第 5 章

CHAPTER 5

线 性 回 归

线性模型是一类统计模型的总称，它包括了线性回归模型、方差分析模型、协方差分析模型和线性混合效应模型（或称方差分量模型）等。许多生物、医学、经济、管理、地质、气象、农业、工业、工程技术等领域的现象都可以用线性模型来近似描述。因此线性模型成为现代统计学中应用最为广泛的模型之一。

5.1 概述

给定样本 $\boldsymbol{x}$，用列向量表示该样本 $\boldsymbol{x}=(x^{(1)},x^{(2)},\cdots,x^{(n)})^{\mathrm{T}}$。样本有 n 种特征，我们用 $x^{(i)}$ 表示样本 $\boldsymbol{x}$ 的第 i 个特征。线性模型（Linear model）的形式为：

$$f(\boldsymbol{x})=\boldsymbol{w}\cdot\boldsymbol{x}+b$$

其中 $\boldsymbol{w}=(w^{(1)},w^{(2)},\cdots,w^{(n)})^{\mathrm{T}}$ 为每个特征对应的权重生成的权重向量，称为权重向量，权重向量直观地表达了各个特征在预测中的重要性。

线性模型中的“线性”其实就是一系列一次特征的线性组合，在二维空间中是一条直线，在三维空间中是一个平面，然后推广到 n 维空间，这样可以理解为广义的线性模型。

线性模型非常简单，易于建模，应用广泛，主要包括岭回归、lasso 回归、Elastic Net、逻辑回归、线性判别分析等。

5.2 普通线性回归

线性回归是一种回归分析技术。回归分析本质上就是一个函数估计的问题（函数估计包括参数估计和非参数估计两类），就是找出因变量和自变量之间的因果关系。回归分析的因变量应该是连续变量，如果因变量为离散变量，则问题转化为分类问题，回归分析是一个有监督学习的问题。

5.2.1 基本概述

给定数据集 $T=\{(\boldsymbol{x}_1,y_1),(\boldsymbol{x}_2,y_2),\cdots,(\boldsymbol{x}_N,y_N)\}$，$\boldsymbol{x}_i\in X\subseteq\mathbf{R}^n$，$y_i\in Y\subseteq\mathbf{R}$，$i=1,2,\cdots,N$，其中 $\boldsymbol{x}_i=(x_i^{(1)},x_i^{(2)},\cdots,x_i^{(n)})^{\mathrm{T}}$。需要学习的模型为：

$$f(\boldsymbol{x})=\boldsymbol{w}\cdot\boldsymbol{x}+b$$

也即根据已知的数据集 T 来计算参数 $\boldsymbol{w}$ 和 b。

对于给定的样本 $\boldsymbol{x}$，其预测值为 $\hat{y}_i=f(\boldsymbol{x}_i)=\boldsymbol{w}\cdot\boldsymbol{x}_i+b$。采用平方损失函数，则在训练集 T 上，模型的损失函数为：

$$L(f)=\sum_{i=1}^{N}(\hat{y}_i-y_i)^2=\sum_{i=1}^{N}(\boldsymbol{w}\cdot\boldsymbol{x}_i+b-y_i)^2$$

我们的目标是损失函数的最小化，即：

$$(\boldsymbol{w}^*,b^*)=\underset{\boldsymbol{w},b}{\operatorname{argmin}}\sum_{i=1}^{N}(\boldsymbol{w}\cdot\boldsymbol{x}_i+b-y_i)^2$$

可以采用梯度下降法来求解上述最优化问题的数值解。在使用梯度下降法时，要注意特征归一化(Feature Scaling)。

特征归一化有两个好处：

(1) 提升模型的收敛速度，比如两个特征 x_1 和 x_2，x_1 的取值为 0～2000，而 x_2 的取值为 1～5，假如只有这两个特征，对其进行优化时，会得到一个窄长的椭圆形，导致在梯度下降时，梯度的方向为垂直等高线的方向而走“之”字形路线，这样会使迭代很慢。相比之下，归一化之后，是一个圆形，梯度的方向为直接指向圆心，迭代就会很快。可见，归一化可以大大减少寻找最优解的时间。

(2) 提升模型精度，归一化的另一个好处是提高精度，这在涉及一些距离计算的算法时效果显著，比如算法要计算欧几里得距离，上面 x_2 的取值范围比较小，涉及距离计算时其对结果的影响远比 x_1 带来的小，所以这就会造成精度的损失。

因此归一化很有必要，它可以让各个特征对结果做出的贡献相同。在求解线性回归的模型时，还有一个问题要注意，那就是特征组合问题，比如房子的长度和宽度作为两个特征参与模型的构造，不如将二者相乘得到的面积作为一个特征来进行求解，这样就在特征选择上做了减少维度的工作。

上述最优化问题实际上是有解析解的，可以用最小二乘法求解解析解，该问题称为多元线性回归(multivariate linear regression)。令：

$$\tilde{\boldsymbol{w}}=(w^{(1)},w^{(2)},\cdots,w^{(n)},b)^{\mathrm{T}}=(\boldsymbol{w}^{\mathrm{T}},b)^{\mathrm{T}}$$

$$\tilde{\boldsymbol{x}}=(x^{(1)},x^{(2)},\cdots,x^{(n)},1)^{\mathrm{T}}=(\boldsymbol{x}^{\mathrm{T}},1)^{\mathrm{T}}$$

$$\boldsymbol{y}=(y_1,y_2,\cdots,y_N)^{\mathrm{T}}$$

则：

$$\sum_{i=1}^{N}(\boldsymbol{w}\cdot\boldsymbol{x}_i+b-y_i)^2=(\boldsymbol{y}-(\tilde{\boldsymbol{x}}_1,\tilde{\boldsymbol{x}}_2,\cdots,\tilde{\boldsymbol{x}}_N)^{\mathrm{T}}\tilde{\boldsymbol{w}})^{\mathrm{T}}(\boldsymbol{y}-(\tilde{\boldsymbol{x}}_1,\tilde{\boldsymbol{x}}_2,\cdots,\tilde{\boldsymbol{x}}_N)^{\mathrm{T}}\tilde{\boldsymbol{w}})$$

令：

$$\boldsymbol{x}=(\tilde{\boldsymbol{x}}_1,\tilde{\boldsymbol{x}}_2,\cdots,\tilde{\boldsymbol{x}}_N)^{\mathrm{T}}=\begin{bmatrix}\tilde{\boldsymbol{x}}_1^{\mathrm{T}}\\ \tilde{\boldsymbol{x}}_2^{\mathrm{T}}\\ \vdots\\ \tilde{\boldsymbol{x}}_N^{\mathrm{T}}\end{bmatrix}=\begin{bmatrix}x_1^{(1)} & x_1^{(2)} & \cdots & x_1^{(n)} & 1\\ x_2^{(1)} & x_2^{(2)} & \cdots & x_2^{(n)} & 1\\ \vdots & \vdots & \ddots & \vdots & \vdots\\ x_N^{(1)} & x_N^{(2)} & \cdots & x_N^{(n)} & 1\end{bmatrix}$$

则：

$$\tilde{\boldsymbol{w}}^{*}=\underset{\tilde{\boldsymbol{w}}}{\operatorname{argmin}}(\boldsymbol{y}-\boldsymbol{x}\tilde{\boldsymbol{w}})^{\mathrm{T}}(\boldsymbol{y}-\boldsymbol{x}\tilde{\boldsymbol{w}})$$

令 $E_{\tilde{\boldsymbol{w}}}=(\boldsymbol{y}-\boldsymbol{x}\tilde{\boldsymbol{w}})^{\mathrm{T}}(\boldsymbol{y}-\boldsymbol{x}\tilde{\boldsymbol{w}})$，求它的极小值。对 $\tilde{\boldsymbol{w}}$ 求导令导数为零，得到解析解：

$$\frac{\partial E_{\tilde{\boldsymbol{w}}}}{\partial\tilde{\boldsymbol{w}}}=2\boldsymbol{x}^{\mathrm{T}}(\boldsymbol{x}\tilde{\boldsymbol{w}}-\boldsymbol{y})=\boldsymbol{0}\Rightarrow\boldsymbol{x}^{\mathrm{T}}\boldsymbol{x}\tilde{\boldsymbol{w}}-\boldsymbol{x}^{\mathrm{T}}\boldsymbol{y}$$

- 当 $\boldsymbol{x}^{\mathrm{T}}\boldsymbol{x}$ 为满秩矩阵或者正定矩阵时，可得：

  $$\tilde{\boldsymbol{w}}^{*}=(\boldsymbol{x}^{\mathrm{T}}\boldsymbol{x})^{-1}\boldsymbol{x}^{\mathrm{T}}\boldsymbol{y}$$

  其中 $(\boldsymbol{x}^{\mathrm{T}}\boldsymbol{x})^{-1}$ 为 $\boldsymbol{x}^{\mathrm{T}}\boldsymbol{x}$ 的逆矩阵。于是得到的多元线性回归模型为：

  $$f(\tilde{\boldsymbol{x}}_i)=\tilde{\boldsymbol{x}}_i^{\mathrm{T}}\tilde{\boldsymbol{w}}^{*}$$

- $\boldsymbol{x}^{\mathrm{T}}\boldsymbol{x}$ 不是满秩矩阵。比如 $N<n$（样本数量小于特征种类的数量），根据 $\boldsymbol{x}$ 的秩小于或等于 (N,n) 中的最小值，即小于或等于 N（矩阵的秩一定小于或等于矩阵的行数和列数）；而矩阵 $\boldsymbol{x}^{\mathrm{T}}\boldsymbol{x}$ 是 $n\times n$ 大小的，它的秩一定小于或等于 N，因此不是满秩矩阵。此时存在多个解析解。常见的做法是引入正则化项，如 L_1 正则化或者 L_2 正则化。以 L_2 正则化为例：

  $$\tilde{\boldsymbol{w}}^{*}=\underset{\tilde{\boldsymbol{w}}}{\operatorname{argmin}}\left[(\boldsymbol{y}-\boldsymbol{x}\tilde{\boldsymbol{w}})^{\mathrm{T}}(\boldsymbol{y}-\boldsymbol{x}\tilde{\boldsymbol{w}})+\lambda\|\tilde{\boldsymbol{w}}\|_2^2\right]$$

  其中，$\lambda>0$ 调整正则化项与均方误差的比例；$\|\cdots\|_2$ 为 L_2 范数。

根据上述原理，得到多元线性回归算法。

- 输入：数据集 $T=\{(\boldsymbol{x}_1,y_1),(\boldsymbol{x}_2,y_2),\cdots,(\boldsymbol{x}_N,y_N)\}$，$\boldsymbol{x}_i\in X\subseteq\mathbf{R}^n$，$y_i\in Y\subseteq\mathbf{R}$，$i=1,2,\cdots,N$，正则化项系数 $\lambda>0$。
- 输出：

  $$f(\boldsymbol{x})=\boldsymbol{w}\cdot\boldsymbol{x}+b$$

- 算法步骤。

  ① 令：

  $$\tilde{\boldsymbol{w}}=(w^{(1)},w^{(2)},\cdots,w^{(n)},b)^{\mathrm{T}}=(\boldsymbol{w}^{\mathrm{T}},b)^{\mathrm{T}}$$
  $$\tilde{\boldsymbol{x}}=(x^{(1)},x^{(2)},\cdots,x^{(n)},1)^{\mathrm{T}}=(\boldsymbol{x}^{\mathrm{T}},1)^{\mathrm{T}}$$
  $$\boldsymbol{y}=(y_1,y_2,\cdots,y_N)^{\mathrm{T}}$$

  计算：

$$\boldsymbol{x}=(\tilde{\boldsymbol{x}}_1,\tilde{\boldsymbol{x}}_2,\cdots,\tilde{\boldsymbol{x}}_N)^{\mathrm{T}}=\begin{bmatrix}\tilde{\boldsymbol{x}}_1^{\mathrm{T}}\\ \tilde{\boldsymbol{x}}_2^{\mathrm{T}}\\ \vdots\\ \tilde{\boldsymbol{x}}_N^{\mathrm{T}}\end{bmatrix}=\begin{bmatrix}x_1^{(1)} & x_1^{(2)} & \cdots & x_1^{(n)} & 1\\ x_2^{(1)} & x_2^{(2)} & \cdots & x_2^{(n)} & 1\\ \vdots & \vdots & \ddots & \vdots & 1\\ x_N^{(1)} & x_N^{(2)} & \cdots & x_N^{(n)} & 1\end{bmatrix}$$

② 求解：

$$\tilde{\boldsymbol{w}}^{*}=\underset{\tilde{\boldsymbol{w}}}{\operatorname{argmin}}\left[(\boldsymbol{y}-\boldsymbol{x}\tilde{\boldsymbol{w}})^{\mathrm{T}}(\boldsymbol{y}-\boldsymbol{x}\tilde{\boldsymbol{w}})+\lambda\|\tilde{\boldsymbol{w}}\|_2^2\right]$$

③ 最终的模型：

$$f(\tilde{\boldsymbol{x}}_i)=\tilde{\boldsymbol{x}}_i^{\mathrm{T}}\tilde{\boldsymbol{w}}^{*}$$

5.2.2 Python 实现

前面对普通线性回归的求解过程进行了介绍，下面直接通过 Python 来实现线性回归的求解。

【例 5-1】 Python 实现线性回归模型。

```
#导入必要的编程库
import numpy as np
from pylab import *
"""训练得到 w 和 b 的向量"""
def train_wb(X, y):
    """
    :param X:N * D 的数据
    :param y:X 对应的 y 值
    :return: 返回(w,b)的向量
    """
    if np.linalg.det(X.T * X) != 0:
        wb = ((X.T.dot(X).I).dot(X.T)).dot(y)
        return wb
def test(x, wb):
    return x.T.dot(wb)
"""获得数据的函数"""
def getdata():
    x = []; y = []
    file = open("ex0.txt", 'r')
    for line in file.readlines():
        temp = line.strip().split("\t")
        x.append([float(temp[0]),float(temp[1])])
        y.append(float(temp[2]))
    return (np.mat(x), np.mat(y).T)
"""画图函数,分别把训练用的数据的散点图还有回归直线画出来了"""
def draw(x, y, wb):
    #画回归直线 y = wx + b
    a = np.linspace(0, np.max(x))                    #横坐标的取值范围
    b = wb[0] + a * wb[1]
```

```
    plot(x, y, '.')
    plot(a, b)
    show()
X, y = getdata()
wb = train_wb(X, y)
draw(X[:, 1], y, wb.tolist())
```

运行程序，效果如图 5-1 所示。

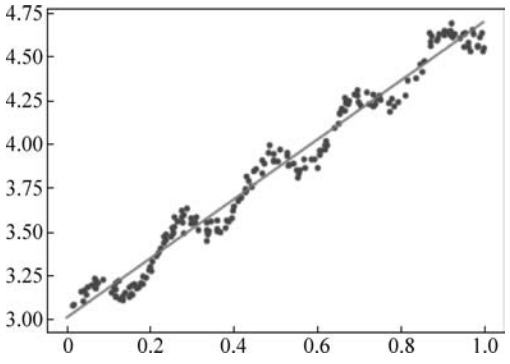


图 5-1　线性回归

【例 5-2】 以表 5-1 为例，针对运输里程、运输次数与运输总时间的关系，建立多元线性回归模型。

表 5-1　运输里程、运输次数与运输总时间的关系

运 输 里 程	运 输 次 数	运输总时间
100	4	9.3
50	3	4.8
100	4	8.9
100	2	6.5
50	2	4.2
80	2	6.2
75	3	7.4
65	4	6.0
90	3	7.6
90	2	6.1

Python 的实现代码为：

```
import numpy as np
from sklearn import datasets,linear_model
#定义训练数据
x = np.array([[100,4,9.3],[50,3,4.8],[100,4,8.9],
              [100,2,6.5],[50,2,4.2],[80,2,6.2],
              [75,3,7.4],[65,4,6],[90,3,7.6],[90,2,6.1]])
```

```
print(x)
X = x[:,:-1]
Y = x[:,-1]
print(X,Y)
#训练数据
regr = linear_model.LinearRegression()
regr.fit(X,Y)
print('coefficients(b1,b2...):',regr.coef_)
print('intercept(b0):',regr.intercept_)
#预测
x_test = np.array([[102,6],[100,4]])
y_test = regr.predict(x_test)
print(y_test)
```

运行程序,输出如下:

```
[[100.    4.    9.3]
 [ 50.    3.    4.8]
 [100.    4.    8.9]
 [100.    2.    6.5]
 [ 50.    2.    4.2]
 [ 80.    2.    6.2]
 [ 75.    3.    7.4]
 [ 65.    4.    6. ]
 [ 90.    3.    7.6]
 [ 90.    2.    6.1]]
[[100.   4.]
 [ 50.   3.]
 [100.   4.]
 [100.   2.]
 [ 50.   2.]
 [ 80.   2.]
 [ 75.   3.]
 [ 65.   4.]
 [ 90.   3.]
 [ 90.   2.]] [9.3 4.8 8.9 6.5 4.2 6.2 7.4 6.  7.6 6.1]
coefficients(b1,b2...): [0.0611346  0.92342537]
intercept(b0): -0.868701466781709
[10.90757981  8.93845988]
```

如果特征向量中存在分类型变量,例如车型,则需要进行特殊处理,数据如表 5-2 所示。

表 5-2 车型、运输里程、运输次数与运输总时间的关系

运输里程	输出次数	车型	隐式转换	运输总时间
100	4	1	010	9.3
50	3	0	100	4.8
100	4	1	010	8.9
100	2	2	001	6.5
50	2	2	001	4.2

续表

运输里程	输出次数	车　　型	隐式转换	运输总时间
80	2	1	010	6.2
75	3	1	010	7.4
65	4	0	100	6.0
90	3	0	100	7.6
100	4	1	010	9.3
50	3	0	100	4.8
100	4	1	010	8.9
100	2	2	001	6.5

Python 的实现代码为：

```
import numpy as np
from sklearn.feature_extraction import DictVectorizer
from sklearn import linear_model
#定义数据集
x = np.array([[100,4,1,9.3],[50,3,0,4.8],[100,4,1,8.9],
              [100,2,2,6.5],[50,2,2,4.2],[80,2,1,6.2],
              [75,3,1,7.4],[65,4,0,6],[90,3,0,7.6],
              [100,4,1,9.3],[50,3,0,4.8],[100,4,1,8.9],[100,2,2,6.5]])
x_trans = []
for i in range(len(x)):
    x_trans.append({'x1':str(x[i][2])})
vec = DictVectorizer()
dummyX = vec.fit_transform(x_trans).toarray()
x = np.concatenate((x[:,:-2],dummyX[:,:],x[:,-1].reshape(len(x),1)),axis=1)
x = x.astype(float)
X = x[:,:-1]
Y = x[:,-1]
print(x,X,Y)
#训练数据
regr = linear_model.LinearRegression()
regr.fit(X,Y)
print('coefficients(b1,b2...):',regr.coef_)
print('intercept(b0):',regr.intercept_)
```

运行程序，输出如下：

```
[[100.    4.    0.    1.    0.    9.3]
 [ 50.    3.    1.    0.    0.    4.8]
 [100.    4.    0.    1.    0.    8.9]
 [100.    2.    0.    0.    1.    6.5]
 [ 50.    2.    0.    0.    1.    4.2]
 [ 80.    2.    0.    1.    0.    6.2]
 [ 75.    3.    0.    1.    0.    7.4]
 [ 65.    4.    1.    0.    0.    6. ]
 [ 90.    3.    1.    0.    0.    7.6]
```

```
 [100.    4.    0.    1.    0.    9.3]
 [ 50.    3.    1.    0.    0.    4.8]
 [100.    4.    0.    1.    0.    8.9]
 [100.    2.    0.    0.    1.    6.5]] [[100.   4.   0.   1.   0.]
 [ 50.   3.   1.   0.   0.]
 [100.   4.   0.   1.   0.]
 [100.   2.   0.   0.   1.]
 [ 50.   2.   0.   0.   1.]
 [ 80.   2.   0.   1.   0.]
 [ 75.   3.   0.   1.   0.]
 [ 65.   4.   1.   0.   0.]
 [ 90.   3.   1.   0.   0.]
 [100.   4.   0.   1.   0.]
 [ 50.   3.   1.   0.   0.]
 [100.   4.   0.   1.   0.]
 [100.   2.   0.   0.   1.]]
[9.3 4.8 8.9 6.5 4.2 6.2 7.4 6.  7.6 9.3 4.8 8.9 6.5]
coefficients(b1,b2...): [ 0.05452507  0.70930079 -0.18019642  0.60821607 -0.42801964]
intercept(b0): 0.19899589563177766
```

5.3 广义线性模型

考虑单调可导函数 $h(\cdot)$，令 $h(y)=\boldsymbol{w}^{\mathrm{T}}\boldsymbol{x}+b$，这样得到的模型称为广义线性模型(generalized linear model)。

广义线性模型的一个典型的例子就是对数线性回归。当 $h(\cdot)=\ln(\cdot)$时的广义线性模型就是对数线性回归，即

$$\ln y=\boldsymbol{w}^{\mathrm{T}}\boldsymbol{x}+b$$

它是通过 $\exp(\boldsymbol{w}^{\mathrm{T}}\boldsymbol{x}+b)$拟合 y 的。它虽然称为广义线性回归，但实质上是非线性的。

【例 5-3】 本例中使用一个 2 次函数加上随机的扰动来生成 500 个点，然后尝试用 1、2、100 次方的多项式对该数据进行拟合。

拟合的目的是使得根据训练数据能够拟合出一个多项式函数，这个函数能够很好地拟合现有数据，并且能对未知的数据进行预测。

```
import matplotlib.pyplot as plt
import numpy as np
import scipy as sp
from scipy.stats import norm
from sklearn.pipeline import Pipeline
from sklearn.linear_model import LinearRegression
from sklearn.preprocessing import PolynomialFeatures
from sklearn import linear_model
'''''数据生成'''
x = np.arange(0, 1, 0.002)
y = norm.rvs(0, size=500, scale=0.1)
```

```
y = y + x ** 2
''''' 均方误差根 '''
def rmse(y_test, y):
    return sp.sqrt(sp.mean((y_test - y) ** 2))
''''' 与均值相比的优秀程度,介于 0～1。0 表示不如均值,1 表示完美预测。这个版本的实现是参考
scikit - learn 官网文档  '''
def R2(y_test, y_true):
    return 1 - ((y_test - y_true) ** 2).sum() / ((y_true - y_true.mean()) ** 2).sum()
def R22(y_test, y_true):
    y_mean = np.array(y_true)
    y_mean[:] = y_mean.mean()
    return 1 - rmse(y_test, y_true) / rmse(y_mean, y_true)
plt.scatter(x, y, s = 5)
degree = [1,2,100]
y_test = []
y_test = np.array(y_test)
for d in degree:
    clf = Pipeline([('poly', PolynomialFeatures(degree = d)),
                    ('linear', LinearRegression(fit_intercept = False))])
    clf.fit(x[:, np.newaxis], y)
    y_test = clf.predict(x[:, np.newaxis])
    print(clf.named_steps['linear'].coef_)
    print('rmse = %.2f, R2 = %.2f, R22 = %.2f, clf.score = %.2f' %
      (rmse(y_test, y),
       R2(y_test, y),
       R22(y_test, y),
       clf.score(x[:, np.newaxis], y)))
    plt.plot(x, y_test, linewidth = 2)
plt.grid()
plt.legend(['1','2','100'], loc = 'upper left')
plt.show()
```

运行程序,输出如下,效果如图 5-2 所示。

```
[ - 0.17646909   1.01838941]
rmse = 0.12, R2 = 0.86, R22 = 0.63, clf.score = 0.86
[ - 0.02872693   0.1283764    0.8917966 ]
rmse = 0.10, R2 = 0.91, R22 = 0.69, clf.score = 0.91
[ - 1.11632925e - 01    4.29980995e + 01  - 4.76106227e + 03   2.49103305e + 05
  - 7.46452218e + 06    1.41084996e + 08  - 1.78775541e + 09   1.58133296e + 10
  - 1.00114941e + 11    4.58895566e + 11  - 1.51772245e + 12   3.53223913e + 12
…
    4.46276417e + 11  - 5.96923498e + 11  - 1.51906488e + 12  - 1.90374020e + 12
  - 1.63247529e + 12  - 7.23567011e + 11    5.91292805e + 11   1.81499049e + 12
    2.17224445e + 12    1.00237307e + 12  - 1.37732005e + 12  - 3.02020489e + 12
    1.81820159e + 12]
rmse = 0.09, R2 = 0.91, R22 = 0.70, clf.score = 0.91
```

这里要注意几点：

(1) 误差分析。做回归分析,常用的误差主要有均方误差根(RMSE)和 R-平方(R2)。RMSE 是预测值与真实值的误差平方根的均值。这种度量方法很流行(Netflix 机器学习比

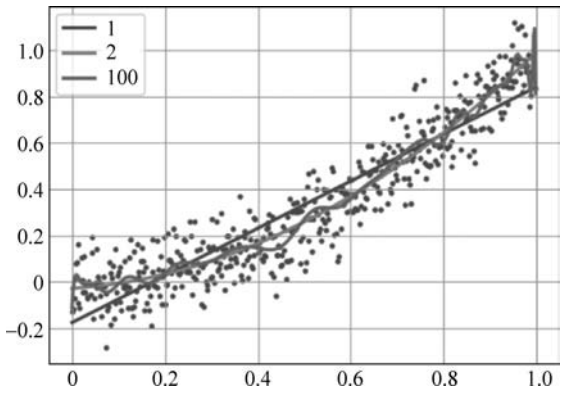


图 5-2　广义线性回归效果 1

赛的评价方法),是一种定量的权衡方法。

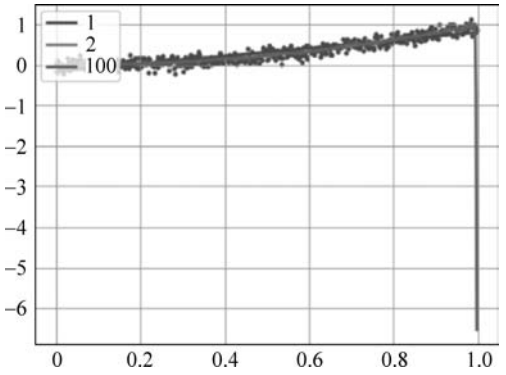
R2 方法是将预测值跟使用均值的情况下相比,看能好多少。R2 通常在(0,1)区间。0 表示还不如什么都不预测,直接取均值的情况,而 1 表示所有预测跟真实结果完美匹配的情况。我们看到多项式次数为 1 的时候,虽然拟合得不太好,但 R2 也能达到 0.82。二次多项式提高到了 0.88。而次数提高到 100 次时,R2 也只提高到了 0.89。

(2) 过拟合。使用 100 次方多项式做拟合,效果确实是好了一些,但该模型的据测能力很差。而且注意看多项式系数,出现了大量的大数值,甚至达到 10 的 12 次方。这里修改代码,将 500 个样本中的最后 2 个从训练集中移除,但在测试中仍然会测试所有 500 个样本。

```
clf.fit(x[:498, np.newaxis], y[:498])
```

这样修改后的多项式拟合结果如下,效果如图 5-3 所示。

```
[-0.16490619  0.97634264]
rmse = 0.13, R2 = 0.83, R22 = 0.59, clf.score = 0.83
[ 0.0046522  -0.04921213  1.03174524]
rmse = 0.10, R2 = 0.90, R22 = 0.68, clf.score = 0.90
……
rmse = 0.35, R2 = -0.31, R22 = -0.15, clf.score = -0.31
```



彩色图片
图 5-3

图 5-3　广义线性回归效果 2

仅仅只是缺少了最后 2 个训练样本，绿线（100 次方多项式拟合结果）的预测发生了剧烈的偏差，R2 也急剧下降到 0.57。而反观一次多项式和二次多项式的拟合结果，R2 反而略微上升了。

这说明高次多项式过度拟合了训练数据，包括其中大量的噪声，导致其完全丧失了对数据趋势的预测能力。前面也看到，100 次多项式拟合出的系数数值无比巨大。人们自然想到通过在拟合过程中限制这些系数数值的大小来避免生成这种畸形的拟合函数。

其基本原理是将拟合多项式的所有系数绝对值之和（L_1 正则化）或者平方和（L_2 正则化）加入惩罚模型中，并指定一个惩罚力度因子 w，来避免产生这种畸形系数。

这样的思想应用在了岭（Ridge）回归（使用 L_2 正则化）、Lasso 法（使用 L_1 正则化）、弹性网（Elastic net，使用 L_1+L_2 正则化）等方法中，都能有效避免过拟合。

下面以岭回归为例看看 100 次多项式的拟合是否有效。将代码修改如下：

```
clf = Pipeline([('poly', PolynomialFeatures(degree=d)),
                ('linear', linear_model.Ridge())])
clf.fit(x[:400, np.newaxis], y[:400])
```

修改程序后，输出如下，效果如图 5-4 所示。

```
[0.          0.76953671]
rmse=0.15, R2=0.78, R22=0.53, clf.score=0.78
[0.          0.29611609 0.62106811]
rmse=0.11, R2=0.88, R22=0.65, clf.score=0.88
…
rmse=0.11, R2=0.88, R22=0.65, clf.score=0.88
```

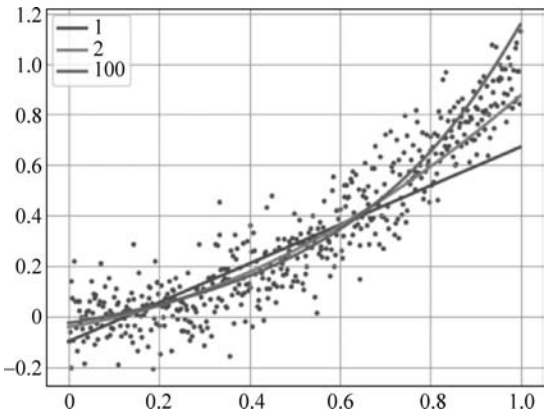


图 5-4　岭回归效果

由图 5-4 可以看到，100 次多项式的系数参数变得很小，大部分都接近于 0。

另外值得注意的是，使用岭回归之类的惩罚模型后，一次多项式或二次多项式回归的 R2 值可能会稍微低于基本线性回归。

这样的模型即使使用 100 次多项式，在训练 400 个样本、预测 500 个样本的情况下不仅

有更小的 R2 误差，还具备优秀的预测能力。

5.4 逻辑回归

线性模型也可以用于分类。考虑二分类问题，给定数据集 $T=\{(\boldsymbol{x}_1,y_1),(\boldsymbol{x}_2,y_2),\cdots,(\boldsymbol{x}_N,y_N)\}$，$\boldsymbol{x}_i\in X\subseteq\mathbf{R}^n$，$y_i\in Y\subseteq\mathbf{R}$，$i=1,2,\cdots,N$，其中 $\boldsymbol{x}_i=(x_i^{(1)},x_i^{(2)},\cdots,x_i^{(n)})^{\mathrm{T}}$。我们需要知道 $P(y/\boldsymbol{x})$，这里用条件概率的原因是：预测的时候都是已知 $\boldsymbol{x}$，然后需要判断此时对应的 y 值。

考虑到 $\boldsymbol{w}\cdot\boldsymbol{x}+b$ 取值是连续的，因此它不能拟合离散变量。可以考虑用它来拟合条件概率 $P(y=1/\boldsymbol{x})$，因为概率的取值也是连续的，但是对于 $\boldsymbol{w}\neq\mathbf{0}$（如果等于零向量，则没有求解的价值），$\boldsymbol{w}\cdot\boldsymbol{x}+b$ 取值是从$-\infty\sim+\infty$，不符合概率取值为 0～1，因此考虑采用广义线性模型，最理想的是单位阶跃函数：

$$P(y=1/\boldsymbol{x})=\begin{cases}0, & z<0\\ 0.5, & z=0\\ 1, & z>0\end{cases},\quad z=\boldsymbol{w}\cdot\boldsymbol{x}+b$$

但是阶跃函数不满足单调可导的性质。退而求其次，我们寻找一个可导的、与阶跃函数相似的函数。对数概率函数(logistic function)就是这样的一个替代函数：

$$P(y=1/\boldsymbol{x})=\frac{1}{1+\mathrm{e}^{-z}},\quad z=\boldsymbol{w}\cdot\boldsymbol{x}+b$$

由于 $P(y=0/\boldsymbol{x})=1-P(y=1/\boldsymbol{x})$，所以有 $\ln\dfrac{P(y=1/\boldsymbol{x})}{P(y=0/\boldsymbol{x})}=z=\boldsymbol{w}\cdot\boldsymbol{x}+b$。$\ln\dfrac{P(y=1/\boldsymbol{x})}{P(y=0/\boldsymbol{x})}$表示样本为正例的可能性与负例的可能性之比，称为概率(odds)，反映了样本作为正例的相对可能性。概率的对数称为对数概率(log odds，也称为 logit)。

下面给出逻辑回归模型参数估计：给定训练数据集 $T=\{(\boldsymbol{x}_1,y_1),(\boldsymbol{x}_2,y_2),\cdots,(\boldsymbol{x}_N,y_N)\}$，$\boldsymbol{x}_i\in X\subseteq\mathbf{R}^n$，$y_i\in\{0,1\}$。模型估计的原理是：用极大似然估计法估计模型参数。

为了便于讨论，将参数 b 吸收进 $\boldsymbol{w}$ 中，令：

$$\tilde{\boldsymbol{w}}=(w^{(1)},w^{(2)},\cdots,w^{(n)},b)^{\mathrm{T}}\in\mathbf{R}^{n+1},\quad\tilde{\boldsymbol{x}}=(x^{(1)},x^{(2)},\cdots,x^{(n)},1)^{\mathrm{T}}\in\mathbf{R}^{n+1}$$

令 $P(y=1/\boldsymbol{x})=\pi(\tilde{\boldsymbol{x}})=\dfrac{\exp(\tilde{\boldsymbol{w}}\cdot\tilde{\boldsymbol{x}})}{1+\exp(\tilde{\boldsymbol{w}}\cdot\tilde{\boldsymbol{x}})}$，$P(y=0/\boldsymbol{x})=1-\pi(\tilde{\boldsymbol{x}})$，则似然函数为：

$$\prod_{i=1}^{N}[\pi(\tilde{\boldsymbol{x}})]^{y_i}[1-\pi(\tilde{\boldsymbol{x}})]^{1-y_i}$$

对数似然函数为：

$$L(\tilde{\boldsymbol{w}})=\sum_{i=1}^{N}[y_i\log\pi(\tilde{\boldsymbol{x}})+(1-y_i)\log(1-\pi(\tilde{\boldsymbol{x}}))]$$

$$=\sum_{i=1}^{N}\left[y_i\log\frac{\pi(\tilde{x})}{1-\pi(\tilde{x})}+\log(1-\pi(\tilde{x}))\right]$$

又由于 $\pi(\tilde{x})=\dfrac{\exp(\tilde{w}\cdot\tilde{x})}{1+\exp(\tilde{w}\cdot\tilde{x})}$，因此：

$$L(\tilde{w})=\sum_{i=1}^{N}[y_i(\tilde{w}\cdot\tilde{x})-\log(1+\exp((\tilde{w}\cdot\tilde{x}))]$$

对 $L(\tilde{w})$求极大值，得到 $\tilde{w}$ 的估计值。设估计值为 $\tilde{w}^*$，则逻辑回归模型为：

$$P(Y=1/X=\tilde{x})=\frac{\exp(\tilde{w}^*\cdot\tilde{x})}{1+\exp(\tilde{w}^*\cdot\tilde{x})}$$

$$P(Y=0/X=\tilde{x})=\frac{1}{1+\exp(\tilde{w}^*\cdot\tilde{x})}$$

提示：通常用梯度下降法或者拟牛顿法来求解该最大值问题。

以上讨论的都是二分类的逻辑回归模型，可以推广到多分类逻辑回归模型。设离散型随机变量 Y 的取值集合为$\{1,2,\cdots,K\}$，则多分类逻辑回归模型为：

$$P(Y=k/\tilde{x})=\frac{\exp(\tilde{w}_k\cdot\tilde{x})}{1+\sum_{k=1}^{K-1}\exp(\tilde{w}_k\cdot\tilde{x})},\quad k=1,2,\cdots,K-1$$

$$P(Y=K/\tilde{x})=\frac{1}{1+\sum_{k=1}^{K-1}\exp(\tilde{w}_k\cdot\tilde{x})},\quad \tilde{x}\in\mathbf{R}^{n+1},\tilde{w}_k\in\mathbf{R}^{n+1}$$

其参数估计方法与二分类逻辑回归模型类似。

【例 5-4】 对给定的数据进行逻辑回归分析。

```
from numpy import *
filename = ('testSet.txt')                          #文件目录
def loadDataSet():                                  #读取数据(这里只有两个特征)
    dataMat = []
    labelMat = []
    fr = open(filename)
    for line in fr.readlines():
        lineArr = line.strip().split()
        dataMat.append([1.0, float(lineArr[0]), float(lineArr[1])])  #代码中的 1.0,表示
方程的常量比如两个特征 X1,X2,共需要三个参数,W1 + W2 * X1 + W3 * X2
        labelMat.append(int(lineArr[2]))
    return dataMat,labelMat
def sigmoid(inX):                                   #sigmoid 函数
    return 1.0/(1 + exp( - inX))
def gradAscent(dataMat, labelMat):                  #梯度上升求最优参数
    dataMatrix = mat(dataMat)                       #将读取的数据转换为矩阵
    classLabels = mat(labelMat).transpose()         #将读取的数据转换为矩阵
    m,n = shape(dataMatrix)
    alpha = 0.001                                   #设置梯度的阈值,该值越大梯度上升幅度越大
```

```
    maxCycles = 500                          #设置迭代的次数,一般看实际数据进行设定,
                                             #有些可能 200 次就够了
    weights = ones((n,1))                    #设置初始的参数,并都赋默认值为 1。注意这里
                                             #权重以矩阵形式表示三个参数
    for k in range(maxCycles):
        h = sigmoid(dataMatrix * weights)
        error = (classLabels - h)            #求导后差值
        weights = weights + alpha * dataMatrix.transpose() * error   #迭代更新权重
    return weights
"""随机梯度上升,当数据量比较大时,每次迭代都选择全量数据进行计算,计算量会非常大。所以采
用每次迭代中一次只选择其中的一行数据进行更新权重。"""
def stocGradAscent0(dataMat, labelMat):
    dataMatrix = mat(dataMat)
    classLabels = labelMat
    m,n = shape(dataMatrix)
    alpha = 0.01
    maxCycles = 500
    weights = ones((n,1))
    for k in range(maxCycles):
        for i in range(m):                   #遍历计算每一行
            h = sigmoid(sum(dataMatrix[i] * weights))
            error = classLabels[i] - h
            weights = weights + alpha * error * dataMatrix[i].transpose()
    return weights
"""改进版随机梯度上升,在每次迭代中随机选择样本来更新权重,并且随迭代次数增加,权重变化
越小。"""
def stocGradAscent1(dataMat, labelMat):
    dataMatrix = mat(dataMat)
    classLabels = labelMat
    m,n = shape(dataMatrix)
    weights = ones((n,1))
    maxCycles = 500
    for j in range(maxCycles):               #迭代
        dataIndex = [i for i in range(m)]
        for i in range(m):                   #随机遍历每一行
            alpha = 4/(1 + j + i) + 0.0001   #随迭代次数增加,权重变化越小
            randIndex = int(random.uniform(0,len(dataIndex)))   #随机采样
            h = sigmoid(sum(dataMatrix[randIndex] * weights))
            error = classLabels[randIndex] - h
            weights = weights + alpha * error * dataMatrix[randIndex].transpose()
            del(dataIndex[randIndex])        #去除已经抽取的样本
    return weights
def plotBestFit(weights):                    #画出最终分类的图
    import matplotlib.pyplot as plt
    dataMat,labelMat = loadDataSet()
    dataArr = array(dataMat)
    n = shape(dataArr)[0]
    xcord1 = []; ycord1 = []
    xcord2 = []; ycord2 = []
    for i in range(n):
```

```
            if int(labelMat[i]) == 1:
                xcord1.append(dataArr[i,1])
                ycord1.append(dataArr[i,2])
            else:
                xcord2.append(dataArr[i,1])
                ycord2.append(dataArr[i,2])
        fig = plt.figure()
        ax = fig.add_subplot(111)
        ax.scatter(xcord1, ycord1, s = 30, c = 'red', marker = 's')
        ax.scatter(xcord2, ycord2, s = 30, c = 'green')
        x = arange( - 3.0, 3.0, 0.1)
        y = ( - weights[0] - weights[1] * x)/weights[2]
        ax.plot(x, y)
        plt.xlabel('X1')
        plt.ylabel('X2')
        plt.show()
    def main():
        dataMat, labelMat = loadDataSet()
        weights = gradAscent(dataMat, labelMat).getA()
        plotBestFit(weights)
    if __name__ == '__main__':
        main()
```

运行程序，效果如图 5-5 所示。

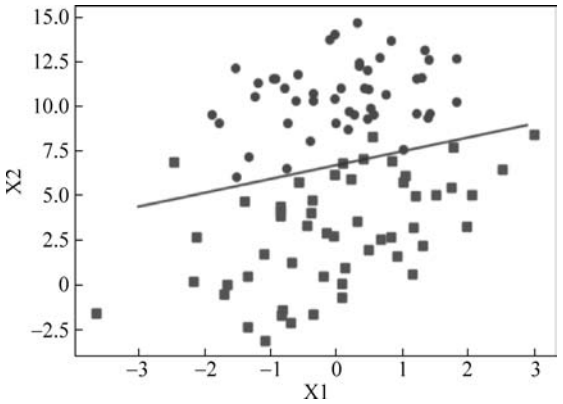


图 5-5　逻辑回归分析

5.5　岭回归

岭回归是在最小二乘法的基础之上的，所以要了解岭回归，首先要了解最小二乘的回归原理。设有多重线性回归模型 $y=\boldsymbol{x}\beta+\varepsilon$，参数 β 的最小二乘估计为：

$$\hat{\beta}=(\boldsymbol{x}^{\mathrm{T}}\boldsymbol{x})^{-1}\boldsymbol{x}^{\mathrm{T}}y$$

当自变量间存在多重共线性，即 $|\boldsymbol{x}^{\mathrm{T}}\boldsymbol{x}|\approx 0$ 时，设想给 $|\boldsymbol{x}^{\mathrm{T}}\boldsymbol{x}|$ 加上一个正常数矩阵（$k>0$），

那么$|\boldsymbol{x}^{\mathrm{T}}\boldsymbol{x}|+k\boldsymbol{I}$接近奇异的程度就会比接近奇异的程度小得多。考虑到变量的量纲问题，要先对数据标准化，标准化捕捉设计矩阵仍用$\boldsymbol{x}$表示，定义称为岭回归估计，其中，k称为岭参数。由于假设x已经标准化，所以就是自变量样本相关阵。y可以标准化也可以未标准化，如果y也经过标准化，那么计算的实际是标准化岭回归估计。k作为β的估计应比最小二乘估计稳定，$k=0$时的岭回归估计就是普通的最小二乘估计。因为岭参数k不是唯一确定的，所以得到的岭回归实际是回归参数的一个估计族。

岭回归的参数估计为：

$$\hat{\beta}(k)=(\boldsymbol{x}^{\mathrm{T}}\boldsymbol{x}+k\boldsymbol{I})^{-1}\boldsymbol{x}^{\mathrm{T}}y$$

【例 5-5】 随机产生100组数据集，每组数据集包含25个点，每个点满足：$y=\sin(2\pi x)+\varepsilon$，这里$x\in\{0.041\times i, i=1,2,\cdots,24\}$，$e$是添加的高斯噪声$(0,0.3^2)$。在每组数据集上用具有不同$\lambda$的7阶多项式进行岭回归拟合。

```
import numpy as np
import matplotlib.pyplot as plt
from tkinter import _flatten
x_arange = 0.041 * np.arange(0, 25, 1)                    #每组数据的 25 个点
y_True = np.sin(2 * np.pi * x_arange)                     #每个数据点对应的值(没有添加噪声)
y_Noise = np.zeros(y_True.shape)                          #添加噪声的值
x_Prec = np.linspace(0, 24 * 0.041, 100)                  #画图范围
mu = 0                                                    #噪声的 mu 值
sigma = 0.3                                               #噪声的 sigma 值
Num = 100                                                 #100 组数据集
n = 8                                                     #7 阶多项式
lamda = [np.exp(1), np.exp(0), np.exp(-5), np.exp(-10)]   #不同的 lambda 值
phi = np.mat(np.zeros((x_arange.size, n)))                #phi 矩阵
x = np.mat(x_arange).T                                    #输入数据矩阵
#phi 矩阵运算
for i_n in range(n):
    for y_n in range(x_arange.size):
        phi[y_n, i_n] = x[y_n, 0] ** i_n
plt.figure(figsize=(15, 10))
index = 221
for i_lamda in lamda:
    plt.subplot(index)
    index += 1
    plt.title("lambda = %f" % i_lamda)
    plt.plot(x_Prec, np.sin(2 * np.pi * x_Prec), color='g')
    for k in range(Num):
        for i in range(x_arange.size):
            y_Noise[i] = y_True[i] + np.random.normal(mu, sigma)
        y = np.mat(y_Noise).T
        #求解 w 参数
        W = (phi.T * phi + i_lamda * np.eye(n)).I * phi.T * y
        ploy = list(_flatten(W.T.tolist()))
        ploy.reverse()
        p = np.poly1d(ploy)
```

```
        if k % 5 == 0:                                    # 只画 20 条曲线
            plt.plot(x_Prec, p(x_Prec), color = 'r')
plt.show()
```

运行程序，效果如图 5-6 所示。

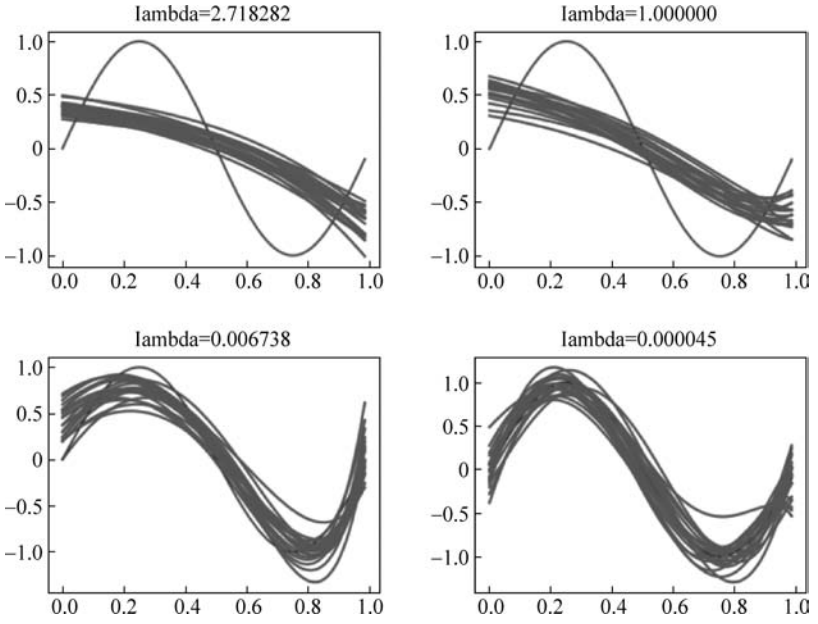


图 5-6　不同的 lambda 值的岭回归效果

5.6　Lasso 回归

Lasso 回归与岭回归非常相似，它们的差别在于使用了不同的正则化项。最终都实现了约束参数从而防止过拟合的效果。Lasso 之所以重要，还有另一个原因，即 Lasso 能够将一些作用比较小的特征的参数训练为 0，从而获得稀疏解。也就是说，用这种方法，在训练模型的过程中实现了降维（特征筛选）的目的。

Lasso 回归的代价函数为：

$$J(\boldsymbol{\theta})=\frac{1}{2m}\sum_{i=1}^{m}(y^{(i)}-(\boldsymbol{w}\boldsymbol{x}^{(i)}+b))^2+\lambda\|\boldsymbol{w}\|_1=\frac{1}{2}\mathrm{MSE}(\boldsymbol{\theta})+\lambda\sum_{i=1}^{m}|\theta_i| \quad (5\text{-}1)$$

式中的 $\boldsymbol{w}$ 是长度为 n 的向量，不包括截距项的系数 θ_0，$\boldsymbol{\theta}$ 是长度为 $n+1$ 的向量，包括截距项的系数 θ_0，m 为样本数，n 为特征数。$\|\boldsymbol{w}\|_1$ 表示参数 $\boldsymbol{w}$ 的 L_1 范数，也是一种表示距离的函数。加入 $\boldsymbol{w}$ 表示三维空间中的一个点 (x,y,z)，那么 $\|\boldsymbol{w}\|_1=|x|+|y|+|z|$，即各个方向上的绝对值（长度）之和。

式(5-1)的梯度为：

$$\nabla_{\boldsymbol{\theta}}\mathrm{MSE}(\boldsymbol{\theta})+\lambda\begin{pmatrix}\mathrm{sigm}(\theta_1)\\ \mathrm{sigm}(\theta_2)\\ \vdots\\ \mathrm{sigm}(\theta_n)\end{pmatrix}$$

其中，$\mathrm{sigm}(\theta_i)$由 θ_i 的符号决定：$\theta_i>0$，$\mathrm{sigm}(\theta_i)=1$；$\theta_i=0$，$\mathrm{sigm}(\theta_i)=0$；$\theta_i<0$，$\mathrm{sigm}(\theta_i)=-1$。

【例 5-6】 利用 Lasso 回归对自带的数据集进行回归分析。

```
import matplotlib.pyplot as plt
import numpy as np
from sklearn import datasets, linear_model, discriminant_analysis, cross_validation
def load_data():
    diabetes = datasets.load_diabetes()
    return cross_validation.train_test_split(diabetes.data, diabetes.target, test_size =
0.25, random_state = 0)
def test_lasso( * data):
    X_train, X_test, y_train, y_test = data
    lassoRegression = linear_model.Lasso()
    lassoRegression.fit(X_train, y_train)
    print("权重向量: % s,b 的值为: % .2f" % (lassoRegression.coef_, lassoRegression.intercept_))
    print("损失函数的值: % .2f" % np.mean((lassoRegression.predict(X_test) - y_test) ** 2))
    print("预测性能得分: % .2f" % lassoRegression.score(X_test, y_test))
#测试不同的 α 值对预测性能的影响
def test_lasso_alpha( * data):
    X_train, X_test, y_train, y_test = data
    alphas = [0.01, 0.02, 0.05, 0.1, 0.2, 0.5, 1, 2, 5, 10, 20, 50, 100, 200, 500, 1000]
    scores = []
    for i, alpha in enumerate(alphas):
        lassoRegression = linear_model.Lasso(alpha = alpha)
        lassoRegression.fit(X_train, y_train)
        scores.append(lassoRegression.score(X_test, y_test))
    return alphas, scores
def show_plot(alphas, scores):
    figure = plt.figure()
    ax = figure.add_subplot(1, 1, 1)
    ax.plot(alphas, scores)
    ax.set_xlabel(r" $ \alpha $ ")
    ax.set_ylabel(r"score")
    ax.set_xscale("log")
    ax.set_title("lasso")
    plt.show()
if __name__ == '__main__':
    X_train, X_test, y_train, y_test = load_data()
    #使用默认的 alpha
    #test_lasso(X_train, X_test, y_train, y_test)
    #使用自己设置的 alpha
    alphas, scores = test_lasso_alpha(X_train, X_test, y_train, y_test)
```

```
show_plot(alphas, scores)
```

运行程序，效果如图 5-7 所示。

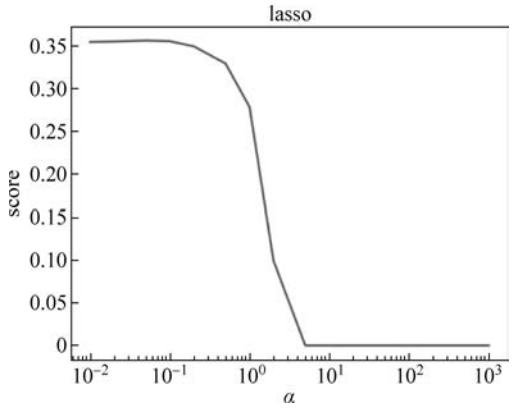


图 5-7　lasso 回归

5.7　弹性网络

弹性网络(Elastic Net)结合了岭回归和 Lasso 回归，由两者加权平均所得。据介绍，这种方法在特征数大于训练集样本数或有些特征之间高度相关时比 Lasso 更加稳定。其代价函数为：

$$J(\theta)=\frac{1}{2}\mathrm{MSE}(\theta)+r\lambda\sum_{i=1}^{m}|\theta_i|+\frac{1-r}{2}\lambda\sum_{i=1}^{n}\theta_i^2$$

其中，r 表示 L_1 所占的比例。

【例 5-7】 用 Python 实现弹性网络算法(多变量)。使用鸢尾花数据集，后 3 个特征作为特征，用来预测第一个特征。

```
#导入必要的编程库,创建计算图,加载数据集
import matplotlib.pyplot as plt
import tensorflow as tf
import numpy as np
from sklearn import datasets
from tensorflow.python.framework import ops
ops.get_default_graph()
sess = tf.Session()
iris = datasets.load_iris()
x_vals = np.array([[x[1], x[2], x[3]] for x in iris.data])
y_vals = np.array([y[0] for y in iris.data])
#声明学习率,批量大小,占位符和模型变量,模型输出
learning_rate = 0.001
batch_size = 50
x_data = tf.placeholder(shape=[None, 3], dtype=tf.float32)   #占位符大小为 3
```

```
y_target = tf.placeholder(shape = [None, 1], dtype = tf.float32)
A = tf.Variable(tf.random_normal(shape = [3,1]))
b = tf.Variable(tf.random_normal(shape = [1,1]))
model_output = tf.add(tf.matmul(x_data, A), b)
#对于弹性网络回归算法,损失函数包括 L1 正则和 L2 正则
elastic_param1 = tf.constant(1.)
elastic_param2 = tf.constant(1.)
l1_a_loss = tf.reduce_mean(abs(A))
l2_a_loss = tf.reduce_mean(tf.square(A))
e1_term = tf.multiply(elastic_param1, l1_a_loss)
e2_term = tf.multiply(elastic_param2, l2_a_loss)
loss = tf.expand_dims(tf.add(tf.add(tf.reduce_mean(tf.square(y_target - model_output)),
e1_term), e2_term), 0)
#初始化变量,声明优化器,然后遍历迭代运行,训练拟合得到参数
init = tf.global_variables_initializer()
sess.run(init)
my_opt = tf.train.GradientDescentOptimizer(learning_rate)
train_step = my_opt.minimize(loss)
loss_vec = []
for i in range(1000):
   rand_index = np.random.choice(len(x_vals), size = batch_size)
   rand_x = x_vals[rand_index]
   rand_y = np.transpose([y_vals[rand_index]])
   sess.run(train_step, feed_dict = {x_data:rand_x, y_target:rand_y})
   temp_loss = sess.run(loss, feed_dict = {x_data:rand_x, y_target:rand_y})
   loss_vec.append(temp_loss)
   if (i + 1) % 250 == 0:
     print('Step#' + str(i + 1) + 'A = ' + str(sess.run(A)) + 'b = ' + str(sess.run(b)))
     print('Loss = ' + str(temp_loss))
#现在能观察到,随着训练迭代后损失函数已收敛
plt.plot(loss_vec, 'k-- ')
plt.title('Loss per Generation')
plt.xlabel('Generation')
plt.ylabel('Loss')
plt.show()
```

运行程序,输出如下,效果如图 5-8 所示。

```
Step#250A = [[1.3357686 ]
 [0.7352283 ]
 [0.12850094]]b = [[ - 1.4265894]]
Loss = [1.9901819]
Step#500A = [[1.3833795 ]
 [0.6989296 ]
 [0.03573647]]b = [[ - 1.2994616]]
Loss = [1.9257231]
Step#750A = [[ 1.3752384e + 00]
 [ 6.9431901e - 01]
 [ - 3.4378259e - 05]]b = [[ - 1.1875116]]
Loss = [1.7963648]
```

```
Step#1000A = [[1.3545375e+00]
 [6.8431729e-01]
 [3.3413176e-05]]b=[[-1.0804418]]
Loss = [1.7084534]
```

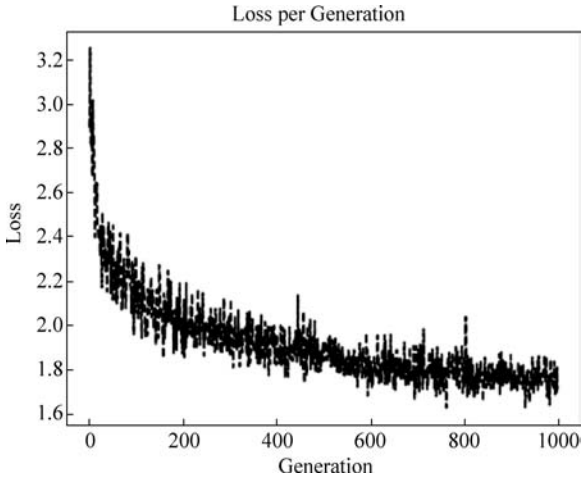


图 5-8　弹性网络算法

5.8　线性判别分析

线性判别分析(Linear Discriminant Analysis,LDA)的思想是：

- 训练时,设法将训练样本投影到一条直线上,使得同类样本的投影点尽可能地接近,异类样本的投影点尽可能地远离。要学习的就是这样的一条直线。
- 预测时,将待预测样本投影到学到的直线上,根据它的投影点的位置来判定它的类别。

5.8.1　线性判别二分类情况

回顾之前的逻辑回归方法,给定 m 个 n 维特征的训练样例,每个 $\boldsymbol{x}^{(i)}$,$i=1,2,\cdots,m$ 对应一个类标签 $y^{(i)}$。我们就是要学习出参数$\boldsymbol{\theta}$,使得 $y^{(i)}=g(\boldsymbol{\theta})^{\mathrm{T}}\boldsymbol{x}^{(i)}$。

现在只考虑二分类情况,也就是 $y=1$ 或 $y=0$。为了方便表示,先换符号重新定义问题,给定特征为 d 维的 N 个样例,$\boldsymbol{x}^{(i)}=(x_1^{(i)},x_2^{(i)},\cdots,x_1^{(i)})$,其中有 N_1 个样例属于类别 w_1,另外 N_2 个样例属于类别 w_2。

现在我们觉得原始特征数太多,想将 d 维特征降到只有一维,又要保证类别能够“清晰”地反映在低维数据上,也就是说,这一维就能决定每个样例的类别。我们将这个最佳的向量称为 $\boldsymbol{w}$(d 维),那么样例 $\boldsymbol{x}$(d 维)到 $\boldsymbol{w}$ 上的投影可以用下式来计算：

$$\boldsymbol{y}=\boldsymbol{w}^{\mathrm{T}}\boldsymbol{x}$$

这里得到的 y 值不是 0/1 值，而是 $\boldsymbol{x}$ 投影到直线上的点到原点的距离。若 $\boldsymbol{x}$ 是二维的，则要找一条直线（方向为 $\boldsymbol{w}$）来做投影，然后寻找最能使样本点分离的直线，如图 5-9 所示。

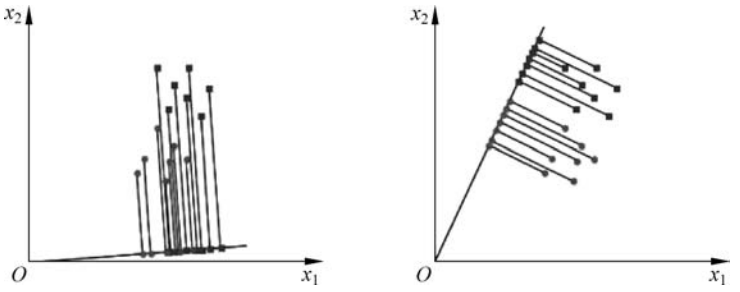


图 5-9　直线投影效果

从直观上看，右图比较好，可以很好地将不同类别的样本点分离。接着从定量的角度来找到这个最佳的 $\boldsymbol{w}$。

首先寻找每类样例的均值（中心点），这里 i 只有两个：

$$\mu_i = \frac{1}{N_i} \sum_{\boldsymbol{x} \in w_i} \boldsymbol{x}$$

由于 $\boldsymbol{x}$ 到 $\boldsymbol{w}$ 投影后的样本点均值为：

$$\tilde{\mu}_i = \frac{1}{N_i} \sum_{y \in w_i} y = \frac{1}{N_i} \sum_{\boldsymbol{x} \in w_i} \boldsymbol{w}^{\mathrm{T}} \boldsymbol{x} = \boldsymbol{w}^{\mathrm{T}} \mu_i$$

由此可知，投影后的均值也就是样本中心点的投影。那什么是最佳直线（$\boldsymbol{w}$）呢？首先发现，能够使投影后的两类样本中心点尽量分离的直线是好的直线，定量表示就是：

$$J(\boldsymbol{w}) = |\tilde{\mu}_1 - \tilde{\mu}_2| = |\boldsymbol{w}^{\mathrm{T}}(\mu_1 - \mu_2)|$$

$J(\boldsymbol{w})$ 越大越好，但只考虑 $J(\boldsymbol{w})$ 是不行的，观察图 5-10。

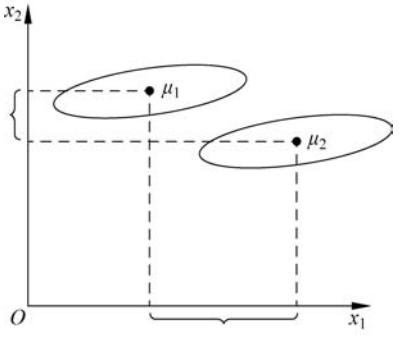


图 5-10　样本点分布图

样本点均匀分布在椭圆里，投影到横轴 x_1 上时能够获得更大的中心点间距 $J(\boldsymbol{w})$，但是由于有重叠，x_1 不能分离样本点。投影到纵轴 x_2 上，虽然 $J(\boldsymbol{w})$ 较小，但是能够分离样本点。因此还需要考虑样本点之间的方差，方差越大，样本点越难以分离。

我们使用另外一个度量值，称作散列值（scatter），对投影后的类求散列值，如下：

$$\tilde{s}_i^2 = \sum_{y \in w_i} (y - \tilde{\mu}_i)^2$$

从上式可以看出，只是少除以样本数量的方差值，散列值的几何意义是样本点的密集程度，值越大，越分散；反之，越集中。而我们想要的投影后的样本点是：不同类别的样本点

越分开越好，同类的越聚集越好，也就是均值差越大越好，散列值越小越好。因此，可以使用 $J(\boldsymbol{w})$ 和 s 来度量，最终的度量公式为：

$$J(\boldsymbol{w})=\frac{|\tilde{\mu}_1-\tilde{\mu}_2|^2}{\tilde{s}_1^2+\tilde{s}_2^2}$$

因此，只需寻找使 $J(\boldsymbol{w})$ 最大的 $\boldsymbol{w}$ 即可。

先把散列值公式展开，

$$\tilde{s}_i^2=\sum_{y\in w_i}(y-\tilde{\mu}_i)^2=\sum_{\boldsymbol{x}\in w_i}(\boldsymbol{w}^{\mathrm{T}}\boldsymbol{x}-\boldsymbol{w}^{\mathrm{T}}\mu_i)^2=\sum_{\boldsymbol{x}\in w_i}\boldsymbol{w}^{\mathrm{T}}(\boldsymbol{x}-\mu_i)(\boldsymbol{x}-\mu_i)^{\mathrm{T}}\boldsymbol{w} \tag{5-2}$$

定义上式的中间部分：

$$s_i=\sum_{x\in w_i}(\boldsymbol{x}-\mu_i)(\boldsymbol{x}-\mu_i)^{\mathrm{T}} \tag{5-3}$$

这个公式就是样例数的协方差矩阵，称为散列矩阵(scatter matrix)。继续定义：

$$s_w=s_1+s_2$$

称 $\boldsymbol{s}_w$ 为类内散度矩阵。回到式(5-2)中，使用 s_i 替换中间部分，得

$$\tilde{s}_i^2=\boldsymbol{w}^{\mathrm{T}}s_i\boldsymbol{w}$$

$$\tilde{s}_1^2+\tilde{s}_2^2=\boldsymbol{w}^{\mathrm{T}}s_w\boldsymbol{w}$$

然后，展开分子

$$(\tilde{\mu}_1-\tilde{\mu}_2)^2=(\boldsymbol{w}^{\mathrm{T}}\mu_1-\boldsymbol{w}^{\mathrm{T}}\mu_2)^2=\boldsymbol{w}^{\mathrm{T}}(\mu_1-\mu_2)(\mu_1-\mu_2)^{\mathrm{T}}\boldsymbol{w}=\boldsymbol{w}^{\mathrm{T}}\boldsymbol{s}_{\mathrm{B}}\boldsymbol{w}$$

其中 $\boldsymbol{s}_{\mathrm{B}}=(\mu_1-\mu_2)(\mu_1-\mu_2)^{\mathrm{T}}$ 称为类间散度矩阵，是两个向量的外积，为一个矩阵，秩为1。那么 $J(\boldsymbol{w})$ 最终可以表示为

$$J(\boldsymbol{w})=\frac{\boldsymbol{w}^{\mathrm{T}}\boldsymbol{s}_{\mathrm{B}}\boldsymbol{w}}{\boldsymbol{w}^{\mathrm{T}}\boldsymbol{s}_w\boldsymbol{w}}$$

在求导前，需要对分母进行归一化，因为不做归一的话，无论 $\boldsymbol{w}$ 扩大多少倍，上式都成立，也就无法确定 w。所以令 $\|\boldsymbol{w}^{\mathrm{T}}\boldsymbol{s}_w\boldsymbol{w}\|=1$，加入拉格朗日乘子后，求导：

$$c(w)=\boldsymbol{w}^{\mathrm{T}}\boldsymbol{s}_{\mathrm{B}}\boldsymbol{w}-\lambda(\boldsymbol{w}^{\mathrm{T}}\boldsymbol{s}_w\boldsymbol{w}-1)$$

$$\Rightarrow\frac{\mathrm{d}c}{\mathrm{d}w}=2\boldsymbol{s}_{\mathrm{B}}\boldsymbol{w}-2\lambda\boldsymbol{s}_w\boldsymbol{w}=0$$

$$\Rightarrow\boldsymbol{s}_{\mathrm{B}}\boldsymbol{w}=\lambda\boldsymbol{s}_w\boldsymbol{w}$$

用到了矩阵微积分，求导时可以简单地把 $\boldsymbol{w}^{\mathrm{T}}\boldsymbol{s}_w\boldsymbol{w}$ 当作 $\boldsymbol{s}_w\boldsymbol{w}^2$ 看待。如果 $\boldsymbol{s}_w$ 可逆，那么将求导后的结果两边都乘以 $\boldsymbol{s}_w^{-1}$，得

$$\boldsymbol{s}_w^{-1}\boldsymbol{s}_{\mathrm{B}}\boldsymbol{w}=\lambda\boldsymbol{w} \tag{5-4}$$

即 $\boldsymbol{w}$ 就是矩阵 $\boldsymbol{s}_w^{-1}\boldsymbol{s}_{\mathrm{B}}$ 的特征向量了。这个公式称为费希尔判别准则。由 $\boldsymbol{s}_{\mathrm{B}}=(\mu_1-\mu_2)(\mu_1-\mu_2)^{\mathrm{T}}$，有 $\boldsymbol{s}_{\mathrm{B}}\boldsymbol{w}=(\mu_1-\mu_2)(\mu_1-\mu_2)^{\mathrm{T}}w=(\mu_1-\mu_2)\lambda_w$，将其代入公式(5-4)中得

$$\boldsymbol{s}_w^{-1}\boldsymbol{s}_{\mathrm{B}}\boldsymbol{w}=\boldsymbol{s}_w^{-1}(\mu_1-\mu_2)\lambda_w=\lambda\boldsymbol{w}$$

由于对 $\boldsymbol{w}$ 扩大缩小任何倍不影响结果，因此可以约去两边的未知常数 λ 和 λ_w，得到：

$$\boldsymbol{w}=\boldsymbol{s}_w^{-1}(\mu_1-\mu_2)$$

至此，我们只需要求出原始样本的均值和方差就可以求出最佳的方向 $\boldsymbol{w}$，这就是 Fisher 于 1936 年提出的线性判别分析。二维样本的投影结果如图 5-11 所示。

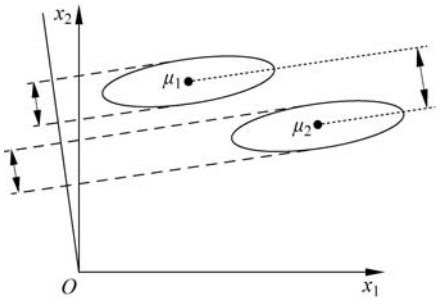


图 5-11　二维样本的投影结果

5.8.2　线性判别多类情况

前面是针对只有两个类的情况，假设类别变成多个了，那么要怎么改变，才能保证投影后类别能够分离呢？之前讨论的是如何将 d 维降到一维，现在类别多了，一维可能已经不能满足要求。假设有 C 个类别，需要 K 维向量（或者叫作基向量）来做投影。

将这 K 维向量表示为 $\boldsymbol{W}=(w_1,w_2,w_K)$。将样本点在这 K 维向量投影后结果表示为 $(y_1,y_2,\cdots,y_K)$，有以下公式成立：

$$y_i=\boldsymbol{w}_i^{\mathrm{T}}\boldsymbol{x}$$

$$y=\boldsymbol{w}^{\mathrm{T}}\boldsymbol{x}$$

仍然从类间散列度和类内散列度来考虑。当样本是二维时，从几何意义上考虑，如图 5-12 所示。

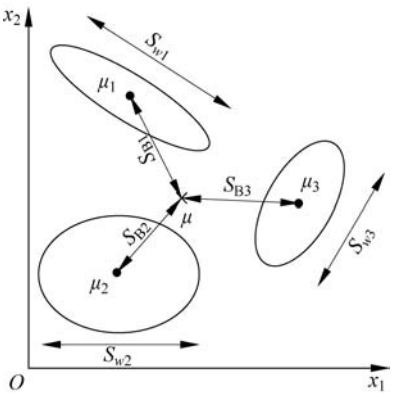


图 5-12　类间散列度和类内散列度几何图

其中，μ_i 和 $\boldsymbol{s}_w$ 与 5.8.1 节的含义一样，$\boldsymbol{s}_{w_1}$ 是类别 1 里的样本点相对于该类中心点 μ_1 的散列度。$\boldsymbol{s}_{\mathrm{B}_1}$ 变成类别 1 中心点相对于样本点心点 $\boldsymbol{\mu}$ 的协方差矩阵，即类 1 相对于 $\boldsymbol{\mu}$ 的散列程度。$\boldsymbol{s}_w$ 为：

$$\boldsymbol{s}_w=\sum_{i=1}^{c}\boldsymbol{s}_{w_i}$$

$\boldsymbol{s}_{w_i}$ 的计算公式不变，仍然类似于类内部样本点的协方差矩阵：

$$\boldsymbol{s}_{w_i}=\sum_{\boldsymbol{x}\in w_i}(\boldsymbol{x}-\mu_i)(\boldsymbol{x}-\mu_i)^{\mathrm{T}}$$

$\boldsymbol{s}_{\mathrm{B}}$ 需要改变，原来度量的是两个均值点的散列情况，现在度量的是每类的均值点相对于样本中心的散列情况。类似于将 μ_i 看作样本点，$\boldsymbol{\mu}$ 是均值的协方差矩阵，如果某类里面的样

本点较多，那么其权重稍大，权重用$\frac{N_i}{N}$表示，但由于$J(\boldsymbol{w})$对倍数不敏感，因此使用N_i。

$$\boldsymbol{s}_{\mathrm{B}}=\sum_{i=1}^{c}N_i(\mu_i-\boldsymbol{\mu})(\mu_i-\boldsymbol{\mu})^{\mathrm{T}}$$

其中，$\boldsymbol{\mu}=\frac{1}{N}\sum_{\forall \boldsymbol{x}}\boldsymbol{x}=\frac{1}{N}\sum_{\boldsymbol{x}\in w_i}N_i\mu_i$是所有样本的均值。

上面讨论的都是在投影前的公式变化，但真正的$J(\boldsymbol{w})$的分子分母都是在投影后计算的。下面看一下样本点投影后的公式改变：

这两个是第i类样本点在某基向量上投影后的均值计算公式：

$$\tilde{\mu}_i=\frac{1}{N_i}\sum_{\boldsymbol{y}\in w_i}\boldsymbol{y}$$

$$\tilde{\boldsymbol{\mu}}=\frac{1}{N}\sum_{\forall y}\boldsymbol{y}$$

下面两个是在某基向量上投影后的$\boldsymbol{s}_w$和$\boldsymbol{s}_{\mathrm{B}}$：

$$\tilde{\boldsymbol{s}}_w=\sum_{i=1}^{c}\sum_{\boldsymbol{y}\in w_i}(\boldsymbol{y}-\tilde{\mu}_i)(\boldsymbol{y}-\tilde{\mu}_i)^{\mathrm{T}}$$

$$\tilde{\boldsymbol{s}}_{\mathrm{B}}=\sum_{i=1}^{c}N_i(\tilde{\mu}_i-\tilde{\boldsymbol{\mu}})(\tilde{\mu}_i-\tilde{\boldsymbol{\mu}})^{\mathrm{T}}$$

其实就是将$\boldsymbol{\mu}$换成$\tilde{\boldsymbol{\mu}}$。

综合各个投影向量$(\boldsymbol{w})$上的$\tilde{\boldsymbol{s}}_w$和$\tilde{\boldsymbol{s}}_{\mathrm{B}}$，更新这两个参数，得到：

$$\tilde{\boldsymbol{s}}_w=\boldsymbol{w}^{\mathrm{T}}\boldsymbol{s}_w\boldsymbol{w}$$

$$\tilde{\boldsymbol{s}}_{\mathrm{B}}=\boldsymbol{w}^{\mathrm{T}}\boldsymbol{s}_{\mathrm{B}}\boldsymbol{w}$$

$\boldsymbol{w}$是基向量矩阵，$\tilde{\boldsymbol{s}}_w$是投影后的各个类内部的散列矩阵之和，$\tilde{\boldsymbol{s}}_{\mathrm{B}}$是投影后各个类中心相对于全样本中心投影的散列矩阵之和。

在5.8.1节的公式$J(\boldsymbol{w})$，分子是两类中心距，分母是每个类自己的散列度。现在投影方向是多维了，分子需要做一些改变，我们不是求两样本中心距之和，而是求每类中心相对于全样本中心的散列度之和。然而，最后的$J(\boldsymbol{w})$形式为：

$$J(\boldsymbol{w})=\frac{|\tilde{\boldsymbol{s}}_{\mathrm{B}}|}{|\tilde{\boldsymbol{s}}_w|}=\frac{|\boldsymbol{w}^{\mathrm{T}}\boldsymbol{s}_{\mathrm{B}}\boldsymbol{w}|}{\boldsymbol{w}^{\mathrm{T}}\boldsymbol{s}_w\boldsymbol{w}}$$

由于得到的分子分母都是散列矩阵，要将矩阵变成实数，需要取行列式。又因为行列式的值实际上是矩阵特征值的积，一个特征值可以表示在该特征向量上的发散程度。因此使用行列式来计算。整个问题又回归求$J(\boldsymbol{w})$的最大值了，固定分母为1，然后求导，得出最后结果：

$$\boldsymbol{s}_{\mathrm{B}}\boldsymbol{w}_i=\lambda\boldsymbol{s}_w w_i$$

与5.8.1节得出的结论一样：$\boldsymbol{s}_w^{-1}\boldsymbol{s}_{\mathrm{B}}w_i=\lambda w_i$。因此还是归结到求矩阵的特征值，首先

求出 $\boldsymbol{s}_w^{-1}\boldsymbol{s}_B$ 的特征值，然后取前 K 个特征向量组成 $\boldsymbol{w}$ 矩阵即可。

5.8.3 线性判别分析实现

5.8.1 节和 5.8.2 节分别介绍了线性判别分析对二分类、多分类过程进行了介绍，下面直接通过例子来演示线性判别分析的实现。

【例 5-8】 演示线性判别分析对随机数据的判别。

```
'''
LDA 算法实现
'''
import os
import sys
import numpy as np
from numpy import *
import operator
import matplotlib
import matplotlib.pyplot as plt
def createDataSet():
 group1 = mat(random.random((2,4)) + 10)
 group2 = mat(random.random((2,4)) + 2)
 return group1, group2
#计算样本均值
#参数 samples 为 n×m 维矩阵,其中 n 表示维数,m 表示样本个数
def compute_mean(samples):
 mean_mat = mean(samples, axis = 1)
 return mean_mat
#计算样本类内离散度
"""参数 samples 表示样本向量矩阵,大小为 nxm,其中 n 表示维数,m 表示样本个数
参数 mean 表示均值向量,大小为 1xd,d 表示维数,大小与样本维数相同,即 d = m"""
def compute_withinclass_scatter(samples, mean):
 #获采样本维数,样本个数
 dimens,nums = samples.shape[:2]
 #将所有样本向量减去均值向量
 samples_mean = samples - mean
 #初始化类内离散度矩阵
 s_in = 0
 for i in range(nums):
        x = samples_mean[:,i]
        s_in += dot(x,x.T)
 return s_in
if __name__ == '__main__':
 group1,group2 = createDataSet()
 print ("group1 :\n",group1)
 print ("group2 :\n",group2)
 mean1 = compute_mean(group1)
 print ("mean1 :\n",mean1)
 mean2 = compute_mean(group2)
 print ("mean2 :\n",mean2)
```

```
 s_in1 = compute_withinclass_scatter(group1, mean1)
 print ("s_in1 :\n",s_in1)
 s_in2 = compute_withinclass_scatter(group2, mean2)
 print ("s_in2 :\n",s_in2)
 #求总类内离散度矩阵
 s = s_in1 + s_in2
 print( "s :\n",s)
 #求 s 的逆矩阵
 s_t = s.I
 print ("s_t :\n",s_t)
 #求解权向量
 w = dot(s_t, mean1 - mean2)
 print ("w :\n",w)
 #判断(2,3)是在哪一类
 test1 = mat([1,1])
 g = dot(w.T, test1.T - 0.5 * (mean1 - mean2))
 print("g(x) :",g)
 #判断(4,5)是在哪一类
 test2 = mat([10,10])
 g = dot(w.T, test2.T - 0.5 * (mean1 - mean2))
 print("g(x) :",g)
```

运行程序，输出如下：

```
 [2.81742415 2.64585055 2.16746183 2.80708945]]
mean1 :
 [[10.54752722]
 [10.58633559]]
mean2 :
 [[2.70746802]
 [2.6094565 ]]
s_in1 :
 [[ 0.70944616 - 0.24457606]
 [ - 0.24457606  0.10698882]]
s_in2 :
 [[0.11233544 0.03508707]
 [0.03508707 0.27899314]]
s :
 [[ 0.8217816   - 0.20948899]
 [ - 0.20948899  0.38598196]]
s_t :
 [[1.41226396 0.76649631]
 [0.76649631 3.00680512]]
w :
 [[17.18648137]
 [29.99429734]]
g(x) : [[ - 139.82117865]]
g(x) : [[284.80582973]]
```

5.9 习题

1. 线性模型是一类统计模型的总称，它包括了_____、_____、_____和_____等。

2. 特征归一化有两个好处，是什么？

3. 拟合时，需要注意几点？

4. Lasso重要的原因有哪些？

5. 创建一组随机三维数据，绘制方程 $z=ax+by+c$ 的空间平面。