Práctica 2 Metaheurísticas.
Búsquedas multiarranque para el problema de la selección de características

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

19 de abril de 2016

Curso 2015-2016
Problema de Selección de Características.
Grupo de prácticas: Viernes 17:30-19:30
Quinto curso del Doble Grado en Ingeniería Informática y Matemáticas.

Algoritmos considerados:

1. Búsqueda Multiarranque Básica
2. GRASP
3. ILS

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 etiqueta, 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 diese 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, tratamos, en base a unos datos, hallar la mejor f posible. De esto trata el aprendizaje supervisado: 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 características.

- 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 a favor de la clase o etiqueta original, al contar siempre con un voto la clase verdadera.
La implementación de la función objetivo (obtener el score para Test) la he realizado

basándome en el código paralelizado realizado para CUDA por Alejandro García Montoro para la función de *Leave One Out*. El pseudocódigo incluido se trata del esquema seguido para cada proceso, puesto que el método en `Python` para pasarle a la GPU los datos de entrenamiento, test, categorías y un puntero con la solución no tiene mayor interés.

```
targetFunction(data_train, categories_train,
               data_test, categories_test):
BEGIN
    test ← Num. Process
    num_test ← length(data_test)

    exit if test > num_test
    my_features ← data_test[test]

    k_nearest ← initialize3Neighbours

    FOR item IN data_train
        distance ← computeDistance(my_features, item)
        k_nearest ← update(item, distance)
    END

    class ← poll(classes of k_nearest)

    RETURN class == categories_test[test]
END
```

Esto en `CUDA` lo que hace es guardarnos, para cada proceso, si se ha acertado o no. Posteriormente se pasa el vector con cada resultado (cada ejecución de este código) de nuevo a `Python` y se calcula el porcentaje de aciertos. Nótese que no se realiza la proyección por las características seleccionadas, esto lo hacemos al pasar los datos.
Para la función de evaluación de las posibles soluciones que se van generando durante la búsqueda utilizo el método realizado por Alejandro García Montoro para usar CUDA. El algoritmo es similar a éste, pero incluye *Leave One Out*:

```
targetFunctionLeaveOneOut(data_train, categories_train):
BEGIN
    sample ← Num. Process
    num_samples ← length(data_train)

    exit if sample > num_samples
    my_features ← data_train[sample]

    k_nearest ← initialize3Neighbours
```

```
    FOR item IN data_train
       if item ≠ sample
        distance ← computeDistance(my_features, item)
        k_nearest ← update(item, distance)
    END

    class ← poll(classes of k_nearest)

    RETURN class == categories_train[sample]
END
```

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
```

En esta práctica también es común a todos los métodos el método de búsqueda local. Para favorecer la reducción de características, considero una mejor solución aquella que con la misma calidad en términos de porcentaje de aciertos, tenga un menor número de características seleccionadas. Esto nos permitirá posteriormente hacer un análisis sobre la reducción de características en el sentido de que sólo se llegará a reducir las características con los métodos con los que se llegue a buenas soluciones cuyos vecinos, si no son peores, al menos tienen menos características activadas.
Como diferencia con respecto a la primera práctica, ahora la solución inicial es un parámetro y no se genera aleatoriamente dentro del método, ya que depende del algoritmo de búsqueda por trayectorias múltiples que usemos:

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

```
        FOR neig IN neighbours
            IF neigh is better than solution THEN
                found_better ← TRUE
                solution ← neig
                BREAK

            END
    WHILE( found_better )

    return solution
END
```

Donde que el vecino sea mejor que la solución actual significa que tiene una mayor tasa de acierto en el conjunto de datos de entrenamiento, o que la tasa es la misma y tiene menos características seleccionadas.

3. Estructura del método de búsqueda

3.1. Búsqueda multiarranque básica

Este método consiste en la realización de diferentes búsquedas locales que parten de soluciones aleatorias. Para ello, simplemente se lanzan, como se indica en el guión de prácticas, 25 búsquedas locales y devolvemos la mejor solución encontrada. El criterio para considerar que una solución es mejor que otra se basa, como dentro de la búsqueda local, en encontrar el vector de características que ofrezca una mejor tasa de acierto y, de entre ellas, la que tenga menos características seleccionadas. Los resultados esperados serán similares a las mejores ejecuciones del apartado de búsqueda local en la práctica anterior. Se podrá observar también si el hecho de incluir aceptar como mejores soluciones con igual tasa de acierto y menor número de características beneficia o perjudica a la tasa de acierto fuera de los datos de entrenamiento.

```
BMB(data, categories) BEGIN
    num_features ← length ( data[0] )
    best_solution ← {F,...,F}

    FOR i = 1,...,25
        solution ← random({True,False},
                                    size = num_features)

        solution ← localSearch(data, categories, solution)
        IF solution is better than best_solution THEN
```

```
            best_solution ← solution
        END

        return best_solution
    END
```

3.2. GRASP

El esquema general del algoritmo `GRASP` es similar al de `BMB`, con la única salvedad de que la solución inicial para cada una de las iteraciones no es un vector aleatorio sino que se trata de una solución aportada por un algoritmo *greedy*. Esto repercute en que es necesario un tiempo mayor de ejecución al ser más costoso lógicamente ejecutar el algoritmo *greedy* que obtener una solución aleatoria. Sin embargo, este tiempo no se pierde en vano, pues resulta lógico pensar *a priori* que si partimos de una buena solución y entonces aplicamos búsqueda local, se llegará a una solución mejor que partiendo de una aleatoria.

```
GRASP(data, categories) BEGIN
    num_features ← length ( data[0] )
    best_solution ← {F,...,F}

    FOR i = 1,...,25
        solution ← getGreedySolution(data, categories)

        solution ← localSearch(data, categories, solution)
        IF solution is better than best_solution THEN
            best_solution ← solution
    END

    return best_solution
END
```

El interés de este método reside, pues en la obtención de la solución *greedy*. Aquí también hay diferencias con respecto a la primera práctica. Ahora no se trata de ir seleccionando la mejor característica cada vez, sino de ir seleccionando una característica al azar de las que sobrepasen un umbral. Esto añade mayor variabilidad al espacio de búsqueda explorado, con lo que tenemos mayor probabilidad de encontrar una mejor solución.

```
getGreedySolution(data, categories) BEGIN
    current_solution ← {0,...,0: size = num_features}
    tolerance ← 0.3

    REPEAT
        char ← getCharWithThreshold(data,categories,
```

```
                                                    solution, tolerance)

        IF flip(solution, char) is better than solution THEN
            solution ← flip(solution, char)

    WHILE flip(solution, char) is better than solution

    RETURN solution
END
```

Aún falta por comentar cómo se obtiene la característica con la que seleccionaremos una característica al azar de entre las que cumplan una cierta condición, que consiste en tener un porcentaje de acierto entre el máximo valor del vecindario y éste menos un 30 % de la diferencia entre él y el menor valor del vecindario. El porcentaje de la diferencia aceptada es lógicamente variable, con un valor cercano a 0 % se trataría del *greedy* `SFS`, mientras que con un valor del 100 % se trata de una selección de un vecino aleatorio, y parando el algoritmo cuando seleccionásemos a un vecino peor que la solución actual.

```
getCharWithThreshold(data, categories, solution, tolerance) BEGIN
    values ← vector( size = num_features )
    FOR i IN {1,..., num_features}

        neig ← neighbours[i]
        values[i] ← score(data, categories, neig)

    END

    max ← max(values)
    min ← min(values)
    above_threshold ← {i ∈ {1,...,N:
                            values[i] > max - tolerance(max-min)}
    char ← random(above_threshold)

    RETURN char
END
```

3.3. ILS

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 3.5.1`. Comencé por el algoritmo del KNN y el algoritmo *greedy* SFS. Sin embargo, viendo los tiempos (la iteración en la base de datos de Arrhythmia 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.
- `-h` o `--help` para mostrar la ayuda y cómo se introducen los parámetros.

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 `Arrhythmia` con la búsqueda tabú extendida y guardar el resultado en un fichero:

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

Para mostrar la introducción de parámetros:

```
python Practica1.py --help
```

6. Experimentos y análisis de resultados

6.1. Descripción de los casos

Los casos del problema planteados son tres, cada uno de ellos asociado a una base de datos:

- WDBC: Base de datos con los atributos estimados a partir de una imagen de una aspiración de una masa en la mama. Tiene 569 ejemplos, 30 atributos y debemos clasificar cada individuo en dos valores.
- Movement Libras: Base de datos con la representación de los movimientos de la mano en el lenguaje de signos LIBRAS. Tiene 360 ejemplos y consta de 91 atributos.
- Arrhythmia: Contiene datos de pacientes durante la presencia y ausencia de arritmia cardíaca. Tiene 386 ejemplos y 278 atributos para categorizar en 5 clases.

Hay que mencionar que tanto la ejecución de la búsqueda tabú como la búsqueda tabú extendida se han realizado no con 15000 comprobaciones como indica el guión de la práctica sino con 5000 debido a que lleva demasiado tiempo ejecutarlas al no tener otra condición de parada que realizar estas ejecuciones. En la medida de los tiempos se ha incluido la calificación del conjunto de test, con lo que se añade un poco de tiempo extra en cada iteración, que no es representativo con respecto a lo que se tarda en obtener la solución. Esto nos da una idea sobre la influencia del tamaño de los datos sobre cada evaluación de la solución actual y la influencia que esto tiene en el tiempo para obtener la solución, pues no sólo influye el tamaño del espacio de búsqueda.

6.2. Resultados

6.2.1. KNN

	WDBC				Movement Libras				Arrhythmia			
	%Clas. in	%Clas. out	red	T	%Clas. in	%Clas. out	%red.	T	%Clas. in	%Clas. out	%red.	T
Particion 1-1	95,4225	96,1404	0	0,3865	70	62,2222	0	0,388	63,5417	64,433	0	0,4155
Particion 1-2	97,8947	94,7183	0	0,3851	66,6667	64,4444	0	0,3824	63,4021	64,5833	0	0,4285
Particion 2-1	95,7746	96,1404	0	0,3794	61,1111	71,1111	0	0,3855	64,5833	60,8247	0	0,4463
Particion 2-2	96,8421	96,1268	0	0,3861	71,6667	67,2222	0	0,3858	61,8557	67,7083	0	0,4315
Particion 3-1	95,7746	95,7895	0	0,3842	61,6667	75,5556	0	0,3853	66,1458	63,9175	0	0,4214
Particion 3-2	96,1404	97,8873	0	0,4135	67,7778	67,2222	0	0,3912	63,9175	65,625	0	0,4154
Particion 4-1	95,4225	96,8421	0	0,4018	72,7778	71,6667	0	0,3876	62,5	66,4948	0	0,4131
Particion 4-2	94,7368	96,1268	0	0,4091	65,5556	70	0	0,3882	65,9794	65,1042	0	0,4158
Particion 5-1	97,1831	95,7895	0	0,4236	74,4444	68,8889	0	0,3959	65,1042	64,9485	0	0,418
Particion 5-2	95,0877	97,1831	0	0,4317	66,6667	69,4444	0	0,3947	63,9175	66,1458	0	0,4202
Media	96,0279	96,2744	0	0,4001	67,8333	68,7778	0	0,3885	64,0947	64,9785	0	0,4226
Max	97,8947	97,8873	0	0,4317	74,4444	75,5556	0	0,3959	66,1458	67,7083	0	0,4463
Min	97,8947	97,8873	0	0,4317	74,4444	75,5556	0	0,3959	66,1458	67,7083	0	0,4463
Desv. Típica	0,9438	0,8198	0	0,0176	4,227	3,5815	0	$4{,}1 \cdot 10^{-3}$	1,3199	1,743	0	$9{,}7 \cdot 10^{-3}$

6.2.2. SFS

	WDBC				Movement Libras				Arrhythmia			
	%Clas. in	%Clas. out	red	T	%Clas. in	%Clas. out	%red.	T	%Clas. in	%Clas. out	%red.	T
Particion 1-1	78,6458	68,5567	98,5612	342,8862	78,6458	68,5567	98,5612	342,8862	78,6458	68,5567	98,5612	342,8862
Particion 1-2	81,9588	74,4792	97,482	562,2888	81,9588	74,4792	97,482	562,2888	81,9588	74,4792	97,482	562,2888
Particion 2-1	78,125	69,0722	98,2014	419,6417	78,125	69,0722	98,2014	419,6417	78,125	69,0722	98,2014	419,6417
Particion 2-2	77,3196	68,75	97,482	559,0299	77,3196	68,75	97,482	559,0299	77,3196	68,75	97,482	559,0299
Particion 3-1	82,2917	74,2268	96,7626	678,8885	82,2917	74,2268	96,7626	678,8885	82,2917	74,2268	96,7626	678,8885
Particion 3-2	81,9588	74,4792	97,482	553,0682	81,9588	74,4792	97,482	553,0682	81,9588	74,4792	97,482	553,0682
Particion 4-1	78,125	69,0722	97,482	535,9987	78,125	69,0722	97,482	535,9987	78,125	69,0722	97,482	535,9987
Particion 4-2	73,7113	67,1875	98,2014	417,0979	73,7113	67,1875	98,2014	417,0979	73,7113	67,1875	98,2014	417,0979
Particion 5-1	77,6042	69,0722	98,9209	268,0201	77,6042	69,0722	98,9209	268,0201	77,6042	69,0722	98,9209	268,0201
Particion 5-2	82,9897	77,0833	96,7626	736,5666	82,9897	77,0833	96,7626	736,5666	82,9897	77,0833	96,7626	736,5666
Media	79,273	71,1979	97,7338	507,3487	79,273	71,1979	97,7338	507,3487	79,273	71,1979	97,7338	507,3487

6.2.3. Búsqueda multiarranque básica

	WDBC				Movement Libras				Arrhythmia			
	%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	66,6667	576,9335	64,5833	63,9175	0	0,4528	64,5833	63,9175	0	0,4528
Particion 1-2	97,8947	94,3662	53,3333	605,2306	62,8866	65,625	0	0,5587	62,8866	65,625	0	0,5587
Particion 2-1	98,2394	95,4386	56,6667	513,1967	63,5417	65,4639	0	0,6544	63,5417	65,4639	0	0,6544
Particion 2-2	97,8947	96,831	60	522,2431	64,9485	65,1042	0	0,5755	64,9485	65,1042	0	0,5755
Particion 3-1	97,8873	95,7895	46,6667	515,0343	65,1042	65,9794	0	0,462	65,1042	65,9794	0	0,462
Particion 3-2	97,8947	96,4789	60	463,9336	67,0103	64,0625	0	0,4653	67,0103	64,0625	0	0,4653
Particion 4-1	97,8873	95,4386	43,3333	516,7684	66,1458	60,8247	0	0,4652	66,1458	60,8247	0	0,4652
Particion 4-2	98,9474	93,662	66,6667	533,918	65,9794	64,5833	0	0,4604	65,9794	64,5833	0	0,4604
Particion 5-1	98,9437	96,1404	53,3333	547,0582	60,4167	67,5258	0	0,4589	60,4167	67,5258	0	0,4589
Particion 5-2	97,8947	96,1268	50	629,6693	65,9794	60,9375	0	0,4559	65,9794	60,9375	0	0,4559
Media	98,278	95,3254	55,6667	542,3986	64,6596	64,4024	0	0,5009	64,6596	64,4024	0	0,5009
Max	99,2958	96,831	66,6667	629,6693								
Min	99,2958	96,831	66,6667	629,6693								
Desv. Típica	0,5312	1,1967	7,461	46,553								

6.2.4. GRASP

	WDBC				Movement Libras				Arrhythmia			
	%Clas. in	%Clas. out	red	T	%Clas. in	%Clas. out	%red.	T	%Clas. in	%Clas. out	%red.	T
Particion 1-1	97,5352	94,0351	0	0,418	64,5833	63,9175	0	0,4528	64,5833	63,9175	0	0,4528
Particion 1-2	95,7895	95,7746	0	0,396	62,8866	65,625	0	0,5587	62,8866	65,625	0	0,5587
Particion 2-1	95,7746	96,8421	0	0,3927	63,5417	65,4639	0	0,6544	63,5417	65,4639	0	0,6544
Particion 2-2	95,4386	97,1831	0	0,3985	64,9485	65,1042	0	0,5755	64,9485	65,1042	0	0,5755
Particion 3-1	97,5352	96,8421	0	0,3917	65,1042	65,9794	0	0,462	65,1042	65,9794	0	0,462
Particion 3-2	96,1404	97,1831	0	0,392	67,0103	64,0625	0	0,4653	67,0103	64,0625	0	0,4653
Particion 4-1	95,7746	97,193	0	0,3948	66,1458	60,8247	0	0,4652	66,1458	60,8247	0	0,4652
Particion 4-2	95,7895	96,1268	0	0,4017	65,9794	64,5833	0	0,4604	65,9794	64,5833	0	0,4604
Particion 5-1	97,5352	95,0877	0	0,431	60,4167	67,5258	0	0,4589	60,4167	67,5258	0	0,4589
Particion 5-2	96,1404	95,0704	0	0,4003	65,9794	60,9375	0	0,4559	65,9794	60,9375	0	0,4559
Media	96,3453	96,1338	0	0,4017	64,6596	64,4024	0	0,5009	64,6596	64,4024	0	0,5009
Max	97,5352	97,193	0	0,431								
Min	97,5352	97,193	0	0,431								
Desv. Típica	0,8014	1,0529	0	0,0122								

6.2.5. ILS

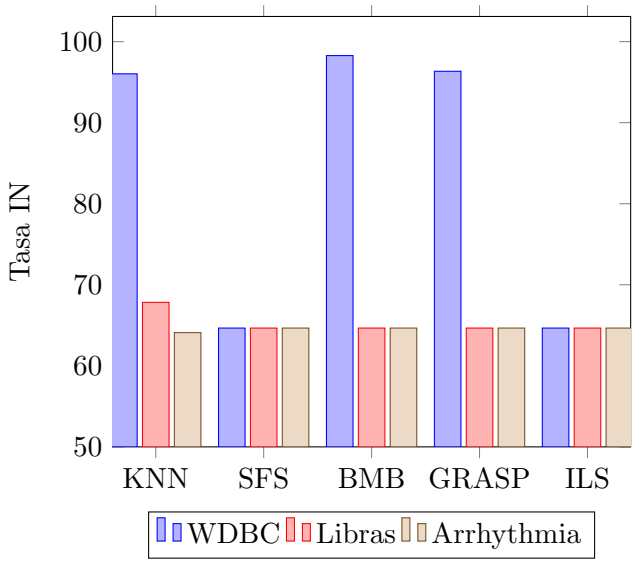
	WDBC				Movement Libras				Arrhythmia			
	%Clas. in	%Clas. out	red	T	%Clas. in	%Clas. out	%red.	T	%Clas. in	%Clas. out	%red.	T
Particion 1-1	64,5833	63,9175	0	0,4528	64,5833	63,9175	0	0,4528	64,5833	63,9175	0	0,4528
Particion 1-2	62,8866	65,625	0	0,5587	62,8866	65,625	0	0,5587	62,8866	65,625	0	0,5587
Particion 2-1	63,5417	65,4639	0	0,6544	63,5417	65,4639	0	0,6544	63,5417	65,4639	0	0,6544
Particion 2-2	64,9485	65,1042	0	0,5755	64,9485	65,1042	0	0,5755	64,9485	65,1042	0	0,5755
Particion 3-1	65,1042	65,9794	0	0,462	65,1042	65,9794	0	0,462	65,1042	65,9794	0	0,462
Particion 3-2	67,0103	64,0625	0	0,4653	67,0103	64,0625	0	0,4653	67,0103	64,0625	0	0,4653
Particion 4-1	66,1458	60,8247	0	0,4652	66,1458	60,8247	0	0,4652	66,1458	60,8247	0	0,4652
Particion 4-2	65,9794	64,5833	0	0,4604	65,9794	64,5833	0	0,4604	65,9794	64,5833	0	0,4604
Particion 5-1	60,4167	67,5258	0	0,4589	60,4167	67,5258	0	0,4589	60,4167	67,5258	0	0,4589
Particion 5-2	65,9794	60,9375	0	0,4559	65,9794	60,9375	0	0,4559	65,9794	60,9375	0	0,4559
Media	64,6596	64,4024	0	0,5009	64,6596	64,4024	0	0,5009	64,6596	64,4024	0	0,5009

6.2.6. Comparación

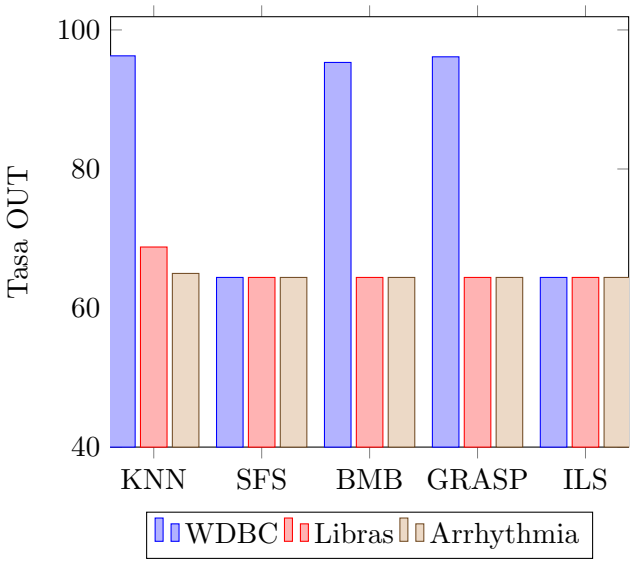
	WDBC				Movement Libras				Arrhythmia			
	%Clas. in	%Clas. out	%red.	T	%Clas. in	%Clas. out	%red.	T	%Clas. in	%Clas. out	%red.	T
KNN	96.0279	96.2744	0.0000	0.4001	67.8333	68.7778	0.0000	0.3885	64.0947	64.9785	0.0000	0.4226
SFS	64.6596	64.4024	0.0000	0.5009	64.6596	64.4024	0.0000	0.5009	64.6596	64.4024	0.0000	0.5009
BMB	98.2780	95.3254	55.6667	542.3986	64.6596	64.4024	0.0000	0.5009	64.6596	64.4024	0.0000	0.5009
GRASP	96.3453	96.1338	0.0000	0.4017	64.6596	64.4024	0.0000	0.5009	64.6596	64.4024	0.0000	0.5009
ILS	64.6596	64.4024	0.0000	0.5009	64.6596	64.4024	0.0000	0.5009	64.6596	64.4024	0.0000	0.5009

6.3. Análisis de los resultados

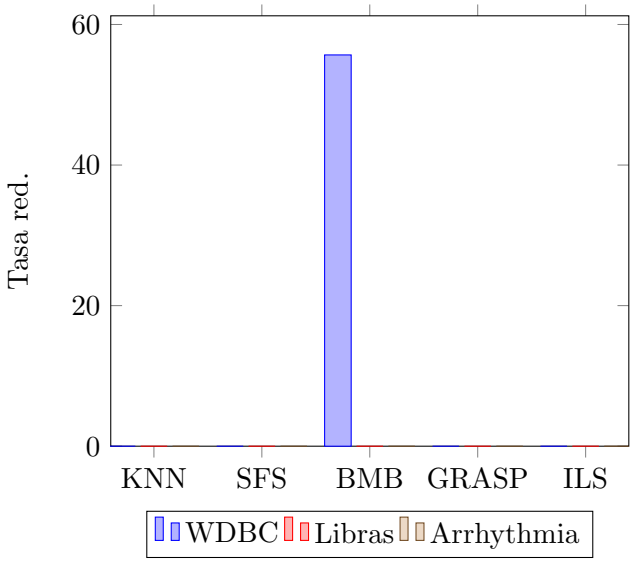
6.3.1. Tasa In



6.3.2. Tasa Out

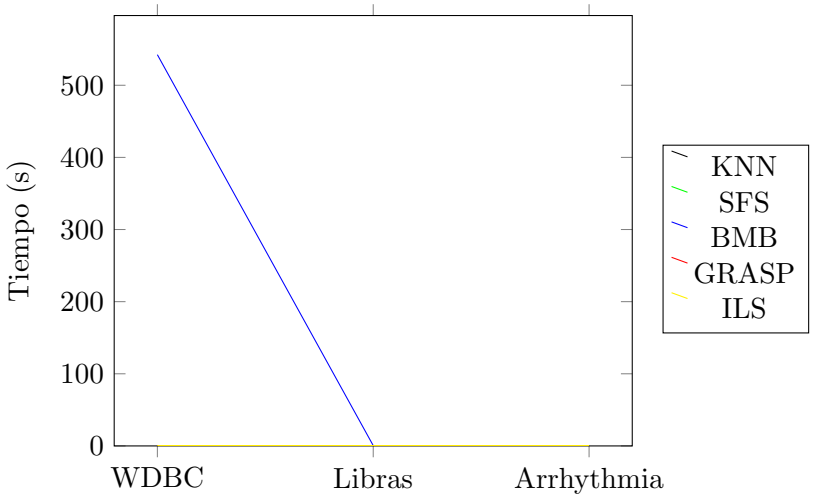


6.3.3. Tasa reducción



6.3.4. Tiempos

DB	0	1	2	3	4
TW	0.4001	0.5009	542.3986	0.4017	0.5009
TL	0.3885	0.5009	0.5009	0.5009	0.5009
TA	0.4226	0.5009	0.5009	0.5009	0.5009



7. Bibliografía

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