Scholar Name Disambiguation

project report

Yifan Lu

May 27, 2018

Abstract This report is for the project of the course Wireless Communication and Mobile Network.In this report,I will show how I solved the existing problems in the scholar name disambiguation on Acemap.KNN is used to merge the scattered papers and the important variables are determined by experiments.

1 Introduction

For regular people, distinguishing the scholars who have the same names also need a certain amount of information. According to conferences, journals, coauthors, years, etc., we can judge whether the author of these papers is the same person, which is a harder task for the literature search system.Therefore, it is very important to design the algorithm and make Acemap distinguish the same-named scholars.

2 Idea

When I joined the project group,the leader Wang Bo had completed a preliminary algorithm but there were still a lot of problems existing in the system.In Wang's program,the heterogeneous networks are used to describe the relationship between each article. An article represents a node in the network. The relationship between each article form the edges of the network. Different types of relationship correspond to edges of different weights.The similarity between article authors can be calculated with some formulas.In general,the similarity between two nodes will be related to the sum of the edges connecting the nodes and the weights of these edges.

Even if two articles do not seem to be related as they look like, it is possible to connect them through several edges in a heterogeneous network.For example,among three papers A,B and C written by the same-named scholar,A and B have the same coauthor and B and C come

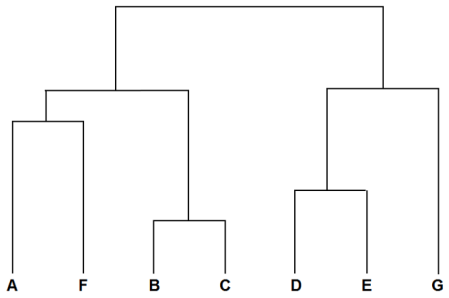

Figure 1: hierarchical clustering

from the same venue.Then we may think A and C are written by the same person,too.We can also infer this from the organation the writers come from or the journal the papes published on .

On the basis of heterogeneous networks, we can use hierarchical clustering to distinguish scholars of the same name. All articles of the same-named scholar will be divided into multiple classes. Articles under each class are regarded as different authors of the same name.

Here are the steps of hierarchical clustering:

(1) Calculate the similarity of each pair of nodes;

(2) Connect the corresponding node pairs by similarity from high to low to form a tree as shown in fig1;

(3) Merge the scattered nodes;

(4) According to the actual situation, cross-cut the tree diagram to obtain the required classes.

The basic idea of the hierarchical clustering method is to compute the similarity between nodes through a certain measure, and sort them in order of similarity from high to low, then gradually reconnect the nodes.The advantage of this method is that it can be stopped at any time. For example, a threshold is set. When the similarity is lower than the threshold, the partition will be stopped to prevent the articles of different authors of the same name from being grouped into a class.

3 Structure

The flowchart in fig2 is the basic structure of Wang's code.He told me that there are three main problems waiting to be fixed:

(1) How can we know how many clusters we will get at last before clustering;

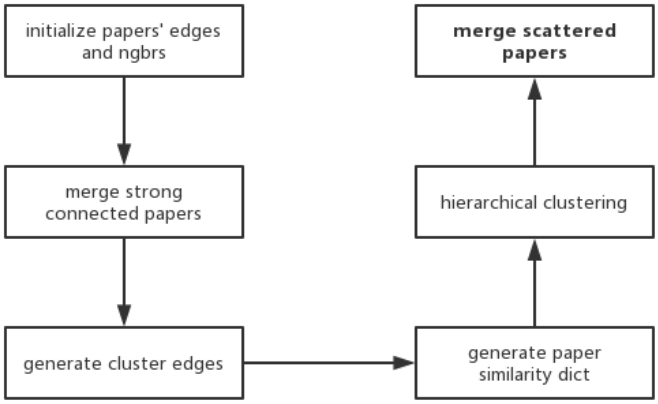

Figure 2: basic structure of Wang's code

(2) When we add a new paper to the data base,how can we get a correct cluster for it without running the hierarchical clustering again;

(3) How to merge the scattered papers;

At last I chose to focus on problem(3).Beacuse Wang's code was written in python,I wrote my own code in python,too,so that I could adapt my work to the whole project easily.My initial conception was to use KNN to merge the scattered papers.KNN is so called k-nearest neighbor learning, which is a commonly used supervised learning method. Its working mechanism is that given the test sample,we can find its nearest k training samples based on some distance metric, and then select the most common category tags among those k-nearest neighbors as the test sample's category tag.

The flowchart in fig3 is the basic structure of my algrothim of merging the scattered papers.

To get the best results,we must pay attention to the distance metric and the vaule of k. Because the scattered papers generally has no edge with any other paper,leading to their low similarity with other papers.If we stick to the old way to calculate similarity,we can't tell which cluster the scattered paper should belong to.So we must introduce a new distance metric.Generally speaking,a scolar may foucus on one area and professional vocabulary in the same area surly have a strong connectivity.So the tiltles written by the same scholar will have higher similarities.On the other hand,the same person are likely to use the same words in the titles of the papers he wrote.So in this project,I use the word vector representations as the distance metric and it needs the library spacy in python.

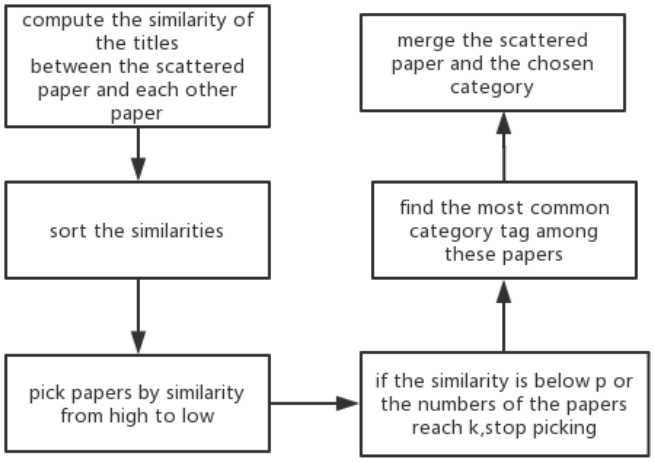

Figure 3: basic structure of my code

Also we know that sometimes the category the scattered paper should belong to may only have a few papers so if we set k to a large number,some papers of low similarity with the scattered paper will be picked.On the other hand,if we pick too many papers,the papers of low similarity will come in,too.When the similarity between the scattered paper and the merged paper is too low,they will lead the most common category tag to a wrong direction.To solve this problem,i introduced k and p to make the algrothim more proper:when the similarity between the scattered paper and the picked paper is below p or the numbers of the papers that have already been selected reach k,the code will stop picking papers by similarity.This lead to the fact that most of the papers picked will belong to the category that the scattered paper should be merged to.

4 Experiment

To determine the vaule of k and p,I need to do more experiments.Thanks to Wang,he gives me a groundtruth dataset so I can get the proper vaule of k and p by the classified papers.At first,I sorted the papers by label beacause in this groundtruth database,papers of the same label belong to the same category.Then I compute the similarity matrix of the papers of a scholar of the same name and the i_{th} line and the j_{th} row of the matrix represents the similarity between the i_{th} paper and the j_{th} paper.I saved the matrix as mat profile so I could open it with Matlab then put it in Excel to do more complicated sort.Because the papers have been sorted,I can obeserve the similarity matrix to know how to choose the right k and p.

I notice that in the groundtruth database,there is a scholar called Daniel Massey and his/her papers are only divided into two categories.In other words,there are only two scholars named Daniel Massey in this database.In addition,the number of the papers in the second category is one.So I can get p by finding the maximum among the similarities between the paper in the second category and each paper in the first category and the minimum among the similarities between the papers from the first category.maixmum=0.2180 ,minimum=0.2308,so I let p=0.22 at last.

There is another scholar named Alok Gupta who has two categories.What is different is that in each category of Alok Gupta,there are many papers.So we can find the k_{th} similarity between papers from the same category is higher than the maximum similarity between papers from different categories and the $(k+1)_{th}$ similarity between papers from the same category is lower than the maximum similarity between papers from different categories.After putting the title similarity matrix in Excel,I got k=10.

5 Practice

After get k andd p,l should adapt my algorithm to Wang's code.So I must be familar with its structure and variable.Besides,the datatabase Wang has been using is on the Acemap group'server.As a result,I must learn how to run python code on a server.I downloaded WinSCP to put my code on the server and PuTTY to run the code to test whether my code was working.Because the database on the server does not have a groundtruth,so I chose scholar Xinbin Wang to do name disambiguation,which I could know whether I put his papers to the same category by checking his information on the Internet.

At first,I ran the code without merging the scattered papers and save the result.Secondly,I ran the code after adding my code.Then I can find the scatter papers in the first run but merged in the second run to see if my code can work effectively to merge the scattered papers to the right categories.Of course in the result csv I can only get the paper id,so I must use Dataclient.py to get their titles by id.In Table1,there are some examples that changed their categories after merging the scattered papers.We can see the second and third one are surly Xinbin Wang's work but the first one is a mistake.This show that my algoritm still has some disadvantages but it dose do a contribution to the whole project.

Table1:examples

a study of influential factors in mathematical modeling of academic achievement of high school students	2011
wired and wireless networks a transport layer perspective	2007
burst queue performance improvement of voice over multi hop 802 11 networks	2008

6 Conclusion

Through this project,I have a preliminary contact with scientific research.This is also my first time to have a chance to contribute to such a large project.At the start of the project,I learned a lot from Wang Bo.He told me the basic idea of hierarchical clustering and the structure of the code he had already written ,he even teached me how to run the code on the server.In the middle of the project,the members of the group share the code on Github.This was also the first time for me to contribute to a project on Github,which made me learn how to cooporate with others more effectively.At last,to do my part of work,I worked hard and get some achievements.Thanks to everyone who have helped me.

7 Appendix

Here is my code.

//merge the scattered papers

```
def merge_scattered_papers(clusters, paper_idx_2_cluster_id, title_sim_matrix, paper_all_ngbrs
                                                , paper_final_edges):
    cluster_merge_pairs = list()
    for cluster_id, cluster in clusters.items():
        if len(cluster.papers) == 1:
            paper_idx = cluster.paper_idx_list[0]
            top_indices = np.argsort(-title_sim_matrix[paper_idx, :])
            list_num = []
            count=0
            flag=0
            for i in top_indices:

                    cluster_small_id = paper_idx_2_cluster_id[paper_idx]
                    cluster_big_id = paper_idx_2_cluster_id[i]
                    if title_sim_matrix[paper_idx, i] > 0.22:
                        list_num.append(cluster_big_id)
                        count=count+1
                    else:
                        flag=1
                    if count==10 or flag==1:
                        counts1=[]
```

```
                        counts2=[]
                        for x in set(list_num):
                            counts1.append(x)
                            counts2.append(list_num.count(x))
                        countsmax=max(counts2)
                        loc=0
                        for i in range(len(counts2)):
                            if counts2[i]==countsmax:
                                break
                            loc=loc+1
                        targetid=counts1[loc]
                        print (targetid)
                        if  len(clusters[cluster_big_id].papers) > 5 \
                            and clusters[cluster_big_id].has_no_conflict(clusters[
                                                                    cluster_small_id],
                                                                paper_final_edges,




                            cluster_merge_pairs.append((cluster_small_id,targetid))
                        break

    for merge in cluster_merge_pairs:
        clusters[merge[1]].unit(clusters[merge[0]], paper_idx_2_cluster_id)
        del clusters[merge[0]]

    return clusters
```

//compute the title similarity matrix

```
import numpy as np
import spacy
import scipy.io as sio

nlp = spacy.load('en_core_web_md')

stopwords = set()
for line in open('data/stopwords_ace.txt'):
    stopwords.add(line.replace('\n', '').replace('\r', ''))


def cleanData(s):
    if(s == "null"):
        return None
    else:
        return s
```

```
def compute_title_similarity(title_A, title_B):
    title_nlp = nlp(title_A)
    title_vector_sum = np.zeros(title_nlp[0].vector.shape)
    word_count = 0
    vector_A = np.zeros(title_nlp[0].vector.shape)
    for word in title_nlp:
        if str(word) not in stopwords and len(str(word)) > 1:
            title_vector_sum += word.vector
            word_count += 1
    if word_count != 0:
        vector_A = title_vector_sum / word_count

    title_nlp = nlp(title_B)
    title_vector_sum = np.zeros(title_nlp[0].vector.shape)
    word_count = 0
    vector_B = np.zeros(title_nlp[0].vector.shape)
    for word in title_nlp:
        if str(word) not in stopwords and len(str(word)) > 1:
            title_vector_sum += word.vector
            word_count += 1
    if word_count != 0:
        vector_B = title_vector_sum / word_count
    if len(np.nonzero(vector_A)[0]) == 0 or len(np.nonzero(vector_B)[0]) == 0:
        return 0
    cos_sim = np.dot(vector_A, vector_B) / (np.linalg.norm(vector_A) * np.linalg.norm(vector_B
                                                    ))
    return cos_sim

title_list = []
label_list = []

with open("Daniel Massey.xml", "r") as filetoread:
    for line in filetoread:
                line = line.strip()
                if "FullName" in line:
                    ego_name = line[line.find('>')+1:line.rfind('<')].strip()
                    print (ego_name)
                elif "<title>" in line:
                    title = line[line.find('>')+1: line.rfind('<')].strip()
                    title_list.append(title)
                elif "<label>" in line:
                    label = cleanData(int(line[line.find('>')+1: line.rfind('<')].strip()))
                    label_list.append(label)


for each_list in title_list:
    if isinstance(each_list,list):
        for new_each in each_list:
```

```
            print (new_each)
    else:
        print (each_list);

old_title_list=title_list
old_label_list=label_list
title_list = []
label_list = []
paper_count = len(old_title_list)
count=0
old_count=count

for i in range(0,25):
    old_count=count
    for j in range(0, paper_count):
            if old_label_list[j]==i:
                title_list.append(old_title_list[j])
                label_list.append(old_label_list[j])
                count=count+1
    if count==old_count:
        break



title_sim_matrix = np.zeros((paper_count, paper_count))
for i in range(paper_count):
  for j in range(i + 1, paper_count):
            title_sim = compute_title_similarity(title_list[i], title_list[j])
            title_sim_matrix[i, j] = title_sim
            title_sim_matrix[j, i] = title_sim
sio.savemat('saveddata.mat', {'xi': title_sim_matrix})
print (title_sim_matrix)
print (label_list)
```