Homework 4
CS540 F24

Assignment Goals

- Process real-world data
- Implement hierarchical clustering
- Visualize the clustering process

You are to perform hierarchical clustering on socioeconomic data from various countries. Each country is defined by a row in the data set. We will use a subset of the data to represent each country with a feature vector. For this assignment, you must represent a country with 6 statistics: 'Population', 'Net migration', 'GDP ($ per capita)', 'Literacy (%)', 'Phones (per 1000)', and 'Infant mortality (per 1000 births)'.

After each country is represented as a 6-dimensional feature vector ($x_1, \ldots, x_6$), you need to cluster the countries with hierarchical agglomerative clustering (HAC). This clustering will allow us to visualize which countries have similar socioeconomic situations. **Using scipy.cluster.hierarchy.linkage() is strictly prohibited and will result in a zero score for this assignment.** Apart from this, you may use anything in python's built-in standard library, scipy(besides hierarchy.linkage), numpy and matplotlib.

Program Overview

The data in CSV format can be found in the file countries.csv. Note, there is no starter code for this assignment. If you feel stuck, refer to the starter code from past assignments. You will have to write a few python functions for this assignment. (**Note: the functions have to be named exactly as in the writeup**) Here is a high level description of each for reference:

1. load_data(filepath) — takes in a string with a path to a CSV file and returns the data points as a list of dicts. Section 0.1
2. calc_features(row) — takes in one row dict from the data loaded from the previous function then calculates the corresponding feature vector for that country as specified above, and returns it as a NumPy array of shape (6,). The dtype of this array should be float64. Section 0.2
3. hac(features) — performs single linkage hierarchical agglomerative clustering on the country with the ($x_1, \ldots, x_6$) feature representation, and returns a NumPy array representing the clustering. Section 0.3
4. fig_hac(Z, names) — visualizes the hierarchical agglomerative clustering on the country's feature representation. Section 0.4

5. normalize_features(features) — takes a list of feature vectors and computes the normalized values. The output should be a list of normalized feature vectors in the same format as the input. Section 0.5

You may implement other helper functions as necessary, but these are the functions we are testing. In particular, your final python file is just a suite of functions, you should not have code that runs outside of the functions. To test your code, you may want a "main" method to put it all together. Make sure, you either delete any testing code running outside functions or wrap it in a if __name__=="__main__":. This is discussed more in Section 0.6.

Program Details

0.1 load_data(filepath)

Summary. [10pts]

- Input: string, the path to a file to be read.
- Output: list, where each element is a dict representing one row of the file read.

Details.

1. Read in the file specified in the argument, `filepath`. The `DictReader` from Python's `csv` module is useful for reading the file. Note that depending on your Python version, this might return `OrderedDicts` instead of normal `dicts`, so make sure to convert them to regular `dicts` if necessary. No type conversion is performed on the data; all values are kept as strings.
2. 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. We preserve all columns from the CSV file exactly as they are.
3. You may assume the file exists and is a properly formatted CSV.

0.2 calc_features(row)

Summary. [10pts]

- Input: dict representing one country.
- Output: NumPy array of shape (6,) and dtype float64. The first element is x_1 and so on with the sixth element being x_6.

Details. This function takes as input the dict representing one country, and computes the feature representation ($x_1, \ldots, x_6$). Specifically,

1. x_1 = 'Population'
2. x_2 = 'Net migration'

3. x_3 = ‘GDP ($ per capita)’
4. x_4 = ‘Literacy (%)’
5. x_5 = ‘Phones (per 1000)’
6. x_6 = ‘Infant mortality (per 1000 births)’

Note, these stats in the dict may not be float types. Make sure to convert each relevant stat to float when computing each x_i. Return a NumPy array having each x_i in order: $x_1, \ldots, x_6$. The shape of this array should be (6,). The dtype of this array should be float64. Remember, this function works for only one country, not all of the ones that you loaded in load_data. Make sure you are outputting the exact data structures with appropriate types as specified or you risk a major reduction in points.

0.3 hac(features)

Summary. [50pts]

- Input: list of NumPy arrays of shape (6,), where each array is an $(x_1, \ldots, x_6)$ feature representation as computed in Section 0.2. The total number of feature vectors, i.e. the length of the input list, is n. Note, we test your code on different n’s as stated in Section 0.6).

- Output: NumPy array of shape $(n-1) \times 4$. For any i, $Z[i, 0]$ and $Z[i, 1]$ represent the indices of the two clusters that were merged in the ith iteration of the clustering algorithm. Then, $Z[i, 2] = d(Z[i, 0], Z[i, 1])$ is the single linkage distance between the two clusters that were merged in the ith iteration (this will be a real value, not integer like the other quantities). Lastly, $Z[i, 3]$ is the size of the new cluster formed by the merge, i.e. the total number of countries in this cluster. Note, the original countries are considered clusters indexed by $0, \ldots, n-1$, and the cluster constructed in the ith iteration ($i \geq 1$) of the algorithm has cluster index $(n-1)+i$. Also, there is a tie-breaking rule specified below that must be followed.

Distance. Using single linkage, perform the hierarchical agglomerative clustering algorithm as detailed in lecture. Use the standard Euclidean distance function for calculating the distance between two points. You may implement your own distance function or use numpy.linalg.norm(). Other distance functions might not work as expected so check your results on Gradescope! You are liable for any reductions in points you might get for using a package distance function.

Outline. Here is one possible path you could follow to implement hac()

1. Number each of your starting data points from 0 to $n-1$. These are their original cluster numbers.

2. Create an $(n-1) \times 4$ array or list. Iterate through this array/list row by row. For each row,

   (a) Determine which two clusters are closest and put their numbers into the first

and second elements of the row, $Z[i, 0]$ and $Z[i, 1]$. The first element listed, $Z[i, 0]$ should be the smaller of the two cluster indexes.

(b) The single-linkage distance between the two clusters goes into the third element of the row, $Z[i, 2]$

(c) The total number of countries in the cluster goes into the fourth element, $Z[i, 3]$

If you merge a cluster containing more than one country, its index (for the first or second element of the row) is given by n + the row index in which the cluster was created.

3. Before returning the data structure, convert it into a NumPy array if it isn't one already.

For this method to run efficiently when n is large you should maintain a distance matrix at the beginning of the function to avoid having to recalculate the distances between points. After an iteration where clusters i and j are merged, you can

1. add a column and row at the end to store the distances to the new cluster {i, j}: the distance from an existing cluster k to the new cluster {i, j} using single-linkage distance is the minimum between the distance from cluster k to i and the distance from cluster k to j;

2. remove columns and rows i and j (or simply change the values in these columns and rows to -1);

3. find the two clusters with the minimum distance between them (they are the column and row index of the minimum non-negative non-diagonal entry in the updated distance matrix), and merge these two clusters in the next iteration.

You should be able to run HAC efficiently with all countries in the CSV file. For example, if you have a NumPy array called distance_matrix then distance_matrix[3,4] is equal to the Euclidean distance between the countries at index 3 and 4 in the features input. You can compare your output with scipy.spatial.distance_matrix, but please note we are using Euclidean distance for this assignment.

Tie Breaking. When choosing the next two clusters to merge, we pick the pair having the smallest single-linkage distance. In the case that multiple pairs have the same distance, we need additional criteria to pick between them. We do this with a tie-breaking rule on indices as follows: Suppose $(i_1, j_1), \ldots, (i_h, j_h)$ are pairs of cluster indices with equal distance, i.e., $d(i_1, j_1) = \cdots = d(i_h, j_h)$, and assume that $i_t < j_t$ for all t (so each pair is sorted). We tie-break by picking the pair with the smallest first index, i. If there are multiple pairs having first index i, we need to further distinguish between them. Say these pairs are $(i, t_1), (i, t_2), \ldots$ and so on. To tie-break between these pairs, we pick the pair with the smallest second index, i.e., the smallest t value in these pairs. Be aware that this tie-breaking strategy may not produce identical results to linkage().

0.4 fig_hac(Z, names)

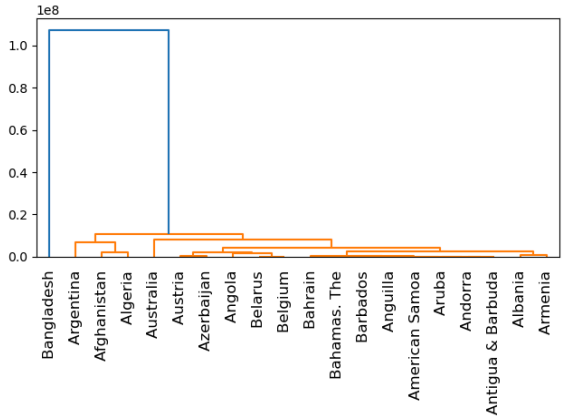
Summary. [10pts]

- Input: NumPy array *Z* output from hac, and list of string names corresponding to country names with length *n*.
- Output: A matplotlib figure with a graph that visualizes hierarchical clustering.

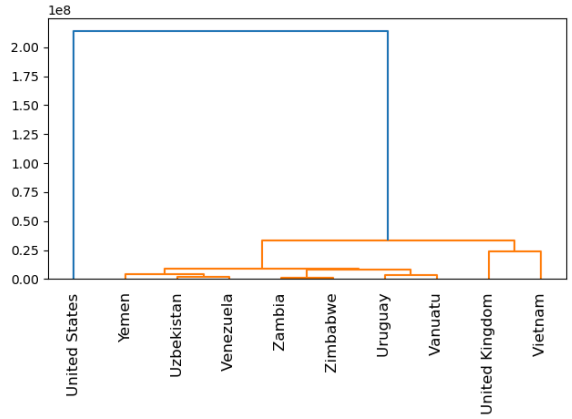
Details. Begin by initializing a figure with ‘fig = plt.figure()`. Then use dendrogram in the SciPy module in [dendrogram — SciPy v1.14.1 Manual] with `labels=names` and `leaf_rotation = ???`. Your plot will likely cut off the x labels. For the graph to look like the below examples you will need to call tight_layout() on the figure.

The visualizations for the first 20 and final 10 countries

N = 20



N = 10



Discussion. Notice that the figures above cluster the countries almost exclusively by population. The values for population are much larger than the values for other columns in the data. Therefore, the population disproportionately contributes to the Euclidean distance between feature vectors. Ideally, all six statistics should contribute to the clustering, so you will create a function to normalize the data in the next section.

0.5 normalize_features(features)

Summary. [20pts]

- Input: The input to this function will be a list of the feature vectors output from calc_features. Each feature vector is a NumPy array with shapes (6,) and dtype float64.
- Output: The output should have an identical format to the input: a list of NumPy arrays with shape (6,) and dtype float64. However, the statistic values in the feature vectors should be replaced with their normalized values.

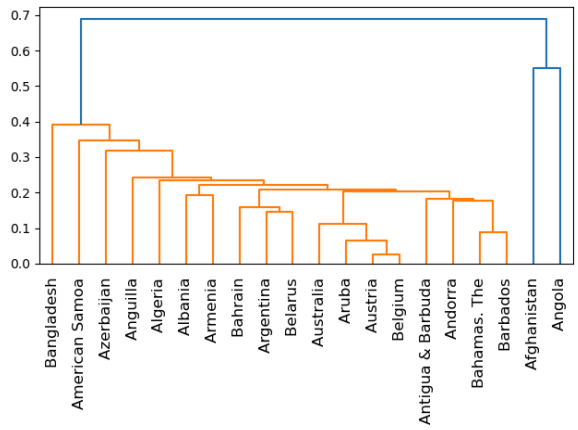
Details. For each of the 6 statistics, calculate the mean and standard deviation across the input data. Use the equation below to calculate the normalized feature values for each data point.

$$\frac{x - col\_min}{col\_max - col\_min}$$

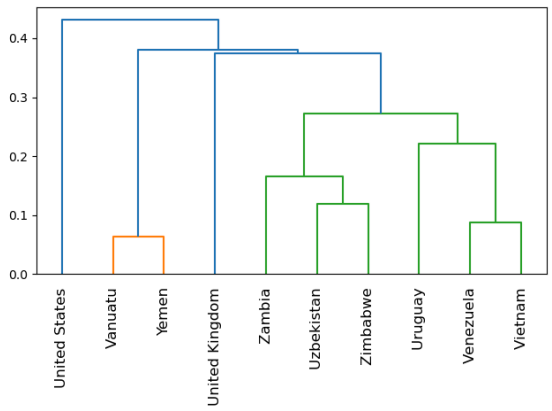
x represents the original value, *col_min* represents the minimum value in the column, and *col_max* represents the maximum value in the column. As a result, every value in the statistics will fall in range between [0, 1]. Applying HAC on the normalized data should produce the plots below.

The visualizations for the first 20 and final 10 countries

N = 20



N = 10



Discussion. These results seem to cluster countries with similar socioeconomic statuses. Normalizing the data creates a similar range of values for all statistics, so they are able to equally contribute to the Euclidean distance.

0.6 Testing

To test your code, try running the following lines in a main method or in a Jupyter notebook for various choices of n, notice that if you want final n rows, you should use [-n:] to replace [:n] :

```
data = load_data("countries.csv")
country_names = [row["Country"] for row in data]
features = [calc_features(row) for row in data]
features_normalized = normalize_features(features)
n = ???
Z_raw = hac(features[:n])
Z_normalized = hac(features_normalized[:n])
fig = fig_hac(Z_raw, country_names[:n])
plt.show()
```

To help you test your code, we have provided the correct output of hac using normalized data for the first 50 countries in 'output.txt'. Note that we still normalize features using the data of all countries, but we select the first 50 normalized data for hac. You should closely follow the test code above if you want to compare your output with 'output.txt'. You can also compare your output of fig_hac to the graphs provided in this writeup.

For the hac function, we will test on both small and large values of *n* up to 204. With the entire dataset, your code should not take more than 15 seconds to run. We will test on n <= 30 for fig_hac.

Extra part: Visualize them on a world map (Not graded)

To better understand what this clustering looks like, we provide a function named *world_map* in **function.py** to visualize the clustering on a world map. You can copy the code into your hw4.py after you complete the assignment's main part.
To use this function, we first need to install the *geopandas* package using the following command.

```
pip install geopandas==0.14.4
```

Note: if you install version >= 1.0.0, you need to alter the code because geopandas.dataset module is deprecated and removed in 1.0.0

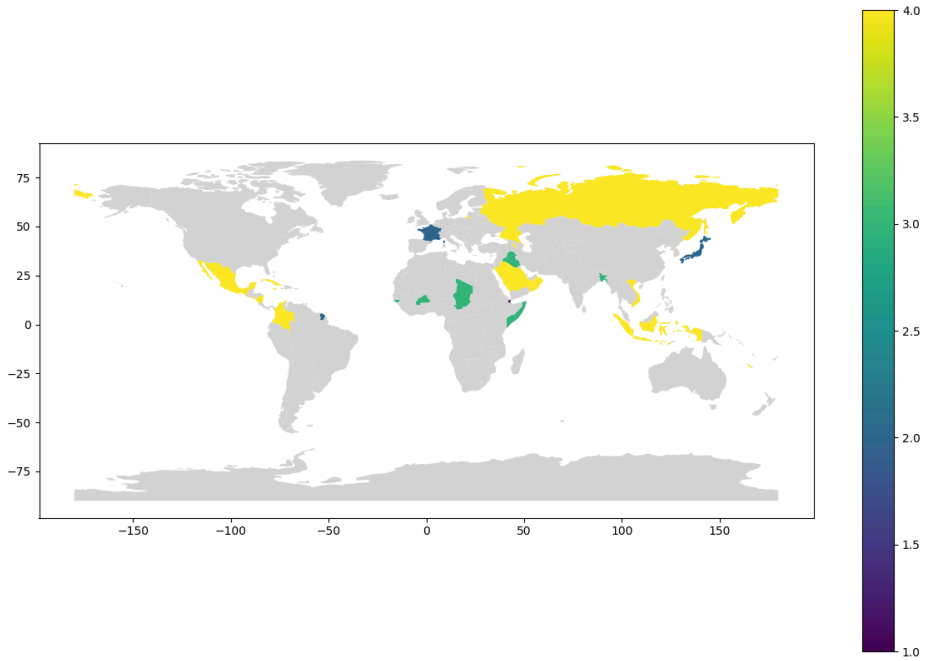
Summary of function

- Input: NumPy array *Z* output from hac, and list of string names corresponding to country names with length *n*, and number of clusters *K.*
- Output: A matplotlib figure with a graph that visualizes clustering

Try it out with different inputs to see yourself!
For example, you can use the following code to select random countries and run the function

```
import random
random_indices = random.sample(range(0, len(names)), 30)

random_names = [names[i] for i in random_indices]

random_features = [features_normalized[i] for i in random_indices]

Z = hac(random_features)

world_map(Z, random_names, 5)
```

Note: since the set of countries' names in the package is not exactly the same as our provided file, some of the countries might not be visualized correctly. You can ignore this problem when you play around with the code.
Here is a sample output of the code, feel free to alter the code and the parameter in world_map function to change the color or outlook of the function.



Submission Details

- Please submit a file named hw4.py on Gradescope.
- All code should be contained in functions or under a 'if __name__=="__main__":'

- Be sure to remove all debugging output before submission.
- The assignment is due Monday, October 7 at 9:30am.