the Readme of MFE Trader——a CTA Backtesting Framework Based on Python

Content

1. Brief Introduction of MFE Trader

The mfe trader is an open source CTA backtesting framework based on python 3.7+, in which the investor can load their own data, and write their own strategies and backtest them easily.

The mfe trader partly uses the order crossing framework of vnpy. Therefore, the visualization of the backtesting of mfe trader is similar to that of vnpy. In addition, the mfe trader uses Worldquant's experience to compute factors just using a string, thereby greatly simplifying the computation.

2. the Structure of this Framework

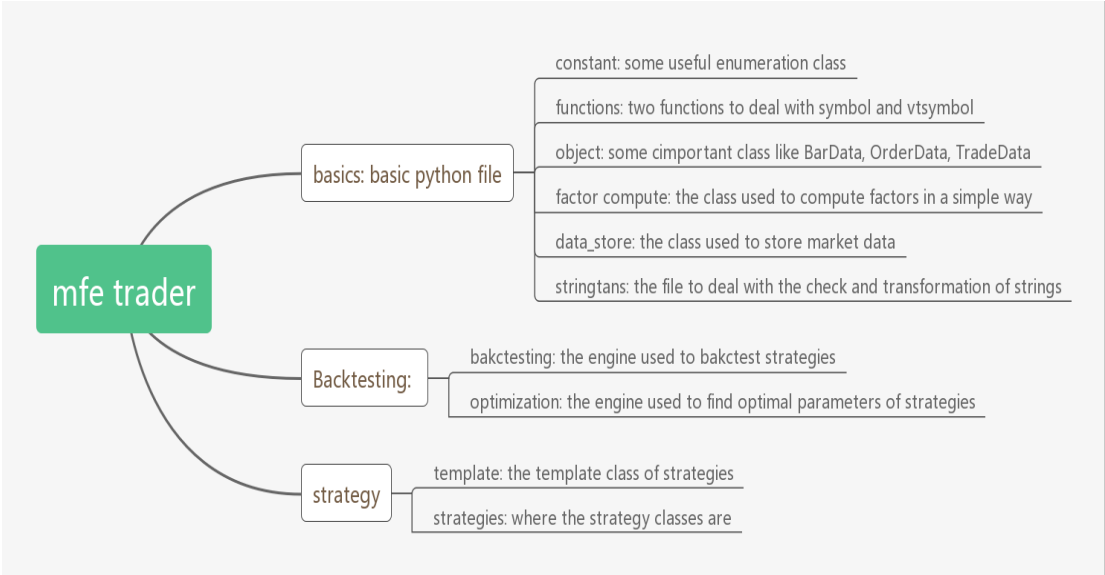


Figure 1： The structure of this framework

3. How to Bakctest a CTA Strategy Using MFE Trader

3.1 Initialize a BacktestingEngine

```
1. from mfe_trader.backtesting.Backtesting import BacktestingEngine
2. engine = BacktestingEngine()
```

We need to import BacktestingEngine and initialize it.

3.2 Add data to the BacktestingEngine

```
from mfe_trader.basics.constant import Interval
setting1 = {"vtsymbol":"HC888.SHFE",
        "interval": Interval.Day_1,
        "path": r"hc888_1d.csv",
        "datetime_row": "datetime"}
engine.add_data(setting1)
```

First, we need import Interval, which is an enumeration class used to show the frequency of bar data. Then we need to use a dict called setting to show the information of the data we want to add. Finally, we use the method engine.add_data to import data we use.

As for the structure of the data, it must be a csv file storing the bar data of a stock, a futures or even a digital currency. The following is a typical example:

Table 1: A example of the data structure

datetime	open	high	low	close	volume	opeinterest
2014/3/24 0:00	3314	3330	3266	3330	86810	47616
2014/3/25 0:00	3326	3400	3320	3378	119854	61256
2014/3/26 0:00	3370	3382	3354	3376	37876	60560
2014/3/27 0:00	3368	3386	3350	3362	40230	61430
2014/3/28 0:00	3376	3386	3360	3382	34290	63080
2014/3/31 0:00	3396	3422	3382	3410	60970	65640
2014/4/1 0:00	3412	3418	3396	3414	17896	64136
2014/4/2 0:00	3410	3426	3388	3390	28570	68700
2014/4/3 0:00	3384	3414	3382	3410	20706	68410
2014/4/4 0:00	3412	3440	3402	3440	21360	70094
2014/4/8 0:00	3450	3488	3442	3480	40286	80872
2014/4/9 0:00	3488	3488	3444	3452	32902	81834
2014/4/10 0:00	3450	3470	3442	3446	19630	81232
2014/4/11 0:00	3430	3452	3430	3434	13100	76840
2014/4/14 0:00	3432	3464	3432	3438	12642	77020
2014/4/15 0:00	3442	3448	3418	3448	20724	75524

Notice: all the name of columns must be the same as what the picture shows. And the openinterest must be written as “opeinterest”

Table 2: The content of setting1

key	**value**
vtsymbol	Asset.Exchange
interval	The frequency of data
path	The path and the file name of data
Datetime_row	The name of the row of datetime

3.3 Set the backtesting parameters

```
setting2 = {"start":datetime(2011,5,1,0,0),
            "end":datetime(2019,11,5,0,0),
            "captial":10000000,
            "rate":0.000185,
            "slippage":0}
engine.set_parameters(setting2)
```

Table 3： The content of setting2

key	**value**
start	The start date of backtesting
end	The end date of backtesting
capital	Investor's capital
rate	The rate of transaction cost
slippage	The amount of slippage

3.4 Run backtesting and show the result

```
engine.run_backtest()
engine.calculate_result()
result = engine.calculate_statistics()
engine.show_chart()
```

4. How to write a strategy class

The strategy class is the class where we can write the logic of our trading strategy and all the python file where the strategy class should be in the folder mfe_trader\strategy\strategies. The following introduces different parts of the strategy class based on a typical example A_Strategy

```
from ..template import Template

class A_Strategy(Template):

    factor_paramList = ["fastLength","slowLength"]
    general_paramList = ["volume"]

    def __init__(self, backtesting_engine,optimization = None):
        super(A_Strategy, self).__init__(backtesting_engine, optimization)
        self.volume = 1

    def OnInit(self):
        self.subscribe_factor("fastMA", "tsmean(close,12)")
```

```
        self.subscribe_factor("slowMA", "tsmean(close,14)")
        self.set_data_size(3)

    def OnBar(self, bar):
        if self.bar_count>=5:
            if self.array.fastMA.iloc[-1]>self.array.slowMA.iloc[-1] and self.pos == 0:
                self.buy(99999,10000)
            elif self.pos<0 and self.array.close.iloc[-1]>self.last_entry_price*(1+0.01):
                self.buy_to_cover(99999,-self.pos)
            elif self.array.fastMA.iloc[-1]<self.array.slowMA.iloc[-1] and self.pos >0:
                self.sell(1,self.pos)
            elif self.array.fastMA.iloc[-1]>self.array.slowMA.iloc[-1] and self.pos<0:
                self.buy_to_cover(99999,-self.pos)
            elif self.array.fastMA.iloc[-1]<self.array.slowMA.iloc[-1] and self.pos == 0:
                self.sell_short(1,10000)

        self.strategy_output(self.pos)

    def OnOrder(self, order):
        pass
        #self.strategy_output(f"下单价格为:{round(order.price,3)},下单量为:{order.volume},状态为:{order.status.value}")

    def OnTrade(self, trade):
        self.last_entry_price = trade.price
```

4.1 class attribution below the name of the strategy class

```
factor_paramList = ["fastLength","slowLength"]
general_paramList = ["volume"]
```

These two paramLists are used when we want to find the optimal parameters of our strategy. The elements of factor_paramList are in parameters in the factors denoted as string, while the elements of general_paramList are those acting as an attribution of the strategy class.

4.2 OnInit

This method is used to init some parameters and subscribe some factors denoted as strings and is called when the strategy is added to a BacktestingEninge. Then, in self.array, you can use the values of subscribed factors and set the length of self.array.

```
self.subscribe_factor("fastMA", "tsmean(close,12)")
self.subscribe_factor("slowMA", "tsmean(close,14)")
```

```
3.  self.set_data_size(3)
```

In this case, you set the data_size as 3 and you can see that two factors fastMA and slowMA attached to the basic bar data.

```
In [25]: engine2.strategy.array
Out[25]:
             open   high    low  ...  opeinterest      fastMA      slowMA
2019-11-01  620.0  630.0  614.0  ...      1571710  621.583333  622.857143
2019-11-04  625.0  626.0  612.5  ...      1533962  621.583333  620.785714
2019-11-05  614.0  624.5  610.5  ...      1573306  622.083333  621.142857

[3 rows x 8 columns]
```

4.3 OnBar

This method is called when a new bar is generated and is used to write the main part of the strategy logic. The user can use four functions buy, sell, sell_short and buy_to_cover to send orders.

```
1.  if self.array.fastMA.iloc[-1]>self.array.slowMA.iloc[-1] and self.pos == 0:
2.      self.buy(99999,10000)
3.  elif self.pos<0 and self.array.close.iloc[-1]>self.last_entry_price*(1+0.01):
4.      self.buy_to_cover(99999,-self.pos)
5.  elif self.array.fastMA.iloc[-1]<self.array.slowMA.iloc[-1] and self.pos >0:
6.      self.sell(1,self.pos)
7.  elif self.array.fastMA.iloc[-1]>self.array.slowMA.iloc[-1] and self.pos<0:
8.      self.buy_to_cover(99999,-self.pos)
9.  elif self.array.fastMA.iloc[-1]<self.array.slowMA.iloc[-1] and self.pos == 0:
10.     self.sell_short(1,10000)
```

4.4 OnOrder

This method is called when the status of an order is updated.

4.5 OnTrade

This method is called when there is a new trade.

5. How to write a correct a correct factor string

The mfe trader backtesting framework provides a simple way to compute factor. Therefore, there is a question. How to write a correct factor string?

Table 4 : Basic factors of factor computing

Basic factor	基本元
open	开盘价

high	最高价
low	最低价
close	收盘价
volume	成交量
opeinterest	持仓量

function	**功能**
+,-,*,/	加减乘除
rank(x)	横截面排序
delta(x,n)	X 当前值与 n 个 bar 之前的值之差
power(x,y)	X 的 y 次方
signedpower(x,y)	sign(x)*power(x,y)
delay(x,n)	X 在 n 个 bar 之前的值
corr(x,y,n)	X 和 y 的时间序列相关系数
cov(x,y)	X 和 y 的时间序列协方差
exp(x)	e^x
log(x)	log(x)
sign(x)	If x>0,sign(x)=1,If x<0,sign(x)<1
arccos(x),arcsin(x),arctan(x)arccosh(x) ,arcsinh(x),arctanh(x),cos(x),sin(x),tan(x), cosh(x),tanh(x)	三角/反三角/双曲函数
absolute(x)	绝对值
ceiling(x)	向上取整
floor(x)	向下取整
tsmin(x,n)	过去 n 个 bar 的最小值
tsmax(x,n)	过去 n 个 bar 的最大值
tsrank(x,n)	过去 n 个 bar 的归一化排序
std(x,n)	时间序列标准差
maximum(x,y)	两者中的最大值
minimum(x,y)	两者中的最小值
multiminimum(x,y,z,…)	多者中的最小值
sum(x,n)	过去 n 个 bar 的总和
product(x,n)	过去 n 个 bar 的乘积
scale(x)	横截面归一化
tsmean(x)	过去 n 个 bar 的平均值
lowday(x,n)	过去 n 个 bar 的最小值距离现在的 bar 数
highday(x,n)	过去 n 个 bar 的最大值距离现在的 bar 数
>,<,!=,==	大于、小于、不等于、等于

&，|	且、或
a?b:c	若 a 则 b，反之则 c

6. Other important attributions of strategies

name	function
self.bar_count	the number of bars since the beginning of backtesting
self.last_entry_price	The trade price of last entry
self.pos	Current position(number)
self.last_pos	The position of last bar
self.data_size	How many bars you need to use in the function OnBar

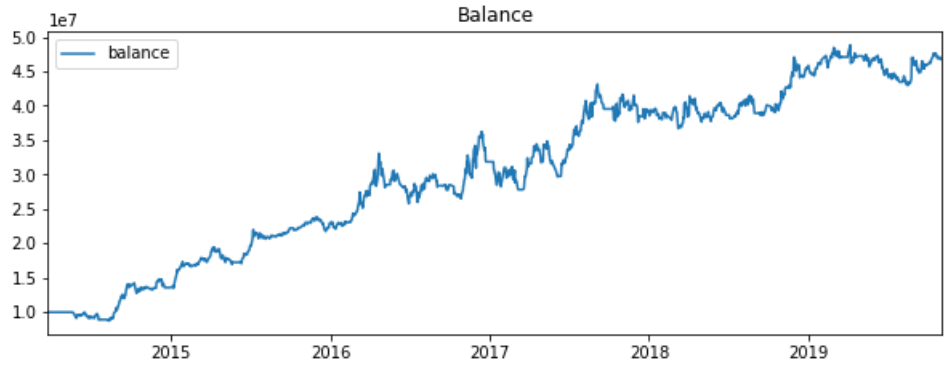
7. Multi-Backtesting combination

If we want to combine different strategies, one simple way is to use operator overloading to achieve it. In this case, we use the magic function __add__ to know how two different strategies can be combined to achieve better results.

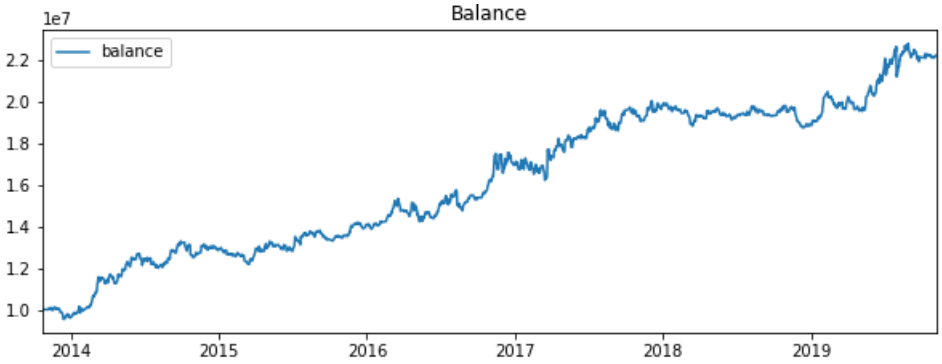
```
def __add__(self, *engine):
        """将若干个回测实例相加，实现多个资金曲线的叠加"""
    engine1 = BacktestingEngine()
    engine1.daily_df = DataFrame(columns = self.daily_df.columns)
    for i in engine:
        engine1.capital = self.capital + i.capital
        engine1.daily_df["net_pnl"] = self.daily_df["net_pnl"] + i.daily_df["net_pnl"]
        engine1.daily_df["commission"] = self.daily_df["commission"]+i.daily_df["commission"]
        engine1.daily_df["slippage"] = self.daily_df["slippage"]+i.daily_df["slippage"]
        engine1.daily_df["turnover"] = self.daily_df["turnover"]+i.daily_df["turnover"]
        engine1.daily_df["trade_count"] = self.daily_df["trade_count"]+i.daily_df["trade_count"]
    engine1.daily_df.dropna(axis=0,how="all",inplace=True)

    engine1.calculate_statistics()
    engine1.calculate_statistics()
    engine1.show_chart()

    return engine1
```

Therefore, if there are two backtesting engines whose strategy has been backtested, you can simply add them together and the result is the backtesting result of the combined strategy.

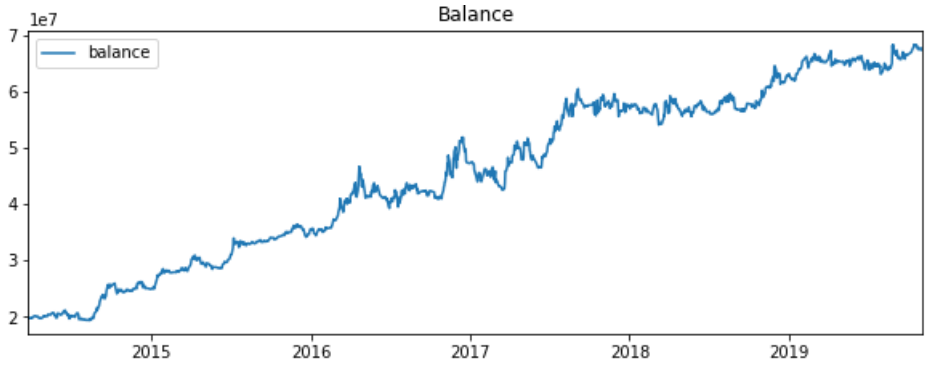
engine1:



engine2:



engine1+engine2:



8. Optimization:

```
1.  from datetime import datetime
2.  from mfe_trader.basics.constant import Interval
3.  from mfe_trader.backtesting.Optimization import Optimization
4.  from mfe_trader.strategy.strategies.A_Strategy import A_Strategy
5.
6.  setting1 = {"params":{"fastLength":range(1,10),
7.                        "slowLength":range(2,20),
8.                        "volume":range(1,10)
```

```
                },
        "target":"sharpe_ratio",
        "backtesting":{"start":datetime(2011,1,5,0,0),
                "end":datetime(2019,1,5,0,0),
                "captial":10000000,
                "rate":0.001,
                "slippage":0},
        "data":{"vtsymbol":"RB888.SHFE",
                "interval": Interval.Day_1,
                "path": "rb888_1d.csv",
                "datetime_row": "datetime"},
        "strategy": A_Strategy
       }
opt = Optimization(setting1)
opt.init_strategy()
opt.run_optimization()
```