CS307/367 Artificial Intelligence : Agent – End-Sem Lab Report

Tausif Ansari
Computer Science and Engineering
Email: 202351148@iiitvadodara.ac.in

Sarthak Daga
Computer Science and Engineering
Email: 202351128@iiitvadodara.ac.in

Swarnim Goyal
Computer Science and Engineering
Email: 202351144@iiitvadodara.ac.in

GitHub Repo Link

***Abstract*—Financial markets transition between distinct volatility regimes that are not directly observable from raw price movements. Gaussian Hidden Markov Models (HMMs) provide a probabilistic framework to infer such hidden market states using observable financial returns and volatility. In this lab, historical stock price data were collected from Yahoo Finance over a 10-year period. Daily log-returns and rolling realized volatility were extracted as features, and multi-state Gaussian HMMs with full covariance were fitted using the expectation–maximization algorithm. Model selection with AIC and BIC over different state counts revealed a five-state model as a good trade-off between fit and complexity. The inferred states correspond to meaningful market regimes, including low-volatility stable periods and high-volatility stressed phases. Visualizations of decoded states, posterior probabilities, and transition matrices demonstrate the ability of HMMs to capture latent financial dynamics, offering practical insights for risk analysis and decision-making.**

I. Modeling Hidden Market Regimes Using Gaussian Hidden Markov Models

Financial time series such as stock prices exhibit nonlinear, non-stationary behavior and often transition between distinct regimes like bullish, bearish, stable, and turbulent phases. These underlying regimes are not directly observable from prices alone. In this experiment, Gaussian Hidden Markov Models (HMMs) are used to infer such *hidden market regimes* from observable features derived from historical stock data.

A. Problem Description

The objective is to model an equity time series using a multi-state Gaussian HMM and to identify latent regimes corresponding to different volatility and return characteristics. The hidden state sequence is assumed to follow a first-order Markov chain, while the observations (returns and volatility) are generated from state-dependent multivariate Gaussian distributions.

This experiment demonstrates:

- how to construct features from raw price data,
- how to fit Gaussian HMMs using the EM (Baum–Welch) algorithm,
- how to select the number of hidden states using information criteria (AIC/BIC),
- and how to interpret the inferred regimes and transition structure.

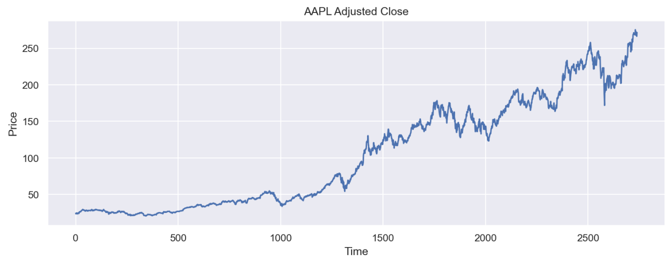

Fig. 1. Adjusted Close prices of AAPL (2015–2025).

B. Data Collection and Preprocessing

1) Data Acquisition: Historical daily price data for the ticker `AAPL` were downloaded from Yahoo Finance using the `yfinance` Python library. The dataset spans approximately 10 years, capturing multiple market cycles. The following fields were collected:

- Date
- Open, High, Low, Close
- Adjusted Close
- Volume

The *Adjusted Close* series was used as the main input for feature construction, as it accounts for stock splits and dividends.

2) Feature Engineering: Two features were derived from the adjusted close prices:

a) Log-returns: Daily log-returns are defined by

$$r_t = \ln(P_t) - \ln(P_{t-1}), \quad (1)$$

where P_t is the adjusted closing price at time t.

b) Rolling Realized Volatility: To capture volatility clustering, a rolling realized volatility feature was constructed as

$$\sigma_t = \sqrt{\frac{1}{n}\sum_{i=0}^{n-1}(r_{t-i} - \bar{r})^2}, \quad (2)$$

with a window size of $n = 10$, and $\bar{r}$ denoting the mean return in the window.

Non-trading days and resulting missing values were removed. Both r_t and σ_t were standardized using `StandardScaler` for numerical stability.

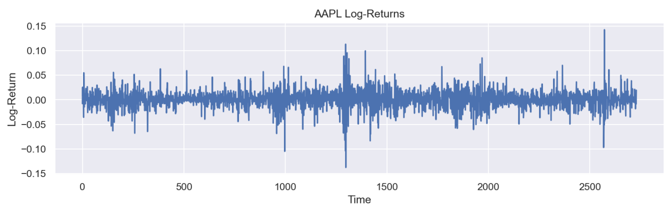

Fig. 2. Daily log-returns of AAPL.

C. Gaussian Hidden Markov Model

1) Model Specification: The observation vector at time t is

$$X_t = \begin{bmatrix} r_t \\ \sigma_t \end{bmatrix}, \tag{3}$$

and the hidden regime is denoted by $z_t \in \{1, \dots, K\}$. The model assumes:

- $\{z_t\}$ forms a first-order Markov chain with transition matrix A,
- conditional independence of observations given the hidden states,
- Gaussian emissions with state-specific parameters.

The emission distribution is

$$X_t \mid z_t = i \sim \mathcal{N}(\boldsymbol{\mu}_i, \Sigma_i), \tag{4}$$

where $\boldsymbol{\mu}_i$ is a 2D mean vector and Σ_i is a full 2×2 covariance matrix for regime i.

The full parameter set is

$$\Theta = \left\{ \boldsymbol{\pi}, A, \{\boldsymbol{\mu}_i, \Sigma_i\}_{i=1}^{K} \right\},$$

where $\boldsymbol{\pi}$ is the initial state distribution.

2) Parameter Estimation via EM: The Baum–Welch algorithm (a special case of EM) is used to estimate Θ:

Algorithm 1 Gaussian HMM Training (Baum–Welch)

```
1: procedure TRAINHMM($X_{1:T}, K$)
2:     Initialize $\boldsymbol{\pi}, A, \{\boldsymbol{\mu}_i, \Sigma_i\}_{i=1}^{K}$ randomly
3:     repeat
4:         E-step: Compute forward–backward probabilities
5:             and posterior responsibilities $\gamma_t(i)$
6:         M-step: Update $\boldsymbol{\pi}, A$ using $\gamma_t(i)$ and $\xi_t(i,j)$
7:             Update $\boldsymbol{\mu}_i, \Sigma_i$ using weighted moments of $X_t$
8:     until convergence of log-likelihood
9:     return $\Theta$
10: end procedure
```

Multiple random initializations and restarts are used to reduce the chance of converging to poor local optima.

D. Model Selection

Gaussian HMMs with $K = 2, 3, 4, 5$ hidden states were fitted on the same feature set. For each K, the maximized log-likelihood $\hat{L}$, Akaike Information Criterion (AIC), and

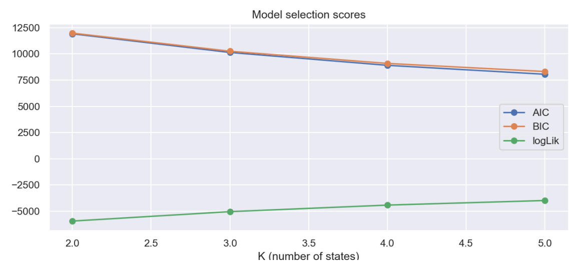

Fig. 3. Model selection: AIC, BIC, and log-likelihood for increasing numbers of hidden states.

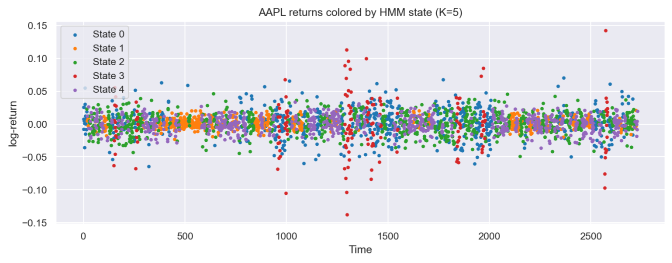

Fig. 4. AAPL log-returns colored by inferred HMM state.

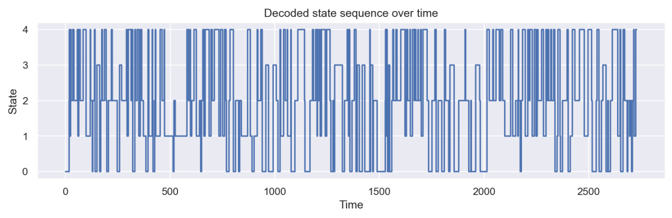

Fig. 5. Decoded hidden state sequence over time.

Bayesian Information Criterion (BIC) were computed. The BIC is defined as

$$\text{BIC} = -2 \log \hat{L} + p \log N, \tag{5}$$

where p is the number of free parameters and N is the number of time points.

The BIC curve attains its minimum at $K = 5$, suggesting a five-regime model provides a good trade-off between fit and complexity.

E. Inference and Interpretation

1) Decoded Hidden States: Given the trained model, the most likely hidden state sequence is recovered using the Viterbi algorithm. Each observation is assigned to

$$\hat{z}_t = \arg\max_i P(z_t = i \mid X_{1:T}).$$

Visual inspection reveals regimes that correspond to:

- low-volatility stable periods,
- moderate volatility with neutral returns,
- high-volatility correction phases,
- crisis-like phases with strongly negative returns,
- recovery or rebound regimes.

TABLE I
EXAMPLE STRUCTURE OF A 5-STATE HMM TRANSITION MATRIX.

	1	2	3	4	5
1	a_{11}	a_{12}	a_{13}	a_{14}	a_{15}
2	a_{21}	a_{22}	a_{23}	a_{24}	a_{25}
3	a_{31}	a_{32}	a_{33}	a_{34}	a_{35}
4	a_{41}	a_{42}	a_{43}	a_{44}	a_{45}
5	a_{51}	a_{52}	a_{53}	a_{54}	a_{55}

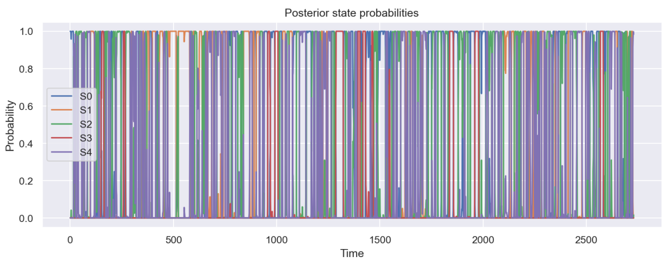

Fig. 6. Posterior probabilities of the five hidden states over time.

2) Transition Matrix: The transition matrix A summarizes regime persistence and switching behavior:

$$A_{ij} = P(z_{t+1} = j \mid z_t = i). \tag{6}$$

Large diagonal entries indicate that once entered, a regime tends to persist for several days, which is typical of financial markets.

The learned transition matrix is saved as `lab5_AAPL_transition_matrix.csv` by the Python script. A schematic representation is shown below (values are placeholders):

F. Posterior Probabilities and Evaluation

1) Posterior State Probabilities: The forward–backward algorithm yields the smoothed posterior probabilities

$$P(z_t = i \mid X_{1:T}),$$

which capture the uncertainty in state assignments over time.

Periods where a single state has high posterior probability correspond to clearly defined regimes, while overlaps indicate ambiguous transitions.

2) Model Behavior: The fitted Gaussian HMM reproduces several stylized facts of financial time series:

- **Volatility clustering:** High-volatility observations concentrate in specific regimes.
- **Regime persistence:** High diagonal transitions in A reflect persistent market phases.
- **Asymmetry:** Certain regimes exhibit negative mean returns with high variance, characteristic of crisis periods.

Choosing too few states merges distinct behaviors, while too many states can overfit and reduce interpretability. The BIC-guided choice of $K = 5$ provides a reasonable balance.

G. Insights and Regime Forecasting

The five inferred regimes can be summarized as:

- State 1: low volatility, steady growth,
- State 2: moderate volatility, neutral or slightly positive returns,
- State 3: volatile correction phases,
- State 4: stressed markets with negative returns,
- State 5: recovery phases after stress.

Given the current state probability vector $P(z_t)$ and transition matrix A, future regime distributions at horizon h are obtained by

$$P(z_{t+h}) = A^h P(z_t). \tag{7}$$

This enables estimating the probability of entering high-risk regimes, providing a useful tool for risk management and decision-making.

II. HOPFIELD NETWORKS FOR ASSOCIATIVE MEMORY AND COMBINATORIAL OPTIMIZATION

A. Learning Objective and Overview

The goal of this lab was to understand the dynamics of Hopfield networks and apply them to three types of problems:

- binary associative memory and storage capacity,
- constraint satisfaction in the Eight-Rook problem,
- and solving a small Traveling Salesman Problem (TSP) instance.

Hopfield networks are fully connected recurrent neural networks with symmetric weights and an energy (Lyapunov) function that decreases under asynchronous updates. This property allows them to act as content-addressable memories and as heuristic solvers for combinatorial optimization problems by encoding constraints into the energy landscape.

B. 10×10 Binary Associative Memory and Capacity

1) Network Model: For the associative memory experiment, a Hopfield network with $N = 10 \times 10 = 100$ binary neurons was used. Each neuron state $s_i \in \{-1, +1\}$ encodes one pixel of a 10×10 pattern (flattened to a 100-dimensional vector).

Given P training patterns $\{\boldsymbol{\xi}^\mu\}_{\mu=1}^P$, the classical Hebbian learning rule was used:

$$w_{ij} = \frac{1}{N} \sum_{\mu=1}^{P} \xi_i^\mu \, \xi_j^\mu, \quad w_{ii} = 0, \tag{8}$$

yielding a symmetric weight matrix $W = [w_{ij}]$ with zero diagonal. The network energy for a state $\mathbf{s}$ is

$$E(\mathbf{s}) = -\frac{1}{2} \sum_{i \neq j} w_{ij} s_i s_j. \tag{9}$$

Asynchronous updates of the form

$$s_i \leftarrow \operatorname{sgn}\left(\sum_j w_{ij} s_j \right) \tag{10}$$

guarantee that the energy is non-increasing, so the dynamics eventually converge to a stable attractor.

2) Storage Capacity: For random, uncorrelated ± 1 patterns, the theoretical storage capacity of a Hopfield network is approximately

$$P_{\max} \approx 0.138\,N, \tag{11}$$

before retrieval quality significantly degrades. For $N = 100$, this gives a capacity of roughly 13 distinct patterns.

In our experiments, we incrementally increased the number of stored patterns and tested retrieval by presenting noisy cues. Empirically, the network was able to reliably recall around 10–12 patterns without significant interference; beyond that, spurious attractors and pattern mixing started to appear, which is consistent with the theoretical limit.

C. Error-Correcting Capability

To study error correction, each stored pattern was corrupted by flipping a fraction of its bits at random, and the noisy pattern was used as initialization for the Hopfield dynamics.

For moderate corruption levels (e.g., up to roughly 20% of bits flipped), the network typically converged back to the correct stored pattern, demonstrating strong error-correcting behavior. As the Hamming distance increased, the probability of converging to a wrong memory or to a spurious attractor increased. This illustrates that each stored pattern corresponds to a basin of attraction in state space; the size of these basins shrinks as more patterns are stored.

D. Eight-Rook Problem with Hopfield Network

1) Problem Formulation: The Eight-Rook problem asks us to place 8 rooks on an 8×8 chessboard such that no two rooks attack each other, i.e., no two share the same row or column. This can be formulated as a binary constraint satisfaction problem and mapped to a Hopfield network.

We define binary neurons $x_{ij} \in \{0, 1\}$ for $i, j \in \{0, \ldots, 7\}$:

$$x_{ij} = \begin{cases} 1, & \text{if there is a rook at position } (i,j), \\ 0, & \text{otherwise.} \end{cases}$$

There are $N = 8 \times 8 = 64$ neurons in total.

The constraints are:

- C1 (Row constraint): exactly one rook per row

$$\sum_{j=0}^{7} x_{ij} = 1 \quad \forall i,$$

- C2 (Column constraint): exactly one rook per column

$$\sum_{i=0}^{7} x_{ij} = 1 \quad \forall j.$$

2) Energy Function: Using a quadratic penalty method, the energy function is defined as

$$E = \frac{A}{2} \sum_{i=0}^{7} \left(\sum_{j=0}^{7} x_{ij} - 1 \right)^2 + \frac{B}{2} \sum_{j=0}^{7} \left(\sum_{i=0}^{7} x_{ij} - 1 \right)^2, \tag{12}$$

where $A, B > 0$ are penalty weights. Any violation of the row or column constraints increases the energy, and configurations that satisfy all constraints have minimal energy ($E = 0$).

3) Weight Matrix and Biases: Expanding (12) shows how constraints induce couplings:

- For neurons in the same row i (positions (i, j) and (i, k) with $j \neq k$), the cross-term is

$$A \sum_{i} \sum_{j \neq k} x_{ij} x_{ik},$$

  which translates into a negative weight

$$w_{(i,j),(i,k)} = -A, \quad j \neq k.$$

  This discourages multiple rooks in the same row.
- Similarly, for neurons in the same column j (positions (i, j) and (k, j) with $i \neq k$), we obtain

$$w_{(i,j),(k,j)} = -B, \quad i \neq k,$$

  discouraging multiple rooks in the same column.

The corresponding bias term for each neuron arises from the linear part when expanding $(\sum x - 1)^2$. In the implemented network,

$$\theta_{ij} = -A - B,$$

which encourages activating exactly one neuron per row and per column to minimize the penalties.

The resulting weight matrix W and threshold vector $\boldsymbol{\theta}$ were constructed explicitly in the code, yielding a 64×64 symmetric matrix with zero diagonal and structured negative couplings along rows and columns.

4) Dynamics and Asynchronous Update Rule: The network uses an asynchronous update rule in binary $\{0, 1\}$ form. For each neuron indexed by (i, j) (flattened to index k), the local field is

$$h_k = \sum_{\ell} w_{k\ell} x_{\ell} - \theta_k,$$

and the update rule is

$$x_k \leftarrow \begin{cases} 1, & \text{if } h_k > 0, \\ 0, & \text{otherwise.} \end{cases}$$

Neurons are updated in random order, and the process repeats until the state stabilizes or a maximum number of iterations is reached.

5) Experimental Results: The implemented class `EightRookHopfield` constructs the weights and thresholds from the energy function and provides:

- a solver that tries multiple random initializations and reports valid solutions,
- an energy function and constraint violation counters,
- visualization routines for board configurations.

From multiple random starts, the network consistently converged to valid configurations (one rook per row and column). Different initializations led to different valid solutions among the $8! = 40{,}320$ possible placements, illustrating that the Hopfield dynamics explore the space of feasible solutions while respecting the encoded constraints.

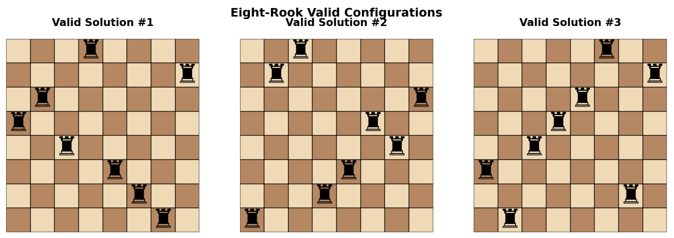

Fig. 7. Example valid Eight-Rook configurations found by the Hopfield network.

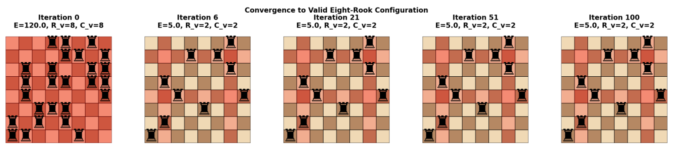

Fig. 8. Convergence of a random initial board to a valid Eight-Rook configuration (energy and violations decreasing over iterations).

E. Traveling Salesman Problem (10 Cities) with Hopfield Network

1) Encoding and Energy Function: For a Traveling Salesman Problem (TSP) with n cities, the standard Hopfield formulation uses n^2 neurons v_{xi}, where

$$v_{xi} = \begin{cases} 1, & \text{if city } x \text{ is visited at tour position } i, \\ 0, & \text{otherwise,} \end{cases}$$

with $x \in \{1, \dots, n\}$ and $i \in \{1, \dots, n\}$.

The constraints are:

- each city appears exactly once in the tour:

$$\sum_{i=1}^{n} v_{xi} = 1 \quad \forall x,$$

- exactly one city is visited at each tour position:

$$\sum_{x=1}^{n} v_{xi} = 1 \quad \forall i.$$

These constraints are encoded as quadratic penalties in the energy function. The tour length is also added as a term:

$$E = \frac{A}{2}\sum_{x}\left(\sum_{i} v_{xi} - 1\right)^2 + \frac{B}{2}\sum_{i}\left(\sum_{x} v_{xi} - 1\right)^2 + \frac{C}{2}\sum_{i}\sum_{x}\sum_{y} d_{xy}\, v_{xi}\, v_{y,i+1}, \tag{13}$$

where d_{xy} is the distance between cities x and y, the index $i+1$ is taken modulo n, and A, B, C are positive constants. Minimizing E encourages the network to satisfy the constraints and to choose a short tour.

2) Number of Weights for 10-City TSP: The TSP Hopfield network for n cities has

$$N = n^2$$

neurons. Assuming a fully connected symmetric Hopfield network with zero self-connections, the total number of distinct weights is

$$\#\text{weights} = \frac{N(N-1)}{2}.$$

For $n = 10$ cities:

$$N = 10^2 = 100, \quad \#\text{weights} = \frac{100 \times 99}{2} = 4950.$$

Thus, solving a 10-city TSP with this Hopfield formulation requires a 100×100 weight matrix with 4950 independent weights, in addition to the bias terms derived from the constraint penalties.

F. Discussion

Across these experiments, Hopfield networks demonstrated:

- associative memory behavior with finite storage capacity and error correction,
- the ability to encode hard combinatorial constraints (Eight-Rook) via quadratic penalty terms,
- and a principled way to tackle optimization problems like TSP by shaping the energy landscape to penalize constraint violations and long tours.

While Hopfield networks do not guarantee globally optimal solutions for NP-hard problems, they provide an elegant energy-based framework that unifies memory, constraint satisfaction, and heuristic optimization within a single recurrent neural architecture.

III. Matchbox Educable Noughts and Crosses Engine (MENACE) and N-Armed Bandits

A. Learning Objective and Overview

The goal of this lab was to:

- understand basic data structures for state-space search in simple games,
- use randomization for Markov Decision Processes (MDP) and Reinforcement Learning (RL),
- and study the exploitation–exploration trade-off in n-armed bandit problems using ε-greedy algorithms.

We implemented a modern variant of the classical **MENACE** (Matchbox Educable Noughts and Crosses Engine) for tic-tac-toe and combined it with bandit environments (binary and non-stationary 10-armed bandits) to empirically study ε-greedy behavior.

B. MENACE for Tic-Tac-Toe

1) State Representation and Symmetry Reduction: The tic-tac-toe board is represented as a 9-character string (or list) over the alphabet {`'X'`, `'O'`, `'.'`}, for example:

```
X.O...O..
```

for a 3×3 board in row-major order.

Directly treating every distinct board as a separate state leads to redundancy, since many boards are symmetric under rotations and reflections. To reduce the state space, we:

- generate all 8 symmetries of a board (4 rotations $\times$ 2 reflections),
- and choose the lexicographically smallest representation as the **canonical board**.

This canonicalization significantly reduces the number of distinct matchboxes MENACE needs to store.

Each canonical state maps to a *matchbox* represented as a `Counter` from legal moves (empty cell indices) to bead counts:

$$\texttt{boxes[canon\_state] : move} \mapsto \texttt{beads}.$$

This encodes MENACE's stochastic policy.

2) Action Selection and Game Play: MENACE always plays as `'X'` (first player). At its turn:

1) The current board is canonicalized to obtain `canon_state`.
2) The corresponding matchbox is initialized if unseen, assigning a default number of beads to each legal move.
3) A move is sampled *proportionally to bead counts*, implementing a weighted random choice:

$$P(\text{move} = m \mid \text{state}) \propto \text{beads}(m).$$

The opponent can be:

- **random**: chooses a uniformly random legal move,
- **greedy**: chooses center if free, otherwise corners, else random.

After each game finishes, we check for:

- MENACE win (reward $+1$),
- loss (reward -1),
- or draw (reward 0).

The sequence of `(canonical_state, move)` pairs used by MENACE during the game is recorded as its *history*.

3) Reinforcement Mechanism: At the end of each game, the bead counts are updated based on the outcome:

- **Win**: add `win_add` beads to each chosen move,
- **Draw**: add `draw_add` beads,
- **Loss**: remove `lose_remove` beads, but never below a `min_beads` floor.

This implements a simple form of reinforcement learning:

- successful moves become more likely in similar future states,
- unsuccessful moves lose beads and become less likely, but are not completely forbidden (allowing continued exploration).

Over many games, MENACE's empirical win rate increases and it learns to avoid obviously losing moves, illustrating how a purely combinatorial, matchbox-based system can implement a policy improvement mechanism.

C. n-Armed Bandit Environments and ε-Greedy Agents

1) Bandit Environments:

a) Binary Bandits: Two simple stationary bandits were implemented, each with two Bernoulli arms:

- **BinaryBanditA**: $p_1 = 0.1$, $p_2 = 0.2$,
- **BinaryBanditB**: $p_1 = 0.8$, $p_2 = 0.9$.

Pulling arm $a \in \{0, 1\}$ returns a reward $r \in \{0, 1\}$ with probability p_a. The optimal arm is the one with larger p_a.

b) Non-Stationary 10-Armed Bandit: To model changing environments, a 10-armed non-stationary bandit was implemented:

- each arm a has an underlying mean μ_a,
- at each time step, all means evolve via a random walk:

$$\mu_a \leftarrow \mu_a + \mathcal{N}(0, \sigma_{\text{walk}}),$$

- pulling arm a yields reward

$$r \sim \mathcal{N}(\mu_a, \sigma_{\text{reward}}).$$

The optimal arm thus changes over time, making stationary estimators suboptimal.

2) Epsilon-Greedy Agents: Two ε-greedy agents were implemented.

a) Sample-Average ε-Greedy: For stationary bandits, the sample-average agent maintains:

- action-value estimates $Q_t(a)$ for each arm,
- action counts $N_t(a)$.

At each time step:

- with probability ε: **explore** by choosing a random arm,
- with probability $1 - \varepsilon$: **exploit** by choosing an arm with maximal $Q_t(a)$ (ties broken randomly).

After observing reward r_t, the estimate for the selected arm a_t is updated via incremental sample average:

$$N_{t+1}(a_t) = N_t(a_t) + 1, \tag{14}$$

$$Q_{t+1}(a_t) = Q_t(a_t) + \frac{1}{N_{t+1}(a_t)} \left(r_t - Q_t(a_t)\right). \tag{15}$$

This converges to the true mean for stationary environments.

b) Constant Step-Size ε-Greedy: For non-stationary bandits, a constant step-size α is preferable, since older data should be discounted. The constant-α agent uses:

$$Q_{t+1}(a_t) = Q_t(a_t) + \alpha \left(r_t - Q_t(a_t)\right), \tag{16}$$

with the same ε-greedy action selection. This implements an exponential moving average over rewards and allows the agent to track drifting means.

3) Exploitation–Exploration Trade-off: The ε-greedy strategy implements a simple but powerful exploitation–exploration mechanism:

- **Exploitation**: choosing the empirically best arm based on current Q estimates,
- **Exploration**: occasionally trying a random arm with probability ε to discover potentially better options.

A too-small ε leads to premature exploitation and getting stuck with suboptimal arms, while a too-large ε wastes time exploring even after good arms have been identified.

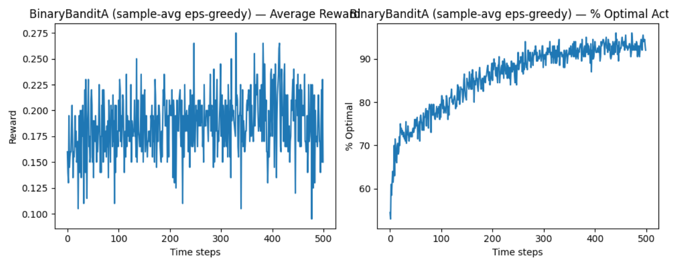

Fig. 9. BinaryBanditA: sample-average ε-greedy agent. Left: average reward; right: % optimal action over time.

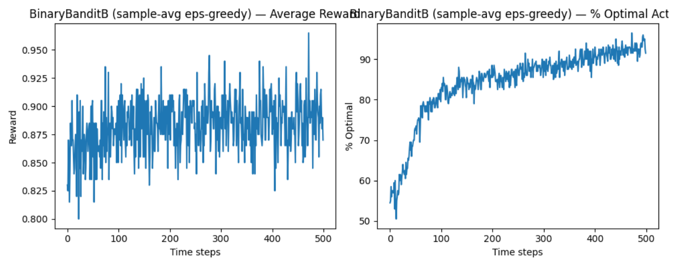

Fig. 10. BinaryBanditB: sample-average ε-greedy agent in an easier (high-reward) setting.

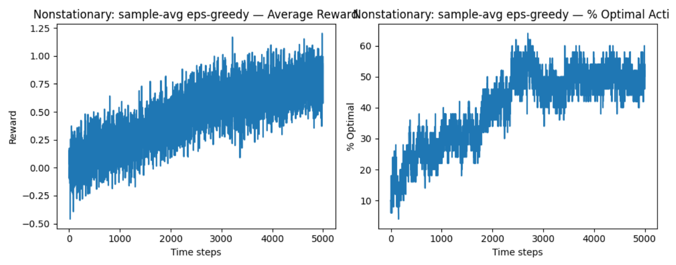

Fig. 11. Non-stationary 10-armed bandit: sample-average ε-greedy (stationary assumption).

D. Experimental Results

1) Binary Bandits: We ran the sample-average ε-greedy agent (with $\varepsilon = 0.1$) on both BinaryBanditA and BinaryBanditB over multiple runs. For each time step, we recorded:

- average reward,
- percentage of times the agent selected the optimal arm.

In both environments, the percentage of optimal actions increases over time, and the average reward curves approach the optimal achievable reward. This confirms that with a fixed ε the agent successfully balances exploration and exploitation in stationary settings.

2) Non-Stationary 10-Armed Bandit: For the non-stationary bandit, we compared:

- sample-average ε-greedy (stationary-style),
- constant-α ε-greedy (non-stationary-aware).

The constant-α agent achieves:

- higher long-run average reward,
- and a higher percentage of optimal actions,

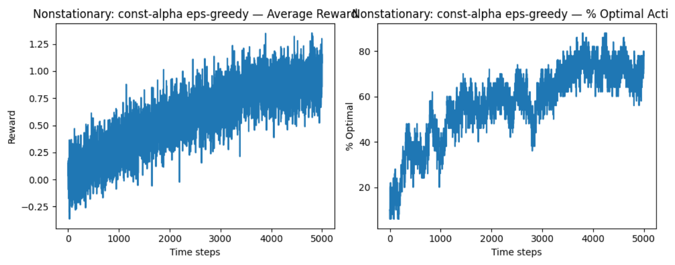

Fig. 12. Non-stationary 10-armed bandit: constant-α ε-greedy (tracks drifting means better).

because it discounts outdated information and continuously adapts to the changing reward distribution, whereas the sample-average agent's estimates lag behind due to averaging over the entire past.

E. Discussion

This lab connects classical matchbox-based learning (MENACE) with modern RL concepts:

- **MENACE** illustrates state-space encoding, symmetry reduction, and tabular reinforcement through bead updates.
- **Bandit experiments** formalize the exploitation–exploration trade-off using ε-greedy policies, highlighting the role of step-size choice in stationary vs non-stationary settings.

Together, these experiments provide an intuitive bridge from simple game learning mechanisms to the core ideas underlying MDPs and RL algorithms.

IV. Policy Iteration for the Gbike Bicycle Rental Problem

A. Learning Objective and Overview

This experiment extends the classical "Jack's Car Rental" problem into a bicycle rental scenario (Gbike) with additional real-world constraints. The objectives are to:

- formulate the rental system as a Markov Decision Process (MDP),
- derive the Bellman equations for policy evaluation and improvement,
- implement Policy Iteration to compute an optimal bike-transfer policy,
- incorporate a free shuttle and parking penalties into the reward and transition structure.

Two rental locations share a finite pool of bicycles. Each day, stochastic rental demands and returns occur at both locations. Overnight, the manager chooses how many bikes to move between locations to maximize expected long-term discounted profit.

B. MDP Formulation

1) State and Action Space: A state s is defined by the number of bikes at each location at the end of a day:

$$s = (i, j),$$

where i is the number of bikes at location 1 and j is the number at location 2. To keep the state space finite, both i and j are truncated to a maximum of 20:

$$0 \leq i, j \leq 20.$$

An action a denotes the number of bikes moved *overnight* from location 1 to location 2:

$$a \in \{-5, \dots, 5\},$$

where $a > 0$ means moving bikes from location 1 to 2, and $a < 0$ denotes moving bikes from 2 to 1, subject to feasibility constraints so that bike counts never become negative.

After choosing a, the post-move inventory becomes

$$i' = i - a, \tag{17}$$
$$j' = j + a, \tag{18}$$

before the next day's rental process.

2) Stochastic Demand and Returns: Rental requests at locations 1 and 2 are modeled as Poisson random variables:

$$D_1 \sim \text{Poisson}(\lambda_1), \quad D_2 \sim \text{Poisson}(\lambda_2).$$

Actual rentals are limited by available bikes:

$$r_1 = \min(i', D_1), \tag{19}$$
$$r_2 = \min(j', D_2). \tag{20}$$

Bike returns are also Poisson random variables:

$$R_1 \sim \text{Poisson}(\mu_1), \quad R_2 \sim \text{Poisson}(\mu_2).$$

The next state (i_{t+1}, j_{t+1}) after rentals and returns is

$$i_{t+1} = \min(i' - r_1 + R_1, 20), \tag{21}$$
$$j_{t+1} = \min(j' - r_2 + R_2, 20). \tag{22}$$

This defines the transition probabilities $P(s' \mid s, a)$ via the Poisson distributions.

C. Reward Function and Modified Cost Structure

1) Rental Revenue: Each successful rental yields a fixed reward (e.g., \$10 per bike). For a given (s, a) and random demands,

$$\text{rental reward} = 10(r_1 + r_2),$$

and we work with its expected value in $R(s, a)$.

2) Movement Cost with a Free Shuttle: There is a free shuttle that automatically moves one bike from location 1 to location 2 each night. The movement cost function is:

$$C_{\text{move}}(a) = \begin{cases} 2(a-1), & a > 1, \\ 0, & a = 1, \\ 2|a|, & a \leq 0, \end{cases} \tag{23}$$

so:

- moving exactly one bike from 1 to 2 is free (covered by the shuttle),
- moving more than one from 1 to 2 pays for *extra* bikes beyond the free one,
- moving bikes in the opposite direction or not using the shuttle still incurs a normal per-bike transport cost.

3) Parking Penalty: Parking more than ten bikes at any location incurs a fixed penalty to reflect limited storage and congestion:

$$C_{\text{park}}(i', j') = 4\,\mathbb{I}(i' > 10) + 4\,\mathbb{I}(j' > 10), \tag{24}$$

where $\mathbb{I}(\cdot)$ is the indicator function.

4) One-Step Expected Reward: The expected immediate reward is

$$R(s, a) = \mathbb{E}\big[10(r_1 + r_2)\big] - C_{\text{move}}(a) - C_{\text{park}}(i', j'). \tag{25}$$

This reward, combined with the transition model, fully specifies the MDP.

D. Policy Iteration

1) Value Functions and Bellman Equations: For a fixed policy π mapping each state to an action, the state-value function is

$$V^\pi(s) = R(s, \pi(s)) + \gamma \sum_{s'} P(s' \mid s, \pi(s)) V^\pi(s'), \tag{26}$$

with discount factor $\gamma \in (0, 1)$. The optimal value function V^* satisfies:

$$V^*(s) = \max_a \left[R(s, a) + \gamma \sum_{s'} P(s' \mid s, a) V^*(s') \right]. \tag{27}$$

2) Policy Evaluation: Given a current policy π, policy evaluation iteratively updates $V(s)$ until convergence:

$$V_{k+1}(s) \leftarrow R(s, \pi(s)) + \gamma \sum_{s'} P(s' \mid s, \pi(s)) V_k(s'), \tag{28}$$

for all states s, stopping when the maximum change $|V_{k+1}(s) - V_k(s)|$ is below a small threshold θ.

3) Policy Improvement: With $V(s)$ fixed, policy improvement updates the policy greedily:

$$\pi_{\text{new}}(s) = \arg\max_a \left[R(s, a) + \gamma \sum_{s'} P(s' \mid s, a) V(s') \right]. \tag{29}$$

If $\pi_{\text{new}}(s) = \pi(s)$ for all s, the policy is stable and therefore optimal.

4) Overall Policy Iteration Loop: Policy Iteration alternates between policy evaluation and policy improvement until the policy becomes stable. For the Gbike setup with the modified cost structure, the implementation converged in a small number of outer iterations (4 policy improvement steps).

E. Results

1) Optimal Policy: The optimal policy can be visualized as a 2D heatmap over (i, j), where each cell shows the optimal action $a^*(i, j)$:

- positive values indicate moving bikes from location 1 to location 2,
- negative values indicate moving bikes from 2 to 1,
- zero indicates no movement.

The learned policy reflects several intuitive behaviors:

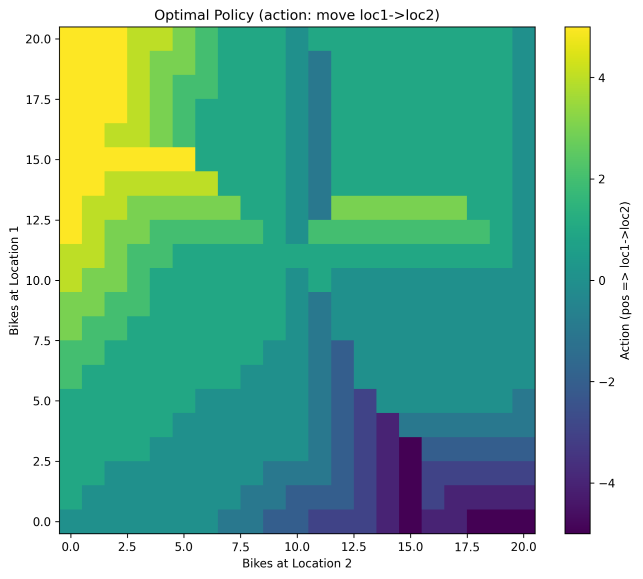

Fig. 13. Optimal action (number of bikes to move from location 1 to 2) for each state (i, j).

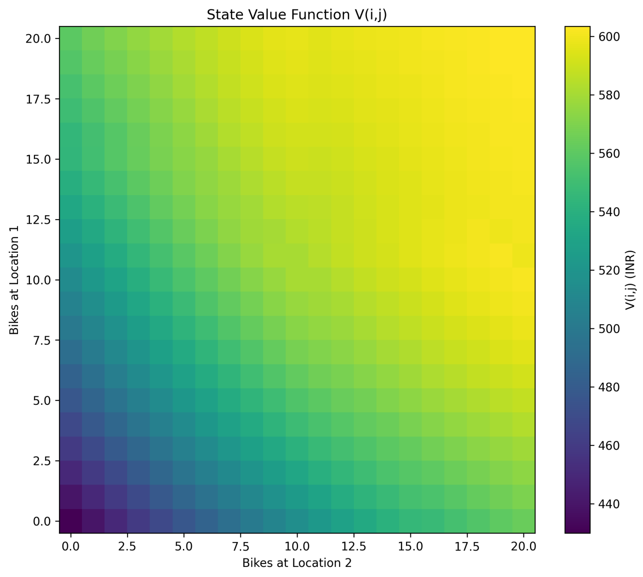

Fig. 14. Optimal state-value function $V^*(i, j)$ over bike counts at locations 1 and 2.

- it exploits the free shuttle by frequently using $a = 1$ when beneficial,
- it avoids states with more than ten bikes at a location to reduce parking penalties,
- it shifts bikes towards locations with systematically higher demand.

2) State-Value Function: The corresponding optimal value function $V^*(i, j)$ quantifies the long-term expected discounted return from each state under the optimal policy.

Higher values cluster around balanced inventories that can flexibly serve demand at both locations while avoiding excessive parking penalties.

F. Discussion

This experiment demonstrates how:

- real-world operational constraints (free transfer capacity, parking limits) can be encoded directly in the reward function of an MDP,
- Poisson-based stochastic modeling provides realistic transition dynamics for rental systems,
- Policy Iteration efficiently computes an optimal stationary policy in a finite state and action space.

The Gbike problem thus serves as a concrete example of how dynamic programming methods in Reinforcement Learning can produce interpretable and practically meaningful control strategies in stochastic resource allocation tasks.

ACKNOWLEDGMENT

This work is based on the Week 5 lab assignment under the module covering graphical models and HMM-based inference in the CS307/367 Artificial Intelligence: Agent course.

REFERENCES

[1] W. Zucchini, I. L. MacDonald, and R. Langrock, *Hidden Markov Models for Time Series*. CRC Press, 2016.
[2] Yahoo Finance API. [Online]. Available: https://pypi.org/project/yfinance/
[3] “hmmlearn documentation.” [Online]. Available: https://hmmlearn.readthedocs.io/
[4] AI Lab End-Sem Report – GitHub Repository. [Online]. Available: https://github.com/drowningFi5h/AI_LAB_endsem