ПРЕЗИДЕНТСКАЯ АКАДЕМИЯ

2024

Проект по курсу «Продвинутые методы машинного обучения»

Многорукие бандиты: Торговый бот

Выполнили:
студенты 4 курса
Клевайчук Александр,
Михеева Екатерина,
Молвинская Алина.

Москва, 2024

РАНХиГС

ПРЕЗИДЕНТСКАЯ АКАДЕМИЯ

Многорукие бандиты: торговый бот

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.

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

Математическая постановка

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.

Метрики

В качестве метрик в нашей задаче мы использовали:

• **total rewards** (Сумма всех полученных наград за время эксперимента)

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

• **cumulativer regret** (Потери из-за выбора не оптимальных рук)

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

ПРЕЗИДЕНТСКАЯ АКАДЕМИЯ

Датасеты и данные

Набор данных о ценах 234 российских акций. Датасет содержит исторические цены открытия, максимума, минимума, закрытия и объема акций, торгуемых на финансовых рынках Российской Биржи. Взяты почасовые данные закрытия в период с 01.06.2023 по 27.08.2024

datetime	open	high	low	close	volume
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

4437 rows × 5 columns

Ход работы

1. Создаём класс Strategy - основа для реализации различных стратегий многоруких бандитов.
2. Thompson Sampling
3. Добавляем транзакционные издержки
4. Создаём среду, имитирующую поведение рынка акций
5. Создаём класс Bandit, который объединяет среду и стратегию.
6. Запускаем модель

Метод 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).

Также добавлен параметр "Жадности» и скорости ее возрастания.

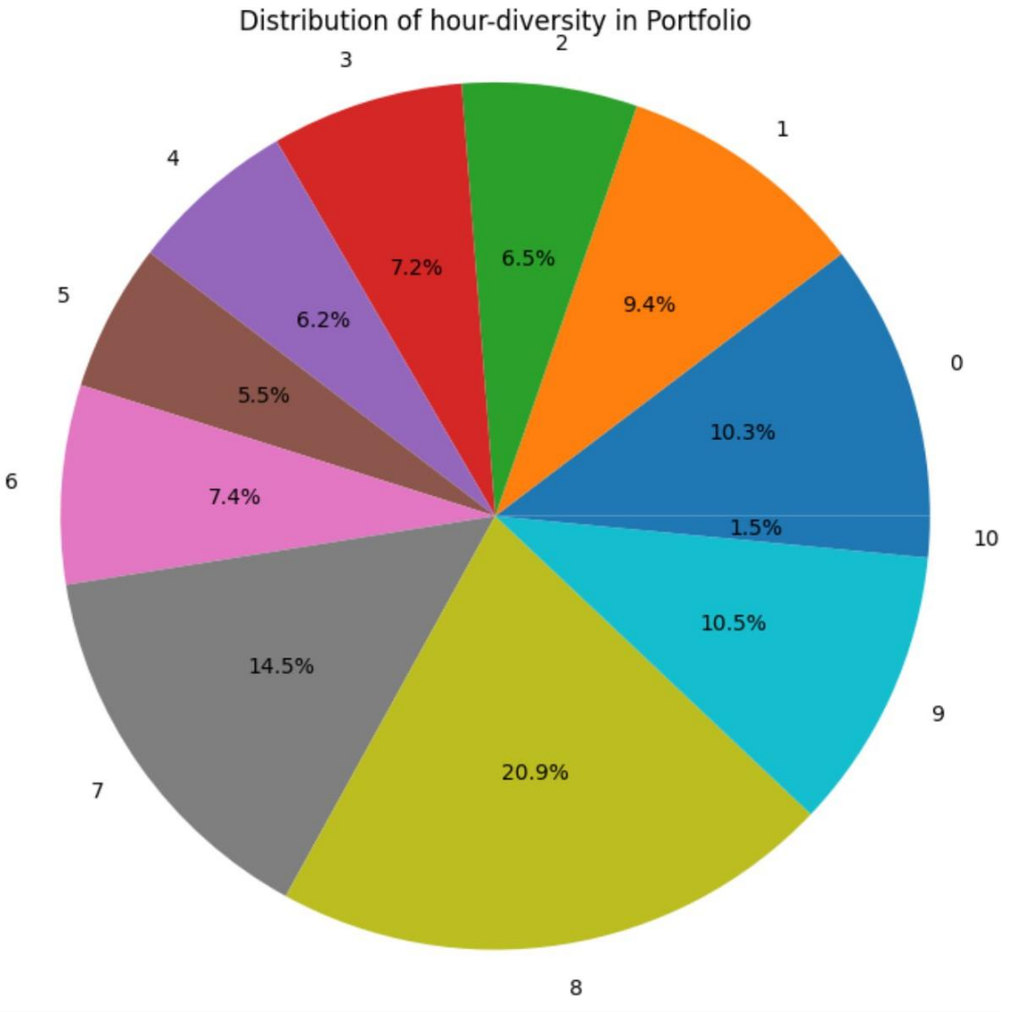
Помимо этого предусмотрена система штрафов за объём покупок - транзакционные издержки, связанные с реальными рынками.

Ход работы

1. Создаём класс Strategy - основа для реализации различных стратегий многоруких бандитов.
2. Thompson Sampling
3. Добавляем транзакционные издержки и параметр жадности
4. Создаём среду, имитирующую поведение рынка акций
5. Создаём класс Bandit, который объединяет среду и стратегию.
6. Запускаем модель

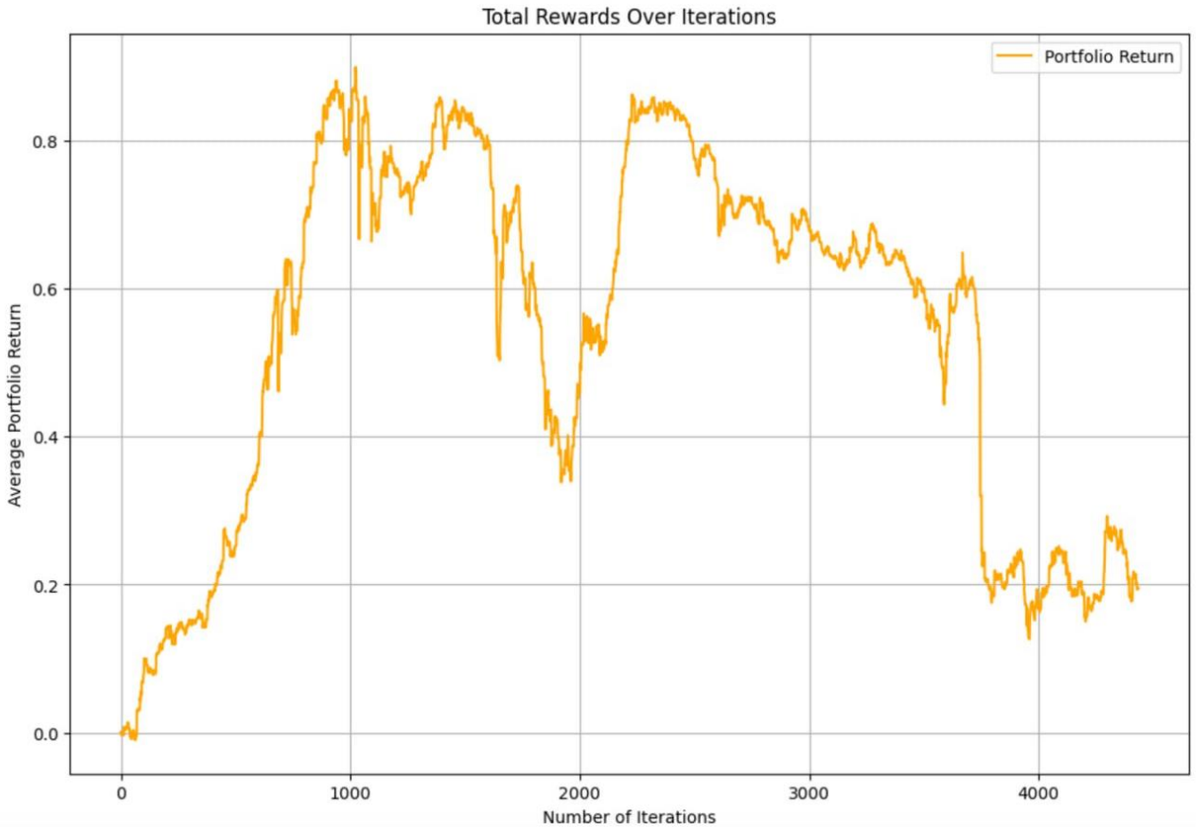
ПРЕЗИДЕНТСКАЯ АКАДЕМИЯ

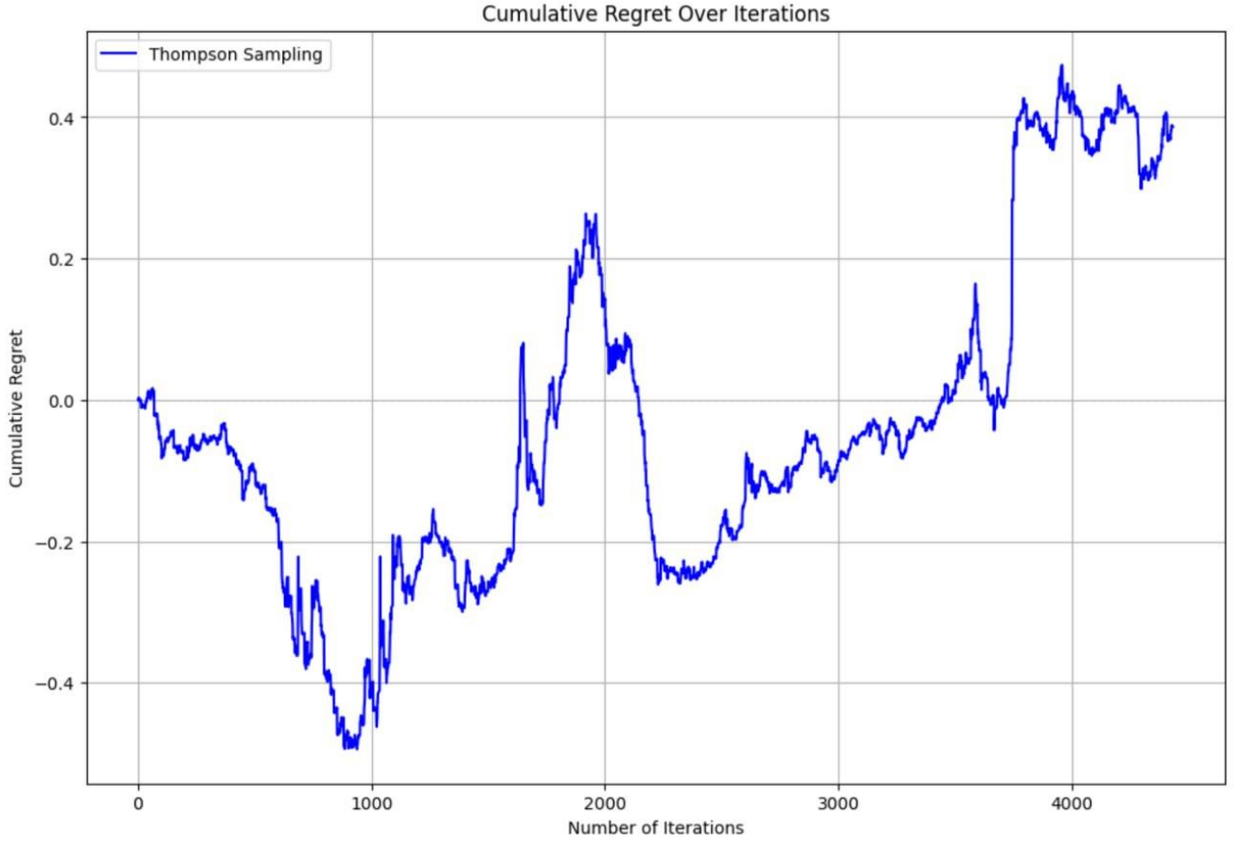
Результаты



ПРЕЗИДЕНТСКАЯ АКАДЕМИЯ

Результаты





ПРЕЗИДЕНТСКАЯ АКАДЕМИЯ

СПАСИБО ЗА ВНИМАНИЕ!

Москва, 2024

2024

РАНХиГС

```python
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
```

```python
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
```

```
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
```

```python
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
```

```python
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)
```

```python
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
```