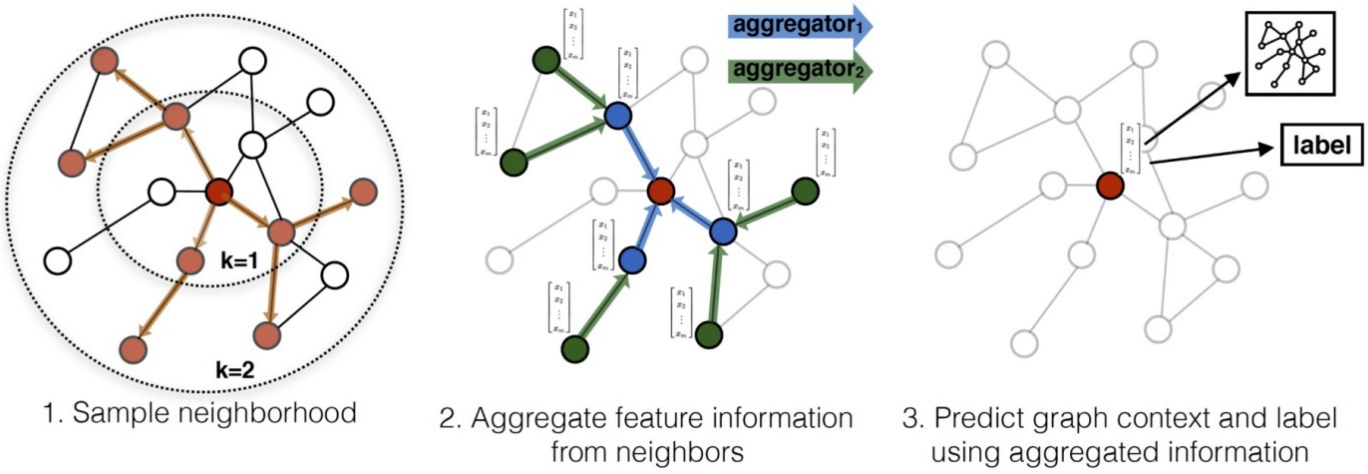
知乎 首发于 浅梦的学习笔记



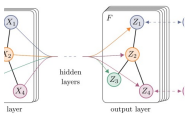
1. Sample neighborhood　2. Aggregate feature information from neighbors　3. Predict graph context and label using aggregated information

【Graph Neural Network】GraphSAGE: 算法原理，实现和应用

浅梦
阿里巴巴 推荐算法搬砖工

在上一篇文章中介绍了GCN

浅梦：【Graph Neural Network】GCN: 算法原理，实现和应用
zhuanlan.zhihu.com



GCN是一种在图中结合拓扑结构和顶点属性信息学习顶点的embedding表示的方法。然而GCN要求在一个确定的图中去学习顶点的embedding，无法直接泛化到在训练过程没有出现过的顶点，即属于一种直推式(transductive)的学习。

本文介绍的GraphSAGE则是一种能够利用顶点的属性信息高效产生未知顶点embedding的一种归纳式(inductive)学习的框架。

其核心思想是通过学习一个对邻居顶点进行聚合表示的函数来产生目标顶点的embedding向量。

GraphSAGE算法原理

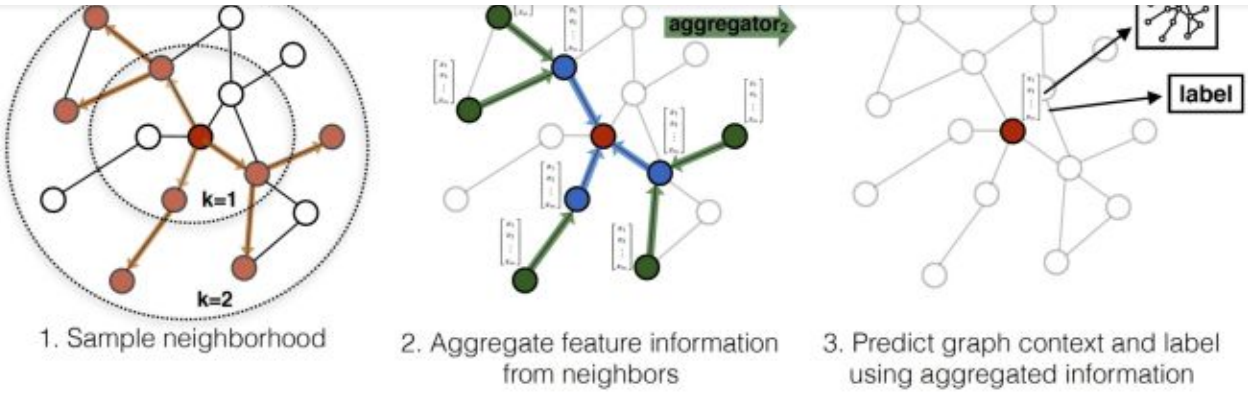


Figure 1: Visual illustration of the GraphSAGE sample and aggregate approach.

GraphSAGE 是Graph SAmple and aggreGatE的缩写，其运行流程如上图所示，可以分为三个步骤

1. 对图中每个顶点邻居顶点进行采样

2. 根据聚合函数聚合邻居顶点蕴含的信息

3. 得到图中各顶点的向量表示供下游任务使用

采样邻居顶点

出于对计算效率的考虑，对每个顶点采样一定数量的邻居顶点作为待聚合信息的顶点。设采样数量为k，若顶点邻居数少于k,则采用有放回的抽样方法，直到采样出k个顶点。若顶点邻居数大于k，则采用无放回的抽样。

当然，若不考虑计算效率，我们完全可以对每个顶点利用其所有的邻居顶点进行信息聚合，这样是信息无损的。

生成向量的伪代码

$\mathbf{W}^k, \forall k \in \{1, ..., K\}$; 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 $v \in \mathcal{V}$

1 $\mathbf{h}_v^0 \leftarrow \mathbf{x}_v, \forall v \in \mathcal{V}$;
2 **for** $k = 1...K$ **do**
3 **for** $v \in \mathcal{V}$ **do**
4 $\mathbf{h}_{\mathcal{N}(v)}^k \leftarrow \text{AGGREGATE}_k(\{\mathbf{h}_u^{k-1}, \forall u \in \mathcal{N}(v)\})$;
5 $\mathbf{h}_v^k \leftarrow \sigma\left(\mathbf{W}^k \cdot \text{CONCAT}(\mathbf{h}_v^{k-1}, \mathbf{h}_{\mathcal{N}(v)}^k)\right)$
6 **end**
7 $\mathbf{h}_v^k \leftarrow \mathbf{h}_v^k / \|\mathbf{h}_v^k\|_2, \forall v \in \mathcal{V}$
8 **end**
9 $\mathbf{z}_v \leftarrow \mathbf{h}_v^K, \forall v \in \mathcal{V}$

这里K是网络的层数，也代表着每个顶点能够聚合的邻接点的跳数，如K=2的时候每个顶点可以最多根据其2跳邻接点的信息学习其自身的embedding表示。

在每一层的循环k中，对每个顶点v，首先使用v的邻接点的k-1层的embedding表示 h_u^{k-1} 来产生其邻居顶点的第k层聚合表示 $h_{N(v)}^k$,之后将 $h_{N(v)}^k$ 和顶点v的第k-1层表示 h_v^{k-1} 进行拼接，经过一个非线性变换产生顶点v的第k层embedding表示 h_v^k 。

聚合函数的选取

由于在图中顶点的邻居是天然无序的，所以我们希望构造出的聚合函数是对称的（即改变输入的顺序，函数的输出结果不变），同时具有较高的表达能力。

• MEAN aggregator

$$h_v^k \leftarrow \sigma(\boldsymbol{W} \cdot \mathbf{MEAN}(\{h_v^{k-1}\} \cup \{h_u^{k-1}, \forall u \in N(v)\})$$

上式对应于伪代码中的第4-5行，直接产生顶点的向量表示，而不是邻居顶点的向量表示。 mean aggregator将目标顶点和邻居顶点的第k-1层向量拼接起来，然后对向量的每个维度进行求均值的操作，将得到的结果做一次非线性变换产生目标顶点的第k层表示向量。

• Pooling aggregator

$$AGGREGATE^{pool}k = \max(\{\sigma(\boldsymbol{W}poolh_{u_i}^k + b), \forall u_i \in N(v)\})$$

Pooling aggregator 先对目标顶点的邻接点表示向量进行一次非线性变换，之后进行一次

• LSTM aggregator

LSTM相比简单的求平均操作具有更强的表达能力，然而由于LSTM函数不是关于输入对称的，所以在使用时需要对顶点的邻居进行一次乱序操作。

参数的学习

在定义好聚合函数之后，接下来就是对函数中的参数进行学习。文章分别介绍了无监督学习和监督学习两种方式。

• 无监督学习形式

基于图的损失函数希望临近的顶点具有相似的向量表示，同时让分离的顶点的表示尽可能区分。目标函数如下

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

其中v是通过固定长度的随机游走出现在u附近的顶点， p_n 是负采样的概率分布， Q 是负样本的数量。

与DeepWalk不同的是，这里的顶点表示向量是通过聚合顶点的邻接点特征产生的，而不是简单的进行一个embedding lookup操作得到。

• 监督学习形式

监督学习形式根据任务的不同直接设置目标函数即可，如最常用的节点分类任务使用交叉熵损失函数。

GraphSAGE的实现

这里以MEAN aggregator简单讲下聚合函数的实现

```
features, node, neighbours = inputs

node_feat = tf.nn.embedding_lookup(features, node)
neigh_feat = tf.nn.embedding_lookup(features, neighbours)
```

```
output = tf.matmul(concat_mean, self.neigh_weights)
if self.use_bias:
    output += self.bias
if self.activation:
    output = self.activation(output)
```

对于第 k 层的aggregator， `features` 为第 $k-1$ 层所有顶点的向量表示矩阵， `node` 和 `neighbours` 分别为第k层采样得到的顶点集合及其对应的邻接点集合。

首先通过 `embedding_lookup` 操作获取得到顶点和邻接点的第 $k-1$ 层的向量表示。然后通过 `concat` 将他们拼接成一个 `(batch_size,1+neighbour_size,embeding_size)` 的张量，使用 `reduce_mean` 对每个维度求均值得到一个 `(batch_size,embedding_size)` 的张量。

最后经过一次非线性变换得到 `output` ，即所有顶点的第 k 层的表示向量

- GraphSAGE 下面是完整的GraphSAGE方法的代码

```
def GraphSAGE(feature_dim, neighbor_num, n_hidden, n_classes, use_bias=True, activatio
              aggregator_type='mean', dropout_rate=0.0, l2_reg=0):

    features = Input(shape=(feature_dim,))
    node_input = Input(shape=(1,), dtype=tf.int32)
    neighbor_input = [Input(shape=(l,),dtype=tf.int32) for l in neighbor_num]

    if aggregator_type == 'mean':
        aggregator = MeanAggregator
    else:
        aggregator = PoolingAggregator

    h = features
    for i in range(0, len(neighbor_num)):
        if i > 0:
            feature_dim = n_hidden
        if i == len(neighbor_num) - 1:
            activation = tf.nn.softmax
            n_hidden = n_classes
        h = aggregator(units=n_hidden, input_dim=feature_dim, activation=activation, l
                       dropout_rate=dropout_rate, neigh_max=neighbor_num[i])(
            [h, node_input,neighbor_input[i]])#
```

```
    return model
```

其中 `feature_dim` 表示顶点属性特征向量的维度， `neighbor_num` 是一个 `list` 表示每一层抽样的邻居顶点的数量， `n_hidden` 为聚合函数内部非线性变换时的参数矩阵的维度， `n_classes` 表示预测的类别的数量， `aggregator_type` 为使用的聚合函数的类别。

GraphSAGE应用

本例中的训练，评测和可视化的完整代码在下面的git仓库中

shenweichen/GraphNeuralNetwork
github.com

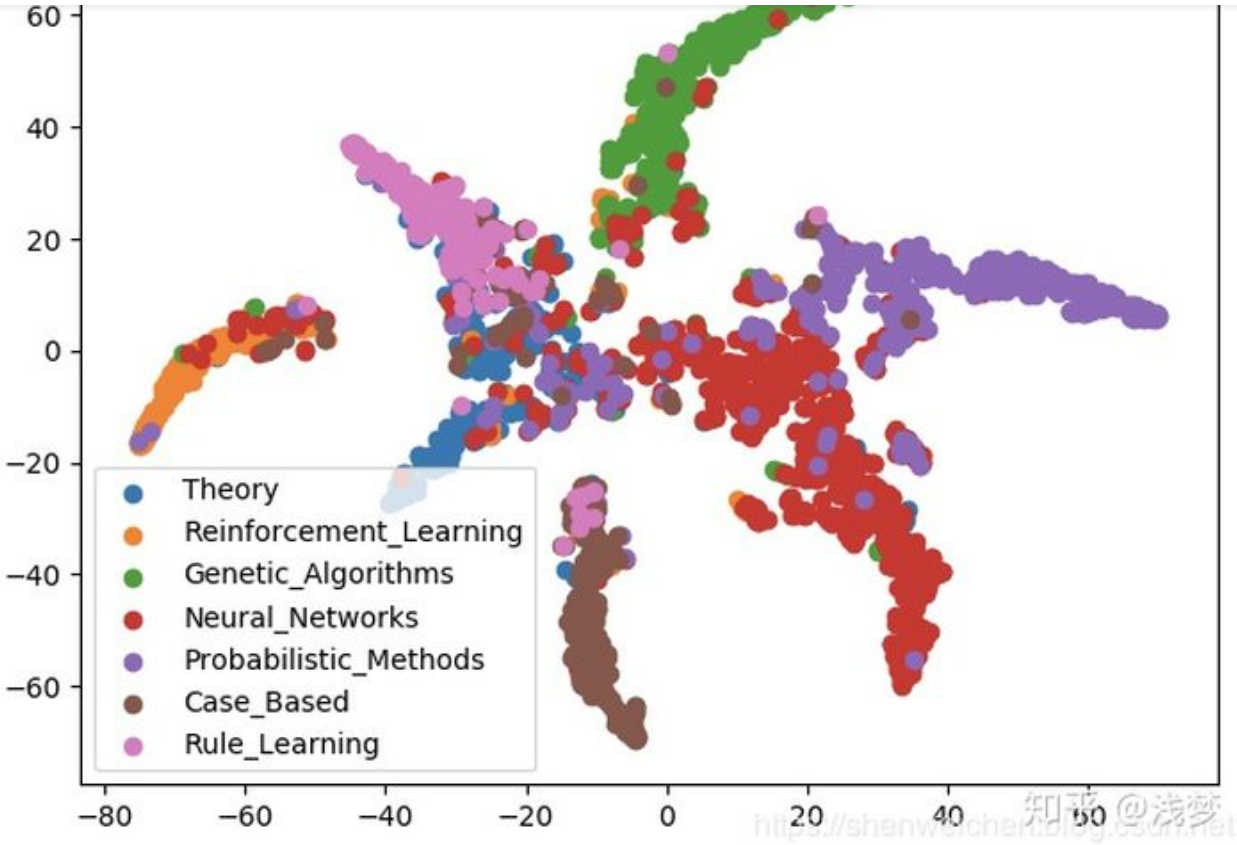
这里我们使用引文网络数据集Cora进行测试，Cora数据集包含2708个顶点, 5429条边,每个顶点包含1433个特征，共有7个类别。

按照论文的设置，从每个类别中选取20个共140个顶点作为训练，500个顶点作为验证集合，1000个顶点作为测试集。 采样时第1层采样10个邻居，第2层采样25个邻居。

- 节点分类任务结果

通过多次运行准确率在0.80-0.82之间。

- 节点向量可视化



参考资料

- Hamilton W, Ying Z, Leskovec J. Inductive representation learning on large graphs[C]//Advances in Neural Information Processing Systems. 2017: 1024-1034. (papers.nips.cc/paper/67...)

为了方便大家学习，我把一些相关的经典文章和代码实现进行了打包汇总并放在了github仓库里，感兴趣的同学可以看看～

https://github.com/shenweichen/AlgoNotes
github.com