Práctica 1 Metaheurísticas.
Búsqueda por trayectorias para el problema de la selección de características

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

7 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. Greedy Sequential Forward Selection.
2. Búsqueda Local.
3. Enfriamiento Simulado.
4. Búsqueda Tabú Básica.
5. Búsqueda Tabú Extendida.

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 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	0,42	73,3333	70,5556	0	0,4966	64,5833	63,9175	0	0,4528
Particion 1-2	97,8947	94,7183	0	0,4188	63,3333	68,8889	0	0,4193	62,8866	65,625	0	0,5587
Particion 2-1	95,7746	96,1404	0	0,4157	66,1111	71,1111	0	0,4255	63,5417	65,4639	0	0,6544
Particion 2-2	96,8421	96,1268	0	0,4109	58,8889	76,6667	0	0,4233	64,9485	65,1042	0	0,5755
Particion 3-1	95,7746	95,7895	0	0,4307	68,3333	72,7778	0	0,421	65,1042	65,9794	0	0,462
Particion 3-2	96,1404	97,8873	0	0,4277	63,3333	71,6667	0	0,4283	67,0103	64,0625	0	0,4653
Particion 4-1	95,4225	96,8421	0	0,4198	63,8889	71,6667	0	0,4209	66,1458	60,8247	0	0,4652
Particion 4-2	94,7368	96,1268	0	0,4123	65	75,5556	0	0,4251	65,9794	64,5833	0	0,4604
Particion 5-1	97,1831	95,7895	0	0,4089	63,3333	65,5556	0	0,4249	60,4167	67,5258	0	0,4589
Particion 5-2	95,0877	97,1831	0	0,4166	67,2222	70,5556	0	0,4298	65,9794	60,9375	0	0,4559
Media	96,0279	96,2744	0	0,4181	65,2778	71,5	0	0,4315	64,6596	64,4024	0	0,5009

En primer lugar analizaremos las diferencias en las tasas de acierto en las bases de datos. La mayor tasa de acierto se obtiene en la base de datos `WDBC`. Es comprensible puesto que hay muchos más datos con respecto al número de características y clases que en las otras dos bases de datos. Puesto que en este método los datos de entrenamiento y test no tienen ninguna influencia en la solución, se observa que la diferencia entre la tasa de acierto dentro y fuera de la muestra es muy pequeña, y que incluso en la base de datos de `Libras` fuera de la muestra hay una mayor tasa de acierto. Esto es simplemente por las particiones realizadas. Observamos también que, aunque la diferencia en los tiempos sea mínima, esto es el tiempo en evaluar un conjunto de test, que es lo que iremos realizando repetidas veces en los demás métodos, con lo que se notará aún más.

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	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

Con este simple algoritmo *greedy* ya se aprecia un incremento en la tasa de acierto. Las tasas de reducción son muy elevadas, ya que hemos partido de la solución sin ninguna característica.

6.2.3. Local Search

	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	95,7746	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

En la búsqueda local, cuyos resultados debemos poner en contexto, se comprueba cómo con un algoritmo muy simple, y con unos tiempos de ejecución menores que el algoritmo *greedy* se llega a un buen resultado (que será un óptimo local).

6.2.4. Simulated Annealing

	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	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

En cambio, para el enfriamiento simulado los tiempos comienzan a crecer en mayor medida, en especial para el caso de `Arrhythmia`. Vemos como pese a haber realizado un mayor número de comprobaciones, los resultados son muy semejantes, e incluso un poco peores.

6.2.5. Tabu Search

	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	98,2394	95,4386	53,3333	2,020,4381	77,2222	70,5556	56,6667	1,686,2091	71,875	65,9794	52,518	1,344,9732
Particion 1-2	98,9474	94,7183	43,3333	2,060,2107	77,2222	68,8889	57,7778	1,698,4953	77,3196	65,1042	50,3597	1,281,406
Particion 2-1	97,8873	95,7895	60	2,021,0413	74,4444	71,1111	60	1,751,7003	70,8333	67,5258	56,1151	1,230,1321
Particion 2-2	99,6491	95,0704	40	2,257,6403	80,5556	70,5556	61,1111	1,720,3758	74,2268	67,1875	49,2806	1,240,9565
Particion 3-1	98,9437	94,386	46,6667	2,357,571	74,4444	70,5556	55,5556	1,944,607	75,5208	68,5567	51,4388	1,238,8125
Particion 3-2	97,8947	96,4789	40	2,127,7061	74,4444	67,2222	51,1111	1,994,9234	70,1031	65,1042	48,9209	1,245,0687
Particion 4-1	98,5915	94,386	53,3333	1,850,0274	74,4444	75	55,5556	1,823,5719	73,4375	58,2474	50,3597	1,226,1568
Particion 4-2	98,5965	95,7746	56,6667	1,865,9643	80,5556	76,1111	51,1111	1,588,2414	72,6804	65,625	52,8777	1,245,1324
Particion 5-1	99,6479	95,4386	46,6667	1,866,1031	76,6667	67,7778	53,3333	1,573,9598	71,3542	70,1031	51,7986	1,230,7279
Particion 5-2	97,5439	96,4789	50	1,867,056	76,6667	72,7778	54,4444	1,579,1599	75,2577	66,6667	53,9568	1,226,157
Media	98,5941	95,396	49	2,029,376	76,6666	71,0555	55,6666	1,736,1243	73,2608	66,0099	51,7625	1,250,9523

Los datos se han obtenido realizando 5000 evaluaciones. Aquí ya el tiempo, debido al número de comprobaciones realizadas crece mucho más. Sin embargo, sí que se aprecia una mejora con respecto a otros métodos.

6.2.6. Extended Tabu Search

	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	93,6842	43,3333	1,697,3548	76,6667	68,3333	45,5556	1,944,4471	71,875	69,0722	54,6763	1,836,2342
Particion 1-2	98,2456	94,3662	70	1,580,3348	71,1111	69,4444	48,8889	1,903,7602	73,7113	64,0625	52,1583	1,817,7184
Particion 2-1	98,2394	95,7895	53,3333	1,543,9258	73,8889	72,7778	60	1,837,5674	72,3958	65,4639	52,518	1,913,1097
Particion 2-2	97,8947	96,4789	50	1,554,7377	77,2222	72,2222	57,7778	1,841,9416	74,7423	66,6667	52,518	1,912,6772
Particion 3-1	97,8873	96,8421	40	1,546,7758	70,5556	64,4444	43,3333	2,140,9244	69,7917	68,5567	50,7194	1,861,1709
Particion 3-2	97,8947	95,4225	36,6667	1,561,2486	74,4444	72,2222	43,3333	2,191,2154	71,134	63,5417	52,518	2,115,6083
Particion 4-1	97,8873	96,1404	53,3333	1,556,0564	76,1111	70	60	2,164,7245	70,3125	63,9175	47,1223	2,042,8539
Particion 4-2	98,9474	95,7746	46,6667	1,569,4185	72,7778	72,7778	47,7778	2,069,99	72,6804	65,1042	52,1583	2,187,3337
Particion 5-1	98,9437	95,7895	50	1,512,9416	71,1111	70,5556	55,5556	1,931,1436	67,7083	67,0103	50	2,326,1933
Particion 5-2	97,5439	93,662	50	1,500,526	77,2222	67,7778	51,1111	1,825,534	70,1031	65,625	47,8417	2,112,4418
Media	98,278	95,395	49,333	1,562,332	74,1111	70,0556	51,3333	1,985,1248	71,4454	65,902	51,223	2,012,5341

Los datos se han obtenido realizando 5000 evaluaciones. Para la búsqueda tabú extendida vemos como el tiempo se incrementa un poco más que en la tabú y sin embargo el resultado no es mejor. Esto nos hace pensar que quizá debemos tomar otra estrategia a la hora de reinicializar las soluciones o llevar la memoria a largo plazo.

6.2.7. Comparación

	WDBC				Movement Libras				Arrhythmia			
	%Clas. in	%Clas. out	%red.	T	%Clas. in	%Clas. out	%red.	T	%Clas. in	%Clas. out	%red.	T
3-NN	96.0279	96.2744	0.0000	0.4181	65.2778	71.5000	0.0000	0.4315	64.6596	64.4024	0.0000	0.5009
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	98.5941	95.396	49.0	2029.376	76.6666	71.0555	55.6666	1736.1243	73.2608	66.0099	51.7625	1250.9523
BT extendida	98.278	95.395	49.333	1562.332	74.111111	70.055556	51.333333	1985.124829	71.4454	65.9020	51.2230	2012.5341

6.3. Análisis de los resultados

De la comparación de la tasa de acierto dentro de la muestra para cada algoritmo y base de datos se desprende que para el problema de clasificación, realizar una selección de características siempre es recomendable, pues se obtiene mayor tasa de acierto que usando sólo KNN. Esto no es así para `Libras` y búsqueda local y enfriamiento simulado. Esto se puede deber al elevado número de características y clases, en relación al número de datos. Se

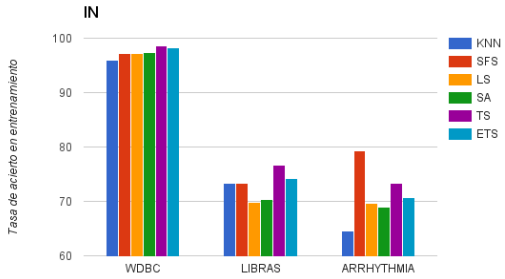

Figura 1: Tasa de acierto en el conjunto de entrenamiento

aprecia también que la búsqueda tabú mejora los resultados del enfriamiento simulado, con lo que quizá deberíamos pensar en otra exploración de vecinos. Por último, la alta tasa de acierto para SFS en `Arrhythmia` nos indica que para un número elevado de características y pocas clases, se puede dar que pocas características sean las que contienen la información, y que por tanto ir cogiendo las mejores sea una buena estrategia. Podríamos tener más certeza sobre esto si, por ejemplo, realizamos una búsqueda sobre el espacio de soluciones que sólo tengan un número fijo y pequeño de características (usaríamos otra representación del problema) y comparar los resultados (teniendo en cuenta que esto también sería heurística y que nos costaría tanto o más obtener la mejor selección).

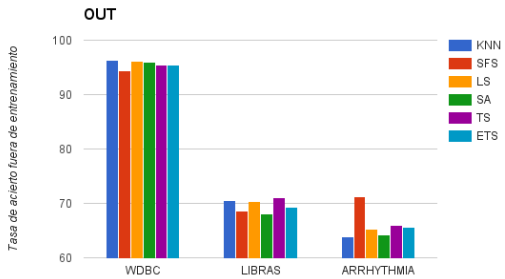

Figura 2: Tasa de acierto fuera del conjunto de entrenamiento

Para la tasa de acierto fuera de la muestra, vemos como el orden de los algoritmos es distinto y que ahora no tomar ninguna característica no es una opción a descartar. Esto se explica sencillamente porque los algoritmos de búsqueda han ajustado las características a los datos, y sin embargo, de manera general puede no resultar la mejor opción. Se aprecia ahora que en la base de datos `WDBC` las tasas son siempre más parejas y mejores (aunque

en este caso se requiera una mayor certeza que en el de `Libras`, por ejemplo) que en las otras. También podemos deducir que para el caso de `Libras` y `Arrhythmia` el resultado es más desigual en especial para la búsqueda tabú.

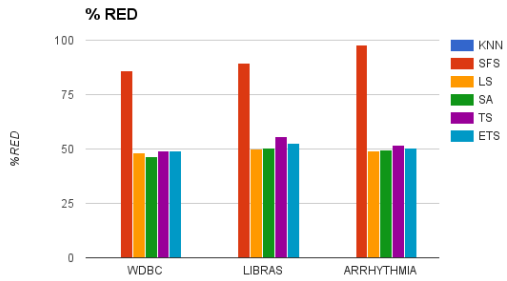

Figura 3: Porcentaje de reducción de las características

En cuanto al porcentaje de reducción, no tiene mucho sentido hacer un análisis más allá de ver cómo el SFS en `Arrhythmia` tiene un porcentaje muy elevado y es el mejor algoritmo. Esto va en la línea de lo anteriormente comentado. Como la función de evaluación no incluía nada sobre tener el menor número posible de características, los resultados en los demás algoritmos son poco más que aleatorios. Sin embargo, podemos relacionar el buen comportamiento en `Arrhythmia` de la búsqueda tabú y la ligera mayor reducción del número de características con que ese algoritmo va en la buena dirección.

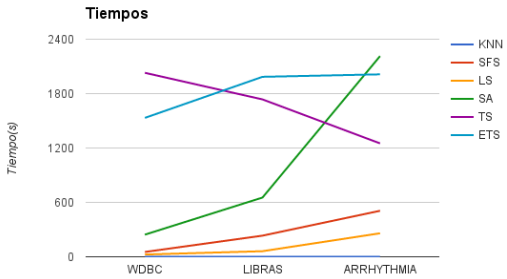

Figura 4: Tiempos empleados por el algoritmo

En cambio, para los tiempos, incluso teniendo en cuenta que la búsqueda tabú y la tabú extendida no han realizado todas las comprobaciones, sino sólo 5000, el tiempo es mucho mayor que para los demás métodos. Lo que sí es más estable con respecto al tamaño del problema. Mientras que la búsqueda local y el *greedy* SFS tienen un crecimiento más

limitado, el enfriamiento simulado aumenta mucho más. La búsqueda tabú tarda un mayor tiempo en `WDBC`, y esto se puede deber a que la lista tabú coincide en tamaño con el número de características, limitando más la variación de las soluciones, también limitado porque en esta base de datos se encuentra un muy buen resultado en poco tiempo (los datos son más semejantes y hay menos clases) y las mejoras son más difíciles de encontrar.

7. Bibliografía

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