Metaheurísticas

P3 - Algoritmos Genéticos 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
```

Elementos de los algoritmos genéticos

Generar Población Aleatoria

El pseudocódigo es el siguiente:

```
subproceso funcion generarPoblacionRandom
    mientras que i < tamaño de la población
    mientras que j < numero de atributos
        generamos valor aleatorio para una posición
        cambiamos esa posición de valor
    añadimos individuo a la población
    limpiamos los valores del individuo
    devolvermos la población
```

Selección Torneo Binario

El pseudocódigo es el siguiente:

```
subproceso funcion seleccionTorneoBinario
    generamos un número aleatorio para un contricante
    generamos un número aleatorio para el ganador
    si la evaluación del contrincante es mayor que la evaluación del ganador
        el ganador se iguala al contrincante
    devolvemos el ganador
```

Cruce en Dos Puntos

El pseudocódigo es el siguiente:

```
subproceso funcion cruceDosPuntos
   generamos dos números aleatorios para los puntos de corte
   mientras que i < número de atributos
       si i > primer punto de corte y i < segundo punto de corte
           añadimos el cromosoma del padre en ese punto al primer hijo y el cromosoma
           de la madre a un segundo hijo
       en caso contrario
           añadimos el cromosoma de la madre en ese punto al primer hijo y el cromosoma
           del padre al segundo hijo
   devolvemos los dos hijos
```

Mutación

El pseudocódigo sería el siguiente:

```
subproceso funcion mutacion
   calculamos el número de cromosomas a mutar
   mientras que i < número de mutaciones
       calculamos un número aleatorio para elegir el individio de la población
       calculamos un número aleatorio para elegir el cromosoma de ese individuo
       cambiamos el valor del cromosoma de ese individuo
       indicamos que ese individuo no está evaluado
```

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
```

AGG

El pseudocódigo de este algoritmo es el siguiente:

```
subproceso funcion AGG
    generamos una población inicial aleatoria
    evaluamos esa población incial
    mientras que  i < tamaño de la población
        comprobamos el valor de cada individuo para encontrar el mejor
        que será el valor élite
    mientras que el número de evaluaciones sea menor a 15000
        mientras que i < tamaño de la población
            realizamos la seleccion por torneo binario y vamos guardando los
            resultados
        calculamos el número de cruces a realizar
        mientras que i < tamaño de la población
            si i <  número de cruces a realizar
                realizamos el cruce en dos puntos y los vamos guardando
                en una población de hijos
            en caso contrario
                no realizamos el cruce y los vamos guardando
                en una población de hijos
        calculamos el número de mutaciones a realizar
        mientras que i < numero de mutaciones
            calculamos dos números aleatorios, uno para el cromosoma
            y otro para el individuo, y cambiamos el valor de ese cromosoma
        realizamos el reemplazamiento con la evaluación de la población
        realizamos el elitismo sustituyendo el mejor de la población anterior por el peor de la po
        mientras que  i < tamaño de la población
```

```
        comprobamos el valor de cada individuo para encontrar el mejor
        que será el valor élite
        cambiamos la población actual por la población de los hijos
    devolvemos el mejor individuo de la población
```

AGE

El pseudocódigo de este algoritmo es el siguiente:

```
subproceso funcion AGE
    generamos una población inicial aleatoria
    evaluamos esa población incial
    mientras que  i < tamaño de la población
        comprobamos el valor de cada individuo para encontrar el mejor
        que será el valor élite
    mientras que el número de evaluaciones sea menor a 15000
        mientras que i < tamaño de la población
            realizamos la seleccion por torneo binario y vamos guardando los
            resultados
        calculamos el número de cruces a realizar
        mientras que i < tamaño de la población
            si i <  número de cruces a realizar
                realizamos el cruce en dos puntos y los vamos guardando
                en una población de hijos
            en caso contrario
                no realizamos el cruce y los vamos guardando
                en una población de hijos
        calculamos el número de mutaciones a realizar
        mientras que i < numero de mutaciones
            calculamos dos números aleatorios, uno para el cromosoma
            y otro para el individuo, y cambiamos el valor de ese cromosoma
        realizamos el reemplazamiento con la evaluación de la población
        mientras que i < tamaño de la población
            comprobamos cuales son los dos peores individuos de la
            población
        los sustituimos en la población por los dos mejores de la anterior población
    devolvemos el mejor individuo de la población
```

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	90.52	46.66	4660.58s	44.44	70	3009.48s	75.12	53.95	7499.78s
Partición 1-2	90.52	46.66	4661.078s	44.44	70	3010.02s	75.12	53.95	7500.67s
Partición 2-1	96.49	50	4696.94s	62.77	50	3153.96s	75.12	46.76	7550.71s
Partición 2-2	96.49	50	4695.41s	62.77	50	3152.54s	75.12	46.76	7548.78s
Partición 3-1	94.03	60	4678.64s	56.11	46.66	3114.37s	72.53	47.84	7478.0092s
Partición 3-2	94.03	60	4679.212s	56.11	46.66	3115.23s	72.53	47.84	7479.613s
Partición 4-1	95.78	53.33	4636s	60	51.11	3083.44s	74.09	50.33	7527.24s
Partición 4-2	95.78	53.33	4635.61s	60	51.11	3084.23s	74.09	50.33	7526.02s
Partición 5-1	96.49	50	4635.56s	53.88	42.22	3144.64s	73.57	49.64	7535.63s
Partición 5-2	96.49	50	4633s	53.88	42.22	3146.19s	73.57	49.64	7538.91s
MEDIA	94.692	51.998	4661.544s	55.44	51.998	3101.21s	74.086	49.704	7518.44s

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

	Wdbc			Movement_Libras			Arrythmia		
	%_Clas	%_red	T	%_Clas	%_red	T	%_Clas	%_red	T
Partición 1-1	89.82	50	4615.14s	42.22	61.11	3058.84s	73.57	47.84	7564.07s
Partición 1-2	89.82	50	4617.078s	42.22	61.11	3056.82s	73.57	47.84	7567.67s
Partición 2-1	96.49	60	4650.21s	63.33	47.77	3121.24s	73.57	43.16	7552.71s
Partición 2-2	96.49	60	4649.891s	63.33	47.77	3120.084s	73.57	43.16	7553.43s
Partición 3-1	94.03	66.66	4645.79s	57.77	38.88	3153.66s	72.53	41.72	7399.86s
Partición 3-2	94.03	66.66	4644.212s	57.77	38.88	3155.83s	72.53	41.72	7397.313s
Partición 4-1	95.08	60	4639.46s	58.88	38.88	3096.14s	73.05	47.84	7528.944s
Partición 4-2	95.08	60	4638s	58.88	38.88	3098.33s	73.05	47.84	7529.83s
Partición 5-1	95.78	70	4607.99s	50.55	42.22	3135.45s	70.98	45.68	7369.9s
Partición 5-2	95.78	70	4698.167s	50.55	42.22	3134.985s	70.98	45.68	7370.12s
MEDIA	94.24	61.332	4631.56s	54.55	45.77	3113.066s	72.7	45.248	7483.096s

Table 4: Resultados obtenidos por el alg. AGE 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
AGG	94.692	51.998	4661.544s	55.44	51.998	3101.21s	74.086	49.704	7518.44s
AGE	94.24	61.332	4631.56s	54.55	45.77	3113.066s	72.7	45.248	7483.096s

Table 5: Comparación media de todos los algoritmos

Vamos a analizar los resultados:
En primer caso tenemos los resultados del Greedy 8 que han sido analizados en las anteriores prácticas.
A continuación, tenemos los resultados del algoritmo AGG 8. En la base de datos arrythmia obtenemos buenos resultado, reduciendo hasta la mitad las características y obteniendo buenos resultados de clasificación. Obtenemos casi la misma tasa de clasificación que el algoritmo Greedy 8 pero con más características. Esto podría indicar, que aún teniendo más características, obtenemos un porcentaje cercano de clasificación, lo que podría producir que al llegar nuevos elementos a clasifica,r estas características de más podrían beneficiarnos a la hora de su clasificación.En la base de datos Movement la clasificación baja sustancialmente, y la reducción de características es de media un poco superior al 50%. En cambio, en la base de datos Wdbc mejora los resultados del algoritmo Greedy8 en clasficación, y reduce también un poco más del 50%.

Por último, tenemos el algoritmo AGE 8. En este caso obtiene resultados muy parecidos al AGG 8, peores en la base de datos Arrythmia y Movement, pero mejora sustancialmente la tasa de reducción en la base de datos Wdbc, llegando a superer el 60%, obteniendo la misma tasa de clasificación que el algoritmo Greedy 8. Esto, siguiendo el mismo razonamiento que se ha seguido anteriormente con el AGG, podría llegar a ser bueno para la clasificación de nuevas instancias. Algunas gráficas de comparación de los resultados serían las siguientes:

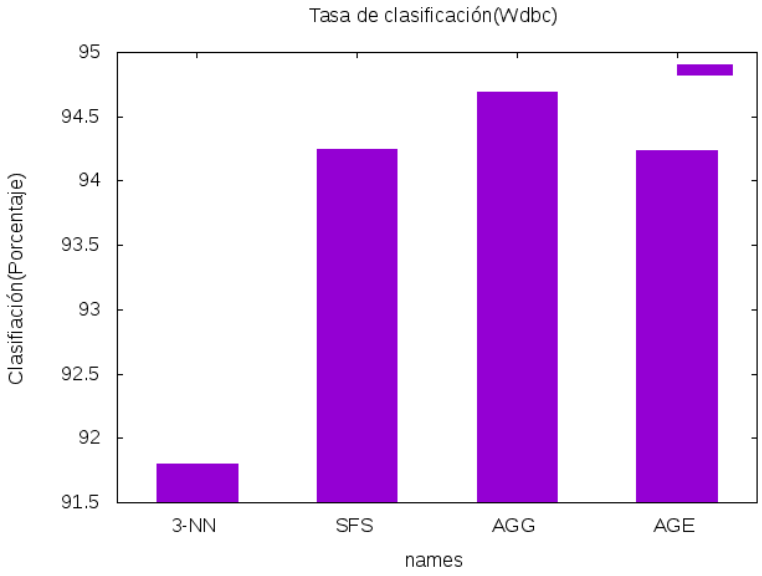

Figure 1: Diferencia de %_Clas en Wdbc

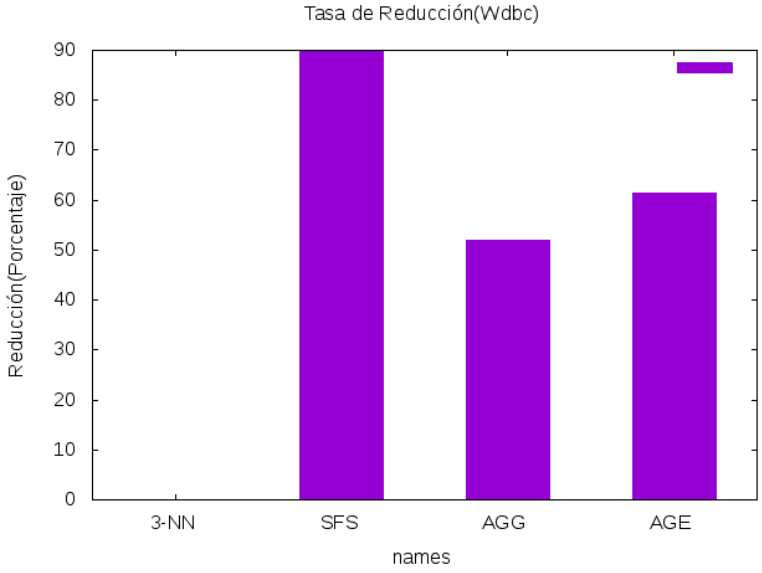

Figure 2: Diferencia de %_red en Wdbc

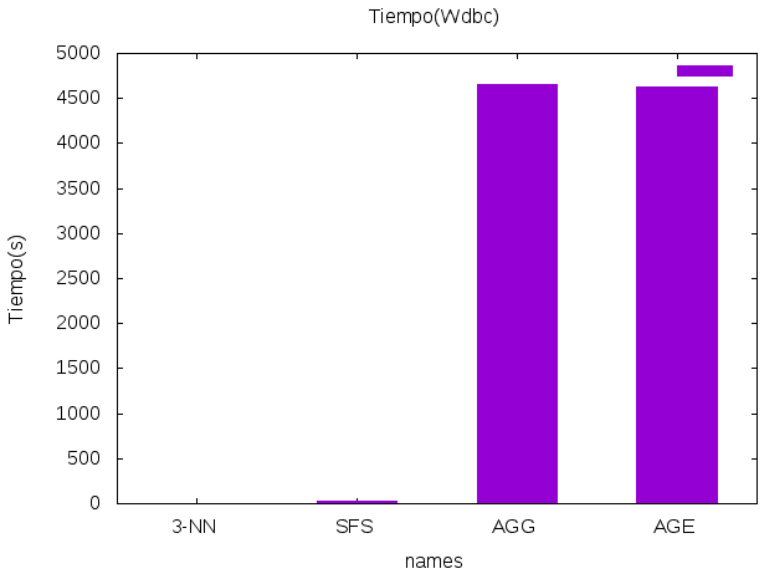

Figure 3: Diferencia de tiempo en Wdbc

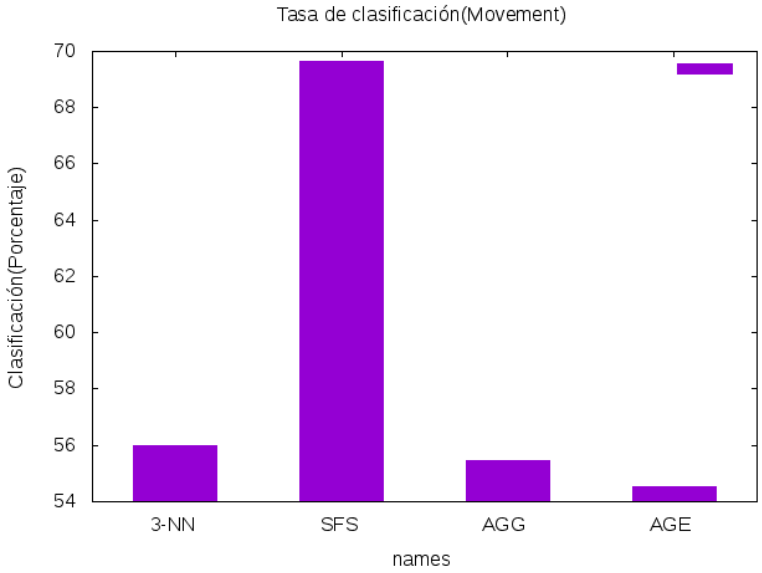

Figure 4: Diferencia de %_Clas en Movement Libras

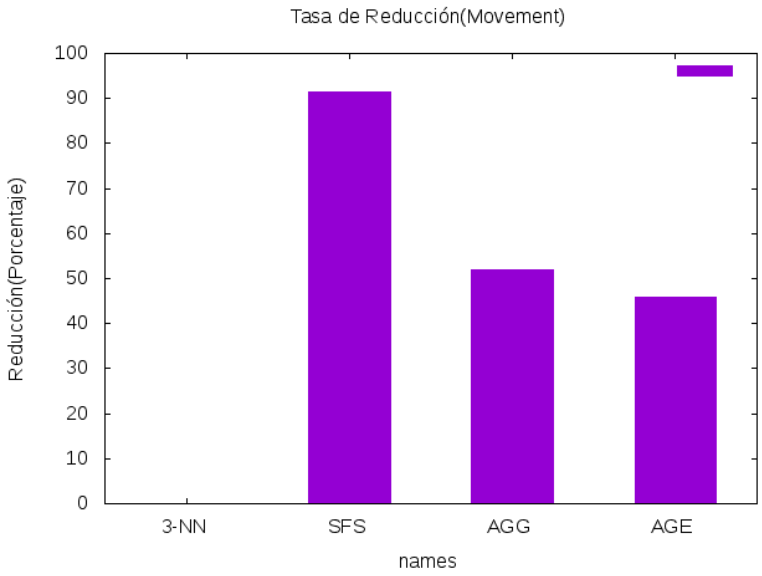

Figure 5: Diferencia de %_red en Movement Libras

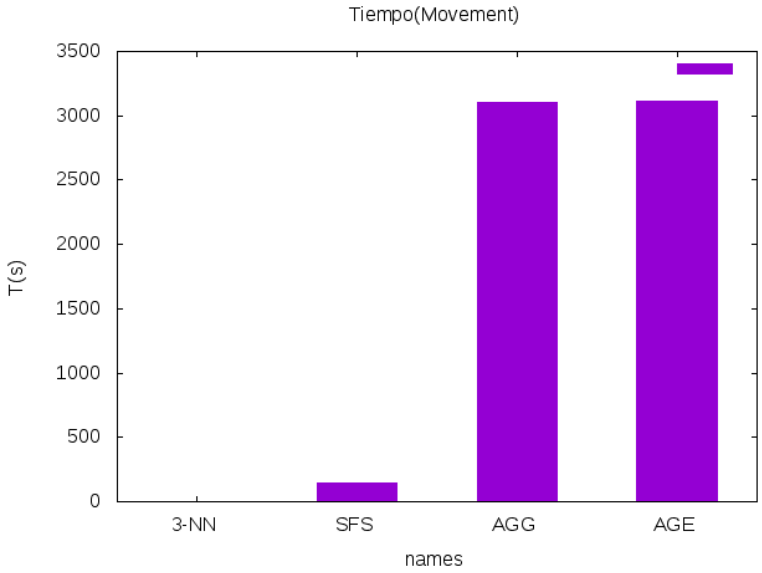

Figure 6: Diferencia de tiempo en Movement Libras

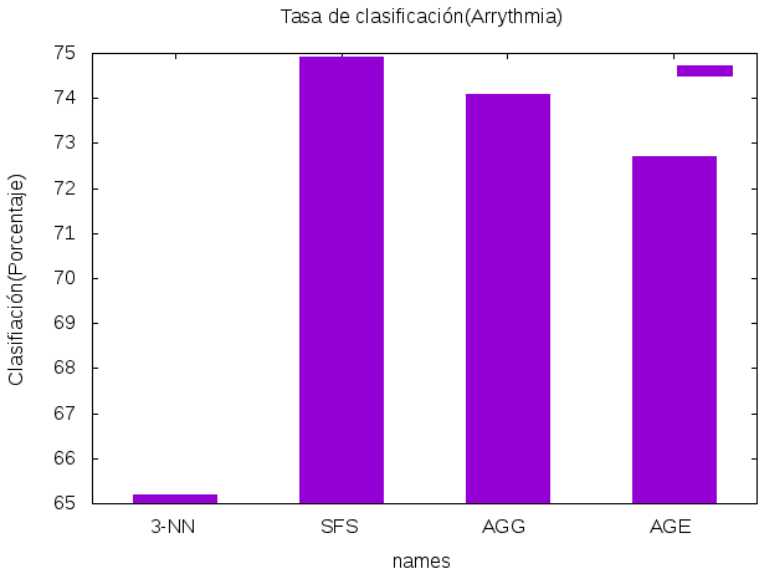

Figure 7: Diferencia de %_Clas en Arrythmia

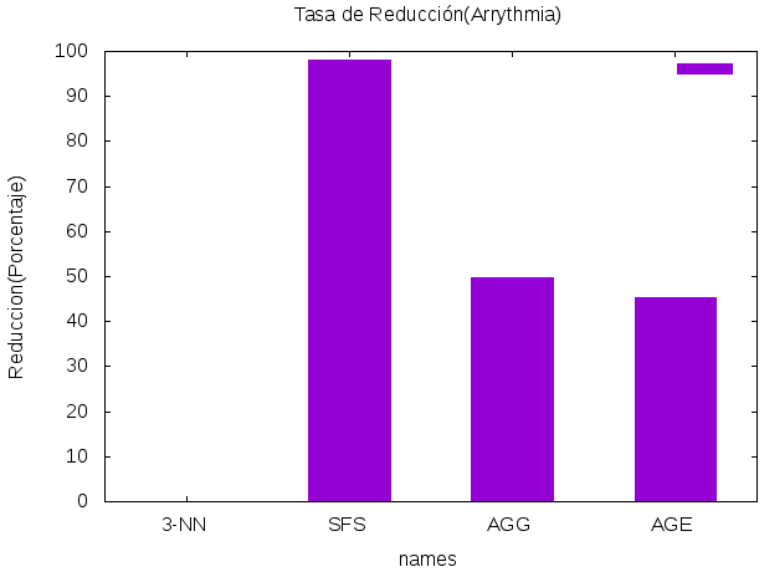

Figure 8: Diferencia de %_red en Arrythmia

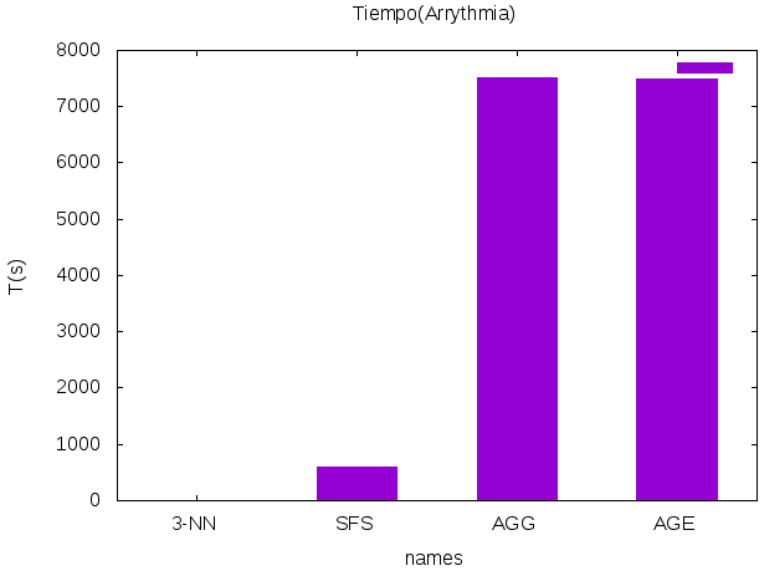

Figure 9: Diferencia de tiempo en Arrythmia

Conclusiones Finales

Como podemos observar, el algoritmo Greedy supera en las gráficas excepto en la de tasa de clasificación de Wdbc. Pero si observamos los resultados individualmente. los algoritmos al conseguir tasas de clasificación parecidas, excepto en la base de datos Movement, con un número mayor de características, podría beneficiarnos a la hora de clasificar nuevos elementos que lleguen.