经济管理学院

课　程　报　告

（复杂网络与社会计算）

题　　目：week11 课程作业

课程教师：赵吉昌

学院/专业：信息管理与信息系统

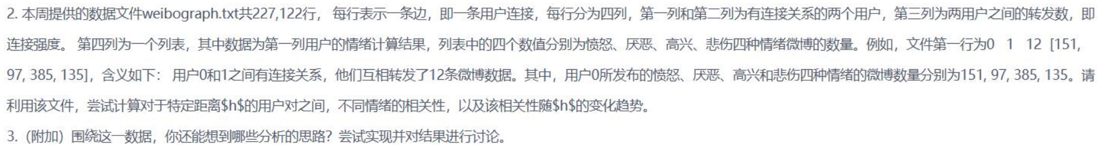
学生姓名：范春

学　　号：21377061

2024 年 5 月 7 日

北京航空航天大學
BEIHANG UNIVERSITY

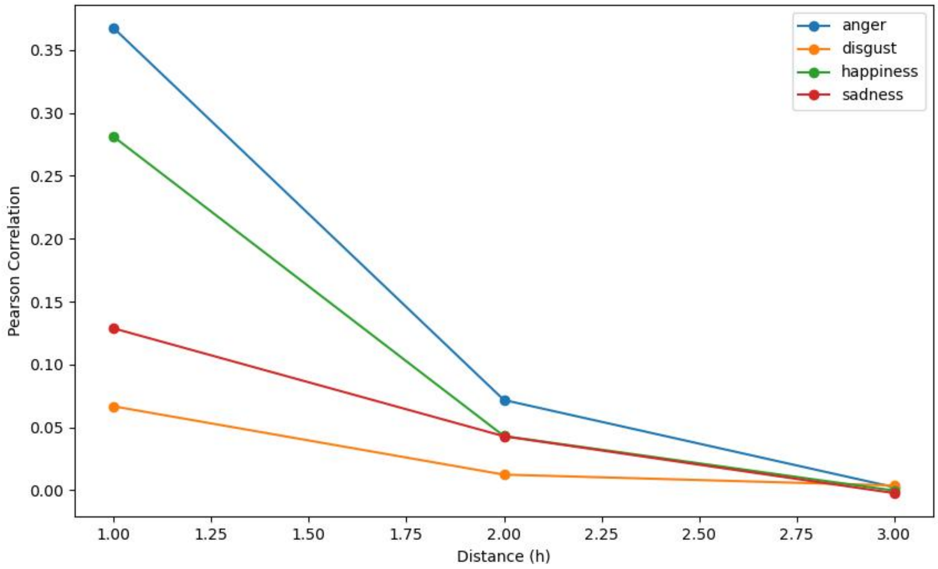
作业要求如下：



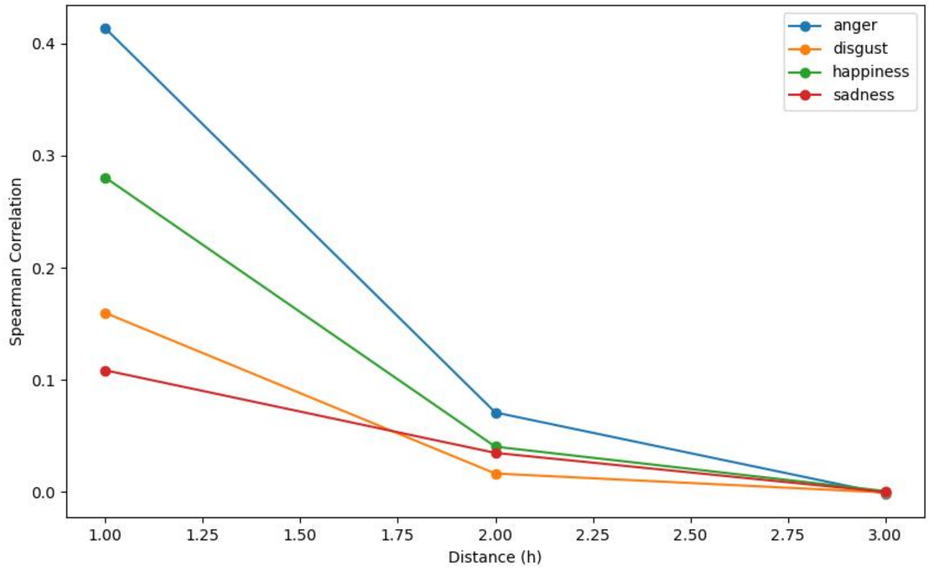
2. 本周提供的数据文件weibograph.txt共227,122行， 每行表示一条边，即一条用户连接，每行分为四列，第一列和第二列为有连接关系的两个用户，第三列为两用户之间的转发数，即连接强度。 第四列为一个列表，其中数据为第一列用户的情绪计算结果，列表中的四个数值分别为愤怒、厌恶、高兴、悲伤四种情绪微博的数量。例如，文件第一行为0　1　12［151, 97, 385, 135]，含义如下： 用户0和1之间有连接关系，他们互相转发了12条微博数据。其中，用户0所发布的愤怒、厌恶、高兴和悲伤四种情绪的微博数量分别为151, 97, 385, 135。请利用该文件，尝试计算对于特定距离h的用户对之间，不同情绪的相关性，以及该相关性随h的变化趋势。

3.（附加）围绕这一数据，你还能想到哪些分析的思路？尝试实现并对结果进行讨论。

1、Pearson 相关性



2、Spearman 相关性



由上图可知，在不同的距离下，愤怒情绪的相关性都是最大的，且随着 h 的增大，相关性减少。

完整代码如下：

```
import pymongo
from pymongo import MongoClient
import networkx as nx
import scipy.stats
import random
import json
```

```
from collections import defaultdict
import matplotlib.pyplot as plt

client = MongoClient("mongodb://localhost:27017")
db = client['week11']
collection = db['week11']
def generateGraph(filePath):
    emotionsDic = {}
    G = nx.Graph()
    with open(filePath, 'r') as file:
        for line in file:
            user1, user2, weight, emotions = line.strip().split('\t')
            G.add_edge(int(user1), int(user2))
            emotions = list(map(int, emotions[1:-1].split(',')))
            total_sum = sum(emotions)
            if total_sum == 0:
                emotions = [0, 0, 0, 0]
            else:
                emotions = [x / total_sum for x in emotions]
            emotionsDic[int(user1)] = emotions
    return G, emotionsDic
def shortestPathLength(G):
    for node in G.nodes:
        paths = nx.single_source_shortest_path_length(G, node)
        str_paths = {str(k): v for k, v in paths.items()}
        collection.insert_one({'node': str(node), 'distances': dict(str_paths)})

    return collection
def get_random_samples(collection, distance):
    cursor = collection.find({})
    all_distances = []

    for doc in cursor:
        node1 = int(doc['node'])
        distances = {int(k): v for k, v in doc['distances'].items() if v == distance}
        for node2 in distances.keys():
            all_distances.append((node1, node2))
    return all_distances
def create_emotion_list():
    return ([], [])
def calculate_average_correlations(collection, emotionsDic, num_samples, num_repetitions):
    correlation_results = {emotion: {h: [] for h in range(1, 4)} for emotion in ['anger', 'disgust',
'happiness', 'sadness']}
    for h in range(1, 4):
        all_distances = get_random_samples(collection, h)
        for i in range(num_repetitions):
            if len(all_distances)<num_samples:
                samples = all_distances
            else:
                samples = random.sample(all_distances, num_samples)
            emotion_data = defaultdict(create_emotion_list, {emotion: create_emotion_list() for emotion in ['anger', 'disgust', 'happiness', 'sadness']})
            for node1, node2 in samples:
                for emotion, idx in zip(['anger', 'disgust', 'happiness', 'sadness'], range(4)):
                    emotion_data[emotion][0].append(emotionsDic[node1][idx])
                    emotion_data[emotion][1].append(emotionsDic[node2][idx])

            for emotion in emotion_data.keys():
                pearson_corr, _ = scipy.stats.pearsonr(emotion_data[emotion][0], emotion_data[emotion][1])
                spearman_corr, _ = scipy.stats.spearmanr(emotion_data[emotion][0], emotion_data[emotion][1])
                correlation_results[emotion][h].append((pearson_corr, spearman_corr))

    average_correlation_results = {emotion: {h: (0, 0) for h in range(1, 4)} for emotion in ['anger', 'disgust', 'happiness', 'sadness']}
    for emotion in correlation_results.keys():
        for h in correlation_results[emotion].keys():
            if correlation_results[emotion][h]:
                avg_pearson = sum([x[0] for x in correlation_results[emotion][h]]) / len(correlation_results[emotion][h])
                avg_spearman = sum([x[1] for x in correlation_results[emotion][h]]) / len(correlation_results[emotion][h])
                average_correlation_results[emotion][h] = (avg_pearson, avg_spearman)

    return average_correlation_results
def main():
    filePath = "C:\\Users\\范春\\Desktop\\week11\\weibograph.txt"
    G, emotionsDic = generateGraph(filePath)
    collection = shortestPathLength(G)
    average_correlation_results = calculate_average_correlations(collection, emotionsDic, 1000,50)
    results_path = 'C:\\Users\\范春\\Desktop\\week11\\correlation_results.json'
    with open(results_path, 'w') as f:
        json.dump(average_correlation_results, f)
    plt.figure(figsize=(10, 6))
    for emotion in ['anger', 'disgust', 'happiness', 'sadness']:
        plt.plot(range(1, 4), [corr[0] for corr in average_correlation_results[emotion].values()],
label=emotion, marker='o')
    plt.xlabel('Distance (h)')
    plt.ylabel('Pearson Correlation')
    plt.legend()
    plt.savefig('C:\\Users\\范春\\Desktop\\week11\\pearson_correlation.png')
    plt.show()
    plt.figure(figsize=(10, 6))
    for emotion in ['anger', 'disgust', 'happiness', 'sadness']:
        plt.plot(range(1, 4), [corr[1] for corr in average_correlation_results[emotion].values()],
label=emotion, marker='o')
    plt.xlabel('Distance (h)')
    plt.ylabel('Spearman Correlation')
    plt.legend()
    plt.savefig('C:\\Users\\范春\\Desktop\\week11\\spearman_correlation.png')
    plt.show()
if __name__ == '__main__':
    main()
```