Awesome-AI-Trader

Backgrounds

The traditional strategy making in quantitative trading only involves in static monitoring, quick data fetching.

Some traders will like to get some first-hand information, analyze the signal, and follow this signal. This strategy is indeed working: when I follow the large traders in backtesting using CoT data (a delayed data of commitment of traders), I got handsome return. However, this kind of intonation is **difficult to get and difficult to analyze**.

Some trader will use some **Technical analysis**, and make some strategy taking them into account. And during a long period of backtesting, one strategy will not always profit, as there are bearish, bullish, or neither. How to recognize one trend regardless of the current status?

Human will think too much somtimes because they maybe **affected by temptation**. That's why algo trading comes out. Some popular trading strategy includes SMACross, BOLLlilnger bands, RSI Strategy.

But as far as I know, there are **few AIs shown on the market**.

Project design

This AI-based trader app will analyze the serialized price data for one stock.

It will firstly extract all Technical analysis from the price data. If there are some other observed data like from some where, then can be also be added into the data feeds.

Data preprocess

To make the AI training for efficient, we need to keep the useful data with information. Before straining start, a data proprocess will be executed:

1. Drop the columns with correlation 1, for there are duplicated or function-ralated columns. Then, it will drop the columns with little correlation.
2. Use MIC to capture the columns with non-linear, asyncronized relation. Asyncronized means that there will be some delayed but useful data. This will again drop some columns.
3. Use a Random forest as a classifier, to random permutate through all candidate columns, selecting the best columns. This will again drop some columns.

runAI.py | extractor_v0.py M | readme.md M | selector_v0.py | documer ...

data_feature_selection > selector_v0.py > ...

```python
import pandas as pd
from sklearn.ensemble import RandomForestRegressor
from sklearn.model_selection import train_test_split
import numpy as np

# Assuming that df is your DataFrame and it has been loaded properly

def select_feature(df: pd.DataFrame, test_size=0.2,m = 48):
    """Select feature from dataframe with extracted columns.
    It will calculated and preserve columns which are most correlated  with Clos

    Args:
        @df (pd.DataFrame): dataframe to be seleced. It should at least have dat
        @m (int): top m columns you want to preserve.
    """

    # Convert 'Date' to datetime if it's not
    df['date'] = pd.to_datetime(df['date'])

    # Set 'Date' as the index
    df.set_index('date', inplace=True)

    # Drop the non-numerical columns
    df = df.select_dtypes(include=[np.number])
```

extractor_v0.py M

data_feature_extraction > extractor_v0.py > ...

```python
import os
# Load datas
def extract_data(datasoursepath, finalextracteddatapath, nCorrTop=50, nMICTop=20
    """
    Given price data, extract tas, and return top n correlated cols for feature
    This function will also manually drop some unusefull columns.
    """
    # if os.path.exists(finalextracteddatapath):
    #     print("ouput file already exsist, just read this file")
    #     return pd.read_csv(finalextracteddatapath)
    print("extracting_data is working on:", os.getcwd())
    df = pd.read_csv(datasoursepath)
    # Clean NaN values
```

Model: LSTM with positional encoding

1. LSTM makes the AI to memorized a period of history.
2. Positional encoding makes the AI know what to ignore/remember within a time period, even between input features.

```
<class 'models.LSTMWithAttention.StockPredictor3'>
model: StockPredictor3(
  (lstm): LSTM(9, 128, batch_first=True, bidirectional=True)
  (attention): MultiheadAttention(
    (attention): MultiheadAttention(
      (out_proj): NonDynamicallyQuantizableLinear(in_features=256, out_features=256, bias=True)
    )
  )
  (dropout): Dropout(p=0.2, inplace=False)
  (fc1): Linear(in_features=256, out_features=8, bias=True)
)
```

Training:

1. **Train-test data**: In about 10 years of data, AI will always train itself on the 95% percent of data, and test through the rest 5%.
2. **Self tuning:** The AI will tune itself by early stopping, to avoid overfitting.

```
)
Epoch 0, Loss: 0.01285730051024009
Epoch 10, Loss: 0.006747777679484865
Epoch 20, Loss: 0.004161861600583638
Epoch 30, Loss: 0.003564995340047424
Epoch 40, Loss: 0.0031586417143586575
Execution of job "multi (trigger: cron[second='30'], next run at: 2023-07-20 05:06:30 UTC)" ski
Epoch 50, Loss: 0.0031928932180288754
Epoch 60, Loss: 0.003107985684470027
Early stopping after 62 epochs.
Finished Training
Evaluation on test data:                 Total loss: 0.07336302846670151
pred:  (108, 8)
Accuracy for prediction length 1: 52.34%
Accuracy for prediction length 2: 49.06%
Accuracy for prediction length 3: 45.71%
Accuracy for prediction length 4: 53.85%
Accuracy for prediction length 5: 51.46%
Accuracy for prediction length 6: 50.00%
Accuracy for prediction length 7: 54.46%
Training finised, saving output file to.. /app/outputsByAI/^DJI_2023-07-20_processed.csv
```

Backtesting

Backtrader is a powerful and reliable tool for backtesting.

A backtrader module has been integrated into the app, and be used given an AI outpupt.

Each run will produce a detailed trading logs during backtesting period.

```
Running backtest for data stroed at: /app/outputsByAI/^IXIC_2023-07-20_processed.csv
init
self.data0:
('close', 'low', 'high', 'open', 'volume', 'openinterest', 'datetime', 'Col_1', 'Col_2
Backtesting starts...
2023-02-14, SELL EXECUTED, Price: 11807.02, Cost: -89358.43, Comm 26.81
2023-02-17, BUY EXECUTED, Price: 11778.68, Cost: -89358.43, Comm 26.74
2023-02-17, BUY EXECUTED, Price: 11778.68, Cost: 89143.94, Comm 26.74
2023-02-17, OPERATION PROFIT, GROSS 214.50, NET 160.95
2023-02-28, SELL EXECUTED, Price: 11449.90, Cost: 89143.94, Comm 26.00
2023-02-28, SELL EXECUTED, Price: 11449.90, Cost: -86655.70, Comm 26.00
2023-02-28, OPERATION PROFIT, GROSS -2488.24, NET -2540.98
2023-03-13, BUY EXECUTED, Price: 11042.56, Cost: -86655.70, Comm 25.07
2023-03-13, BUY EXECUTED, Price: 11042.56, Cost: 83572.85, Comm 25.07
2023-03-13, OPERATION PROFIT, GROSS 3082.85, NET 3031.79
2023-03-16, SELL EXECUTED, Price: 11383.73, Cost: 83572.85, Comm 25.85
2023-03-16, SELL EXECUTED, Price: 11383.73, Cost: -86154.89, Comm 25.85
2023-03-16, OPERATION PROFIT, GROSS 2582.04, NET 2531.12
2023-03-27, BUY EXECUTED, Price: 11869.73, Cost: -86154.89, Comm 26.95
2023-03-27, BUY EXECUTED, Price: 11869.73, Cost: 89833.02, Comm 26.95
2023-03-27, OPERATION PROFIT, GROSS -3678.13, NET -3730.93
2023-05-01, SELL EXECUTED, Price: 12208.83, Cost: 89833.02, Comm 27.72
2023-05-01, SELL EXECUTED, Price: 12208.83, Cost: -92399.43, Comm 27.72
2023-05-01, OPERATION PROFIT, GROSS 2566.41, NET 2511.74
2023-05-04, BUY EXECUTED, Price: 11998.54, Cost: -92399.43, Comm 27.24
2023-05-04, BUY EXECUTED, Price: 11998.54, Cost: 90807.91, Comm 27.24
2023-05-04, OPERATION PROFIT, GROSS 1591.52, NET 1536.56
Backtesting ends.
--------------- AnnualReturn -----------------
OrderedDict([(2023, 0.2133012851722189)])
```

Result: As shown in screenshot, this AI gives a strategy of **21.3 return, in year 2023**. It will be larger for one year.

Note:

The key point of this project is making test period is short, because AI need to train on latest data to get most of the latest trend.

Server-side development

1. **Deploy:** This AI is wrapped into a flask app, and can be developed onto a server with GPU ,and runs forever.
2. **Output:** This application will save its output (original price data, trained model, backtest result) into some folder for retrieve. If needed, they can be also stored into some database, and be forwarded to some frontend to user interface.
3. **Job scheduling:**
   For now, the app can iteratively train on and run backtest on a list of tickers, saving the output.