Not Over Thinking

Dispersion Trading

Algorithmic Trading Strategy with Full Code

Haixiang

2023.12 | Vol 53.

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

STRATEGY & ECONOMIC RATIONALE

The investment universe consists of stocks from the S&P 100 index. Trading vehicles are options on stocks from this index and also options on the index itself. The investor uses analyst forecasts of earnings per share from the Institutional Brokers Estimate System (I/B/E/S) database and computes for each firm the mean absolute difference scaled by an indicator of earnings uncertainty. Each month, investor sorts stocks into quintiles based on the size of belief disagreement. He buys puts of stocks with the highest belief disagreement and sells the index puts with Black-Scholes deltas ranging from -0.8 to -0.2.

BUY	SELL
puts of stocks with the highest belief disagreement	sells the index puts with Black-Scholes deltas ranging from -0.8 to -0.2

PARAMETER & VARIABLES

PARAMETER	VALUE
MARKETS TRADED	Equity
FINANCIAL INSTRUMENTS	Options
REGION	United States
PERIOD OF REBALANCING	Monthly
NO. OF TRADED INSTRUMENTS	21
WEIGHTING	
LOOKBACK PERIODS	
LONG/SHORT	

ALGORITHM

```
class DispersionTrading(QCAlgorithm):

    def Initialize(self):
        self.SetStartDate(2010, 1, 1)
        self.SetCash(1000000)

        self.min_expiry = 20
        self.max_expiry = 60

        self.index_symbol = self.AddIndex('SPX').Symbol
        self.percentage_traded = 1.0

        self.spx_contract = None
        self.selected_symbols = []
        self.subscribed_contracts = {}

        self.coarse_count = 100
        self.UniverseSettings.Resolution = Resolution.Minute
        self.AddUniverse(self.CoarseSelectionFunction)
        self.SetSecurityInitializer(lambda x:
x.SetDataNormalizationMode(DataNormalizationMode.Raw))
```

```
        self.UniverseSettings.DataNormalizationMode = DataNormalizationMode.Raw

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

    def CoarseSelectionFunction(self, coarse):
        # rebalance on SPX contract expiration (should be on monthly basis)
        if len(self.selected_symbols) != 0:
            return Universe.Unchanged

        # select top n stocks by dollar volume
        selected = sorted([x for x in coarse if x.HasFundamentalData and x.Market == 'usa'
and x.Price > 5],
                key=lambda x: x.DollarVolume, reverse=True)[:self.coarse_count]

        self.selected_symbols = [x.Symbol for x in selected]

        return self.selected_symbols

    def OnData(self, data):
        # liquidate portfolio, when SPX contract is about to expire in 2 days
        if self.index_symbol in self.subscribed_contracts and
self.subscribed_contracts[self.index_symbol].ID.Date.date() - timedelta(2) <=
self.Time.date():
            self.subscribed_contracts.clear()   # perform new subscribtion
            self.selected_symbols.clear()       # perform new selection
            self.Liquidate()

        if len(self.subscribed_contracts) == 0:
            if self.Portfolio.Invested:
                self.Liquidate()

            # NOTE order is important, index should come first
            for symbol in [self.index_symbol] + self.selected_symbols:
                # subscribe to contract
                contracts = self.OptionChainProvider.GetOptionContractList(symbol,
self.Time)
                # get current price for stock
                underlying_price = self.Securities[symbol].Price

                # get strikes from stock contracts
                strikes = [i.ID.StrikePrice for i in contracts]

                # check if there is at least one strike
                if len(strikes) <= 0:
                    continue

                # at the money
                atm_strike = min(strikes, key=lambda x: abs(x-underlying_price))

                # filtred contracts based on option rights and strikes
```

```
                atm_puts = [i for i in contracts if i.ID.OptionRight == OptionRight.Put
and
                                                    i.ID.StrikePrice == atm_strike and
                                                    self.min_expiry <= (i.ID.Date -
self.Time).days <= self.max_expiry]

                # index contract is found
                if symbol == self.index_symbol and len(atm_puts) == 0:
                    # cancel whole selection since index contract was not found
                    return

                # make sure there are enough contracts
                if len(atm_puts) > 0:
                    # sort by expiry
                    atm_put = sorted(atm_puts, key = lambda item: item.ID.Date,
reverse=True)[0]

                    # add contract
                    option = self.AddOptionContract(atm_put, Resolution.Minute)
                    option.PriceModel = OptionPriceModels.CrankNicolsonFD()
                    option.SetDataNormalizationMode(DataNormalizationMode.Raw)

                    # store subscribed atm put contract
                    self.subscribed_contracts[symbol] = atm_put

        # perform trade, when spx and stocks contracts are selected
        if not self.Portfolio.Invested and len(self.subscribed_contracts) != 0 and
self.index_symbol in self.subscribed_contracts:
            index_option_contract = self.subscribed_contracts[self.index_symbol]
            # make sure subscribed SPX contract has data
            if self.Securities.ContainsKey(index_option_contract):
                if self.Securities[index_option_contract].Price != 0 and
self.Securities[index_option_contract].IsTradable:
                    # sell SPX ATM put contract
                    self.Securities[index_option_contract].MarginModel =
BuyingPowerModel(2)
                    price = self.Securities[self.index_symbol].Price
                    if price != 0:
                        q = floor((self.Portfolio.TotalPortfolioValue *
self.percentage_traded) / (price*100))
                        self.Sell(index_option_contract, q)

                    # buy stock's ATM put contracts
                    long_count = len(self.subscribed_contracts) - 1     # minus index
symbol
                    for stock_symbol, stock_option_contract in
self.subscribed_contracts.items():
                        if stock_symbol == self.index_symbol:
                            continue

                        if self.Securities[stock_option_contract].Price != 0 and
self.Securities[stock_option_contract].IsTradable:
                            # buy contract
```

```
                        self.Securities[stock_option_contract].MarginModel =
BuyingPowerModel(2)
                        if self.Securities.ContainsKey(stock_option_contract):
                            price = self.Securities[stock_symbol].Price
                            if price != 0:
                                q = floor(((self.Portfolio.TotalPortfolioValue /
long_count) * self.percentage_traded) / (price*100))
                                self.Buy(stock_option_contract, q)

# 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	0.414%	Sharpe Ratio	0.393
Total Trades	323	Average Win	1.55%
Average Loss	-0.94%	Compounding Annual Return	4.565%
Drawdown	31.600%	Expectancy	0.404
Net Profit	80.917%	Loss Rate	47%
Win Rate	53%	Profit-Loss Ratio	1.66
Alpha	-0.012	Beta	0.491
Annual Standard Deviation	0.09	Annual Variance	0.008
Information Ratio	-0.651	Tracking Error	0.092
Treynor Ratio	0.072	Total Fees	$2.43
Estimated Strategy Capacity	$810000000.00	Lowest Capacity Asset	SPX 325YVH019A35A|SPX 31
Portfolio Turnover	0.07%		

Fig 2. Performance Metrics

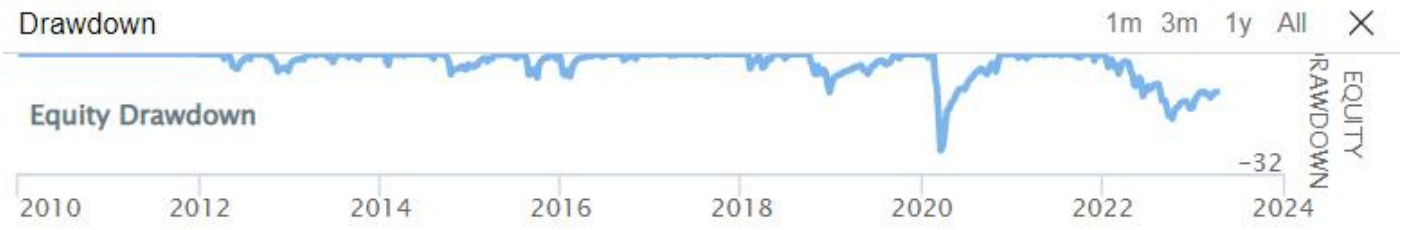

Fig 3. Drawdown

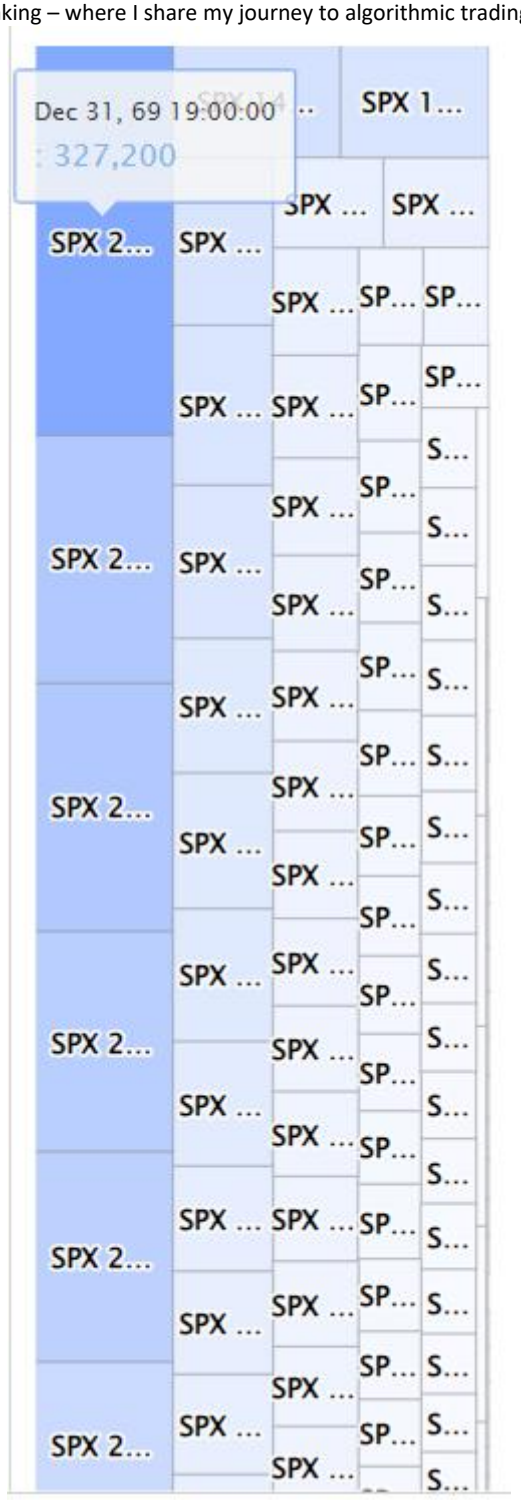

Fig 4. Assets Sales Volume