HW4: Clustering

Start Assignment

Due Tuesday by 10:59am **Points** 100 **Submitting** a file upload **File Types** zip
Available Oct 4 at 11:59pm - Oct 12 at 11:15am 7 days

Assignment Goals

- Process real-world data
- Implement hierarchical clustering
- Randomly generate your own data
- Imshow your linkage process

Summary

In this assignment, you'll be performing clustering using the publicly available Pokemon stats. Each Pokemon is defined by a row in the data set. Because there are various ways to characterize how strong a Pokemon is, it is often desirable to convert a raw stats sheet into a shorter feature vector. For this assignment, you will represent a Pokemon's strength by two numbers: "x" and "y". "x" will represent the Pokemon's total offensive strength, which is defined by Attack + Sp. Atk + Speed. Similarly, "y" will represent the Pokemon's total defensive strength, which is defined by Defense + Sp. Def + HP. After each Pokemon becomes that two-dimensional feature vector, you will cluster the first 20 Pokemon with hierarchical agglomerative clustering (HAC).

Packages Needed for this Project

In this project, you'll need the following packages:

```
import csv
import numpy as np
import matplotlib.pyplot as plt
```

Program Specification

Download the data in CSV format: **Pokemon.csv**
(https://canvas.wisc.edu/courses/258491/files/21751866/download?download_frd=1)

Write the following Python functions:

1. **load_data(filepath)** — takes in a string with a path to a CSV file formatted as in the link above, and returns the first 20 data points (**without** the Generation and Legendary columns but retaining all other columns) in a single structure.
2. **calculate_x_y(stats) —** takes in one row from the data loaded from the previous function, calculates the corresponding x, y values for that Pokemon as specified above, and returns them in a single structure.
3. **hac(dataset)** — performs single linkage hierarchical agglomerative clustering on the Pokemon with the (x,y) feature representation, and returns a data structure representing the clustering.
4. **random_x_y(m)** — takes in the number of samples we want to randomly generate, and returns these samples in a single structure.
5. **imshow_hac(dataset)** — performs single linkage hierarchical agglomerative clustering on the Pokemon with the (x,y) feature representation, and imshow the clustering process.

You may implement other helper functions as necessary, but these are the functions we will be testing.

Load Data

Read in the file specified in the argument (the DictReader from [Python's csv module (https://docs.python.org/3/library/csv.html#csv.DictReader)](https://docs.python.org/3/library/csv.html#csv.DictReader) will be of use) and return a list of dictionaries, where each row in the dataset is a dictionary with the column headers as keys and the row elements as values. These dictionaries should **not** include the Generation and Legendary columns, as we will not be using them in this program. The integer strength of Pokemons (6 columns) and the 2 other columns (#, Total) should be converted to integers. Also, you only should have the **first 20** Pokemon in this structure. Note the first 20 refers to row 1 through row 20 (From Bulbasaur to BeedrillMega Beedrill excluding the headers).

You may assume the file exists and is a properly formatted CSV.

```
>>> pokemons = load_data("Pokemon.csv")
>>> len(pokemons)
20
>>> pokemons
[{'#': 1, 'Name': 'Bulbasaur', 'Type 1': 'Grass', 'Type 2': 'Poison', 'Total': 318, 'HP': 45,
'Attack': 49, 'Defense': 49, 'Sp. Atk': 65, 'Sp. Def': 65, 'Speed': 45}, ..., {'#': 15, 'Name':
'BeedrillMega Beedrill', 'Type 1': 'Bug', 'Type 2': 'Poison', 'Total': 495, 'HP': 65, 'Attack': 150,
'Defense': 40, 'Sp. Atk': 15, 'Sp. Def': 80, 'Speed': 145}]
```

Calculate Feature Values

This function takes in the data from a single row of the raw dataset as read in the previous function (i.e. a single dictionary, **without** the Generation and Legendary values but retaining all other columns). This function should return the x, y values in a tuple, formatted as (x, y).

```
>>> pokemons[1]
{'#': 2, 'Name': 'Ivysaur', 'Type 1': 'Grass', 'Type 2': 'Poison', 'Total': 405, 'HP': 60, 'Attack':
62, 'Defense': 63, 'Sp. Atk': 80, 'Sp. Def': 80, 'Speed': 60}
>>> calculate_x_y(pokemons[1])
(202, 203)
```

Perform HAC

For this function, we would like you to mimic the behavior of **SciPy's HAC function (https://docs.scipy.org/doc/scipy/reference/generated/scipy.cluster.hierarchy.linkage.html)** , linkage(). You may not use this function in your implementation, but we strongly recommend using it to verify your results!

Input: A collection of *m* observation vectors in *n* dimensions may be passed as an *m* by *n* array (for us, this will be a list of tuples, not a numpy array like for linkage()!). All elements of the condensed distance matrix must be finite, i.e. no NaNs or infs. In our case, *m* is the number of Pokemon (20) and *n* is 2 (the x and y features for each Pokemon). You need to check for invalid inputs. If there is any, pop them out.

Using **single linkage**, perform the hierarchical agglomerative clustering algorithm as detailed on **slides (http://pages.cs.wisc.edu/~yliang/cs540_1_fall21/documents/CS540-UL1.pdf)** . Use a standard Euclidean distance function for calculating the distance between two points.

Output: An (*m*-1) by 4 matrix Z. At the *i*-th iteration (i starts from 0), clusters with indices Z[i, 0] and Z[i, 1] are combined to form cluster ***m + i***. A cluster with an index less than *m* corresponds to one of the *m* original observations. The distance between clusters Z[i, 0] and Z[i, 1] is given by Z[i, 2]. The fourth value Z[i, 3] represents the number of original observations in the newly formed cluster.

That is:

- Number each of your starting data points from 0 to *m*-1. These are their original cluster numbers.
- Create an (*m*-1) x 4 array or list. Iterate through the list row by row.
- For each row of the output matrix Z, determine which two clusters you will merge and put their numbers into the first and second elements of the row. The first point listed should be the smaller of the two cluster indexes. The single-linkage distance between the two clusters goes into the third element of the row. The total number of points in the cluster goes into the fourth element.
- Before returning the data structure, convert it into a NumPy matrix.

If you follow these guidelines for input and output, your result should match the result of scipy.cluster.hierarchy.linkage() and you can use that function to verify your results.

Tie-Breaking

In the event that there are multiple pairs of points with equal distance for the next cluster:

Given a set of pairs with equal distance {(xi, xj)} where i < j, we prefer the pair with the smallest first cluster index *i*. If there are still ties (xi, xj), ... (xi, xk) where *i* is that smallest first index, we prefer the pair with the smallest second cluster index.

Be aware that this tie-breaking strategy may not always produce identical results to scipy.cluster.hierarchy.linkage().

```
>>> pokemons_x_y = []
>>> for row in pokemons:
...     pokemons_x_y.append(calculate_x_y(row))
...
```

```
>>> hac(pokemons_x_y)
```

```
matrix([[ 7.        ,  8.        ,  4.24264069,  2.        ],
        [15.        , 18.        ,  7.07106781,  2.        ],
        [ 1.        , 21.        , 13.34166406,  3.        ],
        [13.        , 16.        , 14.14213562,  2.        ],
        [14.        , 17.        , 14.14213562,  2.        ],
        [ 5.        , 22.        , 14.56021978,  4.        ],
        [ 3.        , 12.        , 16.64331698,  2.        ],
        [23.        , 24.        , 22.36067977,  4.        ],
        [10.        , 25.        , 22.627417  ,  5.        ],
        [ 0.        ,  9.        , 22.8035085 ,  2.        ],
        [ 2.        , 11.        , 26.40075756,  2.        ],
        [ 4.        , 29.        , 32.44996148,  3.        ],
        [ 6.        , 30.        , 38.01315562,  3.        ],
        [28.        , 31.        , 55.47071299,  8.        ],
        [19.        , 32.        , 58.52349955,  4.        ],
        [20.        , 26.        , 59.46427499,  4.        ],
        [34.        , 35.        , 68.24221567,  8.        ],
        [27.        , 33.        , 77.1038261 , 12.        ],
        [36.        , 37.        , 84.85281374, 20.        ]])
```

For example, in row 3, column 2 of the output, 21 is the cluster index that represents the cluster formed at the 2nd iteration (i = 1).

Generate Random Data

Given a positive integer m, your aim is to uniformly randomly generate m Pokemons' "x" and "y", which satisfy 0<x<360, 0<y<360, and where x,y are both integers. The output of this **random_x_y(m)** function will further be used as the input to **hac(dataset).**

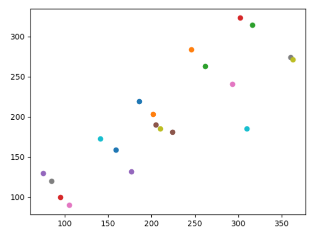
```
>>> random_x_y(20)
[[350, 144], [187, 76], [3, 149], [319, 281], [305, 249], [179, 213], [199, 47], [45, 80], [273, 242],
[16, 89], [172, 337], [292, 353], [328, 2], [297, 171], [163, 197], [230, 211], [70, 343], [311, 250],
[349, 282], [133, 44]]
```

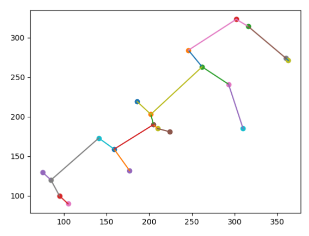
Note: your output does not need to match the given output, since the data are randomly generated, but be aware of the bound and the distribution from which these data are generated.

Imshow HAC Process

You need to imshow the HAC process by including the start status (m points) and the results of (m-1) linkage processes. You should **only have one** plt.figure(), so in total you will need to plot m times on the **same** figure. Pause 0.1 seconds after each plotting, and do not close or clean the figure at the end of your function. An example of the start and the end status are shown as below. (**Hint**: consider using plt.scatter(), plt.plot(), plt.pause(); you can reuse/revise the code of **hac(dataset)** by embedding the calls to appropriate plotting functions into the HAC process code.)

A useful intro (https://www.geeksforgeeks.org/matplotlib-pyplot-pause-in-python/) to plt.pause().





Submission Notes

Please submit your files in a zip file named **hw4_<netid>.zip**, where you replace <netid> with your netID (your wisc.edu login). Inside your zip file, there should be **only** one file named: **pokemon_stats.py**. Do not submit a Jupyter notebook .ipynb file.

All code should be contained in functions or under a

```
if __name__=="__main__":
```

check so that it will not run if your code is imported to another program.

Be sure to **remove all debugging output** before submission. Failure to remove debugging output will be **penalized (10pts)**.

Cheating results in -100 pts and further punishment.

Note: From homework 4 and on, we will deduct 5 points if the submitted zip file contains subfolder when the grader unzips the zip file

This assignment is due on 10/12/2021 at 10:59 am. It is preferable to first submit a version well before the deadline (at least one hour before) and check the content/format of the submission to make sure it's the right version. Then, later update the submission until the deadline if needed.

Some Rubric

Criteria	Ratings		Pts
load_data(filepath)	**20 pts** **Full Marks**	**0 pts** **No Marks**	20 pts
calculate_x_y(stats)	**20 pts** **Full Marks**	**0 pts** **No Marks**	20 pts
hac(dataset)	**20 pts** **Full Marks**	**0 pts** **No Marks**	20 pts
random_x_y(m)	**20 pts** **Full Marks**	**0 pts** **No Marks**	20 pts
imshow_hac(dataset)	**20 pts** **Full Marks**	**0 pts** **No Marks**	20 pts
			Total Points: 100