K-means, PCA and Linear Autoencoder for Classification and Image Processing

Liu Chengtao 12212708

***Abstract*—This article mainly explains the effectiveness of using K-means and soft K-means for classification with different numbers of clusters, and using PCA, Linear Autoencoder and soft K-means to deal with image with different dimensions. And they are all implemented using NumPy and Python.**

***Index Terms*—Python, Numpy, K-means, soft K-means, PCA, Linear Autoencoder, classification, image processing, Machine Learning**

I. Introduction

We know that k-means is a good method for handling classification problems. In this article, we will use both k-means and soft k-means methods, varying the number of clusters, to observe their effects on the classification of the seed dataset.

Additionally, in the article, we will explore PCA, linear autoencoders, and the soft k-means method, examining their respective effects on image processing. Furthermore, we will investigate the reconstruction effects of images when altering the retained dimensions and the number of clusters.

II. K-means

A. Basic information

My K-means exists in "AI_kmeans". In k-means, there are four parameters: the first one is the number of clusters, the second one indicates whether to use the split-and-merge strategy, and the last two are the maximum iteration count which is initially set to 300 and tolerance which is initially set to 1e-5. In k-means, our objective is to minimize the following function: (Where C_i is the ith cluster, μ_i is the center of the ith cluster, and x is a sample point in cluster C_i.)

$$J = \sum_{i=1}^{k} \sum_{x \in C_i} ||x - \mu_i||^2 \tag{1}$$

B. Basic implementation of classificationn

To minimize the above function, inside the "fit" method, first, we use seed number one to generate random numbers, and then randomly generate cluster centers within a suitable range. Then, we calculate which cluster center is closest to each point and assign it to that cluster center. Next, for each cluster center, we take the average of the points within that cluster as the new cluster center. We then check whether the distance between the new and original cluster centers is less than the tolerance. If it is, we consider the current cluster centers as the final ones and stop the iteration. If the distance between them is greater than the tolerance, then the iteration continues to eventually obtain the clustering results.

C. Label adjustment

After the "fit" method concludes, we obtain our corresponding prediction results. However, we notice that the output labels are arranged in a way such as 0, 1, 2, and as the number of clusters increases, the output labels may not match our actual labels. Therefore, to better understand the model's classification results, we use the "adjust_labels" method to adjust the output labels.

In the "adjust_labels" method, we compare the true labels of each sample within a cluster, find the label with the highest proportion, and replace the labels of that cluster with the label from the actual classification. By using this method to replace the labels, we can effectively measure the classification performance of the model through accuracy.

D. Split and merge

In k-means, the "split_and_merge" method has been added to adjust the clustering. In the initial idea, the goal is to detect, in each iteration, the average distance from each point within a class to the cluster center. When it exceeds a certain threshold, the class is split, or it is merged with the nearest cluster center. However, I found it challenging to obtain an effective threshold using this approach, and it could easily lead to an explosion in the number of clusters.

Therefore, I later changed the strategy. In each iteration, I calculate and identify the cluster with the maximum average distance from its points to the cluster center, label it as "max_distance_cluster," and split this cluster. Then, I compute the distances between each pair of cluster centers, identify the two closest clusters, label them as "closest_clusters," and merge them. This approach effectively avoids falling into local optima while ensuring the stability of the number of clusters.

III. Soft K-means

A. Basic information

My soft K-means exists in "AI_Soft_kmeans" and Soft k-means has five initial parameters, with the first four set similarly to k-means, and the last one setting beta to 0.7. The classification logic of soft k-means is basically similar to k-means, but it introduces a parameter "w" to describe the likelihood of belonging to a certain cluster. In k-means, our objective is to minimize the following function: (Where x_i is the ith data point, μ_j is the center of the jth cluster and w_{ij} is the probability that data point x_i belongs to cluster j.)

$$J = \sum_{i=1}^{n} \sum_{j=1}^{k} w_{ij} ||x_i - \mu_j||^2 \tag{2}$$

This is the definition of w_{ij}: (Where β is the set of parameter beta.)

$$w_{ij} = \frac{e^{-\beta||x_i-\mu_j||^2}}{\sum_{l=1}^{k} e^{-\beta||x_i-\mu_l||^2}} \tag{3}$$

For other methods within the class, in each iteration, Soft K-means first calculates the probability w_{ij} for each data point belonging to each cluster. Then, based on these probabilities, it updates the cluster centers μ_j:

$$\mu_j = \frac{\sum_{i=1}^{n} w_{ij}x_i}{\sum_{i=1}^{n} w_{ij}} \tag{4}$$

The implementation approach for the "adjust_labels" and "split_and_merge" methods in Soft K-means is essentially similar to that in k-means. I won't go into too much detail here since the basic idea remains the same.

B. Initialize centroids

In testing, I found that the results of Soft K-means are significantly influenced by the selection of initial points. Therefore, I further adopted a strategy to filter the initial centroids. Before the iteration, initialize an empty array centroids to store the coordinates of the centroids for each cluster. Randomly select a data point as the initial centroid for the first cluster. Through a loop, iteratively select centroids for each cluster, ensuring that the new centroid is sufficiently far from existing centroids. When selecting each centroid, calculate the probability distribution based on the square of distances to make a more targeted selection for the next centroid. This initialization method helps improve the performance of the K-means clustering algorithm, making it more likely to converge to the global optimum. Similarly, I attempted to add the same strategy in K-means, but the results were not satisfactory. The accuracy varied each time, so I ultimately decided not to incorporate this strategy into K-means.

IV. Classification for the Seed Dataset

A. Different tolerance levels when k=3

Model	Max Iterations	Tolerance	Accuracy
KMeans	300	1e-3	0.7952
KMeans	300	1e-4	0.7952
KMeans	300	1e-5	0.7952
SoftKMeans	300	1e-3	0.8952
SoftKMeans	300	1e-4	0.8952
SoftKMeans	300	1e-5	0.8952

TABLE I
Test results of KMeans and SoftKMeans with different tolerances

From the TABLE I, we can see that changing the tolerance did not have a significant impact on the model. Therefore, using a tolerance of 1e-5 to terminate the iteration process early is acceptable.

Model	Max Iterations	Tolerance	Accuracy
KMeans	100	1e-5	0.7333
KMeans	200	1e-5	0.8905
KMeans	300	1e-5	0.7952
KMeans	400	1e-5	0.7619
KMeans	500	1e-5	0.7667
SoftKMeans	50	1e-5	0.8952
SoftKMeans	100	1e-5	0.8952
SoftKMeans	200	1e-5	0.8952
SoftKMeans	300	1e-5	0.8952

TABLE II
Test Results for KMeans and SoftKMeans

B. Different numbers of iterations when k=3

From the TABLE II, we can see that the accuracy of the KMeans model fluctuates significantly at different maximum iteration numbers, while the accuracy of the SoftKMeans model remains stable across different maximum iteration numbers.

For the KMeans model, the highest accuracy is achieved when the maximum iteration number is set to 200. This may indicate that, for this dataset, increasing the number of iterations does not always improve the model's accuracy. Beyond a certain point, excessive iterations may lead to overfitting, resulting in a decrease in performance on the test set.

In contrast, for the SoftKMeans model, the accuracy remains at 0.8952 regardless of the maximum iteration number. This suggests that, for this dataset, increasing the iteration number does not alter the model's accuracy. This may be because, after selecting the initial centroids, the model converges in fewer iterations. Therefore, for subsequent testing, we choose a maximum iteration number of 200.

C. Performance comparison

Model	Clusters	Split and Merge	Accuracy
KMeans	3	False	0.8905
SoftKMeans	3	False	0.8952
KMeans	10	False	0.8619
SoftKMeans	10	False	0.9
KMeans	10	True	0.8619
SoftKMeans	10	True	0.8905

TABLE III
KMeans and SoftKMeans Test Results

From the TABLE III we can observe that regardless of the initial conditions, the performance of SoftKMeans is consistently superior to KMeans.

Next, let's take a different perspective. Within the same type of model, first, let's examine the results of the KMeans model. When the number of clusters is 3, the model achieves an accuracy of 0.8905. However, as the number of clusters increases to 10, the accuracy drops to 0.8619. This may suggest that, for your dataset, increasing the number of clusters does not always improve the model's accuracy. Additionally, whether or not the split-and-merge strategy is applied, the model's accuracy remains at 0.8619. This may indicate that, for your dataset, employing the split-and-merge strategy does not alter the model's accuracy. Moving on, let's examine the results of the SoftKMeans model. When the number of clusters is 3, the model achieves an accuracy of 0.8952. As the number of clusters increases to 10, the accuracy improves to 0.9. This may suggest that, for the dataset, increasing the number of clusters could enhance the model's accuracy. However, when applying the split-and-merge strategy, the model's accuracy decreases to 0.8905. This might indicate that, for the dataset, employing the split-and-merge strategy could lower the model's accuracy.

In summary, the SoftKMeans model appears to perform better on this specific problem, as it achieves higher accuracy and is less sensitive to parameter choices.

V. PCA

My PCA exists in "AI_PCA" and PCA accepts only one parameter, which is "n_components," representing the number of dimensions to be retained. In the "fit", firstly, for the input data, a process of centering is performed. Then, by computing the covariance matrix and performing singular value decomposition according to the following two equations, the principal n components are obtained.

$$C = \frac{1}{n} X_{\text{centered}}^T X_{\text{centered}} \tag{5}$$

$$C = U \Sigma V^T \tag{6}$$

Use the "transform" method to project the data onto the principal component space. (Where w represents the first n principal components: W = V[:, :k])

$$X_{\text{projected}} = X_{\text{centered}} W \tag{7}$$

Then, use the "inverse_transform" method to restore the data from the principal component space to the original space.

$$X_{\text{recovered}} = X_{\text{projected}} W^T + \mu \tag{8}$$

VI. Linear Autoencoder

A. Basic information

My PCA exists in "AI_LinearAutoencoder" and it takes two parameters, namely "input_size" and "encoding_size". During the initialization of weights, they are multiplied by a factor of 0.00001, but not set to zero. This is done to prevent symmetry issues while maintaining numerical stability and preventing the problem of gradient explosion. (This data is obtained after numerous experiments).

B. Encode and decode

In the encoding and decoding process, we perform operations on E for encoding and D for decoding. Here, E represents the encoded representation, D represents the decoded data, X represents the input data, W_e represents the encoder weights, and W_d represents the decoder weights.

$$E = X \cdot W_e \tag{9}$$

$$D = E \cdot W_d \tag{10}$$

C. Train

The iterative update principles of a linear autoencoder and an MLP (Multi-Layer Perceptron) are similar, except that a linear autoencoder does not involve non-linear activation functions. For gradient calculation, the error gradient can be represented as:

$$\nabla R = 2 \cdot \frac{D - X}{n} \tag{11}$$

The gradient of the encoder and decoder weights can be represented as: (The method of gradient clipping is adopted to ensure that the gradients are within a reasonable range.)

$$\nabla W_e = X^T \cdot (\nabla R \cdot W_d^T) + W_e \tag{12}$$

$$\nabla W_d = E^T \cdot \nabla R + W_d \tag{13}$$

Next is the method for updating weights:

$$W_d = W_d - lr \cdot \nabla W_d \tag{14}$$

$$W_e = W_e - lr \cdot \nabla W_e \tag{15}$$

VII. Different Methods for Image Reconstruction

A. Img choose

For image processing, the requirement is to import the three-dimensional RGB colors, calculate the main dimensions, retain them, and then reconstruct the image. In testing, it was found that if the pixel size of the image is too large, it can easily result in slow computation, which is not ideal for testing the model's effectiveness. Additionally, during the reconstruction of highly colorful images, missing 1 to 2 dimensions of RGB can result in poor restoration effects. Therefore, in the end, an image of a German soldier from World War I was chosen as the object for restoration.

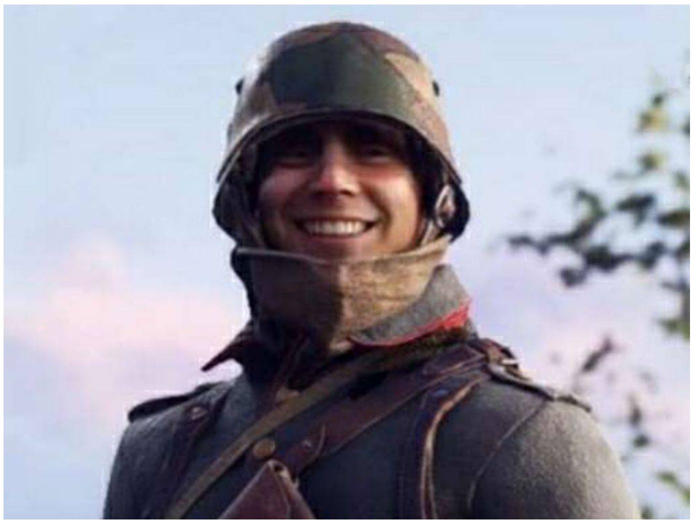

Fig. 1. The Image Used

B. PCA

We can observe that when retaining only one dimension of the principal component, the reconstructed image becomes a grayscale image. However, when retaining two dimensions of the principal components, due to the overall similarity in color, the resulting image exhibits only slight color variations, providing a rough reconstruction of the image. Retaining three dimensions is equivalent to not removing any components, resulting in the original image.

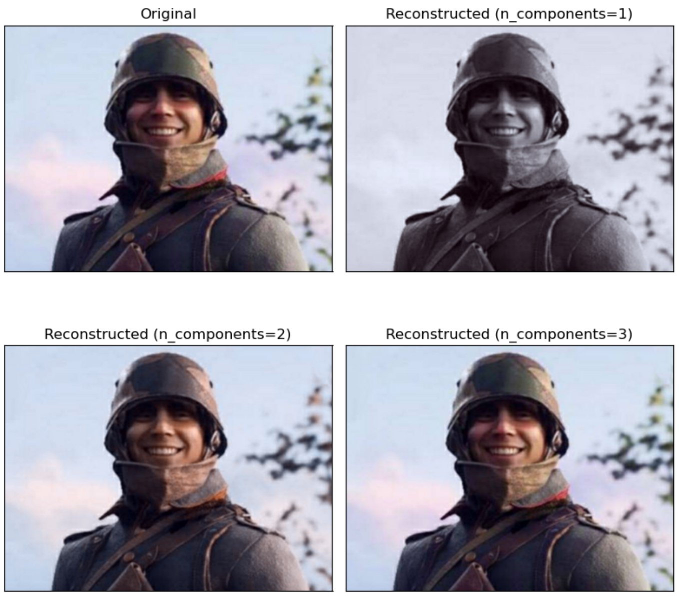
Original
Reconstructed (n_components=1)
Reconstructed (n_components=2)
Reconstructed (n_components=3)

Fig. 2. Image Reconstructed via PCA

C. linear autoencoder

When using a linear autoencoder to reconstruct the image, the resulting reconstructed image is roughly similar to the one obtained through PCA. It resembles a grayscale image when retaining one dimension of the principal component, closely resembles the original image's color when retaining two dimensions, and almost identical to the original image when retaining three dimensions.

However, it is worth noting that when reconstructing images with autoencoders, the choice of initial weights is highly sensitive and can easily lead to the problem of gradient explosion. Therefore, random allocation of initial weights may result in the failure of image reconstruction. To ensure better image reconstruction, preliminary filtering of initial weights is recommended.

D. Soft K-Means

Next, we use Soft KMeans to reconstruct images, setting the number of clusters from 1 to 9 to observe their reconstruction. We can see that as the number of clusters increases, the degree of reconstruction continues to improve, and the color gradually increases with the increase in the number of clusters. Finally, when the number of clusters increases to 9, the image can be fairly well reconstructed.

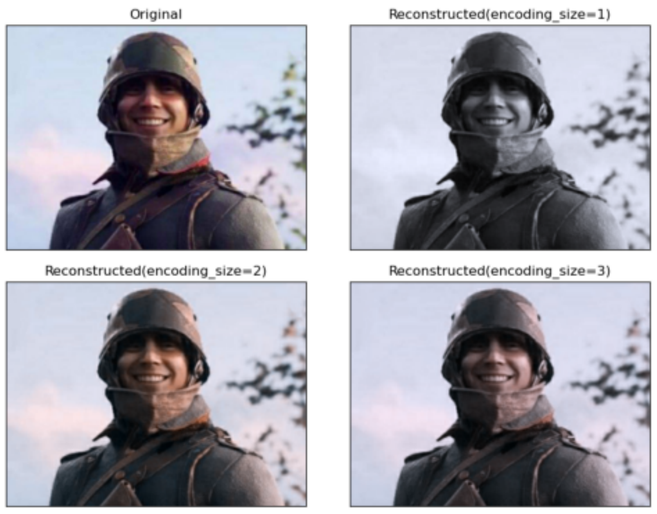
Original
Reconstructed(encoding_size=1)
Reconstructed(encoding_size=2)
Reconstructed(encoding_size=3)

Fig. 3. Image Reconstructed via linear autoencoder

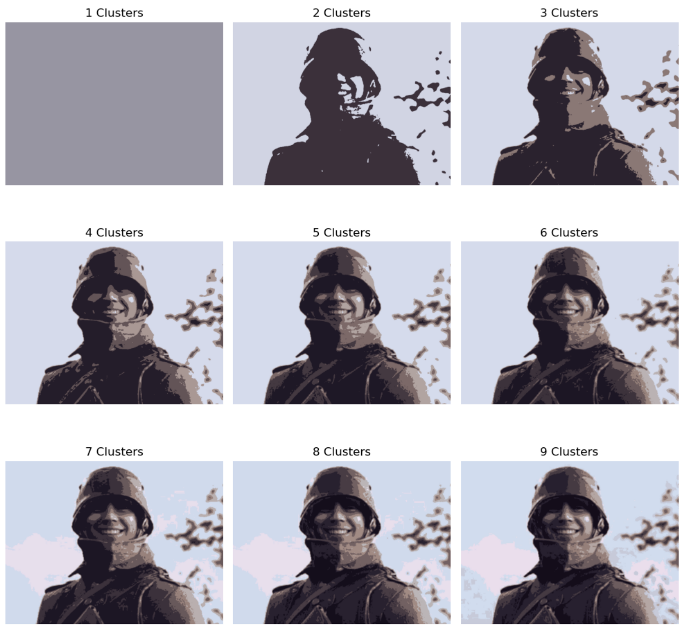

Fig. 4. Image Reconstructed via soft K-means

Furthermore, adding the split-and-merge strategy for image reconstruction yields a similar reconstruction effect, with only slight color variations.

VIII. Conclusio and future work

In this article, we explored the application of K-means and Soft K-means for classification using the seed dataset. The Soft K-means consistently outperformed K-means, showing higher accuracy and robustness to parameter variations. Additionally, we investigated the impact of different tolerance levels and maximum iterations on the models' performance.

Furthermore, we delved into the use of PCA and Linear Autoencoder for image processing. Both methods were applied

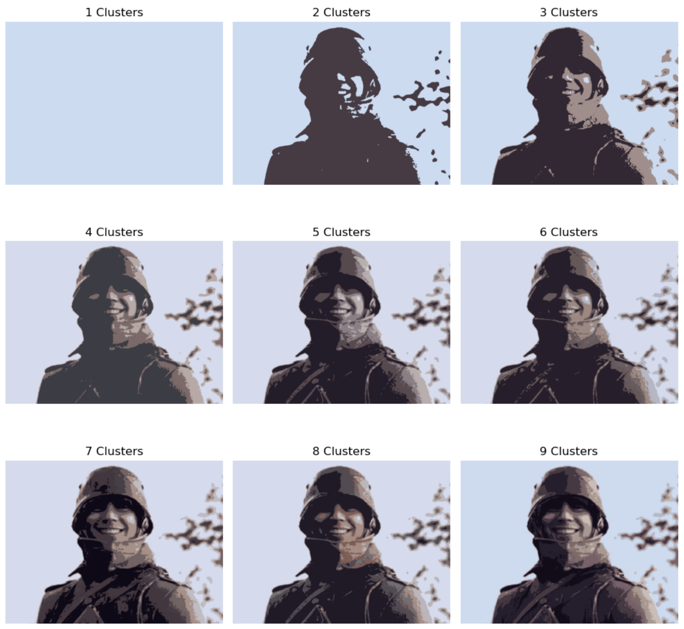

Fig. 5. soft K-means with split-and-merge

to reconstruct an image, and their effectiveness was compared. Additionally, we discussed the sensitivity of Linear Autoencoder to the choice of initial weights and the importance of filtering them for better performance.

In terms of image reconstruction using Soft K-means, we observed improvements as the number of clusters increased, achieving satisfactory results when the number of clusters reached 9. The split-and-merge strategy was also applied, demonstrating a similar reconstruction effect.

In the end, there are still some limitations in the experiments. Despite incorporating controls on initializing centroids or weights in the experiments, occasional issues related to gradient explosion due to random generation may arise. Additionally, there is a lack of further validation of the model's handling capabilities when faced with other datasets. It is hoped that, given the opportunity, more in-depth research can be conducted to address these aspects!