基于深度学习的股市预测

Stock Prediction Based on GAN Neural Network

2019.11.30

一、程序运行环境

- ▶Anaconda3， python3.7，安装了各种所需的包。
- ▶其中Tensorflow 的包是1.15.0，其他包都直接在命令端pip install xxx（xxx为包的名字），即系统预设下载的版本。
- ▶sklearn建议先用conda install sklearn 再使用pip install sklearn。
- ▶在安装sklearn一些子包时可以直接用pip install sklearn.xxx（xxx为子包的名字，不必用conda pip sklearn.xxx）

二、从雅虎上读取GS 的2010年到2018年的全部数据

代码

```
import os
import numpy as np
import pandas as pd
import pandas_datareader.data as web
import datetime as dt
nyyh=web.DataReader('GS','yahoo',dt.datetime(2010,1,1),dt.datetime(2018,12,31))
nyyh.tail()
print(type(nyyh))

data=pd.read_csv('dataGS.csv')
print(data)
```

打印的采集数据

```
In [9]: runfile('C:/Users/tcx/stockprediction2.py', wdir='C:/Users/tcx')
<class 'pandas.core.frame.DataFrame'>
            Date        High         Low  ...       Close      Volume   Adj Close
0     2009-12-31  170.130005  166.929993  ...  168.839996   6401800.0  147.799942
1     2010-01-04  174.250000  169.509995  ...  173.080002   9135000.0  151.511627
2     2010-01-05  176.259995  172.570007  ...  176.139999  11659400.0  154.190262
3     2010-01-06  175.380005  173.759995  ...  174.259995   7381100.0  152.544586
4     2010-01-07  178.750000  173.949997  ...  177.669998   8727400.0  155.529587
...          ...         ...         ...  ...         ...         ...         ...
2260  2018-12-24  160.000000  154.309998  ...  156.350006   3783500.0  154.055573
2261  2018-12-26  163.110001  151.699997  ...  162.929993   7054700.0  160.539001
2262  2018-12-27  165.410004  159.020004  ...  165.410004   4973000.0  162.982620
2263  2018-12-28  165.949997  162.020004  ...  163.029999   4110500.0  160.637543
2264  2018-12-31  167.119995  163.779999  ...  167.050003   4550000.0  164.598572

[2265 rows x 7 columns]
There are 2265 number of days in the dataset.
```

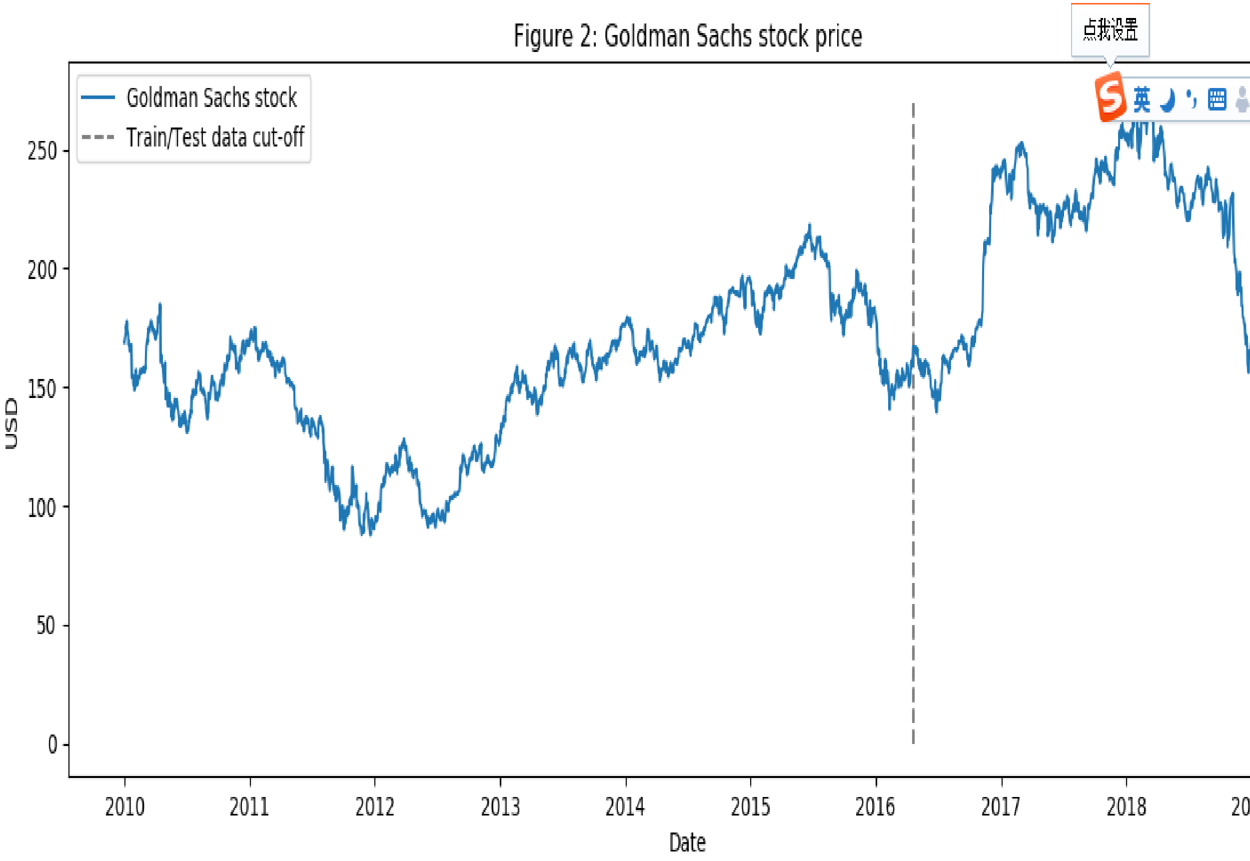
三、做出股市行情图，和预测分割线

代码

```
###zuochushujufengexian
import warnings
import mxnet as mx
import matplotlib.pyplot as plt
warnings.filterwarnings("ignore")
context = mx.cpu(); model_ctx=mx.cpu()
mx.random.seed(1719)
def parser(x):
 return dt.datetime.strptime(x,'%Y-%m-%d')
dataset_ex_df = pd.read_csv('dataGS.csv', header=0, parse_dates=[0], date_parser=parser)
dataset_ex_df[['Date', 'Close']].head(3)
print('There are {} number of days in the dataset.'.format(dataset_ex_df.shape[0]))
plt.figure(figsize=(14, 5), dpi=100)
plt.plot(dataset_ex_df['Date'], dataset_ex_df['Close'], label='Goldman Sachs stock')
plt.vlines(dt.date(2016,4, 20), 0, 270, linestyles='--', colors='gray', label='Train/Test data cut-off')
plt.xlabel('Date')
plt.ylabel('USD')
plt.title('Figure 2: Goldman Sachs stock price')
plt.legend()
plt.show()
```

```
num_training_days = int(dataset_ex_df.shape[0]*.7)
print('Number of training days: {}. Number of test days: {}.'.format(num_training_days,
 dataset_ex_df.shape[0]-num_training_days))
```

行情图



四、提炼股票技术指标，并作图

代码

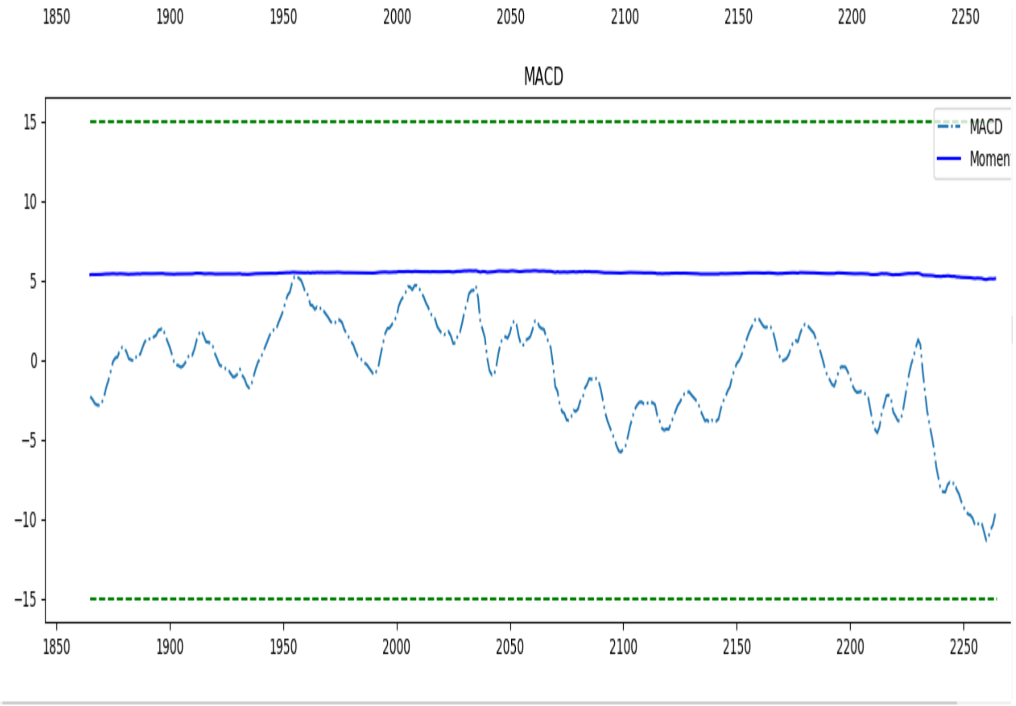
```python
import math
def get_technical_indicators(dataset):
 # Create 7 and 21 days Moving Average
 dataset['ma7'] = dataset['Close'].rolling(window=7).mean()
 dataset['ma21'] = dataset['Close'].rolling(window=21).mean()

# # Create MACD
 dataset['26ema'] = pd.DataFrame.ewm(dataset['Close'], span=26).mean()
 dataset['12ema'] = pd.DataFrame.ewm(dataset['Close'], span=12).mean()
 dataset['MACD'] = (dataset['12ema']-dataset['26ema'])
 # Create Bollinger Bands
 dataset['20sd'] = data['Close'].rolling(20).std()
 dataset['upper_band'] = dataset['ma21'] + (dataset['20sd']*2)
 dataset['lower_band'] = dataset['ma21'] - (dataset['20sd']*2)

 # Create Exponential moving average
 dataset['ema'] = dataset['Close'].ewm(com=0.5).mean()

 # Create Momentum
 dataset['momentum'] = dataset['Close']-1

 #Create log_momentum
 dataset['log_momentum'] = dataset['momentum'].apply(lambda x:math.log(x))

 return dataset
dataset_TI_df = get_technical_indicators(dataset_ex_df[['Close']]) #####jishuzhibiao
dataset_TI_df.head()


print(dataset_TI_df)###
```

代码

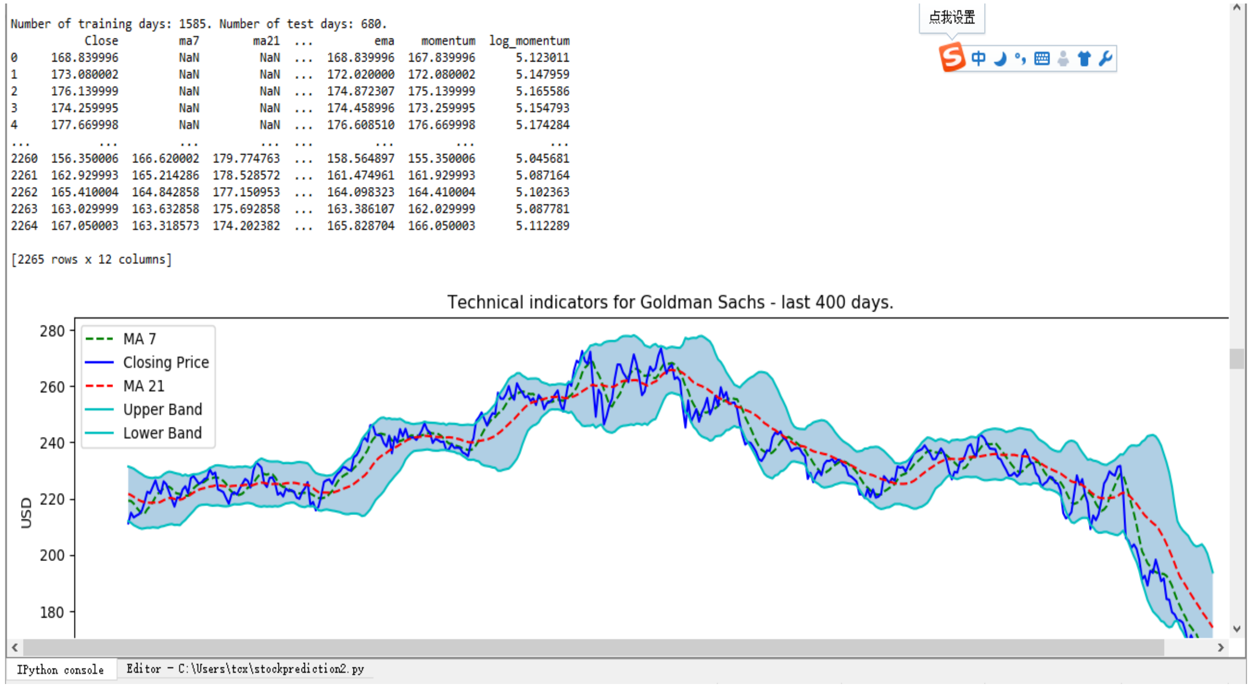
```python
def plot_technical_indicators(dataset, last_days):
 plt.figure(figsize=(16, 10), dpi=100)
 shape_0 = dataset.shape[0]
 xmacd_ = shape_0-last_days

 dataset = dataset.iloc[-last_days:, :]
 x_ = range(3, dataset.shape[0])
 x_ =list(dataset.index)

 # Plot first subplot
 plt.subplot(2, 1, 1)
 plt.plot(dataset['ma7'],label='MA 7', color='g',linestyle='--')
 plt.plot(dataset['Close'],label='Closing Price', color='b')
 plt.plot(dataset['ma21'],label='MA 21', color='r',linestyle='--')
 plt.plot(dataset['upper_band'],label='Upper Band', color='c')
 plt.plot(dataset['lower_band'],label='Lower Band', color='c')
 plt.fill_between(x_, dataset['lower_band'], dataset['upper_band'], alpha=0.35)
 plt.title('Technical indicators for Goldman Sachs - last {} days.'.format(last_days))
 plt.ylabel('USD')
 plt.legend()
 # Plot second subplot
 plt.subplot(2, 1, 2)
 plt.title('MACD')
 plt.plot(dataset['MACD'],label='MACD', linestyle='-.')
 plt.hlines(15, xmacd_, shape_0, colors='g', linestyles='--')
 plt.hlines(-15, xmacd_, shape_0, colors='g', linestyles='--')
 plt.plot(dataset['log_momentum'],label='Momentum', color='b',linestyle='-')
 plt.legend()
 plt.show()
plot_technical_indicators(dataset_TI_df, 400)
```

四、提炼股票技术指标，并作图

效果



效果



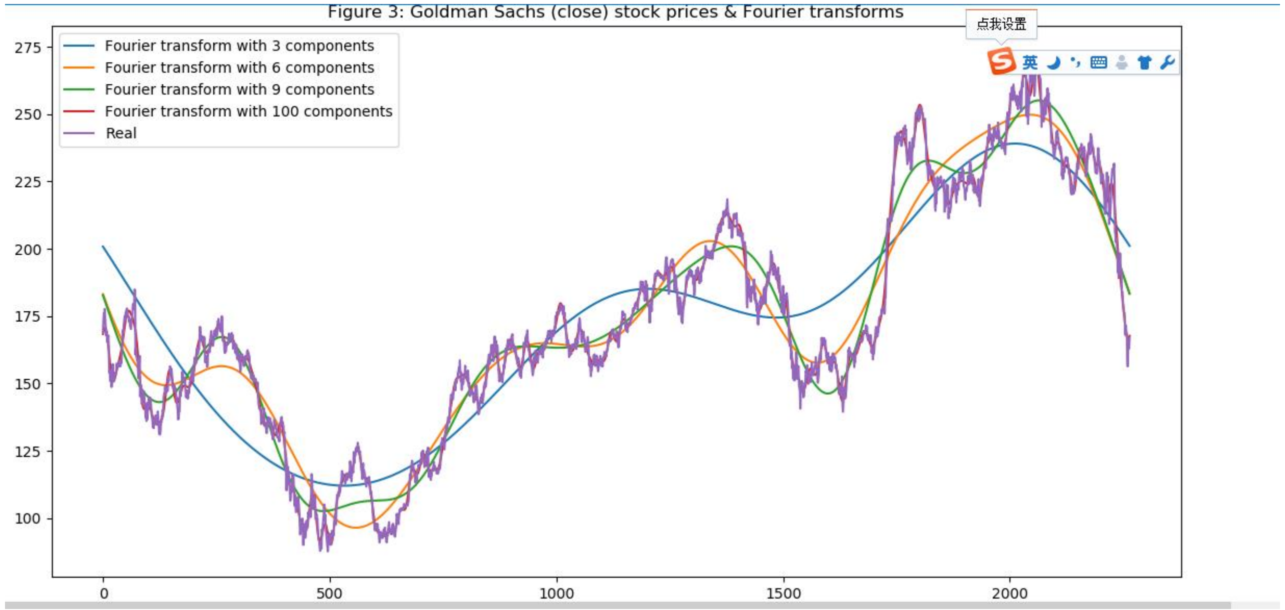
```
Number of training days: 1585. Number of test days: 680.
           Close         ma7        ma21  ...         ema    momentum  log_momentum
0     168.839996         NaN         NaN  ...  168.839996  167.839996      5.123011
1     173.080002         NaN         NaN  ...  172.020000  172.080002      5.147959
2     176.139999         NaN         NaN  ...  174.872307  175.139999      5.165586
3     174.259995         NaN         NaN  ...  174.458996  173.259995      5.154793
4     177.669998         NaN         NaN  ...  176.608510  176.669998      5.174284
...          ...         ...         ...  ...         ...         ...           ...
2260  156.350006  166.620002  179.774763  ...  158.564897  155.350006      5.045681
2261  162.929993  165.214286  178.528572  ...  161.474961  161.929993      5.087164
2262  165.410004  164.842858  177.150953  ...  164.098323  164.410004      5.102363
2263  163.029999  163.632858  175.692858  ...  163.386107  162.029999      5.087781
2264  167.050003  163.318573  174.202382  ...  165.828704  166.050003      5.112289

[2265 rows x 12 columns]
```

五、股票的走势与傅里叶指标

代码

```
data_FT = dataset_ex_df[['Date', 'Close']]
close_fft = np.fft.fft(np.asarray(data_FT['Close'].tolist()))
fft_df = pd.DataFrame({'fft':close_fft})
fft_df['absolute'] = fft_df['fft'].apply(lambda x: np.abs(x))
fft_df['angle'] = fft_df['fft'].apply(lambda x: np.angle(x))
plt.figure(figsize=(14, 7), dpi=100)
fft_list = np.asarray(fft_df['fft'].tolist())
for num_ in [3, 6, 9, 100]:
 fft_list_m10= np.copy(fft_list); fft_list_m10[num_:-num_]=0
 plt.plot(np.fft.ifft(fft_list_m10), label='Fourier transform with {} components'.format(num_))
plt.plot(data_FT['Close'], label='Real')
plt.xlabel('Days')
plt.ylabel('USD')
plt.title('Figure 3: Goldman Sachs (close) stock prices & Fourier transforms')
plt.legend()
plt.show()
```

走势图



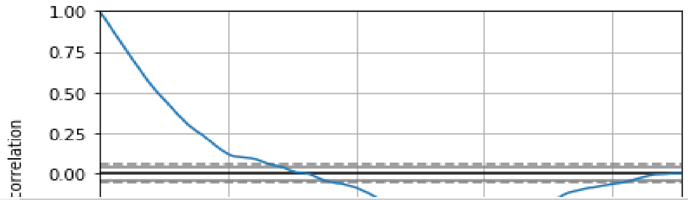
六、股票的ARIMA指标

代码

```python
from statsmodels.tsa.arima_model import ARIMA
from pandas import DataFrame
from pandas import datetime
series = data_FT['Close']
model = ARIMA(series, order=(5, 1, 0))
model_fit = model.fit(disp=0)
print(model_fit.summary())  ####
from pandas.plotting import autocorrelation_plot
autocorrelation_plot(series)
plt.figure(figsize=(10, 7), dpi=80)
plt.show()
from pandas import read_csv
from pandas import datetime
from statsmodels.tsa.arima_model import ARIMA
from sklearn.metrics import mean_squared_error
X = series.values
size = int(len(X) * 0.66)
train, test = X[0:size], X[size:len(X)]
history = [x for x in train]
predictions = list()
for t in range(len(test)):
 model = ARIMA(history, order=(5,1,0))
 model_fit = model.fit(disp=0)
 output = model_fit.forecast()
 yhat = output[0]
 predictions.append(yhat)
 obs = test[t]
 history.append(obs)
error = mean_squared_error(test, predictions)
print('Test MSE: %.3f' % error)
#Plot the predicted (from ARIMA) and real prices
plt.figure(figsize=(12, 6), dpi=100)
plt.plot(test, label='Real')
plt.plot(predictions, color='red', label='Predicted')
plt.xlabel('Days')
plt.ylabel('USD')
plt.title('Figure 5: ARIMA model on GS stock')
plt.legend()
plt.show()
```

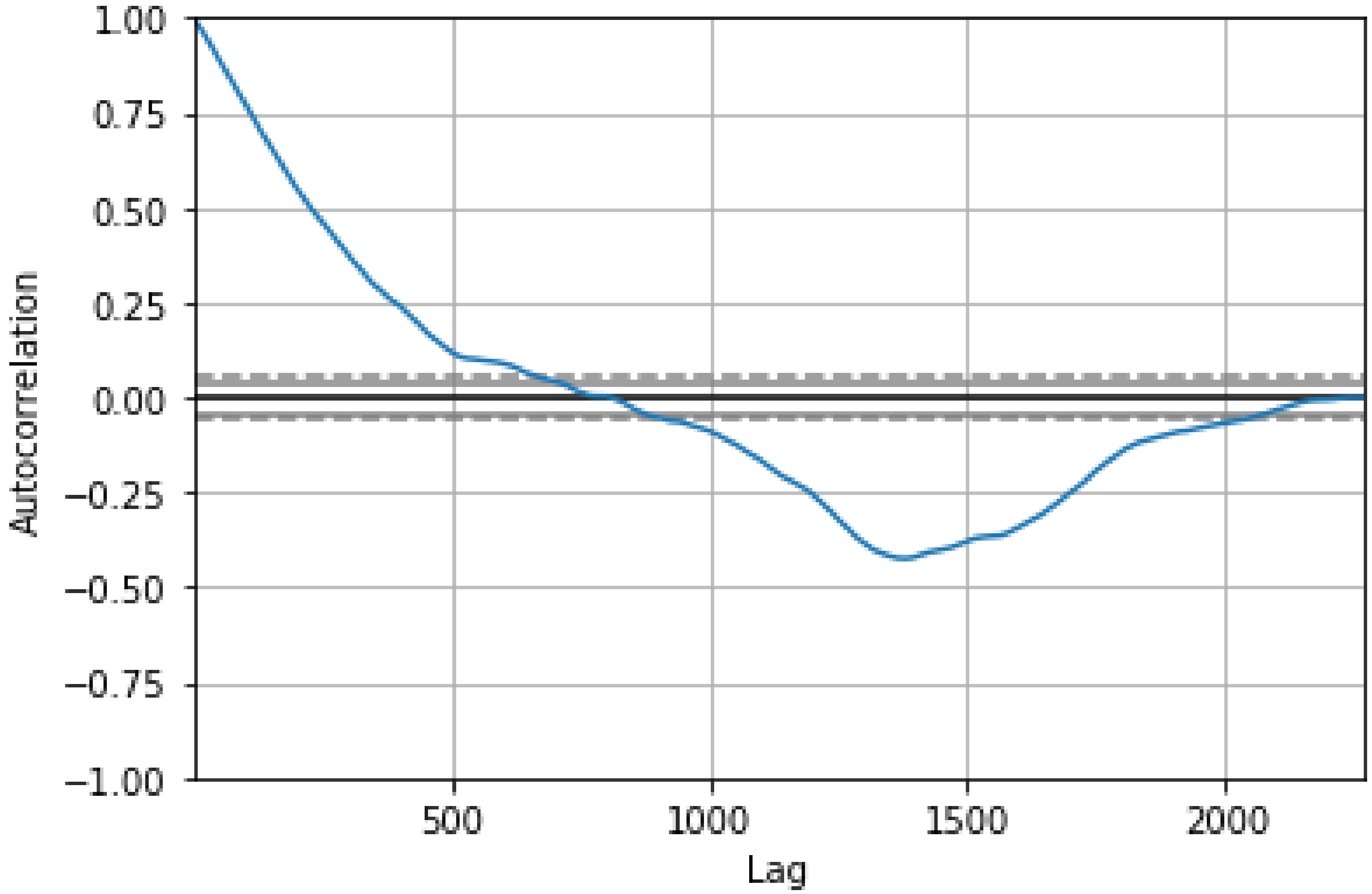
运行结果

```
                             ARIMA Model Results
==============================================================================
Dep. Variable:                D.Close   No. Observations:                 2264
Model:                 ARIMA(5, 1, 0)   Log Likelihood               -5465.888
Method:                       css-mle   S.D. of innovations              2.706
Date:                Sat, 30 Nov 2019   AIC                          10945.777
Time:                        16:54:17   BIC                          10985.851
Sample:                             1   HQIC                         10960.399

=================================================================================
                    coef    std err          z      P>|z|      [0.025      0.975]
---------------------------------------------------------------------------------
const            -0.0011      0.054     -0.020      0.984      -0.106       0.104
ar.L1.D.Close    -0.0205      0.021     -0.974      0.330      -0.062       0.021
ar.L2.D.Close     0.0140      0.021      0.665      0.506      -0.027       0.055
ar.L3.D.Close    -0.0030      0.021     -0.141      0.888      -0.044       0.038
ar.L4.D.Close     0.0026      0.021      0.122      0.903      -0.039       0.044
ar.L5.D.Close    -0.0522      0.021     -2.479      0.013      -0.093      -0.011
                                    Roots
=============================================================================
                  Real          Imaginary           Modulus         Frequency
-----------------------------------------------------------------------------
AR.1           -1.7595           -0.0000j            1.7595           -0.5000
AR.2           -0.5700           -1.7248j            1.8165           -0.3008
AR.3           -0.5700           +1.7248j            1.8165            0.3008
AR.4            1.4743           -1.0616j            1.8168           -0.0993
AR.5            1.4743           +1.0616j            1.8168            0.0993
-----------------------------------------------------------------------------
```



六、股票的ARIMA指标

运行结果

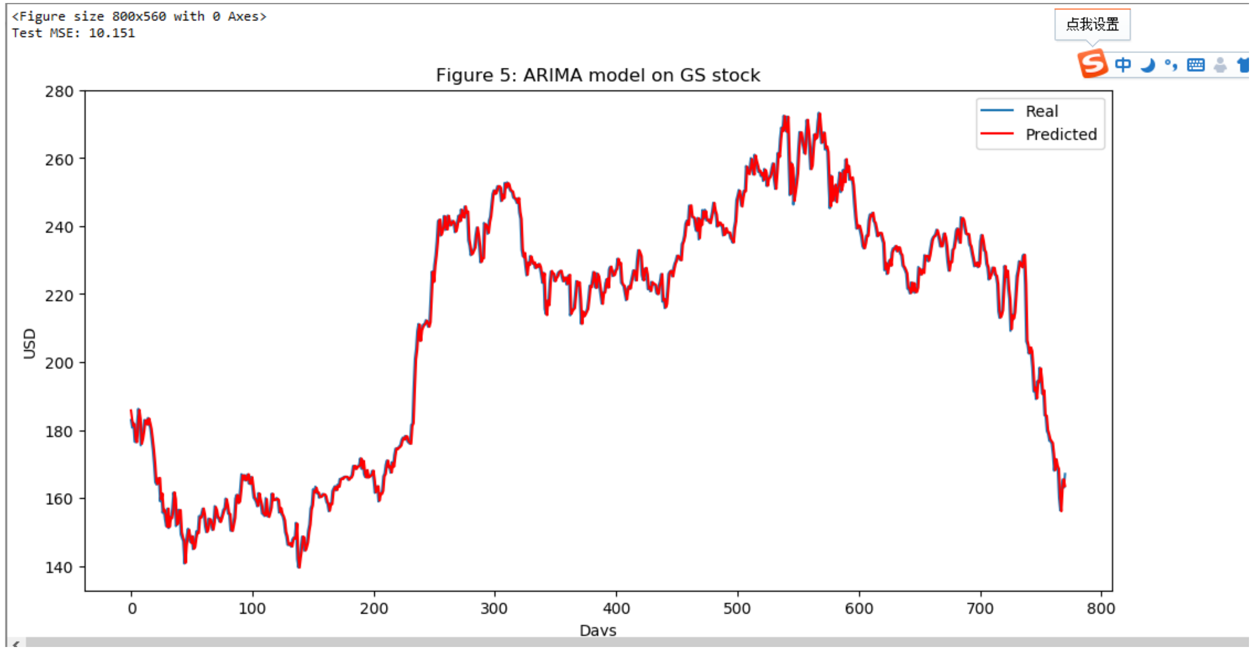


六、股票的ARIMA指标

打印mse方差和提炼的特征数

ARIMA模型的结果图

```
#dataset_total_df.shape
print('Total dataset has {} samples, and {} features.'.format(dataset_TI_df.shape[0],dataset_TI_df.shape[1]))
print(dataset_TI_df)
```



七、以12个技术指标为例，进一步挖掘股票特征

```python
v=['ma7','ma21','26ema','12ema','MACD','20sd','upper_band','lower_band','ema','momentum','log_momentum']
import xgboost as xgb
def get_feature_importance_data(data_income,s):
 data = data_income.copy()
 y = data[s]
 X = data.iloc[:, 1:]

 train_samples = int(X.shape[0] * 0.65)

 X_train = X.iloc[:train_samples]
 X_test = X.iloc[train_samples:]
 y_train = y.iloc[:train_samples]
 y_test = y.iloc[train_samples:]

 return (X_train, y_train),(X_test, y_test)
# Get training and test data
(X_train_FI, y_train_FI), (X_test_FI, y_test_FI) = get_feature_importance_data(dataset_TI_df,'ma7')
regressor = xgb.XGBRegressor(gamma=0.0,n_estimators=150,base_score=0.7,colsample_bytree=1,learning_rate=0.05)
xgbModel = regressor.fit(X_train_FI,y_train_FI,
 eval_set = [(X_train_FI, y_train_FI), (X_test_FI, y_test_FI)],
 verbose=False)
eval_result = regressor.evals_result()
training_rounds = range(len(eval_result['validation_0']['rmse']))
bar_width=0.2
fig = plt.figure(figsize=(8,8))
plt.xticks(rotation='vertical')
plt.bar([i for i in range(len(xgbModel.feature_importances_))], xgbModel.feature_importances_.tolist(), tick_label=X_test_FI.columns)


(X_train_FI, y_train_FI), (X_test_FI, y_test_FI) = get_feature_importance_data(dataset_TI_df,'ma21')
regressor = xgb.XGBRegressor(gamma=0.0,n_estimators=150,base_score=0.7,colsample_bytree=1,learning_rate=0.05)
xgbModel2 = regressor.fit(X_train_FI,y_train_FI,
 eval_set = [(X_train_FI, y_train_FI), (X_test_FI, y_test_FI)],
 verbose=False)
eval_result = regressor.evals_result()
training_rounds = range(len(eval_result['validation_0']['rmse']))
plt.bar([i for i in range(len(xgbModel2.feature_importances_))], xgbModel2.feature_importances_.tolist(), tick_label=X_test_FI.columns)
```

```python
(X_train_FI, y_train_FI), (X_test_FI, y_test_FI) = get_feature_importance_data(dataset_TI_df,'20sd')
regressor = xgb.XGBRegressor(gamma=0.0,n_estimators=150,base_score=0.7,colsample_bytree=1,learning_rate=0.05)
xgbModel3 = regressor.fit(X_train_FI,y_train_FI,
 eval_set = [(X_train_FI, y_train_FI), (X_test_FI, y_test_FI)],
 verbose=False)
eval_result = regressor.evals_result()
training_rounds = range(len(eval_result['validation_0']['rmse']))
plt.bar([i for i in range(len(xgbModel3.feature_importances_))], xgbModel3.feature_importances_.tolist(), tick_label=X_test_FI.columns)


(X_train_FI, y_train_FI), (X_test_FI, y_test_FI) = get_feature_importance_data(dataset_TI_df,'Close')
regressor = xgb.XGBRegressor(gamma=0.0,n_estimators=150,base_score=0.7,colsample_bytree=1,learning_rate=0.05)
xgbModel4 = regressor.fit(X_train_FI,y_train_FI,
 eval_set = [(X_train_FI, y_train_FI), (X_test_FI, y_test_FI)],
 verbose=False)
eval_result = regressor.evals_result()
training_rounds = range(len(eval_result['validation_0']['rmse']))
plt.bar([i for i in range(len(xgbModel4.feature_importances_))], xgbModel4.feature_importances_.tolist(), tick_label=X_test_FI.columns)


plt.scatter(x=training_rounds,y=eval_result['validation_0']['rmse'],label='Training Error')
plt.scatter(x=training_rounds,y=eval_result['validation_1']['rmse'],label='Validation Error')
plt.xlabel('Iterations')
plt.ylabel('RMSE')
plt.title('Training Vs Validation Error')
plt.legend()
plt.show()

(X_train_FI, y_train_FI), (X_test_FI, y_test_FI) = get_feature_importance_data(dataset_TI_df,'MACD')
regressor = xgb.XGBRegressor(gamma=0.0,n_estimators=150,base_score=0.7,colsample_bytree=1,learning_rate=0.05)
xgbModel4 = regressor.fit(X_train_FI,y_train_FI,
 eval_set = [(X_train_FI, y_train_FI), (X_test_FI, y_test_FI)],
 verbose=False)
eval_result = regressor.evals_result()
training_rounds = range(len(eval_result['validation_0']['rmse']))
plt.bar([i for i in range(len(xgbModel4.feature_importances_))], xgbModel4.feature_importances_.tolist(), tick_label=X_test_FI.columns)
```

七、以12个技术指标为例，进一步挖掘股票特征

代码

```python
(X_train_FI, y_train_FI), (X_test_FI, y_test_FI) = get_feature_importance_data(dataset_TI_df,'26ema')
regressor = xgb.XGBRegressor(gamma=0.0,n_estimators=150,base_score=0.7,colsample_bytree=1,learning_rate=0.05)
xgbModel5 = regressor.fit(X_train_FI,y_train_FI,
 eval_set = [(X_train_FI, y_train_FI), (X_test_FI, y_test_FI)],
 verbose=False)
eval_result = regressor.evals_result()
training_rounds = range(len(eval_result['validation_0']['rmse']))
plt.bar([i for i in range(len(xgbModel5.feature_importances_))], xgbModel5.feature_importances_.tolist(), tick_label=X_test_FI.columns)


(X_train_FI, y_train_FI), (X_test_FI, y_test_FI) = get_feature_importance_data(dataset_TI_df,'12ema')
regressor = xgb.XGBRegressor(gamma=0.0,n_estimators=150,base_score=0.7,colsample_bytree=1,learning_rate=0.05)
xgbModel6 = regressor.fit(X_train_FI,y_train_FI,
 eval_set = [(X_train_FI, y_train_FI), (X_test_FI, y_test_FI)],
 verbose=False)
eval_result = regressor.evals_result()
training_rounds = range(len(eval_result['validation_0']['rmse']))
plt.bar([i for i in range(len(xgbModel6.feature_importances_))], xgbModel6.feature_importances_.tolist(), tick_label=X_test_FI.columns)

(X_train_FI, y_train_FI), (X_test_FI, y_test_FI) = get_feature_importance_data(dataset_TI_df,'upper_band')
regressor = xgb.XGBRegressor(gamma=0.0,n_estimators=150,base_score=0.7,colsample_bytree=1,learning_rate=0.05)
xgbModel7 = regressor.fit(X_train_FI,y_train_FI,
 eval_set = [(X_train_FI, y_train_FI), (X_test_FI, y_test_FI)],
 verbose=False)
eval_result = regressor.evals_result()
training_rounds = range(len(eval_result['validation_0']['rmse']))
plt.bar([i for i in range(len(xgbModel7.feature_importances_))], xgbModel7.feature_importances_.tolist(), tick_label=X_test_FI.columns)

(X_train_FI, y_train_FI), (X_test_FI, y_test_FI) = get_feature_importance_data(dataset_TI_df,'lower_band')
regressor = xgb.XGBRegressor(gamma=0.0,n_estimators=150,base_score=0.7,colsample_bytree=1,learning_rate=0.05)
xgbModel8 = regressor.fit(X_train_FI,y_train_FI,
 eval_set = [(X_train_FI, y_train_FI), (X_test_FI, y_test_FI)],
 verbose=False)
eval_result = regressor.evals_result()
training_rounds = range(len(eval_result['validation_0']['rmse']))
plt.bar([i for i in range(len(xgbModel8.feature_importances_))], xgbModel8.feature_importances_.tolist(), tick_label=X_test_FI.columns)
```

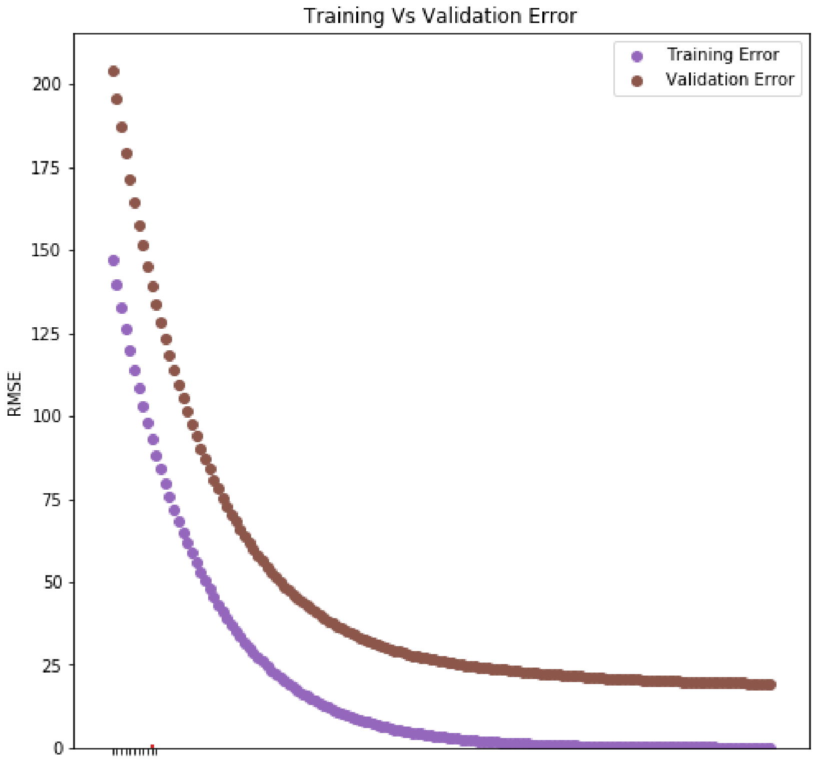
代码

```python
(X_train_FI, y_train_FI), (X_test_FI, y_test_FI) = get_feature_importance_data(dataset_TI_df,'ema')
regressor = xgb.XGBRegressor(gamma=0.0,n_estimators=150,base_score=0.7,colsample_bytree=1,learning_rate=0.05)
xgbModel9 = regressor.fit(X_train_FI,y_train_FI,
 eval_set = [(X_train_FI, y_train_FI), (X_test_FI, y_test_FI)],
 verbose=False)
eval_result = regressor.evals_result()
training_rounds = range(len(eval_result['validation_0']['rmse']))
plt.bar([i for i in range(len(xgbModel9.feature_importances_))], xgbModel9.feature_importances_.tolist(), tick_label=X_test_FI.columns)

(X_train_FI, y_train_FI), (X_test_FI, y_test_FI) = get_feature_importance_data(dataset_TI_df,'momentum')
regressor = xgb.XGBRegressor(gamma=0.0,n_estimators=150,base_score=0.7,colsample_bytree=1,learning_rate=0.05)
xgbModel10 = regressor.fit(X_train_FI,y_train_FI,
 eval_set = [(X_train_FI, y_train_FI), (X_test_FI, y_test_FI)],
 verbose=False)
eval_result = regressor.evals_result()
training_rounds = range(len(eval_result['validation_0']['rmse']))
plt.bar([i for i in range(len(xgbModel10.feature_importances_))], xgbModel10.feature_importances_.tolist(), tick_label=X_test_FI.columns)


(X_train_FI, y_train_FI), (X_test_FI, y_test_FI) = get_feature_importance_data(dataset_TI_df,'log_momentum')
regressor = xgb.XGBRegressor(gamma=0.0,n_estimators=150,base_score=0.7,colsample_bytree=1,learning_rate=0.05)
xgbModel11 = regressor.fit(X_train_FI,y_train_FI,
 eval_set = [(X_train_FI, y_train_FI), (X_test_FI, y_test_FI)],
 verbose=False)
eval_result = regressor.evals_result()
training_rounds = range(len(eval_result['validation_0']['rmse']))
plt.bar([i for i in range(len(xgbModel11.feature_importances_))], xgbModel11.feature_importances_.tolist(), tick_label=X_test_FI.columns)


plt.title('Figure 6: Feature importance of the technical indicators.')
plt.show()
print(dataset_TI_df['20sd'])
print(X_test_FI.columns)
```

七、以12个技术指标为例，进一步挖掘股票特征

跟踪训练误差与真实误差，防止过拟合（若过拟合需要正则化参数）

最终提取了12个技术特征。事实上原project还合并了傅里叶特征，ARIMA特征，一共提炼了112个特征，如果再用ARIMAmodel去分析所有特征内存完全不够用。



Days

```
Total dataset has 2265 samples, and 12 features.
           Close         ma7        ma21  ...         ema    momentum  log_momentum
0     168.839996         NaN         NaN  ...  168.839996  167.839996      5.123011
1     173.080002         NaN         NaN  ...  172.020000  172.080002      5.147959
2     176.139999         NaN         NaN  ...  174.872307  175.139999      5.165586
3     174.259995         NaN         NaN  ...  174.458996  173.259995      5.154793
4     177.669998         NaN         NaN  ...  176.608510  176.669998      5.174284
...          ...         ...         ...  ...         ...         ...           ...
2260  156.350006  166.620002  179.774763  ...  158.564897  155.350006      5.045681
2261  162.929993  165.214286  178.528572  ...  161.474961  161.929993      5.087164
2262  165.410004  164.842858  177.150953  ...  164.098323  164.410004      5.102363
2263  163.029999  163.632858  175.692858  ...  163.386107  162.029999      5.087781
2264  167.050003  163.318573  174.202382  ...  165.828704  166.050003      5.112289
```

七、以12个技术指标为例，进一步挖掘股票特征

MACD，em26等指标最为重要，但事实上都可以考虑入最后的学习。

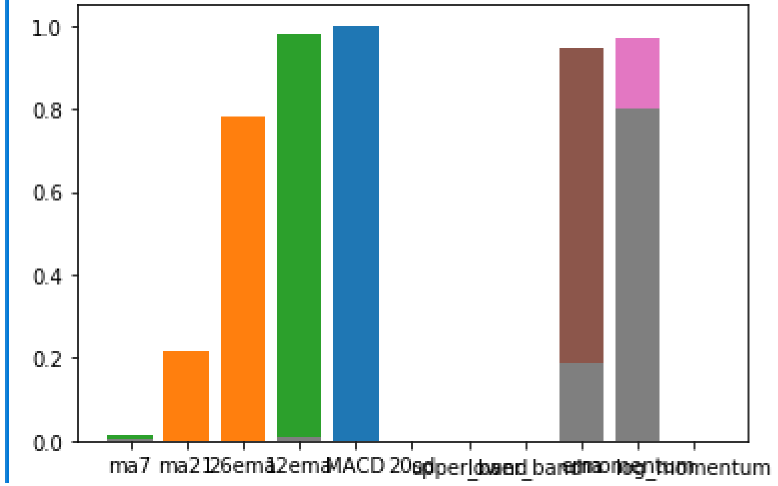


Figure 6: Feature importance of the technical indicators.

八利用自动编码解码器，自我学习，进一步提炼挖掘股票特征数据

代码

```python
#import warnings
from mxnet import nd, autograd, gluon
from mxnet.gluon import nn, rnn
import time
VAE_data=dataset_TI_df
batch_size = 64
n_batches = VAE_data.shape[0]/batch_size
VAE_data = VAE_data.values
train_iter = mx.io.NDArrayIter(data={'data': VAE_data[:num_training_days,:-1]},
 label={'label': VAE_data[:num_training_days, -1]}, batch_size = batch_size)
test_iter = mx.io.NDArrayIter(data={'data': VAE_data[num_training_days:,:-1]},
 label={'label': VAE_data[num_training_days:,-1]}, batch_size = batch_size)
model_ctx = mx.cpu()
class VAE(gluon.HybridBlock):
    def __init__(self, n_hidden=400, n_latent=2, n_layers=1, n_output=784,
    batch_size=100, act_type='gelu', **kwargs):
        self.soft_zero = 1e-10
        self.n_latent = n_latent
        self.batch_size = batch_size
        self.output = None
        self.mu = None
        super(VAE, self).__init__(**kwargs)
        with self.name_scope():
            self.encoder = nn.HybridSequential(prefix='encoder')

        for i in range(n_layers):
            self.encoder.add(nn.Dense(n_hidden, activation=act_type))
            self.encoder.add(nn.Dense(n_latent*2, activation=None))
            self.decoder = nn.HybridSequential(prefix='decoder')
        for i in range(n_layers):
            self.decoder.add(nn.Dense(n_hidden, activation=act_type))
            self.decoder.add(nn.Dense(n_output, activation='sigmoid'))
    def hybrid_forward(self, F, x):
        h = self.encoder(x)
        print(h)
        mu_lv = F.split(h, axis=1, num_outputs=2)
        mu = mu_lv[0]
        lv = mu_lv[1]
```

代码

```python
        self.mu = mu
        eps = F.random_normal(loc=0, scale=1, shape=(self.batch_size, self.n_latent), ctx=model_ctx)
        z = mu + F.exp(0.5*lv)*eps
        y = self.decoder(z)
        self.output = y
        KL = 0.5*F.sum(1+lv-mu*mu-F.exp(lv),axis=1)
        logloss = F.sum(x*F.log(y+self.soft_zero)+ (1-x)*F.log(1-y+self.soft_zero), axis=1)
        loss = -logloss-KL
        return loss
n_hidden=400 # neurons in each layer
n_latent=2
n_layers=3 # num of dense layers in encoder and decoder respectively
n_output=VAE_data.shape[1]-1

net = VAE(n_hidden=n_hidden, n_latent=n_latent, n_layers=n_layers, n_output=n_output, batch_size=batch_size, act_type='relu')
net.collect_params().initialize(mx.init.Xavier(), ctx=mx.cpu())
net.hybridize()
trainer = gluon.Trainer(net.collect_params(), 'adam', {'learning_rate': .01})
print(net)
#
#
#batch.data=dataset_TI_df

n_epoch = 150
print_period = n_epoch // 10
start = time.time()

training_loss = []
validation_loss = []
for epoch in range(n_epoch):
    epoch_loss = 0
    epoch_val_loss = 0

    train_iter.reset()
    test_iter.reset()

    n_batch_train = 0
    for batch in train_iter:
        n_batch_train +=1
```

八利用自动编码解码器，自我学习，进一步提炼挖掘股票特征数据

代码

```python
n_epoch = 150
print_period = n_epoch // 10
start = time.time()

training_loss = []
validation_loss = []
for epoch in range(n_epoch):
    epoch_loss = 0
    epoch_val_loss = 0

    train_iter.reset()
    test_iter.reset()

    n_batch_train = 0
    for batch in train_iter:
        n_batch_train +=1
        data = batch.data[0].as_in_context(mx.cpu())


        with autograd.record():
            loss = net(data)
        loss.backward()
        trainer.step(data.shape[0])
        epoch_loss += nd.mean(loss).asscalar()

    n_batch_val = 0
    for batch in test_iter:
        n_batch_val +=1
        data = batch.data[0].as_in_context(mx.cpu())
        loss = net(data)
        epoch_val_loss += nd.mean(loss).asscalar()

    epoch_loss /= n_batch_train
    epoch_val_loss /= n_batch_val

    training_loss.append(epoch_loss)
```

代码

```python
    training_loss.append(epoch_loss)
    validation_loss.append(epoch_val_loss)

    """if epoch % max(print_period, 1) == 0:
        print('Epoch {}, Training loss {:.2f}, Validation loss {:.2f}'.
            format(epoch, epoch_loss, epoch_val_loss))"""

end = time.time()
print('Training completed in {} seconds.'.format(int(end-start)))
#
dataset_TI_df['Date'] = dataset_ex_df['Date']
vae_added_df = mx.nd.array(dataset_TI_df.iloc[:, :-1].values)
print('The shape of the newly created (from the autoencoder) features is {}.'.format(vae_added_df.shape))
print(vae_added_df.shape)
```

八利用自动编码解码器，自我学习，进一步提炼挖掘股票特征数据

激活函数gelu，relu（把原代码中解码网络的激活函数改为其他的，不使用编码器的gelu）

打印编码器，解码器，训练时间，提取的特征数

Figure 7: GELU as an activation function for autoencoders

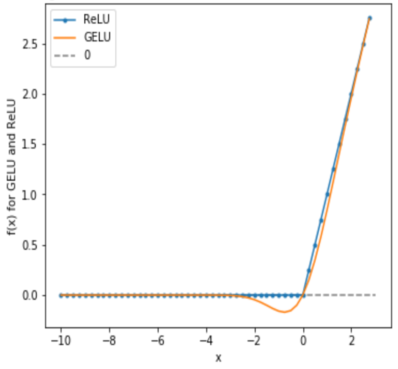
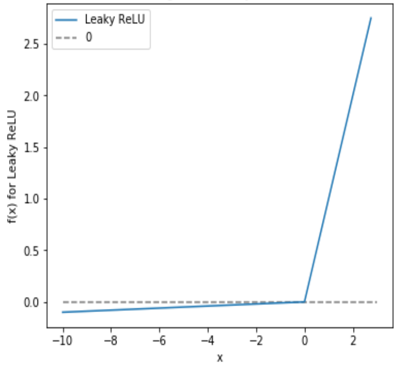


Figure 8: LeakyReLU




```
VAE(
  (encoder): HybridSequential(
    (0): Dense(None -> 400, Activation(relu))
    (1): Dense(None -> 4, linear)
    (2): Dense(None -> 400, Activation(relu))
    (3): Dense(None -> 4, linear)
    (4): Dense(None -> 400, Activation(relu))
    (5): Dense(None -> 4, linear)
  )
  (decoder): HybridSequential(
    (0): Dense(None -> 400, Activation(relu))
    (1): Dense(None -> 11, Activation(sigmoid))
    (2): Dense(None -> 400, Activation(relu))
    (3): Dense(None -> 11, Activation(sigmoid))
    (4): Dense(None -> 400, Activation(relu))
```

```
VAE(
  (encoder): HybridSequential(
    (0): Dense(None -> 400, Activation(relu))
    (1): Dense(None -> 4, linear)
    (2): Dense(None -> 400, Activation(relu))
    (3): Dense(None -> 4, linear)
    (4): Dense(None -> 400, Activation(relu))
    (5): Dense(None -> 4, linear)
  )
  (decoder): HybridSequential(
    (0): Dense(None -> 400, Activation(relu))
    (1): Dense(None -> 11, Activation(sigmoid))
    (2): Dense(None -> 400, Activation(relu))
    (3): Dense(None -> 11, Activation(sigmoid))
    (4): Dense(None -> 400, Activation(relu))
    (5): Dense(None -> 11, Activation(sigmoid))
  )
)
<Symbol dense29_fwd>
Training completed in 99 seconds.
The shape of the newly created (from the autoencoder) features is (2265, 12).
(2265, 12)
RNNModel(
  (rnn): LSTM(12 -> 500, TNC)
  (decoder): Dense(500 -> 1, linear)
)
```

九*、利用PCA（K-means，tsne，Isomap等也可考虑）分析主成分，进一步降维，减少数据

代码（直接调取sklearn的数据挖掘包，但已注释，不使用）

- 由于是以技术指标为例子，经自我学习后，一共只有12个特征，没必要也不适合再进一步降维。
- 原项目中一共112个特征，可以使用PCA等方法进一步归纳

```
#from utils import *
#import time
#import numpy as np
#from mxnet import nd, autograd, gluon
#from mxnet.gluon import nn, rnn
#import mxnet as mx
#import datetime
#import seaborn as sns
#import matplotlib.pyplot as plt
#from sklearn.decomposition import PCA
#import math
#from sklearn.preprocessing import MinMaxScaler
#from sklearn.metrics import mean_squared_error
#from sklearn.preprocessing import StandardScaler
#import xgboost as xgb
#from sklearn.metrics import accuracy_score
#import warnings
```

十、用时间序列生成器作为对抗网络的生成器（generator）

代码

```python
gan_num_features = vae_added_df.shape[1]
sequence_length = 17
class RNNModel(gluon.Block):
    def __init__(self, num_embed, num_hidden, num_layers, bidirectional=False,
                 sequence_length=sequence_length, **kwargs):
        super(RNNModel, self).__init__(**kwargs)
        self.num_hidden = num_hidden
        with self.name_scope():
            self.rnn = rnn.LSTM(num_hidden, num_layers, input_size=num_embed,
                                bidirectional=bidirectional, layout='TNC')

            self.decoder = nn.Dense(1, in_units=num_hidden)

    def forward(self, inputs, hidden):
        output, hidden = self.rnn(inputs, hidden)
        decoded = self.decoder(output.reshape((-1, self.num_hidden)))
        return decoded, hidden

    def begin_state(self, *args, **kwargs):
        return self.rnn.begin_state(*args, **kwargs)

lstm_model = RNNModel(num_embed=gan_num_features, num_hidden=500, num_layers=1)
lstm_model.collect_params().initialize(mx.init.Xavier(), ctx=mx.cpu())
trainer = gluon.Trainer(lstm_model.collect_params(), 'adam', {'learning_rate': .01})
loss = gluon.loss.L1Loss()

print(lstm_model)
```

打印时间序列生成器

```
RNNModel(
  (rnn): LSTM(12 -> 500, TNC)
  (decoder): Dense(500 -> 1, linear)
)
```

十一，超参数优化器——学习率优化（其实单就程序实现而言，可以直接手段选取较小的学习率）

代码

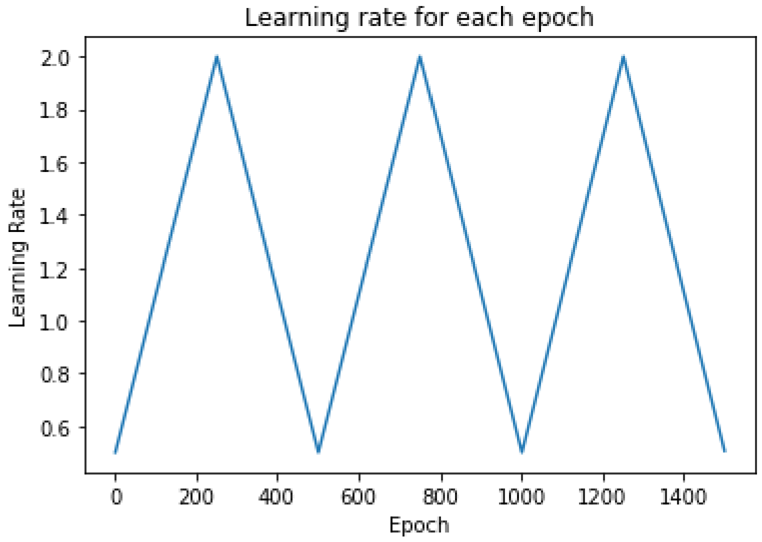
```python
class TriangularSchedule():
    def __init__(self, min_lr, max_lr, cycle_length, inc_fraction=0.5):
        self.min_lr = min_lr
        self.max_lr = max_lr
        self.cycle_length = cycle_length
        self.inc_fraction = inc_fraction

    def __call__(self, iteration):
        if iteration <= self.cycle_length*self.inc_fraction:
            unit_cycle = iteration * 1 / (self.cycle_length * self.inc_fraction)
        elif iteration <= self.cycle_length:
            unit_cycle = (self.cycle_length - iteration) * 1 / (self.cycle_length * (1 - self.inc_fraction))
        else:
            unit_cycle = 0
        adjusted_cycle = (unit_cycle * (self.max_lr - self.min_lr)) + self.min_lr
        return adjusted_cycle
class CyclicalSchedule():
    def __init__(self, schedule_class, cycle_length, cycle_length_decay=1, cycle_magnitude_decay=1, **kwargs
        self.schedule_class = schedule_class
        self.length = cycle_length
        self.length_decay = cycle_length_decay
        self.magnitude_decay = cycle_magnitude_decay
        self.kwargs = kwargs

    def __call__(self, iteration):
        cycle_idx = 0
        cycle_length = self.length
        idx = self.length
        while idx <= iteration:
            cycle_length = math.ceil(cycle_length * self.length_decay)
            cycle_idx += 1
            idx += cycle_length
        cycle_offset = iteration - idx + cycle_length

        schedule = self.schedule_class(cycle_length=cycle_length, **self.kwargs)
        return schedule(cycle_offset) * self.magnitude_decay**cycle_idx
schedule = CyclicalSchedule(TriangularSchedule, min_lr=0.5, max_lr=2, cycle_length=500)
iterations=1500
plt.plot([i+1 for i in range(iterations)],[schedule(i) for i in range(iterations)])
```

代码

```python
    def __init__(self, schedule_class, cycle_length, cycle_length_decay=1, cycle_magnitude_decay=1, **kwargs):
        self.schedule_class = schedule_class
        self.length = cycle_length
        self.length_decay = cycle_length_decay
        self.magnitude_decay = cycle_magnitude_decay
        self.kwargs = kwargs

    def __call__(self, iteration):
        cycle_idx = 0
        cycle_length = self.length
        idx = self.length
        while idx <= iteration:
            cycle_length = math.ceil(cycle_length * self.length_decay)
            cycle_idx += 1
            idx += cycle_length
        cycle_offset = iteration - idx + cycle_length

        schedule = self.schedule_class(cycle_length=cycle_length, **self.kwargs)
        return schedule(cycle_offset) * self.magnitude_decay**cycle_idx
schedule = CyclicalSchedule(TriangularSchedule, min_lr=0.5, max_lr=2, cycle_length=500)
iterations=1500
plt.plot([i+1 for i in range(iterations)],[schedule(i) for i in range(iterations)])
plt.title('Learning rate each epoch')
plt.xlabel("Epoch")
plt.ylabel("Learning Rate")
plt.show()
```

十一，超参数优化器——学习率优化（其实单就程序实现而言，可以直接手段选取较小的学习率）

打印各步学习率



十二、以cnn网络为辨别器(discriminator)

代码

```python
num_fc = 512
# ... other parts of the GAN
cnn_net = gluon.nn.Sequential()
with net.name_scope():

 # Add the 1D Convolutional layers
     cnn_net.add(gluon.nn.Conv1D(32, kernel_size=5, strides=2))
     cnn_net.add(nn.LeakyReLU(0.01))
     cnn_net.add(gluon.nn.Conv1D(64, kernel_size=5, strides=2))
     cnn_net.add(nn.LeakyReLU(0.01))
     cnn_net.add(nn.BatchNorm())
     cnn_net.add(gluon.nn.Conv1D(128, kernel_size=5, strides=2))
     cnn_net.add(nn.LeakyReLU(0.01))
     cnn_net.add(nn.BatchNorm())

 # Add the two Fully Connected layers
     cnn_net.add(nn.Dense(220, use_bias=False), nn.BatchNorm(), nn.LeakyReLU(0.01))
     cnn_net.add(nn.Dense(220, use_bias=False), nn.Activation(activation='relu'))
     cnn_net.add(nn.Dense(1))

# ... other parts of the GAN
print(cnn_net)
#class GAN():
```

打印cnn网络

```
Sequential(
  (0): Conv1D(None -> 32, kernel_size=(5,), stride=(2,))
  (1): LeakyReLU(0.01)
  (2): Conv1D(None -> 64, kernel_size=(5,), stride=(2,))
  (3): LeakyReLU(0.01)
  (4): BatchNorm(axis=1, eps=1e-05, momentum=0.9, fix_gamma=False, use_global_stats=False, in_channels=None)
  (5): Conv1D(None -> 128, kernel_size=(5,), stride=(2,))
  (6): LeakyReLU(0.01)
  (7): BatchNorm(axis=1, eps=1e-05, momentum=0.9, fix_gamma=False, use_global_stats=False, in_channels=None)
  (8): Dense(None -> 220, linear)
  (9): BatchNorm(axis=1, eps=1e-05, momentum=0.9, fix_gamma=False, use_global_stats=False, in_channels=None)
  (10): LeakyReLU(0.01)
  (11): Dense(None -> 220, linear)
  (12): Activation(relu)
  (13): Dense(None -> 1, linear)
)
            High         Low        Open       Close      Volume   Adj Close
0     170.130005  166.929993  167.289993  168.839996   6401800.0  147.799942
1     174.250000  169.509995  170.050003  173.080002   9135000.0  151.511627
2     176.259995  172.570007  173.000000  176.139999  11659400.0  154.190262
3     175.380005  173.759995  175.380005  174.259995   7381100.0  152.544586
4     178.750000  173.949997  174.320007  177.669998   8727400.0  155.529587
...          ...         ...         ...         ...         ...         ...
2260  160.000000  154.309998  159.000000  156.350006   3783500.0  154.055573
2261  163.110001  151.699997  157.000000  162.929993   7054700.0  160.539001
2262  165.410004  159.020004  160.119995  165.410004   4973000.0  162.982620
2263  165.949997  162.020004  165.639999  163.029999   4110500.0  160.637543
2264  167.119995  163.779999  163.779999  167.050003   4550000.0  164.598572

[2265 rows x 6 columns]
MSE Train: 0.06754635
MSE Test: 0.23441519
```

十三、用tensorflow架构神经网络，实现对GS的股票预测

代码

```
mport tensorflow as tf
rom sklearn.preprocessing import MinMaxScaler
rom tensorflow import keras
rom tensorflow.python.framework import ops
ata=pd.read_csv('dataGS.csv')
ata.drop('Date',axis=1,inplace=True)
rint(data)
eedtotrain = data.iloc[:int(data.shape[0] * 0.85), :]
eedtotest = data.iloc[int(data.shape[0] * 0.85):, :]
caler = MinMaxScaler(feature_range=(-1, 1))
caler.fit(needtotrain)
imout = 1
ayer1 = 1112
ayer2 = 556
ayer3 = 278
ayer4 = 189
atch_size = 256
pochs = 12
eedtotrain = scaler.transform(needtotrain)
eedtotest = scaler.transform(needtotest)
needtotrain = needtotrain[:, 1:]
needtotrain = needtotrain[:, 0]
needtotest = needtotest[:, 1:]
needtotest = needtotest[:, 0]
imin = Xneedtotrain.shape[1]
ps.reset_default_graph()
 = tf.placeholder(shape=[None, dimin], dtype=tf.float32)
 = tf.placeholder(shape=[None], dtype=tf.float32)
1 = tf.get_variable('a1', [dimin, layer1], initializer=tf.contrib.layers.xavier_initializer(seed=1))
1 = tf.get_variable('b1', [layer1], initializer=tf.zeros_initializer())
2 = tf.get_variable('a2', [layer1, layer2], initializer=tf.contrib.layers.xavier_initializer(seed=1))
2 = tf.get_variable('b2', [layer2], initializer=tf.zeros_initializer())
3 = tf.get_variable('a3', [layer2, layer3], initializer=tf.contrib.layers.xavier_initializer(seed=1))
3 = tf.get_variable('b3', [layer3], initializer=tf.zeros_initializer())
4 = tf.get_variable('a4', [layer3, layer4], initializer=tf.contrib.layers.xavier_initializer(seed=1))
4 = tf.get_variable('b4', [layer4], initializer=tf.zeros_initializer())
5 = tf.get_variable('a5', [layer4, dimout], initializer=tf.contrib.layers.xavier_initializer(seed=1))
5 = tf.get_variable('b5', [dimout], initializer=tf.zeros_initializer())
1 = tf.nn.relu(tf.add(tf.matmul(X, a1), b1))
```

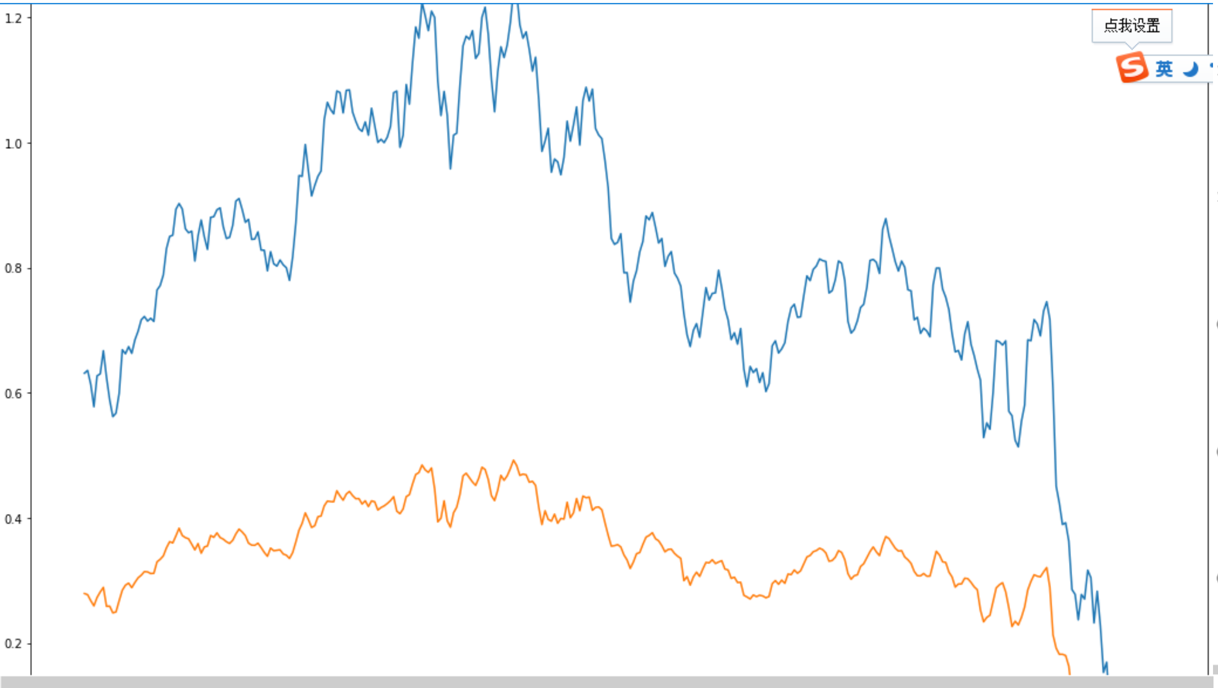
代码

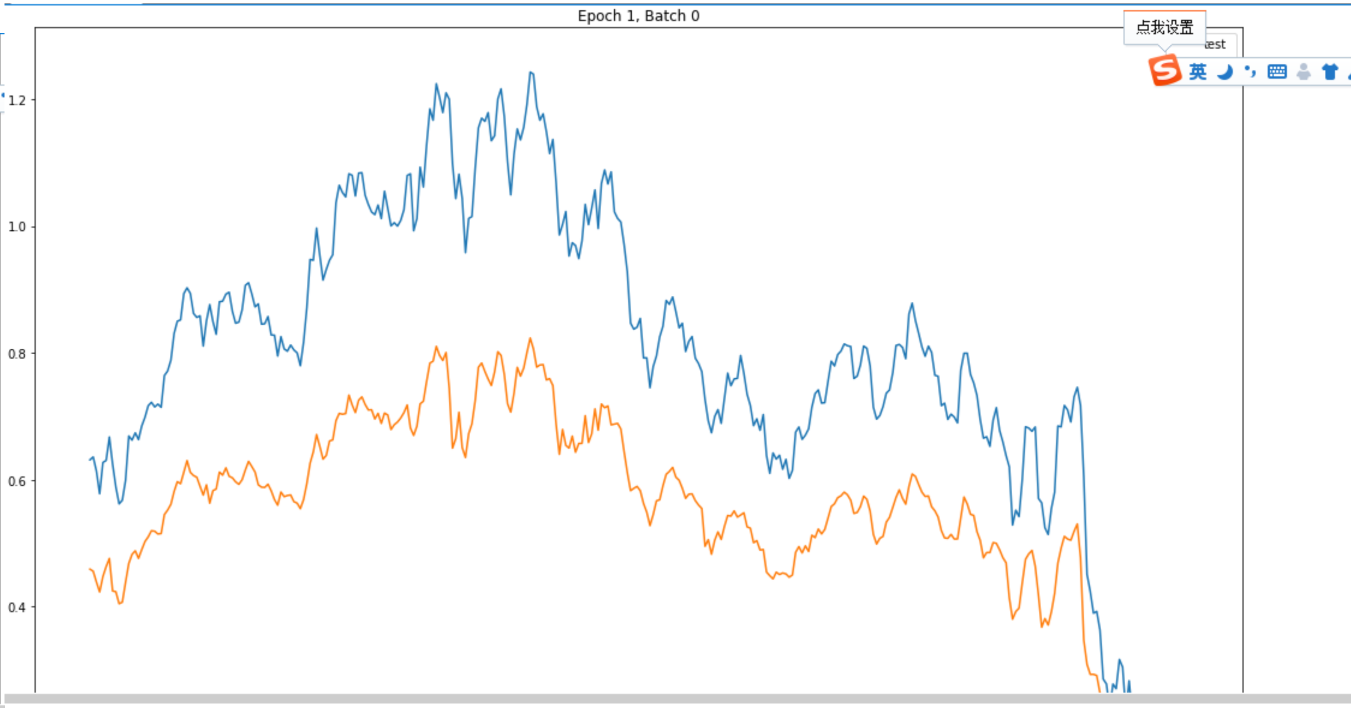
```
b2 = tf.get_variable('b2', [layer2], initializer=tf.zeros_initializer())
a3 = tf.get_variable('a3', [layer2, layer3], initializer=tf.contrib.layers.xavier_initializer(seed=1))
b3 = tf.get_variable('b3', [layer3], initializer=tf.zeros_initializer())
a4 = tf.get_variable('a4', [layer3, layer4], initializer=tf.contrib.layers.xavier_initializer(seed=1))
b4 = tf.get_variable('b4', [layer4], initializer=tf.zeros_initializer())
a5 = tf.get_variable('a5', [layer4, dimout], initializer=tf.contrib.layers.xavier_initializer(seed=1))
b5 = tf.get_variable('b5', [dimout], initializer=tf.zeros_initializer())
c1 = tf.nn.relu(tf.add(tf.matmul(X, a1), b1))
c2 = tf.nn.relu(tf.add(tf.matmul(c1, a2), b2))
c3 = tf.nn.relu(tf.add(tf.matmul(c2, a3), b3))
c4 = tf.nn.relu(tf.add(tf.matmul(c3, a4), b4))
output = tf.transpose(tf.add(tf.matmul(c4, a5), b5))
loss = tf.reduce_mean(tf.squared_difference(output, Y))
optimizer = tf.train.AdamOptimizer().minimize(loss)

with tf.Session() as sess:
    sess.run(tf.global_variables_initializer())
    for e in range(epochs):
        shuffle_indices = np.random.permutation(np.arange(yneedtotrain.shape[0]))
        Xneedtotrain = Xneedtotrain[shuffle_indices]
        yneedtotrain = yneedtotrain[shuffle_indices]
        for i in range(yneedtotrain.shape[0] // batch_size):
            start = i * batch_size
            batch_x = Xneedtotrain[start : start + batch_size]
            batch_y = yneedtotrain[start : start + batch_size]
            sess.run(optimizer, feed_dict={X: batch_x, Y: batch_y})
            if i % 50 == 0:
                print('MSE Train:', sess.run(loss, feed_dict={X: Xneedtotrain, Y: yneedtotrain}))
                print('MSE Test:', sess.run(loss, feed_dict={X: Xneedtotest, Y: yneedtotest}))
                y_pred = sess.run(output, feed_dict={X: Xneedtotest})
                y_pred = np.squeeze(y_pred)
                plt.figure(figsize=(18, 15))
                plt.plot(yneedtotest, label='test')
                plt.plot(y_pred, label='pred')
                plt.title('Epoch ' + str(e) + ', Batch ' + str(i))
                plt.legend()
                plt.show()
```

十三、用tensorflow架构神经网络，实现对GS的股票预测

Epoch=0

Epoch=1

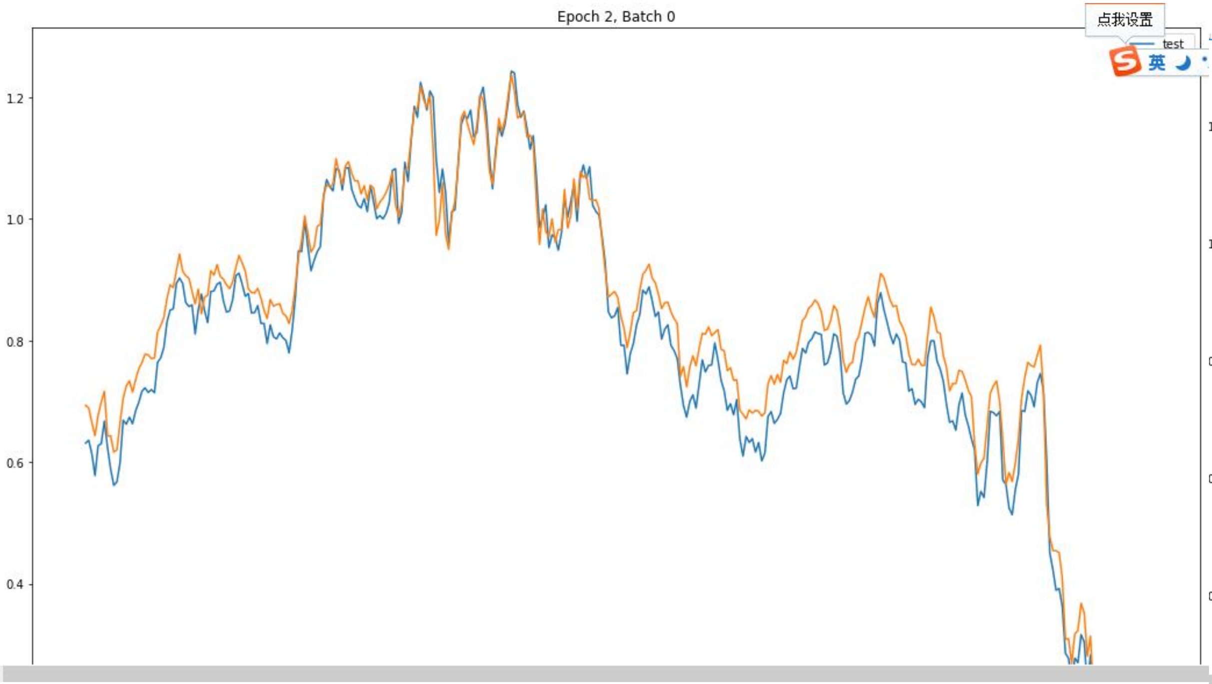


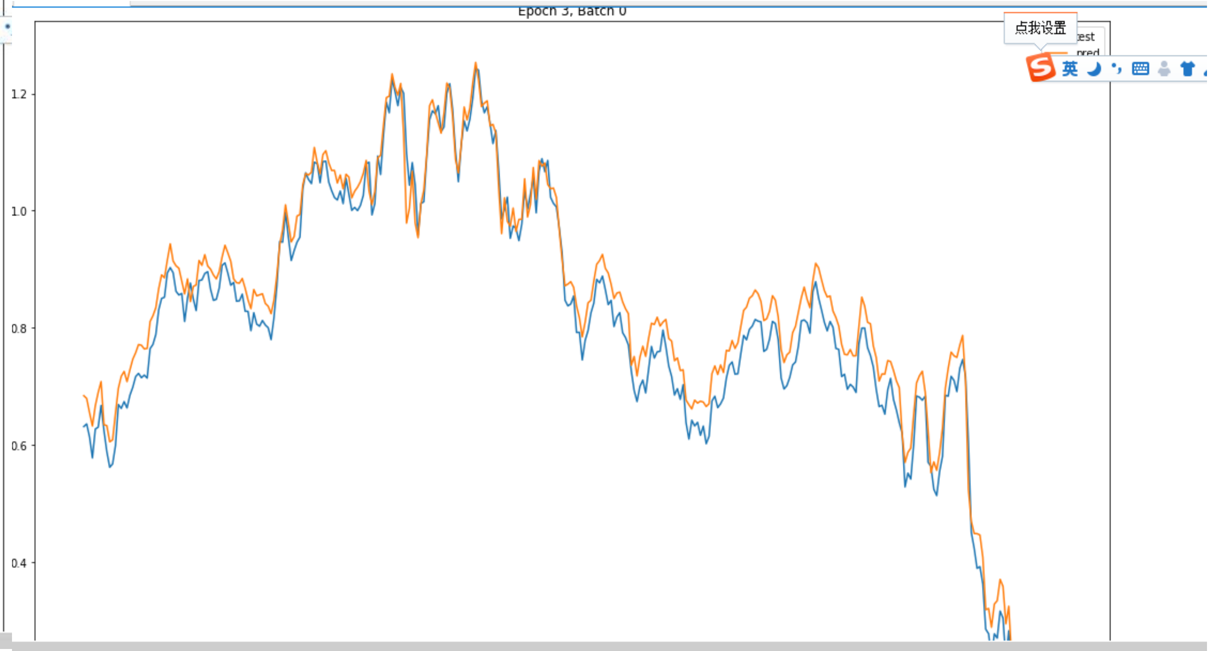


十三、用tensorflow架构神经网络，实现对GS的股票预测

Epoch=2

Epoch=3

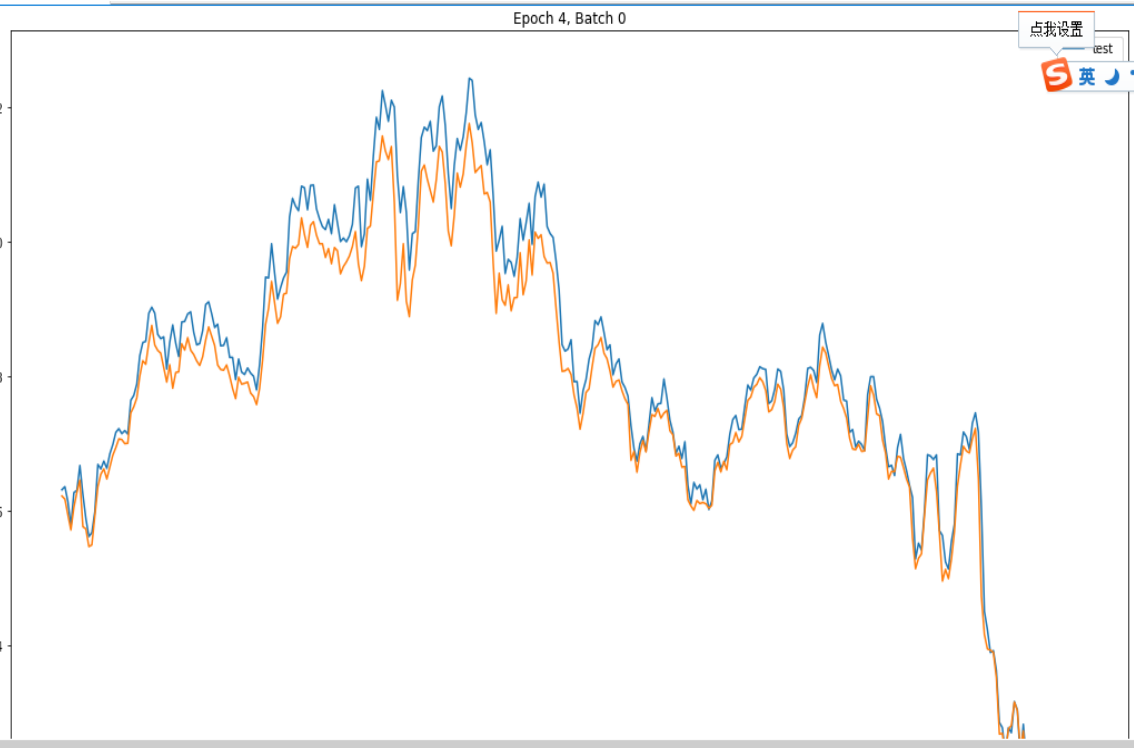


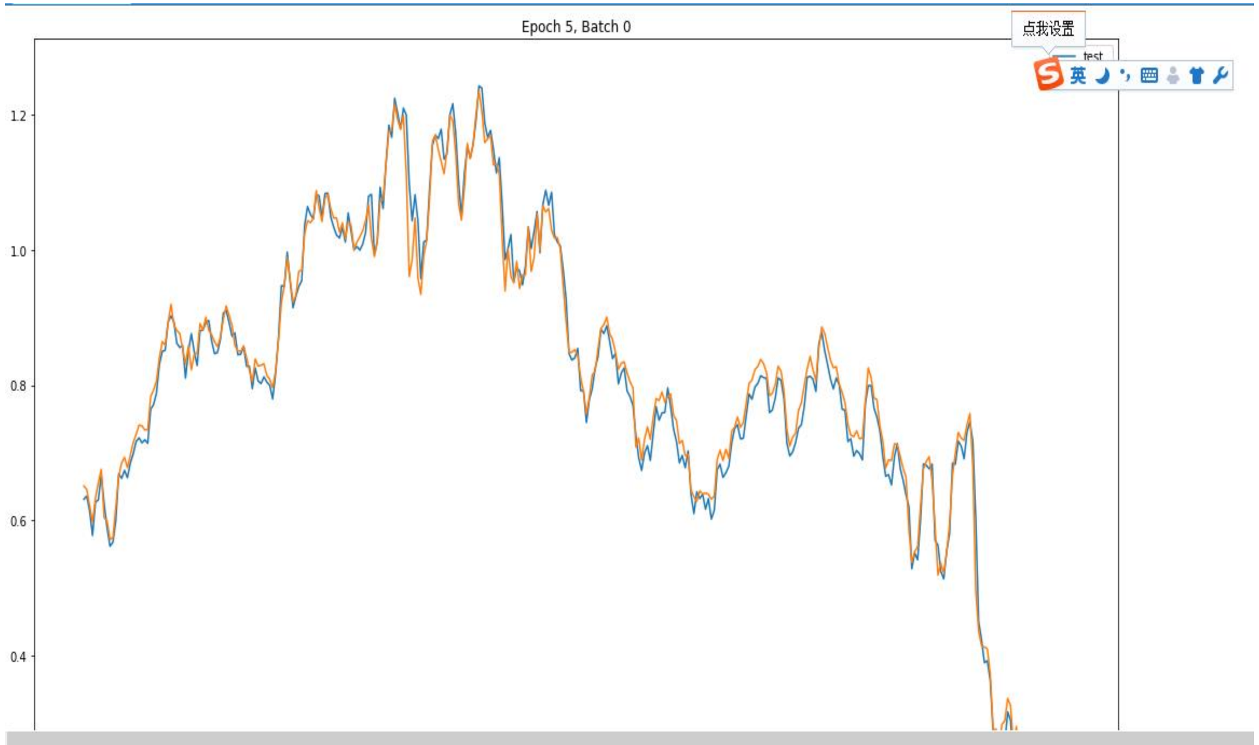


十三、用tensorflow架构神经网络，实现对GS的股票预测

Epoch=4

Epoch=5





十四 展望——优化超参数

- 使用贝叶斯模型，rainbow，PPO等去不断优化学习率以外的其他重要超参数，使得模型几乎成为一个无监督学习的模型