Not Over Thinking

Post-Earnings Announcement Effect

Algorithmic Trading Strategy with Full Code

Haixiang

2023.02 | Vol 10.

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

STRATEGY & ECONOMIC RATIONALE

The selection of stocks for investment is based on the NYSE, AMEX, and NASDAQ, excluding financial and utility companies, and stocks that cost less than $5.

The stocks are divided into quintiles according to their EAR and SUE. To avoid any look-ahead bias, data from the previous quarter are used for sorting the stocks.

All the stocks in each quintile are given equal weightage, and the portfolio is rebalanced every quarter.

The trading strategy only involves the long leg since the research paper suggests that the long leg contributes significantly to the performance of the strategy.

LONG	SHORT
goes long stocks from the intersection of top SUE and EAR quintiles the second day after the actual earnings announcement and holds the portfolio one quarter (or 60 working days).	goes short stocks from the intersection of the bottom SUE and EAR quintiles the second day after the actual earnings announcement and holds the portfolio one quarter (or 60 working days).

PARAMETER & VARIABLES

Two factors are used: EAR (Earnings Announcement Return) and SUE (Standardized Unexpected Earnings).

- SUE is constructed by dividing the earnings surprise (calculated as actual earnings minus expected earnings; expected earnings are computed using a seasonal random walk model with drift) by the standard deviation of earnings surprises.

- EAR is the abnormal return for firms recorded over a three-day window centered on the last announcement date, in excess of the return of a portfolio of firms with similar risk exposures.

PARAMETER	VALUE
MARKETS TRADED	NYSE, AMEX, NASDAQ
FINANCIAL INSTRUMENTS	Stocks
PERIOD OF REBALANCING	Quarterly
NO. OF TRADED INSTRUMENTS	1000
PRICE LIMIT	<$5
LOOK-AHEAD BIAS AVOIDED?	Yes
WEIGHTING	Equal in each quantile
HOLDING PERIODS	60 working days
LONG/SHORT	Long Only

DATA SOURCE

- Universe consists of stocks, with earnings data from https://www.nasdaq.com/market-activity/earnings available.
    - At least 4 years of seasonal earnings data is required to calculate earnigns surprise.

- At least 4 years of earnings surprise values are required for SUE calculation.

ALGORITHM

```
from AlgorithmImports import *
import numpy as np
from collections import deque
from pandas.tseries.offsets import BDay
from dateutil.relativedelta import relativedelta

## inherent from parent class QCAlgorithm
class PostEarningsAnnouncementEffect(QCAlgorithm):

    def Initialize(self):
        self.SetStartDate(2010, 1, 1)  ## did not set end date >will run till today
        self.SetCash(100000)

        self.earnings_surprise = {}
        self.min_seasonal_eps_period = 4  ## 4 years
        self.min_surprise_period = 4  ## 4 years

        self.long = []

        # SUE and EAR history for previous quarter used for statistics.
        self.sue_ear_history_previous = []
        self.sue_ear_history_actual = []
        ## prepared for rolling window, current/newest 3-month data will overwrites in sue_ear_
history_previous; since SUE and EAR are both calculated in 3-month window, hence can use same s
et of placeholders

        # EPS data keyed by tickers, which are keyed by dates
        self.eps_by_ticker = {}  ## for symbols that do not have EPS data, ignore them

        # daily price data
        self.price_data_with_date = {}
        self.price_period = 63
        ## 60-day holding period + 3-day calculation window after the announcement

        self.market = self.AddEquity('SPY', Resolution.Daily).Symbol
        self.price_data_with_date[self.market] = deque(maxlen=self.price_period)
        ## deque has the methods for adding and removing elements which can be invoked directly
 with arguments > it extract 63 days of data for SPY

        # parse earnings dataset
        self.first_date:datetime.date|None = None
        earnings_data:str = self.Download('data.quantpedia.com/backtesting_data/economic/earnin
gs_dates_eps.json')
        ## first download data and save in "str"

        earnings_data_json:list[dict] = json.loads(earnings_data)
        ## json.loads parse a valid JSON string and convert it into a Python Dictionary > earni
ngs_data_json becomes distionary

        for obj in earnings_data_json:
            date:datetime.date = datetime.strptime(obj['date'], "%Y-%m-%d").date()
            ## convert datetime format and save it under key "date"

            if not self.first_date: self.first_date = date
            ## if there is no self.first date, set it to "date"

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

```
                ## save ticker under key “ticker”


                if stock_data['eps'] == '':
                    continue
                ## good practice to check if the “eps” column is empty, before storing data in
it

                # initialize dictionary for dates for specific ticker
                if ticker not in self.eps_by_ticker:
                    self.eps_by_ticker[ticker] = {}

                # store EPS value keyed date, which is keyed by ticker
                self.eps_by_ticker[ticker][date] = float(stock_data['eps'])

        self.month = 12
        self.selection_flag = False
        self.UniverseSettings.Resolution = Resolution.Daily
        self.AddUniverse(self.CoarseSelectionFunction, self.FineSelectionFunction)
        self.Schedule.On(self.DateRules.MonthStart(self.market), self.TimeRules.AfterMarketOpen
(self.market), self.Selection)

    def OnSecuritiesChanged(self, changes):
    ## manage (i) AddedSecurities; (ii) RemovedSecurities
    ## if there is any change in the portfolio, the change object is saved in variable “change
s”
        for security in changes.AddedSecurities:
            security.SetFeeModel(CustomFeeModel())  ## calling class defined “CustomeFeeModel”
            security.SetLeverage(5)  ## for newly added securities, set the leverage to 5x

        # remove earnings surprise data so it remains consecutive
        for security in changes.RemovedSecurities:
            symbol = security.Symbol
            if symbol in self.earnings_surprise:
                del self.earnings_surprise[symbol]
                ## remove the earnings_surprise data for the deleted symbols

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

            if symbol in self.price_data_with_date:
            ## price_data_with_date saves the securities’ prices by date
                self.price_data_with_date[symbol].append((self.Time.date(), stock.AdjustedPric
e))

        if not self.selection_flag:
            return Universe.Unchanged
        self.selection_flag = False
        ## for securities not selected, they will have no impact on universe

        # filter only symbols, which have earnings data from csv
        selected = [x.Symbol for x in coarse if x.Symbol.Value in self.eps_by_ticker]

        # warmup price data
        for symbol in selected:
            if symbol in self.price_data_with_date:
            ## filter the symbols that have EPS data also have price data by date
                continue

            self.price_data_with_date[symbol] = deque(maxlen=self.price_period)
            history = self.History(symbol, self.price_period, Resolution.Daily)
```

```
            if history.empty:
                self.Log(f"Not enough data for {symbol} yet.")
                continue
            ## housekeeping check if there is enough history data to warm up

            closes = history.loc[symbol].close
            for time, close in closes.iteritems():
                self.price_data_with_date[symbol].append((time.date(), close))

        # market price data is not ready yet
        if len(self.price_data_with_date[self.market]) != self.price_data_with_date[self.marke
t].maxlen:
            return Universe.Unchanged

        return [x for x in selected if len(self.price_data_with_date[x]) == self.price_data_wit
h_date[x].maxlen]
        ## final output from CoarseSelectionFunction function >> securities that have (i) EPS d
ata; (ii) price data by date; (iii) sufficient history data

    def FineSelectionFunction(self, fine):
        # SUE and EAR data
        sue_ear = {}

        current_date = self.Time.date()
        prev_three_months = current_date - relativedelta(months=3)

        for stock in fine:
            symbol = stock.Symbol
            ticker = symbol.Value

            recent_eps_data = None  ## placeholder

            # store all EPS data since previous three months window
            for date in self.eps_by_ticker[ticker]:
                if date < current_date and date >= prev_three_months:
                    EPS_value = self.eps_by_ticker[ticker][date]

                    # create tuple (EPS date, EPS value of specific stock)
                    recent_eps_data = (date, EPS_value)
                    break

            if recent_eps_data:  ## if recent_eps_data exists
                last_earnings_date = recent_eps_data[0]

                # get earnings history until previous earnings
                earnings_eps_history = [(x, self.eps_by_ticker[ticker][x]) for x in self.eps_by
_ticker[ticker] if x < last_earnings_date]

                # seasonal earnings for previous years
                # prev_month_date = last_earnings_date - relativedelta(months=1)
                # next_month_date = last_earnings_date + relativedelta(months=1)
                # month_range = [prev_month_date.month, last_earnings_date.month, next_month_da
te.month]

                # seasonal_eps_data = [x for x in earnings_eps_history if x[0].month in month_r
ange]
                seasonal_eps_data = [x for x in earnings_eps_history if x[0].month == last_earn
ings_date.month]

                if len(seasonal_eps_data) >= self.min_seasonal_eps_period:
                ## min_seasonal_eps_period defined as 4
                    # make sure we have a consecutive seasonal data. Same months with one year
difference
                    year_diff = np.diff([x[0].year for x in seasonal_eps_data])
```

```
                    if all(x == 1 for x in year_diff):
                        # SUE calculation
                        seasonal_eps = [x[1] for x in seasonal_eps_data]
                        diff_values = np.diff(seasonal_eps)
                        drift = np.average(diff_values)

                        last_earnings_eps = seasonal_eps[-1]
                        expected_earnings = last_earnings_eps + drift
                        actual_earnings = recent_eps_data[1]

                        earnings_surprise = actual_earnings - expected_earnings

                        # initialize suprise data
                        if symbol not in self.earnings_surprise:
                            self.earnings_surprise[symbol] = []

                        # surprise data is ready.
                        elif len(self.earnings_surprise[symbol]) >= self.min_surprise_period:
                            earnings_surprise_std = np.std(self.earnings_surprise[symbol])
                            sue = earnings_surprise / earnings_surprise_std

                            # EAR calculation
                            min_day = last_earnings_date - BDay(2)
                            max_day = last_earnings_date + BDay(1)
                            stock_closes_around_earnings = [x for x in self.price_data_with_date[symbol] if x[0] >= min_day and x[0] <= max_day]
                            market_closes_around_earnings = [x for x in self.price_data_with_date[self.market] if x[0] >= min_day and x[0] <= max_day]

                            if len(stock_closes_around_earnings) == 4 and len(market_closes_around_earnings) == 4:
                                stock_return = stock_closes_around_earnings[-1][1] / stock_closes_around_earnings[0][1] - 1
                                market_return = stock_closes_around_earnings[-1][1] / stock_closes_around_earnings[0][1] - 1

                                ear = stock_return - market_return
                                sue_ear[symbol] = (sue, ear)

                                # store pair in this month's history
                                self.sue_ear_history_actual.append((sue, ear))

                        self.earnings_surprise[symbol].append(earnings_surprise)

        # wait until we have history data for previous three months.
        if len(sue_ear) != 0 and len(self.sue_ear_history_previous) != 0:
            # Sort by SUE and EAR.
            sue_values = [x[0] for x in self.sue_ear_history_previous]
            ear_values = [x[1] for x in self.sue_ear_history_previous]

            top_sue_quintile  = np.percentile(sue_values, 80)
            bottom_sue_quintile = np.percentile(sue_values, 20)

            top_ear_quintile = np.percentile(ear_values, 80)
            bottom_ear_quintile = np.percentile(ear_values, 20)

            self.long = [x[0] for x in sue_ear.items() if x[1][0] >= top_sue_quintile and x[1][1] >= top_ear_quintile]

        return self.long

    def OnData(self, data):
        # trade execution
        invested = [x.Key for x in self.Portfolio if x.Value.Invested]
```

```
        ##self object has a built-in attribute “.Portfolio”

        for symbol in invested:
            if symbol not in self.long:  ## if not labelled as long, liquidate it
                self.Liquidate(symbol)

        long_count = len(self.long)

        for symbol in self.long:
            if symbol in data and data[symbol]:
                self.SetHoldings(symbol, 1 / long_count)   ## equal weighting
                ## buy in the securities labelled “long”

        self.long.clear()  ## clear the long list once purchased, so do not purchase again

    def Selection(self):
        self.selection_flag = True

        # store new EAR and SUE values every three months
        if self.month % 3 == 0:
        ## ask for remainder > if = 0, means self.month is a multiple of 3

            # Save previous month history.
            self.sue_ear_history_previous = self.sue_ear_history_actual
            self.sue_ear_history_actual.clear()  ## prepare for next 3-month period’s input

        self.month += 1
        if self.month > 12:  ## when a year is ended, redstart from 1
            self.month = 1

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

BACKTESTING PERFORMANCE

Fig 1. Overall Performance

PSR	0.020%	Sharpe Ratio	0.399
Total Trades	1038	Average Win	0.48%
Average Loss	-0.55%	Compounding Annual Return	7.124%
Drawdown	54.700%	Expectancy	0.435
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

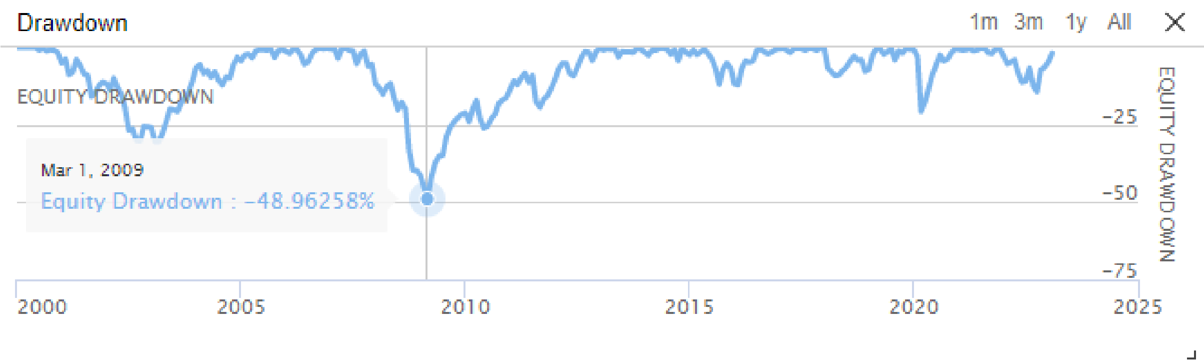

Fig 3. Drawdown

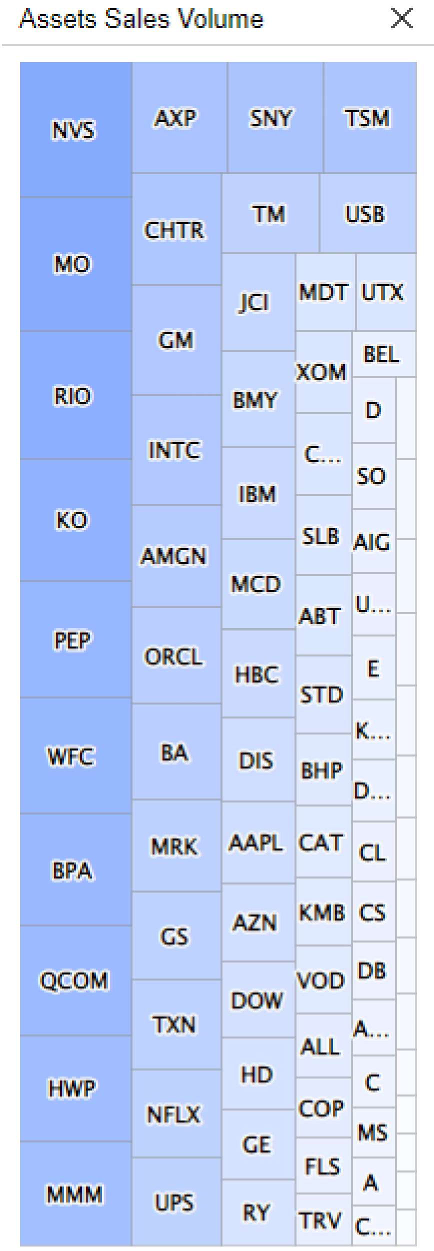

Fig 4. Assets Sales Volume