Práctica 1 Metaheurísticas.
Búsqueda por trayectorias para el problema de la selección de características
Curso 15-16
Algoritmos: SFS, LS, SA, TS, Extended TS

Jacinto Carrasco Castillo
N.I.F. 32056356-Z
jacintocc@correo.ugr.es

7 de abril de 2016

Índice

1. Descripción del problema

El problema que nos ocupa es un problema de clasificación. Partimos de una muestra de los objetos que queremos clasificar y su clasificación, es decir, la clase a la que pertenece y pretendemos, en base a esta muestra, poder clasificar nuevas instancias que nos lleguen. La clasificación se realizará en base a una serie de características, que nos permitan determinar si un individuo pertenece a un grupo u otro. Por tanto, tendremos individuos de una población Ω representados como un vector de características: $\omega \in \Omega; \omega = (x_1(\omega), \dots x_n(\omega))$, donde ω es un individuo de la población y $x_i, i = 1, \dots n$ son las n características sobre las que se tiene información. Buscamos $f : \Omega \longrightarrow C = \{C_1, \dots, C_M\}$, donde $C = \{C_1, \dots, C_M\}$ es el conjunto de clases a las que podemos asignar los objetos.
El problema de clasificación está relacionado con la separabilidad de las clases en el sentido de que existirá la función f anteriormente mencionada siempre que las clases sean separables, es decir, siempre que un individuo con unas mismas características pertenzcan a una misma clase. Sin embargo, si se da que dos individuos $\omega_1, \omega_2 \in \Omega$, $(x_1(\omega_1), \dots x_n(\omega_1)) = (x_1(\omega_2), \dots x_n(\omega_2))$ y sin embargo $f(\omega_1) \neq f(\omega_2)$, no podrá existir f. En todo caso, querríamos obtener la mayor tasa de acierto posible.
Por tanto, queremos, en base a unos datos, hallar la mejor f posible. De esto trata el aprendizaje clasificado: Se conocen instancias de los datos y las clases a las que pertenecen. Usaremos como técnica de aprendizaje supervisado la técnica estadística conocida como k vecinos más cercanos. Se trata de buscar los k vecinos más cercanos y asignar al objeto la clase que predomine de entre los vecinos. En caso de empate, se seleccionará la clase con más votos más cercana.
Pero no nos quedamos en el problema de clasificación, sino que buscamos reducir el número de características. Con esto pretendemos seleccionar las características que nos den un mejor resultado (por ser las más influyentes a la hora de decidir la categoría). Usaremos los datos de entrenamiento haciendo pruebas mediante diferentes metaheurísticas hasta obtener la mejor selección que seamos capaces de encontrar.
El interés en realizar la selección de características reside en que se aumentará la eficiencia, al requerir menos tiempo para construir el clasificador, y que se mejoran los resultados al descartar las características menos influyentes y que sólo aportan ruido. Esto hace también que se reduzcan los costes de mantenimiento y se aumente la interpretabilidad de los datos.
Las funciones de evaluación pueden estar basadas en la consistencia, en la Teoría de la Información, en la distancia o en el rendimiento de clasificadores. Nosotros usaremos el rendimiento promedio de un clasificador $3 - NN$.

2. Descripción de la aplicación de los algoritmos

2.1. Representación de soluciones

Para este problema tenemos varias formas posibles de representar las soluciones:

- Representación binaria: Cada solución está representada por un vector binario de longitud igual al número de características, donde las posiciones seleccionadas tendrán un 1 o `True` y las no seleccionadas un 0 o `False`. Esta opción, que será la que tomaremos, sólo es recomendable si no tenemos restricciones sobre el número de características seleccionadas.

- Representación entera: Cada solución es un vector de tamaño fijo $m \leq n$ con las características seleccionadas. Esta representación sí es adecuada si tenemos restricciones sobre el número de características tomadas ya que no podemos hacerlo con más de m.

- Representación de orden: Cada solución es una permutación de n elementos, ordenados según la importancia de cada característica. Aquí también se maneja el cumplimiento de restricciones pues una vez encontrada la solución, tomaremos sólo las primeras m características.

Se ha de mencionar que en las dos últimas representaciones el espacio de soluciones es mayor que el espacio de búsqueda, justificado en la representación de orden porque da más información (podríamos tomar soluciones de longitud variable), pero que en la representación entera sólo es razonable asumir si tenemos una restricción de longitud fija. Además, otra ventaja de la representación binaria es la facilidad para aplicarle operadores (de vecindario, en posteriores prácticas de cruce...) manteniendo la consistencia.

2.2. Función objetivo

La función objetivo será el porcentaje de acierto en el conjunto de test para el clasificador $3 - NN$ obtenido usando las distancias de los individuos ω en las dimensiones representadas por las características seleccionadas en el vector solución para el conjunto de entrenamiento. El objetivo será maximizar esta función. A la hora de buscar esta solución sólo estaremos manejando los datos de entrenamiento, luego aquí la función objetivo será la media de tasa de acierto para cada uno de los datos de entrenamiento con respecto a todos los demás, por lo que tenemos que usar la técnica de *Leave-One-Out*. Esta técnica consiste en quitar del conjunto de datos cada uno de los elementos, comprobar el acierto o no para este dato en concreto, y devolverlo al conjunto de datos. Así evitamos que los resultados estén sesgados.

```
targetFunction(data_train, categories_train, data_test,
                                    categories_test, solution):
```

```
BEGIN
    num_items ← length(data_test)
    sum_score ← 0

    data_train' ← {col_i from data_train if solution_i is True}
    classifier ← Make3NNClasifier(data_train', categories_train)

    data_test' ← {col_i from data_test if solution_i is True}

    FOR item IN data_test'
        predicted_class ← classifier(item)
        IF predicted_class == categories_test_item THEN
            sum_score ← sum_score + 1
    END

    RETURN sum_score / num_items * 100
END
```

La función de evaluación incluida en el código no la he realizado yo sino que he utilizado el paquete `sklearn` de `Python` por razones de eficiencia. A lo largo de la práctica (en los métodos de búsqueda) haré referencia a una función $getValue(data, categories, solution)$ que representa la función de evaluación con la técnica del *Leave-One-Out* explicada anteriormente.

2.3. Operadores comunes

Entenderemos como vecindario de una solución a los vectores que sólo difieren en una posición. Por tanto, el operador para movernos a una solución vecina consistirá en cambiar una posición determinada:

```
flip(solution, positon):
BEGIN
        neighbour ← copy(solution)
        actual_value ← solution_position
        neighbour_position ← NOT actual_value
        RETURN neighbour
END
```

3. Estructura del método de búsqueda

3.1. Búsqueda local

En el algoritmo de búsqueda local partimos de una solución aleatoria y exploramos el vecindario buscando mejorar la solución actual. El procedimiento será buscar vecino por vecino y quedarnos con el primero de ellos que mejore la solución actual. Esto se conoce como búsqueda local del primer vecino. Nos pararemos cuando no haya ningún vecino que mejore la función objetivo, o bien cuando realicemos 15000 comprobaciones. Es claro que no podemos afirmar que hayamos encontrado la solución global, pero sí al menos que hemos llegado a un óptimo local.

```
localSearch(data, categories) BEGIN
    solution ← generateRandomSolution()
    previous_value ← getValue(data, categories, solution)

    REPEAT
        first_neig ← random{1,..., num_features}
        neighbours ← {flip(solution,i): i=first_neig,...,first_neig-1}
        found_better ← FALSE

        FOR neig IN neighbours
            current_value ← getValue(data, categories, neig)

            if( current_value > previous_value) THEN
                found_better ← TRUE
                solution ← neig
                BREAK

            END
    WHILE( found_better )

    return solution
END
```

Debido a la función objetivo tomada y a la complejidad para sacar la variación del coste de una solución a partir de otra (tendríamos que modificar la matriz de distancias entre los elementos, volver a entrenar al clasificador y evaluarlo), no consideramos ninguna factorización. Esto ocurrirá también para los demás algoritmos.

3.2. Enfriamiento simulado

El método de enfriamiento simulado partirá de una solución también aleatoria y explorará un cierto número de vecinos, moviéndose a ellos si bien mejoran la solución actual, o bien es una solución peor en base a una probabilidad que variará según un valor que se modificará a lo largo de la ejecución del algoritmo, conocido como temperatura.
La idea es que queremos explorar un mayor subespacio del espacio de búsqueda, por eso la probabilidad de aceptar una solución peor será mayor al comienzo de la ejecución, o bien si las soluciones son muy semejantes y no avanzamos.
El algoritmo de ES incluye las siguientes componentes:

- Esquema de enfriamiento: Usaremos el esquema de Cauchy modificado, que nos permite fijar de antemano el número de enfriamientos a realizar. Cada enfriamiento consisitirá en actualizar la temperatura, y con ella, la probabilidad de aceptar una solución peor. Para T_k la temperatura en la iteración k, T_0 la temperatura inicial, T_f la final y $M = \frac{1500}{max\_vecinos}$ el número de iteraciones a realizar:

$$T_{k+1} = \frac{T_k}{1 + \beta \cdot T_k}; \quad \beta = \frac{T_0 - T_f}{M \cdot T_0 \cdot T_f}$$

- La exploración del entorno en cada iteración consistirá en seleccionar aleatoriamente un máximo de $10 \cdot num\_caracteristicas$ vecinos.

- Condición de enfriamiento: Pasaremos a la siguiente iteración, enfriando por tanto la temperatura, cuando hemos recorrido todos los vecinos generados o bien cuando hemos aceptado (ya sea porque mejora a la solución de ese momento o bien debido a la probabilidad en función de la temperatura) un número de vecinos igual al número de características.

- La condición de parada será haber realizado un número máximo de iteraciones (equivalente a que la temperatura sea mayor que la temperatura final) o bien que de entre los vecinos generados no haya acpetado ninguno.

- La temperatura inicial se calcula en función del valor de la solución inicial y dos parámetros μ, ϕ : $T_0 = \frac{\mu \cdot C(S_0)}{-\log(\phi)}$. Tomaremos $\mu = \phi = 0{,}3$

```
simulatedAnnealing(data, categories) BEGIN
    solution ← generateRandomSolution()
    current_value ← getValue(data, categories, solution)

    best_solution ← copy(solution)
    best_value ← current_value
```

```
    T ← initializeTemperature(current_value)

    REPEAT
        neighbours ← {random{1,..., num_features},
                                  size=num_max_neighbours}

            num_succeses ← 0

        FOR j in neighbours IF ( num_succeses < max_sucesses &
                                  num_checks < max_checks)
            last_sol ← flip(solution, j)
            last_value ← getValue(data, categories, neig)

            num_checks ← num_checks + 1
            change ← last_value − current_value

            if( change > 0 OR acept_by_temp(change,T)) THEN
                current_value ← last_value
                solution ← last_sol
                num_succeses ← num_succeses + 1

                if( last_value > best_value ) THEN
                    best_solution ← solution
                    best_value ← current_value

            T ← updateTemperature()
        END
    WHILE( Condicion de parada )
    RETURN solution
END
```

3.3. Búsqueda tabú

Para la versión básica de la lista tabú, los elementos propios de este método que se incluyen son la lista tabú, la exploración del entorno y el criterio de aspiración. El funcionamiento consiste en moverse al mejor vecino posible siempre que ese movimiento no se haya realizado en un pasado próximo (usamos para saberlo la lista tabú) o bien si el movimiento cumple el criterio de aspiración.

- Lista tabú: El tamaño será un tercio del número de características. Almacenará los últimos movimientos realizados e impedirá que se realicen estos movimientos. La

usaremos como si fuera una estructura circular, facilitando saber qué elemento lleva más tiempo colocado y debe salir de la lista.

- Exploración del entorno: En cada iteración se generan 30 soluciones vecinas aleatorias diferentes. Puesto que iremos por orden buscando un vecino que cumpla el criterio de aspiración o ser la mejor solución que no esté en la lista tabú, lo que hago es ordenar a los vecinos por su porcentaje de acierto, facilitando la exploración.
- Criterio de aspiración: El criterio de aspiración es tener una mejor tasa de acierto que la mejor solución encontrada hasta el momento.

```
tabuSearch(data, categories) BEGIN
    solution ← generateRandomSolution()
    current_value ← getValue(data, categories, solution)

    best_solution ← copy(solution)
    best_value ← current_value

    tabu_list ← initializeTabuList()

    FOR j in {1,...,max_iterations}
        characteristics ← {random{1,..., num_features}, size=30
                                without repetition}
        neighbours ← {flip(solution,i): i in characteristics}

        sort (neighbours, characteristics)
              by getValue(data, categories, neighbours)
              order descending


        FOR neigh,char IN (neighbours, characteristics)
            current_value ← getValue(data, categories, neigh)
            IF last_value > best_value THEN
                best_value ← current_value
                solution ← neigh
                add char to tabu_list
                BREAK
            ELSE IF char NOT IN tabu_list
                solution ← neigh
                add char to tabu_list
                BREAK
        END
```

```
    RETURN solution
END
```

Sobre el algoritmo expuesto en clase, he realizado la modificación de ordenar el vector de soluciones y características por lo antes mencionado.

3.4. Búsqueda tabú extendida

La búsqueda tabú extendida que implementaremos parte de la versión básica, a la que se le añaden una memoria a largo plazo y un procedimiento para reinicializar la solución si no se ha obtenido una mejora en los últimos pasos.
La memoria a largo plazo se encargará de saber las características que han sido más veces modificadas. Esto nos permitirá, cuando queramos reiniciar la solución, diversificar el espacio de búsqueda, dando más probabilidad de ser escogidas las características menos utilizadas. La memoria a largo plazo no la reiniciaremos cada vez que lo hagamos con la solución.
Cuando llevemos diez pasos sin haber encontrado una solución mejor que la encontrada hasta entonces, reiniciaremos la solución. Lo haremos de tres maneras:

- Solución aleatoria: Creamos, como al inicio de la búsqueda, un vector solución aleatorio.
- Volvemos a la mejor solución encontrada hasta el momento con la intención de explorar más a fondo su entorno por si nos llevara a una mejor solución.
- Partir de una solución aleatoria generada mediante la memoria a largo plazo.

Además de esto, con la intención de cambiar la estrategia seguida hasta el momento, se cambia la longitud de la lista tabú de forma aleatoria en un 50 %. Si el tamaño de la lista tabú aumenta, estaremos almacenando más posiciones con lo que seremos más restrictivos a la hora de cambiar una característica recientemente modificada, en cambio si la hacemos más pequeña, sí que permitiremos estos cambios.

```
extendedTabuSearch(data, categories) BEGIN
    solution ← generateRandomSolution()
    current_value ← getValue(data, categories, solution)

    best_solution ← copy(solution)
    best_value ← current_value
    last_improvement ← 0

    tabu_list ← initializeTabuList()
    long_term_memory ← {0,...,0: size = num_features}
```

```
    FOR j in {1,...,max_iterations}
        characteristics ← {random{1,..., num_features}, size=30
                                    without repetition}
        neighbours ← {flip(solution,i): i in characteristics}

        sort (neighbours, characteristics)
              by getValue(data, categories, neighbours)
              order descending


        FOR neigh,char IN (neighbours, characteristics)
            current_value ← getValue(data, categories, neigh)
            IF last_value > best_value THEN
                best_value ← current_value
                solution ← neigh
                add char to tabu_list
                last_improvement ← j
                BREAK
            ELSE IF j-last_improvement > 10 THEN
                tabu_list ← changeTabuList()
                solution ← restartSolution()
            ELSE IF char NOT IN tabu_list THEN
                solution ← neigh
                add char to tabu_list
                BREAK
        END

    RETURN solution
END
```

4. Algoritmo de comparación

Como algoritmo de comparación tenemos el algoritmo *greedy* SFS. Partiendo de un vector con ninguna característica seleccionada, exploramos por el entorno y nos quedamos con el vecino que genera una mejor tasa de acierto. Repetimos este proceso hasta que ningún vecino aporta una mejora a la solución obtenida.

```
greedySFS(data, categories) BEGIN
    solution ← {0,...,0: size = num_features}
    current_value ← getValue(data, categories, solution)
```

```
    REPEAT
        neighbours ← {flip(solution,i): i in characteristics}

        best_value ← máx_neighbours getValue(data, categories, ·)

        IF best_value > current_value THEN
            solution ← argmax_neighbours getValue(data, categories, ·)

    WHILE(best_value > current_value)

    RETURN solution
END
```

5. Procedimiento para desarrollar la práctica

El código de la práctica está realizado en `Python`. Comencé por el algoritmo del KNN y el algoritmo *greedy* SFS. Sin embargo, viendo los tiempos (la iteración en la base de datos de Arrythmia duraba en torno a 10 minutos) me decanté por usar el módulo de `Python` `sklearn` para realizar la normalización de los datos, la validación cruzada y el algoritmo KNN, permitiendo entrenar el clasificador con los datos y obtener un *score* al comprobar si concuerda la clase predicha con la original.

Los paquetes utilizados son `sklearn`, como se ha mencionado, `scipy` para leer de una manera sencilla la base de datos, `numpy` para el manejo de vectores y matrices y tratar que sea algo más eficiente en lugar de las listas de `Python` y `ctype` para importar el generador de números aleatorios en `C` disponible en la página web de la asignatura. La semilla con la que he realizado las ejecuciones es 3141592, insertada tanto en el generador en `C` como por el generador de número de aleatorios de `numpy`. He usado los dos porque pretendía usar el primero, que es con el que se realizan las particiones, pero al llegar a los métodos que usan los generadores de números pseudoaleatorios en su funcionamiento me di cuenta de que tendría que repetir el código de importación del módulo en `C` para cada método, por lo que opté por usar en los métodos el `random` de `numpy`. Mientras he ido realizando la práctica he ido creando funciones para permitir un más cómodo manejo del programa intentando que el código fuese entendible.

5.1. Ejecución del programa

La salida de cada ejecución (10 iteraciones de un algoritmo con una base de datos) se puede elegir entre mostrar por pantalla o redirigir a un archivo `.csv` para manejarlo

posteriormente, por ejemplo para incluir la tabla en LaTeX.
Los parámetros que acepta el programa son:

- Base de datos: Será una letra `W,L,A` que representa cada una de las bases de datos a utilizar. Este parámetro es el único obligatorio.
- Algoritmo utilizado: Por defecto es el KNN. Para introducir uno distinto, se usa `-a` seguido de una letra entre `K,G,L,S,T,E` que se corresponden con KNN, *greedy* SFS, *local search*, *simulated annealing*, *tabu search* y *extended tabu search*, respectivamente.
- Semilla. Para incluir una semilla, se añade `-seed` seguido del número que usaremos como semilla. Por defecto es 3141592.
- Salida por pantalla o a fichero. Se utiliza con el parámetro opcional `-write` para escribir en un fichero en una carpeta llamada `Resultados`. El nombre del fichero será la primera letra de la base de datos utilizada seguida por las iniciales del algoritmo. Incluye también la media para cada columna en la última fila.

Por tanto, la ejecución del programa se hará de la siguiente manera:

```
python Practica1.py base_de_datos [-a algoritmo -seed semilla -write T/F ]
```

Si por ejemplo queremos lanzar la base de datos de `WDBC` con la búsqueda local, semilla 123456 y que los resultados se muestren por pantalla, escribimos

```
python Practica1.py W -a L -seed 123456
```

Si optamos por la base de datos `Arrythmia` con la búsqueda tabú extendida y guardar el resultado en un fichero:

```
python Practica1.py A -a E -write True
```

6. Experimentos y análisis de resultados

6.1. KNN

	WDBC				Movement Libras				Arrythmia			
	%Clas. in	%Clas. out	red	T	%Clas. in	%Clas. out	%red.	T	%Clas. in	%Clas. out	%red.	T
Particion 1-1	97,5352	67,3684	0	0,3813	71,1111	15,5556	0	0,3754	64,5833	63,4021	0	0,7567
Particion 1-2	95,7895	64,4366	0	0,3878	66,1111	17,7778	0	0,3864	63,4021	45,3125	0	0,7367
Particion 2-1	95,7746	67,0175	0	0,3945	66,6667	18,8889	0	0,3739	64,5833	44,3299	0	0,7208
Particion 2-2	95,4386	66,9014	0	0,3922	64,4444	17,7778	0	0,3738	65,4639	63,5417	0	0,7409
Particion 3-1	97,5352	66,6667	0	0,4139	65	13,3333	0	0,3863	63,5417	43,8144	0	0,7291
Particion 3-2	96,1404	67,6056	0	0,3921	68,8889	15,5556	0	0,3732	65,9794	63,5417	0	0,7653
Particion 4-1	95,7746	68,7719	0	0,3875	72,2222	15	0	0,3756	66,6667	44,3299	0	0,7553
Particion 4-2	95,7895	67,9577	0	0,3901	60	15	0	0,3726	63,4021	63,5417	0	0,7374
Particion 5-1	97,5352	66,6667	0	0,3917	65,5556	18,3333	0	0,3727	64,0625	46,9072	0	0,7292
Particion 5-2	96,1404	64,0845	0	0,4008	66,1111	15,5556	0	0,3716	68,5567	43,75	0	0,7679
Media	96,3453	66,7477	0	0,3932	66,6111	16,2778	0	0,3761	65,0247	52,2471	0	0,7439

6.2. SFS

	WDBC				Movement Libras				Arrythmia			
	%Clas. in	%Clas. out	red	T	%Clas. in	%Clas. out	%red.	T	%Clas. in	%Clas. out	%red.	T
Particion 1-1	97,5352	92,6316	86,6667	48,1518	73,8889	62,2222	90	224,3986	78,6458	68,5567	98,5612	342,8862
Particion 1-2	96,4912	93,662	90	42,1664	72,2222	72,2222	91,1111	200,0296	81,9588	74,4792	97,482	562,2888
Particion 2-1	96,831	93,6842	90	45,4098	70,5556	71,1111	88,8889	248,4451	78,125	69,0722	98,2014	419,6417
Particion 2-2	97,8947	96,831	80	74,2571	80,5556	67,2222	84,4444	330,4517	77,3196	68,75	97,482	559,0299
Particion 3-1	97,5352	94,7368	83,3333	59,1569	73,8889	69,4444	92,2222	179,5272	82,2917	74,2268	96,7626	678,8885
Particion 3-2	97,193	95,4225	86,6667	50,8267	73,3333	68,8889	91,1111	203,2817	81,9588	74,4792	97,482	553,0682
Particion 4-1	95,0704	93,3333	90	41,7483	75	71,6667	87,7778	265,5968	78,125	69,0722	97,482	535,9987
Particion 4-2	98,2456	95,0704	86,6667	53,662	70	67,2222	90	219,4789	73,7113	67,1875	98,2014	417,0979
Particion 5-1	98,9437	96,1404	76,6667	74,2299	71,6667	69,4444	91,1111	206,3513	77,6042	69,0722	98,9209	268,0201
Particion 5-2	96,4912	92,2535	90	40,5726	72,7778	67,2222	88,8889	251,1997	82,9897	77,0833	96,7626	736,5666
Media	97,2231	94,3766	86	53,0181	73,3889	68,6667	89,5556	232,876	79,273	71,1979	97,7338	507,3487

6.3. Local Search

	WDBC				Movement Libras				Arrythmia			
	%Clas. in	%Clas. out	red	T	%Clas. in	%Clas. out	%red.	T	%Clas. in	%Clas. out	%red.	T
Particion 1-1	95,7746	95,0877	46,6667	17,171	73,3333	73,8889	55,5556	84,7747	69,7917	64,433	52,518	243,7015
Particion 1-2	97,8947	95,7746	60	24,825	68,8889	68,8889	42,2222	35,9078	67,5258	65,1042	48,5612	402,6252
Particion 2-1	95,7746	96,8421	53,3333	18,3397	68,3333	67,7778	46,6667	74,8362	70,8333	63,9175	48,9209	238,1666
Particion 2-2	99,2982	95,0704	53,3333	29,5761	71,1111	72,2222	53,3333	44,242	69,0722	62,5	48,5612	169,2401
Particion 3-1	98,2394	95,4386	40	27,5047	73,8889	67,2222	47,7778	51,2763	73,4375	69,0722	55,7554	177,1898
Particion 3-2	97,193	97,1831	50	22,5515	70	69,4444	52,2222	46,0677	70,6186	63,0208	43,8849	367,7941
Particion 4-1	98,2394	97,5439	46,6667	19,7762	63,3333	72,7778	52,2222	70,8918	70,3125	67,0103	51,0791	312,14
Particion 4-2	97,193	95,7746	50	41,9689	66,6667	70	42,2222	44,835	67,5258	64,5833	43,1655	212,3571
Particion 5-1	98,2394	96,8421	40	19,8152	73,3333	72,2222	50	44,7807	69,2708	66,4948	49,6403	210,3203
Particion 5-2	94,0351	95,0704	43,3333	27,7983	69,4444	68,8889	56,6667	116,2701	68,0412	65,625	50,7194	257,8143
Media	97,1882	96,0627	48,3333	24,9326	69,8333	70,3333	49,8889	61,3882	69,6429	65,1761	49,2808	259,1349

6.4. Simulated Annealing

	WDBC				Movement Libras				Arrythmia			
	%Clas. in	%Clas. out	red	T	%Clas. in	%Clas. out	%red.	T	%Clas. in	%Clas. out	%red.	T
Particion 1-1	98,9437	95,0877	36,6667	257,0997	70	61,6667	45,5556	650,7325	68,75	62,8866	53,2374	2,394,1383
Particion 1-2	96,8421	96,831	43,3333	247,0133	72,7778	67,2222	42,2222	644,7271	69,0722	62,5	52,1583	2,307,905
Particion 2-1	96,1268	96,1404	43,3333	246,5534	70,5556	67,2222	40	655,5704	67,7083	63,9175	46,0432	2,423,5679
Particion 2-2	97,5439	96,4789	46,6667	242,0824	75	68,8889	50	631,5949	67,0103	66,1458	47,1223	2,450,4524
Particion 3-1	98,2394	96,8421	56,6667	242,1037	70,5556	66,1111	57,7778	607,3304	69,2708	67,5258	47,8417	2,069,8188
Particion 3-2	96,8421	96,831	40	242,5344	70	75	46,6667	636,6247	70,6186	60,9375	54,6763	2,032,5481
Particion 4-1	96,831	94,386	50	236,8387	67,2222	65	52,2222	625,6984	70,8333	68,0412	48,5612	2,190,608
Particion 4-2	97,8947	94,3662	60	243,4523	71,1111	71,6667	53,3333	647,37	67,5258	62,5	48,9209	2,119,1946
Particion 5-1	97,1831	96,4912	36,6667	243,9444	65	70,5556	53,3333	726,6959	70,3125	61,3402	46,7626	2,061,4179
Particion 5-2	97,193	95,7746	53,3333	238,9827	70,5556	66,6667	63,3333	705,6328	68,0412	67,1875	48,9209	2,086,0557
Media	97,364	95,9229	46,6667	244,0605	70,2778	68	50,4444	653,1977	68,9143	64,2982	49,4244	2,213,5706

6.5. Tabu Search

	WDBC				Movement Libras				Arrythmia			
	%Clas. in	%Clas. out	red	T	%Clas. in	%Clas. out	%red.	T	%Clas. in	%Clas. out	%red.	T
Particion 1-1	99,2958	92,9825	46,6667	1,892,1081	78,3333	64,4444	51,1111	1,575,1115	76,5625	65,4639	55,036	1,663,1604
Particion 1-2	98,2456	96,831	53,3333	1,956,6236	75,5556	71,6667	54,4444	1,642,2937	72,1649	68,2292	54,3165	1,827,9728
Particion 2-1	98,5915	95,4386	53,3333	1,902,1341	72,2222	66,6667	55,5556	1,589,2539	74,4792	63,9175	51,4388	1,800,5556
Particion 2-2	98,2456	96,831	63,3333	1,871,5005	75	73,3333	64,4444	1,554,1889	72,6804	67,1875	50	1,861,8212
Particion 3-1	98,2394	95,7895	40	1,901,2344	72,7778	72,2222	61,1111	1,561,8718	72,9167	68,5567	55,3957	1,840,2515
Particion 3-2	98,9474	96,1268	63,3333	1,886,4052	76,6667	71,6667	50	1,597,1963	71,134	63,5417	45,6835	1,903,2971
Particion 4-1	98,2394	95,4386	56,6667	1,880,3792	78,8889	72,7778	53,3333	1,592,4958	70,8333	66,4948	51,0791	1,764,4242
Particion 4-2	98,9474	95,0704	43,3333	1,889,3204	71,1111	71,1111	61,1111	1,538,8533	70,6186	63,0208	47,8417	1,716,1352
Particion 5-1	98,9437	96,1404	43,3333	1,892,5992	76,6667	72,2222	46,6667	1,579,7857	72,9167	67,0103	55,3957	1,619,0616
Particion 5-2	97,8947	95,4225	46,6667	1,894,4346	74,4444	69,4444	61,1111	1,539,2319	72,1649	64,5833	50	1,618,1602
Media	98,5591	95,6071	51	1,896,6739	75,1667	70,5556	55,8889	1,577,0288	72,6471	65,8005	51,6187	1,761,484

6.6. Extended Tabu Search

	WDBC				Movement Libras				Arrythmia			
	%Clas. in	%Clas. out	red	T	%Clas. in	%Clas. out	%red.	T	%Clas. in	%Clas. out	%red.	T
Particion 1-1	99,2958	92,9825	46,6667	1,973,0726	81,1111	66,1111	51,1111	1,921,3485	69,2708	63,4021	52,518	1,849,7371
Particion 1-2	98,2456	94,7183	66,6667	2,351,6747	72,2222	63,3333	54,4444	2,066,7994	71,6495	64,5833	52,8777	1,853,7098
Particion 2-1	98,2394	95,4386	50	2,252,4871	73,3333	67,7778	53,3333	2,023,6205	71,875	66,4948	48,9208	1,924,6708
Particion 2-2	97,8947	96,4789	53,3333	2,259,8204	72,2222	71,6667	54,4444	2,037,1413	68,5567	62,5	44,2446	2,043,1038
Particion 3-1	97,8873	96,1404	43,3333	2,277,9373	75	71,6667	42,2222	2,088,212	68,75	65,4639	50,3597	2,094,3816
Particion 3-2	97,8947	95,4225	33,3333	2,432,1442	71,6667	70,5556	54,4444	2,164,1564	74,7423	65,625	51,4388	2,051,756
Particion 4-1	97,8873	96,4912	56,6667	2,398,2382	71,1111	70	61,1111	2,140,9972	72,9167	70,1031	51,4388	2,050,121
Particion 4-2	98,9474	95,4225	43,3333	2,373,8497	76,1111	72,7778	47,7778	2,136,7662	66,4948	63,5417	52,518	1,996,8307
Particion 5-1	98,9437	95,7895	46,6667	2,357,7207	74,4444	68,3333	53,3333	2,084,4679	73,4375	68,5567	47,8417	1,853,0846
Particion 5-2	97,5439	94,3662	53,3333	2,141,4067	74,4444	70,5556	53,3333	2,104,5299	69,0722	65,1042	51,7985	1,915,6438
Media	98,278	95,3251	49,3333	2,281,8352	74,1667	69,2778	52,5556	2,076,8039	70,6765	65,5375	50,3957	1,963,3039

6.7. Comparación

	WDBC				Movement Libras				Arrythmia			
	%Clas. in	%Clas. out	%red.	T	%Clas. in	%Clas. out	%red.	T	%Clas. in	%Clas. out	%red.	T
3-NN	96.34532	66.7477	0.0	0.3932	66.6111	16.2778	0.0	0.3761	65.0247	52.2471	0.0	0.7439
SFS	97.2231	94.3766	86.0	53.0181	73.3889	68.6667	89.5556	232.8760	79.273	71.1979	97.7338	507.3487
BL	97.1882	96.0627	48.3333	24.9326	69.8333	70.3333	49.8889	61.3882	69.6429	65.1761	49.2808	259.1349
ES	97.364	95.9229	46.6667	244.06049	70.2778	68.0	50.4444	653.1977	68.9143	64.2982	49.4244	2213.5706
BT básica	98.5591	95.6071	51.0	1896.6739	75.16667	70.5556	55.8889	1577.0288	72.6471	65.8005	51.6187	1761.484
BT extendida	98.278	95.3251	49.3333	2281.8352	74.16667	69.2778	52.5556	2076.8039	70.6765	65.5375	50.3957	1963.3039

7. Bibliografía

- Módulo en `scikit` para KNN
- Para realizar las tablas Manual PGFPLOTSTABLE