# Práctica 2: Técnicas de Búsqueda basadas en Poblaciones

## Problema de Aprendizaje de Características por Peso

**Autor**: Eloy Bedia García **DNI**: 45119918J

**Grupo**: A2 / Martes, 17:30 – 19:30

**E-Mail**: eloybg97@correo.ugr.es

**Algoritmos Utilizados:**

AGG-BLX AGG-CA AGE-BLX AGE-CA

AMG_BLX_10_01 AMG_BLX_10_01mej AMG_BLX_10_1

# Sumario

# Descripción del Problema

A lo largo de este documento nos enfrentaremos a un problema de *machine learning*, en concreto, resolveremos un problema de clasificación binaria aplicando técnicas de búsqueda basadas en poblaciones.

En un problema de clasificación binaria, se parte de un conjunto de datos en los que cada uno de los elementos tiene un conjunto de características (X) y una etiqueta (Y). Está etiqueta será la encargada de marcar que elementos pertenecen a una clase u otra.

El objetivo de un problema de clasificación, es ser capaz de encontrar una función *f*, que sea capaz de predecir con el mínimo error posible (entiéndase error, como fallo en la clasificación) la etiqueta (Y) asociada al conjunto de características (X).

$$f : X \rightarrow Y$$

Normalmente en los problemas de *machine learning*, se tiene acceso a dos conjuntos de datos. El conjunto de *training*, a partir del cual aproximaremos la función *f*. Y el conjunto de test, este es un conjunto de datos con el que probaremos cómo de bien clasifica nuestra función *f*.

En este caso tan solo tenemos acceso al conjunto de *training*, por lo que aplicaremos la técnica de 5-*fold cross validation* para dividir el conjunto en dos conjuntos diferentes, uno de ellos lo usaremos como *training* y el otro lo usaremos como *test*.

Esta técnica consiste en dividir el conjunto de *training* en 5 partes del 20% (cada una de ellas mantendrá la proporción de clases del conjunto original). A continuación, uniremos 4 de esas partes para formar un conjunto del 80% y otro conjunto del 20%, el primero actuará como *training* y el otro como *test*. Aplicaremos esto hasta que existen 5 conjuntos *test* diferentes y por consiguiente, 5 conjuntos *training* también diferentes.

En este momento, ha quedado explicado en qué consiste el problema que se va a resolver. Lo siguiente será presentar los 3 conjuntos de datos con los que se resolverá el problema.

1. Parkinsons: Conjunto orientado a distinguir entre la presencia y la ausencia de la enfermedad de Parkinsons en una serie de pacientes a partir de unas medidas biomédicas de la voz. Consta de 195 ejemplos con 23 características.
2. Ozone: Conjunto de datos para la detección del nivel de ozono según las medidas realizadas a lo largo del tiempo. Consta de 320 ejemplos (seleccionados de los 2536 originales, eliminando los ejemplos con valores perdidos y manteniendo la distribución de clases al 50%) con 73 características.
3. Spectf-heart: Conjunto de datos de detección de enfermedades cardíacas a partir de imágenes médicas de tomografía computarizada (SPECT) del corazón de pacientes. Consta de 267 ejemplos con 44 características.

Los conjuntos de datos se han sometido previamente a un proceso de normalización para evitar que una característica tenga más importancia por el mero hecho de tener un rango de valores más alto.

# Algoritmos Genéticos

Como ya se ha mencionado en el apartado anterior, para resolver el problema se aplicaran técnicas de búsqueda basadas en poblaciones. En resumidas cuentas, esta técnica se basa en la teoría de la evolución de las especies.

Utilizaremos dos tipos de algoritmos: los genéticos y los meméticos. Estos últimos difieren en los genéticos en una única cosa que explicaremos más adelante.

Tenemos una población de posibles soluciones, cada una de ellas se reproducirá (o no) y mutará (o no), de forma que con el paso del tiempo obtendremos una población adaptada al problema para el que se diseñó. El individuo de la población que mejor se adapte será la mejor solución del problema, que en nuestro caso será el vector ω que mejor clasifica.

El vector ω tendrá unas propiedades especificas tales como:

- $X, \omega \in \mathbb{R}^N$ (X y ω tienen la misma dimensión)
- $\forall i < N, 0 \leq \omega_i \leq 1$

(Este ω colaborará en la elaboración de *f*, en breve se explicará cómo).

Se ha dicho anteriormente que la población se adapta al problema, ¿Pero cómo es capaz la población de adaptarse? La respuesta está en la función objetivo.

La función objetivo es una función que asignará a cada individuo de la población una valoración, de forma que la población evolucionará en base a aquellos individuos con mayor puntuación. Haciendo una analogía con la Teoría de Darwin, la selección natural actuará en beneficio de aquellos individuos con mayor valoración.

**Algorithm 1** fObjetivo

```
1: procedure FOBJETIVO(W, training)
2:     tasa_clas = tasaClas(W, training)
3:     tasa_red = tasaRed(W, 0'2)
4:     return (tasa_clas,tasa_red)
```

Esta puntuación es un número real entre 0 y 100. ¿Y cómo se calcula esa puntuación? En primer lugar, el 50% de esa puntuación es el número de aciertos a la hora de clasificar y en segundo lugar, el otro 50% es el número de valores del vector que son menores que 0’2. Donde *tasa_clas* es el número de aciertos de ω y *tasa_red* es el número de componentes del vector ω con valor inferior a 0’2.

**Algorithm 1** tasaClas

```
    procedure TASACLAS(W, training)
2:      bien_training = validacionTraining(W, training)
        return ( bien_training / training.getSize() * 100 )
```

**Algorithm 1** tasaRed

```
procedure TASARED(W, tasa)
    j = 0
    for i:=0 to W.size() do
        if W_i < tasa then
            j = j + 1
    return (j / W.size() * 100)
```

A continuación se insertará el pseudocódigo de las funciones que colaboran en el calculo de *tasa_clasificación* y *tasa_reducción*

**Algorithm 1** validacionTraining

```
procedure VALIDACIONTRAINING(W, training)
    bien_clasificados = 0
    for i=0 to training.size() do
        leave = training ∩ {training_i}
        if clasifica(W, leave, training_i) = label(training_i) then
            bien_clasificados = bien_clasificados + 1
    return bien_clasificados
```

**Algorithm 1** clasifica

```
procedure CLASIFICA(W, conocimiento, I)
    min = training_0
    dist_min = ||W(min − I)||_2
    for i = 1 to conocimiento.size() do
        if dist_min > ||W(training_i − I)||_2 then
            dist_min = ||W(training_i − I)||_2
            min = training_i
    return label(min)
```

Antes se ha mencionado que ω colaborará en la elaboración de *f* pero todavía no se ha explicado cómo lo hará, este es el momento de hacerlo.

$f: X, \omega \rightarrow Y$ Se define como: $\alpha \in Training | \nexists \beta \in Training \|\omega \cdot (\alpha - X)\|_2 > \|\omega \cdot (\beta - X)\|_2$

Es decir, *f* clasifica un elemento X con la misma etiqueta (Y) que un elemento α del t*raining*, de forma que la distancia euclidea entre X y α es mínima. ( ω se encarga de ponderar la importancia de cada una de las características).

Ya tenemos definidos los elementos básicos de toda metaheurística, la función objetivo, las entradas y la salida. Pero hasta ahora se ha tratado a la técnica como una caja negra. En este instante se comenzará a explicar los elementos propios de la técnica.

**Algorithm 1** GeneticoGenerico

```
    procedure GeneticoGenerico
        t = 0
3:      InicializarPoblacion(P(t))
        EvaluarPoblacion(P(t))
        repeat
6:          t = t + 1
            padres = SeleccionarPadres(P(t))
            hijos = Cruzar(Padres)
9:          Mutar(hijos)
            Remplazar(P(t), P(t-1), hijos)
            EvaluarPoblacion(P(t))
12:     until condicion de parada
```

En la figura podemos observar seis procedimientos que llaman la atención: IniciarPoblación, EvaluarPoblación, SeleccionarPadres, Cruzar, Mutar y Reemplazar. A continuación, se detallará que papel juega cada uno de estos procedimientos en el algoritmo que nos atañe.

IniciarPoblación, en el algoritmo implementado en esta práctica, también incorpora la funcionalidad de EvaluarPoblación. Cuando hablemos de evaluar la población, de ahora en adelante, lo entenderemos como aplicar la Función Objetivo previamente descrita a cada uno de los individuos de la población.

Esta función genera, en este caso particular, treinta $\omega$ aleatoriamente. Los valores de cada una de las componentes del vector, se toman de una distribución uniforme (recuérdese que $\omega$ siempre toma valores entre 0 y 1)

$$\forall i<N, \omega_i \in U(0,1)$$

Una vez generados todos los $\omega$, se evalúan cada uno de ellos.

**Algorithm 1** StartPopulation

```
    procedure StartPopulation(training, M, seed)
        population = ∅
3:      InicializarSecuanciaPseudoAleatoria(seed)
        for i=0 to M do
            for j=0 to dim(training_0) do
6:              aux = aux ∪ U(0, 1)                    ▷ aux ∈ R
            evaluacion = fObjetivo(aux, training)
            population = pupulation ∪ (aux, evaluacion)
9:          aux = ∅
        return population
```

Después de inicializar la población aleatoriamente, procederemos a seleccionar los padres que se encargaran de reproducirse para dar lugar a la nueva población, esto se hace con SeleccionarPadres. Este procedimiento utiliza la técnica de torneo binario para llevar a cabo su tarea.

La técnica de torneo binario consiste en elegir a dos individuos al azar de la población y quedarse con el mejor de ellos, es decir, el que mejor puntuación tenga. Esta técnica se aplicará tantas veces como individuos se quieran seleccionar.

**Algorithm 1** Seleccionar

```
    procedure SELECCIONAR(P, N, seed)
        seleccionados = ∅
3:      for i=0 to N do
            select = torneoBinario(P,seed)
            seleccionados = seleccionados ∪ select
        return seleccionados
```

**Algorithm 1** torneoBinario

```
    procedure TORNEOBINARIO(P, seed)
        population = ∅
3:      InicializarSecuanciaPseudoAleatoria(seed)
        oponente1 = U(0,P.size() - 1)
        repeat
6:          oponente2 = U(0,P.size() - 1)
        until oponente1 ≠ oponente2
        coste_1 = P_oponente1.coste
9:      coste_2 = P_oponente2.coste
        if coste_1 < coste_2 then
            mejor_oponente = oponente2
12:     else
            mejor_oponente = oponente1
        return mejor_oponente
```

El procedimiento Cruzar, genera un número de hijos $\omega_{HN}$ fruto de una secuencia de operaciones con los padres $\omega_{P1}$ y $\omega_{P2}$. Para resolver este problema hemos probado dos cruces diferentes. El hecho de usar un cruce excluye del hecho de usar el otro.

El primero es el cruce BLX-α. Este cruce genera dos hijos $\omega_{H1}$ y $\omega_{H2}$. Ambos hijos son generados de la misma forma.

$$\omega_{P1}=(c_{11},c_{12},\dots,c_{1n}) \quad \omega_{P2}=(c_{21},c_{22},\dots,c_{2n}) \quad \omega_{Hk}=(x_1,x_2,\dots,x_n) \quad k=1,2$$

Donde el valor de $x_i$ se toma de una distribución uniforme en el intervalo descrito a continuación:

$$[C_{min}-I\alpha, C_{max}+I\alpha] \quad \alpha\in[0,1]$$

$$C_{min}=min\{c_{1i},c_{2i}\} \quad C_{max}=max\{c_{1i},c_{2i}\} \quad I=C_{max}-C_{min}$$

$$\forall x\in\omega_{Hk}, x\in U(C_{min}-I\alpha, C_{max}+I\alpha)$$

Para nosotros, α tendrá un valor de 0’3.

**Algorithm 1** CruceBLX

```
procedure CRUCEBLX(C_1, C_2, training, seed)
    ALPHA = 0'3
    InicializarSecuanciaPseudoAleatoria(seed)
    for i=0 to C_1.caracteristicas.size() do
        minimo = min C_1.caracteristicas_i , C_2.caracteristicas_i
        maximo = max C_1.caracteristicas_i , C_2.caracteristicas_i
        I = max - min
        hijo_0.caracteristicas_i = U(min - I * ALPHA, max + I * ALPHA)
        hijo_1.caracteristicas_i = U(min - I * ALPHA, max + I * ALPHA)
    hijo_0.coste = fObjetivo(hijo_0, training)
    hijo_1.coste = fObjetivo(hijo_1, training)
    return (hijo_0, hijo_1)
```

El segundo cruce que usamos es el Cruce Mean. Este cruce es mas sencillo de calcular y a diferencia que el anterior, solo genera un único hijo. Este hijo es fruto de hacer la media componente a componente de las características de ambos padres.

$$\omega_H = \frac{1}{2}(\omega_{P1} + \omega_{P2})$$

**Algorithm 1** CruceMean

```
procedure CRUCEMEAN(C_1, C_2, training)
    InicializarSecuanciaPseudoAleatoria(seed)
    for i=0 to C_1.caracteristicas.size() do
        hijo_0.caracteristicas_i = (C_1.caracteristicas_i + C_2.caracteristicas_i) / 2
    hijo_0.coste = fObjetivo(hijo_0, training)
    return hijo_0
```

Usar cruces ayuda a converger a una única solución, es decir, exploramos el espacio de búsqueda guiado por la población. Volviendo a la Teoría de Darwin, los individuos que porten características que le ayuden a adaptarse mejor, a procrear, sus hijos también las tendrán y si no las tienen la selección natural hará lo propio.

¿Pero que pasa si hay una característica que ayudaría a la adaptación pero ningún individuo es portador de ella? O dicho de otra forma, ¿Y si la solución no esta en el espacio de búsqueda en el que nos encontramos?. La solución a esto esta en el siguiente procedimiento, Mutar.

Mutar se encarga de introducir diversidad en el espacio de búsqueda. Esto se hace alterando al menos una componente del hijo fruto del anterior procedimiento. En nuestro caso, a cada característica (gen) $\omega_i$ le suma un valor tomado de una distribución normal.

$$\forall i < N, \omega_i \in N(0, 0'3)$$

Una vez llegados aquí, ya tenemos lista una población fruto de una secuencia de cruces y mutaciones.  Ahora hay que ver qué es lo que se hace con ella. De esta misión se encarga el último procedimiento, Remplazar.

Este también es algo mas complejo ya que depende del modelo de evolución elegido, por ello los explicaremos en el siguiente aparatado. Pero básicamente se encarga de decidir cuál de esos hijos pasará a formar parte de la población actual y que individuos de la población actual pasarán a no formar parte de ella .

# Modelo de Evolución

El modelo de evolución como su propio nombre indica, es la manera en la que la población irá evolucionando con cada iteración del algoritmo. Existen dos modelos de evolución:

- El modelo generacional.
- El modelo estacionario.

Los algoritmos descritos en el siguiente apartado, hacen uso de ambos modelos.

## El Modelo Generacional

En este modelo, la población anterior remplaza íntegramente por la nueva.

**Algorithm 1** RemplazoGeneracional

```
procedure REMPLAZOGENERACIONAL(P, R)
    OrdenarDecreciente(P)
    OrdenarDecreciente(R)
    if P_0.coste > R_n.coste then
        R_n = P_0
    P = R
```

Además, no siempre se permite que se de un cruce o una mutación. Generalmente se suele dar una probabilidad de cruce $P_c$ y una probabilidad de mutación $P_m$. Pero ¿Que pasa cuando no se produce un cruce? En ese caso, para suplir todos los cruces que no se han llevado a cabo, se introducen individuos seleccionados hasta completar el tamaño de la población.

En este modelo, se aplica la técnica conocida como elitismo. Esta consiste en mantener en la población actual la mejor solución de la población anterior, esto nos ayuda a converger a una mejor solución.

## El Modelo Estacionario

En el modelo estacionario, tan solo dos padres son seleccionados. Estos dos padres siempre va a cruzarse, es decir, $P_c = 1$. La mutación si sigue manteniendo una probabilidad $P_m$.

A la hora de remplazar la población, el/los hijos, compiten con los 2 peores individuos de la población para poder acceder a ella. Realmente no existe un remplazo de la población completa.

**Algorithm 1** RemplazoEstacionario

```
procedure REMPLAZOESTACIONARIO(P, R)
    OrdenarCreciente(P)
    for i=0 to R.size() do
        if P_i.coste < R_i.coste then
            R_i = P_i
```

# Algoritmos Meméticos

Los algoritmos meméticos, a diferencia de los algoritmos genéticos están basados en la Teoría de Lamark. Esta teoría defiende que los descendientes no solo heredan los genes de sus progenitores si no que también heredan sus experiencias.

Esto lo podemos entender en el problema que nos atañe como llevar a un determinado ω hacia su optimo local más cercano. Entenderemos optimo local como la máxima evaluación alcanzable partiendo desde ω (aunque esto no es del todo cierto). Esto lo hacemos aplicando una Búsqueda Local sobre ω.

La búsqueda local que aplicamos en este caso es una búsqueda “Primero Mejor”, por eso dijimos antes que no siempre alcanzaremos el optimo local mas cercano. Esta búsqueda consiste en explorar el espacio de búsqueda de una determinada solución hasta encontrar la primera solución en ese espacio de búsqueda que mejore la evaluación de la solución anterior. Esto se hace hasta que la puntuación deje de aumentar.

**Algorithm 1** BusquedaLocalAprendizaje

```
    procedure BusquedaLocalAprendizaje(W, training, seed)
        k = 0
3:      nCaracteristicas = training_0.size()
        InicializarSecuanciaPseudoAleatoria(seed)
        repeat
6:          i = 0
            costeSolucion = fObjetivo(W, training)
            repeat
9:              aux = generarVecino(W, imodnCaracteristicas, seed)
                costeAux = fObjetivo(aux, training)
                i = i + 1
12:         until costeAux > costeSolucion ∨ i < 2 * nCaracteristicas
            if costeAux > costeSolucion then
                W = aux
15:             costeSolucion = costeAux
            nEjecuciones = nEjecuciones + 1
            k = k + i
18:     until (costeAux ≤ costeSolucion) ∧ (k < 2 * nCaracteristicas)
        return (W, costeSolucion)
```

Este proceso de “adquisición de experiencia”, se introduce cada vez que obtenemos una nueva población (también llamada generación).

**Algorithm 1** MemeticoGenerico

```
procedure MEMETICOGENERICO
    t = 0
    InicializarPoblacion(P(t))
    EvaluarPoblacion(P(t))
    repeat
        t = t + 1
        AdquiereExperiencia(P(t))
        padres = SeleccionarPadres(P(t))
        hijos = Cruzar(Padres)
        Mutar(hijos)
        Remplazar(P(t), P(t-1), hijos)
        EvaluarPoblacion(P(t))
    until condicion de parada
```

# Algoritmos Aplicados

En este apartado se va a proceder a explicar cada uno de los algoritmos con los que se ha experimentado, pero antes explicaremos la notación de los algoritmos, ya que no es algo trivial.

- Los algoritmos genéticos comienzan por la terminación AG, mientras que los algoritmos meméticos comienzan por AM.
- La letra que sigue determina el modelo de evolución que sigue el algoritmo. Una G denota el esquema generacional y la letra E, el estacionario.
- A continuación, se especifica el tipo de cruce con el que se va a proceder. BLX denota cruce BLX-α y CA, denota cruce mean.
- En el caso de los algoritmos meméticos, ha esto anterior le siguen 2 números:
    - El primero, indica cada cuantas generaciones se aplica el proceso de “adquisición de experiencia”
    - El segundo indica a que porcentaje de población se le aplica.
- Además si el último de estos números va acompañado del sufijo “mej”, indica que se aplica sobre los mejores individuos de la población.

Ya que todos los elementos de los algoritmos que se detallara a continuación ya están explicados, nos limitaremos a especificar la probabilidad de cruce y mutación que se utiliza en cada uno de ellos.

En los algoritmos genéticos el tamaño de la población es 30, mientras que en los meméticos es 10.

## Especificación de Algoritmos Genéticos

### AGG_BLX

En este algoritmo la probabilidad de cruce es 0’7, mientras que la probabilidad de mutación es de 0’01 por gen.

**Algorithm 1** AGG_BLX

```
procedure AGG_BLX(training, seed)
    nEjecuciones = 0
    M = 30
    population = StartPopulation(training, M, seed)
    nEjecuciones = nEjecuciones + M
    repeat
        idxPadres = Seleccionar(population, M, seed)
        for i = 0 to P_c * M do
            hijos = CruceBLX(population_{idxPadres[i]}, population_{idxPadres[i+1]}, training, seed)
            nEjecuciones = nEjecuciones + 2
            replazo_i = hijos_0
            replazo_{i+1} = hijos_1
        for i = P_c * M to M do replazo_i = population_{idxPadres[i]}
        for i = 0 to M do
            muta = Mutar(replazo_i,training, seed)
            if muta then
                nEjecuciones = nEjecuciones + 1
        RemplazoGeneracional(population,remplazo)
    until nEjecuciones > 15000
    OrdenarDecreciente(population)
    return population_0
```

## AGG_CA

En este algoritmo la probabilidad de cruce es 0’7, mientras que la probabilidad de mutación es de 0’01 por gen.

**Algorithm 1** AGG_CA

```
    procedure AGG_CA(training, seed)
        nEjecuciones = 0
 3:     M = 30
        population = StartPopulation(training, M, seed)
        nEjecuciones = nEjecuciones + M
 6:     repeat
            idxPadres = Seleccionar(population, M, seed)
            for i = 0 to P_c * M do
 9:             hijo = CruceMean(population_idxPadres[i], population_idxPadres[i+1], training)
                nEjecuciones = nEjecuciones + 1
                remplazo = remplazo ∪ hijo
12:         for i = P_c * M to M do remplazo = remplazo ∪ population_idxPadres[i]
            for i = 0 to M do
                muta = Mutar(remplazo_i, training, seed)
15:             if muta then
                    nEjecuciones = nEjecuciones + 1
            RemplazoGeneracional(population,remplazo)
18:     until nEjecuciones > 15000
        OrdenarDecreciente(population)
        return population_0
```

## AGE_BLX

En este algoritmo la probabilidad de cruce es 1, mientras que la probabilidad de mutación es de 0’01 por gen.

**Algorithm 1** AGE_BLX

```
    procedure AGE_BLX(training, seed)
        nEjecuciones = 0
 3:     M = 30
        population = StartPopulation(training, M, seed)
        nEjecuciones = nEjecuciones + M
 6:     repeat
            idxPadres = Seleccionar(population, 2, seed)
            hijos = CruceBLX(population_idxPadres[0], population_idxPadres[1], training, seed)
 9:         nEjecuciones = nEjecuciones + 2
            remplazo = remplazo ∪ {hijos_0, hijos_1}
            for i = 0 to M do
12:             muta = Mutar(remplazo_i,training, seed)
                if muta then
                    nEjecuciones = nEjecuciones + 1
15:         RemplazoEstacionario(population,remplazo)
        until nEjecuciones > 15000
        OrdenarDecreciente(population)
         return population_0
```

## AGE_CA

En este algoritmo la probabilidad de cruce es 1, mientras que la probabilidad de mutación es de 0’01 por gen.

**Algorithm 1** AGE_CA

```
    procedure AGE_BLX(training, seed)
        nEjecuciones = 0
 3:     M = 30
        population = StartPopulation(training, M, seed)
        nEjecuciones = nEjecuciones + M
 6:     repeat
            idxPadres = Seleccionar(population, 2, seed)
            hijo = Mean(population_idxPadres[0], population_idxPadres[1], training, seed)
 9:         nEjecuciones = nEjecuciones + 2
            remplazo = remplazo ∪ {hijo}
            for i = 0 to M do
12:             muta = Mutar(remplazo_i,training, seed)
                if muta then
                    nEjecuciones = nEjecuciones + 1
15:         RemplazoEstacionario(population,remplazo)
        until nEjecuciones > 15000
        OrdenarDecreciente(population)
         return population_0
```

## Especificación de  Algoritmos Meméticos

### AMG_BLX_10_01

En este algoritmo la probabilidad de cruce es 0’7, mientras que la probabilidad de mutación es de 0’7 por cromosoma y 0’01 por gen.

**Algorithm 1** AMG_BLX_10_01

```
procedure AMG_BLX_10_01(training, seed)
    nEjecuciones = 0
    M = 10
    nGeneration = 0
    population = StartPopulation(training, M, seed)
    nEjecuciones = nEjecuciones + M
    repeat
        if nGeneration mod 10 = 0 then
            aux2 = ElegirAleatorio(population, M * 0'1, seed)
            for i = 0 to M * 0'1 do
                aux = BusquedaLocalAprendizaje(population_{aux2[i]}.caracteristicas,
training, seed, nEjecucionesAux)
                population_{aux2[i]} = aux
                nEjecuciones = nEjecuciones + nEjecucionesAux
        idxPadres = Seleccionar(population, M, seed)
        for i = 0 to P_c * M do
            hijos = CruceBLX(population_{idxPadres[i]}, population_{idxPadres[i+1]}, training, seed)
            nEjecuciones = nEjecuciones + 2
            remplazo_i = hijos_0
            remplazo_{i+1} = hijos_1
        for i = P_c * M to M do remplazo_i = population_{idxPadres[i]}
        for i = 0 to M do
            if U(0,1) ¡ 0'7 then
                muta = Mutar(remplazo_i,training, seed)
                if muta then
                    nEjecuciones = nEjecuciones + 1
        RemplazoGeneracional(population,remplazo)
        nGeneration = nGeneration + 1
    until nEjecuciones > 15000
    OrdenarDecreciente(population)
     return population_0
```

## AMG_BLX_10_01mej

En este algoritmo la probabilidad de cruce es 0’7, mientras que la probabilidad de mutación es de 0’7 por cromosoma y 0’01 por gen.

**Algorithm 1** AMG_BLX_10_01mej

```
    procedure AMG_BLX_10_01MEJ(training, seed)
        nEjecuciones = 0
 3:     M = 10
        nGeneration = 0
        population = StartPopulation(training, M, seed)
 6:     nEjecuciones = nEjecuciones + M
        repeat
            if nGeneration mod 10 = 0 then
 9:             OrdenarDecreciente(population)
                for i = 0 to M * 0'1 do
                    aux = BusquedaLocalAprendizaje(population_i.caracteristicas,
    training, seed, nEjecucionesAux)
12:                 population_aux2[i] = aux
                    nEjecuciones = nEjecuciones + nEjecucionesAux
            idxPadres = Seleccionar(population, M, seed)
15:         for i = 0 to P_c * M do
                hijos = CruceBLX(population_idxPadres[i], population_idxPadres[i+1], training, seed)
                nEjecuciones = nEjecuciones + 2
18:             remplazo_i = hijos_0
                remplazo_i+1 = hijos_1
            for i = P_c * M to M do remplazo_i = population_idxPadres[i]
21:         for i = 0 to M do
                if U(0,1) ¡ 0'7 then
                    muta = Mutar(remplazo_i,training, seed)
24:                 if muta then
                        nEjecuciones = nEjecuciones + 1
            RemplazoGeneracional(population,remplazo)
27:         nGeneration = nGeneration + 1
        until nEjecuciones > 15000
        OrdenarDecreciente(population)
         return population_0
```

## AMG_BLX_10_1

En este algoritmo la probabilidad de cruce es 0’7, mientras que la probabilidad de mutación es de 0’7 por cromosoma y 0’01 por gen.

**Algorithm 1** AMG_BLX_10_1

```
    procedure AMG_BLX_10_1(training, seed)
        nEjecuciones = 0
 3:     M = 10
        nGeneration = 0
        population = StartPopulation(training, M, seed)
 6:     nEjecuciones = nEjecuciones + M
        repeat
            if nGeneration mod 10 = 0 then
 9:             aux2 = ElegirAleatorio(population, M * 0'1, seed)
                for i = 0 to M do
                    aux = BusquedaLocalAprendizaje(population_{aux2[i]}.caracteristicas,
    training, seed, nEjecucionesAux)
12:                 population_{aux2[i]} = aux
                    nEjecuciones = nEjecuciones + nEjecucionesAux
            idxPadres = Seleccionar(population, M, seed)
15:         for i = 0 to P_c * M do
                hijos = CruceBLX(population_{idxPadres[i]}, population_{idxPadres[i+1]}, training, seed)
                nEjecuciones = nEjecuciones + 2
18:             remplazo_i = hijos_0
                remplazo_{i+1} = hijos_1
            for i = P_c * M to M do remplazo_i = population_{idxPadres[i]}
21:         for i = 0 to M do
                if U(0,1) ¡ 0'7 then
                    muta = Mutar(remplazo_i,training, seed)
24:                 if muta then
                        nEjecuciones = nEjecuciones + 1
            RemplazoGeneracional(population,remplazo)
27:         nGeneration = nGeneration + 1
        until nEjecuciones > 15000
        OrdenarDecreciente(population)
         return population_0
```

# Procedimiento de Uso

Para replicar los experimentos que se presentarán en el próximo punto de esta memoria, debe seguir los siguientes pasos:

1. Situarse en el directorio Práctica2
2. Ejecutar el *makefile* proporcionado
3. Ejecutar: ‘./BIN/<nombre_algoritmo> DAT/<nombre_fichero> <seed_utilizado>’

También se proporciona un *script* que automatiza todo el proceso de experimentación, ya que algunos algoritmos pueden llegar a tardar varias horas en terminar su ejecución. Para utilizarlo debe ejecutar la siguiente orden:

1. sh experimentar.sh

# Análisis y Conclusiones

En este apartado final analizaremos las prestaciones que nos ofrecen las técnicas de búsqueda basadas en poblaciones frente a las técnicas usadas en la práctica anterior.

Cada una de las tablas está dividida en 3 columnas, cada una pertenece a un conjunto de datos; y tiene 6 filas, 5 son fruto de las 5 particiones obtenidas al aplicar la técnica 5-fold cross validation, y la última es la media aritmética de las filas anteriores.

Para cada fichero, se obtiene de izquierda a derecha, la tasa de clasificación del conjunto *training*, la tasa de clasificación en el conjunto *test*, la tasa de reducción, la media de las dos tasas anteriores y el tiempo invertido en el algoritmo.

Esta tabla pertenece al algoritimo RELIEF, este es un algoritmo voraz que ajusta ω teniendo en cuenta la distancia euclidea del individuo que tiene mas lejos y el más cercano.

Tabla 5.2: Resultados obtenidos por el algoritmo RELIEF en el problema del APC

| Seed: 123564891 | Ozone | | | | | Parkinsons | | | | | Spectf-heart | | | | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| | *%_tra* | *%_clas* | *%red* | *Agr.* | T | *%_tra* | *%_clas* | *%red* | *Agr.* | T | *%_tra* | *%_clas* | *%red* | *Agr.* | T |
| **Partición 1** | 83,20 | 78,16 | 0,00 | 39,08 | 0,08 | 91,08 | 92,10 | 4,55 | 48,33 | 0,02 | 75,36 | 91,18 | 27,27 | 59,23 | 0,09 |
| **Partición 2** | 77,34 | 78,13 | 2,78 | 40,46 | 0,08 | 91,08 | 76,32 | 4,55 | 40,44 | 0,02 | 74,64 | 88,24 | 29,55 | 58,90 | 0,08 |
| **Partición 3** | 80,08 | 73,44 | 5,56 | 39,50 | 0,08 | 91,08 | 89,47 | 4,55 | 47,01 | 0,02 | 75,36 | 75,00 | 27,27 | 51,14 | 0,08 |
| **Partición 4** | 79,30 | 82,81 | 2,78 | 42,80 | 0,08 | 92,99 | 76,32 | 4,55 | 40,44 | 0,02 | 74,29 | 73,53 | 27,27 | 50,40 | 0,08 |
| **Partición 5** | 80,47 | 82,81 | 1,39 | 42,10 | 0,08 | 89,81 | 63,16 | 0,00 | 31,58 | 0,02 | 80,71 | 75,00 | 25,00 | 50,00 | 0,08 |
| **Media** | 80,08 | 79,07 | 2,50 | 40,79 | 0,08 | 91,21 | 79,47 | 3,64 | 41,56 | 0,02 | 76,07 | 80,59 | 27,27 | 53,93 | 0,08 |

La siguiente tabla pertenece al al algoritmo de búsqueda local, este algoritmo esta explicado en el apartado de Algoritmos Meméticos.

Tabla 5.1: Resultados obtenidos por el algoritmo busqueda local en el problema del APC

| Seed: 123564891 | Ozone | | | | | Parkinsons | | | | | Spectf-heart | | | | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| | *%_tra* | *%_clas* | *%red* | *Agr.* | T | *%_tra* | *%_clas* | *%red* | *Agr.* | T | *%_tra* | *%_clas* | *%red* | *Agr.* | T |
| **Partición 1** | 86,72 | 73,44 | 19,44 | 46,44 | 46,39 | 90,45 | 81,57 | 18,18 | 49,88 | 3,46 | 74,29 | 95,58 | 20,45 | 58,02 | 32,02 |
| **Partición 2** | 81,64 | 84,38 | 12,50 | 48,44 | 47,38 | 89,81 | 84,21 | 22,72 | 53,47 | 3,65 | 78,93 | 76,47 | 13,64 | 45,06 | 32,13 |
| **Partición 3** | 82,03 | 78,13 | 20,83 | 54,93 | 48,92 | 92,35 | 100,00 | 18,18 | 59,09 | 3,44 | 80,36 | 72,06 | 11,36 | 41,71 | 30,27 |
| **Partición 4** | 82,03 | 81,25 | 18,06 | 54,82 | 67,29 | 87,26 | 60,53 | 9,09 | 34,81 | 3,58 | 72,50 | 79,41 | 18,18 | 48,80 | 28,23 |
| **Partición 5** | 80,07 | 78,13 | 31,94 | 54,74 | 84,65 | 91,72 | 52,63 | 13,64 | 33,14 | 3,83 | 73,21 | 73,53 | 25,00 | 49,27 | 28,33 |
| **Media** | 82,50 | 79,07 | 20,55 | 51,87 | 58,93 | 90,32 | 75,79 | 16,36 | 46,08 | 3,59 | 75,86 | 79,41 | 17,73 | 48,57 | 30,20 |

El último algoritmo usado en la práctica anterior es el Clasificador-1NN, este algoritmo clasifica directamente con un ω cuyas componentes está todas a 0. Más tarde analizaremos las consecuencias de tener una componente de ω con valor 0.

Tabla 5.3: Resultados obtenidos por el algoritmo Clasificador1NN en el problema del APC

| Seed: 123564891 | Ozone | | | | | Parkinsons | | | | | Spectf-heart | | | | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| | *%_tra* | *%_clas* | *%red* | *Agr.* | T | *%_tra* | *%_clas* | *%red* | *Agr.* | T | *%_tra* | *%_clas* | *%red* | *Agr.* | T |
| **Partición 1** | 85,56 | 40,63 | 0,00 | 20,32 | 0,04 | 89,26 | 45,13 | 0,00 | 22,57 | 0,01 | 80,61 | 38,65 | 0,00 | 19,33 | 0,04 |
| **Partición 2** | 82,75 | 39,53 | 0,00 | 19,77 | 0,04 | 89,77 | 43,85 | 0,00 | 21,93 | 0,01 | 80,90 | 35,63 | 0,00 | 17,82 | 0,04 |
| **Partición 3** | 80,56 | 40,94 | 0,00 | 20,47 | 0,04 | 88,23 | 45,64 | 0,00 | 22,82 | 0,01 | 80,32 | 36,49 | 0,00 | 18,25 | 0,04 |
| **Partición 4** | 82,44 | 41,25 | 0,00 | 20,63 | 0,04 | 88,23 | 44,62 | 0,00 | 22,31 | 0,01 | 78,02 | 35,92 | 0,00 | 17,96 | 0,04 |
| **Partición 5** | 83,06 | 40,78 | 0,00 | 20,39 | 0,04 | 90,28 | 44,36 | 0,00 | 22,18 | 0,01 | 81,76 | 36,35 | 0,00 | 18,18 | 0,04 |
| **Media** | 82,87 | 40,63 | 0,00 | 20,31 | 0,04 | 89,15 | 44,72 | 0,00 | 22,36 | 0,01 | 80,32 | 36,61 | 0,00 | 18,30 | 0,04 |

A partir de ahora se mostrarán los resultados obtenidos de los algoritmos presentados en este documento. Mostraremos los resultados en el orden en los que hemos presentado los algoritmos en el documento. Primero se mostraran los algoritmos genéticos y finalmente los meméticos.

Los

Tabla 5.4: Resultados obtenidos por el algoritmo AGG_BLX en el problema del APC

| Seed: 123564891 | Ozone | | | | | Parkinsons | | | | | Spectf-heart | | | | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| | *%_tra* | *%_clas* | *%red* | *Agr.* | T | *%_tra* | *%_clas* | *%red* | *Agr.* | T | *%_tra* | *%_clas* | *%red* | *Agr.* | T |
| **Partición 1** | 86,33 | 84,38 | 25,00 | 54,69 | 465,03 | 85,35 | 86,84 | 40,91 | 63,88 | 84,11 | 85,71 | 97,06 | 36,36 | 66,71 | 455,23 |
| **Partición 2** | 76,17 | 76,17 | 29,17 | 52,67 | 466,23 | 86,44 | 84,21 | 36,36 | 60,29 | 86,65 | 81,07 | 85,29 | 43,18 | 64,24 | 453,98 |
| **Partición 3** | 77,34 | 71,88 | 30,56 | 54,93 | 466.81 | 86,62 | 100,00 | 31,82 | 65,91 | 84,13 | 84,29 | 76,47 | 34,09 | 55,28 | 453,59 |
| **Partición 4** | 81,64 | 75,00 | 27,78 | 54,82 | 466.59 | 90,45 | 68,42 | 36,36 | 52,39 | 83,80 | 66,07 | 86,76 | 38,64 | 62,70 | 453,26 |
| **Partición 5** | 79,30 | 79,69 | 27,78 | 54,74 | 466.79 | 92,36 | 76,31 | 40,91 | 58,61 | 83,41 | 75,14 | 79,41 | 29,54 | 54,48 | 455,34 |
| **Media** | 80,16 | 77,42 | 28,06 | 54,37 | 465,63 | 88,24 | 83,16 | 37,27 | 60,21 | 84,42 | 78,46 | 85,00 | 36,36 | 60,68 | 454,28 |

Tabla 5.5: Resultados obtenidos por el algoritmo AGG_CA en el problema del APC

| Seed: 123564891 | Ozone | | | | | Parkinsons | | | | | Spectf-heart | | | | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| | *%_tra* | *%_clas* | *%red* | *Agr.* | T | *%_tra* | *%_clas* | *%red* | *Agr.* | T | *%_tra* | *%_clas* | *%red* | *Agr.* | T |
| **Partición 1** | 82,42 | 81,25 | 26,39 | 53,82 | 457,08 | 85,35 | 86,84 | 36,36 | 61,60 | 83,32 | 70,36 | 92,65 | 29,55 | 61,10 | 448,32 |
| **Partición 2** | 76,17 | 81,25 | 29,17 | 55,21 | 460,13 | 83,44 | 84,21 | 36,36 | 60,29 | 84,38 | 78,93 | 80,88 | 34,09 | 57,49 | 444,96 |
| **Partición 3** | 77,34 | 71,88 | 30,56 | 54,93 | 458,67 | 86,62 | 100,00 | 31,82 | 65,91 | 83,63 | 75,36 | 77,94 | 34,09 | 56,02 | 448,84 |
| **Partición 4** | 80,85 | 82,81 | 27,78 | 54,82 | 458,03 | 90,46 | 68,42 | 36,36 | 52,39 | 85,18 | 66,07 | 86,76 | 38,64 | 62,70 | 453,81 |
| **Partición 5** | 79,29 | 79,69 | 27,78 | 54,74 | 462,38 | 90,46 | 71,10 | 36,36 | 53,73 | 85,18 | 75,14 | 79,41 | 29,55 | 54,48 | 454,88 |
| **Media** | 79,21 | 79,38 | 28,34 | 54,70 | 459,26 | 87,27 | 82,11 | 35,45 | 58,78 | 86,28 | 73,17 | 83,53 | 33,18 | 58,36 | 450,16 |

En estos resultados podemos ver una comparación detallada entre ambos tipos de cruces en el modelos generacional. Podemos observar que a nivel de solución, es decir, la media entre ambas tasas, no se percibe una diferencia significativa; pero en términos de tiempo existe una pequeña diferencia del , siendo los tiempos del BLX un 2% mayores que el Mean.

Tabla 5.7: Resultados obtenidos por el algoritmo AGE_BLX en el problema del APC

| Seed: 123564891 | Ozone | | | | | Parkinsons | | | | | Spectf-heart | | | | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| | *%_tra* | *%_clas* | *%red* | *Agr.* | T | *%_tra* | *%_clas* | *%red* | *Agr.* | T | *%_tra* | *%_clas* | *%red* | *Agr.* | T |
| **Partición 1** | 82,81 | 75,00 | 62,50 | 68,75 | 463,53 | 83,36 | 92,11 | 81,82 | 86,97 | 83,36 | 76,42 | 92,65 | 47,73 | 70,19 | 400,92 |
| **Partición 2** | 80,86 | 84,38 | 48,61 | 66,50 | 464,75 | 87,32 | 68,42 | 63,64 | 66,03 | 87,32 | 74,28 | 83,82 | 63,64 | 73,73 | 407,96 |
| **Partición 3** | 88,28 | 73,44 | 61,11 | 54,93 | 464,75 | 87,31 | 89,47 | 77,27 | 83,37 | 87,31 | 77,50 | 83,82 | 50,00 | 66,91 | 416,13 |
| **Partición 4** | 83,59 | 85,94 | 56,94 | 54,82 | 463,61 | 84,10 | 71,05 | 72,73 | 71,89 | 84,10 | 83,57 | 76,47 | 50,00 | 63,24 | 415,71 |
| **Partición 5** | 82,42 | 84,34 | 54,17 | 54,74 | 463,79 | 84,19 | 42,11 | 68,18 | 55,15 | 84,19 | 80,71 | 75,00 | 56,82 | 65,91 | 418,31 |
| **Media** | 83,59 | 80,62 | 56,67 | 59,95 | 464,09 | 85,26 | 72,63 | 72,73 | 72,68 | 85,26 | 78,50 | 82,35 | 53,64 | 68,00 | 411,81 |

Tabla 5.6: Resultados obtenidos por el algoritmo AGE_CA en el problema del APC

| Seed: 123564891 | Ozone | | | | | Parkinsons | | | | | Spectf-heart | | | | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| | *%_tra* | *%_clas* | *%red* | *Agr.* | T | *%_tra* | *%_clas* | *%red* | *Agr.* | T | *%_tra* | *%_clas* | *%red* | *Agr.* | T |
| **Partición 1** | 83,59 | 76,56 | 47,22 | 61,89 | 457,08 | 87,90 | 89,47 | 68,18 | 78,83 | 84,69 | 75,00 | 95,59 | 43,18 | 69,39 | 439,77 |
| **Partición 2** | 82,81 | 81,25 | 41,67 | 61,46 | 460,13 | 90,45 | 86,84 | 63,64 | 75,24 | 85,35 | 76,07 | 83,82 | 50,00 | 66,91 | 429,96 |
| **Partición 3** | 81,25 | 70,31 | 44,44 | 54,93 | 458,67 | 88,54 | 97,37 | 59,09 | 78,23 | 83,61 | 81,07 | 73,53 | 40,91 | 57,22 | 447,34 |
| **Partición 4** | 82,03 | 85,93 | 48,61 | 54,82 | 458,03 | 75,80 | 63,18 | 72,73 | 67,96 | 84,46 | 77,14 | 85,29 | 40,91 | 63,10 | 425,51 |
| **Partición 5** | 82,42 | 81,25 | 45,83 | 54,74 | 462,38 | 94,27 | 63,16 | 40,91 | 52,04 | 85,46 | 78,57 | 76,47 | 38,64 | 57,56 | 426,22 |
| **Media** | 82,42 | 79,06 | 45,55 | 57,57 | 459,26 | 87,39 | 80,00 | 60,91 | 70,46 | 84,71 | 77,57 | 82,94 | 42,73 | 62,83 | 433,76 |

En estas tablas podemos ver la comparación entre cruces en el modelo estacionario, que al igual que pasaba en el modelo generacional, no existen diferencias en la calidad de la solución pero se sigue manteniendo la diferencia del 2% en el tiempo de ejecución.

Si paramos a observar las diferencias entre los distintos modelos, si podemos ver una diferencia en la solución. En todos los casos, tanto en el cruce BLX como en el Mean, el esquema estacionario genera mejores soluciones que el esquema generacional.

Esto puede ser debido a que en el estacionario lo hijos deben competir por entrar en la población, por lo que siempre aseguras que la población siempre mejore en cada iteración; mientras que en el modelo generacional, aunque se mantenga siempre la mejor solución encontrada hasta el momento puede haber peores soluciones que en la anterior generación.

A continuación se presentarán los datos recogidos sobre los algoritmos meméticos. Recordemos que el algoritmo memético utilizado está montado sobre el algoritmo AGG_BLX, que es con el cual estableceremos la comparación.

Tabla 5.8: Resultados obtenidos por el algoritmo AMG_10_01 en el problema del APC

| Seed: 123564891 | Ozone | | | | | Parkinsons | | | | | Spectf-heart | | | | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| | *%_tra* | *%_clas* | *%red* | *Agr.* | T | *%_tra* | *%_clas* | *%red* | *Agr.* | T | *%_tra* | *%_clas* | *%red* | *Agr.* | T |
| **Partición 1** | 85,94 | 78,13 | 52,78 | 65,46 | 1268,37 | 87,9 | 100,00 | 54,55 | 77,28 | 165,68 | 82,50 | 89,71 | 40,91 | 65,31 | 937,10 |
| **Partición 2** | 83,59 | 79,69 | 45,83 | 62,76 | 1232,36 | 85,35 | 78,94 | 54,55 | 66,75 | 169,21 | 80,00 | 79,41 | 50,00 | 64,71 | 926,98 |
| **Partición 3** | 82,03 | 67,19 | 41,67 | 54,93 | 1240,37 | 80,25 | 94,74 | 63,64 | 79,19 | 158,19 | 82,50 | 64,71 | 38,64 | 51,68 | 936,99 |
| **Partición 4** | 84,77 | 81,25 | 33,33 | 54,82 | 1217,96 | 92,36 | 52,63 | 63,64 | 58,14 | 156,21 | 79,64 | 75,00 | 50,00 | 62,50 | 925,64 |
| **Partición 5** | 86,72 | 84,38 | 84,38 | 54,74 | 1219,32 | 89,17 | 28,94 | 63,64 | 46,29 | 159,59 | 83,57 | 80,88 | 52,27 | 66,58 | 922,28 |
| **Media** | 84,61 | 78,13 | 51,60 | 58,54 | 1235,68 | 71,05 | 60,00 | 65,53 | 65,53 | 161,78 | 81,64 | 77,94 | 46,36 | 62,15 | 929,80 |

Tabla 5.9: Resultados obtenidos por el algoritmo AMG_10_01mej en el problema del APC

| Seed: 123564891 | Ozone | | | | | Parkinsons | | | | | Spectf-heart | | | | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| | *%_tra* | *%_clas* | *%red* | *Agr.* | T | *%_tra* | *%_clas* | *%red* | *Agr.* | T | *%_tra* | *%_clas* | *%red* | *Agr.* | T |
| **Partición 1** | 85,55 | 71,88 | 62,50 | 67,19 | 1253,76 | 89,17 | 100,00 | 45,45 | 72,73 | 140,67 | 82,86 | 97,06 | 36,36 | 66,71 | 844,80 |
| **Partición 2** | 80,86 | 84,38 | 48,61 | 66,50 | 1246,56 | 88,53 | 86,84 | 40,91 | 63,88 | 148,94 | 81,76 | 86,76 | 29,55 | 58,16 | 828,75 |
| **Partición 3** | 83,59 | 73,44 | 61,11 | 54,93 | 1252,00 | 81,83 | 71,05 | 68,18 | 69,62 | 132,17 | 80,36 | 75,00 | 40,91 | 57,96 | 826,00 |
| **Partición 4** | 86,72 | 82,81 | 56,94 | 54,82 | 1234,86 | 93,63 | 44,74 | 68,18 | 56,46 | 125,59 | 82,50 | 75,00 | 47,73 | 61,37 | 829,05 |
| **Partición 5** | 83,98 | 79,69 | 54,17 | 54,74 | 1238,59 | 89,17 | 57,89 | 63,64 | 60,77 | 125,02 | 86,07 | 79,41 | 25,00 | 52,21 | 823,59 |
| **Media** | 84,14 | 78,44 | 56,67 | 59,63 | 1245,15 | 88,47 | 72,10 | 57,27 | 64,69 | 134,48 | 82,71 | 82,65 | 35,91 | 59,28 | 830,44 |

Se puede apreciar que existe una diferencia clara entre la calidad de la solución del AGG_BLX y el AMG_10_01 o el AMG_10_mej, siendo la calidad de estos últimos mayor que el algoritmo genético.

Sin embargo, al contrario de lo que podríamos pensar, AMG_10_01 genera mejores soluciones que el algoritmo AMG_10_01mej. Esto podría ser debido a que este último en ocasiones es muy selectivo con las características.

La tasa de reducción tiene un papel protagonista en esta selección de características. Recordemos que la tasa de reducción es el número de componentes de ω cuyo valor es inferior a 0’2. Estos valores, en el momento de calcular la tasa de clasificación, tanto en el *training* como en el *test*, se reducen a 0; esto quiere decir que esta característica no es relevante para la clasificación.

Esto se utiliza para hacer un clasificador más simple, sin embargo en casos extremos (entiéndase extremo como caso en el que se obtenga tasa de reducción muy alta), puede llegar a ser perjudicial, ya que conforme aumenta la tasa de reducción, aumenta también el parecido con el algoritmo Clasificador-1NN, en el cual el proceso de aprendizaje de características es nulo.

Como última observación, en todas las tablas mostradas en este apartado, podemos ver que en ocasiones la tasa de clasificación de *training* es mayor que la tasa de clasificación en *test*. Esto es debido a que el clasificador generado por el algoritmo, se ajusta demasiado al conjunto *training*, por lo que el error en *test* es alto. Un ejemplo de esto se puede ver en la tabla 5.7, en la partición 5 del conjunto de datos de Parkinsons.

Esto puede deberse a que la muestra de *training* no es suficientemente grande o que el conjunto de *training* no es representativo de la muestra, lo que explicaría la gran tasa de error en el *test*.

En esta última tabla se puede observar una comparación de todos los algoritmos mostrados hasta ahora

| | Ozone | | | | | Parkinsons | | | | | Spectf-heart | | | | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| | *%_tra* | *%_clas* | *%red* | *Agr.* | T | *%_tra* | *%_clas* | *%red* | *Agr.* | T | *%_tra* | *%_clas* | *%red* | *Agr.* | T |
| 1-NN | 82,87 | 40,63 | 0,00 | 20,31 | 0,04 | 89,15 | 44,72 | 0,00 | 22,36 | 0,01 | 80,32 | 36,61 | 0,00 | 18,30 | 0,04 |
| RELIEF | 80,08 | 79,07 | 2,50 | 40,79 | 0,08 | 91,21 | 79,47 | 3,64 | 41,56 | 0,02 | 76,07 | 80,59 | 27,27 | 53,93 | 0,08 |
| BL | 82,50 | 79,07 | 20,55 | 51,87 | 58,93 | 90,32 | 75,79 | 16,36 | 46,08 | 3,59 | 75,86 | 79,41 | 17,73 | 48,57 | 30,20 |
| AGG_BLX | 80,16 | 77,42 | 28,06 | 54,37 | 465,63 | 88,24 | 83,16 | 37,27 | 60,21 | 84,42 | 78,46 | 85,00 | 36,36 | 60,68 | 454,28 |
| AGG_CA | 79,21 | 79,38 | 28,34 | 54,70 | 459,26 | 87,27 | 82,11 | 35,45 | 58,78 | 86,28 | 73,17 | 83,53 | 33,18 | 58,36 | 450,16 |
| AGE_BLX | 83,59 | 80,62 | 56,67 | 59,95 | 464,09 | 85,26 | 72,63 | 72,73 | 72,68 | 85,26 | 77,57 | 82,94 | 42,72 | 62,83 | 433,76 |
| AGE_CA | 82,42 | 79,06 | 45,55 | 57,57 | 459,26 | 87,39 | 80,00 | 60,91 | 70,46 | 84,71 | 77,57 | 82,94 | 42,73 | 62,83 | 433,76 |
| AMG_10_01 | 84,61 | 78,13 | 51,60 | 58,54 | 1235,68 | 71,05 | 60,00 | 65,53 | 65,53 | 161,78 | 81,64 | 77,94 | 46,36 | 62,15 | 929,80 |
| AMG_10_01mej | 84,14 | 78,44 | 56,67 | 59,63 | 1245,15 | 88,47 | 72,10 | 57,27 | 64,69 | 134,48 | 82,71 | 82,65 | 35,91 | 59,28 | 830,44 |