高级量化交易技术

闫涛
科技有限公司
北京 2021.05.08
{yt7589}@qq.com

31st Conference on Neural Information Processing Systems (NIPS 2017), Long Beach, CA, USA.

第一篇量化交易框架

第 1 章行情数据处理

Abstract

在本章中我们将通过 AKshare 库，获取 A 股分钟级行情数据，并将其进行预处理，变为深度学习可用的数据集。

1 行情数据处理概述

1.1 获取原始行情数据

我们首先通过 apps.fmts.ds.akshare_data_source.AkshareDataSource 获取原始的行情数据，-将其保存到 csv 文件中。如果存在该 csv 文件，则直接从该文件中读出数据并返回。数据格式为：

```
1 ......
2 ['2021-08-17 14:55:00', 30.339999999999996, 30.339999999999996,
     30.339999999999996, 30.339999999999996, 200.0]
3 ['2021-08-17 14:55:01', 30.339999999999996, 30.339999999999996,
     30.339999999999996, 30.339999999999996, 200.0]
4 ......
```

Listing 1: 行情数据格式

1.2 行情数据预处理

1.2.1 价格折线图

我们以收盘价为例，收盘价的折线图绘制程序如下所示：

```
1     class OhlcvProcessor(object):
2     # 价格折线图模式
3     PCM_DATETIME = 1
4     PCM_TICK = 2
5
6     @staticmethod
7     def draw_close_price_curve(stock_symbol: str, mode=1) -> None:
8         '''
9         绘制收盘价折线图, 横轴为时间, 纵轴为收盘价
10        '''
11        data = AkshareDataSource.get_minute_bars(stock_symbol=
    stock_symbol)
12        x = [v[0] for v in data[0:1000]]
13        y = [v[4] for v in data[0:1000]]
14        if mode == OhlcvProcessor.PCM_DATETIME:
15            OhlcvProcessor._draw_date_price_curve(x, y)
16        else:
17            OhlcvProcessor._draw_tick_price_curve(y)
18
19    def _draw_date_price_curve(x: List, y: List) -> None:
```

```
20          x = [datetime.datetime.strptime(di, '%Y-%m-%d %H:%M:%S')
    for di in x]
21          fig, axes = plt.subplots(1, 1, figsize=(8, 4))
22          plt.rcParams['font.sans-serif']=['SimHei'] #用来正常显示中
    文标签
23          plt.rcParams['axes.unicode_minus'] = False #用来正常显示负
    号
24          # 最大化绘图窗口
25          figmanager = plt.get_current_fig_manager()
26          figmanager.window.state('zoomed')     #最大化
27          # 绘制收盘价格折线图
28          axes.plot_date(x, np.array(y), '-', label='Net Worth')
29          # 设置横轴时间显示格式
30          axes.xaxis.set_major_formatter(DateFormatter('%Y-%m-%d %H
    :%M:%S'))
31          plt.gcf().autofmt_xdate()
32          # 显示图像
33          plt.show()
34
35      def _draw_tick_price_curve(y: List) -> None:
36          x = range(len(y))
37          fig, axes = plt.subplots(1, 1, figsize=(8, 4))
38          plt.rcParams['font.sans-serif']=['SimHei'] #用来正常显示中
    文标签
39          plt.rcParams['axes.unicode_minus'] = False #用来正常显示负
    号
40          # 最大化绘图窗口
41          figmanager = plt.get_current_fig_manager()
42          figmanager.window.state('zoomed')     #最大化
43          # 绘制收盘价格折线图
44          plt.title('收盘价折线图')
45          axes.set_xlabel('时间刻度')
46          axes.set_ylabel('收盘价')
47          axes.plot(x, np.array(y), '-', label='Net Worth')
48          plt.show()
```

Listing 2: 收盘价折线图

代码解读如下所示：

- 第 3、4 行：定义收盘价曲线绘制方式，一种是横轴为时间，另一种横轴为行情序号；
- 第 6~10 行：定义收盘价绘制方法，参数为股票代码和绘制模式，缺省值为横轴为时间（以分钟为单位），这种模式的缺点是从上一日收盘到下一日开盘有较大的时间间隔；
- 第 11 行：获取分钟线行情数据，格式为：[[dateteime, open, high, low, close, volume]]；
- 第 19 行：以横轴为行情时间值绘制收盘价曲线；

- 第 20 行：将时间变为'2021-08-21 12:56:00' 格式的列表；
- 第 21 行：设置显示图形；
- 第 22 行：设置字体使 matplotlib 可以正确显示汉字；
- 第 23 行：使 matplotlib 可以显示负号；
- 第 24~26 行：使 matplotlib 绘图窗口最大化；
- 第 27、28 行：绘制收盘价时间曲线；
- 第 29~31 行：设置横坐标轴时间显示格式为'2021-08-21 12:56:00'，并自动调整为 45 度角倾斜，以节省显示空间；

图 1: 以时间为横轴的收盘价折线图

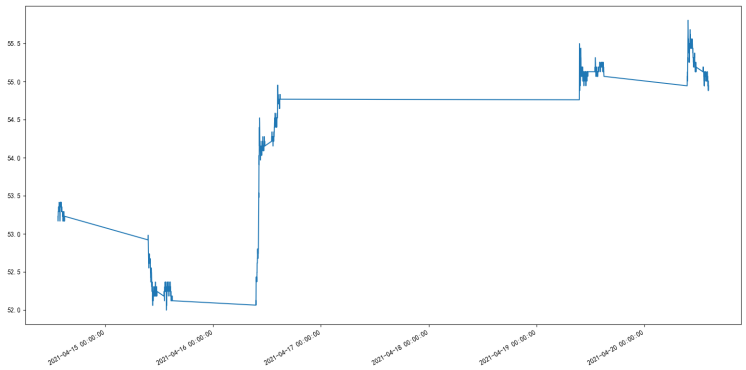
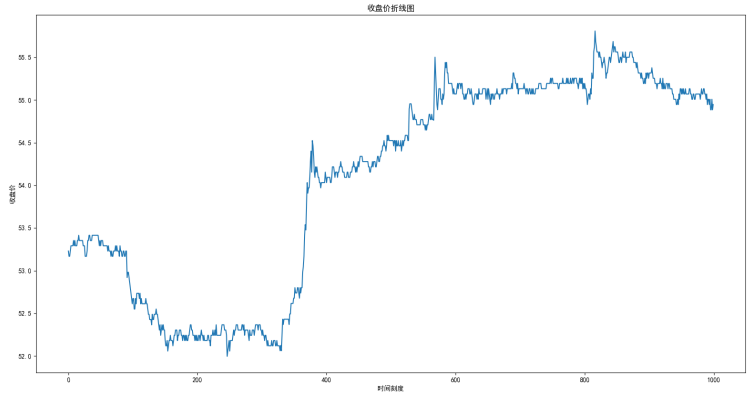


图 2: 以序号为横轴的收盘价折线图



如1所示，图中每天收盘到第二天开盘间没有行情数据，所以图形不太好看出规律，而图2则可以较好的反映价格的变化规律，因此我们在通常情况下，选择图2的形式。

1.2.2 对数差分序列

我们都知道，原始的行情数据，不具备平稳性，即无法通过历史数据来预测未来，而对数差分序列则具有平稳性，可以用来进行预测。如下所示：

```
1     @staticmethod
2     def gen_1d_log_diff_norm(stock_symbol, items):
3         '''
```

```
4         从原始行情数据，求出一阶对数收益率log(day2)-log(day1)，然
    后求出每列均值和标准差，利用
5         (x-mu)/std进行标准化，分别保存原始信息和归整后信息
6         参数:
7             stock_symbol 股票编号
8             items 由AkshareDataSource.get_minute_bars方法获取到
9         '''
10        datas = np.array([x[1:] for x in items])
11        log_ds = np.log(datas)
12        log_diff = np.diff(log_ds, n=1, axis=0)
13        log_diff_mu = np.mean(log_diff, axis=0)
14        log_diff_std = np.std(log_diff, axis=0)
15        ld_ds = (log_diff - log_diff_mu) / log_diff_std
16        # 保存原始信息
17        raw_file = './apps/fmts/data/{0}_1m_raw.txt'.format(
    stock_symbol)
18        with open(raw_file, 'w', encoding='utf-8') as fd:
19            for item in items[1:]:
20                fd.write('{0},{1},{2},{3},{4},{5}\n'.format(item
    [0], item[1],
21                                                            item[2], item[3], item[4], item[5]))
22        # 保存规整化后数据
23        ld_file = './apps/fmts/data/{0}_1m_ld.csv'.format(
    stock_symbol)
24        np.savetxt(ld_file, ld_ds)
25
26# 测试程序
27    def test_gen_1d_log_diff_norm_001(self):
28        stock_symbol = 'sh600260'
29        items = AkshareDataSource.get_minute_bars(stock_symbol=
    stock_symbol)
30        OhlcvProcessor.gen_1d_log_diff_norm(stock_symbol, items)
```

Listing 3: 收盘价折线图

代码解读如下所示：

- 第 2 行：items 由 AkshareDataSource.get_minute_bars 方法获取到，格式为 [..., ['2021-08-19 15:00:00', 1.1, 1.5, 1.0, 1.2, 1000], ...]；
- 第 10 行：把 items 中的条目，去除掉日期列后，生成 ndarray；
- 第 11 行：对所有元素取对数，以自然数 e 为底，np.log2 是以 2 底，np.log10 是以 10 为底；
- 第 12 行：取一阶差分，其中 n=1 代表是一阶差分，即后面一个元素减前面一个元素，out[i] = x[i+1]-x[i]，因为 axis=0，所以 i 代表行；
- 第 13 行：求出行方向的均值；
- 第 14 行：求出行方向的标准差；

- 第 15 行：进行归一化：$\hat{x} = \frac{x-\mu}{\sigma}$；
- 第 18 21 行：保存原始的行情信息，因为取了一阶差分，所以去掉了第 1 行；
- 第 23、24 行：保存一阶差分规整化后的数据；

1.3 数据集支撑数据

在每一个时间点，我们向前看 window_size 个时间点，缺省是 10 个，然后再加上当前时间点的数据：开盘、最高、最低、收盘、交易量，所以共有 10×5+5 = 55 个数据，我们的算法会根据这 55 维向量，我们以当前时刻收盘价为标准，确定判断为上涨趋势的最低价格（一旦超过该值即视为上涨），判断为下跌的最高价格（一旦低于该值即视为下跌），向后连续读取指定个时刻，缺省值为 100，如果未来价格首先高于上涨趋势的最低价格，则将当前时刻判断为上涨状态，如果我们有资金，就应该进行买入操作；如果未来价格首先低于下跌趋势的最高价格，则将当前时刻判断为下跌状态，此时如果我们有持仓，则应卖出持有的股票，如果既没高于上涨趋势的最低价格，也没低于下跌趋势的最高价格，则将当前时刻判断为震荡状态，此时不进行任何操作。我们先来看数据的生成：

```
1     @staticmethod
2     def get_ds_raw_data(stock_symbol: str, window_size: int=10,
      forward_size: int=100) -> Tuple[np.ndarray, np.ndarray, List[
      str]]:
3         '''
4         获取数据集所需数据
5         stock_symbol 股票代码
6         window_size 从当前时间点向前看多少个时间点
7         forward_size 向后看多少个时间点确定市场行情是上涨、下跌和
      震荡
8         返回值
9             X 连续11个时间点的OHLCV的数据，形状为n*55，一阶Log差分
      形式
10            y 某个时间点及其前10个时间点行情数据组成的shapelet对应
      的行情（按Box方式确定）：0-震荡；1-上升；2-下跌；
11            info 当前时间刻行情的真实值
12        '''
13        print('获取数据集数据')
14        # 获取行情数据
15        quotations = OhlcvProcessor.get_quotations(stock_symbol)
16        # 获取归整化行情数据
17        log_1d_datas = []
18        log_1d_file = './apps/fmts/data/{0}_1m_ld.csv'.format(
      stock_symbol)
19        with open(log_1d_file, 'r', encoding='utf-8') as fd:
20            for row in fd:
21                row = row.strip()
22                arrs = row.split(' ')
23                item = [arrs[0], arrs[1], arrs[2], arrs[3], arrs
      [4]]
24                log_1d_datas.append(item)
```

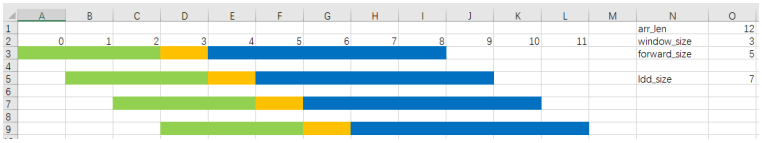
```
25          #
26          ldd_size = len(log_1d_datas) - forward_size
27          print('ldd_size: {0};'.format(ldd_size))
28          X_raw = []
29          for pos in range(window_size, ldd_size, 1):
30              item = []
31              for idx in range(pos-window_size, pos):
32                  item += log_1d_datas[idx]
33              item += log_1d_datas[pos]
34              X_raw.append(item)
35          X = np.array(X_raw, dtype=np.float32)
36          ds_X_csv = './apps/fmts/data/{0}_1m_X.csv'.format(
    stock_symbol)
37          np.savetxt(ds_X_csv, X, delimiter=',')
38          # 获取行情状态
39          y = np.zeros((X.shape[0],), dtype=np.int64)
40          OhlcvProcessor.get_market_state(y, quotations, window_size
    , forward_size)
41          # 获取日期和真实行情数值
42          raw_datas = []
43          raw_data_file = './apps/fmts/data/{0}_1m_raw.txt'.format(
    stock_symbol)
44          seq = 0
45          with open(raw_data_file, 'r', encoding='utf-8') as fd:
46              for row in fd:
47                  if seq > window_size and seq<=ldd_size:
48                      row = row.strip()
49                      arrs = row.split(',')
50                      item = [arrs[0], float(arrs[1]), float(arrs
    [2]), float(arrs[3]), float(arrs[4]), float(arrs[5])]
51                      raw_datas.append(item)
52                  seq += 1
53          a1 = len(raw_datas)
54          return X[:a1], y[:a1], raw_datas
```

Listing 4: 获取数据集后面原始数据

代码解读如下所示：

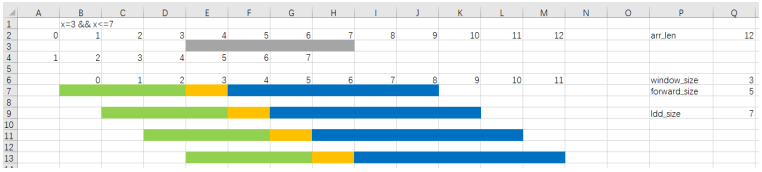
- 第 15 行：获取行情数据，格式为：[..., [open, high, low, close, volume], ...]，类型为 numpy.ndarray；
- 第 17~24 行：从文件中读出一阶对数差分形式的数据集，包括规整化后的开盘、最高、最低、收盘、交易量信息；
- 第 26 行：我们需要在当前时间点向后看 forward_size 个时间点，因此最后一个时间点索引值为 ldd_size=length-forward_size，不包括该值；

图 3: 读取行情数据原理示意图



- 第 28~34 行：起点为 window_size，如上图行 3 橙色单元格，终点为 ldd_size=length-forward_size=12-5=7，不包括该值，每步循环时，当前时刻向右移一格，将前面 window_size=3 个时间点数据加入到 item 中，最后将当前时间点 pos 加入到 item 中，最后将 item 作为一个样本，加入到原如数据集中；
- 第 35~37 行：将其转变为 np.ndarray，并保存到如 sh600260_1m_X.csv 中；
- 第 39、40 行：生成与 X 同长度的 y，每个时间点是一个 0、1、2 中三个数中的一个数，分别代表上涨、下跌和震荡，具体实现在下面的进行详细介绍；

图 4: 读取原始行情数据原理示意图



- 第 42~52 行：由于我们在上步读取数据集时，针对的是一阶对数差分序列，所以其比原始数据（上图上部）少一列数据。
  - 初始时 seq=0，不满足 if 条件，不处理第 0 时刻数据，seq=1；
  - seq=1 不满足 if 条件，不处理第 1 时刻数据，seq=2；
  - seq=2 不满足 if 条件，不处理第 2 时刻数据，seq=3；
  - seq=3 不满足 if 条件，不处理第 3 时刻数据，seq=4；
  - seq=4 满足 if 条件，将第 4 时刻数据加入到行情数据中，seq=5；
  - seq=5 满足 if 条件，将第 5 时刻数据加入到行情数据中，seq=6；
  - seq=6 满足 if 条件，将第 5 时刻数据加入到行情数据中，seq=7；
  - seq=7 满足 if 条件，将第 5 时刻数据加入到行情数据中，seq=8；
  - seq=8 满足 if 条件，将第 5 时刻数据加入到行情数据中，seq=9；
  - ……；
- 第 54 行：返回 X、y 和原始行情数据 raw_datas；

1.4 确定市场行情

下面我们来看怎样根据前 10 个时刻及当前时刻的行情数据，确定在未来是上涨、下跌或震荡行情，从而根据仓位选择合理的操作。

```
# apps/fmts/ds/ohlcv_processfor.py
    @staticmethod
    def get_market_state(y: np.ndarray, quotation: np.ndarray,
    window_size: int, forward_size: int) -> None:
        '''
```

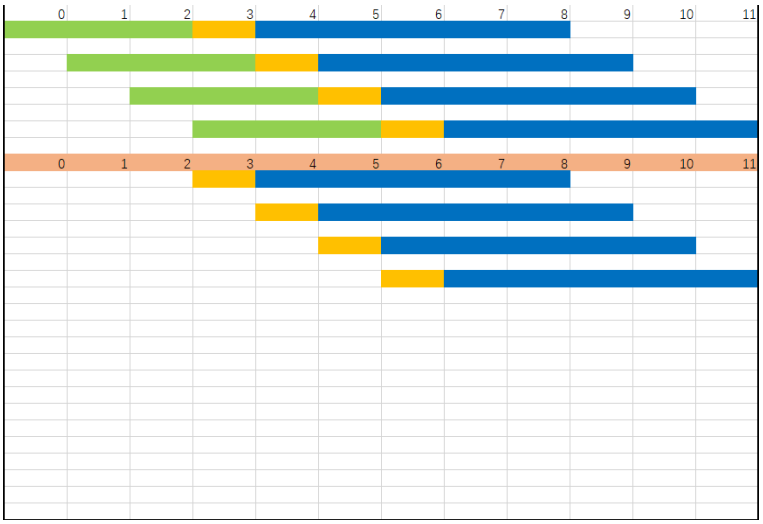
```
        针对收盘价，从window_size处开始，向后看forward_size条记录，上限为当前收盘价*1.01，下限为当前收盘价*0.95，当
        在forward_size窗口内收盘价高于上限时返回0表示上升行情需要买入，低于下限时返回1表示下跌行情需要卖出，
        否则返回2表示震荡行情，将该值写入y中
        '''
        cnt = y.shape[0]
        q_cnt = quotation.shape[0]
        for idx in range(0, cnt, 1):
            curr_price = quotation[idx+window_size][3]
            high_limit = curr_price * 1.01
            low_limit = curr_price * 0.995
            market_regime = 2
            for pos in range(idx+window_size+1, idx+window_size+1+forward_size, 1):
                future_price = quotation[pos][3]
                if future_price >= high_limit:
                    market_regime = 0
                elif future_price <= low_limit:
                    market_regime = 1
            y[idx] = market_regime


# 测试程序: utcs/apps/fmts/ds/t_ohlcv_processor.py
    def test_get_market_state002(self):
        # 开始下标: window_size, 结束下标: cnt-forward_size-1（包含）
        random.seed(1.0)
        y = np.zeros((10,), dtype=np.int64)
        quotation_raw = []
        cnt_y = 10
        curr_price = 5.36
        high_delta = 1.01
        low_delta = 0.995
        window_size = 3
        forward_size = 4
        cnt = cnt_y+window_size+forward_size
        for idx in range(cnt):
            close_price = random.uniform(curr_price*low_delta, curr_price*high_delta)
            item = [0.01, 0.02, 0.03, close_price, 0.04]
            quotation_raw.append(item)
        print('quotation_raw: {0}: {1};'.format(len(quotation_raw), quotation_raw))
        quotation = np.array(quotation_raw)
```

```
44          OhlcvProcessor.get_market_state(y, quotation, window_size,
    forward_size)
45          print('y: {0}; {1};'.format(y.shape, y))
46          plt.ion()
47          fig, axes = plt.subplots(1, 1, figsize=(8, 4))
48          for idx in range(cnt_y):
49              plt.draw()
50              plt.pause(0.1)
51              plt.cla()
52              # 绘制价格变化曲线
53              x = range(cnt)
54              close_prices = [ix[3] for ix in quotation]
55              plt.plot(x, close_prices, color='goldenrod', marker='*
    ')
56              # 绘制最左侧竖线
57              low_limit = quotation[idx+window_size][3]*low_delta
58              high_limit = quotation[idx+window_size][3]*high_delta
59              x1 = np.array([idx+window_size, idx+window_size])
60              y1 = np.array([low_limit, high_limit])
61              plt.plot(x1, y1, color='darkblue', marker='o')
62              # 绘制上限
63              x2 = np.array([idx+window_size, idx+window_size+
    forward_size])
64              y2 = np.array([high_limit, high_limit])
65              plt.plot(x2, y2, color='darkblue', marker='o')
66              # 绘制下限
67              x3 = np.array([idx+window_size, idx+window_size+
    forward_size])
68              y3 = np.array([low_limit, low_limit])
69              plt.plot(x3, y3, color='darkblue', marker='o')
70              # 绘制右侧竖线
71              x4 = np.array([idx+window_size+forward_size, idx+
    window_size+forward_size])
72              y4 = np.array([low_limit, high_limit])
73              plt.plot(x4, y4, color='darkblue', marker='o')
74              # 标注市场状态
75              plt.title('市场状态: {0};'.format(y[idx]))
76              msg = input('please input msg:')
77          plt.show(block=True)
```

Listing 5: 确定当前时刻市场状态

代码原理如下图所示：

图 5: 确定市场行情状态代码原理示意图



代码解读如下所示：

- 第 3 行：y 保存每个时刻的市场状态，quotation 为本类中 get_quotations 方法获取的数据，格式为：[..., [datetime, open, high, low, close, volume], ...]，window_size 为向前看的时刻，forward_size 为向后看多少个时刻作为行情判断窗口，其中除 datetime 外均为 float 类型；
- 第 11 行：循环处理 y 中的每个时刻；
- 第 12 行：取出当前时刻的实际收盘价，我们需要向前看 window_size 个时刻，因此当前时刻是 idx+window_size，因为 quotation 的格式为：[open, high, low, close, volume]，因此序号 3 为收盘价；
- 第 13 行：判断为上涨行情时的最低价格，在当前时刻后 forward_size 个时间点内的收盘价，只要高于该值，就判断为上涨行情；
- 第 14 行：判断为下跌行情时的最高价格，在当前时刻后 forward_size 个时间点内的收盘价，只要低于该值，就判断为下跌行情，因为股市上一旦亏钱，再赚回来就比较难，因此下跌行情的标准为更严格一些；
- 第 15 行：缺省为震荡行情；
- 第 16 行：循环处理未来 forward_size 个时刻；
- 第 17 行：求出未来某个时刻的收盘价；
- 第 18~20 行：当未来时刻收盘价高于上涨行情判断的最低价格时，判断为上涨行情，并退出对未来时刻判断的内循环；
- 第 21~23 行：当未来时刻收盘价低于下跌行情判断的最高价格时，判断为下跌行情，并奶出对未来时刻判断的内循环；
- 第 24 行：确定当前时刻市场状态；

1.5 载入数据集

下面我们载入训练数据集和测试数据集，如下所示：

```
1     def load_stock_dataset(self, stock_symbol, batch_size):
```

```
2         '''
3         获取股票数据集
4         '''
5         X, y, info = OhlcvProcessor.get_ds_raw_data(stock_symbol,
  window_size=10, forward_size=100)
6         print('X: {0};'.format(X.shape))
7         # 生成训练数据集
8         train_persent = 0.9
9         train_test_sep = int(X.shape[0] * train_persent)
10        X_train = X[:]
11        y_train = y[:]
12        info_train = info[:]
13        train_ds = OhlcvDataset(X_train, y_train, info_train)
14        train_loader = DataLoader(
15            train_ds,
16            batch_size = batch_size,
17            num_workers = 0,
18            shuffle = True,
19            drop_last = True
20        )
21        train_iter = train_loader
22        # 生成测试数据集
23        X_test = X[train_test_sep:]
24        y_test = y[train_test_sep:]
25        info_test = info[train_test_sep:]
26        test_ds = OhlcvDataset(X_test, y_test, info_test)
27        test_loader = DataLoader(
28            test_ds,
29            batch_size = batch_size,
30            num_workers = 0,
31            shuffle = True,
32            drop_last = True
33        )
34        test_iter = test_loader
35        return train_iter, test_iter
```

Listing 6: 载入数据集

代码解读如下所示：

- 第 5 行：获取支撑数据集的原始数据，X 为当前时刻加向前看 window_size 个时刻，每个时刻包括进行过标准化一阶对数差分的如下数据：开盘价、最高价、最低价、收盘价、交易量，共 5+10*5=55，即 55 维向量，y 为每个时刻的市场行情状态，info 为原始的市场行情数据，包括：开盘价、最高价、最低价、收盘价；
- 第 8、9 行：将 90% 的数据作为训练集，将 10% 的数据作为测试集；
- 第 10~21 行：载入训练集数据，OhlcvDataset 将在下面详细介绍；

- 第 23~34 行：载入测试集数据；

1.5.1 数据集类

下面我们来看 OhlcvData 类，如下所示：

```
1  class OhlcvDataset(Dataset):
2      def __init__(self, X, y, info):
3          self.name = 'apps.fmts.ds.ohlcv_ds.OhlcvDs'
4          self.X = X
5          self.y = y
6          self.info = info
7
8      def __len__(self):
9          return self.X.shape[0]
10
11     def __getitem__(self, idx):
12         return self.X[idx], self.y[idx], self.info[idx]
13
14     @staticmethod
15     def get_quotation_from_info(info):
16         return {
17             'date': info[0][0],
18             'open': info[1][0],
19             'high': info[2][0],
20             'low': info[3][0],
21             'close': info[4][0],
22             'volume': info[5][0]
23         }
```

Listing 7: 数据集类 OhlcvDataset

这个类相对来说比较简单，唯一需要注意的是 info 属性，其包括的是分钟线的原始数据：日期、开盘、最高、最低、收盘、交易量信息。

1.6 总结

在本章中，我们通过 Akshare 来获取分钟级股票数据集，并将其根据统计特性，形成深度学习可以使用的数据集。在下一章中，我们将介绍用于金融市场的 Transformer 网络。

第 2 章 Transformer 网络

Abstract

在本章中我们将详细讲解用于金融交易的 Transformer 网络。

2 Transformer 网络概述

2.1 自注意力机制

2.2 总结

第 3 章模型训练

Abstract

在本章中我们将详细讲解用于金融交易的 Transformer 网络的模型训练和预测过程。

3 模型训练与预测概述概述

3.1 训练过程

下面我们来看模型的训练过程，训练入口程序如下所示：

```
    def train(self):
        cmd_args = self.parse_args()
        stock_symbol = 'sh600260'
        batch_size = cmd_args.batch_size
        NUM_CLS = 3
        cmd_args.embedding_size = 5
        seq_length = 11
        cmd_args.num_heads = 4
        cmd_args.depth = 6 # 原始值为2
        train_iter, test_iter = self.load_stock_dataset(stock_symbol, batch_size)
        cmd_args.num_heads = 8
        model = FmtsTransformer(emb=cmd_args.embedding_size, heads=cmd_args.num_heads, depth=cmd_args.depth, \
                    seq_length=seq_length, num_tokens=cmd_args.vocab_size, num_classes=NUM_CLS, \
                    max_pool=cmd_args.max_pool)
        model.to(self.device)
        opt = torch.optim.Adam(lr=cmd_args.lr, params=model.parameters())
        sch = torch.optim.lr_scheduler.LambdaLR(opt, lambda i: min(i / (cmd_args.lr_warmup / cmd_args.batch_size), 1.0))
        if cmd_args.continue_train:
            e, model_dict, optimizer_dict = self.load_ckpt(self.ckpt_file)
            model.load_state_dict(model_dict)
            opt.load_state_dict(optimizer_dict)
        # training loop
        cmd_args.num_epochs = 3
        seen = 0
        # early stopping参数
        best_acc = -1
        acc_up = 0.0
        min_acc_up = 0.000001 # 识别为精度提高的最小阈值
        non_acc_up_epochs = 0 # 目前多少个epoch精度未提高
```

```
30          max_no_acc_up_epochs = 50 # 如果精度在这些epoch后还没提高
    则终止训练过程
31          for epoch in range(cmd_args.num_epochs):
32              print(f'\n epoch {epoch}')
33              model.train(True)
34              for batch in tqdm.tqdm(train_iter):
35                  opt.zero_grad()
36                  X, y = self.get_stock_batch_sample(batch,
    batch_size, cmd_args.embedding_size)
37                  y_hat = model(X)
38                  loss = F.nll_loss(y_hat, y)
39                  loss.backward()
40                  # clip gradients
41                  # - If the total gradient vector has a length > 1,
     we clip it back down to 1.
42                  if cmd_args.gradient_clipping > 0.0:
43                      nn.utils.clip_grad_norm_(model.parameters(),
    cmd_args.gradient_clipping)
44                  opt.step()
45                  sch.step()
46                  seen += X.size(0)
47              with torch.no_grad():
48                  model.train(False)
49                  tot, cor= 0.0, 0.0
50                  for batch in tqdm.tqdm(test_iter):
51                      X, y = self.get_stock_batch_sample(batch,
    batch_size, cmd_args.embedding_size)
52                      y_hat = model(X).argmax(dim=1)
53                      tot += float(X.size(0))
54                      cor += float((y == y_hat).sum().item())
55                  acc = cor / tot
56                  # 获取当前最佳测试集精度, 并保存对应的模型
57                  if best_acc < acc:
58                      acc_up = acc - best_acc
59                      if acc_up > min_acc_up:
60                          best_acc = acc
61                          non_acc_up_epochs = 0
62                          print('保存模型参数')
63                          self.save_ckpt(self.ckpt_file, epoch,
    model, opt)
64                  else:
65                      non_acc_up_epochs += 1
66                      if non_acc_up_epochs > max_no_acc_up_epochs:
67                          print('模型已经处于饱合状态, 停止训练过程'
    )
```

```
68                         break
69                 print(f'-- {"test" if cmd_args.final else "
   validation"} accuracy {acc:.3}')
```

Listing 8: 模型训练入口

代码解读如下所示：

- 第 5 行：Transformer 网络输出三种市场状态：上涨、下跌、震荡；
- 第 6 行：我们将每个时刻等价为一个单词，每个时刻可以用开盘、最高、最低、收盘、交易量来表示，因此相当于每个单词是 5 维向量；
- 第 7 行：我们通常考虑当前时刻，同时向前看 10 个时刻，因此每个样本包括 11 个时刻，相当于每个句子有 11 个单词，所以序列长度为 11；
- 第 8 行：共有 6 层结构；
- 第 10 行：共有 8 个自注意头；
- 第 12~15 行：初始化模型，其中 vocab_size=50000 为缺省值，max_pool 代表使用最大池化，并将其放入 GPU 中；
- 第 16 行：使用 adam 优化算法；
- 第 17 行：使用带有 warmup 的学习调整计划算法：...；
- 第 18~21 行：在之前训练的基础上继续训练；
- 第 22 行：训练集训练遍数；
- 第 23 行：共训练了多少个样本；
- 第 26 行：当前所取得的最佳精度；
- 第 27 行：当前精度提高的数值；
- 第 28 行：视为精度有提高的最小阈值，用于避免拢动情况；
- 第 29 行：目前已经有多少个 epoch 精度没有提高，如果达到最大允许的精度没有提高的 epoch 数，则停止训练过程；
- 第 30 行：允许连续多少个 epoch 精度没有显著提高；
- 第 31 行：训练指定个 epoch；
- 第 33 行：将模型置为训练状态；
- 第 34 行：读出训练集的每个迷你批次，循环进行处理；
- 第 35 行：所有参数的梯度设置为 0；
- 第 36 行：获取样本集和标签集，具体实现细节将在后面详细讲解；
- 第 37 行：调用 FmtsTransformer 模型计算网络输出结果；
- 第 38 行：使用负对数似然函数为代价函数，计算代价函数的值；
- 第 39 行：将代价函数值反向传播，求出梯度值；
- 第 42、43 行：当梯度所组成的向量长度大于 1 时，调整梯度值，使长度变为 1；
- 第 44 行：调用优化器对参数进行优化；
- 第 45 行：调用学习率变化方法，按需调整学习率；
- 第 46 行：统计当前学习了多少个样本；

- 第 47、48 行：将其变为验证过程；
- 第 49 行：定义 tot 代表总样本数，cor 为结果正确的样本数；
- 第 50 行：按批次循环处理测试数据集；
- 第 51 行：取出一个批次的样本集和标签集；
- 第 52 行：调用 FmtsTransformer 模型的前向传播过程，求出网络层输出 logits，然后在每个样本结果向量中取出最大值的索引值（取值范围为 0~2），作为预测值；
- 第 53 行：将批次数加入到样本总数中；
- 第 54 行：y=y_hat 的运算结果为一个新的同维度 ndarray，对于第 i 个元素，如果 $y_i == \hat{y}_i$，则该元素为 1，否则为 0。对新数据取 sum 就可以计算出在本批次中正确的样本数量；
- 第 55 行：当处理完所有测试集后，求出在测试集上的精度；
- 第 57~63 行：如果当前精度高于最佳精度，求出精度的提升量，如果提升量高于认为有提升的阈值，则更新最佳精度至当前精度，将连续没有精度提高的 epoch 数置为 0，保存模型至文件；
- 第 64~68 行：如果没有精度提升，则连续没有精度提升的 epoch 数加 1，如果该值大于允许连续没有精度提升 epoch 的最大值，则退出训练过程；

3.2 预测过程

下面来看预测过程，代码如下所示：

```
1  def predict(self):
2      stock_symbol = 'sh600260'
3      cmd_args = self.parse_args()
4      batch_size = cmd_args.batch_size
5      NUM_CLS = 3
6      cmd_args.embedding_size = 5
7      seq_length = 11
8      cmd_args.depth = 6 # 原始值为2
9      cmd_args.num_heads = 8
10     model = FmtsTransformer(emb=cmd_args.embedding_size, heads=
       cmd_args.num_heads, depth=cmd_args.depth, \
11                     seq_length=seq_length, num_tokens=cmd_args.
       vocab_size, num_classes=NUM_CLS, \
12                     max_pool=cmd_args.max_pool)
13     model.to(self.device)
14     e, model_dict, optimizer_dict = self.load_ckpt(self.ckpt_file)
15     model.load_state_dict(model_dict)
16     # 获取测试样本
17     train_iter, test_iter = self.load_stock_dataset(stock_symbol,
       batch_size)
18     batch = iter(test_iter).next()
19     batch_X, batch_y = self.get_stock_batch_sample(batch,
       batch_size, cmd_args.embedding_size)
20     X = batch_X[:1, :, :]
```

```
21     y = batch_y[:1]
22     print('##### X: {0}; y: {1};'.format(X.shape, y.shape))
23     y_hat = model(X).argmax(dim=1)
24     print('y_hat: {0}; y: {1};'.format(y_hat.item(), y.item()))
```

Listing 9: 模型预测入口

代码解读如下所示：

- 第 4 行：批次大小 batch_size 的缺省值为 4；
- 第 5 行：输出层大小，代表市场状态：上涨、下跌、震荡；
- 第 6 行：每个时刻行情数据：开盘、最高、最低、收盘、交易量，可以视为 NLP 中的单词，设定其维度为 5 维；
- 第 7 行：包括当前时刻，再向前取 10 个时刻，共 11 个时刻，相当于 NLP 中规定每个句子的长度为 11；
- 第 8 行：共有 6 层 Encoder 层；
- 第 9 行：共有 8 个自注意力头；
- 第 10~ 12 行：初始化模型，其中 vocab_size=50000 为缺省值，max_pool 代表使用最大池化；
- 第 13 行：将其放入 GPU 中；
- 第 14、15 行：载入预训练好的模型；
- 第 17 行：载入数据集；
- 第 18 行：取出第一个批次；
- 第 19 行：取出这个批次中的样本集和标签集；
- 第 20 行：取出样本集中第 1 个样本；
- 第 21 行：取出标签集中对应的标签；
- 第 23 行：调用 FmtsTransformer 模型求出输出信号 $\hat{y} \in R^3$，通过 argmax 求出最大值元素的索引号作为输出值；

3.3 总结

第 4 章回测平台

Abstract

在本章中我们将详细讲解基于强化学习环境的回测平台。

4 回测平台概述

在本章中，我们基于强化学习环境，创建一个策略的回测平台。

4.1 启动入口

下面我们来看启动的入口函数：

```
def run(self):
    stock_symbol = 'sh600260'
    model = self.build_model()
    env = FmtsEnv(stock_symbol)
    obs = env.reset()
    done = False
    action = env.action_space.sample()
    idx = 0
    while not done:
        X = self.get_model_X(obs)
        quotation_state = self.get_quotation_state(model, X)
        if quotation_state == 0:
            # 买入
            action[0] = 0.5
            action[1] = 1.0
        elif quotation_state == 1:
            # 卖出
            action[0] = 1.5
            action[1] = 1.0
        else:
            # 持有
            action[0] = 2.5
            action[1] = 0.5
        obs, reward, done, info = env.step(action)
        if env.current_step > 200:
            done = True

def build_model(self):
    '''
    强化学习环境Reset中需要调用本函数，初始化模型
    '''
    cmd_args = self.parse_args()
    print('command line args: {0};'.format(cmd_args))
    batch_size = cmd_args.batch_size
    NUM_CLS = 3
```

```
36      cmd_args.embedding_size = 5
37      seq_length = 11
38      cmd_args.depth = 6
39      cmd_args.num_heads = 8
40      model = FmtsTransformer(emb=cmd_args.embedding_size, heads=
        cmd_args.num_heads, depth=cmd_args.depth, \
41                  seq_length=seq_length, num_tokens=cmd_args.
        vocab_size, num_classes=NUM_CLS, \
42                  max_pool=cmd_args.max_pool)
43      model.to(self.device)
44      e, model_dict, optimizer_dict = self.load_ckpt(self.ckpt_file)
45      model.load_state_dict(model_dict)
46      return model
```

Listing 10: 回测平台入口

代码解读如下所示：

- 第 3 行：创建模型，具体代码见第 28 行；
- 第 4 行：创建强化学习环境，具体见下一节；
- 第 5 行：重置强化学习环境，并获得系统初始观察值；
- 第 6 行：将是否结束回测置为否；
- 第 7 行：从行动空间中随机抽样出一个行动，行动是一个二维数组，第 1 个元素代表操作，在 [0,1] 之间为买入，在 (1,2] 之间为卖出，(2,+∞) 时为持有，第二个元素为操作的百分比；
- 第 9 行：如果结束回测标志不为真则一直循环；
- 第 10 行：根据环境的观察求出用于模型的输入信号；
- 第 11 行：通过运行模型，确定当前时刻的市场行情状态：0-上涨、1-下跌、2-震荡；
- 第 12~15 行：当处于上涨行情时，如果还有现金，则执行买入操作，action[0] 取小于 1 的数，action[1] 指定使用当前现金的百分比；
- 第 16~19 行：当处于下跌行情时，如果持有股票，则执行卖出操作，action[0] 取 1 至 2 之间的数，action[1] 指定使用当前持股数的百分比；
- 第 20~23 行：当处于震荡行情时，action[0] 取大于 2 的数，action[1] 的值被忽略；
- 第 24 行：将行动传给环境执行，环境会返回：obs-当前的状态（执行完行动后）、reward-该行动获得的奖励、done-是否结束回测标志、info-其他附加信息；

4.2 强化学习环境

2021.10.01

4.3 总结

5 附录 X