

王耳学推荐 | (十) Node2Vec

原创 王耳 sad tom cat 2020-06-24

收录于话题

#王耳学推荐

14个

仍然在看embedding的内容，这周看的是Node2Vec。笔者水平有限，如果文中有什么纰漏或者逻辑错误，还请读者朋友不吝斧正，万分感激。

简易目录

- 介绍Node2Vec
- Node2Vec的部分代码
- 小结

1、介绍Node2Vec

Node2Vec^[1]是Stanford在2016年发表的文章，同样是在图结构上学习结点embedding表示的论文。上一篇稿子提到过的Deep Walk和LINE，在Node2Vec原文里的评价只有一个词：rigid（译为死板的，僵硬的。这个词出现了不止一次）。所以现在好好捋一捋Node2Vec的想法。

1.1、结点关系

Deep Walk僵硬、死板的地方在于，它的训练结果是依靠随机游走产生的序列的质量，没有解释结点间的关系。Node2Vec在此基础上提出结点间的同义性（**homophily**）和同构性（**structural equivalence**）。

根据一张原文里的图，详细解释结点的关系：

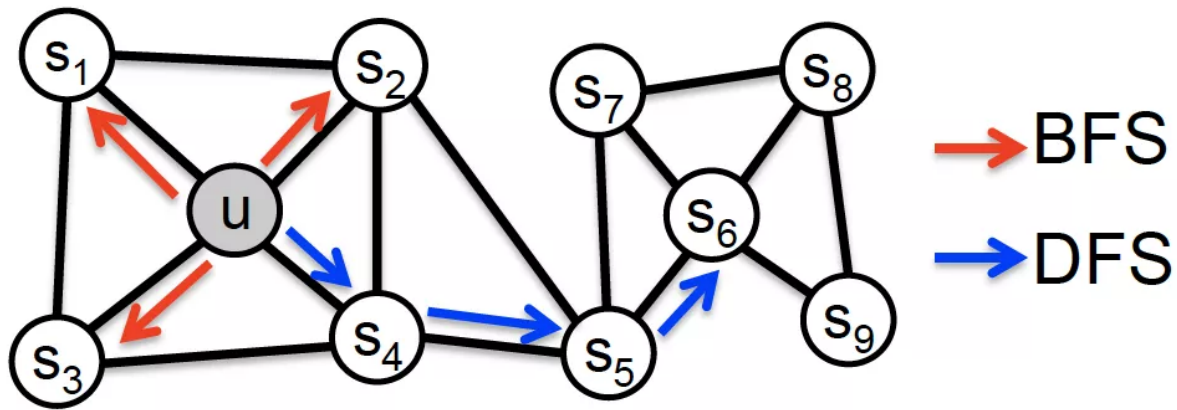


Figure 1: BFS and DFS search strategies from node u ($k_{\text{BFS}} = 3$).

图中的结点大致可以分为两个community: 一个是 u, s_1, s_2, s_3, s_4 组成的, 以 u 为中心; 另一个是有 s_5, s_6, s_7, s_8, s_9 , 以 s_6 为中心。

- **同义性**。同义性强调一个community里结点间的高度相互连接。例如图中的结点 u 和结点 s_1 。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的结构性的作用。例如图中的结点 u 和结点 s_6 。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。

1.2、参数设计

Node2Vec通过对同义性和同构性的定义，区分出结点间的关系。那么在随机游走的过程中，样本侧重不同的结构特点，最终就能获得不同的embedding表达。所以接下来，要约定侧重的“力度”。

BFS和DFS产生的随机游走序列按照一定比例混合在一起，作为训练样本，确实可以影响模型的训练方向。但需要注意的是，在实际情况中，产生的序列并不是这么完美的，每个序列可能都会是BFS和DFS的混合结果。Node2Vec采用了名叫“搜索偏差”（Search bias）的量影响游走方向，记作 α 。

先来一张原文中的图，拿来说明：

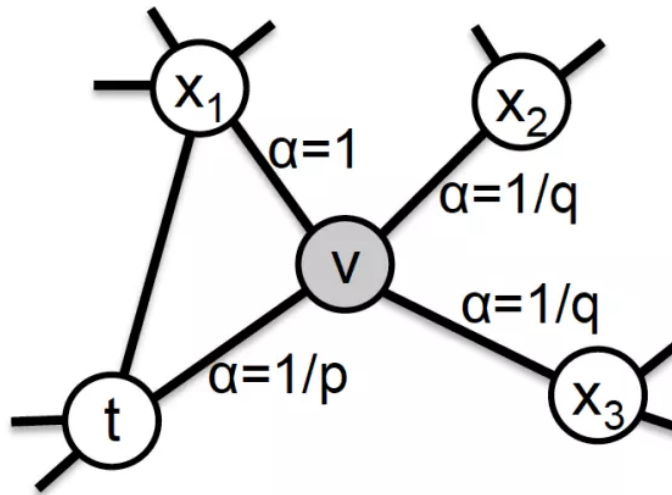


Figure 2: Illustration of the random walk procedure in *node2vec*. The walk just transitioned from t to v and is now evaluating its next step out of node v . Edge labels indicate search biases α

上图是，游走的序已经列从点 t 移动至点 v ，现在点 v 面临的是四个选择， $\{t, x_1, x_2, x_3\}$ 。

接下来是分类讨论。

- 点 t 。若从点 v 重新返回至点 t ，这是一种“重返”行为，由重返参数（Return Parameter） p 控制，此时的搜索偏差 $\alpha = 1/p$ ；点 v 的前后两个点都是点 t ，它们的最短路径长为0。如果 p 越小，游走的序列里“BFS”的成分就越浓。
- 点 x_1 。若从点 v 移动至点 x_1 ，点 x_1 与点 t 也有边连接，它们的最短路径长为1。此时的搜索偏差 $\alpha = 1$ 。
- 点 x_2 和点 x_3 。若从点 v 移动至点 x_2 或者点 x_3 ，点 x_2 、点 x_3 与点 t 都没有边连接，它们的最短路径长为2。此时的搜索偏差 $\alpha = 1/q$ 。 q 被定义为出入参数(In-Out Parameter)。如果 q 越小，游走的序列里“DFS”的成分就越浓。

上述过程在给定当前结点时，要根据前一个结点和后一个结点的最短距离给出搜索偏差；搜索偏差 α 和边权重 w 相乘给出转移概率 π 。

edge (t, v) and now resides at node v (Figure 2). The walk now needs to decide on the next step so it evaluates the transition probabilities π_{vx} on edges (v, x) leading from v . We set the unnormalized transition probability to $\pi_{vx} = \alpha_{pq}(t, x) \cdot w_{vx}$, where

$$\alpha_{pq}(t, x) = \begin{cases} \frac{1}{p} & \text{if } d_{tx} = 0 \\ 1 & \text{if } d_{tx} = 1 \\ \frac{1}{q} & \text{if } d_{tx} = 2 \end{cases}$$

sad torn cat

将转移概率 π 归一化，得到最终的转移概率。

Formally, given a source node u , we simulate a random walk of fixed length l . Let c_i denote the i th node in the walk, starting with $c_0 = u$. Nodes c_i are generated by the following distribution:

$$P(c_i = x \mid c_{i-1} = v) = \begin{cases} \frac{\pi_{vx}}{Z} & \text{if } (v, x) \in E \\ 0 & \text{otherwise} \end{cases}$$

where π_{vx} is the unnormalized transition probability between nodes v and x , and Z is the normalizing constant.

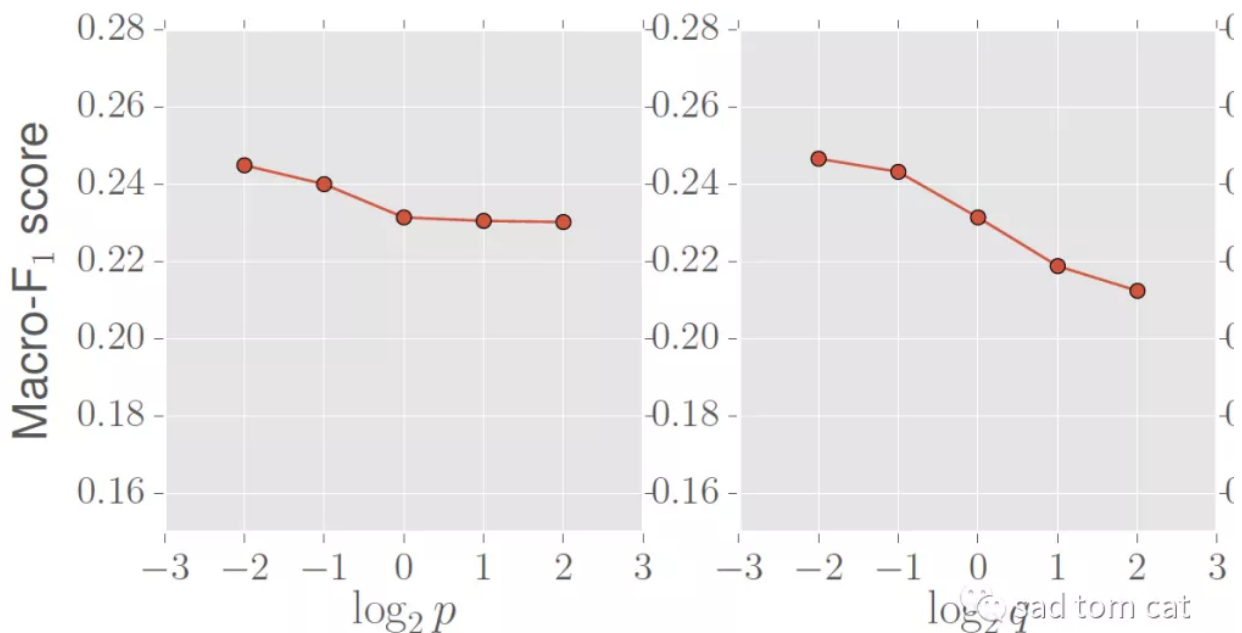
sad torn cat

综上所述，这个转移过程是一个2阶马尔科夫过程。经过若干轮的循环遍历，Node2Vec就产生若干个序列，输入给skip-gram模型，得出结点的Embedding表示。

1.3、讨论模型

参数敏感度

笔者只截取了 p 和 q 的参数敏感度比较，如下图所示：



在公开数据集BlogCatalog上，以 $Macro - F_1 score$ (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 p 增大，还是 q 增大，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。

那为什么没有一个折中的参数组合呢？可惜的是原文也没有给出 p 和 q 组合的网格搜索结果。感兴趣的读者朋友可以试试。

可扩展性

此处讨论的是结点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。

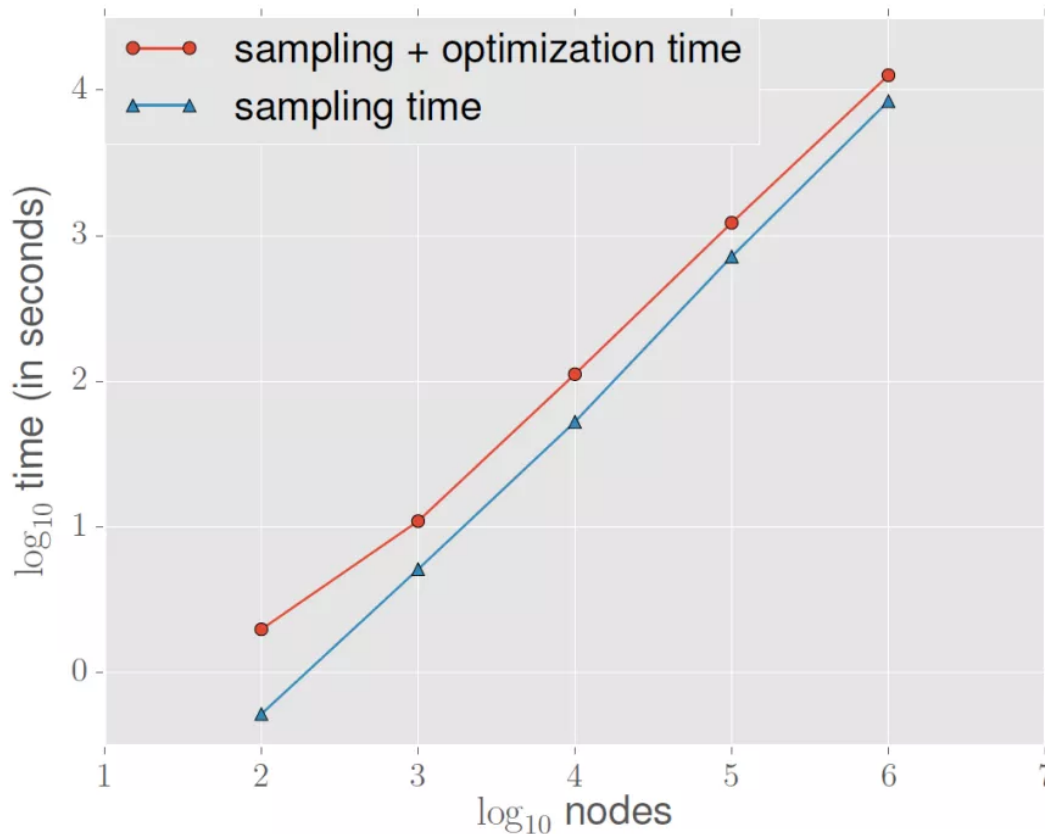


Figure 6: Scalability of *node2vec* on Erdos-Renyi graphs with an average degree of 10.

sad torn cat

Emmmmmmm...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，给有意使用Node2Vec的人心里有个底。


2、Node2Vec的部分代码

Node2Vec的伪代码：

Algorithm 1 The *node2vec* algorithm.

LearnFeatures (Graph $G = (V, E, W)$, Dimensions d , Walks per node r , Walk length l , Context size k , Return p , In-out q) $\pi = \text{PreprocessModifiedWeights}(G, p, q)$ $G' = (V, E, \pi)$ Initialize *walks* to Empty**for** $iter = 1$ **to** r **do** **for all** nodes $u \in V$ **do** $walk = \text{node2vecWalk}(G', u, l)$ Append *walk* to *walks* $f = \text{StochasticGradientDescent}(k, d, walks)$ **return** f

node2vecWalk (Graph $G' = (V, E, \pi)$, Start node u , Length l)Initialize *walk* to $[u]$ **for** $walk_iter = 1$ **to** l **do** $curr = walk[-1]$ $V_{curr} = \text{GetNeighbors}(curr, G')$ $s = \text{AliasSample}(V_{curr}, \pi)$ Append s to *walk***return** *walk*

 sad torn cat

相对应地写了python代码，加入了写注释：

```

from joblib import Parallel, delayed
from collections import defaultdict, OrderedDict
import numpy as np

class Node2Vec:
    def __init__(self, data_cate_path, data_link_path, walk_per_vertex,
                 walk_length, p, q, n_workers=1, verbose=50, **kwargs):

        self.vertex, self.edge = read_graph_data(data_cate_path, data_link_path)
        self.walk_per_vertex = walk_per_vertex          # r defined in paper
        self.walk_length = walk_length                  # l defined in paper
        self.inverse_p = 1.0 / p                        # p defined in paper
        self.inverse_q = 1.0 / q                        # q defined in paper
        self.vertex_list = sum(list(self.vertex.values()), [])

```

```

self.process_modified_weights()
self.run(walk_per_vertex=walk_per_vertex, n_workers=n_workers, verbose=verbose)

def process_modified_weights(self):
    self.second_markov_weight_dict = defaultdict(dict) # pi defined in paper
    for pre_one in self.vertex_list:
        for curr_one in self.edge[pre_one]:
            weight_dict = OrderedDict()
            for next_one in self.edge[curr_one]:
                if next_one not in self.edge[pre_one]:
                    weight_dict[next_one] = self.inverse_q # distance = 2
                elif next_one == pre_one:
                    weight_dict[next_one] = self.inverse_p # distance = 0
                else:
                    weight_dict[next_one] = 1 # distance = 1

            item_list = list(weight_dict.keys())
            value_list = list(weight_dict.values())
            value_list = list(map(lambda x:x/sum(value_list), value_list))

            self.second_markov_weight_dict[pre_one][curr_one] = (item_list, value_list)

def node2vec(self, walk_per_vertex, walk_length):
    res = []
    for _ in range(walk_per_vertex):
        for start_node in self.vertex_list:
            walk_path = [start_node]
            if not self.edge[start_node]:
                continue
            else:
                walk_path.append(np.random.choice(self.edge[start_node]))

            for _ in range(2, walk_length):
                pre_one, curr_one = walk_path[-2], walk_path[-1]
                try:
                    tmp = np.random.choice(self.second_markov_weight_dict[pre_one][curr_one][0])
                    p=self.second_markov_weight_dict[pre_one][curr_one][1]
                    walk_path.append(tmp)
                except:
                    break
            res.append(walk_path)
    return res

def run(self, walk_per_vertex, n_workers=4, verbose=50):

```

```
# parallel
res = Parallel(n_jobs=n_workers, verbose=verbose)(
    delayed(self.node2vec)(num, self.walk_length)
    for num in partition_num(walk_per_vertex, n_workers))
self.walks_generated = sum(res, [])
```

其中的辅助函数与上次DeepWalk时相同，就不赘述了。

当然，代码也有偷懒的部分，默认把边权重都看作是1了。如果需要权重，可以在函数process_modified_weights里加入权重变量。

小结

Node2Vec的精彩之处，在于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。原文也给出了Node2Vec与其他方法的对比结果，感兴趣的读者朋友可以看看原文。

参考资料

- [1] node2vec: Scalable Feature Learning for Networks: <https://link.zhihu.com/?target=https%3A//www.kdd.org/kdd2016/papers/files/rfp0218-groverA.pdf>,

喜欢此内容的人还喜欢

南京传媒学院2021年全仿真考试演练开通在即，点击获取攻略！

南京传媒学院招生办

85岁老教授在农村“种文化” | 道德的力量

中国文明网