Explanation of the RobustnessAnalyzer Class (MONTE CARLO ROBUSTNESS TEST)

Tuche Daddy :)

February 14, 2025

1 Overview

The `RobustnessAnalyzer` class is designed to perform additional robustness tests on your trading strategy by simulating small variations in parameters (via Monte Carlo simulation) and, optionally, applying stress testing. At the moment before running this module, you already have a final set of parameters and have executed a backtest using these new parameters. The purpose of this class is to test how sensitive and stable your strategy is under slight variations in these parameters and different market conditions.

2 What the Code Does

2.1 Initialization (`__init__`)

- The constructor accepts three arguments:
  - **strategy:** A reference to the `Strategy` class.
  - **params:** Your final (base) set of parameters.
  - **df_list:** The historical market data for the various trading pairs.
- These values are stored within the object for use in simulations.

2.2 Monte Carlo Simulation (`monte_carlo_simulation`)

- The method `monte_carlo_simulation` runs a series of simulations (default 100, but can be set higher) to test the strategy's robustness.
- **For each simulation:**
  - **Perturb Parameters:** The method `_perturb_parameters` generates slightly varied (perturbed) parameters based on the final set. For example, if the moving average window is 10, a perturbation within ±10% might adjust it to a value around 9 to 11. Similar perturbations are applied to the envelope values and the trade size.
  - **Data Usage:** In this version, the code uses a copy of your original data (`df_list.copy()`) rather than synthetic data. (In a more advanced setting, you could generate synthetic data to further stress-test the strategy.)
  - **Run Backtest:** A new instance of the `Strategy` is created with these perturbed parameters and the (copied) market data. The strategy's indicators are populated, buy/sell signals are generated, and a backtest is executed.
  - **Calculate Metrics:** The method `_calculate_metrics` extracts key performance metrics from the backtest results, including the Sharpe ratio, maximum drawdown, total return, win rate, and profit factor.
- All valid metrics are stored in a list.
- Once all simulations are complete, the method calls `_analyze_simulation_results` to aggregate the results, compute confidence intervals, produce visualizations, and calculate an overall stability score.

2.3 Perturbing Parameters (`_perturb_parameters`)

- This method takes your base (optimal) parameters and applies a random perturbation (within a defined range, default ±10%).
- It also includes bounds (e.g., ensuring the moving average window is at least 5, envelope values stay within 0.001 and 0.5, and the trade size stays between 0.01 and 1.0) to prevent nonsensical parameter values.

2.4 Calculating Metrics (`_calculate_metrics`)

- This method processes the backtest result to compute:
  - **Sharpe Ratio:** Annualized, using either the percentage change or a fallback.
  - **Maximum Drawdown:** The worst decline in the equity curve.
  - **Total Return:** The percentage change from the first to the last wallet balance.
  - **Win Rate:** The fraction of trades that were profitable.
  - **Profit Factor:** The ratio of total profits to total losses.
- If there is an error (e.g., if the days or trades DataFrames are empty), it returns `None` so that the simulation can continue.

2.5 Analyzing Simulation Results (`_analyze_simulation_results`)

- Once all simulation runs are complete, this method:
  - Converts the list of metrics into a DataFrame.
  - Computes confidence intervals for each metric (e.g., the 95% confidence interval).
  - Visualizes the distribution of each metric (Sharpe, total return, maximum drawdown, win rate) with histograms and KDE plots.
  - Calculates an overall stability score based on the coefficient of variation (a lower stability score indicates more robust performance).

2.6 Helper Methods

- `calculate_sharpe` and `calculate_max_drawdown` are helper functions that compute the respective metrics.
- `_plot_simulation_results` creates visual plots for each metric with confidence interval lines.
- `_calculate_stability_score` aggregates the coefficient of variation across metrics to provide an overall robustness indicator.

3 How It Fits Into Your Pipeline

At this point, you already have a final set of parameters and have executed a backtest using those parameters. The role of the `RobustnessAnalyzer` is to **test the sensitivity and stability** of your strategy by:

1. **Perturbing the Final Parameters:** By making small random changes to your optimal parameters, you simulate the uncertainty or slight deviations that might occur in live trading.
2. **Running Multiple Backtests:** Each simulation run uses a slightly different set of parameters on the same (or similar) market data, producing a distribution of performance metrics.
3. **Analyzing the Distribution:** By computing confidence intervals and a stability score, you gain insight into whether your strategy's performance is robust. If performance metrics vary widely with small parameter changes, your strategy may be overly sensitive.

4 Summary

In essence, the `RobustnessAnalyzer`:

- Perturbs your final set of parameters.
- Runs multiple backtests on either the original or (optionally) synthetic data.

- Collects key performance metrics (such as Sharpe ratio, drawdown, return, win rate, and profit factor).
- Aggregates these metrics to compute confidence intervals and a stability score.
- Visualizes the results so that you can assess the robustness of your strategy.

This process helps ensure that the final parameters are not only optimal on historical data but also remain effective when faced with small changes or market uncertainties—crucial for reliable live trading.