Metaheurísticas

P5 - Búsquedas Híbridas 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
     elegido en la anterior iteración
           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
```

AGGAM-(10,1.0)

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 población actual
      si el número de evaluaciones es múltiplo de 10 y el número de
      evaluaciones es distinto de 0
      mientras que i < tamaño de la problación y no salir
           si el número de evaluaciones es mayor que 15000
               salir = true
           en caso contrario
               realizamos una búsqueda local primer mejor
               sobre el cromosoma i de la población
      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
```

AGGAM-(10,0.1)

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 población actual
      si el número de evaluaciones es múltiplo de 10 y el número de
      evaluaciones es distinto de 0
          saco un número aleatorio
          realizamos una búsqueda local primer mejor
          sobre el cromosoma que corresponda al número
          aleatorio de la población
      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
```

AGGAM-(10,0.1mej)

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 población actual
      si el número de evaluaciones es múltiplo de 10 y el número de
      evaluaciones es distinto de 0
         realizamos una búsqueda local primer mejor
         sobre el mejor cromosoma de la población
      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
```

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.

Es necesario que los ficheros de las bases de datos estén en la carpeta /bin.

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	36.66	992.75s	38.88	41.1	1129.8s	66.32	47.84	2248.86s
Partición 1-2	89.82	36.66	998.78s	38.88	41.1	1139.2s	66.32	47.84	2243.04s
Partición 2-1	93.68	33.33	956.94s	81.66	50	1080.99s	66.32	51.79	2367.08s
Partición 2-2	93.68	33.33	960.41s	81.66	50	1086.54s	66.32	51.79	2369.08s
Partición 3-1	93.33	40	978.64s	71.66	48.88	1158.66s	66.32	51.79	2234.23s
Partición 3-2	93.33	40	979.212s	71.66	48.88	1148.663s	66.32	51.79	2232.02s
Partición 4-1	94.03	70	965.89s	78.88	51.11	1251.34s	67.87	50.35	2317.6s
Partición 4-2	94.03	70	964s	78.88	51.11	1253.23s	67.87	50.35	2316.14s
Partición 5-1	95.43	70	976.06s	71.11	47.77	1493.32s	73.57	53.19	2619.5s
Partición 5-2	95.43	70	977.56s	71.11	47.77	1481.12s	73.57	53.59	2617.63s
MEDIA	93.158	49.998	974.056s	68.438	47.772	1222.822s	68.08	51.062	2357.08s

Table 3: Resultados obtenidos por el alg. AGGAM-(10,1.0) en el problema de la SC

	Wdbc			Movement_Libras			Arrythmia		
	%_Clas	%_red	T	%_Clas	%_red	T	%_Clas	%_red	T
Partición 1-1	91.92	46.66	1961.16s	42.22	55.55	1248.86s	73.05	50.71	2704.13s
Partición 1-2	91.92	46.66	1954.20s	42.22	55.55	1258.16s	73.05	50.71	2714.21s
Partición 2-1	96.49	40	1813.33s	85	52.22	1212.13s	73.57	47.12	2669.42s
Partición 2-2	96.49	40	1803.12s	85	52.22	1224.34s	73.57	47.12s	2665.98s
Partición 3-1	94.38	53.33	1895.94s	75.55	53.33	1227.35s	73.57	56.11	2655.16s
Partición 3-2	94.38	53.33	1885.94s	75.55	53.33	1201.89s	73.57	56.11	2635.90s
Partición 4-1	95.43	56.66	1916.05s	81.11	65.55	1257.27s	72.02	48.20	2710.7s
Partición 4-2	95.43	56.66	1936.9s	81.11	65.55	1250s	72.02	48.20	2700.29s
Partición 5-1	96.49	40	1855.17s	75.55	51.11	1166.91s	70.46	54.31	2757.19s
Partición 5-2	96.49	40	1831.2s	75.55	51.11	1176.81s	70.46	54.31	2759.73s
MEDIA	94.942	47.33	1888.066s	71.886	55.552	1472.5s	72.534	51.29	2699.32s

Table 4: Resultados obtenidos por el alg. AGGAM-(10,0.1) en el problema de la SC

	Wdbc			Movement_Libras			Arrythmia		
	%_Clas	%_red	T	%_Clas	%_red	T	%_Clas	%_red	T
Partición 1-1	91.92	40	3153.44s	35.55	54.44	1719.04s	69.43	50.71	3529.29s
Partición 1-2	91.92	40	3133.98s	35.55	54.44	1720.61s	69.43	50.71	3512.30s
Partición 2-1	96.49	33.33	2740.1s	78.33	56.66	1507.03s	67.87	52.15	3420.78s
Partición 2-2	96.49	33.33	2741.20s	78.33	56.66	1527.35s	67.87	52.15	3424.08s
Partición 3-1	93.68	36.66	2694.94s	68.88	54.44	1693.6s	67.87	42.44	3173.63s
Partición 3-2	93.68	36.66	2664.11s	68.88	54.44	1671.012s	67.87	42.44	3159.19s
Partición 4-1	94.73	43.33	2915.7s	73.33	54.44	1724.12s	70.46	47.84	3396.57s
Partición 4-2	94.73	43.33	2911.9s	73.33	54.44	1715.7s	70.46	47.84	3371.01s
Partición 5-1	95.78	40	2894.49s	70.55	47.77	1469.11s	67.87	55.39	4215.83s
Partición 5-2	95.78	40	2890.92s	70.55	47.77	1480.73s	67.87	55.39	4215.83s
MEDIA	94.52	38.664	2875.842s	65.328	53.55	1620.58s	67.5	49.706	3547.22s

Table 5: Resultados obtenidos por el alg. AGGAM-(10,0.1mej) 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
AGGAM-(10,1.0)	93.158	49.998	974.056s	68.438	47.772	1222.822s	68.08	51.062	2357.08s
AGGAM-(10,0.1)	94.942	47.33	1888.066s	71.886	55.552	1472.5s	72.534	51.29	2699.32s
AGGAM-(10,0.1mej)	94.52	38.664	2875.842s	65.328	53.55	1620.58s	67.5	49.706	3547.22s

Table 6: Comparación media de todos los algoritmos

Lo primero que he de decir es que **se ha reducido el número de evaluaciones a 5000** debido al exagerado tiempo que se dedicaba al cálculo, y a que mi pobre portátil empezaba a morir lentamente. Disculpe las molestias pero me adaptaré en cuanto a la revisión de los resultados según está característica.
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 de el algoritmo memético al cuál le aplicamos la búsqueda local a todos los cromosomas 8. Podemos observar cómo los resultados no son muy destacables, pero si los comparamos con los obtenidos en la Práctica 3 mejoran bastante. Esto es debido a la gran diversidad que aportan los algorítmos genéticos y a la intensificación de los algoritmos de búsqueda local, con lo que se obtiene un buen equilibrio diversidad e intesificación.
Con respecto a la segunda variante, en la cuál se aplicaba la búsqueda local solo a uno de los cromosomas escogido aleatoriamente 8 podemos observar como mejora alos resultados del anterior 8 considerablemente, excepto en la tasa de reducción en la base de datos Wdbc, y en los tiempos de ejecución. Esto es debido en que al realizar solo una búsqueda local, se consumen muchas menos evaluaciones en la búsqueda local que es más rápida.
Por último, en la última variante en la cual se aplicaba la búsqueda local al mejor cromosoma de la población 8, el cuál obtiene peores tiempos que los otros. Esto puede ocurrir debido a que al intensificar al mejor de la población, estamos intensificando demasiado y haciéndo que la mayoría de cromosomas convergan hasta los mismos óptimos o parecidos.
Vamos a mostrar distintas gráficas ilustrando los resultados:

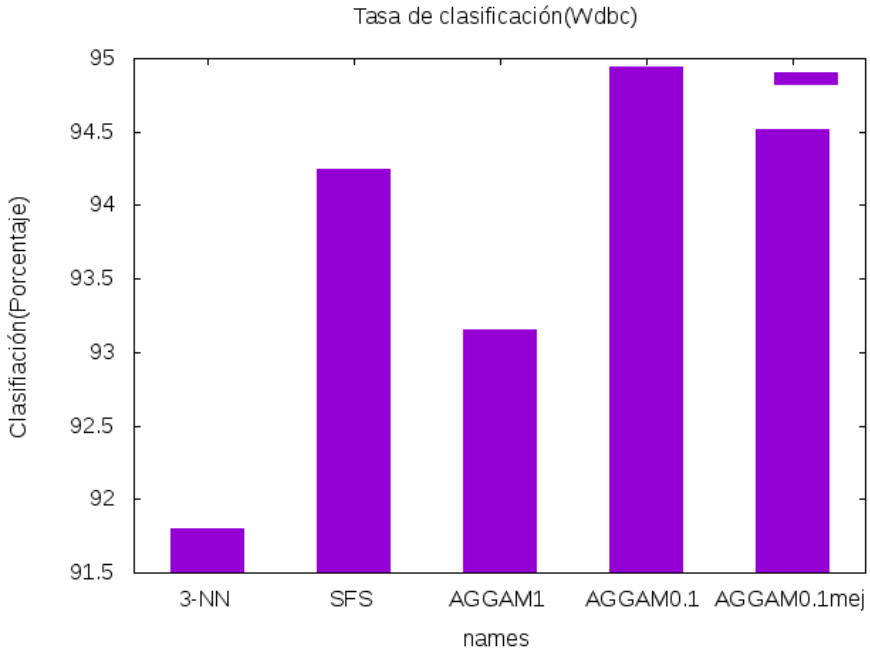

Figure 1: Tasa de clasificación en la base de datos Wdbc

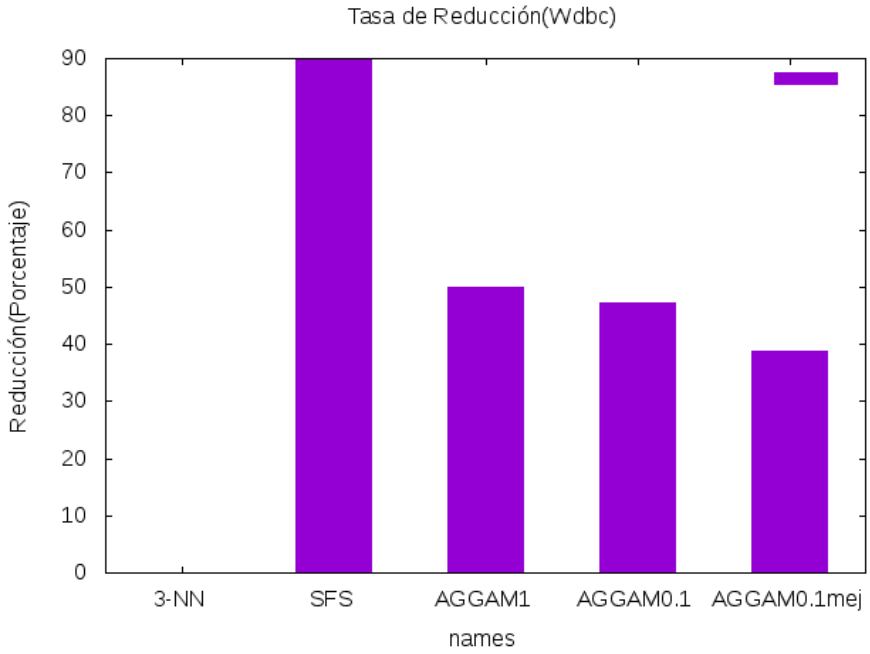

Figure 2: Tasa de reducción en la base de datos Wdbc

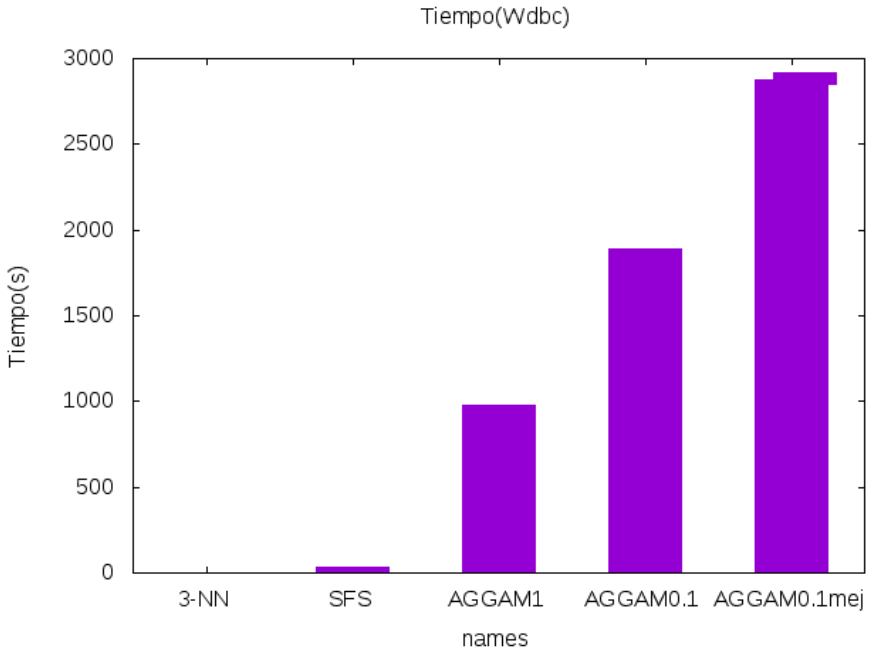

Figure 3: Tiempo de ejecución en la base de datos Wdbc

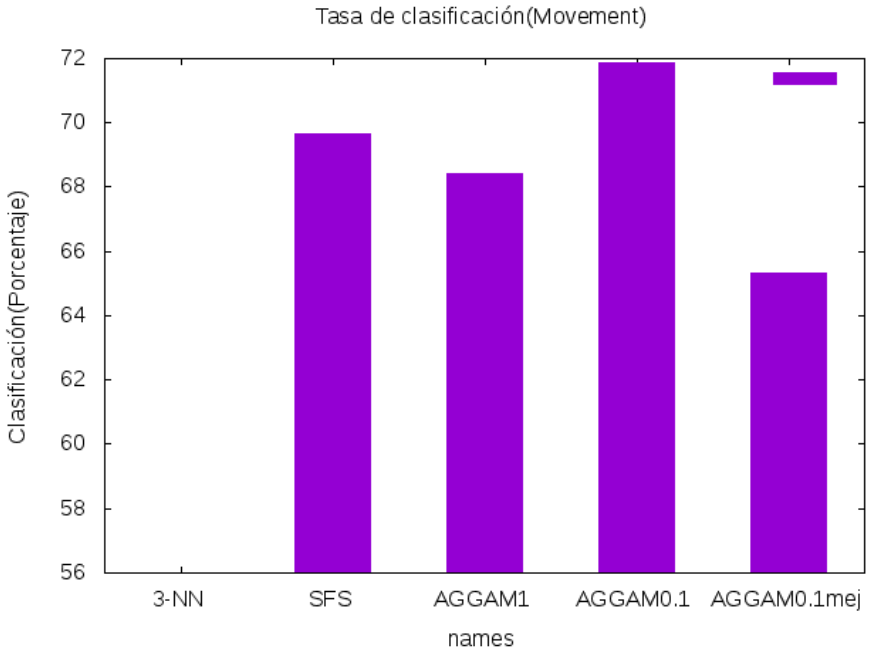

Figure 4: Tasa de clasificación en la base de datos Movement

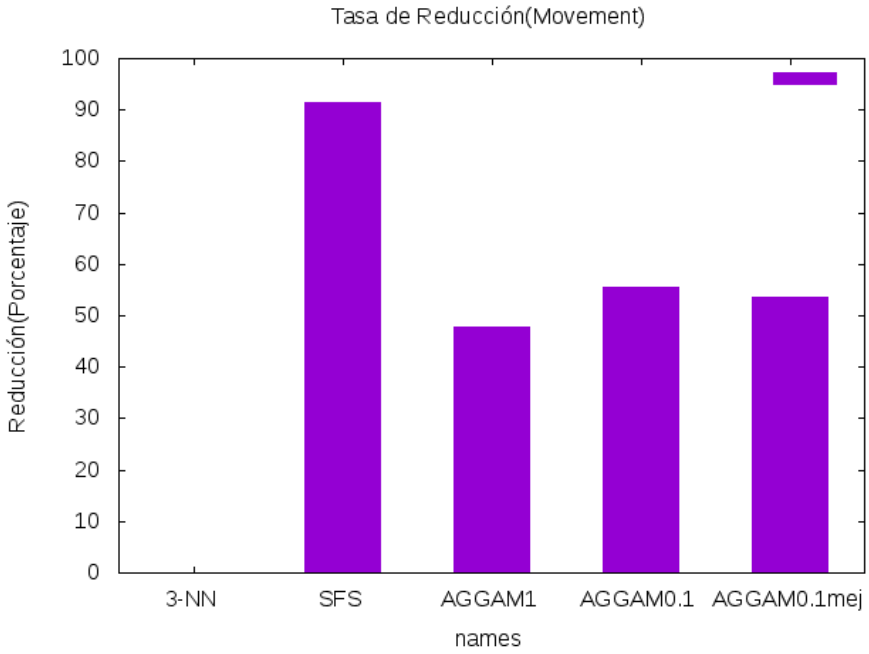

Figure 5: Tasa de reducción en la base de datos Movement

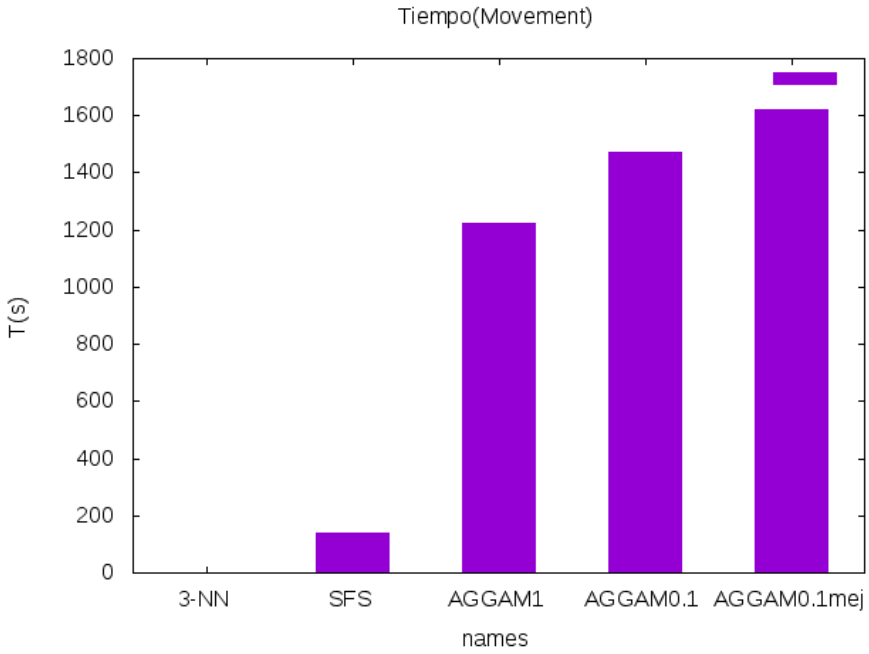

Figure 6: Tiempo de ejecución en la base de datos Movement

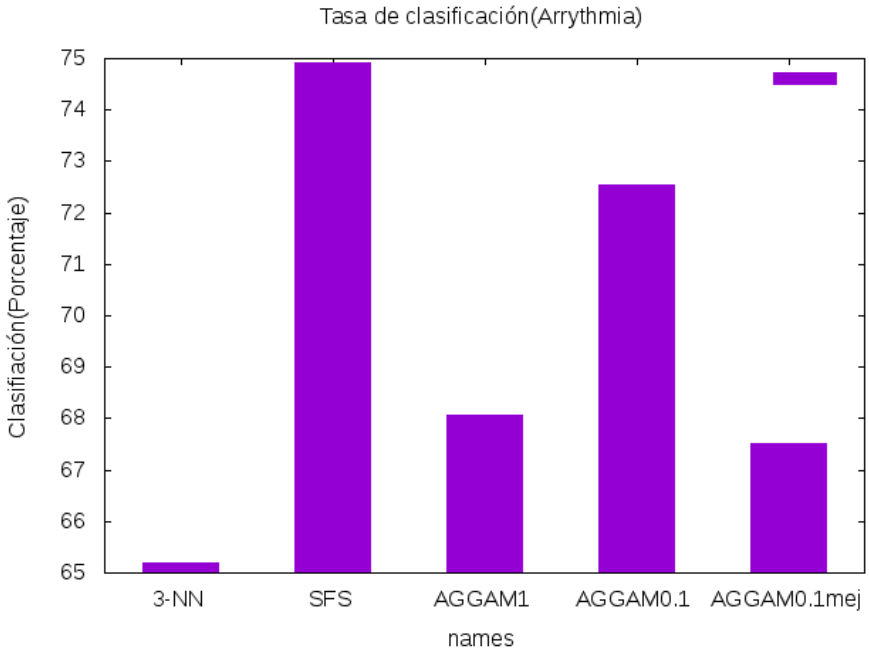

Figure 7: Tasa de clasificación en la base de datos Arrythmia

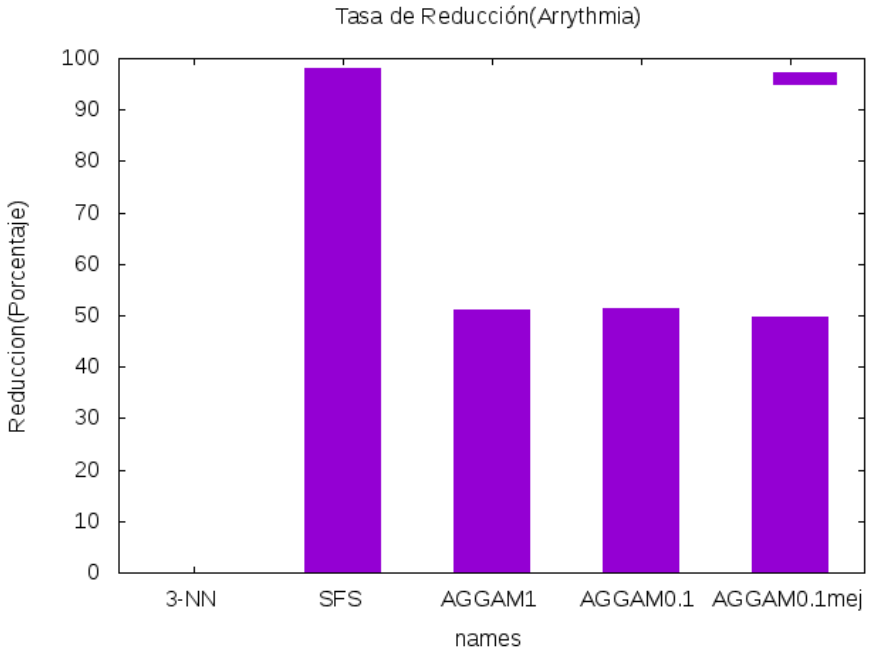

Figure 8: Tasa de reducción en la base de datos Arrythmia

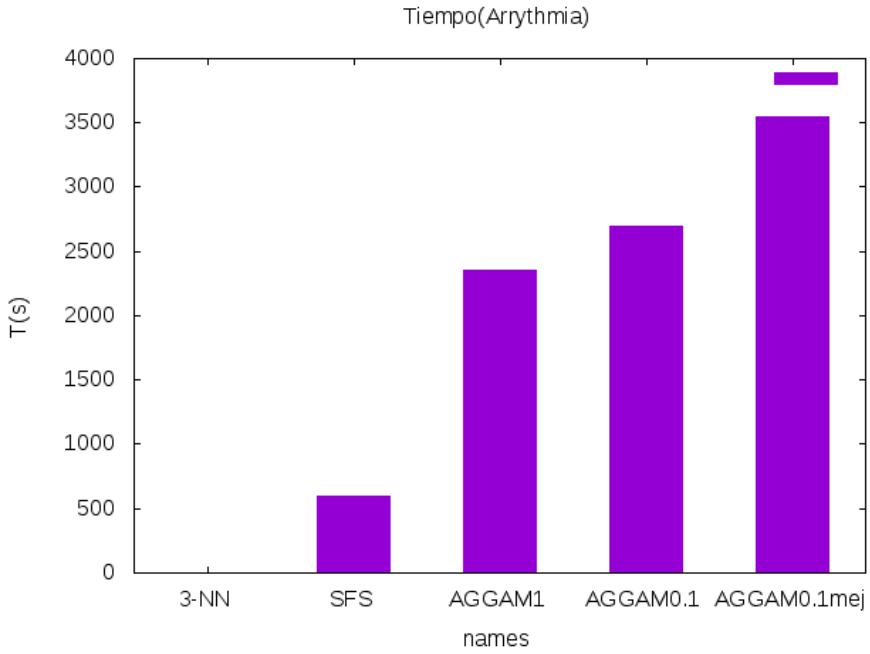

Figure 9: Tiempo de ejecución en la base de datos Arrythmia

Conclusiones Finales

Como podemos observar, el algoritmo Greedy supera en las gráficas excepto en la de tasa de clasificación de Wdbc. Claramente el hecho es que se ha reducido el número de iteraciones por lo que no se ha podido dar todos los buenos resultados que se podrían conseguir. Los algorítmos genéticos son buenos diversificando por lo que necesita muchas iteraciones para intensificar. Se puede observar la ayuda que le ofrece la búsqueda local, ya que se obtienen mejores resultados comparados con los de la Práctica 3 con menos iteraciones.
Los algoritmes meméticos me parecen una gran opción, ya que tienen un equilibrio muy bueno entre diversifiación e intensificación.