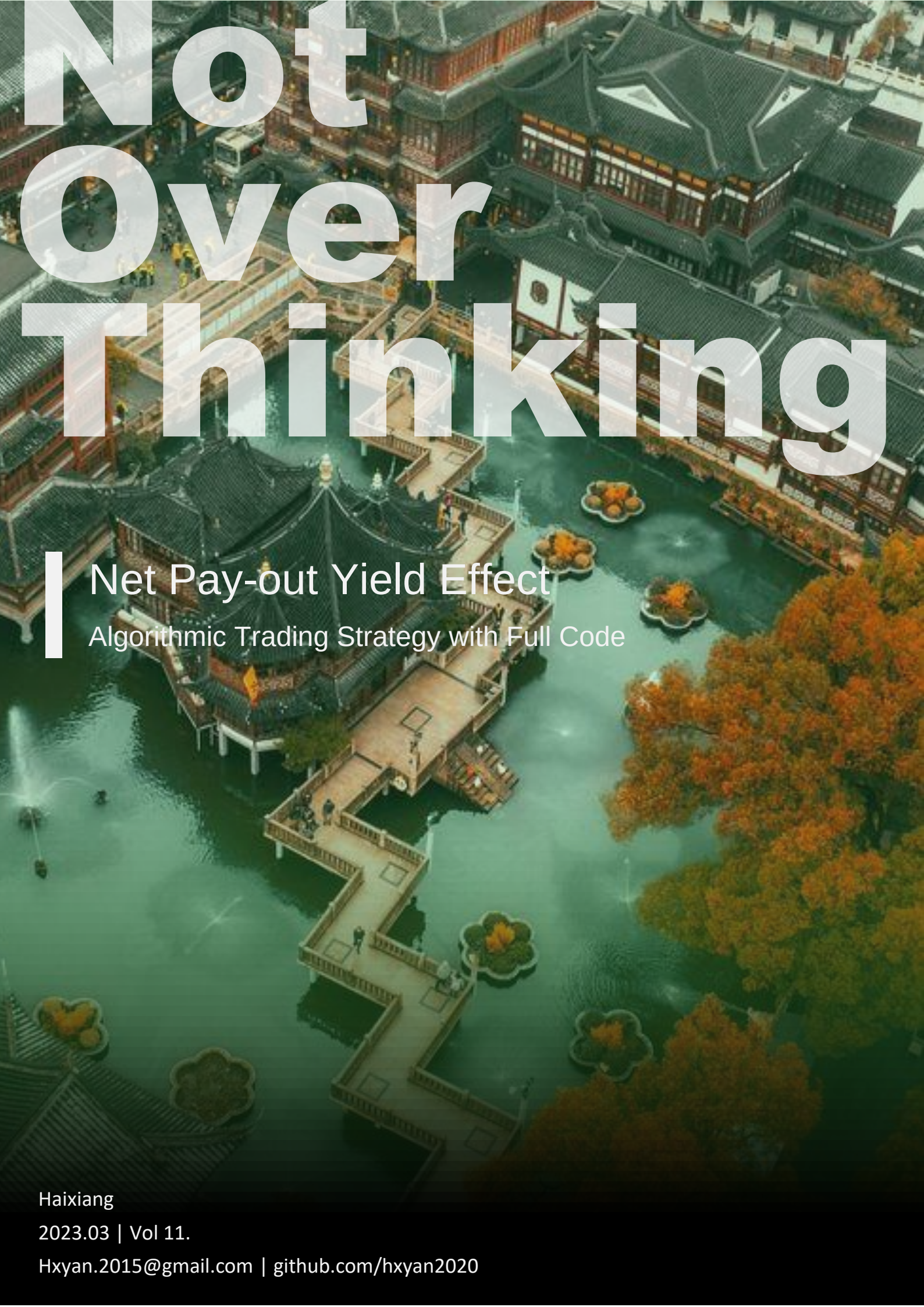

Not Over Thinking

Net Pay-out Yield Effect

Algorithmic Trading Strategy with Full Code

Haixiang

2023.03 | Vol 11.

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

STRATEGY & ECONOMIC RATIONALE

The investment universe consists of all stocks on NYSE, AMEX, and NASDAQ. At the end of June of each year t, ten portfolios are formed based on ranked values net pay-out yield.

LONG	SHORT
Portfolio with the highest net pay-out ratio and held for 1 year	Portfolio with the lowest net pay-out ratio and held for 1 year

PARAMETER & VARIABLES

- The net pay-out yield is the ratio of dividends plus repurchases minus common share issuances in year t to year-end market capitalization. There are two measures of pay-out yield, one based on the statement of cash flows, the other based on the change in Treasury stocks. For the net pay-out yield, we use the cash flow-based measure of repurchases.

PARAMETER	VALUE
MARKETS TRADED	NYSE, AMEX, NASDAQ
FINANCIAL INSTRUMENTS	Stocks
PERIOD OF REBALANCING	1 year
NO. OF TRADED INSTRUMENTS	500
WEIGHTING	Equal weighting
HOLDING PERIODS	1 year
LONG/SHORT	Long Only

ALGORITHM

```
from AlgorithmImports import *

class NetPayoutYieldEffect(QCAlgorithm):

    def Initialize(self):
        self.SetStartDate(2000, 1, 1)
        self.SetCash(100000)

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

        self.coarse_count = 500  ## coarse universe will consist 500 stocks
        self.quantile = 10
        self.leverage = 5

        self.long = []

        self.selection_flag = False
        self.UniverseSettings.Resolution = Resolution.Daily
        self.AddUniverse(self.CoarseSelectionFunction, self.FineSelectionFunction)
        self.Schedule.On(self.DateRules.MonthEnd(self.symbol),
self.TimeRules.AfterMarketOpen(self.symbol), self.Selection)
```

```
    def OnSecuritiesChanged(self, changes):
        for security in changes.AddedSecurities:
            security.SetFeeModel(CustomFeeModel())
            security.SetLeverage(self.leverage)

    def CoarseSelectionFunction(self, coarse):
        if not self.selection_flag:
            return Universe.Unchanged  ## make sure up till now there is no symbols selected

        selected = sorted([x for x in coarse if x.HasFundamentalData and x.Market == 'usa'],
            key=lambda x: x.DollarVolume, reverse=True)
        ## only symbols that (i) have fundamental data; (ii) traded in the US market are
selected; then sorted by trading volume

        return [x.Symbol for x in selected[:self.coarse_count]]
        ## [:self.coarse_count] means select from the first to the last item in the carse
universe

    def FineSelectionFunction(self, fine):
        fine = [x for x in fine if x.MarketCap !=0 and x.ValuationRatios.TotalYield != 0 and
x.FinancialStatements.CashFlowStatement.CommonStockIssuance.TwelveMonths != 0 and
            ((x.SecurityReference.ExchangeId == "NYS") or (x.SecurityReference.ExchangeId ==
"NAS") or (x.SecurityReference.ExchangeId == "ASE"))]
    ## only select the stocks that (i) actively traded (aka market cap is not 0); (ii) pay
dividend or generate cashflows (aka total yield is not 0); (iii) have stocks issued in the past
12 months; (iv) traded in NYS, NAS, or ASE stock exchange.

        # Sorting by net payout.
        sorted_by_payout = sorted(fine, key = lambda x: ( (x.ValuationRatios.TotalYield *
(x.MarketCap)) - \
                                                            (x.FinancialStatements.CashFlowStat
ement.CommonStockIssuance.TwelveMonths / (x.MarketCap))), reverse=True)

        if len(sorted_by_payout) >= self.quantile:
        ## check if there if more than 10 quantiles generated
            decile = int(len(sorted_by_payout) / self.quantile)
        ## if there is 32 items in “sorted_by_payout”, divided by 10 quantiles, equals 3.2 >
then int( ) > equal to 3 > this is the number of items in the 1st quantile, that we will buy
            self.long = [x.Symbol for x in sorted_by_payout[:decile]]

        return self.long
        ## Fine universe will output with a list of securities that we will long

    def OnData(self, data):
        if not self.selection_flag:
            return
        self.selection_flag = False

        stocks_invested = [x.Key for x in self.Portfolio if x.Value.Invested]
        for symbol in stocks_invested:
            if symbol not in self.long:
                self.Liquidate(symbol)
```

```
        for symbol in self.long:
            if symbol in data and data[symbol]:
                self.SetHoldings(symbol, 1 / len(self.long))  ## equal weighting

        self.long.clear()

    def Selection(self):
        if self.Time.month == 6:
            self.selection_flag = True  ## only start to select stocks to long in June

# 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	0.020%	Sharpe Ratio	0.399
Total Trades	1038	Average Win	0.48%
Average Loss	-0.54%	Compounding Annual Return	7.123%
Drawdown	54.700%	Expectancy	0.437
Net Profit	390.227%	Loss Rate	24%
Win Rate	76%	Profit-Loss Ratio	0.89
Alpha	0.009	Beta	0.891
Annual Standard Deviation	0.153	Annual Variance	0.023
Information Ratio	0.059	Tracking Error	0.053
Treynor Ratio	0.068	Total Fees	$136.30
Estimated Strategy Capacity	$92000000.00	Lowest Capacity Asset	NVS RULY784EQ6AT

Fig 2. Performance Metrics

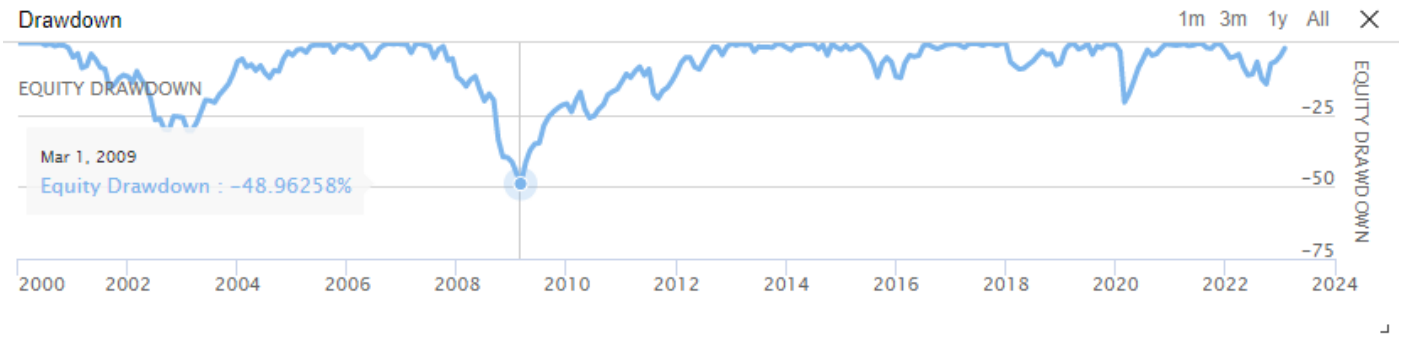

Fig 3. Drawdown

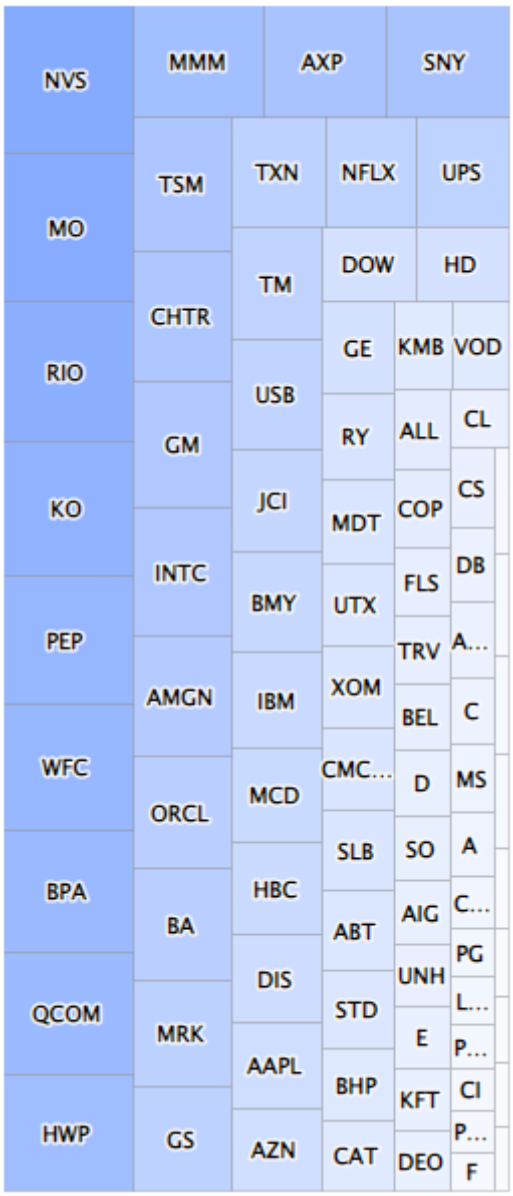

Fig 4. Assets Sales Volume