GraphST

Release 1.1

Yahui Long

Mar 08, 2023

CONTENTS

CHAPTER

ONE

SPATIALLY INFORMED CLUSTERING, INTEGRATION, AND DECONVOLUTION OF SPATIAL TRANSCRIPTOMICS WITH GRAPHST

1.1 Installation

The GraphST package is developed based on the pytorch framework and can be implemented on both GPU and CPU. We recommend running the package on GPU. Please ensure that pytorch and CUDNN are installed correctly. To run GraphST, all dependencies included in the file 'requirement.txt' need to be installed. We provide two ways to install the package of GraphST.

Please note that the current GraphST version offers full support of Linux operating system. Further version for other operating systems would be released soon.

1.1.1 1. Python

Dowloading the package from https://github.com/JinmiaoChenLab/GraphST/

```
pip install GraphST

or

git clone https://github.com/JinmiaoChenLab/GraphST.git

cd GraphST

python setup.py build

python setup.py install --user
```

1.1.2 2. Anaconda

For convenience, we suggest using a separate conda environment for running GraphST. Please ensure annaconda3 is installed.

Create conda environment and install GraphST package.

```
#create an environment called GraphST
conda create -n GraphST python=3.8

#activate your environment
```

(continues on next page)

(continued from previous page)

```
conda activate GraphST

#install package

pip install GraphST

or

git clone https://github.com/JinmiaoChenLab/GraphST.git

cd GraphST

python setup.py build

python setup.py install --user

#To use the environment in jupyter notebook, add python kernel for this environment.

pip install ipykernel

python -m ipykernel install --user --name=GraphST
```

1.2 Tutorial 1: 10X Visium

In this tutorial, we show how to apply GraphST to identify spatial domains on 10X Visium data. As a example, we analyse the 151673 sample of the dorsolateral prefrontal cortex (DLPFC) dataset. Maynard et al. has manually annotated DLPFC layers and white matter (WM) based on the morphological features and gene markers.

We derived the preprocessed data from the spatialLIBD package, including manual annotations. Before running the model, please download the input data via https://drive.google.com/drive/folders/1DocCbwz5_ADyO_lnarjAIi1KKLSqtizB.

1.2.1 Loading package

[65]:
```
import os
import torch
import pandas as pd
import scanpy as sc
from sklearn import metrics
import multiprocessing as mp
```

[66]:
```
from GraphST import GraphST
```

[67]:
```
# Run device, by default, the package is implemented on 'cpu'. We recommend using GPU.
device = torch.device('cuda:1' if torch.cuda.is_available() else 'cpu')

# the location of R, which is necessary for mclust algorithm. Please replace the path␣
↪below with local R installation path
os.environ['R_HOME'] = '/scbio4/tools/R/R-4.0.3_openblas/R-4.0.3'
```

[68]:
```
# the number of clusters
n_clusters = 7
```

[69]:
```
dataset = '151673'
```

1.2.2 Reading ST data

The necessary input files includes: 1) The gene expression matrix: filtered_feature_bc_matrix.h5; 2) Spatial coordinates: position.txt; 3) Histology image: the format should be .png.

In the example, position inforamtion has been saved in adata.obsm['spatial']. To make the model can read the data sucessfully, please ensure the same format input file as example.

[70]:
```
# read data
file_fold = '/home/yahui/Yahui/Projects/data/' + str(dataset) #please replace 'file_fold
↪' with the download path
adata = sc.read_visium(file_fold, count_file='filtered_feature_bc_matrix.h5', load_
↪images=True)
adata.var_names_make_unique()
```

```
/home/yahui/anaconda3/envs/STGAT/lib/python3.8/site-packages/anndata/_core/anndata.py:
↪1830: UserWarning: Variable names are not unique. To make them unique, call `.var_
↪names_make_unique`.
  utils.warn_names_duplicates("var")
```

[71]:
```
adata
```

[71]:
```
AnnData object with n_obs × n_vars = 3639 × 33538
    obs: 'in_tissue', 'array_row', 'array_col'
    var: 'gene_ids', 'feature_types', 'genome'
    uns: 'spatial'
    obsm: 'spatial'
```

1.2.3 Training the model

GraphST model aims to learn the representations for spots by making full use of gene expressions and spatial location information in a self-supervised learning way. After model training, the learned representations will be saved in adata.obsm['emb'], and can be used for spatial clustering.

[72]:
```
# define model
model = GraphST.GraphST(adata, device=device)

# train model
adata = model.train()
```

```
Begin to train ST data...
```

```
100%|| 600/600 [00:07<00:00, 84.92it/s]
```

```
Optimization finished for ST data!
```

[73]:
```
adata
```

[73]:
```
AnnData object with n_obs × n_vars = 3639 × 33538
    obs: 'in_tissue', 'array_row', 'array_col'
    var: 'gene_ids', 'feature_types', 'genome', 'highly_variable', 'highly_variable_rank
↪', 'means', 'variances', 'variances_norm', 'mean', 'std'
    uns: 'spatial', 'hvg', 'log1p'
    obsm: 'spatial', 'distance_matrix', 'graph_neigh', 'adj', 'label_CSL', 'feat', 'feat_
↪a', 'emb'
```

1.2.4 Spatial clustering and refinement

In the clustering result, some spots may be wrongly assigned to spatially disparate domains. We consider such occurrences to be noise and their presence may influence downstream biological analysis. Therefore, we extend our GraphST model with an optional optimization step to remove the noises. In short, for a given spot, its lable will be re-assigned as the same domain as the most common lable of its surronding spots (please refer to the manuscript for more details). To do so, parameter 'radius' is set to specify the number of neighbors.

Please note that this step is not recommended for ST data with fine-grained domains (e.g., mouse brain anterior and posterior), Stereo-seq, and Slide-seqV2. In this study, we only applied this refinement step to the human brain DLPFC and the human breast cancer dataset.

After model training, the representation for spots are generated and used as input of clustering tool for spatial clustering. Here we provid three available kinds of tools for spaital clustering, including mclust, leiden, and louvain. In our experiment, we find mclust performs better than leiden and louvain on spatial data in most cases. Therefore, we recommend using mclust.

[74]:
```
# set radius to specify the number of neighbors considered during refinement
radius = 50

tool = 'mclust' # mclust, leiden, and louvain

# clustering
from GraphST.utils import clustering

if tool == 'mclust':
   clustering(adata, n_clusters, radius=radius, method=tool, refinement=True) # For␣
↪DLPFC dataset, we use optional refinement step.
elif tool in ['leiden', 'louvain']:
   clustering(adata, n_clusters, radius=radius, method=tool, start=0.1, end=2.0,␣
↪increment=0.01, refinement=False)
```

```
fitting ...
  |========================================================================| 100%
```

1.2.5 Visualization

For DLPFC data, the original authors manually annotated the slices. The annotation (metadata.tsv) for 151673 slice can be downloaded from https://drive.google.com/drive/folders/1DocCbwz5_ADyO_lnarjAIi1KKLSqtizB. For quantitative assessment, we use well-known ARI metric to evaulate the performance. Since not all of spots were annotated, we filtered out NA nodes before the ARI calculation and visualization.

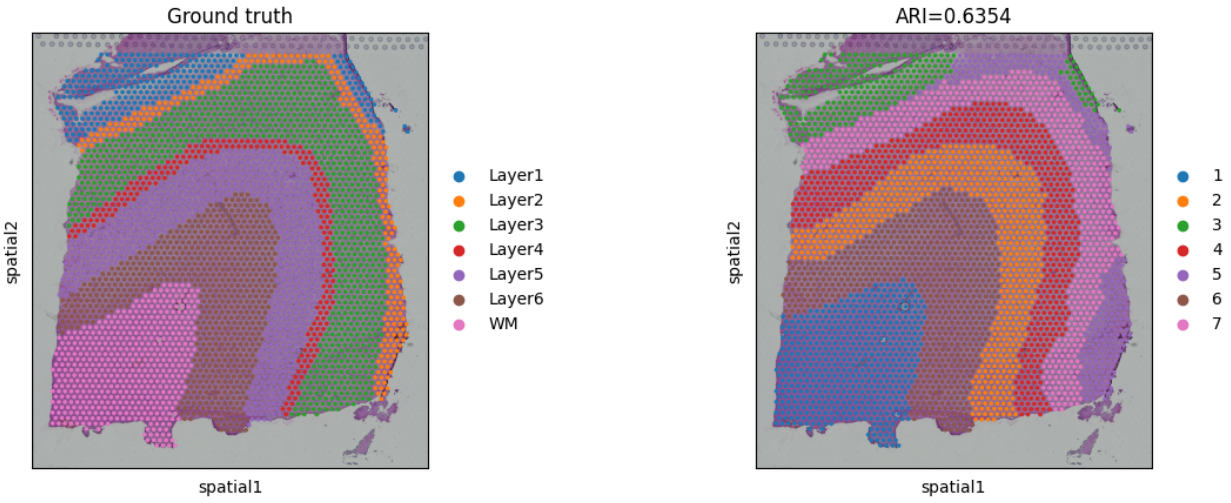
[75]:
```
# add ground_truth
df_meta = pd.read_csv(file_fold + '/metadata.tsv', sep='\t')
df_meta_layer = df_meta['layer_guess']
adata.obs['ground_truth'] = df_meta_layer.values
```

[76]:
```
# filter out NA nodes
adata = adata[~pd.isnull(adata.obs['ground_truth'])]

# calculate metric ARI
ARI = metrics.adjusted_rand_score(adata.obs['domain'], adata.obs['ground_truth'])
adata.uns['ARI'] = ARI

print('Dataset:', dataset)
print('ARI:', ARI)
```

```
/home/yahui/anaconda3/envs/STGAT/lib/python3.8/site-packages/anndata/compat/_overloaded_
↪dict.py:106: ImplicitModificationWarning: Trying to modify attribute `._uns` of view,␣
↪initializing view as actual.
  self.data[key] = value
```

```
Dataset: 151673
ARI: 0.63535759181513
```

[77]:
```
# plotting spatial clustering result
sc.pl.spatial(adata,
              img_key="hires",
              color=["ground_truth", "domain"],
              title=["Ground truth", "ARI=%.4f"%ARI],
              show=True)
```



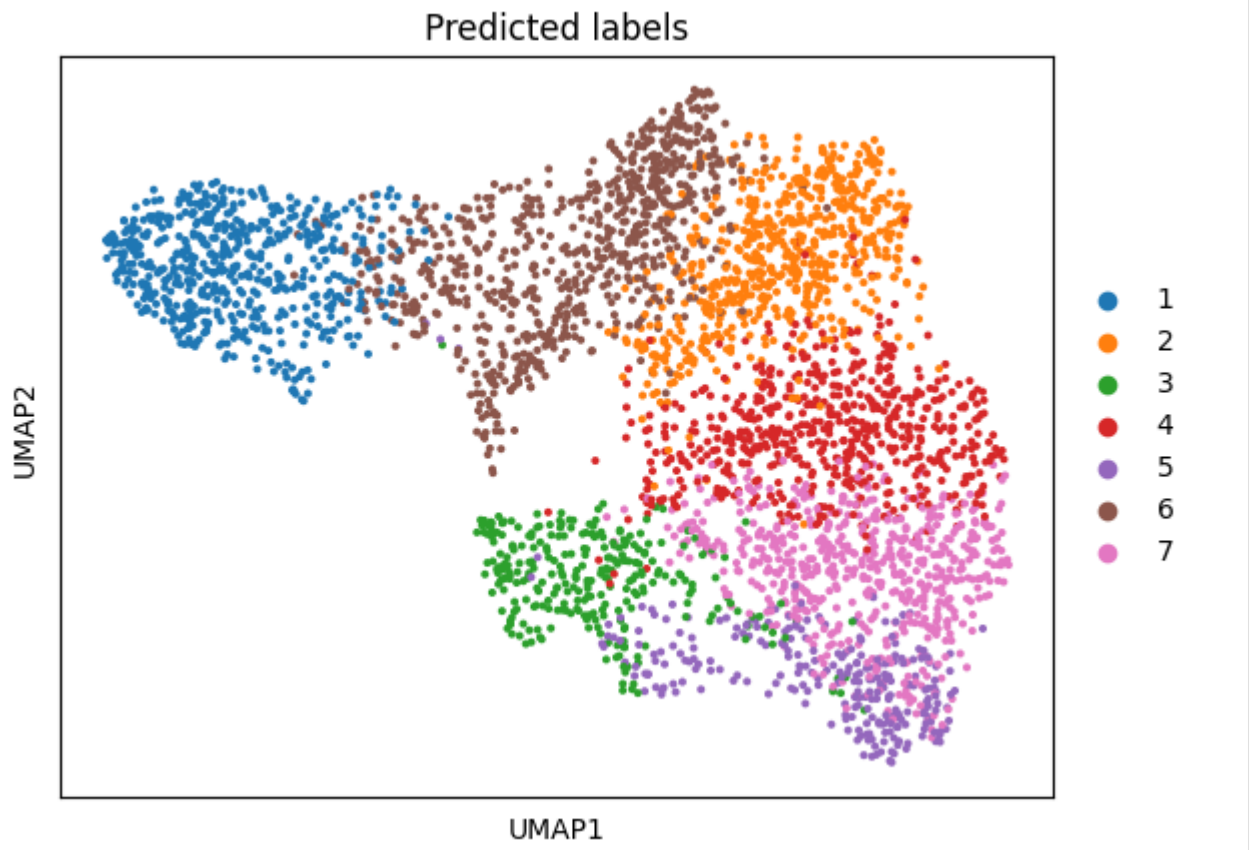
The learned representations will be incluced in adata.obsm[‘emb’] or adata.obsm[‘emb_pca’] (PCA dimension reduc-

tion), which can be used for UMAP visualization.

[78]:
```
# plotting predicted labels by UMAP
sc.pp.neighbors(adata, use_rep='emb_pca', n_neighbors=10)
sc.tl.umap(adata)
sc.pl.umap(adata, color='domain', title=['Predicted labels'], show=False)
```

```
/home/yahui/anaconda3/envs/STGAT/lib/python3.8/site-packages/scanpy/plotting/_tools/
↪scatterplots.py:392: UserWarning: No data for colormapping provided via 'c'.␣
↪Parameters 'cmap' will be ignored
  cax = scatter(
```

[78]:
```
<AxesSubplot: title={'center': 'Predicted labels'}, xlabel='UMAP1', ylabel='UMAP2'>
```



[ ]:

1.3 Tutorial 2: scRNA and ST data integration (deconvolution)

In this tutorial, we show how to apply GraphST to integrate scRNA-seq and ST data, i.e., deconvolution. Taking human lymph node dataset as example, both scRNA-seq and ST data were downloaded from an existing study by Kleshchevnikov et al. and provided at https://drive.google.com/drive/folders/1ns-EsWBu-SNrJ39j-q-AFIV5U-aXFwXf.

After downloading the data, we can obtain 'scRNA.h5ad' and 'ST.h5ad' files, which are corresponding reference and spatial transcriptomics data respectively. Cell type information is included in scRNA.obs['cell_type'].

[8]:
```
import scanpy as sc
from GraphST import GraphST
```

[9]:
```
dataset = 'Human_Lymph_Node'
```

1.3.1 Reading ST data

[10]:
```
# read ST data
file_fold = '/home/yahui/Yahui/Projects/data/Human_Lymph_Node/' #Please replace 'file_
↪fold' with the ST download path
adata = sc.read_h5ad(file_fold + 'ST.h5ad')

#For '10X' ST data, please read it instead as:
#file_fold = '/home/yahui/Yahui/Projects/data/' + str(dataset) #Please replace it with␣
↪the ST download path
#adata = sc.read_visium(file_fold, count_file='filtered_feature_bc_matrix.h5', load_
↪images=True)

adata.var_names_make_unique()
```

1.3.2 Pre-processing for ST data

[11]:
```
# preprocessing for ST data
GraphST.preprocess(adata)

# build graph
GraphST.construct_interaction(adata)
GraphST.add_contrastive_label(adata)
```

1.3.3 Reading reference data

[12]:
```
# read scRNA daa
file_path = '/home/yahui/Yahui/Projects/data/Human_Lymph_Node/scRNA.h5ad' # Please␣
↪replace 'file_path' with the scRNA download path.
adata_sc = sc.read(file_path)
adata_sc.var_names_make_unique()
```

1.3.4 Pre-processing for reference data

[13]:
```
# preprocessing for scRNA data
GraphST.preprocess(adata_sc)
```

1.3.5 Finding overlap genes between ST and reference data

[14]:
```
# find overlap genes
from GraphST.preprocess import filter_with_overlap_gene
adata, adata_sc = filter_with_overlap_gene(adata, adata_sc)
```

```
Number of overlap genes: 1313
```

```
/home/yahui/anaconda3/envs/STGAT/lib/python3.8/site-packages/anndata/compat/_overloaded_
↪dict.py:106: ImplicitModificationWarning: Trying to modify attribute `._uns` of view,␣
↪initializing view as actual.
  self.data[key] = value
```

1.3.6 Extracting features for ST data

[15]:
```
# get features
GraphST.get_feature(adata)
```

1.3.7 Implementing GraphST for cell type deconvolution

[16]:
```
import torch
# Run device, by default, the package is implemented on 'cpu'. We recommend using GPU.
device = torch.device('cuda:1' if torch.cuda.is_available() else 'cpu')

# Train model
model = GraphST.GraphST(adata, adata_sc, epochs=1200, random_seed=50, device=device,␣
↪deconvolution=True)
adata, adata_sc = model.train_map()
```

```
Begin to train ST data...
```

```
100%|| 1200/1200 [00:14<00:00, 80.63it/s]
```

```
Optimization finished for ST data!
Begin to train scRNA data...
```

```
100%|| 1200/1200 [00:20<00:00, 59.78it/s]
```

```
Optimization finished for cell representation learning!
Begin to learn mapping matrix...
```

```
100%|| 1200/1200 [02:02<00:00,  9.80it/s]
```

```
Mapping matrix learning finished!
```

1.3.8 Visualization of single cell data distribution in ST tissue

After model training, we can obtain the learned mapping matrix with dimension 'n_spot x n_cell' in adata.obsm['map_matrix']. Each element in the mapping matrix denotes the mapping probability of a cell in a given spot. To filter out noise, we only consider the top 'retain_percent' cell values for each spot.

We usually set the 'retain_percent' value as 0.15. Users can change the parameter according to your requirement.

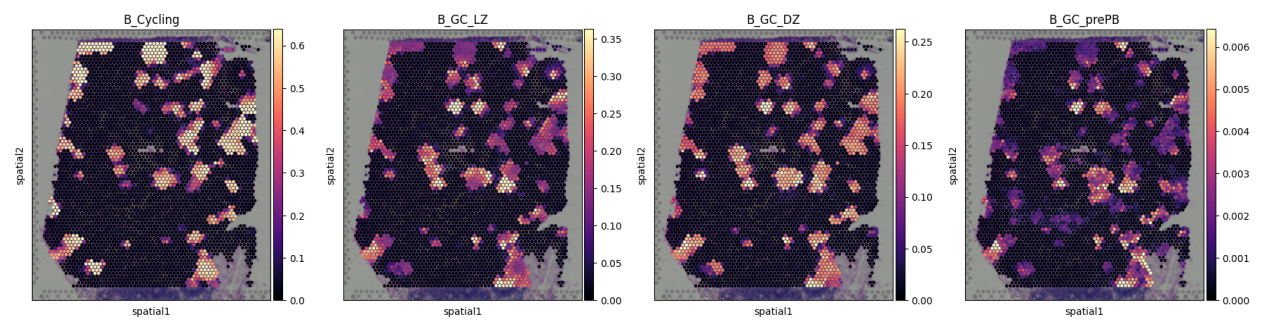
[17]:
```
# Project cells into spatial space
from GraphST.utils import project_cell_to_spot
project_cell_to_spot(adata, adata_sc, retain_percent=0.15)
```

After projection, the probability distributions of each cell type in spots are saved in adata.obs.

[18]:
```
adata
```

[18]:
```
AnnData object with n_obs × n_vars = 4035 × 1313
    obs: 'in_tissue', 'array_row', 'array_col', 'sample', 'B_Cycling', 'B_GC_DZ', 'B_GC_
↪LZ', 'B_GC_prePB', 'B_IFN', 'B_activated', 'B_mem', 'B_naive', 'B_plasma', 'B_preGC',
↪'DC_CCR7+', 'DC_cDC1', 'DC_cDC2', 'DC_pDC', 'Endo', 'FDC', 'ILC', 'Macrophages_M1',
↪'Macrophages_M2', 'Mast', 'Monocytes', 'NK', 'NKT', 'T_CD4+', 'T_CD4+_TfH', 'T_CD4+_
↪TfH_GC', 'T_CD4+_naive', 'T_CD8+_CD161+', 'T_CD8+_cytotoxic', 'T_CD8+_naive', 'T_TIM3+
↪', 'T_TfR', 'T_Treg', 'VSMC'
    var: 'feature_types', 'genome', 'SYMBOL', 'MT_gene', 'highly_variable', 'highly_
↪variable_rank', 'means', 'variances', 'variances_norm', 'mean', 'std'
    uns: 'spatial', 'hvg', 'log1p', 'overlap_genes'
    obsm: 'MT', 'spatial', 'distance_matrix', 'graph_neigh', 'adj', 'label_CSL', 'feat',
↪'feat_a', 'emb_sp', 'map_matrix'
```

[19]:
```
# Visualization of spatial distribution of scRNA-seq data
import matplotlib as mpl
import matplotlib.pyplot as plt
with mpl.rc_context({'axes.facecolor':  'black',
                     'figure.figsize': [4.5, 5]}):

        sc.pl.spatial(adata, cmap='magma',
                  # selected cell types
                  color=['B_Cycling', 'B_GC_LZ', 'B_GC_DZ', 'B_GC_prePB'],
                  ncols=5, size=1.3,
                  img_key='hires',
                  # limit color scale at 99.2% quantile of cell abundance
                  vmin=0, vmax='p99.2',
                  show=True
                 )
```



[ ]:

1.4 Tutorial 3: Stereo-seq

In this tutorial, we demonstrate how to apply GraphST to Stereo-seq data for spatial domains identification. We take mouse embryo 9.5 data as example and set the number of clusters as 22. Mouse embryo Stereo-seq data were downloaded from https://db.cngb.org/stomics/mosta/ and provided at https://drive.google.com/drive/folders/1QWHFMzhQ7WorVNLwx88xT-rbojf4nh9T.

Before running the model, please download input data by the link above.

[1]:
```
import os
import torch
import pandas as pd
import scanpy as sc
from sklearn import metrics
import multiprocessing as mp
```

```
/home/yahui/anaconda3/envs/STGAT/lib/python3.8/site-packages/tqdm/auto.py:22:␣
↪TqdmWarning: IProgress not found. Please update jupyter and ipywidgets. See https://
↪ipywidgets.readthedocs.io/en/stable/user_install.html
  from .autonotebook import tqdm as notebook_tqdm
```

[2]:
```
from GraphST import GraphST
```

[3]:
```
dataset = 'Mouse_Embryo'
```

[4]:
```
# Run deviceby default, the package is implemented on 'cpu'. We recommend using GPU.
device = torch.device('cuda:3' if torch.cuda.is_available() else 'cpu')

# the location of R, which is necessary for mclust algorithm. Please replace it with␣
↪local R installation path
os.environ['R_HOME'] = '/scbio4/tools/R/R-4.0.3_openblas/R-4.0.3'
```

[5]:
```
# the number of clusters
n_clusters = 22
```

1.4.1 Reading data

[6]:
```
# read data
file_path = '/home/yahui/anaconda3/work/CellCluster_DEC/data//Mouse_Embryo/' #please␣
↪replace 'file_path' with the download path
adata = sc.read_h5ad(file_path + 'E9.5_E1S1.MOSTA.h5ad')
adata.var_names_make_unique()
```

1.4.2 Implementing GraphST for spatial clustering

[7]:
```
# define model
model = GraphST.GraphST(adata, datatype='Stereo', device=device)

# run model
adata = model.train()
```

```
/home/yahui/anaconda3/envs/STGAT/lib/python3.8/site-packages/scanpy/preprocessing/_
↪highly_variable_genes.py:62: UserWarning: `flavor='seurat_v3'` expects raw count data,␣
↪but non-integers were found.
  warnings.warn(
```

```
Graph constructed!
Building sparse matrix ...
Begin to train ST data...
```

```
100%|| 600/600 [00:14<00:00, 42.39it/s]
```

```
Optimization finished for ST data!
```

1.4.3 Spatial clustering

After model training, the representation for spots are generated and used as input of clustering tool for spatial clustering. Here we provid three available kinds of tools for spaital clustering, including mclust, leiden, and louvain. In our experiment, we find mclust performs better than leiden and louvain on spatial data in most cases. Therefore, we recommend using mclust.

[8]:
```
# clustering
from GraphST.utils import clustering

tool = 'mclust' # mclust, leiden, and louvain

# clustering
from GraphST.utils import clustering

if tool == 'mclust':
   clustering(adata, n_clusters, method=tool)
elif tool in ['leiden', 'louvain']:
   clustering(adata, n_clusters, method=tool, start=0.1, end=2.0, increment=0.01)
```

```
R[write to console]:     __  ___________    __  ______________
   /  |/  / ____/ /   / / / / ___/_  __/
```

(continues on next page)

(continued from previous page)

```
  / /|_/ / /   / /   / / / /\__ \ / /
 / /  / / /___/ /___/ /_/ /___/ // /
/_/  /_/\____/_____/\____//____//_/    version 5.4.9
Type 'citation("mclust")' for citing this R package in publications.
```

```
fitting ...
  |======================================================================| 100%
```

1.4.4 Visualization

[10]:
```
#import matplotlib.pyplot as plt
#adata.obsm['spatial'][:, 1] = -1*adata.obsm['spatial'][:, 1]
#plt.rcParams["figure.figsize"] = (3, 4)
#plot_color=["#F56867","#556B2F","#C798EE","#59BE86","#006400","#8470FF",
#            "#CD69C9","#EE7621","#B22222","#FFD700","#CD5555","#DB4C6C",
#            "#8B658B","#1E90FF","#AF5F3C","#CAFF70", "#F9BD3F","#DAB370",
#           "#877F6C","#268785", '#82EF2D', '#B4EEB4']

#ax = sc.pl.embedding(adata, basis="spatial",
#                     color="domain",
#                     s=30,
#                     show=False,
#                     palette=plot_color,
#                     title='GraphST')
#ax.axis('off')
#ax.set_title('Mouse Embryo E9.5')
```

1.5 Tutorial 4: Horizontal Spatial Transcriptomics Integration

In this tutorial, we demonstrate how to analyse multiple tissue slices in horizontal integration. Here we take mouse anterior and posterior brain as example. ST data were downloaded from https://www.10xgenomics.com/. Before inputting the model, alignment algorithm was implemented to align mouse anterior and posterior brain data.

Please note that aligned position information must be saved in adata.obsm[‘spatial’] before running the model.

The prepocessed data can be accessible and downloaded via https://drive.google.com/drive/folders/1jDmx8IjiGhOD__spuuhFB1fWVDJt

[29]:
```
import os
import torch
import pandas as pd
import scanpy as sc
from sklearn import metrics
import multiprocessing as mp
```

[30]:
```
from GraphST import GraphST
```

[31]:
```
# Run device, by default, the package is implemented on 'cpu'. We recommend using GPU.
device = torch.device('cuda:3' if torch.cuda.is_available() else 'cpu')
```

(continues on next page)

(continued from previous page)

```
# the location of R, which is necessary for mclust algorithm. Please replace it with␣
↪local R installation path
os.environ['R_HOME'] = '/scbio4/tools/R/R-4.0.3_openblas/R-4.0.3'
```

[32]:
```
# the number of clusters
n_clusters = 26
```

1.5.1 Reading data

[33]:
```
# read data
#file_fold = './Mouse_Brain/' #please replace 'file_fold' with the download path
file_fold = '/home/yahui/anaconda3/work/CellCluster_DEC/data/Mouse_Brain_Merge_Anterior_
↪Posterior/'
#adata = sc.read_h5ad(file_fold + 'mouse_anterior_posterior_brain_merged.h5ad')
adata = sc.read_h5ad(file_fold + 'filtered_feature_bc_matrix.h5ad')
adata.var_names_make_unique()
```

```
/home/yahui/anaconda3/envs/long/lib/python3.8/site-packages/anndata/_core/anndata.py:
↪1828: UserWarning: Observation names are not unique. To make them unique, call `.obs_
↪names_make_unique`.
  utils.warn_names_duplicates("obs")
```

1.5.2 Implementing GraphST for multi-sample integration

[34]:
```
# define model
model = GraphST.GraphST(adata, device=device, random_seed=50)

# run model
adata = model.train()
```

```
  1%|                                                                                | 8/
↪600 [00:00<00:08, 71.41it/s]
```

```
Begin to train ST data...
```

```
100%|| 600/600 [00:09<00:00, 66.15it/s]
```

```
Optimization finished for ST data!
```

1.5.3 Spatial clustering

After model training, the representation for spots are generated and used as input of clustering tool for spatial clustering. Here we provid three available kinds of tools for spaital clustering, including mclust, leiden, and louvain. In our experiment, we find mclust performs better than leiden and louvain on spatial data in most cases. Therefore, we recommend using mclust.

[35]:
```
# clustering
from GraphST.utils import clustering
```

(continues on next page)

(continued from previous page)

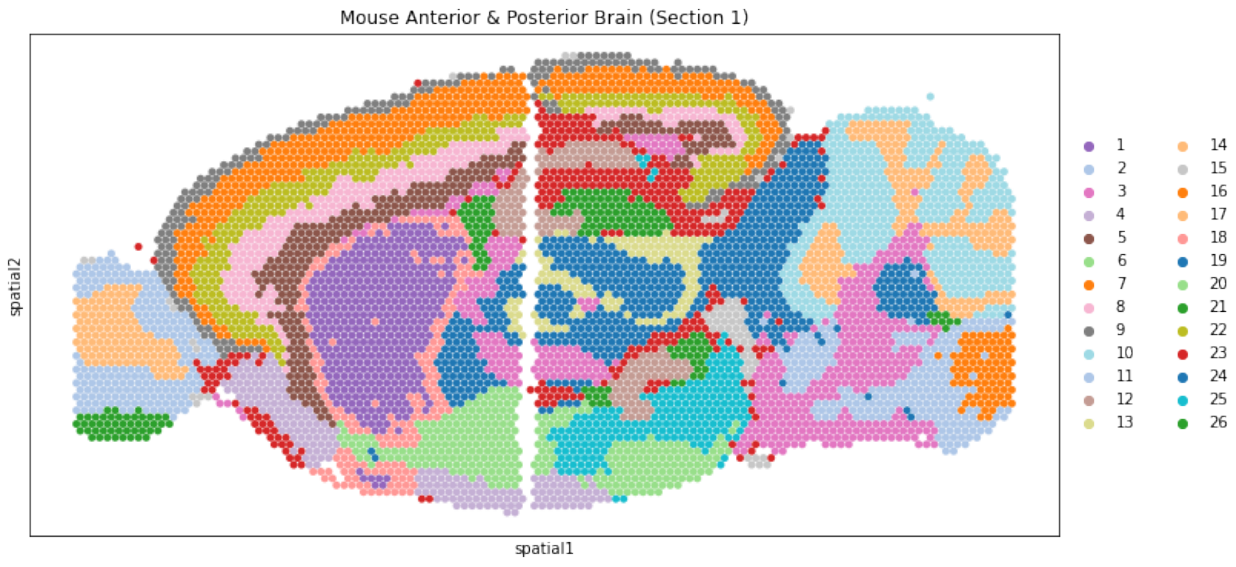
```
tool = 'mclust' # mclust, leiden, and louvain

# clustering
from GraphST.utils import clustering

if tool == 'mclust':
   clustering(adata, n_clusters, method=tool)
elif tool in ['leiden', 'louvain']:
   clustering(adata, n_clusters, method=tool, start=0.1, end=2.0, increment=0.01)
```

```
fitting ...
  |========================================================================| 100%
```

1.5.4 Visualization

[36]:
```
# plotting spatial clustering result
import matplotlib.pyplot as plt
import seaborn as sns
adata.obsm['spatial'][:,1] = -1*adata.obsm['spatial'][:,1]
rgb_values = sns.color_palette("tab20", len(adata.obs['domain'].unique()))
color_fine = dict(zip(list(adata.obs['domain'].unique()), rgb_values))

plt.rcParams["figure.figsize"] = (12, 6)
sc.pl.embedding(adata, basis="spatial",
                color="domain",
                s=100,
                palette=color_fine,
                show=False,
                title='Mouse Anterior & Posterior Brain (Section 1)')
```

[36]:
```
<AxesSubplot:title={'center':'Mouse Anterior & Posterior Brain (Section 1)'}, xlabel=
↪'spatial1', ylabel='spatial2'>
```



[ ]:

1.6 Tutorial 5: Vertical Spatial Transcriptomics Integration

In this tutorial, we demonstrate how to analyse multiple tissue slices in vertical integration. Here we take mouse breast cancer sample1 as example. The ST data were generated from our lab (Jinmiao Chen's Lab). Before inputting the model, alignment algorithm was implemented to align sections 1 and 2.

Please note that aligned position information must be saved in adata.obsm['spatial'] before running the model.

The prepocessed data can be accessible and downloaded via https://drive.google.com/drive/folders/1zwGqgC84gVfDeFea5VSRU6U_Qac

[26]:
```
import os
import torch
import pandas as pd
import scanpy as sc
from sklearn import metrics
import multiprocessing as mp
import matplotlib.pyplot as plt
```

[27]:
```
from GraphST import GraphST
```

[28]:
```
# Run device, by default, the package is implemented on 'cpu'. We recommend using GPU.
device = torch.device('cuda:1' if torch.cuda.is_available() else 'cpu')

# the location of R, which is necessary for mclust algorithm. Please replace it with␣
↪local R installation path
os.environ['R_HOME'] = '/scbio4/tools/R/R-4.0.3_openblas/R-4.0.3'
```

[29]:
```
# the number of clusters
n_clusters = 10
```

1.6.1 Reading data

[30]:
```
# read data
file_fold = '/home/yahui/Yahui/Projects/data/S1_A1_S3_A1/' #please replace 'file_fold'␣
↪with the download path
adata = sc.read_h5ad(file_fold + 'filtered_feature_bc_matrix.h5ad')
adata.var_names_make_unique()
```

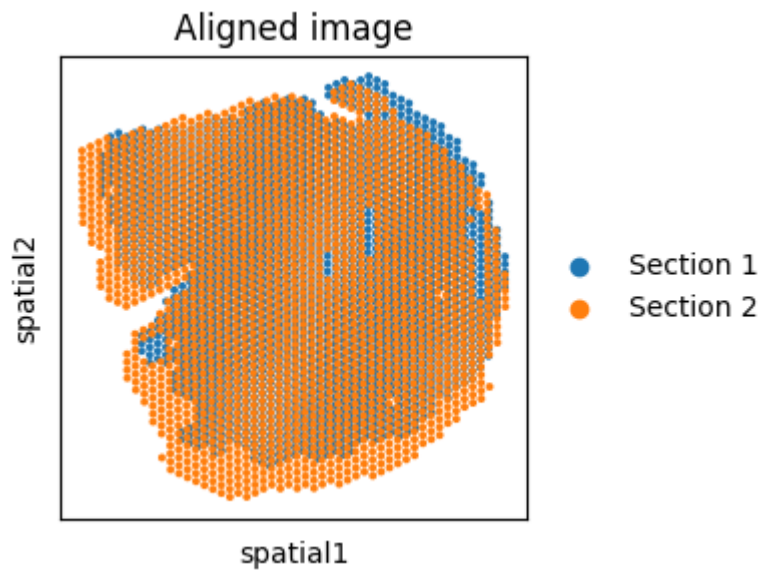
1.6.2 Plotting aligned image

Sample labels are saved in adata.obs[‘data’]. ‘S1’ denotes Section1 while ‘S3’ denotes Section 2

[31]:
```
plt.rcParams["figure.figsize"] = (3, 3)
adata.obsm['spatial'][:, 1] = -1*adata.obsm['spatial'][:, 1]
adata.obs['data'].replace({'S1':'Section 1', 'S3':'Section 2'}, inplace=True)
ax = sc.pl.embedding(adata, basis='spatial',
                color='data',
                show=False)
ax.set_title('Aligned image')
#ax.axis('off')
```

```
/home/yahui/anaconda3/envs/STGAT/lib/python3.8/site-packages/scanpy/plotting/_tools/
↪scatterplots.py:392: UserWarning: No data for colormapping provided via 'c'.␣
↪Parameters 'cmap' will be ignored
  cax = scatter(
```

[31]:
```
Text(0.5, 1.0, 'Aligned image')
```



1.6.3 Implementing GraphST for batch integration

[32]:
```
# define model
model = GraphST.GraphST(adata, device=device)

# run model
adata = model.train()
```

```
Begin to train ST data...
```

```
100%|| 600/600 [00:06<00:00, 90.18it/s]
```

```
Optimization finished for ST data!
```

1.6.4 Joint spatial clustering

After model training, the representation for spots are generated and used as input of clustering tool for spatial clustering. Here we provid three available kinds of tools for spaital clustering, including mclust, leiden, and louvain. In our experiment, we find mclust performs better than leiden and louvain on spatial data in most cases. Therefore, we recommend using mclust.

[33]:
```
# clustering
from GraphST.utils import clustering

tool = 'mclust' # mclust, leiden, and louvain

# clustering
from GraphST.utils import clustering

if tool == 'mclust':
   clustering(adata, n_clusters, method=tool) # For DLPFC dataset, we use optional␣
↪refinement step.
elif tool in ['leiden', 'louvain']:
   clustering(adata, n_clusters, method=tool, start=0.1, end=2.0, increment=0.01)
```

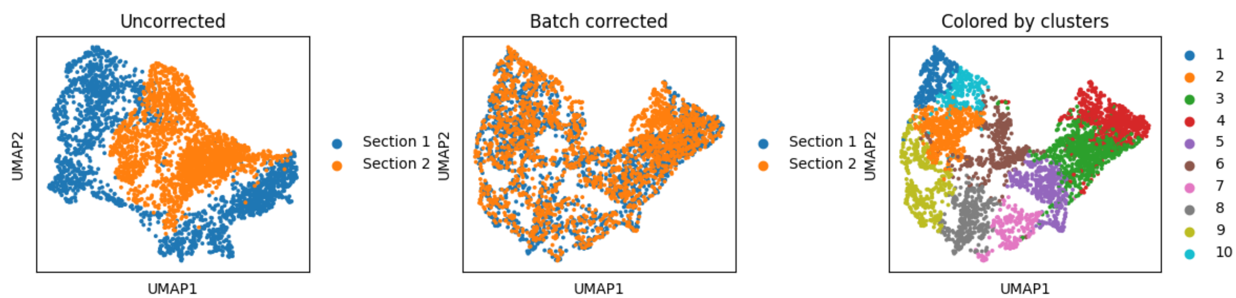
```
fitting ...
  |========================================================================| 100%
```

1.6.5 Plotting UMAP before and after batch effect correction

[34]:
```
fig, ax_list = plt.subplots(1, 3, figsize=(12, 3))

### Plotting UMAP before batch effect correction
sc.pp.normalize_total(adata)
sc.pp.log1p(adata)
sc.pp.pca(adata)

sc.pp.neighbors(adata, use_rep='X_pca', n_neighbors=10, n_pcs=40)
sc.tl.umap(adata)
sc.pl.umap(adata, color='data', title='Uncorrected',
                  ax = ax_list[0],
                  show=False)

### Plotting UMAP after batch effect correction
sc.pp.neighbors(adata, use_rep='emb_pca', n_neighbors=10)
sc.tl.umap(adata)
sc.pl.umap(adata,
           color='data',
           ax=ax_list[1],
           title='Batch corrected',
           #legend_loc = 'bottom margin',
           show=False)

### Color by predicted domains
sc.pl.umap(adata, color='domain', ax=ax_list[2], title='Colored by clusters', show=False)

plt.tight_layout(w_pad=0.02)
```

```
WARNING: adata.X seems to be already log-transformed.
```

```
/home/yahui/anaconda3/envs/STGAT/lib/python3.8/site-packages/scanpy/plotting/_tools/
↪scatterplots.py:392: UserWarning: No data for colormapping provided via 'c'.␣
↪Parameters 'cmap' will be ignored
  cax = scatter(
/home/yahui/anaconda3/envs/STGAT/lib/python3.8/site-packages/scanpy/plotting/_tools/
↪scatterplots.py:392: UserWarning: No data for colormapping provided via 'c'.␣
↪Parameters 'cmap' will be ignored
  cax = scatter(
/home/yahui/anaconda3/envs/STGAT/lib/python3.8/site-packages/scanpy/plotting/_tools/
↪scatterplots.py:392: UserWarning: No data for colormapping provided via 'c'.␣
↪Parameters 'cmap' will be ignored
  cax = scatter(
```



1.6.6 Plotting joint clustering results

For mouse breast cancer sample1, we manually annotated section2 according to H&E image. The ground truth labels are avaialbe at https://drive.google.com/drive/folders/1zwGqgC84gVfDeFea5VSRU6U_QacpSnwT.

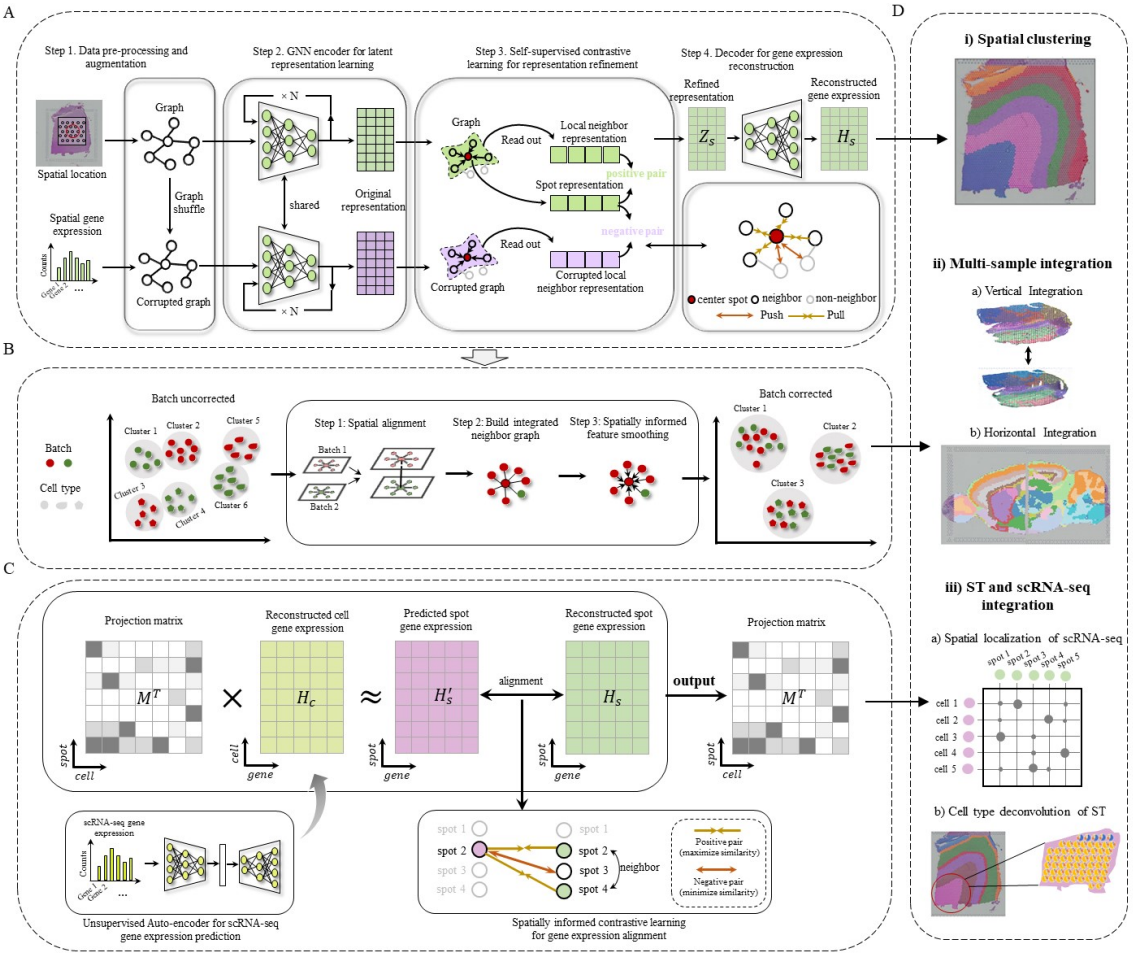
[36]:
```
#from sklearn import metrics
### Splitting adata into Section 1 and Section 2
#adata_section1 = adata[adata.obs['data']=='Section 1', :]
#adata_section2 = adata[adata.obs['data']=='Section 2', :]

#fig, ax_list = plt.subplots(1, 2, figsize=(7, 3))
#sc.pl.embedding(adata_section1,
#                basis='spatial',
#                color='domain',
#                show = False,
#                s=50,
#                title='Section 1',
#                ax = ax_list[0])

#sc.pl.embedding(adata_section2,
#                basis='spatial',
#                color='domain',
#                show = False,
#                s=50,
#                title = ['Section 2'],
#                ax = ax_list[1])
```

(continues on next page)

(continued from previous page)

```
#plt.tight_layout(w_pad=0.2)
```

[ ]:



CHAPTER

TWO

OVERVIEW

GraphST is a versatile graph self-supervised contrastive learning model that incorporates spatial location information and gene expression profiles to accomplish three key tasks, spatial clustering, spatial transcriptomics (ST) data integration, and single-cell RNA-seq (scRNA-seq) transfer onto ST. GraphST combines graph neural networks (GNNs) with self-supervised contrastive learning to learn spot representations in the ST data by modeling gene expressions and spatial locaiton information. After the representation learning, the non-spatial alignment algorithm is used to cluster the spots into different spatial domains. Each cluster is regarded as a spatial domain, containing spots with similar gene expression profiles and spatially proximate. GraphST can jointly analyze multiple ST samples while correcting batch effects, which is achieved by smoothing features between spatially adjacent spots across samples. For the scRNA-seq transfer onto ST data, a mapping matrix is trained via an augmentation-free contrastive learning mechanism, where the similarity of spatially adjacent spots are maximized while those of spatially non-adjacent spots are minimized. With the learned mapping matrix, arbitrary cell attributes (e.g., cell type and sample type) can be flexibly projected onto spatial space.

CHAPTER

THREE

CITATION

Long et al. Spatially informed clustering, integration, and deconvolution of spatial transcriptomics with GraphST. **Nature Communications**. 14(1), 1155 (2023)