Node2Vec:如何成为TA最好的朋友

原创 penny本妮 penny菜鸟学cs 2020-04-20

来deepwalk一下我的社交网络

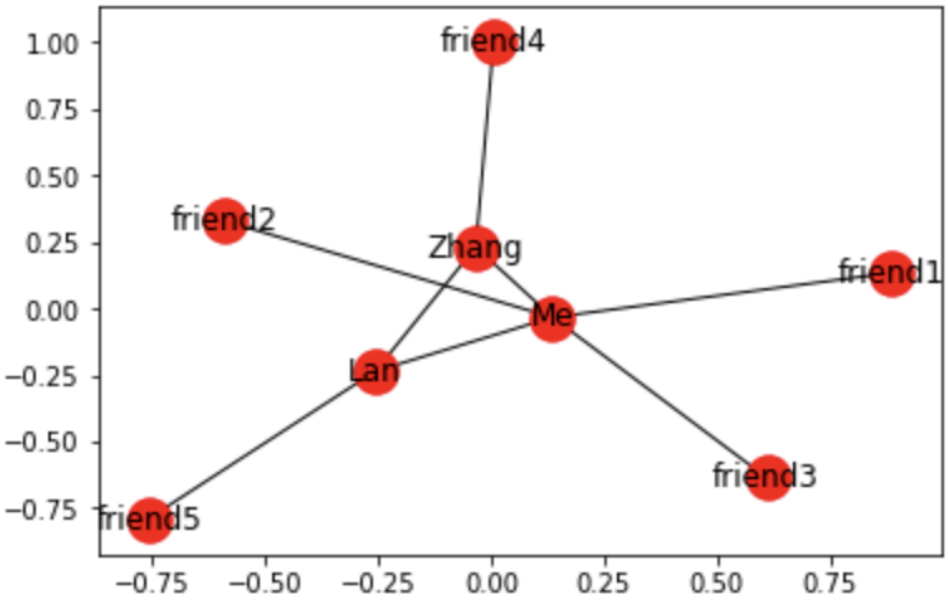
在之前的这篇文章里，我用deepwalk做了一下我的社交小网络的嵌入，最后发现，我辛苦码了这么久的字和代码，而我竟然自己不是小张最好的朋友:(

然后评论有人指出我的社交网络里，我的节点出度最大，只有1/5的概率游走到小张，而小蓝有1/3的概率游走到小张。且他们在网络中的结构还很相似，所以用Word2Vec训练出来，肯定是她俩的相似性更高！

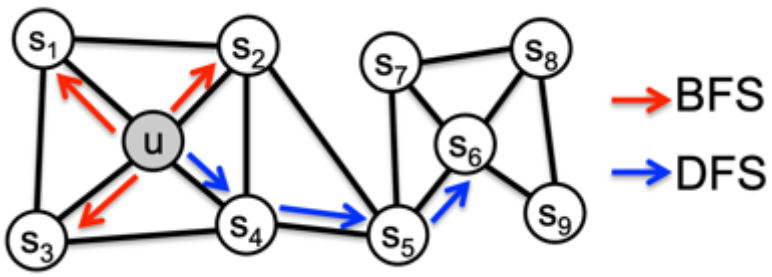
但事实上，这不应该呀，毕竟我独享小张的喜爱！所以如何成为小张最好的朋友呢？那就**加大权重**，**加大权重**，**加大权重**！

我重新修改了权重，把我和小张之间的权重改到了最大。

```
import networkx as nx
import gensim
import numpy as np
import matplotlib.pyplot as plt
node_list=['friend1','friend2','friend3','Me','Zhang','Lan','friend4','friend5']
edge_list=[('friend1','Me',1),('friend2','Me',1),('friend3','Me',1),('Zhang','Me',5.20)
,('Lan','Me',2.50),('friend4','Zhang',1),('friend5','Lan',1),('Zhang','Lan',2.50)]
#创建空图
G=nx.Graph()
#从一个列表中添加节点
G.add_nodes_from(node_list)
#根据（边，边，权重）加载
G.add_weighted_edges_from(edge_list)
#plot
nx.draw_networkx(G,node_list=G.nodes(),edges=G.edges())
```

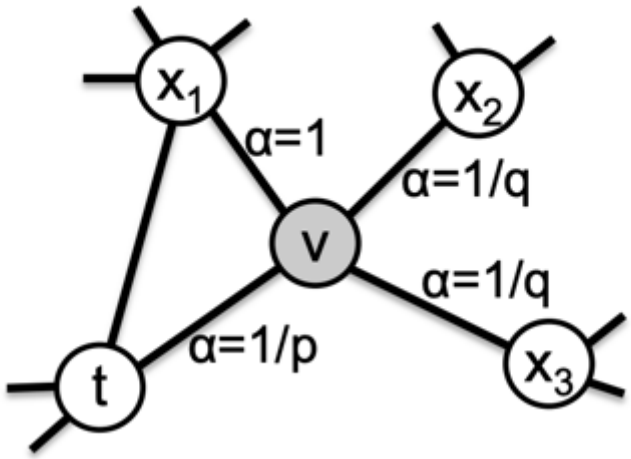


但deepwalk提出时为了简单没有对权重操作呀，而且deepwalk的随机游走方式类似DFS（深度优先搜索），很有可能沿着一条路径就一直走下去了。但网络中显然邻居节点和节点之间的关系也尤为重要，那应该也要一定程度的BFS（广度优先搜索）才行。所以后来的Node2Vec中基于这两点，提出了修改：1.同时结合BFS和DFS，改善游走方式。2.加入对带权图的操作。



（BFS和DFS）

已知DeepWalk里对下一个节点的采样就是直接从该节点的邻居中随机选择一个就好了，但Node2Vec中同时考虑BFS和DFS，文章提出了如下采样方式：



如图所示，假设已经采样了(t,v)两个节点了，此时对于采样v的下一个节点，不同的位置有一个不同的权重设置：

1. 这个节点既与v相连也与t相连，此时自身权重再乘上alpha=1。（这代表了对t的BFS）
2. 这个节点与v相连而不与t相连，此时自身权重在乘上alpha=1/q。（这代表了对t的DFS）
3. 从v回到t，此时v到t自身的权重乘上alpha=1/p。

p为返回参数，q为出入参数。

若p>max(q,1)，则采样会尽量不往回走。若小于，则倾向于在这个节点附近来回地走。
若q>1，则代表游走会倾向于BFS。
若q<1，则代表游走会倾向于DFS。
当p=1,q=1时，则等同于DeepWalk的随机游走。

代码如下：

```
def get_alias_edge(G,t,v,p,q):
    '''t:previous node v:current node'''
    unnormalized_probs=[]
    for x in sorted(G.neighbors(v)):
        weight=G.get_edge_data(x,v)['weight']
        if x==t:  #回走
            unnormalized_probs.append(weight/p)
        elif G.has_edge(x,t): #BFS
            unnormalized_probs.append(weight)
        else: #DFS
            unnormalized_probs.append(weight/q)
    norm=sum(unnormalized_probs)
    normalized_probs=list(map(lambda x:x/norm,unnormalized_probs))
    return create_alias_table(normalized_probs)
```

改善了游走方式以后，就要考虑如何对带权的边进行采样。这里使用了AliasSample的采样方式。

可以参考这篇博客：https://www.cnblogs.com/arachis/p/alias_sample.html

简述一下AliasSample就是：

假如随机事件分别出现的概率是[1/2，1/3，1/12，1/12]，此时乘上事件数N（这里N=4），然后会发现把这个概率*事件数的直方图画出来，面积为N。

那么Alias Sample想要做的就是，把面积大于1的部分填充到小于1的部分，并且最后每一列只有最多两个事件。如图所示：

考虑到这样做的解可能并不唯一，所以代码实现时，找到一个可行解就行。

代码实现的过程如下，small和large分别存放面积大于1和小于1的事件下标。

accept存放第i列对应的事件i矩形的面积百分比，alias存放第i列除了第i个事件的另一个事件下标。

每次循环中把large中大于1的事件pop出来一个，small中小于1的事件也pop出来一个，用large的面积填充small的面积，保证small的面积为1。此时large的面积若>1，则继续放回large中，若小于1则放入small中，接着循环操作。

这样一定可以产生一个可行解。

生成accept和alias以后，进行采样。随机采样1-N之间的整数i，决定落在哪一列。随机采样0-1之间的一个概率值，如果小于accept[i]，则采样i，若大于则采样alias[i]。

代码如下：

```
#通过alias sample创建alias table
import numpy as np
def create_alias_table(area_ratio):
    '''
    area_ratio[i]代表
    '''
    N=len(area_ratio)
    accept,alias=[0]*N,[0]*N
    small,large=[],[]
    area_ratio=np.array(area_ratio)*N
    for i,prob in enumerate(area_ratio):
        if prob<1:
            small.append(i)
        else:
            large.append(i)
    while large and small:
        small_idx,large_idx=small.pop(),large.pop()
        accept[small_idx]=area_ratio[small_idx]
        alias[small_idx]=large_idx
        area_ratio[large_idx]=area_ratio[large_idx]-(1-alias[small_idx])
        if area_ratio[large_idx]<1:
            small.append(large_idx)
        else:
            large.append(large_idx)

    while large:
        large_idx=large.pop()
        accept[large_idx]=1
    while small:
        small_idx=small.pop()
        accept[small_idx]=1

    return accept,alias
```

```
def alias_sample(accept,alias):
    N=len(accept)
    i=int(np.floor(np.random.rand()*N))
    r=np.random.rand()
    if r<accept[i]:
        return i
    else:
        return alias[i]
```

那么定义好Node2Vec中最重要的两部分后，我要来生成我的“语料库了”。
首先我要生成每个节点在转移过程中的转移概率，然后生成alias table。

```
def preprocess_transition_probs(G,p,q):
    alias_nodes={}
    for node in sorted(G.nodes()):
        unnormalized_probs=[G.get_edge_data(node,nbr)['weight'] for nbr in sorted(G.neighbors(node))]
        norm=sum(unnormalized_probs)
        normalized_probs=list(map(lambda x:x/norm,unnormalized_probs))
        alias_nodes[node]=create_alias_table(normalized_probs)
    alias_edges={}
    for edge in list(G.edges()):
        alias_edges[edge]=get_alias_edge(G,edge[0],edge[1],p,q)
        alias_edges[(edge[1],edge[0])]=get_alias_edge(G,edge[1],edge[0],p,q)
    return alias_nodes,alias_edges
```

```
alias_nodes,alias_edges=preprocess_transition_probs(G,2,0.8)
alias_nodes
```

```
{'Lan': ([1, 1, 0.5], [0, 0, 1]),
 'Me': ([1, 1, 0.4672897196261683, 0.4672897196261683, 0.4672897196261683],
  [0, 0, 1, 1, 1]),
 'Zhang': ([0.8620689655172415, 1, 0.3448275862068966], [1, 0, 1]),
 'friend1': ([1], [0]),
 'friend2': ([1], [0]),
 'friend3': ([1], [0]),
 'friend4': ([1], [0]),
 'friend5': ([1], [0])}
```

```
alias_edges
```

```
{('friend1',
  'Me'): ([1,
   1,
   0.19801980198019803,
   0.49504950495049505,
   0.49504950495049505], [0, 0, 1, 1, 1]),
 ('Me', 'friend1'): ([1], [0]),
 ('friend2',
  'Me'): ([1,
   1,
   0.49504950495049505,
   0.19801980198019803,
   0.49504950495049505], [0, 0, 1, 1, 1]),
 ('Me', 'friend2'): ([1], [0]),
 ('friend3',
  'Me'): ([1,
   1,
   0.49504950495049505,
   0.49504950495049505,
   0.19801980198019803], [0, 0, 1, 1, 1]),
 ('Me', 'friend3'): ([1], [0]),
```

进行alias sample，决定下一个走向哪个节点，生成一次游走：

```
def node2vec_walk(G,walk_length,start_node,alias_nodes,alias_edges):
    walk=[start_node]
    while len(walk)<walk_length:
        cur=walk[-1]
        cur_nbrs=sorted(G.neighbors(cur))
        if len(cur_nbrs)>0:
            if len(walk)==1:
                walk.append(cur_nbrs[alias_sample(alias_nodes[cur][0],alias_nodes[cur][1])])
            else:
                prev=walk[-2]
                next=cur_nbrs[alias_sample(alias_edges[(prev,cur)][0],alias_edges[(prev,cur)][1])]
                walk.append(next)
        else:
            break
    return walk
```

```
node2vec_walk(G,10,'Me',alias_nodes,alias_edges)
```

```
['Me', 'Zhang', 'Me', 'Lan', 'Zhang', 'Me', 'friend3', 'Me', 'friend3', 'Me']
```

最后多次游走，生成"语料库"：

```
import random
def simulate_walks(G,num_walks,walk_length,alias_nodes,alias_edges,rand=random.Random()):
    walks=[]
    nodes=list(G.nodes())
    for i in range(num_walks):
        rand.shuffle(nodes)
        for node in nodes:
            walks.append(node2vec_walk(G,walk_length,node,alias_nodes,alias_edges))
    return walks
```

```
walks=simulate_walks(G,10,10,alias_nodes,alias_edges,rand=random.Random())
```

```
from gensim.models import Word2Vec
model = Word2Vec(walks, size=20, window=2, min_count=0, sg=1,iter=20)
model.wv['Zhang']
```

```
array([-0.13974336, -0.00535731, -0.07112405, -0.12068465, -0.04678886,
       -0.04699158,  0.13902713, -0.13237877, -0.20016567,  0.14598444,
       -0.03026583,  0.04862763,  0.03326393, -0.0760393 , -0.00311179,
        0.06341337,  0.02879526, -0.01971069,  0.03543255, -0.05967962],
      dtype=float32)
```

让我来看看此时小张最好的朋友是谁了！

```
model.wv.most_similar('Zhang',topn=1)
```

```
[('Me', 0.9778188467025757)]
```

of course，是我了！！！

参考论文：

DeepWalk: Online Learning of Social Representations https://arxiv.org/abs/1403.6652

Node2Vec : Scalable Feature Learning for Networks https://cs.stanford.edu/~jure/pubs/node2vec-kdd16.pdf

LINE: Large-scale Information Network Embedding https://arxiv.org/abs/1503.03578

（这里就不写LINE了）

（开心到把用jupyter notebook写的DeepWalk/Node2Vec/LINE放到GitHub上，一起玩吧:），点击左下方链接即可。
https://github.com/ninoxjy/graph-embedding)

阅读原文