PRÁCTICA3.B.: Algoritmos Genéticos para el Problema de la Selección de Característica

4º CURSO DEL GRADO EN INGENIERÍA INFORMÁTICA Y MATEMÁTICAS

MARINA ESTÉVEZ ALMENZAR
75571215 – T
marinaestal@correo.ugr.es

GRUPO DE PRÁCTICAS: JUEVES
PROFESOR: ÓSCAR CORDÓN

Índice

BREVE DESCRIPCIÓN DEL PROBLEMA

Se desarrollará el problema de **selección de características**, en el que se proporciona un conjunto inicial de características relativas a un conjunto de elementos, y cuyo objetivo es extraer un subconjunto de las mismas que maximice el acierto de clasificación de dichos elementos. La clasificación hace uso del clasificador considerado para esta práctica, el **3-NN**.
El clasificador 3-NN recibirá un conjunto de características seleccionadas de un elemento en concreto, y determinará, según dicha selección, la clase del elemento (que posteriormente será comparada con la clase que corresponde y se contará o no como un acierto).
Así, para combinar el trabajo de la selección de características y el clasificador, se han usado los 3 conjuntos de datos especificados en el guión de prácticas (**WDBC**, **Movement_Libras** y **Arrhythmia**), y el método de validación **5x2-cross**.
El algoritmo de selección de características implementado es **AGG** (a falta de tiempo, queda pendiente la implementación de **AGE**)**.** También se ha implementado, con el fin de comparar los algoritmos nombrados, el greedy *Sequenctial Forward Selection* (**SFS**).

BREVE DESCRIPCIÓN DE LA APLICACIÓN DE LOS ALGORITMOS EMPLEADOS AL PROBLEMA

El algoritmo **AGG** hace uso de los siguientes recursos, ya explicados y utilizados en prácticas anteriores:

Lectura de archivos .arff
Puesto que los datos nos vienen dados en ficheros con formato .arff, se ha implementado un método específico para leer este tipo de archivos. Dicho método recibe como argumento el nombre del archivo y devuelve una matriz NxN en la que cada fila corresponde a un elemento, y contiene N-1 características del mismo. En la posición N se encuentra la clase a la que pertenece dicho elemento. Este método de lectura de archivos .arff será común en la ejecución de cada algoritmo. Sin embargo, para el caso del archivo **WDBC** ha sido necesario implementar **un método de lectura distinto** que el de los archivos de Movement_Libras y Arrhythmia, ya que la disposición de los datos de wdbc.arff es distinta a las otras dos (la clase de cada elemento, en este caso B o M, aparece al principio de cada línea, y no al final).

5x2 Cross – Validation
Para las 10 ejecuciones de cada algoritmo, los conjuntos considerados (conjunto de prueba y conjunto de entrenamiento) son los obtenidos en el método de validación 5x2. Aunque dicho método es el mismo para todos los algoritmos, su implementación varía dependiendo de la estructura de nuestros datos; se diferencia principalmente en el número de clases que se estén considerando. A continuación se muestra el pseudocódigo de este método aplicado al WDBC, que contiene únicamente dos tipos de clases:

```
// 5 x 2 Cross - Validation para el problema WDBC

    datos // es nuestra matriz con los datos leídos de wdbc.arff
    vector claseM; // contendrá los índices de las filas cuya clase es M
    vector claseB; // contendrá los índices de las filas cuya clase es B
    vector posiciones // contendrá los índices de las filas que ya han sido asignadas
                                // al conjunto de prueba o al conjunto de entrenamiento
    int posicion;

    for (i = 0; i < datos.size(); i++) {
```

```
        if (datos[i][posicion_clase] == 'M') claseM < --i;
        else if (datos[i][posicion_clase] == 'B') claseB < --i;
}
// Se escogen aleatoriamente los datos que contendrá el conjunto de prueba
for (i = 0; i < claseM.size() / 2; i++) {
        while (posicion ya esté en posiciones) {
                posicion = Rand(0, claseM.size() - 1);
        }
        prueba <-- datos[ claseM[posicion] ];
        posiciones <-- posicion;
}

// Se cogen los datos restantes para el conjunto de entrenamiento
for (i = 0; i < claseM.size(); i++) {
        if (i no está en posiciones) entrenamiento <-- datos[ claseM[i] ];
}

// Se repite este proceso para los datos contenidos en claseB
```

Esquema de representación de soluciones y generación de solución inicial

Se considera como posible solución un vector binario de tamaño N (número de características) que representa qué características son tomadas para la clasificación.

Las soluciones iniciales, en este caso, son aleatorias: Se considera una población de 30 cromosomas (30 soluciones iniciales aleatoias).

Pseudocódigo de la función objetivo y de la generación de vecino flip(s, i)

```
FuncionObjetivo {
        for (i = 0; i < N; i++) {
                clase = Clas3NN(prueba[i],matriz);
                if (clase == prueba[i][posicion_clase]) aciertos++;
        }
        tasa = 100 * (aciertos / prueba.size());
        return tasa;
}


Clas3NN(matriz conjunto, matriz conjunto_comparacion, vector s) {
        vector distancias; // quedarán almacenadas las 3 distancias más cortas
        distancias[0] = 9999.99;
        distancias[1] = 9999.99;
        distancias[2] = 9999.99;
        vector posiciones;

        if(conjunto == conjunto_comparacion){
                cs = Multiplica(conjunto, s); // se multiplica cada fila de la
    matriz conjunto por el vector s para obtener el conjunto con las características
    seleccionadas
                for(i=0; i<cs.size(); i++){
                        for(k=0; k<cs.size(); k++){
                                if(i != k){
                                        distancia = CalculaDistancias(cs[i], cs[k]);
                                        if(distancia < distancias[2]) {
                                                if(distancia < distancias[1]){
                                                        if(distancia < distancias[0]){
                                                                distancias[2] =
    distancias[1];
                                                                posiciones[2] =
    posiciones[1];
                                                                distancias[1] =
    distancias[0];
                                                                posiciones[1] =
    posiciones[0];
```

```
                                                distancias[0] = distancia;
                                                posiciones[0] = k;
                                          }
                                          else{
                                                distancias[2] =
distancias[1];
                                                posiciones[2] =
posiciones[1];
                                                distancias[1] = distancia;
                                                posiciones[1] = k;
                                          }
                                    else{
                                          distancias[2] = distancia;
                                          posiciones[2] = k;
                                    }
                              }
                        }
                  } // fin del segundo bucle for

                  // en distancias quedan almacenadas las 3 distancias más
cortas
                  // en posiciones quedan almacenadas las posiciones a esas 3
distancias
                  POS_CLASE = cs[posiciones[0]].size()-1;
                  if(cs[posiciones[0]][POS_CLASE] == cs[posiciones[1]]
[POS_CLASE] || cs[posiciones[0][POS_CLASE] == cs[posiciones[2]][POS_CLASE])
                        clase = cs[posiciones[0]][POS_CLASE];

                  else if(cs[posiciones[1]][POS_CLASE] == cs[posiciones[2]]
[POS_CLASE])
                        clase = cs[posiciones[1]][POS_CLASE];
                  else
                        clase = cs[posiciones[0]][POS_CLASE];


                  if(clase == cs[i][POS_CLASE]) aciertos++;

      } // fin del primer bucle for

      else{ // conjunto != conjunto_comparacion

            cs = Multiplica(conjunto, s); // se multiplica cada fila de la
matriz conjunto por el vector s para obtener el conjunto con las características
seleccionadas
            csc = Multiplica(conjunto_comparacion, s); // se multiplica cada
fila de la matriz conjunto_comparacion por el vector s para obtener el conjunto
con las características seleccionadas
            for(i=0; i<cs.size(); i++){
                  for(k=0; k<csc.size(); k++){
                              distancia = CalculaDistancias(cs[i], csc[k]);
                              if(distancia < distancias[2]) {
                                    if(distancia < distancias[1]){
                                          if(distancia < distancias[0]){
                                                distancias[2] =
distancias[1];
                                                posiciones[2] =
posiciones[1];
                                                distancias[1] =
distancias[0];
                                                posiciones[1] =
posiciones[0];
                                                distancias[0] = distancia;
                                                posiciones[0] = k;
                                          }
                                          else{
```

```
                                                distancias[2] =
        distancias[1];
                                                posiciones[2] =
        posiciones[1];
                                                distancias[1] = distancia;
                                                posiciones[1] = k;
                                        }
                                    else{
                                        distancias[2] = distancia;
                                        posiciones[2] = k;
                                    }
                                }
                            } // fin del segundo bucle for

                            // en distancias quedan almacenadas las 3 distancias más
        cortas
                            // en posiciones quedan almacenadas las posiciones a esas 3
        distancias
                            POS_CLASE = csc[posiciones[0]].size()-1;
                            if(csc[posiciones[0]][POS_CLASE] == csc[posiciones[1]]
        [POS_CLASE] || csc[posiciones[0][POS_CLASE] == csc[posiciones[2]][POS_CLASE])
                                    clase = csc[posiciones[0]][POS_CLASE];

                            else if(csc[posiciones[1]][POS_CLASE] == csc[posiciones[2]]
        [POS_CLASE])
                                    clase = csc[posiciones[1]][POS_CLASE];
                            else
                                    clase = csc[posiciones[0]][POS_CLASE];


                            if(clase == cs[i][POS_CLASE]) aciertos++;

                } // fin del primer bucle for

                 return aciertos;
        }
```

// Generación de vecino

```
    N = s.size() - 1;
    i → {0,1,2,...,N}
    matriz prueba; // matriz de datos
    vecino = flip(s, i);
    for ( j = 0; j < prueba.size(); j++) {
           for (int k = 0; k < vecino.size(); k++) {
                  v.push_back(prueba[j][k] * vecino[k]);
           }
               v.push_back(prueba[j][vecino.size()]); // donde se encuentra la clase a la
        que
                                                pertenece
           prueba_vecino.push_back(v);// prueba_vecino es la prueba con las
                                        caracteristicas elegidas en vecino = flip(s,i)
           v.clear();
    }
```

Pseudocódigo del mecanismo de selección considerado en AGG

```
/* Selección
 * Se escogen dos individuos no repetidos aletoriamente y nos quedamos con el
mejor.
 * Repetimos ésto tantas veces como individuos haya en la población */

// Sea C nuestra población de cromosomas
// Almacenamos en C1 la nueva población seleccionada

for (int i = 0; i < C.size(); i++) {
    i1 = Randint(0, C.size() - 1);
    i2 = Randint(0, C.size() - 1);
    while (i1 == i2) {
        i2 = Randint(0, C.size() - 1);
    }
    if (aciertos[i1] > aciertos[i2]) {
        C1 ← C[i1]);
    }
    else {
        C1 ← C[i2];
    }
}
```

Pseudocódigo del operador de cruce en AGG

/* Cruce

```
 * Como el mecanismo de selección ya tiene una componente aleatoria, se realiza
un emparejamiento fijo:
 * el primero con el segundo, el tercero con el cuarto, etc.
 * Calculamos N = número esperado de cruces, y cruzamos las N primeras parejas
 * Realizamos el cruce clásico en dos puntos */

for (madre = 0; madre < num_cruces*2; madre+=2){
    padre = madre + 1;

    // Generamos los puntos de cruce

    i1 = Randint(0, C1[madre].size()-1);
    if (i1 != C1[madre].size()-1) i2 = Randint(i1+1, C1[madre].size()-1);
    else i2 = i1;

    // Cruzamos las partes contenidas entre ambos puntos

    for (int i = i1; i < i2; i++){
        aux = C1[madre][i1];
        C1[madre][i1] = C1[padre][i1];
        C1[padre][i1] = aux;
    }
    // En por_evaluar[] guardamos las posiciones de aquellos cromosomas
      que necesitarán ser reevaluados, en este caso por haber sido cruzados
    por_evaluar ← madre;
    por_evaluar ← padre;
}
```

Pseudocódigo del operador de mutación en AGG

/* Mutación

```
 * Fijamos un número fijo de mutaciones
 * Escogemos aleatoriamente qué cromosoma y qué gen de ese cromosoma mutar */

for(int i = 0; i < num_mut; i++) {
    CROM = Randint(0, C1.size() - 1);
    GEN = Randint(0, C1[CROM].size() - 1);
    C1[CROM] = flip(C1[CROM],GEN);

      // Comprobamos si este cromosoma ya está considerado en por_evaluar[]
        En caso contrario, lo metemos
    contenido = false;
    for (int j = 0; j < por_evaluar.size(); j++){
        if (por_evaluar[j] == CROM) contenido = true;
    }
    if (!contenido) por_evaluar ← CROM;
}
```

PSEUDOCÓDIGOS Y DESCRIPCIONES

Pseudocódigo del esquema de evolución y de reemplazamiento

/* Selección elitista y evaluación

```
  * Comprobamos si se ha mantenido mejorS (mejor cromosoma de la población
inicial)
  *   - Si no se ha mantenido: se sustituye el peor cromosoma de la población
      actual por mejorS
  *   - Si se ha mantenido no se hace nada
  * ESTA COMPROBACIÓN, CON EL FIN DE AHORRAR TIEMPO DE CÁLCULO, SE EVITA Y
      DIRECTAMENTE SE SUSTITUYE EL PEOR CROMOSOMA DE LA NUEVA POBLACIÓN POR EL
      MEJOR DE LA POBLACIÓN ANTERIOR (MAX)

  * Se evalúan solo aquéllos cromosomas que han sido alterados (aquéllos cuyas
      posiciones están en por_evaluar[] */

 MIN = 999.99;
 for( int i = 0; i < por_evaluar.size(); i++){
     aciertos1[por_evaluar[i]] = clas3nn(prueba, prueba, C1[por_evaluar[i]]);
     eval++;
     if (aciertos1[por_evaluar[i]] >= MAX){
         MAX = aciertos1[por_evaluar[i]];
         MEJOR = C1[por_evaluar[i]];
     }
     //if(!mantenido) {
```

```
            if (aciertos1[por_evaluar[i]] < MIN) {
                MIN = aciertos1[por_evaluar[i]];
                pos_min = por_evaluar[i];
            }
        //}
 }
 /*if(!mantenido)*/ C1[pos_min] = mejorS;
```

El esquema general, por tanto, es:

```
/* Generación de la población inicial

 30 cromosomas generados aleatoriamente */

      matriz C;       // C = Población, cada fila es un cromosoma
      int evaluaciones = 0;

/* Evaluacion de los individuos de la poblacion inicial
 * 30 evaluaciones */

      int MAX =0;        // almacena los aciertos del mejor s de la población
      aciertos;        // almacena la evaluación de los cromosomas
      mejorS;          // almacena el mejor s de la población

      for( int i = 0; i < C.size(); i++){
            aciertos[i] ← clas3nn(prueba, prueba, C[i]));
            evaluaciones++;
            if (aciertos[i] > MAX){
                  MAX = aciertos[i];
                  mejorS = C[i];
            }
      }

/* AGG */

      while(eval < 15000){

            * SELECCIÓN
            * CRUCE
            * MUTACIÓN
            * SELECCIÓN ELITISTA Y EVALUACIÓN (eval++ por cada evaluación)
            * Actualización de estructuras para el siguiente bucle del  while

      }
```

DESCRIPCIÓN DEL ALGORITMO DE COMPARACIÓN (SFS)

A continuación se muestra el pseudocódigo del algoritmo de comparación:

```
// SFS
    hay_mejora = true;
    tasa_anterior = 0;
    mejor_tasa = 0;

    while (hay_mejora) {
          for (i = 0; i < s.size(); i++) {
                if (s[i] != 1) {
                      s[i] = 1;
```

```
                        tasa = CalculaCoste(s);

                        if (tasa > mejor_tasa) {
                                mejor_tasa = tasa;
                                mejor_s = s;
                        }
                        s[i] = 0;
                }
        }
        s = mejor_s;
        if (tasa_anterior >= mejor_tasa) hay_mejora = false;
        else tasa_anterior = mejor_tasa;
}
```

PROCEDIMIENTO CONSIDERADO PARA DESARROLLAR LA PRÁCTICA

La implementación del código ha sido iniciada desde cero (no se ha usado el código proporcionado en la plataforma de la asignatura ni ningún otro). Tampoco se ha hecho uso de ningún *framework*. Tan solo se ha hecho uso del generador aleatorio proporcionado en la web de la asignatura.

EXPERIMENTOS Y ANÁLISIS DE RESULTADOS

CASOS DEL PROBLEMA Y VALORES DE PARÁMETROS CONSIDERADOS

- **Semillas consideradas:**

Para la primera partición del conjunto de datos se considera una semilla con valor **47**.
Para la segunda partición del conjunto de datos se considera una semilla con valor **18**.
Para la tercera partición del conjunto de datos se considera una semilla con valor **95**.
Para la cuarta partición del conjunto de datos se considera una semilla con valor **6**.
Para la quinta partición del conjunto de datos se considera una semilla con valor **33**.

Estas semillas son las mismas para las distintas ejecuciones de todos los algoritmos.

RESULTADOS OBTENIDOS: TABLAS

Tabla de resultados obtenidos por el algoritmo SFS para el problema de SC

	WDBC			*MOVEMENT_LIBRAS*			*ARRHYTHMIA*		
	%CLAS	**%RED**	**T (seg)**	**%CLAS**	**%RED**	**T(seg)**	**%CLAS**	**%RED**	**T(seg)**
1 – 1	95'7895	83'3333	2'849	71'6667	91'1111	6'832	75'7732	95'6835	19'604
1 – 2	94'3662	93'3333	1'248	72'2222	90'0000	7'251	68'2292	97'8417	12'501
2 – 1	94'386	86'6667	1'89	72'7778	92'2222	6'137	77'3169	97'1223	18'965
2 – 2	93'4958	90'6667	2'075	66'6667	92'2222	4'903	69'7917	97'482	14'747
3 – 1	94'3667	83'3333	2'901	65'6667	93'3334	5'0871	75'8993	95'6667	20'102
3 – 2	92'7778	94'4443	1'907	71'6667	90'0000	7'005	69'7726	98'1223	15'2245
4 – 1	95'0333	83'3333	2'778	74'2222	87'7778	4'9728	77'2849	97'201	16'0961
4 – 2	93'7778	86'6667	1'08	63'3333	91'1111	10'221	66'998	98'8411	10'334
5 – 1	92'9825	93'3333	1'065	74'4444	87'7778	10'767	69'0722	98'2014	10'674
5 – 2	94'7183	83'3333	2'248	63'3333	87'7778	9'339	73'9583	97'482	15'452
MEDIA	**94'5218**	**87'3333**	**1'701**	**69'6667**	**90'3334**	**7'1289**	**72'3726**	**97'4667**	**15'5927**

Tabla de resultados obtenidos por el algoritmo del Clasificador 3-NN para el problema de SC

	WDBC			*MOVEMENT_LIBRAS*			*ARRHYTHMIA*		
	%CLAS	**%RED**	**T (seg)**	**%CLAS**	**%RED**	**T(seg)**	**%CLAS**	**%RED**	**T(seg)**
1 – 1	95'7895	X	0'078	76'1111	X	0'071	67'5258	X	0'268
1 – 2	96'1268	X	0'041	73'8889	X	0'085	61'9792	X	0'255
2 – 1	95'0877	X	0'128	76'1111	X	0'076	65'4639	X	0'337
2 – 2	96'831	X	0'181	67'7778	X	0'079	65'625	X	0'235
3 – 1	97'386	X	0'102	77'1111	X	0'091	61'9792	X	0'301
3 – 2	96'1267	X	0'069	73'7778	X	0'082	65'625	X	0'276
4 – 1	94'0877	X	0'061	67'8889	X	0'07	64'3882	X	0'227
4 – 2	96'7895	X	0'083	76'1111	X	0'071	66'2441	X	0'316
5 – 1	94'386	X	0'078	73'8889	X	0'083	64'433	X	0'321
5 – 2	97'8873	X	0'093	73'8889	X	0'071	65'625	X	0'237
MEDIA	**96'3887**	**X**	**0'091**	**73'7778**	**X**	**0'0779**	**66'2258**	**X**	**0'2773**

Tabla de resultados obtenidos en AGG para el problema de SC

	WDBC			*MOVEMENT_LIBRAS*			*ARRHYTHMIA*		
	%CLAS	**%RED**	**T (seg)**	**%CLAS**	**%RED**	**T(seg)**	**%CLAS**	**%RED**	**T(seg)**
1 – 1	97'1831	33'3333	607'993	72'7778	56'6667	454'526			
1 – 2	94'7368	40	541'53	71'6667	62'2222	385'826			
2 – 1	98'3455	50'6667	539'087	74'7778	50	503'48			
2 – 2	97'7039	33'3333	482'933	73'1111	48'8889	483'213			
3 – 1	96'6778	43'4444	398'19	72'2222	66'6667	490'228			
3 – 2	95'1833	40	602'201	74'2222	47'2222	511'492			
4 – 1	98'2394	40	484'077	74'4444	47'7778	487'715			
4 – 2	95'4386	56'6667	357'714	73'6667	61'7778	477'362			
5 – 1	95'0704	40	566'119	72'7778	47'7778	516'782			
5 – 2	96'4912	50	514'229	72'2222	51'1111	501'375	67'5258	53'2374	1655'3
MEDIA	**96'5849**	**42'7533**	**509'493**	**73'4444**	**54'7778**	**467'378**			

ANÁLISIS DE LOS RESULTADOS

Puesto que no me ha dado lugar a desarrollar el algoritmo **AGE** (Algoritmo Genético Estacionario) no podemos realizar una comparación entre éste y el **AGG**. Sin embargo, sí podemos comparar los resultados obtenidos con **AGG y SFS**.

A la vista de las **tasas de clasificación**, es notorio que las soluciones proporcionadas por AGG son significativamente mejores que las que nos proporciona SFS. Esto se debe a las múltiples ventajas que aportan las características de los Algoritmos Genéticos frente a los Greedy: **paralelismo intrínseco, diversidad en el desarrollo**, etc. Sin embargo, si nos fijamos en las **tasas de reducción** y en los **tiempos**, tenemos que en este caso los proporcionados por AGG son peores; en concreto, este incremento del tiempo se debe a que, por lo general, los Algoritmos Genéticos son **computacionalmente muy costosos**.