Project Report: Identifying Groups of Similar Wines

Objective

The primary goal of this project is to utilize machine learning techniques to cluster wines based on key attributes, such as alcohol content, acidity, and color intensity. By implementing a clustering algorithm, the objective is to group similar wines, providing valuable insights for stakeholders in the wine industry, such as producers, distributors, and consumers, to make data-driven decisions regarding selection, pairing, and marketing.

Introduction

The wine industry is rich in diversity, with numerous varieties exhibiting unique characteristics. Understanding these differences is essential for producers, distributors, and consumers alike. In this project, I wrote code that utilizes machine learning techniques to cluster wines based on their attributes, such as alcohol content, acidity, and color intensity. By implementing a clustering algorithm, the aim was to identify groups of similar wines, providing valuable insights into their relationships. This approach offers a data-driven perspective that can enhance the exploration and appreciation of wine varieties, helping stakeholders make informed decisions about selection, pairing, and marketing.

Processes

The implementation of the project involved several key processes to ensure accurate clustering of the wines.

Class: Matrix

The Matrix class is central to our implementation and includes several key methods:

- **__init__(filename, standardize=True)**
  - **Purpose**: Initializes the class by loading data from a specified CSV file into a 2D NumPy array (array_2d). If the standardize parameter is set to True, the data is standardized upon loading. This is essential for ensuring that different features contribute equally to the distance calculations in clustering, allowing for a more accurate grouping of wines based on their characteristics.
- **load_from_csv(filename)**
  - **Purpose**: Reads a CSV file and loads its data into array_2d. This function enables the software to dynamically read wine data from an external source, facilitating easy updates to the dataset

without changing the code.
  - **Parameters:**
    - filename: The path to the CSV file.
  - **Return Value**: A NumPy array containing the data.
- **standardize()**
  - **Purpose**: Standardizes the array_2d data to make it more comparable by adjusting the values to have a mean of 0 and a standard deviation of 1. This preprocessing step is crucial in clustering algorithms, as it ensures that features with larger ranges do not dominate the distance calculations, thereby improving the overall performance and accuracy of the clustering results.
  - **Return Value**: None. The operation modifies the data in place.
- **get_distance(other_matrix, row_index)**
  - **Purpose**: Calculates the Euclidean distance between a specified row of the current matrix and all rows of another matrix. This function is fundamental for the clustering process, as it determines how similar or different two wines are based on their characteristics. By computing distances, the algorithm can identify which wines should be grouped together.
  - **Parameters**:
    - other_matrix: An instance of Matrix.
    - row_index: The index of the row in the current matrix.
  - **Return Value**: A column matrix of distances.
- **get_weighted_distance(other_matrix, weights, row_index)**
  - **Purpose**: Computes the weighted Euclidean distance between a specified row and all rows of another matrix using given weights. This function allows for the incorporation of feature importance in the distance calculations, enabling the algorithm to prioritize certain attributes over others based on their relevance to the clustering task.
  - **Parameters**:
    - other_matrix: An instance of Matrix.
    - weights: A Matrix containing the weights.
    - row_index: The index of the row in the current matrix.
  - **Return Value**: A column matrix of weighted distances.

Standalone Functions

The Code includes several standalone functions that operate outside the Matrix class:

- **get_count_frequency(S)**
  Purpose: Counts the frequency of unique values in S. This function is useful for analyzing the distribution of wines across different clusters, providing insights into how many wines belong to each group. Understanding these frequencies can help assess the effectiveness of the clustering process and identify any potential imbalances.
  Parameters: None (operates on S).
  Return Value: A dictionary mapping each unique element to its frequency count, or 0 if S has more than one column.
- **get_initial_weights(num_columns)**
  Purpose: Generates random weights that sum to one. This function is vital for initializing the clustering process, providing a starting point for the weights that will influence the distance calculations and ultimately affect the clustering outcome. Random weights help ensure that the algorithm explores different clustering configurations.
  Parameters: num_columns: The number of columns for the weights matrix.
  Return Value: A matrix with one row and num_columns that sums to one.

- **get_centroids(data, S, K)**
  Purpose: Computes the centroids of the clusters based on the current assignments. Centroids represent the average position of all the data points in a cluster, serving as a reference point for future distance calculations. This function is key to refining cluster assignments and enhancing the accuracy of the clustering process.
  Parameters:
  data: An instance of Matrix containing the data.
  S: The current group assignments.
  K: The number of clusters.
  Return Value: A matrix containing the centroids of the clusters.
- **get_separation_within(data, centroids, S, K)**
  Purpose: Calculates the separation within each cluster, quantifying how closely packed the data points are around their respective centroids. A low separation within a cluster indicates that the points are similar to each other, while a high separation suggests diversity within the cluster. This function is important for assessing the quality of the clusters formed.
  Parameters:
  data: An instance of Matrix containing the data.
  centroids: The centroids of the clusters.
  S: The current group assignments.
  K: The number of clusters.
  Return Value: A matrix containing the separation within clusters.
- **get_separation_between(data, centroids, S, K)**
  Purpose: Calculates the separation between different clusters, measuring how distinct the clusters are from one another. This function helps evaluate the effectiveness of the clustering algorithm; a larger separation between clusters typically indicates better-defined groupings.
  Parameters:
  data: An instance of Matrix containing the data.
  centroids: The centroids of the clusters.
  S: The current group assignments.
  K: The number of clusters.
  Return Value: A matrix containing the separation between clusters.
- **get_groups(data, K)**
  Purpose: Assigns each data point to a group based on the nearest centroid. This function is critical for determining the final cluster assignments and enables the algorithm to categorize wines into specific groups based on their characteristics.
  Parameters:
  data: An instance of Matrix containing the data.
  K: The number of groups to create.
  Return Value: A matrix S containing group assignments.
- **get_new_weights(data, centroids, old_weights, S, K)**
  Purpose: Updates the weights based on the spread of clusters. This function plays a crucial role in refining the clustering process by adjusting the influence of features based on how well they help distinguish between clusters. The new weights help improve the clustering results over successive iterations.
  Parameters:
  data: An instance of Matrix.
  centroids: The centroids of the clusters.
  old_weights: The current weights matrix.

S: The current group assignments.
K: The number of clusters.
Return Value: A new weights matrix.

- **run_test()**
  Purpose: Runs tests to evaluate the clustering algorithm for different values of K. This function allows for systematic evaluation of the algorithm's performance by clustering the data for varying numbers of clusters, helping to determine the optimal K that yields the best results in terms of separation and grouping.
  Operation: Loads data from Data.csv, performs clustering for K values ranging from 2 to 10, and prints the frequency of group assignments.

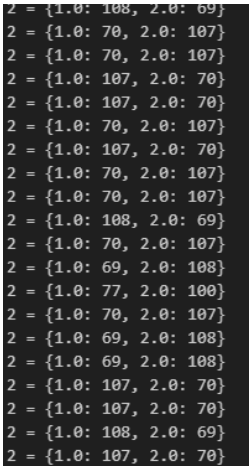
Evaluation

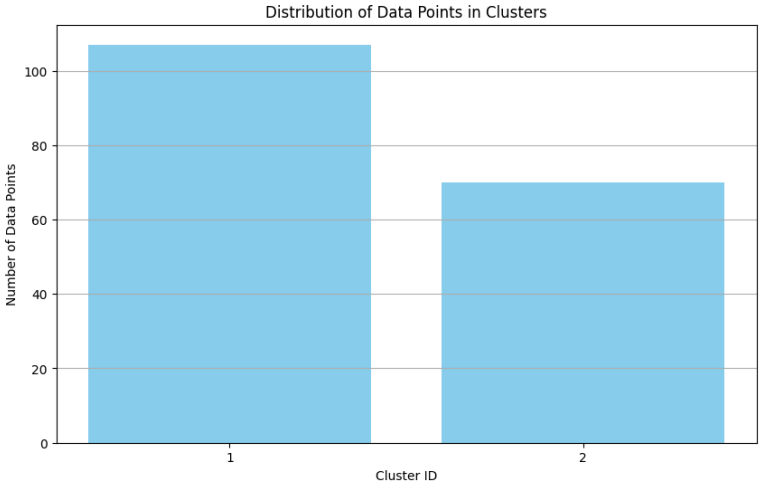
To evaluate the clustering results, I used two main criteria:

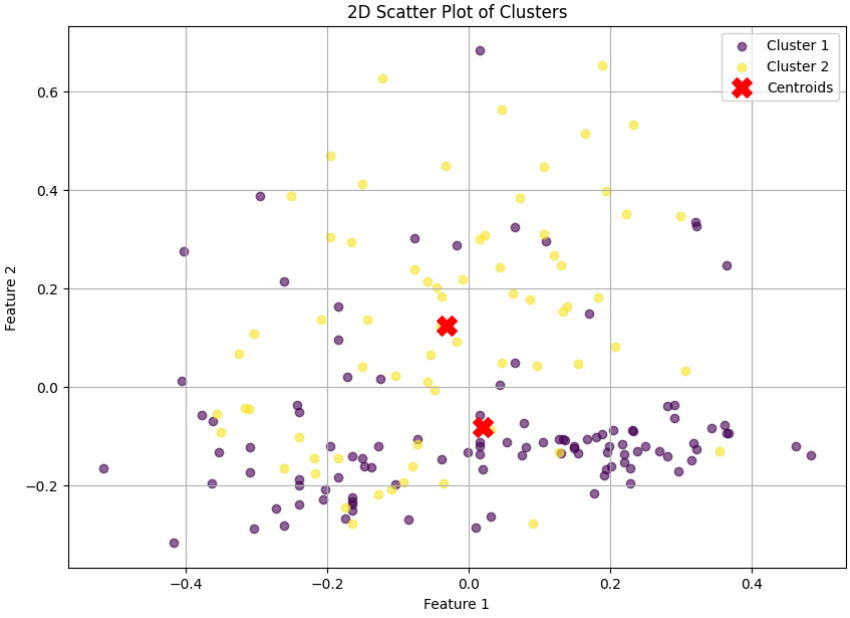
1. **Separation Within Clusters:**
   - This metric measures how close the wines within a cluster are to each other. Low separation within a cluster indicates high similarity, which is desirable for well-formed clusters.
2. **Separation Between Clusters:**
   - This metric assesses how distinct each cluster is from the others. Higher separation between clusters means the clusters are well-separated, indicating better-defined groups of wines.

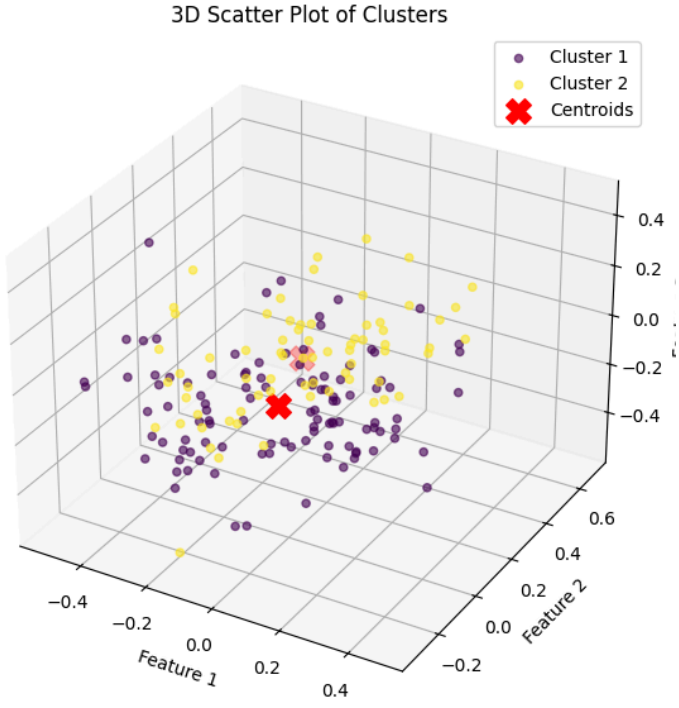
The code was tested for different values of K (number of clusters), ranging from 2 to 10. The optimal number of clusters was determined based on these evaluations, with a focus on maximizing inter-cluster distance and minimizing intra-cluster distance.

Visual Result of the code:




Distribution of Data Points in Clusters
Number of Data Points
Cluster ID
0
20
40
60
80
100
1
2


2D Scatter Plot of Clusters
Cluster 1
Cluster 2
Centroids
Feature 1
Feature 2
−0.4
−0.2
0.0
0.2
0.4
−0.2
0.0
0.2
0.4
0.6


3D Scatter Plot of Clusters
Cluster 1
Cluster 2
Centroids
Feature 1
Feature 2
−0.4
−0.2
0.0
0.2
0.4
−0.2
0.0
0.2
0.4
0.6
0.4
0.2
0.0
−0.2
−0.4

Applications

The clustering results from this project have several potential applications in the wine industry:

- **Wine Pairing:** Sommeliers can use the grouping of wines to recommend pairings that complement each other in flavor profiles and characteristics.
- **Product Development:** Wine producers can use the clustering insights to better understand product segmentation and create targeted wine varieties.
- **Marketing Strategies:** Retailers can group similar wines to create marketing campaigns and recommendations for customers based on clusters.

Conclusion

The code I wrote successfully implements a clustering algorithm to group similar wines based on their attributes. By utilizing a structured approach with the Matrix class and various standalone functions, the code can load data, standardize it, calculate distances, and group wines effectively. The inclusion of frequency counting allows for meaningful analysis of the clustering results, providing insights into the distribution of wines across different groups.

Github Link:

Github Link