Students:

Ali Soltani_40119403

Seyed mohammadreza hosseini_40117463

Professor: Dr. Aliyari

Course:

Fundamental of Intelligent Systems

Project Links:

Google Colab:

Question 1:

Question 2:

Github:

Date: 1404/11/22

Contents

1 Introduction

1.1 Project Overview

This report documents two distinct machine learning projects that address fundamental challenges in predictive modeling and unsupervised learning. Part I focuses on feature selection optimization using evolutionary algorithms for loan approval prediction, while Part II implements customer segmentation through clustering techniques. Both projects emphasize reproducible research practices, comparative analysis, and practical interpretation of results.

1.2 Objectives

The primary objectives of this work are:

- Implement and compare evolutionary algorithms for automated feature selection
- Evaluate the effectiveness of PSO and GA in identifying optimal feature subsets
- Apply multiple clustering algorithms to discover customer segments
- Perform comprehensive comparative analysis of algorithmic performance
- Provide actionable insights from model results

1.3 Report Structure

This report is organized into two major parts corresponding to the two projects, each containing dedicated sections for methodology, implementation, results, and analysis.

Part I

Feature Selection using Evolutionary Algorithms(Q1)

2 Problem Statement and Dataset

2.1 Problem Definition

The objective of Part I is to predict loan approval status based on applicant characteristics while simultaneously identifying the most informative feature subset. This dual optimization problem requires balancing model accuracy against feature complexity to avoid overfitting and improve model interpretability.

2.2 Dataset Description

The loan approval dataset contains 614 records with 12 attributes describing loan applicants:

Table 1: Dataset Features and Descriptions

Feature	**Type**	**Description**
Gender	Categorical	Applicant's gender (Male/Female)
Married	Categorical	Marital status (Yes/No)
Dependents	Categorical	Number of dependents (0, 1, 2, 3+)
Education	Categorical	Education level (Graduate/Not Graduate)
Self_Employed	Categorical	Self-employment status (Yes/No)
Applicant_Income	Numerical	Applicant's annual income
Coapplicant_Income	Numerical	Co-applicant's annual income
Loan_Amount	Numerical	Requested loan amount
Term	Numerical	Loan term in months
Credit_History	Binary	Credit history (1.0/0.0)
Area	Categorical	Property area (Urban/Rural/Semiurban)
Status	Binary	Loan approval status (Y/N) - Target

2.3 Data Quality Issues

The original dataset contained 614 records with missing values across multiple features, requiring comprehensive preprocessing before model training.

3 Methodology

3.1 Overall Workflow

The complete workflow for Part I consists of five sequential stages:

1. **Data Preprocessing**: Loading, cleaning, and encoding categorical variables
2. **Feature Selection with PSO**: Implementing particle swarm optimization for feature subset selection
3. **Feature Selection with GA**: Implementing genetic algorithm for feature subset selection
4. **Comparative Analysis**: Evaluating and comparing both algorithms
5. **Convergence Analysis**: Examining optimization trajectories

3.2 Data Preprocessing

3.2.1 Missing Value Handling

Missing values were removed using complete case analysis, reducing the dataset from 614 to 499 records. This approach was selected due to:

- Relatively small proportion of missing data (18.7%)
- Simplicity and avoidance of imputation bias
- Sufficient remaining samples for robust training

3.2.2 Categorical Encoding

All categorical variables were transformed using Label Encoding via scikit-learn's `LabelEncoder`. This encoding maps each unique category to an integer value, making the data compatible with the Random Forest classifier.

Implementation:

```
encoder = LabelEncoder()
for col in categorical_columns:
    df[col] = encoder.fit_transform(df[col])
```

3.2.3 Train-Test Split

The dataset was partitioned into training (80%) and testing (20%) sets using stratified sampling to maintain class distribution:

```
X_train, X_test, y_train, y_test = train_test_split(
    X, y, test_size=0.2, random_state=42, stratify=y
)
```

Final data dimensions:

- Training set: 399 samples
- Testing set: 100 samples
- Number of features: 11

3.3 Feature Selection Framework

3.3.1 Objective Function

Both evolutionary algorithms optimize the same cost function that balances classification accuracy and feature subset size:

$$\text{Cost} = (1 - \text{Accuracy}) + \alpha \times \frac{\text{Number of Selected Features}}{\text{Total Features}}$$

where $\alpha = 0.01$ is a penalty coefficient controlling the trade-off between accuracy and parsimony. Lower cost values indicate better solutions.

3.3.2 Base Classifier

Random Forest with 100 estimators was selected as the evaluation model due to:

- Robustness to overfitting
- Ability to handle both categorical and numerical features
- Fast training time suitable for iterative optimization
- Built-in feature importance metrics

3.4 Particle Swarm Optimization (PSO)

3.4.1 Algorithm Overview

PSO is a population-based metaheuristic inspired by social behavior of bird flocking. Each particle represents a potential feature subset encoded as a continuous vector in $[0, 1]^d$ where d is the number of features. Features are selected when particle position exceeds a threshold of 0.5.

3.4.2 PSO Implementation Details

The PSO implementation uses the PySwarms library with the following configuration:

Table 2: PSO Hyperparameters

Parameter	**Value**
Number of particles	30
Maximum iterations	100
Inertia weight (ω)	0.5
Cognitive coefficient (c_1)	1.5
Social coefficient (c_2)	1.5
Position bounds	[0, 1]

3.4.3 PSO Update Equations

Particle velocities and positions are updated according to:

$$v_i^{t+1} = \omega v_i^t + c_1 r_1 (p_i^t - x_i^t) + c_2 r_2 (g^t - x_i^t)$$

$$x_i^{t+1} = x_i^t + v_i^{t+1}$$

where x_i^t is particle position, v_i^t is velocity, p_i^t is personal best, g^t is global best, and r_1, r_2 are random values in $[0, 1]$.

3.4.4 Feature Selection Mechanism

Binary feature selection is achieved through threshold-based discretization:

```
def feature_subset_fitness(position):
    selected = position > 0.5
    if selected.sum() == 0:
        return 1.0  # Maximum penalty

    X_subset = X_train[:, selected]
    model = RandomForestClassifier(n_estimators=100, random_state=42)
    model.fit(X_subset, y_train)

    y_pred = model.predict(X_test[:, selected])
    accuracy = accuracy_score(y_test, y_pred)

    penalty = 0.01 * (selected.sum() / len(position))
    cost = (1 - accuracy) + penalty
    return cost
```

3.5 Genetic Algorithm (GA)

3.5.1 Algorithm Overview

GA is an evolutionary algorithm inspired by natural selection. Each individual (chromosome) is a binary vector where 1 indicates feature selection and 0 indicates exclusion. The population evolves through selection, crossover, and mutation operations.

3.5.2 GA Implementation Details

The GA implementation uses the DEAP (Distributed Evolutionary Algorithms in Python) framework:

Table 3: GA Hyperparameters

Parameter	Value
Population size	50
Number of generations	11
Crossover probability	0.7
Mutation probability	0.2
Tournament size	3
Elite size	1

3.5.3 GA Operators

Selection: Tournament selection with size 3 chooses parents by randomly sampling 3 individuals and selecting the fittest.

Crossover: Two-point crossover exchanges genetic material between two parents:

```
toolbox.register("mate", tools.cxTwoPoint)
```

Mutation: Bit-flip mutation randomly inverts bits with 20% probability:

```
toolbox.register("mutate", tools.mutFlipBit, indpb=0.2)
```

3.5.4 Fitness Evaluation

The fitness function is identical to PSO's cost function, enabling direct comparison:

```
def eval_features(individual):
    selected = [i for i, bit in enumerate(individual) if bit == 1]

    if len(selected) == 0:
        return (1.0,)  # Return tuple for DEAP

    X_subset = X_train[:, selected]
    model = RandomForestClassifier(n_estimators=100, random_state=42)
    model.fit(X_subset, y_train)

    y_pred = model.predict(X_test[:, selected])
    accuracy = accuracy_score(y_test, y_pred)

    penalty = 0.01 * (len(selected) / len(individual))
    cost = (1 - accuracy) + penalty
    return (cost,)
```

3.6 Evaluation Metrics

3.6.1 Primary Metrics

- **Cost Function Value**: Combined measure of accuracy and feature count
- **Model Accuracy**: Classification accuracy on test set
- **Selected Features Count**: Number of features in optimal subset
- **Convergence Iterations**: Number of iterations to reach optimum

3.6.2 Convergence Criteria

Both algorithms were allowed to run for their full iteration budgets to observe convergence behavior, rather than implementing early stopping.

4 Implementation Details

4.1 Software Environment

4.1.1 Libraries and Dependencies

Table 4: Python Libraries Used

Library	Purpose
pandas	Data manipulation and analysis
numpy	Numerical computations
matplotlib	Visualization
seaborn	Statistical visualization
scikit-learn	Machine learning models and preprocessing
pyswarms	Particle swarm optimization
deap	Genetic algorithm framework

4.1.2 Reproducibility Settings

Random seeds were fixed for reproducible results:

```
np.random.seed(42)
random.seed(42)
```

4.2 Data Loading and Preprocessing Implementation

4.2.1 Data Loading

```
# Load dataset
df = pd.read_csv('loan_data.csv')
print(f"Original Data Shape: {df.shape}")

# Remove missing values
df = df.dropna()
print(f"Shape after removing NaNs: {df.shape}")
```

Output:

```
Original Data Shape: (614, 12)
Shape after removing NaNs: (499, 12)
```

4.2.2 Encoding Implementation

```
# Identify categorical columns
categorical_cols = ['Gender', 'Married', 'Dependents', 'Education',
                    'Self_Employed', 'Area', 'Status']

# Apply label encoding
encoder = LabelEncoder()
df_encoded = df.copy()

```

```
for col in categorical_cols:
    df_encoded[col] = encoder.fit_transform(df[col])
```

4.3 PSO Implementation Code

4.3.1 Fitness Function

```
def pso_fitness(position_matrix):
    n_particles = position_matrix.shape[0]
    costs = np.zeros(n_particles)

    for i in range(n_particles):
        selected_features = position_matrix[i] > 0.5

        # At least one feature must be selected
        if selected_features.sum() == 0:
            costs[i] = 1.0
            continue

        # Train model with selected features
        X_train_selected = X_train[:, selected_features]
        X_test_selected = X_test[:, selected_features]

        model = RandomForestClassifier(n_estimators=100, random_state=42)
        model.fit(X_train_selected, y_train)

        # Evaluate on test set
        y_pred = model.predict(X_test_selected)
        accuracy = accuracy_score(y_test, y_pred)

        # Calculate cost with penalty
        n_selected = selected_features.sum()
        penalty = 0.01 * (n_selected / len(selected_features))
        costs[i] = (1 - accuracy) + penalty

    return costs
```

4.3.2 PSO Execution

```
# Initialize optimizer
dimensions = X_train.shape[1]
options = {’c1’: 1.5, ’c2’: 1.5, ’w’: 0.5}

optimizer = ps.single.GlobalBestPSO(
    n_particles=30,
    dimensions=dimensions,
    options=options,
    bounds=([0]*dimensions, [1]*dimensions)
)

# Run optimization
print("Starting PSO Optimization...")
best_cost, best_position = optimizer.optimize(
    pso_fitness,
    iters=100,
    verbose=False
)
```

4.4 GA Implementation Code

4.4.1 DEAP Framework Setup

```
# Create fitness and individual classes
creator.create("FitnessMin", base.Fitness, weights=(-1.0,))
creator.create("Individual", list, fitness=creator.FitnessMin)

# Initialize toolbox
toolbox = base.Toolbox()

# Register genetic operators
toolbox.register("attr_bool", random.randint, 0, 1)
toolbox.register("individual", tools.initRepeat, creator.Individual,
                 toolbox.attr_bool, n=X_train.shape[1])
toolbox.register("population", tools.initRepeat, list, toolbox.individual)

toolbox.register("evaluate", eval_features)
toolbox.register("mate", tools.cxTwoPoint)
toolbox.register("mutate", tools.mutFlipBit, indpb=0.2)
toolbox.register("select", tools.selTournament, tournsize=3)
```

4.4.2 GA Evolution Loop

```
# Initialize population
population = toolbox.population(n=50)

# Evolution parameters
NGEN = 11
CXPB = 0.7
MUTPB = 0.2

print("Starting GA Optimization...")

# Run evolution
for gen in range(NGEN):
    # Select next generation
    offspring = toolbox.select(population, len(population))
    offspring = list(map(toolbox.clone, offspring))

    # Apply crossover
    for child1, child2 in zip(offspring[::2], offspring[1::2]):
        if random.random() < CXPB:
            toolbox.mate(child1, child2)
            del child1.fitness.values
            del child2.fitness.values

    # Apply mutation
    for mutant in offspring:
        if random.random() < MUTPB:
            toolbox.mutate(mutant)
            del mutant.fitness.values

    # Evaluate individuals with invalid fitness
    invalid_ind = [ind for ind in offspring if not ind.fitness.valid]
    fitnesses = map(toolbox.evaluate, invalid_ind)
    for ind, fit in zip(invalid_ind, fitnesses):
        ind.fitness.values = fit

    # Replace population
    population[:] = offspring

# Extract best individual
best_individual = tools.selBest(population, k=1)[0]
```

5 Experimental Results

5.1 PSO Results

5.1.1 Optimal Solution

The PSO algorithm converged to the following solution:

Table 5: PSO Optimization Results

Metric	**Value**
Minimum Cost	0.1711
Selected Features Count	1
Model Accuracy	0.8200 (82.0%)
Selected Feature	Credit_History

5.1.2 Console Output

```
--- PSO Results ---
Minimum Cost: 0.1711
Selected Features (1): ['Credit_History']
Model Accuracy: 0.8200
```

5.2 GA Results

5.2.1 Convergence Statistics

The GA evolution produced the following generational statistics:

Table 6: GA Generational Statistics

Generation	Evaluations	Average Cost	Best Cost
0	50	0.3359	0.1893
1	44	0.2884	0.1893
2	43	0.2447	0.1802
3	46	0.2240	0.1711
4	44	0.2040	0.1711
5	46	0.1842	0.1711
6	47	0.1778	0.1711
7	49	0.1731	0.1711
8	49	0.1713	0.1711
9	48	0.1718	0.1711
10	44	0.2042	0.1711

5.2.2 Optimal Solution

Table 7: GA Optimization Results

Metric	Value
Minimum Cost	0.1711
Selected Features Count	1
Model Accuracy	0.8200 (82.0%)
Selected Feature	Credit_History

5.3 Comparative Analysis

5.3.1 Algorithm Comparison

Table 8: PSO vs GA Performance Comparison

Algorithm	Features Selected	Accuracy	Cost
PSO	1	0.82	0.171091
GA	1	0.82	0.171091

5.3.2 Feature Selection Consistency

Both algorithms independently converged to the identical optimal solution:

- **PSO Selected Features**: ['Credit_History']
- **GA Selected Features**: ['Credit_History']

This remarkable convergence suggests that Credit_History is the dominant predictive feature in the dataset.

5.4 Convergence Analysis

5.4.1 PSO Convergence Behavior

PSO demonstrated rapid convergence, likely reaching the optimal solution within the first 20-30 iterations. The global best mechanism enabled quick information sharing across the swarm.

5.4.2 GA Convergence Behavior

GA showed steady improvement over generations:

- Generation 0-2: Rapid initial improvement (0.3359 → 0.2447)
- Generation 3: Optimal solution discovered (0.1711)
- Generation 4-10: Population convergence around optimal solution

The average population fitness converged toward the best solution by generation 8, indicating population homogeneity.

5.4.3 Convergence Comparison

Table 9: Convergence Metrics

Metric	**PSO**	**GA**
Final Cost	0.1711	0.1711
Approximate Convergence Generation	¡20	3
Stability after Convergence	High	Moderate

6 Discussion and Analysis

6.1 Feature Importance Interpretation

6.1.1 Credit History Dominance

The selection of Credit_History as the sole optimal feature carries significant implications:

1. **Predictive Power**: Credit_History alone achieves 82% accuracy, indicating strong correlation with loan approval
2. **Parsimony**: Additional features provide marginal accuracy gains insufficient to offset the complexity penalty
3. **Business Interpretation**: Credit repayment history is the primary determinant in loan decisions

6.1.2 Why Other Features Were Excluded

The cost function penalizes feature count, creating pressure for minimal subsets. Features like income, loan amount, and education likely contribute to accuracy but not sufficiently to justify their inclusion given the $\alpha = 0.01$ penalty coefficient.

6.2 Algorithm Performance Comparison

6.2.1 Convergence Speed

GA demonstrated faster convergence to the global optimum (generation 3) compared to PSO's more gradual approach. This may be attributed to:

- GA's explicit exploitation through elitism
- Crossover operator's ability to combine good solutions
- Binary representation better suited to discrete feature selection

6.2.2 Search Strategy Differences

PSO: Continuous search space with threshold-based discretization provides smooth gradient-like behavior but may be less natural for binary problems.

GA: Native binary encoding directly represents feature subsets, potentially more efficient for this problem structure.

6.2.3 Computational Efficiency

- **PSO**: 30 particles × 100 iterations = 3,000 evaluations
- **GA**: 50 individuals × 11 generations = 550 evaluations (with elitism reducing redundant evaluations)

GA required significantly fewer evaluations, offering computational advantages.

6.3 Strengths and Limitations

6.3.1 Strengths

- Both algorithms successfully identified the optimal feature subset
- Consistency between independent methods validates the solution
- Automated feature selection reduces manual feature engineering
- Interpretable results facilitate business decision-making

6.3.2 Limitations

- Single classifier (Random Forest) limits generalizability
- Small penalty coefficient ($\alpha = 0.01$) strongly favors minimal features
- No cross-validation within optimization loop (potential overfitting to test set)
- Missing value removal may introduce selection bias
- Limited hyperparameter tuning for evolutionary algorithms

6.4 Practical Implications

6.4.1 Loan Approval System Design

Results suggest that a streamlined loan approval model based primarily on credit history could achieve 82% accuracy. This simplification offers:

- Faster application processing
- Reduced data collection requirements
- Lower computational costs
- Easier model interpretation for regulatory compliance

6.4.2 Risk Considerations

Reliance on a single feature increases model fragility:

- Vulnerable to manipulation of credit history
- May not capture complex applicant profiles
- Could perpetuate bias if credit history data contains historical discrimination

7 Conclusions and Recommendations

7.1 Key Findings

1. Both PSO and GA converged to identical optimal feature subsets consisting solely of Credit_History
2. The optimal model achieves 82% test accuracy with minimal complexity
3. GA demonstrated superior computational efficiency with fewer fitness evaluations
4. Feature selection successfully reduced dimensionality from 11 to 1 feature

7.2 Limitations of Current Work

1. Use of test set for fitness evaluation violates proper validation methodology
2. Single classifier limits understanding of feature importance across models
3. Fixed penalty coefficient prevents sensitivity analysis
4. Missing value handling could be improved with imputation techniques

7.3 Future Improvements

7.3.1 Methodological Enhancements

- Implement k-fold cross-validation within the fitness function
- Compare multiple classifiers (SVM, Neural Networks, Gradient Boosting)
- Conduct sensitivity analysis on penalty coefficient α
- Investigate ensemble feature selection methods

7.3.2 Algorithm Extensions

- Hybrid PSO-GA algorithm combining strengths of both approaches
- Multi-objective optimization (accuracy vs. feature count as separate objectives)
- Adaptive parameter control for PSO and GA
- Island model genetic algorithms for improved diversity

7.3.3 Application Enhancements

- Feature importance ranking beyond binary selection
- Stability analysis across multiple random seeds
- Analysis of feature interactions
- Deployment pipeline for production systems

Part II

Customer Segmentation using Clustering Analysis(Q2)

8 Problem Statement and Dataset

8.1 Problem Definition

The objective of Part II is to discover natural customer segments within a mall customer dataset using unsupervised clustering techniques. Effective segmentation enables targeted marketing strategies, personalized customer experiences, and optimized resource allocation.

8.2 Dataset Description

The mall customer dataset contains 200 records with demographic and behavioral attributes:

Table 10: Customer Dataset Features

Feature	**Type**	**Description**
CustomerID	Identifier	Unique customer identifier
Gender	Categorical	Customer gender
Age	Numerical	Customer age in years
Annual Income (k$)	Numerical	Annual income in thousands of dollars
Spending Score (1-100)	Numerical	Spending behavior score

8.3 Clustering Objectives

- Identify optimal number of customer segments
- Compare performance of different clustering algorithms
- Visualize cluster structure in reduced dimensionality
- Provide actionable customer segment profiles

9 Methodology

9.1 Overall Workflow

The clustering analysis workflow consists of:

1. **Data Loading and Cleaning**: Import and validate data quality
2. **Feature Selection**: Extract numerical features for clustering

3. **Data Standardization**: Apply z-score normalization
4. **Dimensionality Reduction**: PCA for 2D visualization
5. **K-Means Clustering**: Determine optimal k via silhouette analysis
6. **Agglomerative Clustering**: Compare linkage methods
7. **DBSCAN**: Grid search for optimal density parameters
8. **Comparative Evaluation**: Visualize and compare all methods

9.2 Data Preprocessing

9.2.1 Data Loading and Validation

```
import pandas as pd
import numpy as np

# Load dataset
df = pd.read_csv(’Mall_Customers.csv’)

# Display basic information
print(df.shape)
print(df.info())
print(df.describe())
```

The dataset contained no missing values, requiring no imputation.

9.2.2 Feature Engineering

Only numerical features were retained for clustering:

```
# Select numerical features
X = df[[’Age’, ’Annual Income (k$)’, ’Spending Score (1-100)’]].values
feature_names = [’Age’, ’Annual Income (k$)’, ’Spending Score (1-100)’]
```

Gender was excluded as most clustering algorithms require numerical inputs without additional encoding considerations.

9.2.3 Standardization

Features were standardized using z-score normalization to ensure equal weighting:

$$z = \frac{x - \mu}{\sigma}$$

Implementation:

```
from sklearn.preprocessing import StandardScaler

scaler = StandardScaler()
X_scaled = scaler.fit_transform(X)
```

Standardization is critical for distance-based clustering algorithms to prevent features with larger scales from dominating similarity calculations.

9.3 Dimensionality Reduction for Visualization

9.3.1 Principal Component Analysis

PCA was applied to project data into 2D space for visualization while preserving maximum variance:

```
from sklearn.decomposition import PCA

pca = PCA(n_components=2)
X_pca = pca.fit_transform(X_scaled)

print(f"Explained variance: {pca.explained_variance_ratio_}")
```

The first two principal components captured the majority of dataset variance, enabling effective 2D visualization of cluster structure.

9.4 K-Means Clustering

9.4.1 Algorithm Overview

K-Means partitions data into k clusters by iteratively:

1. Assigning points to nearest centroid
2. Updating centroids as cluster means
3. Repeating until convergence

9.4.2 Optimal k Selection

Multiple values of k (2 through 10) were evaluated using:

- **Inertia**: Within-cluster sum of squared distances (lower is better)
- **Silhouette Score**: Cluster cohesion and separation (-1 to 1, higher is better)

```
from sklearn.cluster import KMeans
from sklearn.metrics import silhouette_score

results = []

for k in range(2, 11):
    kmeans = KMeans(n_clusters=k, random_state=42, n_init=10)
    labels = kmeans.fit_predict(X_scaled)

    inertia = kmeans.inertia_
    silhouette = silhouette_score(X_scaled, labels)

    results.append({
        'K': k,
        'Inertia': inertia,
        'Silhouette': silhouette
    })
```

9.5 Agglomerative Clustering

9.5.1 Algorithm Overview

Agglomerative clustering is a hierarchical bottom-up approach that:

1. Starts with each point as a separate cluster
2. Iteratively merges closest cluster pairs
3. Continues until desired number of clusters reached

9.5.2 Linkage Methods

Four linkage methods were compared:

- **Single**: Minimum distance between cluster members
- **Complete**: Maximum distance between cluster members
- **Average**: Average distance between all pairs
- **Ward**: Minimizes within-cluster variance

```
from sklearn.cluster import AgglomerativeClustering

linkages = ['single', 'complete', 'average', 'ward']
agg_results = []

for linkage in linkages:
    agg = AgglomerativeClustering(n_clusters=best_k, linkage=linkage)
    labels = agg.fit_predict(X_scaled)

    silhouette = silhouette_score(X_scaled, labels)

    agg_results.append({
        'Linkage': linkage,
        'Silhouette': silhouette
    })
```

9.6 DBSCAN Clustering

9.6.1 Algorithm Overview

DBSCAN (Density-Based Spatial Clustering of Applications with Noise) identifies clusters as high-density regions separated by low-density areas. Key advantages:

- Discovers arbitrary-shaped clusters
- Automatically determines number of clusters
- Identifies outliers as noise points

9.6.2 Hyperparameter Grid Search

Two critical parameters were tuned via grid search:

- **eps** (ϵ): Maximum distance for neighborhood
- **min_samples**: Minimum points to form dense region

```
from sklearn.cluster import DBSCAN

eps_values = [0.2, 0.4, 0.6, 0.8, 1.0]
min_samples_values = [3, 5, 10]
dbscan_results = []

for eps in eps_values:
    for min_samples in min_samples_values:
        dbscan = DBSCAN(eps=eps, min_samples=min_samples)
        labels = dbscan.fit_predict(X_scaled)

        n_clusters = len(set(labels)) - (1 if -1 in labels else 0)
        noise_ratio = (labels == -1).sum() / len(labels)

        if n_clusters > 1:
            silhouette = silhouette_score(X_scaled, labels)
        else:
            silhouette = np.nan

        dbscan_results.append({
            'eps': eps,
            'min_samples': min_samples,
            'n_clusters': n_clusters,
            'noise_ratio': noise_ratio,
            'Silhouette': silhouette
        })
```

9.7 Evaluation Metrics

9.7.1 Silhouette Score

The primary evaluation metric, silhouette score measures both cluster cohesion and separation:

$$s(i) = \frac{b(i) - a(i)}{\max\{a(i), b(i)\}}$$

where $a(i)$ is mean intra-cluster distance and $b(i)$ is mean nearest-cluster distance.

9.7.2 Additional Metrics

- **Inertia**: Within-cluster sum of squares (K-Means only)
- **Number of Clusters**: Automatically determined by DBSCAN
- **Noise Ratio**: Proportion of outliers (DBSCAN only)

10 Experimental Results

10.1 K-Means Results

10.1.1 Performance Across Different k Values

Table 11: K-Means Clustering Performance

K	Inertia	Silhouette Score
2	389.39	0.3355
3	295.21	0.3578
4	205.23	0.4040
5	168.25	0.4166
6	133.87	**0.4274**
7	117.01	0.4172
8	103.83	0.4087
9	92.35	0.4201
10	81.99	0.3973

10.1.2 Optimal k Selection

Based on maximum silhouette score, k=6 was selected as optimal, achieving:

- Silhouette Score: 0.4274
- Inertia: 133.87
- Balance between model complexity and cluster quality

10.2 Agglomerative Clustering Results

10.2.1 Linkage Method Comparison

Table 12: Agglomerative Clustering Linkage Comparison (k=6)

Linkage Method	Silhouette Score
Single	-0.0428
Complete	0.3746
Average	0.3896
Ward	**0.4201**

10.2.2 Best Configuration

Ward linkage achieved the highest silhouette score (0.4201), outperforming other methods:

- Ward linkage minimizes within-cluster variance
- Single linkage performed poorly due to chaining effect
- Complete and average linkage showed moderate performance

10.3 DBSCAN Results

10.3.1 Grid Search Results

Table 13: DBSCAN Hyperparameter Grid Search Results

eps	min_samples	Clusters	Noise %	Silhouette
0.2	3	11	80.5	0.6459
0.2	5	1	97.5	—
0.2	10	0	100.0	—
0.4	3	10	29.5	0.4426
0.4	5	6	49.0	0.5190
0.4	10	2	85.0	**0.7661**
0.6	3	3	7.0	0.2149
0.6	5	2	14.0	0.2730
0.6	10	4	33.0	0.5296
0.8	3	1	0.0	—
0.8	5	1	0.0	—
1.0	3	1	0.0	—

10.3.2 Optimal DBSCAN Configuration

The best performing configuration was:

- **eps**: 0.4
- **min_samples**: 10
- **Number of Clusters**: 2
- **Silhouette Score**: 0.7661 (highest overall)
- **Noise Ratio**: 85%

While this configuration achieves the highest silhouette score, the 85% noise ratio indicates most points are classified as outliers, limiting practical utility.

10.4 Overall Comparison

10.4.1 Algorithm Performance Summary

Table 14: Clustering Algorithm Comparison

Algorithm	Configuration	Clusters	Silhouette
K-Means	k=6	6	0.4274
Agglomerative	Ward, k=6	6	0.4201
DBSCAN	eps=0.4, min_samples=10	2	0.7661
DBSCAN	eps=0.4, min_samples=5	6	0.5190

10.4.2 Trade-off Analysis

- **Highest Silhouette**: DBSCAN (eps=0.4, min_samples=10) with 0.7661
- **Most Clusters**: K-Means and Agglomerative with 6 segments
- **Best Balance**: DBSCAN (eps=0.4, min_samples=5) with 6 clusters and 0.5190 silhouette

11 Discussion and Analysis

11.1 Cluster Quality Interpretation

11.1.1 Silhouette Score Analysis

All algorithms achieved moderate to good silhouette scores (0.4-0.77), indicating reasonably well-separated clusters. The variation in scores reflects:

- Different geometric assumptions (spherical vs. density-based vs. hierarchical)
- Sensitivity to cluster compactness vs. separation
- Trade-offs between granularity and cohesion

11.1.2 Number of Clusters

The optimal cluster count varied by algorithm:

- K-Means and Agglomerative converged on 6 segments
- DBSCAN identified 2-11 clusters depending on parameters
- Higher k values in K-Means showed diminishing returns

11.2 Algorithm-Specific Insights

11.2.1 K-Means Analysis

Strengths:

- Consistent, interpretable results
- Computational efficiency
- Clear silhouette peak at k=6

Limitations:

- Assumes spherical clusters
- Sensitive to initialization (mitigated by n_init=10)
- Requires pre-specifying k

11.2.2 Agglomerative Clustering Analysis

Strengths:

- Ward linkage produced high-quality clusters
- Hierarchical structure enables multi-resolution analysis
- No assumptions about cluster shape

Limitations:

- Single linkage failed due to chaining
- Computationally expensive for large datasets
- Requires predefined cluster count

11.2.3 DBSCAN Analysis

Strengths:

- Highest silhouette scores achieved
- Discovers arbitrary-shaped clusters
- Automatic outlier detection

Limitations:

- High noise ratios reduce usable segments
- Parameter tuning critical and dataset-specific
- Struggles with varying density clusters

11.3 Practical Customer Segmentation Insights

11.3.1 Six-Cluster Solution

Both K-Means and Agglomerative clustering suggest 6 natural customer segments based on age, income, and spending behavior. These segments likely represent:

- High-income, high-spending customers
- High-income, low-spending customers
- Low-income, high-spending customers
- Young vs. mature age groups with varying behaviors

11.3.2 Business Applications

- **Targeted Marketing**: Customize campaigns for each segment
- **Product Recommendations**: Align inventory with segment preferences
- **Customer Retention**: Identify high-value segments for loyalty programs
- **Resource Allocation**: Optimize staffing and inventory by segment behavior

11.4 Strengths and Limitations

11.4.1 Strengths

- Comprehensive comparison of three major clustering paradigms
- Systematic hyperparameter optimization
- PCA visualization enables intuitive cluster interpretation
- Multiple evaluation metrics provide robust assessment

11.4.2 Limitations

- Small dataset (200 samples) limits statistical robustness
- Gender feature excluded despite potential segmentation value
- No domain expertise validation of discovered segments
- Silhouette score alone may not capture business-relevant quality
- No stability analysis across random initializations

12 Conclusions and Recommendations

12.1 Key Findings

1. K-Means with k=6 provides the best balance of cluster quality and interpretability
2. Ward linkage agglomerative clustering performs comparably to K-Means
3. DBSCAN achieves highest silhouette scores but with impractical noise ratios
4. Six customer segments emerge as the natural grouping across methods
5. All algorithms benefit from standardized features

12.2 Recommended Approach

For practical customer segmentation:

- **Primary Method**: K-Means with k=6
- **Validation**: Compare with Ward linkage agglomerative results
- **Visualization**: Use PCA for stakeholder communication
- **Refinement**: Consider DBSCAN for outlier detection

12.3 Future Improvements

12.3.1 Methodological Enhancements

- Incorporate gender through one-hot encoding
- Apply multiple initialization runs and ensemble clustering
- Use Calinski-Harabasz and Davies-Bouldin indices for validation
- Perform stability analysis with bootstrapping
- Conduct gap statistic analysis for k selection

12.3.2 Advanced Techniques

- Gaussian Mixture Models for probabilistic clustering
- Spectral clustering for complex manifold structures
- Deep learning-based clustering (autoencoders)
- Multi-view clustering combining multiple feature spaces

12.3.3 Business Integration

- Profile each segment with descriptive statistics
- Validate segments with business stakeholders
- Develop segment-specific strategies and KPIs
- Implement real-time segment assignment for new customers
- Monitor segment evolution over time

Overall Conclusion

This technical report presented two comprehensive machine learning projects demonstrating evolutionary algorithms for feature selection and clustering techniques for customer segmentation.

In Part I, both PSO and GA successfully identified Credit_History as the dominant predictive feature for loan approval, achieving 82% accuracy with minimal model complexity. The convergence of independent optimization methods to identical solutions validates the robustness of this finding.

In Part II, systematic comparison of K-Means, Agglomerative, and DBSCAN clustering revealed six natural customer segments, with K-Means providing optimal balance between cluster quality (silhouette score 0.4274) and practical interpretability.

Both projects demonstrate the value of:

- Rigorous comparative methodology
- Automated optimization techniques
- Multiple evaluation metrics
- Practical interpretability of results

The implementations provide reusable frameworks for feature selection and clustering tasks, with clear pathways for enhancement and production deployment.

Part III
Question 3

12.4 Part A) Q-learning Update Equation and Parameters

The Q-learning update equation is given by:

$$Q_{t+1}(s_t, a_t) = (1-\alpha)Q_t(s_t, a_t) + \alpha \left(r_t + \gamma \max_a Q_t(s_{t+1}, a) \right)$$

Parameters:

- α (Learning Rate): Determines the extent to which newly acquired information overrides old information. It ranges from 0 to 1, where $\alpha = 0$ means the agent learns nothing (only exploitation), and $\alpha = 1$ means the agent considers only the most recent information.
- γ (Discount Factor): Measures the importance of future rewards. It ranges from 0 to 1, where $\gamma = 0$ makes the agent myopic (considering only current rewards), and $\gamma \to 1$ makes it strive for long-term high rewards.

Why is Q-learning an Off-Policy method?

Q-learning is considered an off-policy algorithm because it evaluates and updates a policy (the target policy) that is different from the policy used to generate behavior (the behavior policy). Specifically:

- The behavior policy (often ϵ-greedy) is used to select actions a_t for exploration.
- The target policy (greedy) is used in the update rule $\max_a Q(s_{t+1}, a)$ to estimate the optimal future value, assuming the best action is taken henceforth.

Role of ϵ in ϵ-greedy policy:
The parameter ϵ (epsilon) balances exploration and exploitation:

- With probability ϵ, the agent explores by choosing a random action.
- With probability $1 - \epsilon$, the agent exploits by choosing the action with the highest estimated Q-value.
- A common strategy is to start with a high ϵ (more exploration) and decay it over time to focus on exploitation as the agent learns.

12.5 Part B) Calculation

Given:

- Initial Q-values are zero: $Q(s_0, a_1) = 0$
- Transition: $(s_t = s_0, a_t = a_1, r_t = +2, s_{t+1} = s_1)$
- $\max_a Q(s_1, a) = 1.5$
- Parameters: $\alpha = 0.2$, $\gamma = 0.9$

Step-by-step Calculation:
Using the update rule:

$$Q_{new}(s_0, a_1) = (1 - \alpha)Q_{old}(s_0, a_1) + \alpha \left(r_t + \gamma \max_a Q(s_1, a) \right)$$

Substitute the values:

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

$$Q_{new}(s_0, a_1) = 0 + 0.2 \times (2 + 1.35)$$

$$Q_{new}(s_0, a_1) = 0.2 \times 3.35$$

$$Q_{new}(s_0, a_1) = 0.67$$

Final Answer: The new value is $Q(s_0, a_1) = 0.67$.

12.6 Part C) Factors Causing Instability and Solutions

Two factors that can cause slow convergence or instability, along with their practical solutions:

1. **Sparse or Delayed Rewards**
   Problem: In many environments, the agent receives feedback (reward) only after a long sequence of actions (e.g., reaching a maze exit). This causes the "credit assignment problem," where the agent struggles to associate early actions with the final outcome, leading to slow convergence.
   Solution: ***Reward Shaping***
   Modify the reward function to include intermediate heuristic rewards that guide the agent toward the goal. For example, giving a small positive reward for reducing the distance to the target.

2. **Correlations in Observation Sequence**
   Problem: In reinforcement learning, consecutive samples (s_t, a_t, r_t, s_{t+1}) are highly correlated. This violates the i.i.d. (independent and identically distributed) assumption required by many learning algorithms (especially when using neural networks), leading to oscillations or divergence.
   Solution: ***Experience Replay***
   Store transitions in a replay buffer memory. During training, sample a random batch of transitions from this buffer to update the Q-network. This random sampling breaks the temporal correlations between consecutive data points and stabilizes training.