Application of Clustering Models and Dimensionality Reduction Models

SDM274 2023 Fall Final Project

1st JingJun Xu
Department of Electronic and Electrical Engineering
Southern University of Science and Technology
Shenzhen, China
12210357@mail.sustech.edu.cn

***Abstract*—This project embarks on a detailed exploration of unsupervised learning, focusing on clustering techniques and image reconstruction. Employing Python and NumPy, we develop and analyze the KMeans and SoftKMeans algorithms for clustering, applying them to a wheat seed dataset. The project further innovates these algorithms by incorporating non-local split-and-merge moves and different initialized methods. Additionally, we delve into selecting the principal color of an image and image reconstruction after dimensionality reduction through the creation and application of PCA, LinearAutoEncoder and SoftKMeans classes. Our study is comprehensive, covering the implementation of these models, their application on real-world data, and a thorough analysis of their performance, especially concerning various hyperparameters. The insights gained from this project are aimed at deepening the understanding of these fundamental machine-learning techniques, their practical applications, and their potential for future advancements in the field.**

***Index Terms*—KMeans, Soft KMeans, Non-local Split-and-Merge, PCA, Linear AutoEncoder**

I. Introduction

In this project, we delve into the realms of unsupervised learning through the lens of clustering and dimensionality reduction. By implementing models like KMeans, Soft KMeans, PCA, and Linear AutoEncoder from scratch using Python and Numpy, this project aims to face the challenges of Clustering and Image Reconstruction. By doing so, it provides a hands-on approach to understanding the mechanics and nuances of these algorithms.

The central aim of this project is to develop and refine several models including KMeans, SoftKMeans, PCA, and LinearAutoEncoder. We'll be crafting these models as distinct classes in Python, using Numpy for computational efficiency. The KMeans model is housed in kmeans.py, SoftKMeans in soft_kmeans.py, PCA in pca.py, and LinearAutoEncoder in linear_autoencoder.py. Our experimentation will span various model parameters, such as the number of clusters and principal components, to gauge the models' adaptability and performance under diverse configurations.

To enhance the clustering process, this project incorporates non-local split-and-merge algorithms into the KMeans and SoftKMeans models. The enhanced versions, KMeans with non-local split-and-merge, are contained in adjusted_kmeans.py, while SoftKMeans with the same enhancements is in adjusted_soft_kmeans.py. Furthermore, these models also feature various methods for initializing cluster centers, adding to their robustness and flexibility in clustering diverse datasets.

Our project aims to build and assess models like KMeans and SoftKMeans under various scenarios. We focus on evaluating their effectiveness through diverse analytical approaches, with experiments detailed in clustering.ipynb, principal_color.ipynb and image_reconstruction.ipynb. Additionally, utilities.py contains essential functions supporting this analysis.

The report is structured to provide a comprehensive view of the project, starting with problem formulation, followed by detailed descriptions of the methods and algorithms used. Subsequent sections will present experimental results, including visual plots and tables, and conclude with an analysis of these findings. The final section will discuss potential future enhancements and applications of this work.

By the end of this project, we expect to have a deeper understanding of KMeans, SoftKMeans, PCA, and Linear AutoEncoder algorithms, along with insights into their practical application and performance in clustering, selecting image principal colors, and image reconstruction tasks.

II. Problem formulation

In this project, we address two primary challenges: The Clustering Problem and the Image Reconstruction Problem.

Writing the model: For the Clustering Problem, we intend to develop classes named KMeans and SoftKMeans using Python and Numpy. KMeans will implement the standard KMeans algorithm, while SoftKMeans will handle the Soft KMeans algorithm. Both will integrate non-local split-and-merge moves to potentially enhance clustering outcomes. For the Image Reconstruction Problem, we'll create PCA and LinearAutoEncoder classes. PCA will be responsible for executing the PCA algorithm, and LinearAutoEncoder will develop a linear autoencoder model.

Test the performance of the model: These models' performance will be tested on a given dataset, experimenting with different cluster numbers for KMeans and SoftKMeans, and analyzing their clustering results, particularly after the addition of non-local split-and-merge moves. In Selecting the Principal Color of an image and Image Reconstruction, we'll employ PCA, Linear AutoEncoder, and Soft KMeans with varying principal components, displaying and critically analyzing the reconstructed images to gauge the efficacy of these algorithms.

III. Method and Algorithms

A. KMeans

KMeans is a clustering algorithm, which aims to partition n observations into k clusters in which each observation belongs to the cluster with the nearest mean, serving as a prototype of the cluster.

The standard algorithm for KMeans clustering is as follows: [2]

1) Initialization: Choose k initial centroids either randomly from the data points (the simplest method) or using more complex methods like KMeans++ or furthest-points initialization which tries to choose initial centroids that are far apart, improving the chances of convergence.

2) Assignment step: Assign each data point to the nearest centroid, with the nearest defined using Euclidean distance. This forms k clusters.

3) Update step: Calculate the new centroids (based on the mean) of the observations in the new clusters from the assignment step.

4) Convergence check: Repeat the assignment and update steps until the centroids do not change significantly between iterations, meaning the algorithm has converged. Alternatively, the algorithm can be stopped after a predefined number of iterations or when the improvement falls below a certain threshold.

B. Soft KMeans

Soft KMeans is a variant of the KMeans algorithm that allows for a "softer" assignment of points to clusters. Instead of assigning each data point to a single cluster, Soft KMeans assigns a probability or degree of belonging to each cluster, indicating the strength of the association with each cluster.

The basic framework of the Soft KMeans algorithm is the same as the KMeans algorithm. The main difference lies in the Assignment step. [2]

1) Soft Assignment Step: For each data point, compute the probability of belonging to each cluster. This probability can be based on the distance from the point to each centroid. Often a softmax function is used, which includes a hyper-parameter β that controls the softness of the assignment:

$$p_{ij} = \frac{exp(-\beta \cdot d(x_i, c_j))}{\sum_{l=1}^{k} exp(-\beta \cdot d(x_i, c_l))} \tag{1}$$

where p_{ij} is the probability of data point x_i belonging to cluster j, and $d(x_i, c_j)$ is the distance from point x_i to centroid c_j.

C. non-local split-and-merge in KMeans

The KMeans algorithm with non-local split-and-merge moves is an enhanced version of the standard KMeans algorithm. It is used to adjust the clustering results obtained by the kmeans or soft kmeans algorithm. This can potentially overcome some of the limitations of standard K-means, such as its sensitivity to initial conditions and difficulty in detecting clusters of varying sizes and densities.

The algorithm of non-local split-and-merge moves is as follows: [2]

1) Split the most scattered cluster: By calculating the variance of a class of data points, the class with the highest degree of discreteness is selected, and the two furthest points in this class are selected as the new class center. The labels in this class can be updated through the kmeans algorithm, and finally, recalculate the new class center (mean of data points) based on the latest Label.

2) Merge the two closest clusters: Calculate the distance between each class center separately to select the two closest class centers. Update the labels of the data points belonging to the two class centers to the same class. Finally, re-update the class center according to the new labels.

3) Convergence Check: Repeat the split-and-merge moves until the centroids stabilize or another convergence criterion is met.

D. Different Initialization algorithms in KMeans

1) KMeans++ Initialization: KMeans++ aims to choose initial centroids that are far away from each other. This approach tends to spread out the initial centroids, which can lead to better final clustering results compared to randomly selecting initial centroids, as in the standard KMeans. [1]

The KMeans++ algorithm for initializing centroids is as follows:

Choose the First Centroid: Randomly select the first centroid c_1 from the data points X.

Compute Squared Distances for Each Data Point: For each data point x in the dataset X, calculate the squared distance $D(x)^2$ to the nearest centroid that has already been chosen.

Choose Subsequent Centroids Based on Squared Distances: The next centroids are chosen randomly from the dataset, where the probability of choosing point x as the next centroid is proportional to $D(x)^2$. This means points farther away from existing centroids have a higher chance of being selected as centroids.

$$Probability = \frac{D(x)^2}{\sum_{x\prime \in X} D(x\prime)^2} \tag{2}$$

Repeat Until All k Centroids Are Chosen.

2) Farthest Point Initialization: This method is based on the idea of maximizing the distance between centroids.

The Farthest Point Initialization algorithm is as follows:

Choose the First Centroid: Randomly select the first centroid c_1 from the dataset X.

Compute Distances for Each Data Point: For each remaining data point X, calculate the distance $D(x,c)$ to the nearest already chosen centroid in centroids set C.

Choose Subsequent Centroids Based on Distances: Select the data point with the maximum distance $D(x,c)$ as the next centroid and add it to set C.

Repeat Until All k Centroids Are Chosen.

E. PCA

Principal Component Analysis (PCA) is a statistical technique used for dimensionality reduction while preserving as much of the variance in the dataset as possible. It achieves this by transforming the data to a new coordinate system where the greatest variances lie on the first few coordinates. [3]

1) Standardize the Dataset: Given a dataset X of dimensions nxm (where n is the number of data and m is the number of the features of the data), standardize each variable (along the column).

$$x_{ij}\prime = \frac{x_{ij} - \bar{x}_j}{s_j} \tag{3}$$

where $\bar{x}_j$ is the mean and s_j is the standard deviation of the j-th feature.

2) Covariance Matrix Computation: Compute the covariance matrix C of the standardized dataset. The covariance matrix is an mxm matrix where each element represents the covariance between two variables.

$$C = \frac{1}{N} \sum_{n=1}^{N} (x\prime^{(n)} - \bar{x})(x\prime^{(n)} - \bar{x})^T \tag{4}$$

where $\bar{x}$ is the mean of the dataset along the column, x^n is one of the data, and N is the number of data.

3) Eigenvalue Decomposition: Perform eigenvalue decomposition on the covariance matrix C. The results in a set of eigenvalues and their corresponding eigenvectors. The eigenvectors determine the directions of the new feature space, and the eigenvalues determine their magnitude.

4) Select Principal Components: Sort the eigenvectors by decreasing eigenvalues and choose the top k eigenvectors. These eigenvectors are the principal components of the dataset.

5) Project the Data: Project the standardized data X onto the selected principal components to obtain the transformed data in the reduced dimension. The transformed data Y is given by:

$$Y = XV \tag{5}$$

where V is the matrix containing the selected eigenvectors.

F. Linear AutoEncoder

A linear autoencoder is a specific type of autoencoder used in machine learning for dimensionality reduction, similar in some respects to Principal Component Analysis (PCA). An autoencoder is a type of neural network that is trained to copy its input to its output with a hidden layer (or layers) that represents a code used to represent the input. [3]

There are two important concepts in linear autoencoder, namely encoding and decoding.

1) Encoder: The encoder maps the input $x \in R^d$ (where d is the dimension of the input data) to a hidden representation $z \in R^p$ (where p is the dimension of the hidden representation, with $p < d$)

$$z = W_{encoder}x + b_{encoder} \tag{6}$$

where $W_{encoder} \in R^{pxd}$ is the weight matrix and $b_{encoder} \in R^p$ is the bias vector for the encoder.

2) Decoder: The decoder maps the hidden representation z back to a reconstructed input $\hat{x} \in R^d$

$$\hat{x} = W_{decoder}x + b_{decoder} \tag{7}$$

where $W_{decoder} \in R^{dxp}$ is the weight matrix and $b_{decoder} \in R^d$ is the bias vector for the decoder.

3) Training Algorithm: Initialization: Initialize the weights and biases of the encoder and decoder.

Forward: Compute the hidden representation z for each input x using the encoder. Compute the reconstructed input $\hat{x}$ using the decoder.

Backward: Compute the gradient of the loss function concerning the parameters. Usually, MSE is used to measure the error between the input x and the reconstructed input $\hat{x}$.

$$MSELoss = \frac{1}{2}|x - \hat{x}|^2 \tag{8}$$

According to the equation of the Encoder(eq.6) and the equation of the Decoder(eq.7), the gradients of the parameters are computed as follows.

$$\begin{aligned} \frac{\partial L}{\partial W_{encoder}} &= \frac{\partial L}{\partial \hat{x}} \cdot \frac{\hat{x}}{\partial z} \cdot \frac{\partial z}{\partial W_{encoder}} \\ &= (\hat{x} - x) \cdot W_d \cdot x^T \end{aligned} \tag{9}$$

$$\begin{aligned} \frac{\partial L}{\partial b_{encoder}} &= \frac{\partial L}{\partial \hat{x}} \cdot \frac{\hat{x}}{\partial z} \cdot \frac{\partial z}{\partial b_{encoder}} \\ &= (\hat{x} - x) \cdot W_d \end{aligned} \tag{10}$$

$$\begin{aligned} \frac{\partial L}{\partial W_{decoder}} &= \frac{\partial L}{\partial \hat{x}} \cdot \frac{\hat{x}}{\partial W_{decoder}} \\ &= (\hat{x} - x) \cdot z^T \end{aligned} \tag{11}$$

$$\frac{\partial L}{\partial b_{decoder}} = \frac{\partial L}{\partial \hat{x}} \cdot \frac{\hat{x}}{\partial b_{decoder}} \tag{12}$$
$$= \hat{x} - x$$

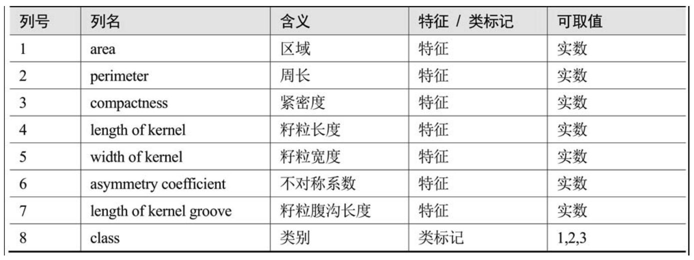
Update the parameters using an optimization algorithm like gradient descent, according to the gradients above.

Repeat the forward and backward passes for a specified number of epochs or until convergence.

IV. Experiment results and analysis

A. Clustering

In the clustering problem of this project, we consider the wheat seed data set as given in seeds_dataset.txt. This dataset contains 210 data samples. Each sample has seven input features and one output label, which is described in the following table.



列号	列名	含义	特征 / 类标记	可取值
1	area	区域	特征	实数
2	perimeter	周长	特征	实数
3	compactness	紧密度	特征	实数
4	length of kernel	籽粒长度	特征	实数
5	width of kernel	籽粒宽度	特征	实数
6	asymmetry coefficient	不对称系数	特征	实数
7	length of kernel groove	籽粒腹沟长度	特征	实数
8	class	类别	类标记	1,2,3

Fig. 1. Basic information about the data set

To evaluate the quality of our clustering model, both quantitative and qualitative methods were utilized.

For the quantitative method: In the first case, when the correct labels are known, we compare the accuracy ($accuracy\_score$ from $sklearn.metrics$) of clustering between the clustering labels from our model and the standard clustering labels and the Adjust Rand coefficient ($adjusted\_rand\_score$ from $sklearn.metrics$) of the clustering labels. To solve the problem that the cluster labels obtained by our clustering model do not correspond to the standard cluster labels, the method named $labels\_optimal\_mapping()$ is written. It can find the optimal mapping relationship between the clustering labels of our clustering model and the standard clustering labels, and at the same time modify our clustering labels based on the optimal mapping relationship and output the final clustering result. In the second case, when the correct labels are unknown, we calculate the Silhouette Coefficient ($silhouette\_score$ from $sklearn.metrics$) to evaluate the results.

For the qualitative method: We use the PCA model to reduce the dimension of the data features to two and visualize the classification results on a two-dimensional plane. The implemented functions are $plot\_graph\_with\_labels()$ and $plot\_graph()$ respectively. In the graph, different classes are marked with different colors, and the center of each class is represented by a cross icon. At the same time, for the case with correct labels (the first function), the points with red borders in the plot represent points with incorrect clustering.

1) Normal Kmeans & SoftKmeans: In this experiment, the number of classes was set to 3 and 10 respectively to cluster the data. The method of initializing the class center is random.

When the number of classes is 3, the results are as follows:

TABLE I
Normal Kmeans & Soft_Kmeans clustering results (K=3)

	Kmeans	Soft Kmeans
Accuracy_score	0.8905	0.9095
Adjusted_rand_score	0.7103	0.7519

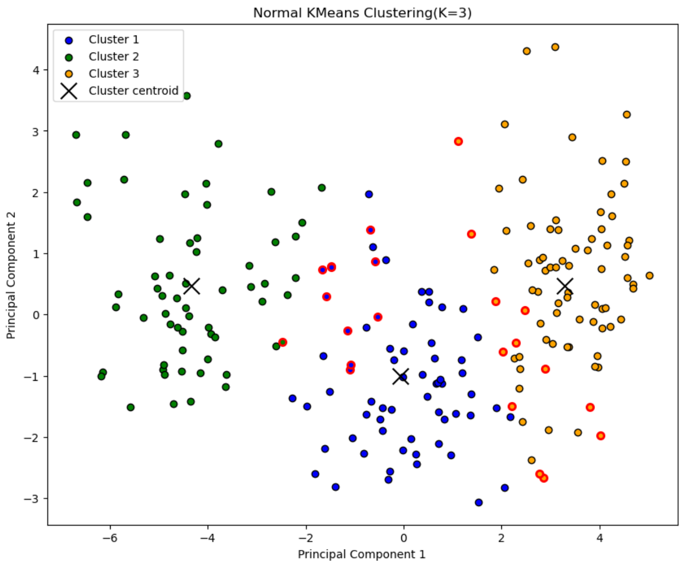


Fig. 2. Visualized results of Kmenas Clustering (K=3)

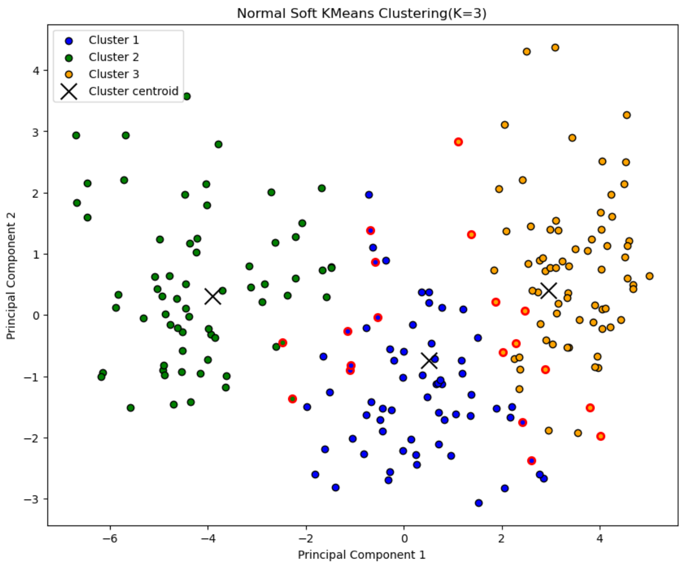


Fig. 3. Visualized results of Soft Kmenas Clustering (K=3)

The accuracy rate of both is close to 90%, and the adjusted Holland coefficient is also 0.7. In the visual results, only a few points are misclassified, and the class centers are also distributed relatively evenly. The clustering results are quite good. Comparing Kmeans and Soft Kmeans, it can be seen that the results of Soft Kmeans are slightly better than Kmeans, as can be seen from the accuracy rate, adjusted Holland coefficient, and visualization results.

When the number of classes is 10, the results are as follows:

TABLE II
NORMAL KMEANS & SOFT_KMEANS CLUSTERING RESULTS (K=10)

	KMeans	Soft Kmeans
silhouette_score	0.3603	0.3673

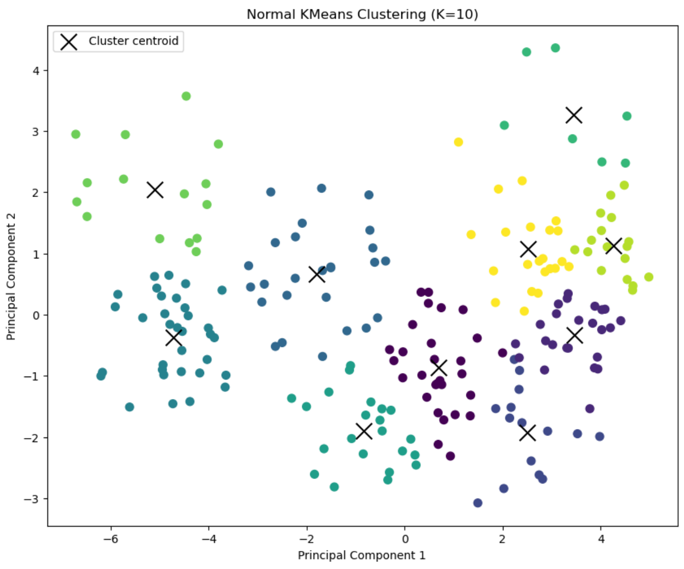


Fig. 4. Visualized results of Kmenas Clustering (K=10)

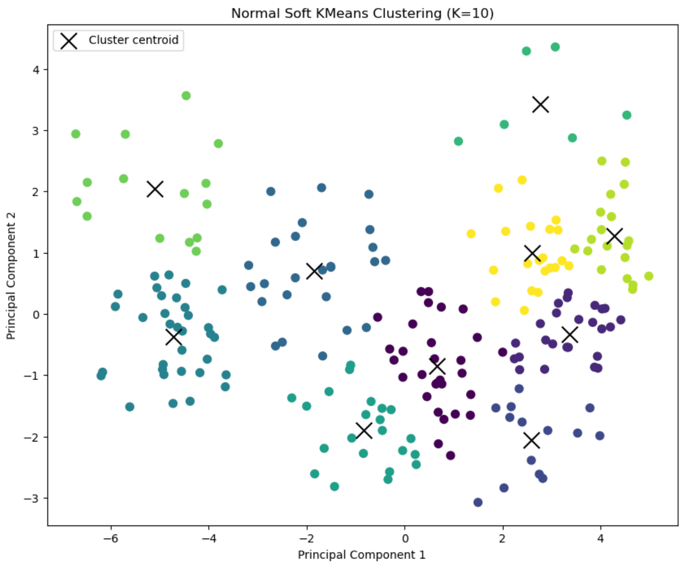


Fig. 5. Visualized results of Soft Kmenas Clustering (K=10)

The silhouette_score of both is approximately 0.36. In the visual results, the class centers are also distributed relatively evenly. Moreover, the results of Soft Kmeans are slightly better than Kmeans. It seems that the clustering results are not good. However, the cause of the problem is not the model itself, but the data set. After all, the data set itself is of three categories, and it may not have the potential to be divided into ten categories.

Based on the results of the above two experiments, soft kmeans can better complete clustering. This is because Soft K-Means provides a membership score for each point, indicating the degree to which the point belongs to each cluster. This allows data points to be assigned probabilistically into multiple clusters rather than strictly into a single cluster. This can capture the ambiguous nature of data points, especially when the boundaries between clusters are not very clear. At the same time, in traditional KMeans, an outlier far away from the cluster center will have a greater impact on the calculation of the cluster center, while in Soft KMeans, its impact will be reduced due to the reduction of the membership degree, which has a greater impact on noise and outliers. More robust. Therefore, Soft Kmeans will have a better classification effect than Kmeans to a certain extent.

2) Add non-local split-and-merge moves: This experiment adds non-local split-and-merge moves to Normal Kmeans and Normal Soft Kmeans, and sets the number of clusters to 10. By comparing the effect of non-split-and-merge moves with the effect of the previous experiment, we can verify whether each other has entered the local minimum situation.

The results are as follows:

TABLE III
ADJUSTED KMEANS & SOFT_KMEANS CLUSTERING RESULTS (K=10)

	Adjusted KMeans	Adjusted Soft Kmeans
silhouette_score	0.3609	0.3520

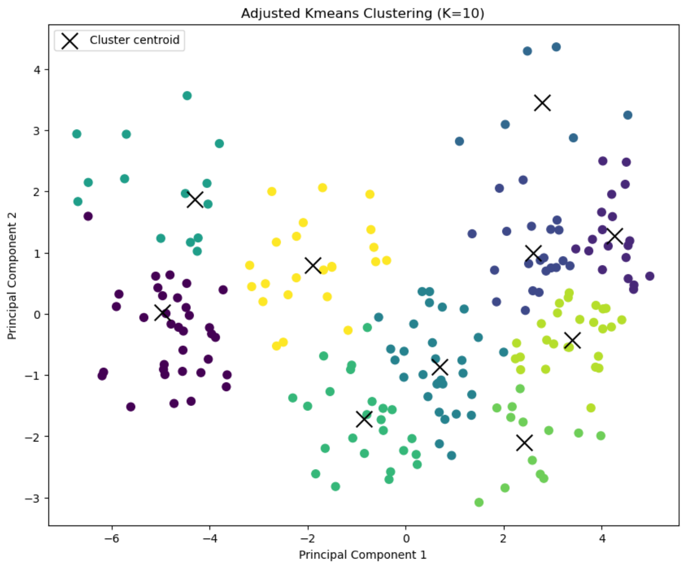


Fig. 6. Visualized results of Adjusted Kmenas Clustering (K=10)

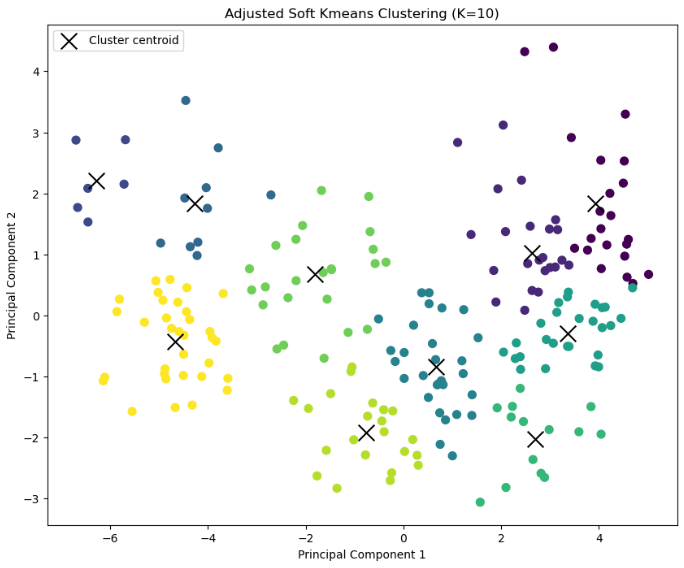


Fig. 7. Visualized results of Adjusted Soft Kmenas Clustering (K=10)

Through comparison, we found that adding non-local split-and-merge moves will bring a slight improvement to the normal kmeans, but will not bring an improvement to the normal soft kmeans. Therefore, we can guess that in the previous experiment, kmeans and soft kmeans did not fall into the local optimum, or in a better local optimum. After adding non-local split-and-merge moves, soft kmeans will reach a worse local optimal place.

3) Optimize the initialization of class center: After doing the above experiments, we found that it is easier to obtain better clustering results by choosing a better initial class center. Compared to randomly initializing the class centers, making the class centers more evenly dispersed in space will make clustering easier. Through online searches, the Kmeans++ algorithm and the initialization method of selecting the farthest point can make the initialization of the class center more dispersed and uniform.

This experiment selected five random states and used three initialization methods to cluster the data in each random state with the Kmeans algorithm. The quantitative results are as follows:

TABLE IV
DIFFERENT INITIALIZATIONS PERFORMANCES

	Random	Kmeans++	Furthest_points
20: Accuracy_score	0.8905	0.8952	0.8952
20: Adjusted_rand_score	0.7103	0.7166	0.7166
666: Accuracy_score	0.8905	0.8905	0.8952
666: Adjusted_rand_score	0.7103	0.7103	0.7166
2023: Accuracy_score	0.8952	0.8952	0.8952
2023: Adjusted_rand_score	0.7166	0.7103	0.7166
20041128: Accuracy_score	0.8952	0.8952	0.8952
20041128: Adjusted_rand_score	0.7103	0.7166	0.7166
20040918: Accuracy_score	0.8952	0.8952	0.8952
20040918: Adjusted_rand_score	0.7166	0.7166	0.7166

Theoretically, the ranking of clustering effects should be that Kmeans++ initialization is better than farthest points initialization and better than random initialization. Compared with random initialization, the initialization of the first two will make the initialized class centers more dispersed, which is beneficial to convergence. Since kmeans++ selects the centroid based on the probability distribution of the square of the distance, which has a certain degree of randomness, it usually produces more diverse initial centroid combinations and is more flexible than the farthest point initialization.

TABLE V
DIFFERENT INITIALIZATIONS AVERAGE PERFORMANCES

	Random	Kmeans++	Furthest_points
accuracy_score	0.8933	0.8943	0.8952
adjusted_rand_score	0.7141	0.7141	0.7166

It can be seen from the average that Kmeans++ and the furthest points are slightly better than the random initialization method. However, it can be seen that the effect is not particularly obvious. This may be because the data set itself has relatively good separability.

B. *Find Principal Color of an image*

In this experiment, we used PCA, linear autoencoder, and soft kmeans to find the principal color of an image with different numbers. For PCA and linear autoencoder, the image data is an array of (16384x3), and each pixel corresponds to three main colors (RGB). Therefore, when reducing the dimension of the data, it can only be reduced to 1 and 2, that is, in three dimensions The color space selects one or two main color axes, and reconstructing the image maps other points to these axes. For soft kmeans, we assign the color of all points away from the class center to the color of the class center, so we can choose many colors. In this experiment, we selected 1, 2, 3, 6, 10, and 20 respectively. The specific operations are written in three functions respectively: PCA_principal_colors(), LinearAutoEncoder_principal_colors(), and SoftKmeans_principal_colors().

This experiment qualitatively evaluates the reconstruction effect by visualizing the reconstructed image, and gaining a deeper understanding of these three machine learning algorithms.

Below are the two pictures used in this experiment, one is a grayscale image and the other is a color image. The pixels of the pictures are all 128x128.

Fig. 8. Original Input Gray Image (128x128)

Fig. 9. Original Input RGB Image (128x128)

Firstly, PCA is used to find the principal color component of the Image. The visualized results are as follows.

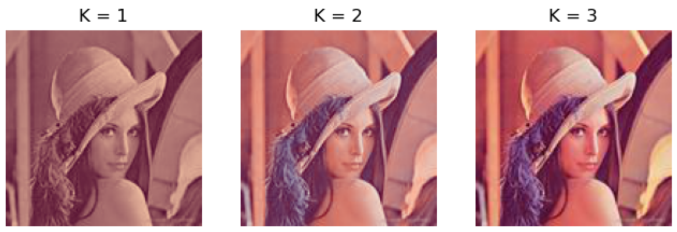


Fig. 10. Reconstructed Gray Images using PCA

Secondly, a linear autoencoder is used to find the principal color component of the Image. The visualized results are as follows.

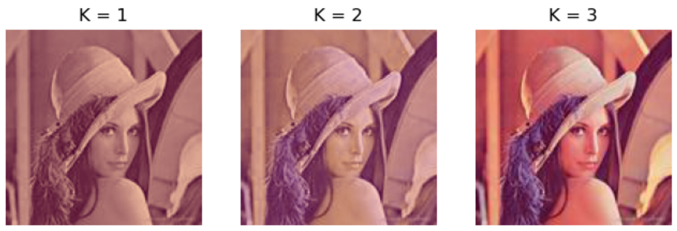


Fig. 11. Reconstructed Gray Images using PCA

Finally, Soft KMeans is used to find the principal color component of the Image. The visualized results are as follows.

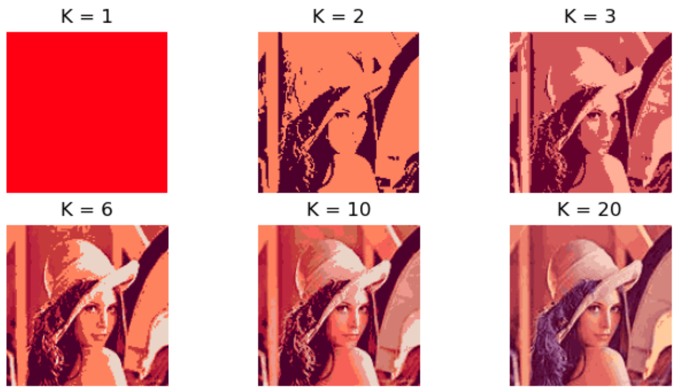


Fig. 12. Reconstructed Gray Images using PCA

For PCA and linear autoencoder, it can be seen that when the number of main colors is selected as 1, the image correctly selects the main color of red. When the number of main colors is selected as 2, the image also correctly selects red and blue as two colors. Comparing PCA and linear autoencoder, it is found that the image restoration effect of linear autoencoder is slightly worse than PCA. The reason may be that the linear autoencoder is not trained well.

For soft kmeans, it can be seen that the number of colors in the reconstructed image is the same as the number we set, and the correctness of its reconstruction has been verified. Compared with linear autoencoder and PCA, its color distribution is more regional. When the number of colors is relatively small, more obvious color blocks appear.

C. *Image Reconstruction*

In this experiment, we used PCA, linear autoencoder, and soft kmeans to reduce the dimensionality and reconstruct the image respectively, and selected different numbers of principal components, which were 1, 10, 20, 50, 100, and 200 respectively. The process of reconstructing the image is to reduce the dimensionality of the features (384) of the image data (128x384), and then use the reduced data to reconstruct the image. The specific operations are written in three functions respectively: PCA_img_reconstruction(), LinearAutoEncoder_img_reconstruction(), and SoftKmeans_img_reconstruction().

This experiment qualitatively evaluates the reconstruction effect by visualizing the reconstructed image and gains a deeper understanding of these three machine learning algorithms.

Firstly, PCA is used to reconstruct the Image. The visualized results are as follows.

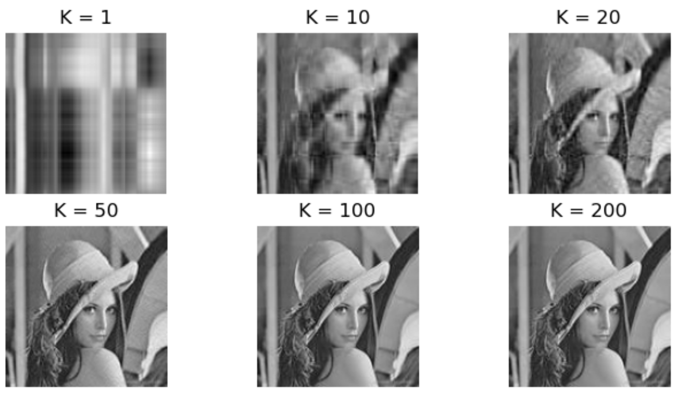


Fig. 13. Reconstructed Gray Images using PCA

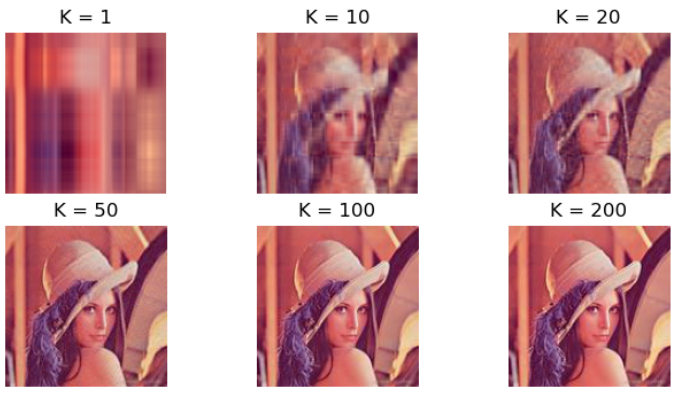


Fig. 14. Reconstructed RGB Images using PCA

Judging from the results, when using PCA to reconstruct the image, the general meaning of the image can be seen when K=10, and when K=50, the clarity is not much different from the original image. The quality of image reconstruction is quite high.

Secondly, Linear AutoEncoder is used to reconstruct the Image. The visualized results are as follows.

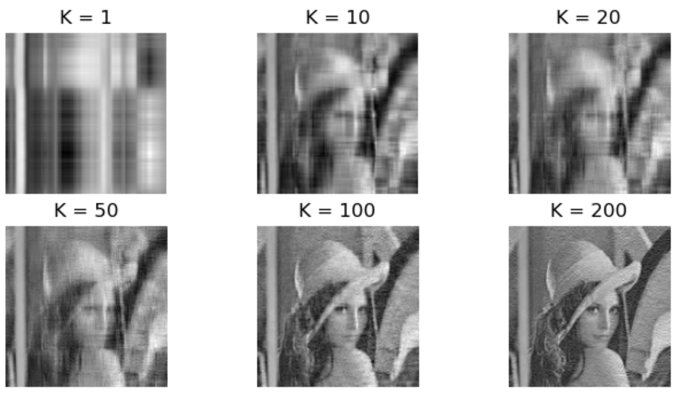


Fig. 15. Reconstructed Gray Images using Linear AutoEncoder

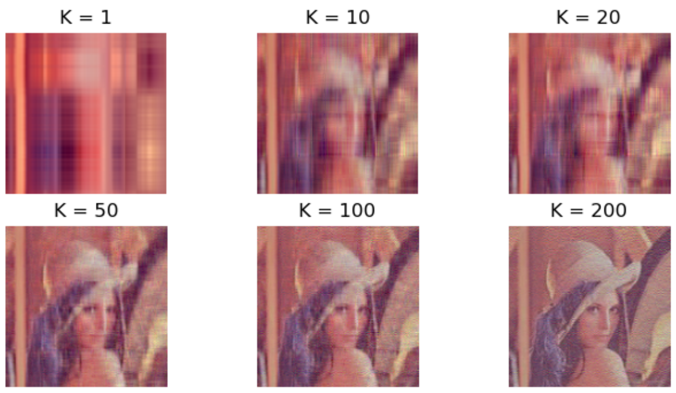


Fig. 16. Reconstructed RGB Images using Linear AutoEncoder

Judging from the results, using a linear autoencoder to reconstruct the image can roughly read the image information when K=10, but it is still quite blurry. When K=100, the clarity is not much different from the original image. The image reconstruction quality is not bad.

Thirdly, Soft KMeans is used to reconstruct the Image. The visualized results are as follows.

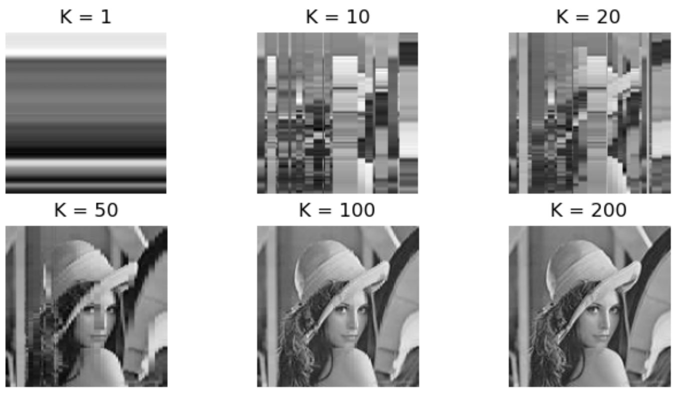


Fig. 17. Reconstructed Gray Images using Soft KMeans

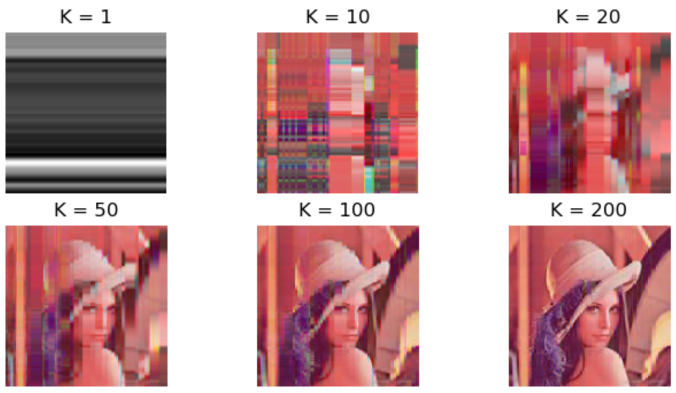


Fig. 18. Reconstructed RGB Images using Soft KMeans

Judging from the results, using Soft Kmeans to reconstruct the image can roughly read the image information when K=20, and there are some abrupt color blocks. When K=100, the clarity is not much different from the original image. The image reconstruction quality is very poor when K is small, but when K increases to a certain extent, such as around 50, it is not bad.

Comparing the three image reconstruction methods, PCA has the best effect, and it is also the easiest to train.

When using a linear autoencoder to reconstruct images, from the final result, the image seems to be covered with a layer of gray. This may be caused by the loss of data accuracy during the training process. At the same time, the process of training linear autoencoder is the most difficult.

When using soft kmeans to reconstruct the image, the final result is still good. It is difficult to distinguish the difference from the original image with the naked eye. However, when K is very small, there will be obvious abrupt color blocks, which will greatly reduce the readability of images.

By comparison, we will find that the reconstructed images of PCA and liner autoencoder are relatively smooth, while the reconstructed image of soft kmeans is not smooth when K is relatively small. This is because PCA and linear autoencoder consider the global situation when selecting principal components, and Soft kmeans is essentially a clustering process, so the selection of its principal components (class centers) is regional, which leads to the appearance of color patches in the image reconstruction.

V. Conclusion and future problems

This project has significantly enhanced my academic journey, deepening my understanding of essential concepts covered in our coursework. Developing and coding clustering algorithms (KMeans, SoftKMeans) and dimensionality reduction techniques (PCA, LinearAutoEncoder) in Python using numpy has markedly sharpened my programming skills. Furthermore, this practical application has given me a more profound comprehension of the fundamental theories and algorithms.

Through hands-on experimentation with the wheat seed dataset, valuable insights into the functionality and performance of the clustering models are gained. Our analysis highlighted the effectiveness of clustering algorithms in grouping data. Moreover, to further refine our clustering results, non-local split-and-merge moves and different initialization methods are integrated into KMeans and SoftKMeans, demonstrating the potential of algorithmic enhancements in practical applications.

The practical experimentation with Lena's image offered a significant opportunity to understand and evaluate various dimensionality reduction techniques. This analysis not only showcased the capability of these algorithms in preserving vital image information and color information for reconstruction but also brought forth a comparative understanding of their distinct methodologies. Delving into the nuances of each technique enriched my comprehension of their operational mechanisms and effectiveness in the context of image processing.

Moving forward, several areas present opportunities for further research. Firstly, exploring more complex datasets and real-world applications would validate the versatility of these models. Additionally, some more advanced algorithms like gmm can be utilized for the clustering problem, and experimenting with non-linear activation functions in neural network-based models like AutoEncoder could unveil more sophisticated patterns in image reconstruction problems. Finally, another promising avenue is the integration of advanced optimization techniques like Adam Optimizer to improve the speed and accuracy of these algorithms.

References

[1] Arthur, David, and Sergei Vassilvitskii. ”K-means++ the advantages of careful seeding.” Proceedings of the eighteenth annual ACM-SIAM symposium on Discrete algorithms. 2007.

[2] Z. Lin, ”Lecture note on Clustering,” 2023. [Online]. Available: Lecture Note

[3] Z. Lin, ”Lecture note on PCA,” 2023. [Online]. Available: Lecture Note