Trabajo Práctico 2

Clustering

21 de octubre de 2018 — Algoritmos y Estructura de Datos III

Integrante	LU	Correo electrónico
Buceta, Diego	001/17	diegobuceta35@gmail.com
Springhart, Gonzalo	308/17	glspringhart@gmail.com

Instancia	Docente	Nota
Primera entrega		
Segunda entrega		

Facultad de Ciencias Exactas y Naturales
Universidad de Buenos Aires
Ciudad Universitaria - (Pabellón I/Planta Baja)
Intendente Güiraldes 2610 - C1428EGA
Ciudad Autónoma de Buenos Aires - Rep. Argentina
Tel/Fax: (++54 +11) 4576-3300
http://www.exactas.uba.ar

1. Introducción al problema

En este informe informe vamos a ver distintos problemas cuyas posibles soluciones involucran árboles generadores mínimos y algoritmos que los generan. En particular los problemas son el **Clustering** y el **Arbitraje**.

2. Clustering

En el mundo del machine learning, problema de Clustering consiste en poder agrupar datos en distintos grupos llamados clusters, donde se espera que los datos de un cluster tengan algún tipo de relación o característica que los distinga de los datos en otros clusters. Este problema es de gran interés ya que tiene muchos usos como por ejemplo (...). El problema de Clustering es un problema difícil, ya que no existe un algoritmo exacto que pueda generar clusters en tiempo polinomial a partir de un conjunto de datos, además el concepto de un cluster es ambiguo, dado un grafo no existe una única configuración de clusters que separe los datos, sino que hay varias configuraciones que a simple vista pueden parecer válidas.

Por lo tanto para resolver el problema utilizamos una **heuristica**, la misma esta basada en el método de detección de clustes explicado en el paper "Graph-Theoretical Methods for Detecting and Describing Gestalt Clusters"de Charles T. Zahn, y su forma de calcular los clusters es la siguiente: Consideramos nuestros datos como una serie de puntos en un plano, tomando un grafo completo G usando los puntos como vertices y las distancias entre ellos como pesos en las aristas. Luego calculamos el Árbol Generador Mínimo de G y lo llamamos T. Para poder encontrar los clusters hay que eliminar de T a los ejes **inconsistentes**, según el paper un eje e es inconsistente si su peso supera en cierta cantidad de desviaciones estandard a las medias de los pesos de las aristas que estén a distancia k de los vértices que están en sus puntas. Al remover un eje inconsistente, las componentes conexas formadas son identificadas con un indice, entonces luego de recorrer todas las aristas, todas las componentes conexas van a tener un indice distinto y podemos interpretarlas como los clusters del grafo. Para poder calcular el AGM del grafo G se implementaron dos algoritmos, el algoritmo de Kruskal y el algoritmo de Prim, ambos levemente modificados para utilizar estructuras de Lista de Incidencia y Lista de Adyacencia respectivamente. Además, se implentaron dos versiones de Kruskal, una con path-compression y otra sin, a fin de poder compararlos en la experimentación.

3. Justificación teórica

4. Algoritmos presentados

4.1. Kruskal

KruskalSinPathComp($ListaIncidencia : grafoCompleto, cantNodos : entero$)

padre ← vector de enteros de tamaño de cantNodos y cargado con el valor de su posición en cada posición
AGM ← lista de incidencia vacia, de tamaño cantNodos-1
OrdenarPorPeso(grafoCompleto)
for e:Arista ∈ grafoCompleto
if $getPadre(indice(e.primerNodo), padre) = getPadre(indice(e.segundoNodo), padre)$ **then**
    agregar(e,agm)

Devolver AGM

getPadre($entero : indice, padre : vectordeenteros$)

if $padre[indice] == indice$ **then**
    Devolver indice
  else
    getPadre(indice(padre[indice]),padre)

getPadreConPathComp($entero : indice, padre : vectordeenteros, altura : vectordeenteros, nivelesSubidos : entero$

```
if padre[indice] == indice then
        altura[indice] = nivelesSubidos
        Devolver indice
    else
        padre[indice] = getPadreConPathComp(indice(padre[indice]),padre,altura,nivelesSubidos+1)
        Devolver padre[indice]
```

unirPadres($indiceNodo1 : entero, indiceNodo2 : entero, padre : vectordeenteros$)

```
padreNodo1 ← getPadre(indiceNodo1, padre)
padre[indiceNodo2]=padreNodo1
```

unirPadresConPathComp($indiceNodo1 : entero, indiceNodo2 : entero, padre : vectordeenteros, altura : vectordeenteros, nivelesSubidos : entero$)

```
padreNodo1 ← getPadreConPathComp(indiceNodo1, padre,altura,0)
padreNodo2 ← getPadreConPathComp(indiceNodo2, padre,altura,0)
padreMenosAltura ← min(altura[padreNodo1],altura[padreNodo2])
padreMasAltura ← max(altura[padreNodo1],altura[padreNodo2])
padre[padreMenosAltura]=padreMasAltura
```

armarGrafoCompleto($nodos : vectordeNodos$)

```
listaAristas ← inicializar lista de incidencia
matrizAristas ← inicializar matriz de adyacencia
for i ← 0 to tam(nodos)
for j ← i + 1 to tam(nodos)
armar arista con datos de v[i] y v[j]
agregar arista a listaAristas
armar matriz de adyacencia con la lista de incidencia
Devolver Matriz de adyacencia y Lista de incidencia
```

retirarEjesInconsistentes($listaAristas : listaincidencia, \sigma_T, profVecindario, f_T, forma, cantidadDeCluster$
$vectordeenteros$)

```
for e : listaAristas
calcular media y desviacion respecto del vecindario de profVecindario de profundidad de cada extremo de e. (u
si es inconsistente
sacar e de las listas
recorrer en la lista de ady todos los nodos alcanzables de uno de los extremos y modificar su representante en
aumentar en 1 el valor de cantidadDeClusters
```

4.2. Prim

4.3. Complejidad

5. Experimentación

5.1. Variaciones

Los experimentos estarán centrados en analizar las diferentes tipos clusterizaciones que pueden realizarse variando las definiciones de eje inconsistente. Intentaremos analizar las configuraciones necesarias para que se pueda alcanzar una clusterización lo más cercana a la de la percepción humana y los resultados interesantes al que pueden llegarse.

Dados los siguientes: f_T multiplicador del promedio, σ_T multiplicador de la desviación, y la profundidad del vecindario de los extremos del eje candidato, W(XY) el peso del eje candidato, y sea X e Y sus nodos extremos, definiremos un eje inconsistente:

- Forma 1: $\frac{W(XY)}{Promedio(Vecindario(X))} > f_T$ y $\frac{W(XY)}{Promedio(Vecindario(Y))} > f_T$,
  Es decir, la proporción entre el peso del eje candidato y el promedio de peso del vecindario de sus extremos es mayor al coeficiente dado.
- Forma 2: $W(XY) > Promedio(Vecindario(X)) + \sigma_T * desviacion(Vecindario(X))$ y $W(XY) > Promedio(Vecindario(Y)) + \sigma_T * desviacion(Vecindario(Y))$,
  Es decir, que el peso del eje candidato supere al promedio del vecindario de sus extremos por al menos σ_T unidades de la desviación del vecindario del extremo.
- Forma 3: Que se cumpla ambas

6. Arbitraje

7. Justificación teórica

8. Algoritmos presentados

8.1. Complejidad

9. Experimentación

10. Comparaciones de algoritmos de los problemas resueltos

11. Bibliografía