Not Over Thinking

Earnings Announcements Combined with Stock Repurchases

Algorithmic Trading Strategy with Full Code

Haixiang

2023.12 | Vol 55.

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

STRATEGY & ECONOMIC RATIONALE

The investment universe consists of stocks from NYSE/AMEX/Nasdaq (no ADRs, CEFs or REITs), bottom 25% of firms by market cap are dropped. Each quarter, the investor looks for companies that announce a stock repurchase program (with announced buyback for at least 5% of outstanding stocks) during days -30 to -15 before the earnings announcement date for each company. Investor goes long stocks with announced buybacks during days -10 to +15 around an earnings announcement. The portfolio is equally weighted and rebalanced daily.

BUY	SELL
goes long stocks with announced buybacks during days -10 to +15 around an earnings announcement	The opposite

PARAMETER & VARIABLES

PARAMETER	VALUE
MARKETS TRADED	Equity
FINANCIAL INSTRUMENTS	Stocks
REGION	United States
PERIOD OF REBALANCING	Daily
NO. OF TRADED INSTRUMENTS	100
WEIGHTING	Equal weighting
LOOKBACK PERIODS	Depends
LONG/SHORT	Long only

ALGORITHM

```
from AlgorithmImports import *
import numpy as np
#endregion

class EarningsAnnouncementsCombinedWithStockRepurchases(QCAlgorithm):

    def Initialize(self):
        self.SetStartDate(2011, 1, 1) # Buyback data strats at 2011
        self.SetCash(100000)

        self.fine = {}
        self.price = {}
        self.managed_symbols = []
        self.earnings_universe = []

        self.earnings = {}
        self.buybacks = {}

        self.max_traded_stocks = 40 # maximum number of trading stocks
        self.quantile = 4

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

        # load earnings dates
```

```
        csv_data = self.Download('data.quantpedia.com/backtesting_data/economic/earning_dates.c
sv')
        lines = csv_data.split('\r\n')

        for line in lines:
            line_split = line.split(';')
            date = line_split[0]

            if date == '' :
                continue

            date = datetime.strptime(date, "%Y-%m-%d").date()
            self.earnings[date] = []

            for ticker in line_split[1:]: # skip date in current line
                self.earnings[date].append(ticker)

                if ticker not in self.earnings_universe:
                    self.earnings_universe.append(ticker)

        # load buyback dates
        csv_data = self.Download('data.quantpedia.com/backtesting_data/equity/BUY_BACKS.csv')
        lines = csv_data.split('\r\n')

        for line in lines[1:]: # skip header
            line_split = line.split(';')
            date = line_split[0]

            if date == '' :
                continue

            date = datetime.strptime(date, "%d.%m.%Y").date()
            self.buybacks[date] = []

            for ticker in line_split[1:]: # skip date in current line
                self.buybacks[date].append(ticker)

        self.months_counter = 0
        self.selection_flag = False
        self.UniverseSettings.Resolution = Resolution.Daily
        self.AddUniverse(self.CoarseSelectionFunction, self.FineSelectionFunction)
        self.Schedule.On(self.DateRules.MonthStart(self.symbol), self.TimeRules.AfterMarketOpen
(self.symbol), self.Selection)

    def OnSecuritiesChanged(self, changes):
        for security in changes.AddedSecurities:
            security.SetFeeModel(CustomFeeModel())
            security.SetLeverage(5)

    def CoarseSelectionFunction(self, coarse):
        # update stocks last prices
        for stock in coarse:
            ticker = stock.Symbol.Value

            if ticker in self.earnings_universe:
                # store stock's last price
                self.price[ticker] = stock.AdjustedPrice

        # rebalance quarterly
        if not self.selection_flag:
            return Universe.Unchanged
        self.selection_flag = False

        # select stocks, which had spin off
```

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

        return selected

    def FineSelectionFunction(self, fine):
        fine = [x for x in fine if x.MarketCap != 0 and
                                   ((x.SecurityReference.ExchangeId == "NYS") or
                                   (x.SecurityReference.ExchangeId == "NAS") or
                                   (x.SecurityReference.ExchangeId == "ASE"))]

        if len(fine) < self.quantile:
            return Universe.Unchanged

        # exclude 25% stocks with lowest market capitalization
        quantile = int(len(fine) / self.quantile)
        sorted_by_market_cap = sorted(fine, key = lambda x: x.MarketCap)
        selected = sorted_by_market_cap[quantile:]
        self.fine = {x.Symbol.Value : x.Symbol for x in selected}

        return list(self.fine.values())

    def OnData(self, data:Slice) -> None:
        remove_managed_symbols = []
        # maybe there should be BDay(15)
        liquidate_date = self.Time.date() - timedelta(15)

        # check if bought stocks have 15 days after earnings annoucemnet
        for managed_symbol in self.managed_symbols:
            if managed_symbol.earnings_date >= liquidate_date:
                remove_managed_symbols.append(managed_symbol)

                # liquidate stock by selling it's quantity
                self.MarketOrder(managed_symbol.symbol, -managed_symbol.quantity)

        # remove liquidated stocks from self.managed_symbols
        for managed_symbol in remove_managed_symbols:
            self.managed_symbols.remove(managed_symbol)

        # maybe there should be BDay(10)
        after_current = self.Time.date() + timedelta(10)

        if after_current in self.earnings:
            # this stocks has earnings annoucement after 10 days
            stocks_with_earnings = self.earnings[after_current]

            # 30 days before earnings annoucement
            buyback_start = self.Time.date() - timedelta(20)
            # 15 days before earnings annoucement
            buyback_end = self.Time.date() - timedelta(5)

            stocks_with_buyback = [] # storing stocks with buyback in period -30 to -15 days be
fore earnings annoucement

            for buyback_date, tickers in self.buybacks.items():
                # check if buyback date is in period before earnings annoucement
                if buyback_date >= buyback_start and buyback_date <= buyback_end:
                    # iterate through each stock ticker for buyback date
                    for ticker in tickers:
                        # add stock ticker if it isn't already added, it has earnings annouceme
nt after 10 days and was selected in fine
                        if (ticker not in stocks_with_buyback) and (ticker in stocks_with_earni
ngs) and (ticker in self.fine):
                            stocks_with_buyback.append(self.fine[ticker])
```

```
            # buying stocks buyback in period -30 to -15 days before earnings annoucement
            # and stocks, which have earnings date -10 days before current date
            for symbol in stocks_with_buyback:
                # check if there is a place in Portfolio for trading current stock
                if not len(self.managed_symbols) < self.max_traded_stocks:
                    continue

                # calculate stock quantity
                weight = self.Portfolio.TotalPortfolioValue / self.max_traded_stocks
                quantity = np.floor(weight / self.price[symbol.Value])

                # go long stock
                self.MarketOrder(symbol, quantity)

                # store stock's ticker, earnings date and traded quantity
                if symbol in data and data[symbol]:
                    self.managed_symbols.append(ManagedSymbol(symbol, after_current, quantity))

    def Selection(self):
        # quarterly selection
        if self.months_counter % 3 == 0:
            self.selection_flag = True
        self.months_counter += 1

class ManagedSymbol():
    def __init__(self, symbol, earnings_date, quantity):
        self.symbol = symbol
        self.earnings_date = earnings_date
        self.quantity = quantity

# 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	305	Average Win	0.34%
Average Loss	-0.02%	Compounding Annual Return	14.997%
Drawdown	68.800%	Expectancy	4.923
Net Profit	420.323%	Sharpe Ratio	0.515
Probabilistic Sharpe Ratio	1.659%	Loss Rate	74%
Win Rate	26%	Profit-Loss Ratio	21.47
Alpha	0.004	Beta	1.632
Annual Standard Deviation	0.289	Annual Variance	0.084
Information Ratio	0.319	Tracking Error	0.189
Treynor Ratio	0.091	Total Fees	$68.68
Estimated Strategy Capacity	$970000.00	Lowest Capacity Asset	LBTYB SZC2UFSQNK9X

Fig 2. Performance Metrics