订单量预测 OPOP 模型：
基于季节性时间序列分析模型 ARIMA-S 的
权重门混合预测模型

深信服 质量与运营 数据分析 原东升

2022.1.13 (1.0)

1. 前言

1.1 背景简介

经过一个多月的工作和观察，该报告发现供应链的数字化是非常有价值的，能够有效提高团队的工作效率，优化用户体验，减少不良率，创造更多的利润。但是，供应链整体的效率的提升必须解决瓶颈和短板。目前在数字化进程中遇到的问题和挑战有很多，亟待解决的问题包括数据的积累和规范化，不同平台之间数据的拉通，预测指标不准确等。数据的基础建设是一个长期的系统工程，无法通过个人短期的努力迅速解决。但是，预测模型是当前就可以优化和实施的。目前供应链对主要产品订单量的预测已经形成了一套模型和理论，但是缺乏专门的整理和分析，没有系统评估其有效性，也没有人提出优化方法。在预测工作中，预测数据是否准确，决定供应链整体的工作效率。如果预测结果不准确，交期会被拉长，交付的压力会很大，客户的负面反馈也会增加。但是当前的预测主要还是依赖有经验的工程师来做，预测的准确性取决于工程师的能力和经验，一旦负责人员更换，预测的效果就会有波动，同时，以往积累的经验和先验知识无法直接传递和积累，这些都会导致预测数据波动和不准确。

所以，该报告将以量化分析的形式对目前执行的订单数预测方法进行评估和优化，并尝试设计和构建更合理的订单预测模型。新的模型应该具有几个优点：既能充分反映数据自身的增长趋势，也会综合考虑外源性指标，如市场预期，供应商的供货稳定性评估等，同时还能和工程师的主观判断形成“争论”，在不断的预测工作中评估自身能力和工程师主观判断的合理性，并逐步降低对人的主观性的依赖。限于本文作者的能力和视角，模型可能是不完善的，但是该文的主要目的是为数字驱动决策提供一个具体上手的思路。同时，人机交互对抗学习模型业界少有，该文能够为构建实际的工程项目提供一种可能有效的构想。该方法不仅仅可以用于订单数量的预测，还可以用于不良率的因子分析，供应商管理等等领域。

模型的落地需要良好的工程实现。基于当前工作实际，该报告尝试构建封装的计算模块 OPOP（Operational data-driven Order Prediction system），源码在附录中，后续会考虑脱敏上传至 git，后续工作中，工程师可以根据本文附录提供的 API 接口编写新的应用或者创建数据看板。

1.2 研究问题

该报告主要围绕以下问题进行研究和讨论：

研究问题一：当前的预测模型是否合理？需要什么样的改进？

研究问题二：订单需求是否存在时间上的规律？使用环比和固定增长率来预测是否合理？该规律能否用于每个型号的产品的预测？使用自回归模型能否准确预测？

研究问题三： 如何将数据自相关趋势和工程师的主观判断结合起来，形成一个系统性的预测模型？

2. 已有预测模型评估

经过学习和了解，当前我们团队的工程师进行预测主要有两种思路，一种是基于变量本身的自源性数据进行线性的预测。如方案（1）：

2.1 方案（1）：

(a) 判断每个月的订单数在全年的占比，(b) 再用一个年增量率预测下一年的整体增长，(c) 用每个月的订单数占比乘以全年总量来计算下一年各个月的订单数。

该计算模型实际进行了两个处理：计算各个月份在全年的占比，尝试消除季节变动的波动；用增长率计算下一年全年数量，消除同比增长。

该模型的数学表达为：

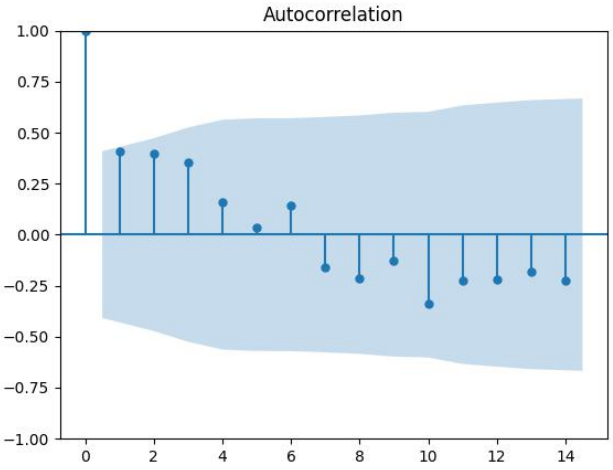
假设 t 年的全年产量为 w，1-12 月份的每月订单占比为p_i，下标 i 为第 i 个月，α 为预定的年增长率。则 $t+1$ 年的第 j 月预测数据f_j为：

$$f_j = (1+\alpha)w \cdot p_i \qquad (1)$$

可以看出，该方案是基于订单数据的自相关性，构建了一个简易的线性分析模型。该方案是否有效需要判断订单数据是否符合时间序列分析模型的重要前提：经过调整后的数据是否是**平稳（Stationary）**的。平稳的定义是，数据的均值和方差保持相对稳定，数据呈现正态分布，这是进行回归建模的前提。但是实际情况是，订单数量样本较小时，极容易收到各种外源因素的影响，如供应商交付周期的变化，市场整体供需情况，甚至是政治因素，疫情因素等等，数据有可能是不平稳的。所以该模型的有效性有待考证。

根据时序分析平稳性检测的普遍方法，先对数据进行季节差分消除季节波动，再进行一差分消除整体增长造成的波动，最后使用自相关系数 ACF 来量化数据的自相关性。如果数据是平稳的，ACF 指标应当在自相关分裂阶数（一种检测方法中的参数）增长时迅速减少至 0 左右，否则无法证明处理后的数据时平稳的。平稳性检测采用卡方检验，取单端拒绝域为 1%，5%，10%。如果检验统计量低于评估节点，则拒绝原假设，并认为数据是非平稳的。

在进行数据处理之前，先对未处理的原数据进行分析，自相关性系数 ACF 图为：



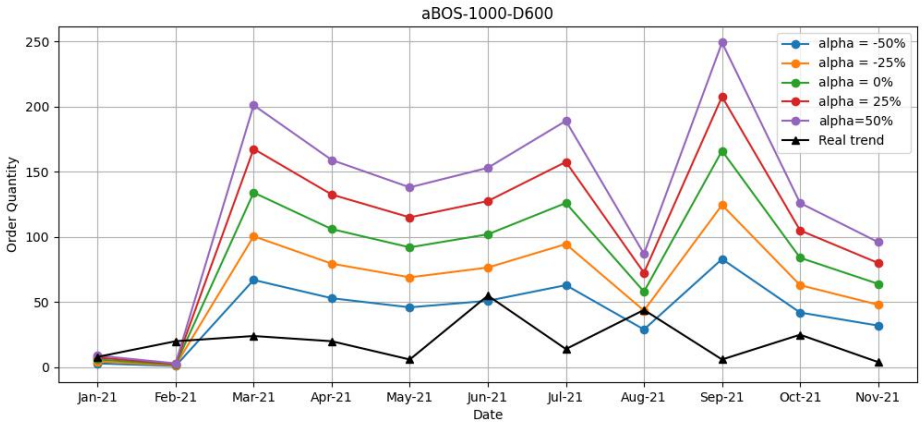
根据 ACF 的特性，如果数据是平稳的，在某一个阶数后，会迅速下降到接近 0 的水平，但是上图中的水平一直无法收敛到一个较低的水平，并且无法通过假设检验。对模型进行 2 次差分之后，依然无法通过假设检验，说明未经处理的原始数据是不平稳的，直接差分后建模也是不合理的。（季节性变动明显）

根据观察，订单数据具有明显的季节性，为了考虑季节对订单数量的影响，进行季节差分。但是订单数据样本量太少，所以使用 2018 年 9 月-2021 年 12 月的发货数据作为代替，进行季节差分（12 阶差分）后，再进行一次差分来消除增长或者下降的影响，最终数据表现良好，能够通过假设检验。

DF 卡方检验结果	
Test Statistic	-1.052977e+01
p-value	9.216151e-19
Number of Observations Used	2.500000e+01
Critical Value (1%)	-3.723863e+00
Critical Value (5%)	-2.986489e+00
Critical Value (10%)	-2.632800e+00

根据一般的假设检验原则，假设数据本身不平稳，当检验统计量低于 p 值时，拒绝原假设，即数据本身是平稳的。平稳，意味着模型能够消除数据的波动，即对数据有着较好的解释性。上表中，检验统计量远远小于 p 值，意味数据时平稳的。换句话说，产品 aBOS-1000-D600 出货数据波动的主要原因就是季节性变动和整体增长率。在经过季节差分和一次差分后，进行回归建模就能够有效消除波动，解释历史数据，并得到较好的预测结果。

但是，方案（1）的模型在实际数据中表现并不好。以服务器产品 aBOS-1000-D600 （已退市）2020 年 1 月-2021 年 12 月的出货数据为例，以 2020 年的数据为基础，尝试不同的增长率α预测 2021 年的表现，并和实际数据进行对比：



上图中，黑色线为 21 年实际订单数据，其余线为不同增长率下该模型的预测结果。

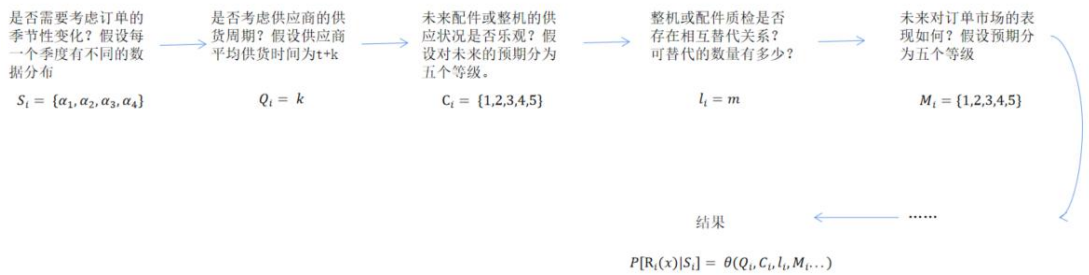
由图可见，增长率α从-50%到+50%，预测数据始终无法和实际数据相匹配，所以即使是准确预知未来一年市场的整体增长率，这种方法的预测结果始终是不理想的。2021 年是该产品下架的前一年，销量整体下降，和往年相比增长率的变化是非常规的，这样的变化是无法用固定的增长率捕捉到的。但是即使准确知悉最优的调整率α，和实际偏差也是比较大的。造成这样的结果主要是因为方案（1）的模型过于简单，仅针对上一年度数据进行建模，数据样本缺乏多样性；整体增长缺乏回归预测过程，凭主观预测，缺乏客观性。

另外，还有的工程师采用的是非线性的预测方式，如方案（2）：

2.2 方案（2）：

该方案以数据的线性增长为基础，并根据工程师观测到的市场变化，对供应商的供货能力评估，配件的可替代性等因素进行一系列的调整，最终得出一个综合性的结果。该模型的数学描述如下：

同样假设 t 年的全年产量为 w，1-12 月份的每月订单占比为p_i，下标 i 为第 i 个月。则 $t+1$ 年的第 j 月预测数据$f_j^{(2)}$为：



当决策者的判断较为准确时这种相对复杂的预测方法能够比较准确地预判一些市场上出现的黑天鹅事件，要比线性预测方法更有效一些。但是一旦决策者判断不准确，就会和实际情

况偏差很大。换句话说，该方案的预测效果缺少下限保证，模型表现很大程度上依赖于决策者的研判能力。另外，预测结果和实际结果之间缺乏对比和修正，积累的经验没能有效的保存下来，下次预测可能会犯同样的错误。

总之，当前的预测方法各有一些优势和问题。自相关模型缺乏利用外部信息进行调整的能力，而工程师自己进行研判又缺乏模型来参考和兜底，无法表达数据的客观规律。如果能够结合两种方案构建一个系统性的预测模型，并用工程实践来将方案落地成可以实施的构件，将会优化当前的预测模式，同时也是数据驱动决策的重要探索

3. 基于季节性时间序列分析模型 ARIMA-S 的权重门混合预测模型 OPOP

3.1 整体设计

结合上文的分析，主要的思路是，将数据的客观表达和外部因素，工程师的主观判断结合起来。同时，这种结合不能是机械的加总，应当是有机结合，模型应当在不断的预测过程中积累经验，学习如何判断哪些人工的干预是有效的，那些是不合理的。要构建这样的有机系统，深度学习是最好的选择，但是大型的深度学习模型需要海量数据和多轮的迭代才能形成有效的预测性能，并且深度学习模型缺乏解释性，是黑盒模型，决策者无法通过主观意志干预预测结果，也无法从模型的自身学习中获得有用的结构信息。（如数据预测关系式中各个参数的值等）但是深度学习中的理念和方法是可以借鉴的。同时，数据的自源性变化是重要的先验知识（模型在学习之前可以率先得到的有用信息），如果能够以这个变化信息作为基础，后续学习过程中不断优化，将大幅提升模型收敛速度（收敛是指参数不再大幅变化，模型整体趋于稳定，说明模型已经优化到了一个成熟阶段），并为预测结果提供整体变化预测的下限。

基于这样的思路，可以设计这样的计算模型：

其中，*w1，w2，Alpha，Beta* 为参数，需要模型在不断的预测中进行学习，找到最优的解。*x1,x2* 为两个外源性指标，随着今后预测工作的细化，指标数量可以不断增加。

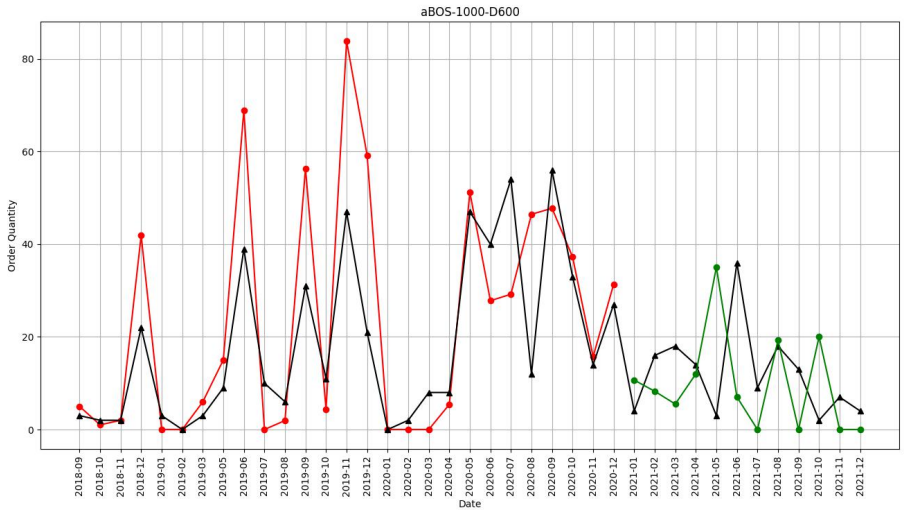
3.2 各个模块详细介绍：

3.2.1 时间序列分析模型 S(t)

构建该模型的主要挑战是，需要确定不同参数的最优值。一个完整的乘积季节 ARIMA 模型需要确定的参数有：

季节周期，一般是 12；差分次数，可以通过差分后是否平稳来得到最小差分次数；自回归的阶数 p,和移动回归的阶数 q。一般对数处理后,季节时间序列模型的参数一般都在 3 以下，所以可以通过网格搜索，或者就是一个一个找，找到基于现有数据的前提下，最有的参数组合。在一般的时序分析模型中采用 AIC 和 BIC 方法（两种衡量误差的方法），但没有考虑结构风险最小（结构风险最小会考虑整体表现，包括训练集数据表达能力，泛化能力等），可能会导致过拟合。所以本报告把数据分为训练集和测试集，通过算法（这里用 Grid Search，后期优化方向可以考虑贝叶斯优化）搜索在测试集中表现更有的参数组合。

经过参数的选择和模型的构建，初步得到的结果为：



选用的参数组合为：

参数	最优组合包
P order	3
q order	1
d order	2

图中红色部分为训练集的拟合结果，绿色部分为测试集的训练结果。从结果上看，综合 MSE 误差平均在 5-6 之间，相对与原本的方案（1）有较大提高。但是，对于突然的非正常波动预测出现了滞后性：从往年来看，五月份六月份该型号一般会出现一个交货的高峰期，但是在 2021 年，该交货高峰在六月才开始出现，而模型的结果显然过早估计这样的峰值。该问题体现了自相关模型的局限性。除此之外，该模型还经常出现预测结果是负数的情况（上图中的而结果是取非负数后的结果）。不过该模型优势明显，能够判断周期性变化，所依赖的数据较少，参数的优化较快，能够迅速判断出数据的自源性变化，如果能够作为模型的先验信息，再加以调整，就会得到不错的预测效果。

3.2.2 神经门的使用

自回归模型缺乏对外源性数据，如对市场的预期，对供应商未来生产能力的评估等的理解和吸收，并且没有办法吸纳工程师主观意见来得到更好的预测结果，所以考虑对这个模型进行改进和优化。

上文分析了该方案预测能力差的主要原因，即无法结合工程师的判断，市场预期，供应商评估等信息，单纯依赖数据自身自回归的结果。但是如果使用深度学习等复杂模型，模型的解释性会下降，因为参数众多，也无法从模型当中得到有效的解释性信息，并且模型参数的估计工作复杂，耗费时间，数据量需求也很大。所以一个折中的想法是，构建一个精心调教的时间序列分析模型，为整体模型提供一个先验知识，模型应该在此基础上，和工程师的意见等外源信息，进行对比更正，修正时间序列分析模型的结果。

对于具体的模型，可以借鉴了 CNN 神经网络中一种特殊的结构：一种神经门结构（gate），这种神经门能够学习到不同渠道的信息所应该采纳的比重，结合自身掌握的决策能力，得到一个综合的评估或者是预测的结果。结合服务器数据的具体实际，使用完整的神经元可能难以收敛，特别是激活函数结构（将线性输入转换为非线性输出来增加搜索空间）下，可能会加剧梯度消失的问题（即由于激活函数的特性，学习到的系数信息会很小，并且多层结构，在传递中小数乘以小数会更小，最终在抵达需要调整的参数时，变得无限接近于 0），需要多次学习和迭代才能构建有效的模型，但是迭代次数过多又会造成过拟合的情况。所以，该模型仅借鉴“门”这个构件的基本结构，用一个参数来决定某一个渠道信心的采纳程度，并用优化方法来优化这个参数，这将大大降低学习难度。模型的整体结构为：

$$A(t) = \alpha[S(t) + (w_1x_1 + w_2x_2)] + \beta H(t) + \varepsilon_1$$
$$\beta = 1 - \alpha$$

数据的特征选择为：

信息维度	评估值集
工程师主观判断 H(t)	R^+
供应商交货效率 x1	{1,2,3,...,9}
市场预期 x2	{1,2,3,...,9}

数据使用：

日期	x1	x2	H(t)	S(t)
18-Sep	0	0	4	3
18-Oct	0	0	3	2
18-Nov	0	0	3	2
18-Dec	0	0	23	22
19-Jan	0	0	4	3
19-Feb	0	0	1	0
19-Mar	0	0	4	3
19-May	0	0	10	9

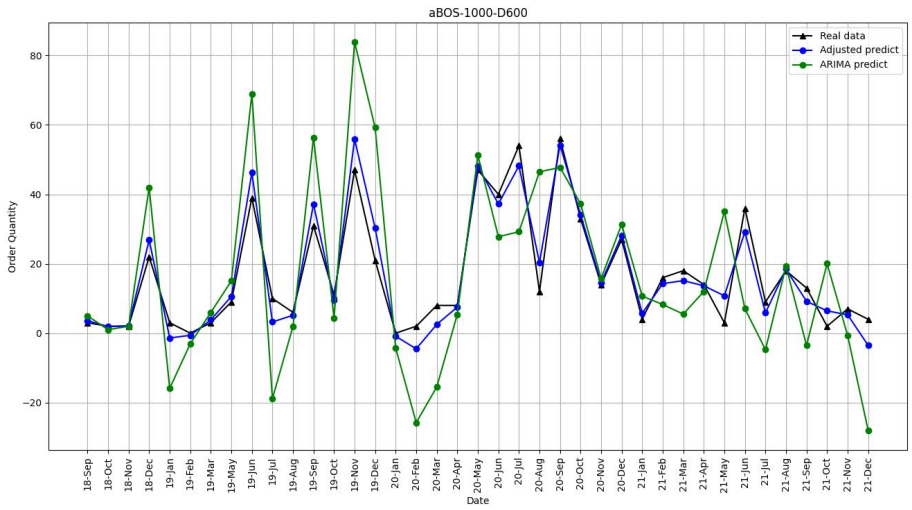
工程实现：

该模型的参数优化采用梯度下降优化方法，计算框架是基于 TensorFlow 的张量计算框架，核心代码简单易懂，维护较为容易。核心 API 包括：

```
Opop(train_set)
    -model_train()
    -predict(test_set, plot_result)
    -generate_mode_sheet(path)
    -ARIMA_train(training_set_rate, plot_result)
    -ARIMA_predict(step)
```

使用时，只需要用 generate_mode_sheet()方法生成一个模板表格，在模板表格中填写相应的数据就可以进行训练和实际的预测。后续将添加模型的效果评估，数据适配性的分析，模型算法的拓展等窗口，方便后续的工程师对模型进行优化。

效果评估

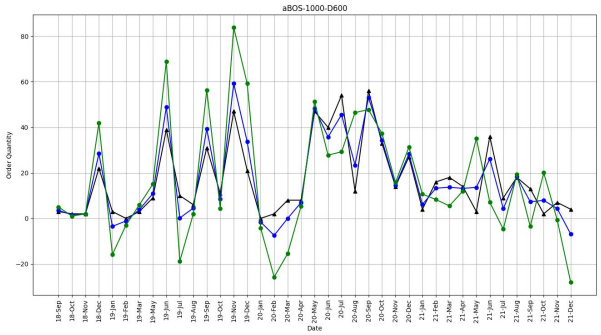


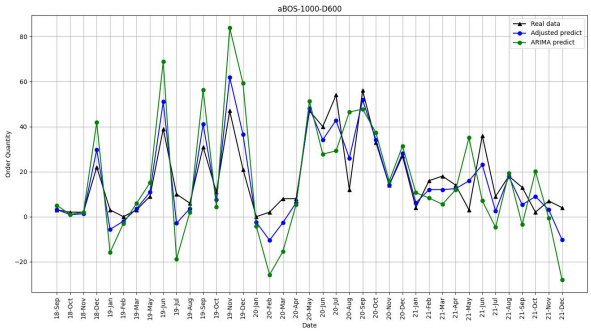
由图可见，黑色曲线是实际数据，绿色曲线代表自回归结果，蓝色曲线代表通过加权调整后的结果。通过 200 次的迭代后，系统找到了更优的人工数据的采纳比例，并通过人工干预提升了数据预测的准确性，在 2021 年的数据当中，成功地修正了五月份的非周期性变化预测不准的问题。根据 MSE 的计算，修正后的预测结果比未预测前增加 70%左右。这样的结果是符合预期的，因为实验中由很多的假设前提对实验结果进行了保证。

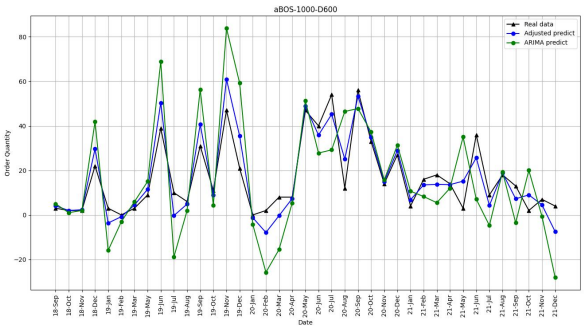
3.3 模型在放松决策人决策能力假设下的表现

在实际预测过程中，决策人的决策水平参差不齐，对模型的指导可能是正面的也可能是负面的。所以，我调整数据中拟真的决策人决策水平，具体做法是，每次把决策人的 100%正确的决策随机加减一个值，并不断扩大这个值，来评估决策人能力对模型表现的影响：

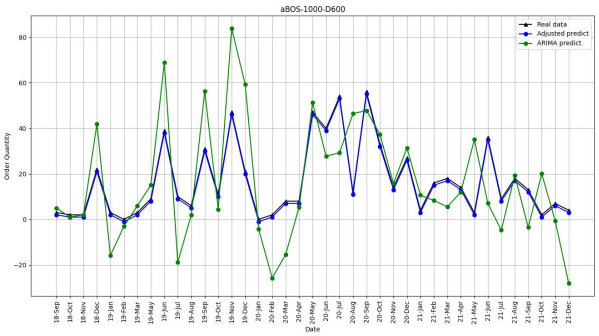
随机加减数从 1 到 3，迭代数保持在 100 次：







从图上看，变化并不明显，说明该模型对于人的判断质量敏感性并不高。具体的量化敏感系数会后续更新，思路是用 MSE 和判断偏差变化的比值来表示敏感性系数。同时，模型对迭代次数非常敏感，迭代次数增加至 200 次时，模型在训练集上的预测结果和实际结果几乎贴合，出现了严重的过拟合问题所以模型迭代次数作为超参数需要进一步优化。



同时，其他风险依然是存在的，当前的模型自身修正的能力是比较弱的，因为仅构建了神经门的部分构造，并且神经元数量少，今后可以考虑进一步增加更加复杂的神经网络结构来增强模型的自身预测能力。

4. 附录

```
import tensorflow as tf
import numpy as np
import pandas as pd
from matplotlib import pyplot as plt
from statsmodels.graphics.tsaplots import plot_acf
import statsmodels.api as sm
from tensorflow import keras
from tqdm import trange



class Opop():
    def __init__(self, train_set):
        self.train_set = self.data_processor(train_set)
        self.index = self.train_set.index
        self.n_echo = 100
        self.learning_rate = 0.1
        self.A_mod = 0.5
        self.A_results = None
        self.w = tf.Variable(initial_value=[[0.5],[0.5],[0.5]])
        self.alpha = tf.Variable(initial_value=[[0.5]])
        self.bias = 0.5



    def data_processor(self, data):
        return pd.DataFrame(data.iloc[:, 1:].values, index=data.iloc[:, 0])

    def standardization(self,data):
        mu = np.mean(data, axis=0)
        sigma = np.std(data, axis=0)
        return (data - mu) / sigma

    def model(self, echo=200):
        S = self.ARIMA_train().values
        X_set = self.train_set.iloc[:, :-2]
        bias_df = pd.DataFrame(np.ones([np.shape(X_set)[0], 1]),index=X_set.index)
        X_set = pd.concat([X_set, bias_df], axis=1)
        Y_set = self.train_set.iloc[:, -1]
        Z_set = self.train_set.iloc[:, -2]
        # shift to numpy and T
        S = np.array(S)
        S = S.reshape(S.shape[0], 1)
        Y_set = np.array(Y_set)
```

```
    Y_set = Y_set.reshape(Y_set.shape[0],1)
    Z_set = np.array(Z_set)
    Z_set = Z_set.reshape(Z_set.shape[0],1)

    # shift to tf obj
    m, n = X_set.shape
    self.w = tf.Variable(tf.random.uniform([n, 1], -1.0, 1.0), name="w")
    self.alpha = tf.Variable(tf.random.uniform([1, 1], 0.0, 1.0), name="alpha")
    S = tf.constant(S, dtype=tf.float32)
    X_set = tf.constant(X_set, dtype=tf.float32)
    Y_set = tf.constant(Y_set, dtype=tf.float32)
    Z_set = tf.constant(Z_set, dtype=tf.float32)
    optimizer = tf.keras.optimizers.Adadelta(0.1)
    for i in range(echo):
        with tf.GradientTape(persistent=True) as t:
            system_predict = self.alpha * tf.add(S, tf.matmul(X_set, self.w))
            human_predict = (1 - self.alpha) * Z_set
            y_predict = tf.add(system_predict, human_predict)
            Loss_obj = tf.keras.losses.KLDivergence()
            Loss = Loss_obj(Y_set, y_predict)
        gradients = t.gradient(target=Loss, sources=[self.w, self.alpha])
        optimizer.apply_gradients(zip(gradients, [self.w, self.alpha]))
    print(f'the pras are {[self.w,self.alpha]}')

def predict(self, data, plot_result=False):
    #data should be a df
    data = self.data_processor(data)
    a_data = data.iloc[:,-1]
    S = self.ARIMA_predict([1,len(a_data)]).values
    S2 = S
    #A = sm.tsa.statespace.SARIMAX(a_data.values, trend='n', order=pra, seasonal_order=(1, 1, 1, 12))
    #results = A.fit()
    #S = results.predict(1,len(a_data))
    #S shift to numpy and T
    S = np.array(S)
    S = S.reshape(S.shape[0],1)
    X_set = data.iloc[:, :-2]
    bias_df = pd.DataFrame(np.ones([np.shape(X_set)[0], 1]), index=X_set.index)
    X_set = pd.concat([X_set, bias_df], axis=1)
    #z set shift to numpy and transpose
    Z_set = data.iloc[:, -2].values
    Z_set = Z_set.reshape([39,1])
    #shift to tf obj
    S = tf.constant(S, dtype=tf.float32)
```

```
        X_set = tf.constant(X_set, dtype=tf.float32)
        Z_set = tf.constant(Z_set, dtype=tf.float32)
        #compute the result
        system_predict = self.alpha * tf.add(S, tf.matmul(X_set, self.w))
        human_predict = (1 - self.alpha) * Z_set
        y_predict = tf.add(system_predict, human_predict)
        #print(y_predict.numpy)
        total = pd.Series(data.iloc[:,-1].values, index=data.index)
        if plot_result == True:
            f1 = plt.figure(figsize=(16, 8))
            plt.grid()
            plt.xlabel("Date")
            plt.ylabel("Order Quantity")
            plt.title("aBOS-1000-D600")
            plt.plot(total, marker='^', color='k')
            plt.plot(y_predict, marker='o', color='b')
            plt.plot(S2, marker='o', color='g')
            plt.legend(['Real data','Adjusted predict','ARIMA predict'])
            plt.xticks(rotation=90)
            plt.show()




    def ARIMA_train(self, training_set_rate=0.7, plot_result=False):

        d1 = self.train_set.iloc[:,-1]
        train_set_index = round(training_set_rate*len(d1))
        total = pd.Series(d1.values, index=d1.index)
        train_set = pd.Series(d1.iloc[:train_set_index].values, index=list(d1.index)[:train_set_index])
        test_set = pd.Series(d1.iloc[train_set_index:].values, index=list(d1.index)[train_set_index:])
        # ARIMA-S
        new_op = Optimizer(train_set, test_set)
        best_p_list = new_op.grid_search(vrange=[0, 3])
        self.A_mod = sm.tsa.statespace.SARIMAX(train_set.values, trend='n', order=best_p_list, seasonal_order=(1,
1, 1, 12))
        self.A_results = self.A_mod.fit()
        start_po = len(train_set) + 1
        stop_po = start_po + len(test_set) - 1
        #predict = pd.Series(np.maximum(A_results.predict(start=start_po, end=stop_po, dynamic=True),
0),index=test_set.index)
        #fit_result = pd.Series(np.maximum(A_results.predict(1, start_po - 1), 0), index=train_set.index)
        overall_predict = pd.Series(np.maximum(self.A_results.predict(1, end=stop_po), 0),
```

```
                                    index=dl.index)
        if plot_result == True:
            f1 = plt.figure(figsize=(16, 8))
            plt.grid()
            plt.xlabel("Date")
            plt.ylabel("Order Quantity")
            plt.title("aBOS-1000-D600")
            plt.plot(total, marker='^', color='k')
            plt.plot(overall_predict, marker='o', color='b')
            plt.xticks(rotation=90)
            plt.show()
        return overall_predict
    def ARIMA_predict(self,step):
        #data should be a Series
        new_result = self.A_mod.fit()
        temp_res = new_result.predict(step[0], step[1])
        return pd.Series(temp_res, index=data.index)
    def generater(self,path):
        import shutil
        shutil.copy(r'\res\TrainingSetModeSheet.xlsx',path)
        shutil.copy(r'\res\Prediction.xlsx',path)
        self.root_path = path
class Optimizer():
    def __init__(self, train_set, test_set):
        self.train_set = train_set
        self.test_set = test_set
        self.cut_p = len(train_set)+1
        self.end_p = self.cut_p+len(test_set)-1

    def grid_search(self, vrange=[0, 3]):
        res_list = []
        min_v = 1000000
        best_pra = [0, 0, 0]
        v_list = []
        for p in trange(vrange[0], vrange[1] + 1):
            for q in range(vrange[0], vrange[1] + 1):
                for d in range(vrange[0], vrange[1] + 1):
                    try:
                        temp_v = self.model(pra=(p, q, d))
                    except:
                        temp_v = 10000
                    v_list.append(temp_v)
```

```
                    if temp_v < min_v:
                        min_v = temp_v
                        best_pra = [p, q, d]
        print(f'the best pra is {best_pra}')
        return best_pra

    def model(self, pra=(1, 1, 1)):
        mod = sm.tsa.statespace.SARIMAX(self.train_set.values, trend='n', order=pra, seasonal_order=(1, 1, 1,
12))
        results = mod.fit()
        predict = results.predict(start=self.cut_p, end=self.end_p, dynamic=True)
        MSE = np.sqrt((predict - self.test_set.iloc[0]) ** 2)
        return MSE.mean()
if __name__=="__main__":
    data = pd.read_csv(r'C:\Users\SXF-Admin\Desktop\分析报告\res\example2.csv',encoding='ANSI')
    model1 = Opop(data)
    model1.model()
    model1.predict(data, plot_result=True)
```