PA10: K-Means Clustering

This programming assignment is to be completed in either Java or Python (your choice) and submitted on Gradescope.

Collaboration is restricted to just 2 people (other than yourself), and it must be "Whiteboard Only". In summary, this means that you may discuss the assignment with 2 others, but you cannot preserve any artifacts (digital or physical) from these discussions. In particular, this means you may not take and keep photos, notes, video, audio recordings, whiteboard contents, etc. from any discussions.

All collaborators and other resources consulted should be listed as a comment at the top of your code submission. Course staff and official course materials do not need to be listed.

Motivation

For PA6 you implemented a clustering algorithm that involved minimum spanning trees. For that clustering algorithm our goal was to maximize the amount of distance between our clusters. That is, given a set of points, we wanted to arrange them into clusters so that the distance between the closest two clusters is maximized. One issue with this method of clustering, however, is that if we receive additional points in the future it is difficult to assign them to a cluster. For this programming assignment we will implement an algorithm called k-means clustering. This clustering method has a more clear application for predicting which cluster future points would belong to.

Here's a reminder of our motivating example for clustering. The following test is quoted from PA6.

> Suppose I had a bunch of photos of Labradors (a dog breed) that I wanted to separate into black labs, yellow labs, and chocolate labs (different varieties of Labradors different only by

coat color). If I manually pre-labelled these photos with the correct variety then I could use them to train a supervised learning algorithm to label future photos with the correct variety of Labrador.

An unsupervised model, on the other hand, does not require any pre-labelling of the training data. Instead, unsupervised models attempt to find patterns in the training data. For example, if I had photos of Labradors we might be able to plot these photos as points in some cartesian space (as a simple example, we might have a 3-dimensional point per picture which represents the RGB of the coat color). With these points, an unsupervised model might identify clusters of points that are similar to one another and conclude that points in the same cluster must be the same Labrador variety.

Problem Description

For this problem set we will implement k-means clustering. Consider that we have n items in our data set. The input to your algorithm is going to be a list of n 2-dimenstional points (i.e. x,y coordinates), and an integer k. This clustering algorithm will form the items into k clusters based on their distance to some central point. That is, we want to select some "good" set of k reference points, and then cluster the points based on which reference point each is closest to. We will pick "good" reference points by picking an arbitrary set of reference points initially, doing a clustering using this choice, improving our reference points, and then repeating. In summary, the k-means clustering algorithm behaves as follows:

1. Given a list of n points and a number k, first select the first k points in the list as reference points.
2. Next cluster the points according to which of the k reference points each is closest to.
3. Next find the centroid (i.e. the average point) of each cluster. Use these as your new reference points.
4. repeat steps 2-3 until your reference points do not change between iterations.

Input: A number k indicating the number of clusters to break the items into and a list of n pairs of doubles representing the points

Output: Your algorithm should return a 2-dimensional list of points (i.e. a list of list of points) to represent the clustering. The outer list should be of length k, and each inner list should contain the points that are in the same cluster.

Starter Code Summary

The starter code provided consists of two files:

– `KMeansCluster.java`/`k_means_cluster.py`: This is the program you will modify and submit. Ultimately your goal is to have a correct implementation of the `k_means` function, which receives an integer k indicating the number of clusters, and a list of 2-dimensional points. It returns a list of list of points indicating the clusters (so the outer-list must be of length k). Additional functions are provided for you. You're welcome to use them or not, but I found them helpful in my implementation, so I recommend you use them.

  - `distance` Returns the euclidean distance between two points. I strongly recommend that you use this function unchanged.
  - `centroid` Returns the centroid of a set of points. To compute the centroid, return a point whose x coordinate is the average of the x coordinates from the input set whose y coordinate is the average of the y coordinates.
  - `cluster` Given a set of reference points and the set of points it will split up the given points according to which reference point it is closest to.
  - texttk_means As mentioned, this is the function you must implement. Do not change the signature of this function. Additionally, there is currently a line in this function which establishes the first k points in the list as the initial reference points. Do not start with a different set of points, this may cause for you to get a different clustering compared to the autograder.

- `ReadInput.java/read_input.py`:You should not edit or submit this program. The autograder will ineract with your program in the same way that this one does, but it will behave differently. Currently this program reads the input from the command line, calls your programs `k_means` function, then prints the clusters you gave it.
- `Point.java`: This program exists for java only. It defines a point class which has two double parameters called `x` and `y`. You should not submit this file.

Input Format

Input will be given in the console (same as with the previous PAs). To provide an input with n points, you will have $n + 2$ lines in your input as follows:

- The first line contains the value of k, i.e. the number of clusters to break the items into.
- The second line contains the value of n, i.e. the number of items.
- The remaining n lines each contain a space-separated pair of floats/doubles which indicate the x,y coordinates of a point to be clustered

Here is an example input:

```
3
15
0 0
20 3
14 6
17 20
20 22
18 18
18 5
15 10
2 1
1 2
1 1
15 18
19 20
13 14
1 3
```

If you give this as input to the input reader program a correct answer would look like this (note that your order of points and clusters may be different, but not which points are vs. are not in the same cluster).

```
cluster 0:
   (0.0, 0.0)
   (2.0, 1.0)
   (1.0, 2.0)
   (1.0, 1.0)
   (1.0, 3.0)
cluster 1:
   (20.0, 3.0)
   (14.0, 6.0)
   (18.0, 5.0)
   (15.0, 10.0)
cluster 2:
   (17.0, 20.0)
   (20.0, 22.0)
   (18.0, 18.0)
   (15.0, 18.0)
   (19.0, 20.0)
   (13.0, 14.0)
```