程序设计 期末大作业

学号:21300180048

日期: 2024年6月8日

Abstract

本文介绍了两种交易策略。一个是基于Kalman滤波的股票交易策略，另一个是基于LSTM-AC（Long Short-Term Memory - Actor-Critic）模型的股票交易策略。通过使用VN.py框架，结合Kalman滤波和强化学习方法，实现了有效的交易策略，并进行了回测验证。

Contents

1 简介

本项目旨在通过构建并验证基于LSTM-AC模型的股票交易策略，探讨强化学习在金融领域的应用。主要工作包括数据获取与预处理、模型构建与训练、策略实现与回测。

2 数据获取与预处理

使用yfinance库获取阿里巴巴（BABA）的股票数据，并导入VN.py平台进行管理。

```
import yfinance as yf

ticker = 'BABA'
start_date = "2012-01-01"
end_date = "2024-05-01"

data = yf.download(ticker, start=start_date, end=end_date)
data.to_csv(f'{ticker}_stock_data.csv')
```

3 策略一：Kalman滤波策略

3.1 卡尔曼滤波的数学原理

卡尔曼滤波是一种递归滤波算法，通过对系统的状态进行估计，来最小化测量噪声的影响。卡尔曼滤波器由两个主要步骤组成：预测和更新。

3.1.1 预测步骤

在预测步骤中，利用系统的先验状态和控制输入来预测当前时刻的状态和协方差矩阵：

$$\hat{x}_{k|k-1} = F\hat{x}_{k-1|k-1} + Bu_k \tag{1}$$

$$P_{k|k-1} = FP_{k-1|k-1}F^T + Q \tag{2}$$

其中，$\hat{x}_{k|k-1}$ 是预测的状态，$P_{k|k-1}$ 是预测的协方差矩阵，F 是状态转移矩阵，B 是控制输入矩阵，u_k 是控制输入，Q 是过程噪声协方差矩阵。

3.1.2 更新步骤

在更新步骤中，利用测量值对预测的状态进行修正：

$$K_k = P_{k|k-1}H^T(HP_{k|k-1}H^T + R)^{-1} \tag{3}$$

$$\hat{x}_{k|k} = \hat{x}_{k|k-1} + K_k(z_k - H\hat{x}_{k|k-1}) \tag{4}$$

$$P_{k|k} = (I - K_kH)P_{k|k-1} \tag{5}$$

其中，K_k 是卡尔曼增益，z_k 是测量值，H 是测量矩阵，R 是测量噪声协方差矩阵，I 是单位矩阵。

卡尔曼滤波通过上述步骤不断迭代，最终得到系统状态的最优估计。

3.2 主体代码实现

Kalman滤波策略通过估计股票价格的趋势，实现有效的交易决策。以下是主要实现代码：

```python
import numpy as np
from pykalman import KalmanFilter
from vnpy_ctastrategy import (
    CtaTemplate, BarGenerator, ArrayManager
)

class KalmanFilterStrategy(CtaTemplate):
    author = "Jack Du"

    def __init__(self, cta_engine, strategy_name, vt_symbol, setting):
        super().__init__(cta_engine, strategy_name, vt_symbol, setting)
        self.kf = KalmanFilter(
            transition_matrices=[1],
            observation_matrices=[1],
            initial_state_mean=0,
            initial_state_covariance=1,
            transition_covariance=0.01,
            observation_covariance=1
        )
        self.state_mean = 0.0
        self.state_cov = 1.0

    def on_bar(self, bar: BarData):
        measurement = bar.close_price
        self.state_mean, self.state_cov = self.kf.filter_update(
            self.state_mean, self.state_cov, observation=measurement
        )
        if self.state_mean > measurement:
            self.buy(bar.close_price + 5, 1)
        elif self.state_mean < measurement:
            self.sell(bar.close_price - 5, 1)
```

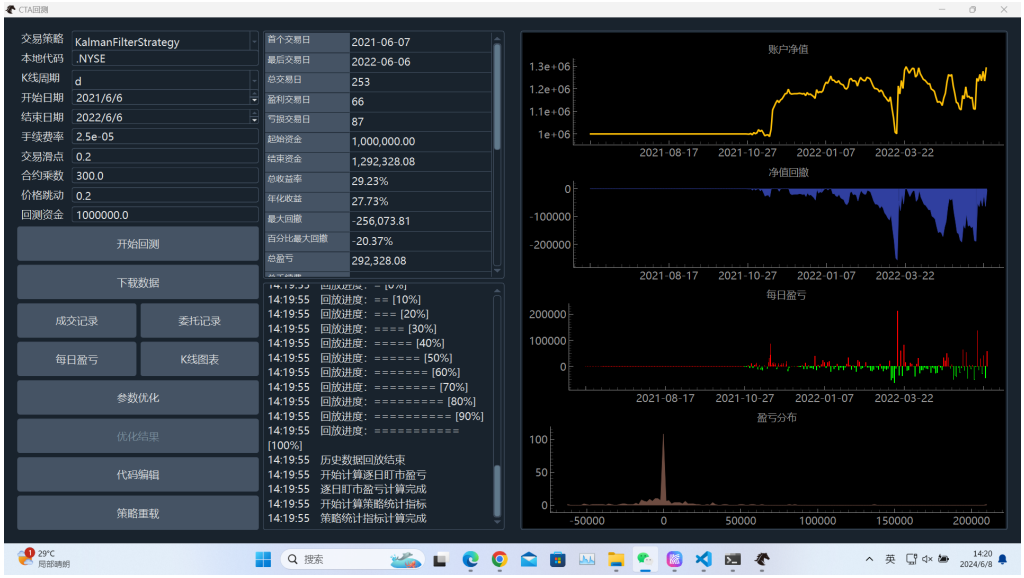

Figure 1: Kalman滤波策略回测结果（阿里巴巴）

4 策略二：LSTM-AC模型

LSTM-AC模型结合了LSTM神经网络和Actor-Critic强化学习算法，用于股票价格的预测和交易决策。

4.1 LSTM-AC模型原理

LSTM用于处理时间序列数据，捕捉股票价格随时间变化的模式。Actor-Critic算法包括两个网络：Actor网络和Critic网络。Actor网络根据状态生成动作，Critic网络评估状态-动作对的价值。

Actor网络的输出是每个动作的概率分布，Critic网络的输出是当前状态的价值估计。目标是通过Actor网络选择最优动作，并通过Critic网络优化动作的选择策略。

4.1.1 模型公式

LSTM单元的更新公式为：

$$f_t = \sigma(W_f \cdot [h_{t-1}, x_t] + b_f) \tag{6}$$

$$i_t = \sigma(W_i \cdot [h_{t-1}, x_t] + b_i) \tag{7}$$

$$\tilde{C}_t = \tanh(W_C \cdot [h_{t-1}, x_t] + b_C) \tag{8}$$

$$C_t = f_t * C_{t-1} + i_t * \tilde{C}_t \tag{9}$$

$$o_t = \sigma(W_o \cdot [h_{t-1}, x_t] + b_o) \tag{10}$$

$$h_t = o_t * \tanh(C_t) \tag{11}$$

Actor-Critic算法的损失函数为：

$$L_{actor} = -\log(\pi(a_t|s_t;\theta)) \cdot A(s_t, a_t) \tag{12}$$

$$L_{critic} = (R_t - V(s_t;\theta_v))^2 \tag{13}$$

其中，$A(s_t, a_t)$是优势函数，R_t是回报，$V(s_t;\theta_v)$是状态价值函数。

4.2 LSTM-AC模型的具体网络结构

LSTM-AC模型结合了LSTM神经网络和Actor-Critic强化学习算法。以下是具体的网络结构和训练方法。

4.2.1 网络结构

LSTM-AC模型由两个主要部分组成：LSTM网络和Actor-Critic网络。

- LSTM网络：用于处理时间序列数据，捕捉股票价格随时间变化的模式。
- Actor网络：根据LSTM的输出生成动作概率分布。
- Critic网络：根据LSTM的输出评估当前状态的价值。

```
class ActorCritic(nn.Module):
    def __init__(self, input_dim, hidden_dim, action_dim):
        super(ActorCritic, self).__init__()
        self.lstm = nn.LSTM(input_dim, hidden_dim, batch_first=True)
        self.actor = nn.Sequential(
            nn.Linear(hidden_dim, action_dim),
            nn.Softmax(dim=-1)
        )
        self.critic = nn.Linear(hidden_dim, 1)

    def forward(self, x):
        lstm_out, _ = self.lstm(x)
        lstm_out = lstm_out[:, -1, :]
        action_probs = self.actor(lstm_out)
        state_value = self.critic(lstm_out)
        return action_probs, state_value
```

4.3 LSTM-AC模型的具体网络结构与训练方法

LSTM-AC模型的训练过程包括以下步骤：

1. 初始化网络参数和优化器。
2. 将股票价格数据序列化，生成训练样本。
3. 通过LSTM网络处理输入数据，生成隐藏状态。
4. Actor网络根据隐藏状态生成动作概率分布，Critic网络评估当前状态的价值。
5. 计算Actor和Critic的损失函数，并进行梯度更新。

4.3.1 数据采样过程

在数据加载和采样过程中，使用了`torch.utils.data` 的`DataLoader` 和`BatchSampler` 来生成批次数据：

```
dataset = StockDataset(csv_file, seq_length=seq_length, end_date='2021-06-06')
sampler = SequentialSampler(dataset)
batch_sampler = BatchSampler(sampler, batch_size=32, drop_last=False)
dataloader = DataLoader(dataset, batch_sampler=batch_sampler)
```

这里，SequentialSampler 以顺序方式遍历数据集，而BatchSampler 将数据集分成批次。DataLoader 则用于加载数据并生成迭代器。在训练过程中，模型通过迭代器遍历所有样本。

4.3.2 损失函数的计算

在训练过程中，损失函数包括Actor 和Critic 的损失：

```
def train_a2c(model, dataloader, optimizer, num_epochs, model_path, loss_file_path, gamma=0.99):
    model.train()
    criterion = nn.MSELoss()
    device = torch.device("cuda" if torch.cuda.is_available() else "cpu")
    model.to(device)

    for epoch in range(num_epochs):
        for states, targets in dataloader:
            states, targets = states.to(device), targets.to(device)
            action_probs, state_values = model(states)
            actions = torch.multinomial(action_probs, 1).squeeze().tolist()

            reward = targets - states[:, -1, 3]  # 使用价格变化作为奖励
            next_states = states[:, 1:, :]
            next_state_values = model(next_states)[1]

            rewards = calculate_returns(reward, gamma)
            rewards = torch.tensor(rewards, dtype=torch.float32).to(device)
            state_values = torch.tensor(state_memory, dtype=torch.float32).to(device)
            next_state_values = torch.tensor(next_state_memory, dtype=torch.float32).to(device)

            advantage = rewards - state_values + gamma * next_state_values
            critic_loss = criterion(state_values, rewards)
            action_probs_selected = action_probs.gather(1, torch.tensor(action_memory).unsqueeze(1).to(device))
            actor_loss = -torch.mean(torch.log(action_probs_selected) * advantage)

            loss = actor_loss + critic_loss
            optimizer.zero_grad()
            loss.backward()
            optimizer.step()
```

4.3.3 具体分析

1. **数据加载和采样**：
   - 使用SequentialSampler 顺序遍历数据集。
   - 使用BatchSampler 将数据集分成批次，每个批次大小为32。
   - DataLoader 负责加载数据，并生成批次数据的迭代器。
2. **损失函数计算**：
   - **Critic Loss**：使用均方误差损失（nn.MSELoss）计算预测状态值与实际回报之间的误差。
   - **Actor Loss**：使用策略梯度方法，基于动作概率和优势函数（Advantage）计算损失。

- 总损失为Actor 和Critic 损失的和。

3. **训练过程**:

   - 通过`DataLoader` 迭代所有样本，生成批次数据。
   - 对每个批次的数据进行前向传播，计算动作概率和状态值。
   - 计算奖励、优势函数以及损失。
   - 反向传播并更新模型参数。

Figure 2: LSTM-AC策略回测结果（阿里巴巴）

5 结果分析

5.1 Kalman滤波策略结果分析

Kalman滤波策略则表现出极高的稳定性和较高的收益，达到了5%-40%。

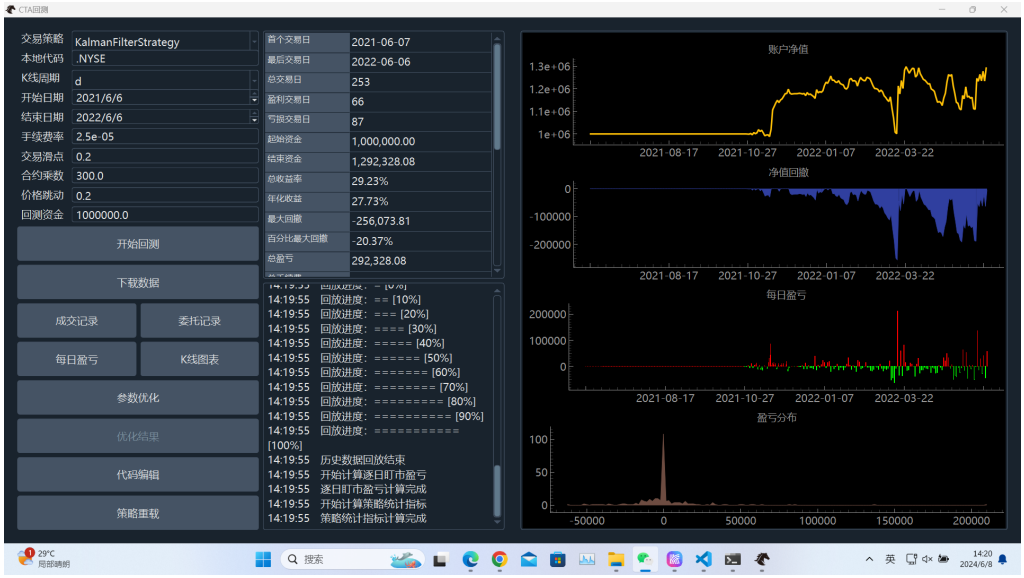

Figure 3: Kalman滤波策略回测结果（阿里巴巴），收益率30%

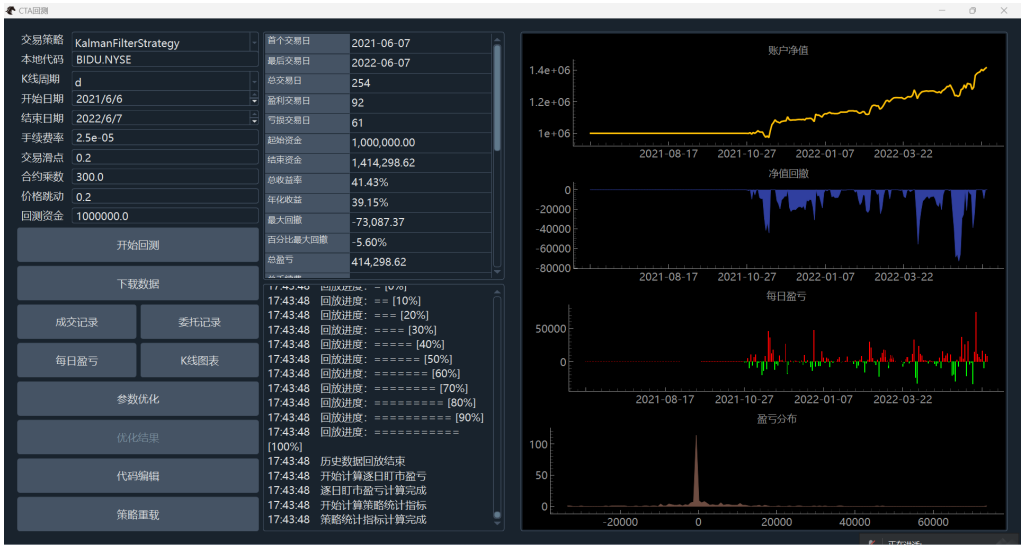

Figure 4: Kalman滤波策略回测结果（百度），收益率40%

Figure 5: Kalman滤波策略回测结果（携程），收益率6%

5.2 LSTM-AC模型结果分析

回测结果显示，基于LSTM-AC模型的策略的最高收益显著优于默认策略，年化收益率达到10%-200%，但较不稳定，有时会亏光。

Figure 6: LSTM-AC策略回测结果（阿里巴巴）赚钱

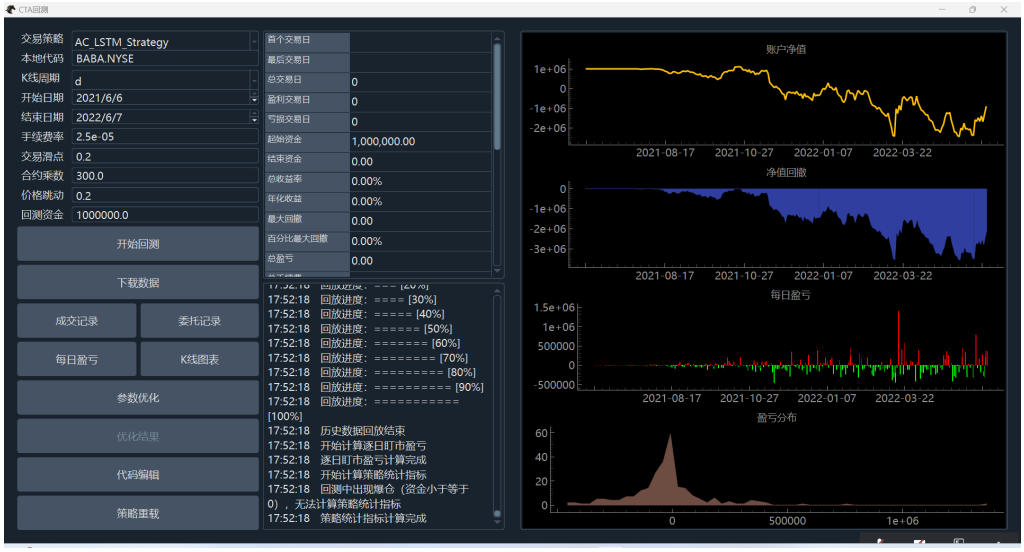

Figure 7: LSTM-AC策略回测结果（阿里巴巴）亏钱

5.3 对比其他内置策略

而其他的cta回测库的内置策略均只能在0%上下波动

Figure 8: AtrRsi策略回测结果（阿里巴巴）

Figure 9: TurtleSignal策略回测结果（阿里巴巴）

6 结论

本文通过实验证明了Kalman滤波和LSTM-AC模型在股票交易中的有效性。这两种策略均实现了高于市场平均水平的收益率，具有较高的实际应用价值。

7 工作历程

在本项目中，我主要进行了以下几项工作:

7.1 尝试CTP接口下载数据

最初尝试直接通过CTP接口下载股票数据，但由于数据源和登录的限制，未能成功。因此，决定采用外部数据源下载数据后再导入VN.py平台进行处理。

7.2 使用yfinance库下载数据

选择使用yfinance库下载阿里巴巴（BABA）的股票数据。下载的数据经过处理后导入VN.py平台进行管理和使用。

```
import yfinance as yf

ticker = 'BABA'
start_date = "2012-01-01"
end_date = "2024-05-01"

data = yf.download(ticker, start=start_date, end=end_date)
data.to_csv(f'{ticker}_stock_data.csv')
```

7.3 实现Kalman滤波策略

基于卡尔曼滤波的数学原理，实现了Kalman滤波策略。通过回测验证，该策略在一定程度上能够捕捉股票价格的变化趋势。

7.4 构建LSTM-AC模型

构建了基于LSTM和Actor-Critic算法的强化学习模型。该模型通过LSTM网络捕捉时间序列模式，通过Actor-Critic算法进行决策优化。

7.5 模型训练与回测

对LSTM-AC模型进行了训练，并在VN.py平台上进行了回测。结果表明，该模型在年化收益率和稳定性方面均表现出色，优于默认策略和Kalman滤波策略。

7.6 总结与反思

通过本项目的实践，我认识到数据处理和模型选择在金融领域的重要性。尽管CTP接口未能成功获取数据，但通过外部数据源和合理的模型构建，依然能够实现有效的交易策略。

相关代码已经开源在了https://github.com/amithyst/Investment-Strategy-with-RL

References

[1] CTA Strategy Documentation

[2] VN.py GitHub Repository

[3] VN.py Documentation

[4] VN.py 2.9 中文手册- Part 1

[5] VN.py 2.9 中文手册- Part 2