PRÁCTICA 1.B: BÚSQUEDAS POR TRAYECTORIAS 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

ALGORITMOS CONSIDERADOS:

BÚSQUEDA LOCAL
ENFRIAMIENTO SIMULADO
BÚSQUEDA TABÚ

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 de selección de características implementados son **Búsqueda Local**, **Enfriamiento Simulado** y **Búsqueda Tabú**. 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 (**Búsqueda Local**, **Enfriamiento Simulado** y **Búsqueda Tabú**) 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. Asimismo, la solución inicial, con la que da comienzo cualquiera de los algoritmos, se genera de forma aleatoria.

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

```
FuncionObjetivo {
    for (i = 0; i < N; i++) {
        clase = Clasificador3NN(prueba[i],matriz);
        if (clase == prueba[i][posicion_clase]) aciertos++;
    }
    tasa = 100 * (aciertos / prueba.size());
    return tasa;
}


Clasificador3NN(vector prueba, matriz conjunto) {
    vector distancias;
    vector clases;
    distancias = CalculaDistancias(prueba, conjunto);
    while (i<3) {
        for (j = 0; j < distancias.size(); j++) {
            if (distancias[j] < MIN && distancias[j] != 0) {
                MIN = distancias[j];
                pos_MIN = j;
            }
        }
        clases.push_back(conjunto[pos_MIN][prueba.size() - 1]); // prueba.size()-1
    corresponde a la posición de la clase
        distancias[pos_MIN] = 0;
        i++;
    }
}
```

```
// Generación de vecino

    i = Randint(0, s.size() - 1); // (*)
    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();
    }
```

```
// (*) En la Búsqueda Tabú se ha de comprobar que el índice i no esté repetido entre los índices que ya hayan generado los vecinos que hay hasta ese momento. En la Búsqueda Local, la generación del índice i no es aleatoria, sino que se recorre en orden
```

Criterios de aceptación y de parada

En Enfriamiento Simulado se considera:

Condición de enfriamiento:

- Número máximo de vecinos generados (aceptados y no aceptados)
- Número máximo de vecinos aceptados (éxitos)

Condición de parada:

- Número máximo de evaluaciones alcanzadas (en nuestro caso, 15000 evaluaciones)
- Número máximo de enfriamientos alcanzados (en nuestro caso, M = 15000 / max_vecinos)
- Número de éxitos en una iteración del bucle interno igual a cero
- **Adicionalmente**, se estableces también como condición de parada que la mejor tasa calculado hasta el momento alcance el 100%

En Búsqueda Tabú se considera:

Criterio de aceptación:

- Cuando uno de los 30 vecinos (sin repetición) que han de generarse en el bucle interno resulta estar en la Lista Tabú, se comprueba este criterio: Si la tasa generada con dicho vecino supera la mejor tasa (global) encontrada hasta el momento, entonces se acepta el vecino entre esos 30, de los que se escogerá al mejor.
  Como **comprobación adicional** en este criterio, además de comparar la tasa del vecino con la mejor global, se comparará también con la mejor tasa encontrada entre los vecinos generados en el bucle interno, de forma que si la tasa del vecino es más baja que la mejor tasa de los vecinos generados hasta ese momento, aunque sea más alta que la mejor tasa global, dicho vecino no es aceptado, evitando así hacer esta comparación más adelante.

Condición de parada:

- Número máximo de evaluaciones alcanzadas (de nuevo 15000 evaluaciones)

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);

    while (i < s.size()) {
        vecino = CalculaVecino(s);
        tasa_vecino = CalculaCoste(vecino);

        if (tasa_vecino > mejor_tasa) {
            mejor_tasa = tasa_vecino;
            s = vecino;
            i = 0;
        }
        else i++;
    }
```

3.2 PSEUDOCÓDIGO DEL MÉTODO DE ENFRIAMIENTO SIMULADO

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

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

    for (i = 0; evaluaciones < 15000 && !criterio_parada && enfriamientos <
    num_enfriamientos; i++) {
        for (vecinos = 0; vecino < max_vecinos && !enfriamiento; vecinos++) {
            vecino = CalculaVecino(s);
            tasa_vecino = CalculaCoste(vecino); // evaluaciones++

            if (metropolis(vecino)) {
                num_exitos++;
                s = vecino;
                if (tasa_vecino > mejor_tasa) {
                    mejor_tasa = tasa_vecino;
                }
            }
            if (num_exitos > max_exitos) enfriamiento = true;
        }
        Enfriamiento();
        enfriamientos++;
        enfriamiento = false;
        if (num_exitos == 0) parada = true;
        num_exitos = 0;
    }
```

3.3 PSEUDOCÓDIGO DEL MÉTODO DE BÚSQUEDA TABÚ

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

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

    vector posiciones;

    for (i = 0; evaluaciones < 15000; i++) {
        for (j = 0; j < 30; j++) {
            r = Randint(0, s.size() - 1); //s.size()-1 es el número de características (*)
            while (r está en posiciones) {
                r = Randint(0, s.size() - 1);
            }
            posiciones <-- r;
            for (t = 0; t < listatabu.size(); t++) {
                if (listatabu[t] == r) enListaTabu = true;
            }

            if (enListaTabu) {
                vecino = CalculaVecino(s);
                tasa_vecino = CalculaCoste(vecino);
                if(tasa_vecino>mejor_tasa && tasa_vecino>mejor_tasa_vecino) criterio=true;
            }

            if (!enListaTabu || criterio) {

                if (criterio) {    // en este caso la tasa ya está calculada y ya
                                   // se ha comprobado que tasa_vecino > mejor_tasa_vecino
                    mejor_tasa_vecino = vecino;
                    mejor_s = vecino;
                    mejor_r = r;
                }
                else if (!enListaTabu) { // en este caso la tasa no ha sido calculada aun
                    vecino = CalculaVecino(s);
                    tasa_vecino = CalculaCoste(vecino);
                    if (tasa_vecino > mejor_tasa_vecino) {
                        mejor_tasa_vecino = vecino;
                        mejor_s = vecino;
                        mejor_r = r;
                    }
                }
            }
            enListaTabu = false;
            criterio = false;
        }
        mejor_r ---> ListaTabu[];
        if (mejor_tasa_vecino > mejor_tasa) {
            mejor_tasa = mejor_tasa_vecino;
            caracteristicas = mejor_vecino;
        }
    }
```

```
(*) Puesto que en el caso de WDBC el número de características es 30, que coincide
con el número de vecinos a considerar en el bucle interno, podemos ahorrarnos la
generación aleatoria de r y hacer coincidir r = j en cada iteración de j (porque el
orden de la generación de estos vecinos no afecta al resultado)
```

3.4 Pseudocódigo del método de exploración de entorno de la Búsqueda Local

Se trata de la búsqueda local del primer mejor; la exploración de entorno consiste en explorar los vecinos y aplicar el movimiento en cuanto se encuentre una solución vecina que sea mejor que la actual.

La exploración de los vecinos se hace mediante el método `flip(s,i)`, con pseudocódigo:

```
// FLIP
    flip(vector s, int i) {
        vector vecino;
        vecino = s;
        if (vecino[i] == 0) vecino[i] = 1;
        else if (vecino[i] == 1) vecino[i] = 0;

        return vecino;
    }
```

Este método devuelve un nuevo vector, el vector `vecino`, al que se le calcula el coste para ser comparado con el mejor coste obtenido hasta el momento. Si el coste del vecino es mejor, se aplica el movimiento, y se vuelve a empezar de cero la exploración del nuevo vector. Si el coste del vecino no es mejor, se siguen explorando los siguientes vecinos.

3.5 Descripción del cálculo de la temperatura inicial y del esquema de enfriamiento del método de Enfriamiento Simulado

Para el cálculo de la temperatura inicial tomada en el método de Enfriamiento Simulado se hace uso del valor del **coste de la solución inicial $C(S_0)$.** Esta solución inicial se obtiene de forma aleatoria. Así, el valor de la temperatura inicial es:

$$T_0 = \frac{\mu \cdot C(S_0)}{-\ln \phi}$$

donde:

$$\mu = \phi = 0.3$$

El esquema de enfriamiento es el siguiente:

$$T_{k+1} = \frac{T_k}{1 + \beta \cdot T_k}$$

donde:

$$\beta = \frac{T_0 - T_f}{M \cdot T_0 \cdot T_f}$$

$$M = \frac{15000}{\max\_vecinos}$$

$$T_f = 10^{-3}$$

3.6 Pseudocódigo del manejo de la lista tabú en la Búsqueda Tabú básica

Para llevar un buen control de la Lista Tabú, declaro dicha lista como un vector de tamaño $n/3$, donde n es el tamaño del vector binario s, que indica las características tomadas (con valores 1) y las características ignoradas (con valores 0). Por tanto es importante manejar adecuadamente las posiciones de dicha lista.

Supongamos que, en el desarrollo del algoritmo de Búsqueda Tabú, se ha completado una ejecución completa del bucle interno, es decir; hemos considerado 30 vecinos (los índices i que generan dichos vecinos) que cumplen los requisitos exigidos (no estar en la Lista Tabú o, en caso de estarlo, haber superado el criterio de aceptación). De esos 30 vecinos nos hemos quedado con el que ofrece una mejor tasa, y entonces aplicamos el movimiento. Es momento de introducir en la Lista Tabú el índice que ha generado dicho vecino. Como muestra el pseudocódigo:

```
// CONTROL DE LA LISTA TABÚ

    // i es el contador del bucle externo
    // j es la posición de la Lista Tabú en la que escribir
    // r es el índice que generó el mejor vecino de los 30 considerados

    if (i < listatabu.size()) j = i;
    else if (j == listatabu.size() - 1) j = 0;
    else j++;

    listatabu[j] = r;
```

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 **47**.
Para la segunda partición del conjunto de datos se considera una semilla con valor **33**.
Para la tercera partición del conjunto de datos se considera una semilla con valor **21**.
Para la cuarta partición del conjunto de datos se considera una semilla con valor **87**.
Para la quinta partición del conjunto de datos se considera una semilla con valor **14**.

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

- **Parámetros considerados en Enfriamiento Simulado:**

Para $\max_{vecinos} = 10n$ generan demasiados vecinos por cada enfriamiento.

Se considerará entonces $\max_{vecinos} = n$
Los demás parámetros se consideran tal y como se sugiere en el guión.

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	62.6761	96.6667	239.56						
1 – 2	62.807	96.6667	245.897						
2 – 1	62.6761	96.6667	229.941						
2 – 2	62.807	96.6667	232.539						
3 – 1	62.6761	96.6667	246.151						
3 – 2	62.807	96.6667	231.926						
4 – 1	62.6761	96.6667	277.972						
4 – 2	62.807	96.6667	255.898						
5 – 1	62.6761	96.6667	235.6						
5 – 2	62.807	96.6667	236.152						
MEDIA									

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	95.4225	-	3.993	61.1111	-	4.054	63.5417	-	12.926
1 – 2	96.1404	-	3.998	62.7778	-	3.905	63.4021	-	12.897
2 – 1	97.8873	-	4.035	61.1111	-	4.029	63.5417	-	12.848
2 – 2	95.4386	-	3.989	66.6667	-	3.896	63.4021	-	13.065
3 – 1	97.1831	-	3.973	61.1111	-	3.99	63.5417	-	12.868
3 – 2	94.7368	-	3.972	63.8889	-	4.033	63.4021	-	13.262
4 – 1	95.0704	-	3.999	61.1111	-	4.01	63.5417	-	12.699
4 – 2	95.0877	-	4.001	60.5556	-	4.005	63.4021	-	13.001
5 – 1	95.0704	-	3.951	66.6667	-	3.966	62.5417	-	12.846
5 – 2	95.4386	-	4.014	57.7778	-	3.981	63.4021	-	12.909
MEDIA									

Tabla de resultados obtenidos por el algoritmo de Búsqueda Local para el problema de SC

	WDBC			*MOVEMENT_LIBRAS*			*ARRHYTHMIA*		
	%CLAS	**%RED**	**T (seg)**	**%CLAS**	**%RED**	**T(seg)**	**%CLAS**	**%RED**	**T(seg)**
1 – 1	93.3099	23.3333	1164.31	49.4444	31.1111	4137.83			
1 – 2	62.807	66.6667	115.877	53.7778	37.7778	2341.67			
2 – 1	97.5352	16.6667	892.485	45.5556	38.8889	1168.86			
2 – 2	94.7368	16.6667	730.492	55.0000	27.7778	3343.32			
3 – 1	98.2394	20	1695.76	54.1111	25.5556	3880.23			
3 – 2	62.807	60	118.869	52.5556	36.6667	2788.98			
4 – 1	97.8873	10	1902.82	57.7778	25.5556	3921.23			
4 – 2	62.807	73.3333	123.979	41.1111	37.7778	1598.69			
5 – 1	94.7183	23.3333	602.013	47.7778	27.7778	3098.77			
5 – 2	97.8947	23.3333	745.383	42.2222	38.8889	2053.5			
MEDIA	**86.2743**	**32.3333**	**809.199**	**48.4127**	**32.5397**	**2760.31**			

Tabla de resultados obtenidos por el algoritmo de Enfriamiento Simulado para el problema de SC

	WDBC			*MOVEMENT_LIBRAS*			*ARRHYTHMIA*		
	%CLAS	**%RED**	**T (seg)**	**%CLAS**	**%RED**	**T(seg)**	**%CLAS**	**%RED**	**T(seg)**
1 – 1									
1 – 2									
2 – 1									
2 – 2									
3 – 1									
3 – 2									
4 – 1	97.4887*	27.7778	4033.99						
4 – 2	98.2349*	16.6667	5002.20						
5 – 1	95.4225	33.3333	4800.93						
5 – 2	98.2456*	23.3333	5280.19						
MEDIA									

Tabla de resultados obtenidos por el algoritmo de Búsqueda Tabú para el problema de SC

	WDBC			*MOVEMENT_LIBRAS*			*ARRHYTHMIA*		
	%CLAS	**%RED**	**T (seg)**	**%CLAS**	**%RED**	**T(seg)**	**%CLAS**	**%RED**	**T(seg)**
1 – 1									
1 – 2									
2 – 1									
2 – 2									
3 – 1									
3 – 2									
4 – 1									
4 – 2									
5 – 1	96.1233	23.3333	59473.7						
5 – 2									
MEDIA									

6.3 ANÁLISIS DE RESULTADOS

Los resultados se han obtenido con la aplicación de **clasificadores creados erróneamente** (conclusión tras la tutoría del jueves 7 de abril con el profesor Óscar Cordón). Dicho error es causante de algunos de los resultados que resultan poco lógicos, como es el caso del algoritmo **SFS**. Este error también influye en la estimación de los tiempos, ya que en la confusión sobre la creación del clasificador se estaban considerando los conjuntos erróneos (que además eran de mayor tamaño). Estos malos tiempos también son provocados por consideraciones de estructuras en la evaluación de distancias que se podrían evitar y que, al igual que el error de la creación de clasificadores, **quedan pendientes de corregir para la siguiente práctica**.

No obstante, a pesar del mal enfoque del clasificador, se pueden extraer algunos análisis que sí tienen un poco más de sentido.

Por ejemplo, en el algoritmo de **Búsqueda Local** encontramos algunas particiones cuya tasa de clasificación es notoriamente más baja que el resto; esto puede deberse a la tendencia de dicho algoritmo de caer en **máximos locales**.

También es interesante observar la diferencia de tasas de clasificación entre los distintos algoritmos; podemos ver que del algoritmo SFS se obtienen resultados peores que de los demás algoritmos (**Búsqueda Local, Enfriamiento Simulado y Búsqueda Tabú**), los cuales introducen exploraciones más amplias mediante elementos aleatorios, métodos de intensificación y diversificación, etc.

7 BIBLIOGRAFÍA CONSULTADA

- https://es.wikipedia.org/wiki/Validaci%C3%B3n_cruzada