
高级量化交易技术

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

第零篇深度学习

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

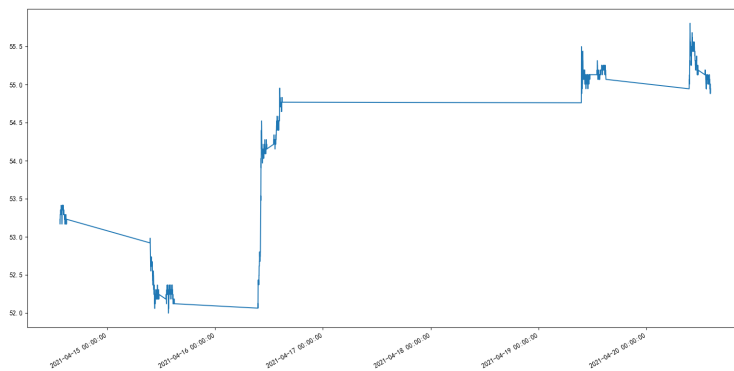
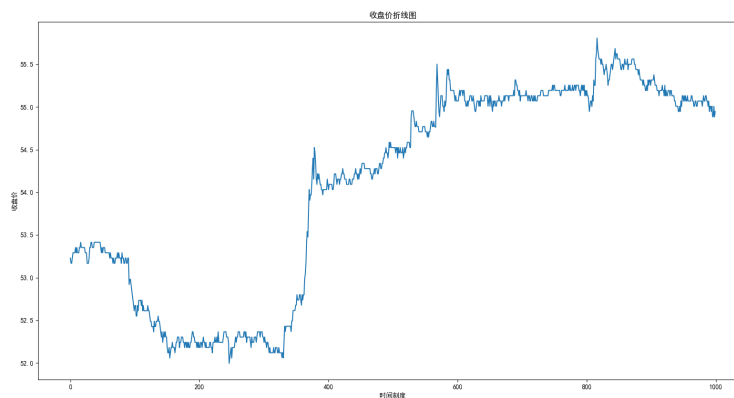


图 2: 以序号为横轴的收盘价折线图



如??所示，图中每天收盘到第二天开盘间没有行情数据，所以图形不太好看规律，而图??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??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 $\log(\text{day2}) - \log(\text{day1})$ ，然
      后求出每列均值和标准差，利用
5       $(x - \mu) / \text{std}$ 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 为底，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， $\text{out}[i] = x[i+1] - x[i]$ ，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行情数据：开盘、最高、最低、收盘、交易量，所以共有 $10 \times 5 + 5 = 55$ 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,
3                          forward_size: int=100) -> Tuple[np.ndarray, np.ndarray, List[
4                          str]]:
5          '''
6          获取数据集所需数据
7          stock_symbol 股票代码
8          window_size 从当前时间点向前看多少个时间点
9          forward_size 向后看多少个时间点确定市场行情是上涨、下跌和
10         震荡
11         返回值
12         X 连续11个时间点的OHLCV的数据，形状为n*55，一阶Log差分
13         形式
14         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
15         行情（按Box方式确定）：0-震荡；1-上升；2-下跌；
16         info 当前时间刻行情的真实值
17         '''
18         print('获取数据集数据')
19         # 获取行情数据
20         quotations = OhlcvProcessor.get_quotations(stock_symbol)
21         # 获取归整化行情数据
22         log_1d_datas = []
23         log_1d_file = './apps/fmts/data/{0}_1m_1d.csv'.format(
24             stock_symbol)
25         with open(log_1d_file, 'r', encoding='utf-8') as fd:
26             for row in fd:
27                 row = row.strip()
28                 arrs = row.split(' ')
29                 item = [arrs[0], arrs[1], arrs[2], arrs[3], arrs
30                         [4]]
31                 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

	A	B	C	D	E	F	G	H	I	J	K	L	M	N	O
1															
2	0	1	2	3	4	5	6	7	8	9	10	11		arr_len	12
3														window_size	3
4														forward_size	5
5														ldd_size	7
6															
7															
8															
9															

- 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

	A	B	C	D	E	F	G	H	I	J	K	L	M	N	O	P	Q
1																	
2	0	1	2	3	4	5	6	7	8	9	10	11	12			arr_len	12
3																	
4																	
5																	
6																window_size	3
7																forward_size	5
8																ldd_size	7
9																	
10																	
11																	
12																	
13																	

- 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。

```

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

```

```

5     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
6     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，
7     否则返回2表示震荡行情，将该值写入y中
8     '''
9     cnt = y.shape[0]
10    q_cnt = quotation.shape[0]
11    for idx in range(0, cnt, 1):
12        curr_price = quotation[idx+window_size][3]
13        high_limit = curr_price * 1.01
14        low_limit = curr_price * 0.995
15        market_regime = 2
16        for pos in range(idx+window_size+1, idx+window_size+1+
forward_size, 1):
17            future_price = quotation[pos][3]
18            if future_price >= high_limit:
19                market_regime = 0
20            elif future_price <= low_limit:
21                market_regime = 1
22            y[idx] = market_regime
23
24
25 # 测试程序：utcs/apps/fmts/ds/t_ohlc_v_processor.py
26 def test_get_market_state002(self):
27     # 开始下标：window_size，结束下标：cnt-forward_size-1（包
    含）
28     random.seed(1.0)
29     y = np.zeros((10,), dtype=np.int64)
30     quotation_raw = []
31     cnt_y = 10
32     curr_price = 5.36
33     high_delta = 1.01
34     low_delta = 0.995
35     window_size = 3
36     forward_size = 4
37     cnt = cnt_y+window_size+forward_size
38     for idx in range(cnt):
39         close_price = random.uniform(curr_price*low_delta,
curr_price*high_delta)
40         item = [0.01, 0.02, 0.03, close_price, 0.04]
41         quotation_raw.append(item)
42         print('quotation_raw: {0}: {1};'.format(len(quotation_raw)
, quotation_raw))
43     quotation = np.array(quotation_raw)

```

```

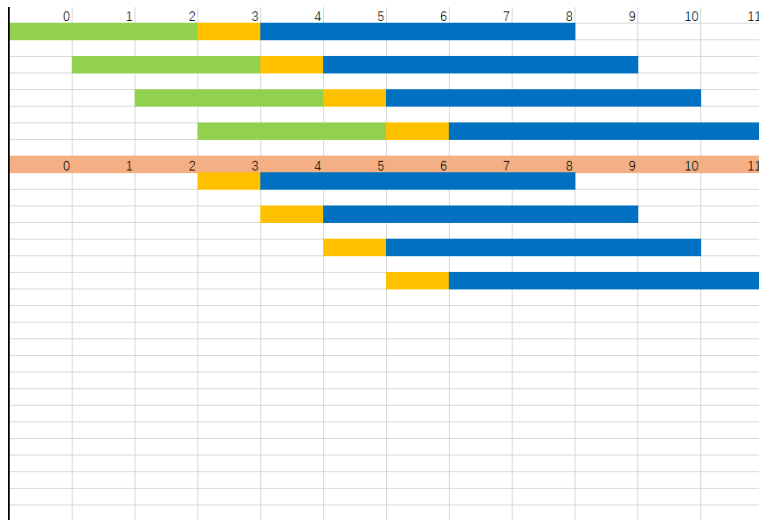
44     OhlcvProcessor.get_market_state(y, quotation, window_size,
forward_size)
45     print('y: {0}; {1};'.format(y.shape, y))
46     plt.ion()
47     fig, axes = plt.subplots(1, 1, figsize=(8, 4))
48     for idx in range(cnt_y):
49         plt.draw()
50         plt.pause(0.1)
51         plt.cla()
52         # 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，并退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。

