★★★

严谨　认真　持续

用股票池择时框架开发一个量化策略

量化投资实战交易体系必修课

系列4 – 快速开发一个量化策略

普量学院
PLOUTO-QUANTS

目录

CONTENT

逻辑概述

PART ONE

策略开发

PART TWO

策略改进

PART THREE

逻辑概述

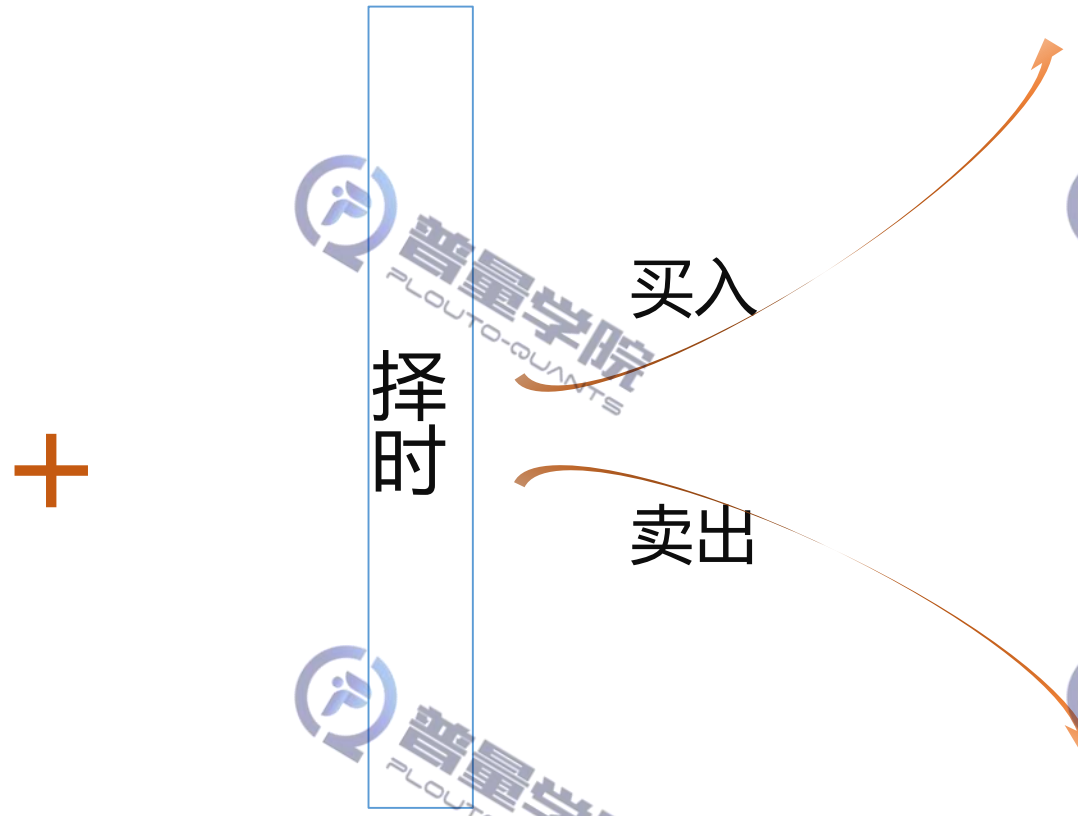
PART ONE

股票池择时基本框架

Alpha 的非常规定义

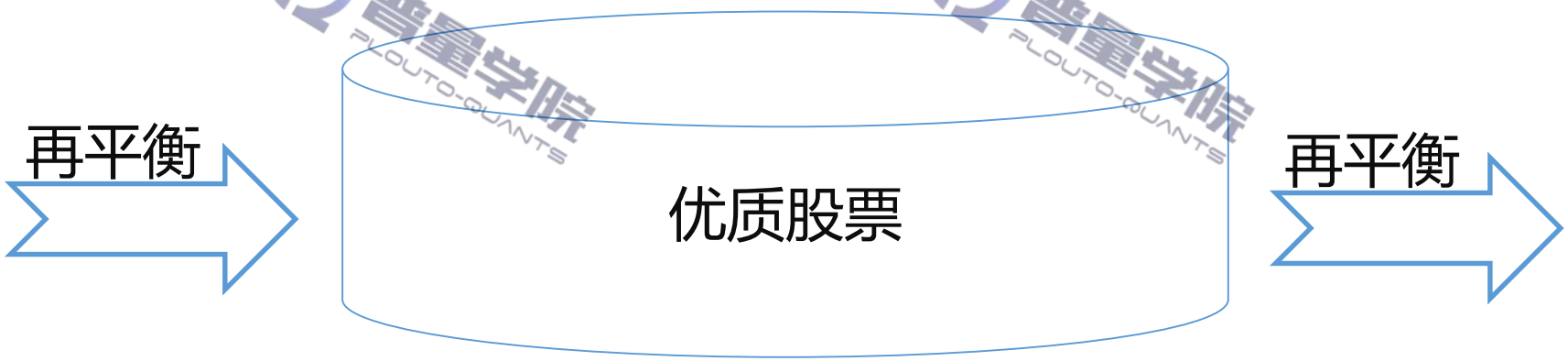
在交易中关于持有头寸选择和买卖时机把握的技巧

股票池择时基本框架



沪深300指数(000300.SH/399300.SZ)

构建股票池：多因子分析



择时

择时

买入

卖出

- 什么时候应该买?
- 什么时候应该卖?

策略开发

PART TWO

我们的策略

- 股票池
  - 条件
    - 0 < pe_ttm < 30
    - pe_ttm正序排列，最多取100只
    - 排除停牌
  - 调整周期
    - 7个交易日

我们的策略

- 择时
  - 买入
    - K线上穿10日均线，第二日开盘买入
  - 卖出：
    - K线下穿10日均线，第二日开盘卖出
    - 被调出股票池

开始编码之前

daily

```
{
'code': '000651',
'time': '2018-03-21',
'open': 52.25,
'high': 52.61,
'low': 51.68,
'close': 51.70,
'adjfactor': 162.45,
'pe_ttm': 15.82,
'trade_status': '交易',
'chg_pct': -0.96
}
```

股票代码
日期
开盘价
最高价
最低价
收盘价
复权因子
滚动市盈率
交易状态
涨跌幅

MongoDB

Python

实现股票池

```
def stock_pool(begin_date, end_date):
    """
    股票池
    :param begin_date: 开始日期
    :param end_date: 结束日期
    :return: tuple, 所有调整日, 以及调整日和代码列表对应的dict
    """

    adjust_date_codes_dict = dict()

    # 获取指定时间区间的所有交易日
    time_cursor = db.daily.find(
        {'code': '000001.SH', 'time':{'$gte': begin_date, '$lte': end_date}},
        sort=[('time', ASCENDING)],
        projection={'time': True},
        batch_size=1000
        )
    all_dates = [x['time'] for x in time_cursor]
```

实现股票池

```
# 上一期的所有股票代码
last_phase_codes = []
# 调整周期是7个交易日
adjust_interval = 7
# 所有的调整日
all_adjust_dates = []
# 在调整日调整股票池
for _index in range(0, len(all_dates), adjust_interval) :
    # 保存调整日
    adjust_date = all_dates[_index]
    all_adjust_dates.append(adjust_date)

    # 查询出调整当日，0 < pe_ttm < 30，且非停牌的股票
    # 最重要的一点是，按照pe_ttm正序排列，只取前100只
    daily_cursor = db.daily.find(
        {'time': adjust_date, 'pe_ttm': {'$lt': 30, '$gt': 0}, 'trade_status': '交易'},
        sort=[('pe_ttm', ASCENDING)],
        projection={'code': True},
        limit=100
        )
```

实现股票池

```
codes = [x['code'] for x in daily_cursor]

# 本期股票列表
this_phase_codes = []

# 查询出上次股票池中正在停牌的股票
if len(last_phase_codes) > 0:
    suspension_cursor = db.daily.find(
        {'code': {'$in': last_phase_codes}, 'time':adjust_date, 'trade_status': '停牌'},
        projection={'code': True}
        )
    suspension_codes = [x['code'] for x in suspension_cursor]

    # 保留股票池中正在停牌的股票
    this_phase_codes = suspension_codes

# 用新的股票将剩余位置补齐
this_phase_codes += codes[0 : 100 - len(this_phase_codes)]
# 将本次股票设为下次运行的时的上次股票池
last_phase_codes = this_phase_codes
```

实现股票池

```
        # 建立该调整日和股票列表的对应关系
        adjust_date_codes_dict[adjust_date] = this_phase_codes

    # 返回结果
    return (adjust_dates, adjust_date_codes_dict)
```

评价股票池

```
def evaluate_stock_pool():
    """
    对股票池做一个简单的评价
    """
    # 设定评测周期
    adjust_dates, codes_dict = stock_pool('2015-01-01', '2018-05-25')

    # 用DataFrame保存收益
    df_profit = pd.DataFrame(columns=['profit', 'hs300'])
    # 设置第一天是股票池和基准的收益都为0
    df_profit.loc[adjust_dates[0]] = {'profit': 0, 'hs300': 0}

    # 找到基准的第一天收盘数值
    hs300_begin_value = db.daily.find_one({'code': '000300.SH', 'time': adjust_dates[0]})['close']

    # 通过净值计算累计收益，将净值的初始值设置为1
    net_value = 1
```

评价股票池

```
# 按照调整日一期一期股票池进行统计
for _index in range(1, len(adjust_dates) - 1):
    # 拿到上一期的调整日
    last_adjust_date = adjust_dates[_index - 1]
    # 获取上一期的股票池
    codes = codes_dict[last_adjust_date]

    # 拿到当期的调整日
    current_adjust_date = adjust_dates[_index]

    # 构建股票代码和后复权后买入价格的股票
    code_buy_close_dict = dict()
    buy_daily_cursor = db.daily.find(
        {'code': {'$in': codes}, 'time': last_adjust_date},
        projection={'close':True, 'adjfactor': True, 'code': True}
        )

    for buy_daily in buy_daily_cursor:
        code = buy_daily['code']
        code_buy_close_dict[code] = buy_daily['close'] * buy_daily['adjfactor']
```

评价股票池

```
# 获取到期的股价
sell_daily_cursor = db.daily.find(
    {'code': {'$in': codes}, 'time': current_adjust_date},
    projection={'close':True, 'adjfactor': True, 'code': True}
    )

# 计算单期收益
profit_sum = 0
for sell_daily in sell_daily_cursor:
    code = sell_daily['code']

    buy_close = code_buy_close_dict[code]
    sell_close = sell_daily['close'] * sell_daily['adjfactor']

    profit_sum += (sell_close - buy_close)/buy_close

profit = round(profit_sum/len(codes), 4)

# 找到基准在当期调整日的收盘值
hs300_close = db.daily.find_one({'code': '000300.SH', 'time': current_adjust_date})['close']
```

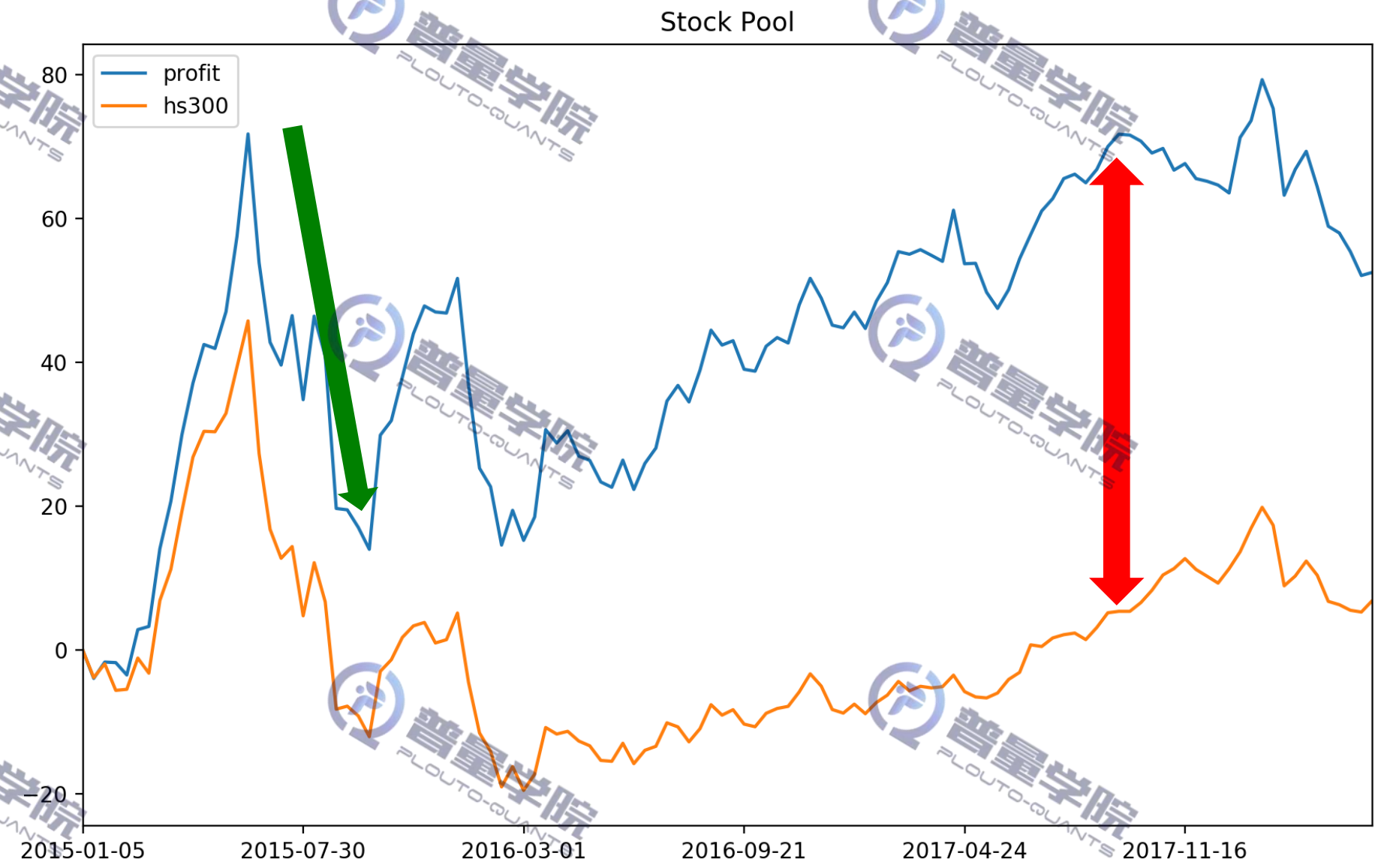
评价股票池

```
    # 找到基准在当期调整日的收盘值
    hs300_close = db.daily.find_one({'code': '000300.SH', 'time': current_adjust_date})['close']

    # 计算净值和累积收益
    net_value = net_value * (1 + profit)
    # 累积收益=当前净值 - 首期净值
    df_profit.loc[current_adjust_date] = {
    'profit': round((net_value - 1) * 100, 4),
    'hs300': round((hs300_close - hs300_begin_value) * 100/hs300_begin_value, 4)}

# 绘制曲线
df_profit.plot(title='Stock Pool', kind='line')
# 显示图像
plt.show()
```

评价股票池



计算当日收盘价和MA10的关系

```python
def compare_close_2_ma_10(dailies):
    """
    比较当前的收盘价和MA10的关系
    :param dailies: 日线列表，10个元素，最后一个是当前交易日
    :return: 0 相等，1 大于， -1 小于，None 结果未知
    """

    # 如果用来计算的日线列表小于10个，则返回None，表示无效值
    if len(dailies) < 10:
        return None

    # 当前日期为最后一个元素
    current_daily = dailies[9]
    close_sum = 0
    count = 0
```

计算当日收盘价和MA10的关系

```
for daily in dailies:
    # 10天当中，只要有一天停牌则返回False
    if daily['trade_status'] == '停牌' or 'adjfactor' not in daily:
        return None

    # 用后复权累计
    close_sum += daily['close'] * daily['adjfactor']

# 计算MA10
ma_10 = close_sum/10

# 判断收盘价和MA10的大小
post_adjuested_close = current_daily['close'] * current_daily['adjfactor']
differ = post_adjuested_close - ma_10
if differ > 0:
    return 1
elif differ < 0 :
    return -1
else:
    return 0
```

计算买入信号

```
def is_k_up_break_ma10(code, _date):
    """
    判断某只股票在某日是否满足K线上穿10日均线

    :param code: 股票代码
    :param _date: 日期
    :return: True/False
    """

    # 如果股票当日停牌或者是下跌，则返回False
    current_daily = db.daily.find_one(
        {'code': code, 'time': _date, 'trade_status': '交易'})
    if current_daily is None or current_daily['pct_chg'] < 0:
        return False
```

计算买入信号

```
# 计算MA10
daily_cursor = db.daily.find(
    {'code': code, 'time': {'$lte': _date}},
    sort=[('time', DESCENDING)],
    limit = 11,
    projection={'close': True, 'adjfactor': True, 'trade_status': True}
    )

dailies = [x for x in daily_cursor]

# 如果日线个数不足以判断两个交易日和其MA10的关系，则认为不符合条件
if len(dailies) < 11:
    return False

# 将列表的排序方向倒置
dailies.reverse()
```

计算买入信号

```
# 判断前一个交易日的收盘价和其对应的10日均线的关系
last_close_2_last_ma10 = compare_close_2_ma_10(dailies[0:10])
# 判断当前交易日的收盘价和其对应的10日均线的关系
current_close_2_current_ma10 = compare_close_2_ma_10(dailies[1:])

# 如果有一个关系的计算出现问题，则认为不符合条件
if last_close_2_last_ma10 is None or current_close_2_current_ma10 is None:
    return False

# 判断收盘价和MA10的大小
return last_close_2_last_ma10 <= 0 and current_close_2_current_ma10 == 1
```

计算卖出信号

```
def is_k_down_break_ma10(code, _date):
    """
    判断某只股票在某日是否满足K线下穿10日均线

    :param code: 股票代码
    :param _date: 日期
    :return: True/False
    """

    # 如果股票当日停牌或者是下跌，则返回False
    current_daily = db.daily.find_one(
        {'code': code, 'time': _date, 'trade_status': '交易'})
    if current_daily is None or current_daily['pct_chg'] > 0:
        return False
```

```
# 判断收盘价和MA10的大小
return last_close_2_last_ma10 >= 0 and current_close_2_current_ma10 == -1
```

找到调出股票

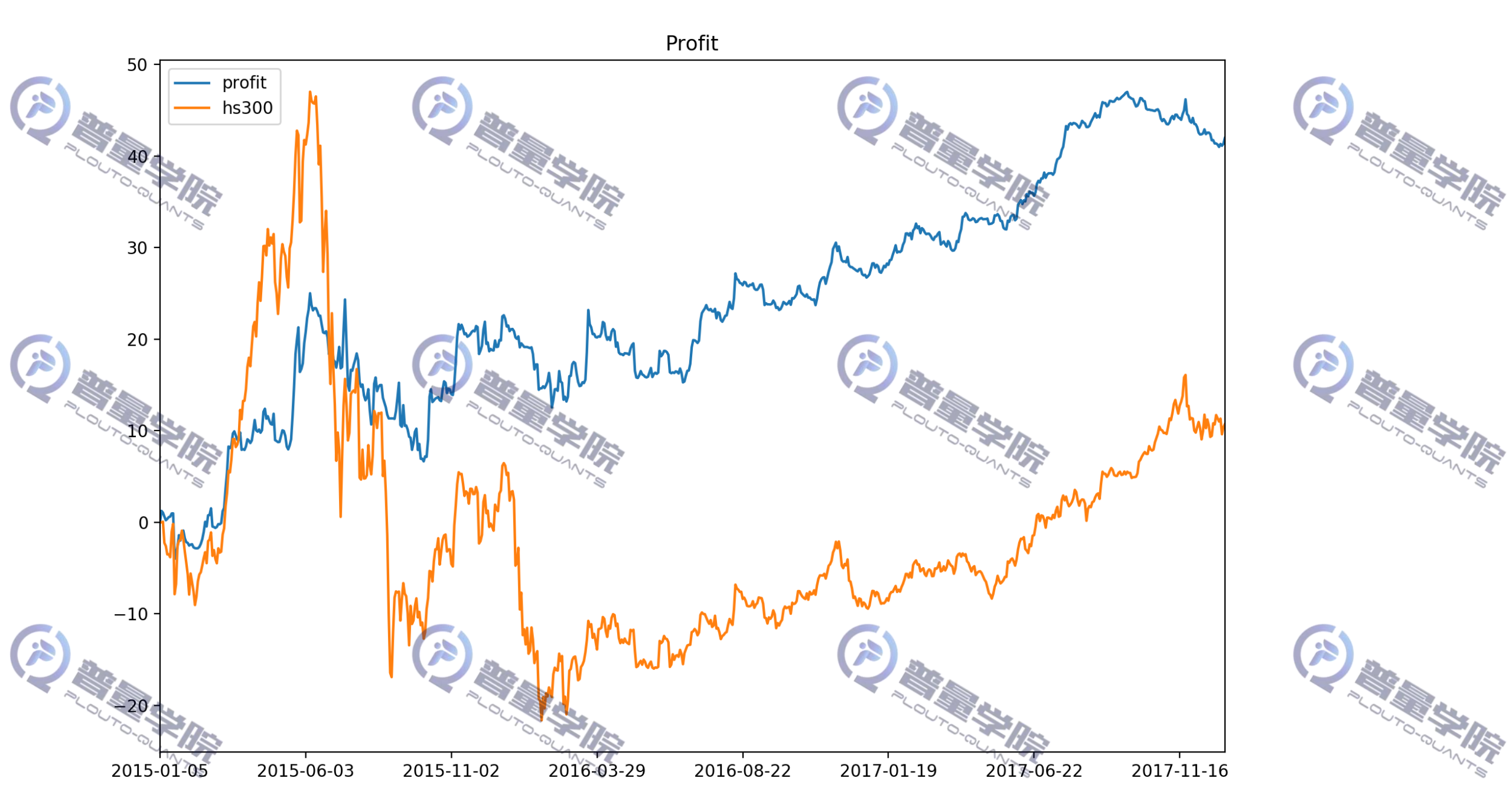
```
def find_out_stocks(last_phase_codes, this_phase_codes):
    """
    找到上期入选本期被调出的股票，这些股票将必须卖出
    :param last_phase_codes: 上期的股票列表
    :param this_phase_codes: 本期的股票列表
    :return: 被调出的股票列表
    """
    out_stocks = []

    for code in last_phase_codes:
        if code not in this_phase_codes:
            out_stocks.append(code)

    return out_stocks
```

开始回测

- 总资金：
  - 1000万元
- 仓位
  - 均仓
  - 每只20万元



策略改进

PART THREE

成功策略的六大要素

- 可靠度、胜率、或者盈亏时间比例
- 在以最小单位（比如1手股票或者1手期货合约）进行交易时，相对于亏损而言，你的盈利水平的相对规模
- 每笔交易的成本
- 获得交易机会的频率
- 头寸规模的确定，或者说，一次交易多少个单位
- 你的投资资本规模，或者说本金的规模

更多

- 信号改进
  - 更细颗粒度
  - 尝试其他信号
- 控制亏损
  - 止盈止损
- 资金管理策略
  - 头寸管理
  - 加仓管理

答疑热线

牛小秘

微信：niuxiaomi1

普量学院
PLOUTO-QUANTS