PROBLEMA DE APRENDIZAJE DE PESOS EN CARACTERÍSTICAS
ALGORITMOS GENÉTICOS GENERACIONALES, ESTACIONARIOS Y MEMÉTICOS
MEMORIA P2 METAHEURÍSTICAS

Ignacio Garach Vélez MH Viernes 17:30

20 de mayo de 2022

Índice

1. Introducción

En estas prácticas vamos a abordar la implementación y aplicación de varios algoritmos para el cálculo de pesos de calidad de un clasificador 1-NN de modo que se optimice tanto la tasa de acierto del problema de clasificación, como la simplicidad del clasificador, esto es, la detección de que características no influyen de forma determinante en la labor de clasificación y por tanto podrían no tenerse en cuenta.
Como preliminares introducimos brevemente el problema de clasificación. Se trata de construir un modelo que a partir de las características (datos descritos numéricamente) de un objeto sea capaz de predecir su clase (tipo de objeto entre un número finito de posibilidades). Para ello se entrena al modelo con un conjunto de datos correctamente clasificados (aprendizaje supervisado) y despues se prueba su valía con un conjunto independiente de test. Existen numerosos enfoques para resolver este problema con gran éxito, nosotros nos vamos a centrar en probablemente el más sencillo, se trata del algoritmo $1 - NN$ conocido popularmente como el del vecino más cercano.

Este algoritmo, simplemente almacena los datos de entrenamiento en memoria junto con su clase asociada, matemáticamente esto se puede representar como un conjunto de puntos de $\mathbf{R}^n$ junto con un entero representando su clase asociada. Y para cada punto que quiera clasificar, calcula la distancia euclídea con todos los puntos de entrenamiento y escoge la clase del punto para el cuál es mínima, su vecino más cercano.
Nuestro trabajo va a consistir en conseguir pesos que ponderen la importancia en la distancia final de cada característica en cada caso concreto de entrenamiento, es decir si la distancia euclídea usual es:

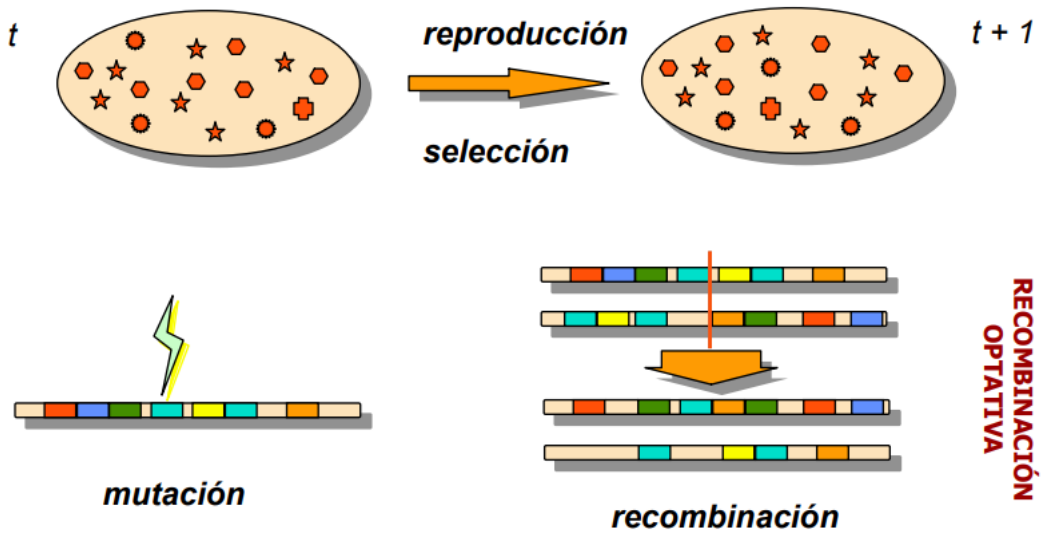
$$d(x, y) = \sqrt{\sum_{i=1}^{n} (x_i - y_i)^2}$$

nosotros consideramos el cálculo de los w_i tal que sea:

$$d(x, y) = \sqrt{\sum_{i=1}^{n} w_i (x_i - y_i)^2}$$

Nuestro objetivo es dual, por un lado queremos conseguir buenas tasas de acierto en el problema de clasificación y por otro obtener pesos bajos en las características menos influyentes para la clasificación. El problema de aprendizaje de pesos en características es determinar dichos pesos a partir del conjunto de datos de entrenamiento.
En la primera práctica se implementaron un algoritmo de búsqueda local con trayectorias simples y un algoritmo greedy (voraz) y se comparó su desempeño con el algoritmo 1-NN habitual.

En esta segunda práctica vamos a explorar diferentes técnicas de búsqueda basadas en poblaciones. Por un lado se implementarán dos versiones de un algoritmo genético generacional (AGGs) que diferiran en el operador de cruce usado y otras dos versiones de un algoritmo genético estacionario (AGEs) variando de nuevo en la forma de combinarse. Valorando los resultados se escogerá el AGG con mejor desempeño y se implementaran 3 versiones de algoritmos meméticos hibridándolo con una búsqueda local ligera que se realiza en algunas generaciones.



Esquema de reemplazamiento en un algoritmo genético

2. Datasets considerados

Para poder probar los algoritmos desarrollados se utilizarán los siguientes conjuntos de datos, todos ellos con clase binaria y algunos de ellos con distribución de clases no homogénea.

2.1. Ionosphere

Conjunto de datos recogidos en Goose Bay por un sistema con 16 antenas de alta frecuencia, el objetivo era captar electrones libres en la ionosfera, una devolución positiva indicaba la existencia de algún tipo de estructura. Los datos son las partes reales e imaginarias de 17 señales electromagnéticas complejas.

- 352 instancias
- 34 características
- 2 clases

2.2. Parkinsons

Conjunto de datos recogidos por el Centro Nacional de estudio de la voz de Oxford. Se trata de medidas asociadas a 195 grabaciones de voz de personas, algunas de ellas con enfermedad de Parkinson. La clase sería la presencia de dicha enfermedad.

- 195 instancias
- 22 características
- 2 clases

2.3. Spectf-Heart

Conjunto de datos de medidas asociadas a imágenes tomadas por tomografía computerizada cardíaca tomadas por el colegio de médicos de Ohio. Cada paciente se clasifica entre normal o con anomalías.

- 267 instancias
- 44 características
- 2 clases

3. Elementos comunes del problema

En este apartado vamos a describir las funciones que serán comunes durante todo el desarrollo de las prácticas de la asignatura:

- Para representar nuestros conjuntos de entrenamiento y test utilizaremos un vector de la STL de una nueva clase a la que llamaremos Instance que representará una instancia del problema, es decir su vector de características y su clase.
- Los datos proporcionados se encuentran en formato ARFF, un conjunto de cabeceras indicando las características y a continuación cada instancia es una fila con espacios como separadores y con la clase al final de la línea. Esto facilita su lectura para su posterior postprocesado. Hemos desarrollado para ello la funcion *read_arff* que recibe como parámetro el nombre del archivo y un par de vectores donde se almacenan las características y su clase. A posteriori en la función principal se genera el *vector de Instance* y se rellena con dichos datos.
- Los datos requieren ser normalizados para que no influyan las distintas escalas en nuestros algoritmos y no distorsionen los datos. Para esto se ha implementado la función *normalize* que modifica el vector de instancias, mediante escalado y translaciones hasta el intervalo $[0, 1]$, para ello se hace uso de la siguiente fórmula en cada característica:

$$x_i = \frac{x_i - mín(x_i)}{máx(x_i) - mín(x_i)}$$

- Para el cálculo de las distancias se implementa la función *weightedDistance* que recibe un vector de pesos y 2 vectores de características y devuelve la distancia ponderando por sus pesos.
- Para nuestro problema de aprendizaje de pesos, utilizaremos una versión del clasificador 1-NN que permitirá su generalización con la utilización de pesos. Recibe un valor booleano indicando si debe utilizar un vector de pesos al calcular las distancias. Atendiendo a estas características tendríamos el siguiente pseudocódigo:

Clasificador 1-NN (Instance, TrainingSet, Weighted?, Weights[])

Si no weighted
Entonces: Inicializar a 1 todos los pesos W
Para toda instancia de TrainingSet distinta de Instance (Leave One Out)
Hacer:

Calcular la distancia con respecto a Instance.
Si la distancia disminuye respecto al anterior guardamos su posición en pos.

Fin - Para todo

Devolver la clase de TrainingSet[pos]

3.1. Métricas utilizadas

Como medidas de la bondad de funcionamiento de los algoritmo se van a tomar las siguientes 4, relacionadas con los parámetros que se quieren optimizar, a saber, exactitud de clasificación, simplicidad y tiempo:

%**class** : Porcentaje de acierto en la labor de clasificación. Se utiliza el método de prueba basado en entrenamiento y test de 5 particiones cruzadas (se crean mediante la función *makeKFolds* inspirada en las prácticas de la asignatura de aprendizaje automático y que divide de forma homogénea y equilibrada las clases) y se realiza la media.

$$\%_{class} = 100\frac{\text{nº de instancias clasificadas correctamente en Test}}{\text{nº de instancias en Test}}$$

%**red** : Porcentaje de características que podrían obviarse, vamos a considerar aquellas cuyo peso asociado sea menor a 0.1. En efecto, para el algoritmo base no se reduce nada, porque todos los pesos son 1.

$$\%_{red} = 100\frac{\text{nº de características con peso menor que 0.1}}{\text{nº de características}}$$

Agregado : Valor de la función objetivo de nuestro problema. Se trata del promedio ponderado de las tasas de acierto y reducción, en nuestro caso consideramos una ponderación equiparada luego es simplemente la media de ambos valores

Tiempo : Tiempo de ejecución del algoritmo. En nuestro caso la media de 5 ejecuciones.

Para la implementación del cálculo de la tasa de clasificación se utilizará la función *score*, simplemente recorre el vector de predicciones del clasificador y cuenta los aciertos, devolviendo la división entre el número de instancias.
Para el cálculo de la tasa de reducción se hace algo análogo en *reduction* con el vector de pesos obtenido contando todos aquellos menores que 0.1 y devolviendo la división entre el número de características.
La función objetivo agregada se implementa de forma génerica con un parámetro α que pondera la importancia de precisión y simplicidad, aunque en nuestro caso usaremos $\alpha = 0.5$ luego coincidirá con la media de *score* y *reduction*, esto se realiza en *funcionObjetivo*.
Por último para el cálculo de tiempos se usará la librería *chrono* de C++ tomando el tiempo actual antes y después de las ejecuciones con *high_resolution_clock* y calculando su diferencia. Lo mediremos en segundos.

4. Algoritmos genéticos

Los algoritmos genéticos son técnicas de programación bioinspiradas en la reproducción de los seres vivos que se utilizan como estrategias para resolver problemas de optimización. La idea básica es hacer evolucionar una población de individuos que representa a un grupo de soluciones sometiendola a recombinaciones de cruce, mutaciones aleatorias, procesos de pervivencia del más fuerte, etc... Como breve nota histórica cabe destacar al profesor John Holland de la Universidad de Michigan al que se considera inventor de dichos algoritmos en los años 70. A partir de entonces comenzaron a utilizarse con gran éxito tanto para problemas teóricos matemáticos de alta dificultad computacional como para problemas mucho más concretos de ingeniería pero en los que era muy complicado obtener el óptimo. En esta práctica trataremos de adaptar varios de ellos a nuestro problema de aprendizaje de pesos. En primer lugar estableceremos nuevas estructuras de datos y métodos que nos permitan tratar de forma eficiente y eficaz los algoritmos:

- Vamos a almacenar nuestro vector de pesos solución junto con su valor de la función objetivo (fitness) en una estructura que llamaremos Cromosoma, de esta manera tendremos ligada claramente una solución con su calidad y además sabremos si dicha solución está evaluada para no volver a evaluarla ahorrando así tiempos de cómputo.

- Se ha definido una nueva función *computeFitness* para calcular el fitness de un cromosoma, simplemente se le pasa el conjunto de entrenamiento y el propio cromosoma por referencia cálculando su fitness mediante Leave-One-Out y con las funciones de tasa de acierto y reducción de la práctica 1.

- Por otro lado, creamos un comparador de Cromosomas en función de su fitness que será útil para mantenerlos ordenados en su nueva estructura de datos.

- Finalmente consideramos un multiconjunto para representar a la población de cromosomas en cada generación. La elección de este contenedor tiene muchas ventajas, por un lado la ordenación y admisión de repetidos, es posible que en nuestra población en algún momento tengamos 2 elementos iguales. El orden interno del multiset que construimos con nuestro comparador de cromosomas almacena siempre en último lugar al cromosoma con mejor valor de fitness y al peor en primer lugar. De esta forma tenemos garantizado un fácil acceso a los mismos.

- En el comienzo de una ejecución de un algoritmo genético siempre hay que inicializar la población, para ello utilizamos el método *launchPopulation* que recibe como argumentos el conjunto de entrenamiento, el número de individuos que ha de tener la población y el número de genes de cada individuo. Los genes se inicializan de forma aleatoria.

LaunchPopulation (TrainingSet, Nº Individuos, Nº Genes, Poblacion)

```
Inicializar  |Nº Individuos| cromosomas como sigue
Hacer:
    Declarar cromosoma y redimensionar su vector de pesos a |Nº Genes|.
    Para cada gen en el cromosoma asignar valor aleatorio en (0, 1).
    Evaluar Cromosoma con computeFitness(TrainingSet, Cromosoma)
    Añadir Cromosoma evaluado a la población.
En Población queda inicializada la misma.
```

4.1. Operadores comunes

Describiremos aquí los operadores que son comunes a todos los algoritmos genéticos o a algunos de ellos y por extensión a los meméticos. En general el esquema de un algoritmo genético es simple, se inicializa y evalua la población, se seleccionan individuos atendiendo a algún criterio, se recombinan

y mutan con cierta probabilidad y finalmente se crea una nueva población a partir de los nuevos individuos de manera que se mejore a la anterior, o como mínimo se iguale aumentando la diversidad. Este sistema se ha probado empíricamente que es provechoso para diversas labores de optimización. Pasamos a definir como funcionará cada operador:

4.1.1. Selección por torneo binario

El operador de selección es muy simple, se seleccionan 2 individuos al azar de la población y se enfrentan comparando su valor de fitness, el mejor se selecciona. Usaremos el método advance para mover el iterador tantas veces como indique un número aleatorio.

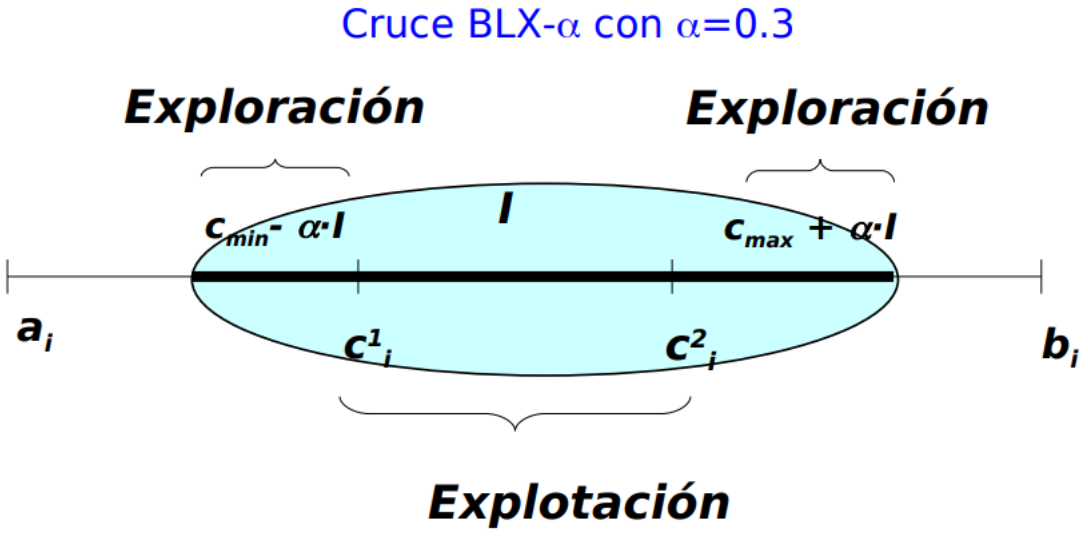
selectByTournament(Poblacion)

```
aleatorio = Ramdom_Int(0, |Poblacion|-1)
p1 = advance(Poblacion.begin(), aleatorio)
aleatorio = Ramdom_Int(0, |Poblacion|-1)
p2 = advance(Poblacion.begin(), aleatorio)

if p1->fitness > p2->fitness return *p1
else return *p2
```

4.1.2. Cruce BLX-α

El operador de cruce BLX recibe como argumentos dos padres y devuelve dos hijos. Los genes de los hijos son generados aleatoriamente en un intervalo determinado por los genes de los padres. Este intervalo depende del parámetro α que nosotros tomaremos como 0.3. Se genera tomando los mínimos y máximos de los valores de cada peso en ambos padres y se amplía en ambas direcciones con tamaño producto de la longitud del intervalo de los padres por α, se muestra en la siguiente figura tomada de las diapositivas de teoría:



Se observa muy bien el compromiso exploración-explotación, lo habitual será explotar el espacio de soluciones en el que se encontraban los padres pero también exploraremos valores cercanos fuera del mismo que pueden ser prometedores y aportan una mayor diversidad. Este operador se utilizará en una versión del algoritmo generacional y en otra del estacionario, dependiendo del algoritmo se utilizará más o menos veces.

alphaBLX(Cromosoma1, Cromosoma2)

```
Inicializar 2 cromosomas hijos h1 y h2.
Para cada peso w[i] en los cromosomas de los padres hacer:
    cmin = Mínimo de Cromosoma1.w[i] y Cromosoma2.w[i]
    cmax = Máximo de Cromosoma1.w[i] y Cromosoma2.w[i]
    I = cmax-cmin
    h1.w[i] = Aleatorio en el intervalo (cmin - alpha*I, cmax + alpha*I)
    h2.w[i] = Aleatorio en el intervalo (cmin - alpha*I, cmax + alpha*I)
    Si los pesos hijos han salido de (0, 1) se truncan a 0 o a 1.
Return pair<h1, h2>
```

4.1.3. Cruce aritmético

El operador de cruce aritmético recibe como argumentos dos padres y devuelve un hijo. Los genes de los hijos son generados cada uno como la media aritmética de los genes de los padres. Este operador es muy sencillo pero trata de generar soluciones promedio a partir de 2 soluciones buenas esperando obtener alguna prometedora. De nuevo se implementará una variante generacional y otra estacionaria con este operador.

arithmeticCross(Cromosoma1, Cromosoma2)

```
Inicializar cromosoma hijo h.
Para cada peso w[i] en los cromosomas de los padres hacer:
    h.w[i] = Promedio de Cromosoma1.w[i] y Cromosoma2.w[i]
Return h
```

4.1.4. Mutación normal

Misma mutación que se usaba en la búsqueda local para generar vecinos. Se suma a cierto gen aleatorio un valor de una distribución normal de media 0 y varianza 0.3. Puede darse el caso de tener que truncar los pesos como ocurría con el cruce BLX.

mutateWeight(Cromosoma, gen)

```
Cromosoma.w[gen] += Normal(0, 0.3)
Cromosoma.fitness = -1 indica la necesidad de recalcularla
Si el peso ha salido de (0, 1) se trunca a 0 o a 1.

Return Cromosoma
```

4.2. Variantes generacionales

En esta sección se detalla el pseudocódigo de los algoritmos generacionales, la única variación entre AGG-BLX y AGG-CA es el operador de cruce usado, en consecuencia como BLX genera 2 hijos cada vez y CA sólo uno, habrá que hacer el doble de torneos en este segundo caso.

Como indicación de los parámetros se utilizó una población de 30 cromosomas con probabilidad de cruce 0.7 y de mutación 0.1, el criterio de parada será efectuar 15000 evaluaciones de la función objetivo. Para evitar coste computacional de la generación de números aleatorios se genera al principio del algoritmo la cantidad de cruces y mutaciones esperadas mediante la esperanza matemática.

AGG(TrainingSet, Pesos)

```
Evaluaciones = 0
Generación = 1
Nº Genes = |Pesos| * |Población|
Nº Cruces = Prob[Cruce] * |Población| / 2

launchPopulation(p, 30, |Pesos|, TrainingSet) //Inicializar Población
Evaluaciones = 30

Mientras Evaluaciones < 15000:
    PoblacionIntermedia = vector<Cromosoma>
    MejorSolución = Mejor Cromosoma de Población Antigua

    Intermedia = SelectByTournament(Población) 30 veces

    Intermedia = Hacer Nº Cruces veces CruceBLX o
              Ó Hacer 2|Nº Cruces| veces Aritmético

    Calcular mutaciones esperadas y tantas como haya hacer:
        Mutar Gen Aleatorio en Cromosoma Aleatorio (No Repetir)

    Calcular fitness de los genes no evaluados
    Aumentar Evaluaciones en Consecuencia.
    Insertar Intermedia en Población Nueva

    Si el mejor de Nueva es peor que el mejor de Población:
        Poner al antiguo mejor donde el nuevo peor.

    Población = Nueva
    Generación++
Fin-Mientras

Almacenar en Pesos los del mejor Cromosoma

return Generación
```

4.3. Variantes estacionarias

Estas variantes generan poblaciones intermedias mucho más reducidas, realmente habrá 2 nuevos individuos trás el cruce (que será obligatorio) y la mutación. En la selección el cruce BLX sólo requiere 2 individuos mientras que el aritmético necesita 4. La población general es de 30 individuos de nuevo y trás la reproducción compiten por estar en la nueva población los 2 peores de la población y los 2 nuevos quedándose los 2 mejores de entre ellos.

AGE(TrainingSet, Pesos)

```
Evaluaciones = 0
Generación = 1
Nº Genes = |Pesos| * |Población|
Nº Cruces = |Intermedia| / 2
Nº Mutaciones = P[Mutación] * 2 * Nº Genes

launchPopulation(p, 30, |Pesos|, TrainingSet) //Inicializar Población
Evaluaciones = 30

Mientras Evaluaciones < 15000:
    PoblacionIntermedia = vector<Cromosoma>
    MejorSolución = Mejor Cromosoma de Población Antigua

    Intermedia = SelectByTournament(Población) 2 veces(BLX)
              Ó SelectByTournament(Población) 4 veces(CA)
    Intermedia = Hacer Nº Cruces veces CruceBLX o
              Ó Hacer 2|Nº Cruces| veces Aritmético

    Calcular mutaciones esperadas y tantas como haya hacer:
        Mutar Gen Aleatorio en Cromosoma Aleatorio (No Repetir)

    Calcular fitness de los genes no evaluados
    Aumentar Evaluaciones en Consecuencia.
    Insertar Intermedia en Población Nueva

    Si los 2 hijos son mejores que los 2 peores de la antigua ponlación:
        Sustituir en Población los 2 peores por los 2 nuevos.
    Sino:
        Si el mejor nuevo es mejor que el peor antiguo:
            Sustituirlo

    Generación++
Fin-Mientras

Almacenar en Pesos los del mejor Cromosoma

return Generación
```

5. Algoritmos meméticos

Los algoritmos meméticos son técnicas de optimización que combinan de forma sinérgica elementos tomados de otras metaheurísticas junto con técnicas de búsqueda local. Son un avance en la mejora de calidad de soluciones pues combinan a los algoritmos de búsqueda local que son buenos explotadores y malos exploradores (recorren muy bien una zona del espacio de soluciones pero pueden estancarse en óptimos locales) con los algoritmos evolutivos que son buenos exploradores pero malos explotadores por su factor aleatorio. De esta manera conseguimos mezclar lo mejor de ambos mundos obteniendo una exploración muy homogénea del espacio de búsqueda.

En nuestro caso vamos a hibridar la búsqueda local con el algoritmo generacional que mejor resultado ha obtenido sobre nuestros datos, el cuál coincide en los tres datasets y es el que utiliza cruce BLX.

Se realizará un estudio de tres versiones en las cuáles se aplicará la búsqueda local cada 10 generaciones y variando desde que cromosomas se inicia la misma:

- AM-(10, 1.0): Se lanza la búsqueda desde todos los cromosomas de la población.
- AM-(10, 0.1): Se lanza la búsqueda desde un 10 por ciento de los cromosomas de la población elegidos al azar.
- AM-(10, 0.1M): Se lanza la búsqueda desde el 10 por ciento de mejores cromosomas de la población.

Las poblaciones tendrán 10 cromosomas en estos algoritmos y se mantienen el resto de parámetros de los algoritmos genéticos generacionales. En general el pseudocódigo es muy parecido al de AGG, sólo varía como se escogen los cromosomas a explorar con la búsqueda.

AM(TrainingSet, Pesos)

```
Ejecutar algoritmo AGG-BLX
Cada 10 generaciones hacer:
    Aplicar búsqueda local ligera a todos los cromosomas
                        Ó  al 10 por ciento aleatorio de ellos
                        Ó  al 10 por ciento mejor de ellos
    y almacenar en cada uno el mejor obtenido a partir de él.
    Aumentar Evaluaciones en consecuencia.
    Insertar en la nueva población.
    Avanzar de generación.

Terminar por el mismo criterio que AGG-BLX.
```

Para la búsqueda local utilizada se implementa la siguiente versión ligera.

5.1. Búsqueda local ligera

Debemos adaptar la búsqueda para las nuevas estructuras de datos que hemos usado en esta práctica, además solo exploraremos 2n vecinos por cada cromosoma siendo n el tamaño del mismo. Se tendría entonces el siguiente pseudocódigo:

lightLocalSearch(TrainingSet[], Cromosoma)

```
Evaluaciones = 0
ind = Mezclar índices de cada gen para la aleatoriedad
MejorSolución = Cromosoma.fitness

Mientras Evaluaciones < 2 * Nº Genes
    ComponenteAMutar = ind[Evaluaciones Mod Nº Genes]
    CromosomaMutado = Cromosoma
    CromosomaMutado[ComponenteAMutar] += Normal(0, 0.3)
    Si el peso ha salido de (0, 1) se trunca a 0 o a 1.

    Calcular fitness de CromosomaMutado
    Evaluaciones++

    Si el CromosomaMutado es mejor que MejorSolución:
        Cromosoma = CromosomaMutado
        MejorSolución = CromosomaMutado.fitness

    Si Evaluaciones Mod Nº Genes == 0
        ind = Mezclar índices de cada gen para la aleatoriedad
Fin-Mientras
En Cromosoma queda almacenado el mejor encontrado.
Return Evaluaciones para el criterio de parada del memético
```

6. Implementación y guía de usuario

Para la implementación se ha utilizado el lenguaje C++, las librerías habituales de entrada y salida y lectura de fichero, toma de tiempos y funciones auxiliares. Se usó un generador de números aleatorios inicializado con la semilla 2022.

Se ha especificado por secciones la estructura en funciones del código, por simplicidad se ha optado por implementar todo en un mismo archivo sin usar cabeceras.

Se utiliza optimización en la compilación con g++, concretamente se usa la siguiente orden para compilar:

g++ -std=c++11 -O3 main.cpp

Para ejecutar se llama en terminal a ./a.out seguido de la semilla que se desee.

7. Análisis y presentación de resultados

Recordemos en primer lugar los resultados obtenidos en la práctica 1:

7.1. 1NN, Relief y Búsqueda Local

Trás la ejecución de los algoritmos, encontramos que era preocupante el tiempo de ejecución del algoritmo de búsqueda local, pues se elevaban a 30 minutos todas las ejecuciones. Sin embargo añadiendo optimización O3 al compilar lo reducimos en torno a los 5 minutos totales de ejecución. Se obtienen entonces los siguientes resultados, las primeras 5 filas son sobre las particiones y la sexta es la media:

Cuadro 1: Cuadro de resultados para 1-NN

Ionosphere				**Parkinsons**				**Spectf-heart**			
%cl	***%red***	***Agr.***	**T**	***%cl***	***%red***	***Agr.***	**T**	***%cl***	***%red***	***Agr.***	**T**
84.51	0.00	42.25	0.003	95.00	0.00	47.50	0.00	58.57	0.00	29.29	0.002
87.32	0.00	43.66	0.003	92.50	0.00	46.25	0.00	71.43	0.00	35.71	0.002
87.32	0.00	43.66	0.003	95.00	0.00	47.50	0.00	70.00	0.00	35.00	0.002
91.55	0.00	45.77	0.001	97.50	0.00	48.75	0.00	65.71	0.00	32.86	0.002
80.60	0.00	40.30	0.001	100.00	0.00	50.00	0.00	63.77	0.00	31.88	0.002
86.26	0.00	43.13	0.002	96.00	0.00	48.00	0.00	65.90	0.00	32.95	0.002

Cuadro 2: Cuadro de resultados para Greedy Relief

Ionosphere				**Parkinsons**				**Spectf-heart**			
%cl	***%red***	***Agr.***	**T**	***%cl***	***%red***	***Agr.***	**T**	***%cl***	***%red***	***Agr.***	**T**
83.10	2.94	43.02	0.01	95.00	0.00	47.50	0.002	72.86	13.64	43.25	0.01
90.14	2.94	46.54	0.01	92.50	0.00	46.25	0.002	74.29	20.45	47.37	0.01
88.73	2.94	45.84	0.01	97.50	0.00	48.75	0.002	74.29	20.45	47.37	0.01
90.14	2.94	46.54	0.01	95.00	0.00	47.50	0.002	74.29	22.73	48.51	0.01
80.60	2.94	41.77	0.01	100.00	0.00	50.00	0.002	71.01	31.82	51.42	0.01
86.54	2.94	44.74	0.01	96.00	0.00	48.00	0.002	73.35	21.82	47.58	0.01

Cuadro 3: Cuadro de resultados para búsqueda local

Ionosphere				**Parkinsons**				**Spectf-heart**			
%cl	***%red***	***Agr.***	**T**	***%cl***	***%red***	***Agr.***	**T**	***%cl***	***%red***	***Agr.***	**T**
94.37	88,24	91.30	18.52	90.00	95.45	92.73	4.52	65.71	75.00	70.36	28.28
90.14	85.29	87.72	19.53	87.50	63.64	75.57	1.48	77.14	70.45	73.80	26.08
84.51	88.24	86.37	18.72	97.50	77.27	87.39	2.807	74.29	68.18	71.23	18.67
87.32	79.41	83.37	18.96	92.50	100.00	96.25	3.761	75.71	68.18	71.95	24.53
88.06	91.18	89.62	18.21	97.14	77.27	87.21	4.197	71.01	72.73	71.87	18.26
88.88	86.47	87.68	18.79	92.93	82.73	87.83	3.353	72.77	70.91	71.84	23.16

En primer lugar, observamos que el algoritmo 1-NN obtiene buenos resultados para Ionosphere, excelentes para Parkinsons y bastante malos para Spectf-Heart (un clasificador aleatorio tendría teóricamente un 50 por ciento de acierto). Evidentemente no puede reducir pesos por construcción, luego la función objetivo está acotada superiormente por 0.5.

Al comparar con el greedy Relief observamos que en general mejora ligeramente los porcentajes de acierto en clasificación para los 3 datasets. Por otro lado, reduce una característica en Ionosphere aunque era trivial de reducir pues si observamos los datos en crudo, casi todos sus valores son iguales

Cuadro 4: Cuadro resumen de resultados generales

	Ionosphere				**Parkinsons**				**Spectf-heart**			
	%cl	***%red***	***Agr.***	**T**	***%cl***	***%red***	***Agr.***	**T**	***%cl***	***%red***	***Agr.***	**T**
1-NN	86.26	0.00	43.13	0.002	96.00	0.00	48.00	0.002	65.90	0.00	32.95	0.002
RELIEF	86.54	2.94	44.74	0.01	96.00	0.00	48.00	0.002	73.35	21.82	47.58	0.01
BL	88.88	86.47	87.68	18.79	92.93	82.73	87.83	3.35	72.77	70.91	71.84	23.16

y por tanto no discriminan y pueden eliminarse. En Parkinson no logra reducir y por tanto se limita mucho la función objetivo. Finalmente en Spectf-Heart si logra reducir en torno a un 20 por ciento de características pero sigue obteniendo mala función objetivo agregada pues se le da gran importancia a la reducción en la ponderación.

Para búsqueda local la desventaja es el tiempo de ejecución, aunque podría acotarse cambiando las condiciones de parada. Para nuestros datasets es asumible. Observamos que se encuentran resultados mucho mejores que con Relief en todos los datasets, en Ionosphere y Parkinsons, la función objetivo se dispara casi al 90 por ciento, consiguiendose además reducir en torno al 80 por ciento de características. En Spectf-Heart conseguimos un resultado similar pero no tan bueno, nuestra tasa de clasificación sube al 72 por ciento y podemos descartar un 70 por ciento de características.

Por ello consideramos que es un buen punto de partida el algoritmo de búsqueda local para compararlo con los modelos bioinspirados de las siguientes prácticas y ver su funcionamiento, converge a un óptimo local al menos para nuestros casos aunque pueda tener periodos de oscilación por como se mutan las componentes, el algoritmo Greedy por su parte es muy rápido y genera una buena primera aproximación, aunque como damos mucha importancia a la simplificación de características no obtiene buenos resultados en estos datasets.

7.2. Algoritmos genéticos

Pasamos a mostrar y comentar los resultados de la ejecución de los algoritmos genéticos implementados en esta práctica. Como nomenclatura adicional en la última tabla añadimos el sufijo 1 para las versiones del algoritmo que utilicen el operador de cruce BLX y el sufijo 2 para el cruce aritmético.

Cuadro 5: Cuadro de resultados para AGG con cruce BLX

Ionosphere				**Parkinsons**				**Spectf-heart**			
%cl	***%red***	***Agr.***	**T**	***%cl***	***%red***	***Agr.***	**T**	***%cl***	***%red***	***Agr.***	**T**
77.46	70.59	74.03	118.73	90.00	86.36	88.18	30.98	72.86	59.09	65.97	126.74
88.73	73.53	81.13	119.36	95.00	72.73	83.86	30.88	85.71	56.82	71.27	126.74
83.10	67.65	75.37	120.57	85.00	90.91	87.95	30.91	67.14	68.18	67.66	127.12
90.14	61.76	75.95	117.26	85.00	77.27	81.14	30.98	68.57	61.36	64.97	126.05
88.06	73.53	80.79	121.82	94.29	81.82	88.05	32.96	73.91	56.82	65.37	126.79
85.50	69.41	77.46	119.55	89.86	81.82	85.84	31.34	73.64	60.45	67.05	126.69

Cuadro 6: Cuadro de resultados para AGG con cruce aritmético

Ionosphere				**Parkinsons**				**Spectf-heart**			
%cl	***%red***	***Agr.***	**T**	***%cl***	***%red***	***Agr.***	**T**	***%cl***	***%red***	***Agr.***	**T**
90.14	58.82	74.48	119.41	87.50	72.73	80.11	30.97	74.29	59.09	66.69	126.87
95.77	67.65	81.71	118.27	85.00	81.82	83.41	30.76	71.43	56.82	64.12	126.12
85.92	67.65	76.78	119.24	100.00	77.27	88.64	30.86	74.29	54.55	64.42	126.57
85.92	64.71	75.31	119.03	92.50	72.73	82.61	31.03	72.86	56.82	64.84	127.58
85.07	67.65	76.36	121.56	82.86	72.73	77.79	33.04	55.07	56.82	55.95	127.14
88.56	65.29	76.93	119.50	89.57	75.45	82.51	31.33	69.59	56.82	63.20	126.86

En primer lugar con respecto a los AGG se observa que el cruce BLX considera resultados consistentemente mejores que el cruce aritmético para todos los datasets y sobre todo se acentúa en Parkinsons y Spectf-Heart. Creemos que esto es debido a la capacidad de exploración extra que tiene el cruce BLX fuera del intervalo de los padres mientras que el aritmético esta delimitado de forma estricta por los padres.
Por otro lado, si comparamos con el mejor algoritmo de la P1 que era búsqueda local, los resultados son algo peores con AGG-BLX que con búsqueda local, aunque similares. Sobre todo la tasa de clasificación es igual de buena pero reduce algo menos perdiendo por tanto mucho fitness al no conseguir tanta simplicidad como la búsqueda local. En los tiempos salen perdiendo claramente los genéticos aunque no por ello deben descartarse ya que factores como la naturaleza de los datasets, la sesgadez de las particiones de validación y los hiperparámetros usados en los algoritmos. Entran por tanto en juego los teoremas de No-Free-lunch para darnos certeza en que es probable que estos algoritmos sean útiles para otros problemas.

Cuadro 7: Cuadro de resultados para AGE con cruce BLX

Ionosphere				**Parkinsons**				**Spectf-heart**			
%cl	***%red***	***Agr.***	**T**	***%cl***	***%red***	***Agr.***	**T**	***%cl***	***%red***	***Agr.***	**T**
85.92	97.06	91.49	118.86	95.00	100.00	97.50	30.87	65.71	70.45	68.08	127.50
95.77	94.12	94.95	119.25	95.00	100.00	97.50	30.87	81.43	77.27	79.35	126.07
91.55	97.06	94.30	119.53	92.50	100.00	96.25	31.98	77.14	70.45	73.80	126.85
88.73	91.18	89.95	118.61	92.50	100.00	96.25	31.15	81.43	75.00	78.21	126.14
82.09	94.12	88.10	124.78	100.00	100.00	100.00	33.19	81.16	75.00	78.08	126.99
88.81	94.71	91.76	120.21	95.00	100.00	97.50	31.61	77.37	73.64	75.51	126.71

Cuadro 8: Cuadro de resultados para AGE con cruce aritmético

Ionosphere				**Parkinsons**				**Spectf-heart**			
%cl	***%red***	***Agr.***	**T**	***%cl***	***%red***	***Agr.***	**T**	***%cl***	***%red***	***Agr.***	**T**
88.73	100.00	94.37	119.65	85.00	95.45	90.23	31.26	70.00	79.55	74.77	127.05
94.37	97.06	95.71	119.32	95.00	100.00	97.50	30.99	74.29	72.73	73.51	125.95
90.14	97.06	93.60	119.08	95.00	100.00	97.50	30.96	74.29	79.55	76.92	126.03
90.14	100.00	95.07	121.14	87.50	100.00	93.75	31.01	75.71	75.00	75.36	127.65
83.58	100.00	91.79	122.69	100.00	100.00	100.00	33.40	73.91	79.55	76.73	128.89
89.39	98.82	94.11	120.38	92.50	99.09	95.80	31.52	73.64	77.27	75.46	127.11

Por otro lado con respecto a las variantes estacionarias, se observa que los cruces obtienen resultados muy similares, quizá algo mejor el cruce aritmético. Esto puede ser debido a que como el elitismo mantiene a los mejores en la población se explota muy bien una zona muy prometedora del espacio de soluciones. Se produce un fenómeno curioso con las tasas de reducción que llegan al 100 por 100 aunque si observamos dichos pesos pese a estar por debajo del umbral de 0.1 realmente son parejos, por tanto se esta produciendo un escalado para que mejore el fitness pero realmente la tasa de reducción está hinchada y no simplifica tanto como indica ya que no se podrían eliminar todas las características evidentemente.

En resumen esta última tabla nos muestra una importante superioridad de AGE frente a AGG e incluso frente a búsqueda local, sobre todo en Ionosphere y Parkinsons aunque como hemos comentado antes las tasas de reducción pueden producir resultados algo engañosos.

Cuadro 9: Cuadro resumen de resultados generales

	Ionosphere				Parkinsons				Spectf-heart			
	%cl	*%red*	*Agr.*	T	*%cl*	*%red*	*Agr.*	T	*%cl*	*%red*	*Agr.*	T
1-NN	86.26	0.00	43.13	0.002	96.00	0.00	48.00	0.002	65.90	0.00	32.95	0.002
RELIEF	86.54	2.94	44.74	0.01	96.00	0.00	48.00	0.002	73.35	21.82	47.58	0.01
BL	88.88	86.47	87.68	18.79	92.93	82.73	87.83	3.35	72.77	70.91	71.84	23.16
AGG1	85.50	69.41	77.46	119.55	89.86	81.82	85.84	31.34	73.64	60.45	67.05	126.69
AGG2	88.56	65.29	76.93	119.50	89.57	75.45	82.51	31.33	69.59	56.82	63.20	126.86
AGE1	88.81	94.71	91.76	120.21	95.00	100.00	97.50	31.61	77.37	73.64	75.51	126.71
AGE2	89.39	98.82	94.11	120.38	92.50	99.09	95.80	31.52	73.64	77.27	75.46	127.11

7.3. Algoritmos meméticos

Mostramos en esta sección los resultados de los algoritmos meméticos. La nomenclatura utilizada en las tablas es AM1 para AM(10, 1), AM2 para AM(10, 0.1) y AM3 para AM(10, 0.1M).

Cuadro 10: Cuadro de resultados para AM1 con cruce BLX

Ionosphere				Parkinsons				Spectf-heart			
%cl	*%red*	*Agr.*	T	*%cl*	*%red*	*Agr.*	T	*%cl*	*%red*	*Agr.*	T
92.96	94.12	93.54	122.58	92.50	86.36	89.43	30.78	77.14	75.00	76.07	130.32
95.77	85.29	90.53	123.18	87.50	90.91	89.20	30.67	77.14	79.55	78.34	131.16
90.14	82.35	86.25	122.44	92.50	95.45	93.98	30.90	74.29	65.91	70.10	130.54
94.37	91.18	92.77	123.42	92.50	100.00	96.25	30.66	77.14	70.45	73.80	129.95
88.06	94.12	91.09	126.89	97.14	90.91	94.03	32.73	76.81	72.73	74.77	131.63
92.26	89.41	90.84	123.70	92.43	92.73	92.58	31.15	76.51	72.73	74.62	130.72

Cuadro 11: Cuadro de resultados para AM2 con cruce BLX

Ionosphere				Parkinsons				Spectf-heart			
%cl	*%red*	*Agr.*	T	*%cl*	*%red*	*Agr.*	T	*%cl*	*%red*	*Agr.*	T
91.55	88.24	89.89	119.16	95.00	95.45	95.23	30.76	75.71	77.27	76.49	125.37
95.77	79.41	87.59	118.59	92.50	86.36	89.43	30.73	74.29	70.45	72.37	124.91
88.73	76.47	82.60	119.94	92.50	86.36	89.43	30.74	68.57	81.82	75.19	125.11
91.55	88.24	89.89	118.76	95.00	90.91	92.95	30.62	75.71	68.18	71.95	124.80
88.06	88.24	88.15	123.40	100.00	90.91	95.45	32.46	60.87	79.55	70.21	126.45
91.13	84.12	87.63	119.97	95.00	90.00	92.50	31.06	71.03	75.45	73.24	125.33

Cuadro 12: Cuadro de resultados para AM3 con cruce BLX

Ionosphere				Parkinsons				Spectf-heart			
%cl	*%red*	*Agr.*	T	*%cl*	*%red*	*Agr.*	T	*%cl*	*%red*	*Agr.*	T
84.51	88.24	86.37	118.04	90.00	90.91	90.45	30.73	67.14	70.45	68.80	125.09
88.73	82.35	85.54	118.95	87.50	95.45	91.48	30.78	78.57	75.00	76.79	125.25
92.96	88.24	90.60	117.13	92.50	95.45	93.98	30.83	82.86	68.18	75.52	125.22
92.96	82.35	87.66	117.64	82.50	95.45	88.98	30.58	77.14	75.00	76.07	125.28
80.60	97.06	88.83	121.91	97.14	100.00	98.57	32.56	69.57	77.27	73.42	126.48
87.95	87.65	87.80	118.74	89.93	95.45	92.69	31.10	75.06	73.18	74.12	125.46

Cuadro 13: Cuadro resumen de resultados generales

	Ionosphere				**Parkinsons**				**Spectf-heart**			
	%cl	***%red***	***Agr.***	**T**	***%cl***	***%red***	***Agr.***	**T**	***%cl***	***%red***	***Agr.***	**T**
1-NN	86.26	0.00	43.13	0.002	96.00	0.00	48.00	0.002	65.90	0.00	32.95	0.002
RELIEF	86.54	2.94	44.74	0.01	96.00	0.00	48.00	0.002	73.35	21.82	47.58	0.01
BL	88.88	86.47	87.68	18.79	92.93	82.73	87.83	3.35	72.77	70.91	71.84	23.16
AGG1	85.50	69.41	77.46	119.55	89.86	81.82	85.84	31.34	73.64	60.45	67.05	126.69
AGG2	88.56	65.29	76.93	119.50	89.57	75.45	82.51	31.33	69.59	56.82	63.20	126.86
AGE1	88.81	94.71	91.76	120.21	95.00	100.00	97.50	31.61	77.37	73.64	75.51	126.71
AGE2	89.39	98.82	94.11	120.38	92.50	99.09	95.80	31.52	73.64	77.27	75.46	127.11
AM1	92.26	89.41	90.84	123.70	92.43	92.73	92.58	31.15	76.51	72.73	74.62	130.72
AM2	91.13	84.12	87.63	119.97	95.00	90.00	92.50	31.06	71.03	75.45	73.24	125.33
AM3	87.95	87.65	87.80	118.74	89.93	95.45	92.69	31.10	75.06	73.18	74.12	125.46

Destacamos que la comparación clara de estos algoritmos ha de ser con los algoritmos generacionales pues se basan en ellos, se nota claramente que la búsqueda local explotadora unida al poder explorador generacional genera un algoritmo con muy buenos resultados, mejores que ambas por separado (búsqueda local y AGGs). Se acercan también bastante a los logrados por las variantes estacionarias. Concluimos que como parecía indicar la lógica las mejores versiones meméticas son las que exploraban todos los cromosomas y la que exploraba a los mejores de ellos, quedando algo por detrás la aleatoria.