**Introduction to Artificial Intelligence" Experiment report**

|  |
| --- |
| The experimental report gives the content elements in the following order:   1. Procedure flow chart 2. Experimental results and analysis diagram    1. Handwritten code realizes K-means clustering, which is used for classification of 2D data sets.    2. K-means algorithm is used for image compression.    3. Using the K-means clustering in the sklearn toolkit, compare the handwriting algorithm with the sklearn toolkit and compare the difference between different algorithms    4. Compare the difference of results under different parameter K and different criterion selection. 3. Source code and necessary comments 4. Experimental summary and experience   Note: The experiment report is required to be completed independently, and students are allowed to discuss with each other, but plagiarism is absolutely not allowed. Once found, this experiment is recorded as 0 points.  **Learning websites you can refer:**   1. [**https://realpython.com/k-means-clustering-python/**](https://realpython.com/k-means-clustering-python/) **(how to use sklearn toolkit to use kmeans)** 2. [**https://www.datacamp.com/tutorial/k-means-clustering-python**](https://www.datacamp.com/tutorial/k-means-clustering-python) **(how to use sklearn toolkit to use kmeans)** 3. [**https://www.geeksforgeeks.org/k-means-clustering-introduction/**](https://www.geeksforgeeks.org/k-means-clustering-introduction/)**(how to use numpy to write kmeans algorithm)** 4. [Image compression using K-means clustering - GeeksforGeeks](https://www.geeksforgeeks.org/image-compression-using-k-means-clustering/) **(how to use k**means **algorithm to d**o image compression**)**   **K-Means Clustering Algorithm**  Flow Chart of K-Means Clustering Algorithm:   * **Start:** Begin the K-Means clustering experiment. * **Generate Data:** Create a 2D multi-class dataset with randomly generated data. * **Initialize Centroids:** Choose initial centroids for each cluster. * **Assign Data Points:** Assign each data point to the nearest centroid. * **Update Centroids:** Recalculate the centroids based on the assigned data points. * **Convergence Check:** Check for convergence (stable centroids or a set number of iterations). * **Output Results:** Display the clustered data. * **End:** Conclude the K-Means clustering experiment.   **C:\Users\Alex Joshua Chirwa\Pictures\Screenshots\Screenshot (159).png**  **Handwritten K-Means Clustering**   * Handwritten K-means clustering refers to the application of the K-means clustering algorithm on datasets that consist of handwritten characters or digits.   **Experimental Implementation:**  Imported Libraries:  import numpy as np import matplotlib.pyplot as plt from sklearn.cluster import KMeans from sklearn.utils import shuffle from sklearn.datasets import make\_blobs from sklearn.metrics import pairwise\_distances\_argmin\_min  Generating Random Data:  # Generate random data X, y = make\_blobs(n\_samples=300, centers=3, random\_state=42)   * In this section it generates a two-dimensional multi-class dataset with 300 samples 3 centers using the ‘make\_blobs’ function from ‘sklearn.datasets’   Handwritten K-Means Clustering:  def kmeans\_clustering(X, n\_clusters, n\_iterations=100):  # Initialize centroids randomly  initial\_centroids = X[np.random.choice(len(X), n\_clusters, replace=False)]  centroids = initial\_centroids.copy()   for \_ in range(n\_iterations):  # Assign points to nearest centroid  labels = pairwise\_distances\_argmin\_min(X, centroids)[0]   # Update centroids  new\_centroids = np.array([X[labels == i].mean(axis=0) for i in range(n\_clusters)])   # Check for convergence  if np.all(centroids == new\_centroids):  break   centroids = new\_centroids   return labels, initial\_centroids, centroids   * This function implements a K-Means clustering algorithm for 2D data with a specified number of clusters(‘n\_clusters’). It returns the cluster labels, initial centroids, and final centroids after a specified number of iterations (100).   **Source code:**  import numpy as np import matplotlib.pyplot as plt from sklearn.cluster import KMeans from sklearn.utils import shuffle from sklearn.datasets import make\_blobs from sklearn.metrics import pairwise\_distances\_argmin\_min  X, y = make\_blobs(n\_samples=300, centers=3, random\_state=42)  def kmeans\_clustering(X, n\_clusters, n\_iterations=100):  # Initialize centroids randomly  initial\_centroids = X[np.random.choice(len(X), n\_clusters, replace=False)]  centroids = initial\_centroids.copy()   for \_ in range(n\_iterations):  # Assign points to nearest centroid  labels = pairwise\_distances\_argmin\_min(X, centroids)[0]   # Update centroids  new\_centroids = np.array([X[labels == i].mean(axis=0) for i in range(n\_clusters)])   # Check for convergence  if np.all(centroids == new\_centroids):  break   centroids = new\_centroids   return labels, initial\_centroids, centroids  **Result:**  C:\Users\Alex Joshua Chirwa\AppData\Local\Packages\Microsoft.Windows.Photos_8wekyb3d8bbwe\TempState\ShareServiceTempFolder\Screenshot (150).jpeg  **Elbow Method for Optimal Clusters**   * The Elbow Method is a technique used to find the optimal number of clusters (k) in a K-Means clustering algorithm. * The basic Idea is to run K-Means clustering on the dataset for a range of values of k and, for each value, calculate the sum of squared distances from each point to its assigned center. * Distortion is calculated as the sum of squared distances between each point and its assigned center. * The formula for distortion is often given as: **Distortion =∑ni=1 ∑kj=1 ‖xi – cj‖2** * Where xi is a data point, cj is the center of cluster j, and n is the number of data points. * The value of k at which the distortion starts to decrease at a slower rate is considered the ‘**elbow**’ and is typically chosen as the optimal number of clusters.   **Implementation of the experiment:**  def elbow\_method(X, max\_clusters=10):  distortions = []  for i in range(1, max\_clusters + 1):  kmeans = KMeans(n\_clusters=i, random\_state=42, n\_init=10)  kmeans.fit(X)  distortions.append(kmeans.inertia\_)  return distortions   * This function implements the Elbow Method to find the optimal number of clusters. It calculates distortions for different cluster numbers and returns a list of distortions.   **Handwritten K-Means Plot and Elbow Method Plot**  # Handwritten K-Means clustering for 3 clusters handwritten\_labels, initial\_centroids, final\_centroids = kmeans\_clustering(X, n\_clusters=3)  # Plot the handwritten K-Means clustering results with initial and final centroids plt.scatter(X[:, 0], X[:, 1], c=handwritten\_labels, cmap='viridis', edgecolor='k', s=50, label='Final Clusters') plt.scatter(initial\_centroids[:, 0], initial\_centroids[:, 1], c='blue', marker='o', s=200, label='Initial Centroids') plt.scatter(final\_centroids[:, 0], final\_centroids[:, 1], c='red', marker='X', s=200, label='Final Centroids') plt.title('Handwritten K-Means Clustering with Initial and Final Centroids') plt.legend() plt.show()  # Elbow method plot distortions = elbow\_method(X, max\_clusters=10) plt.plot(range(1, 11), distortions, marker='o') plt.title('Elbow Method for Optimal Number of Clusters') plt.xlabel('Number of Clusters') plt.ylabel('Distortion') plt.show()   * This section uses the previously defined functions to perform K-Means clustering, plot the clusters with initial and final centroids, and display the Elbow Method plot.   **Source Code:**  def elbow\_method(X, max\_clusters=10):  distortions = []  for i in range(1, max\_clusters + 1):  kmeans = KMeans(n\_clusters=i, random\_state=42, n\_init=10)  kmeans.fit(X)  distortions.append(kmeans.inertia\_)  return distortions  # Handwritten K-Means clustering for 3 clusters handwritten\_labels, initial\_centroids, final\_centroids = kmeans\_clustering(X, n\_clusters=3)  # Plot the handwritten K-Means clustering results with initial and final centroids plt.scatter(X[:, 0], X[:, 1], c=handwritten\_labels, cmap='viridis', edgecolor='k', s=50, label='Final Clusters') plt.scatter(initial\_centroids[:, 0], initial\_centroids[:, 1], c='blue', marker='o', s=200, label='Initial Centroids') plt.scatter(final\_centroids[:, 0], final\_centroids[:, 1], c='red', marker='X', s=200, label='Final Centroids') plt.title('Handwritten K-Means Clustering with Initial and Final Centroids') plt.legend() plt.show()  # Elbow method plot distortions = elbow\_method(X, max\_clusters=10) plt.plot(range(1, 11), distortions, marker='o') plt.title('Elbow Method for Optimal Number of Clusters') plt.xlabel('Number of Clusters') plt.ylabel('Distortion') plt.show()  **Result:**  C:\Users\Alex Joshua Chirwa\AppData\Local\Packages\Microsoft.Windows.Photos_8wekyb3d8bbwe\TempState\ShareServiceTempFolder\Screenshot (150).jpeg  C:\Users\Alex Joshua Chirwa\AppData\Local\Packages\Microsoft.Windows.Photos_8wekyb3d8bbwe\TempState\ShareServiceTempFolder\Screenshot (151).jpeg  **Sklearn K-Means Comparison**  The purpose of this comparison is to visually assess how well the handwritten K-Means algorithm performs in comparison to the well-established K-Means implementation provided by the sklearn toolkit.  The scatter plot allows for a side-by-side comparison of the clusters generated by both methods.  The use of different colors and markers for each set of clusters helps differentiate between the handwritten and sklearn-generated clusters.  In this section involves the following:   1. **Using Sklearn K-Means:**  * The ‘KMeans’ class from the sklearn toolkit is used to perform K-Means clustering on the same dataset (‘X’) * ‘n\_clusters=3’ specifies the number of times the algorithm will run with different centroid seeds. The best result is kept.  1. **Comparison Plot:**  * Scatter plots are created to visualize the clusters generated by both the handwritten K-Means algorithm and the Sklearn K-Means algorithm. * ‘plt.scatter’ is used to plot the data points with different colors for each cluster. * The ‘viridis’ colormap is used, and points are differentiated using the edgecolor and marker style. * The plot is titled “Comparison: Handwritten vs. Sklearn K-Means.” * A legend is added to distinguish between the two sets of clusters.   **Source Code:**  # Using K-Means clustering in sklearn toolkit kmeans\_sklearn = KMeans(n\_clusters=3, random\_state=42, n\_init=10).fit(X) sklearn\_labels = kmeans\_sklearn.labels\_  # Compare handwritten K-Means with sklearn toolkit plt.scatter(X[:, 0], X[:, 1], c=handwritten\_labels, cmap='viridis', edgecolor='k', s=50, label='Handwritten K-Means') plt.scatter(X[:, 0], X[:, 1], c=sklearn\_labels, cmap='viridis', marker='x', s=50, label='Sklearn K-Means') plt.title('Comparison: Handwritten vs. Sklearn K-Means') plt.legend() plt.show()  **Result:**  **C:\Users\Alex Joshua Chirwa\Pictures\Screenshots\Screenshot (152).png**  **Image Compression with K-Means**  #Image compression with K-Means clustering image = plt.imread("C:/Users/Alex Joshua Chirwa/Downloads/dog-gbfe9c6841\_1920\_2.jpg")  plt.figure(figsize=(5, 5)) plt.title('Original Image') plt.imshow(image) plt.axis('off') plt.show()  # Normalize the image to the [0, 1] range image\_normalized = image / 255.0  image\_flat = image\_normalized.reshape((-1, 3))  n\_colors = 16 image\_flat\_sample = shuffle(image\_flat, random\_state=42)[:1000] kmeans = KMeans(n\_clusters=n\_colors, random\_state=42) kmeans.fit(image\_flat\_sample)  labels = kmeans.predict(image\_flat)  # Create a compressed image using the cluster centers as colors image\_compressed = kmeans.cluster\_centers\_[labels].reshape(image.shape)  # Clip the values to the valid range [0, 1] image\_compressed\_clipped = np.clip(image\_compressed, 0, 1)  plt.figure(figsize=(5, 5)) plt.title('Compressed Image ({} colors)'.format(n\_colors)) plt.imshow(image\_compressed\_clipped) plt.axis('off') plt.show()  **Steps Implemented:**   1. **Load and Display the Original image:**  * The ‘plt.imread’ function is used to load an image from a specified file path. * The original image is displayed using ‘plt.imshow’. * The image is normalized to the [0,1] range to ensure consistent handling of pixel values.  1. **Image Flattening and K-Means Clustering:**  * The image is flattened into a one-dimensional array of RGB values. * A sample of 1000 pixels is randomly selected using ‘shuffle’ for computational efficiency. * K-Means clustering is applied to the sampled flattened image with a specified number of colors (‘n\_colors=16’). * The cluster labels are obtained using ‘predict’.  1. **Create Compressed image:**  * A compressed image is created by replacing the RGB values of each pixel with the corresponding centroid of its assigned cluster. * The reshaped image is formed using the original shape of the image.  1. **Clip Values and Display Compressed Image:**  * The compressed image is clipped to ensure that pixel values are within the valid range [0,1]. * The compressed image is displayed to visualize the effect of image compression.   **Results:**  C:\Users\Alex Joshua Chirwa\Pictures\Screenshots\Screenshot (160).png  **C:\Users\Alex Joshua Chirwa\Pictures\Screenshots\Screenshot (161).png**   * The original image is transformed into a compressed version by reducing the number of unique colors. * The number of colors (‘n\_colors=16’) determines the granularity of the compression. * A lower number of colors generally results in a loss of detail as represented of the original image, while a higher number of colors generally results in a more faithful representation.   **Conclusion**  In conclusion, K-Means clustering proves to be versatile and effective tool for image compression, offering a balance between reduced file size and preserved visual information. |