**Using the Python package ‘DensitySurf’**

**Kristen Feher**

**27-11-2024**

**Introduction**

If you are unfamiliar with using Python, please read ‘A minimal introduction to Python for R users’ first. It will guide you through setting up Python on your computer, as well as a very brief introduction to importing and using packages.

This package is designed so that it can be used in a few simple lines of codes. There are save methods that save output in a R-friendly format so that it can seamlessly integrate with your usual bioinformatics workflows. This guide will only step through the Python code and the output will be interpreted in a separate guide.

*Code in this font is python code in a .py file or .ipynb Jupyter notebook*.

**Processing one sample**

After navigating to your working directory and opening up a python session, create either a new .py file or .ipynb Jupyter notebook. Start your code like this:

*import densitysurf as ds*

*import scanpy as sc*

*# The data is publicly available 10x Visium data.*

*# Data from other technologies may need a different method to import*

*data=sc.read\_10x\_h5("/Users/a123/Desktop/Human\_breast\_cancer\_blockA\_section1/V1\_Breast\_Cancer\_Block\_A\_Section\_1\_filtered\_feature\_bc\_matrix.h5")*

*data.var\_names\_make\_unique()*

*data = data.to\_df()*

Now we have imported a count matrix, which must be a pandas DataFrame where the rows are cells/bins and the columns are genes. The row labels (data.index) must be unique cell identifiers, and the column labels (data.columns) much be unique gene identifiers. Next, there are 3 or 4 steps (depending on whether the data is spatial or not): Tranform (including a separate UMAP method), Cluster, SpecificityNetwork, NeighbourhoodFlow (spatial data only). The first 3 steps can be performed with the following code:

*# ncomps = 20 so it runs quickly, it most cases it will be higher.*

*Y = ds.Transform(data, ncomps = 20, n\_iter = 100, transform = True, goodness\_of\_fit = True)*

*Y.umap(cell = True, gene = True, ncomp\_cell = 10, ncomp\_gene = 10)*

*C\_cell = ds.Cluster(Y, K\_nn = 5, clus\_steps = 1000, mode = 'cells', similarity\_threshold = 0.2)*

*C\_gene = ds.Cluster(Y, K\_nn = 5, clus\_steps = 1000, mode = 'genes', similarity\_threshold = 0.2)*

*S = ds.SpecificityNetwork(Y, C\_cell, C\_gene, similarity\_threshold = 0.2)*

Most of the arguments shown above are the default setting.

When creating the Transform object, the goodness\_of\_fit argument is used to generate data to choose the number of components in a dimension reduced model. If you have many samples, you will first create a Transform with goodness\_of\_fit = True along with ncomps = 500 for a small number of representative samples first. After saving the results and generating the diagnostic plot, you will probably be able to choose a much lower value of ncomps, e.g. 100, and create a Transform for all samples with goodness\_of\_fit = False. Otherwise, using goodness\_of\_fit = True for many samples with a high ncomps will increase the time needed.

For spatial data, it is necessary to first import the XY coordinates in order to create a NeighbourhoodFlow object. The XY coordinates is a DataFrame with 3 columns. Two columns are the X and Y coordinates, and these need to be named ‘X’ and ‘Y’. Another column is the cell or bin label, and these should match the labels in the input data (data.index). It doesn’t matter what this column is called, so long as it has a name, and this exact name is supplied as an argument of NeighbourhoodFlow.

*import pandas as pd*

*XY=pd.read\_csv("/Users/a123/Desktop/Human\_breast\_cancer\_blockA\_section1/spatial/tissue\_positions\_list.csv", header = None).iloc[:, [0, 4, 5]]*

*XY.columns = ['bin\_id', 'X', 'Y']*

*N = ds.NeighbourhoodFlow(C\_cell, XY = XY, K\_nn = 6, dist\_thres = 300, cell\_join\_id = 'bin\_id')*

In NeighbourhoodFlow, the argument K\_nn is the number of spatial nearest neighbours. For 10x Visium data, it is set to 6 as the bins are on a hexagonal grid. The argument dist\_thres is set to 300 as this is the distance between the bins, and deals with edge effects, i.e. spatial nearest neighbour distances greater than 300 are excluded. For other technologies such as 10x Xenium, you need to set these parameters accordingly.

Saving the output is very easy and to save everything, simply give a path to the save directory. There is the option to selectively only save some of the output. While you can save the output wherever you want, I recommend staying organised. To help with that, you can use the ‘directory\_structure’ function to create a convenient directory structure before saving results. If the directories already exist and are not empty, an error will be raised to stop you accidentally overwriting results.

*ds.directory\_structure(path = "")*

This is the directory structure it will result in:

A screenshot of a computer

Description automatically generated

To save all output, use the following code:

*Y.save('output/transform/')*

*C\_cell.save('output/cells/')*

*C\_gene.save('output/genes/')*

*S.save('output/specificity\_network/')*

*N.save('output/neighbourhood\_flow/')*

The following files will appear in each directory (except for figures, which will be used later):

A screenshot of a cell phone

Description automatically generated

A blue line on a white background

Description automatically generated

A screenshot of a computer

Description automatically generated

A screenshot of a computer

Description automatically generated

A screenshot of a computer

Description automatically generated

**Workflow Function**

To streamline the process further, there is a Workflow function.

It assumes you will do your downstream analysis in R. If you want to continue working in Python, you will have to reload the objects.

This function takes many arguments. The first block of arguments supplies the input data and algorithm parameters. The subsequent block of parameters control the work flow. The default work flow parameters assume you are processing one sample in a fresh working directory, i.e. the standard output directory hasn’t been created yet. The standard output directory will then be automatically created. Running Workflow in the default will create a Transform object, Cluster(mode = ‘cells’) and Cluster(mode = ‘genes’) and SpecificityNetwork object. To create a NeighbourhoodFlow, use the argument neighbourhood\_flow = True. Using transform\_input = False means the data won’t undergo a correspondence analysis transformation prior to singular value decomposition, and so you can apply a custom transformation to your input data prior to using the workflow (e.g. convert to logCPM).

The default behaviour of Workflow is to avoid overwriting pre-existing results. If you want to overwrite results, you will have to explicitly force this using the relevant arguments.

The default behaviour of Workflow is to automatically save all output, and parameters in ‘info.txt’, along with an additional pickle file (.pkl) of each object. Saving a pickle file means you can re-run some steps of the analysis without having to re-compute everything. For example, you can load ‘output/transform/transform.pkl’ and re-run Cluster. The standard caveat with pickle files is that they are \*possibly\* a security risk, so be sensible. At this stage, a non-default work flow is still experimental and not all argument combinations have been tested. The arguments are hopefully self-explantory, but a more detailed guide may be developed at a later stage.

To repeat the analysis in the previous section using the Workflow function, use the following code:

*import densitysurf as ds*

*import scanpy as sc*

*data=sc.read\_10x\_h5("/Users/a123/Desktop/Human\_breast\_cancer\_blockA\_section1/V1\_Breast\_Cancer\_Block\_A\_Section\_1\_filtered\_feature\_bc\_matrix.h5")*

*data.var\_names\_make\_unique()*

*data = data.to\_df()*

*import pandas as pd*

*XY=pd.read\_csv("/Users/a123/Desktop/Human\_breast\_cancer\_blockA\_section1/spatial/tissue\_positions\_list.csv", header = None).iloc[:, [0, 4, 5]]*

*XY.columns = ['bin\_id', 'X', 'Y']*

*ds.Workflow(*

*# supply the input DataFrame*

*input\_dataframe = data,*

*# The path isn’t used to import data, but included in ‘info.txt’*

*input\_data\_path = "/Users/a123/Desktop/Human\_breast\_cancer\_blockA\_section1/",*

*#create in current working directory*

*output\_parent\_directory = "",*

*ncomps = 20,*

*n\_iter = 100,*

*cell\_K\_nn = 5,*

*gene\_K\_nn = 5,*

*gene\_clus\_steps = 1000,*

*similarity\_threshold\_clus\_cell = 0.2,*

*similarity\_threshold\_clus\_gene = 0.2,*

*similarity\_threshold\_specificity = 0.2,*

*neighbourhood\_flow\_K\_nn = 6,*

*dist\_thres = 300,*

*cell\_join\_id = 'bin\_id',*

*# supply input spatial coordinates DataFrame*

*XY = XY,*

*# the path isn’t used to import data, but included in ‘info.txt’*

*XY\_path = "/Users/a123/Desktop/Human\_breast\_cancer\_blockA\_section1/spatial/tissue\_positions\_list.csv",*

*neighbourhood\_flow = True*

*)*

**Joint sample analysis**

It can sometimes be useful to jointly analyse multiple samples. The output of DensitySurf might give you all the information you need about how your samples overlap (or not), or you can use it as a tool to facilitate your other work. For example, you might use it to diagnose external data integration methods, or use to interrogate the results of differential expression analysis on pseudo-bulked single cell data.

Firstly, each sample will be transformed. All other options will be set to False, including creating UMAP and goodness of fit output in Transform.

*import os*

*data\_paths = [*

*"/Users/a123/Desktop/Human\_breast\_cancer\_blockA\_section1/V1\_Breast\_Cancer\_Block\_A\_Section\_1\_filtered\_feature\_bc\_matrix.h5",*

*"/Users/a123/Desktop/Human\_breast\_cancer\_blockA\_section2/V1\_Breast\_Cancer\_Block\_A\_Section\_2\_filtered\_feature\_bc\_matrix.h5",*

*"/Users/a123/Desktop/Human\_breast\_cancer\_ductal\_carcinoma/Visium\_FFPE\_Human\_Breast\_Cancer\_filtered\_feature\_bc\_matrix.h5"*

*]*

*# python index starts at 0, and finishes at N-1*

*# indentation is very important in python, it controls flow e.g. for loops*

*for i in range(0, len(data\_paths)):*

*data = sc.read\_10x\_h5(data\_paths[i])*

*data.var\_names\_make\_unique()*

*data = data.to\_df()*

*# create unique folders for output directory structures*

*os.mkdir("data" + str(i + 1))*

*ds.Workflow(*

*input\_dataframe = data,*

*input\_data\_path = data\_paths[i],*

*# use the new folders created above*

*output\_parent\_directory = 'data' + str(i + 1) + '/',*

*ncomps = 20,*

*umap = False,*

*transform\_goodness\_of\_fit = False,*

*cluster\_cells = False,*

*cluster\_genes = False,*

*specificity\_network = False,*

*neighbourhood\_flow = False*

*)*

Next, use MultiSampleConcat to combine the individual samples into one DataFrame

*# location of individually transformed samples*

*# this is a list comprehension, the Python version of lapply in R*

*path\_names = ['data' + str(i + 1) + '/output/transform/' for i in range(0, len(data\_paths))]*

*# concatenate the samples. The index is modified to indicate separate # samples. It is possible to supply custom sample names to this function.*

*D = ds.MultiSampleConcat(path\_names)*

Finally, use the Workflow function to analyse this matrix. It is important to use transform\_input = False because the individual samples are already transformed.

*# create a new directory for joint analysis*

*os.mkdir("multisample")*

*# Mostly using default parameter settings to reduce typing*

*ds.Workflow(D,*

*input\_data\_path = '\*Multi-sample analysis\*',*

*output\_parent\_directory = 'multisample/',*

*ncomps = 20,*

*# the input data is already the concatenation of transformed data*

*transform\_input = False*

*)*

**Iterative analysis on subsets of data**