Data analysis

Lesson 8 – Prof. Calogero

Correction of the exercise of the previous lesson → read the notes of lesson 7.

CLUSTERING

I need something to assign the features of differentially expressed genes in bulk RNA. The steps that have been done until now are: evaluation of the quality of the fastQC → calculation of the count table → evaluation of the sample distribution with PCA → differential expression analysis (which has to be done) → try to associate some biological functions to the differentially expressed genes (this will be done the next time).

In case of sc: evaluation of the quality fo the experiment → dimensionality reduction: data are represented in an organised and handling space. This part has been already loaded, and it is made of 5 lung cancer cell line → I want to investigate the ability of the tool to separate cells in different groups without knowing what they are. Data reduction is done with UMAP, then I have to find a way to assign clusters (or groups) to the genes I have. For the exercise, I have to use a clustering method that requires their use (?) to assign the number of clusters (I must have an idea about the number of clusters I need). Other methods do this by themselves, but I will work on a limited number of data because of the limitations of my computer. Data reduction provides a plot and, by eye, I can decide what are the groups; this is a biased approach. On the other hand, some algorithms work automatically with mathematical formulae.

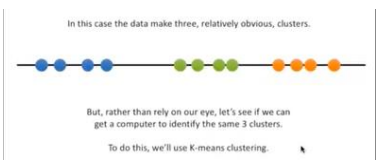
There are 2 critical points in clustering sc data:

1) We don't know how many subpopulations are present in the dataset
2) We don't know which are the optimal partition parameters to be used.

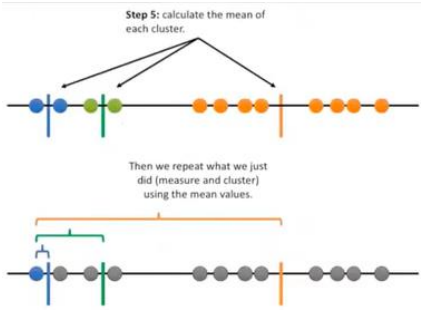
In the slide, the division is made according to the material of the bottles; however, data can be organized according to what is present in the container (beer, milk, H2O) or according to the shape of the container. None of these partitions is wrong, but the parameters used for the partition are different. In sc, I assume that the information written in the gene is enough to discriminate among cell types present in the dataset. The criticality in clustering is that I don't know what are the parameters that drive the separation of the data. When I have a dataset and I don't know how it is organized, I have to find a way to obtain clusters; working with subset of genes that are characteristic of cell types can help in further data characterization. Initially, I try to assign a partition that is given by the transcription information that is written in my cells. The system works well if the separation if big (= if there are many cell types), while it works worse if I want to separate subpopulations of the same cell types or of similar cell types. T cells, for example, are difficult to be separated because of a similar transcription profile and a few genes distinguishing each other.

There are 3 groups of approaches:

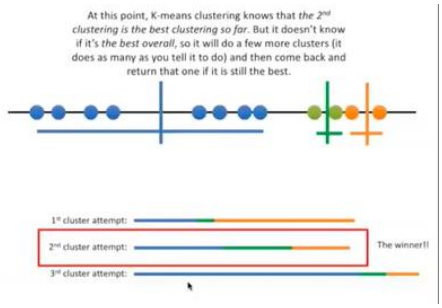
1) Based on k-mean
2) Based on Louvain modularity
3) Based on the dataset splitting and on aggregating clusters according to the split dataset.



K-mean clustering can be done on the output obtained from UMAP. Let's see its approach. In the slide, gene expression is clustered in 3 groups. How can a software detect 3 clusters? K-mean clustering assumes that I know how many clusters I am going to take into consideration. Foe example, I know that 3 is a good number, so I use 3 as a starting point. K-mean clustering selects 3 random values in the dataset (the blue, green and yellow dots), which represent the starting points. Then, it looks at the distances between the genes and the cluster centers that have been set up: genes are clustered according to the



distance from the "reference" point. 3 clusters are obtained, but they are very bad wit respect to the expected one. For each cluster, I calculate the mean and re-evaluate if the genes are within the cluster, being the center near to the various points. So, I analyse the distance between each gene and the center: if the distance does not change, the cluster is done; if it changes, data have to be re-organised and re-clustered until the gene(s) don't move from one cluster to the other. At the end, the cluster I generated is different from the one I saw by eye. Then, I calculate the variance that is assigned to all the clusters (which depends on the dimensions of the cluster). The blue and the green clusters are very good, while the yellow one is bad because it is large. So,



what can I do? I start again: I select 3 new random numbers, I re-assign the genes to the centroids according to their distance from the center, I calculate the mean and assign the genes according to their nearest position with respect to the mean → I obtain 3 clusters, which are very similar to what I saw by eye at the beginning. Finally, I calculate the relative variance associated to each cluster. I do these steps multiple times, I rank the overall variance of all the clusters I obtained and select the cluster having the smallest variance. If I do the procedure 3 times, the second cluster has the smallest overall variance among the other clusters. The idea is to make approximations on the data separation because I don't know where my clusters are. The analysis must be repeated many times to find the aggregation of cells characterised by the smallest variance. Limitations emerge when data are more compact than the one represented in the slide: during clustering, a lot depends on how big is the separation between the various elements I'm interested in.

How do I decide what is the best number of clusters? Ex: I start with K=1 → I calculate the variance → the dispersion of data is going to be very bad. K=2 → better separation. K=3 → the best separation. K=4 or K=5 → the distance among clusters continues to shrink, and the optimal situation is represented by having one cluster for each gene: the distance is going to be very small and the partition seems to be the best. Actually I am assigning each gene to a specific position. The overall idea for the selection of the best K-mean derives from the Elbow plot, which tries to see how big is the reduction of the differences that I see: if my cluster is characterised by the optimal aggregation, I obtain the minimal point in which I have the separation. After that, the slope of the curve changes. From 1 to 3 I have a progressive reduction of the variation of the size of the cluster, which becomes smaller in cluster 3. The slope of the curve indicates which is the best number of K (in this case, 3). The Elbow plot work well with different data: monocytes, NK, B cells are very different → good separation. T cells, NK T cells → bad separation, because cells belong to the same category (T cells) → no slope change and no identification of the optimal number of clusters.

<u>Exercise</u>

Exercise

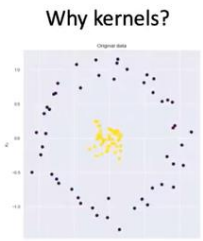
- Apply UMAP/tSne on the dataset provided in this lesson:
  - A mix of five human lung adenocarcinoma cell lines:
    - 1242 cells of A549,
    - 436 cells belong to H1975,
    - 749 cells of H2228,
    - 879 cells belong to H838,
    - 598 cells of HCC827 cells.
  - The dataset count table has cell names and cell lines associated, e.g. Lib90_00000.HCC827, Lib90_00002.H838.
- Apply k-mean clustering using Bioconductor package mbkmeans.
- Using ggplot2, plot clustering results on UMAP output data.
- Apply k-mean clustering on the UMAP output using the package ClusterR
- Using ggplot2, plot clustering results on UMAP output data.

Data are imputed already. Imputation tries to moderate the problem of the drop-out events that are typical of sc: the gene is in a position, but I cannot detect it because it does not stick on the beads I use for the separation. Imputation borrows information from the various genes, and tries to reconstruct what is the expected transcription profile of the cell. SAVER is RAM-consuming and it cannot be run on the computer → data are already imputed.

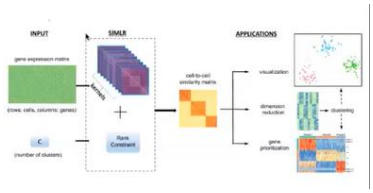
5 cell lines are used, but each one has subpopulations inside → 5 clusters may not be the optimal solution.

1) I must to a plot with UMAP, which generates a table telling me where cells are located. It just does dimensionality reduction. Then, I must work with 2 approaches:
2) Approach #1 (which will probably give me the best result): I install the ClusterR package and apply k-mean on the table containing the cell on the row and the 2 dimensions from UMAP on the columns. In this case, I use dimensionality reduction as a starting point to see if I can obtain a cluster from these data. Finally, I have to select the potential number of clusters (suggestion: start with 5). Then, cells have to be coloured according to the cluster they belong to. This must be done by generating a file that I can use with ggplot2. Row = cell; column = UMAP coordinates.
3) Then, I detach the cells according to their names and I put them in another column: each cell name has the cell line it belongs to. The new column should be a factor that separates the subsets of cells according to the features I know: this is going to be my control. So, I make a plot that colours cells according to the cell line they belong to; then, I make another plot which colours cells according to the cluster that has been learnt by the K-mean approach. So, the next column is the assignment to different clusters, which is done with ClusterR. The reason why I have to use factors for the cell name is because factors are numbers, and colours are used on the basis of the cells that are associated to a factor number. These cells can be coloured to be better differentiated.
4) Approach #2: use another K-mean tool which works on the initial table (not on the reduced data), on which I do clustering. Then, I overlay the clustering obtained with the UMAP results, to see if this clustering fits the dimensionality reduction approach. The results can change between approach #1 and #2, but in this case they may be similar because cells are different from each other. Colours can be given by the cluster names (which are numbers) or by the cell lines when they are used as factors.

<u>SIMLR</u>



k-mean is relativelt simple to be understood, and it is used also by SIMLR. The problem is that sc is very noise, so a simple dimensionality reduction may not always help in identifying the differences that exist between the various cells. SIMLR tries to do this in a different way, transforming data before doing data reduction. The transformation is made according to the selection of the Kernel that better fits the number of clusters that I would like to generate: I actually represent cells in different formats to select the one that better fits one of the 3 clusters. This dataset is difficult to be separated: yellow cells are in the middle of the blue cells → there is no way to obtain a good separation. If I change the representation and I get a sort of projection (remember the concept of tSNE, where the positions of the cells are projected on a graph), I do a kernel: data are transformed in order to be related to a normal distribution. Kernel = projection in a different space, which can be a normal distribution, a t-distribution or whatever. In the slide, data are represented in another way after transformation. In order to transform data, I can select the PCA that provides the best separation which fits the number of clusters that I want. The PCA may explain even a piece of the variance, but it must represent all my 3 clusters. Moreover, I use more than one kernels finding the one that converge to the same clustering results.

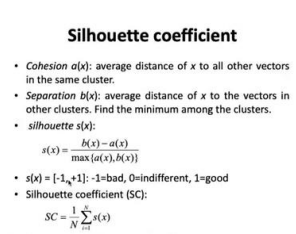


As seen in the slide, I have many projections of the input data and the number of clusters that I would like to generate. Then I look at the similarity among cells over the various kernels and I select only the subsets of kernels fitting the 3 clusters. Then I do dimensionality reduction with tSNE or UMAP and I add the k-mean clustering. The clustering is always the same, but I use a better set of data that has been selected to fit the number of clusters that I want.

In order to select the optimal K, the Elbow plot can be a good solution, but it is not perfect: as well as for the "perfect" number of clusters, thousands of methods are designed to find the optimal number of K. An example is the Silhouette coefficient, used to define the optimal number of clusters. The optimal number of clusters provides the best compromise between clusters derived from similar and very different cells.

silhouette tries to explain these features: it goes from -1 (bad clustering) to 1 (good clustering), considering two elements: cohesion among the cells belonging to the same cluster with respect to the separation of the cells in a cluster and the ones in other clusters. These 2 elements define the Silhouette coefficient. A tight cluster is associated to a good cohesion, which in turn must be associated to a good separation among clusters. Example: I have a large cluster near the other ones → cohesion is bad. If the distance between clusters is small with respect to the cluster I consider good → bad cohesion.

Here I have a graphical representation of these concepts: cohesion is the relative measure with respect to all the points that are in the cluster; separation indicates the distance between the points present in the cluster.



The coefficient can be calculated from the centroid (the average distance between all the points present in the same cluster) → I calculate the distance centroid – cells → I select the minimal set of distances between the cells and the centroid. Then, I make the calculation (separation information – the cohesion information)/the maximal value of these 2 parameters = Silhouette score, which goes from -1 to 1.

Sc is characterized by being very noisy: I take my cells and separate them → I go again to the initial dataset and I take out the 10% of the cells (which is a good percentage in order to not perturb the dataset so much) → if the cells I removed were not important, the cluster will be the same, otherwise the cluster was not robust enough. This means that I can associate, to each cell, a score of how many times it stays in the same cluster together with the other cells; this score tells me which clusters are unstable, due to the fact that cells move from one cluster to others.

Louvain modularity

Griph and Seurat decide by themselves how many clusters are going to be generated. They work in a similar way to hierarchical clustering (they start from all the elements disaggregated until they become all aggregated). The idea of these methods is related to the level of connectivity among nodes: higher the connectivity, higher the association of two elements to the same group. They compare the connectivity in a random set of nodes with the same size with respect to the dataset that I have.

- Heuristic indicates that I don't have an optimal solution: I have to find another one fitting the real one. Numerical solutions are difficult to be obtained because they require too much computing time or they are impossible to be considered. Heuristic provides the best solution, even though it is not identical to the real one; however, it is a way to obtain a good solution in a reasonable amount ot time.
- Greedy algorithm is an heuristic approach that tries to find the best choice to simulate the global solutions, which are numerically impossible.
- Coarse-grained models are simplified.

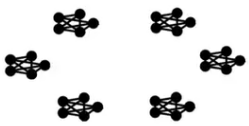
In these algorithms there are 2 steps:

1) Assignment of the nodes to community: greedy tries to find different edges to aggregate different nodes to see what is the best set of cells that stay together initially. Since I do a heuristic analysis, it s probable that I only get a partial picture of it.
2) I transform the aggregation in a model in which my aggregation becomes a single model → this representation is going to be placed again on the first step. Example: I have 2 clusters made of 5 cells each; I aggregate cells, looking at the edges. How are these cells connected? Via one element only.

An important concept is the one of "clique" = a subset of vertices that are fully connected by edges. This concept is needed to evaluate the connectivity between nodes of the Louvain modularity, which uses this

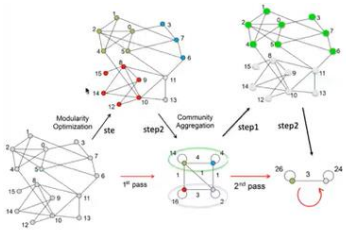
information to calculate how many nodes are connected to the others, being the number of connections the most important part of this community aggregation.

Example



I have 6 communities, each one made of 5 cells. Firstly, I connect the communities using random edges to generate a close network containing my data. 1st step of the analysis: search for the optimal way to connect all the elements together. As seen in the picture, 10, 11, 12, 13 and 14 are connected together. What happens if I take the 11th element and move it to the next cluster? Is the connectivity improved or worsen? What happens if I move other elements to other clusters? Greedy tests various elements until it finds a reasonable LOCAL solution. Finally, there are 6 communities. 2nd step: the communities are represented by the number of edges of each community. The number in the black dot does not mean anything (all the elements collapsed in a unique one), and the elements are connected by 9 edges. Now, whenever I attach the various elements, the only thing that changes is the increment of 1 in the various nodes.

Example



Step 1: I start to find the optimal aggregation, and I obtain the red, white, green and blue communities. How many edges represent these communities? Instead of calling 6 cells, I will call a unique cell with the number of edges that are connecting the cells. The elements are connected by various edges. The colour of the elements changes according to the highest number of edges connecting different communities. Next step: the white and green groups are finally aggregated by 3 edges, until anything can be aggregated anymore. The advantage of this system is that the communities are decided by the algorithm itself, and it is not anymore a problem of mine to define the K number.

Seurat uses Louvain modularity again.