Optimización basada en Colonias de Hormigas para el Problema de la Selección de Características

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

En esta práctica se estudia la aplicación de dos algoritmos de optimización basados en Colonias de Hormigas: Sistema de Colonias de Hormigas(SHC) y Sistema de Hormigas Max-Min(SHMM),ambos incorporando Búsqueda Local, para optimizar las soluciones obtenidas al problema de la selección de caracteríticas, en tres bases de datos distintas.

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.

Técnicas de Colonias de Hormigas estudiadas en esta práctica

SCH-BL

En este algoritmo se trata de imitar el comportamiento que las hormigas siguen en colonia, para resolver el problema de maximizar la función objetivo considerada.

Para ello, se consideran 10 hormigas en la colonia, y cada una de ellas construirá una solución a lo largo de cada ejecución del algoritmo. Cada una de esas soluciones, será posteriormente optimizada por una Búsqueda Local que realiza una única iteración, y se devolverá la mejor solución obtenida después de este proceso, a lo largo de todas las iteracciones.
Para la implementación del algoritmo, se tienen en cuenta dos rastros de feromona:

- τ-car_{f_i} : se asocia un rastro de este tipo a cada nodo(a cada característica). Indica la preferencia memorística de seleccionar esa característica.
- τ-num-car_{nc} : se asocia un rastro a cada valor posible del número de características a seleccionar por cada hormiga.

τ-car_{f_i} se inicializa a 10^{-6} mientras que τ-num-car_{nc} se inicializa a $\frac{1}{N_c}$. Mientras que los primeros se actualizan local y globalmente durante el algoritmo, los segundos lo hacen solo una vez. Es decir τ-car_{f_i} se actualiza durante la construcción de las solucciones por las hormigas, mediante la regla de actualización local de feromona:

$$\tau_{rs}(t) = (1 - \phi) \cdot \tau_{rs}(t - 1) + \phi \cdot \tau_0$$

pero también se actualizan estos rastros de feromona globalmente, mediante la fórmula: $\tau_{rs}(t) = (1-\rho)\cdot\tau_{rs}(t-1)+\rho\cdot\Delta\tau_{rs}^{mejor\_global}$ donde $\Delta\tau_{rs}^{mejor\_global}$=Coste($S_{mejor\_global}$), es decir, solo aquella hormiga que generó la mejor solución hasta ahora modificará globalmente esta feromona.

En cuanto a los segundos rastros de feromona: τ-num-car_{nc} , se actualizan una vez que se han calculado las soluciones por cada una de las hormigas, antes de aplicar sobre ellas la búsqueda local. Para ello se utiliza la regla:

$$\tau_{rs}(t) = (1 - \rho) \cdot \tau_{rs}(t - 1) + \sum_{k=1}^{m} \Delta\tau_{rs}^{k}$$

donde m es el número de hormigas, y $\Delta\tau_{rs}^{k}$=Coste(S_k), es decir, el coste de

cada solución.
Con estos rastros de feromona, se usa una distribución de probabilidad con la que se construye una ruleta que se gira, una vez por hormiga (por iteración), para escoger el número de caracteristicas a seleccionar por esa hormiga en esa iteración.
Para construir esa **ruleta**, creo una distribución de probabilidad con los rastros de feromona, usando la función **dnorm**. Sobre esta distribución, calculo las probabilidades cumulativas de los elementos, y finalmente, para simular el giro de la ruleta, tomo un valor entre el mínimo y el máximo de estas probabilidades cumlativas y devuelvo el primer elemento cuya Prob.Cumulativa sea mayor a dicho valor.

Con respecto a qué características escoge la hormiga, hay que tener en cuenta la **información heurística**, que se define como la preferencia heurística de escoger individualmente la caracteríctica f_i con respecto las clases C, como:

$$\eta_{f_i} = I(C, f_i) = \sum_{c=1}^{N_c} \sum_{f_{ij}=1}^{N_{fi}} P(c, f_{ij}) \cdot \log_2(\frac{P(c, f_{ij})}{P(c) \cdot P(f_{ij})})$$

Donde se discretiza cada característica en 10 intervalos(si es necesario discretizarla), por frecuencias, para lo que uso la función **discretize(..)** del paquete **arules**.
Pseudocódigo del cálculo de la heurística:

Algorithm 4 Información Heurística

procedure HEURISTIC(TRAINING,FEATURE)
    discretizedFeature ← 0
    heuristic ← 0

    if feature toma valores 0,1 **then** discretizedFeature ← factores(feature)

    else discretizedFeature ← discretizar(feature,freq,h=10)
loop: para cada clase
    loop: para cada f_{ij}
    heuristic ←heuristic $+P(clase, f_{ij}) \cdot \log_2(\frac{P(clase, f_{ij})}{P(clase) \cdot P(f_{ij})})$
    end loop
end loop

return suma(heuristic)

Esa información junto con los rastros de feromona τ-car_{f_i} serán los criterios usados por la hormiga a la hora de tomar la característica i-ésima.

Por lo tanto, la regla de transición a aplicar será:

$$s = \begin{cases} argmax_{u \in J_k(r)[\tau_{ru}]^\alpha \cdot [\eta_{ru}]^\beta} & si \text{ q} \leq q_0 \\ S & \text{en otro caso} \end{cases}$$

donde α=1, β=2, q_0=0.8 y S es la probabilidad con que la hormiga situada en k decide moverse a la ciudad(tomar la característica) s:

$$p_k(r,s) = \begin{cases} \frac{[\tau_{ru}]^\alpha \cdot [\eta_{ru}]^\beta}{\sum_{u \in J_k(r)} [\tau_{ru}]^\alpha \cdot [\eta_{ru}]^\beta} & si \text{ s} \in J_k(r) \\ 0 & \text{en otro caso} \end{cases}$$

En psecódigo, la función de trancisión se define como:

Algorithm 5 Transition Rule

procedure TRANSITION_RULE(ANTSOL,HEUR,TAOCAR)
alpha ← 1
beta ← 2
q_0 ← 0,8
q ← random entre 0:1
selectedFeature ← 0
probs ← 0
pheromoneDotHeuristic ← 0
pheromoneSum ← 0

loop: para cada caracteristica factible
pheromoneDotHeuristic ← $taoCar^{\alpha} \cdot heur^{\beta}$
end loop
pheromoneSum ←$\sum (pheromoneDotHeuristic)$

if q ≤ q_0 **then** selectedFeature ← Index(Max(pheromoneDotHeuristic))

else
loop: para cada caracteristica calcular prob de ser tomada

if es factible **then** probs ← pheromoneDotHeuristic/pheromoneSum

else probs ← 0
end loop

selectedFeature ← Index(Max(probs))
return selectedFeature

Definido todo esto, se describe en pseudocódigo el SHC-LS como:

Algorithm 6 SHC-LS

procedure SHC_LS(TRAINING,TEST)
tao_car $\leftarrow 10^{-6}$, tao_num_car $\leftarrow \frac{1}{N_c}$, nEval $\leftarrow 0$, nAnts $\leftarrow 10$
featuresToSelect $\leftarrow$ 0(vector vacio inicialmente) , $\phi \leftarrow$ 0,2
heuristics $\leftarrow$ 0(vector vacio inicialmente) , $\rho \leftarrow$ 0,2
best_Global $\leftarrow$ 0(inicialmente mejor S.Global es 0)
best_Actual $\leftarrow 0$, solutionsCostSum $\leftarrow 0$
loop: para cada z=1 hasta numero Caracteristicas
heuristics $\leftarrow$ HEURISTIC(training,training[[z]])
end loop
loop: while(nEval < maximo de Iteraciones)
loop: para i=1 hasta nAnts
L[i] $\leftarrow$ 0(se crean vacias las soluciones de las hormigas)
end loop

tao_num_car $\leftarrow$distribución Prob de tao_num_car
loop: para i=1 hasta numero Caracteristicas
featuresToSelect[i] $\leftarrow$giro ruleta
end loop
loop: para i=1 hasta numero Caracteristicas
loop: para j=1 hasta nAnts
transition $\leftarrow$Transition_Rule(L[[j]],heuristics,tao_car),L[j][transition] $\leftarrow 1$
tao_car[transition] $\leftarrow$ actualizacionLocalDeFeromona
end loop end loop

loop: para r=1 hasta nAnts
solutionsCostSum $\leftarrow$ solutionsCostSum + Coste(L[r])
end loop
loop: para i=1 hasta nAnts
tao_num_car $\leftarrow$ Actalizar feromona según regla
end loop
loop: para t=1 hasta nAnts
L[t] $\leftarrow$ BL(training,test,L[t])
end loop
nEval $\leftarrow$ nEval + 20 , best_Actual $\leftarrow$ Mejor de L

if best_Actual > best_Global **then** best_Global $\leftarrow$ best_Actual
loop: para i=1 hasta numero Caracteristicas
tao_car[i] $\leftarrow$ actualizacion Global de feromona
end loop
end loop
return (best_Global)

La Búsqueda Local empleada es la **LocalSearchModified** de las prácticas anteriores pero modificada para que realice una única iteracción, encuentre mejora sobre la solución o no.
Coste es el coste de esa solución, es decir, la tasa de acierto test obtenida por el clasificador para esa selección de características. La suma de los costes obtenidos por las soluciones iniciales de las hormigas (antes de aplicar BL), serán usados para actalizar la feromona τ-num-car, mediante aplicación de la regla del Sistema de Hormigas, descrita anteriormente.

En cuanto a la ruleta, como ya expliqué arriba, se implementa usando Probabilidades cumulativas , pero no se dan detalles en el pseudocódigo por simplicidad, al igual que ocurre con la devolución del algoritrmo: a pesar de que en el pseudocódigo soolo se devuelve la mejor solución Global encontrada, en la implementación en código devuelvo también su coste (ya mejorado), así como su coste inicial.
Por último, incremento el número de evaluaciones de 20 en 20 porque lo incremento justo después de aplicar Búsqueda local sobre las soluciones, y es dentro del algoritmo de búsqueda local donde calculo el coste de la solución inicial y el coste de la mejorada.

SHMM-BL

Este algoritmo supone una variación sobre el anterior. En este caso el número de hormigas considerado y el número de evaluaciones siguen siendo los mismos, al igual que los parámetros α, β y ρ. También se mantiene la regla de transición del SHC y el cómputo de la información heurística, así como los dos rastros de feromona considerados, y en ninguno de los dos algoritmos implementados se aplica reinicialización de la búsqueda.
Los cambios que incluye son:

- nuevo mecanismo de actualización de feromona
- topes para los rastros de feromona

Esta vez la actualización de feromona τ-car se realiza de forma más agresiva, aportando sólo los rastros de feromona de la mejor solución, por lo tanto, estos rastros de feromona ya no se actualizan local y globalmente, sino que se actualizan conforme a la mejor solución global de esa iteración, siguiendo la regla:

$$\tau_{rs}(t) = (1 - \rho) \cdot \tau_{rs}(t - 1) + \Delta\tau_{rs}^{mejor}$$

donde mejor es la mejor solución global. Es decir, el aporte a realizar por la hormiga será el coste de la mejor solución global.
En este algoritmo, se incorporan también unos límites superior e inferior sobre estos rastros de feromona: τ_{max} y τ_{min}.
Antes de la ejecución del algoritmo se genera una solución aleatoria , inicializando τ_{max} a Coste($S_{aleatoria}$)/ρ y τ_{min} a $\tau_{max}/500$. después se inicializa τ-car$_{fi0}$ a τ_{max} y comienza la ejecución del algoritmo. Cada vez que se actalice la mejor solución global del algoritmo, se actualizarán también τ_{max} a Coste(S_{mejor_global})/ρ y τ_{min} a $\tau_{max}/500$.

Cuando se actualiza la feromona τ-car, se deben recorrer todos esos rastros y comprobar si exceden los límites inferior/superior, en cuyo caso deben truncarse.
En canto a los otros rastros de feromona, τ-num-car , se actualizan como antes, en función del número de características escogidas por cada hormiga y siguiendo la regla del Sistema de Hormigas expuesta en el algoritmo anterior.
La descripción en pseudocódigo de este algoritmo, es:

Algorithm 7 SHMM-LS

procedure SHMM_LS(TRAINING,TEST)
$\rho \leftarrow$ 0,2 , RandomSolIni $\leftarrow$ (generar solucion aleatoria)
AccuracyInitial $\leftarrow$ Coste(RandomSolIni)
best_Accuracy_Global ,best_Accuracy_Actual $\leftarrow$ AccuracyInitial
best_Global $\leftarrow$ 0(inicialmente mejor S.Global es 0)
$\tau_{max} \leftarrow$ (AccuracyInitial/ρ) , $\tau_{min} \leftarrow$ (τ_{max}/500) , τ-car $\leftarrow \tau_{max}$
tao_num_car $\leftarrow \frac{1}{N_c}$, nEval $\leftarrow$ 0 , nAnts $\leftarrow$ 10 , heuristics $\leftarrow$ 0
featuresToSelect $\leftarrow$ 0(vector vacio inicialmente), solutionsCostSum $\leftarrow$ 0

loop: para cada z=1 hasta numero Caracteristicas
heuristics $\leftarrow$ HEURISTIC(training,training[[z]])
end loop

loop: while(nEval < numero iteracciones deseadas)
loop: para i=1 hasta nAnts
L[i] $\leftarrow$ 0(se crean vacias las soluciones de las hormigas)
end loop

tao_num_car $\leftarrow$distribución Prob de tao_num_car
loop: para i=1 hasta numero Caracteristicas
featuresToSelect[i] $\leftarrow$giro ruleta
end loop

loop: para i=1 hasta numero Caracteristicas
loop: para j=1 hasta nAnts
transition $\leftarrow$Transition_Rule(L[[j]],heuristics,tao_car),L[j][transition] $\leftarrow$ 1
end loop
end loop

loop: para t=1 hasta nAnts
L[t] $\leftarrow$ BL(training,test,L[t])
end loop

nEval $\leftarrow$ nEval + 20 , best_Actual $\leftarrow$ Mejor de L

if best_Accuracy_Actual > best_Accuracy_Global **then** best_Global $\leftarrow$ best_Actual , $\tau_{max} \leftarrow$ (best_Accuracy_Global/ρ) , $\tau_{min} \leftarrow$ (τ_{max}/500)

loop: para r=1 hasta nAnts
solutionsCostSum $\leftarrow$ solutionsCostSum + Coste(L[r])
end loop

Algorithm 8 SHMM-LS

loop: para i=1 hasta nAnts
tao_num_car ← Actalizar feromona según regla
end loop

loop: para i=1 hasta numero Caracteristicas
tao_car[i] ← actualizacion Global de feromona según regla
end loop
loop: para t=1 hasta num τ-car
truncar τ-car[i] si excede limites
end loop
return (best_Global)

De nuevo, el algoritmo en pseudocógido devuelve la mejor solución global por simplicidad. En la implementación en código devuelve tanto la mejor solución global, como la inicial de esta, y su coste.
Las reglas de actualización de feromona mencionadas en el pseucodódigo han sido detalladas anteriormente. Para la implementación de la ruleta de usa el mismo mecamisno que en el SHC-BL, y para truncar las feromonas τ-car simplemente se comprueba que si excede el máximo, se le asigna τ_{max} y si excede el mínimo, se le asigna τ_{min}.
En cuanto al número iteracciones deseadas, serán las evaluaciones de la función objetivo que deseemos hacer antes que el algoritmo se detenga.

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 gran pérdida en eficiencia, llegando a tener en algunos casos ejecuciones de hasta 14h, para ejecutar 10 particiones.

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.

Además de usar **caret** he usado otras librerías como **foreign** para leer datos en arff y **parallel** para paralelizar(aunque esta vez no he podido lanzar la versión paralela por falta de memoria). También he usado la librería **arules** para usar funciones de discretización.
Tanto las metaheurísticas como funciones auxiliares necesarias son de implementación propia. Se sustituyen bucles por versiones apply.

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.

A pesar de que se piden 15.000 evaluaciones, para que me diera tiempo he tenido que bajar el número de iteracciones a **9.000** para SHC-BL y SHMM-BL. El Algoritmo greedy usado como comparativa, es el usado en prácticas anteriores, que en pseudocódigo se puede definir como:

Algorithm 9 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 aplicando Greedy y SCH-BL llegamos a mejorar el acierto de test que obtenemos con el 3NN tomando todas las características (menos la clase), por lo que realmente se están seleccionando características relevantes de los conjuntos que nos ayudan a optimizar la solución.
Algo diferente pasa con SHMM-BL, que solo consigue mejorar al 3NN para el caso de arritmia, anque en las otras bases de datos alcanzan resltados similares, y en ningún caso queda por encima de los resultados obtenidos por SCH-BL.

Esto puede deberse a que la búsqueda llega un momento en el que necesita reinicializar o a algún problema en la implementación, pero SHMM-BL da peores resultados de los que esperaba.

En cuanto a los tiempos de ejecución Parece en los resultados de las tablas que el algoritmo que menos tarda en ejecutarse es el Greedy, ya que obtiene los mínimos tiempos medios entre los tres algoritmos estudiados, para las tres bases de datos. En una primera impresión creo que se debe a la condición de parada del Greedy de que si no encuentra mejora, acaba, lo que le ahorra evaluaciones, además de que los algoritmos de hormigas tienen más cómputo interno, dado que tienen que calcular la heurística, función de transición...

En cuanto a la calidad de los resultados En primer lugar he de decir que el número de evaluaciones a considerar era 15.000 y yo lo he reducido a 9.000 por reducir los tiempos, por lo que seguramente se habrían podido obtener soluciones mejores a lo largo de 6.000 evaluaciones más.
Aún así, se obtiene que el SCH-BL da mejores resultados que Greedy en WDBC y Movement Libras, mientras que Greedy obtiene mejores resultados en Arritmia.
Observando las tasas de reducción en media, observamos que para WDBC son de 82.903224 con Greedy, 47.096773 con SHC-BL y 33.548386 con SHMM-BL , mientras que para Movement Libras son de 90.9645888889 con Greedy, 62.637363 con SHC-BL y 41.428572 con SHMM-BL, lo cual nos da una idea de que, parece que en WDBC y Movement Libras funciona más tomar un número moderado de características, que es más propio de lo que hacen las hormigas.
Sin embargo, para Arritmia obtenemos como tasas de reducción 97.8127755556 para greedy, 72.362205 para SHC-BL y 84.803149 para SHMM-BL. El hecho de que en este caso fncione mejor un Greedy puede deberse a que selecciona más características que los otros dos y a que además las selecciona mejores. Las

hormigas se guían por los niveles de feromona para tomar una característica u otra. Greedy se basa en el coste adquirido al añadir esa característica.

Conlusión: si hubiera que decantarse por uno de los 3 algoritmos comparados para resolver el problema de la selección de características **en estas bases de datos** , en base a los resultados obtenidos con 9.000 evaluaciones, me parece que el mejor balance Calidad-tiempo-Complejidad lo tiene Greedy.(Aunque me duele admitirlo).

Ilustración de resultados

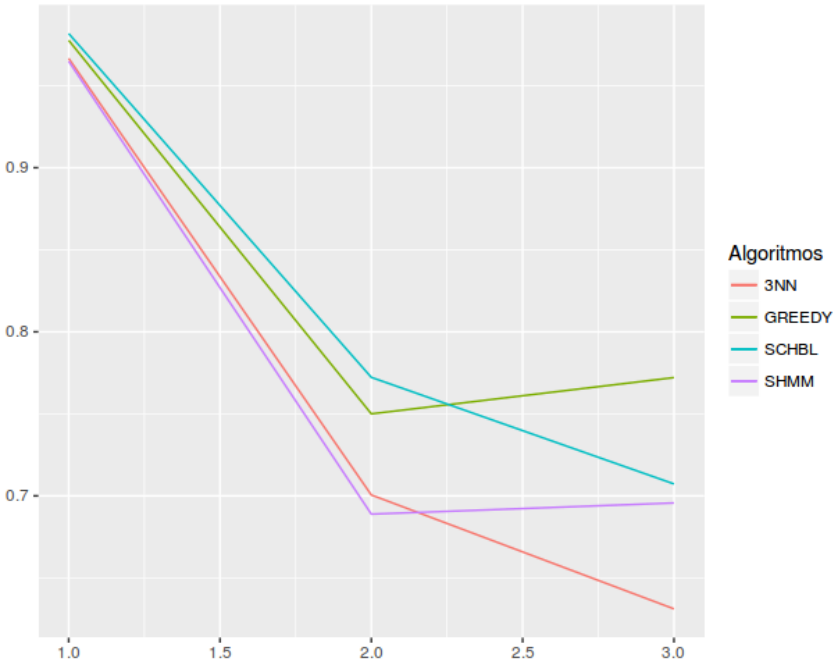

Figura 1: *Comparativa global de rendimiento*

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