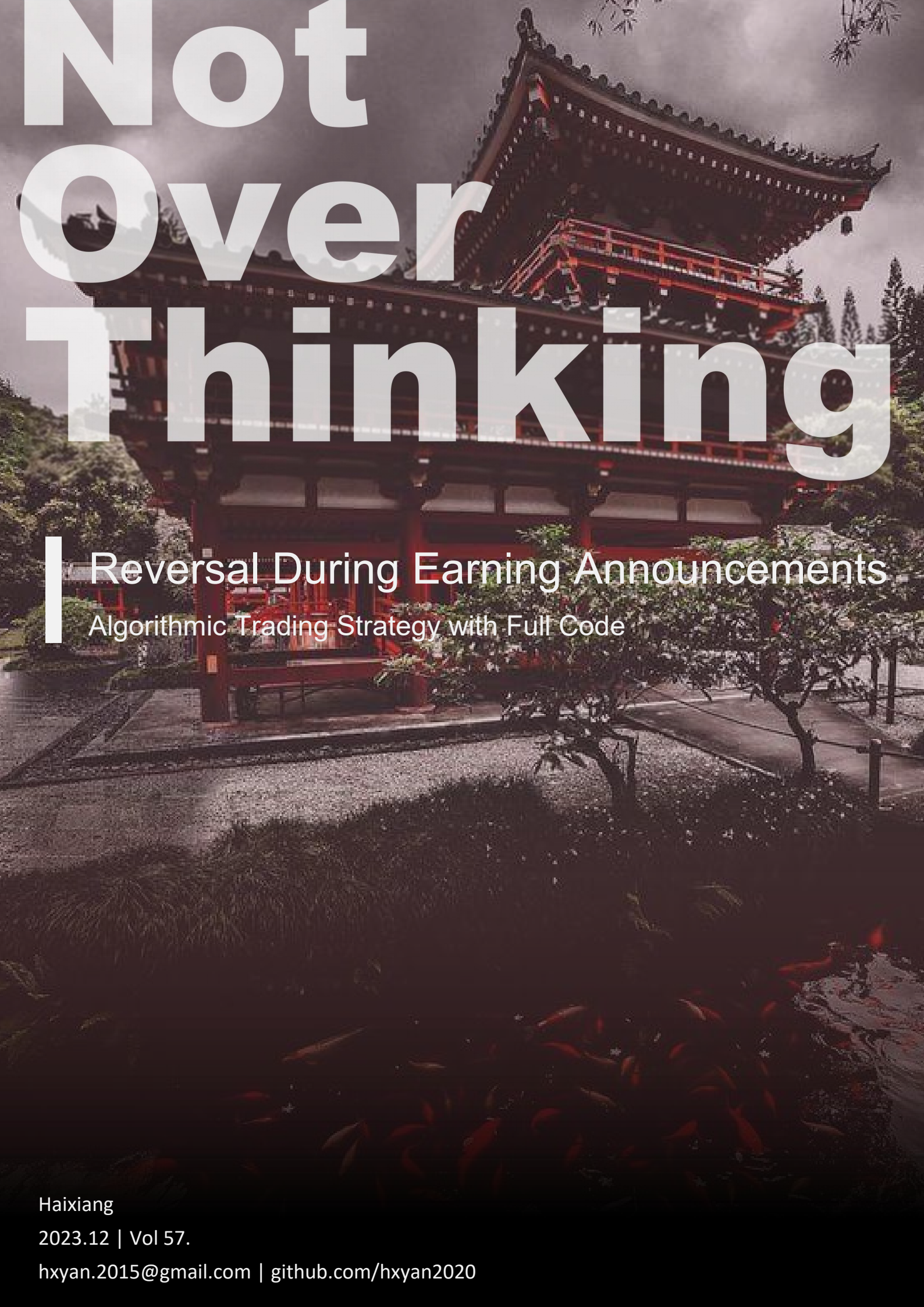

Not Over Thinking

Reversal During Earning Announcements

Algorithmic Trading Strategy with Full Code

Haixiang

2023.12 | Vol 57.

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

STRATEGY & ECONOMIC RATIONALE

The investment universe consists of stocks listed at NYSE, AMEX, and NASDAQ, whose daily price data are available at the CRSP database. Earnings-announcement dates are collected from Compust at. Firstly, the investor sorts stocks into quintiles based on firm size. Then he further sorts the stocks in the top quintile (the biggest) into quintiles based on their average returns in the 3-day window between t-4 and t-2, where t is the day of the earnings announcement. The inve stor goes long on the bottom quintile (past losers) and short on the top quintile (past winners) and holds the stocks during the 3-day window between t-1, t, and t+1. Stocks in the portfolios are weighted equally.

BUY	SELL
goes long on the bottom quin tile (past losers)	short on the top quintile (past winners)

PARAMETER & VARIABLES

PARAMETER	VALUE
MARKETS TRADED	Equity
FINANCIAL INSTRUMENTS	Stocks
REGION	United States
PERIOD OF REBALANCING	Daily
NO. OF TRADED INSTRUMENTS	1000
WEIGHTING	Equal weighting
LOOKBACK PERIODS	Depends
LONG/SHORT	Long & short

ALGORITHM

<data_tools.py>

```
from AlgorithmImports import *

import numpy as np
import statsmodels.api as sm

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

# NOTE: Manager for new trades. It's represented by certain count of equally weighted brackets for long and short positions.
# If there's a place for new trade, it will be managed for time of holding period.
class TradeManager():
    def __init__(self, algorithm, long_size, short_size, holding_period):
        self.algorithm = algorithm  # algorithm to execute orders in.

        self.long_size = long_size
        self.short_size = short_size

        self.long_len = 0
        self.short_len = 0
```

```
        # Arrays of ManagedSymbols
        self.symbols = []

        self.holding_period = holding_period    # Days of holding.

    # Add stock symbol object
    def Add(self, symbol, long_flag):
        # Open new long trade.
        managed_symbol = ManagedSymbol(symbol, self.holding_period, long_flag)

        if long_flag:
            # If there's a place for it.
            if self.long_len < self.long_size:
                self.symbols.append(managed_symbol)
                self.algorithm.SetHoldings(symbol, 1 / self.long_size)
                self.long_len += 1
            else:
                self.algorithm.Log("There's not place for additional trade.")

        # Open new short trade.
        else:
            # If there's a place for it.
            if self.short_len < self.short_size:
                self.symbols.append(managed_symbol)
                self.algorithm.SetHoldings(symbol, - 1 / self.short_size)
                self.short_len += 1
            else:
                self.algorithm.Log("There's not place for additional trade.")

    # Decrement holding period and liquidate symbols.
    def TryLiquidate(self):
        symbols_to_delete = []
        for managed_symbol in self.symbols:
            managed_symbol.days_to_liquidate -= 1

            # Liquidate.
            if managed_symbol.days_to_liquidate == 0:
                symbols_to_delete.append(managed_symbol)
                self.algorithm.Liquidate(managed_symbol.symbol)

                if managed_symbol.long_flag: self.long_len -= 1
                else: self.short_len -= 1

        # Remove symbols from management.
        for managed_symbol in symbols_to_delete:
            self.symbols.remove(managed_symbol)

    def LiquidateTicker(self, ticker):
        symbol_to_delete = None
        for managed_symbol in self.symbols:
            if managed_symbol.symbol.Value == ticker:
                self.algorithm.Liquidate(managed_symbol.symbol)
                symbol_to_delete = managed_symbol
                if managed_symbol.long_flag: self.long_len -= 1
                else: self.short_len -= 1

                break

        if symbol_to_delete: self.symbols.remove(symbol_to_delete)
        else: self.algorithm.Debug("Ticker is not held in portfolio!")

class ManagedSymbol():
    def __init__(self, symbol, days_to_liquidate, long_flag):
```

```
        self.symbol = symbol
        self.days_to_liquidate = days_to_liquidate
        self.long_flag = long_flag
```

<main.py>

```
import data_tools
from AlgorithmImports import *
import numpy as np
from collections import deque

class ReversalDuringEarningsAnnouncements(QCAlgorithm):

    def Initialize(self):
        self.SetStartDate(2010, 1, 1) # earnings dates start in 2010
        self.SetCash(100000)

        self.ear_period = 4
        self.symbol = self.AddEquity('SPY', Resolution.Daily).Symbol

        # Daily price data.
        self.data = {}

        # Import earnigns data.
        self.earnings_data = {}

        # Available symbols from earning_dates dataset.
        self.tickers:Set(str) = set()

        self.first_date:datetime.date|None = None

        earnings_data:str = self.Download('data.quantpedia.com/backtesting_data/economic/earnin
gs_dates_eps.json')
        earnings_data_json:list[dict] = json.loads(earnings_data)

        for obj in earnings_data_json:
            date:datetime.date = datetime.strptime(obj['date'], "%Y-%m-%d").date()
            self.earnings_data[date] = []

            if not self.first_date: self.first_date = date

            for stock_data in obj['stocks']:
                ticker:str = stock_data['ticker']

                self.earnings_data[date].append(ticker)
                self.tickers.add(ticker)

        # EAR history for previous quarter used for statistics.
        self.ear_previous_quarter = []
        self.ear_actual_quarter = []

        # 5 equally weighted brackets for traded symbols. - 20 symbols long , 20 for short, 3 d
ays of holding.
        self.trade_manager = data_tools.TradeManager(self, 20, 20, 3)

        self.month:int = 0
        self.selection_flag = False
        self.rebalance_flag = False
        self.UniverseSettings.Resolution = Resolution.Daily
        self.AddUniverse(self.CoarseSelectionFunction)
        self.Schedule.On(self.DateRules.MonthEnd(self.symbol), self.TimeRules.AfterMarketOpen(s
elf.symbol), self.Selection)

    def OnSecuritiesChanged(self, changes):
```

```
        for security in changes.AddedSecurities:
            security.SetFeeModel(data_tools.CustomFeeModel())
            security.SetLeverage(5)

    def CoarseSelectionFunction(self, coarse):
        # update daily prices
        for stock in coarse:
            symbol = stock.Symbol

            if symbol in self.data:
                self.data[symbol].Add(stock.AdjustedPrice)

        if not self.selection_flag:
            return Universe.Unchanged
        self.selection_flag = False

        selected = [x.Symbol for x in coarse if x.Symbol.Value in self.tickers]

        for symbol in selected:
            if symbol in self.data:
                continue

            self.data[symbol] = RollingWindow[float](self.ear_period)
            history = self.History(symbol, self.ear_period, Resolution.Daily)
            if history.empty:
                self.Log(f"Not enough data for {symbol} yet")
                continue

            closes = history.loc[symbol].close
            for time, close in closes.iteritems():
                self.data[symbol].Add(close)

        return selected

    def OnData(self, data):
        date_to_lookup = (self.Time + timedelta(days=1)).date()

        # Liquidate opened symbols after three days.
        self.trade_manager.TryLiquidate()

        ret_t4_t2 = {}

        for symbol in self.data:
            # Data is ready.
            if self.data[symbol].IsReady:
                # Earnings is in next two day for the symbol.
                if date_to_lookup in self.earnings_data and symbol.Value in self.earnings_data
[date_to_lookup]:
                    closes = [x for x in self.data[symbol]]
                    # Calculate t-4 to t-2 return.
                    ret = (closes[0] - closes[-1]) / closes[-1]
                    ret_t4_t2[symbol] = ret

                    # Store return in this month's history.
                    self.ear_actual_quarter.append(ret)

        # Wait until we have history data for previous three months.
        if len(self.ear_previous_quarter) != 0:
            # Sort by EAR.
            ear_values = self.ear_previous_quarter
            top_ear_quintile = np.percentile(ear_values, 80)
            bottom_ear_quintile = np.percentile(ear_values, 20)

            # Store symbol to set.
```

```
            long = [x[0] for x in ret_t4_t2.items() if x[1] <= bottom_ear_quintile and x[0] in
data and data[x[0]]]
            short = [x[0] for x in ret_t4_t2.items() if x[1] >= top_ear_quintile and x[0] in da
ta and data[x[0]]]

            # Open new trades.
            for symbol in long:
                self.trade_manager.Add(symbol, True)
            for symbol in short:
                self.trade_manager.Add(symbol, False)

    def Selection(self):
        # There is no earnings data yet.
        if self.Time.date() < self.first_date:
            return

        self.selection_flag = True

        # Every three months.
        if self.month % 3 == 0:
            # Save quarter history.
            self.ear_previous_quarter = [x for x in self.ear_actual_quarter]
            self.ear_actual_quarter.clear()

        self.month += 1
```

BACKTESTING PERFORMANCE

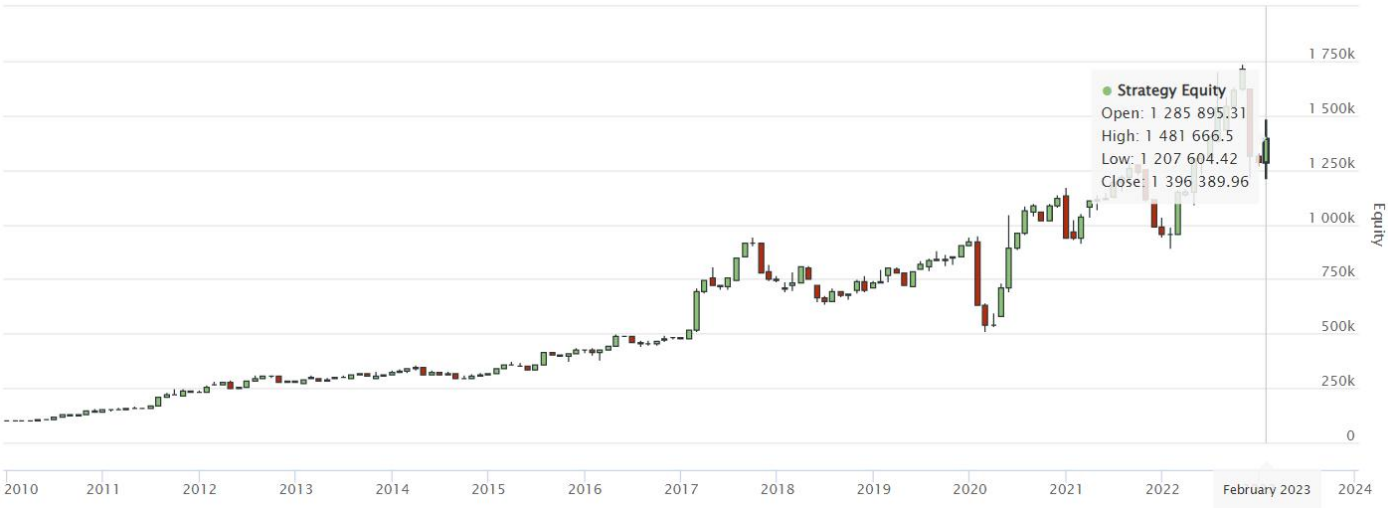

Fig 1. Overall Performance

Total Trades	32273	Average Win	0.37%
Average Loss	-0.35%	Compounding Annual Return	22.158%
Drawdown	46.300%	Expectancy	0.052
Net Profit	1296.390%	Sharpe Ratio	0.745
Probabilistic Sharpe Ratio	7.016%	Loss Rate	49%
Win Rate	51%	Profit-Loss Ratio	1.05
Alpha	0.169	Beta	0.128
Annual Standard Deviation	0.244	Annual Variance	0.059
Information Ratio	0.317	Tracking Error	0.274
Treynor Ratio	1.421	Total Fees	$51199.30
Estimated Strategy Capacity	$7000.00	Lowest Capacity Asset	LTPBV VTER7CYO778L

Fig 2. Performance Metrics