目　录

果实杯金融设计开发大赛 选题 1 设计报告

1 整体框架

1.1 开发环境

```
"""
操作系统：Windows 7 Service Park 1
开发语言：Python 3.6
第三方库：Matplotlib 2.2.2
          Numpy 1.14.2
          Pandas 0.22.0
          PyMySQL 0.8.0
"""
```

1.2 代码结构

代码结构如下。对于算法较为复杂的部分，请参考下一节“2 算法解析”；并同时参考源代码中的注释部分：

```
"""
{work_dir}                                  # 程序工作文件夹

--main.py                                   # 程序主文件

--dataLoad.py                               # 数据读取与保存
    --read_csv(csv_file_path)               # 从 csv 文件中读取股票数据
    --create_mysql()                        # 新建数据库"guoshi"，若已存在，先删除后新建
    --insert_mysql(id, date, close)         # 插入数据到数据库
    --query_mysql()                         # 从数据库读取数据

--dataProcess.py                            # 数据处理，计算 RSI、BOLL 指标
    --calculate_RSI(stock_data, N)          # 计算 RSI 指标，改变 N 可计算长期和短期 RSI
    --calculate_BOLL(stock_data, N, M=2)    # 计算 BOLL 指标

--judge.py                                  # 判断是否发出 买入/卖出/警告信号
    --judge(BOLL, short_RSI, long_RSI, start_index)

```

```
--backtesting.py                                # 收益回测
    --backtest(stock_data, buy, sell)

--drawCurves.py                                 # 绘制 BOLL、RSI、买入卖出信号、收益回测等曲线
    --draw_BOLL_and_RSI(stock_data, BOLL, short_RSI, long_RSI)
    --draw_trade_signals(stock_data, BOLL, short_RSI, long_RSI, buy, warning, sell)
    --draw_backtesting(backtest_result)

"""
```

1.3 整体流程

代码的整体运行流程如下。对于算法较为复杂的部分，请参考下一节“2 算法解析”；并同时参考源代码中的注释部分：：

```
# 1.从 csv 文件读取股票数据，并存入 MySQL 数据库中
read_csv()
create_mysql()
insert_mysql()

# 2.从 MySQL 数据库中读取股票数据
query_mysql()

# 3.计算 RSI 指标（含长期 RSI、短期 RSI）和 BOLL 指标
calculate_RSI()
calculate_BOLL()

# 4.判断买入卖出信号
judge()

# 5.收益回测
backtesting()

# 6.绘制各种曲线
draw_BOLL_and_RSI()
draw_trade_signals()
draw_backtesting()
```

2 算法解析

源代码中注释丰富，可以直接阅读源代码理解。对于其中算法较为复杂的部分，本节给出相应的伪代码帮助理解。

2.1 计算 RSI 及 BOLL 指标

RSI 和 BOLL 计算的伪代码如下。参数 N 为天数，可手动调节，依据量化投资的默认值，short_RSI 取 N=6，long_RSI 取 N=12，BOLL 取 N=20。在下一节“3 结果分析”中提到，该取值并非使得单支股票收益最大的参数。但比赛中只提供一支股票的信息，如果对该股票进行调参，则会发生过拟合情况（即所调出的参数仅对该股票效果较好，对其他股票没有很好的适应能力），故在参数选择上，所有参数均选择量化投资默认值。所有参数均对用户留有修改接口，用户使用时，可根据不同股票，方便地更换参数。

```
def calculate_RSI(stock_data, N):          # stock_data 为股票数据，N 为天数
    差值 = stock_data['收盘价'] - stock_data['前一天收盘价']

    上涨 = 差值.copy()                          # 从"差值"复制出"上涨"
    上涨[上涨 < 0] = 0                          # 把"上涨"中小于 0 的，置为 0
    下跌 = 差值.copy()                          # 从"差值"复制出"下跌"
    下跌[下跌 > 0] = 0                          # 把"下跌"中大于 0 的，置为 0

    上涨_roll = 指数移动平均（数据=上涨，  天数=N）
    下跌_roll = 指数移动平均（数据=下跌，  天数=N）

    RS = 上涨_roll / 下跌_roll
    RSI = 100.0 - (100.0 / (1.0 + RS))


def calculate_BOLL(stock_data, N, M=2)       #  N 为天数，M 为宽度系数
    中轨线 = 简单移动平均（数据=stock_data['收盘价']，天数=N）
    标准差 = 标准差（数据=stock_data['收盘价']，天数=N）
    上轨线 = 中轨线 + M * 标准差
    下轨线 = 中轨线 - M * 标准差
```

2.2 判断是否买卖

买卖信号的判断依据如下，函数 judge()判断以下四个情况是否有其一被满足，并相应地把“买入”、“卖出”、“警告”信号记录到数组中

```
"""
    condition_1: short_RSI 在 50 以下，上穿 long_RSI，形成金叉，有价格在 BOLL_middle 上方，
                 买入信号
    condition_2: short_RSI 下穿 70，下穿 long_RSI 形成死叉，卖出信号
    condition_3: 价格下穿 BOLL 中轨，卖出信号
    condition_4: 价格下穿 BOLL 上轨，卖出警告
"""
```

Condition 1：

```
def judge_condition_1(BOLL, short_RSI, long_RSI, start_index):
    """ condition_1: short_RSI 在 50 以下，上穿 long_RSI，形成金叉，
                     有价格在 BOLL_middle 上方，买入信号

    start_index:
        由于简单移动平均的计算方式，前 N 天的 BOLL 指标是没有意义的，
        参数 start_index 指示从第几天开始判断 condition 是否满足
    """
    condition_1 = 全为 0 的数组          # 记录 condition 是否满足
    RSI_up_cross = 全为 0 的数组         # 记录 RSI 是否发生上穿越

    for 所有可能的 index:
        if short_RSI 在 50 以下，上穿 long_RSI，形成金叉：
            RSI_up_cross[index] = 最大等待天数       # 等待"有价格在中轨以上"的耐心值
        else:
            if RSI_up_cross[index - 1] >= 1:  # 前面出现过金叉，且耐心值还没有消耗完
                RSI_up_cross[index] = RSI_up_cross[index - 1] - 1     # 递减耐心值
            else:                                    # 前面没有出现金叉或耐心值消耗完
                RSI_up_cross[index] = 0

    begin_new_search = False                         # 每一个金叉，至多一次买入信号
    for 所有可能的 index:
        if RSI_up_cross[index] == 最大等待天数:          # 说明这天是出现金叉的第一天
            begin_new_search = True              # 开始新一轮"有价格在中轨以上"的搜索
```

```
        if begin_new_search and RSI_up_cross[index] >= 1 and 有价格在中轨以上:
            condition_1[index] = 1                          # condition 满足，买入信号
            begin_new_search = False    # 这次金叉已经发出一次买入信号，不再发出第二次

    return condition_1
```

Condition 2:

```
def judge_condition_2(short_RSI, long_RSI, start_index):
    """ condition_2: short_RSI 下穿 70，下穿 long_RSI 形成死叉，卖出信号

    start_index:
        由于简单移动平均的计算方式，前 N 天的 BOLL 指标是没有意义的，
        参数 start_index 指示从第几天开始判断 condition 是否满足
    """
    condition_2 = 全为 0 的数组             # 记录 condition 是否满足
    RSI_down_cross = 全为 0 的数组          # 记录 RSI 是否发生下穿 70

    for 所有可能的 index:
        if 发生下穿 70:
            RSI_down_cross[index] = 最大等待天数       # 等待"下穿 long_RSI 形成死叉"的耐心值
        else:
            if RSI_down_cross[index - 1] >= 1:    # 前面出现过下穿 70，且耐心值还没有消耗完
                RSI_down_cross[index] = RSI_down_cross[index - 1] - 1       # 递减耐心值
            else:
                RSI_down_cross[index] = 0              # 前面没有出现下穿 70 或耐心值消耗完

    begin_new_search = False                                # 每一次下穿 70，至多一次卖出信号
    for 所有可能的 index:
        if RSI_down_cross[index] == 最大等待天数:           # 说明这天是出现下穿 70 的第一天
            begin_new_search = True                    # 开始新一轮"有价格在中轨以上"的搜索
        if begin_new_search and RSI_down_cross[index] >= 1:
            if 下穿 long_RSI 形成死叉:
                condition_2[index] = 1                          # condition 满足，卖出信号
                begin_new_search = False # 这次下穿 70 已经发出一次买入信号，不再发出第二次

    return condition_2
```

Condition 3：

```
def judge_condition_3(BOLL, start_index):
    """ condition_3: 价格下穿 BOLL 中轨，卖出信号

    start_index:
        由于简单移动平均的计算方式，前 N 天的 BOLL 指标是没有意义的，
        参数 start_index 指示从第几天开始判断 condition 是否满足
    """
    condition_3 = 全为 0 的数组              # 记录 condition 是否满足
    for 所有可能的 index:
        if 价格下穿 BOLL 中轨:
            condition_3[index] = 1        # condition 满足，卖出信号

    return condition_3
```

Condition 4：

```
def judge_condition_4(BOLL, start_index):
    """ condition_4: 价格下穿 BOLL 上轨，卖出警告

    start_index:
        由于简单移动平均的计算方式，前 N 天的 BOLL 指标是没有意义的，
        参数 start_index 指示从第几天开始判断 condition 是否满足
    """
    condition_4 = 全为 0 的数组              # 记录 condition 是否满足
    for 所有可能的 index:
        if 价格下穿 BOLL 上轨:
            condition_4[index] = 1        # condition 满足，卖出警告

    return condition_4
```

2.3 收益回测

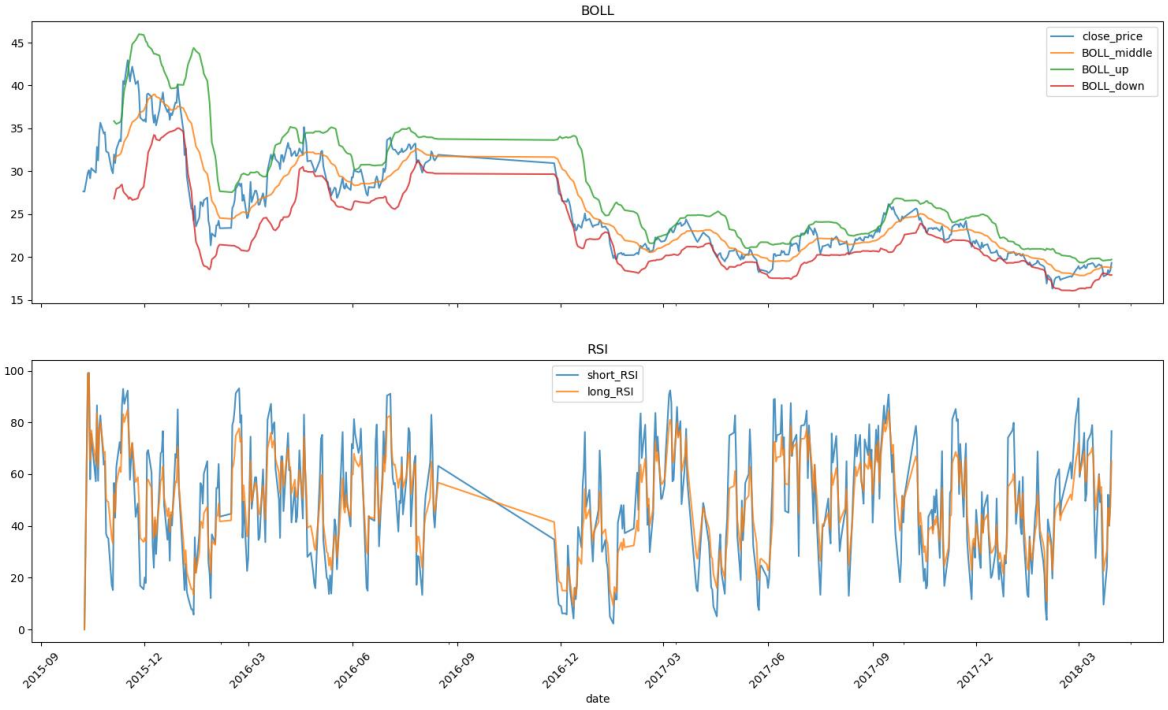
以一千万为初始资金，进行收益回测。若发出买入信号且资金不为 0，则全部资金买入该股票；若发出卖出信号且股票持仓不为 0，则全部卖出该股票。由于比赛不提供大盘数据，收益回测时，以“一直持有该股票”的收益作为基准收益。伪代码如下：

```
def backtest(stock_data, 买入信号, 卖出信号):
    """ 收益回测
        由于没有大盘数据，使用一直持有该股票不卖出的收益曲线作为基准收益曲线
    """

    资金 = 10000000.0                    # 初始资金
    股票数 = 0.0                         # 初始股票数
    策略收益 = 全为0的数组                # 使用策略的收益
    基准收益 = 全为0的数组                # 基准收益
    基准股票数 = 初始资金 / 第0天股价      # 在第0天全部买入股票的股票数

    for 所有可能的index:
        if 前一天发出了卖出信号 and 股票数不为0:    # 第n-1天发出卖出信号，第n天卖出股票
            资金 = 该天全部卖出股票所得资金
            股票数 = 0
        elif 前一天发出了买入信号 and 资金不为0:    # 第n-1天发出买入信号，第n天买入股票
            股票数 = 该天全部买入股票的股票数
            资金 = 0
        策略收益[index] = (资金 +  股票数 * 当天股价) / 初始资金 - 1
        基准收益[index] = 基准股票数 * 当天股价 / 初始资金 - 1
```

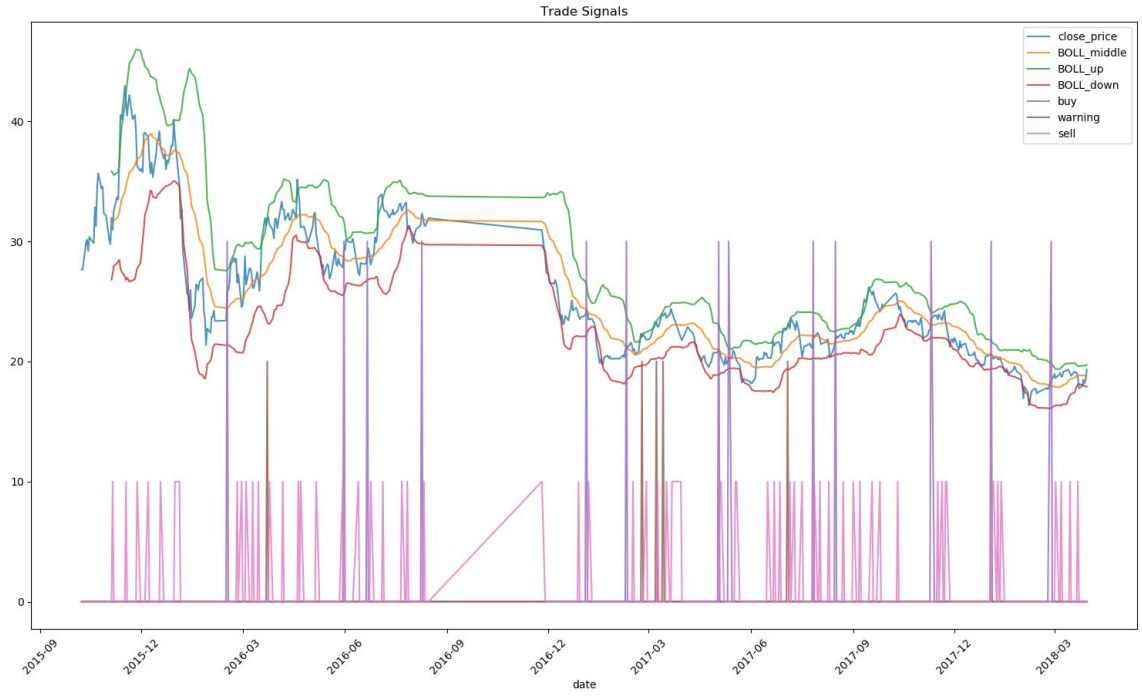
3 结果分析

3.1 股价走势与 BOLL、RSI 指标

股价走势与 BOLL、RSI 指标如下图所示。可以看到该股票在初期有些许上涨，然后又快速下跌，之后缓慢回到初始股价，中期出现停牌现象，停牌结束后，股价一直萎靡不振。



3.2 买入、卖出、警告信号



程序对各项指标进行运算后，给出的信号如上图所示。可以发现，程序给出卖出信

号非常频繁，而给出买入信号则比较谨慎。这与比赛给定的买入策略比较严格，给出的卖出策略比较松有关。

依据量化投资的默认值，short_RSI 取 N=6，long_RSI 取 N=12，BOLL 取 N=20。该取值并非使得单支股票收益最大的参数。但比赛中只提供一支股票的信息，如果对该股票进行调参，则会发生过拟合情况（即所调出的参数仅对该股票效果较好，对其他股票没有很好的适应能力），故在参数选择上，所有参数均选择量化投资默认值。所有参数均对用户留有修改接口，用户使用时，可根据不同股票，方便地更换参数。

3.3 回测收益

回测收益如下图所示，由于该股票一直大多数时间处于下跌状态，基准收益在开始阶段为正，之后一直为负。

使用策略后，策略收益也存在收益为负的情况，但相对基准收益而言，损失大为降低。但如果从“事后诸葛亮”的角度考虑，存在最优策略可以使得收益为正，该程序给出的策略并非最佳策略。

前面已经提到过，为防止调参过拟合，在参数选择和策略选择上，保持默认值。如果有针对性地对不同性质的股票采用不同的参数和策略，将有望获得更高的策略收益。