**1.标签传播算法（LPA）：**

**Near linear time algorithm to detect community structures in large-scale networks**

社团的定义没有统一定义，但是普遍认为，同一个社团里面的节点紧密连接，不同社团之前稀疏连接。  
第一步: 为所有节点指定一个唯一的标签；   
第二步: 逐轮刷新所有节点的标签，直到达到收敛要求（每一个节点和它所有邻居的标签众数相同）为止。对于每一轮刷新，节点标签刷新的规则如下: 对于某一个节点，考察其所有邻居节点的标签，并进行统计，将出现个数最多的那个标签赋给当前节点。当个数最多的标签不唯一时，随机选一个。考虑到子图的对称性会导致标签的震荡，采取非同步更新的策略（一部分使用上一轮的标签，一部分使用本轮的标签，均匀随机划断），作者称五轮迭代大概95%的节点标签分对。但是这样缺点在于，不便于并行，引入随机处理不稳定，可能会导致巨型社区（monster community）和少数一些小型社区（several small communities)的聚类结果。

**2.改进:Community Detection via**

**Semi–Synchronous Label Propagation Algorithms**

采用涂色方法（相邻点颜色不一样),把相同颜色的放到一组，每一轮迭代，外层循环是颜色组别，内层循环是每一组里面的节点，进行标签传播更新，可以避免随机不稳定，同时方便并行，更快收敛。

# **3.改进：SLPA: Uncovering Overlapping Communities in Social Networks via A Speaker-listener Interaction Dynamic Process**

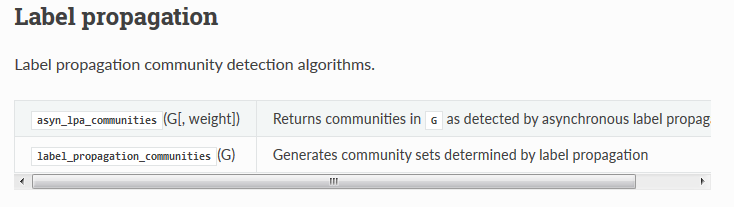
# SLPA 改进，引入了 Listener 和 Speaker 两个比较形象的概念，在刷新节点标签的过程中，任意选取一个节点作为 listener，则其所有邻居节点就是它的 speaker 了，speaker 通常不止一个，需要制定一个倾听规则。

在 LPA 中，以出现次数最多的标签来做决断，是一种规则。在 SLPA 框架里，规则的选取比较多（可以由用户指定）。

与 LPA 相比，SLPA 最大的特点在于：它会记录每一个节点在刷新迭代过程中的历史标签序列（例如迭代 T 次，则每个节点将保存一个长度为 T 的序列），当迭代停止后，对每一个节点历史标签序列中各（互异）标签出现的频率做统计，按照某一给定的阀值过滤掉那些出现频率小的标签，剩下的即为该节点的标签（通常有**多个**）。

Python实现：

**networkx**：基于python的图计算包、文档清晰、接口使用方便，但自带的社区发现算法目前只有k-clique和LPA(label propagation algorithms)，k-clique算法仅能找出一些紧密团且找出的社区会有部分重叠，LPA算法引入了随机性、导致结果不稳定。

Networkx实现了两种：

文档：

1. <https://networkx.github.io/documentation/stable/reference/algorithms/generated/networkx.algorithms.community.label_propagation.asyn_lpa_communities.html#networkx.algorithms.community.label_propagation.asyn_lpa_communities>对应论文1
2. [https://networkx.github.io/documentation/stable/reference/algorithms/community.html?highlight=label%20propagation#module-networkx.algorithms.community.label\_propagation](https://networkx.github.io/documentation/stable/reference/algorithms/community.html?highlight=label%20propagation#module-networkx.algorithms.community.label_propagation，)对应论文2
3. <https://github.com/GraphProcessor/CommunityDetectionCodes/tree/master/Algorithms/2011-SLPA，> 对应论文3，Python版基于networkx,除了Python版还有Java版，另一个Java版：https://github.com/kbalasu/SLPA

**igraph**：c实现的，对R语言的支持更好，也有python版但是支持不好，社区发现算法实现的比较全

文档：<http://igraph.org/python/doc/igraph.Graph-class.html#community_label_propagation> 对应论文1

Spark graphx 实现：<http://spark.apache.org/docs/latest/api/scala/index.html#org.apache.spark.graphx.lib.LabelPropagation$> 论文1

1. **改进：Finding overlapping communities in networks by label propagation**

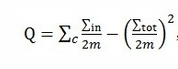
不同于原算法，该算法节点能属于多个社区（overlapping）

graphX实现：https://github.com/bhardwajank/graphx-overlapping-community/blob/db79e77f7cd185570cf5beda50372f3cda26fdde/README.md

**5.Louvain算法：Fast unfolding of communities in large networks**

基于优化Modularity值的思路，[Modularity值](http://arxiv.org/pdf/physics/0602124.pdf) 用于评估社区发现的效果，对比社区发现结果与随机图（Null Model）的差异。对于同一个输入图进行不同的社区发现策略，取得Modularity值较高的策略性能较好。

模块度的计算：

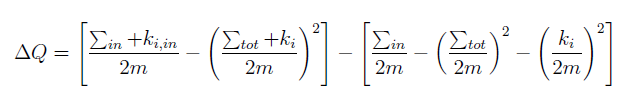


其中，∑in 表示一个社区内部的连线数，∑tot 表示一个社区所有节点的度数之和。

主要步骤：

1.初始化，将每个节点划分在不同的社区中。

2.逐一选择各个节点，根据下面公式计算将它划分到它的邻居社区中得到的Modularity增益。如果最大增益大于0，则将它划分到对应的邻居社区；否则，保持归属于原社区。重复步骤，直到节点的社区不再发生变化（不能再增大模块度）。

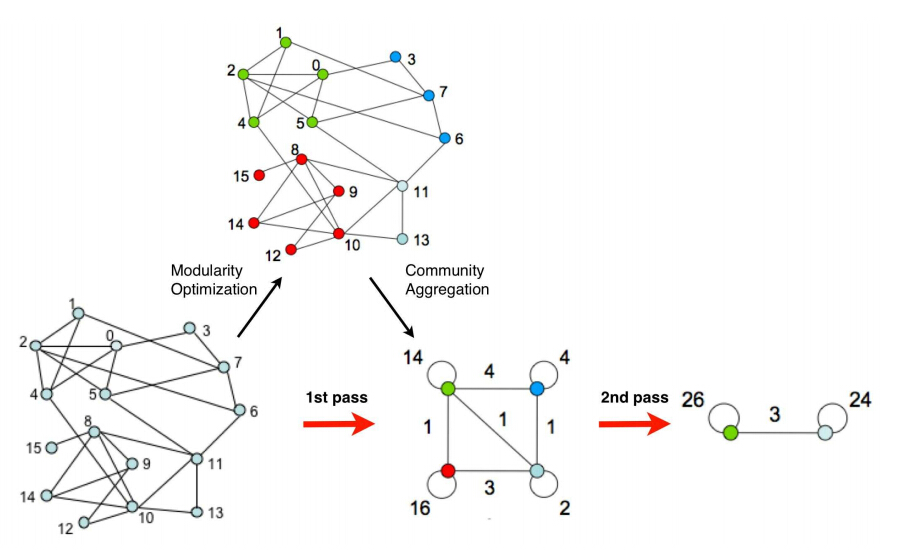


3.构建新图。新图中的点代表上一阶段产生的不同社区，边的权重为两个社区中所有节点对的边权重之和。重复步骤2，直到获得最大的Modularity值。

可以将上述步骤分为两阶段（Pass）

* 1st Pass: 包含步骤1至2，用于设定各节点的归属社区，直到不再发生变化
* 2nd Pass: 由步骤3组成，用于构建新图，并重新执行1st Pass的操作，直到Modularity值不再增加

FastUnfolding算法的一大优点会不断合并节点构造新图，大大减少了计算量。



Python实现:

https://python-louvain.readthedocs.io/en/latest/api.html

graphx实现:

https://github.com/Sotera/spark-distributed-louvain-modularity

**6.基于谱分析的社团发现算法** Community detection in graphs

将图用特定矩阵表示出来，比如拉普拉斯矩阵L=D-W，D为以每个节点的度为对角元的对角矩阵，W为图的邻接矩阵。同一社团的节点对应的特征分量近似相等，这是谱分析方法实现社团发现的理论基础。将节点对应的矩阵特征分量看作空间坐标，将网络中的节点映射到多维特征向量空间中，用传统的聚类方法将节点聚类成社团。

**7.基于团渗透改进的重叠社团发现** Uncovering the overlapping community structure of complex networks in nature and society

社团是由一系列相互可达的k-团（大小为k的完全子图）组成，通过合并相邻的k-团实现社团发现。

快速团渗透算法：

1.      将网络的边按顺序（权值大小）插入到网络中，同时检测出现的k-团

2.      将检测到的k-团根据是否与已有k-社团近邻，并入k-社团形成新的k-社团。

基于团渗透思想的算法需要以团为基本单位来发现重叠，这对于真实网络，特别是稀疏网络而言条件过于严格，只能发现少量的重叠社团。

Networkx包实现了k-clique算法，文档：

https://networkx.github.io/documentation/stable/reference/algorithms/generated/networkx.algorithms.community.kclique.k\_clique\_communities.html#networkx.algorithms.community.kclique.k\_clique\_communities

半监督分类问题：

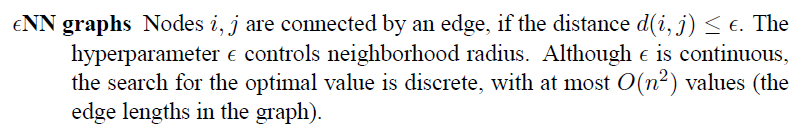
Semi-Supervised Learning with Graphs，

Learning from labeled and unlabeled data with label propagation

**相似矩阵构建：**

为所有的数据构建一个图，图的节点就是一个数据点，包含labeled和unlabeled的数据。节点i和节点j的边表示他们的相似度，叫做**Fully connected graphs**。常用的一种高斯径向核RBF：exp**-weighted graphs：**



还有knn图，也就是只保留每个节点的k近邻权重，其他的为0，也就是不存在边，因此是稀疏的相似矩阵，叫做**Sparse graphs-**k**NN graphs**。还有：



# 相似矩阵的基本要求是对称，元素非负。图的拉普拉斯矩阵（graph Laplacian）可以使其半正定。

这些方法都有超参数，参数学习方法在Semi-Supervised Learning with Graphs第7章。

**标签传播：**

通过节点之间的边传播label。边的权重越大，表示两个节点越相似，那么label越容易传播过去

定义一个N\*N的概率转移矩阵P：



假设有C个类和L个labeled样本，定义一个LxC的label矩阵YL，第i行表示第i个样本的标签指示向量，即如果第i个样本的类别是j，那么该行的第j个元素为1，其他为0。同样，也给U个unlabeled样本一个UxC的label矩阵YU。把他们合并，得到一个N\*C的soft label矩阵F=[YL;YU]。soft label表示保留样本i属于每个类别的概率，而不是互斥性的（这个样本以概率1只属于一个类）。在最后确定这个样本i的类别的时候，要取概率最大的那个类作为它的类别的。YU，未知，可以随机初始化或者都设为-1。

传播的步骤：

1）**执行传播：F=PF，**将矩阵P和矩阵F相乘，这一步，每个节点都将自己的label以P确定的概率传播给其他节点。如果两个节点越相似（在欧式空间中距离越近），那么对方的label就越容易被自己的label赋予，越容易形成聚集效应

 2）**重置F中labeled样本的标签：FL=YL，**因为labeled数据的label是事先确定的，每次传播后都需要重置回事先确定的label

 3）**重复步骤1）和2）直到F收敛，**随着labeled数据不断的将自己的label传播出去，最后的类边界会穿越高密度区域，而停留在低密度的间隔中，相当于每个不同类别的labeled样本划分了聚集范围。

改进：

每次迭代都是计算一个soft label矩阵，只计算FU就可以了，将矩阵P做以下划分：

，

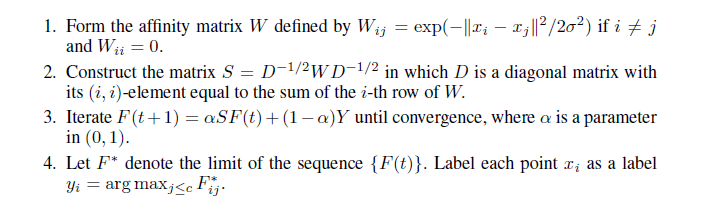
只需要计算;,迭代这个步骤直到收敛，不但取决于labeled数据的标签及其转移概率（这个也可以事先算好，以便之后每次调用），还取决了unlabeled数据的前一步label和转移概率。

当unlabeled数据非常多，而且类别也很多的时候，计算很慢的，同时占用的内存量也非常大。构造Graph需要计算两两的相似度，O(n2)复杂度，当数据的特征维度很大的时候，计算量也是非常大。可以考虑并行。对FU切分。

Python实现参考：<http://blog.csdn.net/zouxy09/article/details/49105265>

Sklearn实现：<http://scikit-learn.org/stable/modules/generated/sklearn.semi_supervised.LabelPropagation.html#sklearn.semi_supervised.LabelPropagation>

改进：Learning with Local and Global Consistency



定义了一个矩阵W，这个矩阵W表达了样本集合X所构成的图的各个边的权值。在步骤3中，等式右边的式子可以分为两个部分，第一个部分表示每个样本从其邻居点中得到标签信息；第二个部分表示每个样本需要保留其最原始的标签信息。这个方法和前面的一个方法相比多增加了保留原始标签和邻居标签信息的机制。在相似矩阵的构造上一样。

Sklearn实现：<http://scikit-learn.org/stable/modules/generated/sklearn.semi_supervised.LabelSpreading.html#sklearn.semi_supervised.LabelSpreading>

部分实验：<http://10.17.110.234:12307/notebooks/zhouhe/%E6%A0%87%E7%AD%BE%E4%BC%A0%E6%92%AD_python.ipynb>

http://scikit-learn.org/stable/modules/generated/sklearn.semi\_supervised.LabelSpreading.html