PONTIFICIA UNIVERSIDAD CATÓLICA DEL PERÚ

FACULTAD DE CIENCIAS E INGENIERÍA

Evaluación de las propiedades de los elementos químicos en la construcción de árboles consensus usando las metodologías de Adams y regla del mayor

Tesis para obtener el título profesional de Licenciado en Química

Autor

Alberto Einstein Flores Turpo

Asesor

Maynard Jorge Kong Moreno

Lima, Diciembre de 2021

Resumen

El presente trabajo de investigación desarrolla una metodología para analizar cómo varían los árboles consensus respecto al número de propiedades que se usan para su construcción. Para tal motivo, en primer lugar y como marco teórico, se muestran brevemente las metodologías usadas en el análisis jerárquico de clústers para obtener dendrogramas y árboles consensus (Adams y regla del mayor). Asimismo, también se muestran algunos conceptos puntuales que nos ayudan a estudiar la similitud y disimilitud de una base de datos de prueba (subconjunto de elementos químicos del sistema periódico). En segundo lugar, se muestra la metodología para analizar la variación de los árboles consensus respecto al número de propiedades de la base de datos de prueba. Esta consiste en proponer algunos algoritmos que ayuden a obtener y analizar los árboles consensus cuantificando la similitud y disimilitud, para luego aplicar estos conceptos sobre los subconjuntos de elementos comparables del espacio de base de datos. Luego, se discuten estos resultados considerando un conjunto de 32 elementos químicos descritos por 8 propiedades haciendo énfasis en el costo computacional y en el estudio del número de propiedades para obtener los árboles consensus. Finalmente se analizan los resultados obtenidos y se evalúa una propuesta alternativa basada en redes neuronales que permite reducir el costo computacional. De esta manera se da una posible respuesta más completa a una de las preguntas abiertas hasta ahora sobre el sistema periódico.

Agradecimientos

En principio, agradezco a Dios por haberme permitido acceder a la grata experiencia de ser estudiante universitario.

A mi familia, por el apoyo incondicional que me brindaron sin importar la distancia que nos separa.

A mi universidad y sus docentes, por la formación integral que me proporcionaron.

A los Drs. Guillermo Restrepo y Maynard Kong, por su apoyo y orientación constante durante el desarrollo de este trabajo de investigación.

Dedicatoria

Dedicado a mis padres y a todas aquellas personas sin cuyo apoyo e inspiración no hubiera sido posible este trabajo de investigación.

Evaluación de las propiedades de los elementos químicos en la construcción de los árboles consensus usando las metodologías de Adams y regla del mayor

Índice

Índice de figuras

Índice de tablas

1. Marco Teórico

1.1. El sistema periódico

El sistema periódico de los elementos químicos ha sido parte de una de las muchas discusiones que se ha tenido en el ámbito científico [1]. Sin embargo, es común que no se tenga en claro su definición cuando se compara con la ley periódica o la tabla periódica. Si bien ambas definiciones están muy relacionadas entre sí, lo cierto es que cada una hace referencia a cosas totalmente distintas. En efecto, estas se pueden definir de la siguiente manera:

* Sistema periódico: Estructura que se forma cuando se considera el orden y similitud entre los elementos químicos.
* Tabla periódica: Mapa del sistema periódico hacia un espacio bidimensional. La estructura de este espacio bidimensional puede tomar diferentes formas.
* Ley periódica: Es el conjunto de oscilaciones observadas respecto de algunas propiedades de los elementos químicos en función al número atómico (Z).

Como se puede observar, el estudio del sistema periódico es aquello que resulta de considerar la similitud y el orden; mientras que la ley periódica es el producto del conjunto de oscilaciones de las propiedades de los elementos cuando se considera que están en función del número atómico. Sobre esta última, cuando se intentó incluir todas aquellas propiedades que no dependen del número atómico (esto es, tratar de generalizar la ley periódica), se demostró que ello no era posible [2].

Una manera de abordar este estudio parte del punto de vista matemático. El estudio matemático da origen a estudios no fenomenológicos como los de Carbó Dorca y Robert (2000) y/o Khramov (2006) -los cuales se basan en descriptores atómicos y moleculares, además de densidades electrónicas- [3, 4] y a estudios fenomenológicos. Estos últimos, a su vez se pueden dividir en 3 grandes grupos o metodologías: la primera se basa en agrupamiento y relaciones difusas; la segunda, en redes neuronales de Kohonen; y la tercera, en el análisis jerárquico de clústers, respecto de la cual se hará énfasis más adelante. Cabe resaltar que dentro de los estudios fenomenológicos, el avance del análisis jerárquico de clústers es el que más se ha desarrollado y gracias a ello se logró explicar hasta cierto punto la ley periódica [5–13].

1.2. Metodologías del estudio del sistema periódico

1.2.1. Agrupamiento y relaciones difusas

Esta metodología está basada en determinar relaciones *max-min transitivas* (μ_R), las cuales se obtienen de una relación de compatibilidad difusa en base a un teorema [14, 15]. Esta relación de compatibilidad es obtenida de la matriz de distancias. La obtención de la matriz de distancias está determinada por la selección

de un conjunto de elementos, cada uno con propiedades ya normalizadas. Por ejemplo, si la métrica es la euclidiana, la relación a usar es:

$$\mu_R(x_i, x_k) = 1 - \frac{\left[\sum_{j=1}^{m}(x_{ij} - x_{jk})^2\right]^{1/2}}{d}$$

Para todo $i, k \in X \times X$ y además la métrica euclidiana definida como:

$$d = max\left\{\left(\sum_{j=1}^{m}(x_{ij} - x_{jk})^2\right)^{1/2} : 1 \leq i, k \leq n\right\}$$

Teniendo en cuenta que se usan n elementos y m propiedades. La relación max-min transitiva genera agrupamientos dependiendo del grado de similitud α. A estos agrupamientos se les conoce como $\alpha-$ cortes, pues cada uno indica un agrupamiento distinto. Estos $\alpha-$ cortes son determinados en base a la siguiente relación:

$$\mu_{R_T^\alpha} = \begin{cases} 1, & \text{Si } \mu_{R_T}(x, y) \geq \alpha \\ 0, & \text{Si } \mu_{R_T}(x, y) < \alpha \end{cases} \tag{1}$$

De esta manera, Georgiou y Karakasidis (2010), con $\alpha \in \langle 0{,}9451110, 0{,}9520259]$, usando 2 propiedades físicas (primera energía de ionización y número atómico) lograron obtener la distribución de los periodos de la tabla periódica. El agrupamiento que obtuvieron fue [6]:

$\{$
$\{H\}$
$\{He\}$
$\{Li, Be, B, C, N, O, F, Ne\}$
$\{Na, Mg, Al, Si, P, S, Cl, Ar\}$
$\{K, Ca, Sc, Ti, V, Cr, Mn, Fe, Co, Ni, Cu, Zn, Ga, Ge, As, Se, Br, Kr\}$
$\{Rb, Sr, Y, Zr, Nb, Mo, Tc, Ru, Rh, Pd, Ag, Cd, In, Sn, Sb, Te, I, Xe\}$
$\{Cs, Ba, La, Ce, Pr, Nd, Sm, Eu, Gd, Tb, Dy, Yb, Lu, Hf, Ta, W, Re, Os, Ir, Pt,$
$Au, Hg, Tl, Pb, Bi, Po, Rn\}$
$\{Ra, Ac\}$
$\}$

1.2.2. Redes neuronales de Kohonen

Para entender la aplicación de las redes neuronales al estudio del sistema periódico, es menester mencionar la definición de red neuronal y tener conocimiento de su forma de operación, por lo que se pasará a describir estos puntos evitando formalismos matemáticos. En ese sentido, por red neuronal se puede entender a una colección de unidades (neuronas) que están conectadas por capas, de tal manera que es posible transmitir información entre dichas capas [16]. Una representación de la red neuronal se puede apreciar en la Figura 1.

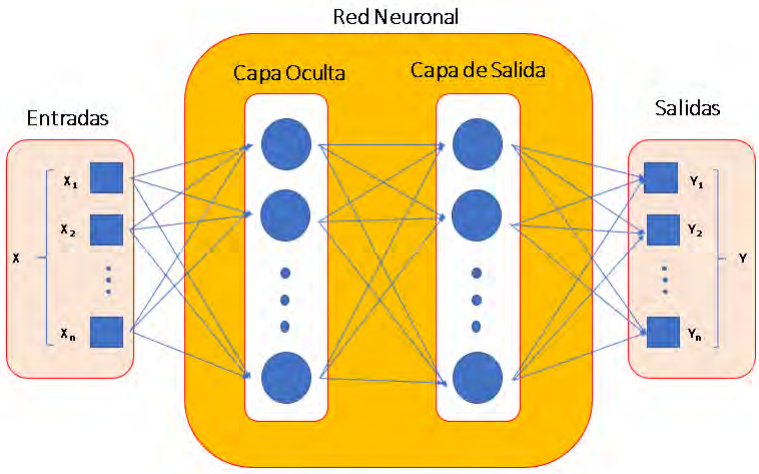

Figura 1: Representación de una red neuronal. [16]

En la Figura 1, cada neurona es un cuadrado o círculo de color azul, las conexiones están representadas por flechas. Se debe tener en cuenta que la cantidad de subcapas en la capa oculta es arbitraria. En general, para que una red neuronal funcione, es necesario tener una entrada conocida y su salida correspondiente. Esta entrada se ingresa en la capa de entrada y se obtiene una salida, la cual puede ser correcta o no. Si no es correcta, lo que se hace es corregir el error, pero desde la capa de salida hacia la capa de entrada con un algoritmo (i.e *backpropagation*). Se repite este procedimiento muchas veces, a fin de que luego de un tiempo la red neuronal logre predecir correctamente los resultados. A este procedimiento se le denomina *aprendizaje de la red neuronal*.

Las redes neuronales de Kohonen (Figura 2) no necesitan esta etapa de aprendizaje, pues se podría decir que estas redes neuronales aprenden por cuenta propia. Su funcionamiento está basado en que cada neurona está mapeada a un espacio bidimensional de coordenadas y se evalúan entradas de manera aleatoria de tal manera que estas causan que las coordenadas del espacio bidimensional sean ajustadas. Así, a medida que se repite esta operación, las posiciones finales de las neuronas tienen a converger. Esta convergencia es lo importante, pues la disposición final de las posiciones nos da indicios de grupos de similaridad; y, de esta manera, se pueden evaluar cuáles grupos de similaridad poseen los elementos químicos.

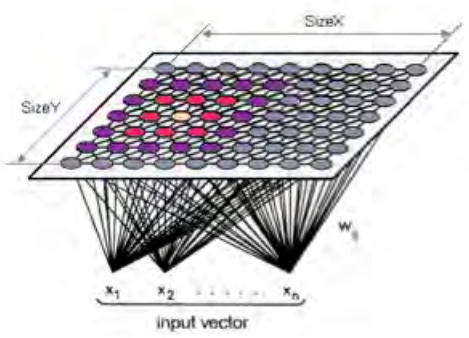

Figura 2: Representación gráfica de una red neuronal de Kohonen. [17]

Chen (2010) usó estas redes neuronales haciendo uso de un espacio de 12 x 8, un vector de dimensión 10 para describir las propiedades de los elementos, obteniendo 3 clústers (metales, no metales y semimetales), y distribuciones de similitud entre los elementos químicos como se muestra en la Figura 3 [11].

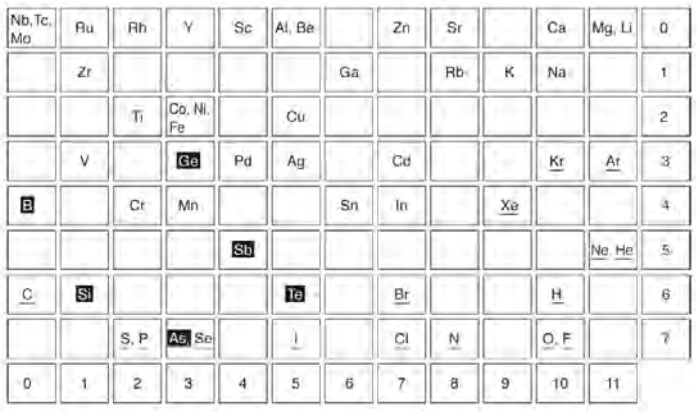

Figura 3: Representación del espacio topológico obtenido por Chen (2010) en base a redes neuronales de Kohonen [11].

1.2.3. Análisis jerárquico de clústers

El análisis jerárquico de clústers puede ser estudiado de dos maneras, las cuales se diferencian en el uso de la función de (dis)similitud. Para ambos métodos, es necesario representar cada elemento químico como usualmente se hace en quimiometría, por una *n-tupla* de sus propiedades fisicoquímicas $(p_{i1}, p_{i2}, p_{i3}, \ldots, p_{in})$ [18]. Luego se procede a normalizar estas propiedades en base a la ecuación:

$$\overline{X}_{jA} = \frac{X_{jA} - X_{jmin}}{X_{jmax} - X_{jmin}} \tag{2}$$

Donde X_{jA} es el valor de la propiedad j del elemento químico A; X_{jmin} y X_{jmax} son el mínimo y máximo j sobre todos los elementos químicos considerados en el conjunto. Luego suelen usarse métricas sobre Q a las que se les conoce como funciones de similitud (en general, pues no toda función de similitud es una métrica).

Para determinar las distancias entre los clúster que se van formando, es necesario considerar la relación introducida por "Lance-Williams", según [18].

$$f(k,i) = \alpha_A * f(A,i) + \alpha_B * f(B,i) + \beta * f(A,B) + \gamma * [f(A,i) - f(B,i)] \tag{3}$$

Donde A y B son objetos a agrupar, k es la unión de A y B, $f(A,i)$, $f(B,i)$, $f(A,B)$ son las funciones de similitud entre A y i,B y i, y A y B respectivamente. Los valores de α_A, α_B, β, γ dependen de la metodología a seguir. En la Tabla 1 se muestra la lista de algunas funciones de similitud, y en la Tabla 2 algunas de las metodologías empleadas usualmente.

Tabla 1: Funciones de similitud empleadas usualmente.

Distancia de Hamming	$d(A,B) = d(B,A) = \sum_{i=1}^{n} abs\left(x_{iA} - x_{iB}\right)$
Distancia euclidiana	$d(A,B) = d(B,A) = \left[\sum_{i=1}^{n} \left(x_{iA} - x_{iB}\right)^2\right]^{1/2}$
Distancia Gower	$d(A,B) = d(B,A) = \left[1 - \frac{\sum_{i=1}^{n} x_{iA} * x_{iB}}{\sum_{i=1}^{n} x_{iA}^2 + \sum_{i=1}^{n} x_{iB}^2 - \sum_{i=1}^{n} x_{iA} * x_{iB}}\right]^{1/2}$
Similitud coseno	$S(A,B) = \frac{\sum_{i=1}^{n} x_{iA} * x_{iB}}{\left[\sum_{i=1}^{n} x_{iA}^2 * \sum_{i=1}^{n} x_{iB}^2\right]^{1/2}}$

Tabla 2: Metodologías usadas en análisis de clústers.

Metodología	α_A	α_B	β	γ
Enlace promedio no ponderado*	$\frac{n_A}{n_A+n_B}$	$\frac{n_B}{n_A+n_B}$	0	0
Enlace hacia el centro *	$\frac{n_A}{n_A+n_B}$	$\frac{n_B}{n_A+n_B}$	$-\frac{n_A*n_B}{(n_A+n_B)^2}$	0
Enlace completo	0.5	0.5	0	0.5
Enlace simple	0.5	0.5	0	-0.5

En base estas funciones de similitud y metodologías, se procede a formar la matriz de distancias. Luego se va agrupando cada dos elementos, considerando la distancia mínima sobre toda la matriz. Estos dos nuevos elementos forman un clúster, y luego se vuelve a construir la matriz de distancias, la cual posee un elemento menos que la inicial. Se repite este procedimiento hasta que sólo quede un par de elementos, los cuales finalmente representan el último clúster. Todas estas operaciones realizadas pueden ser representadas como un árbol binario, al cual se le denomina *dendrograma* (Figura 4).

Los dendrogramas son determinados con una metodología y una función de similitud. Al existir muchas metodologías y funciones de similitud, existen muchos dendrogramas. Por tal motivo, se procede a determinar un dendrograma representativo al cual se le conoce como *árbol consensus*. La obtención del árbol consensus también está determinado por una metodología, pero estas se basan en cómo agrupar sub-árboles de un dendrograma; de esta forma existen muchas maneras de obtener un árbol consensus. En el presente trabajo se usan las metodologías de Adams y regla del mayor; y para su mejor entendimiento y comprensión se definen algunos conceptos basados en [19].

Los árboles consensus pueden ser considerados como grafos acíclicos conexos. Las entidades que clasifican a estos grafos se denominan *taxones*. Un grupo es un subconjunto del conjunto de taxones. Se denota como $ab|c$ al agrupamiento entre a y b relativo a c. Si decimos que $ab|c$ es una terna de un árbol raíz, significa que el ancestro común más profundo de a y b pertenece al subárbol con raíz en c.

La metodología de Adams está definida para aquellos árboles que tienen raíces. En primer lugar, supongamos que $\pi_1, \pi_2, \pi_3, \ldots, \pi_k$ son todas las particiones del conjunto de taxones. El producto de estas particiones

es la partición π, para la cual, dos particiones a y b pertenecen al mismo bloque si y solo si están en el mismo bloque para cada una de las particiones $\pi_1, \pi_2, \pi_3, \ldots, \pi_k$. Ahora, el *clúster maximal* de un árbol raíz T es el clúster propio más largo de T. La *partición clúster maximal* de T es la partición $\pi(T)$ del conjunto de taxones con bloques iguales al clúster maximal de T.

Por ejemplo, el producto de $ab|cde$ y $ac|bde$ es $a|b|c|de$, puesto que d y e pertenecen a un mismo bloque en ambas particiones. Ahora, sean $T_1 = (((((a,b),c),d),e),f)$ y $T_2 = (((((d,e),f),a),b),c)$ (notación *newick*), la partición clúster maximal de T_1 es $abcde|f$ y la partición clúster maximal de T_2 es $def ab|c$, por tanto el producto de estas dos particiones es $abde|c|f$. Ahora queda analizar el taxón $\{a,b,c,e\}$. Este genera los clústers $(((a,b),d),e)$ y $(((d,e),a),b)$ para T_1 y T_2 respectivamente. La partición clúster maximal es ahora $abd|e$ y $dea|b$, generando como producto $ad|b|e$. De esta manera, el árbol consensus para T_1 y T_2 usando la metodología de Adams es $(((a,d),b,e),c,f)$.

En principio, la metodología de la regla del mayor se define como aquella que construye el árbol consensus de una lista de árboles usando clústers que aparecen en más de la mitad de esta lista. Sin embargo, es posible dar una definición generalizada de esta metodología fijando un parámetro extra. Este parámetro indicaría el porcentaje mínimo de ocurrencia de los clústers en común en la lista de árboles. Por ejemplo, dado una colección de 3 árboles raíz $(((a,(b,c)),d),(((a,b),c),d),(((a,b),d),c)$, los clústers $\{a,b\},\{a,b,c\},\{a,b,c,d\}$ aparecen 2 de las 3 veces posibles; por tanto, el árbol consensus obtenido por la metodología de la regla del mayor usando el parámetro 0.66 es $(((a,b),c),d)$.

Los estudios topológicos de similaridad se hacen sobre los árboles consensus. Para realizar un estudio topológico de similaridad, es necesario determinar las bases topológicas, y estas se obtienen definiendo el valor de n para los n - subárboles maximales. El valor de n puede ser calculado en base a: si n es óptimo, entonces el valor del número de selección S_n es el máximo posible entre todos los posibles valores de n. Es importante mencionar que existen muchos valores de n que maximicen el número de selección. Este valor puede ser calculado mediante la siguiente relación [13].

$$S_n = C_n * \prod_i |C_{ni}| \tag{4}$$

Donde C_n reprensenta el número de clústers formados y C_{ni} la cardinalidad del i-ésimo clúster cuando se hacen los cortes.

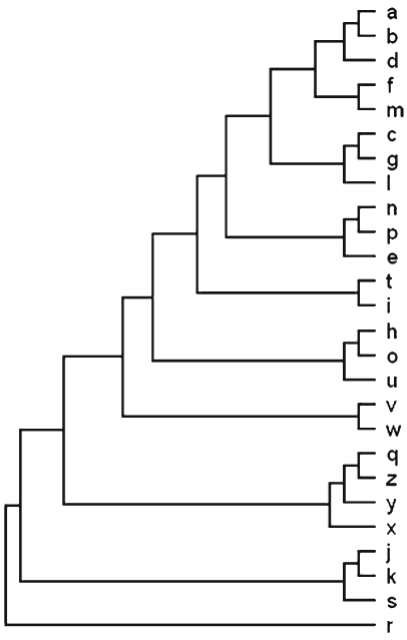

Figura 4: Representación gráfica de un dendrograma de 26 elementos.

Sobre los dos métodos que se mencionaron con anterioridad, estos sólo se diferencian cuando se usa una función de similitud o disimilitud. Las funciones de similitud son las que se muestran en la Figura 1 y para estas solo es necesario trabajar con las propiedades de los elementos; mientras que una función de disimilitud es aquella que considera la diferencia simétrica como métrica y, por tal motivo, se trabaja sobre compuestos binarios.

La idea radica en considerar como distancia la cardinalidad del conjunto que se forma al considerar los elementos que no son comunes a ambos compuestos: a menor cantidad de este valor, mayor será la similitud que guardan los elementos. Por su parte, el resto del procedimiento sigue tal y como ya se mencionó.

El análisis jerárquico de clústers ha logrado obtener relaciones de similitud como su naturaleza (metales, no metales o metaloides, por ejemplo), además de mostrar la relación diagonal, el principio de singularidad, efecto par inerte, la formación de los grupos (alcalinos, calcógenos, gases nobles, etc), las cuales son conocidas como relaciones periódicas [7–10, 12, 13, 20, 21].

Finalmente, se pueden representar todas las investigaciones sobre el sistema periódico en un esquema que muestra algunas consideraciones generales de los trabajos de investigación (estos se muestran en la Tabla 3). En adición, algunas de las relaciones entre elementos que se obtuvieron sobre estos estudios se muestran en la Figura 5.

Tabla 3: Estudios fenomenológicos del sistema periódico.

Autor(s) /Año	Zhou et. al. (2000)	Sneath (2000)	Karakasidis (2004)	Restrepo et. al. (2004).	Restrepo et. al. (2006)	Chen (2010)	Leal et. al. (2012)
Elementos/ número atómico	H-Sn/Z=1-50	H-La, Hf-Bi/Z=1-57, 73-83	H-Nd, Sm-Dy, Y-Lu, Hf-Po, Rn, Ra-Ac/ Z=1-60, 62-66, 70-71, 72-84, 86, 88-89	H-La, Hf-Rn/ Z=1-57, 72-86	H-La, Hf-Rn/ Z=1-57, 72-86	H-Xe/ Z=1-54	H, Li-F, Na-Cl, K-At, Ra-Es/ Z=1, 3-9, 11-17, 19-85, 88-89
Número de elementos	50	69	83	72	72	54	94
Número de propiedades	7	54	2	31	128	10	4700
Tipo de propiedades	físicas	físicas y químicas	físicas	físicas y químicas	físicas y químicas	físicas y químicas	químicas
Función de (dis)similaridad	Similaridad difusa	coeficiente de Gower	Similaridad difusa	funciones de Hamming, Gower, Euclidiana y coseno	funciones de Hamming, Gower, Euclidiana y coseno	Red de Kohonen	Diferencia Simétrica

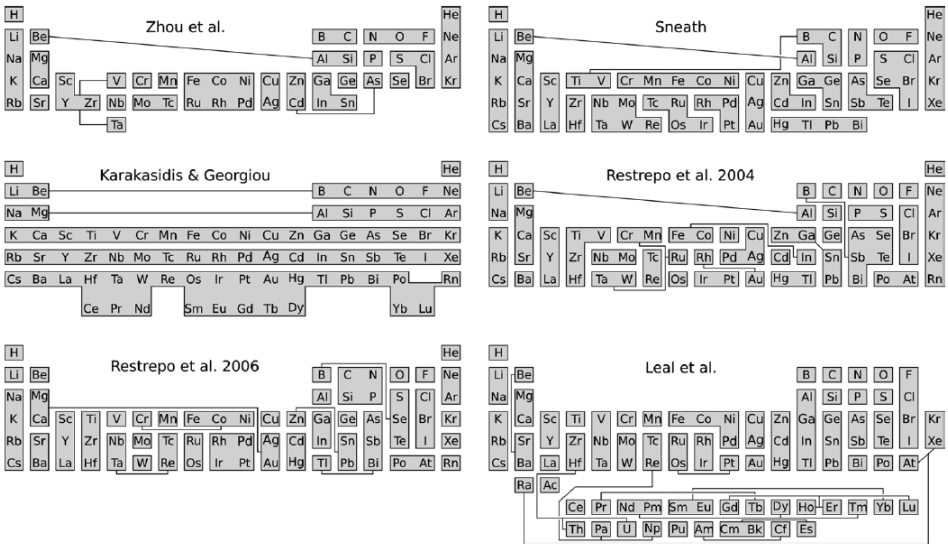

Figura 5: Clasificación de los estudios fenomenológicos del sistema peródico. [22]

Es importante tener en cuenta el trabajo de Zhou del 2000 [23], pues este dio inicio al desarrollo sobre el estudio del análisis jerárquico de clústers usando funciones de similitud como se aprecia en la Tabla 3 y la Figura 5. Asimismo, ya desde este trabajo, se formaron interrogantes sobre la ley periódica. P. H. A. Sneath (2004) dio algunas respuestas a las interrogantes, respecto de las cuales, G. Restrepo demostró más tarde que algunas de sus respuestas carecían de generalidad, puesto que solamente trabaja sobre dendrogramas y no sobre árboles consensus. Puede tomarse como ejemplo la propuesta dada por P. H. A. Sneath (2004), quien afirmó determinar con un análisis de PCA, que muchas de las propiedades guardan correlaciones entre sí, lo que indica que muchos de estos valores no son necesarios para determinar los dendrogramas. De esta manera, es natural plantearse calcular el número mínimo de propiedades que son necesarios para determinar las relaciones de similitud entre elementos de un dendrograma. Esta idea puede extrapolarse sobre árboles consensus y confirmar así, si las propiedades guardan correlaciones, como menciona P. H. A. Sneath.

2. Objetivos

2.1. Objetivo general

El objetivo principal del trabajo de investigación es determinar el número mínimo e identidad de las propiedades periódicas necesarias para obtener árboles consensus, de tal manera que mantengan gran parte de las relaciones de similitud y disimilitud. Manteniendo la similaridad de estos árboles, se propone un método basado en las metodologías de "la regla del mayor" y "Adams".

2.2. Objetivos específicos

Como objetivos específicos, se espera:

* Evaluar el costo computacional y eficiencia entre este método propuesto, en comparación a otras alternativas, en la determinación del número mínimo e identidad de las propiedades periódicas.
* Proponer mejoras basadas en redes neuronales, en cuanto a reducir el costo computacional del método propuesto para determinar el número mínimo e identidad de las propiedades periódicas.

3. Parte Experimental

La estrategia de trabajo empezó con la búsqueda de la base de datos, la cual se recolectó desde Kaggle [I]. Se organizó y filtró esta base de datos y luego se exportó hacia un repositorio Github [II]. Esta base de datos proporcionó la mayor parte de la información usada. Sin embargo, ha requerido una revisión y tratamiento inicial, debido a la falta de información en las propiedades de algunos elementos; por tal motivo, se buscaron individualmente los datos de estos y se completaron sus propiedades. Es menester recalcar que, si bien es cierto que es importante contar con una base de datos, esto no es tan relevante respecto al objetivo principal de la investigación, puesto que se pretende desarrollar una metodología que pueda funcionar correctamente, independiente de la base de datos sobre la cual se desee aplicar esta metodología. Se lograron analizar 8 propiedades (electronegatividad, radio atómico, radio de van der Waals, primera energía de ionización, electroafinidad, punto de fusión, punto de ebullición y densidad) para un conjunto de 32 elementos (H, Li, C, N, O, F, Na, Mg, Si, P, S, Cl, K, Ni, Cu, Zn, Ga, As, Se, Br, Pd, Ag, Cd, In, Sn, Te, I, Pt, Au, Hg, Tl, Pb).

Se trabaja sobre el lenguaje de programación Python 3, por lo que son necesarias algunas librerías (como Scipy, Matplotlib, Pandas, Numpy, Pylab, Biopython, Json y Sklearn, ver Anexos) y con una laptop (equipo usado) que tiene las siguientes especificaciones.

- Nombre del sistema operativo: Microsoft Windows 10 Pro
- Versión: 10.0.19041 Build 19041
- Manufacturador del sistema: LENOVO
- Modelo del sistema: 20U1S18400
- SKU del sistema: LENOVO_MT_20U1_BU_Think_FM_ThinkPad L14 Gen 1
- Tipo del sistema: x64 basado en PC
- Procesador: Intel(R) Core(TM) i7-10510U CPU @1.80GHz, 2304 Mhz, 4 Core(s), 8 Logical Processor(s)
- BIOS (versión): LENOVO R17ET31W (1.14)
- RAM: 8GB

[I] Kaggle es una popular plataforma con excelentes recursos para aquellos que quieran aprender Machine Learning y ciencia de los datos. Kaggle dispone de decenas de miles de conjuntos de datos de todos los tipos y tamaños diferentes que puedes descargar y usar de manera gratuita.

[II] Github es una de las principales plataformas para crear proyectos abiertos de herramientas y aplicaciones, y se caracteriza sobre todo por sus funciones colaborativas que ayudan a que cualquier usuario pueda hacer su aporte, ya que estos proyectos pueden ser descargados y revisados libremente. La base de datos usada se encuentra disponible y puede ser visualizada en: https://github.com/phibrainDK/tesis

En primer lugar se necesitan obtener dendrogramas, para lo cual se hace uso de la librería "Scipy". Se consideran una lista de funciones de similitud *(ward, single, complete, average, weighted, centroid)* y una lista de métricas *(euclidean, cosine, correlation, hamming)*. Al ser cada función de similitud dependiente o no de una métrica (es decir, que cada función de similitud puede ser usada con un determinado conjunto de métricas, ver Anexos), para las listas seleccionadas, se filtran las funciones de similitud *ward* y *centroid* respecto a que si la métrica usada es o no *euclidean* (para cada función de similitud, no necesariamente es posible usar todas las metodologías, ver Figura 7). Los dendrogramas generados son guardados usando la función gcf() de pyplot y savefig() de Matplotlib.

Cada uno de los dendrogramas tiene que ser guardado apenas es generado, a fin de evitar que se sobrescriban. Iterando sobre todas las listas generadas se obtienen 18 dendrogramas, siendo que cada uno de ellos está guardado en formato de diccionario. Ahora se sabe que para generar los árboles consensus es necesario leer archivos con extensión *.xml* (esto gracias a la documentación de la librería Biopython) para luego convertirlos en formato *newick*; empero, como los dendrogramas son diccionarios, es necesario convertirlos al formato *.xml*, o *newick*.

Revisando la documentación de Biopython [III], se pudo notar que esta posee opciones de análisis de expresiones (parsing), las cuales permiten convertir archivos con determinadas extensiones a formato *newick*. Se sabe que se puede convertir desde el tipo de dato *string* a *newick* directamente con la función Phylo.read(). Por tanto, la idea es transformar los dendrogramas al formato *string*. Hasta este punto, una primera idea fue usar la librería Json para lograr tal objetivo, pero no se logró obtener una expresión necesaria que representa a un dendrograma. Una segunda idea, fue representar el dendrograma como un árbol usando una matriz de adyacencia. Para este motivo, se tuvo que revisar la documentación de Scipy, y se logró representar los dendrogramas como listas de adyacencia. Lo siguiente a desarrollar fue un algoritmo que sea capaz de obtener *strings* que representan la información de los dendrogramas a partir de estas listas de adyacencia. Esta representación también se conoce como notación *newick* de un dendrograma, y básicamente es obtener el recorrido *in - order* de las hojas del dendrograma en forma de string (ver Figura 6).

[III] Biopython es un conjunto de herramientas y librerías que usa la bioinformática, el cual esta basado principalmente en Python. Todos estas herramientas y librerias estan desarrolladas como proyectos que son de acceso libre y pueden ser visualizados desde su repositorio en Github: `https://github.com/biopythons`.
Toda la documentación de Biopython se encuentra en: `https://biopython.org/`

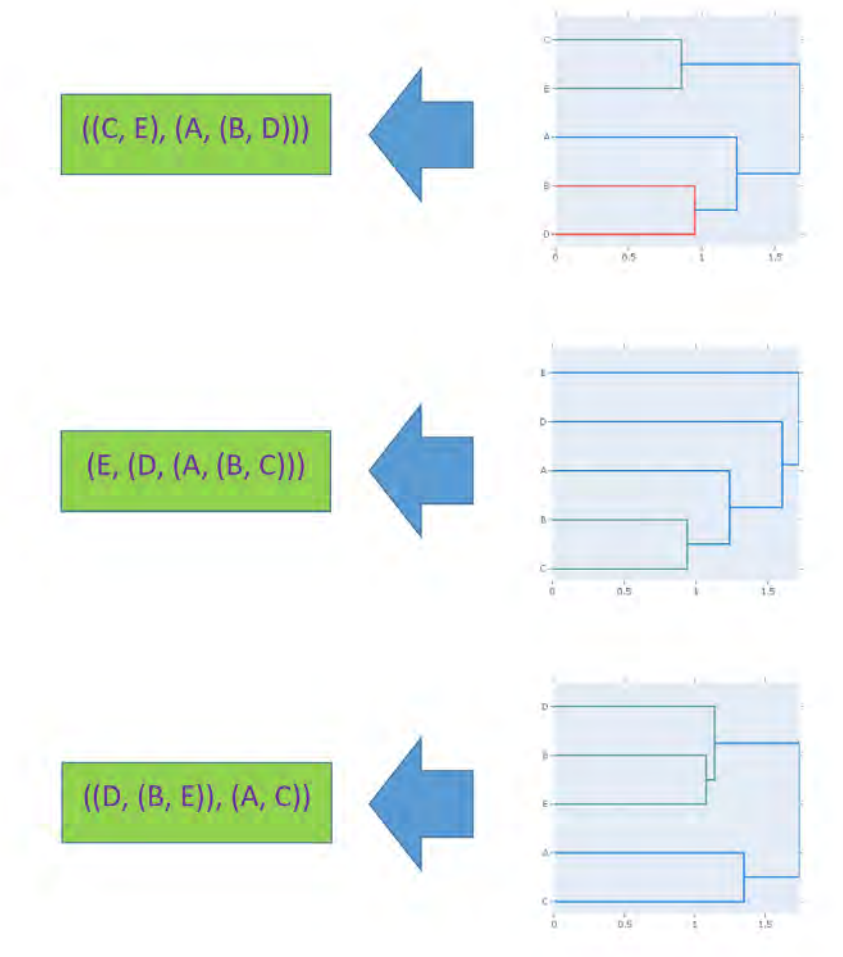

Figura 6: Transformación de un dendrograma a un string.

Se logró plantear un algoritmo que obtiene *strings* a partir de los árboles binarios (ver Anexos). La idea principal radica en usar el orden de las hojas de un árbol binario en un recorrido por profundidad (DFS: Depth First Search). Se guardaron los tamaños de cada uno de los subárboles, el respectivo nodo padre y la profundidad de cada vértice del árbol binario. Con esta información, se tiene en cuenta que cada clúster se forma con nodos de la misma profundidad, y por tanto iterando sobre las hojas en el orden del recorrido de profundidad, se analiza cada nodo y se reemplaza cada clúster con el nodo que lo representa. De esta manera se puede observar que el *string* obtenido representa al dendrograma.

Una vez que se logran obtener *strings* a partir de los dendrogramas, se hace uso de la librería Biopython

para convertir estos *strings* en árboles en formato *nexus*, y transformar estos al formato *newick*. Una vez que se tiene cada dendrograma en formato *newick*, se pueden obtener árboles consensus a partir de una lista de estos. Por lo tanto, cada dendrograma obtenido se agrega a una nueva lista, y a partir de esta, se logra obtener los árboles consensus (la obtención de los árboles consensus depende de una lista de árboles y un parámetro, este último se considera dependiendo de la metodología a usar). Un esquema que representa el procedimiento seguido para la obtención de los árboles consensus se muestra en la Figura 7.

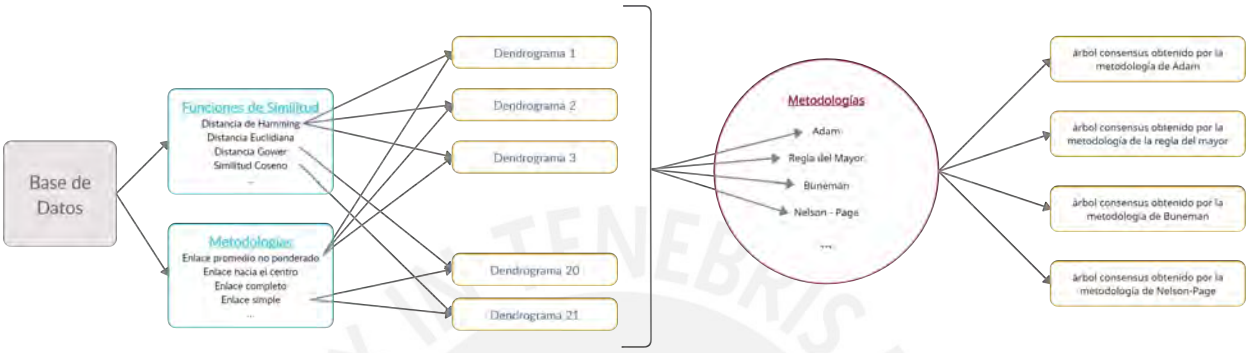

Figura 7: Representación del proceso de obtención de los árboles consensus.

Una vez obtenido los árboles consensus, fue necesario obtener el valor de n que maximiza el número de selección S_n para cada árbol consensus, para tal motivo es necesario recorrer el árbol consensus. En primer lugar, es importante notar que independientemente de la estrategia a usar para obtener el valor de S_n, es necesario guardar la información de los árboles consensus en listas de adyacencia. Esto es exactamente lo que se hizo con los dendrogramas, pero con la diferencia que se deben recorrer árboles consensus definidos dentro de la librería Biopython. Realizar este paso no es sencillo, puesto que los datos obtenidos de los árboles consensus están guardados en estructuras de tipo "Clade()", donde los nodos que no son hojas están identificados por el mismo nombre. Para poder realizar este paso, se tuvo que consultar a la documentación de Bio.Phylo.BaseTree, y en base a esta se logró determinar la raíz del árbol consensus.

Una vez obtenida la raíz, se hace uso de funciones definidas dentro la estructura "Clade()". Con el uso de estas funciones, se notó que existe más de una manera de poder guardar la información en listas de adyacencia. Cada una de estas está directamente implicada con la resolución de un problema de grafos; es decir, en base a la "función distancia entre nodos", es posible construir el árbol consensus, pero el algoritmo constructivo que se planteó para resolver este problema tiene una complejidad de $O(n^2)$, puesto que se tienen que consultar dicha función para todas las posibles combinaciones de pares de hojas. Otra manera es usando la función "ancestro común con menor profundidad", para el cual se planteó un algoritmo con complejidad $O(n^2)$. Se resolvió uno de estos problemas y de tal modo se guardó la información de los árboles consensus en listas de adyacencia.

En base a las listas de adyacencia, se procedió a plantear un algoritmo que nos permita obtener todos los

valores posibles S_n. El algoritmo planteado está basado en guardar la información del tamaño del subárbol de cada nodo y el nodo padre (ver Anexos). Se hace una iteración sobre todos los nodos, y solo se quedan aquellos que corresponden a los $n-$ subárboles maximales, dando así una complejidad total de $O(n^2)$. Algo importante que se logró notar es que existían más de un valor de n que maximiza S_n. Para un determinado árbol consensus, se hallaron todos estos posibles valores.

En base a todo lo desarrollado hasta el momento, se pueden determinar y generar bases topológicas de los árboles consensus obtenidos a partir de un grupo de metodologías y funciones de similitud para una determinada base de datos. Ahora, lo que se plantea resolver es determinar una metodología que nos permita evaluar la dependencia de los resultados respecto a las propiedades involucradas. Esta metodología debe cumplir con los siguientes requisitos:

- Conservar en la medida de lo posible la información que nos brinda la base de datos total, es decir, aquella de donde se consideran todas las propiedades.
- Conservar las relaciones de similitud que se generan cuando se usa la base de datos total.
- Conservar las relaciones de disimilitud que se obtienen cuando se trabaja con la base de datos total.
- Usar el mínimo número posible de propiedades de tal manera que se cumplan todos los puntos anteriores.

Suponiendo que de alguna manera podemos conservar las relaciones de similitud y disimilitud, entonces esta situación se transforma a un problema de reducción de dimensionalidad [24]. Una primera idea sobre determinar este posible número es usando PCA (Análisis de Componentes Principales). Si bien es cierto, PCA reduce la dimensionalidad de la base de datos, los componentes que representan ya no son parte del conjunto de propiedades que se tenía inicialmente. Esto es debido a que PCA halla los valores propios y vectores propios de la matriz de covarianza generada por la base de datos [25]. La matriz de covarianza es una matriz cuadrada que resulta del producto de la matriz de base de datos inicial y su transpuesta. Por lo tanto, es claro notar que cada una de las propiedades fueron transformadas, pues ahora son una combinación lineal del conjunto inicial de propiedades. Como se puede apreciar, estas nuevas propiedades o "componentes principales" ya no representan a ningún subconjunto del conjunto de propiedades que se tenía inicialmente en la base de datos, por lo que esto no es lo que se pretende hallar. En cuanto a lo planteado hasta el momento, aquellas técnicas que usan la esencia o trasfondo matemático de PCA para resolver el problema de "reducción de dimensionalidad" no son viables.

Consultando la literatura sobre cómo resolver este problema, se logra encontrar que existen metodologías basadas en redes neuronales y machine learning, algunas de las cuales se pueden desarrollar usando las librerías SKlearn o Tensorflow. Todas estas metodologías se basan en particionar la base de datos en dos subconjuntos, uno de entrenamiento y el otro de prueba. Se escoge la metodología "Random Forest" de Sklearn y se obtienen resultados. A priori, fijando todos los parámetros que usa esta metodología y variando la cantidad de árboles de decisión a usar, se logró notar algo inesperado. Cada vez que se variaba la cantidad

de árboles de decisión, los resultados obtenidos variaron demasiado, tanto en las propiedades obtenidas en sí, como también en la cantidad de propiedades.

La idea detrás de estas metodologías está en particionar la base de datos, puesto que hacen que la red neuronal interna entrene con algunos de ellos, y que luego evalúe el resto. Por lo tanto se necesita tener una columna de objetivo o "target". La columna target que se usaba como referencia eran las etiquetas o número atómico. Estos datos son únicos, es decir solo se presentan en la base de datos una vez por elemento, por tanto hacer de algún modelo y luego tratar de predecir resultados en base a un "target" no tienen mucho sentido, puesto que cada dato de entrenamiento y evaluación siempre retornará un valor distinto. De esta manera, se puede entender que el uso de estas metodologías tampoco es viable.

La información que brinda la base de datos sobre las relaciones de similitud y disimilitud se reduce a la información que puede brindar la estructura interna de los árboles consensus o información que brinda un conjunto de clústers, si hallamos las bases topológicas (generar las particiones). Por lo tanto, el problema se reduce a determinar una metodología que mantenga invariante parte de la estructura interna de un conjunto de árboles consensus dado, o a estudiar la invarianza de los clústers que se forman a partir de las bases topológicas de cada árbol consensus. Ambos problemas pueden estudiarse desde el punto de vista matemático, pero conllevan tener conocimiento de conceptos matemáticos avanzados sobre teoría de matrices y conjuntos [26], por lo que tratar de usar alguna de estas metodologías no es viable.

Sobre todo lo abordado hasta el momento, se puede notar que no se puede estudiar el problema usando alguna técnica o estudio conocido, lo que conlleva a tratar de plantear una metodología nueva que permita estudiar el problema planteado. Para ello, es importante tener en cuenta porque las metodologías que se intentaron plantear no son útiles. Estas se pueden resumir en:

1 Sobre la reducción de dimensionalidad, las técnicas relacionadas con el fundamento matemático de PCA no son válidas, puesto que cada componente principal es una combinación lineal de las propiedades iniciales de cada elemento en la base de datos.

2 Sobre la reducción de dimensionalidad, las técnicas de machine learning que usan un modelo de redes neuronales no son viables, puesto que el conjunto de entrenamiento y el de evaluación corresponden a un "target" único. De esta manera, tratar de usar alguna metodología no tiene sentido, puesto que las propiedades que se obtienen son muy variables (distintas listas de propiedades variando un parámetro).

3 Tener en cuenta que incluso si se tiene una técnica que pueda reducir la dimensionalidad, es necesario plantear una metodología que permita mantener las relaciones de similitud y disimilitud de la base de datos total.

4 Mantener la similaridad y disimilaridad de una base de datos puede reducirse a determinar estructuras

invariantes de una lista de árboles o determinar clústers invariantes dado una lista de particiones de un mismo conjunto. Estos estudios requieren conocimiento de conceptos matemáticos avanzados.

Sobre el punto 2, las metodologías que engloban el uso de redes neuronales, no dan buenos resultados, puesto que no se tiene un "target". Es necesario crear este "target" para poder obtener resultados. Sobre el punto 3 y 4, una manera posible de cuantificar la similaridad y disimilaridad es en base al número de partición y al $n-$ maximal, respectivamente. Si usamos estos valores como "target", estamos manteniendo la similaridad y disimilaridad, y además estamos resolviendo el problema de reducción de dimensionalidad, puesto que ya tenemos un "target" definido. Según esto, se puede resolver el problema haciendo uso de redes neuronales. Lo único que falta es determinar cómo considerar los "elementos" de la base de datos, ya que de nuestra base de datos inicial, solo se obtienen dos valores, S_n y n.

La idea principal para resolver este problema, es considerar el espacio de base de datos que se origina a partir de la base de datos inicial. La idea es considerar un subconjunto de elementos y un subconjunto de propiedades y esta nueva matriz de datos, forma parte de un elemento que pertenece a lo que denominaremos "espacio de base de datos". Cada elemento que pertenece al espacio de base de datos, corresponde a un valor de S_n y n determinado. Es decir, se tiene una transformación $f : \mathbb{R}^{mxn} \to \mathbb{R}^2$.

Las transformaciones obtenidas tienen que ser comparables para poder considerarlas dentro de un mismo conjunto de elementos. Por lo que ahora, es necesario considerar que: elementos comparables son aquellos que contienen el mismo conjunto de elementos químicos. Ahora que tenemos un "target", nos falta la nueva base de datos. A priori, se puede considerar que esta nueva base de datos son los elementos que pertenecen a un mismo conjunto en el "espacio de base de datos", pero estos elementos tienen un problema: las propiedades que los caracterizan tienen que ser redefinidas, puesto que ahora ya no son constantes para todos estos nuevos elementos. Una idea para poder resolver este problema parcialmente es usar PCA, puesto que podemos reducir la cantidad de propiedades de un "nuevo elemento". La razón por la que no se logra resolver con totalidad el problema usando esta idea radica en que existe un número k, tal que si nuestro "nuevo elemento" tiene menos de k propiedades, este "nuevo elemento" ya no es comparable con los otros elementos del conjunto al que pertenece. Usando esta idea con $k = 4$ y teniendo en cuenta que el costo computacional de los árboles consensus es viable, se planteó un modelo de "machine learning", específicamente el "perceptrón multicapas" y se evaluaron cada uno de los parámetros de esta red neuronal en base a los resultados reales obtenidos para las metodologías de "Adams" y "regla del mayor" (ver Anexos). Los resultados obtenidos se discutirán en la siguiente sección.

Es posible plantear otra metodología basada en considerar a los elementos del espacio de base de datos y sus transformaciones. La idea es generar todos los subconjuntos posibles y evaluar cada una de sus transformaciones fijando un determinado porcentaje de error respecto al valor óptimo posible (aquel que se obtiene a partir de considerar todas las propiedades para el determinado subconjunto). Es claro que esto tomaría un

costo computacional elevado, puesto que si existen m propiedades y n elementos en la base de datos inicial, entonces nuestro espacio de base de datos contiene 2^{m+n} elementos, los cuales forman 2^m conjuntos. Se debe notar que, incluso para la base de datos con la que se cuenta, esto no es computacionalmente realizable.

Un subconjunto en el espacio de base de datos es aquel que posee elementos comparables; es decir, elementos que poseen el mismo grupo de elementos químicos (ver Figura 8). No es necesario estudiar todos los subconjuntos del espacio de base de datos, sino que se puede analizar un subconjunto determinado. Haciendo uso de esta información, es importante notar que todavía la cantidad de elementos que puede tener nuestros conjuntos es demasiado grande (2^m), si se considera utilizar esta idea sobre los estudios de análisis jerárquico de clústers, en donde se consideraban 128 propiedades [9] o 4700 propiedades [13], la metodología propuesta no es factible computacionalmente.

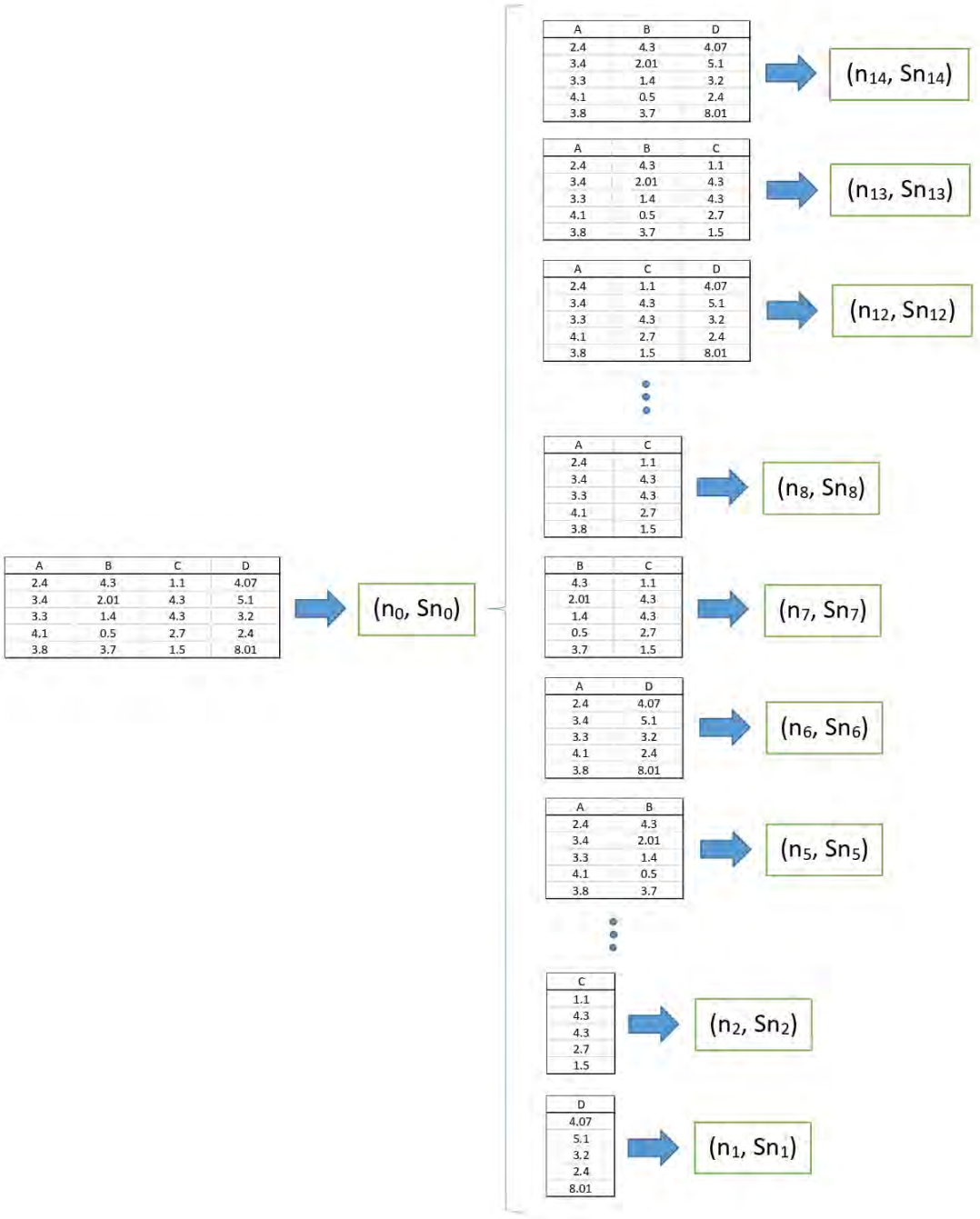

Figura 8: Subconjunto de elementos comparables a partir de una base de datos inicial.

Para reducir la carga (esfuerzo) computacional, es posible aplicar alguna heurística del tipo "selección de subconjuntos", los cuales se enfocan en buscar maneras óptimas de seleccionar subconjuntos de un con-

junto dado, de tal manera que estos sean representativos y nos den información suficiente para tener una idea general de cómo está distribuida nuestra información. A priori, no es posible determinar qué heurística se acopla mejor para nuestro propósito, por lo que para tener una idea, es necesario hacer estudios de la metodología propuesta en base a heurísticas. Se debe recalcar que cada una de estas heurísticas se tiene que comparar respecto a los datos que se obtienen usando todos los elementos del espacio de base de datos.

En adición, respecto al trabajo de P. H. A. Sneath [5], en sus resultados, este muestra un valor al que denomina "dimensión efectiva", el cual según [27] se calcula en base a los datos obtenidos por el PCA (inversa de la sumatoria de los cuadrados de los eigenvalores de la matriz de covarianza). El valor reportado por Sneath fue de 8.46 para una cantidad de 6 componentes principales. Esto indica que existen dependencias entre las propiedades que usó en sus cálculos. Este resultado impresionó a Sneath, respecto a lo cual, mencionó que posiblemente esto se debe a que muchas de las propiedades consideradas están muy relacionadas con las últimas 2 capas de electrones de cada elemento, y por tanto no es de extrañarse que las propiedades guardan correlaciones entre ellas. También se calcula este valor en base a la librería SKlearn y los resultados obtenidos se discutirán en la siguiente sección.

4. Resultados y discusión

La eficiencia de la obtención de resultados (el tiempo de ejecución del programa desarrollado) está directamente relacionada con la complejidad algorítmica que presenta. El programa desarrollado tarda aproximadamente 23.4 minutos, por lo que es necesario analizar su complejidad y en adición discutir acerca de las posibles optimizaciones que se pueden realizar sobre las ya realizadas.

4.1. Análisis de complejidad

Para hacer cálculos sobre la complejidad total del programa desarrollado, se hace uso de la "notación de Landau - Bachmann" [IV]. Para poder analizar la complejidad, es menester fijar algunas variables. Definamos el número de metodologías como p, el número de funciones de similitud q, la cantidad de elementos a analizar n y la cantidad de propiedades usadas como m.

Primero analicemos la complejidad que toma determinar y construir cada dendrograma. La distancia inicial entre 2 elementos se calcula en complejidad $O(m)$. La cantidad de relaciones que se tienen que calcular al inicio son $0{,}5 * (n^2 - n)$; por lo tanto, la construcción de la matriz de distancias inicial tiene un costo de $O(n^2 m)$. Ahora, cada agrupamiento está dado por la relación de "Lance Williams". De esta manera, se puede considerar que actualizar un elemento de la matriz tiene un costo de $O(1)$. Se tienen que realizar n iteraciones para determinar el dendrograma. En cada iteración es necesario actualizar $2 * n$ elementos de la matriz y, en adición, se tiene que encontrar el mínimo valor de la matriz. Encontrar el valor mínimo de la matriz puede ser calculado usando una estructura de datos tipo "árbol binario balanceado de búsqueda". Esto podría optimizar la búsqueda del mínimo valor hasta $O(log(n^2)) = O(2 * log(n)) \sim O(log(n))$. Si no es el caso, se puede encontrar el mínimo en complejidad $O(n^2)$. Algo importante que resaltar es que, si se usa un "árbol binario balanceado de búsqueda", en cada iteración, se insertan/remueven n elementos, por tanto, la complejidad resultante es $O(n^2 log(n))$. Caso contrario, no importan los valores que se insertan o remueven de la matriz, puesto que se busca el valor mínimo iterando sobre todos los valores de matriz, lo que hace una complejidad de $O(n^3)$. De esta manera, se puede concluir que el costo computacional de cada dendrograma pertenece al intervalo $[O(n^2 m + n^2 log(n)), O(n^2 m + n^3)] = [O(n^2(m + log(n))), O(n^2(m + n))]$.

Ahora, analizamos la complejidad que toma determinar y construir cada árbol consensus. Para obtener estos, es necesario tener una lista de dendrogramas. Considerando el supremo del costo computacional de un dendrograma, esta lista tiene un costo computacional de $O(n^2(m + n)pq)$. Ahora, representar cada dendrograma como una lista de adyacencia tiene un costo de $O(n)$. Transformar estos dendrogramas a *strings* tiene una complejidad de $O(n)$. Cada *string* es transformado a formato *nexus* y estos a *newick*, cada una de

[IV] Es común usar la notación de Landau - Bachmann en ciencias de la computación para acotar de manera asintótica el crecimiento de un tiempo de ejecución a que esté dentro de factores constantes por arriba y por abajo, es decir:

Sean f y g, funciones definidas en los $\mathbb{R}$, decimos que $f(x) = O(g(x))$, si existen enteros positivos a y b, tal que $f(n) = b * g(n), \forall b \leq a$ [28]

estas transformaciones tiene un costo de $O(n)$. Ahora, dependiendo de qué tipo de árbol consensus se quiere determinar, se tienen diferentes complejidades. Se usaron las metodologías de "Adams" y "regla del mayor" para obtener árboles consensus. La implementación de estos algoritmos en python poseen una complejidad de $O(pqn^2)$ y $O(p^2 * q^2 * n^2)$, respectivamente, usando la librería Phylo. De esta manera, la complejidad total de la obtención de los árboles consensus tiene un costo computacional de $O(n^2pq(pq + m + n))$.

Cada árbol consensus se guarda en forma de listas de adyacencia para poder obtener topologías. Esto tiene un costo de $O(n)$. La determinación de número de partición tiene una complejidad de $O(n^2)$, puesto que se analizan todas las posibles topologías inducidas por el árbol consensus. Por lo tanto, calcular el número de partición, para una base de datos de n elementos, con m propiedades cada uno, usando p metodologías y q funciones de similitud tiene un costo de $O(n^2pq(pq + n + m) + n^2) = O(n^2pq(pq + n + m))$.

Determinar los elementos del espacio de base de datos tiene una complejidad de $O(2^{m+n}n^2pq(pq+n+m)$. Como se mencionó anteriormente, solo se analizó un elemento del espacio de base de datos. Esto tiene un costo de $O(2^m n^2pq(pq + n + m))$. Para la base de datos de prueba, $n = 32, m = 8, p = 5, q = 4$, lo cual es $\sim 7 * 10^8$ operaciones, lo cual en Python3 toma alrededor de 23.4 minutos.

Una vez calculada la complejidad total de los algoritmos planteados, es necesario analizarlos y poder evaluar si es viable o no realizar algunas optimizaciones.

- Implementar la obtención de los dendrogramas usando un "árbol binario balanceado de búsqueda". De esta manera los dendrogramas pueden obtenerse en complejidad $O(n^2(m + log(n))$.
- Se puede notar que las funciones de transformación de formatos (dendrograma - string, string - nexus, nexus - newick) tienen complejidad $O(n)$, lo cual no es comparable con el cálculo de los árboles consensus $O(n^2p^2q^2)$.
- La obtención de los números de partición tienen un costo de $O(n^2)$ una vez obtenidos estos árboles consensus, lo que nuevamente no es comparable con el costo de obtener los árboles consensus. Por lo tanto es necesario tratar de optimizar la obtención de los árboles consensus.
- Una idea para poder disminuir esta complejidad es implementar estos algoritmos sin usar las librerías que las invocan, ya que es muy difícil modificar los algoritmos que se encuentran propuestos en estas. Una mejor optimización para la obtención de los árboles consensus se encuentra en [29, 30], donde logran plantear algoritmos que son capaces de mejorar la complejidad hasta $O(npq)$ en base al algoritmo de "Day", para el caso de la metodología de la regla del mayor, y una complejidad de $O(pqnlog(n))$, en base a algoritmos de "Centroid Decomposition", "Heavy-Light Decomposition", "Segment Tree" y "Wavelet Tree", para el caso de la metodología de Adams.

4.2. Sobre el número de propiedades y su variación en los árboles consensus

Para poder evaluar qué resultados se obtienen con la metodología propuesta, es menester usar una base de datos y evaluar los resultados obtenidos. Si bien la base de datos evaluada (desde Kaggle) proporcionó la mayor parte de la información usada, faltó información acerca de las propiedades de algunos de los elementos. Por lo que se esperaría que estos datos no conducirían a buenas relaciones de similitud-disimilitud. Esto se puede observar en los árboles consensus obtenidos (ver Figuras 9 y 10). Por ejemplo, el método conduce a agrupar a elementos como As, K, Ni y C, que no guardan gran similitud entre sus propiedades. Si bien no se muestran buenos resultados, estos sirven para poder evaluar parcialmente la metodología propuesta. Por otro lado, una mejor base de datos mostraría mejores resultados (esto es, una base de datos con más elementos que guarden similitud entre sí y más propiedades que los describan), pero el costo computacional sería un problema, recordando que la complejidad computacional es de $O(2^m n^2 pq(pq+n+m))$, y que se tendría que evaluar en función de parámetros tales como n (número de elementos), m (número de propiedades), etc.

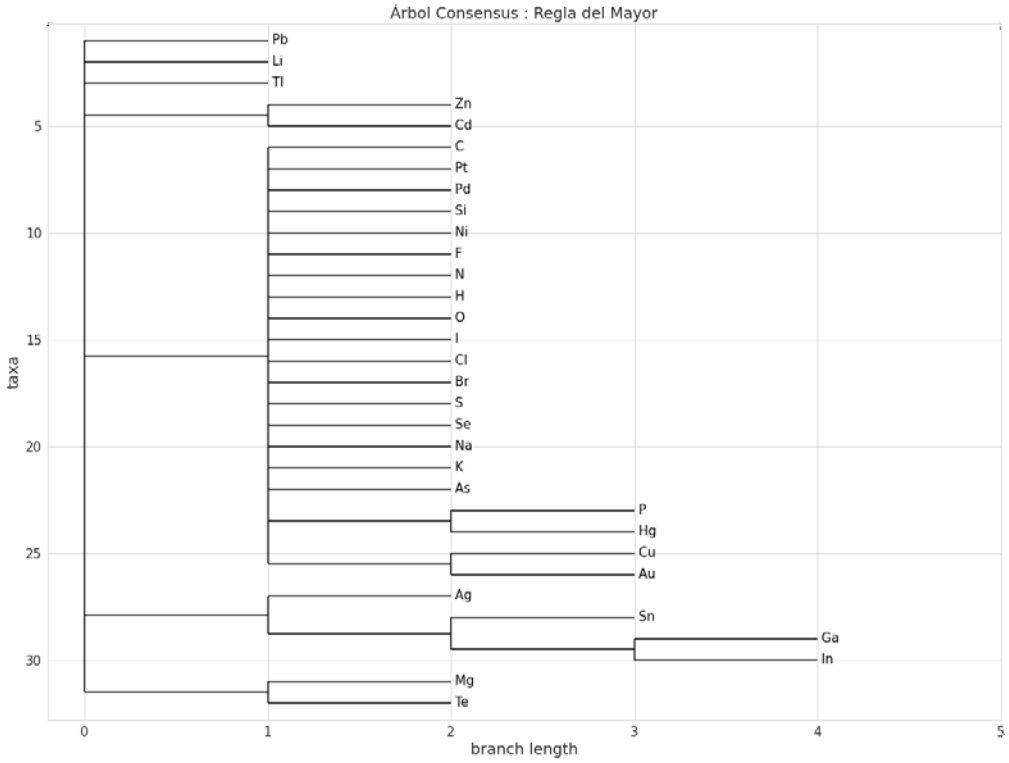

Figura 9: Árbol consensus obtenido en base a la metodología "regla del mayor".

En la Figura 9 se muestra el esquema de un árbol consensus mediante la aplicación de la "regla del mayor". En esta metodología es necesario un parámetro entre 0 y 1, el cual indica qué fracción de la lista de árboles contiene al clúster a analizar. El valor usado para la evaluación del método fue de 0.5.

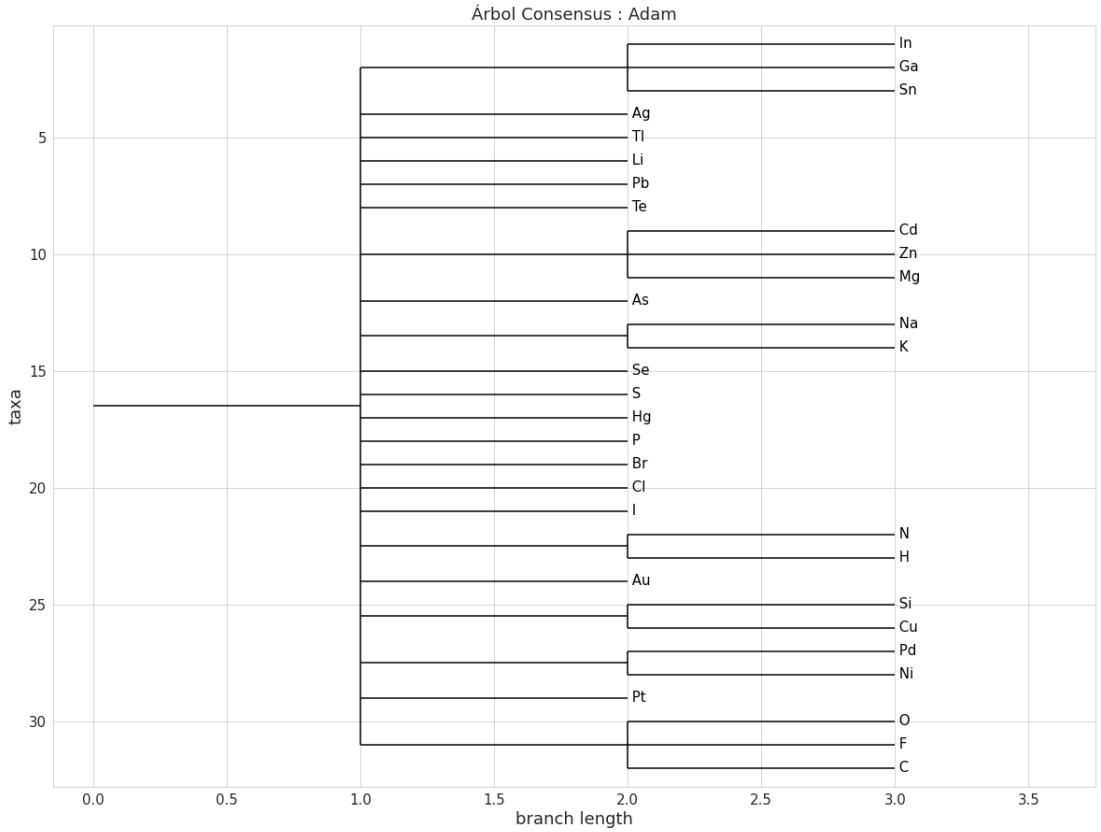

Figura 10: Árbol consensus obtenido en base a la metodología "Adams".

En la Figura 10 se muestra el esquema de un árbol consensus mediante la aplicación de la metodología de "Adams". En este esquema se puede apreciar una mejor distribución de las similaridades respecto a aquella mostrada en la Figura 9, y que corresponde a la metodología de la "regla del mayor".

Como se mencionó con anterioridad, para evaluar el método, es necesario obtener los valores de n y Sn de cada elemento de un subconjunto que pertenece al espacio de base de datos. El subconjunto escogido, es aquel que posee el mismo conjunto de elementos que la base de datos inicial. Se generaron todos los posibles valores de n y Sn para cada elemento. Luego, se compararon estos valores con los que corresponden a los valores de los árboles consensus obtenidos. Se escogieron aquellos valores que guardan un 99 % de similitud para el valor del número de partición y un valor de n dentro de los posibles valores para este número de partición. Se debe recordar que para cada árbol consensus se obtiene que existen muchos valores de n que maximizan el número de partición. Sobre todos los valores que cumplen esta condición, se escogieron aquellos que poseen el mínimo número de propiedades. Los valores obtenidos fueron usando el conjunto de elementos ["electronegatividad",

"radio atómico", "radio de van der Waals", "primera energía de ionización", "electroafinidad","punto de fusión" , "punto de ebullición", "densidad"]. Este conjunto se representa como [1, 2, 3, 4, 5, 6, 7, 8] para fines prácticos y estos resultados se pueden apreciar en las Tablas 4 y 5.

Tabla 4: Resultados obtenidos según la metodología "regla del mayor" considerando un 99 % de similaridad.

% de similaridad	subconjunto de propiedades
100.0	[1, 2, 3, 4, 5, 6, 7, 8]
99.869	[1, 2, 3, 5, 6, 7]
99.869	[1, 6, 7, 8]
99.869	[2, 3, 4, 5, 6]
99.869	[1, 4, 7]
99.869	[1, 4, 7, 8]
99.869	[1, 7]
99.869	[2, 4, 7]
99.869	[2, 4, 7, 8]
99.869	[4, 7, 8]
99.74	[1, 2, 3, 5, 7, 8]
99.734	[1, 3]
99.734	[2, 3, 7, 8]
99.734	[3, 7, 8]
99.598	[2, 3, 4, 5, 6, 8]
99.598	[2, 4, 6]
99.487	[1, 3, 5, 6, 8]
99.487	[1, 4, 6]
99.487	[3, 5, 6]
99.487	[2, 5, 6]
99.24	[2, 3, 5, 8]
99.036	[1, 2, 4, 5, 8]
99.036	[2, 4, 5, 8]
98.891	[5, 6, 7]
98.725	[2, 4, 6, 8]
98.594	[1, 6, 7]

Tabla 5: Resultados obtenidos según la metodología "Adams" considerando un 99 % de similaridad

% de similaridad	subconjunto de propiedades
100.0	[1, 2, 3, 4, 5, 6, 7, 8]
100.0	[1, 2, 5, 6]
100.0	[4, 5, 8]
99.89	[1, 2, 3, 7]
99.89	[4, 5, 6]
99.49	[1, 2, 4, 5, 6, 7, 8]
99.49	[2, 4, 5, 6, 7, 8]
99.362	[3, 4]
99.193	[1, 2, 3, 4, 5, 6]
99.193	[1, 3, 4, 5]

De esta manera, se puede concluir que el número mínimo de propiedades es de dos considerando árboles consensus obtenidos por la regla del mayor (ver Tabla 4), y de dos para los árboles consensus obtenidos por la metodología de Adams (ver Tabla 5). Ambos valores son obtenidos respecto a la base datos evaluada. Estos resultados muestran que las propiedades usadas guardan correlación entre sí, lo cual no es de extrañar (Sneath también llegó a un resultado similar). En adición, el valor de la "dimensión efectiva" obtenido fue de 2.967 para un número de componentes principales de 3. Esto confirma que las propiedades usadas guardan correlación entre ellas.

Por otro lado, sobre el costo computacional de la metodología propuesta, como ya se mencionó con anterioridad, la metodología propuesta presenta un costo computacional alto, motivo por el cual se planteó una metodología alternativa basada en machine learning, que usa los datos ya obtenidos y evalúa el poder predictivo del modelo del perceptrón multicapas para predecir estos resultados. La importancia del uso de redes neuronales radica en que podemos usar estas para poder predecir los valores de n y Sn usando el modelo del perceptrón multicapas, entrenando la red neuronal y evaluando el poder predictivo al que se puede llegar. Para poder evaluar estos datos, se fija un valor de k, tal que cualquier elemento del subconjunto a evaluar que posee menos de k no es considerado. Para el resto, se usa PCA para poder reducir la cantidad de propiedades de cada uno de estos elementos a k, y así se obtienen matrices de igual dimensión. Se hace una transformación de estas matrices a un vector y se usan estos vectores como base de datos. La cantidad de elementos generados fueron 169, con 164 propiedades. Se entrena el modelo usando el 60 % de la base de datos generada como conjunto de entrenamiento y el resto como conjunto de prueba. Los parámetros que se usaron fueron:

- Se usaron 3 capas ocultas de 700 nodos cada una.
- La función de activación usada fue "relu".

- La cantidad máxima de iteraciones usada fue de 100000, en caso el modelo no llegue a converger.
- El método para la optimización de pesos usado fue "lbfgs" (familia de métodos cuasi Newton).
- La velocidad de aprendizaje inicial usado fue de 0.01.
- El valor del número de inicialización del generador interno aleatorio fue 2 (random state).

Los valores obtenidos para el error medio absoluto, error cuadrático medio y el coeficiente de determinación (R^2) fueron de 0.2, 0.1 y 0.93 respectivamente usando los árboles consensus obtenidos por la metodología de la regla del mayor, mientras que según la metodología de Adams fueron de 0.15, 0.08 y 0.95 en ese orden correspondientemente. Esto muestra que el modelo de machine learning muestra buenos resultados y este puede ser usado para predecir los valores de n y Sn, reduciendo así considerablemente el costo computacional de la metodología propuesta. Notar que en base a esta red neuronal, ahora solo es necesario transformar los elementos del subconjunto a analizar usando PCA, y una vez entrenada la red neuronal, usar esta para poder predecir los valores de Sn y Sn. Cada uno de estos valores predichos en base al modelo de machine learning son comparados con el valor base y se consideran solo aquellos que guardan un 99 % de similaridad. Seguidamente, se seleccionan aquellos que poseen una mínima cantidad de propiedades. Esto puede ser usado cuando la base de datos inicial sea más completa.

Se debe notar que luego de usar el PCA sobre cada elemento del subconjunto a analizar, los valores de las propiedades iniciales fueron transformados, por este motivo, se tiene que guardar cada elemento como el subconjunto de propiedades que este presenta inicialmente, seguido por la transformación de reducción de dimensionalidad. Una vez que se predicen los valores usando el perceptrón multicapas, estos se relacionan con el subconjunto de propiedades que representaban inicialmente a este elemento. Es a partir de este subconjunto de propiedades que se evalúa el objetivo de la presente investigación. Un esquema que representa este procedimiento usando un valor de $k = 3$ para la reducción de dimensionalidad de cada elemento que posee más de 3 propiedades se muestra en la Figura 11.

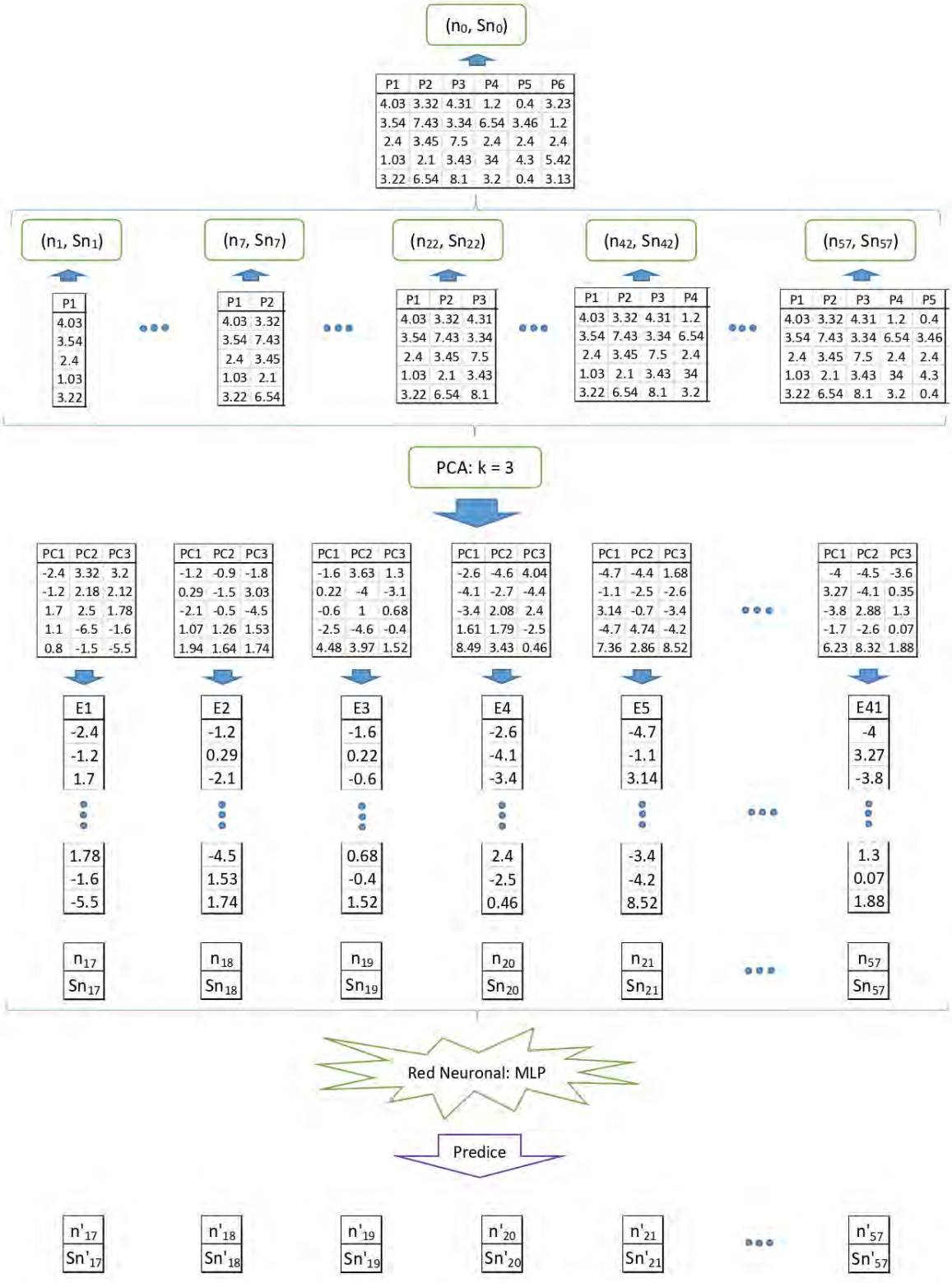

Figura 11: Esquema que representando a la metodología seguida para predecir los valores de n y Sn usando el modelo del perceptrón multicapas y un valor de $k = 3$ para la reducción de dimensionalidad.

Para poder hacer uso de la red neuronal, es necesario generar un determinado número de elementos para el entrenamiento. Estos datos son generados en base a la metodología, lo cual puede conllevar a tiempos de ejecución elevados si la base de datos inicial presenta más elementos y más propiedades. Para tales casos, es necesario el uso de heurísticas que nos ayuden a seleccionar elementos representativos de cada subconjunto a analizar. Determinar la mejor heurística consiste en comparar los resultados obtenidos por las distintas heurísticas respecto a los resultados reales obtenidos con la metodología propuesta. Dicho estudio y análisis no fue abordado en el presente trabajo de investigación.

5. Conclusiones

En el presente trabajo se presentó una metodología que logra estudiar la variación de los árboles consensus respecto al subconjunto de propiedades que se considera para su construcción. La metodología se basa en estudiar subconjuntos del "espacio de base de datos" y hacer transformaciones sobre cada uno de los elementos del subconjunto.

La metodología descrita presenta una complejidad que depende del costo computacional que toma la construcción de los árboles consensus. Si bien en el presente trabajo no se implementaron los algoritmos más eficientes para así reducir el costo computacional, se presentó una manera alternativa para poder evaluar de manera más eficiente las transformaciones de cada elemento de los subconjuntos del espacio de base de datos. La estrategia propuesta se basa en el uso del modelo perceptrón multicapas y en encontrar los mejores parámetros que se deben considerar para poder predecir los valores correspondientes a las transformaciones. Para este propósito, se transformó cada elemento seleccionando un $k = 4$ y usando PCA. Los datos son transformados a vectores y luego estos forman nuevos elementos de entrada para la red neuronal. Los resultados obtenidos muestran un coeficiente de determinación (R^2) de 0.93 en el peor de los casos, lo que muestra que la estrategia propuesta es viable.

Los resultados obtenidos muestran que las propiedades usadas guardan correlaciones entre sí, ya sea evaluando el número de propiedades considerando solo aquellos valores que guardan un 99 % de similaridad, o calculado el valor de la dimensión efectiva en base a PCA. Finalmente, agregar que la metodología puede ser evaluada de manera más eficiente si se hacen uso de heurísticas del tipo selección de subconjuntos. El estudio y evaluación del uso de heurísticas no se abordaron en el presente trabajo de investigación.

6. Referencias bibliográficas

[1] W. H. E. Schwarz. Theoretical basis and correct explanation of the periodic system: Review and update. Journal of Chemical Education, 87:435–443, 2010.

[2] W. Leal and G. Restrepo. Formal structure of periodic system of elements. Proc. R. Soc. A, 87(475):1–34, 2019.

[3] D. Robert and R. Carbó-Dorca. On the extension of quantum similarity to atomic nuclei: Nuclear quantum similarity. J. Math. Chem., 23:317–351, 1998.

[4] D. Robert and R. Carbó-Dorca. General trends in atomic and nuclear quantum similarity measures. Int. J. Quantum Chem., 77:685–692, 2000.

[5] P. H. A. Sneath. Numerical classification of the chemical elements and its relation to the periodic system. Foundations of Chemistry, 2:237–263, 2004.

[6] T. E. Karakasidis and D. N. Georgiou. Partitioning elements of the periodic table via fuzzy clustering technique. Soft Computing, 8:231–236, 2004.

[7] H. Mesa, E. J. Llanos, J. L. Villaveces and G. Restrepo. Topological study of the periodic system. J. Chem. Inf. Comput. Sci., 44:68–75, 2004.

[8] G. Restrepo and H. Mesa. Chemotopology: Beyond neighbourhoods. Current Computer-Aided Drug Design, 7:90–97, 2011.

[9] H. Mesa and G. Restrepo. Topological space of the chemical elements and its properties. Journal of Mathematical Chemistry, 39(2):401–416, 2006.

[10] G. Restrepo, H. Mesa and J. L. Villaveces. On the topological sense of chemical sets. Journal of Mathematical Chemistry, 39(2):363–376, 2006.

[11] D. Z. Chen. A new method for studying the periodic system based on a kohonen neural network. Journal of Chemical Education, 87(4):433–434, 2010.

[12] G. Restrepo. Building classes of similar chemical elements from binary compounds and their stoichiometries. ACS Symposium Series, pages 95–110, 2017.

[13] W. Leal, G. Restrepo and A. Bernal. A network study of chemical elements: From binary compounds to chemical trends. MATCH Commun. Math. Comput. Chem., 68:417–442, 2012.

[14] G. J. Klir and B. Yuan. Fuzzy Sets and Fuzzy Logic: theory and aplications. Prentice Hall PTR., 1^{fst} edition, 1995.

[15] H. Hashimoto. Canonical form of a transitive fuzzy matrix. Science Direct, 11:157–162, 1983.

[16] D. J. Matich. Redes neuronales: Conceptos básicos y aplicaciones. Catedrá de infórmatica a la Ingeniería de procesis - Orientación I, pages 1–55, 2001.

[17] C. Brauer. An introduction to self-organizing maps: Proseminar articial intelligence. Hamburg University, Department of Informatics, pages 1–16, 2012.

[18] Matthias O. Chemometrics: Statistics and Computer Application in Analytical Chemistry. Wiley-VCH, 3^{th} edition, 2017.

[19] F. R. McMorris, B. Mirkin, M. F. Janowitz, F. J. Lapointe and F. S. Roberts. Bioconsensus: DIMACS Working Group Meetings on Bioconsensus. American Mathematical Soc., 61 edition, 2000 - 2001.

[20] G. Restrepo and J. L. Villaveces. From trees (dendrograms and consensus trees) to topology. Croatica Chemica Acta, 78:275–281, 2005.

[21] H. Mesa and G. Restrepo. On dendrograms and topologies. MATCH Commun. Math. Comput. Chem, 60:371–384, 2008.

[22] E. Scerri and G. Restrepo. Mendeleev to Organesson: A multidisciplinary Perspective on the Periodic Table. Oxford University Press, 1^{fst} edition, 2018.

[23] G. Q. Chen, Z. X. Fan, X. Z. Zhou, K. H. Wei and J. J. Zhan. Fuzzy cluster analysis of chemical elements. Computers and Applied Chemistry, 17(2):167–168, 2000.

[24] P. Cunningham. Machine Learning Techniques for Multimedia. University College Dublin, 1^{fst} edition, 2008.

[25] C. J. C. Burges. Dimension reduction: A guided tour. Foundations and Trends in Machine Learning, pages 1–57, 2013.

[26] Y. Kovchegov and I. Zaliapin. Invariance and attraction properties of Galton-Watson trees. arXiv, (3):1–36, 2019.

[27] P. H. A. Sneath. Distortions of taxonomic structure from incomplete data on a restricted set of reference strains. Journal of General Microbiology, 129(4):1045 – 1073, 1982.

[28] R. L. Rivest, T. H. Cormen, C. E. Leiserson and C. Stein. Introduction to Algorithms. Cambridge, Mass. : MIT Press., 3^{fst} edition, 2009.

[29] C. Shen, J. Jansson, R. Rajaby and W. K. Sung. Algorithms for the majority rule (+) consensus tree and the frequency difference consensus tree. IEEE, 15:15–26, 2018.

[30] Z. Li, J. Jansson and W. K. Sung. On finding the Adams consensus tree. Information and Computation, 256:334–347, 2017.

7. Anexos

La metodología propuesta se desarrolla en Python. A continuación se detallan algunas de las secciones del código usado.

* Lineas 1 - 70: Se declaran las librerías usadas y se importan algunas funciones específicas.
* Lineas 72 - 64: Se carga la base de datos y se filtran los datos que se pueden usar.
* Lineas 89 - 90: Funciones de similitud y metodologías a usar.
* Lineas 101 - 184: Algoritmos usados para transformar la información de los dendrogramas a strings (formato newick)
* Lineas 188 - 209: Función que encuentra todos los posibles dendrogramas, guarda la información de estos dendrogramas en el formato newick de la librería Phylo.
* Lineas 219 - 234: Guarda la información del árbol consensus en listas de adyacencia.
* Lineas 239 - 287: Función que halla todas las topologías dado un árbol consensus.
* Lineas 292 - 299: Función que halla los árboles consensus.
* Lineas 303 - 306: Función que dibuja y muestra en consola un determinado árbol consensus.
* Lineas 309 - 346: Algoritmo que halla los posibles candidatos que tienen similaridad y disimilaridad muy cercana a la que genera la base de datos inicial.
* Lineas 349 - 362: Algoritmo que halla las bases de datos a estudiar usando el parámetro x, el cual indica el mínimo número de propiedades independientes cuando se usa PCA.
* Lineas 384 - 400: Modelo de machine learning: el perceptrón multicapas
* Lineas 402 - 490: Uso de las funciones y algoritmos mencionados. También se muestran algunos resultados en consola.

El código usado se muestra a continuación.

```
!pip install scikit-bio
!pip install biopython

from collections import defaultdict
from scipy.spatial.distance import pdist, squareform
from scipy.cluster.hierarchy import linkage, dendrogram
from matplotlib.colors import rgb2hex, colorConverter
from scipy.cluster.hierarchy import set_link_color_palette
import pandas as pd
import numpy as np
import scipy.cluster.hierarchy as sch
```

```
%pylab inline
from pylab import rcParams
rcParams['figure.figsize'] = 20, 15
rcParams['axes.labelsize']='large'
rcParams['font.size']=15
import seaborn as sns
sns.set_style("whitegrid")
from scipy.cluster import hierarchy
from skbio import TreeNode
from google.colab import files

#-------------------------------------------------------
# Consensus Tree

from io import StringIO
from Bio import Phylo
from Bio.Phylo.Consensus import majority_consensus
from Bio.Phylo.Consensus import *
from itertools import permutations
import xml
from xml.dom.minidom import Document
import copy
import json
from sklearn.preprocessing import StandardScaler
from sklearn.decomposition import PCA

import random
import itertools
import email
import networkx, pylab
from io import BytesIO
from Bio.Phylo.Applications import PhymlCommandline
from Bio.Phylo.PAML import codeml
from Bio.Phylo import PhyloXMLIO
from Bio.Phylo.PhyloXML import Phylogeny
from ast import literal_eval
from Bio.Phylo import BaseTree

import matplotlib.pyplot as plt
import networkx as nx
import pydot
from networkx.drawing.nx_pydot import graphviz_layout

from sklearn.model_selection import train_test_split
from sklearn.preprocessing import StandardScaler
from sklearn.preprocessing import MinMaxScaler
from sklearn.decomposition import PCA
```

```
from sklearn.neural_network import MLPRegressor
from sklearn.datasets import load_boston
import matplotlib
import seaborn as sns
import statsmodels.api as sm
%matplotlib inline
from sklearn.linear_model import LinearRegression
from sklearn.feature_selection import RFE
from sklearn.linear_model import RidgeCV, LassoCV, Ridge, Lasso
from sklearn.ensemble import RandomForestClassifier
from sklearn.feature_selection import SelectFromModel
from sklearn import metrics

url='https://raw.githubusercontent.com/phibrainDK/tesis/master/Periodic%20Table'
dataset=pd.read_csv(url)
dataset_t=dataset.iloc[:, [1, 6, 7, 9, 10, 11, 15, 16, 17]]
dataset_t=dataset_t.dropna()
data_labels=dataset_t.iloc[:, 0]
data_values=dataset_t.iloc[:, [1, 2, 3, 4, 5, 6, 7, 8]]
# display(data_values)
list_features=[]
for item in data_values:
    list_features.append(item)
# x=data_values.loc[:, list_features].values
data_labels=data_labels.transpose()
Number_elements = len(data_labels)
# print(Number_elements)
# print(list_features)
#-------------------------------------------------------------

list_metricas=['euclidean', 'cosine', 'correlation', 'hamming']
list_metodos=['ward', 'single', 'complete', 'average', 'weighted', 'centroid']
# ind=[1, 2, 3, 4]

#testear la cantidad de null types
# print(dataset_t.isnull().sum().sum())
trees=[]
# list_names=[]


#bfs necesario para hallar realizar el parser newick

def bfs(u, graph, dep, order, p, depth):
  dep[u] = depth
  order.append(u)
  for v in graph[u]:
    p[v] = u
```

```
106     bfs(v, graph, dep, order, p, depth + 1)
107 
108 #arma el grafo recorriendo el dendrograma, se pasa el root del dendrograma
109 
110 def dfs(u, graph):
111   if(u.count == 1):
112     return
113   graph[u.id].append(u.left.id)
114   graph[u.id].append(u.right.id)
115   dfs(u.left, graph)
116   dfs(u.right, graph)
117 
118 
119 #Parseamos los grafos a strings -> newick parser
120 
121 def go(u, n):
122   global data_labels
123   graph = []
124   for i in range(2*n - 1):
125     graph.append([])
126   dfs(u, graph)
127   dep = [0]*(2*n - 1)
128   order, leaves, p, list_names_leaves = [], [], [], []
129   for i in range(2*n - 1):
130     p.append(-1)
131   bfs(2*n - 2, graph, dep, order, p, 0)
132   for node in order:
133     if(node < n):
134       leaves.append(node)
135   for item in data_labels:
136     list_names_leaves.append(item)
137   ans = ""
138   stack = []
139   for node in leaves:
140     if(len(stack) == 0):
141       stack.append(node)
142       if(len(ans) > 0):
143         ans += ","
144       ans += "("
145       ans += list_names_leaves[node]
146     else:
147       ret = stack[-1]
148       cur = node
149       if(dep[cur] == dep[ret]):
150         while(len(stack) > 0 and cur != -1):
151           ret = stack[-1]
152           if(dep[ret] != dep[cur]):
```

```
            break
          if(cur < n):
            ans += ","
            ans += list_names_leaves[cur]
          ans += ")"
          stack.pop()
          cur = p[cur]
        if(cur != -1):
          stack.append(cur)
      else:
        stack.append(cur)
        ans += ","
        ans += "("
        ans += list_names_leaves[cur]
  ansL, ansR = 0, 0
  for ch in ans:
    if(ch == "("):
      ansL = ansL + 1
    if(ch == ")"):
      ansR = ansR + 1
  if(ansR > ansL):
    base = ""
    for i in range(ansR - ansL):
      base += "("
    ans = base + ans
  else:
    base = ""
    for i in range(ansL - ansR):
      base += ")"
    ans = ans + base
  return ans
tunes = 0;

# Encontramos todos los dendrogramas, guardamos la informacion en L

def Find_all_dendrograms(L, idx, metrica, metodo):
    global Number_elements
    cur=dataset_t.iloc[:, idx]
    current_data_values=pdist(cur)
    if((metodo=='ward' or metodo == 'centroid') and metrica != 'euclidean'):
        return
    den_cur=linkage(cur, metric=metrica, method=metodo)
    cur_tree=dendrogram(den_cur, labels=list(data_labels))
    root_node, nodelist = hierarchy.to_tree(den_cur, rd = True)
    # matplotlib.pyplot.gcf()
    name_fig, name_plot = "", ""
    name_fig += metrica+"_"+metodo+".png"
```

```
200     name_plot += metrica + " " + metodo
201     plt.title(name_plot)
202     cur_dendro_string = go(root_node, Number_elements)
203     handle = StringIO(cur_dendro_string)
204     tree = Phylo.read(handle, "newick")
205     L.append(tree)
206     plt.savefig(name_fig)
207     # plt.same(nametree)
208     # plt.show()
209     plt.clf()
210 
211 # ------------------------------------------------------------------------------
212 # ------------------------------------------------------------------------------
213 list_indices=[1, 2, 3, 4, 5, 6, 7, 8]
214 #-------------------------------------------------------------------------------
215 #-------------------------------------------------------------------------------
216 
217 #Armo el grafo usando el root del consensus tree
218 
219 def dfs1(u, node, graph, leaves):
220   children = list(u)
221   valor = node
222   for item in children:
223     cur_name = str(item.name)
224     if(cur_name == "None"):
225       valor = valor + 1
226       graph[node].append(valor)
227       graph[valor].append(node)
228       valor = dfs1(item, valor, graph, leaves)
229     else:
230       valor = valor + 1
231       graph[node].append(valor)
232       graph[valor].append(node)
233       leaves[valor] = str(item.name)
234   return valor
235 
236 #Para hallar las topologias, necesito los cortes, para esto, se hace bfs1 sobre
237 #el grafo del consensus tree
238 
239 def bfs1(u, p, tsz, pa, vis, graph):
240   tsz[u], pa[u] = vis[u], p
241   for v in graph[u]:
242     if(v != p):
243       bfs1(v, u, tsz, pa, vis, graph)
244       tsz[u] += tsz[v]
245 
246 #hallo el grafo del consensus y los cortes para las  topoligias
```

```
247
248 def build_consensusT(n, root):
249   graph, leaves, tsz, pa, L, vis, n_max, cut_list = [], {}, [], [], [], [], [], {}
250   for i in range(4*n):
251     graph.append([])
252     tsz.append(0)
253     pa.append(-1)
254     vis.append(0)
255   last = dfs1(root, 0, graph, leaves)
256   for item in leaves:
257     vis[item] += 1
258   bfs1(0, -1, tsz, pa, vis, graph)
259   for i in range(0, last + 1, 1):
260     L.append([tsz[i], i])
261   L.sort()
262   for n_maximal in range(1, last + 1, 1):
263     cur, cur_list = [], []
264     selection_number = 0
265     for item in L:
266       cur.append(item)
267     while(len(cur) > 0):
268       a, b = cur.pop()
269       if(a <= n_maximal):
270         if(pa[b] == -1):
271           cur_list.append([a, b])
272         elif tsz[pa[b]] > n_maximal:
273           cur_list.append([a, b])
274     selection_number += np.log(len(cur_list))
275     cut_list[n_maximal] = cur_list
276     for item in cur_list:
277       a, b = item
278       selection_number += np.log(a)
279     n_max.append([selection_number, n_maximal])
280   n_max.sort()
281   a, b = n_max.pop()
282   while(len(n_max) > 0):
283     x, y = n_max.pop()
284     if(x == a):
285       b = y
286   best_cutlist = cut_list[b]
287   return a, b, best_cutlist, graph, leaves, tsz, pa, vis
288
289 #encuentra todos los arboles consensus
290 #usando la libreria biophyton
291
292 def Find_Consensus(lista_ID, x):
293   lista_de_dendrogramas1 = []
```

```
294   for i in range(len(list_metricas)):
295     for j in range(len(list_metodos)):
296       Find_all_dendrograms(lista_de_dendrogramas1, lista_ID, list_metricas[i], list_metodos[
      j])
297   ma_Tree_MR = majority_consensus(lista_de_dendrogramas1, x)
298   ma_Tree_A = adam_consensus(list(lista_de_dendrogramas1))
299   return ma_Tree_MR, ma_Tree_A
300
301 #dibuja un arbol consensus
302
303 def Draw_Consensus(ma_T1):
304   T = ma_T1.as_phyloxml()
305   majority_tree = Phylogeny.from_tree(T)
306   Phylo.draw_ascii(majority_tree)
307
308
309 def dfs_CUT(u, p, graph, vis, List_nodes):
310   if(vis[u] != 0):
311     List_nodes.append(u)
312   for v in graph[u]:
313     if(v != p):
314       dfs_CUT(v, u, graph, vis, List_nodes)
315
316
317 def comp(a, b, x, y):
318   p = abs(a - x)/a
319   q = abs(b - y)/b
320   return p, q
321
322 def Generate_DATA(Base_T, Lista_subConjuntos):
323   Sn, N_maxi, Best_cutM, Graph, Leaves, Tsz, Pa, Vis = build_consensusT(Number_elements,
      Base_T.root)
324   List_CL, List_CN = [], []
325   for item in Best_cutM:
326     a, b = item
327     cur_CUT, cur_LABELS = [], []
328     dfs_CUT(b, Pa[b], Graph, Vis, cur_CUT)
329     for node in cur_CUT:
330       cur_LABELS.append(Leaves[node])
331     List_CL.append(cur_LABELS)
332     List_CN.append(cur_CUT)
333   # print(List_CL)
334   # print(List_CN)
335   List_SubSetsIndex, Lista_Total = [], []
336   for i in range(m + 1):
337     List_SubSetsIndex.append([])
338   for i in range(2, m + 1, 1):
```

```
339     for cur_consensus, cur_list in Lista_subConjuntos[i]:
340       s_n, n_maximal, best_cutM, graph, leaves, tsz, pa, vis = build_consensusT(
        Number_elements, cur_consensus.root)
341       Lista_Total.append([cur_list, s_n, n_maximal])
342       a, b = comp(Sn, N_maxi, s_n, n_maximal)
343       if(a < 0.01 and (n_maximal < N_maxi or b <= 0.2)):
344         List_SubSetsIndex[i].append(cur_list)
345
346   return List_SubSetsIndex, Lista_Total
347
348
349 def Pure_PCA(lista_ID, x):
350   data = dataset_t.iloc[:, lista_ID]
351   pca = PCA(n_components = x)
352   pca.fit(data)
353   data_pca = pca.transform(data)
354   return data_pca
355
356 def Find_DATA(x):
357   Data_GEN = []
358   m = len(list_indices)
359   for i in range(1<<m):
360     cur_list = []
361     for j in range(m):
362       if(i&(1<<j)):
363         cur_list.append(list_indices[j])
364     if(len(cur_list) >= x):
365       cur_data = Pure_PCA(cur_list, x)
366       cur_L = []
367       for item in cur_data:
368         for values in item:
369           cur_L.append(values)
370       Data_GEN.append([cur_L, cur_list])
371   return Data_GEN
372
373 def Build_Data_PCA(total_base, base_tengo):
374   ans = []
375   for [input, lista] in base_tengo:
376     for [cur_lista, valor] in total_base:
377       if(cur_lista == lista):
378         input.append(valor)
379         flag = 1
380         break
381     ans.append(input)
382   return ans
383
384 def MLMethod_eval(ret):
```

```
  #ret es la base de datos en formato dataframe, donde ya esta coeficiente con el ajuste
  x_vars = ret.drop('Coeficiente de similaridad - disimilaridad', axis = 1)
  y_vars = ret['Coeficiente de similaridad - disimilaridad']
  x_train, x_test, y_train, y_test = train_test_split(x_vars, y_vars, train_size = 0.6,
    random_state = 2)
  scaler = MinMaxScaler()
  scaler.fit(x_train)
  x_train = scaler.transform(x_train)
  x_test = scaler.transform(x_test)
  pd.DataFrame(x_train).describe()
  pd.DataFrame(x_test).describe()
  MLMethod = MLPRegressor(hidden_layer_sizes = (900, 900, 900, ), activation = 'relu' ,
    max_iter = 100000, solver = 'lbfgs', learning_rate_init = 0.01, random_state = 2)
  MLMethod.fit(x_train, y_train)
  mae = metrics.mean_absolute_error(y_train, MLMethod.predict(x_train))
  mse = metrics.mean_squared_error(y_train, MLMethod.predict(x_train))
  rsq = metrics.r2_score(y_train, MLMethod.predict(x_train))
  return mae, mse, rsq

m = len(list_indices)
adj, big_consensus = [], {}

base_index = []
for i in range(m):
  base_index.append(list_indices[i])

Base_T, Base_TA = Find_Consensus(base_index, 0.5)
Base_T.rooted, Base_TA.rooted = True, True

Phylo.draw(Base_T, branch_labels=lambda c: c.branch_length, do_show = False)
name_F1 = "ReglaDelMayor.png"
name_T1 = " rbol  Consensus : Regla del Mayor"
plt.title(name_T1)
plt.savefig(name_F1)
plt.show()
# files.download(name_F1)
plt.clf()
#------------------------------------------------------------------------------
Phylo.draw(Base_TA, branch_labels=lambda c: c.branch_length, do_show = False)
name_F3 = "Adam.png"
name_T3 = " rbol  Consensus : Adam"
plt.title(name_T3)
plt.savefig(name_F3)
plt.show()
# files.download(name_F3)
plt.clf()

```

```
Lista_ConsensusTree1, Lista_ConsensusTree2 = [], []
for i in range(0, m + 1, 1):
  Lista_ConsensusTree1.append([])
  Lista_ConsensusTree2.append([])

for i in range(1<<m):
  cur_index = []
  for j in range(m):
    if(i&(1<<j)):
      cur_index.append(list_indices[j])
  tam = len(cur_index)
  if(tam > 1):
    a, b = Find_Consensus(cur_index, 0.5)
    Lista_ConsensusTree1[tam].append([a, cur_index])
    Lista_ConsensusTree2[tam].append([b, cur_index])

ans1, dataTotal1 = Generate_DATA(Base_T, Lista_ConsensusTree1)
ans2, dataTotal2 = Generate_DATA(Base_TA, Lista_ConsensusTree2)
lista_ans1, lista_ans2 = [], []
for a, b, c in dataTotal1:
  lista_ans1.append([a, b])
for a, b, c in dataTotal2:
  lista_ans2.append([a, b])
#me halla la data posible sin el header
for a, b, c in dataTotal1:
  aea1 = []
  for item in a:
    aea1.append(item)
  aea1.append(b)
  aea1.append(c)
  for item in aea1:
    print(item, end = ' ')
  print("")
for a, b, c in dataTotal2:
  aea1 = []
  for item in a:
    aea1.append(item)
  aea1.append(b)
  aea1.append(c)
  for item in aea1:
    print(item, end = ' ')
  print("")

def GOES(prueba_lista, x):
  cur_data = Find_DATA(x)
  a, b = cur_data[0]
  sz, L_cur = len(a), []
```

```
  for j in range(1, sz + 1, 1):
    name = "PC - " + str(j)
    L_cur.append(name)
  L_cur.append("Coeficiente de similaridad - disimilaridad")
  ret = Build_Data_PCA(prueba_lista, cur_data)
  DATA = pd.DataFrame(ret, columns = L_cur)
  display(DATA)
  x, y, z = MLMethod_eval(DATA)
  return x, y, z

x1, y1, z1 = GOES(lista_ans1, 4)
x2, y2, z2 = GOES(lista_ans2, 4)
print(x1, y1, z1)
print(x2, y2, z2)

# for i in range(2, m + 1, 1):
  # if(len(adj[i]) > 1):
    # ret_Consensus = majority_consensus(adj[i], 0.4)
    # big_consensus[i] = ret_Consensus
    # print("El consensus tree de tamanho ", i, "es:")
    # Draw_Consensus(ret_Consensus)
    # n_maximal, best_cutM, graph, leaves, tsz, pa, vis = build_consensusT(Number_elements,
    ret_Consensus.root)
    # print("Se encontro un n-maximal de ", n_maximal)
    # List_CUTLABELS, List_CUTNODES = [], []
    # for item in best_cutM:
      # a, b = item
      # cur_CUT, cur_LABELS = [], []
      # dfs_CUT(b, pa[b], graph, vis, cur_CUT)
      # for node in cur_CUT:
        # cur_LABELS.append(leaves[node])
      # List_CUTLABELS.append(cur_LABELS)
      # List_CUTNODES.append(cur_CUT)
    # print(List_CUTLABELS)
    # print(List_CUTNODES)
# """

# net1 = Phylo.to_networkx(ma_T1)
# networkx.draw(net1, with_labels = True, font_weight = 'bold')
# pylab.show()

# pos = graphviz_layout(ma_T1, prog="dot")
# nx.draw(ma_T1, pos)
# plt.show()

# T = ma_T1.as_phyloxml()
# majority_tree = Phylogeny.from_tree(T)
```

```
# Phylo.draw_ascii(majority_tree)
# print("TODO ESTA HECHO")
# strict_tree = strict_consensus(trees)
# majority_tree = majority_consensus(trees, 0.5)
# adam_tree = adam_consensus(trees)
# Phylo.draw_ascii(strict_tree)
# Phylo.draw_ascii(majority_tree)
# Phylo.draw_ascii(adam_tree)
```