Practicum 10 - k-means clustering, part 2

DS2001 - Computer Science Practicum
Fall 2021
November 17th - 18th, 2021
Practicum Deadline: November 19th, 2021 at 12pm (noon!) Boston time

Make sure to check the presentation schedule for when you'll be presenting!
(if this time doesn't work for you, you MUST contact Prof. Felix this week (by the end of 11/19/2021))

What we're practicing today:
- functions
- dicts

Handouts:
- dicts (from DS 2000)

What to do if you miss practicum this week:
- Fill out the "I'm missing class" form ASAP
- Follow up in office hours with Felix (find office hours links on the course website). **Note that this is required.**
- Come to practicum next week if you are well again. **Do not come to class if you are not feeling well.**

How far we expect you to get this week:
- We expect you to complete **Task 2** and would recommend an attempt **Task 3.**
    - For task 2, we are looking for you to graph the centroids and the data, but the legend and the colors/shapes need not be correct.

Today will be part 2 of a two-part series in which you will be implementing an algorithm from machine learning called k-means clustering. This algorithm is one that figures out groups ("clusters") of points that belong together. At its basis, it is composed of a few steps:
1) graph your data
2) pick some means ("centroids") by randomly plotting k new data points (e.g. k=2 for us)
3) assign each data point to the closest centroid
4) adjust the centroids to be closer to the mean of the group of points that they "belong to"
5) Repeat steps 2 & 3 until the centroids no longer move

Task 0: Open up your kmeans.py or download p9_kmeans.py from the course website

If you work from our copy, we recommend:
1) renaming your kmeans.py from practicum 9 to something else, e.g. "kmeans_mine_p9.py"
2) renaming p9_kmeans.py to kmeans.py

If you finished Task 4 (writing your function to determine the closest point to a target point from a list of points), you'll have enough implemented to work from for today! Regardless, we recommend looking at our solution and copy + pasting as needed!

Task 1: Assign data points to clusters

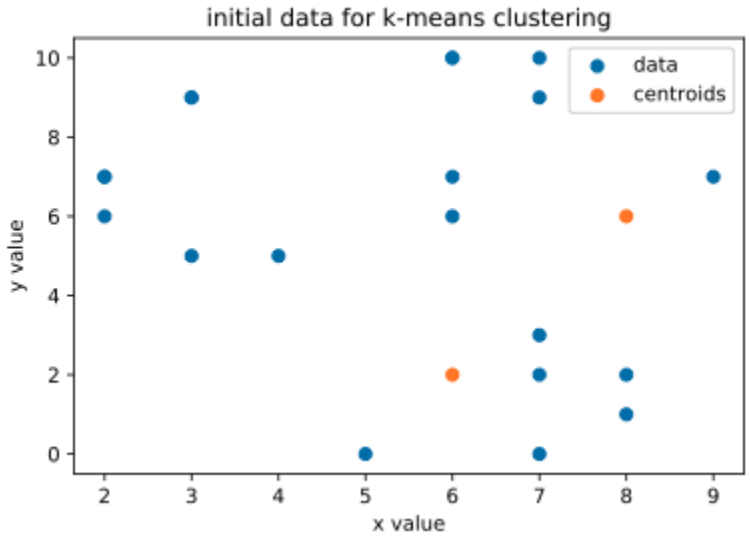
We're ready to start on the main part of the k-means algorithm now! (we're on step 3 of the algorithm summary at the top of this document)

For this step, you'll want to create a dictionary that maps **tuples** (each one of your centroids will be a key) to **lists of points** (these will be the data points that "belong" to each centroid).

Iterate through your data points and assign each one to a list of points matching the centroid that it is closest to.

Example output from last practicum (we added a line of asterisks to separate the output of the previous practicum from this one):

```
data: [(7, 9), (6, 10), (6, 10), (8, 1), (3, 9), (6, 6), (3, 9), (3, 5), (7, 3), (4, 5),
(7, 2), (2, 6), (7, 10), (6, 7), (2, 7), (2, 7), (9, 7), (5, 0), (8, 2), (7, 0)]
centroids: [(6, 2), (8, 6)]
```




```
Testing distance function
distance from (0, 0) to (1, 1): 1.4142135623730951
distance from (-10, 0) to (10, 0): 20.0

Testing closest function
closest centroid to (0, 0): (6, 2)
closest centroid to (10, 10): (8, 6)

counts of points assigned to clusters
{(8, 6): 11, (6, 2): 9}
***************
```

Example new output:

```
points assigned to clusters:
```

```
{(8, 6): [(7, 9), (6, 10), (6, 10), (3, 9), (6, 6), (3, 9), (7, 10), (6, 7), (2, 7), (2, 7), (9, 7)], (6, 2): [(8, 1), (3, 5), (7, 3), (4, 5), (7, 2), (2, 6), (5, 0), (8, 2), (7, 0)]}
```

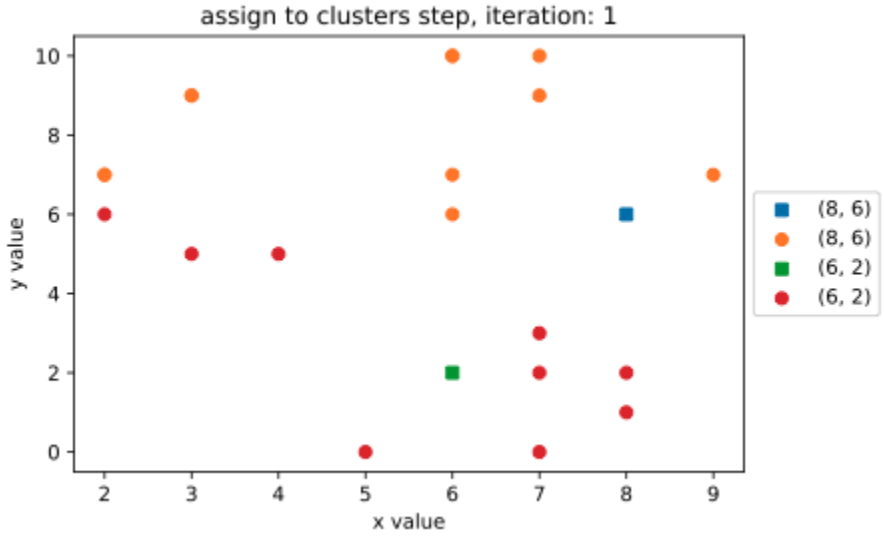
Hint: you may find copy+pasting the code from Task 5 from the last practicum to be a useful place to start from for this task!

Group question: how did you "set up" your dictionary? What did you need to do the first time that a centroid is assigned a point? Was this different from the second time that a centroid is assigned a point?

Task 2: Graph the initial assignments

Next, create a graph that displays the initial assignment of points to centroids. We recommend writing a function that does this task. This function should not have a return value, since it's only job should be displaying the current state of the algorithm.

Example output (we used the default color progression and squares to differentiate the centroids from the data. Feel free to use shape and/or color to differentiate these! Your graphs do ***not*** need to exactly match ours. We labeled each data cluster with the coordinates of the centroid that it belongs to.)



Optionally:
If you'd like to make your legend display to the right of the graph, you can update your call to `plt.legend()` like:

!*However*! this line **will cause** the autograder to break. Either don't worry about it (we won't penalize you) or change this to just `plt.legend()` when you turn in your code.

```
plt.legend(loc='center left', bbox_to_anchor=(1, 0.5))
```

To customize the shape of the points that are graphed:

```
plt.scatter(xs, ys, label=str(centroid), marker="s")  # for a square, use "s"
```

If you want to look at other available shape codes, take a look at the matplotlib markers documentation. Feel free to use whatever shape(s) and/or colors that you'd like!

Hint: we looped over the keys of our clusters dictionary and for every iteration made two calls to plt.scatter—one for the centroid (the key) and one for the points assigned to it. Don't forget to use the get_column function to help you out!

Hint 2: Note that your centroids may start out on top of some of your data points. This is fine, don't panic. :)

Task 3: Choose better centroids

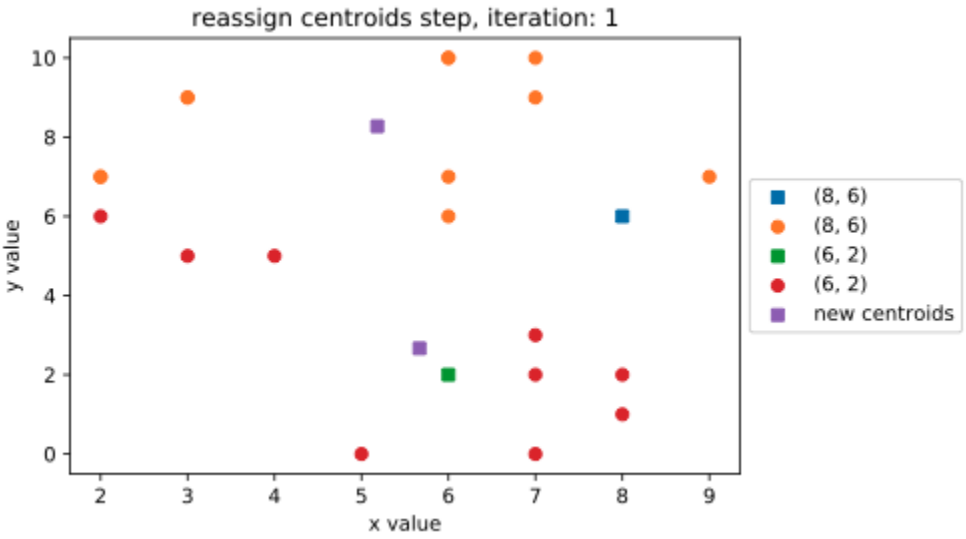
Next, we'll implement step 4 from the algorithm at the top of the document: choose better centroids. For each cluster, you'll want to calculate the average x-value of the points assigned to it and the average y-value of the points assigned to it.

Then, you'll save these new points in a list of new centroids that we'll soon replace our old centroids with. (But not quite yet!)

Display the new centroids alongside your others on a graph. Again, we recommend using a function to do this for nice modularization of your code!

Example output (we used purple to differentiate the new centroids from the old ones. Feel free to use shape and/or color to differentiate these! Your graphs do ***not*** need to exactly match ours.):

```
updated centroids: [(5.181818181818182, 8.272727272727273), (5.666666666666667,
2.6666666666666665)]
```

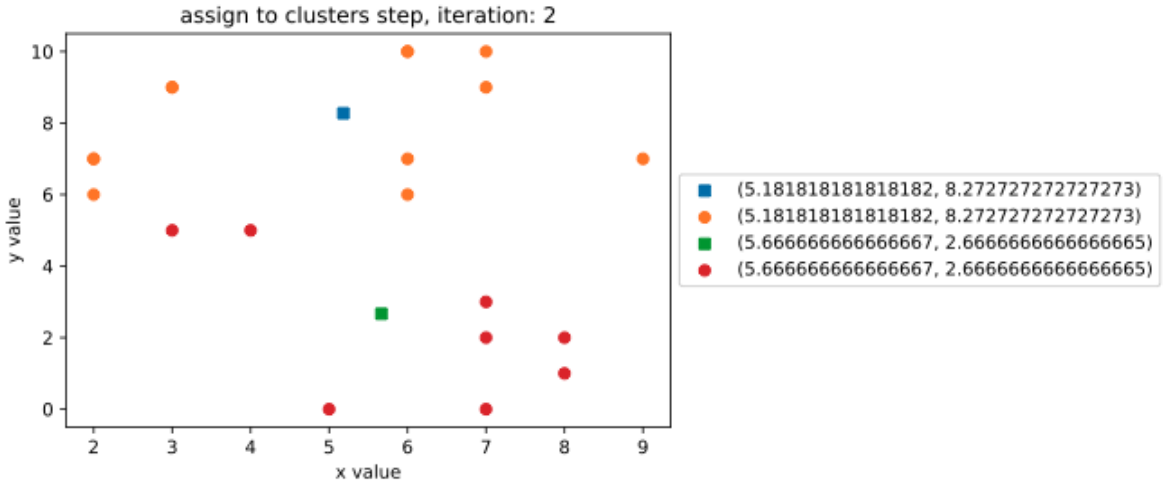


Task 4: Wrap everything up in a loop that iterates for a specific number of iterations

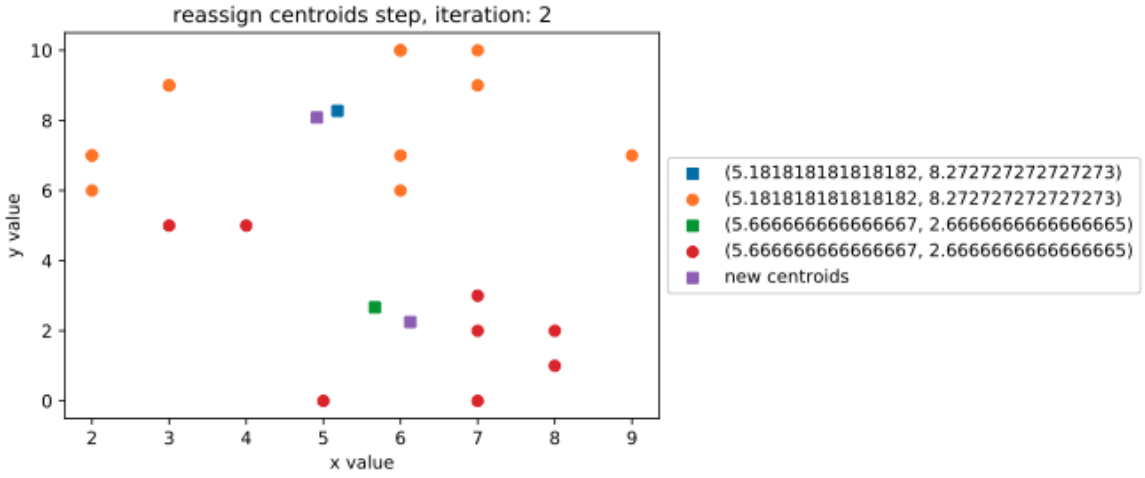
Finally, we're going to make it so that the algorithm repeats steps 3 & 4 for a certain number of iterations. Wrap your code from Tasks 1 - 3 in a for loop or a while loop so that it iterates for a certain number of ITERATIONS (define this constant at the top of your file. Start with a small-ish number, like 2, 3, or 4).

Here's the output of our program for the second iteration for the example output:

```
**********
Iteration: 2
points assigned to clusters:
{(5.181818181818182, 8.272727272727273): [(7, 9), (6, 10), (6, 10), (3, 9), (6, 6), (3, 9), (2, 6), (7, 10), (6, 7), (2, 7), (2, 7), (9, 7)], (5.666666666666667, 2.6666666666666665): [(8, 1), (3, 5), (7, 3), (4, 5), (7, 2), (5, 0), (8, 2), (7, 0)]}
```




```
updated centroids: [(4.916666666666667, 8.083333333333334), (6.125, 2.25)]
```



Run your program multiple times. Are the centroids still "moving" when the program stops?

Group question: what should the "reassign centroids step" graph look like when the centroids stop moving?

Task 5: Update your loop so that it iterates *until the centroids no longer move*

Now, rather than iterating for ITERATIONS, update your loop so that it runs until the centroids no longer move.

Hints:

1) You'll need a loop that can run for an unknown number of repeats (since we can't pre-calculate how many iterations we'll need).
2) You'll want to measure the amount of change between your old centroids and your new ones (calculated in Task 3) to determine whether or not you're done.
3) You don't need to use `break` for this task. :)

If you finish...

- make it so that your graphing function from Task 2 can *also* create the graph for Task 3.
    - Hint: default parameters are your friends here!
- work on your final project!