**بسمه تعالی**

گروه‏های که مقالات یافتن اجتماعات را پیاده‏سازی می‏نمایند:

برای بررسی صحت خروجی الگوریتم‏ها، از متریکی به نام NMI استفاده می‏نماییم که میزان شباهت اجتماعات یافته شده و اجتماعات واقعی را مشخص می‏نمایید. مقدار این متریک بین یک تا صفر است که عدد یک نشان دهنده‏ی حداکثر شباهت می‏باشد. این متریک بصورت یک تابع تعریف شده است که vector از اجتماعات یافته شده توسط شما را با اجتماعات واقعی شبکه مقایسه می‏نماید.

تست کیس‎ها در قالب دو فایل متنی با نام‏های network.txt و community.txt ارایه شده‏اند. فایل اول شامل گراف به فرمت CSV است و فایل دوم شامل اجتماعات حقیقی هر راس می‏باشد. توجه شود که فایل‏ها را تغییر ندهید.

ورودی تابع NMI یک برداری است که هر عنصر آن برچسپ مربوط به یک یال را نمایش می‏دهد. عنصر صفر این بردار باید صفر باشد و عناصر بعدی بیانگر برچسپ هر راس باشد. بعنوان مثال اندیس 10 باید بیانگر برچسپ راس 10 باشد. ورودی دوم این تابع محل فایل متنی community.txtی است که اجتماعات آن شبکه را محاسبه نموده‏اید. توجه شود که برچسپ‏های شما باید اعداد طبیعی باشند.

**توجه:**

NMI به اسم برچسپ‏های حساس نیست. یعنی اگر اجتماعی در فایل اصلی برچسپ 6 را دارد ولی الگوریتم شما همان اجتماع را برچسپ 5 زده باشد، باز هم تابع NMI می‏تواند مشابهت بین این دو اجتماع را تشخیص دهد.

همچنین برای شما یک تابع طراحی نموده ایم که یک بردار را گرفته و آن بردار را shuffle می‏کند.

import java.io.BufferedReader;  
import java.io.FileReader;  
import java.util.Collection;  
import java.util.Hashtable;  
import java.util.Iterator;  
import java.util.Vector;

public static float NMI(Vector<Integer> Prediction, String TrueCommunityPathTXT )throws Exception {  
 Vector<Integer> TrueLabel = new Vector<Integer>();  
 Prediction.clear();  
 Prediction.add(1);  
 int countGuess = 0, countGold = 0;  
 float NMI = 0, up = 0, down = 0;  
 int n = 0;  
 TrueLabel.add(0);  
 BufferedReader br = new BufferedReader(new FileReader(TrueCommunityPathTXT));  
 String line = br.readLine();  
 while (line != null) {  
 String[] parts = line.split(" ");  
 int node = Integer.*parseInt*(parts[0]);  
 int label = Integer.*parseInt*(parts[1]);  
 TrueLabel.add(label);  
 Prediction.add(label \* 2);  
 n++;  
 line = br.readLine();  
 }  
 br.close();  
 if (n != Prediction.size() - 1)  
 return -1;  
 Hashtable<Integer, Integer> temp = new Hashtable<Integer, Integer>();  
 int k = 1;  
 for (int i = 1; i <= n; i++) {  
 if (temp.containsKey((Integer) Prediction.get(i)))  
 Prediction.set(i, temp.get(Prediction.get(i)));  
 else {  
 temp.put(Prediction.get(i), k);  
 Prediction.set(i, temp.get(Prediction.get(i)));  
 k++;  
 }  
 }  
 for (int i = 1; i <= n; i++) {  
 if (Prediction.get(i) > countGuess)  
 countGuess = Prediction.get(i);  
 if (TrueLabel.get(i) > countGold)  
 countGold = TrueLabel.get(i);  
 }  
 float NRow[] = new float[countGold];  
 float NCol[] = new float[countGuess];  
 float matrix[][] = new float[countGold][countGuess];  
 for (int i = 0; i < countGold; i++)  
 matrix[i] = new float[countGuess];  
 for (int i = 1; i <= n; i++) {  
 matrix[TrueLabel.get(i) - 1][Prediction.get(i) - 1]++;  
 NRow[TrueLabel.get(i) - 1]++;  
 NCol[Prediction.get(i) - 1]++;  
 }  
 for (int i = 0; i < countGold; i++)  
 if (NRow[i] != 0)  
 down += NRow[i] \* *log2*(NRow[i] / (float) (n));  
 for (int i = 0; i < countGuess; i++)  
 if (NCol[i] != 0)  
 down += NCol[i] \* *log2*(NCol[i] / (float) (n));  
 for (int i = 0; i < countGold; i++)  
 for (int j = 0; j < countGuess; j++)  
 if (matrix[i][j] != 0)  
 up += matrix[i][j] \* *log2*((matrix[i][j] \* (float) (n) / ((NRow[i] \* NCol[j]))));  
 up \*= (float) -2;  
 NMI = up / down;  
 return NMI;  
 }

public static void random\_shuffle(Vector<Integer> input) {  
 for (int i = 0; i < input.size(); i++) {  
 int random = i + (int) (Math.*random*() \* ((input.size() - 1 - i) + 1));  
 int temp = input.get(random);  
 input.set(random, input.get(i));  
 input.set(i, temp);  
 }  
}