Búsquedas Híbridas para el Problema de la Selección de Características

Mª Cristina Heredia Gómez, *Metaheurísticas, Universidad de Granada*

En esta práctica se estudia el funcionamiento de Algoritmos Meméticos, y su aplicación en el problema de la selección de características, para lo que se construyen tres variantes de algoritmo híbrido (Genético + Búsqueda Local) que se comparan con Greedy y 3nn.

Descripción del problema

El problema de la selección de características, consiste en seleccionar aquellas características de una base de datos dada que resulten las más representativas, de forma que dicha selección nos permita maximizar la tasa de acierto test de un clasificador.

En estas prácticas tomamos un clasificador Knn con k=3 como función objetivo, por lo que buscamos maximizar la tasa de acierto de clasificación, aplicando metaheurísticas que nos ayuden a elegir correctamente que características seleccionar en cada base de datos.
KNN es un clasificador sencillo basado en distancias, concretamente , con k=3 tenemos en cuenta el voto de los 3 vecinos más cercanos para decidir la clase de un dato dado.

Por lo tanto, nuestra función a maximizar es:

$$tasaclass = 100 \cdot \frac{n^{o}instancias\_bien\_clasificadas}{n^{o}total\_instancias}$$

donde las características se codifican de forma binaria para ser representadas en el problema(1 representa seleccionada/ 0 representa no seleccionada). Para realizar los experimentos se han trabajado con tres bases de datos:

- Wdbc (Wisconsin Database Breast Cancer)
- Movement_Libras
- Arrhythmia

de diferentes tamaños.

Representación del problema

En este problema las soluciones se representan como un vector de ceros y unos, que tendrán la longitud del número de columnas de la base de datos considerada en ese momento, menos uno (menos la clase).
Así pues una posible solución sería s=(0110...00) donde 0 significa que la característica i-ésima de la base de datos no se selecciona y 1 lo contrario.

La **función objetivo** consiste en maximizar la tasa de clasificación mostrada anteriormente. Para ello, **se parte de una solución inicial** que será generada aleatoriamente en todos los casos (excepto en el Greedy que parte de una solución vacía, con todas las características a 0) y se tratará, a lo largo de la ejecución del algoritmo, de mejorar esa solución.

Para esto es fundamental a lo largo de la práctica, el uso de tres funciones: la que genera una solución vecina a partir de una solución dada, siguiendo un **esquema de inversión a nivel de bits**

Algorithm 1 flip

```
procedure FLIP(SOLUCION,INDEX)
    if solucion[index] == 1 then
        solucion[index] ← 0
    else
        solucion[index] ← 1
     return solucion
```

Esta función llamada **flip** recibe como argumento una solución de 0 y 1, y un índice, que es la posición del vector en la que deseamos cambiar el bit, y realiza la inversión del bit presente en dicha posición.

Otra función clave es **getFeaturesForm** que recibe como argumentos un vector de ceros y unos y una base de datos, y devuelve una fómula compuesta por los nombres de las características de la base de datos especificada que están a 1 en el vector. Devuelve una fórmula con forma:

$$class \sim feature_1 + feature_2 + ... + feature_n$$

que posteriormente se la pasaremos al clasificador.

Algorithm 2 getFeaturesForm

```
procedure GETFEATURESFORM(SELECTED,DATASET)
    names ← 0
    featuresList ← 0
    i ← 0
    formula ← 0
    names ← obtener nombres de las columnas del dataset

loop: for i in selected (itera sobre selected)
    if selected[i] == 1 then
        featuresList ← names[i] (mete nombre en la lista)
end loop
        formula ← paste(class,∼,featuresList,separador='+')(componer formula)
return formula
```

La última función, **Adjust3nn** es la que se invoca para ajustar el modelo usando el clasificador 3NN. Recibe como argumentos una fórmula como la mencionada anteriormente y los datos de training de donde tomará las características indicadas en la fórmula, y devuelve el modelo ajustado con los datos

de train.
Para poder hacerlo más modular puse la clase de todos los datasets al final de estos y las nombré como **class** en los tres conjuntos de datos.

Algorithm 3 Adjust3nn

procedure ADJUST3NN(FORMULA,TRAINING DATA)
    modelo ← 0 (inicialmente modelo no contiene nada)
    modelo ← ajusta modelo(formula,training data)
return modelo

Para ajustar el modelo en el código uso el método **train** de la librería **caret**. Para calcular las predicciones y el acierto de test uso **predict** y **postResample**, también de esta librería.

En esta práctica, he creado una función fusionando las dos anteriores en una, pues la función fitness que recibe la función **ga(..)** empleada solo puede recibir un vector binario como único argumento. Unifico por tanto las dos funciones anteriores en una:

Algorithm 4 myfitness

procedure MYFITNESS(SELECTED)
    names ← 0
    featuresList ← 0
    i ← 0
    formula ← 0
    names ← obtener nombres de las columnas del dataset
    evalua ← 0

loop: for i in selected (itera sobre selected)
    if selected[i] == 1 **then**
        featuresList ← names[i] (mete nombre en la lista)
end loop

        formula ← paste(class,∼,featuresList,separador='+')(componer formula)
        modelo ← 0 (inicialmente modelo no contiene nada)
        modelo ← ajusta modelo(formula,training data)
        evalua ← acierto test (modelo)

return evalua

Algoritmos Meméticos estudiados en esta práctica

AM-(10,0.1)

Se implementa un genético estacionario con una población de 10 cromosomas, y cada 10 generaciones se aplica Búsqueda Local sobre un subconjunto aleatorio de cromosomas con probabilidad = 0.1 para cada cromosoma.

Las probabilidades de cruce y mutación son 0.7 y 0.001 respectivamente. El algoritmo se detiene cuando realiza 15000 evaluaciones de la función objetivo, o cuando lleva 500 iteraciones sin mejorar la mejor solución actual. Esta segunda condición la he añadido con el objetivo de acortar el tiempo de ejecución, suponiendo que si en 500 ejecuciones no ha mejorado es porque la búsqueda se estanca, bien porque llega al óptimo, bien por otra razón.
Para el genético, la población inicial se genera de forma aleatoria, y se implementa un esquema generacional con elitismo donde en cada generación se conserva la mejor solución generada,sustituyendo a la peor de la población.
El **operador de selección** empleado es el de torneo binario. El **operador de cruce** que se emplea es el de cruce uniforme, donde los bits se copian aleatoriamente del primer o del segundo padre:

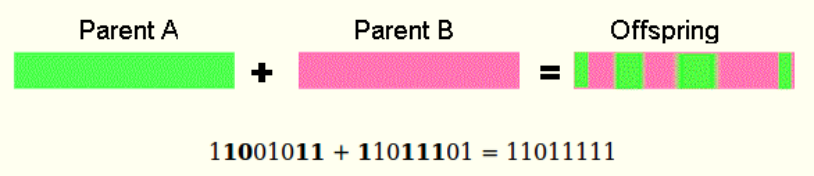

Figura 1: *Operador de cruce uniforme*

El **operador de mutación** empleado es el de mutación aleatoria uniforme, por lo que se toma un bit aleatoriamente y se invierte.

Pseudocódigo del algoritmo Memético:

Algorithm 5 Algoritmo Memético

procedure AM(TAMPOP,PRESION)
popSize ← tamPop
pressel ← presion
nBits ← ncol(dataset)-1
population ← generar Poblacion Aleatoria
pcrossover ← 0,7
pmutation ← 0,001
elitism ← 0,1
poptim ← 0,1

loop: MIENTRAS(no cumpla condición de parada)

evaluar fitness de cada individuo
seleccionar padres por torneo
cruzar padres con prob=pcrossover
mutar individuo con prob=pmutation
optimizar solucion aplicando BL con prob=poptim y pressel
reemplazar población y peor individuo

end loop

El algoritmo recibe como parámetros el tamaño de la población y la presión de selección. De esta forma, con una presión de selección baja, se tienden a seleccionar probilidades prácticamente iguales para cada solución, y por tanto sería como tomarlas aleatoriamente. Para este caso fijo un valor de presión de 0.1. El tamaño de la población es 10 cromosomas. La variable poptim que aparece en la BL es la probabilidad con la que esta se aplica. Como hay que aplicarla cada 10 generaciones, la prob será 0.1.

AM-(10,0.1*N)

En este caso se aplicará el mismo Algoritmo Memético de nuevo,aplicando Búsqueda Local cada 10 generaciones, pero esta vez no sobre un subconjunto aleatorio, sino sobre los 0.1·N mejores cromosomas de la población actual, donde N es el tamaño de ésta.
El algoritmo es el mismo que el del apartado anterior, y se emplean los mismos operadores y el mismo tamaño de población, sin embargo ahoa se fija la presión

de selección a 0.1·N, por lo que la BL empezará en una solución seleccionada por su fitness. A más pressión de selección, selecciona soluciones con mayor valor fitness.

Procedimiento considerado

En la realización de la práctica he usado lenguaje R y el IDE RStudio, beneficiándome de las facilidades que R aporta por no ser un lenguaje orientado a objetos ni fuertemente tipado, así como de las potentes librerías de las que dispone. Por otra parte el empleo de R ha supuesto una pérdida en eficiencia.

Para hacer los Meméticos que se piden, he usado una la librería **GA** (Genetic Algorithms) cuyo autor es Lucca Scrucca de la Università degli Studi di Perugia, ya que es una librería bastante amplia para genéticos, además de muy configurable, que permite aplicar métodos de optimización/mejora sobre las soluciones obtenidas por el genético, como Búsueda local o enfriamiento simulado.

El procedimiento a seguir ha sido el siguiente: en primer lugar se realiza la carga de los datos especificando la ruta de los mismos, y luego he realizado un pequeño "preprocesamiento"que consiste en:

- si hay columnas en los datos con todos los valores iguales, eliminarlas.
- normalizar los datos, sin normalizar la clase.
- tomar la columna de la clase y colocarla al final de todas las bases de datos.
- nombrar igual a las columnas con la clase de los 3 datasets."**class**", para hacerlo más modular en las funciones.

Una vez listos los datos, he usado, como mencioné arriba, el knn de la librería caret de R, fijando el k a 3, como clasificador y su acierto test como resultado a maximizar.
También he usado el método **createDataPartition** de esta librería, para hacer el particionamiento estratificado de los datos. Parto los datos cada vez con una semilla, que es i·9876543, donde i va de 1 a 5(cada interación de CV), y que se replica para cada ejecución de cada algoritmo distinto lanzada con cada base de datos. Luego se intercambian las particiones.
El objetivo es que los resultados obtenidos para cada algoritmo sean realmente

comparables.

Para lanzar el memético, la especificación es la siguiente:
$GA <- ga(type = "binary", fitness = myfitness, nBits = ncols, population = gabin_Population, selection = gabin_tourSelection, crossover = gabin_uCrossover, mutation = gabin_raMutation, popSize = 10, pcrossover = 0,7, pmutation = 0,001, elitism = 0,1, maxiter = 15000, run = 500, optim = TRUE, optimArgs = list(method = "L-BFGS-B", poptim = 0,1, pressel = (0,1*N), control = list(maxit = 1)))$

donde:

- **type** es el tipo de codificación del problema
- **fitness** es la función fitness a maximizar
- **nBits** número de bits de una solución. En este caso ncol(dataset)-1 (le quitamos la clase)
- **population** como generamos la población inicial. En este caso, generamos aleatoriamente cromosomas binarios
- **selection** operador de selección. Elegimos *gabin_tourSelection* que es el torneo binario
- **crossover** operador de cruce. Hay varios, seleccionamos el op de cruce binario uniforme
- **mutation** operador de mutación. Seleccionamos el binario aleatorio uniforme
- **popSize** tamaño de la población
- **pcrossover** probabilidad de cruce
- **pmutation** probabilidad de mutación
- **elitism** número de mejores soluciones que se conservan entre generaciones. Conservamos solo la mejor, luego lo ponemos a 0.1
- **maxiter** número máximo de iteracciones antes de que el algoritmo se detenga

- **run** máximo de iteracciones consecutivas sin mejora antes de que el algoritmo se detenga. añado esta opción y la pongo a 500 iteracciones porque permite ganar mucho en tiempo de ejecución.
- **optim** si se pone a TRUE indica que se va a aplicar método de optimización sobre las soluciones
- **optimArgs** lista de argumentos de configuración del método de optimización a aplicar
- **method** nombre del método de optimización a aplicar, en este caso, Búsqueda Local primero el mejor
- **poptim** probabilidad de ejecutar la Bl en esa iteracción
- **pressel** presión de selección
- **control** lista con parámetros de control para la BL como el número de iteracciones a realizar

El 5x2 está hecho de forma funcional y se encuentra en la sección de ejecución del algoritmo correspondiente, en el Rscript. Se separan ejecuciones de 5 en 5, en train versus test y test versus train. Las tasas de reducción correspondientes de calculan justo debajo.

El Algoritmo greedy usado como comparativa, es el usado en prácticas anteriores, que en pseudocódigo se puede definir como:

Algorithm 6 greedyRndm

procedure GREEDYRNDM(TRAINING,TEST,SEED)
featuresList ← lista de caracteristicas del dataset
final ← FALSE
selectedAndCandidate ← 0 (inicialmente no hay seleccionadas)
ganancias ← 0 (inicialmente está vacía)
selected ← 0 (inicialmente está vacía)
cmejor ← 0
cpeor ← 0
umbral ← 0
alpha ← 0.3
randomFeature ← 0
evalua ← 0
bestAccu ← 0 (mejor acierto test hasta ahora)
LRC ← 0 (inicialmente está vacía)

loop: mientras(featuresList!=0 AND !final)
ganancias ←calcular ganancia test de cada caracteristica por separado
cmejor ←max(ganancias)
cpeor ←min(ganancias)
umbral ←cmejor-(alpha(cmjor-cpeor))
LRC ←caracteristicas cuyas (ganancias>=umbral)
randomFeature ← caracteristica alatoria de LRC
selectedAndCandidate[randomFeature] ← 1
evalua ← ajustar de nuevo usando las caracteristicas de selectedAndCandidate

if evalua > bestAccu **then**
bestAccu ← evalua
selected ← selectedAndCandidate
featuresList[randomFeature] ← no se puede volver a seleccionar

else final ← TRUE

selectedAndCandidate ← selected

end loop
return list(selected,bestAccu)

Análisis de los resultados

Se observa en las tablas de resultados(ver resultados_tablas.ods) que el algoritmo híbrido da los mejores resultados para este problema. Aunque solo se compare con Greedy y 3nn, en otras prácticas se han probado otros algoritmos y este es, sin duda, el que mejores resultados da en estas bases de datos.

Greedy obtiene un resultado algo superior que el Memético que aplica Bl sobre un subconjunto aleatorio de cromosomas, pero inferior para el resto de casos de estudio.
En cuanto a los tiempos de ejecución Greedy al tener menos cómputo es el que menos tarda. Le sigue AM(10,0.1) que tarda menos que AM(10,0.1·N) debido a la selección aleatoria de cromosomas con prob=0.1, ya que evita tener que buscar los 0.1N mejores.

En cuanto a la calidad de los resultados El mejor algoritmo en términos de calidad, al menos para las bases de datos chica y mediana, parece que es el memético que aplica BL sobre las mejores soluciones, pues obtiene en media las mejore marcas de aciertos en test. Sin embargo, no es así para el caso de Arritmia, donde observamos que obtiene peores resultados en calidad que la otra versión de AM implementado e incluso que Greedy. Aunque esto puede deberse en parte a la población inicial, pues he observado que en las 5 particiones parte de una población inicial mala y además se estanca pronto la búsqueda.

En cuanto a las tasas de reducción, se observa que ambos meméticos seleccionan cerca de la mitad de características que selecciona greedy y superan sus resultados en media por lo que muchas de las características que selecicona greedy son prescindibles.

A pesar de que los resultados no son malos, creo que se podrían mejorar ajustando mejor los parámetros. Se podría intentar elevando el grado de elitismo (ahora mismo sólo se conserva la mejor solución entre poblaciones) y aplicando BL con más probabilidad o más interacciones , o incluso cambiando la BL por otro método de optimización como S.Annealing. En calquier caso, con los ajustes actuales ganamos en tiempo.
He observado que las búsquedas siempre se estancan antes de las 15000 evaluaciones, por lo que, como mencioné arriba, he incorporado una segunda condición para que si el algoritmo lleva sin mejorar la mejor solución 500 iteraciones, se detenga, ahorrando así en tiempo y cómputo.

Ilustración de resultados

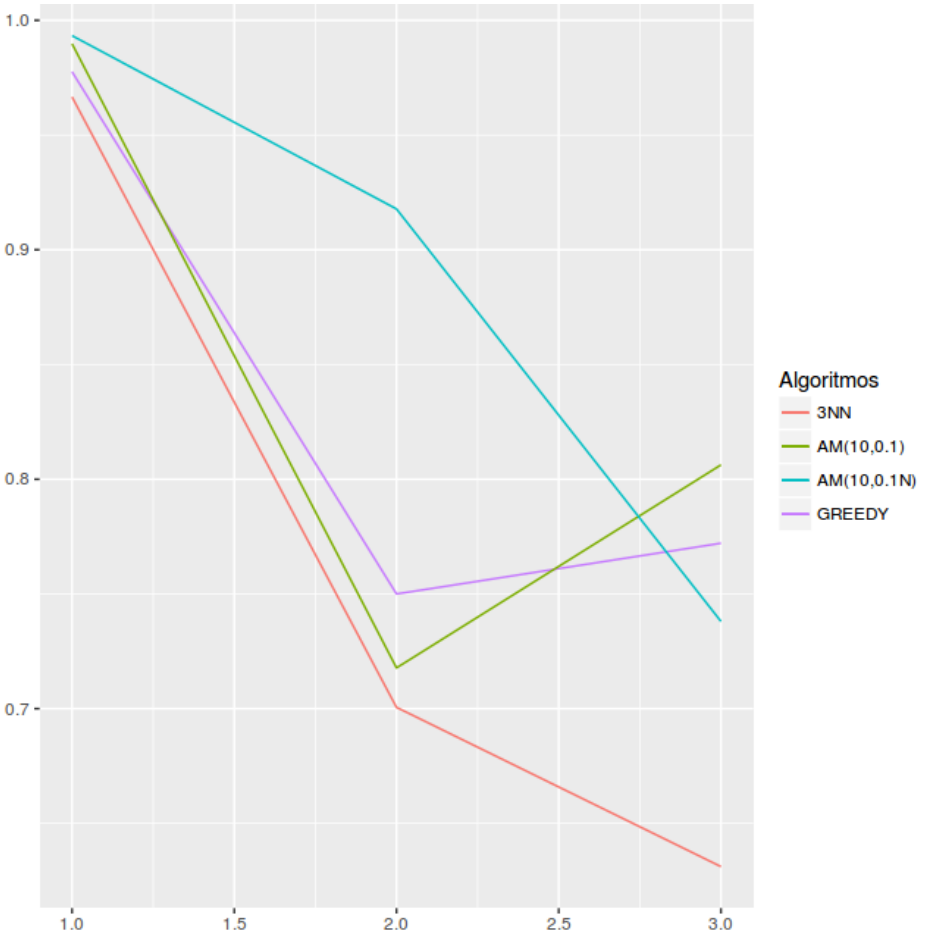

Figura 2: *Comparativa global de rendimiento*

Donde en el eje y se representa de 0 a 1 el % de clasificación test obtenido, y en el eje X,tenemos :

- 1 : representa WDBC
- 2 : representa Movement Libras
- 3 : representa Arritmia

Referencias

https://cran.r-project.org/web/packages/GA/GA.pdf https://github.com/luca-scr/GA/blob/master/R/ga.R