Universidad de Granada

PRÁCTICA 1:
Técnicas de Búsqueda Local y Algoritmos Greddy para el Problema del Aprendizaje de Pesos en Características

Metaheurísticas.
Grupo 2. Martes 17:30-19:30

Realizado por:
Francisco Solano López Rodríguez
DNI: 20100444P
Email: fransol0728@correo.ugr.es

DOBLE GRADO EN INGENIERÍA INFORMÁTICA Y MATEMÁTICAS.
CUARTO CURSO

Índice

1. Descripción del problema. 2

2. Descripción de la aplicación de los algoritmos empleados al problema. 3

3. Descripción en pseudocódigo de la estructura del método de búsqueda 5

4. Descripción de los algoritmos de comparación. 6

5. Procedimiento considerado para el desarrollo de la práctica y manual de usuario. 6

6. Experimentos y análisis de resultados. 8

7. Referencias 13

1. Descripción del problema.

El problema del APC o Aprendizaje de Pesos en Características consiste en optimizar el rendimiento de un clasificador mediante la obtención de un vector de pesos con el que ponderar las carácteristicas de un objeto. Tendremos un conjunto de objetos $\{O_1, ...O_m\}$ donde cada O_i tiene unas características asociadas $\{a_1, ...a_n, C\}$, en las cuales las primeras n características son atributos del objeto y la última característica C es la clase a la que pertenece el objeto. El vector de pesos con el que pretendemos ponderar dichos atributos vendrá dado por $\{w_1, ..., w_n\}$, donde cada w_i pertenece al intervalo $[0, 1]$.

Vamos a usar el clasificador 1-NN, es decir la clase que vamos a asignar al objeto a clasificar es la del vecino más cercano. Para determinar la distancia de un objeto a otro usaremos la siguiente distancia:

$$d_e(e_1, e_2) = \sqrt{\sum_i w_i(e_1^i - e_2^i)^2 + \sum_j w_j d_h(e_1^j, e_2^j)}$$

Donde d_h corresponde a la distancia de Hamming para el caso de variables nominales, aunque en los conjuntos de datos que vamos a utilizar nosotros todas las variables son numéricas.

El objetivo será obtener un sistema que nos permita clasificar nuevos objetos de manera automática. Usaremos la técnica de validación cruzada 5-fold cross validation. Para ello dividiremos en 5 particiones disjuntas al 20 %, con la distribución de clases equilibrada. Aprenderemos el clasificador utilizando 4 de las particiones y validaremos con la partición restante. Este proceso se puede realizar de 5 formas diferentes con lo que obtendremos un total de 5 valores de porcentaje de clasificación en el conjunto de prueba.

Buscaremos optimizar tanto a precisión como la complejidad del clasificador. La función que queremos maximizar es la siguiente:

$$F(W) = \alpha \cdot tasa\_clas(w) + (1 - \alpha) \cdot tasa\_red(W)$$

En donde *tasa_clas* y *tasa_red* corresponden a las siguientes funciones:

$$tasa\_clas = 100 \cdot \frac{n^o\ instancias\ bien\ clasificadas\ en\ T}{n^o\ instancias\ en\ T}$$

$$tasa\_red = 100 \cdot \frac{n^o\ valores\ w_i < 0{,}2}{n^o\ caracteristicas}$$

T es el conjunto de objetos a clasificar, $\alpha \in [0, 1]$ pondera la importancia entre el acierto y la reducción de la solución encontrada y W es el vector de pesos.

2. Descripción de la aplicación de los algoritmos empleados al problema.

Una solución en el problema del Aprendizaje de Pesos en Características es un vector de pesos $W = \{w_1, w_2, \cdots w_n\}$ con los que ponderar la distancia entre 2 objetos. cada valor $w_i \in [0, 1]$, en donde si el valor es menor de 0,2 la característica no se tiene en cuenta para el cálculo de la distancia, si es igual a 1 se tiene totalmente en cuenta y un valor intermedio ponderara la importancia de dicha característica.

La distancia entre dos objetos $e1$ y $e2$ ponderada por el vector W viene dada por la siguiente función:

```
1  Funcion distancia(e^1, e^2, w)
2      sum = 0
3
4      Para i = 0 hasta w.size
5          Si w_i >= 0,2
6              sum = sum + w_i * (e_i^1 - e_i^2)^2
7
8      Devolver sqrt(sum)
9  Fin
```

Para clasificar un nuevo dato e_{new} usaremos el algoritmo 1-NN donde para el cálculo de la distancia se usará un vector de pesos. Viene dada por el siguiente pseudocódigo, en donde el parámetro out indica el indice del elemento que vamos a dejar fuera en el 'leave one out'.

```
1  Funcion KNN(T, new_e, w, out = -1)
2      d_min = 9999999
3
4      Para i = 0 hasta T.size
5          Si out != i
6              d = distancia(T[i], new_e, w)
7              Si d < d_min
8                  c_min = T[i][T[i].size-1]
9                  d_min = d
10
11     Devolver c_min
12 Fin
```

La función objetivo es la combinación con pesos de la tasa de acierto y la complejidad del clasificador donde el valor de α considerado vale 0,5. El objetivo es maximizar dicha función. El parámetro Data corresponde al conjunto de datos sobre el que clasificaremos y T el conjunto de datos que pretendemos clasificar, el parámetro leave_one_out es un booleano que indica si se realizará dicha técnica, la cual será necesaria para clasificar al conjunto de entrenamiento.

```
1  Funcion F(Data, T, w, leave_one_out)
2      alpha = 0.5
3      Devolver alpha*tasaClas(Data, T, w, leave_one_out) + (1-alpha)*tasaRed(w)
4  Fin
```

La función tasaClas calcula la tasa de acierto del clasificador contando el número de aciertos y devolviendo el porcentaje de acierto que ha tenido.

```
1  Funcion tasaClas(Data, T, w, leave_one_out)
2      clasify_ok = 0
3
4      Para i = 0 hasta T.size
5          Si leave_one_out = true
6              out = i
7          Si No
8              out = -1
9
10         Si clasify(Data, T[i], w, out) = T[i][T[i].size-1]
11             clasify_ok = clasify_ok + 1
12
13     Devolver 100.0*clasify_ok/T.size
14 Fin
```

La función tasaRed calcula la tasa de reducción de características con respecto al conjunto original, para ello cuenta el número de elementos del vector de pesos cuyo valor esta por debajo de 0,2, los cuales no serán tomados en cuenta en el cálculo de la distancia.

```
1  Funcion tasaRed(w)
2      num = 0
3
4      Para i = 0 hasta w.size
5          Si w_i < 0,2
6              num = num + 1
7
8      Devolver 100.0*num/w.size
9  Fin
```

La generación de un vecino se realizará mediante la alteración de una componente del vector de pesos W.

$$Mov(W, \sigma) = (w_1, \cdots, w_i + z_i, \cdots, w_n)$$

Donde $z_i \sim N(0; 0{,}4)$, es decir es un valor aleatorio que sigue una distribución normal de media 0 y varianza 0,4. Si el valor de w_i queda fuera de su dominio lo trucamos a $[0, 1]$.

```
1  Funcion nuevoVecino(w, i)
2      z = aleatorio ~ Normal(0, 0.4)
3      w_i = w_i + z
4
5      Si w_i > 1
6          w_i = 1
7      Si w_i < 0
8          w_i = 0
9
10     Devolver w
11 Fin
```

3. Descripción en pseudocódigo de la estructura del método de búsqueda

Algoritmo: Búsqueda local

Primero se inicializa el vector de pesos utilizando una distribución uniforme en $[0, 1]$. También se crea un vector de indices que se permutará cada vez que se haya recorrido entero, y será usado para la exploración del vecindario.

A continuación se ejecuta el algoritmo hasta que se superen las 15000 evaluaciones de la función objetivo o no se encuentre mejora tras generar un máximo de $20 \cdot num\_caracteristicas$ vecinos. Dentro del bucle principal se generará un vecino mutando la componente iésima del vector de pesos que corresponda según el vector de indices comentado antes. La mutación se realizará según la descripción que se realizó en el apartado anterior mediante la función `newNeighbour`.

```
1  Funcion BL(T)
2
3      Para i = 0 hasta num_atributos-1
4          w_i = aleatorio ∈ [0,1]
5          indices[i] = i
6
7      valor = F(T,T,w)
8
9      iteraciones = 0
10
11     Mientras iteraciones < 15000 y nn < 20*num_atributos
12
13         Si iteraciones%indices.size == 0
14             shuffle(indices)
15
16         k = indices[iteraciones%indices.size]
17         copia = w_k
18
19         nuevoVecino(w, k)
20
21         nuevo_valor = F(T, w)
22         iteraciones = iteraciones + 1
23         nn = nn + 1
24
25         Si nuevo_valor > valor
26             nn = 0
27             valor = nuevo_valor
28         Si No
29             w[k] = copia
30
31     Devolver w
32
33 Fin
```

4. Descripción de los algoritmos de comparación.

Algoritmo: Relief

La idea de este algoritmo se basa en modificar los pesos disminuyendo el valor de estos si la distancia al amigo más cercano (e_a) es mayor que a la del enemigo más cercano (e_e) y aumentando en caso contrario, ($w_j = w_j + |e_i - e_e| - |e_i - e_a|$).

```
1  Funcion Relief(T)
2     Para i = 0 hasta T.size
3        enemigo = enemigoMasCercano(T, i)
4        amigo = amigoMasCercano(T, i)
5
6        Para j = 0 hasta w.size
7           w_j = w_j + abs(T[i][j]-T[enemigo][j]) - abs(T[i][j]-T[amigo][j])
8
9     w_max = maximo(w)
10
11    Para i = 0 hasta w.size
12       Si w_i < 0
13          w_i = 0
14       Si No
15          w_i = w_i/w_max
16
17    Devolver w
18 Fin
```

Nota: Para hallar al amigo más cercano tenemos que utilizar la técnica del 'leave one out'.

```
1  Funcion masCercano(T, i, amigo)
2     Para k = 0 hasta n
3        Si amigo
4           Si k != i and T[k][num_atributos-1] == T[i][num_atributos-1]
5              d = distancia(T[i], T[k])
6              Si d < d_min
7                 d_min = d
8                 min = k
9        Si No Si T[k][num_atributos-1] != T[i][num_atributos-1]
10          d = distancia(T[i], T[k])
11          Si d < d_min
12             d_min = d
13             min = k
14
15    Devolver min
16 Fin
```

```
1  Funcion enemigoMasCercano(T, i)
2     Devolver masCercano(T, i, false)
3  Fin
```

```
1  Funcion amigoMasCercano(T, i)
2     Devolver masCercano(T, i, true)
3  Fin
```

5. Procedimiento considerado para el desarrollo de la práctica y manual de usuario.

La práctica ha sido realizada en C++, el código en su mayoría ha sido desarrollado por mi, incluyendo la lectura de datos. Para la generación de números pseudoaleatorios he utilizado la biblioteca `random` la cual proporciona métodos para obtener números aleatorios que sigan una función de distribución dada, en el caso de esta práctica se han usado la función de distribución uniforme y la función distribución normal. Otra biblioteca utilizada es ctime, utilizada para medir los tiempos de ejecución de los algoritmos.

Para el algoritmo de búsqueda local se ha necesitado la generación de números aleatorios, por lo que para no obtener resultados diferentes en cada ejecución se ha inicializado la semilla con valor 14, el motivo de fijarla con este número se debe simplemente a que probé varios valores y este fue con el que mejores resultados obtuve en media.

Para poder ejecutar el programa se ha incluido un makefile en la carpeta FUENTES, por lo que para generar el ejecutable tan solo se deberá de escribir `make` en la terminal. La compilación se ha realizado utilizando clang++ por lo que debería poder compilarse en un Mac. Yo en mi caso he realizado la práctica en Ubuntu. He incluido en la carpeta BIN dos ejecutables uno para Linux (`practica1_linux`) y otro para Mac (`practica1_mac`).

Podemos ejecutar la práctica escribiendo `./practica1`, tras lo cual se mostrará el mensaje siguiente por pantalla:

```
Pulse el número que desee ejecutar:
1: ozone-320.arff (1-NN, relief, BL)
2: parkinsons.arff (1-NN, relief, BL)
3: spectf-heart.arff (1-NN, relief, BL)

Parte voluntaria:
4: ozone-320.arff (relief, relief modificado)
5: parkinsons.arff (relief, relief modificado)
6: spectf-heart.arff (relief, relief modificado)
7: parkinsons.arff (BL, BL alfa = 0.2, BL alfa = 1)
```

Tras pulsar alguno de los números se ejecutarán los algoritmos indicados utilizando los datos del fichero elegido, y se mostrarán los resultados obtenidos. Ejemplo de ejecución:

```
Opcion: 2
parkinsons.arff
1NN
tclass / tasa_red / funcion / tiempo
97.4359 0 48.7179 0.000938268
94.8718 0 47.4359 0.000511219
......
```

6. Experimentos y análisis de resultados.

Bases de datos utilizadas:

- **Ozone:** base de datos para la detección del nivel de ozono, consta de 320 ejemplos, cada uno con 73 atributos y consta de 2 clases.
- **Parkinsons:** base de datos utilizada para distinguir entre la presencia y la ausencia de la enfermedad. Consta de 195 ejemplos, con 23 atributos y 2 clases.
- **Spectf-heart:** base de datos utilizada para determinar si la fisiología del corazón analizado es correcta o no. Consta de 267 ejemplos con 45 atributos y 2 clases.

Comentar que los ficheros de datos proporcionados contenían más ejemplos de los comentados, el motivo era que había varias líneas repetidas, con lo cual muchos ejemplos aparecían varias veces. Para evitar esto he filtrado los datos para eliminar repetidos y con ello ya se cumplen las cifras comentadas. Además los datos han sido también normalizados utilizando la fórmula

$$x_j^N = (x_j - Min_j)/(Max_j - Min_j)$$

Las prácticas han sido implementadas en C++, y ejecutadas en un ordenador con procesador Intel Core i3, 12 GB de RAM y disco duro SSD en el sistema operativo Ubuntu 16.04 LTS.

La práctica también ha sido ejecutada en un Mac y comprobé que los resultados obtenidos, a pesar de haber fijado la semilla, son diferentes a los obtenidos en Ubuntu.

Resultados obtenidos

1-NN	**Ozone**				**Parkinsons**				**Spectf-heart**			
	%clas	**%red**	**Agr.**	**T**	**%clas**	**%red**	**Agr.**	**T**	**%clas**	**%red**	**Agr.**	**T**
P1	79.68	0	39.84	0.0021	97.43	0	48.71	0.0006	75.92	0	37.96	0.0015
P2	82.81	0	41.40	0.0021	94.87	0	47.43	0.0006	64.81	0	32.40	0.0015
P3	81.25	0	40.62	0.0021	94.87	0	47.43	0.0006	67.92	0	33.96	0.0019
P4	77.77	0	38.88	0.0022	97.43	0	48.71	0.0005	71.69	0	35.84	0.0015
P5	80.95	0	40.47	0.0024	97.43	0	48.71	0.0002	73.58	0	36.79	0.0015
Media	80.49	0	40.24	0.0022	96.41	0	48.20	0.0005	70.78	0	35.39	0.0016

Relief	**Ozone**				**Parkinsons**				**Spectf-heart**			
	%clas	**%red**	**Agr.**	**T**	**%clas**	**%red**	**Agr.**	**T**	**%clas**	**%red**	**Agr.**	**T**
P1	82.81	13.88	48.35	0.0225	94.87	4.545	49.70	0.0034	83.33	38.63	60.98	0.0108
P2	78.12	18.05	48.09	0.0202	94.87	4.545	49.70	0.0031	70.37	38.63	54.50	0.0107
P3	79.68	19.44	49.56	0.0198	97.43	4.545	50.99	0.0036	69.81	36.36	53.08	0.0100
P4	80.95	13.88	47.42	0.0200	97.43	4.545	50.99	0.0032	75.47	43.18	59.32	0.0096
P5	79.36	26.38	52.87	0.0206	97.43	0	48.71	0.0040	67.92	40.90	54.41	0.0116
Media	80.18	18.33	49.26	0.0206	96.41	3.636	50.02	0.0034	73.38	39.54	56.46	0.0106

BL	Ozone				Parkinsons				Spectf-heart			
	%clas	%red	Agr.	T	%clas	%red	Agr.	T	%clas	%red	Agr.	T
P1	79.68	79.16	79.42	16.652	89.74	81.81	85.78	1.1339	72.22	63.63	67.92	5.5271
P2	76.56	81.94	79.25	26.105	89.74	81.81	85.78	0.8149	72.22	68.18	70.20	10.118
P3	75	70.83	72.91	21.232	100	72.72	86.36	0.6046	71.69	75	73.34	8.0806
P4	71.42	77.77	74.60	16.267	92.30	90.90	91.60	0.9606	71.69	81.81	76.75	10.382
P5	77.77	84.72	81.25	22.167	94.87	72.72	83.79	0.5796	81.13	79.54	80.33	7.2339
Media	76.09	78.88	77.49	20.484	93.33	80	86.66	0.8187	73.79	73.63	73.71	8.2686

	Ozone				Parkinsons				Spectf-heart			
	%clas	%red	Agr.	T	%clas	%red	Agr.	T	%clas	%red	Agr.	T
1-NN	80.49	0	40.24	0.0022	96.41	0	48.20	0.0005	70.78	0	35.39	0.0016
Relief	80.18	18.33	49.26	0.0206	96.41	3.636	50.02	0.0034	73.38	39.54	56.46	0.0106
BL	76.09	78.88	77.49	20.484	93.33	80	86.66	0.8187	73.79	73.63	73.71	8.2686

Empezamos analizando la tasa de clasificación, para ello se muestra a continuación una gráfica para facilitar la interpretación de los resultados obtenidos.

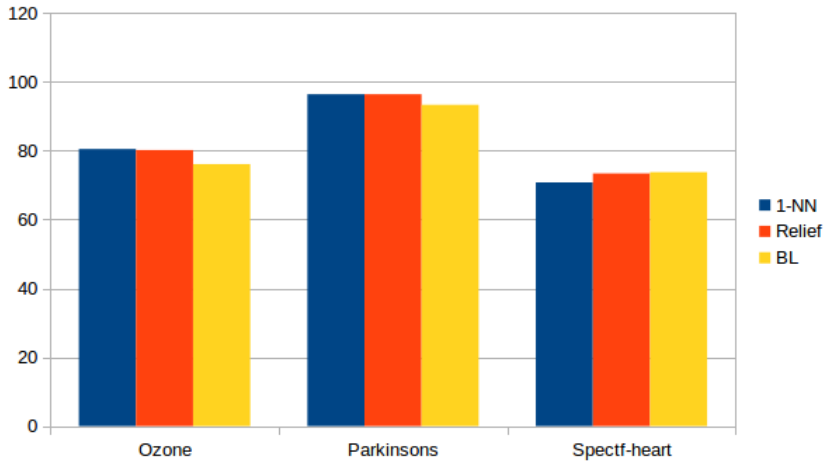

Figura 1: Tasa de clasificación

A la vista de los datos podemos ver que respecto a tasa de clasificación entre el 1-NN y relief no hay gran diferencia aunque el relief está minimamente por encima en media, y el que peores resultados ha dado, a diferencia de lo que se podría esperar en un principio, ha sido el BL, esto podría ser debido a que solo dimos una importancia del 50 % a tasaClas en el algoritmo de búsqueda local, por lo que este más bien busca un equilibrio entre tasa de acierto y tasa de reducción. También es cierto que los resultados obtenidos en la tasa de clasificación van a depender en cierta medida de los datos que dispongamos, así por ejemplo el BL a pesar de estar por debajo del 1-NN y relief en media, en el caso de spectf-heart ha sido el mejor superando por muy poco al relief.

Analicemos ahora la tasa de reducción:

Si nos fijamos en la tasa de reducción claramente, como era de esperar, el BL ha tenido unos resultados absolutamente mejores. Esto no es de extrañar, pues la idea en la que se basa el relief de disminuir el peso si difiere más del amigo más cercano que del enemigo más cercano y aumentar en caso contrario, no se tiene en ningún momento como objetivo mejorar la tasa de reducción, luego si se ha conseguido una leve mejora en la tasa de reducción ha sido más bien por probabilidad. Y nada que decir del 1-NN ya que el vector de pesos es directamente de unos. El único que tenía como objetivo, además de mejorar tasaClas, mejorar la tasa de reducción era el BL.

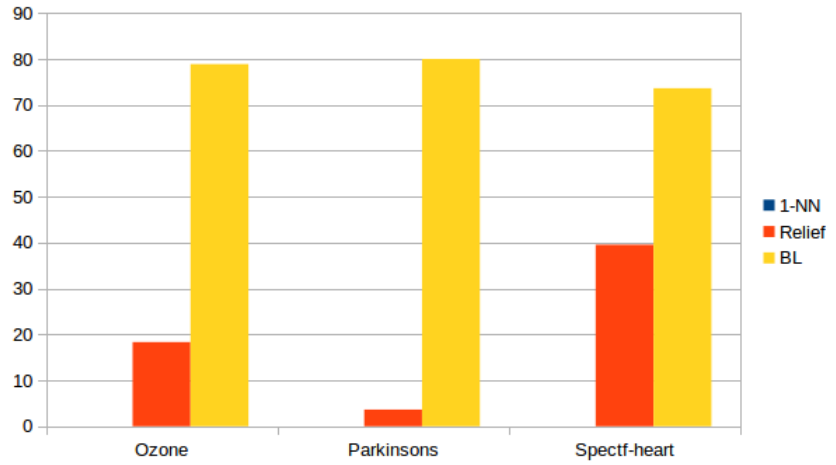

Figura 2: Tasa de reducción

Veamos los resultados de la función objetivo.

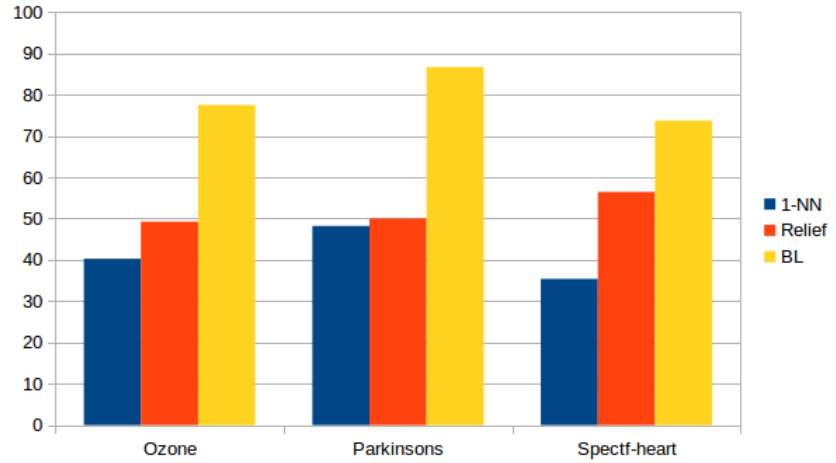

Figura 3: Agregado

Respecto al valor del agregado o valor de la función objetivo, como era evidente, el BL ha obtenido resultados muchos mejores, pues aunque está un poco por debajo en los

resultados de la tasa de clasificación, superá al 1-NN y relief con gran diferencia en la tasa de reducción.

Por último comentar que aunque el BL pueda dar buenos resultados, el tiempo de ejecución es considerablemente alto y más teniendo en cuenta que el tamaño de los datos de los que disponemos no es excesivamente grande, por lo que si el tiempo es una restricción a tener en cuenta para obtener una solución de nuestro problema, tal vez podría ser mas conveniente hacer uso del algoritmo relief o 1-NN. El algoritmo con mejores tiempos ha sido el 1-NN, cosa evidente ya que no realiza cálculos para obtener una solución, directamente su vector de pesos es de unos por lo que clasifica usando la distancia euclidea usual sin realizar ponderaciones. El relief requiere de algo de más tiempo que el 1-NN, pues realiza algunos cálculos para obtener el vector de pesos, pero obtiene resultados algo mejores en clasificación que el 1-NN y además obtiene una tasa de reducción mayor por lo que puede ser preferible.

Experimentos extra

He realizado una modificación en el algoritmo relief en la que solo realizo la búsqueda del amigo y enemigo más cercano para el 20 % de los datos de forma aleatoria. Para ello creo un vector de indices con el tamaño del vector de datos de entrenamiento, realizo una permutación de dicho vector y ejecuto el relief solo para el primer 20 % de dichos índices. Para verlo más claro se muestra el pseudocódigo:

```
function Relief(T)
    for i = 0, i < T.size, i++
        indices.push_back(i)

    shuffle(indices)

    for i = 0, i < T.size/5, i++
        k = indices[i]
        near_enemy = findNearestEnemy(T, k)
        near_friend = findNearestFriend(T, k)

        for j = 0, j < w.size, j++
            w[j] = w[j] + abs(T[k][j]-T[near_enemy][j]) - abs(T[k][j]-T[near_friend][j])
        end for
    end for

    w_max = maximo(w)

    for i = 0, i < w.size, i++
        if w[i] < 0
            w[i] = 0
        else
            w[i] = w[i]/w_max
        end if
    end for

    return w
end
```

Los resultados obtenidos con esta modificación son los siguientes:

Relief 2	Ozone				Parkinsons				Spectf-heart			
	%clas	%red	Agr.	T	%clas	%red	Agr.	T	%clas	%red	Agr.	T
P1	84.37	37.5	60.93	0.0067	92.30	13.63	52.97	0.0017	79.62	56.81	68.22	0.0031
P2	79.68	41.66	60.67	0.0063	92.30	22.72	57.51	0.0010	68.51	43.18	55.85	0.0034
P3	75	50	62.5	0.0065	97.43	31.81	64.62	0.0010	71.69	45.45	58.57	0.0027
P4	84.12	50	67.06	0.0061	100	31.81	65.90	0.0010	77.35	43.18	60.27	0.0034
P5	73.01	66.66	69.84	0.0063	97.43	4.545	50.99	0.0013	73.58	45.45	59.51	0.0032
Media	79.24	49.16	64.20	0.0064	95.89	20.90	58.40	0.0012	74.15	46.81	60.48	0.0032

En la siguiente tabla se muestra la comparación con el algoritmo relief original.

	Ozone				Parkinsons				Spectf-heart			
	%clas	%red	Agr.	T	%clas	%red	Agr.	T	%clas	%red	Agr.	T
Relief	80.18	18.33	49.26	0.0206	96.41	3.636	50.02	0.0034	73.38	39.54	56.46	0.0106
Relief 2	79.24	49.16	64.20	0.0064	95.89	20.90	58.40	0.0012	74.15	46.81	60.48	0.0032

Podemos ver que en la tasa de clasificación el relief modificado en media es levemente inferior, pero las diferencias obtenidas son insignificantes. La gran sorpresa nos la llevamos en la tasa de reducción que se ha visto bastante beneficiada, con lo cual se ha obtenido también una mejora en el valor de la función objetivo. Una de las principales ventajas de este relief modificado es que al utilizar solo el 20 % de los datos los tiempo obtenidos han sido bastante mejores, pero hay que matizar que ela eficiencia del algoritmo sigue siendo la misma, y aunque los tiempos sean más cercanos a los del 1-NN con forme el tamaño de los datos crezca las diferencias en tiempo con el 1-NN van a ser cada vez mayores en proporción.

El siguiente experimento realizado ha sido realizado con el BL en el que se ha utilizado la función objetivo con diferentes valores de α. Primero se ha dado un valor de 1, es decir la función objetivo solo tiene en cuenta la tasa de clasificación. Después se ha dado un valor de 0.2, con lo cual se le ha dado una mayo importancia a la tasa de reducción que a la de clasificación. Los resultados han sido los siguientes:

BL $\alpha = 1$	Parkinsons			
	%clas	%red	Agr.	T
P1	94.87	27.27	61.07	0.4537
P2	100	27.27	63.63	0.5471
P3	100	27.27	63.63	0.4530
P4	94.87	22.72	58.79	0.6790
P5	100	31.81	65.90	0.5029
Media	97.94	27.27	62.61	0.5271

BL $\alpha = 0.2$	Parkinsons			
	%clas	%red	Agr.	T
P1	89.74	81.81	85.78	0.569243
P2	79.48	90.90	85.19	0.827177
P3	92.30	90.90	91.60	0.665967
P4	92.30	90.90	91.60	0.896655
P5	64.10	95.45	79.77	1.09578
Media	83.58	90	86.79	0.810964

Los resultados eran de esperar, se ha obtenido un valor más alto en la tasa de clasificación para el de $\alpha = 1$, de hecho más grande que el BL con $\alpha = 0{,}5$, el 1-NN y el relief. La tasa de reducción obviamente ha sido mayor en el de $\alpha = 0{,}2$, y el agregado ha sido mayor para este mismo también ya que el primero a pesar de tener un valor muy alto en tasaClas ha obtenido un valor muy bajo en la tasa de reducción, debido a que la función

objetivo solo tenía en cuenta la tasa de clasificación.

A continuación se muestra la comparación de los 3 BL con diferentes valores de α para los datos del fichero parkinsons.arff.

	Parkinsons			
	%clas	**%red**	**Agr.**	**T**
BL $\alpha = 0{,}5$	93.33	80	86.66	0.8187
BL $\alpha = 1$	97.94	27.27	62.61	0.5271
BL $\alpha = 0{,}2$	83.58	90	86.79	0.810964

Por último se muestra una gráfica con los valores de la función objetivo de todos los algoritmos utilizando los datos del fichero parkinsons.arff.

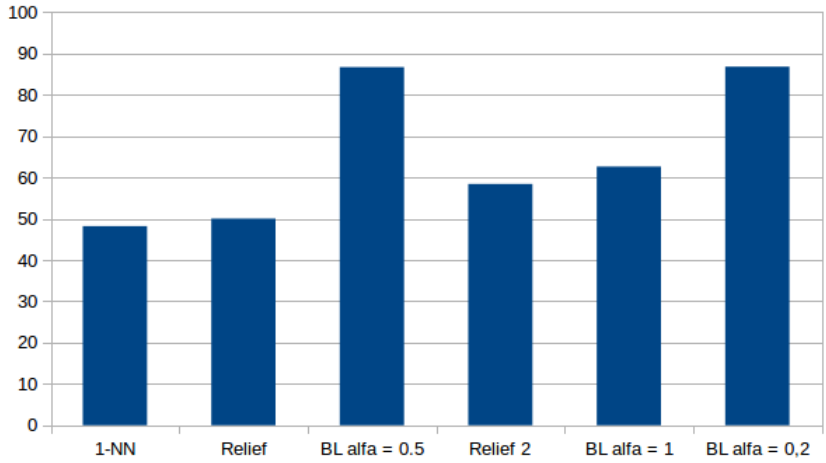

Figura 4: Agregado (parkinsons.arff)

Los dos que han tenido mejores resultados han sido el BL con $\alpha = 0{,}5$ y el BL con $\alpha = 0{,}2$, del primero era de esperar pero el segundo puede ser que sorprenda algo más. Que hayan sido estos dos los que mejores resultados han obtenido puede ser debido principalmente a que eran los únicos que buscaban la maximización tanto de la tasa de clasificación como la de reducción, ya que por ejemplo el BL con $\alpha = 1$ solo tenía en cuenta el acierto dando una importancia del 0 % a la tasa de reducción.

7. Referencias

Además del material proporcionado en la asignatura he consultado la siguiente referencia `http://www.cplusplus.com/reference/random/` en la cual aparece información sobre la biblioteca `<random>` para c++, de la cual necesitaba información acerca del uso de las distribuciones normal y uniforme, para la generación de números pseudoaleatorios que sigan dichas distribuciones.