Metaheurísticas

P2 - Búsquedas Multiarranque para el problema de la selección de características

Tercero Grado Ing. Informática

Francisco Carrillo Pérez
Grupo 1 jueves a las 17.30

Contents

Abstract

El problema de la selección de características consiste en minimizar el número de características necesarias para obtener un buen porcentaje de aciertos en la predicción de distintas clases en problemas de aprendizaje automático. En nuestro caso hemos utilizado como clasificador el 3-NN que se basa en la clase de los tres vecinos más cercanos.

Definición matemática

$$tasa\_clase(3-NN(s))=100*\frac{n^{O}deinstanciasbienclasificadasdeT}{n^{O}totaldeinstanciasdeT}$$

Donde:

- $s = \{0, 1\}, 1 \leq i \leq n$, es decir, un vector binario de tamaño n que define si una característica ha sido seleccionada o no.
- 3-NN es el clasificador que utilizaremos, que obtiene tres vecinos utilizando las características seleccionadas.
- Nos devuelve el porcentaje de aciertos que ha obtenido nuestro clasificador con esas características.

Intención de la selección de características

Con la selección de características nuestro objetivo es reducir el número de características necesarias para ser usadas en un problema de clasificación. Esto nos permite quitar características redundantes o que no aporten ningún tipo de valor, reduciendo así el tiempo de ejecución para la clasificación.

Aplicación de los algoritmos empleados para el problema

Descripción del esquema de representación de soluciones

Las soluciones se presentan como un vector binario de N posiciones, siendo este el número de atributos de la base de datos correspondiente. Si el valor de la posición está a 0, significa que esta característica no es considerada, mientras que si se encuentra a 1 esa característica se cuenta en consideración.

Generación de un vecino

El pseudocódigo de la generación de un vecino nuevo es la siguiente:

```
subproceso funcion generarVecinoRandom
para i = 0; i < num_atributos-1; i++
        num-aleatorio = generamos un número aleatorio
        si num-aleatorio >= 0.5
             cambiamos 0 por 1 o 1 por 0
```

Cambio de una posición del vector de atributos

El pseudocódigo de la función flip, usada para cambiar el valor de un elemento del vector de atributos, sería la siguiente:

```
subproceso funcion flip
si el valor de la posición es 0
    valor de la posición = 1
en caso contrario
    valor de la posición = 0
```

Función Objetivo

El pseudocódigo de la función objetivo es el siguiente:

```
subproceso funcion getExito
si en el conjunto seleccionado hay algún atributo a 1
    para i < tamaño  de los conjuntos
        obtenemos los tres vecinos más cercanos con la distancia euclídea
        se vota la clase a la que pertenece el elemento sin clasificar, siendo
        la que más se repite y si las tres son diferentes la primera de ellas
        guardamos la predicción
    para i < tamaño de los conjuntos
        comprobamos cuántas de nuestras predicciones son correctas
    calculamos el porcentaje de correctas
devolvemos el porcentaje de correctas, es decir, el éxito
```

Algoritmos implementados

Greedy SFS

El pseudocódigo de este algoritmo es el siguiente:

```
subproceso funcion greedy
mientras que mejoremos el exito o hayamos recorrido la lista de atributos
     para cada atributo, probamos uno a uno cuál produce mayor éxito
           si el max. éxito es menor que el éxito producido
              actualizamos el éxito al máximo que hemos encontrado
     Si el mejor éxito anterior es mejor que el mejor éxito conseguido y no es el mismo que hemos
           ponemos la característica a 1 y actualizamos el mejor éxito
     En caso contrario acabamos
     Si hemos recorrido todos los atributos acabamos
Devolvemos la mejor solución obtenida
```

BMB

El pseudocódigo de este algoritmo es el siguiente:

```
subproceso función BMB

generamos la solución inicial aleatoria y obtenemos su éxito
mientras que i < 25
     realizamos una búsqueda local primer mejor sobre la solución actual
     obtenemos el éxito de la solución nueva
     si el éxito nuevo es mejor que el mejor éxito hasta el momento
        actualizamos el mejor éxito y actualizamos la mejor solución
     generamos una nueva solución para la siguiente iteración
devolvemos la mejor solución
```

ILS

El pseudocódigo de este algoritmo es el siguiente:

```
subproceso función ILS

generamos una solución inicial aleatoria
realizamos una búsqueda local sobre esa solución
mientras que i < 25
    mientras que j < (0.1*(numero de atributos -1))
```

```
        realizamos una mutación de la mejor solución que hemos encontrado
    realizamos una búsqueda local primer mejor a la solución mutada
    obtenemos el éxito de la solución mutada
    si el éxito de la solución mutada es mejor que el éxito de la mejor
    solución encontrada hasta el momento
        actualizamos el éxito mejor
        actualizamos la solución mejor
    igualamos la solución mutada a la solución mejor que hayamos encontrado hasta
    el momento
devolvemos la mejor solución
```

GRASP

El pseudocódigo del algoritmo Greedy aleatorizado es el siguiente:

```
subproceso función greedyAleatorio

mientras que no acabemos o la lista de atributos no esté vacía
    mientras que i < numero de atributos
        si no tenemos cogida esa características
            ponemos la característica a 1 y obtemos el éxito
                si el éxito es mayor que el mayor obtenido
                  lo actualizamos como el mejor
                si el éxito es peor que el menor obtenido
                  lo actualizamos como el peor
                ponemos la característica a 0
        calculamos el umbral de aceptación
        metemos en un vector las características cuyo éxito superan el umbral
        cogemos una características de forma aleatoria
        obtenemos el éxito de la solución
        si el éxito es mejor que el mejor éxito hasta el momento
        y no se repite la característica
            actualizamos la mejor solución
        en caso contrario
            acabamos
        si hemos recorrido todos los atributos
        acabamos
devolvemos la mejor solución
```

El pseudocódigo del algoritmo GRASP es el siguiente:

```
subproceso función GRASP

mientras que i < 25
    generamos una solución greedy aleatoria
    realizamos una búsqueda local primer mejor sobre esa solución
    obtenemos el éxito de la nueva solución
    si el éxito nuevo es mejor que el mejor éxito encontrado hasta el momento
        actualizamos el éxito
        actualizamos la solución
devolvemos la mejor solución encontrada
```

Implementación

La implementación se ha realizado en C++ y la toma de tiempos se ha realizado con la librería *chrono* de C++.
En la carpeta de software se proporciona un Makefile que compila el main. En el archivo src/main.cpp

se puede comentar y descomentar dependiendo del algoritmo y la base de datos que queramos utilizar.
En la función **introducirDatos** se define que datos se van a coger con el último elemento:

- Si ponemos un 0, cogeremos la base de datos **arrythmia**.
- Si ponemos un 1, cogeremos la base de datos **movement-libras**.
- Por último, si ponemos un 2, cogeremos la base de datos **wdbc**.

Lo mismo ocurre con la función **dividirDatos**, el último elemento define de la forma en que dividimos los datos:

- Si ponemos un 0, dividimos los datos en dos conjuntos justo por la mitad.
- Si ponemos un 1, dividimos los datos introduciendo los atributos pares en un conjunto y los impares en otro.
- Si ponemos un 2, dividimos los conjuntos dividiendo los elementos intercaladamente de dos en dos.
- Si ponemos un 3, dividimos los conjuntos dividiendo los atributos intercaladamente de tres en tres.
- Por último, si ponemos un 4, dividimos los conjuntos dividiendo los atributos intercaladamente de cinco en cinco.

Resultados

La semilla utilizada para todos los experimentos ha sido **111**.
A continuación vamos a observar los resultados obtenidos:

	Wdbc			Movement_Libras			Arrythmia		
	%_Clas	%_red	T	%_Clas	%_red	T	%_Clas	%_red	T
Partición 1-1	88	0	0.33s	38	0	0.25s	67	0	0.54s
Partición 1-2	88	0	0.33s	38	0	0.25s	67	0	0.54s
Partición 2-1	93	0	0.34s	68	0	0.25s	63	0	0.56s
Partición 2-2	93	0	0.34s	68	0	0.25s	63	0	0.56s
Partición 3-1	92	0	0.33s	68	0	0.25s	65	0	0.57s
Partición 3-2	92	0	0.33s	68	0	0.25s	65	0	0.57s
Partición 4-1	92	0	0.37s	76	0	0.24s	66	0	0.56s
Partición 4-2	92	0	0.37s	76	0	0.24s	66	0	0.56s
Partición 5-1	94	0	0.34s	67	0	0.24s	65	0	0.56s
Partición 5-2	94	0	0.34s	67	0	0.24s	65	0	0.56s
MEDIA	91.8	0	0.346s	63.4	0	0.246s	65.2	0	0.558s

Table 1: Resultados obtenidos por el alg. 3-NN en el problema de la SC

	Wdbc			Movement_Libras			Arrythmia		
	%_Clas	%_red	T	%_Clas	%_red	T	%_Clas	%_red	T
Partición 1-1	93.33	90	33.8586s	42.77	94.44	95.4092s	73	98.2	436.088s
Partición 1-2	93.33	90	34.45s	42.77	94.44	97.067s	73	98.2	435.188s
Partición 2-1	92.98	90	26.67s	85.55	85.6	230.535s	76.16	98.2	654.997s
Partición 2-2	92.98	90	25.7201s	85.55	85.6	232.235s	76.16	98.2	656.002s
Partición 3-1	93.33	93.3	25.6504s	77.7	90	158.897s	76.68	98.2	614.678s
Partición 3-2	93.33	93.3	25.0789s	77.7	90	157.113s	76.68	98.2	612.929s
Partición 4-1	95.43	86.7	41.389s	66.11	94.44	95.5108s	75.12	98.2	654.238s
Partición 4-2	95.43	86,7	40.039s	66.11	94.44	97.567s	75.12	98.2	652.128s
Partición 5-1	96.14	90	33.3646s	76.11	92.22	126.993s	73.57	98.2	651.001s
Partición 5-2	96.14	90	35.0007s	76.11	92.22	123.676s	73.57	98.2	653.521s
MEDIA	94.242	90	32.15027s	69.648	91.34	141.5003s	74.906	98.2	602.083s

Table 2: Resultados obtenidos por el alg. Greedy en el problema de la SC

	Wdbc			Movement_Libras			Arrythmia		
	%_Clas	%_red	T	%_Clas	%_red	T	%_Clas	%_red	T
Partición 1-1	89.82	53.33	109.541s	38.88	51.11	72.3681s	71.50	46.04	119.885s
Partición 1-2	89.82	53.33	110.926s	38.88	51.11	70.683s	71.50	46.04	117.678s
Partición 2-1	96.14	43.33	96.56s	81.1	53.33	128.687s	67.8	46.76	84.688s
Partición 2-2	96.14	43.33	98.675s	81.1	53.33	129.456s	67.8	46.76	85.456s
Partición 3-1	94.03	56.66	81.672s	70	53.33	462.567s	67.87	52.87	97.167s
Partición 3-2	94.03	56.66	83.346s	70	53.33	461.862s	67.87	52.87	95.156s
Partición 4-1	95.08	56.66	102.56s	77.22	51.11	468.547s	68.91	53.23	146.568s
Partición 4-2	95.08	56.66	100.789s	77.22	51.11	466.914s	68.91	53.23	144.789s
Partición 5-1	95.78	56.66	101.762s	71.66	53.33	343.678s	67.35	52.15	86.567s
Partición 5-2	95.78	56.66	99.734s	71.66	53.33	342.694s	67.35	52.15	84.765s
MEDIA	94.17	53.33	98.38s	67.772	52.442	295.16s	68.672	50.214	106.271s

Table 3: Resultados obtenidos por el alg. BMB en el problema de la SC

	Wdbc			Movement_Libras			Arrythmia		
	%_Clas	%_red	T	%_Clas	%_red	T	%_Clas	%_red	T
Partición 1-1	90.17	63.33	87.089s	37.77	43.33	55.9213s	70.46	51.079	104.786s
Partición 1-2	90.17	63.33	89.564s	37.77	43.33	54.217s	70.46	51.079	105.160s
Partición 2-1	96.14	43.33	77.134s	81.11	50	130.178s	70.98	56.11	190.563s
Partición 2-2	96.14	43.33	76.56s	81.11	50	129.874s	70.98	56.11	189.65s
Partición 3-1	94.03	66.66	68.54s	69.44	58.88	427.789s	67.87	54.31	74.56s
Partición 3-2	94.03	66.66	69.098s	69.44	58.88	430.67s	67.87	54.31	79.78s
Partición 4-1	95.08	53.33	116.672s	78.88	45.55	470.618s	69.94	48.56	108.134s
Partición 4-2	95.08	53.33	115.45s	78.88	45.55	471.002s	69.94	48.56	107.567s
Partición 5-1	95.78	53.33	100.222s	71.66	53.33	378.027s	68.91	54.67	241.789s
Partición 5-2	95.78	53.33	99.602s	71.66	53.33	379.097s	68.91	54.67	240.512s
MEDIA	94.24	55.996	89.99s	67.77	50.21	292.507s	69.632	52.9458	144.25s

Table 4: Resultados obtenidos por el alg. ILS en el problema de la SC

A continuación, voy a mostar la tabla del algoritmo GRASP. Como se puede observar, la columna de Arrythmia está completamente vacía. Esto es debido a que los tiempos de ejecución eran desorbitados, ya que se dejó calculando durante 9 horas y no finalizaba, seguía, en algunas iteraciones, metiendo variables por el umbral, pero a un ritmo muy lento.

	Wdbc			Movement_Libras			Arrythmia		
	%_Clas	%_red	T	%_Clas	%_red	T	%_Clas	%_red	T
Partición 1-1	91.92	53.33	1690.23s	47.77	43.33	505.723s			
Partición 1-2	91.92	53.33	1692.73s	47.77	43.33	506.290s			
Partición 2-1	95.08	36.66	2020.26s	81.66	51.11	530.671s			
Partición 2-2	95.08	36.66	2021.256s	81.66	51.11	520.021s			
Partición 3-1	94.03	50	2181.08s	71.66	52.22	3798.03s			
Partición 3-2	94.03	50	2180.71s	71.66	52.22	3799.59s			
Partición 4-1	95.08	50	1139s	78.88	46.66	3710.21s			
Partición 4-2	95.08	50	1139.785s	78.88	46.66	3708.261s			
Partición 5-1	96.14	63.33	1253.9s	71.11	53.33	501.664s			
Partición 5-2	96.14	63.33	1256.42s	71.11	53.33	500.784s			
MEDIA	94.45	50.664	1656.89s	70.21	49.33	1806.73s			

Table 5: Resultados obtenidos por el alg. GRASP en el problema de la SC

	Wdbc			Movement Libras			Arrythmia		
	%_Clas	%_red	T	%_Clas	%_red	T	%_Clas	%_red	T
3-NN	91.8	0	0.346s	56	0	0.246s	65.2	0	0.558s
SFS	94.242	90	32.15027s	69.648	91.34	141.5003s	74.906	98.2	602.083s
BMB	94.17	53.33	98.38s	67.772	52.442	295.16s	68.672	50.214	106.271s
ILS	94.24	55.996	89.99s	67.77	50.21	292.507s	69.632	52.9458	144.25s
GRASP	94.45	50.664	1656.89s	70.21	49.33	1806.73s			

Table 6: Comparación media de todos los algoritmos

Vamos a enfocarnos en los resultados de media.
El algoritmo greedy 7 obtiene una alta reducción de las características en cualquiera de las bases de datos.Y sus resultados son bastante buenos. Su único problema es el tiempo que invierte cuando el número de características aumenta, el cuál a su vez aumenta el tiempo de ejecución de una forma muy considerable si lo comparamos con el resto de algoritmos. Esto se puede observar en las columnas de la base de datos Arrythmia 7.
El algoritmo BMB 7 obtiene siempre una tasa de reducción por encima del 50%, y unos resultados aceptables, sin ser los más destacables. Tiene un tiempo de ejecución alto en las bases de datos Movement y Wdbc pero reduce sustancialmente el tiempo del Greedy 7 en la base de datos Arrythmia.
El algoritmo ILS 7 obtiene también una tasa de reducción por encima del 50% superando al BMB 7 y al GRASP 7 en la base de datos Wdbc, y al BMB 7 en la base de datos Arrythmia. En cuanto a acierto,también supera en BMB 7 en las tres bases de datos.
Por último el algoritmo GRASP 7, es el que mejores resultados de clasificación obtiene en las bases de datos Wdbc y Movement.En contra, el tiempo de ejecución se dispara de una forma muy notable, lo que ha impedido ejecutarlo sobre la base de datos Arrythmia.

Algunas gráficas con los resultados obtenidos en la media serían:

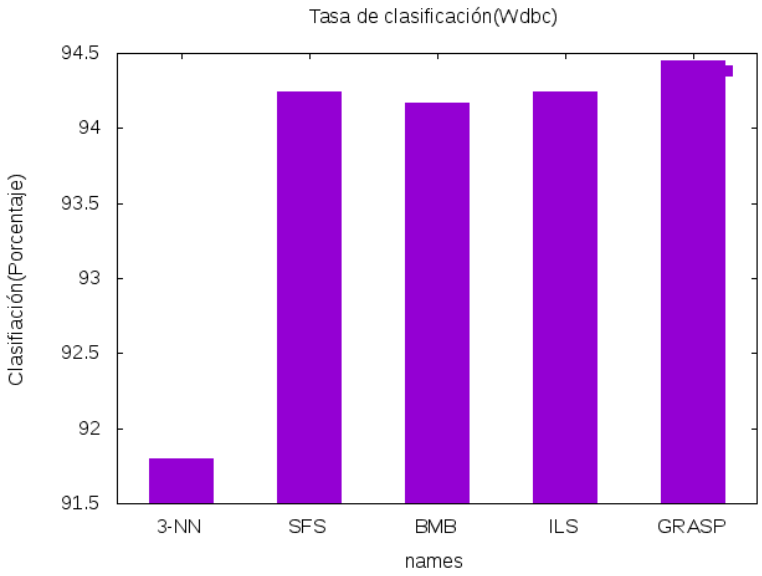

Figure 1: Diferencia de %_Clas en Wdbc

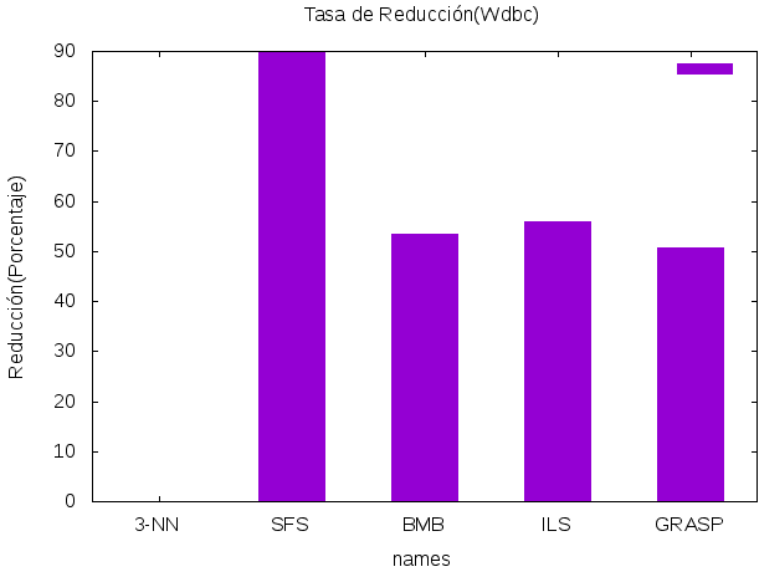

Figure 2: Diferencia de %_red en Wdbc

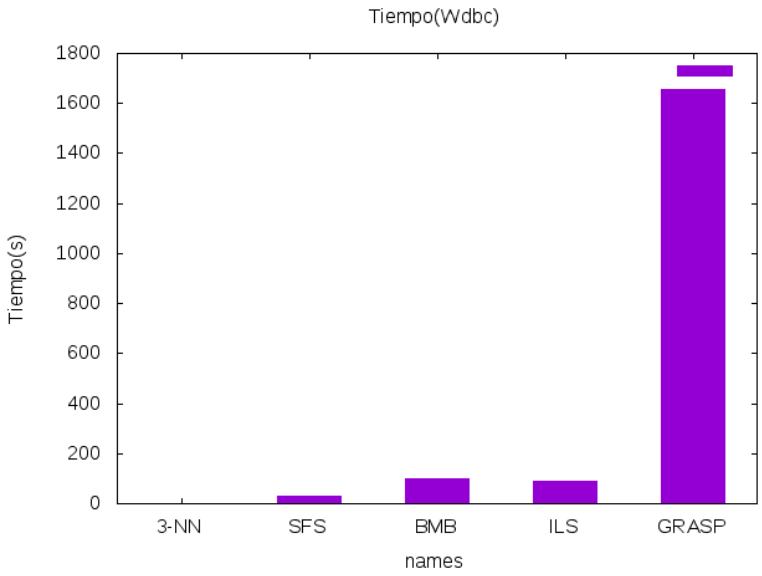

Figure 3: Diferencia de tiempo en Wdbc

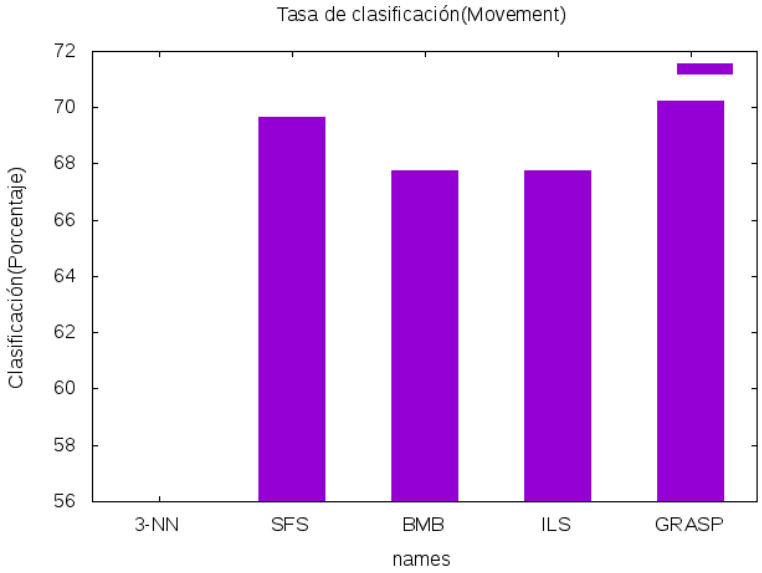

Figure 4: Diferencia de %_Clas en Movement Libras

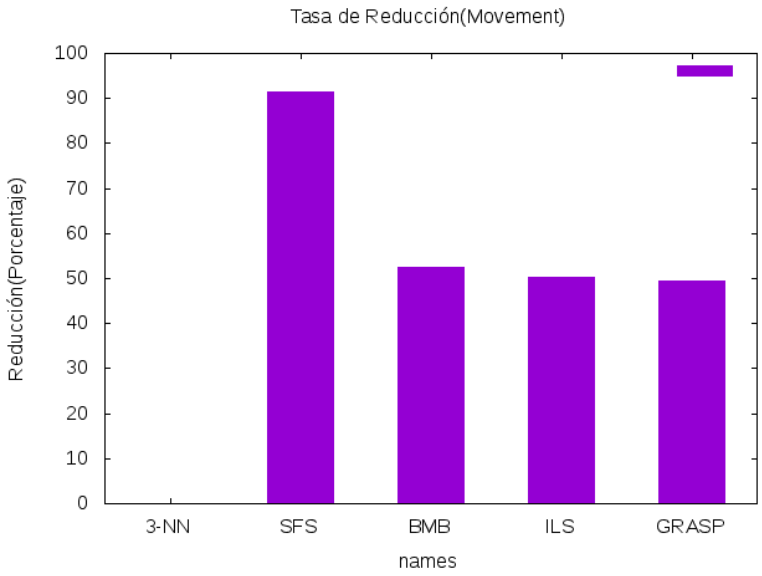

Figure 5: Diferencia de %_red en Movement Libras

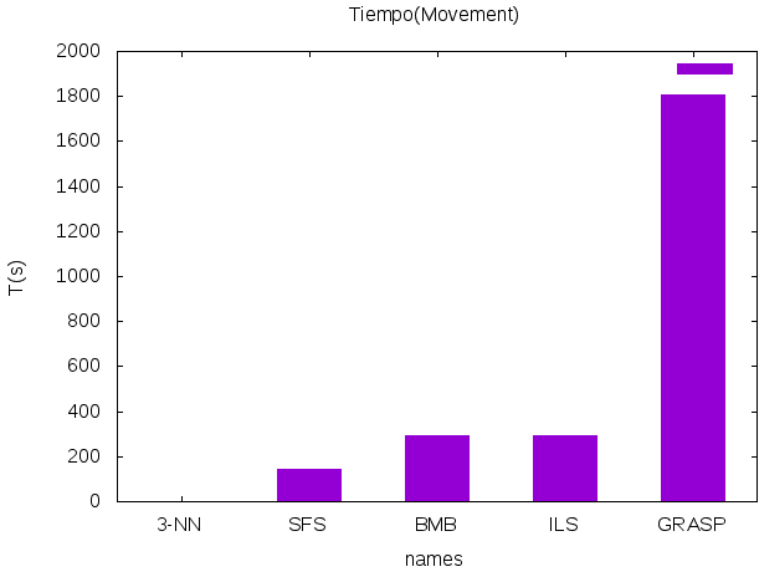

Figure 6: Diferencia de tiempo en Movement Libras

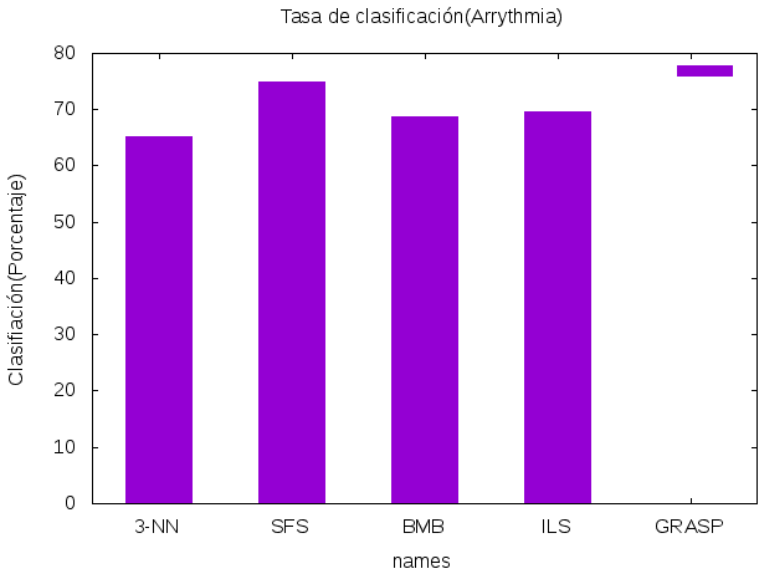

Figure 7: Diferencia de %_Clas en Arrythmia

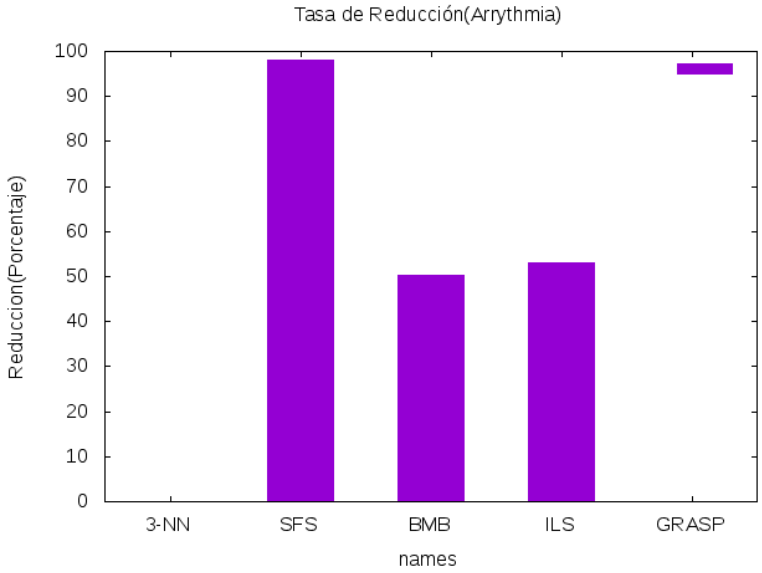

Figure 8: Diferencia de %_red en Arrythmia

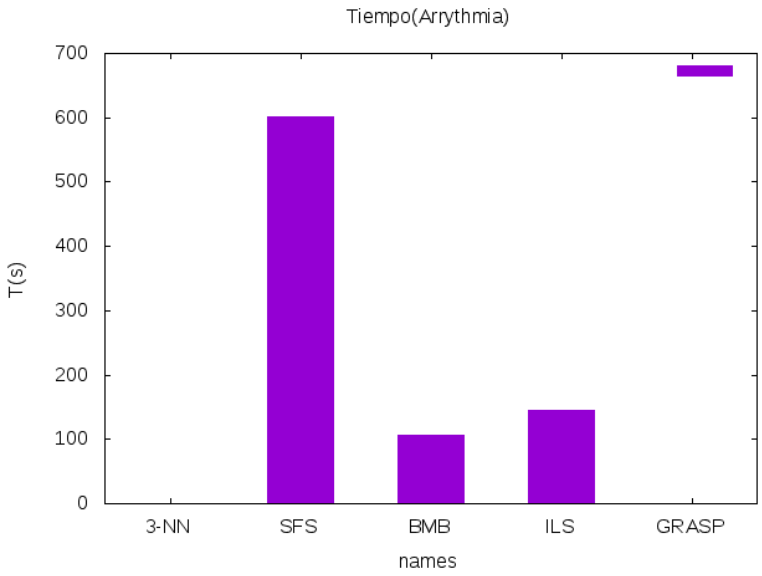

Figure 9: Diferencia de tiempo en Arrythmia

Conclusiones Finales

Como conclusión, en el caso de tener que utilizar un algoritmo dependería mucho. Si tengo una base de datos grande, con muchos ejemplos y muchas características, y quisiera tener una buena reducción de ellas en un tiempo aceptable, utilizaría uno de los algoritmos ILS o BMB, ya que en poco tiempo dan una reducción de más de 50% y unos resultados de clasificación aceptables, prefiriendo el ILS ya que da mejores resultados que el BMB. En cambio si tuviera una base de datos más pequeña utilizaría un GRASP aun que tarde más tiempo, ya que produce una clasificación mejor, con la misma tasa de reducción.