Not Over Thinking

Betting Against Beta Factor in International Equities

Algorithmic Trading Strategy with Full Code

Haixiang
2023.09 | Vol 36.
hxyan.2015@gmail.com | github.com/hxyan2020

STRATEGY & ECONOMIC RATIONALE

The investment universe consists of all country ETFs. The beta for each country is calculated with respect to the MSCI US Equity Index using a 1-year rolling window. ETFs are then ranked in ascending order based on their estimated beta. The ranked ETFs are assigned to one of two portfolios: low beta and high beta. Securities are weighted by the ranked betas, and the portfolios are rebalanced every calendar month.

Both portfolios are rescaled to have a beta of one at portfolio formation. The “Betting-Against-Beta” is the zero-cost zero-beta portfolio that is long on the low-beta portfolio and that shorts the high-beta portfolio. There are a lot of simple modifications (like going long on the bottom beta decile and short on the top beta decile), which could probably improve the strategy’s performance.

BUY	SELL
low-beta portfolio	high-beta portfolio

PARAMETER & VARIABLES

PARAMETER	VALUE
MARKETS TRADED	Equity
FINANCIAL INSTRUMENTS	ETFs, futures
REGION	Global
PERIOD OF REBALANCING	Monthly
NO. OF TRADED INSTRUMENTS	13
WEIGHTING	Equal weighting
LOOKBACK PERIODS	1 year
LONG/SHORT	Long and Short

ALGORITHM

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

class BettingAgainstBetaFactorinInternationalEquities(QCAlgorithm):

    def Initialize(self):
        self.SetStartDate(2002, 2, 1)
        self.SetCash(100000)

        self.countries = [
                        "EWA",  # iShares MSCI Australia Index ETF
                        "EWO",  # iShares MSCI Austria Investable Mkt Index ETF
                        "EWK",  # iShares MSCI Belgium Investable Market Index ETF
                        "EWZ",  # iShares MSCI Brazil Index ETF
                        "EWC",  # iShares MSCI Canada Index ETF
                        "FXI",  # iShares China Large-Cap ETF
                        "EWQ",  # iShares MSCI France Index ETF
                        "EWG",  # iShares MSCI Germany ETF
```

```
                    "EWH",  # iShares MSCI Hong Kong Index ETF
                    "EWI",  # iShares MSCI Italy Index ETF
                    "EWJ",  # iShares MSCI Japan Index ETF
                    "EWM",  # iShares MSCI Malaysia Index ETF
                    "EWW",  # iShares MSCI Mexico Inv. Mt. Idx
                    "EWN",  # iShares MSCI Netherlands Index ETF
                    "EWS",  # iShares MSCI Singapore Index ETF
                    "EZA",  # iShares MSCI South Africe Index ETF
                    "EWY",  # iShares MSCI South Korea ETF
                    "EWP",  # iShares MSCI Spain Index ETF
                    "EWD",  # iShares MSCI Sweden Index ETF
                    "EWL",  # iShares MSCI Switzerland Index ETF
                    "EWT",  # iShares MSCI Taiwan Index ETF
                    "THD",  # iShares MSCI Thailand Index ETF
                    "EWU",  # iShares MSCI United Kingdom Index ETF
                ]

        self.leverage_cap = 5

        # Daily price data.
        self.data = {}
        self.period = 12 * 21

        self.symbol = 'SPY'

        for symbol in self.countries + [self.symbol]:
            data = self.AddEquity(symbol, Resolution.Daily)
            data.SetFeeModel(CustomFeeModel())
            data.SetLeverage(15)

            self.data[symbol] = RollingWindow[float](self.period)

        self.recent_month = -1

    def OnData(self, data):
        for symbol in self.data:
            symbol_obj = self.Symbol(symbol)
            if symbol_obj in data.Keys:
                if data[symbol_obj]:
                    price = data[symbol_obj].Value
                    if price != 0:
                        self.data[symbol].Add(price)

        if self.recent_month == self.Time.month:
            return
        self.recent_month = self.Time.month

        beta = {}
        for symbol in self.countries:
            # Data is ready.
            if self.data[self.symbol].IsReady and self.data[symbol].IsReady and
self.symbol in data and symbol in data:
                market_closes = np.array([x for x in self.data[self.symbol]])
```

```
                asset_closes = np.array([x for x in self.data[symbol]])

                market_returns = (market_closes[1:] - market_closes[:-1]) /
market_closes[:-1]
                asset_returns = (asset_closes[1:] - asset_closes[:-1]) / asset_closes[:-1]

                cov = np.cov(asset_returns, market_returns)[0][1]
                market_variance = np.var(market_returns)
                beta[symbol] = cov / market_variance

        weight = {}
        if len(beta) != 0:
            # Beta diff calc.
            beta_median = np.median([x[1] for x in beta.items()])

            long_diff = [(x[0], abs(beta_median - x[1])) for x in beta.items() if x[1] <
beta_median]
            short_diff = [(x[0], abs(beta_median - x[1])) for x in beta.items() if x[1] >
beta_median]

            # Beta rescale.
            long_portfolio_beta = np.mean([beta[x[0]] for x in long_diff])
            long_leverage = 1 / long_portfolio_beta

            short_portfolio_beta = np.mean([beta[x[0]] for x in short_diff])
            short_leverage = 1 / short_portfolio_beta

            # Cap long and short leverage.
            long_leverage = min(self.leverage_cap, long_leverage)
            long_leverage = max(-self.leverage_cap, long_leverage)
            short_leverage = min(self.leverage_cap, short_leverage)
            short_leverage = max(-self.leverage_cap, short_leverage)

            # self.Log(f"long: {long_leverage}; short: {short_leverage}")

            total_long_diff = sum([x[1] for x in long_diff])
            total_short_diff = sum([x[1] for x in short_diff])

            # Beta diff weighting.
            weight = {}
            for symbol, diff in long_diff:
                weight[symbol] = (diff / total_long_diff) * long_leverage
            for symbol, diff in short_diff:
                weight[symbol] = - (diff / total_short_diff) * short_leverage

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

        for symbol, w in weight.items():
            self.SetHoldings(symbol, w)
```

```
# 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	0.000%	Sharpe Ratio	0.136
Total Trades	9990	Average Win	0.48%
Average Loss	-0.48%	Compounding Annual Return	1.190%
Drawdown	39.300%	Expectancy	0.018
Net Profit	27.077%	Loss Rate	49%
Win Rate	51%	Profit-Loss Ratio	0.99
Alpha	0.018	Beta	-0.074
Annual Standard Deviation	0.091	Annual Variance	0.008
Information Ratio	-0.313	Tracking Error	0.193
Treynor Ratio	-0.167	Total Fees	$6899.07
Estimated Strategy Capacity	$7000.00	Lowest Capacity Asset	EWK R735QTJ8XC9X

Fig 2. Performance Metrics

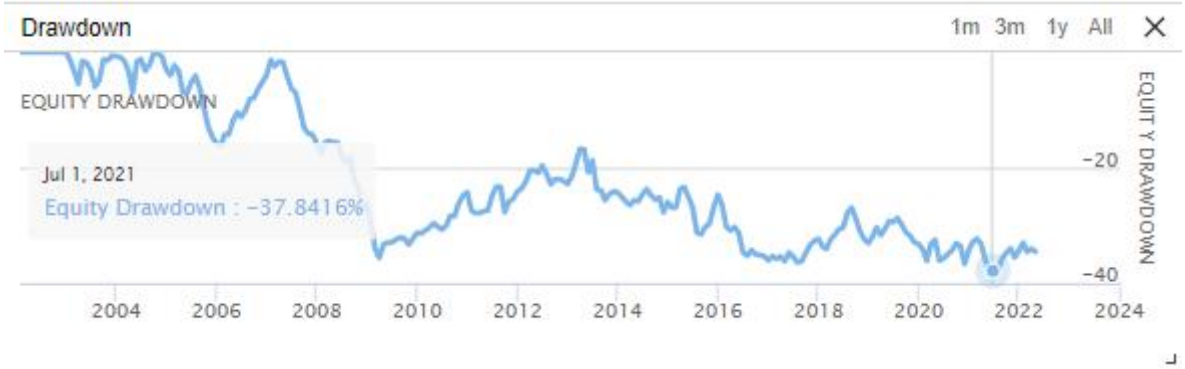

Fig 3. Drawdown

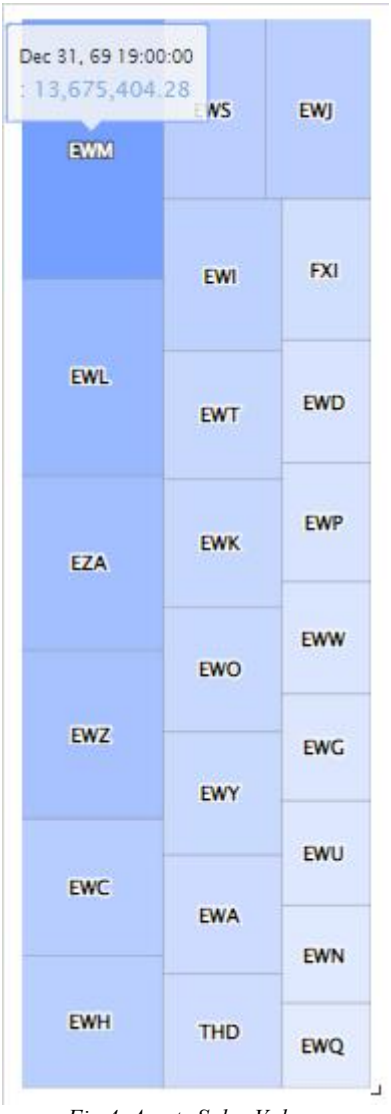

Fig 4. Assets Sales Volume