线性模型

刘德华

2023 年 6 月 13 日

例子 1

使用线性回归模型拟合糖尿病数据集[a]。

[a]实现的程序见例1

例子 2

使用非负线性回归模型拟合回归数据集[a]。

[a]实现的程序见例1

例子 3

使用岭回归模型拟合回归数据集[a]。

岭回归通过对系数的大小施加惩罚来解决普通最小二乘法的一些问题。岭回归的系数的估计是最小化惩罚的残差平方和：

$$\arg\min_{\beta} \|y - X\beta\|_2^2 + \alpha\|\beta\|_2^2$$

其中 $\alpha > 0$。它越大，惩罚力度越大，对共线性问题的解决越稳健。

[a]实现的程序见例1

例子 4

使用岭回归分类器对稀疏特征进行分类[a]。

岭回归分类器是将针对目标 y 的取值 $\{-1, 1\}$ 进行回归，本质上仍然是一个回归问题。

岭回归分类器也叫做线性核的最小二乘支持向量机。

[a]实现的程序见例1

例子 5

使用岭回归配合 CV 选择的方法进行最优参数选择[a]。

[a]实现的程序见例1

例子 6

使用 Lasso 回归模型和弹性网回归模型拟合数据集，并比较二者的结果[a]。

Lasso 回归模型是在高维稀疏假定下的一种模型，它是最小化下面的目标函数来求解系数的：

$$\arg\min_{\beta} \frac{1}{2n}\|y - X\beta\|_2^2 + \alpha\|\beta\|_1$$

[a]实现的程序见例1

例子 7

使用 Lasso 回归模型拟合 openml 数据集，wage 是预测变量[a]。

[a]实现的程序见例1

例子 8

使用信息准则 IC，建立 Lasso 回归模型做模型选择[a]。

[a]实现的程序见例1

例子 9

使用交叉验证 CV，建立 Lasso 回归模型做模型选择[a]。

[a]实现的程序见例1

例子 10

使用信息准则 IC，建立 Lasso 回归模型在糖尿病数据集上做模型选择[a]。

[a]实现的程序见例1

例子 11

使用 MultiLasso 回归模型，同时对多个回归模型选择同一组特征[a]。

MultiLasso 最小化的目标函数如下：

$$\arg\min_{W} \frac{1}{2n}\|Y - XW\|_{Fro}^2 + \alpha\|W\|_{21}$$

其中 $\|W\|_{21} = \sum_i \sqrt{\sum_j a_{ij}^2}$，$\|A\|_{Fro} = \sqrt{\sum_{ij} a_{ij}^2}$。

[a]实现的程序见例1

例子 12

建立弹性网回归模型对数据拟合[a]。

弹性网 Elastic Net 最小化的目标函数如下：

$$\arg\min_{\boldsymbol{\beta}} \frac{1}{2n}\|\boldsymbol{y} - \boldsymbol{X}\boldsymbol{\beta}\|_2^2 + \alpha\rho\|\boldsymbol{\beta}\|_1 + \frac{\alpha(1-\rho)}{2}\|\boldsymbol{\beta}\|_2^2$$

其中 ρ 表示 l_1 正则化的比例，越大越稀疏。

[a]实现的程序见例1

例子 13

绘制 Lasso 和弹性网回归模型的系数路径图[a]。

[a]实现的程序见例1

例子 14

使用最小角回归算法求解 lasso 模型，并构建回归模型[a]。

[a]实现的程序见例1

例子 15

使用正交匹配追踪求解带有下面这种约束的线性回归问题[a]。

$$\arg\min_{\boldsymbol{\beta}} \|\boldsymbol{y} - \boldsymbol{X}\boldsymbol{\beta}\|_2^2$$

$$s.t.\ \|\boldsymbol{\beta}\|_0 \leq t \quad \text{or} \quad s.t. \|\boldsymbol{y} - \boldsymbol{X}\boldsymbol{\beta}\|_2^2 \leq eps$$

[a]实现的程序见例1

例子 16

使用贝叶斯岭回归模型来拟合数据[a]。

贝叶斯岭回归使用的先验分布是：

$$p(\boldsymbol{\beta}|\lambda) = N(\boldsymbol{\beta}|\mathbf{0}, \lambda^{-1}\boldsymbol{I})$$

假定响应变量 $\boldsymbol{y}$ 的分布是

$$p(\boldsymbol{y}|\boldsymbol{X}, \boldsymbol{\beta}, \alpha) = N(\boldsymbol{y}|\boldsymbol{X}\boldsymbol{\beta}, \alpha)$$

所以后验分布是：

$$p(\boldsymbol{\beta}|\boldsymbol{y}) \propto p(\boldsymbol{y}|\boldsymbol{X}, \boldsymbol{\beta}, \alpha) \cdot p(\boldsymbol{\beta}|\lambda)$$

而参数 α 和 λ 是来自伽马分布的，因此严格地讲后验分布还需要乘以伽马分布的概率密度。

[a] 实现的程序见例1

例子 17

使用 l_1 惩罚来做多项 logistic 回归模型[a]。

带惩罚项的多项 logistic 回归模型的目标函数如下：

$$\arg\min_{\boldsymbol{\beta}} -C\sum_{i=1}^{n}\sum_{k=0}^{K-1}[y_i = k]\log(\hat{p}_k(\boldsymbol{X}_i)) + r(\boldsymbol{\beta})$$

其中的 $\hat{p}_k(\boldsymbol{X}_i)$ 表达如下：

$$\hat{p}_k(\boldsymbol{X}_i) = \exp\{\boldsymbol{X}_i\boldsymbol{\beta}_k\} / \sum_{l=0}^{K-1}\exp\{\boldsymbol{X}_i\boldsymbol{\beta}_l\}$$

其中的 $\boldsymbol{\beta}$ 表示由 $\boldsymbol{\beta}_l,\ l = 1, 2\cdots, K$ 构成的系数矩阵。

其中的 $r(\boldsymbol{\beta})$ 表示对系数矩阵的惩罚项，具体有

l_1, l_2 惩罚以及弹性网惩罚。

$$r(\boldsymbol{\beta}) = \begin{cases} \|\boldsymbol{\beta}\|_{1,1} = \sum_{i=1}^{n}\sum_{j=1}^{K}|\beta_{ij}| \\ \frac{1}{2}\|\boldsymbol{\beta}\|_F^2 = \frac{1}{2}\sum_{i=1}^{n}\sum_{j=1}^{K}\beta_{ij}^2 \\ \frac{1-\rho}{2}\|\boldsymbol{\beta}\|_F^2 + \rho\|\boldsymbol{\beta}\|_{1,1} \end{cases}$$

[a] 实现的程序见例1

例子 18

分别使用 l_1 惩罚与不使用惩罚项来做二元 logistic 回归模型，在 iris 数据集上[a]。

带惩罚项的二元 logistic 回归模型的目标函数如下：

$$\arg\min_{\boldsymbol{\beta}} -C\sum_{i=1}^{n}[y_i \log(\hat{p}_k(\boldsymbol{X}_i)) + (1-y_i)\log(1-\hat{p}_k(\boldsymbol{X}_i))] + r(\boldsymbol{\beta})$$

其中的 $\hat{p}_k(\boldsymbol{X}_i)$ 表达如下：

$$\hat{p}_k(\boldsymbol{X}_i) = \exp\{\boldsymbol{X}_i\boldsymbol{\beta}_k\}/(1+\exp\{\boldsymbol{X}_i\boldsymbol{\beta}_l\})$$

[a] 实现的程序见例1

例子 19

对于多个类别的数据，建立 Logistic 回归模型进

行分类，分别比较 ovr 方法和 multibinomial 方法下的分类效果[a]。

[a]实现的程序见例1

例子 20

使用泊松回归来拟合数据[a]。

[a]实现的程序见例1

例子 21

使用稳健回归来拟合数据，RANSAC 算法[a]。

[a]实现的程序见例1

例子 22

使用稳健回归来拟合数据，Theil-Sen 算法[a]。

[a]实现的程序见例1

例子 23

使用稳健回归模型来拟合数据，Huber Regression 算法[a]。

HuberRegression 的目标函数如下：

$$\arg\min_{\beta,\sigma} \sum_{i=1}^{n} (\sigma + H_\varepsilon(\frac{X_i^T\beta - y_i}{\sigma})\sigma) + \alpha\|\beta\|_2^2$$

其中的 $H_\varepsilon(\cdot)$ 如下定义：

$$H_\varepsilon(z) = \begin{cases} z^2, & |z| < \varepsilon \\ 2\varepsilon|z| - \varepsilon^2, & otherwise \end{cases}$$

[a]实现的程序见例1

例子 24

使用分位数回归模型来拟合数据[a]。

分位数回归的目标函数如下：

$$\arg\min_{\beta} \frac{1}{n} \sum_{i} PB_q(y_i - X_i^T \beta) + \alpha \|\beta\|_1$$

其中的 $PB_q(\cdot)$ 表示如下：

$$PB_q(t) = \begin{cases} qt, & t > 0 \\ 0, & t = 0 \\ (q-1)t, & t < 0 \end{cases}$$

[a] 实现的程序见例1

1 代码

Python 代码 1

```
# 导入操作系统库
import os
# 更改工作目录
os.chdir(r"D:\softwares\applied statistics\pythoncodelearning\chap1\sourcecode")
# 导入绘图库
import matplotlib.pyplot as plt
# 导入糖尿病数据集
from sklearn.datasets import load_diabetes
# 导入线性回归模型
from sklearn.linear_model import LinearRegression
# 导入回归模型评价函数，均方误差和 R 方
from sklearn.metrics import mean_squared_error, r2_score
# 导入绘图库中的字体管理包
from matplotlib import font_manager
# 实现中文字符正常显示
font = font_manager.FontProperties(fname=r"C:\Windows\Fonts\SimKai.ttf")
# 使用 seaborn 风格绘图
plt.style.use("seaborn-v0_8")
```

```
# 读取数据
diabetes_X, diabetes_y = load_diabetes(return_X_y=True, as_frame=True)
# 取出第三列
diabetes_X = diabetes_X["bmi"]
# 划分数据集，留下 20 个作为测试集
diabetes_X_train = diabetes_X.iloc[:-20,]
diabetes_X_test = diabetes_X.iloc[-20:,]
diabetes_y_train = diabetes_y.iloc[:-20,]
diabetes_y_test = diabetes_y.iloc[-20:,]
# 构造回归模型
regr = LinearRegression()
# 模型拟合，X 必须是二维数据集
regr.fit(diabetes_X_train.values.reshape(-1,1), diabetes_y_train)
# 在测试上做预测，X 必须是二维数据集
diabetes_y_pred = regr.predict(diabetes_X_test.values.reshape(-1,1))
# 估计系数
print("Coefficients: \n", regr.coef_)
# 均方误差
print("Mean squared error: {}".format(mean_squared_error(diabetes_y_test,␣
 ↪diabetes_y_pred)))
# R2
print("Coefficient of determination: {}".format(r2_score(diabetes_y_test,␣
 ↪diabetes_y_pred)))
# 绘制图形
fig, ax = plt.subplots(figsize=(6,6))
# 测试集的散点
ax.scatter(diabetes_X_test, diabetes_y_test, color="black")
# 预测结果线条
ax.plot(diabetes_X_test, diabetes_y_pred, color="blue", linewidth=3)
plt.show()
fig.savefig("../codeimage/code1.pdf")

Coefficients:
 [938.23786125]
Mean squared error: 2548.07239872597
Coefficient of determination: 0.47257544798227147
```

Python 代码 2

```
# 导入操作系统库
import os
# 更改工作目录
os.chdir(r"D:\softwares\applied statistics\pythoncodelearning\chap1\sourcecode")
```

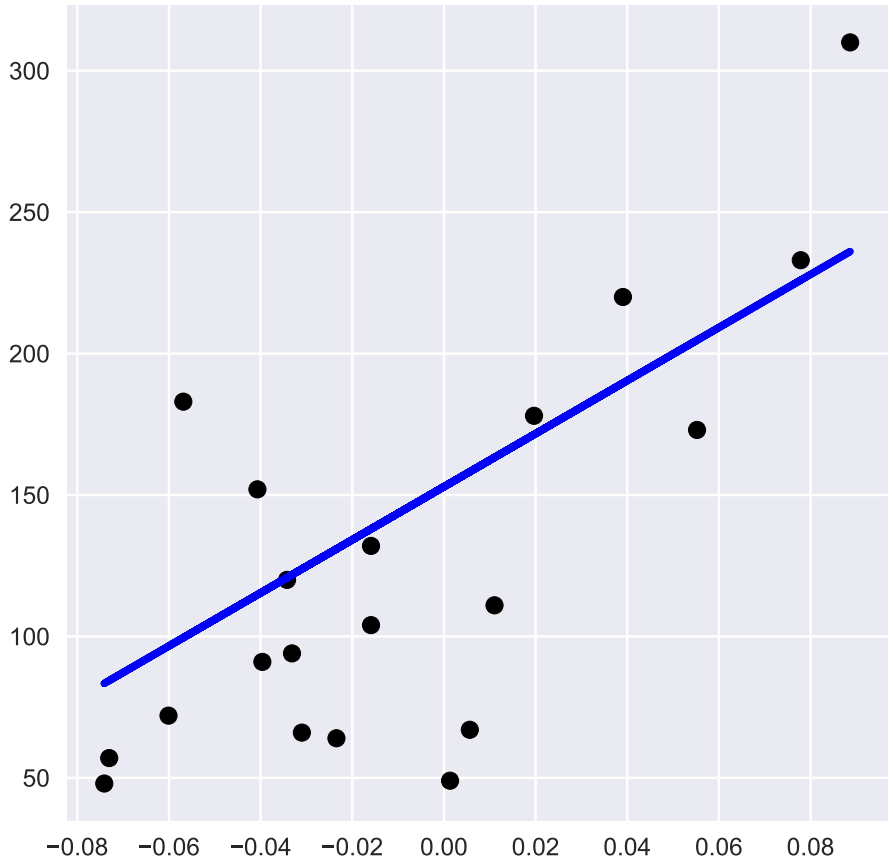

图 1: code1

```python
# 导入基础计算库
import numpy as np
# 导入绘图库
import matplotlib.pyplot as plt
# 导入数据集划分工具
from sklearn.model_selection import train_test_split
# 导入线性回归模型
from sklearn.linear_model import LinearRegression
# 导入回归模型评价函数，均方误差和 R 方
from sklearn.metrics import mean_squared_error, r2_score
# 导入绘图库中的字体管理包
from matplotlib import font_manager
# 实现中文字符正常显示
font = font_manager.FontProperties(fname=r"C:\Windows\Fonts\SimKai.ttf")
# 使用 seaborn 风格绘图
plt.style.use("seaborn-v0_8")
# 设置随机数种子
np.random.seed(42)
# 设置样本量和 X 的维度
n_samples, n_features = 200, 50
# 生成 X
X = np.random.randn(n_samples, n_features)
# 设置真实系数
true_coef = 3 * np.random.randn(n_features)
# 对系数做非负稀疏限制
true_coef[true_coef < 0] = 0
# 生成 y
y = np.dot(X, true_coef) + 5 * np.random.normal(
    size=(n_samples,)
)
# 划分数据集
X_train, X_test, y_train, y_test = train_test_split(
    X, y, test_size=0.5
)
# 构造非负回归模型
reg_nnls = LinearRegression(positive=True)
# 模型拟合
reg_nnls.fit(X_train, y_train)
# 预测
y_pred_nnls = reg_nnls.predict(X_test)
# R 方
r2_score_nnls = r2_score(y_test, y_pred_nnls)
# MSE
```

```
mse_nnls = mean_squared_error(y_test, y_pred_nnls)
print("NNLS R2 score", r2_score_nnls)
print("NNLS MSE", mse_nnls)
# 拟合 OLS 模型
reg_ols = LinearRegression()
# 模型拟合
reg_ols.fit(X_train, y_train)
# 预测
y_pred_ols =  reg_ols.predict(X_test)
# Rfang
r2_score_ols = r2_score(y_test, y_pred_ols)
# mse
mse_ols = mean_squared_error(y_test, y_pred_ols)
print("OLS R2 score", r2_score_ols)
print("OLS mse", mse_ols)
# 比较非负最小二乘的拟合系数和 OLS 的拟合系数
fig, ax = plt.subplots(figsize=(6,6))
# 两组系数的散点图
ax.scatter(reg_ols.coef_, reg_nnls.coef_, marker=".")
# 获取 xy 轴的范围
low_x, high_x = ax.get_xlim()
low_y, high_y = ax.get_ylim()
low = max(low_x, low_y)
high = min(high_x, high_y)
# 绘制系数的拟合回归线
ax.plot([low, high], [low, high], ls="--", c=".3", alpha=0.5)
# 绘制横纵轴标签
ax.set_xlabel(
    "OLS 拟合的回归系数",
    fontweight="bold",
    fontproperties=font,
    fontsize=14
)
ax.set_ylabel(
    "NNLS 拟合的回归系数",
    fontweight="bold",
    fontproperties=font,
    fontsize=14
)
plt.show()
fig.savefig("../codeimage/code2.pdf")

NNLS R2 score 0.8225220806196525
NNLS MSE 39.58532304418648
```

```
OLS R2 score 0.7436926291700342
OLS mse 57.16773167239297
```

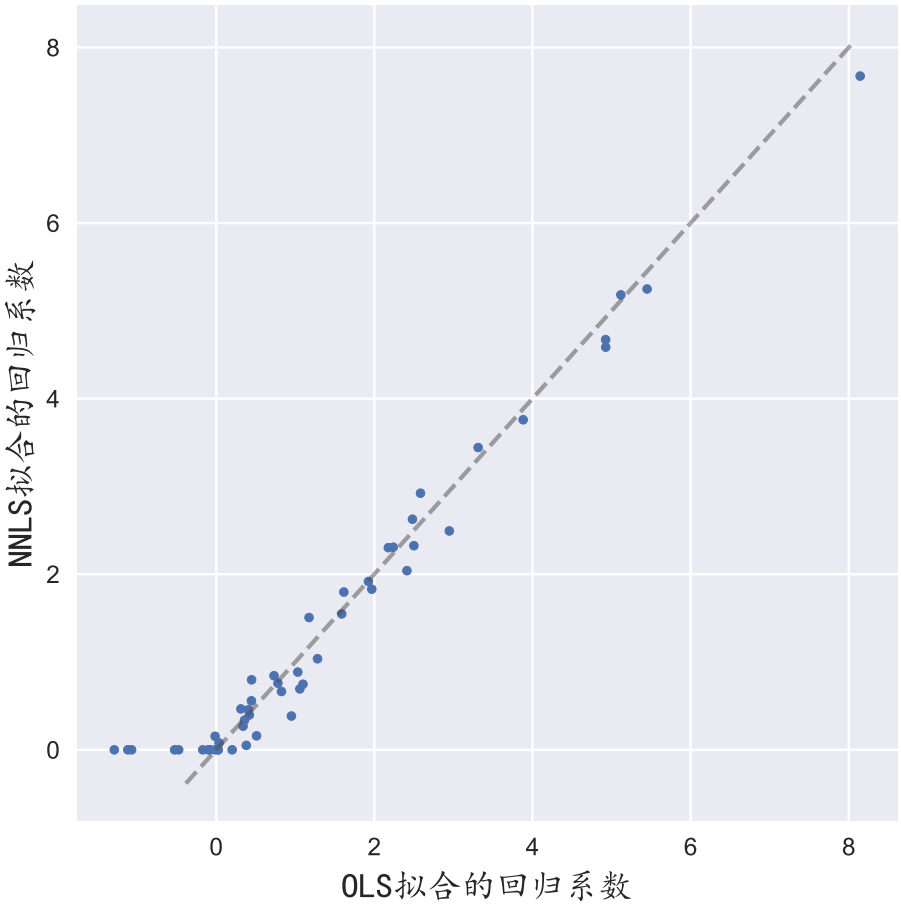

图 2: code2

Python 代码 3

```python
# 导入操作系统库
import os
# 更改工作目录
os.chdir(r"D:\softwares\applied statistics\pythoncodelearning\chap1\sourcecode")
# 导入基础计算库
import numpy as np
# 导入绘图库
import matplotlib.pyplot as plt
# 导入线性回归模型
from sklearn.linear_model import Ridge
```

```
# 导入绘图库中的字体管理包
from matplotlib import font_manager
# 实现中文字符正常显示
font = font_manager.FontProperties(fname=r"C:\Windows\Fonts\SimKai.ttf")
# 使用 seaborn 风格绘图
plt.style.use("seaborn-v0_8")
# 生成 Hilbert 矩阵作为 X
X = 1.0 / (np.arange(1, 11) + np.arange(0, 10)[:, np.newaxis])
y = np.ones(10)
# 设置不同的惩罚系数 alpha 的个数
n_alphas = 200
# alpha 系数向量
alphas = np.logspace(-10, -2, n_alphas)
# 用来存储估计系数的列表
coefs = []
for a in alphas:
    # 构造岭回归模型
    ridge = Ridge(alpha=a, fit_intercept=False)
    # 模型拟合
    ridge.fit(X, y)
    # 将估计的系数放到列表中
    coefs.append(ridge.coef_)
# 开始绘图
fig, ax = plt.subplots(figsize=(6,6))
# coefs 是一个二维列表，每一列是同一个 alpha 下同一个变量前的系数估计量
ax.plot(alphas, coefs)
# 最横轴做 log 变换
ax.set_xscale("log")
# 翻转 X 轴
ax.set_xlim(ax.get_xlim()[::-1])
# 设置横纵轴标签
plt.xlabel("惩罚系数 alpha 的对数值", fontproperties=font, fontsize=12)
plt.ylabel("系数的估计值（路径）", fontproperties=font, fontsize=12)
plt.axis("tight")
plt.show()
fig.savefig("../codeimage/code3.pdf")
```

Python 代码 4

```
# 导入操作系统库
import os
# 更改工作目录
os.chdir(r"D:\softwares\applied statistics\pythoncodelearning\chap1\sourcecode")
# 导入基础计算库
```

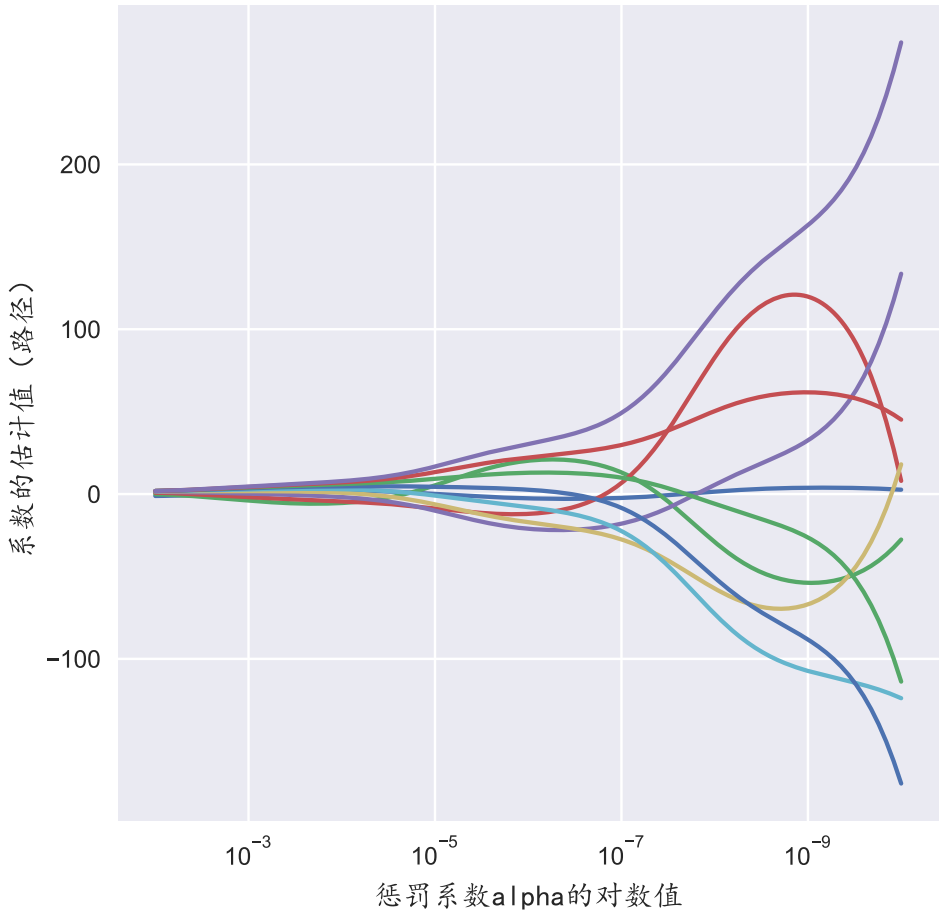

图 3: code3

```python
import numpy as np
# 导入绘图库
import matplotlib.pyplot as plt
# 导入线性回归模型
from sklearn.linear_model import RidgeClassifier
# 导入数据集划分工具
from sklearn.model_selection import train_test_split
# 导入混淆矩阵
from sklearn.metrics import ConfusionMatrixDisplay, confusion_matrix
# 导入绘图库中的字体管理包
from matplotlib import font_manager
# 实现中文字符正常显示
font = font_manager.FontProperties(fname=r"C:\Windows\Fonts\SimKai.ttf")
# 使用 seaborn 风格绘图
plt.style.use("seaborn-v0_8")
# 维度
p = 200
# 样本量
samplesize = 100
# 真实的系数值
coef = np.random.normal(size=p)
# 生成 X 数据
x = np.random.randn(samplesize, p)
# 生成概率
p = np.exp(x.dot(coef))/(1+np.exp(x.dot(coef)))
# 生成 y
y = np.random.binomial(1, p)
x_train, x_test, y_train, y_test = train_test_split(
    x, y, test_size=0.4, random_state=10
)
# 构造 Ridge 分类器
clf = RidgeClassifier(tol=1e-2, solver="sparse_cg")
# 模型拟合
clf.fit(x_train, y_train)
# 预测
y_pred = clf.predict(x_test)
# 混淆矩阵
res = confusion_matrix(y_test, y_pred, labels=[0,1])
print("混淆矩阵为: ", res, sep="\n")
# 开始绘图
fig, ax = plt.subplots(figsize=(6, 6))
# 绘制混淆矩阵图
ConfusionMatrixDisplay.from_predictions(y_test, y_pred, ax=ax)
```

```
# 设置横纵轴刻度
ax.xaxis.set_ticklabels([0,1])
ax.yaxis.set_ticklabels([0,1])
plt.show()
fig.savefig("../codeimage/code4.pdf")
```

```
混淆矩阵为:
[[14 10]
 [ 7  9]]
```

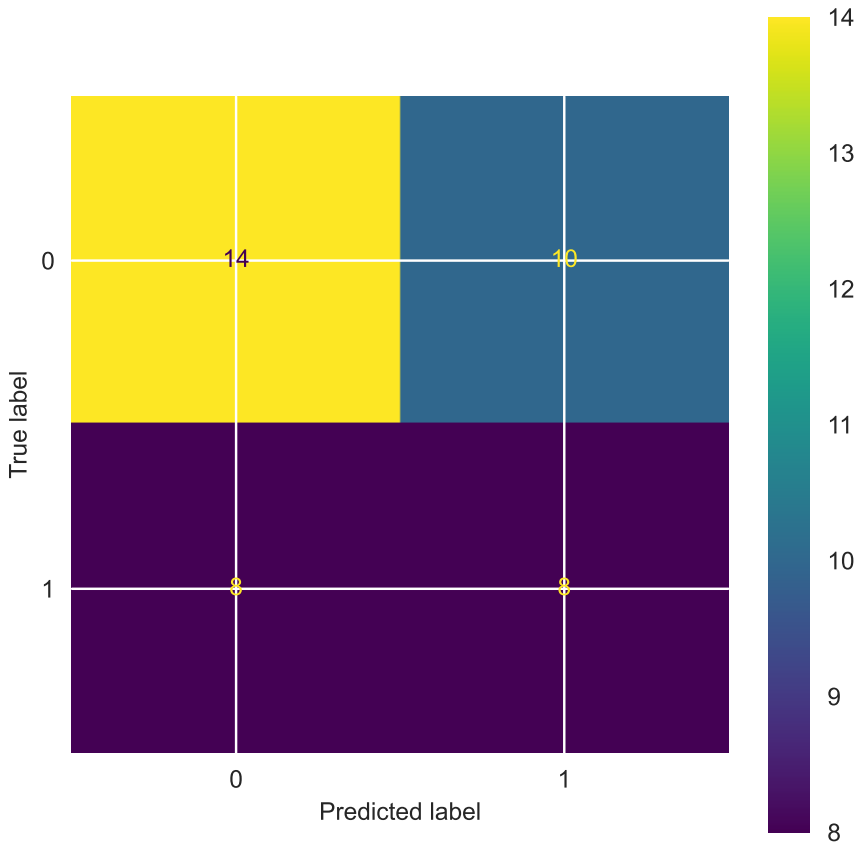

图 4: code4

Python 代码 5

```
# 导入操作系统库
import os
# 更改工作目录
```

```
os.chdir(r"D:\softwares\applied statistics\pythoncodelearning\chap1\sourcecode")
# 导入基础计算库
import numpy as np
# 导入绘图库
import matplotlib.pyplot as plt
# 导入数据集生成工具
from sklearn.datasets import make_regression
# 导入线性回归模型
from sklearn.linear_model import RidgeCV
# 导入数据集划分工具
from sklearn.model_selection import train_test_split
# 导入模型评估的工具
from sklearn.metrics import mean_squared_error, r2_score
# 导入绘图库中的字体管理包
from matplotlib import font_manager
# 实现中文字符正常显示
font = font_manager.FontProperties(fname=r"C:\Windows\Fonts\SimKai.ttf")
# 使用 seaborn 风格绘图
plt.style.use("seaborn-v0_8")
# 生成 Hilbert 矩阵作为 X
X, y = make_regression(
    n_features=20, # 维度
    n_samples=300, # 样本量
    n_informative=10 # 有效显著变量的个数
)
# 划分数据集
x_train, x_test, y_train, y_test = train_test_split(
    X, y, test_size=0.3, random_state=10
)
# 设置不同的惩罚系数 alpha 的个数
n_alphas = 200
# alpha 系数向量
alphas = np.linspace(0.01, 2, n_alphas)
# 构造岭回归模型
ridge = RidgeCV(
    alphas=alphas, # 惩罚系数
    fit_intercept=False, # 不拟合
    cv=5 # 五折交叉验证
)
# 模型拟合
ridge.fit(x_train, y_train)
# 预测
y_pred = ridge.predict(x_test)
```

```
# 测试集上的 mse
mse = mean_squared_error(y_test, y_pred)
r2 = r2_score(y_test, y_pred)
print("在测试集上的 MSE 为: ", mse)
print("在测试集上的 R 方为: ", r2)
```

```
在测试集上的 MSE 为:  0.00012988709824135518
在测试集上的 R 方为:  0.9999999974707636
```

Python 代码 6

```
# 导入操作系统库
import os
# 更改工作目录
os.chdir(r"D:\softwares\applied statistics\pythoncodelearning\chap1\sourcecode")
# 导入基础计算库
import numpy as np
# 导入绘图库
import matplotlib.pyplot as plt
# 导入线性回归模型
from sklearn.linear_model import Lasso, ElasticNet
# 导入模型评估的工具
from sklearn.metrics import mean_squared_error, r2_score
# 导入绘图库中的字体管理包
from matplotlib import font_manager
# 实现中文字符正常显示
font = font_manager.FontProperties(fname=r"C:\Windows\Fonts\SimKai.ttf")
# 使用 seaborn 风格绘图
plt.style.use("seaborn-v0_8")
# 设置样本量和维度
n_samples, n_features = 50, 100
# 设置随机数
np.random.seed(10)
# 生成 X
X = np.random.randn(n_samples, n_features)
# 生成真实的系数
idx = np.arange(n_features)
coef = (-1) ** idx * np.exp(-idx / 10)
# 对系数产生稀疏性
coef[10:] = 0
# 生成 y
y = np.dot(X, coef) + 0.01 * np.random.normal(size=n_samples)
# 划分数据集
n_samples = X.shape[0]
```

```
X_train, y_train = X[: n_samples // 2], y[: n_samples // 2]
X_test, y_test = X[n_samples // 2 :], y[n_samples // 2 :]
# 设置 lasso 的惩罚系数
alpha = 0.1
# 构建 lasso 模型
lasso = Lasso(alpha=alpha)
# 模型拟合
lasso.fit(X_train, y_train)
# 预测
y_pred = lasso.predict(X_test)
# R 方测试集
r2 = r2_score(y_test, y_pred)
# MSE 测试集
mse = mean_squared_error(y_test, y_pred)
print("r^2 on test data : {}".format(r2))
print("mse on test data : {}".format(mse))
# 构建弹性网模型
enet = ElasticNet(alpha=alpha, l1_ratio=0.7)
# 模型拟合
enet.fit(X_train, y_train)
# 预测
y_pred_enet = enet.predict(X_test)
r2_enet = r2_score(y_test, y_pred_enet)
mse_enet = mean_squared_error(y_test, y_pred_enet)
print("r^2 on test data : {}".format(r2_enet))
print("mse on test data : {}".format(mse_enet))
# 开始绘图
fig, ax = plt.subplots(figsize=(6,6))
m, s, _ = ax.stem(
    np.where(enet.coef_)[0],
    enet.coef_[enet.coef_ != 0],
    markerfmt="x",
    label="Elastic net coefficients",
)
plt.setp([m, s], color="#2ca02c")
m, s, _ = ax.stem(
    np.where(lasso.coef_)[0],
    lasso.coef_[lasso.coef_ != 0],
    markerfmt="x",
    label="Lasso coefficients",
)
plt.setp([m, s], color="#ff7f0e")
ax.stem(
```

```
    np.where(coef)[0],
    coef[coef != 0],
    label="true coefficients",
    markerfmt="bx",
)
ax.legend(loc="best")
ax.set_title(
    "Lasso $R^2$: %.3f, Elastic Net $R^2$: %.3f" % (r2, r2_enet)
)
plt.show()
fig.savefig("../codeimage/code5.pdf")
```

```
r^2 on test data : -0.5148835447539641
mse on test data : 5.842405459587561
r^2 on test data : -0.33978748237581957
mse on test data : 5.167117781975022
```

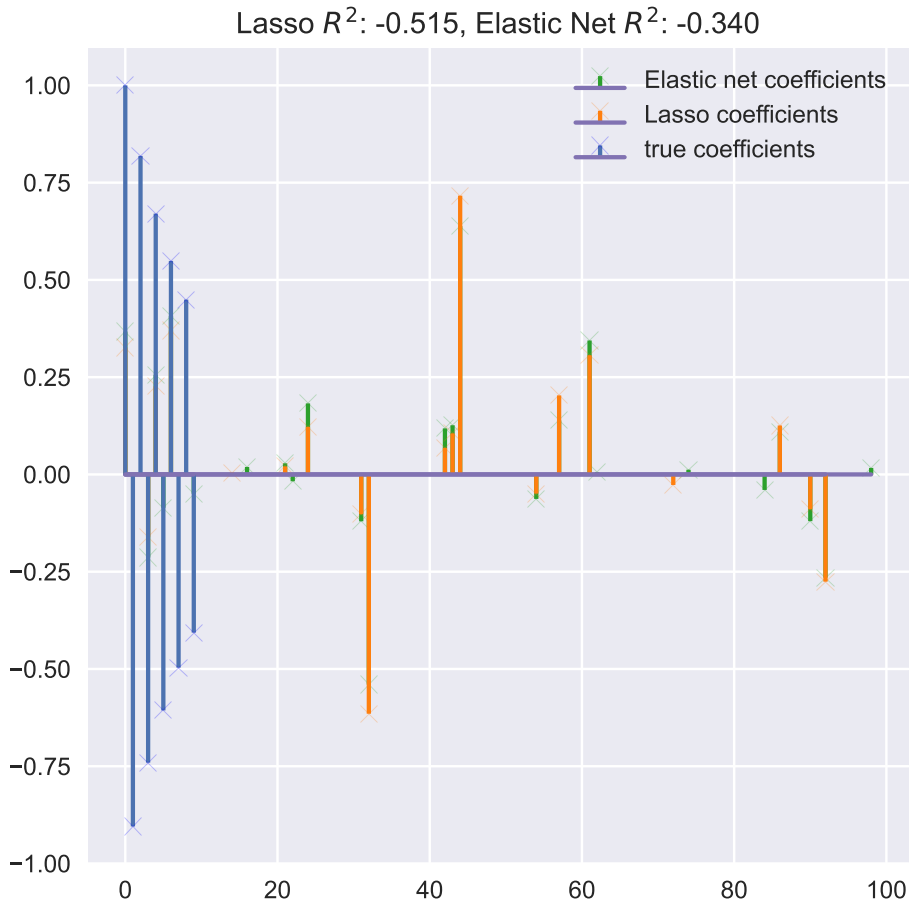

图 5: code5

Python 代码 7

```
# 导入操作系统库
import os
# 更改工作目录
os.chdir(r"D:\softwares\applied statistics\pythoncodelearning\chap1\sourcecode")
# 导入基础计算库
import numpy as np
# 导入绘图库
import matplotlib.pyplot as plt
# 导入线性回归模型
from sklearn.linear_model import Ridge, LassoCV
# 导入管道处理工具
from sklearn.pipeline import make_pipeline
# 导入数值计算库
import scipy as sp
# 导入数据分析库
import pandas as pd
# 导入模型评估的工具
from sklearn.metrics import mean_squared_error, r2_score
# 导入数据集获取工具
from sklearn.datasets import fetch_openml
# 导入元回归估计器
from sklearn.compose import TransformedTargetRegressor
# 导入数据集划分工具
from sklearn.model_selection import train_test_split
# 导入模型评估工具
from sklearn.metrics import median_absolute_error, PredictionErrorDisplay
# 导入列转换工具
from sklearn.compose import make_column_transformer
# 导入 one-hot 编码工具
from sklearn.preprocessing import OneHotEncoder
# 导入统计绘图库
import seaborn as sns
# 导入绘图库中的字体管理包
from matplotlib import font_manager
# 实现中文字符正常显示
font = font_manager.FontProperties(fname=r"C:\Windows\Fonts\SimKai.ttf")
# 使用 seaborn 风格绘图
plt.style.use("seaborn-v0_8")

# 导入数据集
survey = fetch_openml(data_id=534, as_frame=True, parser="pandas")
# 协变量 X
```

```
X = survey.data[survey.feature_names]
print("解释变量 X 的描述性统计表如下：", X.describe(include="all"), sep="\n")
# 响应变量 y
y = survey.target.values.ravel()
print("y 的前五行为：", survey.target.head(), sep="\n")
```

```
解释变量 X 的描述性统计表如下：
         EDUCATION SOUTH   SEX  EXPERIENCE       UNION         AGE   RACE  \
count   534.000000   534   534  534.000000         534  534.000000    534
unique         NaN     2     2         NaN           2         NaN      3
top            NaN    no  male         NaN  not_member         NaN  White
freq           NaN   378   289         NaN         438         NaN    440
mean     13.018727   NaN   NaN   17.822097         NaN   36.833333    NaN
std       2.615373   NaN   NaN   12.379710         NaN   11.726573    NaN
min       2.000000   NaN   NaN    0.000000         NaN   18.000000    NaN
25%      12.000000   NaN   NaN    8.000000         NaN   28.000000    NaN
50%      12.000000   NaN   NaN   15.000000         NaN   35.000000    NaN
75%      15.000000   NaN   NaN   26.000000         NaN   44.000000    NaN
max      18.000000   NaN   NaN   55.000000         NaN   64.000000    NaN

       OCCUPATION SECTOR     MARR
count         534    534      534
unique          6      3        2
top         Other  Other  Married
freq          156    411      350
mean          NaN    NaN      NaN
std           NaN    NaN      NaN
min           NaN    NaN      NaN
25%           NaN    NaN      NaN
50%           NaN    NaN      NaN
75%           NaN    NaN      NaN
max           NaN    NaN      NaN
y 的前五行为：
0    5.10
1    4.95
2    6.67
3    4.00
4    7.50
Name: WAGE, dtype: float64
```

```
# 划分数据集
X_train, X_test, y_train, y_test = train_test_split(
    X, y, random_state=42, test_size=0.25
)
# 复制一份训练集
```

```
train_dataset = X_train.copy()
# 插入一列数据，作为第一列
train_dataset.insert(0, "WAGE", y_train)
# 绘制矩阵散点图
fig = sns.PairGrid(train_dataset)
# 对角线上的图形
fig.map_diag(sns.kdeplot)
# 非对角线上的图形
fig.map_offdiag(sns.scatterplot)
fig.savefig("../codeimage/code6.pdf")

# 查看下数据变量的变量情况
print("数据集变量的情况：")
survey.data.info()

数据集变量的情况：
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 534 entries, 0 to 533
Data columns (total 10 columns):
 #   Column      Non-Null Count  Dtype
---  ------      --------------  -----
 0   EDUCATION   534 non-null    int64
 1   SOUTH       534 non-null    category
 2   SEX         534 non-null    category
 3   EXPERIENCE  534 non-null    int64
 4   UNION       534 non-null    category
 5   AGE         534 non-null    int64
 6   RACE        534 non-null    category
 7   OCCUPATION  534 non-null    category
 8   SECTOR      534 non-null    category
 9   MARR        534 non-null    category
dtypes: category(7), int64(3)
memory usage: 17.2 KB

# 对分类变量进行 one-hot 编码
# 分类变量的列名
categorical_columns = [
    "RACE", "OCCUPATION", "SECTOR",
    "MARR", "UNION", "SEX", "SOUTH"
]
# 数值变量的列名
numerical_columns = ["EDUCATION", "EXPERIENCE", "AGE"]
# 进行分类变量列之间的 one-hot 编码
preprocessor = make_column_transformer(
    (
```

```
        OneHotEncoder(drop="if_binary"), # one-hot 编码
        categorical_columns # 对这些分类变量
    ),
    remainder="passthrough", # 保留非分类变量
    verbose_feature_names_out=False
)
# 构造岭回归模型，惩罚系数非常小，接近于 OLS
model = make_pipeline(
    preprocessor, # preprocess 对象
    TransformedTargetRegressor(
        regressor=Ridge(alpha=1e-10), # 模型
        func=np.log10, # 它将作用于目标变量 wage 上
        inverse_func=sp.special.exp10
    ),
)
# 模型拟合
model.fit(X_train, y_train)
# 预测
y_train_fit = model.predict(X_train)
# 训练集上的绝对误差的中位数
mae_train = median_absolute_error(y_train, y_train_fit)
# 预测
y_pred = model.predict(X_test)
# 测试集上的绝对误差中位数
mae_test = median_absolute_error(y_test, y_pred)
scores = {
    "MedAE on training set": "{:.2f} $/hour".format(mae_train),
    "MedAE on testing set": "{:.2f} $/hour".format(mae_test)
}
# 开始绘图
fig2, ax = plt.subplots(figsize=(6, 6))
display = PredictionErrorDisplay.from_predictions(
    y_test, y_pred,
    kind="actual_vs_predicted",
    ax=ax,
    scatter_kwargs={"alpha": 0.5}
)
ax.set_title("Ridge model, small regularization")
# 添加图例
for name, score in scores.items():
    ax.plot([], [], " ", label=f"{name}: {score}")
ax.legend(loc="lower right")
plt.tight_layout()
```

```
plt.show()
fig2.savefig("../codeimage/code7.pdf")

# 查看下岭回归的系数估计值
# 系数对应的变量名
feature_names = model[:-1].get_feature_names_out()
# 构造 dataframe
coefs = pd.DataFrame(
    model[-1].regressor_.coef_,
    columns=["Coefficients"],
    index=feature_names,
)
print("系数估计值为: ", coefs, sep="\n")
# 图形展示系数估计值
fig3, ax = plt.subplots(figsize=(6,6))
# 水平柱状图
coefs.plot(kind="barh", ax=ax)
# 设置标题
ax.set_title("Ridge model, small regularization")
# 绘制一条竖直线
ax.axvline(x=0, color=".5")
# 不显示图例，默认显示
ax.legend([])
# 设置横纵标签
ax.set_xlabel("Raw coefficient values")
plt.show()
fig3.savefig("../codeimage/code8.pdf")
```

```
系数估计值为:
                         Coefficients
RACE_Hispanic               -0.013558
RACE_Other                  -0.009114
RACE_White                   0.022555
OCCUPATION_Clerical          0.000062
OCCUPATION_Management        0.090545
OCCUPATION_Other            -0.025084
OCCUPATION_Professional      0.071981
OCCUPATION_Sales            -0.046619
OCCUPATION_Service          -0.091036
SECTOR_Construction         -0.000188
SECTOR_Manufacturing         0.031265
SECTOR_Other                -0.031015
MARR_Unmarried              -0.032405
UNION_not_member            -0.117154
SEX_male                     0.090808
```

```
SOUTH_yes                   -0.033823
EDUCATION                    0.054699
EXPERIENCE                   0.035005
AGE                         -0.030867
```

```
# 使用 lasso 模型来拟合
# lasso 惩罚系数
alphas = np.logspace(-10, 10, 21)
# 构建 lassoCV 模型
model = make_pipeline(
    preprocessor,
    TransformedTargetRegressor(
        regressor=LassoCV(alphas=alphas, max_iter=100000),
        func=np.log10,
        inverse_func=sp.special.exp10,
    ),
)
# 模型拟合
model.fit(X_train, y_train)
print("所选的 lasso 模型对应的系数为: ", model[-1].regressor_.alpha_, sep="\n")
# 模型预测训练集
y_pred_lasso_train = model.predict(X_train)
mae_train = median_absolute_error(y_train, y_pred_lasso_train)
# 模型预测测试集
y_pred_lasso_test = model.predict(X_test)
mae_test = median_absolute_error(y_test, y_pred_lasso_test)
# 开始绘图
fig4, ax = plt.subplots(figsize=(6, 6))
display = PredictionErrorDisplay.from_predictions(
    y_test, y_pred,
    kind="actual_vs_predicted",
    ax=ax,
    scatter_kwargs={"alpha": 0.5}
)
ax.set_title("Lasso model, optimum regularization")
# 设置图例
for name, score in scores.items():
    ax.plot([], [], " ", label=f"{name}: {score}")
ax.legend(loc="lower right")
plt.show()
fig4.savefig("../codeimage/code9.pdf")
```

```
所选的 lasso 模型对应的系数为:
0.001
```

```
# 绘制系数估计的条行图
```

```
fig5, ax = plt.subplots(figsize=(6,6))
coefs = pd.DataFrame(
    model[-1].regressor_.coef_,
    columns=["Coefficients importance"],
    index=feature_names,
)
coefs.plot(kind="barh", ax=ax)
# 不显示图例，默认显示
ax.legend([])
ax.set_title("Lasso model, optimum regularization, normalized variables")
ax.axvline(x=0, color=".5")
plt.show()
fig5.savefig("../codeimage/code10.pdf")
```

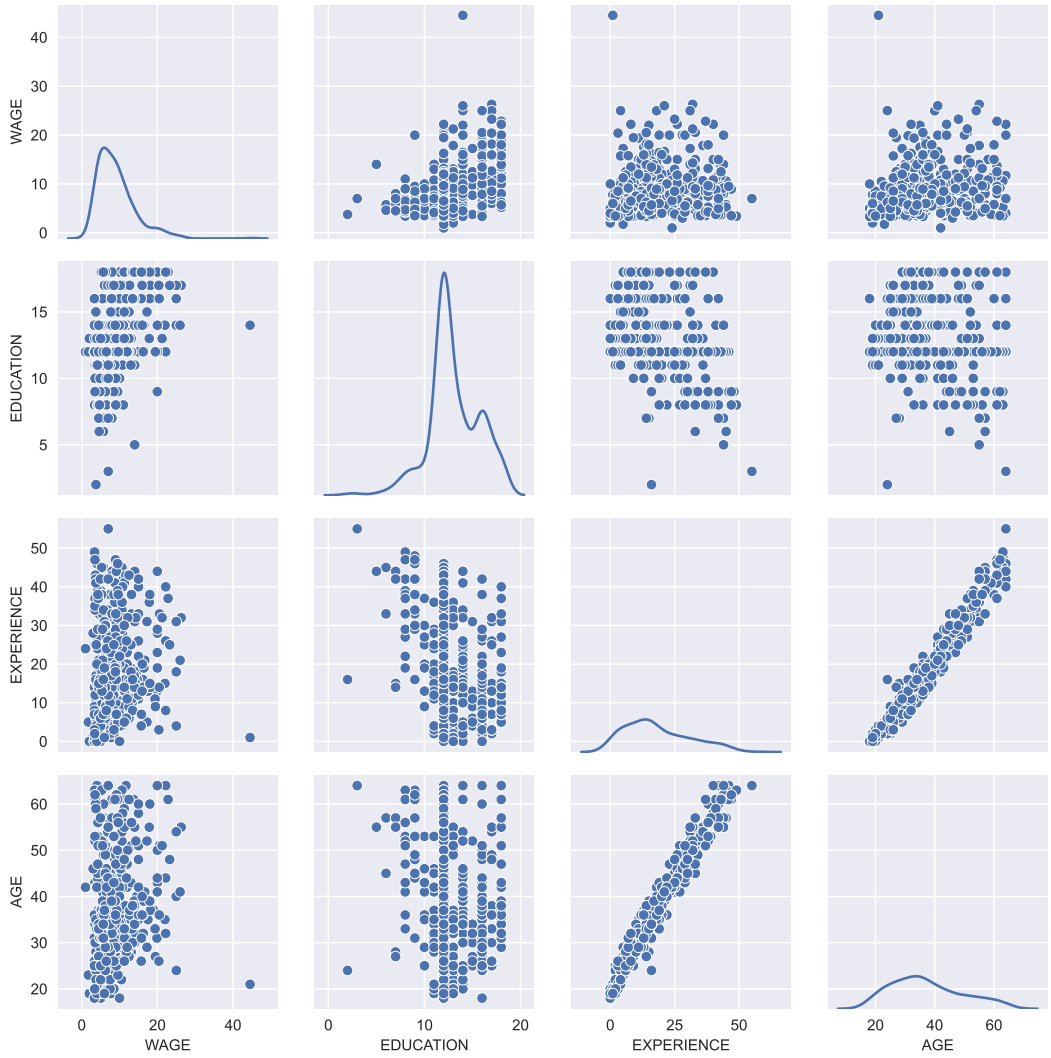

图 6: code6

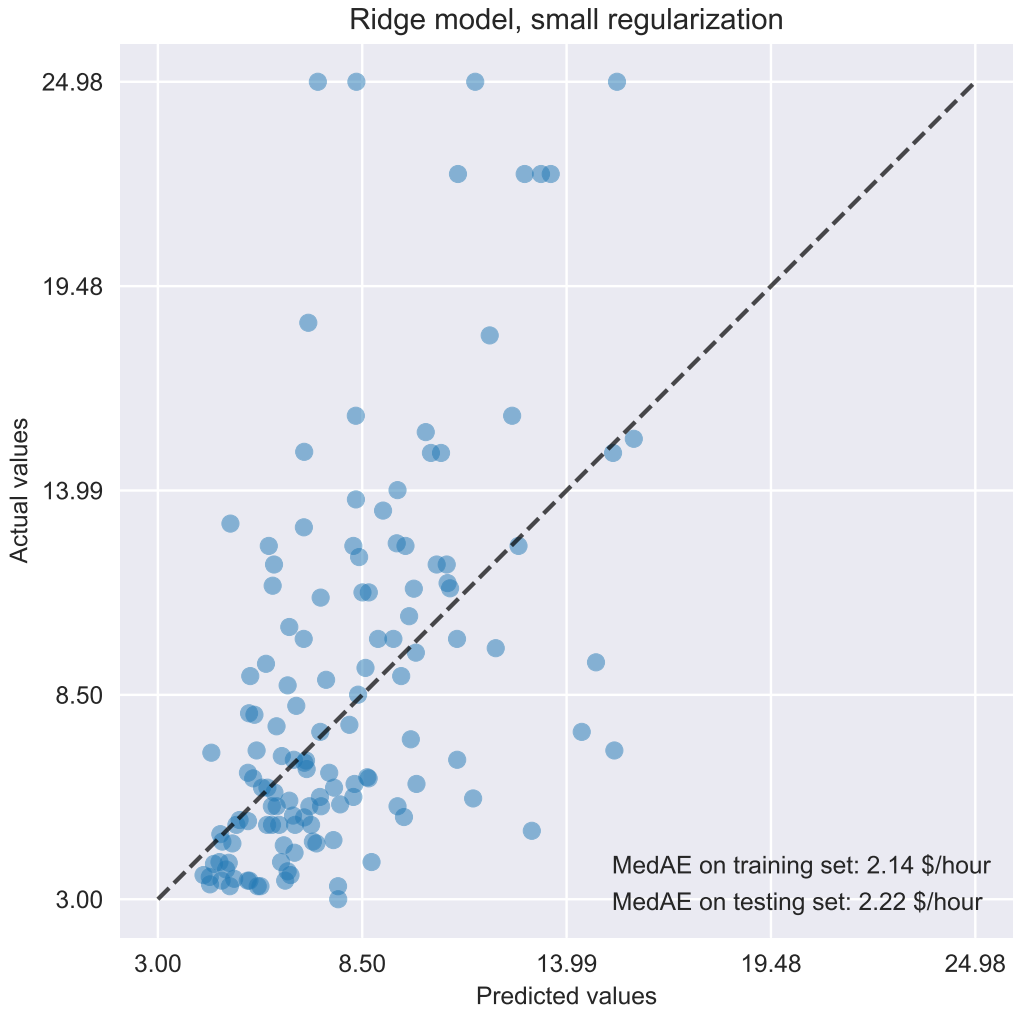
Ridge model, small regularization
24.98
19.48
13.99
8.50
3.00
Actual values
3.00
8.50
13.99
19.48
24.98
Predicted values
MedAE on training set: 2.14 $/hour
MedAE on testing set: 2.22 $/hour

图 7: code7

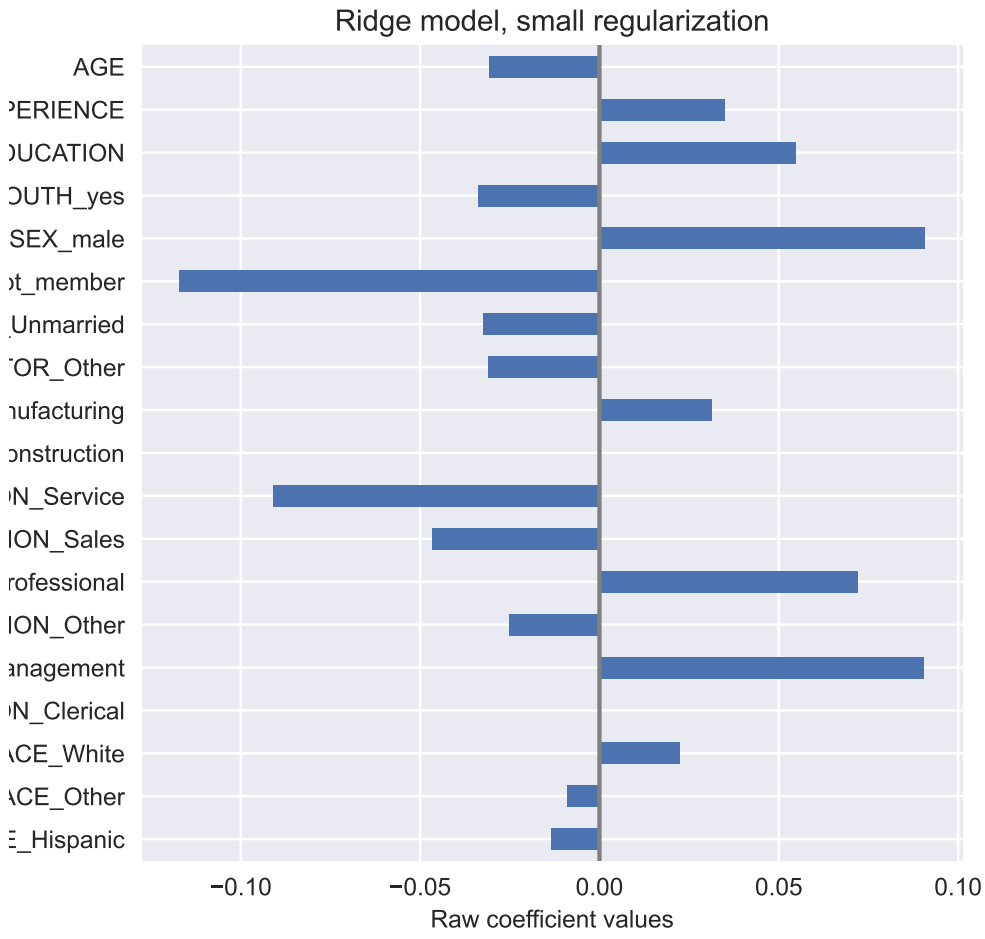
Ridge model, small regularization
AGE
ERIENCE
UCATION
UTH_yes
SEX_male
t_member
Unmarried
OR_Other
ufacturing
onstruction
N_Service
ION_Sales
rofessional
ION_Other
anagement
N_Clerical
CE_White
CE_Other
_Hispanic
−0.10
−0.05
0.00
0.05
0.10
Raw coefficient values

图 8: code8

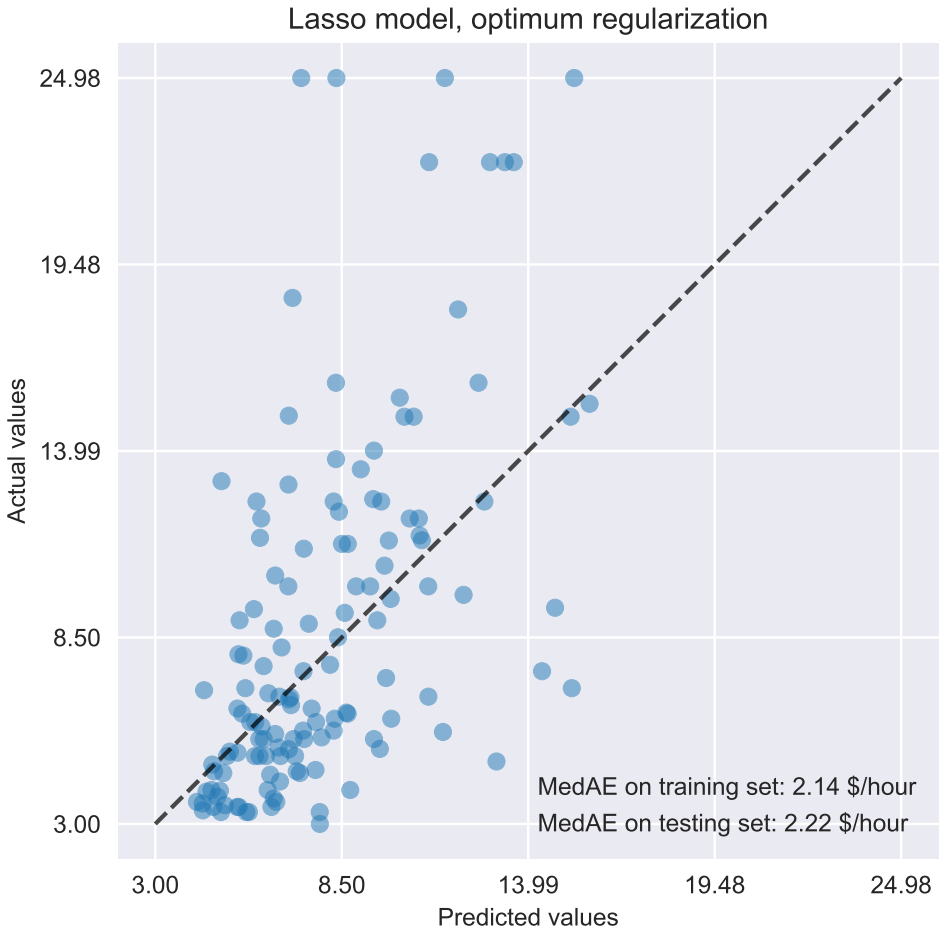
Lasso model, optimum regularization
24.98
19.48
13.99
8.50
3.00
Actual values
3.00
8.50
13.99
19.48
24.98
Predicted values
MedAE on training set: 2.14 $/hour
MedAE on testing set: 2.22 $/hour

图 9: code9

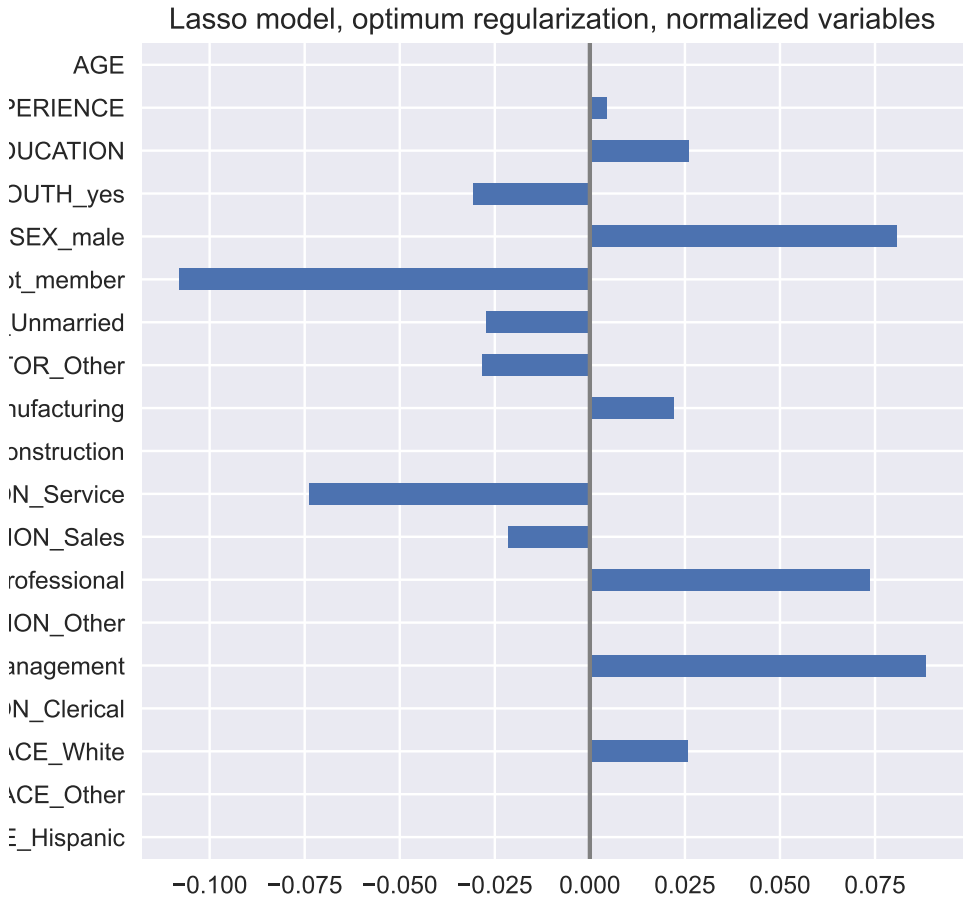
Lasso model, optimum regularization, normalized variables
AGE
ERIENCE
UCATION
UTH_yes
SEX_male
t_member
Unmarried
OR_Other
ufacturing
onstruction
N_Service
ION_Sales
rofessional
ION_Other
anagement
N_Clerical
CE_White
CE_Other
_Hispanic
−0.100
−0.075
−0.050
−0.025
0.000
0.025
0.050
0.075

图 10: code10

Python 代码 8

```
# 导入操作系统库
import os
# 更改工作目录
os.chdir(r"D:\softwares\applied statistics\pythoncodelearning\chap1\sourcecode")
# 导入基础计算库
import numpy as np
# 导入绘图库
import matplotlib.pyplot as plt
# 导入数据分析库
import pandas as pd
# 导入模型评估的工具
# 导入数据集获取工具
from sklearn.datasets import load_diabetes
# 导入标准化处理工具
from sklearn.preprocessing import StandardScaler
# 导入 Lasso 信息准则估计器
from sklearn.linear_model import LassoLarsIC
# 导入管道操作
from sklearn.pipeline import make_pipeline
# 导入时间库
import time
# 导入绘图库中的字体管理包
from matplotlib import font_manager
# 实现中文字符正常显示
font = font_manager.FontProperties(fname=r"C:\Windows\Fonts\SimKai.ttf")
# 使用 seaborn 风格绘图
plt.style.use("seaborn-v0_8")
# 导入数据集
X, y = load_diabetes(return_X_y=True, as_frame=True)
# 在原始数据集中加入一些随机特征，增加变量
np.random.seed(42)
# 特征数
n_random_features = 14
# 生成随机的 X
X_random = pd.DataFrame(
    np.random.randn(X.shape[0], n_random_features),
    columns=[f"random_{i:02d}" for i in range(n_random_features)],
)
# 合并 X
X = pd.concat([X, X_random], axis=1)
# 查看下数据
print(X[X.columns[::3]].head())
```

```
# 计时开始
start_time = time.time()
# 建立 lassoIC 模型，它的 alpha 惩罚系数是自动生成的，无法指定
lasso_lars_aic = make_pipeline(
    StandardScaler(), # 数据标准化
    LassoLarsIC(criterion="aic") # 使用 aic 准则
)
# 模型拟合
lasso_lars_aic.fit(X, y)
# 记录模型使用的 alpha
alpha_aic = lasso_lars_aic[-1].alpha_
# 建立 lassoIC 模型，它的 alpha 惩罚系数是自动生成的，无法指定
lasso_lars_bic = make_pipeline(
    StandardScaler(), # 数据标准化
    LassoLarsIC(criterion="bic") # 使用 aic 准则
)
# 模型拟合
lasso_lars_bic.fit(X, y)
# 拟合时间
fit_time = time.time() - start_time
print("模型拟合的时间为: ", fit_time, sep="\n")
# 记录模型使用的 alpha
alpha_bic = lasso_lars_bic[-1].alpha_
# 将 alpha 和 AIC, BIC 存储起来
results = pd.DataFrame(
    {
        "alphas": lasso_lars_aic[-1].alphas_,
        "AIC criterion": lasso_lars_aic[-1].criterion_,
        "BIC criterion": lasso_lars_bic[-1].criterion_
    }
).set_index("alphas")

# 定义一个函数，选择出最小的 AIC 对应的 alpha
def highlight_min(x):
    x_min = x.min()
    return ["font-weight: bold" if v == x_min else "" for v in x]
# 高亮标记
results.style.apply(highlight_min)
```

```
        age        bp        s3        s6  random_02  random_05  random_08  \
0  0.038076  0.021872 -0.043401 -0.017646   0.647689  -0.234137  -0.469474
1 -0.001882 -0.026328  0.074412 -0.092204  -1.012831  -1.412304   0.067528
2  0.085299 -0.005670 -0.032356 -0.025930  -0.601707  -1.057711   0.208864
3 -0.089063 -0.036656 -0.036038 -0.009362  -1.478522   1.057122   0.324084
```

```
4  0.005383  0.021872  0.008142 -0.046641   0.331263  -0.185659   0.812526

   random_11
0  -0.465730
1   0.110923
2   0.196861
3   0.611676
4   1.003533
模型拟合的时间为:
0.06775593757629395

# 最后，我们可以绘制不同 alpha 值的 AIC 和 BIC 值。
# 图中的垂直线对应于为每个标准选择的 alpha。所选择的  对应于 AIC 或 BIC 准则的最小值。
fig1, ax = plt.subplots(figsize=(6,6))
ax = results.plot(ax=ax)
# 画竖直线
ax.vlines(
    alpha_aic,
    results["AIC criterion"].min(),
    results["AIC criterion"].max(),
    label="alpha: AIC estimate",
    linestyles="--",
    color="tab:blue",
)
ax.vlines(
    alpha_bic,
    results["BIC criterion"].min(),
    results["BIC criterion"].max(),
    label="alpha: BIC estimate",
    linestyle="--",
    color="tab:orange",
)
ax.set_xlabel(r"$\alpha$")
ax.set_ylabel("criterion")
ax.set_xscale("log")
# 展示图例
ax.legend()
ax.set_title(
    f"Information-criterion for model selection (training time {fit_time:.2f}s)"
)
plt.show()
fig1.savefig("../codeimage/code11.pdf")
```

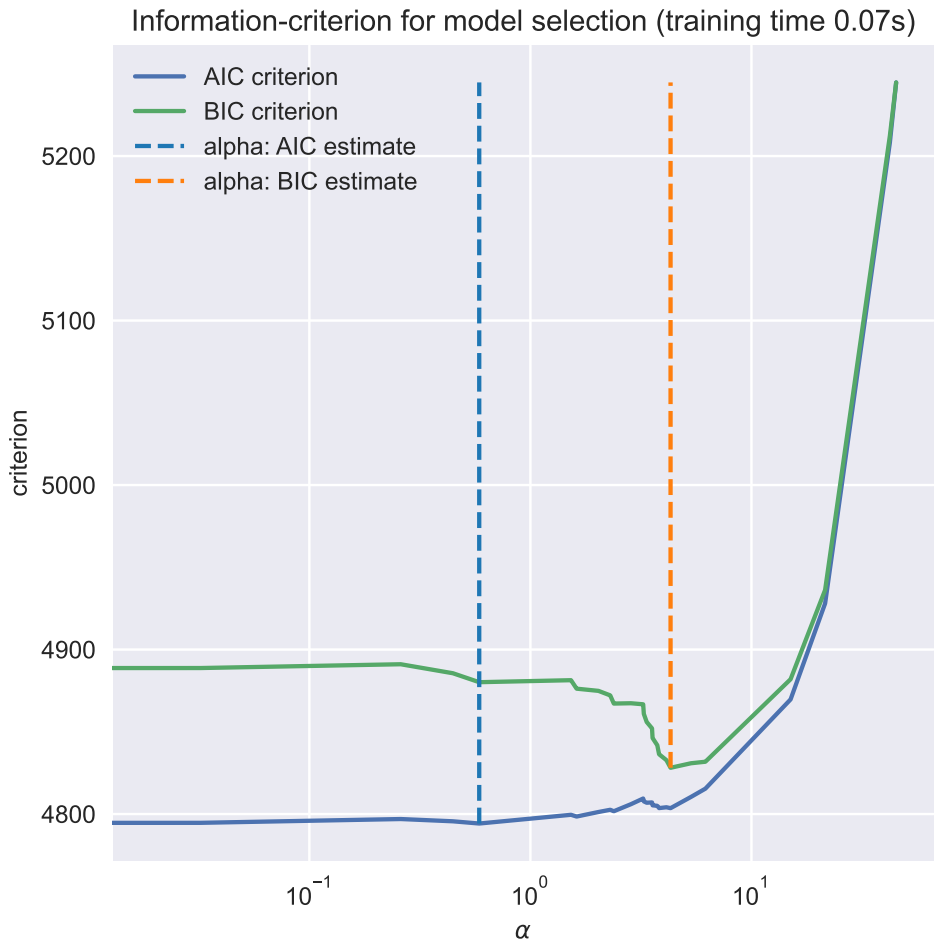
Information-criterion for model selection (training time 0.07s)
AIC criterion
BIC criterion
alpha: AIC estimate
alpha: BIC estimate
criterion
5200
5100
5000
4900
4800
10^{-1}
10^{0}
10^{1}
α

图 11: code11

Python 代码 9

```
# 导入操作系统库
import os
# 更改工作目录
os.chdir(r"D:\softwares\applied statistics\pythoncodelearning\chap1\sourcecode")
# 导入基础计算库
import numpy as np
# 导入绘图库
import matplotlib.pyplot as plt
# 导入数据分析库
import pandas as pd
# 导入模型评估的工具
# 导入数据集获取工具
from sklearn.datasets import load_diabetes
# 导入标准化处理工具
from sklearn.preprocessing import StandardScaler
# 导入 LassoCV 模型
from sklearn.linear_model import LassoCV, LassoLarsCV
# 导入管道操作
from sklearn.pipeline import make_pipeline
# 导入时间库
import time
# 导入绘图库中的字体管理包
from matplotlib import font_manager
# 实现中文字符正常显示
font = font_manager.FontProperties(fname=r"C:\Windows\Fonts\SimKai.ttf")
# 使用 seaborn 风格绘图
plt.style.use("seaborn-v0_8")
# 导入数据集
X, y = load_diabetes(return_X_y=True, as_frame=True)
# 在原始数据集中加入一些随机特征，增加变量
np.random.seed(42)
# 特征数
n_random_features = 14
# 生成随机的 X
X_random = pd.DataFrame(
    np.random.randn(X.shape[0], n_random_features),
    columns=[f"random_{i:02d}" for i in range(n_random_features)],
)
# 合并 X
X = pd.concat([X, X_random], axis=1)
# 查看下数据
print(X[X.columns[::3]].head())
```

```
# 开始计时
start_time = time.time()
# 建立 Lasso CV 模型
model1 = make_pipeline(
    StandardScaler(), LassoCV(cv=20) # 自动赋 alpha 惩罚系数
)
# 模型拟合
model1.fit(X, y)
# 拟合时间
fit_time = time.time() - start_time
ymin, ymax = 2300, 3800
# 获取 lasso 模型
lasso = model1[-1]
fig1, ax = plt.subplots(figsize=(6,6))
# 绘制每一折下 mse 的值
plt.semilogx(lasso.alphas_, lasso.mse_path_, linestyle=":")
ax.plot(
    lasso.alphas_,
    lasso.mse_path_.mean(axis=-1), # 每一个 alpha 下，交叉验证的 mse 的平均值
    color="black",
    label="Average across the folds",
    linewidth=2,
)
# 画竖直线
ax.axvline(
    lasso.alpha_,
    linestyle="--",
    color="black",
    label="alpha: CV estimate"
)
# 设置横纵轴范围
ax.set_ylim(ymin, ymax)
ax.set_xlabel(r"$\alpha$")
ax.set_ylabel("Mean square error")
# 添加图例
ax.legend()
ax.set_title(
    f"Mean square error on each fold: coordinate descent (train time: {fit_time:.
 ↪2f}s)"
)
plt.show()
fig1.savefig("../codeimage/code12.pdf")
```

```
        age        bp        s3        s6  random_02  random_05  random_08  \
```

```
0  0.038076  0.021872 -0.043401 -0.017646   0.647689  -0.234137  -0.469474
1 -0.001882 -0.026328  0.074412 -0.092204  -1.012831  -1.412304   0.067528
2  0.085299 -0.005670 -0.032356 -0.025930  -0.601707  -1.057711   0.208864
3 -0.089063 -0.036656 -0.036038 -0.009362  -1.478522   1.057122   0.324084
4  0.005383  0.021872  0.008142 -0.046641   0.331263  -0.185659   0.812526

   random_11
0  -0.465730
1   0.110923
2   0.196861
3   0.611676
4   1.003533
```

```
# 开始计时
start_time = time.time()
# 建立 LassolarCV 模型
model2 = make_pipeline(
    StandardScaler(),
    LassoLarsCV(cv=20)
)
# 模型拟合
model2.fit(X, y)
# 拟合时间
fit_time = time.time() - start_time
ymin, ymax = 2300, 3800
# 获取 lasso 模型
lasso = model2[-1]
fig2, ax = plt.subplots(figsize=(6,6))
# 绘制每一折下 mse 的值
plt.semilogx(lasso.cv_alphas_, lasso.mse_path_, linestyle=":")
ax.plot(
    lasso.cv_alphas_,
    lasso.mse_path_.mean(axis=-1), # 每一个 alpha 下，交叉验证的 mse 的平均值
    color="black",
    label="Average across the folds",
    linewidth=2,
)
# 画竖直线
ax.axvline(
    lasso.alpha_,
    linestyle="--",
    color="black",
    label="alpha: CV estimate"
)
```

```
# 设置横纵轴范围
ax.set_ylim(ymin, ymax)
ax.set_xlabel(r"$\alpha$")
ax.set_ylabel("Mean square error")
# 添加图例
ax.legend()
ax.set_title(
    f"Mean square error on each fold: coordinate descent (train time: {fit_time:.
 ↪2f}s)"
)
plt.show()
fig2.savefig("../codeimage/code13.pdf")
```

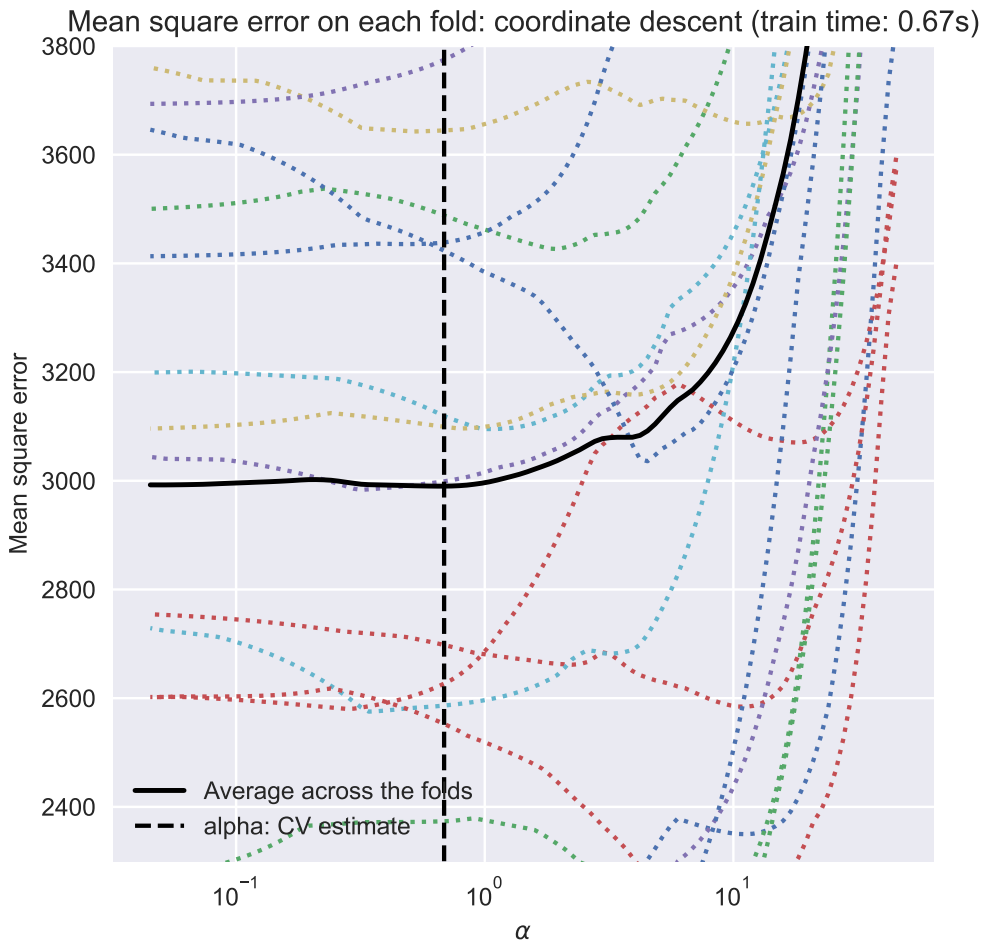

图 12: code12

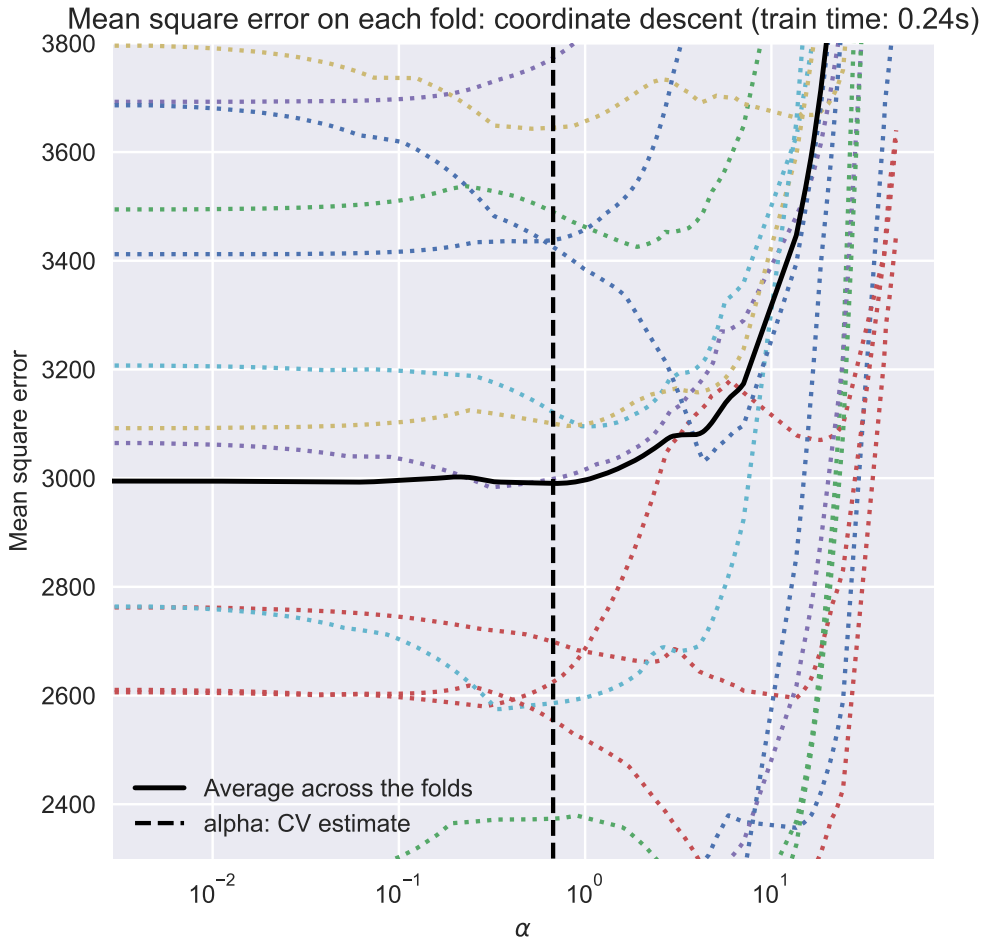
Mean square error on each fold: coordinate descent (train time: 0.24s)
Mean square error
3800
3600
3400
3200
3000
2800
2600
2400
Average across the folds
alpha: CV estimate
10^{-2}
10^{-1}
10^{0}
10^{1}
α

图 13: code13

Python 代码 10

```
# 导入操作系统库
import os
# 更改工作目录
os.chdir(r"D:\softwares\applied statistics\pythoncodelearning\chap1\sourcecode")
# 导入基础计算库
import numpy as np
# 导入绘图库
import matplotlib.pyplot as plt
# 导入数据集获取工具
from sklearn.datasets import load_diabetes
# 导入标准化处理工具
from sklearn.preprocessing import StandardScaler
# 导入 LassoCV 模型
from sklearn.linear_model import LassoLarsIC
# 导入管道操作
from sklearn.pipeline import make_pipeline
# 导入绘图库中的字体管理包
from matplotlib import font_manager
# 实现中文字符正常显示
font = font_manager.FontProperties(fname=r"C:\Windows\Fonts\SimKai.ttf")
# 使用 seaborn 风格绘图
plt.style.use("seaborn-v0_8")
# 导入数据集
X, y = load_diabetes(return_X_y=True, as_frame=True)
# 样本量
n_samples = X.shape[0]
# 构建模型
lasso_lars_ic = make_pipeline(
    StandardScaler(),
    LassoLarsIC(criterion="aic")
)
# 模型拟合
lasso_lars_ic.fit(X, y)
# 重新定义信息准则
def zou_et_al_criterion_rescaling(criterion, n_samples, noise_variance):
    """Rescale the information criterion to follow the definition of Zou et al."""
    return criterion - n_samples * np.log(2 * np.pi * noise_variance) - n_samples
# 缩放后的 AIC
aic_criterion = zou_et_al_criterion_rescaling(
    lasso_lars_ic[-1].criterion_,
    n_samples,
    lasso_lars_ic[-1].noise_variance_,
```

```
)
# 选出 AIC 最小对应的那个 alpha 所在的下标
index_alpha_path_aic = np.flatnonzero(
    lasso_lars_ic[-1].alphas_ == lasso_lars_ic[-1].alpha_
)[0]
# 不用重新拟合模型，直接修改参数即可，这就是 pipeline 的好处
lasso_lars_ic.set_params(
    lassolarsic__criterion="bic"
)
# 模型拟合
lasso_lars_ic.fit(X, y)
# 缩放后的 BIC
bic_criterion = zou_et_al_criterion_rescaling(
    lasso_lars_ic[-1].criterion_,
    n_samples,
    lasso_lars_ic[-1].noise_variance_,
)
# 选出 BIC 最小对应的那个 alpha 所在的下标
index_alpha_path_bic = np.flatnonzero(
    lasso_lars_ic[-1].alphas_ == lasso_lars_ic[-1].alpha_
)[0]
# 开始绘图
fig, ax = plt.subplots(figsize=(6,6))
ax.plot(
    aic_criterion,
    color="tab:blue",
    marker="o",
    label="AIC criterion"
)
ax.plot(
    bic_criterion,
    color="tab:orange",
    marker="o",
    label="BIC criterion"
)
ax.vlines(
    index_alpha_path_bic,
    bic_criterion.min(),
    bic_criterion.max(),
    color="black",
    linestyle="--",
    label="Selected alpha",
)
```

```
ax.vlines(
    index_alpha_path_aic,
    aic_criterion.min(),
    aic_criterion.max(),
    color="red",
    linestyle="--",
    label="Selected alpha",
)
# 显示图例
ax.legend()
ax.set_ylabel("Information criterion")
ax.set_xlabel("Lasso model sequence")
ax.set_title("Lasso model selection via AIC and BIC")
plt.show()
fig.savefig("../codeimage/code14.pdf")
```

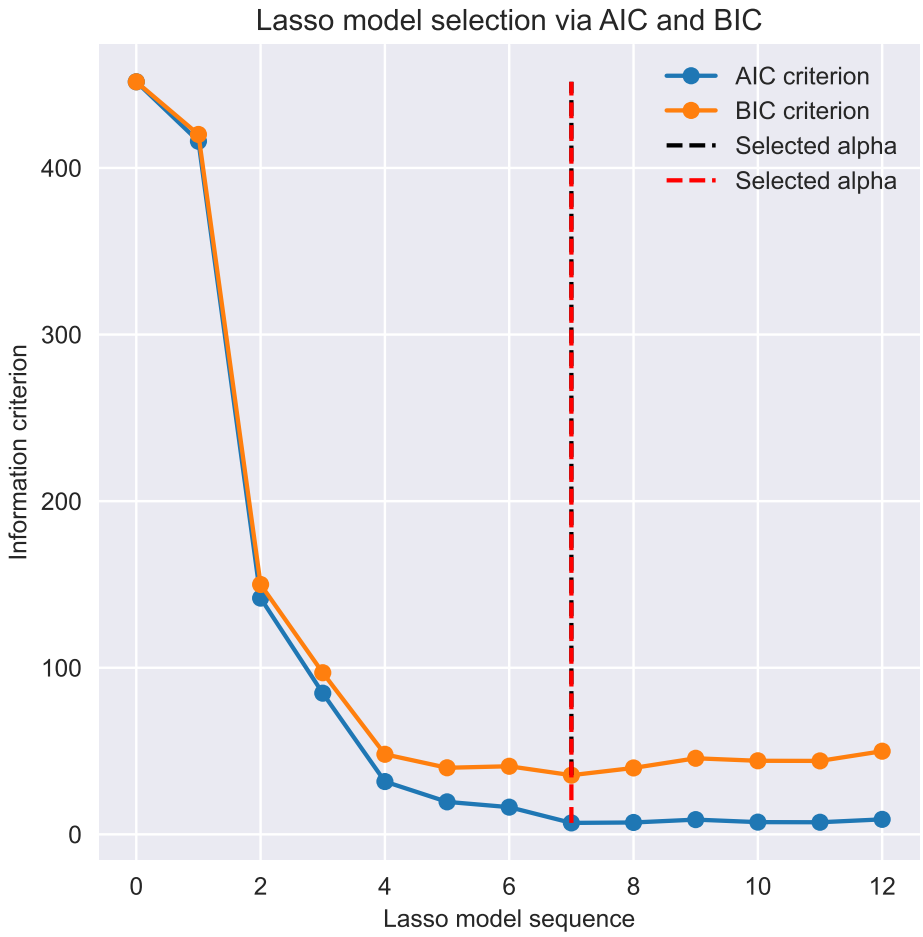

图 14: code14

Python 代码 11

```
# 导入操作系统库
import os
# 更改工作目录
os.chdir(r"D:\softwares\applied statistics\pythoncodelearning\chap1\sourcecode")
# 导入基础计算库
import numpy as np
# 导入绘图库
import matplotlib.pyplot as plt
# 导入 Lasso 模型
from sklearn.linear_model import MultiTaskLasso, Lasso
# 导入绘图库中的字体管理包
from matplotlib import font_manager
# 实现中文字符正常显示
font = font_manager.FontProperties(fname=r"C:\Windows\Fonts\SimKai.ttf")
# 使用 seaborn 风格绘图
plt.style.use("seaborn-v0_8")
# 设置样本量，维度，回归模型中 y 的维度（响应变量的多元回归）
n_samples, n_features, n_tasks = 100, 30, 40
# 显著变量的个数
n_relevant_features = 5
# 初始化真实系数，是一个矩阵
coef = np.zeros((n_tasks, n_features))
# 时刻
times = np.linspace(0, 2 * np.pi, n_tasks)
# 设置随机数种子
np.random.seed(10)
# 生成真实系数
for k in range(n_relevant_features):
    coef[:, k] = np.sin(
        (1.0 + np.random.randn(1)) * times + 3 * np.random.randn(1)
    )
# 生成 X
X = np.random.randn(n_samples, n_features)
# 生成 Y
Y = np.dot(X, coef.T) + np.random.randn(n_samples, n_tasks)
print("查看多元响应变量 Y 的情况：", Y[:5, :2], sep="\n")
# 建立 Lasso 模型，分别对 Y 的每一个分量做，提取系数
coef_lasso_ = np.array(
    [
        Lasso(alpha=0.5).fit(X, y).coef_ for y in Y.T
    ]
)
```

```
# 建立 MultiLasso 模型，提取系数
coef_multi_task_lasso_ = MultiTaskLasso(alpha=1.0).fit(X, Y).coef_
# 开始绘图
fig, axs = plt.subplots(nrows=1, ncols=2, figsize=(8, 5))
# 用于展示稀疏二维数组的图形
axs[0].spy(coef_lasso_)
axs[0].set_xlabel("Feature")
axs[0].set_ylabel("Time (or Task)")
axs[0].text(10, 5, "Lasso")
axs[1].spy(coef_multi_task_lasso_)
axs[1].set_xlabel("Feature")
axs[1].set_ylabel("Time (or Task)")
axs[1].text(10, 5, "MultiTaskLasso")
fig.suptitle("Coefficient non-zero location")
plt.show()
fig.savefig("../codeimage/code15.pdf")
# 绘制第一个特征前的系数
feature_to_plot = 0
# 开始绘图
fig1, ax = plt.subplots(figsize=(6,6), tight_layout=True)
# 绘制 coef 的线图
ax.plot(
    coef[:, feature_to_plot],
    color="seagreen",
    linewidth=2,
    label="Ground truth"
)
# 绘制 coef_lasso 的线图
ax.plot(
    coef_lasso_[:, feature_to_plot],
    color="cornflowerblue",
    linewidth=2,
    label="Lasso"
)
# 绘制 coef_task_lasso 的线图
ax.plot(
    coef_multi_task_lasso_[:, feature_to_plot],
    color="gold",
    linewidth=2,
    label="MultiTaskLasso",
)
# 显示图例
ax.legend(loc="best")
```

```
# 设置纵轴范围
ax.set_ylim([-1.1, 1.1])
plt.show()
fig1.savefig("../codeimage/code16.pdf")

查看多元响应变量 Y 的情况:
[[1.89931525 1.64556889]
 [2.08468805 2.38111156]
 [1.41645186 0.98673872]
 [0.18540611 1.80601826]
 [0.69800933 0.40984124]]
```

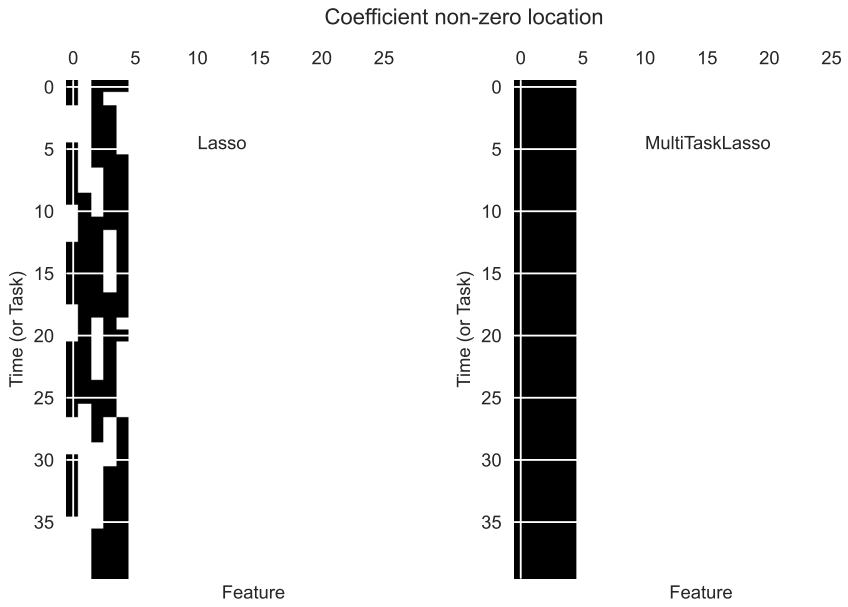

图 15: code15

Python 代码 12

```
# 导入操作系统库
import os
# 更改工作目录
os.chdir(r"D:\softwares\applied statistics\pythoncodelearning\chap1\sourcecode")
# 导入基础计算库
import numpy as np
# 导入绘图库
import matplotlib.pyplot as plt
# 导入 Lasso 模型
from sklearn.linear_model import ElasticNet, Lasso
# 导入模型评价工具
```

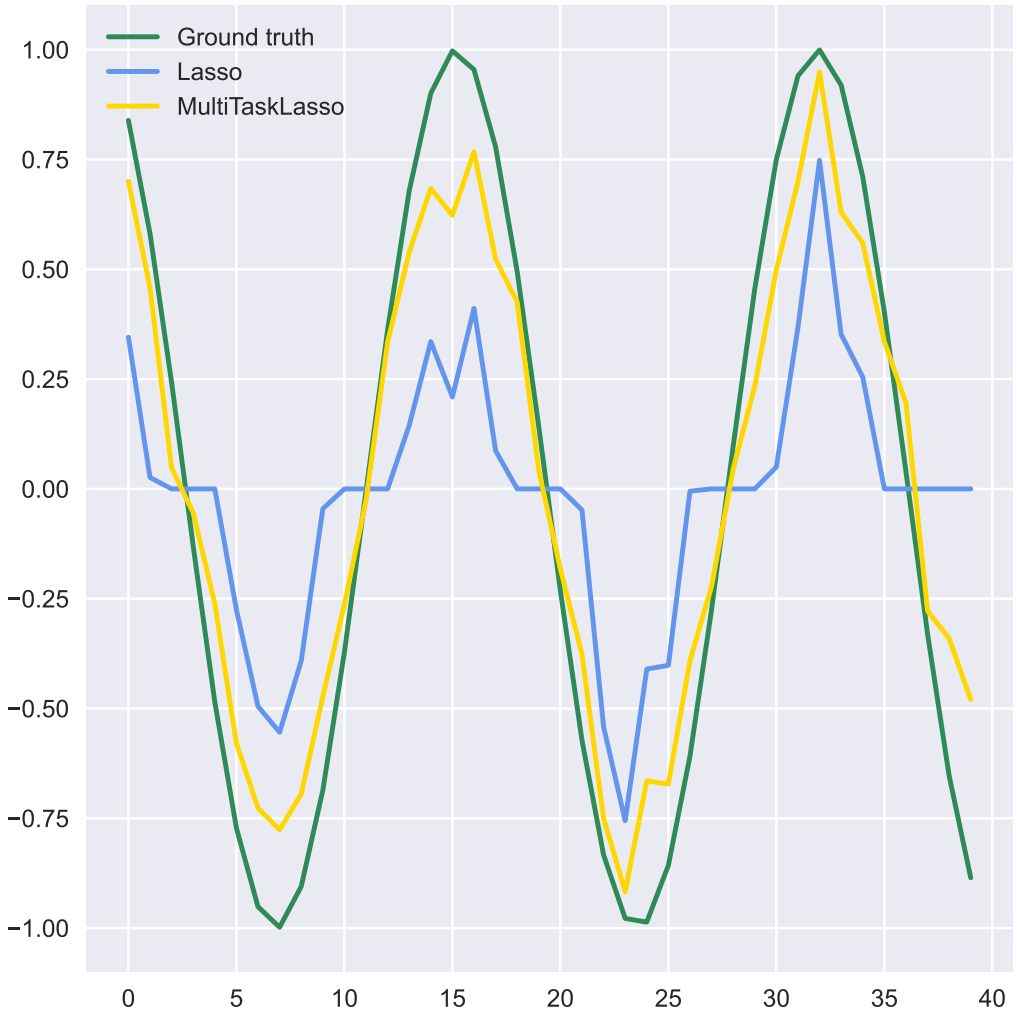
Ground truth
Lasso
MultiTaskLasso
1.00
0.75
0.50
0.25
0.00
−0.25
−0.50
−0.75
−1.00
0
5
10
15
20
25
30
35
40

图 16: code16

```
from sklearn.metrics import r2_score
# 导入绘图库中的字体管理包
from matplotlib import font_manager
# 实现中文字符正常显示
font = font_manager.FontProperties(fname=r"C:\Windows\Fonts\SimKai.ttf")
# 使用 seaborn 风格绘图
plt.style.use("seaborn-v0_8")
# 设置样本量和维度
n_samples, n_features = 50, 100
# 生成 X
X = np.random.randn(n_samples, n_features)
# 生成真实系数
idx = np.arange(n_features)
coef = (-1) ** idx * np.exp(-idx / 10)
# 部分系数为零，稀疏情况
coef[10:] = 0  # sparsify coef
# 生成 y
y = np.dot(X, coef) + 0.01 * np.random.normal(size=n_samples)
# 划分数据集
n_samples = X.shape[0]
X_train, y_train = X[: n_samples // 2], y[: n_samples // 2]
X_test, y_test = X[n_samples // 2 :], y[n_samples // 2 :]
# 设置 Lasso 的惩罚系数
alpha = 0.1
# 建立 lasso 模型
lasso = Lasso(alpha=alpha)
# 模型拟合
lasso.fit(X_train, y_train)
# 预测
y_pred_lasso = lasso.predict(X_test)
# R 方
r2_score_lasso = r2_score(y_test, y_pred_lasso)
print("r^2 on test data of lasso : %f" % r2_score_lasso)
# 建立 elasticnet 模型
enet = ElasticNet(alpha=alpha, l1_ratio=0.7)
# 模型拟合
enet.fit(X_train, y_train)
# 预测
y_pred_enet = enet.predict(X_test)
# R 方
r2_score_enet = r2_score(y_test, y_pred_enet)
print("r^2 on test data of elastic net: %f" % r2_score_enet)
# 开始绘图
```

```
fig, ax = plt.subplots(figsize=(6,6))
m, s, _ = ax.stem(
    np.where(enet.coef_)[0],
    enet.coef_[enet.coef_ != 0],
    markerfmt="x",
    label="Elastic net coefficients",
)
plt.setp([m, s], color="#2ca02c")
m, s, _ = ax.stem(
    np.where(lasso.coef_)[0],
    lasso.coef_[lasso.coef_ != 0],
    markerfmt="x",
    label="Lasso coefficients",
)
plt.setp([m, s], color="#ff7f0e")
ax.stem(
    np.where(coef)[0],
    coef[coef != 0],
    label="true coefficients",
    markerfmt="bx",
)
# 显示图例
ax.legend(loc="best")
ax.set_title(
    "Lasso $R^2$: %.3f, Elastic Net $R^2$: %.3f" % (r2_score_lasso, r2_score_enet)
)
plt.show()
fig.savefig("../codeimage/code17.pdf")
```

```
r^2 on test data of lasso : 0.626469
r^2 on test data of elastic net: 0.589165
```

Python 代码 13

```
# 导入操作系统库
import os
# 更改工作目录
os.chdir(r"D:\softwares\applied statistics\pythoncodelearning\chap1\sourcecode")
# 导入循环工具
from itertools import cycle
# 导入基础计算库
import numpy as np
# 导入绘图库
import matplotlib.pyplot as plt
```

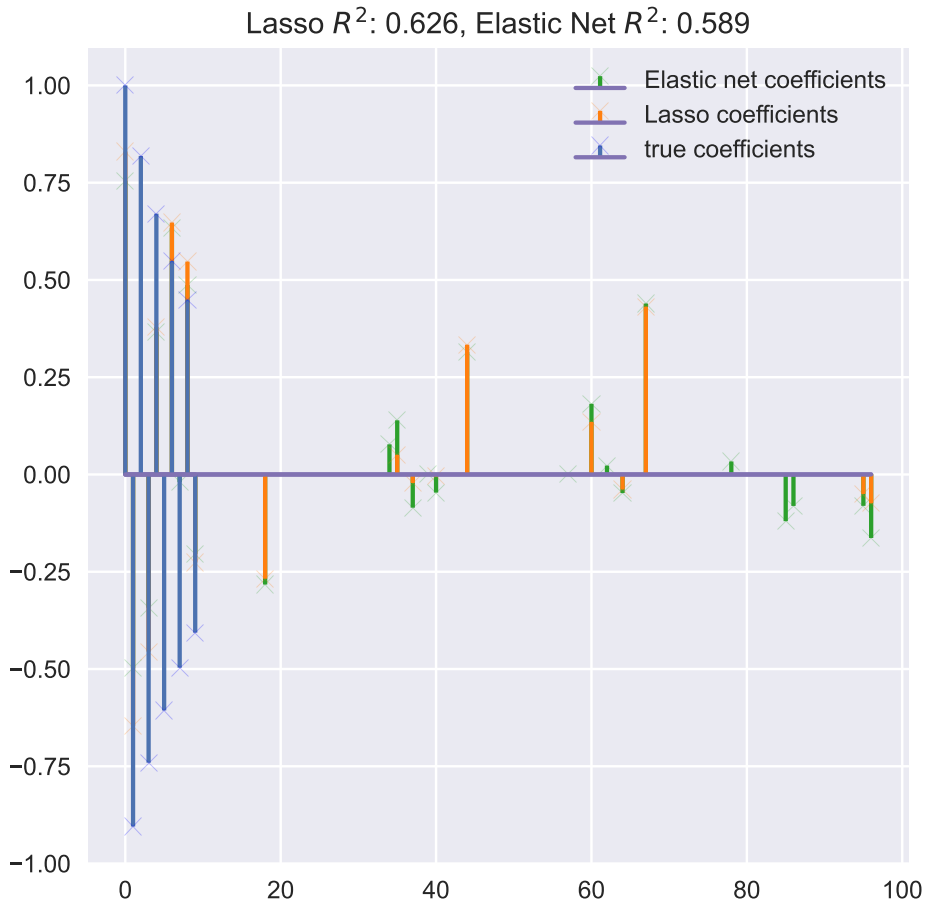
Lasso R^2: 0.626, Elastic Net R^2: 0.589
Elastic net coefficients
Lasso coefficients
true coefficients
1.00
0.75
0.50
0.25
0.00
−0.25
−0.50
−0.75
−1.00
0
20
40
60
80
100

图 17: code17

```
# 导入 lasso 系数路径
from sklearn.linear_model import lasso_path, enet_path
# 导入糖尿病数据集
from sklearn.datasets import load_diabetes
# 导入标准化工具
from sklearn.preprocessing import StandardScaler
# 导入绘图库中的字体管理包
from matplotlib import font_manager
# 实现中文字符正常显示
font = font_manager.FontProperties(fname=r"C:\Windows\Fonts\SimKai.ttf")
# 使用 seaborn 风格绘图
plt.style.use("seaborn-v0_8")
X, y = load_diabetes(return_X_y=True, as_frame=True)
# 对 X 进行标准化
X = StandardScaler().fit(X).transform(X)
# 用于自动生成 alpha
eps = 5e-3
print("Computing regularization path using the lasso...")
# lasso 模型的路径
alphas_lasso, coefs_lasso, _ = lasso_path(X, y, eps=eps)
print("Computing regularization path using the positive lasso...")
# positive lasso 模型的路径
alphas_positive_lasso, coefs_positive_lasso, _ = lasso_path(
    X, y, eps=eps, positive=True
)
print("Computing regularization path using the elastic net...")
# elastic net 模型的路径
alphas_enet, coefs_enet, _ = enet_path(X, y, eps=eps, l1_ratio=0.8)
print("Computing regularization path using the positive elastic net...")
# positive elastic net 模型的路径
alphas_positive_enet, coefs_positive_enet, _ = enet_path(
    X, y, eps=eps, l1_ratio=0.8, positive=True
)
# 开始绘图
fig1, ax = plt.subplots(figsize=(6,6), tight_layout=True)
colors = cycle(["b", "r", "g", "c", "k"])
# 非负对数化
neg_log_alphas_lasso = -np.log10(alphas_lasso)
# 非负对数化
neg_log_alphas_enet = -np.log10(alphas_enet)
for coef_l, coef_e, c in zip(coefs_lasso, coefs_enet, colors):
    l1 = ax.plot(neg_log_alphas_lasso, coef_l, c=c)
    l2 = ax.plot(neg_log_alphas_enet, coef_e, linestyle="--", c=c)
```

```
ax.set_xlabel("-Log(alpha)")
ax.set_ylabel("coefficients")
ax.set_title("Lasso and Elastic-Net Paths")
# 设置图例
ax.legend((l1[-1], l2[-1]), ("Lasso", "Elastic-Net"), loc="lower left")
plt.show()
fig1.savefig("../codeimage/code18.pdf")
# 开始绘图
fig2, ax = plt.subplots(figsize=(6,6), tight_layout=True)
neg_log_alphas_positive_lasso = -np.log10(alphas_positive_lasso)
for coef_l, coef_pl, c in zip(coefs_lasso, coefs_positive_lasso, colors):
    l1 = ax.plot(neg_log_alphas_lasso, coef_l, c=c)
    l2 = ax.plot(neg_log_alphas_positive_lasso, coef_pl, linestyle="--", c=c)

ax.set_xlabel("-Log(alpha)")
ax.set_ylabel("coefficients")
ax.set_title("Lasso and positive Lasso")
ax.legend((l1[-1], l2[-1]), ("Lasso", "positive Lasso"), loc="lower left")
fig2.savefig("../codeimage/code19.pdf")
# 开始绘图
fig3, ax = plt.subplots(figsize=(6,6), tight_layout=True)
neg_log_alphas_positive_enet = -np.log10(alphas_positive_enet)
for coef_e, coef_pe, c in zip(coefs_enet, coefs_positive_enet, colors):
    l1 = ax.plot(neg_log_alphas_enet, coef_e, c=c)
    l2 = ax.plot(neg_log_alphas_positive_enet, coef_pe, linestyle="--", c=c)

ax.set_xlabel("-Log(alpha)")
ax.set_ylabel("coefficients")
ax.set_title("Elastic-Net and positive Elastic-Net")
ax.legend((l1[-1], l2[-1]), ("Elastic-Net", "positive Elastic-Net"), loc="lower␣
 ↪left")
plt.show()
fig3.savefig("../codeimage/code20.pdf")

Computing regularization path using the lasso…
Computing regularization path using the positive lasso…
Computing regularization path using the elastic net…
Computing regularization path using the positive elastic net…
```

Python 代码 14

```
# 导入操作系统库
import os
```

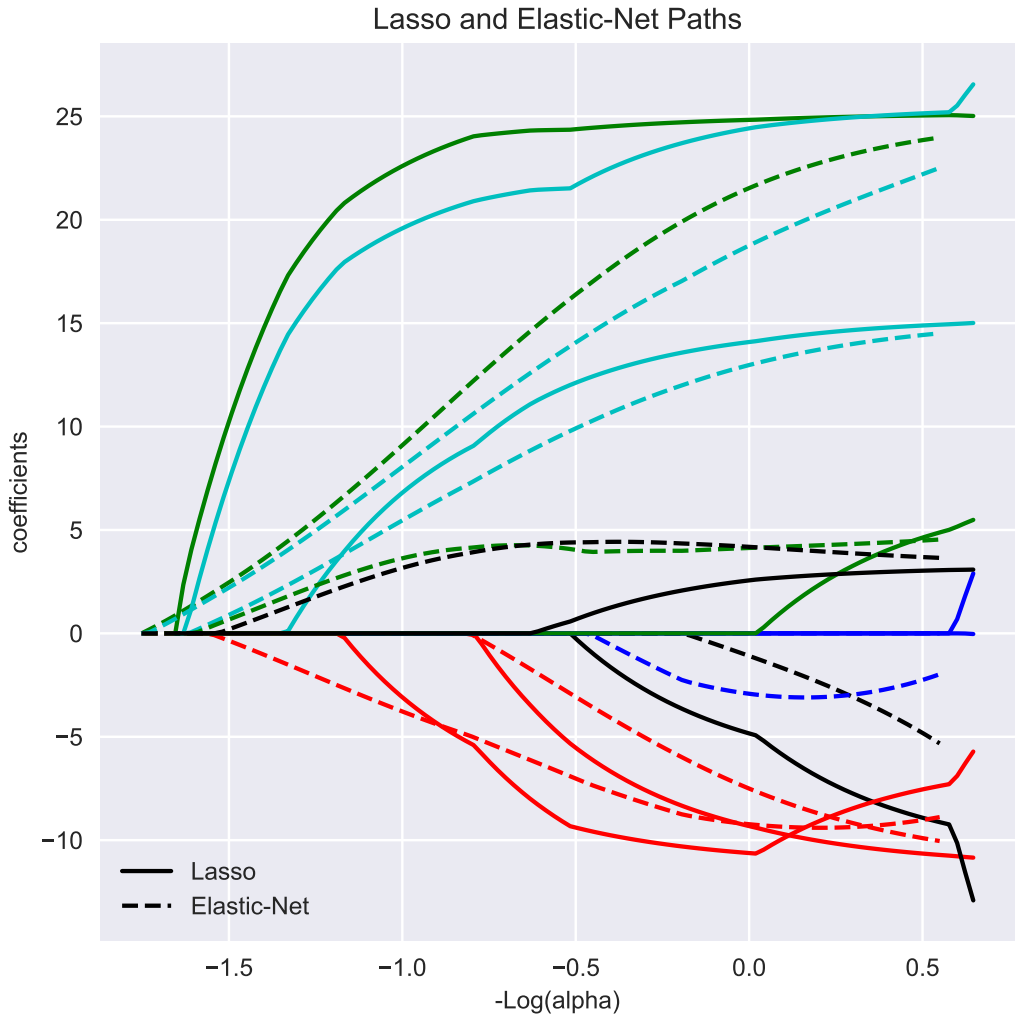
Lasso and Elastic-Net Paths
coefficients
25
20
15
10
5
0
−5
−10
Lasso
Elastic-Net
−1.5
−1.0
−0.5
0.0
0.5
-Log(alpha)

图 18: code18

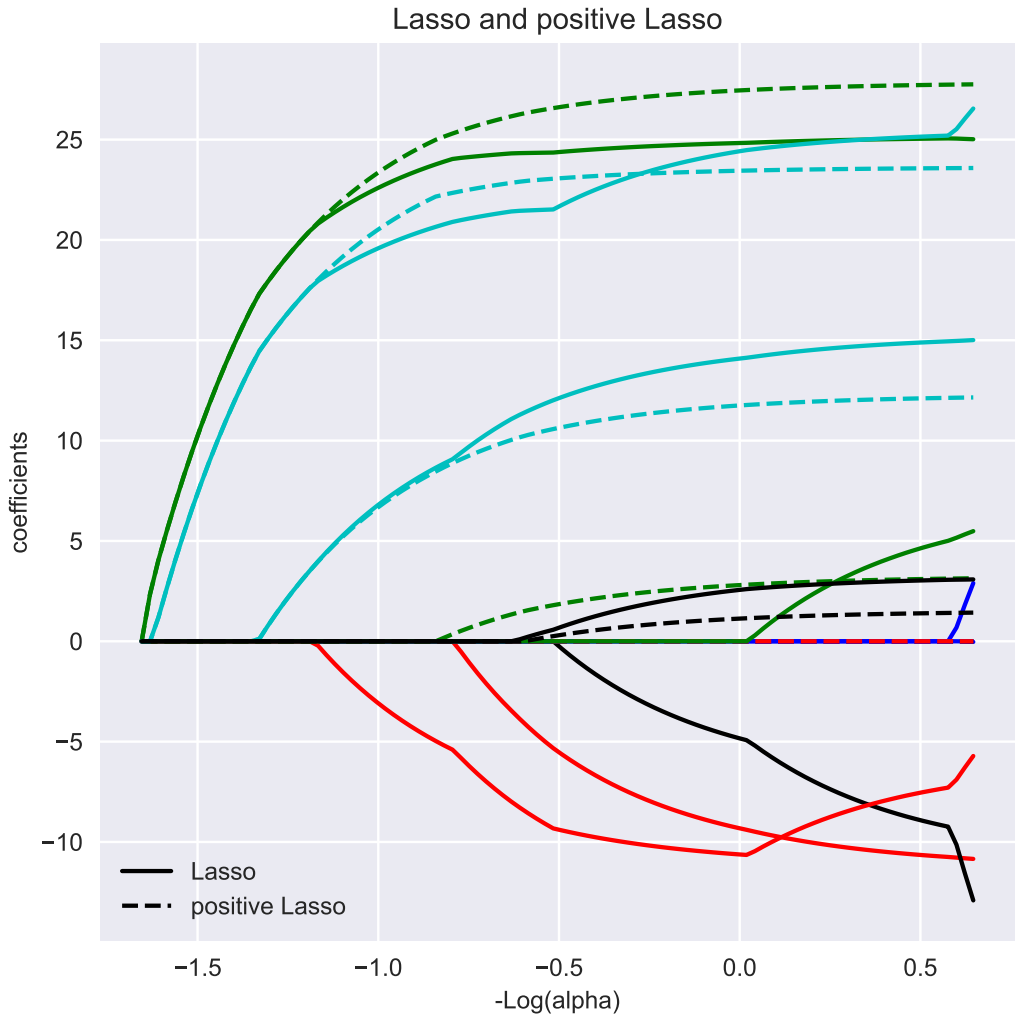
Lasso and positive Lasso
coefficients
25
20
15
10
5
0
−5
−10
−1.5
−1.0
−0.5
0.0
0.5
-Log(alpha)
Lasso
positive Lasso

图 19: code19

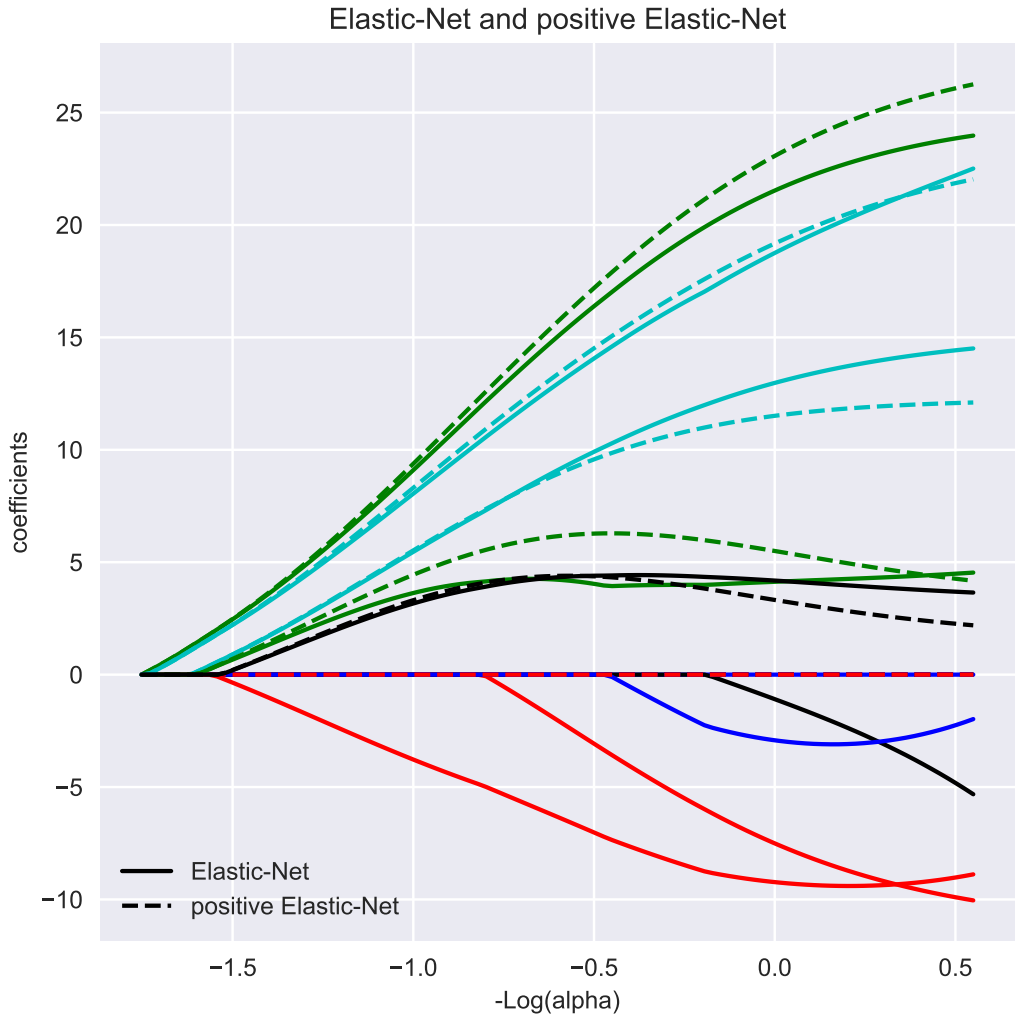
Elastic-Net and positive Elastic-Net
coefficients
25
20
15
10
5
0
−5
−10
−1.5
−1.0
−0.5
0.0
0.5
-Log(alpha)
Elastic-Net
positive Elastic-Net

图 20: code20

```
# 更改工作目录
os.chdir(r"D:\softwares\applied statistics\pythoncodelearning\chap1\sourcecode")
# 导入循环工具
from itertools import cycle
# 导入基础计算库
import numpy as np
# 导入绘图库
import matplotlib.pyplot as plt
# 导入 lasso 系数路径
from sklearn.linear_model import lars_path
# 导入糖尿病数据集
from sklearn.datasets import load_diabetes
# 导入标准化工具
from sklearn.preprocessing import StandardScaler
# 导入绘图库中的字体管理包
from matplotlib import font_manager
# 实现中文字符正常显示
font = font_manager.FontProperties(fname=r"C:\Windows\Fonts\SimKai.ttf")
# 使用 seaborn 风格绘图
plt.style.use("seaborn-v0_8")
X, y = load_diabetes(return_X_y=True, as_frame=True)
# 对 X 进行标准化
X = StandardScaler().fit(X).transform(X)
print("Computing regularization path using the LARS ...")
_, _, coefs = lars_path(X, y, method="lasso", verbose=True)
# 对系数做一个 scale
xx = np.sum(np.abs(coefs.T), axis=1)
xx /= xx[-1]
# 开始绘图
fig, ax = plt.subplots(figsize=(6,6), tight_layout=True)
# 绘制系数路径图
ax.plot(xx, coefs.T)
# 获取 Y 轴的最大，小值
ymin, ymax = ax.get_ylim()
# 绘制竖直线
ax.vlines(xx, ymin, ymax, linestyle="dashed")
ax.set_xlabel("|coef| / max|coef|")
ax.set_ylabel("Coefficients")
ax.set_title("LASSO Path")
plt.show()
fig.savefig("../codeimage/code21.pdf")

Computing regularization path using the LARS …
.
```

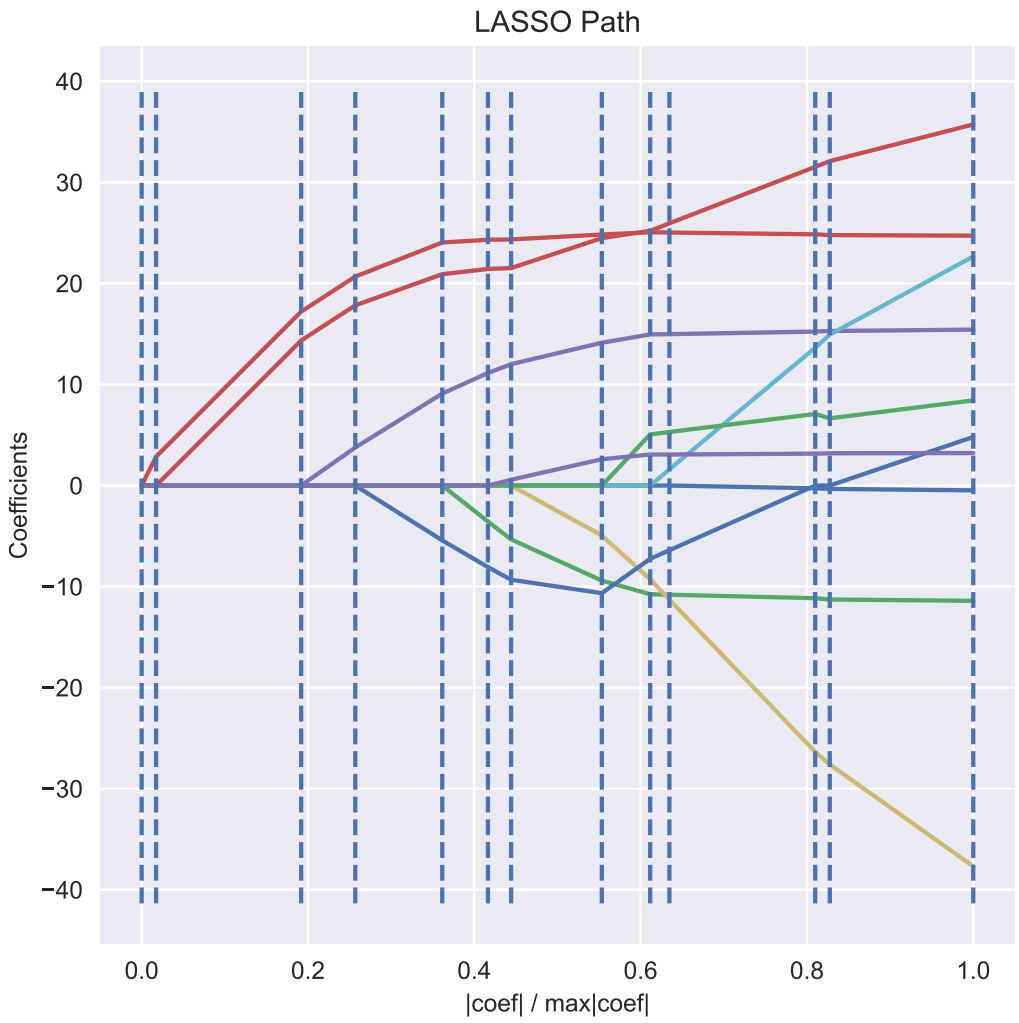
LASSO Path
Coefficients
40
30
20
10
0
−10
−20
−30
−40
0.0
0.2
0.4
0.6
0.8
1.0
|coef| / max|coef|

图 21: code21

Python 代码 15

```
# 导入操作系统库
import os
# 更改工作目录
os.chdir(r"D:\softwares\applied statistics\pythoncodelearning\chap1\sourcecode")
# 导入基础计算库
import numpy as np
# 导入绘图库
import matplotlib.pyplot as plt
# 导入 OMP 模型
from sklearn.linear_model import OrthogonalMatchingPursuit
from sklearn.linear_model import OrthogonalMatchingPursuitCV
# 导入数据集生成工具
from sklearn.datasets import make_sparse_coded_signal
# 导入绘图库中的字体管理包
from matplotlib import font_manager
# 实现中文字符正常显示
font = font_manager.FontProperties(fname=r"C:\Windows\Fonts\SimKai.ttf")
# 使用 seaborn 风格绘图
plt.style.use("seaborn-v0_8")
# 设置样本量和维度，（这里有点倒置了）
n_components, n_features = 512, 100
# 设置非零系数的个数
n_nonzero_coefs = 17
# 生成稀疏数据集
y, X, w = make_sparse_coded_signal(
    n_samples=1,
    n_components=n_components,
    n_features=n_features,
    n_nonzero_coefs=n_nonzero_coefs,
    random_state=0,
    data_transposed=True,
)
# 非零值的下标
(idx,) = w.nonzero()
# 添加一些噪声
y_noisy = y + 0.05 * np.random.randn(len(y))
# 建立 OMP 模型
omp = OrthogonalMatchingPursuit(n_nonzero_coefs=n_nonzero_coefs)
# 模型拟合，无噪声
omp.fit(X, y)
# 提取系数
coef1 = omp.coef_
```

```
# 提起系数的非零下标
(idx_r1,) = coef1.nonzero()
# 模型拟合有噪声
omp.fit(X, y_noisy)
coef2 = omp.coef_
(idx_r2,) = coef2.nonzero()
# 建立 CVOMP 模型
omp_cv = OrthogonalMatchingPursuitCV()
omp_cv.fit(X, y_noisy)
coef3 = omp_cv.coef_
(idx_r3,) = coef3.nonzero()
# 绘制真实系数图
fig, axs = plt.subplots(nrows=4, ncols=1, figsize=(6,10))
axs[0].set_xlim(0, 512)
axs[0].set_title("Sparse signal")
axs[0].stem(idx, w[idx])
axs[1].set_xlim(0, 512)
axs[1].set_title("Recovered signal from noise-free measurements")
axs[1].stem(idx_r1, coef1[idx_r1])
axs[2].set_xlim(0, 512)
axs[2].set_title("Recovered signal from noisy measurements")
axs[2].stem(idx_r2, coef2[idx_r2])
axs[3].set_xlim(0, 512)
axs[3].set_title("Recovered signal from noisy measurements with CV")
axs[3].stem(idx_r3, coef3[idx_r3])
plt.subplots_adjust(0.06, 0.04, 0.94, 0.90, 0.20, 0.38)
plt.suptitle("Sparse signal recovery with Orthogonal Matching Pursuit",␣
 ↪fontsize=16)
plt.show()
fig.savefig("../codeimage/code22.pdf")
```

Python 代码 16

```
# 导入操作系统库
import os
# 更改工作目录
os.chdir(r"D:\softwares\applied statistics\pythoncodelearning\chap1\sourcecode")
# 导入基础计算库
import numpy as np
# 导入绘图库
import matplotlib.pyplot as plt
# 导入贝叶斯岭回归模型
from sklearn.linear_model import BayesianRidge
```

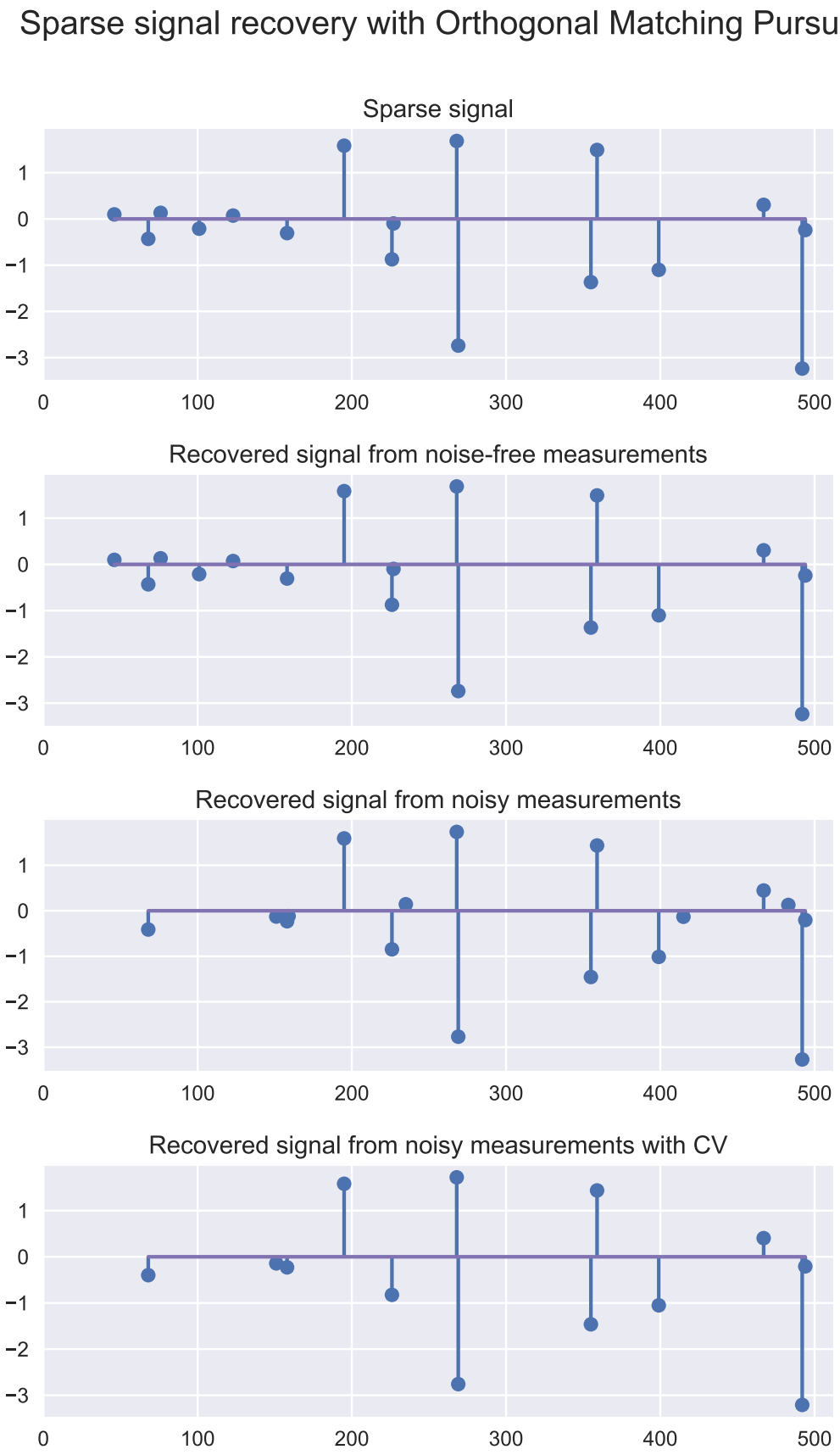
Sparse signal recovery with Orthogonal Matching Pursuit
Sparse signal
Recovered signal from noise-free measurements
Recovered signal from noisy measurements
Recovered signal from noisy measurements with CV
1
0
−1
−2
−3
0
100
200
300
400
500

图 22: code22

```
# 导入绘图库中的字体管理包
from matplotlib import font_manager
# 实现中文字符正常显示
font = font_manager.FontProperties(fname=r"C:\Windows\Fonts\SimKai.ttf")
# 使用 seaborn 风格绘图
plt.style.use("seaborn-v0_8")
# 生成 y
def func(x):
    return np.sin(2 * np.pi * x)
# 样本量
size = 25
# 设置随机数种子
np.random.seed(1234)
# 训练集
x_train = np.random.uniform(0.0, 1.0, size)
y_train = func(x_train) + np.random.normal(scale=0.1, size=size)
# 测试集
x_test = np.linspace(0.0, 1.0, 100)
# 范德蒙行列式的阶数
n_order = 3
# 生成范德蒙矩阵的训练集
X_train = np.vander(x_train, n_order + 1, increasing=True)
X_test = np.vander(x_test, n_order + 1, increasing=True)
# 构建贝叶斯岭回归模型
reg = BayesianRidge(tol=1e-6, fit_intercept=False, compute_score=True)
# 开始绘图
fig, axes = plt.subplots(1, 2, figsize=(8, 4), tight_layout=True)
for i, ax in enumerate(axes):
    # Bayesian ridge regression with different initial value pairs
    if i == 0:
        init = [1 / np.var(y_train), 1.0]  # Default values
    elif i == 1:
        init = [1.0, 1e-3]
        # 设置参数的初始值
        reg.set_params(alpha_init=init[0], lambda_init=init[1])
    # 模型拟合
    reg.fit(X_train, y_train)
    # 测试集上预测
    ymean, ystd = reg.predict(X_test, return_std=True)
    # 真实散点测试集，无噪声
    ax.plot(x_test, func(x_test), color="blue", label="sin($2\\pi x$)")
    # 真实散点训练集，有噪声
    ax.scatter(x_train, y_train, s=50, alpha=0.5, label="observation")
```

```
    # 测试集预测值
    ax.plot(x_test, ymean, color="red", label="predict mean")
    # 置信带
    ax.fill_between(
        x_test, ymean - ystd, ymean + ystd, color="pink", alpha=0.5,␣
↪label="predict std"
    )
    ax.set_ylim(-1.3, 1.3)
    # 设置图例
    ax.legend()
    title = "$\\alpha$_init$={:.2f},\\ \\lambda$_init$={}$".format(init[0],␣
↪init[1])
    if i == 0:
        title += " (Default)"
    ax.set_title(title, fontsize=12)
    text = "$\\alpha={:.1f}$\n$\\lambda={:.3f}$\n$L={:.1f}$".format(
        reg.alpha_, reg.lambda_, reg.scores_[-1]
    )
    ax.text(0.05, -1.0, text, fontsize=12)

plt.show()
fig.savefig("../codeimage/code23.pdf")
```

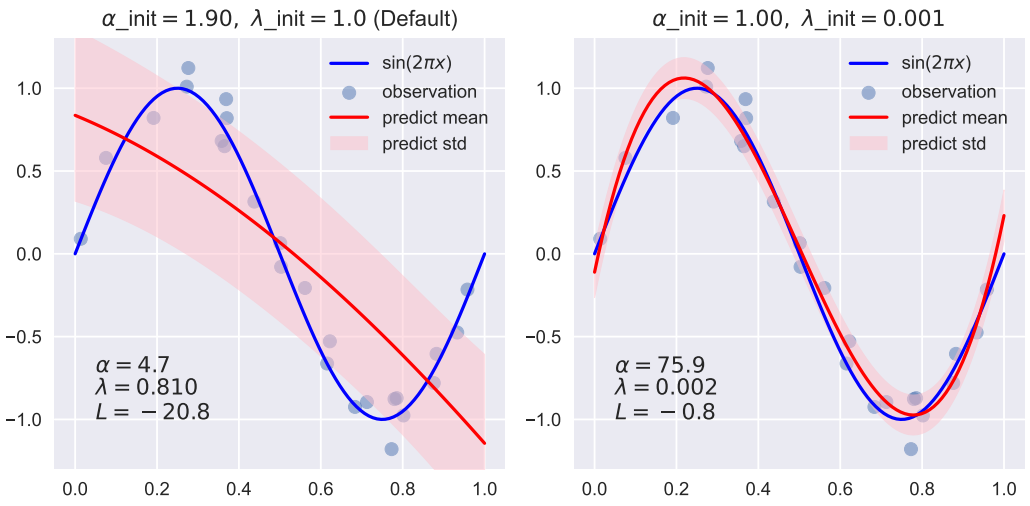

图 23: code23

Python 代码 17

```
# 导入操作系统库
import os
# 更改工作目录
```

```
os.chdir(r"D:\softwares\applied statistics\pythoncodelearning\chap1\sourcecode")
# 导入基础计算库
import numpy as np
# 导入绘图库
import matplotlib.pyplot as plt
# 导入 logistic 回归模型
from sklearn.linear_model import LogisticRegression
# 导入数据集
from sklearn.datasets import load_digits
# 导入标准化工具
from sklearn.preprocessing import StandardScaler
# 导入绘图库中的字体管理包
from matplotlib import font_manager
# 实现中文字符正常显示
font = font_manager.FontProperties(fname=r"C:\Windows\Fonts\SimKai.ttf")
# 使用 seaborn 风格绘图
plt.style.use("seaborn-v0_8")
# 生成数据及
X, y = load_digits(return_X_y=True)
# 标准化
X = StandardScaler().fit_transform(X)
# 取大于 4 的类别作为标签
y = (y > 4).astype(int)
# 弹性网的比例
l1_ratio = 0.5
# 开始绘图
fig, axes = plt.subplots(3, 3, figsize=(9,9))
for i, (C, axes_row) in enumerate(zip((1, 0.1, 0.01), axes)):
    # Increase tolerance for short training time
    clf_l1_LR = LogisticRegression(
        C=C,
        penalty="l1",
        tol=0.01,
        solver="saga"
    )
    clf_l2_LR = LogisticRegression(
        C=C,
        penalty="l2",
        tol=0.01, solver="saga"
    )
    clf_en_LR = LogisticRegression(
        C=C,
        penalty="elasticnet",
```

```
        solver="saga",
        l1_ratio=l1_ratio,
        tol=0.01
    )
    # 模型拟合
    clf_l1_LR.fit(X, y)
    clf_l2_LR.fit(X, y)
    clf_en_LR.fit(X, y)
    # 提取模型的系数
    coef_l1_LR = clf_l1_LR.coef_.ravel()
    coef_l2_LR = clf_l2_LR.coef_.ravel()
    coef_en_LR = clf_en_LR.coef_.ravel()
    # 系数的稀疏程度，以含有零的比例
    sparsity_l1_LR = np.mean(coef_l1_LR == 0) * 100
    sparsity_l2_LR = np.mean(coef_l2_LR == 0) * 100
    sparsity_en_LR = np.mean(coef_en_LR == 0) * 100
    print("C=%.2f" % C)
    print(
        "{:<40} {:.2f}%".format(
            "Sparsity with L1 penalty:",
            sparsity_l1_LR
        )
    )
    print(
        "{:<40} {:.2f}%".format(
            "Sparsity with Elastic-Net penalty:",
            sparsity_en_LR
        )
    )
    print(
        "{:<40} {:.2f}%".format(
            "Sparsity with L2 penalty:",
            sparsity_l2_LR
        )
    )
    print(
        "{:<40} {:.2f}".format(
            "Score with L1 penalty:",
            clf_l1_LR.score(X, y) # 分类准确率
        )
    )
    print(
        "{:<40} {:.2f}".format(
```

```
            "Score with Elastic-Net penalty:",
            clf_en_LR.score(X, y)
        )
    )
    print(
        "{:<40} {:.2f}".format(
            "Score with L2 penalty:",
            clf_l2_LR.score(X, y)
        )
    )

    if i == 0:
        axes_row[0].set_title("L1 penalty")
        axes_row[1].set_title("Elastic-Net\nl1_ratio = %s" % l1_ratio)
        axes_row[2].set_title("L2 penalty")

    for ax, coefs in zip(axes_row, [coef_l1_LR, coef_en_LR, coef_l2_LR]):
        ax.imshow(
            np.abs(coefs.reshape(8, 8)),
            interpolation="nearest",
            cmap="binary",
            vmax=1,
            vmin=0,
        )
        ax.set_xticks(())
        ax.set_yticks(())

    axes_row[0].set_ylabel("C = %s" % C)

plt.show()
fig.savefig("../codeimage/code24.pdf")
```

```
C=1.00
Sparsity with L1 penalty:                4.69%
Sparsity with Elastic-Net penalty:       4.69%
Sparsity with L2 penalty:                4.69%
Score with L1 penalty:                   0.90
Score with Elastic-Net penalty:          0.90
Score with L2 penalty:                   0.90
C=0.10
Sparsity with L1 penalty:                25.00%
Sparsity with Elastic-Net penalty:       14.06%
Sparsity with L2 penalty:                4.69%
Score with L1 penalty:                   0.90
```

```
Score with Elastic-Net penalty:          0.90
Score with L2 penalty:                   0.90
C=0.01
Sparsity with L1 penalty:                84.38%
Sparsity with Elastic-Net penalty:       68.75%
Sparsity with L2 penalty:                4.69%
Score with L1 penalty:                   0.86
Score with Elastic-Net penalty:          0.88
Score with L2 penalty:                   0.89
```

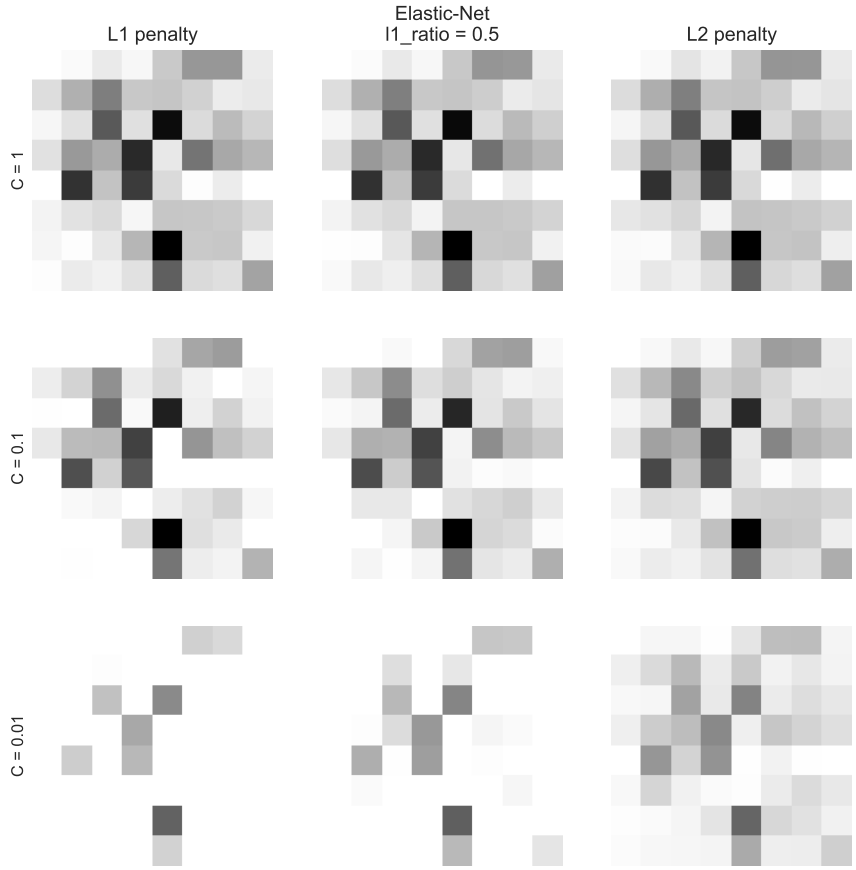

图 24: code24

Python 代码 18

```
# 导入操作系统库
import os
# 更改工作目录
```

```
os.chdir(r"D:\softwares\applied statistics\pythoncodelearning\chap1\sourcecode")
# 导入基础计算库
import numpy as np
# 导入绘图库
import matplotlib.pyplot as plt
# 导入 logistic 回归模型
from sklearn.linear_model import LogisticRegression
# 导入数据集
from sklearn.datasets import load_iris
# 导入标准化工具
from sklearn.preprocessing import StandardScaler
# 导入绘图库中的字体管理包
from matplotlib import font_manager
# 实现中文字符正常显示
font = font_manager.FontProperties(fname=r"C:\Windows\Fonts\SimKai.ttf")
# 使用 seaborn 风格绘图
plt.style.use("seaborn-v0_8")
# 生成数据集
X, y = load_iris(return_X_y=True, as_frame=True)
# 选择 y 的两个标签对应的数据
X = X[y != 2]
y = y[y != 2]
# 对 X 标准化
X = StandardScaler().fit_transform(X)
# 建立带惩罚的 Logistic 模型
clf1 = LogisticRegression(
    penalty="l1",
    solver="liblinear"
)
# 建立不带惩罚的 Logistic 模型
clf2 = LogisticRegression(
    penalty=None
)
# 模型拟合
clf1.fit(X, y)
clf2.fit(X, y)
# 输出系数
print(clf1.coef_)
print(clf2.coef_)
# 标签预测
y1_pred = clf1.predict(X)
y2_pred = clf2.predict(X)
# 概率预测
```

```
prob1_pred = clf1.predict_proba(X)
prob2_pred = clf2.predict_proba(X)
# 输出标签
print(y1_pred)
print(y2_pred)
# 输出概率
print(prob1_pred[:10,])
print(prob2_pred[:10,])
```

```
[[ 0.         -0.7448795   2.4336159   1.94042766]]
[[ 2.10062976 -3.87615098  7.68741035  8.08451415]]
[0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
 0 0 0 0 0 0 0 0 0 0 0 0 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1
 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1]
[0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
 0 0 0 0 0 0 0 0 0 0 0 0 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1
 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1]
[[0.99402636 0.00597364]
 [0.98703596 0.01296404]
 [0.9919501  0.0080499 ]
 [0.98687688 0.01312312]
 [0.99488669 0.00511331]
 [0.98947643 0.01052357]
 [0.99017504 0.00982496]
 [0.99175032 0.00824968]
 [0.98487479 0.01512521]
 [0.99067091 0.00932909]]
[[9.99999995e-01 4.53177934e-09]
 [9.99999863e-01 1.37248738e-07]
 [9.99999992e-01 8.19323638e-09]
 [9.99999961e-01 3.86307599e-08]
 [9.99999999e-01 1.44538132e-09]
 [9.99999959e-01 4.11653960e-08]
 [9.99999992e-01 8.31066378e-09]
 [9.99999987e-01 1.25390963e-08]
 [9.99999940e-01 5.97705622e-08]
 [9.99999975e-01 2.46149485e-08]]
```

Python 代码 19

```
# 导入操作系统库
import os
# 更改工作目录
os.chdir(r"D:\softwares\applied statistics\pythoncodelearning\chap1\sourcecode")
```

```python
# 导入基础计算库
import numpy as np
# 导入绘图库
import matplotlib.pyplot as plt
# 导入 logistic 回归模型
from sklearn.linear_model import LogisticRegression
# 导入分类数据集生成工具
from sklearn.datasets import make_blobs
# 导入决策边界可视化工具
from sklearn.inspection import DecisionBoundaryDisplay
# 导入绘图库中的字体管理包
from matplotlib import font_manager
# 实现中文字符正常显示
font = font_manager.FontProperties(fname=r"C:\Windows\Fonts\SimKai.ttf")
# 使用 seaborn 风格绘图
plt.style.use("seaborn-v0_8")
# 三个类别的中心
centers = [[-5, 0], [0, 1.5], [5, -1]]
# 生成数据集
X, y = make_blobs(
    n_samples=1000,
    n_features=2,
    centers=centers,
    random_state=40
)
# 两个特征，所以是两个维度
transformation = [[0.4, 0.2], [-0.4, 1.2]]
# X 做一个转换
X = np.dot(X, transformation)
# 在两个方法下绘图
for multi_class,num in zip(["multinomial", "ovr"], [25, 26]):
    # 建立 Logistic 回归模型
    clf = LogisticRegression(
        solver="sag",
        max_iter=100,
        random_state=42,
        multi_class=multi_class
    )
    # 模型拟合
    clf.fit(X, y)
    # 训练集的分类准确率
    print("training score : %.3f (%s)" % (clf.score(X, y), multi_class))
    # 开始绘图
```

```
    fig, ax = plt.subplots(figsize=(6,6), tight_layout=True)
    # 绘制决策边界图
    DecisionBoundaryDisplay.from_estimator(
        clf, # 模型
        X, # X
        response_method="predict", # 预测
        cmap=plt.cm.Paired, # 颜色
        ax=ax # 图形
    )
    ax.set_title("Decision surface of LogisticRegression (%s)" % multi_class)
    # 绘制训练集的散点
    colors = "bry"
    for i, color in zip(clf.classes_, colors):
        idx = np.where(y == i)
        # 散点
        ax.scatter(
            X[idx, 0], X[idx, 1],
            c=color,
            edgecolor="black",
            s=20
        )
    # 绘制 ovr 分类边界
    xmin, xmax = ax.get_xlim()
    ymin, ymax = ax.get_ylim()
    # 获取系数和偏置
    coef = clf.coef_
    intercept = clf.intercept_
    # 绘制超平面的函数
    def plot_hyperplane(c, color):
        def line(x0):
            return (-(x0 * coef[c, 0]) - intercept[c]) / coef[c, 1]

        ax.plot([xmin, xmax], [line(xmin), line(xmax)], ls="--", color=color)
        ax.set_xlim([xmin, xmax])
        ax.set_ylim([ymin, ymax])
    # 绘制超平面
    for i, color in zip(clf.classes_, colors):
        plot_hyperplane(i, color)

    plt.show()
    fig.savefig("../codeimage/code{}.pdf".format(num))
```

```
training score : 0.995 (multinomial)

training score : 0.976 (ovr)
```

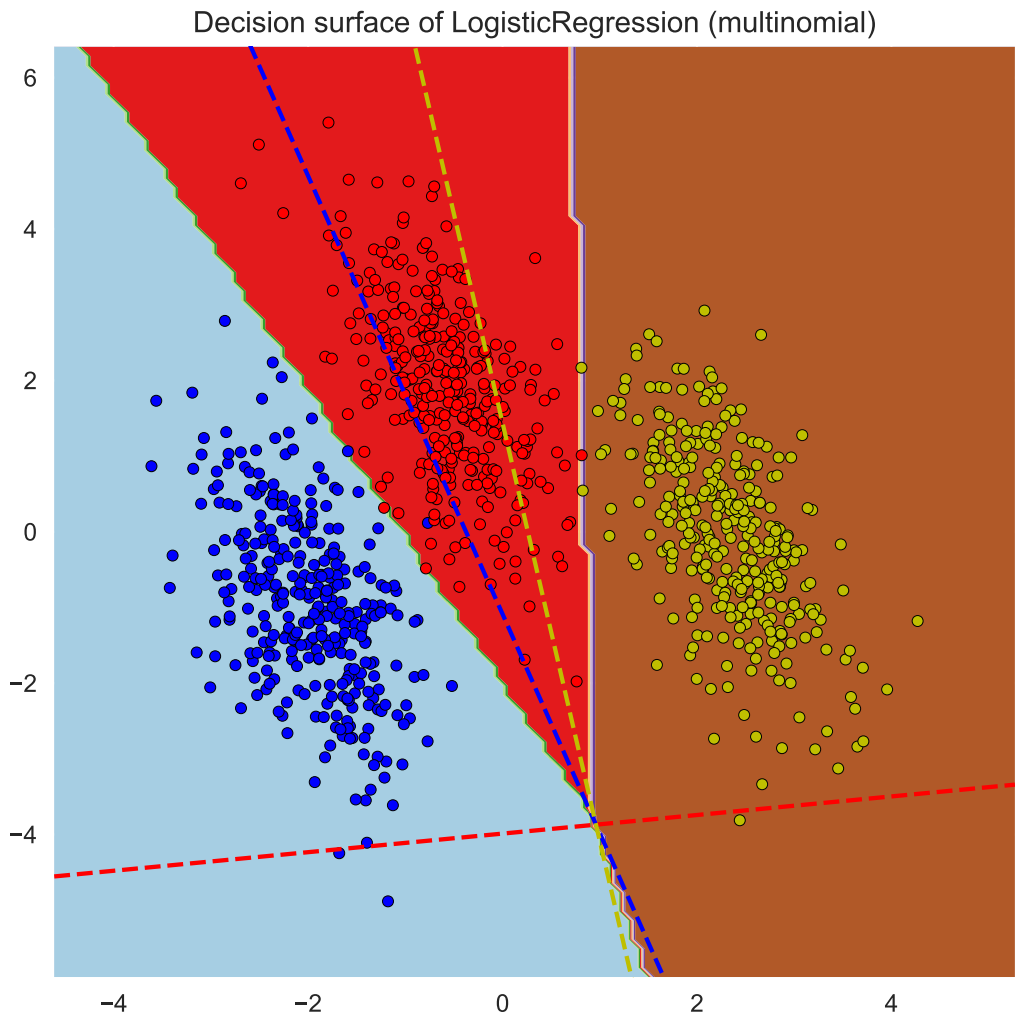
Decision surface of LogisticRegression (multinomial)
6
4
2
0
−2
−4
−4
−2
0
2
4

图 25: code25

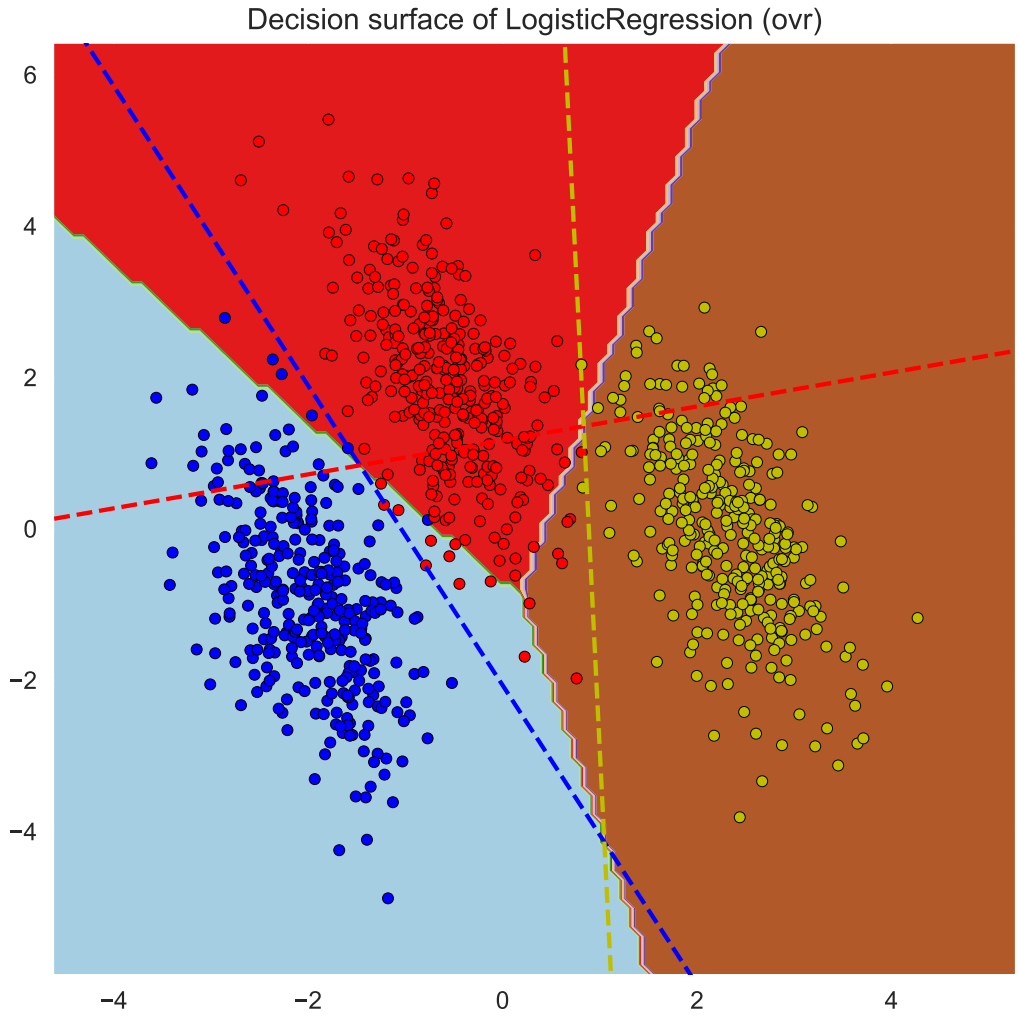

图 26: code26

Python 代码 20

```
# 导入操作系统库
import os
# 更改工作目录
os.chdir(r"D:\softwares\applied statistics\pythoncodelearning\chap1\sourcecode")
# 导入基础计算库
import numpy as np
# 导入绘图库
import matplotlib.pyplot as plt
# 导入泊松回归模型
from sklearn.linear_model import PoissonRegressor
# 导入管道工具
from sklearn.pipeline import make_pipeline, Pipeline
# 导入数据划分工具
from sklearn.model_selection import train_test_split
# 导入预处理工具
from sklearn.preprocessing import FunctionTransformer, OneHotEncoder
from sklearn.preprocessing import StandardScaler, KBinsDiscretizer
# 导入列变换工具
from sklearn.compose import ColumnTransformer
# 导入数据获取工具
from sklearn.datasets import fetch_openml
# 导入模型评估工具
from sklearn.metrics import mean_squared_error
# 导入绘图库中的字体管理包
from matplotlib import font_manager
# 实现中文字符正常显示
font = font_manager.FontProperties(fname=r"C:\Windows\Fonts\SimKai.ttf")
# 使用 seaborn 风格绘图
plt.style.use("seaborn-v0_8")
# 获取数据
df = fetch_openml(data_id=41214, as_frame=True, parser="pandas").frame
print(df.head())
# 新增一类
df["Frequency"] = df["ClaimNb"] / df["Exposure"]
# 标准化数据
# 构造管道，取对数变换，并且对其做标准化
log_scale_transformer = make_pipeline(
    FunctionTransformer(np.log, validate=False), StandardScaler()
)
# 列变换
linear_model_preprocessor = ColumnTransformer(
    [
```

```
        (
            "passthrough_numeric", # 变换名称
            "passthrough", # 变换方式
            ["BonusMalus"] # 变换对象
        ),
        (
            "binned_numeric", # 变换名称
            KBinsDiscretizer(n_bins=10, subsample=int(2e5), random_state=0), # 变换
方式
            ["VehAge", "DrivAge"] # 变换对象
        ),
        (
            "log_scaled_numeric", # 变换名称
            log_scale_transformer, # 变换方式
            ["Density"] # 变换对象
        ),
        (
            "onehot_categorical", # 变换名称
            OneHotEncoder(), # 变换方式
            ["VehBrand", "VehPower", "VehGas", "Region", "Area"] # 变换对象
        )
    ],
    remainder="drop" # 剩下的变量的处理方式
)
# 划分数据集，这是对一个 dataframe 进行划分
df_train, df_test = train_test_split(df, test_size=0.33, random_state=0)
# 训练集的样本量
n_samples = df_train.shape[0]
# 管道工具，建立泊松回归模型
poisson_glm = Pipeline(
    [
        (
            "preprocessor",
            linear_model_preprocessor
        ), # 变量预处理
        (
            "regressor",
            PoissonRegressor(
                alpha=1e-12, solver="newton-cholesky"
            )
        ) # 泊松回归模型
    ]
)
```

```
# 模型拟合
poisson_glm.fit(
    df_train, df_train["Frequency"], # X,  Y
    regressor__sample_weight=df_train["Exposure"] # 权重
)
# 模型预测
y_pred = poisson_glm.predict(df_test)
# MSE
mse = mean_squared_error(
    df_test["Frequency"],
    y_pred,
    sample_weight=df_test["Exposure"]
)
print("PoissonRegressor evaluation:", mse, sep="\n")
```

```
   IDpol  ClaimNb  Exposure Area  VehPower  VehAge  DrivAge  BonusMalus  \
0    1.0        1      0.10    D         5       0       55          50
1    3.0        1      0.77    D         5       0       55          50
2    5.0        1      0.75    B         6       2       52          50
3   10.0        1      0.09    B         7       0       46          50
4   11.0        1      0.84    B         7       0       46          50

  VehBrand     VehGas  Density Region
0      B12  'Regular'     1217    R82
1      B12  'Regular'     1217    R82
2      B12   'Diesel'       54    R22
3      B12   'Diesel'       76    R72
4      B12   'Diesel'       76    R72
PoissonRegressor evaluation:
0.5598099236977848
```

Python 代码 21

```
# 导入操作系统库
import os
# 更改工作目录
os.chdir(r"D:\softwares\applied statistics\pythoncodelearning\chap1\sourcecode")
# 导入基础计算库
import numpy as np
# 导入绘图库
import matplotlib.pyplot as plt
# 导入回归器
from sklearn.linear_model import RANSACRegressor, LinearRegression
# 导入数据集生成工具
```

```
from sklearn.datasets import make_regression
# 导入绘图库中的字体管理包
from matplotlib import font_manager
# 实现中文字符正常显示
font = font_manager.FontProperties(fname=r"C:\Windows\Fonts\SimKai.ttf")
# 使用 seaborn 风格绘图
plt.style.use("seaborn-v0_8")
# 设置样本量和离群值的样本量
n_samples = 1000
n_outliers = 50
# 生成数据集
X, y, coef = make_regression(
    n_samples=n_samples,
    n_features=1,
    n_informative=1,
    noise=10,
    coef=True,
    random_state=0,
)
# 添加离群值
# Add outlier data
np.random.seed(0)
X[:n_outliers] = 3 + 0.5 * np.random.normal(size=(n_outliers, 1))
y[:n_outliers] = -3 + 10 * np.random.normal(size=n_outliers)
# 构建线性模型
lr = LinearRegression()
# 模型拟合
lr.fit(X, y)
# 构建稳健回归
ransac = RANSACRegressor()
# 模型拟合
ransac.fit(X, y)
inlier_mask = ransac.inlier_mask_
outlier_mask = np.logical_not(inlier_mask)
# 预测
line_X = np.arange(X.min(), X.max())[:, np.newaxis]
# 线性模型预测
line_y = lr.predict(line_X)
# 稳健回归预测
line_y_ransac = ransac.predict(line_X)
print("Estimated coefficients (true, linear regression, RANSAC):")
print(coef, lr.coef_, ransac.estimator_.coef_,sep="\n")
fig, ax = plt.subplots(figsize=(6,6), tight_layout=True)
```

```
# 散点图
ax.scatter(
    X[inlier_mask], y[inlier_mask],
    color="yellowgreen", marker=".", label="Inliers"
)
ax.scatter(
    X[outlier_mask], y[outlier_mask],
    color="gold", marker=".", label="Outliers"
)
ax.plot(line_X, line_y, color="navy", linewidth=2, label="Linear regressor")
ax.plot(
    line_X,
    line_y_ransac,
    color="cornflowerblue",
    linewidth=2,
    label="RANSAC regressor",
)
ax.legend(loc="lower right")
ax.set_xlabel("Input")
ax.set_ylabel("Response")
plt.show()
fig.savefig("../codeimage/code27.pdf")

Estimated coefficients (true, linear regression, RANSAC):
82.1903908407869
[54.17236387]
[82.08533159]
```

Python 代码 22

```
# 导入操作系统库
import os
# 更改工作目录
os.chdir(r"D:\softwares\applied statistics\pythoncodelearning\chap1\sourcecode")
# 导入时间库
import time
# 导入基础计算库
import numpy as np
# 导入绘图库
import matplotlib.pyplot as plt
# 导入回归器
from sklearn.linear_model import RANSACRegressor, LinearRegression,␣
 ↪TheilSenRegressor
# 导入绘图库中的字体管理包
```

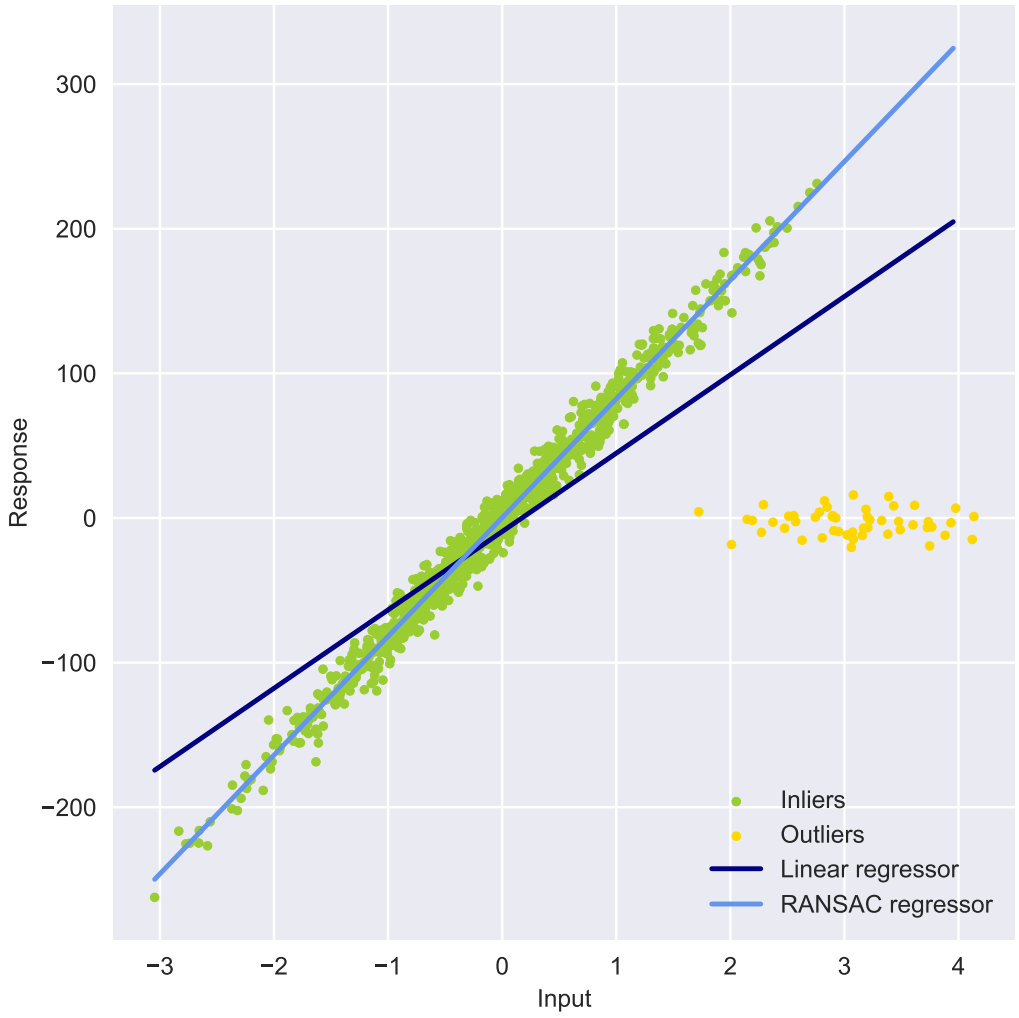
300
200
100
0
−100
−200
Response
−3
−2
−1
0
1
2
3
4
Input
Inliers
Outliers
Linear regressor
RANSAC regressor

图 27: code27

```
from matplotlib import font_manager
# 实现中文字符正常显示
font = font_manager.FontProperties(fname=r"C:\Windows\Fonts\SimKai.ttf")
# 使用 seaborn 风格绘图
plt.style.use("seaborn-v0_8")
# 构建模型
estimators = [
    ("OLS", LinearRegression()),
    ("Theil-Sen", TheilSenRegressor(random_state=42)),
    ("RANSAC", RANSACRegressor(random_state=42)),
]
colors = {"OLS": "turquoise", "Theil-Sen": "gold", "RANSAC": "lightgreen"}
lw = 2
# 生成数据集，离群值在 X
np.random.seed(0)
n_samples = 200
# Linear model y = 3*x + N(2, 0.1**2)
x = np.random.randn(n_samples)
w = 3.0
c = 2.0
noise = 0.1 * np.random.randn(n_samples)
y = w * x + c + noise
# 10% outliers
y[-20:] += -20 * x[-20:]
X = x[:, np.newaxis]
# 开始绘图
fig1, ax = plt.subplots(figsize=(6,6), tight_layout=True)
ax.scatter(x, y, color="indigo", marker="x", s=40)
# 预测点的 X
line_x = np.array([-3, 3])
for name, estimator in estimators:
    t0 = time.time()
    # 模型拟合
    estimator.fit(X, y)
    elapsed_time = time.time() - t0
    # 预测
    y_pred = estimator.predict(line_x.reshape(2, 1))
    ax.plot(
        line_x,
        y_pred,
        color=colors[name],
        linewidth=lw,
        label="%s (fit time: %.2fs)" % (name, elapsed_time),
```

```
    )
ax.legend(loc="upper left")
ax.set_title("Corrupt y")
plt.show()
fig1.savefig("../codeimage/code28.pdf")

# 生成数据集，离群值在 y
np.random.seed(0)
# Linear model y = 3*x + N(2, 0.1**2)
x = np.random.randn(n_samples)
noise = 0.1 * np.random.randn(n_samples)
y = 3 * x + 2 + noise
# 10% outliers
x[-20:] = 9.9
y[-20:] += 22
X = x[:, np.newaxis]
# 开始绘图
fig2, ax = plt.subplots(figsize=(6,6), tight_layout=True)
ax.scatter(x, y, color="indigo", marker="x", s=40)
# 预测点的 X
line_x = np.array([-3, 10])
for name, estimator in estimators:
    t0 = time.time()
    # 模型拟合
    estimator.fit(X, y)
    elapsed_time = time.time() - t0
    # 预测
    y_pred = estimator.predict(line_x.reshape(2, 1))
    ax.plot(
        line_x,
        y_pred,
        color=colors[name],
        linewidth=lw,
        label="%s (fit time: %.2fs)" % (name, elapsed_time),
    )
ax.legend(loc="upper left")
ax.set_title("Corrupt x")
plt.show()
fig2.savefig("../codeimage/code29.pdf")
```

Python 代码 23

```
# 导入操作系统库
import os
```

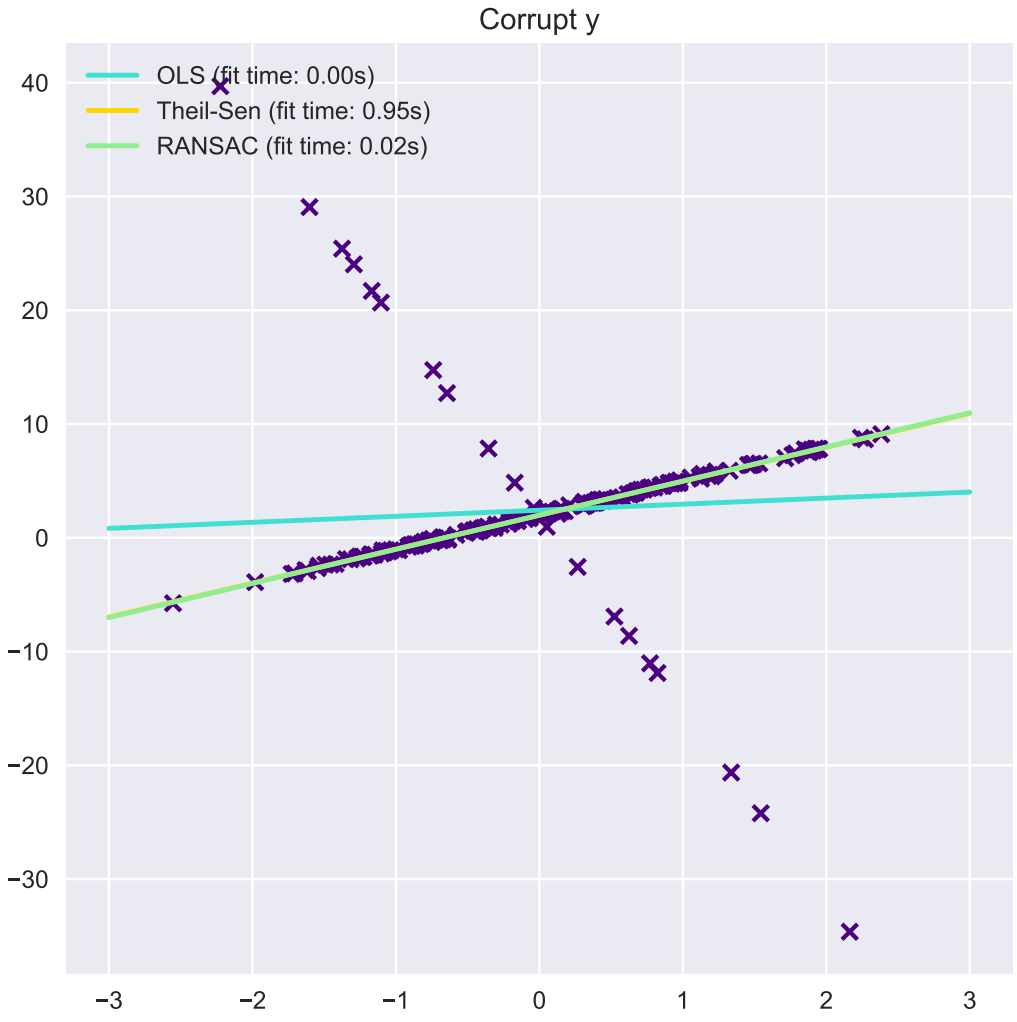
Corrupt y
OLS (fit time: 0.00s)
Theil-Sen (fit time: 0.95s)
RANSAC (fit time: 0.02s)
40
30
20
10
0
−10
−20
−30
−3
−2
−1
0
1
2
3

图 28: code28

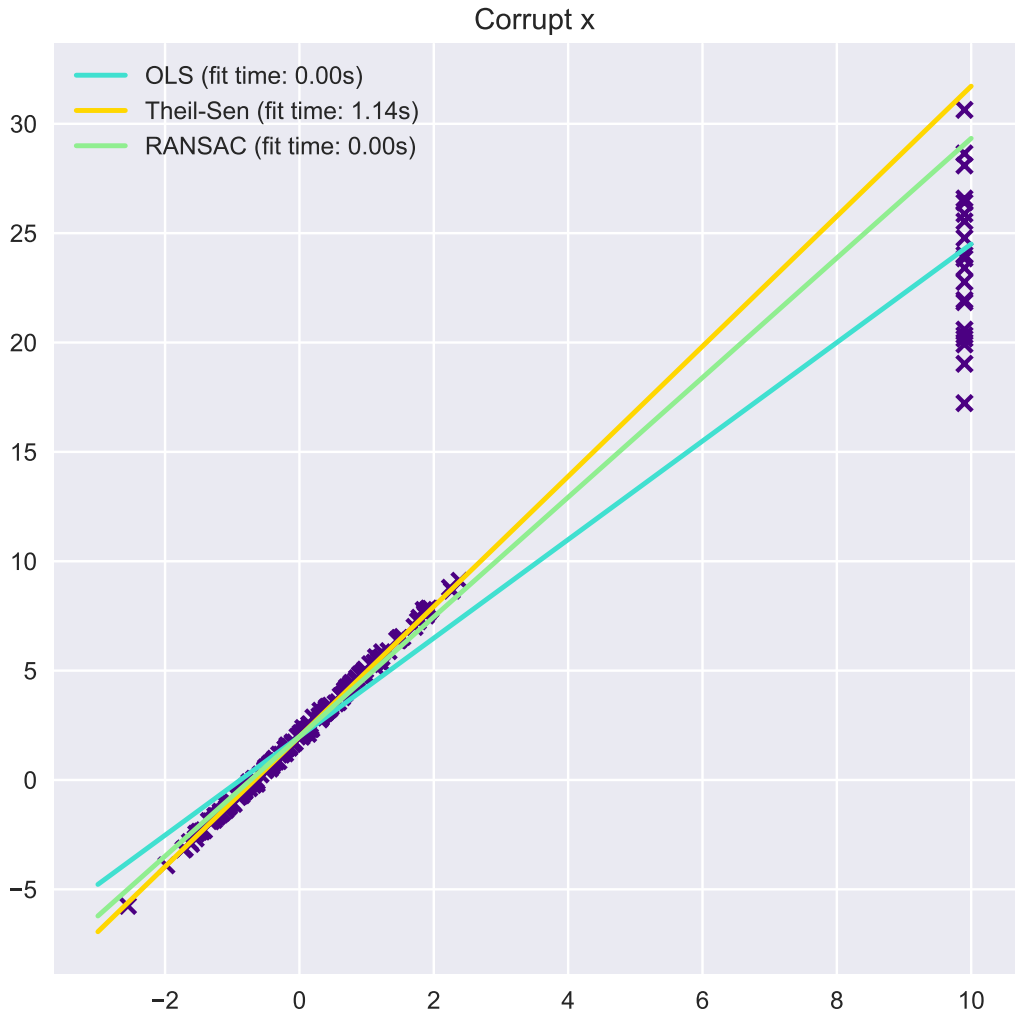
Corrupt x
OLS (fit time: 0.00s)
Theil-Sen (fit time: 1.14s)
RANSAC (fit time: 0.00s)
30
25
20
15
10
5
0
−5
−2
0
2
4
6
8
10

图 29: code29

```
# 更改工作目录
os.chdir(r"D:\softwares\applied statistics\pythoncodelearning\chap1\sourcecode")
# 导入基础计算库
import numpy as np
# 导入绘图库
import matplotlib.pyplot as plt
# 导入回归器
from sklearn.linear_model import HuberRegressor, Ridge
# 导入数据生成工具
from sklearn.datasets import make_regression
# 导入绘图库中的字体管理包
from matplotlib import font_manager
# 实现中文字符正常显示
font = font_manager.FontProperties(fname=r"C:\Windows\Fonts\SimKai.ttf")
# 使用 seaborn 风格绘图
plt.style.use("seaborn-v0_8")
# 生成数据
np.random.seed(0)
X, y = make_regression(
    n_samples=20, n_features=1,
    random_state=0,
    noise=4.0,
    bias=100.0
)
# 添加噪声
X_outliers = np.random.normal(0, 0.5, size=(4, 1))
y_outliers = np.random.normal(0, 2.0, size=4)
X_outliers[:2, :] += X.max() + X.mean() / 4.0
X_outliers[2:, :] += X.min() - X.mean() / 4.0
y_outliers[:2] += y.min() - y.mean() / 4.0
y_outliers[2:] += y.max() + y.mean() / 4.0
X = np.vstack((X, X_outliers))
y = np.concatenate((y, y_outliers))
# 开始绘图
fig, ax = plt.subplots(figsize=(6,6))
ax.plot(X, y, "b.")
# 颜色和线型
colors = ["r-", "b-", "y-", "m-"]
# 预测的 x
x = np.linspace(X.min(), X.max(), 7)
# epsilon 的值
epsilon_values = [1, 1.5, 1.75, 1.9]
for k, epsilon in enumerate(epsilon_values):
```

```
    # 构建 Huber 回归模型
    huber = HuberRegressor(alpha=0.0, epsilon=epsilon)
    # 模型拟合
    huber.fit(X, y)
    # 预测
    y_pred = huber.coef_ * x + huber.intercept_
    ax.plot(x, y_pred, colors[k], label="huber loss, %s" % epsilon)

# 岭回归模型，没有惩罚，就是 OLS
ridge = Ridge(alpha=0.0, random_state=0)
# 模型拟合
ridge.fit(X, y)
# 预测值
y_pred = ridge.coef_ * x + ridge.intercept_
ax.plot(x, y_pred, "g-", label="ridge regression")
ax.set_title("Comparison of HuberRegressor vs Ridge")
ax.set_xlabel("X")
ax.set_ylabel("y")
ax.legend(loc=0)
plt.show()
fig.savefig("../codeimage/code30.pdf")
```

Python 代码 24

```
# 导入操作系统库
import os
# 更改工作目录
os.chdir(r"D:\softwares\applied statistics\pythoncodelearning\chap1\sourcecode")
# 导入基础计算库
import numpy as np
# 导入绘图库
import matplotlib.pyplot as plt
# 导入回归器
from sklearn.linear_model import QuantileRegressor
# 导入版本号
from sklearn.utils.fixes import sp_version, parse_version
# 导入绘图库中的字体管理包
from matplotlib import font_manager
# 实现中文字符正常显示
font = font_manager.FontProperties(fname=r"C:\Windows\Fonts\SimKai.ttf")
# 使用 seaborn 风格绘图
plt.style.use("seaborn-v0_8")
# 生成数据
np.random.seed(42)
```

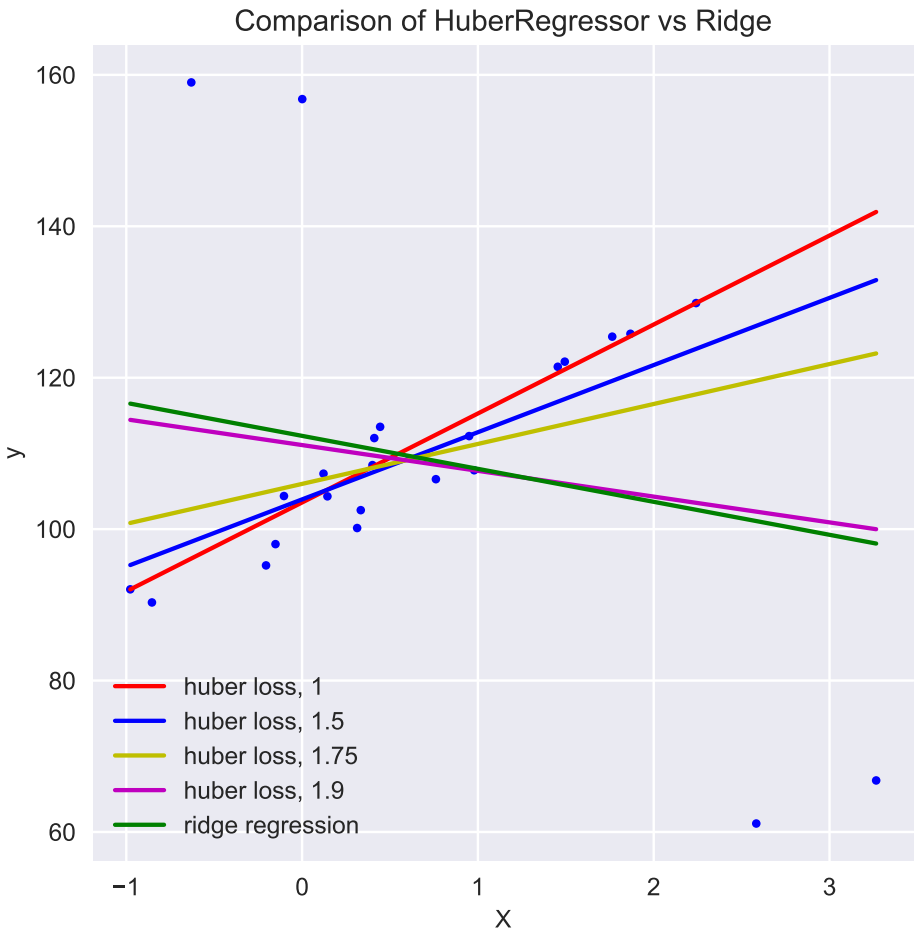
Comparison of HuberRegressor vs Ridge
160
140
120
100
80
60
y
−1
0
1
2
3
X
huber loss, 1
huber loss, 1.5
huber loss, 1.75
huber loss, 1.9
ridge regression

图 30: code30

```
x = np.linspace(start=0, stop=10, num=100)
X = x[:, np.newaxis]
y_true_mean = 10 + 0.5 * x
# 添加异方差的正态误差
y_normal = y_true_mean + np.random.normal(
    loc=0,
    scale=0.5 + 0.5 * x,
    size=x.shape[0]
)
# 添加非对称的帕累托误差
a = 5
y_pareto = y_true_mean + 10 * (np.random.pareto(
    a, size=x.shape[0]) - 1 / (a - 1)
)
# 开始绘图
fig1, axs = plt.subplots(nrows=1, ncols=2, figsize=(15, 7), sharex="row",␣
 ↪sharey="row")
# 第一幅图
axs[0].plot(x, y_true_mean, label="True mean")
axs[0].scatter(x, y_normal, color="black", alpha=0.5, label="Observations")
axs[0].set_title("Dataset with heteroscedastic Normal distributed targets")
axs[0].legend()
axs[0].set_ylabel("y")
axs[0].set_xlabel("x")
# 第二幅图
axs[1].plot(x, y_true_mean, label="True mean")
axs[1].scatter(x, y_pareto, color="black", alpha=0.5, label="Observations")
axs[1].set_title("Dataset with asymmetric Pareto distributed target")
axs[1].legend()
axs[1].set_ylabel("y")
axs[1].set_xlabel("x")
plt.show()
fig1.savefig("../codeimage/code31.pdf")

solver = "highs" if sp_version >= parse_version("1.6.0") else "interior-point"
# 分位点
quantiles = [0.05, 0.5, 0.95]
# 预测值容器
predictions = {}
# 离群值预测
out_bounds_predictions = np.zeros_like(y_true_mean, dtype=np.bool_)
for quantile in quantiles:
    # 建立模型
```

```
    qr = QuantileRegressor(
        quantile=quantile,
        alpha=0,
        solver=solver
    )
    # 模型拟合
    qr.fit(X, y_normal)
    # 预测
    y_pred = qr.predict(X)
    # 加入字典中
    predictions[quantile] = y_pred
    # 判断是否离群值
    if quantile == min(quantiles):
        out_bounds_predictions = np.logical_or(
            out_bounds_predictions, y_pred >= y_normal
        )
    elif quantile == max(quantiles):
        out_bounds_predictions = np.logical_or(
            out_bounds_predictions, y_pred <= y_normal
        )
fig2, ax = plt.subplots(figsize=(6,6))
ax.plot(
    X, y_true_mean,
    color="black", linestyle="dashed",
    label="True mean"
)
# 预测直线
for quantile, y_pred in predictions.items():
    ax.plot(X, y_pred, label=f"Quantile: {quantile}")
# 散点图
ax.scatter(
    x[out_bounds_predictions],
    y_normal[out_bounds_predictions],
    color="black",
    marker="+",
    alpha=0.5,
    label="Outside interval",
)
# 散点
ax.scatter(
    x[~out_bounds_predictions],
    y_normal[~out_bounds_predictions],
    color="black",
```

```
    alpha=0.5,
    label="Inside interval",
)
# 显示图例
ax.legend()
ax.set_xlabel("x")
ax.set_ylabel("y")
ax.set_title("Quantiles of heteroscedastic Normal distributed target")
plt.show()
fig2.savefig("../codeimage/code32.pdf")
```

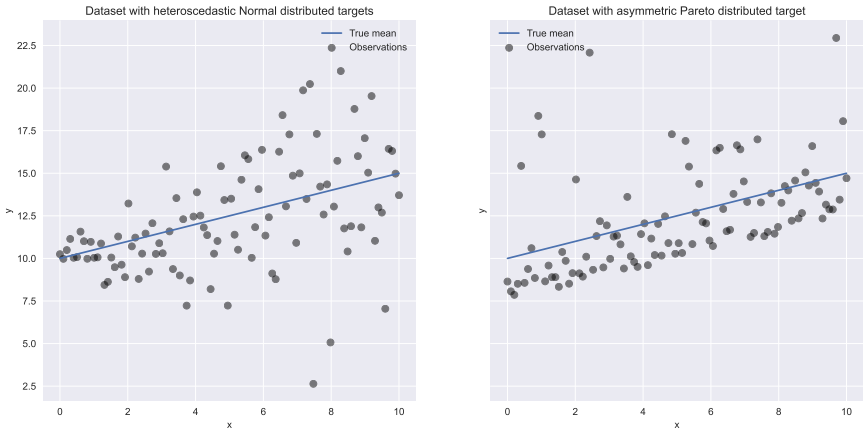

图 31: code31

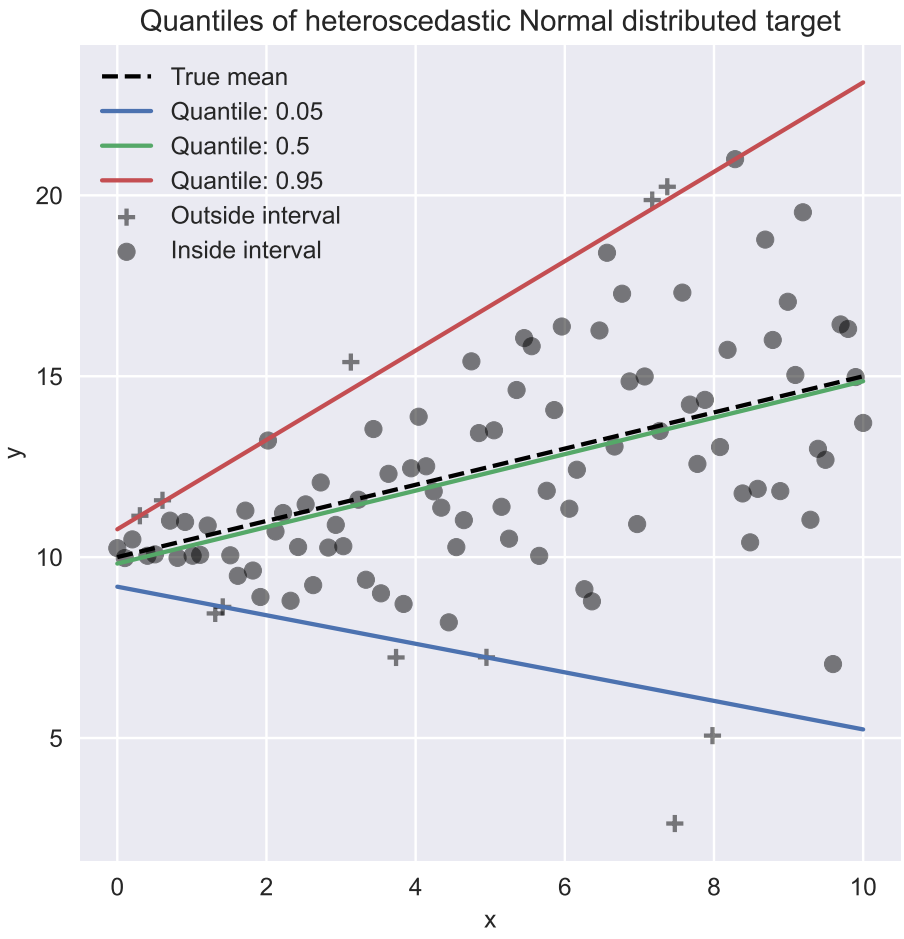
Quantiles of heteroscedastic Normal distributed target
True mean
Quantile: 0.05
Quantile: 0.5
Quantile: 0.95
Outside interval
Inside interval
20
15
10
5
y
0
2
4
6
8
10
x

图 32: code32