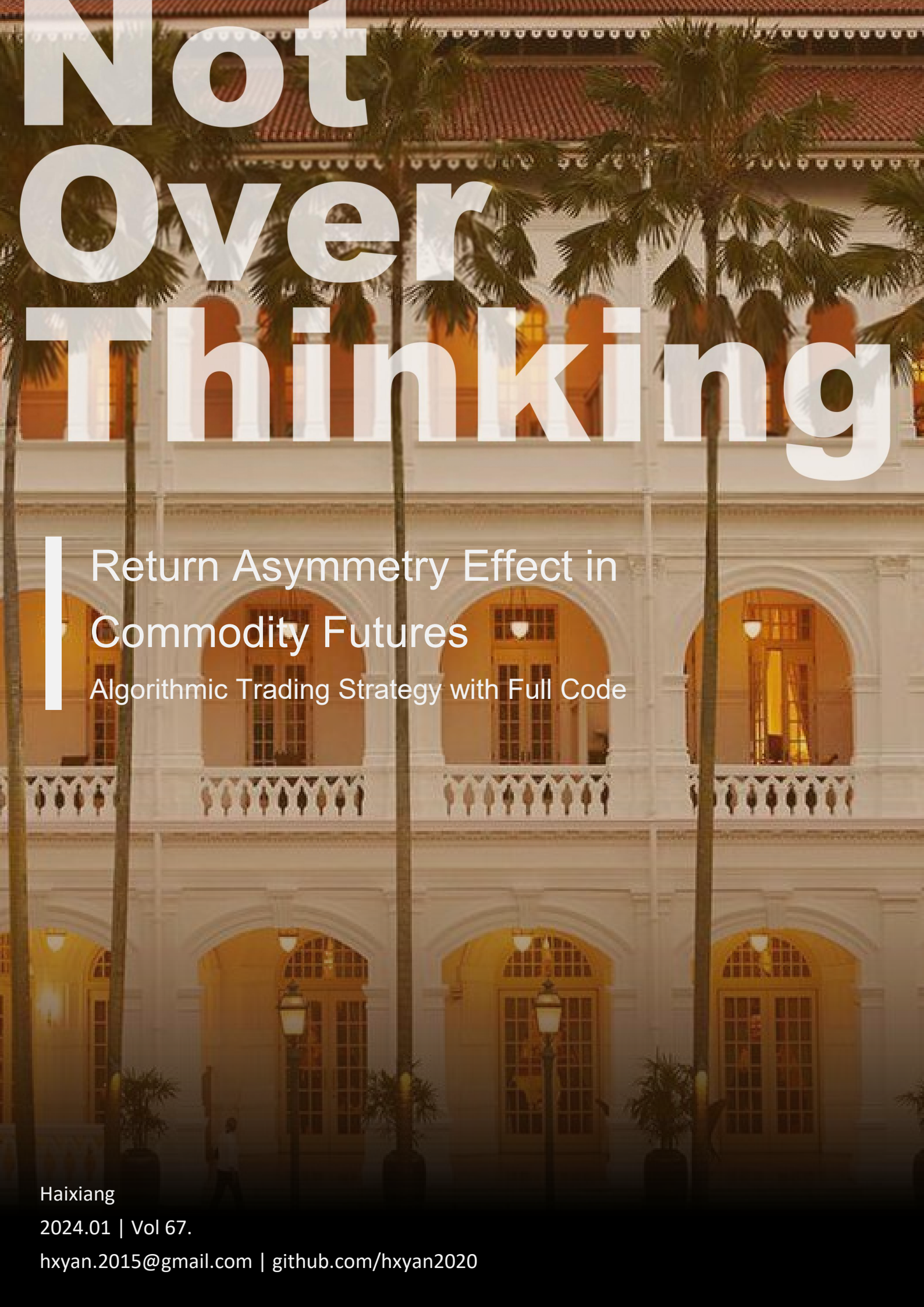

Not Over Thinking

Return Asymmetry Effect in Commodity Futures

Algorithmic Trading Strategy with Full Code

Haixiang

2024.01 | Vol 67.

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

STRATEGY & ECONOMIC RATIONALE

The investment universe consists of 22 commodity futures, namely: soybean oil, corn, cocoa, cotton, feeder cattle, gold, copper, heating oil, coffee, live cattle, lean hogs, natural gas, oats, orange juice, palladium, platinum, soybean, sugar, silver, soybean meal, wheat, and crude oil. Firstly, at the beginning of each month, construct the asymmetry measure (IE) for each commodity based on the latest 260 daily returns using the following formula (the formula originally consists of theoretical density and integrals, however the solution is simple when empirical distribution is utilized): IE = (number of trading days when the daily return is greater than the average plus two standard deviations) – (number of trading days when the daily return is smaller than the average minus two standard deviations).

Then rank the commodities according to their IE. Buy the bottom seven commodities with the lowest IE in the previous month and sell the top seven commodities with the highest IE in the previous month. Weigh the portfolio equally and rebalance monthly.

BUY	SELL
Buy the bottom seven commodities with the lowest IE in the previous month	sell the top seven commodities with the highest IE in the previous month.

PARAMETER & VARIABLES

PARAMETER	VALUE
MARKETS TRADED	Commodity
FINANCIAL INSTRUMENTS	Futures
REGION	Global
PERIOD OF REBALANCING	Monthly
NO. OF TRADED INSTRUMENTS	22
WEIGHTING	Equal weighting
LOOKBACK PERIODS	Month
LONG/SHORT	Long & short

ALGORITHM

```
from AlgorithmImports import *
class ReturnAsymmetryEffectInCommodityFutures(QCAlgorithm):

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

        self.tickers = [
            "CME_S1",   # Soybean Futures, Continuous Contract
            "CME_W1",   # Wheat Futures, Continuous Contract
            "CME_SM1",  # Soybean Meal Futures, Continuous Contract
            "CME_BO1",  # Soybean Oil Futures, Continuous Contract
            "CME_C1",   # Corn Futures, Continuous Contract
            "CME_O1",   # Oats Futures, Continuous Contract
            "CME_LC1",  # Live Cattle Futures, Continuous Contract
            "CME_FC1",  # Feeder Cattle Futures, Continuous Contract
            "CME_LN1",  # Lean Hog Futures, Continuous Contract
            "CME_GC1",  # Gold Futures, Continuous Contract
            "CME_SI1",  # Silver Futures, Continuous Contract
```

```
            "CME_PL1",  # Platinum Futures, Continuous Contract
            "CME_CL1",  # Crude Oil Futures, Continuous Contract
            "CME_HG1",  # Copper Futures, Continuous Contract
            "CME_LB1",  # Random Length Lumber Futures, Continuous Contract
            # "CME_NG1",  # Natural Gas (Henry Hub) Physical Futures, Continuous Contract
            "CME_PA1",  # Palladium Futures, Continuous Contract
            "CME_RR1",  # Rough Rice Futures, Continuous Contract
            "CME_RB2",  # Gasoline Futures, Continuous Contract
            "CME_KW2",  # Wheat Kansas, Continuous Contract

            "ICE_CC1",  # Cocoa Futures, Continuous Contract
            "ICE_CT1",  # Cotton No. 2 Futures, Continuous Contract
            "ICE_KC1",  # Coffee C Futures, Continuous Contract
            "ICE_O1",   # Heating Oil Futures, Continuous Contract
            "ICE_OJ1",  # Orange Juice Futures, Continuous Contract
            "ICE_SB1"   # Sugar No. 11 Futures, Continuous Contract
            "ICE_RS1",  # Canola Futures, Continuous Contract
            "ICE_GO1",  # Gas Oil Futures, Continuous Contract
            "ICE_WT1",  # WTI Crude Futures, Continuous Contract
        ]

        self.data = {} # storing objects of SymbolData class keyed by comodity symbols

        self.period = 261 # need 261 daily prices, to calculate 260 daily returns
        self.buy_count = 7 # buy n comodities on each rebalance
        self.sell_count = 7 # sell n comodities on each rebalance

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

        # subscribe to futures contracts
        for ticker in self.tickers:
            security = self.AddData(QuantpediaFutures, ticker, Resolution.Daily)
            security.SetFeeModel(CustomFeeModel())
            security.SetLeverage(5)

            self.data[security.Symbol] = SymbolData(self.period)

        self.rebalance_flag = False
        self.Schedule.On(self.DateRules.MonthStart(self.symbol), self.TimeRules.BeforeMarketClo
se(self.symbol, 0), self.Rebalance)

    def OnData(self, data):
        # update daily closes
        for symbol in self.data:
            if symbol in data and data[symbol]:
                close = data[symbol].Value
                self.data[symbol].update_closes(close)

        # rebalance monthly
        if not self.rebalance_flag:
            return
        self.rebalance_flag = False

        IE = {}

        for symbol, symbol_obj in self.data.items():
            # check if comodity has ready prices
            if not symbol_obj.is_ready():
                continue

            # calculate IE
            IE_value = symbol_obj.calculate_IE()

            # store IE value under comodity symbol
```

```
            IE[symbol] = IE_value

        # make sure, there are enough comodities for rebalance
        if len(IE) < (self.buy_count + self.sell_count):
            return

        # sort commodities based on IE values
        sorted_by_IE = [x[0] for x in sorted(IE.items(), key=lambda item: item[1])]

        # select long and short parts
        long = sorted_by_IE[:self.buy_count]
        short = sorted_by_IE[-self.sell_count:]

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

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

        for symbol in short:
            self.SetHoldings(symbol, -1 / self.sell_count)

    def Rebalance(self):
        self.rebalance_flag = True
            class SymbolData():
    def __init__(self, period):
        self.closes = RollingWindow[float](period)

    def update_closes(self, close):
        self.closes.Add(close)

    def is_ready(self):
        return self.closes.IsReady

    def calculate_IE(self):
        closes = np.array([x for x in self.closes])
        daily_returns = (closes[:-1] - closes[1:]) / closes[1:]

        average_daily_returns = np.average(daily_returns)
        two_daily_returns_std = 2 * np.std(daily_returns)

        avg_plus_two_std = average_daily_returns + two_daily_returns_std
        avg_minus_two_std = average_daily_returns - two_daily_returns_std

        over_avg_plus_two_std = 0 # counting number of daily returns, which were over avg_plus_
two_std
        under_avg_minus_two_std = 0 # counting number of daily returns, which were under avg_mi
nus_two_std

        for daily_return in daily_returns:
            if daily_return > avg_plus_two_std:
                over_avg_plus_two_std += 1
            elif daily_return < avg_minus_two_std:
                under_avg_minus_two_std += 1

        IE_value = over_avg_plus_two_std - under_avg_minus_two_std

        return IE_value
        # Quantpedia data.# NOTE: IMPORTANT: Data order must be ascending (datewise)class Quant
pediaFutures(PythonData):
    def GetSource(self, config, date, isLiveMode):
```

```
        return SubscriptionDataSource("data.quantpedia.com/backtesting_data/futures/{0}.csv".fo
rmat(config.Symbol.Value), SubscriptionTransportMedium.RemoteFile, FileFormat.Csv)

    def Reader(self, config, line, date, isLiveMode):
        data = QuantpediaFutures()
        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['back_adjusted'] = float(split[1])
        data['spliced'] = float(split[2])
        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

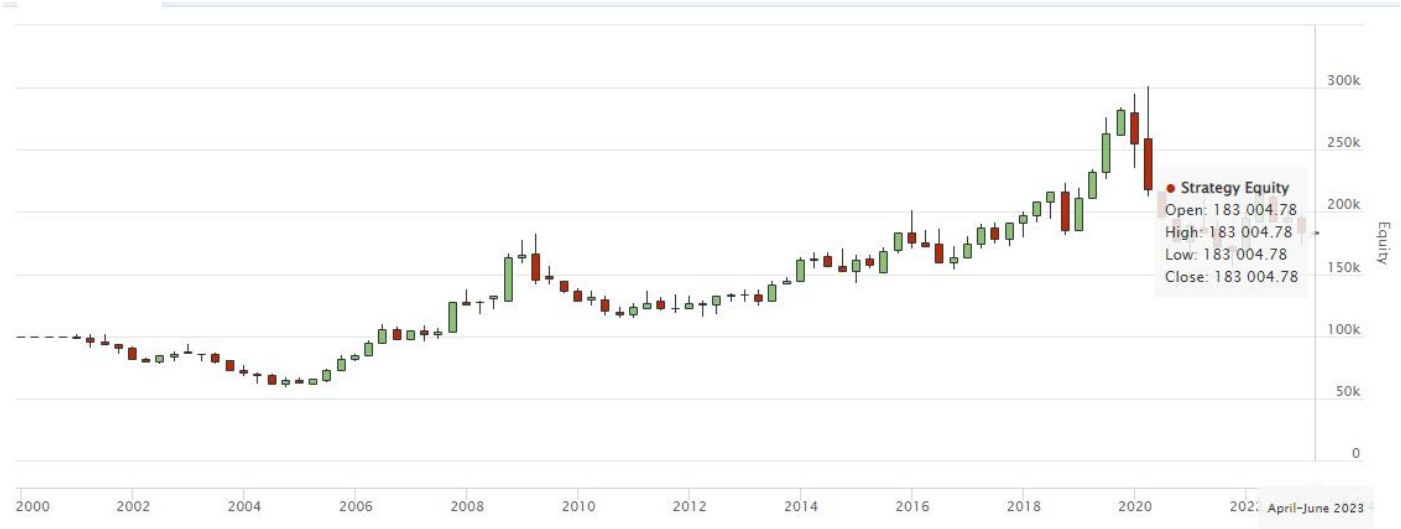

Fig 1. Overall Performance

Total Trades	3998	Average Win	0.80%
Average Loss	-0.78%	Compounding Annual Return	2.631%
Drawdown	48.200%	Expectancy	0.057
Net Profit	83.005%	Sharpe Ratio	0.204
Probabilistic Sharpe Ratio	0.000%	Loss Rate	48%
Win Rate	52%	Profit-Loss Ratio	1.03
Alpha	0.028	Beta	-0.014
Annual Standard Deviation	0.134	Annual Variance	0.018
Information Ratio	-0.145	Tracking Error	0.212
Treynor Ratio	-1.947	Total Fees	$1986.76
Estimated Strategy Capacity	$0	Lowest Capacity Asset	CME_LB1.QuantpediaFutures 2S
Portfolio Turnover	3.23%		

Fig 2. Performance Metrics