8 - Clustering

TOPICS	K-means clustering Louvain modularity-based clustering
DATE	@May 4, 2022 2:00 PM
LAST EDITED BY	
LAST EDITED TIME	@May 16, 2022 9:02 AM
MADE BY	
PROF	Calogero
Recording	
STATUS	☐

SUMMARY

For bulk analysis we have done the quality control analysis, calculating the count table, performed the PCA dimensionality reduction and the differential expression analysis to identify the differential expressed genes comparing two conditions (treated and untreated). Next time we will try to associate some biological function to the genes.

For single-cell analysis we have done the quality control analysis, we calculate the count table, performed the dimensional reduction with tSNE or UMAP and then we have to performe the **clustering.**

Clustering

Clustering is used to identify cell subpopulations in single cell RNAseq data. For SC, clustering can be used to identify the different subpopulations present in the sample.

For clustering we don't know two parameter: the number of cluster and the optional parameter to use. The parameter that can be use for partition are different.

There is not just one correct clustering approach: the elements in our dataset can be clustered according to multiple parameters. But we do not know which are the most appropriate ones, i.e. the ones driving the most relevant differences among the data.

Doing clustering we assume that the info contained in the gene is enough to discriminate the different subpopulations. Clustering algorithms work well when the differences in the populations are big: the more the cells are similar in their transcription profile, the more difficult the analysis.

We will see 3 groups of approaches: K-means, Louvain modularity or Random projection based.

K-means clustering

https://www.youtube.com/watch?v=4b5d3muPQmA

▼ Last year transcript on K-means

There are an enormous amount of clustering tools. There are many ways to do clustering and can be added to different tools. The critical point is finding the same matrix that helps you to understand how your clusters are built and how they are really homogeneous or not. Unfortunately, we use transcriptional profiles (a set of information) in order to reorganize the data, but we are no guarantee that the way of the data is reorganized is the way that would like to have the data. The computer algorithm needs to find by itself the possible cluster. K-mean clustering requires that the users know how many clusters he would like to use. In our example K=3 and it takes three random points over the space. The three points are going to be the center of the potential cluster. Then the algorithm starts to calculate the distance between a cell and all the centers of the various random clusters and assign the cell to the cluster that has the smaller distance. And repeat this mechanism to each of the cells. After the first assignment, it takes the mean of the cluster and calculates again the distance between the cell to the mean and reassign the cell to the specific cluster. What happens is that it recalculates the distance between the mean and the cell and reassigns to the various clusters until the cell does not change clusters anymore. So, the system tries to find the best position for the center of the cluster that we are assigned. The critical point is: if the centers are wrong initially, the adaptation that it can do moving a little bit the mean of the center of the cluster is very limited. Then we store the information of the quality of the cluster by taking the maximum variation that exists between the elements of the cluster. But this thing is not done one time but many times (thousand or ten thousand) changing the position of our starting point in our space. At the end we take all the stored information among the variance and we can get the best variation within the cluster. The important point is to do enough interaction to be able to find the optimal solution. Another critical point is that it needs to know how many clusters have to make. There is a way to define the optimal number of clusters. If the number of clusters increases the amount of variance associated to each cluster decreases and we obtain the minimum shrinking when the cluster corresponds to a cell. Is possible to see an elbow plot that is generated after the optimal clustering condition is reached. The amount of reduction in variation grows very rapidly until you get the optimal number of clusters and then the growing with more clusters is getting more flunk. In principle this system works, but in reality many times doesn't work. There are many other approaches to calculate the optimal number of clusters and we use another clustering algorithm to do that. There are the same advantages to K-mean clustering because we can control the number of clusters.

You have gene expression going linearly on a line. You see by eye that there are 3 clusters.

Start with a set of dots that indicate the expression.

How can a software detect that there are three clusters?

With all K-means-based clustering approaches, the user is required to provide the number of clusters (K) to be generated.

The software selects totally randomly a number of points identical to the # of clusters you want, for example 3. Then it measures the distance of each other datapoint to these three cluster centroids → assigns those data points to the closest starting point.

The starting point is the selection of 3 random value. And are the starting point for the 3 cluster. Then it calculate the distance between the center of the cluster and assign the point to the cluster.

Then it calculates the mean for all the clusters and then it evaluates if the genes previously assigned would still be assigned to the same cluster.

This process is reiterated till every points remains assigned to the same cluster after the cluster mean calculation.

You then calculate the variance (how much the points are dispersed) assigned to every cluster: a good cluster is very tight and has << variance. You select 3 new random numbers and you repeat everything as before. In this second iteration we have 3 clusters that are very similar to the one we see by eye.

We calculate again the variance and we repeat (usually 10k times for normal biological datasets). At the end we rank the overall variance of the clusters and select the best cluster (i.e. the ones with the smaller variance).

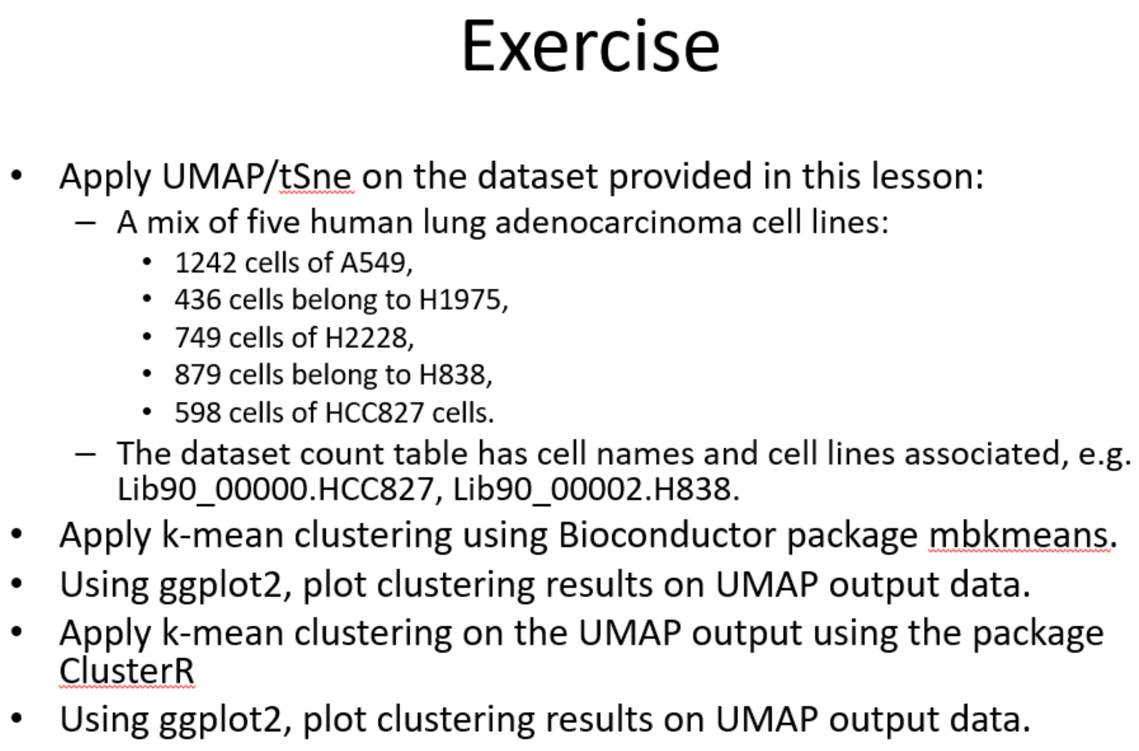
The whole process is repeated several times until no dots move anymore. At the end, the overall variance of the clusters attempts and then the one with the lowest variance is selected.

Limitations → as the data become more complex, the iterations required are ++. Moreover, the bigger the dataset, the more overlapping the clusters will be and hence the differentiation between different clusters will be harder.

Elbow plot for the definition of the K → you measure how big is the cluster:

If the cluster is optimally aggregated, there is the best separation with the other datapoints → after this point, called elbow point, the variation does not go down very fast.

K-Means exercise



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.

> ⚠️ the dataset is already on moodle called lesson8 dataset

First we have to perform data reduction with UMAP and tSNE (they should be 5 clusters, so you know from the beginning). Then we have to use a clustering approch that is unbiased.

> 📌 The problem is often to select the optimal way to do the partition of the data. Data reduction provides a plot and by eye you can decide what samples of the UMAP graph compose a cluster, but this is biased.

First you have to do data reduction mainly with UMAP. Then you have to cluster, for the clustering we will use a methods that require that the user decide and assign the number of cluster.

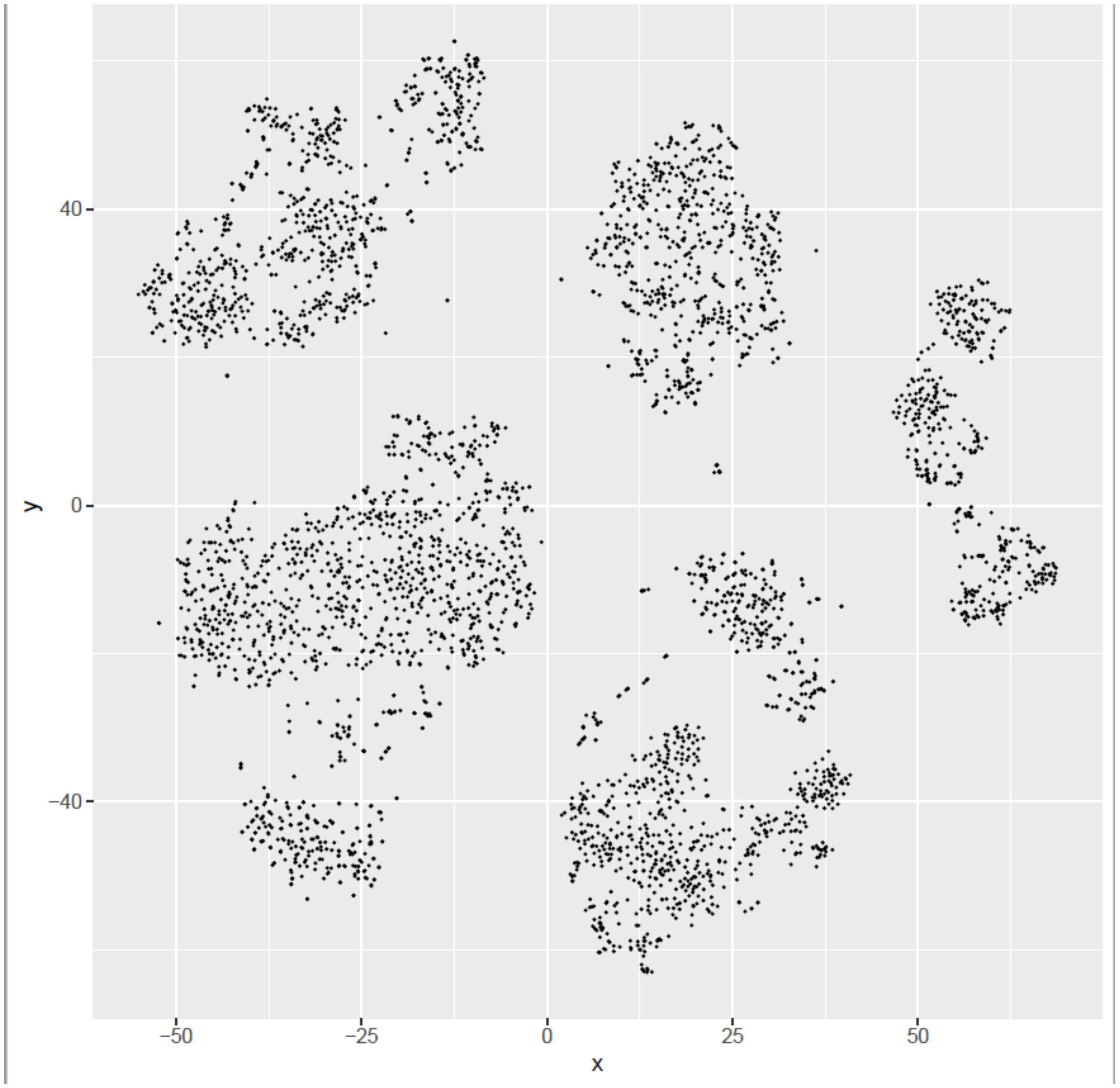
We do clustering on simple datasets because our PC are not enough to perform normal analysis. The dataset for the exercise 8 contain a **saver imputed (**moderates the drop-out events of the single cell sequencing process) file contaning the data from a scRNA-seq experiment on five different cell lines. The data are already imputed with SAVER because the SAVER analysis cannot be done in our computers since it is quite RAM-consuming. This dataset (population of cells) have already some population inside (sub-population).

- Apply UMAP/tSNE on the dataset and plot the results

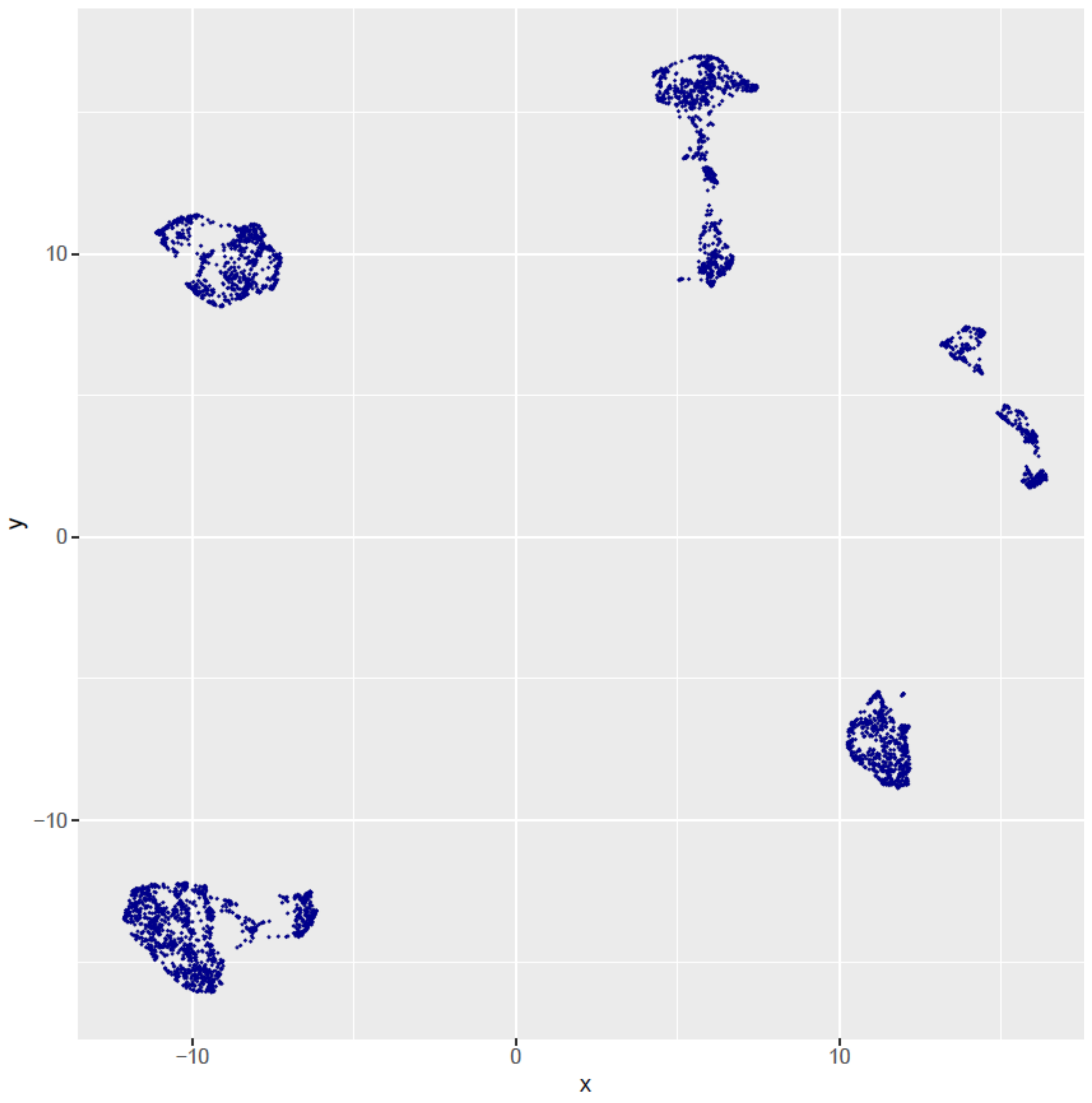
  ▼ **UMAP and tSNE without colours**

  Before to run tSNE or UMAP you have to perform the conversion in `log2cpm`. Then you have to perform tSNE or UMAP and finnaly use the `ggplot2` package to plot the results of UMAP and tSNE.

```
#before to run tSNE generate the log2cmp file
counts2cpm <- function(file, sep = ","){
  tmp <- read.table(file, sep=sep, header=T, row.names=1)
  col.sum <- apply(tmp, 2, sum)
  tmp1 <- t(tmp)/col.sum
  tmp1 <- t(tmp1)
  tmp1 <- tmp1 * 1000000
  write.table(tmp1, "cpm.csv", sep=",",
              col.names=NA)
  write.table(log2(tmp1 + 1), "log2cpm.csv",
              sep=",", col.names=NA)
}

# Recall the function and run it on your ctrl file
counts2cpm(file="saver_RNA-5c.csv", sep=",")
file.rename(from="log2cpm.csv", to="RNA-5c_log2CPM.csv")

##load library
library(Rtsne)
library(ggplot2)

#prepare data to run tSNE
tmp <- read.table("RNA-5c_log2CPM.csv", sep=",", header=T, row.names=1)
tmp.labels <- sapply(strsplit(names(tmp), '\\.'), function(x)x[2])
cell_line <- as.factor(tmp.labels)
set.seed(111)

# We do not want any PCA step before doing tSNE -> set pca parameter pca =FALSE
tsne_out <- Rtsne(as.matrix(t(tmp)), pca=FALSE, perplexity=30,
                  theta=0.0)
t=data.frame(x = as.numeric(tsne_out$Y[,1]),y = tsne_out$Y[,2])

# plotting tSNE
sp <- ggplot(t, aes(x=x,y=y,)) + geom_point(pch=19, cex=0.3)
pdf("tSNE_RNA-5c.pdf")
print(sp)
dev.off()
```



```
#load library
library(umap)
library(ggplot2)

#run UMPA
umap<- read.table("RNA-5c_log2CPM.csv", sep=",", header=T, row.names=1)
umap.labels <- sapply(strsplit(names(ctrl), '\\.'), function(x)x[2])
cell_line <- as.factor(umap.labels)
sc.umap <- umap(t(ctrl), random_state=111, n_epochs = 1000)
u=data.frame(x=as.numeric(sc.umap$layout[,1]),y=as.numeric(sc.umap$
                                                               layout[,2]))
#plotting UMAP, in this case if you want only to change the color of all the plot you can put color="color that you want"
sp <- ggplot(u, aes(x=x,y=y)) + geom_point(pch=19, cex=0.3, color='darkblue')
pdf("UMAP_RNA-5c.pdf")
print(sp)
dev.off()
```

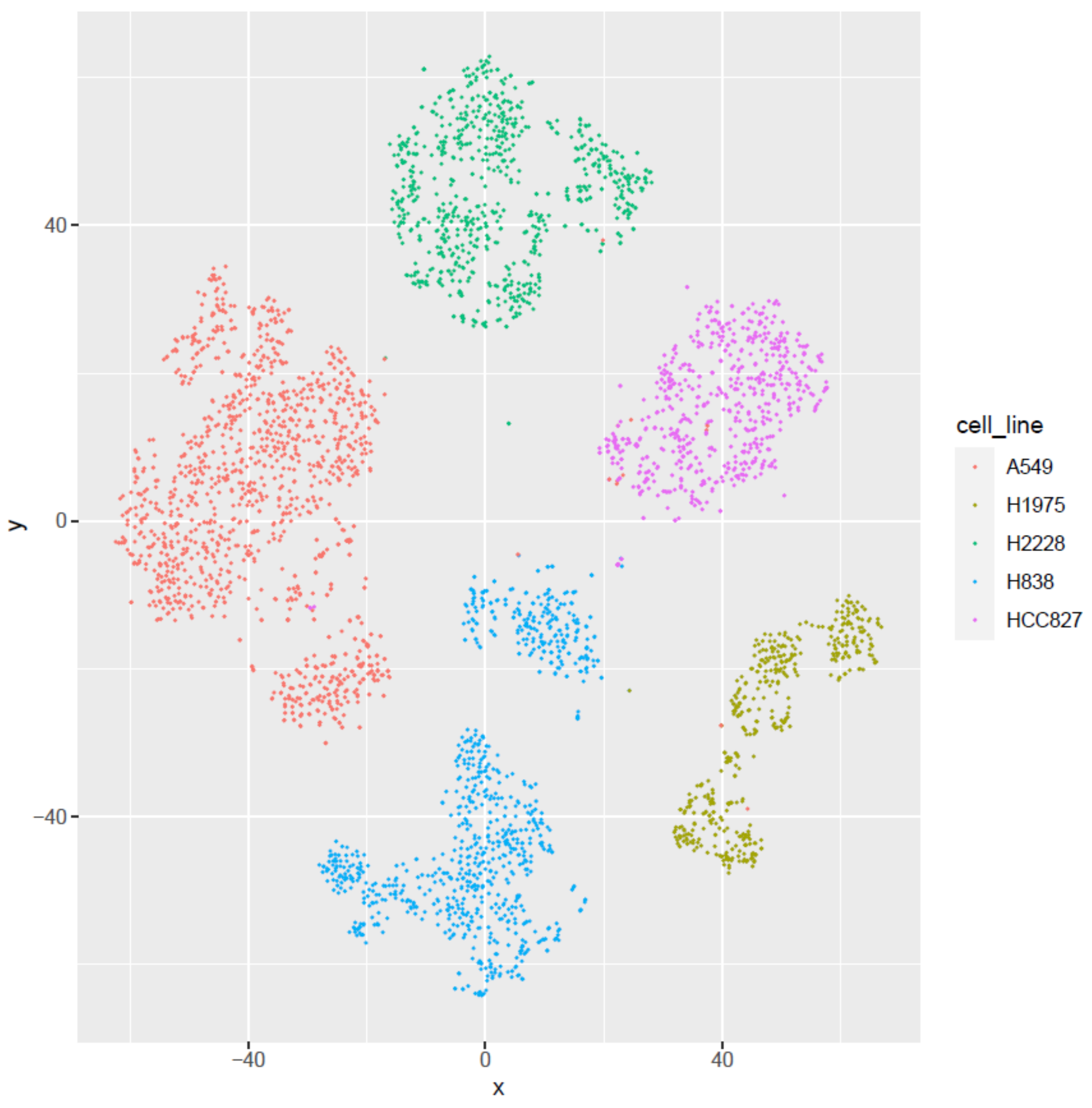


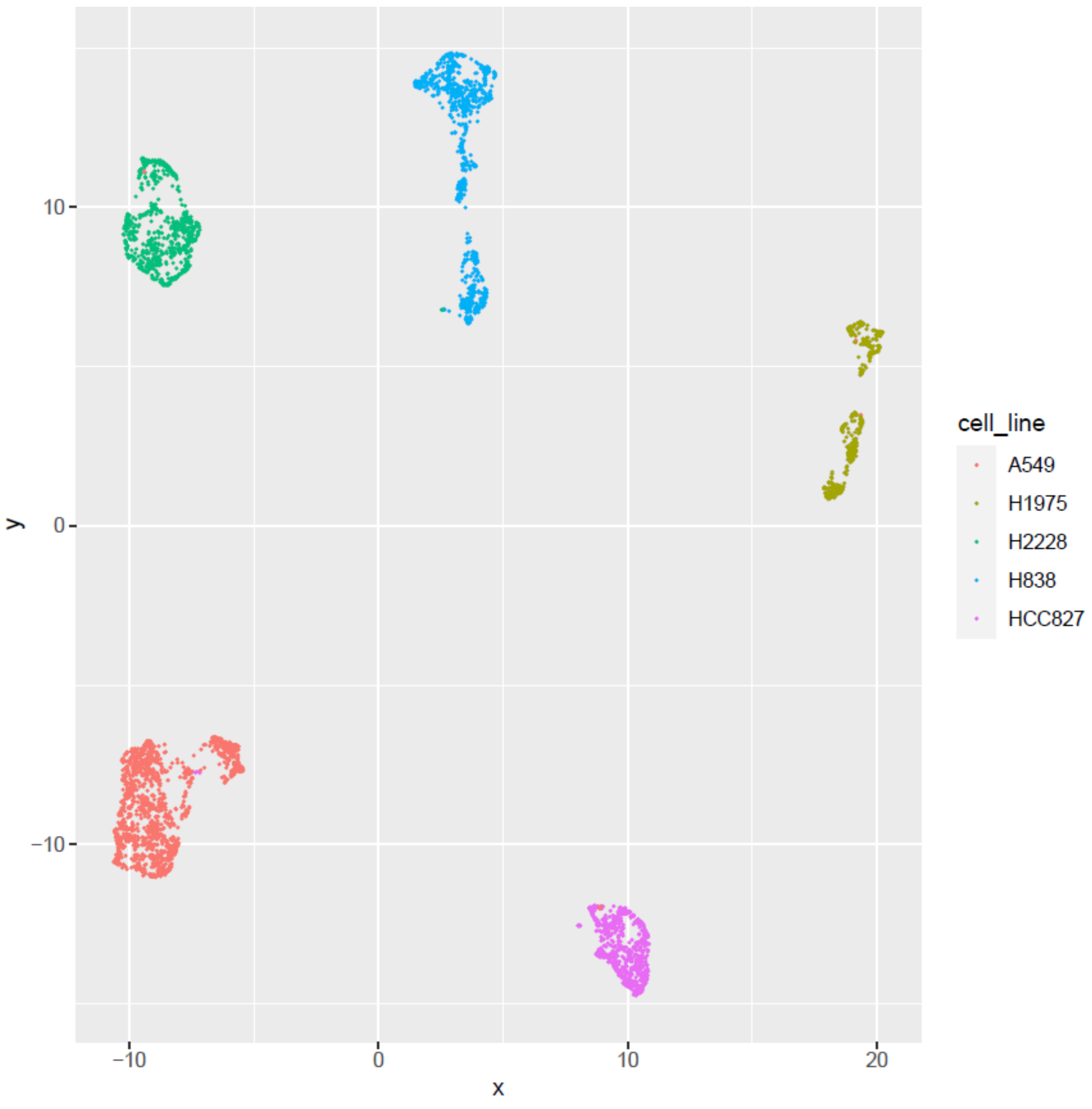
▼ **UMAP and tSNE with colours**

If you want to color the group of the results of tSNE and UMAP you have to colour the cells considering the group. To do that you have to add a column to the tSNE and the UMAP results with the cell line of each cells and then a factor column for the cell names.

```
#before to run tSNE generate the log2cmp file
counts2cpm <- function(file, sep = ","){
  tmp <- read.table(file, sep=sep, header=T, row.names=1)
  col.sum <- apply(tmp, 2, sum)
  tmp1 <- t(tmp)/col.sum
  tmp1 <- t(tmp1)
  tmp1 <- tmp1 * 1000000
  write.table(tmp1, "cpm.csv", sep=",",
              col.names=NA)
  write.table(log2(tmp1 + 1), "log2cpm.csv",
              sep=",", col.names=NA)
}

#at this time in not necessary we have to start from the file
saver_RNA <- read.table("saver_RNA-5c.csv", sep=",", header= TRUE, row.names = 1)

#run the function and change the name of the file
```

```
counts2cpm(file="saver_RNA-5c.csv", sep=",")
file.rename(from="log2cpm.csv", to="RNA-5c_log2CPM.csv")


#1.2 Running tSNE on the log2cpm file
#load library
library(Rtsne)
library(ggplot2)

#put the log2cpm data in a variable
log2cpm_RNA <- read.table("RNA-5c_log2CPM.csv", sep=",", header= TRUE, row.names = 1)

#We do not want any PCA step before doing tSNE -> set pca parameter pca =FALSE
tsne_out <- Rtsne(as.matrix(t(log2cpm_RNA)), pca=FALSE, perplexity=30,theta=0.0)
t=data.frame(x = as.numeric(tsne_out$Y[,1]),y = tsne_out$Y[,2])

#You need to apply the tSNE and UMAP but you need to differentiate the cell lines in the table, which are contained in the column n
cell_names <- strsplit(colnames(log2cpm_RNA), "\\.")

#Adding a column with the cell lines to the t dataframe and factor it
matrix_cellnames <- matrix(unlist(cell_names), ncol =2, byrow=TRUE)
t$cell_line=matrix_cellnames[,2]
t$cell_line <- factor(t$cell_line)

#adding the rownames to t
rownames(t) <- colnames(log2cpm_RNA)

# plotting tSNE
sp <- ggplot(t, aes(x=x,y=y,color=cell_line)) + geom_point(pch=19, cex=0.3)
pdf("tSNE_RNA-5c.pdf")
print(sp)
dev.off()

#1.3 Run UMAP
#load library
library(umap)
library(ggplot2)

#run UMAP
umap <- log2cpm_RNA
umap.labels <- matrix_cellnames[,2]
cell_line <- as.factor(umap.labels)
umap_5c <- umap(t(umap), random_state=111, n_epochs = 1000)
u=data.frame(x=as.numeric(umap_5c$layout[,1]),y=as.numeric(umap_5c$layout[,2]), cell_line=cell_line)

#adding the rownames to u (ocnatning the UMAP results)
rownames(u) <- colnames(log2cpm_RNA)
head(u)

#plotting UMAP
sp <- ggplot(u, aes(x=x,y=y, color=cell_line)) + geom_point(pch=19, cex=0.3)
pdf("UMAP_RNA-5c.pdf")
print(sp)
dev.off()
```


40
0
−40
y
−40
0
40
x
cell_line
A549
H1975
H2228
H838
HCC827



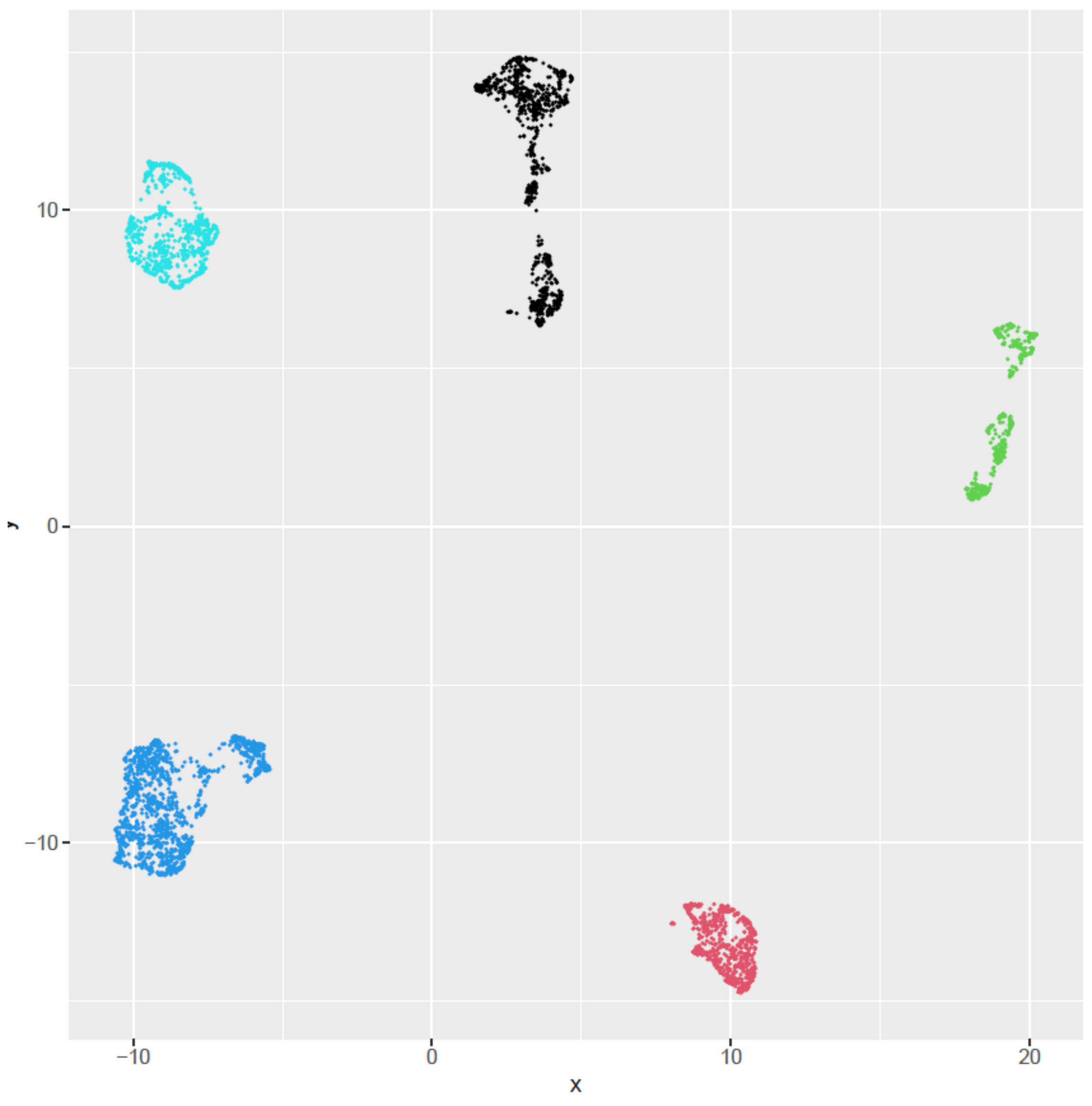
From that you can apply two different approaches:

1. Apply Kmeans clustering using mbkmeans package

First you have to install on R studio the mbkmeans packages with the function below.

```
#You have to start from the umap output. Since the umap output (u dataframe) has been already modify to put the color the plot you have to
#you can se the structure of the umap output with str(umap_5c).

str(umap_5c) #this str function give you the idea how the UMAP output data hare organized

#To do kmeans we have to use kmeans function on the UMAP result that are in the umap_5c variable only taking the layout component of the co
u_kmeans<-kmeans(umap_5c$layout , 5)


#Then you have plot the results that are contaneid in the kmeans results on the cluster component of the complex kmean result object of cla
#To plot and color the important thing is to have the color as.numeric otherwise the ggplot function doesn't work.
plot<- ggplot(u, aes(x=x,y=y)) + geom_point(pch=19, cex=0.3, color=as.numeric(u_kmeans$cluster))

pdf("u_kmeans.pdf")
print(plot)
dev.off()
```

Apply K-means on the UMAP output data that contain the x-y coordinates. Remember that the UMAP results gives a data.frame with the rows that are the cell (rownames), while the column that are the value from the dimensionality reduction (x-y coordinates from). From UMAP, we'll obtain a result in which cells on the row, x-y components of UMAP in the columns (the information you want to use). If the results are not organized in this way you have to make a table organized in this way to be used by `ggplot` function. Then, you have to generate a file that contain the colours for each cluster (the colours has to be assocaided to the cell types). To do that you have to add a column contaning the cell names as factor variable. At the end you can then make 2 graphs in which one is coloured by cell line and the other is coloured by cluster.



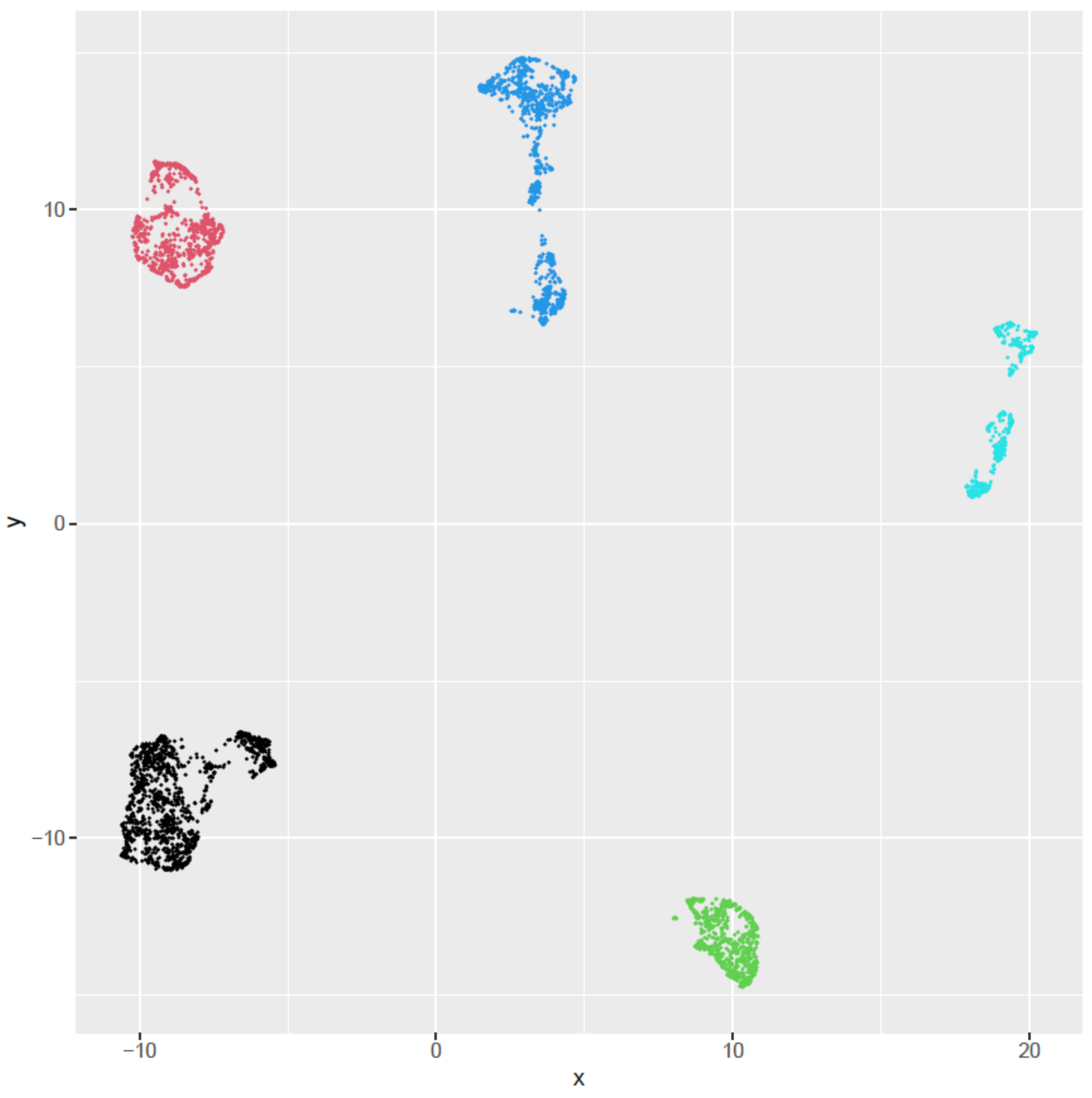
2. Apply Kmeans clustering using ClusterR package

ther type of K-mean tool working on the initial data → you take the initial table you use for UMAP and then overlay the clustering results with this approach to the UMAP results.

This is done to see if this clustering fits the dimensionality results fits the dimensional reduction.

```
#First you have to install the ClusterR packages
install.packages("ClusterR")
```

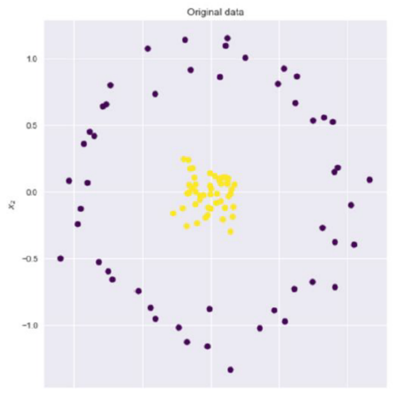
```
#then you have to load the libraries
library(ClusterR)
library(ggplot2)

#use the Kmeans_arma function to do the clustering
km <- KMeans_arma(umap_5c$layout, clusters = 5, n_iter = 10, seed_mode = "random_subset", verbose = T, CENTROIDS = NULL)

pr <- predict_KMeans(umap_5c$layout , km)

plotR<- ggplot(u, aes(x=x,y=y)) + geom_point(pch=19, cex=0.3, color=as.numeric(pr))

pdf("ClusterR_kmeans.pdf")
print(plotR)
dev.off()

#I am not sure of this part see correction
```

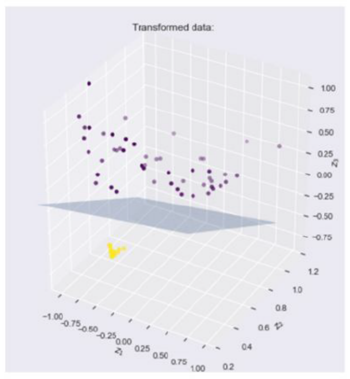


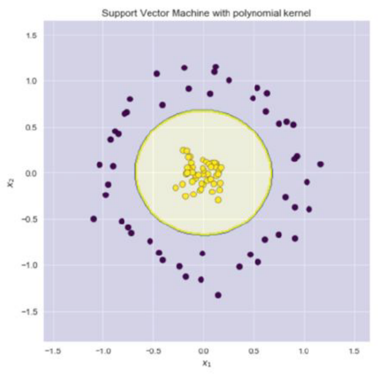
SIMLR

▼ Transcript of last year

SIMLR
The peculiarity of this method is the way we do the data reduction. It's basically Kmean but with a different kind of data reduction approach. It's actually using kernels that are a subspace representation of our data. The SIMLR select the representation (kernel) that better fits the discrimination that you are expecting to your data which if defined by the Rank Contrain wich depends on the number of clusters that you input. Then you create a similarity matrix using the kernels selected. Then this matrix is used to observe it and get some graphics out of it.
A kernel is an alternative projection. For example this data it's almost impossible to separate but if we change the perspective of the plot we can separate the data using this hyperplane. Actually a kernel a value that is given by the parameter that has given to your cells represented in this way.
So you can move the projection from a 2 dimensional to a 3 dimensional perspective in which you can find out where the separation is. Different kernels are different 3 dimensional projections, we will look for the projection that let us separate the data the best.
One of the limitations of this method is that it doesn't scale well when the number of cells increases. The limitation is around 2000 cells. Another limitation i that the number of genes must always be greater than the number of cells.







Not always the simple k-means is enough: data reduction is not always sufficient for ??.

It is a kernel based approach: this dataset is very difficult to cluster → yellow cells in the middle with all the blue ones all around.

Kernel = transformation of initial data to other representation i.e. a projection on a different space (ex: what we did with the t-distribution in t-SNE.

By changing the representation of the data and by finding the best representation possible, you can find some way of separating your data.

SIMLR does various projection of the data and various clustering: you select only the kernels that better fit the cluster number provided by the user. You make a similarity matrix that use all the kernel information and only after you do dimensionality reduction.

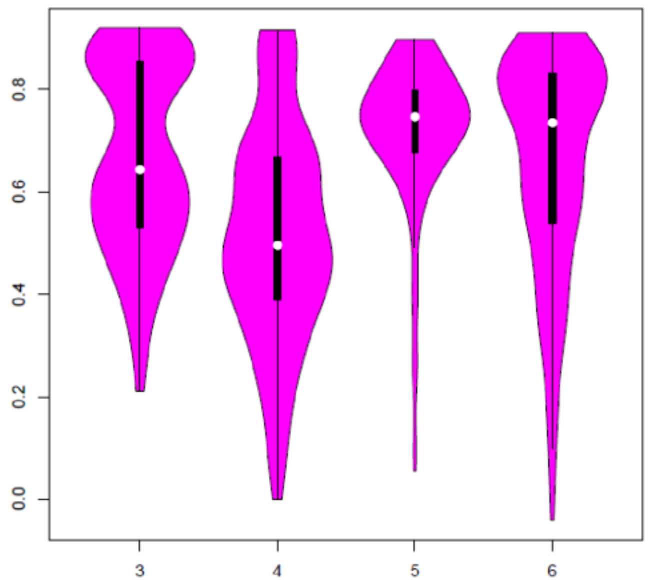
We are sort of subsetting our data to make a better representation.

Which is the best number of clusters?

There are many many different ways to find it. There is no public consensus on which is the best one.

▼ Last year transcript

Silhouette plot
We have the setA.vioplot.pdf that is referring to a matrix that measure the quality of the cluster that is called silhouette plot that measure how similar are the cell that belong to the cluster and this measure is better when the cell in a cluster are very far to another cluster (the optimal situation is one e the worst is zero).



With three clusters the average of the silhoutette plot is 0.7, it drops down with four clusters, gets better with five clusters and with six clusters is not good as the situation with five clusters (the best situation).

One of these is called Silhouette coefficient.

Silhouette value is a score from -1 to 1 on the base of 2 parameters: cohesion among the cells belonging to the same cluster and separation from the cell in a cluster to the cells of other clusters.

Ideal parameters: high cohesion and high separation.

Cohesion = relative measure with respect to the points of the cluster

Separation = distance to the points of one cluster to the other. Select the minimum value to calculate the silhouette score. If with this the SS is good, it means that there is a good separation.

Take the cluster, calculate the center of the cluster, calculate the distance between the center of the cluster to the oder cell and then select the set of distances that have the minimal distance between the center of the cluster and the oder cell.

You then take the separation info - cohesion info divided by max value of the 2 parameters → this is the silhouette score. -1 = very bad clustering to 1 = very good clustering from the separation and cohesion POV.

❗ This silhouette method works well only if you have a good separation of the clusters.

Single cell experiments are very noisy → we can take out a random subset of cells (10%): if the cells are not so important the cluster will be the same. Maybe there are some unstable clusters that are perturbed by this perturbation of the dataset.

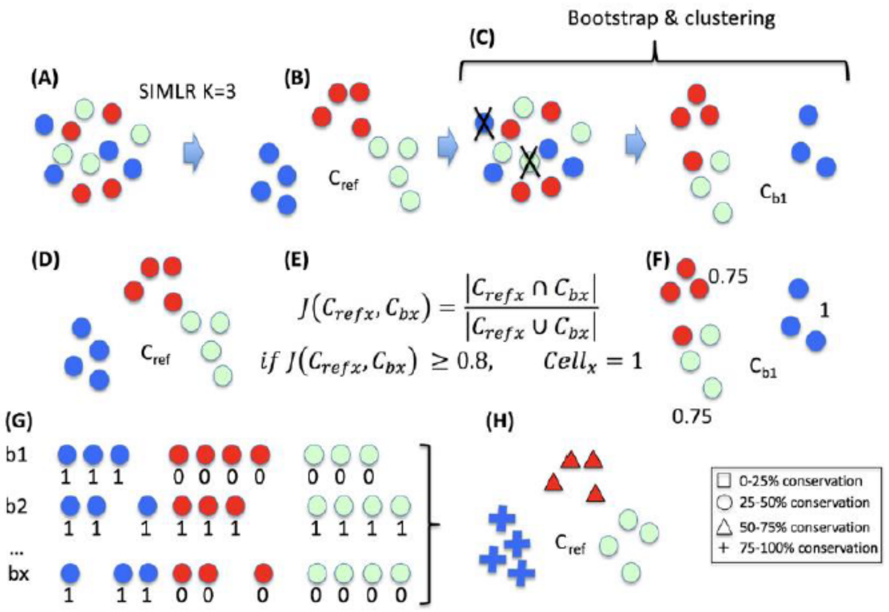
The process is repeated multiple times and we calculate how many times the cells remain in the same cluster.

▼ Last year transcript

Reproducibility of the cluster to the idea that starts to look if the cells are always being part of the same cluster.

Start from a set of cell, do clustering with K=3, obtain 3 cluster that became the reference cluster (Cref), then I take the initial data set and take out randomly a subset of cells that could be 10% or 5% or 15%, do the clustering again (first permutation cluster). Then we use the Jaccard index to compare the composition of the cell of the first permutation cluster to the reference cluster. If the cells are always in the same cluster the score is 1, instead if the cells are not getting over the threshold that was set to identify belonging to that cell to a specific cluster the cell does not get the score. That means that in the same cases if the cluster is not robust enough to contain the same cell each cell of the cluster is not scored. Then repeat multiple times this kind of permutation and at the end we can have the representation of the clusters more cell oriented. A stability of more than 75% is collectible if you have at least 90/93% of the cell of the same type (e.g. setA). The stability of the cluster starts to reduce if the

cluster is not anymore made by a unique type of cell (e.g. setC). So, the cell stability score (CSS) is inversely correlated with the cells' heterogeneity of the cluster, better is the CSS much more pure is the cluster. If we do not get a good CSS we can change the clustering tools or the number of clusters. The cells that are at the boundaries of two clusters are characterized by cells that have a lower CSS (transition cell).
If you increase the perturbation (different percentage of the cells that are removed) the same cell is able to move from one cluster to another. When you take out 50% of the cell you are generating a new data set and the overall stability drops in general.



Louvain modularity-based methods: Griph and Seurat

▼ Last year transcript

GRIPH AND SEURAT CLUSTERING (estimating the number of clusters)
It's an approach that estimates the number of clusters so we don't have to input them. One of the advantages of this approach is how quick it is. Sometimes since it's so quick we may use it just to get a rough idea of what could be the optimal partition (the optimal number of clusters) then knowing that we run a different clustering method like SIMLR.
Seurat and Griph share the same clustering approach which is called Louvain Modularity which agrates cells looking for similar communities. The main difference between the two methods is that Seurat doesn't use the full data set for the analysis but it's extrapolating. Also Seurat will require a PCA value of the data while Griph uses a tSne visualization as the data reduction approach.

Louvain Modularity
In this case our clusters are the identification of communities of cells that like to stay together.
Communities are groups of nodes within a network that are more densely connected to one another than to other nodes, while Modularity is a metric that quantifies the quality of an assignment of nodes to communities by evaluating how much more densely connected the nodes within a community are compared to how connected they would be, on average, in a suitably defined random network.
Before **going any further we need to learn some now concepts:**

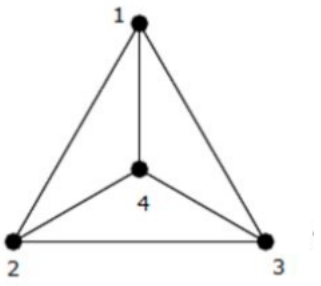
- **Heuristics**
  A heuristic is any approach to problem solving or self-discovery that employs a practical method that is not guaranteed to be optimal, perfect or rational, but which is nevertheless sufficient for reaching an immediate, short-term goal.
  Where finding an optimal solution is impossible or impractical, heuristic methods can be used to speed up the process of finding a satisfactory solution.
- **Greedy algorithm**
  A greedy algorithm is any algorithm that follows the problem-solving heuristic of making the locally optimal choice at each stage with the intent of finding a global optimum.
  In many problems, a greedy strategy does not usually produce an optimal solution, but nonetheless a greedy heuristic may yield locally optimal solutions that approximate a globally optimal solution in a reasonable amount of time.
- **Coarse-Grained modeling**
  Is nothing else than a simplified representation of a network

Louvain Modularity applies main steps in order to maximise the modularity:

1. The first step is a "greedy" assignment of nodes to communities, favoring local optimizations of modularity. Basically it defines potential group of nodes to stay together so they may be part of the same community
2. The second step is the definition of a new coarse-grained network in terms of the communities found in the first step.

These two steps are repeated, changing the nodes from one community to another and calculating different modularities, until no further modularity-increasing reassignments of communities are possible.

- Cliques



a Clique is a group of points in which every point is fully connected with the rest. In this graph we can calculate how many edges connect the points, the amount of connections: k(k-1)/2 (where K is the number of points). This number (the number of connections) is actually the modularity of this network because it indicates how many connections exist between this network.

Griph and Seurat → they decide by themselves how many clusters have to be generated.

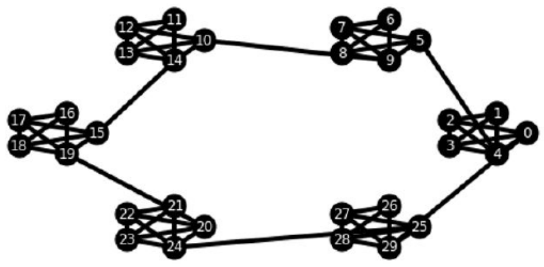
Level of connectivity among nodes → the more connected, the easiest to define that they belong to the same group of cells. You are comparing connectivity you can have in a random set of nodes to the one you have.

Heuristics → these approaches are used when you do not have an optimal solution because the numerical solution is impossible to have. They provide the best solution possible for you without guarantee that is the actual best (maybe only local minimum very far from the absolute minimum).

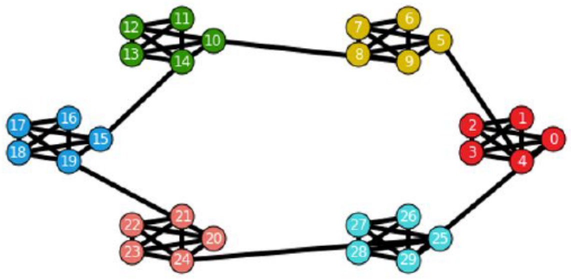
Greedy algorithms → any algorithm that follows the problem-solving heuristic of making the locally optimal choice at each stage.

Coarse grained → simplified representation of the data.

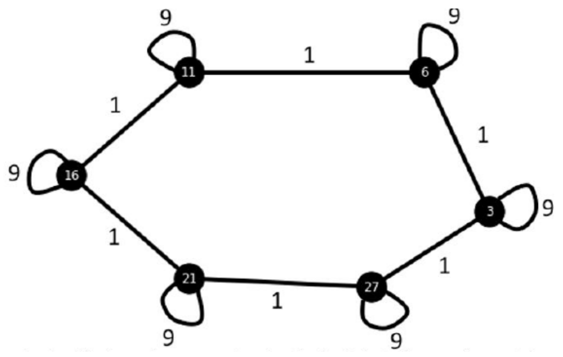
Example



On the first step we will aggregate randomly the 30 nodes in communities expecting to have the highest number of connections between them, aka modularity (we don't know the connection at first).



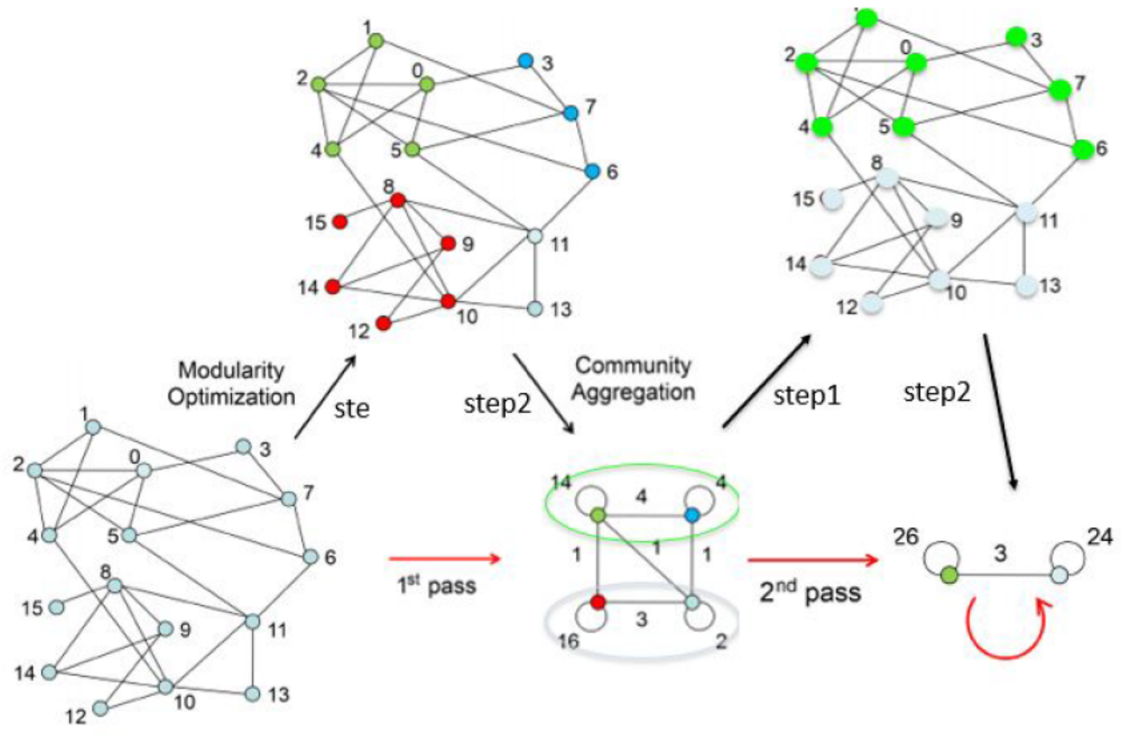
Then we do the coarse-grained modelling and in which each community is converted to a node with an associated value that is the number of connections between the nodes of the community.



In this graph, the black numbers associated with the links indicate edge weights, but the white numbers on the nodes are just labels for the communities. In this case we already have found the highest modularity but normally this process will have to be repeated many times until we find the best one.

It tries to move the points until it finds the best association between the points.

More complex example



Missing part in the transcript → riascoltare!

Advantage → communities are defined by themselves and so there is no problem of the parameters.