COMP 565: Assignment 4: unsupervised learning of single-cell transcriptome data using NMF and ETM

March 8, 2023

This assignment is worth 8% of your total grade and due at **23:59 on March 27, 2023**. *This assignment may be time consuming for some of you so **do start early**.*

The Google Drive directory named comp565_A4 contains a starter Python file `comp565_A4.py`, already implemented `etm.py` for ETM, and the single-cell RNA-seq data.

Start this assignment with `comp565_A4.py`. Submit the single completed file `comp565_A4.py` with your name and student ID on the top of the file to MyCourses once you finish all of the tasks (starting in Section 1). **Do not change any code in this file that you are not asked**.

The TA should be able to run your script to generate all of the plots (except for Figure 4) shown below.

Setting up Python environment & libraries on your computers

Set up the Python computing environment by installing the necessary Python libraries specified in the top of the `comp565_a4.py` script. You will need Python 3.7 above to run the code properly.

The easiest way to install most of these Python libraries is via either command line command `pip` or Anaconda (`https://www.anaconda.com/`) command line command `conda`. Both are free.

For example, in single-cell analysis, a very popular library is called `scanpy`. You can use `pip` to install it (`https://scanpy.readthedocs.io/en/stable/installation.html`):

```
pip install scanpy
```

In the last part of this assignment, we will be using PyTorch to train the deep learning variational autoencoder (VAE) model called scETM. To install `pytorch`, as instructed by their website

(https://pytorch.org/get-started/locally/), we enter in command line: `conda install pytorch torchvision torchaudio -c pytorch`

If you have trouble, installing these libraries, let the TA or the instructor know as soon as possible to get help. Do it early.

Mouse Pancreas single-cell RNA-sequencing data

In the Google Drive directory there is a folder called `data`, which contains the single-cell Mouse Pancreas dataset obtained from [1]. The dataset has 1886 cells measured over 4194 genes by scRNA-seq. The dataset were processed and filtered for the purpose of this assignment. Here are the individual files:

- `MP.pickle`: The single-cell expression count matrix (matrix of integers) of dimension 1886 cells by 4194 genes plus 2 columns indicating the batch IDs and cell types
- `MP_genes.pickle`: gene names for the 4194 genes
- `sample_info.csv`: a 1886 by 3 information matrix with the rows as the1886 cells and columns for the cell IDs, batch IDs (not used in this assignment), and cell type labels
- `cell_IDs.pkl`: cell IDs as a list of strings

The code at the beginning of the file `comp565_A4.py` will load the scRNA-seq data into a Pandas DataFrame `df` as well as the M gene names `genes` using a Python library called `pickle`. It will reorder the rows and columns and save the final $N \times M$ matrix X into a Numpy `ndarray` for our subsequent matrix factorization tasks.

In addition, using `anndata` library we create an AnnData object called `mp_anndata`, which as the input ndarray matrix X and the ground-truth cell type labels for the 1886 cells for evaluation purpose.

Evaluating clustering by adjusted Rand Index

Adjusted Rand Index is a popular metric used to evaluate the unsupervised clustering by comparing the consistency between the predicted clusters and the ground-truth clusters (i.e., cell type labels in this assignment). In `comp565_A4.py` file, you are provided with a function called `evaluate_ari(cell_embed, adata)`, which takes $N \times K$ input cell embedding `cell_embed` with

K as the embedding dimensions and the annotated cell label data as `adata`. It will first run UMAP to compute the distance between cells based on their embeddings and then run Louvain clustering using the cells-cells distance matrix from UMAP to cluster cells into groups defined by the resolution parameter (default: 0.15). Finally, it computes the ARI based on the Louvain clusters and the ground-truth cell type using `adjusted_rand_score` from `Scikit-learn`. We will be using `evaluate_ari` to evaluate the cell embedding quality by NMF and scETM below.

1 Implement NMF to model scRNA-seq counts by minimizing sum of squared reconstruction error (1%)

Implement the non-negative matrix factorization (NMF) function `nmf_sse` based on slide 19 in Lecture 12. A function header, Docstring, and function call have been written for you.

Note that the NMF takes an input matrix of dimension $M \times N$ with M genes and N samples whereas the input matrix X is a $N \times M$ dimension. Also, the matrix H is a $K \times N$ matrix where the `evaluate_ari(cell_embed, adata)` above expects the cell embedding to have dimension $N \times K$. So you will need to provide these functions with the transposed of X and H, respectively.

Besides returning the final matrices W and H, save the mean squared error (MSE) $MSE = ||X - WH||^2/(N * M)$ and ARI at each iteration into a 3-column ndarray called `perf` with the first column as the iteration index and return `perf` as the third output from the function `nmf_sse`.

Run your NMF for 100 iterations and return W, H, and `perf` for the following analyses. This should take less than 3 minutes depending on your local computer. On a 10-core Xeon iMac Pro, it takes less than a minute to finish.

2 Monitor training progress (1%)

Implement a function called `monitor_perf` that displays the SSE and ARI at each iteration from the above NMF-SSE training and save the plot as nmf_sse.eps. If your implementation is correct, you will observe that the SSE drops quite rapidly at the first few iterations and the ARI increases in a zigzag way because it is an independent metric from the training objective.

Use Figure 1 as your reference. Although you may see a different progress with different random initialization of W and H, the behaviour of your implemented NMF model should be very similar and this is true for all of the following tasks.

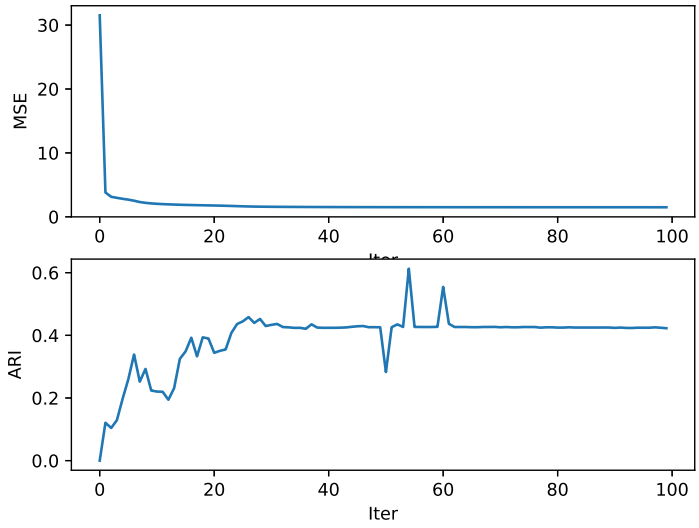

Figure 1: SSE and ARI of NMF at each iteration.

We are going to use `monitor_perf` to monitor the performances of the other two models that we are going to implement next.

3 Implement NMF to model scRNA-seq counts by maximizing log Poisson likelihood (1%)

Implement the NMF function `nmf_psn` that maximizes the log Poisson likelihood w.r.t. W and H s.t. $W, H \geq 0$ based on slide 20 in Lecture 12. Save the average log Poisson likelihood $(X \log WH - WH)/(N \times M)$ and ARI at each iteration.

Note that because the NMF algorithm requires element-wise division, to avoid dividing by zeros, you can set the zero values to a small value like so: `np.where(A > 0, A, 1e-16)`.

Check your implementation with `monitor_perf` (Figure 2).

Comparing the two models, ws observe that the NMF-Poisson model led to higher ARI than the NMF-SSE model (Figure 3). This implies that the Poisson likelihood is a better objective function to model the discrete read counts of the scRNA-seq data than the SSE loss. The latter is equivalent to maximizing the log of a univariance Gaussian likelihood.

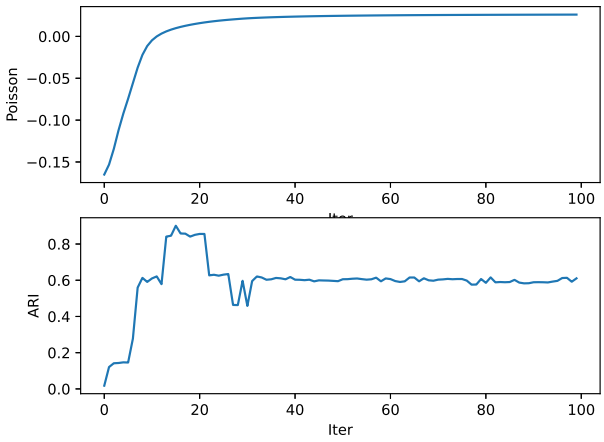

Figure 2: Poisson likelihood and ARI of NMF at each iteration.

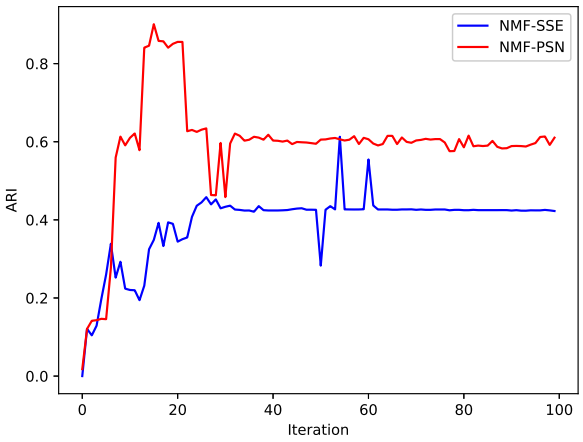

Figure 3: NMF-SSE versus NMF-Poisson in terms of ARI.

4 Implement a wrapper function to train Embedding Topic Model (ETM) (1%)

The same Google Drive directory contains another Python file called `etm.py`, which implements a simpler strip-down version of the scETM (Figure 4) [2]. The code is the same as the original ETM implementation [3] that is available from `https://github.com/adjidieng/ETM`. **Do not change the `etm.py` file**.

The code to instantiate an ETM model object is already written for you in `comp565_A4.py` file with pre-specified hyperparameters (i.e., topic number, hidden size, learning rates, weight decay penalty, etc). You want to start with these settings. Once you get the model working, you may play around with these parameters to get higher ARI. But that is not mandatory.

Implement the wrapper function called `train_scETM` that uses the helper functions `train_scETM_helper` to train a ETM and another completed helper function called `get_theta` (do not change) to compute cell embedding topic mixture θ. The latter two functions have been provided to you. **Do not modify them.**

PyTorch by default trace the error to calculate the gradient for backpropagation. When evaluating ARI, you want to turn off this automatic gradient trace by coding under the code block `with torch.no_grad()`.

Note here that `get_theta` takes the normalized gene counts by the total count per cell stored in `X_tensor_normalized` as the input to the encoder neural network (Figure 4b left-most side). The final reconstruction categorical likelihood is based on the unnormalized count data stored in `X_tensor` (Figure 4b right-most side).

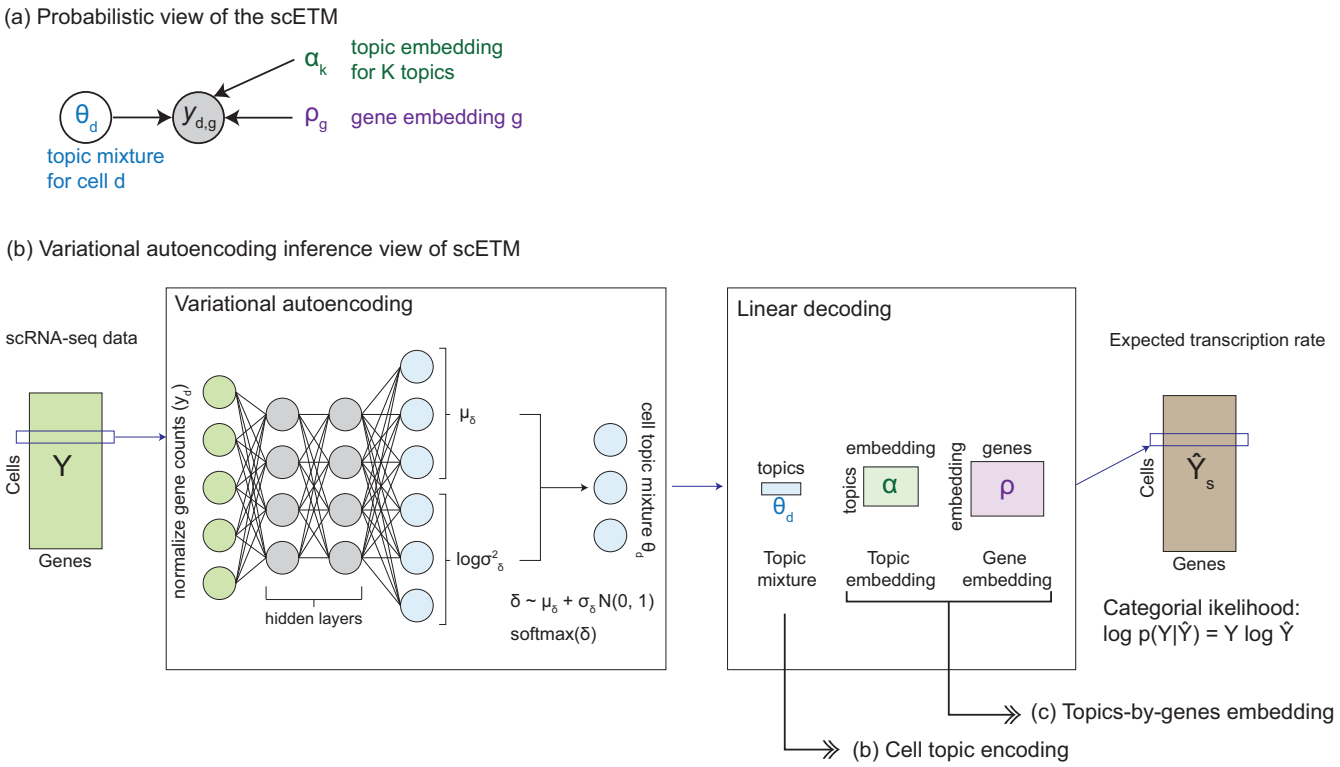

Figure 4: Modeling scRNA-seq data using single-cell embedded topic model (scETM) [2]. (a) probabilistic graphical model view of the scETM. (b) Variational autoencoder inference of the scETM. Each scRNA-seq profile serves as an input to a variational autoencoder (VAE) as the normalized gene counts. The encoder network produces a stochastic sample of the latent topic mixture (θ_d for cell $d = 1, \ldots, N$), which can be used for clustering cells. The linear decoder learns topic embedding and gene embedding, which can be used to examine connections among topics and genes.

Here ARI is computed based on the Louvain clustering the cell embedding θ against the ground-truth cell types stored in the `mp_anndata` object.

Training a neural network (i.e., the VAE in our case) requires many iterations because of the gradient descent updates with small learning rate. Run the model for 1000 iterations. Record the negative ELBO loss and ARI at each iteration and then use `monitor_perf` to display the training progress (Figure 5).

Have a cup of coffee (or tea) and let it train for about 5-10 minutes. You may try the same code on a NVIDIA GPU machine to observe a major speed-up compared to a CPU machine. But that's not required in this assignment.

5 Compare ETM with NMF-Poisson (1%)

Now run NMF-Poisson also for 1000 iterations and compare the ARI scores with scETM over the 1000 iterations. While the NMF model has converged to a local optimal after only 200 iterations (Figure 6), the scETM continues to improve (Figure 5). We observe some improvement from scETM over the NMF model especially after 200 iterations (Figure 7). This highlights the benefits of having the non-linear encoder function and perhaps the tri-factorization design in the scETM. When training on massive number of single-cell samples, using stochastic gradient training on minibatches, scETM will confer much bigger improvement over the linear model. For this particular dataset, with batch-effect correction, a fine-tuned scETM can reach 0.90 ARI [2].

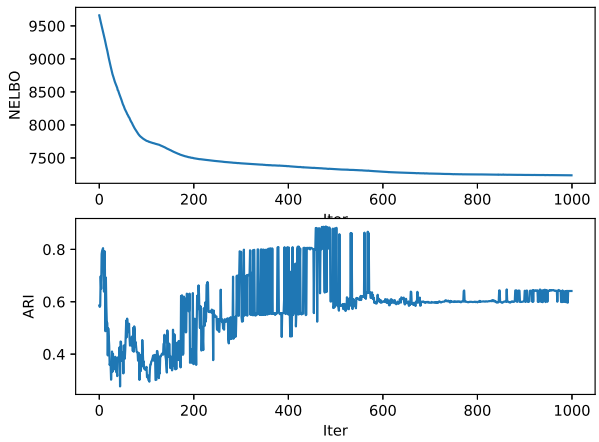

Figure 5: Negative ELBO and ARI of scETM at each iteration of the scETM training.

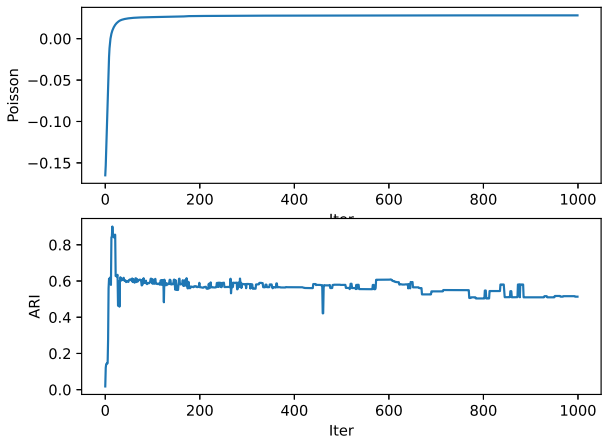

Figure 6: NMF-Poisson trained for 500 iterations.

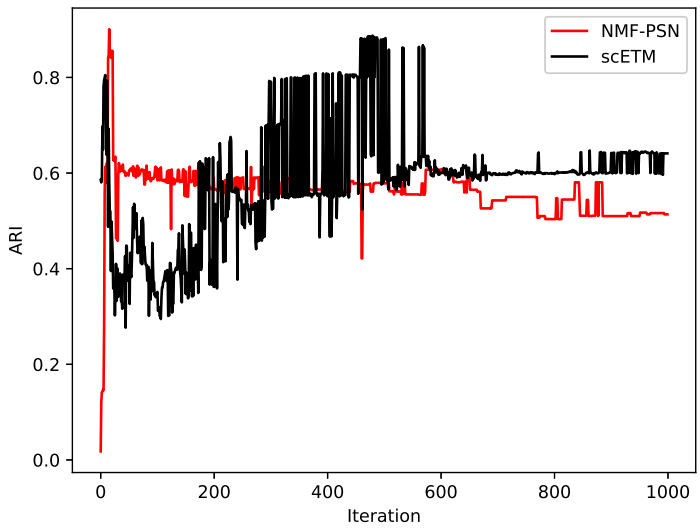

Figure 7: NMF-Poisson versus scETM in terms of ARI.

6 Generate t-SNE to visualize cell embeddings (1%)

Use the `model` object of ETM saved from the previous training over the 1000 iterations to infer final cell topic embedding θ and generate the two-dimensional t-SNE plot. Also, use the `H` matrix from the NMF-Poisson model (trained after 1000 iterations) to generate another two-

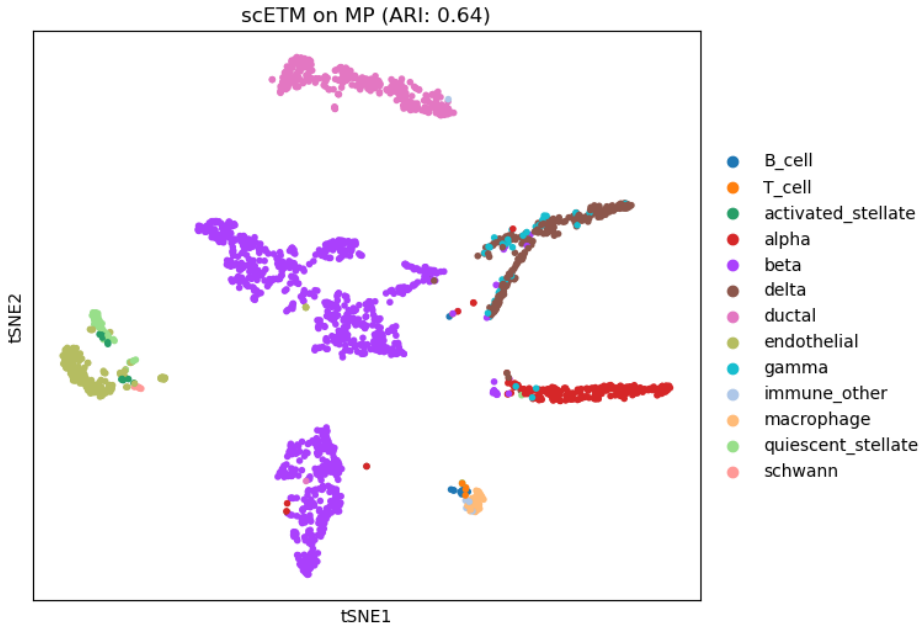

Figure 8: t-SNE plot for the MP cell-embedding using θ from the 16-topic scETM. Cells are colored based on one of the 13 cell types.

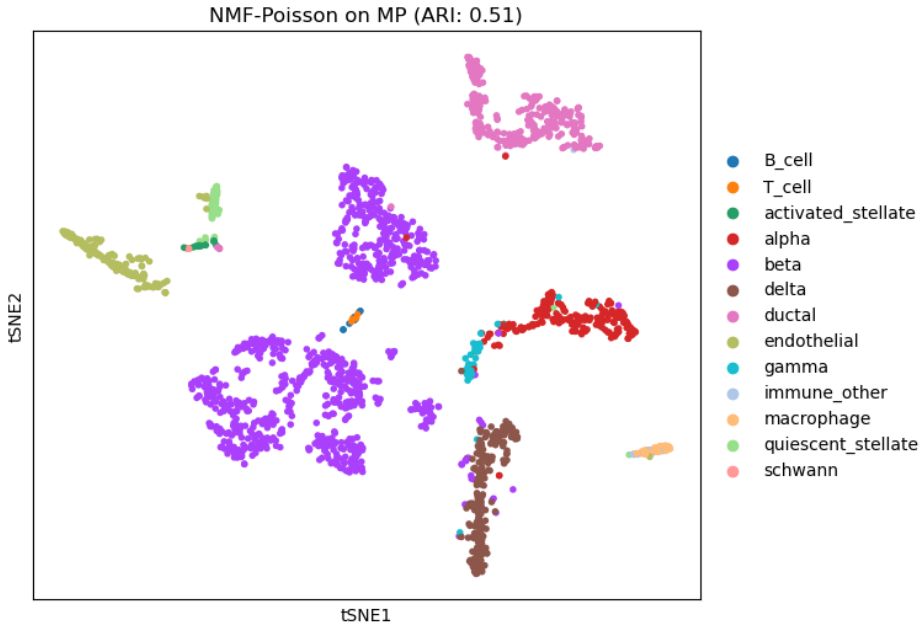

Figure 9: t-SNE of cell embedding ($H_{K\times N}$) from NMF-Poisson.

dimensional t-SNE plot. Compare them side-by-side as shown in Figure 8 and Figure 9, respectively. We observe a slightly better separation of the alpha cells from other cells from the scETM compared to the NMF-Poisson model.

The plot was generated using a `Scanpy` function called `sc.tl.tsne`. Learn how to use it from its documentation.

7 Plot heatmap for the cells under each topic (1%)

An alternative way to present the cell cluster is by heatmap. Heatmap is more effective in identifying which topics correlates well with which cell types. Generate a heatmap plot for the same cells-by-topics matrix θ using *all of the 1886 cells over the 16 topics*. Your heatmap should look similar to Fig. 10, which was generated using `seaborn.clustermap`.

From here, we see that topic 2 correlates well with alpha cell type, topic 12 with endothelial,

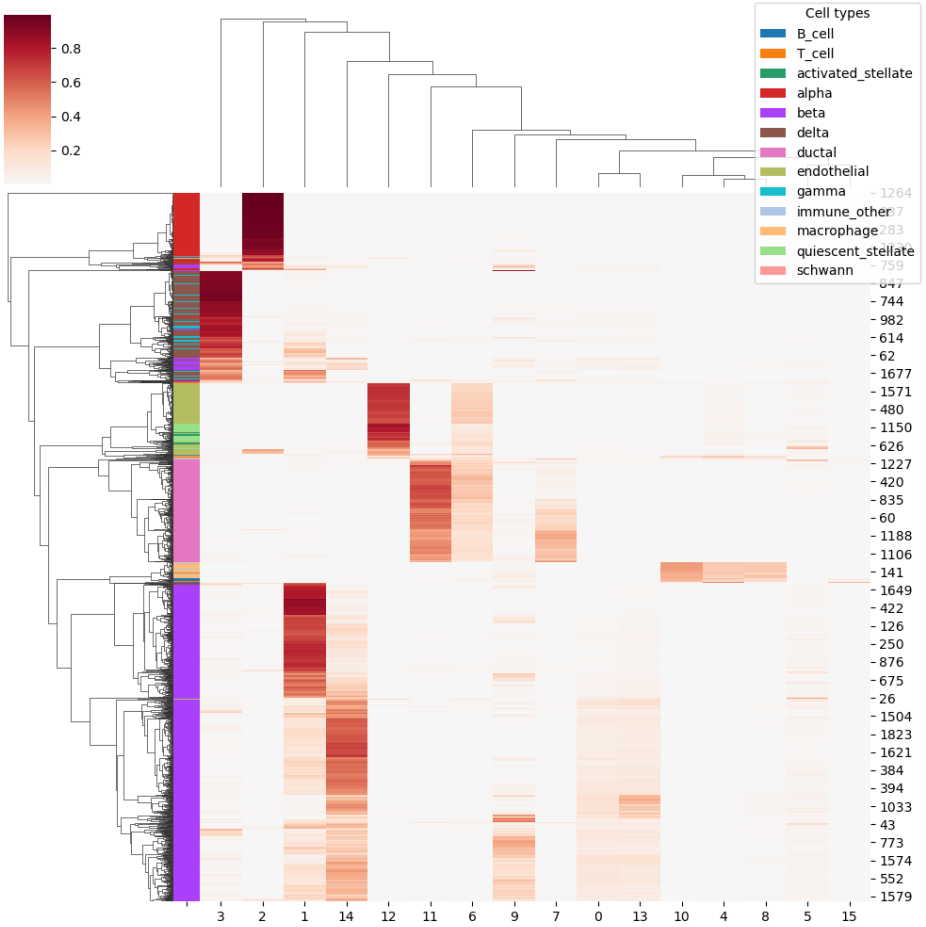

Figure 10: Cells by topics heatmap.

topic 11 with ductal cell type, and so forth. Note that you will get different topic indices matching with different cell types as the order of these topics are not the same at different runs.

8 Plot heatmap for the top genes under each topic (1%)

To get cell-type-specific gene signature, we can visualize the genes by topics heatmap. Here we will plot the top 5 genes per topic in heatmap. Your heatmap should look similar to Fig. 11,

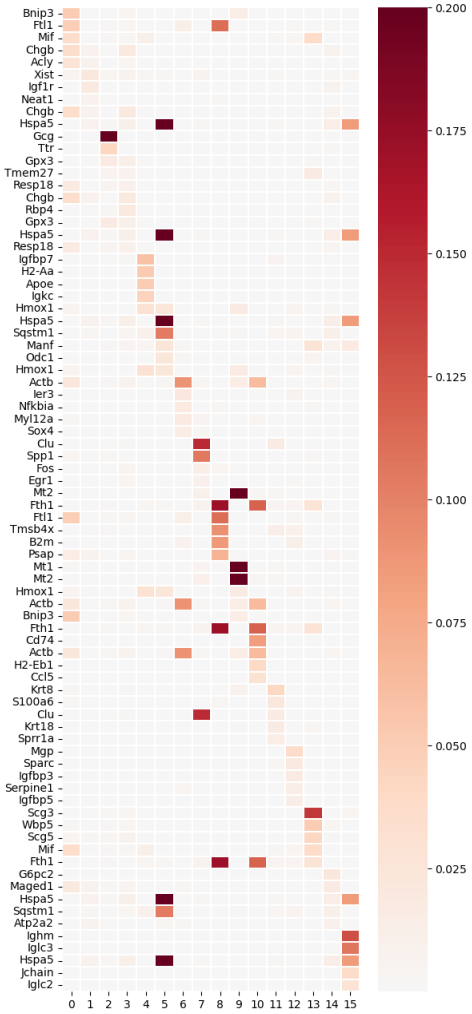

Figure 11: Heatmap of the top 5 genes per topic.

which was generated using `seaborn.heatmap`. In plotting the heatmap, I capped the max value at 0.2 instead of letting it set to 1 to make the red intensities more prominent for some of the genes with low absolute value under some of the topics.

What cell-type-specific gene signatures can you find in your analysis? For example, now we know that topic 2 corresponds to alpha cell type (Fig. 10). The gene *Gcg*, which codes for protein Glucagon has the highest probability under topic 2. Based on an online source, Glucagon is indeed a pancreatic hormone that counteracts the glucose-lowering action of insulin by stimulating glycogenolysis and gluconeogenesis!

References

[1] Maayan Baron, Adrian Veres, Samuel L Wolock, Aubrey L Faust, Renaud Gaujoux, Amedeo Vetere, Jennifer Hyoje Ryu, Bridget K Wagner, Shai S Shen-Orr, Allon M Klein, et al. A single-cell transcriptomic map of the human and mouse pancreas reveals inter-and intra-cell population structure. *Cell systems*, 3(4):346–360, 2016.

[2] Yifan Zhao, Huiyu Cai, Zuobai Zhang, Jian Tang, and Yue Li. Learning interpretable cellular and gene signature embeddings from single-cell transcriptomic data. *Nature Communications*, 12(1):5261, 2021.

[3] Adji B Dieng, Francisco JR Ruiz, and David M Blei. Topic modeling in embedding spaces. *Transactions of the Association for Computational Linguistics*, 8:439–453, 2020.