Trading Strategy: Back testing with Backtrader

Kyle Li
Aug 13, 2018 · 4 min read

Photo by Chris Liverani on Unsplash

Note: I start to migrate my blogs to https://kylelix7.github.io/. I will start posting new ideas there as well.

...

In one of my older post, I demonstrates how to compute technical indicators which can be combined logically to build a trading strategy. As emphasized in the post, one should validate how well the strategy does with backtesting before applying it in real market.

> *Backtesting is the process of applying a trading strategy or analytical method to historical data to see how accurately the strategy or method would have predicted actual results.*
> *- from investing answers*

Backtest is like cross validation in machine learning. But it's not exactly the same. Backtest requires splitting data into two parts like cross validation. One set is for training, the other is for validation purpose. The difference is training testing split can be randomly done for cross validation. While in trading backtesting, your data is time series. Your training data must be older than your testing data. Otherwise you peek in future which results in incorrect measurement of your strategy. So dataset split cannot be random in backtesting. Backtesting involves market simulation in real world. It could be hard and error-prone to implement your own backtesting libraries. Luckily there's Backtrader.

Backtrader is an awesome open source python framework which allows you to focus on writing reusable trading strategies, indicators and analyzers instead of having to spend time building infrastructure. It supports backtesting for you to evaluate the strategy you come up with too!

With that being said, it is a free and complete solution for technical people to build their own strategies. Let’s start to have a glance.

Installation and setup

```
pip install backtrader[plotting]
```

Build a strategy

Backtrader has defined a strategy interface for you. You need to create a class with implement this interface. An important method is next() where you should make decision whether you should BUY, SELL or DO NOTHING based on the technical indicators in a specific day. A simple strategy looks like this.

```
import backtrader as bt
class MyStrategy(bt.Strategy):

    def __init__(self):
        self.sma =
bt.indicators.SimpleMovingAverage(period=15)

    def next(self):
        if self.sma > self.data.close:
            # Do something
            pass

        elif self.sma < self.data.close:
            # Do something else
            pass
```

As you can see, backtrader has shipped with a set of common technical indicators. It means you don't need to reply on your self or TA lib to compute technical indicators.

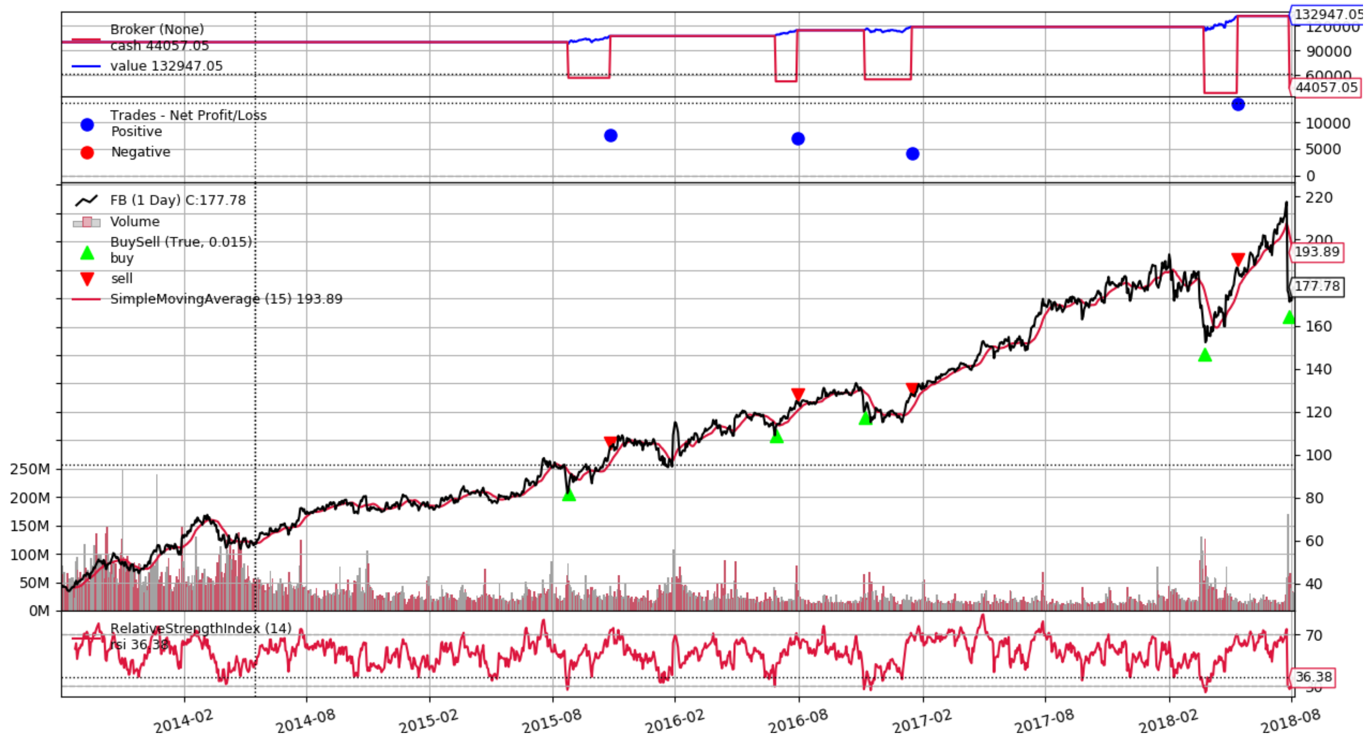
Backtrader also offers features in simulating trading in the marking. Once can factor the commission in your trading operation based on dollar or percentage.

```
cerebro.broker.setcommission(commission=0.001)
```

Below is the whole example for demonstration of backtesting with Facebook historical market data. Note that, historical trading data is downloaded from Yahoo Finance. It also supports pandas dataframe. I have a post about collecting trading data with pandas here. The example consists of a simple TestStrategy and a driver piece of code that kick of the backtesting. The simple strategy only considers RSI for BUY/SELL signal. You should add more logics for your selected stocks.

```
from __future__ import (absolute_import, division,
print_function,
                        unicode_literals)

import datetime
import os.path
import sys
import backtrader as bt

class TestStrategy(bt.Strategy):
    def log(self, txt, dt=None):
        dt = dt or self.datas[0].datetime.date(0)
        print('%s, %s' % (dt.isoformat(), txt))

    def __init__(self):
        self.dataclose = self.datas[0].close
        self.order = None
        self.buyprice = None
        self.buycomm = None

        self.sma = 
bt.indicators.SimpleMovingAverage(self.datas[0], 
period=15)
        self.rsi = bt.indicators.RelativeStrengthIndex()

    def notify_order(self, order):
        if order.status in [order.Submitted, 
order.Accepted]:
            return

        if order.status in [order.Completed]:
            if order.isbuy():
                self.log(
                    'BUY EXECUTED, Price: %.2f, Cost: 
%.2f, Comm %.2f' %
                    (order.executed.price,
                     order.executed.value,
                     order.executed.comm))

                self.buyprice = order.executed.price
                self.buycomm = order.executed.comm
            else:  # Sell
                self.log('SELL EXECUTED, Price: %.2f, 
Cost: %.2f, Comm %.2f' %
                         (order.executed.price,
                          order.executed.value,
                          order.executed.comm))

            self.bar_executed = len(self)

        elif order.status in [order.Canceled, 
order.Margin, order.Rejected]:
            self.log('Order Canceled/Margin/Rejected')

        # Write down: no pending order
        self.order = None

    def notify_trade(self, trade):
        if not trade.isclosed:
            return

        self.log('OPERATION PROFIT, GROSS %.2f, NET %.2f'
%
                 (trade.pnl, trade.pnlcomm))

    def next(self):
        self.log('Close, %.2f' % self.dataclose[0])
        print('rsi:', self.rsi[0])
        if self.order:
            return

        if not self.position:
            if (self.rsi[0] < 30):
                self.log('BUY CREATE, %.2f' %
self.dataclose[0])
                self.order = self.buy(size=500)

        else:
            if (self.rsi[0] > 70):
                self.log('SELL CREATE, %.2f' %
self.dataclose[0])
                self.order = self.sell(size=500)


if __name__ == '__main__':
    cerebro = bt.Cerebro()
    cerebro.addstrategy(TestStrategy)
    cerebro.broker.setcommission(commission=0.001)

    datapath = 'FB.csv'

    # Create a Data Feed
    data = bt.feeds.YahooFinanceCSVData(
        dataname=datapath,
        fromdate=datetime.datetime(2013, 1, 1),
        todate=datetime.datetime(2018, 8, 5),
        reverse=True)

    cerebro.adddata(data)
    cerebro.broker.setcash(100000.0)
    print('Starting Portfolio Value: %.2f' %
cerebro.broker.getvalue())
    cerebro.run()
    print('Final Portfolio Value: %.2f' %
cerebro.broker.getvalue())
    cerebro.plot()
```

At the end of execution, you can find out your final value of portfolio. As well, you are able to plot the stock price, technical indicators, your BUY/SELL operations and your portfolio value with regard to the time.



As you can see, this simple strategy works ok with FB as it captures a few buy and sell opportunities.

That's it for backtesting with backtrader. If you want to dive deeper, I encourage you visit backtrader's doc for more advanced usage. Happy coding and trading!

. . .

Recommended reading:

What Hedge Funds Really Do

Python for Data Analysis: Data Wrangling with Pandas, NumPy, and IPython

. . .

My other posts:

Reinforcement Learning: Introduction to Q Learning

Trading Strategy: Technical Analysis with Python TA-Lib

Portfolio Optimization for Minimum Risk with Scipy — Efficient Frontier Explained

Reducing Risk by Building Portfolio

Trading: Calculate Technical Analysis Indicators with Pandas 🐼

Collect Trading Data with Pandas

Python Data Science Trading Analytics Finance