CSCI 4372/5372/6397: Data Clustering

Phase 2: Normalization and Initialization

Submission Deadline: **October 27 (23:59:59)**

Objective: Normalize your data and implement an alternative initialization method for k-means.

Part I – Normalization: In clustering applications, attribute normalization is a common preprocessing step that is necessary to prevent attributes with large ranges from dominating the distance calculations and to avoid numerical instabilities in the computations. In this phase, you will implement the "min-max normalization" method to normalize your attributes prior to clustering. You can easily find the description of this simple normalization method in any Data Mining book, or on the WWW, for example, from here
https://www3.nd.edu/~rjohns15/cse40647.sp14/www/content/lectures/07%20-%20Data%20Transformation.pdf

Warning: Do **not** normalize your input data matrix across the rows. It should be normalized across the columns. In other words, each attribute of the data set should be normalized independently of other attributes.

Bonus for CSCI 4372 Students [10 points]; Mandatory for CSCI 5372 and 6397 Students: In addition to "min-max normalization", implement the "z-score normalization" method.

Hint: When implementing either normalization method, you should **avoid** divisions by zero.

Part II - Initialization: In Phase 1, you implemented the standard batch k-means algorithm. Unfortunately, k-means is highly sensitive to initialization. In other words, different initial centers can result in vastly different results. Numerous initialization methods have been proposed to address this problem. In this phase, you will implement another initialization method for k-means and compare it with the "random selection" method that you implemented in Phase 1. You will compare the two initialization methods with respect to the following factors:

- **Initial SSE**: This is the SSE value calculated after the initialization phase, before the clustering phase. It gives us a measure of the effectiveness of an initialization method by itself.
- **Final SSE**: This is the SSE value calculated after the clustering phase. It gives us a measure of the effectiveness of an initialization method when its output is refined by k-means. Note that this is the objective function of the k-means algorithm.
- **Number of Iterations**: This is the number of iterations that k-means takes until reaching convergence when initialized by a particular initialization method. It is an efficiency measure independent of programming language, implementation style, compiler, and CPU architecture.

In Phase 1, you implemented the "random selection" method. In this phase, you will implement the "random partition" method. In this method, starting from empty clusters, first assign each point to a cluster selected uniformly at **random**. You then take the **centroids** of these initial clusters as the initial centers.

Bonus for CSCI 4372 and 5372 Students [10 points]; Mandatory for CSCI 6397 Students: In addition, to "random partition", implement the "maximin method". This method chooses the first center **arbitrarily** from the data points and the remaining (K − 1) centers are chosen successively as follows. In iteration i (i = 2, 3, …, K), the i-th center is chosen to be the point with the greatest squared Euclidean distance to the nearest previously selected (i − 1) centers. In other words, center 2 is chosen to be the farthest point to center 1. Center 3 is chosen to be the point with the greatest distance to the nearer of centers 1 and 2, that is, point **x** with the greatest min (d(**x**, $\mathbf{c}_1$), d(**x**, $\mathbf{c}_2$)) value, where $\mathbf{c}_1$ and $\mathbf{c}_2$ are the first, and second centers, respectively, 'd' is the squared Euclidean distance, and the function "min (a, b)" returns the smaller of 'a' and 'b'. Center i is chosen to be the point **x** with the greatest min (d(**x**, $\mathbf{c}_1$), d(**x**, $\mathbf{c}_2$), …, d(**x**, $\mathbf{c}_{i-1}$)) value. By using additional memory, maximin can be implemented in O(NDK) time, where N, D, and K respectively denote the numbers of points, attributes, and clusters. A naïve implementation, however, requires $O(N^2DK)$ time. If you go for a naïve implementation, you **are** going to have difficulty completing your runs.

Hint: 'Arbitrary' does **not** necessarily mean 'random'.

Hint: Maximin works with both (ordinary) Euclidean and squared Euclidean distances. In both cases, the result is the same. Therefore, there is **no** good reason to use the ordinary Euclidean distance (recall that 'sqrt' is an expensive operation).

If your initialization method is randomized, execute it <R> times and tabulate the best result for each performance measurement separately (best initial SSEs, best final SSEs, and best # iterations) for each data set. Otherwise, a simple table of the data sets and the corresponding measurements will be sufficient. There are many ways to tabulate this kind of data; try to find an intuitive way.

Output: Microsoft Excel or Apache OpenOffice Calc table(s) of the performance measurements for each combination of normalization and initialization methods. In other words, {10 data sets} x {1 or 2 normalization methods} x {2 or 3 initialization methods} x {3 performance measures} = 60 or 180 values. Of course, you can split this table into smaller tables. For example, you can put 6 or 18 values into one table that represents a particular data set.

Language: C, C++, or Java. You may **only** use the built-in facilities of these languages. In other words, you may **not** use any external libraries or APIs.

Submission: Submit your source code and output file(s) through Blackboard. Do **not** submit your entire project folder or the input data files that I provided to you (I already have them 😊). In addition, do **not** submit your files individually; pack them in a single archive (*e.g.*, zip) and submit the archive file.

Grading: *Functional correctness* and *adherence to good programming practices* are respectively worth **90%** and **10%** of your grade. Given a valid input, a functionally correct program is one that produces the correct output. There are a lot of generic or language-specific guides on good programming practices on the WWW. You should identify an appropriate one, include its URL at the top of your program source code(s), and use it throughout this project. In general, you should pay more attention to structural issues (*e.g.*, modularity) than cosmetic ones (*e.g.*, naming and formatting). Keep in mind that if your program is incorrect, whether or not you adhered to good programming practices is immaterial.