时间序列分析（3） Linear Regression


随风
大数据、人工智能

关注他

19 人赞同了该文章

1 前言

在上一章中，我们介绍了 ARIMA 模型，从**自相关性**的角度对时间序列进行了建模。本章将介绍线性回归（linear regression）模型，从特征的角度对时间序列进行建模。 时间序列分析（2） ARIMA 模型：

语言：python3

数据集：余额宝在2014-04-01~2014-08-10期间每日申购的总金额（数据来自天池大赛）

数据下载地址：tianchi.aliyun.com/comp

2 线性回归模型

给定 d 个属性 $x=(x_1,x_2,...,x_d)$ ，其中 x_i 是 x 在第 i 个属性上的取值，线性回归模型试图学得一个通过属性的线性组合来进行预测的函数，即：

$$f(x)=w_1x_1+w_2x_2+...+w_dx_d+b\ .$$

一般用向量的形式表示：

$$f(x)=wx^T+b, w=(w_1,w_2,...,w_d).$$

模型要学习的变量就是 w 和 b 。

对于给定数据集 $D=(x_1,y_1),(x_2,y_2),...,(x_m,y_m)$ ，其中

$x_i=(x_{i1},x_{i2},...,x_{id})$， $y_i \epsilon R$ 。线性回归试图学得一个线性模型以尽可能准确地预测标记，即使得 $f(x_i)\simeq y_i$ 。

如何确定 w 和 b 呢？关键在于如何衡量 $f(x)$ 与 y 之间的差别。通常采用均方误差最小化：

$$(w^*,b^*)=arg\min_{w,b}\Sigma_{i=1}^m(f(x_i)-y_i)^2,\quad f(x_i)=w^Tx_i+b\ .$$

均方误差有非常好的几何意义，它对应欧几里得距离，简称“欧氏距离”。基于均方误差最小化来进行模型求解的方法称为“最小二乘法”。在线性回归中，最小二乘法试图找到一条直线，使得所有样本到直线的欧氏距离之和最小。我们把 w 和 b 写成向量的形式 $\tilde{w}=(w,b)$ 相应的，把数据集 D 表示为一个 $m*(d+1)$ 的矩阵 X ，其中每行对应于一个示例，该行前 d 个元素对应于示例的 d 个属性值，最后一个元素恒置为1，即

$$X=\begin{pmatrix} x_{11} & x_{12} & \dots & x_{1d} & 1 \\ x_{21} & x_{22} & \dots & x_{2d} & 1 \\ \dots & \dots & \dots & \dots & 1 \\ x_{m1} & x_{m2} & \dots & x_{md} & 1 \end{pmatrix}=\begin{pmatrix} x_1^T & 1 \\ x_2^T & 1 \\ \dots & \dots \\ x_m^T & 1 \end{pmatrix}$$

再把标记也改写成向量的形式 $y=(y_1,y_2,\dots,y_m)$ ，则有

$$\tilde{w}^*=arg\min_{\tilde{w}}(y-X\tilde{w})^T(y-X\tilde{w})$$.

令 $E_{\tilde{w}}=(y-X\tilde{w})^T(y-X\tilde{w})$ ，对 $\tilde{w}$ 求导得 $\frac{\partial E_{\tilde{w}}}{\partial \tilde{w}}=2X^T(X\tilde{w}-y)$

令导数为零，可得 $\tilde{w}$ 的最优解：

$$2X^T(X\tilde{w}-y)=0\rightarrow X^TX\tilde{w}=X^Ty$$

由于 X^TX 与 X^Ty 已知，而 $\tilde{w}$ 为待求变量，上式可以看成 $Ax=B$ 的形式，即解线性方程组，其中 $A=X^TX$， $B=X^Ty$， $x=\tilde{w}$.

当 A 可逆时（一般用矩阵是否满秩来判断），解得

$$x=A^{-1}B\rightarrow \tilde{w}^*=(X^TX)^{-1}X^Ty=(w^*,b^*)$$

当 A 不可逆时，原方程有多组解，即 $\tilde{w}^*$ 不唯一。这样就会有很多个 $\tilde{w}^*$,他们都能使均方误差最小化，**选择哪一个 $\tilde{w}^*$ 将由算法的归纳偏好决定，常见的做法是引入正则项。**

线性回归模型形式简单、易于建模，由于 w 直观表达了各属性值在预测中的重要性，因此线性回归模型具有很好的解释性。

3 多重共线性

对于线性回归模型，一个重要的基本假设是属性之间是相互独立的。令 $g_i = (x_{1i}, x_{2i}, \ldots, x_{mi})$ ，则 $X = (g_1, g_2, \ldots, g_m, 1)$ 。

（1）对于等式 $k_1 g_1 + k_2 g_2 +, \ldots, + k_m g_m = 0$ ，若存在不全为零的 $k_1, k_2, \ldots, k_m$ 使等式成立，则称属性之间完全共线性。

（2）对于等式 $k_1 g_1 + k_2 g_2 +, \ldots, + k_m g_m = 0$ ，若 $k_1, k_2, \ldots, k_m$ 必须全部为零才能使等式成立，则称属性之间无共线性。

（3）对于等式 $k_1 g_1 + k_2 g_2 +, \ldots, + k_m g_m + v = 0$ ，若存在不全为零的 $k_1, k_2, \ldots, k_m$ 且 $v \neq 0$ 使等式成立，则称属性之间近似共线性。其中 v 为随机误差项。

其实，多重共线性的本质就是属性向量之间线性相关，在使用线性回归模型的时候，我们往往需要引入多个属性，这样就很容易造成属性向量之间线性相关。通常遇到的是第（3）种情况，近似共线性。

多重共线性的存在不仅会导致模型的过拟合，而且还会导致回归模型的稳定性和准确性大大的降低。

解决多重共线性的方法有很多，例如：

（1）保留重要特征，去掉次要特征。

（2）前向逐步回归法：前向逐步回归的基本思想是将特征逐个引入模型，每引入一个特征后都要进行 **F 检验**，并对已经选入的特征逐个进行 **T 检验**，当原来引入的特征由于后面特征的引入变得不再显著时，则将其删除。以确保每次引入新的特征之前回归方程中只包含显著性属性。这是一个反复的过程，直到既没有显著的特征选入回归方程，也没有不显著的特征从回归方程中剔除为止。以保证最后所得到的特征集最优。

（3）主成分分析：主成分分析对于一般的多重共线性问题还是适用的，尤其是对共线性较强的变量之间。当采取主成分提取了新的特征后，往往这些变量间的组内差异小而组间差异大，起到了消除共线性的问题，常见的方法有 **PCA 降维**。

（4）Lasso 回归：Lasso 回归就是在线性回归中引入 L1 正则，它不仅可以解决过拟合问题，而且可以在参数缩减过程中，将一些重复的参数直接缩减为零，带来**稀疏特征**。这可以达到提取有效特征的作用。但是Lasso回归的计算过程复杂，毕竟一范数不是连续可导的。

4 Linear Regression 实战

使用的数据与上一章相同，我们读取 user_balance_table.csv 文件，将2014-04-01~2014-07-31 的数据作为训练集，将 2014-08-01~2014-08-10 的数据作为测试集。

```
import pandas as pd


def generate_purchase_seq():
    dateparse = lambda dates: pd.datetime.strptime(dates, '%Y%m%d')
    user_balance = pd.read_csv('./user_balance_table.csv', parse_dates=['report_date']
                               index_col='report_date', date_parser=dateparse)

    df = user_balance.groupby(['report_date'])['total_purchase_amt'].sum()
    purchase_seq = pd.Series(df, name='value')

    purchase_seq_train = purchase_seq['2014-04-01':'2014-07-31']
    purchase_seq_test = purchase_seq['2014-08-01':'2014-08-10']

    purchase_seq_train.to_csv(path='./purchase_seq_train.csv', header=True)
    purchase_seq_test.to_csv(path='./purchase_seq_test.csv', header=True)


generate_purchase_seq()
```

建立线性回归模型的步骤可以分为**特征提取**和**模型拟合**两个部分。其实在很多机器学习建模的过程中，特征选择是非常重要的一个环节，即需要选择哪些属性列来拟合标签。借用一句机器学习中比较流行的话：**特征决定了模型的上界，对于模型的参数调优就是不断地逼近这个上界。**

这里我们选取了 31 个特征，如下：

（1）一周七天，正常周一到周五上班，周六周日休息：

workday1：工作日1、 workday2：工作日2、 workday3：工作日3 workday4：工作日4、 workday5：工作日5、 weekend1：周末1、 weekend2：周末2、 two_days_holiday：两天休假

（2）小长假影响，休息三天：

holiday1：休假第一天、 holiday2：休假第二天、 holiday3：休假第三天
complement_workday：小长假补班、 three_days_holiday：三天休假

（3）月初、月中、月末：

begin_month：月初、 mid_month：月中、 end_month：月末

（4）假期的影响：

before_the_holiday：放假前的最后一个工作日、 after_the_holiday：放假后的第一个工作日、workday：工作日、 holiday：休息日

after_last_workday_num_holidays_byte1\after_last_workday_num_holidays_byte2： 放假前的最后一个工作日，在这一天后，放几天假。这里采用二进制表示。例如（byte1、byte2）取（0、1）表示一天假，（1、0）表示两天假，（1、1）表示三天假。

before_first_workday_num_holidays_byte1\before_first_workday_num_holidays_byte2： 放假后的第一个工作日，在这一天前，放了几天假。表示方法同上。

（5）新股的影响：

new_share：新股申购日、 before_1_new_share：新股申购前一天、before_2_new_share：新股申购前两天、 before_3_new_share：新股申购前三天、after_1_new_share：新股申购后一天、 after_2_new_share：新股申购后两天、after_3_new_share：新股申购后三天

```
import pandas as pd

 # 新股申购日期   新股数量
    # 06.18         4
    # 06.19         1
    # 06.20         1
    # 06.23         2
    # 06.24         1
    # 07.10         1
    # 07.23         5
    # 07.24         4
    # 07.30         1
    # 08.13         1
    # 08.14         1

def generate_common_feature(purchase_seq):
    report_date = purchase_seq.index
    weekday = []
    for i in range(len(report_date)):
        weekday.append(report_date[i].weekday())

    purchase_seq['weekday'] = weekday
    purchase_seq['workday1'] = 0
    purchase_seq['workday2'] = 0
    purchase_seq['workday3'] = 0
    purchase_seq['workday4'] = 0
    purchase_seq['workday5'] = 0
    purchase_seq['weekend1'] = 0
```

```
purchase_seq['weekend2'] = 0

purchase_seq['holiday1'] = 0
purchase_seq['holiday2'] = 0
purchase_seq['holiday3'] = 0

purchase_seq['before_the_holiday'] = 0
purchase_seq['after_the_holiday'] = 0

purchase_seq['workday'] = 0
purchase_seq['holiday'] = 0

purchase_seq['begin_month'] = 0
purchase_seq['mid_month'] = 0
purchase_seq['end_month'] = 0

purchase_seq['after_last_workday_num_holidays_byte1'] = 0
purchase_seq['after_last_workday_num_holidays_byte2'] = 0

purchase_seq['before_first_workday_num_holidays_byte1'] = 0
purchase_seq['before_first_workday_num_holidays_byte2'] = 0

purchase_seq['two_days_holiday'] = 0
purchase_seq['three_days_holiday'] = 0
purchase_seq['complement_workday'] = 0

purchase_seq['before_1_new_share'] = 0
purchase_seq['before_2_new_share'] = 0
purchase_seq['before_3_new_share'] = 0
purchase_seq['new_share'] = 0
purchase_seq['after_1_new_share'] = 0
purchase_seq['after_2_new_share'] = 0
purchase_seq['after_3_new_share'] = 0

#  常规周处理
for index, row in purchase_seq.iterrows():
    if (index.day < 11):
        row['begin_month'] = 1
    elif (11 <= index.day < 21):
        row['mid_month'] = 1
    else:
        row['end_month'] = 1

    if row['weekday'] == 0:
        row['workday1'] = 1
```

```
            row['after_the_holiday'] = 1
            row['workday'] = 1
            row['before_first_workday_num_holidays_byte1'] = 1
            row['before_first_workday_num_holidays_byte2'] = 0
        elif row['weekday'] == 1:
            row['workday2'] = 1
            row['workday'] = 1
        elif row['weekday'] == 2:
            row['workday3'] = 1
            row['workday'] = 1
        elif row['weekday'] == 3:
            row['workday4'] = 1
            row['workday'] = 1
        elif row['weekday'] == 4:
            row['workday5'] = 1
            row['before_the_holiday'] = 1
            row['workday'] = 1
            row['after_last_workday_num_holidays_byte1'] = 1
            row['after_last_workday_num_holidays_byte2'] = 0
        elif row['weekday'] == 5:
            row['weekend1'] = 1
            row['holiday1'] = 1
            row['holiday'] = 1
            row['two_days_holiday'] = 1
        elif row['weekday'] == 6:
            row['weekend2'] = 1
            row['holiday2'] = 1
            row['holiday'] = 1
            row['two_days_holiday'] = 1

    return purchase_seq


def generate_train_feature(purchase_seq):

    # 新股处理
    purchase_seq['before_1_new_share'][purchase_seq.index == '2014-06-17'] = 1
    purchase_seq['before_2_new_share'][purchase_seq.index == '2014-06-16'] = 1
    purchase_seq['before_3_new_share'][purchase_seq.index == '2014-06-15'] = 1
    purchase_seq['new_share'][purchase_seq.index == '2014-06-18'] = 1
    purchase_seq['after_1_new_share'][purchase_seq.index == '2014-06-19'] = 1
    purchase_seq['after_2_new_share'][purchase_seq.index == '2014-06-20'] = 1
    purchase_seq['after_3_new_share'][purchase_seq.index == '2014-06-21'] = 1

    purchase_seq['before_1_new_share'][purchase_seq.index == '2014-06-18'] = 1
```

```
purchase_seq['before_2_new_share'][purchase_seq.index == '2014-06-17'] = 1
purchase_seq['before_3_new_share'][purchase_seq.index == '2014-06-16'] = 1
purchase_seq['new_share'][purchase_seq.index == '2014-06-19'] = 1
purchase_seq['after_1_new_share'][purchase_seq.index == '2014-06-20'] = 1
purchase_seq['after_2_new_share'][purchase_seq.index == '2014-06-21'] = 1
purchase_seq['after_3_new_share'][purchase_seq.index == '2014-06-22'] = 1

purchase_seq['before_1_new_share'][purchase_seq.index == '2014-06-19'] = 1
purchase_seq['before_2_new_share'][purchase_seq.index == '2014-06-18'] = 1
purchase_seq['before_3_new_share'][purchase_seq.index == '2014-06-17'] = 1
purchase_seq['new_share'][purchase_seq.index == '2014-06-20'] = 1
purchase_seq['after_1_new_share'][purchase_seq.index == '2014-06-21'] = 1
purchase_seq['after_2_new_share'][purchase_seq.index == '2014-06-22'] = 1
purchase_seq['after_3_new_share'][purchase_seq.index == '2014-06-23'] = 1

purchase_seq['before_1_new_share'][purchase_seq.index == '2014-06-22'] = 1
purchase_seq['before_2_new_share'][purchase_seq.index == '2014-06-21'] = 1
purchase_seq['before_3_new_share'][purchase_seq.index == '2014-06-20'] = 1
purchase_seq['new_share'][purchase_seq.index == '2014-06-23'] = 1
purchase_seq['after_1_new_share'][purchase_seq.index == '2014-06-24'] = 1
purchase_seq['after_2_new_share'][purchase_seq.index == '2014-06-25'] = 1
purchase_seq['after_3_new_share'][purchase_seq.index == '2014-06-26'] = 1

purchase_seq['before_1_new_share'][purchase_seq.index == '2014-06-23'] = 1
purchase_seq['before_2_new_share'][purchase_seq.index == '2014-06-22'] = 1
purchase_seq['before_3_new_share'][purchase_seq.index == '2014-06-21'] = 1
purchase_seq['new_share'][purchase_seq.index == '2014-06-24'] = 1
purchase_seq['after_1_new_share'][purchase_seq.index == '2014-06-25'] = 1
purchase_seq['after_2_new_share'][purchase_seq.index == '2014-06-26'] = 1
purchase_seq['after_3_new_share'][purchase_seq.index == '2014-06-27'] = 1

purchase_seq['before_1_new_share'][purchase_seq.index == '2014-07-09'] = 1
purchase_seq['before_2_new_share'][purchase_seq.index == '2014-07-08'] = 1
purchase_seq['before_3_new_share'][purchase_seq.index == '2014-07-07'] = 1
purchase_seq['new_share'][purchase_seq.index == '2014-07-10'] = 1
purchase_seq['after_1_new_share'][purchase_seq.index == '2014-07-11'] = 1
purchase_seq['after_2_new_share'][purchase_seq.index == '2014-07-12'] = 1
purchase_seq['after_3_new_share'][purchase_seq.index == '2014-07-13'] = 1

purchase_seq['before_1_new_share'][purchase_seq.index == '2014-07-22'] = 1
purchase_seq['before_2_new_share'][purchase_seq.index == '2014-07-21'] = 1
purchase_seq['before_3_new_share'][purchase_seq.index == '2014-07-20'] = 1
purchase_seq['new_share'][purchase_seq.index == '2014-07-23'] = 1
purchase_seq['after_1_new_share'][purchase_seq.index == '2014-07-24'] = 1
purchase_seq['after_2_new_share'][purchase_seq.index == '2014-07-25'] = 1
```

```
purchase_seq['after_3_new_share'][purchase_seq.index == '2014-07-26'] = 1

purchase_seq['before_1_new_share'][purchase_seq.index == '2014-07-23'] = 1
purchase_seq['before_2_new_share'][purchase_seq.index == '2014-07-22'] = 1
purchase_seq['before_3_new_share'][purchase_seq.index == '2014-07-21'] = 1
purchase_seq['new_share'][purchase_seq.index == '2014-07-24'] = 1
purchase_seq['after_1_new_share'][purchase_seq.index == '2014-07-25'] = 1
purchase_seq['after_2_new_share'][purchase_seq.index == '2014-07-26'] = 1
purchase_seq['after_3_new_share'][purchase_seq.index == '2014-07-27'] = 1

purchase_seq['before_1_new_share'][purchase_seq.index == '2014-07-29'] = 1
purchase_seq['before_2_new_share'][purchase_seq.index == '2014-07-28'] = 1
purchase_seq['before_3_new_share'][purchase_seq.index == '2014-07-27'] = 1
purchase_seq['new_share'][purchase_seq.index == '2014-07-30'] = 1
purchase_seq['after_1_new_share'][purchase_seq.index == '2014-07-31'] = 1

# 节假日处理
# 清明
purchase_seq['after_last_workday_num_holidays_byte1'][purchase_seq.index == '2014-
purchase_seq['after_last_workday_num_holidays_byte2'][purchase_seq.index == '2014-

purchase_seq['two_days_holiday'][purchase_seq.index == '2014-04-05'] = 0
purchase_seq['three_days_holiday'][purchase_seq.index == '2014-04-05'] = 1

purchase_seq['two_days_holiday'][purchase_seq.index == '2014-04-06'] = 0
purchase_seq['three_days_holiday'][purchase_seq.index == '2014-04-06'] = 1

purchase_seq['workday1'][purchase_seq.index == '2014-04-07'] = 0
purchase_seq['after_the_holiday'][purchase_seq.index == '2014-04-07'] = 0
purchase_seq['workday'][purchase_seq.index == '2014-04-07'] = 0
purchase_seq['before_first_workday_num_holidays_byte1'][purchase_seq.index == '201
purchase_seq['before_first_workday_num_holidays_byte2'][purchase_seq.index == '201
purchase_seq['holiday3'][purchase_seq.index == '2014-04-07'] = 1
purchase_seq['holiday'][purchase_seq.index == '2014-04-07'] = 1
purchase_seq['three_days_holiday'][purchase_seq.index == '2014-04-07'] = 1

purchase_seq['after_the_holiday'][purchase_seq.index == '2014-04-08'] = 1
purchase_seq['before_first_workday_num_holidays_byte1'][purchase_seq.index == '201
purchase_seq['before_first_workday_num_holidays_byte2'][purchase_seq.index == '201

# 五一
purchase_seq['before_the_holiday'][purchase_seq.index == '2014-04-30'] = 1
purchase_seq['after_last_workday_num_holidays_byte1'][purchase_seq.index == '2014-
purchase_seq['after_last_workday_num_holidays_byte1'][purchase_seq.index == '2014-
```

```
purchase_seq['workday4'][purchase_seq.index == '2014-05-01'] = 0
purchase_seq['workday'][purchase_seq.index == '2014-05-01'] = 0
purchase_seq['holiday1'][purchase_seq.index == '2014-05-01'] = 1
purchase_seq['holiday'][purchase_seq.index == '2014-05-01'] = 1
purchase_seq['three_days_holiday'][purchase_seq.index == '2014-05-01'] = 1

purchase_seq['workday5'][purchase_seq.index == '2014-05-02'] = 0
purchase_seq['before_the_holiday'][purchase_seq.index == '2014-05-02'] = 0
purchase_seq['workday'][purchase_seq.index == '2014-05-02'] = 0
purchase_seq['after_last_workday_num_holidays_byte1'][purchase_seq.index == '2014-
purchase_seq['after_last_workday_num_holidays_byte2'][purchase_seq.index == '2014-
purchase_seq['holiday2'][purchase_seq.index == '2014-05-02'] = 1
purchase_seq['holiday'][purchase_seq.index == '2014-05-02'] = 1
purchase_seq['three_days_holiday'][purchase_seq.index == '2014-05-02'] = 1

purchase_seq['holiday1'][purchase_seq.index == '2014-05-03'] = 0
purchase_seq['holiday3'][purchase_seq.index == '2014-05-03'] = 1
purchase_seq['three_days_holiday'][purchase_seq.index == '2014-05-03'] = 1

purchase_seq['weekend2'][purchase_seq.index == '2014-05-04'] = 0
purchase_seq['holiday2'][purchase_seq.index == '2014-05-04'] = 0
purchase_seq['holiday'][purchase_seq.index == '2014-05-04'] = 0
purchase_seq['two_days_holiday'][purchase_seq.index == '2014-05-04'] = 0
purchase_seq['after_the_holiday'][purchase_seq.index == '2014-05-04'] = 1
purchase_seq['workday'][purchase_seq.index == '2014-05-04'] = 1
purchase_seq['before_first_workday_num_holidays_byte1'][purchase_seq.index == '201
purchase_seq['before_first_workday_num_holidays_byte2'][purchase_seq.index == '201
purchase_seq['complement_workday'][purchase_seq.index == '2014-05-04'] = 1

# 端午
purchase_seq['after_last_workday_num_holidays_byte1'][purchase_seq.index == '2014-
purchase_seq['after_last_workday_num_holidays_byte2'][purchase_seq.index == '2014-

purchase_seq['two_days_holiday'][purchase_seq.index == '2014-05-31'] = 0
purchase_seq['three_days_holiday'][purchase_seq.index == '2014-05-31'] = 1

purchase_seq['two_days_holiday'][purchase_seq.index == '2014-06-01'] = 0
purchase_seq['three_days_holiday'][purchase_seq.index == '2014-06-01'] = 1

purchase_seq['workday1'][purchase_seq.index == '2014-06-02'] = 0
purchase_seq['after_the_holiday'][purchase_seq.index == '2014-06-02'] = 0
purchase_seq['workday'][purchase_seq.index == '2014-06-02'] = 0
purchase_seq['before_first_workday_num_holidays_byte1'][purchase_seq.index == '201
purchase_seq['before_first_workday_num_holidays_byte2'][purchase_seq.index == '201
purchase_seq['holiday3'][purchase_seq.index == '2014-06-02'] = 1
```

```
    purchase_seq['holiday'][purchase_seq.index == '2014-06-02'] = 1
    purchase_seq['three_days_holiday'][purchase_seq.index == '2014-06-02'] = 1

    purchase_seq['after_the_holiday'][purchase_seq.index == '2014-06-03'] = 1
    purchase_seq['before_first_workday_num_holidays_byte1'][purchase_seq.index == '201
    purchase_seq['before_first_workday_num_holidays_byte2'][purchase_seq.index == '201

    return purchase_seq


def generate_test_feature(purchase_seq):
    # 新股处理
    purchase_seq['after_2_new_share'][purchase_seq.index == '2014-08-01'] = 1
    purchase_seq['after_3_new_share'][purchase_seq.index == '2014-08-02'] = 1
    purchase_seq['before_3_new_share'][purchase_seq.index == '2014-08-10'] = 1

    # 节假日处理
    return purchase_seq


dateparse = lambda dates: pd.datetime.strptime(dates, '%Y-%m-%d')
purchase_seq_train = pd.read_csv('./purchase_seq_train.csv', parse_dates=['report_date
                                 index_col='report_date', date_parser=dateparse)

purchase_seq_test = pd.read_csv('./purchase_seq_test.csv', parse_dates=['report_date']
                                index_col='report_date', date_parser=dateparse)

purchase_seq_train = generate_train_feature(generate_common_feature(purchase_seq_train
purchase_seq_test = generate_test_feature(generate_common_feature(purchase_seq_test))
```

特征提取部分的代码较多，这里做如下解释：首先将数据作为常规周处理，然后分别对训练集和测试集中的小长假、新股申购做处理（新股申购日期代码中已经给出），其中训练集中包含了清明、五一、端午三个假期，而测试集中没有。我们直接用这31个特征训练模型，然后查看拟合效果和预测效果。**在模型评价中引入残差散点图和 R2 评分指标。**所谓残差散点图，横轴是日期，纵轴是模型预测值与真实值的差，这样可以更加直观的看到模型的效果。R2 评分指标：

$$R^2 = 1 - \frac{SS_{res}}{SS_{tot}}, \quad SS_{res} = \sum (y_i - f_i)^2, \quad SS_{tot} = \sum (y_i - \bar{y})^2$$

其中 y_i 表示真实值， f_i 表示预测值， $\bar{y}$ 表示样本 y_i 的均值； SS_{res} 表示真实值与预测值的残差平方和， SS_{tot} 表示真实值与其平均值的残差的平方和。**其实 R2 评分指标，就是将拟合模型与数据的“均值模型”相比较，当模型效果与取数据均值一样时，R2 为 0，通常情况下拟合模型的效果肯定比取均值要好，所以 SS_{res} 比 SS_{tot} 要小，结果在 0~1 之间，R2越接近 0，**

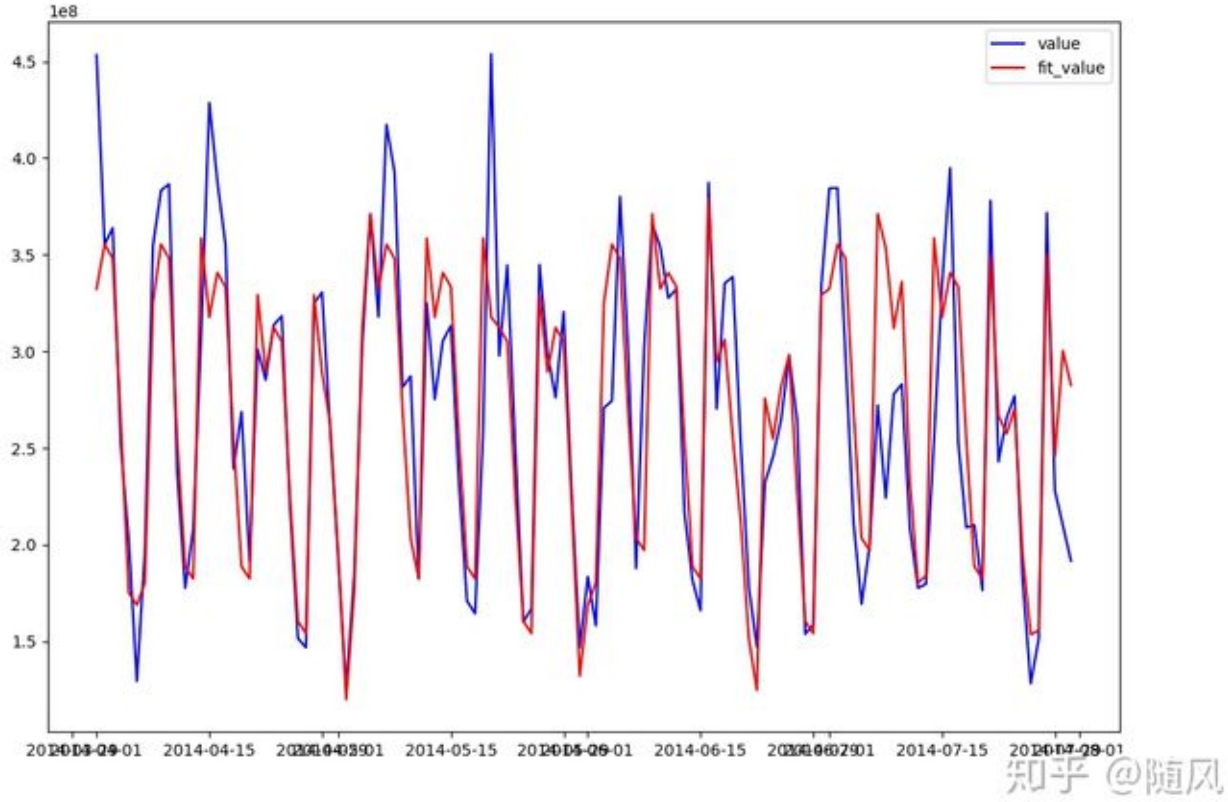
模型效果越差，R2 越接近 1，模型效果越好。如果 R2 为负数，说明拟合模型还不如均值模型，很有可能是因为数据不存在任何线性关系。

```
import matplotlib.pyplot as plt
import pandas as pd
from sklearn.linear_model import LinearRegression


dateparse = lambda dates: pd.datetime.strptime(dates, '%Y-%m-%d')
purchase_seq_train = pd.read_csv('./purchase_seq_train.csv', parse_dates=['report_date
                                 index_col='report_date', date_parser=dateparse)

purchase_seq_test = pd.read_csv('./purchase_seq_test.csv', parse_dates=['report_date']
                                index_col='report_date', date_parser=dateparse)

purchase_seq_train = generate_train_feature(generate_common_feature(purchase_seq_train
purchase_seq_test = generate_test_feature(generate_common_feature(purchase_seq_test))

feature = ['workday1', 'workday2', 'workday3', 'workday4', 'workday5', 'weekend1', 'we
           'holiday3', 'before_the_holiday', 'after_the_holiday', 'workday', 'holiday'
           'end_month', 'after_last_workday_num_holidays_byte1', 'after_last_workday_n
           'before_first_workday_num_holidays_byte1', 'before_first_workday_num_holida
           'three_days_holiday', 'complement_workday', 'before_1_new_share', 'before_2
           'new_share', 'after_1_new_share', 'after_2_new_share', 'after_3_new_share']

lr_model = LinearRegression()
lr_model.fit(purchase_seq_train[feature], purchase_seq_train['value'])
w = lr_model.coef_  # 回归系数
b = lr_model.intercept_  # 截距

print("w: ", w)
print("b: ", b)

purchase_seq_train['fit_value'] = lr_model.predict(purchase_seq_train[feature])

plt.figure(figsize=(12, 8))
plt.plot(purchase_seq_train['value'], label='value', color='blue')
plt.plot(purchase_seq_train['fit_value'], label='fit_value', color='red')
plt.legend(loc='best')
plt.show()

purchase_seq_test['fit_value'] = lr_model.predict(purchase_seq_test[feature])

plt.figure(figsize=(12, 8))
```

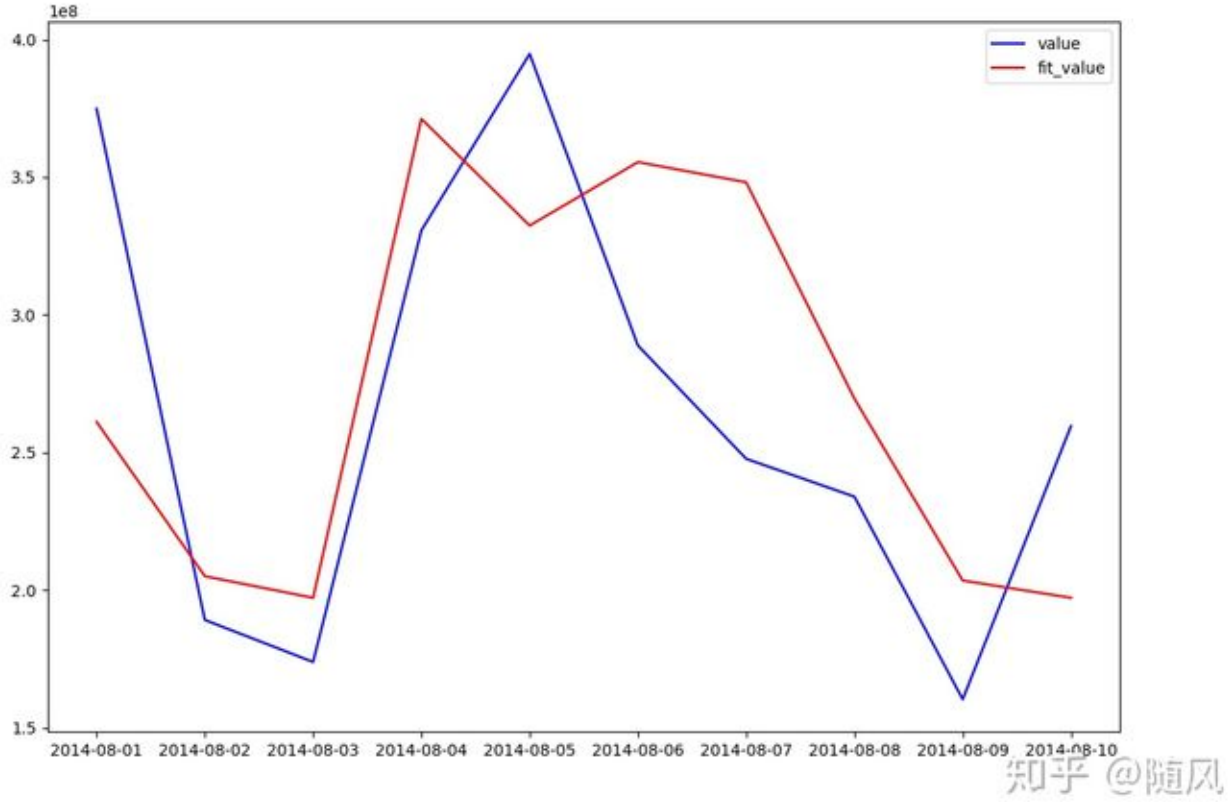
```
plt.plot(purchase_seq_test['value'], label='value', color='blue')
plt.plot(purchase_seq_test['fit_value'], label='fit_value', color='red')
plt.legend(loc='best')
plt.show()

# 残差评估方法
train_score = lr_model.score(purchase_seq_train[feature], purchase_seq_train['value'])
test_score = lr_model.score(purchase_seq_test[feature], purchase_seq_test['value'])
print("train_score: ", train_score)
print("test_score: ", test_score)

plt.scatter(purchase_seq_train.index, purchase_seq_train['fit_value'] - purchase_seq_t
            c='blue', marker='o', label='Training data')
plt.scatter(purchase_seq_test.index, purchase_seq_test['fit_value'] - purchase_seq_tes
            c='lightgreen', marker='s', label='Test data')
plt.xlabel('Date')
plt.ylabel('Residuals')
plt.legend(loc='upper left')
plt.hlines(y=0,xmin='2014-04-01', xmax='2014-08-10', lw=2, colors='red')
plt.show()
```

通过打印可以看到模型的回归系数 w 和截距 b，从 w 的值可以看出哪些特征对模型更加重要。train_score 表示模型拟合的 R2 评分，test_score 表示模型预测的 R2 评分。

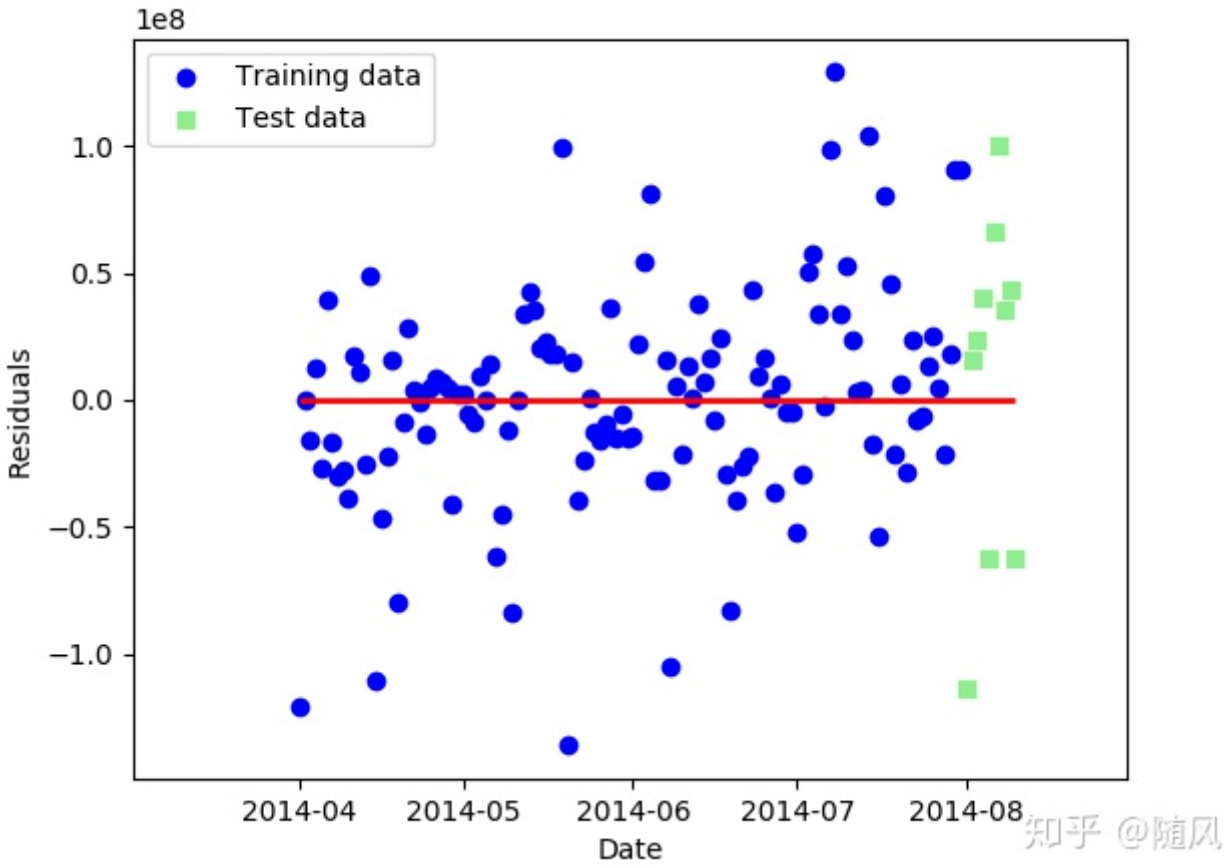
```
w:  [ 2.82057845e+22  6.10640696e+21  6.10640696e+21  6.10640696e+21
  6.10640696e+21 -2.05702014e+07  4.86649160e+07  8.72343851e+20
  8.72343851e+20  8.72343851e+20 -2.63373694e+07 -1.10496888e+22
 -2.61703155e+21  2.61703155e+21  1.93936004e+07  4.54385535e+06
 -2.37903970e+07 -2.33308160e+07 -5.07904000e+06 -1.10496888e+22
  2.20993776e+22  1.67772160e+07 -1.14032640e+07  6.10640696e+21
 -4.34176000e+07  2.07093760e+07 -2.62144000e+05 -1.18620160e+07
 -2.25443840e+07 -8.25753600e+06  1.53600000e+06]

b:  -3.4893754040412925e+21

train_score:  0.7106076316281829
test_score:  0.32084229771319894
```



模型拟合效果



模型预测效果



模型拟合和预测残差效果

从残差散点图可以看出，模型的效果并不好，尤其是 test_score ，要知道，这是我们最关心的，毕竟训练模型是为了做预测。

6 Lasso Regression

逐步回归法确实可以消除多重共线性，筛选出跟数据标签相关性更强的特征。但是在很多实际业务中，特征维度通常达到千万甚至上亿级别，带来**“维数灾难”**问题：

需要更多的样本，样本随着数据维度的增加呈指数型增长。
数据变得更稀疏，导致数据灾难。
在高维数据空间，预测将变得不再容易。
导致模型过拟合。
大量的特征难免引入多重共线性问题。

显然，在高维空间中采用逐步回归法筛选特征是不可行的（计算量太大）。**这里介绍 Lasso 回归，通过降维的方法筛选特征。该方法不仅适用于线性模型，也适用于非线性模型。Lasso 是基于惩罚（L1正则）对样本数据进行特征选择，通过对原本的系数进行压缩，将原本很小的系数直接压缩至 0，从而将这部分系数所对应的特征视为非显著性变量，将不显著的变量直接去掉：**

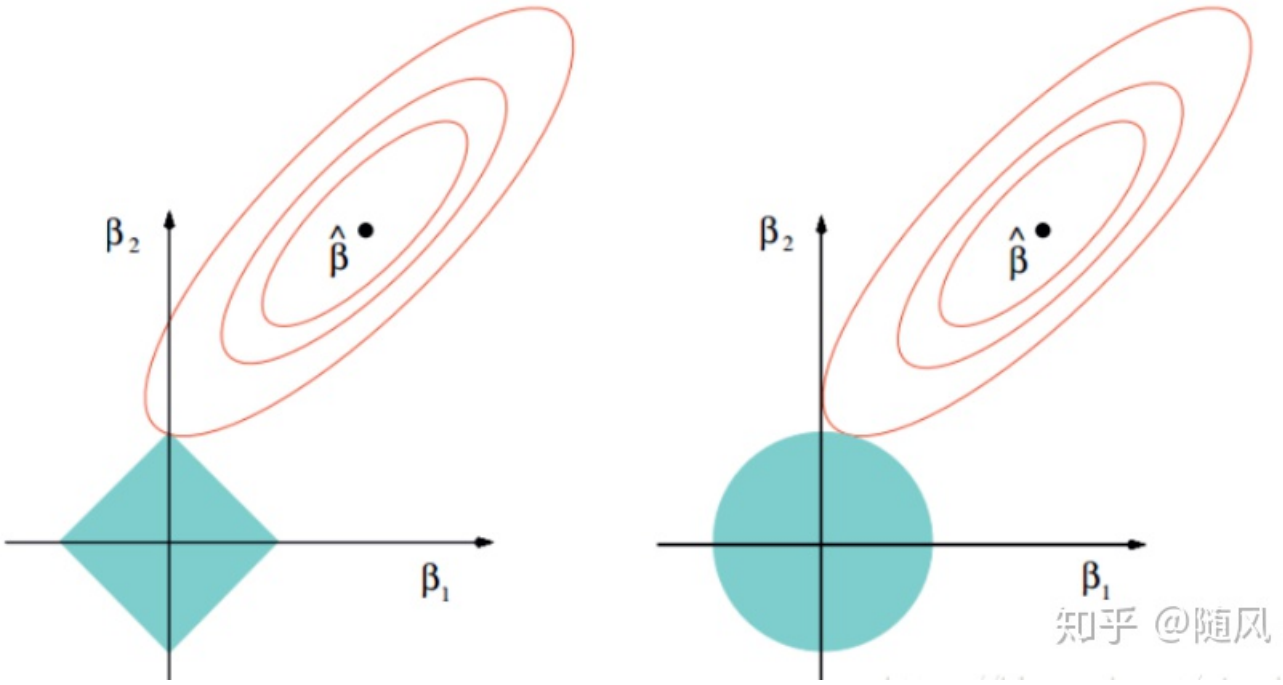
$$(w^*,b^*)=arg\min_{w,b}\Sigma_{i=1}^{m}(f(x_i)-y_i)^2+\lambda\sum_{j=1}^{d}|w_j|,\quad f(x_i)=w^Tx_i+b$$

从表达式可以看出，Lasso 回归就是在线性回归的基础上，引入了 L1 正则项 $\lambda\sum_{j=1}^{d}|w_j|$ ，如果引入 L2 正则项：

$$(w^*,b^*)=arg\min_{w,b}\Sigma_{i=1}^{m}(f(x_i)-y_i)^2+\lambda\sum_{j=1}^{d}|w_j|^2,\quad f(x_i)=w^Tx_i+b$$

称为**岭回归。在第二节我们谈到了 W 存在多组解的问题（ X^TX 不满秩），在第三节谈到了多重共线性问题，其实本质问题是特征向量之间是否线性相关，可以看出，当 X 的秩等于它的列数（即特征向量的维数），也就是特征向量之间线性无关时， W 存在唯一解，且不存在多重共线性问题。**

Lasso 回归和岭回归都通过压缩特征的方法解决特征向量之间线性相关的问题。下图给出了两种回归在二维空间中的比较。从图中可以看出，Lasso 回归（左图）的可行解区域更加尖锐，目标函数容易与顶点相交，这样将 β_1 特征压缩为 0。岭回归（右图）的可行解区域更加平滑，对特征 β_1 确实起到了压缩作用，但是一般不会压缩到 0，而是一个很小的数值。



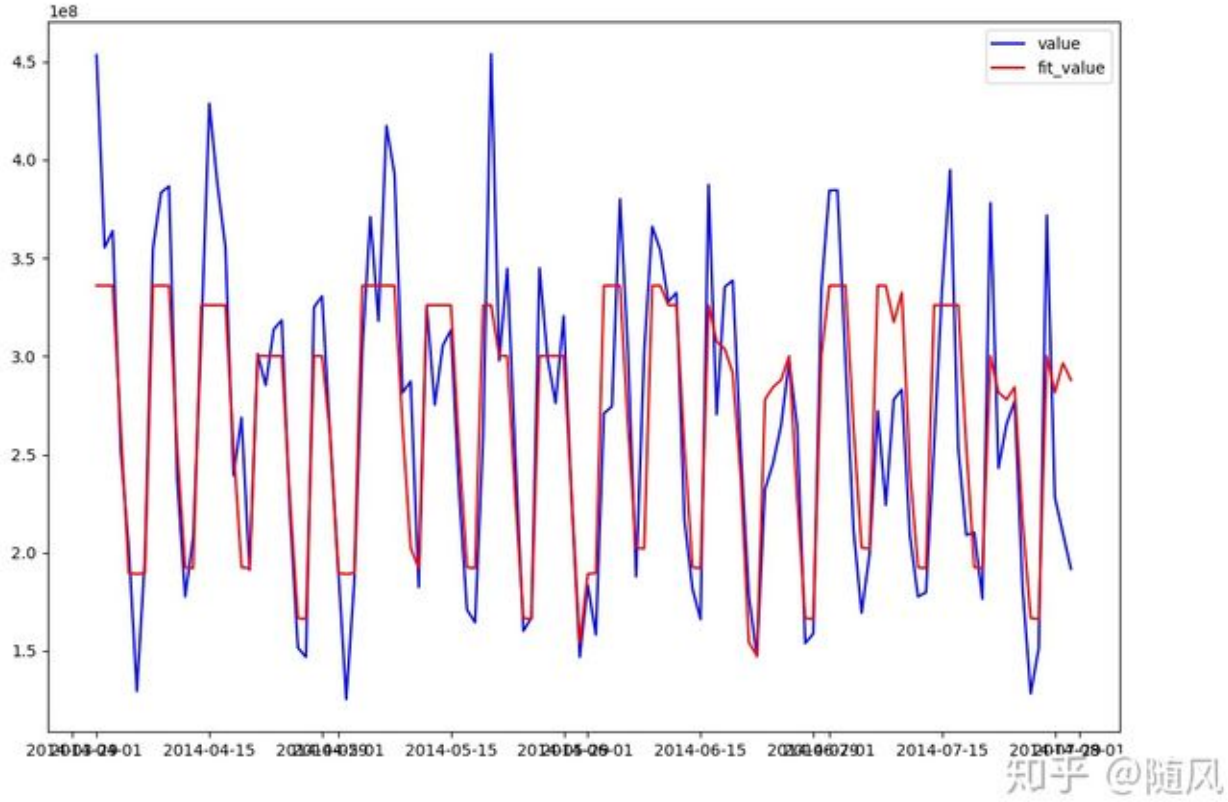
Lasso 回归中加入了超参数 λ ，该参数需要通过交叉验证法找到最优解。这里通过 LassoCV 函数进行 10 折交叉验证确定 λ （alpha），再生成 Lasso 模型。

```
import pandas as pd
import matplotlib.pyplot as plt
from sklearn.linear_model import Lasso, LassoCV

dateparse = lambda dates: pd.datetime.strptime(dates, '%Y-%m-%d')
purchase_seq_train = pd.read_csv('./purchase_seq_train.csv', parse_dates=['report_date
                                 index_col='report_date', date_parser=dateparse)

purchase_seq_test = pd.read_csv('./purchase_seq_test.csv', parse_dates=['report_date']
                                index_col='report_date', date_parser=dateparse)

purchase_seq_train = generate_train_feature(generate_common_feature(purchase_seq_train
purchase_seq_test = generate_test_feature(generate_common_feature(purchase_seq_test))

feature = ['workday1', 'workday2', 'workday3', 'workday4', 'workday5', 'weekend1', 'we
           'holiday3', 'before_the_holiday', 'after_the_holiday', 'workday', 'holiday'
           'end_month', 'after_last_workday_num_holidays_byte1', 'after_last_workday_n
           'before_first_workday_num_holidays_byte1', 'before_first_workday_num_holida
           'three_days_holiday', 'complement_workday', 'before_1_new_share', 'before_2
           'new_share', 'after_1_new_share', 'after_2_new_share', 'after_3_new_share']

lassocv_model = LassoCV(cv=10).fit(purchase_seq_train[feature], purchase_seq_train['va
alpha = lassocv_model.alpha_
print('best alpha: ', alpha)

lr_model = Lasso(max_iter=1000, alpha=alpha)  # 调节alpha可以实现对拟合的程度
lr_model.fit(purchase_seq_train[feature], purchase_seq_train['value'])  # 线性回归建模

w = lr_model.coef_  # 回归系数
b = lr_model.intercept_  # 截距

print("w: ", w)
print("b: ", b)
# 使用模型预测
purchase_seq_train['fit_value'] = lr_model.predict(purchase_seq_train[feature])
purchase_seq_test['fit_value'] = lr_model.predict(purchase_seq_test[feature])

plt.figure(figsize=(12, 8))
plt.plot(purchase_seq_train['value'], label='value', color='blue')
plt.plot(purchase_seq_train['fit_value'], label='fit_value', color='red')
plt.legend(loc='best')
plt.show()

plt.figure(figsize=(12, 8))
```

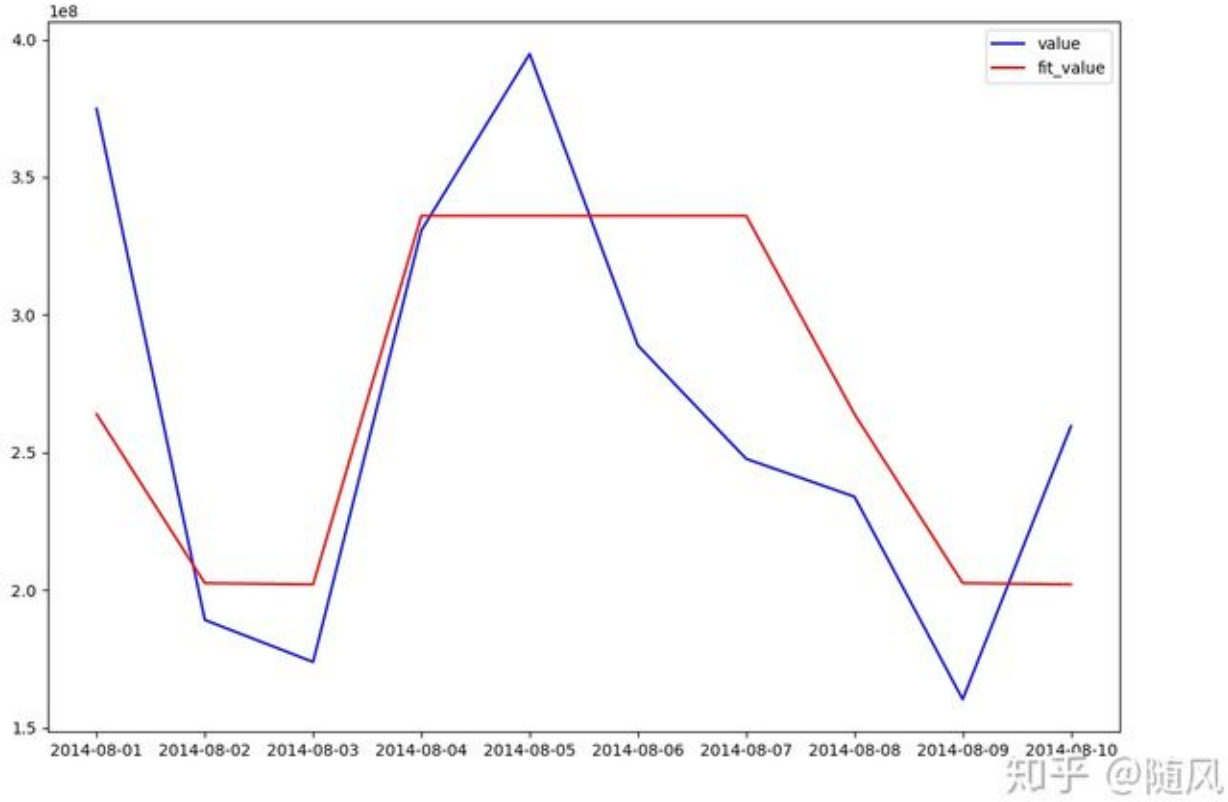
```
plt.plot(purchase_seq_test['value'], label='value', color='blue')
plt.plot(purchase_seq_test['fit_value'], label='fit_value', color='red')
plt.legend(loc='best')
plt.show()

# 残差评估方法
train_score = lr_model.score(purchase_seq_train[feature], purchase_seq_train['value'])
test_score = lr_model.score(purchase_seq_test[feature], purchase_seq_test['value'])
print("train_score: ", train_score)
print("test_score: ", test_score)

plt.scatter(purchase_seq_train.index, purchase_seq_train['fit_value'] - purchase_seq_t
            c='blue', marker='o', label='Training data')
plt.scatter(purchase_seq_test.index, purchase_seq_test['fit_value'] - purchase_seq_tes
            c='lightgreen', marker='s', label='Test data')
plt.xlabel('Date')
plt.ylabel('Residuals')
plt.legend(loc='upper left')
plt.hlines(y=0, xmin='2014-04-01', xmax='2014-08-10', lw=2, colors='red')
plt.show()
```

从 W 的结果可以看出，Lasso 回归把一些特征去除了（回归系数为0，特征不再起作用），train_score 低于线性回归中 train_score，而 test_score 高于线性回归中 test_score，这说明将所有特征引入模型产生了过拟合。

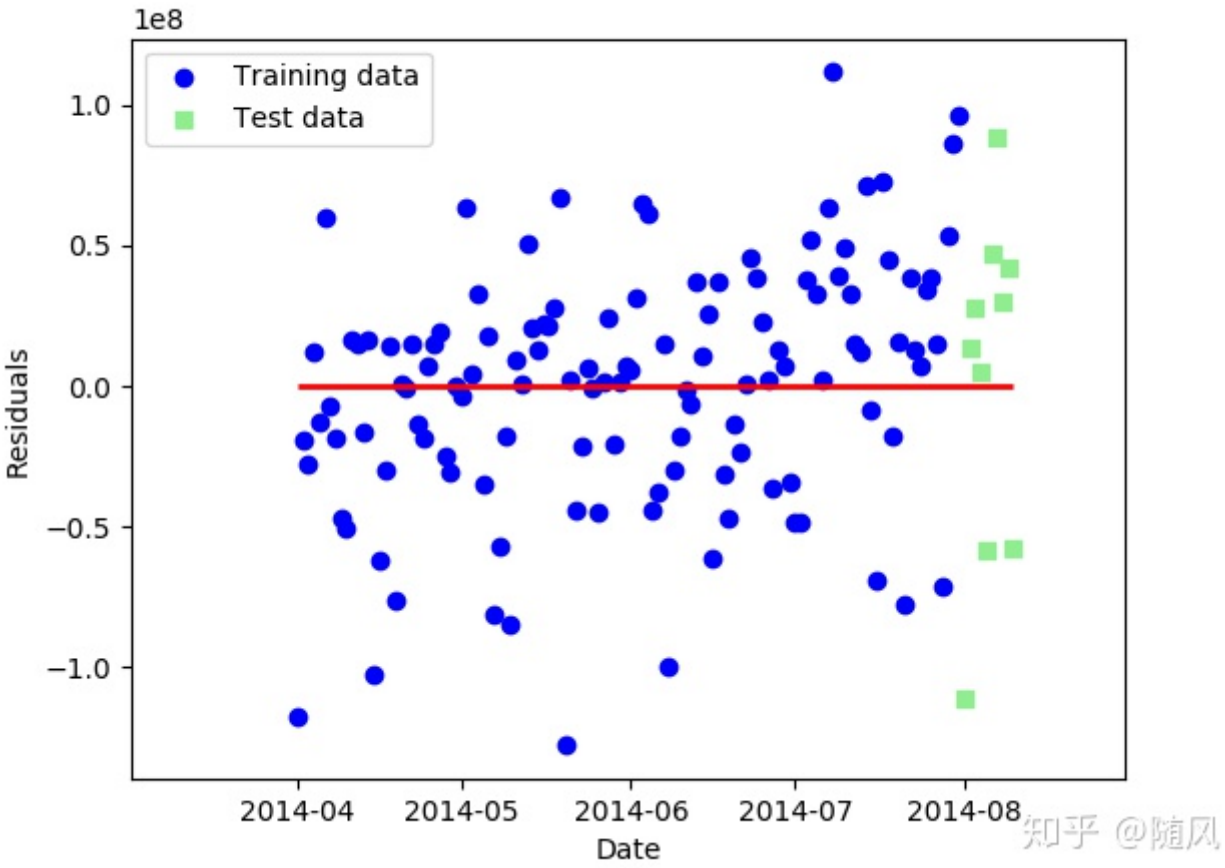
```
w:  [ 0.00000000e+00  0.00000000e+00  0.00000000e+00  0.00000000e+00
 -3.18522930e+07  0.00000000e+00 -0.00000000e+00  0.00000000e+00
 -4.77494980e+05 -0.00000000e+00 -0.00000000e+00  0.00000000e+00
  1.33477647e+08 -5.43359353e-08  9.95055577e+06  0.00000000e+00
 -2.58682931e+07 -4.02168365e+07 -0.00000000e+00  0.00000000e+00
 -0.00000000e+00 -0.00000000e+00 -1.28881002e+07 -0.00000000e+00
 -1.86251686e+07  0.00000000e+00  0.00000000e+00 -3.55143402e+06
 -1.22922712e+07 -0.00000000e+00 -0.00000000e+00]

b:  192537717.3047737

train_score:  0.7032902242462391
test_score:  0.454659055041409
```



模型拟合效果



模型预测效果



模型拟合和预测残差效果

从 R2 评分来看，采用 Lasso 回归的效果好于线性回归，这是因为通过引入 L1 正则，消除了一些不必要的特征，此外，采用 L1 正则也能够减少计算量，提升效率。从残差散点图来看，Lasso 回归的效果依然不理想。

7 结语

如果说 ARIMA 模型是从时间序列的深度去建模（通过自相关性，研究时间序列过去与未来的关系），那么线性回归模型就是从空间的广度去建模（研究每一个时刻，该时刻的特征与取值的关系）。从结果来看，采用线性回归的效果并不理想，这可能是因为选取的特征不好，这里说一下特征与标签的关系：

特征与标签没有任何关系，这样的特征是无效的，不适用于任何模型。
特征与标签有线性关系，这样的特征适用于线性模型。
特征与标签有非线性关系（比如平方关系），这样的特征适用于非线性模型。

其实，由于实际业务的复杂性，对于时间序列建模可能不是单一模型可以解决的。即使是采用线性回归模型，像蚂蚁金服每日资金申购情况，需要更多的特征进行筛选，比如国家金融政策、股票期货市场。纸上得来终觉浅，绝知此事要躬行。