《数据挖掘应用实验》课程报告

实验 2

yunwuyue e-mail Student ID

Lanzhou University

Date: 2023 年 4 月 1 日

1 数据集

1.1 正弦周期数据集 D1

生成 D1 的步骤如下：

1. 随机产生一个包含两个正弦周期的数据集：首先，定义样本数与振幅，然后使用 linspace 函数生成一个包含样本数个元素的等差数列 t，起点为 0，终点为 2π，这里 endpoint 值为 False 表示不包含终点，即区间取值为 [0, 2π)。接着用两个正弦函数计算出 y，并在 y 中添加随机扰动。
2. 对数据集进行均匀采样，形成数据集 D1：定义采样数，使用 linspace 函数生成 indices（包含 n_samples_D1 个整数的等差数列），起点为 0，终点为 n_samples-1。然后创建一个 shape 为 (n_samples_D1, 2) 的零矩阵 D1，将 t 和 y 中的 indices 对应的值分别赋给 D1 中的第一列和第二列。
3. 可视化数据集：使用 matplotlib 库绘制了原始数据集 y 的图像（如图1所示），同时在图中用散点表示了采样得到的数据集 D1。

```
import numpy as np
import matplotlib.pyplot as plt

np.random.seed(1234)

n_samples = 200
amplitude = 0.5
t = np.linspace(0, 2 * np.pi, n_samples, endpoint=False)
y = amplitude * np.sin(t) + amplitude * np.sin(2 * t)

noise = np.random.randn(n_samples) * 0.1
y += noise

n_samples_D1 = 20
```

```
indices = np.linspace(0, n_samples-1, n_samples_D1, dtype=np.int32)
D1 = np.zeros((n_samples_D1, 2))
D1[:, 0] = t[indices]
D1[:, 1] = y[indices]

plt.plot(t, y, label='original data')
plt.scatter(D1[:, 0], D1[:, 1], label='D1')
plt.legend()
plt.show()
```

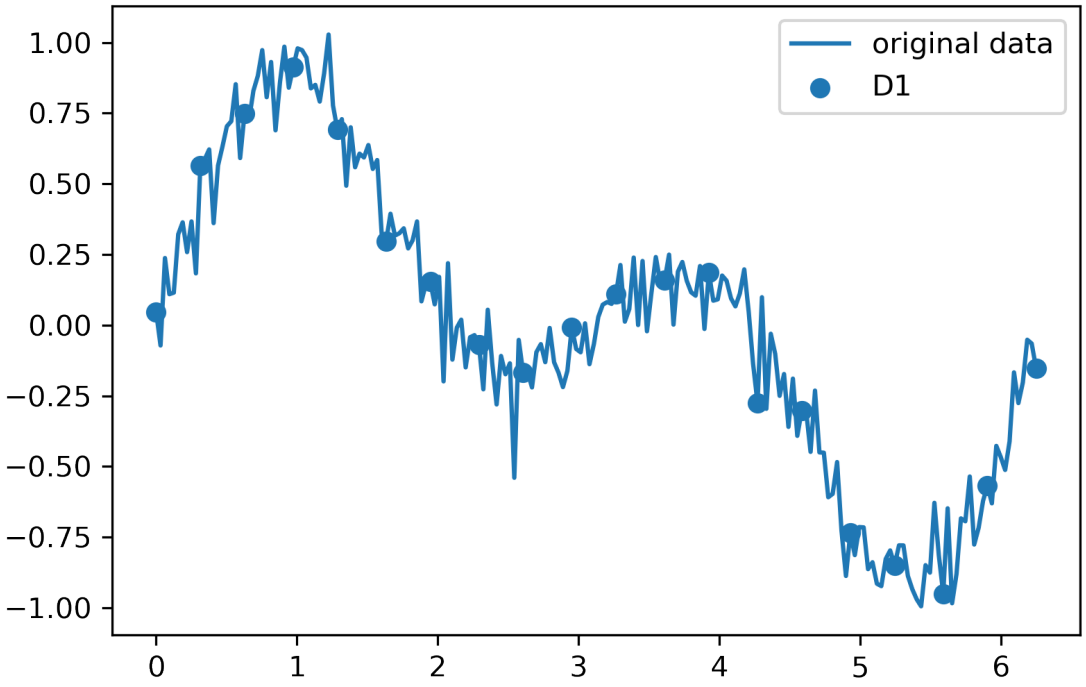

图 1: D1_visual

在均匀采样并添加扰动之后，使用 savetxt 函数将处理后的数据集保存为 D1。

```
np.savetxt('data.csv', D1, delimiter=',', header='t,y', comments='')
```

表1展示了 D1 数据集的前十行数据。

1.2 UCI 数据集

从 UCI dataset repository 中下载一个数据集 Winequality(red)[1], 其部分数据如表2所示。

该数据集包含红酒和白酒样本，特征中包含一列符号型数据'Quality'，作为样本的质量标签，同时包含 n 列连续的数值型数据，这些数据包括酒的理化特性，如酸度、挥发性酸度、柠檬酸含量、

t	y
0.000000	0.047144
0.314159	0.563405
0.628319	0.749156
0.973894	0.912172
1.288053	0.691416
1.633628	0.296149
1.947787	0.153412
2.293363	-0.068651
2.607522	-0.168370
2.953097	-0.008712

表 1: 数据集 D1（部分）

残糖含量等。该数据集可以用于预测红酒和白酒的质量。

free sulfur dioxide	total sulfur dioxide	density	pH	sulphates	alcohol	quality
11.0	34.0	0.9978	3.51	0.56	9.4	5
25.0	67.0	0.9968	3.20	0.68	9.8	5
15.0	54.0	0.9970	3.26	0.65	9.8	5
17.0	60.0	0.9980	3.16	0.58	9.8	6
11.0	34.0	0.9978	3.51	0.56	9.4	5
13.0	40.0	0.9978	3.51	0.56	9.4	5
15.0	59.0	0.9964	3.30	0.46	9.4	5
15.0	21.0	0.9946	3.39	0.47	10.0	7
9.0	18.0	0.9968	3.36	0.57	9.5	7
17.0	102.0	0.9978	3.35	0.80	10.5	5

表 2: 数据集 wine_quality（部分）

2 数据预处理

2.1 数据标准化处理

2.1.1 min-max 方法

在数据离散化中，$min-max$ 方法是一种常用的数据标准化方法，它的目标是将原始数据映射到一个固定的区间内，通常是 [0,1] 或 [-1,1]（这里我们映射到 [0,1] 区间）。这个方法的核心思想是将原始数据中的最小值映射为 0，最大值映射为 1，其他数值按比例映射到这个区间内，这个方法

可以有效地将数据映射到固定的区间内，并保持数据的相对大小关系。

具体地说，$min-max$ 方法可以用以下公式表示：

$$x' = \frac{x - min}{max - min} \tag{1}$$

- x 是原始数据；
- min 是原始数据中的最小值；
- max 是原始数据中的最大值；
- x' 是标准化后的数据。

```
minmax = (df[num_col] - df[num_col].min()) / (df[num_col].max() - df[num_col].min())
```

在 python 中，可以用上述代码实现 $min-max$ 方法，处理过的数据集部分如下（表3）所示。

free sulfur dioxide	total sulfur dioxide	density	pH	sulphates	alcohol	quality
0.140845	0.098940	0.567548	0.606299	0.137725	0.153846	0.4
0.338028	0.215548	0.494126	0.362205	0.209581	0.215385	0.4
0.197183	0.169611	0.508811	0.409449	0.191617	0.215385	0.4
0.225352	0.190813	0.582232	0.330709	0.149701	0.215385	0.6
0.140845	0.098940	0.567548	0.606299	0.137725	0.153846	0.4
0.169014	0.120141	0.567548	0.606299	0.137725	0.153846	0.4
0.197183	0.187279	0.464758	0.440945	0.077844	0.153846	0.4
0.197183	0.053004	0.332599	0.511811	0.083832	0.246154	0.8
0.112676	0.042403	0.494126	0.488189	0.143713	0.169231	0.8
0.225352	0.339223	0.567548	0.480315	0.281437	0.323077	0.4

表 3: minmax 标准化后的数据集（部分）

2.2 认识数据集 D2

Wine Quality 数据集是一个用于回归分析的数据集，包含了红葡萄酒和白葡萄酒的数据。该数据集共有 12 个特征（或者说是列），分别是：

- fixed acidity：固定酸度，以 $g(tartaricacid)/dm^3$ 为单位。
- volatile acidity：挥发性酸度，以 $g(aceticacid)/dm^3$ 为单位。
- citric acid：柠檬酸，以 g/dm^3 为单位。
- residual sugar：残糖，以 g/dm^3 为单位。
- chlorides：氯化物，以 $g(sodiumchloride)/dm^3$ 为单位。
- free sulfur dioxide：游离二氧化硫，以 mg/dm^3 为单位。
- total sulfur dioxide：总二氧化硫，以 mg/dm^3 为单位。
- density：密度，以 g/cm^3 为单位。

- pH：酸碱度，无量纲。
- sulphates：二氧化硫酸盐，以 $g(potassiumsulphate)/dm^3$ 为单位。
- alcohol：酒精含量，以 %vol 为单位。
- quality：葡萄酒质量，用 0-10 的整数表示，作为回归目标。

每一列的单位和含义如上所述。这些特征可以用来预测葡萄酒的质量评分，可以使用回归分析来建立一个预测模型。

在 Wine Quality 数据集中，每个特征都可能对葡萄酒的质量评分产生影响。不过通常来说，影响最大的特征是与葡萄酒的口感和风味有关的特征，例如酸度、残糖、酒精含量等。具体来说，可以通过计算每个特征与目标变量 quality 之间的相关系数来了解特征对 quality 的影响大小。

可以通过以下代码来计算每个特征与 quality 之间的相关系数，并将结果按照绝对值从大到小排序，以了解每个特征对 quality 的影响大小：

```
import pandas as pd

# 计算每个特征与 quality 之间的相关系数
corr = df.corr()['quality'].abs().sort_values(ascending=False)

# 打印结果
print(corr)
```

上述代码输出了每个特征与 quality 之间的相关系数，按照绝对值从大到小排序，可以根据排序结果来判断哪些特征对 quality 的影响较大。在 Wine Quality 数据集中，可能的结果是 alcohol（酒精含量）,volatile acidity（挥发性酸度）,sulphates(二氧化硫酸盐) 三列数据对 quality 的影响较大。

1. quality：1.000000
2. alcohol：0.476166
3. volatile acidity：0.390558
4. sulphates：0.251397
5. citric acid：0.226373
6. total sulfur dioxide：0.185100
7. density：0.174919
8. chlorides：0.128907
9. fixed acidity：0.124052
10. pH: 0.057731
11. free sulfur dioxide : 0.050656
12. residual sugar : 0.013732

为了便于回归分析，我们选取这三列数据，作为自变量 $x1$、$x2$、$x3$。

2.2.1 形成 D1_discrete

下列代码展示了选取三个与回归目标 quality 相关系数最大的三列数据，如下表4所示。

```
df_d2 = df_minmax[['alcohol','volatile acidity','sulphates','quality']]
outputpath='C:/Users/shenxuan/Desktop/D2.csv'
df_d2.to_csv(outputpath,sep=',',index=False,header=False)
```

alcohol	volatile acidity	sulphates	quality
0.153846	0.397260	0.137725	0.4
0.215385	0.520548	0.209581	0.4
0.215385	0.438356	0.191617	0.4
0.215385	0.109589	0.149701	0.6
0.153846	0.397260	0.137725	0.4
0.153846	0.369863	0.137725	0.4
0.153846	0.328767	0.077844	0.4
0.246154	0.363014	0.083832	0.8
0.169231	0.315068	0.143713	0.8
0.323077	0.260274	0.281437	0.4

表 4: 数据集 D2（部分）

3 回归分析

3.1 一元多项式回归

一元多项式回归是一种基于多项式函数来建模的回归方法，它通过对输入数据进行多项式特征转换，将原始输入数据的一元特征映射到多项式空间中 [2]，并在该空间中拟合一个线性模型来预测输出变量。通过这种方式，可以比较不同多项式阶数的模型的性能，并选择最适合数据的多项式阶数。

3.1.1 不同多项式阶数下的拟合曲线

```
t_plot = np.linspace(0, 2 * np.pi, 1000, endpoint=False)
X_plot_poly = poly.transform(t_plot.reshape(-1, 1))
y_plot = model.predict(X_plot_poly)

plt.plot(t_plot, y_plot.flatten(), label=f'degree={degree}')
```

上述代码绘制了多项式拟数 m 取不同值时得到的拟合曲线（图2）。

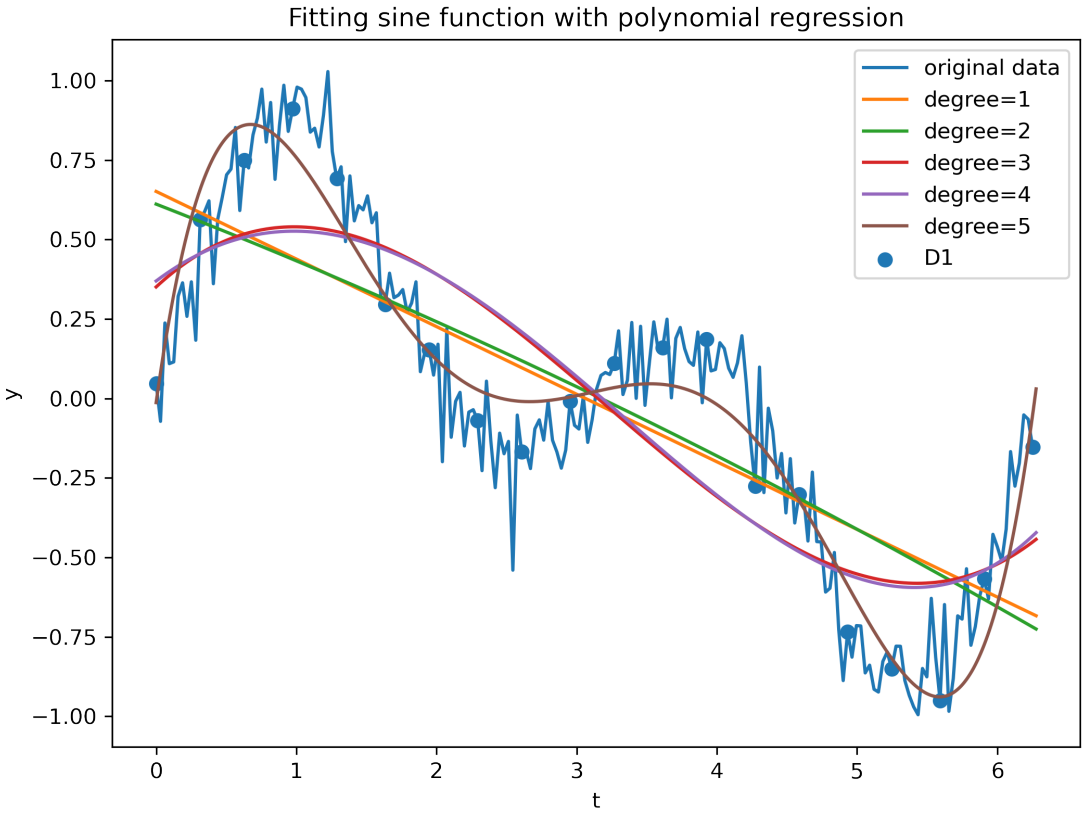

图 2: 不同阶数的拟合曲线

3.1.2 分析 MAE 与 RMSE

首先，我们可以定义一个要测试的多项式阶数列表，并将数据集拆分成训练集和测试集。对于每个多项式阶数，通过 PolynomialFeatures 函数将数据集转换成多项式特征，然后使用 LinearRegression 模型训练模型。训练后的模型用于对测试数据进行预测，并计算预测结果的平均绝对误差（MAE）和均方根误差（$RMSE$）。最后，输出每个多项式阶数对应的 MAE 和 $RMSE$。

```
polynomial_degrees = [1, 2, 3, 4, 5]

X = D1[:, 0].reshape(-1, 1)
y = D1[:, 1].reshape(-1, 1)
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state
    =1234)

for degree in polynomial_degrees:
    from sklearn.preprocessing import PolynomialFeatures
    poly = PolynomialFeatures(degree)
    X_train_poly = poly.fit_transform(X_train)
```

```
    X_test_poly = poly.transform(X_test)

    model = LinearRegression()
    model.fit(X_train_poly, y_train)

    y_pred = model.predict(X_test_poly)
    mae = mean_absolute_error(y_test, y_pred)
    rmse = np.sqrt(mean_squared_error(y_test, y_pred))
```

上述代码演示了如何使用一元多项式回归来拟合数据，并评估模型的性能。在该代码中，首先定义了多项式的阶数，然后将数据集划分为训练集和测试集，然后对每个多项式阶数进行训练和测试，计算并输出模型的 MAE 和 $RMSE$ 指标。

将结果转换为 LaTeX 表格，如下表5所示：

Degree	MAE	RMSE
1	0.363	0.414
2	0.384	0.429
3	0.320	0.360
4	0.347	0.401
5	0.130	0.139

表 5: 不同阶数的结果分析

通过分析图2与图3，可以得出以下结论：

- 多项式阶数为 3 和 5 时，模型在测试集上表现最好，MAE 和 $RMSE$ 值最小。
- 多项式阶数为 1 和 4 时，模型在测试集上表现较差，MAE 和 $RMSE$ 值较大。
- 多项式阶数为 2 时，模型在测试集上表现一般，MAE 和 $RMSE$ 值介于其他多项式阶数之间。

因此，我们可以选择多项式阶数为 3 或 5 的模型来对这个数据集进行拟合。同时，我们也需要注意过拟合的问题，因为在多项式阶数过高时，模型可能会在训练集上表现得很好，但在测试集上表现较差。因此，在实际应用中，需要根据实际情况进行选择。

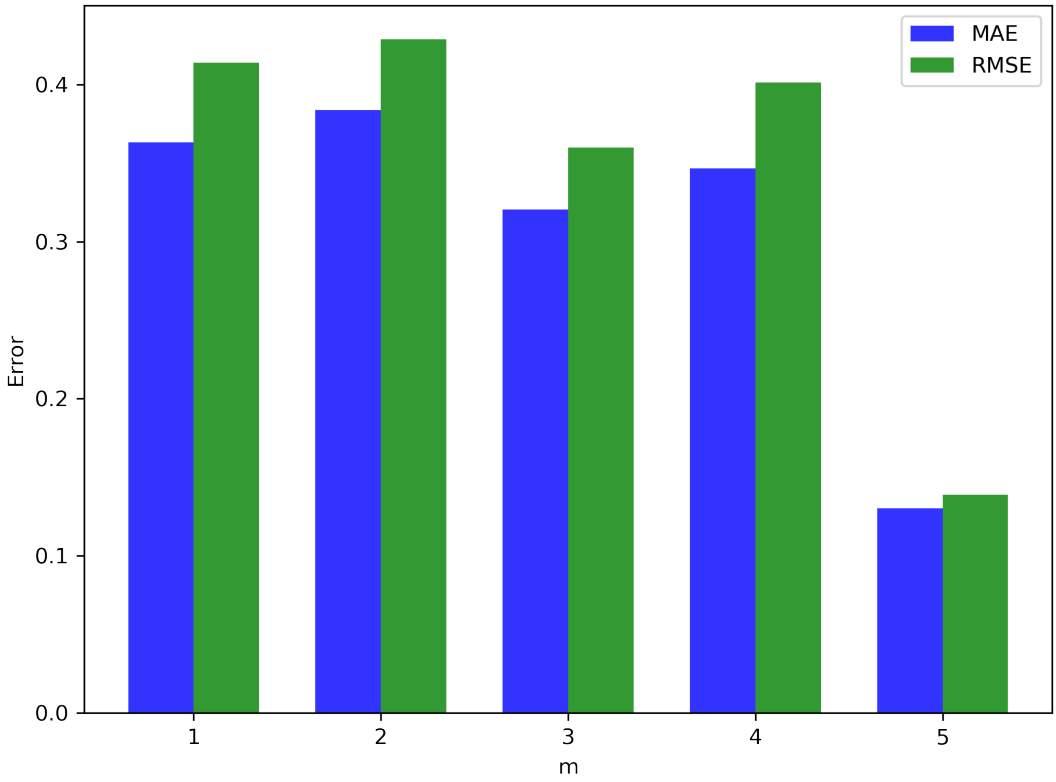

图 3: 不同阶数下的 MSE 与 RMSE 的柱状图

3.2 Ridge 回归

3.2.1 确定回归系数与超参数

Ridge 回归是一种用于线性回归的正则化方法。Ridge 回归通过向目标函数添加一个 L2 正则化项，限制回归系数的平方和不超过一个预定的常数，从而降低模型的复杂度和泛化误差。

Ridge 回归的目标函数可以表示为：

$$\text{minimize}_{\boldsymbol{w}} \left[\sum_{i=1}^{n} (y_i - \boldsymbol{w}^T \boldsymbol{x}_i)^2 + \alpha \sum_{j=1}^{p} w_j^2 \right]$$

其中，$\boldsymbol{w}$ 是回归系数向量，y_i 是第 i 个样本的真实值，$\boldsymbol{x}_i$ 是第 i 个样本的特征向量，p 是特征的数量，α 是正则化系数，用于控制正则化项的影响程度。

Ridge 回归的优化问题可以使用梯度下降等优化算法求解。在实践中，通常使用类似于线性回归的封闭形式解法，即：

$$\boldsymbol{w} = (\boldsymbol{X}^T \boldsymbol{X} + \alpha \boldsymbol{I})^{-1} \boldsymbol{X}^T \boldsymbol{y}$$

其中，$\boldsymbol{X}$ 是 $n \times p$ 的特征矩阵，$\boldsymbol{y}$ 是 $n \times 1$ 的标签向量，$\boldsymbol{I}$ 是 $p \times p$ 的单位矩阵。

Ridge 回归可以有效地处理多重共线性问题，即当特征之间存在较强的相关性时，Ridge 回归可以避免出现过拟合的情况。但是，与 Lasso 回归不同，Ridge 回归不会将某些回归系数归零，因此 Ridge 回归不能用于特征选择 [3]。

```
from sklearn.linear_model import Ridge
from sklearn.preprocessing import StandardScaler
from sklearn.model_selection import GridSearchCV

X = df_d2.drop('quality', axis=1)
y = df_d2['quality']

scaler = StandardScaler()
X_scaled = scaler.fit_transform(X)

ridge = Ridge()
```

在回归模型中，正则化参数是一种用于控制模型复杂度的超参数，用于解决过拟合问题。

为了防止过拟合，可以在模型损失函数中加入一个正则化项，使得模型倾向于选择更简单的模型。正则化项一般是模型参数的范数，而正则化参数则是用于控制正则化项权重的超参数。较大的正则化参数会惩罚较大的参数权重，使得模型更倾向于选择较小的参数权重，从而减小过拟合风险。

在 Ridge 回归中，正则化参数 λ 用于控制 L2 范数的权重，即使得回归系数向量的平方和越小，从而使得模型更倾向于选择较小的回归系数。

主要用来防止模型过拟合，直观上理解就是 L2 正则化是对于大数值的权重向量进行严厉惩罚。鼓励参数是较小值，如果 w 小于 1，那么 w^2 会更小。

这段代码实现了使用网格搜索 (GridSearchCV) 选择最佳正则化系数，然后在最佳正则化系数下重新训练 Ridge 模型并获取其正则化路径 (coefs)，最后使用 RidgeCV 确定稳定的超参数取值。

具体来说，对于一个 Ridge 模型，它有一个正则化参数 $alpha$，通过调整 $alpha$，可以得到一系列的模型，这些模型中的每个都对数据的拟合有所不同。一般来说，$alpha$ 越大，模型就越倾向于简单的系数。在这个例子中，param_grid 定义了一个包含 100 个值的 $alpha$ 范围，从 10 的-4 次方到 10 的 4 次方之间的对数空间。然后，使用网格搜索 (GridSearchCV) 从这些 alpha 中选择最佳正则化系数。接下来，使用选择的最佳正则化系数重新创建一个 Ridge 模型，重新训练它，并获取它的正则化路径 (coefs)，即不同 $alpha$ 下的回归系数。最后，使用 RidgeCV 确定稳定的超参数取值，即最能够泛化到新数据的超参数值（图4）。

```
param_grid = {'alpha': np.logspace(-4, 4, 100)}

grid_search = GridSearchCV(ridge, param_grid, cv=5)
grid_search.fit(X_scaled, y)
best_alpha = grid_search.best_params_['alpha']

ridge = Ridge(alpha=best_alpha)
```

```
ridge.fit(X_scaled, y)

alphas = np.logspace(-4, 4, 100)
coefs = []
for a in alphas:
    ridge = Ridge(alpha=a)
    ridge.fit(X_scaled, y)
    coefs.append(ridge.coef_)
coefs = np.array(coefs)

from sklearn.linear_model import RidgeCV

ridge_cv = RidgeCV(alphas=alphas, scoring='neg_mean_squared_error', cv=5)
ridge_cv.fit(X_scaled, y)
print('Stable alpha:','%.3f'%ridge_cv.alpha_)
```

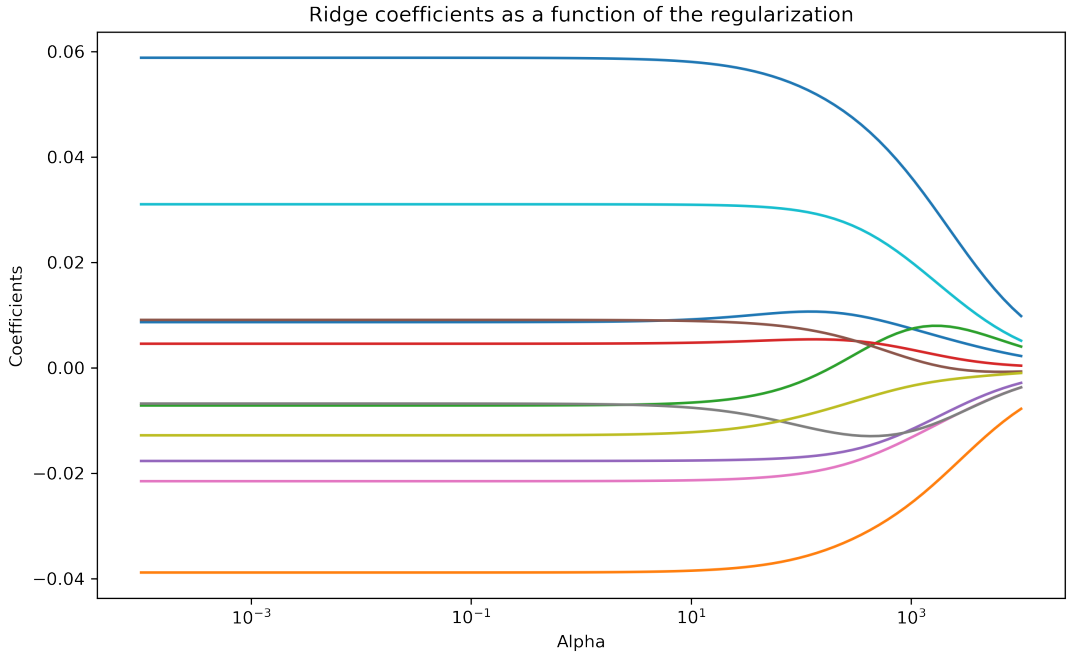

图 4: Ridge coefficients

首先将数据读入一个 DataFrame，然后将特征和标签分离。接着，对特征进行标准化处理，使得每个特征的平均值为 0，方差为 1。接着使用 GridSearchCV 进行网格搜索来选择最佳正则化系数，并使用该系数训练一个新的 Ridge 模型。接下来，获取正则化路径数据，然后绘制正则化路径图。最后，使用 RidgeCV 确定稳定的超参数 λ 的取值。

下面是按 80:20 比例划分数据集、使用 Ridge 回归模型进行训练和测试，并计算 MAE 和 $RMSE$ 值的代码实现。其中，使用的评价指标为 scikit-learn 库中的 mean_absolute_error 和 mean_squared_error。

```
X_train, X_test, y_train, y_test = train_test_split(df_d2[['alcohol', 'volatile
    acidity', 'sulphates']], df_d2['quality'], test_size=0.2)

ridge = Ridge(alpha=37.649)

ridge.fit(X_train, y_train)

y_pred = ridge.predict(X_test)

mae = mean_absolute_error(y_test, y_pred)
rmse = mean_squared_error(y_test, y_pred, squared=False)
```

得到结果如下。

- MAE: 0.11462532819902446
- RMSE: 0.14188830588800205

3.2.2 更改训练集与学习集比例

为了获取多组 MAE 和 $RMSE$ 值，可以将上述代码放在一个循环中，重复划分数据集、训练模型和计算指标的过程，如下所示：

```
alpha = 37.649
n_iter = 5
maes = []
rmses = []

for i in range(n_iter):
    X_train, X_test, y_train, y_test = train_test_split(df_d2[['alcohol', 'volatile
        acidity', 'sulphates']], df_d2['quality'], test_size=0.2)

    ridge = Ridge(alpha=alpha)

    ridge.fit(X_train, y_train)

    y_pred = ridge.predict(X_test)

    mae = mean_absolute_error(y_test, y_pred)
    rmse = mean_squared_error(y_test, y_pred, squared=False)

    maes.append(mae)
    rmses.append(rmse)
```

上述代码中，循环次数 n_iter 可以根据需要进行调整。每次循环都会划分一次数据集、训练模型并计算 MAE 与 $RMSE$, 接下来，画出每次划分后的 D2 相应的 MAE 与 $RMSE$（图5）。

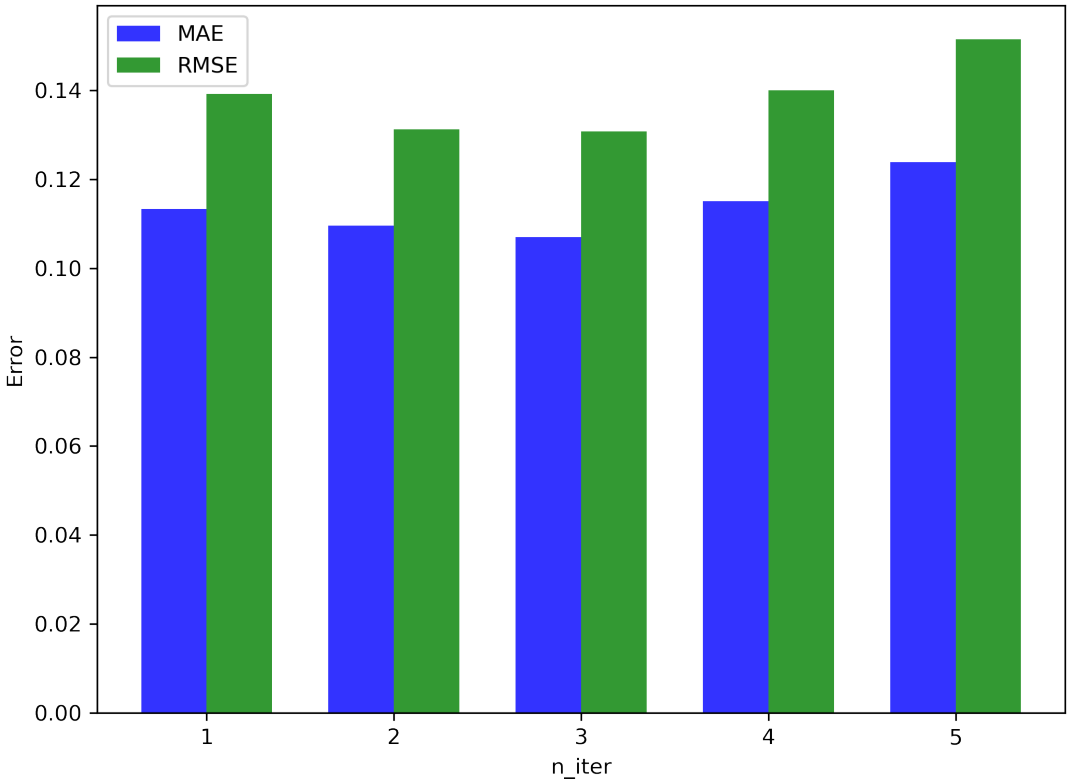

图 5: Ridge coefficients

可以看到，随着迭代次数的增加，MAE 与 $RMSE$ 均在下降，但是变化幅度较小，这可能是因为数据集中的噪声较少或者模型本身就能很好地适应数据，因此超参数的微小变化对模型的表现没有显著影响。此外，也有可能是模型训练的次数过少，不能够捕捉到超参数对模型表现的微小变化。

4 结论与思考

在完成实验作业的过程中，我发现在进行一元多项式和 Ridge 回归分析时，有以下几点需要注意的地方：

1. 数据预处理：在进行多项式回归之前，需要对数据进行预处理，包括特征缩放和归一化等操作，以避免不同特征之间的数量级差异对模型造成不利影响，这也是本次实验首先需要完成的工作；
2. 防止过拟合：在进行多项式回归和 Ridge 回归分析时，需要注意防止过拟合。在多项式回归中，可以使用交叉验证来选择最佳的多项式次数；本次实验我使用了网格搜索的方法来选择最佳的正则化参数，交叉验证也是可以选择的方式；

3. 参数解释：在多项式回归中，模型中的参数通常较难解释，因为它们与输入变量的幂次相关。在 Ridge 回归中，正则化参数 λ 控制着模型的复杂度，λ 越大，模型的复杂度越小，反之亦然。
4. 模型评估：在进行多项式回归和 Ridge 回归分析时，需要对模型进行评估。常用的评估指标包括均方误差（MSE）、均方根误差（$RMSE$）和平均绝对误差（MAE）等。在进行交叉验证时，需要对不同模型的评估结果进行比较，选择表现最好的模型，本次实验也要求获取多组 MSE 与 MAE 的值，从而更好的进行评估。

5 致谢

本文的 LaTeX 模板来源于 ElegantPaper。
本文中部分知识引用自课程 数据挖掘应用实验中的课件。

参考文献

[1] UCI Machine Learning Repository. *Wine Quality Data Set*. 2009. URL: http://archive.ics.uci.edu/ml/datasets/Wine+Quality (visited on 10/07/2009).

[2] L. P. Neves. A. Alves da Silva. “A method for estimating the parameters of a polynomial function from data contaminated by a noise signal.” In: *Signal Processing* 81.1 (2001), pp. 75–87. DOI: 10.1016/s0165-1684(00)00282-8.

[3] Robert W. Kennard. Arthur E. Hoerl. “Ridge regression: Biased estimation for nonorthogonal problems.” In: *Technometrics* 12.1 (1970), pp. 55–67. DOI: 10.1080/00401706.1970.10488634.