北京郵電大學

人工智能原理实验报告

LNG 站点聚类

学　　院　计算机学院

专　　业　计算机科学与技术

班　　级　2020211302

小组成员　鄭毓恒

学　　号　2020211262

小组成员　刘言

学　　号　2020211268

小组成员　李梓硕

学　　号　2020211250

2022 年　12　月

运行环境

硬件信息

- CPU: AMD Ryzen 7 5800H
- 内存条: 2 * 8GB

软件和操作系统

- 操作系统: Windows 11
- 编程语言: Python
- Python 版本: 3.10

操作指令

- 输入以下指令安装 scikit-learn, numpy, pandas
  pip install scikit-learn
  pip install numpy
  pip install pandas

- 输入以下指令运行程序
  python lng_station.py

数据预处理和算法分析

数据预处理

```
cur_dir = os.getcwd()
data_dir = cur_dir + '/data/lng2.csv'
data = pd.read_csv(data_dir, delim_whitespace=True, header=None)
data.columns = ['mmsi', 'time', 'state', 'speed', 'longtitude', 'latitude', 'draft']
data.dropna()
data = data[data['draft'] != 0]
state = data['state'][:]
data = data[(state == 1) | (state == 5) | (state == 15)]
```

以上是数据预处理部分的代码。Csv 文件存储在同一目录的 data 文件夹内，读 csv 至 dataframe 中。首先删除含有空值的行，然后由于船舶吃水度不可能为 0，删除吃水度为 0 的无效数据。根据网上查询，航行状态 1 和 5 分别代表抛锚和系泊，船舶是停止的。同时，航行状态 15 是默认状态，船舶也是静止的。因此，只取航行状态为 1、5 和 15 的数据。

算法分析

经过预处理后，大概还有五百多万条数据。数据中还有许多噪点，而且 LNG 站点和锚点的数量未知。因此，综合时间效率和聚类方法考虑，使用 DBSCAN 算法，以密度为聚类标准。而由于 DBSCAN 算法需要 $O(n^2)$的内存空间，实验硬件不足以直接对 500 万条数据进行聚类，需要先进行分片再聚类。

```
db_start = time.time()
coor_df = data[['longtitude', 'latitude']]
cluster_center = pd.DataFrame(columns=['longtitude', 'latitude'])
i = 0
while i < len(coor_df.index):
    j = 0
    if i + 30000 < len(coor_df.index):
        j = i + 30000
    else:
        j = len(coor_df.index)

    coor_tmp = coor_df.iloc[i:j]
    dbscan_result = DBSCAN(eps=5/6371, min_samples=300, metric='haversine', algorithm='ball_tree',
                           n_jobs=12).fit_predict(np.radians(coor_tmp))
    n_clusters_tmp = len(set(dbscan_result)) - (1 if -1 in dbscan_result else 0)
    for k in range(n_clusters_tmp):
        k_group = coor_tmp[dbscan_result == k]
        mean_lon = k_group['longtitude'].mean()
        mean_lat = k_group['latitude'].mean()
        cluster_center = pd.DataFrame(np.insert(cluster_center.values, len(cluster_center.index), [mean_lon, mean_lat],
                                                axis=0), columns=['longtitude', 'latitude'], dtype=float)
    i = j
```

以上是分片聚类部分的代码。按存储数据的 Dataframe 的索引顺序进行分片，一次为 30000 条数据进行聚类。DBSCAN 函数中的 eps 参数为 5/6371 表示簇范围是 5km，参数 min_sample 表示每簇至少有 300 点，参数 N_jobs=12 表示使用 12 个线程。因为参数 metric 是 haversine，只接受弧度，所以用 np.radians 函数进行变换。每次聚类后，计算每个簇的中心点并保存。

```
dbscan_result = DBSCAN(eps=30/6371, min_samples=1, metric='haversine', algorithm='ball_tree',
                       n_jobs=12).fit_predict(np.radians(cluster_center))
n_clusters_ = len(set(dbscan_result)) - (1 if -1 in dbscan_result else 0)
clusters = pd.DataFrame(columns=['longtitude', 'latitude'])
for k in range(n_clusters_):
    k_group = cluster_center[dbscan_result == k]
    mean_lon = k_group['longtitude'].mean()
    mean_lat = k_group['latitude'].mean()
    clusters = pd.DataFrame(np.insert(clusters.values, len(clusters.index), [mean_lon, mean_lat], axis=0),
                            columns=['longtitude', 'latitude'], dtype=float)
db_end = time.time()
```

完成所有数据的分片聚类后，得到的所有中心点再进行一次 DBSCAN 聚类。这次聚类的目的是合并接近的中心点，所以调到 eps 参数到 30/6371，合并范围 30km 的点。聚类完后再进行一次计算，得到每个簇的中心点，为聚类的最终结果。

```
clusters.insert(cluster_center.shape[1], 'isLNG', False)
clusters.insert(cluster_center.shape[1], 'IN', False)
clusters.insert(cluster_center.shape[1], 'OUT', False)
```

聚类完毕后，还需要给站点进行分类。先插入三列，用于存储后续分类结果。

```
station_data = data[['mmsi', 'longtitude', 'latitude', 'draft']]
pre = station_data.iloc[0]
avg = [pre['longtitude'], pre['latitude']]
count = 0
draft_first, draft_final = pre['draft'], pre['draft']
for index, row in station_data.iterrows():
    cur = row
    if distance([cur['longtitude'], cur['latitude']], [pre['longtitude'], pre['latitude']]) < 30 \
            and cur['mmsi'] == pre['mmsi']:
        avg[0] += cur['longtitude']
        avg[1] += cur['latitude']
        draft_final = cur['draft']
        pre = cur
        count += 1
    else:
        if count > 0:
            avg[0] = avg[0] / (count + 1)
            avg[1] = avg[1] / (count + 1)
            draft_change = draft_final - draft_first

            closest_pt = get_closest_pt(avg, clusters[['longtitude', 'latitude']])
            if closest_pt != -1:
                if draft_change > 5:
                    clusters.at[closest_pt, 'isLNG'] = True
                    clusters.at[closest_pt, 'OUT'] = True
                elif draft_change < -5:
                    clusters.at[closest_pt, 'isLNG'] = True
                    clusters.at[closest_pt, 'IN'] = True
        count = 0
        pre = cur
        avg = [pre['longtitude'], pre['latitude']]
```

对原数据中的每一条进行判断。假如与上一条的距离小于 30km 且 mmsi 一样，视为同一艘船在同一经纬度的数据。计算这些数据的经纬度的平均值得到中心点，然后找到聚类结果中最近的点。根据吃水量变化，给点进行分类。吃水量增加就是出口 LNG，站点是出站；吃水量减少就是入口 LNG，站点是入站。吃水量不变或变化不超过 5，视站点为锚点。

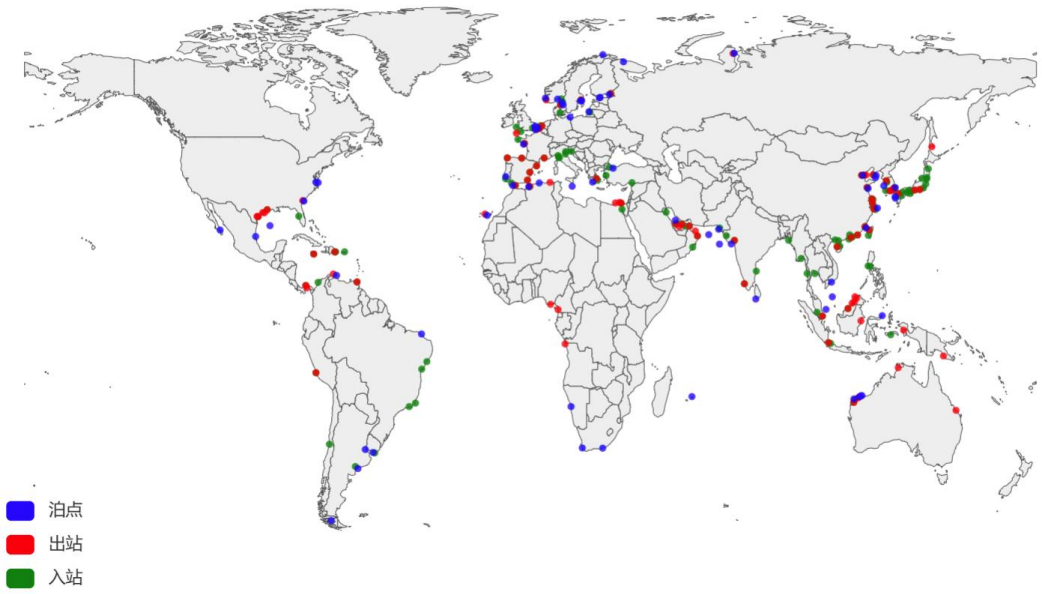
代码运行结果及分析

```
D:\python\sem5hw\LNG>python lng_station.py
LNG Station Number: 157
IN Station Number: 126
OUT Station Number: 84
Mooring Point Number: 67
DBSCAN Clustering Time: 426.4456830024719 second
Total Time: 1118.1872277259827 second
```

程序总共分出了 157 个 LNG 站点，其中有 126 个入站和 84 个出站，以及 67 个锚点。DBSCAN 聚类部分运行时间是 426 秒，程序总运行时间是 1118 秒。


```
[
    {"code": 1, "latitude": 17.838487065044685, "longtitude": -77.09839685424812, "isLNG": true, "IN": true},
    {"code": 2, "latitude": 17.838487065044685, "longtitude": -77.09839685424812, "isLNG": true, "IN": false},
    {"code": 3, "latitude": 47.104516199668495, "longtitude": -2.463455989483638, "isLNG": false, "IN": null},
    {"code": 4, "latitude": 27.880349686005733, "longtitude": -97.26863731972527, "isLNG": true, "IN": false},
    {"code": 5, "latitude": 51.35293744755563, "longtitude": 3.2136503313963614, "isLNG": true, "IN": true},
    {"code": 6, "latitude": 51.35293744755563, "longtitude": 3.2136503313963614, "isLNG": true, "IN": false},
    {"code": 7, "latitude": 21.42732651137711, "longtitude": 109.5226229124564, "isLNG": true, "IN": true},
    {"code": 8, "latitude": 55.50636178944267, "longtitude": 10.533654519201237, "isLNG": true, "IN": true},
    {"code": 9, "latitude": 57.596029284824745, "longtitude": 10.664729987719628, "isLNG": true, "IN": true},
    {"code": 10, "latitude": 57.596029284824745, "longtitude": 10.664729987719628, "isLNG": true, "IN": false},
    {"code": 11, "latitude": 22.807025055714597, "longtitude": 120.18505971709794, "isLNG": true, "IN": true},
    {"code": 12, "latitude": 37.33754705998886, "longtitude": 126.58588981315705, "isLNG": true, "IN": true},
    {"code": 13, "latitude": 37.33754705998886, "longtitude": 126.58588981315705, "isLNG": true, "IN": false},
    {"code": 14, "latitude": 43.469074218677164, "longtitude": -8.229932961078115, "isLNG": true, "IN": true},
    {"code": 15, "latitude": 43.469074218677164, "longtitude": -8.229932961078115, "isLNG": true, "IN": false},
    {"code": 16, "latitude": 51.01798982653901, "longtitude": 1.7385921194821632, "isLNG": false, "IN": null},
    {"code": 17, "latitude": 32.54893697603283, "longtitude": 121.43126654789613, "isLNG": true, "IN": true},
    {"code": 18, "latitude": 32.54893697603283, "longtitude": 121.43126654789613, "isLNG": true, "IN": false},
    {"code": 19, "latitude": 35.41358970486309, "longtitude": 139.78793146161996, "isLNG": true, "IN": true},
    {"code": 20, "latitude": 25.118120053514435, "longtitude": 118.99938884455727, "isLNG": true, "IN": true},
    {"code": 21, "latitude": 25.118120053514435, "longtitude": 118.99938884455727, "isLNG": true, "IN": false},
    {"code": 22, "latitude": 46.412200733460196, "longtitude": 142.76807497705042, "isLNG": true, "IN": false},
    {"code": 23, "latitude": 22.229136148134675, "longtitude": 114.00608254575218, "isLNG": true, "IN": true},
    {"code": 24, "latitude": 22.229136148134675, "longtitude": 114.00608254575218, "isLNG": true, "IN": false},
```


输出的 json 文件内容如上，符合题目要求格式。



使用输出的 json 文件，画出以上地图，标出了 LNG 出入站和船只泊点。从上图可见得到的 LNG 站点分布。经过简单检查，出站基本分布于 LNG 的主要出口国，入站基本分布于主要入口国。

仔细观察地图，还是可以看见一些应当属于同一点的坐标被分为了不同点。原因可能在于 DBSCAN 聚类时的 eps 参数设置太小，但经过测试，假如设置再大一点，得到的 LNG 站点数量减少。虽然确实可以合并同一坐标的点，但也失去了一些正确的站点坐标。

程序的运行时间大概有 20 分钟，其中超过一半是在给聚类结果分类。这是因为分类的代码是一个 for 循环，将循环五百多万次，花费了很多时间。想到的一个改进方法是，不用原数据，而是用分片聚类后，二次聚类之前的数据来判断站点类型，可以将循环次数从百万减少至数万。但是，由于担心这样做会影响 LNG 站点分类的准确性，所以没有采用。

实验总结

经过这次实验我们学到了很多知识，对各种聚类算法有了一定的了解，了解了它们算法的原理和应用。实验过程中，花费了许多时间在于调整 DBSCAN 聚类算法的参数。

一开始我们计划尽可能通过预处理数据，减少数据量，使得程序可以通过一次聚类解决。但这样对参数要求很高，稍有不慎就会导致发生内存不足，而结果精确度也很低。改为分片聚类后，即使可以通过少量数据测试程序，但参数的实际效果还得看在原始数据上的运行结果。每次运行都需要十几二十分钟，才能知道判断是不是理想的参数，花费了很多时间。这次实验是少有的，面对巨大数据量的实验，在编写代码和测试程序时需要更加谨慎，是一次难得的体验，对以后的研究和工作都很有帮助。