基于时间序列分析与 LSTM 深度学习的拆单策略研究

摘要

针对问题一，本文以平安银行为例，详细分析其价格、价格变化量以及交易量的时间序列特征。发现交易价格的上升和下降呈现对称模式，交易量存在日模式，即时间序列呈 U 型循环，存在周期性。对所有股票进行相同分析后对比发现，这种特性在 10 支股票上都有出现，具有一定的普适性。我们随后分析股票截面的价格和交易量变化趋势，得到不同股票价格存在相关性的强弱，且交易量的截面呈现出某种聚集模式。

针对问题二，我们首先尝试用简单的多元线性回归模型进行预测，但囿于高频数据噪声大、变化频繁的特性，其预测效果并不佳。故我们以该模型作为参考基线，利用 LSTM（长短期记忆神经网络）进行预测，借助其短期记忆的特性成功预测了 30 个 ticks 内的股票价格与交易量变动情况。并对结果以及模型优缺点进行了分析。

针对问题三，我们基于平均拆分的思想，借助上题的预测信息对平均拆分的周期、买入量等参数进行优化。结果表明，根据其个股价格选择合适的周期以及根据预测价格选择合适的加权买入量有利于降低流动性风险与提高卖出完成率以及提高卖出收益。然而，我们发现诸如美的集团的高价股票存在流动性不足的情况，故我们认为针对这些股票不应选在在一天之内将其卖完，否则将面临市场价格变动幅度大、证监会预警等一系列问题。

关键词: 时间序列分析　截面数据分析　长短期记忆神经网络　深度学习　拆单策略

目录

1 问题重述

1.1 问题背景

证券二级市场中，机构投资者们往往每天需要买卖大量的证券，交易的金额通常又比较大，在进行大额股票交易的时候，一次大额的交易会导致目标证券价格发生异动，从而增加交易的成本。为了减少交易成本，大额交易者通常会将大单拆分成大量较小的交易，分批执行，减少对市场的冲击、降低机会成本和风险。这些复杂的次数频繁的交易操作很难仅仅通过传统的人工交易方式来完成，一是传统的人工下单会面对执行时间不足的问题，二是在执行时间不充裕时，人工很难争取到好的交易价格。因此，算法交易开始在市场上流行。

1.2 问题要求

问题一要求画出股票时间序列和截面的交易量与价格的趋势图并进行分析，但在具体分析时，我们发现仅仅只进行这些分析不足以支撑下面的建模，所以我们在第一题就对数据进行了完整的分析，以防叙述冗余。

问题二要求根据给出的历史数据预测这些股票未来连续 30 个 tick 时间的成交量和价格。

问题三要求设计一个拆单算法，目标使总成交金额尽量大，有以下三个约束：

(1) 每只股票有 4 万手的股票仓位，需要在 1 天内卖出；

(2) 在 100 个 tick 时间内（不能连续，至少相隔 6 秒以上）将股票挂单完毕；“若超过 100 个 tick 没有成交完毕，即用 100 个 tick 中的最低成交价格成交。）

(3) 每个 tick 时间的挂单量上限不超过 1 万手（<=25%）。

2 个股价格与交易量分析

2.1 高频金融数据的特点

观察题目所给数据，我们发现该数据以 3 秒为一个 tick 采样，属于高频金融数据。不同于常见的日线、月线、年线。高频金融数据有其独有的特点，具体如下：

1. 不规则交易间隔

与传统的低频观测数据 (如年数据、月数据、周数据) 相比，金融高频数据呈现出一些独有的特征。最为明显的特征便是数据记录间隔的不相等，市场交易的发生并不以相等时间间隔发生，因此所观测到的金融高频数据也是不等间隔的。从而交易间的时间持续期变得非常重要，并且可能包含了关于市场微观结构 (如交易强度) 的有用信息。

2. 离散取值

金融数据的一个非常重要的特征是价格变化是离散的，而金融高频的价格取值变化受交易规则的影响，离散取值更加集中于离散构件附近。价格的变化在不同的证券交易所设置不同的离散构件，称之为变化档位，我国证券交易所规定股价变化的最小档位为 0.01 元；在纽约证券交易所 (NYSE) 中，最小档位在 1997 年 6 月 24 日以前是 1 / 8 美元，2001 年 1 月 29 日以前是 1 / 16 美元。

3. 日内模式

金融高频数据还存在明显的日内模式，如波动率的日内 “u” 型走势。每天早上开盘和下午收盘时交易最为活跃，而中午休息时间交易较平淡，随之而来的交易间的时间间隔也呈现出日内循环模式的

特征 [1]。Mclnish 和 Wood(1992) 对价格波动率的日内模式进行了探索，发现波动率在早上开盘和下午收盘时往往较大，交易量以及买卖价差也呈现出同样的变化模式。Engle 和 Russell(1998) 对交易持续时间 (duration) 的日内模式进行了研究，也得出了类似的结论，从图形上来看变化模式类似于倒 “U” 型。

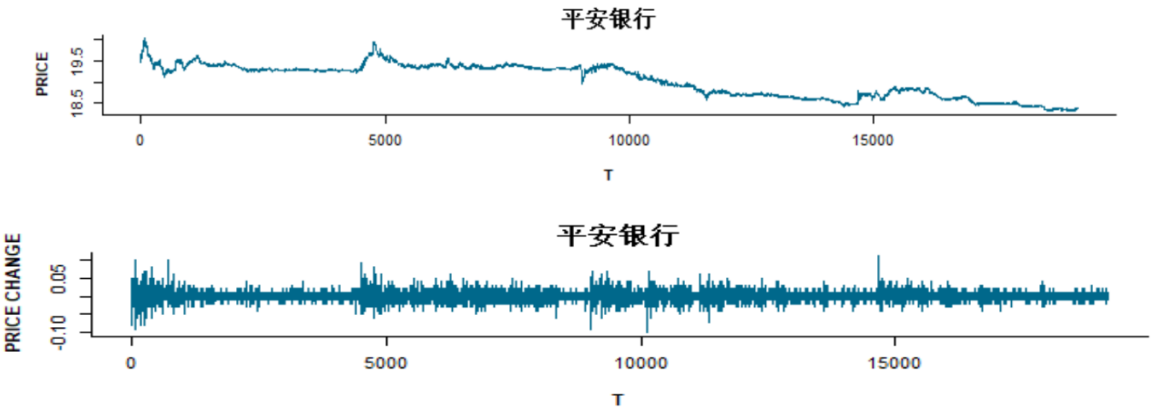
4. 自相关性

高频数据与低频数据一个非常大的区别在于高频时间序列具有非常强的自相关性。高频数据的离散取值以及买卖价差等因素是导致强自相关性的原因，还有一些因素，如一些大额交易者往往将头寸分散交易以实现最优的交易价格，这可能导致价格同方向变动从而引起序列的强自相关性。此外，还有许多其他因素导致高频数据的强自相关性。金融高频数据的特征远不止这些，数据还包含众多的信息维度，如交易的时间间隔、交易量、买卖价差等。这些不同的信息维度对于理解市场微观结构具有相当重要的作用，正是由于金融高频数据的独特特征，传统的计量分析模型在实际应用中遇到了许多问题。

2.2 时间序列分析

本小节以平安银行为例详细分析，其余股票数据主要呈现结果。

下面是平安银行 2021 年 8 月 23 日开盘至 2021 年 8 月 25 日收盘的价格时序图与价格变化率时序图：



价格整体呈现下降趋势，日内价格变动非常频繁，但基本在一个范围内波动。为探究价格变动的规律，我们绘制价格变动率 [-0.02,0.02] 内的频数表，如下所示：

价格变化	<-0.02	[-0.02,-0.01)	[-0.01,0)	0	(0,0.01]	(0.01,0.02]	>0.02
频数	986	3657	449	0	392	3494	989
百分比	0.099	0.367	0.045	0	0.039	0.351	0.099

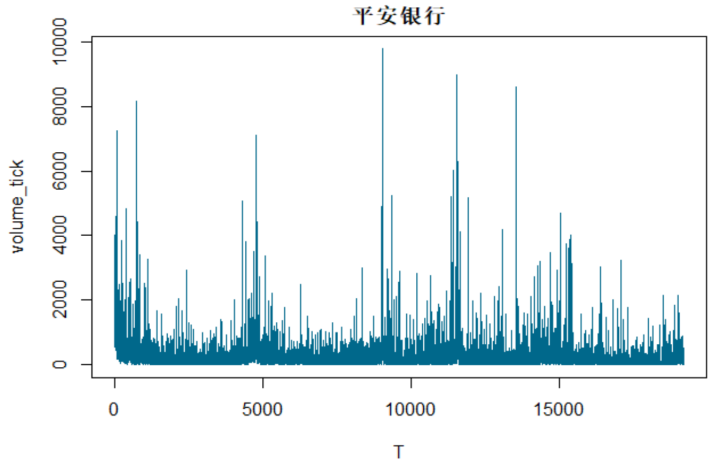
从上图，我们发现:

1) 不存在 0% 的价格变化率，说明价格变动非常频繁；

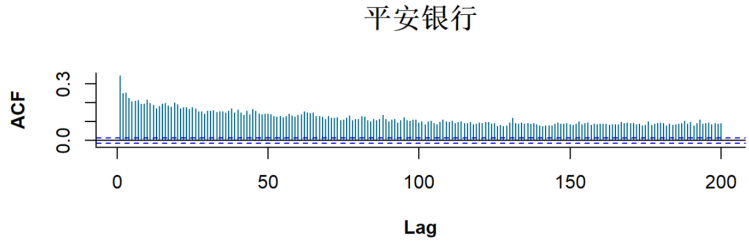
2）价格变化主要集中在 [-0.02,0.01) 和 (0.01,0.02]，有 71.8% 的盘中交易与它有关；

3）平安银行的交易价格上升和下降呈对称模式。

同样地，我们对平安银行的交易量绘图，由于受到集合竞价的影响，交易量数据在收盘时会有一次异常大值，为方便观察，对交易量进行每次分析时，均去除此异常点。结果如下：

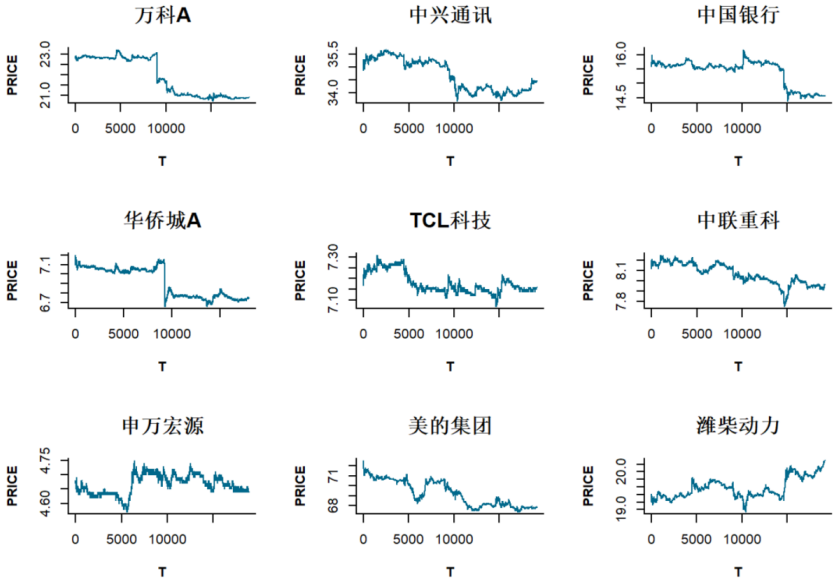


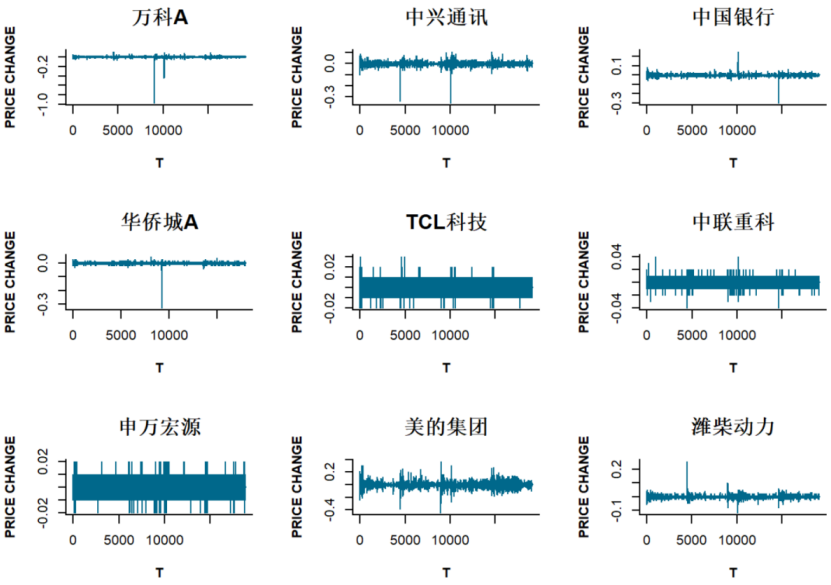
时序图存在明显的日模式，交易间的时间间隔也呈现出日内循环模式的特征，一天中大概有 15 个这样的 U 型循环。为了更好探究其循环规律，绘制其 Autocorrelation Function (ACF) 自相关函数的图像，结果如下：



从图形中可以发现，ACF 呈现明显的周期性，结合时序图，我们可得出循环大概为每 25 分钟一次，故我们将时间尺度缩小到 25 分钟，继续对每 25 分钟内交易量数据分析其时序图和 ACF，可得出周期大多集中在 2.5、5、10 分钟，这个发现对后续拆单算法交易时点的选择有很大参考意义。

通过实验证明，这种规律具有普遍性，故不再单独对其余个股进行分析，只展示其最终图像：

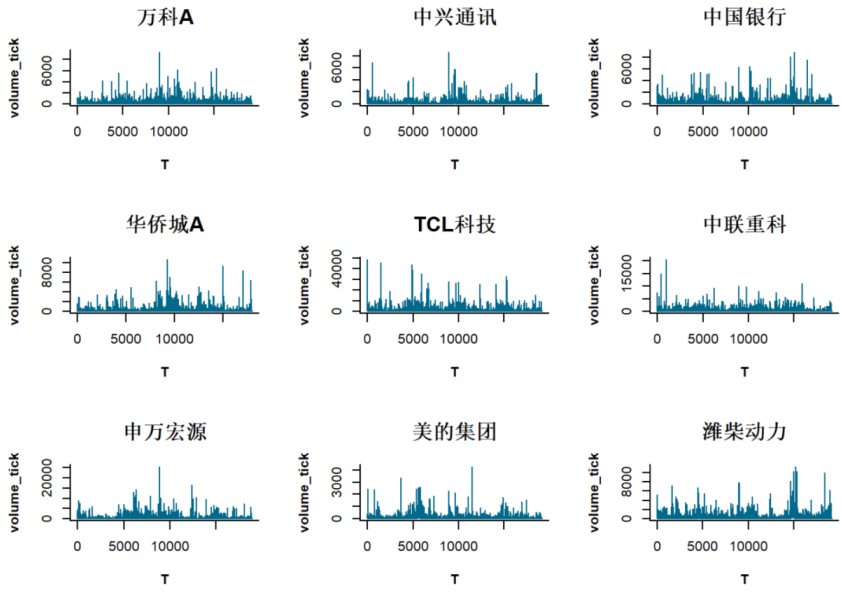


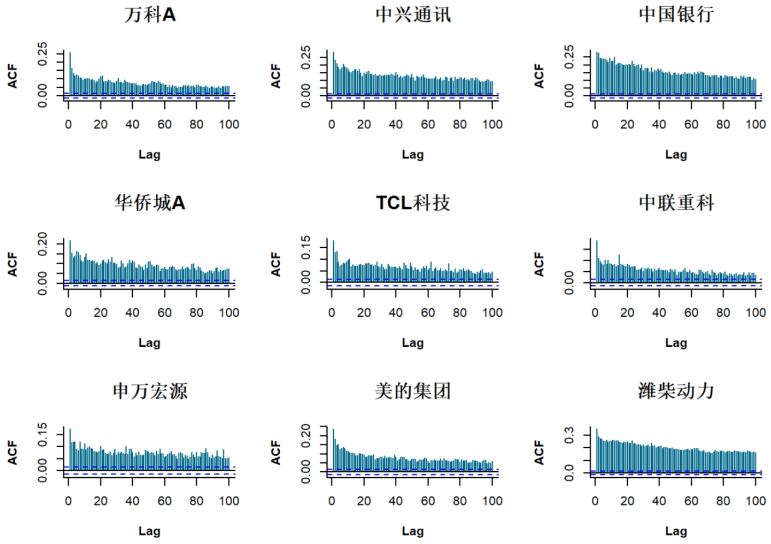


整体趋势上，除申万宏源和潍柴动力的价格整体上升外，其余股票价格均下降。注意万科 A 和华侨城存在一个跳空低开的点。

价格变化图中，万科城 A 和华侨城 A 有一个尤其大的价格变动，与价格时序图的跳跃点对应。其余股票价格变化都在一定范围内波动，与平安银行股价变化分析相似。

交易量的相关图像如下所示：

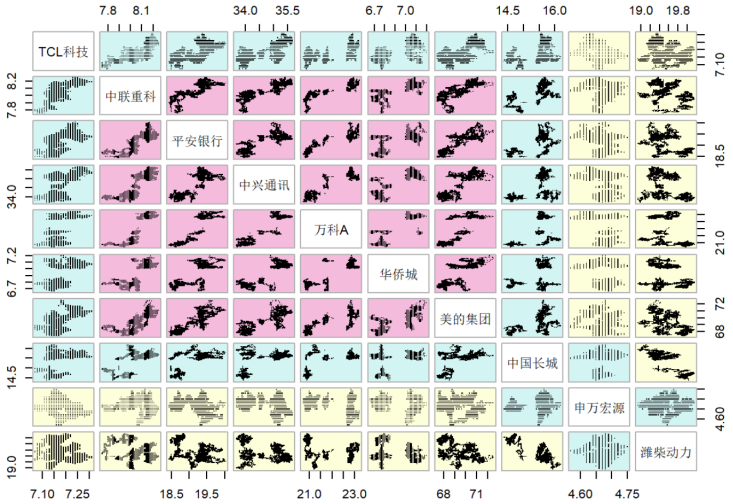




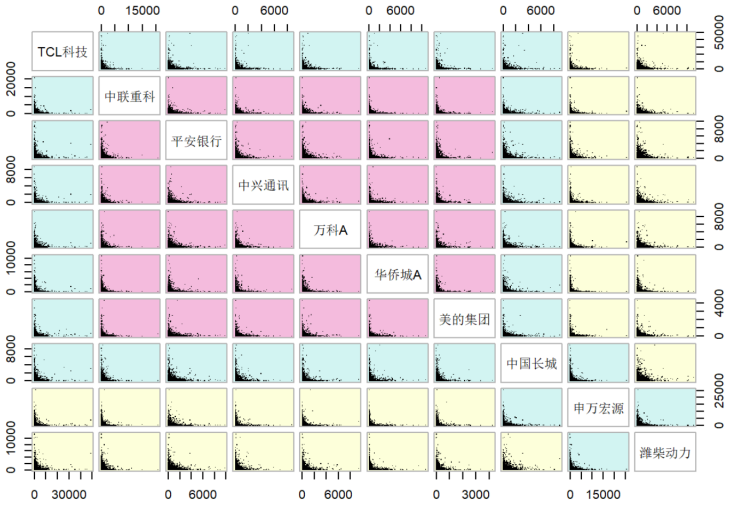
可看出交易量时序图都大致呈 U 型循环，循环数量集中在 10 上下。证明这种规律具有一定的普遍意义。

2.3 截面数据分析

利用 R 的 cpairs 函数 [2] 画出十只股票相同时点价格的散点图，并进行相关性分析；其中相关性最强为紫色，其次为黄色，最弱为蓝色。以平安银行和中联重科的紫色散点图为例，基本可以看出同一时点，若中联重科的股票价格高，那么平安银行也相应较高。



利用 R 的 cpairs 函数画出十只股票相同时点交易量的散点图，并进行相关性分析；其中相关性最强为紫色，其次为黄色，最弱为蓝色。我们可以发现散点图的走势非常相似，以平安银行和中联重科作具体分析，可看出同一时点中，股票交易量散点图呈现出聚集模式，有少部分点游离在聚集区外。这可以说明交易者倾向在相同时点进行交易，但偶尔有大单交易时会破坏这种模式。以中联重科和平安银行为例说明，少数点脱离聚集区说明在中联重科正常交易时，有大额买入或者卖出导致点游离在聚集区外。



3 价格预测

基于上述定性分析，我们将建立合适的模型来对数据进行刻画与预测

3.1 特征工程

首先将买卖序列从一至五分离开，便于后续进行数据处理。由于题目所给的数据为多位小数，考虑到 A 股市场股价最小变动单位为 0.01 元，故对其进行四舍五入运算。

根据题目所给数据，其包含信息仅有价量两种，能提供的相关性有限，故考虑对其进行特征工程处理。在处理过程中，需注意未来函数的引入，例如原始的 $volume_tick$ 变量，其代表了在该 tick 上的成交量，即 $tick = t$ 时刻的 $volume_tick$ 应发生在 t 时刻的末尾，此时的 price 已经不是未知量了。故我们需要将 $volume_tick$ 列整体下移一个单位，以避免未来函数带来的虚假结果。同样的，$tick = t$ 时刻的交易簿信息不应用于预测 t 时刻的价格，同样做下移一个单位处理。

其次，我们通过对现有数据的计算，开发新特征。考虑到该数据为超高频价格数据，我们采用短线常用的布林线（Boll）指标。布林线指标通过计算股价的标准差来计算股价的信赖区间，可以有效地判断短期股价高低估情况，公式如下：

$$\frac{\sum(price_n)}{n} \pm 2\delta$$

其中 δ 是价格的标准差

由于短期股价波动一定程度上取决于市场参与者的短时心理活动，故我们计算心理线指标作为特征之一。心理线指标主要基于市场投机者认为物极必反的原理，在一个长期的下跌序列中，参与者认为其接下来上涨的可能性比较大，反之则下跌可能性大。这种心理形成一种大众所共有的预期，而预期达到一定规模则兑现，这就是心理线的工作原理，其公式为：

$$psy = up_n/n$$

其中 up_n 是 n 日内上涨的天数。

由于布林线与心理线常用于日级行情，缺少其对于过于微观的秒级行情的研究数据。故我们尝试将两种指标的周期 n 设定为 20ticks，即实际时间一分钟参与预测。

基于上述特征工程，我们得到参与预测的变量如下表：

变量名	记号	处理方式
成交量	last_v	下移一个单位
成交额	amount_tick	下移一个单位
买 i 量	Ask_i	下移一个单位
买 i 价	Ask_pi	下移一个单位
卖 i 量	Bid_i	下移一个单位
卖 i 价	Bid_pi	下移一个单位
布林线阻力线	up	加上两倍的标准差
布林线均值	mid	均值
布林线支撑线	down	减去两倍的标准差
心理线	psy	n 取 20
上一交易日涨跌情况	trend	下移一个单位

3.2 模型评判标准建立与基线

由于此数据为高频数据，每个时间点的价格变动极小，故用传统统计拟合优度检验方法不甚有效。例如我们不建立任何模型，仅用上一个 tick 的价格作为本 tick 的预测价格，其拟合优度 R 方也高达 90.68%，故用该指标进行评判毫无意义。

此处我们选择均方误差（MSE）进行评判，由于 MSE 是一个绝对指标，其本身不具有评判优度的意义，故我们不进行建模，用上一个 tick 的价格作为本 tick 的预测价格，算得 $MSE_{NoModel} = 0.291111$，这意味着任何高于该数值的预测结果都是无效的。

3.3 多元线性回归模型

建立形如：

$$y = \sum_{i=0}^{n} \theta_i x_i + \mu$$

的多元线性回归模型，用最小二乘法作为损失函数，用梯度下降法进行迭代计算。计算得到 MSE = 0.251633，与不建模的基线 MSE 相差不大。其系数矩阵如下所示：

```
[[-6.33185652e-07  2.83588316e-09 -9.77148407e-07  5.63939445e-07
   1.08631911e-06  1.79436610e-07  1.82947232e-07  1.20682828e-06
   2.96157289e-07 -2.06854242e-07 -5.00462673e-07  2.26979498e-07
   4.85887450e-01 -2.97014828e-01  1.83942076e-01  2.62991754e-02
   4.39211470e-02  5.16200964e-01  1.47943249e-02  2.27527610e-01
  -3.43005860e-01  1.43449840e-01  2.49372265e-02 -1.35014205e-05
  -2.49642293e-02 -8.17287407e-05]]
```

不难发现，有几个变量的系数远远高出其他变量四至五个数量级。为了探究线性回归模型为什么失效，我们在损失函数中加入 L2 惩罚项，限制非 0 系数个数，所得结果如下：

```
[0.         0.         0.         0.         0.         0.
 0.         0.         0.         0.         0.         0.
 0.31038636 0.         0.         0.         0.         0.47077889
 0.         0.         0.         0.         0.         0.
 0.         0.       ]
```

其中两个非 0 项分别为买一价与卖一价。此时便很好解释为什么线性回归模型失效了。由于模型更偏向于预测一个更贴近真实值的价格，而该时刻的价格总是在买一价与卖一价附近波动，故模型偏向于给予他们差不多的权重，来得到一个较低的损失函数。这样回归得到的结果更像是随机选择一个买一价或卖一价，所以，这样的回归结果不具备泛化与预测能力。

另一方面，由于本题要求往后预测 30 个 ticks，而并未给予后 30 个 ticks 的自变量值，故普通的多元线性回归并不适合对该数据建模，接下来将探索更多可能的模型。

3.4 LSTM 深度学习模型

3.4.1 LSTM 神经网络原理与结构

由于股价的波动影响因素众多，可以采用神经网络等不确定模型进行建模。用于时间序列预测最常见的模型是 RNN 模型。

递归神经网络（RNN），是两种人工神经网络的总称。一种是时间递归神经网络（recurrent neural network），另一种是结构递归神经网络（recursive neural network）。时间递归神经网络的神经元间连接构成矩阵，而结构递归神经网络利用相似的神经网络结构递归构造更为复杂的深度网络。RNN 一般指代时间递归神经网络。而由于 RNN 递归神经网络本时刻的隐藏层信息只来源于当前输入和上一时刻的隐藏层信息，因而没有记忆功能。容易出现梯度消失问题而导致学习失败。在此基础上，LSTM 应运而生。

LSTM（长短期记忆神经网络）的网络结构相比于 RNN 复杂了很多。从微观上看，LSTM 引入了细胞状态，并使用输入门、遗忘门、输出门三种门来保持和控制信息。接下来将介绍 LSTM 的工作原理。

LSTM 最重要的概念就是：输入、输出、遗忘门；以及两个记忆：长记忆 C、短记忆 h。

首先，遗忘门结合上一隐藏层状态值 h_{t-1} 和当前输入 x_t ，通过 sigmoid 函数，决定舍弃哪些旧信息。其次，输入门和 tanh 函数决定从上一时刻隐藏层激活值 h_{t-1} 和当前输入值 x_t 中保存哪些新信息，并得到候选值 $\tilde{c}_t$。接下来，结合遗忘门和输入门进行信息的舍弃和保存，得到当前时刻的细胞状态 c_t 最后，输出门结合 tanh 函数决定 h_{t-1}、x_t、c_t 中哪些信息输出为本时刻的隐藏层状态 h_t[3]。

由上所述我们可以发现，LSTM 网络具有短期记忆的特性，对于给定 t 时刻的特征，网络将有能力向后预测 n 个时间长度的目标值，十分适合本题所要求的预测 30 个 ticks 的任务。

3.4.2 LSTM 建模与调参

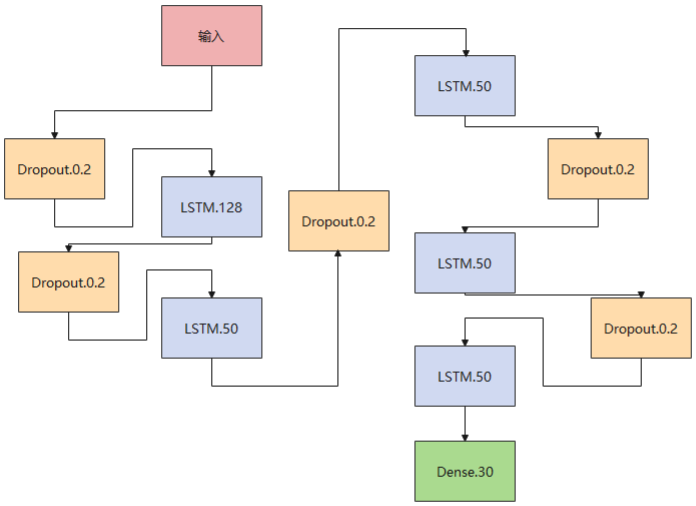
LSTM 预测的精确度依赖于众多超参数，其中起关键作用的是网络的堆叠结构、每层网络的神经元数目，学习率、学习轮次等参数。我们将通过一系列数值试验对参数进行选择。

在网络结构方面，我们采用 4 层 LSTM 与 3 层 Dropout 层交替堆叠，其中 LSTM 层用于对数据进行预测，每层的神经元数目分别为 128，50，50，50。Dropout 层则在每一轮训练中随机对一定比例的参数进行舍弃，有助于避免过拟合的情况，提高网络的预测泛化能力。舍弃的比例一般设定为 20%。

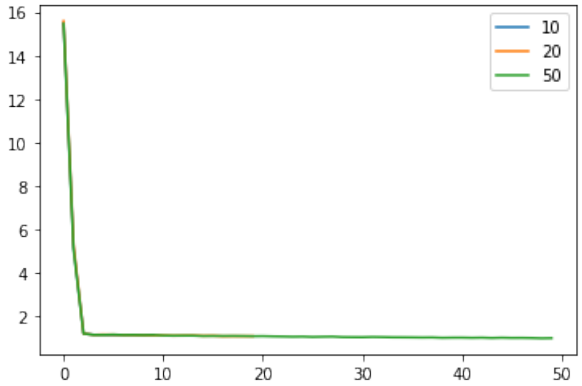
随后，将 4 层 LSTM 与 3 层 Dropout 层连接之后，再与一层全连接层相连，其中全连接层的激活函数采用线性函数。由于我们需要预测后 30 个 ticks 的情况，故全连接层的大小设定为 30，这意味着

网络将输出一个包含 30 个元素的向量，其含义为从 t 时刻向后 30 个时刻的预测值。由此，我们建立的 LSTM 深度学习网络具体结构如下所示：

Layer	(type)	Output Shape	Param
lstm_96	(LSTM)	(None, 1, 128)	79360
dropout_96	(Dropout)	(None, 1, 128)	0
lstm_97	(LSTM)	(None, 1, 50)	35800
dropout_97	(Dropout)	(None, 1, 50)	0
lstm_98	(LSTM)	(None, 1, 50)	20200
dropout_98	(Dropout)	(None, 1, 50)	0
lstm_99	(LSTM)	(None, 1, 50)	20200
dropout_99	(Dropout)	(None, 1, 50)	0
dense_24	(Dense)	(None, 1, 30)	1530

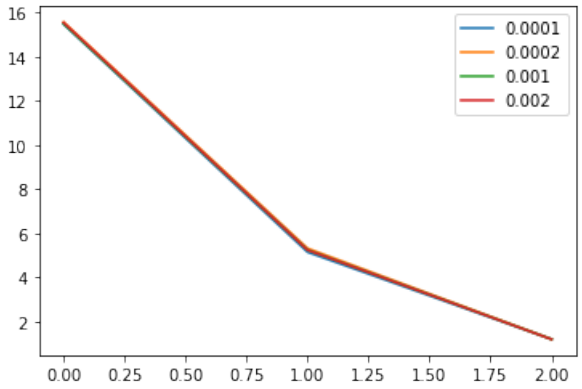


训练轮次方面，我们测试了 10 轮、20 轮、50 轮并绘制了误差曲线，结果如下：



三条曲线基本重叠，同时也可以发现，在第二轮之后虽然曲线有所下降，但是幅度不大。权衡准确率与训练时间，我们最终选择 20 轮作为训练轮次。

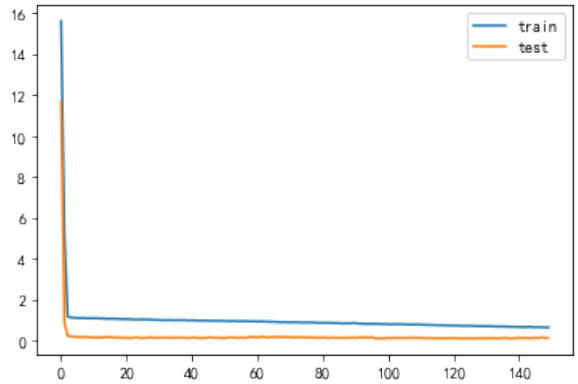
同样地，我们对学习率进行参数选择，分别测试学习率为 0.0001、0.0002、0.001、0.002，绘图如下：



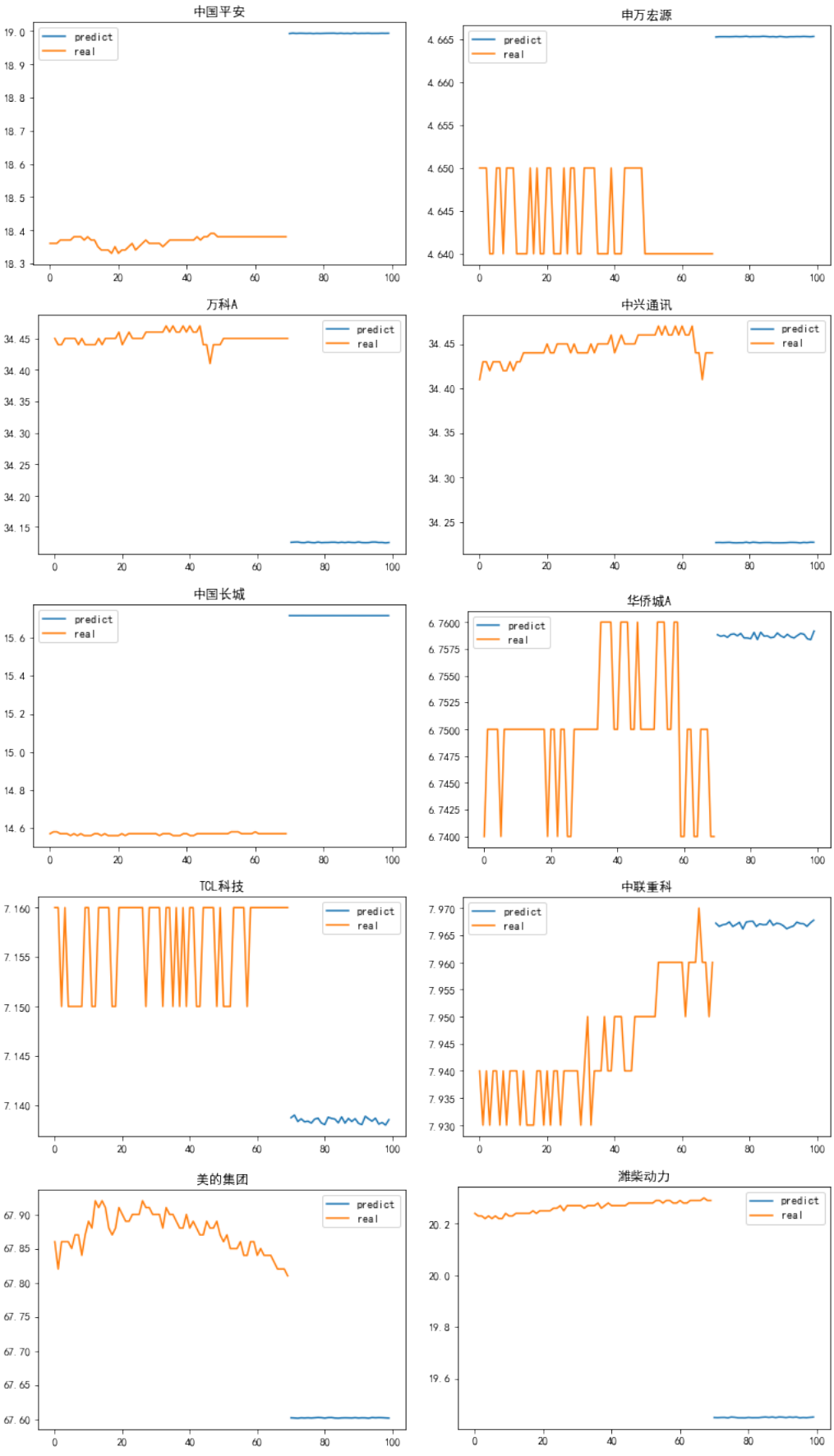
结果发现，四种学习率差别不大，我们选择在该图中收敛最快 $\alpha = 0.001$ 作为学习率。

3.4.3 网络训练与预测

我们将 10 支股票的数据送入神经网络进行训练，并采用滑动窗口法进行预测，由于每支股票训练时的误差曲线基本一致，故下图以中国平安为例，发现收敛性较好：



对每支股票训练结束后，绘制其最后 70 个 ticks 与预测 30 个 ticks 的数据进行对比，其结果如下：


中国平安
predict
real
申万宏源
predict
real
万科A
predict
real
中兴通讯
predict
real
中国长城
predict
real
华侨城A
predict
real
TCL科技
predict
real
中联重科
predict
real
美的集团
predict
real
潍柴动力
predict
real

所有 10 支股票的训练与预测结果如下表所示:

股票名	平安银行	申万宏源	万科 A	中兴通讯	中国长城
预测涨跌幅	3.34%	0.55%	-0.94%	-0.62%	7.88%
Loss	0.139	0.1444	0.1431	0.1333	0.1445
MSE	0.0521	0.0769	0.0322	0.0533	0.134
股票名	**华侨城 A**	**TCL 科技**	**中联重科**	**美的集团**	**潍柴动力**
预测涨跌幅	0.28%	-0.30%	0.10%	-0.31%	-4.13%
Loss	0.1559	0.1588	0.1385	0.1411	0.1432
MSE	0.0571	0.0437	0.139	0.0527	0.0265

同样地，我们采用类似的方法对成交量进行预测，与价格不同的是，成交量无法非常准确地被预测，但是预测其放量或是缩量相对比较准确，且放缩量对于指导交易有显著的作用。我们以放缩量为分类依据进行训练，其训练集准确率为 69.15%，测试集准确率为 65.27%，预测结果较好。

随后对未来 30 个 ticks 进行预测，因为数据所谓未来 30 个 ticks 是下一交易日的前 30 个 ticks，前文研究表明，交易量具有很强的日内循环特征，故我们将预测结果与昨日同期交易量相比来判断其放量或是缩量，其结果如下:

股票名	平安银行	申万宏源	万科 A	中兴通讯	中国长城
未来 30 个 ticks 交易量	放量	缩量	放量	放量	缩量
昨日同期	+4.86%	-11.93%	+0.55%	+12.33%	-8.75%
股票名	**华侨城 A**	**TCL 科技**	**中联重科**	**美的集团**	**潍柴动力**
未来 30 个 ticks 交易量	缩量	缩量	缩量	缩量	缩量
昨日同期	-9.28%	-0.11%	-10.84%	-4.30%	-2.71%

显然地，利用 LSTM 进行预测的 MSE 显著地好于多元线性回归与不建模的情况，故我们认为该模型可以很好地对短期股票价格与成交量进行预测。

3.4.4 LSTM 优缺点分析与改进

首先，由于 LSTM 短期记忆的特性，它可以利用现有的数据对未来多步进行预测，这是其他机器学习回归算法做不到的。其次，短期股票价格与近期的相关性高，与远期的相关性低，如果神经网络记住所有学习过的数据会导致拟合结果不佳，而 LSTM 拥有长期遗忘的特性，可以很好地解决这个问题，特别适合如本题中的高频股票数据。

然而，LSTM 的短期记忆预测能力是有限的，在结果中也可以发现，30 个 ticks 的价格趋势趋于单一，这是由于我们采用滑动窗口来预测时，如果一个窗口内的数据趋势相同，那预测值的趋势也将趋于相同。故 LSTM 能够预测出 30 个 ticks 的总体趋势，但对于 30 ticks 期间的变化趋势却预测乏力。

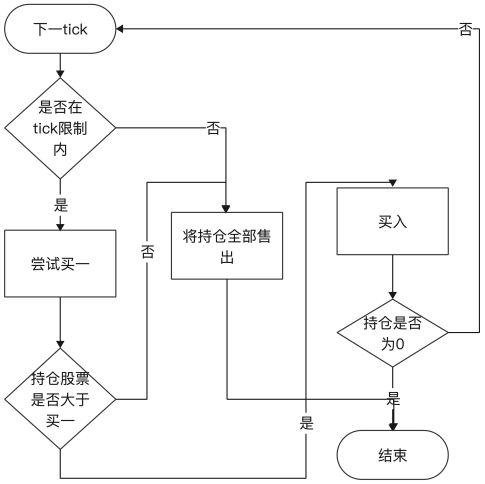
再者，LSTM 拟合的准确度依赖于特征数目与数据量，本题的数据量在 20000 条左右，并不能重复发挥其最大作用。当数据量来到千万级别时，其预测与泛化能力将进一步增强。但与此同时，其运算所耗费的时间也大大增强，以本文实验环境为例，采用 RTX 2070S 进行 GPU 加速，训练预测一支股票的耗时大约在 1s 左右。考虑到 1 tick 为三秒，这个运算速度事实上不足以在实时数据中很好地完成任务。故，该预测模型可以通过加入预训练模型，使用更强大的算力平台，进一步参数优化等方式进行算法加速，最终达到可以实时演算的程度。

4 拆单策略

由于对手盘的存在以及市场流动性有限，当我们需要成交大量订单时，通常不能在同一时间全额挂单。以卖出盘为例，全额挂单会导致在该价格位形成强力压力位，使得后续成交概率变小，且影响成交收益。

此外，单笔大单卖出会导致操作过于明显，容易在交易市场形成恐慌，导致其他散户的从众心理，最终导致股价异动。

为避免种种不利因素，我们需要对大额买单进行拆分，当今市场最常用的就是将大单平均拆分成多个小单，基础策略如下图所示：

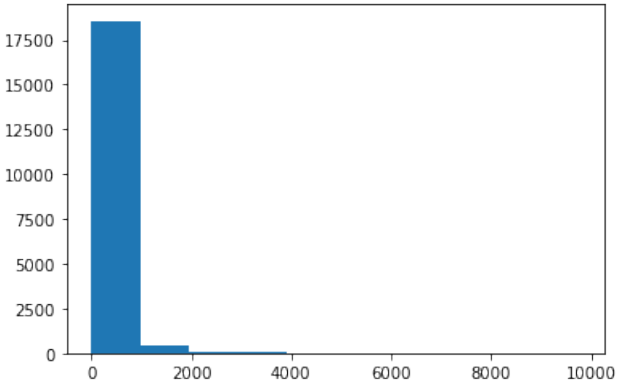


故下文将在该简单策略的基础上进一步优化。

4.1 简单分隔的拆单策略

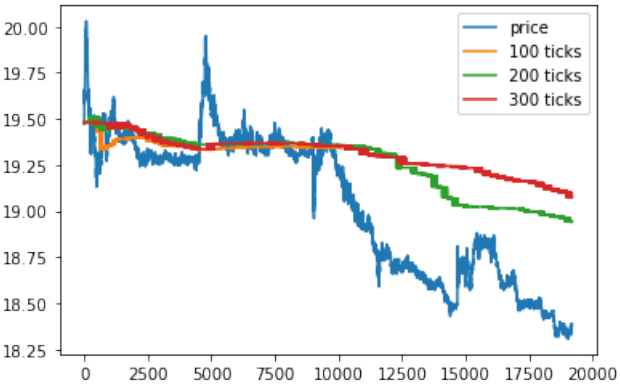
为了降低大单买入（一个大单拆分后仍很大）对市场价格的冲击，我们需要找到市场本身交易量大的时间点进行买入 [5]。基于上述的时间序列分析，我们发现交易量的分布大致具有周期性，故一方面，我们需要识别到何时做第一笔买入，以及取怎么样的买入周期。

观察多只股票的前几个交易点的交易量，我们发现在这在这一时间段中属于较高的交易量。下图是平安银行的交易量分布图，其 2021-08-23 09:30:03 的交易量为 3755，大约为 99% 分位点，我们可以认为其属于一个交易量高峰。



从理论分析，刚开盘的交易量也应该较高。一方面，开盘价格是 15 分钟集合竞价的结果，这个价格可能并不符合许多投资者的预期，故一开盘他们就会以预期价格交易；另一方面，许多中长线量化投资基金也会选择在刚开盘的时间点进行交易，避免一些流动性风险而导致的交易失败。

综上，我们选择第一个 tick 作为交易起点，为了保证市场流动性与避免价格冲击，我们选择一次交易以买一价格的 90% 卖出。改变不同交易周期，我们得到如下结果：

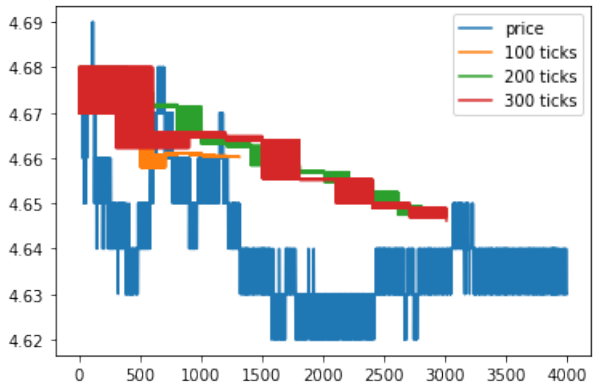


我们发现，以 100 个 ticks 为周期与 300 个 ticks 相差不大。然而，观察其具体数据：

周期	**100ticks**	**200ticks**	**300ticks**
卖出均价	19.31	18.8	18.69
剩余手数	7291	9602	21125

发现如果以 300 ticks 为周期，40000 手将剩余 21125 手没有卖出，故以 100 ticks 为周期更为合理。

然而，这种策略虽然能一定程度上保证市场流动性充足，可是最终卖出成本的多少很大程度上取决于股票自己的走势。例如申万宏源在这段时期内的价格呈涨势，且流动性充足，套用该策略的结果如下所示：



周期	100ticks	200ticks	300ticks
卖出均价	4.64	4.65	4.66
剩余手数	0	0	0

此时，所有股票都可以在规定时期内卖出，且发现，周期越长，卖出均价越高。这是因为该股票的价格走势逐渐变高，故长周期会让更多的股票以更高价格卖出。所以，在保证流动性与卖出完成度的情况下，我们需要依据上文所作出的预测对每一笔卖出的股票数量进行加权调整，以在保证流动性和降低市场冲击的基础上提高卖出的收益。

4.2 基于价格预测的加权拆单策略

4.2.1 同交易周期的拆单策略

我们基于前文的价格预测，将待拆单股票的总体趋势分为上涨和下跌两类，分类如下：

上涨	下跌
申万宏源	平安银行
中国长城	万科 A
华侨城 A	中兴通讯
	TCL 科技
	中联重科
	美的集团
	潍柴动力

由于我们无法精确预测个股何时价格最高或最低，故我们采取均分的方式，将交易期均分为 3 份，对于预测上涨的股票，前 1/3 给予原交易量的 $1+\beta$ 倍卖出，中间以原交易量卖出，后 1/3 以 $1-\beta$ 倍交易量卖出。对于 β 的取值我们选取中国平安进行如下实验：

处理方式	**不加权**	**1.2 倍**	**1.4 倍**
卖出均价	19.31	19.33	19.34
剩余手数	7291	2684	0

综合考虑收益与市场流动性，我们选择 $\beta = 0.2$。

随后将该策略应用于 10 支股票，所得结果如下：

股票名	**平安银行**	**申万宏源**	**万科 A**	**中兴通讯**	**中国长城**
成交均价	19.31	4.66	22.39	34.79	15.57
时期均价	19.03	4.65	21.89	34.69	15.41
剩余手数	7291	0	13820	24149	13566
股票名	**华侨城 A**	**TCL 科技**	**中联重科**	**美的集团**	**潍柴动力**
成交均价	7.06	7.23	8.19	68.64	19.51
时期均价	6.91	7.18	8.07	69.26	19.57
剩余手数	0	0	0	34314	5084

结果表明，该策略可以很好地提高收益与保持市场流动性。但同时要指出的是，我们发现价格大于 10 元的股票大多未在此策略下全部交易成功，这是因为本题给的单位是手而非元，个股单价越高事实上卖出时需要撮合的金额就越大，同时，撮合机会就越小。100ticks 的间隔对于一些高价股而言还是太长了，故接下来考虑基于价格再对交易间隔进行加权。

4.2.2 基于价格加权交易周期的拆单策略

由上述结果可知，个股单价越高，剩余的股数就越多。故我们基于个股的价格给予交易周期一个削减权重，这个权重由如下公式表示：

$$w_i = [\frac{100}{p_i - 10}] \quad p_i > 10$$

其中 p_i 表示第 i 支股票的第一个 ticks 的价格。

基于该权重，我们重新进行实验，结果如下：

股票名	平安银行	申万宏源	万科 A	中兴通讯	中国长城
成交均价	19.33	4.66	22.76	34.79	15.62
时期均价	19.03	4.65	21.89	34.69	15.41
剩余手数	6324	0	13820	21999	8906
股票名	华侨城 A	TCL 科技	中联重科	美的集团	潍柴动力
成交均价	7.06	7.23	8.19	71.08	19.52
时期均价	6.91	7.18	8.07	69.26	19.57
剩余手数	0	0	0	31743	1342

从表中可以发现，股票未售出的情况有所改善，但仍存在卖不完的情况。

其原因是，本题要求的 4 万手相较于某些个股的日成交量占比过大。以美的集团为例，其 2021 年 8 月 23 日一整日的成交量不过 24 万手，待售出的 4 万手约占其 17%（TCL 科技持仓约占日成交量 1%）。而由于本策略更注重于减少对市场价格的冲击，故如此大体量的持仓着实很难在一日内卖完。

5 模型改进与结论

本文从股票数据的时空特征入手，对其时序特征与截面特征进行分析。随后通过理论与实证证明了多元回归不适用于大规模高频数据，于是建立了深度学习模型对数据进行预测，基于 MSE 建立的评判机制表面模型有效性较强。基于上述预测，开发了价格加权交易周期的拆单策略，对题设问题进行了很好回应。遗憾的是，仍有个别股票如美的集团无法在一日 100tick 内完成卖出任务，需要后续进一步研究。

本文主要基于传统时间序列模型与前沿深度学习进行分析预测，在研究过程中不难发现，两种算法都有其优势所在。例如时间序列分析的传统模型具有较强的可解释性，容易找出从参数到现实的联系，然而，当其面对大量高频数据时显得乏力，深度学习网络则具有强大的拟合与泛化能力，理论上它可以拟合任何连续函数。然而，例如本文所建立的 LSTM 模型拥有 15 万参数，但是无法将这些参数与现实建立联系。故本文将两种模型相结合，最后得到既具有解释力，又具有相当程度拟合精度的模型。

然而，囿于本文数据量较小，特征量也比较少。如果数据量能够以 tick 级达到年度数据，再加入诸如市盈率、市净率、ROE 等财务指标，将更有利于预测的精度与模型的可解释性。

为了提高算法的确定性以及提高成交率，本文在挂单交易中只考虑了主动市价单的情况，并未考虑被动挂单。关于被动单对收益率、成交率以及市场冲击的影响还有待后续研究。

参考文献

[1] 镇磊. 基于高频数据处理方法对 A 股算法交易优化决策的量化分析研究 [D]. 中国科学技术大学,2010.

[2] R Core Team (2018). R: A language and environment for statistical computing. R Foundation for Statistical Computing, Vienna, Austria.

[3] 杨青. 王晨蔚. 基于深度学习 LSTM 神经网络的全球股票指数预测研究 [J]. 统计研究,2019,36(03):65-77.

[4] 李静. 徐路路. 基于机器学习算法的研究热点趋势预测模型对比与分析——BP 神经网络、支持向量机与 LSTM 模型 [J]. 现代情报,2019,39(04):23-33.

[5] 敖薇. 中国证券市场动态 VWAP 策略研究 [D]. 上海交通大学,2013.

6 附录

```
####################数据读取##########################

import pandas as pd
data = pd.read_excel("data.xlsx", sheet_name = 1)
import matplotlib.pyplot as plt
import seaborn as sns
plt.plot(data["price"]) ## 画一下价格
plt.show()
## 处理一下买卖价量，由于题目给的是字符型，用literal_eval函数转换
from ast import literal_eval
import numpy as np
bid_price_list = []
bid_volume_list = []
ask_price_list = []
ask_volume_list = []
for i in range(len(data)):
    temp = (literal_eval(data.iloc[i]["bid_price"]))
    for j in range(len(temp)):
        temp[j] = round(temp[j],2)
    bid_price_list.append(temp)

    temp = (literal_eval(data.iloc[i]["ask_price"]))
    for j in range(len(temp)):
        temp[j] = round(temp[j],2)
    ask_price_list.append(temp)

    temp = (literal_eval(data.iloc[i]["bid_volume"]))
    for j in range(len(temp)):
        temp[j] = round(temp[j],0)
    bid_volume_list.append(temp)

    temp = (literal_eval(data.iloc[i]["ask_volume"]))
    for j in range(len(temp)):
        temp[j] = round(temp[j],0)
    ask_volume_list.append(temp)

temp1 = [0]
temp2 = [0]
for i in range(len(ask_volume_list)):
    temp1.append(sum(ask_volume_list[i]))
    temp2.append(sum(bid_volume_list[i]))
temp1.pop(-1)
temp2.pop(-1)
data["total_ask"] = temp1
data["total_bid"] = temp2
volume_tick = data["volume_tick"]
stock_price = data["price"]
```

```
trend = data["trend"]

####################特征工程#########################

volume_tick_x = np.array(volume_tick.iloc[0:4485]).reshape(-1, 1)
trend_x = np.array(trend.iloc[0:4485]).reshape(-1, 1)
rise_y = np.array(data["price"].iloc[0:4485]).reshape(-1, 1)
X = data[["last_v","amount_tick", "price"]].iloc[0:4485]

def add_price(X, length):
    X["ask_1"] = np.array(ask_volume_list).T[0][0:length]
    X["ask_2"] = np.array(ask_volume_list).T[1][0:length]
    X["ask_3"] = np.array(ask_volume_list).T[2][0:length]
    X["ask_4"] = np.array(ask_volume_list).T[3][0:length]
    X["ask_5"] = np.array(ask_volume_list).T[4][0:length]
    X["bid_1"] = np.array(bid_volume_list).T[0][0:length]
    X["bid_2"] = np.array(bid_volume_list).T[1][0:length]
    X["bid_3"] = np.array(bid_volume_list).T[2][0:length]
    X["bid_4"] = np.array(bid_volume_list).T[3][0:length]
    X["bid_5"] = np.array(bid_volume_list).T[4][0:length]
    X["ask_p1"] = np.array(ask_price_list).T[0][0:length]
    X["ask_p2"] = np.array(ask_price_list).T[1][0:length]
    X["ask_p3"] = np.array(ask_price_list).T[2][0:length]
    X["ask_p4"] = np.array(ask_price_list).T[3][0:length]
    X["ask_p5"] = np.array(ask_price_list).T[4][0:length]
    X["bid_p1"] = np.array(bid_price_list).T[0][0:length]
    X["bid_p2"] = np.array(bid_price_list).T[1][0:length]
    X["bid_p3"] = np.array(bid_price_list).T[2][0:length]
    X["bid_p4"] = np.array(bid_price_list).T[3][0:length]
    X["bid_p5"] = np.array(bid_price_list).T[4][0:length]

    ma = [0 for i in range(20)]
    up = [0 for i in range(20)]
    down = [0 for i in range(20)]
    for i in range(20, len(X)):
        temp_std = 0
        ma.append(sum(X['price'].iloc[i-20:i]) / 20)
        for j in range(20):
            temp_std += (X['price'].iloc[i+j-20]-ma[i])**2
        up.append(ma[i]+(temp_std)/20**0.5*2)
        down.append(ma[i]-(temp_std)/20**0.5*2)
    plt.plot(up[20:50])
    plt.plot(down[20:50])
    plt.plot(ma[20:50])
    X["up"] = up[:length]
    X["mid"] = ma[:length]
    X["down"] = down[:length]
    temp_up = 0
    psy = [0 for i in range(20)]

    for i in range(20, len(X)):
        temp_up = 0
        for j in range(20):
            if X["price"].iloc[j+i-20]-X["price"].iloc[j+i-19] > 0:
                temp_up += 1
        psy.append(temp_up/20*100)
    X["psy"] = psy[:length]
```

```
    return X

data = add_price(data, len(data["price"]))
X = add_price(X, 4485)

###由于bull线的加入，前20个数据为0！！！！

####################线性模型##########################

## 基于sklearn 库的一堆模型试验
from sklearn.linear_model import RidgeClassifier
from sklearn.svm import SVC
from imblearn.under_sampling import RandomUnderSampler
from sklearn.neural_network import MLPClassifier
from sklearn.linear_model import LogisticRegression
from sklearn.linear_model import RidgeClassifierCV
from sklearn.linear_model import LinearRegression
from sklearn.linear_model import LassoLarsCV
from sklearn.neural_network import MLPRegressor
from sklearn.preprocessing import StandardScaler
from sklearn.tree import DecisionTreeRegressor
from sklearn.svm import SVR
from sklearn.ensemble import RandomForestRegressor

#sca = StandardScaler()
#sca.fit(X)
#X = sca.transform(X) ##标准化，试过了标准化不影响结果
#print(X)
#rus = RandomUnderSampler(random_state=0)
#X, rise_y = rus.fit_resample(X, rise_y)

#model = RidgeClassifierCV(class_weight = {1:0.4,0:0.6})

#model = LogisticRegression()
#model = LinearRegression()
#model = LassoLarsCV()
#model = MLPRegressor(activation = 'tanh', alpha = 0.01, max_iter = 5000, verbose = True,
                                                    hidden_layer_sizes = (256),random_state=1,tol = 1e-7
                                                    )
#model = DecisionTreeRegressor()
model = RandomForestRegressor(oob_score = True,verbose=3,n_estimators=200)
#model = LinearRegression()
#model = SVR()


model.fit(X, rise_y)
print(model.score(X, rise_y))
temp_predict = model.predict(X)
plt.plot(rise_y[:400])
plt.plot(temp_predict[:400], "g")
#s_mat(temp_predict, rise_y)
#print(model.coef_)
predict_lin = model.predict(X)
print('Baseline RMSE: %.9f' % mean_squared_error(rise_y, predict_lin)**0.5)

ty=[0]
for i in range(len(rise_y)):
```

```
    ty.append((rise_y)[i])
ty.pop(-1)
print('Baseline RMSE: %.9f' % mean_squared_error(rise_y, ty)**0.5)
####################LSTM############################
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
from sklearn.preprocessing import MinMaxScaler
from keras.models import Sequential
from keras.layers import LSTM, Dense, Activation
from sklearn.metrics import mean_squared_error
from keras.models import Sequential
from keras.layers import Dense
from keras.layers import LSTM
from sklearn.preprocessing import StandardScaler
from keras import optimizers
from keras.layers import Dropout


sca = StandardScaler()
sca.fit(X)
X = sca.transform(X) ##标准化，试过了标准化不影响结果
rise_y = np.array(data["price"].iloc[0:4485]).reshape(-1, 1)
#rise_y = np.array(data["volume_tick"].iloc[0:4485]).reshape(-1, 1)


data_X, data_Y = X , rise_y
train_size = int(len(data_X) * 0.7)
test_size = len(data_X) - train_size
train_X = data_X[:train_size]
train_Y = data_Y[:train_size]
test_X = data_X[train_size:]
test_Y = data_Y[train_size:]

train_X = np.array(train_X)
test_X = np.array(test_X)
train_X = train_X.reshape((train_X.shape[0], 1, train_X.shape[1]))
test_X = test_X.reshape((test_X.shape[0], 1, test_X.shape[1]))
print(len(data_X))
print(len(data_Y))

# 设计模型
model = Sequential()

model.add(LSTM(128, input_shape=(train_X.shape[1], train_X.shape[2]), return_sequences=True))
model.add(Dropout(0.2))
model.add(LSTM(50, return_sequences=True))
model.add(Dropout(0.2))
model.add(LSTM(50, return_sequences=True))
model.add(Dropout(0.2))
model.add(LSTM(50, return_sequences=True))
model.add(Dropout(0.2))
model.add(Dense(30))
#Adamax = optimizers.Adamax(learning_rate=0.001, beta_1=0.9, beta_2=0.999, epsilon=None, decay=0.0)
#Adam = optimizers.Adam(lr=0.00005, beta_1=0.9, beta_2=0.999, epsilon=None, decay=0.0, amsgrad=False)
model.compile(loss='mae', optimizer='Adam')
# 拟合模型
```

```
history = model.fit(train_X, train_Y, epochs=150, batch_size=75, validation_data=(test_X, test_Y),
                    verbose=1, shuffle=False)
# 绘制损失趋势线
plt.plot(history.history['loss'], label='train')
plt.plot(history.history['val_loss'], label='test')
plt.legend()
plt.show()

plt.rcParams['font.sans-serif'] = ['SimHei']
%matplotlib inline
predict_list = model.predict(test_X)
predict_list = predict_list[:,0]
plt.plot(range(70,100),predict_list[-1])
plt.plot(range(70),test_Y[-70:])
plt.legend(["predict","real"])
plt.title("平安银行")
#print('Baseline RMSE: %.9f' % mean_squared_error(test_Y, predict_list)**0.5)
####################实测与回测模块##########################
for k in range(1):
    data = pd.read_excel("data.xlsx", sheet_name = k)
    time_lists = data["time_tick"]
    ans_lists = []
    ## 处理一下买卖价量，由于题目给的是字符型，用literal_eval函数转换
    from ast import literal_eval
    import numpy as np
    bid_price_list = []
    bid_volume_list = []
    ask_price_list = []
    ask_volume_list = []
    for i in range(len(data)):
        temp = (literal_eval(data.iloc[i]["bid_price"]))
        for j in range(len(temp)):
            temp[j] = round(temp[j],2)
        bid_price_list.append(temp)

        temp = (literal_eval(data.iloc[i]["ask_price"]))
        for j in range(len(temp)):
            temp[j] = round(temp[j],2)
        ask_price_list.append(temp)

        temp = (literal_eval(data.iloc[i]["bid_volume"]))
        for j in range(len(temp)):
            temp[j] = round(temp[j],0)
        bid_volume_list.append(temp)

        temp = (literal_eval(data.iloc[i]["ask_volume"]))
        for j in range(len(temp)):
            temp[j] = round(temp[j],0)
        ask_volume_list.append(temp)

    temp1 = []
    temp2 = []
    for i in range(len(ask_volume_list)):
        temp1.append(sum(ask_volume_list[i]))
        temp2.append(sum(bid_volume_list[i]))
    data["total_ask"] = temp1
    data["total_bid"] = temp2
```

```
volume_tick = data["volume_tick"]
stock_price = data["price"]
#trend = data["trend"]
buy_weight = []
for i in range(0,int(len(stock_price)/3)):
    buy_weight.append(1-lists_up[k]*0.2)
for i in range(0,int(len(stock_price)/3)):
    buy_weight.append(1)
for i in range(0,int(len(stock_price)/3)):
    buy_weight.append(1+lists_up[k]*0.2)
for i in range(len(stock_price)-len(buy_weight)):
    buy_weight.append(1+lists_up[k]*0.2)

if stock_price[0] > 10:
    fre = int(100/(stock_price[0]-10))
else:
    fre = 100

buy_flag = []
for i in range(0,len(stock_price), fre):
    buy_flag.append(1)
    for j in range(fre):
        buy_flag.append(0)
#plt.plot(stock_price)
#plt.plot(order_in_buy_one(0,buy_flag)[1])
#plt.plot(order_in_buy_one(0,buy_flag,buy_weight)[1])
#plt.legend(["price","no weight","with weight"])
ans_lists = order_in_buy_one(0,buy_flag)
ans_output = pd.DataFrame({'挂单时间':ans_lists[0],"报价":ans_lists[1],"成交手数":ans_lists[2],"成交
                                                    金额":ans_lists[3],"股票名":[sheet_name[k] for _
                                                     in range(len(ans_lists[0]))]})
```