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

Memoria Práctica 2.b

Técnicas basadas en Trayectorias y Evolución Diferencial para el Problema del Aprendizaje de Pesos en Características

Algoritmos a desarrollar:
1-nn
relief
bl
agg-blx
am-(10,0.1mej)
es
ils
de/rand/1
de/current-to-best/1

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

29 de mayo de 2017

Índice

Índice de figuras

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, teniendo en cuenta aquellas soluciones que descarten un mayor número de características. 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 sigue.

dados:

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

$$tasa\_red = 100 \cdot \frac{\text{nº valores } w_i = 0}{\text{nº características}}$$

maximizar:

$$fun\_obj = tasa\_clas \cdot \alpha + tasa\_red \cdot (1-\alpha), \quad \alpha = 0{,}5$$

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 disjuntos aleatorios divididos al 20 % para poder aplicar la técnica de validación $5fcv$ (*5 fold cross validation)*, consistente en entrenar sobre el 80 % de los datos (validando la función objetivo mediante *leave-one-out*) y validar sobre el 20 % restante, es decir entrenamos sobre 4 conjuntos y validamos en el sobrante, dando lugar a 5 posibilidades distintas.

1.2. Nota sobre nueva función objetivo

La nueva función objetivo presenta un error, al considerar que los pesos menores a 0.1 descartan la característica los algoritmos pueden construir soluciones ponderando los pesos con diferencias mínimas, resultando en que todos los pesos son menores a 0.1, lo que produce una $tasa\_red$ igual al 100 % y altos porcentajes de clasificación, luego en la implementación elegida la $tasa\_red$ considerará nulas las características iguales a 0.

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* y maximizar la simplicidad, entendiento por simple aquella solución que use menos características. 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
  $\quad clase = ClasificaDato(A_i, W)$
  $\quad$ Si $clase ==$ a la clase de A_i
  $\quad\quad numeroAciertos + 1$
  End
  Devolver $tasa\_clas = 100 \cdot numeroAciertos/N_d$

  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:

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

La simplicidad de la solución se obtiene como se muestra a continuación:

$nCaracteristicasDescartadas = 0$
Begin desde $i = 0$ hasta N_c
    Si $W_i = 0$
        $nCaracteristicasDescartadas + 1$
End
Devolver $tasa\_red = 100 \cdot nCaracteristicasDescartadas/N_c$

Siendo el resultado final de la función objetivo la suma ponderada al 50 % de $tasa\_class$ y $tasa\_red$.

- 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), cuando se hayan evaluado un número de soluciones distintas o bien dado cuando se realicen un número de iteraciones determinado, especificado concretamente en la sección de cada algoritmo.
- 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 $w_{posicion} > 1, w_{posicion} = 1$

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 el valor de la función objetivo 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. Algoritmo Genético AGG-BLX

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. El operador de mutación 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

El operador de selección, se usará el torneo binario, realizamos tantos torneos como el tamaño de la población. 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
    $padre = rival_2$

El 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 $H_1, \quad H_2$

Luego el esquema general que sigue el algoritmo genético es el siguiente:

$n_{evaluaciones} = 0$
Begin desde $i = 0$ hasta $tam\_poblacion$
    $P_i^0 = generar\_solucion\_inicial$
End
Begin mientras $n_{evaluaciones} < 15000$
    $reproduce()$ //Selección y cruce
    $muta()$
    $integra()$ //Sustitución
End
Devolver mejor solución encontrada

Donde $n_{evaluaciones}$ se aumenta cada vez que se genera una nueva solución, ya sea en la inicialización, cruce o mutación.

6. Algoritmo Memético AM(10,0.1mej)

Se ha desarrollado a partir del algoritmo generacional con el operador de cruce $BLX - \sigma$(AGG-BLX), implementando que cada 10 generaciones se aplique la BL sobre los 0,1$\cdot N$ mejores cromosomas de la población actual (N es el tamaño de ésta).

El algoritmo memético redefine el método de inserción definido en el algoritmo Generacional para aplicar la BL, ya que tenemos que ordenar la población para aplicar la BL a los mejores cromosomas, calculando a priori el número de BLs a realizar. Contabilizamos el número de generación para aplicar la BL cada 10 generaciones, siendo $0{,}1 \cdot N$ el número de optimizaciones locales a realizar, con N igual al tamaño de la población. 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 5.

7. Algoritmo de Enfriamiento Simulado

Componentes del algoritmo:

- Esquema de enfriamiento, Cauchy modificado:

$$T_k = \frac{T_{k-1}}{1 + \beta \cdot T_{k-1}} \qquad \beta = \frac{T_0 - T_f}{M \cdot T_0 \cdot T_f}, \qquad k \geq 1$$

  donde M es el número de iteraciones a realizar, T_0 es la temperatura inicial y T_f es la temperatura final. Se ha denominado $funcG(Temperatura\_anterior)$ a la función que actualiza la temperatura en cada iteración, recibiendo la temperatura anterior como parámetro. T_0, T_f, M y β se definen al inicio del algoritmo.

- Operador de Vecino y exploración del entorno para $L(T)$, en cada iteración del bucle interno se aplica el operador $Mov(W, \sigma)$ sobre una característica aleatoria para generar un vecino que será comparado con la solución actual.

- Condición de enfriamiento $L(T)$, Se enfriará la temperatura, finalizando la iteración actual del bucle interno, bien cuando se haya generado un número máximo de vecinos $max_{vecinos}$ o bien cuando se haya aceptado un número máximo de vecinos max_{exitos}. Se ha denominado $funcL()$ a la función que implementa dicha operación, devolviendo $falso$ en caso de que se cumpla la condicion de parada.

  Begin $funcL()$
  Devolver $n_{vecinos} < max_{vecinos}$ && $n_{exitos} < max_{exitos}$

- Condición de parada, el algoritmo finaliza tras realizar M iteraciones del bucle externo o cuando el número de exitos en el enfriamiento actual sea igual a 0.

Los parámetros usados son los siguientes:

- $T_f = 0{,}001$
- $\mu = \phi = \sigma = 0{,}3$

- $max_{vecinos} = 10 \cdot n$
- $max_{exitos} = 0{,}1 \cdot max_{vecinos}$
- $M = 15000/max_{vecinos}$

La implementación desarrollada se describe a continuación:

$S = solucion\_inicial$
$initT()$
Inicializar parámetros constantes$(\beta, M, max_{vecinos}, max_{exitos})$
$val_S = valoracion(S)$
$actual = S$
$val_{actual} = val_S$
Begin hacer
    Begin $n_{vecinos} = 0$ y $n_{exitos} = 0$ mientras $funcL()$ hacer
        $vecino = Mov(actual, \sigma)$ sobre una característica aleatoria
        $val_{vecino} = valoracion(vecino)$
        $n_{vecinos} + 1$
        $difF = val_{actual} - val_{vecino}$
        Si $aceptaVecino()$
            $actualizaSol()$
            $n_{exitos} + 1$
    End
    $funcG(T_k)$
End mientras $T_f \leq T_k$ y $n_{exitos}! = 0$
Devolver S

Donde $initT()$ inicializa las temperaturas de la siguiente manera:

$T_0 = \frac{\mu \cdot C(S_0)}{-ln(\phi)}$
$T_f = constante$
$T_k = T_0$
Siendo $T_f < T_0$

Donde $aceptaVecino()$ realiza la siguiente operación:

Si $difF < 0$
    Devolver $true$
Si no
    Si $U(0,1) \leq exp(-difF/T_k)$
        Devolver $true$
    Si no
        Devolver $false$

Donde *actualizaSol*() actualiza la solución actual y la mejor solución en caso de ser necesario, como se muestra:

$actual = vecino$
$val_{actual} = val_{vecino}$
Si $val_{actual} > val_S$
    $S = actual$
    $val_S = val_{actual}$

8. Algoritmo de BL Reiterada (ILS)

El algoritmo ILS consiste en generar una solución aleatoria y optimizarla mediante BL, una vez hecho esto se evalua si es mejor que la mejor solución encontrada hasta ahora y se realiza una mutación sobre la mejor solución, este proceso se repite una serie de veces, devolviendo la mejor solución encontrada.

El operador de mutación de ILS consiste en aplicar el operador $Mov(W, \sigma)$ sobre t características aleatorias de W. En pseudocódigo la función *mutacion*() se describe como sigue:

Begin desde $i = 0$ hasta t hacer:
    $k = random\_uniform\_distribuition(0, nCaracteristicas)$
    $Mov(W, \sigma)$ en la característica W_k
End
Devolver W

Los parámetros usados son los siguientes:

- $n_{ejecuciones} = 15$
- $t = 0{,}1 \cdot n$
- $\sigma = 0{,}4$
- $max_{evaluaciones} = 1000$

El algoritmo implementado es el siguiente:

$S = solucion\_inicial()$
$val_S = valoracion(S)$
Begin desde $i = 0$ hasta 14 hacer:
    $s1 = S$
    $mutacion()$ sobre $s1$
    $BL()$ sobre $s1$
    $val_{s1} = valoracion(s1)$
    Si $val_{s1} > val_S$
        $S = s1$
        $val_S = val_{s1}$
End
Devolver S

9. Algoritmos de Evolución Diferencial

La DE es un modelo evolutivo para optimización con parámetros reales que enfatiza en la mutación y utiliza un operador de cruce/recombinación a posteriori. Se han desarrollado dos variantes:

- $DE/Rand/1$, que obtiene el vector de mutación mediante la fórmula:

$$V_{i,G} = X_{r1,G} + F \cdot (X_{r2,G} - X_{r3,G})$$

  Los indices $r1$, $r2$ y $r3$ de cada individuo i en cada generación G se escogen aleatoriamente de forma mutuamente excluyente (incluyendo el vector i-ésimo). En pseudocódigo la función $muta(i)$ que aplica dicha mutación en V para la característica i se define como sigue:

  $V[i] = Pact[r1][i] + F \cdot (Pact[r2][i] - Pact[r3][i])$
  Truncar $V[i]$ al intervalo $[0, 1]$

  Donde $Pact$ es la población actual con n agentes, cada agente formado por n_c características $Pact[n][n_c]$.

- $DE/current\text{-}to\text{-}best/1$, cuyo vector de mutación se obtiene como sigue:

$$V_{i,G} = X_{i,G} + F \cdot (X_{best,G} - X_{i,G}) + F \cdot (X_{r1,G} - X_{r2,G})$$

  Los indices $r1$ y $r2$ de cada individuo i en cada generación G se escogen aleatoriamente de forma mutuamente excluyente (incluyendo el vector i-ésimo). $X_{best,G}$ denota el mejor vector en la generación G.En pseudocódigo la función $muta(i)$ que aplica dicha mutación en V para la característica i se define como sigue:

$V[i] = V[i] + F \cdot (S[i] - V[i]) + F \cdot (Pact[r1][i] - Pact[r2][i])$
Truncar $V[i]$ al intervalo $[0, 1]$

Donde $Pact$ es la población actual con n agentes, cada agente formado por n_c características $Pact[n][n_c]$ y S es el mejor agente en la poblacion $Pact$.

Esta es la única diferencia entre ambos algoritmos, ahora procederemos a describir el resto de componentes comunes a ambos algoritmos:

- Esquema de recombinación binomial, donde un individuo en la población será recombinado como sigue:

$$u_{j,i,g} = \begin{cases} v_{j,i,g} & \text{si } rand_j[0,1] < CR \text{ o } j = j_{rand} \\ x_{j,i,g} & \end{cases}$$

  Esta fórmula nos dice que un agente X_i irá mutando cada una de sus características o no, el pseudocódigo asociado se describe en el esquema general del algoritmo.

- Esquema de remplazamiento uno a uno, cada Agente X_i de la población siguiente $Psig[i]$ compite con $Pact[i]$, ganando el Agente con mejor valoración.

Los parámetros para las ejecuciones establecidos son los siguientes:

- Tamaño de población $nP = 50$
- Probabilidad de cruce $CR = 0{,}5$
- $F = 0{,}5$
- $max_{evaluaciones} = 15000$

El esquema general y el pseudocódigo asociado es el siguiente:

Inicializar constantes($max_{evaluaciones}$, F, CR, $n_{caracteristicas}$, nP)
$n_{evaluaciones} = 0$
Begin desde $i = 0$ hasta nP hacer:
    $Pact[i] = solucion_inicial()$
    $valora(Pact[i])$
    $n_{evaluaciones} + 1$
End
$actualizaSol()$
Begin mientras $n_{evaluaciones} < max_{evaluaciones}$ hacer:
    Begin desde $i = 0$ hasta nP hacer:
        $selecPadres(i)$
        $V = Pact[i]$
        $jRand = uniform\_random\_distribution(0, nP - 1)$
        Begin desde $j = 0$ hasta $n_{caracteristicas}$ hacer:
            Si $U(0, 1) < CR$ o $j == jRand$
                $muta(j)$
        End
        $Psig[i] = V$
        $valora(Psig[i])$
        $n_{evaluaciones} + 1$
    End
    $sustitucion()$
    $actualizaSol()$
End
Devolver S

La función $actualizaSol()$ recorre la población actual $Pact$ y guarda en S el Agente con mejor valoración.

La función $selecPadres(i)$ selecciona los indices para el Agente $Pact[i]$ según el algoritmo:

- $DE/Rand/1$:

    Begin hacer:
        $r1 = uniform\_random\_distribution(0, nP - 1)$
    End mientras $r1 == i$
    Begin hacer:
        $r2 = uniform\_random\_distribution(0, nP - 1)$
    End mientras $r2 == i$ y $r2 == r1$
    Begin hacer:
        $r3 = uniform\_random\_distribution(0, nP - 1)$
    End mientras $r3 == i$ y $r3 == r2$ y $r3 == r1$

- $DE/current\text{-}to\text{-}best/1$:

  Begin hacer:
  $r1 = uniform\_random\_distribution(0, nP - 1)$
  End mientras $r1 == i$
  Begin hacer:
  $r2 = uniform\_random\_distribution(0, nP - 1)$
  End mientras $r2 == i$ y $r2 == r1$

La función $sustitucion()$ se define como sigue:

Begin desde $i = 0$ hasta nP hacer:
Si $Pact[i] < Psig[i]$
$Pact[i] = Psig[i]$
End

10. Procedimiento considerado para el desarrollo de la práctica

He continuado desarrollando desde el esqueleto de la práctica1, los algoritmos han sido implementados en c++ basandome como apoyo en las transparencias de teoria, seminarios y el guión de prácticas. He modificado el programa que crea las particiones para que puedan ser usadas con el nuevo estandar de validación 5fcv. Luego procedí ha actualizar la función objetivo y desarrollar los nuevos algoritmos. Finalizando con las ejecuciones de los algoritmos y y su analisis posterior. Una vez más el material de apoyo utilizado han sido los apuntes de teoría y prácticas[1], e iformación sobre el algoritmo de Evolución Diferencial[2].

11. Resultados

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

11.1. 1-NN

Seed	Sonar				Wdbc				Spambase			
3651278847	%_clas	%_red	Agr.	T	%_clas	%_red	Agr.	T	%_clas	%_red	Agr.	T
Particion 1	80.4878	0.0000	40.2439	6.88E-05	93.8053	0.0000	46.9027	5.30E-05	86.8132	0.0000	43.4066	4.57E-05
Particion 2	85.3659	0.0000	42.6830	1.37E-05	96.4602	0.0000	48.2301	2.68E-05	80.2198	0.0000	40.1099	1.82E-05
Particion 3	80.4878	0.0000	40.2439	1.41E-05	94.6903	0.0000	47.3452	1.73E-05	82.4176	0.0000	41.2088	1.80E-05
Particion 4	87.8049	0.0000	43.9025	1.60E-05	94.6903	0.0000	47.3452	1.36E-05	92.3077	0.0000	46.1539	2.20E-05
Particion 5	90.9091	0.0000	45.4546	1.53E-05	97.4359	0.0000	48.7180	2.35E-05	81.2500	0.0000	40.6250	2.19E-05
Media	85.0111	0.0000	42.5056	2.56E-05	95.4164	0.0000	47.7082	2.68E-05	84.6017	0.0000	42.3008	2.51E-05
Mínimo	80.4878	0.0000	40.2439	1.37E-05	93.8053	0.0000	46.9027	1.36E-05	80.2198	0.0000	40.1099	1.80E-05
Máximo	90.9091	0.0000	45.4546	6.88E-05	97.4359	0.0000	48.7180	5.30E-05	92.3077	0.0000	46.1539	4.57E-05
Desviación	4.5727	0.0000	2.2863	2.42E-05	1.4848	0.0000	0.7424	1.55E-05	4.9869	0.0000	2.4935	1.16E-05

Figura 11.1: Tabla de resultados para el algoritmo 1-NN.

Estos son los resultados del clasificador con todas las características tomadas con igual importancia, obviamente la valoración en cuanto a simplicidad es 0, ya que todos los pesos están a 1. En cuanto al porcentaje de clasificación obtiene unos resultados bastante buenos dada la rapidez y simplicidad del algoritmo. Según los resultados podemos decir que la base Sonar y Spambase requieren de mayor inteligencia para ajustar los pesos de las características, mientras que en Wdbc parecen tener una importancia similar todas las características ya que los resultados de clasificación son muy buenos.

11.2. Relief

Seed	Sonar				Wdbc				Spambase			
3651278847	%_clas	%_red	Agr.	T	%_clas	%_red	Agr.	T	%_clas	%_red	Agr.	T
Particion 1	78.0488	5.0000	41.5244	1.15E-02	94.6903	0.0000	47.3452	4.70E-02	90.1099	5.2632	47.6865	5.12E-02
Particion 2	85.3659	6.6667	46.0163	1.15E-02	95.5752	0.0000	47.7876	4.43E-02	86.8132	5.2632	46.0382	5.15E-02
Particion 3	82.9268	5.0000	43.9634	1.22E-02	92.9204	0.0000	46.4602	4.47E-02	81.3187	5.2632	43.2909	5.17E-02
Particion 4	82.9268	6.6667	44.7967	1.13E-02	95.5752	0.0000	47.7876	4.45E-02	94.5055	1.7544	48.1299	5.10E-02
Particion 5	93.1818	3.3333	48.2576	1.12E-02	95.7265	0.0000	47.8633	4.74E-02	84.3750	8.7719	46.5735	5.00E-02
Media	84.4900	5.3333	44.9117	1.15E-02	94.8975	0.0000	47.4488	4.56E-02	87.4245	5.2632	46.3438	5.11E-02
Mínimo	78.0488	3.3333	41.5244	1.12E-02	92.9204	0.0000	46.4602	4.43E-02	81.3187	1.7544	43.2909	5.00E-02
Máximo	93.1818	6.6667	48.2576	1.22E-02	95.7265	0.0000	47.8633	4.74E-02	94.5055	8.7719	48.1299	5.17E-02
Desviación	5.5383	1.3944	2.4900	3.76E-04	1.1787	0.0000	0.5894	1.50E-03	5.1065	2.4811	1.9013	6.33E-04

Figura 11.2: Tabla de resultados para el algoritmo Relief.

El algoritmo Relief ya tiene características descartadas, pero al no presentar ninguna inteligencia que busque maximizar el número de características descartadas, la valoración de simplicidad sigue siendo muy baja. Presenta resultados muy similares al clasificador 1-NN en cuanto a clasificación y ejecuciones más lentas, del orden de 10^2 veces más lento.

11.3. BL

Seed	Sonar				Wdbc				Spambase			
3651278847	%_clas	%_red	Agr.	T	%_clas	%_red	Agr.	T	%_clas	%_red	Agr.	T
Particion 1	80.4878	60.0000	70.2439	5.9701	92.0354	90.0000	91.0177	34.4237	90.1099	82.4561	86.2830	75.0437
Particion 2	87.8049	70.0000	78.9025	7.1870	95.5752	60.0000	77.7876	13.9783	82.4176	70.1754	76.2965	33.9457
Particion 3	82.9268	65.0000	73.9634	7.6242	90.2655	90.0000	90.1328	17.2473	83.5165	85.9649	84.7407	45.8030
Particion 4	82.9268	68.3333	75.6301	5.4771	95.5752	90.0000	92.7876	28.7689	84.6154	92.9825	88.7990	76.3496
Particion 5	84.0909	63.3333	73.7121	7.3633	94.0171	83.3333	88.6752	19.0287	85.4167	70.1754	77.7961	61.7779
Media	83.6474	65.3333	74.4904	6.7243	93.4937	82.6667	88.0802	22.6894	85.2152	80.3509	82.7830	58.5840
Mínimo	80.4878	60.0000	70.2439	5.4771	90.2655	60.0000	77.7876	13.9783	82.4176	70.1754	76.2965	33.9457
Máximo	87.8049	70.0000	78.9025	7.6242	95.5752	90.0000	92.7876	34.4237	90.1099	92.9825	88.7990	76.3496
Desviación	2.6694	3.9791	3.1497	0.9430	2.3177	12.9957	5.9433	8.5667	2.9610	10.0323	5.4594	18.4874
Comparación con algoritmo Relief (Alg-Relief)												
Media	-0.8426	60.0000	29.5787	6.7128	-1.4038	82.6667	40.6314	22.6438	-2.2092	75.0877	36.4392	58.5329
Mínimo	2.4390	56.6667	28.7195	5.4659	-2.6549	60.0000	31.3274	13.9340	1.0989	68.4210	33.0056	33.8957
Máximo	-5.3769	63.3333	30.6449	7.6121	-0.1513	90.0000	44.9244	34.3763	-4.3956	84.2106	40.6690	76.2979
Desviación	-2.8689	2.5847	0.6596	0.9426	1.1390	12.9957	5.3540	8.5652	-2.1455	7.5512	3.5581	18.4868

Figura 11.3: Tabla de resultados para el algoritmo BL.

Ya podemos observar como la valoración de simplicidad ha aumentado considerablemente respecto al algoritmo Relief, debido a que la BL ya tiene en consideración maximizar la simplicidad, esto tambien ha conllevado una ligera perdida en la tasa de clasificación. El tiempo ha aumentado considerablemente, podemos observar como crece el tiempo de ejecución a medida que aumenta el tamaño de los datos y las características, llegando a ser 58 segundos más lento de media en Spambase que el algoritmo Relief. La desviación estandar ha disminuido y esto es debido a que tiene mayor exploración que Relief, permitiendole alcanzar soluciones de calidad más similar en las distintas ejecuciones.

11.4. AGG-BLX

Seed	Sonar				Wdbc				Spambase			
3651278847	%_clas	%_red	Agr.	T	%_clas	%_red	Agr.	T	%_clas	%_red	Agr.	T
Particion 1	87.8049	46.6667	67.2358	34.2515	89.3805	66.6667	78.0236	141.9240	85.7143	59.6491	72.6817	144.5970
Particion 2	95.1220	50.0000	72.5610	34.2632	97.3451	66.6667	82.0059	141.8780	85.7143	50.8772	68.2958	127.7350
Particion 3	82.9268	35.0000	58.9634	34.1949	92.9204	63.3333	78.1269	111.0890	74.7253	57.8947	66.3100	139.9920
Particion 4	87.8049	60.0000	73.9025	34.1392	92.9204	63.3333	78.1269	112.4180	87.9121	52.6316	70.2719	142.8300
Particion 5	86.3636	41.6667	64.0152	32.9356	97.4359	40.0000	68.7180	117.2460	76.0417	57.8947	66.9682	139.8810
Media	88.0044	46.6667	67.3356	33.9569	94.0005	60.0000	77.0002	124.9110	82.0215	55.7895	68.9055	139.0070
Mínimo	82.9268	35.0000	58.9634	32.9356	89.3805	40.0000	68.7180	111.0890	74.7253	50.8772	66.3100	127.7350
Máximo	95.1220	60.0000	73.9025	34.2632	97.4359	66.6667	82.0059	141.9240	87.9121	59.6491	72.6817	144.5970
Desviación	4.4501	9.3541	6.1554	0.5730	3.4156	11.3039	4.9305	15.6780	6.1434	3.8034	2.5975	6.6079
Comparación con algoritmo Relief (Alg-Relief)												
Media	3.5144	41.3333	22.4239	33.9453	-0.8971	60.0000	29.5515	124.8654	-5.4029	50.5263	22.5617	138.9559
Mínimo	4.8780	31.6667	17.4390	32.9244	-3.5399	40.0000	22.2578	111.0447	-6.5934	49.1228	23.0191	127.6850
Máximo	1.9402	53.3333	25.6449	34.2510	1.7094	66.6667	34.1427	141.8766	-6.5934	50.8772	24.5518	144.5453
Desviación	-1.0882	7.9597	3.6654	0.5727	2.2369	11.3039	4.3411	15.6765	1.0369	1.3223	0.6962	6.6072

Figura 11.4: Tabla de resultados para el algoritmo AGG-BLX.

Observamos nuevamente que, en cuanto la simplicidad, mejora considerablemente al algoritmo Relief. El tiempo es mucho más lento, y esta vez la desviación no mejora, debido a que converge muy rápido no permitiendo alcanzar soluciones de la misma calidad en distintas ejecuciones, esto se observa claramente en el conjunto Spambase, donde el mínimo y el máximo de clasificación son 74 % y 87 % respectivamente. La desviación en la tasa de simplicidad fluctua mucho más, debido a la rápida convergencia y poca capacidad de exploración en éste ámbito. La poca capacidad de exploración en la simplicidad se debe a que los cambios que se producen en las mutaciones y sobre todo en la operación de cruce son muy progresivos, lo que dificulta alcanzar el valor 0 en los pesos.

11.5. AM-(1,0.1mej)

Seed	Sonar				Wdbc				Spambase			
3651278846	%_clas	%_red	Agr.	T	%_clas	%_red	Agr.	T	%_clas	%_red	Agr.	T
Particion 1	75.6098	65.0000	70.3049	30.5597	87.6106	90.0000	88.8053	122.4440	86.8132	66.6667	76.7400	132.8900
Particion 2	87.8049	68.3333	78.0691	27.4788	93.8053	83.3333	88.5693	118.9540	84.6154	82.4561	83.5358	141.1590
Particion 3	80.4878	61.6667	71.0773	28.5200	92.9204	76.6667	84.7936	120.1150	79.1209	78.9474	79.0342	136.3510
Particion 4	80.4878	70.0000	75.2439	28.0579	91.1504	86.6667	88.9086	120.7370	85.7143	77.1930	81.4537	152.3200
Particion 5	86.3636	66.6667	76.5152	30.0411	87.1795	93.3333	90.2564	137.0150	82.2917	78.9474	80.6196	129.7660
Media	82.1508	66.3333	74.2421	28.9315	90.5332	86.0000	88.2666	123.8530	83.7111	76.8421	80.2766	138.4972
Mínimo	75.6098	61.6667	70.3049	27.4788	87.1795	76.6667	84.7936	118.9540	79.1209	66.6667	76.7400	129.7660
Máximo	87.8049	70.0000	78.0691	30.5597	93.8053	93.3333	90.2564	137.0150	86.8132	82.4561	83.5358	152.3200
Desviación	4.9506	3.2059	3.4035	1.3158	3.0239	6.4118	2.0503	7.4650	3.0634	6.0009	2.5582	8.8096
Comparación con algoritmo Relief (Alg-Relief)												
Media	-2.3392	61.0000	29.3304	28.9200	-4.3643	86.0000	40.8179	123.8074	-3.7134	71.5790	33.9328	138.4461
Mínimo	-2.4390	58.3334	28.7805	27.4676	-5.7409	76.6667	38.3334	118.9097	-2.1978	64.9123	33.4490	129.7160
Máximo	-5.3769	63.3333	29.8115	30.5475	-1.9212	93.3333	42.3932	136.9676	-7.6923	73.6842	35.4058	152.2683
Desviación	-0.5877	1.8114	0.9134	1.3154	1.8451	6.4118	1.4610	7.4635	-2.0431	3.5198	0.6569	8.8090

Figura 11.5: Tabla de resultados para el algoritmo AM-(1,0.1mej).

Al introducir la BL se logra alcanzar mejores soluciones de simplicidad que la variante generacional, manteniendo resultados similares de tasa de clasificación. En cuanto al algoritmo Relief el valor de la función objetivo es superior en todas las mediciones, sacrificando tasa de clasificación. Tiene una desviación baja debido a que logra alcanzar soluciones de calidad similar en las distintas ejecuciones gracias a la capacidad de exploración que añade la búsqueda local.

11.6. ES

Seed	Sonar				Wdbc				Spambase			
3651278846	%_clas	%_red	Agr.	T	%_clas	%_red	Agr.	T	%_clas	%_red	Agr.	T
Particion 1	78.0488	28.3333	53.1911	8.7600	95.5752	43.3333	69.4543	97.1158	84.6154	33.3333	58.9744	122.9400
Particion 2	92.6829	33.3333	63.0081	11.7281	99.1150	40.0000	69.5575	94.3193	80.2198	31.5789	55.8994	61.0693
Particion 3	87.8049	25.0000	56.4025	8.7576	96.4602	40.0000	68.2301	96.3490	76.9231	31.5789	54.2510	89.9662
Particion 4	80.4878	30.0000	55.2439	8.8505	97.3451	46.6667	72.0059	63.8551	80.2198	36.8421	58.5310	90.1038
Particion 5	81.8182	26.6667	54.2425	5.5966	94.8718	36.6667	65.7693	64.0461	83.3333	33.3333	58.3333	86.8166
Media	84.1685	28.6667	56.4176	8.7386	96.6735	41.3333	69.0034	83.1371	81.0623	33.3333	57.1978	90.1792
Mínimo	78.0488	25.0000	53.1911	5.5966	94.8718	36.6667	65.7693	63.8551	76.9231	31.5789	54.2510	61.0693
Máximo	92.6829	33.3333	63.0081	11.7281	99.1150	46.6667	72.0059	97.1158	84.6154	36.8421	58.9744	122.9400
Desviación	5.9632	3.2059	3.8715	2.1692	1.6515	3.8006	2.2684	17.5447	3.0139	2.1487	2.0367	21.9773
Comparación con algoritmo Relief (Alg-Relief)												
Media	-0.3215	23.3333	11.5059	8.7270	1.7759	41.3333	21.5546	83.0915	-6.3622	28.0701	10.8540	90.1281
Mínimo	0.0000	21.6667	11.6667	5.5854	1.9514	36.6667	19.3091	63.8108	-4.3956	29.8245	10.9601	61.0193
Máximo	-0.4989	26.6666	14.7505	11.7159	3.3885	46.6667	24.1427	97.0684	-9.8901	28.0702	10.8444	122.8883
Desviación	0.4249	1.8114	1.3815	2.1689	0.4728	3.8006	1.6791	17.5432	-2.0926	-0.3324	0.1354	21.9767

Figura 11.6: Tabla de resultados para el algoritmo ES.

El algoritmo de enfriamiento simulado parece no ajustarse bien al problema, le falta exploración para alcanzar soluciones mejores y no caer en mínimos locales, probablemente a que enfria demasiado rápido, y el operador de mutación no se ajusta demasiado bien a la nueva función objetivo. Al no tener exploración suficiente finaliza a menudo demasiado rápido, por la condición de parada cuando no se tiene éxito en un enfriamiento, lo corrobora las fluctuaciones en las mediciones de tiempo. Respecto al algoritmo Relief solo podemos concluir que mejora en el aspecto de clasificación, en cuanto a tasa de clasificación en los datos Sonar y Wdbc tiene resultados similares, perdiendo en Spambase con una tasa de clasificación menor de media en 6.36 puntos.

11.7. ILS

Seed	Sonar				Wdbc				Spambase			
3651278846	%_clas	%_red	Agr.	T	%_clas	%_red	Agr.	T	%_clas	%_red	Agr.	T
Particion 1	85.3659	76.6667	81.0163	25.3723	90.2655	93.3333	91.7994	100.4700	82.4176	87.7193	85.0685	118.0730
Particion 2	75.6098	81.6667	78.6383	25.6467	93.8053	93.3333	93.5693	101.2220	85.7143	89.4737	87.5940	127.1930
Particion 3	78.0488	83.3333	80.6911	25.2524	92.9204	93.3333	93.1269	100.7190	89.0110	92.9825	90.9968	129.4120
Particion 4	73.1707	80.0000	76.5854	25.7255	93.8053	93.3333	93.5693	100.5180	83.5165	94.7368	89.1267	126.0720
Particion 5	81.8182	78.3333	80.0758	24.5113	89.7436	93.3333	91.5385	99.4837	89.5833	92.9825	91.2829	123.1320
Media	78.8027	80.0000	79.4013	25.3016	92.1080	93.3333	92.7207	100.4825	86.0485	91.5790	88.8138	124.7764
Mínimo	73.1707	76.6667	76.5854	24.5113	89.7436	93.3333	91.5385	99.4837	82.4176	87.7193	85.0685	118.0730
Máximo	85.3659	83.3333	81.0163	25.7255	93.8053	93.3333	93.5693	101.2220	89.5833	94.7368	91.2829	129.4120
Desviación	4.8644	2.6352	1.8192	0.4824	1.9626	0.0000	0.9813	0.6328	3.2007	2.8827	2.5727	4.3774
Comparación con algoritmo Relief (Alg-Relief)												
Media	-5.6873	74.6667	34.4897	25.2901	-2.7895	93.3333	45.2719	100.4369	-1.3759	86.3158	42.4699	124.7253
Mínimo	-4.8781	73.3334	35.0610	24.5001	-3.1768	93.3333	45.0783	99.4394	1.0989	85.9649	41.7775	118.0230
Máximo	-7.8159	76.6666	32.7587	25.7133	-1.9212	93.3333	45.7061	101.1746	-4.9222	85.9649	43.1530	129.3603
Desviación	-0.6739	1.2408	-0.6708	0.4820	0.7838	0.0000	0.3919	0.6313	-1.9058	0.4017	0.6714	4.3768

Figura 11.7: Tabla de resultados para el algoritmo ILS.

Éste algoritmo se adapta muy bien a la función objetivo, al tener mutaciones más agresivas que el resto de algoritmos le permite descartar con mayor rapidez las carácteristicas con menor importancia, sacrificando explotación en el ajuste de los pesos del resto de características, esto podemos observarlo en como la tasa de clasificación pierde respecto a Relief. El tiempo de ejecución se mantiene similar al resto de algoritmos complejos, superando en gran medida a Relief. Podemos observar una muy baja desviación en las soluciones en cuanto a simplicidad, mientras que en clasificación fluctua en mayor medida debido a lo descrito anteriormente.

11.8. DE/Rand/1

Seed	Sonar				Wdbc				Spambase			
3651278846	%_clas	%_red	Agr.	T	%_clas	%_red	Agr.	T	%_clas	%_red	Agr.	T
Particion 1	75.6098	88.3333	81.9716	26.8366	90.2655	93.3333	91.7994	105.4800	90.1099	92.9825	91.5462	136.5630
Particion 2	73.1707	90.0000	81.5854	26.3444	93.8053	93.3333	93.5693	107.2750	83.5165	92.9825	88.2495	134.1970
Particion 3	73.1707	88.3333	80.7520	26.2902	90.2655	93.3333	91.7994	131.5460	87.9121	91.2281	89.5701	134.0870
Particion 4	85.3659	90.0000	87.6830	26.3019	92.9204	90.0000	91.4602	110.5180	81.3187	94.7368	88.0278	132.5260
Particion 5	79.5455	85.0000	82.2728	25.3822	89.7436	93.3333	91.5385	111.2720	88.5417	92.9825	90.7621	129.8440
Media	77.3725	88.3333	82.8529	26.2311	91.4001	92.6666	92.0334	113.2182	86.2798	92.9825	89.6311	133.4434
Mínimo	73.1707	85.0000	80.7520	25.3822	89.7436	90.0000	91.4602	105.4800	81.3187	91.2281	88.0278	129.8440
Máximo	85.3659	90.0000	87.6830	26.8366	93.8053	93.3333	93.5693	131.5460	90.1099	94.7368	91.5462	136.5630
Desviación	5.1729	2.0412	2.7597	0.5265	1.8313	1.4907	0.8721	10.5141	3.6979	1.2405	1.5354	2.4754
Comparación con algoritmo Relief (Alg-Relief)												
Media	-7.1175	83.0000	37.9412	26.2195	-3.4975	92.6666	44.5846	113.1726	-1.1447	87.7193	43.2873	133.3923
Mínimo	-4.8781	81.6667	39.2276	25.3710	-3.1768	90.0000	45.0000	105.4357	0.0000	89.4737	44.7368	129.7940
Máximo	-7.8159	83.3333	39.4254	26.8244	-1.9212	93.3333	45.7061	131.4986	-4.3956	85.9649	43.4163	136.5113
Desviación	-0.3654	0.6468	0.2696	0.5261	0.6526	1.4907	0.2827	10.5126	-1.4086	-1.2406	-0.3659	2.4748

Figura 11.8: Tabla de resultados para el algoritmo DE/Rand/1.

Este algoritmo también tiene fuertes mutaciones, lo que una vez más, permite al algoritmo alcanzar buena simplicidad sacrificando clasificación. Respecto al algoritmo Relief pierde en clasificación y gran en gran medida en la función objetivo, con un tiempo de ejecución mucho mayor. Al igual que ILS obtiene desviaciones bajas en la simplicidad de las soluciones.

11.9. DE/current-to-best/1

Seed	Sonar				Wdbc				Spambase			
3651278846	%_clas	%_red	Agr.	T	%_clas	%_red	Agr.	T	%_clas	%_red	Agr.	T
Particion 1	82.9268	53.3333	68.1301	26.9855	93.8053	76.6667	85.2360	111.1880	93.4066	45.6140	69.5103	136.9610
Particion 2	87.8049	50.0000	68.9025	26.9871	94.6903	46.6667	70.6785	112.0160	85.7143	47.3684	66.5414	133.9350
Particion 3	85.3659	51.6667	68.5163	27.9261	94.6903	63.3333	79.0118	110.4940	84.6154	52.6316	68.6235	131.3640
Particion 4	80.4878	58.3333	69.4106	27.4648	95.5752	70.0000	82.7876	117.9360	89.0110	47.3684	68.1897	150.9730
Particion 5	88.6364	56.6667	72.6516	26.2053	94.8718	70.0000	82.4359	114.0450	81.2500	52.6316	66.9408	134.3590
Media	85.0444	54.0000	69.5222	27.1138	94.7266	65.3333	80.0300	113.1358	86.7995	49.1228	67.9611	137.5184
Mínimo	80.4878	50.0000	68.1301	26.2053	93.8053	46.6667	70.6785	110.4940	81.2500	45.6140	66.5414	131.3640
Máximo	88.6364	58.3333	72.6516	27.9261	95.5752	76.6667	85.2360	117.9360	93.4066	52.6316	69.5103	150.9730
Desviación	3.3865	3.4561	1.8124	0.6405	0.6310	11.4504	5.6785	2.9956	4.6176	3.2822	1.2194	7.7788
Comparación con algoritmo Relief (Alg-Relief)												
Media	0.5543	48.6667	24.6105	27.1022	-0.1709	65.3333	32.5812	113.0902	-0.6250	43.8596	21.6173	137.4673
Mínimo	2.4390	46.6667	26.6057	26.1941	0.8849	46.6667	24.2183	110.4497	-0.0687	43.8596	23.2504	131.3140
Máximo	-4.5454	51.6666	24.3940	27.9139	-0.1513	76.6667	37.3728	117.8886	-1.0989	43.8597	21.3804	150.9213
Desviación	-2.1518	2.0616	-0.6776	0.6401	-0.5477	11.4504	5.0892	2.9941	-0.4889	0.8011	-0.6819	7.7781

Figura 11.9: Tabla de resultados para el algoritmo DE/current-to-best/1.

Esta versión del algoritmo DE lidia bastante peor con la simplicidad de las soluciones, esto se debe a que las soluciones iniciales aleatorias rara vez presentan pesos con valor igual a 0, lo que provoca que las mutaciones exploren más el espacio de clasificación que el espacio de simplicidad. Esto lo podemos observar en la figura11.9, donde la mitad de las soluciones iniciales estan inicialicadas al valor 0 en todos los pesos. Observamos un aumento de la tasa de clasificación respecto DE/Rand/1 por lo comentado anteriormente. Respecto a Relief obtiene tasas de clasificación muy similares y tasas de simplicidad notablemente mayores, al costo de un tiempo de ejecución mayor.

Seed	Sonar			
3651278846	%_clas	%_red	Agr.	T
Particion 1	75.6098	83.3333	79.4716	26.5722
Particion 2	75.6098	86.6667	81.1383	26.4194
Particion 3	75.6098	88.3333	81.9716	26.3516
Particion 4	80.4878	88.3333	84.4106	29.6643
Particion 5	79.5455	90.0000	84.7728	25.5254
Media	77.3725	87.3333	82.3529	26.9066
Mínimo	75.6098	83.3333	79.4716	25.5254
Máximo	80.4878	90.0000	84.7728	29.6643
Desviación	2.4366	2.5276	2.2368	1.5945

Figura 11.10: Tabla de resultados para el algoritmo DE/current-to-best/1, alterando soluciones iniciales.

11.10. Resultados Globales

Seed	Sonar				Wdbc				Spambase			
3651278846	%_clas	%_red	Agr.	T	%_clas	%_red	Agr.	T	%_clas	%_red	Agr.	T
1-NN	85.0111	0.0000	42.5056	6.88E-05	95.4164	0.0000	47.7082	2.68E-05	84.6017	0.0000	42.3009	2.51E-05
Relief	84.4900	5.3333	44.9117	1.15E-02	94.8975	0.0000	47.4488	4.56E-02	87.4245	5.2632	46.3438	5.11E-02
ES	84.1685	28.6667	56.4176	8.7386	96.6735	41.3333	69.0034	83.1371	81.0623	33.3333	57.1978	90.1792
ILS	78.8027	80.0000	79.4014	25.3016	92.1080	93.3333	92.7207	100.4825	86.0485	91.5790	88.8138	124.7764
DE/Rand/1	77.3725	88.3333	82.8529	26.2311	91.4001	92.6666	92.0334	113.2182	86.2798	92.9825	89.6312	133.4434
DE/current-to-best/1	85.0444	54.0000	69.5222	27.1138	94.7266	65.3333	80.0300	113.1358	86.7995	49.1228	67.9612	137.5184
BL	83.6474	65.3330	74.4902	6.7243	93.4973	82.6667	88.0820	22.6894	85.2152	80.3509	82.7831	58.5840
AGG-BLX	88.0044	46.6670	67.3357	33.9569	94.0005	60.0000	77.0003	124.9110	82.0215	55.7895	68.9055	139.0070
AM (1,0.1mej)	82.1508	66.3333	74.2421	28.9315	90.5332	86.0000	88.2666	123.8530	83.7111	76.8421	80.2766	138.4972

Figura 11.11: Tabla de resultados globales.

Comenzaremos con una comparativa global de los algoritmos, de peor a mejor, y terminaremos con unas conclusiones sobre el problema y la nueva función objetivo.

El clasificador 1-NN y Relief, como ya hemos observado, tienen unas valoraciones de simplicidad practicamente nulas y en consideración las valoraciones globales más bajas de todos los algoritmos. Luego sigue el algoritmo ES, que debido a la falta de exploración que acarrea no es capaz de encontrar soluciones de calidad. Prosiguen los algoritmos DE/current-to-best/1 y AGG-BLX que no se han adaptado bien a la nueva función objetivo. Ya nos acercamos a los algoritmos que mejores soluciones han sacado, empezando por BL y AM(1,0.1mej), que han sacado resultados muy similares debido a la BL que proporciona exploraciones similares en ambos algoritmos. Dejando a los algoritmo ILS y DE/Rand/1 ganadores en cuanto a valoración de la función objetivo, esto es debido a que la exploración en estos algoritmos es mayor, mutaciones más bruscas, pudiendo explorar más soluciones de simplicidad. Por lo tanto a la vista de los resultados, si buscamos ejecuciones rápidas la BL local es el algoritmo, a la que aplicando una mutación más agresiva, mejoraría. Si nos centramos en soluciones de calidad ILS y DE/Rand/1 están a la par, con bajas desviaciones, buena clasificación y soluciones muy simples.

Algoritmos que en la anterior práctica sacaron excelentes resultados, como son los genéticos y meméticos no se han adaptado bien a la nueva función objetivo, debido a que ésta requiere de mutaciones más bruscas para explorar las soluciones con pesos igualados a 0. La búsqueda local sufre de éste mísmo problema, y observamos como en ILS al aumentar la agresividad de las mutaciones obtiene mejores resultados. La generación de soluciones aleatorias tampoco está enfocada a explorar las soluciones de simplicidad, ya que se generan muy pocos pesos igualados a 0, afectando a los algoritmos de rápida convergencia, genéticos y ES sobre todo, ya observamos en la figura11.9 el impacto de añadir soluciones iniciales con todos los pesos inicializados a 0.

Referencias

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

[2] https://en.wikipedia.org/wiki/Differential_evolution.