Using the Python package 'DensitySurf'
Kristen Feher
25-05-2025

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.

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
# A full guide to importing different data formats from different
technologies may appear at a later date, but for now, the python
package scanpy is a good place to start.
data=sc.read_10x_h5("/Users/a123/Desktop/Human_breast_cancer_blockA_sec
tion1/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): Transform, Cluster, SpecificityNetwork, NeighbourhoodFlow (spatial data only). They can be performed with the following code, mainly using default arguments:

```
# specify the path to the top level output directory, to save results
path = "path_to_output_directory/"
# create a directory structure to save results
ds.directory_structure(path)

# transform the data
# data must be a Pandas DataFrame, with cells in the rows and genes in
the columns. The index (row labels) are used to uniquely label the
cells and track each cell throughout the analysis. The columns must be
labelled with the gene names.

T = ds.Transform(data, ncomps = 200)
# Carry out goodness of fit
T.goodness_of_fit()
# Use the output from goodness of fit to choose 40 components and
create the UMAPs
T.umap(ncomp_cell = 40, ncomp_gene = 40)
# Save  results
T.save(path + '/output/transform/')

# Perform clustering on transform object
C_cells = ds.Cluster(T, mode = 'cells', ncomp_clus = 40)
C_genes = ds.Cluster(T, mode = 'genes', ncomp_clus = 40)
# Save clustering results
C_cells.save(path + '/output/cells/')
C_genes.save(path + '/output/genes/')

# Create biomarker specificity network using Transform and Clustering
objects
SN = ds.SpecificityNetwork(T, C_cells, C_genes)
# Save specificity network results
SN.save(path + '/output/specificity_network/')

# (Only for spatial data) Create Neighbourhood Flow object using cell
Clustering object and a pandas DataFrame XY containing the spatial
coordinates. XY must have two columns labelled ‘X’ and ‘Y’ and its
index (i.e. cell labels) must correspond to that of input data.
NF = ds.NeighbourhoodFlow(C_cells, XY, K_nn = 5, dist_thres = 500,
edge_quantile_threshold=0.95)
NF.save(path + '/output/neighbourhood_flow/')
```

Some notes on the spatial analysis:

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). Alternatively the cell label can be in the index of XY. 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')
```

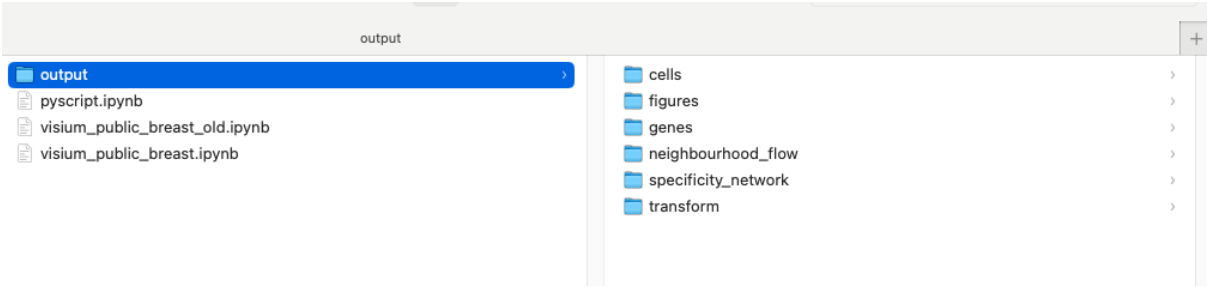
In NeighbourhoodFlow, the argument K_nn is the number of spatial nearest neighbours. For 10x Visium data, it is set to 6 as the bins are on a hexagonal grid. The argument dist_thres is set to 300 as this is the distance between the bins, and deals with edge effects, i.e. spatial nearest neighbour distances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

Some notes on saving:

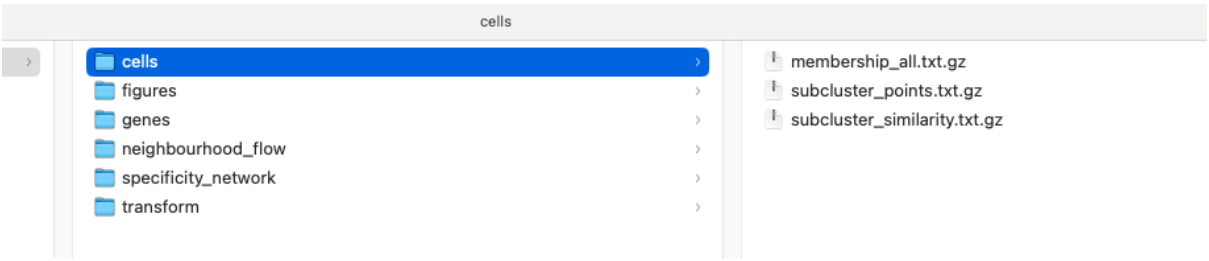
Saving the output is very easy and to save everything, simply give a path to the save directory. There is the option to selectively only save some of the output. While you can save the output wherever you want, I recommend staying organised. To help with that, you can use the ‘directory_structure’ function 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.

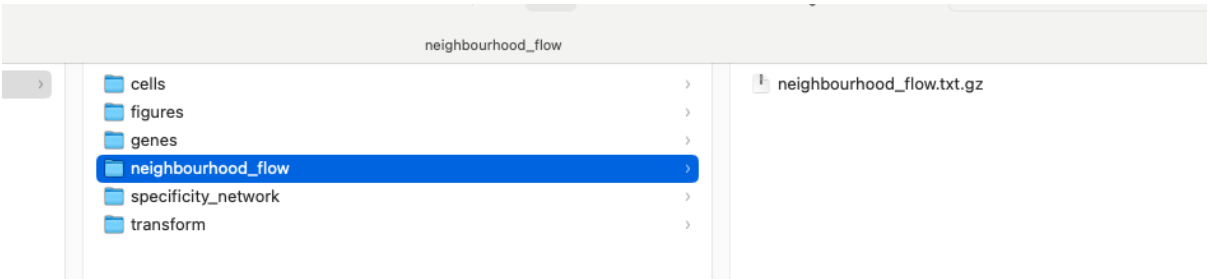
ds.directory_structure(path = "")

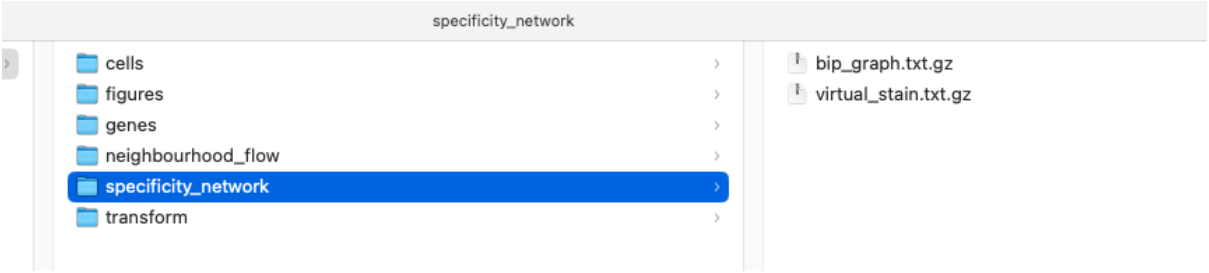
This is the directory structure it will result in:

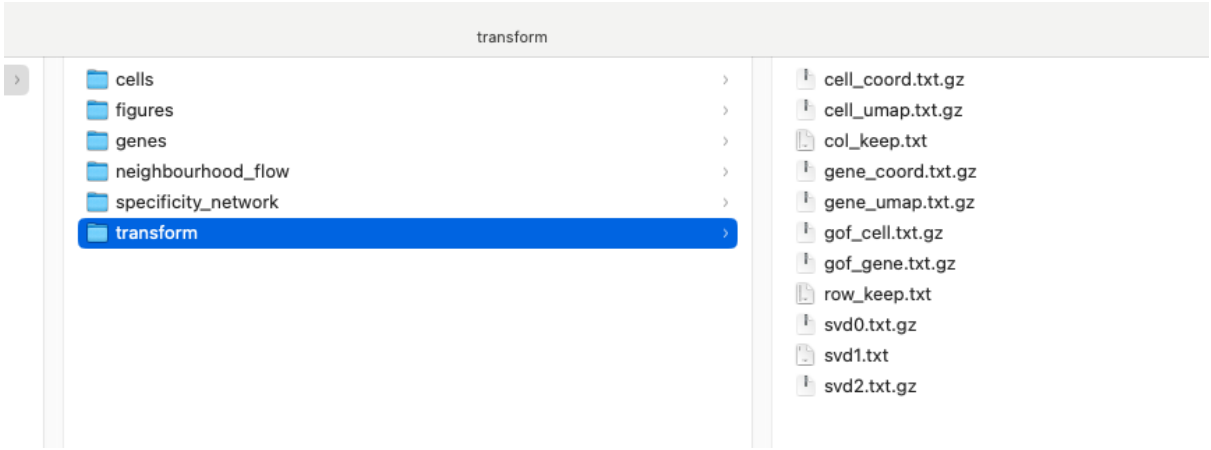


After saving the output from Transform, Cluster, SpecificityNetwork and NeighbourhoodFlow, the directories will be populated with R- friendly output as shown below. The file names may be updated from time to time so you may see something different than shown here.








transform
cells
figures
genes
neighbourhood_flow
specificity_network
transform
cell_coord.txt.gz
cell_umap.txt.gz
col_keep.txt
gene_coord.txt.gz
gene_umap.txt.gz
gof_cell.txt.gz
gof_gene.txt.gz
row_keep.txt
svd0.txt.gz
svd1.txt
svd2.txt.gz