DSCI553 Foundations and Applications of Data Mining

Summer 2021

Assignment 4 Clustering

Deadline: June 29, 11:59PM PST

1. Overview of the Assignment

In Assignment 4, you will implement the K-Means and Bradley-Fayyad-Reina (BFR) algorithm. Doing the assignment helps you get familiar with clustering algorithms using various distance measurements. The datasets you are going to use are synthetic.

2. Assignment Requirements

2.1 Programming Language and Library Requirements

a. You must use **Python** to implement all the tasks. You can only use standard Python libraries (i.e., external libraries like numpy or pandas are not allowed). **Spark RDD is optional for Python**. If you want to use Spark, please specify the following environment in your code:

```
os.environ['PYSPARK_PYTHON'] = '/usr/local/bin/python3.6'
os.environ['PYSPARK_DRIVER_PYTHON'] = '/usr/local/bin/python3.6'
```

a. There will be 10% bonus for Scala implementation in each task. **Spark RDD is required for Scala**. You can get the bonus only when both Python and Scala implementations are correct.

b. Spark DataFrame and DataSet are not allowed.

2.2 Programming Environment

Python 3.6, Scala 2.11, and Spark 2.3.0

We will use Vocareum to automatically run and grade your submission. You must test your scripts on **the local machine** and **the Vocareum terminal** before submission.

2.3 Write your own code

Do not share code with other students!!

For this assignment to be an effective learning experience, you must write your own code! We emphasize this point because you may find Python implementations of some of the required functions on the Web. Please do not look for or at any such code!

Plagiarism detection will combine all the code we can find from the Web (e.g., Github) as well as other students' code from this and other (previous) sections. We will report all detected plagiarism to the university.

3. Dataset

Since the BFR algorithm has a strong assumption that the clusters are **normally distributed with independent dimensions**, we generated synthetic datasets by the following steps: 1) Initializing some random centroids 2) For each centroid, sampling data points following the normal distribution in each dimension. Together, the mean of the normal distribution in each dimension is the centroid of the clusters, and the standard deviation is set manually 3) For each centroid, we added some data points as the **outliers.** Figure 1 shows an example of the data points (in CSV format). The first column is the data point indexes. The other columns represent the features/dimensions of the data points.

```
0,54.722990189380965,32.469491844072955,-8.209508911147147
1,-416.4462895782093,-160.55306801341678,-17.198404168038866
2,54.738895724180495,32.74716260306027,-10.145727460163124
3,-27.09232274011507,23.1495267294037,-12.20191767243553
4,57.22493136954117,-217.26570525550395,-235.67658210557272
```

Figure 1: An example of the data points with 3 dimensions

You can access and download the following datasets either under the directory (resource/asnlib/publicdata/) on Vocareum or Google Drive (USC email only): https://drive.google.com/drive/folders/1mZgx9vHpsjwtjwbZGdnVjrVypKoZJaP9?usp=sharing

a. Five folders are named from "test1" to "test5". Each folder contains multiple files storing data points. We will treat these files as separate data chunks. In each iteration, you will load one file (one chunk of points) to the memory and process these data points with the BFR algorithm.

b. Five files are named from "cluster1.json" to "cluster5.json". Each file provides the ground truth cluster for the data points in the corresponding folder from "test1" to "test5". The key is the data point index (as a string). The value is its corresponding cluster index. The cluster index of the outliers is -1. The number of clusters for each dataset is **test1: 10, test2: 10, test3: 5, test4: 8, test5: 15.**

c. We have generated ten testing sets using the same method. Five of them are provided for your implementation. The rest of them are used for grading. **Notice that the number of dimensions, the number of files, and the number of data points in each dataset could vary.**

4. Task (12.5pts)

You need to submit the following files on Vocareum: (all lowercase)

a. [REQUIRED] Python scripts: bfr.py

b. [REQUIRED FOR SCALA] Scala scripts: bfr.scala; one Jar package: hw4.jar

c. [OPTIONAL] You can include other scripts to support your programs (e.g., callable functions).

4.1 Task description

You will write the K-Means and Bradley-Fayyad-Reina (BFR) algorithms from scratch. You should implement K-Means as the in-memory clustering algorithm that you will use in BFR. You will iteratively

load the data points from a file and process these data points with the BFR algorithm. See below pseudocode for your reference.

```
for file in input_path:
    data_points = load(file)
    if first round:
        run K-Means for initialization
    else:
        run BFR(data_points)
```

In BFR, there are three sets of points that you need to keep track of: **Discard set (DS), Compression set (CS), Retained set (RS)**. For each cluster in the DS and CS, the cluster is summarized by:

N: The number of points

SUM: the sum of the coordinates of the points

SUMSQ: the sum of squares of coordinates

The conceptual steps of the BFR algorithm: Please refer to the slides.

The implementation details of the BFR algorithm: Please refer to Section 7.

4.2 Execution commands

Python command: $ python3 bfr.py <input_path> <n_cluster> <out_file1> <out_file2>

Scala command: $ spark-submit --class bfr hw4.jar <input_ path> <n_cluster> <out_file1> <out_file2>

Params <input_path>: the folder containing the files of data points

<n_cluster>: the number of clusters

<out_file1>: the output file of cluster results

<out_file2>: the output file of intermediate results

4.3 Output format

a. You must write your clustering results in the JSON format (see Figure 2). The key is the data point index (as string). The value is its corresponding cluster index. The indexes for both data points and clusters start from 0.

```
{"0": 0, "1": 0, "2": 1, "3": 1, "4": 2}
```

Figure 2: An example of the output clustering results

b. You must output the intermediate results in the CSV format (use the same headers in Figure 3). Each line in the intermediate results represents the following information about each iteration/data trunk:

- round id (starting from 1): # of rounds must be # of data chunks in the folder.
- the number of clusters in the discard set
- the total accumulated number of the discarded points: The number should only go up with iterations.

- the number of clusters in the compression set
- the total number of the compressed points
- the number of points in the retained set

```
round_id,nof_cluster_discard,nof_point_discard,nof_cluster_compression,nof_point_compression,nof_point_retained
1,10,2898,20,147,82
2,10,5326,14,256,15
3,10,7642,10,345,0
...
```

Figure 3: An example of the intermediate results

4.3 Grading

We will use normalized mutual information (NMI) score to evaluate your clustering results. To obtain the full point(1.25pt) for each dataset, the NMI should be **above 0.8,** and the intermediate result is correct. If either criterion (i.e., NMI and intermediate result) does not meet, you will lose all points for a dataset. The submission report will tell you which part of your intermediate results is incorrect. We will grade your code on all ten datasets.

5. About Vocareum

a. Your code can directly access the datasets under the directory: ../resource/asnlib/publicdata/

b. You should upload the required files under your workspace: work/

c. You must test your scripts on both the local machine and the Vocareum terminal before submission.

d. During submission period, the Vocareum will run and evaluate the results for **test1**.

e. You will receive a submission report after Vocareum finishes executing your scripts. The submission report shows **NMI** for each dataset. If the intermediate results are incorrect, the submission shows the reason(s).

f. The total execution time of submission period should be less than 600 seconds. The execution time of grading period need to be less than 3,000 seconds.

g. Please start your assignment early! You can resubmit any script on Vocareum. We will only grade on your last submission.

h. You could add --driver-memory 4g --executor-memory 4g to your spark-submit command to limit its memory usage, in case your code could work properly in your local (with more resources) but would run into memory error. You can also use sc.setSystemProperty('spark.driver.memory', '4g') sc.setSystemProperty('spark.executor.memory', '4g') in your code.

i. Printing out too many outputs (printout) may crash the vocareum, so we recommend to use sc.setLogLevel("OFF") for your submission.

j. You can find a tutorial about Vocareum in the Week1 folder in D2L.

6. Grading Criteria

(% penalty = %penalty of possible points you get)

1. We do not have partial credits. For example, you will get **0** although your result covers 80% answer. **You will also get 0 if your outputs do not follow the format requirements (such as lexicographical order).** There will be no point if the total execution time exceeds the Section 4 evaluation metric.
2. **You are not able to use 5-day extension for this final HW**.
3. There will be 20% penalty for the late submission within one week and no point after that. If you use your late days, there wouldn't be the 20% penalty.
4. There will be a 10% bonus for each task if both your python Scala implementations are correct. **The Scala bonus will not be calculated if your Python results are not correct.** There is no partial point for Scala.
5. There will be no point if your programs cannot be executed on Vocareum Please start your assignment early! You can resubmit on Vocareum. We will grade your last submission.
6. There is no regrading. Once the grade is posted on D2L, we will only regrade your assignments if there is a grading error. No exceptions.
7. There will be no point if your submission falls into the following situations:
    a. The submission cannot be executed on Vocareum. Each task will be run five times and graded on the best run.
    b. The execution failure on Vocareum is because of script naming issues, output file format issues.
    c. The code works on your local but it does not properly on Vocareum.

7. Appendix

The implementation details of the BFR algorithm **(you can/should have your own implementation; this is only for reference).** Suppose the number of clusters is *K* and the number of dimensions is *d*.

1. Load the data points from one file.
2. Run K-Means on the data points or a random subset of the data points. For the implementation, you will apply K-means on a subset since you cannot handle all data points in the first file. Initialize the algorithm with a large number of centroids (e.g., 3 or 5 times of *K*) and use the Euclidean distance as the similarity measurement.
3. Among the result clusters from step 2, move all the clusters that contain only one or very few points (you define "very few") to **RS** as the outliers. You will now have two groups of data points: the outlier data points and inlier data points.
4. Run the clustering algorithm again to cluster the inlier data points into *K* clusters. Use these *K* clusters as your **DS**. Discard these points and generate the **DS** statistics.
5. Run the clustering algorithm again to cluster the outlier data points using a large number of clusters (e.g., 3 or 5 times of *K*). Generate **CS** and their statistics from the clusters with more than one data point and use the remaining as your new **RS**.
6. The above steps finish the initialization of DS. So far, you have *K* **DS** (from step 4), some number of **CS** (from step 5), and some number of **RS** (from step 5).

7. Load the data points from another file (step 2 loads a portion of the first file, load the remaining data points from the first file).
8. For the new data points, compare them to each DS using the Mahalanobis Distance and assign them to the nearest DS clusters if the distance is < $\alpha\sqrt{d}$ (e.g., $2\sqrt{d}$).
9. For the new data points which are not assigned to any **DS** cluster, compare them to each of the **CS** using the Mahalanobis Distance and assign them to the nearest **CS** clusters if the distance is < $\alpha\sqrt{d}$ (e.g., $2\sqrt{d}$).
10. For the new data points which are not assigned to any **DS** or **CS** cluster, add them to your **RS**.
11. Run the clustering algorithm on the RS with a large number of centroids (e.g., 3 or 5 time of ***K***). Generate **CS** and their statistics from the clusters with more than one data point and add them to your existing **CS** list. Use the remaining points as your new **RS**.
12. Merge CS clusters that have a Mahalanobis Distance < $\alpha\sqrt{d}$ (e.g., $2\sqrt{d}$).
13. Output your intermediate result after all the data points in the data file have been evaluated.

    Repeat step 6 to 12.

If this is the last run (after the last chunk of data), merge your CS and RS clusters into the closest DS clusters.