Pasos realizados de SLICER

Octubre 2018

1. Primer paso: Elección de genes a partir de la Neighbourhood variance

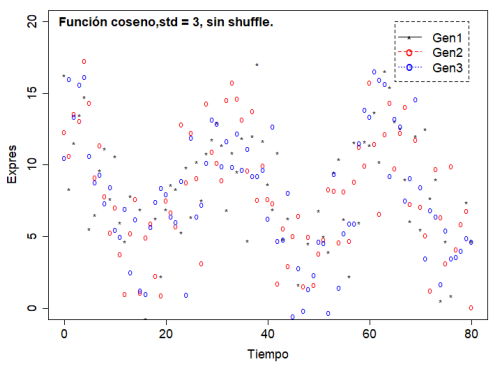
Primero lo que hicimos fue construir una matriz con 100 genes sampleados con 80 muestras (que representarían el paso del tiempo, una muestra al día de vida, otra a la semana y así). Simulamos 5 tipos de funciones regulatorias que contienen 20 genes cada una con las siguientes funciones dependientes del tiempo:

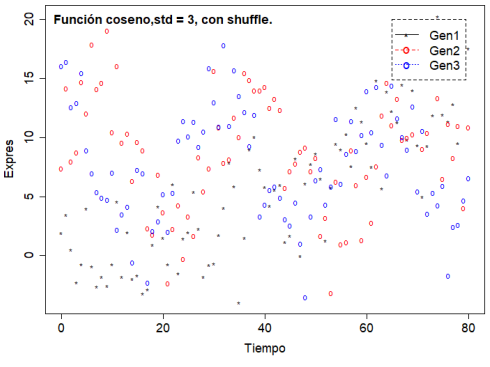
$$\begin{aligned} f_1 &= (5cos(t/5) + 8) + c \\ f_2 &= (5sin(t/5) + 8) + c \\ f_3 &= \sqrt{t} + c \\ f_4 &= (\tfrac{t}{20})^2 + c \\ f_5 &= (16 - \tfrac{t}{20})^2 + c \end{aligned} \tag{1}$$

donde c es una función normalmente distribuida que depende del valor de la desviación estándar que uno le pone de la campana de Gauss (`rnorm()`) .

ACLARACIÓN DE NOTACIÓN: COLUMNAS SON TIEMPO Y FILAS GENES QUE ES AL REVES QUE EN EL TRABAJO [1]

Después de hecho esto es importante borrarle la dependencia explicita con el tiempo. Entonces, a partir de los que proponen en el paper [1] (anteúltimo párrafo pag 13) lo que hacemos es simular genes que no estén relacionados con el proceso. Entonces permutamos los valores simulados de algunos genes para remover así la dependencia con el tiempo. El método consiste en tomar la expresión de 5 genes, uno de cada grupo funcional, y aplicarles `sample()` para hacer un shuffle y luego, renombrar la expresión de dichos genes con la nueva expresión que sale del shuffle (ver Apéndice). Podemos ver en los siguientes gráficos de la expresión de 3 genes del mismo grupo funcional (f_1) en función del tiempo. En un caso con el shuffle y en otro, sin.





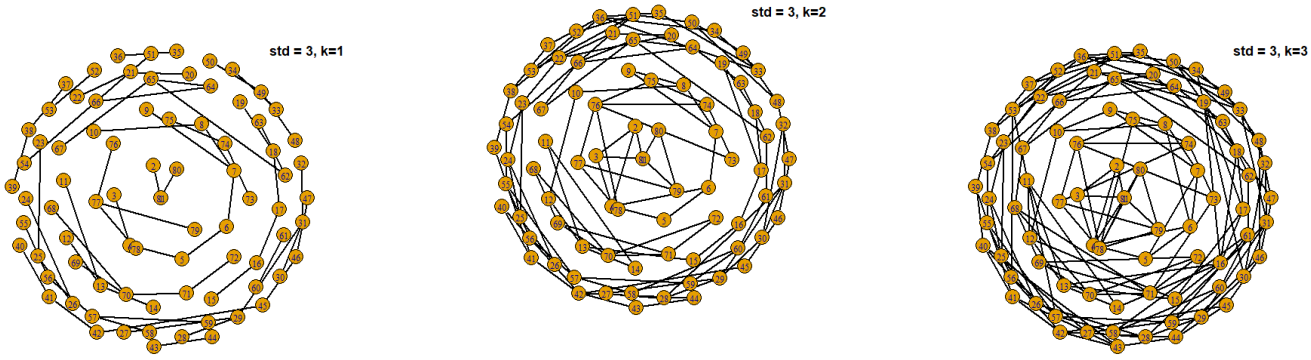
La neighbourhood variance se define para el gen g como:

$$S_g^2 = \frac{1}{nk_c - 1} \sum_{i=1}^{n} \sum_{i=1}^{k_c} (e_{gi} - e_{gN(i,j)})^2 \tag{2}$$

donde para una dada sample i sumamos sobre sus samples vecinas N(i,j). Consideramos una cantidad de vecinos k_c igual para todas las muestras tal que el grafo queda conectado. Y n es la cantidad de muestras total y e_{gi} es la expresión del gen g en la muestra i.

Previo a poder calcular la variance neighbourhood tenemos que averiguar cuales son las muestras vecinas para cada muestra. Lo que hicimos fue tomar a cada vector sample que es cada una de las columnas de la matriz de expresión y calculamos la distancia euclideana con cada una de las demas muestras y armamos la matriz S de samples donde el elemento S_{ij} es la distancia entre el vector sample i y el vector j. Ahora a cada una de las filas de S (`apply()`) les aplicamos la función `rank()` de forma de saber cuáles son sus muestras vecinas más cercanas.

Para hallar el valor de k_c que es la cantidad de vecinos tal que el grafo queda conectado lo que hicimos fue ir probando varios valores. Por ejemplo, si decimos que $k_c = 2$, es decir que cada muestra tiene 2 vecinos, entonces aquellos elementos de S que son mayores a 2 se convierten en 0 y aquellos que son menor se convierten en 1 y así logramos una matriz de adyacencia de las muestras y vemos sus grafos.



Se puede ver que se logra un grafo conectado para un $k_c = 3$ dada nuestra matriz de expresión. Entonces, para cada uno de los genes (100 en total) calculamos la variance neighbourhood y la comparamos con la desviación estandar. Es decir que vimos qué tan varía la expresión de un gen con respecto a sus vecinos y la variación con la media. El criterio para quedarnos o desechar a ese gen es que $\sigma_g^2 > S_g^2$. En la siguiente figura 1 podemos los resultados donde un 90 porciento de los genes fueron aceptados.

2. Segundo paso: La matriz de los pesos de reconstruccion W

Trabajamos con el LLE (Locally linear embedding) que se basa en la idea de que tenemos N vectores de dimension D. Esperamos que cada punto y sus vecinos se encuentren cerca del camino local lineal del manifold. Esta geometria local de los caminos se ve en que podemos reconstruir a cada punto a partir de una combinacion lineal de sus vecinos. Entonces, el error que se comete debe ser minimo.

$$\epsilon(W) = \sum_i (E_i - \sum_{j=1}^{k} W_{ij} E_j)^2 \tag{3}$$

donde E_i es el vector muestra i-ésima donde tiene la expresión de todos los genes aceptados en esa muestra

Otra forma de escribir esto es como un problema de cuadrados mínimos:

$$\epsilon(W) = ||Ax - b||^2 \tag{4}$$

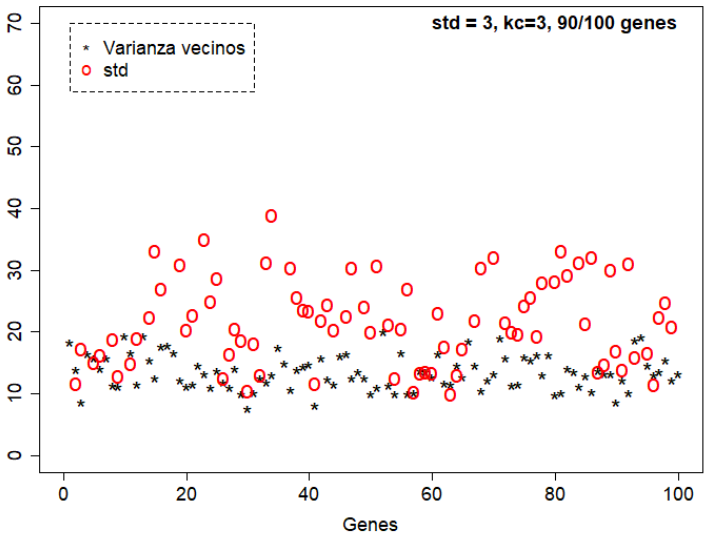


Figura 1

donde b es la columna i ésima de la matriz de expresión, el vector sample i, A es la matriz donde cada columna son los vectores samples vecinos de la sample i y x es el vector que queremos conocer al minimizar que será la columna i ésima de la matriz W.

Después de obtener la matriz W vamos a poder tener a la matriz de expresión en baja dimensionalidad:

$$L = argmin_L \sum_{i=1}^{n} |L_i - \sum_{j=1}^{k} w_{ij} L_j| \tag{5}$$

La bibliografia [2] afirma que se trata de un problema de NxN autovalores donde los d (baja dimensionalidad) no zero autovectores proveen el set ortogonal de coordenadas L_i. Tambien impone algunas condiciones de forma que llega a una matriz M que es la dice que hay que diagonalizar.

3. Apéndice

3.1. Shuffle

```
for (i in seq(1,length(genes)/grupos-1,1)){
      matrixForReshuffle = matrix(nrow = grupos, ncol=length(t))
      matrixReshuffled = matrix(nrow = grupos, ncol=length(t))
      matrixForReshuffle[1,] = matrixExpression[i,]
      matrixForReshuffle[2,]= matrixExpression[i+20,]
      matrixForReshuffle[3,]= matrixExpression[i+40,]
      matrixForReshuffle[4,]= matrixExpression[i+60,]
      matrixForReshuffle[5,]= matrixExpression[i+80,]
      matrixReshuffled = matrixForReshuffle[sample(1:grupos),]
      matrixExpression[i,] = matrixReshuffled[1,]
      matrixExpression[i+20,] = matrixReshuffled[2,]
      matrixExpression[i+40,] = matrixReshuffled[3,]
      matrixExpression[i+60,] = matrixReshuffled[4,]
      matrixExpression[i+80,] = matrixReshuffled[5,]
}
```

Figura 2: Caption

donde *grupos* = 5 la cantidad de grupos funcionales, genes es un vector del 1 al 100, t el vector de las samples, del 1 al 80

En el recuadro rojo tomo a estos 5 vectores de expresión de 5 genes (uno de cada grupo), en el recuadro verde le aplicamos la función `sample()` para hacer un shuffle y por último, en el recuadro azul reescribimos con esos vectores la `matrixExpression` que es la matriz de expresión.

Referencias

[1] WELCH, Joshua D.; HARTEMINK, Alexander J.; PRINS, Jan F. SLICER: inferring branched, nonlinear cellular trajectories from single cell RNA-seq data. Genome biology, 2016, vol. 17, no 1, p. 106.

[2] ROWEIS, Sam T.; SAUL, Lawrence K. Nonlinear dimensionality reduction by locally linear embedding. science, 2000, vol. 290, no 5500, p. 2323-2326.