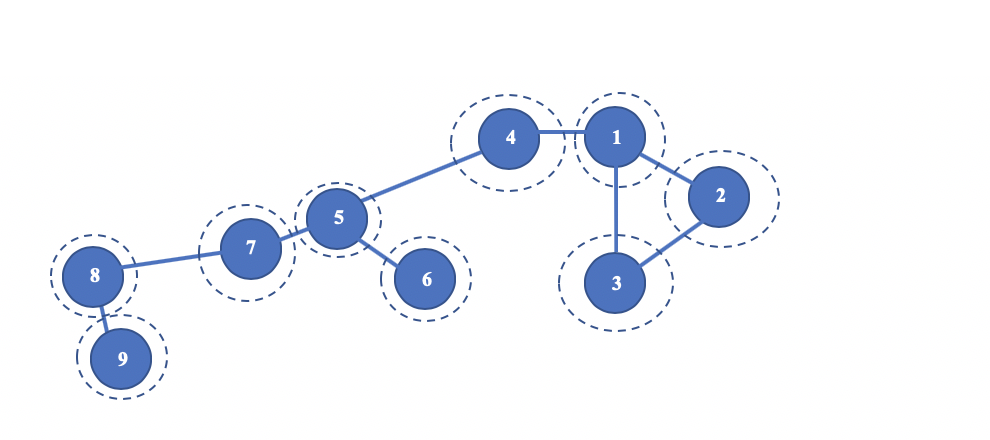
**Aprendizaje de la semana del 31/05/21:**

Grafo de KNN:

El grafo de KNN se construye a partir de un determinado número de componentes principales (PCs) o UMAPs de cada una de las células. A partir de esos PCs se establecen las distancias euclideanas entre las células de un dataset y se determina, para cada célula, cuales son las k vecinas más cercanas y se unen estas vecinas con un edge (conexión). Cuando se dice que se obtiene un Grafo de KNN, de lo que se está hablando es que se obtiene minimamente una matriz de adjacencia. Esta es una matriz cuadrada (de células x células) llena de 0s y 1s que indican con un 1 si la célula *x* es vecina de la célula *i.* Gráficamente se puede presentar como un gráfico donde las células son nodos y están unidas con sus k células vecinas por edges. Los edges podrían tener algún grosor de acuerdo al peso o weigth que se les asigne, pero creo que no es este el caso.

Louvain:

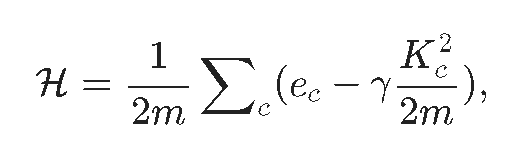
El método parte de un grafo de KNN en donde las células están cada una asignada a un cluster diferente (osea, cada célula es su propio cluster):

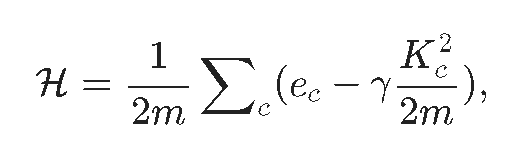


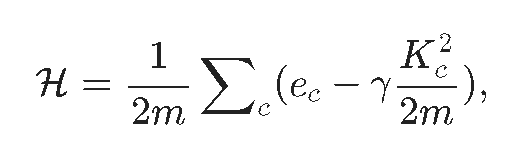
<https://medium.com/walmartglobaltech/demystifying-louvains-algorithm-and-its-implementation-in-gpu-9a07cdd3b010>

A partir de esta situación los nodos s van a mover de una cluster a otro (u otra manera de pensarlo es que los cluster se van a fusionar) sí y solo sí la calidad del clustering aumenta.

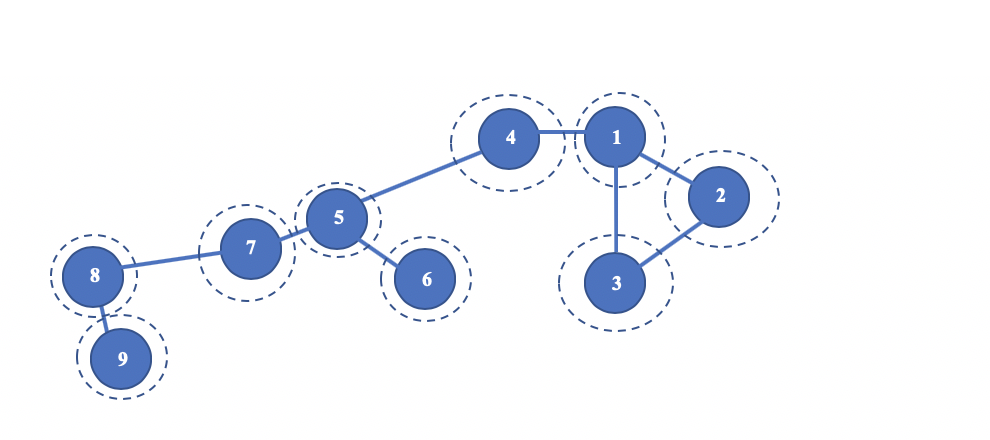
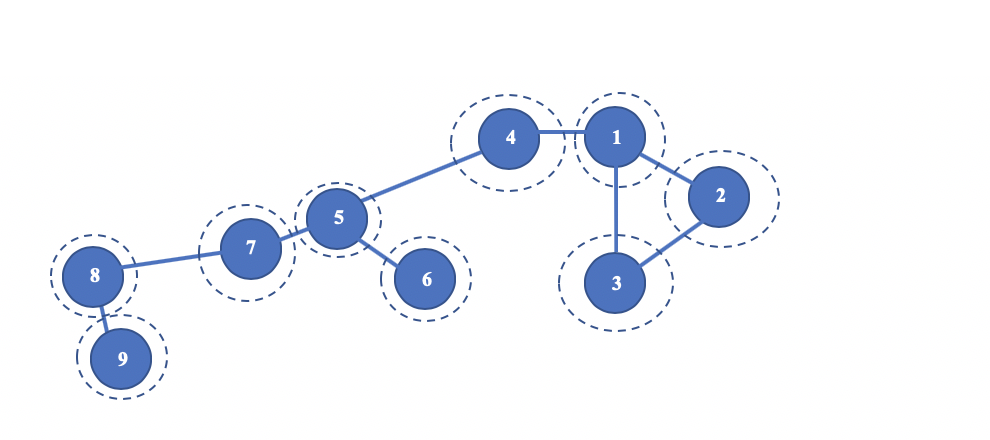
¿Qué es esto? Bueno, la calidad de clustering se puede medir con alguna función en particular (en general se usa la función de Modularidad o la de Constant Potts Model (CPM) que son muy parecidas, pero la que usan Seurat y Monocle3 es la de Modularidad). La función de Modularidad se puede expresar de la siguiente manera:



Donde  *ec* es el número de edges en la comunidad c (si fusiono el cluster 1 y 2 será 1; si además fusiono el 3 será 3.).  es el número esperado de edges si las células estuvieran conectadas al azar (esto sería como pensar que las conexiones entre las células están “equitativamente” distribuidas en todo el dataset, entonces nos preguntamos cuántos edges espermos entre el cluster 1,2,3 fusionado en ese caso). Dentro de esta fórmula *Kc* es la suma de los grados de los nodos de la comunidad c (grados = a cuántos nodos está conectado cada nodo de forma directa) y *m* es el número total de conexiones en la red.

En este caso  es el parámetro de resolución que modula cuántos clusters como máximo se podrían obtener sin que colapse el algoritmo (ya que sino existiría el límite de resolución del cual después voy a hablar).

La calidad del clustering va a aumentar, cuando la función de modularidad aumente. Es decir, cuando el número actual de edges para todos los nodos de la red sea mayor al esperado por azar, es decir, mayor al de una situación donde todos los edges están equitativamente distribuidos:

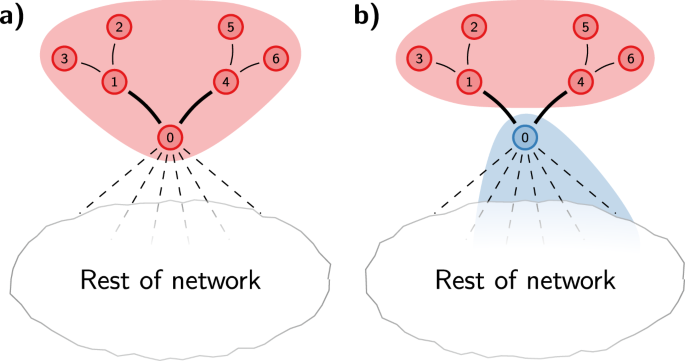
Esquema de conexiones no al azar y formación de clusters

Esquema aprox. de conexión al azar

Entonces, volviendo a Louvain… ¿Qué es lo que hacía el algoritmo? Este va moviendo nodos de un cluster a otro (y obviamente que si en la situación inicial dentro de un cluster hay 1 solo nodo, este cluster desaparece) siempre y cuando la función de Modularidad aumente. El movimiento es nodo a nodo y el algoritmo puede pasar más de una vez por un nodo hasta que ya ningún movimiento se puede hacer para que la Modularidad siga aumentando. Una vez que que esto sucede, se procede a la etapa de agregado, donde se intenta ver si se pueden mergear los clusters. En esta etapa, los clusters se convierten en los nuevos nodos y se procede de la misma manera, moviendo un nodo a un cluster nuevo solo si la modularidad aumenta. Entiendo que si el nodo original tenía un peso ahora el nuevo nodo (ex cluster) tiene la suma del peso de los nodos que contenía cuando era un cluster, pero… ¿Esto vale en el contexto de single cell? ¿O los nodos acá no tienen peso? ¿El número de nodos dentro de un excluster influye en el peso del nuevo nodo?

Si bien este algoritmo es simple y elegante, tiene algunos problemas:

Puede generar comunidades mal conectadas o hasta desconectadas. Esto puede suceder porque algún nodo que hacia de puente entre dos secciones de un mismo cluster, en algun momento el algoritmo decide que debe pertenecer a otra comunidad (porque la función de Modularidad aumentaría), entonces lo mueve, dejando un cluster con dos subclusters deconectados. Esto está muy bien explicado en el paper de Leiden, en donde muestran esta figura:



https://www.nature.com/articles/s41598-019-41695-z#Sec11

En esta figura se puede ver que, al moverse el nodo 0 a otra comunidad, la comunidad rosa queda con dos sub-clusters (1,2,3 y 4,5,6) desconectados, ya que la única conexión entre ellos era el nodo 0. Lo que parecería más lógico sería dividir el cluster rosa en dos, pero Louvain no puede contemplar hacer esto ya que no puede dividir cluster ya formados, solo mueve nodos de a uno si optimiza la modularidad y en este caso no hay forma de optimizarla moviendo por ej. el nodo 1 a un nuevo cluster donde esté él solo. Si bien este es un caso un poco extremo (el de sub-clusters desconectados), el concepto se puede extrapolar a otras situaciones donde el efecto sea más sutil y no genere desconexión pero sí comunidades mal conectadas.

¿Se puede iterar e método de Louvain para conseguir mejores resultados? Sí, se puede iterar partiendo de los clusters generados en el paso previo y volviendo a mover los nodos dentro de los clusters, de un cluster a otro. Si bien esto se puede iterar varias veces hasta que el clustering no cambie (y esto significaría que los clusters está óptimamente separados y que ningún nodo se puede mover), el problema de los clusters mal conectados va a agravarse más aún, porque cuanto más mueva los nodos, más chance de que quite de ciertos clusters los nodos puente.

Laiden:

Laiden surge como una solución al problema de Louvain ya que el concepto es muy similar pero tiene un par de pasos más y no solo 2 como Louvain (que tiene la fase de movimiento de nodos y la fase de agregación).

Lo que hace Leiden se puede dividir en tres pasos:

1. Moviemiento local de los nodos: es exactamente igual al primer paso de Louvain
2. Refinamiento de los clusters generados: Los clusters generados se parten en sub-clusters si es posible
3. Agregación: igual que antes, se toman a los clusters generados anteriormente y se los utiliza como nodos y se repiten los pasos 1 y 2.

Entonces ¿cómo es el paso del refinamiento? El refinamiento consiste en partir de nuevo de la situación inicial en donde cada nodo es una propia comunidad e ir mergeando nodos. Sin embargo, los nodos que se pueden mergear son los nodos que pertenecen a la misma comunidad establecida en el paso 1 (movimiento local de nodos). Además, los nodos SOLO se mergean si está bien conectados con la comunidad establecida en el paso 1. Después de esta fase, las comunidades del paso 1 pudieron ser divididas, aunque no siempre es el caso.

Otra diferencia, además del paso 2, que tiene Leiden con Lovain es que en el paso 1, en este nuevo algoritmo no se visitan los nodos un montón de veces. Lo que pasa en Louvain es que un nodo pede ser visitado más de una vez para saber si moverlo aumenta la modularidad o no lo hace, esto genera que se visiten nodos que tal vez no vayan a ser movidos. Computacionalmente esto hace que el algoritmo sea mas lento. Por el contrario Leiden utiliza un mecanismo que se llama Fast Local Moving, en donde solo se visitan los nodos para los cuales su vecindario haya cambiado ¿Qué quiere decir esto? Quiere decir que se visitan nodos para la cuales, alguno de sus vecinos se haya movido a otra comunidad. Este mecanismo parte de la base de un sistema de listas, pero que está bien explicado en el paper y no es tan importante para los efectos de este texto. En el paper dicen que esto se asemeja a la idea de “pruning” (¿será similar a prunned KNN?)

Límite de Resolución (Copiado de Wikipedia y traducido porque me pareció muy claro):

La modularidad compara el número de edges dentro de un clúster con el número esperado de bordes que uno encontraría en el clúster si la red fuera una red aleatoria con el mismo número de nodos y donde cada nodo mantiene su grado, pero los edges se conectan aleatoriamente. Este modelo nulo aleatorio asume implícitamente que cada nodo puede conectarse a cualquier otro nodo de la red. Sin embargo, esta suposición no es razonable si la red es muy grande, ya que el horizonte de un nodo incluye una pequeña parte de la red, ignorando la mayor parte. Además, esto implica que el número esperado de edges entre dos grupos de nodos disminuye si aumenta el tamaño de la red (asumo yo que es porque el grado del nodo se mantiene). Entonces, si una red es lo suficientemente grande, el número esperado de edges entre dos grupos de nodos en el modelo nulo de modularidad puede ser menor que 1. Si esto sucede, 1 solo edge entre los dos clústeres sería interpretado por modularidad como un signo de una fuerte correlación entre estos, y la optimización de la modularidad conduciría a la fusión de los mismos, independientemente de las características de los clústeres. Por lo tanto, incluso los clusters débilmente interconectados, que tienen la mayor densidad posible de edges internos y representan las mejores comunidades identificables, se fusionarían mediante la optimización de la modularidad si la red fuera lo suficientemente grande. (Esta última frase está reinterpretada por mí, originalmente se refiere a grafos débilmente conectados)

<https://en.wikipedia.org/wiki/Modularity_(networks)>

Es por esto que se agrega el parámetro de resolución, gamma, al término del modelo nulo (es decir el modulo con conexiones al azar), para controlar la importancia entre los edges internos de cada cluster. Si el parámetro de resolución aumenta, la importancia de los edges internos aumenta, entonces eso hace que la diferencia entre las conexiones actuales y las esperadas por azar sea más chica. Por ende, evita que la modularidad aumente mucho y por lo tanto restringe ciertos movimientos o mergeos de los nodos. Entonces la inclusión gamma, va a ser es generar más clusters que si este no estuviera y cuanto más grande gamma, más clusters genera.

Que tiene Monocle3:

Monocle3 puede correr tanto Louvain como Leiden, sin embargo, si corre Louvain elige automáticamente el valor de resolución a utilizar y lo que sí se puede modificar es el k del KNN y

Que tiene Seurat:

El método de Seurat se puede correr tanto Louvain como Leiden y se puede modificar el parámetro de resokución, el k del KNN, y

Funciones de integración de Seurat

La integración en Seurat (al menos en la versión 4) se puede dividir en dos grandes pasos: FindIntegrationAnchors (o traducido: encontrar las anclas de integración) que es encontrar el par de cada célula de un dataset en el otro (siempre que sea posible) y IntegrateData que es el paso en donde las counts se corrigen para que un dataset se ajuste al otro en función de los pares previamente encontrados y el “peso” o “confianza” de cada par.

La función FindIntegrationAnchors trabaja de la siguiente manera:

A partir de un set de genes (o features) definido (puede ser definido arbitrariamente o mediante la función SelectIntegrationFeatures) genera una reducción dimensional (*p=reduction; df = CCA*) de estas variables y toma los primeros n componentes de esta (*p=dim; df = 30*). Con estos n componentes se hace un Mutual Nearest Neighbourhood (MNN). Este consiste en, para cada célula del dataset A encontrar los k vecinos más cercanos (*p = k.anchor; df = 5*) en el dataset B e igual pero viceversa, es decir, para cada célula del dataset B, encontrar los k vecinos más cercanos en el dataset A. Todas las células que resulten mutuamente vecinas formaran un par o “ancla” (anchor). Ejemplo:

Dataset B

w

x

z

Dataset A

j

i

k

En azul vemos marcadas las k vecinas de la célula i en el dataset B y en rojo vemos marcadas las k vecinas de las células x en el dataset A. Vemos acá que las células i y x son mutuamente vecinas

Algo que también puede pasar es que una célula tenga más un ancla. Supongamso Este caso:

Dataset B

w

x

z

Dataset A

j

i

k

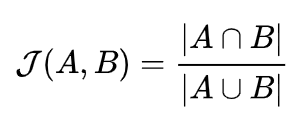
En este caso: i y x son anclas y j y x son anclas.

Una vez hecho esto, se filtran los anchors poco confiables, pero ¿Cómo se define un anchor poco confiable?

Un anchor poco confiable se define como el anchor que no se corresponde como tal en el espacio dimensional no reducido, osea en el perfil de expresión sin reducir. Pero ¿cuál es este espacio? Es el espacio X, dado el perfil de expresión usando solo los 1eros n genes (*p = max.features ; df = 200*) más variables (osea, para este default sería como un espacio de 200 dimensiones, cada dimensión es la expresión de 1 gen expresada en counts normalizadas). El hecho de que en número de dimensiones sea más grande hace que la distancia entre las células esté representada por datos más específicos y el hecho de que no esté reducido hace que las distancia entre las células de una dataset vs. el otro sean más grandes (tengo que entender bien por qué es esto pero creo que tiene que ver con que en los datos reducidos la escala está acotada). Entonces, en este espacio de dimensiones que separa más el dataset A y B de nuestro ejemplo, si la célula *i* todavía se puede encontrar como la k vecina de la célula *x* (*p = k.filter; df = 200*) – no importa que la vecindad sea mutua – entonces el anchor es confiable. De lo contrario el anchor es poco confiable y se remueve.

Finalmente la función asigna un score al anchor con un método similar al Shared Nerest Neighbourhood (SNN) y este score es el “peso” que va a tener el anchor. Para calcular el SNN se usa un determinado k de vecinos (*p = k.score; df = 30*). Lo que hace puntualmente es calcular el Shared Neigbour Overlap (o también conocido como índice de Jaccard) del anchor (osea, entre la célula i y x de nuestro ejemplo).

El Índice de Jaccard es una medida de similitud entre las células de un mismo cluster. SE basa en la suposición de que si dos células pertenecen a un mismo cluster, entonces son similares y entonces es esperable que compartan un buen número de vecinos. Lo que hace el coeficiente de Jaccard es calcular la relación entre el número de shared neighbours (vecinos compartidos) y el número de vecinos no compartidos entre dos pares de células. En su forma general también se puede expresar como el número de elementos del conjunto intersección de los conjuntos A y B sobre el número de elementos del conjunto de unión de A y B, siendo esto:



https://es.wikipedia.org/wiki/%C3%8Dndice\_Jaccard

Poniendo como ejemplo el esquema anterior, podríamos pensar que en además de i y x ser vecinas, estas comparten un número de vecinas comunes:

Dataset B

w

x

z

Dataset A

j

i

k

l

m

u

v

g

h

f

s

t

r

En este caso el Shared Neighbour Overlap sería:

O resulto de otra manera:

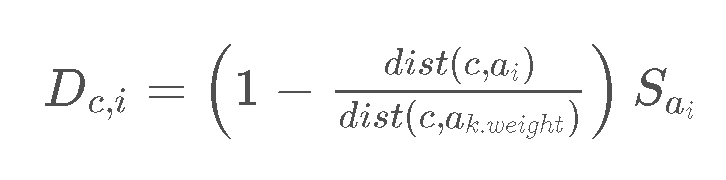
Y a su vez este es el score (S) del anchor, *a*:

En realidad en score del anchor se contruye luego de obtener todos los SON para todos los anchors, tomando nada más los scores que van entre el cuantil 0.01 y 0.90 (para evitar la influencia de outliers) y reescalando estos valores para que entren en la escala de 0 a 1 (que es el rango espero del SNO).

Ahora, volviendo al onjetivo de todo esto…

La función IntegrateData trabaja de la siguiente manera:

Lo primero que hace es construir una matriz de “peso” o weight de los anchors. Recordemos para esto que una célula puede tener más de un anchor (es decir puede estar pareada con dos células distintas). Para cada anchor, el **peso** se define como:



https://www.cell.com/cell/fulltext/S0092-8674(19)30559-8?\_returnURL=https%3A%2F%2Flinkinghub.elsevier.com%2Fretrieve%2Fpii%2FS0092867419305598%3Fshowall%3Dtrue#secsectitle0075

Que se puede traducir cómo:

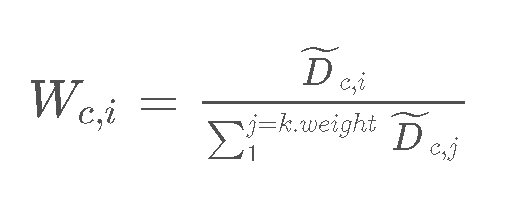
Peso

Vendría a ser una medida de qué tan cercano es ese par respecto al resto (el valor de distancia al “último” par indica el máximo de distancia ente una célula y sus pares). Si la célula es mucho más cercana a este par respecto al resto, el término se achica y D aumenta.

Recordar que es una medida de cuan similares son la célula y su par

El *k*ésimo par se define con *p = k.weight; df = 100.*

Una vez que calcula todos los pesos, los transforma mediante una distribución Gaussiana (no entiendo muy bien si es lo mismo que calcular la probabilidad de encontrar ese peso dada una distribución normal) y luego normaliza el peso de cada par en función del peso de todos los pares de una célula (para tener un peso de un par relativo al resto y si una célula tiene pares que pesan poco respecto a los pares de otra célula, igual poder sopesar el par “mejor pesado” en la integración):



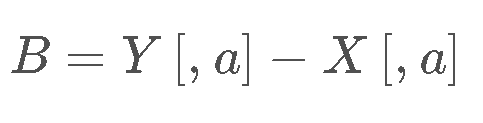
https://www.cell.com/cell/fulltext/S0092-8674(19)30559-8?\_returnURL=https%3A%2F%2Flinkinghub.elsevier.com%2Fretrieve%2Fpii%2FS0092867419305598%3Fshowall%3Dtrue#secsectitle0075

Hasta acá entonces tenemos el peso de cada par, ahora lo que haremos es corregir el perfil de expresión de uno de los datasets usando este peso.

¿Cómo se hace esto?

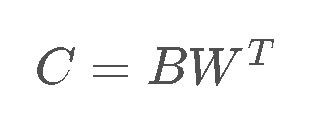
Bueno, en principio lo que se hace es restarle el perfil de expresión del dataset B al dataset A (para seguir con nuestro ejemplo).

Si el perfil de expresión de B está dado por la matriz de expresión Y y el perfil de expresión de A está dado por la matriz de expresión X. Tomamos tanto de Y como de X las columnas que corresponden a células que tienen un par o anchor (*a*) y calculamos la diferencia de entre las counts normalizadas de un dataset y el otro:



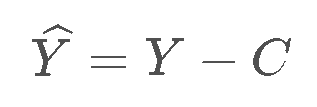
Generando entonces una matriz de la diferencias de expresión, llamada B.

Luego lo que se hace es multiplicar la diferencia de expresión por el peso del anchor, visto en forma de matriz sería mutiplicar la matriz B por la transpuesta de la matriz de pesos:



Generando una matriz de diferencias transformadas, llamada C.

Finalmente se restan las diferencias transformadas al valor de expresión del dataset B, para que se acerque al dataset A, teniendo en cuenta el peso de los anchors. Esto sería expresado como matrices:



Donde Y^ es la matriz de integración, osea la matri de expresión corregida del dataset B, que se va a usar para los futuros análisis (clustering, DE, trayectorias, etc.). Sin embargo en realidad lo que se hace es pegarle a X la matrix Y^, entoncs, tenemos el datasetA original y el dataset B corregido en una única matriz y este ésta la que se usa para las siguiente pasos de integración. Una cosa más a notar es que en esta matriz solo se incluye como genes a las features utilizadas para calcular los anchors pero sí se incluyen a todas las células independientemente de si tienen o no anchors. Todo esto último me dí cuenta corriendo un script de prueba con los datos de Alzheimer y un downsample de 1000 células por cada muestra.

Todo esto, ejemplificado para un solo par podría decirse que se calcula así:

Si la expresión del gen-a en la célula x es 10 y en su par i es 5, entonces:

B x gen-a = 10 – 5 = 5

Si el peso del par xi es 0.80, entonces:

C x gen-a = 5 . 0.80 = 4

Entonces:

Y^ x gen-a = 10 – 4 = 6

Entonces pasamos de tener una situación en donde el gen-a en la célula x es 10 y en su par i es 5, a que en la célula x es 6 y en su par i es 5.

¿Qué pasa cuando integramos más de dos datasets?

Lo que pasa es lo mismo que vimos antes, se generan todos los pares de anchors entre todos los pares de datasets posibles. Luego se determina la similitud entre dos datasets de a acuerdo al número de células en el dataset mas chico del par dividido el numero de anchors totales entre el par de datasets. De esta manera se calculan las distancias entre pares de células y se genera una matriz de distancias entre células para la cual se hace un clustering jerárquico. Esto va a devolver un árbol de jerarquía que va a servir como guía para ir mergeando los datasets de a pares haciendo el proceso de integración que vimos antes (resta y tranformacion de matrices y blah). Una vez que se tienen dos datasets integrado, se integra la matrix de integración resultante con el dataset mas cercano a ese par y así.

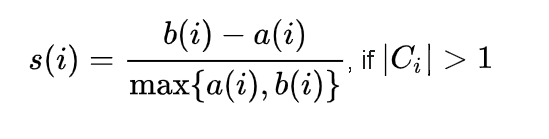
*p* = parámetro ; *df* = default

Metodo ChooseR de Silloutte:

El método de ChooseR sirve para seleccionar automáticamente los parámetros de un clustering. Se basa en evaluar el Coeficiente de Silloutte que va entre -1 y 1 y mide cuan juntas están las c;elula intracluster vs inter cluster. En realidad esta medición la hace para cada célula por separado y lo que normalmente se evalúa para definir la optimicidad del clustering es la media o mediana del Coeficiente de Silloutte para todas la células del dataset.

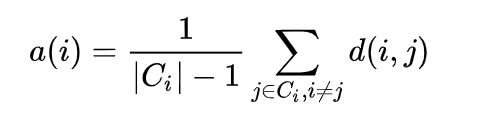
Antes de adentrarnos en el algoritmo en sí, para tener una idea más acabada, expliquemos el Coeficiente de Silloutte:

El Índice de Silluotte de cada punto se define de la siguiente forma



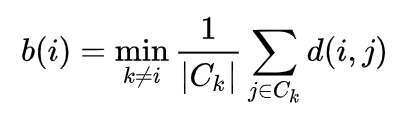
Donde *b(i)* es es la distancia de la célula *i* con la células de uno de los clusters más cercanos y *a(i)* es la distancia de la célula *i* con las células de su propio cluster y *Ci*  es el número de células del cluster donde está la célula *i*.

*a(i)* se puede definir cómo:



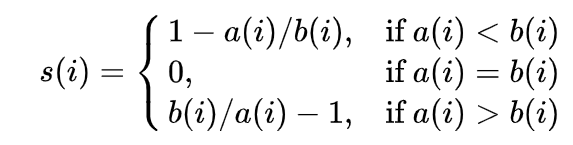
Y esto se puede expresar como el promedio de las distancias entre la célula i y todas las células del cluster *j* más cercano (*d(i,j*)). Sin embargo, en vez de dividir por el número total de distancias, se divide por el número de células en el cluster de la célula *i,* que sería igual a hacer todas las ditancias posibles incluida la célula *i* consigo misma, menos 1 porque no contamos esta última.

*b(i)* se puede expresar cómo:



El promedio de las distancias entre la célula *i* y todas las células del cluster más cercano. Se puede entender ésta con la misma lógica que la ecuación anterior, solo que ahora el *min* del comienzo de la ecuación indica que se va elegir el cluster *k*  para el cual el promedio sea el menor, por lo que se espera que este cluster sea el más cercano (el que va a contener células que minimicen la distancia con *i* respecto a cualquier otro grupo de células de otro cluster).

Lo que se evalua del Índice de Silloutte es en definitiva la relación entre *a(i) y b(i),* porque:

**

*https://es.wikipedia.org/wiki/Silhouette\_(clustering)*

Como dijistme al principio, lo que se evalúa es el prmedio del índice de Silloutte que indica que tan bien están agrupadas las células con las células de su conjunto. El Coeficiente de Silloutte entonces se define como el Índice de Silloutte promedio máximo luego de realizar el clustering iterando una algún parámetro en particular.

Ahora, volviendo a ChooseR:

Este método tiene como base la idea de que hacer múltiples clusters con downsample de las muestras es una mejor representación de la estructura de los datos vs hacer un único cluster para averiguarla. Por lo tanto el método de ChooseR hace el clustering *n* veces (*df = 1000*) usando cada vez un *m* porcentaje de células del dataset original, tomadas al azar y obviamente sin repetición (*df = 0.80*). A partir de estos clusterings calcula la distancia entre dos células cualquiera como 1 - la frecuencia de co-clustering de éstas en las *n* veces corridas, osea, cuantas veces estas dos células clusterizan juntas respecto al total de intentos (es decir, el número total de veces en los que ambas células fueron seleccionados por el downsample). Cuanto mayor sea la frecuencia en la que estas células co-clusterizen, menor será la distancia y como vemos, esta distancia puede ir de 0 a 1.

En éste método en vez de ver el promedio del Indice de Solloutte, lo que calculan es un promedio por cluster (los autores lo llaman Silloutte Score) y luego evalúa la forma de la distribución de estos Silloutte Score para cada modificación de parámetro. A la hora de poner un valor de corte, lo que hicieron fue toma la mediana del Silloute Score por parámetro. En realidad no toman solo la mediana sino que toman un IC 95% de la mediana y luego toman como umbral de score de Silloutte aceptable el límite superior de este IC. El clustering más aceotable entonces no es aquel que tiene una mejor mediana del Solloutte Score, sino el que tiene más clusters y mantiene una mediana de Silloutte Score igual o mayor al umbral del corte.

¿Por qué ChooseR?

Alternative approaches include testing clustering across various parameter regimes and creating a merged consensus clustering, or iteratively applying a machine learning classifier to identify clusters as candidates for re-merging [[14](https://bmcbioinformatics.biomedcentral.com/articles/10.1186/s12859-021-03957-4#ref-CR14),[15](https://bmcbioinformatics.biomedcentral.com/articles/10.1186/s12859-021-03957-4#ref-CR15),[16](https://bmcbioinformatics.biomedcentral.com/articles/10.1186/s12859-021-03957-4#ref-CR16),[17](https://bmcbioinformatics.biomedcentral.com/articles/10.1186/s12859-021-03957-4#ref-CR17)]. However, these approaches are all post hoc, will not split clusters in the case of under-clustering, and assume that over-clustered populations can be hierarchically merged into properly clustered cell types, which is not always the case (Fig. [1](https://bmcbioinformatics.biomedcentral.com/articles/10.1186/s12859-021-03957-4#Fig1)a).

Mail de juan:

<https://pubmed.ncbi.nlm.nih.gov/15944136/>

La distnacia euclideana no es una buena distancia a tomar para datsets de muchas dimensiones (muchos genes).