# 量化交易平台开发手册

**闫涛**
阿尔山金融科技有限公司
北京
{yt7589}@qq.com

## Abstract

基于 tushare.org 开放数据集，构建本地量化交易开发平台。

## 1 概述

### 1.1 环境搭建

创建开源项目，项目网址：https://github.com/yt7589/aqp ，本地环境为：d:/awork/aftdc/incubate/aqp，虚拟环境激活：d:/aadesk/dev/python/quant/Script/activate ，这个是通过 python -m venv quant 来创建的虚拟环境。

### 1.2 整体架构

app_main.py：程序主入口；
app_registry.py：管理程序中所有全局性配置和变量；
controller 目录：所有业务逻辑实现类；
model 目录：所有数据库操作类；

### 1.3 数据服务商

我们采用的是 tushare.pro 提供的数据服务：https://tushare.pro 。

### 1.4 Python 单元测试

我们采用类似测试驱动开发方式，主要功能都要有单元测试，因此需要使用单元测试框架。我们采用 unittest 单元测试框架，下面我们以一个具体的例子为例，来说明怎样使用单元测试框架。我们来看怎样从 t_account_io 表的某个账户中转出现金的 10%。我们首先建立 test 目录，存放所有单元测试用例。
首先建立测试用例目录，其名称为 tdd，注意：不要叫 test 目录，这个目录会有问题。
我们建立账户流水控制器类 controller/c_account_io.py，如下所示：

```
class CAccountIo(object):
def __init__(self):
    self.name = 'CAccountIo'

@staticmethod
def withdraw(account_id, amount):
    '''
    从指定账户取出指定金额，增加资金转出金额记录，并更新用户现金资产
    '''
    print('CAccountIo.withdraw')

```

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

```
#股票预测:
#https://medium.com/m/global-identity?redirectUrl=https%3A%2F%2
    Ftowardsdatascience.com%2Faifortrading-2edd6fac689d
```

Listing 1: 账户流水类

在测试目录下创建对应的测试类 tdd/controller/t_c_account_io.py，如下所示：

```
import unittest
from controller.c_account_io import CAccountIo

class TCAccountIo(unittest.TestCase):
    def test_withdraw(self):
        account_id = 1
        amount = 100
        CAccountIo.withdraw(account_id, amount)
        self.assertEqual('abc', 'abc')

if '__main__' == __name__:
    unittest.main()
```

Listing 2: 账户流水测试类

测试入口类 app_test.py，如下所示：

```
from tdd.controller.t_c_account_io import TCAccountIo
```

Listing 3: 单元测试入口类

运行单元测试用例命令为：

```
python -m unittest -v app_test.py
python -m unittest -v tdd.model.t_m_account_io.TMAccountIo.test_withdraw
```

Listing 4: 测试程序命令行执行

第一种方式是在一个文件中统一进行处理，第二种方式直接指定预测试的类甚至欲测试的方法。运行结果如下所示：

Figure 1: 单元测试运行结果

```
(quant) arxanfintechdeMacBook-Pro:aqp arxanfintech$ python -m unittest -v app_test.py
test_withdraw (tdd.controller.t_c_account_io.TCAccountIo) ... CAccountIo.withdraw
ok

----------------------------------------------------------------------
Ran 1 test in 0.000s

OK
(quant) arxanfintechdeMacBook-Pro:aqp arxanfintech$
```

### 1.5 Flask 入门

Flask 框架目前已经超越 Django，成为 Python 语言 Web 开发中最流行的框架。

#### 1.5.1 入门程序

使用 flask 进行 Web 开发非常简单，因为实际应用中 python 只用于提供 API 接口，因此我们以 API 服务器为例：

```
import json
from flask import Flask

app = Flask(__name__)

```

```
@app.route('/')
def hello_world():
    resp = {'code': 1}
    resp['result'] = {}
    resp['result']['userId'] = 101
    return json.dumps(resp)

app.run(port=5080, debug=True)
```

Listing 5: Hello World 级 Flask 程序

上述程序中，监听在 5080 端口，返回值为 JSON 格式。

### 1.5.2 路径中的参数

在很多情况下，我们需要将路径中的部分作为参数，具体处理方式如下所示：

```
@app.route('/user/<int:user_id>/<string:user_name>/<email>', methods=['
    GET'])
def process_path_params(user_id, user_name, email):
    resp = {'code': 1}
    resp['result'] = {}
    resp['result']['userId'] = user_id
    resp['result']['userName'] = user_name
    resp['result']['email'] = email
    return json.dumps(resp)
```

Listing 6: 处理 path 中的参数

参数可以带类型说明也可以不带，同时分号两边不能有空格。methods 可以为 GET 或 POST。

### 1.5.3 处理 QueryString

获取 URL 中的参数可以采用下面的方法：

```
@app.route('/get_url_params/t1', methods=['GET'])
def get_url_params():
    resp = {'code': 1}
    resp['result'] = {}
    resp['result']['url'] = url_for('get_url_params')
    resp['result']['userId'] = request.args['userId']
    # args类型为ImmutableMultiDict
    params = request.args.to_dict()
    resp['result']['params'] = params
    return json.dumps(resp)
```

Listing 7: 获取 URL 中的参数

如上所示，url_for 可以打印出完整的 URL 路径，参数为路径中第一个分量。request.args 为不可变字典，也可以将其转变为普通字典。

### 1.5.4 处理 form

当请求为 POST 时，参数是通过 form 来传递的，这时需要解析 form 中的参数：

```
@app.route('/get_post_params/t2', methods=['POST'])
def get_post_params():
    resp = {'code': 1}
    resp['result'] = {}
    resp['result']['userName'] = request.form['userName']
    params = request.form.to_dict()
    return json.dumps(json)
```

Listing 8: 获取 URL 中的参数

### 1.5.5 session 处理

略。

### 1.5.6 cookie 处理

略。

### 1.5.7 文件上传处理

略。

### 1.5.8 数据库连接池

首先需要通过 pip install DBUtils 模块，该模块提供数据库连接池支持。
我们定义数据库操作类 model/m_mysql.py，在其中定义数据库增、删、改、查操作，如下所示：

```
import pymysql
from DBUtils.PooledDB import PooledDB

RD_DBP = PooledDB(
        creator=pymysql,  # 使用链接数据库的模块
        maxconnections=6,  # 连接池允许的最大连接数，0和None表示不限制连接数
        mincached=2,  # 初始化时，链接池中至少创建的空闲的链接，0表示不创建
        maxcached=5,  # 链接池中最多闲置的链接，0和None不限制
        maxshared=3,    #这个参数没多大用，    最大可以被大家共享的链接
        # 链接池中最多共享的链接数量，0和None表示全部共享。PS: 无用，因为pymysql和MySQLdb等模块的 threadsafety都为1，所有值无论设置为多少，_maxcached永远为0，所以永远是所有链接都共享。
        blocking=True,  # 连接池中如果没有可用连接后，是否阻塞等待。True，等待；False，不等待然后报错
        maxusage=None,  # 一个链接最多被重复使用的次数，None表示无限制
        setsession=[],  # 开始会话前执行的命令列表。如: ["set datestyle to ...", "set time zone ..."]
        ping=0,
        # ping MySQL服务端，检查是否服务可用。# 如: 0 = None = never, 1 = default = whenever it is requested, 2 = when a cursor is created, 4 = when a query is executed, 7 = always
        host='127.0.0.1',
        port=3306,
        user='quant',
        password='Quant2019',
        database='QuantDb',#链接的数据库的名字
        charset='utf8'
    )

WT_DBP = PooledDB(
    ......
    )

def get_rdb_connection():
    return RD_DBP.connection()

def get_wdb_connection():
    return WT_DBP.connection()

def close_db_connection(conn):
    conn.close()

def query(sql, params):
```

```
    conn = get_rdb_connection()
    result = query_t(conn, sql, params)
    close_db_connection(conn)
    return result

def query_t(conn, sql, params):
    cursor = conn.cursor()
    cursor.execute(sql, params)
    rowcount = cursor.rowcount
    rows = cursor.fetchall()
    cursor.close()
    return (rowcount, rows)

def insert(sql, params):
    conn = get_wdb_connection()
    result = insert_t(conn, sql, params)
    close_db_connection(conn)
    return result

def insert_t(conn, sql, params):
    cursor = conn.cursor()
    affected_rows = cursor.execute(sql, params)
    conn.commit()
    cursor.close()
    pk = cursor.lastrowid
    return (pk, affected_rows)

def delete(sql, params):
    conn = get_wdb_connection()
    result = delete_t(conn, sql, params)
    close_db_connection(conn)
    return result

def delete_t(conn, sql, params):
    cursor = conn.cursor()
    affected_rows = cursor.execute(sql, params)
    conn.commit()
    cursor.close()
    return (0, affected_rows)

def update(sql, params):
    conn = get_wdb_connection()
    result = update_t(conn, sql, params)
    close_db_connection(conn)
    return result

def update_t(conn, sql, params):
    cursor = conn.cursor()
    affected_rows = cursor.execute(sql, params)
    conn.commit()
    cursor.close()
    return (0, affected_rows)
```

Listing 9: 获取 URL 中的参数

如**??**所示，定义了读、写数据库连接池，后缀为 _t 的函数用于数据库事务中使用。

## 2 数据处理

数据处理包括从 tushare.org 网站获取数据，将其转化为量化平台所需的数据格式。

### 2.1 获取沪深市场所有挂牌股票

获取在沪深两市挂牌的所有股票的基本信息。

### 2.1.1 接口定义

获取股票基本信息接口为 stock_basic，其参数为：

Table 1: stock_basic 接口输入参数说明

| 名称 | 类型 | 必选 | 描述 |
|---|---|---|---|
| is_hs | str | N | 是否沪深港通标的，N 否 H 沪股通 S 深股通 |
| list_status | str | N | 上市状态：L 上市 D 退市 P 暂停上市 |
| exchange | str | N | 交易所：SSE 上交所 SZSE 深交所 HKEX 港交所 |

返回值为：

Table 2: stock_basic 接口返回结果说明

| 名称 | 类型 | 描述 |
|---|---|---|
| ts_code | str | TS 代码 |
| symbol | str | 股票代码 |
| name | str | 股票名称 |
| area | str | 所在地域 |
| industry | str | 所属行业 |
| fullname | str | 股票全称 |
| enname | str | 英文全称 |
| market | str | 市场类型（主板/中小板/创业板） |
| exchange | str | 交易所代码 |
| curr_type | str | 交易货币 |
| list_status | str | 上市状态：L 上市 D 退市 P 暂停上市 |
| list_date | str | 上市日期 |
| delist_date | str | 退市日期 |
| is_hs | str | 是否沪深港通标的，N 否 H 沪股通 S 深股通 |

调用格式为：

```
import tushare as ts
pro = ts.pro_api()
data = pro.stock_basic(exchange='', list_status='L',
                       fields='ts_code,symbol,name,area,industry,list_date')
data = pro.query('stock_basic', exchange='', list_status='L',
                 fields='ts_code,symbol,name,area,industry,list_date')
```

Listing 10: 获取股票基本信息

见**??**所示结果示例：

```
     ts_code  symbol     name  area industry  list_date
0  000001.SZ  000001   平安银行  深圳     银行   19910403
1  000002.SZ  000002    万科A  深圳   全国地产   19910129
2  000004.SZ  000004   国农科技  深圳   生物制药   19910114
3  000005.SZ  000005   世纪星源  深圳   房产服务   19901210
4  000006.SZ  000006   深振业A  深圳   区域地产   19920427
5  000007.SZ  000007    全新好  深圳   酒店餐饮   19920413
```

Listing 11: 获取股票基本信息结果示例

### 2.1.2 区域信息

如代码**??**所示，地区是以字符串形式返回的。我们可能需要按地区来统计股票表现，因此需要将地区统计出来，放到单独的一个表中进行管理。

**数据库设计** 数据库结构表结构如下所示：

```sql
create table t_area(
    area_id int primary key auto_increment,
    area_name varchar(200)
);
```

Listing 12: 地区表数据结构

**信息处理** 当我们读到返回结果的一行时，我们取出地区信息，然后查询 t_area 表中是否包含该地区，如果包含则返回对应的 area_id，否则将该地区添加到 t_area 表中，并返回其 area_id。

#### 2.1.3 行业信息

**数据库设计**

**信息处理**

#### 2.1.4 股票信息

**接口定义**

**获取并处理数据**

### 2.2 获取日线行情数据

## 3 量化模型

### 3.1 时间序列分析

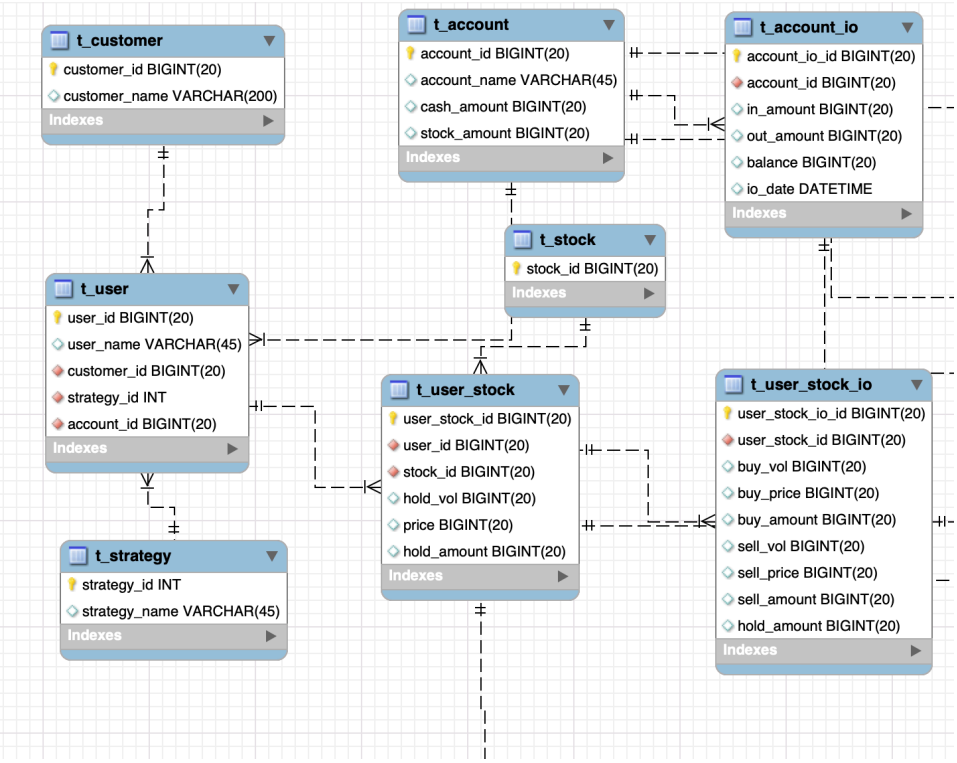
## 4 回测系统

### 4.1 数据库系统设计

客户是自然人，用 t_customer 表示。客户加上账户再加上量化策略，形成我们系统的用户，用 t_user 来表示。账户中具有现金资产和股票资产。账户具有资金的入和出，有股票的买入和卖出。用户持有一些股票，可以对股票进行买卖。用户可以买入和卖出指定数量股票，与账户资金变动相关联股票参数表：买入费率、印花税；卖出费率、印花税等，计入交易成本中。

### 4.1.1 ER 图

Figure 2: 数据库表 ER 图




### 4.1.2 客户表

表示自然人客户，结构如下所示：

Table 3: 客户表 (t_custome)

| 字段 | 名称 | 类型 | 描述 |
|---|---|---|---|
| customer_id | 客户编号 | bigint | 主键且自增长 |
| customer_name | 客户姓名 | varchar | 真实姓名 |

### 4.1.3 策略表

主要包括股票交易策略：包括 SVM、XGBoost、LSTM、ARIMA、GARCH 等，客户可以订购某个策略而成为我们的用户。

Table 4: 策略表 (t_strategy)

| 字段 | 名称 | 类型 | 描述 |
|---|---|---|---|
| strategy_id | 策略编号 | bigint | 主键且自增长 |
| strategy_name | 策略名称 | varchar | 真实姓名 |

### 4.1.4 账户表

客户拥有账户，每个客户对应的用户具有唯一的账户。账户中有现金和股票资产。

Table 5: 账户表 (t_account)

| 字段 | 名称 | 类型 | 描述 |
|---|---|---|---|
| account_id | 账户编号 | bigint | 主键且自增长 |
| account_name | 账户名称 | varchar | 易于记忆的名称 |
| cash_amount | 现金资产 | bigint | 以分为单位 |
| stock_amount | 股票资产 | bigint | 以分为单位 |

其有对应的历史表 t_account_hist，除上述字段外，还加上 hist_date 字段，用于记录每一天的资产。

#### 4.1.5 账户流水表

显示用户现金账户资金进出情况，与股票流水表主键相同，用于表示股票买卖过程中资金的变化情况。

Table 6: 账户流水表 (t_account_io)

| 字段 | 名称 | 类型 | 描述 |
|---|---|---|---|
| account_io_id | 账户流水编号 | bigint | 主键且自增长 |
| account_id | 账户编号 | bigint | t_account 表外键 |
| in_amount | 转入金额 | bigint | 以分为单位 |
| out_amount | 转出金额 | bigint | 以分为单位 |
| balance | 余额 | bigint | 以分为单位 |
| io_date | 发生日期 | datetime | |

#### 4.1.6 用户股票表

用于表示用户当前拥有的股票。

Table 7: 用户股票表 (t_user_stock)

| 字段 | 名称 | 类型 | 描述 |
|---|---|---|---|
| user_stock_id | 账户流水编号 | bigint | 主键且自增长 |
| user_id | 用户编号 | bigint | t_account 表外键 |
| stock_id | 股票编号 | bigint | 以分为单位 |
| hold_vol | 持有量 | bigint | |
| price | 价格 | bigint | |
| hold_amount | 转出金额 | bigint | 以分为单位 |
| balance | 余额 | bigint | 以分为单位 |
| io_date | 发生日期 | datetime | |

有对应的历史表，在上述字段基础上添加 hist_date 字段，用于记录每一天的资产。

#### 4.1.7 用户股票流水表

记录用户股票买卖情况，主键与 t_account_id 相同，股票买卖是账户流水的一个子类。

Table 8: 用户股票表 (t_user_stock_io)

| 字段 | 名称 | 类型 | 描述 |
|---|---|---|---|
| user_stock_io_id | 账户流水编号 | bigint | 主键且自增长 |
| user_stock_id | 用户编号 | bigint | t_account 表外键 |
| buy_vol | 买入量 | bigint | 手数 |
| buy_price | 买入价格 | bigint | |
| buy_cost | 买入成本 | bigint | |
| buy_amount | 买入金额 | bigint | 包括印花税、手续费、所得税等成本 |
| sell_vol | 卖出量 | bigint | 以分为单位 |
| sell_price | 卖出价格 | bigint | 以分为单位 |
| sell_cost | 卖出成本 | bigint | 包括印花税、手续费、所得税等成本 |
| sell_amount | 卖出金额 | bigint | |
| hold_vol | 持仓量 | bigint | |
| hold_price | 收盘价 | bigint | |
| hold_amount | 市值 | bigint | |

对应有历史表，除上述字段外，还有 hist_date 字段。

### 4.2 准备实验数据

创建一个新客户：

```
INSERT INTO `QuantDb`.`t_customer` (`customer_id`, `customer_name`)
VALUES (1, '王俊锋');
```

Listing 13: 创建新客户

添加一个新策略：

```
INSERT INTO `QuantDb`.`t_customer` (`customer_id`, `customer_name`)
VALUES (1, '王俊锋');
```

Listing 14: 创建新策略

创建账户并添加初始现金：

```
INSERT INTO `QuantDb`.`t_account_io` (`account_io_id`, `account_id`,
`io_date`, `in_amount`, `out_amount`, `balance`) VALUES (1,
100000000, '2019-02-26', 100000000, 0, 1);
INSERT INTO `QuantDb`.`t_account` (`account_id`, `account_name`, `
cash_amount`, `stock_amount`) VALUES (1, '股票投资账户', 100000000,
0);
```

Listing 15: 创建账户并入资

有了上述数据之后，我们就可以从 2018-01-01 开始，利用 SVM 模型，以收盘价为成交价，进行回测研究了。
具体股票买卖逻辑为：选定某一支股票，利用 50% 现金购买该股票；每日利用 SVM 进行预测，如果涨则拿剩余现金的 10% 买入，如果跌卖出持有股份的 10% 买入：先将钱从账户现金中转出，减少账户现金资产；在股票流水中增加进项，然后增加股票持有量，将钱加入的账户的股票资产下卖出：先减少股票持有量，股票流水中转出资金，账户股票资金减少，账户增加资金进入，将现金增加到现金资产上交易的结算价格按每日收盘价计算绘制每日用户总资产（现金资产加股票资产）

### 4.3 回测平台开发

#### 4.3.1 买入股票

获取前一个交易日的收盘价

```
class StockBacktest(object):
    def __init__(self):
        self.name = 'StockBacktest'

    def buy_stock(self, ts_code, curr_date):
        '''
        在指定日期买入指定股票
        @param ts_code: 股票编码
        @param curr_date: 指定日期
        @version v0.0.1 闫涛 2019-03-05
        '''
        close_price = CStockDaily.get_real_close(ts_code, current_date)
        cash_amount, stock_amount = CAccount.get_current_amounts(
    account_id)
```

Listing 16: 获取收盘价

获取客户现金资产，根据用户现金资产，使用 10% 的现金购买股票，如下所示：

```
    cash_amount, stock_amount = CAccount.get_current_amounts(account_id)
    percent = 0.1
    buy_shares = AshareStrategy1.calculate_buy_money(cash_amount, percent
    , close_price)
```

Listing 17: 确定需要购买的数量

buy_shares 是将要购买的股票数，这里使用了 A 股的第一个策略，这个也是用户购买的策略。
在 t_account_io 表中产生一条资金转出记录，减少用户现金资产，如下所示：

```
    rst = CAccount.withdraw(account_id, buy_amount)
```

Listing 18: 账户流水表产生资金流出记录

减少用户的现金资产，如下所示：

```
    CAccount.update_cash_amount(account_id, cash_amount - buy_amount)
```

Listing 19: 减少现金资产

增加用户持股量：

```
    stock_id = CStock.get_stock_id_by_ts_code(ts_code)
    CUserStock.buy_stock_for_user(user_id, stock_id, buy_shares,
    close_price)
```

Listing 20: 增加用户持股量

增加用户股票资产：

更新用户股票持有量

在 t_user_stock_io 表中产生一条股票购买记录，增加 t_user_stock 表中持股量，增加客户股票资产

### 4.4 回测引擎开发

接收当日收盘数据，将其作为参数调用策略类，策略类进行分析和预测，返回参数为买或卖，以及买入或卖出的数量，回测引擎进行买入和卖出操作。每个应用在 app 下是一个目录，对于 A 股日线策略研究，对应的目录为 asde。该类以 startup 为入口。

#### 4.4.1 获取要研究的股票池

我们首先需要实现获取股票池中单一股票的信息，如下所示：

```
# app.asde.asde_btn.py
    def get_stock_vo(self, stock_id, ts_code, start_dt, end_dt):
        '''
        获取单支股票的基本信息和数据集
        @param stock: 股票编号
        @param ts_code; 股票编码
        @param start_dt: 开始日期
        @param end_dt: 结束日期
        @return 返回股票基本信息，均值、方差和训练样本集、验证样本集、测试样本集
        @version v0.0.1 闫涛 2019.03.12
        '''
        stock_vo = {'stock_id': stock_id, 'ts_code': ts_code}
        # 求出均值和方差
        stock_vo['train_x'], stock_vo['train_y'], \
                    stock_vo['validate_x'], stock_vo['validate_y'], \
                    stock_vo['test_x'] = CStockDaily.\
                        generate_stock_daily_ds(ts_code, start_dt, end_dt)
        stock_vo['mus'], stock_vo['stds'] = AsdeDs.get_mean_stds(stock_vo['train_x'])
        # 对原始数据集进行归一化
        AsdeDs.normalize_datas(stock_vo['train_x'], stock_vo['mus'], stock_vo['stds'])
        return stock_vo
```

Listing 21: 读出单支股票信息

为了保证模型的效果，这里对原始数据进行了归一化，具体方法为减去均值然后再除以方差，代码如下所示：

```
# app.asde.asde_ds.py
    @staticmethod
    def normalize_data(datas, idx, mus, stds):
        ''' 归一化方法：减去均值再除以标准差 '''
        datas[:, idx:idx+1] = (datas[:, idx:idx+1] - mus[idx]) / stds[idx]

    @staticmethod
    def normalize_datas(datas, mus, stds):
        ''' 对开盘价、最高价、最低价、收盘价等进行归一化 '''
        AsdeDs.normalize_data(datas, AsdeDs.open_idx, mus, stds)
        AsdeDs.normalize_data(datas, AsdeDs.high_idx, mus, stds)
        AsdeDs.normalize_data(datas, AsdeDs.low_idx, mus, stds)
        AsdeDs.normalize_data(datas, AsdeDs.close_idx, mus, stds)
        AsdeDs.normalize_data(datas, AsdeDs.pre_close_idx, mus, stds)
        AsdeDs.normalize_data(datas, AsdeDs.amt_chg_idx, mus, stds)
        AsdeDs.normalize_data(datas, AsdeDs.pct_chg_idx, mus, stds)
        AsdeDs.normalize_data(datas, AsdeDs.vol_idx, mus, stds)
        AsdeDs.normalize_data(datas, AsdeDs.amount_idx, mus, stds)
```

Listing 22: 数据集归一化

每个应用如果需要处理数据集，可以通过相应的数据集类来实现。我们这里定义 asde_ds 类，在其中定义对数据集进行归一化的方法，如下所示：

```
    def get_stocks(self, start_dt, end_dt):
        '''
        获取股票池中股票的基本信息、均值、方差、训练样本集、验证样本集和测试样本集
        @param start_dt: 开始时间
        @param end_dt: 结束时间
        @return 股票池中股票信息列表
        @version v0.0.1 闫涛 2019-03-12
        '''
```

```
        stocks = []
        stock_vo = self.get_stock_vo(69, '603912.SH', start_dt, end_dt)
        stocks.append(stock_vo)
        #stock_vo = self.get_stock_vo(1569, '300666.SZ', start_dt, end_dt
    )
        #stocks.append(stock_vo)
        return stocks
```

Listing 23: 获取股票池信息

#### 4.4.2 训练模型

在上一步中，我们已经获取到了每支股票的训练样本集、验证样本集和测试样本集，接下来我们需要为每个股票训练一个机器学习模型，代码如下所示：

```
    # 训练初始模型
    for stock in stocks:
        stock['svm'] = AsdeSvm()
        stock['svm'].train(stock['train_x'], stock['train_y'])
    print('svm:{0}'.format(stocks[0]['svm']))
    test_x = [stocks[0]['train_x'][0]]
    rst = stocks[0]['svm'].predict(test_x)
    print('预测结果: {0}'.format(rst))
```

Listing 24: 训练 svm 模型

#### 4.4.3 策略类

策略类负责训练机器学习模型，根据行情数据决定购买还是卖出，以具体数量，是量化交易平台的核心。在实际应用中会有多个策略类，客户根据自己的判断选择不同的策略类，形成系统的用户。

```
# A股日线策略类
class AsdeStrategy1(object):
    def __init__(self):
        self.name = 'AsdeStrategy1'

    def setup_stock_ml_model(self, stock):
        '''
        初始化每支股票的机器学习模型，在本策略中使用支撑向量机
        @param stock: 股票值对象，有样本集
        @version v0.0.1 闫涛 2019-03-15
        '''
        stock['svm'] = AsdeSvm()
        stock['svm'].train(stock['train_x'], stock['train_y'])
        print('svm:{0}'.format(stock['svm']))
        test_x = [stock['train_x'][0]]
        rst = stock['svm'].predict(test_x)
        print('rst:{0}'.format(rst))

# 在回测引擎中的调用
# 训练初始模型
for stock in stocks:
    strategy.setup_stock_ml_model(stock)
```

Listing 25: 策略类之机器模型训练

#### 4.4.4 账户历史信息维护

**将 $\mathbf{t}_account$ $t_account_hist$** 股票总资产的统计：首先获取用户的持股列表，求出每支股票本日收盘价，计算每支股票的资产，然后求和。

**添加 $\mathbf{c}_account$**

### 4.4.5 绘制收益曲线

我们可以以天为单位，绘制用户的总资产变化情况折线图。在用户总资产日变化曲线绘制中，有两个难点问题，分别是中文显示问题和横坐日期显示问题，具体处理方法如下所示：

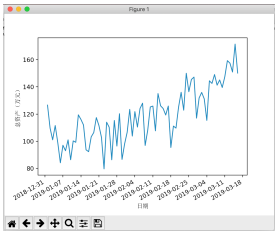
```
    def draw_curve_demo():
        dates = []
        idx = 0.1
        today = AppUtil.get_today_obj()
        curr_date = AppUtil.parse_date('20190101')
        ys = []
        mu = 0
        sigma = 10
        num = 100
        rand_data = np.random.normal(mu, sigma, num)
        print(rand_data)
        i = 0
        while curr_date <= today:
            dates.append(AppUtil.format_date(curr_date, AppUtil.DF_HYPHEN
    ))
            curr_date += timedelta(days=1)
            ys.append(idx*idx + 100 + rand_data[i])
            idx += 0.1
            i += 1

        xs = [datetime.datetime.strptime(d, '%Y-%m-%d').date() for d in
    dates]
        font = FontProperties(fname='./work/simsun.ttc') # 载入中文字体
        plt.rcParams['axes.unicode_minus']=False # 正确显示负号
        plt.gca().xaxis.set_major_formatter(mdates.DateFormatter('%Y-%m-%
    d'))
        plt.gca().xaxis.set_major_locator(mdates.WeekdayLocator(byweekday
    =(mdates.MO)))
        plt.title('总资产变化曲线', fontproperties=font)
        plt.xlabel('日期', fontproperties=font)
        plt.ylabel('总资产（万元）', fontproperties=font)
        plt.plot(xs, ys)
        plt.gcf().autofmt_xdate()
        plt.show()
```

Listing 26: 总资产变化曲线绘制技术

绘制出的曲线如下图所示：

Figure 3: 用户总资产变化曲线



# 5 深度学习入门

## 5.1 Tensorflow 底层技术

### 5.1.1 依赖控制

由于 tensorflow 相当于一门语言，我们在定义计算图时，相当于进行编程，这些程序只有在 session 中才会执行，而 session 中是会根据依赖关系来决定执行哪些代码，所以我们必须通过依赖关系来告诉 tensorflow 执行哪些代码，如下所示：

## 5.2 线性回归

线性回归的定义：假设有一个问题，观察到的样本为 $\boldsymbol{x} \in R^n$ 且共有 $m$ 个训练样本，同时每个样本 $\boldsymbol{x}_i$ 对应一个数值 $y_i$，并且我们假设其对应关系为：$y = \boldsymbol{w} \cdot \boldsymbol{x} + b$，整个问题可以表示为：

$$\boldsymbol{y} = X \cdot \boldsymbol{w} + \boldsymbol{b} \tag{1}$$

其中矩阵 $X \in R^{m \times n}$，每一行代表一个样本。
我们的代价函数定义为最小平方误差函数：

$$\mathcal{L} = \frac{1}{n} \sum_{i=1}^{m} (y_i - \boldsymbol{w}\boldsymbol{x}_i + b_i) \tag{2}$$

这时我们的任务就变为：

$$\arg\min_{\boldsymbol{w},\boldsymbol{b}} \frac{1}{n} \sum_{i=1}^{m} (y_i - \boldsymbol{w}\boldsymbol{x}_i + b_i) \tag{3}$$

由于这个问题比较简单，解这个问题有两种方法：解析法和迭代法。其中解析法的解为：

$$\begin{aligned} \bar{\boldsymbol{x}} &= \frac{1}{n} \sum_{i=1}^{n} \boldsymbol{x}_i \\ \bar{y} &= \frac{1}{n} \sum_{i=1}^{n} y_i \end{aligned} \tag{4}$$

其解析解为：

$$\begin{aligned} \hat{\boldsymbol{w}} &= \frac{\sum_{i=1}^{n} (\boldsymbol{x} - \bar{\boldsymbol{x}})(y_i - \bar{y})}{\sum_{i=1}^{n} (\boldsymbol{x}_i - \bar{\boldsymbol{x}})^2} \\ \hat{\boldsymbol{b}} &= \bar{y} - \boldsymbol{w}\bar{\boldsymbol{x}} \end{aligned} \tag{5}$$

如果是迭代法则为梯度下降算法，参数更新公式为：

$$\begin{aligned} \boldsymbol{w}_{t+1} &= \boldsymbol{w}_t - \alpha \frac{\partial \mathcal{L}}{\partial \boldsymbol{w}_t} \\ \boldsymbol{b}_{t+1} &= \boldsymbol{b}_t - \alpha \frac{\partial \mathcal{L}}{\partial \boldsymbol{b}_t} \end{aligned} \tag{6}$$

在讲解具体的代码之前，我们先来讲解一下 Python 中的 __call__ 函数。我们知道，在 Python 中，类实例也是可调用对象，只需要在类定义中添加 __call__ 函数定义，即使用类实例加括号的形式进行调用了。我们利用这一特性定义线性回归类，如下所示：

```
class LinearRegression(object):
def __init__(self):
    self.w = tf.get_variable('w',
                dtype=tf.float32, shape=[],
                initializer=tf.zeros_initializer()
    )
    self.b = tf.get_variable(
        'b', dtype=tf.float32, shape=[],
        initializer=tf.zeros_initializer()
    )
```

```
11
12     def __call(self, x):
13         return self.w * x + self.b
```

Listing 27: 线性回归类定义

参考文献: **?—?—?**