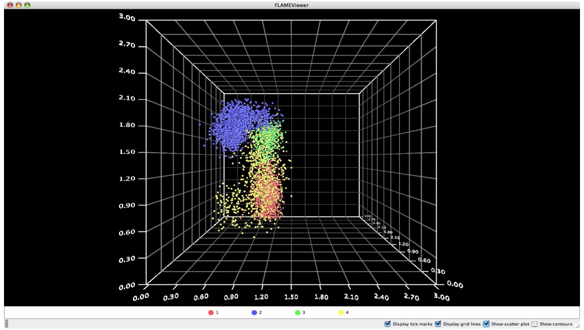
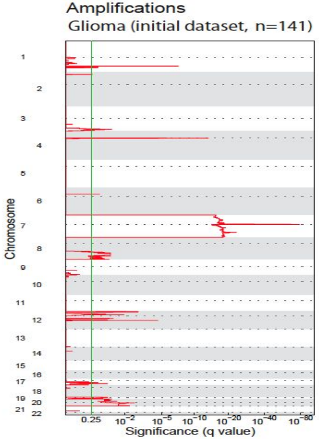
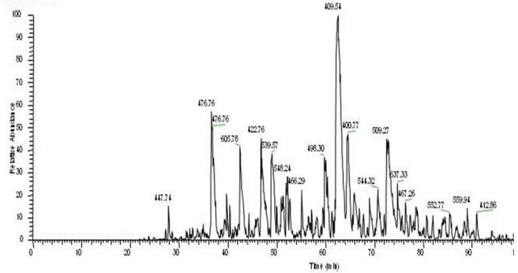
Other GenePattern Features


FLAMEViewer
3.00
2.70
2.40
2.10
1.80
1.50
1.20
0.90
0.60
0.30
0.00
● 1
● 2
● 3
● 4
☑ Display tick marks ☑ Display grid lines ☑ Show scatter plot □ Show contours

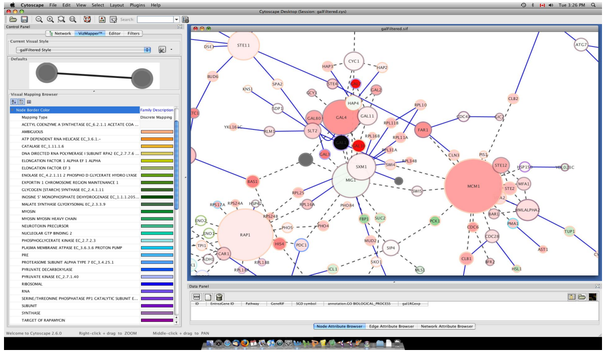
Flow Cytometry



Sequence Variation Analysis

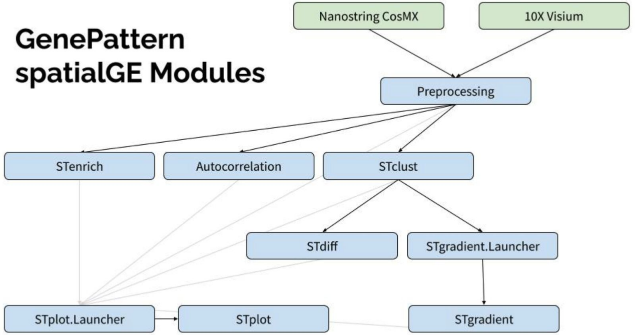


Proteomics



Network Analysis

GenePattern spatialGE Modules



Nanostring CosMX

10X Visium

Preprocessing

STenrich

Autocorrelation

STclust

STdiff

STgradient.Launcher

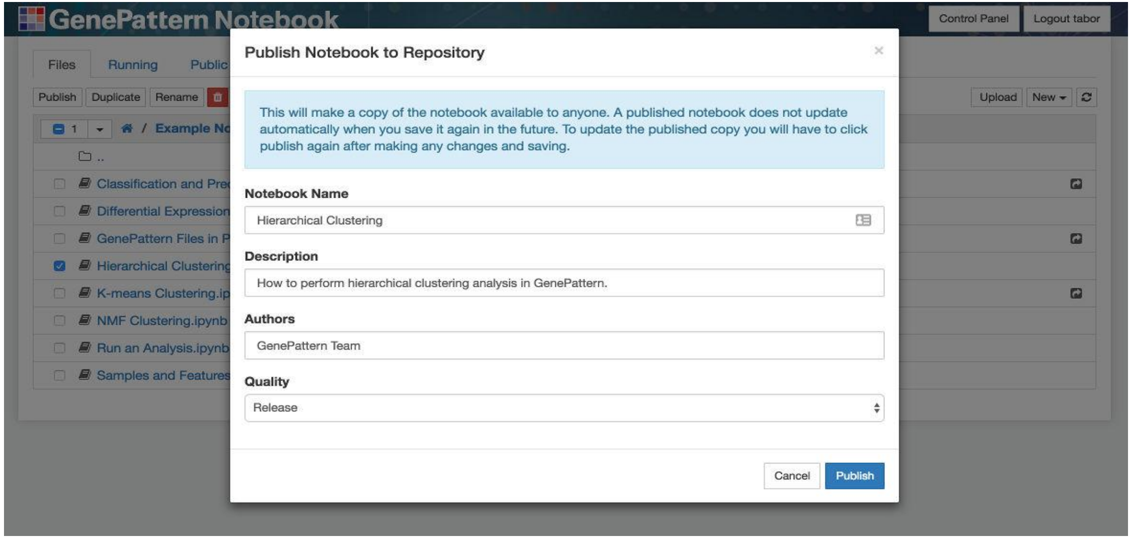
STplot.Launcher

STplot

STgradient

Publish Your Notebooks

- Publish your own notebooks on the g2nb workspace.
- Share notebooks with others.



Publish Notebook to Repository

This will make a copy of the notebook available to anyone. A published notebook does not update automatically when you save it again in the future. To update the published copy you will have to click publish again after making any changes and saving.

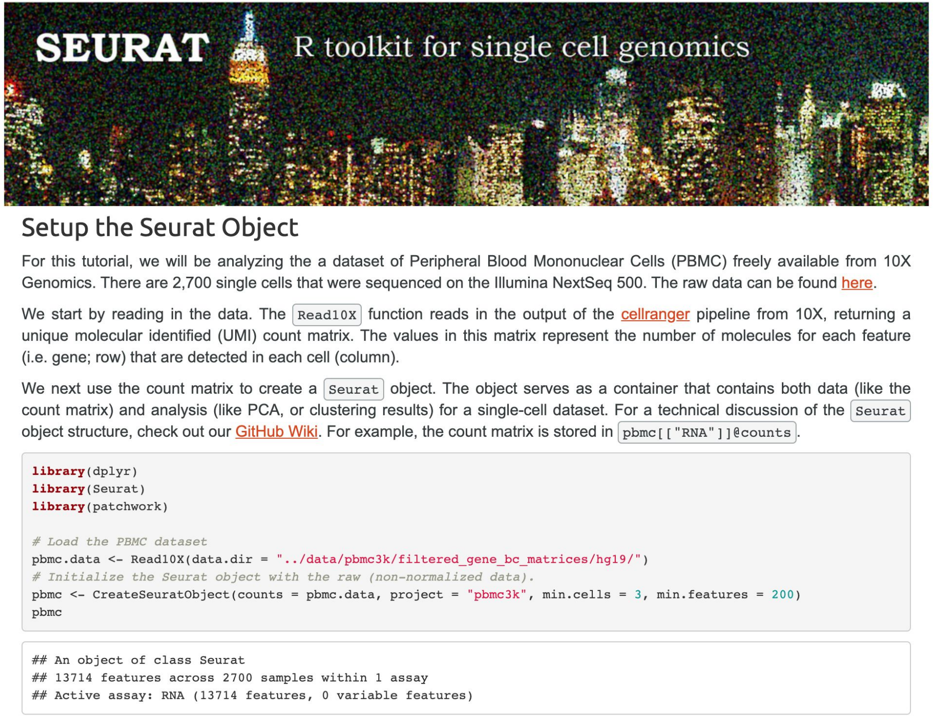
Notebook Name
Hierarchical Clustering

Description
How to perform hierarchical clustering analysis in GenePattern.

Authors
GenePattern Team

Quality
Release

The Seurat GenePattern Notebook



Setup the Seurat Object

For this tutorial, we will be analyzing the a dataset of Peripheral Blood Mononuclear Cells (PBMC) freely available from 10X Genomics. There are 2,700 single cells that were sequenced on the Illumina NextSeq 500. The raw data can be found here.

We start by reading in the data. The `Read10X` function reads in the output of the cellranger pipeline from 10X, returning a unique molecular identified (UMI) count matrix. The values in this matrix represent the number of molecules for each feature (i.e. gene; row) that are detected in each cell (column).

We next use the count matrix to create a `Seurat` object. The object serves as a container that contains both data (like the count matrix) and analysis (like PCA, or clustering results) for a single-cell dataset. For a technical discussion of the `Seurat` object structure, check out our GitHub Wiki. For example, the count matrix is stored in `pbmc[["RNA"]]@counts`.

```r
library(dplyr)
library(Seurat)
library(patchwork)

# Load the PBMC dataset
pbmc.data <- Read10X(data.dir = "../data/pbmc3k/filtered_gene_bc_matrices/hg19/")
# Initialize the Seurat object with the raw (non-normalized data).
pbmc <- CreateSeuratObject(counts = pbmc.data, project = "pbmc3k", min.cells = 3, min.features = 200)
pbmc
```

```
## An object of class Seurat
## 13714 features across 2700 samples within 1 assay
## Active assay: RNA (13714 features, 0 variable features)
```

https://satijalab.org/seurat/v3.1/pbmc3k_tutorial.html

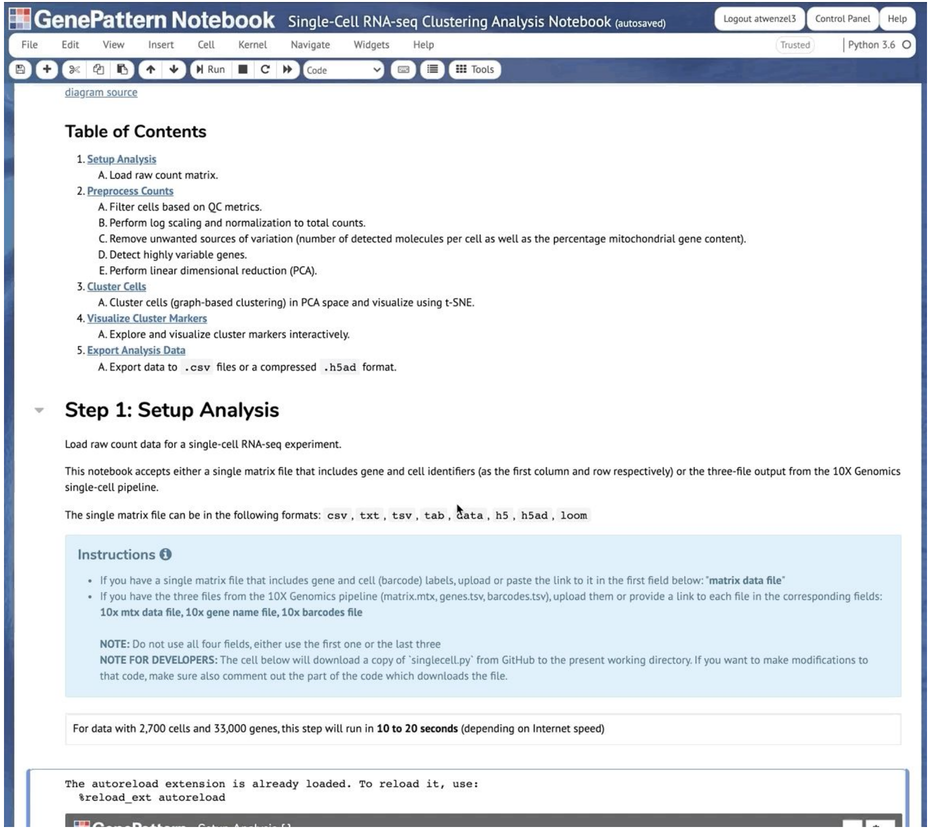


Table of Contents

1. Setup Analysis
   A. Load raw count matrix.
2. Preprocess Counts
   A. Filter cells based on QC metrics.
   B. Perform log scaling and normalization to total counts.
   C. Remove unwanted sources of variation (number of detected molecules per cell as well as the percentage mitochondrial gene content).
   D. Detect highly variable genes.
   E. Perform linear dimensional reduction (PCA).
3. Cluster Cells
   A. Cluster cells (graph-based clustering) in PCA space and visualize using t-SNE.
4. Visualize Cluster Markers
   A. Explore and visualize cluster markers interactively.
5. Export Analysis Data
   A. Export data to `.csv` files or a compressed `.h5ad` format.

Step 1: Setup Analysis

Load raw count data for a single-cell RNA-seq experiment.

This notebook accepts either a single matrix file that includes gene and cell identifiers (as the first column and row respectively) or the three-file output from the 10X Genomics single-cell pipeline.

The single matrix file can be in the following formats: `csv` , `txt` , `tsv` , `tab` , `data` , `h5` , `h5ad` , `loom`

Instructions

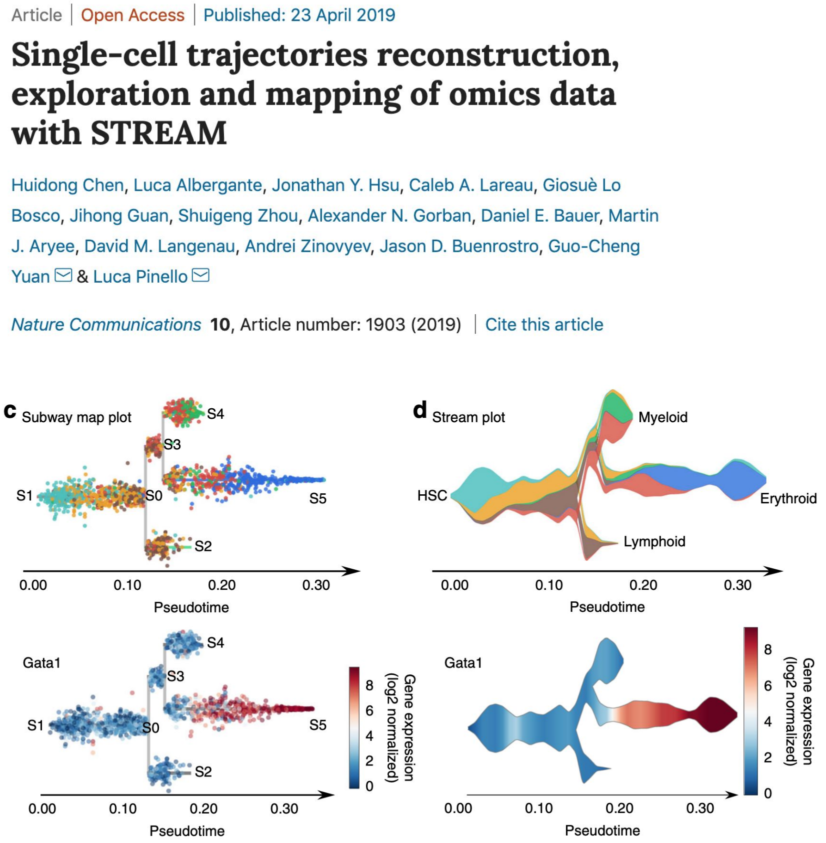
- If you have a single matrix file that includes gene and cell (barcode) labels, upload or paste the link to it in the first field below: "**matrix data file**"
- If you have the three files from the 10X Genomics pipeline (matrix.mtx, genes.tsv, barcodes.tsv), upload them or provide a link to each file in the corresponding fields: **10x mtx data file, 10x gene name file, 10x barcodes file**

  NOTE: Do not use all four fields, either use the first one or the last three
  NOTE FOR DEVELOPERS: The cell below will download a copy of `singlecell.py` from GitHub to the present working directory. If you want to make modifications to that code, make sure also comment out the part of the code which downloads the file.

For data with 2,700 cells and 33,000 genes, this step will run in **10 to 20 seconds** (depending on Internet speed)

```
The autoreload extension is already loaded. To reload it, use:
  %reload_ext autoreload
```

The STREAM Module and Notebook

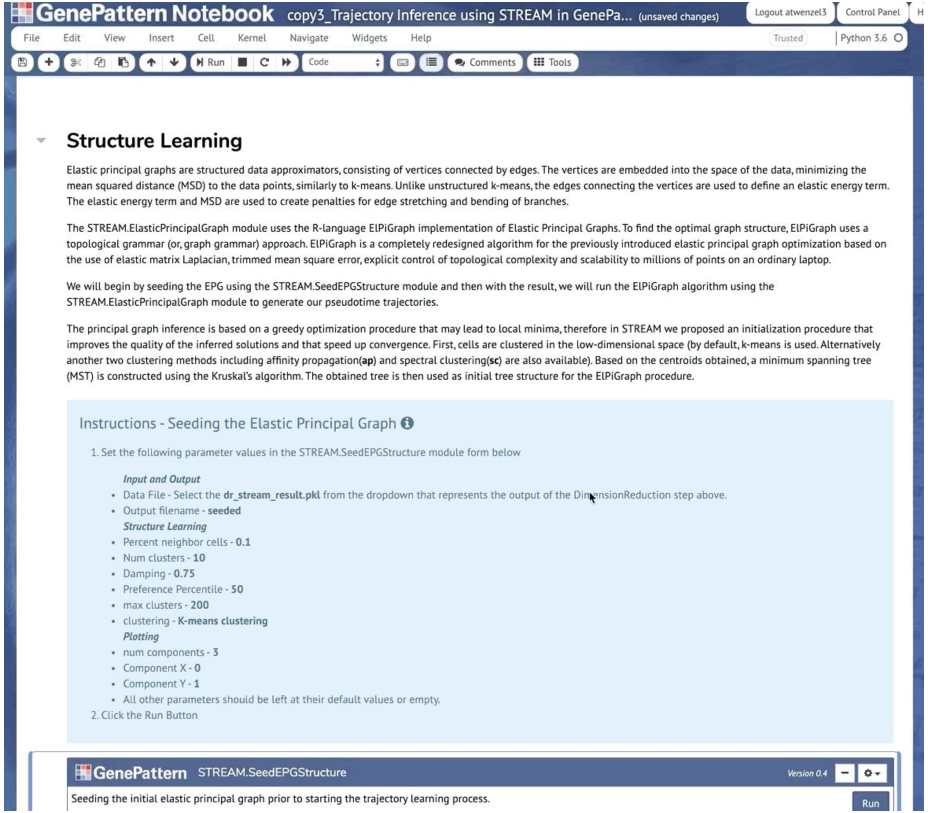


Article | Open Access | Published: 23 April 2019

Single-cell trajectories reconstruction, exploration and mapping of omics data with STREAM

Huidong Chen, Luca Albergante, Jonathan Y. Hsu, Caleb A. Lareau, Giosuè Lo Bosco, Jihong Guan, Shuigeng Zhou, Alexander N. Gorban, Daniel E. Bauer, Martin J. Aryee, David M. Langenau, Andrei Zinovyev, Jason D. Buenrostro, Guo-Cheng Yuan ✉ & Luca Pinello ✉

Nature Communications **10**, Article number: 1903 (2019) | Cite this article



Structure Learning

Elastic principal graphs are structured data approximators, consisting of vertices connected by edges. The vertices are embedded into the space of the data, minimizing the mean squared distance (MSD) to the data points, similarly to k-means. Unlike unstructured k-means, the edges connecting the vertices are used to define an elastic energy term. The elastic energy term and MSD are used to create penalties for edge stretching and bending of branches.

The STREAM.ElasticPrincipalGraph module uses the R-language ElPiGraph implementation of Elastic Principal Graphs. To find the optimal graph structure, ElPiGraph uses a topological grammar (or, graph grammar) approach. ElPiGraph is a completely redesigned algorithm for the previously introduced elastic principal graph optimization based on the use of elastic matrix Laplacian, trimmed mean square error, explicit control of topological complexity and scalability to millions of points on an ordinary laptop.

We will begin by seeding the EPG using the STREAM.SeedEPGStructure module and then with the result, we will run the ElPiGraph algorithm using the STREAM.ElasticPrincipalGraph module to generate our pseudotime trajectories.

The principal graph inference is based on a greedy optimization procedure that may lead to local minima, therefore in STREAM we proposed an initialization procedure that improves the quality of the inferred solutions and that speed up convergence. First, cells are clustered in the low-dimensional space (by default, k-means is used. Alternatively another two clustering methods including affinity propagation(**ap**) and spectral clustering(**sc**) are also available). Based on the centroids obtained, a minimum spanning tree (MST) is constructed using the Kruskal's algorithm. The obtained tree is then used as initial tree structure for the ElPiGraph procedure.

Instructions - Seeding the Elastic Principal Graph

1. Set the following parameter values in the STREAM.SeedEPGStructure module form below

   Input and Output
   - Data File - Select the **dr_stream_result.pkl** from the dropdown that represents the output of the DimensionReduction step above.
   - Output filename - **seeded**

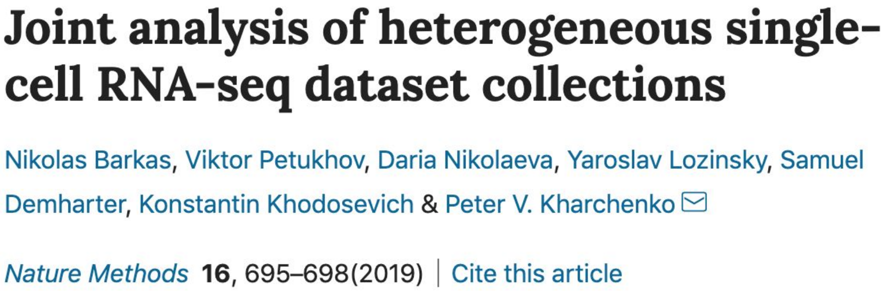
   Structure Learning
   - Percent neighbor cells - **0.1**
   - Num clusters - **10**
   - Damping - **0.75**
   - Preference Percentile - **50**
   - max clusters - **200**
   - clustering - **K-means clustering**

   Plotting
   - num components - **3**
   - Component X - **0**
   - Component Y - **1**
   - All other parameters should be left at their default values or empty.
2. Click the Run Button

GenePattern STREAM.SeedEPGStructure *Version 0.4*

Seeding the initial elastic principal graph prior to starting the trajectory learning process.

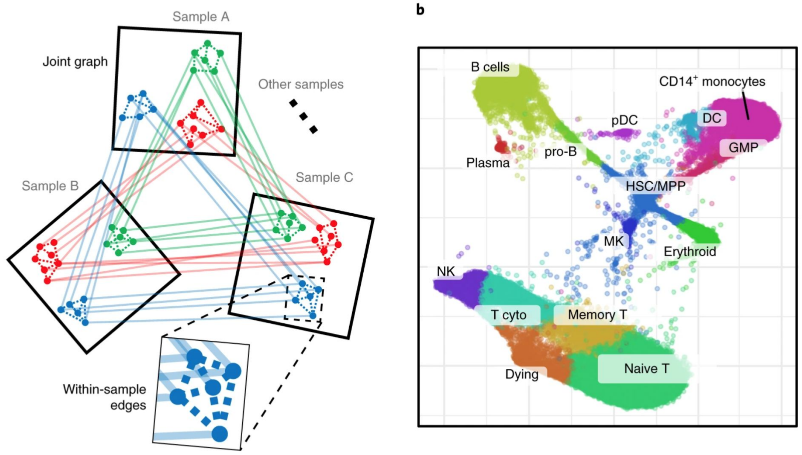
The CONOS Module and notebook

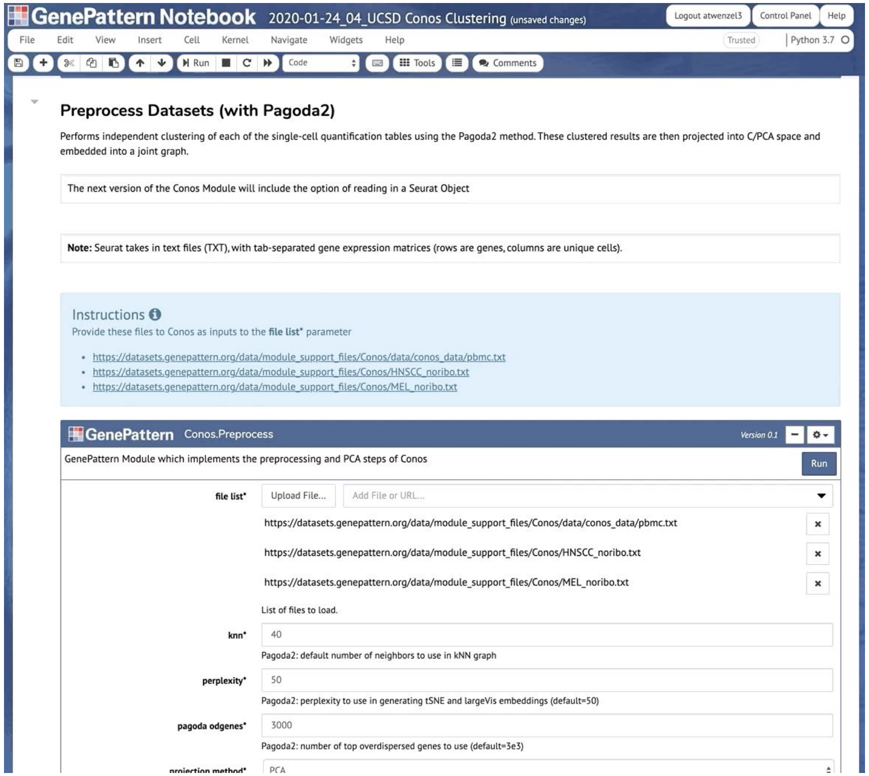


Joint analysis of heterogeneous single-cell RNA-seq dataset collections

Nikolas Barkas, Viktor Petukhov, Daria Nikolaeva, Yaroslav Lozinsky, Samuel Demharter, Konstantin Khodosevich & Peter V. Kharchenko

Nature Methods **16**, 695–698(2019) | Cite this article





2020-01-24_04_UCSD Conos Clustering (unsaved changes)

Preprocess Datasets (with Pagoda2)

Performs independent clustering of each of the single-cell quantification tables using the Pagoda2 method. These clustered results are then projected into C/PCA space and embedded into a joint graph.

The next version of the Conos Module will include the option of reading in a Seurat Object

Note: Seurat takes in text files (TXT), with tab-separated gene expression matrices (rows are genes, columns are unique cells).

Instructions

Provide these files to Conos as inputs to the **file list*** parameter

- https://datasets.genepattern.org/data/module_support_files/Conos/data/conos_data/pbmc.txt
- https://datasets.genepattern.org/data/module_support_files/Conos/HNSCC_noribo.txt
- https://datasets.genepattern.org/data/module_support_files/Conos/MEL_noribo.txt

Conos.Preprocess

GenePattern Module which implements the preprocessing and PCA steps of Conos

file list*
https://datasets.genepattern.org/data/module_support_files/Conos/data/conos_data/pbmc.txt
https://datasets.genepattern.org/data/module_support_files/Conos/HNSCC_noribo.txt
https://datasets.genepattern.org/data/module_support_files/Conos/MEL_noribo.txt
List of files to load.

knn* 40
Pagoda2: default number of neighbors to use in kNN graph

perplexity* 50
Pagoda2: perplexity to use in generating tSNE and largeVis embeddings (default=50)

pagoda odgenes* 3000
Pagoda2: number of top overdispersed genes to use (default=3e3)

projection method* PCA

GenePattern Python Library

- Control a GenePattern server via Python
- Automatic integration with GenePattern cell data

```
import gp

# Create a GenePattern server proxy instance
gpserver = gp.GPServer('http://localhost:8080/gp','myusername', 'mypassword')

# Obtain GPTask by module name
module = gp.GPTask(gpserver, "PreprocessDataset")

# Load module parameter data
module.param_load()

# Create a job specification
job_spec = module.make_job_spec()

# Upload a file to the server
uploaded_file = gpserver.upload_file("file_name", "/path/to/the/file/file_name")
job_spec.set_parameter("input.filename", uploaded_file.get_url())

# Submit the job to the GenePattern server
job = gpserver.run_job(job_spec)
```

Resources

TOOL	URL
GenePattern	www.genepattern.org
g2nb	www.g2nb.org
IGV	www.igv.org
GSEA and MSigDB	www.gsea-msigdb.org

Keep in touch!

Online forum for feature requests, bug reports, and general help	https://groups.google.com/g/genepattern-help
Mailing list to receive GenePattern news	www.genepattern.org/gp_mail.html
X/Twitter	@GenePattern
Mastodon	genepattern

Our Team

Anthony Castanza
Ted Liefeld
Michael Reich
Alex Wenzel

Jim Robinson
Thorin Tabor
Pablo Tamayo
Helga Thorvaldsdottìr

PI
Jill P.Mesirov

www.mesirovlab.org