Not Over Thinking

Momentum in Mutual Fund Return

Algorithmic Trading Strategy with Full Code

Haixiang

2023.10 | Vol 39.

hxyan.2015@gmail.com | github.com/hxyan2020

STRATEGY & ECONOMIC RATIONALE

The investment universe consists of equity funds from the CRSP Mutual Fund database. This universe is then shrunk to no-load funds (to remove entrance fees). Investors then sort mutual funds based on their past 6-month return and divide them into deciles. The top decile of mutual funds is then picked into an investment portfolio (equally weighted), and funds are held for three months. Other measures of momentum could also be used in sorting (fund's closeness to 1 year high in NAV and momentum factor loading), and it is highly probable that the combined predictor would have even better results than only the simple 6-month momentum.

BUY	SELL
top decile of mutual funds	The opposite

PARAMETER & VARIABLES

PARAMETER	VALUE
MARKETS TRADED	Equity
FINANCIAL INSTRUMENTS	Funds
REGION	United States
PERIOD OF REBALANCING	Quarterly
NO. OF TRADED INSTRUMENTS	10
WEIGHTING	Equal weighting
LOOKBACK PERIODS	3 months
LONG/SHORT	Long only

ALGORITHM

```
class MomentuminMutualFundReturns(QCAlgorithm):

    def Initialize(self):
        # NOTE: most of the data start from 2014 and until 2015 there wasn't any trade
        self.SetStartDate(2014, 1, 1)
        self.SetCash(100000)

        self.data = {}
        self.symbols = []

        self.period = 21 * 6 # Storing 6 months of daily prices
        self.quantile = 10

        self.symbol = self.AddEquity('SPY', Resolution.Daily).Symbol

        # Load csv file with etf symbols and split line with semi-colon
        etf_symbols_csv = 
self.Download("data.quantpedia.com/backtesting_data/equity/mutual_funds/symbols.csv")
        splitted_csv = etf_symbols_csv.split(';')

        for symbol in splitted_csv:
            self.symbols.append(symbol)
            # Subscribe for QuantpediaETF by etf symbol, then set fee model and leverage
            data = self.AddData(QuantpediaETF, symbol, Resolution.Daily)
            data.SetFeeModel(CustomFeeModel())
            data.SetLeverage(5)

            self.data[symbol] = RollingWindow[float](self.period)

        self.recent_month = -1

    def OnData(self, data):
        # Update daily prices of etfs
        for symbol in self.symbols:
            if symbol in data and data[symbol]:
                price = data[symbol].Value
                self.data[symbol].Add(price)

        if self.recent_month == self.Time.month:
            return
        self.recent_month = self.Time.month

        # Rebalance quarterly
        if self.recent_month % 3 != 0:
            return

        performance = {}

        for symbol in self.symbols:
            # If data for etf are ready calculate it's 6 month performance
            if self.data[symbol].IsReady:
                if self.Securities[symbol].GetLastData() and (self.Time.date() - 
self.Securities[symbol].GetLastData().Time.date()).days <= 3:
                    prices = [x for x in self.data[symbol]]
                    performance[symbol] = (prices[0] - prices[-1]) / prices[-1]

        if len(performance) < self.quantile:
            self.Liquidate()
            return

        decile = int(len(performance) / self.quantile)
        # sort dictionary by performance and based on it create sorted list
        sorted_by_perf = [x[0] for x in sorted(performance.items(), key=lambda item: 
item[1], reverse=True)]
        # select top decile etfs for investment based on performance
        long = sorted_by_perf[:decile]

        # Trade execution
        invested_etfs = [x.Key for x in self.Portfolio if x.Value.Invested]
        for symbol in invested_etfs:
            if symbol not in long:
                self.Liquidate(symbol)

        long_length = len(long)
        for symbol in long:
            self.SetHoldings(symbol, 1 / long_length)

# Quantpedia data
# NOTE: IMPORTANT: Data order must be ascending (datewise)
class QuantpediaETF(PythonData):
    def GetSource(self, config, date, isLiveMode):
        return
SubscriptionDataSource("data.quantpedia.com/backtesting_data/equity/mutual_funds/{0}.csv".
format(config.Symbol.Value), SubscriptionTransportMedium.RemoteFile, FileFormat.Csv)

    def Reader(self, config, line, date, isLiveMode):
        data = QuantpediaETF()
        data.Symbol = config.Symbol

        if not line[0].isdigit(): return None
        split = line.split(';')

        data.Time = datetime.strptime(split[0], "%d.%m.%Y") + timedelta(days=1)
        data['settle'] = float(split[1])
        data.Value = float(split[1])

        return data

# Custom fee model
class CustomFeeModel(FeeModel):
    def GetOrderFee(self, parameters):
        fee = parameters.Security.Price * parameters.Order.AbsoluteQuantity * 0.00005
        return OrderFee(CashAmount(fee, "USD"))
```

BACKTESTING PERFORMANCE

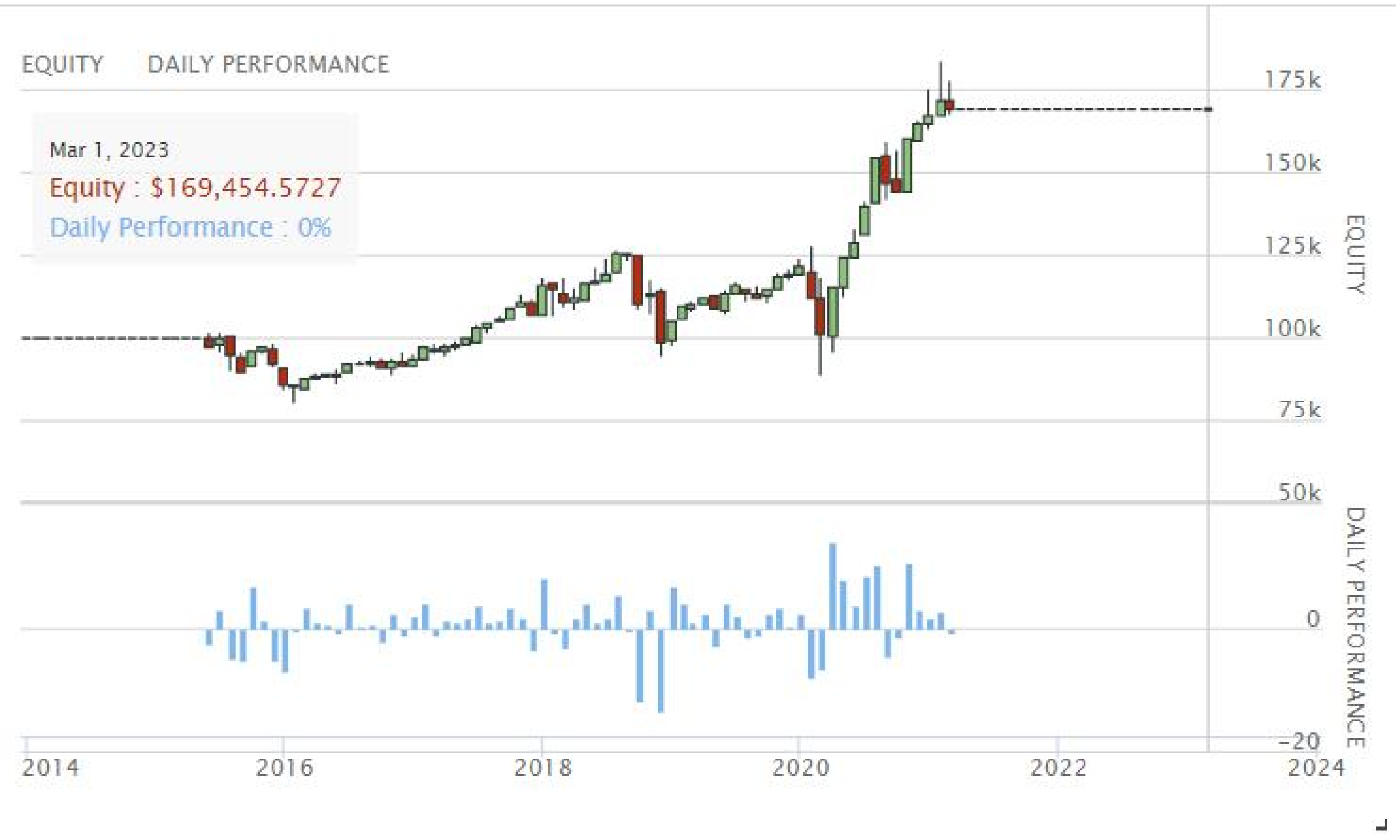
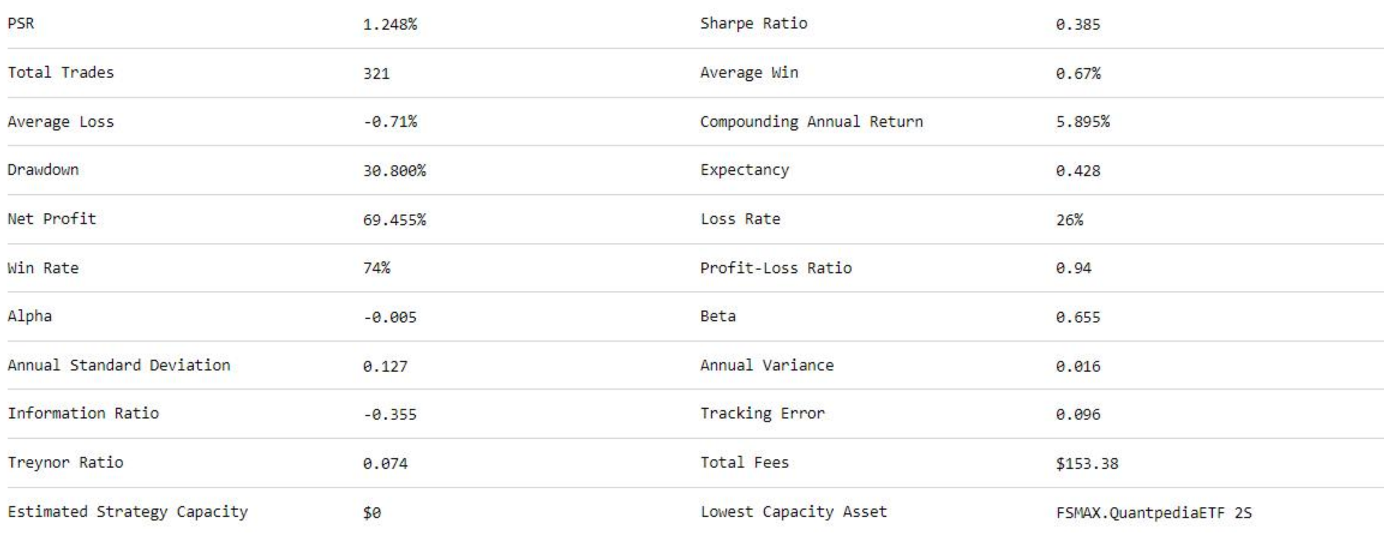


Fig 1. Overall Performance



PSR	1.248%	Sharpe Ratio	0.385
Total Trades	321	Average Win	0.67%
Average Loss	-0.71%	Compounding Annual Return	5.895%
Drawdown	30.800%	Expectancy	0.428
Net Profit	69.455%	Loss Rate	26%
Win Rate	74%	Profit-Loss Ratio	0.94
Alpha	-0.005	Beta	0.655
Annual Standard Deviation	0.127	Annual Variance	0.016
Information Ratio	-0.355	Tracking Error	0.096
Treynor Ratio	0.074	Total Fees	$153.38
Estimated Strategy Capacity	$0	Lowest Capacity Asset	FSMAX.QuantpediaETF 2S

Fig 2. Performance Metrics

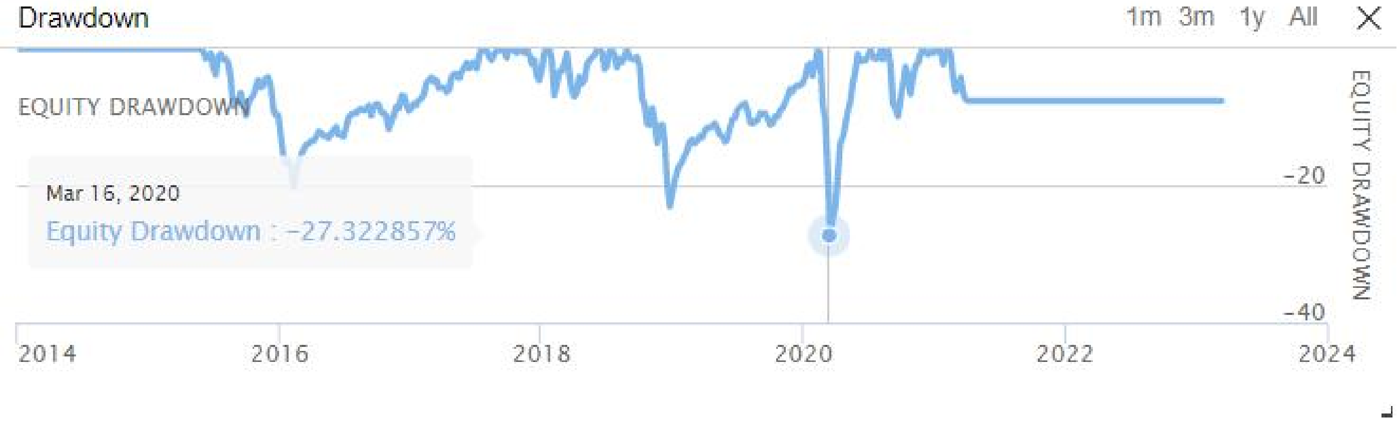


Fig 3. Drawdown

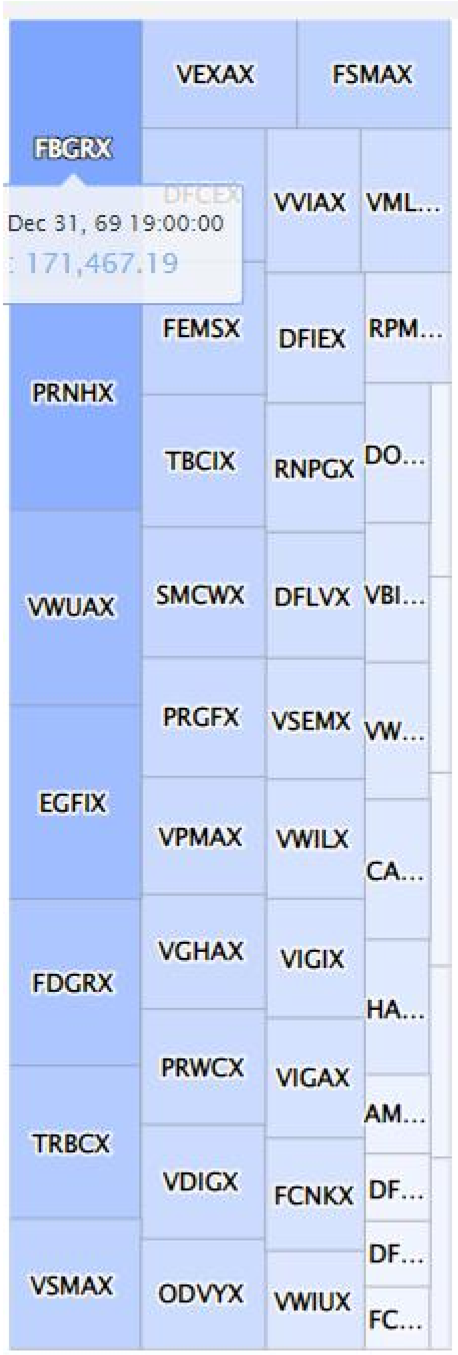


Fig 4. Assets Sales Volume