K-Means and Gaussian Mixture Model (GMM)

Anushka Hada

11-25-2025

1 Introduction

For this project, unsupervised learning algorithms such as **K-Means** and **Gaussian Mixture Model (GMM)** were applied on the **hw4 dataset**. This dataset contains only two features, **length** and **width** of unidentified species. Since the species labels were not provided, the goal of the clustering algorithms was to group the data into different clusters that might represent different species. The correctness of the algorithm could only be visually determined or compared with Scikit-Learn's implementations of K-Means and GMM.

The following figure depicts the dataset prior to applying any clustering algorithm:

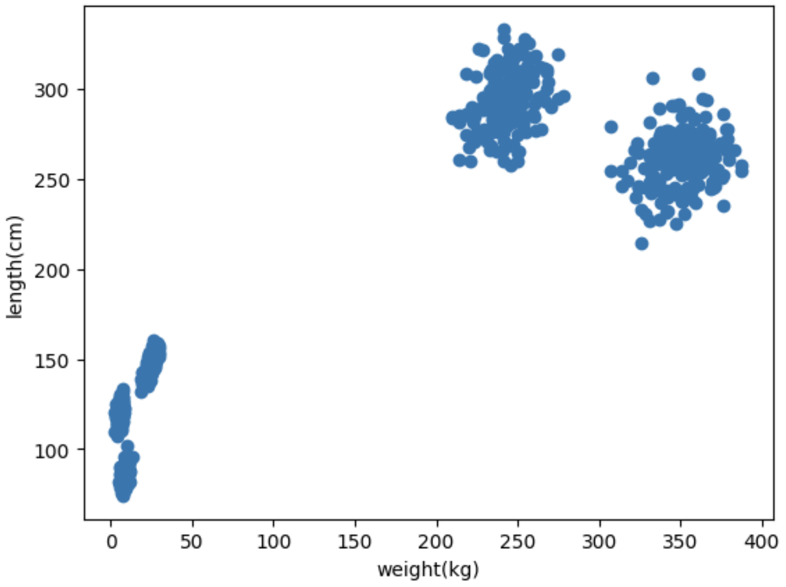

Figure 1: HW 4 dataset before clustering

2 K-means algorithm

The first model, K-means, was implemented by iteratively assigning points to the nearest centroid and updating those centroids based on the cluster means. The classes parameters include the number of clusters and the max iterations. At the start of training, the algorithm randomly selects k distinct samples from the dataset as starting centroid. From there the class splits into two main steps. First, each data point is assigned to the nearest centroid using the euclidean distance. The cluster label is the index of the centroid with the minimum distance. After this, for each cluster the mean of all points is computed. If a cluster has no points, meaning it is empty, the centroid is updated using a random data point. After each update, if the centroids show no significant change, the algorithm stops training.

3 GMM algorithm

The second model, GMM, applies the Expectation–Maximization (EM) algorithm to fit a Gaussian Mixture Model with a certain number of components. Unlike, K-means, GMM shows each cluster as a multivariate Gaussian distribution and assigns points based on probabilities. The model takes in the number of components and initializes covariance, weight, and means randomly. During training, the algorithm finds the **responsibility** values, which represent the probability that each data point belongs to a component. The probabilities are computed using the Gaussian probability density function weighted by the component weights and are normalized across components. After this E-step, the model enters the M-step and updates the parameters of each distribution. The EM loop goes back and forth between the two steps, until the model converges. Once trained, the model finds the best groupings by selecting the component with the highest responsibility for each data point.

4 Results

4.0.1 K-means Algorithm vs. Scikit-Learn K-Means

The two figures below show the clustering results generated by the implemented K-Means and Scikit-Learn K-Means algorithms, using maximum iterations ranging from 5 to 200 and cluster counts varying from 2 to 4.

The results show several noticeable differences. At k = 2 both had the same results. It was with k = 3 and k = 4 where the models disagreed on the clustering pattern. The implemented K-means consistently separated the bottom left group of data points into two or three separate clusters at k = 4. The Scikit-Learn K-Means kept the same clustering pattern for k = 4 with the bottom left only being separated into two clusters, regardless of the number of iterations. This may indicate a certain level of uncertainty in the implemented K-means algorithm as it had noticeable difference in clustering each iteration. Though the two models reached separate conclusions for k = 3 and k = 4, Scikit-Learn K-means had a more consistent clustering pattern throughout the experiment.

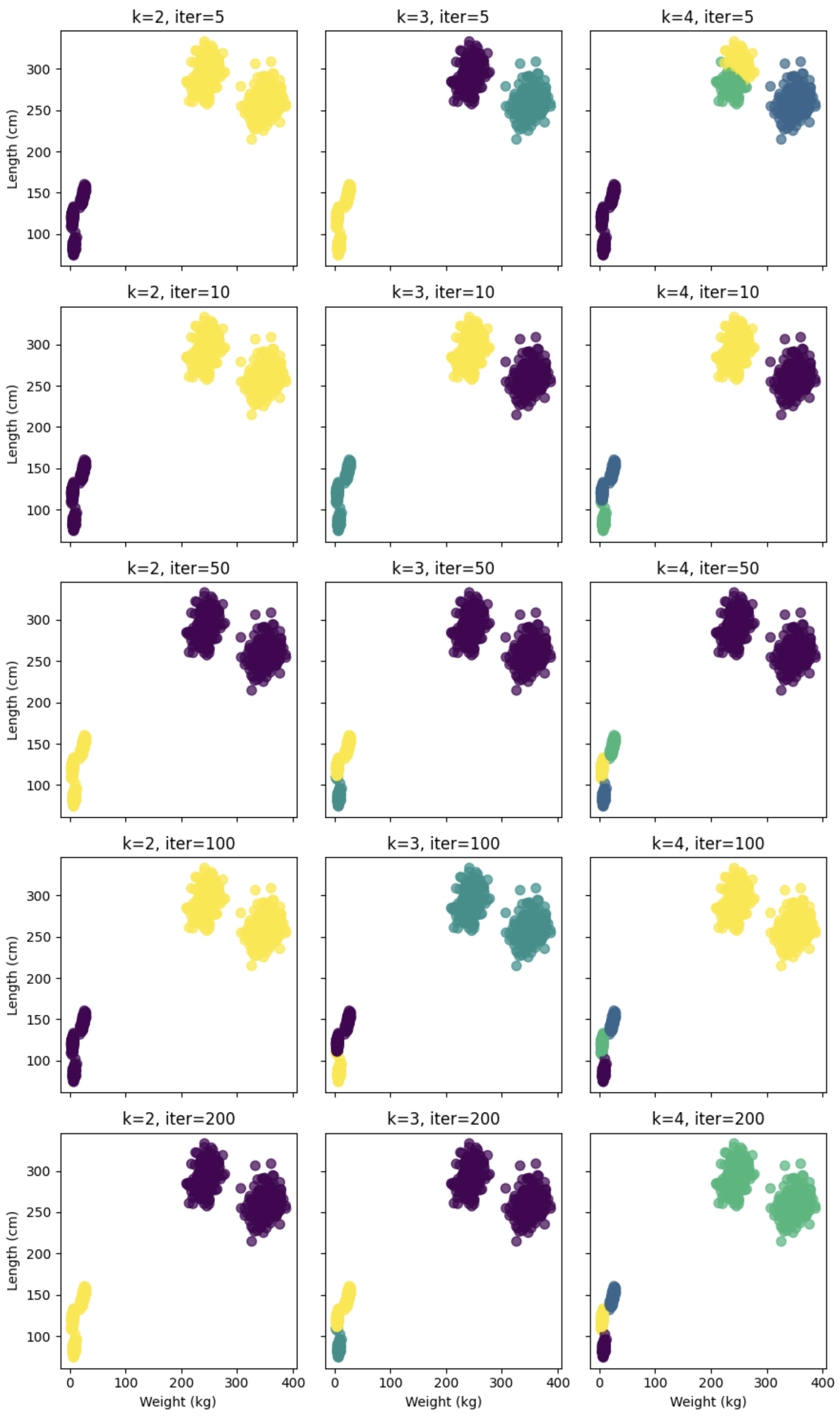

Figure 2: K-Means clustering results for varying values of k and iteration limits using the custom implementation.

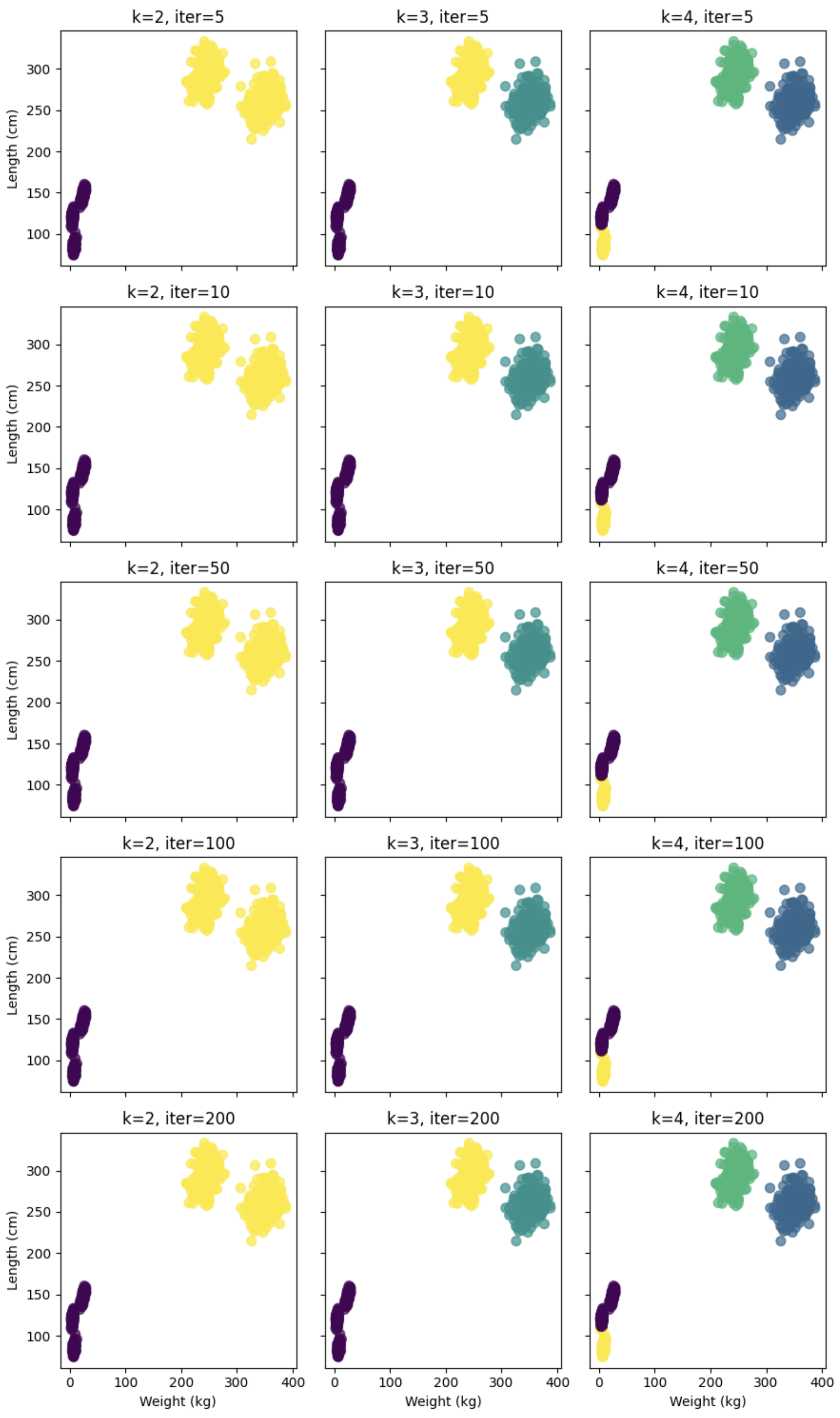

Figure 3: Scikit-Learn K-Means clustering results for varying values of k and iteration limits.

4.0.2 GMM Algorithm vs. Scikit-Learn GMM

The two figures below show the clustering results generated by the implemented GMM algorithm and the Scikit-Learn GMM model, using maximum iterations ranging from 5 to 200 and component counts varying from 2 to 4.

For the implemented GMM model, it had widely different clustering patterns each iteration and component amounts. This shows a high level of uncertainty where the model is unable to find the best fit cluster for each group of data points. This is further validated by the fact that at k = 4 and iter = 50, the clustering is unreasonably placed. The pattern of the purple cluster is completely different from all other parameter combinations and does not visually indicate proper grouping. Another example of unreasonable grouping would be at iter = 5 and iter = 10 for k = 4. By comparison, Scikit-Learn GMM has the same clustering pattern regardless of iteration, showing consistency in its training. Furthermore, compared with the implemented K-means, the implemented GMM performed significantly worse then all other models overall, showing a need for further optimization.

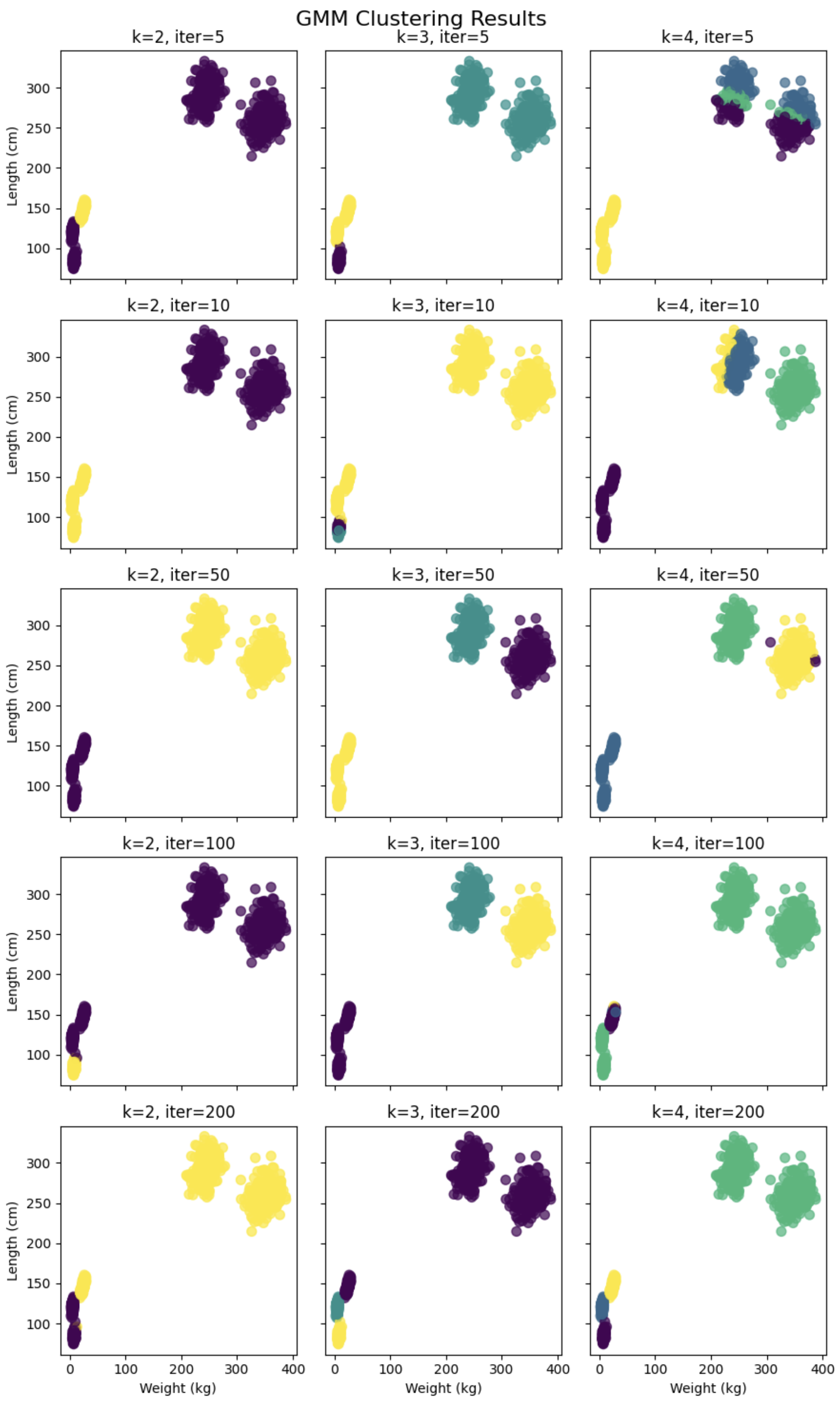

Figure 4: Custom GMM clustering results for varying component numbers and iteration limits.

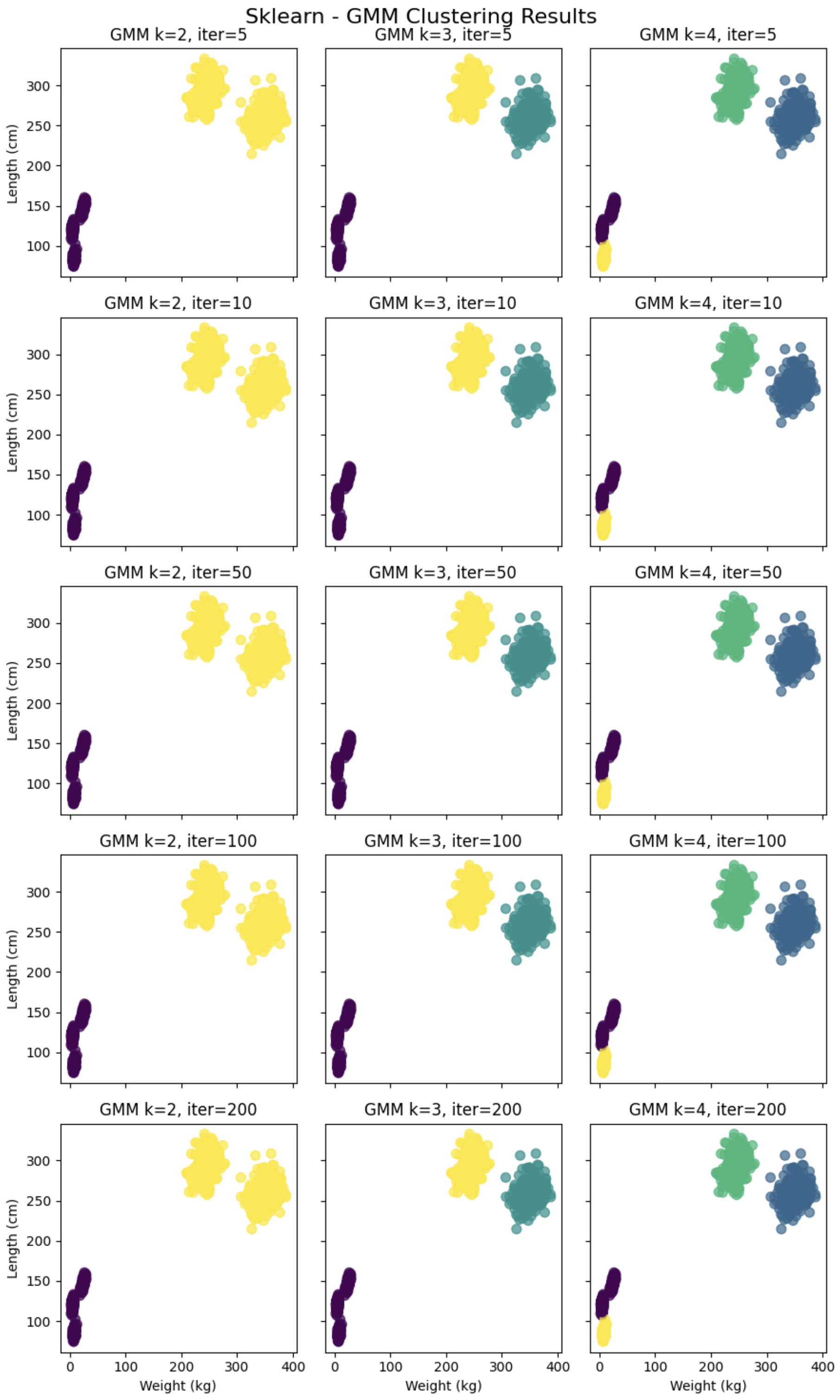

Figure 5: Scikit-Learn GMM clustering results for varying component numbers and iteration limits.

5 Conclusion

In the end, several conclusions can be drawn from the four models. Scikit-Learn models had consistent outcomes, with barely some differences between iterations. The implemented K-means performed similarly to Scikit-Learn K-means, with some differences at k = 4 and k = 3. The implemented GMM performed the worse overall and barely resembled the Scikit-Learn GMM. In the future, though it is difficult to verify how accurate a clustering algorithm is with unlabeled data, the results of these models reveals a lot about the dataset. They point out clustering patterns that may be different from those chosen visually using the human eye.