Economics Meets Machine Learning

Seasons of Code

Final Report

Saksham Rathi
Second-Year Computer Science Undergraduate
22B1003

Under the mentorship of **Tejas Pagare**

1 Objective

This project involves implementing various Economics problems as a Markov Decision Process. Some economic problems that we dealt with include Matching Markets, Auctions, and allocation problems.

2 An Overview of the Submission

All the work produced during this summer has been added to a github repository (https://github.com/sakshamrathi21/Seasons-of-Code---Economics-Meets-Machine-Learning). This folder includes a presentation video which will explain briefly all the codes and presentations produced as a part of this Project. It also includes some of the books and research papers we followed.

3 Work Accomplished

The work which was completed by me has been summarized in the following two sub-sections. I had chosen **Stock Exchange** as my model for setting up double auction. The overall code structure of the double auction was taken from a research paper shared by our mentor.

3.1 Part 1 of the Problem Statement

Part 1

Design a Double Auction model, which based on several properties, outputs true valuation of each buyer B_i and seller S_i.

Algorithm 1: Double Auction (Ride Sharing)

Input: luxury, urgency, tour length, time of the ride, traffic, etc.
Output: True valuation of each rider (buyer) and driver (seller)
E.g. If rider (buyer) 1, needs to reach the airport in night urgently, then true valuation can be
True valuation of rider 1: $\underset{\text{night charges}}{10\$} + \underset{\text{tour length charges}}{100\$} + \underset{\text{urgency charges}}{50\$} + \underset{\text{luxury charges}}{50\$} = 210\$$

The question statement of the part 1 is shown in the above figure 3.1. My double auction model used various properties including buyer's target, buyer's stop-loss, cash in hand of the buyer, the market sentiment (overall market is bullish or bearish), stock quantity of each buyer and seller and so on. Based on these properties, the bid value or the true valuation was changed. So, suppose if the market sentiments are bullish (positive), then the buyer's bid value was increased and it's target was also increased. If the market was up for a continuous period of 5 days, then the buyer got switched from short-term to a long-term player in the auction. Similarly, each participant has a random variable associated with it for the calculations of regret and utilities. I have also created space so that we can add more number of stocks and thereby players can diversify their portfolio.

```python
class Participant:
    def __init__(self, id, type, participating, cash_in_hand, stock, estimate, stock_quantity, random_variable, long_only, target, stoploss):
        self.id = id
        self.type = type
        self.participating = participating
        self.cash_in_hand = cash_in_hand
        self.stock = stock
        self.estimate = estimate
        self.stock_quantity = stock_quantity
        self.random_variable = random_variable
        self.long_only = long_only
        self.bid = None
        self.utility = []
        self.target = target
        self.stoploss = stoploss
```

Figure 1: Participant Class in the Python Code Implemented

Part 2

Learning? No buyer and seller knows their true valuations and only observe the noisy profit after each bidding round. (Refer to the Paper)

Assumption: The algorithm requires that each seller sells a **single type** of item which are indistinguishable across sellers.

Modify the model made in Part 1 such that it satisfies the assumption.
E.g. for ride sharing, one can have many drivers (sellers) waiting at a single stop, however they might have different types of cars and hence different valuations. Riders (sellers) in this case, might be people traveling at the same time to the airport, some prefer luxury some may not, and hence different valuations. These drivers and riders will constitute of the participating agents in the auction.

Problem Statement: No buyer and seller knows the true valuation. They need to identify their true valuations by interacting with the platform for multiple double auction bidding rounds.
Solution: Each buyer and seller will strategically bid using the LCB and UCB estimates. (Refer to the paper)

3.2 Part 2 of the Problem Statement

The question statement of the part 2 is shown in the above figure 3.2. In this part of the code, various changes were made by me so as to satisfy the assumption. So, all the traders (buyers and sellers) in the stock exchange were trading a single stocks. I also diminished the properties of the buyers and sellers involved so that no buyer/seller is favoured on the basis of the his/her targets and stop-losses. Then the complete code was run assuming each buyer/seller don't know their true valuation. After each round their valuation is updated on the basis of a parameter alpha and the number of rounds the participant has played. The auction followed the average price mechanism i.e. the trading price is the average of the last profitable buyer and seller. This structure of running the auction and mechanism to calculate subsequent valuations was adopted from the research paper provided by our Mentor. With this code, the buyer and seller updated their bids, and hence a

```
def run_auction(self, num_rounds):
    number_rounds_buyers = [0]*self.num_buyers
    number_rounds_sellers = [0]*self.num_sellers
    alpha = 0.05
    for t in range(1, num_rounds+1):
        sorted_buyer_bids = sorted(self.buyers, key = lambda x: x.estimate, reverse = True)
        sorted_seller_bids = sorted(self.sellers, key = lambda x: x.estimate)
        sorted_buyer_bids = [x for x in sorted_buyer_bids if x.participating]
        sorted_seller_bids = [y for y in sorted_seller_bids if y.participating]
        K=0

        while K<min(len(sorted_buyer_bids), len(sorted_seller_bids)) and sorted_buyer_bids[K].estimate >= sorted_seller_bids[K].estimate:
            K+=1

        participating_buyers = sorted_buyer_bids[:K]
        participating_sellers = sorted_seller_bids[:K]
        for buyer in participating_buyers:
            number_rounds_buyers[buyer.id-1] += 1
            buyer.estimate = buyer.estimate + np.sqrt(alpha*np.log(t)/number_rounds_buyers[buyer.id-1])
        for seller in participating_sellers:
            number_rounds_sellers[seller.id-1] += 1
            seller.estimte = seller.estimate - np.sqrt(alpha*np.log(t)/number_rounds_sellers[seller.id-1])
        trading_prices = {}

        for stock, price in self.stock_prices.items():
            trading_price = (participating_buyers[-1].estimate + participating_sellers[-1].estimate)/2
            trading_prices[stock] = trading_price

        self.trading_prices[t] = trading_prices
```

Figure 2: Function running the auction code

significant learning was involved as compared to the previous part. The code also plotted a graph showing the results. As we can see in the figure 3, two graphs are plotted. One is the trading price

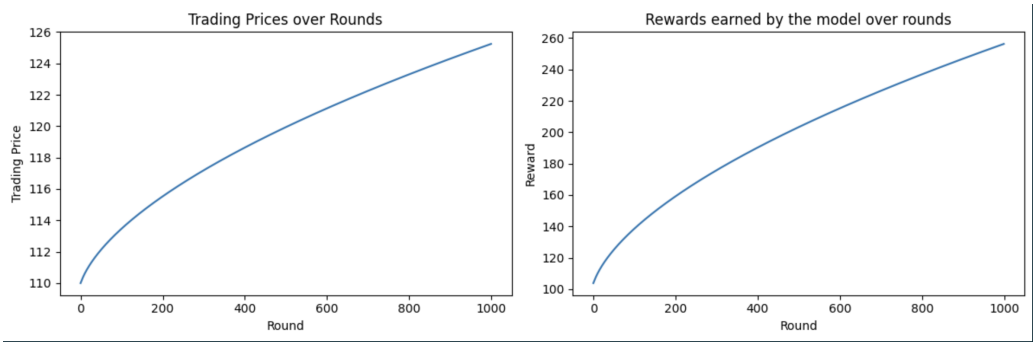

Figure 3: Graphs showing the outputs produced

vs the rounds for which the auction ran. As we can see the trading price increased till a optimum was reached after large number of rounds. The model also shows that the total reward earned in the stock exchange increased over the number of rounds. This clearly demonstrates the correctness of the algorithm and the code implemented.