工业大数据比赛—钢卷仓储吞吐趋势预测
技术文档

队员：栾睿琦 阮琳雄

成绩：第九名

所属单位：树根互联 AI Lab

日期：2019 年 1 月 6 日

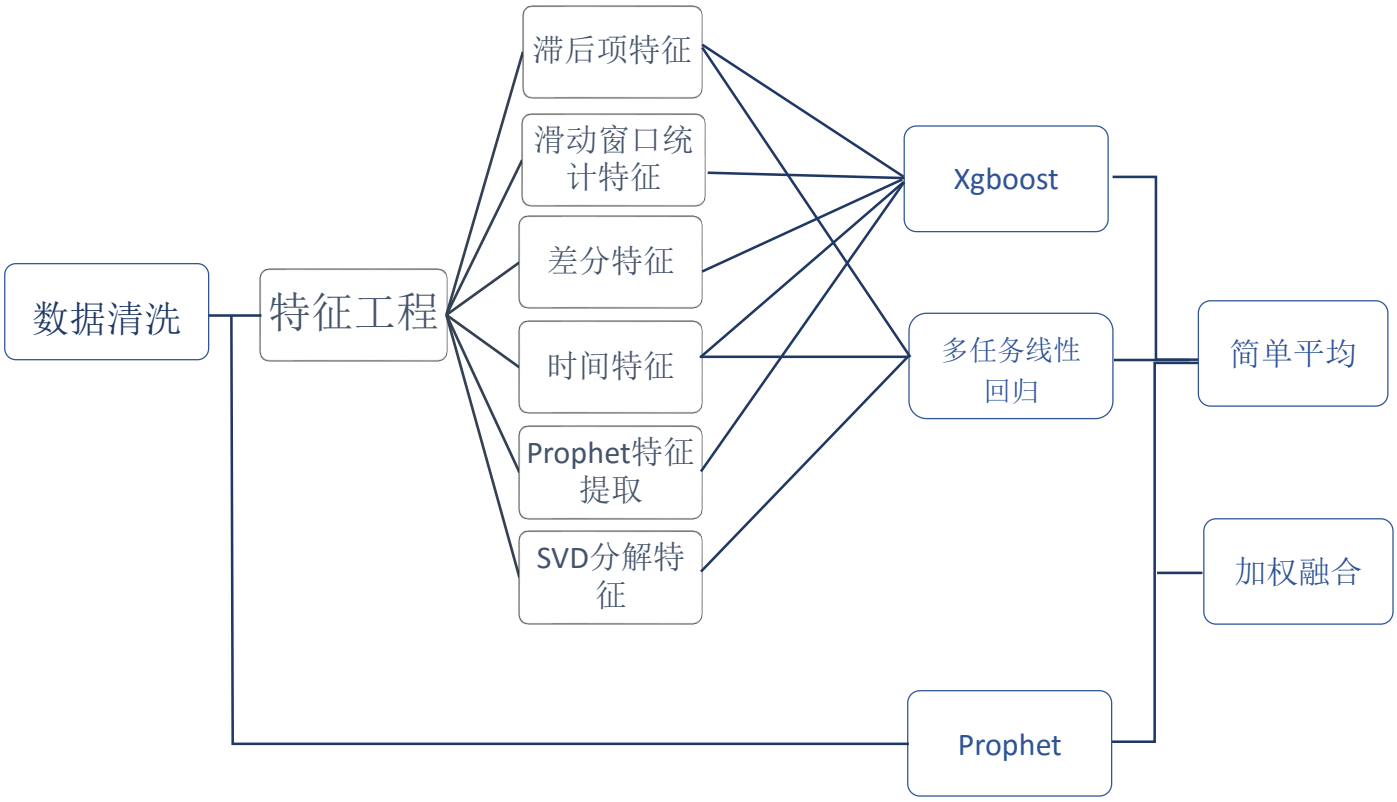
目 录

摘 要

本次大赛的以某钢铁产业园过去四年内的钢卷仓储实际数据作为分析案例，在已有数据的基础上对未来四周的钢卷吞吐量进行预测。针对主要问题和数据特征，本团队首先对数据进行清洗得到两类钢卷的出库、入库时间序列，并对每个时间序列提取一系列时间特征、统计特征以及SVD分解特征，并利用多任务回归、Xgboost、Prophet等多种算法对两类钢卷的出库、入库分别建模，并基于模型的差异性进行了模型融合。综合来看，模型具有较好的预测能力，最终在竞赛中取得了初赛线上得分81.2924，决赛第9，线上得分87.0928的成绩。

关键词：时间序列预测、Xgboost、多任务回归、Prophet、模型融合

技术路线


数据清洗
特征工程
滞后项特征
滑动窗口统计特征
差分特征
时间特征
Prophet特征提取
SVD分解特征
Xgboost
多任务线性回归
Prophet
简单平均
加权融合

1.数据预处理

1.1 数据说明

比赛数据主要分为两大类：仓储数据和基础数据，其中仓储数据中有库存数据，入库数据以及出库数据；基础数据包括字段数据以及储户费用数据。我们本次比赛主要用到的数据是出库数据表以及入库数据表，图 1.1 所示为原始出库数据。

	ext_doc_num	user_id	product_id	product_name	product_count	weight	doc_state	doc_t0	doc_t1
0	CK20140310001	169	3133	圆钢	21	32.7390	11	2014-03-10 22:31:47	2014-03-10 22:37:29
1	CK20140311001	169	3133	圆钢	7	14.4970	11	2014-03-11 12:24:33	2014-03-11 12:25:25
2	CK20140312001	169	3133	圆钢	71	95.2672	11	2014-03-12 18:03:37	2014-03-12 18:03:37
3	CK20140313002	173	3133	圆钢	8	15.7990	11	2014-03-13 13:42:16	2014-03-13 13:48:37
4	CK20140313001	173	3133	圆钢	32	39.6270	11	2014-03-13 11:59:07	2014-03-13 13:51:51
5	CK20140318001	169	3133	圆钢	31	15.6730	11	2014-03-18 10:50:48	2014-03-18 10:53:02

图 1.1 原始出库数据表

1.2 数据预处理

在数据预处理阶段，我们首先对入库数据和出库数据分别进行了整理，将每天的热卷、冷卷的出库、入库数据分别整合，并对未记录的日期添零。以冷卷出库(图 1.2)为例，

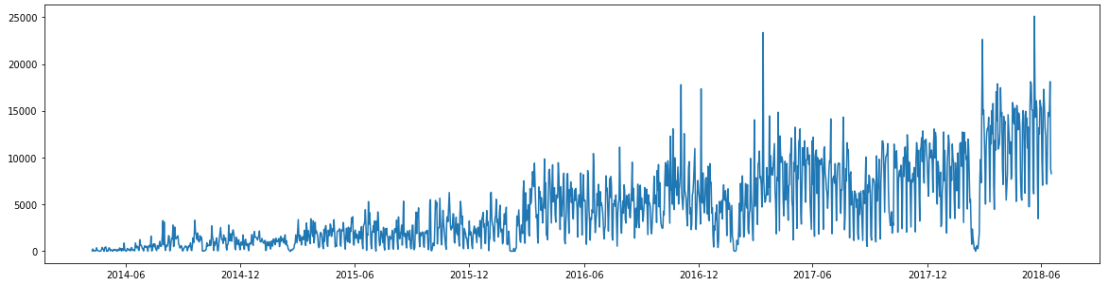


图 1.2 冷卷出库时间序列

可以看出，该时间序列从 2014 年年初开始，持续到 2018 年，初期出库量很小，随着时间推移越来越大，并且呈现方差扩大的趋势

我们采用了多种模型来进行预测，如树模型（Xgboost）,线性回归以及 prophet 模型，针对不同的模型有不同的数据预处理方法。对于线性回归模型，我们截取 2016 年 1 月 1 日之后的数据，并取对数（为避免对 0 取对数，所有数据加 0.1）（图 1.3），来消除异方差。

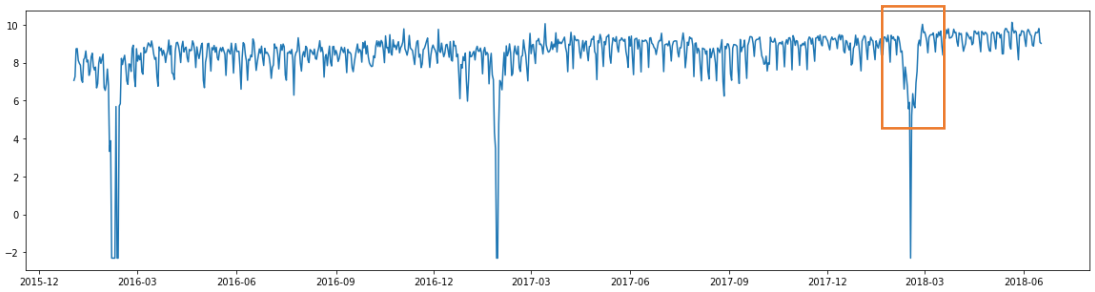


图 1.3 冷卷出库对数时间序列

可以看出时间序列基本平稳，异方差也消除了。但图中显示了每年 2 月都有异常值。对照每年的节假日，不难得出结论，这是由于春节造成的异常。处理方式为对节假日进行标注，并额外标注节假日前与节假日后，这是因为节假日前客户会缩减订单，节假日后仓库需要把积压库存快速转移。所以节假日前，出库量会减小，节假日后，出库量会增加（图中橙色框标注的区域也证明了这一点）。

2. 特征工程

2.1 滞后项特征

根据时间序列偏自相关分析（图 2.1），可以看出当前日期的 7 天滞后项很重要，并且有周循环趋势，14 天循环趋势，28 天循环趋势等。所以我们提取每一个当前日期的最近半个月（1-14 天）的滞后项，以及 21，28，35，56 天滞后项，半年以及年滞后项等作为特征。

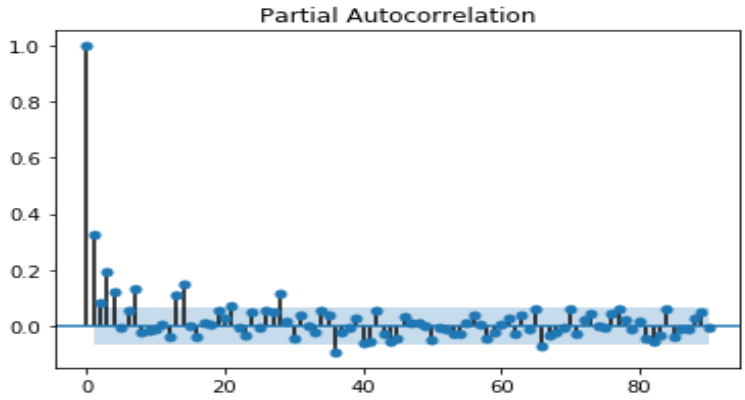


图 2.1 冷卷出库偏自相关分析

2.2 滑动窗口统计特征

虽然偏自相关分析显著的滞后项只有以上部分，但经验告诉我们，时间序列预测中的长

期趋势也能够帮助我们更好地进行预测，所以我们提取了一定的滑动窗口统计特征（2.1），

$$y_t \quad \overbrace{\underbrace{y_{t-1} \quad \cdots \quad y_{t-l}}_{mean,std,middle,\min,\max}}^{1st} \quad \overbrace{y_{t-1-l} \quad \cdots \quad y_{t-2l}}^{2rd} \cdots \quad \overbrace{y_{t-1-(n-1)l} \quad \cdots \quad y_{t-nl}}^{nth} \quad \cdots \quad y_0 \qquad (2.1)$$

具体做法如下：

（1）定义滑动窗口的长度l 和个数n ；

（2）对于每一个当前日期y_t，取其l个滞后项为第一个时窗，并计算窗口内的均值、标准差、中位数、最小值以及最大值 5 个统计特征；

（3）重复（2）中步骤得到n个时窗的统计特征。

根据上面的步骤，我们分别提取了$(l,n)\in\{(7,20),\ (14,12),\ (30,12),\ (90,8)\}$四种时窗中的上述统计特征。

2.3 差分特征

在得到了上面的滑动窗口统计特征之后，我们对同一类相邻窗口的均值特征进行一阶差分得到一系列的一阶差分特征（2.2）。

$$y_t \quad \overbrace{\overbrace{y_{t-1} \quad \cdots \quad y_{t-l}}^{\text{mean1}} \quad \overbrace{y_{t-1-l} \quad \cdots \quad y_{t-2l}}^{mean2}}^{\text{mean2}-mean1} \cdots \quad \overbrace{\overbrace{y_{t-1-(n-2)l} \quad \cdots \quad y_{t-(n-1)l}}^{\text{mean1}} \quad \overbrace{y_{t-1-(n-1)l} \quad \cdots \quad y_{t-nl}}^{mean2}}^{\text{mean(n-1)}-mean(n)} \cdots \quad y_0 \qquad (2.2)$$

2.4 时间特征

对时间我们提取了星期几，是否是周末，是否是法定节假日这几种特征。其中对于节假日特征，我们按照国家统一公布的国家法定节假日时间进行了节假日标注，并且我们发现节假日对其前后临近天数的出库、入库量也会产生影响，所以除了节假日标注之外，我们也对节假日前后效应进行了标注，具体做法如下：

方法一：

对标注的节假日做和选取滞后项同样的操作，即：一个月滞后、一季度滞后，半年滞后，一年滞后。按距离节假日的天数倒数加权，累计得到。如 2018 年节假日后延效应(图 2.2)

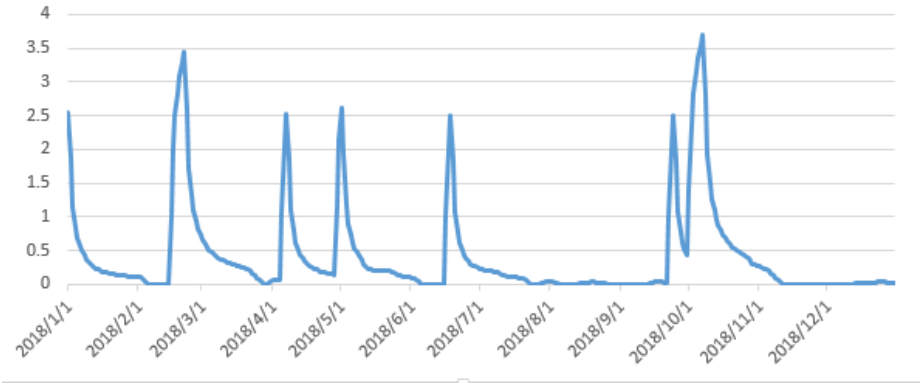


图 2.2 2018 年节假日后延效应

春节等节假日期间，效应为 0，节假日后，有一个短暂的大的冲击效应，后续归零。节假日越长(如春节、国庆)，节假日效应高于短的节假日。(这里可以考虑人为提高春节重要性)。

方法二：

在对所有节假日进行标注之后，我们设置每一个节假日对前后的影响天数，如对于春节设定前 7 天以及后 14 天作为影响范围，将 Prophet 模型分解出的节假日效应作为节假日效应特征（图 2.3）。

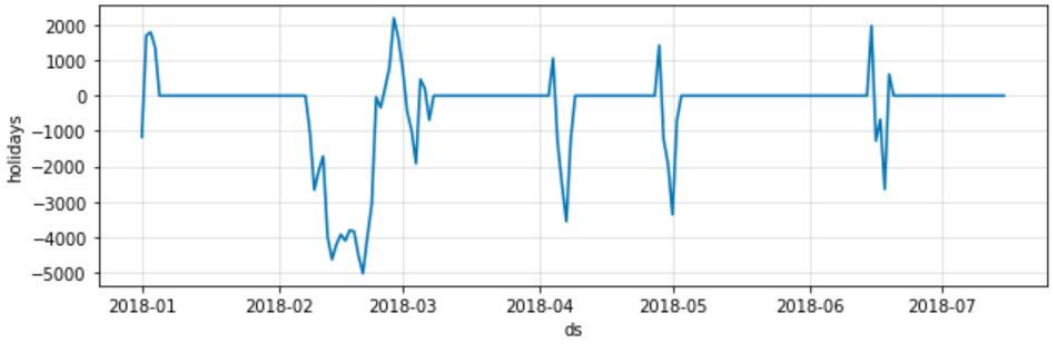


图 2.3 prophet 提取 2018 年节假日效应

从图 2.3 中可以看出，在春节的影响显著地高于其他节假日，Prophet 提取的节假日效应更为合理。

2.5 Prophet 提取时间序列趋势特征

由于 Prophet 是一个可加模型，它将时间时间序列分解成了不同时间频率下的趋势(日，周，年)，然后简单加和（2.2）。

$$y(t) = g(t) + s(t) + h(t) + \varepsilon_t \qquad (2.2)$$

式 2.2 中

$g(t)$：用于拟合时间序列中的分段线性增长或逻辑增长等非周期变化；

$s(t)$：周期变化（如：每周/每年的季节性）；

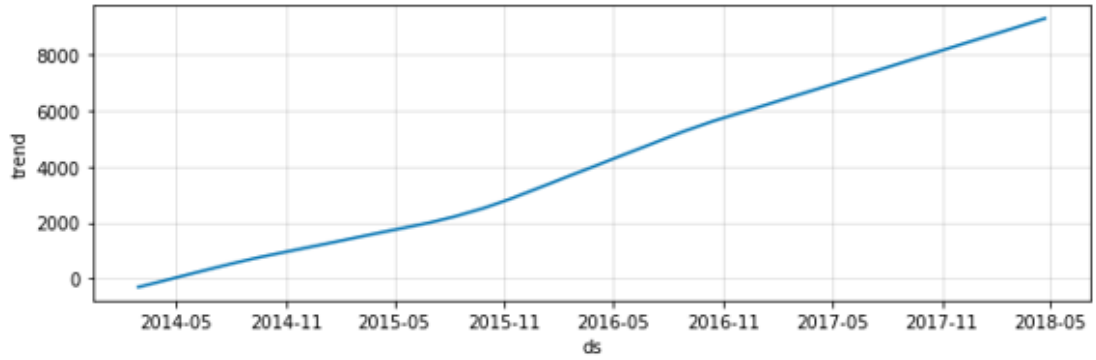
$h(t)$：非规律性的节假日效应（用户造成）；

ε_t：误差项用来反映未在模型中体现异常变动。

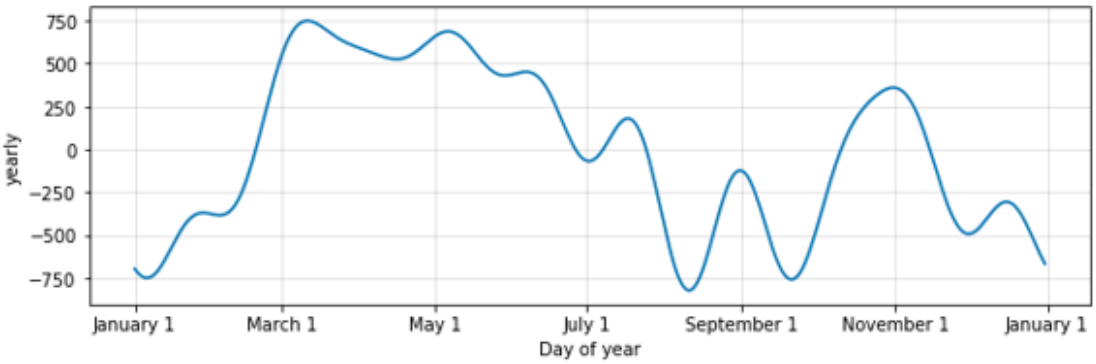
基于这一特点，我们可以将趋势信息更广义地应用在监督学习中（2.3），使模型更好地学习到时间序列的长期趋势信息，

$$\hat{y}(t) = \beta_0 g(t) + \beta_1 s(t) + \beta_2 h(t) \tag{2.3}$$

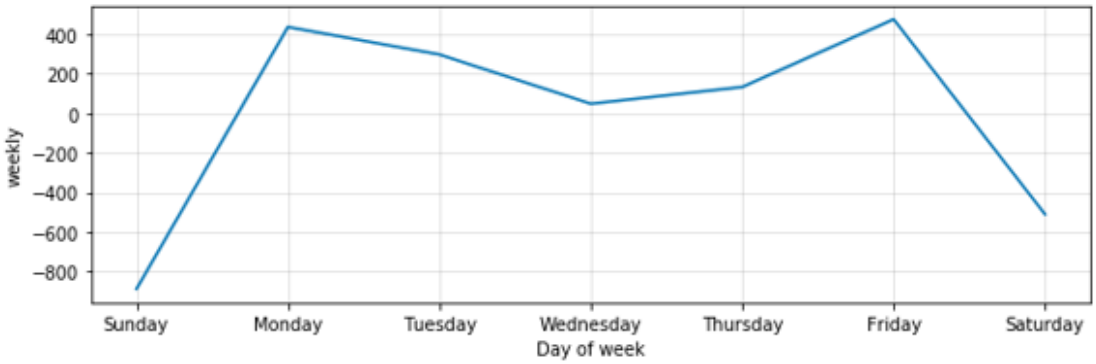
将 Prophet 对整个时间序列分析得到的日，周，年趋势作为特征（图 2.4）



(a)



(b)



(c)

图 2.4　Prophet 提取冷卷出库(a)趋势(b)年季节性趋势(c)周季节性趋势

图 2.4（a）反映了冷卷入库时间序列的增长趋势，而(b)(c)则为以年和周为单位时间的周期性变化，可以看到在 2.4(b)的年趋势中，在 1-2 月与 9-10 月的出库量会有比较明显的波动，可

以认为是受到了春节、国庆等节假日的影响，而在 2.4(c)的周趋势中，出库量在周末会有一个显著下降。

2.6 SVD 特征

原始出库、入库数据中的储户 id、产品 id、出库量、时间，可以理解为用户对商品的购买行为，进而使用推荐系统来分析该问题。由于商品只有两种(冷卷、热卷)，因此不能使用推荐系统来预测，否则，未来的预测几乎等于客户历史购买的重量。但是，我们可以使用推荐系统来挖掘特征，以 SVD 分解为例（2.4）：

$$A = U\Sigma V^T \tag{2.4}$$

V 是 $n\times n$ 的正交阵， V 是 $m\times m$ 的正交阵， $\sum$ 是 $m\times n$ 的对角阵。

通过对每年的用户-商品矩阵分解，可以得到用户特征（图 2.4）和商品特征（图 2.5），之后可以按每天的购买发生情况加和，如：对于 2017 年的用户特征，2017 年 1 月 1 日，A 客户、B 客户和 C 客户中，只有 A 和 B 产生了购买行为，那么 2017 年 1 月 1 日的用户特征就是 A 的特征加上 B 的特征（可以考虑按购买量加权，该比赛中未考虑）。

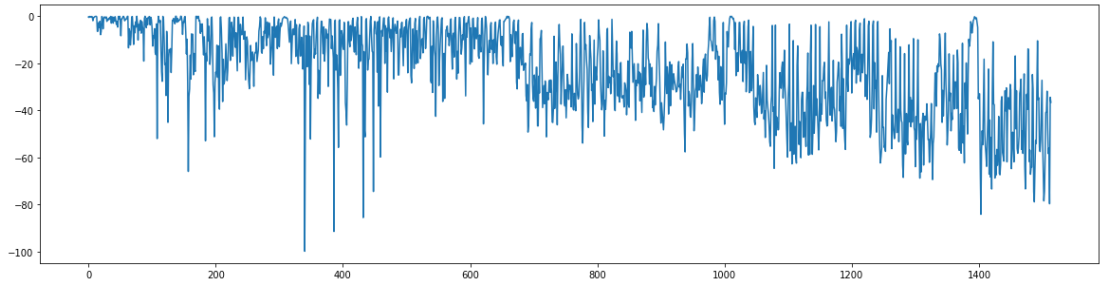


图 2.4 SVD 分解冷卷出库用户特征

同时能得到冷卷特征和热卷特征，第一维冷卷特征如下：

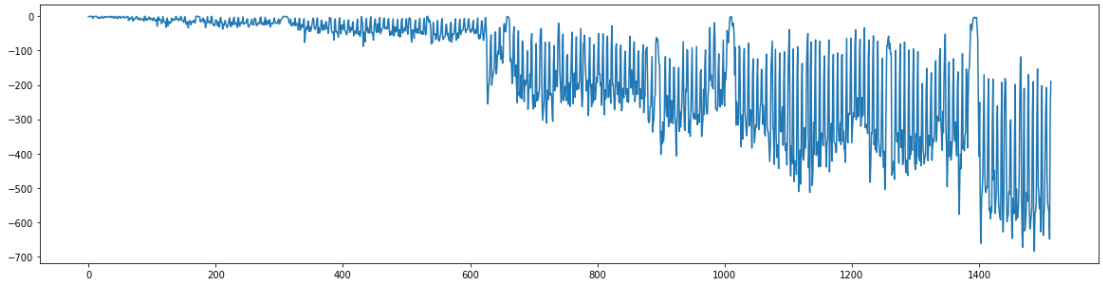


图 2.5 SVD 分解冷卷出库产品（冷卷）特征

这些特征都是原始出库量的某种变换，类似于时间序列的可加模型思想，将这些特征加入后续的监督模型中，可以提升预测效果。（未来的用户特征和商品特征也需要建立时间序列来预测，所以工作量较大，比赛中使用简单的 AR(7)来拟合。）

3.算法选择及检验方法

3.1 单模型

3.1.1 Xgboost

ARIMA 模型是解决时间序列预测问题的经典方法，但 ARIMA 模型（AR 模型）核心也是通过时间滞后项来预测未来，因此，可以通过提取时间滞后项的方式，把时间序列模型转化为监督模型，这样用机器学习的模型来预测时间序列，这样也方便增加特征。

在利用 Xgboost 训练之前，我们选择了上述时间特征、滞后项特征、滑动窗口统计特征、差分特征、Prophet 提取的趋势特征作为 Xgboost 的特征，当天的真实出库、入库重量数据作为标签进行训练。

在训练阶段，我们首先采用了嵌套交叉验证的方法来选择最合适的参数。在时间序列预测中，为了避免信息泄露，选取的验证集的时间范围一定要在训练集之后，所以我们选取 7 天的数据为验证集，而在这 7 天之前的数据为训练集，并将验证集向后滑窗得到 5 组这样的验证集和训练集（图 3.1），在这五组训练集与验证集上对不同的参数组合进行交叉验证，计算不同参数组合在这 5 组训练集与验证集上的平均损失（mae），并应用最小平均损失的模型对应的参数训练全部数据，并进行 28 天的预测，并将每 7 天的数据进行加和得到周预测结果。

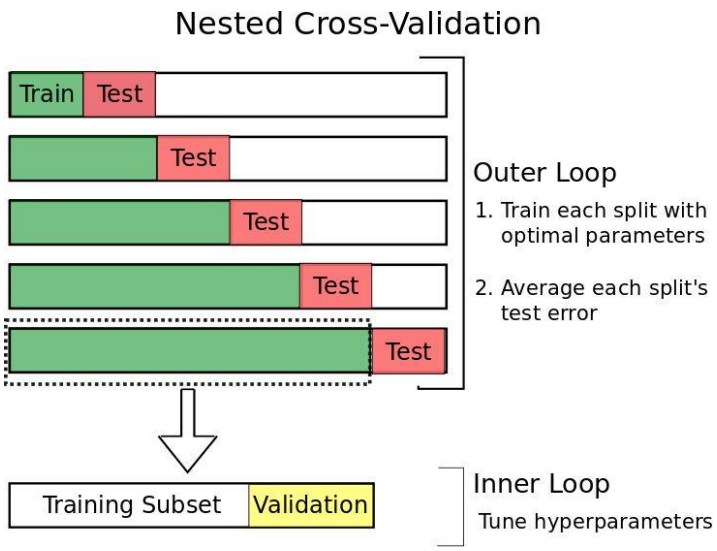


图 3.1 嵌套交叉验证数据集划分

除此之外，我们也尝试了不同的训练集与验证集组合来训练模型，并设置 early-stopping 来防止过拟合，如以测试集起始时间戳的前 7 天为验证集，验证集的前一年的数据为训练集来训练模型,而这种划分方法得到了初赛阶段 Xgboost 单模型的最高成绩 80.9。

3.1.2 多任务-岭回归

该问题有 8 个需要预测的方向，数据上，有出库训练数据和入库训练数据，孤立的使用数据，不能最大化利用数据信息。出库量和入库量并不是独立的，仓库的容量有限，因此，出库量、入库量满足(3.1)：

$$(E(ext-in))^2 \le t \tag{3.1}$$

其中，E(*)表示*的均值,t 是一个常数。

以岭回归为例，损失函数为(3.2)：

$$L(x,y,\beta)=(y-x\beta)^2+\lambda\beta^2 \tag{3.2}$$

其中，x 是自变量，y 是因变量，β 是线性回归系数，λ 是正则项的系数。乘法为矩阵乘法。当x_1，y_1，λ_1表示出库数据的自变量、因变量、正则系数，x_2，y_2，λ_2表示入库数据的自变量、因变量、正则系数，加上上述的出库入库平衡正则项（系数λ_0），损失函数为(3.3)：

$$L=(y_1-x_1\beta_1)^2+(y_2-x_2\beta_2)^2+\lambda_1\beta_1^2+\lambda_2\beta_2^2+\lambda_0(x_1\beta_1-x_2\beta_2)^2 \tag{3.3}$$

通过最小化损失函数，求得(3.4)：

$$\hat{\beta}_1=((1+\lambda_0)x_1^Tx_1+\lambda_1 I)^{-1}(x_1^Ty_1+\lambda_0x_1^Tx_2\beta_2)$$

$$\hat{\beta}_2=((1+\lambda_0)x_2^Tx_2+\lambda_2 I)^{-1}(x_2^Ty_2+\lambda_0x_2^Tx_1\beta_1) \tag{3.4}$$

这样，出库数据和入库数据就可以联合预测，相互修正。

该算法属于迁移模型。迁移模型最好的算法框架是 DNN，但由于时间仓促，比赛中只实现了岭回归的迁移。该算法的损失函数在 $\lambda_0=0$ 时，变为岭回归的损失函数，所以该算法的效果不小于岭回归。

对出库量做数据预处理（取对数、标准化）后，特征采用滞后项特征和时间特征，使用入库量数据（预处理后）帮助预测出库量。最后一阶段比赛中，交叉验证选取参数 $\lambda_0=0.667,\lambda_1=255,\lambda_2=244$ 。选取最后 7 天作为验证集，2016 年 1 月 1 日之后的其他数据作为训练集，

训练集拟合（标准化、对数逆运算后）:（红色为真实值，蓝色为预测值）（图 3.2）

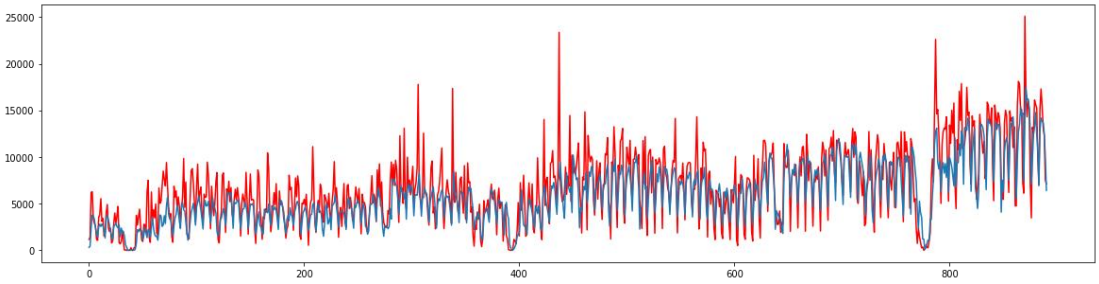


图 3.2 多任务回归训练集拟合结果

验证集拟合结果为图 3.3 所示

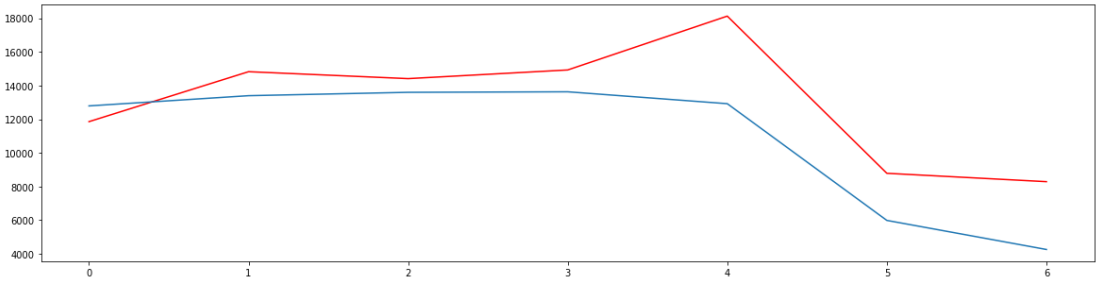


图 3.3 多任务预测验证集拟合结果

3.1.3 Prophet 时间预测模型

$$y(t) = g(t) + s(t) + h(t) + \varepsilon_t \tag{3.5}$$

使用 fbprophet 模块对原始时间序列(可加模型对异方差并没有要求)进行预测。在预测之前对国家法定节假日进行标注，并设定节前节后影响时窗，春节影响最重要，时窗跨度也最大，其他节假日跨度较小。

3.2 模型融合

在进行模型融合时，高效的单模型简单平均一般就能得到好的效果。进一步，在融合中，一个更高效的单模型应该有一个大的权重。单模型在验证集上的预测结果 $\hat{y}_1$和$\hat{y}_2$ ，对真实值 y 再建立一个线性回归模型，即最小化均方误差 mse：

$$\begin{aligned} &\min : mse(y, \hat{y}) \\ &\hat{y} = k\hat{y}_1 + (1-k)\hat{y}_2 \end{aligned} \tag{3.6}$$

简单求导，得到：

$$k = \frac{mse_2 - \rho\sqrt{mse_1 mse_2}}{mse_1 + mse_2 - 2\rho\sqrt{mse_1 mse_2}} \tag{3.7}$$

$$\rho = \frac{E(y-y_1)(y-y_2)}{\sqrt{mse_1 mse_2}} \tag{3.8}$$

$$mse1 = mse(y, \hat{y}_1), mse2 = mse(y, \hat{y}_2) \tag{3.9}$$

将 mse 替换为 mape 的平方，ρ 是衡量两个预测的相似度，用

$$\rho = (\frac{min(mape_1, mape_2)}{max(mape_1, mape_2)})^i \tag{3.10}$$

替代。I 越大，相似度区分越严格（即：相似度越小）。得到：

$$k = \frac{mape_2^{\ 2} - \rho mape_1 mape_2}{mape_1^{\ 2} + mape_2^{\ 2} - 2\rho mape_1 mape_2} \tag{3.11}$$

使用该系数进行两模型的加权平均。比赛中由于我们认为两种方法（fbprophet 和 xgboost+多任务岭回归）区别较大，因此选择了 i=100,即：ρ→0 的权重。该系数是 mse 的简单推广，但保证了 mape 小的模型有一个大的权重。

4.算法代码实现说明

4.1 数据清洗

```
import pandas as pd
import numpy as np
import datetime as dt
from pandas.tseries.offsets import Day
from fbprophet import Prophet

train_ext_data = pd.read_csv('data/trainData_EXT.csv')
train_ext_data = pd.DataFrame(train_ext_data)
train_ext_data.columns = (['ext_doc_num','user_id', 'product_id',
                       'product_name','product_count', 'weight',
                       'doc_state', 'doc_t0', 'doc_t1'])
train_in_data = pd.read_csv('data/trainData_IN.csv')
train_in_data = pd.DataFrame(train_in_data)
train_in_data.columns = (['in_doc_numb', 'user_id', 'product_id',
                      'product_name', 'product_count', 'weight',
                      'doc_state', 'doc_t0', 'doc_t1'])
dataset1 = train_ext_data[(train_ext_data.product_name=='热卷
')|(train_ext_data.product_name=='圆钢')]
dataset2 = train_ext_data[(train_ext_data.product_name!='热卷
')&(train_ext_data.product_name!='圆钢')]
```

```
dataset3 = train_in_data[(train_in_data.product_name=='热卷
')|(train_in_data.product_name=='圆钢')]
dataset4 = train_in_data[(train_in_data.product_name!='热卷
')&(train_in_data.product_name!='圆钢')]

date_range_in = pd.date_range('27/2/2014','25/2/2018',freq='D')
d_list = pd.DataFrame(date_range_in,columns =
['doc_t0']).astype(str)
date_range_out = pd.date_range('3/10/2014','25/2/2018',freq='D')
d_list_out = pd.DataFrame(date_range_out,columns =
['doc_t0']).astype(str)
#热卷出库
t1 = dataset1[['weight','doc_t0']].copy()
t1.doc_t0 = t1.doc_t0.apply(lambda x : x.split(' ')[0])
t1 = t1.groupby(['doc_t0']).sum().reset_index()
t1=pd.merge(t1,d_list_out,how="right",on="doc_t0")
t1.sort_values(by=['doc_t0'],inplace=True)
t1.weight.fillna(0,inplace=True)
t1.rename(columns={t1.columns[0]:'date'},inplace=True)
t1.index = t1.date
#冷卷出库
t2 = dataset2[['weight','doc_t0']].copy()
t2.doc_t0 = t2.doc_t0.apply(lambda x : x.split(' ')[0])
t2 = t2.groupby(['doc_t0']).sum().reset_index()
t2=pd.merge(t2,d_list_out,how="right",on="doc_t0")
t2.sort_values(by=['doc_t0'],inplace=True)
t2.weight.fillna(0,inplace=True)
t2.rename(columns={t2.columns[0]:'date'},inplace=True)
t2.index = t2.date
#热卷入库
t = dataset3[['weight','doc_t0']].copy()
t.doc_t0 = t.doc_t0.apply(lambda x : x.split(' ')[0])
t = t.groupby(['doc_t0']).sum().reset_index()
t=pd.merge(t,d_list,how="right",on="doc_t0")
t.sort_values(by=['doc_t0'],inplace=True)
t.weight.fillna(0,inplace=True)
t.rename(columns={t.columns[0]:'date'},inplace=True)
t.index = t.date
#冷卷入库
t4 = dataset4[['weight','doc_t0']].copy()
t4.doc_t0 = t4.doc_t0.apply(lambda x : x.split(' ')[0])
t4 = t4.groupby(['doc_t0']).sum().reset_index()
t4=pd.merge(t4,d_list,how="right",on="doc_t0")
t4.sort_values(by=['doc_t0'],inplace=True)
```

```
t4.weight.fillna(0,inplace=True)
t4.rename(columns={t4.columns[0]:'date'},inplace=True)
t4.index = t4.date
```

4.2 特征工程

4.2.1 滞后项特征

以提取 14 天滞后项为例，

```
def get_delay(df,lag,fillna=False):
    for i in range(1,lag+1):
        df['weight_d%d' %i] = df['weight'].shift(i)
    if fillna:
        df.fillna(0, inplace=True)
    return df
t1 = get_delay(t1,14)
t2 = get_delay(t2,14)
t = get_delay(t,14)
t4 = get_delay(t4,14) #对四个数据集分别提取
```

4.2.2 滑动窗口统计特征

以 20 个 7 天滑动窗口统计特征为例，核心函数如下：

```
def roll_back_week(df,time_window_num):
      for n in range(1,time_window_num+1):
          df['weight_w%d_max' %(n)] = 0
          df['weight_w%d_min' %(n)] = 0
          df['weight_w%d_mean' %(n)] = 0
          df['weight_w%d_std' %(n)] = 0
          df['weight_w%d_mid' %(n)] = 0
          for i in range(df.shape[0]):
              start = df.date.iloc[i] - n*7*Day()
              end = start + n*7*Day() - Day()
              ts = df[(df.date>=start)&(df.date<=end)]
              df['weight_w%d_max' %(n)].iloc[i] = ts.weight.max()
              df['weight_w%d_min' %(n)].iloc[i] = ts.weight.min()
              df['weight_w%d_mean' %(n)].iloc[i] = ts.weight.mean()
              df['weight_w%d_std' %(n)].iloc[i] = ts.weight.std()
              df['weight_w%d_mid' %(n)].iloc[i] = ts.weight.median()
      return df
t = roll_back_week(t,20)
t1 = roll_back_week(t1,20)
t2 = roll_back_week(t2,20)
```

```
t4 = roll_back_week(t4,20)#对四个数据集分别提取
```

4.2.3 差分特征

相邻同类时窗的均值进行一阶差分：

```
for i in range(2,21):
    t['mean w%d - w%d '%(i,i-1)] = t['weight_w%d_mean' %i]-
t['weight_w%d_mean' %(i-1)]
    t1['mean w%d - w%d '%(i,i-1)] = t1['weight_w%d_mean' %i]-
t1['weight_w%d_mean' %(i-1)]
    t2['mean w%d - w%d '%(i,i-1)] = t2['weight_w%d_mean' %i]-
t2['weight_w%d_mean' %(i-1)]
    t4['mean w%d - w%d '%(i,i-1)] = t4['weight_w%d_mean' %i]-
t4['weight_w%d_mean' %(i-1)]
```

4.2.4 Prophet 提取节假日效应及趋势项

需要我们首先对所有的节假日进行标注，再利用 Prophet 提取节假日效应及趋势项。

```
#元旦
yd = pd.DataFrame({'holiday': "yuandan", 'ds' :
pd.to_datetime(['2015-1-1','2015-1-2','2015-1-3', '2016-1-
1','2016-1-2','2016-1-3', '2016-12-31','2017-1-1','2017-1-2',
'2017-12-30','2017-12-31','2018-1-1']), 'lower_window':-3,
'upper_window':3})
#春节
cj = pd.DataFrame({'holiday': "chunjie", 'ds' :
pd.to_datetime(['2015-2-18','2015-2-19','2015-2-20', '2015-2-
21','2015-2-22','2015-2-23', '2015-2-24', '2016-2-7','2016-2-8',
'2016-2-9', '2016-2-10','2016-2-11','2016-2-12', '2016-2-13',
'2017-1-27','2017-1-28','2017-1-29', '2017-1-30','2017-1-
31','2017-2-1', '2017-2-2', '2018-2-15','2018-2-16','2018-2-17',
'2018-2-18','2018-2-19','2018-2-20', '2018-2-21']),
'lower_window':-7, 'upper_window':14})
#清明
qm = pd.DataFrame({'holiday': "qingming", 'ds' :
pd.to_datetime(['2014-4-5','2014-4-6','2014-4-7', '2015-4-
4','2015-4-5','2015-4-6', '2016-4-2','2016-4-3','2016-4-4',
'2017-4-2','2017-4-3','2017-4-4', '2018-4-5','2018-4-6','2018-4-
7']), 'lower_window':-1, 'upper_window':1})
#五一
wy = pd.DataFrame({'holiday': "wuyi", 'ds' :
pd.to_datetime(['2014-5-1','2014-5-2','2014-5-3', '2015-5-
1','2015-5-2','2015-5-3', '2016-4-30','2016-5-1','2016-5-2',
'2017-4-29','2017-4-30','2017-5-1', '2018-4-29','2018-4-
```

```
30','2018-5-1']), 'lower_window':-1, 'upper_window':1})
#端午
dw = pd.DataFrame({'holiday': "duanwu", 'ds' :
pd.to_datetime(['2014-5-31','2014-6-1','2014-6-2', '2015-6-
20','2015-6-21','2015-6-22', '2016-6-9','2016-6-10','2016-6-11',
'2017-5-28','2017-5-29','2017-5-30', '2018-6-16','2018-6-
17','2018-6-18']), 'lower_window':-1, 'upper_window':1})
#中秋
zq = pd.DataFrame({'holiday': "zhongqiu", 'ds' :
pd.to_datetime(['2014-9-6','2014-9-7','2014-9-8', '2015-9-
26','2015-9-27','2015-9-28', '2016-9-15','2016-9-16','2016-9-
17']), 'lower_window':-1, 'upper_window':1})
#国庆
gq = pd.DataFrame({'holiday': "guoqing", 'ds' :
pd.to_datetime(['2014-10-1','2014-10-2','2014-10-3', '2014-10-
4','2014-10-5','2014-10-6', '2014-10-7', '2015-10-1','2015-10-
2','2015-10-3', '2015-10-4','2015-10-5','2015-10-6', '2015-10-7',
'2016-10-1','2016-10-2','2016-10-3', '2016-10-4','2016-10-
5','2016-10-6', '2016-10-7','2017-10-1','2017-10-2','2017-10-3',
'2017-10-4','2017-10-5','2017-10-6', '2017-10-7','2017-10-8']),
'lower_window':-1, 'upper_window':1})
holiday = pd.concat((yd,cj,qm,wy,dw,zq,gq))
```

提取节假日效应与趋势项的主函数：

```
def prophet_feature(df):
   ts = df[['date','weight']].copy()
   ts = ts.loc[:'2018-01-28',:]
   ts.columns = ['ds','y']
   m_prophet = Prophet(holidays=holiday)
   m_prophet.fit(ts)
   future = m_prophet.make_future_dataframe(periods=28)
   forecast = m_prophet.predict(future)
   tr=forecast[['ds','trend','holidays','weekly','yearly']].copy()
    tr.index = tr.ds
    df['trend'] = tr.trend
    df['holiday'] = tr.holidays
    df['weekly'] = tr.weekly
    df['yearly'] = tr.yearly
    return df
t1 = prophet_feature(t1)
t2 = prophet_feature(t2)
t = prophet_feature(t)
t4 = prophet_feature(t4)# 对四个数据集分别提取
```

4.2.5 SVD 特征

使用 numpy 中的 linalg.svd 实现 SVD 分解，核心代码：（1）整合用户-商品矩阵：

```
def user_item_peryear(data):##data 有四列，列名：储户 id，产品名称，重量，单据创建时间
    data.单据创建时间=year(data.单据创建时间)
    data.loc[data.单据创建时间==2014,'单据创建时间']=2015
    data.loc[data.单据创建时间==2018,'单据创建时间']=2017
    ui=[]
    for i in list(np.unique(data.单据创建时间)):
        data_tem=data[data.单据创建时间==i]
        user_item=pd.DataFrame(np.zeros((len(np.unique(data.储户 id)),len(np.unique(data.产品名称)))))
        user_item.columns=np.unique(data.产品名称)
        user_item.index=np.unique(data.储户 id)
        data_tem_clear=data_tem.groupby(["单据创建时间","储户 id","产品名称"]).sum().reset_index()
        for m in user_item.columns:
            for n in user_item.index:
                if len(data_tem_clear[(data_tem_clear.储户 id==n)&(data_tem_clear.产品名称==m)].重量)!=0:
                    #print(data_tem_clear[(data_tem_clear.储户 id==n)&(data_tem_clear.产品名称==m)].重量.values)

user_item.loc[n,m]=data_tem_clear[(data_tem_clear.储户 id==n)&(data_tem_clear.产品名称==m)].重量.values
        ui.append(user_item)
    return ui
```

（2）SVD 分解：

```
from numpy import linalg as la
U_2015,sigma_2015,VT_2015=la.svd(ui[0])
U_2016,sigma_2016,VT_2016=la.svd(ui[1])
U_2017,sigma_2017,VT_2017=la.svd(ui[2])
```

4.3 模型选择及预测

4.3.1 Xgboost

```
import pandas as pd
import xgboost as xgb
from pandas.tseries.offsets import Day
```

```
date_range_in = pd.date_range('27/2/2014','25/2/2018',freq='D')
date_range_out = pd.date_range('3/10/2014','25/2/2018',freq='D')

t1 = pd.read_csv('hot_ext_clear.csv')
t1 = pd.DataFrame(t1)
t1.index = t1.date
t1.date = date_range_out
t2 = pd.read_csv('cool_ext_clear.csv')
t2 = pd.DataFrame(t2)
t2.index = t2.date
t2.date = date_range_out
t=pd.read_csv('hot_in_clear.csv')
t = pd.DataFrame(t)
t.index = t.date
t.date = date_range_in
t4 = pd.read_csv('cool_in_clear.csv')
t4 = pd.DataFrame(t4)
t4.index = t4.date
t4.date = date_range_in

data1 = t1.copy()
data1 = data1.drop(['date'],axis=1)
data2 = t2.copy()
data2 = data2.drop(['date'],axis=1)
data = t.copy()
data = data.drop(['date'],axis=1)
data4 = t4.copy()
data4 = data4.drop(['date'],axis=1)

params={
    'booster':'gbtree',
    'objective': 'reg:linear',
    'eval_metric':'mae',
    'gamma':0.1,
    'min_child_weight':3,
    'max_depth':7,
    'lambda':0,
    'subsample':0.7,
    'colsample_bytree':0.7,
    'colsample_bylevel':0.7,
    'eta': 0.01,
    'tree_method':'exact',
    'seed':500,
    'nthread':12
```

```
        }
plst = list(params.items())

def mxgboost(train_x,train_y,test_x,test_y,plst):
    xgb_train = xgb.DMatrix(train_x, label=train_y)
    xgb_test = xgb.DMatrix(test_x, label=test_y)
    del(train_x,train_y,test_x,test_y)
    watchlist = [(xgb_train, 'train'),(xgb_test, 'val')]
    model=xgb.train(plst,
xgb_train,num_boost_round=2000,evals=watchlist,early_stopping_rou
nds=50)
    return model

def mpre_xgb(model,data):
    xgb_data=xgb.DMatrix(data)
    return model.predict(xgb_data)

hot_in_train=data.iloc[-401:-36,:]
hot_in_valid=data.iloc[-36:-29,:]
hot_in_train_x=hot_in_train.drop('weight',axis=1)
hot_in_train_y=hot_in_train['weight']
hot_in_valid_x=hot_in_valid.drop('weight',axis=1)
hot_in_valid_y=hot_in_valid['weight']
model_hot_in=mxgboost(hot_in_train_x,hot_in_train_y,hot_in_valid_
x,hot_in_valid_y,plst)
model_hot_in.save_model('hot_in.model')

cool_in_train=data4.iloc[-401:-36,:]
cool_in_valid=data4.iloc[-36:-29,:]
cool_in_train_x=cool_in_train.drop('weight',axis=1)
cool_in_train_y=cool_in_train['weight']
cool_in_valid_x=cool_in_valid.drop('weight',axis=1)
cool_in_valid_y=cool_in_valid['weight']
model_cool_in=mxgboost(cool_in_train_x,cool_in_train_y,cool_in_va
lid_x,cool_in_valid_y,plst)
model_cool_in.save_model('cool_in.model')

hot_ext_train=data1.iloc[-401:-36,:]
hot_ext_valid=data1.iloc[-36:-29,:]
hot_ext_train_x=hot_ext_train.drop('weight',axis=1)
hot_ext_train_y=hot_ext_train['weight']
hot_ext_valid_x=hot_ext_valid.drop('weight',axis=1)
hot_ext_valid_y=hot_ext_valid['weight']
model_hot_ext=mxgboost(hot_ext_train_x,hot_ext_train_y,hot_ext_va
```

```
lid_x,hot_ext_valid_y,plst)
model_hot_ext.save_model('hot_ext.model')

cool_ext_train=data2.iloc[-401:-36,:]
cool_ext_valid=data2.iloc[-36:-29,:]
cool_ext_train_x=cool_ext_train.drop('weight',axis=1)
cool_ext_train_y=cool_ext_train['weight']
cool_ext_valid_x=cool_ext_valid.drop('weight',axis=1)
cool_ext_valid_y=cool_ext_valid['weight']
model_cool_ext=mxgboost(cool_ext_train_x,cool_ext_train_y,cool_ex
t_valid_x,cool_ext_valid_y,plst)
model_cool_ext.save_model('cool_ext.model')

def get_delay(cur_time,df,i):
    df.loc[cur_time,'weight_d%d' %i] =
df[df.date==(df.loc[cur_time].date-i*Day())].weight[0]
    return df
def roll_b_week(cur_time,df,time_window_num):
    for n in range(1,time_window_num+1):
        start = df.loc[cur_time].date - n*7*Day()
        end = start + n*7*Day() - Day()
        ts = df[(df.date>=start)&(df.date<=end)]
        df.loc[cur_time,'weight_w%d_max' %(n)] = ts.weight.max()
        df.loc[cur_time,'weight_w%d_min' %(n)] = ts.weight.min()
        df.loc[cur_time,'weight_w%d_mean' %(n)] = ts.weight.mean()
        df.loc[cur_time,'weight_w%d_std' %(n)] = ts.weight.std()
        df.loc[cur_time,'weight_w%d_mid' %(n)] = ts.weight.median()
    return df
def roll_b_month(cur_time,df,time_window_num):
    for n in range(1,time_window_num+1):
        start =  df.loc[cur_time].date - n*30*Day()
        end = start + n*30*Day() - Day()
        ts = df[(df.date>=start)&(df.date<=end)]
        df.loc[cur_time,'weight_m%d_max' %(n)] = ts.weight.max()
        df.loc[cur_time,'weight_m%d_min' %(n)] = ts.weight.min()
        df.loc[cur_time,'weight_m%d_mean' %(n)] = ts.weight.mean()
        df.loc[cur_time,'weight_m%d_std' %(n)] = ts.weight.std()
        df.loc[cur_time,'weight_m%d_mid' %(n)] = ts.weight.median()
    return df
def roll_b_quarter(cur_time,df,time_window_num):
    for n in range(1,time_window_num+1):
        start = df.loc[cur_time].date - n*90*Day()
        end = start + n*90*Day() - Day()
        ts = df[(df.date>=start)&(df.date<=end)]
```

```
        df.loc[cur_time,'weight_q%d_max' %(n)] = ts.weight.max()
        df.loc[cur_time,'weight_q%d_min' %(n)] = ts.weight.min()
        df.loc[cur_time,'weight_q%d_mean' %(n)] = ts.weight.mean()
        df.loc[cur_time,'weight_q%d_std' %(n)] = ts.weight.std()
        df.loc[cur_time,'weight_q%d_mid' %(n)] = ts.weight.median()
    return df
def roll_b_halfmonth(cur_time,df,time_window_num):
    for n in range(1,time_window_num+1):
        start = df.loc[cur_time].date - n*14*Day()
        end = start + n*14*Day() - Day()
        ts = df[(df.date>=start)&(df.date<=end)]
        df.loc[cur_time,'weight_h%d_max' %(n)] = ts.weight.max()
        df.loc[cur_time,'weight_h%d_min' %(n)] = ts.weight.min()
        df.loc[cur_time,'weight_h%d_mean' %(n)] = ts.weight.mean()
        df.loc[cur_time,'weight_h%d_std' %(n)] = ts.weight.std()
        df.loc[cur_time,'weight_h%d_mid' %(n)] = ts.weight.median()
    return df
def get_new_feature(cur_time,test_data_d):

    for i in range(1,15):
        test_data_d = get_delay(cur_time,test_data_d,i)
    test_data_d = get_delay(cur_time,test_data_d,21)
    test_data_d = get_delay(cur_time,test_data_d,28)

    test_data_d = roll_b_week(cur_time,test_data_d,20)
    for i in range(2,21):
        test_data_d.loc[cur_time,'mean w%d - w%d '%(i,i-1)] =
test_data_d.loc[cur_time]['weight_w%d_mean' %i]-
test_data_d.loc[cur_time]['weight_w%d_mean' %(i-1)]

    test_data_d = roll_b_month(cur_time,test_data_d,12)
    for i in range(2,13):
        test_data_d.loc[cur_time,'mean m%d - m%d '%(i,i-1)] =
test_data_d.loc[cur_time]['weight_m%d_mean' %i]-
test_data_d.loc[cur_time]['weight_m%d_mean' %(i-1)]

    test_data_d = roll_b_quarter(cur_time,test_data_d,8)
    for i in range(2,9):
        test_data_d.loc[cur_time,'mean w%d - q%d '%(i,i-1)] =
test_data_d.loc[cur_time]['weight_q%d_mean' %i]-
test_data_d.loc[cur_time]['weight_q%d_mean' %(i-1)]

    test_data_d = roll_b_halfmonth(cur_time,test_data_d,12)
    for i in range(2,13):
```

```
        test_data_d.loc[cur_time,'mean h%d - h%d '%(i,i-1)] =
test_data_d.loc[cur_time]['weight_h%d_mean' %i]-
test_data_d.loc[cur_time]['weight_h%d_mean' %(i-1)]
    return test_data_d.loc[cur_time]

def predict(data_clear,model):

    date_range_test =
pd.date_range('18/6/2018','15/7/2018',freq='D')
    d_list_test = pd.DataFrame(date_range_test,columns =
['date']).astype(str)

    tt = data_clear.copy()
    res = []
    for i in range(len(d_list_test)):
        cur_time = d_list_test.iloc[i].date
        if i==0:
            cur_test_data = tt.loc[cur_time]
        else :
            cur_test_data = get_new_feature(cur_time,tt)

        cur_test_data =
cur_test_data.drop(['date','weight','product_name'])
        cur_test_data = pd.to_numeric(cur_test_data)
        cur_test_data = pd.DataFrame(cur_test_data).T

        new_weight = mpre_xgb(model,cur_test_data)
        res.extend(new_weight)
        tt.loc[cur_time,'weight'] = float(new_weight)
    res.append(sum(res[0:7]))
    res.append(sum(res[7:14]))
    res.append(sum(res[14:21]))
    res.append(sum(res[21:28]))
    return res
predict(t1,model_hot_ext)
predict(t2,model_cool_ext)
predict(t,model_hot_in)
predict(t4,model_cool_in)
```

4.3.2 多任务岭回归

使用(3.4)导出的系数计算公式,采用cholesky分解对对称正定阵求逆,最终得到 $\hat{\beta}_1$ 。核心代码如下：

```
import numpy as np
from sklearn.metrics import mean_absolute_error
import pandas as pd
from scipy.linalg import solve_triangular
def multitask(x1,y1,x2,y2,lambda1,lambda2,lambda3,times=100):
##x1,x2 样本量必须相同
    (n,p1)=x1.shape
    (n,p2)=x2.shape
    beta1=np.zeros(p1+1).reshape(-1,1)
    beta2=np.zeros(p2+1).reshape(-1,1)
    cont = np.ones(n)
    x1=np.insert(x1, 0, values=cont, axis=1) #插入常数项
    x2=np.insert(x2, 0, values=cont, axis=1)
    x1x1=np.dot(x1.T,x1)
    x1y1=np.dot(x1.T,y1).reshape(-1,1)
    x1x2=np.dot(x1.T,x2)
    x2x2=np.dot(x2.T,x2)
    x2y2=np.dot(x2.T,y2).reshape(-1,1)

    def beta1hat(beta2): #计算 β̂1
        M1=(1+lambda1)*x1x1+lambda2*np.identity(p1+1)
        L1=np.linalg.cholesky(M1)
        b_1 = solve_triangular(L1,
(x1y1+np.dot(lambda1*x1x2,beta2)).reshape((-1,1)), lower=True)
        beta1 = solve_triangular(L1.T, b_1, lower=False)
        return beta1

    def beta2hat(beta1): #计算 β̂2
        M2=(1+lambda1)*x2x2+lambda3*np.identity(p2+1)
        L2=np.linalg.cholesky(M2)
        b_2 = solve_triangular(L2,
(x2y2+np.dot(lambda1*(x1x2.T),beta1)).reshape((-1,1)),
lower=True)
        beta2 = solve_triangular(L2.T, b_2, lower=False)
        return beta2
    i=0
    while True:
        i+=1
        b2=beta2
        b1=beta1
        beta1=beta1hat(beta2)
        beta2=beta2hat(beta1)
        if (np.mean((beta1-b1)**2)+np.mean((beta2-b2)**2))<0.001:
            return {"beta1":beta1,"beta2":beta2}
```

```
        if i>=times:
            return {"beta1":beta1,"beta2":beta2}
```

4.3.3 prophet 模型

使用 facebook 开源的 fbprophet 模块。核心代码：

```
from fbprophet import Prophet
ts1=cool_ext['重量'].reset_index()
ts1.columns=['ds','y']
m = Prophet(holidays=nn) #nn 是自定义的节假日，具体见完整代码
m.fit(ts1)
future = m.make_future_dataframe(periods=length)
forecast = m.predict(future)
print(forecast)
```