Introduction

In the realm of algorithmic trading, backtesting plays a crucial role in validating a trading strategy before it is deployed in the live market. For our project, we are developing and evaluating a trading strategy based on the Inverse Cramer Effect, which exploits the negative correlation between Jim Cramer's top and bottom picks from Mad Money. However, Selecting the right backtesting framework can be a daunting task, especially with the myriad of options available in Python.

In this blog post, we will explore the challenges in picking a suitable backtesting framework for our Inverse Cramer Strategy, our thought process behind choosing QuantConnect, and the development experience of using their platform. We'll consider factors such as the trade-off of event-driven vs vectorized frameworks, the learning curve, support for multi-asset portfolios, and more.

The Backtesting Framework Landscape

The Python ecosystem offers numerous backtesting frameworks, ranging from vectorized libraries like vectorbt to more sophisticated event-driven platforms such as Backtrader and QuantConnect. Choosing the right framework depends on various factors, including the desired level of complexity, speed, accuracy, and flexibility.

Event-driven frameworks, like QuantConnect, model the market conditions more realistically and offer greater flexibility. However, they can be slower and more complex to implement. On the other hand, vectorized frameworks are faster and easier to implement but may be less accurate and flexible due to their simplified nature.

Some of the backtesting frameworks suggested by the lecture notes are no longer maintained. For example, PyAlgoTrade has not been updated since 2018 and is no longer compatible with the latest version of Pandas. As such, we decided to explore other options.

The Decision Process: Why QuantConnect?

After evaluating the options, we decided on QuantConnect for several reasons:

Built-in data sources

For other frameworks, we would have to source and manage data from external source. Since we have discovered that the inverse Cramer strategy would require fetching data for a large universe of stocks, we decided that it would be more efficient to use a framework that provides built-in data sources, circumventing the need to set up a database.

QuantConnect provides free access to high-quality historical data for multiple asset classes, including equities, futures, and forex. This eliminates the need to source and manage data from external sources, which can be time-consuming and costly.

Comprehensive support for multiple asset portfolios

Compared to other frameworks like backtest.py which only supports the trading of a single asset at a time, QuantConnect supports the trading of multiple assets. This is crucial for our strategy, which requires us to maintain a dynamic portfolio of long/short positions.

A cloud-based platform

QuantConnect is a cloud-based platform that eliminates the need for local setup and provides scalable resources. This is especially useful for us as we are working on a shared project and do not want to deal with the hassle of setting up a local environment.

Comprehensive backtesting reports

QuantConnect provides comprehensive backtesting reports that include performance metrics, risk management, and transaction details. This allows us to evaluate the strategy's effectiveness and identify areas for improvement, without having to implement these features ourselves.

Overcoming the Learning Curve with QuantConnect

Initially, learning the ins and outs of QuantConnect's LEAN engine was challenging. However, the platform offers comprehensive documentation, community forums, and example algorithms that aided in overcoming the learning curve. These resources provided valuable insights into best practices, optimizing performance, and debugging.

Developing an Algorithm in QuantConnect

We set out to develop an algorithm exploring the Inverse Cramer Effect. Throughout the development process, QuantConnect proved to be a powerful platform. Key features included:

Data handling and custom data classes

Importing custom data, in our case a CSV file containing the top picks from Jim Cramer's Mad Money, was straightforward. We created a custom data class to parse the CSV file and store the data in a format that can be easily accessed by the algorithm.

```
CSV_PATH = "https://raw.githubusercontent.com/Edward-
choi/FINA4350/main/data/backtest/cramer_top_picks_2017-2022.csv"

class CramerStock(PythonData):
    """Custom Data class for Jim Cramer's Top Picks from Mad Money from 2017-2022

    > Date,Top5,Bottom5
    > 2017-01-31,"['M', 'TGT', 'KSS', 'UA', 'FSLR']","['AMZN', 'BAC', 'JPM',
'AAPL', 'UNH']"
    """

    def __init__(self):
        self.Algorithm = None


    def GetSource(self, config, date, isLiveMode):
        # config.SetParameter("algorithm", self.Algorithm)
        return SubscriptionDataSource(CSV_PATH,
SubscriptionTransportMedium.RemoteFile)
```

```
    def Reader(self, config, line, date, isLiveMode):
         if not (line.strip() and line[0].isdigit()):
            return None

        index = CramerStock()
        index.Symbol = config.Symbol
        index.Date = date

        # Parse the CSV line
        date_str, top5_str, bottom5_str = re.split(',(?![^[]*\])', line)
        index.Date = datetime.strptime(date_str, "%Y-%m-%d").date()
        index.Top5 = eval(eval(top5_str))
        index.Bottom5 = eval(eval(bottom5_str))

        if self.Algorithm is not None:
            self.Algorithm.Debug(f"Reader: Date {index.Date}, Top5 {index.Top5},
Bottom5 {index.Bottom5}")

        return index
```

Implementing universe selection and rebalancing

The platform's built-in functions facilitated the creation of dynamic stock universes and periodic rebalancing.

Managing risk and performance evaluation

The platform provided risk management features and detailed performance metrics to assess the strategy's effectiveness, and also probability of overfitting.

Encountering and Overcoming Technical Challenges

During the development process, we encountered several technical and conceptual challenges. For example, we faced difficulties in handling custom data and managing the rebalancing schedule. By seeking help from the community and experimenting with alternative approaches, we were able to overcome these obstacles. These experiences have deepened our understanding of algorithmic trading.

Conclusion

Our journey with QuantConnect has been both rewarding and insightful. The platform met our initial criteria and proved to be a powerful tool for developing and backtesting trading strategies. While selecting the right backtesting framework ultimately depends on one's needs and preferences, we believe that QuantConnect strikes a good balance between complexity and usability. We look forward to exploring the platform further and developing more sophisticated trading strategies in the future.