CS690 Assignment 2 - Team 1

Advaith Kannan 210072
Aniruddh Pramod 210142
Debarpita Dash 220328

September 29, 2024

1 Introduction

This report analyzes three computational tools for spatialomics data: CytoCommunity [2], Spatial LDA [1], and SPIRAL. Each tool is tested on my dataset to see how well they identify cellular neighborhoods and interactions. CytoCommunity uses graph-based algorithms in both supervised and unsupervised modes. Spatial LDA enhances the traditional Latent Dirichlet Allocation (LDA) by incorporating a spatial coherence prior. SPIRAL compensates for batch fitting as well as makes efficient spatial alignments. By comparing these methods, this report explores their strengths, weaknesses, and suitability for various biological datasets.

1.1 Work Distribution

Each of us worked on a separate paper for this assignment. Advaith Kannan covered SPIRAL, which is described in Section 2, Aniruddh Pramod covered CytoCommunity which is described in Section 3 and Debarpita Dash covered Spatial LDA which is covered in Section 4.

Section	Name	Method
2	Advaith Kannan	SPIRAL
3	Aniruddh Pramod	CytoCommunity
4	Debarpita Dash	Spatial LDA

Table 1: Work Distribution Table

2 SPIRAL

Recent advancements in spatially resolved transcriptomics (SRT) have revolutionized the study of gene expression by providing spatial context to cellular data. However, integrating data from different sources remains challenging due to batch effects and coordinate misalignments. In this assignment, SPIRAL was employed, a novel algorithm designed for effective batch-effect removal and spatial coordinate alignment.

2.1 Spatial Feature Plots

Figures 1 and 2 depict spatial feature plots generated from SPIRAL-based clustering of the datasets. These plots demonstrate the spatial consistency achieved by SPIRAL, even across datasets with different experimental conditions.

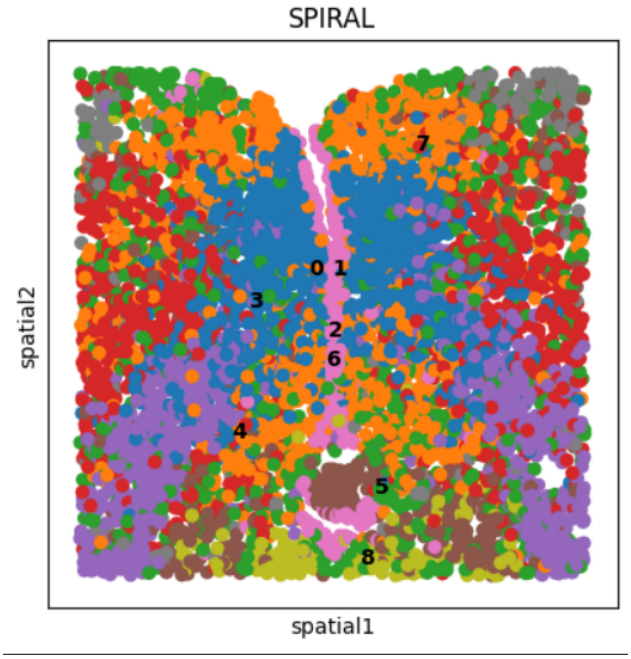

Figure 1: SPIRAL: Spatial feature plot of dataset 1.

2.2 Conclusion

In this assignment, SPIRAL was applied to integrate multiple SRT datasets. SPIRAL's ability to preserve spatial domain continuity and generalize to unseen datasets makes it a valuable tool for SRT data analysis.

3 CytoCommunity

CytoCommunity [2] is a computational tool developed for the discovery of tissue cellular neighborhoods (TCNs) from cell phenotypes in spatial omics data. It uses both unsupervised and supervised learning frameworks to analyze cell-cell interactions and spatial arrangements within tissues. In this assignment, I only focus on the unsupervised variant, since we don't have access to the true clusters. By leveraging graph-based algorithms, it identifies TCNs across different spatial scales and biological conditions, offering insights into tissue organization and cell communication patterns in health and disease. This method is versatile, being applicable to both transcriptomics and proteomics datasets.

3.1 Environment Setup

CytoCommunity comes with instructions to build an environment to run its code. This can be accomplished using both conda and pip. This environment is needed to run its scripts.
However, I recommend using a separate environment for all the preprocessing and post-processing steps. This is because CytoCommunity uses some dated libraries which are not forward compatible. In particular, ScanPy is unable to use its UMAP or Clustering capabilities if you try to install it alongside CytoCommunity.
To setup the CytoCommunity environment, use the
`/A2_Aniruddh/CytoCommunity/environment.yml` file.
To setup the preprocessing environment,use the `/A2_Aniruddh/environment.yml` file. This environment includes ScanPy, CellTypist, ipykernel and other general libraries that are useful for preprocessing tasks.
Additionally, a working installation of R with version 4.4 is recommended. The only package we need is diceR, however this relies on a package called MASS which recently underwent a huge overhaul, and only works with R version 4.4 and above. It should be possible to use diceR by installing the older versions of the required dependencies, however it is much more convenient to just use version 4.4

3.2 Preprocessing

The paper requires input data organized into a directory, stored in two types of files. An `{Image_Name}_Coordinates.txt` file and an `{Image_Name}_CellTypeLabels.txt` file. The former should be a tab separated file with the x and y coordinates of each spot. The latter is an assignment of each spot to a cell type. We also need an `ImageNameList.txt` file which the script will read to identify what Image Names are available.
Thus, the only inputs to this model are a cell type label, treated as a categorical variable and the coordinates of the spots. This method never looks at the gene expression matrix. Since the cell type label is treated as a categorical variable, I have not performed celltype annotation when using Leiden Clustering. The cluster numbers are used as a cell type label directly.
Preprocessing is implemented in the file `preprocessing.ipynb`. It runs the clustering algorithm and organizes the data from an h5ad file into the format required.

I have tested two clustering methods, one being the default Leiden clustering we have done previously in this course, and the other being CellTypist's auto-assignment [3]. The reason for testing CellTypist [3] is that it relies on overclustering the data. It can provided extremely detailed cell-types, which should serve as better quality input for the TCN model. The first section of the notebook performs Leiden clustering, whereas the second section of the notebook performs CellTypist autoassignment. This code should be run with the preprocessing environment.
When using Leiden clustering I tried to use the resolution that gave me a similar number of clusters as specified in the Assignment PDF, ie. 8 for MERFISH and 11 for OSMFISH.
The output from this notebook is stored under the data directory, with the corresponding {Image_Name}. CellTypist's output is prefixed with 'typist' before the {Image_Name}.
Using CellTypist gave better results on the MERFISH dataset, but Leiden Clustering gave better results on the OSMFISH dataset, and so these are reflected in the final script.

3.3 Model Scripts

The scripts labelled Step1 through 4 are modifications of the paper's original scripts with a system argument parser. The reason for this is to make hyperparameter tuning simpler, as will be demonstrated in the next subsection. Step1 prepares the cellular spatial graphs, Step 2 contains code to train the model using unsupervised learning and perform soft TCN (Tissue Cellular Neighbourhood) Assignment, Step 3 prepares the ensemble model and does the final assignment and Step 4 does the visualization. Steps 1 and 2 are implemented in Python, and must be run in the CytoCommunity environment. Step 3 has been implemented in R, and Step 4 must be run in Python as well.
Another modification made was in Step 2, where I appended the result of a single run of this step into a Results file, along with the hyperparameters used in this run.
Step 4 also needed some modification since it relied on an older version of matplotlib and seaborn, and procedures for using them together have changed since then. I also increased the palette size to 50 to reflect the fact that CellTypist assigned around 44 cell types to the dataset.

3.4 Hyperparameter Tuning

A Jupyter notebook instance cannot run multiple Python Kernels at the same time in general. There are two main workarounds for this. The first involves using magic commands to change the environment, while the second involves the subprocess module.
The magic command method involves using the !conda command to activate the new environment, and the !python command to run the script. These must be stitched together using && to ensure that !python is run with the environment given by !conda. This will let us temporarily create an instance of a kernel and run code on it. However, the results are not persistent, and no output will be visible.
The subprocess module allows us to send a command to the terminal from within the notebook. This can change the environment being used by a cell. However this fails because the Jupyter notebook prefers using its original kernel for magic commands, and so calling the

scripts will still use the Jupyter kernel as its Python instance, not the one specified by the subprocess module. It should be possible to use the subprocess module in a Python script file to perform hyperparameter tuning.
The final hurdle is the fact that Step 3 is implemented in R. I was not able to set up a local R in the conda environment with my required version and hence I needed to deactivate conda before running this script.
The final solution I settled on was using a batch script to do the hyperparameter tuning. It tries a couple of combinations of learning rate, embedding dimensionality and K (from the KNN) and the script from Step 2 compiles the results in a `Results.csv` file in the corresponding Step2_Output folder. The `tuner.bat` script can be used to test a variety of values for the various hyperparameters. Image_name must be specified in the script.
Some combinations are missing from the results file, and this is because they did not lead to convergence. The script in Step 2 asks for a Loss Threshold, below which it deems underfitting and restarts the run. If a particular combination is constantly running into this problem, it might get stuck in an infinite loop, unable to climb the plateau.
Based on the testing, these are the optimal hyperparameters for both datasets -

Hyperparameter	**MERFISH**	**OSMFISH**
Clustering Method	CellTypist	Leiden
K_KNN	60	80
TCN	8	11
Epochs	2500	2500
Learning Rate	0.005	0.001
Embedding Dimensionality	256	128

Table 2: Hyperparameter values for MERFISH and OSMFISH

Reproducing these results can be done by running the `preprocessing.ipynb` notebook completely to prepare the data and then simply calling the `best\_res.bat` script which will run all required files with the correct environments, provided PATH is configured correctly on Windows. Alternatively you can run the scripts yourself in the same order as specified in the file.

3.5 Results

On MERFISH, the best score achieved on Kaggle was **0.37251**, using the hyperparameters specified above. Notably, the method is not very sensitive to hyperparameter values on this dataset and is very stable. The Cell Type and TCN Label Graphs are shown below in figures 3 and 4.
On OSMFISH, the best score achieved on Kaggle was **0.32621**, using the hyperparameters specified above. However, the method was extremely unstable on this data. Minor changes to hyperparameters could lead to the model not converging. I am uncertain as to what the reason for this could be. Using CellTypist preprocessing also usually did not lead to convergence with this method. The Cell Type and TCN Label Graphs are shown below in figures 5 and 6.

These graphs are also available in the Step4_Output_typistmerfish and Step4_Output_osmfish directory.

4 Spatial LDA

Spatial-LDA is a probabilistic topic model for identifying characteristic cellular microenvironments from in-situ multiplexed imaging data. It was applied to reveal distinct populations of B cells in the mouse spleen and explore the immune environment in triple-negative breast cancer tumors.

4.1 Environment Setup and Scripts

To setup the environment for spatial LDA use the environment.yml file. This can be done using both conda and pip. There are multiple .py files that are used for spatial LDA. A brief overview of python files.

- `featurization.py`: Responsible for extracting features from spatial transcriptomics data, including neighborhood identification and anchor cell selection.
- `model.py`: Defines the core Latent Dirichlet Allocation (LDA) model and implements the learning algorithms for estimating topic distributions.
- `online_lda.py`: Facilitates online or incremental training of the LDA model, allowing updates as new data arrives.
- `primal_dual.py`: Implements primal-dual optimization techniques for training the LDA model, focusing on efficient calculations of gradients and Hessians.
- `visualization.py`: Creates visual representations of LDA results, such as topic distributions and spatial organization of cell types.
- `admm.py`: Implements the Alternating Direction Method of Multipliers (ADMM) for optimizing LDA model parameters while managing primal and dual variables.

4.2 Modifications to Feature Extraction methods

4.2.1 Initial Clustering Approach

I initially performed Leiden clustering before applying spatial LDA, which was crucial for feature extraction. Functions such as `neighborhood_to_cluster` and `neighborhood_to_marker` depend on specific labels—like cluster labels or gene marker labels—to extract and summarize data from designated indices. When I attempted to run the analysis using randomly sampled cells instead of clustering, the model took a long time to train and did not yield satisfactory results.

4.2.2 Anchor Cell Selection

For the featurization, this method selects an anchor cell as a reference point in the spatial dataset. In the paper, tumor cells from the triple-negative breast cancer (TNBC) dataset were considered as anchor cells. I identified the largest cluster as the anchor cell for both datasets. This approach was advantageous because the largest cluster is likely to represent the most abundant and prominent cell type in the tissue microenvironment, providing a stable reference point for analysis. However, I also attempted to select anchor cells based on gene expression values, but this approach did not yield appropriate results because the expression patterns did not sufficiently capture the spatial context within the cells.

4.2.3 Generating Multiple Samples

Multiple samples are required for spatial LDA because a single sample often leads to poor clustering results, as evidenced by my attempt with just one sample, which resulted in clustering all cells into a single class. For effective clustering to occur, it is essential to have multiple samples that capture the heterogeneity of the cellular microenvironment. To address this, I defined a distance threshold (`x_threshold`) to split the data into two samples based on spatial coordinates. This approach was motivated by the idea that distance could serve as a meaningful parameter for distinguishing between different cellular environments. Additionally, I attempted to create separate samples using KMeans clustering to partition the AnnData object into two samples based on the spatial coordinates (x and y). Although this method generated a new `spatial_cluster` column in `adata.obs`, the resulting division was not satisfactory and did not reflect the distinct cellular environments effectively.

4.2.4 Neighborhood Identification

After feature extraction, neighbourhood identification was performed via KDTree and query ball tree method.A KDTree is built for both anchor cells and neighboring cells based on their spatial coordinates. Using the query ball tree method, the function identifies which neighboring cells which fall within the specified radius of each anchor cell. The value of radius has to be varied in such a manner that we get minimum non zero values in our feature extraction matrix.This means that we want to identify neighborhoods where there are sufficient neighboring cells contributing to the analysis without including too many irrelevant cells.

4.2.5 Modifications to Feature Functions

Once neighborhoods are defined, features can be extracted using the specified feature functions, which perform operations such as counting the occurrence of clusters and summing specified markers. It also creates graphical representations of the relationships between cells using a nearest neighbor graph and a minimum spanning tree.
I modified the `neighborhood_to_marker` function to ensure that the sum of `neighborhood_genes` is greater than zero instead of relying on a specific threshold. The change was made because setting a threshold could lead to the exclusion of certain relevant data, resulting in an

expression matrix with predominantly zero values. This was likely due to the inherent characteristics of our dataset, where some neighborhoods may not reach the threshold despite having biologically significant expression levels.

4.2.6 Visualization of Relationships

I also did a visualisation of relationships or differences in the features between the samples.

4.3 Model

During the training phase, the model alternates between two key steps: the M-step and the E-step. In the M-step, a standard Latent Dirichlet Allocation (LDA) model is trained using the sample features to estimate the topic distributions. In the E-step, the model uses the counts computed earlier to update the topic weights, applying difference matrices that account for spatial regularization. Since there was a mismatch in size I utilised padding for resizing.
Once the model is trained, it can infer topic weights for new sample data by reusing the learned structure and topic components. This inference process updates the topic weights for the new data to ensure they align with the previously established model.
The function also computes the primal objective under Gaussian assumptions, calculating its gradient and Hessian. A dual variable is derived to check for convergence. The ADMM optimization iterates between adjusting the primal and dual variables to minimize the objective while meeting constraints. Primal-dual variables are updated using Newton-like steps and projected to maintain non-negativity.
Although I also explored a primal-dual based implementation of the model, the ADMM-based solution is preferred due to its significantly faster performance.
Key parameters that can be tuned during training include the `difference_penalty`, which applies a penalty to the topic priors of adjacent index cells. A higher penalty can lead to more distinct topic assignments among these cells, enhancing the separation of cell populations. Another parameter is `max_admm_iter`; increasing this value allows for more iterations, helping the model converge to an optimal solution. We can modify `N_TOPICS_LIST` to get desired number of clusters.

4.4 Results

The scores achieved on MERFISH and OsmFISH were 0 and 0.06643, respectively. The very low score can be attributed to several factors. First, Spatial LDA often results in a reduced number of clustered cells compared to the total number of cells in the dataset. This occurs because Spatial LDA operates under specific assumptions about data distribution; cells that do not conform to these assumptions may be grouped differently or excluded from the final clustering output. Additionally, the choices made during sample splitting and the selection of anchor cells could have contributed to the poor results.

5 Conclusion

On the MERFISH dataset, CytoCommunity performs the best, reaching a score of **0.37251**, and on the OSMFISH dataset, SPIRAL performs the best, reaching a score of **0.46213**.

References

[1] Zhaokai Chen et al. "Modeling Multiplexed Images with Spatial-LDA Reveals Novel Tissue Microenvironments". In: *Journal of Computational Biology* 27.8 (2020), pp. 1204–1218. ISSN: 1066-5277. DOI: 10.1089/cmb.2019.0340. URL: https://www.ncbi.nlm.nih.gov/pmc/articles/PMC7415889/.

[2] Yuxuan Hu et al. "Unsupervised and supervised discovery of tissue cellular neighborhoods from cell phenotypes". In: *Nature Methods* 21.2 (Jan. 2024), pp. 267–278. ISSN: 1548-7105. DOI: 10.1038/s41592-023-02124-2. URL: http://dx.doi.org/10.1038/s41592-023-02124-2.

[3] Chuan Xu et al. "Automatic cell-type harmonization and integration across Human Cell Atlas datasets". In: *Cell* 186.26 (2023), 5876–5891.e20. ISSN: 0092-8674. DOI: https://doi.org/10.1016/j.cell.2023.11.026. URL: https://www.sciencedirect.com/science/article/pii/S0092867423013120.

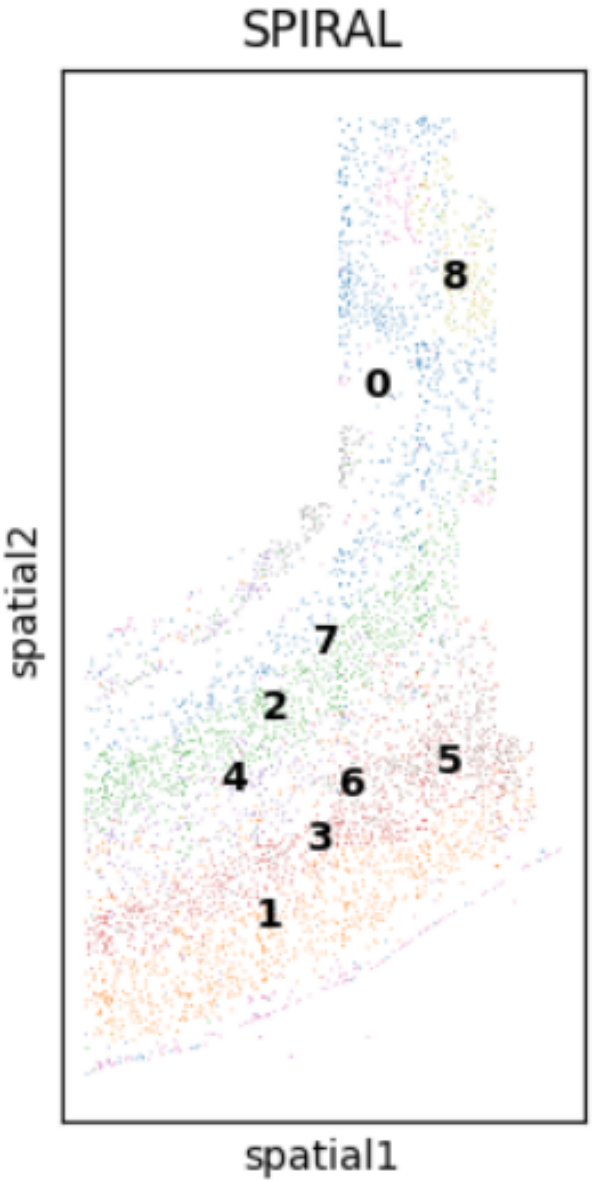

Figure 2: SPIRAL: Spatial feature plot of dataset 2.

Figure 3: CytoCommunity: MERFISH Cell Type Labels

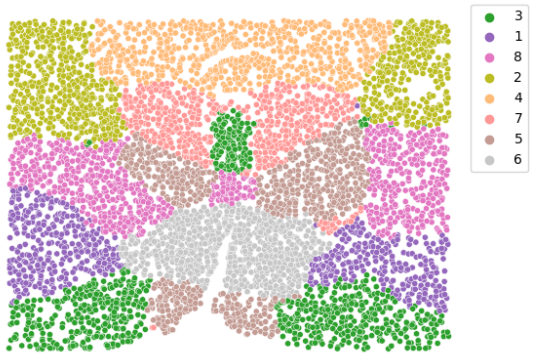

Figure 4: CytoCommunity: MERFISH TCN Labels

Figure 5: CytoCommunity: OSMFISH Cell Type Labels

Figure 6: CytoCommunity: OSMFISH TCN Labels

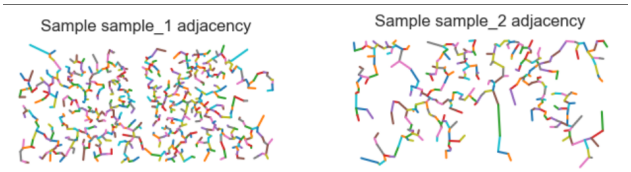

Figure 7: MERFISH adjacency graph

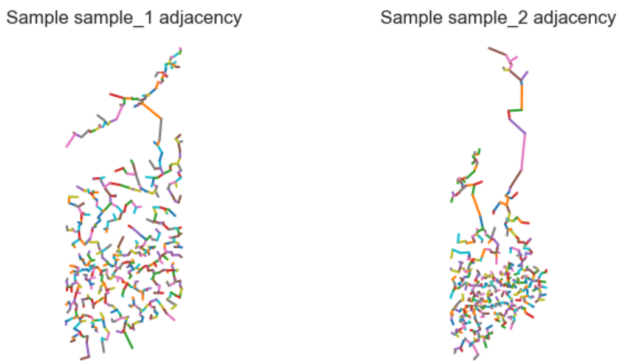

Figure 8: OsmFISH adjacency graph

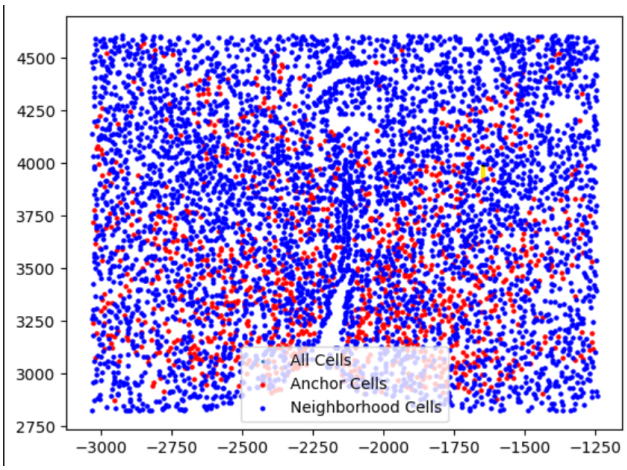

Figure 9: MERFISH clustering