K-Means Clustering Implementation and Evaluation

CS470 Homework 3

November 4, 2025

1 Datasets

1.1 Iris Dataset

For this assignment, I used the classic Iris dataset, which includes 150 flower samples described by 4 measurements: sepal length, sepal width, petal length, and petal width. The original dataset has 3 distinct iris species with 50 samples each, but I removed the class labels to test how well k-means could discover these natural groupings on its own.

1.2 Wine Dataset

I also chose the Wine dataset from the UCI repository, which contains 178 wine samples characterized by 13 chemical properties like alcohol content, malic acid, ash, magnesium, and various phenols. This dataset originally has 3 wine cultivars, and I stripped away the labels to see if k-means could identify them based purely on their chemical signatures.

2 Implementation and Experimental Setup

I implemented the k-means algorithm in Python using NumPy for efficient numerical operations. My implementation uses Euclidean distance to measure similarity between data points, which is the standard choice for this algorithm. Before running k-means, I applied z-score normalization to all features so that attributes with larger scales wouldn't dominate the distance calculations. To make my results reproducible, I initialized the centroids by randomly selecting data points with a fixed random seed. The algorithm stops when the centroids barely move (changes less than 10^{-4}) or after 100 iterations, whichever comes first.

To evaluate the clustering quality, I used two complementary metrics. The Sum of Squared Errors (SSE) measures how tightly points cluster around their centroids by summing up the squared distances. The Silhouette Coefficient is more sophisticated—it ranges from -1 to 1 and tells us whether points are well-matched to their own cluster compared to neighboring clusters. Higher silhouette scores mean better-defined, more separated clusters.

I ran experiments with k ranging from 2 to 10 on both datasets to see how the number of clusters affects performance.

3 Results

Tables 1 and 2 show the SSE and Silhouette scores I obtained for different values of k on both datasets.

Table 1: Results on Iris dataset

k	SSE	Silhouette
2	223.73	0.5799
3	141.15	0.4617
4	115.68	0.4135
5	104.69	0.3914
6	80.76	0.3409
7	71.92	0.3500
8	67.21	0.3382
9	59.91	0.3563
10	57.55	0.3251

Table 2: Results on Wine dataset

k	SSE	Silhouette
2	1659.01	0.2683
3	1277.93	0.2849
4	1184.37	0.2657
5	1132.91	0.1810
6	1089.31	0.1800
7	1032.93	0.1709
8	1000.21	0.1296
9	946.43	0.1332
10	907.85	0.1261

4 Discussion and Conclusions

Looking at the Iris results, I found something interesting: the highest silhouette score actually occurred at $k = 2$ (0.5799), but when I examined the SSE elbow curve, there's a noticeable bend at $k = 3$. Since we know the Iris dataset has 3 species, $k = 3$ makes more sense. The decent silhouette score of 0.4617 at $k = 3$ suggests the clusters are reasonably well-defined, though there's definitely some overlap between species in the feature space.

The Wine dataset told a different story. Here, the silhouette coefficient peaked right at $k = 3$ (0.2849), which matches the actual number of wine cultivars in the dataset. However, I noticed that Wine's silhouette scores were consistently lower than Iris across all k values, suggesting the wine cultivars aren't as cleanly separated based on their chemical properties. The SSE curve decreased more gradually without a sharp elbow, making it harder to pick an optimal k.

This assignment taught me several valuable lessons. First, normalization really matters without z-score normalization, features with larger ranges would have dominated the clustering, leading to poor results. Second, I learned why it's crucial to use multiple evaluation metrics. SSE always decreases as you add more clusters, so relying on it alone would mislead you into thinking more clusters are always better. The silhouette coefficient gave me a much better sense of actual cluster quality.

I also discovered that not all datasets cluster equally well. Iris has much clearer natural groupings than Wine, which shows that kmeans works best when your data actually has well-separated clusters. During implementation, I had to handle tricky edge cases like what to do when a cluster becomes empty during iteration. I ended up reinitializing it with a random data point. Finally, while I presented numerical tables here, I found that visualizing the results with elbow curves and silhouette plots made patterns much easier to spot.

Overall, I successfully implemented k-means and got it working on both datasets. The algorithm correctly identified the true number of clusters in each case, and using a fixed random seed ensured my results were reproducible across multiple runs.