Linear Algebra and Optimization for Machine Learning: Project 1

Zhaonan Meng (5841003) Alvedian Mauditra Aulia Matin (5689252) Zuhair (5523435)

October 2022

1 K-means Clustering

In this section, the first part of our project is presented. First, we will explain how K-means clustering of the given dataset is done using standard squared Euclidean distance. Then, we will report the process and results of clustering using kernelized distance. The code for this part is in `Kmeans_clustering.py`.

1.1 K-means using standard squared Euclidean distance

Before we begin the clustering, we preprocess our data first. We ensure that there are no missing data or NaN/null values of features for each data entry. Then, we ensure that every feature is numerical data. It turns out that the given data is already clean, and all features are in `int64` datatype, which is numerical, even though `cc1_miles`, `cc2_miles`, and `cc3_miles` can be seen as categorical, and `Award?` as boolean. However, for simplicity and to prevent too many additional features (if we applied dummy variables or one-hot encoding), which could lead to a hard-to-interpret result of clustering, we use the mentioned features as they are already given (numerical data). Then, we described the statistics of all features to ensure they are on the same scale. Since the distributions of features are highly diverse, we implement a standard scaling (`standard_scaling()`), such that every feature has a mean of approximately zero and a variance of one. Now, our dataset is clean and ready to be clustered.

Since we will not cluster our data based on the ID of passengers, we drop the `#ID` feature and implement K-means clustering on the other 11 features. Then, we analyze the result of the clustering and try to get the interpretation for each cluster.

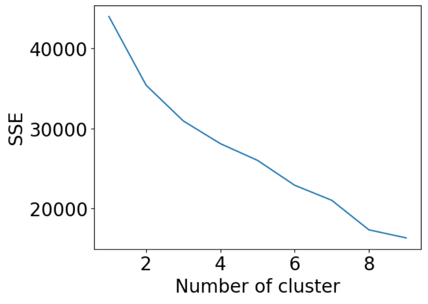


Figure 1.1: The elbow plot.

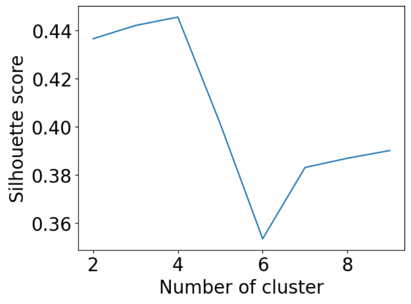


Figure 1.2: The silhouette plot.

To decide the optimal number of clusters (K), we utilize the elbow and silhouette methods. For the elbow method, we plot the sum of squared errors (SSE):

$$SSE(C_1, \dots, C_K) = \sum_{k=1}^{K} \sum_{i \in C_k} \|x_i - c_k\|_2^2,$$

where c_k is the center of the k-th cluster, for different number of clusters ($1 \leq K < 10$), which is called as an elbow plot (`SSE_scratch()`). Then, we ideally choose the number of clusters that gives the largest drop in SSE. For the silhouette method, we calculate the silhouette score, which measures the similarity of objects in their cluster compared to other clusters, for ($1 < K < 10$) (`silhouette_score_scratch()`). Then, we choose K be-

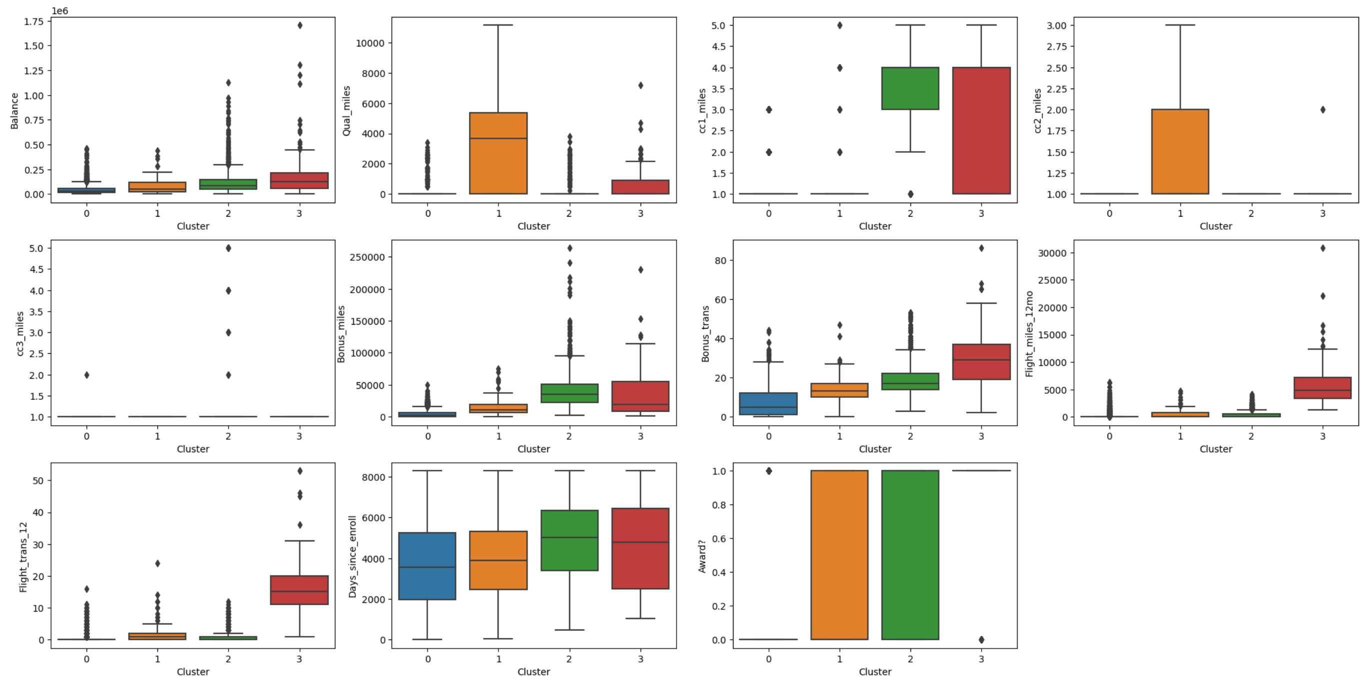


Figure 1.3: The distribution of each feature grouped by clusters. The result of clustering using our K-means standard squared distance.

fore the sudden drop in the silhouette score. For our dataset, we obtain the elbow plot in Fig. 1.1 and the silhouette plot in Fig. 1.2. From Fig. 1.1, it is quite hard to see the 'elbow' of the graph because the decrease of the SSE seems generally steady. However, we can see a slightly more significant drop at $K = 2$ and $K = 4$, where SSE of $K = 4$ is less than half of $K = 2$. From Fig. 1.2, we know that $K = 2$ gives the largest silhouette score, but $K = 4$, which has a slightly lower score, is just before a significant drop. Thus, based on these two plots, we choose $K = 4$.

Then, clustering using our standard Euclidean distance K-means clustering (`Kmeans_Eucl()`) is conducted. We use the whole data set (3999 rows) and all features except the `#ID` feature. The distribution of each feature grouped by the obtained clusters is depicted in Fig. 1.3. We also calculate the number of data points of each cluster and the correlation between features (heat map). Based on the distributions, we infer the following interpretation of clusters:

- **Cluster 0**: the largest cluster (2528 data points) but has the lowest mean values for all features. We call this cluster as **"Ordinary Passengers"**.

- **Cluster 1**: the smallest cluster (89 data points) and the only cluster of passengers that actively used Rewards credit card to earn the number of miles (`cc2_miles`) and had the highest mean of the number of miles qualifying for Topflight status (`Qual_miles`). It has the second highest averages of the number of flights (`Flight_miles_12mo`) and the number of transactions (`Flight_trans_12`), which those two features have a high positive correlation (0.87). Thus, we name it as the cluster of **"Rewards Credit Card Users (Active passengers)"**.

- **Cluster 2**: the distributions for all features are quite similar to Cluster 1 but has larger data points (1245) and has the largest mean of number of miles earned with frequent flyer credit card (`cc1_miles`) and number of non-flight bonus transactions (`Bonus_miles`), where both features have high positive correlation (0.83). Hence, this is the cluster of **"Freq. Flyer Credits Card Users (Active passengers)"**.

- **Cluster 3**: the highest mean for almost all features, which is the cluster of **"Top 4% passengers"** (137 data points). This cluster has significantly higher `Flight_miles_12mo` and `Flight_trans_12` than other clusters.

If we obverse the last box plot in Fig. 1.3 (`Award?`), we know that almost all the **"Ordinary Passengers"** (Cluster 0) had no award flight, whereas; in contrast, almost all the **"Top 4% passengers"** had it. The **"Credit Cards Users (Active Passengers)"** (Cluster 1 and 2) had uniform distributions of receiving free flights.

From the results of clustering, we obtain the prominent features that distinguish the passengers, which are `cc2_miles` or `Qual_miles`, `cc1_miles` or `Bonus_miles`, and `Flight_miles_12mo` or `Flight_trans_12`. Then, we choose `cc2_miles`, `cc1_miles`, and `Flight_trans_12`, only to visualize the result of our clustering based on the eleven features using the whole dataset (Fig. 1.4).

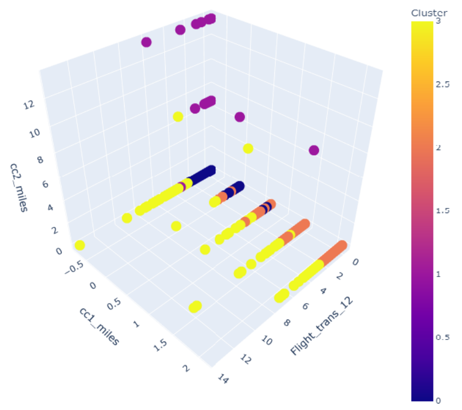


Figure 1.4: The scattered plot of the three features with clusters based on K-means using eleven features.

The SSE and silhouette score of the clustering are 28090.01 and 0.45, respectively, with the running time of 4.47 seconds.

1.2 K-means using kernelized distance

The data may be non-linearly separable, in which case it is useful to group the data into clusters using a kernel function. A K-means clustering algorithm is implemented with the Gaussian/Radial Basis Function kernel

$$K(x, x') = \exp\left(-\gamma \|x - x'\|_2^2\right),$$

where the parameter γ is decided by calculating the SSE of each cluster using the modified SSE formula

$$\sum_{k=1}^{4}\sum_{j=1}^{N} K(x_j, x_j) - \frac{2}{|C_k|}\sum_{i\in C_k} K(x_j, x_i) + \frac{1}{|C_k|^2}\sum_{i,l\in C_k} K(x_i, x_l).$$

As discussed in the previous subsection, we use $K = 4$ as the number of clusters. Then, we plot the SSE of clustering results against various values of γ (Fig. 1.5). From this plot, we can see that the decrease in SSE is far less significant when $\gamma < 10^{-2}$.

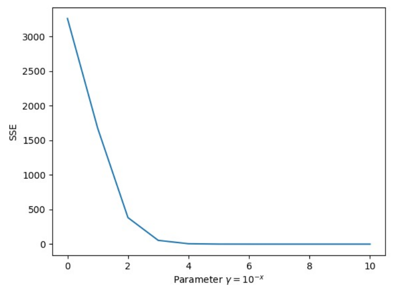
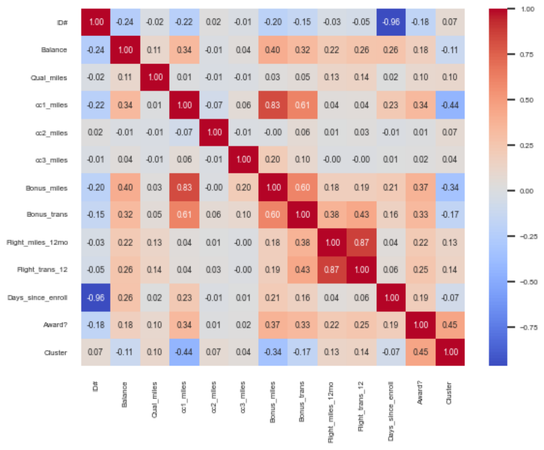


Figure 1.5: The elbow plot for kernelized K-means.

The whole dataset is used (3999 rows) and all features are taken into account except for `#ID`. Fig. 1.3 shows the distribution of each feature in the obtained clusters. The SSE of this result is 382.82.



	ID#	Balance	Qual_miles	cc1_miles	cc2_miles	cc3_miles	Bonus_miles	Bonus_trans	Flight_miles_12mo	Flight_trans_12	Days_since_enroll	Award?	Cluster
ID#	1.00	-0.24	-0.02	-0.22	0.02	-0.01	-0.20	-0.15	-0.03	-0.05	-0.96	-0.18	0.07
Balance	-0.24	1.00	0.11	0.34	-0.01	0.04	0.40	0.32	0.22	0.26	0.26	0.18	-0.11
Qual_miles	-0.02	0.11	1.00	0.01	-0.01	-0.01	0.03	0.05	0.13	0.14	0.02	0.10	0.10
cc1_miles	-0.22	0.34	0.01	1.00	-0.07	0.06	0.83	0.61	0.04	0.04	0.23	0.34	-0.44
cc2_miles	0.02	-0.01	-0.01	-0.07	1.00	-0.01	-0.00	0.06	0.01	0.03	-0.01	0.01	0.07
cc3_miles	-0.01	0.04	-0.01	0.06	-0.01	1.00	0.20	0.10	-0.00	-0.00	0.01	0.02	0.04
Bonus_miles	-0.20	0.40	0.03	0.83	-0.00	0.20	1.00	0.60	0.18	0.19	0.21	0.37	-0.34
Bonus_trans	-0.15	0.32	0.05	0.61	0.06	0.10	0.60	1.00	0.38	0.43	0.16	0.33	-0.17
Flight_miles_12mo	-0.03	0.22	0.13	0.04	0.01	-0.00	0.18	0.38	1.00	0.87	0.04	0.22	0.13
Flight_trans_12	-0.05	0.26	0.14	0.04	0.03	-0.00	0.19	0.43	0.87	1.00	0.06	0.25	0.14
Days_since_enroll	-0.96	0.26	0.02	0.23	-0.01	0.01	0.21	0.16	0.04	0.06	1.00	0.19	-0.07
Award?	-0.18	0.18	0.10	0.34	0.01	0.02	0.37	0.33	0.22	0.25	0.19	1.00	0.45
Cluster	0.07	-0.11	0.10	-0.44	0.07	0.04	-0.34	-0.17	0.13	0.14	-0.07	0.45	1.00

Figure 1.7: Heatmap of features.

- **Cluster 0**: The second-largest cluster, with 917 members. It has the highest `Bonus_miles` mean value and second-highest `Bonus_trans` mean values and this cluster's mean values of flight-related features are smaller compared to Cluster 3, making it

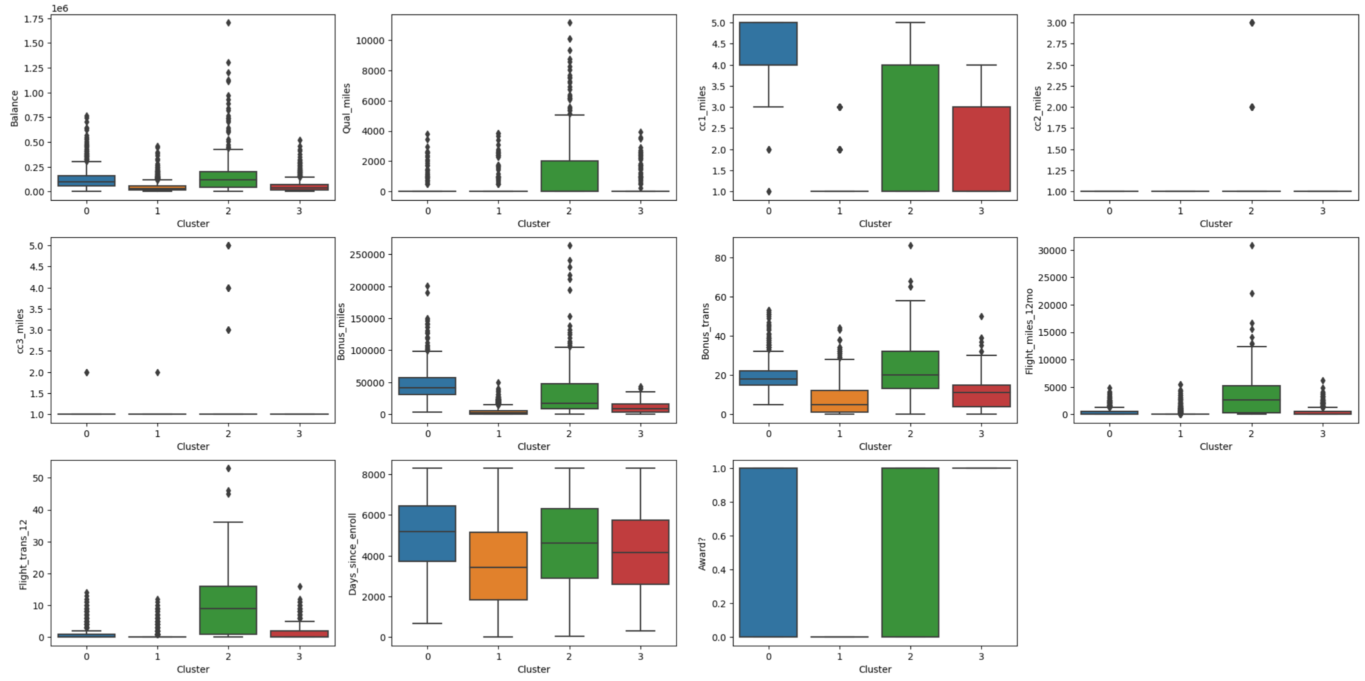


Figure 1.6: The distribution of each feature grouped by clusters. The result of clustering using our K-means kernelized distance.

the **"Bonus Focused"** cluster.

- **Cluster 1**: This cluster has the largest number of members (2046), and the lowest mean value for almost all features. This is the **"Ordinary Passengers"** cluster.

- **Cluster 2**: This cluster is the smallest, with 241 members. It has the highest mean values in almost every feature except for `cc1_miles`, `Bonus_miles`, and `Days_since_enrol`. This cluster's mean values for `Flight_trans_12` and `Qual_miles` far exceed all other clusters' by at least five times. This is the **"Top Frequent Flyers"** cluster.

- **Cluster 3**: This cluster has 795 members. It has the second-highest mean values of `Flight_trans_12`, `Flight_miles_12mo`, `Qual_miles`, and the highest mean value of `Award?`, with lower mean values of `Bonus_trans` and `Bonus_miles` compared to Cluster 0. This cluster is **"Flight-Focused."**

Since the features `Flight_trans_12` and `Flight_miles_12mo` are highly correlated, as are `Bonus_miles` and `Bonus_trans` (Fig. 1.7), one of each pair is chosen as a representative and plotted, along with the clustered data points.

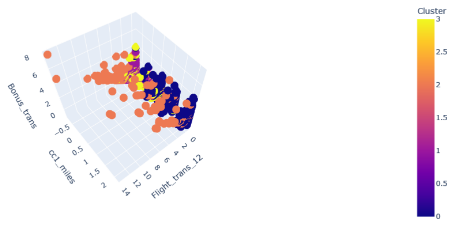


Figure 1.8: 3D plot of clustered data points against Flight_trans_12, Bonus_trans, and cc1_miles.

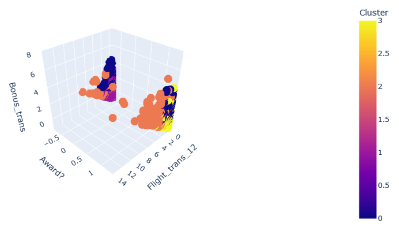


Figure 1.9: 3D plot of clustered data points against Flight_trans_12, Bonus_trans, and Award.

1.3 Comparison of K-means results

The running time needed for K-means clustering using the kernelized distance is 0.31 seconds, much faster than the Euclidean distance (4.47 seconds). However, if we compare the cluster distributions in Fig. 1.3 and Fig. 1.6, it is easier to get the interpretation of the clusters of the K-means using Euclidean distance.

In Fig. 1.3, the distributions of different clusters are quite distinguishable in some features. For instance, we can easily spot cluster 1 in `cc2_miles`, cluster 2 in `cc1_miles`, and cluster 3 in `Flight_trans_12`. Moreover, there is a recurring pattern in the distribution across different features, such as cluster 0 is always distributed mainly in the lowest level of the feature's values, followed by clusters 1, 2, and 3, respectively, which are distributed in higher values. Hence, we can see a 'stair-case' pattern of the distributions in some features, such as in `Balance`, `Bonus_miles`, `Bonus_trans`, `Flight_miles_12mo`, `Flight_trans_12`, and `Days_since_enroll`. In contrast, the distributions in Fig. 1.6 are less distinguishable between each cluster. Nevertheless, it still gives meaningful clusters, as explained in the previous subsection.

Additionally, the kernelized distance K-means clustering result only provides information on how passengers spend their miles, whereas the results from the Euclidean distance K-means clustering helps highlight how passengers who are not in the **"Ordinary Passengers"** or **"Top Passengers"** earn and spend their miles. Members of cluster 1 from the Euclidean distance clustering result use Rewards credit cards to earn miles and spend more miles on flights, while members of cluster 2 earn miles using frequent flyer credit cards and spend more miles on non-flight features. In contrast, the clusters obtained using kernelized K-means are difficult to distinguish by mean values of `cc2_miles` or `cc3_miles` (Fig. 1.6). On average, members of cluster 0 tend to spend more miles on non-flight features whereas members of cluster 3 tend to spend more miles on flights, but the clustering results do not help us discern how these passengers earned their miles. In conclusion, standard squared Euclidean distance K-means gives more interpretable results.

2 Spectral Clustering

Welcome to the second part of our project. Before starting answering the project questions 2.(a) and 2.(b), we believe it would be better to make sure the correctness and robustness of our spectral clustering algorithm at first. Therefore, in section 2.1, we will first show how our algorithm performs on some test data. And then we will apply the algorithm with two different eigensolvers to the given data set *EastWestAirlinesCluster.csv*.

2.1 Before the Airline data: Correctness Test

In this section, we show how our spectral clustering algorithm performs on some circle data (figure 2.1).

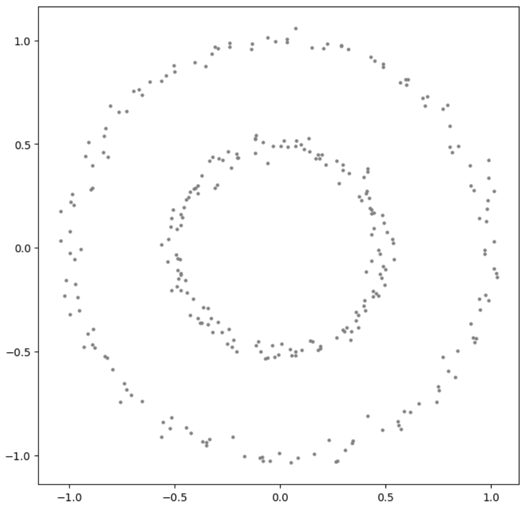


Figure 2.1: Circle data set.

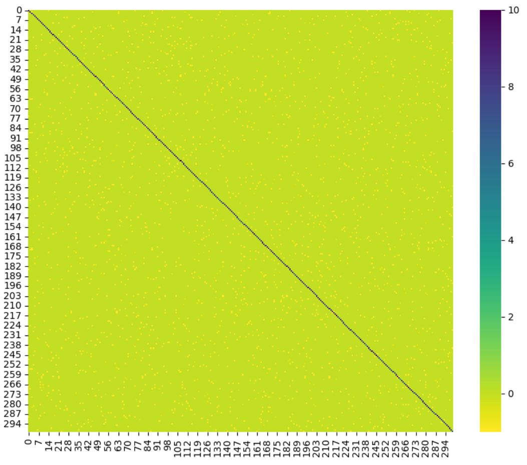


Figure 2.2: Graph Laplacian matrix.

The first step is to construct a graph Laplacian matrix given by $L = D - W$, where $D_{i,i} = \sum_{j=1}^{N} w_{ij}$ and W can be an adjacency matrix or a similarity matrix. In this section we choose the adjacency matrix instead of using kernel, since for a circle data set, the adjacency matrix can perform better. As shown in figure 2.2, the built Laplacian matrix is diagonal dominant.

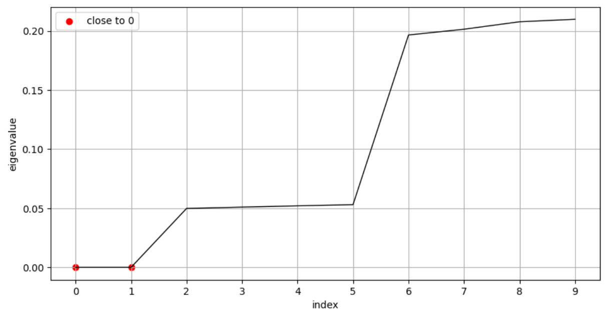


Figure 2.3: 10 smallest eigenvalues.

eigvec_1	eigvec_2
-0.00206103	0.08163636
-0.00206103	0.08163636
-0.08162364	-0.00147381
...	...
-0.00206103	0.08163636
-0.00206103	0.08163636

Table 2.1: Corresponding eigenvectors.

From figure 2.3, you can find that the smallest two eigenvalues of the Laplacian matrix are essentially 0. We can consider their corresponding eigenvectors (table 2.1) and use our own K-means algorithm to cluster the points according to the rows of the matrix whose columns are those eigenvectors. Figure 2.4 shows the final clustering result. As you can see, our algorithm performs very well on partitioning the circle data set.

2.2 Spectral Clustering with numpy.linalg.eigh

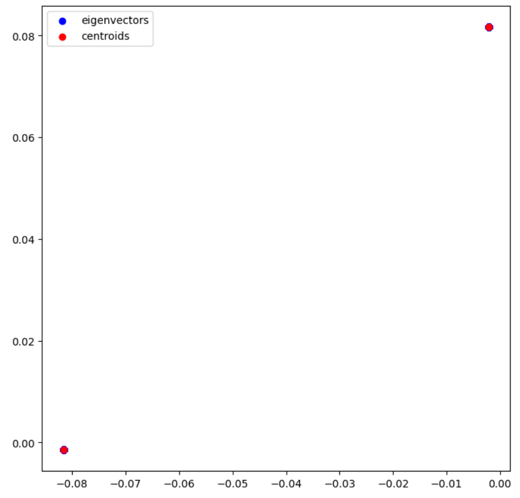
After testing the correctness of our spectral clustering algorithm, now we can believe our algorithm can produce some reasonable results, and the only work left is to modify the Laplacian matrix and eigenvalue solver. Now we use the Gaussian kernel $w_{ij} = \exp(-\gamma\|x_i - x_j\|_2^2)$ to construct the similarity matrix $W \in \mathbb{R}^{N\times N}$, where $\gamma = 1$. Additionally, by deciding on a threshold value ϵ such that two points are considered possibly related if $w_{ij} > \epsilon$ and otherwise not, one can construct an adjacency matrix W where

$$w_{ij} = \begin{cases} 1, & \text{if } w_{ij} > \epsilon, \\ 0, & \text{otherwise.} \end{cases}$$

While we don't apply the threshold technique here, since deciding ϵ is tricky in this data set, and slight change of ϵ can steer the clustering procedure to a totally different result. Therefore, here only the similarity matrix W is used to construct the Laplacian matrix L (figure 2.5):

$$\text{Unnormalized spectral clustering: } L = D - W,$$
$$\text{Normalized spectral clustering: } L = I - D^{-1/2} W D^{-1/2},$$
$$\text{where } D_{i,i} = \sum_{j=1}^{N} w_{ij}.$$

To compute the eigen pairs of the Laplacian matrix, we first try **numpy.linalg.eigh**, a built-in hermitian eigensolver from numpy library in python. Due to the similarity matrix W, the Laplacian matrix is dense, full of nonzero entries, even though most of them are very close to zero. So unfortunately, we miss the chance to take advantage of any kinds of sparse eigensolvers. But what brings a silver lining is that the efficiency of **eigh** is still acceptable with given data. Figure 2.6 illustrates the five smallest eigenvalues of the Laplacian matrix. We can expect that the final clustering result for the unnormalized Laplacian matrix will hardly make any sense since all of its eigenvalues except the smallest one are much bigger than zero. To be consistent with the first part of the project, we pick the smallest four eigenvalues and corresponding eigenvectors to which we apply Euclidean K-Means clustering. The convergence of the clustering is shown in figure 2.9.
The final clustering results of the two cases are very different. From figure 2.7, one can find that in the case of unnormalized spectral clustering, the clustering doesn't work and almost all data points are grouped into one class, as we had expected based on figure 2.6. On the contrary, normalized spectral clustering gives a reasonable classification (figure 2.8). To extract some practical information from the clustering results, further statistical analysis is needed. Since many analysis details have been discussed in the first section, here we will put more emphasis on the algorithm instead of interpretation. In the next section, **numpy.linalg.eigh** is replaced by an eigensolver developed by ourselves.



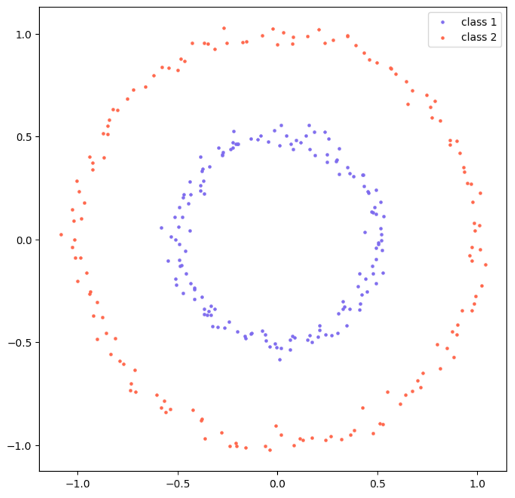


Figure 2.4: The left scatter plot shows that two centroids hit the data set formed by every row of the eigenvector matrix very well. The right side is the final clustering result for the test data.

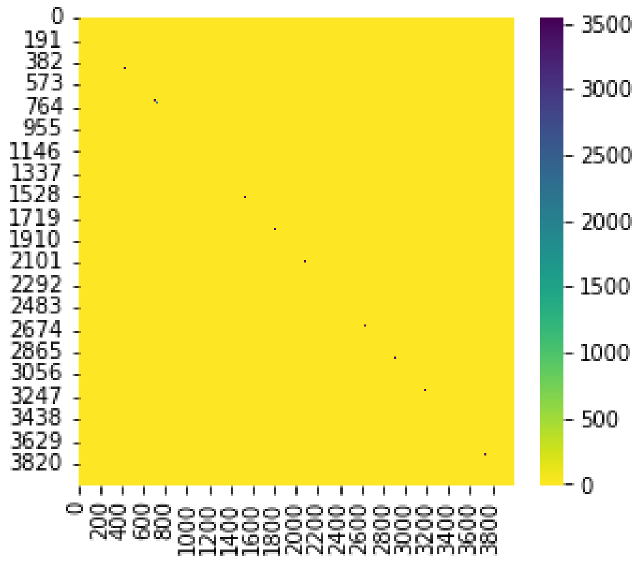
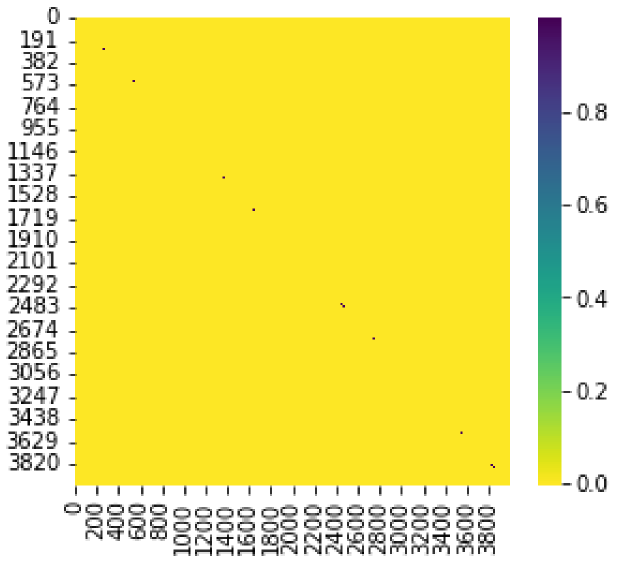
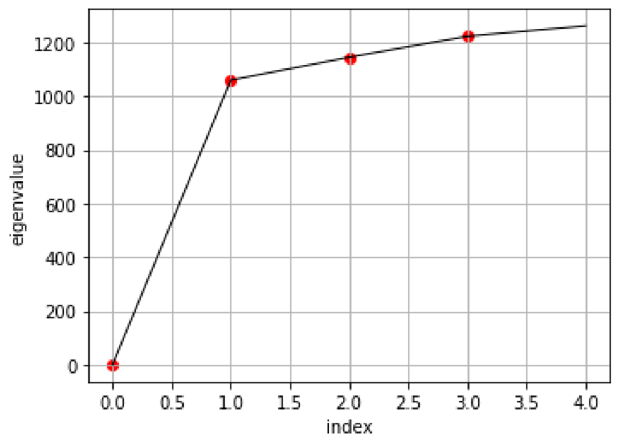



Figure 2.5: The unnormalized (left) and normalized (right) Laplacian matrix for the airline data.



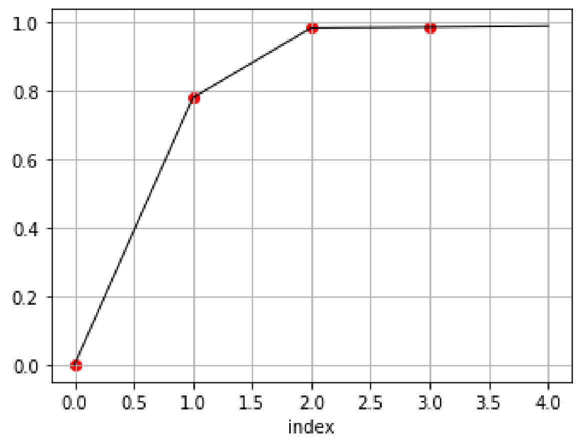


Figure 2.6: 5 smallest eigenvalues of the unnormalized (left) and normalized (right) Laplacian matrix. To be consistent with what we have done in the first part, the smallest four eigenpairs (indicated by red points), or in other words, four controids, are used in the following clustering algorithm.

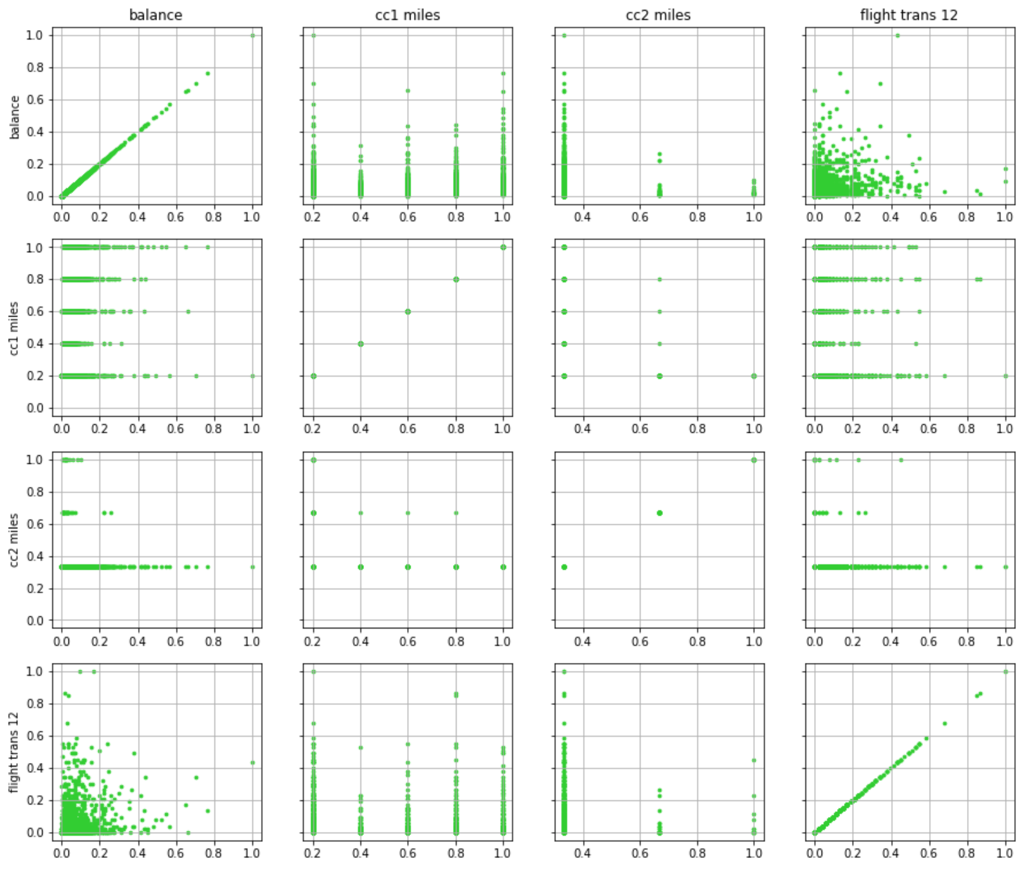


Figure 2.7: Final result of unnormalized spectral clustering on the whole data set.

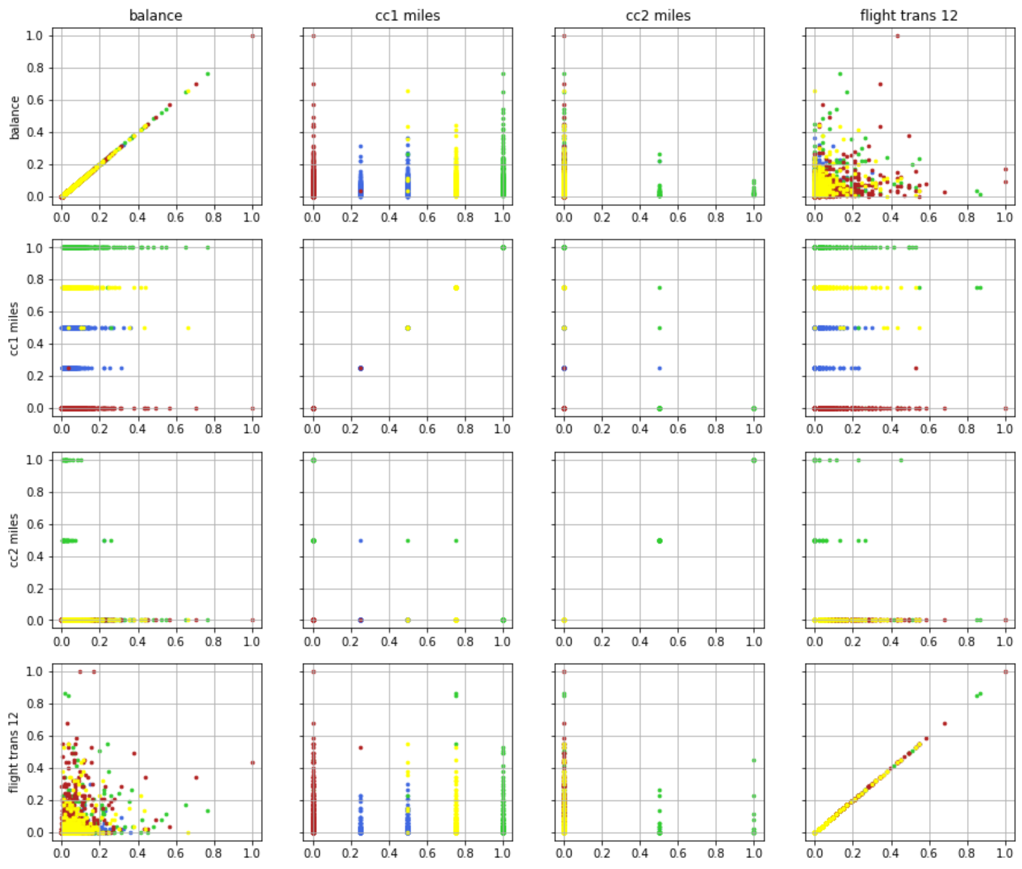


Figure 2.8: Final result of normalized spectral clustering on the whole data set.

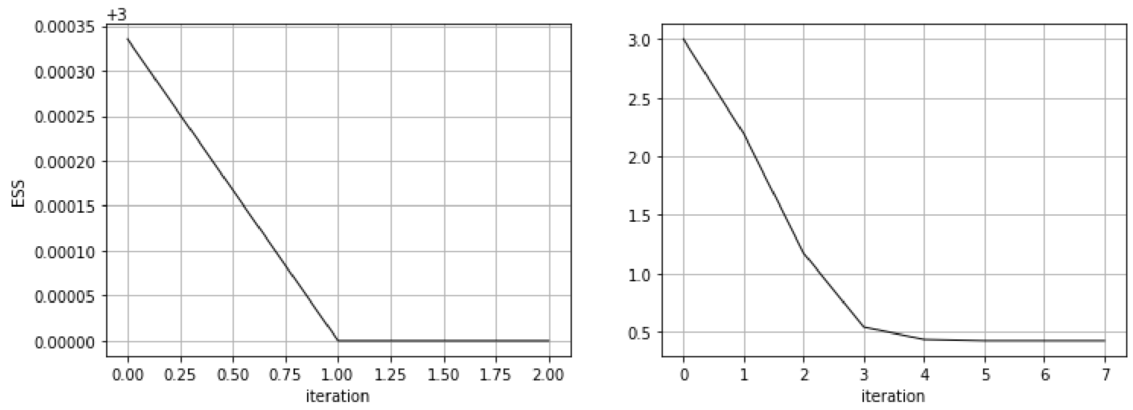


Figure 2.9: Convergence history of the Euclidean K-Means clustering for the unnormalized (left) and normalized (right) case.

2.3 Spectral Clustering with Lanczos Solver

One may ask if there are any more efficient ways to compute or approximate the smallest eigenvalues of a large matrix. The answer is yes. The Lanczos algorithm, an iterative method adapted from power methods, can compute the "dominant" eigenvalues and eigenvectors of an Hermitian matrix very efficiently.

The algorithm of the Lanczos method is given as follows.

```
Start: $r \leftarrow v$, starting vector
$\beta_0 \leftarrow \|r\|_2$
**for** $j = 1, 2, \cdots$, until convergence **do**
    $v_j \leftarrow r/\beta_{j-1}$
    $r \leftarrow Av_j$
    $r \leftarrow r - v_{j-1}\beta_{j-1}$
    $\alpha_j \leftarrow v_j^* r$, where $v_j^*$ is the hermitian transpose of $v_j$
    $r \leftarrow r - v_j\alpha_j$
    re-orthogonalize if necessary
    $\beta_j \leftarrow \|r\|_2$
    $T_j \leftarrow diag([\alpha\ \beta\ \beta], [0, -1, 1])$
    compute eigenvalues of $T_j = S\Theta S^*$
**end for**
compute eigenvectors $X = V_j S$
```

Note that in the last steps of the Lanczos algorithm, we need to compute eigenpairs of the smaller tridiagonal matrix T_j. Given that the matrix now is much smaller, we can apply QR algorithm which is less efficient but more numerically stable. The algorithm of the QR method is given below.

```
$A_0 \leftarrow A,\ k \leftarrow 0$
**for** $k = 1, 2, \cdots$, until convergence **do**
    $A_k \leftarrow Q_k R_k$
    $A_{k+1} \leftarrow R_k Q_k$
    $L_{k+1} + D_{k+1} + U_{k+1} \leftarrow A_{k+1}$
    $k \leftarrow k + 1$
**end for**
eigevalues $\leftarrow diagonal(A)$
```

The diagonal entries of the converged $A_k = Q^\top AQ$ are good approximations to the eigenvalues of A. For the symmetric case, the columns of Q are also good approximations to the eigenvectors of A.

Fortunately, the Laplacian matrix is hermitian positive definite, perfectly suitable for both algorithms. Even more fortunately, since the performance of approximations of the Lanczos method highly depends on the distribution of eigenvalues, it does not necessarily approximate only largest eigenvalues. Through some numerical experiments, we find that the Lanczos algorithm can also approximate a few smallest eigenpairs very well for the normalized Laplacian matrix (figure 2.10). This good property can save us from spending enormous computational resource on inverting L with tiny shift.

The convergence history of the smallest eigenvalue computed by the Lanczos algorithm can be seen in figure 2.11. And table 2.2 compares the smallest eigenvalues of the normalized L computed by **numpy.linalg.eigh** and our own Lanczos eigensolver. It is impressive that without shifting and inverting L, the Lanczos eigensolver takes less than one second, much faster than **numpy.linalg.eigh** which takes around 15 seconds to compute eigen pairs. One reason is that the number of iteration of the Lanczos algorithm is very small, as only a few smallest eigenvalues are needed and it's unnecessary to iterate too much to compute other eigenvalues.

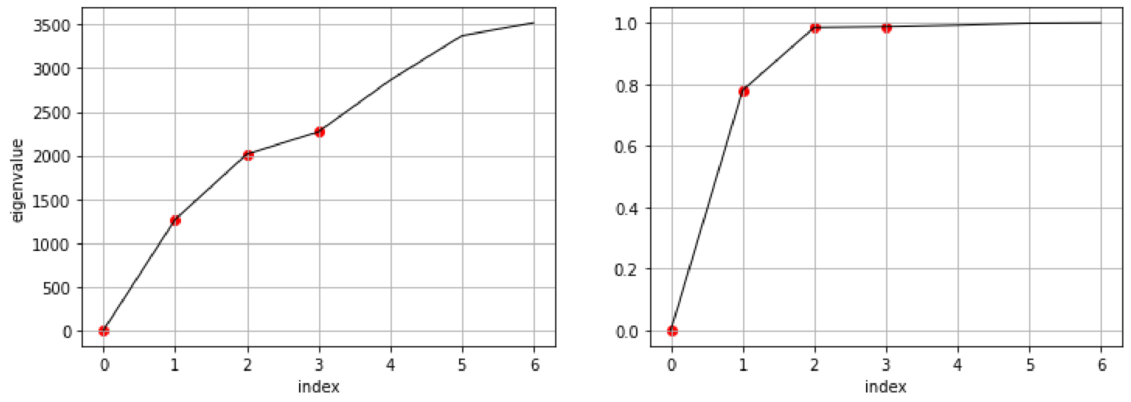


Figure 2.10: 5 smallest eigenvalues of the unnormalized (left) and normalized (right) Laplacian matrix computed by combination of the Lanczos and QR algorithm. For the normalized case, the algorithm can approximate the smallest eigenvalues very well without inverting L.

The final result of normalized spectral clustering is illustrated in figure 2.12. The result of unnormalized spectral clustering is not shown here, since it is the same as the result of the previous section.

Table 2.2: 5 smallest eigenvalues.

numpy.linalg.eigh	Lanczos method
4.71061630e-16	4.91619531e-17
7.80211891e-01	7.80211891e-01
9.84511036e-01	9.84512981e-01
9.87029008e-01	9.87058068e-01
9.90321101e-01	9.96170222e-01

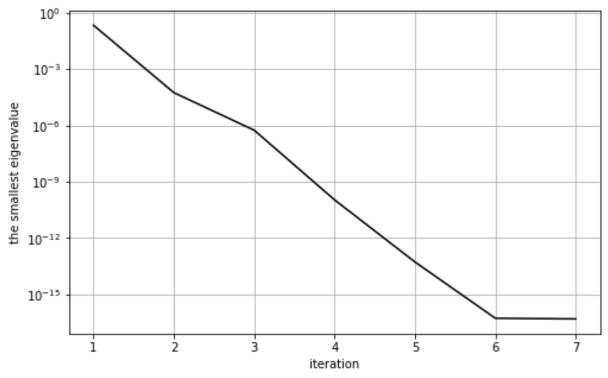


Figure 2.11: Convergence history for the smallest eigenvalue.

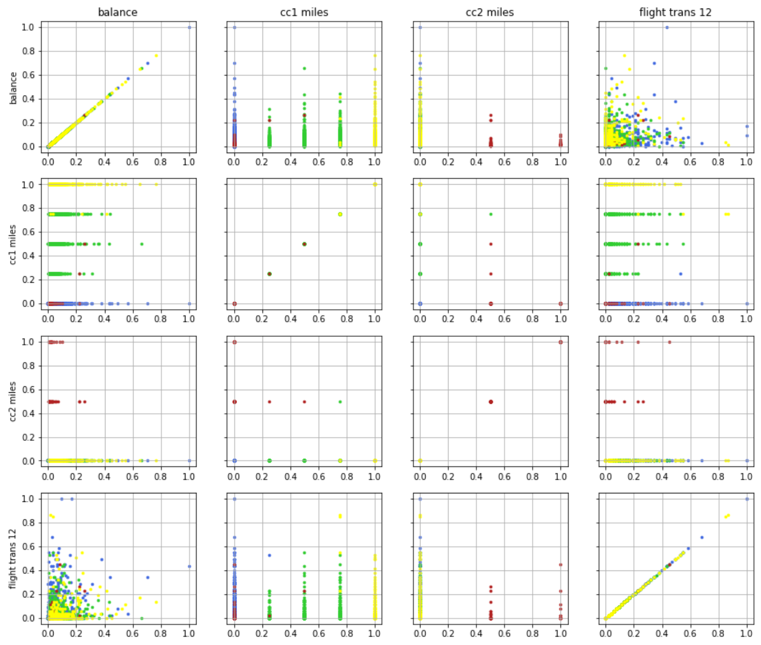


Figure 2.12: Final result of normalized spectral clustering on the whole data set.