Metaheurísticas (2016-2017)
Grado en Ingeniería Informática
Universidad de Granada

Memoria Práctica 1.b

Técnicas de Búsqueda basadas en Poblaciones para el Problema del Aprendizaje de Pesos en Características

---

Algoritmos a desarrollar:
1-nn
relief
bl
agg-blx
agg-ca
age-blx
age-ca
am(10,1.0)
am(10,0.1)
am(10,0.1mej)

Javier Béjar Méndez
45337539p
javierbejarmendez@correo.ugr.es
Grupo 1 (Viernes de 17:30 a 19:30)

17 de abril de 2017

Índice

1. Problema del Aprendizaje de Pesos en Características **3**
1.1. Conjuntos de datos considerados 3

2. Aspectos generales de aplicación de los algoritmos **4**

3. Algoritmo de comparacion Greedy Relief **7**

4. Algoritmo de Búsqueda Local Primero el Mejor **8**

5. Algoritmos Genéticos **8**

6. Algoritmo Meméticos **11**

7. Procedimiento considerado para el desarrollo de la práctica **12**

8. Resultados **13**
8.1. Clasificador 1-NN 13
8.2. Relief 14
8.3. Búsqueda Local 15
8.4. AGG-BLX 16
8.5. AGG-AC 17
8.6. AGE-BLX 18
8.7. AGE-AC 19
8.8. AM-(10,1.0) 20
8.9. AM-(10,0.1) 21
8.10. AM-(10,0.1mej) 22
8.11. Globales 23

Índice de figuras

5.1. Jerarquía de clases algoritmos genéticos. 9
8.1. Tabla de resultados para el algoritmo 1-NN. 13
8.2. Tabla de resultados para el algoritmo Relief. 14
8.3. Tabla de resultados para el algoritmo de búsqueda local. 15
8.4. Tabla de resultados para el algoritmo AGG-BLX. 16
8.5. Tabla de resultados para el algoritmo AGG-AC. 17
8.6. Tabla de resultados para el algoritmo AGE-BLX. 18
8.7. Tabla de resultados para el algoritmo AGE-AC. 19
8.8. Tabla de resultados para el algoritmo AM-(10,1.0). 20
8.9. Tabla de resultados para el algoritmo AM-(10,0.1). 21
8.10. Tabla de resultados para el algoritmo AM-(10,0.1mej). 22
8.11. Tabla de resultados globales. 23

1. Problema del Aprendizaje de Pesos en Características

El problema del APC consiste en optimizar el rendimiento de un clasificador basado en vecinos 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). Así, el problema del APC se puede formular como:

Maximizar $tasa\_clas(1-NN(s)) = 100 \cdot \frac{\text{nº de instancias bien clasificadas de } T}{\text{nº total de instancias de } T}$

sujeto a que:

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

donde:

- $W = (w_1, ..., w_n)$ es una solución al problema que consiste en un vector de números reales de tamaño n que define el peso que pondera 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 la técnica de validación *leave-one-out* y 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 como el de prueba.

1.1. Conjuntos de datos considerados

Trabajaremos con los tres conjuntos de datos siguientes:

- Sonar, conjunto de datos de detección de materiales mediante señales de sónar. 208 ejemplos con 60 características que deben ser clasificados en 2 clases.
- Wdbc (Wisconsin Database Breast Cancer), conjunto de datos de imágenes de celulas. 569 ejemplos con 30 características que deben ser clasificados en 2 clases.
- Spambase, conjunto de datos de detección de SPAM frente a correo electrónico seguro. 460 ejemplos con 57 características que deben ser clasificados en 2 clases.

Los conjutos de datos se dividirán en 5 subconjuntos aleatorios divididos al 50 % para poder aplicar la técnica de validación $5x2cv$, consistente en entrenar sobre la mitad de los datos (validando la función objetivo mediante *leave-one-out*) y validar sobre la otra mitad.

2. Aspectos generales de aplicación de los algoritmos

Aspectos comunes a todos los algoritmos:

- Esquema de representación, la solución se representa como el siguiente vector real:

$$W = (w_1, w_i, ..., w_n), \quad w_i \subset [0, 1]$$

siendo n el número de características, donde w_i representa el peso asociado a la característica c_i del conjunto de datos considerado.

- Función objetivo, maximizar la tasa de acierto del clasificador $1 - NN$ sobre el conjunto de entrenamiento mediante la técnica de validación *leave-one-out*. Dado un lasificador $1 - NN$, con un conjunto de entrenamiento A de N_d datos con N_c características, la tasa de acierto de la solución W se obtiene de la siguiente manera:

$numeroAciertos = 0$
Begin desde $i = 0$ hasta N_d
    $clase = ClasificaDato(A_i, W)$
    Si $clase ==$ a la clase de A_i
        $numeroAciertos + 1$
End
Devolver $tasa\_clas = numeroAciertos/N_d$

Donde la función $ClasificaDato(A_i, W)$ recorre todo el conjunto y devuelve la clase del dato mas cercano a A_i sin contar consigo mismo. La distancia entre dos datos C_1 y C_2 dado un vector de pesos W se define como la distancia euclidea entre todas sus características c_i:

$$Dist_e(C_1, C_2) = \sqrt{\sum_{i=1}^{n} (c_{1i} + c_{2i})^2 \cdot w_i}$$

- Generador aleatorio, se utilizara como generador aleatorio la versión mt19937 implementada en la libreria <random> de c++, basado en el algoritmo Mersenne Twister.

- Generación de la solución inicial, la solución inicial se generará de forma aleatoria utilizando una distribución uniforme en $[0, 1]$ en todos los casos, como sigue:

crear W vacio
Begin desde i hasta n
    $w_i = uniform\_real\_random[0, 1]$
End

- Criterio de aceptación, se considera una mejora cuando se aumenta el valor global de la función objetivo.
- Criterio de parada, se detendrá la ejecución del algoritmo bien cuando no se encuentre mejora al generar un máximo número de vecinos (BL) o bien cuando se hayan evaluado 15000 soluciones distintas.
- Esquema de generación de vecinos y operador de mutación, se empleará el movimiento de cambio por mutación normal $Mov(W, \sigma)$ que altera el vector W sumándole una componente Z generada a partir de una distribución normal de media 0 y varianza σ^2. Es decir, dado un vector W un posible vecino o mutación W' se genera tal que así:

  $$W' = W(w_1, w_i, ..., w_n) + z_k = (w_1, w_i, ..., w_k + z_k, ..., w_n), \quad z_k \subset normal[0, \sigma^2]$$

  Truncando $w_i + z_k$ al intervalo $[0, 1]$. Descripción en pseudocódigo:

  Dado W, $posicion$ y σ
  $w_{posicion} += normal\_random[0, \sigma^2]$
  Si $w_{posicion} < 0, w_{posicion} = 0$
  Si $w_{posicion} > 1, w_{posicion} = 1$

Aspectos comunes a los algoritmos genéticos:

- Operador de selección, se usará el torneo binario, en el caso del esquema generacional realizamos tantos torneos como el tamaño de la población y en el esquema estacionario dos torneos. Seleccionaremos los dos contrincantes generando normalmente un único valor natural aleatorio, que atiende a una distribución uniforme, conceptualmente podemos entenderlo como una matriz cuadrada de n cromosomas, dondel el numero de fila y columna nos dice los dos cromosomas que participaran en el torneo, descartando la diagonal. Es decir, dado un vector de cromosomas $C = (c_0, c_i, ..., c_{n-1})$ con tamaño de poblacion n la operación de torneo binario se define como:

  Do
  $posicion = uniform\_integeer\_random[1, n^2 - 2]$
  Mientras ($posicion$ módulo $(n + 1)) == 0$)
  $rival_1 = V[posicion \backslash n]$
  $rival_2 = V[posicion$ módulo $n]$
  Si $rival_1 > rival_2$
  $padre = rival_1$
  Si no
  $padre = rival_2$

- Operador de cruce $BLX - \sigma$, con un $\sigma = 0{,}3$ generamos dos hijos H_k con dos cromosomas padres C_1 y C_2 de la siguiente manera:

  Dados:
  $$C_1 = (C_{11}, ..., C_{1n}), \quad C_2 = (C_{21}, ..., C_{2n})$$
  Generamos:
  $$H_k = (h_{k1}, ..., h_{kn}), \quad k = 1, 2$$
  Donde h_{ki} se generea mediante una distribución uniforme en el intervalo:
  $$[C_{min} - I \cdot \sigma, C_{max} + I \cdot \sigma]$$
  Con:
  $$C_{min} = min(C_{1i}, C_{2i}), \quad C_{max} = max(C_{1i}, C_{2i}), \quad I = C_{max} - C_{min}$$
  El algoritmo que implementa dicha operación es el siguiente:

  Begin desde $i = 0$ hasta n:
  Si $C_{1i} < C_{2i}$:
  $C_{min} = C_{1i}$
  $C_{max} = C_{2i}$
  Si no:
  $C_{min} = C_{2i}$
  $C_{max} = C_{1i}$
  $I = (C_{max} - Cmin) \cdot \sigma$
  $h_{1i} = uniform\_random[C_{min} - I, C_{max} + I]$
  Si $h_{1i} > 1$:
  $h_{1i} = 1$
  Si $h_{1i} < 0$:
  $h_{1i} = 0$
  $h_{2i} = uniform\_random[C_{min} - I, C_{max} + I]$
  Si $h_{2i} > 1$:
  $h_{2i} = 1$
  Si $h_{2i} < 0$:
  $h_{2i} = 0$
  End
  Devolver $H_1, \quad H_2$

- Operador CA(Cruce aritmético), consistente en la media aritmética de todos los genes de ambos padres:

  Dados:
  $$C_1 = (C_{11}, ..., C_{1n}), \quad C_2 = (C_{21}, ..., C_{2n})$$

Generamos:

$$H_k = (\frac{C_{11} + C_{21}}{2}, ..., \frac{C_{1n} + C_{2n}}{2}), \quad k = 1, 2 \quad H_1 = H_2$$

El algoritmo que implementa dicha operación es el siguiente:

Begin desde $i = 0$ hasta n:
$h_{1i} = h_{2i} = \frac{C_{1i}+C_{2i}}{2}$
End
Devolver $H_1, \quad H_2$

3. Algoritmo de comparacion Greedy Relief

El alritmo greedy Relief se basa en generar un vector de pesos W a partir de las distancias de cada ejemplo D_{ejem} a su enemigo D_e más cercano y a su amigo D_a más cercano. El algoritmo que implementa dicha operación es el siguiente:

$W = (0, ..., 0)$
Begin desde $i = 0$ hasta n:
$minDistAmigo = MAX$
$minDistEnemigo = MAX$
$amigo = -1$
$enemigo = -1$
Begin desde $j = 0$ hasta n:
$dist = distancia_{euclidea}(D_i, D_j)$
Si $clase(D_i) != clase(D_j)$
Si $dist < minDistEnemigo$:
$minDistEnemigo = dist$
$enemigo = j$
Si no y si $i != j$
Si $dist < minDistAmigo$:
$minDistAmigo = dist$
$amigo = j$
End
Begin desde $j = 0$ hasta $nAtributos$:
$w_j = w_j + distancia_{punto}(d_{ij}, d_{enemigoj})$
$w_j = w_j - distancia_{punto}(d_{ij}, d_{amigoj})$
End
End
$Normaliza(W)$
Devuelve W

4. Algoritmo de Búsqueda Local Primero el Mejor

El algoritmo de búsqueda local primero el mejor consiste en ir generando soluciones vecinas hasta encontrar una mejor que la actual, en nuestro caso concreto se detendrá cuando realice 15000 evaluaciones de la función objetivo o cuando genere $20 \cdot n$ vecinos sin encontrar una solución mejor, siendo n el número de atributos. El algoritmo concreto es el siguiente:

$W = solucion\_aleatoria$
$mejorTasa = validaSolEntr(W)$
$posicion = (0, 1, ..., n)$
Begin desde $i = 0$, $j = 0$ hasta $j < maxVecinos$ o $i < maxEvaluaciones$:
    Si $posicion$ esta vacio:
        Begin desde $k = 0$ hasta $nAtributos$:
            $aux = uniform\_integeer\_random[0, nAtributos - 1]$
            $posicion_k = aux$ modulo $nAtributos$
        End
    $W' = genera\_vecino(W, posicion.back())$
    $posicion.pop\_back()$
    $tasa = validaSolEntr(W')$
    Si $tasa > mejorTasa$:
        $mejorTasa = tasa$
        $W = W'$
        $j = 0$
End
Devuelve W

Donde $validaSol(W)$ devuelve la tasa de acierto con el vector de pesos W mediante leave-one-out y $genera\_vecino(W, posicion)$ genera un vecino W' aplicando el operador de generación de vecinos a W en w_i con $i = posicion$.

5. Algoritmos Genéticos

Se han desarrollado 4 variantes:

- Generacional con operador de cruce $BLX - \sigma$ (AGG-BLX)
- Generacional con operador de cruce aritmético (AGG-AC)
- Estacionario con operador de cruce $BLX - \sigma$ (AGE-BLX)
- Estacionario con operador de cruce aritmético (AGE-AC)

El código fuente para los algoritmos genéticos se ha desarrollado de una forma jerárquica de clases, con el siguiente esquema:

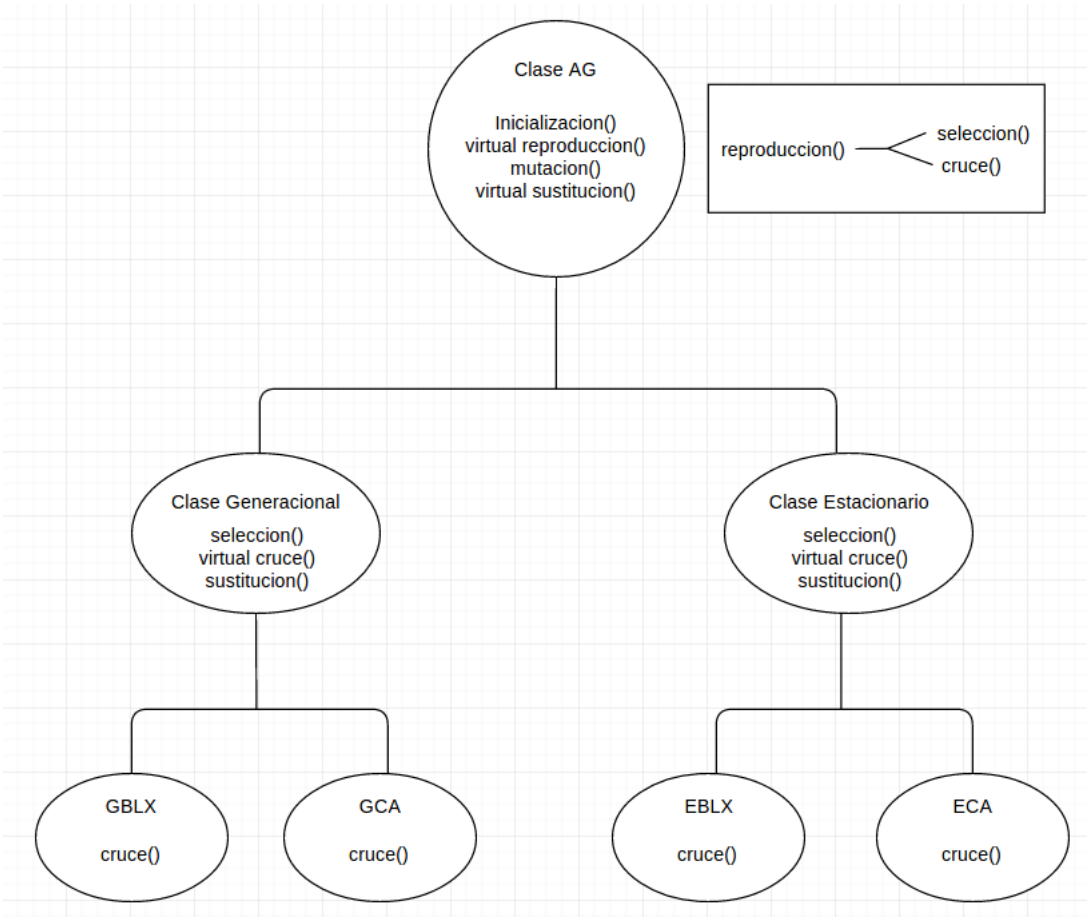

Figura 5.1: Jerarquía de clases algoritmos genéticos.

El algoritmo de evolución consiste en las fases de reproducción(selección + cruce), mutación y sustitución, hasta que se hayan alcanzado 15000 evaluaciones de la función objetivo. Los operadores de cruce , mutación, selección y la inicialización ya han sido descritos en la sección 2. La operación de sustitución en el modelo Generacional consiste en que la nueva población generada tras la selección, cruce y mutación sustituye a la actual, con elitismo, luego el mejor de la actual si no sobrevive se sustituye por la peor de la nueva población. El pseudocódigo es el siguiente:

$mejorEncontrado = false$
$min = MAX$
$max = MIN$
$peor = 0$
$mejor = 0$
$mejorCromosoma =$ mejor cromosoma de la población actual
$cromosoma_i$ son los cromosomas de la población siguiente
Begin desde $i = 0$ hasta $nPoblacion$ o $nValoraciones < 15000$:
    Si $cromosoma_i$ necesita valorar:
        valorar $cromosoma_i$
        $nValoraciones+ = 1$
    Si !$mejorEncontrado$ y $cromosoma_i = mejorCromosoma$:
        $mejorEncontrado = true$
    Si !$mejorEncontrado$ y $cromosoma_i < min$:
        $min =$valoración de $cromosoma_i$
        $peor = i$
    Si $cromosoma_i > max$:
        $max =$valoración de $cromosoma_i$
        $mejor = i$
End
Si !$mejorEncontrado$:
    $cromosoma_{peor} = mejorCromosoma$
    Si $cromosoma_{mejor} > mejorCromosoma$:
        $posicionMejorCromosoma = mejor$
        $mejorCromosoma = cromosoma_{mejor}$
Si no:
    $posicionMejorCromosoma = mejor$
    $mejorCromosoma = cromosoma_{mejor}$
Switch de punteros para que la población actual pase a ser la población siguiente y viceversa
Se elimina la población siguiente para continuar en el siguiente ciclo

En el caso de los algoritmos Estacionales los dos hijos generados, tras la mutación se insertan por los dos peores de la población actual y recorremos la población actual en busca de los dos peores y el mejor, como se muestra a continuación:

$cromosomasActual_{peor1} = cromosomasSiguiente_1$
$cromosomasActual_{peor2} = cromosomasSiguiente_2$
$min1 = MAX$
$min2 = MAX$
$max = MIN$
Begin desde $i = 0$ hasta $nCromosomas$ o $nValoraciones < 15000$:
    Si $cromosomasActual_i$ necesita valorar
        valorar $cromosomasActual_i$
        $nValoraciones+ = 1$
    Si $cromosomasActual_i < min1$:
        Si $min2! = MAX$:
            Si $cromosomasActual_{peor1} < cromosomasActual_{peor2}$:
                $peor2 = peor1$
                $min2 = min1$
        Si no:
            $peor2 = peor1$
            $min2 = min1$
        $peor1 = i$
        $min1 =$ valoración de $cromosomasActual_i$
    Si no:
        Si $cromosomasActual_i < min2$
            $peor2 = i$
            $min2 =$ valoración de $cromosomasActual_i$
    Si $cromosomasActual_i > max$:
        $max =$ valoración de $cromosomasActual_i$
        $posicionMejorCromosoma = i$
End

6. Algoritmo Meméticos

Se han desarrollado 3 variantes, a partir del algoritmo generacional con el operador de cruce $BLX - \sigma$(AGG-BLX):

- AM-(10,1.0): Cada 10 generaciones, se aplica la BL sobre todos los cromosomas de la población.
- AM-(10,0.1): Cada 10 generaciones, se aplica la BL sobre un subconjunto de cromosomas de la población seleccionado aleatoriamente con probabilidad pLS igual a 0.1 para cada cromosoma.

- AM-(10,0.1mej): Cada 10 generaciones, aplicar la BL sobre los $0{,}1 \cdot N$ mejores cromosomas de la población actual (N es el tamaño de ésta).

Situandonos en la jerarquía de clases establecidas el algoritmo memético herada de la clase GBLX, y redefine el método de inserción definido en la clase Generacional para aplicar la BL. La diferencia entre los algoritmos AM-(10,1.0) y AM-(10,0.1) es solo de parámetros, en la primera ejecutamos la búsqueda local sobre toda la población mientras que en la segunda calculamos a priori el número de búsquedas locales a aplicar basados en la porbabilidad pLS. El algoritmo AM-(10,0.1mej) se desarrolla aparte(herada también de GBLX) ya que tenemos que ordenar la población para aplicar la BL a los mejores cromosomas, calculando a priori el número de BLs a realizar.

En las versiones AM-(10,1.0) y AM-(10,0.1) la integración de la BL en el algoritmo genético consiste en contabilizar el número de generaciones para que cada 10 generaciones, antes de valorar los cromosomas, realicemos una búsqueda local con los parámetros establecidos, el número de optimizaciones locales a realizar será N para el primer algoritmo y $0{,}1 \cdot N$ para el segundo algoritmo, con N igual al tamaño de la población, si la generación no es múltiplo de 10 la integración se realiza normalmente, descrita en la sección 6.

En la versión AM-(10,0.1mej) al igual que en las anteriores contabilizamos el número de generación y calculamos el número de optimizaciones locales a realizar de la misma manera. Antes de realizar las optimizaciones locales se ordena el vector de cromosomas de la población siguiente, de tal manera que apliquemos la BL sobre los $0{,}1 \cdot N$ mejores cromosomas de la población, si la generación no es múltiplo de 10 la integración se realiza normalmente, descrita en la sección 6.

7. Procedimiento considerado para el desarrollo de la práctica

He decidio realizar la práctica íntegra, sin uso de frameworks ni código facilitado por el profesorado. Comenzando por un pequeño programa para crear las particiones de datos 5x2cv, y dejarlas en un formato más ligero que el arff. Luego desarrollé las estructuras de datos necesarias para almacenar los datos, implementé el clasificador 1-NN para validar los datos de prueba y los de entrenamiento y el código necesario para cargar los datos, generar números aleatorios y funciones auxiliares de utilidad como las mediciones de tiempos, cálculo de distancias o normalización de los vectores. Con la base ya programada continué desarrollando los algoritmos Relief y BL, Genéticos y Meméticos. Por útimo realicé las distintas ejecuciones y su análisis consiguiente. El material de apoyo utilizado ha sido sobre todo los apuntes de teoría y prácticas[1] y la documentación de c++[2] que es el lenguaje en el que están implementados todos los algoritmos.

8. Resultados

Comenzaremos analizando los resultados de los algoritmos de manera individual y comparandolos con el algoritmo Relief, para terminar con una compración global.

8.1. Clasificador 1-NN

Seed=365 1278846	Sonar		Wdbc		Spambase	
	%_clas	T	*%_clas*	T	*%_clas*	T
Partición 1-1	84.5955	0.000025	95.9581	0.000029	82.3913	0.000045
Partición 1-2	80.7582	0.000022	94.2006	0.000022	82.6087	0.000018
Partición 2-1	80.7859	0.000009	95.7790	0.000012	83.0435	0.000023
Partición 2-2	82.7092	0.000008	95.7808	0.000052	82.6087	0.000017
Partición 3-1	83.6246	0.000009	95.2557	0.000012	81.3043	0.000017
Partición 3-2	84.5955	0.000041	94.9036	0.000012	83.4783	0.000022
Partición 4-1	85.0439	0.000008	95.6054	0.000059	83.2609	0.000017
Partición 4-2	82.6722	0.000008	94.7270	0.000012	85.2174	0.000018
Partición 5-1	83.6986	0.000008	95.2533	0.000012	82.1739	0.000021
Partición 5-2	80.7952	0.000008	95.0803	0.000053	83.2609	0.000017
Media	82.9279	0.000014	95.2544	0.000027	82.9348	0.000022
Mínimo	80.7582	0.000008	94.2006	0.000012	81.3043	0.000017
Máximo	85.0439	0.000041	95.9581	0.000059	85.2174	0.000045
Desviación	1.6692	0.000011	0.5490	0.000020	1.0261	0.000008

Figura 8.1: Tabla de resultados para el algoritmo 1-NN.

Estos son los resultados obtenidos por el clasificador sin ninguna mejora, es decir, todos los atributos tienen un peso de 1. Es muy rápido y obtiene niveles de clasificación aceptables en cuanto a la simplicidad del algoritmo. Se ajusta peor a los conjuntos sonar y Spambase debido a que estos tienen atributos que no son tan relevantes, en el conjunto wdbc parece que casi todos los atributos son igual de importantes.

8.2. Relief

Seed=365 1278846	Sonar		Wdbc		Spambase	
	%_clas	T	*%_clas*	T	*%_clas*	T
Partición 1-1	85.0717	0.0013	95.9600	0.0030	88.2609	0.0034
Partición 1-2	85.0809	0.0008	94.7294	0.0032	83.4783	0.0033
Partición 2-1	80.3282	0.0008	94.1969	0.0029	84.7826	0.0036
Partición 2-2	85.5848	0.0008	96.3090	0.0032	85.4348	0.0032
Partición 3-1	85.5386	0.0007	95.6066	0.0029	85.6522	0.0036
Partición 3-2	86.9949	0.0008	95.2545	0.0030	84.5652	0.0032
Partición 4-1	85.0717	0.0008	96.4844	0.0030	86.3043	0.0034
Partición 4-2	85.0624	0.0008	94.9030	0.0029	87.6087	0.0033
Partición 5-1	81.8123	0.0008	94.7251	0.0029	85.8696	0.0033
Partición 5-2	86.5280	0.0008	96.3084	0.0029	85.0000	0.0033
Media	84.7074	0.0008	95.4477	0.0030	85.6957	0.0034
Mínimo	80.3282	0.0007	94.1969	0.0029	83.4783	0.0032
Máximo	86.9949	0.0013	96.4844	0.0032	88.2609	0.0036
Desviación	2.0575	0.0002	0.8015	0.0001	1.4229	0.0001

Figura 8.2: Tabla de resultados para el algoritmo Relief.

Como podemos observar el algoritmo Relief es un algoritmo muy rápido y con una tasa de acierto aceptable. La diferencia entre el mínimo y el máximo en las tasas de acierto es alta, al igual que la desviación estandar, debido a que el algoritmo relief solo genera una solución que puede adaptarse mejor o peor a la base de datos de prueba.

8.3. Búsqueda Local

Seed=365 1278846	Sonar		Wdbc		Spambase	
	%_clas	T	*%_clas*	T	*%_clas*	T
Partición 1-1	87.9843	2.2078	96.4863	1.5832	83.0435	6.3178
Partición 1-2	85.0532	0.8717	93.8485	2.3617	85.6522	5.4079
Partición 2-1	83.7078	0.9007	96.4826	1.9518	87.3913	5.1741
Partición 2-2	87.0227	1.0658	96.3071	2.3080	86.7391	5.5841
Partición 3-1	85.0809	0.7859	96.3115	2.6581	87.1739	6.9480
Partición 3-2	86.4817	1.2152	95.9563	2.5072	86.0870	5.4870
Partición 4-1	85.0809	0.9331	96.8372	1.8840	85.6522	5.5144
Partición 4-2	88.4235	1.0195	95.4275	2.1033	87.3913	6.3335
Partición 5-1	83.2409	0.8115	95.0822	3.3722	85.8696	6.5823
Partición 5-2	86.5187	2.2954	96.4851	1.8906	86.7391	6.7103
Media	85.8595	1.2107	95.9224	2.2620	86.1739	6.0059
Mínimo	83.2409	0.7859	93.8485	1.5832	83.0435	5.1741
Máximo	88.4235	2.2954	96.8372	3.3722	87.3913	6.9480
Desviación	1.7196	0.5634	0.9029	0.5082	1.2930	0.6376
Respecto al algoritmo Relief:						
Media	1.1521	1.2098	0.4747	2.2590	0.4783	6.0026
Mínimo	2.9127	0.7851	-0.3484	1.5804	-0.4348	5.1708
Máximo	1.4286	2.2942	0.3528	3.3690	-0.8696	6.9444
Desviación	-0.3379	0.5632	0.1014	0.5081	-0.1299	0.6375

Figura 8.3: Tabla de resultados para el algoritmo de búsqueda local.

En comparacion al Relief mejora de media entorno al 1 %, con un coste de tiempo bastante superior 1000 veces más lento, al tener más exploracion da con mejores resultados, pero la explotación es mínima, por lo que apenas mejora. Y como esta fuertemente aleatorizado la diferencia entre ejecuciones es notable.

8.4. AGG-BLX

Seed=365 1278846	Sonar		Wdbc		Spambase	
	%_clas	T	*%_clas*	T	*%_clas*	T
Partición 1-1	87.5358	10.1415	97.3666	40.7288	86.0870	47.1359
Partición 1-2	87.4619	10.5563	96.3084	40.3253	87.3913	48.0431
Partición 2-1	84.1933	10.2791	96.6593	39.5480	88.4783	48.0754
Partición 2-2	86.9949	10.4386	96.3071	40.6225	88.6957	47.8988
Partición 3-1	89.4406	9.9710	96.8384	40.5080	88.0435	46.7172
Partición 3-2	88.4050	10.2753	95.9556	40.6772	89.5652	46.3283
Partición 4-1	88.9552	9.9090	97.0138	39.8667	87.1739	47.6651
Partición 4-2	90.3282	10.3405	95.7796	40.5984	91.0870	48.2897
Partición 5-1	85.6681	9.9894	95.7845	40.0279	88.2609	48.3280
Partición 5-2	88.4050	10.2344	97.1856	40.8492	89.1304	47.3956
Media	87.7388	10.2135	96.5199	40.3752	88.3913	47.5877
Mínimo	84.1933	9.9090	95.7796	39.5480	86.0870	46.3283
Máximo	90.3282	10.5563	97.3666	40.8492	91.0870	48.3280
Desviación	1.8104	0.2108	0.5795	0.4263	1.3833	0.6810
Respecto al algoritmo Relief:						
Media	3.0315	10.2127	1.0722	40.3722	2.6957	47.5843
Mínimo	3.8651	9.9082	1.5827	39.5451	2.6087	46.3251
Máximo	3.3333	10.5550	0.8822	40.8460	2.8261	48.3244
Desviación	-0.2470	0.2106	-0.2220	0.4262	-0.0397	0.6808

Figura 8.4: Tabla de resultados para el algoritmo AGG-BLX.

Podemos observar como el tiempo ha crecido en comparación con los algoritmos anteriores, al realizar las evaluaciones de cada cromosoma generado, este algoritmo tiene una mayor exploración y explotación lo que le permite alcanzar soluciones mejores que el algoritmo Relief. Se percibe que el algoritmo es de mejor calidad ya que las soluciones tienen menor desviación.

8.5. AGG-AC

Seed=365 1278846	Sonar		Wdbc		Spambase	
	%_clas	T	*%_clas*	T	*%_clas*	T
Partición 1-1	87.0411	10.1717	97.5414	40.8934	86.5217	46.3749
Partición 1-2	83.6061	10.1900	96.4832	40.5200	86.7391	48.5246
Partición 2-1	84.1748	9.9522	97.0126	39.7234	88.2609	48.3871
Partición 2-2	87.4896	10.2525	96.3071	40.4623	85.6522	47.4868
Partición 3-1	87.0319	9.9282	96.1366	41.0116	87.6087	47.6229
Partición 3-2	87.4711	10.5302	96.6593	40.6241	88.0435	47.8004
Partición 4-1	86.0610	9.9439	96.3108	40.0257	86.7391	48.2627
Partición 4-2	86.9672	10.4649	96.3078	39.7961	90.0000	48.0170
Partición 5-1	84.2117	10.1637	96.4869	39.8722	88.4783	47.9010
Partición 5-2	86.9857	10.4940	97.8892	39.2150	88.9130	47.9602
Media	86.1040	10.2091	96.7135	40.2144	87.6957	47.8338
Mínimo	83.6061	9.9282	96.1366	39.2150	85.6522	46.3749
Máximo	87.4896	10.5302	97.8892	41.0116	90.0000	48.5246
Desviación	1.5127	0.2287	0.5859	0.5761	1.3011	0.6070
Respecto al algoritmo Relief:						
Media	1.3967	10.2083	1.2658	40.2114	2.0000	47.8304
Mínimo	3.2779	9.9275	1.9397	39.2121	2.1739	46.3717
Máximo	0.4947	10.5289	1.4048	41.0084	1.7391	48.5210
Desviación	-0.5447	0.2285	-0.2156	0.5760	-0.1218	0.6068

Figura 8.5: Tabla de resultados para el algoritmo AGG-AC.

El tiempo de ejecución es similar a su variante con el operador de cruce $BLX - \sigma$, pero la calidad de las clasificaciones es inferior debido a que converge antes(produce dos hijos iguales en cada cruce) produciendo una menor exploración. Sigue estando por encima de la BL y el algoritmo Relief.

8.6. AGE-BLX

Seed=365 1278846	Sonar		Wdbc		Spambase	
	%_clas	T	*%_clas*	T	*%_clas*	T
Partición 1-1	88.0028	9.7480	98.0689	39.4574	88.2609	46.5866
Partición 1-2	85.0254	10.0118	95.9556	39.6533	87.3913	46.0753
Partición 2-1	84.6879	9.7576	95.7833	40.9752	89.1304	45.8796
Partición 2-2	86.5187	10.0586	95.6017	39.8228	88.9130	45.8132
Partición 3-1	88.4882	9.6725	97.3672	39.2339	86.5217	47.1716
Partición 3-2	86.9579	10.0857	96.3065	39.6328	88.4783	46.2544
Partición 4-1	88.0028	9.7047	96.6617	39.3294	88.4783	46.1337
Partición 4-2	86.9672	10.0893	96.1311	39.5644	90.6522	46.4283
Partición 5-1	88.5344	9.6776	96.4869	39.2893	89.1304	46.1372
Partición 5-2	89.3759	10.0345	98.0646	39.6155	89.3478	46.2485
Media	87.2561	9.8840	96.6428	39.6574	88.6304	46.2728
Mínimo	84.6879	9.6725	95.6017	39.2339	86.5217	45.8132
Máximo	89.3759	10.0893	98.0689	40.9752	90.6522	47.1716
Desviación	1.5293	0.1845	0.8990	0.4986	1.1228	0.3907
Respecto al algoritmo Relief:						
Media	2.5488	9.8832	1.1950	39.6544	2.9348	46.2695
Mínimo	4.3597	9.6718	1.4048	39.2310	3.0434	45.8100
Máximo	2.3810	10.0880	1.5845	40.9720	2.3913	47.1680
Desviación	-0.5282	0.1843	0.0975	0.4985	-0.3001	0.3906

Figura 8.6: Tabla de resultados para el algoritmo AGE-BLX.

Sus resultados son prácticamente iguales a la variante generacional.

8.7. AGE-AC

Seed=365 1278846	Sonar		Wdbc		Spambase	
	%_clas	T	*%_clas*	T	*%_clas*	T
Partición 1-1	87.5266	9.6635	97.1905	38.8552	88.9130	45.9245
Partición 1-2	86.0055	9.9852	96.6599	39.6561	88.4783	46.0459
Partición 2-1	83.2409	9.7191	96.8347	39.3103	88.2609	45.8658
Partición 2-2	86.0333	10.0547	96.6593	39.5271	87.1739	45.9447
Partición 3-1	87.5081	9.6545	97.3653	42.7540	86.7391	46.2755
Partición 3-2	85.9963	10.1093	95.4287	44.9376	88.9130	45.9588
Partición 4-1	89.8983	9.7167	96.8372	44.4102	86.5217	45.8030
Partición 4-2	87.9380	10.0633	95.6035	44.4502	90.6522	45.8755
Partición 5-1	83.7356	9.7137	96.1360	42.4077	88.0435	45.9445
Partición 5-2	88.8997	9.9828	97.3622	39.4562	89.7826	45.9818
Media	86.6782	9.8663	96.6077	41.5765	88.3478	45.9620
Mínimo	83.2409	9.6545	95.4287	38.8552	86.5217	45.8030
Máximo	89.8983	10.1093	97.3653	44.9376	90.6522	46.2755
Desviación	2.1110	0.1869	0.6842	2.4628	1.3092	0.1290
Respecto al algoritmo Relief:						
Media	1.9709	9.8655	1.1600	41.5735	2.6522	45.9586
Mínimo	2.9127	9.6537	1.2318	38.8523	3.0434	45.7998
Máximo	2.9034	10.1080	0.8809	44.9344	2.3913	46.2719
Desviación	0.0536	0.1867	-0.1173	2.4627	-0.1138	0.1288

Figura 8.7: Tabla de resultados para el algoritmo AGE-AC.

Sus resultados son prácticamente iguales a la variante generacional.

8.8. AM-(10,1.0)

Seed=365 1278846	Sonar		Wdbc		Spambase	
	%_clas	T	*%_clas*	T	*%_clas*	T
Partición 1-1	92.3162	19.0164	97.5414	76.4350	91.7391	91.2594
Partición 1-2	86.9857	19.3063	96.6593	76.3689	86.0870	90.9030
Partición 2-1	86.0980	19.0474	97.7156	75.1585	87.3913	90.6509
Partición 2-2	90.8599	19.4619	95.9544	77.4280	88.6957	92.8387
Partición 3-1	92.2977	19.4284	96.6636	77.4650	89.1304	93.3145
Partición 3-2	87.9288	19.4890	96.4826	76.1595	88.4783	92.9500
Partición 4-1	90.8507	19.0875	97.7162	76.3678	90.0000	91.7129
Partición 4-2	86.9579	19.4854	95.7796	79.1425	91.0870	90.4833
Partición 5-1	86.1442	19.2115	96.6623	79.6931	90.8696	92.8710
Partición 5-2	89.3851	19.3956	97.0101	77.7368	89.7826	92.6579
Media	88.9824	19.2929	96.8185	77.1955	89.3261	91.9642
Mínimo	86.0980	19.0164	95.7796	75.1585	86.0870	90.4833
Máximo	92.3162	19.4890	97.7162	79.6931	91.7391	93.3145
Desviación	2.4688	0.1882	0.6817	1.3991	1.7435	1.0784
Respecto al algoritmo Relief:						
Media	4.2751	19.2921	1.3708	77.1925	3.6304	91.9608
Mínimo	5.7698	19.0157	1.5827	75.1556	2.6087	90.4801
Máximo	5.3213	19.4877	1.2318	79.6899	3.4782	93.3109
Desviación	0.4113	0.1880	-0.1198	1.3990	0.3205	1.0783

Figura 8.8: Tabla de resultados para el algoritmo AM-(10,1.0).

Al incluir la BL en el algoritmo genético generacional aumenta la tasa de acierto ligeramente, y aumenta el tiempo de ejecución considerablemente. La BL aporta una mayor explotación de las que las variantes genéticas carecen. La desviación sin embargo aumenta y esto sucede porque el algoritmo tiene una rápida convergencia que aumenta con la BL, luego tiende a caer en más minimos locales.

8.9. AM-(10,0.1)

Seed=365 1278846	Sonar		Wdbc		Spambase	
	%_clas	T	*%_clas*	T	*%_clas*	T
Partición 1-1	87.5173	16.6481	97.8929	58.4112	89.1304	77.0596
Partición 1-2	87.9380	17.0480	96.3071	59.1412	85.2174	75.6119
Partición 2-1	84.1840	16.6187	97.0101	65.8769	87.3913	74.4656
Partición 2-2	89.8983	17.1796	95.9550	57.3174	89.1304	76.6949
Partición 3-1	90.8599	16.5407	96.4875	59.4396	87.1739	77.5491
Partición 3-2	87.9288	16.9095	96.1311	59.4781	88.9130	76.4515
Partición 4-1	93.7171	16.6529	97.3653	58.2263	86.3043	77.0760
Partición 4-2	85.0254	16.9562	96.3071	59.9871	89.5652	77.7683
Partición 5-1	88.9921	16.7295	96.4863	59.3729	88.0435	73.5126
Partición 5-2	88.4235	16.8479	97.3604	59.4760	88.6957	74.2743
Media	88.4484	16.8131	96.7303	59.6727	87.9565	76.0464
Mínimo	84.1840	16.5407	95.9550	57.3174	85.2174	73.5126
Máximo	93.7171	17.1796	97.8929	65.8769	89.5652	77.7683
Desviación	2.7340	0.2088	0.6383	2.3172	1.4133	1.4960
Respecto al algoritmo Relief:						
Media	3.7411	16.8123	1.2826	59.6697	2.2609	76.0430
Mínimo	3.8558	16.5400	1.7581	57.3145	1.7391	73.5094
Máximo	6.7222	17.1783	1.4085	65.8737	1.3043	77.7647
Desviación	0.6765	0.2086	-0.1632	2.3171	-0.0096	1.4959

Figura 8.9: Tabla de resultados para el algoritmo AM-(10,0.1).

La calidad de la tasa de acierto es prácticamente idéntica a la variante anterior, sin embargo disminuye el tiempo de ejecición al disminuir el número de optimizaciones locales, y también disminuye la desviación consecuentemente.

8.10. AM-(10,0.1mej)

Seed=365 1278846	Sonar		Wdbc		Spambase	
	%_clas	T	*%_clas*	T	*%_clas*	T
Partición 1-1	87.9935	15.8409	97.1911	57.8067	90.4348	78.0847
Partición 1-2	85.5294	16.2087	96.1317	57.8895	85.4348	77.4645
Partición 2-1	85.6403	15.8658	97.5389	59.3358	86.3043	77.2916
Partición 2-2	87.9658	16.8554	95.9550	61.2881	87.8261	79.9147
Partición 3-1	91.3176	16.1404	96.6630	57.1269	90.2174	84.3739
Partición 3-2	87.9380	16.8511	96.1317	59.6668	87.1739	75.7888
Partición 4-1	93.7448	16.5904	96.8384	60.8335	87.3913	74.4746
Partición 4-2	88.4050	16.9247	95.4281	61.3521	90.8696	75.0746
Partición 5-1	86.1442	16.4053	96.6623	59.4849	88.2609	74.9076
Partición 5-2	88.8997	16.9050	97.7131	59.6373	88.0435	74.2397
Media	88.3578	16.4588	96.6253	59.4422	88.1957	77.1615
Mínimo	85.5294	15.8409	95.4281	57.1269	85.4348	74.2397
Máximo	93.7448	16.9247	97.7131	61.3521	90.8696	84.3739
Desviación	2.5565	0.4275	0.7270	1.4734	1.8045	3.1205
Respecto al algoritmo Relief:						
Media	3.6505	16.4579	1.1776	59.4392	2.5000	77.1581
Mínimo	5.2012	15.8402	1.2312	57.1240	1.9565	74.2365
Máximo	6.7499	16.9234	1.2287	61.3489	2.6087	84.3703
Desviación	0.4991	0.4274	-0.0746	1.4733	0.3815	3.1203

Figura 8.10: Tabla de resultados para el algoritmo AM-(10,0.1mej).

La calidad de la tasa de acierto es prácticamente idéntica a la variante anteriores, el tiempo de ejecución es muy similar a la variante AM-(10,0.1), ya que se realizan el mismo número de optimizaciones locales.

8.11. Globales

Seed=3651278846	Sonar		Wdbc		Spambase	
	%_clas	T	*%_clas*	T	*%_clas*	T
1-NN	82.9279	1.45E-05	95.2544	2.75E-05	82.9348	2.16E-05
RELIEF	84.7074	0.0008238	95.4477	0.0029823	85.6957	0.0033676
BL	85.8595	1.2107	95.9224	2.2620	86.1739	6.4060
AGG-BLX	87.7388	10.2135	96.5199	40.3752	88.3913	47.5877
AGG-CA	86.104	10.2091	96.7135	40.2144	87.6957	47.8338
AGE-BLX	87.2561	9.88402	96.6428	39.6574	88.6304	46.2728
AGE-CA	86.6782	9.86628	96.6077	41.5765	88.3478	45.962
AM-(10,1.0)	88.9824	19.2929	96.8185	77.1955	89.2391	91.9642
AM-(10,0.1)	88.4485	16.8131	96.7303	59.6727	87.9565	76.0464
AM-(10,0.1mej)	88.3578	16.4588	96.6253	59.4421	88.1957	77.1615

Figura 8.11: Tabla de resultados globales.

En definitiva a medida que vamos mejorando la explotación y exploración del entorno los algoritmos ofrecen mejores tasa de clasificación, a costo de un aumento considerable del tiempo de ejecución. Podemos diferenciar que los algoritmos genéticos presentan una mayor calidad, siendo las variantes con el operador $BLX - \sigma$ las que mejores resultados presentan. Los algoritmos meméticos terminan de añadir la explotación de la que los algoritmos genéticos carecen, aumentando la convergencia y posibilidad de caer antes en un mínimo local. Entre los algoritmos meméticos es dificil decantarse por uno dado que las diferencias son mínimas y pueden ser producidas simplemente por la aleatoriedad de los alrgoritmos, habría que realizar un test T para afirmar esto con seguridad. Por otro lado la BL ejecutada independientemente proporciona buenos resultados con un tiempo de ejecución rápido. Todos los algoritmos suponen una mejora frente al algoritmo greedy Relief, a expensas de un tiempo de ejecución mucho mayor.

Referencias

[1] http://sci2s.ugr.es/graduateCourses/Metaheuristicas.

[2] http://www.cplusplus.com/.