# 6 - Dimensional reduction: PCA and tSNE

| | |
|---|---|
| TOPICS | PCA tSNE |
| DATE | @April 20, 2022 2:00 PM |
| LAST EDITED BY | |
| LAST EDITED TIME | @May 7, 2022 4:07 PM |
| MADE BY | |
| PROF | Calogero |
| Recording | |
| STATUS | ☑ |

💡 The installation of devtools of last lesson has to be done on the docker. If the docker contain already R, the installation of devtools should be not so difficult. We will use only few function of the devtools packages (e.g. pca function).

### Summary of what we have done up to now

We went in parallel for both bulkRNAseq and scRNAseq sequencing first looking at the quality control of the data for the bulkRNAseq and for scRNAseq. Normally, for bulkRNAseq you use a fatsQ file to evaluate the quality of the sequencing and the multiQC to get an overview of what is the result (quality of the mapping+ quality of fastq results). We did more or less similar things for scRNAseq in which you can use the fastQC to see the quality of the sequencing. Remember that in the single-cell there are two reads that contained mapping information and the quality is lower. good mapping reads should be characterized to drop in exon coding sequence

To answer the biological question we will start to evaluate with PCA the bulkRNAseq data and with other non-linear dimensional reduction methods for the single-cellRNAseq data. The situation is slightly different among single-cellRNAseq and bulkRNAseq. In BulkRNAseq PCA is mainly used to have an overall view that the data that you have generated are different or similar in the way that you know how to treat them to define the differential expression. For the single-cell, in general dimensional reduction is already part of the characterization of the subpopulations that exist in your single-cell experiment. The two experiments are focused into quite different concepts and answer different questions: BulkRNAseq tried to extract the changes between one condition to another while scRNAseq tried to show what is actually changing within the cell populations present in your sample (if the sample is a complex one).

Our dataset contains bulkRNAseq data coming from PC9 cells that is a lung cancer cell line that is characterized by a positive mutation in EGFR (this means that are sensitive to Osimertinib that is the first line treatment for this kind of tumor). We have the bulkRNAseq data for PC9 cells in three different conditions: untreated (only DMSO), acute treatment (24h treatment) and Chronic treatment (treatment for 21 days). With this kind of experiment you are actually measuring the difference in expression on the RNA that left in the cell that are still growing after the treatment or without the treatment. For single-cell the experiment is different: the

same untreated PC9 cells are injected in mice (to mimic an environment that is similar to the normal tumor growth) and then these mice are divided into treated and control. The treatment is done for 24 hours.

With this experiment you are actually going to see a sort of the first response of the cells upon the treatment. The things you can find are different between bulk and single-cell:

The first question for bulk is: *Are the acute and chronic actually changing the same genes or different genes?.* On the other side, for single-cell the question is: *after treatment, are we going to see the same number of a subpopulation that are related to the cell or the number of cells belonging to this or that subpopulation are changing in some way?* Another question could be: *what happens if I take the genes that are characterizing the acute treatment in the bulkRNAseq and I use only that subset of genes to make the clustering at the level of the single-cellRNAseq?*

Since we have both the bulkRNAseq and scRNAseq of the same experiment you might start to do some investigation on this kind of data. The overall question is always to try to understand what happens and what is the mechanism in terms of RNA expression that induce cells to develop resistance.

Remember: mutated tumors after a certain time develop some level of resistance. The point is trying to understand which are the different ways this resistance can be generated.

so the same cell line used to perform two different investigation in bulk and single cell

# BulkRNAseq and PCA

Principal component analysis (PCA) is performed on bulkRNAseq data and is used to have an overall view of the data generated, if  the expression profile is similar or different. PCA provides also a way to represent the data in a low dimensional space. In scRNAseq experiments PCA can be done (usually more complex dimensional reduction tools are requred, see later) to characterize the different subpopulations present in the sample, so in this case data reduction is already a way to characterize the data that you want to analyze.

> ❗ **YouTube video on PCA:**
>
> https://www.youtube.com/watch?v=FgakZw6K1QQ

These numbers are weights for the importance of each gene to PC1.
In PCA terminology, the weights are called "loadings" and an array of "loadings" for a PC is called an "eigenvector"

| Gene | Influence on PC1 | In numbers |
|---|---|---|
| a | high | 10 |
| b | low | 0.5 |
| c | medium | 3 |
| d | low | -0.2 |
| e | high | 13 |
| f | high | -14 |
| ... | ... | |

we have seen the term weight as defining the importance of the gene to determine a variance of the overall experiment. how we can find these weights?

We will see **Principal Component Analysis (PCA)** one step at time using singular value decomposition (SVD).

| | Mouse 1 | Mouse 2 | Mouse 3 | Mouse 4 | Mouse 5 | Mouse 6 |
|---|---|---|---|---|---|---|
| Gene 1 | 10 | 11 | 8 | 3 | 2 | 1 |
| Gene 2 | 6 | 4 | 5 | 3 | 2.8 | 1 |

We will start from an example to be more clear. In the images there are the values of the transcription of two genes (Gene1 and Gene2) in six different mice. The mice in this case have to be consider individual samples and the genes have to be consider variables for each sample.

this is different ffrom the last elcture example in which we haev more genes but only two samples

PCA analysis provides a tool to do data reduction, from a high dimensional dataset to a 2D plot. From the example done last time, we got a sort of weights that consider the expression of the gene but we didn't really define them. Now we will understand better this concept.

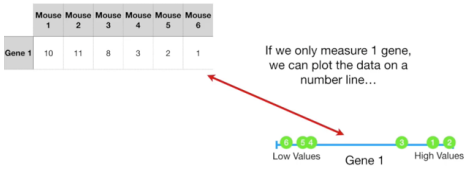
Let's start from the concept of dimensional reduction in general:

- If we only measured 1 gene we could plot the data on a numbered line in which mice 1, mice 2 and mice 3 have relatively high values and mice 4, mice 5 and mice 6 have relatively low values. Even though it's a simple graph it shows us that mice 1, mice 2 and mice 3 are more similar to each other than mice 4, mice 5 and mice 6.
- If we measured 2 genes we could plot the data on a two-dimensional XY graph in which Gene 1 is on the X-axis and spans one of the two dimensions, while Gene 2 is on the Y-axis and spans the other dimension. In this case we can see that mice 1, mice2 and mice 3 cluster on the right side and mice 4, mice5 and mice 6 cluster on the lower left-hand side.

we calculate the mean value for all the point that are represent on each axis

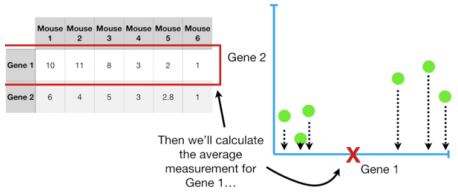
- If we measured 3 genes we would have to add another access to the graph and make it look 3D. In 3-dimensional the smaller dots have larger values for Gene 3 and further away the larger dots have smaller values for Gene 3 and are closer.
- If we measured 4 genes we could no longer plot the data with 4 genes because require 4 dimensions, which cannot be represented on a screen.

PCA can take 4 or more genes, thus four or more dimensions of data and make a two-dimensional plot showing only the principal components, hence the most variable variables (sorry for the play on words) of our sample. PCA plot will show us that similar mice/samples/cells cluster together. PCA can tell us which gene or variable is the most valuable for clustering the data. For example, PCA might tell us that gene 3 is responsible for separating samples along the X-axis. PCA can tell us how accurate the 2D graph is.

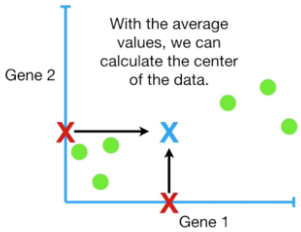


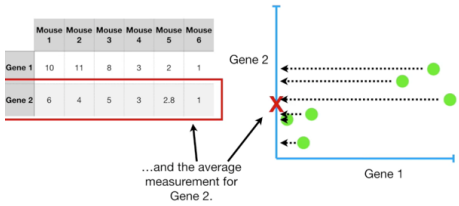

To understand what PCA does and how it works we consider a data set with two genes only. First, we'll start by plotting the data on the X-axis or Y-axis and then we'll calculate the average measurement for gene 1 (X-axis) and the average measurement for gene 2 (Y-axis). With the average values, we can calculate the center of the data.

For gene 1 for example, already from the counts we can appreciate the difference in expression. We can take the expression level of this gene and plot it on the X-axis. X-axis is going to be the axis for gene 1. We can then calculate the mean value, and this value will be the central point on the X-axis. The same is done with gene 2 on the Y-axis, with the mean value set as the central point on the Y-axis.




| | Mouse 1 | Mouse 2 | Mouse 3 | Mouse 4 | Mouse 5 | Mouse 6 |
|---|---|---|---|---|---|---|
| Gene 1 | 10 | 11 | 8 | 3 | 2 | 1 |
| Gene 2 | 6 | 4 | 5 | 3 | 2.8 | 1 |



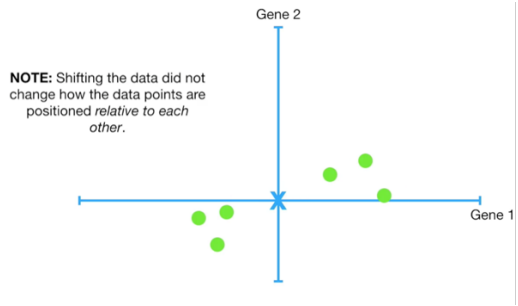




📌 From this point, we will focus on what happens in the graph and we no longer need the original data.

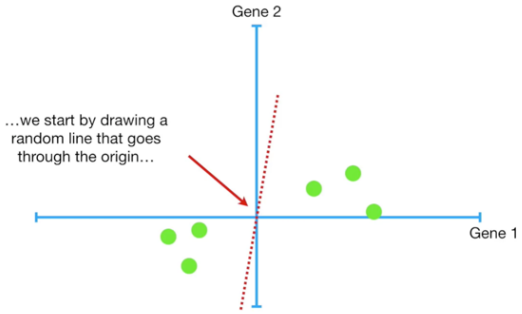
With this procedure we are changing the reference axis: the mean will become the center of the axis.

we are no more considering gene 1 and gene 2 , we are simply considering how the various muice are changing their expression with respect to the average expression. To do this the mean value become the center fo the coordinates

We have shifted the data so that the centre is the origin in the graph. **We have obtained a representation of our data in reference to the mean value.** that become the zero point




> ! Shifting the data did not change the relative position of the data points to each other.

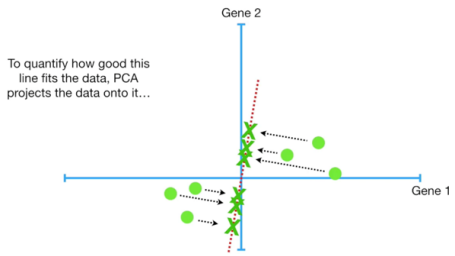



Now that the data are centred on the origin, we try to fit a line to it. To do this we start by drawing a random line that passes through the origin, then we rotate the line until it fits the data as good as it can (it still has to pass through the origini).

this drawing tries to define a projcetion of all the data plotted on the diagram

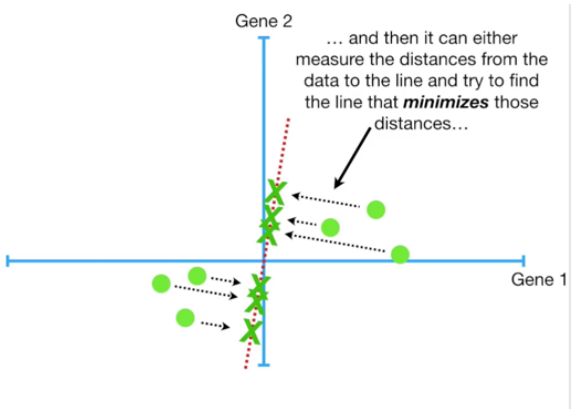
*How does PCA decide if the fitting is good or not?*

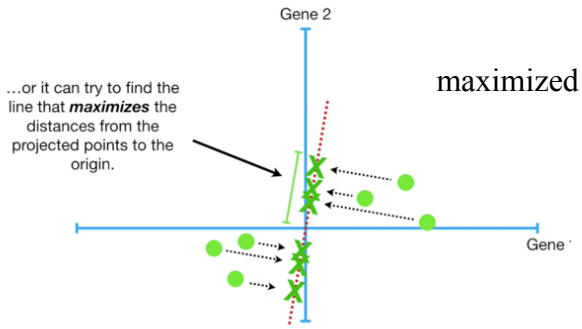
Starting from the original random line passing through the origin, we need a method to quantify how good this line fits the data.




Why we doo this? the first we need to define is the first component representing the greatest amoint of variance from your data so we have 2 possibilities

To do so, PCA projects the data onto the random line and then it measures the distances from the data to the line and either **tries to find the line that minimizes the distances from the line to the projected points or** it can try to find the line that **maximizes** the distances **from the projected points to the origin.**

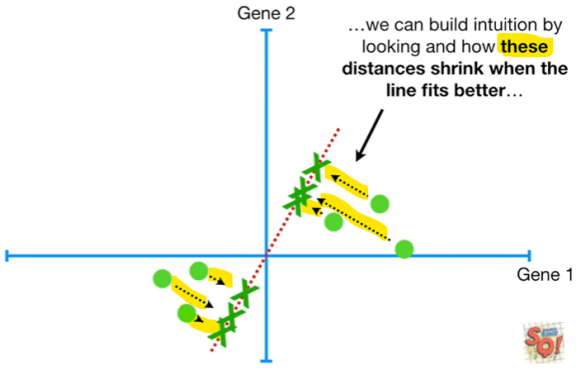


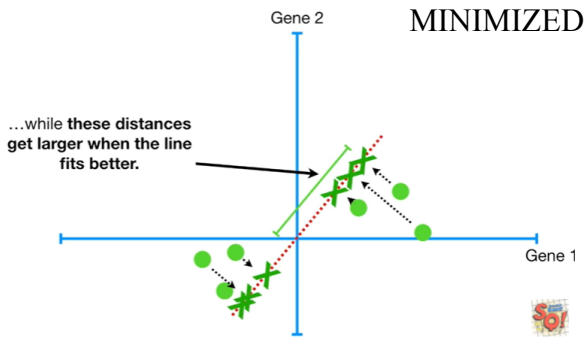




maximized

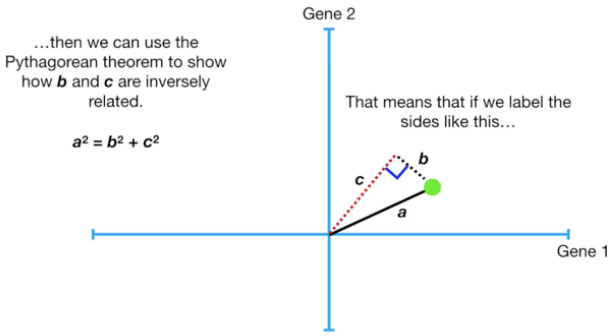
intuitively one of the possibility is trying to rotate the line to match one of the two possibilites described before

Mind that these options are equivalent: intuitively we can demonstrate this by looking that when the line fits better the first distance from the line to the points is minimized while the second from the points to the origin is maximized.



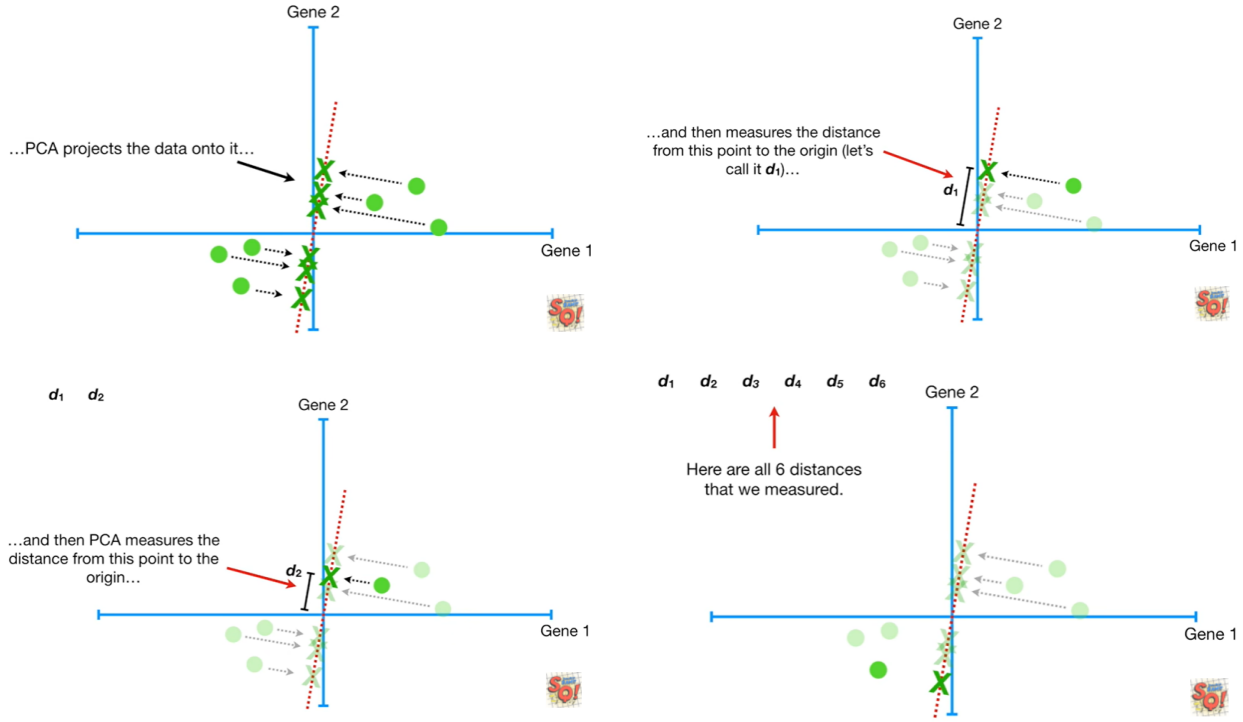

Now to understand what is going on in a mathematical way let's consider just one data point.



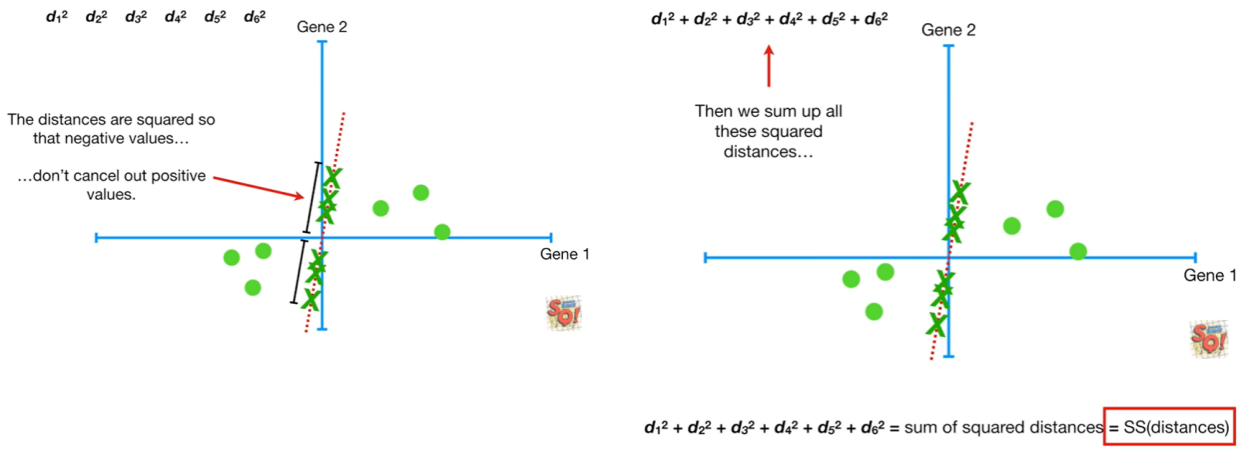

This point is fixed and so is its distance from the point to the origin doesn't change when the red dotted line (random line used to fit) rotates. When we project the point onto the line we get a right (90°) angle between the black dotted line and the red dotted line. If the black dotted line is C, we can use the Pythagorean theorem to show how B and C are inversely related. If B gets bigger then C must get smaller while if C gets bigger B must get smaller thus PCA can either minimise the distance to the line or maximise the distance from the projected point to the origin.

Intuitively, it makes sense to minimize the distance from the point to the line but it's actually easier to calculate the distance from the projected point to the origin. So, **PCA finds the best fitting line by maximising the sum of the squared distances from the projected points to the origin.**

Basically, first PCA draws the random line, then it projects the data onto it and then measures the distance from this point to the origin (d). PCA measures the distance from this point to the origin (d) for all the six dots and obtains six distances (d1 - d6).

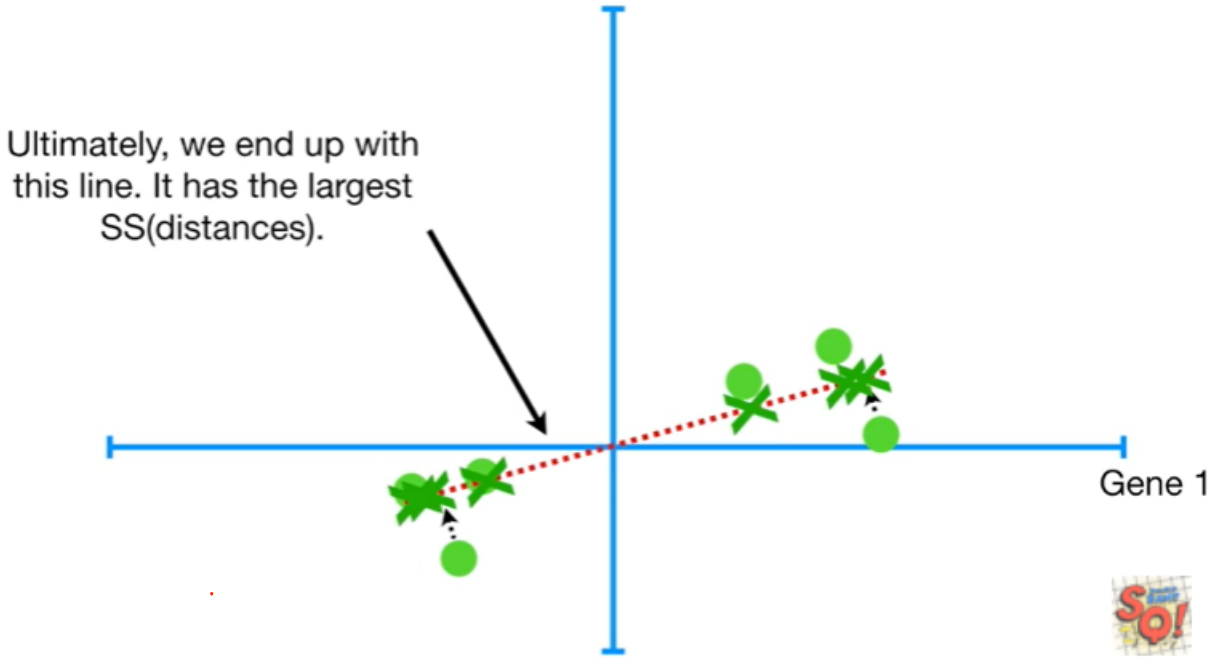


Then, it does the square of all the distances (d1, d2, d3, d4, d5, d6) and in this way are squared so that negative values don't cancel out positive values. Then, PCA sums up all these square distances that equal the **sum of the squared distances** (SS distances or sum of squared distances).




Now we rotate the line, project the data onto the new line and then sum up the squared distances from the projected points to the origin.

We repeat until we end up with the line with the largest sum of squared distances between the projected points and the origin.

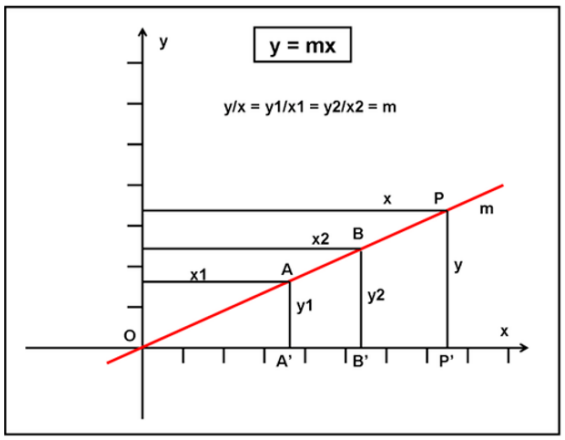


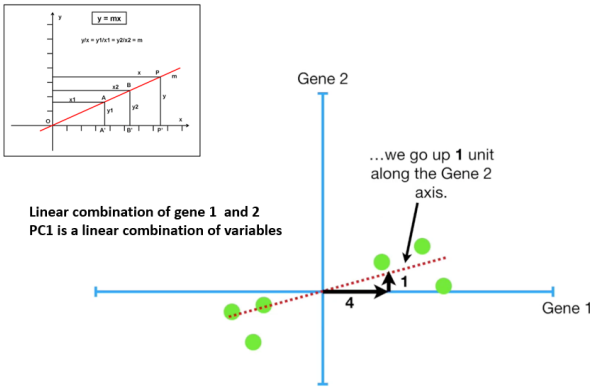

and represent the biggest variance

In our example, we end up with the line that has the largest sum of squared distances. This line is called **principal component 1 or PC1.** PC1 has a slope of 0.25, in other words for every four units that we go out

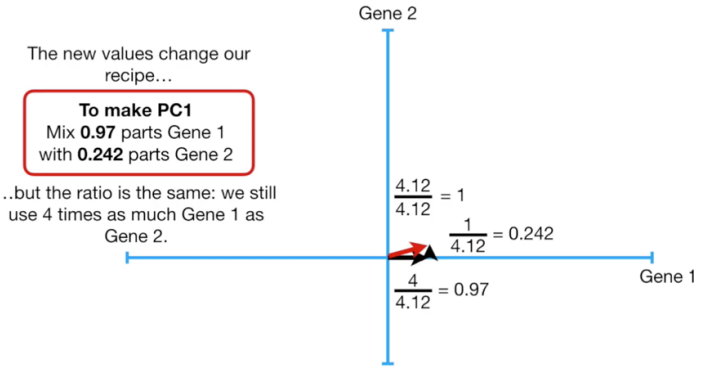
we will get one for each sample but in the majority of the cases
the first 2 or 3 PC are sufficent to describe the variance

along the gene 1 we go up one unit along the gene 2.  This means that the data are mostly spread out along the gene 1 axis and only a little bit spread out along the gene 2 axis.

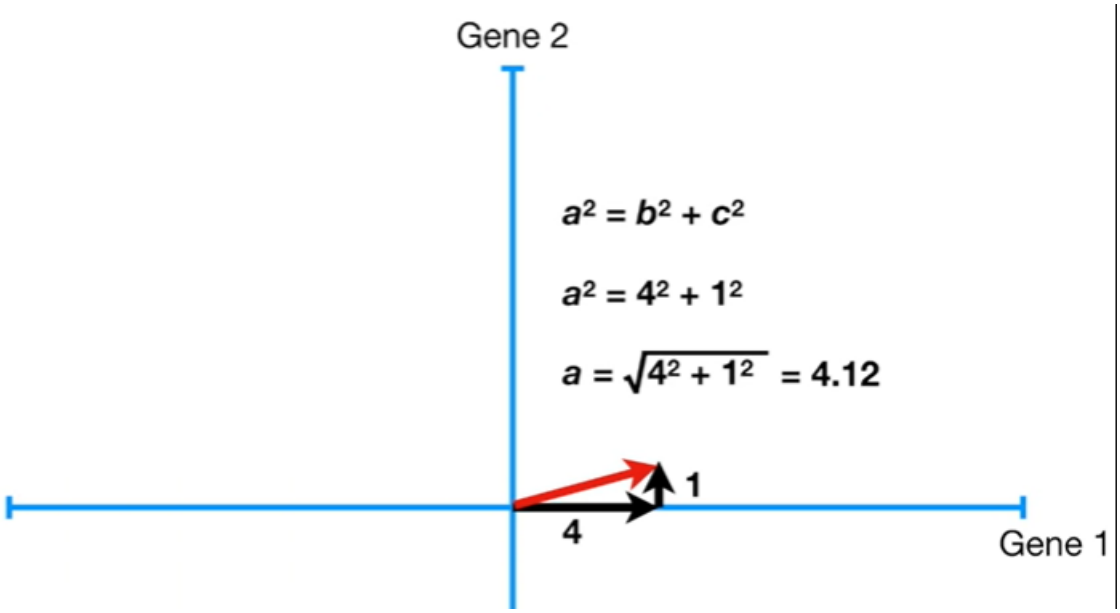


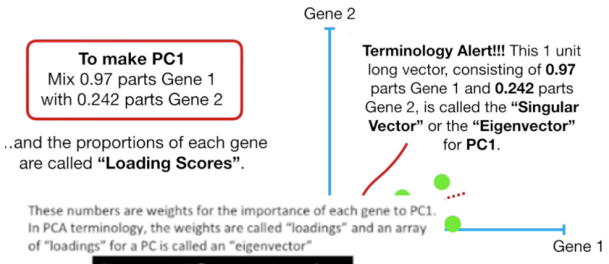

One way to think about PC1 is in terms of a cocktail recipe to make PC1, you have to mix 4 parts of gene 1 with one part gene 2. The ratio of gene 1 over gene 2 tells you that gene 1 is more important when it comes to describing how the data are spread out.




Mathematicians call this **a linear combination** of genes 1 and gene 2. We can solve for the length of the red line using the Pythagorean theorem ($a^2 = b^2 + c^2$). The result is a = 4.12. So, the length of the red line is 4.12.




When you do PCA with SVD the recipe for PC1 is scaled so that this length equals 1. So, we have to perform a sort of **normalization** in which the redline is equals 1 and to do this we need to divide each side by 4.12. The scaled values (new normalized values) change our recipe but the ratio is the same (we still use four times gene 1 and one time gene 2).




| Gene | Influence on PC1 | In numbers |
|---|---|---|
| a | high | 10 |
| b | low | 0.5 |
| c | medium | 3 |
| d | low | -0.2 |
| e | high | 13 |
| f | high | -14 |
| ... | ... | |

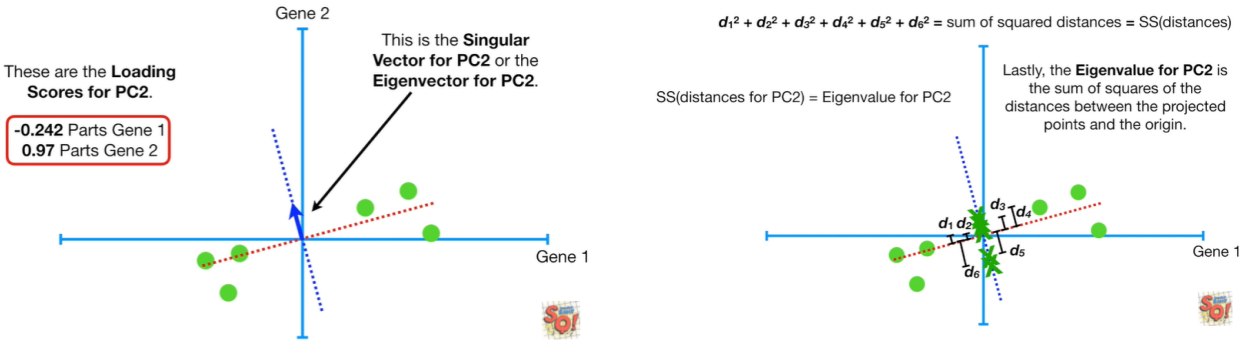
This unit vector we just calculated consisting of 0.97 parts gene 1 and 0.242 parts gene 2 is called the **singular vector or the eigenvector for PC1** and the proportions of each gene are called **loading scores**. PCA calls the sums of squares of the distances for the best fit line **the eigenvalue for PC1**. The square root of the eigenvalue for PC1 is called **the singular value for PC1.**

PCA is a linear data reduction approach, you are evaluaring the variation referring to aline a linear representation

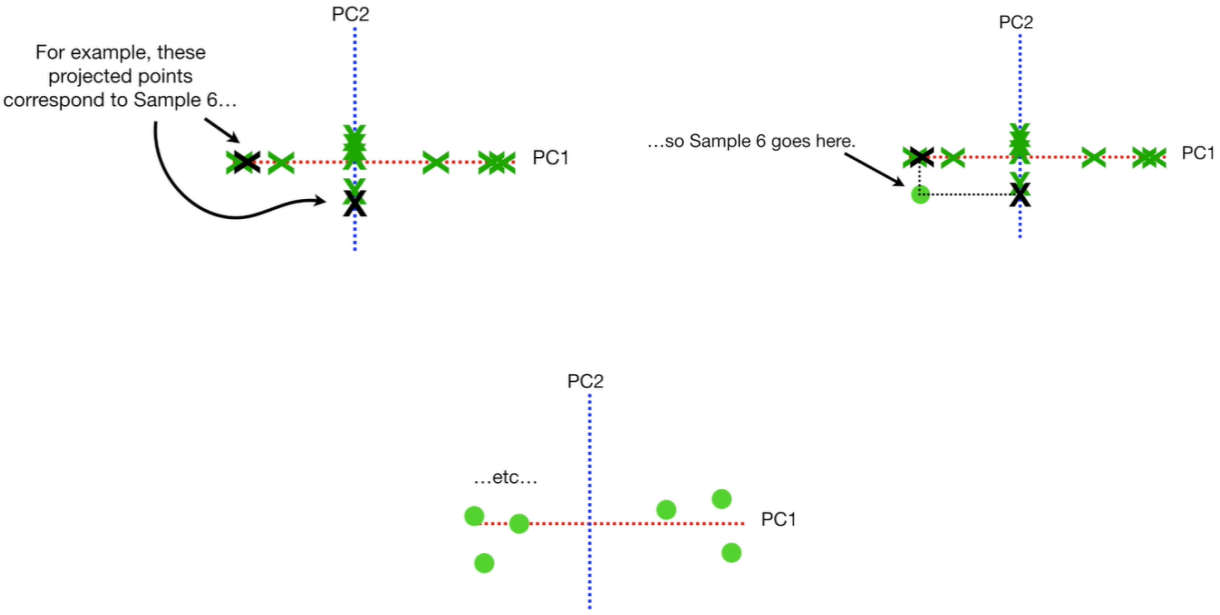
Let's work on PC2 because this is only a two-dimensional graph. PC2 is simply the line through the origin that is perpendicular to PC1 and any further optimization has to be done. The “recipe” for PC2 is - one part gene 1 + four parts gene 2). If we scale everything so that we get a unit vector (same procedure as before) the recipe is -0.242 parts gene 1 and 0.97 parts gene 2. This vector described in this way is the Singular vector for PC2 or the eigenvector for PC2. The components of this vector (-0.242 and 0.97) are the loading scores for PC2.

The loading scores tell us that in terms of how the values are projected onto PC2 gene 2 is four times as important as gene 1.

Lastly, the eigenvalue for PC2 is the sum of squares of the distances between the projected points and the origin.




Now we have PC1 and PC2 and we can draw the final PCA plot. To draw the PCA plot we simply rotate everything so that PC1 is horizontal and then we use the projected points to find where the samples go in the PCA plot.

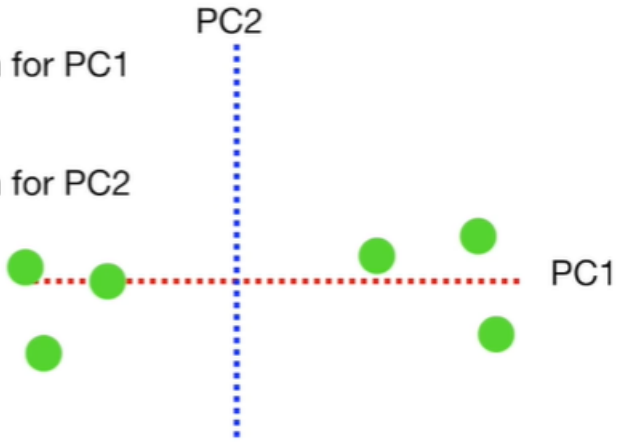



Remember the eigenvalues that we got by projecting the data onto the principal components measuring the distances to the origin, then squaring and adding them together? We can convert them into variations around the origin by dividing by the sample size minus one for the sums.

We can convert them into variation around the origin (0, 0) by dividing by the sample size minus 1 (i.e. $n$ - 1).

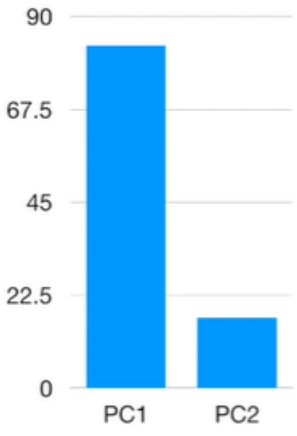
$$\frac{SS(\text{distances for PC1})}{n - 1} = \text{Variation for PC1}$$

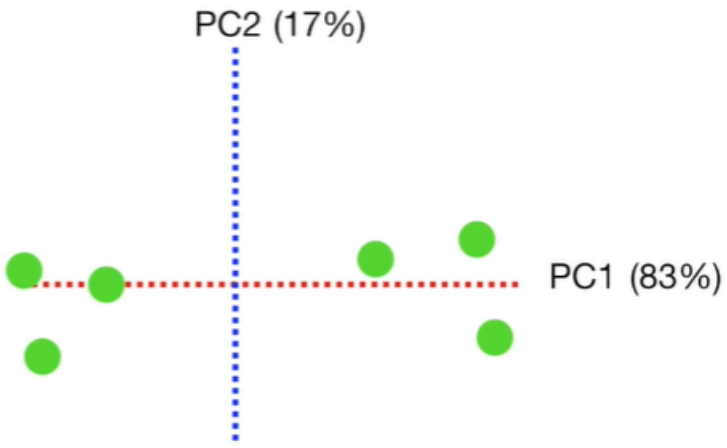
$$\frac{SS(\text{distances for PC2})}{n - 1} = \text{Variation for PC2}$$



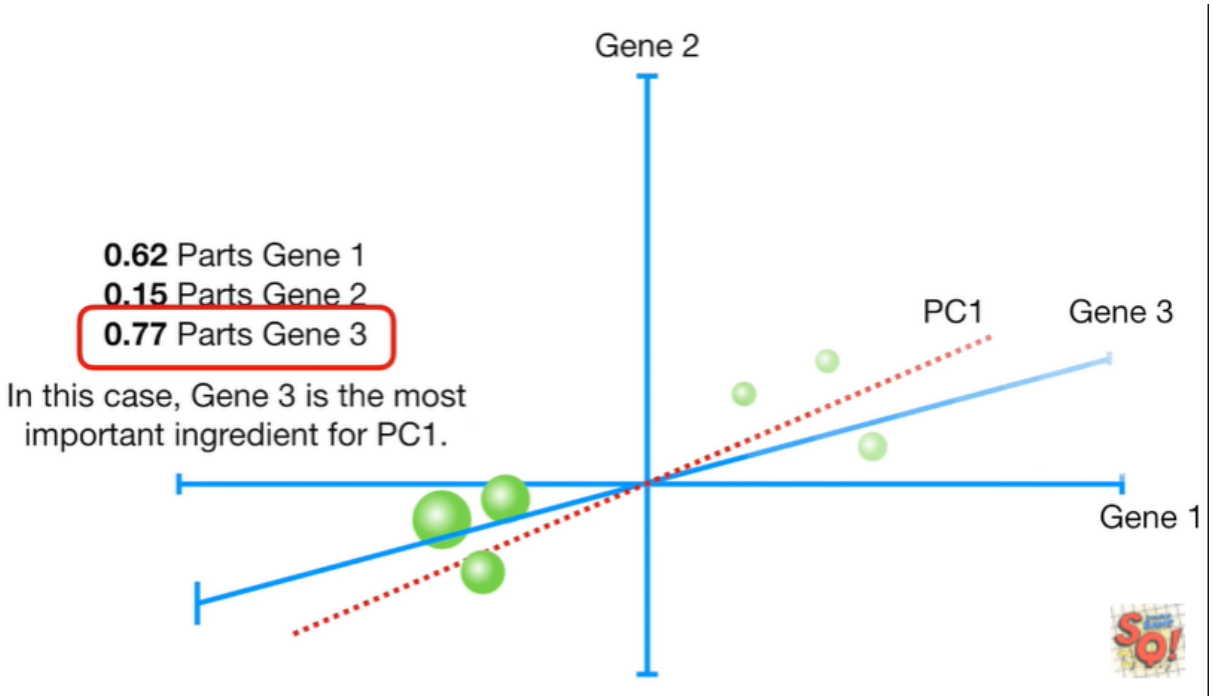

For this example, imagine that the variation for PC1 = 15 and the variation for PC2 = 3 this means that the total variation around both PCs is 15 + 3 = 18 and that means PC1 accounts for 15 / 18 = 0.83 or 83% of the total variation around the PCs, while PC2 accounts for 3 / 18 = 17% of the total variation around the PCs. A **Scree plot** is a graphical representation of the percentages of variation that each PC accounts.

**TERMINOLOGY ALERT!!!!** A **Scree Plot** is a graphical representation of the percentages of variation that each PC accounts for.

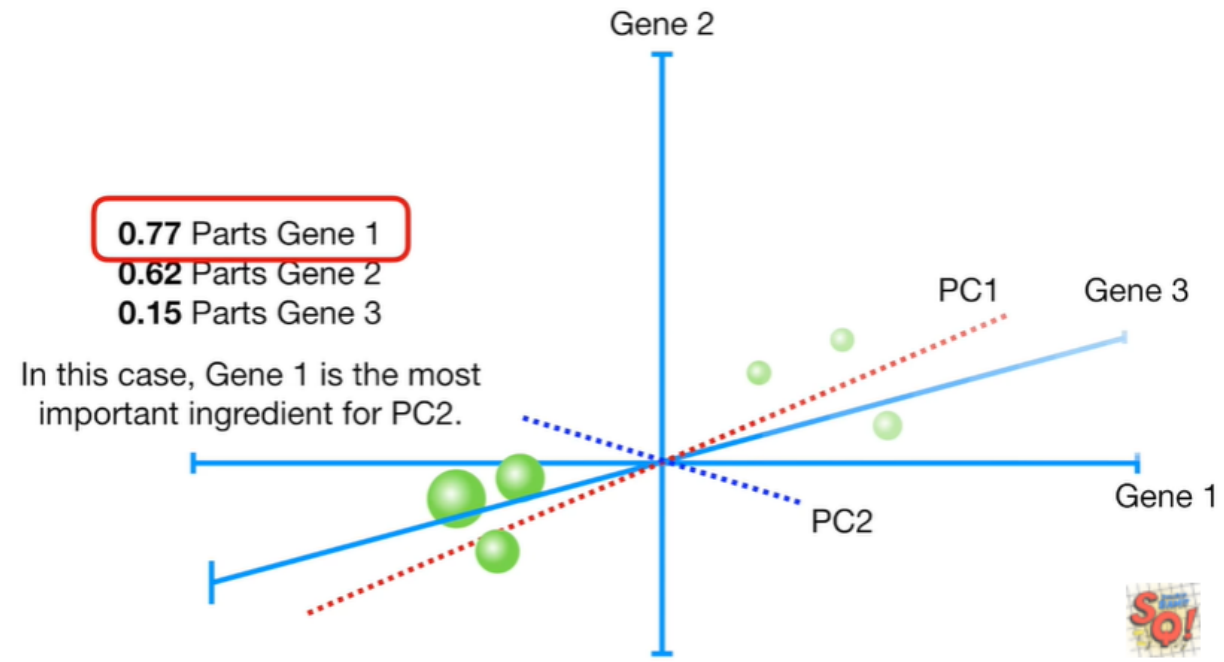


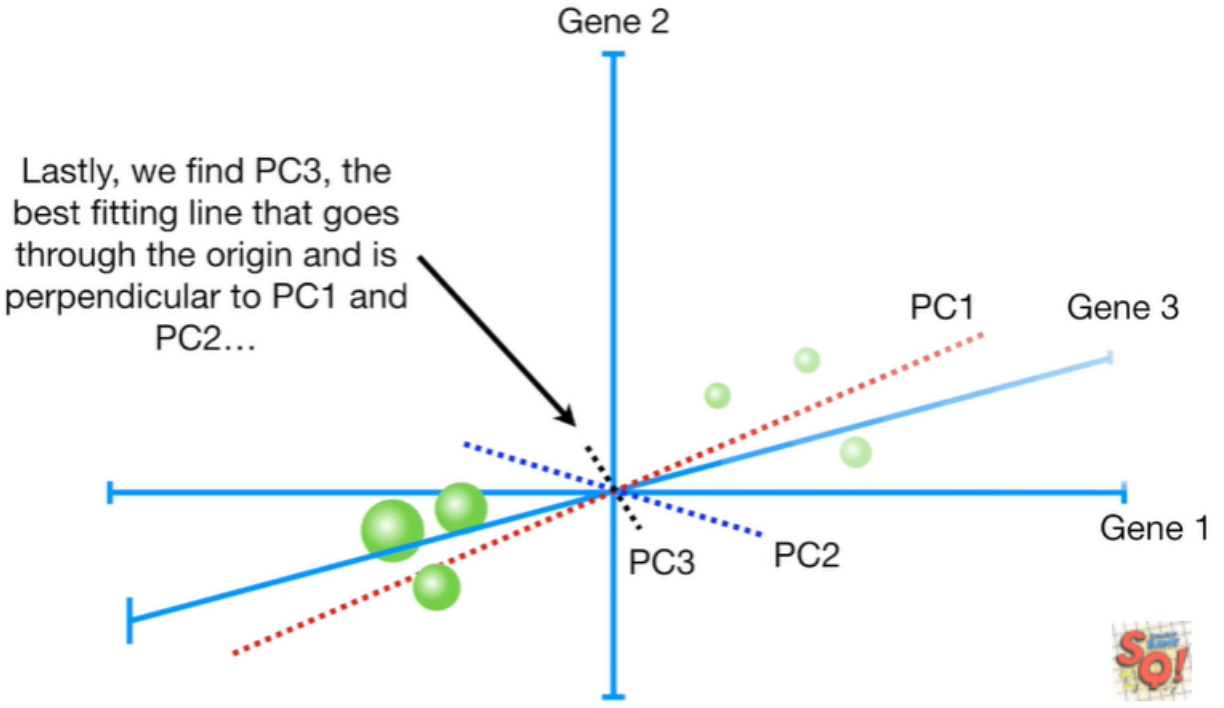
Now let's consider a slightly more complicated example: a PCA with three variables; in this case with three genes is pretty the same as the example before with only two variables. First you centre the data, then find the best fitting line that goes through the origin just like before. And just like before, the best fitting line is PC1 but the "recipe" for PC1 now has three "ingredients". However, in this case gene 3 is the most important ingredient for PC1, i.e. is the gene accounting for most of the variability between our samples.




Then you find the best fitting line for PC2 and it goes through the origin and is perpendicular to PC1. In this case gene one is the gene accounting for the majority of the variability along PC2.

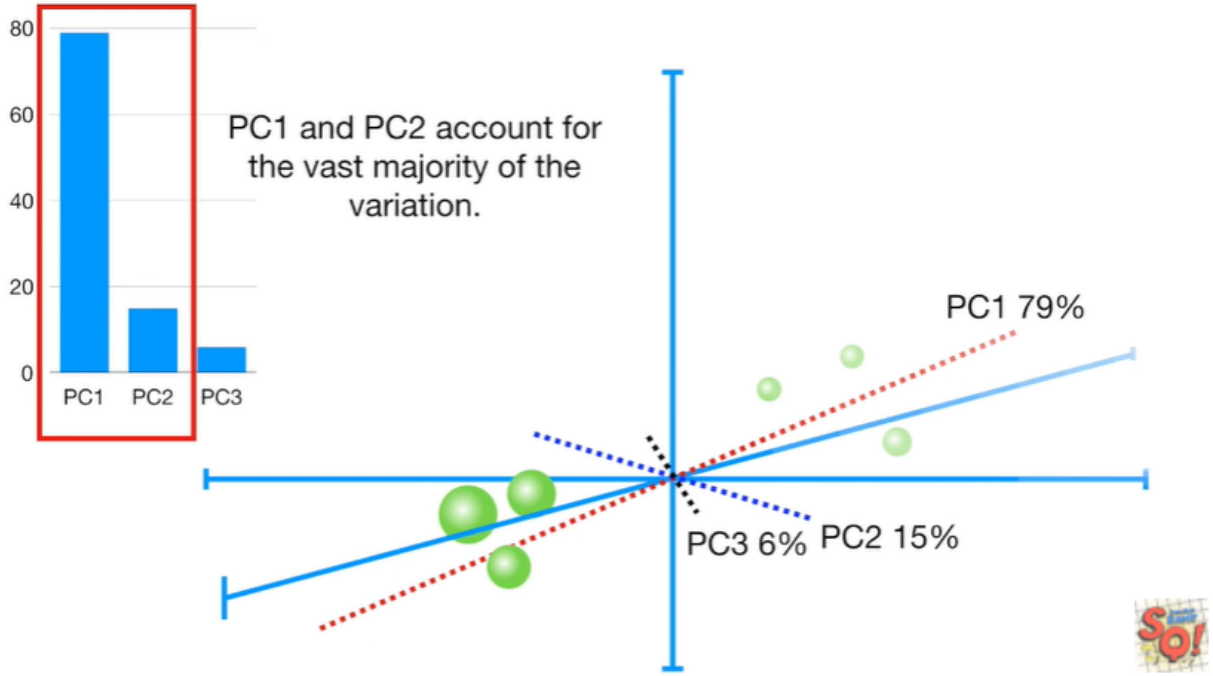



Then we define the best fitting line as PC3 that goes through the origin and is perpendicular PC1 and PC-2.

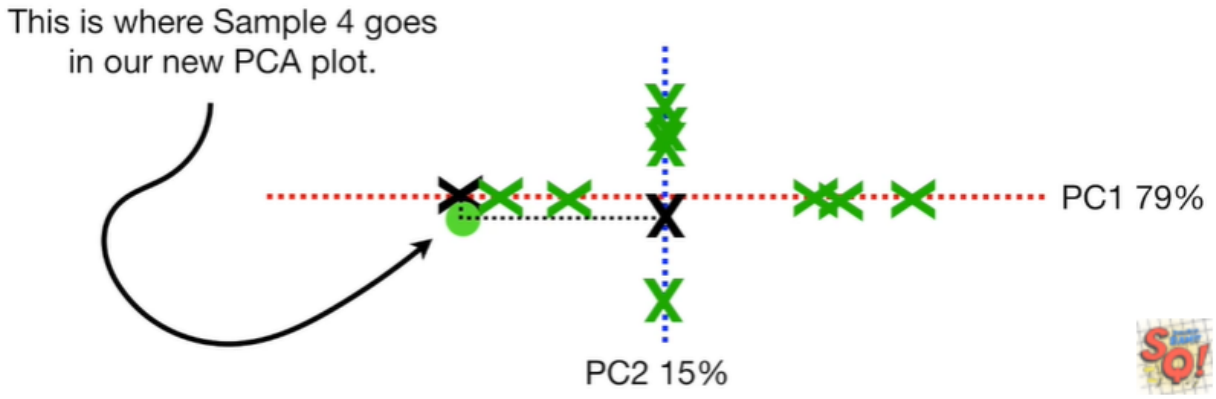



If we had more genes we just keep on finding more and more principle components by adding perpendicular lines and rotating. In theory, there is one PC per gene or variable but in practice the number of PCs is either the number of variables or the number of samples, the one that is smaller.

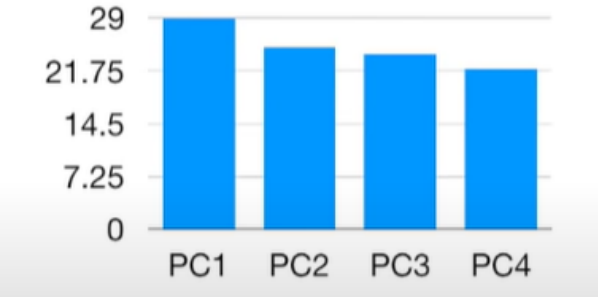
So once you have all the principle components figured out you can use the eigenvalues (the sums of squares of the distances) to determine the proportion of variation that each PC accounts for. In this case PC1 accounts for 79% of the variation, PC2 accounts for 15% of the variation and PC3 accounts for 6% of the variation.




Looking at the Scree plot PC1 and PC2 account for the vast majority of the variation. This means that a 2D graph using just PC1 and PC2 would be a good approximation of this 3D graph since it would account for 94% of the variation in the data. To convert the 3D graph into a two-dimensional PCA graph, we discard everything except PC1 and PC2 and then project the samples onto PC1 and PC2, then we rotate so that PC1 is horizontal and PC2 is vertical. The rotation just makes the graph easier to be looked at. Then you plot your points by using the coordinates of the plotted points on the 2 PC.




The eigenvalues for PC1 and PC2 2D graph would still be very informative in this case.

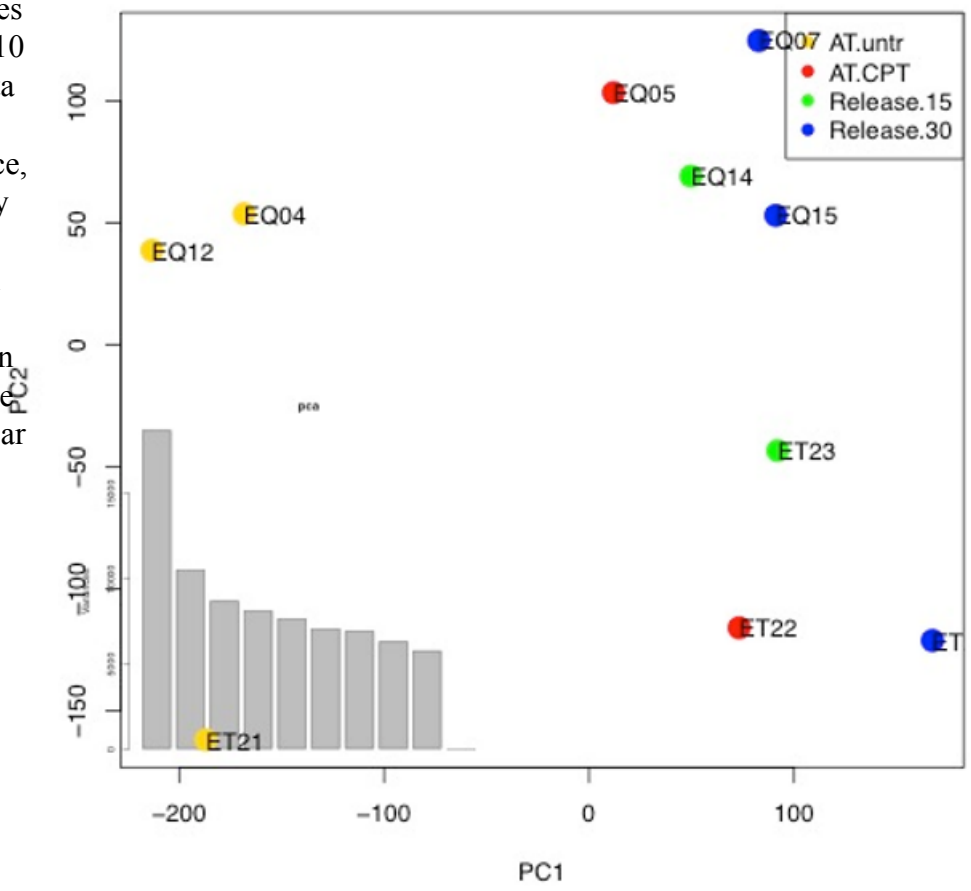



If the scree plot looked like this where PC3 and PC4 account for a substantial amount of variation, just using the first two PCs would not create a very accurate representation of the data. However, even a noisy PCA plot like this can be used to identify clusters of data.

📌 **PCA is a linear dimensionality reduction score**: the variance has a linear dispersion, we are drawing a line and not a curve to fit the data.

# PCA

here there are 10 samples (the grey bar) possibly 10 PC but you c an see thta the first explain the majority of the difference, the seocnd one roughly half of the first one. look at the number on each PC are actually referring specifically on hat component, they are splitted, so even if similar does not represent the same thing



❗ **What is critical in the PCA is the definition of the first component:** the first explain the majority of the variance, the second component explains the 50% of the variance. The last component explain nearly 0 variance and hence is less significative.

**The different PCs are independent** because are based on the expression of different genes, they are based on different weights and represent different things. The last component in the graph above explains a minimal part of the variance (last grey bar).

## Examples

Example1

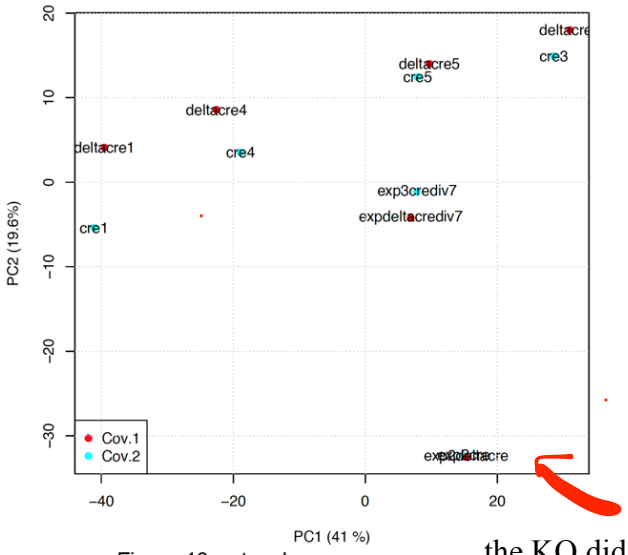



Figure 10: wt vs ko

In this example there are 6 trangenic mice and one WT. This is an inducible KO done on the same fibroblasts in two different conditions. In this way we have the same model with and without the gene. The main difference given by the KO are on the second dimension (PC2).

each couple of point is a mouse and each point in the couple represent the WT or the KO condition

the KO did not worked and we can discard this sample

to differences due to the different animals

We can also see that the readout of the experiment is conditioned by the efficiency of the KO. This is a very important information. after removing the error probablly you will se how PC2 increase its influence

If we directly do differential expression analysis, we do not get significant results because there are differences among individuals that are masking the actual differences. We can build a specific type of differential expression analysis in which we group the animals (**block experiment**) and then as we perform the statistical analysis of this data, the first thing we do is to remove the variance ascribed to the different blocks. This helps you to understand if there is enough difference between the experimental conditions. We can now focus only on the $\Delta$treated / untreated.

✅ PCA helps in identifying the confounding factors that are negatively affecting our analysis.

Example2

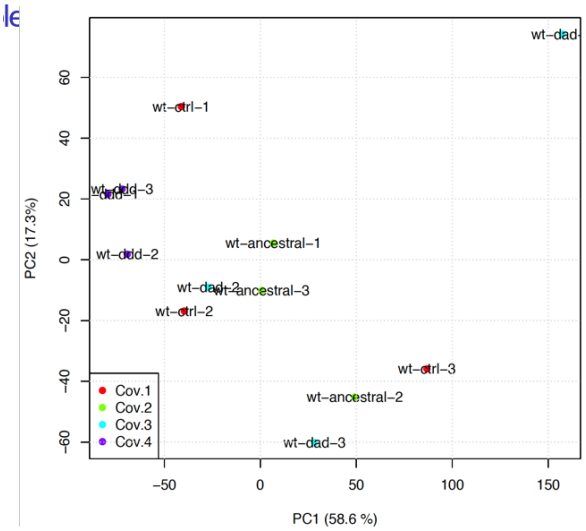



Figure 11: 4 different conditions in 1 cell line

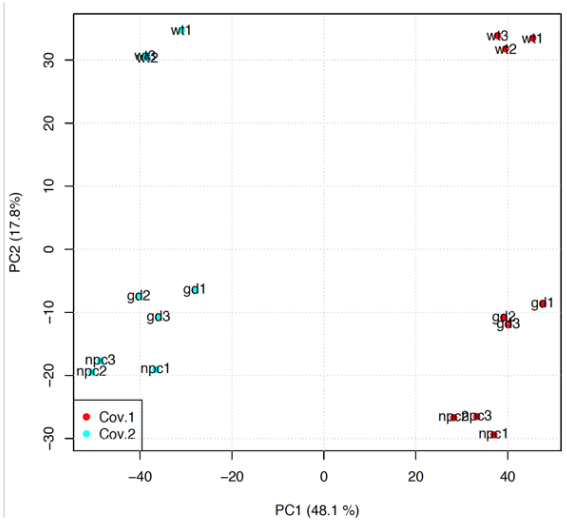
In this case you have only one cell line in 4 different conditions. In this case there isn't a line enabling us to separate the untreated to the treated ones, and what PCA should be able to do is to draw a line that enable to separate the control vs the treatment. So in this second example there is a problem in the analysis. You have to ask to the researcher how they have done the KD. Indeed, in this case, the KD was not done by lentiviral infection but by transfection, whose efficacy was 30%. You don't see any difference because the differences in the transfection are bigger than all the other possible differences.

The differences due to the treatment are masked by inter-individual differences: a fold change of 1 in the expression when the efficiency is so low falls within the noise of the experiment and is not significant.

It is really important to control everything before sending the material for sequencing in a way that we avoid wasting money.

maybe not a lot of genes are changing

Example3




: 3 different cell lines with 2 different conditions

In this third example you have an experiment in which you have two different cell lines (1 and 2) in two different conditions. The first cell line has the control (WT1) and the tretated (T1) sample; this is the same for the second cell line: control sample (WT2) and the treated sample (T2).

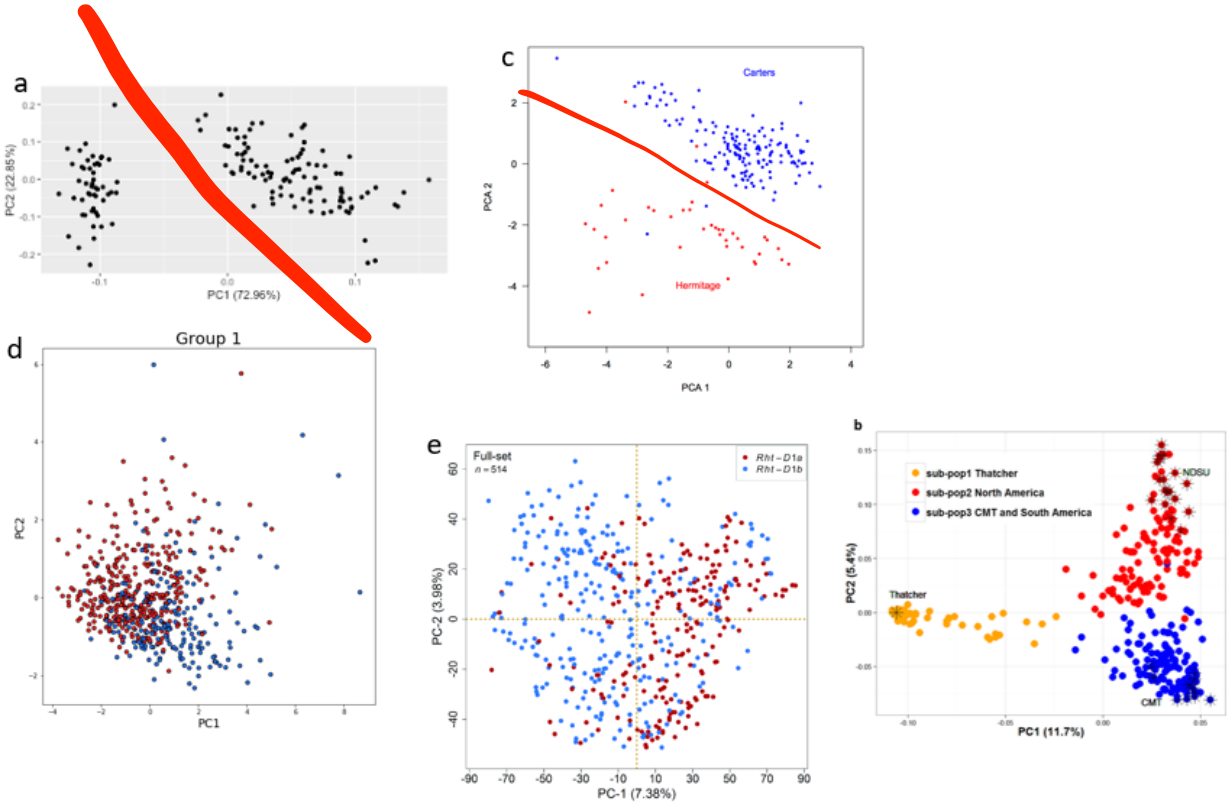
We see that on the 1st component there is clearly a difference between the two cell lines. There is a large difference between the cell lines (PC1, 48.1%), but still a detectable difference in the second PC discriminating different conditions on the same cell line(PC2, 17.8%).

With these data you can perform two different kind of analysis:

- Intersection between the genes that are differentially expressed in both cell lines in the same conditions (ex: WT1 vs WT2 or T1 vs T2).

- Intersection between the genes that are differentially expressed in both cell lines in different conditions (ex: WT1 vs T1 or WT2 vs T2). In this case you can identify differentially expressed genes that are specific for one specific cell line.

**Examples**




In **a)** the PC1 is the most representative. The greatest amount of differences between the two groups is in the first component. There is a very large distance between the two groups and you can note that the first component is the most representative. The two groups are very well separated.

In **c)** the separation between the two groups is mainly on the second component (PC2), which means that there is a wide variance among the various elements of the experiment (this is a single-cell experiment). Since the differences between the two groups are given by the second component, this means that there is a difference but is smaller than the overall differences that exist between the various elements that are representing the two data sets that you are comparing.  It is a little bit more tricky to define the difference between two groups, but you are still able to identify a diagonal that will divide the two groups.

In **d)** the probability that we find something different between the two groups is low because the noise is too big or maybe there is just not enough transcriptional difference. The element are very much overlapping, you can try but the probability to get differential expressed genes is very little.

This latter condition may be the condition of the KO of a membrane receptor in which you can not really appreciating a difference in the downstream signaling.

In **b)** the yellow sample is more different compared to the others because the main difference is in the first component (PC1). The other two are more similar, but can still be divided by the second component (PC2).

**e)** is a noisy sample, but for sure reds are localized on one side and blue on the other. There is the possibility that you loose some of the differentially expressed genes in the noise but still you can identify more or less two groups.

maybe example of KO not going very well

so looking at PCA you can obtain informations

# ✏️ Exercises about PCA

Osimertinib experiment:

- PC9 (EGFR mutated, sensitive to Osimertinib) VS MOCK (DMSO)
- Acute 500 nM osimertinib for 24 hrs VS drug tolerant persister (DTP) treated with 500 nM osimertinib for 21 days (chronic)
- Last time we have made the count table for the RNAseq counts.

Exercise to do:

Calculate PCA and plot it:

- Install in the docker the docker4seq package (https://github.com/kendomaniac/docker4seq)
    1. Open Rstudio and create a R script with the command found in the Github of the Professor (link in the slides)

```
install.packages("devtools") #install devtools
library(devtools) #add the library packages of devtools
install_github("kendomaniac/docker4seq", ref="master") #install docker4seq from gitHub
```

    2. Save the file as a .R file and put it in a folder of your choice (better if in a folder of easy access since you will have to mount it on the docker)
    3. Open the prompt and run the r4 docker mounting in the home the folder containing the .R file (remember to put the option -v and to add ":/home" after the folder path) or only adding the .R file with `docker cp`
    4. When inside the docker go to the home with `cd` and check if the folder is present with `ls`
    5. Enter the folder with `cd` and then use `Rscript nameofthefile.R` to run the .R file with the command for docker4seq installation
    6. Exit and commit

> ❗ I did this without mounting, simply creating the file already within the docker with nano (with `nano nameofthefile`, if the name is new and does not correspond to any other file, nano will create a new file) . I did this also for the other packages and it worked.

- use the PCA function: this function generates PCA plot from counts, FPKM and TPM tables from RNAseqCounts outuputs. You have to adjust the arguments before running it (see after)

```
#PCA function
library(docker4seq)
pca(
  experiment.table = "./_counts.txt",
  type = c("counts", "FPKM", "TPM"),
  covariatesInNames = FALSE,
  samplesName = TRUE,
  principal.components = c(1, 2),
  legend.position = c("bottom", "bottomleft", "left", "topleft", "top", "topright",
    "right", "center"),
  pdf = TRUE,
  output.folder = getwd()
)
```

**Arguments of the function**

**experiment.table** wants the file name and the path of the counts, FPKM or TPM table file (the input file, the one we generated last lesson).

**Type** = counts because we are using row counts. We can use TPM also but here we use the counts because the file we generated is a cpm type of file.

**CovariatesInNames** is a boolean value indicating if covariates are inserted after \ in the filename the files have nameofthesample_cov#. In our case: cov 1 = wt, cov 2 = acute and Cov 3 = chronic treatment. We can assign 3 different colors for the different covariated.

**SamplesName** = TRUE if we want the name plotted over the dot. If it is set false (remember to change the name of the output file if you want to run it multiple times) you do not see the names.

**principal.components** are the elements you want to plot. Usually you plot PC1 and PC2, but you can also plot PC2 and PC3 for example. This is useful when you do not see any differences in the first and second. if you do not see differences produce the same set of gene expression

**legend.position** is a character string indicating the location of the covariates legend

**pfd =** TRUE if results has to be saved in a pdf. False if not (but we do want that pdf so we set it true).

**output.folder** = path/of/the/output/folder ( ❗ it should be working but actually it doesn't - I tried either with or without parenthesis) or simply getwd if you want it to be saved in the working directory.

> ❓ I still do not understand how to give the output file the name I want.

this is a quality control you are evaljuating if the bioogy you want to extarct is robust enough to show what you want to see, a quality control but on a biological question

> ❗ PCA is always a quality control on the experiment, but a quality control on the biology of my experimental question: *do I see the difference I expect?* If not, you may have set your experiment wrong.

### Run PCA

> ⚠️ PCA function don't require that you load the data inside R, but need a tab-delimited file.

First try to do the PCA inside Rstudio, set as working directory the forlder in which are the data. If don't work get inside the docker and use the PCA function in R inside the docker. The output file is a .pdf file. Keep the file, we will discuss on it in the next lesson. The outpuf file will contain the genes that are differentially expressed in the acute or chronic treatments.

> 📌 To look at the PCA function and parameter, you can go in the Rcasc website, references and search for PCA function.

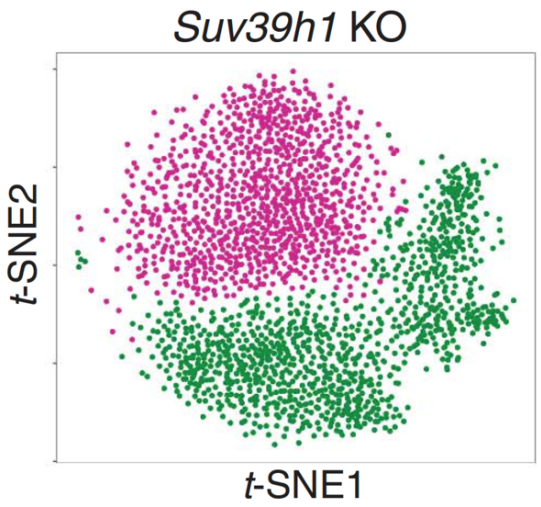
## scRNAseq and tSNE

> 💡 **Youtube video explaining tSNE**
>
> https://www.youtube.com/watch?v=NEaUSP4YerM

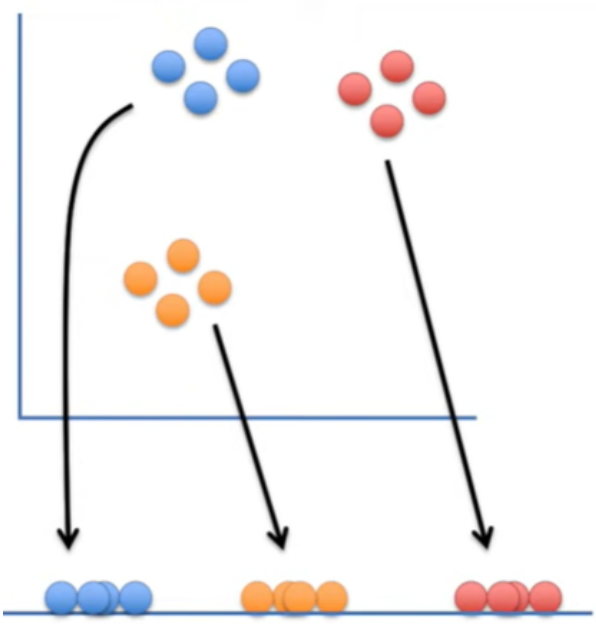
Single cell RNA-seq experiments are more noisy and contain more 0s (there will be more genes whose expression level is 0). For a matter of higher complexity, PCA is not sufficient to make a separation. PCA works well when the first 2 principal component account for most of the variation in the data and so it does not work well for complicated datasets, such as the ones of single cell RNA-seq. A **non-linear dimensional reduction method** is hence needed: we are still reducing the dimensions of our data but the relationship with the original data is not clear anymore. By reducing the dimensions we are separating also the elements, but as we will see later, this is not a proper clustering. Speaking of this separation, here a curve element is required to separate all the data that we have, not a line as for linear dimensional reduction.

There are two main non-linear dimensional reduction methods that are used: **tSne or UMAP**. tSNE was designed to handle big data in the 70ies but it was not biology tailored, at least in the beginning.

This is an example of a tSNE plot. The violet are clearly separated from the green, even though there are some points in the middle. There is only one group of violets and two groups of greens. This is to show that with transcriptomic data two groups can be separated.




The greens are separated into 2 groups.

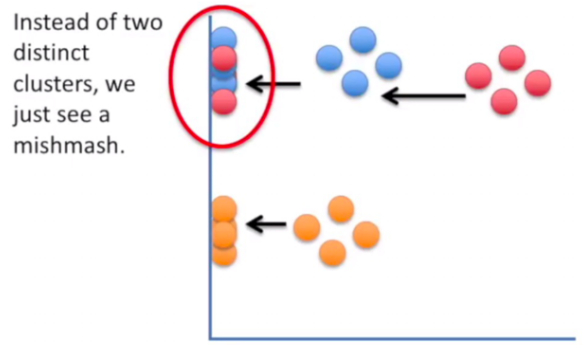


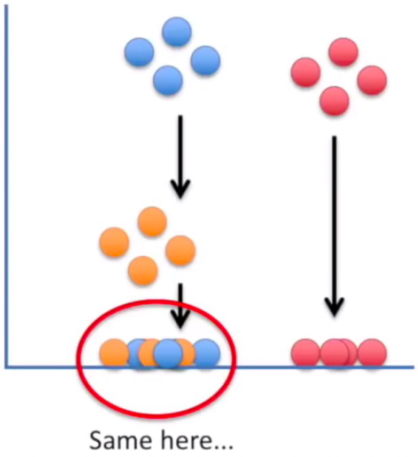
What tSNE does, is to take a highly dimensional dataset and reduce its dimensionality to create a representation that is simpler but retains most of the initial information. The goal of tSNE is getting a dimensionally reduced and easier to visualize representation of the initial data in which we still the main differences between the data.

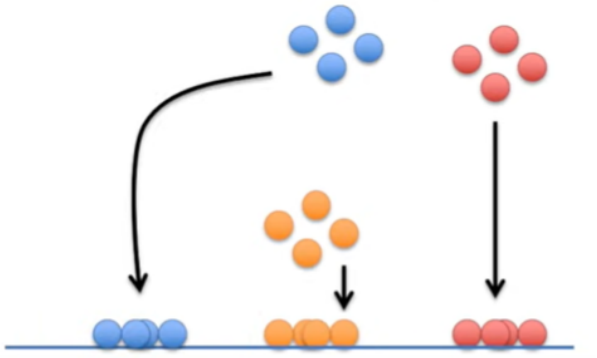
To understand how tSNE works, let's consider this simple dataset:

This is a basic 2D scatter plot, which is transformed on a 1D plot by tSNE.

If we simply **projected the data onto either the y or x axis**, some clusters will be nicely separated and other not, but either way this **does not preserve the original clustering**. We need better separation also in the projection, without clusters coalescing into one.

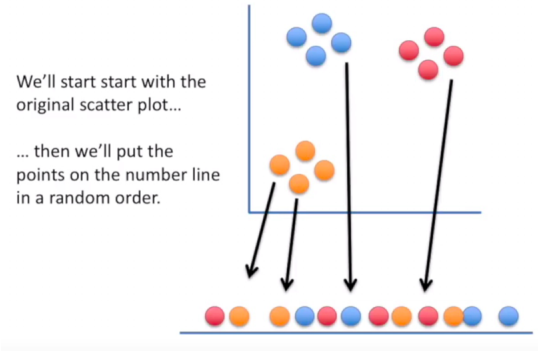


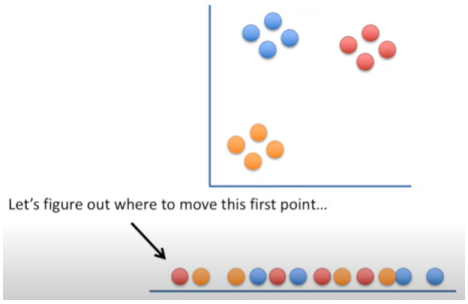



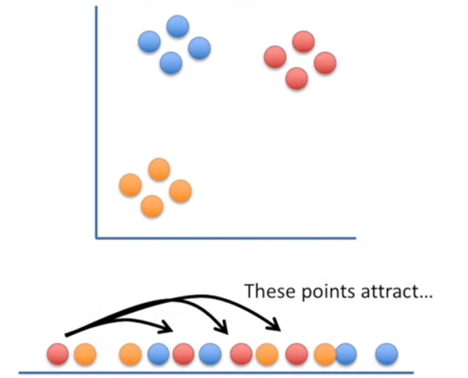
tSNE finds a way to project data into a low dimensional space (which in this case is the 1D line) so that the clustering in the high dimensional space (here the 2D scatter plot) is preserved.

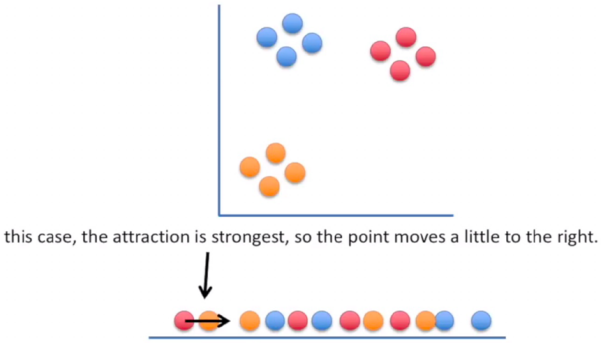
**The initial position in the new low dimensional space is random** and then from there tSNE moves the points a little step at a time until successfully clusters them.

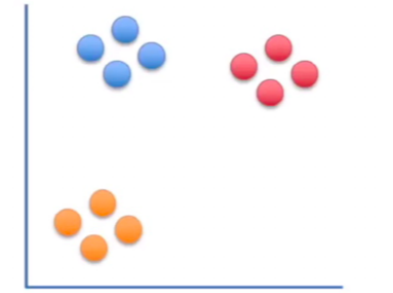


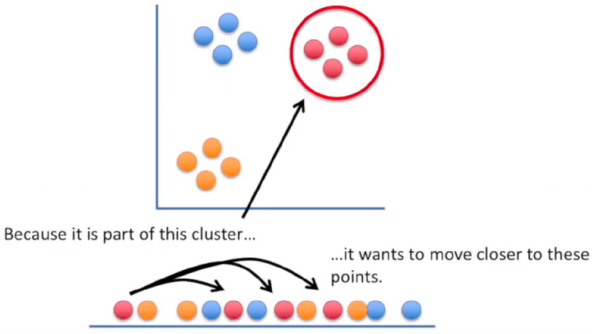

The reds would like to be very near to other reds, as well for yellows which are attracting yellows and blues with blues. tSNE moves one the most left points on the line (the red in this case) to make it closer to their red companions and far from the other yellow and blue points. This is done little by little, thereby constructing the clusters. Same concept of Rubik's cube.

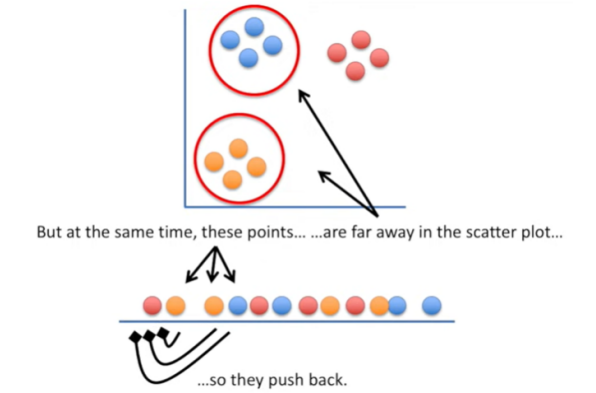


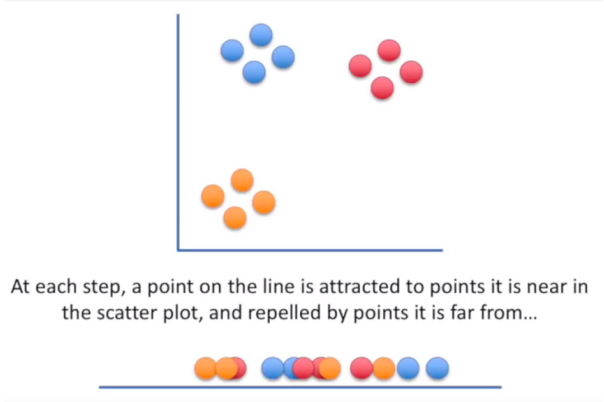





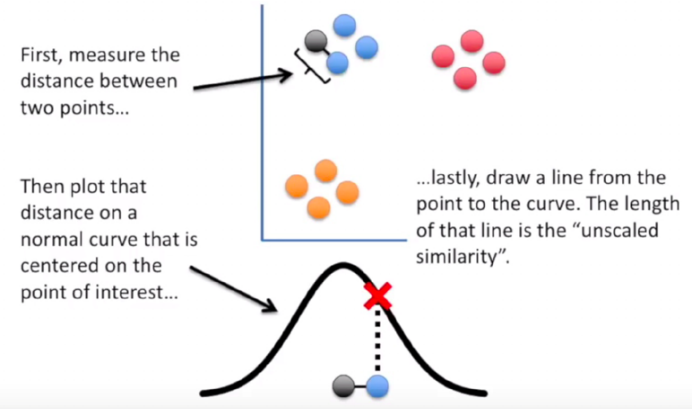




← This will be the final result.

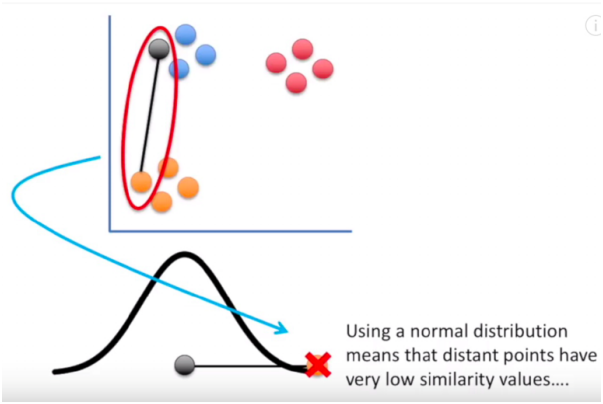
We know what we would like to get and we have to find a mathematical relationship between them so that we can preserve their clustering also in the low dimensional space.

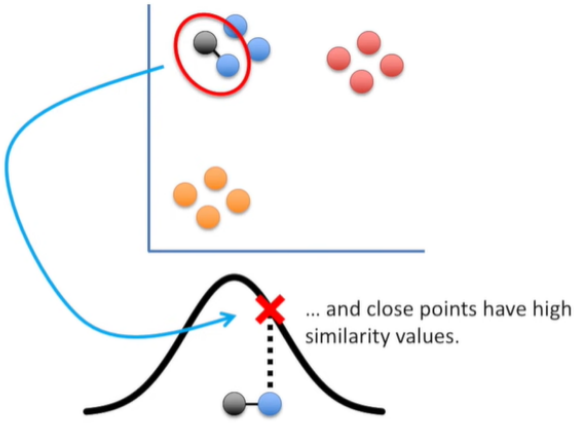
This is the idea at the bases but how does it actually work?

We start from a cell/point in the real space and we use some kind of distance measure to define **similarity** between each of the other points. We take the normal distribution centered on that point/value of interest and then we plot all the distances between the cell of interest and all the other cells on that curve, pair by pair. Cells that are far in the real space will be far also in the normal distribution. We then draw a line from the other point to the curve and the length of this line will be the **unscaled similarity.** Then the unscaled similarity will be calculated in this way for each pair of points in the space.

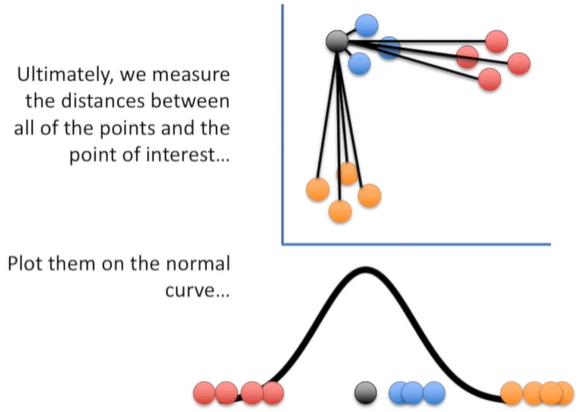



Cells with a similar expression profile to the one I am considering at the moment will be near to the center of the normal distribution, while the cells that are different from my cells will be in the tail of the distribution. Doing this The distance between cells of different clusters will be great and hence they will fall on the tail of the normal distribution.



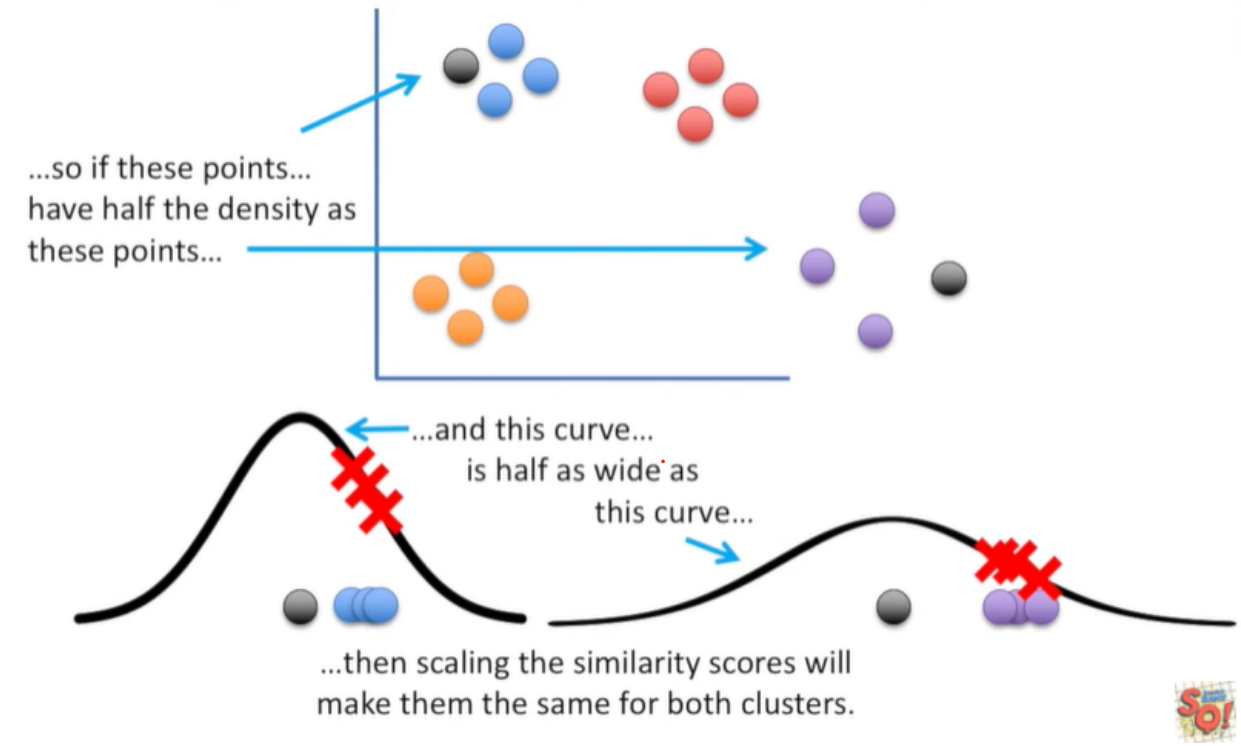

We repeat this procedure for every couple of points and then we measure all the distances from the



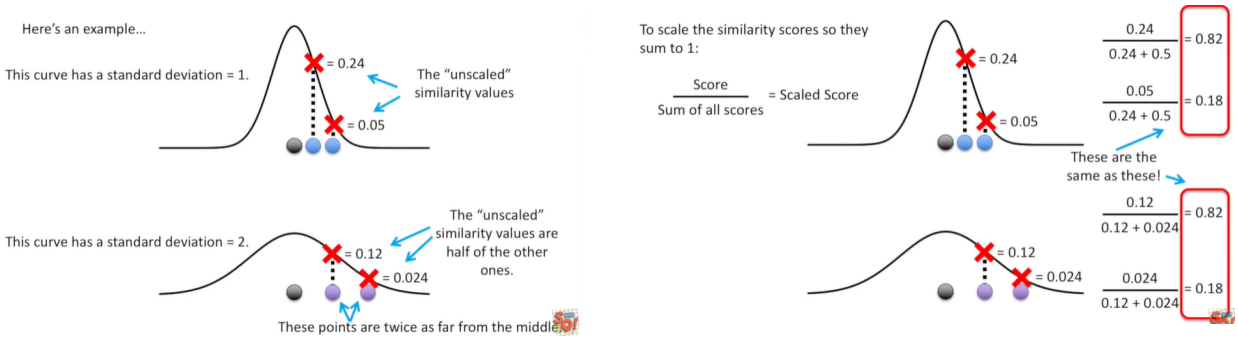

points to the curve to get the unscaled similarity scores with respect to the point of interest.

The next step is to scale the unscaled similarity measures so that they add up to 1 (their sum must be 1). We need to do this because we need to do a sort of **normalization of the distance**: we need to represent distances the same way.

To better understand this concept, let's add another cluster (the violet one) with half of the density of points of the other clusters. The density of the data near the point of interest influence the width of the curve, with less dense regions having wider curves. more shallow distribution



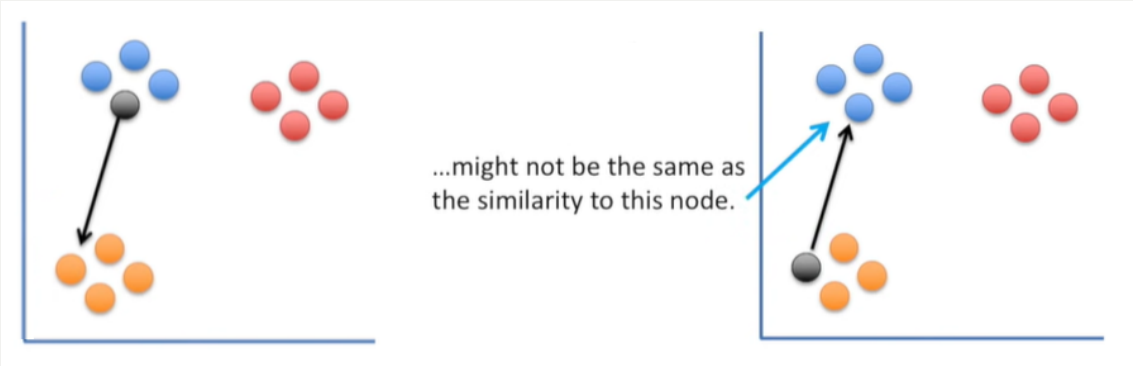

You cannot compare different distributions basically: you have to normalize the data first, and to do this you have to calculate a **scaled score,** which is equal to the score of the point divided by the sum of all the other scores (formula in the image below). In the next pictures a mathematical example is reported:

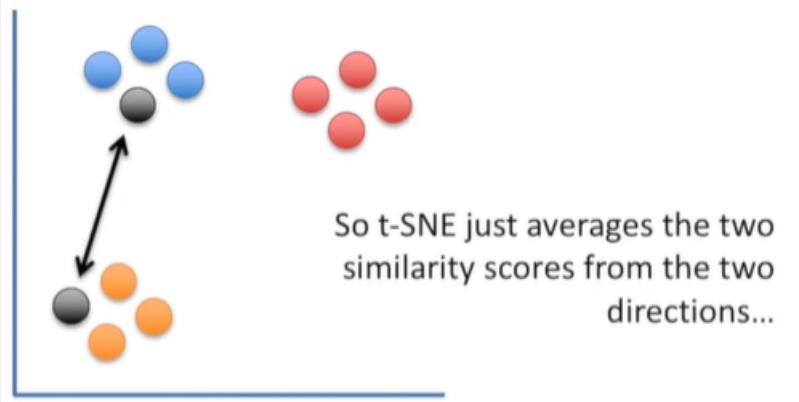



Note that even though the unscaled scores are very different, the scaled scores are equal: we have to weight in the density of points and this is the correct way to do it. We take each of the scores representing the distance and we divide them by the sum of the scores to obtain scaled values. This scaled values have a total sum of 1 and can hence be compared.

This normalization is essential if we want to compare groups that have a different density of points (i.e. the mean distance between 2 points is different).

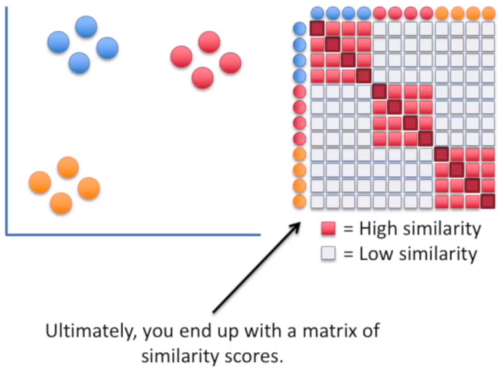
! Because the width of the distribution is based on the density of the surrounding data points, the similarity score between two points may change with the direction:



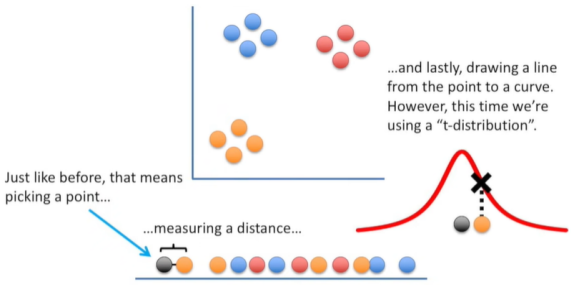



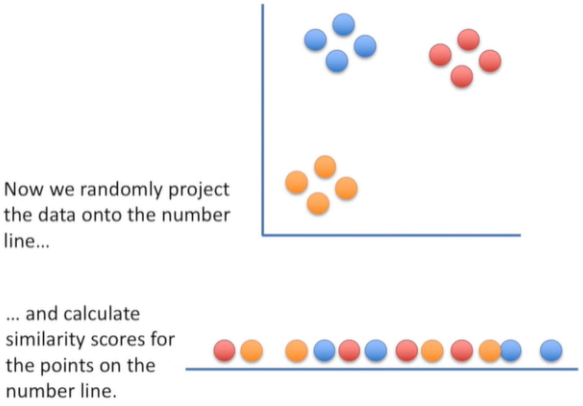
💡 **Perplexity** parameter in tSNE is the parameter that weights in the density.


In the end we will get a **squared similarity matrix** → each row and columns represent the similarity scores calculated from that point of interest. This is hence a matrix telling the difference between the different cells. The diagonal is the line on which the similarity of a point with itself falls. In theory these values will be the maximum values (a point has 100% similarity with itself) but actually for the clustering purpose this is is counterproductive and hence tSNE sets this value as 0.

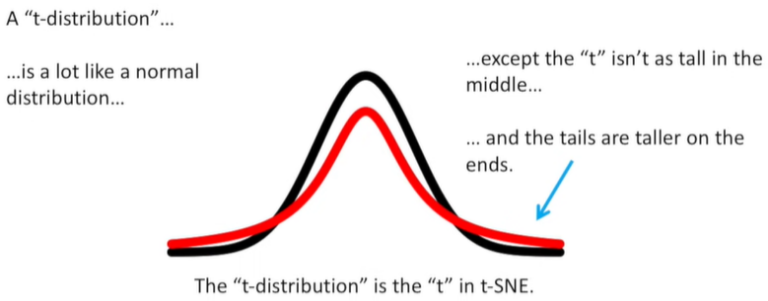



Now we have to repeat this procedure of calculating similarity scores in the new dimensionally reduced space. The procedure is the same but instead of a normal distribution we use a **t-distribution.**

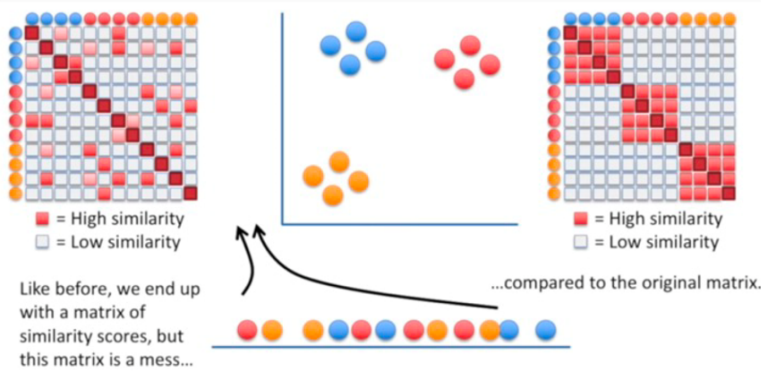


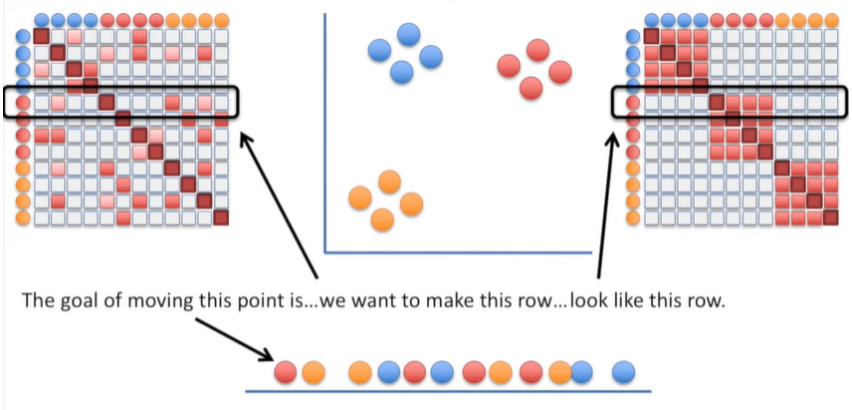



The t distribution's tip is sharper and not as high as the normal tip. With this distribution it is easier to keep those data that are more similar in the high dimensional space also near in the dimensionally reduced space. the t distribution gives the name to t-SNE.

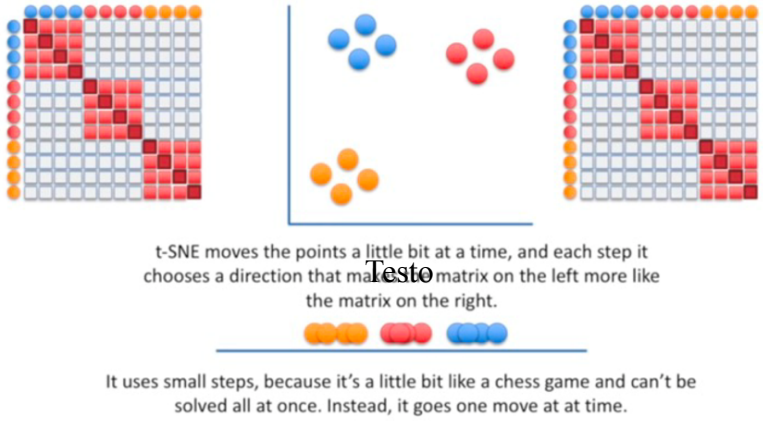



So, using a t-distribution we calculate again all the unscaled similarity scores for all the points and we then scale them as we did before (score/sum of the scores). We end up with the same matrix of before, except that it is a mess and does not represent the matrix of the original space (original matrix).






t-SNE takes one point and make subtle changes trying to put everything that is similar together. After one move, it calculates again the distances and the matrix, comparing the matrix of the dimensionally reduced space to the original one up until the 2 matrices are comparable (ideally equal).




Testo

In the end we reconstruct the original matrix in the dimensionally reduced space. This is done in multiple iterations. Actually there is no guarantee that you reconstruct your matrix perfectly.

> ❗ **Dimensional reduction is actually different from data clustering:** if you look at the separation in the smaller direction, we can see by eye that we have 3 groups of cells. This is defined by our way of looking at the data (for example if the professor looked at us sitting at lecture from the front he would group us in some ways which are different if he made clusters of us from the side → we cannot make groups on the basis on our way to look at the data after dimensional reduction). We want to have an unbiased system assigning the points to a group by similarity of transcriptional profile.
>
> **Clustering requires and is usually done after doing the dimensional reduction** because is simpler. data reduction is simply a way of representing the data but without assigning them to a cluster, just a way to represent your data from a complex situation in a 2 dimensional way

With tSne you can assign your preferred distance matrix: you can measure distance in various ways, and this is up to the operator to decide. When you have decided, tSne makes the representation for you. This renders

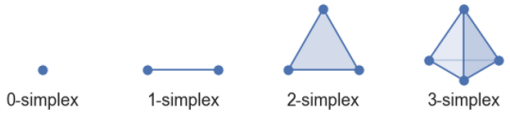
the system flexible and adaptable for all purposes.

# UMAP (Uniform Manifold Approximation and Projection).

> ❗ Youtube video for better understanding: https://www.youtube.com/watch?v=eN0wFzBA4Sc

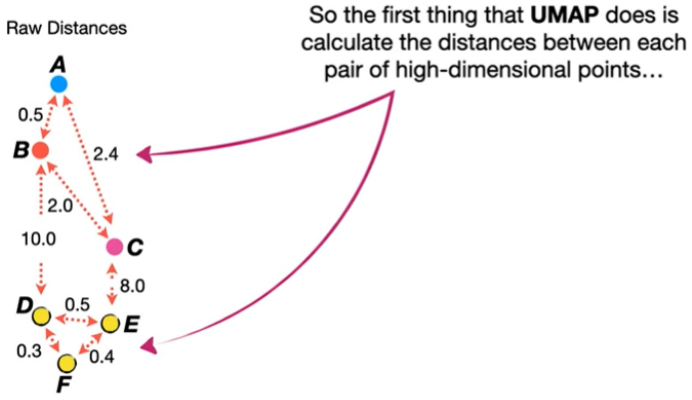
More robust than tSne because it **focus on the big picture, the overall representation of the data** and not the local representation of the elements. Basically it tends to cluster together in the final output the more similar samples, so it is useful for the identification of similarities.

It is also **faster** than tSNE.

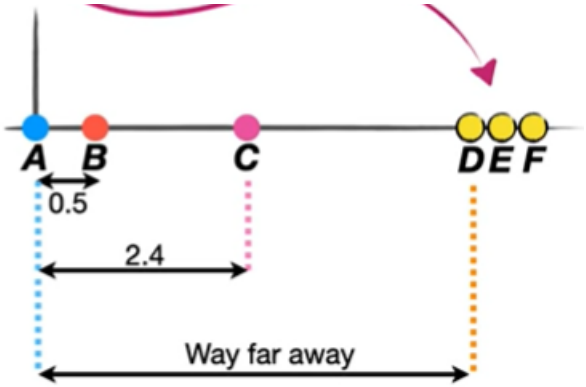


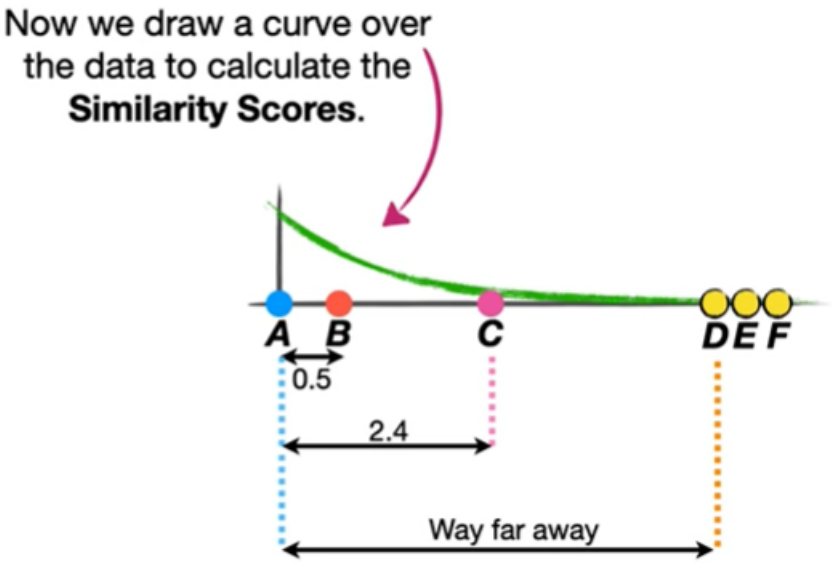

UMAP tries to change the structure of our data by looking at some kind of organization. It tries to define aggregation of points used to induce the position of the single points that we have.

Again UMAP uses a similar concept to tSNE: it does not simply projects the point onto the axis but rather calculates similarity scores that help identify similarities that can preserve the disposition of points also in the low-dimensional space.

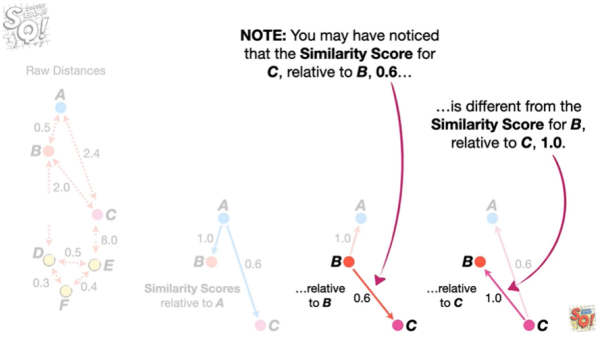


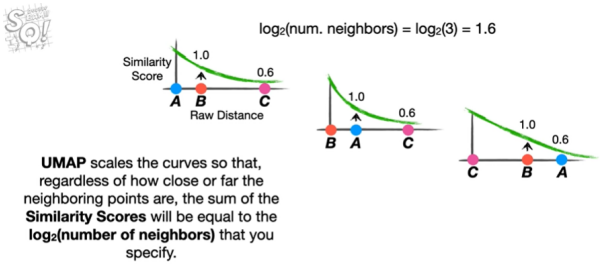

Let's take point A into consideration: after having calculated these raw distances it puts the points on a graph, with the point of interest (A) on the 0. The other points will be at the corresponding distances on the graph, with too far points (i.e. points in other clusters) put way far away from the rest.



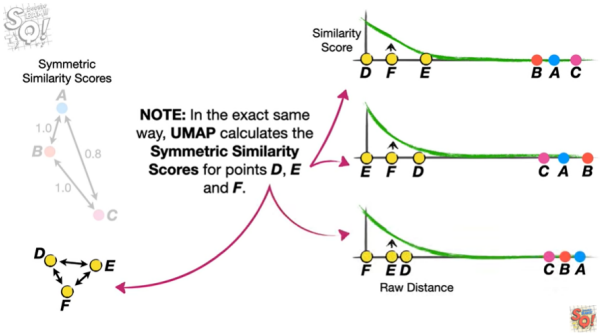




Then a curve is drawn to calculate the similarity scores. The shape of this curve depends on the number of high-dimensional neighbors you wan each point to have (**nearest neighbors parameter,** usually 15, here for this small dataset is 3, and includes the point of interest itself → only 2 other points near the point of interest). It is drawn so that the **sum of the y coordinates of the points is equal to $\log_2$(number of nearest neighbors).** This is a little like the weighing in tSNE. The points that are too far from the point of interest will have a y coordinate which will be almost 0.



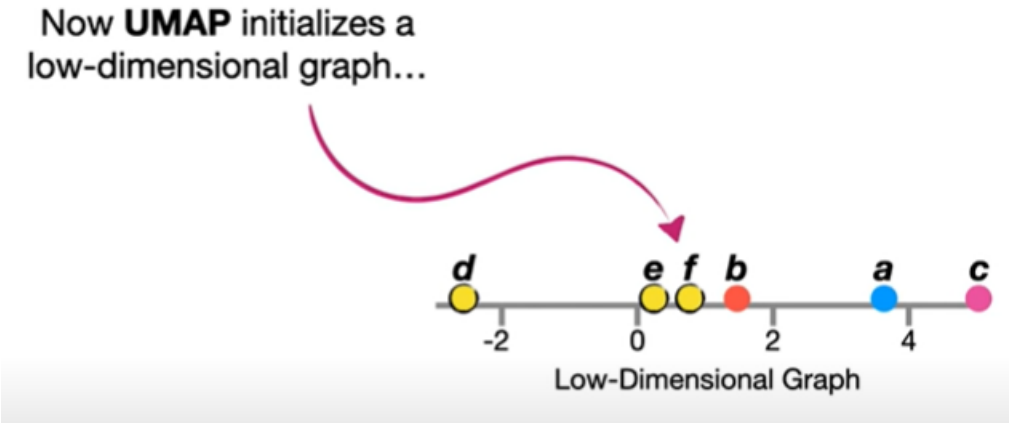



Similarity scores are not symmetrical, but UMAP makes them symmetrical (not discussed, it is like making an average).



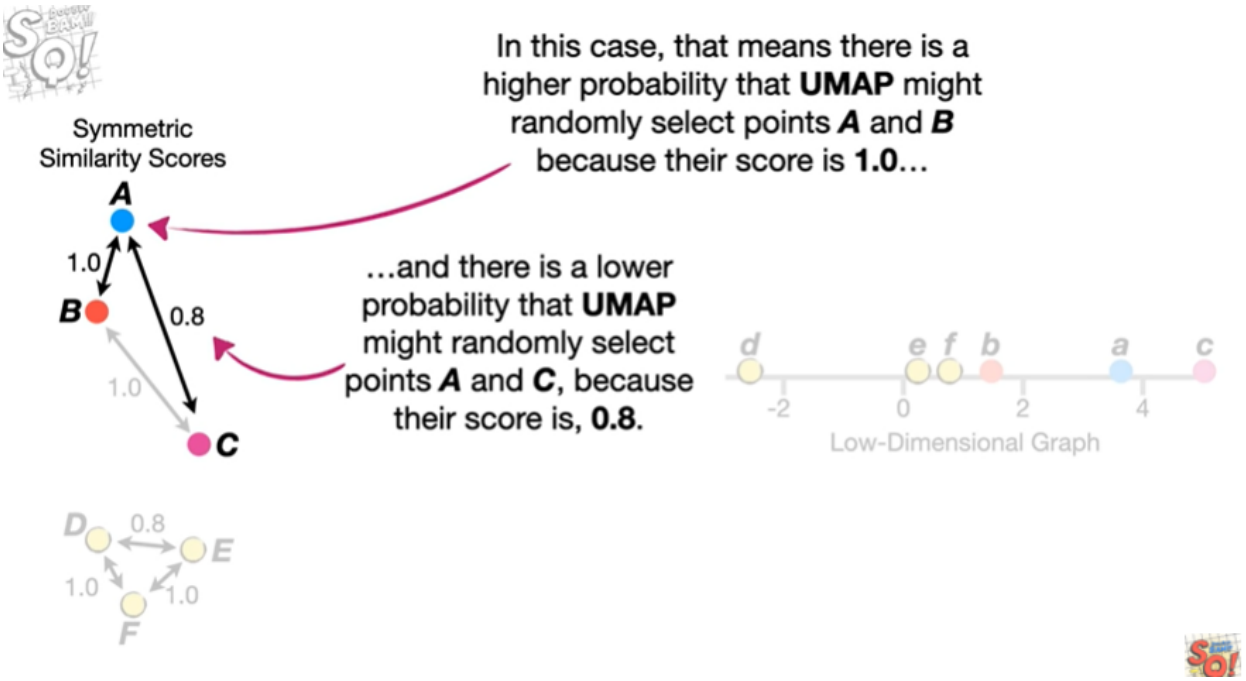

UMAP calculates the same similarity scores of all the other points.

Then UMAP initializes the low dimensional graph.




As we can see it is not ideal, because point b needs to be closer to a and point b need to be further from f to make this low-dimensional graph show the same clusters of the high-dimensional data.

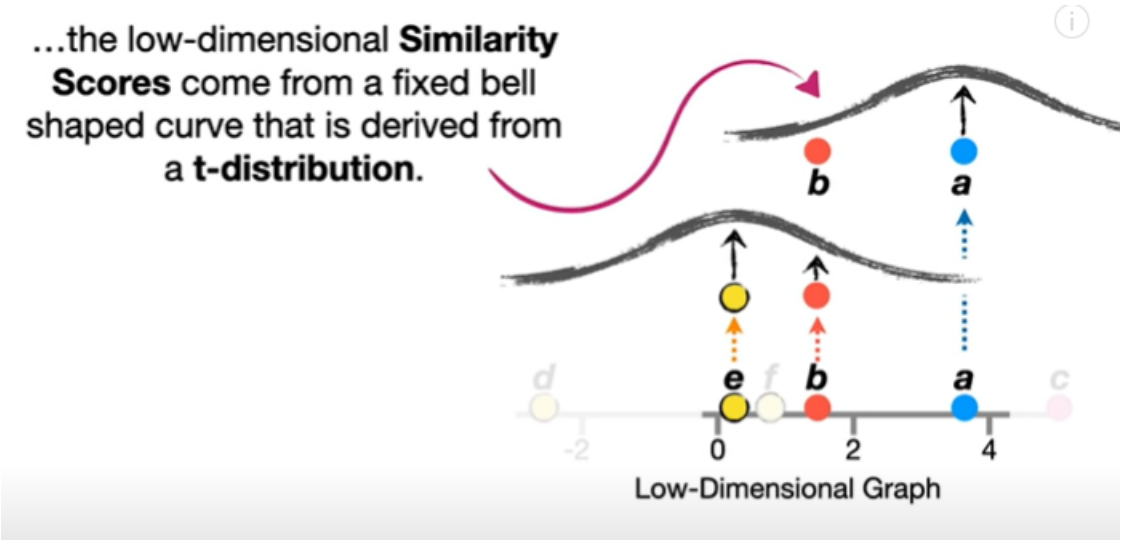
To do this, UMAP generates aggregates of points (different from tSNE where we aggregate only later, here we first aggregate and then we reduce to lower dimension). It picks 2 low-dimensional points that should move closer together in a random way, with a probability which is higher for points that are closer together in the high-dimensional space (i.e. points that have a high similarity score in the high-dimensional space). Let's suppose it picked A and B.



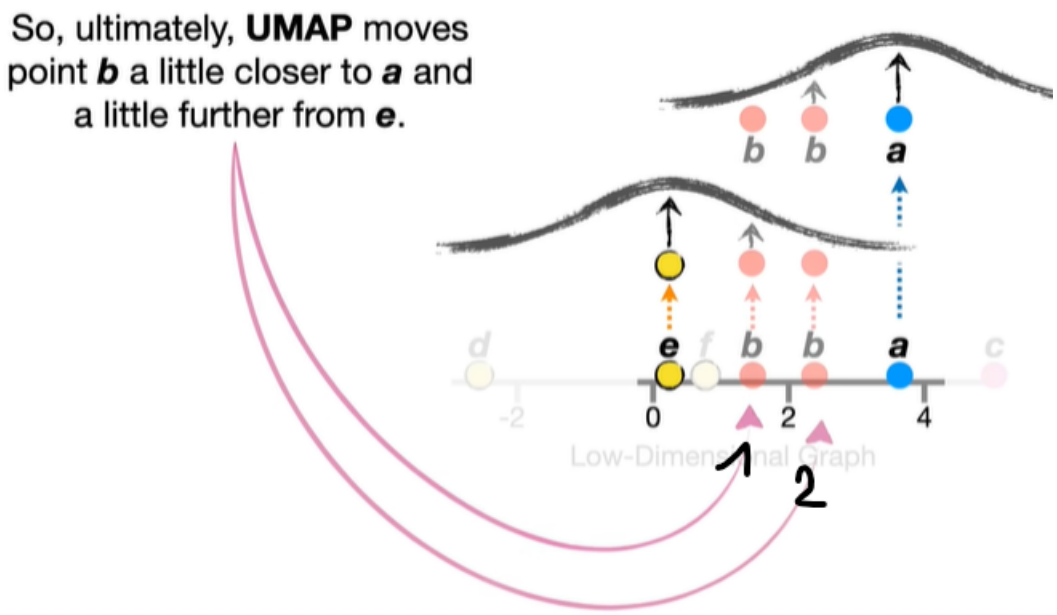

Then UMAP randomly takes a closer to b or b closer to a (50% probability for each decision).

Once it has decided which point to move closer to the other, it picks a point from which the point selected should move further: for example, in this example UMAP choses to move b closer to a and selects a point that b should move further from. It randomly selects one of the points out of B's high-dimensional cluster (D E or F in this example). Differently from before, here the similarity scores does not influence the probability of the choice, meaning that each point has an equal chance to be picked.

At this point, UMAP calculates how much the selected points need to be moved. To do so it calculates **low dimensional similarity scores** for each pair of points (the couple to be moved closer and the couple to me moved further, which in this case are a-b and b-e). Different from before, here the coordinates are the y coordinates of the points on a curve which is a t-distribution centered on the point to move (a and b)



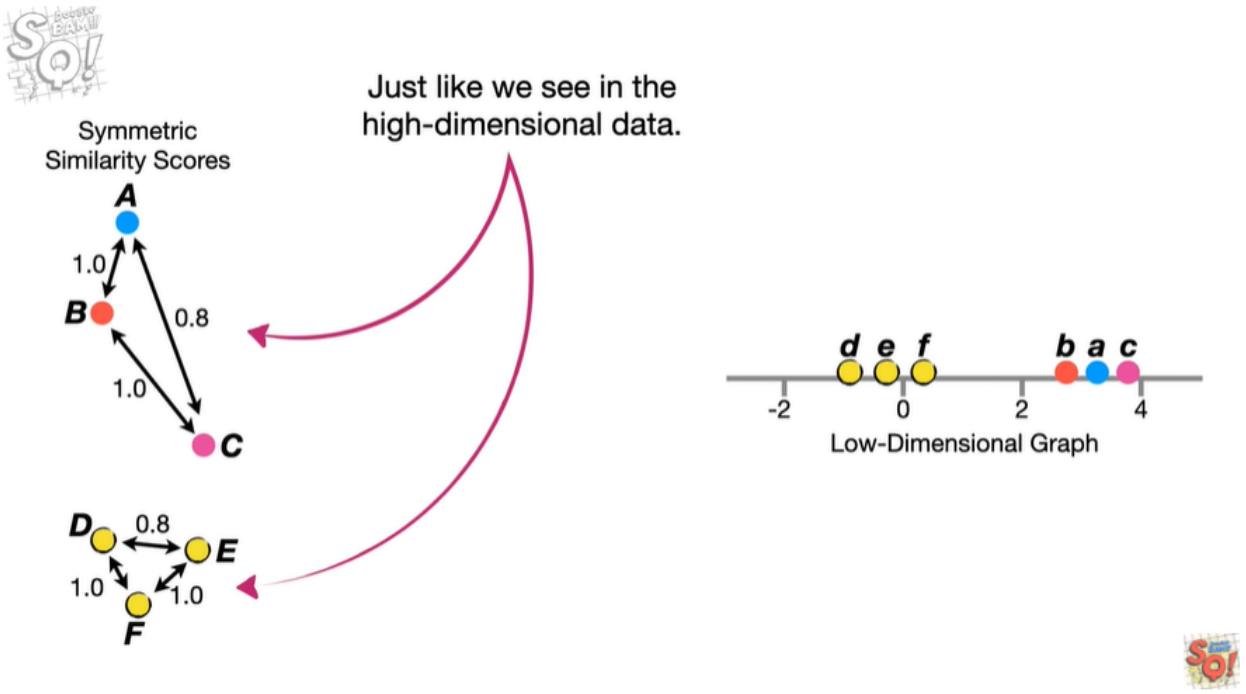
Now, because A and B are in the same cluster or neighborhood, UMAP wants to move b closer to a in order to maximize their low dimensional score. On the contrary, it wants to minimize the score of e and b which are in different clusters in the high-dimensional space.




From 1 to 2

Only a small movement is done: this is key with big data. If it made huge movements it would be more difficult to obtain a nice graph.

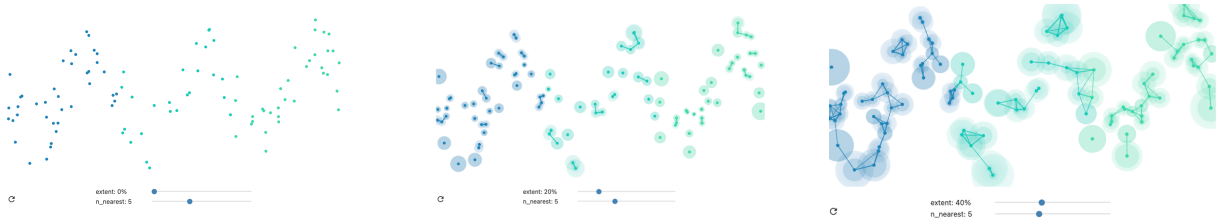
Then it takes another couple and does the same thing. It does this again and again until we have a low-dimensional graph representing correctly our data (here until we have 2 clusters that are relatively far from eachother).




**4 important differences between tSNE and UMAP:**

1. Speed: UMAP >> tSNE
2. tSNE always starts with a random initialization of the low-dimensional graph. Since it is random, every time you run tSNE on a dataset, you will start from a different low-dimensional graph of the data. Instead UMAP uses spectral embedding to initialize the low-dimensional graph which makes the initial low-dimensional graph always the same.
3. tSNE moves every single point a little bit each iteration, while UMAP can move just one point or a small subset of points each time. This helps when we are dealing with big datasets.
4. tSNE is more focused on details while UMAP is more focused on the big picture (actually depends from the n_neighbor parameter, see later).

Example of what happens in dimensional reduction with UMAP:



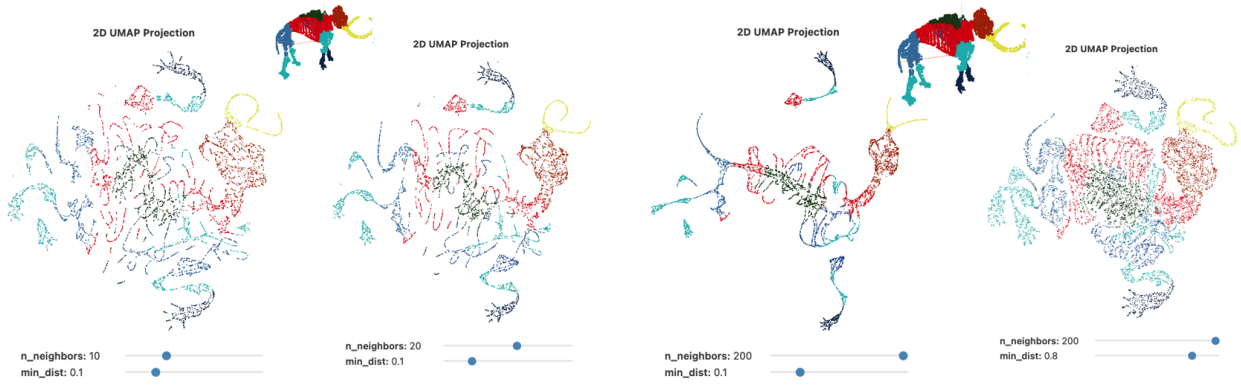
The extent parameter is related to how much I am extending the dots to make one point. Much more extension = much more connection between the dots.

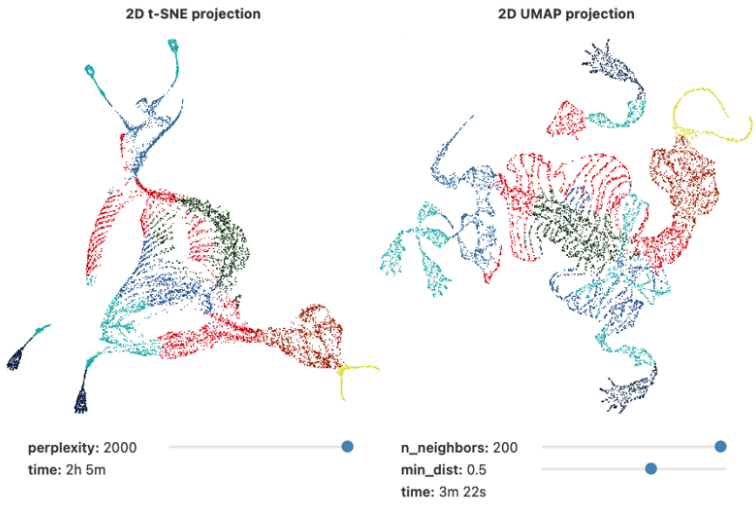
with tSNE distances are designed a priori in this case distances can be designed on the basis of the aggregation of point we can generate with this kind of approach. This help in focusing on the global picture and not on the local

On the basis of the distance you can make a connection. First you can extend to the 20% and some start to become connected, then you extend to 40% and you get more connection. The connectivity is used to make a plot and then a kind of software is used to do dimensional reduction.

✅ This approach helps us focusing on the big picture, even though we do not get to choose the distance measure we prefer as in tSNE.

> ✏️ Exercise: play with n_nerighbor and min_dist  and look how is changing.
>
> - A relatively low number of neighbors results in small independent clusters. Here you focus more on the details and you kind of lose the big picture
> - A relatively large value of the number of neighbors parameter will help us see the big picture without focusing on the details.



main parameter of tSNE but the main problem is the time needed for computing

> 📌 The **perplexity parameter** in tSNE is related to the amount of complexity that you want to visualize.

In a pdf on moodle there are all the commands to compare tSNE and UMAP for single cell RNAseq

> ❗ PCA and tSNE concepts are similar but used for different purposes: PCA is more a quality control while the tSNE is more of a mandatory step for the clustering.

# ✏️ Exercises about tSNE and UMAP

1. Install on your R docker image Rtsne and umap packages

   This is the command for Rtsne:

   ```
   install.packages("Rtsne")
   ```

2. Convert PC9 ctrl and Osimertinib treated cells in cpm.

   Start from the datasets of single cell experiments from moodle. Before doing the dimensionality reduction you have to convert the data in cpm. To do this see the function present in the pdf file on moodle (same concept of last lesson, you just start from another type of input file).

   The files the professor provided are saver imputed csv. Imputation aims to address the sparsity, i.e. the large number of zero observed in single-cell data. SAVER (single-cell analysis via expression recovery) is an expression recovery method for unique molecule index (UMI)-based scRNA-seq data borrowing information across genes and cells to provide accurate expression estimates for all genes.

   After this imputation procedure we have to convert the SAVER expression into $\log_2$cpm. To do this I modified the R script on the pdf file of the professor:

   ```
   # SAVER imputation from raw data (already done by the professor)
   #install.packages("SAVER")
   #library(SAVER)

   # If you work in R studio you have to specify the working directory
   # Be careful to use forward slashes even in windows paths
   #setwd("/somewhere/in_your_computer/path_to_data")
   #INPUT = "RNA-5c.csv"
   #SEPARATOR = ","
   #THREADS = 1 # the number of cores dedicated to run such analysis
   #setwd(WDraw.data <- read.table(INPUT, sep=SEPARATOR,
   #                       header = T, row.names=1)
   #dataset<- as.matrix(raw.data)
   #dataset.saver <- saver(dataset, ncores = THREADS,
   #                       estimates.only = TRUE)
   #write.table(dataset.saver, paste("saver", INPUT,
   #                                 sep="_"), sep=SEPARATOR, col.names=NA)

   # If you have 2 files (we have ctrl + osi) you have to copy this code and run it
   # on the other file.
   # Since both are already SAVER imputed I omitted this part for clarity.
   ```

```
# Start from a SAVER imputed .csv file
# Create the function counts2cpm
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_ctrl.csv", sep=",")
file.rename(from="log2cpm.csv", to="ctrl_log2cpm.csv")

# Recall again the function and run it on your osi file
counts2cpm(file="saver_osi.csv", sep=",")
file.rename(from="log2cpm.csv", to="osi_log2cpm.csv")
```

This is what I (Francesca) did (I tried it on Rstudio only):

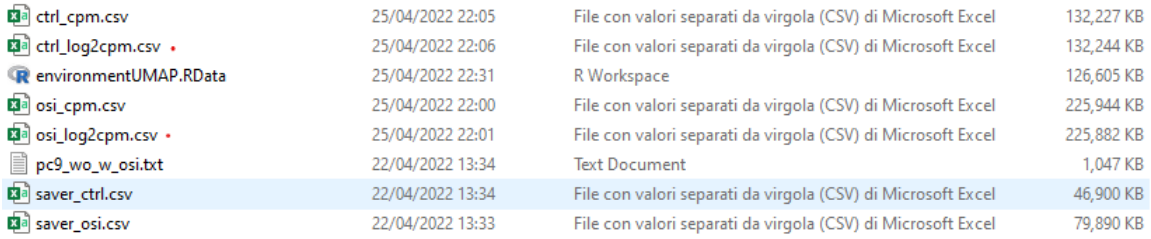
a. I copied the counts2cpm function into a new Rstudio script and used “source” to put it in the environment

b. I chose the working directory going in “Sessions → Set Working Directory → Choose Directory”. The directory chosen MUST contain the saver imputed .csv files
You can do this by line command with the following command:

```
setwd ("path/directory") #for example setwd("C:/Users/user/Desktop/DataAnalysis")
```

c. I run the counts2cpm function on both the saver_osi and the saver_ctrl files

d. I renamed the log2cpm output files as follows (I did it as you normally do from the folder, you can use the line command file.rename if you want)




e. At the end you will only need the log2cpm files (the ones with the red dot). The _cpm.csv and the saver.csv won’t be necessary for this exercise.

3. Run UMAP and Rtsne changing the parameter indicated in the tsne_umap tutorial provided to you: umap min_dist and tSne perplexity

   If the treatment is done in vivo (e.g. scRNAseq experiment were done in vivo), the changes are less dramatic than what you see in a plate.

*How can you maximize the differences between the untreated and the treated?* Play with the UMAP min-dist and n-neighbor parameters or tSNE perplexity parameter.

a. Install the umap and ggplot2 packages

b. Use this command lines to run UMAP.
‼ **BE CAREFUL TO PUT THE RIGHT FILE NAMES IN THE FUNCTION**. DO NOT COPY & PASTE

```
library(umap)
library(ggplot2)
#CTRL samples
ctrl <- read.table("ctrl_log2cpm.csv", sep=",", header=T, row.names=1)
ctrl.umap <- umap(t(ctrl), random_state=111, n_epochs = 1000)
f=data.frame(x=as.numeric(ctrl.umap$layout[,1]),y=as.numeric(ctrl.umap$
                                                              layout[,2]))
#plotting UMAP
sp <- ggplot(f, aes(x=x,y=y)) + geom_point(pch=19, cex=0.3)
pdf("ctrlUMAP.pdf")
print(sp)
dev.off()

#OSI samples
osi <- read.table("osi_log2cpm.csv", sep=",", header=T, row.names=1)
osi.umap <- umap(t(osi), random_state=111, n_epochs = 1000)
f=data.frame(x=as.numeric(osi.umap$layout[,1]),y=as.numeric(osi.umap$
                                                              layout[,2]))
#plotting UMAP
sp <- ggplot(f, aes(x=x,y=y)) + geom_point(pch=19, cex=0.3)

pdf("osiUMAP.pdf")
print(sp)
dev.off()
```

4. To run tSNE:

```
library(Rtsne)
library(ggplot2)
#CTRL samples
tmp <- read.table("ctrl_log2cpm.csv", sep=",", header=T, row.names=1)
tsne.out <- Rtsne(as.matrix(t(tmp)), pca= FALSE, perplexity = 30, theta=0.0)
f=data.frame (x= as.numeric(tsne.out$Y[,1]),y=tsne.out$Y[,2])
#plotting the tsne
sp <- ggplot(f, aes(x=x,y=y)) + geom_point(pch=19, cex=0.3)
pdf("ctrl_tsne_noPCA.pdf")
print(sp)
dev.off()

#OSI samples
tmp <- read.table("osi_log2cpm.csv", sep=",", header=T, row.names=1)
tsne.out <- Rtsne(as.matrix(t(tmp)), pca= FALSE, perplexity = 30, theta=0.0)
f=data.frame (x= as.numeric(tsne.out$Y[,1]),y=tsne.out$Y[,2])
#plotting the tsne
sp <- ggplot(f, aes(x=x,y=y)) + geom_point(pch=19, cex=0.3)
pdf("osi_tsne_noPCA.pdf")
print(sp)
dev.off()
```