DSCI-553 Foundations and Applications of Data Mining

Spring 2023

Assignment 6 Clustering

Deadline: April 20th 11:59 PM PST

1. Overview of the Assignment

In Assignment 6, you will implement the Bradley-Fayyad-Reina (BFR) algorithm. The goal is to let you be familiar with the process of clustering in general and various distance measurements. The datasets you are going to use is synthetic dataset.

2. Assignment Requirements

2.1 Programming Language and Library Requirements

You must use Python to implement the algorithm. You can only use the following external Python libraries: numpy and sklearn. Using Spark RDDs is optional for this assignment but using it may significantly reduce the time taken by your submission to run on large datasets.

2.2 Programming Environment

Python 3.6, JDK 1.8, Scala 2.12 and Spark 3.1.2

We will use these library versions to compile and test your code. You are required to make sure your code works and runs on Vocareum otherwise we won't be able to grade your code.

2.3 Write your own code

Do not share your code with other students!!

We 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 for plagiarism detection. We will report all the detected plagiarism.

3. Dataset

Since the BFR algorithm has a strong assumption that the clusters are **normally distributed with independent dimensions**, we generated synthetic datasets by initializing some random centroids and creating some data points with the centroids and some standard deviations to form the clusters. We also added some other data points as outliers in the dataset to evaluate the algorithm. Data points which are outliers belong to clusters **that are named or indexed as "-1"**. Figure 1 shows an example of a part of the dataset. The first column is the data point index. The second column is the name/index of the cluster that the data point belongs to. The rest columns represent the features/dimensions of the data point.

```
2708,0,2.438665943891622,1.027818493861439
2709,4,127.31966592930607,148.31411942629745
2710,4,133.61619393051998,157.97961951108087
2711,3,-91.58594502448611,-119.9550977954772
2712,4,135.69634514856796,177.55732400164032
2713,0,12.108343663244053,1.4969275135468427
2714,-1,467.7774695478297,670.5177459229164
2715,3,-104.62798147039325,-128.2723232272566
2716,1,-52.994607584888456,64.12806862225437
2717,4,144.02609145460397,177.36349742076746
2718,4,151.4896164272161,170.51611070365897
```

Figure 1: An example of the dataset

a. hw6_clustering.txt is the synthetic clustering dataset. The dataset is available on Vocareum (public data folder).
b. We generate the testing dataset using a similar method. **Notice that the number of the dimensions could be different from the hw6_clustering.txt.** We do not share the testing dataset.

4. Task

You will implement the Bradley-Fayyad-Reina (BFR) algorithm to cluster the data contained in **hw6_clustering.txt**.
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 slide for details):

Implementation details of the BFR algorithm: (just for your reference, the number of input clusters = n_cluster parameter given as input)

Step 1. Load 20% of the data **randomly**.

Step 2. Run K-Means (e.g., from sklearn) with a large K (e.g., 5 times of the number of the input clusters)

on the data in memory using the Euclidean distance as the similarity measurement.

Step 3. In the K-Means result from Step 2, move all the clusters that contain only one point to RS (outliers).
Step 4. Run K-Means again to cluster the rest of the data points with K = the number of input clusters.

Step 5. Use the K-Means result from Step 4 to generate the DS clusters (i.e., discard their points and generate statistics).

The initialization of DS has finished, so far, you have K numbers of DS clusters (from Step 5) and some numbers of RS (from Step 3).

Step 6. Run K-Means on the points in the RS with a large K (e.g., 5 times of the number of the input clusters) to generate CS (clusters with more than one points) and RS (clusters with only one point).

Step 7. Load another 20% of the data **randomly**.

Step 8. For the new points, compare them to each of the DS using the Mahalanobis Distance and assign them to the nearest DS clusters if the distance is < $2\sqrt{d}$.

Step 9. For the new points that are not assigned to DS clusters, using the Mahalanobis Distance and assign the points to the nearest CS clusters if the distance is < $2\sqrt{d}$

Step 10. For the new points that are not assigned to a DS cluster or a CS cluster, assign them to RS.

Step 11. Run K-Means on the RS with a large K (e.g., 5 times of the number of the input clusters) to generate CS (clusters with more than one points) and RS (clusters with only one point).

Step 12. Merge CS clusters that have a Mahalanobis Distance < $2\sqrt{d}$.

Repeat Steps 7 – 12.

If this is the last run (after the last chunk of data), merge CS clusters with DS clusters that have a Mahalanobis Distance < $2\sqrt{d}$.

At each run, including the initialization step, you need to count and output the number of the discard points, the number of the clusters in the CS, the number of the compression points, and the number of the points in the retained set.

Input format: (we will use the following command to execute your code)

/opt/spark/spark-3.1.2-bin-hadoop3.2/bin/spark-submit task.py <input_file> <n_cluster> <output_file>

Param: input_file: the name of the input file (e.g., hw6_clustering.txt), including the file path.
Param: n_cluster: the number of the clusters.
Param: output_file: the name of the output txt file, including the file path.

Output format:

The output file is a text file, containing the following information (see Figure 2):

a. The intermediate results (the line is named as "The intermediate results"). Then **each line should be started with "Round {i}:" and i is the count for the round (including the initialization, i.e., initialization would be "Round 1:".** You need to output the numbers **in the order of** "the number of the discard points", "the number of the clusters in the compression set", "the number of the compression points", and "the number of the points in the retained set".

Leave one line in the middle before writing out the cluster results.

b. The clustering results (the line is named as "The clustering results"), including the data points index and their clustering results after the BFR algorithm. The clustering results should be in **[0, the number of clusters)**. The cluster of outliers should be represented as -1.

```
The intermediate results:
Round 1: 2898,10,20,82
Round 2: 5236,6,147,17
Round 3: 7566,4,234,0
Round 4: 9694,3,306,0
Round 5: 9998,1,2,0
...

The clustering results:
0,1
1,1
2,0
3,0
4,-1
5,1
6,0
7,1
...
```

Figure 2: the output example for text file

Grading:

a. We will compare the percentage of discard points after the last round to our threshold.

Dataset	Percentage of discard points after last round
hw6_clustering.txt	98.5%

b. We will compare the centroids of the clusters you find to those in the ground truth.

c. We will compute the accuracy of your clustering results to the ground truth. We will check the percentage of data points that you issue to the correct cluster (except the outliers) using the following formula:

$$Accuracy = \frac{the\ total\ number\ of\ points\ in\ the\ correct\ clusters}{the\ total\ number\ of\ points\ in\ the\ ground\ truth}$$

Dataset	Accuracy
hw6_clustering.txt	98.0%

d. No point if your runtime is more than or equal to 600 seconds.

Execution format:

Task:

/opt/spark/spark-3.1.2-bin-hadoop3.2/bin/spark-submit task.py <input_file> <n_cluster> <output_file>

/opt/spark/spark-3.1.2-bin-hadoop3.2/bin/spark-submit --class task --executor-memory 4G --driver-memory 4G hw6.jar <input_file> <n_cluster> <output_file>

5. Submission

You need to submit following files on **Vocareum** with exactly the same name:

a. One Python script, named: (all lowercase)

task.py

b. [OPTIONAL] One Scala script and a jar file, named: (all lowercase)

1. task.scala
2. hw6.jar

c. You don't need to include your results or the datasets. We will grade on your code with our testing data (data will be in the same format).

6. Grading Criteria

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

1. You can use your free 5-day extension separately or together but not after one week after the deadline. [Google form link: https://forms.gle/edH8jw1mJjrLFRcm8]
2. There will be a 10% bonus for correct Scala implementation provided the python implementation works correctly.
3. If we cannot run your programs with the command we specified, there will be no regrading.
4. If your program cannot run with the required Python versions, there will be a 20% penalty.
5. There will be a 20% penalty for late submissions within a week and no point after a week.