算法报告目录

[1 数据预处理 1](#_Toc24686)

[1.1 训练和验证数据预处理 1](#_Toc27885)

[1.1.1 概述 1](#_Toc24764)

[1.1.2 数据加载与内存优化 1](#_Toc22298)

[1.1.3 数据基本信息查看 2](#_Toc25994)

[1.1.4 数据合并与筛选 2](#_Toc28880)

[1.1.5 数据分组 2](#_Toc18863)

[1.1.6 特征与标签的对数变换和重构 3](#_Toc13231)

[1.1.7 数据随机化与集合划分 3](#_Toc10700)

[1.1.8 数据封装 4](#_Toc13646)

[1.2 测试数据预处理 4](#_Toc27306)

[2 建模逻辑 4](#_Toc7837)

[2.1 网络结构设计 4](#_Toc13217)

[2.1.1 LSTM层： 4](#_Toc6957)

[2.1.2 1D卷积层： 5](#_Toc1322)

[2.1.3 2D卷积层： 5](#_Toc26625)

[2.1.4 全连接层： 5](#_Toc5382)

[2.2 向前传播设计 6](#_Toc9613)

[3 训练细节 7](#_Toc9379)

[3.1 过拟合优化 7](#_Toc28433)

[3.2 输出的广度 7](#_Toc27558)

[3.3 学习方式选择 7](#_Toc2424)

[3.4 Loss函数的选择 7](#_Toc16322)

[4 预测 8](#_Toc20051)

[5 投资策略 9](#_Toc11330)

算法报告

本算法报告详细描述了一种基于深度学习的股票预测方法。报告首先介绍了数据预处理和特征工程的步骤，然后深入解析了所使用的LSTM网络、卷积网络和全连接网络的混合网络结构模型。在模型训练方面，本团队选择了Kaiming Normal函数进行权重初始化，使用AdamW优化器，并采用了GaussianNLLLoss作为损失函数。本算法报告还包括了预测阶段的代码和逻辑，特别是如何处理测试数据以确保有效的预测。最后，本算法报告介绍了一个基于模型预测结果进行权重分配的投资策略，并考虑了交易成本和数据缺陷等实际问题。

1 数据预处理与特征工程

1.1 训练和验证数据预处理

在量化模型的构建过程中，数据预处理是非常关键的一步。本项目中使用PaddlePaddle作为深度学习框架，并通过其提供的`Dataset`和`DataLoader`进行数据封装和加载。本节详细描述了如何处理训练和验证数据，以便为后续模型训练做好准备。

1.1.1 数据加载与内存优化

首先，我们从不同的CSV文件中读取各类因子数据和标签数据。考虑到数据集可能会非常大，因此编写了一个`reduce\_mem\_usage`函数，通过修改读取数据的数据类型，来减少DataFrame的内存占用。

1. **if** c\_min > np.iinfo(np.int8).min **and** c\_max < np.iinfo(np.int8).max:
2. df[col] = df[col].astype(np.int8)

1.1.2 数据基本信息查看

通过导出每个数据集中的最大值与最小值，预估应该对数据作何种处理。由最大最小值显示，数据集中的最大值与最小数大小、数量级上均差异较大，这一点将纳入之后的数据处理中考虑。

1. max\_min\_for0 = []
2. **for** i in train\_factor\_set\_0.columns:
3. **if** i != 'date' and i != 'asset':
4. max\_min\_for0.append(train\_factor\_set\_0[i].max())
5. max\_min\_for0.append(train\_factor\_set\_0[i].min())

1.1.3 数据合并与筛选

加载完所有数据集后，通过观察数据集形状发现因子集2在较早时间戳上存在数据缺失，经对比检验，之后的数据内容基本不存在缺失。同时基于金融市场动态且高度复杂的考虑，从市场环境来看，我们认为赛方提供的长达十年数据集可能包含了市场的许多个不同状态或者说版本的信息，如果将十年数据全部作为训练数据，可能导致模型学习失败，或者导致训练出来的模型只适合用作长期价值投资。因此，我们根据日期对数据进行筛选并重置索引，将时间段缩放到1年，以提高模型精度（由于比赛时间有限，只能采取这种简单粗暴的方式，实际上我们有考虑通过Embedding层将时间信息和股票代码信息嵌入特征集，但由于工程量较大，没来得及实现）。然后，将所有相关数据集（包括因子数据和标签数据）进行水平合并，以便于样本的特征数据对齐，进行样本维度调整，创造新的样本。

1. train\_stack\_data = pd.concat([
2. train\_factor\_set\_0,
3. train\_factor\_set\_1.drop(['date', 'asset'], axis=1),
4. train\_factor\_set\_2.drop(['date', 'asset'], axis=1),
5. train\_risk\_set.drop(['date', 'asset'], axis=1),
6. train\_label.drop(['date', 'asset'], axis=1)  # 重置索引，以便于连接
7. ], axis=1)

1.1.4 数据分组

接下来，由于赛方提供的数据集是全市场全时间段的数据，导致如果将时间序列作为最外层维度的依据，每一时间维度上的变量维度将不稳定，因为股票在每个时间戳上都有可能入市或退市。故放弃将数据依据时间序列划分维度，转以股票为依据作分组，将数据按照'asset'字段以股票代码为依据进行分组。

1. groups = train\_stack\_data.groupby('asset')

1.1.5 特征与标签的对数变换和重构

根据之前的数据集基本信息可知，数据的分布范围较广，数据的绝对值数量级差异巨大，不利于模型进行训练，但由于股票池动态不定，部分因子特征数据值波动范围较大，不适宜直接进行一些归一化处理，故在数据分组后，对数据集进行对数对称变换，缩小较大的数据，放大较小的数据，将数据集分布集中化，以便模型训练。

1. one\_sample = np.sign(one\_sample) \* np.log1p(np.abs(one\_sample))

对于因子数据集0，我们发现前5列数据主要为股票的价格数据，是股票最原始、基础的数据。依据金融知识可知，股票的价格数据通常非平稳，故在进行对数化处理后，对股票价格数据进行差分处理，构造平稳的价格变动（收益）序列。

1. one\_sample\_price = one\_sample.iloc[:, 0:5].diff()
2. one\_sample\_price = one\_sample\_price.drop(0).reset\_index(drop=True)

在这个步骤中，通过之前观察的label的分布与取值范围，假设label是相应股票和时间戳的周收益率，我们将一个月的数据（在市场中按交易日计时，一年由252天，一个月有21天）打包成一个样本，将这个样本所含的特征数据作为相应的一个label的对应特征。通过遍历每一个股票（asset）相关的特征数据集，迭代生成了训练数据和标签，将整个以’asset’为依据的groups打散成了样本切割自不同股票的样本集，之后通过数组化将样本集列整理成维度正确的数组。

同时，在这里由于我们采用的损失函数是GaussianNLLLoss高斯负对数似然损失，所以在这里我们还需要提取每个股票对应的的label方差。

1. **for** idx **in** range(one\_sample.shape[0]):
2. **if** idx > 20:
3. feat\_data = one\_sample[idx-21:idx]
4. feat\_label = one\_label[idx]
5. train\_dataset\_pre.append(feat\_data)
6. train\_label\_pre.append(feat\_label)
7. train\_label\_variance\_pre.append(one\_label\_variance)

1.1.6 数据随机化与集合划分

在所有预处理完成后，通过随机索引，将样本顺序随机化，提高模型训练的广度。

1. permuted\_indices = np.random.permutation(train\_dataset\_pre.shape[0])
2. train\_dataset\_pre = train\_dataset\_pre[permuted\_indices, :, :]
3. train\_label\_pre = train\_label\_pre[permuted\_indices]
4. train\_label\_variance\_pre = train\_label\_variance\_pre[permuted\_indices]

1.1.7 数据封装

最后，使用自定义的`CustomDataset`类将预处理后的数据封装为`Dataset`对象，以便后续使用PaddlePaddle进行模型训练。

1. train\_dataset = CustomDataset(
2. train\_dataset\_pre[:778280],
3. train\_label\_pre[:778280],
4. train\_label\_variance\_pre[:778280]
5. )
7. val\_dataset = CustomDataset(
8. train\_dataset\_pre[778280:],
9. train\_label\_pre[778280:],
10. train\_label\_variance\_pre[:778280]
11. )

至此，训练和验证数据的预处理工作完成，为后续的模型训练和验证奠定了基础。

1.2 测试数据预处理

对初赛、复赛使用的测试数据集的处理方式与对训练数据集、验证数据集的处理方式基本相同。仅在特征与标签的重构上存在差异，即没有对测试数据集提取每个股票的label以及方差（因为使用模型进行预测时仅需要输入相同的因子特征即可）。

2 建模逻辑

基于比赛数据集的数据量大、因子特征众多的特点，我们选择深度学习作为模型构建方向，以探索大数据中的信息。考虑到因子特征数据集0主要由股票的基础价格数据构成，然而因子特征数据集1、因子特征数据集2以及风险数据集都是经过加工的匿名特征，故考虑使用CNN-LSTM模型联合训练数据集。

2.1 网络结构设计

2.1.1 LSTM层：

由于股票的原始价格数据未经处理，含有众多复合信息，较为驳杂，我们在这里构建了LSTM直接提取股票原始价格序列中的时序信息，以作为全连接层的输入之一。

1. hidden\_dim = 1
2. self.l = nn.LSTM(input\_size=hidden\_dim, hidden\_size=hidden\_dim, num\_layers=4,
3. time\_major=False,
4. direction='bidirect')

2.1.2 1D卷积层：

在时间序列数据中，通常假设一个数据点主要由其附近的数据点影响，即存在局部依赖。于是我们建立1D卷积层，捕捉特征数据集中每个特征的这种局部依赖。考虑到匿名特征都各自具有信息的指向性所含信息不仅有时序，并且1D卷积层在效率上高于LSTM，故我们使用1D卷积层对特征数据集进行处理，以作为全连接层的输入之一。

1. self.conv\_1D = nn.Sequential(
2. nn.Conv1D(83, 83, kernel\_size=1,weight\_attr=weight\_ini),
3. nn.BatchNorm1D(83),
4. nn.Dropout(fc3\_dropout),
5. nn.ReLU(),
6. nn.MaxPool1D(kernel\_size=2),
7. )

2.1.3 2D卷积层：

虽然时间序列都是一维序列，但多元时间序列将数据集的维度提升到了二维，即增加了特征与特征之间的信息，于是我们选择建立2D卷积层以识别数据集中存在的特征间的空间依赖（即时间和特征两个维度）。通过2D卷积层，我们不仅在时间维度上进行卷积，还可以在特征维度（通道）上进行卷积，从而学习跨多个特征的复杂信息，以作为全连接层的输入之一。

1. self.conv\_2D = nn.Sequential(
2. nn.Conv2D(1, 1, kernel\_size=2,weight\_attr=weight\_ini),
3. nn.BatchNorm2D(1),
4. nn.Dropout(fc3\_dropout),
5. nn.ReLU(),
6. nn.MaxPool2D(kernel\_size=2),
7. )

2.1.4 全连接层：

为了综合学习特征数据集的时间依赖、空间依赖以及股价时序信息，我们最终构建了一个全连接层以训练合并后的输入数据。

1. self.f\_model = nn.Sequential(
2. nn.Linear(2280, fc1\_size,weight\_attr=weight\_ini),
3. nn.BatchNorm1D(fc1\_size),
4. nn.ReLU(),
5. nn.Dropout(fc1\_dropout),
6. nn.Linear(fc1\_size, fc2\_size,weight\_attr=weight\_ini),
7. nn.BatchNorm1D(fc2\_size),
8. nn.ReLU(),
9. nn.Dropout(fc2\_dropout),
10. nn.Linear(fc2\_size, fc3\_size,weight\_attr=weight\_ini),
11. nn.BatchNorm1D(fc3\_size),
12. nn.ReLU(),
13. nn.Dropout(fc3\_dropout),
14. nn.Linear(fc3\_size, 1,weight\_attr=weight\_ini),
15. )

2.2 向前传播设计

（1）股价信息处理: 使用 LSTM网络分别处理收盘价、最高价、开盘价、最低价和调整后的成交量加权平均价格。

（2）二维卷积: 对特征数据集进行二维卷积处理，提取空间依赖（相关依赖）。

（3）一维卷积和LSTM: 对特征数据集进行一维卷积处理，提取时间依赖，并使用 LSTM网络处理经过一维卷积得到的时间依赖信息。

（4）拼接: 将所有处理过的特征进行拼接。

（5）最后，将拼接后的数据通过全连接层模型进行预测。

1. **def** forward(self, x):
2. AdjPrice\_Close = paddle.slice(x, axes=[-1], starts=[0], ends=[1]).squeeze(1)
3. AdjPrice\_Close, \_ = self.l(AdjPrice\_Close)
4. AdjPrice\_Close = paddle.reshape(AdjPrice\_Close, (AdjPrice\_Close.shape[0], AdjPrice\_Close.shape[1] \* AdjPrice\_Close.shape[2]))
6. AdjPrice\_High = paddle.slice(x, axes=[-1], starts=[1], ends=[2]).squeeze(1)
7. AdjPrice\_High, \_ = self.l(AdjPrice\_High)
8. AdjPrice\_High = paddle.reshape(AdjPrice\_High, (AdjPrice\_High.shape[0], AdjPrice\_High.shape[1] \* AdjPrice\_High.shape[2]))
10. AdjPrice\_Open = paddle.slice(x, axes=[-1], starts=[2], ends=[3]).squeeze(1)
11. AdjPrice\_Open, \_ = self.l(AdjPrice\_Open)
12. AdjPrice\_Open = paddle.reshape(AdjPrice\_Open, (AdjPrice\_Open.shape[0], AdjPrice\_Open.shape[1] \* AdjPrice\_Open.shape[2]))
14. AdjPrice\_Low = paddle.slice(x, axes=[-1], starts=[3], ends=[4]).squeeze(1)
15. AdjPrice\_Low, \_ = self.l(AdjPrice\_Low)
16. AdjPrice\_Low = paddle.reshape(AdjPrice\_Low, (AdjPrice\_Low.shape[0], AdjPrice\_Low.shape[1] \* AdjPrice\_Low.shape[2]))
18. Vwap\_Adj\_1d = paddle.slice(x, axes=[-1], starts=[4], ends=[5]).squeeze(1)
19. Vwap\_Adj\_1d, \_ = self.l(Vwap\_Adj\_1d)
20. Vwap\_Adj\_1d = paddle.reshape(Vwap\_Adj\_1d, (Vwap\_Adj\_1d.shape[0], Vwap\_Adj\_1d.shape[1] \* Vwap\_Adj\_1d.shape[2]))
22. The\_2D = paddle.unsqueeze(x,axis=1)
23. The\_2D = self.conv\_2D(The\_2D)
24. The\_2D = paddle.reshape(The\_2D, (The\_2D.shape[0], The\_2D.shape[1] \* The\_2D.shape[2] \* The\_2D.shape[3]))
26. x = paddle.transpose(x,perm=[0, 2, 1])
27. x = self.conv\_1D(x)
28. out = paddle.transpose(x,perm=[0, 2, 1])
29. The\_1D\_and\_lstm, \_ = self.lstm(out)
30. The\_1D\_and\_lstm = paddle.reshape(The\_1D\_and\_lstm, (The\_1D\_and\_lstm.shape[0], The\_1D\_and\_lstm.shape[1] \* The\_1D\_and\_lstm.shape[2]))
32. IN = paddle.concat((The\_2D, The\_1D\_and\_lstm, AdjPrice\_Close, AdjPrice\_High, AdjPrice\_Open, AdjPrice\_Low, Vwap\_Adj\_1d), axis=1)
33. out = self.f\_model(IN)
34. **return** out

3 训练细节

3.1 过拟合优化

在训练模型的过程中，为了防止模型过拟合，我们在最终的全连接层的每一层权重后采用了Dropout方法，根据神经元数量递减地设置了Dropout参数，以防止模型过拟合，并达到集成训练的效果。

3.2 输出的广度

在构造卷积网络和全连接网络时，为了确保数据输出的广度，我们对网络权重进行了初始化处理，使用Kaiming Normal函数方法对权重进行初始取值，降低神经元的输出分布存在偏向的可能性，确保输出的广度。

3.3 学习方式选择

由于特征数据集兼具时间和特征两个维度，一定程度上可以视作图像处理，于是我们选择AdamW优化器进行模型的参数学习调整。通过AdamW优化器，模型的学习过程更准确地执行 L2 正则化，具有 Adam 优化器的优点和更有效的权重衰减。

3.4 Loss函数的选择

基于，股价收益分布近似高斯分布的特点，我们选择GaussianNLLLoss作为Loss函数以控制模型预测的分布近似与收益分布。GaussianNLLLoss（Gaussian Negative Log Likelihood Loss）是一种用主要于回归问题的损失函数，特别是当目标（标签）变量近似服从高斯分布（正态分布）的时候，这个损失函数将不仅考虑预测目标值（均值）还将考虑不确定性（标准差或方差）。

4 预测

模型的预测与正常的深度学习相近似，通过fit已经训练好的模型权重，即可直接运行预测过程。

测试数据和训练、验证数据在生成以“asset”为依据的groups时，唯一的区别在于测试数据需要保留“date”和“asset”序列，以便重构预测结果。

在对测试数据进行预处理时，不同于训练、验证数据的处理，在进行数据特征的对数变换和重构时，必须要保证数据的结构性变化可逆，不能产生由删除数据造成的不可逆结构性变化，这样才能在经过模型预测后将输出的结果与预留的“date”和“asset”序列相匹配，进行相应的排序操作。但这样的缺陷就在于：

（1）股票仅有一个时间戳的特征时，前5列价格数据将无法差分，被定义为0值；

（2）每只股票真实应用于预测的数据特征前面需要补充21天的数据特征作为支撑（仅由前面21天数据打包的数据样本将被填充0以维持样本形状）；

（3）对于股票特征数据少于21天的股票数据，模型对其预测性能将大大下降（样本内的数据基本全为0）。

基于以上存在的缺陷，我们选择将用于预测的特征尽量扩大时间范围，用多个月的数据作为支撑，确保我们真实需要预测的结果的稳定。

1. test\_dataset\_pre = []
2. **for** name, group **in** groups\_test:
3. one\_sample\_test = group.sort\_values(by=['date']).iloc[:, 2:].fillna(0).reset\_index(drop=True)
4. one\_sample\_test = np.sign(one\_sample\_test) \* np.log1p(np.abs(one\_sample\_test))
5. one\_sample\_test\_price = one\_sample\_test.iloc[:, 0:5].diff()
6. one\_sample\_test\_price = one\_sample\_test\_price.fillna(0)
7. one\_sample\_test = pd.concat([one\_sample\_test\_price, one\_sample\_test.iloc[:, 5:]],axis=1).values.astype(np.float32)
9. **for** idx **in** range(one\_sample\_test.shape[0]):
10. **if** idx > 20:
11. feat\_data = one\_sample\_test[idx-21:idx]
12. test\_dataset\_pre.append(feat\_data)
14. **else**:
15. pad = np.zeros((20-idx, 83)).astype(np.float32)
16. feat\_data = one\_sample\_test[:idx+1]
17. feat\_data = np.vstack([pad, feat\_data])
19. test\_dataset\_pre.append(feat\_data)
20. **del** one\_sample\_test

5 投资策略

由于我们假设label是股票在当前时间戳上的对应周收益率，并且模型输出时间戳靠后的精度更高，所以我们将直接按时间戳对预测结果进行归一化处理分配权重，将每日的预测权重视为该日后一整周的股票权重，并提取实际有效的预测时间范围对应的权重。

在进行初赛数据预测时，我们采用“周一”更新的方式，以每周一作为节点进行投资权重更新，符合模型预测意义——对周收益的预测，同时也确保不会由于频繁交易累积较高的交易成本（手续费）。

在进行复赛数据预测时，我们沿用初赛的方式，以最新的预测权重进行投资分配，并保持权重一周不变。