Universdad Nacional Autónoma de México

Clustering basado en algoritmo de optimización de la ballena jorobada

Cómputo Evolutivo

Frida Michelle Vargas Bautista

1. Algoritmo de Optimización basado en la ballena gris *(WOA)*

1.1. Inspiración

En 2016, Mirjalili y Lewis propusieron este algoritmo (*WOA*), el cual se ha convertido en uno de los algoritmos poblacionales de optimización global, más ampliamente usados alrededor del mundo en diversos campos, (Nadimi-Shahraki, Zamani, Asghari Varzaneh, y cols., 2023).

Este algoritmo bio-inspirado se basa en el comportamiento de la ballena gris (*Megaptera novaeangliae*) al cazar, pues este sólo ha sido observado en esta especie. El cual es conocido como *Método de alimentación por red de burbujas* y se fundamenta en la preferencia de las ballenas jorobadas de cazar enjambres de krill o de pequeños peces que están muy cerca de la superficie; pues al ver a su presa, las ballenas empiezan a sumergirse a 12 metros de profundidad para después comenzar a emerger a la superficie creando un espiral de burbujas alrededor de su objetivo, (Mirjalili y Lewis, 2016).

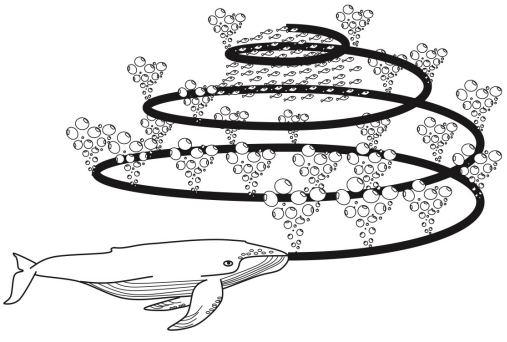


Fig 1: *Método de alimentación por red de burbujas. Figura recuperada del trabajo de (Mirjalili y Lewis, 2016).*

Por otra parte, se ha visto que al momento de buscar alimento lo hacen solas, de hecho, buscan alejarse de sus compañeras y explorar otras zonas; sin embargo, al ya encontrar una presa, empiezan a acorralarla en conjunto, (Mirjalili y Lewis, 2016).

1.2. Consideraciones generales

El algoritmo WOA (*Whale Optimization Algorithm*) simula el comportamiento de las ballenas usando tres estrategias: acorralamiento de la presa, búsqueda de presas (exploración) y el ataque por red de burbujas, (Nadimi-Shahraki y cols., 2023). Cabe mencionar que este algoritmo da el mismo peso a estrategias de exploración y explotación al inicio, pues para un mismo agente (ballena) es equiprobable que se acerque a la ballena con la mejor solución o que se aleje de alguna elegida al azar para explorar el espacio de búsqueda, (Mirjalili y Lewis, 2016).

Al ser un algoritmo poblacional se tienen a agentes (en este caso ballenas) actualizando su posición en el espacio de búsqueda en cada iteración, a partir de las ecuaciones descritas en la siguiente subsección.

1.3. Partes del algoritmo

1.3.1. Acorralar presa

En la realidad, las ballenas jorobadas pueden reconocer la ubicación de la presa y rodearla. Para la implementación, dado que en un problema de optimización no se sabe a priori la ubicación de la mejor solución (presa) en el espacio de búsqueda, vamos a considerar que la mejor solución actual es la presa objetivo o que está muy cerca de ella.

Al acorralar a la presa, la ballena actualiza su posición

para estar en una vecindad de menor radio de la ballena con la mejor evaluación, X^*. La nueva posición de la ballena está descrita por la ecuación:

$$X_{t+1} = X^* - A \cdot D \tag{1}$$

Donde D es el vector de distancias del agente y la ballena con mejor evaluación, reescalada por un factor C, en valor absoluto:

$$D = |C \cdot X^* - X| \tag{2}$$

En el artículo original, se plantea a A y C como vectores, de una manera un poco confusa pues pareciera que se realizan sumas entre vectores y escalares de manera indiscriminada; para evitar esto y operaciones de más, aquí en lugar de verlos como vectores con las mismas entradas los planteamos como escalares:

$$A = 2a \cdot r - a$$

$$C = 2r$$

Donde r es un número aleatorio entre $[0, 1]$ generado en cada iteración, y a decrece de 2 a 0 de manera lineal a lo largo de las iteraciones.
Observemos entonces que si al generar r o ya con a muy pequeñas, obtenemos $|A| < 1$ y significa que se está acercando la ballena de manera lineal a X^*.

1.3.2. Ataque por red de burbujas

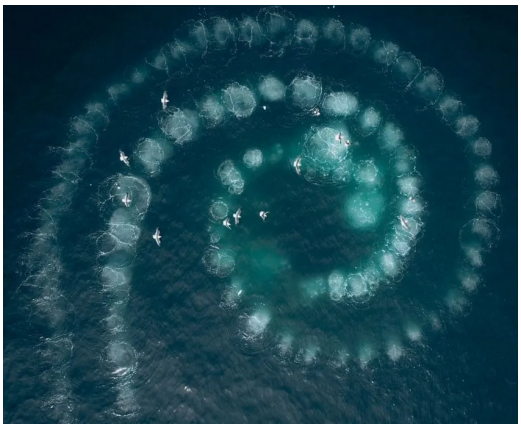


Fig 2: Espirales creado por la red de burbujas. Figura recuperada del trabajo de (Gomede, 2021).

Este ataque es característico por los espirales que se forman, por ende, la actualización de la posición de la ballena siguiendo este método es la ecuación de un espiral:

$$X^{t+1} = D' \; e^{bl} \; cos(2\pi l) + \vec{X}^*(t) \tag{3}$$

Donde D' se calcula con (2) tomando $C = 1$; b es una constante para definir la forma del espiral logarítmico y l un número aleatorio entre -1 y 1.

La observación que hizo (Mirjalili y Lewis, 2016) al modelo es que las ballenas jorobadas nadan alrededor de la presa con círculos que se contraen (acorralamiento) y con rutas en forma de espiral. Entonces para simular este comportamiento, ellos asumieron que ambos eventos son equiprobables para que una ballena actualice su posición a lo largo de la optimización.

$$\vec{X}(t+1) = \begin{cases} X^* - A \cdot D & \text{si } p < 0.5 \\ D' \; e^{bl} \; cos(2\pi l) + \vec{X}^*(t) & \text{si } p \geq 0.5 \end{cases} \tag{4}$$

Donde p es un número aleatorio entre 0 y 1.

1.3.3. Encontrar presas (*exploración*)

Se usa casi la misma ecuación que en (1), sólo que con dos modificaciones clave: la primera es que actualizamos la posición de la ballena no usando la posición de la mejor ballena, sino que con una ballena al azar; la segunda es tomar a A tal que $1 < |A|$ para hacer que la ballena actual se mueva lejos de la ballena de referencia. Esto emula a la realidad pues de hecho, las ballenas buscan a las presas tomando de referencia la ubicación de otras, (Mirjalili y Lewis, 2016). Este comportamiento lo podemos condensar en:

$$X_{t+1} = X_{aleatorio} - A \cdot D \tag{5}$$

$$D = |C \cdot X_{aleatorio} - X|$$

Observemos que, tanto (1) y (5) actualizan la posición de la ballena utilizando el valor de A, tomando de referencia a la mejor ballena y una aleatoria, respectivamente. Sin embargo, como A depende de a que va decreciendo a lo largo de las iteraciones hasta cero, entonces todas las ballenas van a terminar convergiendo al mismo lugar; por ende hacemos notar que este algoritmo converge, cosa que al principio me costaba ver y los autores nunca mencionan explícitamente.

1.4. Pseudocódigo de WOA

A continuación, el pseudocódigo de (Mirjalili y Lewis, 2016) con algunos comentarios personales:

Pseudocódigo de *WOA* (Original)

1. Inicializar la población de ballenas: X_i, $(i = 1, 2, \ldots, n)$
2. Evaluar a cada ballena en la función objetivo
3. X^* : Ballena con mejor evaluación
while $t <$ *num máx de iteraciones* **do**
  for *(cada agente* X_i*)* **do**
    Actualizar parámetros: a, A, C, l, y p
    if $p < 0.5$ **then**
      if $|A| < 1$ **then**
        Actualizar la posición de la ballena actual usando (1) // nos acercamos de manera lineal a la mejor solucion
      end
      else if $|A| \geq 1$ **then**
        Seleccionar ballena al azar, X_r
        Actualizar la posición de la ballena actual usando (5) // nos alejamos de manera lineal de una ballena aleatoria
      end
    end
    else if $p \geq 0.5$ **then**
      Actualizar la posición de la ballena actual usando (3) // un espiral para llegar al mejor
    end
  end
  for *cada agente* X_i **do**
    Calcular la evaluación de cada ballena (*fitness*)
    if *fitness de* X_i *es mejor que* X_{mejor} **then**
      $X_{mejor} \leftarrow X_i$
    end
  end
  $t \leftarrow t + 1$
end
Output: X_{mejor} `// Regresa la mejor solución`

Este no es el pseudocódigo completo que se utiliza en la implementación por tres razones: La primera es que le faltan datos de implementación que uno de los autores sí incluyó en el código de python de su página web, disponible en (Mirjalili, 2024).
La segunda es que la implementación del sitio está escrita en el paradigma de POO, entonces la posición es un atributo del agente que se va actualizando. En mi implementación uso programación estructurada y se crean nuevas posiciones; para que la población se mantenga del mismo tamaño yo agregué un tipo de reemplazo $\mu + \lambda$.
Y por último, y yo creo la más importante, esta implementación así tal cual no es compatible con el problema que deseamos resolver pues, como se verá más adelante, las soluciones no viven en $\mathbb{R}^n$ sino que son matrices con entradas en los reales de $k \times n$.

A pesar de esto, lo incluyo porque es el algoritmo original y es un buen (y necesario) punto de partida. **El algoritmo más detallado y ajustado al problema, se encuentra en la sección de Clustering con WOA**

2. Clustering

En muchos escenarios de la vida real se tienen grandes cantidades de datos u observaciones, y el objetivo es agrupar cuidadosamente (hacer *clusters*) esos datos para poder entender de mejor manera el problema, ver tendencias o patrones en él y decidir de mejor manera un acercamiento a su solución. Para esto usamos el *Clustering*, que se refiere a los algoritmos que clasifican datos (o puntos) basados en similitudes o patrones, (IBM, 2024). Las restricciones consideradas para este problema de clasificación son las siguientes: La unión de los clusters debe de dar todo el conjunto de datos y un mismo punto en el hiperplano (dato) no puede pertenencer a dos clusters distintos.
El Clustering evolutivo hace referencia a la aplicación de algoritmos evolutivos a algoritmos de Clustering. Es decir, la aplicación de estos algoritmos para identificar grupos de un conjunto de datos dado; haciendo que los datos más parecidos estén en un mismo clúster (agrupamiento) y los clusters con datos menos parecidos entre sí, estén más lejos en el espacio uno del otro, (Corne, Handl, y Knowles, 2017).

Si se tiene alguna forma de evaluar (calcular la calidad de) los clústers generados por un algoritmo, entonces el problema de Clustering es considerado un problema NP-difícil (Corne y cols., 2017) y por ende resulta natural pensar en metaheurísticas para abordarlo.

3. Clustering con WOA

3.1. Representación de las soluciones

Un algoritmo de clustering debe devolver los k centros que mejor ajustan a los datos en k clusters o k listas donde cada una contenga los puntos que pertenecen al k-ésimo clúster, (Corne y cols., 2017).

Dados: k el número de clústers deseados y datos en $\mathbb{R}^n$ (tienen n atributos); un individuo en este algoritmo es de la forma:

$$I = (c_1, c_2, \ldots, c_k) \tag{6}$$

Donde c_j es un posible centro para el j-ésimo cluster. Obs, este centroide tiene que tener la misma dimensión que la de los datos:

$$c_j \in \mathbb{R}^n \quad \forall j \in \{1, \ldots, k\}$$

Recordemos que el algoritmo original de WOA trabaja sobre vectores en $\mathbb{R}^n$ y bajo esta representación de soluciones tenemos una matriz de $k \times n$, entonces para adaptarlo, tenemos que hacer un bucle adicional sobre el cuál se optimizará sobre cada uno de los centroides (vectores de n entradas) y esto para cada individuo.

3.2. Métrica empleada

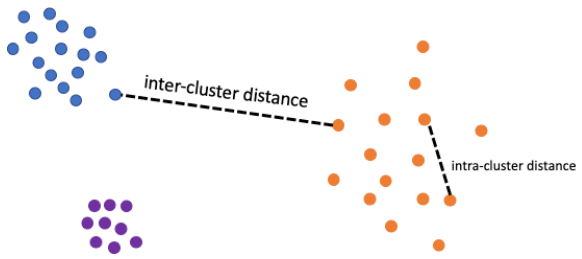


Fig 3: Posibles distancias que evalúa una métrica de Clustering

Para aplicar este algoritmo tenemos que plantear al problema como uno de optimización; con este fin hacemos uso de una de las *métricas de clustering*.

Las métricas de clustering evalúan la coherencia interna y la separación entre cada agrupación (clúster) generada. La coherencia refiere a medir qué tan cercanos están entre sí los elementos de un mismo clúster; lo ideal sería que los elementos que estén en un mismo grupo estén lo más cercanos posibles.

Una de las métricas de clustering más usuales es la **Suma de Errores Cuadrados (SSE)**, que mide la distancia que hay entre cada punto y el centroide de su clúster, (Qaddoura, Aljarah, Faris, y Mirjalili, 2021).

Esto es, sean k el número de clústeres, C_i el conjunto de puntos que pertenecen al clúster con centro en c_i tenemos que:

$$SSE = \sum_{j=1}^{k} \sum_{x \in C_j} |x - c_j|^2 \tag{7}$$

Observemos que si obtenemos valores bajos para la ecuación 7 entonces nos implica que la distancia de los puntos a su centroide es baja y por ende, hay mayor cohesión. A pesar de que una desventaja de esta métrica es que es altamente sensible si existe algún valor muy lejano a los clústeres que ajusten bien a la mayoría, y así desviar o sesgar los resultados obtenidos consideramos que es un buena primera aproximación a la implementación deseada. Así,

> El algoritmo de Clustering con WOA busca optimizar la selección de centroides al minimizar la Suma de Errores Cuadráticos (SSE), 7

3.3. Pseudocódigo de CWOA

En esta sección también incluiremos funciones auxiliares que se usaron en el código.

Pseudocódigo para la función generar población

Input: `num_instancias`: Número de instancias por cluster, entero
`num_clusters`: Número de clusters, entero
`num_individuos`: Número de individuos en la población, entero
`lim_inferior`: Límite inferior para los valores aleatorios, número real
`lim_superior`: Límite superior para los valores aleatorios, número real

Output: `poblacion`: Lista de individuos, cada uno representado por una matriz de tamaño $(num\_clusters, num\_instancias)$

Inicializar `poblacion` como lista vacía;
for $i = 1$ ***to*** `num_individuos` **do**
  `individuo` ← matriz de tamaño $(num\_clusters, num\_instancias)$ con valores aleatorios entre `lim_inferior` y `lim_superior`;
  Agregar `individuo` a `poblacion`;
end
Regresar `poblacion`;

Algorithm 1: Generar población aleatoria de individuos

Pseudocódigo para la función asignar puntos a cluster más cercano

Input: `datos`: Lista de puntos a agrupar, tamaño $m \times d$
`centros`: Lista de centroides, tamaño $k \times d$
Output: `clusters`: Lista de clusters, cada uno con los puntos asignados
Inicializar la lista de k clusters vacíos: `clusters` $\leftarrow$ [`[]`, `[]`, ..., `[]`];
for *cada punto en* `datos` **do**
  Calcular las distancias a cada centroide:
  `distancias` $\leftarrow$ [`distancia(punto, centro)` $\forall$`centro` $\in$ `centros`];
  Encontrar el índice del centroide más cercano: `cluster` $\leftarrow$ `argmin(distancias)`;
  Asignar el punto al cluster más cercano: `clusters[cluster].append(punto)`;
end
Regresar `clusters`;

Pseudocódigo para función objetivo SSE

Input: `puntos_asignados`: Lista de listas, cada sublista contiene los puntos asignados a un cluster.
`centros`: Lista de centros (centroides) de los clusters.
Output: `SSE`: Suma total de las distancias cuadradas entre los puntos y sus respectivos centros.
Inicializar `distancia` como lista de ceros de longitud igual al número de centros;
for $k = 1$ ***to*** *len(centros)* **do**
  Inicializar `distancia[k]` $\leftarrow 0$;
  for *cada punto en puntos_asignados[k]* **do**
    Calcular la distancia entre `punto` y el centro `centros[k]`:
    `distancia[k]` $\leftarrow$ `distancia[k]` $+ (\texttt{norm(punto - centros[k])})^2$;
  end
end
return *SSE* $\leftarrow \sum_{k=1}^{len(centros)}$ *distancia[k]*

Ahora estamos listos para presentar el pseudocódigo de Clustering con WOA

Pseudocódigo de Clustering con *WOA*

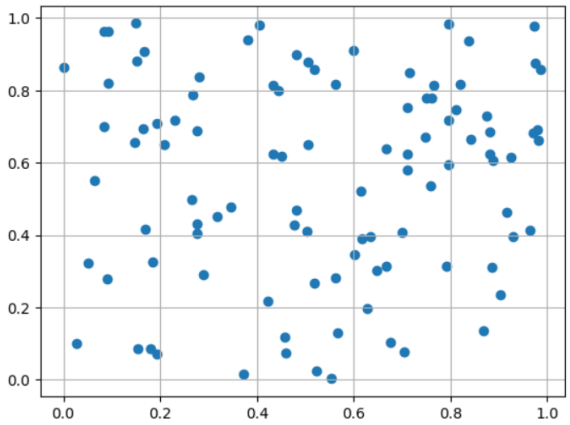
Input: `datos`: conjunto de datos a agrupar.
`num_instancias`: número de puntos a agrupar.
`num_clusters`: número de clusters a formar.
`num_iteraciones`: número de iteraciones de optimización.
`num_individuos`: tamaño de la población en cada iteración.
`lim_inferior`: límite inferior del espacio de búsqueda.
`lim_superior`: límite superior del espacio de búsqueda.
Output: `mejor_solucion`: mejores posiciones de los centroides encontrados.
`sse_minimo`: SSE (Suma de Errores Cuadráticos) de la mejor solución.
Generar población inicial con `Generar_Poblacion`;
Calcular el SSE para cada individuo de la población inicial;
for *cada* ***individuo*** *en la población inicial* **do**
    Asignar los puntos a los clusters con `asignar_puntos`;
    Calcular el SSE de cada individuo;
end
Ordenar la población por el SSE (de mejor a peor);
Establecer `X_mejor` como el individuo con el mejor SSE;
Inicializar `nueva_generacion` con `X_mejor`;
for $t = 1$ ***to*** *num_iteraciones* **do**
    Calcular el parámetro $a = 2 - 2 \cdot (t/\texttt{num\_iteraciones})$;
    for *cada* ***individuo*** *en la población ordenada excepto* *X_mejor* **do**
        Inicializar `clusters_nuevos` como una lista vacía;
        for *cada cluster en el* ***individuo*** **do**
            Calcular los parámetros A, C, l, y b;
            Generar un número aleatorio p entre 0 y 1;
            if $p < 0.5$ **then**
                if $|A| < 1$ **then**
                    Calcular la nueva posición del cluster en trayectoria lineal;
                end
                else
                    Seleccionar un individuo aleatorio de la población;
                    Calcular la nueva posición del cluster alejándose del individuo aleatorio;
                end
            end
            else
                Calcular la nueva posición del cluster en trayectoria espiral;
            end
            Asegurarse de que la nueva posición está dentro de los límites del espacio de búsqueda;
            Agregar la nueva posición a `clusters_nuevos`;
        end
        Agregar `clusters_nuevos` a `nueva_generacion`;
    end
    Calcular el SSE de la nueva generación;
    Ordenar la nueva generación por SSE;
    Limitar la población a `num_individuos`;
    Establecer `X_mejor` como el mejor individuo de la nueva generación;
end
Calcular el SSE de la mejor solución encontrada;
return *X_mejor, sse_minimo*

4. Pruebas de desempeño del algoritmo

Veamos cómo desempeña el algoritmo con diferentes conjuntos de datos, los primeros serán artificiales para después probar con otros datos reales.

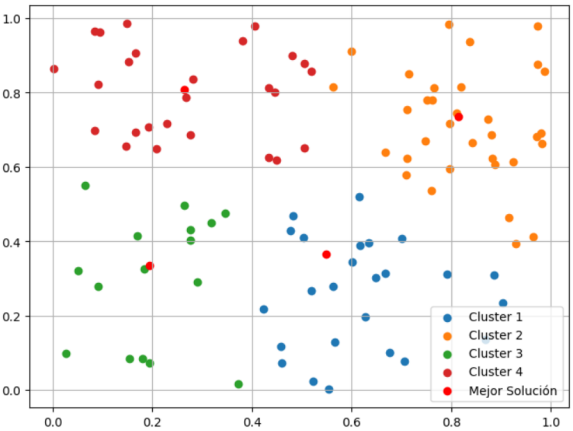
5. Datos aleatorios

Se crearon 100 puntos en $[0,1]\times[0,1]$ y con la biblioteca de matplot.lib de python los visualizamos:



Distribución de puntos aleatorios para hacer Clustering

Ahora usando el algoritmo de clustering con WOA (CWOA) pedimos que encuentre 4 clústers; se usaron 100 iteraciones y 20 ballenas. Arrojando los siguientes resultados:

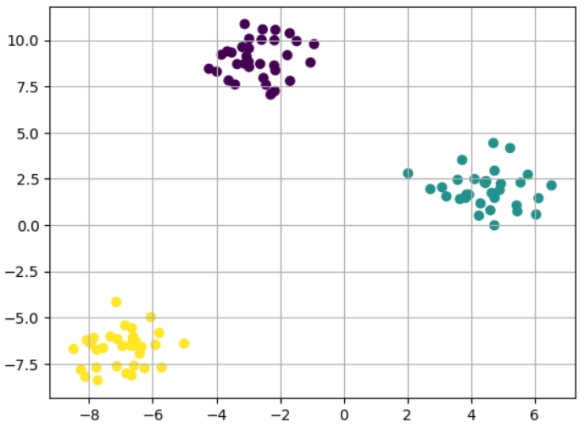


Clusters encontrados con CWOA

Observamos que todos los puntos pertenecen al clúster y se divide al espacio de manera muy razonable en 4 cuadrantes. Un resultado bastante bueno para esta prueba.

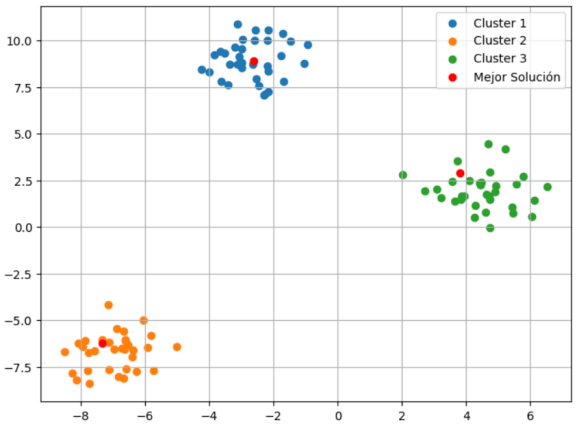
6. Clústers artificiales

La siguiente prueba será usando la función make_blobs de la paquetería skilearn de python. La vamos a usar para crear 100 datos distribuidos en 3 clusters creados de manera artificial. Graficamos y obtenemos:



Clusters creados de manera artificial con 100 puntos

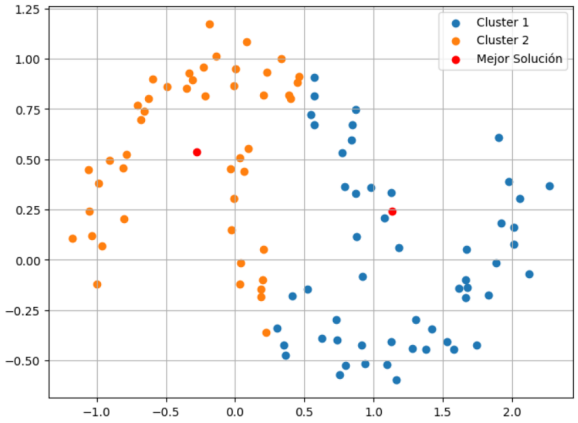
Usamos CWOA que busque en el espacio de búsqueda delimitado por $[-10,10]$ (lo fijamos por insepcción de la gráfica que los puntos pertenecen a esos rangos); con 20 ballenas y 100 iteraciones 3 clústers. Después de 6 minutos aproximadamente, arrojó la siguiente asignación de grupos:



Asignación de clústers con CWOA

Vemos que acertó completamente en la distribución de clusters; notemos que el problema planteado no era tan difícil pues los grupos tenían buena cohesión y distancias intercluster grandes pero sigue siendo un buen desempeño. Respecto al tiempo, sí se tarda un rato pero se hereda esta complejidad a partir de que se tienen

3 bucles *for* anidados por la naturaleza del problema (uno por iteraciones, otro por individuo en población y otro por clúster en individuo) entonces es esperado que este algoritmo sea tardado. Había planteado usar numba para que fuese más rápido pero en ese momento vi mejor conservar el método append() en la función para asignar puntos que usar numba (incompatible con append) pues a priori no se sabe cuántos puntos se le asignan a cada clúster, alguno de ellos incluso puede quedar vacío entonces no sirve de mucho inicializar listas. Además después de un poco de experimentación sí vi que los algoritmos terminaban de ejecutarse en un tiempo decente para mi consideración (menos de media hora).

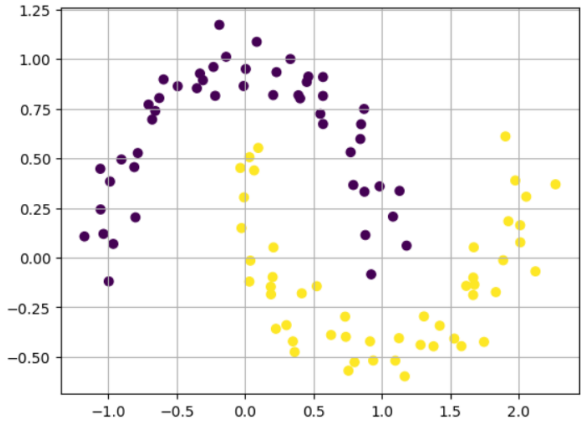


Clusters creados con CWOA

Observamos que aquí ya comete errores de clasificación cuando los clusters deseados son más complejos. Sin embargo, la simetría de los clusters creados alrededor de las mejores soluciones encontradas la encuentro verdaderamente fascinante pues si yo no supiera los verdaderos grupos que se tenían que formar diría que es una buena asignación de puntos en dos grupos.

7. Datos de *Make moon*

En la siguiente prueba también usamos un conjunto de datos de sklearn, llamado make_moon donde se crearon 2 formas de lunas crecientes usando 100 puntos:


Creación de 2 grupos en forma de lunas con 100 puntos

Vemos que aunque el intervalo de búsqueda es menor que en la prueba pasada, los clústers no tienen buena cohesión pues en los extremos de las lunas se ve mas disperso además que las dos lunas están muy pegadas por lo que se presenta como un reto más difícil para el algoritmo.

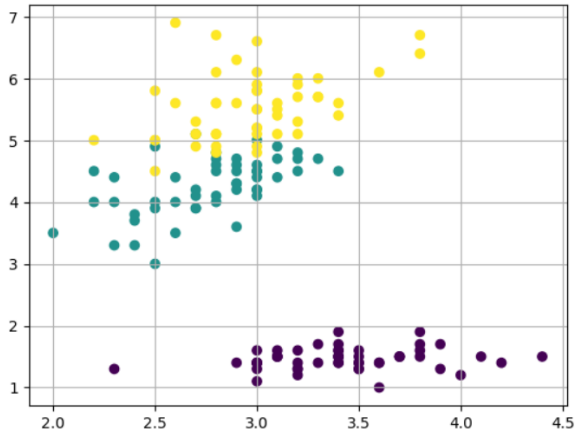
Después de 6 minutos de ejecución, usando 120 iteraciones y 20 ballenas el algoritmo arrojó los siguientes grupos:

8. *Iris* dataset

Pasamos a los conjuntos de datos de la vida real, usando los datos Iris de sklearn.

Este es un conjunto de datos que contiene 150 observaciones de flores de Iris. Estas muestras pueden pertencer a alguna de las especies: Setosa, Versicolor y Virginica. Las observaciones tienen 4 atributos (o instancias) que son: longitud y ancho de pétalos, y longitud y ancho de sépalos. El objetivo aquí es a partir de estas características, determinar a qué especie de Iris pertenece, (Dua y Graff, 2019b).

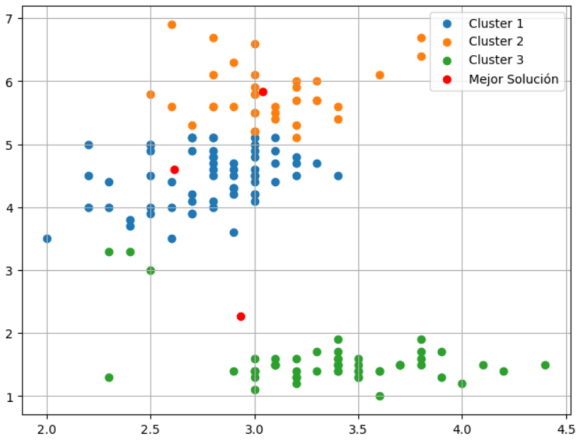
Para esta pequeña prueba y visualizar los datos, sólo usaremos 2 de las 4 características de las muestras (ancho de sépalo y largo de pétalo). Cargamos los datos y coloreamos con los clusters reales (cuáles de esas flores pertenecen a cada una de las especies de setosa). Obtenemos la siguiente gráfica:



Muestras de iris agrupadas por color según su especie. En los ejes se encuentran sus medidas de ancho de sépalo y largo de pétalo.

Cabe mencionar que se usaron esos atributos de las muestras pues por inspección, son las que hacen que los clusters reales tengan meyor separación entre ellos.

Despues de 4 minutos, usando 10 ballenas y 100 iteraciones en el espacio de búsqueda visto en la gráfica anterior, CWOA obtuvo los siguientes clústers.



Clusters creados por CWOA

Aunque tuvo ciertos errores, vemos que en general pudo predecir bien los elementos que corresponden a las 3 especies de iris de las muestras.

9. Clustering con WOA aplicado a base de datos de pacientes con Cáncer de mama

Como aplicación final de la implementación de CWOA, queremos probar el algoritmo en la base de datos de cáncer de mama de sklearn. Son muestras de biopsias de 569 pacientes en Wisconsin. Se analizan 30 características de los núcleos celulares de estas biopsias usando imágenes médicas, (Dua y Graff, 2019a).
Las características consideradas son las siguientes: Radio medio, textura media, perímetro medio, área media, suavidad media, compactación media, concavidad media, puntos cóncavos medios, simetría media, dimensión fractal media, error en el radio, error en la textura, error en el perímetro, error en el área, error en la suavidad, error en la compactación, error en la concavidad, error en los puntos cóncavos, error en la simetría, error en la dimensión fractal, peor radio, peor textura, peor perímetro, peor área, peor suavidad, peor compactación, peor concavidad, peores puntos cóncavos, peor simetría, peor dimensión fractal.

En este problema el espacio de búsqueda para cada coordenada es $\mathbb{R}^{30}$ pues es una dimensión por cada atributo, lo cual como habíamos mencionado antes, aumenta de sobremanera la complejidad del algoritmo y así su tiempo de ejecución, por lo que recurriremos al análisis de componentes principales para replantear el problema en menor dimensión.

9.1. Análisis de componentes principales

El análisis de componentes principales es una técnica de reducción dimensional. En este caso, buscamos reducir un espacio de 30 dimensiones a uno de 2 dimensiones. La base de este nuevo espacio (componente principal 1 y componente principal 2) está formada por combinaciones lineales independientes de los 30 atributos del espacio original (Greenacre, Groenen, Hastie, y cols., 2022). Al realizar esta reducción dimensional (utilizando PCA de la librería sklearn), perdemos parte de la información relativa a la variabilidad del problema original. Si consideramos este problema desde una perspectiva de regresión estadística, donde el objetivo es predecir si el tumor es benigno o maligno a partir de los atributos, la "variabilidad"se refiere a la varianza de las 30 variables explicativas originales.

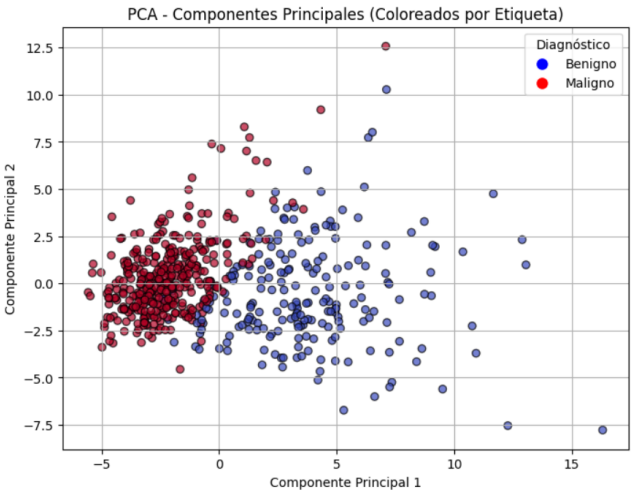
Para evaluar el impacto de esta reducción dimensional, analizamos qué tanto explican estos dos componentes principales respecto a la base de datos original del cáncer. Los resultados del PCA son los siguientes:

Varianza explicada por cada componente:
[0.44272026 0.18971182]

Esto significa que estos dos componentes principales capturan alrededor del 60 por ciento de la dispersión

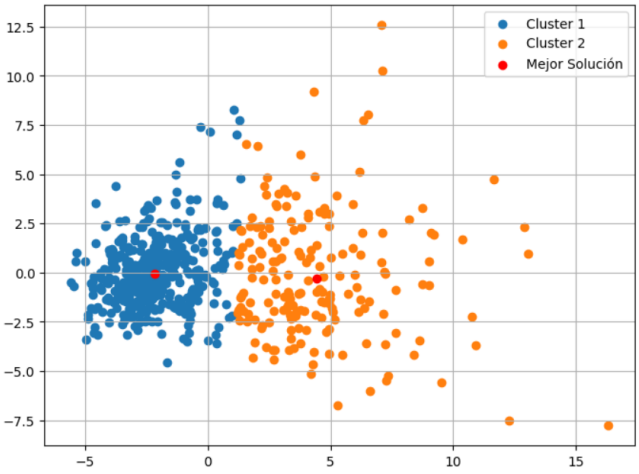
o variabilidad de los datos originales. Este porcentaje es bueno en este contexto, y nos indica que al reducir a dos dimensiones este problema aún capturamos lo .ºencial"de los datos originales. En otras palabras, podemos explicar la presencia de un tumor benigno o maligno utilizando solo estas dos dimensiones (PCA1 y PCA2), que en conjunto representan la información contenida en las 30 dimensiones originales.

Tras verificar que esta reducción dimensional aún mantiene la capacidad de explicar correctamente si un tumor es benigno o maligno utilizando solo PCA1 y PCA2, procedemos a proyectar cada punto del espacio original $\mathbb{R}^{30}$ a las nuevas coordenadas en $\mathbb{R}^2$ y graficamos los resultados:



Pryección de datos en un espacio reducido usando PCA1 y PCA2. Se colorean de acuerdo al diagnóstico que se obtuvo en cada caso en la dimensión original.

Pedimos entonces al algoritmo que encuentre estos dos clústers de los puntos ya reducidos en dos dimensiones. Dimos 100 iteraciones y 20 ballenas y después de 20 minutos estos fueron los clústers que creó CWOA:



Clusters creados por CWOA.

9.2. Análisis de resultados

Por la gráfica mostrada anteriormente podemos decir que desempeñó bastante bien CWOA, sin embargo, queremos mostrar datos de qué tan bien lo hizo y no sólo a ojo de buen cubero. Entonces recurrimos a la matriz de confusión.

9.2.1. Matriz de confusión

Una matriz de confusión (o matriz de error) es una tabla que mide qué tan bueno fue el desempeño de un algoritmo de clasificación. Esto lo logra al desglosar el número de datos que pertenecen a los clusters reales (en este caso cuáles datos fueron para diagnósticos de tumor benignos o malignos en la vida real) y cuáles de ellos el algoritmo predijo bien o mal (IBM, s.f.). Esta matriz la calculamos con confussion_matrix de sklearn y los resultados fueron los siguientes:

Diagnóstico Real	Diagnóstico Predicho	
	Benigno	Maligno
Benigno	175	37
Maligno	11	346

Matriz de confusión.

Aquí observamos que 175 de los que dijo que eran benignos sí lo fueron, y 346 de los malignos también los predijo correctamente. Por otra parte 37 muestras que fueron benignas dijo que eran malignas y 11 malignas dijo incorrectamente que fueron malignas. En cuestión de porcentaje, tuvo correcto 93 % de diagnósticos de tumores benignos correcto y 89 % de diagnósticos malignos correctos. Podemos decir que el algoritmo tuvo un desempeño de alrededor del 90 %.

10. Conclusiones

La implementación del algoritmo bioinspirado WOA para hacer clustering resultó en un algoritmo bastante bueno en las pruebas que se hicieron, resultando en un desempeño del 90 % en la base de datos de biopsias de cancer de mama. Posibles trabajos podrían hacerse para reducir el tiempo de ejecución del algoritmo o hacer varias ejecuciones del algoritmo sobre un mismo conjunto de datos para ver su desempeño promedio y obtener sus medidas de tendencia central para compararlo con otros algoritmos de clustering.

Referencias

Corne, D., Handl, J., y Knowles, J. (2017). *Evolutionary clustering.* Boston, MA: Springer US.

Dua, D., y Graff, C. (2019a). *Breast cancer wisconsin (diagnostic) data set.* `https://archive.ics.uci.edu/dataset/17/breast+cancer+wisconsin+diagnostic`. Descargado de `https://archive.ics.uci.edu/dataset/17/breast+cancer+wisconsin+diagnostic` (Accessed: 2024-11-26)

Dua, D., y Graff, C. (2019b). *Iris data set.* `https://archive.ics.uci.edu/dataset/53/iris`. Descargado de `https://archive.ics.uci.edu/dataset/53/iris`

Gomede, E. (2021). *Unveiling the depths of the whale optimization algorithm (woa).*

Greenacre, M., Groenen, P. J., Hastie, T., y cols. (2022). Principal component analysis. *Nature Reviews Methods Primers*, *2*, 100. Descargado de `https://doi.org/10.1038/s43586-022-00184-w` doi: 10.1038/s43586-022-00184-w

IBM. (s.f.). *Confusion matrix.* Descargado de `https://www.ibm.com/mx-es/topics/confusion-matrix`

IBM. (2024). *Clustering.* Descargado de `https://www.ibm.com/mx-es/topics/clustering`

Mirjalili, S. (2024). *Whale optimization algorithm (woa).* Descargado de `https://seyedalimirjalili.com/woa`

Mirjalili, S., y Lewis, A. (2016). The whale optimization algorithm. *Advances in Engineering Software*, *95*, 51-67. Descargado de `https://www.sciencedirect.com/science/article/pii/S0965997816300163` doi: https://doi.org/10.1016/j.advengsoft.2016.01.008

Nadimi-Shahraki, M., Zamani, H., Asghari Varzaneh, Z., y cols. (2023). A systematic review of the whale optimization algorithm: Theoretical foundation, improvements, and hybridizations. *Archives of Computational Methods in Engineering*, *30*(12), 4113–4159. doi: 10.1007/s11831-023-09928-7

Qaddoura, R., Aljarah, I., Faris, H., y Mirjalili, S. (2021). A grey wolf-based clustering algorithm for medical diagnosis problems. En I. Aljarah, H. Faris, y S. Mirjalili (Eds.), *Evolutionary data clustering: Algorithms and applications* (pp. 73–87). Singapore: Springer Singapore.