ÖZYEĞİN
UNIVERSITY

Introduction to Machine Learning and Artificial Neural Networks

CS 554

HOMEWORK 2 Report
k-Means Clustering for Unsupervised Learning

By: *Amin Deldari Alamdari*
S033174

Course Given By:

Prof. Ethem ALPAYDIN

Faculty of Engineering

Computer Science Deparment

1. INTRODUCTION

Clustering is a popular method of exploratory data analysis that is used to gain insight into the structure of the data. It involves the identification of subgroups within the data, such that data points within the same subgroup (cluster) are highly similar, while data points in different clusters are very different. In other words, the goal is to find homogeneous subgroups within the data, such that data points in each cluster are as similar as possible according to a similarity measure, such as Euclidean-based distance or correlation-based distance. The choice of which similarity measure to use is dependent on the application.

Clustering analysis can be conducted on the basis of features or samples. Here, we will focus on clustering based on features. This technique is used in market segmentation to identify customers with similar behaviors or attributes, image segmentation/compression to group similar regions together, and document clustering to classify topics.

Clustering is an unsupervised learning technique, which means that it does not require a ground truth to compare the output of the algorithm to. Instead, it is used to explore the structure of the data by dividing the data points into distinct subgroups.

2. PROCEDURE AND METHODS

In this homework, our task is to implement **k-means** clustering to calculate cluster centroids on the *Fashion-MNIST* dataset.

The dataset contains 28×28 images from 10 different classes. Each class represents a different type of fashion item but in this homework, we are not going to use the class information. Each row of the file corresponds to one instance and there are 60000 instances. The first column of each row contains the label of the image (which we will ignore), and the remaining 784 columns that represent the grayscale value of each pixel constitute the input image that we will use to calculate the cluster centroids.

We are required to implement k-means clustering for $k = [10, 20, 30]$, where for each k:

1. Plot the reconstruction loss as a function of iterations;
2. Plot the cluster centroids as 28x28 grayscale images. These different centers should roughly look like different variants of examples from the different classes.

2.1. PLOT THE RECONSTRUCTION LOSS AS A FUNCTION OF ITERATIONS

We must import the *matplotlib*, *csv*, *random*, and *math* libraries into our environment.

I load the data from *fashion_mnist.csv* file by ignoring header row and the first column of the table. For this I create load_fashion_mnist_csv(ignore_header=True, ignore_labels=True) function. This function returns list of lists, where each inner list contains the pixel values of a single image.

After that, I define the function initialize_centroids($data$, k) where initializes k cluster centroids randomly within the range of the data. The k is the number of clusters and $data$ is the list of data points. This function returns a list of k cluster centroids.

The initialize_centroids function is essential for the initialization step of the k-means clustering algorithm. It takes two parameters: the dataset ($data$) represented as a list of data points, and the desired number of clusters (k)). The function transposes the dataset to determine the number of dimensions, and for each cluster, it randomly selects values within the range of each dimension to generate a centroid. This ensures that the initial centroids are spread across the entire range of the data, providing a diverse starting point for the k-means algorithm. The function then returns a list of k) cluster centroids, each represented as a list of coordinates in the dataset's feature space. The random initialization helps the algorithm converge to meaningful cluster assignments, allowing for exploration of potential cluster configurations.

The calculate_distance($ponit1$, $point2$) function computes the Euclidean distance between two points in a multidimensional space. The function takes as input two points, each represented as a list of coordinates (`point1` and `point2`). Before proceeding with the distance calculation, the function ensures that both points have the same number of dimensions, raising an assertion error if this condition is not met. The Euclidean distance is then computed using the formula:

$$\text{distance} = \sqrt{\sum_{i=1}^{n} (point1[i] - point2[i])^2}$$

Here, n is the number of dimensions shared by both points. The function utilizes the `math.sqrt` function to obtain the square root of the sum of squared differences between corresponding coordinates. The result is the Euclidean distance between the two points, providing a measure of their dissimilarity in the feature space.

The assign_to_clusters($data$, $centroids$) function is a critical component of the k-means clustering algorithm, tasked with assigning each data point to its nearest cluster centroid. The function takes two parameters: a list of data points (`data`) and a list of cluster centroids (`centroids`). For each data point in the dataset, the function calculates the Euclidean distance to all centroids using the `calculate_distance` function. Subsequently, it identifies the index of the centroid with the minimum distance, representing the nearest cluster. The function compiles these cluster assignments into a list named `clusters`, where each element denotes the index of the assigned cluster for the corresponding data point. The resulting `clusters` list is pivotal in the subsequent stages of the k-means algorithm, influencing the reevaluation of cluster centroids during the iterative optimization process. This assignment process plays a fundamental role in forming distinct clusters based on the proximity of data points to the centroids, iteratively refining the clustering until convergence. The `assign_to_clusters` function encapsulates a key aspect of the k-means algorithm, facilitating the partitioning of data into meaningful clusters based on their spatial relationships with the cluster centroids.

The update_centroids(*data*, *clusters*, *k*) function is a crucial step in the iterative optimization process of the k-means clustering algorithm. This function recalculates the centroids of each cluster by taking the mean of all the data points assigned to that cluster. It takes three parameters: a list of data points (`data`), a list of cluster assignments (`clusters`), and the number of clusters (`k`).

The function initializes an empty list called `new_centroids` to store the updated cluster centroids. It then iterates over each cluster index, extracting the data points that are assigned to the current cluster. For non-empty clusters, the mean of the coordinates is calculated for each dimension, forming the new centroid. The mean is computed using the `sum` function and dividing by the number of data points in the cluster.

If a cluster is empty, indicating that no data points are assigned to it, the centroid remains unchanged, set to a list of zeros with the same length as the data points' dimensions.

The resulting `new_centroids` list contains the updated centroids for all clusters based on the mean of the assigned data points. This process is essential for refining the cluster centroids in each iteration of the k-means algorithm, converging towards a stable clustering configuration that accurately represents the underlying patterns in the data.

The has_converged function is a critical component of the k-means clustering algorithm, designed to assess whether the centroids have undergone significant changes between consecutive iterations. This function takes three parameters: a list of centroids from the previous iteration (`prev_centroids`), a list of centroids from the current iteration (`new_centroids`), and an optional convergence threshold (`threshold`), which defaults to 1×10^{-4}.

The function initiates the evaluation by computing the Euclidean distance between corresponding centroids using the `calculate_distance` function. These distances are stored in the `centroid_distances` list. Subsequently, the function checks whether all centroid distances are below the specified threshold. If this condition holds for all centroids, the function returns `True`, signifying that the algorithm has converged. Otherwise, it returns `False`.

The `has_converged` function serves as a vital convergence criterion for the k-means algorithm. A `True` return indicates minimal changes in centroids, prompting the algorithm to terminate, as further iterations are unlikely to significantly impact cluster assignments. This mechanism ensures the efficient convergence of the k-means algorithm to a stable solution while preventing unnecessary iterations.

The k_means function performs k-means clustering on the given data. It takes three parameters: a Numpy array of data points (data), the desired number of clusters (k), and the maximum number of iterations (max_iterations), with a default value of 15.

The function initializes the cluster centroids using the initialize_centroids function and an empty list called loss_history to record the reconstruction loss for each iteration. It then enters a loop that iteratively updates centroids, assigns data points to clusters, and calculates the reconstruction loss. The loop continues until the maximum number of iterations is reached or convergence is detected using the has_converged function.

Within each iteration, the function records the Euclidean distance-based reconstruction loss, providing insight into the convergence behavior. The result is a tuple containing the final cluster centroids (centroids) and a list (loss_history) detailing the reconstruction loss for each iteration.

This function encapsulates the whole k-means clustering procedure, offering a dependable implementation for dividing data into distinct groups based on centroid assignments.

The plot_loss_all_k function generates a plot illustrating the reconstruction loss as a function of iterations for different values of k in the same plot. It takes three parameters: a list of iteration numbers (iterations), a list of lists containing the reconstruction loss values for each k (loss_history_list), and a list of corresponding k values (k_values).

The function utilizes the matplotlib library to create a plot with a specified figure size. It then iterates over the lists of reconstruction loss values for different k values, plotting each curve with a distinctive marker and label. The resulting plot provides a visual comparison of how the reconstruction loss evolves over iterations for various k values.

The title, xlabel, and ylabel are set to enhance the clarity of the plot, and the legend is added to distinguish the different k values. The function concludes by displaying the generated plot.

This function is useful for assessing the convergence behavior of the k-means algorithm across different k values.

2.2. PLOT THE CLUSTER CENTROIDS AS GRAY-SCALE IMAGES

The plot_centroids function generates a plot visualizing the cluster centroids as grayscale images. It takes a single parameter, a list of cluster centroids represented as NumPy arrays (centroids).

The function calculates the number of centroids (num_centroids) and creates a single plot with subplots for each centroid. The figure size is determined based on the number of centroids. For each centroid, the function reshapes the data into a 28x28 array, treating it as an image. It then plots the image by iterating over the pixels and using the pixel values to determine the grayscale color. The resulting plot provides a visual representation of each cluster centroid as a 28x28 image.

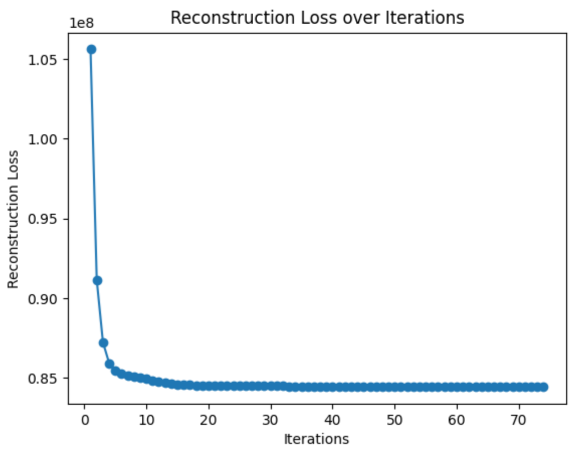
Each subplot is labeled with the corresponding centroid index, and the axis is turned off to focus on the visual representation. The function concludes by displaying the generated plot.

This function is valuable for inspecting and interpreting the characteristics of the cluster centroids in the context of image data, as each centroid is visualized as a representative image.

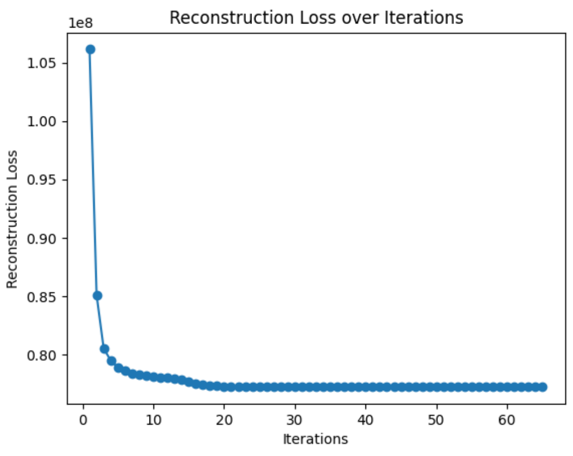
3. RESULTS AND DISCUSSION

3.1. RESULTS

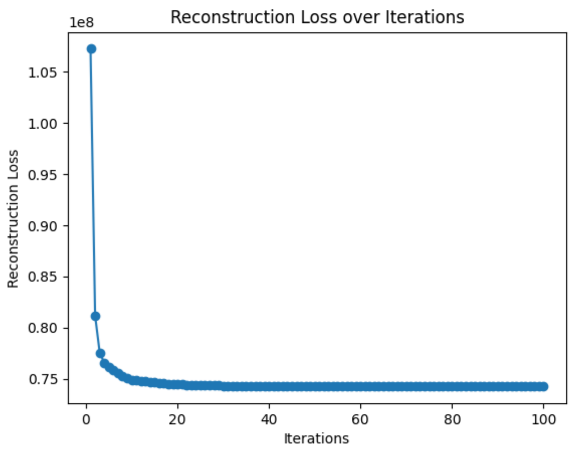
After implementing all the methods and algorithms which are described in the previous section. I ran the program for the 100 iterations and the results for this are shown in Figure 3.1. As one can see from the figures, after $10th$ iteration, all the instances converge. Therefore, I decided to run the program for one more time with less iteration number (15 iterations) which results in less computational time. The results of the second computation are given in Figure



(a) K = 10



(b) K = 20



(c) K = 30

Figure 3.1: Plot of Reconstruction Loss as a Function of Iterations (for 100 iterations).

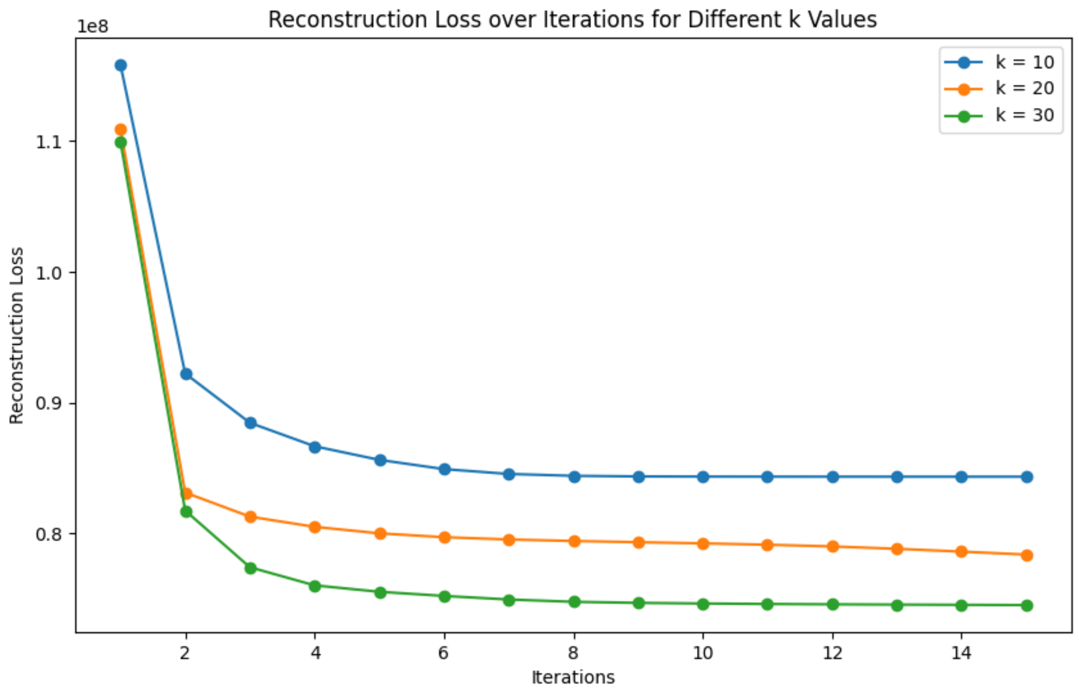


Figure 3.2: Plot of Reconstruction Loss as a Function of Iterations (for 15 iterations).

We are required to plot the cluster centroids as 28×28 gray-scale images. The result of these are shown in Figure 3.3, Figure 3.4, and Figure 3.5 for k values 10, 20, and 30 respectively.

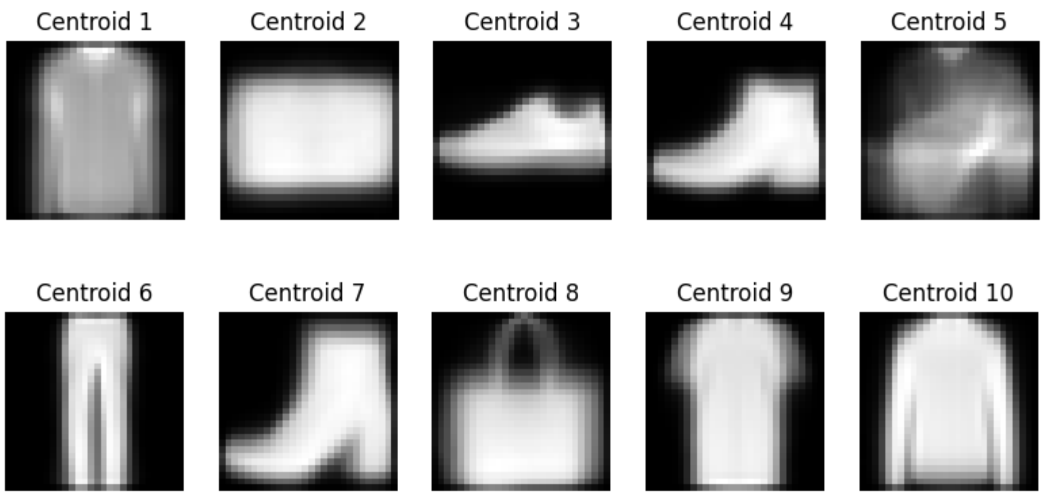


Figure 3.3: Plot of the Cluster Centroids as Gray-Scale Images for k = 10.

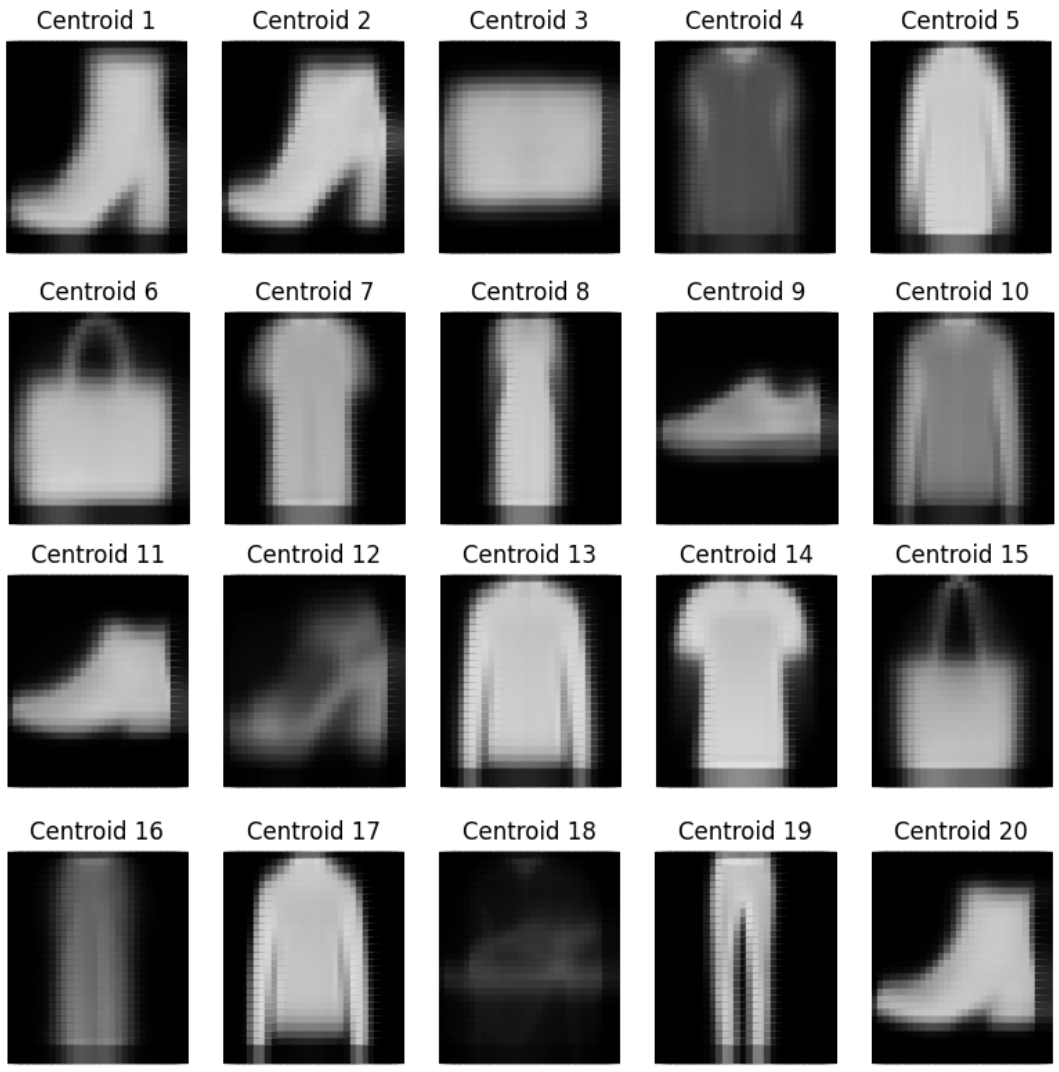


Figure 3.4: Plot of the Cluster Centroids as Gray-Scale Images for k = 20.

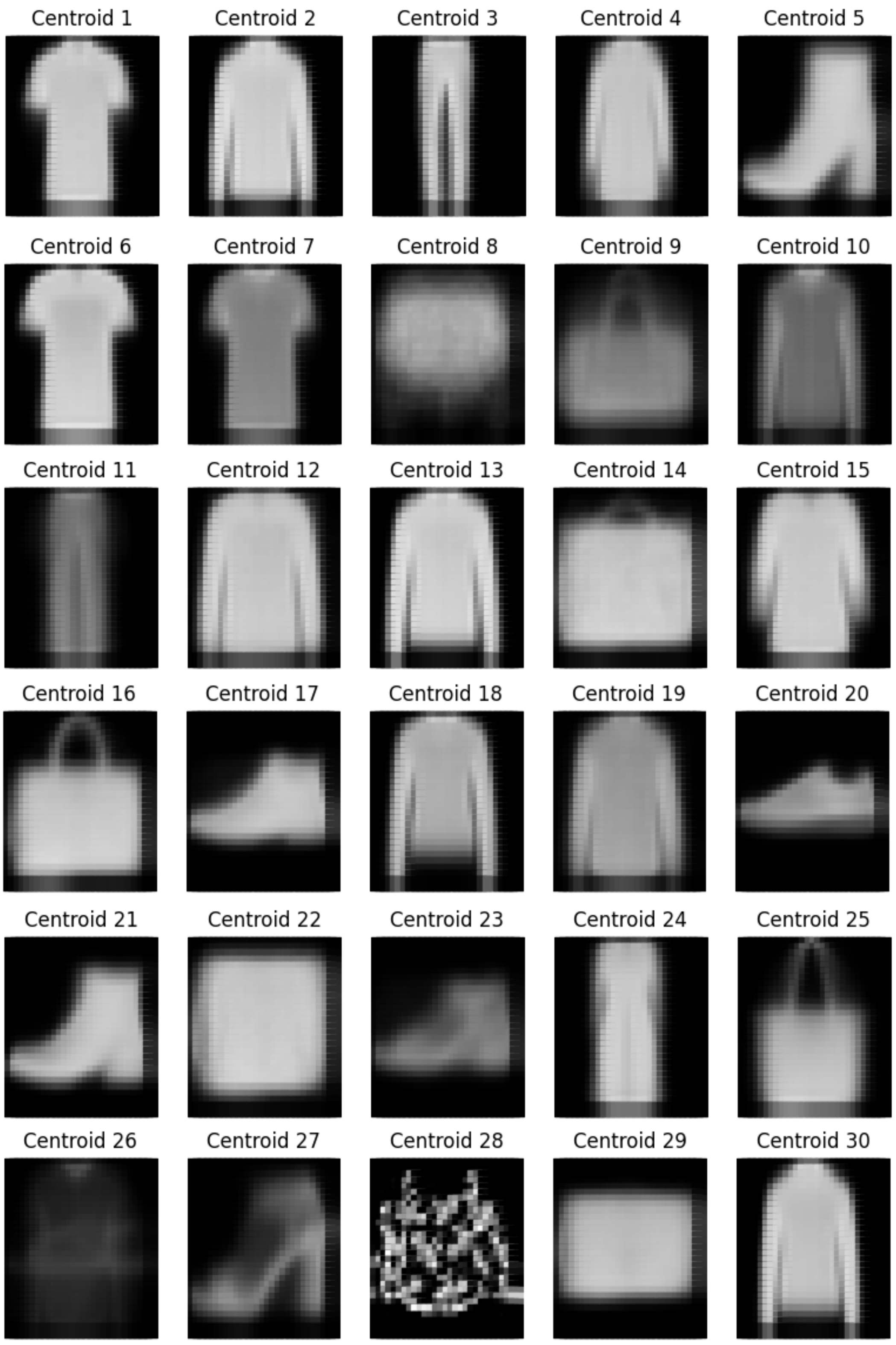


Figure 3.5: Plot of the Cluster Centroids as Gray-Scale Images for k = 30.

3.2. DISCUSSION

The investigation of k-means clustering with different values of k (10, 20, and 30) over a period of 15 iterations revealed the intricate relationship between convergence patterns and reconstruction errors which is shown in Figure 3.2. Each k value had its own distinct characteristics in terms of convergence steadiness and reconstruction error development. Notably, the plot showed an elbow point at k = 20 (iteration 2), which suggested a critical point where the model's accuracy gains started to level off. This highlights the delicate balance between capturing subtle patterns in the data and the diminishing returns of increased clustering precision.

An insightful facet of the discussion relates to the computational implications of varying k values. The experiment exposed a discernible increase in computational costs as k value increased, with k = 30 incurring notably higher expenses in comparison to k = 10 and k = 20. Weighing the pros and cons of a more precise segmentation against the computational load, it is essential to make a wise decision on the value of k. This will ensure that the clusters in the data are optimally distinguished while still adhering to the available resources. Achieving this equilibrium is critical for the successful application of k-means clustering in practical situations.

The analysis of the chosen k values leads to a reflection on their broader applicability within the context of the dataset. The discussion focuses on the interpretability of the results, questioning whether the optimal k not only reduces the reconstruction error but also reflects the structure and complexity of the underlying data. This highlights the significance of contextual relevance, making sure that the chosen clustering configuration is in line with the intricacies of the dataset and enables meaningful insights in the particular problem domain.

In conclusion, the plot indicates that after the second iteration, all k-values begin to converge gradually. Based on this limited analysis, it appears that k = 20 yields the best results in terms of computation time and the steadiness of the convergence.