INF 553 – Homework #4

Hierarchical Clustering

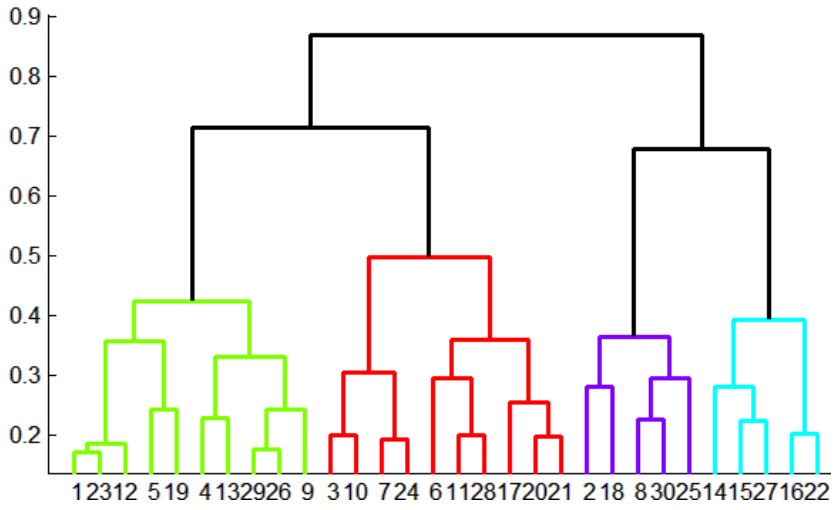
Due: 11/9/2015, Monday, 11:59pm, to Blackboard

In this assignment, you are asked to implement a hierarchical agglomerative clustering algorithm. As shown in class, the algorithm starts by placing each data point in a cluster by itself and then repeatedly merges two clusters until some stopping condition is met.

- **Clustering process:** Your algorithm should stop the clustering process when all data points are placed in a single cluster. That is, the algorithm will perform n – 1 merging if there are n data points.

  Your algorithm should also keep track of the merging process: for each merging step, remember which clusters are merged to produce which new cluster. Such information is similar to a dendrogram (e.g., shown below), except that dendrogram also remembers the distance of two clusters when they are merged.

  With the above information, your algorithm can then allow users to specify a desired number of clusters k, and returns k clusters based on the clustering result. For example, to produce two clusters, the algorithm can simply return the last two clusters that were merged during the process.



- **Distance function:** You may assume that the input to the algorithm consists of a set of n-dimensional data points in the Euclidean space. The distance between two clusters is measured by the Euclidean distance of their centroids.

- **Priority queue:** The algorithm should use a priority queue to store the distance between each pair of clusters and find the two clusters with minimum distance efficiently. You are provided with an implementation of a generic priority queue and required to use it in your algorithm. You are not supposed to modify the provided implementation.
  Recall that during the merging process, distance entries in the queue for the clusters that are merged need to be removed from the queue. However, it may be hard to keep track of entries in the queue that involves a particular cluster, since the queue is undergoing frequent changes. To address this, your algorithm should implement a lazy removal strategy: entries for merged clusters are not removed until they show up in the root of the queue. When such entries are found at the root during the ExactMin operation, they are simply ignored, and ExactMin continues until a pair of clusters that do not contain merged clusters is found at the root of queue, or the queue is exhausted. Note that in this strategy, you do not need to implement a remove function that removes an arbitrary node from the heap.
- **Evaluation:** For the evaluation purpose, the data set input to your algorithm also contains cluster label for each data point. These cluster labels form a gold standard for clustering the data set to produce a specific number of clusters. Note that you can not use the label in the clustering process. Instead, use them in evaluating the performance of your algorithm. The performance is measured by precision and recall of correct pairs. A pair of two data points x and y is correct if they belong to the same cluster according to the cluster label. Recall that precision and recall are discussed in LSH lectures. Precision is the percentage of pairs discovered by the algorithm that are correct, while recall is the percentage of correct pairs that are discovered by the algorithm.
  As an example, consider five data points: 1, 2, 3, 4, 5. Suppose there are 2 clusters: {1, 2, 3} and {4, 5} according to the algorithm, while different 2 clusters: {1, 2} and {3, 4, 5} according to the gold standard. In this case, the algorithm discovers four pairs: (1, 2), (1, 3), (2, 3), and (4, 5), while the gold standard has the pairs: (1, 2), (3, 4), (3, 5), and (4, 5). In this case, precision is 2/4 since only (1, 2) and (4, 5) discovered by the algorithm are correct, among the 4 discovered. Recall is also 2/4, since only 2 correct pairs were discovered among the total 4 in the gold standard.

Input data format:

The data set contains a list of data points, one point per line. For each data point, it gives the value of the point at each dimension, followed by the cluster label of the point. A sample data point is provided for you to test your algorithm. The data contains 150 iris plants (https://archive.ics.uci.edu/ml/datasets/Iris). For each plant, it lists its sepal and petal length and width in centimeter, and also its type (e.g., setosa, see below). 萼片 花瓣

```
5.1,3.5,1.4,0.2,Iris-setosa
4.9,3.0,1.4,0.2,Iris-setosa
4.7,3.2,1.3,0.2,Iris-setosa
…
```

You may assume that input data sets that will be used to test your algorithm will have similar format, that is, one data point per line, which has value of point in each of n dimensions (where n >= 1), followed by a class label.

Input and output format:

Your algorithm should take two arguments: a text file name for the input data, and a value k for the number of desired clusters. For example,

$python hclust.py iris.dat 3

Where hclust.py is your hierarchical clustering algorithm, iris.dat is the input data file, and 3 is the k value.

Your algorithm should output 3 clusters, with each cluster contains a set of data points. Data point are numbered by their positions they appear in the data file: the first data point is numbered 0, second 1, and so on. The data points in the clusters are output in the ascending order of their numbers.

For example, here is an example output.

Cluster 1: [3, 10, 13, …]

Cluster 2: [8, 52, 87, 88, …]

Cluster 3: [100, 105, …]

Your algorithm also output the accuracy of the discovery. For example,

Precision = .8, recall = .5

An example output will be provided later. Be sure to follow its format.

Priority queue module:

Please find heapq.py provided to you. It implements a priority queue with operations for building the heap, adding items, and extracting smallest item from the heap. See the documentation in the script and also the link (https://docs.python.org/2/library/heapq.html) for more details. Note that you are not supposed to modify the provided heap script.

Note that the construction of your heap should take linear time, i.e., time linear to the number of nodes (i.e., one for every distinct pair of clusters) in the heap.

Submission:

Submit a Python script to Blackboard by the deadline. Name your script as follows:

Firstname_lastname_hclust.py

Important notes:

- Finish the homework independently. Do not copy codes from others or the Web. Do not share your code with others. We will use MOSS to check your submissions. There will be serious consequences, should we detect the plagiarism.
- Follow the script naming convention, the input and output formats as described above.
- Do not write any output to a file. Write all outputs to standard output instead.

- Make sure your code compiles before you submit.