量化策略算法操作说明

**项目说明**

本项目以3个月的比特币连续曲线数据作为原始输入，旨在通过训练后得到数据去噪后的波峰/波谷最大全距，以期在测试过程中获得理想的胜率和盈率。

**计算基础和算法框架**

1. 计算连续价格曲线数据的统计指标如下(参考：曲线分类-特征提取(一)：https://blog.csdn.net/vegetable\_bird\_001/article/details/80653986)

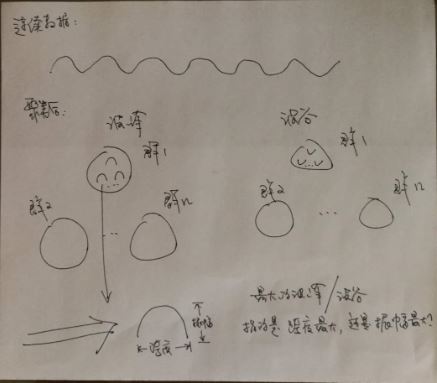
均值(µ)：，其中：为组成单个波峰/波谷的价格数据采集点；

标准差(std)：std=

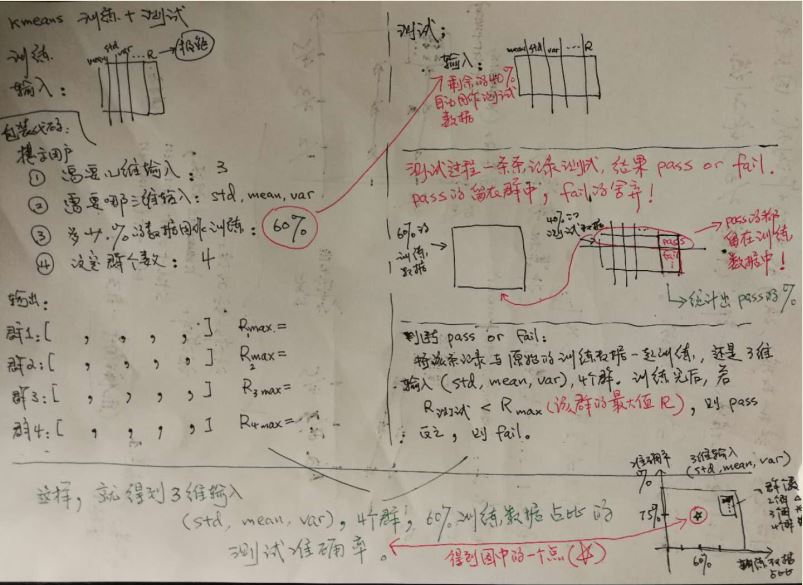
方差(var)：var=

波动率(wave\_rate)：=，其中：(**注：**数据标准/归一化，以全体价格数据作为样本)

1. 算法框架(如下图)



总体来说，对作为原始输入的连续价格曲线数据预处理后，我们得到曲线数据的若干统计指标。波峰/波谷是分开计算的，这里以波峰为例。训练过程中，通过特定的聚类算法，我们可以将统计特征相近的曲线归并到同一个群中，从而得到该群的全距最大值Rmax。全距即上图的振幅，表示该波峰曲线的价格最大值与价格最小值之差。波谷同理。



再进一步展开，从上图我们可以分成左、右、下3块来看：左边是训练过程，主要负责多维度数据提取、训练数据占比和聚类的群cluster个数初始设置，e.g.，一旦原始数据集60%设置为训练数据集，那剩余的40%就作为测试数据集(为验证算法的健壮性，这里采用随机种子来分割训练集数据和测试集数据)；右边是测试+自适应过程，对于测试集记录，首先判断其归属于哪个群，而后与该群的全距最大值(Rmax)进行比较，若，则该测试记录标识为fail；反之，则该测试记录标识为pass。并且，pass的记录都融入到初始分割的训练集数据中，以提高后续测试的精度，并同时完成自适应的流程。最下面那块为典型的一个场景的输出：当组合输入为(均值(mean/µ)、std(标准差)、方差(var))，核心算法Kmeans群cluster个数初始设置为4，训练数据占比为总数据的60%下的胜率(即：图中☆标识的一个点；胜率表示通过的测试记录数与总测试记录数的比值)。

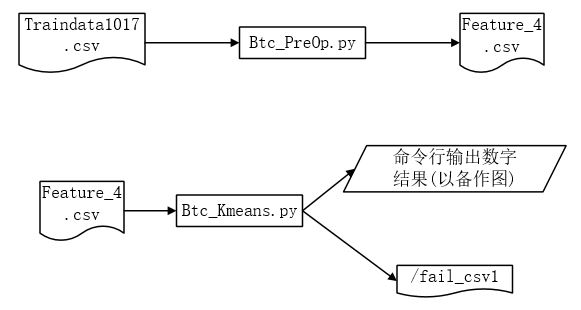
**安装说明**

程序在安装了Python3解释器的机器上都能完美运行。为保证速度，建议至少在6核，8G内存的机器上运行(整个流程运行完毕需要几小时)。

**程序说明**

两个程序，分别负责预处理(Btc\_PreOp.py)、训练+测试+自适应(Btc\_Kmeans.py)环节。为简单起见，建议程序和原始输入数据(traindata1017.csv)安装在同一目录(e.g., /btc\_project)中，且路径中不带中文字符。另，在/btc\_project中自建一空目录(e.g., /fail\_csv1)，用于存放测试失败的记录。

1. 程序流程(如下图)



1. 程序中主要函数

Btc\_PreOp.py：dataCount(path)、dataDenoising(df,form='波谷')、dataFeatureExtraction(data)

def dataCount(path):

"""

整合读入的原始数据(traindata1017.csv)

:path:文件路径

:return:处理后的DataFrame对象

"""

def dataDenoising(df,form='波谷'):

"""

按每个波峰或波谷(form)，通过3-δ筛选噪点，并用噪点前后相邻非噪点数据的均值填充噪点

:df:待去噪的数据

:form:以何种形态去除噪声数据，默认'波谷',可选（'波谷','波峰'）

:return:df(去噪后的DataFrame对象)

"""

def dataFeatureExtraction(data):

"""

特征提取:分别提取/计算数据点个数、均值、中位数、标准差、方差、最大值、最小值、全距、波动率、偏度及峰度

:data:输入数据，待特征提取

:return:已提取的特征数据

"""

Btc\_Kmeans.py：split\_data\_with\_possibility\_and\_kmenas(df\_data, columns=None, base\_dir=None, types=None)、kmeans\_one\_data(df\_train, df\_test, n\_clusters=2, columns=None, save\_fail\_path=None)

def split\_data\_with\_possibility\_and\_kmenas(df\_data, columns=None, base\_dir=None, types=None):

"""

:param df\_data:聚类所需字段

:param columns:要聚类的列，e.g., ['mean', 'std']

:param base\_dir:fail记录存放路径

:param types:波峰or波谷

:return: 字典dict{"k2": [[1,2,3][][]], "k3": [[1,2,3][][]]}

"""

def kmeans\_one\_data(df\_train, df\_test, n\_clusters=2, columns=None, save\_fail\_path=None):

"""

:df\_train:训练数据

:df\_test:测试数据

:n\_cluster:类cluster数

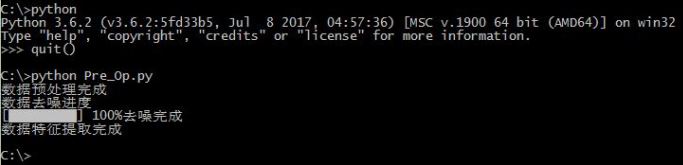
:columns:要聚类的列，e.g., ['mean', 'std']

:return:(测试集pass的记录个数，测试记录总数，胜率，盈率，fail list[456, 454,746,...])

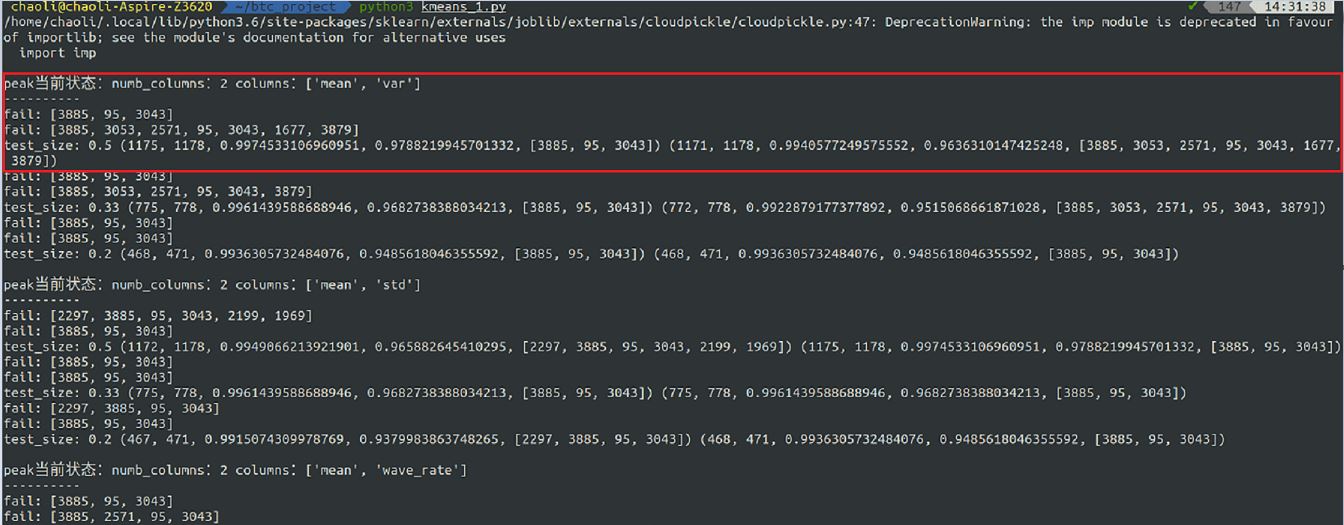
"""

1. 程序运行截图

Btc\_PreOp.py运行截图如下：



**注：**使用Python3解释器，命令行按操作系统、默认路径不同可能需要键入：python3 Pre\_Op.py

Btc\_Kmeans.py运行截图如下：

**注1：**由于运行时间较长，这里只展示了输出结果的开始阶段。

**注2：**结果数据说明(上图红框，从上到下)：

1. peak(波峰数据)，['mean', 'std']:均值和标准差的组合输入；
2. fail:[3885,95,3843]:类数目K=2时的fail记录号列表；
3. fail:[3885,3053,2571,95,3043,1677,3879]: 类数目K=3时的fail记录号列表；
4. test\_size:0.5(测试集数据占比50%),括号中数据从左到右依次是：(K=2时的pass记录数，总测试记录数，胜率，盈率，fail记录号列表)(K=3时的pass记录数，总测试记录数，胜率，盈率，fail记录号列表)。