Universidad de la República

Facultad de Ingeniería

Aprendizaje Automático para Datos en Grafos

Laboratorio 2 | Estructura de grandes redes

Autores:
Federico Bello
Gonzalo Chiarlone

28 de septiembre de 2025

UNIVERSIDAD
DE LA REPÚBLICA
URUGUAY

1. Ejercicio 1: estructura de grandes redes

1.1.

Para calcular la distribución de grados alcanza con utilizar la función *histogram* de *numpy*, normalizando por la cantidad de nodos para obtener la proporción en lugar de la cantidad. De esta forma, se obtiene que la distribución obtenida en este caso es [0.1 0.2 0.4 0.3]. Es decir, el 10 % de los nodos son de grado 0, el 20 % de grado 1, el 40 % de grado 2 y el 30 % de grado 3. Esto se corresponde con el grado observado en el notebook.

1.2.

Se ve que la distribución de la figura 1 se aproxima a una *power law*: se ven altas frecuencias en los grados bajos y bajas frecuencias en los grados altos, asemejándose a una recta (al tomar escala logarítmica) con pendiente descendiente. Sin embargo, el hecho de usar un intervalo equiespaciado para calcular el histograma y luego graficarlo en una escala *log-log* provoca que se acumulen frecuencias y que se expandan los grados (puntos alineados en baja frecuencia). Esto se puede corregir tomando intervalos con particiones logarítmicas en los bins, es decir, tomando bins exponenciales de la forma $2^{n-1} \leq b < 2^n$.

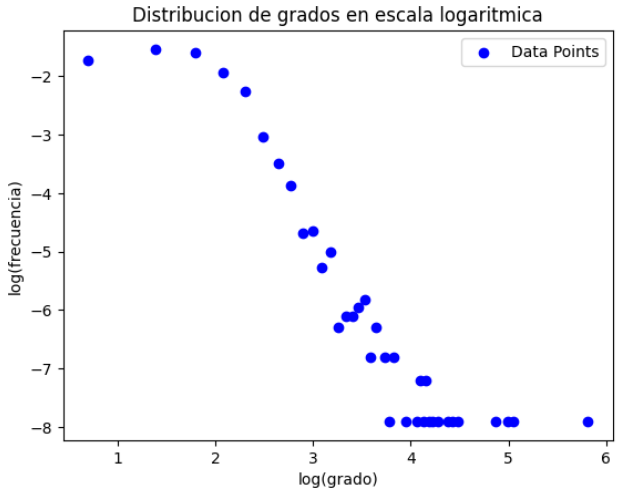

Figura 1: Distribución de grados como función del grado k en escala logarítmica

1.3.

Siguiendo el razonamiento previamente mencionado se creo la figura 2 donde efectivamente se observa una distribución mas parecida a una *power law*, ya que el resultado se asemeja mas a una recta. Esto da mucha mayor seguridad de que la red analizada es una red *scale free*

1.4.

Dada la distribución de Pareto con una función de densidad de probabilidad dada por:

$$p(k) = \begin{cases} Ck^{-\alpha} & \text{si } k \geq k_{\min} \\ 0 & \text{en otro caso} \end{cases}$$

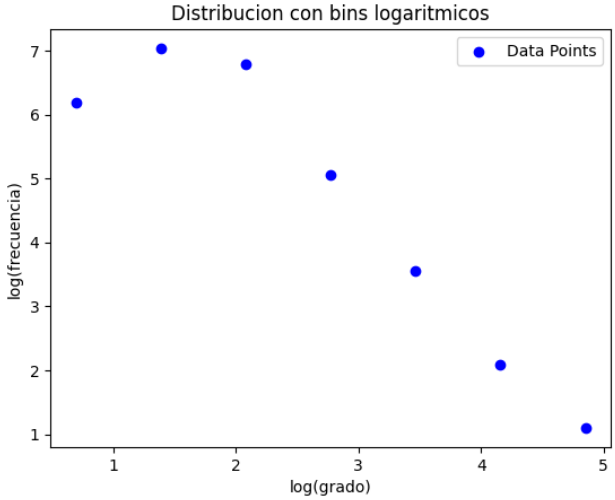

Figura 2: Distribución de grados como función del grado k , en escala logarítmica, con bins exponenciales

se tiene que para que efectivamente sea una distribución se debe cumplir que $C = \frac{\alpha - 1}{k_{min}^{-\alpha+1}}$. El resultado surge fácilmente de imponer que la integral en todo el soporte sea uno, es decir, $\int_{-\infty}^{+\infty} p(k)dk = 1$. De esta forma, se tiene que:

$$\int_{-\infty}^{+\infty} p(k)dk = \int_{k_{min}}^{+\infty} p(k)dk = C \lim_{b \to +\infty} (\frac{k^{-\alpha+1}}{-\alpha+1})\Big|_{k_{min}}^{b} = C(-\frac{k_{min}^{-\alpha+1}}{-\alpha+1}) = 1$$

por lo que despejando C se obtiene el resultado antes mencionado.

La motivación para utilizar esta distribución en lugar de una exponencial es que muchos grafos *scale-free* cumplen esta propiedad a partir de cierto grado. En este caso en particular, parece seguir una *power-law* a partir del grado 10^2 aproximadamente.

1.5.

Para calcular el estimador de máxima verosimilitud para α alcanza con aplicar la formula $\hat{\alpha} = 1 + n\left[\sum_{i=1}^{n} \log\left(\frac{k_i}{k_{\min}}\right)\right]^{-1}$, recordando que $k_i = 0$ si $k_i < k_{min}$. La prueba detrás de este resultado se puede ver en la sección 3. Para calcular este resultado alcanza con computar:

```
alpha_hat = 1 + len(degrees_list) * (np.sum(np.log(degrees_list/k_min)))**-1
```

siempre y cuando el valor del grado sea mayor igual a k_{min}

2. Ejercicio 2: detección de (dos) comunidades

Los algoritmos para dividir los grafos en comunidades se dividen en dos grandes tipos. Los que resuelven el problema de *particionar* un grafo, los cuales consisten en dividir el grafo en comunidades cuyo tamaño esta previamente definido, o los de *detección* de comunidades, que consisten en dividir el grafo pero sin pre-especificar el tamaño de las comunidades. Se define la notación $s_i = \begin{cases} +1 & \text{si el vértice i pertenece al grupo 1} \\ -1 & \text{si el vértice i pertenece al grupo 2} \end{cases}$

2.1.

Los resultados de la modularidad que se calcular fueron los siguientes:

- Modularidad grafo de aeropuertos: **0.108**
- Modularidad grafo de citas: **0.640**

2.2.

Para analizar este resultado, es importante recordad qué es la modularidad. Si bien la formula se enuncia mas abajo, se puede pensar en la modularidad como una noción de qué tan bien separado esta el grafo en comunidades, es decir, que tan modular es el mismo. Teniendo esto en cuenta los resultados anteriores tienen total sentido:

- Por un lado, dado que la clasificación de los aeropuertos es dependiendo el trafico de personas en el mismo, es esperable que este grafo tenga baja modularidad, ya que normalmente los aeropuertos grandes (con muchos transito) tienden a conectar con muchos aeropuertos pequeños. Por otro lado, los aeropuertos pequeños no suelen estar conectados entre si, ya que normalmente se hace transbordo a través de uno grande.
- En este caso, el grafo tiene una alta modularidad. Eso es acorde con lo observado en la realidad: los *papers* tienden a citar *papers* del mismo topico.

2.3. *Spectral partitioning*

Este algoritmo cae dentro de la primer categoría, donde dado un grafo y el tamaño de dos comunidades calcula la partición que minimice el corte[1]. Sin embargo, calcular el mínimo corte de un grafo es un problema *NP-Hard*, por lo que en realidad se desarrolla el método con una función de costo equivalente. Esta consiste en calcular:

$$\hat{s} = \arg\min_{\mathbf{s}} \mathbf{s}^T \mathbf{L} \mathbf{s} \quad \text{s.t.} \quad \mathbf{1}^T \mathbf{s} = n_1 - n_2 \;\; \text{y} \;\; \mathbf{s}^T \mathbf{s} = 1$$

con n_1 y n_2 la cantidad de nodos en cada comunidad. Desarrollando esta función de costo se llega al siguiente algoritmo:

1. Dados un grafo $\mathbf{G}$, su laplaciano $\mathbf{L}$, n_1 y n_2 la cantidad de nodos en cada comunidad calcular λ_2 el segundo valor propio mas chico de $\mathbf{L}$ y $\mathbf{v}_2$ su vector propio asociado.
2. Asociar los n_1 elementos mas grandes de $\mathbf{v}_2$ a la primer clase y los sobrantes a la otra. Calcular el corte.
3. Asociar los n_1 elementos mas chicos de $\mathbf{v}_2$ a la primer clase y los sobrantes a la otra. Calcular el corte.
4. Devolver la partición de las anteriores que minimice el corte.

La razón para elegir el segundo valor propio mas chico así como la explicación del método se pueden ver en [2].

Sin embargo, es de importancia aclarar que es posible obtener mejores resultados utilizando otros algoritmos, en particular, al utilizar mas vectores propios [1]. Intuitivamente, esto se debe a que los vectores

[1] Un corte C de un grafo en grupos V_1 y V_2 es el numero de aristas entre ambos grupos

propios son un *embedding* del grafo, por lo que al utilizar mas se tiene una representación mas fiel del mismo - a costo de utilizar una mayor dimensión.

Finalmente, luego de aplicar el algoritmo previamente descrito, la partición del grafo se puede observar en la figura 3, donde todos los nodos se clasifican correctamente.

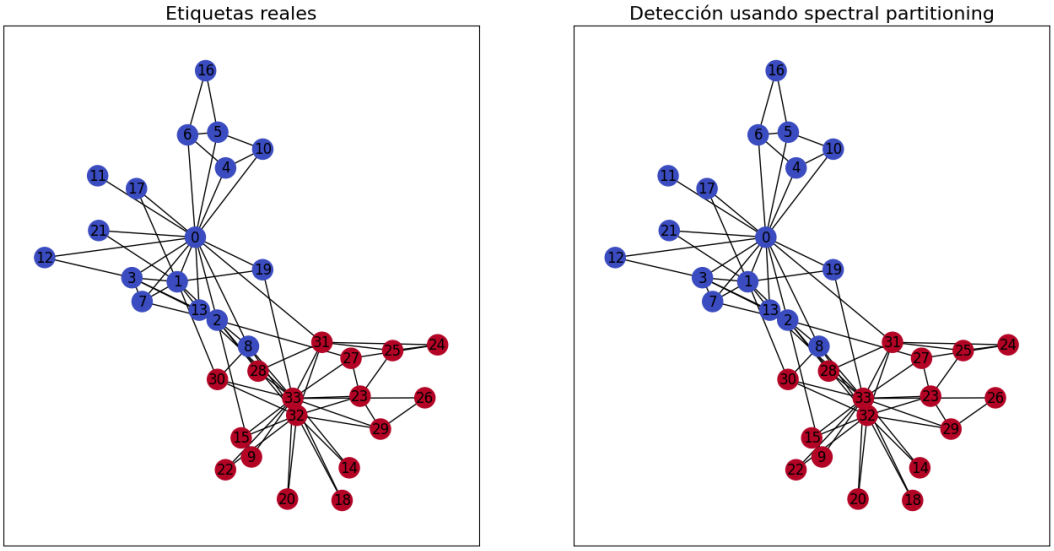

Figura 3: Partición del grafo utilizando *spectral partitioning*

2.4. Spectral modularity maximization

Para este siguiente algoritmo no es necesario conocer previamente la cantidad de nodos en cada comunidad, por lo que se trata de un algoritmo de detección de comunidades.

Similar al anterior, el donde se buscaba minimizar el corte, este busca maximizar la modularidad (Q), la cual es una medida de cohesividad de grupos:

$$Q = \frac{1}{2N_e} \sum_{i,j \ inV} (A_{ij} - \frac{d_i d_j}{2N_e}) \mathbf{1}_{\{g_i = g_j\}}$$

donde g_i y g_j son los grupos de pertenencia de las aristas i y j respectivamente. Notando que $\sum_j B_{ij} = \sum_j A_{ij} - \frac{d_i}{2N_e} \sum_j d_j = d_i - \frac{d_i}{2N_e} 2N_e = 0$ y $\mathbf{1}_{\{g_i = g_j\}} = \frac{1}{2}(s_i s_j + 1)$ se llega a que:

$$Q = \frac{1}{4N_e} \sum_{i,j \in V} B_{ij} s_i s_j = \frac{1}{4N_e} \mathbf{s}^T \mathbf{B} \mathbf{s}$$

donde a $\mathbf{B}$ se le llama matriz de modularidad. Por lo tanto, maximizar la modularidad es equivalente a calcular:

$$\hat{s} = \underset{\mathbf{s} \in \{\pm 1\}^{N_v}}{\arg\max} \ \mathbf{s}^T \mathbf{B} \mathbf{s}$$

Nuevamente, el problema es *NP-hard*, lo que se soluciona fácilmente relajando las restricciones:

$$\hat{s} = \underset{\mathbf{s}}{\arg\max} \ \mathbf{s}^T \mathbf{B} \mathbf{s} \quad \text{s.t.} \ \ \mathbf{s}^T \mathbf{s} = N_v$$

finalmente, utilizando multiplicadores de Lagrange, se llega a que, para maximizar la modularidad, es

necesario elegir el vector propio asociado al valor propio dominante de **B**, tomando el signo de cada entrada para realizar la clasificación.

A modo de resumen, el algoritmo queda de la forma:

1. Dado un grafo G, calcular su matriz de modularidad **B**
2. Elegir el vector propio asociado al valor propio dominante de **B**.
3. Tomar el signo de dicho vector como clasificación de vértices.

En el poco probable caso de que el vector propio tenga alguna entrada cero, se puede elegir indistintamente cualquiera de los grupos dado que ambos son igualmente buenos.

En la figura 4 se observa la partición luego de haber aplicado este método, el cual obtiene un resultado razonablemente bueno, más aún teniendo en cuenta que no se conoce la cantidad de nodos en cada comunidad de antemano. En este caso, solo queda un nodo mal clasificado (el 8), el cual es un error razonable.

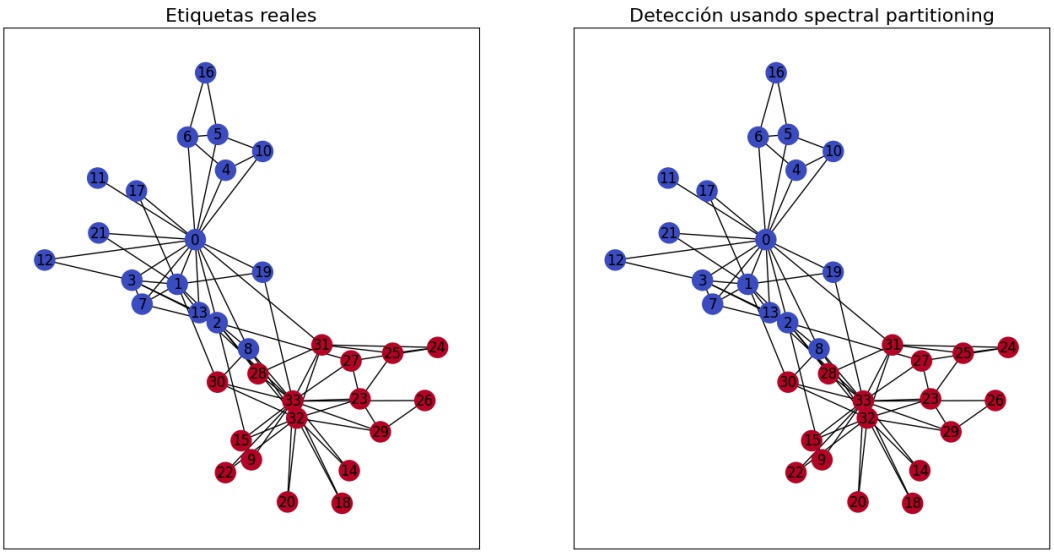

Figura 4: Partición del grafo utilizando *spectral modularity maximization*

Por ultimo, a modo de resumen la tabla 1 muestra de forma comparativa el desempeño de los dos métodos descritos contra clasificar al azar los nodos. La comparación fue utilizando el índice de Rand ajustado y el índice de Fowlkes-Mallows. Ambos comparan las etiquetas predichas contra las etiquetas reales, obteniendo un mayor valor cuanto mas etiquetas tengan en común. Se ve entonces lo esperado, donde el método de *spectral partitioning* tiene el mejor resultado a costo de tener que agregar mas información, mientras que ambos métodos consiguen un resultado ampliamente superior al de clasificar los nodos al azar.

2.5. Problemas con *spectral modularity maximization*

Se observan algunos errores de clasificacion principalmente para los nodos que tienen muchas aristas con la otra comunidadd. Si comparamos las imagenes de la figura 5, en la calsificacion real (izquierda) se ven "puntos amarillos"sobre el gran cluster de nodos violetas, mientras que en la clasificacion calculada (derecha) vemos, en general, dos grandes manchas de diferente color.

Esto nos muestra la deficiencia del algoritmo para clasificar correctamente nodos que no maximizan la modularidad, es decir, nodos con aristas entre comunidades. Si bien es normal que estos nodos existan en grafos reales, es esperable que sea difícil encontrarlos y clasificarlos con este tipo de algoritmos. Mas aun, seria esperable que el algoritmo fallara en clasificar correctamente grafos que tienen una baja modularidad.

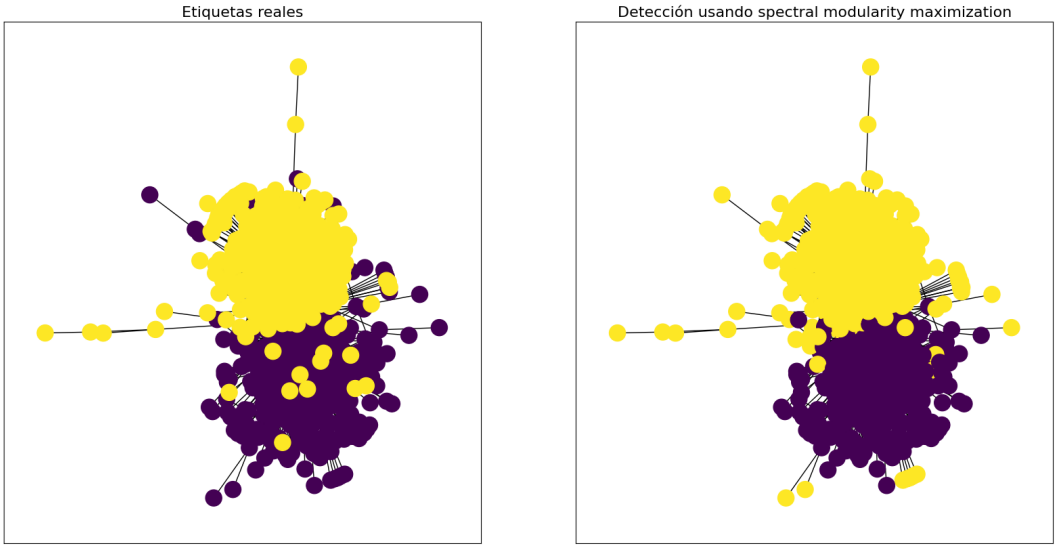

Figura 5: Equiquetas reales vs etiquetas calculadas con *spectral modularity maximization*

Método \ Índice	de Rand	de Fowlkes-Mallows
Spectral Partitioning	1	1
Spectral modularity Maximization	0.882	0.939
Al azar	0	0.696

Cuadro 1: Desempeño de los distintos métodos

3. Ejercicio 3

3.1. Parte 1

Recordando la definición de verosimilitud, se tiene que $\mathcal{L}(\alpha, k_1, \ldots, k_n) \triangleq p(k_1, \alpha)p(k_2, \alpha)\ldots p(k_n, \alpha)$. Aplicando la definición para este caso, se obtiene:

$$\mathcal{L}_n(\alpha, k_{min}, k_1, \ldots, k_n) = \prod_{i=1}^{n} \frac{\alpha - 1}{k_{min}^{-\alpha} k_{min}} k_i^{-\alpha} = \frac{(\alpha-1)^n}{k_{min}^n} \prod_{i=1}^{n} \frac{k_i^{-\alpha}}{k_{min}^{-\alpha}}$$

Por lo tanto, tomando el logaritmo del resultado anterior y aplicando propiedades básicas, se llega a que la log-verosimilitud esta dada por:

$$\ell_n(\alpha, k_{min}, k_1, \ldots, k_n) = n\log(\alpha - 1) - n\log k_{min} - \alpha \sum_{i=1}^{n} \log\left(\frac{k_i}{k_{\min}}\right)$$

3.2. Parte 2

Para conseguir el valor del estimador de máxima verosimilitud, recordar que el mismo es el que maximiza la función $\mathcal{L}_n$. Este valor es el mismo que maximiza la función ℓ_n, ya que el logaritmo es una función

monótona, es decir:

$$\hat{\alpha} = \underset{\alpha \in \mathbb{R}}{\arg\,\max}\ \mathcal{L}_n(\alpha, k_{min}, k_1, \ldots, k_n) = \underset{\alpha \in \mathbb{R}}{\arg\,\max}\ \ell_n(\alpha, k_{min}, k_1, \ldots, k_n)$$

Por lo tanto, alcanza con derivar según α la log-verosimilitud e igualar a 0:

$$\frac{\partial \ell_n(\alpha, k_{min}, k_1, \ldots, k_n)}{\partial \alpha} = \frac{n}{\hat{\alpha} - 1} - \sum_{i=1}^{n} \left(\frac{k_i}{k_{\min}} \right) = 0$$

despejando, se obtiene que $\hat{\alpha}$ vale:

$$\hat{\alpha} = 1 + n \left[\sum_{i=1}^{n} \log \left(\frac{k_i}{k_{\min}} \right) \right]^{-1}.$$

Referencias

[1] Charles J Alpert and So-Zen Yao. Spectral partitioning: The more eigenvectors, the better. In *Proceedings of the 32nd annual ACM/IEEE design automation conference*, pages 195–200, 1995.

[2] L. Hagen and A. Kahng. Fast spectral methods for ratio cut partitioning and clustering. In *1991 IEEE International Conference on Computer-Aided Design Digest of Technical Papers*. IEEE Comput. Soc. Press.