CS540 Spring 2025 Homework 4

Due: Monday, February 24th, 2025, 11:59PM

1 Assignment Goals

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

2 Summary

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, each country is represented with a 9-dimensional feature vector $(x_1, \ldots, x_9)$, using the following statistics in the provided **Country-data.csv** file:

- **child_mort** - Death of children under 5 years of age per 1000 live births
- **exports** - Exports of goods and services per capita. Given as %age of the GDP per capita
- **health** - Total health spending per capita. Given as %age of GDP per capita
- **imports** - Imports of goods and services per capita. Given as %age of the GDP per capita
- **income** - Net income per person
- **inflation** - The measurement of the annual growth rate of the Total GDP
- **life_expec** - The average number of years a new born child would live if the current mortality patterns are to remain the same
- **total_fer** - The number of children that would be born to each woman if the current age-fertility rates remain the same.
- **gdpp** - The GDP per capita. Calculated as the Total GDP divided by the total population.

2.1 Clustering Method

After each country is represented by its feature vector $(x_1, \ldots, x_9)$, hierarchical agglomerative clustering (HAC) will be applied. This clustering approach will allow visualization of countries with similar socioeconomic conditions.

2.2 Implementation Constraints

- The use of `scipy.cluster.hierarchy.linkage()` is strictly prohibited and will result in a zero score.
- You may use Python's built-in standard library, along with `numpy`, `scipy` (excluding `hierarchy.linkage`), and `matplotlib`.

3 Program Overview

The data in CSV format can be found in the file `Country-data..csv`. Note that 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 several Python functions for this assignment.**(Note: the functions have to be named exactly as in the writeup)** Below is a high-level description of each function:

1. **load_data(filepath)** — Takes in a string with a path to a CSV file and returns the data points as a list of dictionaries. (Section 4.1)
2. **calc_features(row)** — Takes in one row dictionary from the data loaded from the previous function, calculates the corresponding feature vector for that country as specified, and returns it as a NumPy array of shape (9,). The dtype of this array should be `float64`. (Section 4.2)
3. **hac(features)** — Performs complete linkage hierarchical agglomerative clustering on the countries using the $(x_1, \ldots, x_9)$ feature representation and returns a NumPy array representing the clustering. (Section 4.3)
4. **fig_hac(Z, names)** — Visualizes the hierarchical agglomerative clustering of the countries' feature representation. (Section 4.4)
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 4.5)

You may implement other helper functions as necessary, but these are the functions being tested. In particular, your final Python file should only contain function definitions without code running outside of functions. To test your code, you may use a "main" method. Ensure that any test code is either deleted or wrapped within:

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

This is further discussed in Section 4.6.

4 Program Details

4.1 load_data(filepath)

Summary: [10pts]

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

Details:

1. Read in the file specified by `filepath`. The `DictReader` from Python's `csv` module is useful, but depending on your Python version, it might return `OrderedDicts` instead of normal dictionaries. Ensure conversion to a standard dictionary if necessary.
2. Return a list of dictionaries, where each row in the dataset is represented as a dictionary with the column headers as keys and the row elements as values.

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

4.2 calc_features(row)

Summary: [10pts]

- **Input:** A dictionary representing one country.
- **Output:** A NumPy array of shape (9,) and dtype `float64`, the first element is x_1 and so on with the last element being x_9.

Details: This function takes as input a dictionary representing one country and computes the feature representation $(x_1, \ldots, x_9)$. Specifically:

$$x_1 = \textbf{child\_mort}$$
$$x_2 = \textbf{exports}$$
$$x_3 = \textbf{health}$$
$$x_4 = \textbf{imports}$$
$$x_5 = \textbf{income}$$
$$x_6 = \textbf{inflation}$$
$$x_7 = \textbf{life\_expec}$$
$$x_8 = \textbf{total\_fer}$$
$$x_9 = \textbf{gdpp}$$

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: $x_1, \ldots, x_9$. The shape of this array should be $(9,)$. 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.

4.3 hac(features)

Summary: [50pts]

- **Input:** A list of NumPy arrays of shape $(9,)$, where each array is an $(x_1, \ldots, x_9)$ feature representation as computed in Section 4.2. The total number of feature vectors, i.e., the length of the input list, is n. Note that we test your code on different values of n as stated in Section 4.6.
- **Output:** A 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 i^{th} iteration of the clustering algorithm. Then,

  $$Z[i, 2] = d(Z[i, 0], Z[i, 1])$$

  is the complete linkage distance between the two clusters that were merged in the i^{th} iteration (this will be a real value, not an 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 that the original countries are considered clusters indexed by $0, \ldots, n-1$, and the cluster constructed in the i^{th} 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 complete linkage, perform the hierarchical agglomerative clustering algorithm as detailed in the 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 responsible for any reductions in points you might get for using a different 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 complete-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. You should be able to run HAC efficiently with all 167 countries. 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 that 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 complete-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()`.

4.4 fig_hac(Z, names)

Summary: [10pts]

- **Input:** A NumPy array Z output from `hac`, and a 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 with:

```
labels=names, leaf_rotation = ???
```

Your plot will likely cut off the x labels. For the graph to look like the examples below, you will need to call `tight_layout()` on the figure.

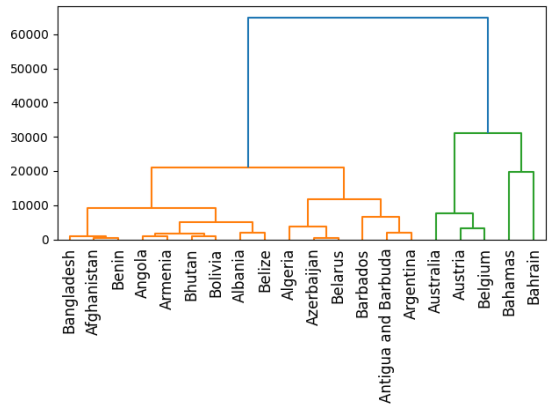

Figure 1: The visualization for the first 20 countries ($N = 20$).

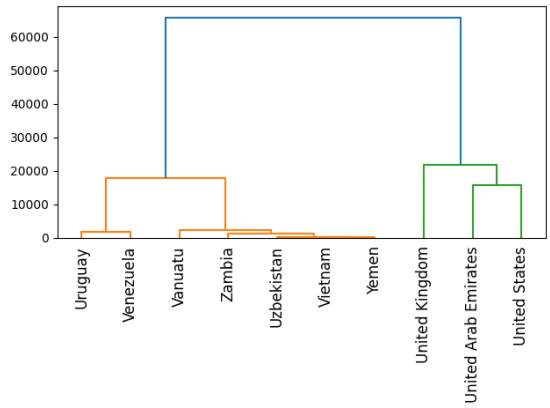

Figure 2: The visualization for the final 10 countries ($N = 10$).

Discussion. Notice that the figures above cluster the countries are affected by some large value in the dataset, such as **income** and **gdpp**, which are much larger than the values for other columns in the data. Therefore, they disproportionately contributes to the Euclidean distance between feature vectors. Ideally, all nine statistics should contribute to the clustering, so you will create a function to normalize the data in the next section.

4.5 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 shape $(9,)$ and dtype `float64`.
- **Output:** The output should have an identical format to the input: a list of NumPy arrays with shape $(9,)$ and dtype `float64`. However, the statistic values in the feature vectors should be replaced with their normalized values.

Details. For each of the 9 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:

$$x' = \frac{x - \mu}{\sigma}$$

where x represents the original value, μ represents the column's mean, and σ represents the column's standard deviation.

Applying HAC on the normalized data should produce the plots below.

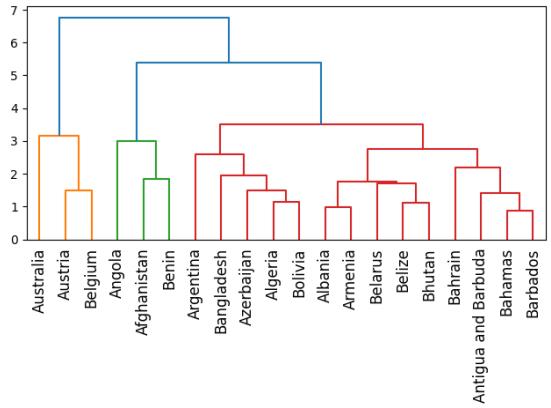

Figure 3: The visualization for the first 20 countries ($N = 20$) after normalization.

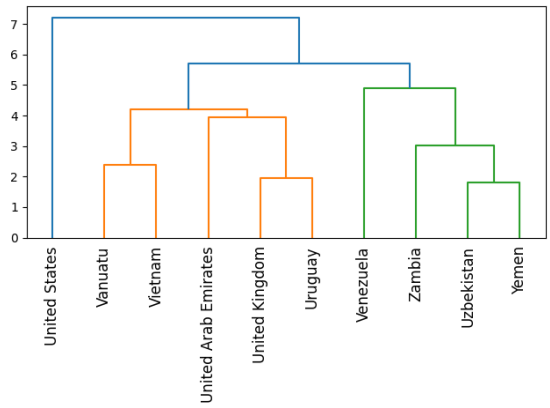

Figure 4: The visualization for the final 10 countries ($N = 10$) after normalization.

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.

4.6 Testing

To test your code, try running the following lines in a main method or in a jupyter notebook for various choices of n:

```
data = load_data("Country-data.csv")
features = [calc_features(row) for row in data]
names = [row["country"] for row in data]
features_normalized = normalize_features(features)
np.savetxt("output.txt", features_normalized)
n = 20
Z = hac(features[:n])
fig = fig_hac(Z, 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 167. With the entire dataset, your code should not take more than 15 seconds to run. We will test on $n \leq 30$ 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.

4.6.1 Installation

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 $\geq$ 1.0.0, you need to alter the code because the `geopandas.dataset` module is deprecated and removed in 1.0.0.

4.6.2 Summary of the function

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

Try it out with different inputs to see the results!

4.6.3 Example Usage

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)), 100)
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, 10)
```

Note: Since the set of country names in the package is not exactly the same as in our provided file, some of the countries might not be visualized correctly. You can ignore this problem when experimenting with the code.

4.6.4 Sample Output

Here is a sample output of the code. Feel free to alter the code and the parameters in the `world_map` function to change the color or appearance of the visualization.

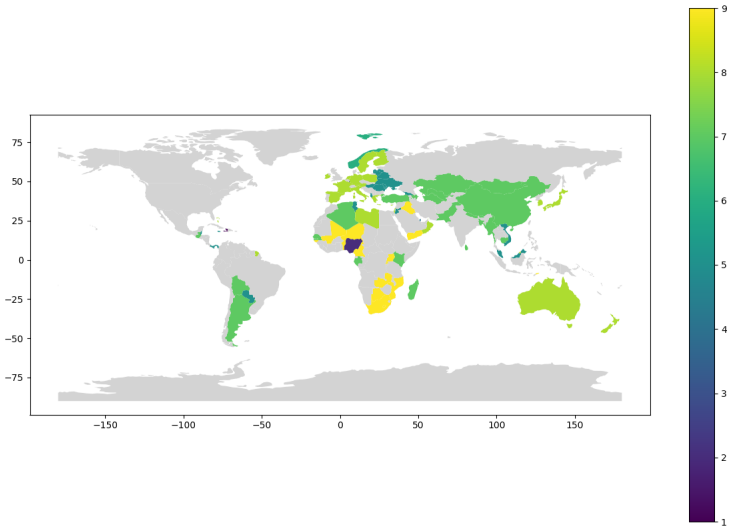

Figure 5: Sample world map visualization of hierarchical clustering.

5 Submission Notes

- Please submit one file named `hw4.py` to Gradescope. Do *not* submit a Jupyter notebook `.ipynb` file.
- 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, February 24th, 11:59PM

ALL THE BEST!