Homework 10: K-Means Clustering

Due: 23 April, 2021 11:59pm ET

This homework asks you to fill in portions of classes that you can then use to perform k-means analysis.

Goals

In this assignment you will:

- Get familiar with using objects and classes by defining some methods and using objects to perform a computation
- Implement k-means

Background

Classes and Objects

Please see the class notes on classes and objects.

k-means

Please see the class notes on clustering and specifically k-means.

Instructions

0) Set up your repository for this homework.

Use the link on Piazza to set up Homework 10.

The repository should contain the following files:

1. This README.
2. `cluster.py` which contains the definition of the `Cluster` class and some testing code.
3. `point.py` which contains the definition of the `Point` class and some testing code.
4. `kmeans.py` which contains the skeleton of the kmeans algorithm and some testing code.

1) Homework Problem 1: Complete Point class

Complete the missing portions of the `Point` class, defined in `point.py`:

1. `distFrom`, which calculates the (Euclidean) distance between the current point and the target point. Be sure to account for the fact that a point may be in more than two dimensions (Euclidean distance generalizes: square the difference in each dimension and take the square root of the sum). It is okay to use `math.sqrt()` to calculate the square root.
2. `makePointList`, which takes in a data p-by-k input matrix `data` and returns a list of p `Point` objects. Hint: Instantiate a point object for every row in the input, `data`.

If you test your code by running `python3 point.py`, you should get the following:

```
[Point: [0.5 2.5], Point: [0.3 4.5], Point: [-0.5  3. ], Point: [0.  1.2], Point: [10. -5.],
Point: [11.  -4.5], Point: [ 8. -3.]]
2.009975124224178
```

(Your floating point numbers may be a little off due to rounding)

2) Homework Problem 2: Complete Cluster class

Complete the missing portions of the `Cluster` class, defined in `cluster.py`:

1. `avgDistance`, which computes the average distance from the center of the cluster (stored in `self.center`) to all of the points currently in the cluster (stored in `self.points`). This can most easily be done by summing the distances between each point and the current center and then dividing the sum by the total number of points.
2. `updateCenter`, which updates the center of the cluster (stored in `self.center`) to be the average position of all the points in the cluster. Note that if there are no points in the cluster, you should return without updating (i.e., if there are no points, just type the command `return`.

Note that we have defined `dim` and `coords` as properties that return information about the center of the cluster -- this means that if you pass a cluster into a method that is expecting a point, operations that access `dim` and `coords` will use the center of the cluster. Think about how that might be useful in conjunction with the `closest` method defined for `Point`.

If you test your code by running `python3 cluster.py`, you should get the following:

```
Cluster: 0 points and center = [0.5, 3.5]
Cluster: 2 points and center = [0.5, 3.5]
1.4976761962286425
Cluster: 2 points and center = [1.75, 2.75]
0.3535533905932738
```

(Your floating point numbers may be a little off due to rounding)

3) Homework Problem 3: Implement k-means

Use the methods in `Point` and `Cluster` to implement the missing `kmeans` method in `kmeans.py`. The basic recommended procedure is outlined in `kmeans.py`.

If you test your code by running `python3 kmeans.py`, you should get the following:

```
Cluster: 4 points and center = [0.075 2.8  ]
   [0.3 4.5]
   [0.  1.2]
   [0.5 2.5]
   [-0.5  3. ]
Cluster: 3 points and center = [ 9.66666667 -4.16666667]
   [ 8. -3.]
   [10. -5.]
   [11.  -4.5]
```

(Your floating point numbers may be a little off due to rounding)

What you need to submit

Push your completed versions of `kmeans.py`, `cluster.py`, and `point.py`.