Python量化交易学习笔记（25）——Data Feeds扩展

2020-05-06

feed python

背景：需要扩展data feeds的场景

在backtrader中，data feeds中包含了被普遍认为是业界标准的几个字段：

- datetime
- open
- high
- low
- close
- volume
- openinterest

可以使用GenericCSVData读取CSV文件，来方便地加载这些数据。但是在很多情况下，还需要在回测框架中使用其他的数据，例如：

1. 利用分类算法，预测出股票是否已经达到买点及卖点（转化为分类问题），在backtrader中按照预测的买点及卖点进行交易，就需要将预测结果读入到框架中进行回测。
2. 利用序列预测算法，预测出模型每日的收盘价，将预测值与其他技术指标综合分析，制定交易策略，然后进行回测，这也需要将预测得到的序列值读入到框架中。

因此，我们要想办法从CSV中读入自定义的数据，来使得这些数据可以在策略中得以应用。本文就针对上面提到的第2个场景，来扩展data feeds，实现自定义数据在backtrader回测框架中的使用。

应用场景设定

我们假定这样的应用场景：

1. 已知某只股票的历史日线数据，包含datetime，open，high，low，close字段；
2. 已经通过序列预测算法，利用历史数据计算出每日收盘价（或开盘价）的预测值；
3. 利用历史日线数据及预测数据对该股票进行策略回测；
4. 策略为：当收盘价小于前1日的预测值时，进行买入；当收盘价大于前1日的预测值时，进行卖出。

实现步骤

1. 生成待读入CSV文件，文件中包含datetime，open，high，low，close及自定义数据字段（这里使用predict进行标识）。由于这里只是做示意，因此简单的使用predict = (high + low) / 2来计算predict的值。合并后CSV文件截图如下：

	A	B	C	D	E	F
1	date	open	high	low	close	predict
2	2019/1/2	9.39	9.42	9.16	9.19	9.29
3	2019/1/3	9.18	9.33	9.15	9.28	9.24
4	2019/1/4	9.24	9.82	9.22	9.75	9.52
5	2019/1/7	9.84	9.85	9.63	9.74	9.74
6	2019/1/8	9.73	9.74	9.62	9.66	9.68
7	2019/1/9	9.74	10.08	9.7	9.94	9.89
8	2019/1/10	9.87	10.2	9.86	10.1	10.03
9	2019/1/11	10.11	10.22	10.05	10.2	10.135
10	2019/1/14	10.22	10.25	10.07	10.11	10.16

```
# 扩展DataFeed
class GenericCSVDataEx(GenericCSVData):
    # 添加自定义line
    lines = ('predict', )
    # openinterest在GenericCSVData中的默认索引是7，这里对自定义的line的索引加1，用户可指定
    params = (('predict', 8),)
```

这里创建了一个GenericCSVData的子类GenericCSVDataEx，定义了一个新的lines对象predict，并且添加了一个新的参数predict，默认值设置为8，在后续调用时，根据自定义数据具体在文件中的哪一列，再对这个参数进行重新赋值。

3. 使用创建的子类导入数据，代码如下：

```
# 创建数据
data = GenericCSVDataEx(
        dataname = datapath,
        fromdate = datetime.datetime(2019, 1, 1),
        todate = datetime.datetime(2019, 12, 31),
        nullvalue = 0.0,
        dtformat = ('%Y/%m/%d'),
        datetime = 0,
        open = 1,
        high = 2,
        low = 3,
        close = 4,
        volume = -1,
        openinterest = -1,
        predict = 5
        )
```

这里使用了新创建的类GenericCSVDataEx来导入数据。在参数中，从datetime开始，等号后面的值均表示对应字段在CSV文件中的列号，-1表示文件没有该字段数据，可以回看前面的CSV文件的截图，确认一下数据的对应关系。

4. 在Strategy中使用自定义字段数据，主要代码如下：

```
class TestStrategy(bt.Strategy):
    params = (
        # 要跳过的K线根数
        ('skip_len', 1),
    )
    def __init__(self):
        # 引用data[0]数据的收盘价数据
        self.dataclose = self.datas[0].close
        self.datapredict = self.datas[0].predict
        # 用于绘制predict曲线
        btind.SMA(self.data.predict, period = 1, subplot = False)
        # 用于记录订单状态
        self.order = None
        self.buyprice = None
        self.buycomm = None
    def next(self):
        # 因为需要在策略中需要和前一日的预测值比较，所以要跳过第1根K线
        if (len(self) <= self.p.skip_len):
            return
        # 检查是否有订单等待处理，如果是就不再进行其他下单
        if self.order:
            return
        # 检查是否已经进场
```

```
            if self.dataclose[0] < self.datapredict[-1]:
                # 买买买
                # 记录订单避免二次下单
                self.order = self.buy()
        # 如果已经在场内，则可以进行卖出操作
        else:
            # 当日收盘价大于前一日预测价
            if self.dataclose[0] > self.datapredict[-1]:
                # 卖卖卖
                # 记录订单避免二次下单
                self.order = self.sell()
```

在init函数中，直接使用self.datas[0].predict就可以访问到我们自定义的predict字段的值，然后将它保存在实例变量self.datapredict中，这样就可以在next函数中进行使用了。

```
        self.datapredict = self.datas[0].predict
```

在next函数中实现策略：当收盘价小于前1日的预测值时，进行买入；当收盘价大于前1日的预测值时，进行卖出。

```
        if not self.position:
            if self.dataclose[0] < self.datapredict[-1]:
                self.order = self.buy()
        else:
            if self.dataclose[0] > self.datapredict[-1]:
                self.order = self.sell()
```

需要说明以下几点：

- 这里的self.datapredict（self.datas[0].predict）是lines对象，在next函数中使用时，索引[0]表示当日的数据，索引[-1]表示前1日的数据
- 由于要和前1日predict的值做比较，而对于第1个交易日而言，是没有前1日predict值的，因此在next函数中，使用下面的代码跳过1根K线

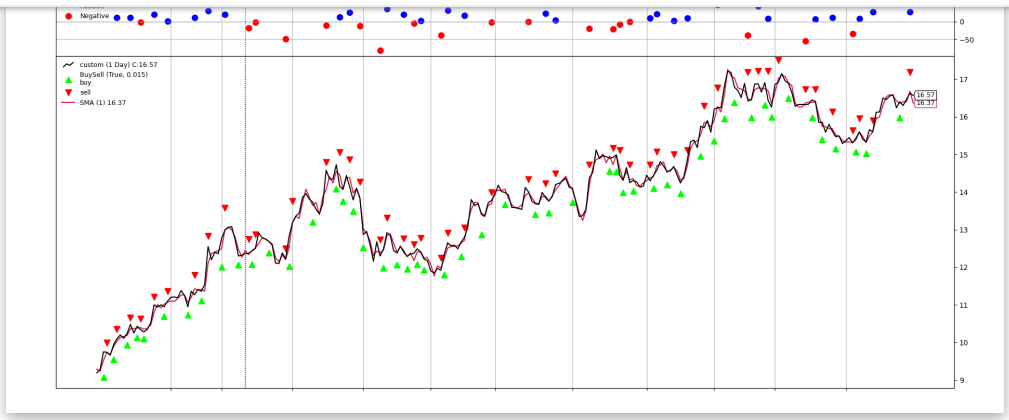
```
        # 因为需要在策略中需要和前一日的预测值比较，所以要跳过第1根K线
        if (len(self) <= self.p.skip_len):
            return
```

- backtrader没有提供自定义的数据绘制功能，可以在init函数中，通过借用单日的简单移动平均线来绘制自定义数据的曲线

```
        # 用于绘制predict曲线
        btind.SMA(self.data.predict, period = 1, subplot = False)
```

回测结果如下图所示：

码农家园



Data Feeds扩展代码：

```
from __future__ import (absolute_import, division, print_function,
                        unicode_literals)
import datetime  # 用于datetime对象操作
import os.path  # 用于管理路径
import sys  # 用于在argvTo[0]中找到脚本名称
import backtrader as bt # 引入backtrader框架
from backtrader.feeds import GenericCSVData # 用于扩展DataFeed
import backtrader.indicators as btind

# 扩展DataFeed
class GenericCSVDataEx(GenericCSVData):
    # 添加自定义line
    lines = ('predict', )
    # openinterest在GenericCSVData中的默认索引是7，这里对自定义的line的索引加1，用户可指
    params = (('predict', 8),)


# 创建策略
class TestStrategy(bt.Strategy):
    params = (
        # 要跳过的K线根数
        ('skip_len', 1),
    )
    def log(self, txt, dt=None):
        ''' 策略的日志函数'''
        dt = dt or self.datas[0].datetime.date(0)
        print('%s, %s' % (dt.isoformat(), txt))
    def __init__(self):
        # 引用data[0]数据的收盘价数据
        self.dataclose = self.datas[0].close
        self.datapredict = self.datas[0].predict
        # 用于绘制predict曲线
        btind.SMA(self.data.predict, period = 1, subplot = False)
        # 用于记录订单状态
        self.order = None
        self.buyprice = None
        self.buycomm = None
    def notify_order(self, order):
        if order.status in [order.Submitted, order.Accepted]:
            # 提交给代理或者由代理接收的买/卖订单 - 不做操作
            return
        # 检查订单是否执行完毕
        # 注意：如果没有足够资金，代理会拒绝订单
        if order.status in [order.Completed]:
            if order.isbuy():
```

```
                    order.executed.value,
                    order.executed.comm))

                self.buyprice = order.executed.price
                self.buycomm = order.executed.comm
            else: # 卖
                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')
        # 无等待处理订单
        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):
        # 因为需要在策略中需要和前一日的预测值比较，所以要跳过第1根K线
        if (len(self) <= self.p.skip_len):
            return
        # 日志输出收盘价数据
        self.log('Close, %.2f' % self.dataclose[0])
        # 检查是否有订单等待处理，如果是就不再进行其他下单
        if self.order:
            return
        # 检查是否已经进场
        if not self.position:
            # 还未进场，则只能进行买入
            # 当日收盘价小于前一日预测价
            if self.dataclose[0] < self.datapredict[-1]:
                # 买买买
                # 记录订单避免二次下单
                self.log('BUY CREATE, %.2f' % self.dataclose[0])
                self.order = self.buy()
        # 如果已经在场内，则可以进行卖出操作
        else:
            # 当日收盘价大于前一日预测价
            if self.dataclose[0] > self.datapredict[-1]:
                # 卖卖卖
                # 记录订单避免二次下单
                self.log('SELL CREATE, %.2f' % self.dataclose[0])
                self.order = self.sell()
# 创建cerebro实体
cerebro = bt.Cerebro()
# 添加策略
cerebro.addstrategy(TestStrategy)
# 先找到脚本的位置，然后根据脚本与数据的相对路径关系找到数据位置
# 这样脚本从任意地方被调用，都可以正确地访问到数据
modpath = os.path.dirname(os.path.abspath(sys.argv[0]))
datapath = os.path.join(modpath, './custom.csv')
# 创建数据
data = GenericCSVDataEx(
        dataname = datapath,
        fromdate = datetime.datetime(2019, 1, 1),
        todate = datetime.datetime(2019, 12, 31),
```

```
        open = 1,
        high = 2,
        low = 3,
        close = 4,
        volume = -1,
        openinterest = -1,
        predict = 5
        )
# 在Cerebro中添加价格数据
cerebro.adddata(data)
# 设置启动资金
cerebro.broker.setcash(100000.0)
# 设置交易单位大小
cerebro.addsizer(bt.sizers.FixedSize, stake = 100)
# 设置佣金为千分之一
cerebro.broker.setcommission(commission=0.001)
# 打印开始信息
print('Starting Portfolio Value: %.2f' % cerebro.broker.getvalue())
# 遍历所有数据
cerebro.run()
# 打印最后结果
print('Final Portfolio Value: %.2f' % cerebro.broker.getvalue())
cerebro.plot()
```