Not Over Thinking

ESG Level Factor Investing Strategy

Algorithmic Trading Strategy with Full Code

Haixiang

2023.12 | Vol 60.

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

STRATEGY & ECONOMIC RATIONALE

As we have previously mentioned, the choice of the database of ESG scores can alter results. This paper uses for the assessments of environment, social, and governance performance of single firms database provided by Asset4. Scores are updated every year, therefore to obtain monthly ESG data, the scores remain unchanged until the next assessment.

The investment universe consists of stocks of the North America region (Canada and the United States) that have ESG scores available. Stocks with a price of less than one USD are excluded. Paper examines the returns as abnormal returns according to the methodology of Daniel et al. (1997). Such methodology controls for risk factors such as size, book-to-market ratio, and momentum. The idea is to match a stock along with the mentioned factors to a benchmark portfolio that contains stocks with similar characteristics. Therefore, for the North America region, we have 4×4 benchmark portfolios. The abnormal return is calculated as the return of stock minus the return of stock´s matching benchmark portfolio return (equation 1, page 13).

Finally, each month stocks are ranked according to their E, S and G scores. Long top 20% stocks of each score and short the bottom 20% stocks of each score. Therefore, we have one complex strategy that consists of three individual strategies (for representative purposes; the paper examines each strategy individually). The strategy is equally-weighted: both stocks in the quintiles and individual strategies. The strategy is rebalanced yearly.

BUY	SELL
(see above)	(see above)

PARAMETER & VARIABLES

PARAMETER	VALUE
MARKETS TRADED	Equity
FINANCIAL INSTRUMENTS	Stocks
REGION	United States
PERIOD OF REBALANCING	Yearly
NO. OF TRADED INSTRUMENTS	1000
WEIGHTING	Equal weighting
LOOKBACK PERIODS	N/A
LONG/SHORT	Long only

ALGORITHM

```python
from AlgorithmImports import *
from numpy import floor
#endregion

class ESGFactorInvestingStrategy(QCAlgorithm):

    def Initialize(self):
        self.SetStartDate(2009, 6, 1)
        self.SetCash(100000)

        # Decile weighting.
        # True - Value weighted
        # False - Equally weighted
        self.value_weighting = True
        self.symbol = self.AddEquity('SPY', Resolution.Daily).Symbol
        self.esg_data = self.AddData(ESGData, 'ESG', Resolution.Daily)

        # All tickers from ESG database.
        self.tickers = []

        self.ticker_deciles = {}

        self.holding_period = 12
        self.managed_queue = []

        self.latest_price = {}

        self.selection_flag = False
        self.UniverseSettings.Resolution = Resolution.Daily
        self.AddUniverse(self.CoarseSelectionFunction, self.FineSelectionFunction)

    def OnSecuritiesChanged(self, changes):
        for security in changes.AddedSecurities:
            security.SetFeeModel(CustomFeeModel())
            security.SetLeverage(10)

    def CoarseSelectionFunction(self, coarse):
        if not self.selection_flag:
            return Universe.Unchanged

        self.latest_price.clear()

        selected = [x for x in coarse if (x.Symbol.Value).lower() in self.tickers]

        for stock in selected:
            symbol = stock.Symbol
            self.latest_price[symbol] = stock.AdjustedPrice

        return [x.Symbol for x in selected]

    def FineSelectionFunction(self, fine):
        fine = [x for x in fine if x.MarketCap != 0]

        # Store symbol/market cap pair.
        long = [x for x in fine if (x.Symbol.Value in self.ticker_deciles) and
                                   \
                                    (self.ticker_deciles[x.Symbol.Value] is not None) and
                                           \
                                    (self.ticker_deciles[x.Symbol.Value] >= 0.8)]

        short = [x for x in fine if (x.Symbol.Value in self.ticker_deciles) and
                                   \
                                    (self.ticker_deciles[x.Symbol.Value] is not None) and
                                           \
                                    (self.ticker_deciles[x.Symbol.Value] <= 0.2)]

        long_symbol_q = []
        short_symbol_q = []

        # ew
        if not self.value_weighting:
            if len(long) != 0:
                long_w = self.Portfolio.TotalPortfolioValue / self.holding_period / len(long)
                long_symbol_q = [(x.Symbol, floor(long_w / self.latest_price[x.Symbol])) for x
    in long]

            if len(short) != 0:
                short_w = self.Portfolio.TotalPortfolioValue / self.holding_period / len(short)
                short_symbol_q = [(x.Symbol, -floor(short_w / self.latest_price[x.Symbol])) for
 x in short]
        # vw
        else:
            if len(long) != 0:
                total_market_cap_long = sum([x.MarketCap for x in long])
                long_w = self.Portfolio.TotalPortfolioValue / self.holding_period
                long_symbol_q = [(x.Symbol, floor((long_w * (x.MarketCap / total_market_cap_lon
g))) / self.latest_price[x.Symbol]) for x in long]

            short_symbol_q = []
            if len(short) != 0:
                total_market_cap_short = sum([x.MarketCap for x in short])
                short_w = self.Portfolio.TotalPortfolioValue / self.holding_period
                short_symbol_q = [(x.Symbol, -floor((short_w * (x.MarketCap / total_market_cap_
short))) / self.latest_price[x.Symbol]) for x in short]

        self.managed_queue.append(RebalanceQueueItem(long_symbol_q + short_symbol_q))
        self.ticker_deciles.clear()

        return [x.Symbol for x in long + short]

    def OnData(self, data):
        new_data_arrived = False

        if 'ESG' in data and data['ESG']:
            # Store universe tickers.
            if len(self.tickers) == 0:
                # TODO '_typename' in storage dictionary?
                self.tickers = [x.Key for x in self.esg_data.GetLastData().GetStorageDictionary
()][:-1]

            # Store history for every ticker.
            for ticker in self.tickers:
                ticker_u = ticker.upper()
                if ticker_u not in self.ticker_deciles:
                    self.ticker_deciles[ticker_u] = None

                decile = self.esg_data.GetLastData()[ticker]
                self.ticker_deciles[ticker_u] = decile

                # trigger selection after new esg data arrived.
                if not self.selection_flag:
                    new_data_arrived = True

        if new_data_arrived:
            self.selection_flag = True
            return

        if not self.selection_flag:
            return
        self.selection_flag = False

        # Trade execution
        remove_item = None

        # Rebalance portfolio
        for item in self.managed_queue:
            if item.holding_period == self.holding_period:
                for symbol, quantity in item.symbol_q:
                    self.MarketOrder(symbol, -quantity)

                remove_item = item
            elif item.holding_period == 0:
                open_symbol_q = []

                for symbol, quantity in item.symbol_q:
                    if symbol in data and data[symbol]:
                        if quantity >= 1:
                            self.MarketOrder(symbol, quantity)
                            open_symbol_q.append((symbol, quantity))

                # Only opened orders will be closed
                item.symbol_q = open_symbol_q

            item.holding_period += 1

        if remove_item:
            self.managed_queue.remove(remove_item)

class RebalanceQueueItem():
    def __init__(self, symbol_q):
        # symbol/quantity collections
        self.symbol_q = symbol_q
        self.holding_period = 0

# ESG data.
class ESGData(PythonData):
    def __init__(self):
        self.tickers = []

    def GetSource(self, config, date, isLiveMode):
        return SubscriptionDataSource("data.quantpedia.com/backtesting_data/economic/esg_decile
s_data.csv", SubscriptionTransportMedium.RemoteFile, FileFormat.Csv)

    def Reader(self, config, line, date, isLiveMode):
        data = ESGData()
        data.Symbol = config.Symbol

        if not line[0].isdigit():
            self.tickers = [x for x in line.split(';')][1:]
            return None

        split = line.split(';')

        data.Time = datetime.strptime(split[0], "%Y-%m-%d") + timedelta(days=1)

        index = 1
        for ticker in self.tickers:
            data[ticker] = float(split[index])
            index += 1

        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

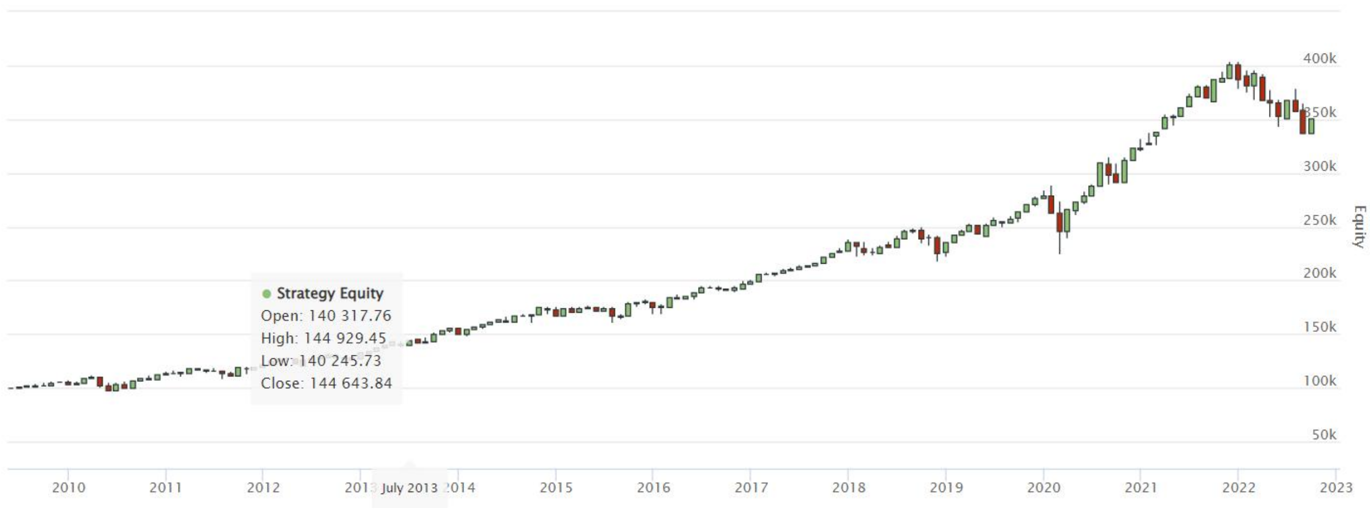
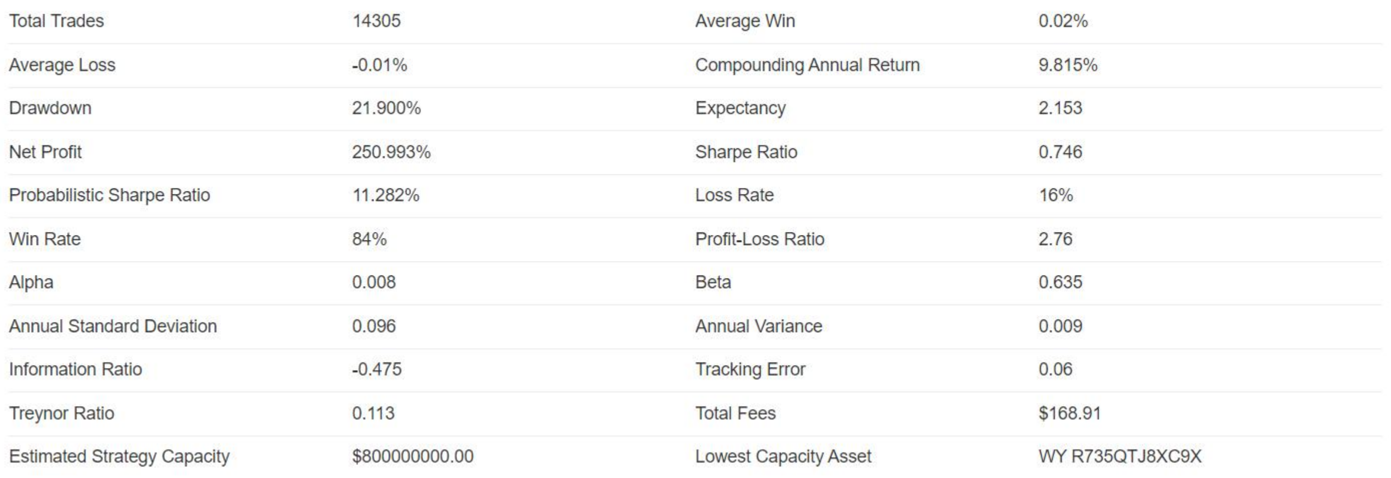


Fig 1. Overall Performance



Total Trades	14305	Average Win	0.02%
Average Loss	-0.01%	Compounding Annual Return	9.815%
Drawdown	21.900%	Expectancy	2.153
Net Profit	250.993%	Sharpe Ratio	0.746
Probabilistic Sharpe Ratio	11.282%	Loss Rate	16%
Win Rate	84%	Profit-Loss Ratio	2.76
Alpha	0.008	Beta	0.635
Annual Standard Deviation	0.096	Annual Variance	0.009
Information Ratio	-0.475	Tracking Error	0.06
Treynor Ratio	0.113	Total Fees	$168.91
Estimated Strategy Capacity	$800000000.00	Lowest Capacity Asset	WY R735QTJ8XC9X

Fig 2. Performance Metrics