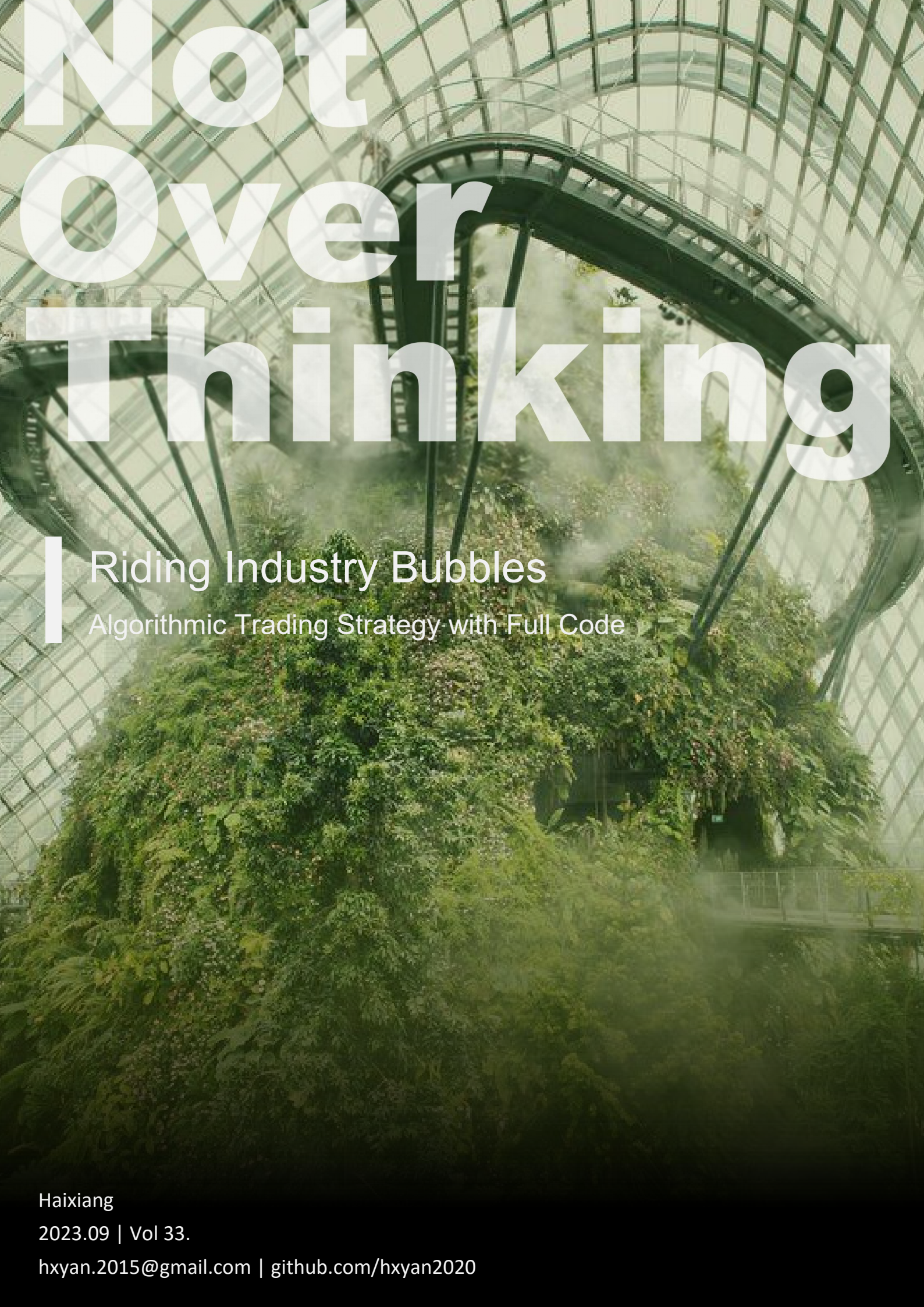

Not Over Thinking

Riding Industry Bubbles

Algorithmic Trading Strategy with Full Code

Haixiang

2023.09 | Vol 33.

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

STRATEGY & ECONOMIC RATIONALE

The investment universe consists of equity industry funds (or ETFs), which are proxy for equity industry indexes. An investor uses ten years of past data to calculate the industry's alpha based on the CAPM model (from the regression model industry_return = alpha + beta*market return, it is possible to use alternative models like the Fama/French 3 factor model).

A bubble in an industry is detected if the industry's alpha is statistically significant (academic source paper uses a 97,5% significance threshold, but it is possible to use other values).

The investor is long in each industry experiencing a bubble by applying 1/N rule (investment is divided equally between industries in a bubble). If no bubble is detected, then he/she makes no investment. Data examination, alpha calculation, and portfolio rebalancing is done monthly.

BUY	SELL
industry experiencing a bubble	The opposite

PARAMETER & VARIABLES

PARAMETER	VALUE
MARKETS TRADED	Equity
FINANCIAL INSTRUMENTS	ETFs, funds
REGION	Global
PERIOD OF REBALANCING	Monthly
NO. OF TRADED INSTRUMENTS	5
WEIGHTING	Equal weighting
LOOKBACK PERIODS	Month
LONG/SHORT	Long only

ALGORITHM

```
import numpy as np
from AlgorithmImports import *
import statsmodels.api as sm

class RidingIndustryBubbles(QCAlgorithm):

    def Initialize(self):
        self.SetStartDate(2008, 1, 1)
        self.SetCash(100000)

        self.spy = 'SPY'
        self.symbols = ['XLF', 'XLV', 'XLP', 'XLY', 'XLI', 'XLE', 'XLB', 'XLK', 'XLU']
        self.period = 10 * 12 * 21
        self.SetWarmUp(self.period)

        # Daily price data.
        self.data = {}
```

```
        for symbol in self.symbols + [self.spy]:
            data = self.AddEquity(symbol, Resolution.Daily)
            self.data[symbol] = RollingWindow[float](self.period)

        self.recent_month = -1

    def OnData(self, data):
        # Store daily price data.
        for symbol in self.symbols + [self.spy]:
            symbol_obj = self.Symbol(symbol)
            if symbol_obj in data and data[symbol_obj]:
                self.data[symbol].Add(data[symbol_obj].Value)

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

        if not self.data[self.spy].IsReady and self.spy in data: return

        market_closes = [x for x in self.data[self.spy]]
        separete_months = [market_closes[x:x+21] for x in range(0, len(market_closes),21)]
        market_monthly_returns = []
        for month in separete_months:
            month_of_prices = [x for x in month]
            market_monthly_returns.append(month_of_prices[0] / month_of_prices[-1] - 1)

        # Prepared for regression.
        market_monthly_returns = np.array(market_monthly_returns).T
        market_monthly_returns = sm.add_constant(market_monthly_returns)

        t_stat = {}
        for symbol in self.symbols:
            if self.data[symbol].IsReady and symbol in data:
                closes = [x for x in self.data[symbol]]
                separete_months = [closes[x:x+21] for x in range(0, len(closes),21)]
                etf_monthly_returns = []
                for month in separete_months:
                    month_of_prices = [x for x in month]
                    etf_monthly_returns.append(month_of_prices[0] / month_of_prices[-1] -
1)

                # alpha t-stat calc.
                model = sm.OLS(etf_monthly_returns, market_monthly_returns)

                results = model.fit()
                alpha_tstat = results.tvalues[0]
                alpha_pvalue = results.pvalues[0]
                t_stat[symbol] = (alpha_tstat, alpha_pvalue)

        long = []
        if len(t_stat) != 0:
            long = [x[0] for x in t_stat.items() if x[1][0] >= 2 and x[1][1] >= 0.025] #
The result is statistically significant, by the standards of the study, when p ≤ α
```

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

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

BACKTESTING PERFORMANCE

Fig 1. Overall Performance

PSR	0.068%	Sharpe Ratio	0.317
Total Trades	108	Average Win	2.78%
Average Loss	-2.33%	Compounding Annual Return	3.403%
Drawdown	23.300%	Expectancy	0.423
Net Profit	66.381%	Loss Rate	35%
Win Rate	65%	Profit-Loss Ratio	1.19
Alpha	0.014	Beta	0.18
Annual Standard Deviation	0.086	Annual Variance	0.007
Information Ratio	-0.298	Tracking Error	0.16
Treynor Ratio	0.151	Total Fees	$793.65
Estimated Strategy Capacity	$73000000.00	Lowest Capacity Asset	XLK RGRPZX100F39

Fig 2. Performance Metrics

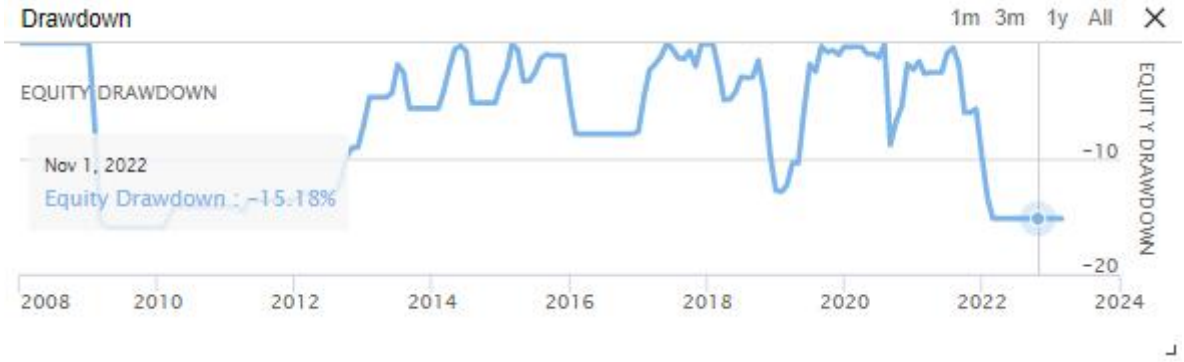

Fig 3. Drawdown

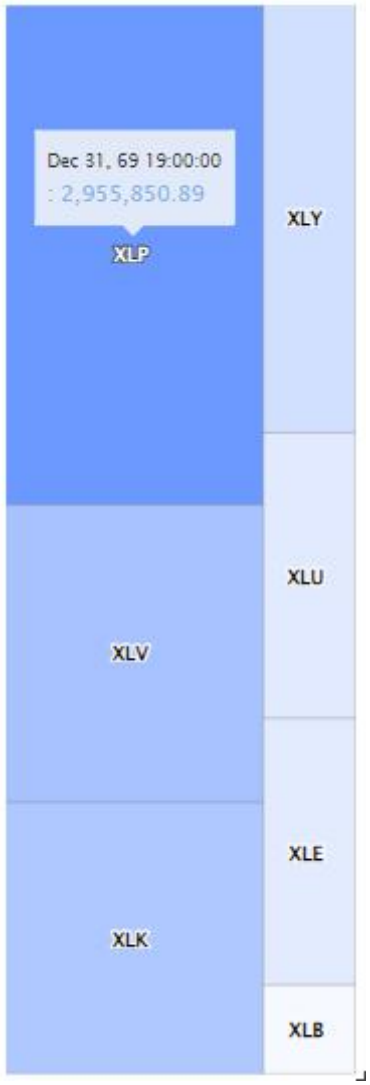

Fig 4. Assets Sales Volume