Node2vec理论与实践

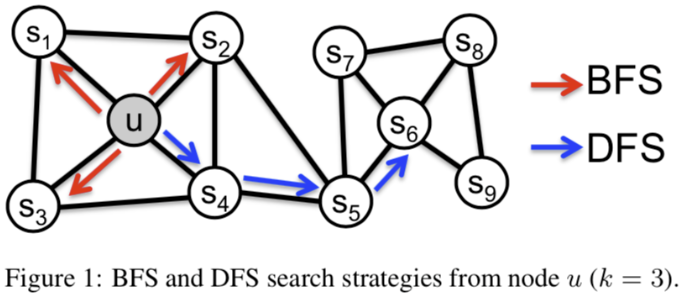
Node2vec是斯坦福大学的同学 2016 年发表于 ACM 的论文《node2vec: Scalable Feature Learning for Networks》。在这篇论文中作者提出了一个半监督学习算法—Node2Vec，采用了有偏的随机游走算法并结合Skip-gram算法学习Network Embedding，Node2Vec可以通过参数设置来控制搜索策略，从而有效的平衡了Embedding的同质性和同构性。

# 简介

Node2vec是Embedding技术在图中的应用，是DeepWalk的升级版（**要详细了解DeepWalk可以查阅：https://mp.weixin.qq.com/s/G3Y7\_8bCzevEc2uRTO2L8A**）。Deepwalk可以认为是random walk + skip-gram，random walk本质上是一个dfs的过程，丢失了bfs的邻居结构信息。在Node2vec论文中多次提到DeepWalk产生词序列的过程是僵硬、死板的，原因是DeepWalk的训练结果是依靠随机游走产生的序列的质量，没有解释节点间的关系。而node2vec可以简单理解为对deepwalk的随机游走过程进行优化，通过调节参数可以使用随机游走偏向bfs或dfs，从而使随机游走产生的词序列隐含节点之间的关系，主要是同义性（homophily）和同构性（structural equivalence）。与DeepWalk一样，Node2vec也是产生词序列，最终训练Embedding的过程仍然是skip-gram。

# 前置知识

论文中提出了两种度量节点相似性的方式，这里先探讨下，以更好的理解Node2vec的细节。



图中的结点大致可以分为两个community：一个是u,s1,s2,s3,s4组成的，以u为中心；另一个是由s5,s6,s7,s8,s9组成的，以s6为中心。

## 同质性

同质性，也称同义性、内容相似，具有直接链接关系的两个节点，即距离近的节点应该有相似的embedding，可以认为是内容相似的。例如上图中的s1和u。

同质性强调一个community里结点间的高度相互连接。例如图中的结点u和结点s1,s2,s3,s4的关系。通过广度优先搜索（BFS）获取的随机游走序列，可以突出结点间的同义性关系（简单理解，使劲转悠还是困在一个community里）。在经过skip-gram之后，同义性高的结点在特征空间里，距离要更近。

## 同构性

同构性强调结点在网络结构中表现出网络拓扑结构组成上是类似的特点，结构相似的节点也应该有相似的embedding。例如图中的结点u和结点s6。通过深度优先搜索（DFS）获取的随机游走序列，可以突出结点间的同构性关系（简单理解，游走的方向更容易跨越community）。在经过skip-gram之后，同构性高的结点在特征空间里，距离也应该更近。

# Node2vec模型

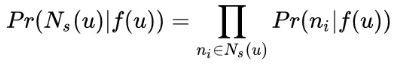
首先引入用于学习节点Embedding的Skip-gram算法，并给出目标函数为：



其中，u为节点，为节点u通过采样策略S得到的邻居节点的集合，f(u)是一个映射矩阵，相当于 将顶点u映射为Embedding的映射函数Work2Vec中的输入向量。

为简便起见，文章提出两个假设：

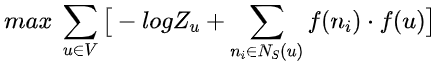
* 条件独立性假设：假设观察一个节点的领域与观察其他节点的领域相互独立，给定源顶点下，其近邻顶点出现的概率与近邻集合中其余顶点无关，所以有：



* 特征空间对称型假设：在特征空间中，源节点与邻节点相互对称。这里是说一个顶点作为源顶点和作为近邻顶点的时候共享同一套embedding向量。在这个假设下，上述条件概率公式可表示为：



基于上面的假设，目标函数改为



其中，由于计算代价昂贵，所以我们用Negative Sampling进行优化。

## 采样策略

由于网络是非线形的，所以需要一个策略来为Skip-gram提供一个线形的输入。一种常见的策略是通过游走的方式来对于给定源节点u的不同邻域进行采样，邻域不仅限与邻近的节点，而是与采样策略S有关。

评价网络节点的相似性我们有同质性和同构性两个概念。

* 同质性是指同属于一个集群的两个节点更加相似，如Figure1中的节点S1和节点u；
* 同构性是指两个具有相似结构的节点更加相似，如Figure1中的节点S6和节点u。

Node2vec兼顾了同质性和同构性。为了使Embedding的结果具有结构性，在随机游走的过程中，需要让游走的过程更倾向于宽度优先搜索（BFS）；另一方面，为了抓住网络的同质性，就需要随机游走更倾向于深度优先搜索（DFS）。

原因是BFS可以获得每个节点的邻居，强调的是局部微观视图，所以通过BFS采样的网络更能体现网络的局部结构，从而Embedding结果更能体现结构性（同构性）；而DFS可以探索更大的网络结构，只有从更高的角度才能观察到更大的集群，所以其Embedding结果更能体现同质性。

这个答案是不是与我们的直觉有所相悖？

在node2vec算法中，通过两个超参数p和q来控制节点间的跳转概率来控制随机游走的倾向。

这种灵活的游走方式使得node2vec可以更好的挖掘出图关系的特征表达，以推荐系统为例：

* 同质性相同的物品可能是同类目、同属性，或者经常被加到同一个购物车的商品；
* 结构性相同的物品可能是爆款、满减凑单，套餐折扣等具有相似结构特征的商品。

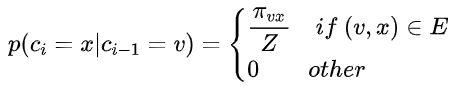
这两种信息都很重要，都可以是推荐的理由。因此，甚至可以通过不同参数的node2vec来召回不同的商品列表。

|  |
| --- |
| Node2vec提供的游走方式可以挖掘出结构相似等序列本身没有的信息。而DeepWalk不可以。 |

## 有偏的随机游走

node2vec依然采用随机游走的方式获取顶点的近邻序列，不同的是node2vec采用的是一种有偏的随机游走。

给定当前顶点v，访问下一个顶点x的概率为：



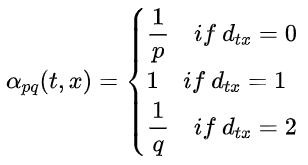
其中，ci表示第i次游走，表示节点x和节点v之间的未归一化之前的转移概率（即从节点v转移到节点x的概率），Z为归一化常数。

如果需要产生有偏的随机游走，一个比较简单的方法是令，但这样就没法适应不同网络结构，也不能引导我们的程序去探索不同类型的网络邻居。另外这里有偏的随机游走策略应该是统筹BFS和DFS的，以平衡同质性和结构等价性。

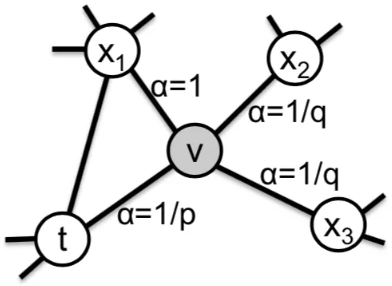
所以我们定义了一个非标准的转移概率 ：



其中，alpha表示搜索偏差：



其中，表示节点t和节点x之间的最短路径，取值为{0, 1, 2}。



结合上图来理解：

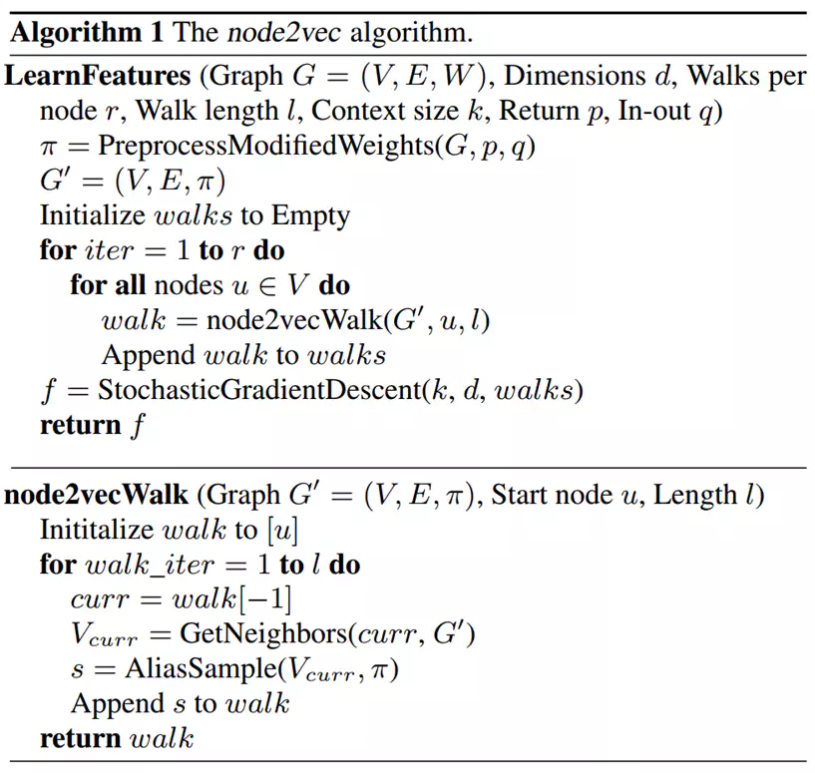
假设随机游走刚通过节点t来到节点v，现在考虑接下来的转移概率，节点x1与节点t的最短路径为1，所以，节点x2与节点t的最短路径为2，所以，节点x3同理。

直觉来看，参数p和q可以用来引导游走，近似地在BFS和DFS之间穿插，从而反映出不同节点在同质性和结构性上的亲和力。

* Return parameter p：参数p控制重复访问刚刚访问过的顶点的概率。注意到p仅作用于的情况，而表示顶点x就是访问当前顶点v之前刚刚访问过的顶点。那么若p较高，则访问刚刚访问过的顶点的概率会变低，反之变高。
* In-out parameter q：参数q允许搜索程序区分“向内”和“向外”的节点。如果q > 1则，倾向于访问靠近节点t的节点，类似于BFS策略；反之，倾向于访问远离节点t的节点，类似于DFS策略。

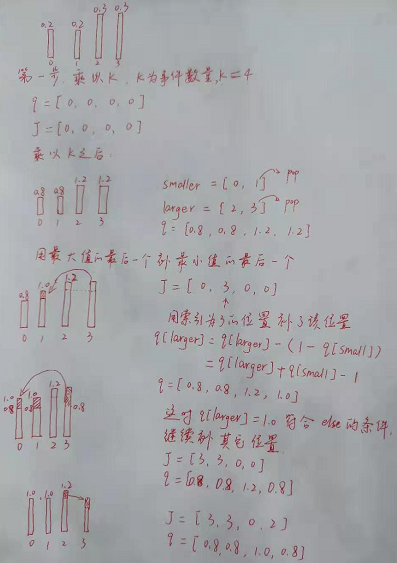
最初的随机游走算法由于要存储所有的边，所以空间复杂度为O(|E|)，而有偏置的随机游走空间复杂度为：，其中a是网络节点的平均度数，其值通常非常小。

Node2Vec的随机游走方法兼容了DFS和BFS的优点，并且具有较低的时间复杂度和空间复杂度。下面是Node2Vec的伪代码：



Node2Vec的算法共分为三个部分：预处理计算转移概率（PreprocessModifiedWeights，这个可以提前计算好），有偏置的随机游走（Node2VecWalk，加权采样使用的是Alias算法，其时间复杂度为O(1)）和异步随机梯度下降。每个阶段都可以并行化处理，这有助于加速 Node2Vec算法的训练。

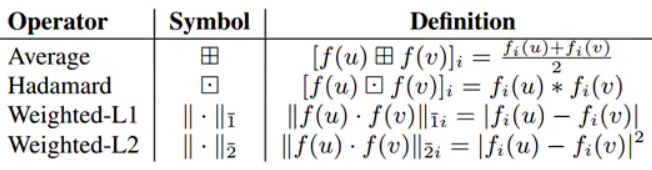
下图是alias采样算法示例，假设4个事件的初始权重分别是0.2,0.2,0.3,0.3。



## Edge embedding

论文还有一个创新的地方在于：除了原本的 Embedding任务还给出了边的预测的任务。

对于两个节点u和v，其Embedding向量表示是f(u)和f(v)，可以映射一个函数表示边的Embedding：g(u,v)，例如。其中，是映射后节点对(u,v)的Embedding维度。可选的操作有：



## Node2vec算法分析

基于随机游走的方法空间复杂度为O(|E|)，Node2vec是二阶方法，即和当前节点的上一个节点相关。这个时候存储临接的临接的节点可以节省时间，空间复杂度为，a是平均的degree。

论文中还提到了一种方法降低每次抽样的时间复杂度。对于k<l，采l长度的序列，可以当作k个(l-k)长度的序列。比如采样一个random walk 序列{u, s4, s5, s6, s8, s9}，长度为6，可以产生3个 {u, s4, s5, s6}, {s4, s5, s6, s8}和{s5, s6, s7, s8}。单次采样一个节点的时间复杂度为。上述方法会引入偏差，作者也指出会带来明显的效率上的提升。

## Node2vec与DeepWalk的差异

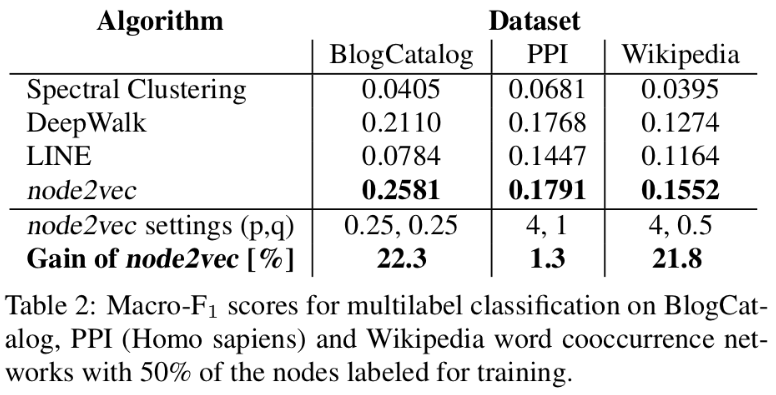
Node2Vec与DeepWalk最大的不同（甚至是唯一的不同）就是在于节点序列的生成机制。DeepWalk在每一步探索下一个节点时，是在其邻居节点中进行随机选择，然后基于深度优先策略生成一个固定长度的节点序列。而Node2Vec在生成节点序列时，引入了更加灵活的机制，通过几个超参数来控制向不同方向生长的概率。

# 实验

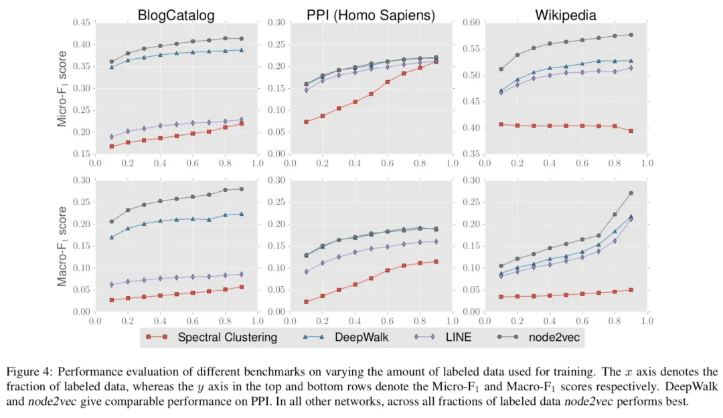
## Multi-label classification

这里主要和三种方法做了对比：

* 光谱聚类：一种矩阵分解方法，将图G的标准化拉普拉斯矩阵的顶部d个特征向量用作节点的特征向量表示。
* DeepWalk：其实就是Node2Vec的特殊情况（当 p=1，q=1 时）。
* LINE：两阶段学习特征，在第一阶段，它通过BFS式的模拟在节点的临近节点上学习d / 2维；在第二阶段，它通过严格从源节点开始以2跳距离采样节点来学习下一个d / 2维。

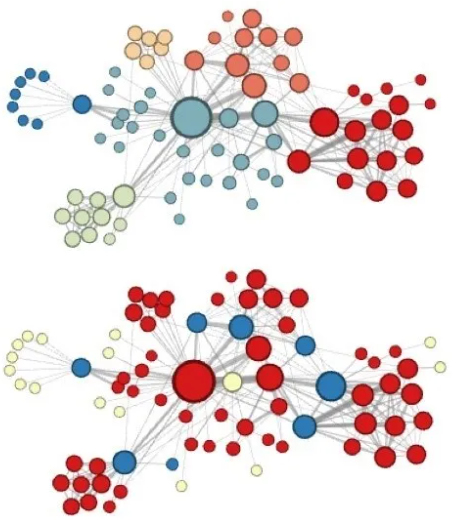


这个 Gain of node2vec 意思是采用pq设置带来的增益。可以看到node2vec相较于deepwalk效果有提升。

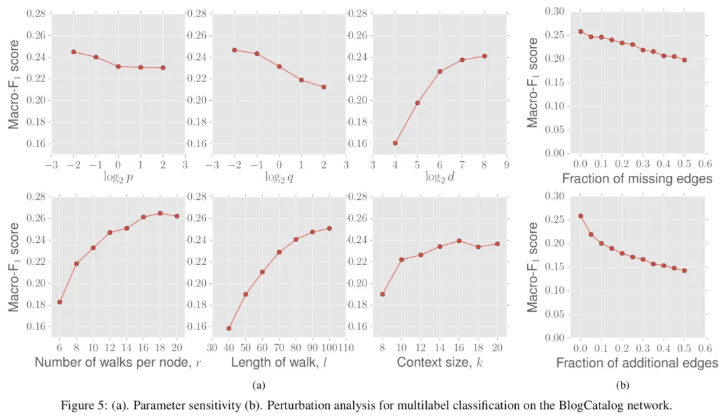


这里的x轴表示训练-测试数据比例。

来看下Node2Vec关于同质性和结构等价性的效果，下图为Node2Vec生成《悲惨世界》共现网络的可视化图，标签颜色可以反映同质性（上）和结构等价性（下）。图片的上半部分参数值为 p=1, q=0.5，图片的下半部分参数值为 p=1, q=2。



## 参数敏感度



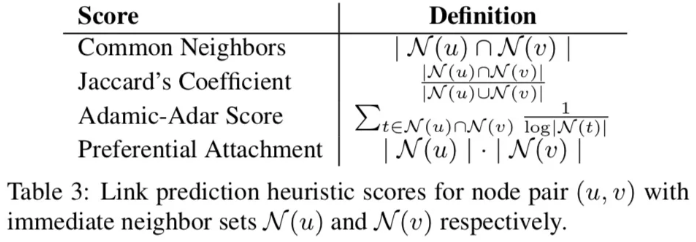
低q倾向于向外探索，低p倾向于在节点附近。d包括r, l, k。k就是训练词向量时的窗口大小。最后一列的两幅图分别表示缺失边和随机边。

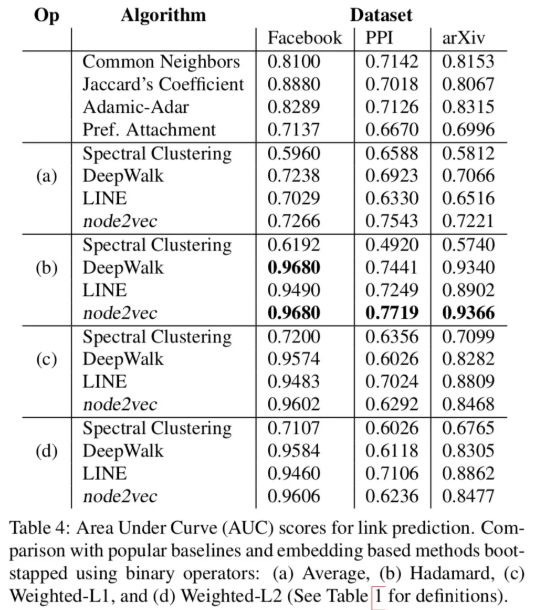
在公开数据集BlogCatalog上，以(多标签分类里有用到)为指标，使用不同的参数，讨论对模型效果的影响。显而易见，不管是增大p，还是增大q，模型的效果都变得更差了。“具体问题具体分析”——针对这个数据集，p小（高重返，偏向BFS）或者q小（高跳出，偏向DFS）对模型分类的结果更友好。

## 边的预测

随机移除50%的边作为正例，负例随机抽样无边的节点对。

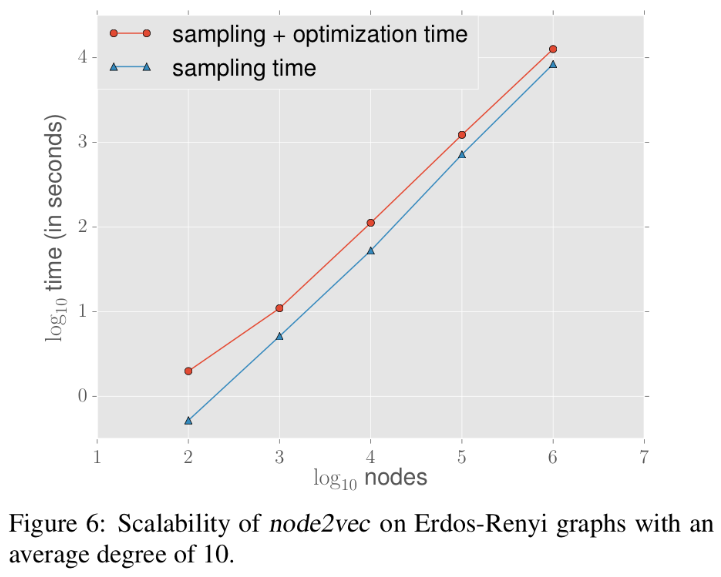
评分方法：





## 可扩展性

此处讨论的是结点个数对模型运行时间的影响。下图是在Erdos-Renyi图上依次采取100到1000000个结点（按10的倍数取），对模型运行时间的影响。



这里只分析红色的折线，在这个图上，算上采样和skip-gram训练模型的时间，时间对数和结点数对数基本上是线性的关系，算法整体的效率是非常好的。

# 实践

参考：

<https://github.com/jpegbert/NLP_Coding/tree/master/node2vec>

包含多个版本，多种实现方式，也有spark实现的demo。

# 总结

Node2Vec是一个新的NetWork Embedding算法，其综合BFS和DFS优缺点，提出了有偏的随机游走算法，给随机游走的过程提供了一个可控的方向，并且为它的控制做出了合理的解释，最终的实验表明其具有良好的性能和可伸缩性。

从这篇文章中主要学到了：

* 一种新的Network Embedding算法——Node2Vec；
* 有偏的随机游走算法；
* BFS和DFS采样方法带来的结构性和同质性；
* 一种边预测的判定条件。

参考

<https://mp.weixin.qq.com/s/1P7VTCQAZdsDCpXQOUQYgg>

<https://mp.weixin.qq.com/s/u8WZqlsIcGCU5BqPH23S4w>

<https://mp.weixin.qq.com/s/mMOTsWiDfB40TG3HGW-z0g>

<https://mp.weixin.qq.com/s/8nV4CIYQX3v31TtFosP99Q>

<https://zhuanlan.zhihu.com/p/54867139>

<https://mp.weixin.qq.com/s/8nV4CIYQX3v31TtFosP99Q>

<https://mp.weixin.qq.com/s/c_9CG0XLahoaQWSAsKrIgg>

<https://mp.weixin.qq.com/s/1G90zK18NMN-nRYXBd2ccA>

<https://mp.weixin.qq.com/s/4g84n7cPF5MMMihvPDhWKg>

<https://mp.weixin.qq.com/s/3u1_zg2StchVcterfYoEuQ>

<https://mp.weixin.qq.com/s/GTtw_guNuZHoMUWm0m3x3g>