高级量化交易技术

闫涛
科技有限公司
北京
{yt7589}@qq.com

第 1 章　时间序列基本特性

Abstract

在本章中我们将讨论时间序列的基本特性，包括自相关性和平稳性。

1　时间序列基本特性

时间序列的自相关性是指时间序列过去与未来存在某种关系，是我们时间序列预测的基础。主要用自协方差函数（Autocovariance Function, AF）、自相关系数函数（Autocorrelation Coefficent Function, ACF）和偏自相关系数函数（Partial Autocorrelation Coefficient Function, PACF）来描述。

1.1　启动过程

首先是 FmeEngine 的构造函数，如下所示：

```
class FmeEngine(object):
    def __init__(self):
        self.name = 'FmeEngine'
        self.env = None
        self.agent = FmeXgbAgent()
        self.test_size = 1000
```

Listing 1: FmeEngine 的构造函数

这里面重要的代码是初始化一个 FmeXgbAgent 类的实例，该实例利用 XGBoost 算法来选择策略。下面来看 FmeXgbAgent 类的构造函数：

```
class FmeXgbAgent(object):
    def __init__(self):
        self.name = 'FmeXgbAgent'
        self.model_file = './work/btc_drl.xgb'
        self.max_min_file = './work/btc_max_min.csv'
        self.bst = None
        self.fme_dataset = FmeDataset()
        self.X, self.y = self.fme_dataset.load_bitcoin_dataset()
        self.model = self.train_baby_agent()
        self.df = None
        self.fme_env = None
        self.max_min_file = './work/btc_max_min.csv'
        self.dataset_size = 10
        self.cached_quotation = np.loadtxt(self.max_min_file, delimiter=
    ',')
```

Listing 2: FmeXgbAgent 的构造函数

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

这段代码中，最重要的是调用 train_baby_agent 方法，其用前 1000 个时间点，训练一个初始化的 XGBoost 模型，该模型可以选择适合的动作。这部分代码在之前已经讲述过，这里就不再复述了。
系统程序入口在 FmeEngine.startup 方法中，如下所示：

```
1  def startup(self):
2      self.env = self.build_raw_env()
3      self.agent.df = self.fme_env.df
4      self.agent.fme_env = self.fme_env
5      obs = self.env.reset()
6      for i in range(self.slice_point):
7          action = self.agent.choose_action(i+self.fme_env.
       lookback_window_size, obs)
8          obs, rewards, done, info = self.env.step([action])
9          if done:
10             break
11         self.env.render(mode="human", title="BTC")
12         # 重新训练模型
13         self.agent.train_drl_agent(info[0]['weight'])
14     print('回测结束 ^_^')
```

Listing 3: 程序入口点

- 第 2 行：创建深度强化学习环境，以比特币分钟级数据为环境，划分训练样本集和测试样本集；
- 第 3、4 行：在 Agent 中保存数据集内容和环境，主要是便于进行训练；
- 第 5 行：重置环境；
- 第 7~13 行：每个时间点之前的几个时间点（5 个）的数据组成一个样本，也是环境的一个状态，Agent 将根据这个状态决定要采取的行动：买入、持有、卖出，在操作之后，根据新的净资产与原来净资产的大小，决定奖励信号，奖励信号为新净资产与老净资产的比值。以上为深度强化学习的一步，循环执行此过程，直到运行完所有时间点。在每个时间点，第 11 行以图形方式绘制交易情况和净资产变化情况，我们将奖励信号作为对应样本的权重，重新训练我们策略网络，这里我们用的是 XGBoost。

深度强化学习中环境是一个非常重要的因素，我们来具体看一下环境的创建过程：

```
1  def build_raw_env(self):
2      ''' 创建原始比特币行情文件生成的env，主要用于深度强化学习试验 '''
3      self.df = pd.read_csv('./data/bitstamp.csv')
4      self.df = self.df.drop(range(FmeDataset.DATASET_SIZE))
5      self.df = self.df.dropna().reset_index()
6      self.df = self.df.sort_values('Timestamp')
7      self.agent.df = self.df
8      self.slice_point = int(len(self.df) - self.test_size)
9      self.train_df = self.df[:self.slice_point]
10     self.test_df = self.df[self.slice_point:]
11     self.fme_env = FmeEnv(self.train_df, serial=True)
12     self.agent.fme_env = self.fme_env
13     return DummyVecEnv(
14         [lambda: self.fme_env])
```

Listing 4: 深度强化学习环境创建

- 第 3 行：从 CSV 文件中读出 DataFrame 格式的比特币分钟级数据；
- 第 4 行：忽略前 1000 个时间点，这些时间点将用于训练一个初始的 Policy Gradient 模型，加快学习进程；
- 第 5、6 行：去掉为空的行并重建索引，同时按时间进行排序；
- 第 8 行：slice_point 是训练样本集和测试样本集的分隔点；
- 第 11 行：是本段程序的重点，其初始化了一个深度强化学习环境；

下面我们来看深度强化学习环境的定义，如下所示：

```
MAX_TRADING_SESSION = 100000

class FmeEnv(gym.Env):
    def __init__(self, df, lookback_window_size=50,
        commission=0.00075, initial_balance=10000,
        serial=False
    ):
        self.name = 'FmeEnv'
        print('Finacial Market Env is starting up...')
        random.seed(100)
        self.buy_rate = 1.0 # 20%机会购买
        self.sell_rate = 1.0 # 15%机会卖
        self.df = df.dropna().reset_index()
        print(self.df.head(10))
        self.lookback_window_size = lookback_window_size
        self.initial_balance = initial_balance
        self.commission = commission
        self.serial = serial  # Actions of the format Buy 1/10, Sell
    3/10, Hold, etc.
        # Observes the OHCLV values, net worth, and trade history
        self.scaler = preprocessing.MinMaxScaler()
        self.viewer = None
        self.action_space = spaces.MultiDiscrete([3, 10])
        self.observation_space = spaces.Box(low=0, high=1, shape=(10,
                    lookback_window_size + 1), dtype=np.float16)
```

Listing 5: 深度强化学习环境类构造函数

由上面的代码可以看出，该类继承自 gym.Env，除了构造函数外，还有 reset 和 step 是需要重载的方法，我们将在后面的流程中进行讲解。
如表3系统会首先调用 FmeEngine.reset 方法，初始化环境，并返回环境的初始状态。下面我们来看 FmeEnv 类的 reset 方法：

```
def reset(self):
    self.balance = self.initial_balance
    self.net_worth = self.initial_balance
    self.btc_held = 0
    self._reset_session()
    self.account_history = np.repeat([
        [self.balance],
        [0],
        [0],
        [0],
        [0]
    ], self.lookback_window_size + 1, axis=1)
    self.trades = []
    return self._next_observation()

def _reset_session(self):
    self.current_step = 0
    if self.serial:
        self.steps_left = len(self.df) - self.lookback_window_size - 1
        self.frame_start = self.lookback_window_size
    else:
        self.steps_left = np.random.randint(1, MAX_TRADING_SESSION)
        self.frame_start = np.random.randint(
            self.lookback_window_size, len(self.df) - self.steps_left)
    self.active_df = self.df[self.frame_start - self.lookback_window_size
    :
                             self.frame_start + self.steps_left]

def _next_observation(self):
    end = self.current_step + self.lookback_window_size + 1
    scaled_df = self.active_df.values[:end].astype(np.float64)
```

```
31      scaled_df = self.scaler.fit_transform(scaled_df)
32      scaled_df = pd.DataFrame(scaled_df, columns=self.df.columns)
33      obs = np.array([
34          scaled_df['Open'].values[self.current_step:end],
35          scaled_df['High'].values[self.current_step:end],
36          scaled_df['Low'].values[self.current_step:end],
37          scaled_df['Close'].values[self.current_step:end],
38          scaled_df['Volume_(BTC)'].values[self.current_step:end],
39      ])
40      scaled_history = self.scaler.fit_transform(self.account_history.
        astype(np.float64))
41      obs = np.append(
42          obs, scaled_history[:, -(self.lookback_window_size + 1):], axis
        =0)
43      return obs
```

Listing 6: 深度强化学习环境类重置方法

- 第 2~4 行：重置资金余额、资产净值和比特币持有量；
- 第 5 行：重置 session，这里的 session 与监督学习中的 epoch 实际上是同一概念，就是遍历所有时间点；
-
  - 第 17 行：current_step 是当前时间点的指针，指向当前时间点，初始值指向第 1 个时间点；
  - 第 18~20 行：self.serial 代表是否从第一个时间点开始，如果是否的话，则从一个随机的时间点开始；这里是从第 1 个时间点开始的情况，lookback_window_size 表示从当前时间点开始，向前取几个时间点形成一个样本数据用于进行操作选择，因此开始时间点应该从第 lookback_window_size 开始，结束时间点应该到最后一个时间点前的 lookback_window_size 个时间点结束；
  - 第 21~24 行：处理从随机的时间点开始的情况；
  - 第 25、26 行：定义 active_df 是活跃的时间点记录；
- 第 6~12 行：设置账户的操作历史，在每个时间点，账户历史信息包括：余额、买入量、买入金额、卖出量、卖出金额；
- 第 13 行：清空交易历史；
- 第 14 行：向 Agent 返回当前状态；
-
  - 第 29 行：求出当前状态的结束时间点 end；
  - 第 30~32 行：对数据进行归一化，设训练样本集最大值为 v_{max}，最小值为 v_{min}，公式为 $\hat{v} = \frac{v - v_{min}}{v_{max} - v_{min}}$，归一化为 $[0, 1]$ 之间的数，这种方法的缺点是对异常点数据敏感，但是比特币交易数据很少会出现异常数据；
  - 第 33~39 行：将开盘价、最高价、最低价、收盘价、成交量所有时间点的数据分别作为一行（与数据集要求每一行代表一个样本相反）；
  - ??????? 作用以后补全；

在一个 Session 中，对于每个时间点，我们首先通过 FmeXgbAgent 来选择合适的操作，代码如下所示：

```
def choose_action(self, idx, obs):
    '''  '''
    commission = self.fme_env.commission
    frame_size = self.fme_dataset.frame_size
    recs = self.df.iloc[idx-frame_size+1:idx+1]
    datas = np.array(recs)
    ds = datas[:, 3:8]
    print('ds.shape:{0}; frame_size={1}; idx={2}'.format(ds.shape,
    frame_size, idx))
    ds = np.reshape(ds, (frame_size*5, ))
    date_quotation = ds[20:25]
    if self.fme_env.btc_held <= 0.00000001:
        x = np.append(ds, [0.0])
    else:
```

```
        x = np.append(ds, [1.0])
    self.add_quotation_tick(self.cached_quotation, [x[20], x[21], x[22], x[23], x[24]])
    ds_max = np.amax(self.cached_quotation, axis=0)
    ds_min = np.amin(self.cached_quotation, axis=0)
    self.normalize_ds(x, ds_max, ds_min)
    print('x:{0:04f}, {1:04f}, {2:04f}, {3:04f}, {4:04f}, {5:04f}, '
          '{6:04f}, {7:04f}, {8:04f}, {9:04f}, {10:04f}, {11:04f},'
          '{12:04f}, {13:04f}, {14:04f}, {15:04f}, {16:04f}, {17:04f}, {18:04f},'
          '{19:04f}, {20:04f}, {21:04f}, {22:04f}, {23:04f}, {24:04f}, {25:04f},'.format(
              x[0], x[1], x[2], x[3], x[4],
              x[5], x[6], x[7], x[8], x[9],
              x[10], x[11], x[12], x[13], x[14],
              x[15], x[16], x[17], x[18], x[19],
              x[20], x[21], x[22], x[23], x[24], x[25]
          ))
    xg = xgb.DMatrix([x], label=x)
    pred = self.model.predict(xg)
    action_type = np.argmax(pred)
    print('pred:{0}; [{1:02f}, {2:02f}, {3:02f}]=>{4}'.format(
        pred.shape, pred[0][0], pred[0][1], pred[0][2],
        np.argmax(pred[0]))
    )
    if 0 == action_type:
        action = np.array([0, 10])
    elif 1 == action_type:
        action = np.array([1, 10])
    else:
        action = np.array([2, 10])
    self.x = x
    self.action = action_type
    return action
```

Listing 7: FmeXgbAgent 选择操作

- 第 3 行：指定手续费费率；
- 第 4 行：frame_size 表示从当前时间点向前看几个时间点，由这几个时间点的行情数据来选择最有利的操作；
- 第 5 行：idx 参数代表当前时间点，一个样本的数据由当前时间点之前 frame_size 个时间点的数据再加上当前时间点的数据，组成一个样本；
-

产品经理常犯的错误：1 自我感觉良好 2 知其然，不知其所以然 3 老板的话是圣旨 4 需求变更频繁 5 不善于沟通 6 不重视需求文档和原型 7 为了做产品而做产品，没有反思和复盘 8 项目管理混乱 9 不做计划和总结

2 汇总

```
t000004
f000022
c000089
e000122
```

Listing 8: 编号

参考文献：tushare and 匠芯量化 [2019]

2.0.1 Transformer 策略

生成训练样本集，采用 MLP 模型进行训练:
取出 lookback_window_size+1 条数据，运行 choose_action 算法，求出其应该是 [0,0,0]，分别对应于买入、卖出、持有，然后采用 TensorFlow 2.0 MLP 算法作为交易策略并进行训练，将 choose_action 替换为 MLP 的 predict 方法，然后运行测试样本集，得到最终的回测结果。
必须包括当前的余额值，所以其有两个记录，分别对应满仓和空仓时的操作。
将策略算法换为 XGBoost。
将策略算法换为 Transformer。

3 XLNet 模型

```
\text{数学公式1}https://meta.wikimedia.org/wiki/Help:Displaying_a_formula
https://arxiv.org/pdf/1805.09692.pdf
https://www.biorxiv.org/content/biorxiv/early/2018/07/03/360537.full.pdf
https://github.com/tensorflow/tensor2tensor/blob/master/tensor2tensor/
    models/evolved_transformer.py#L66
```

Listing 9: 参考链接

References

tushare and 匠芯量化. 机器量化分析（二）——模型评估与仓位管理. *tushare.org*, 平台介绍, 2019. URL https://tushare.pro/document/1?doc_id=68.