PISCES tutorial

### Introduction

The pipeline for Protein Activity Inference in Single Cells (PISCES) is a regulatory-network-based methdology for the analysis of single cell gene expression profiles.

PISCES transforms highly variable and noisy single cell gene expression profiles into robust and reproducible protein activity profiles. PISCES is centered around two key algorithms: the Algorithm for the Reconstruction of Accurate Cellular Networks ARACNe [1]; and the algorithm for Virtual Inference of Protein-activity by Enriched Regulon analysis (VIPER/metaVIPER) [2,3].

Briefly, the ARACNe algorithm is one of the most widely used methods for inferring transcriptional interactions from gene expression data. The VIPER algorithm uses the expression of the ARACNe-inferred regulatory targets of a given protein, such as the targets of a transcription factor (TF), as an accurate reporter of its activity. Typically, PISCES can accurately assess the activity of up to 6000 regulatory proteins from single cell gene expression profiles, significantly increasing the ability to analyze the biological function and relevance of gene products whose mRNAs are undetectable in individual cells (e.g. dropout effect).

### Setup

To run this pipeline, you’ll need to have the following packages installed:

* viper (v1.18.1)
* cluster
* ggplot2
* ggpubr
* umap
* pheatmap
* RColorBrewer
* Matrix
* biomaRt
* psych
* MUDAN

**NOTE:** This tutorial assumes the working directoy is set to the folder containing the PISCES repository. All the data generated by PISCES will be saved in the same directory. This is not recommended for practical use, and can be changed by specifying full paths when loading or saving in your own applications.

#### Getting started

Start by loading in the PISCES functions as well as the provided test data. Note that this pipeline uses ENSMBL Gene IDs by default:

source('functions/process-utils.R')  
source('functions/cluster-functions.R')  
source('functions/viper-utils.R')  
library(ggplot2)  
library(ggpubr)  
library(viper)  
library(pheatmap)  
library(RColorBrewer)  
library(MUDAN)  
raw.mat <- readRDS('tutorial/pbmc.rds')

Alternatively, If your data are in 10x format, you can load the data using the follows (*you don’t need to run this for the purpose of the tutorial*):

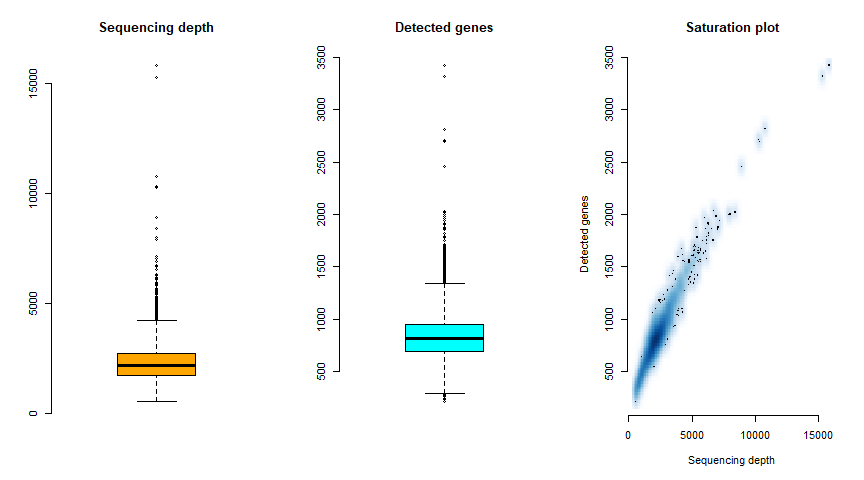
library(Matrix)  
raw.mat <- as.matrix(readMM('data/matrix.mtx'))  
genes <- read.table('data/genes.tsv', sep = '\t')  
barcodes <- read.table('data/barcodes.tsv', sep = '\t')  
colnames(raw.mat) <- barcodes[,1]  
rownames(raw.mat) <- genes[,1]

We recommend saving intermediate data at each step, since many of these steps will take a considerable amount of time. These saving steps are not including in this tutorial, but can be achieved with the saveRDS function in R.

### PreProcessing

First, we perform some cursory QC on the data to check the distribution of read depth and genes detected in each sample:

QCPlots(raw.mat)



In the preprocessing step we remove all the genes with no expression (zero reads across all cells) and low quality cells. Cells with less than 1000 UMIs or more than 100000 UMIs can be considered low quality cells. These thresholds can be adjusted using the arguments of the QCTransform function based on your data. We also recommened to check the percentage of counts associated with mithocondrial genes on a cell by cell basis as, usually, cells with high level micthocondrial genes are considered low quality/dying cells.

Once the data is filtered, we will apply a CPM normalization, then generate a gene expression signature using a double rank transformation:

filt.mat <- QCTransform(raw.mat)

## [1] "Removed 16104 genes and 153 cells."

cpm.mat <- CPMTransform(filt.mat)  
rank.mat <- RankTransform(cpm.mat)

By default, the gene expression signature is generated by performing a “double rank” transformation, which uses the median gene expression of the data set as an internal reference to compute a gene expression signature on a cell by cell basis.

### 1.1 Single Cell Network Generation

**NOTE:** Because ARACNe takes a considerable amount of time to run, we recommend setting up a cluster-based implementation. For ease-of-use in this tutorial, we have included the networks generated in this analysis within the tutorial. For a detailed tutorial on how to use ARACNe-AP, consult the github here: <https://github.com/califano-lab/ARACNe-AP/blob/master/README.md>

The default PISCES approach assumes that there are no known cell type annotation in the data set. PISCES will generate an ARACNe network from all the cells. If different cell types in the data set are experimetally defined (e.g. by FACs) and annotated, cell-type specific networks can be generated based on the cell type annotation. However, we recommend proceeding in an unsupervised manner, as the unsupervised analysis can confirm the experimental design and, potentially, generate novel biological findings.

The data must first be saved in a format that is compatible with the Java based ARACNe-AP implementation included in this pipeline:

ARACNeTable(cpm.mat, 'tutorial/pbmc-cpm.tsv')

ARACNe should be run independently for each regulator set (TFs, COTFs, and Signaling Proteins). The files containing the lists of the candidate master regulator proteins can be found in the *Modules/ARACNe/* directory for both mouse and human data. The newtwork files generated by ARACNE (.tsv’s) for each set of regulators should be merged in single file (this can be easily done through command line). Finally, the merged .tsv’s and the gene expression matrix from which the network was inferred are combined by VIPER to generate a regulon object with the following command:

### 1.2 Clustering

Once the ARACNe network has been generated, we can infer protein activity as following:

r1.net <- readRDS('tutorial/pbmc\_r1-net-pruned.rds')  
r1.pAct <- viper(rank.mat, r1.net, method = 'none')

Single-cell ARACNe networks usually contain fewer regulons than those generated from bulk data due to data sparsity that characterize single cell gene expresion data. However, it is possible to combine single-cell networks with precomputed GTEx bulk networks using the metVIPER algorithm. Briefly, MetaVIPER will identify the GTEX networks that have the best macth with the single cell data, then will use these networks to compute the protein activity only for the proteins whose regulon was not inferred from single cell gene expression profiles (*r1.pAct* object).

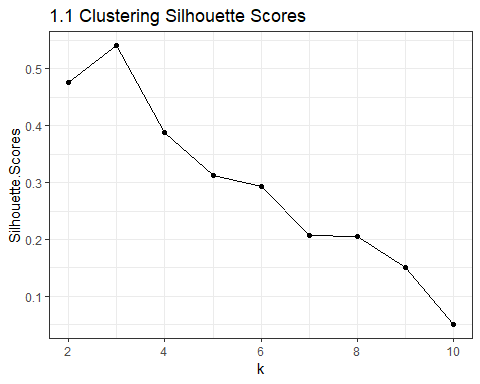
**NOTE:** for the purposes of this pipeline, this is an optional step that you do not have to run. Because this step generates VIPER matrices from more than 30 networks, it will take a considerable amount of time and memory to run. We recommend running this step on a cluster system with 32GB of memory or more allocated for any data set with 1000 or more cells. Finally, this step is not relevant for murine data, as the GTEx networks are generated from human RNAseq.

Once the protein activity has been computed, clustering analysis can be performed on the protein activity-based distance metrix. This pipeline uses *viperSimilarity* as a distance metric and the *PAM* algorithm for clustering, but other methods such as K-Means or louvain can be applied:

r1.viperDist <- as.dist(viperSimilarity(r1.pAct))  
r1.clusts <- PamKRange(r1.viperDist, kmin = 2, kmax = 10)

## [1] "Clustering with k=2..."  
## [1] "Clustering with k=3..."  
## [1] "Clustering with k=4..."  
## [1] "Clustering with k=5..."  
## [1] "Clustering with k=6..."  
## [1] "Clustering with k=7..."  
## [1] "Clustering with k=8..."  
## [1] "Clustering with k=9..."  
## [1] "Clustering with k=10..."

r1.clustSil <- SilScoreEval(r1.clusts, r1.viperDist)  
plot.dat <- data.frame('k' = 2:10, 'Silhouette.Scores' = r1.clustSil)  
ggplot(plot.dat, aes(x = k, y = Silhouette.Scores)) + geom\_point() + geom\_line() +  
 ggtitle('1.1 Clustering Silhouette Scores') + theme\_bw()



This will generate a set of clusters for values of *k* between 2 and 5, then generate a vector of average silhouette scores that indicate cluster quality. A plot of the silhouette scores can be automatically saved by using the optional *plotPath* argument with the *SilScoreEval* function. For the data set used in this tutorial, the optimal clustering occured with *k=3*, as defined by the maximum silhouette score. A silhouette score of 0.25 or above is generalyl considered robust [4, 5]. (Note that in other analyses, the optimal number of clusters will be different.) This clusters will now be used to generate cluster-specific ARACNe networks.

In order to improve the quality of the ARACNe networks, PISCES can transform single cell gene expression profiles into metacell profiles. The PISCES meta-cell inference algorithm aims at overcoming the sparse nature of single cell data due to dropouts (inefficient mRNA capture) by integrating the expression profiles of cells with similar protein activity profiles within each cluster. This is accomplished by generating a K-nearest neighbor (KNN) graph based on the *viperSimilarity* distance method, then integrating the counts of the K nearest cells. Metacells generated within each cluster are subset to 200 to avoid significant overlap. All of these steps can be accomplished with the following command (if you are interested in running these steps seperately in your analysis, there are also stand alone commands for each step):

r1.clustMats <- MakeCMfA(filt.mat, r1.viperDist, clustering = r1.clusts$k3, out.dir = 'tutorial/', out.name = 'pbmc-r1-clusts')

### 2.1 Cluster-specificnetwork generation

**NOTE:** NOTE: As in the **1.1 Network Generation** step, we have included the ARACNe networks generated in this step within the tutorial.

The procedure for this step is the same as in **1.1 Network Generation**, but must be repeated for each cluster generated in the **1.1 Clustering** step.

### 2.2 Protein activity based clustering analysis

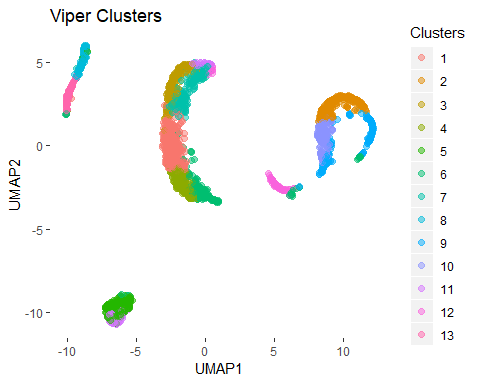
Cluster-specific networks can now be used to infer a protein activity more accurately:

# load in networks  
c1.net <- readRDS('tutorial/pbmc-r2-c1\_pruned.rds')  
c2.net <- readRDS('tutorial/pbmc-r2-c2\_pruned.rds')  
c3.net <- readRDS('tutorial/pbmc-r2-c3\_pruned.rds')  
# infer protein activity  
r2.pAct <- viper(rank.mat, list('c1' = c1.net, 'c2' = c2.net, 'c3' = c3.net), method = 'none')

Next, the most differentially activated proteins within each cell are identified. These cell specific lists are then merged to identify a set of features that best differentiate the data set. These features can then be used to cluster the data using the louvain method:

r2.cbcMRs <- CBCMRs(r2.pAct)  
r2.pAct.cbc <- r2.pAct[ r2.cbcMRs ,]  
r2.louvain <- LouvainClust(r2.pAct.cbc)

These clusters can be visualized using a Uniform Manifold Aproximation and Projection (UMAP) as a dimensionality reduction:



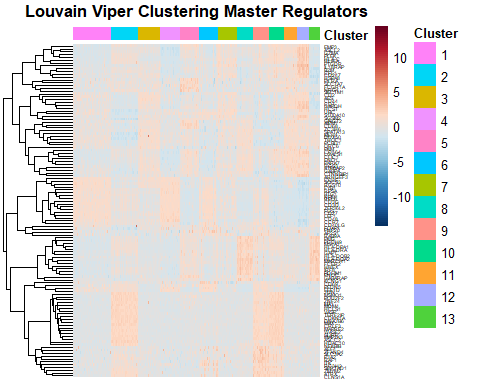
### 3 Identifying Master Regulators

Cluster specific master regulators can now be identified using a bootsrapped T-test. Each cluster is compared against the rest of the data to identify the proteins most specifc to that cluster. Additionally, the data set is converted from ENSG to Gene Names for ease of interpretation:

r2.pAct <- Ensemble2GeneName(r2.pAct)  
cluster.MRs <- BTTestMRs(r2.pAct, r2.louvain)

These master regulators can then be visualized in a heatmap:

ClusterHeatmap(r2.pAct[ MR\_UnWrap(r2.MRs, top = 10) , ], clust = r2.louvain, plotTitle = 'Louvain Viper Clustering Master Regulators')



#### Acknowledgements

Jeremy Dooley - for his advice and expertise in single cell sequencing experiments.  
Hongxu Ding - whose work in the Califano laid the groundwork for the development of this pipeline.  
Evan Paull - for help with software and tutorial development and testing.