# 软件技术基础大作业——聚丙烯熔融指数预测

## 一、 问题背景

### 1. 聚丙烯熔融指数预测

数据来源：聚丙烯生产过程熔融指数数据，共 150 组时序数据，见 excel 表格。问题描述：聚丙烯生产过程中的熔融指数是一个重要质量控制指标（excel 表格中的变量 y），决定了所生产产品的牌号与价格，但是难以在线测量，从而导致生产质量的控制品质大大降低。因此，采用生产中与该质量控制指标相关的可直接测量的操作变量（excel 表格中的变量 x，共 9 个），来在线预测该质量控制指标，从而提高生产质量控制品质。

## 二、 实验任务

根据所给的数据集，建立一个模型，从而能够预测聚丙烯熔指指数指标。

## 三、 设计过程

初步拿到数据，发现有 9 个输入变量，1 个输出变量。题目要求是希望能够在线预测质量控制指标，从而提高生产质量控制品质。所以初步判断这是一个多元的回归问题。根据所学到的知识以及所查阅的资料，我首先的想法是通过使用 sklearn 进行数据集训练与模型建立。同时利用 pandas 库和 numpy 库对数据进行操作。

### 1. 数据处理

题目所提供数据为 xls 格式，如下图所示：

| 序号 | x1 | x2 | x3 | x4 | x5 | x6 | x7 | x8 | x9 | | y |
|---|---|---|---|---|---|---|---|---|---|---|---|
| 1 | 2.18626 | -0.98362 | -0.48902 | 2.655448 | -2.81004 | 0.602695 | 1.307641 | 2.701685 | 0.172023 | | 2.65 |
| 2 | 2.304016 | -0.97855 | -0.49578 | 2.628263 | -3.05133 | 0.607532 | 1.320871 | 2.62639 | 0.351214 | | 2.673333 |
| 3 | 2.421772 | -0.97347 | -0.50254 | 2.601077 | -3.29262 | 0.612369 | 1.3341 | 2.551095 | 0.530405 | | 2.696667 |
| 4 | 2.539528 | -0.9684 | -0.5093 | 2.573892 | -3.53391 | 0.617206 | 1.34733 | 2.4758 | 0.709596 | | 2.72 |
| 5 | 2.127382 | -1.11048 | -0.50254 | 2.336022 | -3.43456 | 0.615824 | 1.35111 | 1.55075 | -0.58954 | | 2.716667 |
| 6 | 1.715235 | -1.25256 | -0.49578 | 2.098152 | -3.3352 | 0.614442 | 1.354889 | 0.6257 | -1.88867 | | 2.713333 |
| 7 | 1.303089 | -1.39465 | -0.48902 | 1.860281 | -3.23585 | 0.61306 | 1.358669 | -0.29935 | -3.18781 | | 2.71 |
| 8 | 1.108791 | -1.3497 | -0.47564 | 1.860281 | -2.98036 | 0.61306 | 1.33599 | 0.082502 | -2.33665 | | 2.72 |
| 9 | 0.914494 | -1.30476 | -0.46226 | 1.860281 | -2.72488 | 0.61306 | 1.313311 | 0.464354 | -1.48549 | | 2.73 |
| 10 | 0.720196 | -1.25981 | -0.44889 | 1.860281 | -2.4694 | 0.61306 | 1.290632 | 0.846206 | -0.63434 | | 2.74 |
| 11 | 0.84384 | -1.27359 | -0.44621 | 1.139874 | -2.34165 | 0.610296 | 1.264173 | 0.926879 | -0.36555 | | 2.703333 |
| 12 | 0.967484 | -1.28736 | -0.44354 | 0.419467 | -2.21391 | 0.607532 | 1.237714 | 1.007552 | -0.09676 | | 2.666667 |
| 13 | 1.091128 | -1.30113 | -0.44086 | -0.30094 | -2.08617 | 0.604768 | 1.211255 | 1.088225 | 0.172023 | | 2.63 |
| 14 | 1.026362 | -1.32433 | -0.43889 | -0.53201 | -1.83069 | 0.604768 | 1.211255 | 1.10436 | 0.216821 | | 2.6 |
| 15 | 0.961596 | -1.34753 | -0.43692 | -0.76309 | -1.5752 | 0.604768 | 1.211255 | 1.120494 | 0.261619 | | 2.57 |
| 16 | 0.89683 | -1.37073 | -0.43495 | -0.99416 | -1.31972 | 0.604768 | 1.211255 | 1.136629 | 0.306417 | | 2.54 |
| 17 | 0.802626 | -1.3468 | -0.44818 | -0.94659 | -0.93649 | 0.606841 | 1.164007 | 1.120494 | 0.306417 | | 2.546667 |
| 18 | 0.708421 | -1.32288 | -0.46142 | -0.89901 | -0.55327 | 0.608914 | 1.116758 | 1.10436 | 0.306417 | | 2.553333 |
| 19 | 0.614216 | -1.29896 | -0.47465 | -0.85144 | -0.17004 | 0.610987 | 1.06951 | 1.088225 | 0.306417 | | 2.56 |
| 20 | 0.649543 | -1.08149 | -0.46353 | -0.77668 | -0.01391 | 0.610987 | 1.060061 | 1.373269 | 0.754394 | | 2.556667 |
| 21 | 0.684869 | -0.86401 | -0.45241 | -0.70192 | 0.14222 | 0.610987 | 1.050611 | 1.658314 | 1.202371 | | 2.553333 |
| 22 | 0.720196 | -0.64654 | -0.44128 | -0.62716 | 0.298349 | 0.610987 | 1.041161 | 1.943359 | 1.650349 | | 2.55 |
| 23 | 0.873279 | -0.54432 | -0.43297 | -0.64075 | 0.411897 | 0.612369 | 1.031712 | 1.803525 | 1.471158 | | 2.543333 |
| 24 | 1.026362 | -0.44211 | -0.42467 | -0.65435 | 0.525446 | 0.613751 | 1.022262 | 1.663692 | 1.291967 | | 2.536667 |
| 25 | 1.179445 | -0.3399 | -0.41636 | -0.66794 | 0.638994 | 0.615133 | 1.012812 | 1.523859 | 1.112776 | | 2.53 |
| 26 | 1.150006 | -0.41311 | -0.41284 | -0.45046 | 0.553833 | 0.613751 | 0.995803 | 1.136629 | 0.620001 | | 2.526667 |

图 1: 原始数据

为了方便 python 操作，将数据第一列删除，'y' 列与左边靠拢，并保存为 csv 格式。

调用 corr 函数，查看各个输入变量 x 与 y 之间的相关系数，并通过 seaborn 库绘制可视化相关程度的热图如下：


图 2: 输入输出相关性

可以看出输入变量 x1-x9 与输出变量 y 的相关性普遍比较低，最高的也只是 x3，x4，x5，x6 这几个变量。

于是我决定先选用 x3，x4，x5，x6 这四个输入作为训练输入，由于无法确定该问题是线性回归还是非线性回归，于是我首先尝试了线性回归。

先利用 train_test_split 函数将数据集划分为训练集和测试集

```
X_train2, X_test2, Y_train2, Y_test2 = train_test_split(X_Poly, y, train_size=0.80,
    random_state=1)
```

这里将 150 个数据随机分成 120 个训练集 X_train1，Y_train1 和 30 个测试集 X_test1，Y_test1。

随后利用 scikit-learn 框架中的 LinearRegression 模型中，使用 fit 函数进行训练，最后得到对应的线性回归方程。得到模型结剧 a，回归系数 b。

```
model = LinearRegression()
model.fit(X_train, Y_train)
print("截距: ",LinearR.intercept_)
print("系数: ",LinearR.coef_)
```

得到结果如下：

```
截距：2.4513199017988554
系数：[-0.0391533 -0.03018913 -0.03880207 0.05311798 -0.00549697 0.01419835
0.05236998 0.01524657 0.0052771 ]
```

```
y = -0.0391533*x1 + -0.03018913*x2 + -0.03880207*x3 + 0.05311798*x4 + -0.00549697*x5 +
    0.01419835*x6 + 0.05236998*x7 + 0.01524657*x8 + 0.0052771*x9
```

在得到结果后，通过模型的 predict 函数对数据进行了预测，并且使用模型的 mean_squared_error 函数和 R2 检测 r2_score 函数进行了模型评分。

```
Y_pred1 = LinearR.predict(X_test1)
# 平均误差相对于样本真实值均值偏差
print("mse相对于测试值平均值的偏差：", np.sqrt(MSE(Y_test1, Y_pred1))/Y_test1.mean())
# R^2检测
print("r2检测：", r2_score(y_true=Y_test1, y_pred=Y_pred1))
```

得到结果如下：MSE 相对于测试值平均值的偏差：0.03954403176473497

$R^2$ 检测结果为：0.2770634859513812

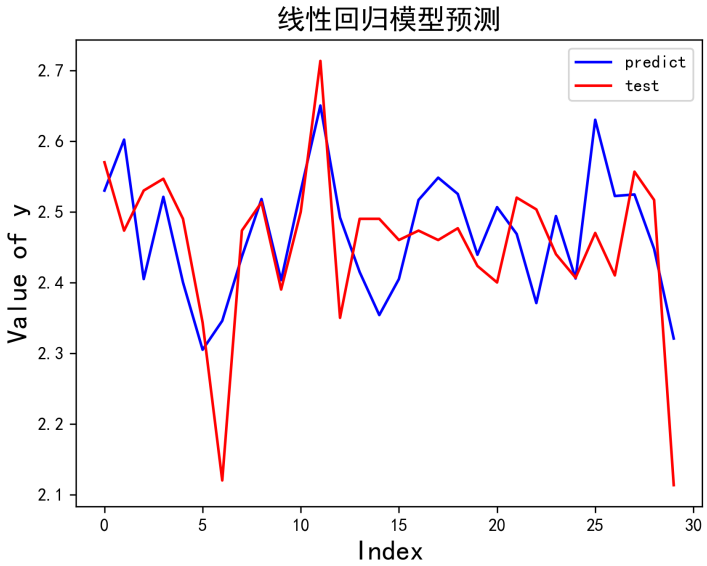


图 3: 预测和测试输出比较

从结果可以看出，训练得到的线性模型性能并不长非常的理想。于是我又尝试用非线性回归模型进行预测。此处我先尝试了用二次多项式回归模型。

首先需要给原来的一次特征 x1 x9 添加二次特征，此处用到了 sklearn 库里的 PolynomialFeatures 函数。并利用 fit_transfrom 对特征进行预处理（标准化、归一化等）。

```
PolyF = PolynomialFeatures(degree=2, interaction_only=False, include_bias=False)
X_Poly = PolyF.fit_transform(x)
```

然后同样对数据集进行分割，并使用 LinearRegression 模型进行训练，得到结果如下：

```
X_train2, X_test2, Y_train2, Y_test2 = train_test_split(
    X_Poly, y, train_size=0.80, random_state=1)
LinearR2 = LinearRegression().fit(X_train2, Y_train2)
print("截距：",LinearR2.intercept_)
print("系数：",LinearR2.coef_)
```

```
截距：  2.573018518031015
系数：  [ 0.04056469 -0.16426964 -0.46139441 -0.00801811 0.32999633 -0.64890163
 -0.43351057 -0.13721673 0.09905394 -0.0620823 -0.00969554 0.16997955
  0.0299021 -0.00500646 0.09598441 0.02758844 -0.02453116 -0.00945735
  0.00353089 0.77368019 -0.00336322 0.21904659 0.55840549 0.04788288
  0.05353425 -0.01597955 -0.06088869 0.0826872 -0.82603222 -0.19639567
 -0.79049372 -0.08128735 0.134005 -0.00179539 -0.00387439 0.0046151
 -0.03845884 0.10717514 -0.0532185 0.03376222 -0.28864873 -0.19171991
  0.01174273 -0.01913421 0.00768826 0.46659538 -0.13289949 0.09817754
 -0.01131589 -0.02820713 0.00396911 0.01439626 0.07039836 -0.04216297]
```

按照同样的方法进行预测并评估模型性能：

```
Y_pred2 = LinearR2.predict(X_test2)
# 平均误差相对于样本真实值均值偏差
print("mse相对于测试值平均值的偏差：", np.sqrt(MSE(Y_test2, Y_pred2))/Y_test2.mean())
# R^2检测
print("r2检测：", r2_score(y_true=Y_test2, y_pred=Y_pred2))
```

结果如下：

mse 相对于测试值平均值的偏差：0.011169084653174191

r2 检测：0.9423268817732235

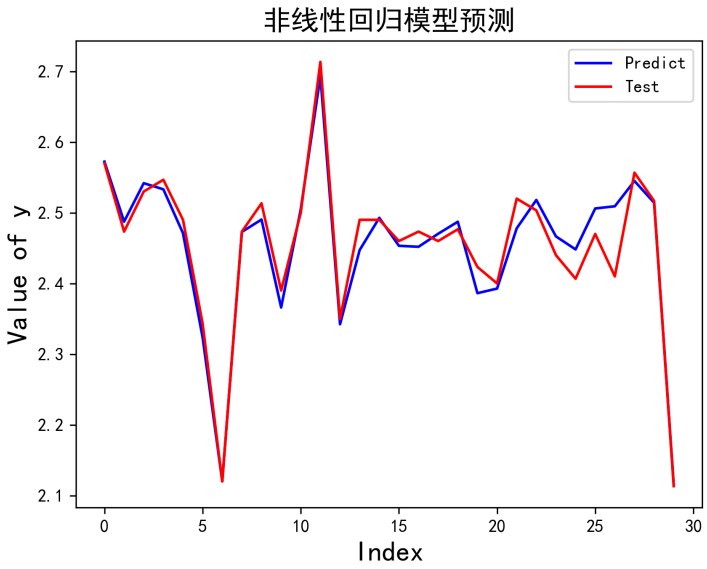


图 4: 预测和测试输出比较

可以看到，利用二次多项式回归模型的性能达到了比较好的水平。

## 四、 实验代码

```
# -*- coding: utf-8 -*-
# 导入库
import pandas as pd
```

```
import numpy as np
import matplotlib.pyplot as plt
import seaborn as sns
from sklearn.model_selection import train_test_split
from sklearn.linear_model import LinearRegression
from sklearn.preprocessing import PolynomialFeatures
from sklearn.metrics import r2_score, mean_squared_error as MSE

#解决图片输出中文显示问题
plt.rcParams['font.sans-serif'] = ['SimHei'] # 指定默认字体
plt.rcParams['axes.unicode_minus'] = False # 解决保存图像是负号'-'显示为方块的问题

# 读入数据
df = pd.read_csv("data.csv")
x = df.iloc[:,:9]
y = df['y']

# 绘制相关性热图
corrmat = df.corr()
f, ax = plt.subplots(figsize=(20, 9))
p1 = sns.heatmap(corrmat, vmax=0.8, square=True, annot = True)
ax.set_title('Heat Map')
s1 = p1.get_figure()
s1.savefig('HeatMap.jpg',dpi=300,bbox_inches='tight')

# 线性回归模型
X_train1, X_test1, Y_train1, Y_test1 = train_test_split(x, y, train_size=.80,
    random_state=1)
LinearR = LinearRegression()
LinearR.fit(X_train1, Y_train1)
print("截距：",LinearR.intercept_)
print("系数：",LinearR.coef_)
Y_pred1 = LinearR.predict(X_test1)

# 平均误差相对于样本真实值均值偏差
print("mse相对于测试值平均值的偏差：", np.sqrt(MSE(Y_test1, Y_pred1))/Y_test1.mean())
# R^2检测
print("r2检测：", r2_score(y_true=Y_test1, y_pred=Y_pred1))

# 生成对比图
plt.close()
plt.figure()
plt.plot(range(len(Y_pred1)), Y_pred1, 'b', label="predict")
plt.plot(range(len(Y_pred1)), Y_test1, 'r', label="test")
plt.legend(loc="upper right")
plt.title("线性回归模型预测",fontsize = 16)
plt.xlabel("Index",fontsize = 16)
plt.ylabel('Value of y',fontsize = 16)
plt.savefig('LinearR', dpi = 300)
plt.show()
```

```
53
54 # 非线性回归模型
55 PolyF = PolynomialFeatures(degree=2, interaction_only=False, include_bias=False)
56 X_Poly = PolyF.fit_transform(x)
57 X_train2, X_test2, Y_train2, Y_test2 = train_test_split(X_Poly, y, train_size=0.80,
       random_state=1)
58 LinearR2 = LinearRegression().fit(X_train2, Y_train2)
59 print("截距: ",LinearR2.intercept_)
60 print("系数: ",LinearR2.coef_)
61 Y_pred2 = LinearR2.predict(X_test2)
62
63 # 平均误差相对于样本真实值均值偏差
64 print("mse相对于测试值平均值的偏差: ", np.sqrt(MSE(Y_test2, Y_pred2))/Y_test2.mean())
65 # R^2检测
66 print("r2检测: ", r2_score(y_true=Y_test2, y_pred=Y_pred2))
67
68 # 生成对比图
69 plt.figure()
70 plt.plot(range(len(Y_pred2)), Y_pred2, 'b', label="Predict")
71 plt.plot(range(len(Y_pred2)), Y_test2, 'r', label="Test")
72 plt.legend(loc="upper right")
73 plt.title("非线性回归模型预测",fontsize = 16)
74 plt.xlabel("Index",fontsize = 16)
75 plt.ylabel('Value of y',fontsize = 16)
76 plt.savefig('LinearR2', dpi = 300 )
77 plt.show()
```

## 五、 实验心得与体会

(1) 本次实验最大的困难其实是如何将上课所讲的知识点实现出来。为此我查阅了大量的资料。因为之前只用过 matlab，所以一开始我想从 matlab 入手解决问题，但是当我查阅资料时发现大部分类似的问题解决方案都是用 python 语言解决的。所以我首花了一个多小时时间配置 pyhton 环境，最终我选择了利用 Anaconda，因为它方便管理和导入各种库。

(2) 在确定语言为 python 后，我对问题进行了初步的分析，决定从建立回归模型角度去解决问题，由此开始查阅资料学习 python 相关的可以进行回归分析训练的库。最后决定使用 scikit-learn 这一个库进行核心的模型建立训练，同时使用 pandas 和 mumpy 库处理数据，以及 seaborn 库分析结果和 matplotlib 库绘制图像。

(3) 本次大作业我最大的收获首先是对于课堂上所学的机器学习的内容有了更深刻的理解，对于许多概念进行了实现，不再是概念了解。其次是对于 pyhton 这门语言有了初步的掌握和理解。