Universidad de Granada

Ingeniería Informática

Computación y Sistemas Inteligentes

Problemas de Optimización con Algoritmos Genéticos y Meméticos

Autor: José Antonio Ruiz Millán
email: jantonioruiz@correo.ugr.es
Curso: 2017-2018
Problema: 1.b Aprendizaje de Pesos en Características (APC)
Algoritmos: Genético y Memético
Grupo: A3
Asignatura: Metaheurísticas
20 de julio de 2018

Índice

1. Descripción del problema

El problema del APC consiste en optimizar el rendimiento de un clasificador basado en vecinos más cercanos a partir de la inclusión de pesos asociados a las características del problema que modifican su valor en el momento de calcular las distancias entre ejemplos. En nuestro caso, el clasificador considerado será el 1-NN (k-NN, k vecinos más cercanos, con k=1 vecino). La variante del problema del APC que afrontaremos busca optimizar tanto la precisión como la complejidad del clasificador. Así, se puede formular como:

Maximizar $F(W) = \alpha \cdot \text{tasa\_clas(W)} + (1-\alpha) \cdot \text{tasa\_red(W)}$

con:

- $\text{tasa\_clas} = 100 \cdot \frac{n^{\text{o}} instancias \quad bien \quad clasificadas \quad en \quad T}{n^{\text{o}} instancias \quad en \quad T}$
- $\text{tasa\_red} = 100 \cdot \frac{n^{\text{o}} valores \quad w_i < 0{,}2}{n^{\text{o}} caracteristicas}$

sujeto a que :

- $w_i = [0, 1], 1 \leq i \leq n$

donde:

- $W = (w_1, ..., w_n)$ es una solución al problema que consiste en un vector de números reales $w_1 \in [0, 1]$ de tamaño n que define el peso que pondera o filtra a cada una de las características f_i.
- 1-NN es el clasificador k-NN con $k = 1$ vecino generado a partir del conjunto de datos inicial utilizando los pesos en W que se asocian a las n características.
- T es el conjunto de datos sobre el que se evalúa el clasificador, ya sea el conjunto de entrenamiento (usando la técnica de validación leave-one-out) o el de prueba.
- $\alpha \in [0, 1]$ pondera la importancia entre el acierto y la reducción de la solución encontrada.

Para la distribución de los datos de entrada utilizaremos la técnica de validación cruzada ***5-fold cross validation*** de la siguiente manera:

- El conjunto de datos se divide en 5 particiones disjuntas al 20 %, con la distribución de clases equilibrada.
- Aprenderemos un clasificador utilizando el 80 % de los datos disponibles (4 particiones de las 5) y validaremos con el 20 % restante (la partición restante) $\rightarrow$ 5 particiones posibles al 80-20 %
- Así obtendremos un total de 5 valores de porcentaje de clasificación en el conjunto de prueba, uno para cada partición empleada como conjunto de validación
- La calidad del método de clasificación se medirá con un único valor, correspondiente a la media de los 5 porcentajes de clasificación del conjunto de prueba

2. Descripción de los algoritmos

2.1. Componentes comunes

Para los algoritmos de esta práctica, vamos a definir primero los elementos comunes que comparten:

- **Esquema de representación**: Se seguirá la representación real basada en un vector W de tamaño n con valores en $[0, 1]$ que indican el peso asociado a cada característica y la capacidad para eliminarla si su peso es menor que 0.2.
- **Función objetivo**: Será la combinación con pesos de las medidas de precisión (tasa de acierto sobre el conjunto de entrenamiento) y la complejidad (la tasa de reducción de características con respecto al conjunto original) del clasificador 1-NN diseñado empleando el vector W. Para calcular la tasa de acierto será necesario emplear la técnica de validación leave-one-out. El valor de σ considerado será $\sigma = 0{,}5$, dándole la misma importancia a ambos criterios. El objetivo será maximizar esta función.
- **Generación de la solución inicial**: La solución inicial se generará de forma aleatoria utilizando una distribución uniforme en $[0, 1]$ en todos los casos.
- **Esquema de generación de vecinos**: Se empleará el movimiento de cambio por mutación normal $Mov(W, \sigma)$ que altera el vector W sumándole otro vector Z generado a partir de una distribución normal de media 0 y varianza $sigma^2$. Su aplicación concreta dependerá del algoritmo específico.
- **Criterio de aceptación**: Se considera una mejora cuando se aumenta el valor global de la función objetivo.

2.2. Algoritmos genéticos

Para esta práctica, definimos los siguientes elementos:

- **Esquema de evolución**: Se considerarán dos versiones, una basada en el esquema generacional con elitismo (AGG) y otra basada en el esquema estacionario (AGE). En el primero se seleccionará una población de padres del mismo tamaño que la población genética mientras que en el segundo se seleccionarán únicamente dos padres.
- **Operador de selección**: Se usará el torneo binario, consistente en elegir aleatoriamente dos individuos de la población y seleccionar el mejor de ellos. En el esquema generacional, se aplicarán tantos torneos como individuos existan en la población genética, incluyendo los individuos ganadores en la población de padres. En el esquema estacionario, se aplicará dos veces el torneo para elegir los dos padres que serán posteriormente recombinados (cruzados).
- **Esquema de reemplazamiento**: En el esquema generacional, la población de hijos sustituye automáticamente a la actual. Para conservar el elitismo, si la mejor solución de la generación anterior no sobrevive, sustituye directamente la peor solución de la nueva población. En el estacionario, los dos descendientes generados tras el cruce y la mutación (esta última aplicada según una determinada probabilidad) sustituyen a los dos peores de la población actual, en caso de ser mejores que ellos.

- **Operador de cruce**: Se emplearán dos operadores de cruce para representación real. Uno de ellos será el operador BLX-σ, con $\sigma = 0{,}3$. El otro será el cruce aritmético. Esto resultará en el desarrollo de cuatro AGs distintos, dos generacionales (AGG-BLX y AGG-CA) y dos estacionarios (AGE-BLX y AGE-CA).
- **Operador de mutación**: Se considerará el operador de mutación normal para representación real. Para aplicarlo, se considerará un valor $\sigma = 0,3$. Coincidrá por tanto con el operador de generación de vecinos de la BL.

El tamaño de la población será de 30 cromosomas. La probabilidad de cruce será 0,7 en el AGG y 1 en el AGE (siempre se cruzan los dos padres). La probabilidad de mutación (por gen) será de 0,001 en ambos casos. El criterio de parada en las dos versiones del AG consistirá en realizar **15000 evaluaciones de la función objetivo**.

2.3. Algoritmos Meméticos

El AM consistirá en hibridar el algoritmo genético generacional (AGG) que mejor resultado haya proporcionado con la BL desarrollada en la Práctica 1.b. Se estudiarán las tres posibilidades de hibridación siguientes:

1. **AM-(10,1.0)**: Cada 10 generaciones, se aplica la BL sobre todos los cromosomas de la población.
2. **AM-(10,0.1)**: Cada 10 generaciones, se aplica la BL sobre un subconjunto de cromosomas de la población seleccionado aleatoriamente con probabilidad p_{LS} igual a 0.1 para cada cromosoma.
3. **AM-(10,0.1mej)**: Cada 10 generaciones, aplicar la BL sobre los 0.1N mejores cromosomas de la población actual (N es el tamaño de ésta).

El tamaño de la población del AGG será de 10 cromosomas. Las probabilidades de cruce y mutación serán 0,7 (por cromosoma) y 0,001 (por gen) en ambos casos. Se detendrá la ejecución de la BL aplicada sobre un cromosoma **cuando se hayan evaluado 2n vecinos distintos en la ejecución**, siendo n el tamaño del cromosoma. El criterio de parada del AM consistirá en realizar **15000 evaluaciones de la función objetivo**, incluidas por supuesto las de la BL.

3. Pseudocodigo algoritmos

Vamos a definir ahora los distintos pseudocódigos necesarios para esta práctica. Naturalmente, las funciones que expliqué en la práctica anterior no aparecerán de nuevo en esta.

En la estructura de datos utilizada, tenemos como valores globales un mejor y un mejor global que utilizaremos en las siguiente definiciones.

3.1. Algoritmo genético generacional

Algunas de estas funciones se utilizarán también en los siguientes apartados, por lo que los definiré aquí y en los apartados siguientes sólo haré referencia a ellos sin tener que expresarlos de nuevo directamente.

Algoritmo Genético Generacional (AGG)

Aquí defino toda la lógica de este algoritmo utilizando como parada los datos que nos establecen en esta practica. **Tanto variable evaluaciones como bestMejor que utilizamos en el algoritmo, son variables globales para la clase** que se modifican cuando llamamos a evaluar[3].

input : datosTraining[][],datosTest[][],K,Tcromosoma,Tpoblacion,P_c,P_m
output: pesos[$x_i, ..., x_n$]
poblacion ← iniciarPoblacion(Tpoblacion,Tcromosoma);
valores ← evaluar(datosTraining,datosTest,K,poblacion,Tcromosoma,Tpoblacion);
while *evaluaciones < 15000* **do**
  poblacion ← seleccion(poblacion,Tpoblacion,valores);
  poblacion ← cruce(poblacion,Tpoblacion,Tcromosoma,[α],P_c);
  poblacion ← mutacion(poblacion,Tpoblacion,Tcromosoma,P_m);
  poblacion ← reemplazo(poblacion);
  valores ← evaluar(datosTraining,datosTest,K,poblacion,Tcromosoma,Tpoblacion);
end
return *bestMejor #Variable global*

Algorithm 1: AGG

Pasamos ahora a definir cada una de las funciones utilizadas en el algoritmo.

IniciarPoblacion

Esta función nos creara una población inicial aleatoria para poder comenzar a explorar las distintas soluciones del problema.

- Dado un número de poblacion determinado, para cada cromosoma, recorremos todos sus genes y les asisgnamos un valor aleatorio generado uniformemente en el rango $[0, 1]$

input : Tpoblacion,Tcromosoma
output: poblacion[x_i[], ..., x_n[]]
poblacion[][] ← ∅;
for $i \leftarrow 1$ **to** *Tpoblacion* **do**
  cromosoma[] ← ∅;
  for $j \leftarrow 1$ **to** *Tcromosoma* **do**
    cromosoma.add(next.Double());
  end
  poblacion.add(cromosoma);
end
return *poblacion*;

Algorithm 2: iniciarPoblacion

Evaluar

Esta función nos evalúa todos los cromosomas de la problacion y los guarda en un contenedor externo. **He de comentar que aunque en el pseudocódigo pongo tanto training como test, los datos son los mismos (training) ya que estos algorimos son para calcular unos pesos con los datos de aprendizaje, sin ver los datos de test.**

- Dada una población, recorre todos los cromosomas de la misma y evalúa el cromosoma para obtener su valor en la función que nosotros utilizamos.
- Una vez obtenido el valor, se guarda en un contenedor externo en el mismo orden en el que se encuentra la población para tener los datos por parejas. En cada iteración, comprobamos si el valor de este cromosoma es el mejor actual de la población.
- Por último, comprobamos si el mejor de esta nueva población es el mejor global de todas las poblaciones vistas.

En este algoritmo, utilizamos la funcion **knn** definida en la practica anterior. Decir también que tenemos la variable global Emejor y Ebestmejor que nos dice el valor de la funcion onbjetivo en el mejor cromosoma actual de la población y el mejor global.

Recordar que los datos de test son una copia de los datos de training, ya que aprendemos sobre éstos.

input : datosTraining[][],datosTest[][],K,poblacion[][],Tcromosoma,Tpoblacion
output: valores[$x_i, ..., x_n$]
valores[] $\leftarrow \emptyset$;
Emejor $\leftarrow$ 0.0 # *Variable global*;
mejor[] $\leftarrow \emptyset$ # *Variable global*;
for $i \leftarrow 1$ **to** *Tpoblacion* **do**
    pesos[] $\leftarrow$ poblacion[i];
    knn $\leftarrow$ new knn(datosTraining,datosTest,pesos,K);
    f $\leftarrow$ knn.knn();
    valores.add(f);
    if $f >$ *Emejor* **then**
        mejor $\leftarrow$ poblacion[i];
        Emejor $\leftarrow$ f;
    end
    evaluaciones $\leftarrow$ evaluaciones+1 # *Variable global*;
end
if *Emejor* $>$ *Ebestmejor* **then**
    bestmejor $\leftarrow$ mejor;
    Ebestmejor $\leftarrow$ Emejor;
end
return *valores*;

Algorithm 3: Evaluar

Selección

Esta función nos selecciona N miembros aleatorios de la poblacion, siendo N en tamaño de la misma.

- Dada una población, genera dos números aleatorios y selecciona estos dos cromosomas.

- Una vez seleccionados, realiza un torneo para elegir el mejor de los dos cromosomas.
- Por último, añadimos este cromosoma seleccionado a la nueva población.

input : poblacion[][],Tpoblacion,valores[]
output: newpoblacion[x_i[], ..., x_n[]]
newpoblacion[][] $\leftarrow \emptyset$;
par1 $\leftarrow$ 0;
par2 $\leftarrow$ 0;
for $i \leftarrow 1$ **to** *Tpoblacion* **do**
  while *par1* == *par2* **do**
    par1 $\leftarrow$ next.Int(Tpoblacion);
    par2 $\leftarrow$ next.Int(Tpoblacion);
  end
  top $\leftarrow$ (valores[par1] > valores[par2])?par1:par2;
  newpoblacion.add(poblacion[top]);
end
return *newpoblacion*;

Algorithm 4: Seleccion

Cruce-BLX-α

Esta función nos cruzará un determinado número de cromosomas aleatorios dada una probabilidad de cruce.

Como después de la selección ya tenemos los cromosomas ordenados aleatoriamente, para este proceso, seleccionamos las parejas en el orden en el que se ecuentran almacenadas. Para la cantida de elementos que necesitamos cruzar, vamos a utilizar la esperanza matemática, esto es que cruzaremos $P_c \cdot Tpoblacion$ elementos o $\frac{P_c \cdot Tpoblacion}{2}$ parejas.

- Calcula el número de cruces que vamos a realizar.
- Realizamos los distintos cálculos que necesita este tipo de cruce.
- Recorremos la poblacion realizando los cruces y almacenándolos en un contenedor nuevo.
- Insertamos los elementos de la poblacion que no han sido cruzados hasta llegar al tamaño de la poblacion que tenemos.

input : poblacion[][],Tpoblacion,Tcromosoma,α,P_c
output: newpoblacion[x_i[], ..., x_n[]]
newpoblacion[][] $\leftarrow \emptyset$;
cruces $\leftarrow P_c \cdot Tpoblacion$;
for $i \leftarrow 1$ **to** *cruces* **;** $i \leftarrow i+2$ **do**
  des1[] $\leftarrow \emptyset$;
  des2[] $\leftarrow \emptyset$;
  for $j \leftarrow 1$ **to** *Tcromosoma* **do**
    cmax $\leftarrow$ (poblacion[i][j] > poblacion[i+1][j])?poblacion[i][j]:poblacion[i+1][j];
    cmin $\leftarrow$ (poblacion[i][j] < poblacion[i+1][j])?poblacion[i][j]:poblacion[i+1][j];
    I $\leftarrow cmax - cmin$;
    min $\leftarrow cmin - I \cdot \alpha$;
    max $\leftarrow cmax + I \cdot \alpha$;
    des1.add($min + (max - min) \cdot next.Double()$);
    des2.add($min + (max - min) \cdot next.Double()$);
    des1[j] $\leftarrow$ Truncar(des1[j]) #[0,1];
    des2[j] $\leftarrow$ Truncar(des2[j]) #[0,1];
  end
  newpoblacion.add(des1);
  newpoblacion.add(des2);
end
ini $\leftarrow Tpoblacion - (Tpoblacion - newpoblacion.size())$;
for $i \leftarrow ini$ **to** *Tpoblacion* **do**
  newpoblacion.add(poblacion[i]);
end
return *newpoblacion*;

Algorithm 5: Cruce-BLX-α

Cruce-CA

Esta función nos cruzará un determinado número de cromosomas aleatorios dada una probabilidad de cruce.

Como después de la selección ya tenemos los cromosomas ordenados aleatoriamente, para este proceso, seleccionamos las parejas en el orden en el que se ecuentran almacenadas. Para la cantida de elementos que necesitamos cruzar, vamos a utilizar la esperanza matemática, esto es que cruzaremos $P_c \cdot Tpoblacion$ elementos o $\frac{P_c \cdot Tpoblacion}{2}$ parejas.

- Calcula el número de cruces que vamos a realizar.
- Realizamos los distintos cálculos que necesita este tipo de cruce.
- Recorremos la poblacion realizando los cruces y almacenándolos en un contenedor nuevo.
- Insertamos los elementos de la poblacion que no han sido cruzados hasta llegar al tamaño de la poblacion que tenemos.

input : poblacion[][],Tpoblacion,Tcromosoma,P_c
output: newpoblacion[x_i[], ..., x_n[]]
newpoblacion[][] $\leftarrow \emptyset$;
cruces $\leftarrow P_c \cdot Tpoblacion$;
for $i \leftarrow 1$ **to** *cruces* **;** $i \leftarrow i+2$ **do**
  des1[] $\leftarrow \emptyset$;
  des2[] $\leftarrow \emptyset$;
  $\alpha \leftarrow$ next.Double();
  for $j \leftarrow 1$ **to** *Tcromosoma* **do**
    des1.add($\frac{poblacion[i][j]+poblacion[i+1][j]}{2}$);
    des2.add($\alpha \cdot poblacion[i][j] + (1-\alpha) \cdot poblacion[i+1][j]$)
    des1[j] $\leftarrow$ Truncar(des1[j]) #[0,1];
    des2[j] $\leftarrow$ Truncar(des2[j]) #[0,1];
  end
  newpoblacion.add(des1);
  newpoblacion.add(des2);
end
ini $\leftarrow Tpoblacion - (Tpoblacion - newpoblacion.size())$;
for $i \leftarrow$ *ini* **to** *Tpoblacion* **do**
  newpoblacion.add(poblacion[i]);
end
return *newpoblacion*;

Algorithm 6: Cruce-CA

Mutación

Esta función nos mutará un determinado número de genes aleatorios dada una probabilidad de mutación.

- Sleccionamos dos números aleatorios, uno para la fila y otro para la columna de la matriz donde tengo almacenada la población.
- Realizamos los distintos cálculos para este gen en concreto.
- Actualizamos el valor de la mutación y nos aseguramos de truncarlo para que no se salga de $[0, 1]$.
- Insertamos los elementos de la poblacion que no han sido cruzados hasta llegar al tamaño de la poblacion que tenemos.

input : poblacion[][],Tpoblacion,Tcromosoma,P_m
output: newpoblacion[x_i[], ..., x_n[]]
mutaciones $\leftarrow P_m \cdot Tpoblacion \cdot Tcromosoma$;
for $i \leftarrow 1$ **to** *mutaciones* **do**
    fila $\leftarrow$ next.Int(Tpoblacion);
    colum $\leftarrow$ next.Int(Tcromosoma);
    poblacion[fila][colum] $\leftarrow$ $poblacion[fila][colum] + next.Gaussian() \cdot 0{,}3$;
    poblacion[fila][colum] $\leftarrow$ Truncar(poblacion[fila][colum]) #[0,1];
end
return *poblacion*;

Algorithm 7: Mutacion

Reemplazo

Esta función nos reemplaza el mejor de cada población en la poblacion nueva.

Para este caso, lo que hago es añadir el mejor de la población sin más, es decir, sin comprobar si ya está y sin eliminar al peor ya que no sabemos quién es. Para paliar esto, elimino un elemento al azar antes de insertar al mejor y una vez eliminado, inserto al mejor.

- Eliminamos un elemento al azar.
- Añadimos el mejor de la poblacion **(variable global)** a la población.

input : poblacion[][]
output: newpoblacion[x_i[], ..., x_n[]]
poblacion.remove(next.Int(poblacion.size()));
poblacion.add(mejor);
return *poblacion*

Algorithm 8: Reemplazo

3.2. Algoritmo Genético Estacionario.

En este algoritmo sólo definiré las funciones que no están definidas aún. Para este algoritmo he utilizado un contenedor externo ordenado donde almaceno de menor a mayor todos los cromosomas de la población.

Tanto la funcion *IniciarPoblacion*[2], *Seleccion*[4], *Evaluar*[3], *Cruce*[5][6] y *Mutacion*[7] son exactamente iguales que en el apartado anterior.

Por lo que sólo nos queda definir la función reemplazo aunque también indicaré breves modificaciones sobre algunas de las funciones anteriores. Aprobechando que toda la población ya la tenemos evaluada, en cada generación cuando realizamos el reemplazo sólo busco el mejor de la población y no realizo una evaluacion de todos los cromosomas ya que estos valores ya los tengo almacenados y ahorramos calculos. Para ello, creo una función que la he llamado evaluarMejor() que se encarga únicamente de actualizar el mejor ya que todos los elementos ya los tenemos correctamente evaluados.

Algoritmo Genético Estacionario (AGE)

Aquí defino toda la lógica de este algoritmo utilizando como parada los datos que nos establecen en esta practica. **Tanto variable evaluaciones como bestMejor que utilizamos en el algoritmo, son variables globales para la clase** que se modifican cuando llamamos a evaluar[3].

input : datosTraining[][],datosTest[][],K,Tcromosoma,Tpoblacion,P_c,P_m
output: pesos[$x_i, ..., x_n$]
poblacion ← iniciarPoblacion(Tpoblacion,Tcromosoma);
valores ← evaluar(datosTraining,datosTest,K,poblacion,Tcromosoma,Tpoblacion);
while *evaluaciones < 15000* **do**
    poblacion ← seleccion(poblacion,Tpoblacion,valores);
    poblacion ← cruce(poblacion,Tpoblacion,Tcromosoma,[α],P_c);
    poblacion ← mutacion(poblacion,Tpoblacion,Tcromosoma,P_m);
    poblacion,valores ← reemplazo(poblacion,datosTraining,datosTest,K);
    evaluarMejor();
end
return *bestMejor #Variable global*

Algorithm 9: AGE

Iniciar Poblacion

Esta función no contempla ningún cambio respecto a la funcion *IniciarPoblacion*[2] anterior.

Evaluar

He de comentar, que la función *Evaluar*[3] tiene un pequeño cambio, ya que ahora tenemos una nueva variable global llamada **ordenados** que alamcena los cromosomas ordenados. Es por ello, que cuando evaluamos un valor, también lo añadimos a ordenados utilizando como clave el valor de la función objetivo y como valor el propio vector de pesos. **Recordar que los datos de test**

son una copia de los datos de training, ya que aprendemos sobre éstos.

input : datosTraining[][],datosTest[][],K,poblacion[][],Tcromosoma,Tpoblacion
output: valores[$x_i, ..., x_n$]
valores[] ← ∅;
Emejor ← 0.0 #*Variable global*;
mejor[] ← ∅ #*Variable global*;
ordenados[key,value] ← ∅ #*Variable global*;
for $i \leftarrow 1$ **to** *Tpoblacion* **do**
    pesos[] ← poblacion[i];
    knn ← new knn(datosTraining,datosTest,pesos,K);
    f ← knn.knn();
    valores.add(f);
    ordenados.add(f,pesos);
    if $f >$ *Emejor* **then**
        mejor ← poblacion[i];
        Emejor ← f;
    end
    evaluaciones ← evaluaciones+1 #*Variable global*;
end
if *Emejor* > *Ebestmejor* **then**
    bestmejor ← mejor;
    Ebestmejor ← Emejor;
end
return *valores*;

Algorithm 10: Evaluar

Seleccion

La funcion *Seleccion*[4] tiene un pequeño cambio también, y es que cuando realizamos N torneos en el generacional, aquí hacemos solo 2, por lo que el cambio en el algoritmo sería cambiar en el bucle, N por 2.

input : poblacion[][],Tpoblacion,valores[]
output: newpoblacion[x_i[], ..., x_n[]]
newpoblacion[][] ← ∅;
par1 ← 0;
par2 ← 0;
for $i \leftarrow 1$ **to** *2* **do**
    while *par1 == par2* **do**
        par1 ← next.Int(Tpoblacion);
        par2 ← next.Int(Tpoblacion);
    end
    top ← (valores[par1] > valores[par2])?par1:par2;
    newpoblacion.add(poblacion[top]);
end
return *newpoblacion*;

Algorithm 11: Seleccion

Cruce

Los cruces son exactamente iguales que en la función *Cruce*[5][6] anterior.

Mutación

La *Mutacion*[7] se realiza de la misma forma que en el apartado anterior.

Reemplazo

En este caso, si tenemos cambios más significativos, es por ello por lo que vamos a definir el método.

- En primer lugar, evaluamos los dos padres que hemos creado de la población nueva.
- Una vez evaluados, los añado a ordenados para tener tanto la poblacion anterior completa mas estos dos nuevos padres ordenados.
- Ahora eliminamos dos elementos de ordenados. Al estar ordenados, aseguramos que los dos eliminados son los 2 cromosomas con menor valor de la población.
- Ponemos los cromosomas ordenados como la nueva poblacion y también actualizamos su valor.

input : poblacion[][],datosTraining[][],datosTest[][],K
output: newpoblacion[x_i[], ..., x_n[]],newvalores[x_i, ..., x_n]
valores ← ∅;
for $i \leftarrow 1$ **to** *poblacion.size()* **do**
    pesos ← poblacion[i];
    knn ← new knn(datosTraining,datosTest,pesos,K);
    valor ← knn.knn();
    ordenados.add(valor,pesos);
end
ordenados.remove(ordenados.firstKey()) # Ordenado de menor a mayor;
ordenados.remove(ordenados.firstKey());
poblacion.clear();
for $i \leftarrow 1$ **to** *ordenados.size()* **do**
    key ← ordenados.getKey(i);
    valor ← ordenados.get(key);
    poblacion.add(valor);
    valores.add(key);
end
return *poblacion,valores*

Algorithm 12: reemplazo AGE

Evaluar Mejor Este método como he indicado anteriormente, lo utilizo porque tenemos el AGE que como la población nueva solo tenemos 2 elementos y los evaluamos para poder compararlos con la problacion anterior, en cada generación la funcion evaluar solo sacara el mejor de cada poblacion y listo, ya que en el reemplazo ya tenemos la poblacion nueva y evaluada. **Las variables Emejor, Ebestmejor,mejor,bestmejor y ordenados** son variables de la clase.

- Al tener los elementos ordenados, sólo tenemos que cojer el último del map y listo.

input :
output:
key ← ordenados.lastKey();
mejor ← ordenados.get(key);
Emejor ← key;
if *Emejor > Ebestmejor* **then**
  bestmejor ← mejor;
  Ebestmejor ← Emejor;
end

Algorithm 13: Evaluar Mejor

3.3. Algoritmo Memético (AM-10,1.0)

Como en los apartados anteriores, solo voy a definir los métodos que no hemos definido aún. Tanto la funcion *IniciarPoblacion*[2], *Seleccion*[4], *Evaluar*[3], *Cruce*[5][6], *Mutacion*[7] y *Reemplazo*[8] son exactamente iguales que en el AGG.

Algoritmo Memético (AM-10,1.0)

Utilizaremos la BL de la practica anterior en la siguiente funcion.

input : datosTraining[][],datosTest[][],K,Tcromosoma,Tpoblacion,P_c,P_m
output: pesos[$x_i, ..., x_n$]
generaciones ← 0;
poblacion ← iniciarPoblacion(Tpoblacion,Tcromosoma);
valores ← evaluar(datosTraining,datosTest,K,poblacion,Tcromosoma,Tpoblacion);
while *evaluaciones < 15000* **do**
  if *generaciones %10 == 0* **then**
    poblacion,valores ← busquedaLocal(poblacion,datosTraining,datosTest,K)
  end
  poblacion ← seleccion(poblacion,Tpoblacion,valores);
  poblacion ← cruce(poblacion,Tpoblacion,Tcromosoma,[α],P_c);
  poblacion ← mutacion(poblacion,Tpoblacion,Tcromosoma,P_m);
  poblacion ← reemplazo(poblacion,datosTraining,datosTest,K);
  valores ← evaluar(datosTraining,datosTest,K,poblacion,Tcromosoma,Tpoblacion);
  generaciones ← generaciones+1;
end
return *bestMejor #Variable global*

Algorithm 14: AM-10,1.0

Busqueda Local

Esta función se encarga de ejecutar la búsqueda local a un numero determinado de elementos de la población utilizando la BL de la practica anterior, con la única diferencia del cambio para la condición de parada que nos indican ahora en esta práctica.

- Recorremos la población completa, y para cada uno de los cromosomas, lanzamos una búsqueda local.

- Actualizamos el cromosoma al que hemos lanzado la búsqueda local por el resultado de la búsqueda y también actualizamos su valor (nos lo da el método BL). Se le pasa por referencia las evaluaciones que llevamos para que las actualice ya que en la BL tambien contamos las propias evaluaciones.
- Comprobamos si se actualiza el mejor.
- Cuando hayamos pasado por toda la población, terminamos.

Recordar que tanto datosTraining como datosTest son iguales, ya que para el calculo de los pesos los test no se utilizan, por lo que los dos contienen lo mismo (datosTraining).

input : poblacion,datosTraining[][],datosTest[][],K
output: poblacion[x_i[], ..., x_n[]],valores[x_i, ..., x_n]
valores ← ∅;
for $i \leftarrow 1$ **to** *poblacion.size()* **do**
    pesos ← poblacion[i];
    bl ← new BL(datosTraining,datosTest,pesos,K);
    poblacion[i],valores[i] ← bl.BL(&evaluaciones);
    if *valores[i] > Emejor* **then**
        mejor ← poblacion[i];
        Emejor ← valores[i];
    end
    if *Emejor > Ebestmejor* **then**
        bestmejor ← mejor;
        Ebestmejor ← Emejor;
    end
end
return *poblacion,valores*

Algorithm 15: busquedaLocal

3.4. Algoritmo Memético (AM-10,0.1)

Como en los apartados anteriores, solo voy a definir los métodos que no hemos definido aún. Tanto la funcion *IniciarPoblacion*[2], *Seleccion*[4], *Evaluar*[3], *Cruce*[5][6], *Mutacion*[7] y *Reemplazo*[8] son exactamente iguales que en el AGG.

Algoritmo Memético (AM-10,0.1)

Para este algoritmo, utilizamos exactamente la misma función[14] que en apartado anterior (AM-10,1.0) ya que el número de generaciones para ejecutar la búsqueda local son las mismas.

Busqueda Local

Esta función realiza lo mismo que la función BL[15] del AM-10,1.0 con la diferencia de que necesitamos ejecutar la búsqueda local solo sobre el 10 % de los valores. Por lo que lo único que tenemos que modificar es la condición de parada del bucle, que en vez de ser toda la población, sera hasta $0{,}1 \cdot$ *poblacion.size*(). En cada iteración genero un número aleatorio para seleccionar una posición

de la población para realizarle la busqueda local y la lanzo sobre ese cromosoma, así hasta llegar a la condición de parada indicada.

```
input  : poblacion,datosTraining[][],datosTest[][],K
output: poblacion[x_i[], ..., x_n[]],valores[x_i, ..., x_n]
valores ← ∅;
elementos ← 0,1 · poblacion.size();
for i ← 1 to elementos do
    pos ← next.Int(poblacion.size());
    pesos ← poblacion[pos];
    bl ← new BL(datosTraining,datosTest,pesos,K);
    poblacion[pos],valores[pos] ← bl.BL(&evaluaciones);
    if valores[pos] > Emejor then
        mejor ← poblacion[pos];
        Emejor ← valores[pos];
    end
    if Emejor > Ebestmejor then
        bestmejor ← mejor;
        Ebestmejor ← Emejor;
    end
end
return poblacion,valores
```

Algorithm 16: busquedaLocal

3.5. Algoritmo Memético (AM-10,0.1mej)

Como en los apartados anteriores, solo voy a definir los métodos que no hemos definido aún. Tanto la funcion *IniciarPoblacion*[2], *Seleccion*[4], *Evaluar*[3], *Cruce*[5][6], *Mutacion*[7] y *Reemplazo*[8] son exactamente iguales que en el AGG.

Algoritmo Memético (AM-10,0.1mej)

Para este algoritmo, utilizamos exactamente la misma función[14] que en apartado anterior (AM-10,1.0) ya que el número de generaciones para ejecutar la búsqueda local son las mismas.

Busqueda Local

Esta función realiza lo mismo que la función BL[16] del AM-10,0.1 con la diferencia de que necesitamos ejecutar la búsqueda local sobre los mejores, en vez de sobre un 10 % aleatorio, por lo que lo único que hago es tener la población almacenada en un contenedor externo como he explicado en el AGE y cuando ejecutamos esta función, realizamos las iteraciones pero en vez de en cada iteración generar un número aleatorio, únicamente tenemos que recorrer el contenedor ordenado y ejecutando la búsqueda local sobre cada uno de los cromosomas hasta llegar a la condición de

parada establecida e indicada en el apartando anterior.

input : ordenados[][],datosTraining[][],datosTest[][],K
output: poblacion[x_i[], ..., x_n[]],valores[x_i, ..., x_n]
poblacion ← ordenados;
valores ← ∅;
elementos ← 0,1 · *poblacion.size*();
for $i \leftarrow 1$ **to** *elementos* **do**
    pesos ← poblacion[i];
    bl ← new BL(datosTraining,datosTest,pesos,K);
    poblacion[i],valores[i] ← bl.BL(&evaluaciones);
    if *valores[i]* > *Emejor* **then**
        mejor ← poblacion[i];
        Emejor ← valores[i];
    end
    if *Emejor* > *Ebestmejor* **then**
        bestmejor ← mejor;
        Ebestmejor ← Emejor;
    end
end
return *poblacion,valores*

Algorithm 17: busquedaLocal

4. Descripción para ejecución

En mi caso, he creado un proyecto con NetBeans para esta practica, por lo que adjuntado a este PDF irá un zip con el proyecto completo, con el .jar para poder ejecutarlo.

El proyecto esta compuesto por los siguientes elementos:

- **Carpeta src/practica2MH**: Donde se encuentran todos los ficheros implementados.
- **Carpeta src/input**: En esta capeta tenemos los 3 ficheros de entrada de la práctica.
- **Carpeta src/output**: Donde se generan los ficheros de salida del programa con la siguiente estructura.

  Se genera un fichero con el nombre de "ficheroEntrada-Algoritmo-train/test-Particion.txt" donde podemos ver por el nombre a qué fichero de entrada pertenecen, el algoritmo utilizado, train o test, donde train indica que son datos de entrenamiento, es decir, devuelve el tiempo y el vector resultado. Y test, que indica que que son datos de test por lo que tendremos los porcentajes de acierto, etc... Por último vemos un numero que indica la partición utilizada como test, por lo que tendremos desde el 0 al 4.

  Para la **ejecución del programa**, debe descomprimir el zip que va adjuntado con la práctica y una vez descomprimido, moverse dentro de él. Podrá encontrar las distintas carpetas y un fichero llamado Practica2MH.jar. Abra un terminal y ejecute **java -jar Practica2MH.jar**. Comenzará la ejecución y cuando finalice, tendrá los ficheros de salida en **src/output**

5. Análisis de los resultados

5.1. Descripción

En este apartado discutiremos los distintos resultados obtenidos de clasificación. En mi caso, **la semilla utilizada** es 3395+(particion*3+fichero).

- **AGG-BLX/CA && AGE-BLX/CA**

  La poblacion estará compuesta por 30 cromosomas, cada gen de estos cromosomas está incializado aleatoriamente siguiendo una distribución uniforme en $[0, 1]$. El valor para $P_c = 0{,}7$ en el AGG y $P_c = 1$ en el AGE, el valor de $P_m = 0{,}001$ en ambos casos. Para la mutación utilizamos un valor aleatorio siguiendo una distribución gaussiana con media 0 y $\sigma = 0{,}3$. El operador de cruce BLX utiliza un $\alpha = 0{,}3$. Estos algoritmos nos devuelven un vector de pesos (el mejor visto en toda la ejecución) y ejecutamos el KNN sobre los datos de test utilizando este vector de pesos, los resultados de las tablas muestran los resultados de este test.

- **AM-X-Y**

  Los algoritmos meméticos los he realizado sobre AGG-BLX ya que es el que mejor resultado me ha dado de los dos AGG. Todos los datos de entrada son los mismos que para el AGG, a excepción de el tamaño de la población que en este caso es de 10 cromosomas. La condición de parada de la BL será de $2 \cdot n$, siendo n el número de genes de un cromosoma.

5.2. Resultados obtenidos

5.2.1. AGG-BLX

Tabla 1: Resultados obtenidos por el algoritmo AGG-BLX en el problema del APC

	Ozone				**Parkinson**				**Spectf-heart**			
	%_clas	**%_red**	**F**	**T**	**%_clas**	**%_red**	**F**	**T**	**%_clas**	**%_red**	**F**	**T**
Particion0	71,88	68,06	69,97	170,91	84,62	50,00	67,31	23,41	97,14	63,64	80,39	108,12
Particion1	84,38	56,94	70,66	180,60	76,92	54,55	65,73	23,52	92,86	63,64	78,25	127,06
Particion2	75,00	65,28	70,14	165,85	87,18	54,55	70,86	19,81	67,14	75,00	71,07	87,19
Particion3	84,38	65,28	74,83	174,59	71,79	59,09	65,44	22,07	77,14	65,91	71,53	123,18
Particion4	85,94	56,94	71,44	181,10	76,92	54,55	65,73	23,61	97,10	77,27	87,19	115,03
Media	80,31	62,50	71,41	174,61	79,49	54,55	67,02	22,48	86,28	69,09	77,68	112,11

Donde en mi caso, particion 1 esta compuesta por:

- **Train** → particiones 1,2,3,4.
- **Test** → particion 0.

Para la particion 2 se cambia el test por particion 1 y el train serían las demás. Y así sucesivamente. Esto se aplica a **todos los casos de todas las tablas**.

5.2.2. AGG-CA

Tabla 2: Resultados obtenidos por el algoritmo AGG-CA en el problema del APC

	Ozone				Parkinson				Spectf-heart			
	%_clas	%_red	F	T	%_clas	%_red	F	T	%_clas	%_red	F	T
Particion0	78,13	68,06	73,09	166,75	84,62	36,36	60,49	24,22	95,71	70,45	83,08	125,43
Particion1	84,38	52,78	68,58	188,21	79,49	40,91	60,20	22,23	88,57	61,36	74,97	98,08
Particion2	71,88	61,11	66,49	184,86	94,87	40,91	67,89	24,54	71,43	56,82	64,12	133,82
Particion3	79,69	54,17	66,93	189,40	58,97	45,45	52,21	23,46	74,29	72,73	73,51	121,58
Particion4	79,69	61,11	70,40	175,08	64,10	31,82	47,96	25,14	100,00	61,36	80,68	105,56
Media	78,75	59,44	69,10	180,86	76,41	39,09	57,75	23,92	86,00	64,55	75,27	116,89

5.2.3. AGE-BLX

Tabla 3: Resultados obtenidos por el algoritmo AGE-BLX en el problema del APC

	Ozone				Parkinson				Spectf-heart			
	%_clas	%_red	F	T	%_clas	%_red	F	T	%_clas	%_red	F	T
Particion0	79,69	45,83	62,76	193,61	71,79	63,64	67,72	23,13	98,57	52,27	75,42	129,53
Particion1	81,25	43,06	62,15	203,9	84,62	45,45	65,03	21,73	88,57	34,09	61,33	117,69
Particion2	76,56	34,72	55,64	212,01	97,44	63,64	80,54	22,25	74,29	27,27	50,78	127,48
Particion3	79,69	41,67	60,68	201,72	64,1	59,09	61,6	23,32	81,43	40,91	61,17	132,59
Particion4	81,25	33,33	57,29	213,69	74,36	63,64	69	22,94	97,1	38,64	67,87	137,27
Media	79,69	39,72	59,7	204,99	78,46	59,09	68,78	22,67	87,99	38,64	63,31	128,91

5.2.4. AGE-CA

Tabla 4: Resultados obtenidos por el algoritmo AGE-CA en el problema del APC

	Ozone				Parkinson				Spectf-heart			
	%_clas	%_red	F	T	%_clas	%_red	F	T	%_clas	%_red	F	T
Particion0	78,13	30,56	54,34	212,91	84,62	36,36	60,49	23,52	95,71	31,82	63,77	130,10
Particion1	82,81	29,17	55,99	227,69	82,05	40,91	61,48	24,28	90,00	29,55	59,77	126,63
Particion2	68,75	22,22	45,49	262,71	92,31	40,91	66,61	24,06	75,71	27,27	51,49	164,30
Particion3	76,56	26,39	51,48	236,23	58,97	45,45	52,21	23,41	81,43	40,91	61,17	130,72
Particion4	81,25	30,56	55,90	229,32	64,10	31,82	47,96	24,84	98,55	31,82	65,18	146,92
Media	77,50	27,78	52,64	233,77	76,41	39,09	57,75	24,02	88,28	32,27	60,28	139,74

5.2.5. AM-10,1.0

Tabla 5: Resultados obtenidos por el algoritmo AM-10,1.0 en el problema del APC

	Ozone				Parkinson				Spectf-heart			
	%_clas	%_red	F	T	%_clas	%_red	F	T	%_clas	%_red	F	T
Particion0	65,63	84,72	75,17	199,91	71,79	81,82	76,81	20,21	98,57	79,55	89,06	122,14
Particion1	75	70,83	72,92	197,05	74,36	90,91	82,63	19,75	92,86	72,73	82,79	122,34
Particion2	70,31	80,56	75,43	204,63	89,74	72,73	81,24	22,54	75,71	70,45	73,08	97,59
Particion3	75	70,83	72,92	195,45	64,1	90,91	77,51	19,66	82,86	77,27	80,06	105,01
Particion4	85,94	72,22	79,08	192,9	71,79	90,91	81,35	19,77	98,55	84,09	91,32	126,26
Media	74,38	75,83	75,1	197,99	74,36	85,45	79,91	20,38	89,71	76,82	83,26	114,67

5.2.6. AM-10,0.1

Tabla 6: Resultados obtenidos por el algoritmo AM-10,0.1 en el problema del APC

	Ozone				Parkinson				Spectf-heart			
	%_clas	%_red	F	T	%_clas	%_red	F	T	%_clas	%_red	F	T
Particion0	70,31	87,50	78,91	176,35	74,36	86,36	80,36	20,62	95,71	68,18	81,95	114,79
Particion1	81,25	75,00	78,13	172,82	71,79	90,91	81,35	18,41	88,57	65,91	77,24	110,28
Particion2	68,75	81,94	75,35	181,14	97,44	77,27	87,35	20,73	77,14	86,36	81,75	112,49
Particion3	73,44	77,78	75,61	183,53	71,79	90,91	81,35	17,65	81,43	81,82	81,62	119,05
Particion4	84,38	70,83	77,60	170,58	71,79	90,91	81,35	19,33	97,10	75,00	86,05	89,08
Media	75,63	78,61	77,12	176,88	77,44	87,27	82,35	19,35	87,99	75,45	81,72	109,14

5.2.7. AM-10,0.1mej

Tabla 7: Resultados obtenidos por el algoritmo AM-10,0.1mej en el problema del APC

	Ozone				Parkinson				Spectf-heart			
	%_clas	%_red	F	T	%_clas	%_red	F	T	%_clas	%_red	F	T
Particion0	71,88	86,11	78,99	187,79	74,36	81,82	78,09	21,70	98,57	68,18	83,38	116,06
Particion1	82,81	76,39	79,60	170,08	76,92	90,91	83,92	19,65	84,29	79,55	81,92	125,38
Particion2	78,13	80,56	79,34	183,35	87,18	81,82	84,50	21,28	65,71	75,00	70,36	98,86
Particion3	73,44	77,78	75,61	174,84	58,97	86,36	72,67	19,62	81,43	77,27	79,35	95,29
Particion4	75,00	75,00	75,00	181,33	71,79	90,91	81,35	16,96	98,55	77,27	87,91	118,53
Media	76,25	79,17	77,71	179,48	73,85	86,36	80,10	19,84	85,71	75,45	80,58	110,83

5.2.8. Extra: AM-1,0.1

Tabla 8: Resultados obtenidos por el algoritmo AM-1,0.1 en el problema del APC

	Ozone				Parkinson				Spectf-heart			
	%_clas	%_red	F	T	%_clas	%_red	F	T	%_clas	%_red	F	T
Particion0	71,88	86,11	78,99	201,28	74,36	81,82	78,09	20,46	95,71	75,00	85,36	111,19
Particion1	81,25	77,78	79,51	200,24	66,67	86,36	76,52	19,72	95,71	72,73	84,22	114,67
Particion2	67,19	79,17	73,18	195,60	89,74	86,36	88,05	16,00	80,00	84,09	82,05	128,15
Particion3	73,44	79,17	76,30	197,98	64,10	90,91	77,51	20,07	77,14	84,09	80,62	114,45
Particion4	78,13	75,00	76,56	200,66	76,92	90,91	83,92	19,18	97,10	93,18	95,14	116,10
Media	74,38	79,44	76,91	199,15	74,36	87,27	80,82	19,09	89,13	81,82	85,48	116,91

5.2.9. Extra: AM-1,0.1mej

Tabla 9: Resultados obtenidos por el algoritmo AM-1,0.1mej en el problema del APC

	Ozone				Parkinson				Spectf-heart			
	%_clas	%_red	F	T	%_clas	%_red	F	T	%_clas	%_red	F	T
Particion0	70,31	81,94	76,13	170,38	84,62	81,82	83,22	19,55	94,29	88,64	91,46	93,58
Particion1	87,50	83,33	85,42	208,14	82,05	90,91	86,48	17,75	88,57	77,27	82,92	91,88
Particion2	75,00	87,50	81,25	187,36	84,62	81,82	83,22	21,06	77,14	84,09	80,62	111,28
Particion3	75,00	81,94	78,47	199,45	74,36	81,82	78,09	20,99	78,57	70,45	74,51	124,53
Particion4	81,25	72,22	76,74	185,90	71,79	90,91	81,35	20,85	97,10	70,45	83,78	121,16
Media	77,81	81,39	79,60	190,25	79,49	85,45	82,47	20,04	87,13	78,18	82,66	108,49

5.2.10. Extra: AM-1,1.0

Tabla 10: Resultados obtenidos por el algoritmo AM-1,1.0 en el problema del APC

	Ozone				Parkinson				Spectf-heart			
	%_clas	%_red	F	T	%_clas	%_red	F	T	%_clas	%_red	F	T
Particion0	75,00	75,00	75,00	225,93	76,92	90,91	83,92	19,87	95,71	88,64	92,18	135,68
Particion1	81,25	73,61	77,43	224,15	82,05	90,91	86,48	20,29	91,43	81,82	86,62	109,68
Particion2	75,00	72,22	73,61	228,44	82,05	86,36	84,21	20,05	85,71	84,09	84,90	113,96
Particion3	76,56	66,67	71,61	217,24	66,67	90,91	78,79	21,45	75,71	81,82	78,77	141,06
Particion4	82,81	69,44	76,13	221,93	61,54	90,91	76,22	21,19	95,65	84,09	89,87	129,84
Media	78,13	71,39	74,76	223,54	73,85	90,00	81,92	20,57	88,84	84,09	86,47	126,04

5.2.11. Tabla resumen

Tabla 11: Resultados resumen en el problema del APC

	Ozone				Parkinson				Spectf-heart			
	%_clas	%_red	F	T	%_clas	%_red	F	T	%_clas	%_red	F	T
1NN	79,69	0,00	39,84	0,00	80,00	0,00	40,00	0,00	68,48	0,00	34,24	0,00
RELIEF	75,95	0,00	37,97	0,10	82,05	0,00	41,03	0,04	71,33	0,00	35,66	0,09
BL	76,88	80,00	78,84	61,75	76,41	82,73	79,57	1,75	79,38	79,09	79,24	26,40
AGG-BLX	80,31	62,50	71,41	174,61	79,49	54,55	67,02	22,48	86,28	69,09	77,68	112,11
AGG-CA	78,75	59,44	69,10	180,86	76,41	39,09	57,75	23,92	86,00	64,55	75,27	116,89
AGE-BLX	79,69	39,72	59,70	204,99	78,46	59,09	68,78	22,67	87,99	38,64	63,31	128,91
AGE-CA	77,50	27,78	52,64	233,77	76,41	39,09	57,75	24,02	88,28	32,27	60,28	139,74
AM-(10,1,0)	74,38	75,83	75,10	197,99	74,36	85,45	79,91	20,38	89,71	76,82	83,26	114,67
AM-(10,0,1)	75,63	78,61	77,12	176,88	77,44	87,27	82,35	19,35	87,99	75,45	81,72	109,14
AM-(10,0,1mej)	76,25	79,17	77,71	179,48	73,85	86,36	80,10	19,84	85,71	75,45	80,58	110,83
AM-(1,0,1)	74,38	79,44	76,91	199,15	74,36	87,27	80,82	19,09	89,13	81,82	85,48	116,91
AM-(1,0,1mej)	77,81	81,39	79,60	190,25	79,49	85,45	82,47	20,04	87,13	78,18	82,66	108,49
AM-(1,1,0)	78,13	71,39	74,76	223,54	73,85	90,00	81,92	20,57	88,84	84,09	86,47	126,04

A fin de hacer los datos un poco más visuales, voy a mostrar una tabla donde se clasifican por colores y por columnas, siendo el amarillo el peor resultado de la columna, el naranja un valor medio, y el rojo el mejor valor de la columna.

	Ozone				Parkinson				Spectf-heart			
	%_clas	%_red	F	T	%_clas	%_red	F	T	%_clas	%_red	F	T
1NN	79,69	0,00	39,84	0,00	80,00	0,00	40,00	0,00	68,48	0,00	34,24	0,00
RELIEF	75,95	0,00	37,97	0,10	82,05	0,00	41,03	0,04	71,33	0,00	35,66	0,09
BL	76,88	80,00	78,84	61,75	76,41	82,73	79,57	1,75	79,38	79,09	79,24	26,40
AGG-BLX	80,31	62,50	71,41	174,61	79,49	54,55	67,02	22,48	86,28	69,09	77,68	112,11
AGG-CA	78,75	59,44	69,10	180,86	76,41	39,09	57,75	23,92	86,00	64,55	75,27	116,89
AGE-BLX	79,69	39,72	59,70	204,99	78,46	59,09	68,78	22,67	87,99	38,64	63,31	128,91
AGE-CA	77,50	27,78	52,64	233,77	76,41	39,09	57,75	24,02	88,28	32,27	60,28	139,74
AM-(10,1,0)	74,38	75,83	75,10	197,99	74,36	85,45	79,91	20,38	89,71	76,82	83,26	114,67
AM-(10,0,1)	75,63	78,61	77,12	176,88	77,44	87,27	82,35	19,35	87,99	75,45	81,72	109,14
AM-(10,0,1mej)	76,25	79,17	77,71	179,48	73,85	86,36	80,10	19,84	85,71	75,45	80,58	110,83
AM-(1,0,1)	74,38	79,44	76,91	199,15	74,36	87,27	80,82	19,09	89,13	81,82	85,48	116,91
AM-(1,0,1mej)	77,81	81,39	79,60	190,25	79,49	85,45	82,47	20,04	87,13	78,18	82,66	108,49
AM-(1,1,0)	78,13	71,39	74,76	223,54	73,85	90,00	81,92	20,57	88,84	84,09	86,47	126,04

5.3. Análisis de resultados

Empezamos analizando la tabla 1 que se encuentra en la página 18 y que hace referencia al **algoritmo AGG-BLX**.

- Vemos en general que respecto a los algoritmos de la practica anterior (1NN,RELIEF,BL), obtenemos una mejora significativa gracias a la tasa_red respecto al 1NN y al RELIEF. Sin embargo, respecto a la búsqueda local obtenemos peores resultados. Si visualizamos los datos, podemos ver que en tasa_clas no obtenemos malos resultados ya que tenemos los mismos o superiores que en los algoritmos anteriores. Por lo que podemos deducir que los datos nos han permitido aprender con un procentaje medio bueno, aunque no suficiente.

  Respecto a los algoritmos desarrollados en esta practica, podemos ver que tanto en Ozone como en Parkinson es el que mejor clasifica aunque no por mucha diferencia con los demás, en

Spectf-heart no consigue el mejor resultado, pero se encuentra en la media. Esto nos dice que este algoritmo ha conseguido modificar los pesos de tal manera que la mayoría de etiquetas estén bien clasificadas. Aunque este no es nuestra meta final.

- El problema radica en tasa_red, al tener pocas evaluaciones y no tener una explotacion intensa ya que este algoritmo usa más la exploración que la explotación, no conseguimos reducir lo suficiente como por ejemplo consigue la BL. Esto hace obtener un valor bastante bajo en reducción y no conseguir lo que buscamos.

  Respecto a los algoritmos de esta practica, vemos que en reduccion, está en la media hacia abajo, esto quiere decir que no reduce lo suficiente, como he indicado anteriormente, no consigue converger y obtener buenas reducciones respecto a tasa_red.

- Vemos que el tiempo que necesita este algoritmo ahora sube considerablemente respecto a los algoritmos de la practica anterior. Es normal por la propia definición del agoritmo, necesita un procesamiento y un cómputo bastante más elevado que los algoritmos anteriores ya que estos son bastantes simples computacionalmente.

  Respecto a los algoritmos de esta práctica vemos que en general todos suelen tardar lo mismo aunque en elgunos haya una duración un poco más alta, tenemos unos tiempos que entre ellos no tienen unas diferencias significativas.

- El valor de la funcion objetivo, al tener $\alpha = 0{,}5$, sabiendo que este algoritmo no consigue reducir lo suficiente tasa_red, obtenemos un resultado demasiado bajo. Por lo que podemos concluir que este algoritmo no nos da unos resultados buenos ni incluso comparandolo con el algoritmo BL de la practica anterior.

Seguimos ahora analizando la tabla 2 que se encuentra en la página 19 y que hace referencia al **algoritmo AGG-CA**.

- Este es exactamente igual que el anterior, cambiando únicamente el operador de cruce que nos hace ahora la media aritmética entre dos cromosomas en vez del operador BLX.

- Vemos por los resultados que al cambiar el operador de cruce, obtenemos peores resultados. Esto claramente nos muestra que el operador CA le cuesta más crear nuevas soluciones de calidad respecto al BLX. La complejidad de los mismos nos dan una pista para saber que CA tiene pinta de por lo general darnos peores resultados ya que el BLX nos permite exploración aparte de explotación mientras que CA no nos permite esto ya que solo hace la media entre dos genes sin permitirnos salir de esos valores, por lo que limita un poco la exploración.

- El valor de la funcion objetivo, al tener $\alpha = 0{,}5$, sabiendo que este algoritmo empeora en todos los apartados respecto al algoritmo anterior, podemos concluir que este algoritmo nos ofrece una solución aún peor que la anterior. Insuficiente para obtener soluciones de calidad.

El siguiente en analizar será el **algoritmo AGE-BLX** que tenemos en la tabla 3 que se encuentra en la página 19.

- Ahora estamos en el mismo caso que en AGG-BLX pero cambiando el generacional por el estacionario. Esto es que cada población que vamos generando se creará con dos padres en vez de con el mismo número de padres que elementos tenga la problación.

- Nos encontramos con el mismo caso que para AGG-BLX, en el que conseguimos clasificar correctamente la mayor parte de los elementos, pero no conseguimos reducir apenas nada,

incluso menos que el AGG-BLX, esto se debe a que al tener un $P_m = 0{,}001$ y una población de padres de tamaño 2 y el número de genes no es muy elevado, no conseguimos mutar nunca ningún gen de los cromosomas que tiene la problación, lo que hace que perdamos diversificación. Por lo que no conseguimos reducir como debería.

- El valor de la funcion objetivo, al tener $\alpha = 0{,}5$, sabiendo que este algoritmo no consigue reducir lo suficiente tasa_red, obtenemos un resultado demasiado bajo. Incluso peor que AGG-BLX e incluso también peor que AGG-CA. Por lo que podemos concluir que este algoritmo no nos da unos resultados buenos ni conseguimos mejorar lo que ya habiamos conseguido con la practica anterior.

Pasamos ahora a analizar el **algoritmo AGE-CA** que tenemos en la tabla 4 que se encuentra en la página 19.

- Ahora estamos en el mismo caso AGE-BLX pero cambiando el operador de cruce como ya he indicado en el paso de AGG-BLX a AGG-CA.
- Volvemos a obtener los mismos resultados que cuando pasabamos de AGG-BLC a ABB-CA, podemos ver por las tablas que este operador nos da peores resultados que el operador BLX, aunque en Spectf-heart consigue clasificar mínimamente mejor, generalmente en todos los casos, tenemos peores resultados que el AGE-BLX.
- El valor de la funcion objetivo, al tener $\alpha = 0{,}5$, como no obtenemos ni mucho menos resultados buenos, concluimos que este algoritmo no conseigue cumplir con los requisitos que nosotros queremos y seguimos sin encontrar un algoritmo que nos realice alguno mejora sobre los algoritmos de la practica 1.

Seguimos ahora analizando la tabla 5 que se encuentra en la página 20 y que hace referencia al **algoritmo AM-10,1.0**.

- Ahora si realizamos cambios sobre los algoritmos anteriores. Vemos que estos algoritmos tienen una exploración amplia pero poca potencia de exploración y más aún con las condiciones que tenemos de parada. Es por ello que ahora utilizamos la busqueda local utilizada en la practica 1 para mejorar la explotación del agoritmo genético. Como el propio nombre indica y como he comentado en apartados anteriores, realizamos la busqueda local cada 10 generaciones y a todos los elementos de la población.
- Podemos ver como ahora, tenemos una tasa_clas que ha disminuido respecto a lo que ya teníamos anteriormente a excepción de Spectf-heart que ha aumentado considerablemente. En este algoritmo dedicamos tiempo tanto a la exploracion como a la explotación y con los requisitos que nos piden, en Ozone y en Parkinson no consigue explorar suficiente para encontrar una tasa_clas tan elevada como en los caso anteriores.
- La mejora respecto al resto la tenemos en tasa_red, como vemos hemos aumentado bastante este valor respecto a los anteriores, lo que hace que nuestra funcion objetivo crezca y empecemos a ver resultados algo mas satisfactorios.
- El valor de la funcion objetivo, al tener $\alpha = 0{,}5$, conseguimos obtener una media considerable, ya que con este algoritmo conseguimos mejorar a los algoritmos de la practica 1, y como consecuencia, a todos los vistos hasta ahora de esta practica. Por lo que podemos concluir que obtenemos unos resultados razonables aunque en Ozone no sean unos valores muy buenos,

pero en media, conseguimos un algoritmo que mejora y nos da unos valores con algo de calidad.

Seguimos ahora analizando la tabla 6 que se encuentra en la página 20 y que hace referencia al **algoritmo AM-10,0.1**.

- Realizamos ahora un cambio respecto al AM-10,1.0 anterior, y es que en vez de cada 10 generaciones realizar una busqueda local sobre toda la poblacion, la realizamos únicamente sobre un 10 % aleatorio de la poblacion. Por lo que estamos dando mas evaluaciones a la exploración que a la explotación respecto al algoritmo anterior.
- Vemos que aunque seguimos teniendo unos valores bajos de tasa_clas, conseguimos en media mejorar a los del algoritmo anterior. Esto nos dice que gracias a la exploración hemos conseguido encontrar un cromosoma con mejor acierto para la clasificación.
- Al igual que con tasa_clas, en tasa_red hemos mejorado en media de nuevo, lo que esto nos lleva a obtener un valor de la función objetivo mejor que en el algoritmo anterior en media de los distintos ficheros ya que para Spectf-heart, al eliminar explotación hemos empeorado mínimamente.
- El valor de la funcion objetivo, al tener $\alpha = 0{,}5$, conseguimos obtener una media considerable, mejorando el algorimo anterior. Comenzamos a ver resultados considerablemente buenos aunque aún no hemos conseguido mejorar a la BL de la practica 1 para el fichero Ozone, en media, tenemos claramente mejor resultado con este algoritmo. Por lo que concluimos que para la función objetivo que tenemos, este algoritmo nos da unos resultados aceptables para los conjuntos de datos utilizados.

Pasamos ahora a analizar el **algoritmo AM-10,0.1mej** que tenemos en la tabla 7 que se encuentra en la página 20.

- La diferencia con el anterior ahora, es que la busqueda local la hacemos sobre el 10 % de los mejores en vez de sobre aleatorios.
- Para tasa_clas en media, empeoramos un poco al anterior, aunque mejora mínimante en Ozone, no es una mejora importante y en los otros dos ficheros empeora, aunque también es un empeoramiento minúsculo.
- Al igual que con tasa_clas, en tasa_red hemos empeorado en media de nuevo, lo que esto nos lleva a obtener un valor de la función objetivo peor que en el algoritmo anterior en media de los distintos ficheros ya que para Ozone, hemos mejorado mínimamente.
- El valor de la funcion objetivo, al tener $\alpha = 0{,}5$, conseguimos obtener una media aceptable, aunque empeorando al anterior, por lo que podemos decir que aunque este algoritmo se asemeja al anterior, seleccionando los mejores no conseguimos mejorarlo.

Ahora voy a resumir las valoraciones sobre **los 3 extras** realizados sobre el resto y por último terminaré con una breve conclusión. Podemos visualizar las tablas de los extras a partir en la página 21.

- La principal diferencia con los anteriores ahora, es cada cuantas generaciones usamos la búsqueda local, ya que en los 3 lo que hago es utilizar la búsqueda local por cada generación, lo que aumenta la explotación aunque perdamos algo de exploración. He realizado la modi-

ficación sobre *AM-10,0.1* , *AM-10,0.1mej* y *AM-10,1.0*, cambiando como acabo de indicar, el número de generaciones necesarias para realizar la busqueda local a 1.

- Vemos que en tasa_clas por lo general seguimos teniendo valores bajos respecto a algoritmos vistos anteriormente, a excepcion de Spectf-heart que hemos visto que cuando metemos explotación conseguimos mejorar la tasa de acierto bastante. No obstante, conseguimos a excepción del Parkinson, unos buenos valores.
- En tasa_red tenemos la verdadera mejora, respecto a los algoritmos anteriores conseguimos subir estos datos, y aunque tengamos unos valores mas bajos en clasificación, conseguimos en media hacer unos resultados bastante buenos, incluso consiguiendo el mejor algoritmo respecto a resultados para esta función objetivo y este conjunto de datos.
- El valor de la funcion objetivo, al tener $\alpha = 0{,}5$, conseguimos mejorar respecto a todos los anteriores, incluso hemos conseguido por fin mejorar con AM(1,0.1mej) a la BL de la practica 1 en el fichero Ozone, consiguiendo en Parkinson el mejor resultado y aunque en Spectf-heart no consiga el mejor resultado, en media tenemos el mejor algoritmo hasta el momento para este problema con los requisitos que nos ponen en el ejercicio. Comentar que aunque este sea el mejor, cualquiera de los 3 nos da un resultado bueno y aunque no sea el ideal, podemos ver que son resultados que empiezan a tener síntomas de soluciones con calidad.

Como conclusión final, observando la rabla resumen[11], hemos avanzado progresivamente, obteniendo resultados más malos en 1NN,RELIEF y BL, consiguiendo una media mejor con los Genéticos y consiguiendo mejorarlo aún más con los Meméticos. También podemos deducir por los resultados, que tanto Ozone como Parkinson deben de tener en el conjunto de características una relevancia entre ellas similar, ya que al eliminar muchas características, empeoramos bastante la tasa de acierto, sin embargo, en Spectf-heart podemos ver que tiene bastantes características que podemos llamar "inútiles", ya que eliminándolas, conseguimos seguir haciendo un buen porcentaje de acierto.

Todas las conclusiones se pueden corroborar a través de la tabla 11 en la página 22 donde se podrá ver las distintas medias para cada uno de los algoritmos utilizados y comprobar todos los datos obtenidos.