Fundamentals of Smart Systems MP4

Students:
HamidReza Eslami (40115563)
Ali Kashipazha (40121723)

Instructor:
Dr. Aliyari

Google Colab Links:

Google Colab Link for Question 1: Link
Google Colab Link for Question 2: Link

GitHub Repository:
Link

February 3, 2026

Technical Report for Course FIS4041

Students: 40121723 and 40115563

February 3, 2026

Contents

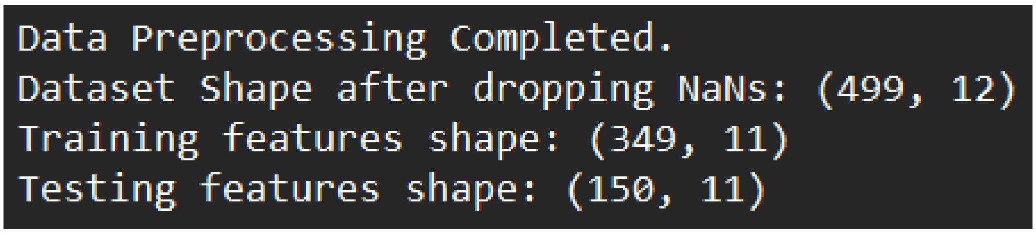
1 Question 1: Feature Selection with Evolutionary Algorithms

This section explores the use of evolutionary algorithms—specifically Particle Swarm Optimization (PSO) and a Genetic Algorithm (GA)—to perform feature selection. The goal is to identify the most impactful subset of features in a dataset to improve model performance and reduce complexity.

1.1 Part A: Data Preprocessing

The initial step involved loading and cleaning the `loan_train.csv` dataset. The following actions were taken to prepare the data for the models:

- **Handling Missing Data:** All rows containing missing (NaN) values were removed. This reduced the dataset from 615 rows to 499 rows.
- **Normalizing Column Values:** In the 'Dependents' column, the string value `'3+'` was converted to the integer 3 to ensure the column was purely numerical.
- **Encoding Categorical Variables:** Machine learning models require numerical input. Therefore, nominal columns such as `Gender, Married, Education, Self_Employed, Area,` and `Status` were converted into integer representations using a `LabelEncoder`.
- **Data Splitting:** The dataset was split into a training set (70% of the data) and a testing set (30% of the data) to evaluate the model's performance on unseen data.



```
Data Preprocessing Completed.
Dataset Shape after dropping NaNs: (499, 12)
Training features shape: (349, 11)
Testing features shape: (150, 11)
```

Figure 1: Output of the data preprocessing stage.

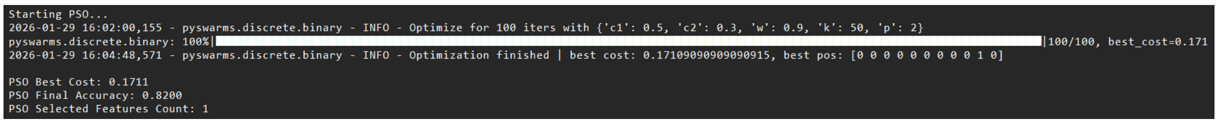
1.2 Part B: Implementing Particle Swarm Optimization (PSO)

PSO is a computational method that optimizes a problem by iteratively trying to improve a candidate solution with regard to a given measure of quality. It solves a problem by having a population of candidate solutions, here dubbed "particles," and moving these particles around in the search-space according to simple mathematical formulae over the particle's position and velocity.

Fitness Function: The objective was to minimize a cost function that balances model accuracy and the number of selected features. The function is defined as:

$$\text{Cost} = \alpha \times (1 - \text{Accuracy}) + (1 - \alpha) \times \frac{N_{\text{selected}}}{N_{\text{total}}}$$

Here, α was set to 0.9, placing a high importance on maximizing accuracy. The second term penalizes models with a large number of features, promoting simplicity. A `RandomForestClassifier` was used to evaluate the accuracy for each selected feature subset.



```
Starting PSO...
2026-01-29 16:02:00,155 - pyswarms.discrete.binary - INFO - Optimize for 100 iters with {'c1': 0.5, 'c2': 0.3, 'w': 0.9, 'k': 50, 'p': 2}
pyswarms.discrete.binary: 100%|██████████████████████████████████████████████████|100/100, best_cost=0.171
2026-01-29 16:04:48,571 - pyswarms.discrete.binary - INFO - Optimization finished | best cost: 0.17109090909090915, best pos: [0 0 0 0 0 0 0 0 0 1 0]

PSO Best Cost: 0.1711
PSO Final Accuracy: 0.8200
PSO Selected Features Count: 1
```

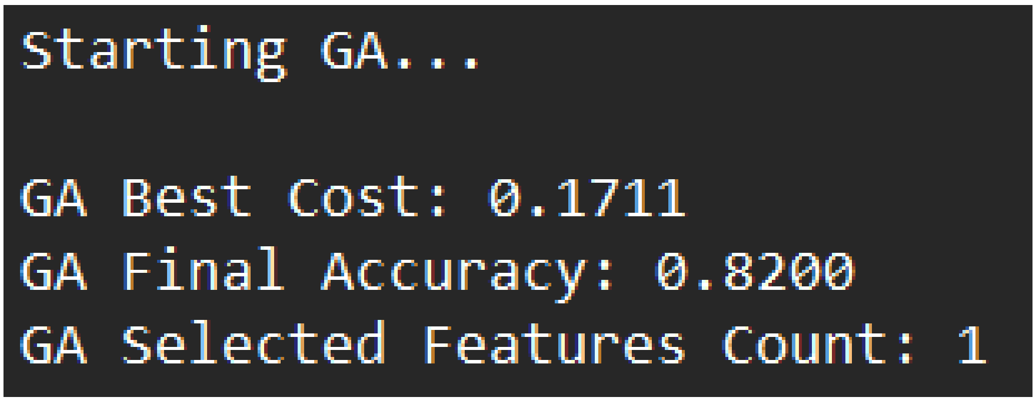
Figure 2: Output from the Particle Swarm Optimization (PSO) execution.

1.3 Part C: Implementing a Genetic Algorithm (GA)

A Genetic Algorithm is a search heuristic inspired by Charles Darwin's theory of natural evolution. This algorithm reflects the process of natural selection where the fittest individuals are selected for reproduction in order to produce offspring of the next generation.

Operators:

- **Selection:** Tournament selection (size=3) was used to choose parent chromosomes.
- **Crossover:** A two-point crossover method was applied.
- **Mutation:** A bit-flip mutation was used, where each gene had a 5% chance of being flipped.

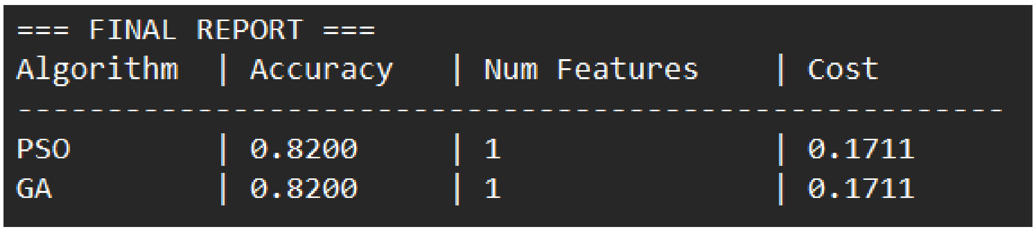


```
Starting GA...

GA Best Cost: 0.1711
GA Final Accuracy: 0.8200
GA Selected Features Count: 1
```

Figure 3: Output from the Genetic Algorithm (GA) execution.

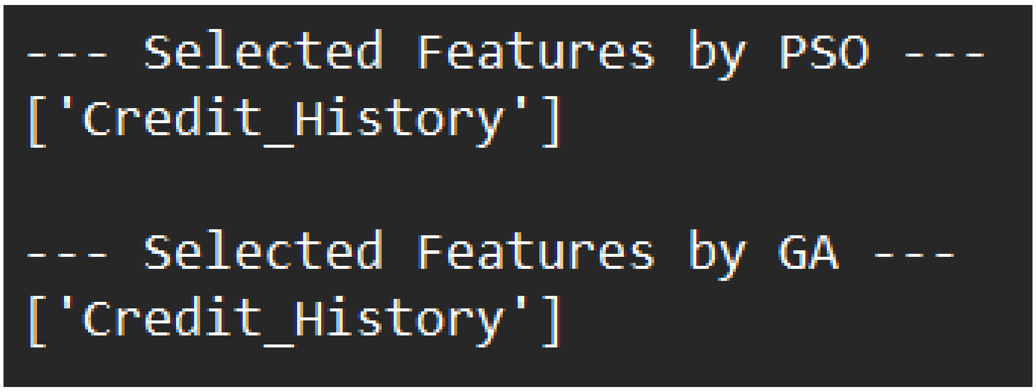
1.4 Part D: Quantitative Results and Comparison

Analysis: Both algorithms converged to the exact same optimal solution. They achieved an accuracy of 82% by selecting only a single key feature, successfully demonstrating the elimination of 10 redundant or less impactful features. This highlights the power of feature selection in reducing model complexity without sacrificing performance.



```
=== FINAL REPORT ===
Algorithm  | Accuracy   | Num Features    | Cost
-------------------------------------------------------
PSO        | 0.8200     | 1               | 0.1711
GA         | 0.8200     | 1               | 0.1711
```

Figure 4: Final performance comparison between PSO and GA.



```
--- Selected Features by PSO ---
['Credit_History']

--- Selected Features by GA ---
['Credit_History']
```

Figure 5: The single feature selected by both PSO and GA.

1.5 Part E: Analysis of the Selected Feature

Reasoning for Selection: In loan application datasets, an applicant's `Credit_History` is almost always the strongest predictor of loan approval status. Both algorithms correctly identified this. They determined that adding other features (like income or education) did not significantly increase accuracy but did increase the model's cost due to the penalty term $(1 - \alpha)$ in the fitness function. Therefore, the single-feature model was correctly identified as the most optimal solution.

1.6 Part F: Convergence and Speed Analysis

Analysis: The Genetic Algorithm (red dashed line) converged extremely quickly, finding the minimum cost within the first few generations. In contrast, the PSO algorithm (blue line) converged in a stepwise manner, taking longer to reach the same optimal solution. For this specific problem, the GA was more computationally efficient at finding the best feature subset.

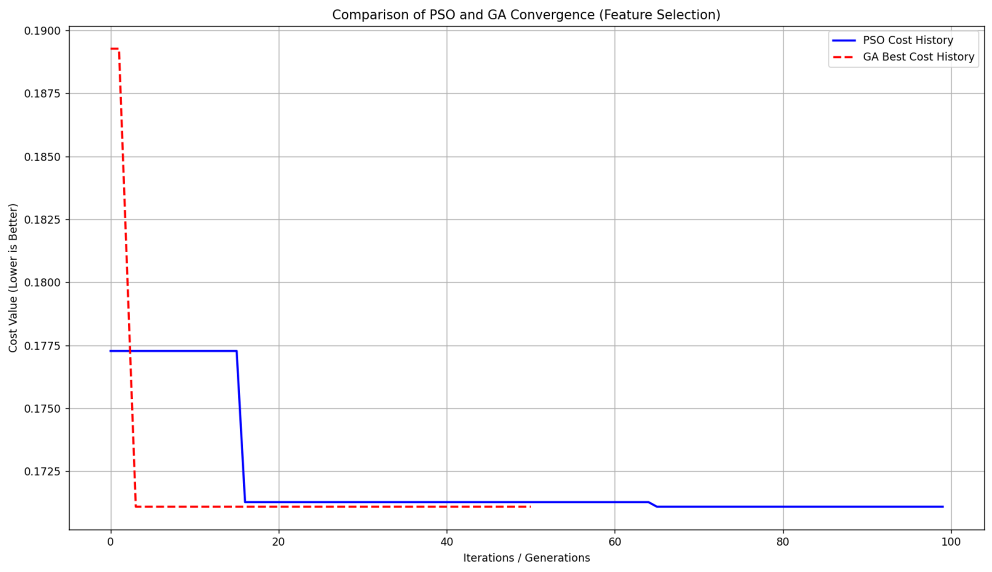


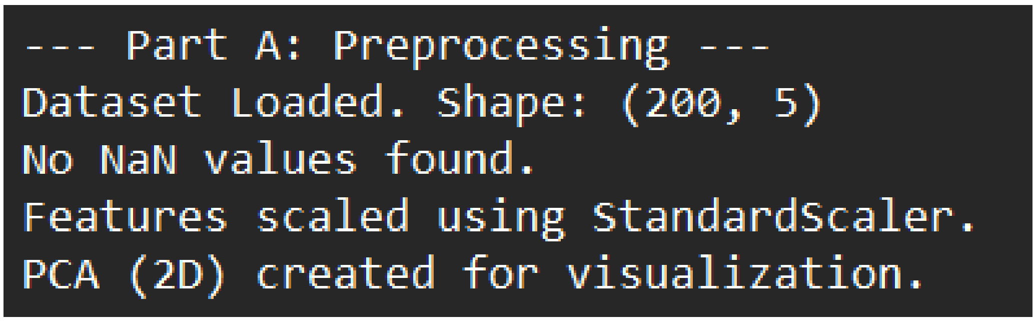
Figure 6: Convergence plot comparing the speed of PSO and GA.

2 Question 2: Clustering Algorithm Comparison

This section implements and compares three common clustering algorithms—K-Means, Agglomerative Clustering, and DBSCAN—on a dataset of mall customers to identify natural groupings within the data.

2.1 Part A: Data Preprocessing

- **Feature Selection:** The numerical features `Age`, `Annual Income`, and `Spending Score` were selected for clustering.
- **Standardization:** Since the features have different scales, a `StandardScaler` was used to normalize the data, ensuring that each feature contributes equally to the distance calculations.
- **Dimensionality Reduction for Visualization:** Principal Component Analysis (PCA) was used to reduce the 3D data to a 2D plane. This step is solely for visualization purposes, allowing us to plot and visually inspect the clusters.



```
--- Part A: Preprocessing ---
Dataset Loaded. Shape: (200, 5)
No NaN values found.
Features scaled using StandardScaler.
PCA (2D) created for visualization.
```

Figure 7: Output of the customer data preprocessing stage.

2.2 Part B: K-Means Algorithm

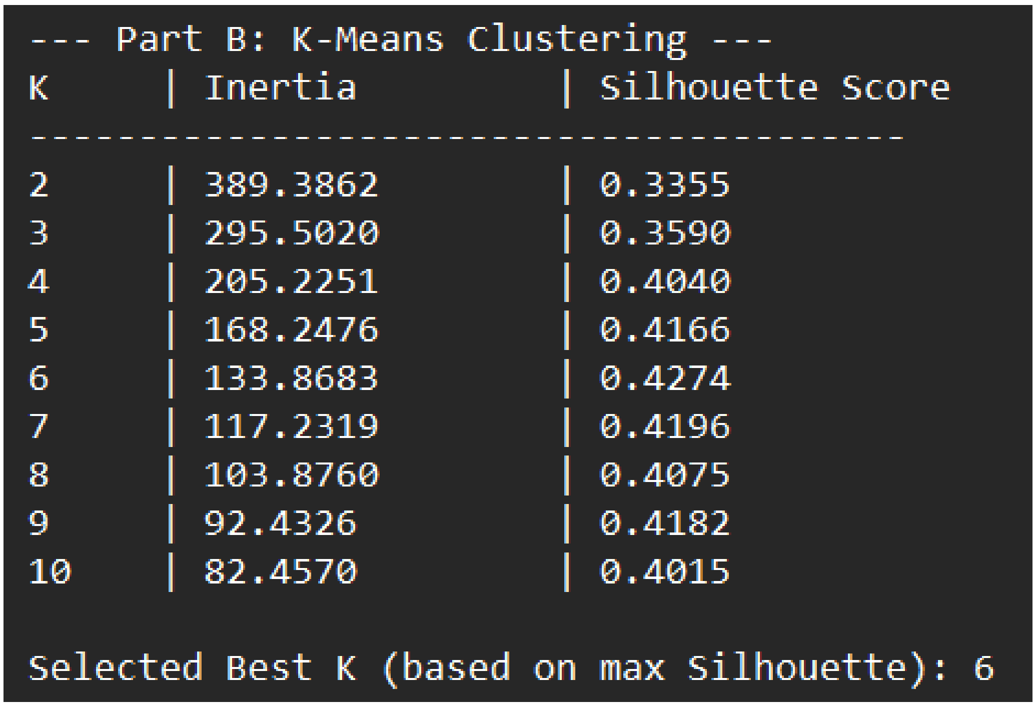
K-Means was run for K values from 2 to 10. To choose the best K, the Silhouette Score was used. This score measures how similar an object is to its own cluster compared to other clusters. A higher score indicates better-defined clusters.

Analysis: The highest Silhouette Score (0.4274) was achieved with $K = 6$. This suggests that the data has six distinct, natural groupings of customers.

2.3 Part C: Agglomerative Clustering

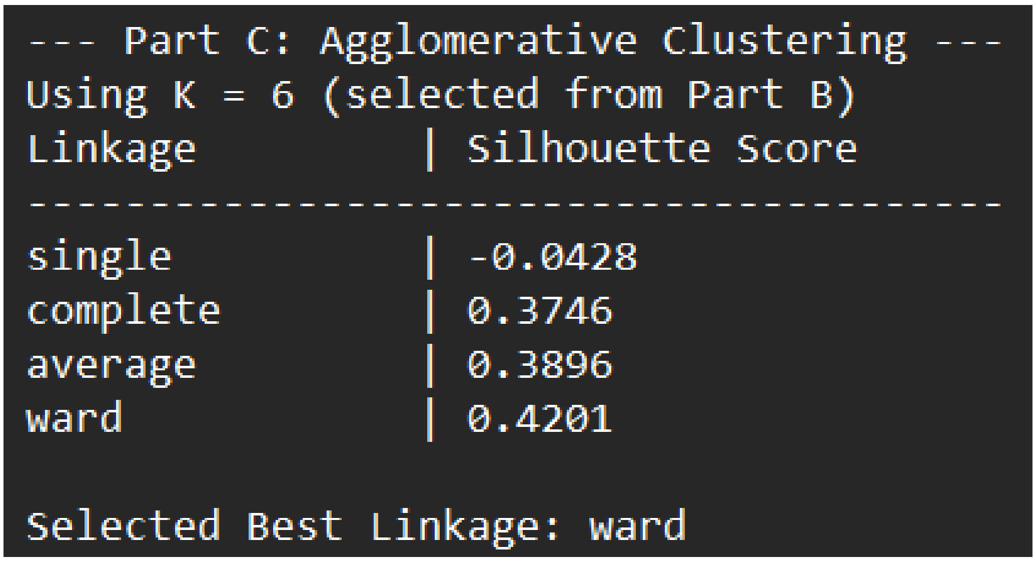
This hierarchical algorithm was run with the number of clusters fixed at 6. Different linkage criteria were tested.

Analysis: The 'Ward' linkage method yielded the best Silhouette Score. This method aims to minimize the variance within each cluster, and its strong performance suggests that the underlying clusters are relatively spherical, similar to what K-Means assumes.



```
--- Part B: K-Means Clustering ---
K     | Inertia          | Silhouette Score
----------------------------------------
2     | 389.3862         | 0.3355
3     | 295.5020         | 0.3590
4     | 205.2251         | 0.4040
5     | 168.2476         | 0.4166
6     | 133.8683         | 0.4274
7     | 117.2319         | 0.4196
8     | 103.8760         | 0.4075
9     | 92.4326          | 0.4182
10    | 82.4570          | 0.4015

Selected Best K (based on max Silhouette): 6
```

Figure 8: K-Means results for different values of K.



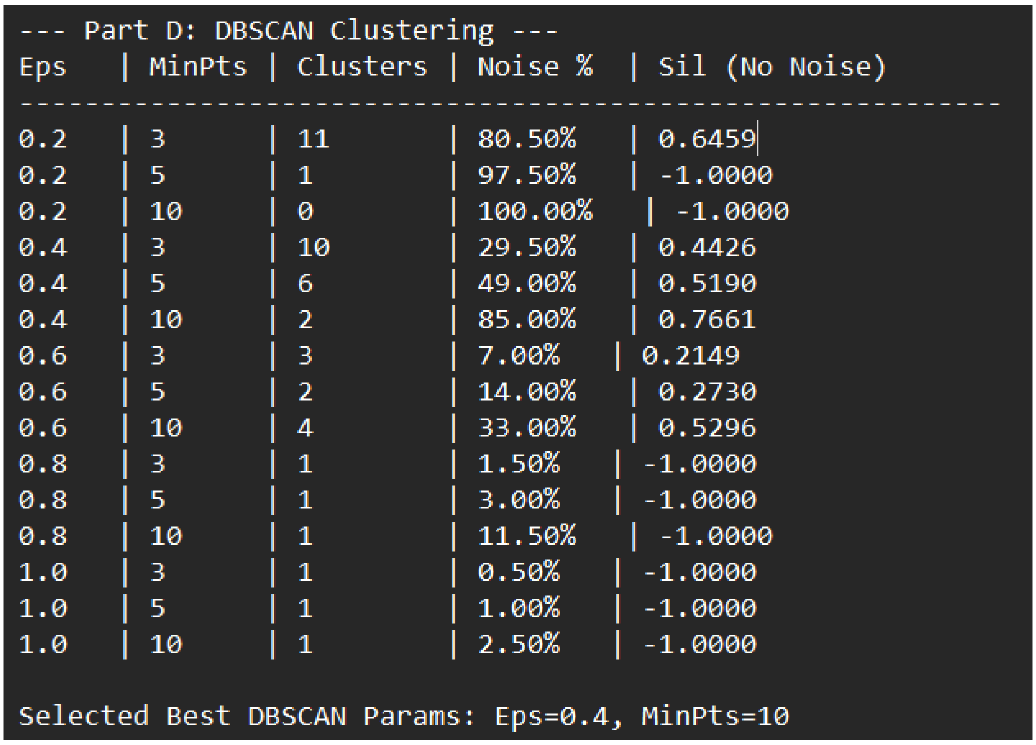
```
--- Part C: Agglomerative Clustering ---
Using K = 6 (selected from Part B)
Linkage         | Silhouette Score
----------------------------------------
single          | -0.0428
complete        | 0.3746
average         | 0.3896
ward            | 0.4201

Selected Best Linkage: ward
```

Figure 9: Comparison of linkage methods for Agglomerative Clustering.

2.4 Part D: DBSCAN Algorithm

DBSCAN (Density-Based Spatial Clustering of Applications with Noise) groups together points that are closely packed together, marking as outliers points that lie alone in low-density regions.

Analysis: DBSCAN performed poorly on this dataset. Although one parameter combination yielded a high Silhouette Score, it classified 85% of the data as "noise." This happens because DBSCAN struggles when clusters have significantly different densities, which appears to be the case here.



```
--- Part D: DBSCAN Clustering ---
Eps    | MinPts | Clusters | Noise %  | Sil (No Noise)
------------------------------------------------------------
0.2    | 3      | 11       | 80.50%   | 0.6459
0.2    | 5      | 1        | 97.50%   | -1.0000
0.2    | 10     | 0        | 100.00%   | -1.0000
0.4    | 3      | 10       | 29.50%   | 0.4426
0.4    | 5      | 6        | 49.00%   | 0.5190
0.4    | 10     | 2        | 85.00%   | 0.7661
0.6    | 3      | 3        | 7.00%   | 0.2149
0.6    | 5      | 2        | 14.00%   | 0.2730
0.6    | 10     | 4        | 33.00%   | 0.5296
0.8    | 3      | 1        | 1.50%   | -1.0000
0.8    | 5      | 1        | 3.00%   | -1.0000
0.8    | 10     | 1        | 11.50%   | -1.0000
1.0    | 3      | 1        | 0.50%   | -1.0000
1.0    | 5      | 1        | 1.00%   | -1.0000
1.0    | 10     | 1        | 2.50%   | -1.0000

Selected Best DBSCAN Params: Eps=0.4, MinPts=10
```

Figure 10: Grid search results for DBSCAN parameters.

2.5 Part E: Final Analysis and Visualization

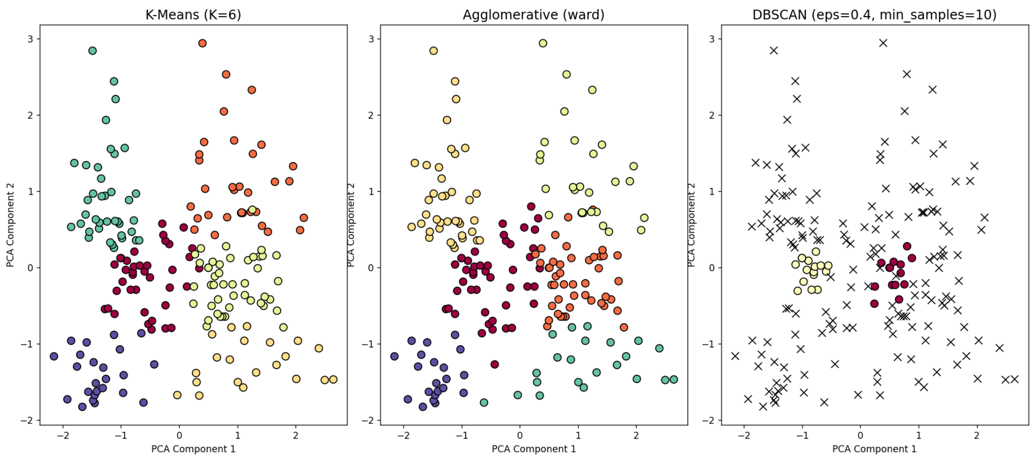


Figure 11: Visual comparison of the final clustering results.

Conclusion: Both K-Means (with K=6) and Agglomerative Clustering (with Ward linkage) produced high-quality, interpretable clusters. Their outputs are visually very similar, confirming the presence of six distinct customer segments. DBSCAN was not suitable for this dataset due to its variable density.

3 Question 3: Q-learning Concepts

3.1 Part A: Update Equation, Parameters, and Off-Policy Nature

3.1.1 1. The Q-learning Update Equation

The core of Q-learning is its update rule, which refines the value of taking an action 'a' in a state 's':

$$Q(s_t, a_t) \leftarrow (1-\alpha)Q(s_t, a_t) + \alpha \left[r_t + \gamma \max_a Q(s_{t+1}, a) \right]$$

3.1.2 2. Role of the Parameters

- α **(Learning Rate):** Determines how much new information overrides old information. A high value favors recent information.
- γ **(Discount Factor):** Determines the importance of future rewards. A value near 1 gives high importance to long-term rewards.
- ϵ **(in ϵ-greedy):** Manages the trade-off between exploration (trying new actions) and exploitation (choosing the best-known action).

3.1.3 3. Why Q-learning is an "Off-Policy" Algorithm

An algorithm is "off-policy" if the policy it learns about (the "target policy") is different from the policy it uses to act (the "behavior policy").

- **Behavior Policy:** The agent often acts using an ϵ-greedy policy, meaning it sometimes explores by choosing a random action.
- **Target Policy:** However, the update rule uses the term $\max_a Q(s_{t+1}, a)$. This part of the formula assumes the agent will act greedily (i.e., choose the absolute best action) in the next state, regardless of what action the ϵ-greedy behavior policy might actually choose.

Because the algorithm learns about the optimal (greedy) policy while behaving in a potentially non-optimal (exploratory) way, it is considered off-policy.

3.2 Part B: Numerical Problem

Given: $\alpha = 0.2, \gamma = 0.9, r_t = +2, \max_a Q(s_1, a) = 1.5$, and the old value $Q_{\text{old}}(s_0, a_1) = 0$.

Goal: Calculate the new value, $Q_{\text{new}}(s_0, a_1)$.

$$Q_{\text{new}}(s_0, a_1) = (1-0.2) \times 0 + 0.2\,[2 + 0.9 \times 1.5]$$

$$= 0.2\,[2 + 1.35]$$

$$= 0.2 \times 3.35$$

$$= 0.67$$

Final Answer: The new Q-value is **0.67**.

3.3 Part C: Instability Factors and Solutions

3.3.1 1. Factor: Fixed Exploration Rate

- **Problem:** Using a constant, high ϵ value causes the agent to keep taking random actions even after it has learned the optimal policy. This leads to unstable performance.
- **Solution: Epsilon Decay.** Start with a high ϵ (e.g., 1.0) to encourage exploration, and gradually decrease it over time to a small value (e.g., 0.01). This allows the agent to exploit its knowledge more as it becomes more confident.

3.3.2 2. Factor: Large or Unscaled Rewards

- **Problem:** If rewards are very large or have high variance, the Q-value updates will be massive and erratic, making it difficult for the model to converge to a stable solution.
- **Solution: Reward Clipping or Normalization.** A common practice is to "clip" all rewards to a fixed range, such as

  $$-1, +1$$

  . This prevents single, large reward events from destabilizing the entire learning process and promotes more stable convergence.