

二零年秋 商务智能大作业

Regression Analysis

姓名： 王帅

学号： 2018302412

邮箱： wangshuai.excellent@foxmail.com

\mathcal{PL}

西北工业大学计算机学院

目录

1	数据集介绍	2
1.1	数据集信息	2
1.2	数据集内容	2
1.3	数据清洗与数据划分	3
1.4	数据相关程度分析	5
2	线性回归	6
2.1	算法介绍	6
2.2	算法实现	7
2.3	单变量数据之间的线性性	8
2.4	多变量数据之间的线性关系	9
3	多项式回归	10
3.1	算法介绍	10
3.2	算法实现	10
3.3	二阶多项式回归效果	11
3.4	三阶多项式回归效果	12
3.5	四阶多项式回归效果	12
3.6	多变量二阶多项式回归效果	13
4	岭回归	14
4.1	算法介绍	14
4.2	算法实现	14
4.3	岭回归误差曲线	16
4.4	岭回归效果	16
5	支持向量回归 SVR	17
5.1	算法介绍	17
5.2	支持向量分类	17
5.3	支持向量回归	18
5.4	算法流程	18
5.5	SVR 效果	19
6	总结	20
6.1	算法性能对比	20
6.2	总结与感想	22
7	Implementation	23

前言

商务智能大作业要求实现关联规则发现，聚类分析，分类，回归，链路预测相关算法中的一种，基于自己对于回归分析的兴趣，我选择回归分析作为本次大作业的研究方向，探讨不同的回归算法对于同一问题的效果，实现回归可视化效果以及误差对比，包括线性回归，多项式回归，岭回归算法，以及最后的支持向量回归 SVR。

$$Y = f(\vec{X}, \beta) + error \quad (1)$$

回归分析算法可以简单描述为：拥有观测数据 X_i ，以及目标数据 Y ，以及未知变量 β ，整个问题可以表达为上式，其中 $f(\vec{X}, \beta)$ 根据各个算法不同有不同的形式



Info: 在本次实验中，部分问题下，我把每个变量看成是随机变量，在短时间内和时间无关，也就是他们不是随机过程，而只是一个简单的随机变量，这样有助于分析问题，后面部分会详细说明。

1 数据集介绍

本次实验选择 UCI 数据集集合中的 大气质量检测数据，实现基于这些数据的回归分析

1.1 数据集信息

The dataset contains 9358 instances of hourly averaged responses from an array of 5 metal oxide chemical sensors embedded in an Air Quality Chemical Multisensor Device. The device was located on the field in a significantly polluted area, at road level, within an Italian city. Data were recorded from March 2004 to February 2005 (one year) representing the longest freely available recordings of on field deployed air quality chemical sensor devices responses. Ground Truth hourly averaged concentrations for CO, Non Metanic Hydrocarbons, Benzene, Total Nitrogen Oxides (NOx) and Nitrogen Dioxide (NO2) and were provided by a co-located reference certified analyzer. Evidences of cross-sensitivities as well as both concept and sensor drifts are present as described in De Vito et al., Sens. And Act. B, Vol. 129, 2, 2008 (citation required) eventually affecting sensors concentration estimation capabilities. Missing values are tagged with -200 value.

翻译：该数据集包含来自空气质量化学多传感器设备中的 5 个金属氧化物化学传感器阵列每小时平均响应的 9358 个实例。该设备位于意大利城市内道路污染严重的地区的田野上。从 2004 年 3 月到 2005 年 2 月（一年）记录了数据，这些数据是现场部署的空气质量化学传感器设备响应的最长免费记录。一氧化碳，非代谢碳氢化合物，苯，总氮氧化物（NOx）和二氧化氮（NO2）的地表真实小时平均浓度由共同定位的参考认证分析仪提供。如 De Vito 等人在《Sens. And Act》中所述，存在交叉敏感性以及概念漂移和传感器漂移的证据。B 卷 129,2,2008（需要引用）最终影响传感器浓度估算能力。缺少的值用-200 值标记。

1.2 数据集内容

(a) Date (DD/MM/YYYY)

- (b) Time (HH.MM.SS)
- (c) True hourly averaged concentration CO in mg/m^3 (reference analyzer)
- (d) PT08.S1 (tin oxide) hourly averaged sensor response (nominally CO targeted)
- (e) True hourly averaged overall Non Metanic HydroCarbons concentration in $microg/m^3$ (reference analyzer)
- (f) True hourly averaged Benzene concentration in $microg/m^3$ (reference analyzer)
- (g) PT08.S2 (titania) hourly averaged sensor response (nominally NMHC targeted)
- (h) True hourly averaged NOx concentration in ppb (reference analyzer)
- (i) PT08.S3 (tungsten oxide) hourly averaged sensor response (nominally NOx targeted)
- (j) True hourly averaged NO2 concentration in $microg/m^3$ (reference analyzer)
- (k) PT08.S4 (tungsten oxide) hourly averaged sensor response (nominally NO2 targeted)
- (l) PT08.S5 (indium oxide) hourly averaged sensor response (nominally O3 targeted)
- (m) Temperature in $^{\circ}C$
- (n) Relative Humidity (%)
- (o) AH Absolute Humidity

1.3 数据清洗与数据划分

阅读相关介绍，我们知道，这部分数据有脏数据，这部分数据是由于传感器故障或者其他原因，都被简单的标注为 -200，本实验的数据清洗正式将这部分有脏数据的数据条目去掉。同时将数据归为验证数据，其余为训练数据。以及最重要的，我本次是对 RH，也就是相对湿度进行回归预测，然后衡量误差范围。

统计脏数据 通过对数据的统计，我们能更加清楚的知道，数据清洗的策略，如果盲目的直接去掉包含-200 的行则会造成大量的浪费，举例来说，NMHC 丢失了几乎所有数据，应该将这些数据直接去掉

数据	丢失数据数目
CO(GT)	1683
PT08.S1(CO)	366
NMHC(GT)	8443
C6H6(GT)	366
PT08.S2(NMHC)	366
NOx(GT)	1639
PT08.S3(NOx)	366
NO2(GT)	1642
PT08.S4(NO2)	366
PT08.S5(O3)	366
T	366
RH	366
AH	366

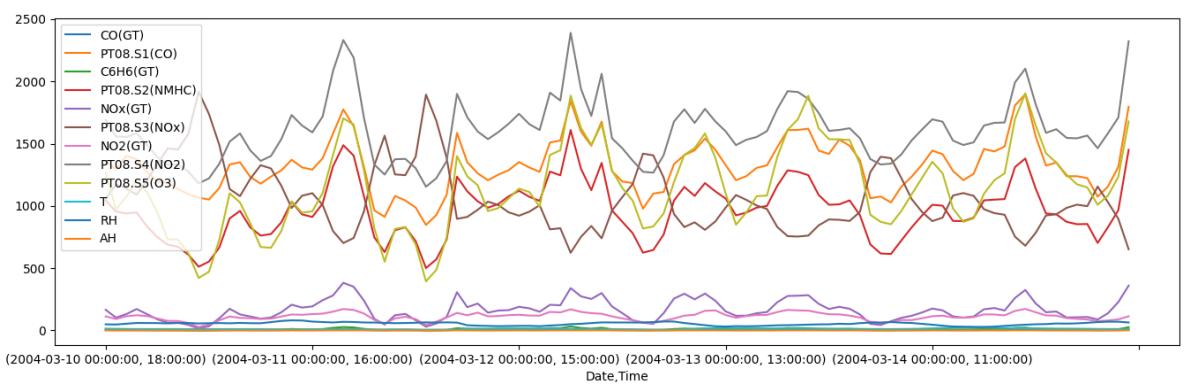


图 1: 数据可视化

属性	CO	S1	C6H6	NOx	S3	NO2	S4	S5	T	RH	AH
count	6941	6941	6941	6941	6941	6941	6941	6941	6941	6941	6941
mean	2.18	1119	10.55	250.6	816.77	113.8	1452	1057	17.75	48.88	0.98
std	1.44	218.7	7.46	208.6	251.89	47.46	353.2	406.5	8.84	17.43	0.40
min	0.1	647.2	0.18	2.0	322.0	2.0	551.0	221.0	-1.8	9.19	0.184
25%	1.1	956.2	4.98	103.0	642.2	79.0	1206	759.5	11.20	35.9	0.6922
50%	1.9	1084	8.788	186.0	785.5	110.0	1457	1006	16.8	49.17	0.95
75%	2.9	1254	14.57	334.6	946.5	142.0	1683	1322	23.29	62.22	1.25
max	11.9	2039	63.74	1479	2682	332.6	2775	2522	44.60	88.72	2.18

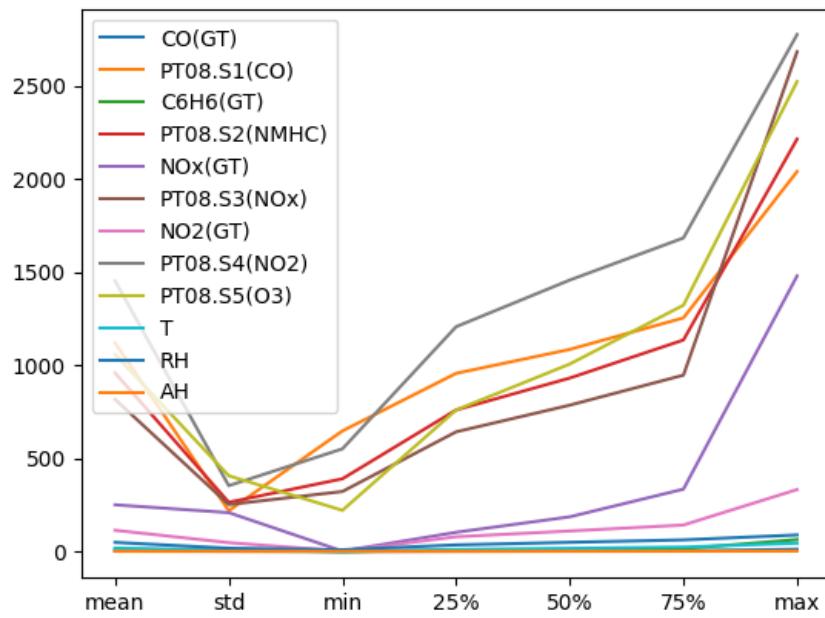


图 2: 数据属性可视化

1.4 数据相关程度分析

相关性矩阵可视化 相关性矩阵是通过计算每个变量间的协方差，然后将协方差的值可视化出来，值越大代表两个变量之间的相关程度越高

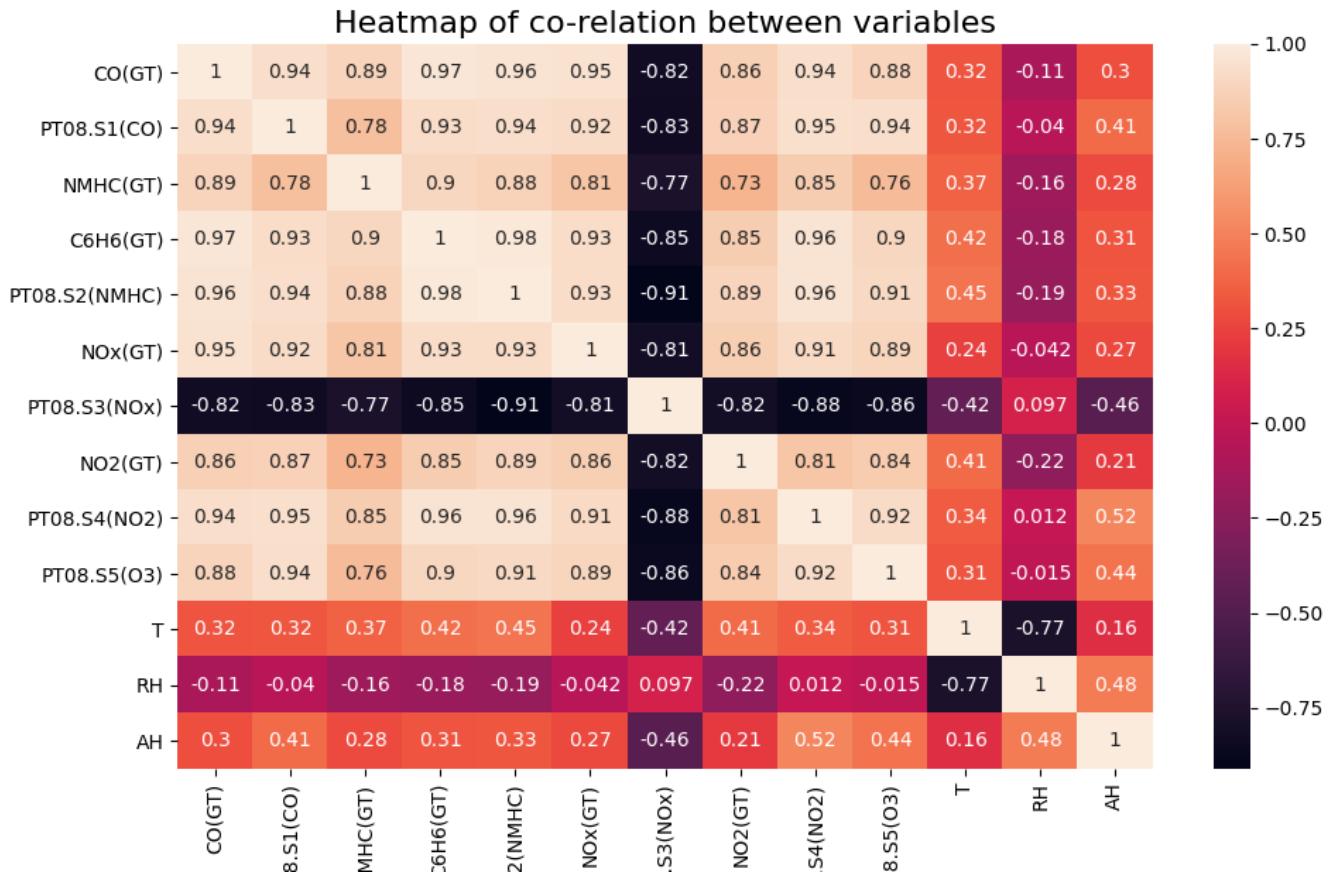


图 3: 相关性矩阵

2 线性回归

2.1 算法介绍

线性回归是利用数理统计中回归分析，来确定两种或两种以上变量间相互依赖的定量关系的一种统计分析方法，运用十分广泛。其表达形式为 $Y = W^T X + e$ ， e 为误差服从均值为 0 的正态分布。回归分析中，只包括一个自变量和一个因变量，且二者的关系可用一条直线近似表示，这种回归分析称为一元线性回归分析。如果回归分析中包括两个或两个以上的自变量，且因变量和自变量之间是线性关系，则称为多元线性回归分析。一般来说，线性回归采用最小二乘法，通过回归来确定系数。

一般地，影响 y 的因素往往不止一个，假设有 $x_1 x_2 \dots x_k$ ， k 个因素，通常可考虑如下的线性关系式：

$$y = \beta_0 + \beta_1 x_1 + \beta_2 x_2 + \dots + \beta_k x_k + \epsilon \quad (2)$$

对 y 与 $x_1 x_2 \dots x_k$ 同时作 n 次独立观察得 n 组观测值 $x_{t1} x_{t2} \dots x_{tk}$ $t = 1, 2, \dots, n$ $n > k + 1$ ，它们满足关系式：

$$y_t = \beta_0 + \beta_1 x_{t1} + \beta_2 x_{t2} + \dots + \beta_k x_{tk} + \epsilon_t \quad (3)$$

其中, ϵ_t 互不相关均是与 ϵ 同分布的随机变量。用矩阵表示上式, 可得:

$$Y = \beta X + \epsilon \quad (4)$$

利用最小二乘法可以解得:

$$\hat{\beta} = (X^T X)^{-1} X^T Y \quad (5)$$

同时就本题来说, 完全可以考虑加权, 因为有些变量方差较大, 变化过大, 可以适当考虑赋予一个权重, 其中权重和方差为反比, 则可以有:

$$\hat{\beta} = (X^T W X)^{-1} X^T W Y \dots \dots \dots (W = R^{-1}) \quad (6)$$

2.2 算法实现

❶

Info: 这部分只贴了核心代码, 第一个 `sigleX` 是单变量线性回归方案, 第二个 `mutiX` 是多变量线性回归方案, 采用 python 代码风格来表达算法流程。同时这里代码非常简短, 就没有采用注释来说明细节。

```
sigleX.py

#!/usr/bin/python
import numpy as np
def sigleX(X, Y):
    X = np.array(X)
    X = np.stack([X, np.ones(X.shape[0])], 1)
    print(X.shape)
    Y = np.array(Y)
    XTXinv=np.linalg.inv(np.matmul(X.transpose(), X))
    XTY = np.matmul(X.transpose(), Y)
    return np.matmul(XTXinv, XTY)
```

```
mutiX.py

#!/usr/bin/python
def mutiX(X, Y):
    X = np.stack([*X, np.ones(X.shape[0])], 1)
    print(X.shape)
    Y = np.array(Y)
    XTXinv=np.linalg.inv(np.matmul(X.transpose(), X))
    XTY = np.matmul(X.transpose(), Y)
    return np.matmul(XTXinv, XTY)
```

2.3 单变量数据之间的线性性

可以观察下面，线性性曲线，可以发现，温度，绝对湿度和相对湿度线性关系较为明显，下文选择这两个变量来研究问题。

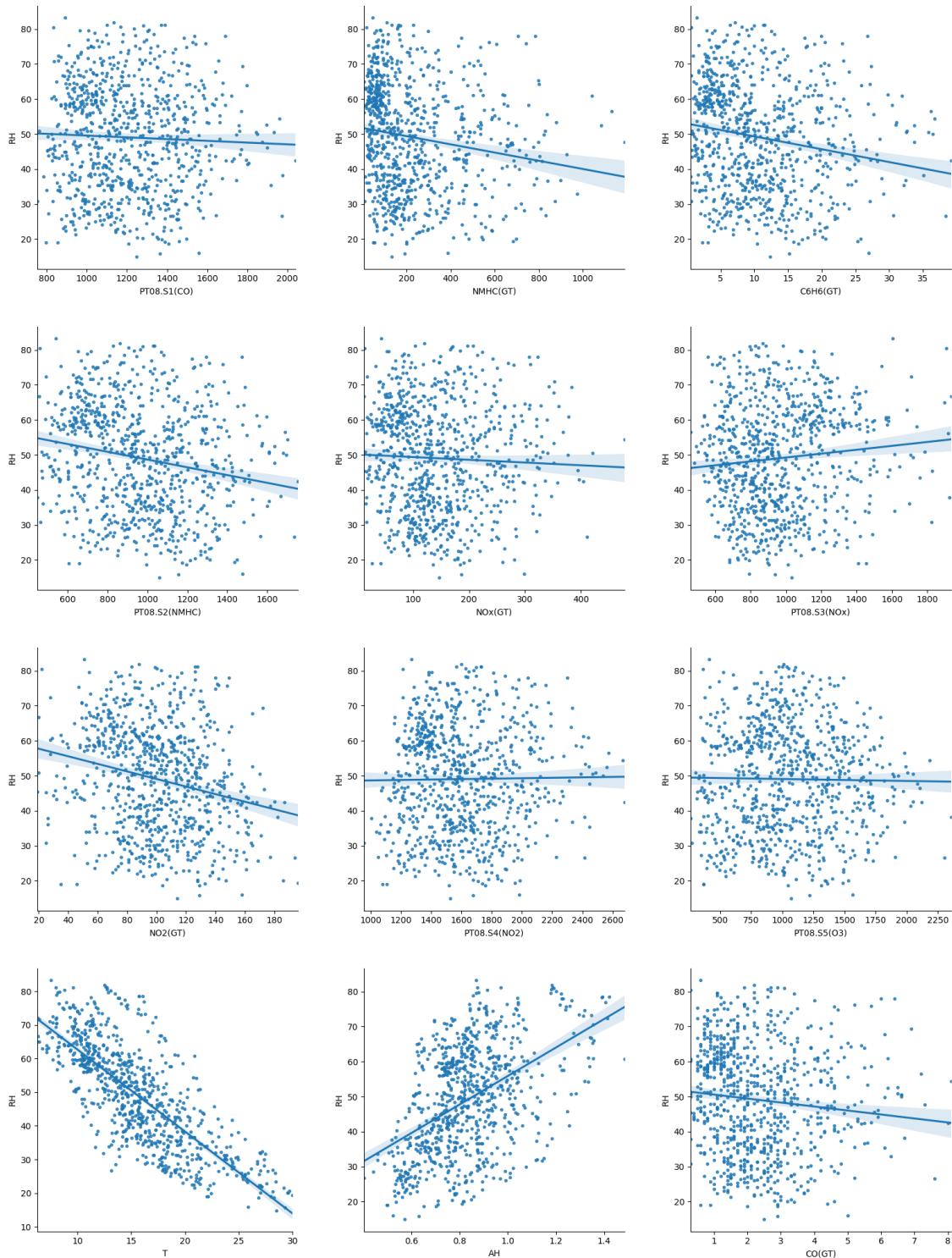


图 4: 线性性关系

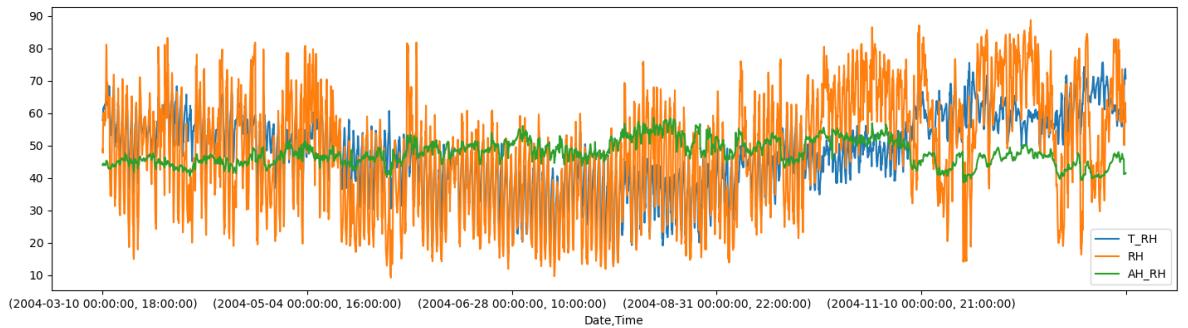


图 5: 利用单变量回归效果 (训练集)

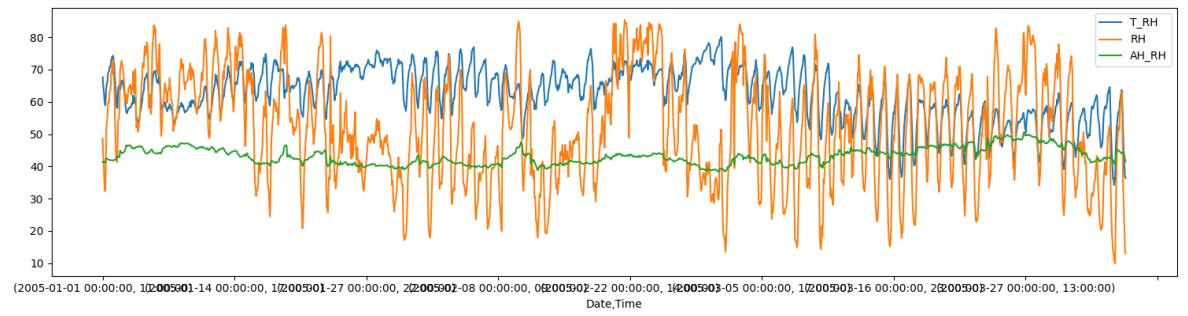


图 6: 利用单变量回归效果 (测试集)

2.4 多变量数据之间的线性关系

直观对比发现，多变量的拟合能力更强，通过其他变量的预测本领更厉害。

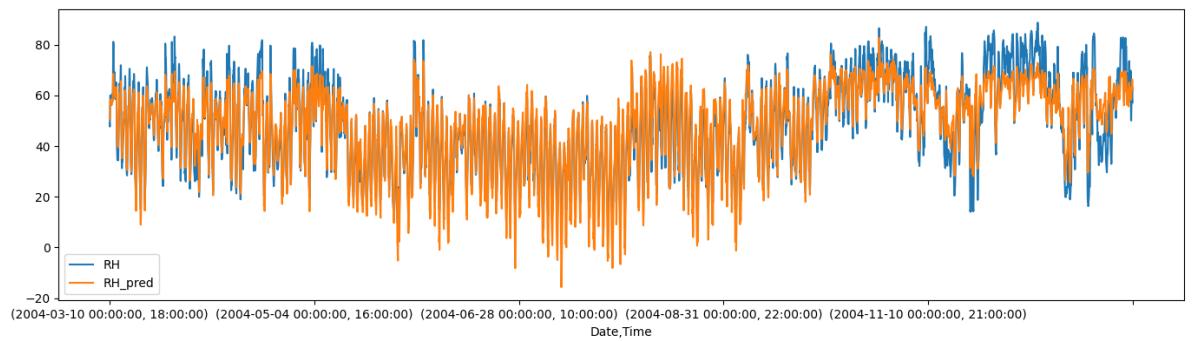


图 7: 利用多变量回归效果 (训练集)

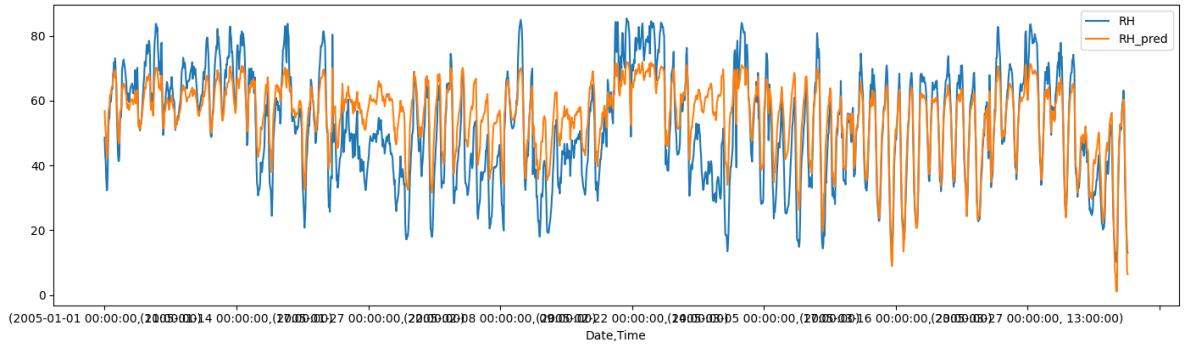


图 8: 利用多变量回归效果 (测试集)

3 多项式回归

3.1 算法介绍

多项式回归是回归分析的一种形式，其中自变量 x 和因变量 y 之间的关系被建模为关于 x 的 n 次多项式。多项式回归拟合 x 的值与 y 的相应条件均值之间的非线性关系，表示为 $E(y|x)$ ，并且已被用于描述非线性现象。虽然多项式回归是拟合数据的非线性模型，但作为统计估计问题，它是线性的。在某种意义上，回归函数 $E(y|x)$ 在从数据估计到的未知参数中是线性的。因此，多项式回归被认为是多元线性回归的特例。

多项式回归模型:

$$y_i = \beta_0 + \beta_1 x_i + \beta_2 x_i^2 + \cdots + \beta_m x_i^m + \varepsilon_i \quad (i = 1, 2, \dots, n) \quad (7)$$

可以写成矩阵形式:

$$\begin{bmatrix} y_1 \\ y_2 \\ y_3 \\ \vdots \\ y_n \end{bmatrix} = \begin{bmatrix} 1 & x_1 & x_1^2 & \cdots & x_1^m \\ 1 & x_2 & x_2^2 & \cdots & x_2^m \\ 1 & x_3 & x_3^2 & \cdots & x_3^m \\ \vdots & \vdots & \vdots & \ddots & \vdots \\ 1 & x_n & x_n^2 & \cdots & x_n^m \end{bmatrix} \begin{bmatrix} \beta_0 \\ \beta_1 \\ \beta_2 \\ \vdots \\ \beta_m \end{bmatrix} + \begin{bmatrix} \varepsilon_1 \\ \varepsilon_2 \\ \varepsilon_3 \\ \vdots \\ \varepsilon_n \end{bmatrix}, \quad (8)$$

根据线性回归的知识可以有:

$$\hat{\beta} = (\mathbf{X}^\top \mathbf{X})^{-1} \mathbf{X}^\top \vec{Y}, \quad (9)$$

3.2 算法实现

poly.py

```

#! /usr/bin/python
def polyX(X, Y, n=1):
    Keep= []
    for i in range(n+1):
        X_ = X**i
        Keep.append(X_)
    X = np.stack(Keep, axis=1)
    Y = np.array(Y)
    XTXinv=np.linalg.inv(np.matmul(X.transpose(),X))
    XTY = np.matmul(X.transpose(), Y)
    return np.matmul(XTXinv, XTY)

```

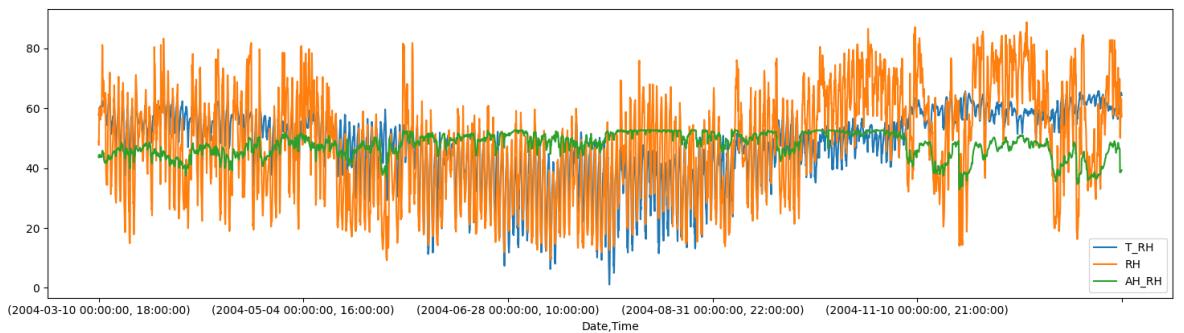
3.3 二阶多项式回归效果

图 9: 二阶多项式回归效果 (训练集)

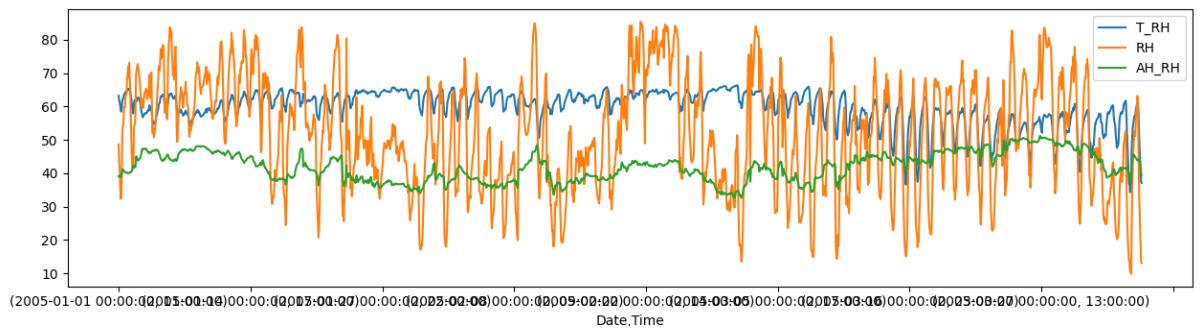


图 10: 二阶多项式回归效果 (测试集)

3.4 三阶多项式回归效果

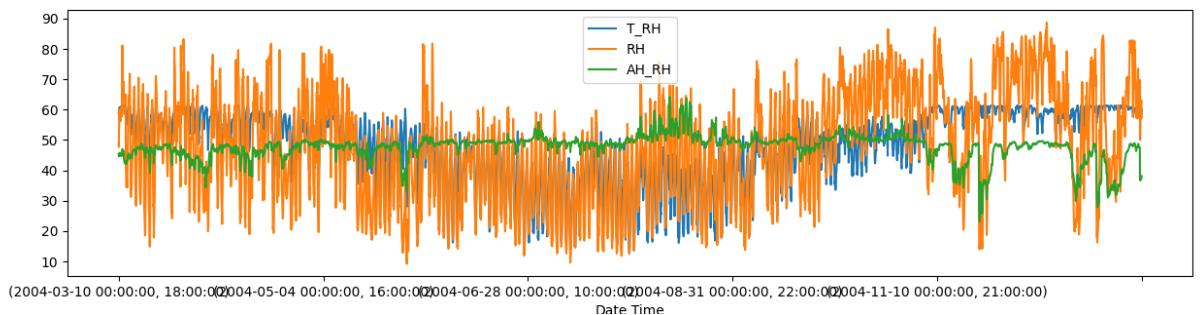


图 11: 三阶多项式回归效果 (训练集)

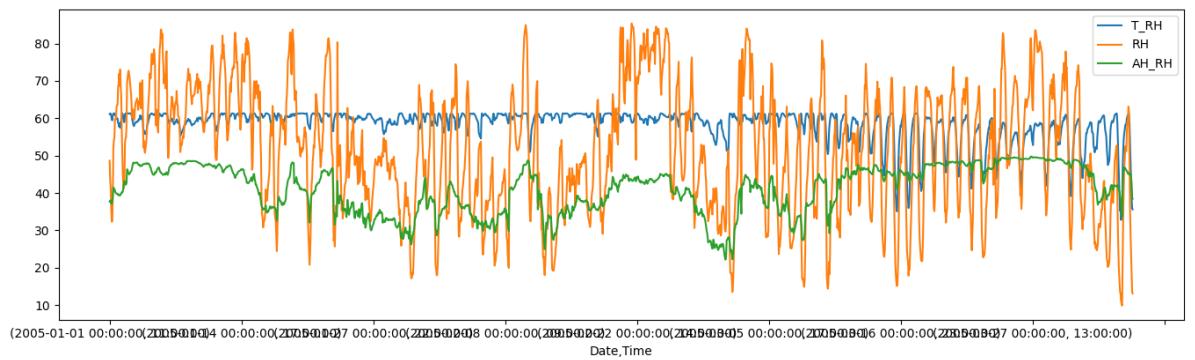


图 12: 三阶多项式回归效果 (测试集)

3.5 四阶多项式回归效果

可以看到四阶多项式已经出现了严重的震荡效果，在实际预测上，出现了肉眼可辨识的大幅度偏差

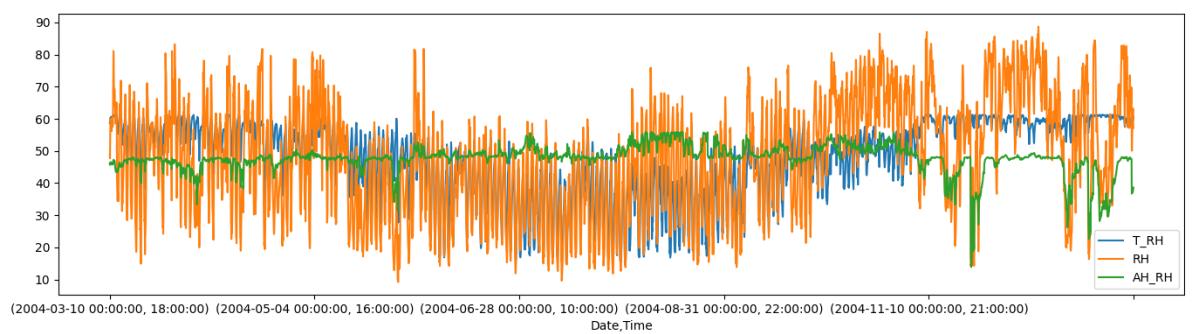


图 13: 四阶多项式回归效果 (训练集)

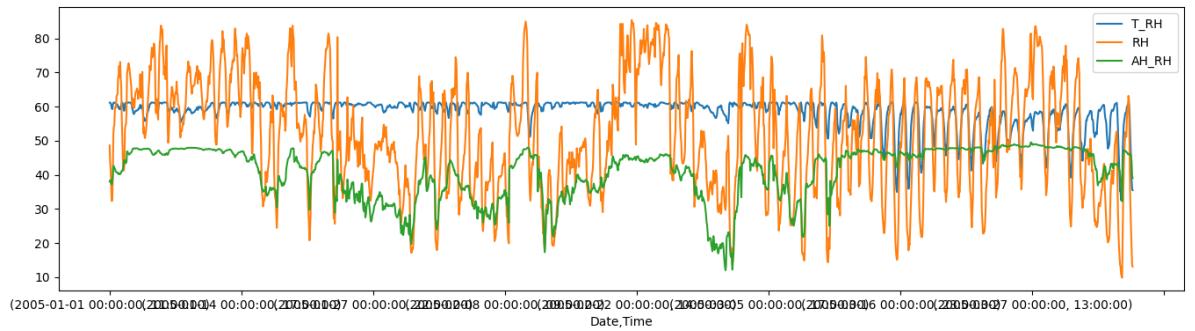


图 14: 四阶多项式回归效果 (测试集)

3.6 多变量二阶多项式回归效果

多变量多项式回归的拟合效果以及预测效果都是很不错的，但是次数过高会导致震荡的发生，因此就没有采用过高次幂的多项式多变量回归。

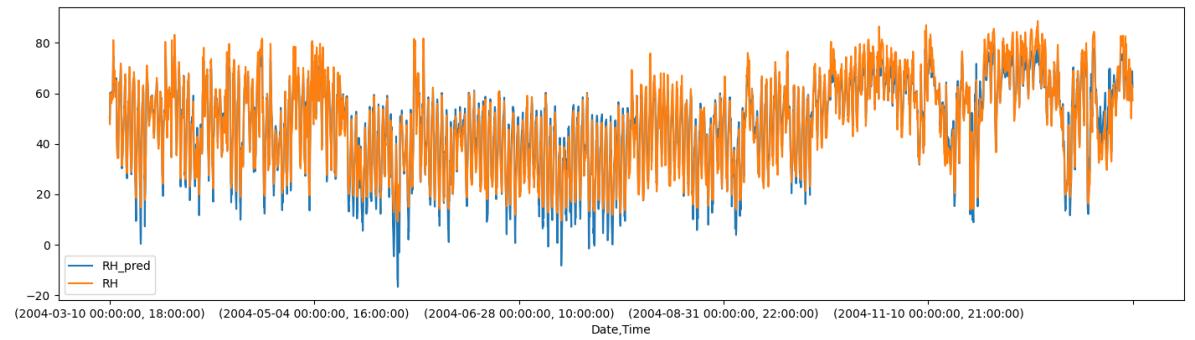


图 15: 多变量多项式回归效果 (训练集)

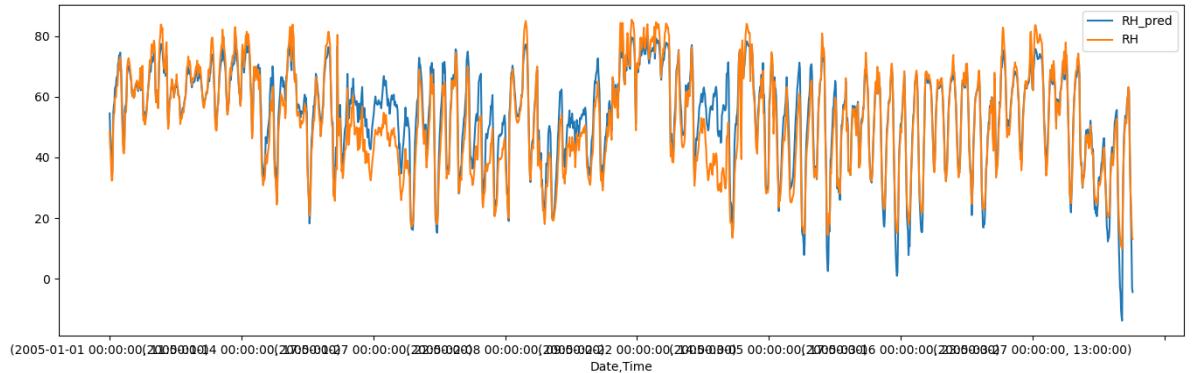


图 16: 多变量多项式回归效果 (测试集)

虽然说阶数越高，拟合本领在局部会越强，这里采用了一个九阶的高阶项，意在说明，震荡会发生，在学习数理方程以及计算方法中，这个现象叫做龙格现象。

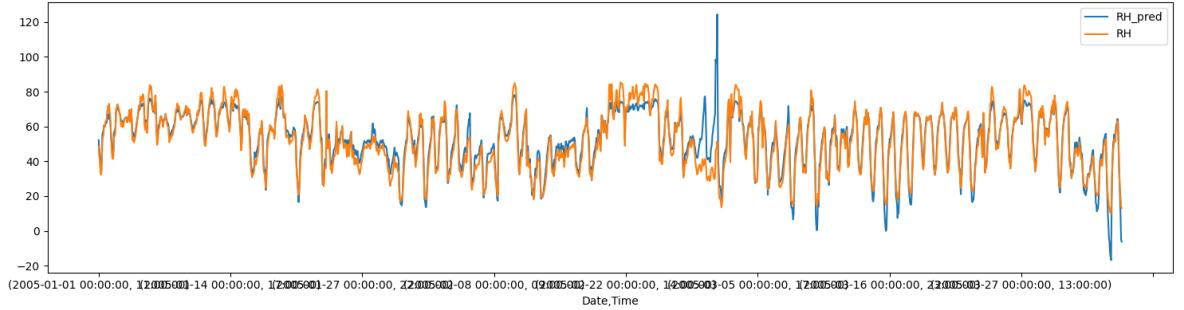


图 17: 九阶多变量多项式回归震荡效果

4 岭回归

4.1 算法介绍

岭回归是一种专用于共线性数据分析的有偏估计回归方法，实质上是一种改良的最小二乘估计法，通过放弃最小二乘法的无偏性，以损失部分信息、降低精度为代价获得回归系数更为符合实际、更可靠的回归方法，对病态数据的拟合要强于最小二乘法。

回归分析中常用的最小二乘法是一种无偏估计。对于一个适定问题， X 通常是列满秩的。当 X 不是列满秩时，或者某些列之间的线性相关性比较大时， $X^T X$ 的行列式接近于 0，即 $X^T X$ 接近于奇异，上述问题变为一个不适定问题，此时，计算 $(X^T X)^{-1}$ 时误差会很大，传统的最小二乘法缺乏稳定性与可靠性。

为了解决上述问题，我们需要将不适定问题转化为适定问题：我们为上述损失函数加上一个正则化项，变为

$$Loss = \|X\beta - Y\|^2 + \|\alpha I\beta\|^2 \quad (10)$$

于是：

$$\beta = (X^T X + \alpha I)^{-1} X^T Y \quad (11)$$

α 是一个未知变量，可以不断调整找到合适的 α 值。在这个过程中 β 在不断地变化，这就是所说的迹岭。

4.2 算法实现

下面列出算法逻辑：

Algorithm 1: 岭回归算法

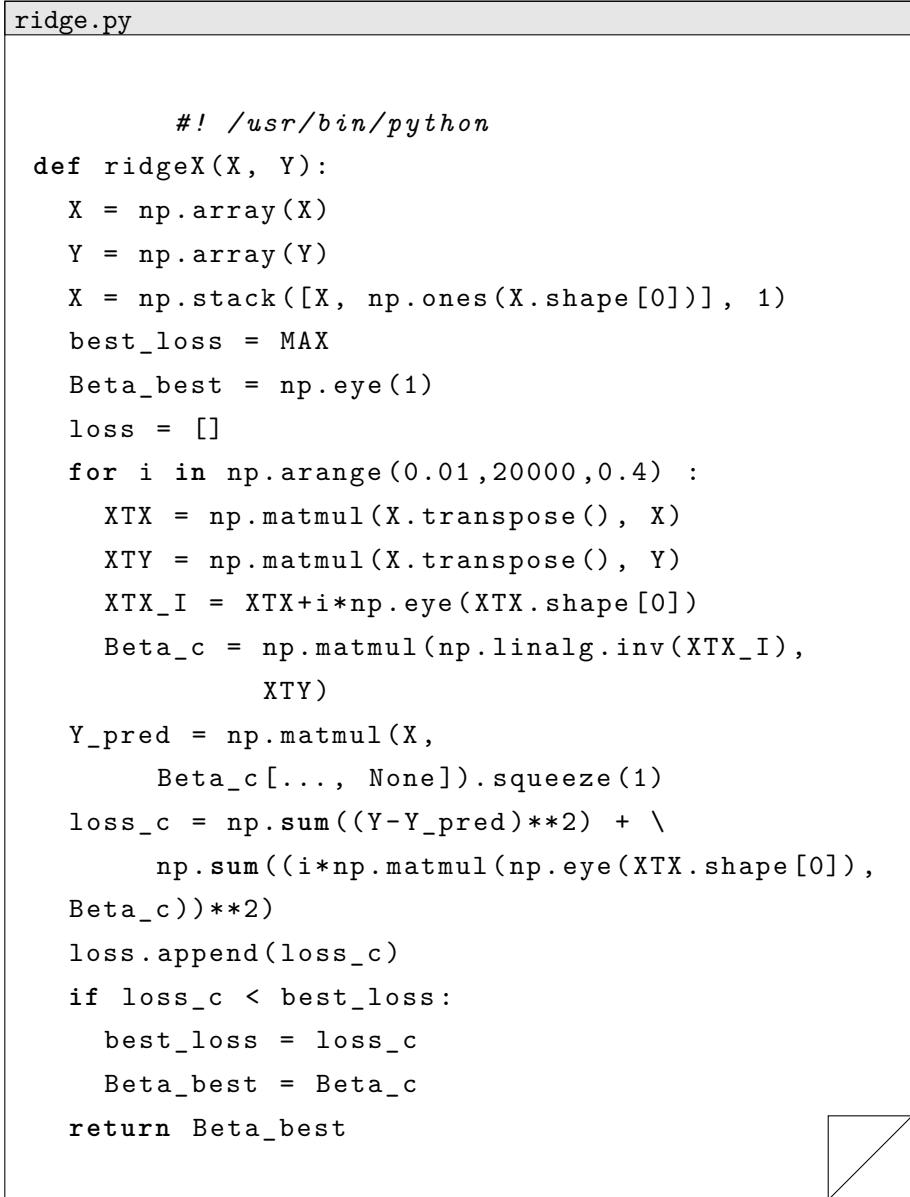
Input: (X, Y) , two floating-point matrix

Result: β , matrix

```

for  $\alpha \in 0.01, 100$  do
     $\beta_{current} = (X^T X + \alpha I)^{-1} X^T Y$  ;
     $loss_{current} = \|X\beta - Y\|^2 + \|\alpha I\beta\|^2$  ;
    if  $loss_{current} < loss_{best}$  then
         $loss_{best} = loss_{current}$  ;
         $\beta_{best} = \beta_{current}$ 
    end
end
return  $\beta_{best}$  ;

```



```

#!/usr/bin/python

def ridgeX(X, Y):
    X = np.array(X)
    Y = np.array(Y)
    X = np.stack([X, np.ones(X.shape[0])], 1)
    best_loss = MAX
    Beta_best = np.eye(1)
    loss = []
    for i in np.arange(0.01, 20000, 0.4) :
        XTX = np.matmul(X.transpose(), X)
        XTY = np.matmul(X.transpose(), Y)
        XTX_I = XTX+i*np.eye(XTX.shape[0])
        Beta_c = np.matmul(np.linalg.inv(XTX_I),
                           XTY)
        Y_pred = np.matmul(X,
                           Beta_c [..., None]).squeeze(1)
        loss_c = np.sum((Y-Y_pred)**2) +
            np.sum((i*np.matmul(np.eye(XTX.shape[0]),
                                Beta_c))**2)
        loss.append(loss_c)
        if loss_c < best_loss:
            best_loss = loss_c
            Beta_best = Beta_c
    return Beta_best

```

4.3 岭回归误差曲线

从全局看，岭迹分析可用来估计在某一具体问题中最小二乘估计是否适用，把回归系数的岭迹都绘制在一张图上，如果这些曲线比较稳定，如图 18 所示，利用最小二乘估计会有一定的把握，这个问题下由于选用的 X 比较合适，可以达到无偏的效果，因此岭回归不适合这个问题，但是依旧可以做一下。

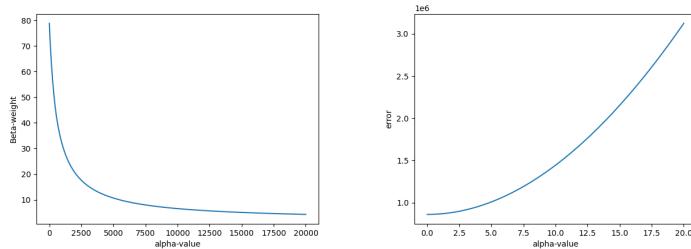


图 18: 岭迹图 (左), 误差曲线 (右)

4.4 岭回归效果

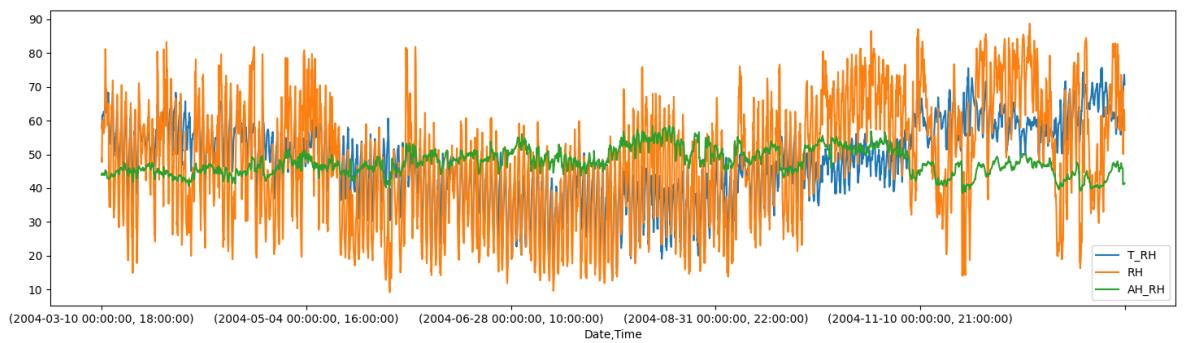


图 19: 岭回归效果 (训练集)

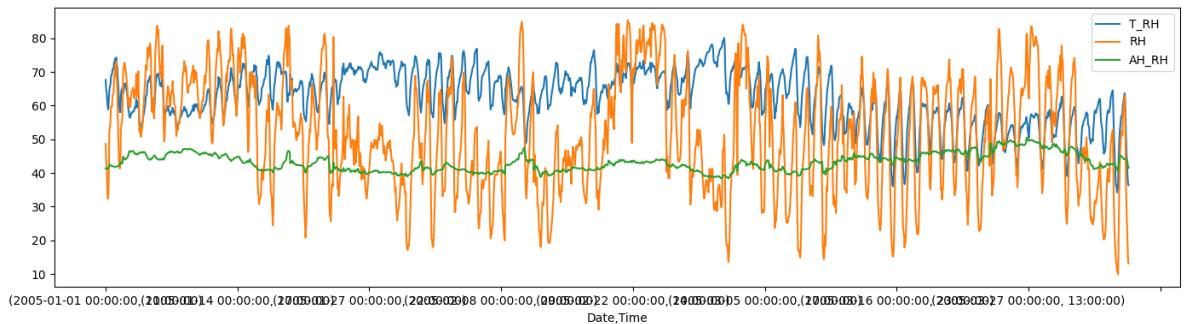


图 20: 岭回归效果 (测试集)

5 支持向量回归 SVR

5.1 算法介绍

支持向量分类的方法可以扩展来解决回归问题。此方法称为支持向量回归。通过支持向量分类生成的模型仅取决于训练数据的子集（支持向量），因为构建模型的成本函数并不关心超出裕度（最大化间隔）的训练点。类似地，由支持向量回归产生的模型仅取决于训练数据的子集，因为成本函数会忽略预测接近其目标的样本。

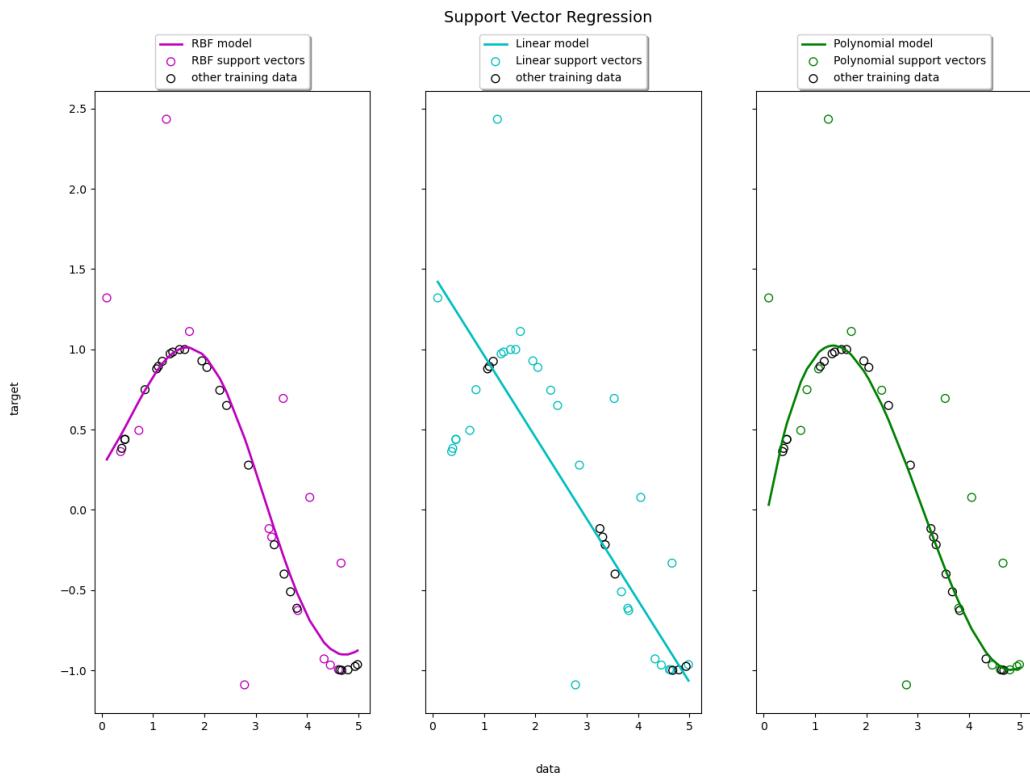


图 21: 支持向量回归算法示意

5.2 支持向量分类

给定训练向量 $x_i \in \mathbb{R}^p$ ，对于一个二分类问题，有分类目标 $y \in \{1, -1\}^n$ ，分类目标可以简述为，寻找参数 w 以及 b ，使得预测 $\text{sign}(w^T \phi(x) + b)$ 对于大多数情况是正确的。

描述软间隔 SVC 问题 (ζ_i 固定为 0，没有惩罚项的情况为硬间隔 SVC 问题) 为：

$$\min_{w,b,\zeta} \frac{1}{2} w^T w + C \sum_{i=1}^n \zeta_i \quad (12)$$

$$\begin{aligned} \text{st. } & y_i(w^T \phi(x_i) + b) \geq 1 - \zeta_i, \\ & \zeta_i \geq 0, i = 1, \dots, n \end{aligned} \quad (13)$$

支持向量机的核心思想是最大化间隔，也就是最小化 $\|W\|^2 = W^T W$ (这里有一定的推导过程，整体在于归一化思想就把最大化转为了最小化)，软间隔的思想是允许一定量的误差可以发生。

上式的对偶问题为：

$$\min_{\alpha} \frac{1}{2} \alpha^T Q \alpha - e^T \alpha \quad (14)$$

$$\text{st. } y^T \alpha = 0 \\ 0 \leq \alpha_i \leq C, i = 1, \dots, n \quad (15)$$

为了应对可能出现的非线性可分情况，需要对输入向量 $x_i \in \mathbb{R}^p$ 做高维映射，通常采用的方法为核函数，改写决策方程：

$$\sum_{i \in SV} y_i \alpha_i K(x_i, x) + b \quad (16)$$

5.3 支持向量回归

给定训练向量 $x_i \in \mathbb{R}^p$ ，以及预测目标 $y \in \mathbb{R}^n$ ，SVR 解决的问题可以描述为下：

$$\min_{w, b, \zeta, \zeta^*} \frac{1}{2} w^T w + C \sum_{i=1}^n (\zeta_i + \zeta_i^*) \quad (17)$$

$$\text{st. } y_i - w^T \phi(x_i) - b \leq \varepsilon + \zeta_i, \\ w^T \phi(x_i) + b - y_i \leq \varepsilon + \zeta_i^*, \\ \zeta_i, \zeta_i^* \geq 0, i = 1, \dots, n \quad (18)$$

其中 ζ_i 与 ζ_i^* 为惩罚项，类似于上述的软间隔方案，可以提供一定的误差空间，两个惩罚项各自代表这上下与回归预测误差空间。

上式的对偶问题为：

$$\min_{\alpha, \alpha^*} \frac{1}{2} (\alpha - \alpha^*)^T Q (\alpha - \alpha^*) + \varepsilon e^T (\alpha + \alpha^*) - y^T (\alpha - \alpha^*) \quad (19)$$

$$\text{st. } e^T (\alpha - \alpha^*) = 0 \\ 0 \leq \alpha_i, \alpha_i^* \leq C, i = 1, \dots, n \quad (20)$$

5.4 算法流程



| **Info:** 这部分采用 sklearn 机器学习库中的内容进行 SVR 实验，具体详见相关代码

```
svr.py

#!/usr/bin/python
def svr(kernel="linear", T="M", test=False):
    from sklearn.svm import SVR
    engine = SVR(kernel)
```

5.5 SVR 效果

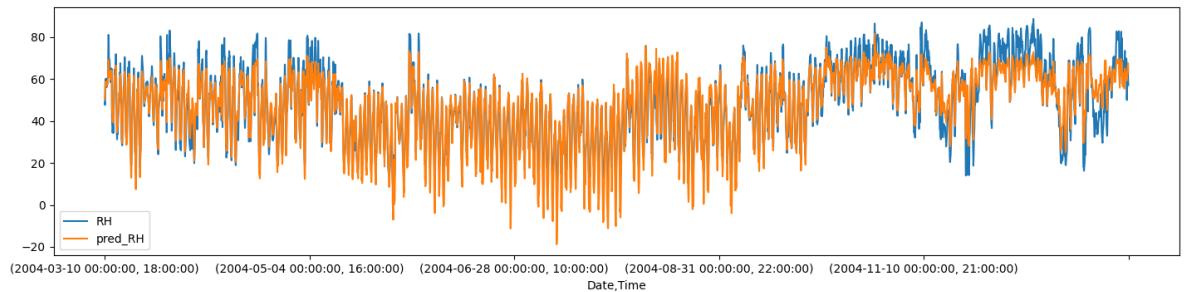


图 22: 多变量线性支持向量回归效果 (训练集)

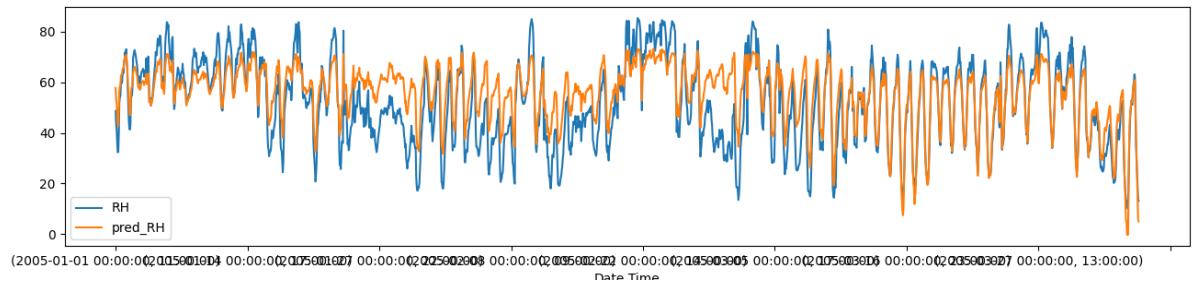


图 23: 多变量线性支持向量回归效果 (测试集)

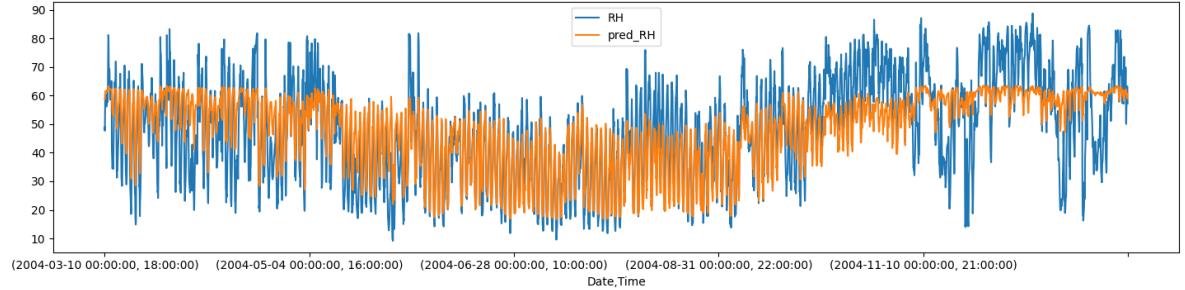


图 24: 多变量 RBF 支持向量回归效果 (训练集)

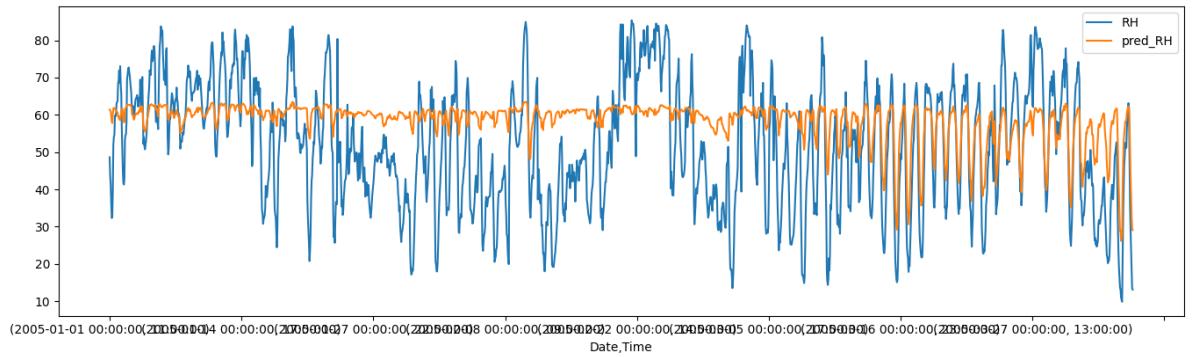


图 25: 多变量 RBF 支持向量回归效果 (测试集)

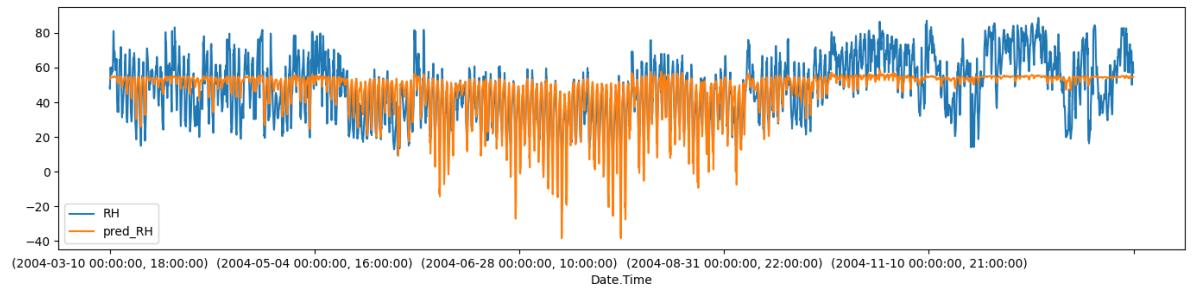


图 26: 多变量多项式支持向量回归效果 (训练集)

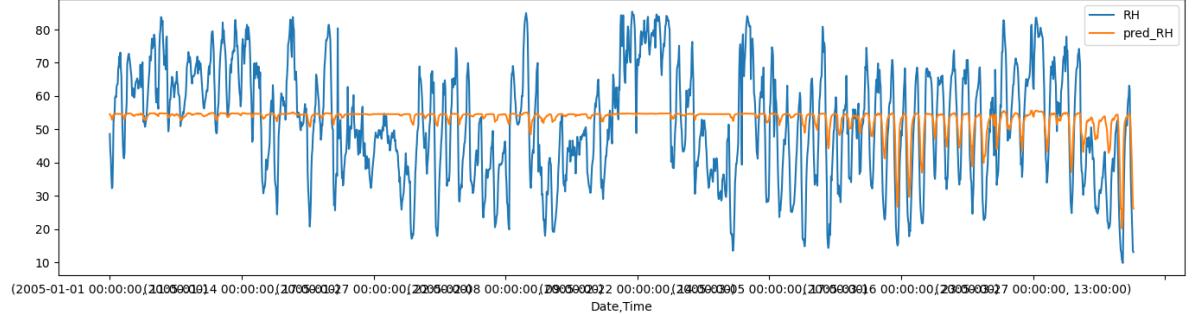


图 27: 多变量多项式支持向量回归效果 (测试集)

6 总结

6.1 算法性能对比

误差曲线对比 下面对上述所有多变量方法进行算法的误差曲线评估，单变量一定效果是不如多变量的，在上文中已经多次验证，因此下文就对多变量效果进行分析。

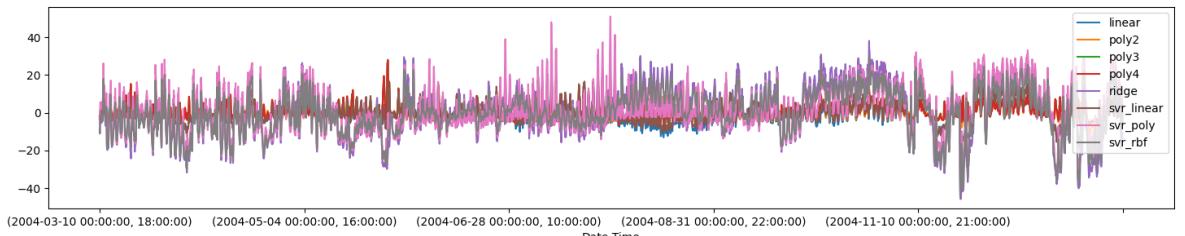


图 28: 多变量误差曲线 (训练集)

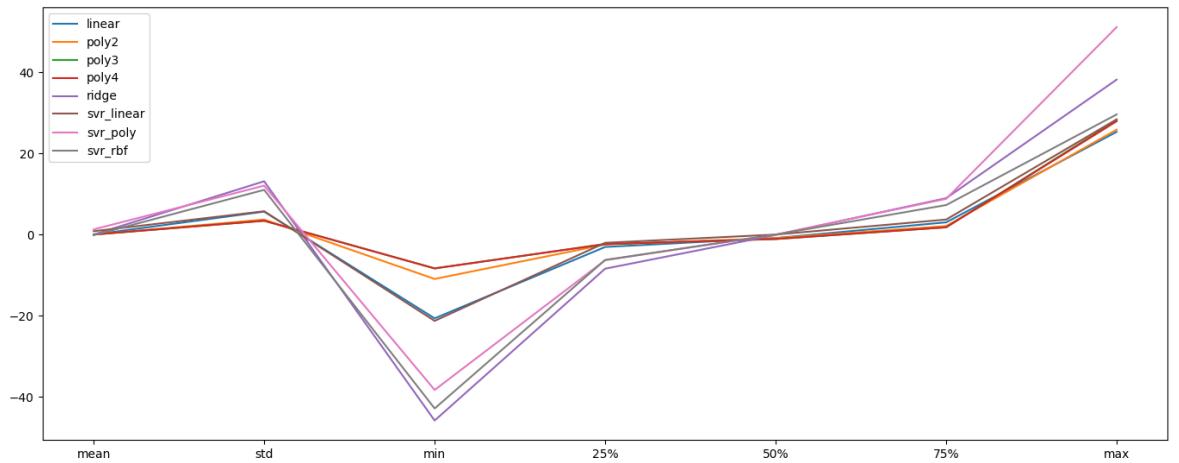


图 29: 多变量误差基本属性

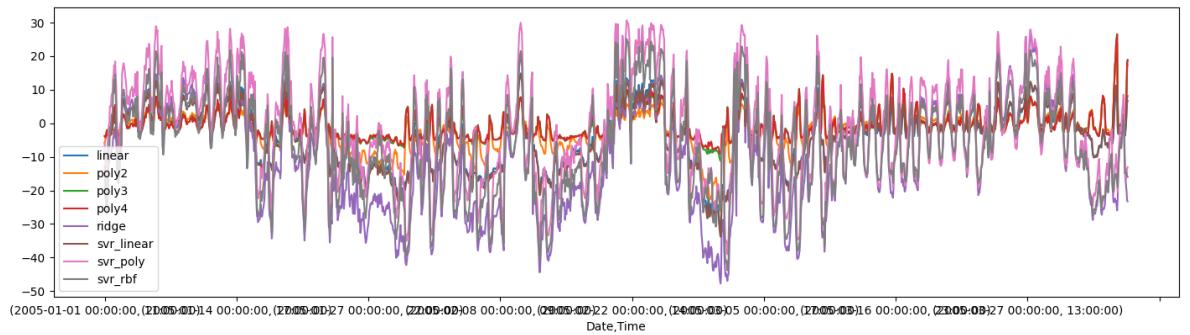


图 30: 多变量误差曲线 (训练集)

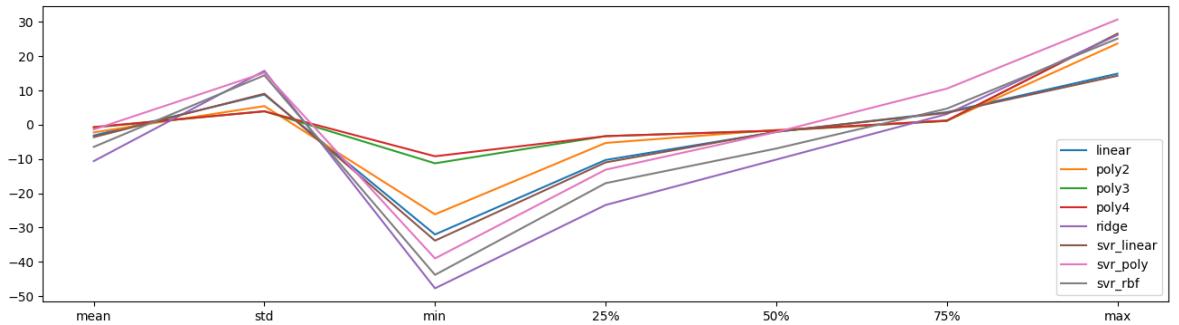


图 31: 多变量误差基本属性

可以看到，多项式型多变量回归模型是效果最好的，同时岭回归和线性模型以及线性支持向量回归属性很相近，但是他们的出发原理是不同的，岭回归由于这个问题下的特殊性，自然是和线性模型很近似，如果我设置的岭回归参数足够小，那么应该理论上会是一个线性模型。

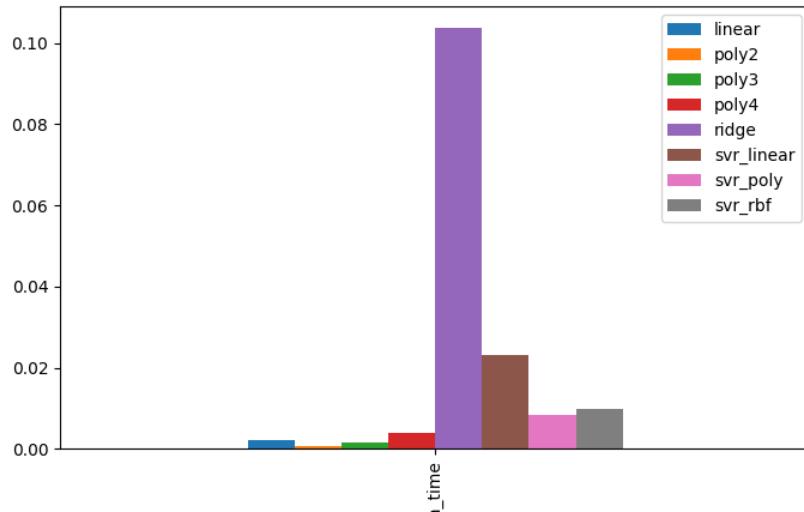


图 32: 算法时间对比

计算时间对比 二阶效率比线性还快，后面查资料发现，是因为二阶模型会使得矩阵变为范德猛矩阵，可以加速计算，同样在支持向量回归的对比上也验证了这点，因为 SVR 我是采用的调包的方式。而岭回归花费时间较长是因为参数的选择，我采用遍历的方式，比较耗费时间。

6.2 总结与感想

在商务智能大作业中，我选择去探究一下回归分析的内容，学习包括线性回归，多项式回归，多变量多项式回归，以及岭回归，了解到岭回归是在矩阵不满秩，变量有强相关性的基础下诞生的，当然，在我选定的题目条件下，这个问题不适用于岭回归的技巧，通过岭迹图可以看到稳定的变化曲线，本次最重要的是，以前都是调包实现回归分析，这次通过自己手写这几个方案，感受到回归算

法不一样的味道，最小二乘法在本次实验中起到了非常重要的作用，其实对于最小二乘还有其他的拓展形式，为了达到在线学习的目的，可以改为递推最小二乘，以及考虑方差的加权最小二乘，和用在数据处理过程中的移动最小二乘。以及最后去学习支持向量回归的内容，以前接触过支持向量机，所以学习的难度不是很大，主要是思维的转变，支持向量机一般被用在分类问题上，在回归问题上还是第一次遇到，思路的确很不错。

7 Implementation

本次实验所有代码采用 python3.6 完成，运行环境为 ubuntu20.04，安装依赖方法为：

Command Line

```
$ pip3 -r install ./requirements.txt
$ python3 ./linear.py #线性回归
$ python3 ./poly.py    #多项式回归
$ python3 ./ridge.py   #岭回归
$ python3 ./svr.py     #支持向量回归
$ python3 ./time.py    #对比算法时间
```



Notice: 代码同步放到 <https://github.com/WANGSSSSSS/regression-learning> 仓库。具体计算方法，详见各个文件代码。