While bulk RNA-seq tries to highlight changes between different conditions, scRNA-seq shows what is changing within a population of cells in a sample. The dataset we used for the previous lesson (PC9 cells treated with EGFRi) came out from a bulk RNA-seq analysis: ctrl (DMSO) sample, acute treatment sample, chronic treatment sample. Different situation in the case of scRNA-seq experiment: PC9 cells are injected in control mice (treated with DMSO) and treated mice (Osim for 24h). There are different biological questions in the two kinds of analysis:

- Bulk RNA-seq → *are acute and chronic treatments changing the expression of the same or different genes?*
- scRNA-seq → *after the treatment, are we going to observe the same number of cells of specific cellular subpopulations or the number of cells in the different subpopulations changes?*

And *what if we perform clustering in the scRNA-seq analysis based on those genes that characterize the acute treatment in the bulk RNA-seq experiment?* We can answer all these questions since we have the data of both bulk and scRNA-seq analyses. The actual biological question is always the same in all cases: ***which genes are involved in drug resistance?***

BULK RNA-SEQ AND PCA

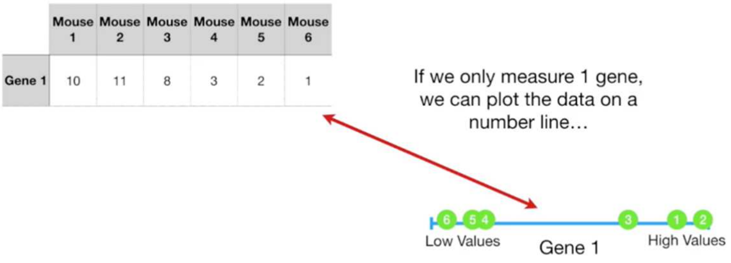
PCA on bulk RNA-seq enables to evaluate the expression profile of different conditions. It also provides a way to represent multidimensional data in a reduced-dimension space. PCA can be done also on scRNA-seq data but usually other more complex tools are used instead.

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

PCA step-by-step:

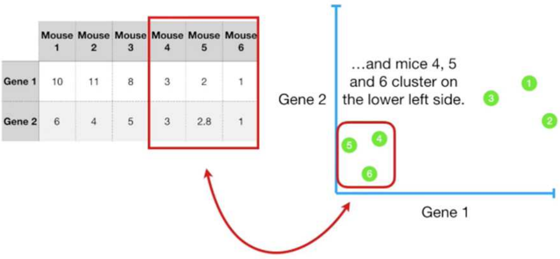
Transcription values of two genes in 6 different mice (let's consider them as individual samples). With PCA we'll pass from a high-dimensional dataset to a 2D plot. PCA takes 4+ dimensions (genes) and plots them in a 2D space, as it only shows the principal components (the most variable dimensions).

Last time we got weights for genes' expression levels. Let's consider a simple situation, where only Gene 1 is considered: these data could lie only on a line (1D) and mice 1, 2, 3 would be close to each other (as their Gene 1 expression values are relatively high), while mice 4, 5, 6 are located further.



	Mouse 1	Mouse 2	Mouse 3	Mouse 4	Mouse 5	Mouse 6
Gene 1	10	11	8	3	2	1

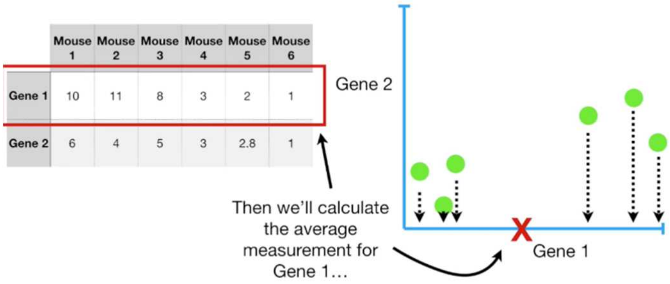
If we only measure 1 gene, we can plot the data on a number line…



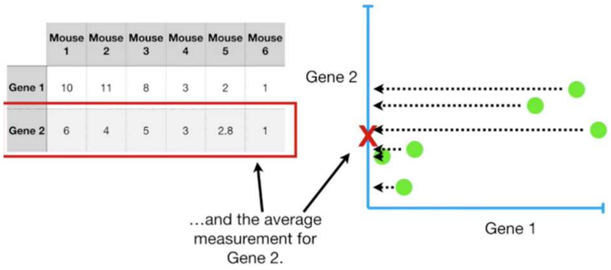
	Mouse 1	Mouse 2	Mouse 3	Mouse 4	Mouse 5	Mouse 6
Gene 1	10	11	8	3	2	1
Gene 2	6	4	5	3	2.8	1

…and mice 4, 5 and 6 cluster on the lower left side.

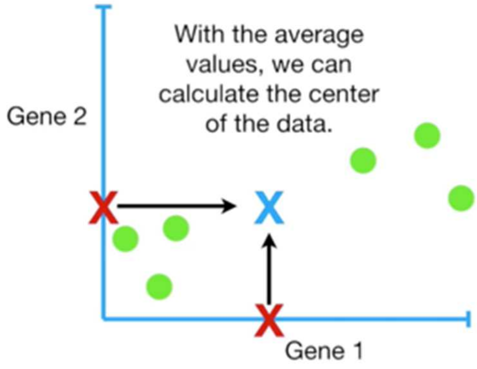
Let's now consider a dataset with 2 genes. We'll first plot data on the X axis or Y axis and then we retrieve the average values for both axes (i.e. genes) to calculate the centre (origin) of axes/data in the actual PCA plot.



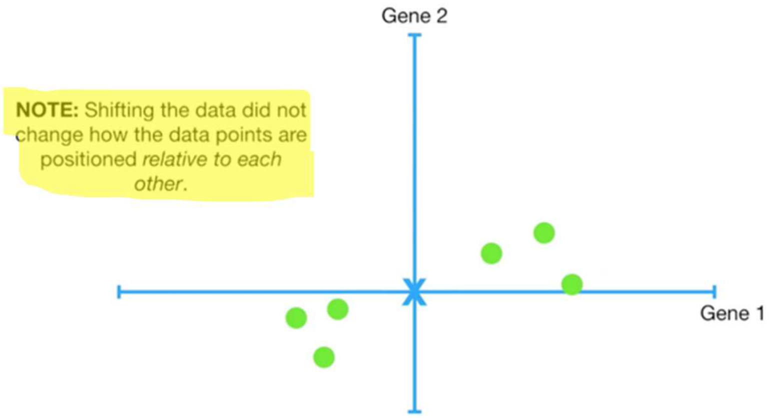
	Mouse 1	Mouse 2	Mouse 3	Mouse 4	Mouse 5	Mouse 6
Gene 1	10	11	8	3	2	1
Gene 2	6	4	5	3	2.8	1



	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



With the average values, we can calculate the center of the data.
Gene 2
X
X
X
Gene 1

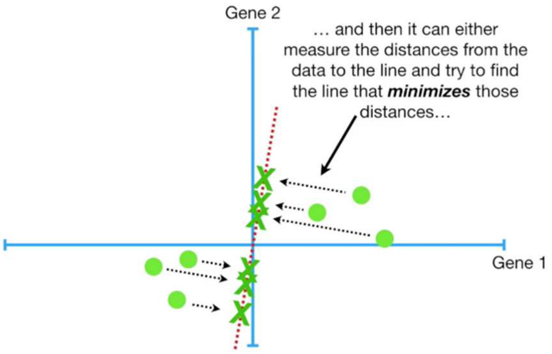


NOTE: Shifting the data did not change how the data points are positioned relative to each other.
Gene 2
Gene 1

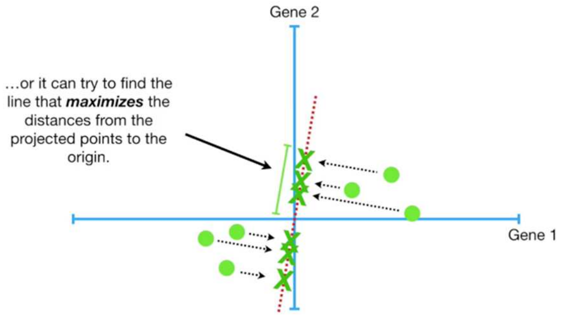
Now we are representing data in reference to the mean expression values.

Now that data are centred to this new origin, let's draw a random line that passes through the origin. This line then should fit our data in the best possible way and for this reason we start rotating it.

To check whether the fitting is actually good, PCA projects data onto this random line and then it can either try to (1) find the line that **minimizes** the distances **from the data** (points) **to the line**:



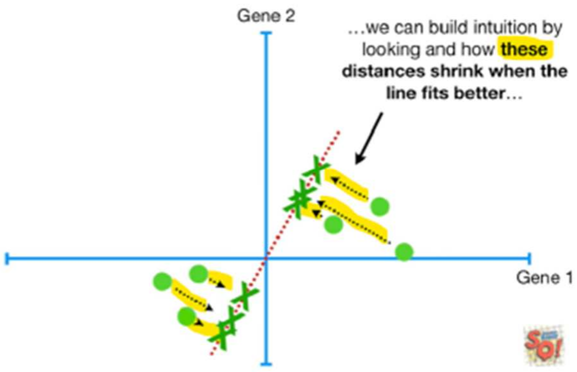
Gene 2
… and then it can either measure the distances from the data to the line and try to find the line that minimizes those distances…
Gene 1

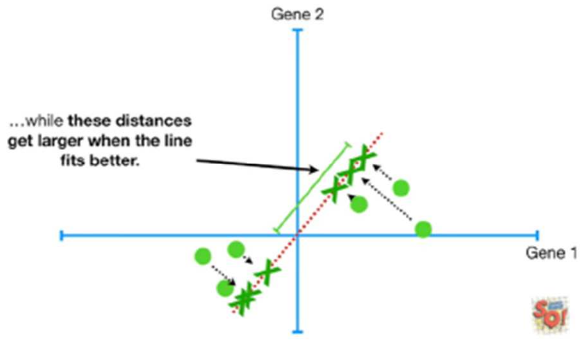


Gene 2
…or it can try to find the line that maximizes the distances from the projected points to the origin.
Gene 1

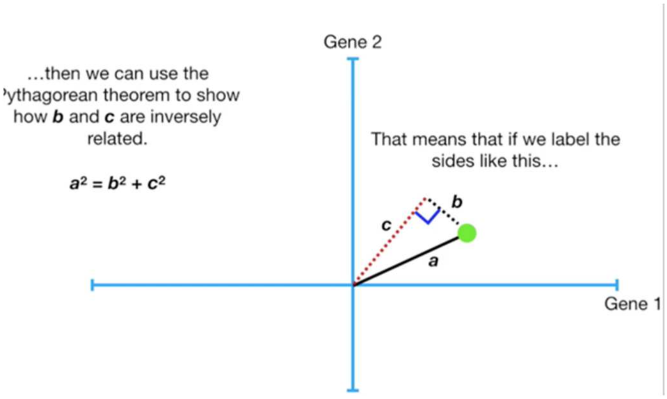
Or PCA can try to find (2) the line that **maximizes** the distances **from the <u>projected</u> points to the origin**:

These options are equivalent. Intuitively, when the line fits better, the distance from the line to the points is minimized while with the distance from the points to the origin is maximised:


Gene 2
…we can build intuition by looking and how these distances shrink when the line fits better…
Gene 1

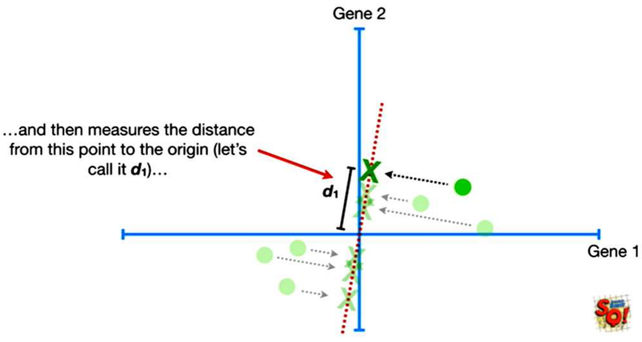

Gene 2
…while these distances get larger when the line fits better.
Gene 1

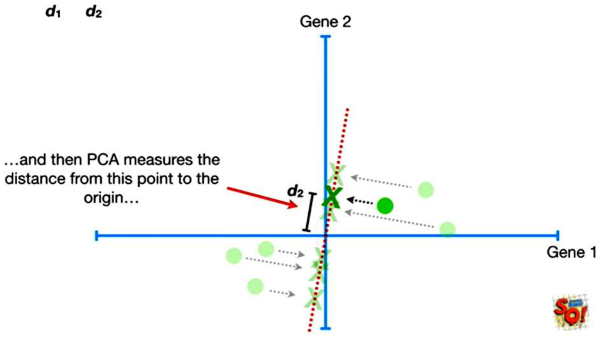
Let's call the distance from a point to the line ***b***, and the distance from the projected point to the origin ***c***. For the Pythagorean theroem, we'll show that *b* and *c* are inversely correlated: $a^2 = b^2 + c^2$. If *b* gets bigger, *c* gets smaller, and vice versa → PCA can either minimize the distance from data to the line, or maximize the distance from the projected data to the origin. In the practice, **PCA fins the best fitting line by <u>maximising</u> the sum of the squared distances from the projected points to** a^2 **the origin** (**c^2**), because it's easier to calculate the distance from the projected points to the origin.


Gene 2
…then we can use the Pythagorean theorem to show how b and c are inversely related.
a² = b² + c²
That means that if we label the sides like this…
b
c
a
Gene 1

PCA steps:

1. It draws a random line passing through the origin
2. It projects data onto this line
3. It measures the distances from this projected points to the origin (*c*) and calls it *d* ($d1^2$, $d2^2$, $d3^2$, … for every point).

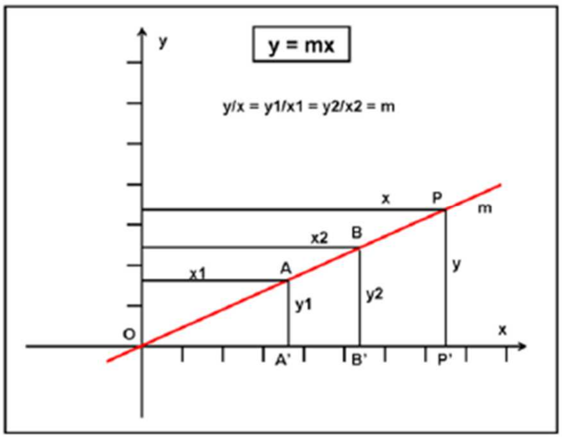

Gene 2
…and then measures the distance from this point to the origin (let's call it d₁)…
d₁
Gene 1

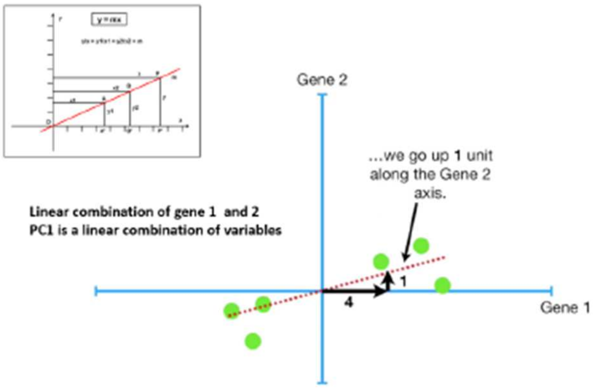

d₁ d₂
Gene 2
…and then PCA measures the distance from this point to the origin…
d₂
Gene 1

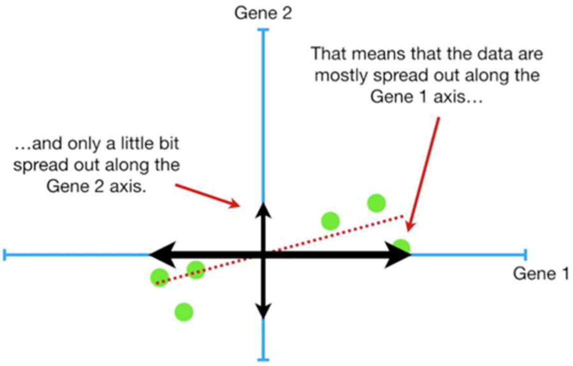
Being squared distances, the negative values won't cancel out the positive ones (i.e. above/below and left/right to the origin of axes).

4. Finally, PCA calculates the **sum of the squared distances** (**SS**) as the sum of all these d^2 values.

<u>These 4 steps are repeated until the PCA ends up with the line with the largest SS value</u>. This line will be the **principal component 1** (**PC1**) and will be characterized by a slope value (e.g., if slope = 0.25, for every 4 units we move along Gene 1 we'll move of just 1 unit along Gene 2, meaning that data are mostly spread out along Gene 1 axis:

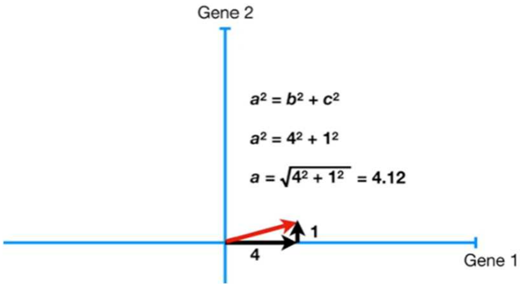


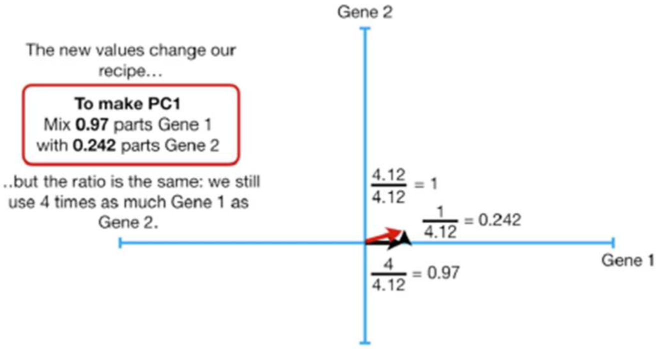

Gene 2
...we go up 1 unit along the Gene 2 axis.
Linear combination of gene 1 and 2
PC1 is a linear combination of variables
1
4
Gene 1


Gene 2
That 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.
Gene 1

To demonstrate that data are mostly spread out along Gene 1 axis, we can do as it follows.

The PC1 component is made as a cocktail: you mix 4 parts of Gene 1 to 1 part of Gene 2. This tells you that Gene 1 is more important than Gene 2 in terms of contribution to the "cocktail" PC1. The recipe for PC1, when doing PCA with SVD, is scaled so that the red line length is equal to 1:



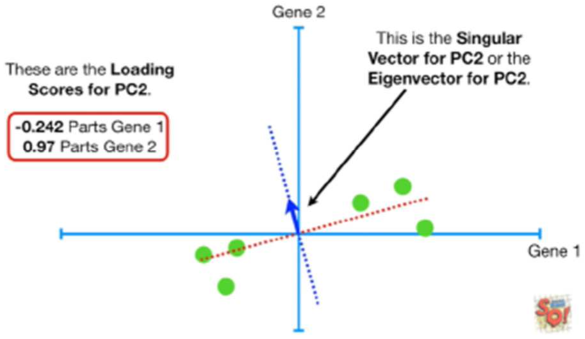


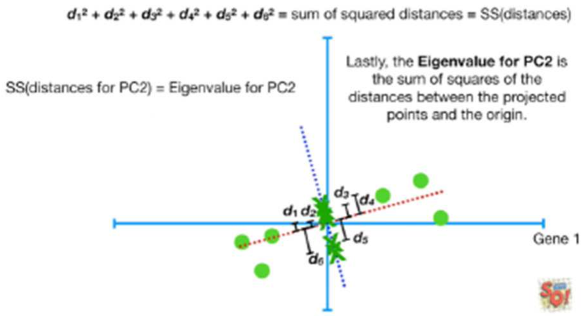
So you divide everything to 4.12. Even by doing this, the contribution of each Gene to the PC1 remains the same, as 4 : 1 = 0.97 : 0.242.

This 1 unit-long vector (red line), which consists of 0.97 parts Gene 1 and 0.242 parts Gene 2, is called *eigenvector* (or Singular Vector) for PC1.

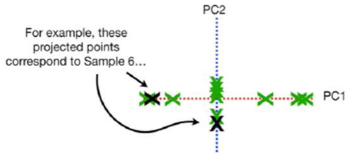
PCA calls the SS for the best fit line the **eigenvalue for PC1**. The square root of the eigenvalue for PC1 is called the **singular value for PC1**.

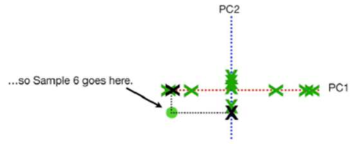
In a 2D plot as this one, the PC2 will simply be the line through the origin perpendicular to PC1**.** No further optimization needed. We do the same as before (SS calculations, PC recipe, scaling) and we obtain the **Singular Vector** or **eigenvector** for PC2. The components of this vector (-0.242 and 0.97) are the **loading scores for PC2**. The **eigenvalue for PC2** is the sum of all the SS.

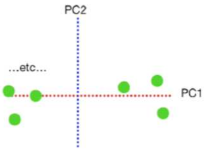

Gene 2
This is the Singular Vector for PC2 or the Eigenvector for PC2.
These are the Loading Scores for PC2.
-0.242 Parts Gene 1
0.97 Parts Gene 2
Gene 1


$d_1^2 + d_2^2 + d_3^2 + d_4^2 + d_5^2 + d_6^2$ = sum of squared distances = SS(distances)
SS(distances for PC2) = Eigenvalue for PC2
Lastly, the Eigenvalue for PC2 is the sum of squares of the distances between the projected points and the origin.
d_1
d_2
d_3
d_4
d_5
d_6
Gene 1

We can now draw the final PCA plot, as we have both PC1 and PC2 that will become the new axes (we rotate everything in such a way that PC1 is horizontal). Then we use projected points to find there the samples do localize in the plot:


PC2
For example, these projected points correspond to Sample 6...
PC1



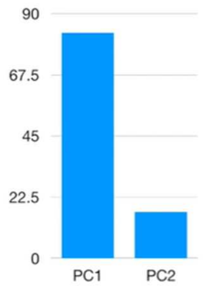


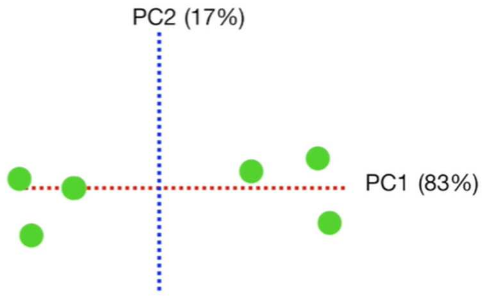
The eigenvalues (SS) for each component can be converted into variations around the origin (0, 0) in this way:

$$\frac{SS(distances\ for\ PC1)}{n-1} = variation\ for\ \boldsymbol{PC1}$$

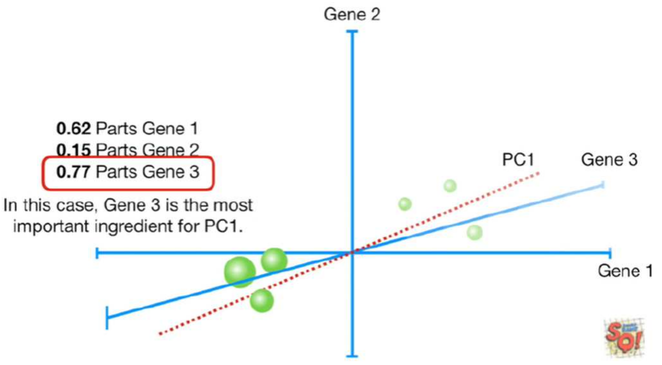
$$\frac{SS(distances\ for\ PC2)}{n-1} = variation\ for\ \boldsymbol{PC2}$$

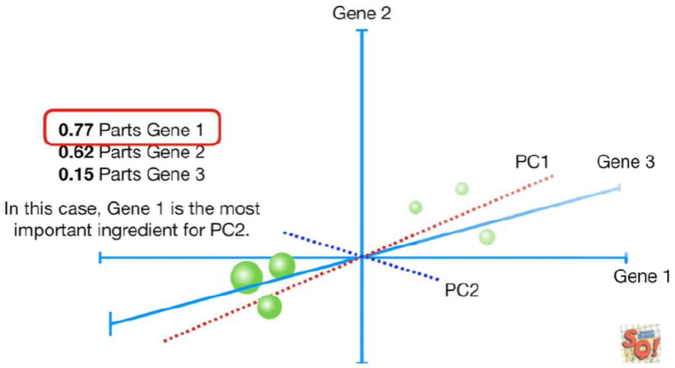
Where *n* = sample size. If variation for PC1 = 15 and variation for PC2 = 3, the total variation around the origin is 15 + 3 = 18, where PC1 accounts for 15/18 = 0.83 = 83% of the total variation. We use the **Scree plot** to graphically represent the percentages of variation for each PC:

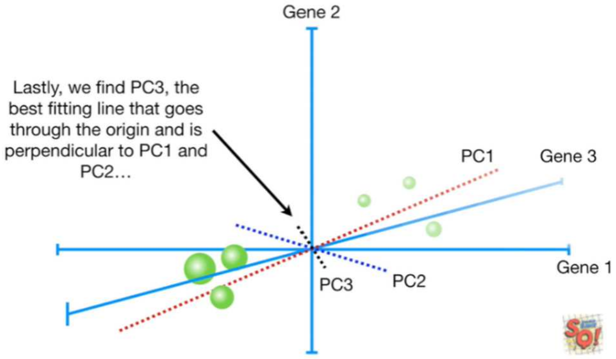




Let's now dive into a more complicated case: a PCA with 3 variables/dimensions (genes). The recipes for each PC component will now have 3 ingredients instead of 2, but the overall procedure is pretty the same also in this case.

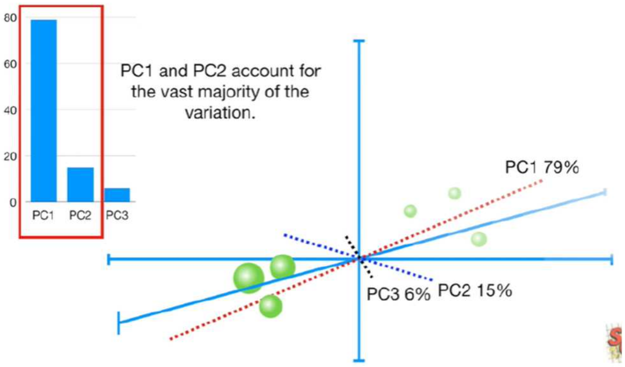




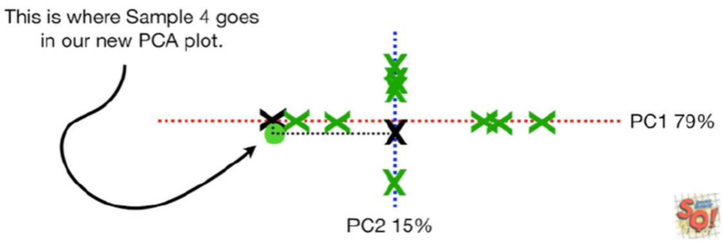

Gene 2
Lastly, we find PC3, the best fitting line that goes through the origin and is perpendicular to PC1 and PC2…
PC1
Gene 3
PC3
PC2
Gene 1

We go like this for every gene added, finding more and more PCs by adding perpendicular lines and rotating. In theory there is 1 PC per gene, but in practice the number of PCs is either the number of variables (genes in this case) or the number of samples: the smallest of the two basically.

Once you've got all the PCs figured out, you use the eigenvalues (SS) for each PC to evaluate the variation around the origin (Scree plot) →



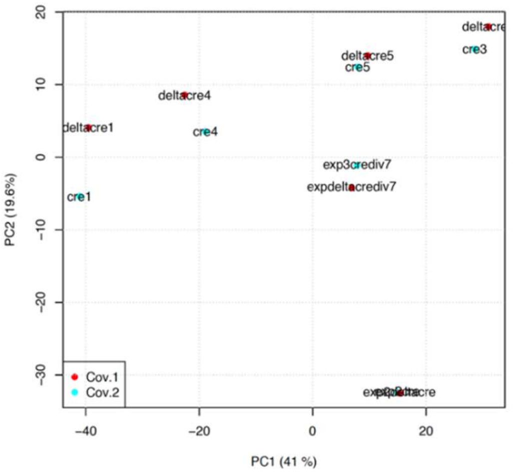
Looking at this Scree plot it emerges that PC1 and PC2 account for the most variation and thus we could use a 2D graph considering only these two components instead of using/having a 3D plot with also Gene 3. To do so, only PC1 and PC2 data points are kept and projected onto PC1 and PC2, then rotation to have PC1 horizontal and PC2 vertical. Now the graph is more intuitive and still informative:



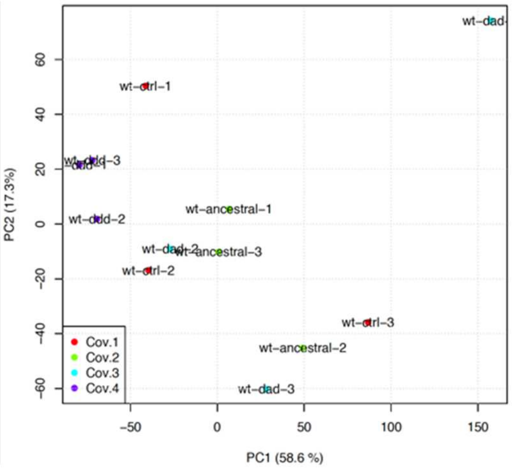
If instead in the Scree plot also Gene 3 (and maybe other genes as well) accounted for a significant variation, this could not have been done because data wouldn't be represented that accurately.

PCA is a linear dimensionality reduction score: the variance has a linear dispersion, we draw a line (not a curve) to fit our data. The different PCs are independent as they're based on the expression of different genes. The criticality in PCA is defining the first PC, which accounts for most of the variance (the second PC explains the 50% of the variance). Later components are less significative in terms of variance instead.

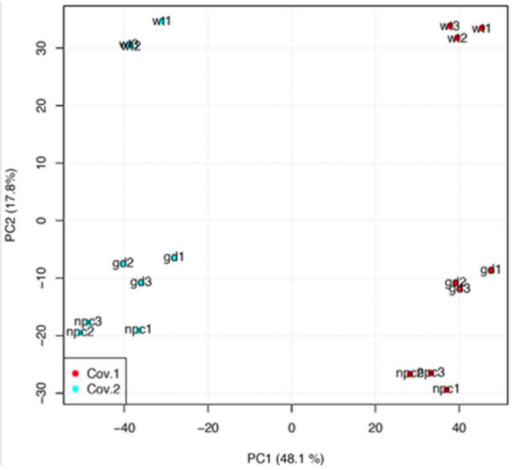
EXAMPLES OF PCA



1) 6 KO mice *vs* 6 WT mouse. The differences that can be attributed to the KO (red dots?) lie on the PC2. We can see that the readout of the experiment strongly depends on the KO efficiency. Also, there are differences between individuals that could mask the actual differential expression between KO and WT individuals. To cancel out these differences first, we perform the **block experiment**. Then we can focus on the actual difference between treated/untreated.



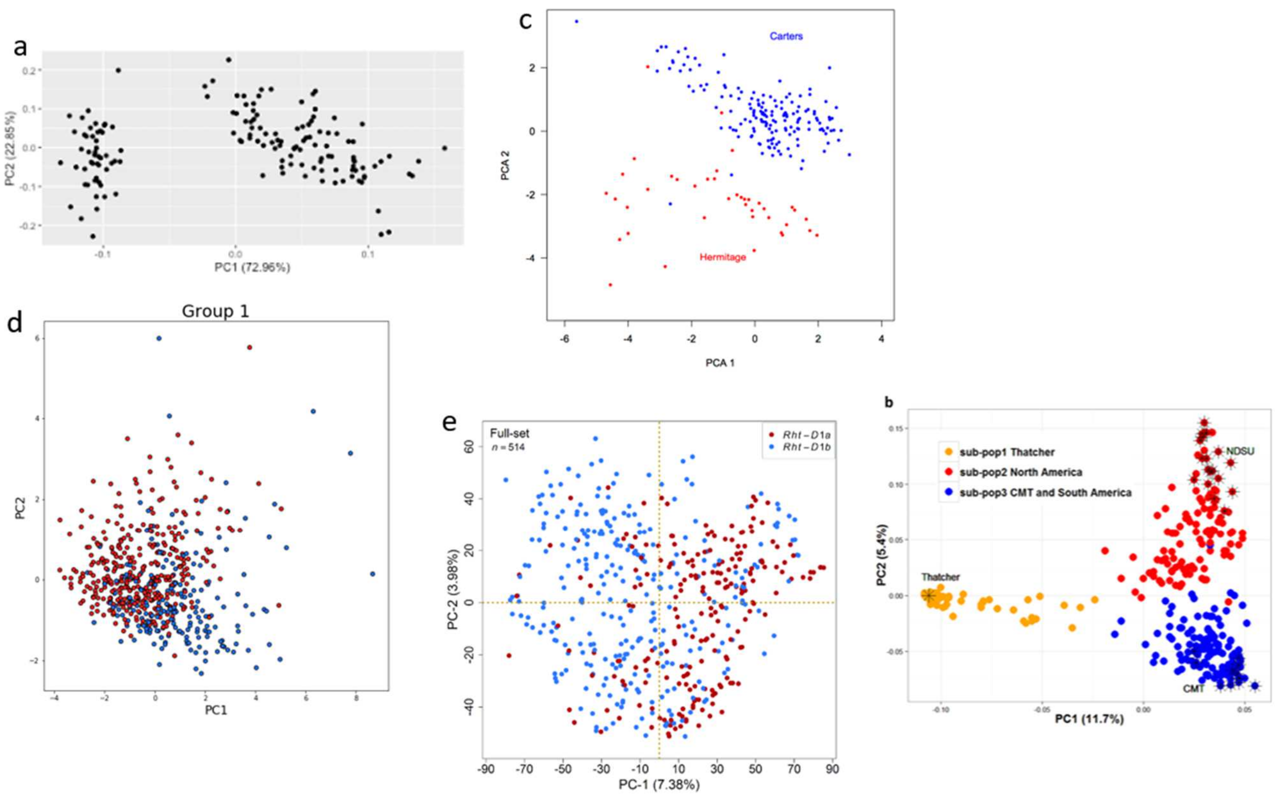
2) Same cell line, 4 different conditions. In this case we're not able to identify a clear separation between different conditions: the problem now lies in the analysis. A problem could be that the KD was done by transfection instead of lentiviral infection, and the former is less efficient than the latter: the differences in transfection efficiency in the different conditions are bigger than any other possible difference we could be interested in.



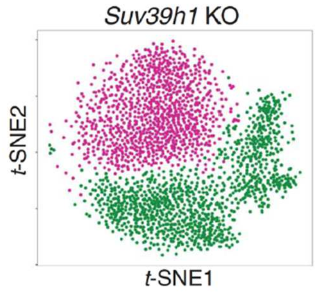
3) 2 cell lines (1, 2) and 2 different conditions (WT, T: control and treated). The PC1 allows to discriminate between the two cell lines, while the PC2 gives info about the differences between control and treated samples within the same cell line (and this info is less informative than that given by PC1, as the variation % suggest). With these data you could do:
   a. Intersection between those genes differentially expressed in both cell lines, in the same conditions (e.g., WT1 *vs* WT2, or T1 *vs* T2).
   b. Intersection between genes differentially expressed in both cell lines, but in different conditions (e.g., WT1 *vs* T1, or WT2 *vs* T2). In this case you'll identify differentially expressed genes that are specific for the cell line.

In the figure below there are some examples of PCA plots:

- In the **a)** plot, PC1 is the most informative dimension as it is the one differentiating the most the data points. Good separation along PC1.
- In the plot **c)** the separation is mainly achieved along PC2.
- In **d)** there is too much noise to find something different between the two groups (due to not enough transcriptional difference, or problems during the analysis). Too much overlapping of elements.
- In **b)** the most differentiated condition/sample is the yellow one, as it is well separated from the other two considering both PC1 and PC2 (while red and blue can be only distinguished based on PC2).
- In **e)** there's a lot of noise, but red and blue localize on two different sides at least, so some info can still be retrieved.

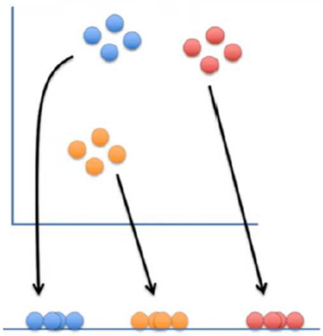


SINGLE-CELL RNA-SEQ AND tSNE



scRNA-seq experiments are noisier and more genes display expression level = 0 (lots of 0 for this reason). PCA is not sufficient to make a separation, because in scRNA-seq there are too complex datasets and the first two PCs do not account for the most variation in the data. We need a **nonlinear dimensional reduction method** (still dimensional reduction, but relationship with the original data is no longer clear). In this case the separation is no longer achieved with a line but with a curve (nonlinear). There are two main methods like this: **tSNE** and **UMAP**. In this example of tSNE plot, 3 groups can be distinguished from their transcriptomic profile:

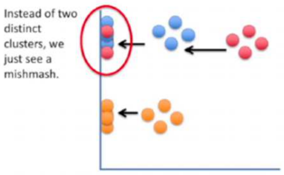
tSNE step-by-step:

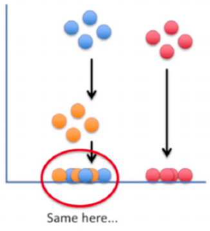


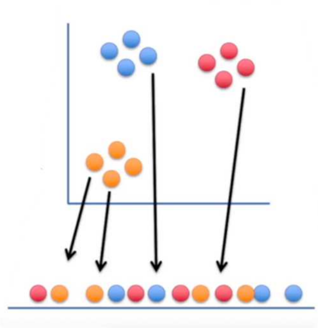
tSNE is able to project data into a low dimensional space (in this case, a 1D line) so that the clustering of the high dimensional space (here the 2D scatter plot) is preserved.

Let's consider a very simple dataset. From this 2D scatter plot, tSNE takes the objects (data) and puts them on a 1D plot (right figure).

However, by sampling projecting points on a single axis, clustering wouldn't be preserved (figures below). How does tSNE preserve the separation then?

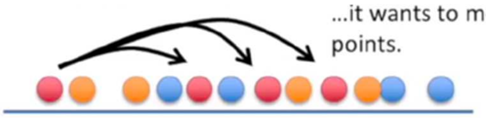






1. At first, the position of points in the low dimensional space is random. tSNE will then progressively move points until the final clustering is obtained. Therefore, the starting situation is this (on the right):

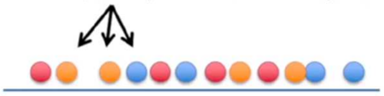
2. Let's now start clustering. How to move the first point (red one)? To the right or to the left? Since it is part of a cluster on the right, this point will want to move closer to the other red points that are located more towards the right:

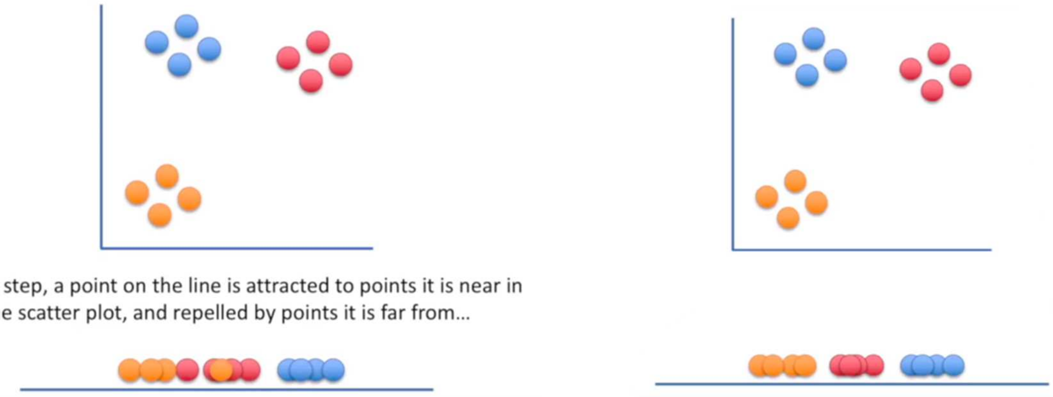


...it wants to move closer to these points.

3. Therefore the red point aims at being closer to its red companions and further from yellow and blue ones, and the yellow and blue point will tend to the same situation with their own companions. This is done progressively and will result in the final clustering.

But at the same time, these points... ...are far away in the scatter plot...



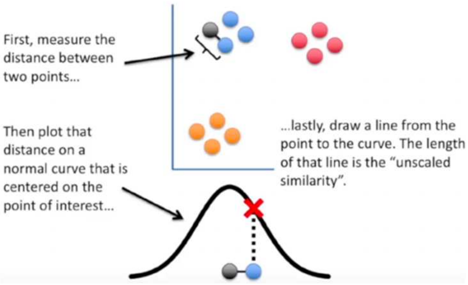


At each step, a point on the line is attracted to points it is near in the scatter plot, and repelled by points it is far from...

This will be the final result

So, the idea is this, but *how does it actually work?*

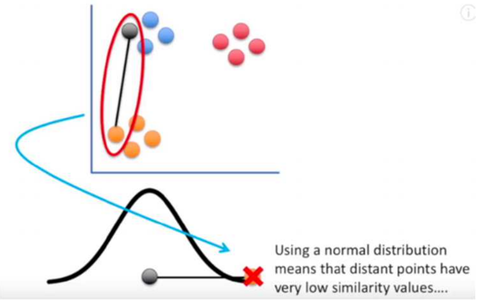
- We consider a point (e.g., a cell of the experiment) in the real space and we measure its distance from another point. We draw a normal distribution curve centred on this point and we plot on this curve the distance from the other point. We draw a line from this other point to the curve: the **unscaled similarity**. We do this between the point of interest and each point in the space (1 couple at a time basically).



First, measure the distance between two points...

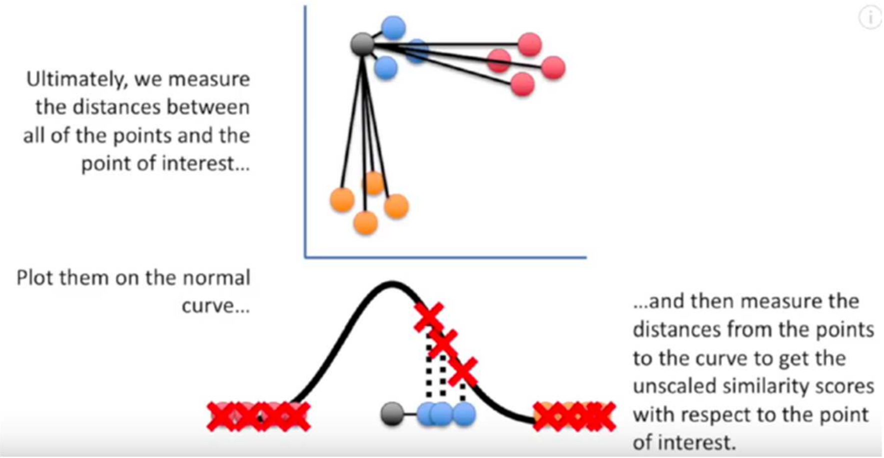
Then plot that distance on a normal curve that is centered on the point of interest...

...lastly, draw a line from the point to the curve. The length of that line is the "unscaled similarity".



Using a normal distribution means that distant points have very low similarity values....

We use a normal distribution curve because in that way distant point will have very low similarity values (shorter lines that link the point to the curve), while closer points have them higher:

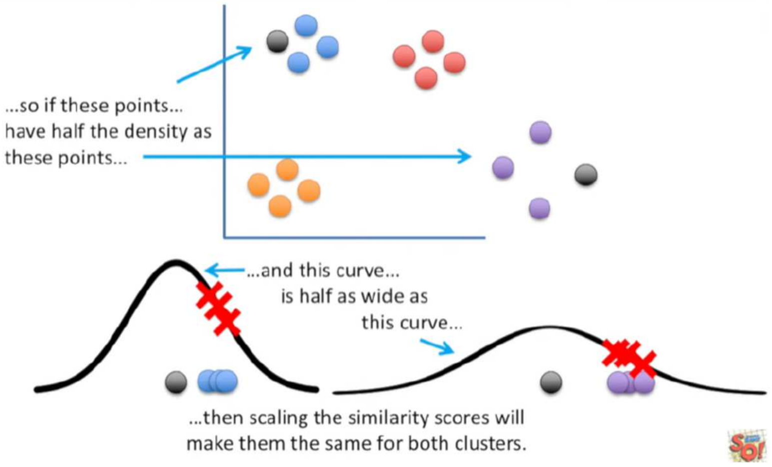


Ultimately, we measure the distances between all of the points and the point of interest...

Plot them on the normal curve...

...and then measure the distances from the points to the curve to get the unscaled similarity scores with respect to the point of interest.

- Now we scale the unscaled similarity values so that their sum is equal to 1 (sort of normalization of the distance, to represent distances somehow). We do this because we can't compare different distributions. *What does this mean?* Let's consider another cluster (in violet), which has half of the point density compared to the other clusters: this displays a wider normal curve (lower density = wider curve). So, we can't compare the blue cluster's curve and the violet's. **Scaling the similarity values is necessary to make the normal distributions the same** for all the clusters (it is a normalization procedure):

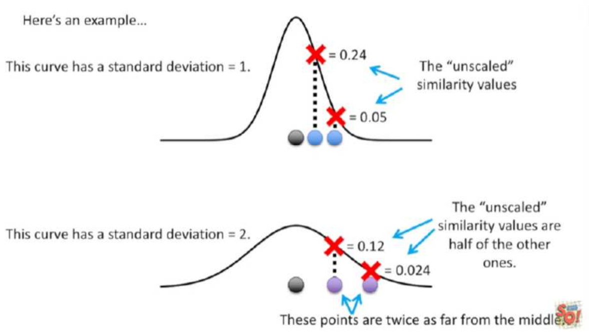


...so if these points... have half the density as these points...

...and this curve... is half as wide as this curve...

...then scaling the similarity scores will make them the same for both clusters.

- From this step we obtain **scaled similarity scores**, which are obtained using the formula below:



Here's an example...

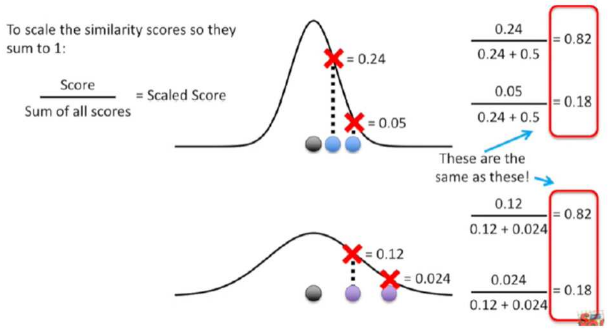
This curve has a standard deviation = 1.

The "unscaled" similarity values

This curve has a standard deviation = 2.

The "unscaled" similarity values are half of the other ones.

These points are twice as far from the middle.



To scale the similarity scores so they sum to 1:

$$\frac{\text{Score}}{\text{Sum of all scores}} = \text{Scaled Score}$$

$$\frac{0.24}{0.24 + 0.5} = 0.82$$

$$\frac{0.05}{0.24 + 0.5} = 0.18$$

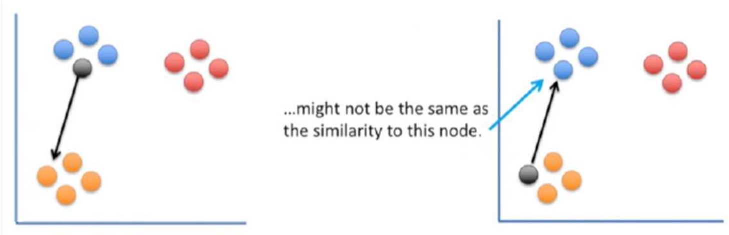
These are the same as these!

$$\frac{0.12}{0.12 + 0.024} = 0.82$$

$$\frac{0.024}{0.12 + 0.024} = 0.18$$

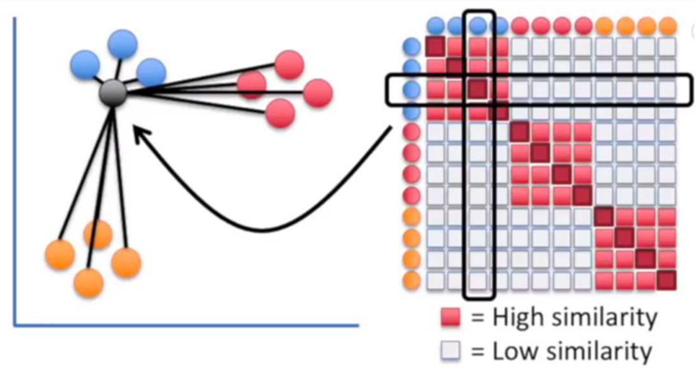
While the unscaled similarity scores are different, the scaled ones are the same. If you sum the scaled scores together, you'll obtain 1, so you can compare different clusters (that have different density of points, i.e. the man distance between 2 points is different).

Because the width of the normal distribution of a point is based on the density of the surrounding points, the similarity score between two points may differ depending on the directionality considered:



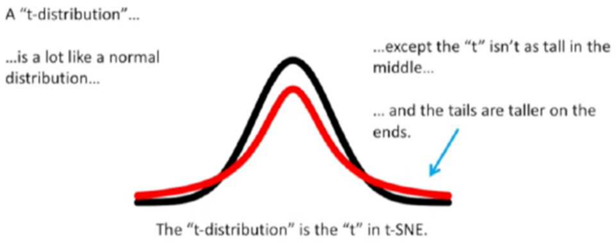
That's why tSNE calculates both scores and makes an average between the two similarity scores to tell the actual score associated to a couple of data points.

- Finally, you end up with a **squared similarity matrix**, which is the result of the calculation of similarity scores for all points in the space.

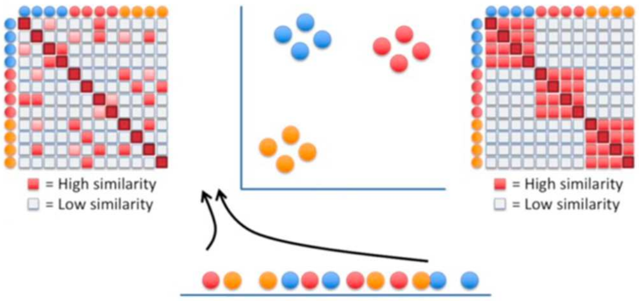


This matrix tells differences between points. The diagonal line links the similarity scores of a point with respect to the same point (100% similarity). However, for the clustering purpose this is not useful and tSNE sets it as 0.

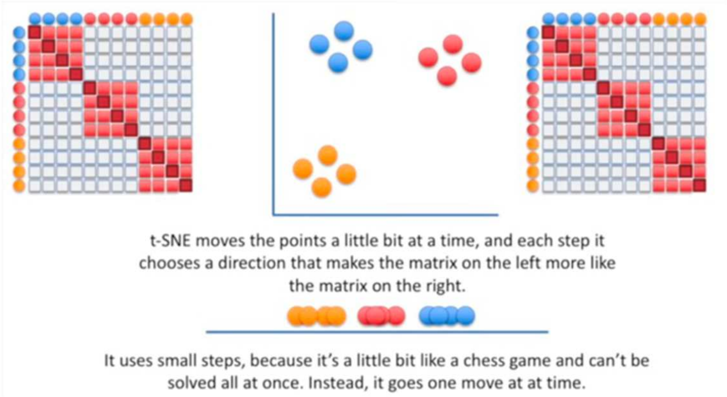
- At this point, we randomly project the data onto the number line and the same procedure (similarity score calculation) is repeated in this dimensionally reduced space (the 1D line). However, in this case we use a **t-distribution**. This distribution is narrower than the normal distribution. We use this distribution because it is easier to keep near to each other in the dimensionally reduced space those data that were similar in the high dimension space. The t-distribution confers the "t" to tSNE.


A "t-distribution"...
...is a lot like a normal distribution...
...except the "t" isn't as tall in the middle...
... and the tails are taller on the ends.
The "t-distribution" is the "t" in t-SNE.

- We calculate again the unscaled similarity scores for all the couple of points on the line, then we scale them as we did before. We'll end up with a matrix again, but this matrix is a mess compared to the previous one (the original space matrix):



- To solve this problem, tSNE takes one point at a time and moves it in order to reach the original situation (of the original matrix). Every time it moves a point, it calculates again the distances and the matrix changes (until it becomes equal/comparable to the original one):



t-SNE moves the points a little bit at a time, and each step it chooses a direction that makes the matrix on the left more like the matrix on the right.

It uses small steps, because it's a little bit like a chess game and can't be solved all at once. Instead, it goes one move at at time.

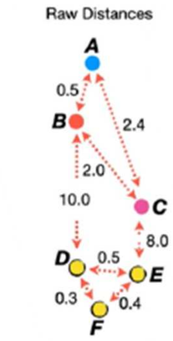
It is not warranted that tSNE will be able to recreate the original matrix.

In tSNE it's the operator that decides how to measure distances.

DIMENSIONAL REDUCTION ≠ DATA CLUSTERING → the perspective defines the separation, thus separation varies according to the perspective. We want to have an unbiased system to assign data points to different groups based on the transcriptional profile, that's why clustering requires a dimensional reduction step before (???)

UMAP (Uniform Manifold Approximation and Projection)

Better than tSNE as it focuses on the whole picture (the overall representation of data). Useful to identify similarities between samples (as it clusters the more similar samples into the final output). Faster than tSNE. UMAP tries to change the structure of our data to fit them in some kind of organization. As tSNE; it uses similarity scores to preserve the disposition of data points from the high-dimension to the low-dimensional space.

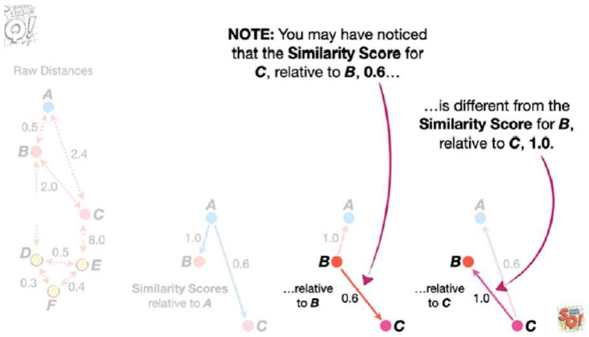


1. UMAP first calculates the distances between each pair of points in the high-dimensional space. If it considers for instance the distance of point A from all the other points, it places then the points on a graph where A is in the position 0. Points belonging to different clusters than that of A will be quite far of course.

2. Then it draws a curve over the data points to calculate similarity scores. The shape of the curve depends on the parameter that you have to set, which is the number of high-dimensional neighbours you want to have for each point (the **nearest neighbours** parameter). Usually this parameter is set on 15, but in this very simple example it is 3 (small dataset, smaller value assigned), and importantly the point of interest (A) is considered among these neighbours (so in this case the points are: A, B, C).
   This curve is drawn in such a way that *the sum of the Y coordinates of the considered points is equal to* $log_2(number\ of\ nearest\ neighbours)$*, and it takes the name of sum of the similarity scores.* It follows that those points too far from the point of interest will have Y coordinate almost equal to 0.

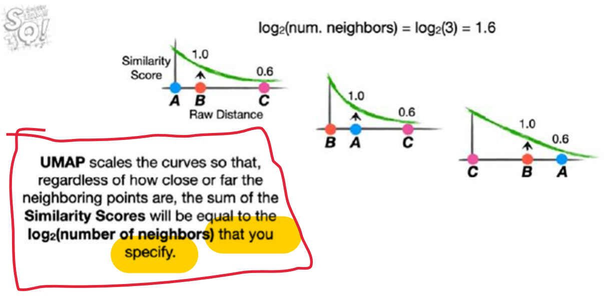
   $$A_y + B_y + C_y = log_2(nearest\ neighbours, i.e.3) = Sum\ of\ similarity\ scores$$

   The similarity scores depend on the direction (the similarity score of A relative to B is not equal to the score of B relative to A), but UMAP scales the curves so that regardless how close or far the neighbouring points are, the sum of the Similarity scores will be equal to the *log* above. So the point is that similarity scores are not symmetrical but UMAP makes them symmetrical.



NOTE: You may have noticed that the **Similarity Score** for ***C***, relative to ***B***, **0.6**…

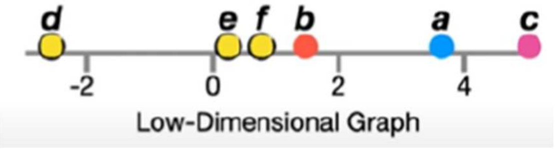
…is different from the **Similarity Score** for ***B***, relative to ***C***, **1.0**.

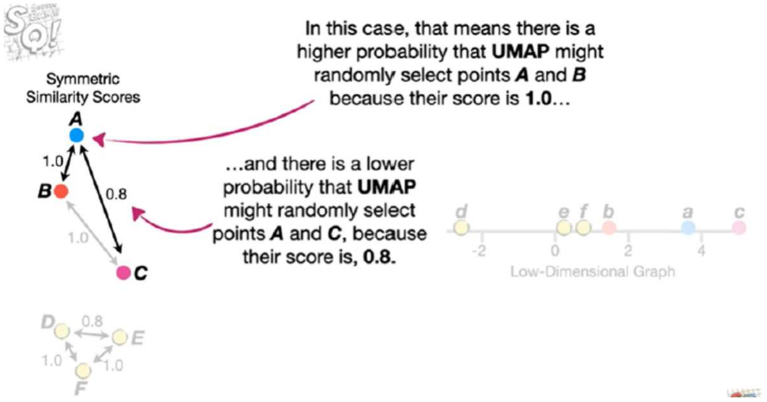


log$_2$(num. neighbors) = log$_2$(3) = 1.6

UMAP scales the curves so that, regardless of how close or far the neighboring points are, the sum of the **Similarity Scores** will be equal to the **log$_2$(number of neighbors)** that you specify.

3. At this point UMAP initializes the **low-dimensional graph**, however this does not display the exact clustering situation that we had in the high-dimensional graph. Differently from tSNE, which first spreads out points on the low-dimensional graph and then grouped them properly, UMAP first aggregates these points and then reduces them to the low-dimensional space. Since this low-dimensional graph is not precise as the high-dimensional one, UMAP picks 2 low-dimensional points that should move closer to each other (e.g., point A selected to move closer to point B, the second point selected → there's higher probability for UMAP to pick up points A and B as their similarity score is higher than that of other possible combinations). It selects A closer to B or B closer to A with 50% probability for each selection as they were made identical.



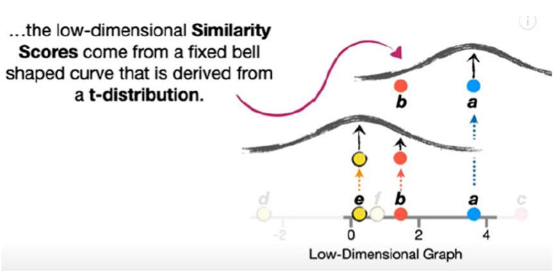


In this case, that means there is a higher probability that **UMAP** might randomly select points ***A*** and ***B*** because their score is **1.0**…

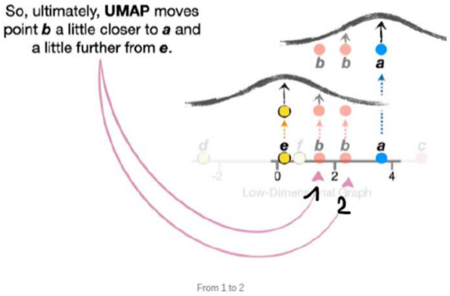
…and there is a lower probability that **UMAP** might randomly select points ***A*** and ***C***, because their score is, **0.8**.

Let's consider this example: B was selected to move closer to A. However, UMAP also selects a point that B should move far from, and thus it selects one point out of B's high-dimensional cluster, let's say one point between D, E, or F. Each of these 3 points has equal probability to be picked up, as their similarity scores do not affect the choice.

UMAP calculates now how much these selected points should be moved on the low-dimensional graph in order to better reproduce the situation of the high-dimensional space (I guess): it calculates **low-dimensional similarity scores** for each pair of points (i.e. the pair of elements to be moved closer, and the pair of points that need to be further). In this case, these similarity scores are again the Y coordinates of the points BUT on a curve that is a t-distribution (centred on the pair of points: in the figure below, A and B). Since A and B fall in the same cluster in the high-dimensional space, UMAP wants to move B closer to A to maximize their low-dimensional similarity score (higher Y coordinate), and on the other hand wants to minimize the score of B and E (different clusters in the high-dimensional space).

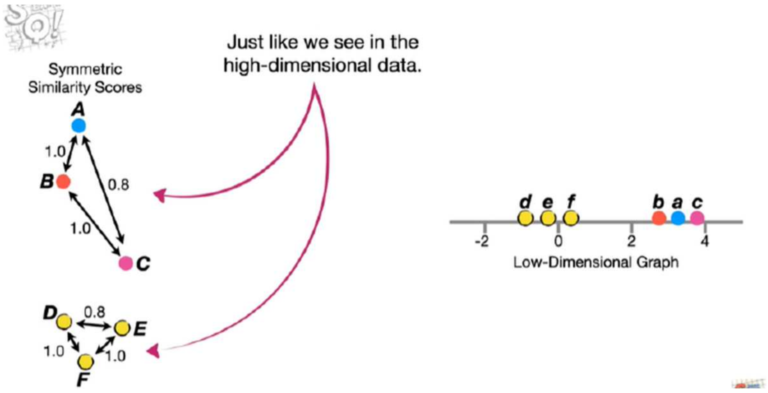


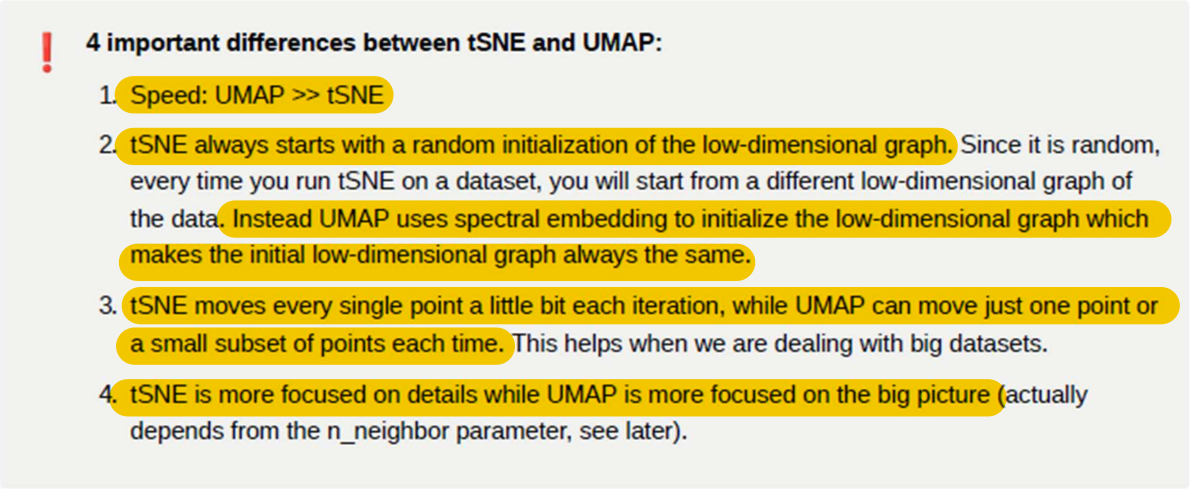
…the low-dimensional **Similarity Scores** come from a fixed bell shaped curve that is derived from a **t-distribution**.



So, ultimately, **UMAP** moves point ***b*** a little closer to ***a*** and a little further from ***e***.

The procedure is repeated for every couple of points until the low-dimensional graph correctly represents the original data points (as we see in the high-dimensional data):


Symmetric Similarity Scores
A
B
C
1.0
0.8
1.0
D
E
F
0.8
1.0
1.0
Just like we see in the high-dimensional data.
d e f
b a c
-2
0
2
4
Low-Dimensional Graph

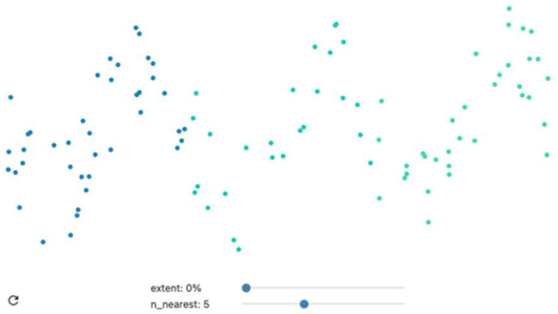


4 important differences between tSNE and UMAP:

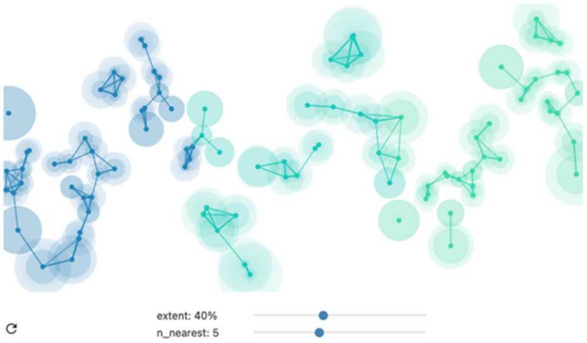
1. Speed: UMAP >> tSNE
2. tSNE always starts with a random initialization of the low-dimensional graph. Since it is random, every time you run tSNE on a dataset, you will start from a different low-dimensional graph of the data. Instead UMAP uses spectral embedding to initialize the low-dimensional graph 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).

Since UMAP is generally better at preserving the global structure in the final projection, the inter-cluster relationships are more meaningful than in tSNE. However, **both UMAP and tSNE deform the high-dimensional shape of** data when they project them to lower-dimensional spaces: for this reason, **axes/distances in these lower dimensions cannot be directly interpreted** as we could do with PCA instead.

We already saw the **n_nearest** parameter (nearest neighbours). The **extent** parameter (a %) is related to how much the dots are extended to make a point in the graph: greater extension = greater connection between dots. Playing on connection is useful to make a plot.



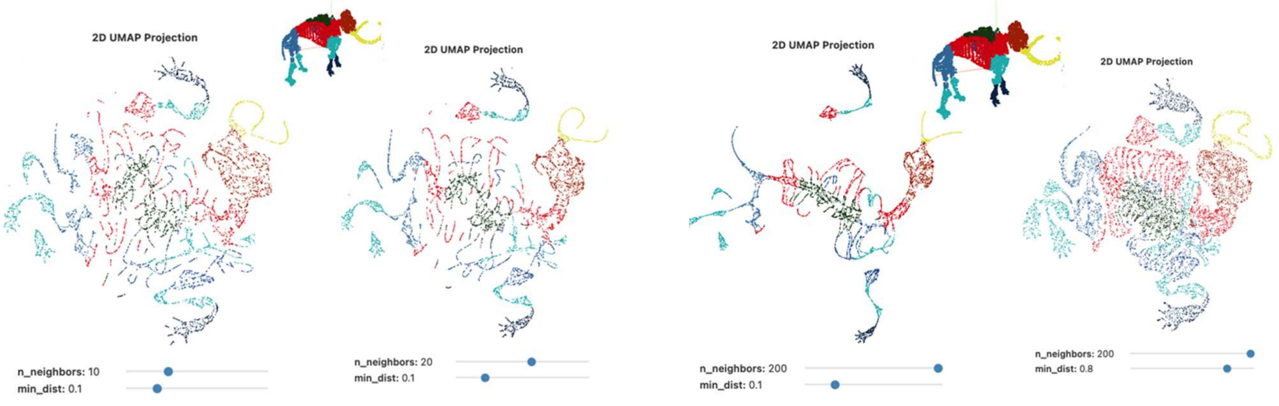
extent: 0%
n_nearest: 5



extent: 40%
n_nearest: 5

Below we played on **n_nearest** and **min_dist** parameters. Lower n_nearest values (low #neighbours) result in smaller and more independent clusters → good to focus on details, but the overall picture idea is lost.

Vice versa (less details, representative overall picture) when n_nearest is set to higher values → This approach helps us focusing on the big picture even though we cannot choose the distance measure we prefer (as we could in tSNE) ???? what's the min_dist then?

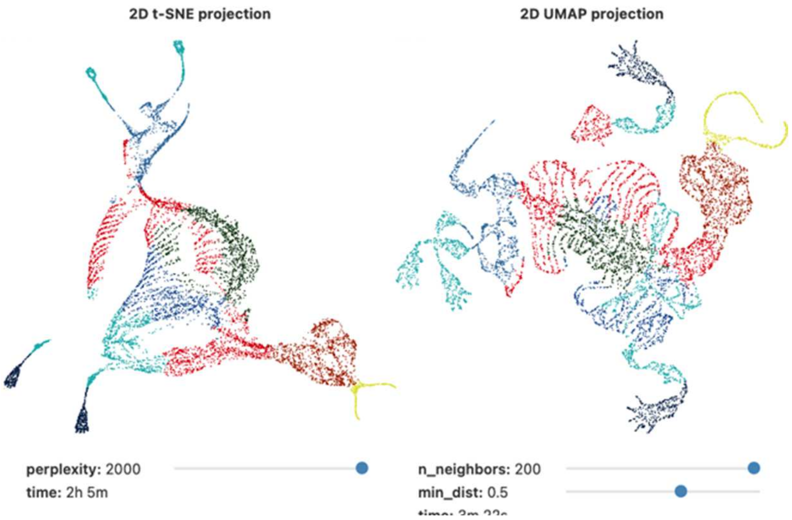


n_neighbors: 10
min_dist: 0.1

n_neighbors: 20
min_dist: 0.1

n_neighbors: 200
min_dist: 0.1

n_neighbors: 200
min_dist: 0.8

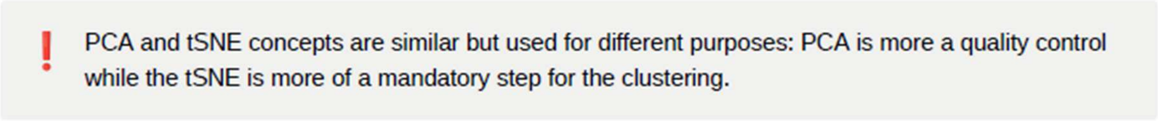


perplexity: 2000
time: 2h 5m

n_neighbors: 200
min_dist: 0.5
time: 3m 22s

Using tSNE (left graph), it takes an extremely high **perplexity** to start noticing the global structure emerging, and at these perplexity values the time to compute is dramatically longer. Moreover, tSNE projections vary a lot from run to run (different pieces of the higher-dimensional data are projected to different locations every time), while with UMAP the resulting projections are surprisingly similar from run to run (and with different parameters) despite the stochastic nature of the algorithm.

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.