The Consequences of Fat-Tailed Distributions

Thomas Misikoff

2024-03-04

Understanding Fat-Tailed Distributions

Overview

Many of the ideas

Software for Improved Understanding

I've put together an R package, `TailRiskAnalyzer` available at https://misikoff.github.io/TailRiskAnalyzer/. This package includes main functions for exploring the implications of fat tails. This functions range the analytical to the demonstrative. A few functions are described below:

- `p_normal_given_event`:
- `create_side_by_side_plot`:
- `draw_lln_with_func_facet`:

Real World Applications in Finance

Considerations for these distributions can be valuable in many real-world applications, like investing in the U.S. stock market. For example, the S&P 500 is one of the most popular stock market indices, containing the 500 largest listed stocks in the U.S. market, by market capitalization (stock price $*$ outstanding shares). This index can be traded by investing in an exchange-traded fund (ETF), such as SPDR S&P 500 ETF Trust (SPY). The daily returns of SPY are fat-tailed, exhibiting a kurtosis much greater than that expected by data generated from a normal distribution. My analysis of the log of daily returns from January 29th, 1993 through March 1st, 2024, found the kurtosis to be greater than 14. The data for this analysis was gathered from Yahoo Finance.

Investing in the S&P 500, via SPY, is a classic buy-and-hold strategy with a great historic record, but this strategy's risk profile may be improved through "insurance" to adjust it's risk profile, in light of the excess kurtosis of the returns. In this case, the insurance we are seeking should have a predictable, fixed cost, and offset losses in SPY. Insuring this portfolio against catastrophic losses can be seen as a way to raise the median outcome in exchange for lowering the mean outcome through insurance premiums. Here the median and the mean can be far apart, due to the excess kurtosis. This could be a great exchange given we don't have the luxury of experiencing all potential paths for the portfolio and must be satisfied with the single path we experience.

Insurance Through Options

Buying options is a natural fit for insuring a portfolio, as they have fixed costs. A 'put' option gives the owner the right to sell a stock at a specified price at any time between the purchase of the contract and it's expiration, as opposed to a 'call' option which gives the owner the right to sell at a specified price. If we wish to offset major downturns in a stock, we may purchase 'put' options that become increasingly profitable as the stock value drops.

Insuring a SPY Portfolio

The portfolio's primary position will be shares of SPY. The remainder of the portfolio will be invested in 'put' options which become profitable when large downturns in SPY occur, or are expected. These options expire over time, so the portfolio must be rebalanced periodically. For the specifics of handling the put Options, I borrowed from Mark Spitznagel's The Dao of Capital, which suggested hedging against tail-risk through put options with a strike price 30% below current market value that expire in 60 options. Every 30 days, these options will be sold and a new batch of options expiring 60 days in the future will be purchased.

Measuring Performance through Portfolio Statistics

o

Beta

$$\beta_p = \frac{\text{Cov}(r_p, r_m)}{\text{Var}(r_m)}$$

- β_p = market beta of asset p
- r_m = average expected rate of return on the market
- r_p = expected return on portfolio p

β measures the variance of a portfolio's returns relative to the variance of the market. It is a limited, but popular proxy for risk. This variance is not only relevant in terms of outcomes, but also in terms of psychology, as intermittent fluctuations may be tough to for investors to bear.

Treynor Ratio

$$T_p = \frac{r_p - r_f}{\beta_p}$$

- r_p = expected return on portfolio p
- r_f = risk-free rate of return
- B_p = beta of portfolio p

The Treynor Ratio measures the excess returns (compared to the risk-free rate of return), compared to the β of the asset. This ratio can be seen as a simple score for an asset or portfolio, where higher numbers represent more favorable outcomes.

Results

I implemented this strategy on the QuantConnect platform to simulate it's performance. Results were simulated for January 1st, 2007 through January 1st 2024. This time period was selected based on the data available on the platform at the time. In the table below, the results of the "insured" strategy are compared to the "naive" buy-and-hold strategy.

Strategy	Return	β_p	T_p
SPY Buy-and-Hold	360.60	0.996	0.062
Insured SPY	388.68	0.597	0.113

As the table above shows, the *beta* of the insured strategy was significantly reduced. In this case, in a welcome surprise, the insurance itself yielded a positive return thanks to the shock of the COVID pandemic, which increased concern for a market crash, creating a huge payoff for the strategy's put options in March of 2020.

These two factors equate to an improvement in the Treynor Ratio of about 82% over the naive "SPY Buy-and-Hold" strategy.

References

- Nassim Nicholas Taleb. (2020). Statistical Consequences of Fat Tails: Real World Preasymptotics, Epistemology, and Applications. Stem Academic Press.
- Mandelbrot, B. B., & Hudson, R. L. (2010). The (Mis)Behaviour of markets. Profile Books.
- Maclean, L. C., Thorp, E. O., & Ziemba, W. T. (2011). The Kelly capital growth investment criterion : theory and practice. World Scientific.
- Ole Peters. (2011). Optimal leverage from non-ergodicity, Quantitative Finance, 11:11, 1593-1602, DOI: 10.1080/14697688.2010.513338
- Ole Peters. (2019). The Ergodicity problem in economics. Nat. Phys. 15, 1216–1221. https://doi.org/10.1038/s41567-019-0732-0
- Ole Peters and Alexander Adamou. (2021). The Time interpretation of expected utility theory. arXiv.org
- Spitznagel, M. (2013). The Dao of capital : Austrian investing in a distorted world. Wiley.
- Spitznagel, M., & Taleb, N. N. (2021). Safe haven: investing for financial storms. Wiley.

Appendix

QuantConnect Algorithm

Parameter Settings:

- option_weight: 0.005
- equity: "SPY"
- OOM: 0.3

```python
from AlgorithmImports import *

class LogicalLightBrownWhale(QCAlgorithm):
    def Initialize(self):
        self.SetStartDate(2007, 1, 1)
        self.SetEndDate(2024, 1, 1)

        self.initCash = 100000
        self.SetCash(self.initCash)

        # Benchmark
        self.MKT = self.AddEquity(self.GetParameter("equity"), Resolution.Daily).Symbol
        self.mkt = []

        option = self.AddOption(self.GetParameter("equity"), Resolution.Daily)
        option.SetFilter(minExpiry = timedelta(50), maxExpiry = timedelta(70))
        self.symbol = option.Symbol

        self.friday_count = 0

        # reset the Friday counter each month
        self.Schedule.On(self.DateRules.MonthStart(self.symbol),
                         self.TimeRules.AfterMarketOpen(self.symbol, 0),
                         self.OnNewMonth)

        # Schedule the monthly rebalancing
        self.Schedule.On(self.DateRules.WeekEnd(),
                         self.TimeRules.AfterMarketOpen(self.symbol, 0),
                         self.Rebalance)

    def OnData(self, data):
        self.data = data

    def OnNewMonth(self):
        self.friday_count = 0

    def Rebalance(self):
        self.friday_count += 1

        if not self.friday_count == 3: return
        if not hasattr(self, "data"): return

        self.updateBenchmark()

        self.Log(f"Rebalancing")
        for symbol in self.getHeldOptionsSymbols():
            self.Liquidate(symbol)

        chain = self.data.OptionChains.get(self.symbol)
        if not chain:
            self.SetHoldings([PortfolioTarget(self.GetParameter("equity"), 1)])
            return

        # filter for put contracts
        contracts = [x for x in chain if x.Right == OptionRight.Put]
        # filter for contracts far enough OOM
        contracts = [x for x in contracts if x.Strike <
            float(1 - float(self.GetParameter("OOM"))) * x.UnderlyingLastPrice]
        # filter for contracts expiring in 2 months
        contracts = [x for x in contracts if (x.Expiry - self.data.Time).days > 50]

        self.Log(f"Current {self.GetParameter("equity")}: "
                 f"{self.Securities[self.GetParameter("equity")].Price}")

        contracts = sorted(contracts, key = lambda x: x.Strike, reverse = True)

        if len(contracts) > 0:
            selected_contract = contracts[0]
            self.Log(f"buying put option with strike: ${selected_contract.Strike}, "
                     f"expiring: {selected_contract.Expiry.date()}")
            self.SetHoldings([PortfolioTarget(
                self.GetParameter("equity"),
                 1 - float(self.GetParameter("option_weight"))),
                 PortfolioTarget(selected_contract.Symbol,
                 float(self.GetParameter("option_weight")))])
        else:
            self.SetHoldings([PortfolioTarget(self.GetParameter("equity"), 1)])

    def updateBenchmark(self):
        mkt_price = self.History(self.MKT,
                                 2,
                                 Resolution.Daily)["close"].unstack(level= 0).iloc[-1]
        self.mkt.append(mkt_price)

        # Does not handle any stock splits that occur
        mkt_perf = self.initCash * self.mkt[-1] / self.mkt[0]
        self.Plot("Strategy Equity", self.MKT, mkt_perf)

    def OnEndOfAlgorithm(self):
        self.Log("is working the end of algo function")
        self.updateBenchmark()

    def getHeldOptionsSymbols(self):
        option_symbol_list = []
        invested = [x.Symbol for x in self.Portfolio.Values if x.Invested]
        for symbol in invested:
            if symbol.SecurityType == SecurityType.Option:
                option_symbol_list.append(symbol)
        return option_symbol_list
```