Representación del Conocimiento

Práctica 4: Aprendizaje de parámetros y grafos

Arahí Fernández Monagas

Iker Martínez Gómez

Carlos Velázquez Fernández

Grado en Ingeniería
Informática
Mención en Computación
Curso 2023-2024

1. Algoritmo de Chow-Liu

El algoritmo de Chow-Liu nos permite aprender un grafo en base a los parámetros. El funcionamiento es el siguiente:

Algorithm 1 Algoritmo de Chow-Liu

Crear un grafo $\mathcal{K} = (V, E, W)$ tal que $w_{ij} = \mathbb{I}(X_i, X_j \mid \mathcal{D})$ para todo $\{v_i, v_j\} \in E$.	▷ Paso1
Buscar un árbol de recubrimiento de peso máximo $\mathcal{G} = (V, E')$ de $\mathcal{K}$.	▷ Paso 2
Elegir $v \in V$ a partir del que asignar direccionalidad a $\mathcal{G}$ hacia afuera.	▷ Paso 3

en donde

$$\mathbb{I}(X, Y|\mathcal{D}) = \sum_{x,y} p_{\mathcal{D}}(x, y) log \frac{p_{\mathcal{D}}(x, y)}{p_{\mathcal{D}}(x) \cdot p_{\mathcal{D}}(y)}$$

Nuestro algoritmo recibe como input el nombre del fichero en el que se encontrarán los datos y devuelve el árbol de Chow-Liu. Los pasos principales se explicarán a continuación.

1.1. Paso 1. Cálculo de la información mutua y grafo ponderado

Información mutua: Esta información es la que nos permitirá asignar pesos a las aristas del grafo. Para el cálculo de $\mathbb{I}$, primero debemos saber en qué medida aparece cada dato.

Dado un conteo del número de veces que aparece cada muestra (una combinación de variables con valores asignados) llevado al leer el fichero, nuestra función *probabilitiesFromCounting* extrae el número total de veces que aparece cada dato (variable y su valor) de forma individual y cada combinación de par datos distintos. Esta información se normaliza y se representa en tablas que nos ayudarán a ver las proporciones resultantes.

Árbol ponderado: La función *buildWeightedGraph* recibe estas tablas como argumento de entrada. Primero crea un grafo completo, siendo los nodos las distintas variables de los datos. Después recorre las aristas del grafo asignándolas pesos en función del valor de $\mathbb{I}(X, Y|\mathcal{D})$. Este valor es calculado siguiendo la fórmula mostrada previamente y extrayendo los datos necesarios de las tablas calculadas. Une vez finalizado, se devuelve el grafo resultante.

1.2. Paso 2. Árbol de recubrimiento de peso máximo

Para calcular el árbol de recubrimiento máximo se ha seguido el algoritmo de Kruskal, implementado en la función *maxSpanningTree*. En primer lugar se ordenan las aristas del grafo calculado en el paso anterior de mayor a menor peso. Después, se recorre esta lista añadiendo las aristas a un nuevo grafo con los mismos nodos pero sin aristas siguiendo este orden. Antes de añadir cada arista, es imprescindible comprobar que no se crearán ciclos, puesto que en tal caso dejaría de ser un árbol.

El resultado es un grafo conexo acíclico cuyas aristas maximizan el peso de dicho árbol.

1.3. Paso 3. Direccionalidad del árbol

La función *outwardDirectionality* recibe un árbol no dirigido y le aplica direccionalidad hacia afuera. El vértice raíz es elegido de forma totalmente aleatoria y, partiendo de este, se recorre el resto del grafo siguiendo el algoritmo de búsqueda en anchura asignando direccionalidad a las aristas.

El resultado de esta última función es el resultado del algoritmo, el árbol de Chow-Liu.

2. Pruebas

Para comprobar el correcto funcionamiento de nuestro algoritmo se ha implementado un programa de prueba que ejecuta tres ejemplos, variando el número de variables n y los valores que puede tomar cada variable k.

Para la generación de los datos se ha usado la función *generateData*. En cada test, se definen posibles valores para cada variable. Para cada combinación, se asigna el número de veces que debe aparecer en el fichero, es decir, el número veces que se recoge esa muestra (esto se hace para comprobar que los resultados obtenidos son los esperados, pero también se da la opción de que cada combinación aparezca un número aleatorio de veces). A partir de aquí, se genera un fichero con los datos.

Para comprobar que los resultados son los correctos, se han desarrollado a mano los tres primeros ejemplos propuestos. Aunque cabe destacar que otra tarea fundamental de estos test es poder evaluar de forma empírica el coste temporal, para lo que se usan otros dos ejemplos diferentes mucho más grandes.

3. Coste temporal

3.1. Coste temporal teórico

Para calcular la complejidad temporal del algoritmo de Chow-Liu repasaremos cual es la complejidad resultante de cada uno de los pasos con el objetivo de encontrar la que más limite al algoritmo. Consideramos n, e y k como el número de variables, aristas y valores máximos de cada variable, respectivamente. Debemos tener en cuenta que, dado que no conocemos las relaciones entre las variables, partimos de un grafo completo. Este grafo tendrá $\frac{n*(n-1)}{2}$ aristas, o en otras palabras, el número de aristas depende cuadráticamente del número de variables.

En el cálculo de la información mutua de dos variables influye tanto el número de aristas como el número de valores que puede tomar cada variable. Para cada arista, recorreremos la tabla de probabilidades correspondiente al par de nodos que conecta. El tamaño de esta tabla será de k^2 filas. Es decir, la complejidad de este paso será de $O(e\ k^2) = O(n^2\ k^2)$.

A la hora de construir el grafo ponderado se ha seguido el algoritmo de Kruskal, el cual tiene una complejidad de $O(e\ log\ e) = O(n^2\ log\ n)$.

Finalmente, para asignar la direccionalidad se ha realizado un recorrido en anchura del grafo, siendo la complejidad temporal $O(n\ +\ e)$, y si tenemos en cuenta que el número de aristas es $e = n - 1$, entonces la complejidad será $O(n)$.

Podemos concluir que la complejidad del algoritmo estará definida por los dos primeros pasos, siendo esta $O(n^2\ (k^2\ +\ log\ n) + 2 \cdot n) = (n^2\ (k^2\ +\ log\ n))$.

3.2. Coste temporal empírico

Para realizar el análisis temporal, usamos los ejemplos que teníamos en nuestro fichero de prueba, cada uno con propiedades diferentes, y así poderlo comparar posteriormente con el teórico. Para obtener un resultado más preciso, se ha ejecutado el algoritmo para cada caso 1.000 veces, y se ha calculado el tiempo promedio. A continuación se exponen los resultados:

Ejemplo	n	k	Tiempo de ejecución
1	a b c	$\{0, 1\}$	161,0387 μs
4	{a b c...j} n = 10	$\{0, 1\}$	$1735,1747\,\mu$s
5	{a b c...t} n= 20	$\{0, 1\}$	$6882,1518\,\mu$s

Cuadro 1: Tiempos medios de ejecución

Como podemos observar, de un ejemplo a otro se aumenta de manera considerable el número de variables, haciendo que aumente por tanto el coste temporal. En cada ejemplo se deben considerar más combinaciones de valores posibles para calcular la información mutua entre las variables, lo que implica un mayor coste computacional.

Para verificar si esto se acerca a nuestros cálculos teóricos, comprobaremos que los tiempos aumenta de la forma esperada siguiendo la complejidad expuesta anteriormente (incluyendo el coste de aplicar la direccionalidad, puesto que para valores pequeños puede ser relevante).

Ejemplo	$O(n^2\ (k^2\ +\ log\ n))$	Resultado	Aumento (con respecto al ejemplo 1)
1	$O(3^2(2^2 + log3))$	40,2941	x 1
4	$O(10^2(2^2 + log10))$	500	x 12,408
5	$O(20^2(2^2 + log20))$	2120,412	x 52,623

Cuadro 2: Slow downs teóricos

Ejemplo	Tiempo	Aumento (con respecto al ejemplo 1)
1	161,0387	x 1
4	1735,1747	x 10,7749
5	6882,1518	x 42,7360

Cuadro 3: Slow downs empíricos

Tras observar los resultados, podemos concluir que los tiempos de ejecución se aproximan a lo que esperábamos tras hacer predicciones mediante la complejidad teórica.