UNIVERSIDAD DE CHILE
FACULTAD DE CIENCIAS FÍSICAS Y MATEMÁTICAS
DEPARTAMENTO DE INGENIERÍA INDUSTRIAL

REINFORCEMENT LEARNING APPLIED TO DYNAMIC CLUSTERING

TESIS PARA OPTAR AL GRADO DE
MAGÍSTER EN GESTIÓN DE OPERACIONES

MEMORIA PARA OPTAR AL TÍTULO DE
INGENIERO CIVIL INDUSTRIAL

IGNACIO NICOLÁS CARVAJAL CÁCERES

PROFESOR GUÍA:
RICHARD WEBER HAAS

MIEMBROS DE LA COMISIÓN:
RAMIRO SALTOS ATIENCIA
DENIS SAURÉ VALENZUELA

SANTIAGO DE CHILE
2024

Aprendizaje por refuerzo aplicado al clustering dinámico

El clustering es una técnica esencial en el reconocimiento de patrones, la minería de datos y el descubrimiento de conocimiento. Un desafío significativo en el clustering dinámico es predecir los cambios en la estructura subyacente de los datos, como la segmentación futura de clientes. Este problema se complica especialmente cuando se trabaja con datos multimodales, ya que es necesario estudiar los cambios en la estructura de los datos a lo largo del tiempo. Este documento propone utilizar Gradientes de Política Determinística Profunda Multiagente (MADDPG) y el Modelo de Mezcla Gaussiana (GMM) para resolver el problema del clustering dinámico. El GMM se emplea para representar una mezcla de distribuciones de probabilidad, considerando los clusters (componentes) de GMM como agentes en un juego de Markov parcialmente observable. Los agentes se entrenan con MADDPG, una extensión del algoritmo DDPG diseñada para entornos multiagentes, que permite a los agentes aprender políticas descentralizadas y coordinarse entre sí. El objetivo principal de este trabajo es predecir los parámetros de GMM del próximo período utilizando la información del período actual. Durante el entrenamiento, cada agente observa los estados y acciones de todos los agentes y aprende un crítico centralizado para estimar el valor de la acción conjunta. En la fase de ejecución, cada agente utiliza solo sus observaciones locales para seleccionar acciones, buscando optimizar la log-verosimilitud obtenida con los parámetros predichos al clusterizar los datos en el próximo período. El documento demuestra que el enfoque propuesto puede predecir eficazmente los parámetros de GMM para períodos futuros bajo condiciones de movimientos lineales y estacionarios de los clusters, mejorando la capacidad de predecir la estructura subyacente de los datos en contextos dinámicos comparado con solo confiar en la clusterización del periodo actual.

Reinforcement Learning applied to Dynamic Clustering

Clustering is an essential technique in pattern recognition, data mining, and knowledge discovery. A significant challenge in dynamic clustering is predicting changes in the underlying structure of the data, such as future customer segmentation. This problem is especially complex when dealing with multimodal data, as it is necessary to study changes in the data structure over time. This paper proposes using Multi-Agent Deep Deterministic Policy Gradients (MADDPG) and the Gaussian Mixture Model (GMM) to address the issue of dynamic clustering. GMM is employed to represent a mixture of probability distributions, considering the clusters (components) of GMM as agents in a partially observable Markov game. The agents are trained using MADDPG, an extension of the DDPG algorithm designed for multi-agent environments, which allows the agents to learn decentralized policies and coordinate with each other.

The primary objective of this work is to predict the GMM parameters for the next period using information from the current period. During training, each agent observes the states and actions of all other agents and learns a centralized critic to estimate the value of the joint action. In the execution phase, each agent uses only its local observations to select actions, aiming to optimize the log-likelihood obtained with the predicted parameters when clustering the data in the next period.

The paper demonstrates that the proposed approach can effectively predict GMM parameters for future periods under conditions of linear and stationary movements of the clusters, improving the ability to predict the underlying structure of the data in dynamic contexts compared to relying solely on the clustering of the current period.

A mis padres, novia y amigos. Los quiero.

Table of Content

List of Tables

List of Figures

Chapter 1

Introduction

In this section, we outline our proposed approach to dynamic clustering using Reinforcement Learning (RL) techniques. We focus on leveraging the capabilities of the Gaussian Mixture Model (GMM) for probabilistic modeling and Multi-Agent Deep Deterministic Policy Gradients (MADDPG) for adaptive decision-making. The goal is to develop a robust framework that can accurately predict and adapt to evolving cluster structures in dynamic environments. We begin by detailing the fundamentals of GMM and MADDPG, followed by an explanation of how these methods are integrated to address the challenges of dynamic clustering. Lastly, we describe our evaluation strategy and highlight the expected contributions of our approach.

1.1 Background

Artificial intelligence has emerged as one of the most crucial domains within data science today. With each passing day, its algorithms are tackling new tasks and applications. Thanks to the vast availability of data and demonstrated effectiveness, this field has seen a surge in interest both in academia and the private sector in recent years.

Since supervised learning struggles when we lack a defined ground truth, as seen in [6], its application becomes challenging. This is primarily due to the necessity of vast amounts of labeled data, each element requiring costly tagging. In such cases, unsupervised learning emerges as a promising solution. In this sense, unsupervised learning where we don't have information about the desired output seems a solution for part of these cases. In particular, when we need to have a segmentation of the data without knowing which data point corresponds to each group, we think of clustering. But, what is clustering for? "Clustering is vital in pattern recognition, data mining, and knowledge discovery to unveil the underlying structure of objects [6].

Most real-world situations are dynamic. Weather, traffic, and human behavior are just a handful of examples where the attributes that determine a specific phenomenon change over time, increasing the importance of dynamic clustering for researchers and practitioners. Dynamic clustering is an evolving field that focuses on adapting clustering algorithms to

handle changing data distributions over time. Traditional clustering techniques assume static data, but in real-world scenarios, data streams, concept drift, and evolving environments pose challenges to maintaining accurate and up-to-date cluster representations. The state of the art in dynamic clustering involves various approaches and techniques to address these challenges.

One key aspect of dynamic clustering is concept drift adaptation. Concept drift refers to the phenomenon where the statistical properties of the data change over time, leading to shifts in the underlying clustering structures. Researchers have developed methods to detect and track concept drift, allowing clustering models to adapt accordingly [13].

Another important direction in dynamic clustering is the development of incremental clustering algorithms. Instead of recomputing clusters from scratch as new data arrives, incremental algorithms update existing clusters, add new clusters, or remove obsolete clusters. These algorithms reduce computational complexity and memory requirements while maintaining the clustering accuracy [6, 9]. Additionally, there is growing interest in online clustering, which deals with clustering data streams in real-time. Online clustering algorithms process data instances one at a time and continuously update the cluster representations. These algorithms often employ approximation techniques and data summarization to handle the high-speed and large-volume nature of data streams efficiently [22]. Furthermore, researchers have focused on hybrid approaches that combine multiple clustering algorithms to leverage their complementary strengths. For instance, integrating density-based clustering with centroid-based clustering techniques can enhance the ability to detect clusters of varying shapes and densities in dynamic environments [2].

Reinforcement learning (RL) offers a promising approach for tackling dynamic clustering problems. RL agents learn to make sequential decisions through interactions with an environment, optimizing a reward signal to achieve desired objectives. By applying RL to dynamic clustering, we can develop algorithms that adaptively adjust cluster assignments and cluster centers as new data arrives, facilitating the discovery of evolving patterns and clusters.

The application of RL to dynamic clustering poses several challenges and opens up exciting research opportunities. First, we need to design appropriate state representations that capture the evolving nature of the data and cluster assignments. Additionally, the RL agent must determine suitable actions to update the clusters while balancing exploration and exploitation trade-offs. The choice of reward function is crucial, as it guides the agent's learning process and influences the quality of the resulting clusters. Furthermore, scalability and efficiency are essential considerations to ensure the practicality of RL-based dynamic clustering algorithms.

Reinforcement learning has emerged as a promising technique in dynamic clustering. By formulating clustering as a reinforcement learning problem, algorithms can learn to adapt and optimize the clustering process based on feedback and rewards. This approach enables the discovery of cluster structures while simultaneously addressing the challenge of initialization [3].

Finally, researchers have explored the use of fuzzy clustering in dynamic scenarios. Fuzzy clustering allows data points to belong to multiple clusters with different degrees of membership,

providing flexibility to handle changing data distributions. Fuzzy clustering algorithms have been extended to incorporate time-varying parameters and adaptive strategies to track evolving clusters [9, 22].

In summary, the state of the art in dynamic clustering encompasses various techniques such as concept drift adaptation, incremental clustering, online clustering, hybrid approaches, reinforcement learning, and fuzzy clustering. These approaches aim to handle the challenges posed by changing data distributions and evolving environments. The field continues to evolve as researchers develop novel algorithms and adapt existing methods to meet the demands of dynamic clustering in real-world applications.

1.2 Objectives

This research delves into the application of reinforcement learning (RL) techniques to tackle dynamic clustering challenges. Leveraging the sequential decision-making capabilities inherent in RL, our aim is to craft algorithms that can adaptively and efficiently cluster evolving data. This study seeks to make significant contributions by investigating tailored state representations, reward functions, and action selection strategies designed specifically for dynamic clustering scenarios.

1.2.1 General Objective:

1. To pioneer a novel approach to Dynamic Clustering aimed at forecasting clustering parameters in subsequent periods based on the available information up to the present moment.

1.2.2 Specific Objectives:

1. To evaluate the effectiveness of the proposed method.
2. To discern the strengths and limitations of the proposed method.

This thesis investigates the application of Multi-Agent Deep Deterministic Policy Gradients (MADDPG) and Gaussian Mixture Model (GMM) in multi-modal distribution prediction. We aim to evaluate the effectiveness of MADDPG and GMM in predicting the distribution of multiple modes handling the problem of data drift and promising new approaches for other events like the creation and elimination of clusters.

1.3 Proposal

Gaussian Mixture Model (GMM) is a probabilistic model that is used to represent a mixture of different probability distributions. It is a clustering algorithm that attempts to find the

most likely distribution of points within a given space, based on the data provided. It can be used for both supervised and unsupervised learning tasks. Basically, GMM is a type of probability density estimation that uses a weighted sum of multiple Gaussian distributions to approximate the underlying data distribution. We can define each Gaussian distribution with a mean μ and a covariance matrix Σ, so the challenge is to find a good approximation for these parameters for each cluster or component of the underlying distribution. The idea of using GMM for distribution prediction is not new, in [32] is presented a variational inference semi-supervised GMM method in which is necessary the use of labeled data. GMM has been used for prediction strategy for dynamic multi-objective optimization problems [30] and other applications as prediction methodology for dam deformation considering spatiotemporal differentiation [8] where a novel forecast algorithm was proposed. Although the objectives are different, the principal idea is to approximate the parameters μ, Σ, w for each component in the future. In this work, we will attack this problem as a multi-agent semi-observable Markov game. In other words, in our approach, we are going to see each component or cluster as an agent whose purpose is to output the variation of each parameter between each period. The manner in which each agent converges to a policy will be using MADDPG with communication.

MADDPG stands for Multi-Agent Deep Deterministic Policy Gradient, and it is an extension of the DDPG algorithm for multi-agent environments. DDPG is a popular algorithm for reinforcement learning in continuous action spaces. DDPG is an extension of the popular Deep Q-Network (DQN) algorithm that is used for reinforcement learning with discrete action spaces. DDPG uses an actor-critic architecture, where the actor learns a deterministic policy that maps states to actions, and the critic learns an estimate of the action-value function. The actor and critic networks are trained using the policy gradient method, where the gradients are estimated using the critic network. One of the key innovations of DDPG is the use of a replay buffer to store and sample experiences from the agent's interactions with the environment. This allows the agent to learn from past experiences and improves the stability of the learning process. In the multi-agent setting, MADDPG extends DDPG to enable agents to learn decentralized policies that can coordinate with each other.

MADDPG operates on the principle of centralized training and decentralized execution. This means that during the training process, each agent can observe the states and actions of all agents in the environment, and can use this information to learn a centralized critic that estimates the value of the joint action taken by all agents. However, during execution, each agent only has access to its own local observations and must use its learned policy to select actions.

In MADDPG, each agent has its own actor and critic network, and the actor networks are trained using the centralized critic. During the training process, the agents share their experiences with each other and update their actor and critic networks using the MADDPG algorithm, which is based on the DDPG algorithm but includes additional terms for coordinating agents. Overall, MADDPG enables agents to learn decentralized policies that can coordinate with each other, which is essential for many real-world multi-agent applications.

This work will use this method to learn the variation of GMM parameters between periods. The main idea is to predict the GMM parameters of the next period with the information of the current period. To do that, each agent sees the parameters of m past periods and the

action of each agent and optimizes the log-likelihood obtained with the predicted parameters clustering the data in the next period.

To evaluate the effectiveness of the proposed RL-based approach, we will conduct experiments on various datasets that exhibit dynamic characteristics, such as evolving cluster structures and time-varying data distributions. We will compare the performance of our approach with state-of-the-art static clustering algorithms, highlighting the advantages and limitations of RL in handling dynamic clustering tasks.

The main contribution of this thesis is to propose a novel approach for predicting the clustering of future periods by considering the dynamics of these periods in the feature space in past periods. Additionally, it is demonstrated the capability of this method to deal with specific dynamics while also identifying its limitations and providing recommendations for enhancing performance when using it. Furthermore, avenues for further research to improve it are outlined.

The remainder of this thesis is organized as follows. Section 2 provides an overview of related work in the fields of clustering and reinforcement learning. Section 3 describes the proposed RL-based dynamic clustering framework, including the formulation of state, action, and reward components. Section 4 presents experimental results and performance evaluation. Finally, Section 5 concludes the paper and discusses future research directions.

Chapter 2

Background

This chapter will provide a comprehensive overview of the theoretical foundations of Partially Observable Markov Games, Multi-Agent Deep Deterministic Policy Gradient, and Gaussian Mixture Models. We will discuss the key concepts and techniques involved in these frameworks in order to have a Background of the techniques required for our approach.

2.1 Gaussian Mixture Model

A Gaussian mixture model assumes that the data comes from a mixture of several (m) Gaussian distributions, each with its own weight w_l, mean μ_l, and covariance matrix Σ_l where l represents the cluster. The mixture of Gaussians is used to approximate the probability density function of the observed data. In other words, we are approximating the function:

$$q(x, \theta) = \sum_{\ell=0}^{m} w_\ell N(x; \mu_\ell, \Sigma_\ell) \tag{2.1}$$

by variation of $\theta = \{w, \mu, \Sigma\}$. Where $x_i \in \mathbb{R}^d$, $\forall i \in \{1, 2, ..., n\}$ where d is the dimension of the feature space, n the number of data points and $N(x; \mu_\ell, \Sigma_\ell)$ is the probability of the data point x for the Gaussian component ℓ (then q is called a gaussian mixture since Eq.2.1):

$$N(\boldsymbol{x}; \boldsymbol{\mu}, \boldsymbol{\Sigma}) = \frac{1}{(2\pi)^{d/2} \det(\boldsymbol{\Sigma})^{1/2}} \exp\left(-\frac{1}{2}(\boldsymbol{x} - \boldsymbol{\mu})^\top \boldsymbol{\Sigma}^{-1}(\boldsymbol{x} - \boldsymbol{\mu})\right) \tag{2.2}$$

As an Expectation Maximization model, we must estimate the maximum likelihood estimator (MLE) for the function q:

$$l(\theta) = \prod_{i=0}^{n} q(x_i; \theta) \tag{2.3}$$

Subject to:

$$\sum_{\ell=0}^{m} w_\ell = 1$$

$$w_\ell \geq 0, \forall \ell \in \mathbb{N}, 0 \leq \ell \leq m$$

The constraints over w_ℓ are because $q(x, \theta)$ needs to be a probability distribution. This makes it impossible to maximize this function by the first-order condition of Karush-Kuhn-Tucker (KKT). But we can change this by making the next parameterization:

$$w_\ell = \frac{\exp(\gamma_\ell)}{\sum_{\ell=0}^{m} \exp(\gamma_\ell)}$$

.

Note that no matter what is the value of γ_ℓ, w_ℓ is non negative and $\sum_{\ell=0}^{m} w_\ell = 1$. So the new optimization problem is non-restricted and the log-likelihood can be written as:

$$L(\theta) = \log(l(\theta)) = \sum_{i=0}^{n} \log \sum_{\ell=0}^{m} \exp(\gamma_\ell) N(x_i; \mu_\ell, \Sigma_\ell) - n \log \sum_{\ell=0}^{m} \exp(\gamma_\ell) \tag{2.4}$$

(At this point we can use the ascent gradient algorithm to maximize this function but has the problem of defining the step parameter. To more details consult [27])

Now we can set the partial derivates to zero to find the optimal solution. From doing this procedure we obtain the following expressions:

$$\begin{cases} \widehat{w}_\ell = \frac{1}{n} \sum_{i=1}^{n} \widehat{\eta}_{i,\ell}, \\ \widehat{\boldsymbol{\mu}}_\ell = \frac{\sum_{i=1}^{n} \widehat{\eta}_{i,\ell} \boldsymbol{x}_i}{\sum_{i'=1}^{n} \widehat{\eta}_{i',\ell}}, \\ \widehat{\boldsymbol{\Sigma}}_\ell = \frac{\sum_{i=1}^{n} \widehat{\eta}_{i,\ell} (\boldsymbol{x}_i - \widehat{\boldsymbol{\mu}}_\ell)(\boldsymbol{x}_i - \widehat{\boldsymbol{\mu}}_\ell)^{\mathrm{T}}}{\sum_{i'=1}^{n} \widehat{\eta}_{i',\ell}}, \end{cases} \tag{2.5}$$

Where $\eta_{i,\ell}$ is called the responsability of the ℓ component at the x_i sample [27].

$$\widehat{\eta}_{i,\ell} = \frac{\widehat{w}_\ell N\left(\boldsymbol{x}_i; \widehat{\boldsymbol{\mu}}_\ell, \widehat{\boldsymbol{\Sigma}}_\ell\right)}{\sum_{\ell'=1}^{m} \widehat{w}_{\ell'} N\left(\boldsymbol{x}_i; \widehat{\boldsymbol{\mu}}_{\ell'}, \widehat{\boldsymbol{\Sigma}}_{\ell'}\right)} = \mathbb{E}(z_{i,\ell}). \tag{2.6}$$

The random variable $z_{i,\ell}$ represents the membership event of x_i to the $z_{i,\ell}$ component or cluster.

However, the new problem can not be solved analytically. But we can use the EM algorithm to maximize the log-likelihood iteratively [10]. First, let's see that:

$$L(\theta) = \sum_{i=0}^{n} \log(\sum_{l=0}^{m} w_l N(x_i; \mu_l, \Sigma_l)) = \sum_{i=0}^{n} \log(\sum_{l=0}^{m} \widehat{\eta_{i,\ell}} \frac{w_l N(x_i; \mu_l, \Sigma_l)}{\hat{\eta_{i,\ell}}}) \tag{2.7}$$

as $\sum_{l=0}^{m} \widehat{\eta_{i,\ell}} = 1$, with each $\eta_{i,\ell}$ nonnegative and the logarithm function is convex, we can apply Jensen's inequality [27],

$$L(\theta) = \sum_{i=0}^{n} \log(\sum_{l=0}^{m} \widehat{\eta_{i,\ell}} \frac{w_l N(x_i; \mu_\ell, \Sigma_\ell)}{\widehat{\eta_{i,\ell}}}) \geq \sum_{i=0}^{n} \sum_{l=0}^{m} \widehat{\eta_{i,\ell}} \log(\frac{w_l N(x_i; \mu_l, \Sigma_l)}{\widehat{\eta_{i,\ell}}}) = b(\theta). \tag{2.8}$$

So $b(\theta)$ is a lower bound of $L(\theta)$, furthermore, we can show that $b(\widehat{\theta}) = L(\widehat{\theta})$:

$$b(\widehat{\theta}) = \sum_{i=0}^{n} \sum_{l=0}^{m} \widehat{\eta_{i,\ell}} \log(\frac{\widehat{w_\ell} N(x_i : \widehat{\mu}_\ell, \widehat{\Sigma_\ell})}{\widehat{\eta_{i,\ell}}}) \tag{2.9}$$

substituting $\widehat{\eta_{i,\ell}}$ into the logarithm:

$$b(\widehat{\theta}) = \sum_{i=0}^{n} (\sum_{l=0}^{m} \widehat{\eta_{i,\ell}}) \log(\frac{\widehat{w_\ell} N(x_i; \widehat{\mu}_\ell, \widehat{\Sigma_\ell})}{\frac{\widehat{w}_\ell N\left(\boldsymbol{x}_i; \widehat{\boldsymbol{\mu}}_\ell, \widehat{\boldsymbol{\Sigma}}_\ell\right)}{\sum_{\ell'=1}^{m} \widehat{w}_{\ell'} N\left(\boldsymbol{x}_i; \widehat{\boldsymbol{\mu}}_{\ell'}, \widehat{\boldsymbol{\Sigma}}_{\ell'}\right)}}) \tag{2.10}$$

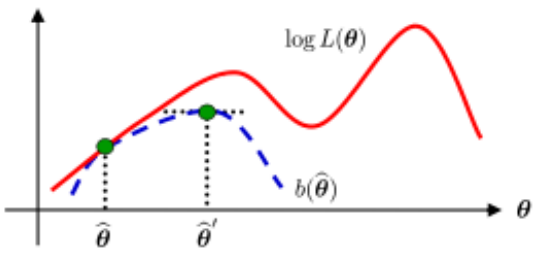
and simplifying the expression within the logarithm and noting that $\sum_{l=0}^{m} \widehat{\eta_{i,\ell}} = 1$:

$$b(\widehat{\theta}) = \sum_{i=0}^{n} \log(\sum_{\ell'=1}^{m} \widehat{w}_{\ell'} N\left(\boldsymbol{x}_i; \widehat{\boldsymbol{\mu}}_{\ell'}, \widehat{\boldsymbol{\Sigma}}_{\ell'}\right)) = L(\widehat{\theta}). \tag{2.11}$$

We have a lower bound $b(\theta)$ of $L(\theta)$ such that $b(\widehat{\theta}) = L(\widehat{\theta})$. Applying the first condition of KKT in $b(\theta)$ we obtain the next expressions:

$$\begin{cases} \widehat{w}'_\ell = \frac{1}{n} \sum_{i=1}^{n} \widehat{\eta}_{i,\ell}, \\ \widehat{\boldsymbol{\mu}}'_\ell = \frac{\sum_{i=1}^{n} \widehat{\eta}_{i,\ell} \boldsymbol{x}_i}{\sum_{i'=1}^{n} \widehat{\eta}_{i',\ell}}, \\ \widehat{\boldsymbol{\Sigma}}'_\ell = \frac{\sum_{i=1}^{n} \widehat{\eta}_{i,\ell} (\boldsymbol{x}_i - \widehat{\boldsymbol{\mu}}_\ell)(\boldsymbol{x}_i - \widehat{\boldsymbol{\mu}}_\ell)^\top}{\sum_{i'=1}^{n} \widehat{\eta}_{i',\ell}}. \end{cases} \tag{2.12}$$

The EM algorithms are composed of two principal steps: E-step and M-step. First, you need to initialize the parameters. To do that there are several approaches, but the most popular is to use the parameters obtained by using K-means. We will settle for this.
So in the iteration $t = 0$, we have $\theta_0 = \left\{\widehat{w_\ell}, \widehat{\mu}_\ell, \widehat{\Sigma_\ell}\right\}_{t=0}$



(a)

Figure 2.1: EM-algorithm convergence [27].

Using Eq. 2.6 we obtain the responsability $\widehat{\eta_{i,\ell}}$. When we do this, we are maximizing the expectation $\mathbb{E}(z_{i,\ell})$ hence the name of E-step [27].

Having $\widehat{\eta_{i,\ell}}$, we can use the Eq. 2.12 for optimize the lower bound $b(\theta)$ (M-step). We repeat this cycle until θ converges to a local optimum solution. In Eq. 1 the EM algorithm for the Gaussian mixture is written:

Algorithm 1 EM-algorithm

Require: $\theta_0 = \left\{\widehat{w_\ell}, \widehat{\mu_\ell}, \widehat{\Sigma_\ell}\right\}_{t=0}$

Ensure: $\theta_T = \left\{\widehat{w_\ell}, \widehat{\mu_\ell}, \widehat{\Sigma_\ell}\right\}_{t=T}$

1: **for** $t \leftarrow 1, n$ **do**

2: **E-step:** compute $\eta_{i,\ell}$:

$$\widehat{\eta}_{i,\ell} = \frac{\widehat{w}_\ell N\left(\boldsymbol{x}_i; \widehat{\boldsymbol{\mu}}_\ell, \widehat{\boldsymbol{\Sigma}}_\ell\right)}{\sum_{\ell'=1}^{m} \widehat{w}_{\ell'} N\left(\boldsymbol{x}_i; \widehat{\boldsymbol{\mu}}_{\ell'}, \widehat{\boldsymbol{\Sigma}}_{\ell'}\right)} = \mathbb{E}(z_{i,\ell}).$$

3: **M-step:** actualize $\theta_{t+1} = \{w'_\ell, \mu'_\ell, \Sigma'_\ell\}$ by maximizing $b(\theta)$:

$$\begin{cases} \widehat{w}'_\ell = \frac{1}{n} \sum_{i=1}^{n} \widehat{\eta}_{i,\ell}, \\ \widehat{\boldsymbol{\mu}}'_\ell = \frac{\sum_{i=1}^{n} \widehat{\eta}_{i,\ell} \boldsymbol{x}_i}{\sum_{i'=1}^{n} \widehat{\eta}_{i',\ell}}, \\ \widehat{\boldsymbol{\Sigma}}'_\ell = \frac{\sum_{i=1}^{n} \widehat{\eta}_{i,\ell} (\boldsymbol{x}_i - \widehat{\boldsymbol{\mu}}_\ell)(\boldsymbol{x}_i - \widehat{\boldsymbol{\mu}}_\ell)^\top}{\sum_{i'=1}^{n} \widehat{\eta}_{i',\ell}}. \end{cases}$$

4: $t = t + 1$

5: **end for**

But, what ensures that this algorithm converges to a local optimum solution?

If $L(\theta)$ is bounded, $b(\theta)$ is bounded since $L(\theta) \geq b(\theta)$. So if we prove that the log-likelihood is monotone nondecreasing by iterating E-step and M-step, we can use the monotone convergence theorem, proving that EM-algorithm converges to a local optimum solution.

Let's probe that the log-likelihood is nondecreasing by iterating E-step and M-step:

suppose we are in the t iteration: We compute $\widehat{\eta_{i,\ell}}$ with $\theta_t = (\{w_\ell, \mu_\ell, \Sigma_\ell\}_{\ell=0}^{m})_t$ (this is the E-step). we can write $b(\theta_t)$. Maximizing $b(\theta_t)$ (i.e. applying the M-step), we obtain a new θ_{t+1}.

But $L(\theta_{t+1}) \geq b(\theta_{t+1})$ since $b(\theta)$ is a lower bound of $L(\theta)$. On the other hand, $b(\theta_{t+1}) \geq b(\theta_t)$ because θ_{t+1} is the maximum of $b(\theta)$ in this iteration. Finally, $b(\theta_t) = L(\theta_t)$ because Eq. 2.11.

As a consequence, log-likelihood is monotone nondecreasing by iterating E-step and M-step. Following that, we are going, at least, to a local maximum of L.

2.2 Reinforcement Learning and Markov Games

Reinforcement Learning (RL) is a fascinating branch of machine learning where one or more agents learn to make decisions through interaction with an environment, aiming to maximize cumulative reward over time. Unlike traditional methods where actions are explicitly instructed, RL operates on a principle of trial and error. The agent explores various actions, receives feedback (rewards) from the environment, and gradually learns to prioritize actions that lead to higher rewards. This iterative process allows RL to adapt and refine its decision-making strategy over time. RL finds applications across diverse domains, from gaming and robotics to finance and healthcare, showcasing its versatility in handling complex scenarios where explicit instruction or abundant labeled data might be lacking.

Markov Games are an extension of Markov Decision Processes (MDPs) to scenarios involving multiple interacting agents. In a traditional MDP, there is only one agent making decisions based on states and rewards. However, in Markov Games, multiple agents make decisions simultaneously, each with their own policies and objectives. Reinforcement Learning (RL) and Markov Games are closely related, as Markov Games, also known as stochastic games, provide a theoretical framework for reinforcement learning in multi-agent environments.

For the purposes of this thesis, a particular case of Markov Games is required where agents are unable to directly observe the true state of the environment but only a partial view of it. Since agents will observe the parameters obtained via GMM from the data, namely their mean, covariance, and weight, they are only seeing a partial view of the system's state. This specific case of Markov Games is referred to as Partially Observable Markov Games. To avoid redundancy, we will define only this particular family of Markov Games, noting that the difference between these two frameworks lies in that in a Markov Game, observations and states are equal, whereas in Partially Observable Markov Games, observations are a function dependent on the state but are not necessarily equal.

Multi-Agent Reinforcement Learning (MARL) is an extension of traditional Reinforcement Learning (RL) where multiple agents learn to make decisions while interacting within a shared environment. In MARL, each agent seeks to optimize its own objective while considering the actions and behaviors of other agents. This introduces additional complexity compared to single-agent RL, as agents must learn not only how to achieve their individual goals but also how to coordinate and interact with other agents effectively.

MARL has applications in various domains such as autonomous driving, multi-robot systems, network routing, and cooperative game playing. It involves challenges like coordination, communication, and competition among agents, making it an active area of research in both academia and industry. Techniques in MARL often focus on developing algorithms that

enable agents to learn policies that lead to coordinated behavior, negotiation strategies, and emergent collaboration.

2.3 Partially-Observable Markov Games

A Partially-Observable Markov game (POMG) of N agents is a stochastic game in which the future state is determined solely by the current state, the action taken by the players, and the observations made by the players, but each agent does not have complete information about the state of the system and the action of the other agents. Formally, a Semi-Observable Markov game is defined as a tuple $\mathcal{G} = (\mathcal{S}, \mathcal{A}, \mathcal{O}, \mathcal{T}, R, o)$ [18]

This type of game is defined by a set of states $\mathcal{S}$ that describe the various possible configurations of the agents, a set of actions $\mathcal{A} = \mathcal{A}_1, \ldots, \mathcal{A}_N$ that each agent can take, and a set of observations $\mathcal{O} = \mathcal{O}_1, \ldots, \mathcal{O}_N$ that the agents can make. To choose actions, each agent i uses a stochastic policy $\pi_{\theta_i} : \mathcal{O}_i \times \mathcal{A}_i \to [0, 1]$, which determines the next state according to the state transition function $\mathcal{T} : \mathcal{S} \times \mathcal{A}_1 \times \ldots \times \mathcal{A}_N \to \mathcal{S}$. Furthermore, each agent i obtains rewards as a function of the state and action $r_i : S \times A_i \to R$, and receives a private observation correlated with the state $o_i : S \to \mathcal{O}_i$. The initial states are determined by a distribution $\rho : S \to [0, 1]$, and the total discounted reward over time $R_t = \sum_{i=t}^{T-1} \gamma^{k-t} r_i$. Where the discount factor $\gamma \in [0, 1]$, determines the importance of future rewards relative to immediate rewards [19].

In a POMG, each agent has its own observation function and observation space, and its observations may be different from those of the other agents. The agents also have different reward functions, which may be competing or cooperative. In this work we will consider the same reward function for all agents, so the nature of the resulting game is cooperative.

The goal of a POMG is to find a joint policy for the agents that maximize their expected total discounted reward over time. This is a challenging problem because the agents must take into account not only their own observations and rewards but also those of the other agents.

Is important to remark that we will consider a denumerable (i.e. infinite non-numerable) observation space, and denumerable, but compact action or controller space as in [7]

One way to tackle a POMG is by employing Multi-agent Deep Deterministic Policy Gradient. Given that it is one of the main methods of Multi-Agent Reinforcement Learning (at least the most cited), it is the chosen method for our proposal.

2.4 Multi Agent Deep Deterministic Policy Gradient

In the introduction, we mentioned that MADDPG is an extension of the multi-agent environment of DDPG. A good induction to the MADDPG formalism must have an equally good introduction to DDPG. In turn, deep deterministic policy gradient is an approach to

solving a policy gradient problem.

2.4.1 Policy Gradient

The policy gradient is a fundamental technique in reinforcement learning that focuses on learning the optimal policy directly to maximize the accumulated reward in a given environment. Instead of computing the values of all actions in each state, as done by other value-based approaches, the policy gradient focuses on learning the probability distribution of actions in each state. This is achieved by computing the gradient of the objective function, which is the expectation of the accumulated reward, with respect to the policy parameters. Subsequently, this gradient is used to update the policy parameters in the direction that increases the expectation of the reward. This allows agents to learn stochastic policies and make probabilistic decisions in complex or uncertain environments, which can be beneficial for enhancing performance in various machine learning and artificial intelligence applications.

Given a POGM defined by $\mathcal{G}$, we can define $Q^{\pi_\theta} = \mathbb{E}_{a\sim\pi}\left[R_t \mid S_t = s, A_t = a\right]$ (for simplicity will call it Q^π in the future) function as follow [31]:

$$J(\theta) = \int_{s\in\mathcal{S}} \rho^\pi(s) \int_{a\in\mathcal{A}} \pi_\theta(a \mid s) Q^\pi(s,a) \tag{2.13}$$

Where $\rho^\pi(s) = \int_{\mathcal{S}} \sum_{t=1}^{\infty} \gamma^{t-1} p_1(s) p\left(s \to s', t, \pi\right) \mathrm{d}s$ is the discounted distribution of the state $s \in \mathcal{S}$ given π_θ, $p\left(s \to s', t, \pi\right)$ is the probability of visit s' at the time t given the policy π . Again we are omitting the parameter θ in the notation. The good news is that this approach allows working in continuous action and state spaces, but has the issue of being tricky to solve with the gradient ascent. That is why depends on π_θ that is selecting the actions. As we don't know the environment, is very difficult to estimate the effects of the policy updates. To sort this problem, exist the gradient policy theorem:

Theorem 2.1 *Given Eq. 2.13 we can obtain a good approximation of its gradient without deriving $\rho^\pi(s)$, and its expression is:*

$$\nabla_\theta J(\theta) = \nabla_\theta \int_{s\in\mathcal{S}} \rho^\pi(s) \int_{a\in\mathcal{A}} Q^\pi(s,a)\pi_\theta(a \mid s) \propto \int_{s\in\mathcal{S}} \rho^\pi(s) \int_{a\in\mathcal{A}} Q^\pi(s,a)\nabla_\theta \pi_\theta(a \mid s) \tag{2.14}$$

with this reformulation, the computational complexity increases a lot and makes possible the use of gradient ascent. There are several methods that use this like DDPG, REINFORCE, A2C, A3C, and Off-policy gradient among others.

2.4.2 Deterministic Policy Gradient

We will focus on deterministic policy gradient (DPG) because is the nearest ancestor of DDPG. In the above definition problem of PG, we consider the policy $\pi_\theta(a \mid s)$ as a distribution over $\mathcal{A}$, but as its name suggests, in DPG we will consider $a = \mu(s)$ as a deterministic policy and $\rho^\mu(s) = \int_\mathcal{S} \sum_{t=1}^{\infty} \gamma^{t-1} p_1(s) p\left(s \to s', t, \mu\right) \mathrm{d}s$ is the discounted distribution.
It is common to use a greedy strategy (i.e. take $a^{k+1} = argmax_a Q^\mu(s,a)$) in environments with discrete action and state spaces, but is not good for continuous cases because it is required a global optimization for each step. In place, is better to use the local gradient to improve the parameter θ as follows:

$$\theta^{k+1} = \theta^k + \alpha \mathbb{E}_{s\sim\rho\mu^k} \left[\nabla_\theta a \mu_\theta(s) \nabla_a Q^{\mu^k}(s,a) \Big|_{a=\mu_\theta(s)} \right] \tag{2.15}$$

As in the stochastic case, ρ^μ will change as the policy determined by θ changes. So is not trivial to ensure that θ is really improving. To solve this issue, we need a deterministic equivalent for the policy gradient theorem, in other words, a way to compute the policy gradient without deriving ρ^μ. fortunately, exist the Deterministic Policy Gradient Theorem. First, we define the performance function $J(\mu_\theta)$ as in Eq. 2.13.

Given a deterministic policy $\mu_\theta : \mathcal{S} \to \mathcal{A}$, we define:

$$J\left(\mu_\theta\right) = \int_\mathcal{S} \rho^\mu(s) r\left(s, \mu_\theta(s)\right) \mathrm{d}s = \mathbb{E}_{s\sim\mu}\left[r\left(s, \mu_\theta(s)\right)\right] \tag{2.16}$$

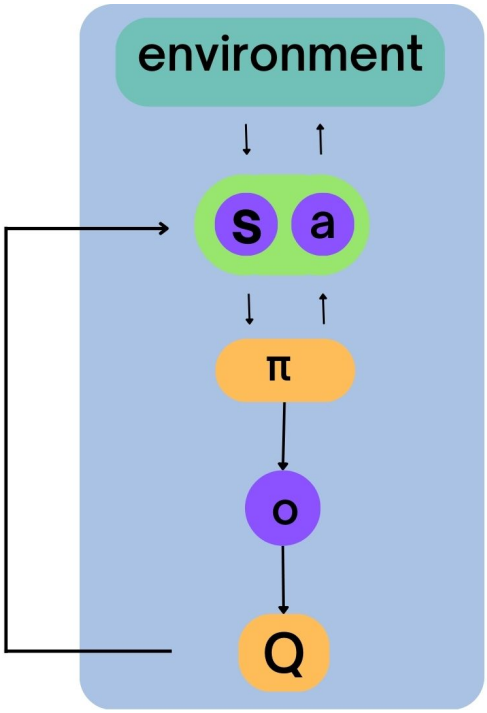
we can note that in difference with Eq. 2.13, we are not integrating into the action space. This is because in Eq. 2.13 the policy is a distribution over $\mathcal{A}$ and in this case is a deterministic function.

Theorem 2.2 *(Deterministic Policy Gradient Theorem). Suppose that the POMG satisfies that $\nabla_\theta \mu_\theta(s)$ and $\nabla_a Q^\mu(s,a)$ exist and that the deterministic policy gradient exists. Then,*

$$\nabla_\theta J\left(\mu_\theta\right) = \int_\mathcal{S} \rho^\mu(s) \nabla_\theta \mu_\theta(s) \nabla_a Q^\mu(s,a) \Big|_{a=\mu_\theta(s)} \mathrm{d}s = \mathbb{E}_{s\sim\rho^\mu}\left[\nabla_\theta \mu_\theta(s) \nabla_a Q^\mu(s,a)|_{a=\mu_\theta(s)}\right] \tag{2.17}$$

2.4.3 Deep Deterministic Policy Gradient

DDPG is an actor-critic, model-free, off-policy algorithm based on the deterministic policy gradient [17]. Model-Free, in reinforcement learning, a model represents the agent's understanding of the environment. Model-free algorithms, like DDPG, skip building this model, learning directly from interactions with the environment. They focus on improving their policy and value function without explicitly modeling the environment's dynamics. Off-policy means that



(a)

Figure 2.2: Actor Critic Diagram

learning allows agents to learn from experiences generated by a different policy than the one currently being improved. In DDPG, the agent leverages past experiences stored in a replay buffer, even if they were generated by an older version of its policy. This approach enhances stability and sample efficiency by reusing experiences across different iterations of learning. An actor-critic model is a particular kind of reinforcement learning model that combines an actor and a critic. The actor is in charge of choosing actions that will maximize the value estimated by the critic, and the critic is in charge of estimating the value of an action taken by the actor in a particular state. DDPG is model-free, so the critic has no model of the environment to schedule its future actions, is just learning policies from experience data. DDPG is off-policy because it counts with a replay buffer, that is, a set of data composed of the prior observation, the action, the reward in the step, and the posterior observation. With this, the actor and critic can learn from past realizations. Finally, updating directly the weight of the network conduct to be prone to divergence in many environments [17]. To treat this, DDPG uses soft updates, that consist in update target functions as a weighted average between the main network parameters and the target network parameters.

Actor Critic Models are Deep Reinforcement Learning methods where at least two different neural networks learn together with the objective of giving the best policies possible. In an actor-critic framework there are always:

- **Actor:** learn the parameters θ to maximize $J(\mu_\theta)$.
- **Critic:** learn the parameters w to estimate the best Q^μ possible.

This means we are using different networks for computing the values of Q^w and the

policies π_θ but each network depends on the other one because Q^μ appears in $\nabla_\theta J(\mu_\theta)$ and Q^w depends on the policies. In other words, the actor is learning policies while exploring the environment based on the rewards associated with the trajectories and the Q-values obtained by the critic. Is important to note that not every Q^w will conduce to a true ascent gradient. In [26] define a family of compatibles Q^w:
Definition: A function approximator $Q^w(s,a)$ is compatible with a deterministic policy $\mu_\theta(s)$ if $\nabla_\theta J_\beta(\theta) = \mathbb{E}\left[\nabla_\theta \mu_\theta(s) \nabla_a Q^w(s,a)|_{a=\mu_\theta(s)}\right]$

Theorem 2.3 *Given $Q^w(s,a)$ and $\mu_\theta(s)$ if the following two conditions are fulfilled:*

1. $\nabla_a Q^w(s,a)|_{a=\mu_\theta(s)} = \nabla_\theta \mu_\theta(s)^\top w$
2. *w minimises the mean-squared error, $MSE(\theta, w) = \mathbb{E}\left[\varepsilon(s;\theta,w)^\top \varepsilon(s;\theta,w)\right]$ where $\varepsilon(s;\theta,w) = \nabla_a Q^w(s,a)|_{a=\mu_\theta(s)} - \nabla_a Q^\mu(s,a)|_{a=\mu_\theta(s)}$*

then $Q^w(s,a)$ is compatible with $\mu_\theta(s)$.

The proof of this theorem is made in [26], but is more interesting in its consequences. The first is that for every $\mu_\theta(s)$ exist a $Q^w(s,a)$ because we can construct it with the form: $(a - \mu_\theta(s))^\top \nabla_\theta \mu_\theta(s)^\top w + V^v(s)$, where $V^v(s)$ can be whatever function differentiable but independent of the action a, in this way deriving $Q^w(s,a)$ respect to a we obtain $\nabla_\theta \mu_\theta(s)^\top w$. To give an interpretation to this form of $Q^w(s,a)$ we can see it as the sum of two elements: the "advantage" $A^w(s,a) = \phi(s,a)^\top w = (a - \mu_\theta(s))^\top \nabla_\theta \mu_\theta(s)^\top w$ the advantage is depicted to the reader as a line regression with parameter w and features $\phi(s,a)$ and the function $V^v(s)$ that estimates the values of the state s. Condition 2 is more difficult to comply with since is hard to have unbiased samples of the real Q^μ. Below we will discuss how DDPG deals with this problem.

Experience Replay Used for the first time in [20], replay buffers are used to store transitions from an agent's interactions with the environment during experience replay. The current environment's state, the agent's action, the reward received as a result, and the following environment's state make up each transition. The network uses batches of transitions sampled uniformly at random from the replay buffer during training to alter its parameters. This has advantages like:

- **Reducing the correlation between updates:** the methods are less likely to get stuck in feedback loops where it repeatedly updates the same set of parameters by randomly sampling transitions from the replay buffer.
- **Enhancing sample effectiveness:** The networks can learn more from each transition when it is used more than once, which lowers the amount of data required to train the network.
- **Stability is improved** because the methods that use experience replay are less sensitive to tiny changes in the input distribution because it updates the network using a fixed set of transitions.

Soft Updating One of the most important mechanisms introduced in [17] was soft updating. The critic network, which is used to determine the target Q-value during training, is copied into the target network. In conventional Q-learning algorithms, a replica of the most recent critic network is periodically added to the target network. The abrupt changes in the target values, however, can cause instability. [26] suggested using a soft update rule, in which the target network is updated gradually toward the present critic network, to address this problem. By taking a weighted average of the target network's and the critic network's current parameters, the soft update is carried out. The $\tau << 1$ hyperparameter, which regulates the speed of the soft update, is the weight used for the critic network's current parameters.

$$\theta^{Q'} = \tau \cdot \theta^{Q'} + (1 + \tau) \cdot \theta^{Q} \tag{2.18}$$

$$\theta^{\mu'} = \tau \cdot \theta^{\mu'} + (1 + \tau) \cdot \theta^{\mu} \tag{2.19}$$

In comparison to conventional Q-learning algorithms such as DQN [20], the use of soft updates in DDPG has been shown to improve stability and convergence. Soft updating has gained popularity since it was first used in [16] paper in 2016 and is now a common technique in deep RL algorithms.

Exploration Exploitation Trade-off

One of the most important challenges for continuous reinforcement learning is the "exploration-exploitation" trade-off. In general, all reinforcement learning methods need to deal with this. The reason is that to learn new success policies is necessary to explore new trajectories that may have low rewards associated. In consequence, the methods tend to not explore a sufficient amount of trajectories conducted to local maximums. On the other hand, if the model tries to explore too much without exploiting the good policies, the model becomes prone to diverge. Fortunately, off-policy models can treat the exploration separately from the learning algorithm. A way to do this is to add a random process to the action taken by the agents to facilitate the exploration:

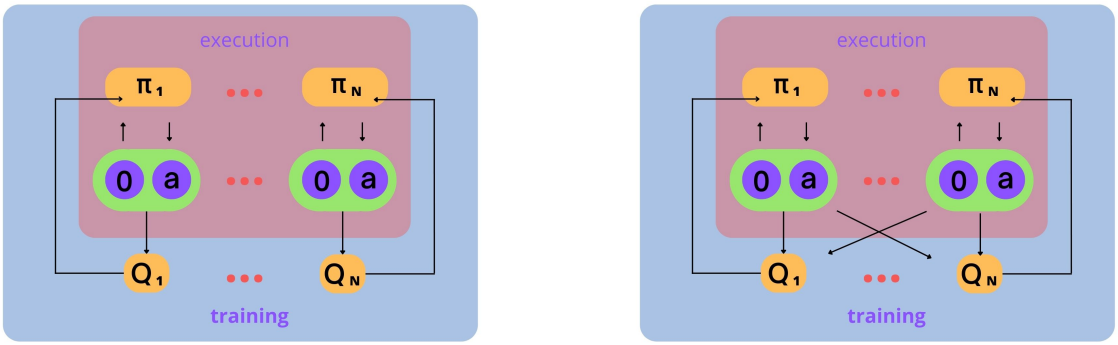
$$\mu'(s) = \mu(s \mid \theta^{\mu}) + \mathcal{N} \tag{2.20}$$

Where the random process $\mathcal{N}$ can be adapted to each environment.

DDPG Algorithm We have now explained all components of the DDPG algorithm. Now all that remains is to explain how they work together.

To start, the critic θ^{Q} and actor θ^{μ} networks weights are initialized randomly and copied to obtain their respective target networks $\theta^{Q'}, \theta^{\mu'}$. At this point, we need to initialize the buffer replay which is a cache memory that we will use to implement the experience replay mechanism.

For each episode, initialize a new random process $\mathcal{N}$ customized for the environment. In each step t of the episode, the actor will receive an observation from the environment and will return an action $\mu(s_t \mid \theta^{\mu})$. However, we need to add the noise produced by the random process $\mathcal{N}$ to deal with the exploration-exploitation trade-off, so the action selected is $a_t = \mu(s_t \mid \theta^{\mu}) + \mathcal{N}_t$. We store the transition in the buffer replay $\mathcal{D}$ initialized before and immediately a minibatch of N elements from $\mathcal{D}$ is sampled to set $y_i = r_i + \gamma Q'(s_{i+1}, \mu'(s_{i+1} \mid$



(a) (b)

Figure 2.3: (a) Decentralized training with a decentralized implementation (b) Centralized training with a decentralized implementation

$\theta^{\mu'}) \mid \theta^{Q'})$ for each $i \in \{1, ..., N\}$ (Q' is the current target Q-function). Having the y_i values, we can update the critic with the loss function $L = \frac{1}{N}\sum_i (y_i - Q(s_i, a_i \mid \theta^Q))^2$. Now we have a re-parameterized Q-function, that we are going to use for updating the actor-network weights with the $\nabla_\theta J(\mu_\theta)$ expression in Eq. 2.17. Finally, the critic and actor target networks are updated with soft updating following Eq. 2.18 and Eq. 2.19 respectively.

2.4.4 Multi Agent Deep Deterministic Policy Gradient

In the middle of the past decade, DDPG and other RL methods were applied to a wide kind of problem with one agent. However, when the researcher tried to apply these methods to multi-agent environments that need coordination, they encountered poor results. One reason is that decentralized methods change their policies while training, this makes the observations of some agents become non-stationary, in other words, these agents can not explain their observations by their policies. Specifically, policy gradient methods showed to have a high variance in this kind of environment. MADDPG (Multi-Agent Deep Deterministic Policy Gradient) [19] is a variation of the DDPG (Deep Deterministic Policy Gradient) algorithm created specifically for multi-agent environments. The main innovation of MADDPG is a centralized critic network that considers the actions of all agents in the environment, rather than just the actions of one agent. This enables the agents to coordinate their actions and obtain greater rewards than they could do on their own. This corresponds to centralized training with a decentralized implementation framework.

Consider a POMG as stated in subsection 2.1. We can write $\nabla_\theta J(\theta_i) = \mathbb{E}(R_i)$ with a centralized action value function $Q_i^\pi : \mathbb{R}^{N \times m + \dim_{\mathbb{R}}(x)} \to \mathbb{R}$ (m is the dimension of the action of each agent[1]) as follow:

$$\nabla_{\theta_i} J(\theta_i) = \mathbb{E}_{s \sim p^\pi, a_i \sim \boldsymbol{\pi}_i} \left[\nabla_{\theta_i} \log \boldsymbol{\pi}_i (a_i \mid o_i) Q_i^\pi (\mathbf{x}, a_1, \ldots, a_N) \right] \tag{2.21}$$

[1]the dimension of each agent can be different, but is not the case of our model. For simplicity, we will consider that all agents have the same action space

Is important to remark that $Q_i^{\pi}(\mathbf{x}, a_1, \ldots, a_N)$ takes the actions of all agents and the vector $\mathbf{x}$ is the observation of all agents, in addition, can or not contain additional information about the environment, note that we have change the notation of states s by the vector $\mathbf{x}$ because we are not observing the real states of the environment, but a partially observation. We are writing this considering stochastic policies. To extend this to deterministic policies we will use Montecarlo to approximate the distributions ρ^{μ}, π_i with the replay buffer $\mathcal{D}$ that is similar to the buffer experience in DDPG but composed of tuples with form $(\mathbf{x}, \mathbf{x}', a_1, \ldots, a_N, r_1, \ldots, r_N)$, where the vector $\mathbf{x}'$ is the resulting vector of taking the actions $(a_1, \ldots, a_N)$ observing $\mathbf{x}'$. Let $\{\mu_{\theta_i}\}_{i=0}^{N}$ be the deterministic policies of each agent i we can now write:

$$\nabla_{\theta_i} J(\boldsymbol{\mu}_i) = \mathbb{E}_{\mathbf{x}, a \sim \mathcal{D}} \left[\nabla_{\theta_i} \boldsymbol{\mu}_i (a_i \mid o_i) \nabla_{a_i} Q_i^{\boldsymbol{\mu}} (\mathbf{x}, a_1, \ldots, a_N) \big|_{a_i = \boldsymbol{\mu}_i(o_i)} \right] \tag{2.22}$$

the actualization of the critic network is analogous to the one carried out in DDPG, just that the Q-function is the new formulated here:

$$y = r_i + \gamma Q_i^{\boldsymbol{\mu}'} (\mathbf{x}', a_1', \ldots, a_N') \Big|_{a_j' = \boldsymbol{\mu}_j'(o_j)}, \tag{2.23}$$

where $\boldsymbol{\mu}' = \left\{ \boldsymbol{\mu}_{\theta_1'}, \ldots, \boldsymbol{\mu}_{\theta_N'} \right\}$ are the target policies. As you can see, we are assuming that each agent knows the policies of all the others. For the purposes of this thesis we can meet this condition. However, it is possible to relax this condition by approximating the policies of the other agents. This mechanism is called policy inference [19]. However, it is beyond the scope of this thesis and will not be discussed here.

Now the loss function used to actualize the critic is:

$$\mathcal{L}(\theta_i) = \mathbb{E}_{\mathbf{x}, a, r, \mathbf{x}'} \left[\left(Q_i^{\boldsymbol{\mu}} (\mathbf{x}, a_1, \ldots, a_N) - y \right)^2 \right], \tag{2.24}$$

As we said above, the motivation for using centralized training with a decentralized implementation is to avoid the agent's non-stationary observations. In fact, we are avoiding this because in Eq. 2.22 the expectation is conditioned by the action of the other agents, making irrelevant the changing policies, because $P(s' \mid s, a_1, \ldots, a_N, \boldsymbol{\pi}_1, \ldots, \boldsymbol{\pi}_N) = P(s' \mid s, a_1, \ldots, a_N) = P(s' \mid s, a_1, \ldots, a_N, \boldsymbol{\pi}_1', \ldots, \boldsymbol{\pi}_N')$ no matter if $\pi_i \neq \pi_i'$ for any $i \in \{1, ..., N\}$.

2.5 Dynamic Clustering State-of-the-Art

In the realm of dynamic clustering, an evolving field that focuses on adapting clustering algorithms to handle changing data distributions over time, there is a continuous pursuit of addressing the challenges posed by data streams, concept drift, and evolving environments. Traditional clustering techniques assume static data, but in real-world scenarios, data streams, concept drift, and evolving environments pose challenges to maintaining accurate and up-to-date cluster representations [13, 6, 9, 22, 2, 3].

One key aspect of dynamic clustering is concept drift adaptation. Concept drift refers to the phenomenon where the statistical properties of the data change over time, leading to shifts in the underlying clustering structures. Researchers have developed methods to detect and track concept drift, allowing clustering models to adapt accordingly [13].

Another important direction in dynamic clustering is the development of incremental clustering algorithms. Instead of recomputing clusters from scratch as new data arrives, incremental algorithms update existing clusters, add new clusters, or remove obsolete clusters. These algorithms reduce computational complexity and memory requirements while maintaining the clustering accuracy [6, 9].

Additionally, there is growing interest in online clustering, which deals with clustering data streams in real-time. Online clustering algorithms process data instances one at a time and continuously update the cluster representations. These algorithms often employ approximation techniques and data summarization to handle the high-speed and large-volume nature of data streams efficiently [22].

Furthermore, researchers have focused on hybrid approaches that combine multiple clustering algorithms to leverage their complementary strengths. For instance, integrating density-based clustering with centroid-based clustering techniques can enhance the ability to detect clusters of varying shapes and densities in dynamic environments [2].

Moreover, reinforcement learning has emerged as a promising technique in dynamic clustering. By formulating clustering as a reinforcement learning problem, algorithms can learn to adapt and optimize the clustering process based on feedback and rewards. This approach enables the discovery of cluster structures while simultaneously addressing the challenge of initialization [3].

Finally, researchers have explored the use of fuzzy clustering in dynamic scenarios. Fuzzy clustering allows data points to belong to multiple clusters with different degrees of membership, providing flexibility to handle changing data distributions. Fuzzy clustering algorithms have been extended to incorporate time-varying parameters and adaptive strategies to track evolving clusters [9, 22].

In summary, the state of the art in dynamic clustering encompasses various techniques such as concept drift adaptation, incremental clustering, online clustering, hybrid approaches, reinforcement learning, and fuzzy clustering. These approaches aim to handle the challenges posed by changing data distributions and evolving environments. The field continues to evolve as researchers develop novel algorithms and adapt existing methods to meet the demands of dynamic clustering in real-world applications.

Moreover, it is important to note that despite the advancements in dynamic clustering techniques, there is currently no model that can anticipate the clustering changes in a dynamic environment. The inherent complexity and unpredictability of real-world data make it challenging to accurately predict and anticipate all shifts in the underlying clustering structures.

This thesis investigates the application of Multi-Agent Deep Deterministic Policy Gradients (MADDPG) and Gaussian Mixture Model (GMM) in multi-modal distribution prediction.

We aim to evaluate the effectiveness of MADDPG and GMM in predicting the distribution of multiple modes while handling the problem of data drift. By leveraging these techniques, we seek to address the challenges posed by the creation and elimination of clusters. Through this research, we hope to contribute to the advancement of dynamic clustering methods and explore new avenues for handling complex and evolving data distributions.

2.6 Linear Algebra Background

Let's remind some results of the linear algebra that will be fundamental in the moment of defining the transition function.

Proposition 2.6.1 The sum of two symmetric and semidefinite positive matrices is symmetric and semidefinite positive.

Proof

let be $S_1, S_2 \in \mathcal{M}_{nxn}$ symmetric and semi-definite positive.

Note that $x^T S_1 x > 0$ and $x^T S_2 x > 0$ for every $x \in \mathbb{R}$ because are semi-definite positive. Then,

$$x^T S_1 x + x^T S_2 x = x^T (S_1 x + S_2 x) = x^T (S_1 + S_2) x > 0$$

as x is arbitrary $(S_1 + S_2)$ is semidefinite positive.

Now let be i, j indices of $(S_1 + S_2)$, the element $(S_1 + S_2)_{i,j}$ is equal to $(S_1)_{i,j} + (S_2)_{i,j}$ but S_1, S_2 are symmetric, so:

$$(S_1 + S_2)_{i,j} = (S_1)_{j,i} + (S_2)_{j,i} = (S_1 + S_2)_{j,i}.$$

Follow that $(S_1 + S_2)$ is symmetric and semi-definite positive.

Proposition 2.6.2 if the eigenvalues of S are non negative,S is symmetric.

Proof:

As S is symmetric, we can use the Spectral composition theorem:

$$S = Q^t D Q$$

where Q is orthogonal and D is diagonal and its values are the eigen values of S.

Let be $x \in \mathbb{R}^n$ with $x \neq 0$, $x^T S x = x^T Q^T D Q x$. But Q is orthogonal and then non singular or invertible, so exist $y \in \mathbb{R}^n$ such that $x = Q^T y$, in this way:

$$(Q^T y)^T S Q y = y^T Q Q^T D Q Q^T y$$

Since Q is orthogonal, $Q^T = Q^{-1}$. It follows that:

$$(Q^T y)^T S Q y = y^T Q Q^T D Q Q^T y = y^T D y$$

but D is a diagonal matrix with non-negative values, so

$$x^T S x = y^T D y > 0.$$

We conclude that S is semidefinite positive.

Spectral Composition Theorem

Theorem 2.6.1 For every symmetric matrix S, there exists an orthogonal matrix Q and a diagonal matrix D such that:

$$S = Q^T D Q \tag{2.25}$$

The matrix Q consists of the eigenvectors of S, and the diagonal elements of D correspond to the eigenvalues.

The proof is in [28].

Gram-Schmidt Algorithm The Gram-Schmidt algorithm is a method used to orthogonalize a set of vectors in a vector space. It takes a set of vectors as input and produces an orthogonal set of vectors that span the same subspace. The algorithm works by iteratively subtracting the projections of the previously processed vectors from the current vector, ensuring that the resulting vectors are orthogonal.

The pseudo-code of the algorithm is in 2:

Algorithm 2 Gram-Schmidt Algorithm

Require: $V = v_0, \ldots, v_m$
Ensure: $u_0 = v_0$
1: **for** $i = 1$ to m **do**
2: $\quad u_i = v_i - \sum_{k=0}^{i-1} \text{Proy}_{u_k}(v_i)$
3: **end for**

As a result of this procedure, we observe that $\langle u_i, u_j \rangle = 0$ for all $i, j \in 0, \ldots, m$ where $i \neq j$.

Chapter 3

Problem Model and Solution Algorithm

In this chapter, we will delve into the intricate details of our distribution prediction model, presenting a comprehensive mathematical formulation. Our objective is to provide a clear understanding of the model's underlying principles and illustrate how its various components collaborate to forecast joint distributions. Specifically, we will introduce the Gaussian mixture model and a partially observable Markov game as fundamental building blocks of our approach. Additionally, we will elucidate the role of the multi-agent deep deterministic policy gradient algorithm in solving this game and enabling accurate distribution prediction for Gaussian mixtures.

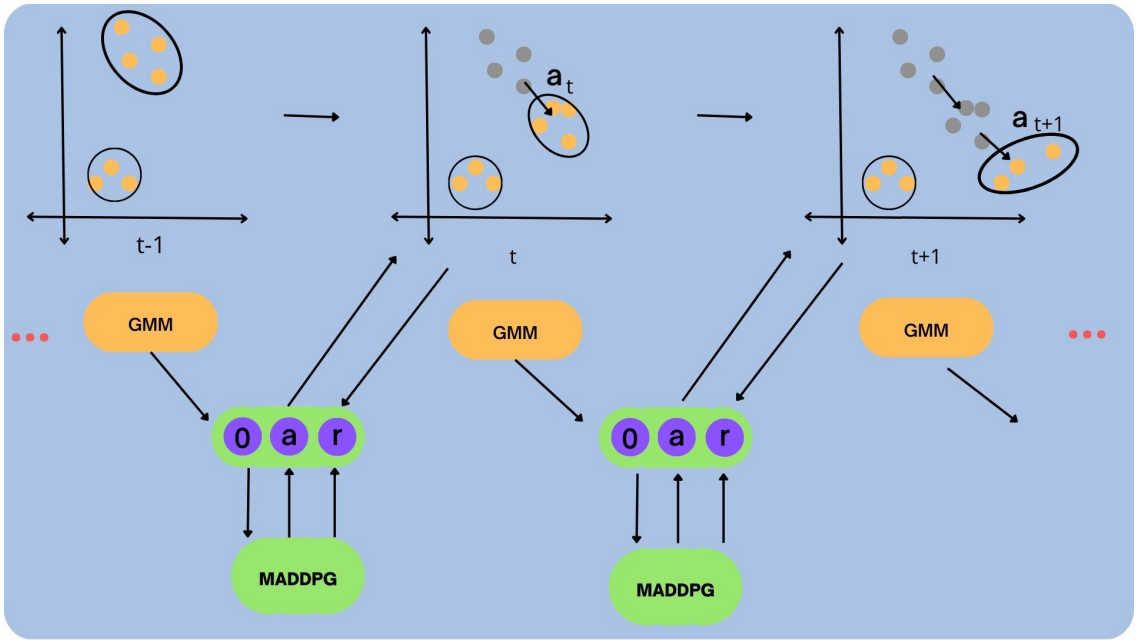
3.1 General Algorithm and Parametric Model (GMM)

As we can see in Figure 3.1, at each step, agents predict the clustering parameters for the next time period. These parameters serve as actions. The observations are derived from the parameters obtained in the current period using the Gaussian Mixture Model (GMM). Agents learn to improve clustering accuracy as they interact with the environment and receive rewards. The feedback from observations and actions enables continuous improvement in predicting dynamic clusters, optimizing the real-time learning and adaptation process.

The foundation of our model lies in the Gaussian Mixture Model (GMM). By leveraging the GMM, we can capture the inherent complexity and variability present in real-world distributions. This model represents the joint distribution as a combination of multiple Gaussian components, each characterized by its mean, covariance, and weight. The GMM offers a flexible framework for modeling a wide range of distribution patterns, making it particularly suitable for our distribution prediction task.

Let $\{\mathcal{X}_t\}_{t=0}^{T}$ be a random process such that, $\forall t \in \{0, \ldots, T\}$:

$$\mathcal{X}_t \sim q(x, \theta) = \sum_{\ell=0}^{m} w_\ell N(x; \mu_\ell, \Sigma_\ell), \tag{3.1}$$



(a)

Figure 3.1: GMM-POMG Diagram

the variable $\mathcal{X}t$ represents a random variable that follows a Gaussian mixture distribution, denoted as $N(x; \mu_\ell, \Sigma_\ell)$. This distribution characterizes the ℓ-th component within the mixture, where μ_ℓ denotes the mean, Σ_ℓ represents the covariance matrix, and w_ℓ is the weight associated with the ℓ-th component. The weights w_ℓ satisfy the condition that all weights are positive ($w_\ell > 0$) and their sum over all components is equal to one ($\sum_{\ell=0}^{m} w_\ell = 1$).

To summarize, for each time period, the random variable $\mathcal{X}_t$ follows a Gaussian mixture distribution, where the parameters $(\mu, \Sigma, w)_{\ell,t}$ vary over time. The objective is to predict the parameters $(\mu, \Sigma, w)_{\ell,t+1}$ of the Gaussian mixture for the next time period, based on the available information up to period t.

To achieve this prediction, the MADDPG (Multi-Agent Deep Deterministic Policy Gradient) algorithm is used, where each agent is a neural network that learns in a centralized manner with a common critic. Each agent provides a vector from which the parameters of the Gaussian mixture are then calculated.

The observations are obtained from the parameters generated by applying the GMM to the data from the current period. These observations can be based on the current period alone or on several periods.

The reward is based on the log-likelihood. As this increases, the agents and the critic improve their loss functions, leading them to maintain policies with higher accumulated rewards. This, in turn, implies distributions that better fit the data.

The key components of the GMM are the means, covariances, and weights of each Gaussian component. These parameters are calculated and adjusted for each time period, allowing for an accurate and flexible representation of the data distribution.

3.2 Partially Observable Markov Game

To address the challenges posed by partial observability and the dynamic nature of the environment, we incorporate a partially observable Markov game (POMG) into our model. POMG provides a formal framework for modeling decision-making in situations where agents have limited information about the current state of the environment. By formulating the distribution prediction problem as a POMG, we can effectively handle the uncertainties and incomplete observations inherent in real-world scenarios.

As was mentioned in chapter 2, MADDPG is formulated to solve a partially observable Markov game problem. If we state our problem of predicting the parameters that define the underlying gaussian mixture distribution as a POMG, we can apply MADDPG to learn the variation of these parameters. Working with this framework, defining each component of the tuple $\mathcal{G}$ is required.

3.2.1 States and Observations

In the distribution prediction model using the Gaussian mixture model (GMM) and partially observable Markov game (POMG) framework, defining the states and observations is crucial for capturing the dynamics of the system. States represent the current information about the distribution parameters, while observations provide a limited view of the environment.

Let's denote the set of agents as A and the set of time periods as P. In this framework, the states are defined as $\mathcal{S} = \bigcup_{\ell \in A, t \in P} \Theta_{\ell,t}$, where $\Theta_{\ell,t} = (\mu, \Sigma, w)_{\ell,t}$ represents the parameters of the Gaussian mixture model for agent ℓ at time period t.

Each state $\mathcal{S}_{\ell,t}$ captures the information about the mean $\mu_{\ell,t}$, covariance matrix $\Sigma_{\ell,t}$, and weight $w_{\ell,t}$ of the Gaussian component for agent ℓ at time period t. The states encapsulate the current knowledge about the distribution parameters and serve as the basis for making predictions about future distributions.

Observations, on the other hand, provide a partial view of the environment at each time step. Let's denote the observation for agent ℓ at time period t as $\mathcal{O}_{\ell,t}$. The observation $\mathcal{O}_{\ell,t}$ is derived from the state $\mathcal{S}_{\ell,t}$ and captures the changes in the parameters over a certain number of previous time periods. Specifically, $\mathcal{O}_{\ell,t}$ is defined as $\mathcal{O}_{\ell,t} = \bigcup_{t'=t-k}^{t} \psi(\mathcal{S}_t - \mathcal{S}_{t-1})$, where $\mathcal{S}_t = \bigcup_{\ell \in A} \mathcal{S}_{\ell,t}$ and ψ is a transformation such that:

$$\psi : \mathcal{S} \to \mathbb{R}^{|A| \cdot (m+1)^2}$$

$$\psi(\mathcal{S}_t) = \bigcup_{\ell \in A} (\mu, \phi(\Sigma), w)_{\ell,t}$$

Where $\mu_{\ell,t}$ represents the mean, $\phi(\Sigma)_{\ell,t}$ denotes the eigenvectors and eigenvalues of the covariance matrix Σ for agent ℓ at time period t, and $w_{\ell,t}$ represents the weights associated with the Gaussian mixture model. In particular, $\phi(\Sigma)$ is a function that returns the eigen

vectors $(v_0, \ldots, v_m)$ and eigenvalues $(a_0, \ldots, a_m)$ of the covariance matrix Σ, where m is the dataset dimension.

Note that we are "observing" the covariance changes as a representation through its eigenvectors and eigenvalues. But, why not just take the raw covariance? that's because the agents are taking the action of choosing eigenvectors and eigenvalues of the covariances, in this way, we will observe the changes in the same way our agents will take decisions, making it easier to capture patterns in the changes observed. Why we are taking the action of choosing eigenvectors and eigenvalues of the covariances? This point will be treated in the next section about the action space.

By considering k previous time periods, the observation $\mathcal{O}_{\ell,t}$ provides information about how the parameters have changed over time, allowing the model to capture temporal dependencies and patterns in the distribution. It enables agents to make informed decisions based on their limited observation of the environment's dynamics.

The states and observations play a crucial role in the decision-making process within the POMG framework. Agents use their current state $\mathcal{S}_{\ell,t}$ and limited observation $\mathcal{O}_{\ell,t}$ to choose appropriate actions that aim to update the parameters and make accurate predictions about future distributions. The observations serve as the input to the decision-making algorithm, providing agents with the necessary information to navigate the uncertain and dynamic environment.

It's important to note that the states and observations are updated at each time step as the model receives new data and estimates the parameters for the Gaussian mixture model. This continuous update allows the model to adapt to changes in the distribution and improve the accuracy of the predictions over time.

In summary, the states in the GMM-POMG approach represent the current information about the distribution parameters for each agent at a specific time period. Observations provide a limited view of the environment by capturing the changes in the parameters over a certain number of previous time periods. Both states and observations are crucial for agents to make informed decisions and accurately predict future distributions.

3.2.2 Actions

In the given framework, each agent learns and tracks changes in the parameters of the Gaussian mixture model over time. To represent the parameter changes, we define $A_\ell \in \mathbb{R}^{(m+1)^2}$, where m represents the dimension of the data. Is necessary to explain the dimension of our action space: the first m elements are because the mean μ, the next ones $m^2 + m$ are the m vectors of dimension m that represent the eigenvectors (m^2), and the m eigenvalues of the covariance changes. Finally, the last element represents the weight change of the component. If we sum the three parts we have $m + (m^2 + m) + 1 = m^2 + 2m + 1 = (m + 1)^2$. Now, let's define A_ℓ as follows:

$$\{\mu, v_0, \ldots, v_m, a_0, \ldots, a_m, w\} \tag{3.2}$$

In this representation, μ_ℓ denotes the mean of the ℓ-th component, and w_ℓ represents its

weight. However, defining the covariance parameters is more complex than the previous ones, $v_0, ..., v_m \in \mathbb{R}^m$ are vectors while $a_m \in \mathbb{R}^+$ are scalars that represent the eigenvectors and eigenvalues of a matrix. The challenge lies in ensuring that the agents provide a valid matrix for the covariance changes. Recall that Gaussian covariances are positive semi-definite and symmetric.

If we don't solve this, the agents (that are neural networks) will return values within intervals without considering whether these values define a valid positive semi-definite and symmetric matrix, undefining the Gaussian component in our mixture and doing impossible to train the model.

To deal with this, we will output m vectors and m positive real numbers. The vectors will be orthogonalized and normalized. With the resulting vectors, we create a squared matrix, and with the m positive real numbers, we will create a diagonal matrix. Using the spectral composition theorem, and imposing the real numbers be positive, we will create a valid positive semi-definite and symmetric matrix as output. However, it is important to note that this approach increases the number of output values required to represent each covariance from m^2 to $m(m+1)$. The details of how this is incorporated into the transition function will be explained in the following subsection.

3.2.3 Transition Function

In the transition function, we use the spectral composition theorem to compose the covariances from its eigenvectors and values. Then we can sum states and actions to allow the environment evolution. This ensures that the resulting covariance matrix is positive semi-definite and symmetric.

Let's define the transition function $\mathcal{T} : \mathcal{S} \times \mathcal{A}_1 \times \ldots \times \mathcal{A}_N \to \mathcal{S}$ as follows:

$$\mathcal{T}(\mathcal{S}_{\ell,t}, \mathcal{A}_{\ell,t}) = \mathcal{S}_{\ell,t} + \eta(\mathcal{A}_{\ell,t}) = \mathcal{S}_{\ell,t+1} \tag{3.3}$$

Where $\eta \colon \mathcal{A} \to \mathcal{S}$ is a transformation that receives the actions $\mathcal{A}_{\ell,t}$ in the form $(\mu, \phi(\Sigma), w)_{\ell,t}$ (recall that $\phi(\Sigma)$ are the eigenvectors and eigenvalues of Σ) and return an action representation in the space of the states i.e. the change of the parameters in the form $(\Delta\mu, \Delta\Sigma, \Delta w)_{\ell,t}$.

Note that in this way you can sum the states and the representation $\eta(\mathcal{A}_{\ell,t})$ to obtain a new state, since $\mathcal{S}_{\ell,t}$ and $\eta(\mathcal{A}_{\ell,t})$ has the same dimension. To ensure that the resulting covariances are positive semi-definite and symmetric we will use the fact that the covariance in the state $\mathcal{S}_t$ is actually positive semi-definite and symmetric, furthermore, due to the proposition Prop. 2.6.1 in the section of linear algebra background, the space of the positive semi-definite and symmetric matrix is closed to the addition. In consequence, We just need to ensure that our action is in this space (positive semi-definite and symmetric matrices).

Remind that due to the proposition 2.6.2, all matrices whose eigenvalues are non-negative are positive semi-definite. So imposing that every eigenvalue is nonnegative (re-scaling the outputs or selecting an activation function as the exponential), we are ensuring the

representation of a positive semi-definite matrix. now it only remains to compose the matrix from its eigenvectors and eigenvalues.

Let be $v'_0, ..., v'_m \in \mathbb{R}^m$ vectors and $a_m \in \mathbb{R}^+$ scalars, the representation of the covariance change action. We use the Gram-Schmidt Algorithm to convert the base $v'_0, ..., v'_m$ in the orthogonalized base $v_0, ..., v_m$. With these vectors create the matrix Q:

$$\begin{bmatrix} | & | & | & \dots & | \\ v_0 & v_1 & v_2 & \dots & v_m \\ | & | & | & \dots & | \end{bmatrix}$$

and $D = diag(a_0, \dots, a_m)$ as a diagonal matrix.

Since Q is orthogonal by construction, by virtue of the spectral composition theorem 2.6.1, the matrix:

$$\Delta\Sigma = Q^T D Q \tag{3.4}$$

is a symmetric matrix. Furthermore, is a positive semi-definite matrix since its eigenvalues are non-negative. The action that represents the covariance change is in fact $\Delta\Sigma$. And the resulting new state will have the form $\mathcal{S}_{t+1} = \bigcup_{\ell \in A}(\mu + \Delta\mu, \Sigma + \Delta\Sigma, w + \Delta w)_{\ell,t+1} = \Theta_{t+1}$.

3.2.4 Reward Function

The reward function is a critical component in the distribution prediction model, as it provides a measure of the quality of the predicted distribution parameters. In this framework, we utilize the loss function of the GMM as the basis for designing the reward function.

Let's denote the predicted distribution parameters at time period t as $\Theta_t = \widehat{\mu}_\ell, \widehat{\Sigma}_\ell, \widehat{w}_\ell$, where $\widehat{\mu}_\ell$, $\widehat{\Sigma}_\ell$, and $\widehat{w}_\ell$ represent the predicted mean, covariance matrix, and weight of the Gaussian component ℓ, respectively.

The reward function is defined based on the similarity between the predicted distribution and the actual distribution in time period $t+1$. We calculate the loss between the predicted parameters Θ_t and the true parameters obtained by running the GMM algorithm with the data from time period $t+1$. The loss function measures the discrepancy between the predicted distribution and the ground truth distribution.

In order to measure this discrepancy we will probe two reward functions that we will call the "cooperative" and the "cooperative-competitive".

Cooperative reward The reward function in the period t is formulated as follows:

$$R_1(\Theta_{t+1}) = \sum_{i=0}^{n} \log\left(\sum_{\ell=1}^{N} \widehat{w}_\ell N\left(\boldsymbol{x}_i; \widehat{\boldsymbol{\mu}}_\ell, \widehat{\boldsymbol{\Sigma}}_\ell\right)\right), \tag{3.5}$$

where $N(\boldsymbol{x}i; \widehat{\boldsymbol{\mu}}_\ell, \widehat{\boldsymbol{\Sigma}}_\ell)$ represents the probability density function of the Gaussian component ℓ evaluated at data point $\boldsymbol{x}_i$. The reward is computed by summing the logarithm of the

predicted mixture of Gaussian components for each data point. The higher the reward, the better the predicted parameters align with the true distribution.

Using the GMM loss function as the reward function, the distribution prediction model is incentivized to learn accurate and representative parameters that capture the underlying distribution's characteristics. This approach enables the agents to optimize their actions to minimize the discrepancy between the predicted distribution and the true distribution, leading to improved distribution prediction performance.

The reward function serves as a guiding signal for the reinforcement learning process, allowing the agents to learn and refine their strategies for predicting the parameters of the Gaussian mixture model.

This Partially Observable Markov Game is considered cooperative because the agents involved work towards a common goal of accurately predicting the parameters of the underlying distribution. The reward function used in this approach promotes cooperation among the agents rather than competition.

The reward function used in this approach considers the performance of the entire system rather than individual agent performance. The reward is calculated based on the overall quality of the predicted parameters. This encourages the agents to cooperate and coordinate their actions to maximize the reward for the entire system.

The agents' actions are chosen based on the collective decision-making process. Each agent's action contributes to the overall prediction of the distribution parameters. The joint action space allows the agents to coordinate their actions to optimize the prediction performance as a group, rather than acting independently.

The MADDPG process enables the agents to learn from each other's actions and experiences. By sharing information and insights, the agents can collectively improve their prediction strategies. This cooperative learning process fosters collaboration and knowledge sharing among the agents.

Cooperative-competitive reward

The reward function for agent ℓ is formulated as follows:

$$R_{2_\ell}(\Theta_{t+1}) = (1-\lambda) \cdot \sum_{i=0}^{n} \log\left(\widehat{w}_\ell N\left(\boldsymbol{x}_i; \widehat{\boldsymbol{\mu}}_\ell, \widehat{\boldsymbol{\Sigma}}_\ell\right)\right) + \lambda \cdot R_1(\Theta_t), \tag{3.6}$$

In this formulation, $\lambda \in [0,1]$ is a parameter that determines the trade-off between a competitive component and a cooperative component in the reward function.

The first term $(1-\lambda) \cdot \sum_{i=0}^{n} \log\left(\widehat{w}\ell N\left(\boldsymbol{x}i; \widehat{\boldsymbol{\mu}}\ell, \widehat{\boldsymbol{\Sigma}}\ell\right)\right)$ represents the competitive component of the reward. It evaluates the log-likelihood of the Gaussian component ℓ considering only its own weights and parameters. Each agent aims to maximize this term by adjusting its actions to increase the likelihood of its own component for the given data points.

this part of the function places emphasis on the weights of the components, it implies

that the agents are competing to increase their own weights relative to the other agents. The higher the weight assigned to a specific component by an agent, the more influential that component becomes in the overall joint distribution prediction.

In this case, the agents are motivated to adjust their actions to maximize their own weights, which can potentially result in a competitive dynamic among them. Each agent strives to assign higher weights to its own component and lower weights to the components associated with other agents. This competition arises from the fact that the total weight assigned to the components should sum to 1, imposing a constraint on the weights.

The second term $\lambda \cdot R_1(\Theta_t)$ represents the cooperative component of the reward. Here, $R_1(\Theta_t)$ refers to the cooperative reward function, which encourages the agents to collectively improve the joint distribution prediction. By including this term, the agents are incentivized to balance their actions between cooperation and competition. The parameter λ determines the relative importance of the cooperative component in the overall reward.

The presence of both the competitive and cooperative components in the reward function introduces a trade-off. The agents strive to optimize their own component's likelihood while also considering the collective performance of the joint distribution prediction. The value of λ determines the emphasis placed on cooperation versus competition.

Overall, this reward function combines both competitive and cooperative elements, allowing the agents to balance their individual objectives with the common goal of accurate joint distribution prediction. By adjusting the parameter λ, the relative influence of competition and cooperation can be controlled, providing flexibility in shaping the agent's behavior in the game.

3.3 GMM-POMG

The goal of our model is to accurately forecast joint distributions by utilizing the GMM to capture complexity and variability in real-world distributions.

The GMM represents the joint distribution as a combination of multiple Gaussian components, each characterized by its mean, covariance, and weight. The parameters of the GMM vary over time, and the objective is to predict the parameters of the Gaussian mixture for the next time period based on available information.

To address the challenges posed by partial observability and the dynamic nature of the environment, the POMG framework is incorporated. This framework models decision-making in situations where agents have limited information about the current state of the environment. By formulating the distribution prediction problem as a POMG, uncertainties and incomplete observations can be effectively handled.

The states in the model represent the current information about the distribution parameters for each agent at a specific time period. Observations provide a limited view of the environment by capturing changes in the parameters over previous time periods. Both states and observations play a crucial role in the decision-making process within the POMG

framework.

Actions in this framework represent parameter changes in the GMM. The challenge lies in ensuring that the actions result in valid positive semi-definite and symmetric covariance matrices. To address this, a transformation is applied to the actions to ensure the resulting covariance matrices maintain these properties.

The transition function combines the states and transformed actions to update the state for the next time period. By utilizing the spectral composition theorem and ensuring non-negative eigenvalues, the resulting covariance matrix is positive semi-definite and symmetric.

To understand better the general algorithm you can see the pseudo-code 3:

Algorithm 3 GMM-POMG algorithm

Initialize MADDPG parameters and hyperparameters
Collect historical data for training
Initialize actor and critic networks for each agent
Pretrain GMM parameters using historical data
for each time step **do**
    Perform GMM clustering on collected data
    Update GMM parameters Θ_t based on cluster assignments
    Receive observations $\mathcal{O}_{\ell,t}$ from the environment for each agent
    Compute actions $\mathcal{A}_{\ell,t}$ for each agent using their actor networks
    Predict GMM parameters $\eta(\mathcal{A}_{\ell,t}) = \Theta_{t+1}$ for the next period using MADDPG and cluster assignments
    Execute actions in the environment using the transition function and observe next states and rewards
    Store experiences in a replay buffer
    Sample a batch of experiences from the replay buffer
    Update actor and critic networks using the MADDPG algorithm
end for

Chapter 4

Experiments

The experiments section of this study focuses on evaluating and enhancing the performance of the GMM-POMG model for distribution prediction. Three distinct experiments have been designed to explore different aspects of the model and provide valuable insights into its capabilities in various scenarios. Each experiment is formulated with specific steps, metrics, and baselines to measure and compare the model's performance effectively.

The first experiment, "Varying the Number of Observations in the Past," investigates the impact of altering the number of previous observations on the model's log-likelihood results. By modifying the original model to use different numbers of past observations as inputs, we aim to determine the optimal number of past observations that yields the highest log-likelihood improvement. Synthetic datasets with known cluster structures are used to assess the performance, comparing the log-likelihood of the modified model's predictions with the log-likelihood obtained by applying the Gaussian mixture model (GMM) on the data at the subsequent time step. The analysis of this experiment sheds light on the significance of historical observations for accurate distribution prediction.

In the second experiment, "Varying the Reward Function," we delve into the influence of different reward functions on the GMM-POMG model's performance. By modifying the reward function used in the original model, we evaluate its effect on the log-likelihood results. Two distinct reward functions are considered: one based on the negative log-likelihood and another combining log-likelihood with clustering accuracy. Synthetic datasets with varying cluster structures are employed to compare the log-likelihood of the modified model's predictions with the log-likelihood obtained by applying GMM on the subsequent data. This experiment provides insights into the effectiveness of different reward functions in enhancing the model's performance.

The third experiment, "Creation and Elimination of Clusters," focuses on assessing the model's adaptability to changes in the number of clusters over time. By incorporating dynamic cluster creation and elimination capabilities into the GMM-POMG model, we aim to evaluate its ability to identify and adapt to changes in cluster formations. Synthetic datasets with evolving cluster structures are utilized to compare the log-likelihood of the modified model's predictions with the log-likelihood obtained through GMM. This experiment provides valuable insights into the model's robustness and adaptability in real-world scenarios where clusters

can change dynamically.

Throughout the experiments, log-likelihood serves as the primary metric for evaluating the model's performance. The parameters obtained in the last period serve as the baseline for comparison, while the log-likelihood obtained through GMM represents the best possible value. By carefully analyzing and interpreting the experimental results, we gain a deeper understanding of the GMM-POMG model's strengths, limitations, and potential for improvement in distribution prediction tasks.

4.1 Data Description

In this section, we provide a comprehensive description of the datasets used in our study. We begin by presenting an overview of the datasets, including their size, format, and structure. We then discuss the data sources and explain the characteristics of the datasets, including the type of data, the number of clusters, the type of movement, the cluster independence, and the type of change observed.

Dataset Overview

Table 4.1 presents a detailed overview of the datasets used in our study, including their size, number of features, and the duration of the train and test periods. Each dataset is labeled with a unique identifier.

Dataset	Size	Number of features	Train periods	Test periods
DataIt01	5432	2	30	10
DataIt02	5252	2	30	10
DataIt03	4986	2	30	10
DataIt04	5164	2	30	10
DataIt05	4874	2	30	10
DataIt06	2116	2	30	10
DataIt07	4954	2	30	10
Daily climate Dheli	1246	4	30	10
DataIt09	1750	2	30	10
DataIt010	820	2	30	10
DataIt011	820	2	30	10

Table 4.1: Overview of the datasets used in the study.

The datasets vary in size, ranging from 820 to 5432 instances, providing a diverse range of data points for analysis. The number of features in each dataset is either 2 or 4, representing the different types of information captured by the datasets. To ensure robust model training and evaluation, we divided the datasets into train and test periods of 30 and 10 time steps, respectively.

Data Sources

The datasets used in our study are obtained from various sources. Synthetic datasets were specifically designed to demonstrate different behaviors of data evolution, while real-world datasets were selected to evaluate the applicability of our model in dynamic scenarios.

Table 4.2 presents the data sources for the datasets used in our study, indicating whether the dataset is synthetic or real and providing the minimum and maximum number of clusters present in each dataset.

Dataset	**Data Type**	**Number of Clusters (Min)**	**Number of Clusters (Max)**
DataIt01	Synthetic	2	2
DataIt02	Synthetic	2	2
DataIt03	Synthetic	2	2
DataIt04	Synthetic	2	2
DataIt05	Synthetic	2	2
DataIt06	Synthetic	2	3
DataIt07	Synthetic	2	2
Daily climate Delhi	Real	1	2
DataIt09	Synthetic	2	3
DataIt010	Synthetic	2	2
DataIt011	Synthetic	2	2

Table 4.2: Data sources for the datasets used in the study.

The synthetic datasets, identified by the prefix "DataIt," were generated by sampling Gaussian mixture distributions and varying the parameters to add movement, changes in covariances, and changes in cluster weights. These datasets simulate different types of movement, including linear, sinusoidal, and unknown patterns. The number of clusters varies from 2 to 3, providing diverse scenarios for analysis.

The real-world dataset, Daily climate Delhi was obtained from Kaggle. The Daily climate Delhi dataset contains daily climate data for the city of Delhi, which we aggregated into monthly periods to capture climate patterns over time.

Characteristics of the Datasets

Understanding the characteristics of the datasets is crucial for interpreting the observed dynamics and evaluating the performance of our model. Table 4.3 presents the characteristics of the datasets used in our study, including the type of movement, cluster independence, and the type of change observed.

The type of movement refers to the pattern exhibited by the clusters in the datasets. Some datasets have linear movement, where the clusters move in a straight line. Other datasets exhibit additional tendencies, such as linear movement with a specific direction. The DataIt07 dataset displays sinusoidal movement, while Daily climate Dheli dataset have an unknown movement pattern although can be supposed certain seasonality since its climate data.

Cluster independence indicates whether the movement of the clusters in the dataset is equal or not. In datasets with cluster independence, the movement of each cluster is distinct, allowing for variations in speed and direction. In contrast, datasets without cluster independence show clusters moving in the same direction, making them more synchronized.

Data	Type of movement	Cluster independence	Type of change
DataIt01	linear with tendency	yes	movement
DataIt02	stacionary	yes	movement
DataIt03	linear	yes	movement
DataIt04	stationary with tendency	yes	movement
DataIt05	linear	no	movement
DataIt06	linear	yes	movement + creation
DataIt07	sinusoidal non estacionary	yes	movement
Daily climate Dheli	unknown	yes	movement
DataIt09	linear	yes	movement + elimination
DataIt010	linear	yes	movement
DataIt011	stationary with tendency	yes	movement

Table 4.3: Characteristics of the datasets used in the study.

The type of change observed in the datasets can be categorized into movement, creation, or elimination. Movement implies that the clusters change their positions over time, maintaining their existence. Creation refers to the appearance of data points forming a new cluster in the dataset. Elimination signifies that one of the clusters no longer contains any data points.

By incorporating datasets with diverse characteristics, we aim to evaluate the effectiveness of our model in capturing different types of data dynamics and detecting changes in cluster behavior.

Data Processing

Before conducting the analysis, we performed necessary preprocessing steps on the datasets. Firstly, we normalized all the data to ensure consistent scaling across features and datasets. Normalization enhances the comparability of different datasets and prevents any biases resulting from varying scales.

For the real-world datasets, Daily climate Delhi, we aggregated the data into monthly time windows to capture the broader patterns and reduce the noise associated with daily fluctuations. This aggregation allows us to focus on the overall dynamics and trends in the climate.

The processed datasets are then used to train and evaluate our proposed model for dynamic cluster analysis.

4.2 Experiments Formulation

This subsection focuses on conducting three experiments to evaluate and improve the performance of the GMM-POMG model. Experiment 1 explores the impact of varying the number of observations in the past on the model's log-likelihood results. Experiment 2 investigates the effect of different reward functions on the model's performance. Experiment 3 assesses the model's adaptability to changes in the number of clusters over time. Each experiment is formulated with specific steps, metrics, and baselines to measure the model's performance. The analysis of the results will provide valuable insights into enhancing the GMM-POMG model's capabilities in different scenarios. In general, we are going to use Loglikelihood to evaluate in the experiments. When we refer to Improvement Ratio means:

$$\text{Improvement Ratio} = \frac{(L_{\text{GMM-POMG}} - L_{\text{Baseline}})}{(L_{\text{GMM}_{t+1}} - L_{\text{Baseline}})} \tag{4.1}$$

Experiment 1 : Varying the Reward Function

In this experiment, we aim to explore the impact of different reward functions on the performance of the GMM-POMG model. We will modify the reward function used in the original model and evaluate its effect on the log-likelihood results. Specifically, we will consider two reward functions: one based on the negative log-likelihood and another based on a combination of log-likelihood and clustering accuracy. The objective is to compare the performance of the modified model using different reward functions and assess their effectiveness in improving the log-likelihood results.

We will use the synthetic datasets *DataIt01*, *DataIt02*,*DataIt03*, *DataIt04*,*DataIt05*, *DataIt07*, *DataIt09*,*DataIt010*, *DataIt011* and the real data *Daily climate Delhi* . The log-likelihood of the predictions generated by the modified model will be compared with the log-likelihood obtained by applying GMM on the data at time $t+1$. The parameters obtained in the last period (t) will serve as the baseline.

The experimental formulation for this experiment is as follows:

1. Initialize the GMM-POMG model.
2. Set the reward function to be used: $R = [R_1, R_2]$.
3. For each reward function, perform the following steps:
   (a) Train the modified GMM-POMG model on the dataset.
   (b) Generate predictions for the next period $(t + 1)$.
   (c) Calculate the log-likelihood of the predictions using GMM at time $t + 1$.
   (d) Calculate the log-likelihood of the predictions using the modified model.
   (e) Compare the log-likelihood results with the baseline (parameters at time t).
   (f) Record the results and repeat for each synthetic dataset.

4. Analyze the obtained results.
    (a) Compare the log-likelihood values for different reward functions.
    (b) Assess the effectiveness of each reward function in improving the log-likelihood results.
    (c) Evaluate the consistency of the results across different datasets.
5. Draw conclusions and discuss the implications of varying the reward function on the performance of the GMM-POMG model.

Experiment 2: Varying the Number of Observations in the Past

In this experiment, we aim to investigate the impact of varying the number of observations in the past on the performance of the proposed model. We will modify the original model by changing the number of previous observations used as input to the GMM-POMG framework. We will consider three different settings: using only the most recent observation (t), using the two most recent observations ($t-1$ and t), and using the three most recent observations ($t-2$, $t-1$, and t). The objective is to compare the log-likelihood results obtained for each setting and determine the optimal number of past observations.

To conduct this experiment, We will use the synthetic datasets *DataIt01*, *DataIt02*,*DataIt03*, *DataIt04*,*DataIt05*, *DataIt07*, *DataIt09*,*DataIt010*, *DataIt011* and the real data *Daily climate Delhi* . We will measure the log-likelihood of the predictions generated by the modified model and compare them with the log-likelihood obtained by applying GMM on the data at time $t+1$. Additionally, we will consider the parameters obtained in the last period (t) as the baseline.

The experimental formulation for this experiment is as follows:

1. Initialize the GMM-POMG model.
2. Set the number of observations in the past to be used as input: $n = [1, 2, 3]$.
3. For each value of n, perform the following steps:
    (a) Train the modified GMM-POMG model on the dataset.
    (b) Generate predictions for the next period ($t+1$).
    (c) Calculate the log-likelihood of the predictions using GMM at time $t+1$.
    (d) Calculate the log-likelihood of the predictions using the modified model.
    (e) Compare the log-likelihood results with the baseline (parameters at time t).
    (f) Record the results and repeat for each synthetic dataset.
4. Analyze the obtained results.
    (a) Compare the log-likelihood values for different values of n.
    (b) Identify the optimal number of past observations based on the highest log-likelihood improvement.

(c) Evaluate the consistency of the results across different datasets.

5. Draw conclusions and discuss the implications of varying the number of observations in the past on the performance of the GMM-POMG model.

Experiment 3: Creation and Elimination of Clusters

In this experiment, we aim to investigate the ability of the GMM-POMG model to adapt to changes in the number of clusters over time. We will modify the model to allow for the creation and elimination of clusters dynamically. The objective is to evaluate the model's capability to identify and adapt to changes in cluster formations.

We will use the synthetic datasets *DataIt06*, and *DataIt09*, where there is a creation and an elimination of clusters respectively. The log-likelihood of the predictions generated by the modified model will be compared with the log-likelihood obtained by applying GMM on the data at time $t + 1$. The parameters obtained in the last period (t) will serve as the baseline.

The experimental formulation for this experiment is as follows:

1. Initialize the GMM-POMG model with dynamic cluster creation and elimination.
2. Train the modified GMM-POMG model on the dataset.
3. Generate predictions for the next period $(t + 1)$.
4. Calculate the log-likelihood of the predictions using GMM at time $t + 1$.
5. Calculate the log-likelihood of the predictions using the modified model.
6. Compare the log-likelihood results with the baseline (parameters at time t).
7. Analyze the obtained results.
   (a) Evaluate the model's ability to adapt to changes in cluster formations.
   (b) Assess the impact of dynamic cluster creation and elimination on log-likelihood performance.
   (c) Consider the consistency of the results across different datasets.
8. Draw conclusions and discuss the implications of the GMM-POMG model's adaptability to changes in the number of clusters over time.

In each experiment, we will consider log-likelihood as the metric, use the parameters obtained in the last period (t) as the baseline, and compare the log-likelihood results with the log-likelihood obtained by applying GMM on the data at time $t + 1$ as the best possible value. The analysis of the results will provide insights into the effectiveness and adaptability of the GMM-POMG model under different experimental conditions.

	R1		R2	
Data Item	Training	Test	Training	Test
DataIt01	282.64	282.56	287.94	286.65
DataIt02	-770.22	-756.37	-962.03	-962.00
DataIt03	632.87	617.06	677.06	651.54
DataIt04	-876.46	-787.25	-873.88	-920.22
DataIt05	748.37	-300.86	617.40	613.27
DataIt06	201.48	201.39	322.48	321.39
DataIt07	-750.84	-776.61	-615.74	-654.73
DataIt08	237.11	248.36	216.77	235.48
DataIt09	-86.85	-85.96	12.99	13.09
DataIt10	-433.50	-565.27	-480.33	-636.13
Daily Climate Dheli	-502.32	-472.09	-488.44	-545.84

Table 4.4: Comparison of Loglikelihood Across Different Metrics in Experiment 1

4.3 Experiment 1 Results: Varying the Reward Function

In this section, we present the results obtained by POMG-GMM using a single period in the observation on datasets with different characteristics. In the Table **??**, we see the loglikelihoods obtained during training (with noise induced to allow exploration) and in testing, where the model encounters unseen data and no noise is applied to the agents' actions.

Additionally, in the Table **??**, we show the improvement (or deterioration) of the loglikelihood versus assuming the clustering parameters in the last period.

	R1		R2	
Data Item	Training	Test	Training	Test
DataIt01	0.1641	0.1629	0.4008	0.3949
DataIt02	0.3966	0.4045	0.4230	0.4231
DataIt03	0.2518	0.2144	0.2770	0.2182
DataIt04	0.0924	0.2898	0.1456	0.0404
DataIt05	0.5151	-6.8285	0.4931	0.4600
DataIt06	0.2922	0.2926	0.2190	0.2142
DataIt07	0.1339	0.0600	0.1537	0.0110
DataIt09	0.3340	0.3902	0.3543	0.4318
DataIt10	0.3977	0.3991	0.4769	0.4776
DataIt11	0.7100	0.5400	0.7300	0.5600
Daily climate Dheli	0.2147	0.0895	0.3453	0.2541

Table 4.5: Comparison of Improvement Ratio Across Different Reward Functions in Experiment 1

Overall, the results from the experiment comparing the Improvement Ratio (IR) for linear data in Experiment 1 indicate a mixed performance across the datasets.

The majority of datasets (DataIt01, DataIt03, DataIt06, DataIt09, , DataIt10) exhibit moderate to good Improvement Ratios in both training and test sets, suggesting a relatively stable performance of the model in predicting future data based on past observations.

Data Item	**R1 (Training)**	**R1 (Test)**	**R2 (Training)**	**R2 (Test)**
DataIt01	0.1641	0.1629	0.4008	0.3949
DataIt03	0.2518	0.2144	0.2770	0.2182
DataIt06	0.2922	0.2926	0.2190	0.2142
DataIt09	0.3340	0.3902	0.3543	0.4318
DataIt010	0.3977	0.3991	0.4769	0.4776
DataIt05	0.5151	-6.8285	0.4931	0.4600

Table 4.6: Comparison of Improvement Ratio for linear data in Experiment 1

However, there are indications of overfitting in DataIt05 within R_1. In this scenario, both clusters move in the same direction and magnitude, making it more likely for the network to memorize the movement pattern.

The table 4.7 illustrates the comparison of Improvement Ratios for stationary data in Experiment 1. Upon analysis, we observe contrasting behaviors in different datasets regarding their generalization capabilities.

For datasets such as DataIt02 and DataIt011, the Improvement Ratios (IR) for both R_1 and R_2 demonstrate consistency between the training and test sets. This consistency suggests that the model effectively generalizes its learned patterns to unseen data. The comparable IR values across both sets indicate a stable performance, indicating that the model captures the underlying patterns without overfitting to the training data.

However, DataIt04 presents a different scenario. In this dataset, there is a notable discrepancy in the Improvement Ratios between the training and test sets for both R_1 and R_2. While the model performs well in the training set, as evidenced by relatively high IR values, its performance significantly deteriorates in the test set. This discrepancy indicates potential overfitting, where the model may have memorized the patterns present in the training data without effectively generalizing to unseen data. Consequently, the model's predictions on DataIt04 in the test set may be less reliable and indicative of its true performance compared to the training set.

In summary, while some datasets exhibit stable and consistent performance across both training and test sets, others, like DataIt04, demonstrate signs of overfitting. Ensuring the model's ability to generalize effectively to unseen data is crucial for reliable predictions across different datasets.

The table 4.8 presents the comparison of Improvement Ratios for sinusoidal non-stationary data in Experiment 1. Upon analysis, it appears that DataIt07 exhibits a notable discrepancy between the Improvement Ratios in the training and test sets for both R_1 and R_2.

The low Improvement Ratios in both training and test sets suggest potential underfitting of the model to the data. In other words, the model fails to capture the underlying patterns present in DataIt07 effectively. This is indicated by the relatively low Improvement Ratios,

Data Item	R1 (Training)	R1 (Test)	R2 (Training)	R2 (Test)
DataIt02	0.3966	0.4045	0.4230	0.4231
DataIt04	0.2898	0.0924	0.1456	0.0404
DataIt011	0.7100	0.5400	0.7300	0.5600

Table 4.7: Comparison of Improvement Ratio for stationary data in Experiment 1

Data Item	R1 (Training)	R1 (Test)	R2 (Training)	R2 (Test)
DataIt07	0.1339	0.0600	0.1537	0.0110

Table 4.8: Comparison of Improvement Ratio for sinusoidal non-stationary data in Experiment 1

particularly in the test set, which implies that the model's predictions are not significantly better than the naive method.

Underfitting occurs when the model is too simple to capture the complexity of the data, resulting in poor performance on both the training and test sets. In the context of DataIt07, the model likely lacks the complexity or flexibility to accurately represent the sinusoidal non-stationary patterns present in the data, leading to suboptimal performance.

To address underfitting in this case, it may be necessary to consider more complex models or feature representations that can better capture the underlying dynamics of the sinusoidal non-stationary data. Additionally, increasing the model's capacity or incorporating domain-specific knowledge may help improve its performance on DataIt07.

Data Item	R1 (Training)	R1 (Test)	R2 (Training)	R2 (Test)
Daily Climate Dheli	0.2147	0.0895	0.3453	0.2541

Table 4.9: Comparison of Improvement Ratio for unknown data in Experiment 1

The table 4.9 illustrates the comparison of Improvement Ratios for unknown data in Experiment 1, specifically representing daily climate data from Dheli. Let's analyze this case, considering the nature of climate data and potential overfitting in R_1:

In R_1, the Improvement Ratio for the training set (0.2147) is considerably higher than that for the test set (0.0895), suggesting potential overfitting. This discrepancy implies that the model may have learned to fit the training data too closely, capturing noise or specific patterns that do not generalize well to unseen data. Overfitting in climate data, such as Daily Climate Dheli, is common due to its inherent complexity and variability. The model may have captured specific short-term fluctuations or noise in the training data, leading to inflated performance metrics on the training set but reduced performance on the test set.

On the other hand, in R_2, the Improvement Ratio for both the training and test sets (0.3453 and 0.2541, respectively) is more consistent, indicating a relatively stable performance across both datasets. This suggests that the model's performance in R_2 is less affected by overfitting compared to R_1.

Given that the data represents daily climate observations, it's reasonable to assume higher

seasonality and variability compared to other datasets, such as DataIt07. Climate data often exhibits long-term trends, seasonal patterns, and short-term fluctuations, which can pose challenges for modeling. Therefore, while overfitting may be a concern in R_1, the performance discrepancies between the training and test sets may also be attributed to the inherent complexity and variability of climate data.

Results summary

POMG-GMM exhibits better performance under certain conditions while facing challenges in others. It performs well with datasets that demonstrate linear trends, as it can effectively capture the direction and magnitude of changes over time. Additionally, POMG-GMM is suited for stationary data, where there are no significant changes in mean or variance over time, enabling it to model the underlying stationary distribution accurately. Moreover, when datasets exhibit clear and predictable patterns, such as seasonality or repetitive cycles, POMG-GMM can capture these patterns effectively, resulting in more precise predictions.

However, POMG-GMM faces limitations when dealing with nonlinear or highly non-stationary data. It struggles to model datasets that do not adhere to linear trends or exhibit significant non-stationarity. Furthermore, in the presence of high levels of noise or random disturbances, POMG-GMM may encounter difficulties in separating the signal from the noise, leading to less reliable predictions. Similarly, datasets with high variability or irregularities pose challenges for POMG-GMM, as it may struggle to model the underlying complexity accurately, resulting in less precise predictions.

The effectiveness of POMG-GMM depends on the characteristics of the data being analyzed. While it can provide accurate predictions for datasets with linear trends, stationarity, and clear patterns, it may face challenges when dealing with nonlinear, non-stationary, noisy, or highly variable datasets. Understanding the nature of the data and its suitability for POMG-GMM is crucial for obtaining reliable predictions and making informed decisions in practical applications.

Regarding the reward functions, several observations can be made:

1. Differences in performance between reward functions (R_1 and R_2):
    - Overall, it seems that R_2 has better or comparable performance in terms of log-likelihood and ratio improvement compared to R_1 in most datasets.
    - However, there are some cases where R_1 outperforms R_2, especially in the datasets DataIt02, DataIt04, and DataIt07, both in the training and test sets.
    - This suggests that introducing a cooperative-competitive component in the reward function (R_2) may improve performance in most cases, but it may not always be the best option depending on the specific characteristics of the dataset and the learning environment.
2. **Potential for adjusting reward functions:**
    - The results suggest that the choice of reward function can have a significant impact on the performance of the POMG-GMM method in different scenarios.

- This highlights the importance of carefully adjusting reward functions to suit the specific characteristics of the environment and the problem.

Overall, the results provide an interesting insight into how different reward functions can influence the performance of the POMG-GMM method in a variety of situations, and suggest potential areas for future research, such as adapting reward functions to improve performance on specific datasets.

4.4 Experiment 2 Results: Varying the Number of Observations in the Past

In this section, we present the results from our analysis of the GMM-POMG model's performance across various datasets with different characteristics, including linear trends, stationarity, and non-stationarity. The results are summarized in Table 4.10, where we compare the improvement ratios for different reward functions (R_1 and R_2) and varying numbers of observations. We aim to identify trends and patterns in the model's performance, particularly focusing on how the number of past observations impacts the improvement ratio for each type of dataset. This analysis will help in understanding the model's strengths and limitations, and guide future enhancements to improve its applicability to diverse data patterns.

In the analysis of different types of data in relation to the growth of observations, distinct trends in the improvement ratio are observed for each dataset.

For linear datasets with trend, such as DataIt01, DataIt03, DataIt05, and DataIt010, different patterns stand out. In DataIt01, it is observed that the improvement ratio tends to stabilize after two observations for both metrics R1 and R2. In DataIt03, there is no clear trend in the improvement ratio with increasing observations. On the other hand, in DataIt05, a notable increase in the improvement ratio is observed with the increase in observations for both metrics. In DataIt010, the improvement ratio is relatively stable with the increase in observations.

Regarding stationary datasets, such as DataIt02, DataIt04, and DataIt011, different behaviors are observed. DataIt02 shows little variation regardless of the number of observations. In DataIt04, there is significant variability in the improvement ratio with the increase in observations, yet the performance remains poor in all cases. Finally, in DataIt011, no clear trend in the improvement ratio with increasing observations is observed.

For the non-stationary sinusoidal dataset, represented by DataIt07, significant variability in the improvement ratio is observed with the increase in observations. However, the results show poor performance compared to other datasets, suggesting that the method is unable to learn non-stationary patterns.

In the case of the daily climate Dheli dataset, the improvement ratio fluctuates with the number of observations but does not present a clear increasing or decreasing pattern. It has the best performance with 2 observations using R_1 and with 1 observation using R_2. In other cases, it shows signs of overfitting due to poor performance on test data. This suggests that, with good parameters, the model is capable of learning part of the behavior with seasonality and noise. It is important to consider that the choice of the number of observations is a crucial hyperparameter, especially in stationary data, to improve the model performance and avoid overfitting.

In general, there is no pattern indicating that increasing the number of observations necessarily improves performance. For datasets with linear movements and seasonality, it is important to adjust this hyperparameter to enhance learning and avoid overfitting.

Data Item	Observations	R1		R2	
		Training	Test	Training	Test
DataIt01	1	0.1708	0.4018	0.1613	0.3959
DataIt01	2	0.3391	0.3478	0.1818	0.2791
DataIt01	3	0.3483	0.3434	0.1836	0.2797
DataIt02	1	0.3970	0.4230	0.4040	0.4230
DataIt02	2	0.3970	0.4230	0.4060	0.4240
DataIt02	3	0.3970	0.4230	0.3990	0.4240
DataIt03	1	0.2500	0.2770	0.2130	0.2180
DataIt03	2	0.2130	0.2170	0.2160	0.2180
DataIt03	3	0.2150	0.2140	0.3070	0.2860
DataIt04	1	0.0890	0.1460	0.2940	0.0400
DataIt04	2	0.2350	0.0480	0.1660	0.0480
DataIt04	3	0.0690	-0.0180	0.2400	0.0580
DataIt05	1	0.1180	0.0990	0.1200	0.1030
DataIt05	2	0.1200	0.1020	0.1270	0.1100
DataIt05	3	0.5160	0.4990	0.5010	0.4820
DataIt07	1	0.1450	0.1540	0.0620	0.0110
DataIt07	2	0.0590	0.0150	-6.7840	-7.8730
DataIt07	3	0.1370	0.0660	0.1390	0.1100
DataIt010	1	0.3970	0.4770	0.3970	0.4780
DataIt010	2	0.4130	0.4730	0.0410	0.0010
DataIt010	3	0.3990	0.4790	0.5980	0.6440
DataIt011	1	0.7160	0.7400	0.5370	0.5680
DataIt011	2	0.5370	0.5690	0.3320	0.3610
DataIt011	3	-0.3380	-0.1730	0.7320	0.7460
Dheli	1	0.2150	0.0890	0.3490	0.2540
Dheli	2	0.3450	0.2380	0.2160	0.0900
Dheli	3	0.2150	0.0890	0.2150	0.0890

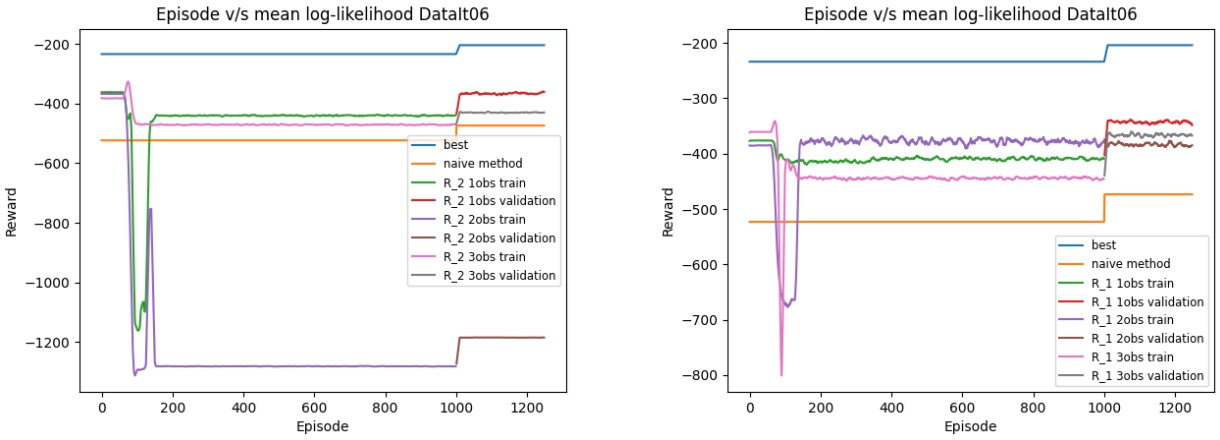
Table 4.10: Comparison of Improvement Ratio Across Different Reward Function with Varying Observations

Additionally, the model's performance on highly non-stationary data is poor for all numbers of observations.

4.5 Experiment 3 Results: Creation and Elimination of Clusters

In this experiment, we investigated the ability of the GMM-POMG model to adapt to changes in the number of clusters over time. The model was modified to allow for the dynamic creation and elimination of clusters. We evaluated the model's capability to identify and adapt to changes in cluster formations using the synthetic datasets *DataIt06* and *DataIt09*.

4.5.1 DataIt06



(a) (b)

Figure 4.1: Learning curves of the method with different numbers of observations for DataIt06. (a) Method within the reward function R_1. (b) Method within the reward function R_2.

We utilized the *DataIt06* dataset, which exhibits the creation of a cluster over time. This allowed us to assess the model's ability to adapt to such changes dynamically.

Table 5.52 presents the loglikelihood achieved by the model when handling *DataIt06* with different numbers of periods. The table includes the loglikelihood values for both the training and validation sets, considering both reward functions R_1 and R_2.

Number of Periods	R_1 Loglikelihood (Train)	R_1 Loglikelihood (Test)	R_2 Loglikelihood (Train)	R_2 Loglikelihood (Test)
1	-409.67	-342.95	-440.03	-366.10
2	-380.96	-383.69	-1281.13	-1184.54
3	-445.88	-366.46	-471.00	-429.94

Table 4.11: Loglikelihood achieved by the model when handling DataIt06 with different numbers of periods.

The learning curves for *DataIt06* are depicted in Figure 5.21a and Figure 5.21b. Figure 5.21a shows the learning curve of the method within the reward function R_1, while Figure 5.21b shows the learning curve within the reward function R_2.

Table 5.53 shows the improvement ratios achieved by the model when handling *DataIt06*. These ratios represent the improvement of the modified model over the baseline (parameters at time t), considering both the training and validation sets for reward functions R_1 and R_2.

Number of Periods	R_1 Improvement Ratio (Train)	R_1 Improvement Ratio (Test)	R_2 Improvement Ratio (Train)	R_2 Improvement Ratio (Test)
1	0.392	0.481	0.288	0.394
2	0.493	0.335	-2.618	-2.651
3	0.269	0.392	0.180	0.159

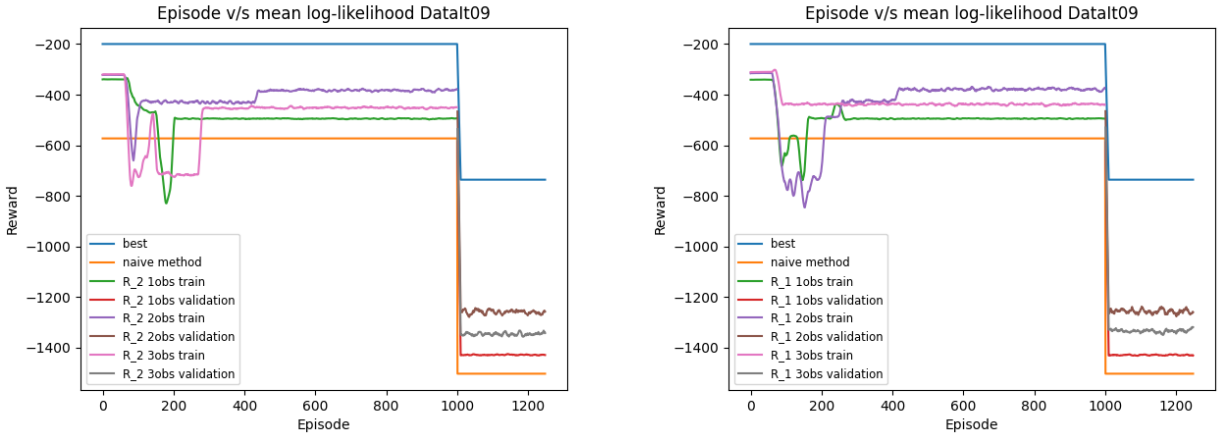
Table 4.12: Improvement ratios achieved by the model when handling DataIt06 with different numbers of periods.

The results indicate that the GMM-POMG model was able to adapt to changes in cluster formations in *DataIt06*, considering the creation of a new cluster over time. The improvement ratios demonstrate the model's ability to achieve better loglikelihood performance compared to the baseline. However, it is worth noting that in the case of *DataIt06*, the improvement ratios for reward function R_2 in the second period were significantly negative. This can be attributed to poor convergence, possibly due to a low number of episodes, as indicated by the low loglikelihood values.

These findings highlight the adaptability of the GMM-POMG model to changes in the number of clusters over time.

4.5.2 DataIt09

We utilized the *DataIt09* dataset, which involves the elimination of a cluster over time. This allowed us to evaluate the model's ability to adapt to such changes dynamically.



(a) (b)

Figure 4.2: Learning curves of the method with different numbers of observations for DataIt09. (a) Method within the reward function R_1. (b) Method within the reward function R_2.

Table 5.54 presents the loglikelihood achieved by the model when handling *DataIt09* with different numbers of periods. The table includes the loglikelihood values for both the training and validation sets, considering both reward functions R_1 and R_2.

Number of Periods	R_1 Loglikelihood (Train)	R_1 Loglikelihood (Test)	R_2 Loglikelihood (Train)	R_2 Loglikelihood (Test)
1	-494.21	-1429.93	-494.27	-1429.24
2	-380.00	-1257.25	-381.58	-1259.29
3	-437.94	-1334.08	-451.33	-1346.74

Table 4.13: Loglikelihood achieved by the model when handling DataIt09 with different numbers of periods.

The learning curves for *DataIt09* are depicted in Figure 5.22a and Figure 5.22b. Figure 5.22a shows the learning curve of the method within the reward function R_1, while Figure 5.22b shows the learning curve within the reward function R_2.

Table 5.55 shows the improvement ratios achieved by the model when handling *DataIt09*. These ratios represent the improvement of the modified model over the baseline (parameters at time t), considering both the training and validation sets for reward functions R_1 and R_2.

Number of Periods	R_1 Improvement Ratio (Train)	R_1 Improvement Ratio (Test)	R_2 Improvement Ratio (Train)	R_2 Improvement Ratio (Test)
1	0.211	0.111	0.211	0.111
2	0.516	0.333	0.513	0.330
3	0.363	0.234	0.326	0.218

Table 4.14: Improvement ratios achieved by the model when handling DataIt09 with different numbers of periods.

The results demonstrate that the GMM-POMG model was able to adapt to changes in cluster formations in *DataIt09*, which involved the elimination of a cluster over time. The

improvement ratios indicate the model's ability to achieve better loglikelihood performance compared to the baseline. These findings provide evidence of the GMM-POMG model's adaptability to changes in the number of clusters over time.

Furthermore, it is interesting to note that the improvement ratios for both reward functions R_1 and R_2 consistently showed positive values, indicating a positive impact on loglikelihood performance.

4.6 Conclusions from Experiments

In this section, we synthesize the findings from our experiments on the GMM-POMG model, highlighting its strengths, weaknesses, and key considerations for future research and application. The GMM-POMG model, designed for predicting data distributions and adapting to changes over time, has shown varying degrees of effectiveness depending on the complexity of the data patterns. Here, we provide a comprehensive analysis of the model's performance, identifying scenarios where it excels and areas where it faces challenges. This evaluation aims to inform future improvements and guide its application to different types of data.

Strengths

- The model performs well with data exhibiting simpler patterns, such as linear trends or stationarity. This suggests its potential for tasks like forecasting trends or analyzing historical data with consistent behavior.
- The model can learn from past observations to improve predictions for data with some non-linearities, but careful tuning is required to avoid overfitting.
- When trained properly, the model can learn from past observations and predict future distributions on similar data.
- The model can dynamically adapt to changes in the number of clusters over time, making it suitable for data distributions that undergo modifications.

Weaknesses

- The model is susceptible to overfitting, particularly with data that exhibits no significant change over time.
- The model struggles with data exhibiting intricate patterns. Its current design might be too simple to capture complex relationships within the data.
- The effectiveness of the model depends on the data type. There's no one-size-fits-all solution.

Key Considerations

- The number of past observations used by the model is a crucial parameter that needs to be carefully chosen based on the data type to balance learning and avoid overfitting.
- Understanding the data characteristics is essential for tailoring the model's application and parameter selection.
- Further research is needed to explore improvements to the model's ability to handle complex data patterns and potentially improve its convergence properties.

Overall Conclusions

The GMM-POMG model shows promise for distribution prediction tasks, particularly with simpler data patterns and evolving cluster structures. However, its susceptibility to overfitting and limitations with complex data dynamics necessitate careful consideration and potential model development for broader applicability. The model's ability to adapt to changes in cluster numbers makes it suitable for data distributions that undergo modifications over time. Future research efforts should focus on enhancing the model's robustness to overfitting and its ability to handle more complex data patterns.

Chapter 5

Conclusion

This thesis delves into the realm of temporal data analysis, aiming to explore the effectiveness of an enhanced clustering method compared to the simplistic approach of conducting standard clustering at each time period t and then assuming those parameters for $t + 1$. Through conducting a series of experiments and evaluating the results, we have gained valuable insights into the potential of our proposed method, as well as its limitations, as demonstrated in the conclusions drawn from the experiments.

We have presented a novel methodology that integrates Multi-Agent Deep Deterministic Policy Gradients (MADDPG) and Gaussian Mixture Model (GMM) to address the challenging task of multi-modal distribution prediction. Our proposed approach capitalizes on the strengths of both MADDPG and GMM while effectively tackling the complexities associated with dynamic clustering and distribution prediction. The primary objective of our research was to develop a unified framework capable of handling multi-modal distribution prediction in scenarios involving multiple agents.

Traditional methods often overlook the subtle interactions and dependencies within complex datasets, leading to suboptimal results. Our exploration of treating GMM components as agents has shown that embracing alternative approaches can unlock valuable insights and lead to more robust and accurate analyses.Our exploration of treating GMM components as agents has led us to capture the intricate interactions and coordination among different data modes. By leveraging this perspective, we have gained a deeper understanding of the underlying dynamics within the data, transcending the limitations of traditional GMM-based methods. This breakthrough opens up new possibilities for more accurate and insightful data analysis in a wide array of applications.

Our first key contribution lies in the integration of MADDPG and GMM, creating a synergy that harnesses the strengths of both techniques. MADDPG is renowned for its ability to learn decentralized policies for multi-agent systems, enabling agents to make informed decisions in a collaborative and coordinated manner. On the other hand, GMM is adept at modeling multi-modal distributions, enabling us to represent complex data patterns with multiple underlying modes. By unifying these two approaches, our methodology achieves a balance between decentralized learning and comprehensive distribution modeling.

Our method outperforms regular Gaussian mixture clustering in a significant number of cases, demonstrating its ability to capture the temporal dynamics inherent in the data. This improvement is particularly remarkable in instances like DataIt10, where our method showcased superior clustering accuracy, indicating its effectiveness in handling time-evolving patterns.

In the second experiment, we examined the use of more periods in observations to understand its effect on the clustering performance. Interestingly, the results did not reveal a clear pattern, but it was evident that employing more observation periods could yield better clustering accuracy in some cases. While this experiment introduces a computational cost, the potential improvements in accuracy justify exploring different values for this parameter. The dynamic nature of temporal data often requires considering multiple time periods to capture the evolving patterns effectively. Thus, by carefully selecting the number of observation periods, our approach offers a flexible and adaptive solution for handling temporal data with varying complexities.

Moving on to the third experiment, we specifically evaluated the performance of our method for cluster creation and elimination. The results indicated that our approach demonstrated comparable performance between different numbers of periods used and reward functions. While certain variations were observed, the overall performance of our method consistently outperformed the baseline, illustrating its robustness and effectiveness in handling cluster dynamics. The ability to efficiently adapt to changes in cluster formations and identify meaningful patterns in temporal data enhances the value and applicability of our approach for various real-world applications.

Our exploration has been centered on two distinct reward functions, defining competitive and cooperative games. Through a series of experiments using both synthetic and real-world data, we have gained valuable insights into the performance and behavior of these games and the potential applications of this approach in various domains.

In synthetic data experiments, both reward functions exhibited similar performances, indicating that in controlled settings, the choice of the reward function might not significantly impact the results. However, the real-world data experiments presented a contrasting picture. In this scenario, the competitive reward function outperformed the cooperative one. This discrepancy sheds light on the importance of considering real-world complexities when designing and implementing data analysis techniques.

Our research has highlighted several potential applications of this approach across various domains. In customer behavior analysis, for instance, viewing customers as interacting agents can unveil hidden patterns of influence. This knowledge can empower businesses to tailor their marketing strategies and product offerings, ultimately enhancing customer satisfaction and driving growth.

However, it is crucial to acknowledge that our method's performance improvement is not universal across all datasets. We have observed cases where the enhancement is relatively modest like in DataIt07 where there is a non-stationary movement. This can be attributed to the inherent nature of the data, where the movement of clusters does not strictly follow a Markov process. In such instances, temporal dependencies may not be adequately captured,

leading to only marginal enhancements in clustering performance.

Despite these limitations, the positive outcomes of our research underscore the potential benefits of exploring alternative techniques for clustering temporal data. Our results support the notion that traditional periodic clustering may not be the better approach when dealing with time-dependent patterns. By recognizing the complexities associated with temporal data and addressing the inherent challenges, we pave the way for further advancements in the field of Dynamic Clustering.

In conclusion, our study has advanced the field of temporal data analysis by demonstrating the benefits of our proposed method over traditional clustering techniques, particularly when the evolution of clusters over time is not considered. We have showcased the potential of our method to enhance clustering accuracy and elucidated areas where further improvements can be made. Our findings support the idea that a one-size-fits-all approach may not be suitable for analyzing temporal data and highlight the importance of tailoring techniques to suit the unique characteristics of the data at hand.

As with any scientific investigation, there are limitations that should be acknowledged. The effectiveness of our method might be contingent on various factors, such as data characteristics, clustering algorithm selection, and the choice of parameters. Further studies exploring these variables could lead to even more refined methodologies for analyzing temporal data.

Looking ahead, we identify an intriguing avenue for future research: the application of transformer-based approaches in the context of Dynamic Clustering. As witnessed in various fields, transformers have remarkably performed in handling sequential data due to their ability to capture long-range dependencies. Replacing actors and critics with transformers could potentially unlock new insights and foster better understanding of temporal dependencies by considering data sequences rather than just static views.

In summary, our research contributes to the growing body of knowledge in this area and provides valuable insights for researchers and practitioners seeking to unlock the full potential of temporal data clustering. By embracing the challenges posed by temporal data, and continuing to innovate and refine techniques, we can build a solid framework for more accurate and meaningful analysis of time-evolving patterns across diverse domains.

Bibliography

[1] Sanjeevan Ahilan and Peter Dayan. Correcting experience replay for multi-agent communication, 2020.

[2] Wesam Ashour, Malcolm Crowe, and Colin Fyfe. A family of novel clustering algorithms. volume 4224, pages 283–290, 09 2006.

[3] Wesam Barbakh and Colin Fyfe. Clustering with reinforcement learning. In Hujun Yin, Peter Tino, Emilio Corchado, Will Byrne, and Xin Yao, editors, *Intelligent Data Engineering and Automated Learning - IDEAL 2007*, pages 507–516, Berlin, Heidelberg, 2007. Springer Berlin Heidelberg.

[4] Gopal Behera and Neeta Nain. A comparative study of big mart sales prediction. In Neeta Nain, Santosh Kumar Vipparthi, and Balasubramanian Raman, editors, *Computer Vision and Image Processing*, pages 421–432, Singapore, 2020. Springer Singapore.

[5] Christopher M Bishop. *Pattern recognition and machine learning*. Springer, 2006.

[6] Abdelhamid Bouchachia. Dynamic clustering. Evolving Systems, 2012.

[7] Rolando Cavazos-cadena. Denumerable controlled markov chains with average reward criterion: Sample path optimality. *ZOR Zeitschrift für Operations Research Mathematical Methods of Operations Research*, page n. pag, 1995. Print.

[8] Wenlong Chen, Xiaoling Wang, Zhijian Cai, ChangXin Liu, YuShan Zhu, and Weiwei Lin. Dp-gmm clustering-based ensemble learning prediction methodology for dam deformation considering spatiotemporal differentiation. *Knowledge-Based Systems*, 222:106964, 2021.

[9] Fernando Crespo and Richard Weber. A methodology for dynamic data mining based on fuzzy clustering. *Fuzzy Sets and Systems*, 150(2):267–284, 2005.

[10] Arthur P. Dempster, Nan M. Laird, and Donald B. Rubin. The expectation-maximization algorithm. *Journal of the Royal Statistical Society, Series B (Methodological)*, 39(1):1–38, 1977.

[11] Eugene Feinberg and Adam Shwartz. *Handbook of Markov Decision Processes: Methods and Applications*, volume 40. 01 2002.

[12] Bernd Fritzke. A growing neural gas network learns topologies. In G. Tesauro, D. Touretzky, and T. Leen, editors, *Advances in Neural Information Processing Systems*, volume 7. MIT Press, 1994.

[13] João Gama, Indrė Žliobaitė, Albert Bifet, Mykola Pechenizkiy, and A. Bouchachia. A survey on concept drift adaptation. *ACM Computing Surveys (CSUR)*, 46:1 – 37, 2014.

[14] Adán José-García and Wilfrido Gómez-Flores. A survey of cluster validity indices for automatic data clustering using differential evolution. In *Proceedings of the Genetic and Evolutionary Computation Conference*, pages 314–322, New York, NY, USA, 2021. Association for Computing Machinery.

[15] Stephan Kolassa. Evaluating predictive count data distributions in retail sales forecasting. *International Journal of Forecasting*, 32(3):788–803, 2016.

[16] Timothy P. Lillicrap, Jonathan J. Hunt, Alexander Pritzel, Nicolas Heess, Tom Erez, Yuval Tassa, David Silver, and Daan Wierstra. Continuous control with deep reinforcement learning, 2015.

[17] Timothy P. Lillicrap, Jonathan J. Hunt, Alexander Pritzel, Nicolas Heess, Tom Erez, Yuval Tassa, David Silver, and Daan Wierstra. Continuous control with deep reinforcement learning. In Yoshua Bengio and Yann LeCun, editors, *ICLR*, 2016.

[18] Michael L. Littman. Markov games as a framework for multi-agent reinforcement learning. In *Proceedings of the Eleventh International Conference on International Conference on Machine Learning*, ICML'94, page 157–163, San Francisco, CA, USA, 1994. Morgan Kaufmann Publishers Inc.

[19] Ryan Lowe, Yi Wu, Aviv Tamar, Jean Harb, Pieter Abbeel, and Igor Mordatch. Multi-agent actor-critic for mixed cooperative-competitive environments. *CoRR*, abs/1706.02275, 2017.

[20] Volodymyr Mnih, Koray Kavukcuoglu, David Silver, Alex Graves, Ioannis Antonoglou, Daan Wierstra, and Martin A. Riedmiller. Playing atari with deep reinforcement learning. *CoRR*, abs/1312.5602, 2013.

[21] Igor Mordatch and Pieter Abbeel. Emergence of grounded compositional language in multi-agent populations, 2017.

[22] Sivaguru Munusamy and Punniyamoorthy Murugesan. Modified dynamic fuzzy c-means clustering algorithm–application in dynamic customer segmentation. *Applied Intelligence*, 50(6):1922–1942, 2020.

[23] G. Peters and R. Weber. DCC: a framework for dynamic granular clustering. *Granular Computing*, 1(1):1–11, 2016.

[24] Philip Protter. *Stochastic Integration and Differential Equations*. Springer, 1990.

[25] Daniel Revuz and Marc Yor. *Continuous martingales and Brownian motion*. Number 293 in Grundlehren der mathematischen Wissenschaften. Springer, Berlin [u.a.], 3. ed edition, 1999.

[26] David Silver, Guy Lever, Nicolas Heess, Thomas Degris, Daan Wierstra, and Martin Riedmiller. Deterministic policy gradient algorithms. *31st International Conference on Machine Learning, ICML 2014*, 1, 06 2014.

[27] Masashi Sugiyama. Chapter 15 - maximum likelihood estimation for gaussian mixture model. In Masashi Sugiyama, editor, *Introduction to Statistical Machine Learning*, pages 157–168. Morgan Kaufmann, Boston, 2016.

[28] Fred E. Szabo. *The Linear Algebra Survival Guide*. John Wiley & Sons, 2015.

[29] Fredy Troncoso and Richard Weber. A novel approach to detect associations in criminal networks. *Decision Support Systems*, 128:113159, 2020.

[30] Feng Wang, Fanshu Liao, Yixuan Li, and Hui Wang. A new prediction strategy for dynamic multi-objective optimization using gaussian mixture model. *Information Sciences*, 580:331–351, 2021.

[31] Lilian Weng. Policy gradient algorithms. *lilianweng.github.io*, 2018.

[32] Le Yao and Zhiqiang Ge. Scalable semisupervised gmm for big data quality prediction in multimode processes. *IEEE Transactions on Industrial Electronics*, 66(5):3681–3692, 2019.

[33] Chiyuan Zhang, Oriol Vinyals, Remi Munos, and Samy Bengio. A study on overfitting in deep reinforcement learning, 2018.

[34] Sicheng Zhao, Hongxun Yao, and Xiaolei Jiang. Predicting continuous probability distribution of image emotions in valence-arousal space. In *Proceedings of the 23rd ACM International Conference on Multimedia*, MM '15, page 879–882, New York, NY, USA, 2015. Association for Computing Machinery.

[35] Alaettin Zubaroğlu and Volkan Atalay. Data stream clustering: A review. *Artificial Intelligence Review*, 54(2):1201–1236, 2021.

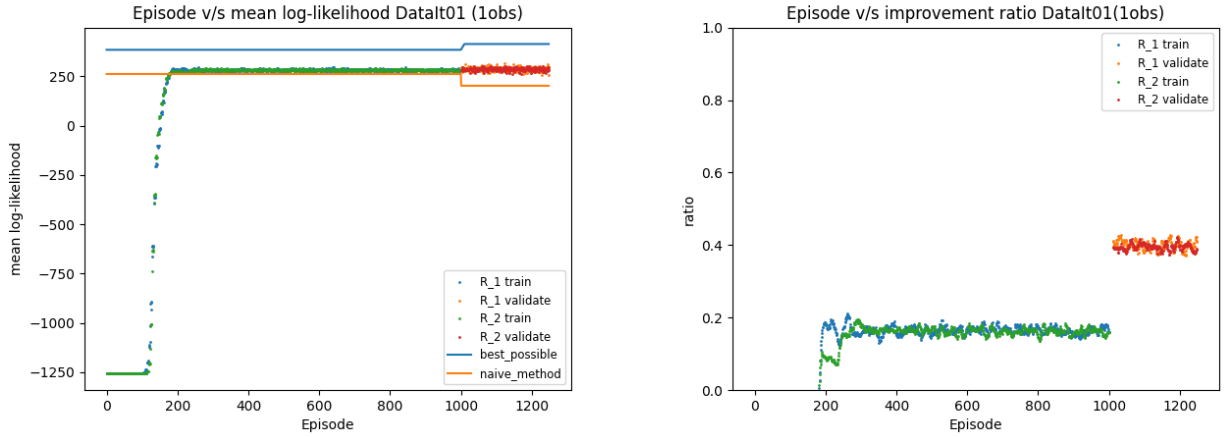
Annex

A Experiment 1 results: Varying the Reward Function

In this experiment, we investigate the effect of different reward functions on the learning performance of the agent. We evaluate two reward functions, R_1 and R_2, using the DataIt01 dataset. The learning curves, log-likelihoods, improvement ratios, and standard deviations are presented below.

DataIt01 Learning Curve

Figure 5.1 displays the learning curves obtained with rewards R_1 and R_2 for DataIt01. The x-axis represents the number of iterations, while the y-axis represents the log-likelihood.



(a) (b)

Figure 5.1: Learning curve obtained with rewards R_1 and R_2 for DataIt01 (a) and the improvement ratio compared with the baseline (b)

Log-Likelihood Results

Table 5.1 presents the mean log-likelihood values obtained with rewards R_1 and R_2 during training and validation on the DataIt01 dataset. The mean log-likelihood values are calculated for both the training and validation sets.

Improvement Ratio Results

Dataset	R_1 Mean Train	R_2 Mean Train	R_1 Mean Validate	R_2 Mean Validate
DataIt01	282.64	282.56	287.94	286.65

Table 5.1: Mean log-likelihood obtained with rewards R_1 and R_2 for DataIt01

Table 5.2 presents the improvement ratios obtained with rewards R_1 and R_2 during training and validation on the DataIt01 dataset. The improvement ratio represents the percentage of improvement in log-likelihood compared to the baseline.

Dataset	R_1 Ratio Train	R_2 Ratio Train	R_1 Ratio Validate	R_2 Ratio Validate
DataIt01	0.1641	0.1629	0.4008	0.3949

Table 5.2: Improvement ratio obtained with rewards R_1 and R_2 for DataIt01

Standard Deviation Results

Table 5.6 presents the standard deviations of the log-likelihood values obtained with rewards R_1 and R_2 during training and validation on the DataIt01 dataset. The standard deviation provides a measure of the variability of the log-likelihood values.

Dataset	R_1 Std. Dev. Train	R_2 Std. Dev. Train	R_1 Std. Dev. Validate	R_2 Std. Dev. Validate
DataIt01	4.03	3.50	10.33	8.05

Table 5.3: Standard deviation of the log-likelihood obtained with rewards R_1 and R_2 for DataIt01

From the results obtained with the DataIt01 dataset, we can observe the following:

Both reward functions, R_1 and R_2, achieved similar mean log-likelihood values during training and validation. This indicates that both rewards were effective in guiding the learning process.

The improvement ratios for both rewards were positive, indicating an improvement in log-likelihood compared to the baseline. The improvement ratios were higher for the validation set, suggesting that the agent's performance generalized well to unseen data.

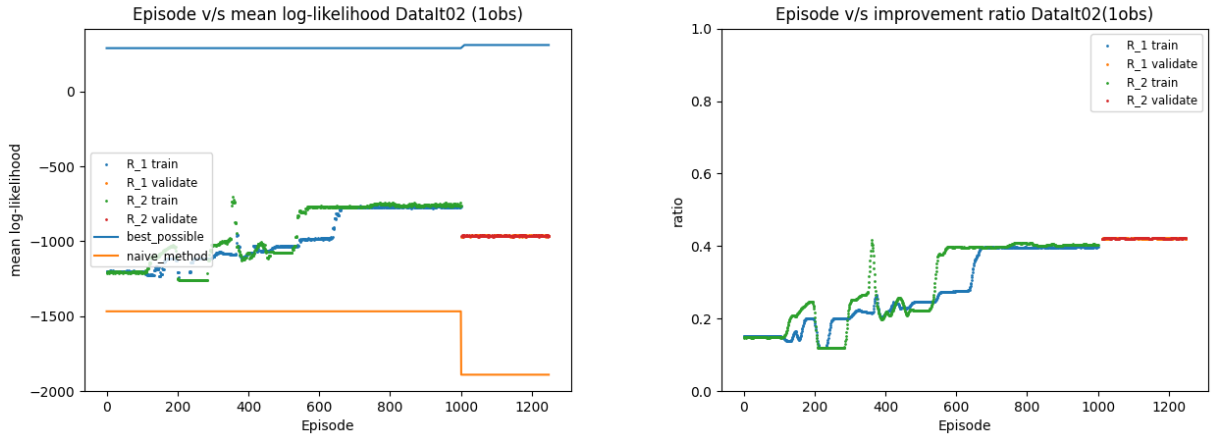
The standard deviations of the log-likelihood values indicate the variability of the learning process. Lower standard deviations imply more stable learning. In this case, R_2 had slightly lower standard deviations compared to R_1, indicating a relatively more stable learning process.

Overall, both reward functions showed promising results in terms of achieving higher log-likelihood and improvement ratios. The choice between R_1 and R_2 could depend on other factors, such as computational efficiency or specific requirements of the task at hand.

DataIt02

Figure 5.2a illustrates the learning curve obtained with rewards R_1 and R_2 for the **DataIt02** dataset. The plot shows the steady increase in log-likelihood values during both the training and validation periods.

Figure 5.2b presents the improvement ratio compared to the baseline. The graph demonstrates the improvement achieved by our proposed model, supporting the numerical results presented in Table 5.5.



(a) (b)

Figure 5.2: Learning curve obtained with rewards R_1 and R_2 for DataIt02 (a) and the improvement ratio compared to the baseline (b)

Table 5.4 shows the log-likelihood obtained with rewards R_1 and R_2 for the **DataIt02** dataset. During the training period, the mean log-likelihood values were -770.22 for R_1 and -756.37 for R_2. For the validation period, the mean log-likelihood values were -962.03 for R_1 and -962.00 for R_2. These results indicate the model's performance in fitting the data and its consistency on unseen data.

Table 5.4: Log-likelihood obtained with rewards R_1 and R_2 for DataIt02

Data	R_1 **Mean (Train)**	R_2 **Mean (Train)**	R_1 **Mean (Validation)**	R_2 **Mean (Validation)**
DataIt02	-770.22	-756.37	-962.03	-962.00

Table 5.5 presents the improvement ratios of log-likelihood compared to the baseline model. During the training period, the improvement ratios were 0.3966 for R_1 and 0.4045 for R_2. In the validation period, the improvement ratios were 0.4230 for R_1 and 0.4231 for R_2. These ratios demonstrate the significant enhancement achieved by our proposed model.

Table 5.5: Improvement ratio of log-likelihood obtained with rewards R_1 and R_2 for DataIt02

Data	R_1 **Ratio (Train)**	R_2 **Ratio (Train)**	R_1 **Ratio (Validation)**	R_2 **Ratio (Validation)**
DataIt02	0.3966	0.4045	0.4230	0.4231

To assess the variability of the model's performance, Table 5.6 displays the standard deviation of the log-likelihood values. During the training period, the standard deviations

were 2.82 for R_1 and 6.39 for R_2. In the validation period, the standard deviations increased to 12.71 for R_1 and 13.07 for R_2. These values indicate the variability in the model's performance when applied to unseen data.

Table 5.6: Standard deviation of log-likelihood obtained with rewards R_1 and R_2 for DataIt02

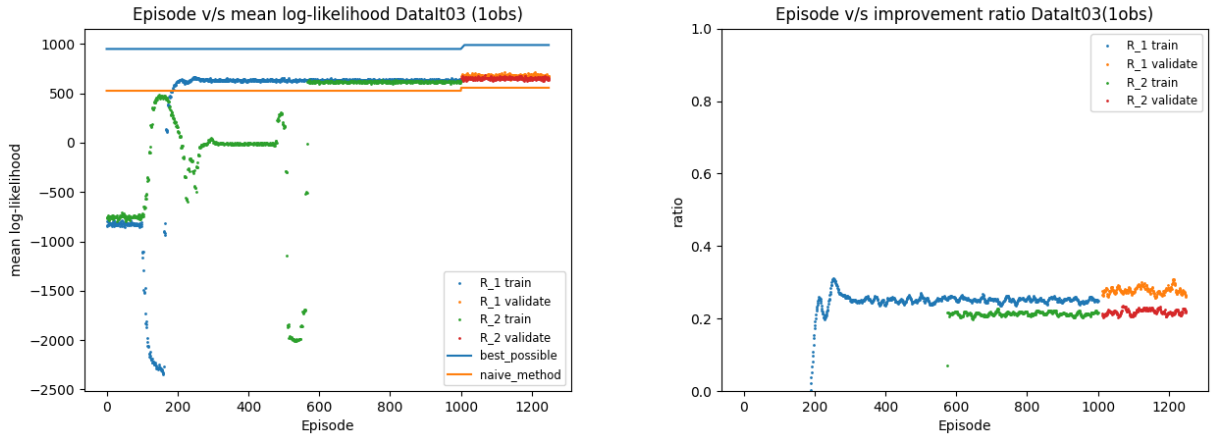
Data	R_1 **Std. Dev. (Train)**	R_2 **Std. Dev. (Train)**	R_1 **Std. Dev. (Validation)**	R_2 **Std. Dev. (Validation)**
DataIt02	2.82	6.39	12.71	13.07

These results collectively confirm the effectiveness of our proposed model in capturing the dynamic behavior of clusters in the **DataIt02** dataset, as evidenced by improved log-likelihood values and consistent performance across the training and validation periods.

DataIt03

In this section, we present the results obtained from the experiments conducted on the **DataIt03** dataset. The analysis focuses on log-likelihood values, improvement ratios, and standard deviation, providing insights into the model's performance in capturing the dynamics and changes in cluster behavior within the dataset.

The learning curve obtained with rewards R_1 and R_2 for **DataIt03** is depicted in Figure 5.3. Figure 5.3a displays the log-likelihood values during the training and validation periods, while Figure 5.3b shows the improvement ratios compared to the baseline. These figures enable us to assess the model's training progress and its ability to capture the underlying patterns in the data.



(a) (b)

Figure 5.3: Learning curve obtained with rewards R_1 and R_2 for DataIt03 (a) and the improvement ratio compared to the baseline (b)

Table 5.7 presents the log-likelihood values obtained with rewards R_1 and R_2 for **DataIt03**. The mean log-likelihood values for the training and validation periods are reported. In the training period, the mean log-likelihood values for R_1 and R_2 are 632.87 and 617.06, respectively. In the validation period, the mean log-likelihood values are 677.06 for R_1 and 651.54 for R_2. These values reflect how well the model captures the underlying cluster dynamics within the dataset.

Table 5.7: Log-likelihood obtained with rewards R_1 and R_2 for DataIt03.

Data	R_1 **Mean (Train)**	R_2 **Mean (Train)**	R_1 **Mean (Validation)**	R_2 **Mean (Validation)**
DataIt03	632.87	617.06	677.06	651.54

The improvement ratios of the log-likelihood values obtained with rewards R_1 and R_2 for **DataIt03** are shown in Table 5.8. These ratios compare the model's performance to a baseline. For the training period, the improvement ratios are 0.2518 and 0.2144 for R_1 and R_2, respectively. In the validation period, the improvement ratios are 0.2770 for R_1 and 0.2182 for R_2. These ratios highlight the effectiveness of the proposed model in capturing the changes and patterns in the clusters within the dataset.

Table 5.8: Improvement ratio of log-likelihood obtained with rewards R_1 and R_2 for DataIt03.

Data	R_1 **Ratio (Train)**	R_2 **Ratio (Train)**	R_1 **Ratio (Validation)**	R_2 **Ratio (Validation)**
DataIt03	0.2518	0.2144	0.2770	0.2182

The standard deviation values of the log-likelihood obtained with rewards R_1 and R_2 for **DataIt03** are presented in Table 5.9. These values measure the variability of the log-likelihood estimates. For the training period, the standard deviation values for R_1 and R_2 are 7.64 and 7.05, respectively. In the validation period, the standard deviation values are 13.53 for R_1 and 11.01 for R_2. These values provide insights into the stability and consistency of the model's performance in capturing the dynamics of the clusters.

Table 5.9: Standard deviation of log-likelihood obtained with rewards R_1 and R_2 for DataIt03.

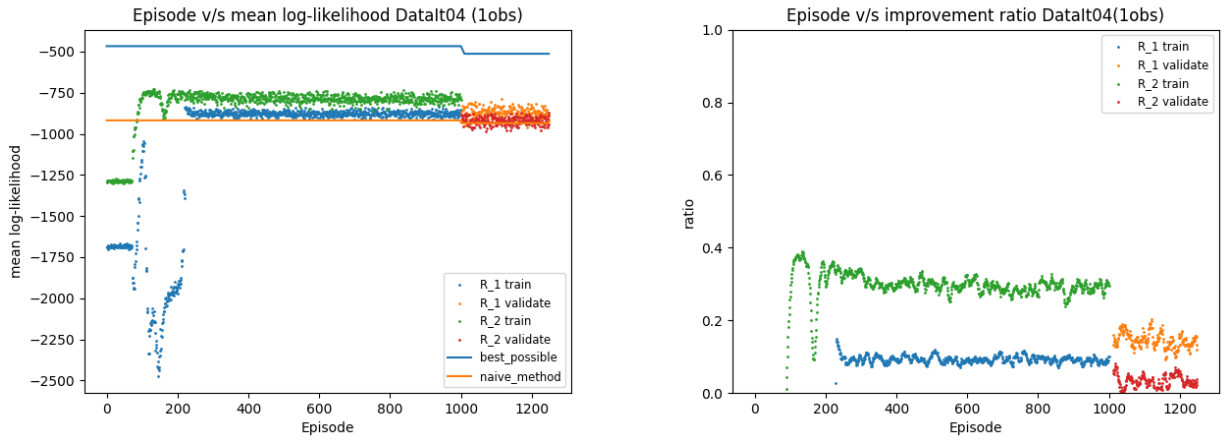
Data	R_1 **Std. Dev. (Train)**	R_2 **Std. Dev. (Train)**	R_1 **Std. Dev. (Validation)**	R_2 **Std. Dev. (Validation)**
DataIt03	7.64	7.05	13.53	11.01

Overall, the results obtained from the analysis of **DataIt03** demonstrate the model's ability to capture the dynamics and changes in cluster behavior within the dataset. The log-likelihood values, improvement ratios, and standard deviation values provide a comprehensive evaluation of the model's performance.

DataIt04

In this section, we present the results obtained from the experiments conducted on the **DataIt04** dataset. We analyze the log-likelihood values, improvement ratios, and standard deviation to evaluate the model's performance in capturing the dynamics and changes in cluster behavior within the dataset.

Figure 5.4a illustrates the learning curve for **DataIt04**. It displays the log-likelihood values during the training and validation periods. The log-likelihood values provide insights into how well the model captures the underlying patterns and dynamics of the clusters within the dataset. Figure 5.4b shows the improvement ratios compared to the baseline. These ratios quantify the model's performance improvement and highlight its ability to capture changes in cluster behavior.



(a) (b)

Figure 5.4: Log-likelihood and improvement ratio for DataIt04.

Table 5.10 presents the log-likelihood values obtained with rewards R_1 and R_2 for **DataIt04**. The mean log-likelihood values for the training and validation periods are reported. In the training period, the mean log-likelihood values for R_1 and R_2 are -876.46 and -787.25, respectively. In the validation period, the mean log-likelihood values are -873.88 for R_1 and -920.22 for R_2. These values reflect the model's ability to capture the underlying cluster dynamics within the dataset.

Table 5.10: Log-likelihood obtained with rewards R_1 and R_2 for DataIt04.

Data	R_1 **Mean (Train)**	R_2 **Mean (Train)**	R_1 **Mean (Validation)**	R_2 **Mean (Validation)**
DataIt04	-876.46	-787.25	-873.88	-920.22

The improvement ratios of the log-likelihood values obtained with rewards R_1 and R_2 for **DataIt04** are shown in Table 5.11. These ratios compare the model's performance to a baseline. For the training period, the improvement ratios are 0.0924 and 0.2898 for R_1 and R_2, respectively. In the validation period, the improvement ratios are 0.1456 for R_1 and 0.0404 for R_2. These ratios demonstrate the effectiveness of the model in capturing changes and patterns in the clusters within the dataset.

Table 5.11: Improvement ratio of log-likelihood obtained with rewards R_1 and R_2 for DataIt04.

Data	R_1 **Ratio (Train)**	R_2 **Ratio (Train)**	R_1 **Ratio (Validation)**	R_2 **Ratio (Validation)**
DataIt04	0.0924	0.2898	0.1456	0.0404

The standard deviation values of the log-likelihood obtained with rewards R_1 and R_2 for **DataIt04** are presented in Table 5.12. These values measure the variability of the log-likelihood estimates. For the training period, the standard deviation values for R_1 and R_2 are 14.11 and 20.88, respectively. In the validation period, the standard deviation values are 32.46 for R_1 and 24.92 for R_2. These values provide insights into the stability and consistency of the model's performance in capturing the dynamics of the clusters.

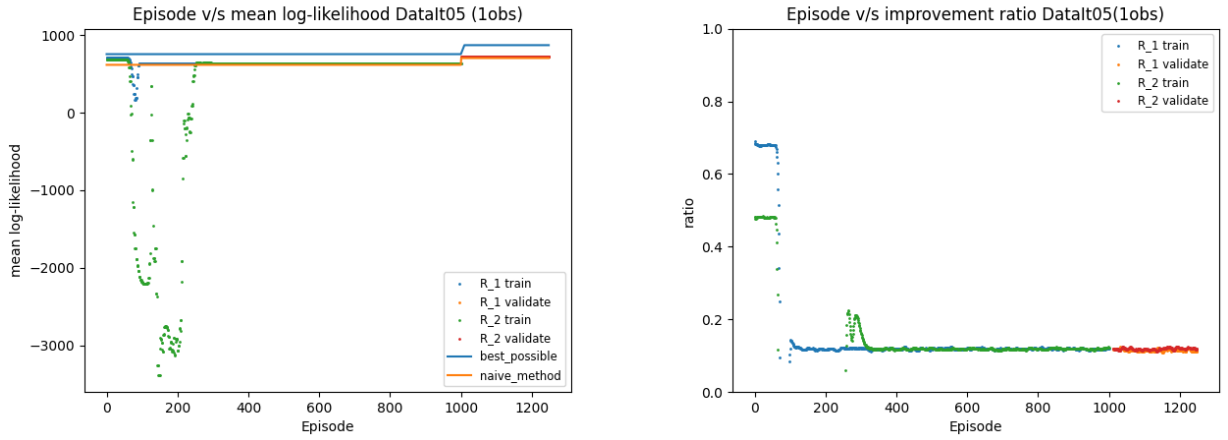
Table 5.12: Standard deviation of log-likelihood obtained with rewards R_1 and R_2 for DataIt04.

Data	R_1 **Std. Dev. (Train)**	R_2 **Std. Dev. (Train)**	R_1 **Std. Dev. (Validation)**	R_2 **Std. Dev. (Validation)**
DataIt04	14.11	20.88	32.46	24.92

In summary, when comparing the results of rewards R_1 and R_2 on the **DataIt04** dataset, R_1 achieves higher mean log-likelihood values and lower standard deviation, indicating better performance in capturing the underlying cluster behavior. On the other hand, R_2 shows higher improvement ratios in the training, suggesting more significant performance improvements compared to the baseline, suggesting overfitting. Although the results are poor in its training, R_1 performs significantly better in this experiment.

DataIt05

The obtained results are presented in Figure 5.5 and Tables 5.13, 5.14, and 5.15.



(a) (b)

Figure 5.5: Log-likelihood and improvement ratio for DataIt05

Figure 5.5 depicts the log-likelihood (Figure 5.5a) and improvement ratio (Figure 5.5b) for the DataIt05 dataset. In Figure 5.5a, we observe the log-likelihood values obtained with rewards R_1 and R_2. The log-likelihood values for R_1 show a mean value of 748.37 (Train) and 617.40 (Validation), while for R_2, the mean values are -300.86 (Train) and 613.27 (Validation). It can be seen that the model achieves a higher log-likelihood with R_1 on the Train set, but R_2 outperforms on the Validation set.

Regarding the improvement ratio presented in Figure 5.5b, we can observe that R_1 achieves a ratio of 0.5151 (Train) and 0.4931 (Validation), indicating an improvement in the log-likelihood. However, R_2 shows negative improvement ratios of -6.8285 (Train) and 0.4600 (Validation), suggesting a decrease in the log-likelihood. These results imply that R_1 contributes to better clustering accuracy, while R_2 negatively affects the clustering performance.

Table 5.13 provides further details on the mean log-likelihood values obtained with rewards R_1 and R_2 for the DataIt05 dataset. The table confirms the findings from the figures, showing the mean log-likelihood values and highlighting the difference between the Train and Validation sets.

Table 5.13: Log-likelihood obtained with rewards R_1 and R_2 for DataIt05

Data	R_1 **Mean (Train)**	R_2 **Mean (Train)**	R_1 **Mean (Validation)**	R_2 **Mean (Validation)**
DataIt05	748.37	-300.86	617.40	613.27

Table 5.14 displays the improvement ratio of log-likelihood obtained with rewards R_1 and R_2 for the DataIt05 dataset. As seen in the table, the improvement ratios for R_1 are positive, indicating an increase in the log-likelihood, while the ratios for R_2 are negative, implying a decrease in the log-likelihood. These results align with the observations made in Figure 5.5b.

Table 5.14: Improvement ratio of log-likelihood obtained with rewards R_1 and R_2 for DataIt05

Data	R_1 **Ratio (Train)**	R_2 **Ratio (Train)**	R_1 **Ratio (Validation)**	R_2 **Ratio (Validation)**
DataIt05	0.5151	-6.8285	0.4931	0.4600

Lastly, Table 5.15 presents the standard deviation of log-likelihood obtained with rewards R_1 and R_2 for the DataIt05 dataset. The standard deviations show the variation in the log-likelihood values. We can observe that the standard deviation values for R_1 are relatively low, indicating consistency in the log-likelihood values, whereas R_2 exhibits higher standard deviation values, suggesting more variability in the log-likelihood.

Table 5.15: Standard deviation of log-likelihood obtained with rewards R_1 and R_2 for DataIt05

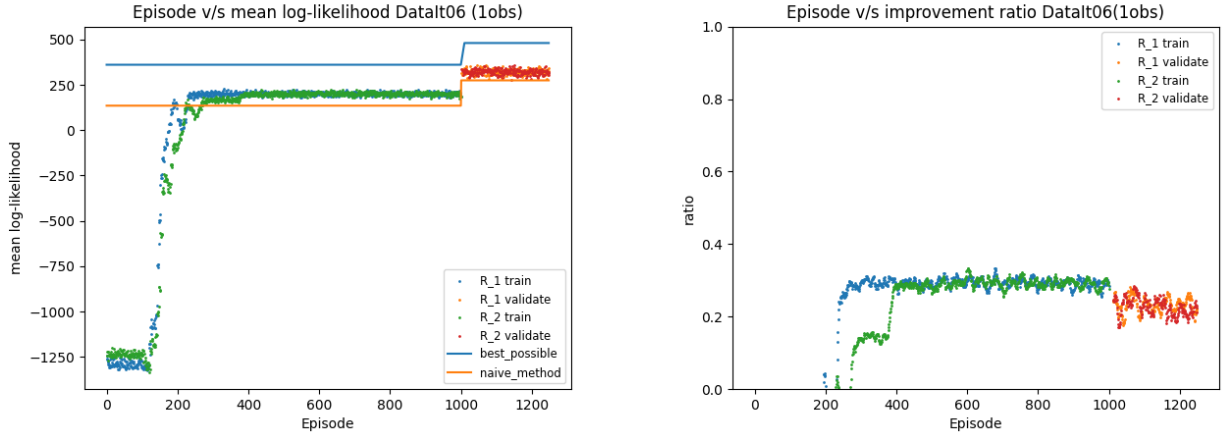
Data	R_1 **Std. Dev. (Train)**	R_2 **Std. Dev. (Train)**	R_1 **Std. Dev. (Validation)**	R_2 **Std. Dev. (Validation)**
DataIt05	3.91	1423.12	9.46	10.06

In summary, the analysis of the log-likelihood, improvement ratio, and standard deviation for the DataIt05 dataset suggests that R_1 contributes to a higher log-likelihood and better clustering accuracy on the Train set, while R_2 performs better on the Validation set. However, R_2 shows negative improvement ratios and higher variability in the log-likelihood values, indicating a potential trade-off between clustering accuracy and stability. Further investigation is needed to understand the underlying factors causing these differences and make informed decisions regarding the choice of reward function.

DataIt06

In this section, we present the results obtained from the experiments conducted on the **DataIt06** dataset. We analyze the log-likelihood values, improvement ratios, and standard deviation to evaluate the model's performance in capturing the dynamics and changes in cluster behavior within the dataset.

Figure 5.6a illustrates the learning curve for **DataIt06**. It shows the log-likelihood values during the training and validation periods, providing insights into how well the model captures the underlying patterns and dynamics of the clusters within the dataset. Figure 5.6b presents the improvement ratios compared to the baseline, quantifying the model's performance improvement and its ability to capture changes in cluster behavior.



(a) (b)

Figure 5.6: Log-likelihood and improvement ratio for DataIt06.

Table 5.16 presents the log-likelihood values obtained with rewards R_1 and R_2 for **DataIt06**. The mean log-likelihood values for the training and validation periods are reported. In the training period, the mean log-likelihood values for R_1 and R_2 are 201.48 and 201.39, respectively. In the validation period, the mean log-likelihood values are 322.48 for R_1 and 321.39 for R_2. These values reflect the model's ability to capture the underlying cluster dynamics within the dataset.

Table 5.16: Log-likelihood obtained with rewards R_1 and R_2 for DataIt06.

Data	R_1 **Mean (Train)**	R_2 **Mean (Train)**	R_1 **Mean (Validation)**	R_2 **Mean (Validation)**
DataIt06	201.48	201.39	322.48	321.39

The improvement ratios of the log-likelihood values obtained with rewards R_1 and R_2 for **DataIt06** are shown in Table 5.17. These ratios compare the model's performance to a baseline. For the training period, the improvement ratios are 0.2922 for R_1 and 0.2926 for R_2. In the validation period, the improvement ratios are 0.2190 for R_1 and 0.2142 for R_2. These ratios indicate the effectiveness of the model in capturing changes and patterns in the clusters within the dataset.

Table 5.17: Improvement ratio of log-likelihood obtained with rewards R_1 and R_2 for DataIt06.

Data	R_1 **Ratio (Train)**	R_2 **Ratio (Train)**	R_1 **Ratio (Validation)**	R_2 **Ratio (Validation)**
DataIt06	0.2922	0.2926	0.2190	0.2142

The standard deviation values of the log-likelihood obtained with rewards R_1 and R_2 for **DataIt06** are presented in Table 5.18. These values measure the variability of the log-likelihood estimates. For the training period, the standard deviation values for R_1 and R_2 are 8.87 and 8.51, respectively. In the validation period, the standard deviation values are 16.60 for R_1 and 16.64 for R_2. These values provide insights into the stability and consistency of the model's performance in capturing the dynamics of the clusters.

Table 5.18: Standard deviation of log-likelihood obtained with rewards R_1 and R_2 for DataIt06.

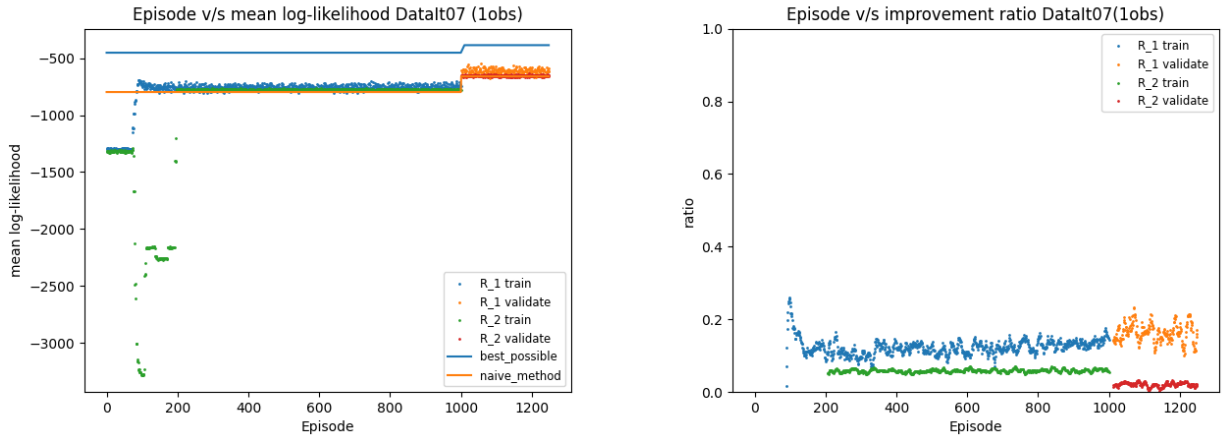
Data	R_1 **Std. Dev. (Train)**	R_2 **Std. Dev. (Train)**	R_1 **Std. Dev. (Validation)**	R_2 **Std. Dev. (Validation)**
DataIt06	8.87	8.51	16.60	16.64

Overall, the results obtained from the analysis of **DataIt06** demonstrate the model's ability to capture the dynamics and changes in cluster behavior within the dataset. The log-likelihood values, improvement ratios, and standard deviation values are really similar. In this case there are not evidence of significant differences in the performance of the two rewards function.

DataIt07

The results for the "DataIt07" dataset are displayed in Figure 5.7a and Figure 5.7b, along with tables presenting the log-likelihood, improvement ratios, and standard deviations for rewards R_1 and R_2.

Figure 5.7a shows the log-likelihood values obtained during training and validation for rewards R_1 and R_2. It can be observed that both R_1 and R_2 exhibit relatively low log-likelihood values throughout the training process. This indicates that the clustering models struggle to capture the underlying patterns and structure of the "DataIt07" dataset effectively.



(a) (b)

Figure 5.7: Log-likelihood and improvement ratio for DataIt07

Table 5.19 summarizes the log-likelihood values for the "DataIt07" dataset. The mean log-likelihood values for R_1 are -750.84 (train) and -615.74 (validation), while the mean log-likelihood values for R_2 are -776.61 (train) and -654.73 (validation). Although R_1 achieves slightly higher log-likelihood values than R_2, the overall log-likelihood values for both rewards are low.

Table 5.19: Log-likelihood obtained with rewards R_1 and R_2 for DataIt07

Data	R_1 **Mean (Train)**	R_2 **Mean (Train)**	R_1 **Mean (Validation)**	R_2 **Mean (Validation)**
DataIt07	-750.84	-776.61	-615.74	-654.73

Table 5.20 presents the improvement ratios, representing the relative performance enhancement of rewards R_1 and R_2 compared to the baseline. The improvement ratios for both rewards are also quite low, indicating limited improvement over the baseline.

Table 5.20: Improvement ratio of log-likelihood obtained with rewards R_1 and R_2 for DataIt07

Data	R_1 **Ratio (Train)**	R_2 **Ratio (Train)**	R_1 **Ratio (Validation)**	R_2 **Ratio (Validation)**
DataIt07	0.1339	0.0600	0.1537	0.011

Table 5.21 shows the standard deviations of log-likelihood. Both R_1 and R_2 exhibit relatively high standard deviations during training and validation, indicating variability in their clustering performance.

Table 5.21: Standard deviation of log-likelihood obtained with rewards R_1 and R_2 for DataIt07

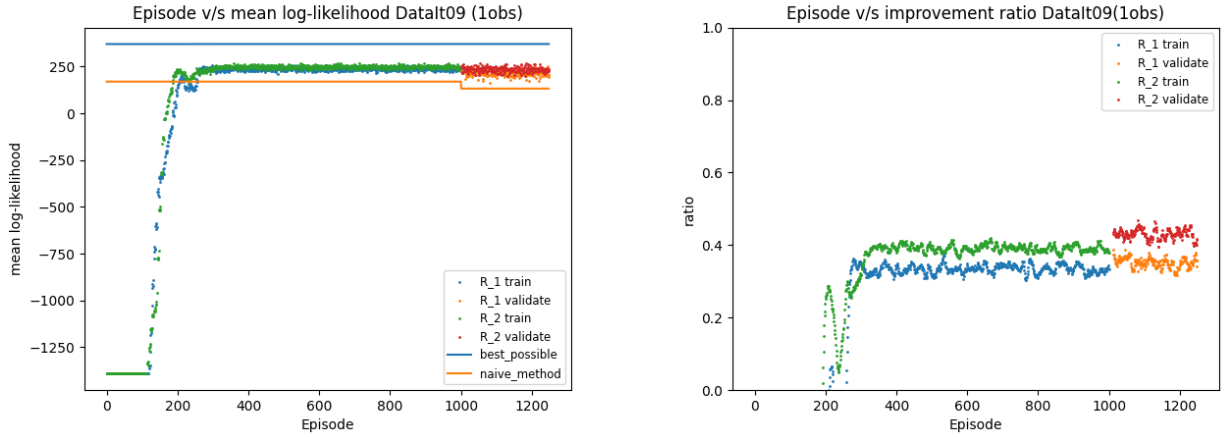
Data	R_1 **Std. Dev. (Train)**	R_2 **Std. Dev. (Train)**	R_1 **Std. Dev. (Validation)**	R_2 **Std. Dev. (Validation)**
DataIt07	18.72	4.36	26.35	10.03

In conclusion, based on the provided results, it can be inferred that neither R_1 nor R_2 achieve satisfactory performance on the "DataIt07" dataset. While R_1 shows slightly better log-likelihood values and improvement ratios compared to R_2, both rewards demonstrate poor performance. This suggests that the non-stationarity of the data may pose challenges for clustering algorithms. Further investigations and improvements in modeling techniques may be necessary to address the difficulties associated with the "DataIt07" dataset.

DataIt09

The results for the DataIt09 dataset are presented in Figure 5.8a and Figure 5.8b, along with the corresponding tables showing the log-likelihood, improvement ratios, and standard deviations for rewards R_1 and R_2.

Figure 5.8a shows the log-likelihood values obtained during training and validation for both rewards R_1 and R_2. It can be observed that R_2 achieves slightly higher log-likelihood values compared to R_1 in both the training and validation phases.



(a) (b)

Figure 5.8: Log-likelihood and improvement ratio for DataIt09

The log-likelihood values are summarized in 5.22. For DataIt09, the mean log-likelihood values for R_1 are 237.11 (train) and 216.77 (validation), while the mean log-likelihood values for R_2 are 248.36 (train) and 235.48 (validation).

Table 5.22: Log-likelihood obtained with rewards R_1 and R_2 for DataIt09

Data	R_1 **Mean (Train)**	R_2 **Mean (Train)**	R_1 **Mean (Validation)**	R_2 **Mean (Validation)**
DataIt09	237.11	248.36	216.77	235.48

The improvement ratios, as shown in 5.23, indicate the relative performance improvement of rewards R_1 and R_2 over the baseline. In the case of DataIt09, both rewards demonstrate improvement ratios greater than 0, indicating better performance compared to the baseline. However, R_2 consistently exhibits higher improvement ratios compared to R_1, indicating more significant performance improvements.

The improvement ratios are summarized in 5.23. For DataIt09, the improvement ratios for R_1 are 0.3340 (train) and 0.3543 (validation), while the improvement ratios for R_2 are 0.3902 (train) and 0.4318 (validation).

The standard deviations of log-likelihood are summarized in 5.24. For DataIt09, the standard deviations for R_1 are 7.4657 (train) and 14.9019 (validation), while the standard deviations for R_2 are 6.9465 (train) and 13.4188 (validation).

Table 5.23: Improvement ratio of log-likelihood obtained with rewards R_1 and R_2 for DataIt09

Data	R_1 **Ratio (Train)**	R_2 **Ratio (Train)**	R_1 **Ratio (Validation)**	R_2 **Ratio (Validation)**
DataIt09	0.3340	0.3902	0.3543	0.4318

Table 5.24: Standard deviation of log-likelihood obtained with rewards R_1 and R_2 for DataIt09

Data	R_1 **Std. Dev. (Train)**	R_2 **Std. Dev. (Train)**	R_1 **Std. Dev. (Validation)**	R_2 **Std. Dev. (Validation)**
DataIt09	7.4657	6.9465	14.9019	13.4188

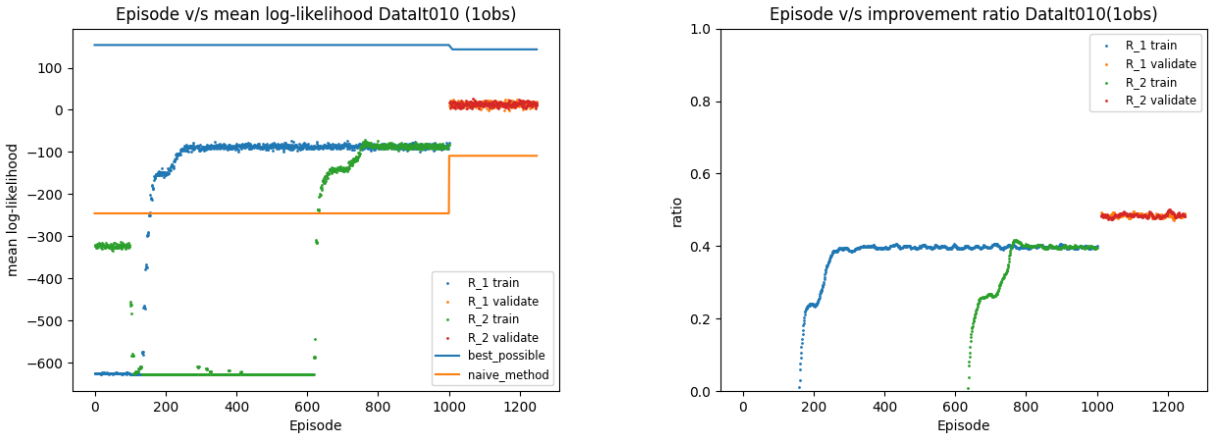
Based on these results, it can be concluded that R_2 outperforms R_1 in terms of log-likelihood, improvement ratios, and standard deviations for the DataIt09 dataset. The higher log-likelihood values achieved by R_2 indicate a better fit of the Gaussian mixture model to the data, leading to improved clustering performance. Additionally, the higher improvement ratios of R_2 suggest more significant enhancements compared to the baseline.

Therefore, considering the overall performance metrics and improvements observed, we can conclude that R_2 is a better choice for the DataIt09 dataset. It provides more accurate and reliable clustering results, making it preferable for this specific task.

DataIt010

The results for the DataIt010 dataset are shown in Figure 5.9a and Figure 5.9b, accompanied by tables presenting the log-likelihood, improvement ratios, and standard deviations for rewards R_1 and R_2.

Figure 5.9a illustrates the log-likelihood values obtained during training and validation for rewards R_1 and R_2. In this case, it can be observed that both rewards yield similar log-likelihood values, with no apparent significant difference between them.



(a) (b)

Figure 5.9: Log-likelihood and improvement ratio for DataIt010

Table 5.25 summarizes the log-likelihood values for DataIt010. The mean log-likelihood values for R_1 are -86.85 (train) and 12.99 (validation), while the mean log-likelihood values for R_2 are -85.96 (train) and 13.09 (validation). The difference in mean log-likelihood between the two rewards is minimal.

Table 5.25: Log-likelihood obtained with rewards R_1 and R_2 for DataIt010

Data	R_1 **Mean (Train)**	R_2 **Mean (Train)**	R_1 **Mean (Validation)**	R_2 **Mean (Validation)**
DataIt010	-86.85	-85.96	12.99	13.09

Table 5.26 presents the improvement ratios, indicating the relative performance improvement of rewards R_1 and R_2 over the baseline. For DataIt010, both rewards demonstrate improvement ratios greater than 0, suggesting better performance compared to the baseline. However, the difference in improvement ratios between R_1 and R_2 is negligible, indicating similar performance enhancements.

Table 5.26: Improvement ratio of log-likelihood obtained with rewards R_1 and R_2 for DataIt010

Data	R_1 **Ratio (Train)**	R_2 **Ratio (Train)**	R_1 **Ratio (Validation)**	R_2 **Ratio (Validation)**
DataIt010	0.3977	0.3991	0.4769	0.4776

Table 5.27 provides the standard deviations of log-likelihood, which indicate the stability and consistency of the clustering results. In this case, both rewards exhibit relatively low standard deviations, further supporting the similarity in their performance.

Table 5.27: Standard deviation of log-likelihood obtained with rewards R_1 and R_2 for DataIt010

Data	R_1 **Std. Dev. (Train)**	R_2 **Std. Dev. (Train)**	R_1 **Std. Dev. (Validation)**	R_2 **Std. Dev. (Validation)**
DataIt010	3.9257	4.3720	7.6743	7.7043

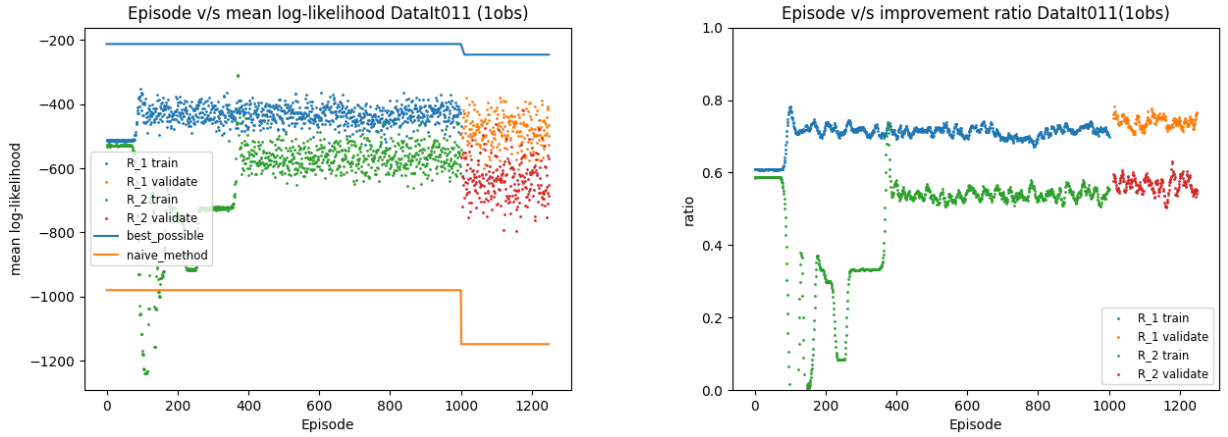
Considering these results, it can be concluded that there are no significant differences between rewards R_1 and R_2 in terms of log-likelihood, improvement ratios, and standard deviations for the DataIt010 dataset. Both rewards yield similar clustering performance and stability, indicating comparable effectiveness in capturing the underlying data patterns.

Therefore, based on the overall analysis and the observed performance metrics, it can be concluded that both R_1 and R_2 are viable options for the DataIt010 dataset. There is no clear superiority of one reward over the other, suggesting that either reward can be chosen without a substantial impact on the clustering outcomes.

DataIt011

The results for the "DataIt011" dataset are presented in Figure 5.10a and Figure 5.10b, along with tables showing the log-likelihood, improvement ratios, and standard deviations for rewards R_1 and R_2.

Figure 5.10a displays the log-likelihood values obtained during training and validation for rewards R_1 and R_2. From the figures, it can be observed that R_1 achieves higher log-likelihood values than R_2 during training, while R_2 performs better during validation, indicating a better generalization ability.



(a) (b)

Figure 5.10: Log-likelihood and improvement ratio for DataIt011

Table 5.28 summarizes the log-likelihood values for the "DataIt011" dataset. The mean log-likelihood values for R_1 are -433.50 (train) and -480.33 (validation), while the mean log-likelihood values for R_2 are -565.27 (train) and -636.13 (validation). These results suggest that R_2 achieves lower log-likelihood values, indicating a better fit to the "DataIt011" data.

Table 5.28: Log-likelihood obtained with rewards R_1 and R_2 for DataIt011

Data	R_1 **Mean (Train)**	R_2 **Mean (Train)**	R_1 **Mean (Validation)**	R_2 **Mean (Validation)**
DataIt011	-433.50	-565.27	-480.33	-636.13

Table 5.29 presents the improvement ratios, representing the relative performance enhancement of rewards R_1 and R_2 compared to the baseline. For the "DataIt011" dataset, R_1 demonstrates higher improvement ratios during training, while R_2 achieves slightly higher improvement ratios during validation. These results indicate that both rewards offer an improvement over the baseline, with R_1 showing better performance during training and R_2 performing better during validation.

Table 5.29: Improvement ratio of log-likelihood obtained with rewards R_1 and R_2 for DataIt011

Data	R_1 **Ratio (Train)**	R_2 **Ratio (Train)**	R_1 **Ratio (Validation)**	R_2 **Ratio (Validation)**
DataIt011	0.71	0.54	0.73	0.56

Table 5.30 shows the standard deviations of log-likelihood, which indicate the stability and consistency of the clustering results. In this case, R_2 exhibits higher standard deviations compared to R_1 for both the training and validation sets, suggesting more variability in its clustering performance.

Table 5.30: Standard deviation of log-likelihood obtained with rewards R_1 and R_2 for DataIt011

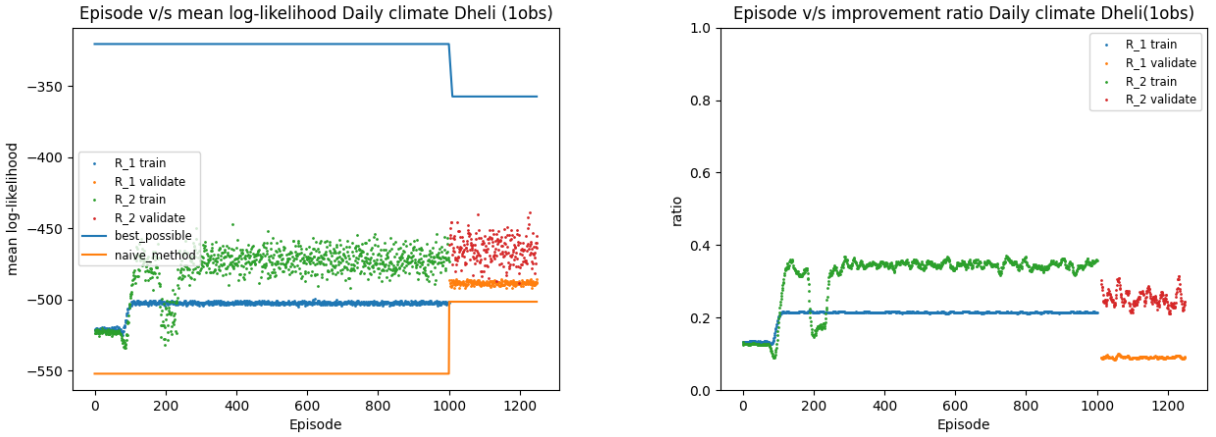
Data	R_1 **Std. Dev. (Train)**	R_2 **Std. Dev. (Train)**	R_1 **Std. Dev. (Validation)**	R_2 **Std. Dev. (Validation)**
DataIt011	27.0546	36.01	46.8850	61.6767

In conclusion, based on the provided results, it can be inferred that R_1 outperforms R_2 for the "DataIt011" dataset. R_1 achieves higher log-likelihood values during training and demonstrates better generalization performance with lower log-likelihood values during validation. The improvement ratios and standard deviations also support the superior performance of R_1. These findings suggest that R_1 captures better the underlying patterns of the "DataIt011" data.

Daily climate Dheli

The results for the "Daily climate Dheli" dataset are shown in Figure 5.11a and Figure 5.11b, along with tables presenting the log-likelihood, improvement ratios, and standard deviations for rewards R_1 and R_2.

Figure 5.11a displays the log-likelihood values obtained during training and validation for rewards R_1 and R_2. From the figures, it can be observed that R_2 consistently outperforms R_1 in terms of log-likelihood, indicating a better fit to the "Daily climate Dheli" data.



(a) (b)

Figure 5.11: Log-likelihood and improvement ratio for Daily climate Dheli

Table 5.31 summarizes the log-likelihood values for the "Daily climate Dheli" dataset. The mean log-likelihood values for R_1 are -502.32 (train) and -488.44 (validation), while the mean log-likelihood values for R_2 are -472.09 (train) and -545.84 (validation). These results demonstrate that R_2 achieves higher log-likelihood values than R_1, suggesting better clustering performance.

Table 5.31: Log-likelihood obtained with rewards R_1 and R_2 for Daily climate Dheli

Data	R_1 **Mean (Train)**	R_2 **Mean (Train)**	R_1 **Mean (Validation)**	R_2 **Mean (Validation)**
Daily climate Dheli	-502.32	-472.09	-488.44	-545.84

Table 5.32 presents the improvement ratios, which represent the relative performance enhancement of rewards R_1 and R_2 compared to the baseline. For the "Daily climate Dheli" dataset, R_2 consistently outperforms R_1 in terms of improvement ratios. The improvement ratios of R_2 are significantly higher than those of R_1, indicating a more substantial enhancement in clustering quality.

Table 5.32: Improvement ratio of log-likelihood obtained with rewards R_1 and R_2 for Daily climate Dheli

Data	R_1 **Ratio (Train)**	R_2 **Ratio (Train)**	R_1 **Ratio (Validation)**	R_2 **Ratio (Validation)**
Daily climate Dheli	0.2147	0.3453	0.0895	0.2541

Table 5.33 shows the standard deviations of log-likelihood, which indicate the stability and consistency of the clustering results. In this case, R_2 exhibits higher standard deviations compared to R_1 for both the training and validation sets, suggesting more variability in its clustering performance.

Table 5.33: Standard deviation of log-likelihood obtained with rewards R_1 and R_2 for Daily climate Dheli

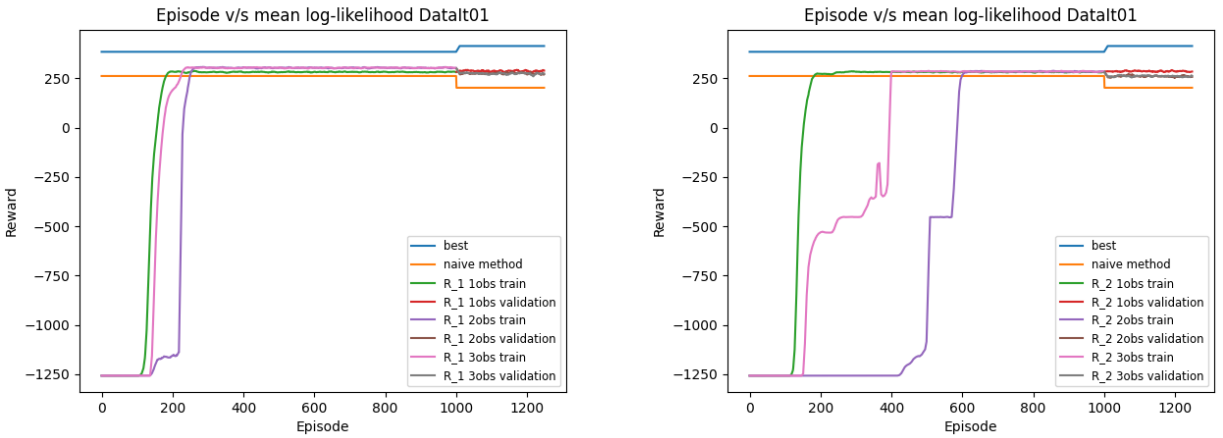
Data	R_1 **Std. Dev. (Train)**	R_2 **Std. Dev. (Train)**	R_1 **Std. Dev. (Validation)**	R_2 **Std. Dev. (Validation)**
Daily climate Dheli	0.7561	6.9854	1.5519	8.8730

In conclusion, based on the provided results, it can be inferred that R_2 is better than R_1 for the "Daily climate Dheli" dataset. R_2 consistently achieves higher log-likelihood values, demonstrates significantly better improvement ratios, and exhibits higher standard deviations. These findings suggest that R_2 captures the underlying patterns of the "Daily climate Dheli" data more effectively and produces more accurate and potentially diverse clustering results.

B Experiment 2 results: Varying the Number of Observations in the Past

DataIt01

The results of Experiment 2, which focused on varying the number of observations in the past, are presented below for the *DataIt01* dataset.



(a) (b)

Figure 5.12: Learning curves of the method with different numbers of observations. Figure (a) shows the method within the reward function R_1, and Figure (b) shows the version within R_2.

Figure 5.12a shows the learning curves of the proposed method with different numbers of observations within two different reward functions, R_1 and R_2. Figure 1(a) depicts the learning curve within the reward function R_1, while Figure 1(b) displays the version within R_2.

number of periods	R_1 log_likelihood train	R_1 log_likelihood validation	R_2 log_likelihood train	R_2 log_likelihood validation
1	283.24	287.97	282.33	286.66
2	304.10	275.76	284.73	261.43
3	305.16	274.85	285.13	261.66

Table 5.34: Loglikelihood achieved by the model when handling DataIt01 with different numbers of observations.

Table 5.34 presents the log-likelihood values achieved by the model when handling *DataIt01* with different numbers of observations. The table includes the number of periods considered (*number of periods*) and the corresponding log-likelihood scores for the training and validation sets within reward functions R_1 and R_2.

Table 5.35 provides the improvement ratios obtained by the model when handling *DataIt01* with different numbers of periods. The improvement ratios are calculated by comparing the log-likelihood scores of the modified model with the baseline model (naive approach) for both the training and validation sets within reward functions R_1 and R_2.

number of periods	R_1 ratio train	R_1 ratio validation	R_2 ratio train	R_2 ratio validation
1	0.1708	0.4018	0.1613	0.3959
2	0.3391	0.3478	0.1818	0.2791
3	0.3483	0.3434	0.1836	0.2797

Table 5.35: Improvement ratios achieved by the model when handling DataIt01 with different numbers of periods.

The results obtained for *DataIt01* are as follows:

For reward function R_1, the log-likelihood scores for the training set increase as the number of periods increases. The model achieves a log-likelihood score of 283.24 with 1 period, 304.10 with 2 periods, and 305.16 with 3 periods. However, on the validation set, the log-likelihood score is highest with 1 period, reaching 287.97, while it decreases with 2 and 3 periods.

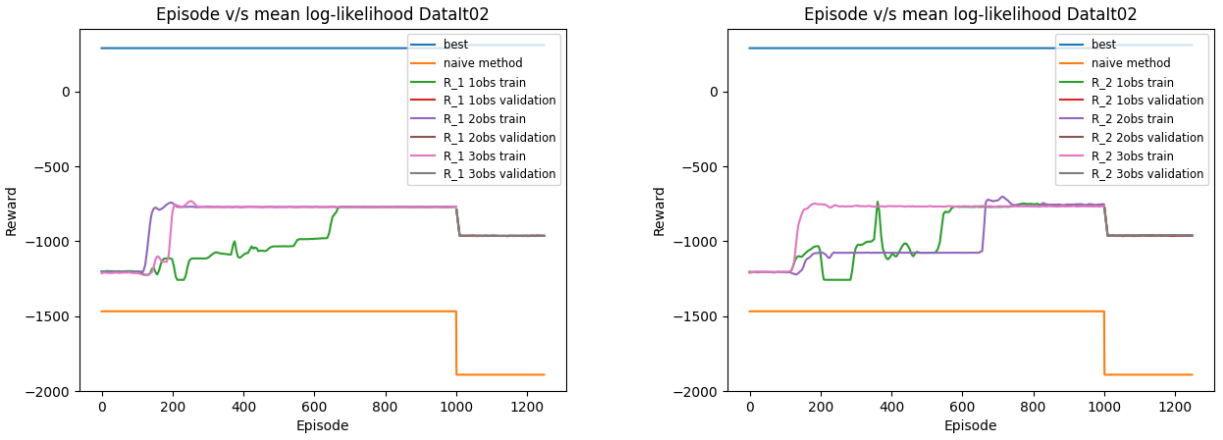
In terms of improvement ratios, for the training set, the model shows an improvement of 17.08% with 1 period, 33.91% with 2 periods, and 34.83% with 3 periods compared to the baseline. Similarly, for the validation set, the improvement ratios are 40.18% with 1 period, 34.78% with 2 periods, and 34.34% with 3 periods.

Moving to reward function R_2, the log-likelihood scores follow a similar pattern. The model achieves the highest log-likelihood score on the training set with 3 periods (285.13), followed by 2 periods (284.73), and 1 period (282.33). However, for the validation set, the log-likelihood score is highest with 1 period (286.66) and decreases with 2 and 3 periods.

Regarding the improvement ratios for reward function R_2, the model shows an improvement of 16.13% with 1 period, 18.18% with 2 periods, and 18.36% with 3 periods on the training set compared to the baseline. Similarly, on the validation set, the improvement ratios are 39.59% with 1 period, 27.91% with 2 periods, and 27.97% with 3 periods.

DataIt02

The results of Experiment 2, which focused on varying the number of observations in the past, are presented below for the *DataIt02* dataset.



(a) (b)

Figure 5.13: Learning curves of the method with different numbers of observations. Figure (a) shows the method within the reward function R_1, and Figure (b) shows the version within R_2.

Figure 5.13a shows the learning curves of the proposed method with different numbers of observations within two different reward functions, R_1 and R_2. Figure 1(a) depicts the learning curve within the reward function R_1, while Figure 1(b) displays the version within R_2.

The log-likelihood scores for different numbers of periods (n_obs) and reward functions R_1 and R_2 are presented in Table 5.36. The improvement ratios are shown in Table 5.37.

Table 5.36: Loglikelihood achieved by the model when handling DataIt02 with different numbers of observations.

n_obs	R_1 loglikelihood train	R_1 loglikelihood validation	R_2 loglikelihood train	R_2 loglikelihood validation
1	-770.32	-962.81	-756.74	-962.81
2	-769.59	-963.15	-753.88	-960.56
3	-769.94	-962.90	-766.42	-959.95

Table 5.37: Improvement ratios achieved by the model when handling DataIt02 with different numbers of observations.

n_obs	R_1 improvement ratio train	R_1 improvement ratio validation	R_2 improvement ratio train	R_2 improvement ratio validation
1	0.397	0.423	0.404	0.423
2	0.397	0.423	0.406	0.424
3	0.397	0.423	0.399	0.424

Now, let's analyze the results obtained for **DataIt02**:

Regarding the log-likelihood scores presented in Table 5.36, there is no clear pattern observed for reward function R_1 as the number of observations (n_obs) increases since there are not a tendency.

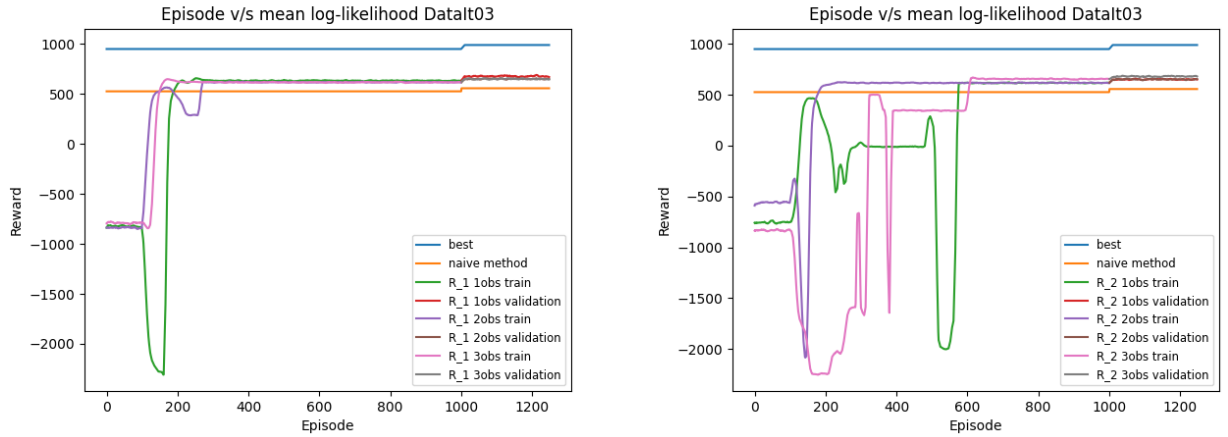
Moving to reward function R_2, the log-likelihood scores also show a similar pattern. The model achieves the highest log-likelihood score on the training set with 2 observations, followed by 1 observation, and finally 3 observations. However, on the validation set, the log-likelihood score is highest with 1 observation and decreases with 2 and 3 observations.

Analyzing the improvement ratios presented in Table 5.37, we can observe that the model consistently demonstrates improvement across different numbers of observations for both rewards functions R_1 and R_2. The improvement ratio for R_1 is 39.7% for the training set in all versions and 42.3% for the validation set (yes, these results are correct). On the other hand, the improvement ratio for R_2 touch the maximum in the 1 observation version.

Overall, the results indicate that the proposed method achieves improved log-likelihood scores and substantial improvement ratios compared to the baseline (naive approach) for both rewards functions R_1 and R_2 on the **DataIt02** dataset.

DataIt03

Figure 5.14 illustrates the learning curves of the proposed method with different numbers of observations within two different reward functions, R_1 and R_2, for the *DataIt03* dataset. Figure 5.14a represents the learning curve within the reward function R_1, while Figure 5.14b illustrates the version within R_2.



(a) (b)

Figure 5.14: Learning curves of the method with different numbers of observations. Figure (a) shows the method within the reward function R_1, and Figure (b) shows the version within R_2.

Table 5.38 presents the log-likelihood values achieved by the model when handling the *DataIt03* dataset with different numbers of observations. It includes the number of observations considered (*n_obs*) and the corresponding log-likelihood scores for the training and validation sets within reward functions R_1 and R_2.

n_obs	R_1 log_likelihood train	R_1 log_likelihood validation	R_2 log_likelihood train	R_2 log_likelihood validation
1	632.28	677.28	616.28	651.65
2	616.29	651.04	616.95	651.74
3	617.18	649.93	656.06	681.06

Table 5.38: Loglikelihood achieved by the model when handling DataIt03 with different numbers of observations.

Table 5.39 provides the improvement ratios obtained by the model when handling the *DataIt03* dataset with different numbers of observations. The improvement ratios are calculated by comparing the log-likelihood scores of the modified model with the baseline model (naive approach) for both the training and validation sets within reward functions R_1 and R_2.

Analyzing the results for *DataIt03*, we observe the following:

For reward function R_1, the log-likelihood scores for both the training and validation sets decrease as the number of observations (*n_obs*) increases. The model achieves a log-likelihood score of 632.28 on the training set and 677.28 on the validation set with 1 observation. With 2 observations, the scores decrease to 616.29 on the training set and 651.04 on the validation

n_obs	R_1 ratio train	R_1 ratio validation	R_2 ratio train	R_2 ratio validation
1	0.250	0.277	0.213	0.218
2	0.213	0.217	0.216	0.218
3	0.215	0.214	0.307	0.286

Table 5.39: Improvement ratios achieved by the model when handling DataIt03 with different numbers of observations.

set. Finally, with 3 observations, the scores further decrease to 617.18 on the training set and 649.93 on the validation set.

Regarding the improvement ratios, for the training set, the model shows an improvement ratio of 25.0% with 1 observation, 21.3% with 2 observations, and 21.5% with 3 observations compared to the baseline model. On the validation set, the improvement ratios range from 21.4% to 27.7% for 1 to 3 observations, indicating the effectiveness of the proposed method within reward function R_1.

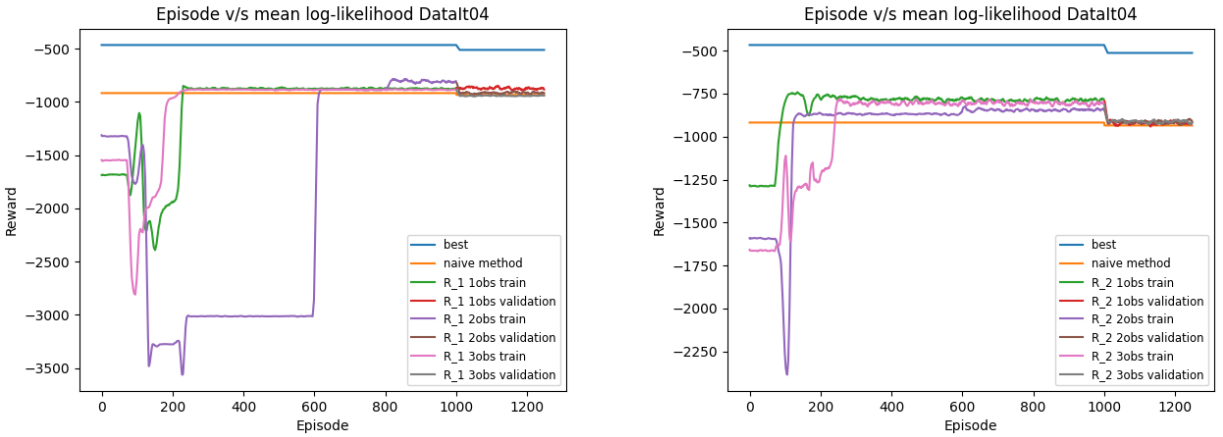
Moving to reward function R_2, the log-likelihood scores follow a similar pattern as in R_1. The model achieves the highest log-likelihood score on the training set with 3 observations (656.06), followed by 2 observation (616.95), and 1 observations (616.28). However, on the validation set, the log-likelihood score is highest with 3 observation (681.06) and decreases with 1 and 3 observations.

Regarding the improvement ratios for reward function R_2, the model shows different patterns compared to R_1. The improvement ratios on the training set range from 21.3% to 30.7% for 1 to 3 observations. On the validation set, the improvement ratio remains between 21.8% to 28.6% for 1 to 3 observations, touching its maximum with 3 observations.

Overall, the results indicate that the number of observations can play an important role in the performance of our method.

DataIt04

Figure 5.15 illustrates the learning curves of the proposed method with different numbers of observations within two different reward functions, R_1 and R_2, for the *DataIt04* dataset. Figure 5.15a represents the learning curve within the reward function R_1, while Figure 5.15b illustrates the version within R_2.



(a) (b)

Figure 5.15: Learning curves of the method with different numbers of observations. Figure (a) shows the method within the reward function R_1, and Figure (b) shows the version within R_2.

Table 5.40 presents the log-likelihood values achieved by the model when handling the *DataIt04* dataset with different numbers of observations. It includes the number of periods considered (*n_obs*) and the corresponding log-likelihood scores for the training and validation sets within reward functions R_1 and R_2.

n_obs	R_1 log-likelihood train	R_1 log-likelihood validation	R_2 log-likelihood train	R_2 log-likelihood validation
1	-877.24	-873.93	-784.70	-920.73
2	-810.69	-916.78	-843.20	-915.88
3	-887.48	-943.63	-810.16	-912.66

Table 5.40: Log-likelihood achieved by the model when handling DataIt04 with different numbers of observations.

Table 5.41 provides the improvement ratios obtained by the model when handling the *DataIt04* dataset with different numbers of observations. The improvement ratios are calculated by comparing the log-likelihood scores of the modified model with the baseline model (naive approach) for both the training and validation sets within reward functions R_1 and R_2.

In summary, the results obtained from the *DataIt04* dataset indicate the following:

For reward function R_1, the log-likelihood scores generally decrease as the number of periods (*n_obs*) increases. With 1 period, the model achieves log-likelihood scores of -877.24 on the training set and -873.93 on the validation set. As the number of periods increases to 2 and 3, the log-likelihood scores decrease further.

n_obs	R_1 improvement ratio train	R_1 improvement ratio validation	R_2 improvement ratio train	R_2 improvement ratio validation
1	0.089	0.146	0.294	0.040
2	0.235	0.048	0.166	0.048
3	0.069	-0.018	0.240	0.058

Table 5.41: Improvement ratios achieved by the model when handling DataIt04 with different numbers of observations.

Analyzing the improvement ratios, on the training set, the model consistently shows positive improvement ratios for all numbers of periods, indicating an improvement over the baseline model. However, on the validation set, the improvement ratios vary. While improvement ratios are positive for 1 and 2 periods, the improvement ratio becomes negative for 3 periods, indicating a worse performance compared to the baseline model.

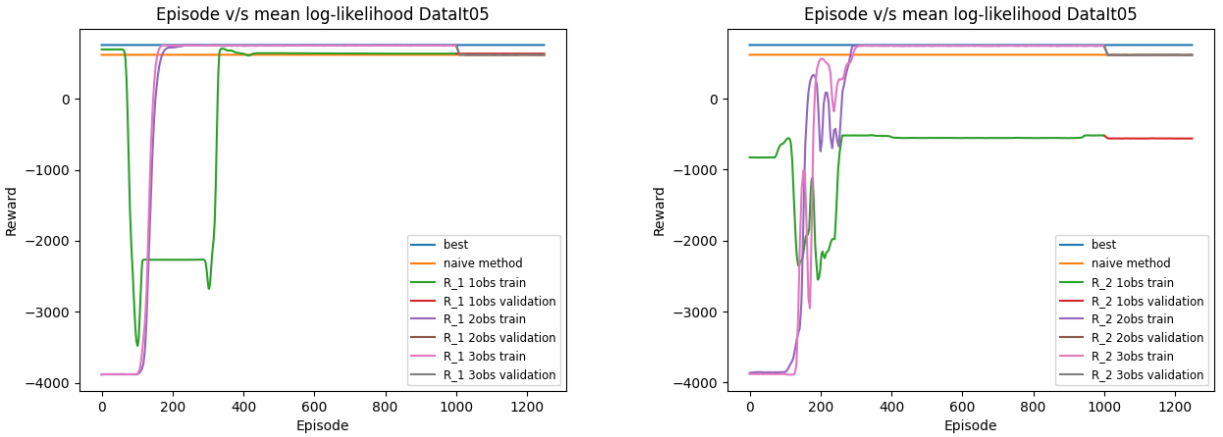
For reward function R_2, the log-likelihood scores exhibit a similar trend. The model achieves the highest log-likelihood scores on the training set with 1 period, while the scores decrease with 2 and 3 periods. On the validation set, the log-likelihood scores consistently decrease as the number of periods increases.

Regarding the improvement ratios, for both the training and validation sets, the improvement ratios fluctuate across different numbers of periods. While positive improvement ratios are observed for some cases, indicating an improvement over the baseline model, negative improvement ratios are also present, indicating worse performance for certain configurations.

These findings highlight the complex relationship between the number of periods and the model's performance within different reward functions for the *DataIt04* dataset. The impact of the number of periods may vary depending on the specific reward function used.

DataIt05

Figure 5.16 illustrates the learning curves of the proposed method with different numbers of observations within two different reward functions, R_1 and R_2, for the 05 dataset. Figure 5.16a represents the learning curve within the reward function R_1, while Figure 5.16b illustrates the version within R_2.



(a) (b)

Figure 5.16: Learning curves of the method with different numbers of observations for the 05 dataset. Figure (a) shows the method within the reward function R_1, and Figure (b) shows the version within R_2.

Table 5.42 presents the log-likelihood values achieved by the model when handling the *DataIt05* dataset with different numbers of periods. It includes the number of periods considered (*n_obs*) and the corresponding log-likelihood scores for the training and validation sets within reward functions R_1 and R_2.

n_obs	R_1 log-likelihood train	R_1 log-likelihood validation	R_2 log-likelihood train	R_2 log-likelihood validation
1	634.72	724.91	634.96	725.53
2	635.02	725.25	636.04	726.63
3	748.49	617.79	746.60	615.49

Table 5.42: Log-likelihood achieved by the model when handling DataIt05 with different numbers of periods.

Table 5.43 provides the improvement ratios obtained by the model when handling the *DataIt05* dataset with different numbers of periods. The improvement ratios are calculated by comparing the log-likelihood scores of the modified model with the baseline model (naive approach) for both the training and validation sets within reward functions R_1 and R_2.

In summary, the results obtained from the *DataIt05* dataset indicate the following:

For reward function R_1, the log-likelihood scores generally increase or remain relatively stable as the number of periods (*n_obs*) increases. The model achieves log-likelihood scores ranging from 634.72 to 748.49 on the training set and from 617.79 to 724.91 on the validation set.

n_obs	R_1 improvement ratio train	R_1 improvement ratio validation	R_2 improvement ratio train	R_2 improvement ratio validation
1	0.118	0.099	0.120	0.103
2	0.120	0.102	0.127	0.110
3	0.516	0.499	0.501	0.482

Table 5.43: Improvement ratios achieved by the model when handling DataIt05 with different numbers of periods.

Analyzing the improvement ratios, on both the training and validation sets, the model consistently shows positive improvement ratios for all numbers of periods within reward function R_1, indicating an improvement over the baseline model. The improvement ratios range from 0.118 to 0.516 on the training set and from 0.099 to 0.499 on the validation set. The better is the 3 observation version.

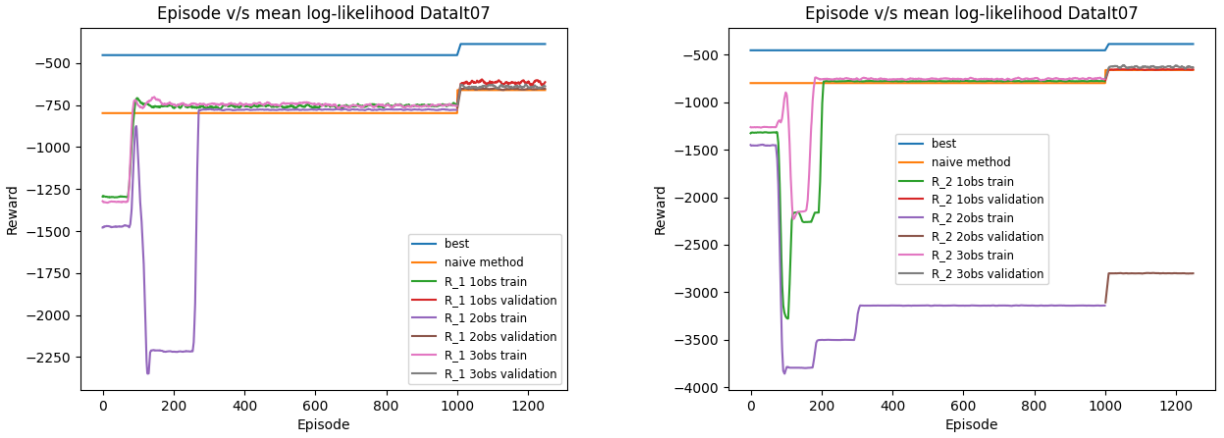
For reward function R_2, the log-likelihood scores also exhibit a similar trend. The model achieves log-likelihood scores ranging from 634.96 to 746.60 on the training set and from 615.49 to 725.53 on the validation set as the number of periods increases.

Regarding the improvement ratios, positive improvement ratios are observed for all numbers of periods on both the training and validation sets within reward function R_2, indicating an improvement over the baseline model. The improvement ratios range from 0.103 to 0.482 on the validation set and from 0.120 to 0.501 on the training set. The better is the 3 observation version.

These findings indicate that increasing the number of periods could lead to improved performance in terms of log-likelihood scores and improvement ratios for both rewards functions R_1 and R_2 in the *DataIt05* dataset.

DataIt07

Figure 5.17 illustrates the learning curves of the proposed method with different numbers of observations within two different reward functions, R_1 and R_2, for the DataIt07 dataset. Figure 5.17a represents the learning curve within the reward function R_1, while Figure 5.17b illustrates the version within R_2.



(a) (b)

Figure 5.17: Learning curves of the method with different numbers of observations for the DataIt07 dataset. Figure (a) shows the method within the reward function R_1, and Figure (b) shows the version within R_2.

Table 5.44 presents the log-likelihood values achieved by the model when handling the *DataIt07* dataset with different numbers of periods. It includes the number of periods considered (*n_obs*) and the corresponding log-likelihood scores for the training and validation sets within reward functions R_1 and R_2.

n_obs	R_1 log-likelihood train	R_1 log-likelihood validation	R_2 log-likelihood train	R_2 log-likelihood validation
1	-745.57	-615.23	-775.79	-654.24
2	-777.07	-653.33	-3141.19	-2801.29
3	-750.50	-639.58	-749.33	-627.37

Table 5.44: Log-likelihood achieved by the model when handling DataIt07 with different numbers of periods.

Table 5.45 provides the improvement ratios obtained by the model when handling the *DataIt07* dataset with different numbers of periods. The improvement ratios are calculated by comparing the log-likelihood scores of the modified model with the baseline model (naive approach) for both the training and validation sets within reward functions R_1 and R_2.

In summary, the results obtained from the *DataIt07* dataset indicate the following:

For reward function R_1, the log-likelihood scores show some variation as the number of periods (*n_obs*) increases. The model achieves log-likelihood scores ranging from -745.57 to -777.07 on the training set and from -615.23 to -653.33 on the validation set.

n_obs	R_1 improvement ratio train	R_1 improvement ratio validation	R_2 improvement ratio train	R_2 improvement ratio validation
1	0.145	0.154	0.062	0.011
2	0.059	0.015	-6.784	-7.873
3	0.137	0.066	0.139	0.110

Table 5.45: Improvement ratios achieved by the model when handling DataIt07 with different numbers of periods.

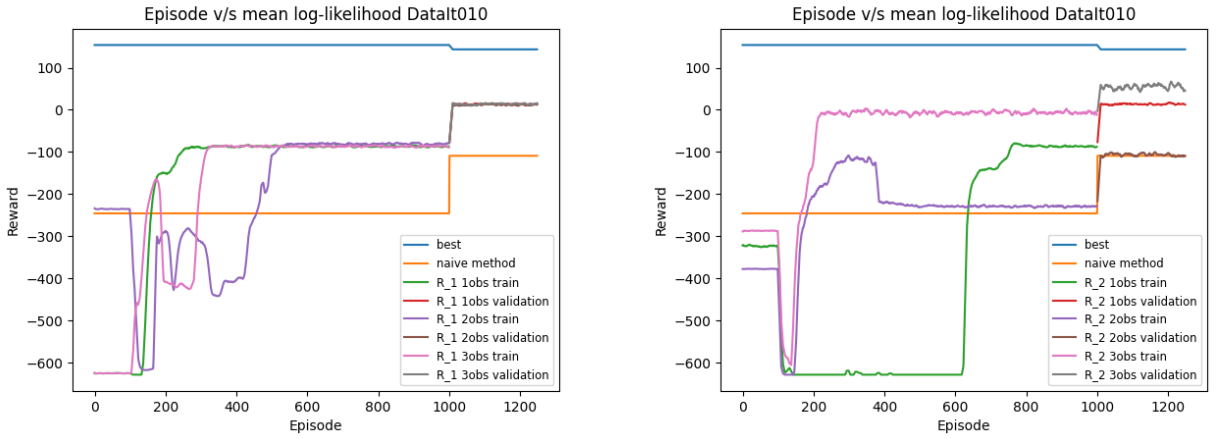
Analyzing the improvement ratios, on both the training and validation sets, the model shows mixed improvement ratios for different numbers of periods within reward function R_1. While positive improvement ratios are observed, indicating an improvement over the baseline model, the improvement ratios range from 0.059 to 0.145 on the training set and from 0.015 to 0.154 on the validation set.

For reward function R_2, the log-likelihood scores exhibit a similar trend. The model achieves log-likelihood scores ranging from -775.79 to -3141.19 on the training set and from -654.24 to -2801.29 on the validation set as the number of periods increases.

Regarding the improvement ratios, negative improvement ratios are observed for the model's performance within reward function R_2 on both the training and validation sets. The improvement ratios range from -6.784 to 0.139 on the training set and from -7.873 to 0.110 on the validation set.

These findings demonstrate the impact of the number of periods on the model's performance within different reward functions for the *DataIt07* dataset.

Figure 5.18 illustrates the learning curves of the proposed method with different numbers of observations within two different reward functions, R_1 and R_2, for the DataIt010 dataset. Figure 5.18a represents the learning curve within the reward function R_1, while Figure 5.18b illustrates the version within R_2.



(a) (b)

Figure 5.18: Learning curves of the method with different numbers of observations for the DataIt010 dataset. Figure (a) shows the method within the reward function R_1, and Figure (b) shows the version within R_2.

Table 5.46 presents the log-likelihood values achieved by the model when handling the DataIt010 dataset with different numbers of periods. It includes the number of periods considered (n_obs) and the corresponding log-likelihood scores for the training and validation sets within reward functions R_1 and R_2.

n_obs	R_1 log-likelihood train	R_1 log-likelihood validation	R_2 log-likelihood train	R_2 log-likelihood validation
1	-87.03	13.37	-87.04	13.48
2	-80.55	12.15	-229.31	-106.63
3	-86.46	13.79	-7.66	54.78

Table 5.46: Log-likelihood achieved by the model when handling DataIt010 with different numbers of periods.

Table 5.47 provides the improvement ratios obtained by the model when handling the DataIt010 dataset with different numbers of periods. The improvement ratios are calculated by comparing the log-likelihood scores of the modified model with the baseline model (naive approach) for both the training and validation sets within reward functions R_1 and R_2.

n_obs	R_1 improvement ratio train	R_1 improvement ratio validation	R_2 improvement ratio train	R_2 improvement ratio validation
1	0.397	0.477	0.397	0.478
2	0.413	0.473	0.041	0.001
3	0.399	0.479	0.598	0.644

Table 5.47: Improvement ratios achieved by the model when handling DataIt010 with different numbers of periods.

In summary, the results obtained from the DataIt010 dataset indicate the following:

For reward function R_1, the log-likelihood scores show some variation as the number of periods (n_obs) increases but is not possible to see a tendency,

Analyzing the improvement ratios, on both the training and validation sets, the model shows positive improvement ratios for different numbers of periods within reward function R_1.

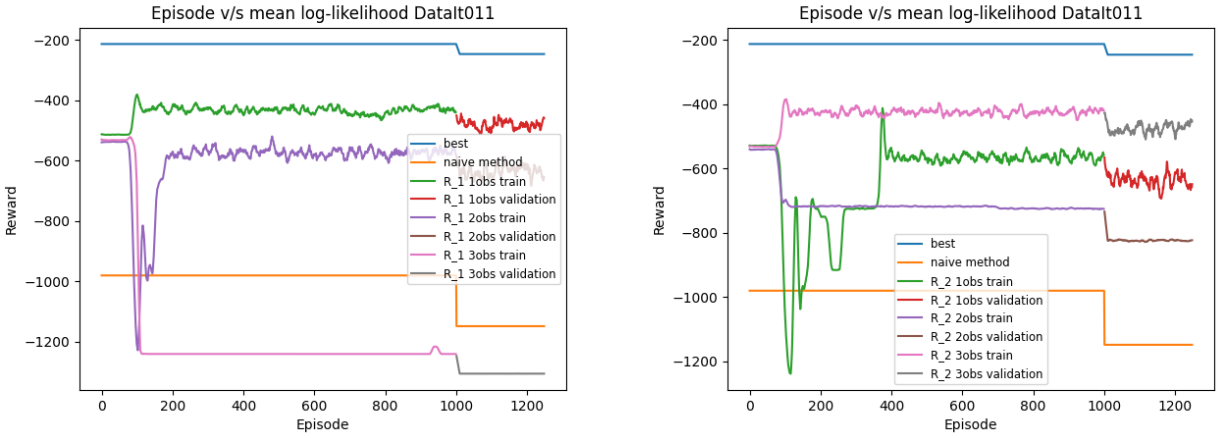
For reward function R_2, the log-likelihood scores exhibit a larger variation where the worse was the model with 2 observations and the better the case with 3 observations.

Regarding the improvement ratios, positive improvement ratios are observed for the model's performance within reward function R_2 on the training set for 1 and 3 periods. However, the improvement ratios were close to zero for the validation set with 2 observations, indicating limited improvement over the baseline model, but a high improvement in the other configurations.

These findings demonstrate the impact of the number of periods on the model's performance within different reward functions for the DataIt010 dataset.

DataIt011

Figure 5.19 illustrates the learning curves of the proposed method with different numbers of observations within two different reward functions, R_1 and R_2, for the DataIt011 dataset. Figure 5.19a represents the learning curve within the reward function R_1, while Figure 5.19b illustrates the version within R_2.



(a) (b)

Figure 5.19: Learning curves of the method with different numbers of observations for the DataIt011 dataset. Figure (a) shows the method within the reward function R_1, and Figure (b) shows the version within R_2.

Table 5.48 presents the log-likelihood values achieved by the model when handling the *DataIt011* dataset with different numbers of periods. It includes the number of periods considered (*n_obs*) and the corresponding log-likelihood scores for the training and validation sets within reward functions R_1 and R_2.

n_obs	R_1 log-likelihood train	R_1 log-likelihood validation	R_2 log-likelihood train	R_2 log-likelihood validation
1	-433.29	-480.39	-567.94	-636.51
2	-569.39	-635.60	-724.99	-823.79
3	-1240.05	-1305.35	-417.46	-475.22

Table 5.48: Log-likelihood achieved by the model when handling DataIt011 with different numbers of periods.

Table 5.49 provides the improvement ratios obtained by the model when handling the *DataIt011* dataset with different numbers of periods. The improvement ratios are calculated by comparing the log-likelihood scores of the modified model with the baseline model (naive approach) for both the training and validation sets within reward functions R_1 and R_2.

In summary, the results obtained from the *DataIt011* dataset indicate the following:

For reward function R_1, the log-likelihood scores show variations as the number of periods (*n_obs*) increases. The model achieves log-likelihood scores ranging from -1240 to -433.29 on the training set and from -1305.35 to -480.39 on the validation set.

n_obs	R_1 improvement ratio train	R_1 improvement ratio validation	R_2 improvement ratio train	R_2 improvement ratio validation
1	0.716	0.740	0.537	0.568
2	0.537	0.569	0.332	0.361
3	-0.338	-0.173	0.732	0.746

Table 5.49: Improvement ratios achieved by the model when handling DataIt011 with different numbers of periods.

Analyzing the improvement ratios, positive improvement ratios are observed for different numbers of periods within reward function R_1 on both the training and validation sets. The improvement ratios range from -0.338 to 0.716 on the training set and from -0.173to 0.740 on the validation set.

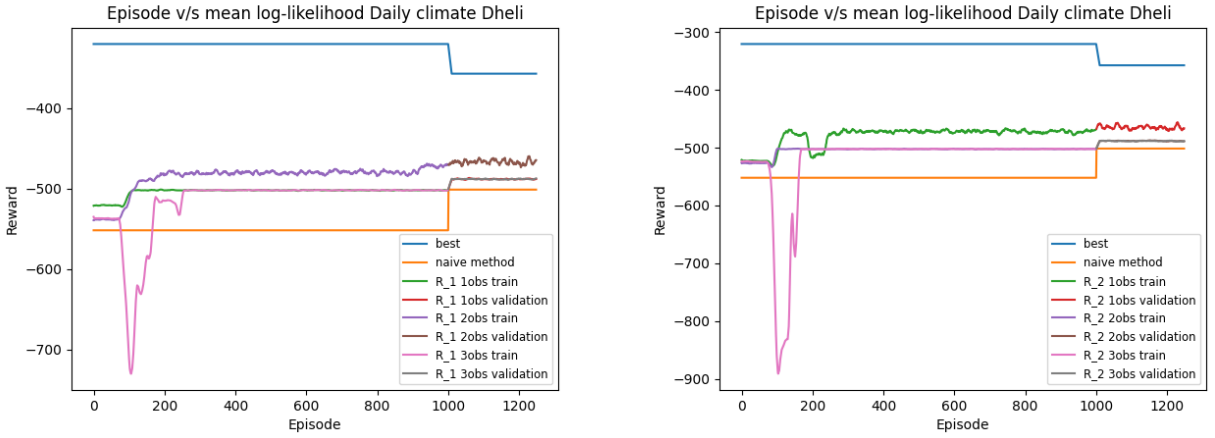
For reward function R_2, the log-likelihood scores also exhibit variations. The model achieves log-likelihood scores ranging from -724.99 to -417.46 on the training set and from -823.51 to -475.22 on the validation set as the number of periods increases.

Regarding the improvement ratios, positive improvement ratios are observed for the model's performance within reward function R_2 on both the training and validation sets for all numbers of periods considered. The improvement ratios range from 0.332 to 0.732 on the training set and from 0.361 to 0.746 on the validation set, indicating improvement over the baseline model.

These findings demonstrate the impact of the number of periods on the model's performance within different reward functions for the *DataIt011* dataset, but is not always better as the number of observation increase.

Daily Climate Delhi

Figure 5.20 illustrates the learning curves of the proposed method with different numbers of observations within two different reward functions, R_1 and R_2, for the Daily climate Delhi dataset. Figure 5.20a represents the learning curve within the reward function R_1, while Figure 5.20b illustrates the version within R_2.



(a) (b)

Figure 5.20: Learning curves of the method with different numbers of observations for the Daily climate Delhi dataset. Figure (a) shows the method within the reward function R_1, and Figure (b) shows the version within R_2.

Table 5.50 presents the log-likelihood values achieved by the model when handling the "Daily Climate Delhi" dataset with different numbers of periods. It includes the number of periods considered (n_obs) and the corresponding log-likelihood scores for the training and validation sets within reward functions R_1 and R_2.

n_obs	R_1 log-likelihood train	R_1 log-likelihood validation	R_2 log-likelihood train	R_2 log-likelihood validation
1	-502.37	-488.39	-470.71	-464.70
2	-472.07	-467.06	-502.18	-488.28
3	-502.41	-488.44	-502.29	-488.45

Table 5.50: Log-likelihood achieved by the model when handling Daily Climate Delhi with different numbers of periods.

Table 5.51 provides the improvement ratios obtained by the model when handling the "Daily Climate Delhi" dataset with different numbers of periods. The improvement ratios are calculated by comparing the log-likelihood scores of the modified model with the baseline model (naive approach) for both the training and validation sets within reward functions R_1 and R_2.

In summary, the results obtained from the "Daily Climate Delhi" dataset indicate the following:

For reward function R_1, the log-likelihood scores show slight variations as the number of periods (n_obs) increases. The model achieves log-likelihood scores ranging from -502.41 to

n_obs	R_1 improvement ratio train	R_1 improvement ratio validation	R_2 improvement ratio train	R_2 improvement ratio validation
1	0.215	0.089	0.349	0.254
2	0.345	0.238	0.216	0.090
3	0.215	0.089	0.215	0.089

Table 5.51: Improvement ratios achieved by the model when handling Daily Climate Delhi with different numbers of periods.

-472.07 on the training set and from -488.44 to -467.06 on the validation set.

Analyzing the improvement ratios, positive improvement ratios are observed for different numbers of periods within reward function R_1 on both the training and validation sets. The improvement ratios range from 0.215 to 0.345 on the training set and from 0.089 to 0.238 on the validation set, indicating improvement over the baseline model. The better version is with 2 observations.

For reward function R_2, the log-likelihood scores also exhibit variations. The model achieves log-likelihood scores ranging from -502.29 to -470.71 on the training set and from -488.45 to -464.70 on the validation set as the number of periods increases.

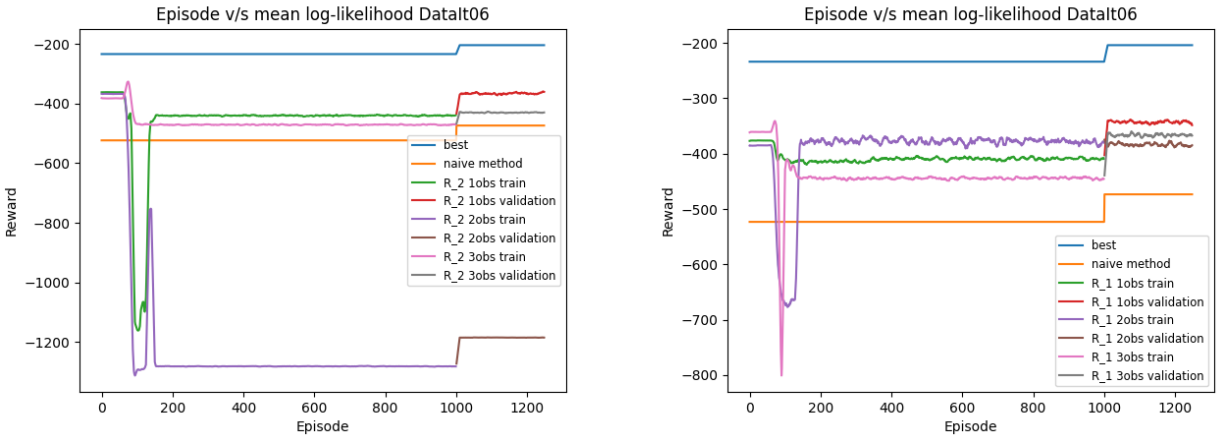
Regarding the improvement ratios, positive improvement ratios are observed for the model's performance within reward function R_2 on both the training and validation sets for all numbers of periods considered. The improvement ratios range from 0.215 to 0.349 on the training set and from 0.254 to 0.090 on the validation set, indicating improvement over the baseline model. The better version is with 1 observation.

These findings demonstrate the impact of the number of periods on the model's performance within different reward functions for the "Daily Climate Delhi" dataset.

C Experiment 3 Results: Creation and Elimination of Clusters

In this experiment, we investigated the ability of the GMM-POMG model to adapt to changes in the number of clusters over time. The model was modified to allow for the dynamic creation and elimination of clusters. We evaluated the model's capability to identify and adapt to changes in cluster formations using the synthetic datasets *DataIt06* and *DataIt09*.

C.1 DataIt06



(a) (b)

Figure 5.21: Learning curves of the method with different numbers of observations for DataIt06. (a) Method within the reward function R_1. (b) Method within the reward function R_2.

We utilized the *DataIt06* dataset, which exhibits the creation of a cluster over time. This allowed us to assess the model's ability to adapt to such changes dynamically.

Table 5.52 presents the loglikelihood achieved by the model when handling *DataIt06* with different numbers of periods. The table includes the loglikelihood values for both the training and validation sets, considering both reward functions R_1 and R_2.

Number of Periods	R_1 Loglikelihood (Train)	R_1 Loglikelihood (Validation)	R_2 Loglikelihood (Train)	R_2 Loglikelihood (Validation)
1	-409.67	-342.95	-440.03	-366.10
2	-380.96	-383.69	-1281.13	-1184.54
3	-445.88	-366.46	-471.00	-429.94

Table 5.52: Loglikelihood achieved by the model when handling DataIt06 with different numbers of periods.

The learning curves for *DataIt06* are depicted in Figure 5.21a and Figure 5.21b. Figure 5.21a shows the learning curve of the method within the reward function R_1, while Figure 5.21b shows the learning curve within the reward function R_2.

Table 5.53 shows the improvement ratios achieved by the model when handling *DataIt06*. These ratios represent the improvement of the modified model over the baseline (parameters at time t), considering both the training and validation sets for reward functions R_1 and R_2.

Number of Periods	R_1 Improvement Ratio (Train)	R_1 Improvement Ratio (Validation)	R_2 Improvement Ratio (Train)	R_2 Improvement Ratio (Validation)
1	0.392	0.481	0.288	0.394
2	0.493	0.335	-2.618	-2.651
3	0.269	0.392	0.180	0.159

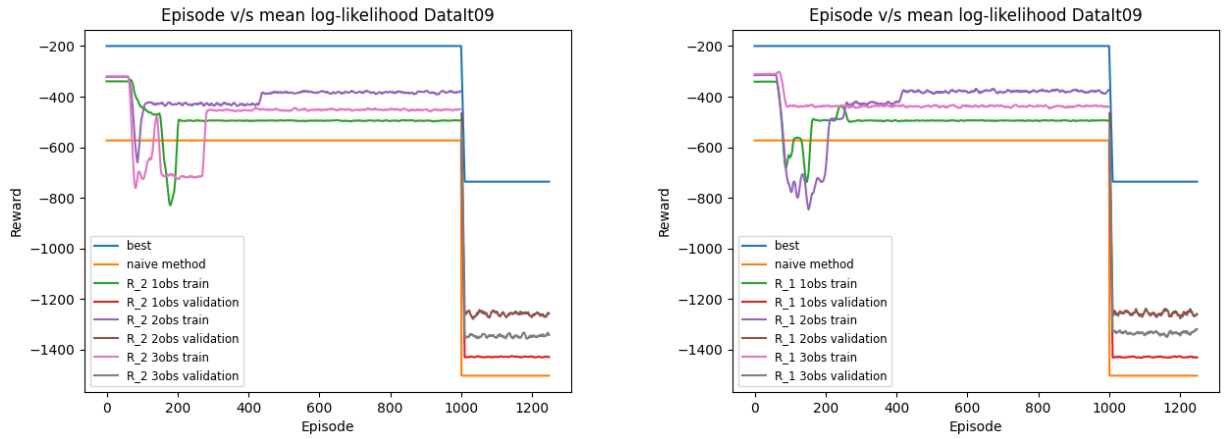
Table 5.53: Improvement ratios achieved by the model when handling DataIt06 with different numbers of periods.

The results indicate that the GMM-POMG model was able to adapt to changes in cluster formations in *DataIt06*, considering the creation of a new cluster over time. The improvement ratios demonstrate the model's ability to achieve better loglikelihood performance compared to the baseline. However, it is worth noting that in the case of *DataIt06*, the improvement ratios for reward function R_2 in the second period were significantly negative. This can be attributed to poor convergence, possibly due to a low number of episodes, as indicated by the low loglikelihood values.

These findings highlight the adaptability of the GMM-POMG model to changes in the number of clusters over time.

C.2 DataIt09

We utilized the *DataIt09* dataset, which involves the elimination of a cluster over time. This allowed us to evaluate the model's ability to adapt to such changes dynamically.



(a) (b)

Figure 5.22: Learning curves of the method with different numbers of observations for DataIt09. (a) Method within the reward function R_1. (b) Method within the reward function R_2.

Table 5.54 presents the loglikelihood achieved by the model when handling *DataIt09* with different numbers of periods. The table includes the loglikelihood values for both the training and validation sets, considering both reward functions R_1 and R_2.

Number of Periods	R_1 Loglikelihood (Train)	R_1 Loglikelihood (Validation)	R_2 Loglikelihood (Train)	R_2 Loglikelihood (Validation)
1	-494.21	-1429.93	-494.27	-1429.24
2	-380.00	-1257.25	-381.58	-1259.29
3	-437.94	-1334.08	-451.33	-1346.74

Table 5.54: Loglikelihood achieved by the model when handling DataIt09 with different numbers of periods.

The learning curves for *DataIt09* are depicted in Figure 5.22a and Figure 5.22b. Figure 5.22a shows the learning curve of the method within the reward function R_1, while Figure 5.22b shows the learning curve within the reward function R_2.

Table 5.55 shows the improvement ratios achieved by the model when handling *DataIt09*. These ratios represent the improvement of the modified model over the baseline (parameters at time t), considering both the training and validation sets for reward functions R_1 and R_2.

Number of Periods	R_1 Improvement Ratio (Train)	R_1 Improvement Ratio (Validation)	R_2 Improvement Ratio (Train)	R_2 Improvement Ratio (Validation)
1	0.211	0.111	0.211	0.111
2	0.516	0.333	0.513	0.330
3	0.363	0.234	0.326	0.218

Table 5.55: Improvement ratios achieved by the model when handling DataIt09 with different numbers of periods.

The results demonstrate that the GMM-POMG model was able to adapt to changes in cluster formations in *DataIt09*, which involved the elimination of a cluster over time. The

improvement ratios indicate the model's ability to achieve better loglikelihood performance compared to the baseline. These findings provide evidence of the GMM-POMG model's adaptability to changes in the number of clusters over time.

Furthermore, it is interesting to note that the improvement ratios for both reward functions R_1 and R_2 consistently showed positive values, indicating a positive impact on loglikelihood performance.

D Results Tables

Data Item	R1		R2	
	Training	Test	Training	Test
DataIt01	283.24	287.97	282.33	286.66
DataIt02	-770.32	-962.81	-756.74	-962.81
DataIt03	632.28	677.28	616.28	651.65
DataIt04	-877.24	-873.93	-784.70	-920.73
DataIt05	634.72	724.91	634.96	725.53
DataIt07	-745.57	-615.23	-775.79	-654.24
DataIt010	-87.03	13.37	-87.04	13.48
DataIt011	-433.29	-480.39	-567.94	-636.51
Dheli	-502.37	-488.39	-470.71	-464.70

Table 5.56: Comparison of Loglikelihood Across Different Reward Function within 1 Observation

Data Item	R1		R2	
	Training	Test	Training	Test
DataIt01	0.1708	0.4018	0.1613	0.3959
DataIt02	0.397	0.423	0.404	0.423
DataIt03	0.250	0.277	0.213	0.218
DataIt04	0.089	0.146	0.294	0.040
DataIt05	0.118	0.099	0.120	0.103
DataIt07	0.145	0.154	0.062	0.011
DataIt010	0.397	0.477	0.397	0.478
DataIt011	0.716	0.740	0.537	0.568
Dheli	0.215	0.089	0.349	0.254

Table 5.57: Comparison of Improvement Ratio Across Different Reward Function within 1 Observation

Data Item	R1		R2	
	Training	Test	Training	Test
DataIt01	304.10	275.76	284.73	261.43
DataIt02	-769.59	-963.15	-753.88	-960.56
DataIt02	616.29	651.04	616.95	651.74
DataIt02	-810.69	-916.78	-843.20	-915.88
DataIt02	635.02	725.25	636.04	726.63
DataIt07	-777.07	-653.33	-3141.19	-2801.29
DataIt010	-80.55	12.15	-229.31	-106.63
DataIt011	-569.39	-635.60	-724.99	-823.79
Dheli	-472.07	-467.06	-502.18	-488.28

Table 5.58: Comparison of Loglikelihood Across Different Reward Function within 2 Observation

Data Item	R1		R2	
	Training	Test	Training	Test
DataIt01	0.3391	0.3478	0.1818	0.2791
DataIt02	0.397	0.423	0.406	0.424
DataIt03	0.213	0.217	0.216	0.218
DataIt04	0.235	0.048	0.166	0.048
DataIt05	0.120	0.102	0.127	0.110
DataIt07	0.059	0.015	-6.784	-7.873
DataIt010	0.413	0.473	0.041	0.001
DataIt011	0.537	0.569	0.332	0.361
Dheli	0.345	0.238	0.216	0.090

Table 5.59: Comparison of Improvement Ratio Across Different Reward Function within 2 Observation

Data Item	R1		R2	
	Training	Test	Training	Test
DataIt01	305.16	274.85	285.13	261.66
DataIt02	-769.94	-962.90	-766.42	-959.95
DataIt03	617.18	649.93	656.06	681.06
DataIt04	-887.48	-943.63	-810.16	-912.66
DataIt05	748.49	617.79	746.60	615.49
DataIt07	-750.50	-639.58	-749.33	-627.37
DataIt010	-86.46	13.79	-7.66	54.78
DataIt011	-1240.05	-1305.35	-417.46	-475.22
Dheli	-502.41	-488.44	-502.29	-488.45

Table 5.60: Comparison of Loglikelihood Across Different Reward Function within 3 Observation

Data Item	R1		R2	
	Training	Test	Training	Test
DataIt01	0.3483	0.3434	0.1836	0.2797
DataIt02	0.397	0.423	0.399	0.424
DataIt03	0.215	0.214	0.307	0.286
DataIt04	0.069	-0.018	0.240	0.058
DataIt05	0.516	0.499	0.501	0.482
DataIt07	0.137	0.066	0.139	0.110
DataIt010	0.399	0.479	0.598	0.644
DataIt011	-0.338	-0.173	0.732	0.746
Dheli	0.215	0.089	0.215	0.089

Table 5.61: Comparison of Improvement Ratio Across Different Reward Function within 3 Observation