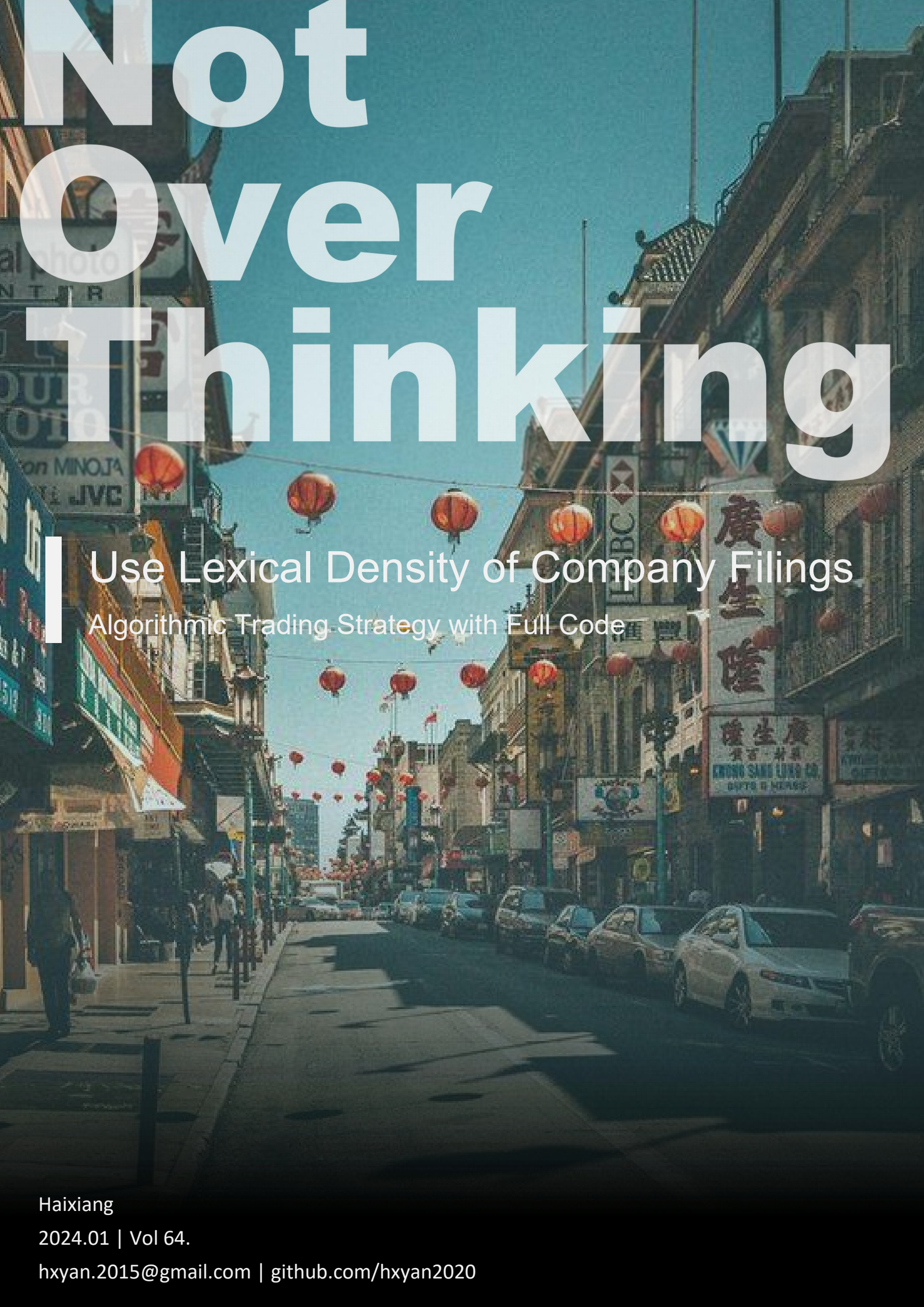
Not Over Thinking



Use Lexical Density of Company Filings

Algorithmic Trading Strategy with Full Code

Haixiang

2024.01 | Vol 64.

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

STRATEGY & ECONOMIC RATIONALE

The investment universe consists of top 500 US stocks by dollar volume. The stocks are sorted b ased on their lexical density and specific density score from the BLMCF dataset. Lexical densit y measures the structure and complexity of human communication in a text. A high lexical densit y indicates a large amount of information-carrying words. Specific density measures how dense t he report's language is from a financial point of view. In other words, how many finance- relat ed words are used in the text. The investor goes long the top decile and short the bottom decil e. Additionally, the portfolio is rebalanced on a monthly basis.

BUY	SELL
goes long the top decile	short the bottom decile

PARAMETER & VARIABLES

PARAMETER	VALUE
MARKETS TRADED	Equity
FINANCIAL INSTRUMENTS	Stocks
REGION	United States
PERIOD OF REBALANCING	Monthly
NO. OF TRADED INSTRUMENTS	500
WEIGHTING	Equal weighting
LOOKBACK PERIODS	N/A
LONG/SHORT	Long & Short

ALGORITHM

```
from AlgorithmImports import *from QuantConnect.DataSource import *import numpy as npfrom enum impor
t Enum#endregion
class HowtoUseLexicalDensityofCompanyFilings(QCAlgorithm):

    def Initialize(self):
        self.SetStartDate(2010, 1, 1)
        self.init_cash = 100000
        self.SetCash(self.init_cash)

        self.market = self.AddEquity('SPY', Resolution.Daily).Symbol
        self.mkt = [] # benchmark chart data

        # metric dictionary with signal optimism flag
        # metric_dictionary:dict[int, (str, bool)] = {
        #     # 1 : ('SentenceCount', True),
        #     # 2 : ('MeanSentenceLength', True),
        #     # 3 : ('Sentiment', True),
        #     # 4 : ('Uncertainty', False),
        #     # 5 : ('Litigious', False),
        #     # 6 : ('Constraining', False),
        #     # 7 : ('Interesting', True),
        #     # 8 : ('Readability', True),
        #     9 : ('LexicalRichness', True),
        #     10 : ('LexicalDensity', True),
        #     11 : ('SpecificDensity', True),
        #     12 : ('SPY', True),
        # }

        self.metric_values = [
```

```
            #'LexicalRichness',  #9
            'LexicalDensity',   #10
            'SpecificDensity'   #11
            ]

        # opt parameters
        # self.metric_property:tuple = metric_dictionary[int(self.GetParameter("metric"))]
        # self.metric_property:tuple = metric_dictionary[11]

        # self.portfolio_size_property:int = int(self.GetParameter("portfolio_size"))
        self.portfolio_size_property:int = 10

        # self.universe_size_property:int = int(self.GetParameter("universe_size"))
        self.universe_size_property:int = 500

        # self.long = []
        # self.short = []
        self.traded_quantity = {}

        self.metric = {}
        self.metric_symbols = {}
        self.price = {}

        self.recent_universe = []

        self.coarse_count = self.universe_size_property

        self.selection_flag = False
        self.rebalance_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)
        # self.Schedule.On(self.DateRules.EveryDay(self.market), self.TimeRules.AfterMarketOpen(self.
market), self.PrintBenchmark)

    def PrintBenchmark(self):
        mkt_price = self.History(self.market, 2, Resolution.Daily)['close'].unstack(level=0).iloc[-1]
        self.mkt.append(mkt_price)
        mkt_perf = self.init_cash * self.mkt[-1] / self.mkt[0]
        self.Plot('Strategy Equity', self.market, mkt_perf)

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

        # remove recently stored metric value
        for security in changes.RemovedSecurities:
            symbol = security.Symbol
            if symbol in self.metric:
                del self.metric[symbol]

    def CoarseSelectionFunction(self, coarse):
        # return old universe if selection is not needed
        if self.rebalance_flag and not self.selection_flag:
            for stock in coarse:
                symbol = stock.Symbol
                if symbol in self.recent_universe:
                    self.price[symbol] = stock.AdjustedPrice

            return self.recent_universe

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

        if self.universe_size_property == 500 or self.universe_size_property == 1000:
```

```
            # select top n stocks by dollar volume
            selected = [x for x in sorted([x for x in coarse if x.HasFundamentalData],
                    key = lambda x: x.DollarVolume, reverse = True)[:self.coarse_count]]
        elif self.universe_size_property == 3000:
            selected = [x for x in coarse if x.HasFundamentalData]

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

            if symbol in self.metric:
                continue

            # create RollingWindow for specific stock symbol
            # self.metric[symbol] = RollingWindow[float](self.period)
            self.metric[symbol] = None

            # subscribe to Brain Language Metrics data
            dataset_symbol = self.AddData(BrainCompanyFilingLanguageMetrics10K , symbol).Symbol

            # warmup Brain Language Metrics data
            history = self.History(dataset_symbol, 3*30, Resolution.Daily)
            # self.Debug(f"We got {len(history)} items from our history request for {dataset_symbol}
")

            if not history.empty:
                metrics = []
                for metric_value in self.metric_values:
                    m = getattr(history['reportsentiment'].iloc[-1], metric_value)
                    metrics.append(m)

                # sent = history['reportsentiment'].iloc[-1].Sentiment
                self.metric[symbol] = (history.iloc[-1].reportdate, metrics[0], metrics[1])#, metric
s[2])

            # store metric symbol under stock symbol
            self.metric_symbols[symbol] = dataset_symbol

        # return stock, which have short interest data ready
        return [x.Symbol for x in selected if x.Symbol in self.metric and x.Symbol in self.price]

    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 self.universe_size_property == 3000:
            fine = sorted(fine, key = lambda x:x.MarketCap, reverse=True)[:self.coarse_count]

        self.recent_universe = [x.Symbol for x in fine]

        metric_cnt = len(self.metric_values)
        for ms_i in range(metric_cnt):
            metric = { stock.Symbol : self.metric[stock.Symbol][ms_i+1] for stock in fine   \
                    if stock.Symbol in self.metric and  \
                    self.metric[stock.Symbol] is not None and   \
                    self.metric[stock.Symbol][ms_i+1] is not None and \
                    (self.Time - self.metric[stock.Symbol][0]).days <= 30
            }

            if len(metric) < self.portfolio_size_property:
                continue

            # sorting by metric
            sorted_by_metric = sorted(metric.items(), key = lambda x: x[1], reverse=True)
            percentile = int(len(sorted_by_metric) / self.portfolio_size_property)
```

```
            long = [x[0] for x in sorted_by_metric[:percentile]]
            short = [x[0] for x in sorted_by_metric[-percentile:]]

            # calculate quantity for every stock in every portfolio
            long_cnt = len(long)
            short_cnt = len(short)
            for symbol in long:
                q = int(((self.Portfolio.TotalPortfolioValue / metric_cnt) / long_cnt) / self.price
[symbol])
                if symbol not in self.traded_quantity:
                    self.traded_quantity[symbol] = 0
                self.traded_quantity[symbol] += q
            for symbol in short:
                q = -int(((self.Portfolio.TotalPortfolioValue / metric_cnt) / short_cnt) / self.pric
e[symbol])
                if symbol not in self.traded_quantity:
                    self.traded_quantity[symbol] = 0
                self.traded_quantity[symbol] += q

            # self.short = []
            # self.long = []

        return list(self.traded_quantity.keys())

    def OnData(self, data):
        # update metric value for each stock
        for stock_symbol, metric_symbol in self.metric_symbols.items():
            # check if there are data for subscribed metric_symbol
            if metric_symbol in data and data[metric_symbol]:
                metrics = []
                for metric_value in self.metric_values:
                    m = getattr(data[metric_symbol].ReportSentiment, metric_value)
                    metrics.append(m)
                # sent = data[metric_symbol].ReportSentiment.Sentiment

                # update metric value for specific stock
                self.metric[stock_symbol] = (self.Time, metrics[0], metrics[1])#, metrics[2])

        # monthly rebalance
        if not self.rebalance_flag:
            return
        self.rebalance_flag = False

        if self.universe_size_property == 3000:
            if self.Time.year in [2014, 2016] and self.Time.month == 6:
                self.Liquidate()
                return

        self.Liquidate()

        for symbol, q in self.traded_quantity.items():
            if q != 0:
                if symbol in data and data[symbol]:
                    self.MarketOrder(symbol, q)

        # long_c = len(self.long)
        # short_c = len(self.short)
        # for symbol in self.long:
        #     self.SetHoldings(symbol, 1/long_c)
        # for symbol in self.short:
        #     self.SetHoldings(symbol, -1/short_c)

        # self.weight.clear()
        # self.long.clear()
        # self.short.clear()
        self.traded_quantity.clear()

    def Selection(self):
```

```
        # if metric is market, hold SPY only without rebalance and selection
        # if self.metric_property[0] == self.market.Value:
        #     if not self.Portfolio[self.market].Invested:
        #         self.SetHoldings(self.market, 1)
        # else:
        # new universe selection every three months

        if self.Time.month % 3 == 0:
            self.selection_flag = True

        # rebalance once a month
        self.rebalance_flag = True
# Custom fee modelclass CustomFeeModel(FeeModel):
    def GetOrderFee(self, parameters):
        fee = parameters.Security.Price * parameters.Order.AbsoluteQuantity * 0.00005
        return OrderFee(CashAmount(fee, "USD"))
```

BACKTESTING PERFORMANCE

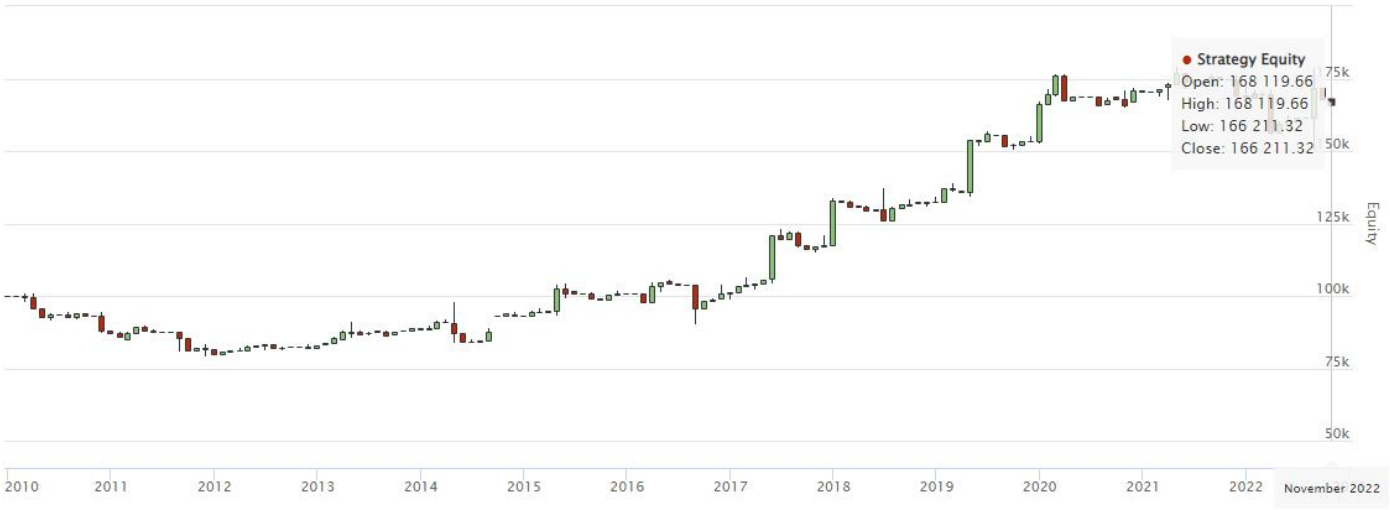


Fig 1. Overall Performance

Total Trades	21725	Average Win	0.12%
Average Loss	-0.12%	Compounding Annual Return	3.948%
Drawdown	21.800%	Expectancy	0.047
Net Profit	64.481%	Sharpe Ratio	0.357
Probabilistic Sharpe Ratio	0.258%	Loss Rate	48%
Win Rate	52%	Profit-Loss Ratio	1.02
Alpha	0.034	Beta	-0.037
Annual Standard Deviation	0.087	Annual Variance	0.008
Information Ratio	-0.357	Tracking Error	0.174
Treynor Ratio	-0.833	Total Fees	$2189.89
Estimated Strategy Capacity	$61000000.00	Lowest Capacity Asset	AMBA VANAJ80VPN51

Fig 2. Performance Metrics