图上的机器学习系列-聊聊GraphSAGE

原创 AaronLou 享受编程的乐趣 2020-04-12

前言

本篇结合《Inductive Representation Learning on Large Graphs》来聊聊GraphSAGE。这是一种已经在工业界得到广泛采纳的图神经网络方法，具有较好的分布式实施特性，推荐大家关注。

方法定义

GraphSAGE是Graph SAmple and aggreGatE的简称，可以猜想该方法在抽样与聚合上一定有其独到之处。其更正式的定义如下（来自参考资料2）：

> GraphSAGE is a framework for inductive representation learning on large graphs. GraphSAGE is used to generate low-
> dimensional vector representations for nodes, and is especially useful for graphs that have rich node attribute information.

可以看到它本质上开展的是一种Graph Embedding的工作，即将图中的每一个节点最终表示成一个更低维空间中的向量。这与我们此前讲过的DeepWalk、Node2Vec就有了共同话题了。但仔细回想一下我们可以发现，后面两者的算法过程中更多是通过将节点所在的网络拓扑结构信息转化成向量，并保持了节点之间的相似性。而GraphSAGE则明确可以包含节点本身的属性信息，所以这一点上是有所进步的。

同时，注意其定义中有一个词叫“inductive”，这个词很重要，很重要，很重要（说三遍）。就像某些人拥有与众不同、令人印象深刻的特征一样，它让GraphSAGE具有了非常独特的气质，开创了一种新的潮流。

从字面上说，inductive可译为“归纳”。归纳法一般是指人们通过总结历史上的一系列经验，得到了一个普遍的规律，并猜想相同的事物都会遵循这个规律。例如“金导电、银导电、铜导电、铁导电、锡导电；所以一切金属都导电”。在GraphSAGE的算法中，它使用这个术语是用于表达它学习到了一种从节点邻居信息中提炼节点向量化表示的方法，该方法可以适用于新的未曾见过的节点。就仿佛被“授人以渔”了。用论文中的原话来讲：

> Instead of training a distinct embedding vector for each node, we train a set of aggregator functions that learn to aggregate feature information from a node’s local neighborhood.

Pretty cool, right?

与之相对地，如果对于新的节点，还要重新训练一翻才能得到新节点的向量化表示，则称这类方法为“transductive”方法。日后别人聊到Graph Embedding时，你可以先问他一句“你这种方法是inductive的，还是transductive”，保证一秒钟让对方知道你也是行内人士：）

GraphSAGE原理消化

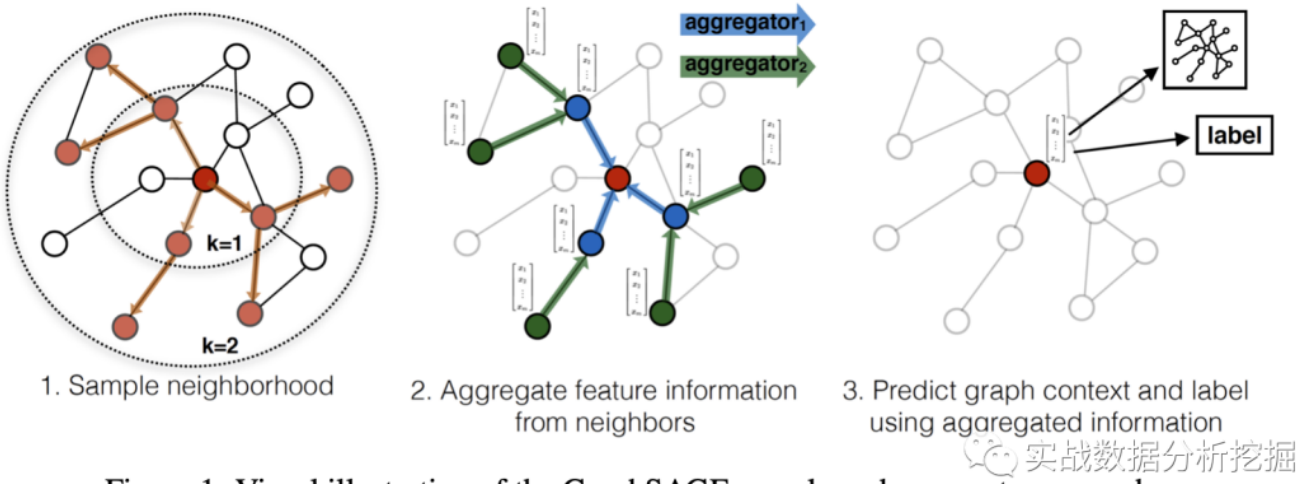

Figure 1: Visual illustration of the GraphSAGE sample and aggregate approach.

这张图可谓浓缩了GraphSAGE的精华，建议深深地印入脑海里。与此同时，我们肯定在脑海中会浮现出几个疑问：

1. 节点的邻域（neighborhood）怎么界定？直接有边相连的？随机游走的？
2. 领域中的信息（包括了节点本身的属性信息，以及网络拓扑结构信息）怎么aggregate起来？
3. 传播、聚合看上去是一个迭代重复进行的过程，那么迭代多少次停止呢？
4. 最终的向量化表达，其中的每个向量元素是怎么来的？

在逻辑上搞清楚了这几个问题，即使暂时不是特别清楚计算机如果实现，也不妨碍我们已经入门了该方法，剩下的无非是理论结合实践多去琢磨就是了。

同时，也把论文中的算法伪代码放进来帮助我们理解算法执行过程。

Algorithm 1: GraphSAGE embedding generation (i.e., forward propagation) algorithm

Input : Graph $\mathcal{G}(\mathcal{V}, \mathcal{E})$; input features $\{\mathbf{x}_v, \forall v \in \mathcal{V}\}$; depth K; weight matrices $\mathbf{W}^k, \forall k \in \{1, ..., K\}$; non-linearity σ; differentiable aggregator functions $\text{AGGREGATE}_k, \forall k \in \{1, ..., K\}$; neighborhood function $\mathcal{N} : v \rightarrow 2^{\mathcal{V}}$

Output: Vector representations $\mathbf{z}_v$ for all $v \in \mathcal{V}$

节点自身的属性在这个时候就传入进来了

$\mathbf{h}_v^0 \leftarrow \mathbf{x}_v, \forall v \in \mathcal{V}$;

for $k = 1...K$ **do**

  for $v \in \mathcal{V}$ **do**

与k有关，所以会有K个这样的聚合函数

    $\mathbf{h}_{\mathcal{N}(v)}^k \leftarrow \text{AGGREGATE}_k(\{\mathbf{h}_u^{k-1}, \forall u \in \mathcal{N}(v)\})$;

    $\mathbf{h}_v^k \leftarrow \sigma\left(\mathbf{W}^k \cdot \text{CONCAT}(\mathbf{h}_v^{k-1}, \mathbf{h}_{\mathcal{N}(v)}^k)\right)$

  end

与SQL中的concat一样，仅仅是把两个向量并起来而已，例如 concat((a,b),(c,d)) = (a,b,c,d)

  $\mathbf{h}_v^k \leftarrow \mathbf{h}_v^k / \|\mathbf{h}_v^k\|_2, \forall v \in \mathcal{V}$

end

$\mathbf{z}_v \leftarrow \mathbf{h}_v^K, \forall v \in \mathcal{V}$

关于第1个问题，论文中是这样表述的：

> we uniformly sample a fixed-
> size set of neighbors, instead of using full neighborhood sets in Algorithm...
> we draw different uniform samples at each iteration, k, in Algorithm 1.

所以，并不是取了每个节点所有邻居来汇总信息，而是指定了邻居个数后随机抽样。可见这也是一个超参数（需要实验者自己进行一翻调参后发现一个比较好的参数值），作者在论文中的实验采用了“neighborhood sample sizes S1 = 25 and S2 = 10”的设定。

关于第2个问题，论文的作者给出了好几种可选的aggregate方法。包括Mean、LSTM、Pooling。通过实验，作者说 LSTM- and pool-based 聚合方法效果更好。

关于第3个问题，这个问题其实等于问K设定多少比较合适。呃，这也是一个超参数，作者在论文中使用了K=2的设定，且效果还不错。

关于第4个问题，我们还记得DeepWalk、Node2Vec中，最终的向量其实来自于Word2Vec算法过程中神经网络隐藏层的权重向量。那么GraphSAGE呢？首先我们看到原始输入中，其实是把每个节点自己的属性信息传入了进来，然后就可以不断计算出每个节点的一个向量化表示。但这其中依赖于W这个权重矩阵，所以还依赖于最终对该参数的最优化。这一个部分的操作与LINE方法类似，通过定义下面这样一个损失函数，再加随机梯度下降的方法来反复更新W，直到损失函数求得极值。

$$J_{\mathcal{G}}(\mathbf{z}_u) = -\log\left(\sigma(\mathbf{z}_u^{\top}\mathbf{z}_v)\right) - Q \cdot \mathbb{E}_{v_n \sim P_n(v)} \log\left(\sigma(-\mathbf{z}_u^{\top}\mathbf{z}_{v_n})\right)$$

于是呢，该方法学习到了aggregrate，以及权重系数W，再来一个新的节点，直接套用计算方法去聚合它的邻居节点信息即可。