Metaheurísticas para el problema de Aprendizaje de Pesos en Características

Stefani Castellanos Erick Silva

Abstract

Inserte una descripción breve del paper.

Palabras claves: Aprendizaje de Pesos en Características, Metaheurísticas, *Machine Learning*.

Introducción

Debido a la gran cantidad de información manejada actualmente, ha surgido la necesidad de reducir el tamaño de dichos datos, sin embargo, esto genera el problema de escoger correctamente que información es relevante para el clasificador.[1]

Reducir el tamaño de los datos es posible a través de los siguientes métodos:

- Seleccionando características del conjunto de datos, lo que reduce el tamaño de las columnas.
- Eliminando las instancias del conjunto de datos que aporte poca información.
- Determinando la importancia de las características, ya que proporciona información que permite guiar al clasificador y así, reducir el tiempo de procesamiento.

Este trabajo se enfocará en esta última, denominado el problema de *Aprendizaje de Pesos en Características*. Se utilizará un algoritmo *greedy* (ávido) denomindado RELIEF, dos metaheurísticas de trayectoría y dos poblacionales, para finalmente compararlas y determinar cual de ellos es más adecuado para el problema.

1 Descripción del problema

Antes de describir el problema APC, es necesario comprender en qué consiste un problema de clasificación; se dispone de un conjunto de posibles clases (C) y un conjunto de datos (X), en donde una instancia x_i es un vector previamente clasificado y de la forma:

$$x_i = \langle f_1, f_2, \ldots, f_n, c \rangle$$

en donde:

- f_i : el valor de cada característica.
- n : la cantidad de característcas.
- c : la clase a la que pertenece dicha instancia, con $c \in C$.
- $|X| = m$: cantidad de instancias.
- $|C| = p$: cantidad de clases.

Es común particionar X en dos subconjuntos que representen el conjunto de entrenamiento y el de prueba, Xe y Xp respectivamente, de manera que $X?e$ pueda ser utilizado para que el algoritmo aprenda los parámetros que le permitan clasificar correctamente todas sus instancias y utilizar X_p para validar los resultados obtenidos.

El problema del APC consiste en optimizar el rendimiento de un clasificador, a partir de la inclusión de pesos asociados a las características del conjunto de datos. Estos pesos ponderan la relevancia de cada característica en y modifican su valor al momento de calcular las distancias entre instancias, de tal forma que los clasificadores que se construyan a partir de estos pesos sean certeros y/o más rápidos. En este caso en particular, el clasificador considerado será el *K-Nearest Neighbor* (K-vecinos más cercanos) con $K = 1$ (1-NN).

El algoritmo K-NN asume que todas las instancias corresponden a puntos en un espacio n-dimensional ($\Re^n$), en donde n es la cantidad de características del conjunto de datos. [7]. El algoritmo es el siguiente:

Algorithm 1: K-Nearest Neighbor

Input: Conjunto de entrenamiento X_e, Conjunto de prueba X_p, K
Output: Conjunto X_p clasificados según C

```
foreach x_i ∈ X_p do
    vecinos = [ ];
    foreach x_j ∈ X_e do
        d = distancia(x_i, x_j);
        agregar(x_j, d, vecinos);
    k_vecinos = seleccionar_cercanos(k, vecinos);
    clasificar(x_i, k_vecinos);
return X_p clasificado
```

El proceso de aprendizaje de este clasificador consiste en almacenar una tabla con las instancias

correspondientes al X_e junto a la clase asociada a cada uno de ellos. Dado una instancia $x_i \in X_p$, se calcula su distancia a todas las otras que pertenecen al conjunto de entrenamiento y se escogen las k más cercanas [4]; usualmente se determina la proximidad entre dos ejemplos utilizando la distancia Euclideana. Finalmente, x_i se clasifica según la clase mayoritaria grupo y se retorna el conjunto de pruebas clasificado.

1.1 Función objetivo

Al ser APC un problema de optimización, es necesario establecer cual es la función que se desea mejorar. En este caso, dadas las características del problema, resulta evidente que obtener una "buena" solución está fuertemente relacionado con la cantidad de instancias clasificadas correctamiente usando 1-NN. El objetivo es encontrar el mejor vector de pesos que permita máximizar la tasa de aciertos del clasificador 1-NN. Más específicamente:

$$Max\ tasa(\text{1-NN}(X, W)) = 100 \times \frac{aciertos(X)}{total(X)} \quad (1)$$

sujeto a:

$$w_i = [0, 1] \quad 1 \leq i \leq n$$

donde:

- $W = \langle w_1, \ldots, w_n \rangle$ es una solución al problema.
- 1-NN es el clasificador k-NN con $k = 1$ vecinos, generado a partir del conjunto de datos inicial.
- X es el conjunto de datos sobre el que se evalúa el clasificador.
- *aciertos* es la cantidad de instancias de X clasificadas correctamente por 1-NN.
- *total* es la cantidad total de instancias de X.

1.2 Representación de la solución

La solución al problema de Aprendizaje de Pesos en Características viene dado por $W = \langle w_1, \ldots, w_n \rangle$, un vector de números reales de tamaño n (cantidad de características) en el que el valor de cada w_i define el peso que pondera a la característica f_i, es decir, representa que tan importante para distinguir un ejemplo de otro.

Cada w_i debe pertenecer al intervalo $[0, 1]$; los valores cercanos a 1 indican que la característica es más importante. En caso de que algún valor quede fuera de este intervalo, se debe normalizar, seleccionando al máximo valor del vector y dividiendo todos los valores entre dicho número.

2 Algoritmo Relief

Es un algoritmo para escoger características inspirado en el aprendizaje basado en instancias, detecta aquellos ejemplos que son estadísticamente relevantes para el concepto objetivo en tiempo lineal ($\Theta(nmp)$). Utiliza un *threshold*, τ entre $0 \leq \tau \leq 1$, que codifica la relevancia de una característica en particular [6]; en esta versión será omitido este parámetro puesto que las metaheurísticas utilizan el vector de pesos y no el vector de relevancias que se obtiene con τ.

Relief utiliza la distancia euclideana n-dimensional para determinar el "amigo más cercano" y el "enemigo más cercano" de una instancia x_i. Se denomina a una instancia "amigo más cercano" o "near-hit" a la instancia que pertenezca a la misma clase de x_i y se encuentre a menor distancia. Se denomina a una instancia "enemigo más cercano" o "near-miss" a la instancia que pertenezca a una clase diferente a x_i y se encuentre a menor distancia [6].

Algorithm 2: RELIEF

Input: Conjunto de entrenamiento X_e
Output: Vector de pesos W

```
1  W = 0, 0 ..., 0
2  foreach x_i ∈ X_e do
3      a = amigo_mas_cercano(x_i);
4      e = enemigo_mas_cercano(x_i);
5
6      /* Actualizar pesos de W */
7      for i ... n do
8          dif_amigo = diferencia(x_i, a);
9          dif_enemigo = diferencia(x_i, e);
10         w_i = w_i - dif_amigo + dif_enemigo;
11 normalizar(W);
12 return W
```

La diferencia entre el valor x_i y el amigo/enemigo más cercano está definido por: - Para los atributos con valores numéricos:

$$\text{diferencia}(a, b) = (a - b)^2$$

- Para los atributos con valores nominales:

$$\text{diferencia}(a, b) = \begin{cases} 0 & \text{son el iguales} \\ 1 & \text{son diferentes} \end{cases}$$

3 Metaheurísticas

Las soluciones a muchos problemas de optimización son intratables, es decir, obtener la mejor respuesta podría tomar un tiempo potencialmente infinito, no ser

resoluble o no se dispone de la capacidad computacional suficiente para resolverlo. Las metaheurísticas constituyen un conjunto de estrategías para guiar heurísticas a encontrar soluciones aceptables en un tiempo razonable para resolver un problema difícil o del que no se dispone información completa [9]. Usualmente poseen un componente estocástico, por lo que la solución depende de las variables aleatorias generadas y se describen los resultados basados en resultados empíricos. Existen tres tipos de metaheurísticas: de trayectoria, poblacionales e hibridas.

En general, las meteheurísticas mejoran soluciones obtenidas anteriormentes. Sus componentes principales son:

- Inicialización. Se requiere alguna solución inicial según la representación del problema particular, esta puede ser generada de manera aleatoria o utilizando algun algoritmo *greedy*.
- Operador de vecindad. Se debe disponer de un algortimo que, a partir de otra solución, genere un conjunto de soluciones "vecinas". Este operador puede entenderse como una pequeña perturbación en alguna componente de la representación utilizada.
- Criterio de selección. Entre las soluciones que pertenecen a la vecindad, se debe escoger la "mejor". Existe diversas políticas, las más conocidas son: el mejor de toda la vecindad, el primer mejor y el mejor de una porción de la vecindad. La escogencia de algun criterio sobre otro depende del problema.
- Criterio de convergencia. Al mejorar una solución a partir de la anterior, es necesario contar con algún mecanismo que permita detener la ejecución de la metaheurística y devolver alguna solución. Los más conocidos son: detenerse al encontrar el óptimo (si este es conocido), luego de un número fijo de iteraciones y luego de un número fijo de iteraciones sin cambiar la mejor solución.

3.1 Metaheurísticas de trayectoria

Al resolver un problema de optimización, las metaheurísticas de trayectoria, utilizan una solución y realizan mejoras sobre esta, iterativamente; pueden ser entendidas como "caminatas" sobre el espacio de búsqueda del problema. Probablemente la metaherística de trayectoria más conocida es *Local Search*, esta toma una solución inicial, encuentra sus vecinos y escoge el mejor según un criterio [9].

Algorithm 3: Local Search

```
   Output: Mejor solución s
1  s = gen_sol();
2  while !criterio_convergencia() do
3  |   V = gen_vecinos(s);
4  |   s' = seleccionar_mejor(V);
5  |   if costo(s) < costo(s) then
6  |   |   s = s';
7  return s
```

3.1.1 Búsqueda Local Iterada (Iterated Local Search)

Local Search podría quedar atrapado en un óptimo local y jamás llegar al global ya que se encarga de intensificar la búsqueda, por esta razón ILS supone una mejora sobre LS. Es una de las metaheurísticas más fáciles de implementar, primero se aplica LS a la solución inicial; luego, en cada iteración, se realiza una perturbación del óptimo local y se repite el proceso hasta un cumplir el criterio de aceptación [9].

Algorithm 4: Iterated Local Search

```
   Output: Mejor solución s
1  s = gen_sol();
2  s = local_search(s);
3  while !criterio_convergencia() do
4  |   perturbar(s);
5  |   s = local_search(s);
6  return s
```

3.1.2 Recocido Simulado (Simulated Annealing)

Su nombre proviene del proceso físico de recocer sólidos, en el que un sólido cristalino es calentado y luego se deja enfriar muy lentamente hasta que alcance su configuración más estable y estructuralmente superior, por lo tanto está libre de defectos del cristal [3]. En términos de resolver un problema de optimización, el sólido representa una solución, "calentarlo" es perturbar la solución hasta que la temperatura se estabilice y el "enfriamiento" es una función no-creciente.

En cada iteración, Simulated Annealing, genera dos posibles soluciones: la actual y otra, las cuales son comparadas. Las soluciones que mejoren la actual son siempre aceptadas y existe una función que permite escoger alguna considerada "inferior" y así, escapar de un mínimo local. La probabilidad de aceptar una solución que no mejora depende de la temperatura,

que típicamente es una función no-creciente en cada iteración del algoritmo y al aproximarse la temperatura a cero la probabilidad de aceptar una solución que no mejora es menor [3]. La temperatura sólo decrece cuando se ha alcanzado una condición de equilibrio.

Algorithm 5: Simulated Annealing

```
   Input: T_inicial
   Output: Mejor solución s
1  s = gen_sol();
2  T = T_inicial;
3  while !criterio_convergencia() do
4      while temp_inestable do
5          /* Temperatura inicial */;
6          s' = gen_sol();
7          if costo(s) < costo(s') then
8              s = s';
9          else
10             r = rand();
11             delta = costo(s) - costo(s');
12             acep = 1/exp^(1+exp(1+delta/T));
13             if r < acep then
14                 s = s';
15     T = actualizar_temp();
16 return s
```

3.2 Metaheurísticas poblacionales

La mayoría de estos algoritmos están inspirados en fenómenos naturales y el comportamiento de los animales. Estas metaheurísticas toman una población inicial que está constituida por un conjunto de posibles soluciones del problema. Luego, iterativamente, se generan nuevos individuos (soluciones) a través de "cruces" y mutaciones en los genes (componentes del vector), los cuales son seleccionados para reemplazar a algunos de la actual población, creando una nueva. Este proceso se repite hasta alcanzar un criterio de convergencia dado [9].

Los cruces puede ser entendidos como una combinación de dos individuos de la población para crea unos nuevos, preservando algunas características de los originales. El operador de mutación efectúa una perturbación de un hijo generado; el propósito de este es realizar variaciones en la población que permitan explorar nuevos lugares en el espacio de solución, es decir, diversificar la búsqueda.

3.2.1 Scatter Search

Es un metaheurística evolutiva y poblacional que recombina soluciones seleccionadas de un "conjunto de referencia" para construir otras nuevas. El método comienza generando una población inicial cuyos individuos satisfacen un criterio de diversidad y calidad. El conjunto de referencia es construido seleccionando buenas representaciones de la población las cuales son combinadas para proveer otras iniciales y luego mejorarlas con procedimientos basados en metaheurísticas de trayectoria como *Local Search*. De acuerdo con este procedimiento, el conjunto de referencia es actualizado para que contenga soluciones de buena calidad y otras que permitan diversificar la búsqueda. El proceso itera hasta que un criterio de parada se satisface [9].

Scatter Search aprovecha la información proveída por una heurística para crear las soluciones "buenas" e intensificar la búsqueda alrededor de estos espacios.

Algorithm 6: Scatter Search

```
   Input: Población P
   Output: Mejor solución p_mejor
1  P = mejorar_pop(P);
2  refSet = inicializar_pop(P);
3  while !criterio_convergencia() do
4      foreach r ∈ refSet do
5          r = combinar(r, r');
6      refSet = mejorar_ref(refSet);
7      P = actualizar(P);
8      p_mejor = mejor(P);
9  return p_mejor
```

3.2.2 Differential Evolution

Es un algoritmo evolutivo que genera su población inicial de manera aleatoria y con un tamaño mayor o igual a 4. Suele utilizarse para problemas de optimización contínuos pues cada individuo tiene componentes con números reales.

El operador de cruce que utiliza está basado en la combinación líneal de las soluciones utilizando la distancia entre las mismas. Dado un padre x y tres otro individuos seleccionados aleatoriamente $r1, r2$ y $r3$ se crea un nuevo vector $u = r1 + F(r2 - r3)$. F representa un factor permite añadir un peso o importancia a la diferencia entre $r2$ y es tal que $r3$ y $F \in (0, 1)$ [3]. Finalmente se reemplaza p, con p', de la siguiente manera:

$$p'_i = \begin{cases} u_i & \text{si } j < CR \\ p_i & \text{si } j \geq CR \end{cases}$$

donde:

- $CR \in (0,1)$: *Crossover Constant.* un parámetro que representa la probabilidad de cruce.
- j : el número aleatorio entre 0 y 1 para escoger un componente u otro.
- $1 \leq i \leq n$: índice para cada característica.

Algorithm 7: Differential Evolution

Input: Población P, F, CR
Output: Mejor solución p_mejor

```
while !criterio_convergencia() do
    foreach p ∈ P do
        for i = 0...n − 1 do
            /* Mutar y cruzar */
            r = rand(0, 1)
            if r < CR then
                u_i = r1_i + F(r2_i − r3_i);
            else
                u_i = p_i
            /* Reemplazar según la regla */
            p'_i = reemplazar(p_i, u_i)
    p_mejor = mejor(P);
return p_mejor
```

4 Implementación

Para desarrollar las metaheurísticas y algoritmos explicados anteriormente y obtener los resultados se utilizó en lenguaje de programación C++ que se caracteriza por ser altamente eficiente.

En cuanto a la implementación de 1-NN, se retorna el valor de la cantidad de instancias clasificada correctamentes para el conjunto de datos dado y el cálculo de la cercanía entre dos instancias se realiza utilizando la fórmula de distancia euclideana pesada con la importancia de cada caraterística:

$$\sqrt{\sum_{i=1}^{n}(w_i * (x_i - x'_i))^2}$$

Para modelar la solución al problema planteado se utilizó un vector de números reales con doble precisión (`double`). El operador para generar vecinos selecciona un número aleatorio, *posToChange*, que representa que posición del vector a modificar, luego se escoge un otro número aleatorio $rand \in [-1, 1]$ y se procede de la siguiente manera:

$$w'_{posToChange} = w_{posToChange} + rand$$

$$W' = \begin{cases} w'_{posToChange} = 0 & \text{si } w'_{posToChange} < 0 \\ normalizar(W') & \text{si } w'_{posToChange} \geq 1 \\ W' & \text{de lo contrario} \end{cases}$$

Con respecto a las metaheurísticas, el criterio para seleccionar el mejor vecino es "el mejor de una porción" ya que el "mejor" de todos no era factible puesto que los vecinos de una solución específica son infinitos. Se utilizaron dos criterios de convergencia: "cantidad fija de iteraciones" y "cantidad de iteraciones sin cambios". A continuación se específica el criterio de convergencia de las metaherísticas implementadas:

- Local Search : Cantidad fija de iteraciones. Si la mejor solución no varía después de dos iteraciones se considera que hubo convergencia.
- Iterated Local Search : Cantidad de iteraciones sin cambios.
- Simulated Annealing: Cantidad fija de iteraciones. Si la mejor solución no varía después de dos iteraciones se considera que hubo convergencia
- Scatter Search: ambas.
- Differential Evolution: ambas.

Como método de perturbación de una solución se utilizó la idea para generar un vecino pero no se modifica una posición del vector sino dos; esto se realiza para aquellas metaheurísticas en las que se requiere un operador de vecidad más amplio como ILS.

Simulated Annealing es uno de los algoritmos implementados que posee más sutilezas y variaciones; requiere que se planifique el enfriamiento de alguna maneta. Para alcanzar el estado de equilibrio (en donde no se varía la temperatura) se realiza un número de transiciones estáticas en las que se fijó una cantidad de iteraciones. Existen diversas funciones para disminuir la temperatura, en este caso se utilizó una función logaritmica que decrece según el número de iteraciones i:

$$\frac{T_{inicial}}{\log(i)}$$

Las metaheurísticas poblacionales requieren de un mecanismo que permita combinar soluciones para crear nuevas, esto se denomina operador de cruce. Debido a que el vector de pesos está compuesto por números reales, se implementó el operador Blend Alpha Crossover que selecciona dos padres, digamos

Y y Z, para generar dos hijos a partir de estos ($C1$ y $C2$) y para cada una de sus componentes, i se genera un número aleatorio dentro del intervalo $[\min(y_i, z_i) - \alpha * diff, \max(y_i, z_i) - \alpha * diff]$ en donde α es un parámetro entre $[0, 1]$ y $diff = |y_i - z_i|$. En particular, se utiliza en Scatter Search ya que Diferential Evolution posee su propio operador de cruce.

Para obtener el subconjunto de entrenamiento y el de pruebas para un conjunto de datos específico se procede a desordenar el arreglo de instancias utilizando un generador de números aleatorios con una semilla de manera que la partición se mantenga en cada corrida del algoritmo. Se seleccionan las primeras para entrenamiento y el resto para pruebas según un número real dado, por ejemplo: 0.7 genera 70% para entrenar y 30% para las pruebas.

5 Experimento

5.1 Conjunto de datos

Con la expansión del área de Inteligencia Artificial, se ha convertido en una necesidad contar con bases de datos que proporcionen información que permita a los investigadores, educadores y estudiantes del área realizar análisis sobre los algoritmos de *Machine Learning*. Una de las librerias más populares en la actualidad es UCI Machine Learning Repository, que mantiene una colección de conjuntos de datos, teorías de dominio y generadores de datos disponibles para toda la comunidad.

Para efectuar el análisis de las metaheurísticas a utilizar para resolver el problema APC se utilizarán cuatro librerias disponibles en UCI:

Iris: este es probablemente el conjunto de datos mejor conocido y citado en la literatura de reconocimiento de patrones. Contiene tres clases de 50 instancias cada una, en donde cada clase corresponde a un tipo de planta Iris. Cada instancia posee las siguientes características: largo del sépalo, ancho del sépalo, largo del pétalo y ancho del pétalo en centímetros. Las clases a predecir son: "Iris-Setosa", "Iris-Versicolor" e "Iris-Virginica". [2].

Sonar: es una base de datos de detección de materiales mediante señales de sónar que "rebotan" en los objetos desde diferentes ángulos y bajo condiciones varias, discriminando entre cilindros metálicos(*mines*) y rocas(*rocks*). Cuenta con 111 *mines* y 97 *rocks* donde cada instancia es un conjunto de 60 números entre 0.0 y 1.0 que representan la energía entre una banda en particular, integrada sobre un periodo dispuestas en orden cresciente según el ángulo. Las clases a predecir son: "R" (*rocks*) y "M" (*mines*). [8].

Winsconsin Diagnostic Breast Cancer: es una base de datos contiene atributos calculados a partir de una imagen digitalizada de una aspiración con aguja fina de una masa en la mama. Se describen las características de los núcleos de las células presentes en la imagen. La tarea consiste en determinar si un tumor encontrado es benigno o maligno. Cuenta con 357 tumores beningnos y 212 malignos en donde cada instancia posee 30 caracteristicas, 10 valores reales computados por cada célula: radio, textura, perímetro, área, suavidad, compacto, concavidad, puntos cóncavos, simetría y dimensión fractal. [10].

Spam Base: es una base de datos de detección de SPAM (correo basura) frente a correo electrónico seguro. Cuenta de 4601 ejemplos y 57 atributos en donde los primeros 48 corresponden al porcentaje de frecuencia de una palabra en particular en un correo, 6 corresponden al porcentaje de frecuencia de símbolos de puntuación y las últimas 3 son el promedio, máximo y la suma de la cantidad de letras en mayúsculas. Las clases a predecir son: 1 (spam), 0 (non-spam) que representan [5].

5.2 Particionamiento de los datos

Los conjuntos de datos mencionados anteriormente fueron particionados en dos subconjuntos: entrenamiento y prueba; se utilizó el 60% y 40% de los datos respectivamente. Cuando se analiza el comportamiento de una metaheurística probabilística, se desearía que el resultado obtenido no estuviera sesgado por una secuencia aleatoria concreta que pueda influir positiva o negativamente en las decisiones tomadas durante su ejecución. Para minimizar el impacto de esto, se realizaron varias particiones utilizando diferentes semillas para desordenar los datos de manera que siempre se diferentes particiones que son permanecen fijas en cada ejecución.

5.3 Descripción del experimento

Se realizaron dos tipos de experimentos. El primero de ellos consistió en utilizar diferentes parámetros para cada una de las metaheurísticas y con sólo dos particiones, con el objetivo de encontrar los parámetros más adecuados (entre los probados) para cada uno de los conjuntos de datos. En las tablas ?,?,? y ? se encuentran los resultados.

El segundo experimento se basa en los mejores parámetros obtenidos según los resultados promedio del primero y se realizan 5 particiones en las que se obtiene la media. En las tablas ?,?,? y ? se encuentran los resultados. En concreto, se consideraron los siguientes vectores de pesos (solución):

- Sin pesos: todas las características tienen el mismo peso (igual a 1).
- Relief: el resultado de ejecutar el algoritmo Relief.
- ILS aleatorio: el resultado de ejecutar Iterated Local Search tomando como solución inicial un vector aleatorio.
- ILS relief: el resultado de ejecutar Iterated Local Search tomando como solución inicial el resultado de relief.
- SA aleatorio: el resultado de ejecutar Simulated Annealing tomando como solución inicial un vector aleatorio.
- SA relief: el resultado de ejecutar Simulated Annealing tomando como solución inicial el resultado de relief.
- SS aleatorio: el resultado de ejecutar Scatter Search en donde todos los individuos de la población son aleatorios.
- SS relief: el resultado de ejecutar Scatter Search en donde uno de los individuos es el resultado de Relief y el resto de la población es aleatorios.
- DE: el resultado de ejecutar Diferential Evolution con una población aleatoria.

6 Análisis de resultados

Inserte Resultados

Conclusiones

Aquí concluyen.

References

[1] Cano, J. Herrera, F. Lozano. M Using evolutionary algorithms as Instance Selection for data reduction in KDD: an experimental study. *IEEE Transaction on Evolutionary computation*, 2003.

[2] Fisher, R.A. (1988). UCI Machine Learning Repository [`https://archive.ics.uci.edu/ml/machine-learning-databases/iris/`]. Irvine, CA: University of California, School of Information and Computer Science.

[3] Glover, F Handbook of metaheuristics. *Operation resarch and management science*, 2003.

[4] Herrera, F. Metaheurísticas. *Seminario 2: Problemas de optimización con técnicas basadas en búsqueda local*, 2017. Disponible en: `http://sci2s.ugr.es/sites/default/files/files/Teaching/GraduatesCourses/Metaheuristicas/Sem02-Problemas-BusquedaLocal-MHs-16-17.pdf`

[5] Hopkins, M. (1999). UCI Machine Learning Repository [`https://archive.ics.uci.edu/ml/machine-learning-databases/spambase/`]. Irvine, CA: University of California, School of Information and Computer Science.

[6] Kira, K., Rendell, A. A practical approach to feature selection, 1992.

[7] Mitchell, T. Machine Learning. *From Book News, Inc*, 1997.

[8] Sejnowski, T (año). UCI Machine Learning Repository [`http://archive.ics.uci.edu/ml/machine-learning-databases/undocumented/connectionist-bench/sonar/`]. Irvine, CA: University of California, School of Information and Computer Science.

[9] Talbi E. Metaheuristics: From desing to implementation *University of Lille*, 2009.

[10] Wolberg, W. (1995). UCI Machine Learning Repository [`https://archive.ics.uci.edu/ml/machine-learning-databases/breast-cancer-wisconsin/`]. Irvine, CA: University of California, School of Information and Computer Science.

Apéndice

Bla.