PRÁCTICA 5.b: Búsquedas Híbridas para el Problema de la Selección de Características

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

1 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**.

Los algoritmos 3 **algoritmos meméticos** de selección de características implementados son:

- Algoritmo memético AM-(10,1.0): Cada 10 generaciones, se aplica la BL sobre todos los cromosomas de la población.
- Algoritmo memético 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.
- Algoritmo memético AM-(10,0.1mej): Cada 10 generaciones, aplicar la BL sobre los 0.1·N mejores cromosomas de la población actual (N es el tamaño de ésta).

También se ha implementado, con el fin de comparar los algoritmos nombrados, el greedy *Sequenctial Forward Selection* (**SFS**).

2 BREVE DESCRIPCIÓN DE LA APLICACIÓN DE LOS ALGORITMOS EMPLEADOS AL PROBLEMA

Los tres algoritmos implementados mantienen algunas similitudes en cuanto a código que cabe destacar.

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

En todos los algoritmos 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 son generadas aleatoriamente en los 3 algoritmos: se genera una población de **10 cromosomas aleatorios**.

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();
    }
```

3 PSEUDOCÓDIGOS Y DESCRIPCIONES

3.1 PSEUDOCÓDIGO DEL MÉTODO DE BÚSQUEDA LOCAL

```
// BÚSQUEDA LOCAL
    datos = readfile();
    5x2cross-validation --> prueba y conjunto;

    s = s_inicial // solucion inicial generada de forma aleatoria
    mejor_tasa = CalculaCoste(s); // clas3nn(prueba, prueba, s)

    while (i < s.size()) {
        vecino = CalculaVecino(s);
        tasa_vecino = CalculaCoste(vecino); // clas3nn(prueba, prueba, vecino)

        if (tasa_vecino > mejor_tasa) {
            mejor_tasa = tasa_vecino;
            s = vecino;
            if (mejor_tasa > MAX) { // para seguir conservando el ELITISMO:
                                    // MAX = mejor resultado de la población anterior
                                    // mejorS = mejor cromosoma de la pob. anterior
                mejorS = vecino;
                MAX = mejor_tasa;
            }
            i = 0;      // se reinicializa el bucle
        }
        else i++;
    }
```

3.2 PSEUDOCÓDIGO DE AM-(10,1.0)

```
// AM-(10,1.0)
    datos = readfile();
    5x2cross-validation(datos) --> matrices prueba y conjunto;
    generaciones = 0;
    matriz C;       // C = Población, cada fila es un cromosoma

    while(eval < 15000){

    /* AGG */

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

        generaciones++;

        // CADA 10 GENERACIONES SE APLICA LA BL SOBRE TODOS LOS CROMOSOMAS DE
LA POBLACIÓN

        if (generaciones == 10){
```

```
            generaciones = 0;
           // Aplico Búsqueda Local sobre todos los cromosomas de C
           for(i=0; i<C.size(); i++){
              contador = 0;
              mejor_solucion = CalculaCoste(C[i]); // clas3nn(prueba, prueba, C[i])

              // BL
              while (contador < C[i].size()){
                 vecino = CalculaVecino(C[i]);  // flip(C[i],contador)
                 solucion = CalculaCoste(vecino); // clas3nn(prueba, prueba,
vecino)
                 if (solucion > mejor_solucion){
                     mejor_solucion ← solucion;
                     C[i] ← vecino;
                     if (mejor_solucion > MAX){
                         mejorS ← vecino;
                         MAX ← mejor_solucion;
                     }
                     contador = 0;
                 }
                 else contador++;
              }
           }
        }
    }
```

3.3 Pseudocódigo de AM-(10,0.1)

```
// AM-(10,0.1)
    datos = readfile();
    5x2cross-validation(datos) --> matrices prueba conjunto;
    matriz C;       // C = Población, cada fila es un cromosoma
    generaciones = 0;

    while(eval < 15000){
    /* AGG */

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

        generaciones++;


    // CADA 10 GENERACIONES SE APLICA LA BL SOBRE UN SUBCONJUNTO DE
    // CROMOSOMAS DE LA POBLACIÓN SELECCIONADO ALEATORIAMENTE CON
    // PROBABILIDAD 0.1 POR CADA CROMOSOMA
```

```
   // Puesto que la probabilidad de que se aplique BL es de 0.1 para cada
cromosoma y nuestra población tiene 10 cromosomas, se aplicará BL sobre un
cromosoma que escogemos aleatoriamente

     if (generaciones == 10){
       generaciones = 0;
       CROM = Randint(0, C.size() - 1);
       mejor_solucion = aciertos[CROM];

       //BL

       while (contador < C[CROM].size()){

           vecino = CalculaVecino(C[CROM]);  // flip(C[CROM],contador)

           solucion = CalculaCoste(vecino); // clas3nn(prueba, prueba, vecino)

           if (solucion > mejor_solucion){
               mejor_solucion ← solucion;
               C[CROM] ← vecino;
               if (mejor_solucion > MAX){
                   mejorS ← vecino;
                   MAX ← mejor_solucion;
               }
               contador = 0;
           }
           else contador++;
       }

     }
   }
```

3.4 Pseudocódigo de AM-(10,0.1mej)

```
// AM-(10,0.1mej)
   datos = readfile();
   5x2cross-validation(datos) --> matrices prueba conjunto;
   generaciones = 0;
   matriz C;       // C = Población, cada fila es un cromosoma

   while(eval < 15000){
   /* AGG */

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

      generaciones++;

   // CADA 10 GENERACIONES SE APLICA LA BL SOBRE LOS 0.1*N MEJORES
```

```
    // CROMOSOMAS DE LA POBLACIÓN ACTUAL
    // Puesto que nuestra población tiene 10 cromosomas, se aplicará BL sobre
el mejor cromosoma de la población

        if (generaciones == 10){
            generaciones = 0;
            CROM = posicionMAX;
            mejor_solucion = MAX;

        // BL
        while (contador < C[CROM].size()){

            vecino = CalculaVecino(C[CROM]);  // flip(C[CROM],contador)

            solucion = CalculaCoste(vecino); // clas3nn(prueba, prueba, vecino)

           if (solucion > mejor_solucion){
               mejor_solucion ← solucion;
               C[CROM] ← vecino;
               mejorS ← vecino;
               MAX ← mejor_solucion;
               contador = 0;
            }
            else contador++;
         }

      }
    }
```

4 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;
    }
```

5 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.

6 EXPERIMENTOS Y ANÁLISIS DE RESULTADOS

6.1 CASOS DEL PROBLEMA Y VALORES DE PARÁMETROS CONSIDERADOS

- **Semillas consideradas:**

Para la primera partición del conjunto de datos se considera una semilla con valor **46**.
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.

6.2 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 AM-(10,1.0) 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	46'6667	568'532	71'7778	53'1111	540'394	77'0655	51'0791	1619'9
1 – 2	95'7895	46'6667	494'646	70	47'5949	464'324	71'9794	55'7554	1393'31
2 – 1	95'1268	53'6667	543'393	69'9043	47'7778	583'395	77'1875	50'6667	1857'34
2 – 2	95'4386	46'6667	594'343	73'8889	45'2222	483'765	69'7917	46'1443	1750'38
3 – 1	97'8821	47'7778	422'949	76'1111	51'1111	593'319	76'6667	44'9664	1664'39
3 – 2	96'4895	52'2222	447'713	70	44'6667	533'437	75'385	53'4338	1635'38
4 – 1	97'0771	49'5556	530'837	71'6667	47'7778	565'166	77'1875	53'9568	1704'928
4 – 2	95'3958	50	529'38	67'7778	44'4444	560'985	69'2784	46'0432	1859'324
5 – 1	96'1268	46'6667	405'595	73'8889	52'2222	498'006	67'7083	47'1223	1711'292
5 – 2	96'8421	26'6667	509'633	73'8889	45'5556	566'863	74'433	44'964	1496'8
MEDIA	**96'3351**	**46'6556**	**504'702**	**71'8904**	**47'9483**	**538'965**	**73'6683**	**49'4132**	**1669'30**

(*) Debido al gran coste computacional que supone la ejecución del Arrhythmia, para éste se han ejecutado 5000 evaluaciones en lugar de 15000

Tabla de resultados obtenidos en AM-(10,0.1) 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	50	476'34	75'7778	48'1111	569'385	76'1111	49'3334	1398'539
1 – 2	95'7895	60	392'191	76'3333	54'8889	473'154	69'7778	54'4938	1290'385
2 – 1	93'3333	53'3333	514'905	69'2222	55'5556	549'557	78'8889	46'6667	1355'201
2 – 2	95'0225	60	416'805	75'1111	52'5556	488'163	69'7917	43'3457	1294'394
3 – 1	97'1831	33'3333	623'97	76'1111	51'1111	563'399	74'4059	45'6533	1328'593
3 – 2	96'4899	43'3333	578'698	73'7778	41'6667	452'813	70'2222	55'3859	1311'429
4 – 1	96'4789	60	493'056	70'4444	42'2222	574'029	79'1875	53'5971	1334'67
4 – 2	95'386	50	644'198	67'7778	51'1111	496'056	65'7938	47'8417	1264'324
5 – 1	95'5747	70	427'068	73'5556	54'4444	536'733	69'1458	46'7626	1353'465
5 – 2	96'4912	40	659'62	74'4444	52'2222	525'707	75'4021	45'6835	1260'282
MEDIA	**96'2532**	**51'9999**	**522'685**	**71'3777**	**50'389**	**557'899**	**72'8727**	**48'8764**	**1319'12**

(*) Debido al gran coste computacional que supone la ejecución del Arrhythmia, para éste se han ejecutado 5000 evaluaciones en lugar de 15000

Tabla de resultados obtenidos en AM-(10,0.1mej) para el problema de SC

	WDBC			*MOVEMENT_LIBRAS*			*ARRHYTHMIA (*)*		
	%CLAS	**%RED**	**T (seg)**	**%CLAS**	**%RED**	**T(seg)**	**%CLAS**	**%RED**	**T(seg)**
1 – 1	98'2394	56'6667	533'805	67'2222	45'5556	641'783	76'6667	50'3958	1288'284
1 – 2	95'4386	46'6667	510'315	74'4444	48'8889	545'722	70'2222	53'3348	1358'49
2 – 1	96'4789	50	446'712	66'1111	47'7778	537'321	78'2199	46'6687	1299'485
2 – 2	95'7895	53'3333	519'970	66'6667	45'5556	641'164	69'7971	50'2222	1340'394
3 – 1	96'831	43'3333	609'44	73'1111	50	546'667	76'0932	45'8689	1319'037
3 – 2	94'186	33'3333	616'710	74'4444	45'3333	623'709	67'8849	46'0494	1260'383
4 – 1	96'831	50	655'155	71'6667	44'4444	614'991	77'2292	53'9568	1330'012
4 – 2	97'6368	46'6667	512'01	70'5556	47'7778	611'277	76'732	47'8417	1282'992
5 – 1	94'0141	50	454'785	73'3333	58'8889	442'987	69'1796	48'5611	1330'558
5 – 2	96'7421	40	511'88	72'7778	51'1111	501'268	73'9794	46'2014	1271'581
MEDIA	**96'1587**	**47**	**517'078**	**71'0333**	**48'5333**	**570'689**	**73'0004**	**48'9101**	**1308'12**

(*) Debido al gran coste computacional que supone la ejecución del Arrhythmia, para éste se han ejecutado 5000 evaluaciones en lugar de 15000

6.3 ANÁLISIS DE RESULTADOS

Empezamos comparando los resultados obtenidos en los tres algoritmos implementados para esta práctica:

- Podemos observar que los mejores resultados en cuanto a tasas de clasificación se obtienen el primer algoritmo **AM-(10-,1.0)**. Ésto puede deberse principalmente a que éste algoritmo, cada 10 generaciones construídas por el algoritmos AGG, aplica Búsqueda Local a los 10 cromosomas que componen la población, sin hacer distinción.
- Ésto podría hacernos pensar que, por contra, el AM-(10,1.0) supondrá una mayor carga computacional en cuanto a tiempo de ejecución se refiere, pero no es así: observamos que también obtenemos los mejores tiempos en este primer algoritmo (sin tener en cuenta el caso de *Arrhythmia*, ya que este se ha ejecutado un número inferior de evaluaciones): puede deberse a que el coste computacional es mayor en el algoritmo AGG, el cual se debe ejecutar más veces en los algoritmos **AM-(10,0.1)** y **AM-(10,0.1mej)** para llegar a las 15000 evaluaciones. En el caso de AM-(10,1.0) se cuenta con el número de evaluaciones procedentes de la Búsqueda Local, que son más que en los otros dos algoritmos.

En cualquier caso, si comparamos cualquiera de estos tres algoritmos con el **SFS** podemos observar como la mejora es notable en el caso de los Algoritmos Meméticos, como cabía esperar. Sin embargo es bastante más notable la diferencia en cuanto a tiempos de ejecución, siendo los AM los más pesados con diferencia, ya que fusionan un AGG, que ya de por sí es un algoritmo pesado, con una BL, que suma pesadez. Es por ello que los Algoritmos Meméticos, a pesar de sus buenos resultados, deben de ser usados en problemas específicos donde conozcamos que funcionan bien y que merece la pena el tiempo empleado en sus ejecuciones.