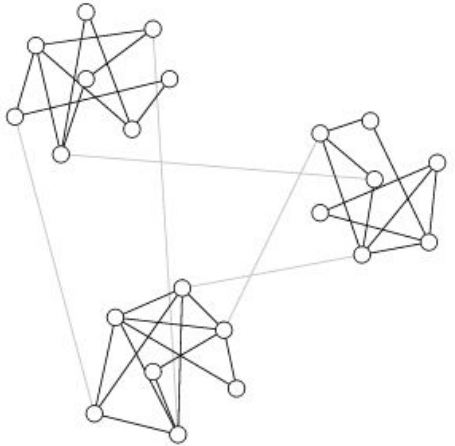
Meeting No.01

Reporter: 常博愛
丁世傑
薛翔宇

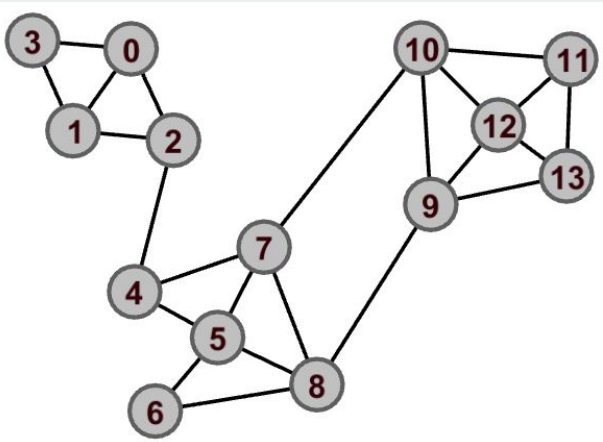
一.Modularity



为了度量实际网络中社区发现方法的好坏，Newman[3]于2004年提出了一个用于测度复杂网络中社区划分质量指标——模块度(Modularity)。在这之后，许多研究者以模块度函数作为目标函数，提出了一系列基于模块度最优的社区发现算法。

Modularity是 Community Detection（社区发现/社团检测） 中用来衡量社区划分质量的一种方法。要理解Modularity，我们先来看社团和社团检测的概念。

Community Detection



社团检测，就是要在一个图（包含顶点和边）上发现社团结构，也就是要把图中的结点进行聚类，构成一个个的社团。关于社团（community），目前还没有确切的定义，一般认为社团内部的点之间的连接相对稠密，而不同社团的点之间的连接相对稀疏。

在计算机中，也可能以邻接矩阵的方式存储这张图：邻接矩A

A_{ij}=1表示 结点i 和 结点j 之间有一条边，

A_{ij}=0表示 结点i 和 结点j 之间没有边。

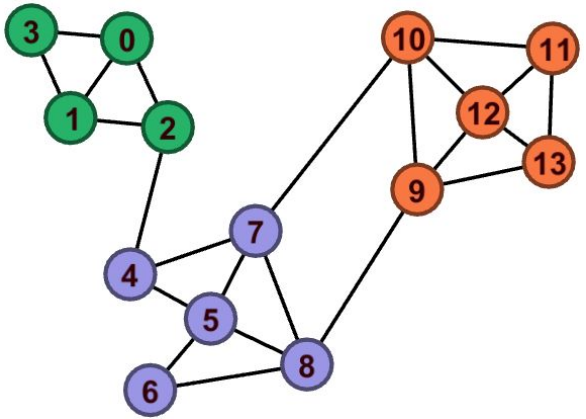
特别的，这里结点自己和自己的连接是0，即A_{ii}=0,不考虑结点自己和自己的连边.

Community Detection

输入图后，社团检测算法会输出一种社团划分，例如下面这个样子：

具体的输出文件可能这样：

```
0 1 2 3
4 5 6 7 8
9 10 11 12 13
```



Modularity

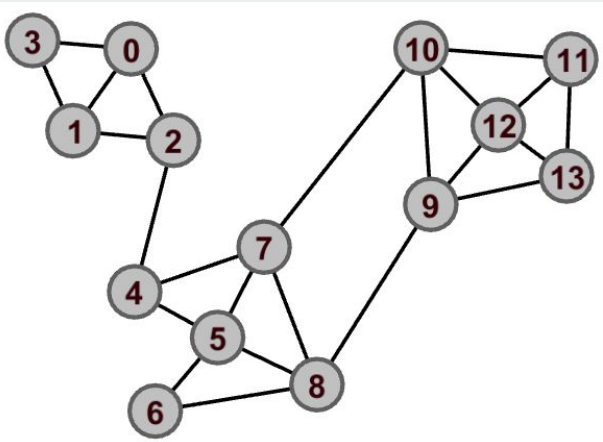
Modularity第一版：

$$Q==\sum_{i}(e_{ii}-a_i^2)=Tr(e)-//\ e^2\ //$$

Modularity第二版：

$$Q=\frac{1}{2m}\sum_{ij}(A_{ij}-\frac{k_ik_j}{2m})\delta(i,j)$$

Modularity--公式中用到的字母



和图相关的

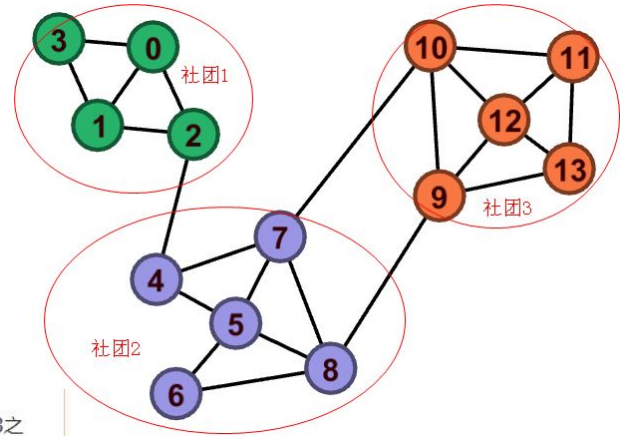
用 **n** 表示图的结点个数，这里n=14（结点编号0到13）

用 **m** 表示图中边的个数，这里m=23（上图中共23条边）

用 **k** 表示图中结点的度（degree），无向图中一个结点的度就是该结点连出去的边的条数。显然图中14个结点都有自己的度，我们用k_i来表示结点i的度。例如：$k_0 = 3, k_5 = 4, k_6 = 2$。

用 **A** 表示图的邻接矩阵。邻接矩阵A，$A_{ij} = 1$表示 结点v 和 结点w 之间有一条边，$A_{ij} = 0$表示 结点i 和 结点j 之间没有边。无向图的邻接矩阵是对称的，即$A_{ij} = A_{ji}$。特别的，这里结点自己和自己的连接是0，即$A_{ii} = 0$，不考虑结点自己和自己的连边。

模块度的定义与发展--原始定义(Q1)



假设社团划分把一个网络划分为k个社团，定义一个k*k的矩阵e。
e_{ij}表示连接 社团i 和 社团j 的边的数目 占 总边数 的比例。

e_{ii}表示的是社团i和社团i之间的边占总边数的比例，也就是社团i内部的边占总边数的比例。

我们这个图中，社团1内部有5条边，社团2内部有7条边，社团3内部有8条边。社团1和社团2之间有1条边，社团1和社团3之间有0条边，社团2和社团3之间有2条边。可以得到e矩阵如下：

	1	2	3
1	$\frac{5}{23}$	$\frac{1}{2}\times\frac{1}{23}$	$\frac{1}{2}\times\frac{0}{23}$
2	$\frac{1}{2}\times\frac{1}{23}$	$\frac{7}{23}$	$\frac{1}{2}\times\frac{2}{23}$
3	$\frac{1}{2}\times\frac{0}{23}$	$\frac{1}{2}\times\frac{2}{23}$	$\frac{8}{23}$

这样，矩阵e的迹 $tr(e)=\sum_i e_{ii}$，也就是矩阵对角元素的和，就表示了社团内部的边的比例。这个值越大，代表社团内部联系越紧密。然而这样有一个缺陷，如果把整张图分成1个社团，这个值就是最大值1 。

模块度的定义与发展--原始定义(Q1)

Trace:

对于一个给定的网络$G=(V,E)$，其中V为网络的节点集，E为网络的边集。将G划分为q个社区，我们用一个$q \times q$的对称矩阵$\mathbf{e}$来表示该划分，$\mathbf{e}$中的每个元素表示连接社区i与社区j的边在G的全部边中所占的比例，显然有$\sum_{i,j} e_{ii} = 1$。矩阵$\mathbf{e}$的迹$Tr(\mathbf{e})$表示连接社区内部节点的边的占比，一个好的社区划分应该有一个数值很高的迹。

但研究发现，单独使用迹$Tr(\mathbf{e})$在一些情况下无法很好地测度划分质量，例如这样一个平凡划分：将所有点放到同一个社区中。此时$Tr(e)$=1为最大值，但该划分在绝大部分情况下显然不具有任何意义。

“So we further define the row (or column) sums $a_i = \sum_j e_{ij}$, which represent the fraction of edges that connect to vertices in community i.”

模块度的定义与发展--原始定义(Q1)

因为上面e矩阵中，我们对不同社团之间的边要除以2，因此这里连接到 社团i 中的边是有不同权重的，完全在社团i中的边权重为1，而只有一端在社团i中的边权重就是$\frac{1}{2}$。

由于权重的不同，如果把一条边看做有两个端点，准确的说法是

$a_i = \sum_j e_{ij}$，它表示连接到 社团i 中的边的端点数（相当于社团中所有点的度相加） 占总端点数（2m） 的比例 ，即

$$a_i = \frac{k_{C_i}}{2m}$$

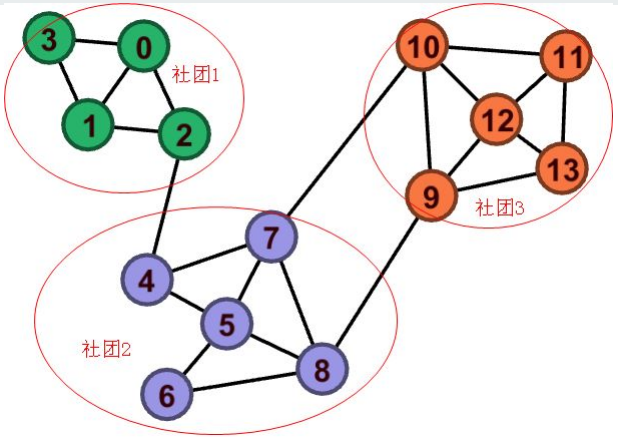
k_{C_i}表示社团i内部所有点的度数之和。

模块度的定义与发展--原始定义(Q1)

例子里我们的a如下：

a	
1	$\frac{5}{23}+\frac{1}{2}\times\frac{1}{23}+\frac{1}{2}\times\frac{0}{23}=\frac{11}{46}$
2	$\boxed{\frac{1}{2}\times\frac{1}{23}+\frac{7}{23}+\frac{1}{2}\times\frac{2}{23}}=\frac{17}{46}$
3	$\frac{1}{2}\times\frac{0}{23}+\frac{1}{2}\times\frac{2}{23}+\frac{8}{23}=\frac{18}{46}$

模块度的定义与发展--模块度(Q1)



Modularity第一版:

Newman[3]认为，考虑一个与G具有相同社区划分、相同边数但节点间随机连接的随机网络G'，若G具有社区结构，社区内部边占总边数的比例应该大于随机情况下的该比例的期望。实际社区内部边与期望的差越大，说明网络与随机网络的差异越大，即网络越具有社区结构。则模块度定义为

$$Q == Tr(\boldsymbol{e}) - \|\boldsymbol{e}\|^2 \qquad (1)$$

理论上有$Q \in [-1, 1]$，但取值为± 1的情况是极为罕见的。G的一个社区划分的模块度为正数时，表明社区结构可能存在。在该定义下，平凡划分(将所有点放到同一个社区中)的模块度为$Q = 0$。

模块度的定义与发展--考虑节点度的模块度(Q2)

让A_{ij}表示 结点i 和 结点j 之间边的数目， 一般无权图中A_{ij} 的取值是0或者1。但可以扩展到两点之间多条边的情形。

这里有2点没提到的：

1. 无向图的邻接矩阵是对称的，即

$A_{ij}=A_{ji}$。因此，如果把矩阵A的所有元素相加，得到的值是图中边的数目的2倍：$2m=\sum Aij$。

2. 自己和自己的连接在矩阵中是0，即 $A_{ii}=0$。

模块度的定义与发展--考虑节点度的模块度(Q2)

Modularity第二版:

上述定义是简单且直接的，但存在一个问题：在构建随机网络G'时，没有考虑到原网络G的节点度(degree)。由于节点的度在一定程度上能够表示该节点被连接的概率，所以Newman[4]于2006年对模块度进行了重新定义。

$$Q = \frac{1}{2}\sum_{i,j}\left(A_{ij} - \frac{k_i k_j}{2m}\right)\delta_{g_i g_j}$$

$$Q = \frac{1}{2m}\sum_{ij}(A_{ij} - \frac{k_i k_j}{2m})\delta(i,j)$$

Newman指出，在实践中，好的划分的Q值应当取值在0.3~0.7之间，更高的数值是很罕见的。这一重新定义的模块度也是目前使用范围最广的模块度。

模块度的定义与发展--考虑节点度的模块度(Q2)

对于随机网络G'，保持原网络G的边数m与各节点的度不变，将这m条边随机地重新定位到任意两个节点间。随机化后，令P_{ij}表示节点i与j相连接的概率，则社区内部边数的期望值为$\frac{1}{2}\sum_{i,j}P_{ij}\delta_{g_ig_j}$。故重新定义的模块度为：

$$
\begin{aligned}
Q &= \frac{1}{m}\left(\frac{1}{2}\sum_{i,j}A_{ij}\delta_{g_ig_j}-\frac{1}{2}\sum_{i,j}P_{ij}\delta_{g_ig_j}\right) \\
&= \frac{1}{2m}\sum_{i,j}(A_{ij}-P_{ij})\delta_{g_ig_j} \\
Q &\in [-0.5,1]
\end{aligned}
\tag{4}
$$

模块度的定义与发展--考虑节点度的模块度(Q2)

对于一个特定的网络，模块度的大小与m不相关，仅与社区划分有关。这一定义使用边数的比例而不是边数的绝对值，这使得不同规模的网络划分的模块度可以进行比较。

注意到，此时平凡划分的模块度为：

$$Q = \frac{1}{2m}\sum_{i,j}(A_{ij} - P_{ij}) = 0 \qquad (5)$$

因为在随机化中，网络中边的数量被视为常数，即$\sum_{ij} P_{ij} = \sum_{ij} A_{ij} = 2m$。

现在需要确定如何计算P_{ij}。这个值依赖于我们用来做随机化的方案。一个简单的方案是，对于任意两个节点i和j，令它们具有相同的连接概率，即对于给定边数m、节点数n的网络，有：

$$P_{ij} = \frac{m}{C_n^2} \qquad (6)$$

模块度的定义与发展--考虑节点度的模块度(Q2)

理论上说，这里P_{ij}应该是节点i与节点j之间的边数的期望而不是i与j相连接的概率，但由于$m << C_n^2$，故期望与概率很接近。

然而实际上，节点间的连接很大程度上依赖于节点的度，但如(6)式表示的P_{ij}没有考虑到节点的度。因此，我们需要一个能够保持节点度数的带约束随机化方案。这里，Newman使用了configuration model产生随机图集合[5],[6]。随机化后，两个节点之间的连接概率为

$$P_{ij} = \frac{k_i k_j}{2m} \qquad (7)$$

其中$k_i = \sum_j A_{ij}$是节点i的度。

至此，我们得到了重新定义的模块度：

$$Q = \frac{1}{2} \sum_{i,j} \left(A_{ij} - \frac{k_i k_j}{2m} \right) \delta_{g_i g_j} \qquad (8)$$

Newman指出，在实践中，好的划分的Q值应当取值在0.3~0.7之间，更高的数值是很罕见的。这一重新定义的模块度也是目前使用范 围最广的模块度。

Modularity Maximization--Greedy Algorithm

Greedy algorithm maximizes modularity at each step [2]:

1. At the beginning, each node belongs to a different community;

2. The pair of nodes/communities that, joined, increase modularity the most, become part of the same community. Modularity is calculated for the full network;

3. Step 2 is executed until one community remains;

4. Network partition with the higher modularity is chosen.

In terms of computational complexity, since modularity variation can be calculated in constant time, step 2 requires *O(L)* calculations. After merging communities, the adjacency matrix of the network is updated in a worst-case scenario of *O(N)*. Each merging event is repeated *N-1* times. Thus, the overall complexity is: *O((L+N)N)* or *O(N^2)*, in a sparse graph.

Modularity Maximization--Resolution limit

The resolution limit depends on the size of the network. One way to overcome this limitation is by subdividing larger communities into smaller ones and partition them.

根据Fortunato&Barthelemy[7]的研究成果，在对(8)式进行最优化来寻找社团划分时有一个缺点：无法找出那些实际具有许多小社团的网络中的社团划分。特别地，当网络中的社团数大于√2m时，通过使(8)最大化找到的社团划分与正确划分可能不符——基于(8)的模块度最优化方法寻找到的社区划分倾向于将一个个小社区合并成更大的社区，这样使得基于模块度最优的方法不能发现网络中较小的社区。

Modularity Maximization--Resolution limit

First, a way to calculate the network's modularity variation when communities A and B are merged is introduced [2]:

$$\Delta M_{AB} = \frac{l_{AB}}{L} - \frac{k_A k_B}{2L^2} \tag{7}$$

This means two communities should be joined whenever:

$$\Delta M_{AB} > 0 \Leftrightarrow \frac{l_{AB}}{L} > \frac{k_A k_B}{2L^2} \tag{8}$$

Modularity Maximization--Resolution limit

Assuming there is one link between communities A and B:

$$1 > \frac{k_A k_B}{2L} \tag{9}$$

In other words, when communities A and B are connected even only by one link, they will be merged if their size is lower than a threshold. Becoming impossible to detect communities below a certain size. Assuming,

$$k_A \approx k_B = k \tag{10}$$

$$1 > \frac{k^2}{2L} \Leftrightarrow k < \sqrt{2L} \tag{11}$$

Modularity Maximization--Modularity maxima

The fourth hypothesis presented at the beginning of the article, relies on the assumption that higher modularity implies a better partition of the network. Although, in some graphs, significantly different partitions may have similar modularity. This becomes a relevant issue when the number of nodes in the network increases. Becoming harder to separate the network's best partition from the lower quality ones. In the case of the algorithms that optimize modularity, this is a central issue, once they iterate until its variation is lower than an input threshold.

Modularity Maximization--Modularity maxima

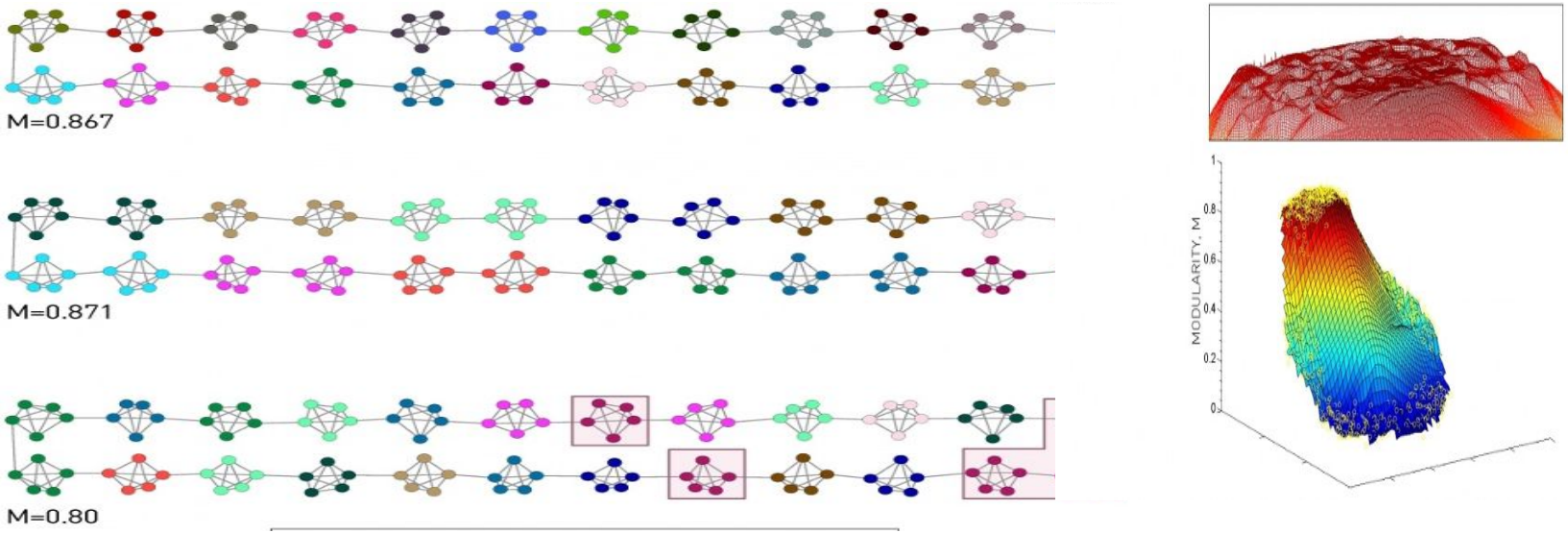


Figure 1 Significantly different partitions of the same network can have similar modularity [2]

Modularity Maximization--Modularity maxima

Upon analyzes of the network in Figure 1, the number of links inside each cluster is approximated to:

$$k_c \approx \frac{2L}{n_c} \tag{12}$$

Considering kA = kB = kC and applying (7) to the previous network, modularity variation is calculated in terms of the number of nodes nc:

$$\Delta M_{AB} = \frac{l_{AB}}{L} - \frac{k_A k_B}{2L^2} = \frac{l_{AB}}{L} - \frac{2}{n_c^2} \tag{13}$$

Merging two random clusters of the previous network in the same community will decrease modularity at most 2/nc^2. In the limit, this variation is undetectable:

$$\lim_{n_c \to \infty} \frac{2}{n_c^2} = 0 \tag{14}$$

Modularity Maximization--Modularity maxima

Empirically, the best partition should be the one that groups each 5-node cluster in different communities. Although, modularity increases by 0.003 whenever two adjacent communities are merged. Moreover, if random 5-node clusters are assigned to a community, even if they are not directly connected, it results in a modularity variation close to 0 around the one detected for the optimal partition. This complies with the vision of the plateau in the modularity graph that may distort the choice of the best partition in Figure 1. This plateau explains why a large number of modularity maximization algorithms can quickly detect high modularity partitions — they are not unique.

Modularity optimization algorithms are part of a larger set of problems that are solved by optimizing a quality function.

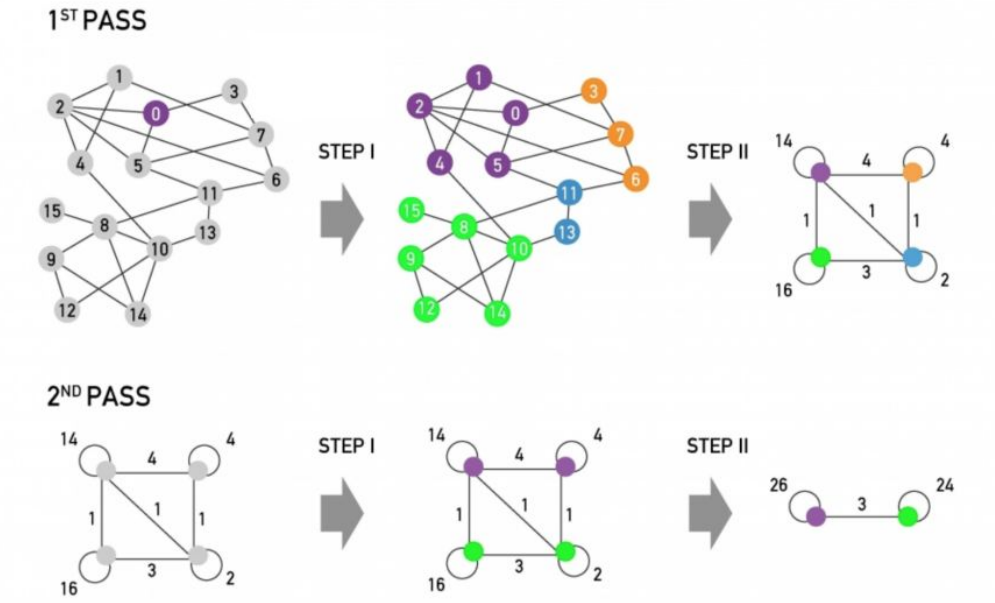
Louvain Method

Two Steps:

1. Find the largest gain in modularity and move node i to node j's community. Stop when it can't achieved more improvement.
2. Construct a new network according to step1.
    a. Weight of the link between two nodes(not in the same community): sum of the weight in the corresponding communities
    b. Weight of the link between two nodes(in the same community): weighted of self-loops

Do the following two steps until there are no more changes and gain max modularity.

Example



Louvain Method

- Faster than other algorithms
- Number of communities decreases drastically after a few steps

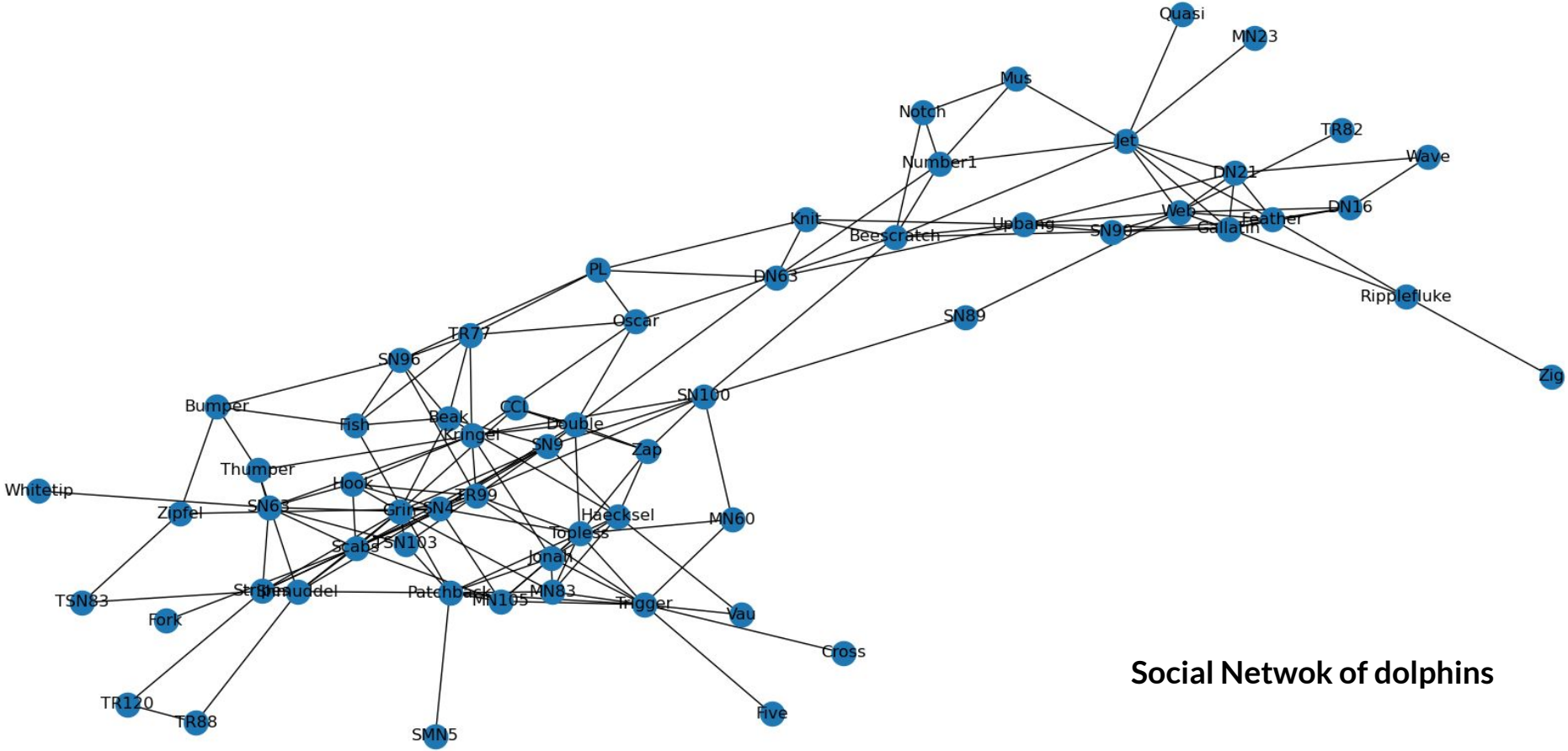
Louvain Method Implementation

source code: https://github.com/anirban-code-to-live/CommunityDetector.git

data:

An undirected social network of frequent acssociations between 62 dolphins in a community of

Doubtful Sound, New Zealand.



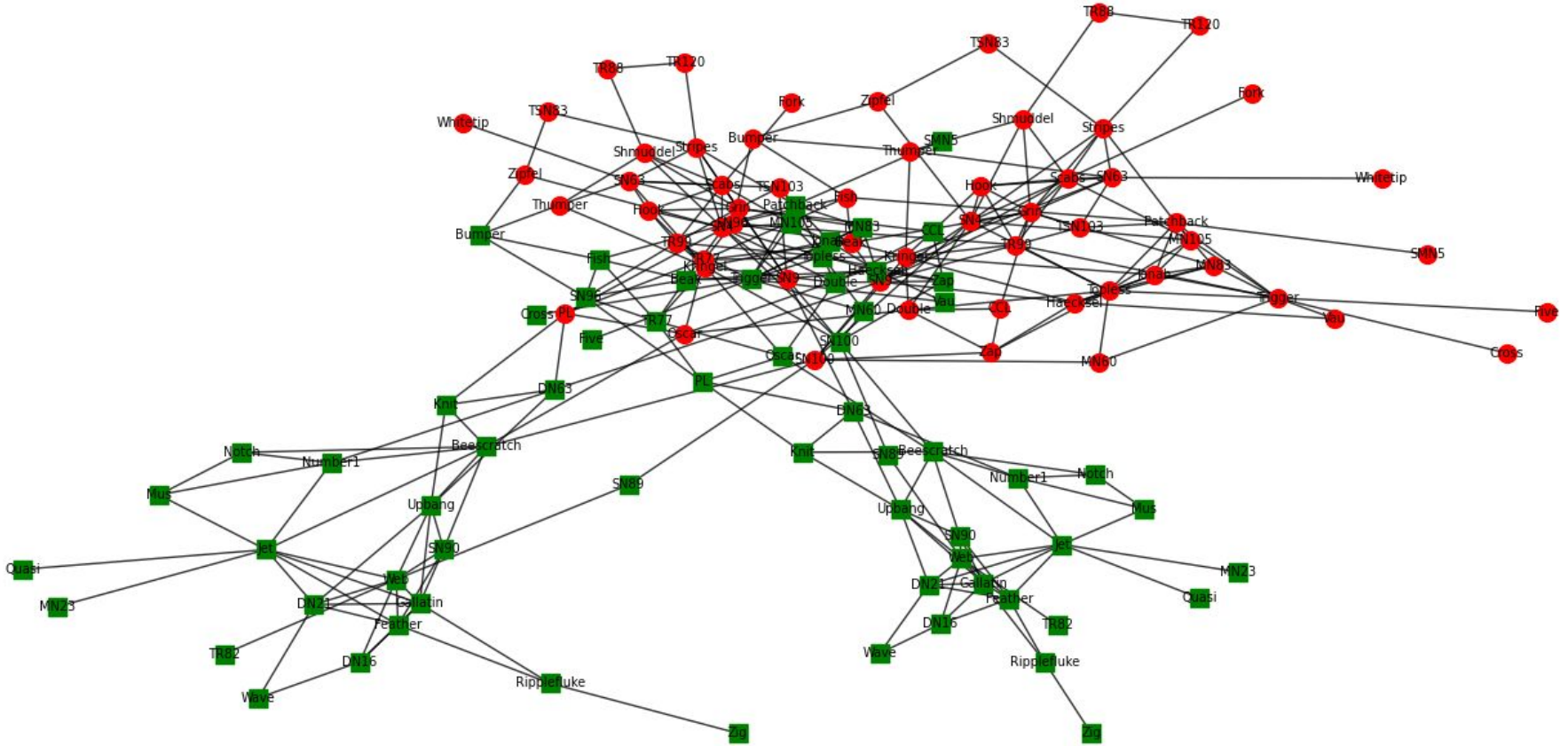
Social Netwok of dolphins

Louvain Method Implementation

Use Louvain Method to identify the undirected social network into two communities.

```
# Solution for part-I :: Using Louvain Method
louvain_method = lm.LouvainMethod(dolphin_graph)
positive_cluster_louvain, negative_cluster_louvain = louvain_method.find_communities()
louvain_method.draw_graph_with_community_structure(positive_cluster_louvain, negative_cluster_louvain)
print('Following are the two clusters obtained using Louvain Method')
print('Cluster-1 ::: ')
print(positive_cluster_louvain)
print('Cluster-2 ::: ')
print(negative_cluster_louvain)
```

Result after using Louvain Method



Louvain Method Implementation

```
Following are the two clusters obtained using Louvain Method
Cluster-1 :::
['Fork', 'Grin', 'Hook', 'Kringel', 'Scabs', 'Shmuddel', 'SN4', 'SN63', 'SN9', 'Stripes', 'Thumper', 'TR120', 'TR88', 'TR99', 'TSN103', 'TSN83', 'Whitetip', 'Zipfel']
Cluster-2 :::
['Beak', 'Bumper', 'DN63', 'Fish', 'Knit', 'PL', 'SN96', 'TR77', 'CCL', 'Double', 'Oscar', 'SN100', 'SN89', 'Zap', 'Cross', 'Five', 'Haecksel', 'Jonah', 'MN105', 'MN60', 'MN83', 'Patchback
', 'SMN5', 'Topless', 'Trigger', 'Vau', 'Beescratch', 'DN16', 'DN21', 'Feather', 'Gallatin', 'Jet', 'MN23', 'Mus', 'Notch', 'Number1', 'Quasi', 'Ripplefluke', 'SN90', 'TR82', 'Upbang', 'Wa
ve', 'Web', 'Zig']
(base) bibibobo@bibibobo-virtual-machine:~/CommunityDetector/Module-II$
```

三、如何評估實驗結果

(介紹Precision、Recall、F-1 score、ROC曲線)

處理數據:二分法矩陣

在machine Learning中，常常可見二分法的使用，如辨識圖片，分類任務等。

		預測Predict	
		T	**F**
實際 Actual	**T**	TP	TN
	F	FP	FN

Topic：分類(性別)

預設男生為 "正"

		預測Predict	
		T	F
實際 Actual	T	TP	TN
	F	FP	FN

預測正確!!

In case：

TP：預測正確的正樣本　　正→正　　男生被機器說成男生

TN：預測正確的負樣本　　負→負　　女生被機器說成女生

Topic：分類(性別)

預設男生為 "正"

		預測Predict	
		T	F
實際 Actual	T	TP	TN
	F	FP	FN

預測錯誤!!

In case：

FP：預測錯誤的負樣本　　負→正　　女生被機器說成男生

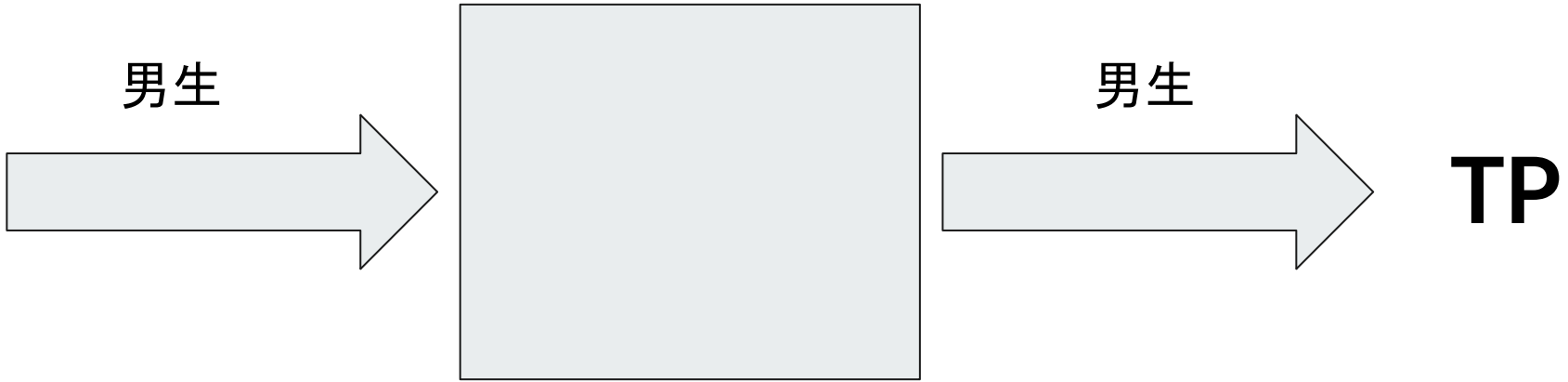
FN：預測錯誤的正樣本　　正→負　　男生被機器說成女生

Topic：分類(性別)

預設男生為 "正"

		預測Predict	
		T	F
實際 Actual	T	TP	TN
	F	FP	FN

輸入數據(實際) **機器** **輸出機器的判決(預測)**

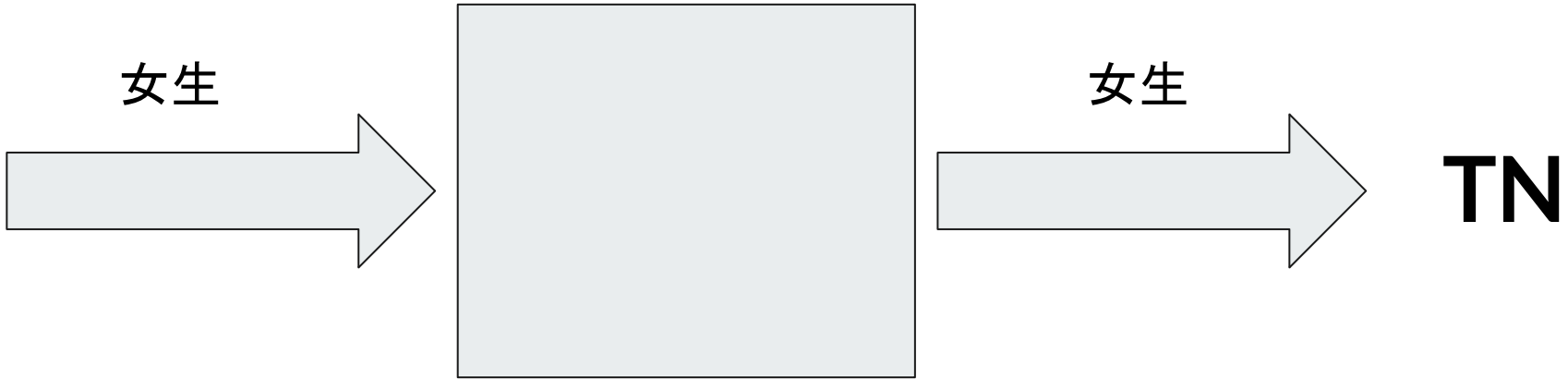


Topic：分類(性別)

預設男生為 "正"

		預測Predict	
		T	F
實際 Actual	T	TP	TN
	F	FP	FN

輸入數據(實際) **機器** **輸出機器的判決(預測)**

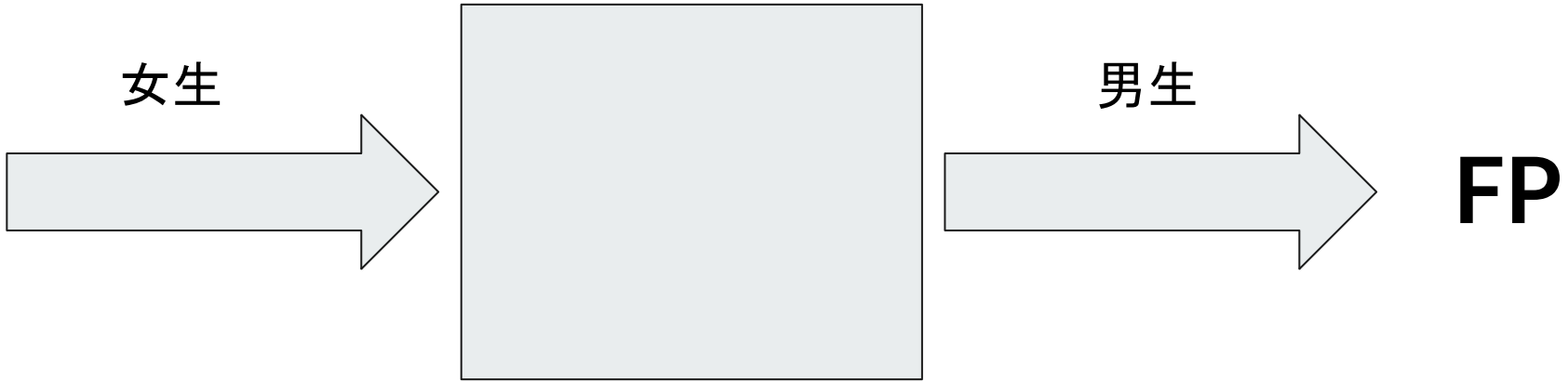


Topic：分類(性別)

預設男生為 "正"

		預測Predict	
		T	F
實際 Actual	T	TP	TN
	F	FP	FN

輸入數據(實際) **機器** **輸出機器的判決(預測)**

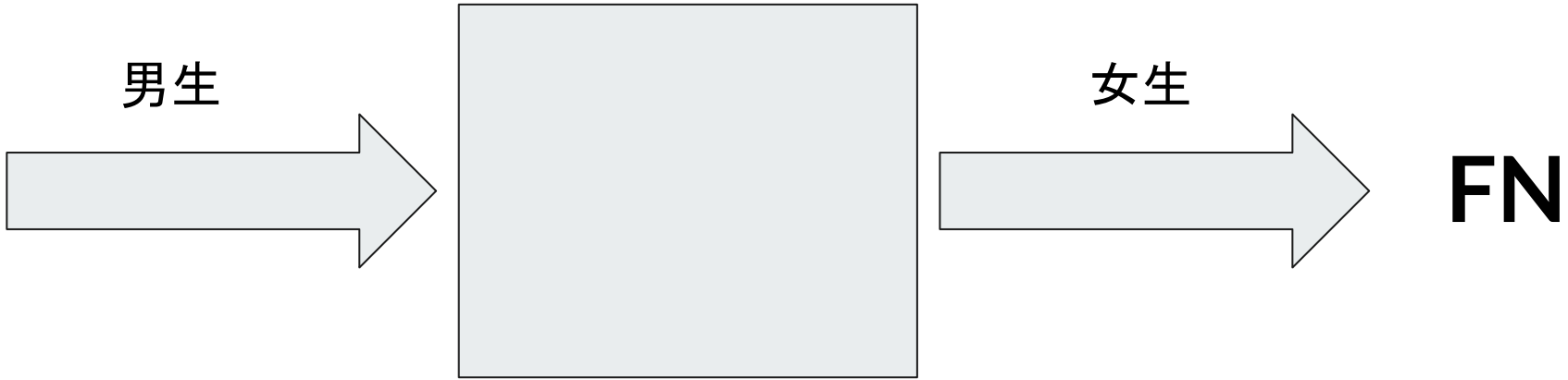


Topic：分類(性別)

預設男生為 "正"

		預測Predict	
		T	F
實際 Actual	T	TP	TN
	F	FP	FN

輸入數據(實際) **機器** **輸出機器的判決(預測)**



應用於community detection (設屬於A社區為 "正")

		預測Predict	
		T	F
實際 Actual	T	TP 實際屬於A社區 被預測屬於A社區的點	TN 實際不屬於A社區 被預測不屬於A社區的點
	F	FP 實際不屬於A社區的點 錯誤預判為屬於A社區	FN 實際屬於A社區的點 錯誤預判為不屬於A社區

分析數據 (Precision、Recall、Accurancy、F-1 score)

Precision (精確率) : TP / (TP + FP)

Recall (召回率) : TP / (TP + FN)

Accurancy (準確率) : (TP + TN) / (TP +TN +FP +FN)

F-1 score : (2 * Recall * Precision) / (Recall + Precision)

※調和平均數 : 2 / [(1 / Precision) + (1 / Recall)]

分析數據

Accurancy (準確率) : TP + TN / (TP + TN + FP + FN)

- 機器判斷正確的比率

分析數據 Topic：臉部辨識門鎖

		預測Predict	
		T	F
實際 Actual	T	TP	TN
	F	FP	FN

Precision (精確率)：TP / (TP + FP)

FP的代價較大!!

主人被辨識成主人 / (主人被辨識成主人 + **外人被錯誤辨識成主人**)

Recall (召回率)：TP / (TP + FN)

主人被辨識成主人 / (主人被辨識成主人 + **主人被錯誤辨識成外人**)

分析數據 Topic：癌症判斷 (有病為"正")

		預測Predict	
		T	F
實際 Actual	T	TP	TN
	F	FP	FN

Precision (精確率) : TP / (TP + FP)

確診的人被診斷有病 / (確診的人被診斷有病 + **沒病的人被誤診成有病**)

Recall (召回率) : TP / (TP + FN)

FN的代價較大!!

確診的人被診斷有病 / (確診的人被診斷有病 + **有病的人被誤診成沒病**)

分析數據 F-1 score

		預測Predict	
		T	F
實際 Actual	T	TP	TN
	F	FP	FN

F-1 score : (2 * Recall * Precision) / (Recall + Precision)

- 為Precision 和 Recall的調和平均數 : 2 / [(1 / Precision) + (1 / Recall)]
- 容易受到極值的影響
- 可分為micro F-1 和 Macro F-1

分析數據 (貓狗豬分群)

		預測Predict		
		貓	狗	豬
實際 Actual	貓			
	狗			
	豬			

分析數據

		預測		
		貓	狗	豬
實際	貓			
	狗			
	豬			

實際	機器判斷
貓	貓
貓	貓
狗	貓
豬	貓
狗	狗
狗	狗
豬	狗
狗	豬
豬	豬

分析數據

		預測		
		貓	狗	豬
實際	貓	2(TP)	0(FN)	0(FN)
	狗	1(FP)	(TN)	(TN)
	豬	1(FP)	(TN)	(TN)

實際	機器判斷
貓	貓
貓	貓
狗	貓
豬	貓
狗	狗
狗	狗
豬	狗
狗	豬
豬	豬

分析數據 三分類矩陣

		預測		
		貓	狗	豬
實際	貓	2	0	0
	狗	1	2	1
	豬	1	1	

實際	機器判斷
貓	貓
貓	貓
狗	貓
豬	貓
狗	狗
狗	狗
豬	狗
狗	豬
豬	豬

分析數據 三分類矩陣

		預測		
		貓	狗	豬
實際	貓	2	0	0
	狗	1	2	1
	豬	1	1	1

實際	機器判斷
貓	貓
貓	貓
狗	貓
豬	貓
狗	狗
狗	狗
豬	狗
狗	豬
豬	豬

分析數據 Micro F-1

		預測		
		貓	狗	豬
實際	貓	2	0	0
	狗	1	2	1
	豬	1	1	1

Micro F-1：看全部

貓：TP=2、TN=5、FP=0、FN=2

狗：TP=2、TN=4、FP=2、FN=1

豬：TP=1、TN=5、FP=2、FN=1

ALL：TP=5、TN=14、**FP=4**、**FN=4**

Precision_micro：

TP_ALL / (TP_ALL + FP_ALL) = 0.55

Recall_micro：

TP_ALL / (TP_ALL + FN_ALL) = 0.55

Micro F-1：0.55

分析數據 Macro F-1

		預測		
		貓	狗	豬
實際	貓	2	0	0
	狗	1	2	1
	豬	1	1	1

Macro F-1：分開看

貓：TP=2、TN=5、FP=0、FN=2
Pre_貓=1、Rec_貓=0.5、F-1_貓=0.66

狗：TP=2、TN=4、FP=2、FN=1
Pre_狗=0.5、Rec_狗=0.67、F-1_狗=0.57

豬：TP=1、TN=5、FP=2、FN=1
Pre_豬=0.33、Rec_豬=0.5、F-1_豬=0.39

Marco F-1：(F-1_貓 + F-1_狗 + F-1_豬) / 3 = 0.54

分析數據 運用時機

- Accurany：最基本的評斷指標，但容易受到正負樣本數差距影響。
- Precision、Recall：FN或FP代價大時適用。
- F-1 score：調和平均數通常有效，但容易受極 值影響。
    - Micro F-1：注重樣本真實分佈，只考慮全域性效果
    - Macro F-1：在各個類別同等重要的情況，可保障小樣本的效能

Precision (精確率) : TP / (TP + FP)

Recall (召回率) : TP / (TP + FN)

分析數據 範例模型

Precision極高，Recall極低　　→　　FP極小，FN極大

e.g. 臉部辨識門鎖(主人為"正")

FP : 外人被誤判成主人、FN : 主人被誤判成外人

→主人可能偶爾會進不去，但外人幾乎進不去

※寧願將正誤判成負，也不願將負誤判成正

Precision (精確率) : TP / (TP + FP)

Recall (召回率) : TP / (TP + FN)

分析數據 範例模型

Precision極低，Recall極高　　→　　FP極大，FN極小

e.g. 海關通緝犯辨識(通緝犯為"正")

FP : 無罪的人被誤判成通緝犯、FN : 通緝犯被誤判成無罪的人

→無罪的人可能偶爾會被誤認成通緝犯，但通緝犯幾乎都會被認出來

※寧願將負誤判成正，也不願將正誤判成負

分析數據 ROC曲線

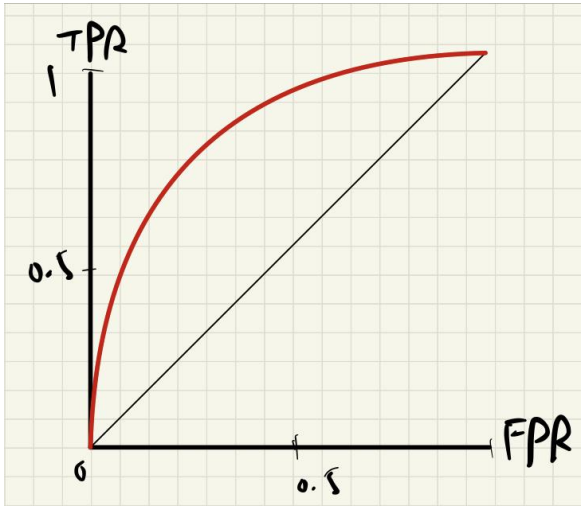
ROC曲線以FPR為X軸，TPR為Y軸，每一個點代表設定不同的門檻而得到不同的FPR與TPR，最後繪製成一條曲線。

sensitivity靈敏度

TPR = TP / (TP + FN) = Recall

FPR = FP / (FP + TN) = 1 - TN / (FP + TN)

specificity特異度



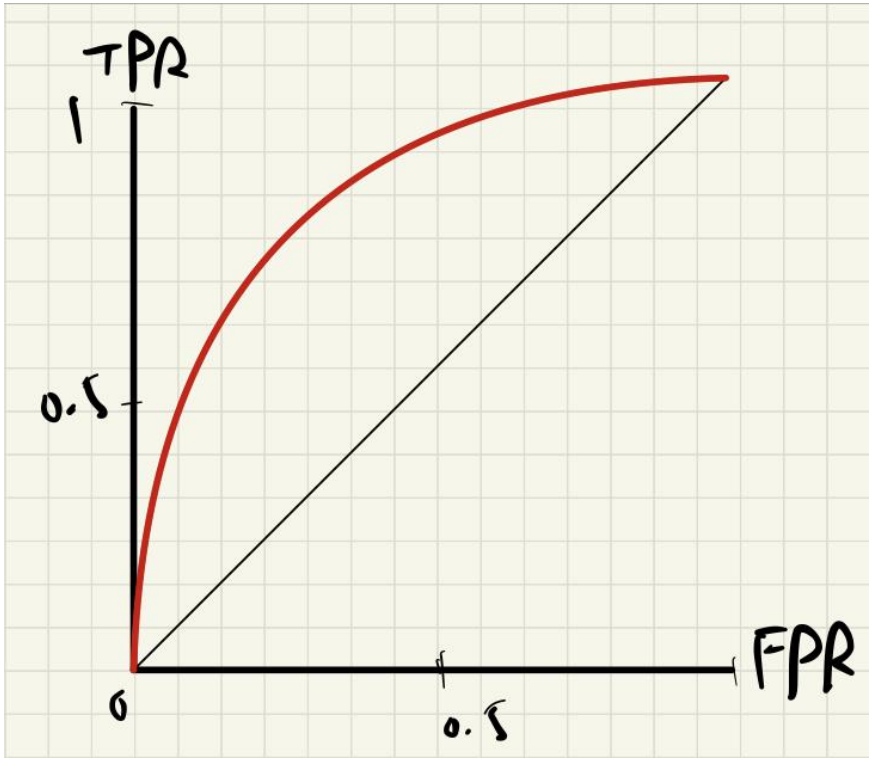
分析數據 ROC曲線

TPR = TP / (TP + FN)

FPR = FP / (FP + TN) = 1 - TN / (FP + TN)

越靠近(0,1)則表現越好

越靠近(1,0)則表現越差



分析數據 ROC曲線

1. 蒐集樣本
2. 將value 由小至大排列
3. 設定門檻
4. 將大於門檻的點的value設為1
5. 計算TPR、FPR、更新門檻
6. 重複4. 、5.
7. 完成

No	class	value
1	1	0.5
2	1	0.6
3	0	0.55
4	0	0.4
5	1	0.7

分析數據 ROC曲線

1. 蒐集樣本
2. 將value 由小至大排列
3. 設定門檻
4. 將大於門檻的點的value設為1
5. 計算TPR、FPR、更新門檻
6. 重複4. 、5.
7. 完成

No	class	value
1	0	0.4
2	1	0.5
3	0	0.55
4	1	0.6
5	1	0.7

分析數據 ROC曲線

threshlod：0.4

1. 蒐集樣本
2. 將value 由小至大排列
3. 設定門檻
4. 將大於門檻的點的value設為1
5. 計算TPR、FPR、更新門檻
6. 重複4. 、5.
7. 完成

No	class	value
1	0	0.4
2	1	0.5
3	0	0.55
4	1	0.6
5	1	0.7

分析數據 ROC曲線

threshlod : 0.4

1. 蒐集樣本
2. 將value 由小至大排列
3. 設定門檻
4. 將大於門檻的點的class設為1
5. 計算TPR、FPR、更新門檻
6. 重複4. 、5.
7. 完成

No	class	value	class_ thrshold
1	0	0.4	1
2	1	0.5	1
3	0	0.55	1
4	1	0.6	1
5	1	0.7	1

分析數據 ROC曲線

threshlod : 0.4 →0.5

1. 蒐集樣本
2. 將value 由小至大排列
3. 設定門檻
4. 將大於門檻的點的class設為1
5. 計算TPR、FPR、更新門檻
6. 重複4. 、5.
7. 完成

TP=3、FP=2
TN=0、FN=0

FPR=1、TPR=1

No	class	value	class_ thrshold
1	0	0.4	1
2	1	0.5	1
3	0	0.55	1
4	1	0.6	1
5	1	0.7	1

分析數據 ROC曲線

threshlod : 0.5

1. 蒐集樣本
2. 將value 由小至大排列
3. 設定門檻
4. 將大於門檻的點的class設為1
5. 計算TPR、FPR、更新門檻
6. 重複4. 、5.
7. 完成

No	class	value	class_ thrshold
1	0	0.4	0
2	1	0.5	1
3	0	0.55	1
4	1	0.6	1
5	1	0.7	1

分析數據 ROC曲線

threshlod : 0.55

1. 蒐集樣本
2. 將value 由小至大排列
3. 設定門檻
4. 將大於門檻的點的class設為1
5. 計算TPR、FPR、更新門檻
6. 重複4. 、5.
7. 完成

No	class	value	class_ thrshold
1	0	0.4	0
2	1	0.5	0
3	0	0.55	1
4	1	0.6	1
5	1	0.7	1

分析數據 ROC曲線

threshlod : 0.6

1. 蒐集樣本
2. 將value 由小至大排列
3. 設定門檻
4. 將大於門檻的點的class設為1
5. 計算TPR、FPR、更新門檻
6. 重複4. 、5.
7. 完成

No	class	value	class_ thrshold
1	0	0.4	0
2	1	0.5	0
3	0	0.55	0
4	1	0.6	1
5	1	0.7	1

分析數據 ROC曲線

threshlod : 0.7

1. 蒐集樣本
2. 將value 由小至大排列
3. 設定門檻
4. 將大於門檻的點的class設為1
5. 計算TPR、FPR、更新門檻
6. 重複4. 、5.
7. 完成

No	class	value	class_ thrshold
1	0	0.4	0
2	1	0.5	0
3	0	0.55	0
4	1	0.6	0
5	1	0.7	1

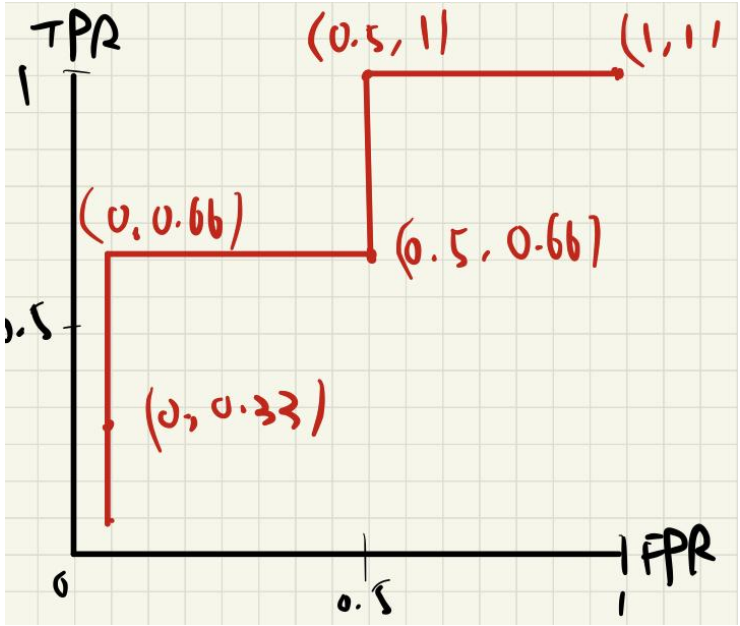
分析數據 ROC曲線

TPR = TP / (TP + FN)

FPR = FP / (FP + TN) = 1 - TN / (FP + TN)

	TP	TN	FP	FN	TPR	FPR
0.4	3	0	2	0	1	1
0.5	3	1	1	0	1	0.5
0.55	2	1	1	1	0.66	0.5
0.6	2	2	0	1	0.66	0
0.7	1	2	2	0	0.33	0

分析數據 ROC曲線



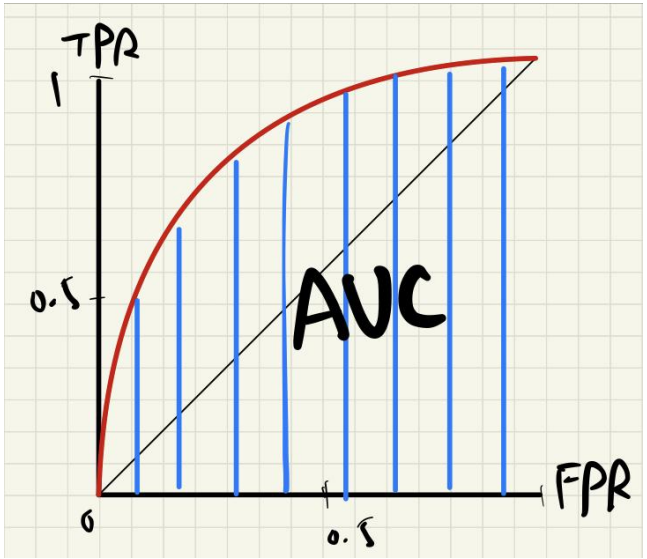
FPR	TPR
1	1
0.5	1
0.5	0.66
0	0.66
0	0.33

分析數據 AUC

AUC (area under curve) :

ROC曲線以下的面積比例

AUC = 1 **(完美表現)**
0.5 > AUC > 1 **(表現尚可)**
AUC = 0.5 **(無參考價值)**
0 < AUC < 0.5 **(反指標)**



謝謝。