Metaheurísticas:
Práctica 1.b: Búsquedas por Trayectorias para el Problema de la Selección de Características

Anabel Gómez Ríos.
DNI: 75929914Z.
E-mail: anabelgrios@correo.ugr.es

6 de abril de 2016

Curso 2015-2016
Problema de Selección de Características.
Grupo de prácticas: Viernes 17:30-19:30
Quinto curso del Doble Grado en Ingeniería Informática y Matemáticas.

Algoritmos considerados:

1. Greedy Sequential Forward Selection.
2. Búsqueda Local.
3. Enfriamiento Simulado.
4. Búsqueda Tabú Básica.
5. Búsqueda Tabú Extendida.

Índice

1. Descripción del problema

Queremos obtener un sistema que permita clasificar un conjunto de objetos en unas determinadas clases que conocemos previamente. Para ello disponemos de una muestra de dichos objetos ya clasificados y una serie de características para cada objeto.

El problema es, por tanto, construir un clasificador que se comporte lo suficientemente bien fuera de la muestra de la que disponemos, es decir, que clasifique bien nuevos datos. Para hacer esto, lo que hacemos es particionar la muestra en dos subconjuntos, uno que utilizaremos de entrenamiento para que el clasificador aprenda y otro que utilizaremos para test, es decir, para ver cómo de bien se comporta el clasificador que hemos obtenido con el primer subconjunto fuera de los datos de entrenamiento. Además haremos distintas particiones, en concreto 5 para mayor seguridad y construiremos un clasificador para cada una de ellas. Podemos comprobar cómo de bien se comporta cada clasificador porque sabemos en todo momento las clases de los objetos que tenemos en la muestra y podemos comparar las verdaderas clases con las que el clasificador obtiene.
Buscamos pues en todo momento optimizar la tasa de acierto del clasificador.
Vamos a utilizar además validación cruzada: es decir, para cada partición en dos subconjuntos primero uno será el de entrenamiento y el otro el de test y después les daremos la vuelta y volveremos a construir un clasificador. La calidad por tanto de cada método será la media de los porcentajes de clasificación (la tasa de acierto) para estas 10 particiones.

Nos queda describir cómo aprende el clasificador con los datos de entrenamiento. Ya que podemos llegar a tener muchas características de las cuales podríamos tener algunas poco o nada significantes, lo que hacemos es elegir un subconjunto de características que describan bien los datos de entrenamiento, de forma que en los datos de test sólo tenemos en cuenta este subconjunto de características a la hora de deducir cuál es la clase de cada nuevo dato. Para esta "deducción"vamos a utilizar la técnica de los 3 vecinos más cercanos: buscamos para cada dato los tres vecinos más cercanos teniendo en cuenta las características seleccionadas hasta el momento y nos quedamos con la clase que más veces aparezca. El cómo exploramos el espacio de búsqueda hasta encontrar la mejor solución (el subconjunto de características óptimo) es en lo que se diferencian los distintos algoritmos que vamos a ver en esta práctica.

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

El primer paso común a todos los algoritmos es normalizar los datos de los que disponemos por columnas (es decir, por características) de forma que todas se queden entre 0 y 1 y no haya así preferencias de unas sobre otras.

A continuación se genera una solución inicial, que estará en todos los casos (menos en el algoritmo greedy, que se empieza desde cero) generada aleatoriamente y se irán generando vecinos de esta solución (o se darán saltos, como ya veremos) y nos quedaremos con la mejor solución obtenida.

Todos los algoritmos tienen una condición de parada común, que es llegar a un número máximo de evaluaciones o soluciones generadas: 15000.

2.1. Representación de soluciones

La representación elegida para las soluciones ha sido binaria: un vector de N posiciones, donde N es el número de características, en el que aparece *True* o *False* en la posición i según si la característica i-ésima ha sido seleccionada o no, respectivamente.

2.2. 3NN

Como hemos comentado, la técnica para clasificar que vamos a utilizar en todos los algoritmos será el 3NN. Consiste en considerar, para cada dato, la distancia euclídea entre el dato y todos los demás (excluyéndose él mismo en caso de que estuviéramos preguntando por algún dato dentro del conjunto de entrenamiento (leave one out)) y quedarnos con las clases de los 3 con la distancia más pequeña. La clase del dato en cuestión será de estas tres la que más se repita o, en caso de empate, la clase que corresponda al vecino más cercano.
En cada momento la distancia euclídea se calcula teniendo en cuenta las características que están seleccionadas en el momento, por lo que no podemos tener una matriz fija de distancias, hay que ir calculándolas sobre la marcha.

2.3. Función de evaluación

La función de evaluación será el rendimiento promedio de un clasificador 3NN en el conjunto de entrenamiento: calcularemos la tasa para cada dato dentro del conjunto de entrenamiento haciendo el leave one out descrito anteriormente y nos quedaremos con la media de las tasas obtenidas. El objetivo será por tanto maximizar esta función. La tasa se calcula como 100*(nº instancias bien clasificadas / nº total de instancias).

El pseudo-código es el siguiente:

Se obtiene el subconjunto de entrenamiento que se va a tener en cuenta según las características que se estén considerando.

```
Para cada dato:
    Se saca del conjunto de entrenamiento (leave one out)
    Se calcula la tasa para este dato
    Se acumula la tasa al resto de tasas
Se divide la acumulación de tasas entre el cardinal del conjunto y
lo se devuelve.
```

La función que hace esto la llamaremos `CalcularTasa(conjunto, caracteristicas)` donde caracteristicas es la máscara que nos indica cuáles estamos considerando (aquellas que estén a True).

2.4. Operadores comunes: Generación de vecinos

Se considerarán como vecinas todas aquellas soluciones que difieran en la pertenencia o no de una única característica (si se diferencian en más de una entonces no es vecina de la considerada). Por ejemplo, las soluciones (True, True, False) y (True, False, False) son vecinas porque se diferencian en una sola característica, la segunda.
`Flip` será el operador de vecino: recibe un vector de máscaras y una posición y cambia esa posición en el vector de máscaras.

3. Descripción de los algoritmos

(estructura del método de búsqueda y operaciones relevantes). Pseudocódigo.

3.1. Búsqueda Local

El algoritmo de búsqueda local implementado ha sido el del primer mejor, es decir, exploramos los vecinos de la solución que tenemos en cada momento y en cuanto obtenemos una mejor nos quedamos con ella y empezamos a generar sus vecinos. Los vecinos se empiezan a generar por una característica aleatoria y a partir de esa característica se van cambiando las demás hasta el final y de nuevo por el principio hasta llegar a la primera que habíamos cambiado. Si no nos quedamos con la solución tenemos que dejar la característica que habíamos cambiado como estaba y si nos la quedamos dejamos de generar vecinos de la solución que teníamos y pasamos a generar vecinos de la nueva solución. El algoritmo para cuando da una pasada entera a los vecinos y no ha encontrado una mejor solución que la que tenía.

Veamos el pseudocódigo.
`generarSecuencia(tamaño)` será la función que empieza desde un número aleatorio (menor que el número de características) y de ahí en orden hasta el total de características y empieza de nuevo hasta el número que se ha generado al principio. Esta función nos dará el orden el que recorreremos los vecinos y la llamaremos cada vez que actualicemos la solución.
Al algoritmo le pasamos como parámetros los datos de entrenamiento y las clases de los datos.

```
Empezar
    caract = solucion aleatoria inicial
    tasa actual = calcularTasa(datos, caract)

    while haya mejora en el vecindario y haya menos de 15000 evaluaciones
     hacer:

        posiciones = generarSecuencia(tam(caract))
        for j en posiciones:
            Flip(caract, j)  # Generamos vecino
            calcularTasa(caract)
            if la tasa del vecino es mejor que la actual:
                actualizamos la tasa actual
                vuelta_completa = False # hemos encontrado mejora antes de
                 # generar todos los vecinos)
                break, salimos del bucle de dentro y empezamos a
                 generar vecinos de la nueva solucion

            else, volvemos a la antigua solucion:
                Flip(caract, j)

            if vuelta_completa:
                no ha habido mejora en el vecindario, break
            else:
                vuelta_completa = True
```

```
        if hemos superado el numero de evaluaciones
            break y salimos del bucle de dentro

    Devuelve caract, tasa actual
Fin
```

3.2. Enfriamiento Simulado

El esquema de enfriamiento que he utilizado ha sido el esquema de Cauchy modificado, en el que la temperatura inicial y la modificación en cada iteración del algoritmo se realiza de la siguiente forma:

```
Tf = 0.001, fi = 0.3, mu = 0.3
Tini = (mu*tasa_inicial)/(-np.log(fi))
beta = (Tini - Tf)/(M*Tini*Tf) #M depende del numero de vecinos a generar
                               #y del numero de evaluaciones
Tk = Tk/(1+beta*Tk)       #Al final del bucle de enfriamiento
```

El algoritmo de enfriamiento simulado intenta mejorar el problema de la búsqueda local de quedarse en el primer mínimo local que encuentra, aceptando para ello soluciones que pueden ser peores que la actual bajo algunas circunstancias, en concreto, la temperatura. Al iniciar el algoritmo la temperatura es más alta y se va reduciendo con las iteraciones: la idea es aceptar más soluciones peores que la actual cuanto mayor es la temperatura, es decir, al inicio del algoritmo, e ir aceptando menos progresivamente según la temperatura va bajando. Devolvemos la mejor solución encontrada en general a lo largo del algoritmo. Necesitamos para ello guardar la mejor solución junto con su tasa y la solución actual junto con su tasa.
En cada enfriamiento generamos vecinos hasta un máximo de forma aleatoria y nos la quedamos si es mejor que la actual o supera la probabilidad de aceptar una solución peor.

Con esto, el pseudocódigo es el siguiente:

```
Empezar
caract = solucion aleatoria inicial
mejor solucion = caract
mejor tasa = calcularTasa(datos, caract)
Se calculan los parametros para la temperatura y los vecinos totales
while haya exitos en el enfriamiento actual y Tk>Tf y haya menos de 15000 evals:
    while no se haya generado el maximo de vecinos ni se sobrepasen los exitos:
        pos = num aleatorio entre 0 y n #n num de caracteristicas
        caract = Flip(caract, pos)
        nueva tasa = calcularTasa(datos, caract)
        Se aumenta num vecinos y num evaluaciones
        delta = nueva tasa - tasa actual
        if delta!=0 y (delta>0 o U(0,1)<=exp(delta/Tk)):
            #Se ha aceptado el vecino
            Se actualiza la tasa actual
            Se aumenta el numero de exitos
            if tasa actual > mejor tasa:
```

```
                Se actualizan la mejor solucion y la mejor tasa

        else:
            #No se ha aceptado el vecino, volvemos a la solucion anterior
            Flip(caract, pos)

        Comprobamos si hemos pasado el numero de evaluaciones
    Fin del while interior

    Comprobamos si ha habido exitos en este enfriamiento
    Ponemos el numero de exitos a 0
    Ponemos el numero de vecinos a 0

    Tk = Tk/(1+beta*Tk) #Actualizamos temperatura
    Fin del while exterior

Devolvemos mejor solucion, mejor tasa
Fin
```

$U(0,1)$ es un número aleatorio extraído de la distribución uniforme en el intervalo $[0,1]$.

He tenido que poner como condición adicional en el `if` para aceptar una solución que `delta!=0` porque si las tasas son iguales y la diferencia es cero entonces la exponencial de $delta/Tk$ es siempre 1 al ser delta 0 y un número entre 0 y 1 es siempre menor o igual que 1, con lo que siempre cogía como éxito una solución con la misma tasa, haciendo que el algoritmo tardara mucho más al haber siempre éxitos en el enfriamiento actual, cuando en realidad no se estaba moviendo ni hacia arriba ni hacia abajo.