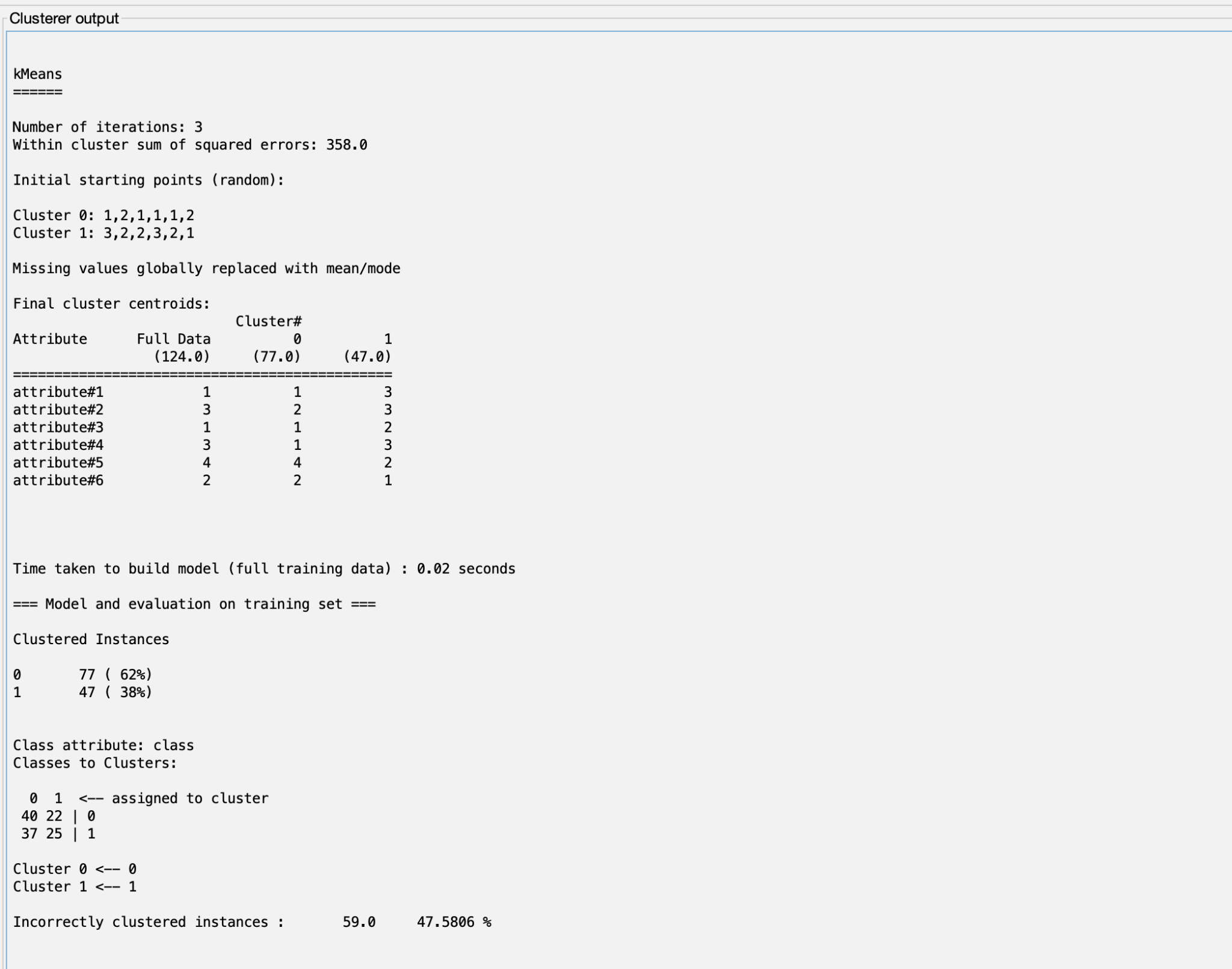
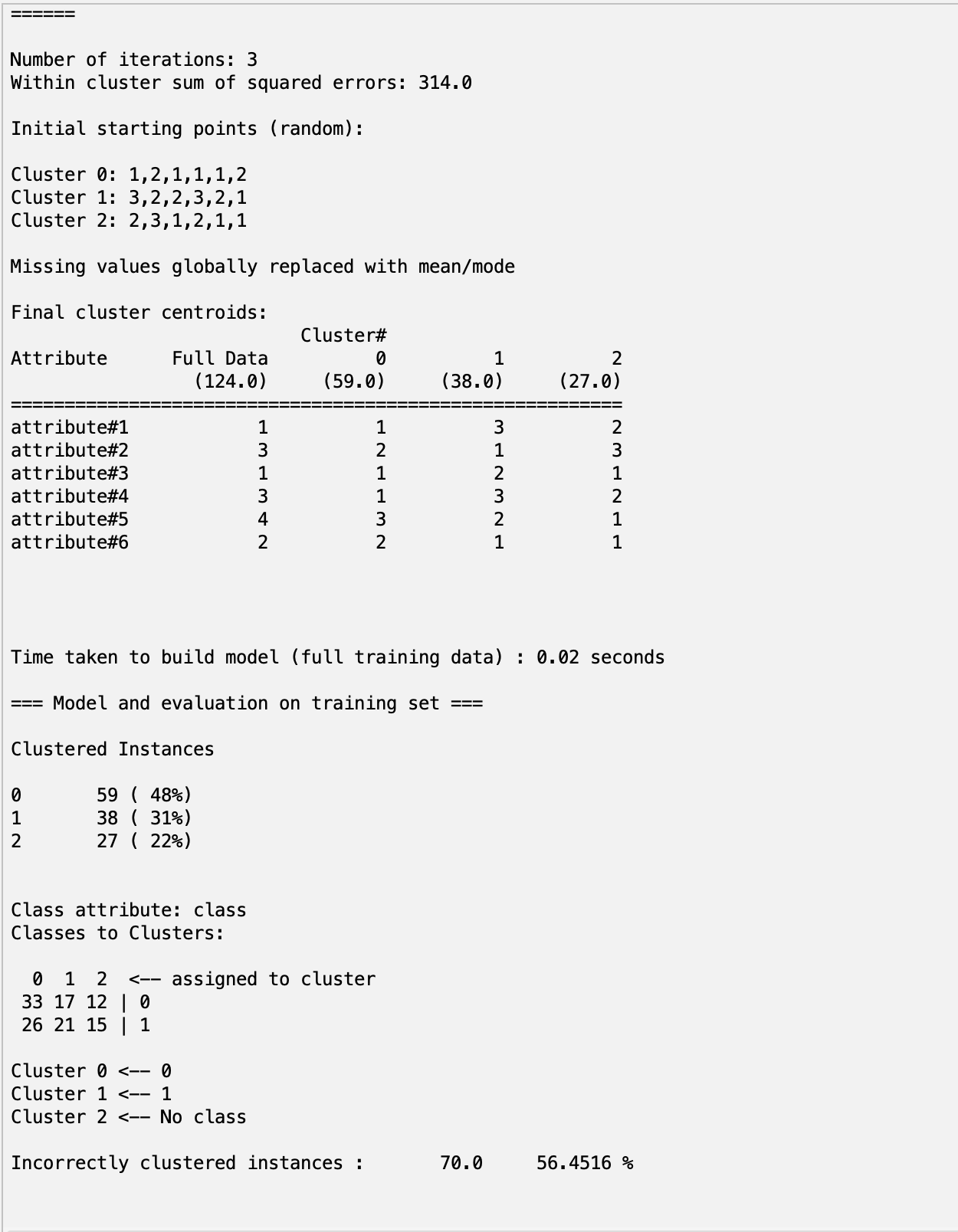
**Clustering Attempts**

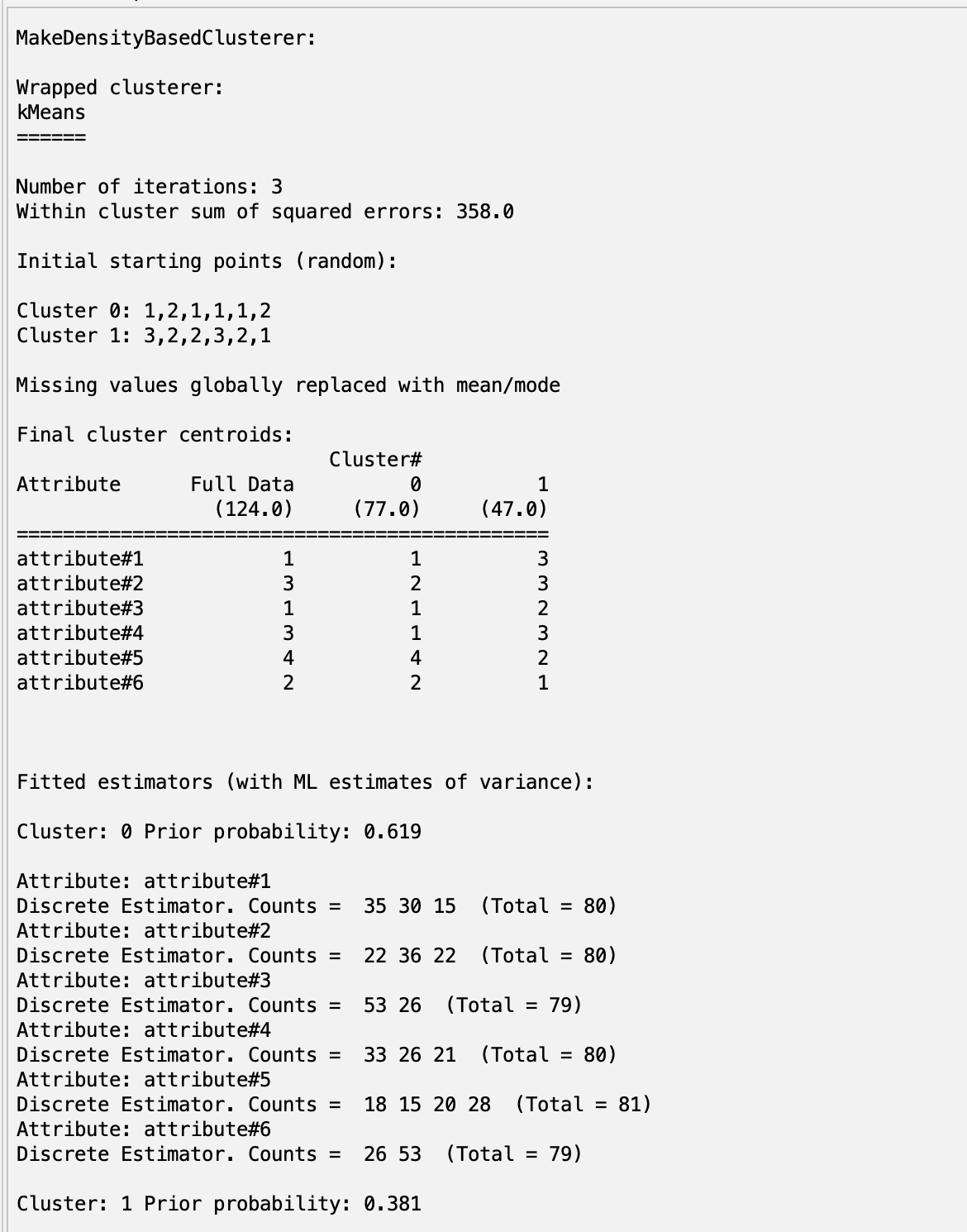
*SimpleKMeans - two clusters*

We started using the SimpleKMeans algorithm with two clusters as we know there are two classes in the data. A little bit less than 50% of the data was misclassified using this setup, which is hardly better than taking a random guess.

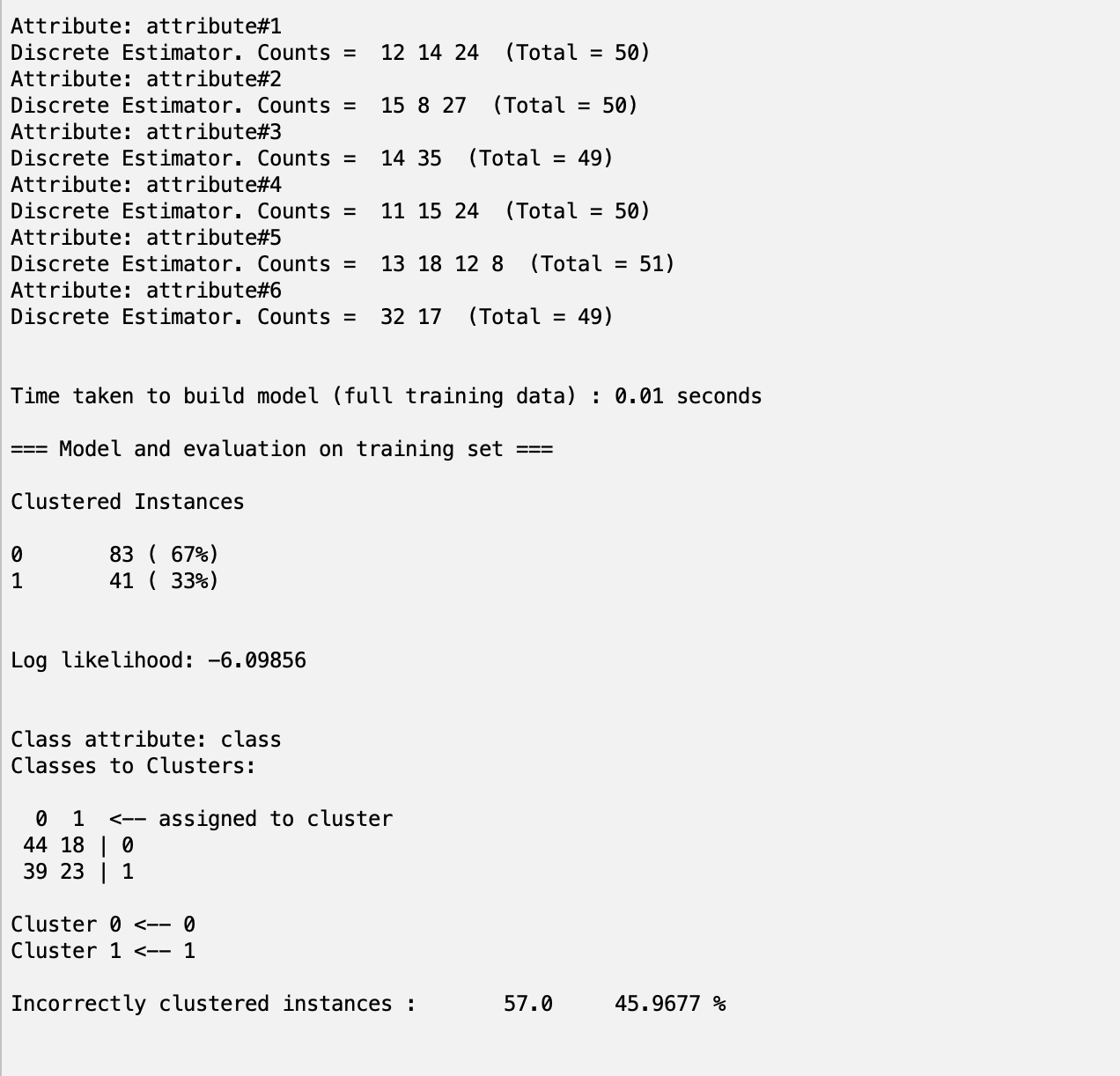


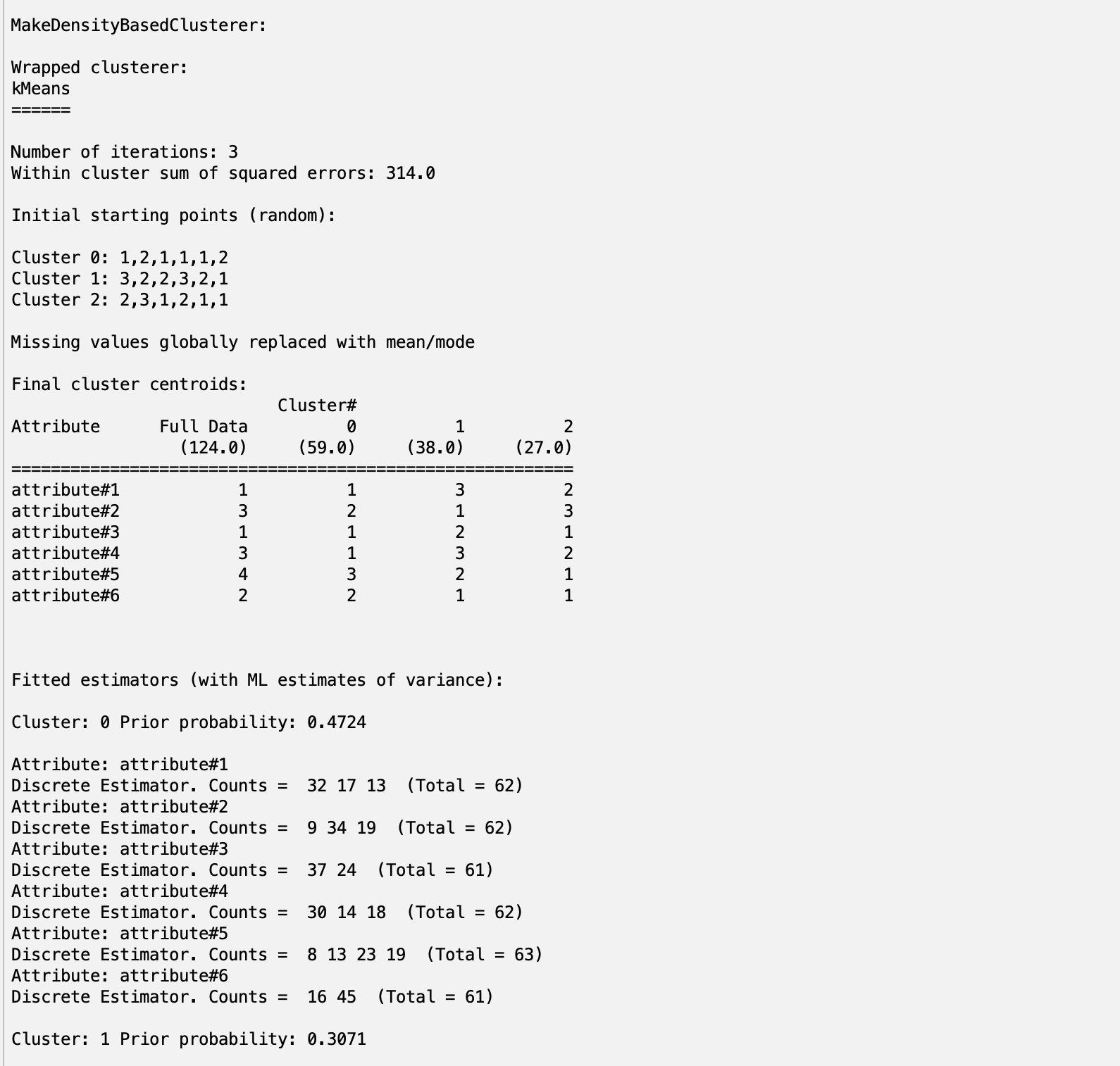
*SimpleKMeans - three clusters*

Increasing the number of clusters by one results in even worse results, incorrectly classifying over 50% of the data. It makes sense that having more clusters than classes will give poor results when comparing to the true classes, since we know that there are only supposed to be two.

*Density-Based - two clusters*

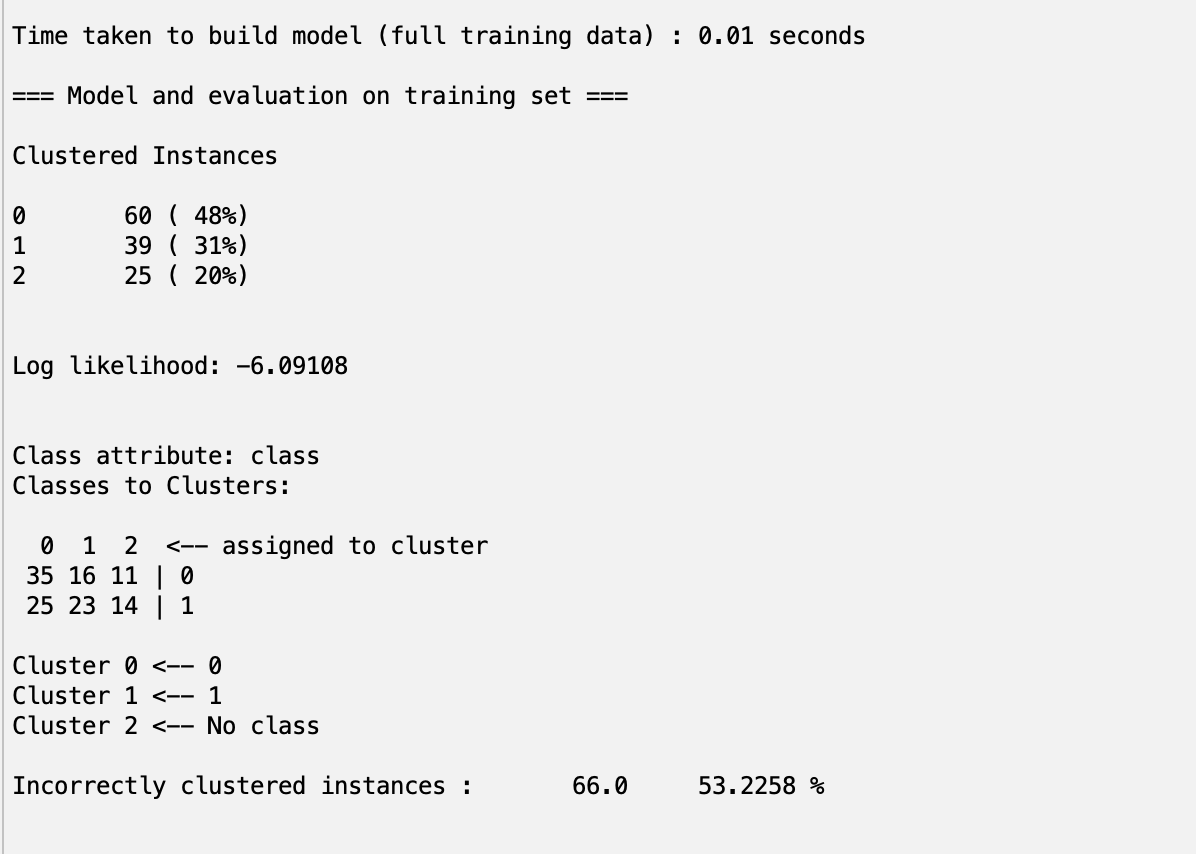
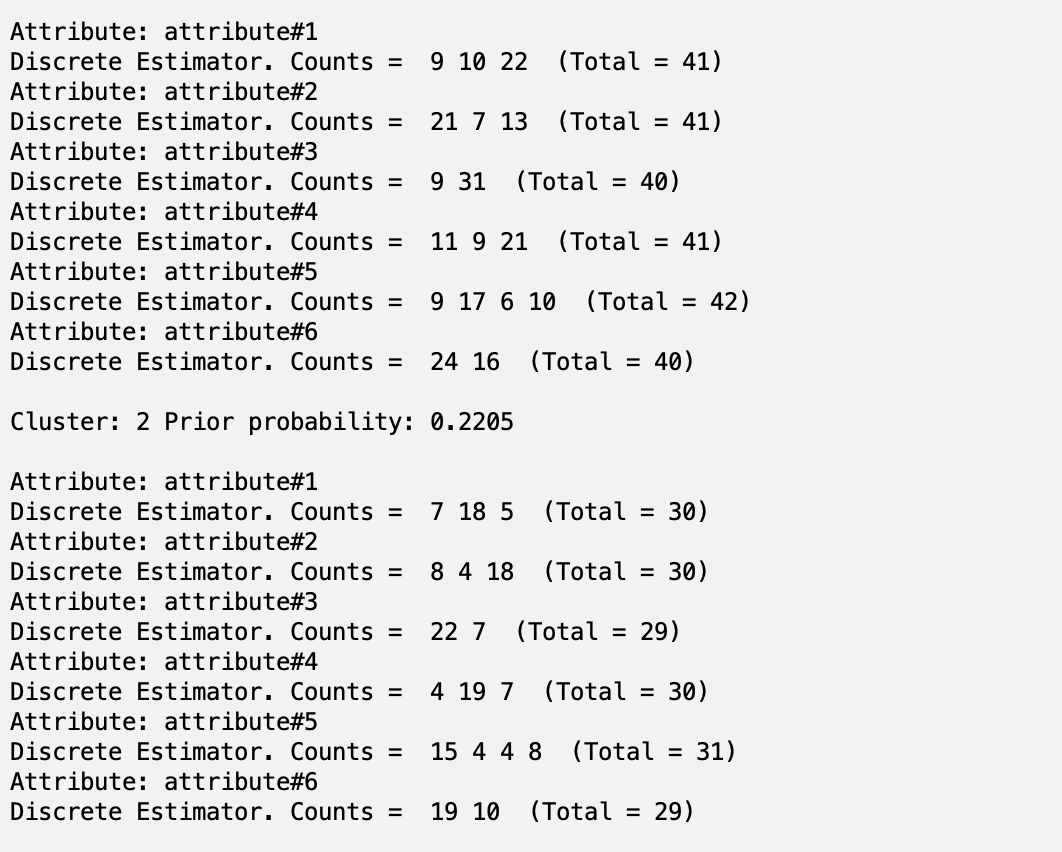
The density-based algorithm also produces poor results when using two clusters, only being slightly better than a random guess.





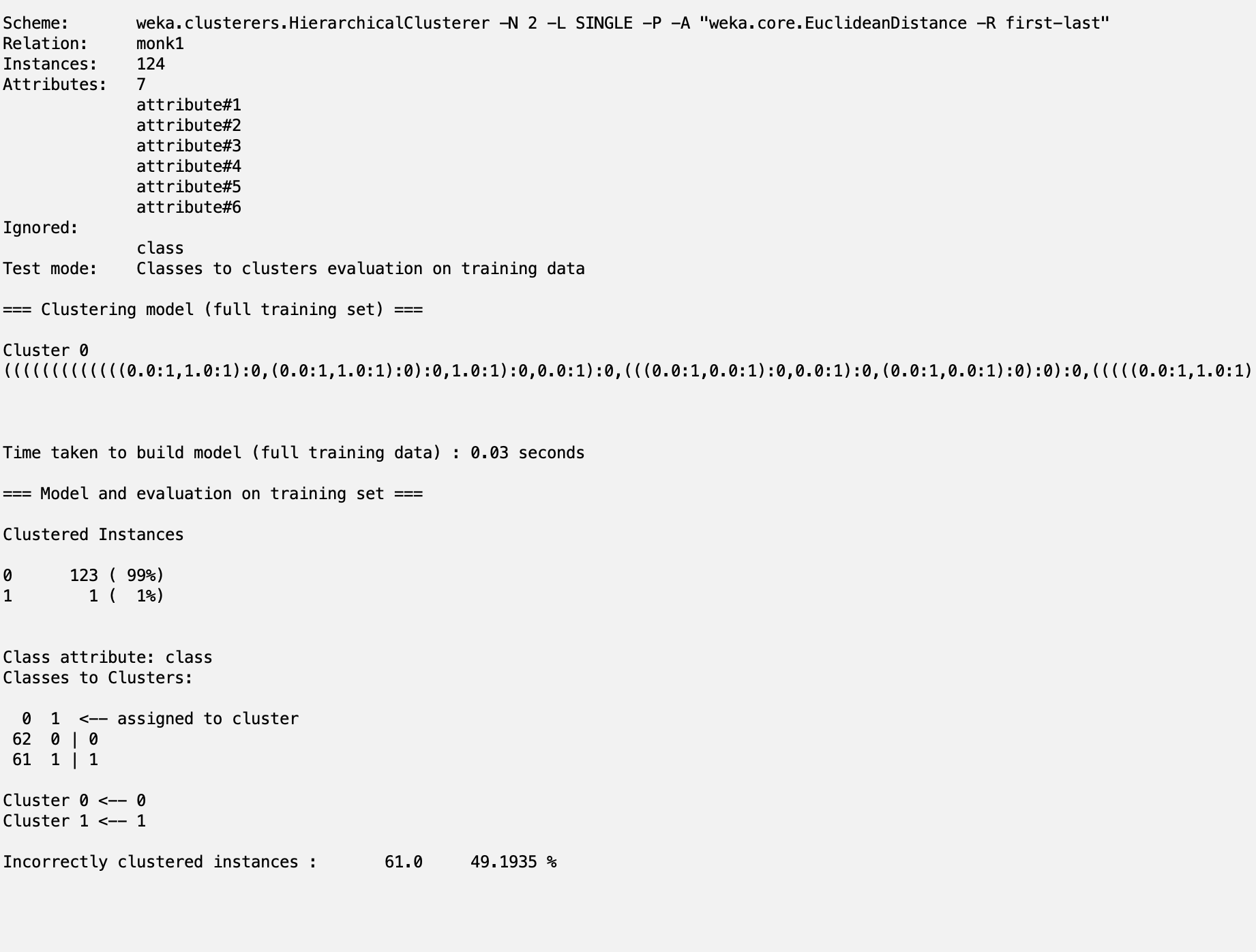
*Density-Based - three clusters*

Again, increasing the number of clusters for the density-based algorithm further decreases the performance of the clustering.



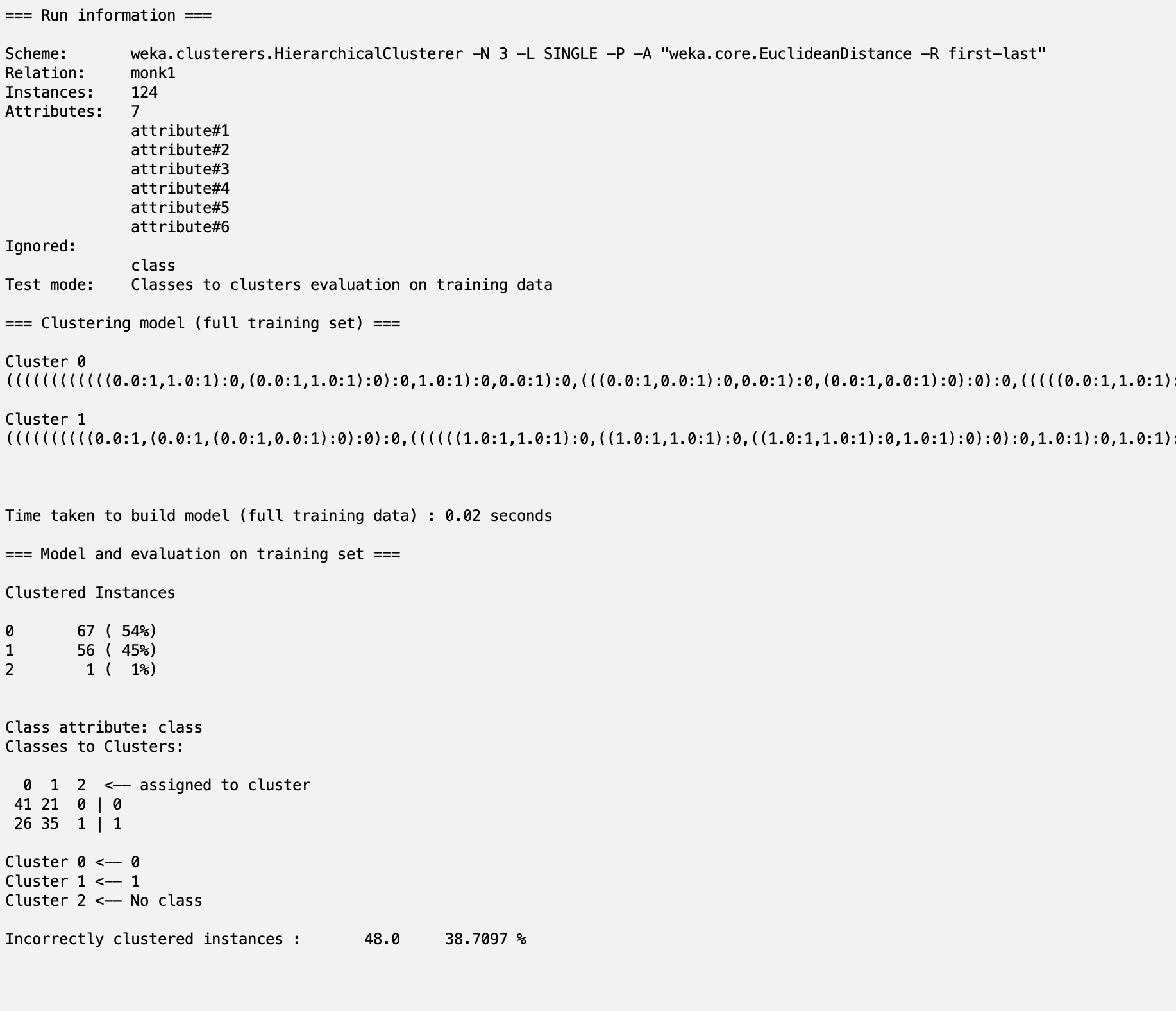
*Hierarchical Clustering - two clusters*

Using two clusters on the hierarchical clustering algorithm gives the same accuracy as a random guess.



*Hierarchical Clustering - three clusters*

Relatively, this performs surprisingly well with only 38.7% of the data assigned to the wrong cluster. Still, it’s not very good.



**Association**

We wrote a small Python script to evaluate our rules, avoiding redundancy to find some different options, and wound up with the four following rules as the best:

1. Attribute 5 = 1 → class = 1
2. Attribute 3 = 3 and attribute 2 = 3 → class = 1
3. Attribute 1 = 2 and attribute 2 = 2 → class = 1
4. Attribute 1 = 3 and attribute 2 = 3 and attribute 6 = 2 → class = 1

The Python script:

import pandas as pd

from scipy.io import arff

data = arff.loadarff(r"C:\Users\marij\Desktop\monk1.arff")[0]

df = pd.DataFrame(data)

df = df.map(lambda x: int(x))

df.loc[df["attribute#5"] == 1, "assoc"] = 1

df.loc[(df["attribute#3"] == 3) & (df["attribute#2"] == 3), "assoc"] = 1

df.loc[(df["attribute#1"] == 2) & (df["attribute#2"] == 2), "assoc"] = 1

df.loc[(df["attribute#1"] == 3) & (df["attribute#2"] == 3) & (df["attribute#6"] == 2), "assoc"] = 1

df.loc[df["assoc"].isna(), "assoc"] = 0

# df.loc[df["attribute#5"] == 1, "assoc"] = 1

# df.loc[(df["attribute#3"] == 3) & (df["attribute#2"] == 3), "assoc"] = 1

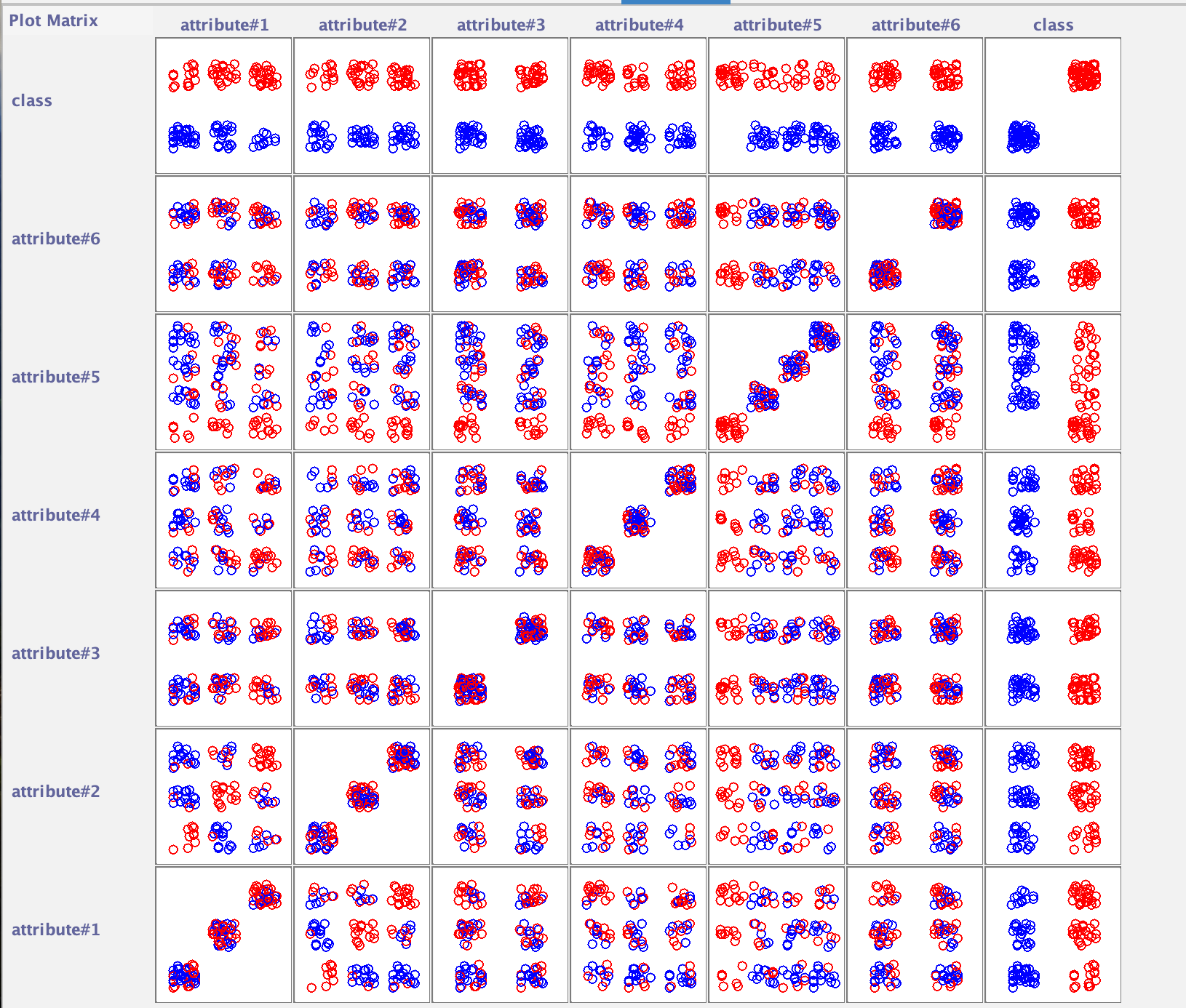
# df.loc[(df["attribute#1"] == 2) & (df["attribute#2"] == 2), "assoc"] = 1

# df.loc[(df["attribute#1"] == 3) & (df["attribute#2"] == 3) & (df["attribute#3"] == 1), "assoc"] == 1

# df.loc[df["assoc"].isna(), "assoc"] = 0

print(f"{len(df.loc[df["class"] != df["assoc"]].index)} misclassified")

**Visualization**



**The Question**

***Why can the clustering algorithms not find a clustering that matches the class division in the database?***

As can be seen from the visual, the data is mostly very mixed between all the attributes. The data is discrete, and there are no clear clusters that show up across all attributes. Association allows us to look at a subset of the attributes at a time and draw conclusions from this. For example, when attribute five has value one this leads to class one 100% of the time. When doing clustering, the clusters have to have centroids in all the attributes. This is perhaps also why the hierarchical clustering algorithm performed slightly better, as it does not rely on these centroids.

***Would you say that the clustering algorithms fail or perform poorly for the monk1 dataset?***

The clustering algorithms perform very poorly on this dataset, often misassigning around 50% of the data points. With accuracies like these on a binary-classed dataset, we may as well randomly assign classes. In this case, association would be a better choice for the reasons explained above.