Многорукие бандиты: торговый бот

18 декабря 2024 г.

Клевайчук Александр, Михеева Екатерина, Молвинская Алина

1 Введение

Многорукий бандит — это алгоритм принятия решений, суть которого заключается в обеспечении баланса между исследованием и использованием какого-то из рассматриваемых вариантов решения с целью максимизации целевой метрики и минимизации убытков.

Этот термин описывает ситуацию, в которой агент сталкивается с несколькими альтернативами (или "руками игровых автоматов"), каждая из которых имеет свою скрытую вероятность выигрыша. Метод многоруких бандитов подходит для задач, связанных с принятием решений в условиях неопределенности. В рамках этого подхода можно разработать торговую стратегию, которая будет адаптироваться и улучшаться на основе собранного опыта и данных о рынке.

Цели и задачи проекта:

Цели

- Разработка торгового бота
- Оптимизация торговой стратегии
- Подключение к реальным данным

Задачи

- Изучение и анализ стратегий
- Сбор и обработка данных
- Проектирование и реализация торговой стратегии
- Тестирование

2 Теоретическая основа

2.1 Математическая постановка

Мы рассматриваем задачу многоруких бандитов (MAB): нам дан игровой автомат с N рукоятками; на каждом временном шаге ($t = 1, 2, 3, \ldots$) необходимо выбрать одну из N рукояток для игры. Каждая рукоятка i при игре приносит случайное вознаграждение, которое распределено по некоторому фиксированному (неизвестному) распределению с поддержкой на интервале $[0, 1]$. Случайные вознаграждения, получаемые при многократной игре на одной рукоятке, являются независимыми и идентично распределёнными и не зависят от игр на других рукоятках. Вознаграждение наблюдается немедленно после игры на рукоятке.

Алгоритм для задачи MAB должен решать, какую рукоятку играть на каждом временном шаге t, исходя из результатов предыдущих $t-1$ игр. Пусть i обозначает (неизвестное) ожидаемое вознаграждение для рукоятки i. Целью является максимизация математического ожидания общего вознаграждения за время T, т.е.

$$E\left[\sum_{t=1}^{T} \mu_{i(t)}\right]$$

где $i(t)$ — это рукоятка, сыгранная на шаге t, и ожидание берётся по случайным выборам $i(t)$, сделанным алгоритмом.

Главная проблема задачи многоруких бандитов заключается в балансе между:

Exploration (исследование): выбор рук, чтобы собрать больше информации о распределениях награды.

Exploitation (использование): выбор наилучшей из уже известных рук для максимизации награды на текущем шаге.

Например, если стратегия будет слишком часто исследовать, она может упустить возможность получить стабильные награды от рук с высокой известной доходностью. С другой стороны, чрезмерная эксплуатация может привести к тому, что стратегия не найдет лучшую руку.

2.2 Основные алгоритмы

- ϵ-greedy;
- UCB (upper confidence bound);
- Thompson sampling

1) ϵ-greedy: с вероятностью 1 - ϵ выбирает руку с наибольшей текущей наградой (использование). С вероятностью ϵ выбирает случайную руку (исследование). Значение ϵ может уменьшаться с течением времени.

2) UCB проводит своё исследование не случайно, а на основе растущей со временем неопределённости у стратегий. В начале работы алгоритм случайно задействует все стратегии, после чего рассчитывается средняя награда каждой. Далее после каждой итерации обновляются средние награды стратегий. С течением времени чем реже выбиралась та или иная стратегия, тем больше будет у неё неопределённость. Окончательный выбор стратегии — это максимальная сумма средней награды и неопределённости среди всех стратегий.

3) Thompson sampling использует байесовский подход для моделирования распределения награды каждой руки с помощью априорных и апостериорных вероятностей.На каждом шаге генерируется случайное значение из распределения для каждой руки, и выбирается рука с наибольшим значением.

2.3 Метрики

В качестве метрик в нашей задаче мы использовали:

- total rewards (Сумма всех полученных наград за время эксперимента)

$$r_{total,t} = \sum_{i=1}^{t} R_i \tag{1}$$

- total regret (Потери из-за выбора не оптимальных рук)

$$reg_t = \sum_{i=1}^{t} R_{opt} - R_t$$

Метрика total rewards была выбрана, поскольку для торгового бота основная цель — максимизация прибыли. Total rewards - метрика, которая показывает, насколько хорошо бот зарабатывает деньги.

Total regret измеряет, насколько меньше вознаграждение, полученное торговым ботом, по сравнению с тем, каким бы оно могло быть, если бы бот покупал каждую акцию по 1, не использовав стратегий.. total regret оценивает, насколько хорошо алгоритм справляется с неопределенностью и принимает решения.

3 Датасет

Набор данных о ценах 249 российских акций. Датасет содержит исторические цены открытия, максимума, минимума, закрытия и объема акций, торгуемых на финансовых рынках Российской Биржи. Важной для нас переменной является цена закрытия (close). Взяты почасовые данные для 234 акций в период с 2023-06-01 по 2024-08-27 (остальные не торговались в 2023 и 2024 году).

	open	high	low	close	volume
datetime					
2023-06-01 10:00:00	2328.0	2356.0	2318.0	2343.6	143039.0
2023-06-01 11:00:00	2343.2	2348.0	2333.2	2333.2	48477.0
2023-06-01 12:00:00	2333.2	2334.2	2304.8	2311.4	86184.0
2023-06-01 13:00:00	2313.2	2336.4	2294.8	2330.0	104030.0
2023-06-01 14:00:00	2329.2	2333.6	2316.0	2326.8	33793.0
...	...	...	...	...	...
2024-08-27 19:00:00	4092.2	4096.4	4071.2	4071.2	9576.0
2024-08-27 20:00:00	4092.2	4096.4	4071.2	4071.2	9576.0
2024-08-27 21:00:00	4092.2	4096.4	4071.2	4071.2	9576.0
2024-08-27 22:00:00	4092.2	4096.4	4071.2	4071.2	9576.0
2024-08-27 23:00:00	4092.2	4096.4	4071.2	4071.2	9576.0

4 Ход работы

4.1 Strategy

Класс Strategy - основа для реализации различных стратегий многоруких бандитов. Он задаёт общую структуру, которую наследуют конкретные стратегии. Он определяет переменные для хранения состояний и действий.

```
class Strategy:
    def __init__(self, n_arms: int, n_portfolio: int):
        self.n_arms = n_arms
        self.n_portfolio = n_portfolio
        self.n_iters = 0
        self.arms_states = np.zeros(n_arms)
        self.arms_actions = np.zeros(n_arms)

    def flush(self):
        self.n_iters = 0
        self.arms_states = np.zeros(self.n_arms)
        self.arms_actions = np.zeros(self.n_arms)

    def update_reward(self, arm: int, reward: float):
        self.n_iters += 1
        self.arms_states[arm] += reward
        self.arms_actions[arm] += 1

    def choose_arm(self):
        raise NotImplementedError
```

4.2 Thompson Sampling

Для выбора оптимального портфеля из акций реализован класс Thompson, основанный на методе бета-распределений: у каждой акции (или "руки") есть параметры успехов и неудач (alphas и betas), которые обновляются после каждой итерации. Модель модифицирует alpha и beta параметры распределения в Томпсоне, основываясь на штрафе - комиссии брокера. В beta-распределении участвуют последние 840 часов для каждой акции (60 последних дней торгов). Сделано для того, чтобы "Забывать"старые просадки или подъемы, но поднять важность последних колебаний, так как мы торгуем на короткой перспективе (час).Также добавлен параметр "Жадности"и скорости ее возрастания. Его идея в том, что если акции растут в цене медленно, то уверенно уменьшать штраф и постепенно приходить к решению купить медленно-растущую акцию.

```
class Thompson(Strategy):
    def __init__(self, n_arms: int, n_portfolio: int, penalty: float, greed:
        float, greed_tempo: float):
        super().__init__(n_arms, n_portfolio, penalty, greed, greed_tempo)
        self.alphas = {i: [1.0] for i in range(n_arms)}
        self.betas = {i: [-1.0] for i in range(n_arms)}
        self.greed = greed
        self.greedness = {i: greed for i in range(n_arms)}
        self.portfolio = []
        self.greed_tempo = greed_tempo
        self.penalty = penalty


    def choose_arm(self):
        sampled_values = {i: np.random.beta(
                                            sum(self.alphas[i][:840]),
                                            abs(sum(self.betas[i][:840]))
                                           )
                             for i in range(self.n_arms)}

        best_arms = sorted(sampled_values, key=sampled_values.get, reverse=
            True)[:self.n_portfolio]

        self.portfolio = best_arms
```

4.3 Транзакционные издержки

В коде предусмотрена система штрафов за объём покупок - транзакционные издержки, связанные с реальными рынками. Эти издержки влияют на обновление наград для акций, которые не попадают в портфель. То есть, для акций, которые не вошли в портфель, награда уменьшается на величину штрафа, пропорционального параметру жадности. Высокие транзакционные издержки снижают привлекательность частых изменений в портфеле.

```
return best_arms

    def update_reward(self, arm: int, reward: float):

         the received reward
        if arm in self.portfolio:
            if reward <= 0:
                self.betas[arm].insert(0,reward)
                self.alphas[arm].insert(0,1e-20 )
                self.greedness[arm] = self.greed
            else:
                self.alphas[arm].insert(0,reward)
                self.betas[arm].insert(0,-1e-20)
                self.greedness[arm] = self.greed
        else:
            reward -= self.penalty / self.greedness[arm]

            if reward  <= 0:
                self.betas[arm].insert(0,reward)
                self.alphas[arm].insert(0,1e-20 )
                if self.greedness[arm] > 1:
                    self.greedness[arm] -= self.greed_tempo
            else:
                self.alphas[arm].insert(0,reward )
                self.betas[arm].insert(0,-1e-20 )
                self.greedness[arm] += self.greed_tempo
```

4.4 Создание среды

Среда StockMarketEnv имитирует поведение рынка акций. Внутри среды загружаются данные по нескольким акциям, и для каждой из них рассчитывается доходность (в виде процентного изменения цен закрытия). Если в данных присутствуют пропуски, они заполняются предыдущими значениями. Также для удобства сравнения создаётся массив ожидаемых доходностей, который обновляется на каждой итерации.

```
class StockMarketEnv:
    def __init__(self, stock_files: list):
        self.stock_data = self.load_stock_data(stock_files)
        self.n_arms = len(self.stock_data)

    def load_stock_data(self, stock_files):
        data = {}
        for stock_file in stock_files:
            df = pd.read_csv(stock_file, parse_dates=['datetime'], index_col
               ='datetime')
            df = df.reindex(ind)
            df = df.fillna(method='ffill')
            df = df.sort_index()
            df = df.loc['2023-06-01 00:00:00':]
            if df['close'].isna().sum() == 0:
                df['return'] = df['close'].pct_change()
                data[stock_file] = df['return'].dropna()
        return data
```

4.5 Механизм многорукого бандита

Создан класс Bandit, который объединяет среду и стратегию. На каждой итерации стратегия выбирает портфель из лучших акций на основе текущих параметров. Для каждой выбранной акции извлекается доходность из среды. Стратегия обновляет свои параметры на основе полученных доходностей. Параллельно с этим считается regret.

```
class Bandit:

    def __init__(self, env: StockMarketEnv, strategy: Strategy):
        self.env = env
        self.strategy = strategy

    def action(self, inde):
        arm = self.strategy.choose_arm()
        reward = self.env.pull_arm(arm, inde)
        self.strategy.update_reward(arm, reward)
```

4.6 Запуск модели и визуализация результатов

Модель была запущена на всём доступном временном ряде, и на каждой итерации фиксировались total rewards, total regret.

Результаты визуализированы в виде двух графиков:
total regret
total rewards

```
def calculate_regret(env: StockMarketEnv, strategy: Strategy):
    strategy.flush()
    bandit = Bandit(env, strategy)
    regrets = []
    total_rewards = []
    for i in range(len(ind)-2):
        expected_returns = env.get_expected_returns(i)
```

```
        optimal_return = np.mean(expected_returns)
        selected_stocks = bandit.strategy.choose_arm()
        strat.append(selected_stocks)

        portfolio_return = np.mean([bandit.env.pull_arm(stock, i) for stock
            in selected_stocks])

        regrets.append(optimal_return-portfolio_return)

        total_rewards.append(portfolio_return)

        for stock in range(len(stock_files)):
            reward = bandit.env.pull_arm(stock, i)
            bandit.strategy.update_reward(stock, reward)

    return regrets, total_rewards
```

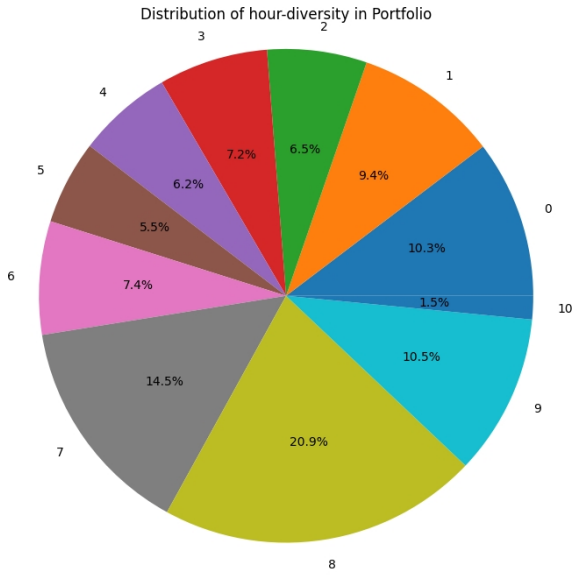
4.7 График распределения часов

Pie-chart, в котором отображено сколько акций в портфеле в этот час осталось из прошлого. Сделано это для того, чтобы чтобы отслеживать, что мы действительно заставляем алгоритм торговать реже. Без транзакционных издержек больше 50 процентов всех часов полностью обновляют свой состав акций, а с издержками около 10.

```
num_stocks_stayed = [sum(j in strat[i+1] for j in s) for i, s in enumerate(
    strat[:-2])]
stock_counts = []
categories = set(num_stocks_stayed)
for i in set(num_stocks_stayed):
    stock_counts.append(sum([j==i for j in num_stocks_stayed]))

plt.figure(figsize=(8, 8))
plt.pie(stock_counts, labels=categories, autopct='%1.1f%%', startangle=0)
plt.title('Distribution of hour-diversity in Portfolio')
plt.axis('equal')

plt.show()
```



5 Результаты

На графике видны заметные пики и впадины, которые говорят о том, что ситуация на рынке довольно изменчива, что очевидно для рынка акций, так как ее стоимость меняется каждую секунду. В процессе многих итераций мы видим, что regret в целом уменьшается, из чего можно сделать вывод о том, что алгоритм лучше адаптируется к условиям задачи по мере работы.

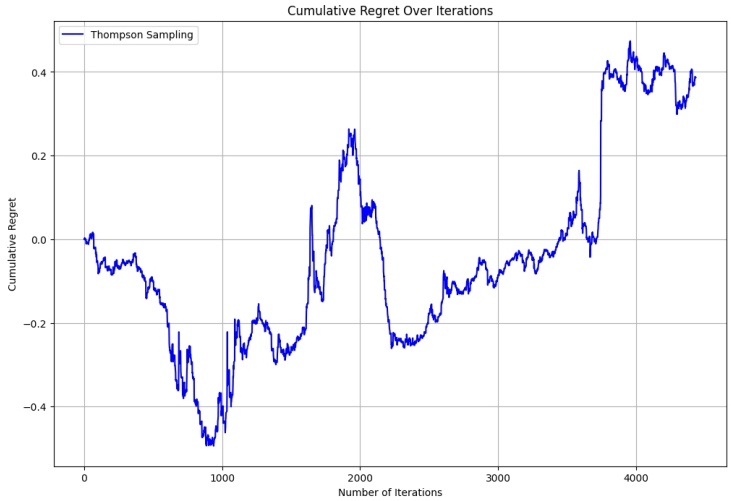
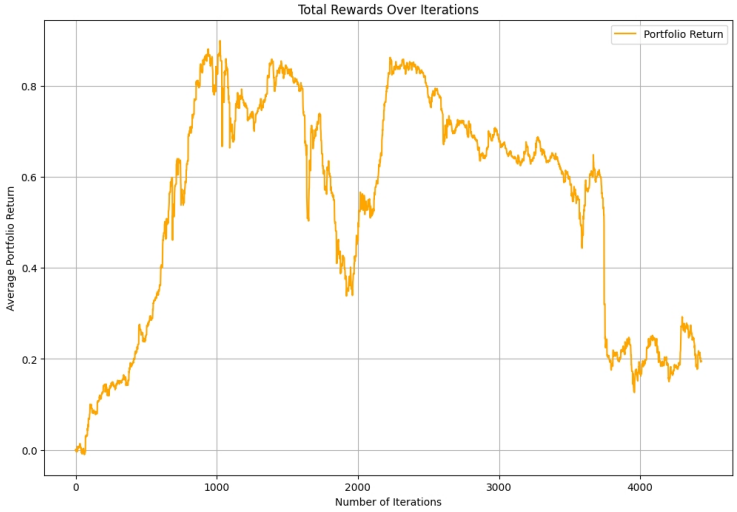


График показывает, что на протяжении всех итераций средняя доходность портфеля в целом увеличивается, это говорит о повышении эффективности алгоритма с течением времени. Колебания показывают, что наш алгоритм сталкивается с изменениями на рынке.



6 Распределение работ

6.1 Клевайчук Александр

- Рассчет Thompson Sampling Strategy
- Транзакционные издержки
- Рассчет среды StockMarketEnv
- Работа с датасетом
- График распределения часов

6.2 Михеева Екатерина

- Презентация
- Частичное написание теоретических основ и хода работы
- Поиск датасета
- Рассчет итоговых значений в коде

6.3 Молвинская Алина

- Оформление файла с описанием работы
- Частичное насписание теоретических основ и хода работы
- Поиск датасета
- Рассчет класса Strategy и класса Bandit в коде