ochs. 2006. "LS-Nmf: A Modified Non-Negative Matrix Factorization Algorithm Utilizing Uncertainty Estimates." *BMC Bioinformatics* 7 (1): 175. doi:10.1186/1471-2105-7-175.