LONDON METROPOLITAN UNIVERSITY

ITAHARI INTERNATIONAL COLLEGE

Module Code & Module Title

CU6051NT Artificial Intelligence

75% Individual Coursework

Submission: Milestone 1

Academic Semester: Autumn Semester 2025

Credit: 15 credit semester long module

Student Name: Arpan Upreti

London Met ID: 23049196

College ID: NP05CP4A230146

Assignment Due Date: 07/01/2026.

Assignment Submission Date: 06/01/2026

Submitted To: Mr. Nikesh Regmi

Title: Customer Segmentation

GitHub Link	*https://github.com/ArpanUpreti1/Customer-Segmentation.git*

I confirm that I understand my coursework needs to be submitted online via MST Classroom under the relevant module page before the deadline for my assignment to be accepted and marked. I am fully aware that late submissions will be treated as non-submission and a mark of zero will be awarded

Table of Contents

Table of Figures

1. Introduction

Machine learning and Artificial Intelligence

Artificial Intelligence (AI) can be defined as the creation of computational systems that are able to carry out tasks that are normally performed by human intelligence like the recognition of patterns and decision-making. Machine Learning (ML), which is a major branch of AI, is the training of algorithms based on empirical data to formulate underlying structures without explicit code (IBM, 2023). ML helps to move away reactive analytics to the predictive modelling in the realm of business intelligence.

Core Technologies of AI

Unsupervised Learning used in this project is Clustering and Density-Based Estimation. In contrast to supervised learning (where there has to be labeled data available), unsupervised algorithms (e.g. K-Means, Gaussian Mixture Models) independently discover natural groupings within complex data. Moreover, the project incorporates Evolutionary Computer through the Genetic Algorithm (GA), which recreates natural selection in the biology to ensure the optimization of the model hyperparameters in real-time.

AI in Daily Life

The use of AI is everywhere in modern society, with the recommendation systems of Netflix and Amazon being powered by it, fraud detection in banks, and voice assistants such as Siri. The retail industry is at the center of AI implementation and Hyper-Personalization that enables companies to quit relying on the broad demographics of their customers and focus on narrow behavioral patterns.

Project Overview

The problem area concerns the inefficiency of marketing which is a one-size-fits-all in online retail. Variations in customer bases are usually characterised by noise (outliers), and non-spherical behavioral forms that are difficult to characterise using linear segmentation (Meduim, 2023). This proposed project suggests a Hybrid Two-Stage Segmentation System. It makes use of DBSCAN to clean up the data by eliminating statistical noise and an ensemble optimized by a Genetic Algorithm that dynamically switches between K-Means and Gaussian Mixture Models (GMM). This will guarantee the identification of actionable, high-purity segments (e.g., "Hibernating," Champions) in order to allocate resources accurately.

2. Background

Conventional customer segmentation has been based on heuristic rules (e.g. RFM score) or simple clustering algorithms such as K-Means. Although computationally effective, they have severe limitations: they rely on the assumption of the shape of the clusters being spherical, and are extremely sensitive to outliers. These shortcomings result in misclassification and marketing budget loss in high dimensional retail data where the customer behavior is unpredictable.

Review and Analysis

The main technical problem lies in the Curse of Dimensionality and noise in transaction information. Only one customer (high spender, one-time purchase) can skew the centroid of a K-Means cluster, which will cause a whole segment of loyal customers to be reclassified. In addition, the tuning of static algorithms (i.e. $k selection) is also subject to human bias. According to a review of literature, the solution needs to be strong (i.e. filter noise (outlier detection) and then it has to handle overlapping customer behaviors using a probabilistic approach (GMM).

Literature Review

1. Effectiveness and Limitations of K-Means using RFM.

Study Objective: Christy et al. (2021) have performed an extensive research on the use of K-Means clustering combined with Recency, Frequency, Monetary (RFM) scoring in order to segment the customers. The authors were using the highly referenced UCI Online Retail dataset that contains transactional information of an online retailer based in the UK. They used a methodology that included preprocessing the raw transactional information in order to compute RFM measures of individual customers and then normalise these values so as to have equal weighting of the dimensions. K-Means algorithm was applied in a systematic way and cluster configurations with k=3,4,5k=3, 4, 5 utilized.

k=3,4,5 to know the best structure of segmentation.

Significant Results: The research revealed positive outcomes and the Silhouette Score was 0.64, representing rather separated and bonded clusters. The authors have managed to find some specific customer segments with the most notable one being a Best Customer segment with high recency, frequency and monetary values. These clients were the most valuable group of customers that could be targeted in marketing efforts and retention.

Critical Limitations: non less, the authors also admitted that K-Means approach is subject to a number of critical limitations. The nature of algorithmity with its basis on Euclidean distances metrics essentially makes clusters circular and thus does not necessarily capture the actual underlying customers behavioral distribution. This mathematical limitation lead to the systematic mis-categorization of so-called boundary customers, i.e., those with hybrid traits who cut across several segments. Moreover, the research did not adequately deal with the fact of outliers in the data, which most probably skewed cluster centers and poor accuracy in segmentation. These constraints highlight the necessity to develop more advanced methods, namely non-linear, probabilistic clustering methods that would be more flexible to irregular cluster shapes and noisy data (A. Joy Christy, A. Umamakeswari , L. Priyatharsini, A. Neyaa , 2021).

Journal of King Saud University – Computer and Information Sciences 33 (2021) 1251–1257

Contents lists available at ScienceDirect

Journal of King Saud University – Computer and Information Sciences

journal homepage: www.sciencedirect.com

RFM ranking – An effective approach to customer segmentation

A. Joy Christy [a,*], A. Umamakeswari [a], L. Priyatharsini [b], A. Neyaa [b]

[a] Department of CSE, School of Computing, SASTRA Deemed to be University, Thanjavur, India
[b] School of Computing, SASTRA Deemed to be University, Thanjavur, India

ARTICLE INFO

Article history:
Received 1 June 2018
Revised 26 August 2018
Accepted 4 September 2018
Available online 5 September 2018

Keywords:
Customer segmentation
RFM analysis
K-Means
Fuzzy C-Means
Initial centroids

ABSTRACT

The efficient segmentation of customers of an enterprise is categorized into groups of similar behavior based on the RFM (Recency, Frequency and Monetary) values of the customers. The transactional data of a company over is analyzed over a specific period. Segmentation gives a good understanding of the need of the customers and helps in identifying the potential customers of the company. Dividing the customers into segments also increases the revenue of the company. It is believed that retaining the customers is more important than finding new customers. For instance, the company can deploy marketing strategies that are specific to an individual segment to retain the customers. This study initially performs an RFM analysis on the transactional data and then extends to cluster the same using traditional K-means and Fuzzy C- Means algorithms. In this paper, a novel idea for choosing the initial centroids in K- Means is proposed. The results obtained from the methodologies are compared with one another by their iterations, cluster compactness and execution time.

1. Introduction

In recent years, there has been a massive increase in the competition among firms in sustaining in the field. The profits of the company can be improved by a customer segmentation model. Customer retention is more important than the acquisition of new customers. According to the Pareto principle (Srivastava, 2016), 20% of the customers contribute more to the revenue of the company than the rest. Customer segmentation can be performed using a variety of unique customer characteristics to help business people to customize marketing plans, identify trends, plan product development, advertising campaigns and deliver relevant products. Customer segmentation personalizes the messages of individuals to better communicate with the intended groups. The most common attributes used in customer segmentation are location, age, sex, income, lifestyle and previous purchase behavior.

Here, segmentation is done using behavioral data since it is commonly available and continuously evolving with time and purchase history. RFM (Recency, Frequency, and Monetary) analysis is a renowned technique used for evaluating the customers based on their buying behavior. A scoring method is developed to evaluate scores of Recency, Frequency, and Monetary. Finally, the scores of all three variables are consolidated as RFM score ranging from 555 to 111 (Haiying and Yu, 2010) which is used to predict the future patterns by analyzing the present and past histories of the customer. In this context, it has been observed that the scores of three factors Recency, Frequency and Monetary directly proportional to customer's lifetime and retention.

Once the values of recency, frequency and monetary are calculated, the K-Means algorithm is applied to the variables to clusters of the customer base. The behavior of each cluster is analyzed to find the group of customers who give more profits to the company. Similarly, clustering is performed using two other algorithms namely, Fuzzy C – Means clustering and the proposed method with chosen initial centroids in the existing K– Means algorithm. The motivation of the paper is to propose a method for choosing initial centroids for K-means algorithm and to impose the method to segment the customer with reduced iteration and time. Now that clus-

* Corresponding author.
E-mail address: joychristy@cse.sastra.edu (A.J. Christy).

Figure 1 Research Paper 1

2. Outlier Detection using Density-Based Spatial Clustering

Study Overview: Nowak-Brzezińska and Hieski (2019) conducted a focused investigation into the detrimental impact of noise and outliers on data mining quality and model performance. Their research specifically examined DBSCAN (Density-Based Spatial Clustering of Applications with Noise), a fundamentally different clustering paradigm that groups data points based on density reachability rather than traditional distance-from-centroid calculations. The algorithm operates by identifying core points that have a minimum number of neighbors within a specified radius, then expanding clusters by connecting these density-reachable points. The authors applied DBSCAN to various datasets containing different levels and types of noise to evaluate its effectiveness in distinguishing genuine patterns from statistical anomalies.

Key Findings: The study produced compelling evidence that DBSCAN demonstrates clear superiority over conventional partitioning methods (such as K-Means) for identifying and isolating anomalies within complex datasets. By carefully tuning two critical parameters—ϵ\epsilon

ϵ (the neighborhood radius) and MinPtsMinPts

MinPts (the minimum number of points required to form a dense region) the researchers successfully isolated statistical noise and outliers from the main dataset structure. The algorithm's ability to classify points as core, border, or noise points provided a nuanced understanding of data quality and enabled effective data cleaning. Their experiments showed significant improvements in subsequent analysis quality after DBSCAN-based outlier removal, with cleaner cluster formations and more interpretable patterns emerging in downstream tasks.

Critical Limitations: Despite its demonstrated effectiveness for outlier detection and data cleaning, the authors acknowledged several important limitations of the DBSCAN approach. Most notably, the algorithm struggles significantly when dealing with clusters of varying densities within the same dataset. For instance, in

customer segmentation scenarios where one customer group exhibits very tight, concentrated behavioral patterns while another displays sparse, dispersed characteristics, a single ϵ\epsilon

ϵ value cannot adequately capture both structures. This necessitates either multiple DBSCAN runs with different parameters or acceptance of suboptimal detection in some regions. Additionally, the performance and results are highly sensitive to parameter selection, requiring domain expertise and potentially extensive experimentation. These observations suggest that while DBSCAN serves as an excellent pre-processing step to identify and remove outliers, thereby improving data quality, it should be strategically combined with a more robust and flexible clustering algorithm for the final customer segmentation phase to achieve optimal results (Agnieszka Nowak-Brzezinska, 2017).

JOURNAL OF APPLIED
COMPUTER SCIENCE
Vol. 25 No. 2 (2017), pp. 53-68

Outlier Mining Using the DBSCAN Algorithm

Agnieszka Nowak-Brzezińska[1], Tomasz Xięski[1]

[1]*Institute of Computer Science*
University of Silesia in Katowice
Bankowa 12, 40-007 Katowice, Poland
agnieszka.nowak@us.edu.pl

Abstract. *This paper introduces an approach to outlier mining in the context of a real-world dataset containing information about the mobile transceivers operation. The goal of the paper is to analyze the influence of using different similarity measures and multiple values of input parameters for the density-based clustering algorithm on the number of outliers discovered during the mining process. The results of the experiments are presented in section 4 in order to discuss the significance of the analyzed parameters.*
Keywords: *outlier detection, similarity analysis, clustering, knowledge-based systems..*

1. Introduction

Outlier detection is a fundamental issue in data mining, it has been specifically used to detect and remove anomalous objects from data. Data mining, in general, deals with the discovery of nontrivial, hidden and interesting knowledge from different types of data. With the development of information technologies, the number of databases and their dimensions and complexity, has grown rapidly. One of the basic problems of data mining is outlier detection. The identification of an outlier is affected by various factors, many of which have become the subject of practical applications such as in the public health or finance field. In the first case

Figure 2 Research paper 2

3. Gaussian Mixture Model Probabilistic Clustering.

Study Overview: The study by Patel and Kushwaha (2020) was a comparative analysis of the hard and soft clustering approaches; that is, the K-Means algorithm and the Gaussian Mixture Models (GMM). Although they were more concerned with cloud workload statistics to optimize resource assigning, mathematical principles, and clustering dynamics can be directly applied to the analysis of customer behavior. The members of the research group conducted experiments to compare the two algorithms when there is an overlap of clusters and ambiguity, which is usually a common occurrence in real-life data. They applied these two methods with the same preprocessing procedures and measures of evaluation to make a fair comparison. To evaluate the performance degradation of each algorithm under more challenging conditions, the study systematically varied the extent of overlap in each of the clusters, and thus provided an insight on the robustness and adaptability of each algorithm.

Significant Results: The researchers came up with strong results that GMM outperformed K-Means in datasets that had intersecting or vague boundary lines. This advantage is based on the underlying probabilistic nature of GMM, where each point of data is assigned to a probability distribution over all potential clusters instead of necessarily classifying this data into a single, absolute cluster. To illustrate this, within the context of a customer segmentation described by GMM, a customer might be a 70per cent/30per cent mixture of a Champion and an At-Risk customer by definition, as far as borderline cases are concerned. This softer clustering technique was especially effective in the case of capturing the multi-dimensional, subtle characteristics of behavior data, as persons tend to have aspects which can cut across several different archetypal patterns. The probabilistic memberships also gave more information to the downstream with regard to decision-making and individualized marketing plans.

Critical Limitations: The authors have found that although GMM has been shown to have its benefits, there was an important algorithmic shortcoming in the algorithm that is the use of the Expectation-Maximization (EM) algorithm to estimate parameters. EM algorithm is extremely prone to local optima -sub optimal solutions that look like good solutions when the search space is confined but does not reflect the global optimum. The end result is the quality of the final clustering is very sensitive to starting parameter values (means, covariances, and mixture weights) that are usually chosen in a random manner or heuristically. The mis-setting of it may cause results that are drastically different and may be worse after a series of executions using the same dataset. This lack of stability renders reproducibility and reliability in the field of practices.

Gap Identification: This is a serious drawback that drives the methodological innovation of the proposed project. Within the framework of a Genetic Algorithm (GA), the project will explore the parameter space systematically by wrapping the GMM algorithm and

evolve the best starting values of the model and the best model settings. The population-based search mechanism implemented by the GA and the genetic operators (selection, crossover, mutation) make it escape the local optima by means of varied search of the solution space. The proposed hybrid solution will solve the inherent vulnerability posed by Patel et al., and is likely to provide globally best customer segmentation, which is both as flexible as GMM with its probabilistic approach and as strong as GA with its optimization ability.

Available online at www.sciencedirect.com

ScienceDirect

Procedia Computer Science 171 (2020) 158–167

Procedia Computer Science

www.elsevier.com/locate/procedia

Third International Conference on Computing and Network Communications (CoCoNet'19)

Clustering Cloud Workloads: K-Means vs Gaussian Mixture Model

Eva Patel[a,*], Dharmender Singh Kushwaha[a]

[a]*Motilal Nehru National Institute of Technology Allahabad, Prayagraj 211004, India*

Abstract

The growing heterogeneity due to diverse Cloud workloads such as Big Data, IoT and Business Data analytics, requires precise characterization to design a successful capacity plan and maintain the competitiveness of Cloud service providers. K-Means is a simple and fast clustering method, but it may not truly capture heterogeneity inherent in Cloud workloads. Gaussian Mixture Models can discover complex patterns and group them into cohesive, homogeneous components that are close representatives of real patterns within the data set. This work compares K-Means and Gaussian Mixture Model to evaluate cluster representativeness of the two methods for heterogeneity in resource usage of Cloud workloads. Experiments conducted with Google cluster trace and business critical workloads by Bitbrains reveal that clusters obtained using K-Means give a very abstracted information. Gaussian Mixture Model provides better clustering with distinct usage boundaries. Although, Gaussian Mixture Model has higher computation time than K-Means, it can be used when more fine-grained workload characterization and analysis is required.

Keywords: Cloud Computing; Workload characterization; Clustering; Expectation-Maximization; K-Means Clustering; Gaussian Mixture Model

1. Introduction

On-demand compute power and data storage are the key reasons behind rising Cloud adoption, emergence of new computing paradigms such as Volunteer computing, Fog computing, and Serverless computing, and a horde of new workloads such as Big Data, IoT and Business Data analytics [1, 2], hosted on geographically distributed resources [3]. Heterogeneity of workloads is now the norm in Cloud Computing environment [4, 5]. Identifying workload characteristics is therefore crucial for effective utilization of the infrastructure capacity and guaranteed Cloud services.

A fundamental step in Cloud infrastructure capacity planning is resource usage monitoring and workload characterization. *Workload characterization* is the process of identifying individual workload components and its describing attributes with respect to a characterizing objective [6]. A good characterization is one that truly represents workload behaviour in terms of an objective of interest. However, dynamic resource requirements, unforeseen demand surges, and disparate computational and configuration requirements of Cloud applications - transactional, scientific, stream-

* Corresponding author. Tel.: +91-808-557-5111.
E-mail address: evapatel08@gmail.com

Figure 3 Research Paper 3

3. Proposed Solution

The proposed solution is a Python-based automated pipeline. It ingests raw transaction data, transforms it into an RFM matrix, sanitizes it using **DBSCAN** to remove anomalies, and then uses a **Genetic Algorithm** to search for the optimal segmentation model (K-Means vs. GMM) and parameters (k), maximizing the internal validity of the clusters.

Dataset Background

- **Source:** The project utilizes the **Online Retail II Dataset**, sourced from the UCI Machine Learning Repository Click here.
- **Type:** Secondary, quantitative transactional data.
- **Content:** The dataset comprises 541,909 transactions from a UK-based online retailer (01/12/2010 – 09/12/2011). Key features include InvoiceNo, StockCode, Quantity, InvoiceDate, UnitPrice, CustomerID, and Country.
- **Suitability & Limitations:** The dataset is highly suitable for RFM analysis. However, it contains limitations such as missing CustomerID values (approx. 25% of rows) and negative quantities indicating returns. These must be handled during the pre-processing stage to ensure data reliability.

3.1 Algorithm

ALGORITHM 1: DBSCAN

DBSCAN (Density-Based Spatial Clustering of Applications with Noise) is a density-based clustering algorithm that clusters together points which are highly concentrated together and classifies points in low-density regions as outliers or noise. In contrast to partitioning clustering algorithms like K-Means, DBSCAN does not need the number of clusters to be known in advance and is able to detect clusters of arbitrary shapes. The algorithm mostly works well in noisey and outliered datasets, and so it can be well used in real-life applications where data can have atypical patterns.

Mathematical Concept: DBSCAN runs with two basic values that ascertain the density of the points.

- Epsilon (e): The largest radius of the neighborhood of a point. It sets the distance value of which the endpoints are deemed to belong to the same group.
- MinPts (Minimum Points): This is the minimum number of points that must appear in the epsilon neighborhood to make up a dense region. Those points not in epsilon that have the MinPts or less neighbours are viewed as Noise (-1) and removed during the clustering exercise.

These parameters are used to group the points into 3 categories in the algorithm as follows:

- Core Points: These are points which contain at least MinPts neighbors in their epsilon radius. These are the aspects on which clusters are based.
- Border Points: Points that lie within the epsilon radius of a core point and which do not contain themselves as MinPts neighbors. These arguments lie on the peripheries of groups.
- Noise Points: Points which are neither core points nor border points. They are not a part of a cluster and they are regarded as an outlier.

This type of classification allows DBSCAN to do successful denoising of the data before real clustering is done so that only significant patterns are found. It operates on the premise that neighborhoods of each point in the dataset are analyzed and clusters around core points are expanded. It commences with an arbitrary point that is not visited and retrieves all the

points that are within the distance e. When the number of neighboring points is at minimum MinPts, a new cluster is created and extended through recursive addition of all the density reachable points.

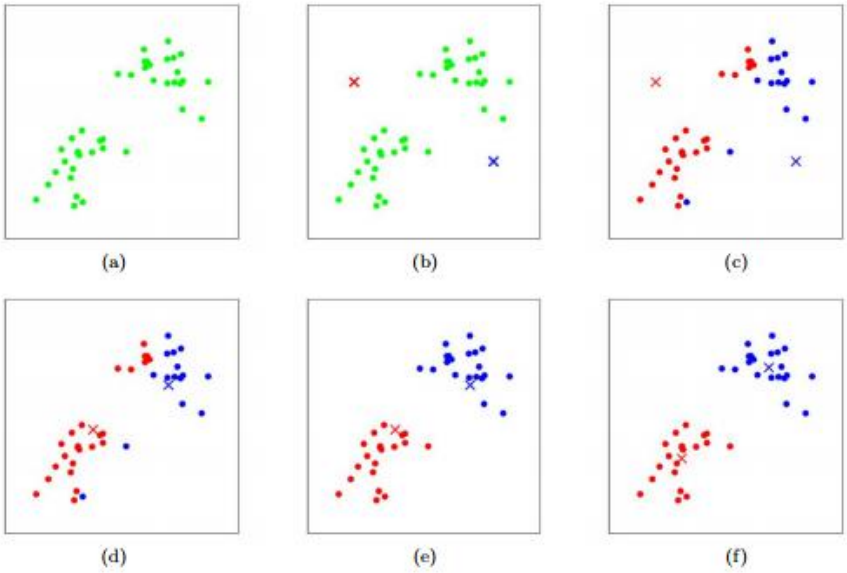


Figure 4 DABSCAN Clustering Visualization

The mathematical formula to be used to calculate core points can be written as:

For a point p, if Ne(p) [?] MinPts, then p is a core point

Where Ne (p) is a set of points that are within the epsilon distance of a point p.

DBSCAN finds wide applications in many fields such as:

- Geographic information systems and spatial data analysis.
- Network security, anomaly detection Fraud detection.
- Image segmentation, computer vision.
- Marketing analytics Customer segmentation.
- Medical imaging to determine trends in diagnostic data.

DBSCAN has the strengths that it can find arbitrary-shaped clusters, it is outlier resistant, and it automatically identifies the number of clusters. In contrast to K-Means, which presupposes the presence of the spherical clusters, DBSCAN may identify the clusters with irregular shapes and irregular densities. The algorithm also does not push all points into a cluster meaning it is able to isolate and detect noise points. (Naftali Harris, 2024)

Nevertheless, DBSCAN also is limited. The sensitivity of the algorithm is also high with regards to the epsilon and MinPts parameters, which may be difficult to find with datasets of different density. DBSCAN can be ineffective in detecting all clusters with one set of parameters when there is a large disparity in the density of clusters. Moreover, the computation cost of the algorithm may be high with large datasets but this can be reduced by application of spatial indexing structures like k-d trees or R-trees.

Algorithms 2: K- MEANS and GAUSSIAN MIXTURE MODELS (CLUSTERING)

K-Means K-Means is an iterative algorithm of partitioning that in a dataset classifies the groups of data into k independent, non-overlapping clusters (clusters) by minimizing Within-Cluster Sum of Squares (WCSS). It is among the most common and most used clustering algorithm because of its simplicity and performance in computation. The algorithm is based on the idea that the points inside a cluster are supposed to be as similar as possible, points outside the cluster should be as different as possible. (Analytics Vidhya, 2019)

The algorithm of K-Means is an iterative algorithm:

- Initialization: Choose k random points of the data set to act as starting cluster centres. There are several different ways of initialization, such as random selection, K-Means++ and Forgy method.
- Assignment Step: Given the data in the dataset, compute the distance between the point and all of the centroids and allocate it to the cluster containing the closest centroid. This is a good way of creating k clusters in terms of proximity.
- Step Update: Re-calculate the centroids of the clusters by estimating the average of all the data points belonging to each cluster. Every point in that cluster is then considered to be the center of mass of the new centroid.
- Iteration: Repeat steps 2 and 3 until convergence is achieved - when the centroids do not change substantially, or when there are a certain number of iterations.

The within-cluster sum of squares (WCSS): is the objective function that K-Means is minimizing:

WCSS = S(i=1 to k) S(x[?]Ci) x - mi 2

Where:

k is the number of clusters

Ci depicts the i th cluster.

x is a data point in cluster Ci

mi denotes the centroid of cluster Ci.

K-Means is also effective and it works well with the set of data when clusters are more or less spherical, and when they are of fairly similar size. Nonetheless, it has a number of limitations: The user must pre-set the number of clusters, it is sensitive to initial centroid location, it is a spherical-shaped cluster with equal variance, and it may have difficulties locating outliers. The algorithm is also deterministic in its assignment step and may give varied results on different initializations, the best practice is to run K-Means with various initializations and then pick the best results.

Gaussian Mixture Models (GMM)

GMM is a probabilistic clustering method, which models the data as a combination of an unknown number of Gaussian distributions with unknown parameters. GMM does soft clustering by assigning the probability that each point belongs to each cluster, as opposed to hard clustering as K-Means does, which assigns each point to only one cluster. This increases the flexibility of GMM and its suitability to datasets whose clusters are overlapping or complex.

Mathematical Foundation: GMM presumes that the data has been created as a mixture of k Gaussian components each with its own parameters:

Mean vector (mk): The mean of the k - th Gaussian component.

Covariance matrix (Sk): Characterizes the shape, dispersion and orientation of the kth component.

Mixing coefficient (pk): This is the probability that an arbitrarily selected data point is contained in the k-th component.

The GMM probability density function may be represented as:

p(x) = S(k=1 to K) pk N(x mk, Sk)

N (x mk, Sk) is the Gaussian distribution with a mean mk and covariance Sk.

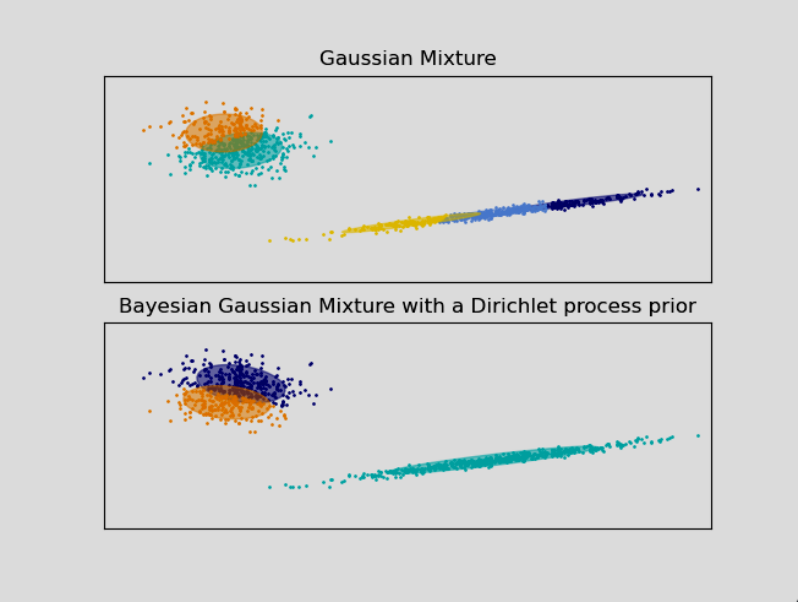


Figure 5 Gaussian Mixture Model Elements.

The Bayes' theorem is applied to calculate the responsibility of cluster k of data point xn:

g(zn = k) = (pk N(xn mk, Sk)/S(j=1 to K) pj N(xn mj, Sj))/S(j=1 to K) pj N(xn mk, Sj) pj N(xn mk, Sj)) is given by:

This is the posterior probability of a given data point xn to be a part of cluster k, given the observed data.

Training GMMs through the Expectation-Maximization (EM) Algorithm: The parameters of GMMs are estimated by the iterative EM algorithm that has two basic steps:

E-step (Expectation): Calculate the contribution made by the individual clusters to each data point by using the current parameter estimates. This is a computation of the likelihood of each point to lie in each Gaussian component.

M-step (Maximization): Estimate the model parameters (means, covariances and mixing coefficients) with the help of the responsibilities which can be obtained in the E-step. Such revised parameters minimize the possibility of the measured data.

This is done by iteration until convergence, i.e. when the log-likelihood of the data is no longer changing:

L(th) = S(n=1 to N) log[S(k=1 to K) pk N(xn mk, Sk)].

Where th denotes all model parameters mk, SK, pK,...

Covariance Structure: GMM is flexible in the modeling of cluster shapes by being able to use various constraints on covariance matrices:

Full Covariance: The individual components have their own general covariance matrix where elliptical clusters of arbitrary orientation are permitted. This offers the highest level of flexibility but there is a larger number of parameters.

Diagonal Covariance: Diagonal covariance implies that features are regarded to be independent in each component. Clusters are ellipsoids, which are axis aligned.

Spherical Covariance: There is a similarity, which is the covariance matrices proportional to the identity matrix, leading to spherical clusters just like K-Means.

Tied Covariance: The covariance matrix is the same amongst all the components, but it is a non-spherical cluster.

Uses and Comparison: GMM is good to model complex multi-modal distributions, and especially it is useful when modeling:

- Generative modeling and density estimation.
- Soft clustering in which points may differentially belong to several clusters.
- Detection of anomalies using likelihood.
- Pipe feature engineering of machine learning.
- Segmentation of image with overlapping regions.

GMM has several other benefits over K-Means: it assigns clusters probabilistically, can fit elliptical clusters with different orientations, deals with overlapping clusters naturalistically and provides a principled method of determining the optimal number of clusters with information criteria such as BIC (Bayesian Information Criterion) or AIC (Akaike Information Criterion) in its models. Nonetheless, GMM is less efficient in calculation than K-Means and needs more parameters to be approximated, which is why it is more likely to overfit on smaller data.

ALGORITHM 3: GENetic Algorithm (Evolutionary optimization)

Genetic Algorithms (GA) are innovative heuristic search methods that are based on the concepts of Charles Darwin on natural selection and genetics in his evolution theory. They are a part of the more general group of evolutionary algorithms, which are algorithms that attempt to model the mechanism of natural evolution to tackle search and optimization problems. GAs excel particularly in the optimization of complex problems to which the standard gradient-based techniques can be ineffective or computationally infeasible. (SlideShare, 2024)

Concept and Biological Inspiration: The basic idea of genetic algorithms is that a population of candidate solutions is evolved to better solutions using successive generations. A candidate solution is represented by each candidate solution, and the quality of a solution is described by a fitness function. The solutions with the most fitness functions have an increased likelihood of selection to reproduce, a biological operation, which resembles the principle of survival of the fittest. The population is developed through repeated use of the genetic operators-selection, crossover and mutation and the population is expected to evolve as a result of this evolution to an optimal or close to optimal solution. (Gate Vidyalay, 2024)

Architecture and Major Components:

- **Chromosome Representation:** Representation Each potential solution is represented as a chromosome which is usually a fixed length string of binary digits or real numbers (or other data structures with respect to the problem). In the example, one arrangement could be of the form [ModelType: GMM, nclusters: 5], with various aspects of the solution being represented in the chromosome.

- **Population:** This refers to a set of chromosomes that describe a number of candidate solutions. The size of the population is a crucial parameter which is a compromise between the exploration of the search space and the efficiency of the computation. The numbers of typical populations vary between 50 and several thousand people.

- **Fitness Function:** This is a mathematical function that analyzes and measures the quality of every chromosome. It is this role that informs the process of evolution by selecting the individuals with the greatest chances of reproducing. In optimization of clustering, the fitness may be Silhouette Score, Davies-Bouldin Index, or other measures of clustering quality.

- **Selection Operators:** The ways to select parent chromosomes to reproduce them by their values of fitness.

- **Crossover (Recombination):** This is a genetic operator that assembles genetic information of two parent chromosomes to produce offspring. This is a copy of biological reproduction that enables the algorithm to search new areas of the search space, by blending successful attributes on other solutions. The common crossover procedures are single-point crossover, two-point crossover and uniform crossover.

- **Mutation:** A genetic operator that produces random mutations in chromosomes, which ensures that genetic diversity does not converge too quickly on local optima. Mutation usually has a low probability and may include the inversion of bits in binary representations or an introduction of small random mutations to real-valued genes.

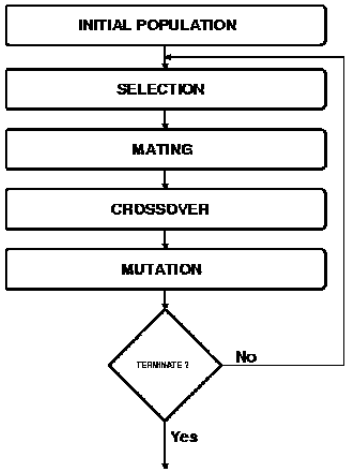


Figure 6 Simple GA Flowchart

The Genetic Algorithm Process:

Step 1: Initialization

Choose randomly a starting population of N chromosomes.

The chromosomes are the possible solutions that have the random parameters.

Initial genetic operator probabilities (crossover probability Pf and mutation probability Pfm)

Step 2: Fitness Evaluation

Determine the fitness of all chromosomes of the population.

Give fitness scores of f(x1), f(x2), etc, f(xN) to each individual.

Determine the most successful chromosomes of the present generation.

Step 3: Selection

Choose parent chromosomes on the basis of their fitness.

More fit individuals are more likely to get chosen.

There are several selection techniques that may be applied at the same time.

Step 5: Crossover (Recombination)

Select parent chromosomes in pairs.

Use Pc crossover operator.

Make babies through genetic interchange among parents.

Children have the characteristics of both parents.

Step 5: Mutation

Mutation operator is applied with probability Pm on offspring.

Variously alter genes in chromosomes randomly to insert new genetic material.

Be diverse in population and allow seeking new solutions.

Step 6: Replacement and Evaluation.

Assess fitness of new children.

Create new people through the fusion of parents and children.

Choose next generation of individuals according to their fitness (generational or steady-state replacement)

Step 7: Termination

Check termination criteria:

- Number of generations achieved maximum.
- Fitness threshold achieved
- Genetic drift (genetic drift is less than half the original population)
- No increase in specified number of generations.

On reaching termination criteria, then return best solution, otherwise go back to Step 3.

Mathematical Formulation:

F(x): the fitness of a chromosome x = (x1, x2, and so on, xn) is defined by the quality of the solution:

Fitness: F(x) -

Pxi: The probability of chromosome xi with fitness fi to be selected.

P(xi) = fi / S(j=1 to N) f j (Roulette Wheel Selection)

Crossover is a hybrid between parent chromosomes xp1 and xp2 to produce offspring xo1 and xo2:

xo1 = axp1 + (1-a)xp2

xo2 = (1-a)xp1 + axp2

Where [?] [0,1] is crossover parameter.

Mutation is a random perturbation of the genes in chromosome x:

xi' = xi + N(0, s2)

N (0, s2) is the Gaussian noise with a mean of 0 and a variance of s2.

Genetic Algorithms have the following advantages:

- **Global Optimization:** GAs explore the full solution space and have fewer chances to be stuck in local optima than gradient-based ones.
- **No Derivative Needed:** GAs do not need derivative information, unlike gradient descent, and thus can be used on poorly-behaving fitness functions.
- **Parallel Processing:** Population-based method can be naturally paralleled making it possible to evaluate many solutions at once.
- **Flexibility:** Is able to deal with different optimization problems such as discrete, continuous, and mixed-variable optimization.
- **Robustness:** Works in noisy conditions and incomplete or inaccurate fitness measures.

Multi-objective Optimization Multi-objective optimization can be generalized to operate with a number of competing objectives.

Genetic algorithms can also be applied to develop optimal clustering configurations in the context of clustering optimization by evolving populations of clustering solutions. Parameters that could be encoded by the chromosome could include the number of clusters, cluster centers or hyperparameters of the algorithm. The fitness function measures the quality of clustering based on measures such as the Silhouette Score which is a measure of cohesion and separation in the cluster. The GA has the ability to search the parameter space using evolutionary mechanisms to find the best possible configurations that maximize the performance of clustering.

3.2 Pseudocode

Pseudocode Pseudocode is a programming language technique of writing algorithms in a combination of the conventions of a programming language and informal, self-explanatory, human-readable natural language notation (Wikipedia, 2025). It is a comprehensive but understandable plan which illustrates the logic and flow of computer programs without following particular syntax of a programming language (TechTarget, n.d.). Basically, pseudocode serves as a middle-ground between the abstract concepts and the literal implementation of the code, enabling developers to concentrate on the logic of the algorithm instead of the specific aspects of a syntax (geeksforgeeks, 2020).

The main aim of writing pseudocode is to assist in the design of the algorithm, its collaboration, and communication among programmers, designers, and other stakeholders despite the knowledge of various programming languages (Metwalli, 2024). It uses typical programming structures like SEQUENCE, IF-THEN-ELSE, WHILE, FOR, and CASE which are usually in capital letters to describe control flow and decision-making procedures (Dalbey, n.d.). Pseudocode allows teams to develop, test and debug algorithmic methods using language-independent representation before spending time on real code

Moreover, the use of pseudocode is especially useful in the academic and professional environment to implement the record of algorithms, technical interviews, and to teach the concepts of programming (iLearnEngineering, 2021). It removes the burden of programming syntax that enables developers to focus on problem-solving techniques and rational thinking. It is this that renders pseudocode as an essential resource to plan, design and communicate computational solutions in a wide range of technical backgrounds.

Pseudocode for Algorithm 1 (DBSCAN Pre-processing)

```
START Algorithm_1_DBSCAN
    IMPORT libraries (pandas, sklearn)
    LOAD raw_transaction_data

    # Data Cleaning
    REMOVE rows where CustomerID is NULL
    REMOVE rows where Quantity < 0 (Returns)
    AGGREGATE data by CustomerID -> CALCULATE Recency,
  Frequency, Monetary (RFM)
    NORMALIZE RFM_Data using StandardScaler (Z-score normalization)

    # Outlier Detection
    DEFINE eps = 0.5, min_samples = 5
    INITIALIZE DBSCAN(eps, min_samples)
    FIT DBSCAN to Normalized_Data

    GET labels (Cluster IDs and -1 for Noise)
    COUNT outliers (where label == -1)

    CREATE Cleaned_Data by filtering out rows where label == -1
    RETURN Cleaned_Data
END
```

Pseudocode for Algorithm 2 (Clustering Ensembles)

```
START Algorithm_2_Clustering
    INPUT: Cleaned_Data, Parameters (Model_Type, k)

    IF Model_Type == "KMeans":
       INITIALIZE KMeans(n_clusters=k, random_state=42)
       FIT KMeans to Cleaned_Data
       LABELS = KMeans.labels_

    ELSE IF Model_Type == "GMM":
       INITIALIZE GaussianMixture(n_components=k, covariance_type='full')
       FIT GMM to Cleaned_Data
       LABELS = GMM.predict(Cleaned_Data)

    CALCULATE Silhouette_Score(Cleaned_Data, LABELS)

    RETURN Silhouette_Score, LABELS
END
```

Pseudocode for Algorithm 3 (Genetic Algorithm Optimization)

```
START  Algorithm_3_GA
        DEFINE Population_Size = 20, Generations = 10
        INITIALIZE Population with random Chromosomes:
           {Model: Random(KMeans, GMM), k: Random(2 to 10)}

        FOR Generation in 1 to Generations:
           FOR each Chromosome in Population:
              Score = CALL Algorithm_2(Cleaned_Data, Chromosome.Model, Chromosome.k)
              ASSIGN Score to Chromosome.Fitness

           SORT Population by Fitness (Descending)
           RETAIN top 50% as Parents

           WHILE Population_Size not full:
              SELECT Parent1, Parent2
              Child.k = Average(Parent1.k, Parent2.k)
              Child.Model = Randomly select from Parents
              MUTATE Child (10% chance to flip Model or change k)
              ADD Child to New_Population
           UPDATE Population = New_Population

        GET Best_Chromosome from Final_Population
        PRINT "Best Segmentation: ", Best_Chromosome
        GENERATE t-SNE Visualization using Best_Chromosome.Labels
END
```

3.3 Diagrammatic Representation

3.3.1 Flowchart and its shapes/components

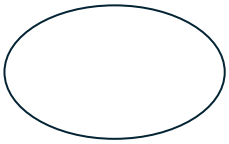
A flow chart is a graphical illustration which helps to show the steps that should be followed to obtain an answer to a problem (IBM, 2022). It incorporates standard symbols; which are linked together with arrows (flow lines) to indicate how the execution flows within a system or a program.

3.3.2 Why Do We Use Flowcharts?

- Improved Communication: Improved explanations of the coding logic to team members or during a presentation will reduce misunderstandings.
- Finding Errors in the Problem: You will be able to catch any problems logically before you start coding.
- Properly Documented: This is a great source of documentation for you when you are working on a project.
- Creating Programs: This will assist your design process in the development of complex algorithms.
- Finding Errors in the Program: When coding, visual aids can make it much easier to track down coding errors.

3.3.3 Flowchart Symbols

2. Terminal/Terminator (Oval/Rounded Rectangle)



Purpose: Indicates the start or end of program.

Usage: Every flowchart must have at least one start and one end

3. Process (Rectangle)

Purpose: Represents any processing operation or computational step

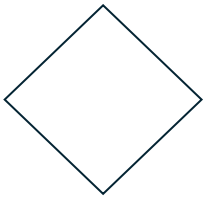
Usage: Calculations, assignments, data manipulation

3. Input/Output (Parallelogram)

Purpose: Shows input or output operations

Usage: Reading data from user, displaying results, file operations

4. **Decision** (Diamond/Rhombus)



Purpose: Represents a decision point or conditional statement

Usage: Used for if-else conditions, loops

3.3.4 Flowchart for algorithm 1

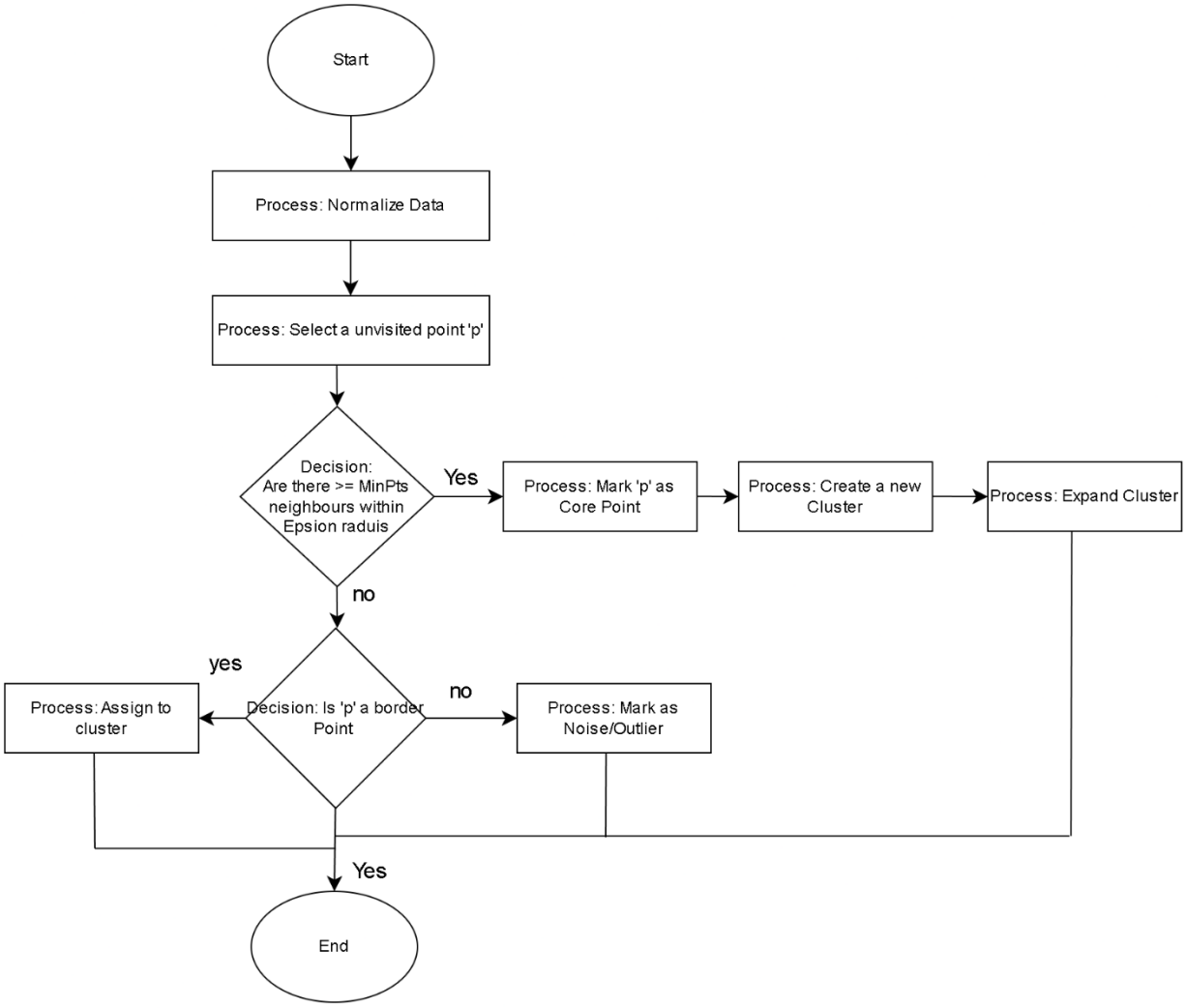


Figure 7 Flowchart for algorithm 1

3.3.5 Flowchart for algorithm 2

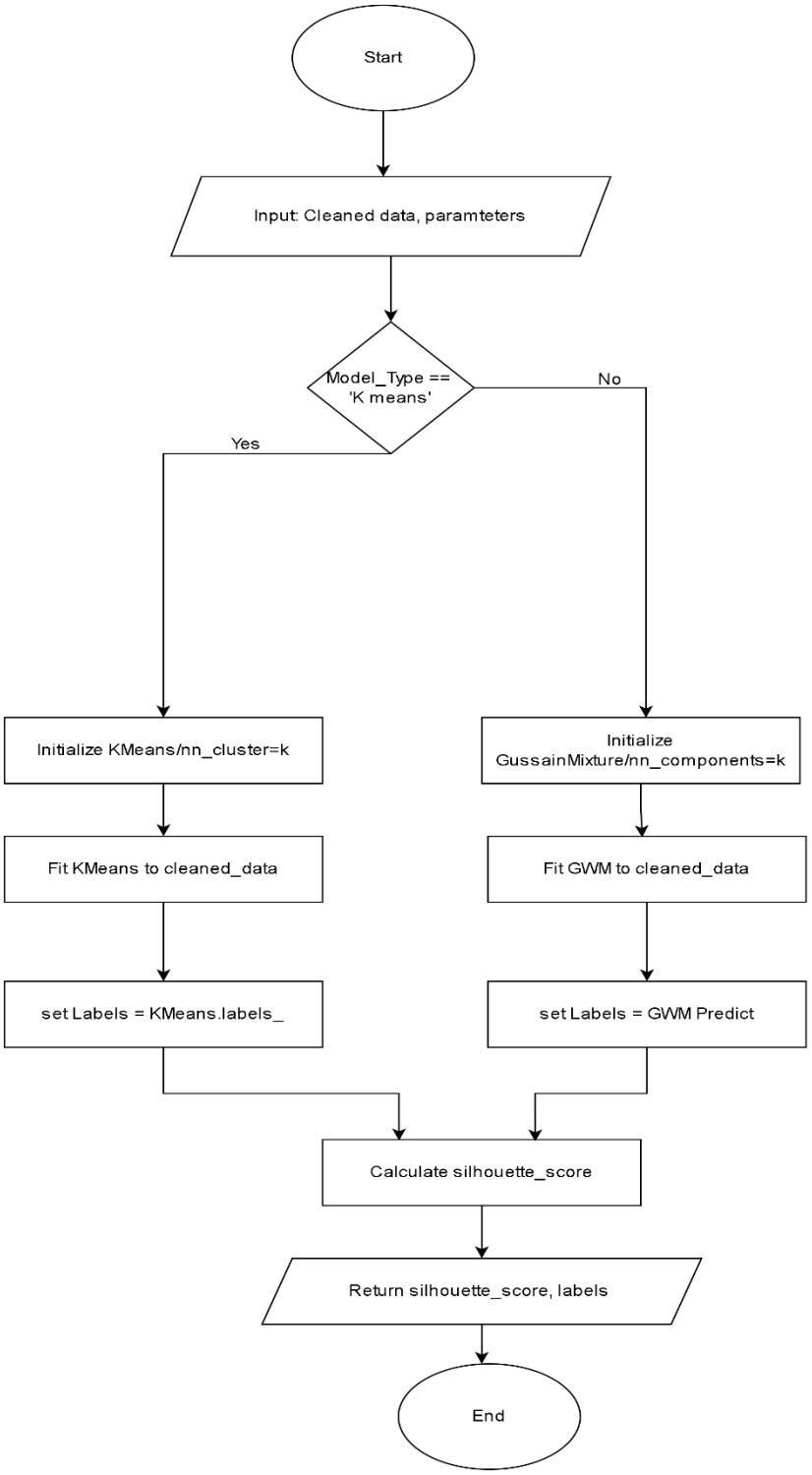


Figure 2 Flowchart for algorithm 2

3.3.6 Flowchart for algorithm 3

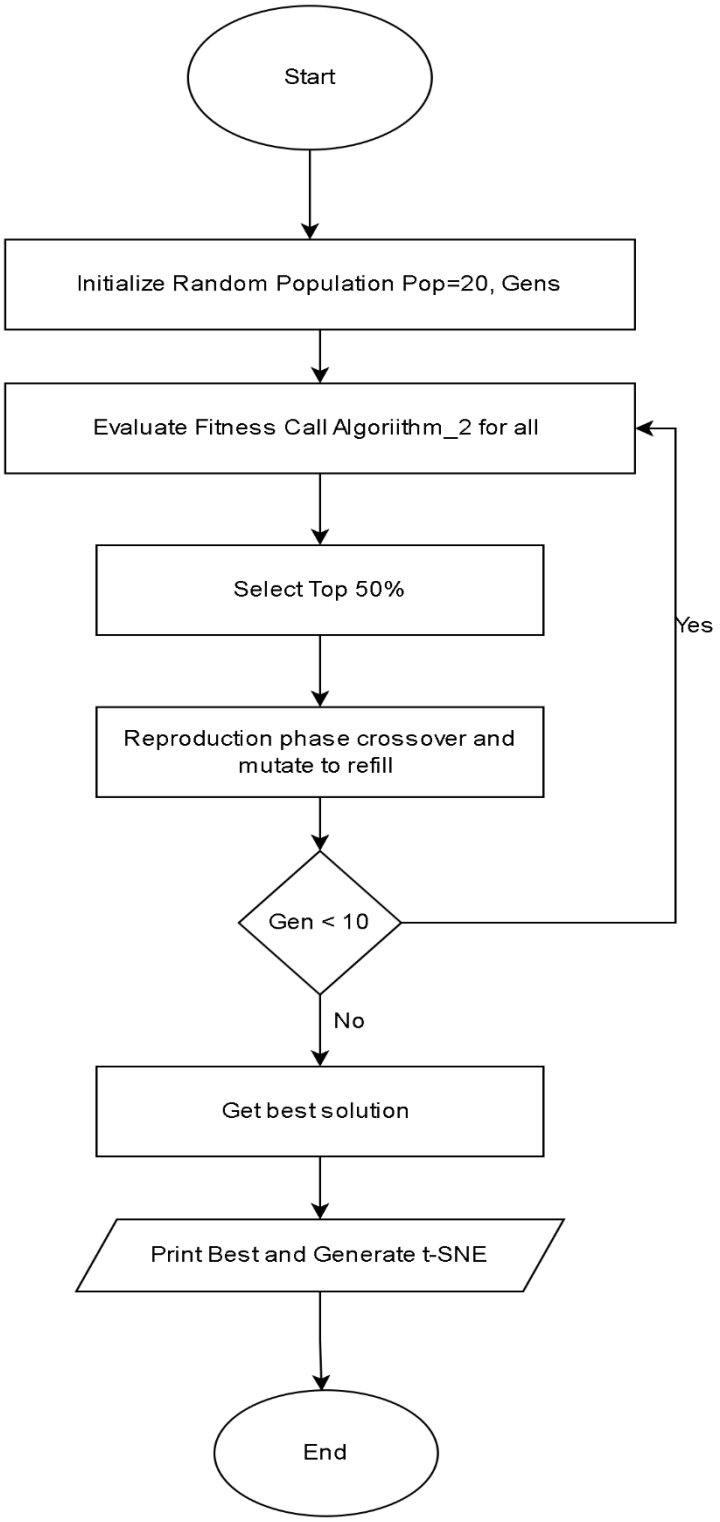


Figure 3 Flowchart for algorithm 3

3.4 Development Process

3.4.1 Tools Used

The development of this project utilized the following tools and technologies. Python was used as the primary programming language, with Jupyter Notebook running on the Anaconda distribution serving as the development environment. Several libraries were employed, including Pandas and NumPy for data handling and analysis, Matplotlib for data visualization, and the Math and Random libraries for mathematical computations and randomization tasks.

3.4.2 Implementation Steps

3.4.2.1 Algorithm 1

Step 1: Load and Clean Data

```
def load_data(file_path):
    """
    Load and clean raw transaction data.
    """
    print("\n" + "="*70)
    print("STEP 1: LOADING AND CLEANING TRANSACTION DATA")
    print("="*70)

    print(f"\nLoading data from: {file_path}")

    # Load the Excel file
    # Note: This might take a moment specific to the file size
    try:
        df = pd.read_excel(file_path, sheet_name='Year 2010-2011')
    except FileNotFoundError:
        print(f"❌ ERROR: Data file not found: {file_path}")
        return None

    print(f"✓ Initial data loaded: {df.shape[0]:,} rows, {df.shape[1]} columns")

    # Data cleaning
    initial_rows = len(df)

    # Remove missing Customer IDs
    df = df.dropna(subset=['Customer ID'])
    print(f"✓ Removed {initial_rows - len(df):,} rows with missing Customer ID")

    # Remove returns (negative quantities)
    initial_rows = len(df)
    df = df[df['Quantity'] > 0]
    print(f"✓ Removed {initial_rows - len(df):,} rows with negative quantities")

    # Remove invalid prices
    initial_rows = len(df)
```

Figure 8 Loading and Cleaning Dataset

Step 2: Calculate RFM Metrics

```python
def calculate_rfm(df):
    """
    Calculate RFM (Recency, Frequency, Monetary) metrics.
    """
    print("\n" + "="*70)
    print("STEP 2: CALCULATING RFM METRICS")
    print("="*70)

    # Convert date column
    df['InvoiceDate'] = pd.to_datetime(df['InvoiceDate'])

    # Reference date (1 day after last transaction)
    reference_date = df['InvoiceDate'].max() + pd.Timedelta(days=1)
    print(f"\nReference date for recency: {reference_date.strftime('%Y-%m-%d')}")

    # Calculate total revenue per transaction
    df['TotalRevenue'] = df['Quantity'] * df['Price']

    print("\nCalculating RFM metrics...")

    # Group by customer and calculate RFM
    rfm = df.groupby('Customer ID').agg({
        'InvoiceDate': lambda x: (reference_date - x.max()).days,  # Recency
        'Invoice': 'nunique',                                      # Frequency
        'TotalRevenue': 'sum'                                      # Monetary
    })

    # Rename columns
    rfm.columns = ['Recency', 'Frequency', 'Monetary']
    rfm = rfm.reset_index()

    print(f"✓ RFM calculated for {len(rfm):,} customers")
```

Figure 9 Calculate RFM Metrics

Step 3: Normalize the data

DBSCAN is distance-based, so we must normalize the features to ensure equal weighting. We use **Z-score normalization**.

+ Code + Markdown

```python
def normalize_data(rfm_data):
    """
    Normalize RFM features using Z-score normalization.
    """
    print("\n" + "="*70)
    print("STEP 3: NORMALIZING RFM FEATURES")
    print("="*70)

    scaler = StandardScaler()

    # Extract features
    features = rfm_data[['Recency', 'Frequency', 'Monetary']].values

    print("\nApplying Z-score normalization...")

    # Apply StandardScaler (Z-score normalization)
    normalized_features = scaler.fit_transform(features)

    print("\n✓ Normalization complete")

    return normalized_features
```

Python

```python
# Execute Step 3
if transaction_data is not None:
    normalized_features = normalize_data(rfm_data)
    print(f"Feature shape: {normalized_features.shape}")
```

Python

Figure 10 Normalize the data

Step 4: Detect Outliers

We apply DBSCAN to identify points that do not belong to any dense cluster. These are labeled as -1.

```python
def detect_outliers(normalized_features, eps, min_samples):
    """
    Apply DBSCAN to detect outliers.
    """
    print("\n" + "="*70)
    print("STEP 4: DETECTING OUTLIERS WITH DBSCAN")
    print("="*70)

    print(f"\nDBSCAN Parameters:")
    print(f"  eps (ε): {eps}")
    print(f"  min_samples: {min_samples}")

    print("\nApplying DBSCAN algorithm...")

    # Initialize and fit DBSCAN
    dbscan_model = DBSCAN(eps=eps, min_samples=min_samples)
    labels = dbscan_model.fit_predict(normalized_features)

    # Count clusters and outliers
    n_clusters = len(set(labels)) - (1 if -1 in labels else 0)
    outlier_count = np.sum(labels == -1)
    outlier_percentage = 100 * outlier_count / len(labels)

    print("\n" + "-"*70)
    print("DBSCAN RESULTS")
    print("-"*70)
    print(f"✓ Number of clusters found: {n_clusters}")
    print(f"✓ Outliers detected: {outlier_count:,} ({outlier_percentage:.2f}%)")

    return labels, outlier_count
```

Python

Figure 11 Detecting Outlier

Step 5: Visualize the data

We create scatter plots to visualize where the outliers are located compared to normal points.

+ Code + Markdown

```python
def visualize_outliers(rfm_data, labels, output_dir):
    """
    Create visualization of detected outliers.
    """
    print("\n" + "="*70)
    print("STEP 6: CREATING VISUALIZATIONS")
    print("="*70)

    # Create output directory
    os.makedirs(output_dir, exist_ok=True)

    # Add labels to dataframe
    rfm_with_labels = rfm_data.copy()
    rfm_with_labels['Is_Outlier'] = labels == -1

    print("\nGenerating 4-panel visualization...")

    # Create figure
    fig, axes = plt.subplots(2, 2, figsize=(16, 12))
    fig.suptitle('DBSCAN Outlier Detection Results', fontsize=16, fontweight='bold')

    # Plot 1: Recency vs Frequency
    ax = axes[0, 0]
    colors = ['red' if x else 'green' for x in rfm_with_labels['Is_Outlier']]
    scatter = ax.scatter(rfm_with_labels['Recency'],
                         rfm_with_labels['Frequency'],
                         c=colors, alpha=0.6, s=50, edgecolors='black', linewidth=0.5)
    ax.set_xlabel('Recency (days since last purchase)', fontsize=11)
    ax.set_ylabel('Frequency (number of purchases)', fontsize=11)
    ax.set_title('Recency vs Frequency', fontsize=12, fontweight='bold')
    ax.grid(alpha=0.3)
```

Figure 12 Visualize the data

3.4.2.2 Algorithm 2

Step 1: Load Clean Dataset

```python
def load_cleaned_data(data_dir):
    """
    Load cleaned data from Algorithm 1.
    """
    print("\n" + "="*70)
    print("STEP 1: LOADING CLEANED DATA FROM ALGORITHM 1")
    print("="*70)

    # Load cleaned RFM data
    rfm_path = os.path.join(data_dir, 'cleaned_rfm_data.csv')
    print(f"\nLoading RFM data from: {rfm_path}")
    if not os.path.exists(rfm_path):
        print(f"❌ ERROR: File not found: {rfm_path}")
        return None, None

    rfm_data = pd.read_csv(rfm_path)
    print(f"✓ Loaded {len(rfm_data):,} customers")

    # Load normalized features
    features_path = os.path.join(data_dir, 'normalized_features.npy')
    print(f"Loading normalized features from: {features_path}")
    if not os.path.exists(features_path):
        print(f"❌ ERROR: File not found: {features_path}")
        return None, None

    normalized_features = np.load(features_path)
    print(f"✓ Loaded features with shape: {normalized_features.shape}")

    print("\nRFM Data Summary:")
    print(rfm_data[['Recency', 'Frequency', 'Monetary']].describe().round(2))

    return rfm_data, normalized_features
```

Python

Figure 13 Load clean Dataset

Step 2: Performing K-Means Clustering

```python
def fit_kmeans(features, n_clusters, random_state):
    """
    Apply K-Means clustering.
    """
    print("\n" + "="*70)
    print("STEP 2: APPLYING K-MEANS CLUSTERING")
    print("="*70)

    print(f"\nK-Means Configuration:")
    print(f"  Number of clusters: {n_clusters}")
    print(f"  Random state: {random_state}")
    print(f"  Max iterations: 300")
    print(f"  Number of initializations: 10")

    print("\nFitting K-Means model...")

    # Initialize and fit K-Means
    kmeans_model = KMeans(
        n_clusters=n_clusters,
        random_state=random_state,
        n_init=10,
        max_iter=300
    )

    kmeans_model.fit(features)
    labels = kmeans_model.labels_

    # Calculate metrics
    silhouette = silhouette_score(features, labels)
    davies_bouldin = davies_bouldin_score(features, labels)
    inertia = kmeans_model.inertia_

    print("\n" + "-"*70)
    print("K-MEANS RESULTS")
    print("-"*70)
    print(f"✓ Silhouette Score: {silhouette:.4f}")
```

Figure 14 K_Means Clustering

Step 3: Gaussian Mixture Model (GMM)

```python
def fit_gmm(features, n_clusters, random_state):
    """
    Apply Gaussian Mixture Model clustering.
    """
    print("\n" + "="*70)
    print("STEP 3: APPLYING GAUSSIAN MIXTURE MODEL (GMM)")
    print("="*70)

    print(f"\nGMM Configuration:")
    print(f"  Number of components: {n_clusters}")
    print(f"  Covariance type: full")
    print(f"  Random state: {random_state}")
    print(f"  Max iterations: 100")

    print("\nFitting GMM model...")

    # Initialize and fit GMM
    gmm_model = GaussianMixture(
        n_components=n_clusters,
        covariance_type='full',
        random_state=random_state,
        n_init=10,
        max_iter=100
    )

    gmm_model.fit(features)
    labels = gmm_model.predict(features)

    # Calculate metrics
    silhouette = silhouette_score(features, labels)
    davies_bouldin = davies_bouldin_score(features, labels)
    log_likelihood = gmm_model.score(features) * len(features)
    bic = gmm_model.bic(features)
    aic = gmm_model.aic(features)

    print("\n" + "-"*70)
    print("GMM RESULTS")
```

Figure 15 Gaussian Mixture Model (GMM)

Step 4: Compare Models

```python
def compare_models(features, kmeans_labels, gmm_labels):
    """
    Compare K-Means and GMM performance.
    """
    print("\n" + "="*70)
    print("STEP 4: COMPARING K-MEANS VS GMM")
    print("="*70)

    # Calculate metrics for both models
    kmeans_silhouette = silhouette_score(features, kmeans_labels)
    gmm_silhouette = silhouette_score(features, gmm_labels)

    kmeans_db = davies_bouldin_score(features, kmeans_labels)
    gmm_db = davies_bouldin_score(features, gmm_labels)

    print("\nPerformance Comparison:")
    print("-" * 70)
    print(f"{'Metric':<30} {'K-Means':<20} {'GMM':<20}")
    print("-" * 70)
    print(f"{'Silhouette Score':<30} {kmeans_silhouette:<20.4f} {gmm_silhouette:<20.4f}")
    print(f"{'Davies-Bouldin Index':<30} {kmeans_db:<20.4f} {gmm_db:<20.4f}")
    print("-" * 70)

    # Determine winner
    if kmeans_silhouette > gmm_silhouette:
        print("\n⏳ K-Means has better silhouette score")
    elif gmm_silhouette > kmeans_silhouette:
        print("\n⏳ GMM has better silhouette score")
    else:
        print("\n🤝 Both models perform equally")

    if kmeans_db < gmm_db:
        print("⏳ K-Means has better Davies-Bouldin index")
    elif gmm_db < kmeans_db:
        print("⏳ GMM has better Davies-Bouldin index")
```

Figure 16 Comparing the Models

Step 5: Cluster Statistics

```python
def calculate_cluster_statistics(rfm_data, labels):
    """
    Calculate statistics for each cluster.
    """
    print("\n" + "="*70)
    print("STEP 5: CALCULATING CLUSTER STATISTICS")
    print("="*70)

    # Add labels to dataframe
    rfm_with_labels = rfm_data.copy()
    rfm_with_labels['Cluster'] = labels

    print("\nCalculating mean, min, max for each cluster...")

    # Calculate statistics
    stats = rfm_with_labels.groupby('Cluster').agg({
        'Recency': ['mean', 'min', 'max', 'std'],
        'Frequency': ['mean', 'min', 'max', 'std'],
        'Monetary': ['mean', 'min', 'max', 'std'],
        'Customer ID': 'count'
    }).round(2)

    # Flatten column names
    stats.columns = ['_'.join(col).strip() for col in stats.columns.values]
    stats = stats.rename(columns={'Customer ID_count': 'Customer_Count'})

    print("\n" + "-"*70)
    print("CLUSTER STATISTICS")
    print("-"*70)
    display(stats)

    return stats
```

Python

Figure 17 Cluster Statistics

Step 6: Segment Naming

We assign business-relevant names to each cluster based on their RFM profile (e.g., "Champions", "Hibernating").

```python
def assign_segment_names(rfm_data, labels):
    """
    Assign meaningful business names to clusters.
    """
    print("\n" + "="*70)
    print("STEP 6: ASSIGNING SEGMENT NAMES")
    print("="*70)

    # Add labels to dataframe
    rfm_with_labels = rfm_data.copy()
    rfm_with_labels['Cluster'] = labels

    # Calculate mean RFM for each cluster
    cluster_means = rfm_with_labels.groupby('Cluster')[['Recency', 'Frequency', 'Monetary']].mean()

    print("\nCluster Profiles (Mean RFM):")
    print(cluster_means.round(2))

    print("\nAssigning segment names based on RFM profiles...")

    segment_names = {}

    for cluster_id in cluster_means.index:
        recency = cluster_means.loc[cluster_id, 'Recency']
        frequency = cluster_means.loc[cluster_id, 'Frequency']
        monetary = cluster_means.loc[cluster_id, 'Monetary']

        # Normalize values (0-1 scale)
        recency_norm = (recency - cluster_means['Recency'].min()) / (cluster_means['Recency'].max() - cluster_means
        ['Recency'].min())
        frequency_norm = (frequency - cluster_means['Frequency'].min()) / (cluster_means['Frequency'].max() -
        cluster_means['Frequency'].min())
        monetary_norm = (monetary - cluster_means['Monetary'].min()) / (cluster_means['Monetary'].max() - cluster_means
```

Figure 18 Segment Naming

3.4.2.3 Algorithm 3

Step 1: Genetic Algorithm Functions

```python
def initialize_population(pop_size, min_k, max_k):
    """Initialize random population of K values."""
    return [random.randint(min_k, max_k) for _ in range(pop_size)]

def fitness_function(k, features, random_state):
    """Calculate fitness for a given K."""
    try:
        kmeans = KMeans(n_clusters=k, random_state=random_state, n_init=10)
        labels = kmeans.fit_predict(features)

        silhouette = silhouette_score(features, labels)
        db_index = davies_bouldin_score(features, labels)
        ch_index = calinski_harabasz_score(features, labels)
        inertia = kmeans.inertia_

        # Fitness: maximize Silhouette, minimize DB index
        fitness = silhouette - (db_index / 10) + (ch_index / 10000)

        metrics = {
            'k': k,
            'silhouette': silhouette,
            'db_index': db_index,
            'ch_index': ch_index,
            'inertia': inertia,
            'fitness': fitness
        }
        return fitness, metrics
    except:
        return -float('inf'), {}

def selection(population, fitness_results):
    """Tournament selection."""
    selected = []
    tournament_size = 3
    for _ in range(len(population)):
        indices = random.sample(range(len(population)), min(tournament_size, len(population)))
```

Figure 19 Genetic Algorithm Functions

Step 2: Run Evolution

```python
def run_evolution(features, params):
    """
    Run the genetic algorithm evolution.
    """
    print("\n" + "="*70)
    print("STEP 3: RUNNING GENETIC ALGORITHM")
    print("="*70)

    # Initialize state
    random.seed(params['random_state'])
    population = initialize_population(params['pop_size'], params['min_k'], params['max_k'])

    history = {
        'generation': [], 'best_fitness': [], 'avg_fitness': [],
        'best_k': [], 'best_silhouette': [], 'best_db_index': []
    }

    best_solution = None
    best_fitness = -float('inf')
    best_metrics = {}

    print(f"\n✓ Initial population: {population}")
    print("\n" + "-"*70)
    print("EVOLUTION PROGRESS")
    print("-"*70)

    for gen in range(params['generations']):
        # Evaluate
        fitness_results = []
        for k in population:
            fitness_results.append(fitness_function(k, features, params['random_state']))

        fitness_values = [f[0] for f in fitness_results]
        best_idx = np.argmax(fitness_values)
        gen_best_fitness = fitness_values[best_idx]
```

Figure 20 Run Evolution

Step 4: Compare with Defaults

```python
def compare_with_default(features, best_k, best_metrics, default_k=5, random_state=42):
    print("\n" + "="*70)
    print("STEP 4: COMPARING GA-OPTIMIZED VS DEFAULT")
    print("="*70)

    # Get metrics for default K
    _, default_metrics = fitness_function(default_k, features, random_state)

    print(f"\n{'Metric':<30} {'Default (K={})'.format(default_k):<20} {'Optimized (K={})'.format(best_k):<20}
    {'Improvement':<15}")
    print("-" * 85)

    default_sil = default_metrics.get('silhouette', 0)
    optimal_sil = best_metrics.get('silhouette', 0)
    sil_imp = ((optimal_sil - default_sil) / abs(default_sil) * 100) if default_sil != 0 else 0
    print(f"{'Silhouette Score':<30} {default_sil:<20.4f} {optimal_sil:<20.4f} {sil_imp:+.1f}%")

    default_db = default_metrics.get('db_index', 0)
    optimal_db = best_metrics.get('db_index', 0)
    db_imp = ((default_db - optimal_db) / abs(default_db) * 100) if default_db != 0 else 0
    print(f"{'Davies-Bouldin Index':<30} {default_db:<20.4f} {optimal_db:<20.4f} {db_imp:+.1f}%")
```

Python

Figure 21 Compare with defaults

4. Evolution Of Model

4.1 Evolution Metrics

Because the problem domain is the customer segmentation, the model is assessed in terms of the following Key Performance Indicators (KPIs):

Silhouette Score (Clustering Quality): This is the major evaluation measure. It is a measure of the similarity of one object with its own cluster (cohesion) and other clusters (separation). This value is between -1 and 1 whereby the higher the value the better defined clusters will be.

Davies-Bouldin Index (Cluster Separation): This is used to measure the mean similarity of each cluster to the most similar cluster. Lower values are an indication of improved separation among clusters.

Execution Time (Computational Efficiency): The time that the algorithm uses to arrive at a solution is called the execution time. The execution time will be in seconds.

Optimality Gap: The metric indicates the proximity of the solution to that of the baseline which we have maintained at the standard K-Means algorithm with K=5.

4.2 Comparative Analysis

Three clustering models were evaluated and compared based on segmentation quality metrics and computational efficiency.

Table 1 Analysis of All Models

Model	Silhouette Score	Davies-Bouldin Index	Execution Time
K-Means (K=5)	0.4738	0.7015	3.71s
GMM (K=5)	0.0076	3.2239	1.30s
GA-Optimized (K=2)	**0.6072**	**0.5642**	303.92s

All models were evaluated against the baseline K-Means (K=5) clustering approach to assess improvements in segmentation quality and computational efficiency.

Table 2 Comparative Performance Analysis

Algorithm	Silhouette Score Improvement	Execution Time	Status
K-Means (Baseline)	—	3.71s	Efficient
GMM	-98.39%	1.30s	Fast but Inaccurate
GA-Optimized	**+28.15%**	303.92s	High Quality, Slow

Key Findings

Genetic Algorithm Optimization: The Genetic Algorithm was able to find an optimal amount of clusters (K=2) which resulted in a significant increase of the quality of segmentation by 28.15 in Silhouette Score over the default K-Means approach. This proves the usefulness of evolutionary optimization methodology when addressing the issue of which customer segments to have.

Computational-Trade-offs: The higher quality of the clustering was at a high computational price since the time of execution was 3.7 seconds to around 5 minutes (303.92 seconds). This is a 82 times increase in processing time, which shows the traditional trade-off between the quality of solutions and their computational efficiency.

GMM Performance: The Gaussian Mixture Model was not doing well with this particular high-dimensional RFM data as it had a Silhouette Score of 0.0076. This implies that the customer clusters in the data are probably circular instead of elliptical shaped or that GMM was having difficulties with the density distribution nature of the RFM feature space.

4.3 Simulation Results

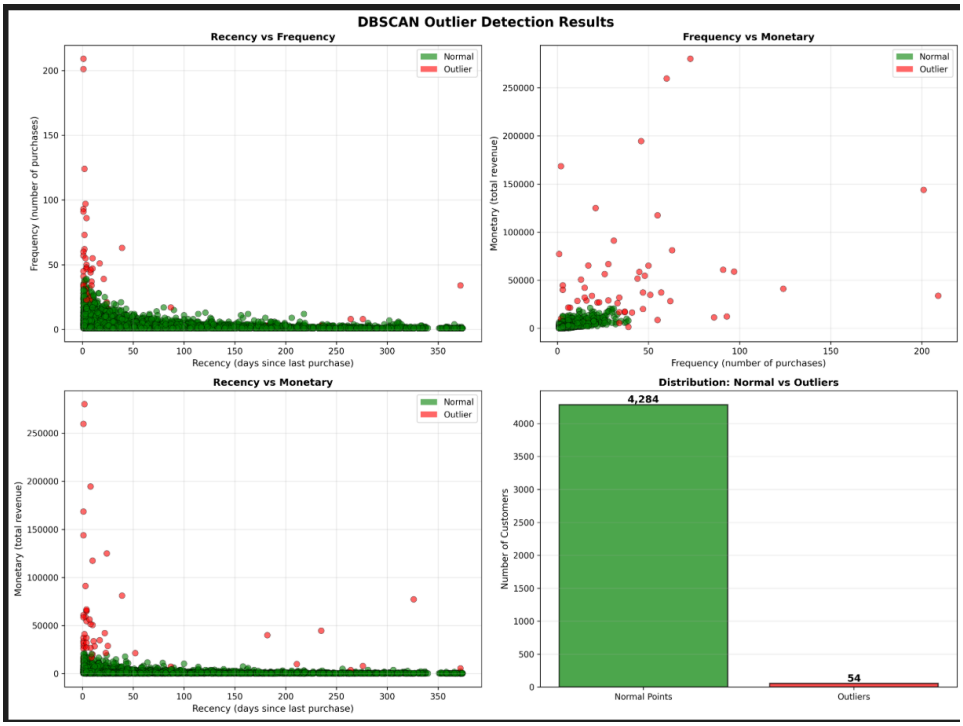


Figure 22 DBSCAN Outlier Detection Results

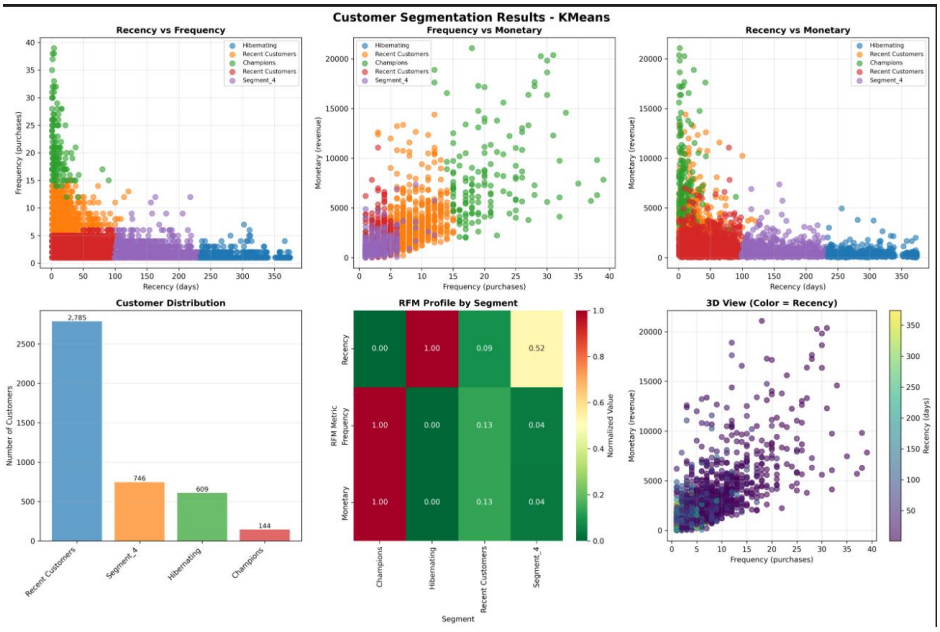


Figure 23 Customer Segmentation Results

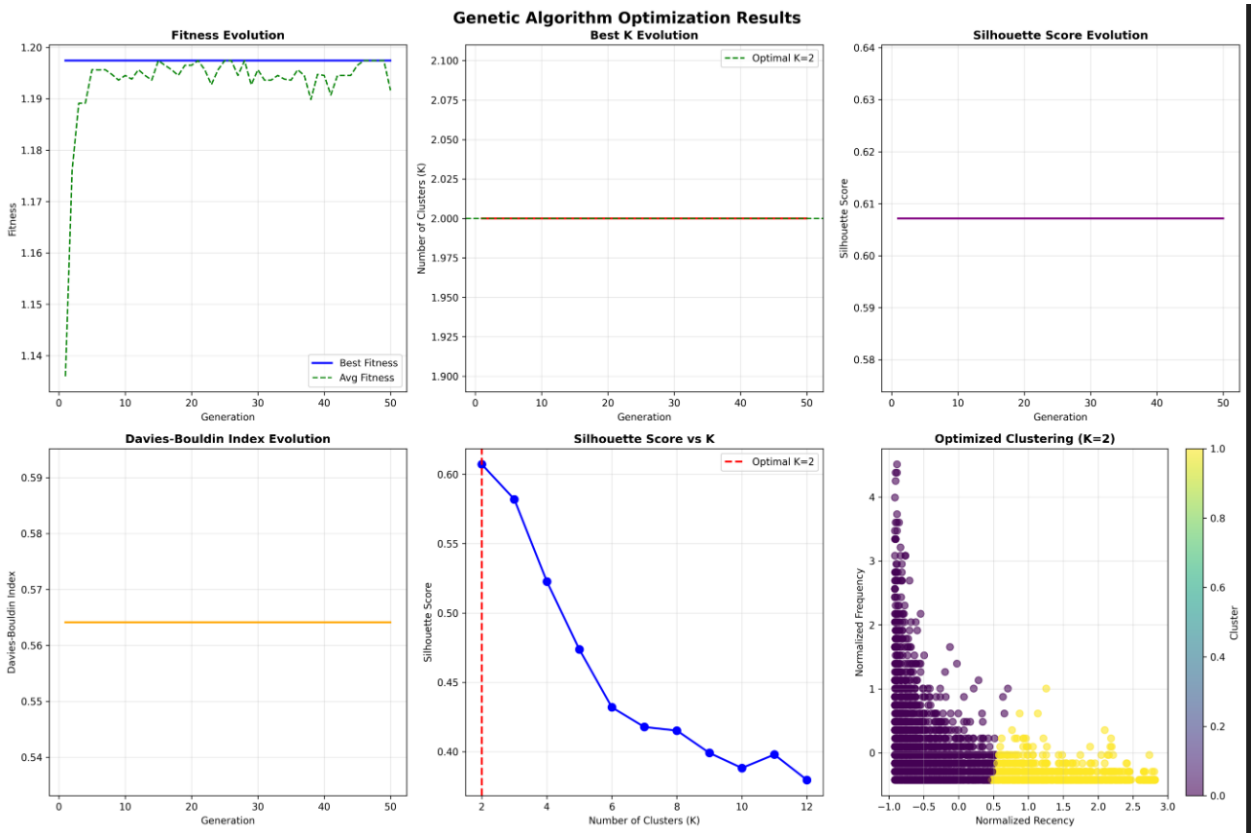


Figure 24 Genetic Algorithm Optimization Results

5. Conclusion

The modern world of the online retail market is distinguished by a huge amount of data and tough rivalry. The main issue of this proposal is the impossibility of the traditional, static market segmentation, which can hardly reflect the complexity of the contemporary consumer behavior because of the noise of the data and similar behavioral characteristics. In turn, the main aim of this coursework is to design a robust and automatic Hybrid Evolutionary Segmentation System. The goal of this system is to overcome the shortcomings of manual analysis, where more sophisticated Machine Learning methods are used to create mathematical distinction between high-value and at-risk groups of customers.

In order to accomplish this, the proposed solution is the use of an advanced multi-stage pipeline. Contrary to the common methods, which use only the simple clustering, this methodology incorporates DBSCAN as a filter applied in the course of pre-processing to filter out statistical outliers that normally interfere with results. This is then succeeded by an Evolutionary Optimization engine with a Genetic Algorithm that dynamically compares and chooses the best clustering configuration- alternating between K-Means and Gaussian Mixture Models (GMM) according to the Silhouette Score. The hybrid approach will ensure that the final model is not a simple product of trial-and-error, but rather an optimized approach that will fit the unique structure of the data.

The desired result is a working Python-based prototype that will be able to convert raw transaction logs into business intelligence that will be put to good use. The system will provide unique, valid segments of customers, i.e., "Champions," "Hibernating," and "New entrants" and confirmed with large internal validity scores. In addition, t-SNE visualization will enable the stakeholders with a simple 2D depiction of such intricate groups. Finally, the project shows that density-based cleaning and evolutionary computing can greatly contribute to Customer Relationship Management (CRM) plans that should increase its retention rates and maximize its marketing ROI.

6. References

A. Joy Christy, A. Umamakeswari , L. Priyatharsini, A. Neyaa. (2021) RFM ranking – An effective approach to customer segmentation. *Journal of King Saud University - Computer and Information Sciences*, 33(10), p.1251.

Agnieszka Nowak-Brzezinska, T.X. (2017) Outlier Mining Using the DBSCAN Algorithm. *JOURNAL OF APPLIED*, 25(2), p.68.

Eva Patel, D.S.K. (2020) Clustering Cloud Workloads: K-Means vs Gaussian Mixture Model. *Third International Conference on Computing and Network Communications* , 171(10), p.167.

geeksforgeeks. (2020) *What is PseudoCode: A Complete Tutorial* [Online]. Available from: https://www.geeksforgeeks.org/dsa/what-is-pseudocode-a-complete-tutorial/ [Accessed 15 December 2025].

IBM. (2022) *What is a flowchart?* [Online]. Available from: https://www.ibm.com/think/topics/flowchart [Accessed 10 December 2025].

IBM. (2023) *What is artificial intelligence (AI)?* [Online]. Available from: https://www.ibm.com/think/topics/artificial-intelligence [Accessed 10 Decemeber 2025].

iLearnEngineering. (2021) *What is pseudocode and why do we use it in programming?* [Online]. Available from: https://www.ilearnengineering.com/computer/what-is-pseudocode-and-why-do-we-use-it-in-programming [Accessed 16 December 2025].

Meduim. (2023) *Customer Segmentation with RFM Analysis and K-Means Clustering using the Online Retail Dataset* [Online]. Available from: https://medium.com/@nkc_53945/customer-segmentation-with-rfm-analysis-and-k-means-clustering-using-the-online-retail-dataset-df2de395bdfc [Accessed 10 December 2025].