Universidad de Granada

# PRÁCTICA 2:
# Técnicas de Búsqueda basadas en Poblaciones para el Problema del Aprendizaje de Pesos en Características

Metaheurísticas.
Grupo 2. Martes 17:30-19:30

---

Realizado por:
Francisco Solano López Rodríguez
DNI: 20100444P
Email: fransol0728@correo.ugr.es

DOBLE GRADO EN INGENIERÍA INFORMÁTICA Y MATEMÁTICAS.
CUARTO CURSO

# Índice

# 1. Descripción del problema.

El problema del APC o Aprendizaje de Pesos en Características consiste en optimizar el rendimiento de un clasificador mediante la obtención de un vector de pesos con el que ponderar las carácteristicas de un objeto. Tendremos un conjunto de objetos $\{O_1, ...O_m\}$ donde cada $O_i$ tiene unas características asociadas $\{a_1, ...a_n, C\}$, en las cuales las primeras n características son atributos del objeto y la última característica C es la clase a la que pertenece el objeto. El vector de pesos con el que pretendemos ponderar dichos atributos vendrá dado por $\{w_1, ..., w_n\}$, donde cada $w_i$ pertenece al intervalo $[0, 1]$.

Vamos a usar el clasificador 1-NN, es decir la clase que vamos a asignar al objeto a clasificar es la del vecino más cercano. Para determinar la distancia de un objeto a otro usaremos la siguiente distancia:

$$d_e(e_1, e_2) = \sqrt{\sum_i w_i(e_1^i - e_2^i)^2 + \sum_j w_j d_h(e_1^j, e_2^j)}$$

Donde $d_h$ corresponde a la distancia de Hamming para el caso de variables nominales, aunque en los conjuntos de datos que vamos a utilizar nosotros todas las variables son numéricas.

El objetivo será obtener un sistema que nos permita clasificar nuevos objetos de manera automática. Usaremos la técnica de validación cruzada 5-fold cross validation. Para ello dividiremos en 5 particiones disjuntas al 20 %, con la distribución de clases equilibrada. Aprenderemos el clasificador utilizando 4 de las particiones y validaremos con la partición restante. Este proceso se puede realizar de 5 formas diferentes con lo que obtendremos un total de 5 valores de porcentaje de clasificación en el conjunto de prueba.

Buscaremos optimizar tanto a precisión como la complejidad del clasificador. La función que queremos maximizar es la siguiente:

$$F(W) = \alpha \cdot tasa\_clas(w) + (1 - \alpha) \cdot tasa\_red(W)$$

En donde *tasa_clas* y *tasa_red* corresponden a las siguientes funciones:

$$tasa\_clas = 100 \cdot \frac{n^o\ instancias\ bien\ clasificadas\ en\ T}{n^o\ instancias\ en\ T}$$

$$tasa\_red = 100 \cdot \frac{n^o\ valores\ w_i < 0{,}2}{n^o\ caracteristicas}$$

T es el conjunto de objetos a clasificar, $\alpha \in [0, 1]$ pondera la importancia entre el acierto y la reducción de la solución encontrada y $W$ es el vector de pesos.

# 2. Descripción de la aplicación de los algoritmos empleados al problema.

Una solución en el problema del Aprendizaje de Pesos en Características es un vector de pesos $W = \{w_1, w_2, \cdots w_n\}$ con los que ponderar la distancia entre 2 objetos. cada valor $w_i \in [0, 1]$, en donde si el valor es menor de 0,2 la característica no se tiene en cuenta para el cálculo de la distancia, si es igual a 1 se tiene totalmente en cuenta y un valor intermedio ponderara la importancia de dicha característica.

La distancia entre dos objetos $e1$ y $e2$ ponderada por el vector $W$ viene dada por la siguiente función:

```
1  Funcion distancia(e^1, e^2, w)
2      sum = 0
3
4      Para i = 0 hasta w.size
5          Si w_i >= 0,2
6              sum = sum + w_i * (e_i^1 - e_i^2)^2
7
8      Devolver sqrt(sum)
9  Fin
```

Antes de utilizar los datos para la ejecución de los algoritmos, estos han sido normalizados. La función es la siguiente:

```
1  Funcion Normalizar(T)
2      Para i = 0 hasta num_atrib-1
3          max = min = T[0][i]
4
5          Para j = 1 hasta T.size
6              Si max < T[j][i]
7                  max = T[j][i]
8              Si min > T[j][i]
9                  min = T[j][i]
10
11         Para j = 0 hasta T.size
12             T[j][i] = (T[j][i]-min)/(max-min)
13 Fin
```

Al inicio del programa después de haber leído los datos y haberlos normalizado, pasamos a crear las particiones con las que realizaremos la técnica de validación cruzada 5-fold cross validation. Las particiones se crearan de forma que se mantengan en cada una la misma proporción de cada clase que había el conjunto original, para ello se insertaran primero los elementos de la clase 1 de forma rotatoria, y después los elementos de la clase 2.

```
1  Funcion crearParticiones()
2      j = 0
3      Para i = 0 hasta N
4          Si e[i][num_atrib-1] == 1
5              particiones[j%5].insertar(e[i])
6              j = j + 1
7      Para i = 0 hasta N
8          Si e[i][num_atrib-1] == 2
9              particiones[j%5].insertar(e[i])
```

```
10            j = j + 1
11    Fin
```

Para clasificar un nuevo dato $e_{new}$ usaremos el algoritmo 1-NN con distancia ponderada por un vector de pesos. Viene dada por el siguiente pseudocódigo, en donde el parámetro out indica el indice del elemento que vamos a dejar fuera en el 'leave one out'.

```
1   Funcion KNN(T, new_e, w, out = -1)
2       d_min = 9999999
3
4       Para i = 0 hasta T.size
5           Si out != i
6               d = distancia(T[i], new_e, w)
7               Si d < d_min
8                   c_min = T[i][T[i].size-1]
9                   d_min = d
10
11      Devolver c_min
12  Fin
```

La función objetivo es la combinación con pesos de la tasa de acierto y la complejidad del clasificador donde el valor de $\alpha$ considerado vale 0,5. El objetivo es maximizar dicha función. El parámetro Data corresponde al conjunto de datos sobre el que clasificaremos y T el conjunto de datos que pretendemos clasificar, el parámetro leave_one_out es un booleano que indica si se realizará dicha técnica, la cual será necesaria para clasificar al conjunto de entrenamiento.

```
1   Funcion F(Data, T, w, leave_one_out)
2       alpha = 0.5
3       Devolver alpha*tasaClas(Data, T, w, leave_one_out) + (1-alpha)*tasaRed(w)
4   Fin
```

La función tasaClas calcula la tasa de acierto del clasificador contando el número de aciertos y devolviendo el porcentaje de acierto que ha tenido.

```
1   Funcion tasaClas(Data, T, w, leave_one_out)
2       clasify_ok = 0
3
4       Para i = 0 hasta T.size
5           Si leave_one_out = true
6               out = i
7           Si No
8               out = -1
9
10          Si clasify(Data, T[i], w, out) = T[i][T[i].size-1]
11              clasify_ok = clasify_ok + 1
12
13      Devolver 100.0*clasify_ok/T.size
14  Fin
```

La función tasaRed calcula la tasa de reducción de características con respecto al conjunto original, para ello cuenta el número de elementos del vector de pesos cuyo valor esta por debajo de 0,2, los cuales no serán tomados en cuenta en el cálculo de la distancia.

```
Funcion tasaRed(w)
    num = 0

    Para i = 0 hasta w.size
        Si w_i < 0,2
            num = num + 1

    Devolver 100.0*num/w.size
Fin
```

La generación de un vecino se realizará mediante la alteración de una componente del vector de pesos W.

$$Mov(W, \sigma) = (w_1, \cdots, w_i + z_i, \cdots, w_n)$$

Donde $z_i \sim N(0; 0{,}4)$, es decir es un valor aleatorio que sigue una distribución normal de media 0 y varianza 0,4. Si el valor de $w_i$ queda fuera de su dominio lo trucamos a $[0, 1]$.

```
Funcion nuevoVecino(w, i)
    z = aleatorio ~ Normal(0, 0.4)
    w_i = w_i + z

    Si w_i > 1
        w_i = 1
    Si w_i < 0
        w_i = 0

    Devolver w
Fin
```

Para las poblaciones de soluciones en los algoritmos genéticos se ha usado una estructura llamada cromosoma que almacena un vector solución $w$, junto con el valor de la función objetivo obtenido con dicho vector $w$. La población estará formada por un vector de cromosomas.

```
    cromosoma{w, valor}
```

A continuación se muestra la función utilizada para generar una población de soluciones aleatorias.

```
Funcion generarPoblacion(poblacion, T, tam_p, num_atrib)

    Para i = 0 hasta tam_p
        w = {}

        Para j = 0 hasta num_atrib-1
            w.insertar(aleatorio ∈ [0,1])

        poblacion.insertar(cromosoma(w, F(T,T,w)))

Fin
```

### Operadores de cruce

Para los algoritmos genéticos se han utilizados dos operadores de cruce diferentes, estos son:

- Cruce aritmético

```
Funcion cruceAritmético(cromosoma1, cromosoma2, T)
    w = {}

    Para i = 0 hasta N
        w.insertar((cromosoma1.w_i + cromosoma2.w_i)/2)

    Devolver cromosoma(w, F(T,T,w))
Fin
```

-Cruce BLX

```
Funcion cruceBLX(cromosoma1, cromosoma2, T)
    w = {}
    a = 0.3

    Para i = 0 hasta N
        c_max = max(cromosoma1.w_i, cromosoma2.w_i)
        c_min = min(cromosoma1.w_i, cromosoma2.w_i)

        I = c_max−c_min

        num = aleatorio ∈ [c_min − I ∗ a, c_max + I ∗ a]

        Si num > 1
            num = 1
        Si num < 0
            num = 0

        w.insertar(num)

    Devolver cromosoma(w, F(T,T,w))
Fin
```

### Mecanismo de selección: Torneo Binario

```
Funcion torneoBinario(poblacion)
    tam_p = poblacion.size

    i1 = entero aleatorio ∈ [0,tam_p−1]

    Hacer
        i2 = entero aleatorio ∈ [0,tam_p−1]
    Mientras i1 = i2

    Si poblacion[i1].valor > poblacion[i2].valor
        Devolver poblacion[i1].valor
    Si No
        Devolver poblacion[i2].valor
Fin
```

El mecanismo de mutación para los genéticos utilizará la misma función que se utiliza para generar un nuevo vecino.

# 3. Algoritmos genéticos

## 3.1. Algoritmo genético estacionario

```
1  Funcion Estacionario(T, (Funcion) cruce)
2     tam_p = 30
3
4     generarPoblacion(poblacion, T, tam_p, num_atrib)
5
6     evaluaciones = 0
7
8     Mientras evaluaciones < 15000
9        padre1 = torneoBinario(poblacion)
10       padre2 = torneoBinario(poblacion)
11       hijo1 = cruce(padre1, padre2, T)
12       evaluaciones = evaluaciones + 1
13
14       padre1 = torneoBinario(poblacion)
15       padre2 = torneoBinario(poblacion)
16       hijo1 = cruce(padre1, padre2, T)
17       evaluaciones = evaluaciones + 1
18
19       Para j = 0 hasta num_atrib-1
20          Si (aleatorio ∈ [0,1] < 0.001)
21             Mutar(hijo1.w_i)
22             hijo1.valor = F(T,T,hijo1.w)
23             evaluaciones = evaluaciones + 1
24          Si (aleatorio ∈ [0,1] < 0.001)
25             Mutar(hijo2.w_i)
26             hijo2.valor = F(T,T,hijo2.w)
27             evaluaciones = evaluaciones + 1
28
29       competicion(poblacion, hijo1, hijo2)
30
31    mejor = 0
32
33    Para i = 1 hasta tam_p
34       Si poblacion[i].valor > poblacion[mejor].valor
35          mejor = i
36
37    Devolver poblacion[mejor].w
38
39 Fin
```

En el algoritmo estacionario, tras la generación de los dos nuevos hijos, estos tienen que competir por entrar en la población. Entraran en la población si son mejores que los dos peores de la población. Dicha función está dada en el siguiente pseudocódigo.

```
1  Funcion competicion(poblacion, hijo1, hijo2)
2
3     Ordenar(poblacion, (ordenar segun cromosoma.valor))
4
5     Si hijo1.valor >= hijo2.valor
6        Si poblacion[tam_p-1].valor < hijo1.valor
7           poblacion[tam_p-1] = hijo1
8           Si poblacion[tam_p-2].valor < hijo2.valor
9              poblacion[tam_p-2].valor = hijo2
10
11    Si No Si poblacion[tam_p-1].valor < hijo2.valor
12       poblacion[tam_p-1] = hijo2
13       Si poblacion[tam_p-2].valor < hijo1.valor
14          poblacion[tam_p-2].valor = hijo1
15
16 Fin
```

## 3.2. Algoritmo genético generacional

Esta función además de corresponderse con un algoritmo genético generacional, también permite hacer búsqueda local sobre ciertas soluciones cada cierto número de generaciones, mediante el parámetro memético. Las opciones son:

- memetico = 0: AM-(10,1.0)
- memetico = 1: AM-(10,0.1)
- memetico = 2: AM-(10,0.1mej)

```
1  Funcion Generacional(T, (Funcion) cruce, memetico = -1)
2     Si memetico == -1
3        tam_p = 30
4     Si No
5        tam_p = 10
6
7     generarPoblacion(poblacion, T, tam_p, num_atrib)
8
9     mejor = 0, peor = 0
10
11    Para i = 1 hasta tam_p-1
12       Si poblacion[i].valor > poblacion[mejor].valor
13          mejor = i
14
15    evaluaciones = 0
16    generaciones = 0
17
18    Mientras evaluaciones < 15000
19       mejor_cromosoma_old = poblacion[mejor]
20
21       Si memetico = 0 y generaciones%10 = 0
22          Para j = 0 hasta tam_p-1
23             poblacion[j] = BL(T, poblacion[j], evaluaciones)
24       Si memetico = 1 y generaciones%10 = 0
25          Para j = 0 hasta tam_p-1
26             Si (aleatorio ∈ [0,1]) <= 0.1
27                poblacion[j] = BL(T, poblacion[j], evaluaciones)
28       Si memetico = 2 y generaciones%10 = 0
29          poblacion[mejor] = BL(T, poblacion[mejor], evaluaciones)
30
31       Para j = 0 hasta tam_p
32          Si (aleatorio ∈ [0,1]) < 0.7
33             padre1 = torneoBinario(poblacion)
34             padre2 = torneoBinario(poblacion)
35             hijo = cruce(padre1, padre2, T)
36             evaluaciones = evaluaciones + 1
37
38             Para k = 0 hasta num_atrib-1
39                Si (aleatorio ∈ [0,1] <= 0.001)
40                   Mutar(hijo1.w_i)
41                   hijo.valor = F(T,T,hijo.w)
42                   evaluaciones = evaluaciones + 1
43
44             poblacion[j] = hijo
45
46          Si poblacion[j].valor > poblacion[mejor].valor
47             mejor = j
48          Si poblacion[j].valor < poblacion[peor].valor
49             peor = j
50
51       generaciones = generaciones + 1
52       poblacion[peor] = mejor_cromosoma_old
53
54    Devolver poblacion[mejor].w
55 Fin
```

La búsqueda local utilizada en el AGG viene dada por el siguiente pseudocódigo.

```
1  Funcion BL(T, cromosoma, evaluaciones)
2
3      Para i = 0 hasta num_atributos-1
4          indices[i] = i
5
6      w = cromosoma.w
7      valor = F(T,T,cromosoma.valor)
8
9      iteraciones = 0
10
11     Mientras iteraciones < 2*num_atributos
12
13         Si iteraciones%indices.size == 0
14             shuffle(indices)
15
16         k = indices[iteraciones%indices.size]
17         copia = w_k
18
19         nuevoVecino(w, k)
20
21         nuevo_valor = F(T, w)
22         iteraciones = iteraciones + 1
23         nn = nn + 1
24         evaluaciones = evaluaciones + 1
25
26         Si nuevo_valor > valor
27             nn = 0
28             valor = nuevo_valor
29         Si No
30             w[k] = copia
31
32     Devolver cromosoma(w, F(T,T,w))
33
34 Fin
```

# 4. Procedimiento considerado para el desarrollo de la práctica y manual de usuario.

La práctica ha sido realizada en C++, el código en su mayoría ha sido desarrollado por mi, incluyendo la lectura de datos. Para la generación de números pseudoaleatorios he utilizado la implemetación aportada en la web de la asignatura, a la cual le he añadido una función para obtener números aleatorios que sigan una distribución normal y un método para permutar de forma aleatoria un vector. Para medir los tiempos de ejecución de los algoritmos he utilizado la biblioteca ctime.

Para no obtener resultados diferentes cada vez que se ejecute el programa, he fijado la semilla para los número pseudoaleatorios con el valor 17.

Para poder ejecutar el programa se ha incluido un makefile en la carpeta FUENTES, por lo que para generar el ejecutable tan solo se deberá de escribir `make` en la terminal. La compilación se ha realizado utilizando clang++ por lo que debería poder compilarse en un Mac. Yo en mi caso he realizado la práctica en Ubuntu.

Podemos ejecutar la práctica escribiendo `./practica2`, tras lo cual se mostrará el mensaje siguiente por pantalla:

```
Pulse el número que desee ejecutar:
1: ozone-320.arff (1-NN, relief, BL)
2: parkinsons.arff (1-NN, relief, BL)
3: spectf-heart.arff (1-NN, relief, BL)
```

Después de elegir el fichero de datos, que vamos a utilizar para la ejecución, tendremos que elegir el algoritmo que deseamos ejecutar:

```
Elija el algoritmo que desee ejecutar:
1: Estacionario Media Aritmética
2: Estacionario BLX
3: Generacional Media Aritmética
4: Generacional BLX
5: Memético Media Aritmética
6: Memético BLX
Voluntario:
7: Estacionario Low
8: Generacional Low
9: Memetico Low
```

Tras pulsar alguno de los números se ejecutarán los algoritmos indicados utilizando los datos del fichero elegido, y se mostrarán los resultados obtenidos.

# 5. Experimentos y análisis de resultados.

Bases de datos utilizadas:

- **Ozone:** base de datos para la detección del nivel de ozono, consta de 320 ejemplos, cada uno con 73 atributos y consta de 2 clases.
- **Parkinsons:** base de datos utilizada para distinguir entre la presencia y la ausencia de la enfermedad. Consta de 195 ejemplos, con 23 atributos y 2 clases.
- **Spectf-heart:** base de datos utilizada para determinar si la fisiología del corazón analizado es correcta o no. Consta de 267 ejemplos con 45 atributos y 2 clases.

Comentar que los ficheros de datos proporcionados contenían más ejemplos de los comentados, el motivo era que había varias líneas repetidas, con lo cual muchos ejemplos aparecían varias veces. Para evitar esto he filtrado los datos para eliminar repetidos y con ello ya se cumplen las cifras comentadas. Además los datos han sido también normalizados utilizando la fórmula

$$x_j^N = (x_j - Min_j)/(Max_j - Min_j)$$

Las prácticas han sido implementadas en C++, y ejecutadas en un ordenador con procesador Intel Core i3, 12 GB de RAM y disco duro SSD en el sistema operativo Ubuntu 16.04 LTS.

### Resultados obtenidos

Primero mostramos los resultados que obtuvimos en la primera práctica para 1-NN, relief y BL.

| **1-NN** | **Ozone** | | | | **Parkinsons** | | | | **Spectf-heart** | | | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|
| | **%clas** | **%red** | **Agr.** | **T** | **%clas** | **%red** | **Agr.** | **T** | **%clas** | **%red** | **Agr.** | **T** |
| **P1** | 79.68 | 0 | 39.84 | 0.0021 | 97.43 | 0 | 48.71 | 0.0006 | 75.92 | 0 | 37.96 | 0.0015 |
| **P2** | 82.81 | 0 | 41.40 | 0.0021 | 94.87 | 0 | 47.43 | 0.0006 | 64.81 | 0 | 32.40 | 0.0015 |
| **P3** | 81.25 | 0 | 40.62 | 0.0021 | 94.87 | 0 | 47.43 | 0.0006 | 67.92 | 0 | 33.96 | 0.0019 |
| **P4** | 77.77 | 0 | 38.88 | 0.0022 | 97.43 | 0 | 48.71 | 0.0005 | 71.69 | 0 | 35.84 | 0.0015 |
| **P5** | 80.95 | 0 | 40.47 | 0.0024 | 97.43 | 0 | 48.71 | 0.0002 | 73.58 | 0 | 36.79 | 0.0015 |
| **Media** | 80.49 | 0 | 40.24 | 0.0022 | 96.41 | 0 | 48.20 | 0.0005 | 70.78 | 0 | 35.39 | 0.0016 |

| **Relief** | **Ozone** | | | | **Parkinsons** | | | | **Spectf-heart** | | | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|
| | **%clas** | **%red** | **Agr.** | **T** | **%clas** | **%red** | **Agr.** | **T** | **%clas** | **%red** | **Agr.** | **T** |
| **P1** | 82.81 | 13.88 | 48.35 | 0.0225 | 94.87 | 4.545 | 49.70 | 0.0034 | 83.33 | 38.63 | 60.98 | 0.0108 |
| **P2** | 78.12 | 18.05 | 48.09 | 0.0202 | 94.87 | 4.545 | 49.70 | 0.0031 | 70.37 | 38.63 | 54.50 | 0.0107 |
| **P3** | 79.68 | 19.44 | 49.56 | 0.0198 | 97.43 | 4.545 | 50.99 | 0.0036 | 69.81 | 36.36 | 53.08 | 0.0100 |
| **P4** | 80.95 | 13.88 | 47.42 | 0.0200 | 97.43 | 4.545 | 50.99 | 0.0032 | 75.47 | 43.18 | 59.32 | 0.0096 |
| **P5** | 79.36 | 26.38 | 52.87 | 0.0206 | 97.43 | 0 | 48.71 | 0.0040 | 67.92 | 40.90 | 54.41 | 0.0116 |
| **Media** | 80.18 | 18.33 | 49.26 | 0.0206 | 96.41 | 3.636 | 50.02 | 0.0034 | 73.38 | 39.54 | 56.46 | 0.0106 |

| BL | Ozone | | | | Parkinsons | | | | Spectf-heart | | | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|
| | %clas | %red | Agr. | T | %clas | %red | Agr. | T | %clas | %red | Agr. | T |
| P1 | 79.68 | 79.16 | 79.42 | 16.652 | 89.74 | 81.81 | 85.78 | 1.1339 | 72.22 | 63.63 | 67.92 | 5.5271 |
| P2 | 76.56 | 81.94 | 79.25 | 26.105 | 89.74 | 81.81 | 85.78 | 0.8149 | 72.22 | 68.18 | 70.20 | 10.118 |
| P3 | 75 | 70.83 | 72.91 | 21.232 | 100 | 72.72 | 86.36 | 0.6046 | 71.69 | 75 | 73.34 | 8.0806 |
| P4 | 71.42 | 77.77 | 74.60 | 16.267 | 92.30 | 90.90 | 91.60 | 0.9606 | 71.69 | 81.81 | 76.75 | 10.382 |
| P5 | 77.77 | 84.72 | 81.25 | 22.167 | 94.87 | 72.72 | 83.79 | 0.5796 | 81.13 | 79.54 | 80.33 | 7.2339 |
| Media | 76.09 | 78.88 | 77.49 | 20.484 | 93.33 | 80 | 86.66 | 0.8187 | 73.79 | 73.63 | 73.71 | 8.2686 |

Los resultados que se muestran a continuación son los correspondientes a esta práctica. Los tiempos obtenidos son más bajos de lo que se podría esperar, esto es debido a que se paralelizaron ciertos bloques del código para ganar velocidad en la ejecución. En el código aportado he suprimido dicha paralelización para que no sea necesario instalar ningún nuevo paquete para poder compilar.

| AGG BLX | Ozone | | | | Parkinsons | | | | Spectf-heart | | | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|
| | %clas | %red | Agr. | T | %clas | %red | Agr. | T | %clas | %red | Agr. | T |
| P1 | 75 | 75 | 75 | 50.745 | 89.74 | 86.36 | 88.05 | 6.6586 | 77.77 | 81.81 | 79.79 | 23.146 |
| P2 | 76.56 | 72.22 | 74.39 | 50.144 | 97.43 | 72.72 | 85.08 | 6.1674 | 75.92 | 56.81 | 66.37 | 26.487 |
| P3 | 79.68 | 72.22 | 75.95 | 49.902 | 94.87 | 81.81 | 88.34 | 6.5433 | 67.92 | 68.18 | 68.05 | 23.576 |
| P4 | 79.36 | 70.83 | 75.09 | 50.933 | 89.74 | 86.36 | 88.05 | 5.9463 | 67.92 | 65.90 | 66.91 | 24.273 |
| P5 | 65.07 | 76.38 | 70.73 | 51.18 | 89.74 | 77.27 | 83.50 | 6.7124 | 75.47 | 68.18 | 71.82 | 24.454 |
| Media | 75.13 | 73.33 | 74.23 | 50.581 | 92.30 | 80.90 | 86.60 | 6.4056 | 73.00 | 68.18 | 70.59 | 24.387 |

| AGG CA | Ozone | | | | Parkinsons | | | | Spectf-heart | | | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|
| | %clas | %red | Agr. | T | %clas | %red | Agr. | T | %clas | %red | Agr. | T |
| P1 | 79.68 | 51.38 | 65.53 | 55.403 | 97.43 | 59.09 | 78.26 | 6.8811 | 79.62 | 40.90 | 60.26 | 26.455 |
| P2 | 81.25 | 56.94 | 69.09 | 54.439 | 89.74 | 50 | 69.87 | 7.9504 | 68.51 | 54.54 | 61.53 | 26.069 |
| P3 | 81.25 | 47.22 | 64.23 | 55.340 | 100 | 45.45 | 72.72 | 12.308 | 67.92 | 45.45 | 56.68 | 26.835 |
| P4 | 79.36 | 51.38 | 65.37 | 55.399 | 92.30 | 81.81 | 87.06 | 6.5982 | 73.58 | 54.54 | 64.06 | 25.630 |
| P5 | 84.12 | 36.11 | 60.11 | 58.377 | 87.17 | 77.27 | 82.22 | 6.5041 | 75.47 | 29.54 | 52.50 | 28.267 |
| Media | 81.13 | 48.61 | 64.87 | 55.792 | 93.33 | 62.72 | 78.03 | 8.0484 | 73.02 | 45 | 59.01 | 26.651 |

| AGE BLX | Ozone | | | | Parkinsons | | | | Spectf-heart | | | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|
| | %clas | %red | Agr. | T | %clas | %red | Agr. | T | %clas | %red | Agr. | T |
| P1 | 79.68 | 77.77 | 78.73 | 46.239 | 84.61 | 86.36 | 85.48 | 5.8437 | 68.51 | 72.72 | 70.62 | 23.529 |
| P2 | 81.25 | 68.05 | 74.65 | 47.465 | 87.17 | 90.90 | 89.04 | 6.3572 | 68.51 | 68.18 | 68.35 | 23.110 |
| P3 | 76.56 | 76.38 | 76.47 | 46.260 | 92.30 | 81.81 | 87.06 | 6.1293 | 69.81 | 75 | 72.40 | 23.014 |
| P4 | 77.77 | 70.83 | 74.30 | 47.055 | 94.87 | 90.90 | 92.89 | 6.5164 | 77.35 | 81.81 | 79.58 | 23.266 |
| P5 | 73.01 | 68.05 | 70.53 | 47.800 | 82.05 | 86.36 | 84.20 | 6.1684 | 66.03 | 70.45 | 68.24 | 23.149 |
| Media | 77.65 | 72.22 | 74.94 | 46.964 | 88.20 | 87.27 | 87.73 | 6.2030 | 70.04 | 73.63 | 71.84 | 23.213 |

| AGE CA | Ozone | | | | Parkinsons | | | | Spectf-heart | | | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|
| | %clas | %red | Agr. | T | %clas | %red | Agr. | T | %clas | %red | Agr. | T |
| P1 | 71.87 | 68.05 | 69.96 | 49.715 | 89.74 | 86.36 | 88.05 | 6.4909 | 68.51 | 68.18 | 68.35 | 25.079 |
| P2 | 85.93 | 70.83 | 78.38 | 51.913 | 84.61 | 81.81 | 83.21 | 6.5580 | 70.37 | 61.36 | 65.86 | 23.942 |
| P3 | 78.12 | 68.05 | 73.09 | 48.008 | 97.43 | 77.27 | 87.35 | 6.4204 | 69.81 | 61.36 | 65.58 | 24.318 |
| P4 | 80.95 | 65.27 | 73.11 | 49.824 | 89.74 | 72.72 | 81.23 | 7.1018 | 75.47 | 63.63 | 69.55 | 24.921 |
| P5 | 82.53 | 70.83 | 76.68 | 48.002 | 89.74 | 77.27 | 83.50 | 6.5423 | 77.35 | 59.09 | 68.22 | 23.452 |
| Media | 79.88 | 68.61 | 74.24 | 49.492 | 90.25 | 79.09 | 84.67 | 6.6227 | 72.30 | 62.72 | 67.51 | 24.342 |

Puesto que los resultados han sido mejores utilizando el cruce BLX, los algoritmos meméticos han sido ejecutados usando dicho cruce.

| AM (10,1.0) | Ozone | | | | Parkinsons | | | | Spectf-heart | | | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|
| | %clas | %red | Agr. | T | %clas | %red | Agr. | T | %clas | %red | Agr. | T |
| P1 | 68.75 | 80.55 | 74.65 | 48.040 | 92.30 | 90.90 | 91.60 | 5.8251 | 77.77 | 77.27 | 77.52 | 21.678 |
| P2 | 70.31 | 83.33 | 76.82 | 47.371 | 89.74 | 86.36 | 88.05 | 6.4872 | 66.66 | 86.36 | 76.51 | 22.520 |
| P3 | 84.37 | 83.33 | 83.85 | 46.589 | 97.43 | 90.90 | 94.17 | 6.1924 | 62.26 | 84.09 | 73.17 | 22.747 |
| P4 | 77.77 | 79.16 | 78.47 | 46.649 | 87.17 | 86.36 | 86.77 | 6.0519 | 58.49 | 88.63 | 73.56 | 23.295 |
| P5 | 74.60 | 84.72 | 79.66 | 47.239 | 87.17 | 86.36 | 86.77 | 7.1323 | 66.03 | 72.72 | 69.38 | 23.918 |
| Media | 75.16 | 82.22 | 78.69 | 47.178 | 90.76 | 88.18 | 89.47 | 6.3378 | 66.24 | 81.81 | 74.03 | 22.832 |

| AM (10,0.1) | Ozone | | | | Parkinsons | | | | Spectf-heart | | | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|
| | %clas | %red | Agr. | T | %clas | %red | Agr. | T | %clas | %red | Agr. | T |
| P1 | 78.12 | 81.94 | 80.03 | 45.311 | 84.61 | 86.36 | 85.48 | 5.9047 | 72.22 | 86.36 | 79.29 | 21.374 |
| P2 | 71.87 | 81.94 | 76.90 | 45.369 | 79.48 | 90.90 | 85.19 | 5.8966 | 75.92 | 75 | 75.46 | 22.536 |
| P3 | 81.25 | 70.83 | 76.04 | 47.104 | 97.43 | 86.36 | 91.89 | 5.8489 | 79.24 | 81.81 | 80.53 | 21.946 |
| P4 | 66.66 | 80.55 | 73.61 | 48.964 | 89.74 | 81.81 | 85.78 | 6.6325 | 73.58 | 79.54 | 76.56 | 22.326 |
| P5 | 84.12 | 80.55 | 82.34 | 46.031 | 92.30 | 90.90 | 91.60 | 5.8514 | 66.03 | 79.54 | 72.79 | 24.435 |
| Media | 76.40 | 79.16 | 77.78 | 46.556 | 88.71 | 87.27 | 87.99 | 6.0268 | 73.40 | 80.45 | 76.92 | 22.523 |

| AM(10, 0.1mej) | Ozone | | | | Parkinsons | | | | Spectf-heart | | | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|
| | %clas | %red | Agr. | T | %clas | %red | Agr. | T | %clas | %red | Agr. | T |
| P1 | 78.12 | 75 | 76.56 | 45.908 | 84.61 | 86.36 | 85.48 | 6.4051 | 70.37 | 68.18 | 69.27 | 22.124 |
| P2 | 76.56 | 83.33 | 79.94 | 45.647 | 82.05 | 86.36 | 84.20 | 6.9623 | 74.07 | 56.81 | 65.44 | 23.428 |
| P3 | 76.56 | 72.22 | 74.39 | 46.875 | 97.43 | 81.81 | 89.62 | 6.1036 | 69.81 | 77.27 | 73.54 | 21.488 |
| P4 | 79.36 | 81.94 | 80.65 | 46.19 | 82.05 | 86.36 | 84.20 | 6.1007 | 75.47 | 84.09 | 79.78 | 22.621 |
| P5 | 76.19 | 83.33 | 79.76 | 47.919 | 89.74 | 90.90 | 90.32 | 5.9096 | 77.35 | 84.09 | 80.72 | 21.387 |
| Media | 77.36 | 79.16 | 78.26 | 46.508 | 87.17 | 86.36 | 86.77 | 6.2963 | 73.41 | 74.09 | 73.75 | 22.21 |

| | Ozone | | | | Parkinsons | | | | Spectf-heart | | | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|
| | %clas | %red | Agr. | T | %clas | %red | Agr. | T | %clas | %red | Agr. | T |
| **1-NN** | 80.49 | 0 | 40.24 | 0.0022 | 96.41 | 0 | 48.20 | 0.0005 | 70.78 | 0 | 35.39 | 0.0016 |
| **Relief** | 80.18 | 18.33 | 49.26 | 0.0206 | 96.41 | 3.636 | 50.02 | 0.0034 | 73.38 | 39.54 | 56.46 | 0.0106 |
| **BL** | 76.09 | 78.88 | 77.49 | 20.484 | 93.33 | 80 | 86.66 | 0.8187 | 73.79 | 73.63 | 73.71 | 8.2686 |
| **G BLX** | 75.13 | 73.33 | 74.23 | 50.581 | 92.30 | 80.90 | 86.60 | 6.4056 | 73.00 | 68.18 | 70.59 | 24.387 |
| **G CA** | 81.13 | 48.61 | 64.87 | 55.792 | 93.33 | 62.72 | 78.03 | 8.0484 | 73.02 | 45 | 59.01 | 26.651 |
| **E BLX** | 77.65 | 72.22 | 74.94 | 46.964 | 88.20 | 87.27 | 87.73 | 6.2030 | 70.04 | 73.63 | 71.84 | 23.213 |
| **E CA** | 79.88 | 68.61 | 74.24 | 49.492 | 90.25 | 79.09 | 84.67 | 6.6227 | 72.30 | 62.72 | 67.51 | 24.342 |
| **M(10,1.0)** | 75.16 | 82.22 | 78.69 | 47.178 | 90.76 | 88.18 | 89.47 | 6.3378 | 66.24 | 81.81 | 74.03 | 22.832 |
| **M(10,0.1)** | 76.40 | 79.16 | 77.78 | 46.556 | 88.71 | 87.27 | 87.99 | 6.0268 | 73.40 | 80.45 | 76.92 | 22.523 |
| M(10,0.1mej) | 77.36 | 79.16 | 78.26 | 46.508 | 87.17 | 86.36 | 86.77 | 6.2963 | 73.41 | 74.09 | 73.75 | 22.21 |

## Análisis de la tasa de clasificación

En primer lugar vamos a analizar la tasa de clasificación. Para facilitar la interpretación de los resultados se muestra la siguiente gráfica.

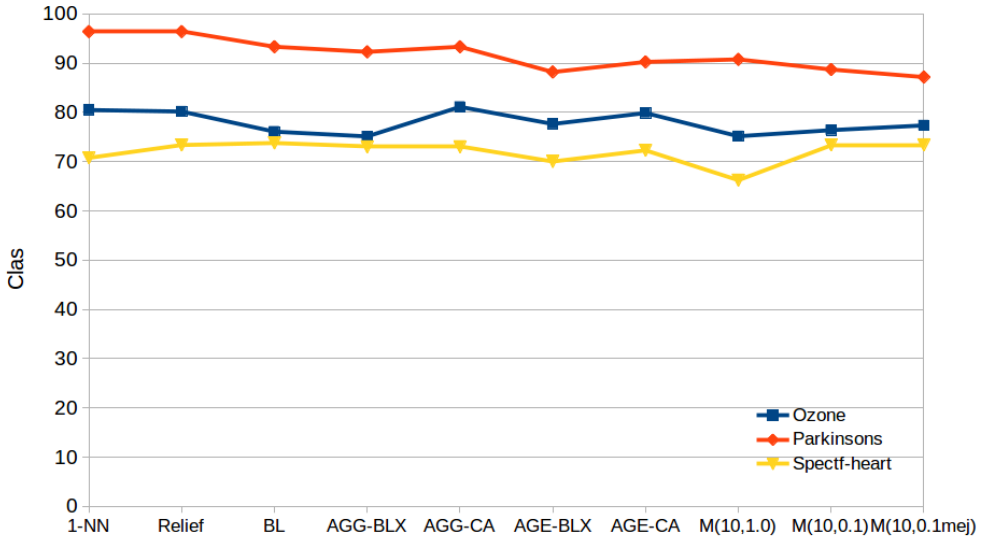


Figura 1: tasaClas

En lo que respecta a la tasa de clasificación no se aprecian grandes diferencias entre los algoritmos. Tal vez pueda resultar curioso que los algoritmos más simples (1-NN, relief), en el caso de los datos para parkinsons han sido mejores que los demás algoritmos. Esto no indica que los otros sean peores, ya que los otros buscaban un equilibrio entre la tasa de clasificación y la tasa de reducción, lo cual puede ser el motivo de que en la tasa de clasificación para el parkinsons se hayan visto superados por 1-NN y relief, los cuales solo buscaban maximizar el acierto.

Si nos fijamos en las diferencias entre el cruce BLX y el cruce aritmético, podemos apreciar que el aritmético supera en la tasa de clasificación al BLX en los tres conjuntos de datos y en los dos genéticos tanto el generacional como el estacionario. Creo que

puede ser debido a que el cruce aritmético dificulta la mejora en la tasa de reducción ($hijo.w_i = (\text{c1.w}_\text{i} + \text{c2.w}_\text{i})/2$ y entonces $hijo.w_i \geq min\{c1.w_i, c2.w_i\}$), ya que los mínimos $w_i$ de la población solo van a poder bajar su valor si por casualidad se produce una mutación que reduzca su valor. La dificultad en la mejora de la tasa de reducción puede verse compensada con mayores valores en la tasa de acierto.

### Análisis de la tasa de reducción

Ahora pasamos a analizar la tasa de reducción.

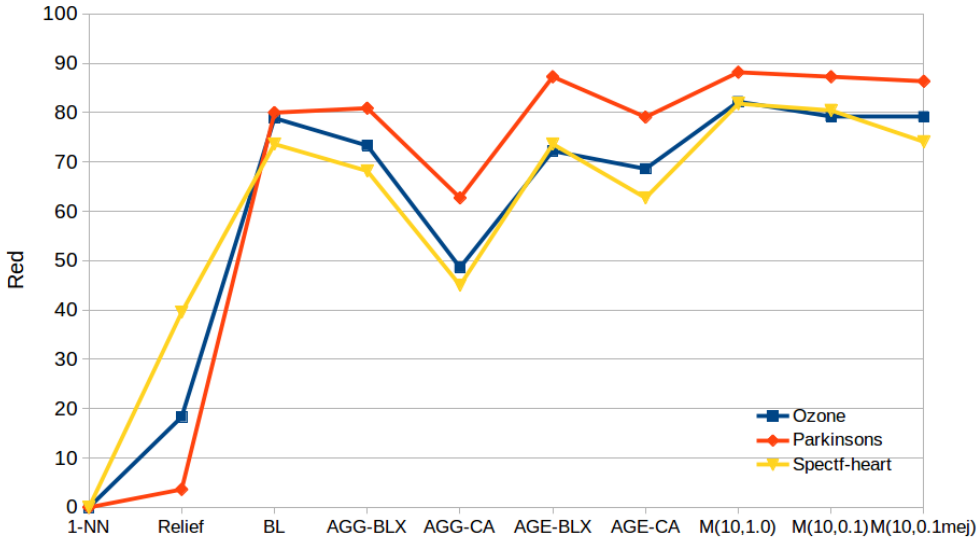


Figura 2: tasaRed

Como es lógico 1-NN tiene un 0 en tasa de reducción, ya que su vector de pesos es directamente de unos. El relief tiene un bajo valor en tasaRed como era de esperar, ya que el algoritmo busca maximizar la tasa de acierto y no tiene como objetivo mejorar la tasa de reducción.

De nuevo hacemos una comparación entre el cruce BLX y aritmético. Esta vez vemos que el BLX supera al cruce aritmético el cual ha tenido unos resultados bastante peores, sobre todo en el generacional. Como ya se ha comentado anteriormente el cruce aritmético dificultad que la tasa de reducción mejore, pues el valor mínimo del atributo iésimo no va a poder reducirse con una media aritmética.

Los mejores resultados en la tasa de reducción han sido obtenidos por los algoritmos meméticos. Los cuales han llegado a superar al BL.

### Análisis del agregado

A continuación vamos a analizar el valor de agregado obtenido por los algoritmos.

Los que han obtenido mejores resultados, en el valor de agregado, han sido los meméticos, siendo el AM-(10,1.0) el mejor para parkinsons y ozone y AM-(10,0.1) para spectf-heart. Estos algoritmos se venefician de la capacidad para explorar el espacio de soluciones que tienen los genéticos y de la rapidez en acercarse a óptimos locales del BL, y han demostrado dar con buenos resultados, superando tanto a los genéticos como al BL.

Los algoritmos genéticos han obtenido resultados por debajo del BL, al contrario de lo que se podría creer debido a su gran potencial para explorar el espacio de soluciones. Esto puede ser debido a que el número de evaluaciones de la función objetivo para los genéticos, solo ha sido de 15000, un número bastante bajo.

También comentar que el cruce BLX ha obtenido valores similares en el generacional y el estacionario, pero el cruce aritmético ha obtenido resultados bastante mejores en el estacionario. Seguramente debido a que en el generacional los valores mínimos de los atributos aumentaban rapidamente tras hacer la media aritmética haciendo que la tasa de reducción se viese perjudicada.

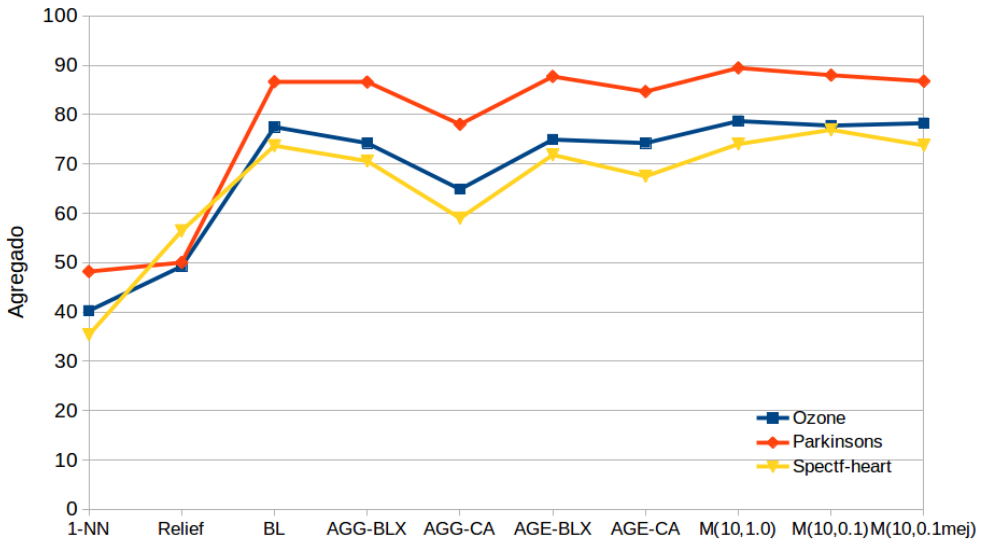


Figura 3: Agregado

### Análisis del tiempo de ejecución

Por último comparemos los tiempos de ejecución. Tener en cuenta al observar los resultados que los genéticos y meméticos han sido ejecutados con paralelización, mientras que los demás no.

El tiempo de ejecución los algoritmos 1-NN y relief ha sido muy bajo por lo que si el tiempo es un factor a tener en cuenta y solo nos interesa la tasa de acierto, estos algoritmo pueden ser de gran utilidad.

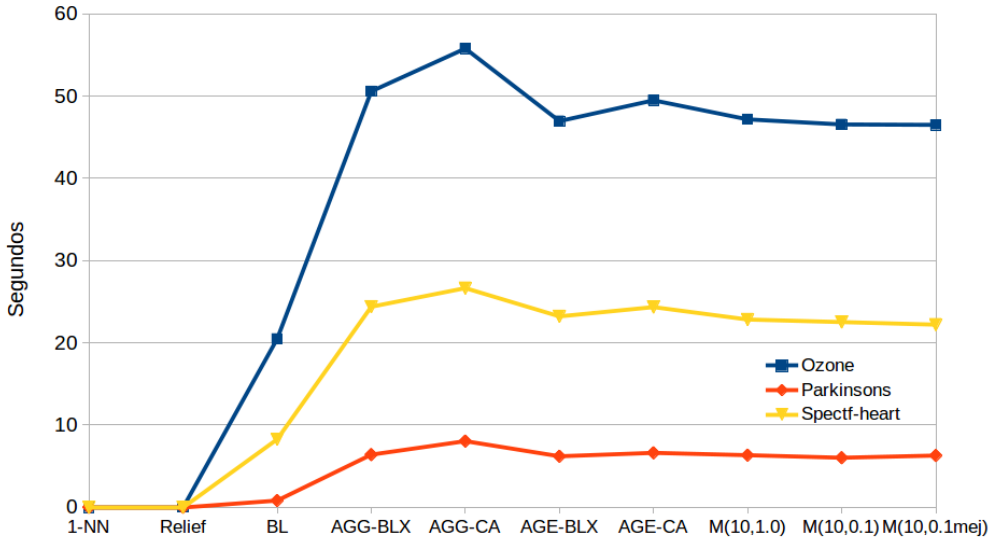


Figura 4: Tiempos

Si comparamos el BL con los genéticos, vemos que hay una gran diferencia de tiempo entre estos, y además estos últimos obtuvieron peores resultados que el BL (lo cual como ya se dijo puede ser debido al reducido número de evaluaciones: 15000). Luego si tuviera que coger alguno de ellos, elegiría el BL.

Si ahora comparamos BL con los meméticos vemos que sigue habiendo una gran diferencia de tiempo, siendo muy superior en los meméticos. Pero a diferencia de los genéticos, estos obtuvieron mejores resultados en el valor del agregado, a pesar de contar también, solo con 15000 evaluaciones de la función objetivo. Por lo tanto si el tiempo no es un problema a tener en cuenta, los meméticos podrían ser una mejor opción que la búsqueda local. En cambio si el tiempo es una restricción importante, habría que valorar si es más conveniente usar BL.

Vemos que los algoritmos que más han tardado han sido los genéticos generacionales. Y si hacemos una comparación entre los cruce vemos que el cruce aritmético ha requerido de más tiempo que el BLX, lo cual puede ser debido a la operación de división que se realiza en el aritmético, al dividir entre 2 la suma de los dos atributos.

### Experimento extra

Como experimento extra he ejecutado los algoritmos genéticos y meméticos con otro operados de cruce. El nuevo operador de cruce esta descrito por el siguiente pseudocódigo:

```
1   Funcion crossLow(cromosoma1, cromosoma2, T)
2     w={}
3
4     Para i = 0 hasta N
5       Si cromosoma1.w_i <= 0.2
6         w.insertar(cromosoma1.w_i)
7       Si No
8         w.insertar(cromosoma2.w_i)
9
```

```
10        Devolver cromosoma(w, F(T,T,w))
11    Fin
```

Es decir el hijo tendrá las características del padre1 que sean menores de 0,2 y el resto del padre2. El objetivo de este cruce es favorecer la tasa de reducción, por este motivo le he llamado cruce Low.

Veamos los resultados obtenidos con este cruce:

| AGG Low | Ozone | | | | Parkinsons | | | | Spectf-heart | | | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|
| | %clas | %red | Agr. | T | %clas | %red | Agr. | T | %clas | %red | Agr. | T |
| P1 | 71.87 | 94.44 | 83.15 | 44.195 | 61.53 | 95.45 | 78.49 | 5.6166 | 66.66 | 95.45 | 81.06 | 22.964 |
| P2 | 64.06 | 91.66 | 77.86 | 46.446 | 79.48 | 90.90 | 85.19 | 6.2455 | 74.07 | 93.18 | 83.62 | 21.143 |
| P3 | 68.75 | 95.83 | 82.29 | 48.496 | 76.92 | 95.45 | 86.18 | 6.7475 | 67.92 | 95.45 | 81.68 | 21.462 |
| P4 | 74.60 | 91.66 | 83.13 | 50.217 | 71.79 | 95.45 | 83.62 | 6.2481 | 75.47 | 95.45 | 85.46 | 21.208 |
| P5 | 71.42 | 94.44 | 82.93 | 48.302 | 66.66 | 95.45 | 81.06 | 5.7329 | 73.58 | 93.18 | 83.38 | 20.490 |
| Media | 70.14 | 93.61 | 81.87 | 47.531 | 71.28 | 94.54 | 82.91 | 6.1181 | 71.54 | 94.54 | 83.04 | 21.453 |

| AGE Low | Ozone | | | | Parkinsons | | | | Spectf-heart | | | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|
| | %clas | %red | Agr. | T | %clas | %red | Agr. | T | %clas | %red | Agr. | T |
| P1 | 79.68 | 93.05 | 86.37 | 46.615 | 89.74 | 81.81 | 85.78 | 6.5844 | 79.62 | 93.18 | 86.40 | 21.368 |
| P2 | 70.31 | 93.05 | 81.68 | 45.716 | 89.74 | 90.90 | 90.32 | 6.3715 | 64.81 | 90.90 | 77.86 | 21.903 |
| P3 | 75 | 93.05 | 84.02 | 43.444 | 92.30 | 90.90 | 91.60 | 6.0386 | 75.47 | 93.18 | 84.32 | 22.285 |
| P4 | 74.60 | 94.44 | 84.52 | 43.553 | 84.61 | 90.90 | 87.76 | 6.0443 | 67.92 | 93.18 | 80.55 | 21.215 |
| P5 | 74.60 | 90.27 | 82.44 | 41.739 | 87.17 | 90.90 | 89.04 | 6.2024 | 67.92 | 86.36 | 77.14 | 22.044 |
| Media | 74.84 | 92.77 | 83.80 | 44.213 | 88.71 | 89.09 | 88.90 | 6.2482 | 71.15 | 91.36 | 81.25 | 21.763 |

| AM(10, 1) Low | Ozone | | | | Parkinsons | | | | Spectf-heart | | | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|
| | %clas | %red | Agr. | T | %clas | %red | Agr. | T | %clas | %red | Agr. | T |
| P1 | 67.18 | 88.88 | 78.03 | 46.325 | 84.61 | 90.90 | 87.76 | 6.7447 | 79.62 | 90.90 | 85.26 | 21.439 |
| P2 | 79.68 | 91.66 | 85.67 | 45.278 | 79.48 | 86.36 | 82.92 | 7.4696 | 77.77 | 93.18 | 85.47 | 22.925 |
| P3 | 71.87 | 91.66 | 81.77 | 43.834 | 94.87 | 86.36 | 90.61 | 6.6470 | 69.81 | 90.90 | 80.36 | 22.202 |
| P4 | 74.60 | 90.27 | 82.44 | 46.076 | 87.17 | 90.90 | 89.04 | 6.5926 | 79.24 | 90.90 | 85.07 | 24.765 |
| P5 | 66.66 | 84.72 | 75.69 | 44.758 | 87.17 | 90.90 | 89.04 | 5.7603 | 67.92 | 90.90 | 79.41 | 23.090 |
| Media | 72.00 | 89.44 | 80.72 | 45.254 | 86.66 | 89.09 | 87.87 | 6.6428 | 74.87 | 91.36 | 83.12 | 22.884 |

Solo lo he ejecutado con el memético AM-(10,1.0). A continuación se muestra una tabla comparativa con los demás genéticos.

| | Ozone | | | | Parkinsons | | | | Spectf-heart | | | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|
| | %clas | %red | Agr. | T | %clas | %red | Agr. | T | %clas | %red | Agr. | T |
| **G BLX** | 75.13 | 73.33 | 74.23 | 50.581 | 92.30 | 80.90 | 86.60 | 6.4056 | 73.00 | 68.18 | 70.59 | 24.387 |
| **G CA** | 81.13 | 48.61 | 64.87 | 55.792 | 93.33 | 62.72 | 78.03 | 8.0484 | 73.02 | 45 | 59.01 | 26.651 |
| **G Low** | 70.14 | 93.61 | 81.87 | 47.531 | 71.28 | 94.54 | 82.91 | 6.1181 | 71.54 | 94.54 | 83.04 | 21.453 |
| **E BLX** | 77.65 | 72.22 | 74.94 | 46.964 | 88.20 | 87.27 | 87.73 | 6.2030 | 70.04 | 73.63 | 71.84 | 23.213 |
| **E CA** | 79.88 | 68.61 | 74.24 | 49.492 | 90.25 | 79.09 | 84.67 | 6.6227 | 72.30 | 62.72 | 67.51 | 24.342 |
| **E Low** | 74.84 | 92.77 | 83.80 | 44.213 | 88.71 | 89.09 | 88.90 | 6.2482 | 71.15 | 91.36 | 81.25 | 21.763 |
| **M BLX** | 75.16 | 82.22 | 78.69 | 47.178 | 90.76 | 88.18 | 89.47 | 6.3378 | 66.24 | 81.81 | 74.03 | 22.832 |
| **M Low** | 72.00 | 89.44 | 80.72 | 45.254 | 86.66 | 89.09 | 87.87 | 6.6428 | 74.87 | 91.36 | 83.12 | 22.884 |

Se puede ver claramente que este nuevo cruce ha cumplido con su objetivo provocando una mejora significativa en la tasa de reducción. El inconveniente es que la tasa de acierto parece haberse visto algo perjudicada. A pesar de ello, la gran mejora en tasaRed, ha provocado una mejora significativa en el agregado, superando los records obtenidos con los otros algoritmos.

Podemos observar que los tiempos con el cruce Low son levemente mejores que el de los otros, debido a la simpleza de este cruce, el cual no hace ninguna operación aritmética.

# 6. Referencias

Principalmente los seminarios y los temas de teoría de la asignatura. Además del material para las bases de datos e implementación en c++ de número aleatorios disponibles en la web de la asignatura.