机器学习实验报告3

Louvain

Fast unfolding of communities in large networks指的就是Louvain算法。

首先将每个节点作为自己的社区标签，每个节点遍历所有的邻居节点，将邻居的社区标签赋值到自己的社区标签，计算模块度增量，选择模块度增量最大的社区标签作为自己的社区标签。不断地循环此步骤，直到每个节点都不可以通过改变自己的社区标签增加模块度。

```
#根据模块度，对所有节点进行优化
def first_step(self):
    stop = True    #用于判断循环是否结束
    random_vid = self._G.keys()
    random.shuffle(list(random_vid))    #随机访问节点

    while True:
        flag = True
        #遍历所有节点
        for i_vid in random_vid:
            i_cid = self._vid_array[i_vid]._cid     #获取该节点的社区编号
            sum_k = sum(self._G[i_vid].values()) + self._vid_array[i_vid]._kin
  #计算该节点的内外边权重之和
            cid_Q = {}       #存储所有模块度增益大于0的社区编号

            #遍历所有与该节点相邻的节点
            for j_vid in self._G[i_vid].keys():
                #如果相邻节点所属的社区的模块度增益大于0，则无需处理
                j_cid = self._vid_array[j_vid]._cid
                if j_cid in cid_Q:
                    continue
                else:
                    #计算该社区的模块度增益
                    tot = sum([sum(self._G[k].values()) +
self._vid_array[k]._kin for k in self._cid_array[j_cid]])
                    if j_cid == i_cid:
                        tot -= sum_k
                    k_v_in = sum([v for k, v in self._G[i_vid].items() if k in
self._cid_array[j_cid]])
                    Q = k_v_in - sum_k * tot / self._edge_num
                    cid_Q[j_cid] = Q

            #获取模块度增益最大的社区编号
            max_cid, max_Q = sorted(cid_Q.items(), key = lambda item: item[1],
reverse = True)[0]
            #模块度增益仍大于0，更改该节点的社区编号，并且继续迭代，直到模块度不再改变
            if max_Q > 0.0 and max_cid != i_cid:
                self._vid_array[i_vid]._cid = max_cid       #更改该节点的社区编号

                self._cid_array[max_cid].add(i_vid)         #新社区添加该节点
                self._cid_array[i_cid].remove(i_vid)    #旧社区去除该节点
                flag = False
                stop = False
```

```
        if flag:
            break

    return stop
```

完成所有节点的模块度优化后，将一个社区里的所有节点合并为一个大节点，每个社区就是一个大节点，根据大节点的内部节点，计算每个大节点之间的边权重，还有每个大节点内部的边权重，形成一个新的图。

```
#每个社区合并为一个新的大节点，大节点的边权重为原始社区内所有节点的边权重之和，分配新的社区
def second_step(self):
    new_vid_array = {}
    new_cid_array = {}
    #遍历所有社区和社区内的节点，更新每个社区内部的边的权重
    for cid, vertexs in self._cid_array.items():
        #如果该社区为空，则跳过
        if len(vertexs) == 0:
            continue
        #创建一个大节点，大节点的节点编号为该社区的社区编号
        big_vertex = Vertex(cid, cid, set())
        #将该社区内的所有节点合并为一个大节点
        for vid in vertexs:
            big_vertex._nodes.update(self._vid_array[vid]._nodes)    #将社区内所有节点添加进该大节点
            big_vertex._kin += self._vid_array[vid]._kin         #计算大节点的内部的边的权重，为社区内所有小节点的和
            #遍历vid的所有邻居，如果邻居也在该社区之内，则添加
            for k, v in self._G[vid].items():
                if k in vertexs:
                    big_vertex._kin += v / 2.0
        new_cid_array[cid] = {cid}           #初始化新的社区编号为当前的社区编号
        new_vid_array[cid] = big_vertex      #将新的大节点添加进该社区

    #根据新的节点和社区，创建新图
    new_G = collections.defaultdict(dict)
    #遍历所有社区，计算所有社区之间的边的权重
    for cid1, vertexs1 in self._cid_array.items():
        if len(vertexs1) == 0:
            continue
        for cid2, vertexs2 in self._cid_array.items():
            #避免冗余计算
            if cid2 <= cid1 or len(vertexs2) == 0:
                continue
            edge_weight = 0.0
            #遍历cid1社区中的节点
            for vid in vertexs1:
                #遍历该节点在cid2社区的邻居，计算cid1和cid2两个社区的边的权重
                for k, v in self._G[vid].items():
                    if k in vertexs2:
                        edge_weight += v
            #更新两个社区之间边的权重
            if edge_weight != 0:
                new_G[cid1][cid2] = edge_weight
                new_G[cid2][cid1] = edge_weight

    #更新数据
    self._cid_array = new_cid_array
```

```
        self._vid_array = new_vid_array
        self._G = new_G
```

不断重复以上两个步骤，直到模块度不再改变。

```
#执行社区发现
def execute(self):
    #不停重复两个步骤，直到模块度不再改变
    while True:
        stop = self.first_step()
        if stop:
            break
        else:
            self.second_step()
    return self.get_communities()
```

数据处理

数据来源：https://snap.stanford.edu/data/ego-Facebook.html

circle文件中，存储了节点的正确社区类标，读取正确类标，与使用louvain算法获得社区类标使用NMI进行比较

```
#获取正确的社区分类
correct_communities = np.zeros(10000)
count = 0       #计算原来一共有多少个社区
file = open("data/facebook1_circle.txt")
line = file.readline().replace('\n','')
while line:
    node = line.split(' ')
    count += len(node) - 1
    for i in range(1, len(node)):
        correct_communities[int(node[i])] = int(node[0])
    line = file.readline().replace('\n','')
file.close()

#获取完成社区发现后新的社区分类
new_communities = np.zeros(10000)
label = 1
for communitie in communities:
    for i in range(len(communitie)):
        new_communities[communitie[i]] = label
    label += 1

#计算NMI
print("NMI1:", NMI(correct_communities, new_communities, count))
```

使用community库所提供的标准louvain社区发现函数，对数据进行处理，实验结果与自己实现的louvain进行比较

```python
#使用community库所提供的标准louvain社区发现函数，对数据进行处理
G = Graph().createGraph(path)
partition = community_louvain.best_partition(G)

#获取标准社区发现的社区分类
best_communities = np.zeros(10000)
for node in partition:
    best_communities[node] = partition[node] + 1

#计算NMI
print("NMI2:", NMI(correct_communities, best_communities, count))
```

实验结果

data1:

```
PS D:\vscode workspace\python\louvain> python my_louvain.py
NMI1: 0.31626327595524883
NMI2: 0.32322848495216555
PS D:\vscode workspace\python\louvain> python my_louvain.py
NMI1: 0.31626327595524883
NMI2: 0.31500192886052597
PS D:\vscode workspace\python\louvain> python my_louvain.py
NMI1: 0.31626327595524883
NMI2: 0.3449401557660151
PS D:\vscode workspace\python\louvain> python my_louvain.py
NMI1: 0.31626327595524883
NMI2: 0.32157988647390734
PS D:\vscode workspace\python\louvain> python my_louvain.py
NMI1: 0.31626327595524883
NMI2: 0.3227097064587836
```

data2:

```
PS D:\vscode workspace\python\louvain> python my_louvain.py
NMI1: 0.5288311700000999
NMI2: 0.5650117347051954
PS D:\vscode workspace\python\louvain> python my_louvain.py
NMI1: 0.5288311700000999
NMI2: 0.5650117347051955
PS D:\vscode workspace\python\louvain> python my_louvain.py
NMI1: 0.5288311700000999
NMI2: 0.540562710190459
PS D:\vscode workspace\python\louvain> python my_louvain.py
NMI1: 0.5288311700000999
NMI2: 0.5500739634346037
PS D:\vscode workspace\python\louvain> python my_louvain.py
NMI1: 0.5288311700000999
NMI2: 0.5306440938144 25
```

data3:

```
PS D:\vscode workspace\python\louvain> python my_louvain.py
NMI1: 0.5727012242729664
NMI2: 0.6398556774965807
PS D:\vscode workspace\python\louvain> python my_louvain.py
NMI1: 0.5727012242729664
NMI2: 0.6619039338353724
PS D:\vscode workspace\python\louvain> python my_louvain.py
NMI1: 0.5727012242729664
NMI2: 0.6607372125754574
PS D:\vscode workspace\python\louvain> python my_louvain.py
NMI1: 0.5727012242729664
NMI2: 0.6488954060528402
PS D:\vscode workspace\python\louvain> python my_louvain.py
NMI1: 0.5727012242729664
NMI2: 0.6424502364872211
```

data4:

```
PS D:\vscode workspace\python\louvain> python my_louvain.py
NMI1: 0.6842069202325065
NMI2: 0.677661506418292
PS D:\vscode workspace\python\louvain> python my_louvain.py
NMI1: 0.6842069202325065
NMI2: 0.7019918975315499
PS D:\vscode workspace\python\louvain> python my_louvain.py
NMI1: 0.6842069202325065
NMI2: 0.6793248865303169
PS D:\vscode workspace\python\louvain> python my_louvain.py
NMI1: 0.6842069202325065
NMI2: 0.6610229057046595
PS D:\vscode workspace\python\louvain> python my_louvain.py
NMI1: 0.6842069202325065
NMI2: 0.6662953117854078
```

data5:

```
PS D:\vscode workspace\python\louvain> python my_louvain.py
NMI1: 0.6281048505907763
NMI2: 0.6442324188301101
PS D:\vscode workspace\python\louvain> python my_louvain.py
NMI1: 0.6281048505907763
NMI2: 0.6407535371656481
PS D:\vscode workspace\python\louvain> python my_louvain.py
NMI1: 0.6281048505907763
NMI2: 0.6431020803211368
PS D:\vscode workspace\python\louvain> python my_louvain.py
NMI1: 0.6281048505907763
NMI2: 0.6356589055779175
PS D:\vscode workspace\python\louvain> python my_louvain.py
NMI1: 0.6281048505907763
NMI2: 0.646646416430122
```

实验分析：

从实验结果可以看到，NMI2大多数都是大于NMI1的，说明标准louvain社区发现函数比我所实现的louvain的分类效果更好，但差距不大，大多数情况都只有0.01~0.04的差距，这种差距可能是在部分极端情况的处理没有标准函数所处理的好。

louvain算法的时间复杂度低，适合大规模的网络，社区划分结果稳定，消除了模块化分辨率的限制。但在社区过大时，可能不能及时收敛，很容易将一些外围的点加入到原本紧凑的社区中，导致一些错误的合并，这种划分有时候在局部视角是优的，但是全局视角下会变成劣的。