数据挖掘课程实验报告

基于 LSTM 模型预测单支股票行情

姓 名: 祁 好 雨
学 号: 1752919
班 级: 计算机一班
指导教师: 向阳

同济大学
计算机科学与技术系

2020 年 12 月 29 日

目 录

1 研究背景与意义

使用数学模型判断股市的走向进行投资，利用大量的数据和计算机技术取代人工判断，即量化交易的应用，对现有的国内外市场投资应用都有着深远的影响。随着深度学习的进一步发展，量化交易不仅在工程领域产生了影响，也为金融、经济等学科的理论研究掀开了新的篇章。

从全球市场的参与主体来看，按照管理资产的规模，2018 年全球排名前四以及前六位中的五家资管机构，都是依靠计算机技术来开展投资决策，而且进入 2019 年由量化及程序化交易所管理的资金规模进一步扩大。从就业人员的薪资水平来看，全球超 70% 的资金交易用计算机或者程序进行，其中一半是由量化或者程序化的管理人来操盘。在国外招聘网站搜索金融工程师 (包括量化、数据科学等关键词) 会出现超过 33 万个相关岗位。

从高校的培养方向来看，已有超过 450 所美国大学设置了金融工程专业，每年相关专业毕业生达到 1.5 万人，市场需求与毕业生数量的差距显著，因此数据科学、计算机科学、会计以及相关 STEM（基础科学）学生毕业后进入金融行业从事量化分析和应用开发的相关工作。

然而，目前量化交易的应用情况在国内外市场间差距较大。目前国内量化投资规模大概是 3500 到 4000 亿人民币，其中公募基金 1200 亿，其余为私募量化基金，数量达 300 多家，占比 3%(私募管理人共 9000 多家)，金额在 2000 亿左右。中国证券基金的整体规模超过 16 万亿，其中公募 14 万亿，私募 2.4 万亿，乐观估计，量化基金管理规模在国内证券基金的占比在 1% 2%，在公募证券基金占比不到 1%，在私募证券基金占比 5% 左右，相比国外超过 30% 的资金来自于量化或者程序化投资，国内未来的增长空间巨大。

除此之外，国内研究量化交易的学者规模也不多。国内的金融工程等涉及计算机技术和金融学科交叉领域的学科建设也还在起步阶段。我们这些计算机科学与技术系的同学在本科阶段了解量化交易的基础，不仅有利于我们未来选择研究方向，更有利于了解当前工程界对于量化交易的处理模式。如果未来选择深入研究量化交易领域，本项目不失为一个良好的起步。

2 问题描述

利用 tushare 下载单支股票的相关行情数据，根据现有的数据划分训练集和验证集，利用 talib 库对获取已有数据的统计特征，预测明日股票走势并评估模型的性能。在本项目中选择的单支股票为'000001.SH'，本项目可通过调整参数选择预测股票的开市价，闭市价和日均价。报告中建立的模型主要任务为预测股票的闭市价。

3 数据获取与清洗

3.1 训练集，验证集，测试集分割说明

本实验采用的数据集为 tushare API 提供的单支股票'000001.SH' 的日线行情数据，主要使用 2010 年 9 月 1 日到 2019 年 9 月 1 日的闭市价作为训练数据和验证数据，其中采取连续 100 天的数据作为一个时间序列，前 49 天的数据作为输入，最后一天的数据作为 target。生成的时间序列数量为`len(daily_data)-50`，选取前 80% 的数据作为训练集，后 20% 的数据作为验证集。测试集为'000001.SH' 从 2019 年 9 月 1 日到 2020 年 12 月 20 日的数据。

3.2 数据预处理

采用 sklearn 提供的 MinMaxScaler 来对数据进行规范化，这样避免了测试集数据和训练集数据价格起点差异较大的情况对模型预测的影响。最后采用`inverse_transform`反向求得在原始的股票闭市价格量纲上的预测结果。MinMaxScaler 操作的数学模型可以概括为下述公式：

$$f(i) = (data(i) - \min_{i=0}^{len(data)} data(i))/\max_{i=0}^{len(data)} data(i) \tag{1}$$

$$predict(i) = f^{-1}(y_i) \tag{2}$$

4 行情特征工程

本项目中关于股票日线行情的统计特征主要利用了 talib 库提取了股票的平均，指数平均，动量趋势等特征，下述章节将会对这些特征进行一一说明并评估特征间的相关性。

4.1 talib 库安装说明

TA-Lib 是一个开源的用于金融工程领域提取股市特征的第三方库。其安装可通过 pip installer 进行安装。但是，在通过该工具进行安装之前，需要先配置 TA-Lib 的安装环境。需要在下述链接中查找到对应工作环境 python 版本号和机器字长的'whl' 安装包并下载。whl 文件下载地址
下载后在工作环境中 pip install [*.whl] 使用 pip installer 安装下载好的 whl 包，再进行 TA-Lib 的安装，否则会出现安装错误的情况。

4.2 特征挑选

参考 talib 的文档中对于不同特征的描述，我挑选了下述特征。(部分特征将会在模型算法与设计一节中详细说明)。

- Simple Moving Average: 该特征用于计算选中的一段时间内的闭市价的平均值。在当前训练数据上应用如下：

```
df1['MA'] = talib.SMA(df1['close'],args.window_size-1)
df1[['close','MA']].plot(figsize=(8,4))
plt.show()
```

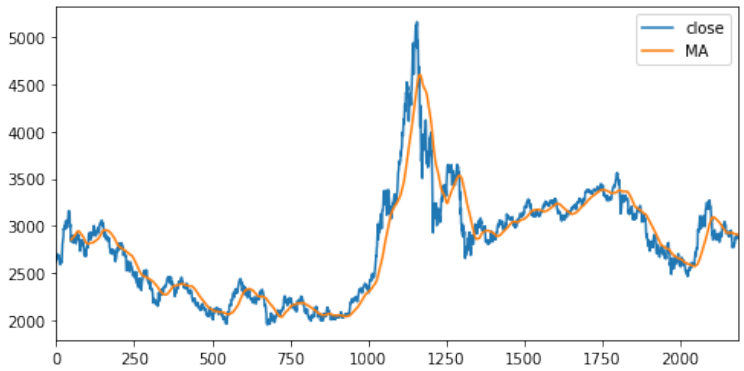


图 1: 特征：Simple Moving Average 图

- Exponential Weighted Moving Average: 该特征也是用于计算选中的一段时间内的闭市价的平均值的算法得出的，但是该特征的计算过程中最近的一段时间的闭市价权重占比更大。在当前训练数据上应用如下：

```
df1['EMA'] = talib.EMA(df1['close'], timeperiod =
    args.window_size-1)
df1[['close','EMA']].plot(figsize=(8,4))
plt.show()
```

- Average Directional Movement Index: 该特征可用来评估总体趋势及其程度，是一个动量指标。在当前训练数据上应用如下：

```
df1['avg'] = talib.ADX(df1['high'],df1['low'], df1['close'],
    timeperiod=args.window_size-1)
df1[['avg']].plot(figsize=(8,4))
```

- Bollinger Bands: 这一类特征是当前一段时间内闭市价的统计特征，用来表示股票闭市价在一段时间程度上的价格和易变性，由 John

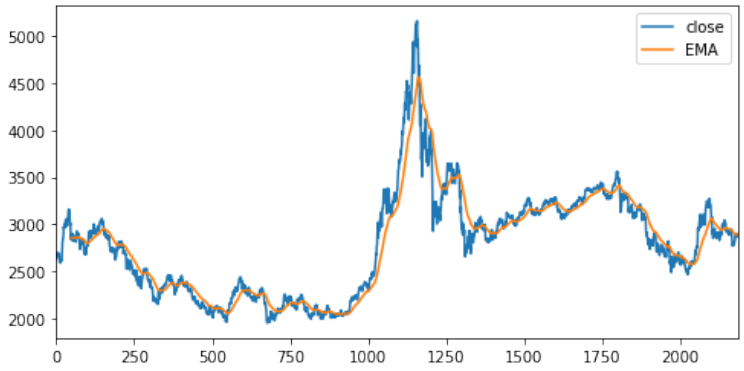


图 2: 特征：Exponential Weighted Moving Average 图

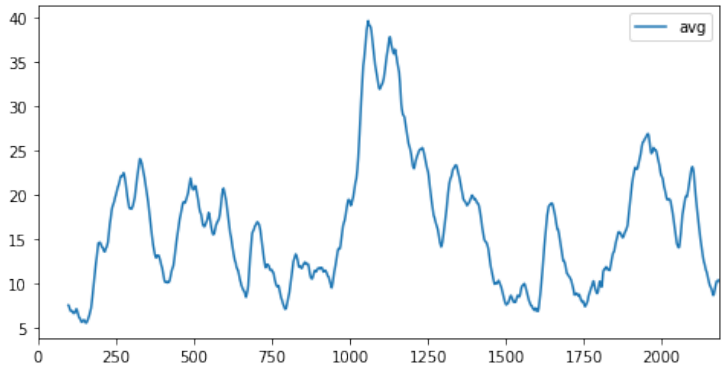


图 3: 特征：Average Directional Movement Index 图

Bollinger 提出的公式计算得出。在当前训练数据上应用如下：

```
df1['up_band'], df1['mid_band'], df1['low_band'] =   \
                    talib.BBANDS(df1['close'], timeperiod
                        =args.window_size-1)
df1[['close','up_band','mid_band','low_band']].plot(figsize=
                                (8,4))
plt.show()
```

- Relative Strength Index: 该特征是用来评估金融市场的一技术特征，根据最近的股市价格来衡量股票或股市当前或历史性的涨跌幅度。在当前训练数据上应用如下：

```
df1['Relative'] = talib.RSI(df1['close'],args.window_size-1)
df1['Relative'].plot(figsize=(8,4))
plt.show()
```

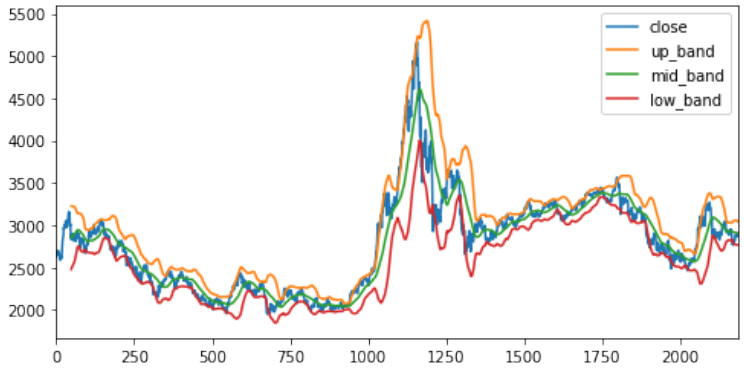


图 4: 特征：Bollinger Bands 图

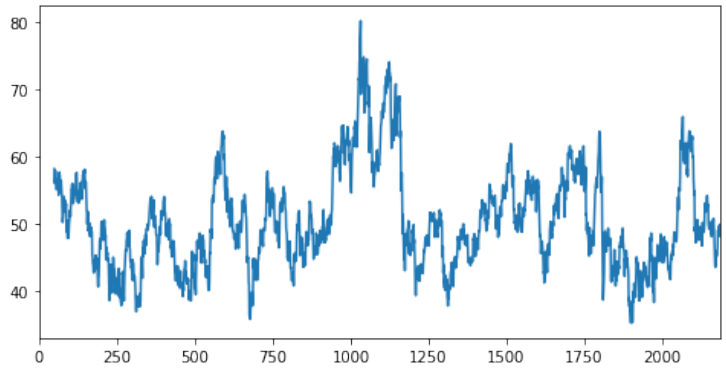


图 5: 特征：Relative Strength Index 图

4.3 特征间相关性评估

通过将选出的特征加入到原 DataFrame 中，并采用 pandas.DataFrame 带有的 corr 方法来评估特征之间两两相关性。pandas 采用的默认相关性评估方法是 pearson 系数法，所以下述相关性主要描述的是特征间的线性相关性。本项目选出的特征及闭市价间线性相关性如下所示：

4.4 加入特征后的数据清洗

需要注意的是在采用了上述特征之后，由于部分特征是需要根据前一段时间计算得出的。这里时间窗口的长度统一采用 args 中的 windowsize-1，也就是封装后的时间序列样本长度。这样做了之后前 49 天的部分特征会得出 NAN 的结果，在本项目中对于这种样本采取的清洗手段是删除掉含有 NAN 的记录。

	close	MA	EMA	avg	up_band	mid_band	low_band	Relative
close	1.000000	0.938284	0.954983	0.322579	0.915107	0.938284	0.888154	0.419223
MA	0.938284	1.000000	0.996809	0.265890	0.975259	1.000000	0.946630	0.152466
EMA	0.954983	0.996809	1.000000	0.272899	0.970114	0.996809	0.946573	0.183448
avg	0.322579	0.265890	0.272899	1.000000	0.395924	0.265890	0.052625	0.170900
up_band	0.915107	0.975259	0.970114	0.395924	1.000000	0.975259	0.851954	0.171131
mid_band	0.938284	1.000000	0.996809	0.265890	0.975259	1.000000	0.946630	0.152466
low_band	0.888154	0.946630	0.946573	0.052625	0.851954	0.946630	1.000000	0.111623
Relative	0.419223	0.152466	0.183448	0.170900	0.171131	0.152466	0.111623	1.000000

图 6: 特征间线性相关性图

5 模型算法设计

5.1 benchmark 概述

本项目采用的 benchmark 是时间序列模型通用的机器学习模型，即 exponential moving average。下面对这种建模方法进行简单介绍。在 exponential moving average 方法中，对 t+1 时刻闭市价格的预测可以简化为下述数学模型：

$$x_{t+1} = Ema_t = \gamma * Ema_{t-1} + (1-\gamma) * x_t \tag{3}$$

$$Ema_0 = 0 \tag{4}$$

其中，Ema_t 代表 t 时刻的 exponential moving average，可将其视为对 t 时刻之前的 x 值的一种记忆，而参数 γ 负责控制模型的输出对前段时间记忆和当前输入采纳的比例。从上述结果中可以看出，当模型只需要预测未来一天的股票走势的时候，这种模型的效果很优越，可以作为 lstm 模型的 benchmark 与之进行比较。但是这种模型也有它的弊端，在实际应用中，仅预测未来一天的股票走势是没有太大意义的，而这种模型在需要预测未来多天时，性能极为不好 [1]。下述的数学公式可以证明，该模型预测的未来多天的输出与未来一天的输出完全一致。

$$x_{t+1} = Ema_t = \gamma * Ema_{t-1} + (1-\gamma) * x_t \tag{5}$$

$$x_{t+2} = Ema_{t+1} = \gamma * Ema_t + (1-\gamma) * x_{t+1} = x_{t+1} \tag{6}$$

所以如果模型需要预测未来多天的数据（这也更符合实际应用情形），我们还是需要采用可以记录短期记忆和长期记忆的 lstm 模型 (long short-term memory model)。但是 exponential moving average 模型可以作为后续 lstm 模型性能评估的标准。

5.2 lstm 模型概述

lstm 模型 (long short-term memory model) 是一个在时间序列数据上性能特别优秀的模型。它可以预测未来任意天数后的结果。下面对 lstm 模型算法进行简单介绍。

lstm 模型脱胎于循环神经网络，顾名思义，即为在神经网络模型中做循环操作。循环神经网络的思想可以由下图来表示。

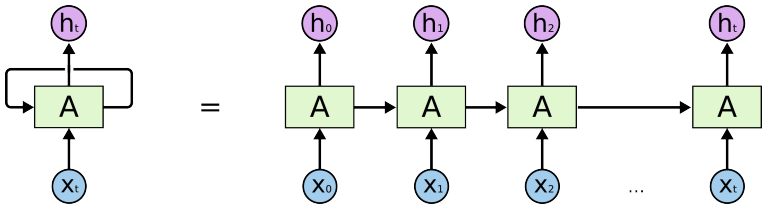


图 7: 循环神经网络模型示意图

上述图中的像锁链一样的结构表达了循环神经网络天然的适用于时间序列或列表状数据。但是循环神经网络也有它的不足之处，以它在自然语言处理方向的应用举例，如果当前输入语句是 “the clouds are in the sky”，当模型需要推算 sky 这个名词的时候，不需要额外的语境（前述输入）就可以简单推算。

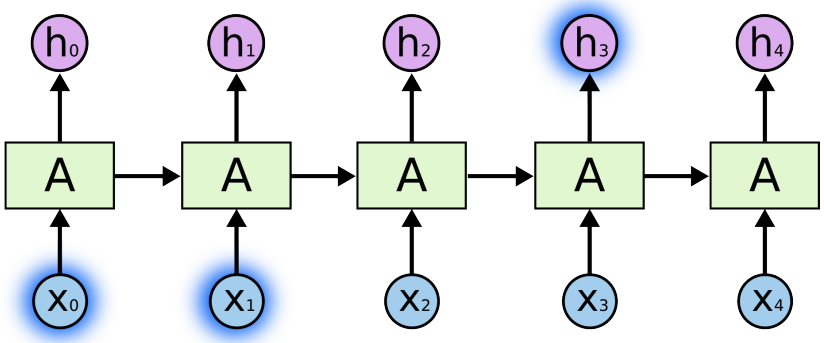


图 8: 循环神经网络模型短期依赖示意图

而当模型遇到输入文章“I’ m a French....I live in France”时，就需要获得很久以前输入的语境了。当需要的信息和现有信息之间的时间间隔很长时，循环神经网络就没有办法准确的将两则信息联系在一起了。这就是循环神经网络的不足：长期依赖问题 (Long Term Dependencies)。

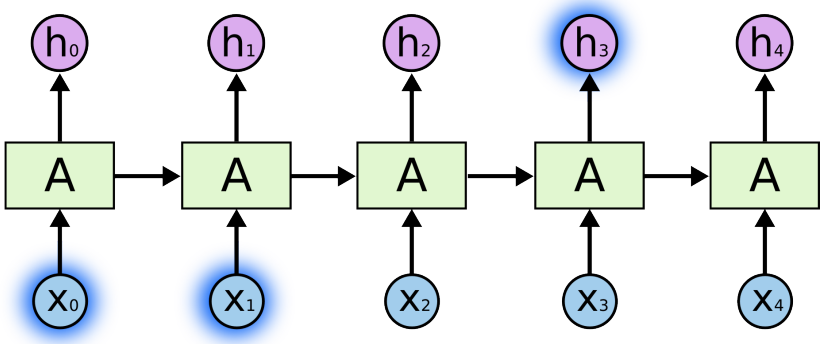


图 9: 循环神经网络模型长期依赖示意图

但是 LSTM 模型的出现解决了上述问题，它既可以利用模型中存储的短期记忆又可以利用模型中存储的长期记忆。它和循环神经网络的不同主要取决于当前输出结点对前继输出和当前输入的结合的处理。如下图所示，循环神经网络一般只简单的对当前输入和前继输出进行一个简单的连接，再用 tanh 函数进行激活。

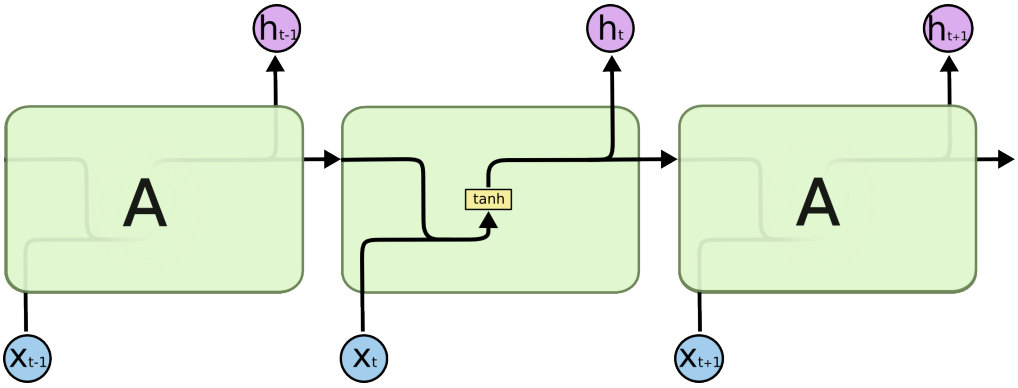


图 10: 循环神经网络模型内部结构图

而 LSTM 模型却在内部设置了三个至关重要的门：遗忘门，选择门和输出门。这三个门决定了当前模型状态对于过去的记忆的遗忘情况，对于新特征的采纳情况和对于输出的综合情况，如下图所示：

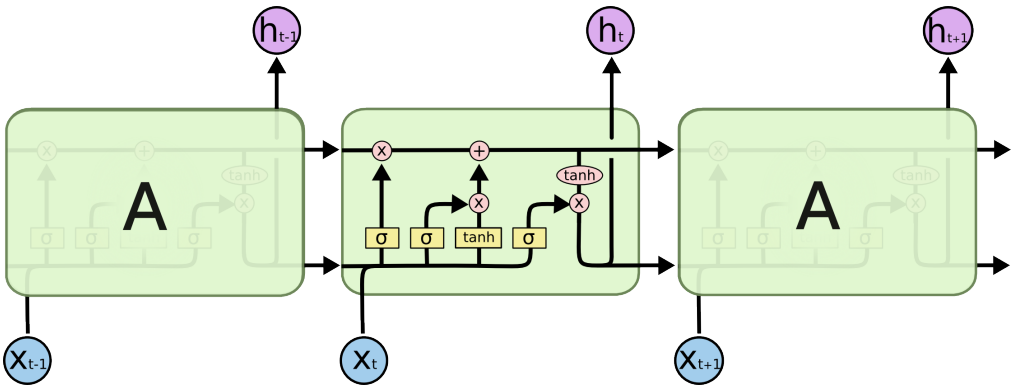
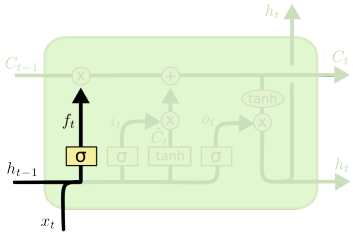


图 11: LSTM 模型内部结构图

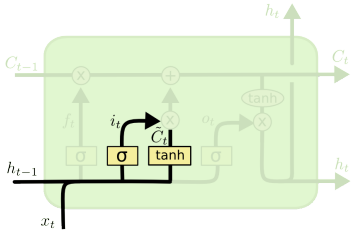
详细说来，LSTM 模型的内部处理分以下几步。第一步：遗忘门根据上一次的输出和当前输入计算得 ft，sigmoid 函数将激活后的值控制在 0 到 1 之间，用 ft 点乘上一次的模型状态 C_{t-1} 就确定了新的模型状态采用多少过去的记忆。



$$f_t = \sigma\left(W_f \cdot [h_{t-1}, x_t] \; + \; b_f\right)$$

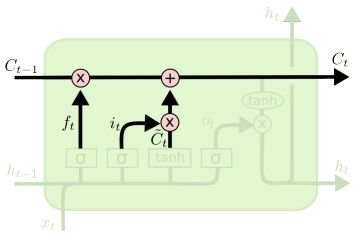
图 12: LSTM 遗忘门运算图

第二步：选择门根据上一次的输出和当前输入再加上 sigmoid 激活函数计算得此次新特征的选择概率，而该中间结果如果用 tanh 函数进行激活即得到此次输入的新特征，将两者点乘起来即为过滤后的新特征。将遗忘门过滤后的状态与选择们过滤后的新特征连接起来记得到本次计算的模型状态。



$$i_t = \sigma\left(W_i \cdot [h_{t-1}, x_t] \ + \ b_i\right)$$
$$\tilde{C}_t = \tanh(W_C \cdot [h_{t-1}, x_t] \ + \ b_C)$$

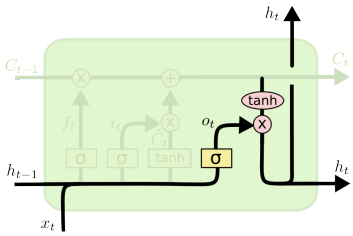
图 13: LSTM 选择门运算图



$$C_t = f_t * C_{t-1} + i_t * \tilde{C}_t$$

图 14: LSTM 当前状态生成图

第三步：输出门结合当前模型状态和当前输入，前次输出计算得当前输出。



$$o_t = \sigma\left(W_o \ [h_{t-1}, x_t] \ + \ b_o\right)$$
$$h_t = o_t * \tanh\left(C_t\right)$$

图 15: LSTM 当前输出生成图

5.3 benchmark 算法实现

benchmark 模型主要采用 sklearn 包的 MinMaxScaler 和我写的简单 exponential moving average 算法来实现，通过计算平均标准差来估计模型的性能。算法实现如下：
参数配置：

```
import tushare as ts
import torch
import pandas as pd
print(f'tushare version:{ts.__version__}')
print(f'torch version:{torch.__version__}')
print(f'pandas version:{pd.__version__}')
# tushare version:1.2.62
# torch version:1.3.1
# pandas version:0.20.3
class Args:
    def __init__(self):
        self.stock_code = '000001.SZ'
        self.startdate = '20100901'
        self.enddate = '20190901'
        self.predict_col='close'
        self.ratio = 0.2
        self.window_size = 100
        self.input_dim = 1
        self.hidden_dim = 32
        self.num_layers = 2
        self.output_dim = 1
        self.num_epochs = 100

args = Args()
```

数据获取及预训练：

```
ts.set_token('9608221f6e83cafa56f538f1d583e9802783b7589b042a8b62e04463')
pro = ts.pro_api()
df = pro.daily(ts_code=args.stock_code, start_date=args.startdate,
    end_date=args.enddate)
df1 = pd.DataFrame(df)
path = args.stock_code + '.csv'
df1.to_csv(path)

df1 = df1[['trade_date','open', 'high', 'close', 'low', 'vol', 'change']]
df1 = df1.sort_values(by='trade_date')

from sklearn.preprocessing import MinMaxScaler

price = df1[[args.predict_col]]
scaler = MinMaxScaler(feature_range=(-1, 1))
price.loc[:,args.predict_col] =
```

```
    scaler.fit_transform(price[args.predict_col].values.reshape(-1,1))

import numpy as np

'''
    for the input stock,
    slice it for a lookback size and make one step further,
    ratio determines the size of train and test set,
    choose the last price of every slice as label
'''
def split_data(stock, ratio=0.2):
    data = stock
    data = np.array(data);
    test_set_size = int(np.round(ratio*data.shape[0]));
    train_set_size = data.shape[0] - (test_set_size);

    train_data = data[:train_set_size]
    test_data = data[train_set_size:]

    return [train_data,test_data]

train_data,test_data = split_data(price.values,args.ratio)
```

训练模型:

```
N = len(train_data)

run_avg_predictions = []
run_avg_x = []

mse_errors = []

running_mean = 0.0
run_avg_predictions.append(running_mean)

decay = 0.5

for pred_idx in range(1,N):
    running_mean = running_mean*decay + (1.0-decay)*train_data[pred_idx-1]
    run_avg_predictions.append(running_mean)
    mse_errors.append((run_avg_predictions[-1]-train_data[pred_idx])**2)
    run_avg_x.append(pred_idx)

print('MSE error for EMA averaging: %.5f'%(0.5*np.mean(mse_errors)))
```

验证模型:

```
N = len(test_data)

run_avg_predictions = []
run_avg_x = []
```

```
mse_errors = []

running_mean = 0.0
run_avg_predictions.append(running_mean)

decay = 0.5

for pred_idx in range(1,N):
    running_mean = running_mean*decay + (1.0-decay)*test_data[pred_idx-1]
    run_avg_predictions.append(running_mean)
    mse_errors.append((run_avg_predictions[-1]-test_data[pred_idx])**2)
    run_avg_x.append(pred_idx)

print('MSE error for EMA averaging: %.5f'%(0.5*np.mean(mse_errors)))
```

数据可视化：

```
import matplotlib.pyplot as plt
%matplotlib inline


plt.figure(figsize = (18,9))
plt.plot(range(0,N),price.iloc[:N,0],color='b',label='True')
plt.plot(range(0,N),run_avg_predictions,color='orange', label='Prediction')
plt.xlabel('Date')
plt.ylabel('Mid Price')
plt.legend(fontsize=18)
plt.show()
```

5.4 LSTM 模型算法实现

LSTM 模型主要采用 pytorch 中的 LSTM 模型和 sklearn 中的 MinMaxScaler 来实现，通过计算平均标准差来估计模型的性能。算法实现如下：
参数配置：

```
import tushare as ts
import torch
import pandas as pd
import talib
import matplotlib.pyplot as plt
%matplotlib inline
print(f'tushare version:{ts.__version__}')
print(f'torch version:{torch.__version__}')
print(f'pandas version:{pd.__version__}')
print(f'talib version:{talib.__version__}')
# tushare version:1.2.62
# torch version:1.3.1
# pandas version:0.20.3
```

```
# talib version:0.4.19
class Args:
    def __init__(self):
        self.stock_code = '000001.SH'
        self.startdate = '20100901'
        self.enddate = '20190901'
        self.predict_col='close'
        self.ratio = 0.2
        self.window_size = 50
        self.input_dim = 8
        self.hidden_dim = 32
        self.num_layers = 2
        self.output_dim = 1
        self.num_epochs = 50
        self.features = []

args = Args()
```

数据获取及预处理:

```
ts.set_token('1fe161e9d8fb8b8e53c863528c68dba6a690f64bdafe6579586b1d17')
pro = ts.pro_api()
df = pro.index_daily(ts_code=args.stock_code, start_date=args.startdate,
    end_date=args.enddate)
df1 = pd.DataFrame(df)
path = args.stock_code + '.csv'
df1.to_csv(path)

df1 = df1[['trade_date','open', 'high', 'close', 'low', 'vol', 'change']]
df1 = df1.sort_values(by='trade_date').reset_index()
df1.drop('index',axis=1,inplace=True)

from sklearn.preprocessing import MinMaxScaler

price = df1[[args.predict_col]]
scaler = MinMaxScaler(feature_range=(-1, 1))
price.loc[:,args.predict_col] =
    scaler.fit_transform(price[args.predict_col].values.reshape(-1,1))

import numpy as np

'''
    for the input stock,
    slice it for a lookback size and make one step further,
    ratio determines the size of train and test set,
    choose the last price of every slice as label
'''
def split_data(stock, lookback, ratio=0.2):
    data_raw = stock
    assert type(data_raw)==np.ndarray,f'{type(data_raw)}'
    data = []
```

```
    # create all possible sequences of length seq_len
    for index in range(len(data_raw) - lookback):
        data.append(data_raw[index: index + lookback])

    data = np.array(data);
    test_set_size = int(np.round(ratio*data.shape[0]));
    train_set_size = data.shape[0] - (test_set_size);

    x_train = data[:train_set_size,:-1,:]
    y_train = data[:train_set_size,-1,:]

    x_test = data[train_set_size:,:-1]
    y_test = data[train_set_size:,-1,:]

    return [x_train, y_train, x_test, y_test]


lookback = args.window_size # choose sequence length
x_train, y_train, x_test, y_test = split_data(price.values,
    lookback,args.ratio)

import torch
import torch.nn as nn
x_train = torch.from_numpy(x_train).type(torch.Tensor)
x_test = torch.from_numpy(x_test).type(torch.Tensor)
y_train_lstm = torch.from_numpy(y_train).type(torch.Tensor)
y_test_lstm = torch.from_numpy(y_test).type(torch.Tensor)
y_train_gru = torch.from_numpy(y_train).type(torch.Tensor)
y_test_gru = torch.from_numpy(y_test).type(torch.Tensor)
```

特征工程:

```
df1['MA'] = talib.SMA(df1['close'],args.window_size-1)
df1[['close','MA']].plot(figsize=(8,4))
plt.show()
df1['EMA'] = talib.EMA(df1['close'], timeperiod = args.window_size-1)
df1[['close','EMA']].plot(figsize=(8,4))
plt.show()
df1['avg'] = talib.ADX(df1['high'],df1['low'], df1['close'],
    timeperiod=args.window_size-1)
df1[['avg']].plot(figsize=(8,4))
df1['up_band'], df1['mid_band'], df1['low_band'] =   \
                        talib.BBANDS(df1['close'], timeperiod
                            =args.window_size-1)
df1[['close','up_band','mid_band','low_band']].plot(figsize=
                                    (8,4))
plt.show()
df1['Relative'] = talib.RSI(df1['close'],args.window_size-1)
df1['Relative'].plot(figsize=(8,4))
plt.show()
```

```
pd.DataFrame(df1,columns=['close','MA','EMA','avg','up_band','mid_band','low_band','Relative']).corr()
args.features =
    ['MA','EMA','avg','up_band','mid_band','low_band','Relative']
args.input_dim = 1+len(args.features)
df1.dropna(inplace=True)
from sklearn.preprocessing import MinMaxScaler

price = df1.loc[:,[args.predict_col]+args.features]
print(price.shape)
for i in range(len(price.columns)):
    scaler = MinMaxScaler(feature_range=(-1, 1))
    price.iloc[:,i] =
        scaler.fit_transform(price.iloc[:,i].values.reshape(-1,1))

import numpy as np

'''
    for the input stock,
    slice it for a lookback size and make one step further,
    ratio determines the size of train and test set,
    choose the last price of every slice as label
'''
def split_data(stock, lookback, ratio=0.2):
    data_raw = stock
    assert type(data_raw)==np.ndarray,f'{type(data_raw)}'
    data = []

    # create all possible sequences of length seq_len
    for index in range(len(data_raw) - lookback):
        data.append(data_raw[index: index + lookback])

    data = np.array(data);
    test_set_size = int(np.round(ratio*data.shape[0]));
    train_set_size = data.shape[0] - (test_set_size);

    x_train = data[:train_set_size,:-1,:]
    y_train = np.expand_dims(data[:train_set_size,-1,0],-1)

    x_test = data[train_set_size:,:-1]
    y_test = np.expand_dims(data[train_set_size:,-1,0],-1)

    return [x_train, y_train, x_test, y_test]


lookback = args.window_size # choose sequence length
x_train, y_train, x_test, y_test = split_data(price.values,
    lookback,args.ratio)
```

训练模型:

```
'''
```

```
    create lstm model with nn.lstm receiving batch-first data
    and fully connected layer as model head
'''
class LSTM(nn.Module):
    def __init__(self, input_dim, hidden_dim, num_layers, output_dim):
        super(LSTM, self).__init__()
        self.hidden_dim = hidden_dim
        self.num_layers = num_layers

        self.lstm = nn.LSTM(input_dim, hidden_dim, num_layers,
            batch_first=True)
        self.fc = nn.Linear(hidden_dim, output_dim)
    def forward(self, x):
        h0 = torch.zeros(self.num_layers, x.size(0),
            self.hidden_dim).requires_grad_()
        c0 = torch.zeros(self.num_layers, x.size(0),
            self.hidden_dim).requires_grad_()
        out, (hn, cn) = self.lstm(x, (h0.detach(), c0.detach()))
        out = self.fc(out[:, -1, :])
        return out

class RMSELoss(torch.nn.Module):
    def __init__(self, eps=1e-6):
        super().__init__()
        self.mse = torch.nn.MSELoss()
        self.eps = eps

    def forward(self,yhat,y):
        loss = torch.sqrt(self.mse(yhat,y) + self.eps)
        return loss

model = LSTM(input_dim=args.input_dim, hidden_dim=args.hidden_dim,
    output_dim=args.output_dim, num_layers=args.num_layers)
criterion = RMSELoss()
optimiser = torch.optim.Adam(model.parameters(), lr=0.01)

import time
hist = np.zeros(args.num_epochs)
start_time = time.time()
lstm = []
for t in range(args.num_epochs):
    y_train_pred = model(x_train)
    loss = criterion(y_train_pred, y_train_lstm)
    print("Epoch ", t, "MSE: ", loss.item())
    hist[t] = loss.item()
    optimiser.zero_grad()
    loss.backward()
    optimiser.step()

training_time = time.time()-start_time
print("Training time: {}".format(training_time))
```

验证模型：

```
with torch.no_grad():
    y_test_pred = model(x_test)
    loss = criterion(y_test_pred, y_test_lstm)
```

数据可视化：

```
import matplotlib.pyplot as plt
%matplotlib inline
final_y_train_pred = model(x_train)



fig =plt.figure(figsize=(24,8))
'''-------------------------------------------------normalized data
    visualization-------------------------------------------------------'''
plt.subplot(121)
t_train = np.arange(0,len(y_train_lstm))
plt.plot(t_train,final_y_train_pred.detach(),color='g',label='predict_y_train')
plt.plot(t_train,y_train_lstm,color='b',label='true_y_train')

t_test = np.arange(len(y_train_lstm),len(y_train_lstm)+len(y_test_lstm))
plt.plot(t_test,y_test_pred.detach(),color='r',label='predict_y_test')
plt.plot(t_test,y_test_lstm,color='b',label='true_y_test')

plt.title('normalized train & test result visualization')
plt.xlabel('time')
plt.ylabel('normalized close price')
plt.legend()

'''-------------------------------------------------real data
    visualization-------------------------------------------------------'''
plt.subplot(122)
real_y_train = scaler.inverse_transform(y_train_lstm)
real_y_train_pred = scaler.inverse_transform(final_y_train_pred.detach())
real_y_test = scaler.inverse_transform(y_test_lstm)
real_y_test_pred = scaler.inverse_transform(y_test_pred.detach())

t_train = np.arange(0,len(y_train_lstm))
plt.plot(t_train,real_y_train_pred,color='g',label='predict_y_train')
plt.plot(t_train,real_y_train,color='b',label='true_y_train')

t_test = np.arange(len(y_train_lstm),len(y_train_lstm)+len(y_test_lstm))
plt.plot(t_test,real_y_test_pred,color='r',label='predict_y_test')
plt.plot(t_test,real_y_test,color='b',label='true_y_test')

plt.title('real train & test result visualization')
plt.xlabel('time')
plt.ylabel('real close price')
plt.legend()
```

```
plt.plot(hist)
plt.title('trainning loss')
plt.xlabel('epoch')
plt.ylabel('MSE loss')
```

测试模型:

```
df = pro.daily(ts_code=args.stock_code, start_date=args.enddate,
    end_date='20201220')
df2 = df[['trade_date','open', 'high', 'close', 'low', 'vol', 'change']]
df2 = df2.sort_values(by='trade_date')

# from sklearn.preprocessing import MinMaxScaler

price = df2[[args.predict_col]]
# scaler = MinMaxScaler(feature_range=(-1, 1))
price.loc[:,args.predict_col] =
    scaler.fit_transform(price[args.predict_col].values.reshape(-1,1))

def test_data_process(stock, lookback):
    data_raw = stock
    assert type(data_raw)==np.ndarray,f'{type(data_raw)}'
    data = []

    # create all possible sequences of length seq_len
    for index in range(len(data_raw) - lookback):
        data.append(data_raw[index: index + lookback])

    data = np.array(data);
    print(data.shape)
    x = data[:,:-1,:]
    y = data[:,-1,:]
    return [x,y]

x,y = test_data_process(price.values,args.window_size)
# x.shape,y.shape

x = torch.from_numpy(x).type(torch.Tensor)
y = torch.from_numpy(y).type(torch.Tensor)

with torch.no_grad():
    y_pred = model(x)
    loss = criterion(y_pred,y)

t_train = np.arange(0,len(y))
plt.plot(t_train, y_pred.detach(),color='g',label='predict_y')
plt.plot(t_train, y.detach(),color='b',label='true_y')

plt.title('normalized test set result visualization')
plt.xlabel('time')
```

```
plt.ylabel('normalized close price')
plt.legend()
```

6 实验分析与模型效果评价

本实验最终利用模型输出明日闭市价格来对股市明日行情进行预测，整体来说如果只预测未来一天的结果，其中 LSTM 模型采用长度为 49 天的时间序列，benchmark 模型采用长度为 99 天的时间序列。benchmark 模型计算 MSE loss，LSTM 模型计算 RMSE loss。最后将两者的预测结果从闭市价转到减去真实的前一天闭市价从而转为分类问题来进行评判准确率。

6.1 验证集结果说明

benchmark 的验证集损失函数值为 0.00078，对整体训练集和验证集闭市价格的预测拟合如下图所示：

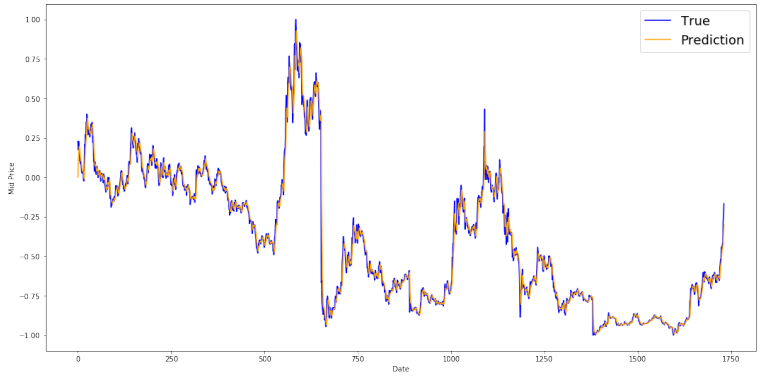


图 16: benchmark 模型拟合图

LSTM 模型经过了几次迭代，在最初的迭代设置中，模型只训练 30 个 Epoch 且没有做特征工程，可以从拟合曲线中明显的看出模型整体呈现欠拟合的趋势。

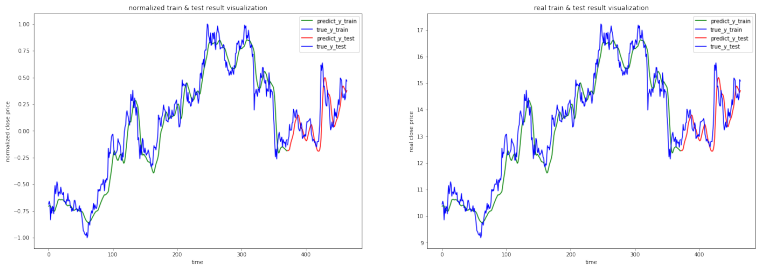


图 17: LSTM 模型训练 30Epoch 拟合图

在后续的迭代设置中，加入了统计特征，加宽了前瞻窗口的尺寸，模型训 50 个 Epoch，可以从拟合曲线中看出此时模型拟合的结果较好。在模型在训练

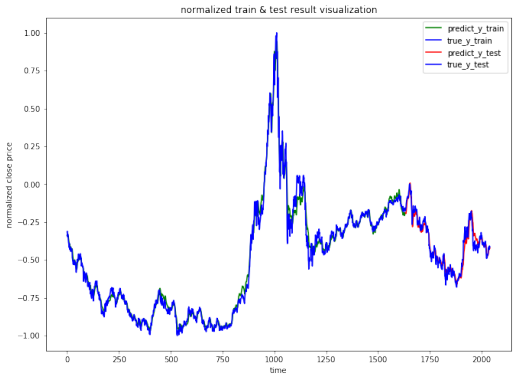


图 18: LSTM 模型训练 50Epoch 拟合图

过程中的 loss 下降折线图中可以看出，在大约 50epoch 之后，损失函数值下降不明显。

最终迭代，模型的验证集 loss 为 0.0391。上述拟合图中，左图为模型的真实输出，即使用 MinMaxScaler 标准化后的预测值，右图为模型的实际预测值，即反向 transform 得到的在原量纲上的模型的预测价格。两种量纲对模型预测的准确率和曲线走势无明显影响。

6.2 测试集结果说明

LSTM 模型在测试集上的 loss 为 0.2707。拟合图如下所示：

6.3 涨跌分类准确率说明

在测试集上 benchmark 模型预测的第 50 天的闭市价是否大于真实前一天的闭市价为 benchmark 预测涨跌，在测试集上 lstm 模型预测的第 50

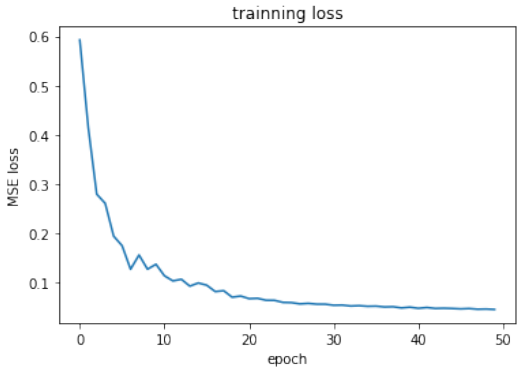


图 19: LSTM 模型训练 loss 图

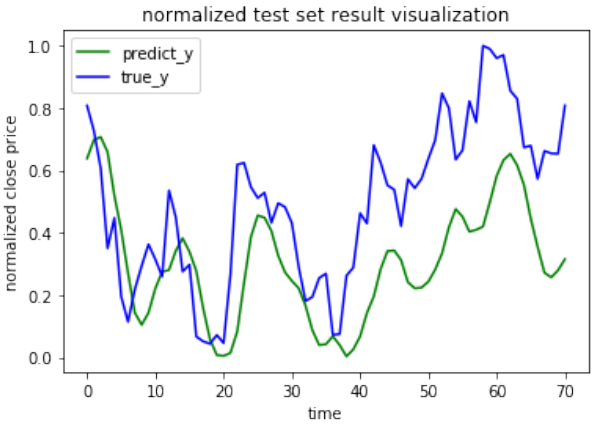


图 20: LSTM 模型测试集预测拟合图

天的闭市价是否大于真实前一天的闭市价为模型预测涨跌，真实的第 50 天的闭市价是否大于真实前一天的闭市价为真实涨跌。通过比较真实涨跌和两种预测涨跌即可得出两个模型在测试集上的分类准确率，最终结果 lstm 模型准确率为 54.93%，benchmark 模型准确率为 50.34%。

7 学习体会

在实现本次项目的过程中，我阅读了很多参考资料和 github 代码。网络上的代码质量参差不齐，在本篇报告里就不赘述了。我在参考文献中列出了两篇博客，都是在本次项目中让我对股票预测模型和 LSTM 模型理解深刻的博客。其中一篇详细的讲述了从机器学习方法预测股票走势到从深度学习方法预测股票走势的各种建模方法以及其利弊 [1]。正是这篇博客让我确定了 benchmark 模型的算法设计，最后的结果也恰如博客所说和公式所

证，对于预测未来一天的股票走势来说，这个最简单的机器学习方法恰是十分有效的。而另一篇博客深刻的讲解了 LSTM 模型的诞生，内部的结构和这样设计的意义，这让我有信心如果下次再遇到本身具有时间序列信息的数据时，能大致知道处理和从中获得信息的方法 [2]。但是，理解模型的设计方法和实际用模型来训练和预测数据还是有很大的不同的。特别是 LSTM 模型在不同的论文中它的内部结构有不同的细微的变形，如果像我这样在项目中使用的是第三方包封装好的模型的话（torch.nn.LSTM），不了解模型内部实现的细节还是不行的。这导致我在模型迭代后期不知道该对模型内部做出怎么样的调试才能将最终预测的点数提上去，还有就是对于模型的规模没有准确的评估。

附录 A　代码

```
data acquisition and preprocessing
import tushare as ts
import torch
import pandas as pd
print(f'tushare version:{ts.__version__}')
print(f'torch version:{torch.__version__}')
print(f'pandas version:{pd.__version__}')
tushare version:1.2.62
torch version:1.3.1
pandas version:0.20.3
class Args:
    def __init__(self):
        self.stock_code = '000001.SZ'
        self.startdate = '20100901'
        self.enddate = '20190901'
        self.predict_col='close'
        self.ratio = 0.2
        self.window_size = 100
        self.input_dim = 1
        self.hidden_dim = 32
        self.num_layers = 2
        self.output_dim = 1
        self.num_epochs = 100

args = Args()
ts.set_token('9608221f6e83cafa56f538f1d583e9802783b7589b042a8b62e04463')
pro = ts.pro_api()
df = pro.daily(ts_code=args.stock_code, start_date=args.startdate,
    end_date=args.enddate)
df1 = pd.DataFrame(df)
path = args.stock_code + '.csv'
df1.to_csv(path)
```

```
df1
ts_code trade_date  open  high  low close pre_close change  pct_chg vol
    amount
0 000001.SZ 20190830  14.29 14.39 14.10 14.16 14.13 0.03  0.2123  798500.57
    1.136120e+06
1 000001.SZ 20190829  14.22 14.24 14.08 14.13 14.27 -0.14 -0.9811 609034.34
    8.611775e+05
2 000001.SZ 20190828  14.26 14.28 14.05 14.27 14.31 -0.04 -0.2795 829657.68
    1.176329e+06
3 000001.SZ 20190827  14.36 14.48 14.24 14.31 14.25 0.06  0.4211
    1356713.37  1.948127e+06
4 000001.SZ 20190826  14.42 14.50 14.15 14.25 14.65 -0.40 -2.7304
    1415122.54  2.018498e+06
5 000001.SZ 20190823  14.37 14.74 14.35 14.65 14.31 0.34  2.3760
    1636867.68  2.388239e+06
6 000001.SZ 20190822  14.40 14.45 14.20 14.31 14.45 -0.14 -0.9689
    1353071.58  1.935208e+06
7 000001.SZ 20190821  14.87 14.89 14.38 14.45 14.99 -0.54 -3.6024
    1990131.95  2.902251e+06
8 000001.SZ 20190820  14.92 15.20 14.77 14.99 14.92 0.07  0.4692
    1921041.50  2.872304e+06
9 000001.SZ 20190819  14.91 14.94 14.52 14.92 14.90 0.02  0.1342
    2292956.13  3.368208e+06
10  000001.SZ 20190816  15.09 15.14 14.78 14.90 14.94 -0.04 -0.2677
    986902.93 1.474900e+06
11  000001.SZ 20190815  14.64 14.96 14.60 14.94 14.97 -0.03 -0.2004
    897376.26 1.333118e+06
12  000001.SZ 20190814  15.14 15.22 14.80 14.97 14.89 0.08  0.5373
    1360546.54  2.038261e+06
13  000001.SZ 20190813  15.00 15.08 14.74 14.89 15.12 -0.23 -1.5212
    1293736.44  1.925835e+06
14  000001.SZ 20190812  14.61 15.12 14.60 15.12 14.52 0.60  4.1322
    2733425.47  4.084498e+06
15  000001.SZ 20190809  14.55 14.85 14.43 14.52 14.38 0.14  0.9736
    2060575.13  3.021884e+06
16  000001.SZ 20190808  13.90 14.50 13.85 14.38 13.54 0.84  6.2038
    2330715.04  3.313302e+06
17  000001.SZ 20190807  13.49 13.64 13.37 13.54 13.37 0.17  1.2715
    793038.99 1.070737e+06
18  000001.SZ 20190806  13.10 13.46 13.03 13.37 13.35 0.02  0.1498
    882499.13 1.166971e+06
19  000001.SZ 20190805  13.60 13.64 13.27 13.35 13.74 -0.39 -2.8384
    893082.42 1.201885e+06
20  000001.SZ 20190802  13.77 13.88 13.66 13.74 14.10 -0.36 -2.5532
    969926.46 1.333401e+06
21  000001.SZ 20190801  14.06 14.19 13.94 14.10 14.13 -0.03 -0.2123
    527981.28 7.423083e+05
22  000001.SZ 20190731  14.30 14.32 14.08 14.13 14.37 -0.24 -1.6701
    634723.48 8.992279e+05
23  000001.SZ 20190730  14.31 14.55 14.29 14.37 14.29 0.08  0.5598
    796634.32 1.148582e+06
```

```
24  000001.SZ 20190729  14.25 14.45 14.18 14.29 14.23 0.06  0.4216
    715576.19 1.023262e+06
25  000001.SZ 20190726  14.18 14.25 14.08 14.23 14.20 0.03  0.2113
    635305.71 9.010741e+05
26  000001.SZ 20190725  13.92 14.27 13.85 14.20 13.88 0.32  2.3055
    1087826.41  1.534890e+06
27  000001.SZ 20190724  13.87 14.01 13.79 13.88 13.76 0.12  0.8721
    570276.22 7.933878e+05
28  000001.SZ 20190723  13.86 13.87 13.65 13.76 13.85 -0.09 -0.6498
    504004.23 6.923284e+05
29  000001.SZ 20190722  13.96 14.02 13.76 13.85 13.99 -0.14 -1.0007
    582283.30 8.072494e+05
... ... ... ... ... ... ... ... ... ... ... ...
2135  000001.SZ 20101026  19.00 19.25 18.55 18.66 18.91 -0.25 -1.3200
    422300.44 7.959873e+05
2136  000001.SZ 20101025  18.55 18.97 18.26 18.91 18.52 0.39  2.1100
    468305.75 8.744752e+05
2137  000001.SZ 20101022  18.55 18.81 18.33 18.52 18.85 -0.33 -1.7500
    538222.05 9.991661e+05
2138  000001.SZ 20101021  19.47 19.50 18.77 18.85 19.47 -0.62 -3.1800
    554999.68 1.052713e+06
2139  000001.SZ 20101020  19.15 19.98 19.15 19.47 19.55 -0.08 -0.4100
    739727.16 1.448846e+06
2140  000001.SZ 20101019  19.12 19.63 18.83 19.55 19.17 0.38  1.9800
    580719.26 1.107908e+06
2141  000001.SZ 20101018  19.40 20.06 19.10 19.17 19.15 0.02  0.1000
    1074081.95  2.102794e+06
2142  000001.SZ 20101015  18.20 19.38 18.18 19.15 18.34 0.81  4.4200
    1168938.21  2.220512e+06
2143  000001.SZ 20101014  18.65 19.29 18.31 18.34 18.54 -0.20 -1.0800
    1003749.25  1.889045e+06
2144  000001.SZ 20101013  17.99 18.55 17.97 18.54 17.97 0.57  3.1700
    825961.78 1.514395e+06
2145  000001.SZ 20101012  18.06 18.15 17.80 17.97 18.04 -0.07 -0.3900
    477594.48 8.568811e+05
2146  000001.SZ 20101011  17.32 18.36 17.32 18.04 17.22 0.82  4.7600
    874030.16 1.564392e+06
2147  000001.SZ 20101008  16.70 17.34 16.50 17.22 16.22 1.00  6.1700
    536715.85 9.148388e+05
2148  000001.SZ 20100929  16.22 16.43 16.11 16.22 16.26 -0.04 -0.2500
    213475.78 3.472608e+05
2149  000001.SZ 20100928  16.67 16.67 16.26 16.26 16.69 -0.43 -2.5800
    246526.46 4.048172e+05
2150  000001.SZ 20100927  16.64 16.75 16.43 16.69 16.61 0.08  0.4800
    188331.73 3.124534e+05
2151  000001.SZ 20100921  16.78 16.83 16.55 16.61 16.70 -0.09 -0.5400
    133992.27 2.232819e+05
2152  000001.SZ 20100920  16.67 16.94 16.51 16.70 16.65 0.05  0.3000
    197686.87 3.304790e+05
2153  000001.SZ 20100917  16.67 16.90 16.61 16.65 16.63 0.02  0.1200
    220792.99 3.698353e+05
```

```
2154  000001.SZ 20100916  17.00 17.14 16.55 16.63 16.92 -0.29 -1.7100
     286076.40 4.783042e+05
2155  000001.SZ 20100915  17.20 17.29 16.92 16.92 17.18 -0.26 -1.5100
     259235.60 4.438044e+05
2156  000001.SZ 20100914  17.15 17.36 17.09 17.18 17.09 0.09  0.5300
     299494.71 5.161963e+05
2157  000001.SZ 20100913  17.21 17.40 16.78 17.09 17.20 -0.11 -0.6400
     502228.27 8.567676e+05
2158  000001.SZ 20100910  17.16 17.41 17.00 17.20 17.22 -0.02 -0.1200
     251847.77 4.325192e+05
2159  000001.SZ 20100909  17.65 17.65 17.17 17.22 17.65 -0.43 -2.4400
     469897.64 8.169379e+05
2160  000001.SZ 20100908  17.91 17.91 17.51 17.65 18.04 -0.39 -2.1600
     490035.78 8.656646e+05
2161  000001.SZ 20100907  18.34 18.34 17.94 18.04 18.20 -0.16 -0.8800
     392852.63 7.112746e+05
2162  000001.SZ 20100906  17.86 18.39 17.77 18.20 17.76 0.44  2.4800
     609069.62 1.106419e+06
2163  000001.SZ 20100903  18.17 18.17 17.67 17.76 18.19 -0.43 -2.3600
     624129.22 1.110874e+06
2164  000001.SZ 20100902  19.00 19.00 17.89 18.19 17.51 0.68  3.8800
     1634542.10  2.984709e+06
2165 rows × 11 columns

df1 = df1[['trade_date','open', 'high', 'close', 'low', 'vol', 'change']]
df1 = df1.sort_values(by='trade_date')
df1
trade_date  open  high  close low vol change
2164  20100902  19.00 19.00 18.19 17.89 1634542.10  0.68
2163  20100903  18.17 18.17 17.76 17.67 624129.22 -0.43
2162  20100906  17.86 18.39 18.20 17.77 609069.62 0.44
2161  20100907  18.34 18.34 18.04 17.94 392852.63 -0.16
2160  20100908  17.91 17.91 17.65 17.51 490035.78 -0.39
2159  20100909  17.65 17.65 17.22 17.17 469897.64 -0.43
2158  20100910  17.16 17.41 17.20 17.00 251847.77 -0.02
2157  20100913  17.21 17.40 17.09 16.78 502228.27 -0.11
2156  20100914  17.15 17.36 17.18 17.09 299494.71 0.09
2155  20100915  17.20 17.29 16.92 16.92 259235.60 -0.26
2154  20100916  17.00 17.14 16.63 16.55 286076.40 -0.29
2153  20100917  16.67 16.90 16.65 16.61 220792.99 0.02
2152  20100920  16.67 16.94 16.70 16.51 197686.87 0.05
2151  20100921  16.78 16.83 16.61 16.55 133992.27 -0.09
2150  20100927  16.64 16.75 16.69 16.43 188331.73 0.08
2149  20100928  16.67 16.67 16.26 16.26 246526.46 -0.43
2148  20100929  16.22 16.43 16.22 16.11 213475.78 -0.04
2147  20101008  16.70 17.34 17.22 16.50 536715.85 1.00
2146  20101011  17.32 18.36 18.04 17.32 874030.16 0.82
2145  20101012  18.06 18.15 17.97 17.80 477594.48 -0.07
2144  20101013  17.99 18.55 18.54 17.97 825961.78 0.57
2143  20101014  18.65 19.29 18.34 18.31 1003749.25  -0.20
2142  20101015  18.20 19.38 19.15 18.18 1168938.21  0.81
```

```
2141  20101018  19.40 20.06 19.17 19.10 1074081.95  0.02
2140  20101019  19.12 19.63 19.55 18.83 580719.26 0.38
2139  20101020  19.15 19.98 19.47 19.15 739727.16 -0.08
2138  20101021  19.47 19.50 18.85 18.77 554999.68 -0.62
2137  20101022  18.55 18.81 18.52 18.33 538222.05 -0.33
2136  20101025  18.55 18.97 18.91 18.26 468305.75 0.39
2135  20101026  19.00 19.25 18.66 18.55 422300.44 -0.25
... ... ... ... ... ... ... ...
29  20190722  13.96 14.02 13.85 13.76 582283.30 -0.14
28  20190723  13.86 13.87 13.76 13.65 504004.23 -0.09
27  20190724  13.87 14.01 13.88 13.79 570276.22 0.12
26  20190725  13.92 14.27 14.20 13.85 1087826.41  0.32
25  20190726  14.18 14.25 14.23 14.08 635305.71 0.03
24  20190729  14.25 14.45 14.29 14.18 715576.19 0.06
23  20190730  14.31 14.55 14.37 14.29 796634.32 0.08
22  20190731  14.30 14.32 14.13 14.08 634723.48 -0.24
21  20190801  14.06 14.19 14.10 13.94 527981.28 -0.03
20  20190802  13.77 13.88 13.74 13.66 969926.46 -0.36
19  20190805  13.60 13.64 13.35 13.27 893082.42 -0.39
18  20190806  13.10 13.46 13.37 13.03 882499.13 0.02
17  20190807  13.49 13.64 13.54 13.37 793038.99 0.17
16  20190808  13.90 14.50 14.38 13.85 2330715.04  0.84
15  20190809  14.55 14.85 14.52 14.43 2060575.13  0.14
14  20190812  14.61 15.12 15.12 14.60 2733425.47  0.60
13  20190813  15.00 15.08 14.89 14.74 1293736.44  -0.23
12  20190814  15.14 15.22 14.97 14.80 1360546.54  0.08
11  20190815  14.64 14.96 14.94 14.60 897376.26 -0.03
10  20190816  15.09 15.14 14.90 14.78 986902.93 -0.04
9 20190819  14.91 14.94 14.92 14.52 2292956.13  0.02
8 20190820  14.92 15.20 14.99 14.77 1921041.50  0.07
7 20190821  14.87 14.89 14.45 14.38 1990131.95  -0.54
6 20190822  14.40 14.45 14.31 14.20 1353071.58  -0.14
5 20190823  14.37 14.74 14.65 14.35 1636867.68  0.34
4 20190826  14.42 14.50 14.25 14.15 1415122.54  -0.40
3 20190827  14.36 14.48 14.31 14.24 1356713.37  0.06
2 20190828  14.26 14.28 14.27 14.05 829657.68 -0.04
1 20190829  14.22 14.24 14.13 14.08 609034.34 -0.14
0 20190830  14.29 14.39 14.16 14.10 798500.57 0.03
2165 rows × 7 columns
```

```
from sklearn.preprocessing import MinMaxScaler

price = df1[[args.predict_col]]
scaler = MinMaxScaler(feature_range=(-1, 1))
price.loc[:,args.predict_col] =
     scaler.fit_transform(price[args.predict_col].values.reshape(-1,1))

import numpy as np

'''
    for the input stock,
```

```
    slice it for a lookback size and make one step further,
    ratio determines the size of train and test set,
    choose the last price of every slice as label
'''
def split_data(stock, ratio=0.2):
    data = stock
    data = np.array(data);
    test_set_size = int(np.round(ratio*data.shape[0]));
    train_set_size = data.shape[0] - (test_set_size);

    train_data = data[:train_set_size]
    test_data = data[train_set_size:]

    return [train_data,test_data]

train_data,test_data = split_data(price.values,args.ratio)
D:\Anaconda\envs\pytorch3.6\lib\site-packages\pandas\core\indexing.py:601:
    SettingWithCopyWarning:
A value is trying to be set on a copy of a slice from a DataFrame.
Try using .loc[row_indexer,col_indexer] = value instead

See the caveats in the documentation:
    http://pandas.pydata.org/pandas-docs/stable/indexing.html#indexing-view-versus-copy
  self.obj[item_labels[indexer[info_axis]]] = value
N = len(train_data)

run_avg_predictions = []
run_avg_x = []

mse_errors = []

running_mean = 0.0
run_avg_predictions.append(running_mean)

decay = 0.5

for pred_idx in range(1,N):
    running_mean = running_mean*decay + (1.0-decay)*train_data[pred_idx-1]
    run_avg_predictions.append(running_mean)
    mse_errors.append((run_avg_predictions[-1]-train_data[pred_idx])**2)
    run_avg_x.append(pred_idx)

print('MSE error for EMA averaging: %.5f'%(0.5*np.mean(mse_errors)))
MSE error for EMA averaging: 0.00154
import matplotlib.pyplot as plt
%matplotlib inline


plt.figure(figsize = (18,9))
plt.plot(range(0,N),price.iloc[:N,0],color='b',label='True')
plt.plot(range(0,N),run_avg_predictions,color='orange', label='Prediction')
```

```
plt.xlabel('Date')
plt.ylabel('Mid Price')
plt.legend(fontsize=18)
plt.show()

N = len(test_data)

run_avg_predictions = []
run_avg_x = []

mse_errors = []

running_mean = 0.0
run_avg_predictions.append(running_mean)

decay = 0.5

for pred_idx in range(1,N):
    running_mean = running_mean*decay + (1.0-decay)*test_data[pred_idx-1]
    run_avg_predictions.append(running_mean)
    mse_errors.append((run_avg_predictions[-1]-test_data[pred_idx])**2)
    run_avg_x.append(pred_idx)

print('MSE error for EMA averaging: %.5f'%(0.5*np.mean(mse_errors)))
MSE error for EMA averaging: 0.00078


data acquisition and preprocessing
import tushare as ts
import torch
import pandas as pd
print(f'tushare version:{ts.__version__}')
print(f'torch version:{torch.__version__}')
print(f'pandas version:{pd.__version__}')
tushare version:1.2.62
torch version:1.3.1
pandas version:0.20.3
class Args:
    def __init__(self):
        self.stock_code = '000001.SZ'
        self.startdate = '20100901'
        self.enddate = '20190901'
        self.predict_col='close'
        self.ratio = 0.2
        self.window_size = 100
        self.input_dim = 1
        self.hidden_dim = 32
        self.num_layers = 2
        self.output_dim = 1
        self.num_epochs = 100
```

```
args = Args()
ts.set_token('9608221f6e83cafa56f538f1d583e9802783b7589b042a8b62e04463')
pro = ts.pro_api()
df = pro.daily(ts_code=args.stock_code, start_date=args.startdate,
    end_date=args.enddate)
df1 = pd.DataFrame(df)
path = args.stock_code + '.csv'
df1.to_csv(path)
df1
ts_code trade_date  open  high  low close pre_close change  pct_chg vol
    amount
0 000001.SZ 20190830  14.29 14.39 14.10 14.16 14.13 0.03  0.2123  798500.57
    1.136120e+06
1 000001.SZ 20190829  14.22 14.24 14.08 14.13 14.27 -0.14 -0.9811 609034.34
    8.611775e+05
2 000001.SZ 20190828  14.26 14.28 14.05 14.27 14.31 -0.04 -0.2795 829657.68
    1.176329e+06
3 000001.SZ 20190827  14.36 14.48 14.24 14.31 14.25 0.06  0.4211
    1356713.37  1.948127e+06
4 000001.SZ 20190826  14.42 14.50 14.15 14.25 14.65 -0.40 -2.7304
    1415122.54  2.018498e+06
5 000001.SZ 20190823  14.37 14.74 14.35 14.65 14.31 0.34  2.3760
    1636867.68  2.388239e+06
6 000001.SZ 20190822  14.40 14.45 14.20 14.31 14.45 -0.14 -0.9689
    1353071.58  1.935208e+06
7 000001.SZ 20190821  14.87 14.89 14.38 14.45 14.99 -0.54 -3.6024
    1990131.95  2.902251e+06
8 000001.SZ 20190820  14.92 15.20 14.77 14.99 14.92 0.07  0.4692
    1921041.50  2.872304e+06
9 000001.SZ 20190819  14.91 14.94 14.52 14.92 14.90 0.02  0.1342
    2292956.13  3.368208e+06
10  000001.SZ 20190816  15.09 15.14 14.78 14.90 14.94 -0.04 -0.2677
    986902.93 1.474900e+06
11  000001.SZ 20190815  14.64 14.96 14.60 14.94 14.97 -0.03 -0.2004
    897376.26 1.333118e+06
12  000001.SZ 20190814  15.14 15.22 14.80 14.97 14.89 0.08  0.5373
    1360546.54  2.038261e+06
13  000001.SZ 20190813  15.00 15.08 14.74 14.89 15.12 -0.23 -1.5212
    1293736.44  1.925835e+06
14  000001.SZ 20190812  14.61 15.12 14.60 15.12 14.52 0.60  4.1322
    2733425.47  4.084498e+06
15  000001.SZ 20190809  14.55 14.85 14.43 14.52 14.38 0.14  0.9736
    2060575.13  3.021884e+06
16  000001.SZ 20190808  13.90 14.50 13.85 14.38 13.54 0.84  6.2038
    2330715.04  3.313302e+06
17  000001.SZ 20190807  13.49 13.64 13.37 13.54 13.37 0.17  1.2715
    793038.99 1.070737e+06
18  000001.SZ 20190806  13.10 13.46 13.03 13.37 13.35 0.02  0.1498
    882499.13 1.166971e+06
19  000001.SZ 20190805  13.60 13.64 13.27 13.35 13.74 -0.39 -2.8384
    893082.42 1.201885e+06
```

```
20  000001.SZ 20190802  13.77 13.88 13.66 13.74 14.10 -0.36 -2.5532
    969926.46 1.333401e+06
21  000001.SZ 20190801  14.06 14.19 13.94 14.10 14.13 -0.03 -0.2123
    527981.28 7.423083e+05
22  000001.SZ 20190731  14.30 14.32 14.08 14.13 14.37 -0.24 -1.6701
    634723.48 8.992279e+05
23  000001.SZ 20190730  14.31 14.55 14.29 14.37 14.29 0.08  0.5598
    796634.32 1.148582e+06
24  000001.SZ 20190729  14.25 14.45 14.18 14.29 14.23 0.06  0.4216
    715576.19 1.023262e+06
25  000001.SZ 20190726  14.18 14.25 14.08 14.23 14.20 0.03  0.2113
    635305.71 9.010741e+05
26  000001.SZ 20190725  13.92 14.27 13.85 14.20 13.88 0.32  2.3055
    1087826.41  1.534890e+06
27  000001.SZ 20190724  13.87 14.01 13.79 13.88 13.76 0.12  0.8721
    570276.22 7.933878e+05
28  000001.SZ 20190723  13.86 13.87 13.65 13.76 13.85 -0.09 -0.6498
    504004.23 6.923284e+05
29  000001.SZ 20190722  13.96 14.02 13.76 13.85 13.99 -0.14 -1.0007
    582283.30 8.072494e+05
... ... ... ... ... ... ... ... ... ... ... ...
2135  000001.SZ 20101026  19.00 19.25 18.55 18.66 18.91 -0.25 -1.3200
    422300.44 7.959873e+05
2136  000001.SZ 20101025  18.55 18.97 18.26 18.91 18.52 0.39  2.1100
    468305.75 8.744752e+05
2137  000001.SZ 20101022  18.55 18.81 18.33 18.52 18.85 -0.33 -1.7500
    538222.05 9.991661e+05
2138  000001.SZ 20101021  19.47 19.50 18.77 18.85 19.47 -0.62 -3.1800
    554999.68 1.052713e+06
2139  000001.SZ 20101020  19.15 19.98 19.15 19.47 19.55 -0.08 -0.4100
    739727.16 1.448846e+06
2140  000001.SZ 20101019  19.12 19.63 18.83 19.55 19.17 0.38  1.9800
    580719.26 1.107908e+06
2141  000001.SZ 20101018  19.40 20.06 19.10 19.17 19.15 0.02  0.1000
    1074081.95  2.102794e+06
2142  000001.SZ 20101015  18.20 19.38 18.18 19.15 18.34 0.81  4.4200
    1168938.21  2.220512e+06
2143  000001.SZ 20101014  18.65 19.29 18.31 18.34 18.54 -0.20 -1.0800
    1003749.25  1.889045e+06
2144  000001.SZ 20101013  17.99 18.55 17.97 18.54 17.97 0.57  3.1700
    825961.78 1.514395e+06
2145  000001.SZ 20101012  18.06 18.15 17.80 17.97 18.04 -0.07 -0.3900
    477594.48 8.568811e+05
2146  000001.SZ 20101011  17.32 18.36 17.32 18.04 17.22 0.82  4.7600
    874030.16 1.564392e+06
2147  000001.SZ 20101008  16.70 17.34 16.50 17.22 16.22 1.00  6.1700
    536715.85 9.148388e+05
2148  000001.SZ 20100929  16.22 16.43 16.11 16.22 16.26 -0.04 -0.2500
    213475.78 3.472608e+05
2149  000001.SZ 20100928  16.67 16.67 16.26 16.26 16.69 -0.43 -2.5800
    246526.46 4.048172e+05
```

```
2150  000001.SZ 20100927  16.64 16.75 16.43 16.69 16.61 0.08  0.4800
     188331.73 3.124534e+05
2151  000001.SZ 20100921  16.78 16.83 16.55 16.61 16.70 -0.09 -0.5400
     133992.27 2.232819e+05
2152  000001.SZ 20100920  16.67 16.94 16.51 16.70 16.65 0.05  0.3000
     197686.87 3.304790e+05
2153  000001.SZ 20100917  16.67 16.90 16.61 16.65 16.63 0.02  0.1200
     220792.99 3.698353e+05
2154  000001.SZ 20100916  17.00 17.14 16.55 16.63 16.92 -0.29 -1.7100
     286076.40 4.783042e+05
2155  000001.SZ 20100915  17.20 17.29 16.92 16.92 17.18 -0.26 -1.5100
     259235.60 4.438044e+05
2156  000001.SZ 20100914  17.15 17.36 17.09 17.18 17.09 0.09  0.5300
     299494.71 5.161963e+05
2157  000001.SZ 20100913  17.21 17.40 16.78 17.09 17.20 -0.11 -0.6400
     502228.27 8.567676e+05
2158  000001.SZ 20100910  17.16 17.41 17.00 17.20 17.22 -0.02 -0.1200
     251847.77 4.325192e+05
2159  000001.SZ 20100909  17.65 17.65 17.17 17.22 17.65 -0.43 -2.4400
     469897.64 8.169379e+05
2160  000001.SZ 20100908  17.91 17.91 17.51 17.65 18.04 -0.39 -2.1600
     490035.78 8.656646e+05
2161  000001.SZ 20100907  18.34 18.34 17.94 18.04 18.20 -0.16 -0.8800
     392852.63 7.112746e+05
2162  000001.SZ 20100906  17.86 18.39 17.77 18.20 17.76 0.44  2.4800
     609069.62 1.106419e+06
2163  000001.SZ 20100903  18.17 18.17 17.67 17.76 18.19 -0.43 -2.3600
     624129.22 1.110874e+06
2164  000001.SZ 20100902  19.00 19.00 17.89 18.19 17.51 0.68  3.8800
     1634542.10  2.984709e+06
2165 rows × 11 columns

df1 = df1[['trade_date','open', 'high', 'close', 'low', 'vol', 'change']]
df1 = df1.sort_values(by='trade_date')
df1
trade_date  open  high  close low vol change
2164  20100902  19.00 19.00 18.19 17.89 1634542.10  0.68
2163  20100903  18.17 18.17 17.76 17.67 624129.22 -0.43
2162  20100906  17.86 18.39 18.20 17.77 609069.62 0.44
2161  20100907  18.34 18.34 18.04 17.94 392852.63 -0.16
2160  20100908  17.91 17.91 17.65 17.51 490035.78 -0.39
2159  20100909  17.65 17.65 17.22 17.17 469897.64 -0.43
2158  20100910  17.16 17.41 17.20 17.00 251847.77 -0.02
2157  20100913  17.21 17.40 17.09 16.78 502228.27 -0.11
2156  20100914  17.15 17.36 17.18 17.09 299494.71 0.09
2155  20100915  17.20 17.29 16.92 16.92 259235.60 -0.26
2154  20100916  17.00 17.14 16.63 16.55 286076.40 -0.29
2153  20100917  16.67 16.90 16.65 16.61 220792.99 0.02
2152  20100920  16.67 16.94 16.70 16.51 197686.87 0.05
2151  20100921  16.78 16.83 16.61 16.55 133992.27 -0.09
2150  20100927  16.64 16.75 16.69 16.43 188331.73 0.08
```

```
2149  20100928  16.67 16.67 16.26 16.26 246526.46 -0.43
2148  20100929  16.22 16.43 16.22 16.11 213475.78 -0.04
2147  20101008  16.70 17.34 17.22 16.50 536715.85 1.00
2146  20101011  17.32 18.36 18.04 17.32 874030.16 0.82
2145  20101012  18.06 18.15 17.97 17.80 477594.48 -0.07
2144  20101013  17.99 18.55 18.54 17.97 825961.78 0.57
2143  20101014  18.65 19.29 18.34 18.31 1003749.25  -0.20
2142  20101015  18.20 19.38 19.15 18.18 1168938.21  0.81
2141  20101018  19.40 20.06 19.17 19.10 1074081.95  0.02
2140  20101019  19.12 19.63 19.55 18.83 580719.26 0.38
2139  20101020  19.15 19.98 19.47 19.15 739727.16 -0.08
2138  20101021  19.47 19.50 18.85 18.77 554999.68 -0.62
2137  20101022  18.55 18.81 18.52 18.33 538222.05 -0.33
2136  20101025  18.55 18.97 18.91 18.26 468305.75 0.39
2135  20101026  19.00 19.25 18.66 18.55 422300.44 -0.25
... ... ... ... ... ... ... ...
29  20190722  13.96 14.02 13.85 13.76 582283.30 -0.14
28  20190723  13.86 13.87 13.76 13.65 504004.23 -0.09
27  20190724  13.87 14.01 13.88 13.79 570276.22 0.12
26  20190725  13.92 14.27 14.20 13.85 1087826.41  0.32
25  20190726  14.18 14.25 14.23 14.08 635305.71 0.03
24  20190729  14.25 14.45 14.29 14.18 715576.19 0.06
23  20190730  14.31 14.55 14.37 14.29 796634.32 0.08
22  20190731  14.30 14.32 14.13 14.08 634723.48 -0.24
21  20190801  14.06 14.19 14.10 13.94 527981.28 -0.03
20  20190802  13.77 13.88 13.74 13.66 969926.46 -0.36
19  20190805  13.60 13.64 13.35 13.27 893082.42 -0.39
18  20190806  13.10 13.46 13.37 13.03 882499.13 0.02
17  20190807  13.49 13.64 13.54 13.37 793038.99 0.17
16  20190808  13.90 14.50 14.38 13.85 2330715.04  0.84
15  20190809  14.55 14.85 14.52 14.43 2060575.13  0.14
14  20190812  14.61 15.12 15.12 14.60 2733425.47  0.60
13  20190813  15.00 15.08 14.89 14.74 1293736.44  -0.23
12  20190814  15.14 15.22 14.97 14.80 1360546.54  0.08
11  20190815  14.64 14.96 14.94 14.60 897376.26 -0.03
10  20190816  15.09 15.14 14.90 14.78 986902.93 -0.04
9 20190819  14.91 14.94 14.92 14.52 2292956.13  0.02
8 20190820  14.92 15.20 14.99 14.77 1921041.50  0.07
7 20190821  14.87 14.89 14.45 14.38 1990131.95  -0.54
6 20190822  14.40 14.45 14.31 14.20 1353071.58  -0.14
5 20190823  14.37 14.74 14.65 14.35 1636867.68  0.34
4 20190826  14.42 14.50 14.25 14.15 1415122.54  -0.40
3 20190827  14.36 14.48 14.31 14.24 1356713.37  0.06
2 20190828  14.26 14.28 14.27 14.05 829657.68 -0.04
1 20190829  14.22 14.24 14.13 14.08 609034.34 -0.14
0 20190830  14.29 14.39 14.16 14.10 798500.57 0.03
2165 rows × 7 columns
```

```
from sklearn.preprocessing import MinMaxScaler

price = df1[[args.predict_col]]
```

```
scaler = MinMaxScaler(feature_range=(-1, 1))
price.loc[:,args.predict_col] =
    scaler.fit_transform(price[args.predict_col].values.reshape(-1,1))
D:\Anaconda\envs\pytorch3.6\lib\site-packages\pandas\core\indexing.py:601:
    SettingWithCopyWarning:
A value is trying to be set on a copy of a slice from a DataFrame.
Try using .loc[row_indexer,col_indexer] = value instead

See the caveats in the documentation:
    http://pandas.pydata.org/pandas-docs/stable/indexing.html#indexing-view-versus-copy
  self.obj[item_labels[indexer[info_axis]]] = value
price
close
2164  0.227041
2163  0.172194
2162  0.228316
2161  0.207908
2160  0.158163
2159  0.103316
2158  0.100765
2157  0.086735
2156  0.098214
2155  0.065051
2154  0.028061
2153  0.030612
2152  0.036990
2151  0.025510
2150  0.035714
2149  -0.019133
2148  -0.024235
2147  0.103316
2146  0.207908
2145  0.198980
2144  0.271684
2143  0.246173
2142  0.349490
2141  0.352041
2140  0.400510
2139  0.390306
2138  0.311224
2137  0.269133
2136  0.318878
2135  0.286990
...  ...
29  -0.326531
28  -0.338010
27  -0.322704
26  -0.281888
25  -0.278061
24  -0.270408
23  -0.260204
```

```
22  -0.290816
21  -0.294643
20  -0.340561
19  -0.390306
18  -0.387755
17  -0.366071
16  -0.258929
15  -0.241071
14  -0.164541
13  -0.193878
12  -0.183673
11  -0.187500
10  -0.192602
9 -0.190051
8 -0.181122
7 -0.250000
6 -0.267857
5 -0.224490
4 -0.275510
3 -0.267857
2 -0.272959
1 -0.290816
0 -0.286990
2165 rows × 1 columns
```

```python
import numpy as np

'''
    for the input stock,
    slice it for a lookback size and make one step further,
    ratio determines the size of train and test set,
    choose the last price of every slice as label
'''
def split_data(stock, lookback, ratio=0.2):
    data_raw = stock
    assert type(data_raw)==np.ndarray,f'{type(data_raw)}'
    data = []

    # create all possible sequences of length seq_len
    for index in range(len(data_raw) - lookback):
        data.append(data_raw[index: index + lookback])

    data = np.array(data);
    test_set_size = int(np.round(ratio*data.shape[0]));
    train_set_size = data.shape[0] - (test_set_size);

    x_train = data[:train_set_size,:-1,:]
    y_train = data[:train_set_size,-1,:]

    x_test = data[train_set_size:,:-1]
    y_test = data[train_set_size:,-1,:]
```

```
    return [x_train, y_train, x_test, y_test]


lookback = args.window_size # choose sequence length
x_train, y_train, x_test, y_test = split_data(price.values,
    lookback,args.ratio)
x_train.shape,y_train.shape,x_test.shape,y_test.shape
((1652, 99, 1), (1652, 1), (413, 99, 1), (413, 1))
import torch
import torch.nn as nn
x_train = torch.from_numpy(x_train).type(torch.Tensor)
x_test = torch.from_numpy(x_test).type(torch.Tensor)
y_train_lstm = torch.from_numpy(y_train).type(torch.Tensor)
y_test_lstm = torch.from_numpy(y_test).type(torch.Tensor)
y_train_gru = torch.from_numpy(y_train).type(torch.Tensor)
y_test_gru = torch.from_numpy(y_test).type(torch.Tensor)
train model
'''
    create lstm model with nn.lstm receiving batch-first data
    and fully connected layer as model head
'''
class LSTM(nn.Module):
    def __init__(self, input_dim, hidden_dim, num_layers, output_dim):
        super(LSTM, self).__init__()
        self.hidden_dim = hidden_dim
        self.num_layers = num_layers

        self.lstm = nn.LSTM(input_dim, hidden_dim, num_layers,
            batch_first=True)
        self.fc = nn.Linear(hidden_dim, output_dim)
    def forward(self, x):
        h0 = torch.zeros(self.num_layers, x.size(0),
            self.hidden_dim).requires_grad_()
        c0 = torch.zeros(self.num_layers, x.size(0),
            self.hidden_dim).requires_grad_()
        out, (hn, cn) = self.lstm(x, (h0.detach(), c0.detach()))
        out = self.fc(out[:, -1, :])
        return out
model = LSTM(input_dim=args.input_dim, hidden_dim=args.hidden_dim,
    output_dim=args.output_dim, num_layers=args.num_layers)
criterion = torch.nn.MSELoss(reduction='mean')
optimiser = torch.optim.Adam(model.parameters(), lr=0.01)
import time
hist = np.zeros(args.num_epochs)
start_time = time.time()
lstm = []
for t in range(args.num_epochs):
    y_train_pred = model(x_train)
    loss = criterion(y_train_pred, y_train_lstm)
    print("Epoch ", t, "MSE: ", loss.item())
```

```
    hist[t] = loss.item()
    optimiser.zero_grad()
    loss.backward()
    optimiser.step()

training_time = time.time()-start_time
print("Training time: {}".format(training_time))
Epoch  0 MSE:  0.419903963804245
Epoch  1 MSE:  0.2619572877883911
Epoch  2 MSE:  0.15795500576496124
Epoch  3 MSE:  0.21542863547801971
Epoch  4 MSE:  0.13975706696510315
Epoch  5 MSE:  0.11317011713981628
Epoch  6 MSE:  0.11205611377954483
Epoch  7 MSE:  0.10496336221694946
Epoch  8 MSE:  0.07866968959569931
Epoch  9 MSE:  0.03687012940645218
Epoch  10 MSE:  0.05467960983514786
Epoch  11 MSE:  0.03630007430911064
Epoch  12 MSE:  0.015137618407607079
Epoch  13 MSE:  0.024544144049286842
Epoch  14 MSE:  0.032580938190221786
Epoch  15 MSE:  0.026643022894859314
Epoch  16 MSE:  0.01580948941409588
Epoch  17 MSE:  0.011203901842236519
Epoch  18 MSE:  0.014602589420974255
Epoch  19 MSE:  0.018984248861670494
Epoch  20 MSE:  0.018489206209778786
Epoch  21 MSE:  0.014672549441456795
Epoch  22 MSE:  0.0117790000513196
Epoch  23 MSE:  0.011444010771811008
Epoch  24 MSE:  0.012616965919733047
Epoch  25 MSE:  0.013577691279351711
Epoch  26 MSE:  0.01327303797006607
Epoch  27 MSE:  0.011721266433596611
Epoch  28 MSE:  0.009867796674370766
Epoch  29 MSE:  0.008903704583644867
Epoch  30 MSE:  0.00919144693762064
Epoch  31 MSE:  0.009802344255149364
Epoch  32 MSE:  0.009538626298308372
Epoch  33 MSE:  0.008365807123482227
Epoch  34 MSE:  0.007320573553442955
Epoch  35 MSE:  0.007169515360146761
Epoch  36 MSE:  0.007669162470847368
Epoch  37 MSE:  0.008025415241718292
Epoch  38 MSE:  0.0077319978736341
Epoch  39 MSE:  0.006978856399655342
Epoch  40 MSE:  0.006403807550668716
Epoch  41 MSE:  0.00642471294850111
Epoch  42 MSE:  0.006770658306777477
Epoch  43 MSE:  0.006818165071308613
```

```
Epoch  44 MSE:  0.006396235898137093
Epoch  45 MSE:  0.005939954426139593
Epoch  46 MSE:  0.0058336639776825905
Epoch  47 MSE:  0.005963372532278299
Epoch  48 MSE:  0.005973777733743191
Epoch  49 MSE:  0.005728844553232193
Epoch  50 MSE:  0.005422933492809534
Epoch  51 MSE:  0.005307021550834179
Epoch  52 MSE:  0.005383350420743227
Epoch  53 MSE:  0.005426882766187191
Epoch  54 MSE:  0.005294414237141609
Epoch  55 MSE:  0.005096082575619221
Epoch  56 MSE:  0.0050134481862187386
Epoch  57 MSE:  0.005058319773525
Epoch  58 MSE:  0.005088934674859047
Epoch  59 MSE:  0.005008184816688299
Epoch  60 MSE:  0.004872514400631189
Epoch  61 MSE:  0.004798537120223045
Epoch  62 MSE:  0.0048060668632388115
Epoch  63 MSE:  0.004804989788681269
Epoch  64 MSE:  0.004734535235911608
Epoch  65 MSE:  0.004644792061299086
Epoch  66 MSE:  0.004608266521245241
Epoch  67 MSE:  0.0046132770366966724
Epoch  68 MSE:  0.004594662692397833
Epoch  69 MSE:  0.004535548854619265
Epoch  70 MSE:  0.004485319368541241
Epoch  71 MSE:  0.004477391019463539
Epoch  72 MSE:  0.0044784643687307835
Epoch  73 MSE:  0.004446892533451319
Epoch  74 MSE:  0.004399052355438471
Epoch  75 MSE:  0.00437491200864315
Epoch  76 MSE:  0.004371159709990025
Epoch  77 MSE:  0.004353496711701155
Epoch  78 MSE:  0.00431539211422205
Epoch  79 MSE:  0.004283997695893049
Epoch  80 MSE:  0.004272155463695526
Epoch  81 MSE:  0.004260445013642311
Epoch  82 MSE:  0.0042343745008111
Epoch  83 MSE:  0.004207186866551638
Epoch  84 MSE:  0.004192489665001631
Epoch  85 MSE:  0.0041814944706857204
Epoch  86 MSE:  0.00416190130636096
Epoch  87 MSE:  0.004138995427638292
Epoch  88 MSE:  0.004123304039239883
Epoch  89 MSE:  0.004111429210752249
Epoch  90 MSE:  0.004094350151717663
Epoch  91 MSE:  0.004074547439813614
Epoch  92 MSE:  0.004059859085828066
Epoch  93 MSE:  0.004048568662256002
Epoch  94 MSE:  0.004034087061882019
```

```
Epoch  95 MSE:  0.004017730709165335
Epoch  96 MSE:  0.004005121532827616
Epoch  97 MSE:  0.003995091654360294
Epoch  98 MSE:  0.003982664551585913
Epoch  99 MSE:  0.003968784585595131
Training time: 150.31389045715332
evaluate model
with torch.no_grad():
    y_test_pred = model(x_test)
    loss = criterion(y_test_pred, y_test_lstm)
loss
tensor(0.0022)
import matplotlib.pyplot as plt
%matplotlib inline
final_y_train_pred = model(x_train)



fig =plt.figure(figsize=(24,8))
'''-------------------------------------------------normalized data
    visualization-------------------------------------------------'''
plt.subplot(121)
t_train = np.arange(0,len(y_train_lstm))
plt.plot(t_train,final_y_train_pred.detach(),color='g',label='predict_y_train')
plt.plot(t_train,y_train_lstm,color='b',label='true_y_train')

t_test = np.arange(len(y_train_lstm),len(y_train_lstm)+len(y_test_lstm))
plt.plot(t_test,y_test_pred.detach(),color='r',label='predict_y_test')
plt.plot(t_test,y_test_lstm,color='b',label='true_y_test')

plt.title('normalized train & test result visualization')
plt.xlabel('time')
plt.ylabel('normalized close price')
plt.legend()

'''-------------------------------------------------real data
    visualization-------------------------------------------------'''
plt.subplot(122)
real_y_train = scaler.inverse_transform(y_train_lstm)
real_y_train_pred = scaler.inverse_transform(final_y_train_pred.detach())
real_y_test = scaler.inverse_transform(y_test_lstm)
real_y_test_pred = scaler.inverse_transform(y_test_pred.detach())

t_train = np.arange(0,len(y_train_lstm))
plt.plot(t_train,real_y_train_pred,color='g',label='predict_y_train')
plt.plot(t_train,real_y_train,color='b',label='true_y_train')

t_test = np.arange(len(y_train_lstm),len(y_train_lstm)+len(y_test_lstm))
plt.plot(t_test,real_y_test_pred,color='r',label='predict_y_test')
plt.plot(t_test,real_y_test,color='b',label='true_y_test')
```

```
plt.title('real train & test result visualization')
plt.xlabel('time')
plt.ylabel('real close price')
plt.legend()
<matplotlib.legend.Legend at 0x21b02153da0>

plt.plot(hist)
plt.title('trainning loss')
plt.xlabel('epoch')
plt.ylabel('MSE loss')
Text(0, 0.5, 'MSE loss')

test model
df = pro.daily(ts_code=args.stock_code, start_date=args.enddate,
    end_date='20201220')
df2 = df[['trade_date','open', 'high', 'close', 'low', 'vol', 'change']]
df2 = df2.sort_values(by='trade_date')

# from sklearn.preprocessing import MinMaxScaler

price = df2[[args.predict_col]]
# scaler = MinMaxScaler(feature_range=(-1, 1))
price.loc[:,args.predict_col] =
    scaler.fit_transform(price[args.predict_col].values.reshape(-1,1))
D:\Anaconda\envs\pytorch3.6\lib\site-packages\pandas\core\indexing.py:601:
    SettingWithCopyWarning:
A value is trying to be set on a copy of a slice from a DataFrame.
Try using .loc[row_indexer,col_indexer] = value instead

See the caveats in the documentation:
    http://pandas.pydata.org/pandas-docs/stable/indexing.html#indexing-view-versus-copy
  self.obj[item_labels[indexer[info_axis]]] = value
price
close
314 -0.417722
313 -0.455696
312 -0.420253
311 -0.384810
310 -0.326582
309 -0.356962
308 -0.392405
307 -0.389873
306 -0.359494
305 -0.417722
304 -0.470886
303 -0.427848
302 -0.318987
301 -0.192405
300 -0.182278
299 -0.232911
298 -0.088608
```

```
297 -0.098734
296 -0.050633
295 -0.129114
294 0.025316
293 0.037975
292 0.035443
291 0.179747
290 0.283544
289 0.273418
288 0.174684
287 0.151899
286 0.103797
285 0.200000
... ...
29  0.440506
28  0.508861
27  0.432911
26  0.394937
25  0.273418
24  0.321519
23  0.437975
22  0.597468
21  0.696203
20  0.698734
19  0.891139
18  0.825316
17  0.749367
16  0.860759
15  0.911392
14  0.921519
13  1.000000
12  0.893671
11  0.870886
10  0.810127
9 0.711392
8 0.660759
7 0.660759
6 0.645570
5 0.612658
4 0.703797
3 0.668354
2 0.736709
1 0.721519
0 0.572152
315 rows × 1 columns
```

```
def test_data_process(stock, lookback):
    data_raw = stock
    assert type(data_raw)==np.ndarray,f'{type(data_raw)}'
    data = []
```

```
    # create all possible sequences of length seq_len
    for index in range(len(data_raw) - lookback):
        data.append(data_raw[index: index + lookback])

    data = np.array(data);
    print(data.shape)
    x = data[:,:-1,:]
    y = data[:,-1,:]
    return [x,y]

x,y = test_data_process(price.values,args.window_size)
x.shape,y.shape
(215, 100, 1)
((215, 99, 1), (215, 1))
x = torch.from_numpy(x).type(torch.Tensor)
y = torch.from_numpy(y).type(torch.Tensor)

with torch.no_grad():
    y_pred = model(x)
    loss = criterion(y_pred,y)
loss
tensor(0.0145)
t_train = np.arange(0,len(y))
plt.plot(t_train, y_pred.detach(),color='g',label='predict_y')
plt.plot(t_train, y.detach(),color='b',label='true_y')

plt.title('normalized test set result visualization')
plt.xlabel('time')
plt.ylabel('normalized close price')
plt.legend()
<matplotlib.legend.Legend at 0x21b02251588>
```

参考文献

[1] Thushan Ganegedara, *Stock Market Predictions with LSTM in Python*, https://www.datacamp.com/community/tutorials/lstm-python-stock-market, Jan.2020

[2] colah's blog, *Understanding LSTM Neural Networks*, http://colah.github.io/posts/2015-08-Understanding-LSTMs/, Aug.2015

[3] Himanshu Sharma, *Technical Analysis of Stocks using TA-Lib*, https://towardsdatascience.com/technical-analysis-of-stocks-using-ta-lib-305614165051, Sep.2020

[4] Ishan Shah, Rekhit Pachanekar. *How to install Ta-Lib in Python*, https://blog.quantinsti.com/install-ta-lib-python/, Dec.2019