Module 5: K-means Clustering

JHU EP 606.206 - Introduction to Programming Using Python

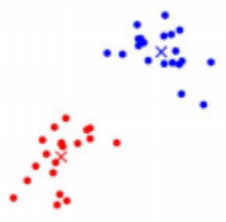
Introduction

In this assignment you will perform data analysis through a basic, unsupervised machine learning approach known as “k-means clustering”. k-means clustering is a way to group objects together that are somehow similar to each other. It is a method that allows you to make inferences about the data based on locality and requires no prior knowledge of correct outcomes. For example, consider the outbreak of the COVID-19 virus. It may help to think of the initial inputs given below as the locations of 4 individuals who are known to be infected. This week we will perform clustering analysis to determine from whom the 96 other patients are most likely to have contracted the virus based on proximity.

Skills: lists, tuples, file I/O, control structures, string functions

K-means Clustering

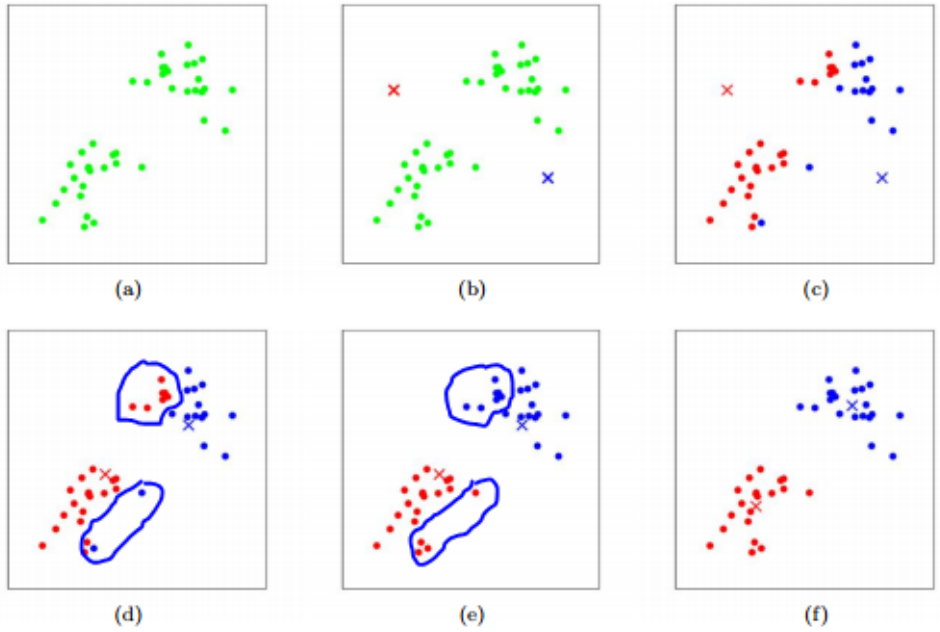
Most often you will begin with a set of points of size N. The 'k' in k-means refers to the number of clusters into which you would like to partition the N points. Here's an example of 2-means clustering:



The algorithm for performing k-means clustering, with k = 2, is as follows:

1. Select (usually at random) k points to serves as "centroids" (cluster centers)
2. For each point in the dataset, determine which of the k centroids the point is closest to by finding the Euclidean distance and add it to that cluster
3. Compute the new centroid for each of the k clusters by calculating the mean point (hence the name "k-means")
    a. $mean_x$ = sum(x-values for all points)/number of points in cluster
    b. $mean_y$ = sum(y-values for all points)/number of points in cluster
    c. $centroid_{new}$ = $mean_x$, $mean_y$
4. Re-cluster by repeating steps 2 and 3 until convergence is achieved
    a. When updating the centroids by calculating the cluster means in Step 3, it is possible for points to switch clusters when we repeat Step 2 again; that's perfectly fine
    b. We iterate through the process until no more points switch clusters. When we reach this final state we say we have achieved "convergence" or "stability" and no further iteration is needed as it will not improve the accuracy of the clustering
    c. Once stable, we can measure a cluster's accuracy by calculating the mean distance of each point in a cluster to its centroid
5. Repeat the entire process of Steps 1-4 multiple times selecting a new, random pair of initial centroids each time. We can then compare the accuracy of each iteration to determine which is most accurate if desired.

K-means Clustering Example



a. All points are initially unassigned
b. Select two centroids randomly, the red and blue x's in in this case
c. Cluster all points geometrically closest to the red 'x' (red) and blue 'x' (blue)
d. Calculate the new position of each centroid using the mean
e. Re-cluster based on new centroids
    a. Notice the circled points which change clusters from steps d-e
    b. I apologize for the awful drawing of the circles!
f. Re-calculate centroid since points changed clusters in step e
    a. If only visually, the clusters are stable and iteration halts

Input File Format

The input files **(points1.txt, points2.txt)** will consist of the maximum number of clustering iterations at the beginning of the input file, followed by N, the total number of points (patients) in the input file, followed by k, the number of clusters (initially infected patients), followed by k centroids (initially infected patients), followed by N-k patients' locations, each on a separate line. The coordinates for all points are integers and are separated by commas. When reading the data in from the 2 files you may find string methods such as strip() and split() useful.

Here is a (truncated) overview of the file contents for **points1.txt**. The comments are provided here for clarity but are not included in the actual input files.

```
50  # max. number of iterations
100 # number of patients in input file (COVID-19 patient locations; 4 initially infected (initial cluster centroids), 96 who contracted the virus from those 4, for a total of 100)
4 # total number of clusters (k, the number of initially infected patients)
30,45 # cluster centroid 1 (the first of the 4 initially infected patients)
55,82 # cluster centroid 2 (the second of the 4 initially infected patients)
1,61 # cluster centroid 3 (the third of the 4 initially infected patients)
96,14 # cluster centroid 4 (the fourth of the 4 initially infected patients)
83,13 # the remaining 96 (N = 100 minus k = 4) patients of the form x,y
…
…
81,32
```

NOTE: Based on questions we've received in the past we wanted to provide clarification on the 100 patients. We know the total number of patients and the number of initially infected patients so we can very easily compute the number of remaining patients to run our clustering algorithm on. You will essentially be splitting them into two groups:

1. The 4 earliest known infected patients (centroids)
2. The 96 patients we want to learn who they were most likely infected by

The approach we took in our solution was to separate them out into 2 lists: one list of the 4 patients which will serve as our initial centroids and a second list containing the other 96 patients. This is why, as you'll see below, our final clusters contain 96 patients and not 100 for points1.txt and 46 patients instead of 50 for points2.txt. That is, we DO NOT consider the 4 earliest infected patients in our clustering algorithm as anything other than our initial centroids. It's a tiny bit wonky from the data science side but hopefully is a fun way to get the Python concepts across.

K-means Clustering Assignment

The goal of this assignment is to try to figure out from which of the 4 initial COVID-19 patients the other 96 are most like to have contracted the virus. You will read a set of parameters as defined in the "Input File Format" section above as well as a number of points from a file and perform naïve k-means clustering on them. Your final output will include the number of iterations required to achieve convergence and, for each cluster, the final location of its centroid, the number of points in the cluster, and a list of points contained in that cluster (examples provided in the "Outputs" section). Please note this is just a helpful guide to get you started. Correctness will primarily be based on your outputs (defined below), but your individual implementation may vary.

We mentioned last week that it's a good idea to write generic code that works on multiple inputs (as long as the input format is the same). This week you will run your code on multiple input files: **points1.txt** and **points2.txt**. They contain different values for k (clusters) and N (number of points). **You will want to write your code in such a way that it can be run against both input files without making any changes**.

In a file named **kmeans.py**, please do the following:

1. Begin by opening kmeans.txt for reading the first 3 fields line-by-line. You'll then want to use the values to dynamically build your data structures for things like your clusters and their corresponding centroids in Step 2. A tuple or list will make the most sense but you'll want to think about the differences between before choosing. There is a solution involving both, though, so feel free to get started using whichever data structure you think makes the most sense. If at some point you decide you want to switch from one data structure to the other, it should only require minimal changes to your code, so no worries there.
2. Create any data structures necessary to store your points, clusters, previous cluster sizes, etc.
3. Create a processing loop to do the following:
    a. For each point in your list:
        i. Compute the distance between the current patient and each of the k centroids
        ii. Add the patient to the cluster to which the nearest centroid belongs
    b. For each cluster, check its size from the previous iteration (see Section c.iv) against the size of the clusters computed directly above in step (a)
        i. If any of the cluster sizes changed from the previous iteration, increment a variable that counts the number of iterations required to achieve convergence
    c. For each cluster:
        i. Compute the mean of all x values
        ii. Compute the mean of all y values
        iii. Update the cluster's centroid with the results from (i) and (ii)
        iv. Store/update the current size of each cluster in a list
            1. These values will be checked during each subsequent iteration to determine if any points changed clusters in step 2b.
    d. If it is not the final iteration of the outermost loop:
        i. Clear the contents of each cluster (they will be refilled in the next iteration)
4. Once complete, you will print your results according to the output format specified below in the Output Format section.

Pseudocode

Here is some pseudocode to help you get started (but you are free to use any approach you want):

```
# read and parse fields from input file and initialize variables
  -  Iterations
  -  Number of total patients (N)
  -  Number of clusters (k)
  -  Number of patients for clustering (N-k)

# create any necessary data structures
  -  Patients (unnested data structure)
  -  Clusters (nested data structure)
  -  Previous cluster sizes (unnested data structure)
  -  Centroids (nested data structure)

# for number of iterations in input file
  # for each patient in the input file
    # calculate distance between current patient and all k centroids
    # assign patient to the cluster with the nearest centroid

  # check if any patients have changed clusters
    # update the iteration count if necessary

  # calculate cluster mean x/y and update centroid for each cluster

  # save current cluster sizes (for comparison in next iteration)

  # empty clusters for re-clustering except on final iteration

# output results in specified format (shown in Output Format below)
```

Output Format

For each input file, points1.txt and points2.txt, please output the following:

1. A list of the 4 initial COVID-19 patients (centroids)
2. The number of iterations required to obtain stability
3. The values for the 4 final centroids
4. For each centroid print:
    a. The centroid ID followed by the final number of patients in the centroid's cluster
    b. A list of the "patients" (points) contained in the cluster

Our solutions are shown below so you can check your solutions against ours.

Solutions

Here is our final solution for **points1.txt**:

```
Initial COVID-19 Patients: [[30, 45], [55, 82], [1, 61], [96, 14]]

Iterations to achieve stability: 7

Final Centroids:
[28.423076923076923, 20.076923076923077]
[62.458333333333336, 84.45833333333333]
[23.192307692307693, 68.38461538461539]
[85.4, 27.75]

Number of patients in Cluster 0: 26
[[0, 27], [33, 30], [34, 7], [21, 23], [40, 41], [46, 46], [34, 38], [56, 17], [38, 45], [36, 9],
[52, 12], [19, 12], [39, 2], [16, 33], [12, 5], [15, 10], [6, 18], [18, 2], [31, 8], [48, 24], [20
, 10], [56, 15], [2, 34], [1, 27], [42, 27], [24, 0]]

Number of patients in Cluster 1: 24
[[37, 95], [40, 96], [93, 73], [64, 91], [97, 99], [51, 80], [42, 86], [53, 82], [41, 95], [47, 97
], [65, 85], [78, 81], [54, 78], [58, 83], [81, 83], [79, 76], [78, 67], [44, 92], [52, 100], [64,
 99], [88, 71], [70, 75], [52, 67], [71, 76]]

Number of patients in Cluster 2: 26
[[5, 58], [40, 64], [37, 61], [22, 58], [2, 56], [5, 53], [40, 74], [11, 95], [22, 81], [44, 62],
[10, 87], [37, 57], [7, 44], [16, 47], [13, 85], [43, 67], [26, 97], [17, 71], [18, 75], [19, 61],
 [30, 97], [41, 49], [23, 71], [31, 67], [1, 91], [43, 50]]

Number of patients in Cluster 3: 20
[[83, 13], [80, 18], [77, 16], [81, 40], [78, 16], [89, 26], [91, 45], [88, 32], [92, 34], [99, 15
], [86, 57], [86, 36], [58, 32], [71, 21], [89, 16], [97, 35], [91, 7], [98, 24], [93, 40], [81, 3
2]]
```

Here is our final solution for **points2.txt**:

```
Initial COVID-19 Patients: [[42, 32], [10, 9], [10, 60]]

Iterations to achieve stability: 3

Final Centroids:
[71.5, 46.22727272727273]
[13.2, 18.0]
[31.35, 74.55]

Number of patients in Cluster 0: 22
[[71, 84], [72, 48], [75, 36], [52, 28], [76, 44], [56, 35], [88, 32], [61, 34], [94, 12], [71, 59],
[76, 75], [62, 1], [71, 4], [78, 91], [100, 98], [49, 34], [49, 40], [57, 48], [66, 67], [97, 3], [66
, 76], [86, 68]]

Number of patients in Cluster 1: 5
[[7, 16], [15, 21], [25, 16], [16, 32], [3, 5]]

Number of patients in Cluster 2: 20
[[36, 52], [11, 69], [32, 74], [10, 74], [51, 85], [10, 75], [55, 96], [34, 64], [54, 100], [60, 85],
 [14, 77], [40, 78], [2, 60], [37, 49], [42, 77], [39, 60], [38, 77], [40, 53], [14, 99], [8, 87]]
```

Additional Skills (Optional): Data Visualization with matplotlib

When using machine learning algorithms like k-means clustering it can be very helpful to see a visual representation of the data. matplotlib is one of the most popular Python libraries for data visualization. As a small challenge, we thought it would be fun for you to visualize your clusters and get some experience with a new Python tool!

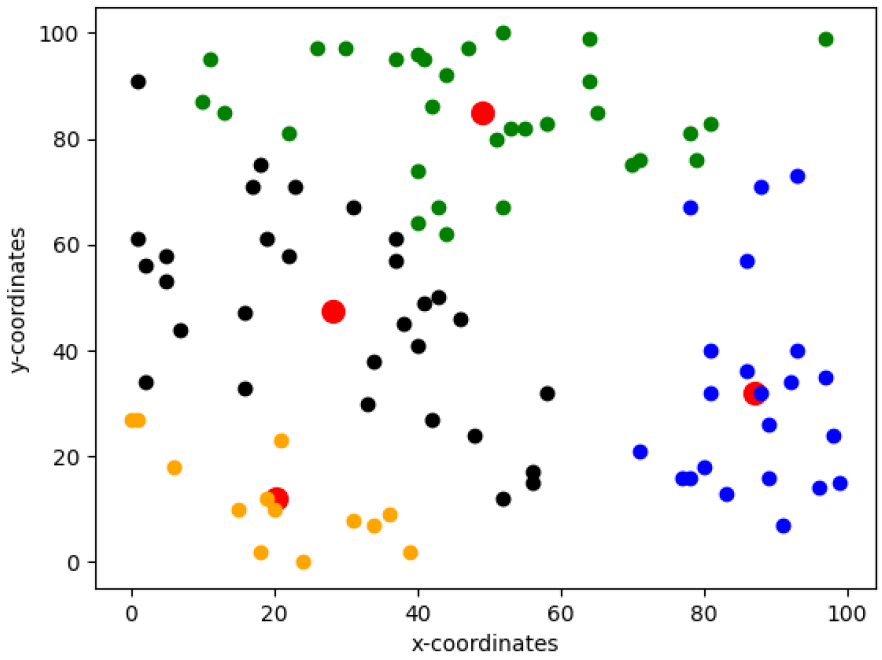
Possible Progress Steps:

1. Your solution to the k-means assignment probably left you with two things: a list of centroids and a list of clusters containing all associated points. Start by using matplotlib to print a scatterplot of these points (which represent the final clustering) at the very end of your code. To help get you started we've provided the following matplotlib started code.
2. The starter code provides a hint at how you can change the color of your datapoints. Change the colors of your points, and centroids, by creating a list of colors. As you iterate through your clusters to print the scatterplot of points, change the color for each cluster and use a separate color for your centroids to make them pop. Feel free to be creative with your color scheme!
3. Once you have your clusters colored, use the other small hint from the starter code to enlarge your centroids so they are more clearly identifiable.
4. Finally, once you have steps 1-3 working to visualize the final state of the algorithm, consider re-using that code to print the state of the centroids/clusters after the very first iteration for comparison.

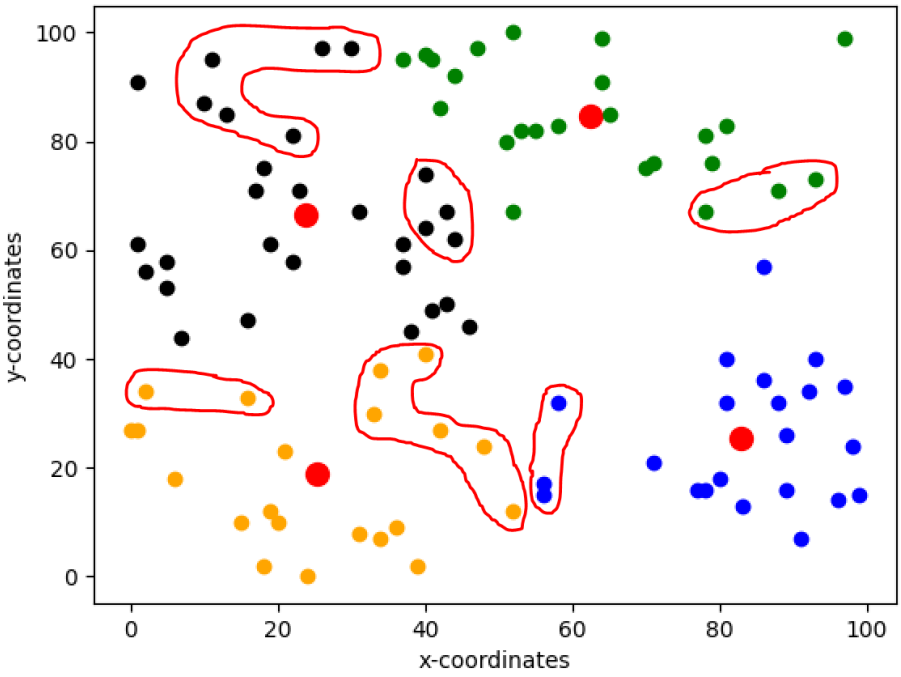
Our data visualizations can be viewed on the next page if you'd like to see an example of what your output might look like.

If you decide to tackle this portion of the Assignment, please keep a few things in-mind:

1. You do not have to submit your code for this optional portion of the Assignment, though you're certainly welcome to.
2. This "challenge" is solely meant to help you further hone your skills and gain some exposure to other tools and parts of the language. As a result, it will not be counted as extra credit, etc.
3. We'll do our best to support any questions on the "Additional Skills" section of this Assignment, but we want to be sure to give priority to those working on the core portion of the Assignment too. As a result, please allow us just a bit of extra time to respond.
4. Our solution to the Assignment, once released, will also contain a solution to Additional Skills challenge as well. Please feel free to review it and compare it against your solution.



Initial clustering after 1 iteration



Final clustering after k-means reaches convergence

(We tried to [manually] circle the points that changed clusters for comparison)

Deliverables

readme.txt

So-called "read me" files are a common way for developers to leave high-level notes about their applications. Here's an example of a README file for the Apache Spark project. They usually contain details about required software versions, installation instructions, contact information, etc. For our purposes, your readme.txt file will be a way for you to describe the approach you took to complete the assignment so that, in the event you may not quite get your solution working correctly, we can still award credit based on what you were trying to do. Think of it as the verbalization of what your code does (or is supposed to do). Your readme.txt file should contain the following:

1. **Name**: Your name and JHED ID
2. **Module Info**: The Module name/number along with the title of the assignment and its due date
3. **Approach**: a detailed description of the approach you implemented to solving the assignment. Be as specific as possible. If you are sorting a list of 2D points in a plane, describe the class you used to represent a point, the data structures you used to store them, and the algorithm you used to sort them, for example. The more descriptive you are, the more credit we can award in the event your solution doesn't fully work.
4. **Known Bugs**: describe the areas, if any, where your code has any known bugs. If you're asked to write a function to do a computation but you know your function returns an incorrect result, this should be noted here. Please also state how you would go about fixing the bug. If your code produces results correctly you do not have to include this section.
5. **Answers to the following 2 questions**:
    a. This particular implementation of k-means is inefficient in terms of the number of the overall number of iterations. Why is it inefficient and how would you fix this in your code?
    b. This implementation may also, depending on the input, yield incorrect answers. Specifically it may converge too early. Why is this the case and how would you fix it?

Please submit your ***kmeans.py*** source code files along with a PDF file containing screenshots of your outputs in a PDF called JHEDID_mod5.pdf (ex: jkovba1_mod5.pdf). Please do not ZIP your files together.

Recap:

1. readme.txt containing the answers to the questions:
2. kmeans.py
3. 1 screenshot of your output from running kmeans.py on points1.txt
4. 1 screenshot of your output from running kmeans.py on points2.txt

Please let us know if you have any questions via Teams or email!