Assignment #2
BME 230B, Computational Genomics and Systems Biology
Spring 2019
Due Tuesday, May 14, 2019 at 11:59pm

You may work in groups of up to four students total on this assignment. Make sure it is clear who is on your team when you turn in your work. Please hand in a zip archive of a folder through the Canvas website.

Overview
In this homework, you will determine attempt to determine the cell types elucidated by RNA sequencing of single cells (scRNA-Seq). Because the field of scRNA-Seq is fairly new the technologies are changing dramatically and new approaches are being introduced every few months. For this reason, it is often necessary to combine data derived from different approaches. 10X genomics created two different types of chemistries to detect RNA in a single cell and applied both techniques to peripheral blood mononuclear cells (PBMCs) that contain different types of immune cells.

whether two independent scRNA-Seq experiments successfully uncover common subtypes via unsupervised Louvain clustering. In addition, you will combine the datasets together into a single dataset, attempting to remove study-specific batch effects, and then re-identifying cell type clusters. You will write an l-k-nearest-neighbor subsampling algorithm to assess the robustness of the resulting Louvain solution.

Resources
The following resources will be relevant for this assignment:

- You will be using the PBMC dataset. It contains two different batches: one where RNA's were sequenced from the 5' end (5prime) and another where RNAs were sequenced from the 3' end (3prime).
    - The data is available as a Scanpy object downloadable from the class website → [H5AD file]. You can load the dataset like this: sc.read(path_to_file). For more information on how to manipulate the scanpy object please consult the python workshop
    - The Chen et al 2018 paper that describes the results of applying the two different chemistries to the immune cell populations can be found at the following link → [PDF]
    - The data has 8098 cells measured on the 3prime platform and 7378 measured on the 5prime platform.
- The paper by Blondel et al 2008 [PDF] describes the Louvain algorithm you will implement in this assignment. You are free to look up other resources to help you construct the algorithm of course.

- All program descriptions can be found in their file
    - euclid_knn.py - link
    - euclid_bbknn.py - link
    - knn_to_graphModule.py - link- Converts an adjacency matrix to a graph structure. Use this if you want to use the Python-igraph or NetworkX library. We recommend you do this as it may make developing the Louvain algorithm easier.
    - louvain_template.py - link
- Go pathways link necessary for GSEA analysis.

Reduce the dimensionality of the data.
Before you get going on the analysis in ernest, run principal component analysis to reduce your feature space while "increasing" the biological signal. Use the sklearn function mentioned in the code snippet provided in the euclid_knn.py file above. Using the first 50 principal components, compute all of the pairwise distances between cells using the Euclidean metric.

1.a. [5 pts] Compute all pairwise distances on the 50 principal components by filling in the get_distances() function; turn in your code. Your function should return an *n*-by-*n* numpy matrix, where *n*=15,476 is the number of cells measured across both batches. Each entry in the matrix should correspond to the euclidean distance between each sample. The diagonal should be set to all zeroes.

K-nearest neighbor graphs
Although the dimensionality reduction applied above may remove some noise dimensions, it is still prudent to seek further steps that help minimize the noise even further. We suspect that larger distances between cells will be less reliable since there is more chance that noise influences the measurements compared to smaller distances. For this reason, we prefer to use a graph representation of the cell-to-cell distances that only records those distances of highly related cells and discards those from very different cells. A natural choice is to form the *k*-nearest neighbor graph in which only the top *k* closest neighbors are recorded for each cell in the dataset. This graph will then be used for all downstream analyses in place of the original RNA-seq data or a derived distance matrix.

1.b. [5 pts] Write a method to determine the k-nearest-neighbor graph (k-NNG) for a given dataset. Turn in your code.

1.c. [5 pts] Turn in a UMAP plot of your 12-NN graph calculated from the combined chemistry PBMC dataset colored by batch (the chemistry used). Run your k-NNG code on the PCA matrix of 50 top components determined from the combined chemistry PBMC dataset. Use k=12 as this value seems to produce reasonable results from our preliminary tests. The function scanpy.api.pl.umap(adata) will create a two-dimensional plot and takes as input the graph created inside the euclid_knn.py by updating the scanpy object. Note this means that you will need to run scanpy.tl.umap(adata) before plotting the 2D UMAP coordinates. Color the cells

in the plot by their batch assignment (3prime or 5prime). Hint: the batch assignment annotation is located within the adata.obs dataframe. You should see a clear separation of the batches in addition to multiple clusters present within each batch.

1.d. [5 pts] Turn in another UMAP plot of your 12-NN graph calculated from the combined chemistry PBMC dataset but colored by cell type. You can find the cell type annotations in the adata.obs dataframe as well. Cell type annotations are located in the "Cell type" column of the adata.obs dataframe.

Louvain Clustering

Recall that community detection approaches often provide a good choice for single cell datasets as they can make use of the sparse graph encoding you obtained above and provide an efficient method for identifying highly similar cells through modularity optimization. The Louvain algorithm is a fairly recent introduction that has become popular for scRNA-Seq analysis. Write your own Louvain clustering algorithm following the pseudocode provided below as a guide:

Input: $G=(V,E)$: graph representation.
Output: C: community sets at each level;
Q: modularity at each level.
Var: $\hat{c}$: vertex u's best candidate community set.

Loop ***outer***
  $C \leftarrow \{\{u\}\}, \forall u \in V$;
  $\Sigma_{in}^{c} \leftarrow \sum w_{u,v}$, $e(u,v) \in E$, $u \in c$ and $v \in c$;
  $\Sigma_{tot}^{c} \leftarrow \sum w_{u,v}$, $e(u,v) \in E$, $u \in c$ or $v \in c$;
  // Phase 1.
  Loop ***inner***
    for $u \in V$ *and* $u \in c$ **do**
      // Find the best community for vertex u.
      $\hat{c} \leftarrow \underset{\forall c', \exists e(u,v) \in E,\, v \in c'}{\text{argmax}} \Delta Q_{u \to c'}$;
      if $\Delta Q_{u \to \hat{c}} > 0$ **then**
        // Update Σ_{tot} *and* Σ_{in}.
        $\Sigma_{tot}^{\hat{c}} \leftarrow \Sigma_{tot}^{\hat{c}} + w(u)$; $\Sigma_{in}^{\hat{c}} \leftarrow \Sigma_{in}^{\hat{c}} + w_{u \to \hat{c}}$;
        $\Sigma_{tot}^{c} \leftarrow \Sigma_{tot}^{c} - w(u)$; $\Sigma_{in}^{c} \leftarrow \Sigma_{in}^{c} - w_{u \to c}$;
        // Update the community information.
        $\hat{c} \leftarrow \hat{c} \cup \{u\}$; $c \leftarrow c - \{u\}$;
    if *No vertex moves to a new community* **then**
      exit ***inner*** **Loop**;
  // Calculate community set and modularity.
  $Q \leftarrow 0$;
  for $c \in C$ **do**
    $Q \leftarrow Q + \frac{\Sigma_{in}^{c}}{2m} - \left(\frac{\Sigma_{tot}^{c}}{2m}\right)^2$;
  $C' \leftarrow \{c\}, \forall c \in C$; print C' and Q ;
  // Phase 2: Rebuild Graph.
  $V' \leftarrow C'$;
  $E' \leftarrow \{e(c,c')\}$, $\exists e(u,v) \in E$, $u \in c$, $v \in c'$;
  $w_{c,c'} \leftarrow \sum w_{u,v}$, $\forall e(u,v) \in E$, $u \in c$, $v \in c'$;
  if *No community changes* **then**
    exit ***outer*** **Loop**;
  $V \leftarrow V'$; $E \leftarrow E'$;

2.a. [10 pts] Turn in a copy of your code. Your method should take in the 12-NNGs that you calculated above. Note that the pseudocode above returns solutions at all levels while your method should only produce a single level, the one that you obtain when no new communities are formed and the outer loop is exited. Although not required, we recommend using the iGraph or NetworkX module to develop the Louvain algorithm. You can use the provided knn_to_graphModule.py to create an igraph or networkX object of the k-NNG you previously made in 1.a.

Cluster both of the batches in the PBMC dataset using your own Louvain algorithm. Feel free to compare the results to running the Louvain algorithm provided by Scanpy (you can see examples of it in the scanpy tutorial under Clustering. Note that this scanpy louvain method is actually a slightly modified version that has an additional parameter called the *resolution* that can determine the coarseness of the community structure. Use *resolution*=1 to make it compatible with the original Louvain optimization function.

2.b. [5 pts] Turn in a UMAP plot of the combined dataset as you did in question #1, but this time, color the cells by their Louvain cluster assignments determined for each cell within each batch as a different color in each plot.

Next, determine how the resulting clusters correspond to the expected cell types. Using the annotations provided for the datasets, perform a hypergeometric overlap analysis to determine which annotation set best overlaps with each of your clusters. Each cluster in each dataset C_i contains a set of cells and each cell type annotation A_j also contains a set of cells. The intersection of cluster C_i with annotation A_j results in a possibly empty set of cells O_{ij}. Use the hypergeometric test to determine if the overlap O_{ij} is significant for all pairs of clusters and cell type annotation sets for each dataset. Then, for each cluster, keep the annotation that has the most significant P-value.

2.c. [5 pts] Turn in a table that lists each cluster and its best-matching cell type annotation. The table should contain the cluster number and its best matching cell-type annotation based on the hypergeometric analysis.

2.d. [5 pts] Turn in a list of top 5 pathways for each cluster in each dataset. You should use the gene expression signature of each cluster to find an associated pathway. A gene signature for a cluster represents the gene expression levels for a characteristic cell that is a member of the cluster. Use the centroid μ_i of the *i*th cluster as the signature. Compute the centroids for each cluster in each dataset. You will next derive a gene-signature based annotation for each cluster using these centroids. Use a list of Gene Ontology Biological Process categories (provided in the Resources section at the top of this homework) and your signatures to perform an all-against-all Gene Set Enrichment Analysis (GSEA). Turn in a table that lists the top 5 pathways for each cluster.

Integrating two datasets using batch-balanced k-nearest neighbors

The previous analysis points to a major problem with the analysis of the combined dataset -- there is a significant batch effect due to the different sequencing chemistries employed. One approach for combining batches is to form a graph representation that attempts to force cells to align with cells from multiple batches in an equal way. A “batch balanced” k-NN graph (bb-k-NNG) forces each cell to have the same number of neighbors from every batch. To form a bb-k-NNG, each cell is connected to its top *k* neighbors for each batch. For example, a *k*=6

bb-k-NN for the combined PBMC dataset would find the top 6 neighbors in the 3prime batch and the top 6 neighbors in the 5prime batch for each cell.

3.a. [5 pts] Write code to form a bb-k-NNG based on the two chemistries (5prime and 3prime). You should fill in the methods of the class named bbknn_graph contained in the euclid_bbknn script. Turn in your code. Compute a bb-k-NNG over the PBMC dataset using k=6 to use for the next clustering step.

3.b. [5 pts] Cluster the integrated dataset using the Louvain method. Re-cluster the data now that you've attempted to remove the batch effect. Turn in a UMAP plot showing the integrated dataset and color the cells in the plot by their Louvain cluster assignments.

We need to assess how well cells are clustering by batch compared to by cell type before and after using the bb-k-NNG. Let's use the UMAP coordinates to quantify this notion. To do this, we will form two quantities representing the mean squared error between cell types (MSE_C) and the mean squared error between batches (MSE_B). Each of these quantities is defined below:

- $MSE_C = 1/(T(T-1)) \sum_{t=2..T} \sum_{s<t} \sum_{w \in \{x,y\}} (\mu_{tw} - \mu_{sw})^2$,
    - where T is the total number of cell types, t and s index over the types, μ_{tx} is the x- and
    - s the y-positions describing the centroid of all cells of type t in the UMAP solution.
- $MSE_B = (1/4) \sum_{b=1..2} (1/m_b) \sum_{i \in B(b)} \sum_{w \in \{x,y\}} (x_{iw} - \mu_{bw})^2$,
    - where m_b is the number of cells in each batch, $B(b)$ is the set of cells in batch b, x_{ix} *is* the x- and x_{iy} is the y-coordinate of cell i in the UMAP solution, and μ_{bx} is the x- and μ_{by} is the y-position of the centroid of all cells in batch b.

The F-statistic to use for measuring how separated the clusters are compared to how separated the batches are is then:

$$F = MSE_C / MSE_B$$

We prefer higher F statistics as these correspond to solutions with higher variance across the cell types (larger separation) compared to across the batches. The degrees of freedom for MSE_C is T*(T-1) = X 9 cell types and the degrees of freedom for the MSE_B is $(n-2)$ where n is again the total number of cells, 15,476.

3.c. [5 pts] Quantitatively estimate the degree to which the bb-k-NNG removed the batch effect using the F-statistic described above. Calculate the F statistic using the UMAP solution derived from the original, non-batch balanced 12-k-NNG. Then calculate the F-statistic

using the bb-6-NNG to make the UMAP solution. Report both F-statistics. Do you see an improvement in the batch correction using the bb-k-NNG?

Robustness analysis with batch balanced l-k-nearest neighbor graphs (bb-l-k-NNGs)

We would like to determine the stability of the above clustering solution. You will assess how the clustering solution various across subsamples of the bb-k-NNG. To do this, borrow the technique described in the Wanderlust algorithm that creates perturbed versions of the bb-k-NNG and then present these subsampled versions to the Louvain clustering method.

An l-k-NNG is a subsampled version of a k-NNG. The first index, l, represents a number less than k and specifies the number of neighbors to subsample for each vertex when building the subsampled graph. To generate a l-k-NNG from a given k-NNG then you simply visit each vertex and randomly sample l out of its k neighbors and keep those that are chosen.

Our slight variation here is that we would like to preserve the influence of the different batches in our subsampling of a bb-k-NNG to make a bb-l-k-NNG. In the subsampled bb-l-k-NNG then, you should keep the number of neighbors from each batch equal. For example, when constructing a bb-3-6-NNG each new vertex in the resulting subsampled bb-3-NNG will have 3 neighbors from batch 1 and 3 neighbors from batch 2. Note that the input graph would contain 6 neighbors to choose from for batch 1 and 6 to choose from for batch 2.

4.a. [5 pts] Turn in code that can compute a bb-l-k-NNG from a given bb-k-NNG. In addition to the input bb-k-NNG, the method should take in a parameter specifying the number of subsampled graphs that should be returned. Each of the returned subsampled graphs should represent a distinct sub-sampled bb-l-k-NNG from the given input graph. Fill in the l_k_bbknn method in euclid_bknn.py. You can set the *l* parameter accordingly. **NOTE:** in order to run this function successfully you will have to run the bbknn() method first, and then run the l_k_bbknn method. Consult example usage within euclid_bbknn.py.

To measure the robustness of the Louvain clustering of the integrated data, use your bb-l-k-NNG algorithm. Let the clustering result obtained from the full bb-k-NNG be called C^0. The idea is to create a bb-l-k-NNG subsample, run the Louvain algorithm using it as input, and then measure how similar the clusters are compared to C^0. To measure the similarity of the solution you should use the Adjusted Rand Index (read the Wikipedia article). The ARI measures the degree to which each pair of cells (i,j) are similarly clustered together (or not) between two solutions.

4.b. [5 pts] Turn in a bar plot of the Adjusted Rand Index (ARI) for Louvain clusters obtained from 10 independently subsampled bb-3-6-NNGs compared to the Louvain clusters obtained on the original bb-6-NNGs. Also report the average and standard

deviations of the ARI. Based on these results, would you conclude these clusters are robust? Justify your answer. Hinr