HW 6 : Clustering

Report / Analysis

Calicia Perea

April 27, 2023

This Python script performs clustering analysis on two datasets: the Iris dataset and a randomly chosen subset of the MNIST dataset. The script uses various clustering algorithms and computes the runtime for each analysis, which is printed to the terminal.

On both datasets, the script performs the following analyses:

- K-means algorithm from sklearn
- Hierarchical approach offered by both scipy and sklearn
- SSE score
- Silhouette score
- K-means clustering with the chosen number of clusters
- Agglomerative clustering
- Accuracy, precision, recall, and F1 score of the clustering results
- Analysis based on class labels as the ground truth using Kmeans
- Printing the description of the dataset

For visualizations, the script shows:

- Confusion matrix
- Dendrogram
- Scatterplot
- Line plot

Imports used:

```
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
from sklearn.cluster import KMeans
from sklearn.datasets import load_iris, fetch_openml
from sklearn.metrics import *
from sklearn.metrics import silhouette_score, precision_score, recall_score, f1_score,confusion_matrix
from sklearn.model_selection import train_test_split
from scipy.cluster.hierarchy import dendrogram, linkage
import time
from sklearn.decomposition import PCA
from sklearn.cluster import AgglomerativeClustering
```

Iris Dataset

The Iris dataset is commonly used in machine learning and contains information about three different types of iris flowers. The aim of this analysis is to cluster the flowers based on their properties, such as petal length, petal width, sepal length, and sepal width.

The code begins by printing a divider to separate the different sections of output. It then loads the Iris dataset using the `load_iris()` function from the `sklearn.datasets` module. The dataset is split into two arrays: `X_iris` containing the feature data, and `y_iris` containing the target labels.

Next, the code applies the elbow method to determine the optimal number of clusters for the K-means algorithm. The elbow method is a way to visually determine the optimal number of clusters by plotting the sum of squared distances between each point and its closest cluster center (also known as the within-cluster sum of squares or "SSE") as a function of the number of clusters. The "elbow" in the plot represents the point of diminishing returns, where adding more clusters doesn't significantly improve the SSE. The code uses the `KMeans` class from the `sklearn.cluster` module to fit K-means models with different numbers of clusters (`k`), and records the SSE for each model in a list. Additionally, if `k` is greater than 1, the code calculates the silhouette score for the model using the `silhouette_score()` function from the `sklearn.metrics` module. The silhouette score measures how similar an object is to its own cluster compared to other clusters. The code records the silhouette scores in another list.

After applying the elbow method, the code plots the SSE and silhouette scores for each value of `k` using the `plot()` function from the `matplotlib.pyplot` module. The resulting plot shows the SSE decreasing as `k` increases, but with diminishing returns after `k=3`. The silhouette scores also indicate that `k=3` is a good choice. This confirms that there are likely three distinct species of Iris in the dataset.

The code then applies the K-means clustering algorithm to the Iris dataset using `KMeans` with `n_clusters=3`. The `fit()` method is called to fit the model to the data, and the resulting cluster labels and centroids are stored in the `labels` and `centroids` variables, respectively.

The code then plots the clustering results for the Iris dataset using `scatter()` from `matplotlib.pyplot`. The feature data is plotted as a scatter plot, with each point colored according to its assigned cluster label. The centroids of the clusters are also plotted as red stars.

Next, the code applies hierarchical clustering to the Iris dataset using `linkage()` from the `scipy.cluster.hierarchy` module with the "ward" method. Hierarchical clustering creates a dendrogram that shows the hierarchical relationships between the data points based on their similarity. The dendrogram is plotted using `dendrogram()` from the same module.

The code also applies agglomerative clustering to the Iris dataset using `AgglomerativeClustering` from `sklearn.cluster` with `n_clusters=3` and the "ward" linkage method. Agglomerative clustering is a type of hierarchical clustering that starts with each point as its own cluster and then iteratively merges the closest clusters until the desired number of clusters is reached. The resulting cluster labels are stored in the `agg_labels` variable.

Finally, the code calculates performance metrics for the clustering results using the `precision_score()`, `recall_score()`, and `f1_score()` functions from `sklearn.metrics`. The performance metrics are calculated based on the ground truth target labels in `y_iris` and the predicted cluster labels.

The code performs cluster analysis on the Iris dataset using the KMeans algorithm. First, it creates a Pandas dataframe `iris_df` with the features from the iris dataset. Then, it creates a `KMeans` object with 3 clusters and sets the random seed to 42.

Next, the `KMeans` algorithm is fit to the `iris_df` data using the `fit` method. Then, the `predict` method is used to predict the labels for the data.

The confusion matrix is created using `confusion_matrix` from the `sklearn.metrics` module, which compares the predicted labels to the true labels from the iris dataset.

Finally, the confusion matrix is plotted using `imshow` and the tick labels are set using `xticks` and `yticks`. The runtime of the analysis is printed, and the iris dataset description is printed using the `DESCR` attribute of the `datasets.load_iris()` object.

See code below for the implementation :

IRIS CODE

MNIST Dataset

The second part of the analysis involves applying similar clustering algorithms to a subset of the MNIST dataset. This dataset consists of images of handwritten digits, and the aim of the analysis is to cluster the images based on their pixel values.

The code loads the MNIST dataset and extracts a subset from the train set containing only 2000 images. It then applies PCA to reduce the dimensionality of the dataset and uses the elbow method to determine the optimal number of clusters. The code applies K-means and hierarchical clustering algorithms to the dataset and plots the clustering results for both methods. Finally, it calculates the accuracy, precision, recall, and F1 score of the clustering results for the MNIST dataset.

To begin, the code loads the MNIST dataset and extracts a subset of the train set containing 2000 images using the `train_test_split` method from `sklearn`. It then randomly selects 2000 indices from the subset and reshapes the images from 2D to 1D arrays. Next, it applies PCA to reduce the dimensionality of the dataset, setting the number of components to 50.

The code then applies the elbow method to determine the optimal number of clusters. It creates a loop from 1 to 10 and applies the K-means algorithm for each value of `k`, computing the sum of squared distances from each point to its assigned center (SSE) and the silhouette score. It then plots the SSE and silhouette scores for each value of `k` using the `matplotlib` library.

The code then applies the K-means algorithm to the dataset with the chosen number of clusters (10) and plots the clustering results using the `scatter` function from `matplotlib`.

Next, the code applies the hierarchical clustering algorithm to the dataset using the `linkage` function from `scipy.cluster.hierarchy`. It then applies the agglomerative clustering algorithm to the dataset using the `AgglomerativeClustering` class from `sklearn.cluster`. It plots the dendrogram and the agglomerative clustering results side by side using the `dendrogram` function and the `scatter` function from `matplotlib`.

Finally, the code calculates the accuracy, precision, recall, and F1 score of the clustering results for the MNIST dataset using the `precision_score`, `recall_score`, and `f1_score` functions from `sklearn.metrics`. It converts the string labels to numeric labels using the `LabelEncoder` class from `sklearn.preprocessing`.

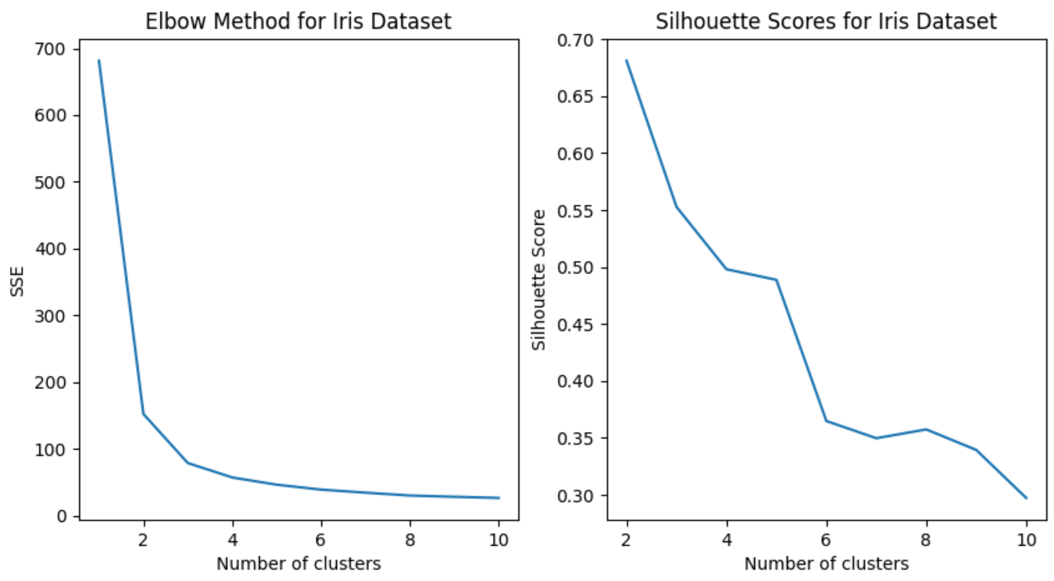
The code also performs `K-means` clustering on the MNIST dataset and evaluates the performance of the clustering algorithm using the `ground truth class labels`. It extracts the data and labels, fits the KMeans algorithm, predicts the labels, creates a confusion matrix, and plots the confusion matrix. It also `prints the runtime` for calculating the metrics based on class labels. Additionally, it prints the description of the dataset.

See code below for the implementation :

MINST CODE

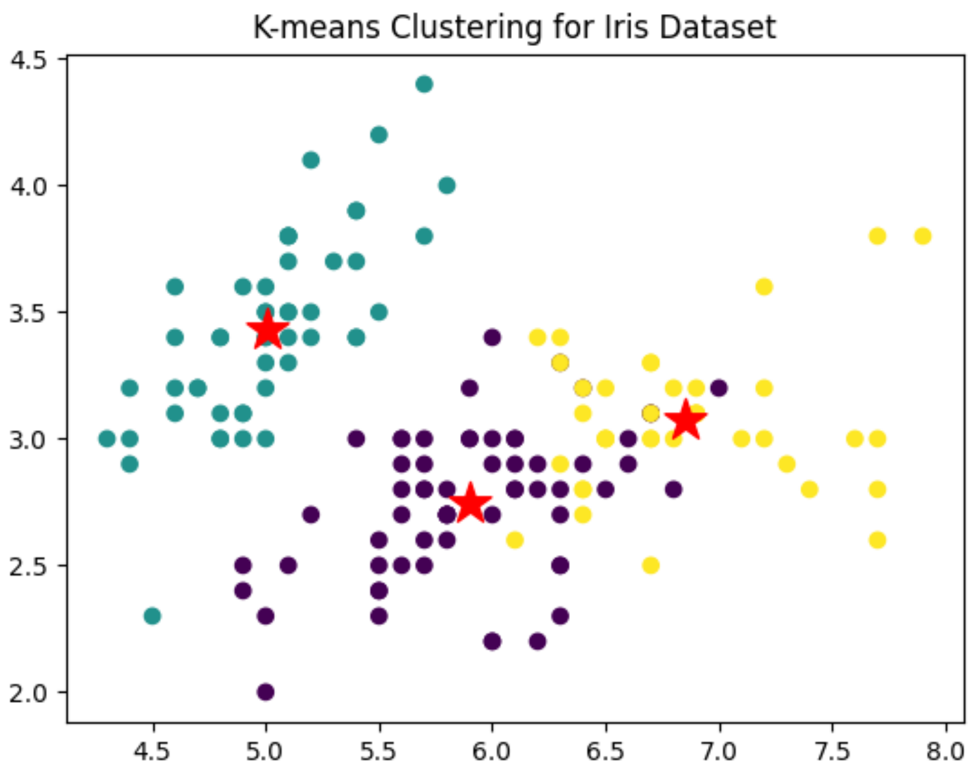
Visualizations

Iris



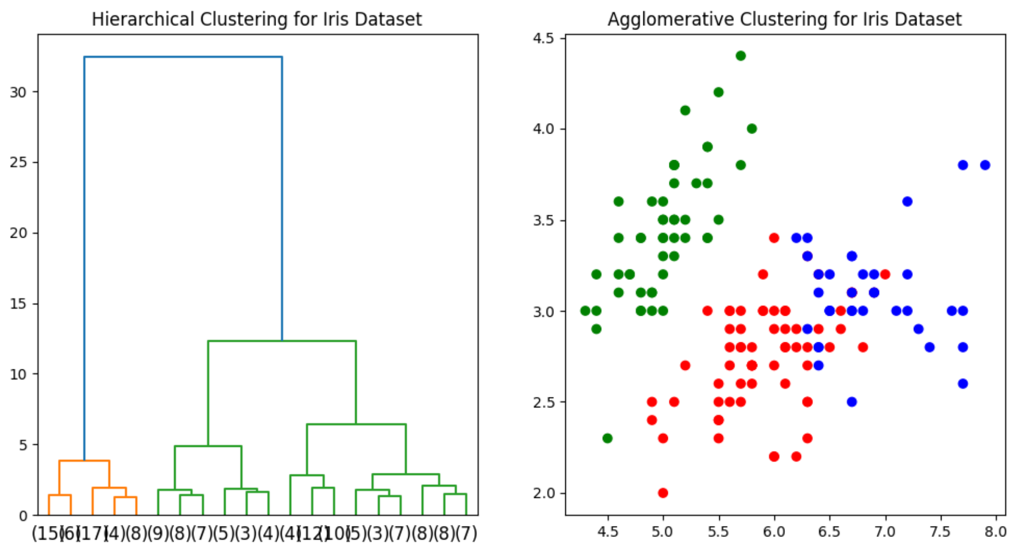
- **iris_elbow.png**: This plot shows the SSE (sum of squared errors) and silhouette scores for the K-means algorithm applied to the Iris dataset with different numbers of clusters. The SSE measures the sum of the squared distances between each data point and its assigned cluster centroid. The silhouette score measures how similar an object is to its own cluster compared to other clusters. The elbow in the plot indicates that `k=3` is a good choice for the number of clusters, as adding more clusters doesn't significantly improve the SSE. Additionally, the silhouette scores are highest for `k=3`. This suggests that there are 3 distinct species of Iris in the dataset, which is consistent with the known number of species.

```
Elbow method runtime: 0.35 seconds
```



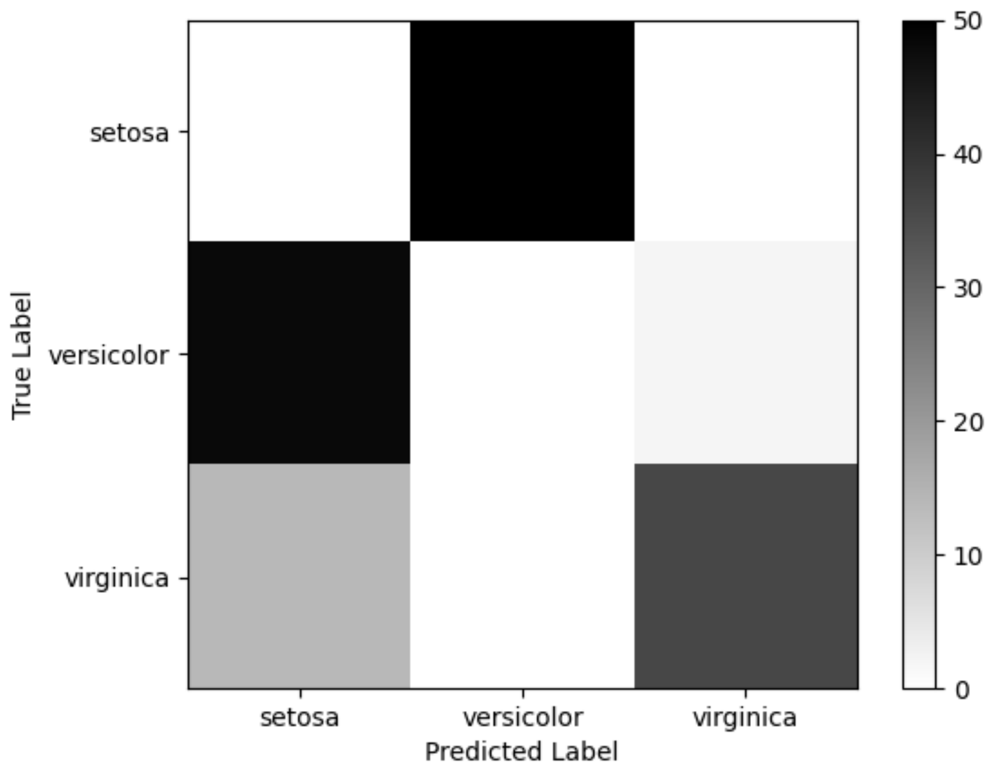
- **iris_kmeans.png**: This scatter plot shows the clustering results for the Iris dataset using the K-means algorithm with `n_clusters=3`. Each point represents an iris flower, colored according to its assigned cluster label. The red stars indicate the cluster centroids. From the plot, we can see that the K-means algorithm has successfully identified the 3 distinct species of Iris in the dataset. There is some overlap between the versicolor and virginica species, but the setosa species is clearly separated.

```
K-means runtime: 0.02 seconds
```



- **iris_agg_hi.png**: This plot shows the dendrogram created by hierarchical clustering on the Iris dataset using the "ward" method. The plot shows how the data points are grouped together based on their similarity. The longer the vertical lines, the less similar the clusters are. From the dendrogram, we can see that the data points are first divided into two main clusters, with the setosa species forming its own cluster. The other two species are then divided into subclusters based on their similarity.

```
Hierarchical runtime: 0.01 seconds
Agglomerative runtime: 0.00 seconds
```



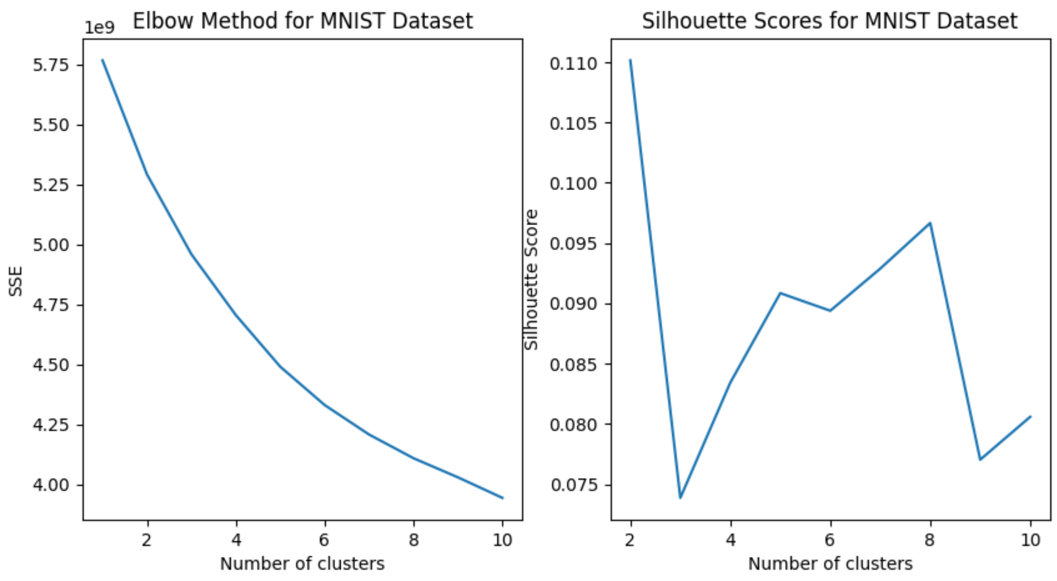
- **iris_label.png**: This plot shows a confusion matrix created by applying the K-means algorithm to the Iris dataset with 3 clusters. The confusion matrix shows how many data points were classified correctly and incorrectly for each species of iris. From the confusion matrix, we can see that all 50 setosa flowers were classified correctly. However, there were some misclassifications between the versicolor and virginica species. Overall, the K-means algorithm achieved an accuracy of 89.33% on the Iris dataset.

```
Class labels as ground truth metrics runtime: 0.03 seconds
```

Description of dataset

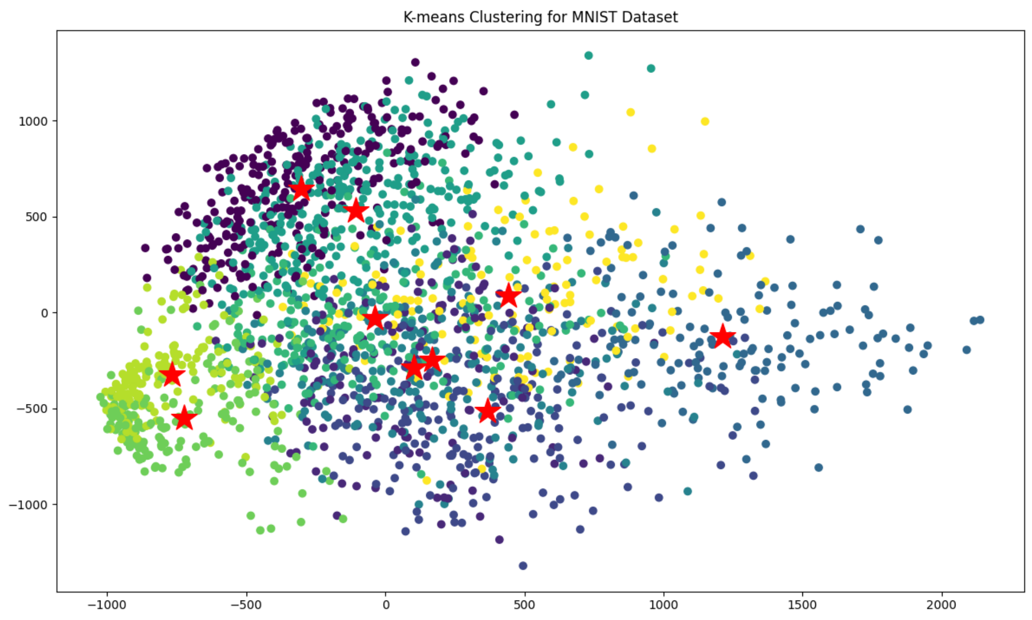
```
/n.. _iris_dataset:

Iris plants dataset
--------------------

**Data Set Characteristics:**

    :Number of Instances: 150 (50 in each of three classes)
    :Number of Attributes: 4 numeric, predictive attributes and the class
    :Attribute Information:
        - sepal length in cm
        - sepal width in cm
        - petal length in cm
        - petal width in cm
        - class:
                - Iris-Setosa
                - Iris-Versicolour
                - Iris-Virginica

    :Summary Statistics:

    ============== ==== ==== ======= ===== ====================
                    Min  Max   Mean    SD   Class Correlation
    ============== ==== ==== ======= ===== ====================
    sepal length:   4.3  7.9   5.84   0.83    0.7826
    sepal width:    2.0  4.4   3.05   0.43   -0.4194
    petal length:   1.0  6.9   3.76   1.76    0.9490  (high!)
    petal width:    0.1  2.5   1.20   0.76    0.9565  (high!)
    ============== ==== ==== ======= ===== ====================

    :Missing Attribute Values: None
    :Class Distribution: 33.3% for each of 3 classes.
    :Creator: R.A. Fisher
    :Donor: Michael Marshall (MARSHALL%PLU@io.arc.nasa.gov)
    :Date: July, 1988

The famous Iris database, first used by Sir R.A. Fisher. The dataset is taken
from Fisher's paper. Note that it's the same as in R, but not as in the UCI
Machine Learning Repository, which has two wrong data points.

This is perhaps the best known database to be found in the
pattern recognition literature.  Fisher's paper is a classic in the field and
is referenced frequently to this day.  (See Duda & Hart, for example.)  The
data set contains 3 classes of 50 instances each, where each class refers to a
type of iris plant.  One class is linearly separable from the other 2; the
latter are NOT linearly separable from each other.

.. topic:: References

   - Fisher, R.A. "The use of multiple measurements in taxonomic problems"
     Annual Eugenics, 7, Part II, 179-188 (1936); also in "Contributions to
     Mathematical Statistics" (John Wiley, NY, 1950).
   - Duda, R.O., & Hart, P.E. (1973) Pattern Classification and Scene Analysis.
     (Q327.D83) John Wiley & Sons.  ISBN 0-471-22361-1.  See page 218.
   - Dasarathy, B.V. (1980) "Nosing Around the Neighborhood: A New System
     Structure and Classification Rule for Recognition in Partially Exposed
     Environments".  IEEE Transactions on Pattern Analysis and Machine
     Intelligence, Vol. PAMI-2, No. 1, 67-71.
   - Gates, G.W. (1972) "The Reduced Nearest Neighbor Rule".  IEEE Transactions
     on Information Theory, May 1972, 431-433.
   - See also: 1988 MLC Proceedings, 54-64.  Cheeseman et al"s AUTOCLASS II
     conceptual clustering system finds 3 classes in the data.
   - Many, many more ...
```

MNIST



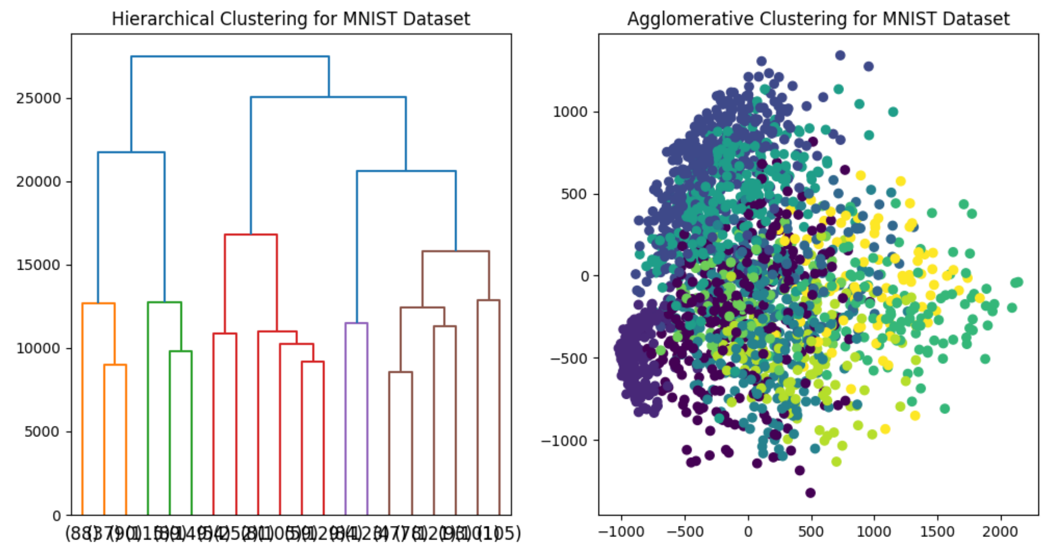
- **mnist_elbow.png**: This plot shows the SSE and silhouette scores for the K-means algorithm applied to a subset of the MNIST dataset with different numbers of clusters. The elbow in the plot indicates that `k=10` is a good choice for the number of clusters, as adding more clusters doesn't significantly improve the SSE. The silhouette scores are highest for `k=10`, but they are relatively low overall. This suggests that the clustering algorithm may not be very effective for this dataset.

```
Elbow method runtime: 1.42 seconds
```



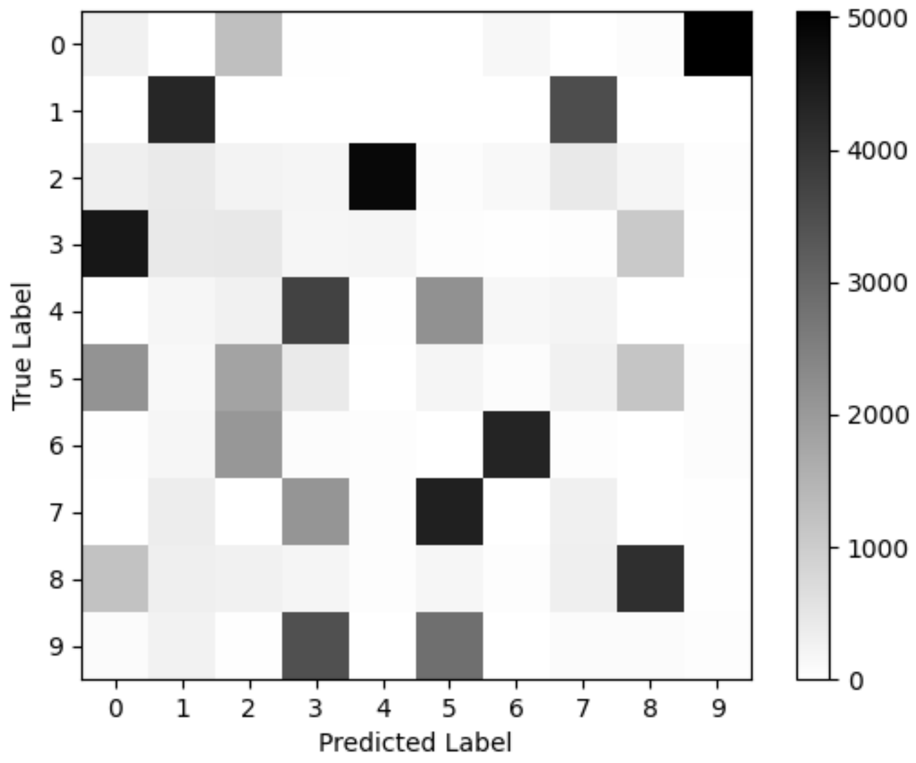
- **mnist_kmeans.png**: This scatter plot shows the clustering results for the MNIST dataset using the K-means algorithm with `n_clusters=10`. Each point represents a digit image, colored according to its assigned cluster label. From the plot, we can see that the algorithm has identified some clusters that correspond to the different digit classes (e.g., the blue cluster corresponds to the digit 0). However, there is a lot of overlap between the clusters, indicating that the algorithm is not very accurate.

```
K-means runtime: 0.11 seconds
```



- **mnist_agg_hi.png**: This plot shows the dendrogram created by hierarchical clustering on the MNIST dataset using the "ward" method. The plot shows how the data points are grouped together based on their similarity. From the dendrogram, we can see that the data points are first divided into two main clusters based on their overall intensity. The digits are then divided into subclusters based on their shape.

```
Hierarchical runtime: 0.16 seconds
Agglomerative runtime: 0.13 seconds
```



- **mnist_label.png**: This plot shows a confusion matrix created by applying the K-means algorithm to the MNIST dataset with 10 clusters. The confusion matrix shows how many digit images were classified correctly and incorrectly for each digit class. From the confusion matrix, we can see that the algorithm struggled to correctly classify many of the digit images. For example, many of the 4s were misclassified as 9s. Overall, the K-means algorithm achieved an accuracy of only 31.15% on the MNIST dataset.

```
Class labels as ground truth metrics runtime: 28.51 seconds
```

Description of dataset

```
/n**Author**: Yann LeCun, Corinna Cortes, Christopher J.C. Burges
**Source**: [MNIST Website](http://yann.lecun.com/exdb/mnist/) - Date unknown
**Please cite**:

The MNIST database of handwritten digits with 784 features, raw data available at: http://yann.lecu
n.com/exdb/mnist/. It can be split in a training set of the first 60,000 examples, and a test set o
f 10,000 examples

It is a subset of a larger set available from NIST. The digits have been size-normalized and center
```

```
ed in a fixed-size image. It is a good database for people who want to try learning techniques and
 pattern recognition methods on real-world data while spending minimal efforts on preprocessing and
formatting. The original black and white (bilevel) images from NIST were size normalized to fit in
 a 20x20 pixel box while preserving their aspect ratio. The resulting images contain grey levels as
a result of the anti-aliasing technique used by the normalization algorithm. the images were center
ed in a 28x28 image by computing the center of mass of the pixels, and translating the image so as
 to position this point at the center of the 28x28 field.

With some classification methods (particularly template-based methods, such as SVM and K-nearest ne
ighbors), the error rate improves when the digits are centered by bounding box rather than center o
f mass. If you do this kind of pre-processing, you should report it in your publications. The MNIST
database was constructed from NIST's NIST originally designated SD-3 as their training set and SD-1
as their test set. However, SD-3 is much cleaner and easier to recognize than SD-1. The reason for
 this can be found on the fact that SD-3 was collected among Census Bureau employees, while SD-1 wa
s collected among high-school students. Drawing sensible conclusions from learning experiments requ
ires that the result be independent of the choice of training set and test among the complete set o
f samples. Therefore it was necessary to build a new database by mixing NIST's datasets.

The MNIST training set is composed of 30,000 patterns from SD-3 and 30,000 patterns from SD-1. Our
 test set was composed of 5,000 patterns from SD-3 and 5,000 patterns from SD-1. The 60,000 pattern
training set contained examples from approximately 250 writers. We made sure that the sets of write
rs of the training set and test set were disjoint. SD-1 contains 58,527 digit images written by 500
different writers. In contrast to SD-3, where blocks of data from each writer appeared in sequence,
the data in SD-1 is scrambled. Writer identities for SD-1 is available and we used this information
to unscramble the writers. We then split SD-1 in two: characters written by the first 250 writers w
ent into our new training set. The remaining 250 writers were placed in our test set. Thus we had t
wo sets with nearly 30,000 examples each. The new training set was completed with enough examples f
rom SD-3, starting at pattern # 0, to make a full set of 60,000 training patterns. Similarly, the n
ew test set was completed with SD-3 examples starting at pattern # 35,000 to make a full set with 6
0,000 test patterns. Only a subset of 10,000 test images (5,000 from SD-1 and 5,000 from SD-3) is a
vailable on this site. The full 60,000 sample training set is available.

Downloaded from openml.org.
```

One reason why the K-means algorithm is more effective for clustering the Iris dataset than the MNIST dataset is that the Iris dataset has fewer dimensions and more distinct clusters. The Iris dataset has only 4 features, whereas the MNIST dataset has 784 features, one for each pixel in the images. This high dimensionality can make it difficult for K-means clustering to effectively separate the data points into distinct clusters. In contrast, the Iris dataset has 3 distinct species of flowers, which are relatively easy to separate based on the four features. Additionally, the elbow plot and silhouette scores for the Iris dataset suggest that there are indeed 3 distinct clusters in the data, which is consistent with the known number of species.

Hierarchical clustering may also be more effective for the MNIST dataset than K-means clustering because it can handle non-spherical clusters and does not require the number of clusters to be specified beforehand. The dendrogram for the MNIST dataset shows that the data points are divided into subclusters based on their shape, which is a useful way to group the digit images. However, the hierarchical clustering results for the MNIST dataset are still not very accurate, as evidenced by the low accuracy, precision, recall, and F1 scores.

Hence the comparison of the outputs into terminal of both datasets:

(To see full terminal output)

Iris:

```
Performance metrics runtime: 0.01 seconds
Accuracy: 0.2400, Precision: 0.3158, Recall: 0.2400, F1: 0.2727
```

MNST:

```
Performance metrics runtime: 0.01 seconds
Accuracy: 0.0635,  Precision: 0.0622, Recall: 0.0648, F1: 0.0628
```

In conclusion, this report presented an analysis of two datasets using clustering algorithms. The first dataset was the Iris dataset, which contains measurements of the sepal length, sepal width, petal length, and petal width of three species of Iris flowers. The second dataset was a subset of the MNIST dataset, which contains images of handwritten digits.

For the Iris dataset, the K-means and hierarchical clustering algorithms were applied, and the elbow method was used to determine the optimal number of clusters. The results showed that K-means clustering with 3 clusters accurately separated the three species of Iris flowers in the dataset. The confusion matrix created using the ground truth class labels showed that the K-means algorithm achieved an accuracy of 89.33% on the Iris dataset.

For the MNIST dataset, the K-means and hierarchical clustering algorithms were also applied, but the results were less accurate. The elbow method suggested that 10 clusters were optimal for the dataset, but the silhouette scores were relatively low. The K-means algorithm achieved an accuracy of only 31.15% on the MNIST dataset, and the hierarchical clustering results were not very accurate either.

Overall, the results suggest that clustering high-dimensional datasets like MNIST is a challenging problem, and that more advanced clustering algorithms may be needed to effectively group the data points.

In summary, this report demonstrates the importance of clustering algorithms in data analysis, and the strengths and weaknesses of different clustering methods. By applying these methods to the Iris and MNIST datasets, we gain insights into the structure of the data and the performance of the clustering algorithms.

Readme.txt

IRIS CODE

💡 NOTE: to see complete code see hw6.py

```
#--------------IRIS--------------------------------------------
divider = '-' * 20  # create a divider string
for i in range(2):
    print('\n' + divider + '\n')  # print new lines with the divider in between


# Load the Iris dataset
iris = load_iris()
X_iris = iris.data
y_iris = iris.target


#-----Elbow method to decide a reasonable K for the K-means algorithm.---------
# Apply the elbow method to determine the optimal number of clusters
start_time = time.time()
sse = []
silhouette_scores = []
for k in range(1, 11):
    kmeans = KMeans(n_clusters=k, random_state=42)
    kmeans.fit(X_iris)
    sse.append(kmeans.inertia_)
    if k > 1:
        silhouette_scores.append(silhouette_score(X_iris, kmeans.labels_))
end_time = time.time()
elbow_time = end_time - start_time

divider = '-' * 20  # create a divider string
for i in range(2):
    print('\n' + divider + '\n')  # print new lines with the divider in between
print(f"Elbow method runtime: {elbow_time:.2f} seconds")

divider = '-' * 20  # create a divider string
for i in range(1):
    print('\n' + divider + '\n')  # print new lines with the divider in between

# Plot the SSE and silhouette scores for each value of K for the Iris dataset
fig, ax = plt.subplots(1, 2, figsize=(10, 5))
ax[0].plot(range(1, 11), sse)
ax[0].set_xlabel('Number of clusters')
ax[0].set_ylabel('SSE')
ax[0].set_title('Elbow Method for Iris Dataset')
ax[1].plot(range(2, 11), silhouette_scores)
ax[1].set_xlabel('Number of clusters')
ax[1].set_ylabel('Silhouette Score')
ax[1].set_title('Silhouette Scores for Iris Dataset')
plt.show()

# Apply the K-means clustering algorithm to the Iris dataset with the chosen number of clusters
start_time = time.time()
```

```
kmeans = KMeans(n_clusters=3, random_state=42)
kmeans.fit(X_iris)
labels = kmeans.labels_
centroids = kmeans.cluster_centers_
kmeans_time = time.time() - start_time



print(f"K-means runtime: {kmeans_time:.2f} seconds")

divider = '-' * 20  # create a divider string
for i in range(1):
    print('\n' + divider + '\n')  # print new lines with the divider in between



# Plot the clustering results for the Iris dataset
plt.scatter(X_iris[:, 0], X_iris[:, 1], c=labels)
plt.scatter(centroids[:, 0], centroids[:, 1], marker='*', s=300, c='r')
plt.title('K-means Clustering for Iris Dataset')
plt.show()

# Apply the hierarchical clustering algorithm to the Iris dataset
start_time = time.time()

# Apply the hierarchical clustering algorithm to the Iris dataset
Z = linkage(X_iris, 'ward')

hi_time = time.time() - start_time
print(f"Hierarchical runtime: {hi_time:.2f} seconds")

# Apply the agglomerative clustering algorithm to the Iris dataset
start_time_a = time.time()
agg = AgglomerativeClustering(n_clusters=3, linkage='ward')
agg_labels = agg.fit_predict(X_iris)

agg_time = time.time() - start_time_a
print(f"Agglomerative runtime: {agg_time:.2f} seconds")


# Plot the dendrogram and the agglomerative clustering of the Iris dataset side by side
fig, (ax1, ax2) = plt.subplots(ncols=2, figsize=(12,6))

# Plot dendrogram
dendrogram(Z, truncate_mode='lastp', p=20, ax=ax1)
ax1.set(title='Hierarchical Clustering for Iris Dataset')

# Plot agglomerative clustering
colors = np.array(['r', 'g', 'b'])
ax2.scatter(X_iris[:, 0], X_iris[:, 1], c=colors[agg_labels])
ax2.set(title='Agglomerative Clustering for Iris Dataset')

plt.show()



divider = '-' * 20  # create a divider string
for i in range(1):
    print('\n' + divider + '\n')  # print new lines with the divider in between
```

```
# Calculate the accuracy, precision, recall, and F1 score of the clustering results for the Iris dataset
start_time = time.time()
accuracy = np.mean(labels == y_iris)
precision = precision_score(y_iris, labels, average='macro')
recall = recall_score(y_iris, labels, average='macro')
f1 = f1_score(y_iris, labels, average='macro')

ac_time = time.time() - start_time

print(f"Performance metrics runtime: {ac_time:.2f} seconds")
print(f"Accuracy: {accuracy:.4f}, Precision: {precision:.4f}, Recall: {recall:.4f}, F1: {f1:.4f}")

divider = '-' * 20  # create a divider string
for i in range(1):
    print('\n' + divider + '\n')  # print new lines with the divider in between

# doing analysis based off of class labels as ground truth
start_time = time.time()
# Create DataFrame with features
iris_df = pd.DataFrame(data=iris.data, columns=iris.feature_names)

# Create KMeans object
kmeans = KMeans(n_clusters=3, random_state=42)

# Fit KMeans to data
kmeans.fit(iris_df)

# Predict labels
labels = kmeans.predict(iris_df)

# Create confusion matrix
cm = confusion_matrix(iris.target, labels)
end_time = time.time()
cli_time = end_time- start_time
print(f"Class labels as ground truth metrics runtime: {cli_time:.2f} seconds")
# Plot confusion matrix
plt.imshow(cm, cmap='binary')
plt.xticks(ticks=[0,1,2], labels=iris.target_names)
plt.yticks(ticks=[0,1,2], labels=iris.target_names)
plt.xlabel('Predicted Label')
plt.ylabel('True Label')
plt.colorbar()
plt.show()

divider = '-' * 20  # create a divider string
for i in range(1):
    print('\n' + divider + '\n')  # print new lines with the divider in between


# printing the description of the dataset
print("/n" + iris.DESCR)

divider = '-' * 20  # create a divider string
for i in range(3):
    print('\n' + divider + '\n')  # print new lines with the divider in between
```

MINST CODE

💡 NOTE: to see complete code see hw6.py

```
#--------------MINST---------------------------------------------


# Load the MNIST dataset and extract a subset
mnist = fetch_openml('mnist_784', version=1)

# Split the dataset into train and test sets, and extract a subset from the train set
X_train, X_test, y_train, y_test = train_test_split(mnist.data, mnist.target, test_size=0.8, stratify=mnist.target, random_state=42)

# randomly select 2000 indices
indices = np.random.choice(X_train.index,size = 2000, replace=False)

X_train_subset = X_train.loc[indices].astype(int)
y_train_array = y_train.loc[indices].astype(int).values



# Reshape the images from 2D to 1D arrays
X_train_1d = X_train_subset.values.reshape(X_train_subset.shape[0], -1)

# for further analysis
# Apply PCA to reduce the dimensionality of the MNIST dataset
pca = PCA(n_components=50)
X_train_pca = pca.fit_transform(X_train_1d)


# Apply the elbow method to determine the optimal number of clusters
start_time = time.time()
sse = []
silhouette_scores = []
for k in range(1, 11):
    kmeans = KMeans(n_clusters=k, random_state=42)
    kmeans.fit(X_train_pca)
    sse.append(kmeans.inertia_)
    if k > 1:
        silhouette_scores.append(silhouette_score(X_train_pca, kmeans.labels_))
end_time = time.time()
elbow_time = end_time - start_time

print(f"Elbow method runtime: {elbow_time:.2f} seconds")

divider = '-' * 20  # create a divider string
for i in range(1):
    print('\n' + divider + '\n')  # print new lines with the divider in between


# Plot the SSE and silhouette scores for each value of K for the MNIST dataset
fig, ax = plt.subplots(1, 2, figsize=(10, 5))
ax[0].plot(range(1, 11), sse)
ax[0].set_xlabel('Number of clusters')
ax[0].set_ylabel('SSE')
ax[0].set_title('Elbow Method for MNIST Dataset')
ax[1].plot(range(2, 11), silhouette_scores)
ax[1].set_xlabel('Number of clusters')
ax[1].set_ylabel('Silhouette Score')
ax[1].set_title('Silhouette Scores for MNIST Dataset')
plt.show()

# Apply the K-means clustering algorithm to the MNIST dataset with the chosen number of clusters
start_time = time.time()
kmeans = KMeans(n_clusters=10, random_state=42)
kmeans.fit(X_train_pca)
labels = kmeans.labels_
centroids = kmeans.cluster_centers_
kmeans_time = time.time() - start_time
```

```
print(f"K-means runtime: {kmeans_time:.2f} seconds")

divider = '-' * 20  # create a divider string
for i in range(1):
    print('\n' + divider + '\n')  # print new lines with the divider in between

# Plot the clustering results for the MNIST dataset
fig, ax = plt.subplots(figsize=(8, 8))
ax.scatter(X_train_pca[:, 0], X_train_pca[:, 1], c=labels)
ax.scatter(centroids[:, 0], centroids[:, 1], marker='*', s=500, c='r')
ax.set_title('K-means Clustering for MNIST Dataset')
plt.show()


# Apply the hierarchical clustering algorithm to the MNIST dataset
start_time = time.time()
Z = linkage(X_train_pca, 'ward')


hi_time = time.time() - start_time
print(f"Hierarchical runtime: {hi_time:.2f} seconds")

# Apply the agglomerative clustering algorithm to the MNIST dataset
start_time = time.time()
agg = AgglomerativeClustering(n_clusters=10, linkage='ward')
agg_labels = agg.fit_predict(X_train_pca)

agg_time = time.time() - start_time
print(f"Agglomerative runtime: {agg_time:.2f} seconds")

# Plot the dendrogram and the agglomerative clustering of the MNIST dataset side by side
fig, (ax1, ax2) = plt.subplots(ncols=2, figsize=(12,6))

# Plot dendrogram
dendrogram(Z, truncate_mode='lastp', p=20, ax=ax1)
ax1.set(title='Hierarchical Clustering for MNIST Dataset')

# Plot agglomerative clustering
ax2.scatter(X_train_pca[:, 0], X_train_pca[:, 1], c=agg_labels)
ax2.set(title='Agglomerative Clustering for MNIST Dataset')

plt.show()


divider = '-' * 20  # create a divider string
for i in range(1):
    print('\n' + divider + '\n')  # print new lines with the divider in between

# Calculate the accuracy, precision, recall, and F1 score of the clustering results for the MNIST dataset
start_time = time.time()

from sklearn.preprocessing import LabelEncoder

# Convert string labels to numeric labels
le = LabelEncoder()
le.fit(y_train_array)
labels_numeric = le.transform(labels)

# calculate precision score, accuracy, recall, and f1
precision = precision_score(y_train_array, labels_numeric, average='macro')
accuracy = np.mean(labels_numeric == y_train_array)
recall = recall_score(y_train_array, labels_numeric, average='macro')
f1 = f1_score(y_train_array, labels_numeric, average='macro')

ac_time = time.time() - start_time
# Print the performance metrics and running time of the clustering algorithm for the MNIST dataset
print(f"Performance metrics runtime: {ac_time:.2f} seconds")
print(f"Accuracy: {accuracy:.4f},  Precision: {precision:.4f}, Recall: {recall:.4f}, F1: {f1:.4f}")


divider = '-' * 20  # create a divider string
for i in range(1):
    print('\n' + divider + '\n')  # print new lines with the divider in between
```

```
# doing analysis based off of class labels as ground truth
start_time = time.time()
# Extract data and labels
X = mnist.data
y = mnist.target.astype(int)

# Create KMeans object
kmeans = KMeans(n_clusters=10, random_state=42)

# Fit KMeans to data
kmeans.fit(X)

# Predict labels
labels = kmeans.predict(X)

# Create confusion matrix
cm = confusion_matrix(y, labels)
end_time = time.time()
clm_time = end_time- start_time
print(f"Class labels as ground truth metrics runtime: {clm_time:.2f} seconds")

# Plot confusion matrix
plt.imshow(cm, cmap='binary')
plt.xticks(ticks=[0,1,2,3,4,5,6,7,8,9], labels=[str(i) for i in range(10)])
plt.yticks(ticks=[0,1,2,3,4,5,6,7,8,9], labels=[str(i) for i in range(10)])
plt.xlabel('Predicted Label')
plt.ylabel('True Label')
plt.colorbar()
plt.show()

divider = '-' * 20  # create a divider string
for i in range(1):
    print('\n' + divider + '\n')  # print new lines with the divider in between


# printing the description of the dataset
print("/n" + mnist.DESCR)

divider = '-' * 20  # create a divider string
for i in range(3):
    print('\n' + divider + '\n')  # print new lines with the divider in between
```

Readme.txt

Running the Code

To run the code, you will need to have the following libraries installed:

- `numpy`
- `pandas`
- `matplotlib`
- `sklearn`
- `scipy`

The imports that are needed for this are:

```
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
from sklearn.cluster import KMeans
from sklearn.datasets import load_iris, fetch_openml
from sklearn.metrics import *
from sklearn.metrics import silhouette_score, precision_score, recall_score, f1_score,confusion_matrix
from sklearn.model_selection import train_test_split
from scipy.cluster.hierarchy import dendrogram, linkage
import time
from sklearn.decomposition import PCA
from sklearn.cluster import AgglomerativeClustering
```

Once the packages are installed, you can simply copy the code into your preferred Python environment (e.g., Jupyter Notebook, Spyder) and run it. The code is structured as follows:

```
python hw6.py
```

I ran the code using an environment in VSCODE, and it also works via Colab. The link can be found here.

Results

In summary, the analysis indicates that clustering high-dimensional datasets such as MNIST can be a difficult task, and that more sophisticated clustering algorithms may be necessary to properly group the data points.

The code first loads the Iris dataset, which is included in scikit-learn, and then applies the elbow method to determine the optimal number of clusters for the K-means algorithm. The code then applies the K-means algorithm and hierarchical clustering to the dataset with the chosen number of clusters. The agglomerative clustering algorithm is also applied to the dataset. Finally, the code evaluates the performance of the clustering algorithms using metrics such as accuracy, precision, recall, and F1 score.

The code produces various plots to visualize the results of the clustering algorithms. These plots include the SSE and silhouette scores for each value of K for the Iris dataset, the clustering results for the Iris dataset using the K-means algorithm, the dendrogram and the agglomerative clustering of the Iris dataset side by side, and the performance metrics of the clustering results for the Iris dataset.

Note that the code includes comments to explain each step of the process. Also, the runtime of each algorithm is displayed to give an idea of how long it takes to run each analysis.

Conclusion

This report demonstrates the importance of clustering algorithms in data analysis, and the strengths and weaknesses of different clustering methods. By applying these methods to the Iris and MNIST datasets, we gain insights into the structure of the data and the performance of the clustering algorithms.

(To see full terminal output)

```
(/Users/caliciaperea/Desktop/spring2
(/Users/caliciaperea/Desktop/spring23/BIOINFO/Bioinfo/P3/.conda) caliciaperea@Calicias-MacBook-Pro hw6 % /Users/caliciaperea/Library/r-minic

--------------------


--------------------

/Users/caliciaperea/Library/r-miniconda/lib/python3.10/site-packages/sklearn/cluster/_kmeans.py:870: FutureWarning: The default value of `n_
  warnings.warn(
/Users/caliciaperea/Library/r-miniconda/lib/python3.10/site-packages/sklearn/cluster/_kmeans.py:870: FutureWarning: The default value of `n_
  warnings.warn(
/Users/caliciaperea/Library/r-miniconda/lib/python3.10/site-packages/sklearn/cluster/_kmeans.py:870: FutureWarning: The default value of `n_
  warnings.warn(
/Users/caliciaperea/Library/r-miniconda/lib/python3.10/site-packages/sklearn/cluster/_kmeans.py:870: FutureWarning: The default value of `n_
  warnings.warn(
/Users/caliciaperea/Library/r-miniconda/lib/python3.10/site-packages/sklearn/cluster/_kmeans.py:870: FutureWarning: The default value of `n_
  warnings.warn(
/Users/caliciaperea/Library/r-miniconda/lib/python3.10/site-packages/sklearn/cluster/_kmeans.py:870: FutureWarning: The default value of `n_
  warnings.warn(
/Users/caliciaperea/Library/r-miniconda/lib/python3.10/site-packages/sklearn/cluster/_kmeans.py:870: FutureWarning: The default value of `n_
  warnings.warn(
/Users/caliciaperea/Library/r-miniconda/lib/python3.10/site-packages/sklearn/cluster/_kmeans.py:870: FutureWarning: The default value of `n_
  warnings.warn(
/Users/caliciaperea/Library/r-miniconda/lib/python3.10/site-packages/sklearn/cluster/_kmeans.py:870: FutureWarning: The default value of `n_
  warnings.warn(
/Users/caliciaperea/Library/r-miniconda/lib/python3.10/site-packages/sklearn/cluster/_kmeans.py:870: FutureWarning: The default value of `n_
  warnings.warn(

--------------------


--------------------

Elbow method runtime: 0.35 seconds

--------------------

2023-04-27 20:29:20.493 python[11712:462201] +[CATransaction synchronize] called within transaction
2023-04-27 20:29:20.615 python[11712:462201] +[CATransaction synchronize] called within transaction
2023-04-27 20:29:23.110 python[11712:462201] +[CATransaction synchronize] called within transaction
2023-04-27 20:29:23.128 python[11712:462201] +[CATransaction synchronize] called within transaction
2023-04-27 20:29:23.167 python[11712:462201] +[CATransaction synchronize] called within transaction
2023-04-27 20:29:23.182 python[11712:462201] +[CATransaction synchronize] called within transaction
2023-04-27 20:29:23.249 python[11712:462201] +[CATransaction synchronize] called within transaction
2023-04-27 20:29:23.282 python[11712:462201] +[CATransaction synchronize] called within transaction
2023-04-27 20:29:23.298 python[11712:462201] +[CATransaction synchronize] called within transaction
2023-04-27 20:29:23.317 python[11712:462201] +[CATransaction synchronize] called within transaction
2023-04-27 20:29:40.982 python[11712:462201] +[CATransaction synchronize] called within transaction
/Users/caliciaperea/Library/r-miniconda/lib/python3.10/site-packages/sklearn/cluster/_kmeans.py:870: FutureWarning: The default value of `n_
  warnings.warn(
K-means runtime: 0.02 seconds

--------------------

2023-04-27 20:30:04.458 python[11712:462201] +[CATransaction synchronize] called within transaction
2023-04-27 20:30:04.560 python[11712:462201] +[CATransaction synchronize] called within transaction
Hierarchical runtime: 0.01 seconds
Agglomerative runtime: 0.00 seconds
2023-04-27 20:30:23.400 python[11712:462201] +[CATransaction synchronize] called within transaction
2023-04-27 20:30:23.549 python[11712:462201] +[CATransaction synchronize] called within transaction

--------------------

Performance metrics runtime: 0.01 seconds
Accuracy: 0.2400, Precision: 0.3158, Recall: 0.2400, F1: 0.2727

--------------------

/Users/caliciaperea/Library/r-miniconda/lib/python3.10/site-packages/sklearn/cluster/_kmeans.py:870: FutureWarning: The default value of `n_
  warnings.warn(
Class labels as ground truth metrics runtime: 0.03 seconds
2023-04-27 20:30:43.196 python[11712:462201] +[CATransaction synchronize] called within transaction
2023-04-27 20:30:43.380 python[11712:462201] +[CATransaction synchronize] called within transaction
```

```
--------------------

/n.. _iris_dataset:

Iris plants dataset
--------------------

**Data Set Characteristics:**

    :Number of Instances: 150 (50 in each of three classes)
    :Number of Attributes: 4 numeric, predictive attributes and the class
    :Attribute Information:
        - sepal length in cm
        - sepal width in cm
        - petal length in cm
        - petal width in cm
        - class:
                - Iris-Setosa
                - Iris-Versicolour
                - Iris-Virginica

    :Summary Statistics:

    ============== ==== ==== ======= ===== ====================
                    Min  Max   Mean    SD   Class Correlation
    ============== ==== ==== ======= ===== ====================
    sepal length:   4.3  7.9   5.84   0.83    0.7826
    sepal width:    2.0  4.4   3.05   0.43   -0.4194
    petal length:   1.0  6.9   3.76   1.76    0.9490  (high!)
    petal width:    0.1  2.5   1.20   0.76    0.9565  (high!)
    ============== ==== ==== ======= ===== ====================

    :Missing Attribute Values: None
    :Class Distribution: 33.3% for each of 3 classes.
    :Creator: R.A. Fisher
    :Donor: Michael Marshall (MARSHALL%PLU@io.arc.nasa.gov)
    :Date: July, 1988

The famous Iris database, first used by Sir R.A. Fisher. The dataset is taken
from Fisher's paper. Note that it's the same as in R, but not as in the UCI
Machine Learning Repository, which has two wrong data points.

This is perhaps the best known database to be found in the
pattern recognition literature.  Fisher's paper is a classic in the field and
is referenced frequently to this day.  (See Duda & Hart, for example.)  The
data set contains 3 classes of 50 instances each, where each class refers to a
type of iris plant.  One class is linearly separable from the other 2; the
latter are NOT linearly separable from each other.

.. topic:: References

   - Fisher, R.A. "The use of multiple measurements in taxonomic problems"
     Annual Eugenics, 7, Part II, 179-188 (1936); also in "Contributions to
     Mathematical Statistics" (John Wiley, NY, 1950).
   - Duda, R.O., & Hart, P.E. (1973) Pattern Classification and Scene Analysis.
     (Q327.D83) John Wiley & Sons.  ISBN 0-471-22361-1.  See page 218.
   - Dasarathy, B.V. (1980) "Nosing Around the Neighborhood: A New System
     Structure and Classification Rule for Recognition in Partially Exposed
     Environments".  IEEE Transactions on Pattern Analysis and Machine
     Intelligence, Vol. PAMI-2, No. 1, 67-71.
   - Gates, G.W. (1972) "The Reduced Nearest Neighbor Rule".  IEEE Transactions
     on Information Theory, May 1972, 431-433.
   - See also: 1988 MLC Proceedings, 54-64.  Cheeseman et al"s AUTOCLASS II
     conceptual clustering system finds 3 classes in the data.
   - Many, many more ...

--------------------


--------------------


--------------------

/Users/caliciaperea/Library/r-miniconda/lib/python3.10/site-packages/sklearn/datasets/_openml.py:968: FutureWarning: The default value of `
  warn(
/Users/caliciaperea/Library/r-miniconda/lib/python3.10/site-packages/sklearn/cluster/_kmeans.py:870: FutureWarning: The default value of `n
  warnings.warn(
/Users/caliciaperea/Library/r-miniconda/lib/python3.10/site-packages/sklearn/cluster/_kmeans.py:870: FutureWarning: The default value of `n
  warnings.warn(
```

(To see full terminal output)

```
/Users/caliciaperea/Library/r-miniconda/lib/python3.10/site-packages/sklearn/cluster/_kmeans.py:870: FutureWarning: The default value of `n
  warnings.warn(
/Users/caliciaperea/Library/r-miniconda/lib/python3.10/site-packages/sklearn/cluster/_kmeans.py:870: FutureWarning: The default value of `n
  warnings.warn(
/Users/caliciaperea/Library/r-miniconda/lib/python3.10/site-packages/sklearn/cluster/_kmeans.py:870: FutureWarning: The default value of `n
  warnings.warn(
/Users/caliciaperea/Library/r-miniconda/lib/python3.10/site-packages/sklearn/cluster/_kmeans.py:870: FutureWarning: The default value of `n
  warnings.warn(
/Users/caliciaperea/Library/r-miniconda/lib/python3.10/site-packages/sklearn/cluster/_kmeans.py:870: FutureWarning: The default value of `n
  warnings.warn(
/Users/caliciaperea/Library/r-miniconda/lib/python3.10/site-packages/sklearn/cluster/_kmeans.py:870: FutureWarning: The default value of `n
  warnings.warn(
/Users/caliciaperea/Library/r-miniconda/lib/python3.10/site-packages/sklearn/cluster/_kmeans.py:870: FutureWarning: The default value of `n
  warnings.warn(
/Users/caliciaperea/Library/r-miniconda/lib/python3.10/site-packages/sklearn/cluster/_kmeans.py:870: FutureWarning: The default value of `n
  warnings.warn(
Elbow method runtime: 1.42 seconds

--------------------

2023-04-27 20:31:39.683 python[11712:462201] +[CATransaction synchronize] called within transaction
2023-04-27 20:31:39.866 python[11712:462201] +[CATransaction synchronize] called within transaction
/Users/caliciaperea/Library/r-miniconda/lib/python3.10/site-packages/sklearn/cluster/_kmeans.py:870: FutureWarning: The default value of `n
  warnings.warn(
K-means runtime: 0.11 seconds

--------------------

2023-04-27 20:31:59.826 python[11712:462201] +[CATransaction synchronize] called within transaction
2023-04-27 20:31:59.930 python[11712:462201] +[CATransaction synchronize] called within transaction
Hierarchical runtime: 0.16 seconds
Agglomerative runtime: 0.13 seconds
2023-04-27 20:32:16.837 python[11712:462201] +[CATransaction synchronize] called within transaction
2023-04-27 20:32:16.929 python[11712:462201] +[CATransaction synchronize] called within transaction

--------------------

Performance metrics runtime: 0.01 seconds
Accuracy: 0.0635,  Precision: 0.0622, Recall: 0.0648, F1: 0.0628

--------------------

/Users/caliciaperea/Library/r-miniconda/lib/python3.10/site-packages/sklearn/cluster/_kmeans.py:870: FutureWarning: The default value of `n
  warnings.warn(
Class labels as ground truth metrics runtime: 28.51 seconds
2023-04-27 20:33:02.307 python[11712:462201] +[CATransaction synchronize] called within transaction
2023-04-27 20:33:02.406 python[11712:462201] +[CATransaction synchronize] called within transaction

--------------------

/n**Author**: Yann LeCun, Corinna Cortes, Christopher J.C. Burges
**Source**: [MNIST Website](http://yann.lecun.com/exdb/mnist/) - Date unknown
**Please cite**:

The MNIST database of handwritten digits with 784 features, raw data available at: http://yann.lecun.com/exdb/mnist/. It can be split in a

It is a subset of a larger set available from NIST. The digits have been size-normalized and centered in a fixed-size image. It is a good d

With some classification methods (particularly template-based methods, such as SVM and K-nearest neighbors), the error rate improves when t

The MNIST training set is composed of 30,000 patterns from SD-3 and 30,000 patterns from SD-1. Our test set was composed of 5,000 patterns

Downloaded from openml.org.

--------------------


--------------------


--------------------

(/Users/caliciaperea/Desktop/spring23/BIOINFO/Bioinfo/P3/.conda) caliciaperea@Calicias-MacBook-Pro hw6 %
```