About My Solution to Work Sample Problems - Data Role

0. Set Up Docker Spark Env

1.Docker install
The link is: https://docs.docker.com/v17.12/docker-for-mac/install/#download-docker-for-mac
And I choose the **Stable channel**.

2.Get Docker and Spark configurations provided
`git clone https://github.com/EQWorks/ws-data-spark.git`. It is found that the directory name is ws-data-spark.

3.Set up Docker Spark Env
Docker compose is a tool for defining and running multi-container Docker applications. With Compose, a YAML file configures the application services. `docker-compose up -d` starts the containers in the background and leaves them running. Check spark UI in `localhost:8080`
Notice the spark version is `version 2.4.1` with command `spark-submit --version`

4.Go into the container shell (master/work)

```
docker exec -it ws-data-spark_master_1_master_1 /bin/bash
docker exec -it ws-data-spark_master_1_worker_1 /bin/bash
```

Notice that there is no editor in the container, install Vim:

```
apt-get update
apt-get install vim
```

1. Cleanup

`CleanUp.py` is in `eq/data-spark/data/tools`.
Run `spark-submit CleanUp.py` in `/tmp/data/tools` on spark to remove records with identical `geoinfo` and `timest`.
The output file is `/tmp/data/DataSampleClean.csv` on spark.

2. Label

The file position is same as that in **Cleanup** part.
Run `spark-submit Label.py` to label the each request with closest POI.
The output file is `JoinPio.csv`.

3. Analysis

The file position is same as that in **Cleanup** part.
Notice that POI1 and POI2 have the same Latitude and Longitude, so we use POI1 to represent POI1 and POI2.
Run `spark-submit Analysis.py` to calculate the average and standard deviation of the distance between the POI to each of its assigned requests as well as the radius and density (requests/area) for each POI.
The output of average and standard deviation is `AvgDevPio.csv` and `CirclePio.csv`

4. Data Science / Engineering Tracks

4a. Model

DataSet:

We need to have the historical request data within a timespan to analyze the popularity of each POI.

Preprosessing

We need to remove the requests with same time and geo info as in the `Cleanup` part.

Assumption

We assume the popularity depends on the number of requests to the POI out of the number of all the requests within a certain time span. In the same time, we need to divide the time in the dataset into several time span with the same length.

Definition

N is the number of time span

t_i is the i-th time span, $i = 1, \ldots, N$

a_i is the total number requests in time span t_i

M is the number of POI points

b_i is the i-th POI point, $i = 1, \ldots, M$

c_i^j is the number of requests of POI b_i in time span t_j

Pr_i^j is the ratio of requests of POI b_i in time span t_j

$\hat{Pr}_i^j$ is the smoothed Pr_i^j

Po_i^j is the popularity of POI b_i in time span t_j

Model

Given we now assume there is just one POI. Then the popularity of the POI b_i in each time span t_j is:

$$Pr_i^j = \frac{c_i^j}{a_j}, a_j \neq 0$$

Then we apply smoothed kernel method with kernel function as $f(x)$ and $\hat{Pr}_i^j$ becomes:

$$\hat{Pr}_i^j = \frac{\sum_{k=1}^{N} Pr_i^k \cdot f(t_k - t_j)}{\sum_{k=1}^{N} f(t_k - t_j)}$$

Finally we normalize $\hat{Pr}_i^j$ from -10 to 10, as following:

$$Po_i^j = \hat{Pr}_i^j * 20 - 10$$

Bonus

Some reasonable hypotheses regarding POIs as well as assumptions, testing steps and conclusions are included in the file `bonus.txt`.

4b. Pipeline Dependency

All the files related to Pipeline Dependency are in directory `pipeline`.

Algorithm

1. Use dfs to find all the tasks need to be done before goal, in topological order
2. Use dfs to find all the tasks already been done before start.
3. Remove tasks already been done before start from the tasks need to be done before goal The algorithm is implemented in file `pipeline.py`

Results

The solution to the pipeline dependency problem is in the directory `pipeline`. Run `python pipeline.py` to get the results, as following:
`['112', '100', '21', '73', '20', '97', '94', '56', '102', '36']`