Metaheurística

Práctica 1.b:

Técnicas de Búsqueda Local y Algoritmos Greedy para el Problema del Agrupamiento con Restricciones

Curso 2019-2020

Javier Gálvez Obispo
javiergalvez@correo.ugr.es
Grupo 2 Jueves 17:30 – 19:30

Índice

1. Descripción del problema

Nos encontramos ante un problema de agrupamiento o clustering, donde nuestro objetivo es clasificar los datos que nos dan de acuerdo a posibles caractarísticas comunes entre ellos.

Además, en nuestro caso nos encontramos con restricciones de instancia a la hora de agrupar los datos, es decir, dada un pareja de instancias del conjuto de datos puede, o no, haber una restricción entre ellos del tipo *Must-Link*, los dos elementos deben estar en el mismo cluster, o del tipo *Cannont-Link*, si las dos instancias deben encontrarse en clusters diferentes.

Vamos a tratar estas restricciones como restricciones débiles, lo que quiere decir que nuestro objetivo será minimizar el número de restricciones que no se cumplen en vez de encontrar una solución en la que el número de restricciones incuplidas sea cero.

Nuestro objetivo es resolver este problema dados tres conjutos de datos diferentes junto a dos listas de restricciones por conjuto de datos, una con un 10% de restricciones y otra con un 20%.

En esta práctica usaremos el algoritmo greedy K-medias restringido débil y la búsqueda local para resolver el problema planteado.

2. Elementos en común entre los algoritmos

Aunque vamos a implementar dos algoritmos distintos la diferencia se encuentra en la propia implementación de éstos. Es por ello, que tienen varios elementos en común que se describirán a continuación.

2.1. Lectura de datos

Ambos algoritmos leen los datos de la misma forma. El conjunto de datos se almacena en una lista en la que cada posición representa una instancia distinta de los datos. A su vez, cada instancia es otra lista que contiene los valores de cada carácterística para esa instancia.
Las restricciones se representan en una matriz. Dada una posición i, j, en la matriz habrá un 0, si no existe ninguna restricción entre las instacias, un 1, si la restricción es *Must-Link* o un -1, si la restricción es *Cannnot-Link*.

2.2. Cálculo de la función objetivo

La función objetivo se define como:

$$f = \vec{C} + (infeasibility * \lambda)$$

donde $\vec{C}$ es la desviación general de la partición C, *infeasibility* el número de restricción incumplidas y *lambda* es un parámetro de escalado.

Para calcular *lambda* tenemos que encontrar los puntos más alejados del conjuto de datos y dividir la distancia entre éstos por el número total de restricciones que hay en el problema.

El psudocódigo para calcular lambda sería el siguiente:

Input: Conjuto de datos X, conjuto de restricciones R.

max_dist, num_restricciones = 0
for i **in** {0, …, N-1} **do**:
  for j **in** {i+1, …, N} **do**:
    - Calcular distancia entre X_i y X_j
    if (distancia > max_dist) guardar nueva distancia
    if (R_{ij} == 1 **or** R_{ij} == -1) num_restricciones += 1
  end for
end for
return lambda = max_dist / num_restricciones

El calculo de la *infeasibility* se haría de la siguiente forma:

Input: Clusters C, conjunto de datos X, conjuto de restricciones R.

infeasibility = 0
for i **in** {0, …, N-2} **do**:
  for j **in** {i+1, …, N-1} **do**:
    if (R_{ij} == 1) **do:**
      - Si X_i y X_j no están en el mismo cluster C_k => infeasibility += 1
    elif (R_{ij} == -1) **do**:
      - Si X_i y X_j están en el mismo cluster C_k => infeasibility += 1
  end for
end for
return infeasibility

Y por último para calcular $\vec{C}$ primero tendremos que calcular la distancia intra-cluster y por tanto los centroides de cada cluster.

Input: Clusters C, conjunto de datos X.

for i **in** {0, …, k-1} **do**:
- Calcular centroide μ_i para cada C_i

end for

for i **in** {0, …, k-1} **do**:
- Calcular distancia media-intracluster $\bar{c}_i$ para cada C_i

end for

desviacion = $\frac{1}{k}\sum \bar{c}_i$

return desviacion

2.3. Representación de las soluciones

La estructura para representar la solución que nos dan los algoritmos es la misma. Las soluciones se escriben en un fichero con nombre "*dataset_porcentaje-de-restricciones_algoritmo_semilla.output"*, por ejemplo el nombre de un fichero de salida válido sería "iris_10_greedy_123.output". La estructura de estos archivos es la siguiente:

Clusters (cada algoritmo representa los cluster de una forma por lo que esta parte de la salida varía)

Tasa_C
Infeasibility
Valor de la función objetivo para nuestra solución
Veces que se llama a la función objetivo (sólo en el caso de la búsqueda local)
Tiempo

Tamaño de los clusters
Cluster : número de elementos

3. Pseudocódigo

3.1. Algoritmo greedy k-medias restringido débil

El pseudocódigo del algoritmo greedy K-medias restringido débil es el siguiente:

Input: Conjunto de datos X, conjuto de restricciones R, número de clusters k.

Generar k centroides aleatorios
rsi = shuffle({1, …, N})
n = tamaño de rsi
cambio = True

while cambio == True **do**:
  for i **in** rsi **do**:
    - Obtener la asignación/es de x_i a c_k que produce menor incremento de infeasibility
    if más de un mínimo **do**:
      - Calcular la mínima distancia de x_i a todos los centroide μ_j
      c_j = cluster más cercano
    end if
    - Añadir x_i al cluster c_j
    - Eliminar i de rsi
  end for
  - Recalcular centroides
  cambio = (tamaño de rsi != n)
  n = tamaño de rsi
end while
return clusters

3.2. Búsqueda local

El pseudocódigo de la búsqueda local quedaría así:

Input: Conjunto de datos X, conjunto de restricciones R, número de cluster k, número de veces que se puede llamar a la función objetivo max_iters.

do:
  S = lista con N números entre 0 y k-1
while algún cluster vacío

- Calcular f para la solución S

while n < max_iters **and** mejora **do**:
  mejora = False
  vecinos = lista con todas las parejas de vecinos (i, c) posibles
  shuffle(vecinos)
  for vecino (i, c) **in** vecinos **do**:
    new_S = S
    new_S[i] = c
    if ningún cluster vacío **do**:
      - Calcular f para la solución new_S
      n += 1
      if f de new_S > f de S **do:**
        - Actualizar S
        mejora = True
        break // Primero el mejor
      end if
    end if
  end for
end while
return S

---

4. Procedimiento considerado para desarrollar la práctica

Para realizar la práctica he utilizado python junto a numpy empezando desde cero. Para poder probar la prática se puede ejecutar el archivo main.py de la siguiente forma:

python3 src/main.py dataset porcentaje-de-restricciones numero-de-clusters gr/ls semilla

un ejemplo de uso sería *"python3 src/main.py iris 10 3 ls 11264"* que ejecutaría el algoritmo de búsqueda local para el dataset iris con un 10% de restricciones utilizando la semilla 11264.

Además, se puede hacer uso del makefile para facilitar las ejecuciones, las reglas definidas en este son las siguientes:

all	Realiza las 60 ejecuciones que se nos piden para esta práctica
gr	Realiza las 30 ejecuciones del algoritmo greedy
ls	Realiza las 30 ejecuciones de la búsqueda local
gr_*dataset*	Realiza las 10 ejecuciones del *dataset* utilizando el algoritmo greedy
ls_*dataset*	Realiza las 10 ejecuciones del *dataset* utilizando la búsqueda local
gr_*dataset*_10	Realiza las 5 ejecuciones del *dataset* con 10% de restricciones utilizando el algoritmo greedy
gr_*dataset*_20	Realiza las 5 ejecuciones del *dataset* con 20% de restricciones utilizando el algoritmo greedy
ls_*dataset*_10	Realiza las 5 ejecuciones del *dataset* con 10% de restricciones utilizando la búsqueda local
ls_*dataset*_20	Realiza las 5 ejecuciones del *dataset* con 20% de restricciones utilizando la búsqueda local

Sustituir *dataset* por iris, ecoli o rand.
Las semillas ya estan fijadas en el makefile, para ejecutar cualquier otra semilla hay que utilizar el método descrito anteriormente.

5. Experimentos y análisis de resultados

Los algoritmos van a ser probados utilizando tres datasets distintos iris, ecoli y rand. Para cada dataset tenemos dos casos distintos, primero con un 10% de restricciones y luego con un 20% de restricciones. Cada algoritmo se ejecutará cinco veces por cada pareja dataset-restricciones utilizando una semilla distinta por cada ejecución. Las semillas han sido generadas aleatoriamente y se utilizan las mismas cinco para todas los casos. Los resultados de las ejecuciones son los siguientes:

Greedy 10% restricciones												
	Iris				Ecoli				Rand			
Seed	Tasa_C	Infea	Agr	T	Tasa_C	Infea	Agr	T	Tasa_C	Infea	Agr	T
11264	0.8234	77	1.3118	0.120	41.2713	440	53.0763	1.1192	1.4432	112	2.2507	0.1259
16438	1.2483	215	2.6121	0.1199	38.0281	497	51.3624	1.0657	1.5587	167	2.7628	0.1284
75645	0.8459	174	1.9496	0.1187	38.8251	458	51.1130	1.0399	1.9359	192	3.3202	0.1253
79856	0.6693	0	0.6693	0.1235	40.1099	494	53.3637	1.0753	1.6420	181	2.9470	0.1305
96867	0.6954	22	0.8350	0.1199	40.3855	381	50.6076	1.0828	0.8430	18	0.9728	0.1239
Media	**0.8565**	**97.6**	**1.4756**	**0.1204**	**39.7240**	**454**	**51.9046**	**1.0766**	**1.4845**	**134**	**2.4507**	**0.1268**

BL 10% restricciones												
	Iris				Ecoli				Rand			
Seed	Tasa_C	Infea	Agr	T	Tasa_C	Infea	Agr	T	Tasa_C	Infea	Agr	T
11264	0.6693	0	0.6693	3.4520	21.9633	70	23.8414	143.2379	0.7573	0	0.7573	3.1748
16438	0.6693	0	0.6693	3.9706	22.0027	72	23.9344	215.8607	0.7573	0	0.7573	3.3952
75645	0.6693	0	0.6693	3.8957	21.9472	91	24.3887	158.5676	0.7573	0	0.7573	3.5758
79856	0.6693	0	0.6693	4.0491	21.3589	93	23.8541	160.8232	0.7573	0	0.7573	4.2563
96867	0.6693	0	0.6693	3.6736	21.9078	64	23.6249	142.7053	0.7573	0	0.7573	3.3039
Media	**0.6693**	**0**	**0.6693**	**3.8082**	**21.8360**	**78**	**23.9287**	**164.2389**	**0.7573**	**0**	**0.7573**	**3.5412**

Resultados globales 10% restricciones												
	Iris				Ecoli				Rand			
Algoritmo	Tasa_C	Infea	Agr	T	Tasa_C	Infea	Agr	T	Tasa_C	Infea	Agr	T
Greedy	0.8565	97.6	1.4756	0.1204	39.7240	454	51.9046	1.0766	1.4845	134	2.4507	0.1268
BL	0.6693	0	0.6693	3.8082	21.8360	78	23.9287	164.238	0.7573	0	0.7573	3.5412

Greedy 20% restricciones												
	Iris				Ecoli				Rand			
Seed	Tasa_C	Infea	Agr	T	Tasa_C	Infea	Agr	T	Tasa_C	Infea	Agr	T
11264	0.6781	17	0.7320	0.1522	38.6349	1041	52.5997	1.6680	0.9251	120	1.3575	0.1518
16438	0.8340	110	1.1827	0.1520	37.0113	492	43.6114	1.6889	0.8767	87	1.1902	0.1483
75645	0.7014	80	0.9550	0.1539	38.2837	342	42.8715	1.6157	0.8359	67	1.0774	0.1473
79856	0.6693	0	0.6693	0.1572	40.8580	515	47.7666	1.70	0.9073	121	1.3434	0.1519
96867	0.6693	0	0.6693	0.1520	36.7383	354	41.4871	1.6836	0.7755	19	0.8439	0.1484
Media	**0.7104**	**41.4**	**0.8417**	**0.1535**	**38.3052**	**548.8**	**45.6672**	**1.6712**	**0.8641**	**82.8**	**1.1625**	**0.1495**

BL 20% restricciones												
	Iris				Ecoli				Rand			
Seed	Tasa_C	Infea	Agr	T	Tasa_C	Infea	Agr	T	Tasa_C	Infea	Agr	T
11264	0.6693	0	0.6693	3.8112	21.7477	173	24.0685	147.1178	0.7573	0	0.7573	3.6745
16438	0.6693	0	0.6693	3.7954	21.7976	175	24.1451	145.8017	0.7573	0	0.7573	3.8983
75645	0.6693	0	0.6693	3.4877	21.7822	181	24.2103	161.7181	0.7573	0	0.7573	3.7972
79856	0.6693	0	0.6693	3.1634	22.2129	167	24.4532	163.1228	0.7573	0	0.7573	3.9377
96867	0.6693	0	0.6693	3.3182	22.4578	158	24.5773	137.0049	0.7573	0	0.7573	3.3381
Media	**0.6693**	**0**	**0.6693**	**3.5152**	**21.9996**	**170.8**	**24.2909**	**150.9531**	**0.7573**	**0**	**0.7573**	**3.7291**

Resultados globales 20% restricciones												
	Iris				Ecoli				Rand			
Algoritmo	Tasa_C	Infea	Agr	T	Tasa_C	Infea	Agr	T	Tasa_C	Infea	Agr	T
Greedy	0.7104	41.40	0.8417	0.1535	38.3052	548.8	45.6672	1.6712	0.8641	82.8	1.1625	0.1495
BL	0.6693	0	0.6693	3.5152	21.9996	170.8	24.2909	150.953	0.7573	0	0.7573	3.7291

Como podemos observar la búsqueda local obtiene mejores resultados que el algoritmo greedy a cambio de tardar mucho más tiempo por ejecución. Esto se debe a que la búsqueda local en cada iteración debe generar todo el entorno de la solución actual y encontrar un vecino mejor, es decir, generamos los N * (k – 1) vecinos posibles y los evaluamos. Cierto es que estamos utilizando un esquema primero el mejor pero, aun así, esto nos libra de evaluar una gran cantidad de vecinos al principio del algoritmo, cuando es muy fácil encontrar una solución vecina que mejore a la que tenemos. Cada vez que vamos a comprobar si el vecino es mejor tenemos que recalcular los centroides, ya que los clusters cambian, y recalcular la infeasibility. Cuando el algoritmo está cerca de encontrar una solución que ya no mejore tendrá que explorar casi todo el entorno, aumentando considerablemente el número de cálculos a realizar y por tanto el tiempo de ejecución del algoritmo.