作业三：State-Space 模型与机器学习算法的比较

作者：刘润迪

一、数据准备

选取**SP100list.Rdata**中的**BAC**股票3000个**Close**数据进行分析和预测。

二、State-Space Model

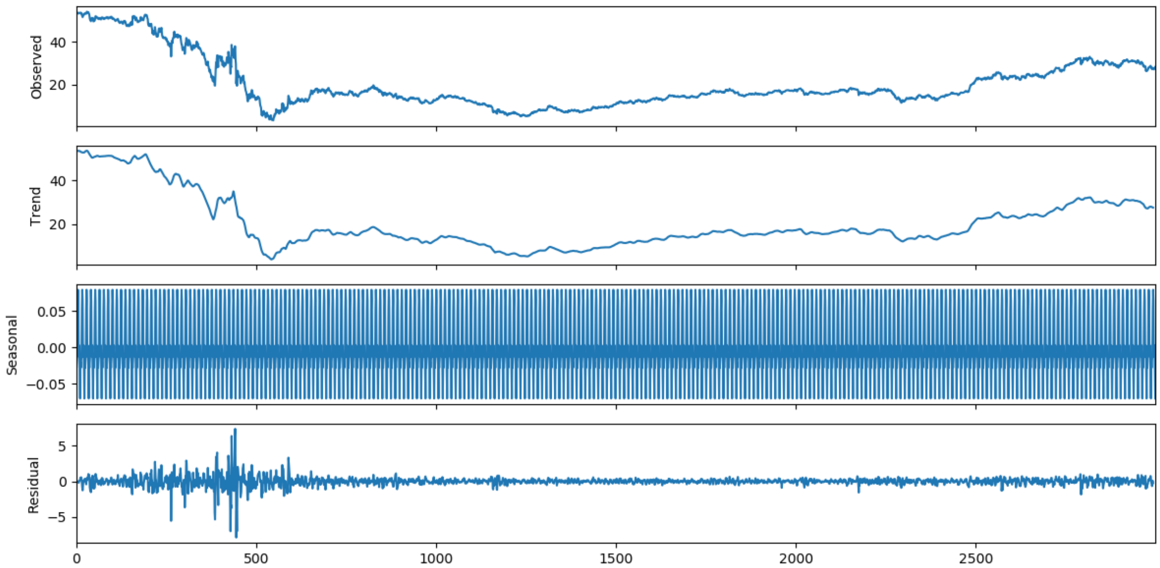
```
# 0. 导入包
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
import datetime
from sklearn.metrics import mean_squared_error
import statsmodels.api as sm
from statsmodels.tsa.stattools import acf
from statsmodels.tsa.stattools import pacf
from statsmodels.tsa.seasonal import seasonal_decompose
from sklearn.metrics import mean_squared_error
from statsmodels.tsa.stattools import adfuller
from statsmodels.tsa.arima_model import ARIMA
import itertools
import warnings
```

```
# 1.加载数据
data0 = pd.read_csv('./BAC.csv')
data=data0['BAC.Open']

train = data[0:3000]
test = data[3000:3005]

train.plot( title= 'BAC_train')
plt.show()
```


```
# 2.分解
decomposition = seasonal_decompose(train, freq=12)
fig = plt.figure()
fig = decomposition.plot()
fig.set_size_inches(12, 6)
plt.show()
```



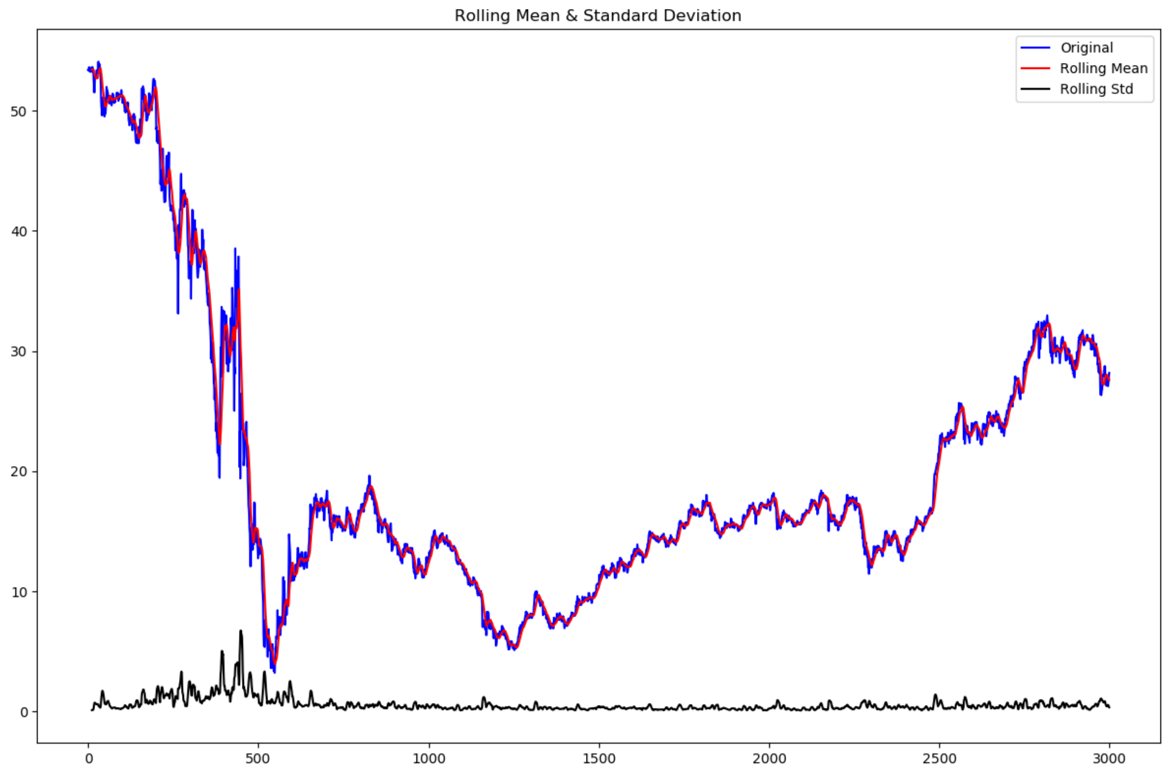

```
# 3.检测稳定性
# 定义检测函数
def test_stationary(train):
    # Determing rolling statistics
    rolmean = train.rolling(window=12).mean()
    rolstd = train.rolling(window=12).std()

    # Plot rolling statistics:
    fig = plt.figure(figsize=(12, 8))
    orig = plt.plot(train, color='blue', label='Original')
    mean = plt.plot(rolmean, color='red', label='Rolling Mean')
    std = plt.plot(rolstd, color='black', label='Rolling Std')
    plt.legend(loc='best')
```

```
    plt.title('Rolling Mean & Standard Deviation')
    plt.show()

    # Perform Dickey-Fuller test:
    print('Results of Dickey-Fuller Test:')
    dftest = adfuller(train, autolag='AIC')
    dfoutput = pd.Series(dftest[0:4], index=['Test Statistic', 'p-value', '#Lags
Used', 'Number of Observations Used'])
    for key, value in dftest[4].items():
        dfoutput['Critical Value (%s)' % key] = value
    print(dfoutput)
# 对训练数据进行稳定性监测
test_stationary(train)
```

Running Result:

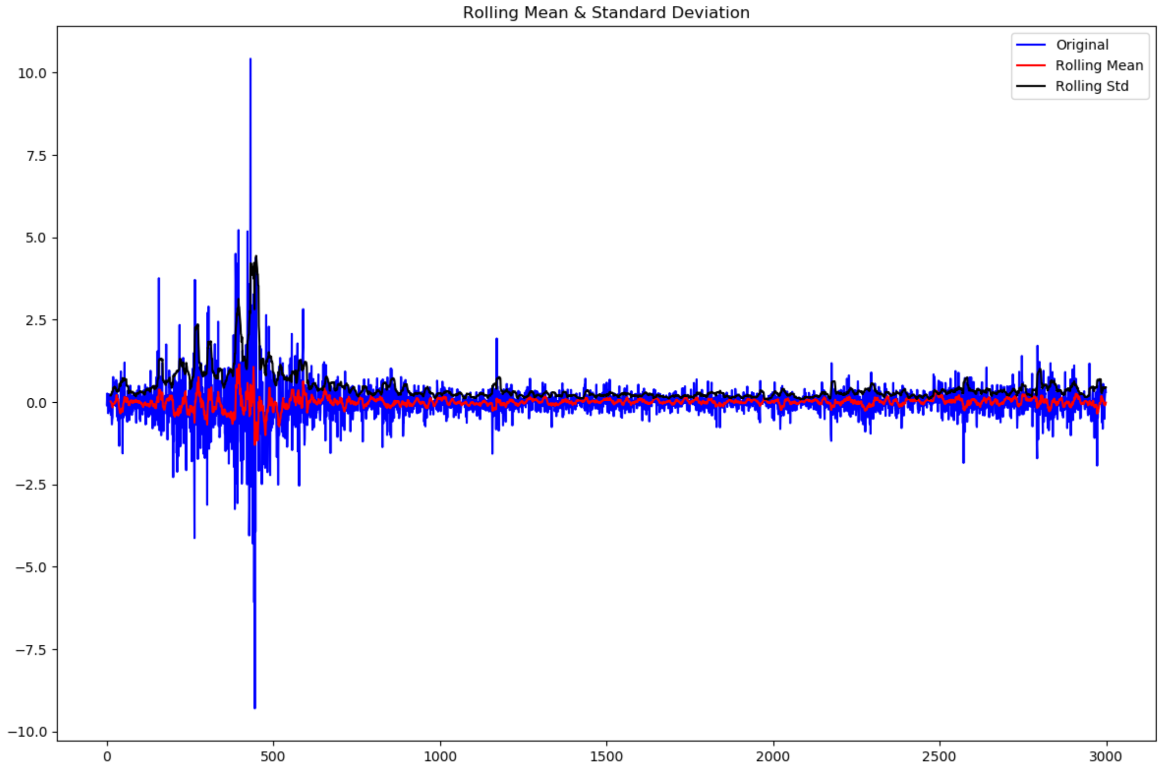



```
Results of Dickey-Fuller Test:
Test Statistic                   -2.883660
p-value                           0.047281
#Lags Used                       26.000000
Number of Observations Used    2973.000000
Critical Value (1%)              -3.432551
Critical Value (5%)              -2.862513
Critical Value (10%)             -2.567288
dtype: float64
```

由上表得：t统计量大于Critical Value（10%）可以得出，整体的序列并没有到达稳定性要求

```
# 4. 进行一阶差分并检测稳定性
first_difference = train.diff(1)
test_stationary(first_difference.dropna(inplace=False))
```

Running Result:




```
Results of Dickey-Fuller Test:
Test Statistic                  -1.126936e+01
p-value                          1.550611e-20
#Lags Used                       2.500000e+01
Number of Observations Used      2.973000e+03
Critical Value (1%)             -3.432551e+00
Critical Value (5%)             -2.862513e+00
Critical Value (10%)            -2.567288e+00
```

结合图片，由上表得：p值几乎为0且t统计量均小于三个Critical Value可以得出，到达稳定性要求。

```
# 5.确定阶数（利用BIC）
pmax = 6
qmax = 6
# bic矩阵
bic_matrix = []
for p in range(pmax + 1):
    tmp = []
    for q in range(qmax + 1):
        # 存在部分报错，所以用try来跳过报错。
        try:
            print(ARIMA(train, (p, 1, q)).fit().aic)
            tmp.append(ARIMA(train, (p, 1, q)).fit().aic)
        except:
            tmp.append(100000)
    aic_matrix.append(tmp)

# 从中可以找出最小值
bic_matrix = pd.DataFrame(bic_matrix)
p, q = aic_matrix.stack().idxmin()
print(u'BIC最小的p值和q值为：%s、%s' % (p, q))
```

Running Result:

```
BIC最小的p值和q值为：5、2
```

```
# 6. 确定季节性参数
p = d = q = range(0, 3)
seasonal_pdq = [(x[0], x[1], x[2], 12) for x in list(itertools.product(p, d,
q))]
score_aic = 1000000.0
warnings.filterwarnings("ignore") # specify to ignore warning messages
for param_seasonal in seasonal_pdq:
    mod = sm.tsa.statespace.SARIMAX(data,
                                    order=[5,1,2],
                                    seasonal_order=param_seasonal,
                                    enforce_stationarity=False,
                                    enforce_invertibility=False)
    results = mod.fit()
    print('x{}12 - AIC:{}'.format(param_seasonal, results.aic))
    if results.aic < score_aic:
        score_aic = results.aic
        params = param_seasonal, results.aic
param_seasonal, results.aic = params
print('x{}12 - AIC:{}'.format(param_seasonal, results.aic))
```

Running Result:

```
1,0,2   6704
x(2, 0, 1, 12)12 - AIC:6701.730989960542

x(2, 0, 1, 12)12 - AIC:6701.730989960542
```

```
# 7.模型训练
mod = sm.tsa.statespace.SARIMAX(train,
                                order=(5, 1, 2),
                                seasonal_order=(2, 0, 1, 12),
                                enforce_stationarity=False,
                                enforce_invertibility=False).fit()
print(mod.summary())
```

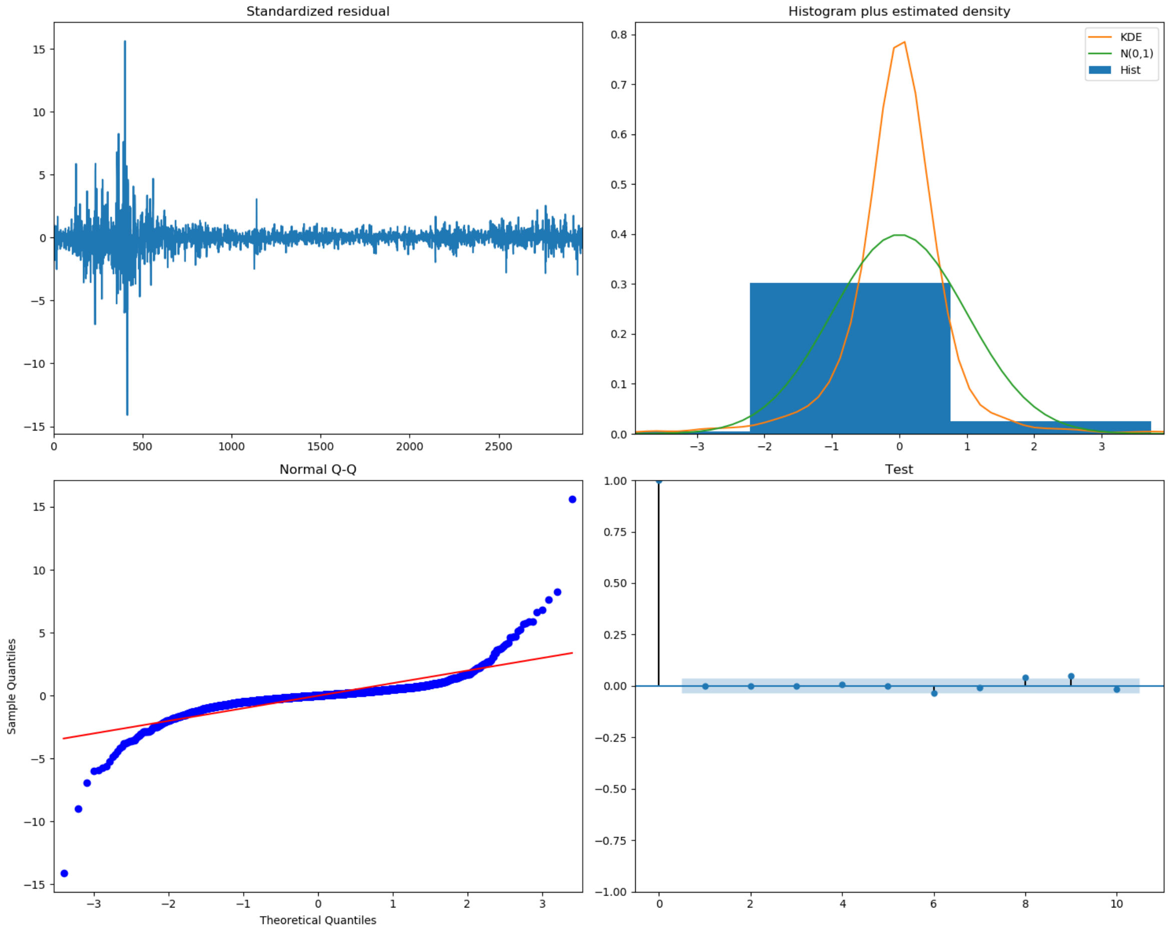
Running Result:

```
 Statespace Model Results
================================================================================
==========
Dep. Variable:                        BAC.Open   No. Observations:
      3000
Model:             SARIMAX(5, 1, 2)x(2, 0, 1, 12)   Log Likelihood
-2906.629
Date:                         Fri, 14 Aug 2020   AIC
  5835.258
Time:                                 23:44:51   BIC
  5901.218
Sample:                                      0   HQIC
 5858.996
                                        - 3000
```

```
Covariance Type:                                 opg

==============================================================================
                 coef    std err          z      P>|z|      [0.025      0.975]
------------------------------------------------------------------------------
ar.L1         -0.3224      0.118     -2.725      0.006      -0.554      -0.091
ar.L2          0.3991      0.125      3.198      0.001       0.155       0.644
ar.L3         -0.0316      0.012     -2.563      0.010      -0.056      -0.007
ar.L4         -0.0474      0.012     -3.899      0.000      -0.071      -0.024
ar.L5         -0.0301      0.012     -2.465      0.014      -0.054      -0.006
ma.L1         -0.5821      0.360     -1.615      0.106      -1.289       0.124
ma.L2         -2.4017      0.712     -3.372      0.001      -3.798      -1.006
ar.S.L12      -0.7487      0.073    -10.277      0.000      -0.892      -0.606
ar.S.L24      -0.0428      0.011     -3.918      0.000      -0.064      -0.021
ma.S.L12       0.7547      0.074     10.250      0.000       0.610       0.899
sigma2         0.0716      0.042      1.685      0.092      -0.012       0.155
================================================================================
===
Ljung-Box (Q):                      176.33   Jarque-Bera (JB):
 256153.65
Prob(Q):                              0.00   Prob(JB):
0.00
Heteroskedasticity (H):               0.12   Skew:
0.52
Prob(H) (two-sided):                  0.00   Kurtosis:
 48.48
================================================================================
===
```

```
# 8.模型评估
mod.plot_diagnostics(figsize=(15, 12))
plt.title("Test")
plt.show()
```




```
# 8. 预测
predictions = mod.predict(start=len(train), end=len(train)+len(test)-1,
dynamic=False,tpy='levels')
predictions=np.matrix(predictions)

test=np.matrix(test)
predictions=np.matrix(predictions)
print(test)
print(predictions)

error = mean_squared_error(test, predictions)
print('Test MSE: %.3f' % error)
```

Running Result:

```
[[27.940001 28.99     28.32     26.23     26.15     ]]
[[28.16275976 28.07093476 28.03462676 27.98376856 27.9475423 ]]
Test MSE: 1.457
```

最终，得到MSE为 1.457.

三、机器学习算法

0. 思路及数据预处理

思路：以过去三天的数据预测后一天的数据

数据预处理：

```
# 数据处理
data = pd.read_csv("./BAC.csv")#收盘价
x1 = data['BAC.Close'][0:2997].reset_index(drop=True)
x2 = data['BAC.Close'][1:2998].reset_index(drop=True)
x3 = data['BAC.Close'][2:2999].reset_index(drop=True)
y =  data['BAC.Close'][3:3000].reset_index(drop=True)

print(dataset)
dataset = pd.DataFrame({'x1':x1,'x2':x2,'x3':x3,'y':y})
dataset.to_csv('BSC_0.csv', sep='\t')
```

1. 多元线性回归

```
y = pd.read_csv("./BAC_0.csv",usecols=[4])
x = pd.read_csv("USB.csv",usecols=[1,2,3])

x_train = x[0:2997]
x_test = x[2997:]

y_train = y[0:2997]
y_test = y[2997:]

model = LinearRegression()

model.fit(x_train,y_train)
a = model.intercept_  # 截距
b = model.coef_  # 回归系数
print("最佳拟合线:截距", a, ",回归系数: ", b[0])
```

Running Result:

```
最佳拟合线:截距 [0.04325545] ,回归系数:  [0.01692409 0.00896636 0.97152243]
```

即：$y_t = 0.043 + 0.97y_{t-1} + 0.009y_{t-2} + 0.017y_{t-3}$

```
MSE = 0
y_predict = [0,0,0,0,0]

y_predict[0] = np.dot(b,x_test.loc[2997] + a)[0]
MSE = MSE + (y_test.loc[2997]-y_predict[0])**2


x_test.loc[2998]['x3'] = y_predict[0]
y_predict[1] = np.dot(b,x_test.loc[2998] + a)[0]
MSE = MSE + (y_test.loc[2998]-y_predict[1])**2


x_test.loc[2999]['x2'] = y_predict[0]
x_test.loc[2999]['x3'] = y_predict[1]
y_predict[2] = np.dot(b,x_test.loc[2999] + a)[0]
MSE = MSE + (y_test.loc[2999]-y_predict[2])**2


x_test.loc[3000]['x1'] = y_predict[0]
```

```
x_test.loc[3000]['x2'] = y_predict[1]
x_test.loc[3000]['x3'] = y_predict[2]
y_predict[3] = np.dot(b,x_test.loc[3000] + a)[0]
MSE = MSE + (y_test.loc[3000]-y_predict[3])**2


x_test.loc[3001]['x1'] = y_predict[1]
x_test.loc[3001]['x2'] = y_predict[2]
x_test.loc[3001]['x3'] = y_predict[3]
y_predict[4] = np.dot(b,x_test.loc[3001] + a)[0]
MSE = MSE + (y_test.loc[3001]-y_predict[4])**2

print('pre',y_predict)
print('Test MSE: %.3f' % MSE)
```

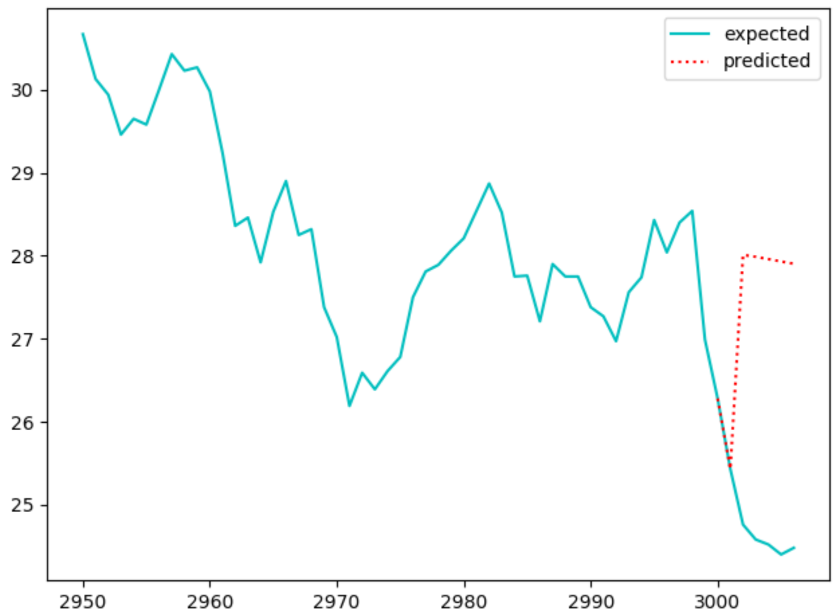
Running Result:

```
pre [28.009021260183037, 27.987104495012314, 27.958933713529373,
27.930844350038015, 27.902931392732867]
Test MSE: 10.238
```

```
y.loc[3002] = y_predict[0]
y.loc[3003] = y_predict[1]
y.loc[3004] = y_predict[2]
y.loc[3005] = y_predict[3]
y.loc[3006] = y_predict[4]

plt.plot(y[2800:3000],'c-',label="expected")
plt.plot(y[3000:],'r-',label="predicted")
plt.legend()
plt.show()
```

为了更好地看清趋势，取最后50个数据和预测的5个数据进行画图：



可以看出，前几天的预测效果较好，越到后面误差越大，最终得到MSE为**10.238**。

2. MLP

```
from sklearn.neural_network import MLPRegressor
from sklearn.preprocessing import StandardScaler
import pandas as pd

y = pd.read_csv("./BAC_0.csv",usecols=[4])
x = pd.read_csv("./BAC_0.csv",usecols=[1,2,3])

x_train = x[0:2997]
x_test = x[2997:]
y_train = y[0:2997]
y_test = y[2997:]

X = x_train
y = y_train

scaler = StandardScaler() # 标准化转换
scaler.fit(X)  # 训练标准化对象
X = scaler.transform(X)   # 转换数据集
```

使用两个隐藏层（第一个隐藏层3个神经元，第二个隐藏层1个神经元）进行训练。

```
#（多层感知器对特征的缩放是敏感的，所以需要归一化你的数据。 例如，将输入向量 X 的每个属性放缩到到 [0, 1] 或 [-1，+1] ，或者将其标准化使它具有 0 均值和方差 1。
# 为了得到有意义的结果，必须对测试集也应用 相同的尺度缩放。 可以使用 StandardScaler 进行标准化。）
# solver=‘sgd',  MLP的求解方法：L-BFGS 在小数据上表现较好，Adam 较为鲁棒，SGD在参数调整较优时会有最佳表现（分类效果与迭代次数）；SGD标识随机梯度下降。
# alpha:L2的参数：MLP是可以支持正则化的，默认为L2，具体参数需要调整
# hidden_layer_sizes=(3, 1) hidden层2层,第一层3个神经元，第二层1个神经元，2层隐藏层，也就有3层神经网络
clf = MLPRegressor(solver='sgd', alpha=1e-5,hidden_layer_sizes=(3,1),
random_state=1)
clf.fit(X, y)

x_0 = scaler.fit_transform(x_test.loc[2997].values.reshape(1,-1))
print('预测结果：', clf.predict(x_0) ) # 预测某个输入对象
y_predict = clf.predict(x_0)
MSE = (y_test.loc[2997]-y_predict)**2

cengindex = 0
for wi in clf.coefs_:
    cengindex += 1  # 表示底第几层神经网络。
    print('第%d层网络层:' % cengindex)
    print('权重矩阵维度:',wi.shape)
    print('系数矩阵：\n',wi)

print('pre',y_predict)
print('Test MSE: %.3f' % MSE)
```

Running Result:

```
预测结果：  [20.08827782]
第1层网络层:
权重矩阵维度: (3, 3)
系数矩阵:
 [[ 0.33392707  0.35127219 -0.48573496]
```

```
 [-0.16823619  0.04766473 -0.64943287]
 [-1.0042373   1.51687317 -0.83869717]]
第2层网络层:
权重矩阵维度: (3, 1)
系数矩阵:
 [[-1.03481188]
 [ 1.39696968]
 [-0.93003716]]
第3层网络层:
权重矩阵维度: (1, 1)
系数矩阵:
 [[4.34420929]]
pre [20.08827782]
Test MSE: 69.085
```

由运行结果可得，在**预测未来一天的情况下，MSE已经达到69**，说明预测效果不好。

3. BP 神经网络

```
import numpy as np
import matplotlib.pylab as plt
import pandas as pd
import random
random.seed(0)
def sigmoid(x):
    return 1/(1+np.exp(-x))

def BP(data_tr, data_te, maxiter=500):
    MSE=0
    # --pandas是基于numpy设计的，效率略低
    # 为提高处理效率，转换为数组
    data_tr, data_te = np.array(data_tr), np.array(data_te)

    # --隐层输入
    # -1:  代表的是隐层的阈值
    net_in = np.array([0.0, 0, -1])
    w_mid = np.random.rand(3, 4)  # 隐层权值阈值（-1x其中一个值：阈值）

    # 输出层输入
    # -1：代表输出层阈值
    out_in = np.array([0.0, 0, 0, 0, -1])
    w_out = np.random.rand(5)  # 输出层权值阈值（-1x其中一个值：阈值）
    delta_w_out = np.zeros([5])  # 存放输出层权值阈值的逆向计算误差
    delta_w_mid = np.zeros([3, 4])  # 存放因此能权值阈值的逆向计算误差
    yita = 1.75  # η:  学习速率
    Err = np.zeros([maxiter])  # 记录总体样本每迭代一次的错误率

    # 1.样本总体训练的次数
    for it in range(maxiter):
        # 衡量每一个样本的误差
        err = np.zeros([len(data_tr)])
        # 2.训练集训练一遍
        for j in range(len(data_tr)):
            net_in[:3] = data_tr[j, :3]  # 存储当前对象前两个属性值
            real = data_tr[j, 2]
            # 3.当前对象进行训练
```

```
            for i in range(4):
                out_in[i] = sigmoid(sum(net_in * w_mid[:, i]))  # 计算输出层输入
            res = sigmoid(sum(out_in * w_out))  # 获得训练结果

            err[j] = abs(real - res)

            # --先调节输出层的权值与阈值
            delta_w_out = yita * res * (1 - res) * (real - res) * out_in  # 权值
调整
            delta_w_out[4] = -yita * res * (1 - res) * (real - res)  # 阈值调整
            w_out = w_out + delta_w_out

            # --隐层权值和阈值的调节
            for i in range(4):
                # 权值调整
                delta_w_mid[:, i] = yita * out_in[i] * (1 - out_in[i]) *
w_out[i] * res * (1 - res) * (real - res) * net_in
                # 阈值调整
                delta_w_mid[2, i] = -yita * out_in[i] * (1 - out_in[i]) *
w_out[i] * res * (1 - res) * (real - res)
            w_mid = w_mid + delta_w_mid
        Err[it] = err.mean()
    plt.plot(Err)
    plt.show()

    # 存储预测误差
    err_te = np.zeros([100])

    # 预测样本5个
    for j in range(5):
        net_in[:3] = data_te[j, :3]  # 存储数据
        real = data_te[j, 2]  # 真实结果

        # net_in和w_mid的相乘过程
        for i in range(4):
            # 输入层到隐层的传输过程
            out_in[i] = sigmoid(sum(net_in * w_mid[:, i]))
        # res = sigmoid(sum(out_in * w_out))  # 网络预测结果输出
        res = (sum(out_in * w_out))
        res0 = res/4
        err_te[j] = abs(real - res)  # 预测误差
        print('res:', res, ' real:', real,res0)
        MSE = MSE + (res-real)**2
        print('Test MSE: %.3f' % MSE)

    plt.plot(err_te)
    plt.show()


if "__main__" == __name__:
    # 1.读取样本
    data = pd.read_csv("./BAC_0.csv")
    data_tr = data[:2997]
    data_te = data[2997:3002]
    BP(data_tr, data_te, maxiter=500)
```

Running Result:

```
real: 28.43 24.413609869649484
Test MSE: 16.05124096
```

由运行结果可得，在**预测未来一天的情况下，MSE已经达到16**，说明预测效果不好。

四、思考

综上可知，**State-Space模型的预测效果要比机器学习算法预测效果好。**

本人认为可能的原因为：

- State-Space 模型在时间序列分析方面已经经过了无数人的研究，所以在预测方面表现好；
- 我们的数据量不大，机器学习算法容易出现过拟合现象；
- 受股票数据本身特质的影响，股票的未来价格受最近几天的影响较大，时间越久远的数据对股票未来价格的影响不大，预测通常从最近几天数据进行。而机器学习算法模型的学习和预测是以庞大的数据集为基础的，这一点会影响对股票未来走势的预测。