CD 732: Data Visualization
Report for Datathon-4

Swasti Shreya Mishra (IMT2017043)
International Institute of
Information Technology Bangalore
Email: SwastiShreya.Mishra@iiitb.org

1. Introduction

We are given the tabular datasets published by the World Health Organization for COVID-19 cases. These include State/Country/Reign-wise time series data (over a period of 246 days starting from January 2020) for confirmed cases, recovered cases and deaths.
Our main aim is to effectively visualize the above data as a matrix and find the underlying patterns or clusters formed using two different methods of matrix seriation which are – Fast optimal leaf ordering and Seriation using Traveling Salesperson Problem formulation.

2. Methods

The data is mainly visualized using *seaborn*, along with *ortools*, *matplotlib*, *pandas* and *scipy* packages to compute various metrics for matrix seriation implementation.

2.1. Matrix Seriation

Seriation is a statistical method that seeks the best enumeration order of a set of described objects. This approach can be used to reorganize rows or columns of a dataset so as to enumerate them in an appropriate order. We use different criteria built from matrix D of dissimilarities or distances between elements to be seriated [1].

2.1.1. Fast Optimal Leaf Ordering (for hierarchical clustering). Ziv Bar-Joseph et al [2] proposed this algorithm where they show how optimal leaf ordering can be used on top of hierarchical clustering to find out underlying patterns of data. They define optimal leaf ordering problem as follows:

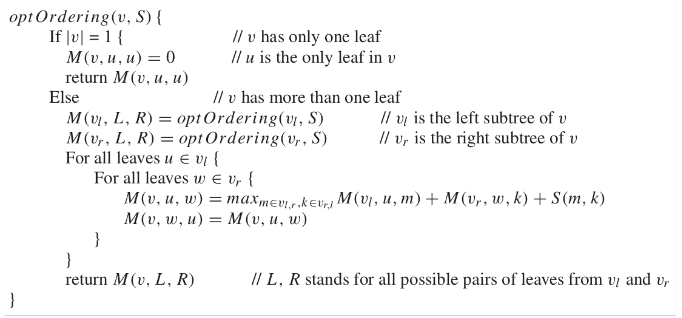
For a tree T with n leaves, where the leaves are denoted by $z_1, ..., z_n$ and the internal nodes are denoted by $v_1, ..., v_{n-1}$. A linear ordering consistent with T is defined to be an ordering of the leaves of T generated by flipping internal nodes in T (that is, changing the order between the two subtrees rooted at v_i, for any $v_i \in T$).

There are can be 2^{n-1} possible linear orderings of the leaves of the tree as T comprises of $n-1$ internal nodes. Then they find an ordering of the tree leaves, that maximizes the sum of the similarities of adjacent leaves in the ordering.

The algorithm uses a bottom up dynamic programming approach to solve the problem. For each internal node v it finds the cost of the optimal ordering of the subtree rooted at v, denoted by $M(v)$. The left and right subtree rooted at v is denoted by v_l and v_r. For every pair of leaves $u \in v_l$ and $w \in v_r$, they compute $M(v, u, w)$ which is the optimal linear ordering of v when the leftmost leaf of v is u and the rightmost leaf of v is w. Therefore, the function M can be defined as:

$$M(v,u,w) = max_{m \in v_{l,r}, k \in v_{r,l}} M(v_l, u, m) + M(v_r, w, k) + S(m,k)$$

where $v_{l,r}$ is the right subtree of v_l and $v_{r,l}$ is the left subtree of v_r and S is the similarity matrix. This algorithm is shown in figure 1. To make this algorithm more efficient Ziv Bar-Joseph et al introduce Early Termination of the search. This is shown in figure 2.



```
$optOrdering(v, S)$ {
    If $|v| = 1$ {                      // $v$ has only one leaf
        $M(v, u, u) = 0$                // $u$ is the only leaf in $v$
        return $M(v, u, u)$
    Else                                // $v$ has more than one leaf
        $M(v_l, L, R) = optOrdering(v_l, S)$        // $v_l$ is the left subtree of $v$
        $M(v_r, L, R) = optOrdering(v_r, S)$        // $v_r$ is the right subtree of $v$
        For all leaves $u \in v_l$ {
            For all leaves $w \in v_r$ {
                $M(v, u, w) = max_{m \in v_{l,r}, k \in v_{r,l}} M(v_l, u, m) + M(v_r, w, k) + S(m, k)$
                $M(v, w, u) = M(v, u, w)$
            }
        }
        return $M(v, L, R)$             // $L$, $R$ stands for all possible pairs of leaves from $v_l$ and $v_r$
}
```

Figure 1. The recursive algorithm for computing $M(v, L, R)$ [2]

My implementation of the algorithm is as follows:

1) We compute the *row order* and *column order* of the tree. The row order of the tree is defined as the optimal ordering of the leaves of the distance matrix of the data given as input. Similarly, the column order is the optimal ordering of the leaves of the distance matrix of the *transpose* data given as input.
2) The distance is either directly given as input to the algorithm, or, it is computed using different metrics

```
$curMax = -\infty$
For all leaves $m$ according to the order of $M(v_l, u, R)$ {
    if $M(v_l, u, m) + M(v_r, w, k_0) + C(v_l, v_r) \leq curMax$        // $k_0$ is first in the order of $M(v_r, w, L)$
      $M(v, u, w) = curMax$, terminate the search
    For all leaves $k$ according to the order of $M(v_r, w, L)$
      if $M(v_l, u, m) + M(v_r, w, k) + C(v_l, v_r) \leq curMax$
        break                        // terminate the inner loop
      if $curMax < M(v_l, u, m) + M(v_r, w, k) + S(m, k)$
        $curMax = M(v_l, u, m) + M(v_r, w, k) + S(m, k)$
    }                            // end of the inner loop
}                            // end of the outer loop
$M(v, u, w) = curMax$
```

Figure 2. The improved algorithm for computing $M(v, L, R)$ [2]

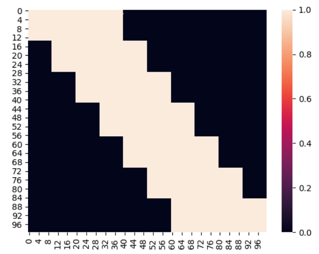


Figure 3. Dummy data with an underlying pattern

of the *scipy.spatial.distance.pdist* function. The tree for both row and column ordering is computed using *scipy.cluster.hierarchy.linkage* function which takes the respective distances.

3) The optimal ordering of the leaves for both row and column order are computed using the algorithm defined in figure 2.
4) The *row order* and *column order* obtained are used to rearrange the rows and columns of the data matrix. This is done by converting the matrix to *pandas* DataFrame.

We can measure the performance of optimal leaf ordering on some synthetic data and infer a few results. The figure 3 represents the dummy data which is a 100 x 100 binary matrix. The figure 4 is achieved after randomly shuffling the rows and columns of the dummy data. The resulting seriated-matrix using fast optimal leaf ordering (OLO) is shown in figure 5. The matrixes are visualized using python's *seaborn* library. The algorithm performs well in this case, but when the data gets more complex, it performs relatively poorly. This is due to the fact that we start with hierarchical clustering. We will compare the results of fast OLO over a more complex dummy data in the next section.

2.1.2. Traveling Salesperson Problem formulation. In the previous algorithm we started with clustering because it intuitively makes sense to group similar rows together, and then we ordered the tree branches to maximize the sum of similarities between adjacent rows (and the same for columns, independently). But instead of clustering the rows and columns, we can just find the order of the rows (and columns) that minimizes the sum of dissimilarities (unconstrained by the clustering tree). This is similar to the Traveling Salesperson Problem (TSP), where we find the

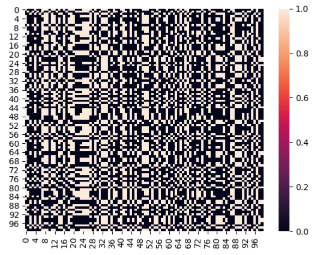


Figure 4. Shuffling the data in figure 3

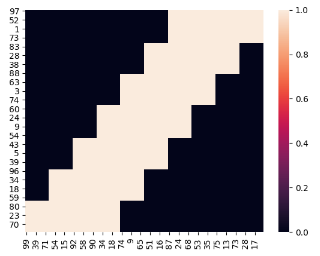


Figure 5. Seriated matrix obtained after applying fast OLO on figure 4

shortest path that visits every city in a set and comes back to its starting point. In the case of seriation, we don't need to worry about coming back to the starting point as we don't care how dissimilar the first and last rows (and columns) of the dissimilarity matrix are. This problem can be reduced to a TSP by the addition of a dummy row with 0 distance to all the others, and then cutting the result at that point [3].

My implementation of the algorithm is as follows:

1) We compute the *row order* and *column order* of distance matrix. The row order is defined as the TSP routing ordering of the rows of the distance matrix of the data given as input. Similarly, the column order is the TSP routing ordering of the columns of the distance matrix of the *transpose* data given as input.
2) The TSP routing order is computed using *ortools* package from Google. We first create a *routing model* by using the distance matrix, specifying the *number of vehicles* as 1 and setting the *start index of the route* as 0 (as data is zero-indexed).
3) We now define a *distance callback* for the routing model to compute the distance from a source vertex to a destination vertex. We then set the *cost of travel* as this *distance callback*.
4) Next, the solution strategy is set to the *cheapest path* parameter. This is internally implemented using heuristics which generally perform better than the naive TSP solver. The reason behind using this is that, the method can both scale well with increasing number of rows and columns, as well as approximate the values well.

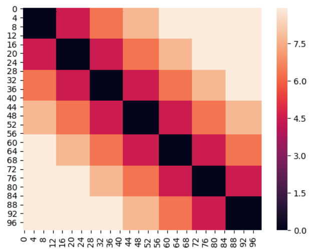


Figure 6. Dummy data with an underlying pattern

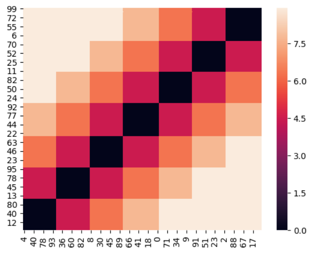


Figure 8. Seriated matrix obtained after applying TSP solver on figure 7

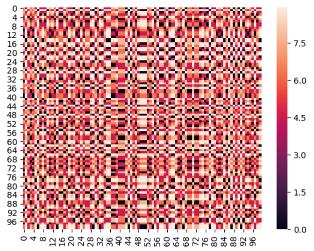


Figure 7. Shuffling the data in figure 6

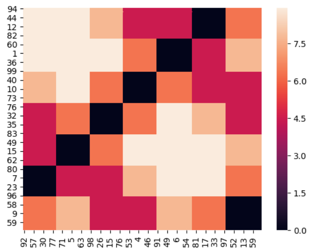


Figure 9. Seriated matrix obtained after applying fast OLO on figure 7

5) Finally, the routing order is obtained by routing all the indices of the rows (and columns). The *row order* and *column order* thus obtained are used to rearrange the rows and columns of the data matrix. This is done by converting the matrix to *pandas* DataFrame.

We can measure the performance of TSP solver on synthetic data. We also compare it to the results given by fast OLO. The figure 6 represents the dummy data which is a 100 x 100 matrix. The figure 7 is achieved after randomly shuffling the rows and columns of this dummy data. The resulting seriated-matrix using TSP solver is shown in figure 8. We also compare it with the resulting seriated-matrix using fast optimal leaf ordering (OLO) as shown in figure 9. The matrixes are visualized using python's *seaborn* library.

From the above example we can see that TSP solver outperforms fast OLO majority of the time. This is because no implicit clustering of data is assumed in TSP solver.

2.2. Data Pre-processing

The raw data has the time series cases for different regions. This data is aggregated based on the countries they belong to. Now, we have a country-wise time series data which can be used to find correlation or similarity of progression of COVID-19 between different countries.

2.2.1. Correlation Matrix. We have a time series dataset for various countries of the world. For this data, we can directly derive a correlation matrix (correlation/similarity values between countries) which will help us determine the correlation scores of progression of deaths/confirmed-cases/recoveries between two different countries.

2.2.2. Weighted Average Time Matrix. In order to transform a time series data into a matrix, various aggregation techniques can be used. We can define the distance between the countries on the basis of approximate time taken for the explosion of COVID-19 cases, this is as per the source [4]. We calculate a weighted average of time taken for a country to incur the total number of deaths till the last date in the dataset. The function used is as follows:

$$deaths(i) = \text{Cumulative death count of } i^{th} \text{ day}$$
$$average\_time(n) = \frac{\sum_{i=1}^{i=n} deaths(i) * i}{\sum_{i=1}^{i=n} deaths(i)}$$

Here, n is the n^{th} day until we are calculating the average time of cases.
Now, distance for country1 (say u) and country2 (say v) can be computed using:

$$\text{distance(u,v) = u[average\_time(n)] - v[average\_time(n)]}$$

The above distances give us the information that, if the progression of COVID-19 was similar in two countries, the distance between them is smaller.

3. Visualizations

The visualization of deaths is more appropriate as, we cannot rely on the number of confirmed cases for every country. For many countries, testing the entire population is not possible. Also, we cannot determine exact the number

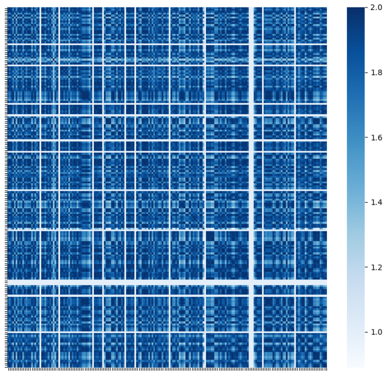


Figure 10. Raw matrix for correlation values of deaths

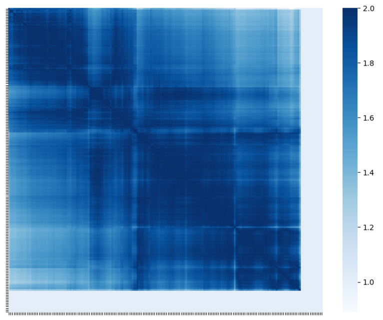


Figure 11. Seriated matrix for correlation values of deaths

of infected people since COVID-19 is characterized by a great number of asymptomatic people. When someone dies, the tests tend to be executed with a higher probability than for those who are infected with no symptoms. This idea is adopted as per source [4].

The matrix obtained before seriation is shown in figure (The country names are deliberately eliminated from the plot as the aim is to show the randomness in data). The figure shows how the matrix has transformed after seriation, here using TSP solver (The country names are again eliminated as this image is added to show the comparision in raw matrix and seriated matrix).

The following are some of the clusters that are obtained both using the TSP solver and fast OLO seriation over the Correlation Matrix. Correlation scores are converted to similarity scores by addind 1 to all the values to make the values lie in the range [0,2]. This is done to ensure that there are no negative distances in any of the computed matrices for TSP solver and fast OLO.

Similar clusters are obtained for the Weighted Average Time Matrix as well. But here the scale is different. Along with the scale, the order of the *colormap* used is revered as here we are using distance values which correspond to dissimilarity score. We feel this is an okay assumption to make as we only intend to find the patterns in data and therefore maintain coherence in visualization of similarity and dissimilarity. The figure shows one of the examples.

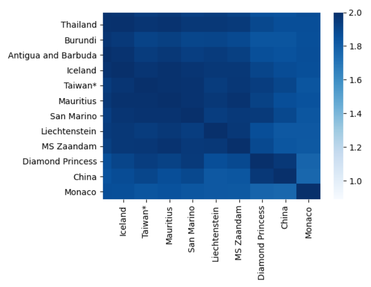


Figure 12. Diamond Princess and China cluster for correlation metric using fast OLO

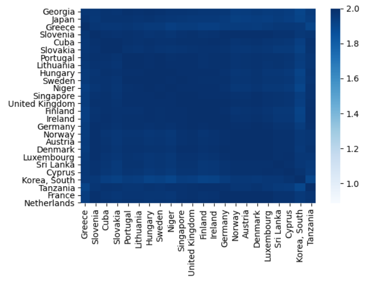


Figure 13. United Kingdom cluster for correlation matrix using fast OLO

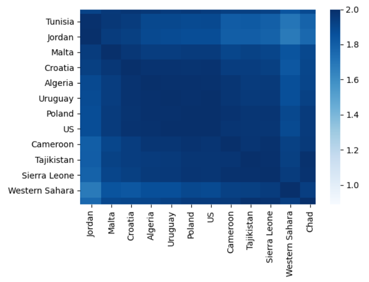


Figure 14. United States cluster for correlation matrix using fast OLO

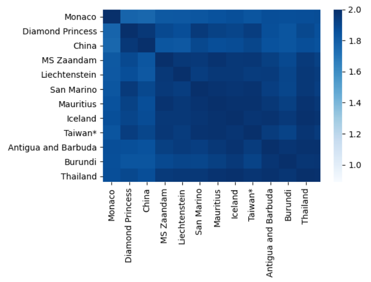


Figure 15. Diamond Princess and China cluster for correlation matrix using TSP solver

4. Inferences

We can observe from the visualizations that both correlation martix and weighted average time matrix produces similar clusters. In the previous datathon we had to threshold the networks and then visualize the underlying clusters.

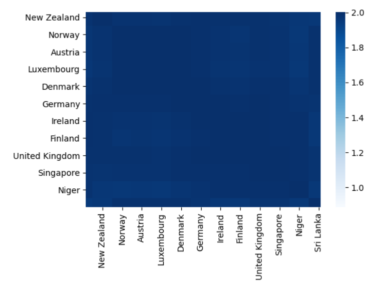


Figure 16. United Kingdom cluster for correlation matrix using TSP solver

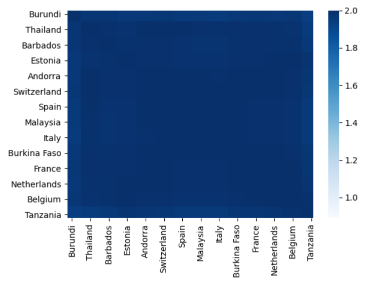


Figure 17. European cluster for correlation matrix using TSP solver

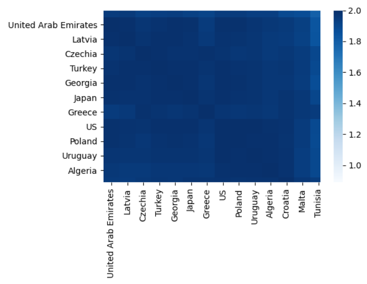


Figure 18. United States cluster for correlation matrix using TSP solver

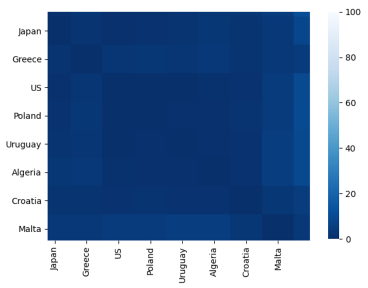


Figure 19. United States cluster for weighted average time matrix using TSP solver

Using seriation, this step can be completely eliminated. It also makes the experiments independent of the actual values of the similarity/dissimilarity.

Using the visualizations produced by seriation of matrix, we can verify the clusters that we obtained in the last datathon. Some of them were, the European cluster 16 and the United States 18 cluster. Even here we can see similar clusters forming. Another new observation here is that, Diamond Princess and China are very closely related 12, both in the case of deaths as well as confirmed cases. Some of the observations like figure 17 also show the geographical clustering that is evident in the data.

5. Conclusion

The possibilities of interpretations of the data we have visualized is very large. I have only been able to look at a few of them but they provide us with a lot of information. I have learnt that seriation is both faster and better than clustering. For example: one of the problems with agglomerative clustering is that it doesn't actually place the rows in a definite order, it merely constrains the space of possible orderings. Take three items A, B and C. If you ignore reflections, there are three possible orderings: ABC, ACB, BAC. If clustering them gives you ((A+B)+C) as a tree, you know that C can't end up between A and B, but it doesn't tell you which way to flip the A+B cluster. It doesn't tell you if the ABC ordering will lead to a clearer-looking heatmap than the BAC ordering. Using fast OLO we can overcome this issue [3].
Note: I have used *time_series_covid_19_deaths.csv*, *time_series_covid_19_recoveries.csv* and *time_series_covid_19_confirmed.csv* for visualization of the above results. I have also taken into account my previous datathon's results and visualizations.

References

[1] Seriation.

[2] Ziv Bar-Joseph, David K Gifford, and Tommi S Jaakkola. Fast optimal leaf ordering for hierarchical clustering. *Bioinformatics*, 17(suppl_1):S22–S29, 2001.

[3] Nicolas Kruchten.

[4] Graph theory: Covid-19 spreading clustering, Jun 2020.