CS 5100 Final Project Report

Abhishek Tuteja

Spring 2025

GITHUB LINK

1. Introduction

As a graduate student in Computer Science with a strong interest in finance and artificial intelligence, I've always been fascinated by the intersection of these two domains. My early experience in stock trading was entirely manual and rule-based, lacking adaptability to changing market trends. With the knowledge gained in CS 5100, especially in Reinforcement Learning and Markov Decision Processes, I was excited to explore whether an AI agent could learn to make profitable trading decisions over time. This project represents a continuation and deepening of my interest in applying machine learning to stock market strategies.

The central question I aim to answer in this project is: Can we utilize Reinforcement Learning, particularly Q-learning, to train an agent that learns an effective buy/sell/hold strategy in a simulated trading environment based on historical stock data?

The scope of the project was kept focused: I used one stock (AAPL), a single-agent offline trading simulation, and avoided complexities like live data, transaction costs, or deep learning models. This allowed me to concentrate on MDP formulation and agent behavior.

2. Analysis

The project implementation included three components:
- Data Preparation: Using yfinance and the ta library, I collected historical daily prices for AAPL and computed technical indicators such as SMA, EMA, RSI, MACD, and Bollinger Bands. This was saved as AAPL_processed.csv.
- Trading Environment: I built a custom simulator (TradingEnv) that modeled each day as a timestep where the agent could Buy, Sell, or Hold. Rewards were computed based on changes in portfolio value.
- Q-Learning Agent: I implemented a tabular Q-learning agent with an epsilon-greedy policy. States were discretized using rounded indicator values. The agent learned over 1000 episodes.

The agent's performance was compared to a Buy & Hold strategy. Metrics such as total reward per episode and final portfolio value were plotted. The agent showed variability across episodes, sometimes exceeding Buy & Hold, and demonstrated adaptive behavior.

2. Analysis (with Code References)

The project was implemented through three Python modules: `data_prep.py`, `trading_env.py`, and `q_learning.py`, each corresponding to a major step in the AI workflow: data preparation, environment simulation, and agent training.

2.1 Data Preparation – `data_prep.py`

This script uses the `yfinance` library to download daily stock data for AAPL from 2015 to 2024. It also uses the `ta` library to generate technical indicators like:
- SMA (Simple Moving Average)
- EMA (Exponential Moving Average)
- RSI (Relative Strength Index)
- MACD (Moving Average Convergence Divergence)
- Bollinger Bands

These features are added as columns to the raw price data, creating a rich dataset for decision-making. The output is saved as `AAPL_processed.csv` for use in the simulator.

2.2 Environment – `trading_env.py`

This file defines a `TradingEnv` class that models stock trading as a Markov Decision Process (MDP). Each day in the historical stock data corresponds to one time step. The environment keeps track of:
- Agent's cash balance
- Number of shares held
- The total portfolio value (cash + shares * current price)

Actions include:
- 0: Hold
- 1: Buy (if cash allows)
- 2: Sell (if shares are held)

Rewards are computed as the change in total asset value after an action is taken, using the next day's closing price.

2.3 Agent & Training Loop – `q_learning.py`

This script contains both the Q-learning agent implementation and the training loop. The `QLearningAgent` class uses a Q-table (dictionary of state-action values) to learn from interaction with the environment. Key components:
- `select_action(state)`: Chooses an action using an epsilon-greedy strategy
- `update(state, action, reward, next_state)`: Applies the Q-learning update rule
- States are discretized tuples derived from rounding selected indicators
- Actions are mapped to 0 (Hold), 1 (Buy), 2 (Sell)

The agent is trained over 1000 episodes, and the script tracks:

- Total reward earned in each episode
- Final portfolio value
- Buy & Hold strategy return for comparison
Results are plotted using `matplotlib` for visual analysis.

Throughout the project, core AI concepts from CS 5100 were applied:
- Markov Decision Processes: Modeling state transitions and rewards
- Reinforcement Learning: Q-learning algorithm with temporal-difference updates
- Exploration vs. Exploitation: Epsilon-greedy policy
These were implemented in a modular and reproducible way, illustrating the real-world applicability of class concepts.

3. Conclusion

The Q-learning agent learned trading behavior that occasionally outperformed Buy & Hold strategies, confirming that reinforcement learning can be effectively applied to financial decision-making. However, the project also revealed the limitations of tabular methods in high-dimensional, noisy environments.

Future improvements include using Deep Q-Learning, modeling multiple stocks, and introducing transaction costs and slippage. Additionally, better state representations and reward shaping could yield more robust strategies.

This project was extremely valuable to me: it solidified my understanding of MDPs and reinforcement learning, and gave me a hands-on opportunity to apply these concepts to a problem domain I'm passionate about. I can see myself extending this work further—either in a future research course or as a personal project in algorithmic trading.