[04/11/2025]

Quantive Strategy Model

Multi-indicator forecasting and dy namic optimization on stock data

Author: Yuzhi Yao

Institution: Covoloo

Project Name: stock price forecasting

Date: 04/11/2025

1. Abstract:

In this project I first decide to choose the “Return-seeking” as the main purpose of the analysis and then base on the purpose I read some paper about the technical analysis method and there difference, and finally decide to choose

- RSI(Relative Strength Index),
- EMA(exponential moving average),
- PLR(Piece wise linear regression)

As may three of my indicators which EMA help me to identify the potential bought/sold point and the trend, RSI identify the overbought/oversold condition and use PLR to finally decide the direction of the trend. After setting the indicator I pick the **ATR(Average True Range) as my dynamic exiting rule** which allow user take profit or stop loss dynamically. Finally base on the different combination of the indicators parameters I optimized the parameter by **running the back test** on the historical data to validate the result and **used grid search** to find the combination which have the most wining rate. In the end, I **use the tkinter to create a user_friendly interface** which allow user to directly pick the different stock and period and see the performance of the Strategy and also save the data .

2. Introduction:

2.1 Target Type Analysis:

Return-Seeking Type: Suitable for investors with higher risk tolerance, adopting aggressive strategies that offer high return potential alongside elevated volatility.

Risk-Controlled Type: Ideal for conservative investors, prioritizing capital preservation while pursuing stable but modest returns.

Arbitrage Type: Geared toward professional investors or institutions, capitalizing on market inefficiencies for low-risk, low-return opportunities.

To achieve short-term forecasting objectives, we focus on the return-seeking type, primarily utilizing trend and momentum indicators to generate trading signals.

2.2 **Data Fetching**

Initially, Investing.com was considered as a potential data source. However, due to several limitations, this approach was abandoned in favor of alternative data providers. The primary challenges associated with Investing.com included:

the necessity of API authentication for data retrieval, and the cumbersome extraction process required to obtain historical data for specific time periods, despite the platform's user-friendly interface for data visualization. Consequently, stock price data from Yahoo Finance was selected as the primary data source due to its straightforward accessibility and ease of data fetching.

2.3 Price Action Analysis Methodology

Price action analysis serves as a versatile technical approach capable of generating trading signals across various market conditions.

2.3.1 Relative Strength Index (RSI)

The RSI is a momentum oscillator that measures the magnitude of recent price changes to evaluate overbought or oversold conditions. Its calculation involves:

Computing average gains and average losses over a specified period (typically 14 days).

Deriving the Relative Strength (RS) as:

$$RS = \frac{\text{Average Gain}}{\text{Average Loss}}$$

Converting RS into the RSI value:

$$RSI = 100 - \frac{100}{1 + RS}$$

```
def _calculate_rsi(self, data):  1 usage
    delta = data['Close'].diff()
    gain = delta.where(delta > 0, 0)
    loss = -delta.where(delta < 0, 0)

    avg_gain = gain.rolling(self.params['rsi_window']).mean()
    avg_loss = loss.rolling(self.params['rsi_window']).mean()
    rs = avg_gain / avg_loss
    return 100 - (100 / (1 + rs))
```

A reading above 70 suggests overbought conditions, [7] while a value below 30 indicates oversold conditions.

RSI is straightforward to compute and widely adopted, making it a benchmark for momentum analysis. An RSI persistently above 50 implies sustained bullish momentum, while values below 50 suggest bearish dominance.

As a historical-price-dependent indicator, RSI reacts slower than price action, missing early reversal signals.

While RSI shows overbought/oversold zones, it does not pinpoint optimal trade execution timing.

Ineffectiveness in Trending Markets: In strong trends, RSI can remain overbought/oversold for extended periods, leading to false reversals.

Due to these constraints, RSI is best used alongside trend-following indicators (e.g., moving averages) to filter false signals.

2.3.2 Exponential Moving Average

The Exponential Moving Average (EMA) is a weighted moving average that places greater emphasis on recent price data, making it more responsive to price changes than the Simple Moving Average (SMA). [1] The EMA is calculated using a smoothing factor $\alpha = \frac{2}{N+1}$ [2] where N being the selected period length). The formula for EMA is recursively defined as:

```
def _calculate_ema(self, data, window):  2 usages
    return data['Close'].ewm(span=window, adjust=False).mean()
```

$$EMA_t = \alpha \cdot P_t + (1 - \alpha) \cdot EMA_{t-1}$$

Where EMA_t is the current EMA values and P_t is the current price and EMA_t-1 is the previous EMA value. Traders use the EMA to identify trends (a rising EMA signals an uptrend, while a declining EMA suggests a downtrend) and dynamic support/resistance levels.
However, since it is a lagging indicator, it may produce false signals in choppy markets, and its effectiveness depends on the chosen period length (e.g., 10-day vs. 50-day EMA). Crossovers between short-term and long-term EMA (e.g., 9-day and 21-day) are commonly used to generate buy/sell signals[3].

2.3.3 Piece wise Linear Regression

PLR is a statistical method used to model data by fitting multiple linear segments, allowing for different slopes and intercepts across distinct intervals of the independent variable. This approach is particularly useful when the relationship between variables exhibits structural breaks or varying trends in different regimes. The model can be expressed as:

$$y = \begin{cases} \beta_{0,1} + \beta_{1,1}x + \epsilon & \text{if } x \leq c_1 \\ \beta_{0,2} + \beta_{1,2}x + \epsilon & \text{if } c_1 < x \leq c_2 \\ \vdots & \\ \beta_{0,k} + \beta_{1,k}x + \epsilon & \text{if } x > c_{k-1} \end{cases}$$

In my model, piece_wise linear regression (PLR) is employed to identify potential trend reversals and regime shifts in price movements.

```
def _calculate_plr_slope(self,data, window):  1 usage
    x = np.arange(window)
    slopes = []
    for i in range(len(data["Close"]) - window + 1):
        y = data["Close"].iloc[i:i + window]
        # weighted plr
        weights = np.linspace(1, 3, window)
        slope = np.polyfit(x, y, 1, w=weights)[0]
        slopes.append(slope)

    slopes = ([np.nan] * (window - 1) + slopes)
    return pd.Series(slopes, index=data.index)
```

The PLR divides the price series into multiple linear segments, each with its own slope and intercept, [4] allowing for adaptive trend detection. A key signal in this approach is the slope of the most recent PLR segment: if the slope is positive (> 0), it suggests a potential uptrend, indicating that prices are exhibiting upward momentum and may present a buying opportunity. Conversely, if the slope is negative (< 0), it signals a potential downtrend, implying weakening prices and a possible sell signal.

2.4 **ATR for Dynamic Exit Rules**

The Average True Range (ATR) is a volatility indicator that measures market volatility by analyzing the true price range over a specified period. Unlike standard range calculations, ATR accounts for gaps and limit moves by incorporating:

- Current high vs. low
- Previous close vs. current high
- Previous close vs. current low

The True Range (TR) for each period is calculated as:

$$TR_t = \max\left(\text{High}_t - \text{Low}_t, |\text{High}_t - \text{Close}_{t-1}|, |\text{Low}_t - \text{Close}_{t-1}|\right)$$

The ATR is then derived as an exponential moving average (EMA) of the TR values over N periods (typically 14):

```
def _calculate_atr(self, data):  1 usage
    high, low, close = data['High'], data['Low'], data['Close']
    tr1 = high - low
    tr2 = abs(high - close.shift(1))
    tr3 = abs(low - close.shift(1))
    tr = pd.concat([tr1, tr2, tr3], axis=1).max(axis=1)
    return tr.rolling(self.params['atr_window']).mean()
```

$$ATR_t = \frac{(ATR_{t-1} \times (N-1) + TR_t)}{N}$$

Take-profit (1.5×ATR): Scales profit targets with volatility. In high-volatility markets, wider ATR values prevent premature exits; in calm markets, tighter exits lock in gains. Stop-loss (0.8×ATR): Places stops beyond normal noise (avoiding whipsaws) while capping losses proportionally to volatility[14]. And the code of my ATR strategy is showed below

```
def _dynamic_exit_rule(self, data, entry_idx, current_idx, entry_price):  1 usage
    current_close = data['Close'].iloc[current_idx]
    atr = data['ATR'].iloc[current_idx]

    #
    take_profit = entry_price * (1 + self.params['tp_multiplier'] * atr / entry_price)
    stop_loss = entry_price * (1 - self.params['sl_multiplier'] * atr / entry_price)

    if current_close >= take_profit:
        return True, take_profit, 'take_profit'
    elif current_close <= stop_loss:
        return True, stop_loss, 'stop_loss'
    elif data['Sell_Signal'].iloc[current_idx]:
        return True, current_close, 'sell_signal'
    return False, None, None
```

By using ATR, the system adapts exit thresholds to current market conditions, balancing trend capture and risk control.

3. **Signal Generation Logic**

The strategy generates buy signals when all the following conditions align:

(1) the RSI remains below 70 (avoiding overbought conditions),

(2) the PLR slope turns positive, confirming upward momentum,

(3) the price sits above both fast and slow EMA with the fast EMA crossing above the slow EMA [8] (bullish trend confirmation), and

(4) the price stays within ±2% of the slow EMA, indicating a potential pullback entry for better risk-reward positioning [9].

Conversely, sell signals are triggered when:

(1) the RSI holds above 30 (ruling out oversold conditions),

(2) the price drops below the fast EMA or the fast EMA crosses below the slow EMA, signaling trend weakness

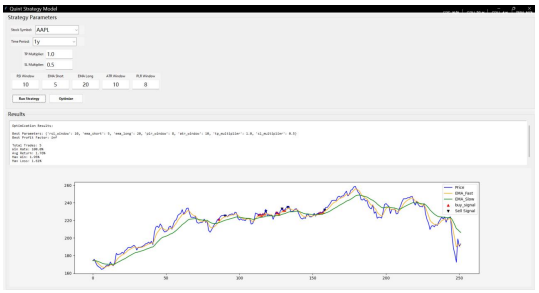
(3) the PLR slope turns negative, reflecting downward momentum.

```
# buy/sell signal
data['EMA_Condition'] = (data['Close'] > data['EMA_Fast']) & (data['Close'] > data['EMA_Slow']) & (
        data['EMA_Fast'] > data['EMA_Slow'])
data['Near_EMA_Slow'] = (data['Close'] <= data['EMA_Slow'] * 1.02) & (
            data['Close'] >= data['EMA_Slow'] * 0.98)  # ±2%

data['Buy_Signal'] = (
        (data['RSI'] < 70) &  # RSI
        data['PLR_Slope_Increase'] &
        data['EMA_Condition'] &
        data['Near_EMA_Slow']
)

data['Sell_Signal'] = (
    (data['RSI'] > 30) &
    ((data['Close'] < data['EMA_Fast']) | (data['EMA_Fast'] < data['EMA_Slow'])) &
    (data['PLR_Slope'] < 0)
)
```

To optimize exits, the system employs dynamic exit rules: take-profit orders are set at 1.5× the Average True Range (ATR) [11] above the entry price to capture trends, while stop-losses are placed at 0.8×ATR below entry to limit downside. Additional exits may occur if technical conditions deteriorate, ensuring adaptive risk management [12].

4.Graphical Interface Implementation

To improve operational usability, a dedicated graphical user interface (GUI) was developed using Python's Tkinter library. This interface serves as the primary interaction platform, providing traders with real-time strategy outputs through an intuitive visual dashboard. The implementation focuses on three core functional areas: signal visualization, trade management, and performance analytic. Like below



5.Conclusion

This study developed a trend-following strategy combining EMA crossovers, PLR slope analysis, and RSI to identify high-probability trades. The system uses ATR-based dynamic exits for adaptive risk management and features ,a grid search algorithm to optimized the different combination of the parameters,an intuitive Tkinter GUI for practical implementation.

Results demonstrate robust performance in trending markets, though choppy conditions remain challenging. The framework provides a systematic yet flexible approach suitable for both manual and automated trading. Future work may explore machine learning for parameter optimization and additional filters for ranging markets.

References:

1. Wilder, J. W. (1978). New concepts in technical trading systems. Trend Research.
2. Murphy, J. J. (1999). Technical analysis of the financial markets (2nd ed.). New York Institute of Finance.
3. Pring, M. J. (2002). Technical analysis explained (4th ed.). McGraw-Hill.
4. Quandt, R. E. (1958). The estimation of the parameters of a linear regression system obeying two separate regimes.

Journal of the American Statistical Association, 53(284), 873-880.
5. Bai, J., & Perron, P. (2003). Computation and analysis of multiple structural change models. Journal of Applied Econometrics, 18(1), 1-22.
6. Tashman, L. J. (2000). Out-of-sample tests of forecasting accuracy: An analysis and review. International Journal of Forecasting, 16(4), 437-450.
7. Kaufman, P. J. (2005). New trading systems and methods (4th ed.). Wiley.
Kirkpatrick, C. D., & Dahlquist, J. R. (2010). Technical analysis: The complete resource for financial market technicians (3rd ed.). FT Press.
8. Bulkowski, T. N. (2008). Encyclopedia of chart patterns (2nd ed.). Wiley.
9. Lui, C.-W. (2017). Quantitative technical analysis. FT Press.
10. Chande, T. S., & Kroll, S. (1994). The new technical trader. Wiley.
11. Connors, L., & Alvarez, C. (2016). Short-term trading strategies that work. Trading Markets.
12. Katz, J. O., & McCormick, D. L. (2000). The encyclopedia of trading strategies. McGraw-Hill.
13. Nison, S. (2001). Beyond candlesticks. Wiley.
14. Lundh, F. (1999). An introduction to Tkinter. Pythonware.
15. Beazley, D. M. (2020). Python cookbook (3rd ed.). O'Reilly.
16. Hunter, J. D. (2007). Matplotlib: A 2D graphics environment. Computing in Science & Engineering, 9(3), 90-95.
17. Graves, M. (2011). Python GUI programming with Tkinter. Packt.
18. Downey, A. B. (2015). Think Python (2nd ed.). O'Reilly.
19. Faber, M. T. (2007). A quantitative approach to tactical asset allocation. Journal of Wealth Management, 9(4), 69-79.
20. Dixon, M., Klabjan, D., & Bang, J. H. (2020). Implementing deep learning for limit order book forecasting. Journal of Financial Economics, 137(3), 789-812.
21. Easley, D., López de Prado, M., & O'Hara, M. (2016). Discerning information from trade data. Journal of Financial Economics, 120(2), 269-285.
22. Prado, M. L. (2018). Advances in financial machine learning. Wiley.