Not Over Thinking

Subsequent to the Submission of 13F Filings

Algorithmic Trading Strategy with Full Code

Haixiang

2023.07 | Vol 17.

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

STRATEGY & ECONOMIC RATIONALE

Create a cohort of actively involved mutual fund managers responsible for actively managing their portfolios. Employ 13F filings to identify the most favorable stocks, commonly referred to as "best idea" stocks, for each manager. Invest in the stocks that garner the highest consensus as the preferred "best ideas" among the majority of the managers.

BUY	SELL
The stocks that are the “best ideas” for the greatest number of managers	When the stocks are no longer the “best ideas”

PARAMETER & VARIABLES

PARAMETER	VALUE
MARKETS TRADED	Equities
FINANCIAL INSTRUMENTS	Stocks
REGION	Global
PERIOD OF REBALANCING	Quarterly
NO. OF TRADED INSTRUMENTS	100
HOLDING PERIODS	Depends
LONG/SHORT	Long Only

DATA SOURCE

- SEC 13F Fillings

ALGORITHM

```
from AlgorithmImports import *
import numpy as np
from dateutil.relativedelta import relativedelta
#endregion

class AlphaCloningFollowing13FFillings(QCAlgorithm):

    def Initialize(self):
        self.SetStartDate(2011, 1, 1)
        self.SetCash(100000)

        self.weight = {}
        self.investors_preferences = {}

        self.symbol = self.AddEquity('SPY', Resolution.Daily).Symbol
        self.months_lag = 2 # Lag for getting investors preferences report

        csv_string_file = self.Download('data.quantpedia.com/backtesting_data/economic/investors_preferences.csv')
        lines = csv_string_file.split('\r\n')
        dates = []
        # Skip csv header in loop
        for line in lines[1:]:
            line_split = line.split(';')
            date = datetime.strptime(line_split[0], "%d.%m.%Y").date()
```

```
            self.investors_preferences[date] = {}

            for ticker in line_split[1:]:
                if ticker not in self.investors_preferences[date]:
                    self.investors_preferences[date][ticker] = 0

                self.investors_preferences[date][ticker] += 1

        self.month_counter = 0
        self.selection_flag = False
        self.UniverseSettings.Resolution = Resolution.Daily
        self.AddUniverse(self.CoarseSelectionFunction)
        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):
        if not self.selection_flag:
            return Universe.Unchanged

        selected_report = None
        min_date = self.Time.date() - relativedelta(months=self.months_lag+1)   # quarterly dat
a
        max_date = self.Time.date()

        for date in self.investors_preferences:
            # Get latest report
            if date >= min_date and date <= max_date:
                selected_report = self.investors_preferences[date]

        # Report might not be selected, because there are no data for that date
        if selected_report is None:
            return Universe.Unchanged

        # Select universe based on report
        selected = [x.Symbol for x in coarse if x.Symbol.Value in selected_report]

        # Calculate total preferences votes for selected report
        total_preferences_votes = sum([x[1] for x in selected_report.items()])

        # Calculate weight for each stock in selected universe
        for symbol in selected:
            # weight = total stock preferences votes / total investor votes in selected report
            self.weight[symbol] = selected_report[symbol.Value] / total_preferences_votes

        return selected

    def OnData(self, data):
        if not self.selection_flag:
            return
        self.selection_flag = False

        # Trade Execution
        stocks_invested = [x.Key for x in self.Portfolio if x.Value.Invested]
        for symbol in stocks_invested:
            if symbol not in self.weight:
                self.Liquidate(symbol)

        for symbol, w in self.weight.items():
            if symbol in data and data[symbol]:
```

```
                self.SetHoldings(symbol, w)

        self.weight.clear()

    def Selection(self):
        if self.Time.month % 3 == 2:
            self.selection_flag = True

# 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

PSR	8.114%	Sharpe Ratio	0.686
Total Trades	4799	Average Win	0.12%
Average Loss	-0.13%	Compounding Annual Return	11.682%
Drawdown	33.900%	Expectancy	0.417
Net Profit	282.387%	Loss Rate	27%
Win Rate	73%	Profit-Loss Ratio	0.93
Alpha	0.019	Beta	0.736
Annual Standard Deviation	0.129	Annual Variance	0.017
Information Ratio	-0.08	Tracking Error	0.081
Treynor Ratio	0.12	Total Fees	$294.71
Estimated Strategy Capacity	$9200000.00	Lowest Capacity Asset	WCBO R735QTJ8XC9X

Fig 2. Performance Metrics

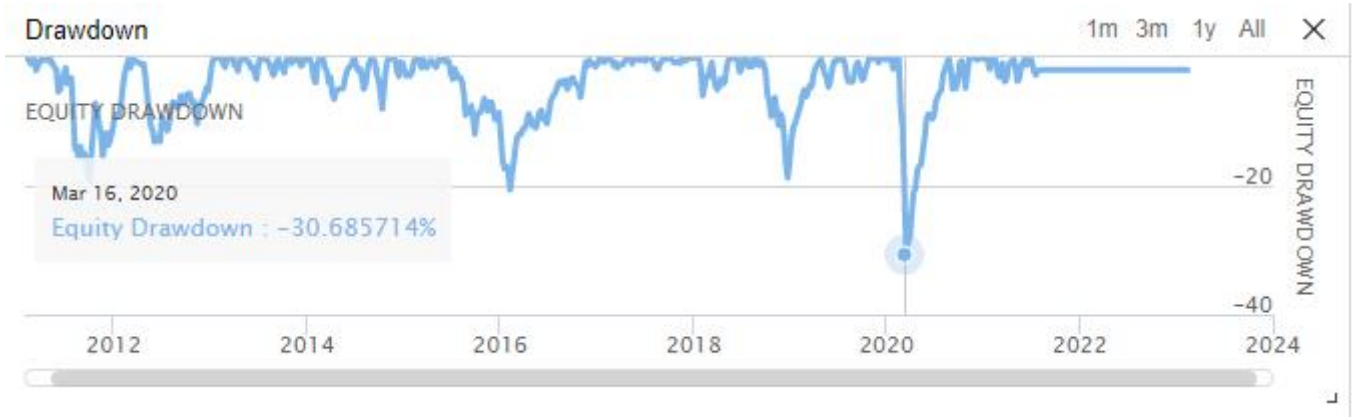

Fig 3. Drawdown

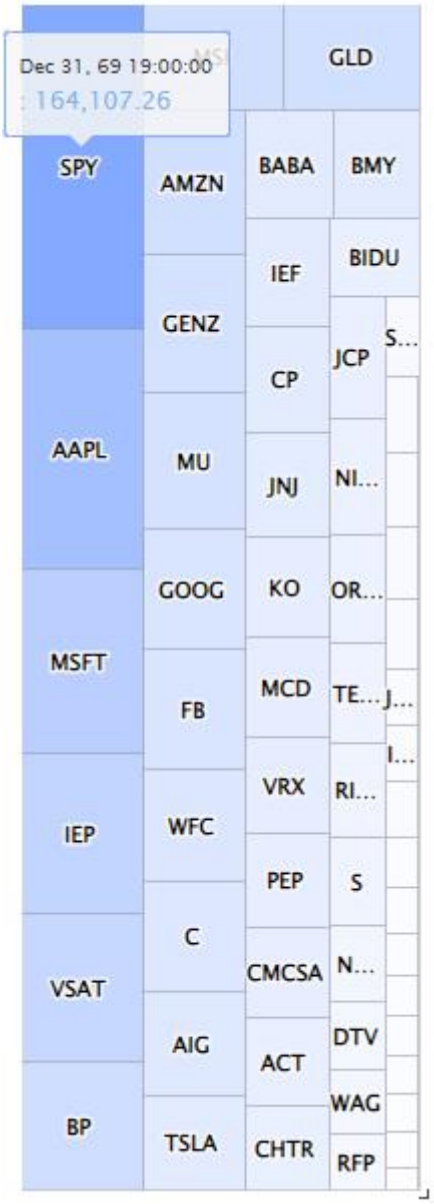

Fig 4. Assets Sales Volume