State Space Model VS Machine Learning

前面我们学习了用State Space model对时间序列进行参数估计，然后对未来数据进行预测，现在我们用机器学习算法和State Space model对SP100股票数据集进行预测，对比机器学习算法和State Space model在预测股票数据方面的性能。

在此我选取的机器学习算法是**岭回归算法**和**多层感知机**。State Space Model算法是**单噪声源的常规卡尔曼滤波**，**多噪声源的常规卡尔曼滤波**，**单噪声源的Theta method**，**多噪声源的Theta method**，**单噪声源的Damped trend model**。

关于算法性能的评价方面，我们需要定义一个指标，用这个指标来定量地衡量算法在预测方面表现的好坏。此处我们不考虑算法的复杂度，而仅以预测误差作为评判标准。和前面评估State Space model的预测性能一样，此处我们用**MASE**和**MAPE**来衡量算法预测性能的好坏，MASE和MAPE越小，表示模型预测越准确。

```
# 导入第三方包
import numpy as np
# from sklearn import linear_model
# from keras.models import Sequential
# from keras.layers import Dense
import matplotlib.pyplot as plt
import pandas as pd
from scipy import optimize
```

```
# 读入SP100数据
f = open('SP100.csv')
df = pd.read_csv('SP100.csv')
del df['Unnamed: 0']
Stock_name=[]
for index, row in df.iteritems():
    # print(index) # 输出列名
    Stock_name.append(index)
```

首先我们创建一个矩阵称为MASE_metric，以预测天数为行，以不同算法为列，矩阵的a_{ij}元素表示第j个算法在第i天预测的MASE；对于MAPE_metric同理。这样我们可以明显的看出不同的算法在不同的窗口下表现好坏。初始化这两个矩阵为全0矩阵。

```
MASE_metric= np.zeros((5, 7))
MAPE_metric= np.zeros((5, 7))
```

下面我们首先分别用单噪声源的常规卡尔曼滤波，多噪声源的常规卡尔曼滤波，单噪声源的Theta method，多噪声源的Theta method，单噪声源的Damped trend model对SP100数据进行预测。

自定义5个State Space model函数，输入已知的时间序列y和待预测天数h，进行时间序列的参数估计，并预测未来h天的数据。

```python
def ForecastKFMS(y,h):

    # 多噪声源状态空间模型的预测

    n=len(y)
    a=np.zeros(n)
    p=np.zeros(n)
    a[0]=y[0]
    p[0]=10000
    k=np.zeros(n)
    v=np.zeros(n)
    def function(mypa):
        q=abs(mypa[0])
        co=abs(mypa[1])
        w=1-np.exp(-abs(mypa[2]))
        z=1
        likelihood=0
        sigmae=0
        for t in range(1,n):
            k[t]=(z*w*p[t-1])/(z**2*p[t-1]+1)
            p[t]=w**2*p[t-1]-w*z*k[t]*p[t-1]+q
            v[t]=y[t]-z*a[t-1]
            a[t]=co+w*a[t-1]+k[t]*v[t]
            sigmae=sigmae+(v[t]**2/(z**2*p[t-1]+1))
            likelihood=likelihood+.5*np.log(2*np.pi)+.5+.5*np.log(z**2*p[t-1]+1)
        likelihood+=.5*n*np.log(sigmae/n)
        return likelihood
    res=optimize.minimize(function,[.2,1,1])
    v=np.zeros(n)
    z=1
    q=abs(res.x[0])
    co=abs(res.x[1])
    w=1-np.exp(-abs(res.x[2]))
    sigmae=0
    for t in range(1,n):
        k[t]=(z*w*p[t-1])/(z**2*p[t-1]+1)
        p[t]=w**2*p[t-1]-w*z*k[t]*p[t-1]+q
        v[t]=y[t]-z*a[t-1]
        a[t]=co+w*a[t-1]+k[t]*v[t]
        sigmae=sigmae+(v[t]**2/(z**2*p[t-1]+1))
    Forecasts=np.zeros(h)
    Forecasts[0]=a[len(y)-1]
    for i in range(1,h):
        Forecasts[i]=co+w*Forecasts[i-1]
    return Forecasts
```

```python
def ForecastKFSS(y,h):

    # 单噪声源状态空间模型的预测

    n=len(y)
    state=np.zeros(n)
    state[0]=y[0]
    v=np.zeros(n)
    def logLikConc(mypa):
        co=abs(mypa[2])
        gamma=abs(mypa[1])
        w=1-np.exp(-abs(mypa[0]))
        for t in range(1,n):
            v[t]=y[t]-state[t-1]
            state[t]=co+w*state[t-1]+gamma*v[t]
        return np.sum(v[1:(n-1)]**2)
    res=optimize.minimize(logLikConc,[1,.2,1])
#     v=np.zeros(n)
    co=abs(res.x[2])
    gamma=abs(res.x[1])
    w=1-np.exp(-abs(res.x[0]))
    for t in range(1,n):
        v[t]=y[t]-state[t-1]
        state[t]=co+w*state[t-1]+gamma*v[t]
    Forecasts=np.zeros(h)
    Forecasts[0]=state[len(y)-1]
    for i in range(1,h):
        Forecasts[i]=co+w*Forecasts[i-1]
    return Forecasts
```

```python
def ForecastThetaSS(y,h):

    # 单噪声Theta method模型的预测

    n=len(y)
    state=np.zeros(n)
    state[0]=y[0]
    v=np.zeros(n)
    def logLikConc(mypa):
        co=abs(mypa[1])
        gamma=abs(mypa[0])
        w=1
        for t in range(1,n):
            v[t]=y[t]-state[t-1]
            state[t]=co+w*state[t-1]+gamma*v[t]
        return np.sum(v[1:(n-1)]**2)
    res=optimize.minimize(logLikConc,[.3,1])
#     v=np.zeros(n)
    co=abs(res.x[1])
    gamma=abs(res.x[0])
    w=1
    for t in range(1,n):
        v[t]=y[t]-state[t-1]
        state[t]=co+w*state[t-1]+gamma*v[t]
    Forecasts=np.zeros(h)
    Forecasts[0]=state[n-1]
    for i in range(1,h):
        Forecasts[i]=co+w*Forecasts[i-1]
    return Forecasts
```

```python
def ForecastThetaMS(y,h):

    # 多噪声源Theta method模型的预测

    n=len(y)
    a=np.zeros(n)
    p=np.zeros(n)
    a[0]=y[0]
    p[0]=10000
    k=np.zeros(n)
    v=np.zeros(n)
    def funTheta(mypa):
        q=abs(mypa[0])
        co=abs(mypa[1])
        w=1
        z=1
        likelihood=0
        sigmae=0
        for t in range(1,n):
            k[t]=(z*w*p[t-1])/(z**2*p[t-1]+1)
            p[t]=w**2*p[t-1]-w*z*k[t]*p[t-1]+q
            v[t]=y[t]-z*a[t-1]
            a[t]=co+w*a[t-1]+k[t]*v[t]
            sigmae=sigmae+(v[t]**2/(z**2*p[t-1]+1))
            likelihood=likelihood+.5*np.log(2*np.pi)+.5+.5*np.log(z**2*p[t-1]+1)
        likelihood+=.5*n*np.log(sigmae/n)
        return likelihood
    res=optimize.minimize(funTheta,[.3,1])
#     v=np.zeros(n)
    z=1
    q=abs(res.x[0])
    co=abs(res.x[1])
    w=1
    for t in range(1,n):
        k[t]=(z*w*p[t-1])/(z**2*p[t-1]+1)
        p[t]=w**2*p[t-1]-w*z*k[t]*p[t-1]+q
        v[t]=y[t]-z*a[t-1]
        a[t]=co+w*a[t-1]+k[t]*v[t]
    Forecasts=np.zeros(h)
    Forecasts[0]=a[n-1]
    for i in range(1,h):
        Forecasts[i]=co+w*Forecasts[i-1]
    return Forecasts
```

```python
def ForecastDampedSS(y,h):

    # 单噪声源的Damped trend模型

    obs=len(y)
    damped=np.zeros(obs*2).reshape(obs,2)
    damped[0][0]=y[0]
    damped[0][1]=0

    inn=np.zeros(obs).reshape(obs,1)
    def fmsoe(param):
        k1=abs(param[0])
        k2=abs(param[1])
        k3=abs(param[2])

        for t in range(1,obs):
            inn[t]=y[t]-damped[t-1][0]-k3*damped[t-1][1]
            damped[t][0] = damped[t-1][0]+k3*damped[t-1][1]+k1*inn[t]
            damped[t][1] = k3*damped[t-1][1]+k2*inn[t]
        return np.sum(inn[0:obs-1]**2)/(obs)
    res=optimize.minimize(fmsoe,list(np.random.uniform(0,1,3)))
    k1=abs(res.x[0])
    k2=abs(res.x[1])
    k3=abs(res.x[2])
    if k3>1:
        k3=1
    for t in range(1,obs):
        inn[t]=y[t]-damped[t-1][0]-k3*damped[t-1][1]
        damped[t][0] = damped[t-1][0]+k3*damped[t-1][1]+k1*inn[t]
        damped[t][1] = k3*damped[t-1][1]+k2*inn[t]
    Forecasts=np.zeros(h)
    Forecasts[0]=damped[obs-1][0]+k3*damped[obs-1][1]
    for i in range(1,h):
        Forecasts[i]=Forecasts[i-1]+damped[obs-1][1]*k3**i
    return Forecasts
```

现在我们就用上面定义好的5个函数来对股票数据进行预测。为了表示某一个算法在某一天的预测性能，我们分别取该算法在这一天对101只股票MASE的均值和MAPE的均值，存储到上面定义好的MASE_metric和MAPE_metric中。

```
for n_steps_out in range(1,6):              # 分别预测未来1, 2, 3, 4, 5天的数据
    for stock in Stock_name:       # 遍历101只股票
        # 收盘价close
        close = np.array(df[stock])
        # 前面已知的数据
        x=close[0:len(close)-n_steps_out]
        # 预测得到的数据
        closeKFMS=ForecastKFMS(close,n_steps_out)
        closeKFSS=ForecastKFSS(close,n_steps_out)
        closeThetaMS=ForecastThetaMS(close,n_steps_out)
        closeThetaSS=ForecastThetaSS(close,n_steps_out)
        closeDampedSS=ForecastDampedSS(close,n_steps_out)
        # 计算MASE和MAPE
        KFMS_MASE=np.mean(abs(close[(len(close)-n_steps_out):len(close)]-
closeKFMS))/np.mean(abs(np.diff(x)))
        KFMS_MAPE=np.mean(200*abs(close[(len(close)-n_steps_out):len(close)]-closeKFMS)/
                  (abs(closeKFMS)+abs(close[(len(close)-n_steps_out):len(close)])))
        KFSS_MASE=np.mean(abs(close[(len(close)-n_steps_out):len(close)]-
closeKFSS))/np.mean(abs(np.diff(x)))
        KFSS_MAPE=np.mean(200*abs(close[(len(close)-n_steps_out):len(close)]-closeKFSS)/
                  (abs(closeKFSS)+abs(close[(len(close)-n_steps_out):len(close)])))
        ThetaMS_MASE=np.mean(abs(close[(len(close)-n_steps_out):len(close)]-
closeThetaMS))/np.mean(abs(np.diff(x)))
        ThetaMS_MAPE=np.mean(200*abs(close[(len(close)-n_steps_out):len(close)]-closeThetaMS)/
                  (abs(closeThetaMS)+abs(close[(len(close)-n_steps_out):len(close)])))
        ThetaSS_MASE=np.mean(abs(close[(len(close)-n_steps_out):len(close)]-
closeThetaSS))/np.mean(abs(np.diff(x)))
        ThetaSS_MAPE=np.mean(200*abs(close[(len(close)-n_steps_out):len(close)]-closeThetaSS)/
                  (abs(closeThetaSS)+abs(close[(len(close)-n_steps_out):len(close)])))
        DampedSS_MASE=np.mean(abs(close[(len(close)-n_steps_out):len(close)]-
closeDampedSS))/np.mean(abs(np.diff(x)))
        DampedSS_MAPE=np.mean(200*abs(close[(len(close)-n_steps_out):len(close)]-closeDampedSS)/
                  (abs(closeDampedSS)+abs(close[(len(close)-n_steps_out):len(close)])))
        MASE_metric[n_steps_out-1][0]+=KFMS_MASE
        MASE_metric[n_steps_out-1][1]+=KFSS_MASE
        MASE_metric[n_steps_out-1][2]+=ThetaMS_MASE
        MASE_metric[n_steps_out-1][3]+=ThetaSS_MASE
        MASE_metric[n_steps_out-1][4]+=DampedSS_MASE

        MAPE_metric[n_steps_out-1][0]+=KFMS_MAPE
        MAPE_metric[n_steps_out-1][1]+=KFSS_MAPE
        MAPE_metric[n_steps_out-1][2]+=ThetaMS_MAPE
        MAPE_metric[n_steps_out-1][3]+=ThetaSS_MAPE
        MAPE_metric[n_steps_out-1][4]+=DampedSS_MAPE
    for i in range(5):              # 5种算法，每个都取101只股票的均值
        MASE_metric[n_steps_out-1][i]=MASE_metric[n_steps_out-1][i]/101
        MAPE_metric[n_steps_out-1][i]=MAPE_metric[n_steps_out-1][i]/101
```

在State Space Model完成预测之后，下面我们使用机器学习算法对上述股票数据进行预测，我选取了**岭回归**和**多层感知机算法**。

使用机器学习算法对时间序列进行预测的难点在于数据的处理，由于机器学习解决回归问题的原理是输入特征向量，输出回归向量，但是时间序列的数据是按照时间顺序排列的一维数组，因此我们必须把时间序列数据改造成特征向量和回归向量的形式。例如：以前5天的数据为特征向量输入模型当中，以接下来3天的数据为回归向量。下面自定义一个函数按照这种方式将数据进行切分。

```
def split_sequence(sequence, n_steps_in, n_steps_out):
    X, y = list(), list()
    for i in range(len(sequence)):
        # 找到结束的位置
        end_ix = i + n_steps_in
        out_end_ix = end_ix + n_steps_out
        # 判断是否超出列表长度
        if out_end_ix > len(sequence):
            break
        # 将输入、输出集成
        seq_x, seq_y = sequence[i:end_ix], sequence[end_ix:out_end_ix]
        X.append(seq_x)
        y.append(seq_y)
    return np.array(X), np.array(y)
```

下面我们使用多层感知机和岭回归对时间序列进行预测。同样是将MASE和MAPE存储到评价矩阵中。

```
for n_steps_out in range(1,6):
    for stock in Stock_name:
        close = np.array(df[stock])
        # 选定预测的时间长度
        n_steps_in = 5
        train_set=close[0:len(close)-n_steps_out]
        test_set=close[len(close)-n_steps_out:len(close)]

        # 划分训练集和测试集
        X_train, y_train = split_sequence(train_set, n_steps_in, n_steps_out)
        X_test=close[len(close)-(n_steps_in+n_steps_out):len(close)-n_steps_out].reshape((1,
n_steps_in))
        y_test=test_set
        # MLP模型训练
        model = Sequential()
        model.add(Dense(100, activation='relu', input_dim=n_steps_in))
        model.add(Dense(n_steps_out))
        model.compile(optimizer='adam', loss='mse')
        model.fit(X_train, y_train, epochs=2000, verbose=0)
        # MLP预测
        MLPprediction=model.predict(X_test.reshape((1, n_steps_in)),verbose=0)
        # 岭回归模型训练
        reg = linear_model.Ridge(alpha=.5)
        reg.fit(X_train, y_train)
        # 岭回归预测
        REGprediction=reg.predict(X_test)
        # 计算MASE和MAPE
        MLP_MASE=np.mean(abs(y_test-MLPprediction[0]))/np.mean(abs(np.diff(close[0:len(close)-
n_steps_out])))
        REG_MASE=np.mean(abs(y_test-REGprediction[0]))/np.mean(abs(np.diff(close[0:len(close)-
n_steps_out])))
        MLP_MAPE=np.mean(200*abs(close[(len(close)-n_steps_out):len(close)]-MLPprediction)/
                 (abs(MLPprediction)+abs(close[(len(close)-n_steps_out):len(close)])))
        REG_MAPE=np.mean(200*abs(close[(len(close)-n_steps_out):len(close)]-REGprediction)/
                 (abs(REGprediction)+abs(close[(len(close)-n_steps_out):len(close)])))

        MASE_metric[n_steps_out-1][5]+=MLP_MASE
        MASE_metric[n_steps_out-1][6]+=REG_MASE
        MAPE_metric[n_steps_out-1][5]+=MLP_MAPE
        MAPE_metric[n_steps_out-1][6]+=MLP_MAPE

        for i in range(5,7):              # 2种算法，每个都取101只股票的均值
            MASE_metric[n_steps_out-1][i]=MASE_metric[n_steps_out-1][i]/101
            MAPE_metric[n_steps_out-1][i]=MAPE_metric[n_steps_out-1][i]/101
```

由于数据比较庞大，模型比较复杂，运行花费时间较长，因此我把最终得到的MASE_metric和MAPE_metric保存到csv文件中，下面加载文件查看结果：

```
# np.savetxt("D:/anaconda3/envs/myTensorflow/Time Series/MASE.csv", MASE_metric, delimiter=",")
# np.savetxt("D:/anaconda3/envs/myTensorflow/Time Series/MAPE.csv", MAPE_metric, delimiter=",")
MASE=pd.read_csv("MASE.csv")

MAPE=pd.read_csv("MAPE.csv")
```

```
MASE
```

day	KFMS	KFSS	ThetaMS	ThetaSS	DampedSS	MLP	REG
1	0.172240	0.210532	0.114968	0.148572	NaN	0.010898	0.013611
2	0.578169	0.578888	0.553786	0.551100	NaN	0.008075	0.010050
3	0.817624	0.801726	0.807575	0.791111	NaN	0.019160	0.019542
4	1.133313	1.107870	1.134893	1.111407	NaN	0.009954	0.009468
5	1.190456	1.160434	1.200189	1.173436	NaN	0.008424	0.008148

```
MAPE
```

day	KFMS	KFSS	ThetaMS	ThetaSS	DampedSS	MLP	REG
1	0.239908	0.290014	0.155116	0.214517	NaN	0.023682	0.023682
2	0.93334	0.937573	0.90018	0.901387	NaN	0.017483	0.017483
3	1.292869	1.276735	1.279704	1.258457	NaN	0.042663	0.042663
4	1.829393	1.802041	1.83034	1.798323	NaN	0.02153	0.02153
5	1.897111	1.863416	1.907733	1.868637	NaN	0.018386	0.018386

1.从结果中可以明显看出，机器学习的两个算法不管预测未来多少天的数据，都要明显的好于State Space Model。

2.对比机器学习的两个算法，可以得知：

（1）从MASE角度来看，**多层感知机**略好于**岭回归**；从MAPE角度来看，二者几乎完全一样。因此总体来看，**多层感知机**是7个算法中表现最好的；

（2）从预测天数来看，并非预测时间越短越精确。因为股票是一个波动数据，相比于预测某一天的数值，这两个模型都更擅长预测一段时间内的趋势。

3.对比State Space Model的5个算法，可以得知：

（1）Damped trend由于模型比较复杂，对于股票这种大型数据，无法用python的optimize优化函数进行参数估计；

（2）在其他4种模型种，短期预测最好的是**多噪声源的Theta method**，长期预测最好的是**单噪声源的Theta method**；

（3）综合来看，在State Space Model的5个算法中，表现最好的是**单噪声源的Theta method**。