Challenge2

The Illusion Of Hidden Personal Data

Yanni CUI
Runlu QU

1.Business understanding

- problem explication

We are data scientists in LinkedIn, working for the data processing and data analysis which can help our company make some business decisions.

Now, we got a new challenge, our colleagues from the Marketing department want to organize an online marketing campaign for their client which is a restaurant in the Bay area. The aim is to attract more people. After they have done some researches about social network analysis, they want us to find the 5 most influential people on the network, and use their influences to let the restaurant be exposed to the public to promote it.

However, the problem is not so easy as they thought. This marketing campaign is for a restaurant located it the Bay area, thus, the first thing which we need to consider is to find the people who are in the same area or near that area. Only the most influential people in the right area can bring real customers.

Generally, only a few users voluntarily provide their attributes in an online profile which leads to the missing information, it is very hard to find the influential people in the right area. There is a chance for us to complete their profile via their social connections. We can intuitively suppose that the connected people will share some same attributes. For example, Some people connect cause they have common friends, the same location, the common colleges...Even for the recommendation friends via LinkedIn, they recommend those people normally because of the sharing of friends.

Thus, the first problem we need to solve is to find the algorithm which could properly

predict the missing attributes by the social connections. After we getting the complete data, it would not be a big problem for finding the 5 most influential people.

- Technical analysis

As the given datasets from LinkedIn, we have some people's profiles and how the connection between them. We can build a social network via these data, and each person is regarded as a node, and the connection between different people is thought of as the edge. For these people, we have the three attributes in their profile: location, college, employer. But not everyone has complete information in their profile. We are aware that only 40% of the users have provided their location in their profile,40% employer information, as well as the 40% college information.

Based on the demand analysis, we can solve the problem by dividing it into 3 tasks:

1.reading the given data, to see which kind of information we have, and proposing some methods to pad the missing information in each attribute.

2. Decide one method to predict the empty information in their profile, which should have higher accuracy than the naive method.

3.Using the graph theory to find the list of influential people, according to their location to pick up 5 people in the right. Or we can do more data mining ,combining other attributes to chose the people who can best promote the restaurant.(like their employer information will show their current career,which makes a great difference in the current life .)

2.Data understanding

Before building a social network, we knew that the data is shown as a text file. For each person, we use a string beginning with a U and appended with digits to identify one 1person. There are 3 attributes in their profile, but not all the profiles are fully filled.

As shown in Table 1, we got two datasets, one is the original data, which contains all the given information via LinkedIn users. As for the 60%-empty-data, the amount of users does not change but lost 60% information of each attribute.

Table 1

	The original data	60% empty data
The number of user	811	811
The users with college info	540	230
The uses with location	811	336
The uses with employer	730	297

In table 2, there are some examples of the data type. There is no formula for that information. Before reading the details, we were supposed to use deep learning to classifier similar attributes, and use the KNN algorithm to cluster. We thought the people in the same cluster would like to share their profile. But according to the data, there are too many different labels we have (e.g for the college, we have more than 40 universities). So it is not possible to use deep learning in Our case.

Table 2

employer	'U1313': ['discovery education', 'ctb mcgraw-hill', 'university of charleston university of south carolina', 'south carolina department of education']
college	'U1313': ['longwood university'], 'U8804': ['università di bologna'], 'U2136': ['university of illinois at urbana-champaign']
location	'U1313': ['norfolk virginia area'], 'U8804': ['bologna area italy'], 'U2649': ['urbana-champaign illinois area']

3.Preparation

- Naive method

It is a basic idea which predicts the value by counting the number of occurrences each value, then taking the most represented attribute value among neighbors.

This method is based on the assumption that the neighbors will share the attributes to each other.It will be a very good method to predict the value if we miss only several values. On the contrary, in our case, we miss 60% information. There are many nodes that are totally empty, which have only one neighbor. In this situation, it is not accurate to predict.

- Our ideas

After the analysis, we come up with two methods.

The first one is based on naive method. We would like to divide the network to several communities using the Louvain algorithm(use twice).
Theoretically,the nodes in the same community have higher probability to share the same attribute. Then we use the naive method to predict the value in each community.

The second one has a similar idea as the first one. Above all, we use the Louvain algorithm to divide the network into some communities. Then
we define two types of nodes(triangle nodes and group node), using there two connections to re-clustering the nodes. For the other nodes, we use the naive method.

3. Illustrate our strategies and our results

In this section, we will clarify the methods and results we use, as well as analyze the strengths and weaknesses of each method. In the end, we will give the thought of the next step.

3.1 Most detailed strategy

First of all, we start with the project, by understanding the naive method. We think that this method is a reasonable and effective method to some extent, so the first method we proposed is to improve on the basis of the naive method.

In our opinion, the defect of the naive method is:

- Suppose there is a set of nodes ***[node_A, node_B, node_C, node_D, node_E, node_F, node_G].***
- ***node_B, node_C, node_D, node_E, node_F, node_G*** are neighbors of ***node_A*** . We have attributes for ***node_B, node_C, node_D, node_E, node_F,***

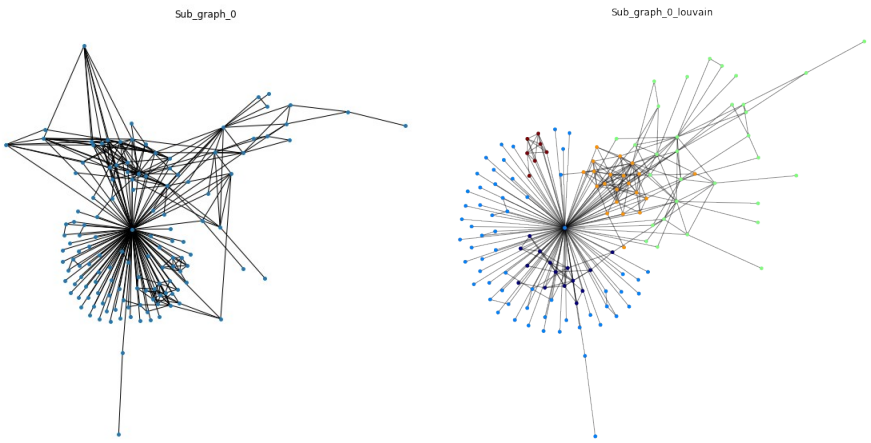
node_G, so we want to predict the value of ***node_A*** according to information of the others.

- Suppose ***attr_false*** appears four times and ***attr_true*** appears two times.

For ***node_A***, depending on its neighbor attribute, ***attr_false*** information exists more frequently, but ***node_A*** itself does not belong to the area where the nodes of ***attr_false*** are located, so the true value of A itself It should be ***attr_true***, but it is predicted to be ***attr_false***.

In order to avoid this situation, we chose to use the Louvain algorithm to partition the entire contact network map first to make a most detailed strategy. Louvain divided the entire map into 20 small communities. Through the picture at the bottom left (sub_graph_0), this is the picture of the first community. There are still many nodes and the connection is still complicated. So, we have partitioned the communities again for each community like the graph at the bottom right (Sub_graph_0_louvain).

Under the condition, we use naive method to predict the nodes' information for each node. After that, we found that there are still a lot of empty nodes, so we restore the child partitions (Sub_graph_x_louvain) in each partition to each partition (Sub_graph_x), and then use the naive method to predict the information. Eventually, we restored the partitions to the entire network and again used the naive method to predict the information that was still not filled.



Through this method, we did not get beautiful results. On the contrary, the results we obtained were lower than the naive method in the whole picture. The accuracy of the prediction results is as follow graph:

```
The results of each type of attributes are:
Employers: 16.228070% of the predictions are true
Locations: 31.932773% of the predictions are true
Colleges: 27.441860% of the predictions are true
```

The reason we get this result is due to the following:

• There is a lot of missing data in the original file, which results in many sub-communities not having single attribute information after two partitions. This situation makes the two partitions not achieve the desired purpose.

• The information could not be fully completed, and even a small amount of information was not filled in during the verification process.

• The division is too fine, resulting in the data in the sub-community after two partitions are too single, which directly causes the entire cell to be filled with the same data, failing to reach a valid prediction.

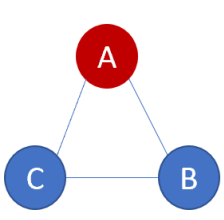
In this case, we even tried to do one partition to simplify the problem, and then use the naive method to fill the information. But the results are still not ideal. The results for employers, locations, and colleges are bellows:

```
19.767442% of the predictions are true
32.330827% of the predictions are true
30.000000% of the predictions are true
```

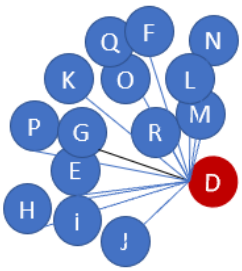
3.2 Prediction based on linkages in the graph

Through the above method, we find that it is not desirable to predict the unknown attribute node only by calculating the frequency of attributes in its neighbors, so we decided to classify the relationships in the graph. For nodes in different relationship types, we use different methods to make predictions.

We mainly divide the links between the nodes in the graph into the following two types, as shown in the following Triangle nodes and Group node:



Triangle nodes



Group nodes

The prediction rules we define for each type of contact are as follows:

• In Triangle nodes, we think relationships between the nodes are very close, so we choose to share the information between the nodes.

• In Group nodes, because the red node is related to all the other nodes, we think that the red node has information in other points so we will add the three most frequent information to the red node. For blue nodes, they have no choice but to get red node information.

• In other situations, we use the naive method for the prediction.

The whole algorithm steps we designed are as follows:

- In order to prevent data redundancy and to subdivide the connections in the graph, we first partition the graph using the Louvain algorithm.
- Then, we restore sub-communities back to the original graph, we add the nodes that are still unpredicted to a list, we look for Triangle nodes and Group nodes in the graph, and then predict and fill the data according to the above rules. We believe that the more neighbors tend to have more information, so we first select the node has least neighbors to start the prediction, and finally predict node has the more neighbors.
- After doing this, we have already filled the information for close contact nodes. For the remaining unpredicted nodes, we predict it using the naive method.

Through this method, our accuracy has been improved. The final prediction accuracy is as follows:

```
The predict results for each type of attributes:
Employers: 25.751073% of the predictions are true
Locations: 43.105111% of the predictions are true
Colleges: 36.244541% of the predictions are true
```

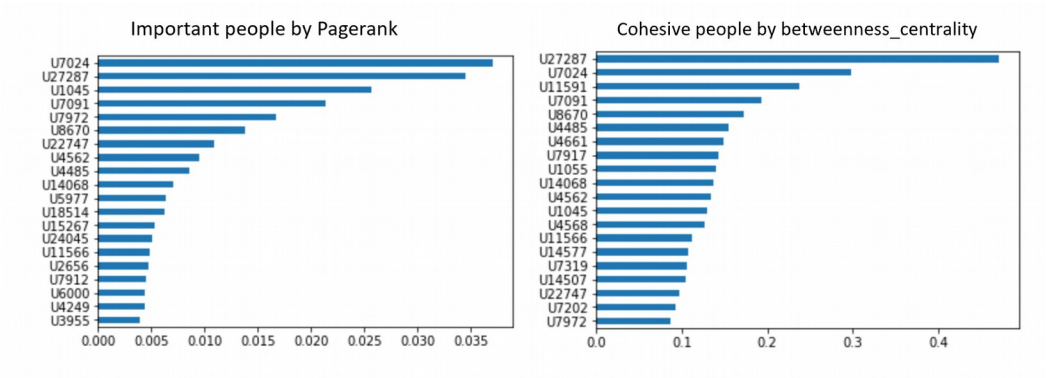
For the ultimate accuracy, this is slightly higher than the basic naive method, because we divide the whole graph and look for people with close relationships. This method is more user-friendly than simply looking for the most frequently occurring information, also has a high accuracy rate. And for the center node, we added more than one prediction to it, which also increases the probability that it will press the correct information. But also, during the process of prediction, many nodes have a common neighbor, that is, for two fixed nodes, they have many triangular relationships, so more than one information will be added, thus greatly increasing the amount of predicted information, which makes the accuracy rate dropped a lot.

3.3Find the five most influential people

Since the purpose of this project is to find five people to recommend a restaurant in the San Francisco Bay area, our choice is based on the following:

- Looking for people located in the San Francisco Bay area.
- Pick the most important and cohesive people.

We use the *PageRank* algorithm to find the most important people, using the *betweenness_centrality* algorithm to find the most cohesive people.



From the pictures above, we find important and cohesive people. Finally, our result for the five most influential people is:

```
['U27287', 'U15267', 'U11566', 'U2656', 'U4568']
```

4.Optimization

The prediction of information can be designed in more detail. We can customize different prediction methods according to different information attributes. For example, for the employee attribute, one person usually has multiple information, and for the location attribute, there is usually only one attribute information for one person. This will greatly improve the accuracy of the prediction. In addition, the shared value of the triangular relationship can be filtered to reduce the prediction of invalid information.

Références:

[1] Rui Li,Chi Wang,Kevin Chen-Chuan Chang--User Profilinginan Ego Network: Co-profiling Attributes and Relationship

APPENDIX

In [ ]:
```
import networkx as nx
import matplotlib.pyplot as plt
from matplotlib import pylab
import numpy as np
import pickle
from collections import Counter
```

In [ ]:
```
import community as c
partition = c.best_partition(G)
```

In [ ]:
```
from collections import defaultdict
comm_dict = defaultdict(list)
for person in partition:
    comm_dict[partition[person]].append(person)
```

In [ ]:
```
comm_0 = comm_dict[0]
comm_1 = comm_dict[1]
comm_2 = comm_dict[2]
comm_3 = comm_dict[3]
comm_4 = comm_dict[4]
comm_5 = comm_dict[5]
comm_6 = comm_dict[6]
comm_7 = comm_dict[7]
comm_8 = comm_dict[8]
comm_9 = comm_dict[9]
comm_10 = comm_dict[10]
comm_11 = comm_dict[11]
comm_12 = comm_dict[12]
comm_13 = comm_dict[13]
comm_14 = comm_dict[14]
comm_15 = comm_dict[15]
comm_16 = comm_dict[16]
comm_17 = comm_dict[17]
comm_18 = comm_dict[18]
comm_19 = comm_dict[19]
```

In [ ]:
```
comm = {}
sub_graph = {}

for c_keys, c_values in comm_dict.items():
    comm[c_keys] = c_values
    sub_graph[c_keys] = G.subgraph(c_values)
```

In [ ]:
```
college={}
location={}
employer={}
# The dictionaries are loaded as dictionaries from the disk (see pickle in Python doc)
with open('mediumCollege_60percent_of_empty_profile.pickle', 'rb') as handle:
    college = pickle.load(handle)
with open('mediumLocation_60percent_of_empty_profile.pickle', 'rb') as handle:
    location = pickle.load(handle)
with open('mediumEmployer_60percent_of_empty_profile.pickle', 'rb') as handle:
    employer = pickle.load(handle)
```

In [ ]:
```
def pred_tri(graph, comm, empty, attr):

    predicted_values = {}
    comm_empty = [] #需要填补信息的点列表
    n_nbr_1 = {}
    nbrs_attr_values = []

    for i in comm:
        for j in empty:
            if i == j:
                comm_empty.append(i)

    #查看每个点的邻居都有谁
    for node in comm_empty:
        n_nbr_1[node] = list(graph.neighbors(node))
    ordered_n_nbr_1 = OrderedDict(sorted(n_nbr_1.items(), key=lambda x: len(x[1])))

    for node, nbrs in ordered_n_nbr_1.items():#遍历需要填补的点和其邻居
        predicted_values[node] = []
        nbrs_attr_values=[]
        for n in nbrs:#遍历该点的邻居
            set_share_nbr = set(nbrs) & set(graph.neighbors(n))#检查需要填补的点和其邻居间有没有共同的邻居
            list_share_nbr = list(set_share_nbr)#将两个点共同的邻居存入列表
            if list_share_nbr:#如果列表不为空
                for nbr_attr in list_share_nbr:#则遍历该表中的点
                    if nbr_attr in attr.keys():
                        for val in attr[nbr_attr]:#获取点的属性值
                            if not val in predicted_values[node]:
                                predicted_values[node].append(val)#并将属性值加入列表

    return predicted_values
```

In [ ]:
```
pre_value_sub_emp_tri = {}
pre_value_sub_loc_tri = {}
pre_value_sub_co_tri = {}

for i in range(0,20):
    pre_value_sub_emp_tri[i] = pred_tri(sub_graph[i], comm[i], empty_nodes, employer)
    pre_value_sub_loc_tri[i] = pred_tri(sub_graph[i], comm[i], empty_nodes, location)
    pre_value_sub_co_tri[i] = pred_tri(sub_graph[i], comm[i], empty_nodes, college)

for j in range(0,20):
    for emp in pre_value_sub_emp_tri[j].keys():
        emp_pre_tri[emp] =  pre_value_sub_emp_tri[j][emp]
    for loc in pre_value_sub_loc_tri[j].keys():
        loc_pre_tri[loc] = pre_value_sub_loc_tri[j][loc]
    for co in pre_value_sub_co_tri[j].keys():
        co_pre_tri[co] = pre_value_sub_co_tri[j][co]
```

In [ ]:
```
from collections import OrderedDict
```

In [ ]:
```
# 给在emptylist中，并且在分区中并没有被成功预测成分的点根据三角关系赋值
def pred_tri_g(graph, attr_pre, empty, attr):

    graph_empty = [] #需要填补信息的点列表
    predicted_values = {}
    for i in list(graph.nodes()):#列出图中的点
        if i in empty:#如果点在未含信息列表中
            for j in attr_pre.keys():
                if i==j:#并且点在已预测过一遍的列表中
                    if len(attr_pre[j])==0  :#并且该点并没有被赋值
                        graph_empty.append(i)#则将该点加入列表

    n_nbr_1 = {}
    n_nbr_2 = {}
    nbrs_attr_values = []

    #查看每个点的邻居都有谁
    for node in graph_empty:
        n_nbr_2[node] = list(graph.neighbors(node))

    ordered_n_nbr_2 = OrderedDict(sorted(n_nbr_2.items(), key=lambda x: len(x[1])))

    for node, nbrs in ordered_n_nbr_2.items():#遍历需要填补的点和其邻居
        predicted_values[node] = []
        for n in nbrs:#遍历该点的邻居
            set_share_nbr = set(nbrs) & set(graph.neighbors(n))#检查需要填补的点和其邻居间有没有共同的邻居
            list_share_nbr = list(set_share_nbr)#将两个点共同的邻居存入列表
            if list_share_nbr:#如果列表不为空
                for nbr_attr in list_share_nbr:#则遍历该表中的点

                    if nbr_attr in attr.keys():
                        for val_a in attr[nbr_attr]:#获取点的属性值
                            if not val_a in predicted_values[node]:
                                predicted_values[node].append(val_a)

                    if nbr_attr in attr_pre.keys():
                        for val_b in attr_pre[nbr_attr]:
                            if not val_b in predicted_values[node]:
                                predicted_values[node].append(val_b)

    return predicted_values
```

In [ ]:
```
pre_value_emp_tri = {}
pre_value_loc_tri = {}
pre_value_co_tri = {}

pre_value_emp_tri = pred_tri_g(G, emp_pre_tri, empty_nodes, employer)
pre_value_loc_tri = pred_tri_g(G, loc_pre_tri, empty_nodes, location)
pre_value_co_tri = pred_tri_g(G, co_pre_tri, empty_nodes, college)
```

In [ ]:
```
for emp_key in emp_pre_tri.keys():
    if emp_key in pre_value_emp_tri.keys():
        for emp_value in pre_value_emp_tri[emp_key]:
            emp_pre_tri[emp_key].append(emp_value)

for loc_key in loc_pre_tri.keys():
    if loc_key in pre_value_loc_tri.keys():
        for loc_value in pre_value_loc_tri[loc_key]:
            loc_pre_tri[loc_key].append(loc_value)

for co_key in co_pre_tri.keys():
    if co_key in pre_value_co_tri.keys():
        for co_value in pre_value_co_tri[co_key]:
            co_pre_tri[co_key].append(co_value)
```

In [ ]:
```
def pred_info_not_tri(graph, attr_pre, attr):

    still_empty = []
    predicted_values = {}

    for attr_key in attr_pre.keys():
        if len(attr_pre[attr_key])==0:#如果原始空值点属性值长度为0，则信息未被补全
            still_empty.append(attr_key)

    for i in still_empty:
        predicted_values[i] = []
        #给大团中心点赋值
        if len(list(G.neighbors(i)))>10:#如果该点的邻居数超过10个，则认为该人是大团的中心点
            attr_collect = []#建立列表以储存中心点的属性值
            for nbr in G.neighbors(i):#遍历中心点邻居
                if nbr in attr.keys():
                    for val_1 in attr[nbr]:#获取邻居点的属性值
                        if not val_1 in predicted_values[i]:
                            predicted_values[i].append(val_1)#并将属性值加入列表

                elif nbr in attr_pre.keys():
                    for val_0 in attr_pre[nbr]:#获取邻居点的属性值
                        if not val_0 in predicted_values[i]:
                            predicted_values[i].append(val_0)#并将属性值加入列表

        #给非大团中心点或边缘节点赋值
        if len(list(G.neighbors(i)))<=10:#如果该点的邻居数不超过10，则认为该人不是大团的中心点
            attr_collect_nbr = []#建立列表以储存邻居的属性值
            for nbr in G.neighbors(i):#遍历邻居
                if nbr in attr.keys():
                    if attr[nbr]:#如果邻居有属性
                        for val_2 in attr[nbr]:#获取邻居点的属性值
                            attr_collect_nbr.append(val_2)#则将属性放到列表里
                elif nbr in attr_pre[nbr]:
                    if attr_pre[nbr]:
            if attr_collect_nbr:#如果列表不为空
                cpt_nbr=Counter(attr_collect_nbr)#则计算各属性值的总数
                a,nb_occurrence=max(cpt_nbr.items(), key=lambda t: t[1])#选取出现频率最高的属性
                predicted_values[i].append(a)#给需填补的点添加属性信息

    return predicted_values
```

In [ ]:
```
pre_value_emp_notri = {}
pre_value_loc_notri = {}
pre_value_co_nptri = {}

pre_value_emp_notri = pred_info_not_tri(G, emp_pre_tri, employer)
pre_value_loc_notri = pred_info_not_tri(G, loc_pre_tri, location)
pre_value_co_notri = pred_info_not_tri(G,  co_pre_tri, college)
```

In [ ]:
```
for em_key in emp_pre_tri.keys():
    if em_key in pre_value_emp_notri.keys():
        for em_value in pre_value_emp_notri[em_key]:
            emp_pre_tri[em_key].append(em_value)

for lo_key in loc_pre_tri.keys():
    if lo_key in pre_value_loc_notri.keys():
        for lo_value in pre_value_loc_notri[lo_key]:
            loc_pre_tri[lo_key].append(lo_value)

for co_key in co_pre_tri.keys():
    if co_key in pre_value_co_notri.keys():
        for co_value in pre_value_co_notri[co_key]:
            co_pre_tri[co_key].append(co_value)
```

In [ ]:
```
for i in employer.keys():
    emp_pre_tri[i] = employer[i]
for j in location.keys():
    loc_pre_tri[j] = location[j]
for k in college.keys():
    co_pre_tri[k] = college[k]
```

5 most influential people

In [ ]:
```
import pandas as pd
```

In [ ]:
```
nodes_bay = []
for loc_key in loc_pre_tri.keys():
    if 'san francisco bay area' in loc_pre_tri[loc_key]:
        nodes_bay.append(loc_key)
```

In [ ]:
```
# plt.figure(num=None, figsize=(15, 15), dpi=100)
page_ranks = pd.Series(nx.algorithms.pagerank(G)).sort_values()
page_ranks.tail(20).plot(kind="barh")
plt.show()
```

In [ ]:
```
page_ranks_nodes = []
```

In [ ]:
```
for obj in page_ranks.tail(50).index:
    page_ranks_nodes.append(obj)
```

In [ ]:
```
page_ranks_bay=[]
for i in reversed(page_ranks_nodes):
    if i in nodes_bay:
        page_ranks_bay.append(i)
```

In [ ]:
```
between = pd.Series(nx.betweenness_centrality(G)).sort_values()
between.tail(20).plot(kind="barh")
plt.show()
```

In [ ]:
```
bt_nodes = []
for obj in between.tail(50).index:
    bt_nodes.append(obj)
```

In [ ]:
```
bt_bay=[]
for j in reversed(bt_nodes):
    if j in nodes_bay:
        bt_bay.append(j)
```

In [ ]:
```
most_influence = []
for pr in page_ranks_bay:
    if pr in bt_bay:
        most_influence.append(pr)
```