如何利用机器学习进行商品期货的高频交易

数据准备工作

In [1]:
```
# 安装 akshare 第三方库
# 不输出 Output
```

In [2]:
```
%%capture
! pip install akshare  --upgrade
```

In [3]:
```
%%capture
# 如何利用机器学习进行商品期货的高频交易

# 导入数据包 :
import pandas as pd
import numpy as np
import akshare as ak
import matplotlib.pyplot as plt
import matplotlib.ticker as ticker

from sklearn.preprocessing import MaxAbsScaler
from sklearn.model_selection import train_test_split
from sklearn import metrics
from sklearn import svm

# pandas DataFrame打印输出列名对齐 (中文列名) :
pd.set_option('display.unicode.ambiguous_as_wide', True)
pd.set_option('display.unicode.east_asian_width', True)

"""
# 导入数据 :
"""

# 接口: futures_zh_minute_sina

# 目标地址: http://vip.stock.finance.sina.com.cn/quotes_service/view/qihuohangqing.html#titlePos_3

# 描述: 新浪财经-期货-分时数据

# 限量: 单次返回指定 symbol 和 period 的分时数据

"""
# 商品期货代码的编码规则: 商品期货的编码规则:“品种”+合约到期年份+合约到期月份
```

```
# 上海期货交易所
# RB2301 : 螺纹钢2301
# 螺纹钢期货交易时间
# 日盘交易时间为交易日 上午9:00--11:30 ; 下午13:30--15:00
# 上午10点15分至10点30分，中间15分钟为盘中休息时间不可交易。
# 夜盘交易时间为交易日 晚上21:00--23:00

RB_df = ak.futures_zh_minute_sina(symbol="RB2301", period="1")
# "1": "1分钟"
print("\n")
print(RB_df)

# 将datetime这一列作为index
RB_df.set_index(["datetime"], inplace=True)

# 简单的线形图
# 使用pandas的plot()函数可以直接绘制线形图 :
# RB_df['open'].plot(color='C1',title = "open")
# RB_df['high'].plot(color='C2',title = "high")
# RB_df['low'].plot(color='C3',title = "low")
# RB_df['close'].plot(color='C4',title = "close")
# RB_df['volume'].plot(color='C5',title = "volume")
# RB_df['hold'].plot(color='C5',title = "hold")

# 数据存储
from pathlib import Path
filepath = Path('C:/Users/c4780/Desktop/desktop/data_RB2301.csv')
filepath.parent.mkdir(parents=True, exist_ok=True)
RB_df.to_csv(filepath)
"""
```

通过以上代码可获取每分钟的行情数据并储存到本地 (备注: 每次只可返回当前时间节点之前的1023行数据)

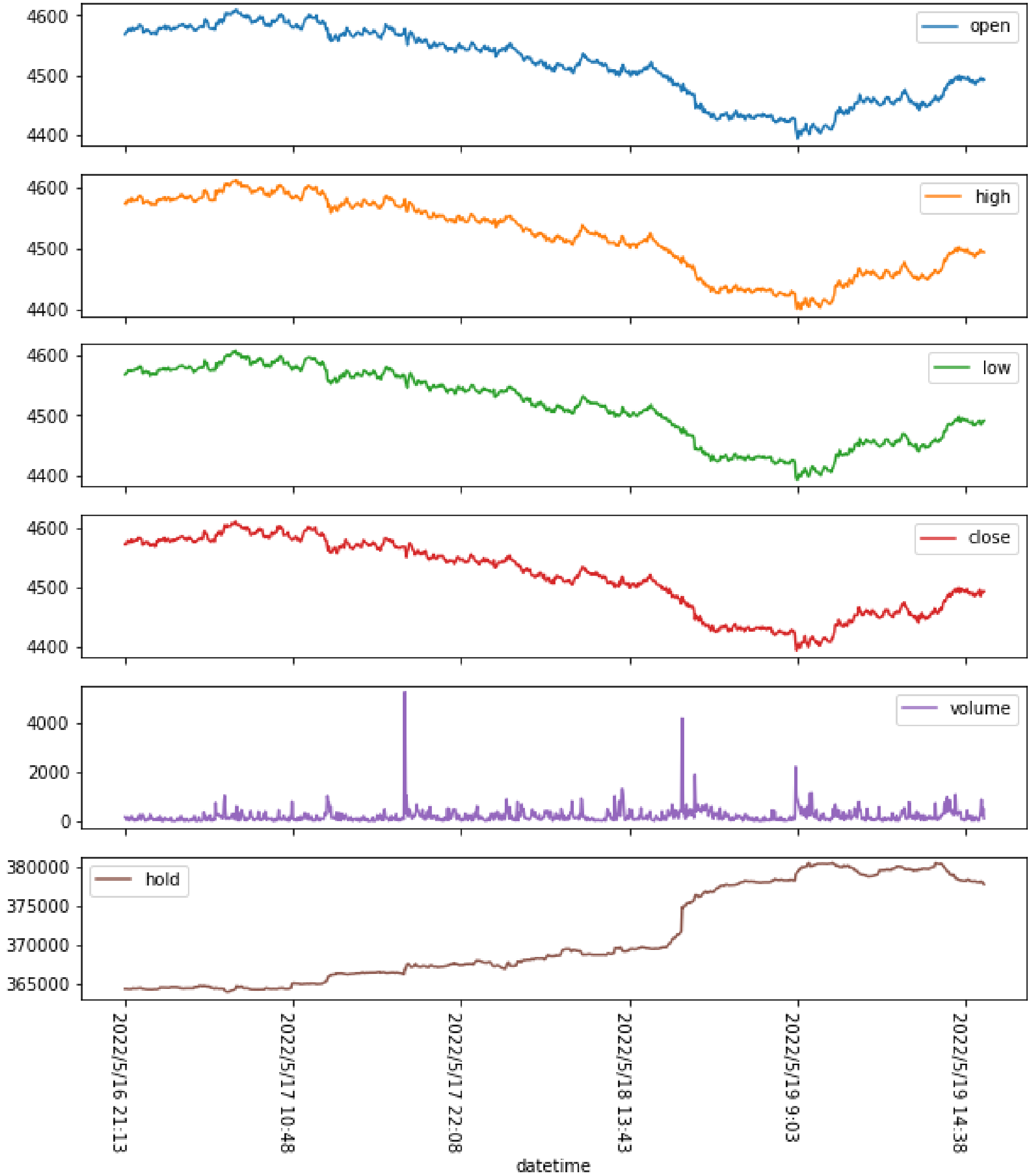
In [4]:

```
import os
os.chdir("C:/Users/c4780/Desktop/desktop")
os.getcwd()

# 读取之前保存到本地的数据
RB_df = pd.read_csv('data_RB2301.csv')

# 将datetime这一列作为index
RB_df.set_index(["datetime"], inplace=True)

# 将所有列的线形图绘制在一张图上可以有效观察数据变化的特点。
RB_df.plot(subplots=True,figsize=(10, 12),rot=270)
```

Out[4]:

```
array([<AxesSubplot:xlabel='datetime'>, <AxesSubplot:xlabel='datetime'>,
       <AxesSubplot:xlabel='datetime'>, <AxesSubplot:xlabel='datetime'>,
```

```
       <AxesSubplot:xlabel='datetime'>, <AxesSubplot:xlabel='datetime'>],
      dtype=object)
```



In [5]:

```
# 持仓量(hold)数据变化区间与其它类别的数据变化区间差别较大 => 故不考虑将hold作为机器学习模型的输入变量
RB_df = RB_df.drop(columns=['hold'])
```

```
print(RB_df)
print("\n")
RB_df.info()
```

```
                 open  high   low  close  volume
datetime
2022/5/16 21:13  4568  4573  4568   4572     185
2022/5/16 21:14  4571  4573  4570   4572     179
2022/5/16 21:15  4572  4572  4570   4572      72
2022/5/16 21:16  4572  4578  4572   4577     129
2022/5/16 21:17  4577  4578  4576   4578     186
...               ...   ...   ...    ...     ...
2022/5/19 14:56  4495  4498  4485   4485     455
2022/5/19 14:57  4494  4494  4485   4490     882
2022/5/19 14:58  4491  4495  4490   4493     256
2022/5/19 14:59  4494  4495  4490   4492     534
2022/5/19 15:00  4492  4494  4491   4493     123

[1023 rows x 5 columns]


<class 'pandas.core.frame.DataFrame'>
Index: 1023 entries, 2022/5/16 21:13 to 2022/5/19 15:00
Data columns (total 5 columns):
 #   Column  Non-Null Count  Dtype
---  ------  --------------  -----
 0   open    1023 non-null   int64
 1   high    1023 non-null   int64
 2   low     1023 non-null   int64
 3   close   1023 non-null   int64
 4   volume  1023 non-null   int64
dtypes: int64(5)
memory usage: 48.0+ KB
```

机器学习模型[Support vector machine (SVM)] - 数据处理

In [6]:

```
# 复制数据
df = RB_df.copy()

# 为确保不将未来数据考虑进模型的计算中，应该对数据进行滞后处理
df['open shifted'] = df['open'].shift(1)
df['high shifted']=df['high'].shift(1)
df['low shifted'] = df['low'].shift(1)
df['close shifted'] = df['close'].shift(1)
df['volume shifted'] = df['volume'].shift(1)

# 计算对数收益率
df['Returns'] = np.log(df['open']/df['open'].shift(1))

# 利用计算出的对数收益率来对每一分钟的交易信号进行分类与标记
```

```
# If returns are positive, it will be labelled 1, otherwise it will be labelled 0
# 如果对数收益率大于等于0，这一分钟的交易信号将被标记为 1(买入信号);
# 如果对数收益率小于0，这一分钟的交易信号将被标记为 -1(卖出信号)
Signal_List = []

for j in df['Returns']:

        if (j>=0):
            Signal_List.append("1") #买入信号

        else :
            Signal_List.append("-1") #卖出信号

df['Signal'] = Signal_List

# 创建模型字典来保存训练集与预测集中的数据
# 去掉数据中的NaN值
# X 将保存模型中的所有特征
# 在 X 中去掉 信号(Signal)、收益率(Returns) 以及 未进行滞后处理的 ohlcv 列
# Y 是需要进行预测的输出量， 即 信号(Signal) 列

Model_Dict = {}

df.dropna(inplace=True)

X = np.array(df.drop(columns=['Signal','Returns','open','high','low','close','volume']))

Y = np.array(df['Signal'])
```

机器学习模型[Support vector machine (SVM)] - 模型测试与筛选

In [7]:

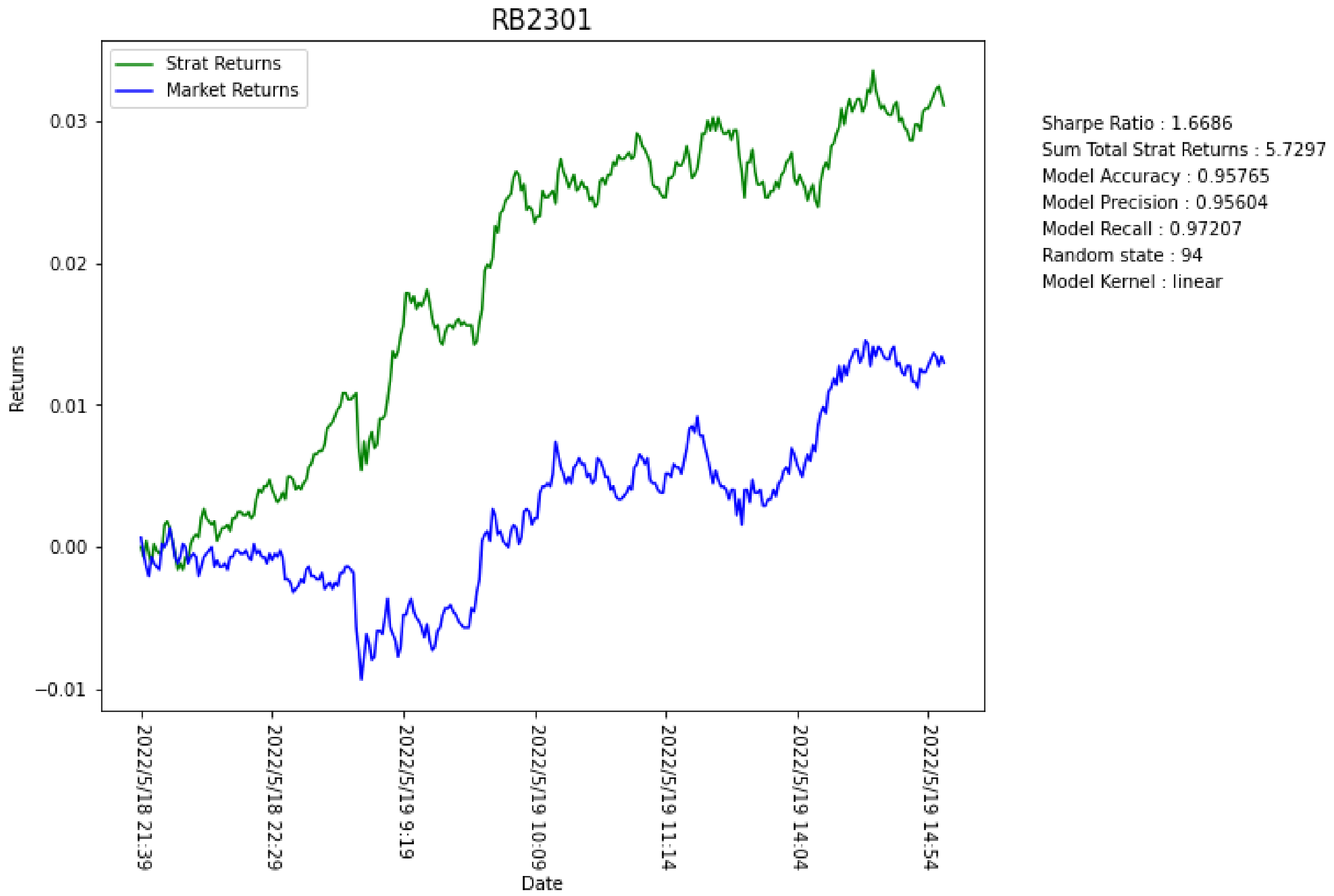
```
import warnings
warnings.filterwarnings('ignore')  # "error", "ignore", "always", "default", "module" or "once"

# 此模型主要拟合训练集中的数据来预测未来的输出变量: 信号(Signal) 列
# 将SVC(support vector machine classifier) 中的常见核函数 :
# 径向基核函数【Radial Basis Function】、线性核【Linear Kernel】、多项式核【Polynomial Kernel】、Sigmoid核
# 全部应用到模型中并找出最优的参数和相对应的模型

for a in range(1,101):

    X_train, X_test, y_train, y_test = train_test_split(X, Y, test_size = 0.3, random_state = a)

    Model_Dict = {}
    Model_Dict['X Train'] = X_train
    Model_Dict['X Test'] = X_test
    Model_Dict['Y Train'] = y_train
```

```python
Model_Dict['Y Test'] = y_test

kernel_names =['rbf','linear','poly','sigmoid']

for b in range(0,4):

    model = svm.SVC(kernel=kernel_names[b])

    # decision_function_shape{'ovo', 'ovr'}, default='ovr' :
    # Note that internally,
    # one-vs-one ('ovo') is always used as a multi-class strategy to train models
    # an ovr matrix is only constructed from the ovo matrix

    model.fit(Model_Dict['X Train'], Model_Dict['Y Train'])
    y_pred = model.predict(Model_Dict['X Test'])

    Model_Dict['Y Prediction'] = y_pred

    # 计算该策略与市场的相对收益
    # 模型的好坏用 Precision指标 来衡量

    prediction_length = len(Model_Dict['Y Prediction'])

    df['SVM Signal'] = 0
    df['SVM Returns'] = 0
    df['Total Strat Returns'] = 0
    df['Market Returns'] = 0

    Signal_Column = df.columns.get_loc('SVM Signal')
    Strat_Column = df.columns.get_loc('SVM Returns')
    Return_Column = df.columns.get_loc('Total Strat Returns')
    Market_Column = df.columns.get_loc('Market Returns')

    df.iloc[-prediction_length:,Signal_Column] = list(map(int,Model_Dict['Y Prediction']))
    df['SVM Returns'] = df['SVM Signal'] * df['Returns'].shift(1)

    df.iloc[-prediction_length:,Return_Column] = np.nancumsum(df['SVM Returns'][-prediction_length:])
    df.iloc[-prediction_length:,Market_Column] = np.nancumsum(df['Returns'][-prediction_length:])

    Model_Dict['Sharpe_Ratio'] = (df['Total Strat Returns'][-1] - df['Market Returns'][-1])/ \
                np.nanstd(df['Total Strat Returns'][-prediction_length:])

    Model_Dict['Accuracy'] = metrics.accuracy_score(Model_Dict['Y Test'], Model_Dict['Y Prediction'])

    Model_Dict['Precision'] = metrics.precision_score(Model_Dict['Y Test'], Model_Dict['Y Prediction'],pos_label=str(1))

    Model_Dict['Recall'] = metrics.recall_score(Model_Dict['Y Test'], Model_Dict['Y Prediction'],pos_label=str(1))
```

```python
# 找出策略总收益大于5.5，夏普比率大于1， 且 Precision指标大于0.95的最优模型
if df['Total Strat Returns'].sum() > 5.5 and Model_Dict['Sharpe_Ratio'] > 1 and Model_Dict['Precision'] > 0.95 :

        # Lastly, we add the accuracy, precision, and recall to our model dictionary

        print("\n")

        print("Random state: " + str(a))

        print("Model Kernel: " + kernel_names[b])

        print("Sharpe Ratio: " + str(Model_Dict['Sharpe_Ratio']))

        print("Accuracy:",metrics.accuracy_score(Model_Dict['Y Test'], Model_Dict['Y Prediction']))

        print("Precision:",metrics.precision_score(Model_Dict['Y Test'], Model_Dict['Y Prediction'],pos_label=str(1)))

        print("Recall:",metrics.recall_score(Model_Dict['Y Test'], Model_Dict['Y Prediction'],pos_label=str(1)))

        print("Sum Total Strat Returns: " +  str(df['Total Strat Returns'].sum()))

        print("##################################################### \n")

        # 最终将选择出的最优模型进行图表可视化

        fig, ax = plt.subplots(figsize=(9, 7))

        ax.plot(df[-prediction_length:].index.values,
                df['Total Strat Returns'][-prediction_length:].values, color='g', label="Strat Returns")

        ax.plot(df[-prediction_length:].index.values,
                df['Market Returns'][-prediction_length:].values, color='b', label="Market Returns")

        ax.set(xlabel= "Date",ylabel="Returns")
        plt.title("RB2301",fontsize=15)
        ax.xaxis.set_major_locator(ticker.AutoLocator())

        plt.figtext(.95,0.78, s="Sharpe Ratio : "+'{0:.5g}'.format(Model_Dict['Sharpe_Ratio']))
        plt.figtext(.95,0.75, s="Sum Total Strat Returns : "+'{0:.5g}'.format(df['Total Strat Returns'].sum()))
        plt.figtext(.95,0.72,s="Model Accuracy : " +'{0:.5g}'.format(Model_Dict['Accuracy']))
        plt.figtext(.95,0.69,s="Model Precision : " +'{0:.5g}'.format(Model_Dict['Precision']))
        plt.figtext(.95,0.66,s="Model Recall : " +'{0:.5g}'.format(Model_Dict['Recall']))
        plt.figtext(.95,0.63,s="Random state : " +'{0:.5g}'.format(a))
        plt.figtext(.95,0.60,s="Model Kernel : " + kernel_names[b])
        plt.xticks(rotation=270)
        plt.legend(loc='best')
        plt.show()
```

```
                best_model =  kernel_names[b]
                best_random_state = a

                break
```

```
Random state: 94
Model Kernel: linear
Sharpe Ratio: 1.6685939901657778
Accuracy: 0.9576547231270358
Precision: 0.9560439560439561
Recall: 0.9720670391061452
Sum Total Strat Returns: 5.729735345292159
#####################################################
```



单从上图来看, 这是个不错的模型; 但在考虑了实际交易中的手续费以及滑点后, 真实收益可能完全没有这么好

机器学习模型[Support vector machine (SVM)] - SVC最优模型资金曲线

In [8]:

```
# SVC最优模型实际收益

X_train, X_test, y_train, y_test = train_test_split(X, Y, test_size = 0.3, random_state = best_random_state)

Model_Dict = {}
Model_Dict['X Train'] = X_train
Model_Dict['X Test'] = X_test
Model_Dict['Y Train'] = y_train
Model_Dict['Y Test'] = y_test

model = svm.SVC(kernel = best_model)
model.fit(Model_Dict['X Train'], Model_Dict['Y Train'])
y_pred = model.predict(Model_Dict['X Test'])
Model_Dict['Y Prediction'] = y_pred

prediction_length = len(Model_Dict['Y Prediction'])

df['SVM Signal'] = 0
df['SVM Returns'] = 0
df['Total Strat Returns'] = 0
df['Market Returns'] = 0

Signal_Column = df.columns.get_loc('SVM Signal')
Strat_Column = df.columns.get_loc('SVM Returns')
Return_Column = df.columns.get_loc('Total Strat Returns')
Market_Column = df.columns.get_loc('Market Returns')

df.iloc[-prediction_length:,Signal_Column] = list(map(int,Model_Dict['Y Prediction']))
df['SVM Returns'] = df['SVM Signal'] * df['Returns'].shift(1)

df.iloc[-prediction_length:,Return_Column] = np.nancumsum(df['SVM Returns'][-prediction_length:])
df.iloc[-prediction_length:,Market_Column] = np.nancumsum(df['Returns'][-prediction_length:])

Model_Dict['Sharpe_Ratio'] = (df['Total Strat Returns'][-1] - df['Market Returns'][-1])/ \
                np.nanstd(df['Total Strat Returns'][-prediction_length:])

Model_Dict['Accuracy'] = metrics.accuracy_score(Model_Dict['Y Test'], Model_Dict['Y Prediction'])

Model_Dict['Precision'] = metrics.precision_score(Model_Dict['Y Test'], Model_Dict['Y Prediction'],pos_label=str(1))

Model_Dict['Recall'] = metrics.recall_score(Model_Dict['Y Test'], Model_Dict['Y Prediction'],pos_label=str(1))

print("Random state: " + str(best_random_state))

print("Model Kernel: " +  best_model)
```

```
print("Precision:" + str(Model_Dict['Precision']))

print("Sharpe Ratio: " + str(Model_Dict['Sharpe_Ratio']))

print("\n")

#初始仓位管理
df['pos']=df['SVM Signal'].shift(1).fillna(value = 0)

#初始化变量
futures_comm_info_df = ak.futures_comm_info(symbol="上海期货交易所")
# print(futures_comm_info_df)

RB2301_info = futures_comm_info_df[futures_comm_info_df['合约代码'].str.contains('rb2301')]
print(RB2301_info)
```

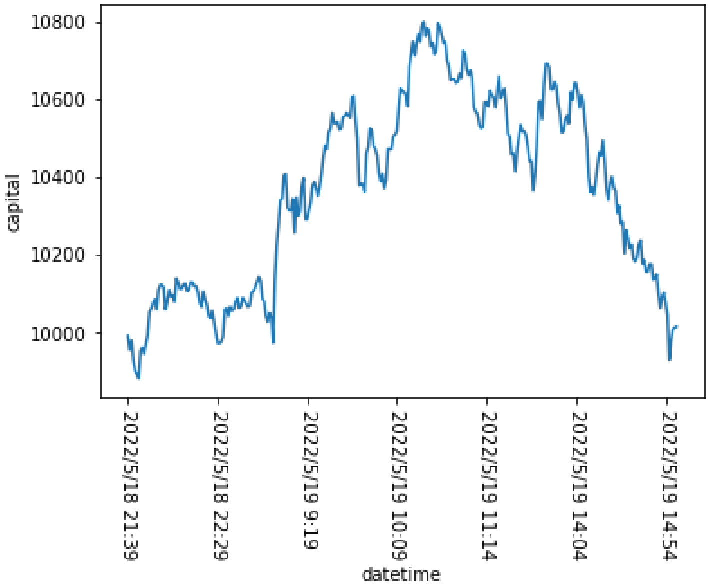
```
Random state: 94
Model Kernel: linear
Precision:0.9560439560439561
Sharpe Ratio: 1.6685939901657778


        交易所名称     合约名称  合约代码     现价    涨停板    跌停板   保证金-买开  \
115  上海期货交易所  螺纹钢2301   rb2301  4485.0  4978.0  3991.0          13

     保证金-卖开  保证金-每手  手续费标准-开仓-万分之  ...  \
115          13       5830.5                  0.0001  ...

    手续费标准-平昨-万分之  手续费标准-平昨-元 手续费标准-平今-万分之  \
115                0.0001     1/万分之(4.5元)                0.0001

     手续费标准-平今-元 每跳毛利  手续费  每跳净利  备注  \
115     1/万分之(4.5元)       10     9.0       1.0   NaN

               手续费更新时间              价格更新时间
115  2022-05-27 21:21:37.090  2022-05-27 21:21:16.051

[1 rows x 21 columns]
```

In [9]:
```
"""
从事期货交易的费用只有手续费
期货手续费是指期货交易者买卖期货成交后按成交合约总价值的一定比例所支付的费用
"""

condition1 = RB2301_info['手续费标准-开仓-万分之'] == RB2301_info['手续费标准-平昨-万分之']
condition2 = RB2301_info['手续费标准-平昨-万分之'] ==  RB2301_info['手续费标准-平今-万分之']

if condition1.bool() == condition2.bool()  :

    print("\n")
```

```python
print(RB2301_info['手续费标准-平今-万分之'])

rate= float(RB2301_info['手续费标准-平今-万分之']) #买卖时的手续费

# 计算每日盈亏和手续费
cash = 10000
size = int(RB2301_info['每跳毛利'])   #每点盈亏
amount = 10 # 10吨/手

slippage = 1.5

# change :  涨跌额(收盘价)
df['change'] = df['close'] - df['close'].shift(1).fillna(value = 0)   # 每分钟涨跌

df['trading_pnl'] = df['change'] * df['pos'] * size   # 盈亏
df['fee'] = 0   # 手续费
df['fee'][df['pos'] != df['pos'].shift(1)] = df['close'] * amount*1  * rate * slippage
df['netpnl'] = df['trading_pnl'] - df['fee']   # 净盈亏

# 汇总求和盈亏，绘制资金曲线
df['cumpnl'] = df['netpnl'].cumsum()
df['capital'] = df['cumpnl'] + cash
df.dropna()
print("\n")
print(df.iloc[-prediction_length:])
df['capital'].iloc[-prediction_length:].plot(x=df.index, kind='line',ylabel='capital')
plt.xticks(rotation=270)
plt.show()
```

```
115    0.0001
Name: 手续费标准-平今-万分之, dtype: float64


                 open  high   low  close  volume  open shifted  high shifted  \
datetime
2022/5/18 21:39  4437  4437  4430   4432     224        4434.0        4439.0
2022/5/18 21:40  4432  4433  4427   4429     220        4437.0        4437.0
2022/5/18 21:41  4428  4428  4424   4426     282        4432.0        4433.0
2022/5/18 21:42  4425  4431  4425   4431      81        4428.0        4428.0
2022/5/18 21:43  4431  4431  4428   4429      52        4425.0        4431.0
...               ...   ...   ...    ...     ...           ...           ...
2022/5/19 14:56  4495  4498  4485   4485     455        4493.0        4497.0
2022/5/19 14:57  4494  4494  4485   4490     882        4495.0        4498.0
2022/5/19 14:58  4491  4495  4490   4493     256        4494.0        4494.0
2022/5/19 14:59  4494  4495  4490   4492     534        4491.0        4495.0
2022/5/19 15:00  4492  4494  4491   4493     123        4494.0        4495.0

                 low shifted  close shifted  volume shifted  ...  SVM Returns  \
```

```
datetime                                                ...
2022/5/18 21:39       4434.0         4437.0           304.0  ...      0.000000
2022/5/18 21:40       4430.0         4432.0           224.0  ...     -0.000676
2022/5/18 21:41       4427.0         4429.0           220.0  ...      0.001128
2022/5/18 21:42       4424.0         4426.0           282.0  ...     -0.000903
2022/5/18 21:43       4425.0         4431.0            81.0  ...     -0.000678
...                      ...            ...             ...  ...           ...
2022/5/19 14:56       4491.0         4496.0           172.0  ...      0.000445
2022/5/19 14:57       4485.0         4485.0           455.0  ...      0.000445
2022/5/19 14:58       4485.0         4490.0           882.0  ...      0.000222
2022/5/19 14:59       4490.0         4493.0           256.0  ...     -0.000668
2022/5/19 15:00       4490.0         4492.0           534.0  ...     -0.000668

                Total Strat Returns  Market Returns  pos  change  trading_pnl  \
datetime
2022/5/18 21:39            0.000000        0.000676  0.0    -5.0         -0.0
2022/5/18 21:40           -0.000676       -0.000451  1.0    -3.0        -30.0
2022/5/18 21:41            0.000451       -0.001354 -1.0    -3.0         30.0
2022/5/18 21:42           -0.000452       -0.002032 -1.0     5.0        -50.0
2022/5/18 21:43           -0.001130       -0.000677  1.0    -2.0        -20.0
...                             ...             ...  ...     ...          ...
2022/5/19 14:56            0.031768        0.013664  1.0   -11.0       -110.0
2022/5/19 14:57            0.032213        0.013441  1.0     5.0         50.0
2022/5/19 14:58            0.032435        0.012773  1.0     3.0         30.0
2022/5/19 14:59            0.031768        0.013441 -1.0    -1.0         10.0
2022/5/19 15:00            0.031100        0.012996  1.0     1.0         10.0

                    fee    netpnl   cumpnl     capital
datetime
2022/5/18 21:39  0.0000   -0.0000  -6.8580   9993.1420
2022/5/18 21:40  6.6435  -36.6435 -43.5015   9956.4985
2022/5/18 21:41  6.6390   23.3610 -20.1405   9979.8595
2022/5/18 21:42  0.0000  -50.0000 -70.1405   9929.8595
2022/5/18 21:43  6.6435  -26.6435 -96.7840   9903.2160
...                 ...       ...      ...         ...
2022/5/19 14:56  6.7275 -116.7275 -69.8055   9930.1945
2022/5/19 14:57  0.0000   50.0000 -19.8055   9980.1945
2022/5/19 14:58  0.0000   30.0000  10.1945  10010.1945
2022/5/19 14:59  6.7380    3.2620  13.4565  10013.4565
2022/5/19 15:00  6.7395    3.2605  16.7170  10016.7170

[307 rows x 23 columns]
```



基于该资金曲线, 我们可以看到 : 在考虑手续费以及滑点为1.5个点位时, 如果初始资金为1万元, 该策略会先赚800块左右,然后又往下跌, 最终在5月19日下午3点时的初始资金仅变为1万零16块

思考: 仅使用每分钟的行情数据作为模型的输入变量来预测对数收益率的正负, 效果不佳; 但如果引入更多的数据, 则应该考虑要对所有数据进行归一化或者标准化处理; 在SVM的核函数选择中, 可以尝试其它不同的核函数 或者 自定义核函数

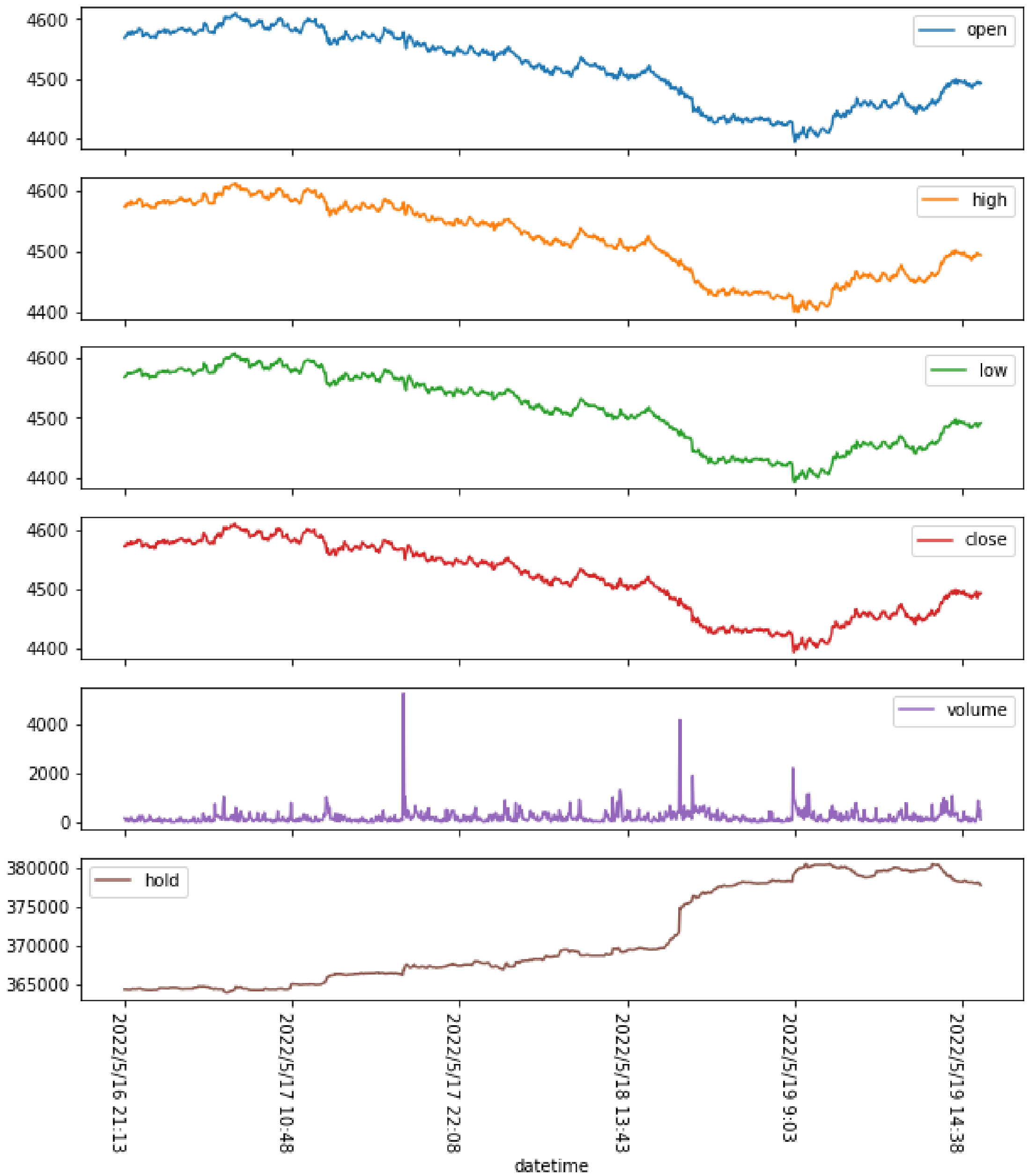
深度学习模型[Long Short Term Memory (LSTM)] - 数据处理

In [10]:

```
# LSTM（Long Short-Term Memory），长短期记忆模型的核心是细胞的状态及其中的门结构

# LSTM的细胞状态由两种激活函数构成（sigmoid和tanh）
# 分别组成遗忘门、输入门和输出门。其中，sigmoid将输入的参数输出为0到1之间的数值
# 它主要用于判定某特征对于模型的影响程度，为1时影响程度最大

import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
import matplotlib.ticker as ticker

from sklearn.model_selection import train_test_split
from sklearn import metrics

# pandas DataFrame打印输出列名对齐 （中文列名）：
pd.set_option('display.unicode.ambiguous_as_wide', True)
pd.set_option('display.unicode.east_asian_width', True)
```

```python
import os
os.chdir("C:/Users/c4780/Desktop/desktop")
os.getcwd()

# 数据读取
RB_df = pd.read_csv('data_RB2301.csv')

# 将datetime这一列作为index
RB_df.set_index(["datetime"], inplace=True)

# 将所有列的线形图绘制在一张图上可以有效观察数据变化的特点。
RB_df.plot(subplots=True,figsize=(10, 12),rot=270)

# 持仓量数据变化区间与其它类别的数据变化区间差别较大 => 故不考虑将hold作为机器学习模型的输入变量
RB_df = RB_df.drop(columns=['hold'])

print("\n")
RB_df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
Index: 1023 entries, 2022/5/16 21:13 to 2022/5/19 15:00
Data columns (total 5 columns):
 #   Column  Non-Null Count  Dtype
---  ------  --------------  -----
 0   open    1023 non-null   int64
 1   high    1023 non-null   int64
 2   low     1023 non-null   int64
 3   close   1023 non-null   int64
 4   volume  1023 non-null   int64
dtypes: int64(5)
memory usage: 48.0+ KB
```



将 open, high, low 以及 volume 的数据 处理成变化率的形式(百分比) ; 利用 close 的数据 计算出对数收益率 :

In [11]:

```
import torch
import torch.nn as nn
```

```python
from torch import optim

import matplotlib.pyplot as plt

# Generating a noisy multi-sin wave

from torch.utils.data import Dataset
from torch.utils.data import DataLoader

dataset = RB_df.copy()

dataset['open_change'] = (dataset['open']-dataset['open'].shift(1))/dataset['open'].shift(1)
dataset['high_change'] = (dataset['high']-dataset['high'].shift(1))/dataset['high'].shift(1)
dataset['low_change'] = (dataset['low']-dataset['low'].shift(1))/dataset['low'].shift(1)
dataset['volume_change'] = (dataset['volume']-dataset['volume'].shift(1))/dataset['volume'].shift(1)

# Log Return
dataset['return'] = np.log(dataset['close']/dataset['close'].shift(1))

dataset = dataset[['open_change','high_change','low_change','volume_change','return']]

dataset.dropna(inplace=True)

#获取对数收益率波动的范围
return_max=dataset["return"].max()
return_min=dataset['return'].min()

print("\n")
print("最高收益率=", return_max)
print("最低收益率=", return_min)
print("波动值=", return_max-return_min)
```

```
最高收益率= 0.0027081939368795564
最低收益率= -0.0038535693159899662
波动值= 0.006561763252869522
```

深度学习模型[Long Short Term Memory (LSTM)] - 模型测试与筛选 & 资金曲线

In [12]:

```python
# 根据序列的长度， 把数据集，构建成序列数据集
# 思路：
# 根据前n分钟的数据，预测下一分钟的对数收益率，
# 前n分钟的所有维度的数据为样本数据，而n+1的对数收益率为标签数据
# sequence的长度，表明了“块”相关数据的长度，即“块长”

# sequence length :  序列长度
# input_size      :  数据的维度
```

```python
total_len = dataset.shape[0]
print("\n")
print("dataset shape =", dataset.shape)
print("dataset len  =", total_len)
print("")
# print("按照序列的长度，重新结构化数据集")
```

```
dataset shape = (1022, 5)
dataset len  = 1022
```

In [13]:

```python
for sequence in range(1,31):
    X=[]
    Y=[]

    print("\n")
    print("sequence : " + str(sequence))

    # 一个连续sequence长度的数据为一个序列（输入序列），一个序列对应一个样本标签（预测值）
    for i in range(dataset.shape[0] - sequence):
        X.append(np.array(dataset.iloc[i:(i+sequence),].values,dtype=np.float32))
        Y.append(np.array(dataset.iloc[(i+sequence),4],dtype=np.float32))

    # print("\n")
    # print("train data  of item  0: \n", X[0])
    # print("train label of item  0: \n", Y[0])

    # 序列化后，样本数据的总长少了sequence length
    # print("\n序列化后的数据形状：")
    X = np.array(X)
    Y = np.array(Y)
    Y = np.expand_dims(Y, 1)
    print("X.shape =",X.shape)
    print("Y.shape =",Y.shape)

    # 划分训练集，验证集

    # 通过切片的方式把数据集且分为训练集+验证集
    # X[start: end; step]
    # 数据集最前面的70%的数据作为训练集
    train_x = X[:int(0.7 * total_len)]
    train_y = Y[:int(0.7*total_len)]

    # 数据集前70%后的数据（30%）作为验证集
    valid_x = X[int(0.7*total_len):]
    valid_y = Y[int(0.7*total_len):]

    print("\n")
```

```python
# print(train_x.shape)
# print(train_y.shape)
# print(valid_x.shape)
# print(valid_y.shape)

# 构造数据迭代器dataloader
class Mydataset(Dataset):

    def __init__(self,x, y, transform = None):
        self.x = x
        self.y = y

    def __getitem__(self, index):
        x1 = self.x[index]
        y1=  self.y[index]
        return x1,y1

    def __len__(self):
        return len(self.x)

# 构建适合dataload的数据集
dataset_train = Mydataset(train_x, train_y)
dataset_valid = Mydataset(valid_x, valid_y)

# 启动dataloader
batch_size = 18
from torch.utils.data.sampler import SequentialSampler
# 关闭shuffle，这样确保数据的时间顺序与走势 和实际情况一致
train_loader = DataLoader(dataset = dataset_train, batch_size = batch_size, shuffle=False,\
                          sampler=SequentialSampler(dataset_train))
test_loader  = DataLoader(dataset = dataset_valid, batch_size = batch_size, shuffle=False,\
                          sampler=SequentialSampler(dataset_valid))
# print(train_loader)
# print(test_loader)

# 构建LSTM网络

#闭环模型
class LSTM(nn.Module):
    # input_size:  输入层样本特征向量的长度
    # hidden_size：隐藏层输出特征向量的长度
    # num_layers：隐藏层的层数
    # output_size：整个网络的输出特征的长度
    def __init__(self, input_size = 5, hidden_size = 24, num_layers = 1, output_size = 1, batch_first=True, batch_size=batch_size, is_close_loo
        super(LSTM, self).__init__()
        # lstm的输入 #batch,seq_len, input_size
        self.input_size  = input_size
        self.hidden_size = hidden_size
```

```
        self.output_size = output_size
        self.batch_first = batch_first
        self.is_close_loop = is_close_loop
        self.hidden0 = torch.zeros(num_layers, batch_size, hidden_size)
        self.cell0   = torch.zeros(num_layers, batch_size, hidden_size)

        # 定义LSTM网络
        # input_size:  输入层样本特征向量的长度
        # hidden_size：隐藏层输出特征向量的长度
        # num_layers：隐藏层的层数
        # batch_first=true:  数据格式为{batch，sequence，input_size}
        self.lstm = nn.LSTM(input_size = self.input_size, hidden_size = self.hidden_size, batch_first = batch_first)

        #定义网络的输出层：
        # hidden_size：输出层的输入，隐藏层的特征输出
        # output_size：输出层的输出
        self.linear = nn.Linear(in_features = self.hidden_size, out_features = self.output_size,  bias=True)

    # 定义前向运算，把各层串联起来
    def forward(self, x):
        #输入层直接送到lstm网络中
        # 输入层数据格式：x.shape = [batch, seq_len, hidden_size]
        # 隐藏层输出数据格式：hn.shape = [num_layes * direction_numbers, batch, hidden_size]
        # 隐藏层输出数据格式：cn.shape = [num_layes * direction_numbers, batch, hidden_size]
        out,(hidden, cell) = self.lstm(x,(self.hidden0, self.cell0))

        # 闭环
        if(self.is_close_loop == True):
            self.hidden0 = hidden
            self.cell0   = cell

        #隐藏层的形状
        a,b,c = hidden.shape

        # 隐藏层的输出，就是全连接层的输入
        # 把隐藏层的输出hidden，向量化后：hidden.reshape(a*b,c)，送到输出层
        out = self.linear(hidden.reshape(a*b,c))

        #返回输出特征
        return out, (hidden,cell)

# 实例化LSTM网络
input_size = 5
hidden_size = 24
n_layers = 1
output_size = 1

lstm_model  = LSTM(input_size = input_size,
```

```
                   hidden_size = hidden_size,
                   num_layers = 1,
                   output_size = 1,
                   batch_first=True,
                   batch_size = batch_size,
                   is_close_loop = False)

# print("\n")
# print(lstm_model)

# 定义loss
criterion = nn.MSELoss()

#定义优化器
Learning_rate = 0.001
optimizer = optim.Adam(lstm_model.parameters(), lr = Learning_rate) # 使用 Adam 优化器

# 训练LSTM网络

# 训练前的准备
n_epochs = 110

lstm_losses = []

# 开始训练
for epoch in range(n_epochs):
    for iter_, (x, label) in enumerate(train_loader):
        if(x.shape[0] != batch_size):
            continue

        pred,(h1,c1) = lstm_model(x)

        #梯度复位
        optimizer.zero_grad()

        #定义损失函数
        loss=criterion(pred, label)

        # 反向求导
        loss.backward(retain_graph=True)

        #梯度迭代
        optimizer.step()

        #记录loss
        lstm_losses.append(loss.item())

# 测试训练效果
```

```python
# 使用验证集进行预测
data_loader = test_loader

# 存放测试序列的预测结果
predicts=[]

# 存放测试序列的实际发生的结果
labels=[]

for idx, (x, label) in enumerate(data_loader):
    if(x.shape[0] != batch_size):
            continue
    #对测试集样本进行批量预测，把结果保存到predict Tensor中
    #开环预测：即每一次序列预测与前后的序列无关。
    predict,(h,c) = lstm_model(x)

    # 把保存在tensor中的批量预测结果转换成list
    predicts.extend(predict.data.squeeze(1).tolist())

    # 把保存在tensor中的批量标签转换成list
    labels.extend(label.data.squeeze(1).tolist())

predicts = np.array(predicts)
labels = np.array(labels)
# print(predicts.shape)
# print(labels.shape)

# 截取日期与时间:
datetime = dataset[(len(dataset)-len(predicts)):]
datetime = datetime.index

# 创建新表
prediction = pd.DataFrame(predicts)
prediction['datetime'] = datetime

# 将datetime这一列作为index
prediction.set_index(["datetime"], inplace=True)

# 修改列名
prediction = prediction.rename(columns={0:'return'})

new_df = pd.merge(prediction,RB_df['close'],\
how='inner',
left_on=['datetime'],
right_on=['datetime'])

# LSTM 模型实际收益
```

```python
Signal_List = []

for r in new_df['return']:

        if (r>=0):
            Signal_List.append("1") #买入信号

        else :
            Signal_List.append("-1") #卖出信号

new_df['signal'] = Signal_List

#初始仓位管理
new_df['pos']=new_df['signal'].shift(1).fillna(value = 0)
new_df['pos']=new_df['pos'].astype(float)

#初始化变量
import akshare as ak
futures_comm_info_df = ak.futures_comm_info(symbol="上海期货交易所")
# print(futures_comm_info_df)

# print("\n")

RB2301_info = futures_comm_info_df[futures_comm_info_df['合约代码'].str.contains('rb2301')]
# print(RB2301_info)

"""
从事期货交易的费用只有手续费
期货手续费是指期货交易者买卖期货成交后按成交合约总价值的一定比例所支付的费用
"""

condition1 = RB2301_info['手续费标准-开仓-万分之'] == RB2301_info['手续费标准-平昨-万分之']
condition2 = RB2301_info['手续费标准-平昨-万分之'] ==  RB2301_info['手续费标准-平今-万分之']

if condition1.bool() == condition2.bool()  :

    # print("\n")

    # print(RB2301_info['手续费标准-平今-万分之'])

    rate= float(RB2301_info['手续费标准-平今-万分之']) #买卖时的手续费

    # 计算每日盈亏和手续费
    cash = 10000
    size = int(RB2301_info['每跳毛利'])   #每点盈亏
    amount = 10 # 10吨/手
```

```python
slippage = 1.5

# change :  涨跌额(收盘价)
new_df['change'] = new_df['close'] - new_df['close'].shift(1).fillna(value = 0)   # 每分钟涨跌

new_df['trading_pnl'] = new_df['change'] * new_df['pos'] * size   # 盈亏
new_df['fee'] = 0   # 手续费

new_df['fee'] = new_df['fee'].loc[new_df['pos'] != new_df['pos'].shift(1).fillna(value=0)]

new_df['fee'] = new_df['fee'].replace(0,99)

new_df['fee'] = new_df['fee'].fillna(value=0)

new_df['fee'] = new_df['fee'].replace(99,np.nan)

new_df['fee'] = new_df['fee'].fillna(value = new_df['close'] * amount * 1 * rate * slippage)


new_df['netpnl'] = new_df['trading_pnl'] - new_df['fee']   # 净盈亏

# 汇总求和盈亏，绘制资金曲线
new_df['cumpnl'] = new_df['netpnl'].cumsum()
new_df['capital'] = new_df['cumpnl'] + cash
new_df.dropna()

if new_df.loc['2022/5/19 15:00','capital'] >= 10700 :

    print("\n")
    print(new_df)

    # 显示预测结果与实际对数收益率的关系
    fig, ax = plt.subplots(figsize=(10, 8))

    ax.plot(datetime,
            predicts, color='r', label="pred")

    ax.plot(datetime,
            labels, color='b', label="real")

    ax.set(xlabel= "Date",ylabel="Return")
    plt.title("RB2301",fontsize=15)
    ax.xaxis.set_major_locator(ticker.AutoLocator())
    plt.xticks(rotation=270)
    plt.legend(loc='best')
    plt.figtext(.95,0.75, s="sequence : " + str(sequence))
    plt.savefig('pred_real.png',bbox_inches = 'tight', dpi=150)
```

```python
            plt.show()
            print("\n")

            # 绘制资金曲线
            new_df['capital'].plot(x=new_df.index, kind='line',ylabel='capital')
            plt.xticks(rotation=270)
            plt.figtext(.95,0.75, s="sequence : " + str(sequence))
            plt.savefig('capital.png',bbox_inches = 'tight', dpi=200)

            plt.show()
            print("\n")

            break
```

```
sequence : 1
X.shape = (1021, 1, 5)
Y.shape = (1021, 1)




sequence : 2
X.shape = (1020, 2, 5)
Y.shape = (1020, 1)




sequence : 3
X.shape = (1019, 3, 5)
Y.shape = (1019, 1)




sequence : 4
X.shape = (1018, 4, 5)
Y.shape = (1018, 1)




sequence : 5
X.shape = (1017, 5, 5)
Y.shape = (1017, 1)




sequence : 6
X.shape = (1016, 6, 5)
```

```
Y.shape = (1016, 1)



sequence : 7
X.shape = (1015, 7, 5)
Y.shape = (1015, 1)



sequence : 8
X.shape = (1014, 8, 5)
Y.shape = (1014, 1)



sequence : 9
X.shape = (1013, 9, 5)
Y.shape = (1013, 1)



sequence : 10
X.shape = (1012, 10, 5)
Y.shape = (1012, 1)



sequence : 11
X.shape = (1011, 11, 5)
Y.shape = (1011, 1)



sequence : 12
X.shape = (1010, 12, 5)
Y.shape = (1010, 1)



sequence : 13
X.shape = (1009, 13, 5)
Y.shape = (1009, 1)
```

```
sequence : 14
X.shape = (1008, 14, 5)
Y.shape = (1008, 1)
```

```
sequence : 15
X.shape = (1007, 15, 5)
Y.shape = (1007, 1)
```

```
sequence : 16
X.shape = (1006, 16, 5)
Y.shape = (1006, 1)
```

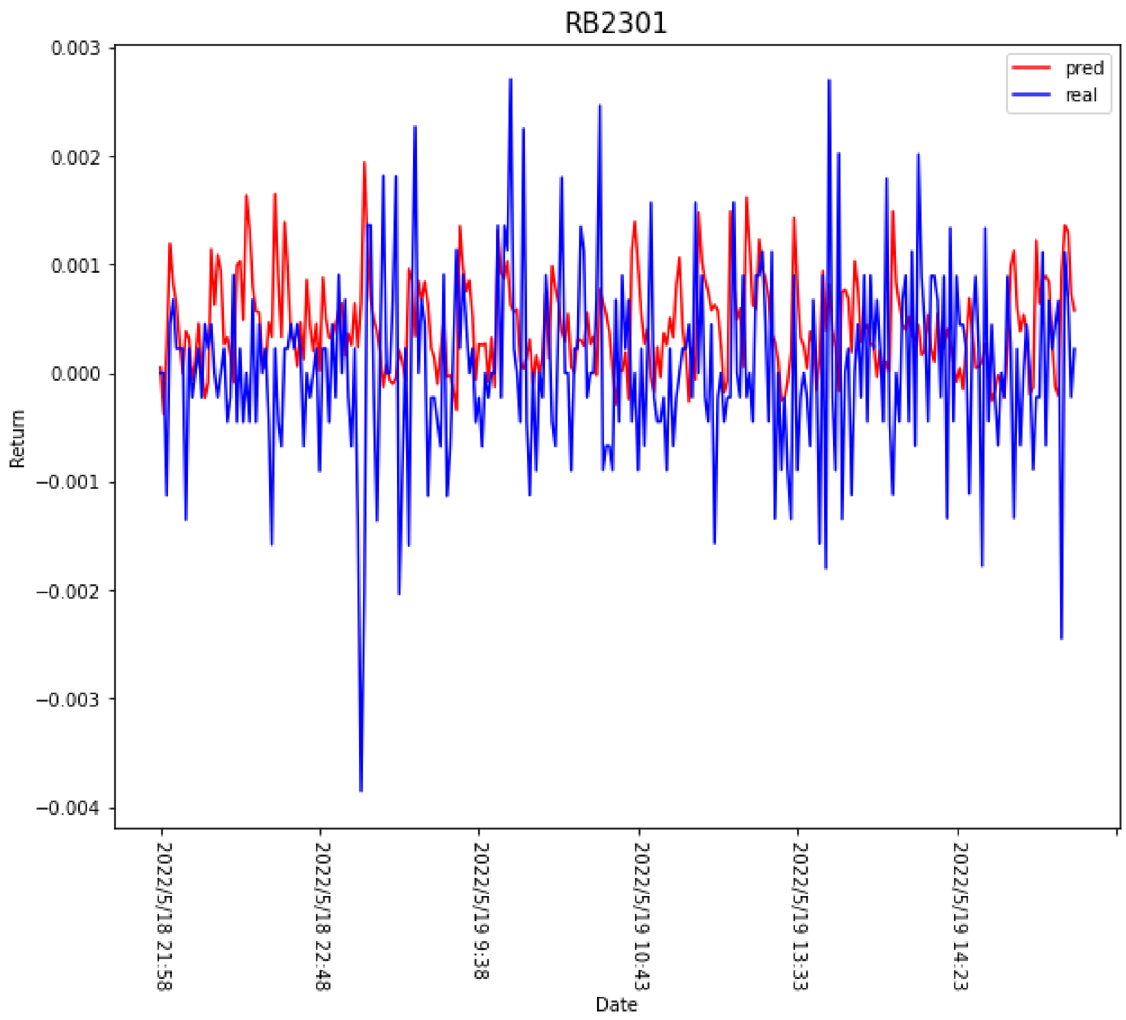
```
sequence : 17
X.shape = (1005, 17, 5)
Y.shape = (1005, 1)
```

```
sequence : 18
X.shape = (1004, 18, 5)
Y.shape = (1004, 1)
```

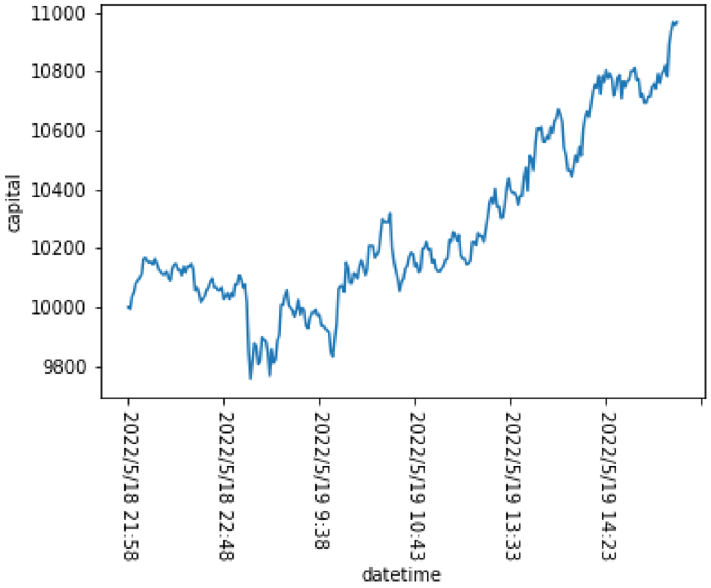
```
sequence : 19
X.shape = (1003, 19, 5)
Y.shape = (1003, 1)
```

```
                   return  close signal  pos  change  trading_pnl     fee  \
datetime
2022/5/18 21:58  0.000051   4431      1  0.0  4431.0          0.0  0.0000
2022/5/18 21:59 -0.000378   4431     -1  1.0     0.0          0.0  6.6465
2022/5/18 22:00  0.000254   4426      1 -1.0    -5.0         50.0  6.6390
2022/5/18 22:01  0.001193   4428      1  1.0     2.0         20.0  6.6420
2022/5/18 22:02  0.000820   4431      1  1.0     3.0         30.0  0.0000
...                   ...    ...    ...  ...     ...          ...     ...
2022/5/19 14:56  0.000931   4485      1 -1.0   -11.0        110.0  0.0000
2022/5/19 14:57  0.001362   4490      1  1.0     5.0         50.0  6.7350
```

```
2022/5/19 14:58  0.001302   4493      1  1.0     3.0         30.0  0.0000
2022/5/19 14:59  0.000729   4492      1  1.0    -1.0        -10.0  0.0000
2022/5/19 15:00  0.000576   4493      1  1.0     1.0         10.0  0.0000

                   netpnl    cumpnl     capital
datetime
2022/5/18 21:58    0.0000    0.0000  10000.0000
2022/5/18 21:59   -6.6465   -6.6465   9993.3535
2022/5/18 22:00   43.3610   36.7145  10036.7145
2022/5/18 22:01   13.3580   50.0725  10050.0725
2022/5/18 22:02   30.0000   80.0725  10080.0725
...                   ...       ...         ...
2022/5/19 14:56  110.0000  893.3010  10893.3010
2022/5/19 14:57   43.2650  936.5660  10936.5660
2022/5/19 14:58   30.0000  966.5660  10966.5660
2022/5/19 14:59  -10.0000  956.5660  10956.5660
2022/5/19 15:00   10.0000  966.5660  10966.5660

[288 rows x 10 columns]
```


RB2301
pred
real
0.003
0.002
0.001
0.000
−0.001
−0.002
−0.003
−0.004
Return
2022/5/18 21:58
2022/5/18 22:48
2022/5/19 9:38
2022/5/19 10:43
2022/5/19 13:33
2022/5/19 14:23
Date

sequence : 19



sequence : 19

基于该资金曲线, 我们可以看到 : 在考虑手续费以及滑点为1.5个点位时, 如果初始资金为1万元, 该策略会先在1万块左右上下浮动,然后不断往上升, 最终在5月19日下午3点时的初始资金变为1万1000块左右

总结 : 深度学习模型(LSTM) 优于 机器学习模型(SVM)