

**Pontificia Universidad Católica Madre y Maestra
(PUCMM)**



Nombre:

Máximo Rodríguez

Matrícula:

(2010-2172)

Nombres de la práctica:

T7 – Clúster

Profesor:

Juan R. Núñez P.

Introducción

Los métodos de agrupamiento o clustering constituyen un tipo de aprendizaje por descubrimiento muy similar a la inducción. En el aprendizaje inductivo, un programa aprende a clasificar objetos basándose en etiquetados proporcionados por un profesor (aprendizaje supervisado). En los métodos de agrupamiento no se suministran los datos etiquetados: el programa debe descubrir por sí mismo las clases naturales existentes.

Por ejemplo, el programa AUTOCLASS usa razonamiento bayesiano para, dado un conjunto de datos de entrenamiento, sugerir un conjunto de clases plausible. Este programa encontró nuevas clases significativas de estrellas a partir de sus datos del espectro infrarrojo, lo que puede considerarse ejemplo de descubrimiento por parte de una máquina (los hechos descubiertos eran desconocidos para los astrónomos).

Las funciones de densidad de probabilidad suelen tener una moda o un máximo en una región; es decir, las observaciones tienden a agruparse en torno a una región del espacio de patrones cercana a la moda. Las técnicas de agrupamiento analizan el conjunto de observaciones disponibles para determinar la tendencia de los patrones a agruparse. Estas técnicas permiten realizar una clasificación asignando cada observación a un agrupamiento [cluster], de forma que cada agrupamiento sea más o menos homogéneo y diferenciable de los demás. Los agrupamientos naturales obtenidos mediante una técnica de agrupamiento mediante similitud resultan muy útiles a la hora de construir clasificadores cuando no están bien definidas las clases (no existe un conocimiento suficiente de las clases en que se pueden distribuir las observaciones), cuando se desea analizar un gran conjunto de datos (“divide y vencerás”) o, simplemente, cuando existiendo un conocimiento completo de las clases se desea comprobar la validez del entrenamiento realizado y del conjunto de variables escogido

Procedimiento

En este problema de programación y el siguiente se te pide codificar el algoritmo de agrupamiento en cluster para calcular un agrupamiento de max-clustering. Se tiene un archivo de datos clustering1.txt . Este archivo describe una función de distancia (que es equivalente, un grafo completo con los costos de aristas). Tiene el siguiente formato:

[cantidad_de_nodos]

[arista 1 nodo 1] [arista 1 nodo 2] [arista 1 costo]

[arista 2 nodo 1] [arista 2 nodo 2] [arista 2 costo]

Hay una arista para cada elección y tenemos el número de nodos. Por ejemplo, la tercera línea del archivo es "1 3 5250", indica que la distancia entre los nodos 1 y 3 (de forma equivalente, el costo de arista (1,3)) es de 5250. Se puede suponer que las distancias son positivas, pero no se debe asumir que son diferentes.

Su tarea en este problema es ejecutar el algoritmo de agrupamiento sobre este conjunto de datos, en el que el número de clusters se establece en 4. ¿Cuál es la distancia máxima de un cluster de 4-?

CONSEJO: Si usted no está recibiendo la respuesta correcta, intente depurar el algoritmo utilizando algunos casos de prueba pequeños.

Podemos decir que dado un conjunto de aristas ordenadas decrecientemente y dado un set de nodos se puede computar un max-spacing con el número de clusters deseados. Este algoritmo está basado Kruskal's minimum spanning tree algorithm.

Código

```
def cluster(num_clusters, desired_num_clusters, sorted_edges, nodes_set):  
    while num_clusters > desired_num_clusters:  
        node_a, node_b, edge_dist = sorted_edges.pop(0)  
        while same_root(node_a, node_b, nodes_set):  
            node_a, node_b, edge_dist = sorted_edges.pop(0)  
        union(node_a, node_b, nodes_set)  
        num_clusters -= 1  
    node_a, node_b, edge_dist = sorted_edges.pop(0)  
    while same_root(node_a, node_b, nodes_set):  
        node_a, node_b, edge_dist = sorted_edges.pop(0)  
    return node_a, node_b, edge_dist
```

```
def same_root(node_a, node_b, nodes_set):  
    a_root = find_set(node_a, nodes_set)  
    b_root = find_set(node_b, nodes_set)  
    return a_root == b_root
```

```
def union(node_a, node_b, nodes_set):  
    a_root = find_set(node_a, nodes_set)  
    b_root = find_set(node_b, nodes_set)  
    link(a_root, b_root, nodes_set)
```

```
def link(node_x, node_y, nodes_set):  
    if nodes_set[node_x][1] > nodes_set[node_y][1]:
```

```

        nodes_set[node_y][0] = node_x
    else:
        nodes_set[node_x][0] = node_y
        if nodes_set[node_x][1] == nodes_set[node_y][1]:
            nodes_set[node_y][1] += 1

def find_set(node, nodes_set):
    if node != nodes_set[node][0]:
        nodes_set[node][0] = find_set(nodes_set[node][0], nodes_set)
    return nodes_set[node][0]

def make_nodes_set(num_nodes):
    return dict([[i, [i, 0]] for i in range(1, num_nodes + 1)])

def main():
    edges = []
    #with open('test.txt') as file_in:
    with open('clustering1.txt') as file_in:
        next(file_in)
        for line in file_in:
            edges.append(map(int, line.strip().split(' ')))
    sorted_edges = sorted(edges, key = lambda edge: edge[-1])
    #num_clusters = int(open('test.txt').readline().strip())
    num_clusters = int(open('clustering1.txt').readline().strip())
    desired_num_clusters = 4

```

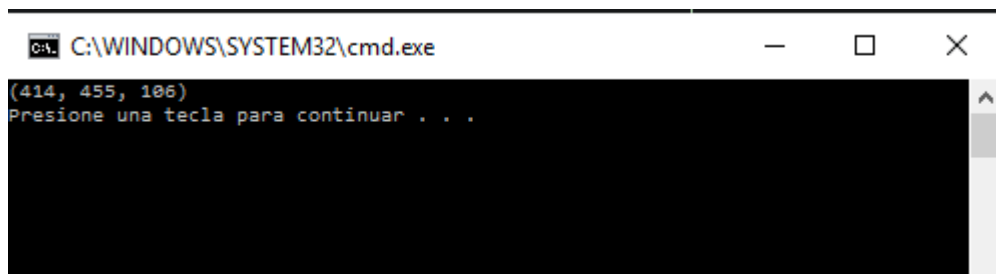
```
nodes_set = make_nodes_set(num_clusters)

return cluster(num_clusters, desired_num_clusters, sorted_edges, nodes_set)
```

```
if __name__ == '__main__':
    print main()
```

Salida

Max clustering de este caso de prueba es el siguiente mostrado.



```
C:\WINDOWS\SYSTEM32\cmd.exe
(414, 455, 106)
Presione una tecla para continuar . . .
```

En este problema, su tarea es nueva vez ejecutar el algoritmo de agrupamiento en clusters, pero en un grafo mucho más grande. Tan grande, de hecho, que las distancias (es decir, costos de aristas) sólo se definen implícitamente, en lugar de ser proporcionados como una lista explícita. El conjunto de datos es `clustering2.txt` . El formato es:

[# de nodos] [# de bits por cada etiqueta de nodo]

[primer bit de nodo 1] ... [ultimo bit de nodo 1]

[primer bit de nodo 2] ... [ultimo bit de nodo 2]

Por ejemplo, la tercera línea del archivo "0 1 1 0 0 1 1 0 0 1 0 1 1 1 1 1 0 1 0 1 1 0 1" se refiere a los 24 bits asociados con el nodo #2 .

La distancia entre dos nodos y en este problema se define como la distancia de Hamming --- el número de bits de diferentes --- entre las etiquetas los dos nodos. Por ejemplo, la distancia de Hamming entre la etiqueta de 24 bits del nodo # 2 arriba y la etiqueta "0 1 0 0 0 1 0 0 0 1 0 1 1 1 1 1 0 1 0 0 1 0 1" es 3 (ya difieren en los bits 3, 7, 21).

La pregunta es: ¿cuál es el valor más grande de tal manera que hay un -clustering con una separación de al menos 3? Es decir, el número de clusters necesarios para asegurar que ningún par de nodos con todos sino de 2 bits en común en se dividan en diferentes grupos?

NOTA: El grafo implícitamente definido por el archivo de datos es tan grande que es probable que no se puede escribir explícitamente, y mucho menos ordenar las aristas por el costo. Por lo que tendrá que ser un poco creativo para completar esta parte de la pregunta. Por ejemplo, ¿hay alguna manera que usted pueda identificar las distancias más pequeñas sin mirar de manera explícita en cada par de nodos?

Codigo

```
def cluster(num_clusters, nodes_set):
    all_nodes = set(nodes_set.keys())
    for node in all_nodes:
        possible_nodes = two_bit_flip(node)
        actual_nodes = possible_nodes.intersection(all_nodes)
        for other_node in actual_nodes:
            if not same_root(node, other_node, nodes_set):
                union(node, other_node, nodes_set)
                num_clusters -= 1
    return num_clusters

def two_bit_flip(node):
    node_list = list(node)
    out = set()
    bit_length = len(node_list)
    for i in range(bit_length):
        for j in range(bit_length):
            new_node = node_list[:]
            if i != j:
                new_node[i] = ('1' if node[i] == '0' else '0')
                new_node[j] = ('1' if node[j] == '0' else '0')
            else:
                new_node[i] = ('1' if node[i] == '0' else '0')
            out.add("".join(new_node))
    return out
```



```
def same_root(node_a, node_b, nodes_set):
```

```
    a_root = find_set(node_a, nodes_set)
```

```
    b_root = find_set(node_b, nodes_set)
```

```
    return a_root == b_root
```

```
def union(node_a, node_b, nodes_set):
```

```
    a_root = find_set(node_a, nodes_set)
```

```
    b_root = find_set(node_b, nodes_set)
```

```
    link(a_root, b_root, nodes_set)
```

```
def link(node_x, node_y, nodes_set):
```

```
    if nodes_set[node_x][1] > nodes_set[node_y][1]:
```

```
        nodes_set[node_y][0] = node_x
```

```
    else:
```

```
        nodes_set[node_x][0] = node_y
```

```
    if nodes_set[node_x][1] == nodes_set[node_y][1]:
```

```
        nodes_set[node_y][1] += 1
```

```
def find_set(node, nodes_set):
```

```
    if node != nodes_set[node][0]:
```

```
        nodes_set[node][0] = find_set(nodes_set[node][0], nodes_set)
```

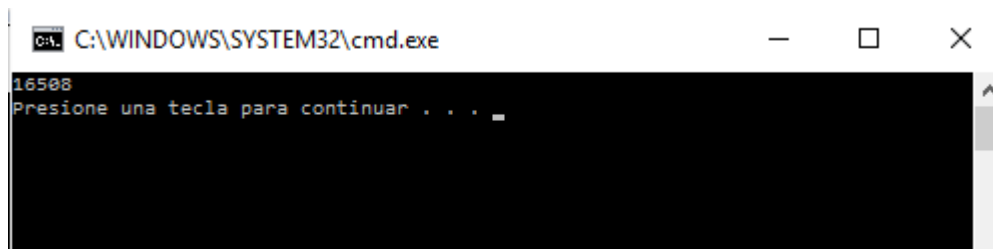
```
    return nodes_set[node][0]
```

```
def main():
```

```
nodes_set = {}  
with open('clustering2.txt') as file_in:  
#with open('clustering_big.txt') as file_in:  
    next(file_in)  
    for line in file_in:  
        node = ".join(line.strip().split(' '))  
        nodes_set[node] = [node, 0]  
#node ids in input file have duplicates  
num_clusters = len(nodes_set)  
return cluster(num_clusters, nodes_set)  
  
if __name__ == '__main__':  
    print main()
```

Salida

El valor más grande es:



A screenshot of a Windows command prompt window. The title bar shows the path 'C:\WINDOWS\SYSTEM32\cmd.exe'. The command prompt displays the memory address '16508' and the text 'Presione una tecla para continuar . . . _' followed by a cursor. The window has standard Windows window controls (minimize, maximize, close) in the top right corner.

Conclusión

Un algoritmo de agrupamiento o clustering es un procedimiento de agrupación de una serie de vectores de acuerdo con un criterio. Esos criterios son por lo general distancia o similitud. La cercanía se define en términos de una determinada función de distancia, como la euclídea, aunque existen otras más robustas o que permiten extenderla a variables discretas. La medida más utilizada para medir la similitud entre los casos es la matriz de correlación entre los $n \times n$ casos. Sin embargo, también existen muchos algoritmos que se basan en la maximización de una propiedad estadística llamada verosimilitud.

Generalmente, los vectores de un mismo grupo (o clústers) comparten propiedades comunes. El conocimiento de los grupos puede permitir una descripción sintética de un conjunto de datos multidimensional complejo. De ahí su uso en minería de datos. Esta descripción sintética se consigue sustituyendo la descripción de todos los elementos de un grupo por la de un representante característico del mismo.

Para resolver estos conjuntos de problemas se aplicara el algoritmo de Kruskal, el cual escoge las aristas mínimas para conectar cada vértice y finalmente al final generar minimum spanning tree, pero en el procedo de escoger las aristas se necesita estar seguro de que no haya ciclos, y para eso se utiliza unión-find. En este problema se termina el proceso de conectar los vértices tan rápido k cluster estén formados.

Bibliografía

<https://web.stanford.edu/class/cs345a/slides/12-clustering.pdf>

https://es.wikipedia.org/wiki/Algoritmo_de_agrupamiento

<http://www.sc.ehu.es/jiwdocoj/remis/docs/GarreAdis05.pdf>

<http://elvex.ugr.es/doc/proyecto/cap8.pdf>