✅

社交网络第三方库

STEP 1 数据预处理

数据集分为两个部分，vertices 是节点，edges 是边，所以将 newmovies.txt 划分为 vertices.txt 和 edges.txt 两个数据集。通过指定行数 +Python 读写文件操作来实现。处理之后的数据保存到 GraphStat 文件夹下。

```python
def copy_part(copied_path, copying_path, start, end):
    '''
    :param copied: 被复制的文件
    :param copying: 复制到的文件
    :param start: 开始的某一行
    :param end: 结束的某一行
    '''
    i = 0
    while True:
        line = f1.readline()
        i += 1
        if i > start and i <= end:
            f2.write(line)
        if i > end:
            break

if __name__ == "__main__":
    f1 = open(r'../newmovies.txt', 'rb')
    f2 = open(r'../edges.txt', 'ab')
    copy_part(f1, f2, 34285, 176712)
```

Python

Ref.

[280]python取txt文件的若干行到另一个文件_周小董-CSDN博客

取movie.txt文件的若干行到movie2.txt#取txt文件 的若干行到另一个txtf1 = open(r'F:\movie.txt','rb')f2= open(r'F:\movie2.txt','ab')i=0while True: line = f1.readline() i+=1 if i&amp;amp;gt;100 and i&amp;amp;lt;150:

blog.csdn.net

STEP 2 构建 NetworkBuilder 模块

STEP 2.1 node.py

STEP 2.1.1 初始化节点

```python
def init_node(id):
    '''
    :param id: 输入需要的ID号
    :return: 输出该节点属性字典
    '''
    sr = vertices.iloc[id]
    node_dict = {}
    node_dict['ID'] = sr[0]
    node_dict['Name'] = sr[1]
    node_dict['Weight'] = sr[2]
    node_dict['Type'] = sr[3]
    node_dict['Info'] = sr[4]
    return node_dict
```

Python

得到：

```
{'ID': 0, 'Name': 'Ann Blyth', 'Weight': 6035, 'Type': 'starring', 'Info': '1928 births;Living people;American film actors;American musical theatre actors;American child actors;People from Westchester County, New York;'}
```

STEP 2.1.2 将节点的属性进行格式化输出

```python
def print_node(id):
    node_dict = init_node(id)
    print('ID: {0[ID]}\nName: {0[Name]}\nWeight: {0[Weight]}\nType:{0[Type]}\nInfo:{0[Info]}'.format(node_dict))
```

得到：

```
ID: 0
Name: Ann Blyth
Weight: 6035
Type:starring
Info:1928 births;Living people;American film actors;American musical theatre actors;American child actors;People from Westchester County, New York;
```

STEP 2.1.3 计算节点的度

对 edge.txt 进行分析，发现第一列是按顺序排列的，所以可以直接对第一列进行计数来统计节点的度。

```python
def get_degree(id):
    sr = list(edges.iloc[:, 0])
    return sr.count(id)
```

STEP 2.2 stat.py

STEP 2.2.1 统计网络中每个节点的度

```python
def cal_every_degree(edges):
    sr = list(edges.iloc[:,0])
    node = set(sr)
    degree_dict={}
    for i in node:
        print(i)
        degree_dict[i]=sr.count(i)
    return degree_dict
```

STEP 2.2.2 计算所有节点度的平均值

```python
def cal_average_degree(degree_dict):
    return np.mean(list(degree_dict.values()))
```

得到：

```
average_degree=6.00653677462888
```

STEP 2.2.3 计算度的分布

```python
def cal_degree_distribution(degree_dict):
    degree_distribution={}
    for i in degree_dict:
        degree_distribution[degree_dict[i]]=degree_distribution.get(degree_dict[i],0)+1
    return sorted(degree_distribution.items(), key=lambda e:e[0])
```

得到：

[(1, 3416), (2, 3088), (3, 2855), (4, 2828), (5, 2546), (6, 2001), (7, 1595), (8, 1130), (9, 876), (10, 642), (11, 443), (12, 322), (13, 268), (14, 196), (15, 165), (16, 173), (17, 120), (18, 98), (19, 93), (20, 93), (21, 82), (22, 57), (23, 44), (24, 55), (25, 50), (26, 41), (27, 39), (28, 15), (29, 28), (30, 31), (31, 24), (32, 26), (33, 32), (34, 16), (35, 11), (36, 14), (37, 13), (38, 10), (39, 16), (40, 9), (41, 8), (42, 5), (43, 10), (44, 7), (45, 7), (46, 5), (47, 8), (48, 4), (49, 2), (50, 4), (51, 6), (52, 3), (53, 3), (54, 5), (55, 4), (56, 2), (57, 4), (58, 2), (59, 6), (61, 2), (62, 4), (63, 3), (64, 3), (66, 5), (67, 2), (69, 3), (71, 1), (73, 1), (75, 1), (76, 3), (77, 1), (78, 1), (79, 1), (81, 1), (82, 1), (83, 1), (85, 1), (86, 1), (90, 1), (91, 1), (92, 2), (93, 1), (96, 1), (97, 1), (99, 2), (100, 1), (104, 1), (106, 1), (109, 1), (113, 1), (119, 1), (125, 1), (130, 1), (138, 1), (142, 1), (162, 1)]

STEP 2.3 graph.py

STEP 2.3.1 存储节点信息和边信息

可以利用 node.py 中的 init_node()和 get_degree()函数，然后合并成为字典的值，键为节点 id。

```python
def init_graph(vertices,edges):
    graph_dict = {}
    nodes = set(list(vertices.iloc[:,0]))
    for i in nodes:
        print(i)
        node_dict = init_node(vertices,i)
        degree = get_degree(edges,i)
        graph_dict[i] = {"node":node_dict,"degree":degree}
    return graph_dict
```

Python

得到的字典为

{...30084: {'node': {'ID': 30084, 'Name': 'Walk Hard: The Dewey Cox Story', 'Weight': 15890, 'Type': 'movie', 'Info': '2007 films;American comedy films;2000s comedy films;Films set in the 1940s;Films set in the 1950s;Films set in the 1960s;Films set in the 1970s;Films set in the 1980s;Films set in the 1990s;English-language films;Columbia Pictures films;Films shot digitally;'}, 'degree': 9},...}

STEP 2.3.2 序列化和反序列化

采用 pickle 模块中的 dumps 和 loads 函数（dump 和 load 函数涉及到文件操作）

```python
def save_graph(graph_dict):
    return pickle.dumps(graph_dict)

def save_graph(series):
    return pickle.loads(series)
```

Python

Ref.

详解python中pickle模块的一些函数_gaishi_hero的博客-CSDN博客

pickle模块是python中用来讲Python对象序列化和解序列化的一个工具。“pickling”是将Python对象转化为字节流的过程，而“unpickling”是相反的过程（将来自“binary file或bytes-like object”的字节流反转为对象的过程）。5种协议Protocol version 0 是最原始一种协议，它向后与以前的

C blog.csdn.net

STEP 3 构建 Visualization 模块

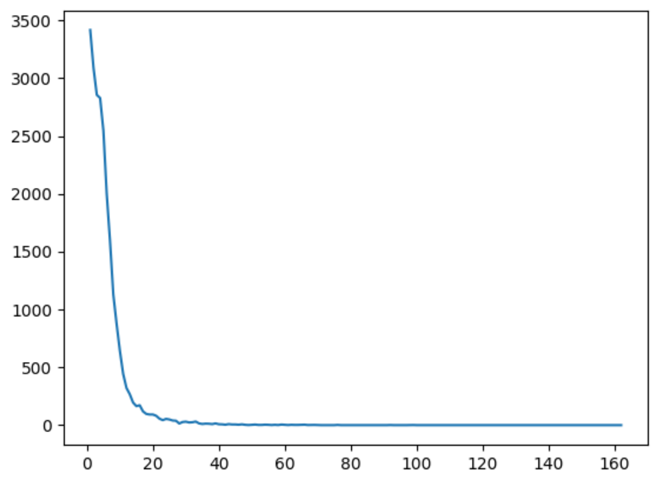
STEP 3.1 绘制度的分布图

可以参照 stat.py 中的 cal_degree_distribution 函数的计算结果。

```python
def plot_distribution(degree_distribution):
    x=[]
    y=[]
    for i in degree_distribution:
        x.append(i[0])
        y.append(i[1])
    plt.plot(x,y)
    plt.show()
```

Python

得到：



横轴为度，纵轴为节点的数量，可见数据集中的节点的度呈长尾分布，度小的节点数量多，度大的节点数量非常少。

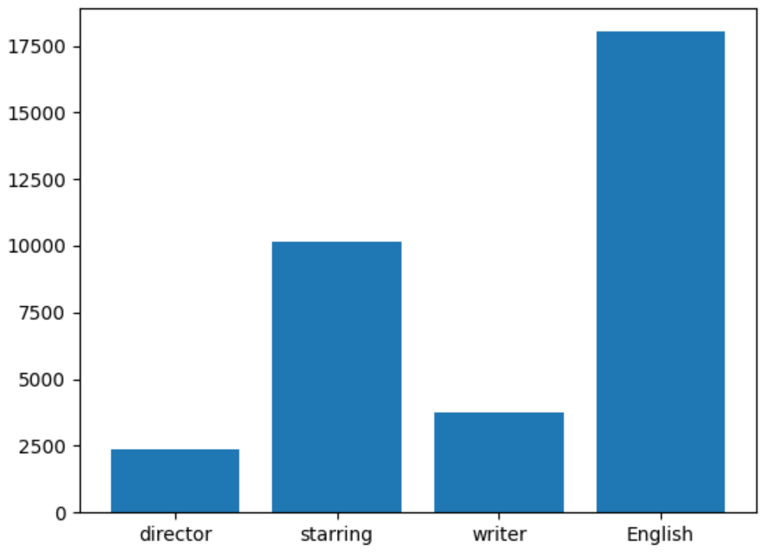
STEP 3.2 绘制图中节点属性的统计图

这里我们选择节点的职业属性来进行统计。

```
def count_job(edges):
    job_list=list(edges.iloc[:,3])
    job_title = ['director','starring','writer','English']
    num = [0,0,0,0]
    for i in job_list:
        if i == 'director':
            num[0]+=1
        elif i == 'starring':
            num[1]+=1
        elif i == 'writer':
            num[2]+=1
        else:
            num[3]+=1
    plt.bar(range(len(num)), num, tick_label=job_title)
    plt.show()
```

Python

得到：


director
starring
writer
English