机器学习实验报告

线性模型

朱天泽

（日期：2022 年 3 月 21 日）

摘要 在《机器学习》第三章中，我学习了诸多线性模型。在此次实验中，我尝试用 Python 语言实现了对率回归、线性判别分析（LDA）等线性模型，并给出了其在西瓜数据集 3.0α 上的结果。此外，我尝试使用了机器学习库 scikit-learn，同自己实现的算法进行了比较。

关键词 对率回归；LDA；分类；scikit-learn；

1 习题 3.3

1.1 编程题目理解

题目要求编程实现对数几率回归，并给出西瓜数据集 3.0α 上的结果。

在西瓜数据集 3.0α 中，西瓜包含“密度”和“含糖率”两个参数，因此可将西瓜看做横坐标为“密度”纵坐标为“含糖率”的坐标系中的点，记为 $\boldsymbol{x} = (x_1, x_2)$。我们使用该数据，找到合适的 $(\boldsymbol{w}, b)$，利用对数几率函数 $\frac{1}{1+e^{-z}}$ 对西瓜进行二分类（好瓜、坏瓜）。

1.2 对数几率回归原理阐述

已知单调可微函数 $g(\cdot)$，令 $y = g^{-1}(\boldsymbol{w}^{\mathrm{T}}\boldsymbol{x} + b)$，这样得到的模型称为“广义线性模型”。

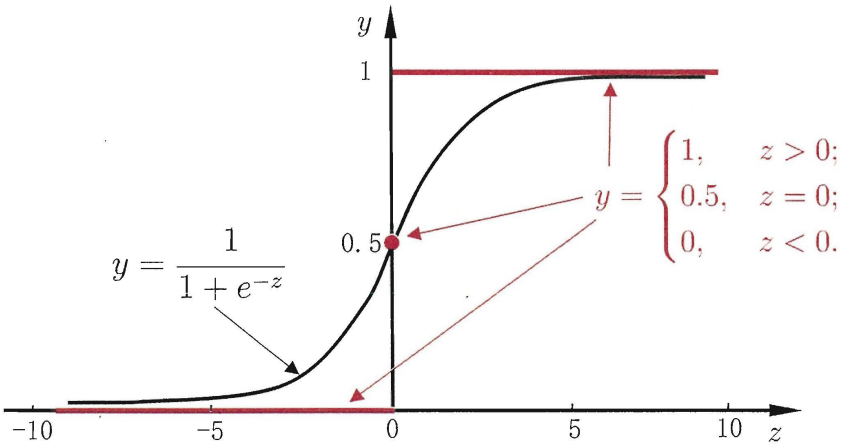

图 1: 单位跃阶函数与对数几率函数

考虑二分类任务，其输出标记 $y \in \{0,1\}$，而线性回归模型阐述的预测值 $z = \boldsymbol{w}^{\mathrm{T}}\boldsymbol{x} + b$；将 z 映射为 0/1 值，最理想的是“单位越阶函数” $y = \begin{cases} 0 & z < 0 \\ 0.5 & z = 1 \\ 1 & z > 1 \end{cases}$。可以看到单位跃阶函数并不连续，不能作为广义线性模型中的 $g^{-1}(\cdot)$，因此我们用对数几率函数 $y = \frac{1}{1+e^{-z}}$ 来近似单位跃阶函数，如图1。

从图1可以看出，对数几率函数将预测值 z 转化为一个接近 0 或 1 的 y 值，并且其输出值在 $z = 0$ 附近变化剧烈。将 z 代入对数几率函数，得到

$$y = \frac{1}{1 + e^{-(\boldsymbol{w}^{\mathrm{T}}\boldsymbol{x}+b)}} \tag{1}$$

式(1)有很好的性质，即

$$\begin{cases} y = \frac{1}{1+e^{-(\boldsymbol{w}^{\mathrm{T}}\boldsymbol{x}+b)}} \\ 1 - y = \frac{e^{-(\boldsymbol{w}^{\mathrm{T}}\boldsymbol{x}+b)}}{1+e^{-(\boldsymbol{w}^{\mathrm{T}}\boldsymbol{x}+b)}} \end{cases} \tag{2}$$

式(2)可进一步等价为

$$\ln \frac{y}{1-y} = \boldsymbol{w}^{\mathrm{T}}\boldsymbol{x} + b \tag{3}$$

若将 y 视为样本 $\boldsymbol{x}$ 作为正例的可能性，则 $1-y$ 是其为反例的可能性，两者的比值 $\frac{y}{1-y}$ 称为“几率”，那么式(3)就称为“对数几率”。

我们将式(1)中的 y 视为类后验概率估计 $p(y=1|\boldsymbol{x})$，将式(1)分子分母同乘 $e^{\boldsymbol{w}^{\mathrm{T}}\boldsymbol{x}+b}$，显然有

$$\begin{aligned} p(y=1|\boldsymbol{x}) &= \frac{e^{\boldsymbol{w}^{\mathrm{T}}\boldsymbol{x}+b}}{1 + e^{\boldsymbol{w}^{\mathrm{T}}\boldsymbol{x}+b}} \\ p(y=0|\boldsymbol{x}) &= \frac{1}{1 + e^{\boldsymbol{w}^{\mathrm{T}}\boldsymbol{x}+b}} \end{aligned} \tag{4}$$

于是，我们可以通过“极大似然法”来估计 $\boldsymbol{w}$ 和 b。给定数据集 $\{\boldsymbol{x}_i, y_i\}_{i=1}^m$，对数几率回归模型最大化“对数似然”

$$\ell(\boldsymbol{w}, b) = \sum_{i=1}^{m} \ln p(y_i|\boldsymbol{x}_i; \boldsymbol{w}, b) \tag{5}$$

即令每个样本属于真实标记的概率越大越好。为方便讨论，令 $\boldsymbol{\beta} = (\boldsymbol{w}; b)$，$\hat{\boldsymbol{x}} = (\boldsymbol{x}; 1)$；再令 $p_{y^*}(\hat{\boldsymbol{x}}; \boldsymbol{\beta}) = p(y = y^*|\hat{\boldsymbol{x}}; \boldsymbol{\beta})$，则式(5)中的似然项可改写为

$$p(y_i|\boldsymbol{x}_i; \boldsymbol{w}, b) = (p_1(\hat{\boldsymbol{x}}_i; \boldsymbol{\beta})^{y_i})(p_0(\hat{\boldsymbol{x}}_i; \boldsymbol{\beta}))^{1-y_i} \tag{6}$$

将式(6)代回式(5)，得

$$
\begin{aligned}
\ell(\boldsymbol{\beta}) &= \sum_{i=1}^{m} \ln[(p_1(\hat{\boldsymbol{x}}_i;\boldsymbol{\beta})^{y_i})(p_0(\hat{\boldsymbol{x}}_i;\boldsymbol{\beta}))^{1-y_i}] \\
&= \sum_{i=1}^{m} [y_i \ln(p_1(\hat{\boldsymbol{x}}_i;\boldsymbol{\beta})) + (1-y_i)\ln(p_0(\hat{\boldsymbol{x}}_i;\boldsymbol{\beta}))] \\
&= \sum_{i=1}^{m} \{y_i[\ln(p_1(\hat{\boldsymbol{x}}_i;\boldsymbol{\beta})) - \ln(p_0(\hat{\boldsymbol{x}}_i;\boldsymbol{\beta}))] + \ln(p_0(\hat{\boldsymbol{x}}_i;\boldsymbol{\beta}))\} \\
&= \sum_{i=1}^{m} \left[y_i \ln\left(\frac{p_1(\hat{\boldsymbol{x}}_i;\boldsymbol{\beta})}{p_0(\hat{\boldsymbol{x}}_i;\boldsymbol{\beta})}\right) + \ln(p_0(\hat{\boldsymbol{x}}_i;\boldsymbol{\beta})) \right] \\
&= \sum_{i=1}^{m} \left[y_i \ln\left(e^{\boldsymbol{\beta}^{\mathrm{T}}\hat{\boldsymbol{x}}_i}\right) + \ln\left(\frac{1}{1+e^{\boldsymbol{\beta}^{\mathrm{T}}\hat{\boldsymbol{x}}_i}}\right) \right] \\
&= \sum_{i=1}^{m} \left[y_i \boldsymbol{\beta}^{\mathrm{T}}\hat{\boldsymbol{x}}_i - \ln\left(1+e^{\boldsymbol{\beta}^{\mathrm{T}}\hat{\boldsymbol{x}}_i}\right) \right]
\end{aligned} \tag{7}
$$

最大化式(7)等价于最小化

$$
\ell(\boldsymbol{\beta}) = \sum_{i=1}^{m} \left[-y_i \boldsymbol{\beta}^{\mathrm{T}}\hat{\boldsymbol{x}}_i + \ln\left(1+e^{\boldsymbol{\beta}^{\mathrm{T}}\hat{\boldsymbol{x}}_i}\right) \right] \tag{8}
$$

式(8)是关于 $\boldsymbol{\beta}$ 的高阶可导连续凸函数。因此当 $\frac{\partial \ell(\boldsymbol{\beta})}{\partial \boldsymbol{\beta}} = 0$ 时，解得 $\boldsymbol{\beta}^* = \arg\min_{\boldsymbol{\beta}} \ell(\boldsymbol{\beta})$。

我们可以使用牛顿迭代法求解 $\frac{\partial \ell(\boldsymbol{\beta})}{\partial \boldsymbol{\beta}} = 0$。迭代式为

$$
\boldsymbol{\beta}' = \boldsymbol{\beta} - \left(\frac{\partial^2 \ell(\boldsymbol{\beta})}{\partial \boldsymbol{\beta} \partial \boldsymbol{\beta}^{\mathrm{T}}}\right)^{-1} \frac{\partial \ell(\boldsymbol{\beta})}{\partial \boldsymbol{\beta}} \tag{9}
$$

$$
\begin{aligned}
\frac{\partial \ell(\boldsymbol{\beta})}{\partial \boldsymbol{\beta}} &= \sum_{i=1}^{m} \left[-y_i \frac{\partial\left(\boldsymbol{\beta}^{\mathrm{T}}\hat{\boldsymbol{x}}_i\right)}{\partial \boldsymbol{\beta}} + \frac{\partial \ln\left(1+e^{\boldsymbol{\beta}^{\mathrm{T}}\hat{\boldsymbol{x}}_i}\right)}{\partial \boldsymbol{\beta}} \right] \\
&= \sum_{i=1}^{m} \left[-y_i \hat{\boldsymbol{x}}_i + \frac{1}{1+e^{\boldsymbol{\beta}^{\mathrm{T}}\hat{\boldsymbol{x}}_i}} \frac{\partial(1+e^{\boldsymbol{\beta}^{\mathrm{T}}\hat{\boldsymbol{x}}_i})}{\partial \boldsymbol{\beta}} \right] \\
&= \sum_{i=1}^{m} \left(-y_i \hat{\boldsymbol{x}}_i + \frac{e^{\boldsymbol{\beta}^{\mathrm{T}}\hat{\boldsymbol{x}}_i}\hat{\boldsymbol{x}}_i}{1+e^{\boldsymbol{\beta}^{\mathrm{T}}\hat{\boldsymbol{x}}_i}} \right) \\
&= -\sum_{i=1}^{m} \hat{\boldsymbol{x}}_i \left(y_i - \frac{e^{\boldsymbol{\beta}^{\mathrm{T}}\hat{\boldsymbol{x}}_i}}{1+e^{\boldsymbol{\beta}^{\mathrm{T}}\hat{\boldsymbol{x}}_i}} \right)
\end{aligned} \tag{10}
$$

$$
\begin{aligned}
\frac{\partial^2 \ell(\boldsymbol{\beta})}{\partial \boldsymbol{\beta} \partial \boldsymbol{\beta}^{\mathrm{T}}} &= \sum_{i=1}^{m} \left[\frac{\partial \left(-y_i \hat{\boldsymbol{x}}_i\right)}{\partial \boldsymbol{\beta}^{\mathrm{T}}} + \frac{\partial \left(\frac{e^{\boldsymbol{\beta}^{\mathrm{T}} \hat{\boldsymbol{x}}_i} \hat{\boldsymbol{x}}_i}{1+e^{\boldsymbol{\beta}^{\mathrm{T}} \hat{\boldsymbol{x}}_i}} \right)}{\partial \boldsymbol{\beta}^{\mathrm{T}}} \right] \\
&= \sum_{i=1}^{m} \hat{\boldsymbol{x}}_i \frac{\partial \left(\frac{e^{\boldsymbol{\beta}^{\mathrm{T}} \hat{\boldsymbol{x}}_i}}{1+e^{\boldsymbol{\beta}^{\mathrm{T}} \hat{\boldsymbol{x}}_i}} \right)}{\partial \boldsymbol{\beta}^{\mathrm{T}}} \\
&= \sum_{i=1}^{m} \hat{\boldsymbol{x}}_i \left[\frac{\partial \left(e^{\boldsymbol{\beta}^{\mathrm{T}} \hat{\boldsymbol{x}}_i} \right)}{\partial \boldsymbol{\beta}^{\mathrm{T}}} \left(1 + e^{\boldsymbol{\beta}^{\mathrm{T}} \hat{\boldsymbol{x}}_i}\right)^{-1} + \frac{\partial \left(1 + e^{\boldsymbol{\beta}^{\mathrm{T}} \hat{\boldsymbol{x}}_i}\right)^{-1}}{\partial \boldsymbol{\beta}^{\mathrm{T}}} e^{\boldsymbol{\beta}^{\mathrm{T}} \hat{\boldsymbol{x}}_i} \right] \\
&= \sum_{i=1}^{m} \hat{\boldsymbol{x}}_i \left[\frac{e^{\boldsymbol{\beta}^{\mathrm{T}} \hat{\boldsymbol{x}}_i} \hat{\boldsymbol{x}}_i^{\mathrm{T}}}{1 + e^{\boldsymbol{\beta}^{\mathrm{T}} \hat{\boldsymbol{x}}_i}} - \frac{e^{2\boldsymbol{\beta}^{\mathrm{T}} \hat{\boldsymbol{x}}_i} \hat{\boldsymbol{x}}_i^{\mathrm{T}}}{(1 + e^{\boldsymbol{\beta}^{\mathrm{T}} \hat{\boldsymbol{x}}_i})^2} \right] \\
&= \sum_{i=1}^{m} \hat{\boldsymbol{x}}_i \hat{\boldsymbol{x}}_i^{\mathrm{T}} \frac{e^{\boldsymbol{\beta}^{\mathrm{T}} \hat{\boldsymbol{x}}_i}}{1 + e^{\boldsymbol{\beta}^{\mathrm{T}} \hat{\boldsymbol{x}}_i}} \left(1 - \frac{e^{\boldsymbol{\beta}^{\mathrm{T}} \hat{\boldsymbol{x}}_i}}{1 + e^{\boldsymbol{\beta}^{\mathrm{T}} \hat{\boldsymbol{x}}_i}} \right)
\end{aligned} \tag{11}
$$

1.3 对数几率回归算法设计思路

西瓜包含“密度”和“含糖率”两个参数，因此可将西瓜看做横坐标为“密度”纵坐标为“含糖率”的坐标系中的向量 $\boldsymbol{x}_i = \begin{pmatrix} x_{i1} \\ x_{i2} \end{pmatrix}$，然后利用 NumPy 的矩阵拼接函数“numpy.c_[a,b]”，将 $\{\boldsymbol{x}_i\}_{i=1}^m$ 扩展为 $\hat{\boldsymbol{x}}_i = (\boldsymbol{x}_i; 1)$。

式(10)和式(11)中有公共的项

$$
p_1(\hat{\boldsymbol{x}}_i; \boldsymbol{\beta}) = \frac{e^{\boldsymbol{\beta}^{\mathrm{T}} \hat{\boldsymbol{x}}_i}}{1 + e^{\boldsymbol{\beta}^{\mathrm{T}} \hat{\boldsymbol{x}}_i}} \tag{12}
$$

不妨设计函数“p1(x, beta)”用于计算该项的值。此外，程序中需要定义函数“get_partial1(beta)”和“get_partial2(beta)”用于计算一阶、二阶偏导数值；两函数中使用函数“p1(x, beta)”计算公共项。

在牛顿迭代函数“newton(ini_beta,error)”中，程序利用式(9)、借助求偏导数的两个函数从 ini_beta 开始迭代。为了设置迭代的终止条件，我们需要设置计算 $\ell(\boldsymbol{\beta})$ 的函数“ell(beta)”，当某一次迭代与上一次迭代得到的 $\ell(\boldsymbol{\beta})$ 值相差不超过设置的误差容许范围 error 时，停止迭代。

综上所述，函数的依赖关系如图2。

最后，我们需要用可视化的方式展示分类结果。根据图1，二分类的界线为 $\boldsymbol{\beta}^{\mathrm{T}} \hat{\boldsymbol{x}} = 0$，这个界线是二维平面上的一条直线 $\beta_1 x_1 + \beta_2 x_2 + \beta_3 = 0$；这条直线的上方的点代入对数几率函数值大于 0.5，因此为正例（好瓜），直线下方的则为反例（坏瓜）。理想的情况是直线异侧随机各取一点真实标记不一致，而直线同侧任意两点真实标记相同。

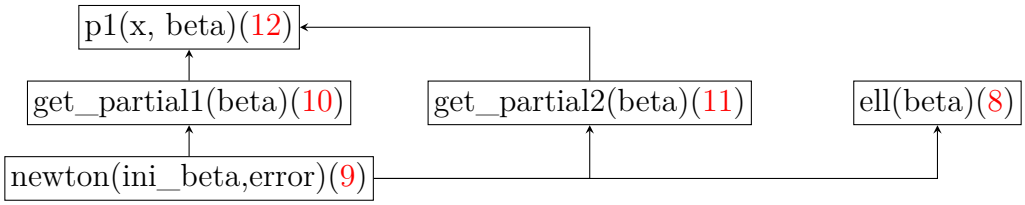

图 2: 函数间依赖关系

1.4 核心代码详解

1.4.1 数据集格式

西瓜数据按照“(密度，含糖率)”的格式存储为 17×2 的矩阵 X，接着新增一列 1 扩展为 17×3 的矩阵 $\hat{X}$。数组 y 用于存储各个西瓜的真实标记。

```
# [密度,含糖率]
X = np.array([
    [0.697, 0.460],
    [0.774, 0.376],
    [0.634, 0.264],
    [0.608, 0.318],
    [0.556, 0.215],
    [0.403, 0.237],
    [0.481, 0.149],
    [0.437, 0.211],
    [0.666, 0.091],
    [0.243, 0.267],
    [0.245, 0.057],
    [0.343, 0.099],
    [0.639, 0.161],
    [0.657, 0.198],
    [0.360, 0.370],
    [0.593, 0.042],
    [0.719, 0.103]
])
# 得到复合矩阵
X = np.c_[X, np.ones(17)]
# 真实标记
y = np.array([1, 1, 1, 1, 1, 1, 1, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0])
```

1.4.2 计算公共项 p_1

函数接收两个参数 $\hat{\boldsymbol{x}}_i$ 和 $\boldsymbol{\beta}$，计算公共项 $p_1(\hat{\boldsymbol{x}}_i, \boldsymbol{\beta})$。由于程序中向量按行存储，因此这里的计算公式为 $\frac{e^{\boldsymbol{\beta}\hat{\boldsymbol{x}}_i^{\mathrm{T}}}}{1+e^{\boldsymbol{\beta}\hat{\boldsymbol{x}}_i^{\mathrm{T}}}}$。

```
# y=1概率
def p1(x, beta):
    return np.exp(beta @ x.T) / (1 + np.exp(beta @ x.T))
```

1.4.3 计算一阶偏导

函数接收参数 $\boldsymbol{\beta}$，按式(10)计算一阶偏导。

```
# 一阶偏导
def get_partial1(beta):
```

```
    ans = np.zeros(3)
    for i in  range(17):
        ans += X[i] * (y[i] - p1(X[i], beta))
    return -ans
```

1.4.4 计算二阶偏导

函数接收参数 $\boldsymbol{\beta}$，按式(11)计算二阶偏导数。值得注意的是，在 NumPy 中，向量使用“@”运算实际上就是点积；我们需要使用“reshape”函数将向量转为矩阵，用“@”实现矩阵运算得到 3×3 的二阶偏导矩阵。

```
# 二阶偏导
def get_partial2(beta):
    ans = np.zeros((3, 3))
    for i in  range(17):
        p = p1(X[i], beta)
        ans += (X[i].reshape((3, 1)) @ X[i].reshape((1, 3))) * p * (1 - p)
    return ans
```

1.4.5 计算目标函数 ℓ

函数接收参数 $\boldsymbol{\beta}$，按式(8)计算目标函数值。在函数中 $\boldsymbol{\beta}^{\mathrm{T}}\hat{\boldsymbol{x}}_i$ 用向量点乘表示。

```
# 目标函数
def ell(beta):
    ans = 0
    for i in  range(17):
        ans = ans - y[i] * np.dot(beta, X[i]) + np.log(1 + np.exp(np.dot(beta, X[i])))
    return ans
```

1.4.6 牛顿迭代过程

函数接收参数为 $\boldsymbol{\beta}$ 初始值、可接受的误差 $error$。函数记录两个值 old_l 和 cur_l，分别表示当前迭代和上一步迭代得到的目标函数值，当 $|\mathrm{old}_l - \mathrm{cur}_l| \leqslant error$ 时停止迭代过程。

```
# 牛顿迭代
def newton(ini_beta, error):
    beta = ini_beta
    old_l = None
    while True:
        subtractor = np.linalg.inv(get_partial2(beta)) @ get_partial1(beta)
        beta = beta - subtractor.reshape((1, 3))[0]
        cur_l = ell(beta)
        if old_l is not None:
            if np. abs(old_l - cur_l) <= error:
                break
```

```
        old_l = cur_l
    return beta
```

1.4.7 调用 scikit-learn

scikit-learn 库提供了类 LogisticRegression，我们可以使用该类实施对数几率回归。对于西瓜数据集 3.0α，其属性 coef_ 为向量 $\boldsymbol{w}$，属性 intercept_ 为截距 b。

```
clf = LogisticRegression(solver='newton-cg').fit(X, y)
# 输出beta
print(np.c_[clf.coef_,clf.intercept_])
```

1.5 实验结果分析

调用函数 newton(np.ones(3),1e-6)，得到 $\boldsymbol{\beta} = \begin{pmatrix} 3.15832966 \\ 12.52119579 \\ -4.42886451 \end{pmatrix}$，分类结果如图3。

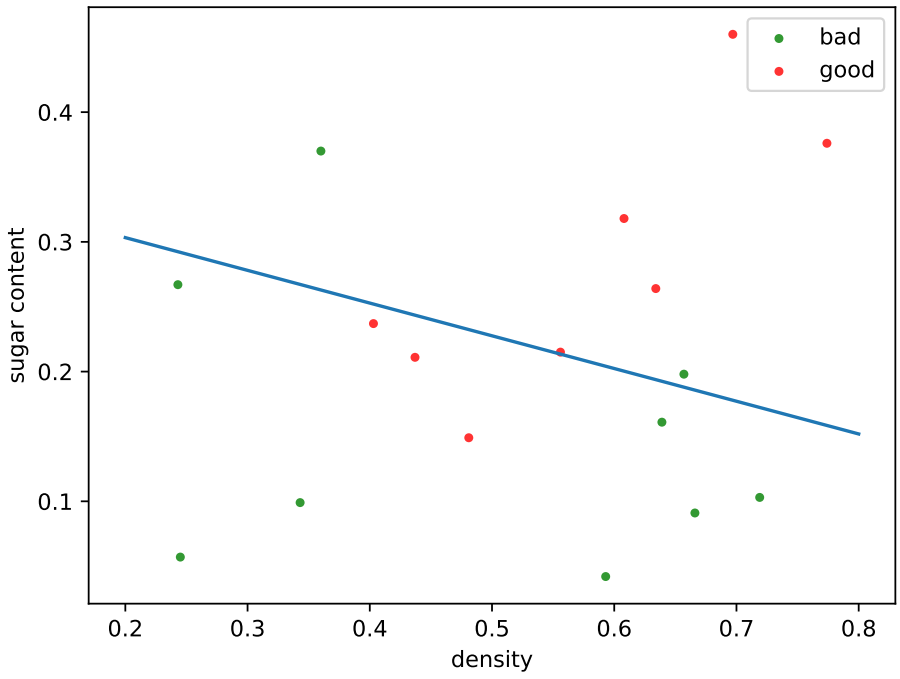

图 3: 对数几率回归在西瓜数据集 3.0α 上的结果。

网络上暂时找不到测试集，若将训练集作为测试集，则模型的分类混淆矩阵为表1。则查准率 $P = \frac{5}{5+2} = 71.43\%$，查全率 $R = \frac{5}{8} = 62.5\%$，总体准确率为 $\frac{12}{17} = 70.59\%$。

真实情况	预测结果	
	正例	反例
正例	5	3
反例	2	7

表 1: 分类结果混淆矩阵

可以看出，由于数据集较少，模型的准确率并不是很高。如图4，使用小规模的西瓜数据集 3.0α 进行训练，$\boldsymbol{\beta}$ 收敛速度极快；若我们需要大规模的数据来训练模型以提高准确率，就需要消耗更长的时间得到最优解，但这样的牺牲在现如今的硬件条件下是可行的。

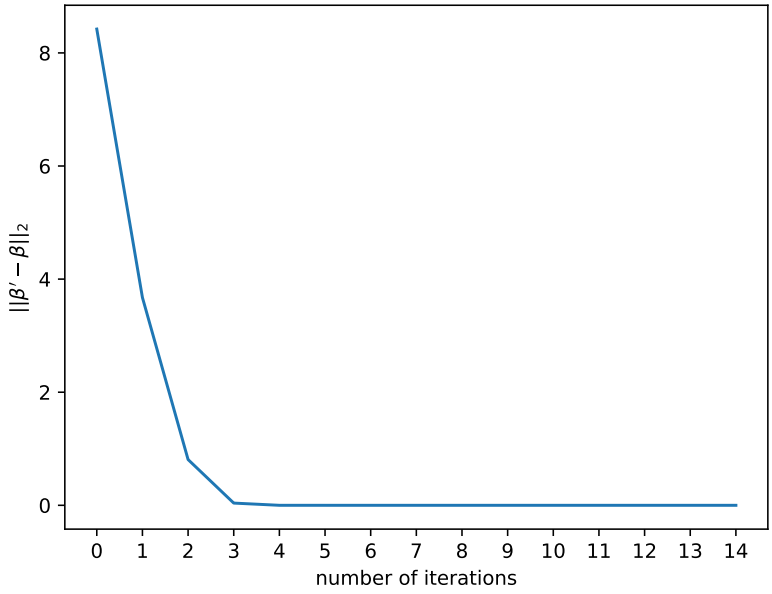

图 4: 迭代收敛速度

1.6 学习收获

在对数几率回归这一节内容，我们使用了极大似然法来得到最优解；而在之前的线性回归一节中，我们使用最小二乘法来得到最优解。我认为这两种方法是得到最优解的不同角度，极大似然法要求趋近真实样本的概率极大，最小二乘法要求均方误差极小。针对回归问题，样本的真实值是不固定的，因此使用最小二乘法较为适合；针对分类问题，样本的真实标记为仅有的几个种类，使用极大似然法得到的模型能在预测未知样本时尽可能接近真实标记。

与自己实现的对数几率回归相比，scikit-learn 提供的函数支持不同的代价函数，可选择求解最优解的方法（梯度下降法、牛顿法等），设置迭代次数、误差范围等参数。

1.7 参考资料

- sklearn.linear_model.LogisticRegression（scikit-learn 中文社区）
- Logistic 回归三分类器（scikit-learn 中文社区）
- 线性模型（scikit-learn 中文社区）
- 《机器学习》3.3；周志华；清华大学出版社
- 《机器学习公式详解》3.27；谢文睿、秦州；人民邮电出版社

2 习题 3.4

2.1 编程题目理解

题目要求，选择 iris 数据集，比较 10 折交叉验证和留一法所估计出的对率回归的错误率。

10 折交叉验证法和留一法只是划分数据集为训练集和测试集的两种方案，本身不涉及具体的数学原理，自己动手实现的意义不大；而且前面已经实现了对率回归算法，这里不再进行重复工作；我们不妨熟悉一下机器学习库 scikit-learn，用其自带函数比较 10 折交叉验证和留一法。

2.2 交叉验证法阐述

“交叉验证法”先将数据集 D 划分为 k 个大小相似的互斥自己，即 $D = D_1 \cup D_2 \cup \cdots \cup D_k$，$D_i \cap D_j = \varnothing\ (i \neq j)$。每个子集 D_i 都尽可能保持数据分布的一致性，即从 D 中通过分层采样得到。然后，每次用 $k-1$ 个子集的并集作为训练集，余下的那个子集作为测试集；这样就可以得到 k 组训练/测试集，从而可进行 k 次训练和测试，最终返回的是这 k 个测试结果的均值。

当上述交叉验证的参数 $k = 10$ 时，称为 10 折交叉验证。

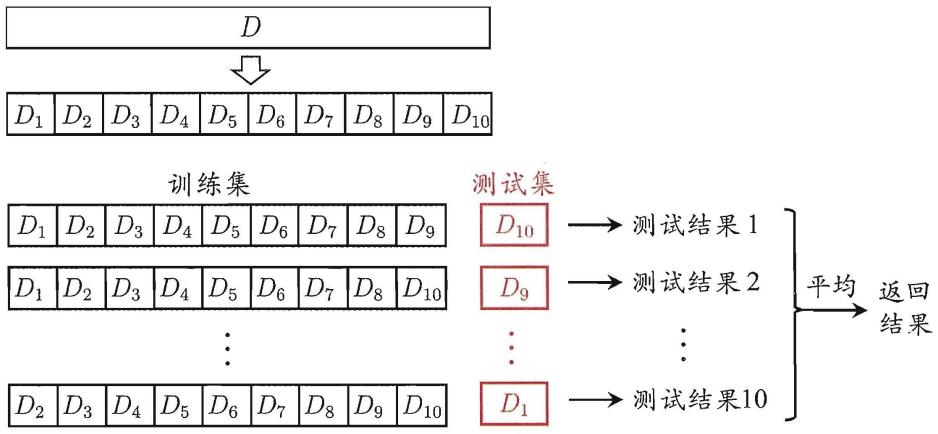

图 5: 10 折交叉验证

假定数据集 D 中包含 m 个样本，若令 $k = m$，则得到了交叉验证法的特例——留一法。显然，留一法不受随机样本划分方式的影响。

2.3 算法设计思路

iris 数据集可从 sklearn.datasets 中获得。sklearn 提供的 iris 降维示例如图6，可以发现 iris 共有三个分类，由于对数几率回归只能进行二分类任务，因此我们在代码中只取其中两个分类进行检验。

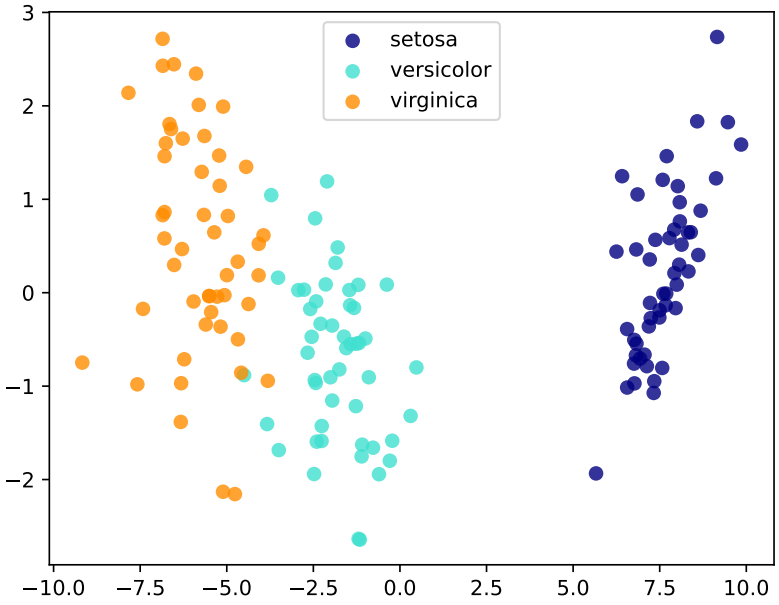

图 6: iris 数据集分类

关于评估方法的对象都在 sklearn.model_selection 软件包中，交叉验证法函数为 cross_val_predict，留一法对象为 LeaveOneOut。

cross_val_predict 为输入数据生成交叉验证的估计值。cross_val_predict 接收对象 estimator，作为拟合数据的模型，在这里就是对数几率回归对象 LogisticRegression。在程序中，调用 cross_val_predict 需要指定模型 estimator、数据集（iris 的参数和真实值）、交叉验证参数 cv（10 折交叉验证 cv=10）。cross_val_predict 返回预测结果数组，调用 sklearn.metrics 包中的 accuracy_score 函数将预测值与真实值比较，可得到准确率。

使用留一法检验，首先需要创建实例。调用 LeaveOneOut.split 函数可以将样分为训练集（99 个）和测试集（1 个），调用 LogisticRegression.fit 可以使用训练集训练对数几率回归模型，接着可用 LogisticRegression.predict 得到测试集的预测值。如此进行 100 次留一法检验，可得到最后的正确率。

2.4 核心代码分析

2.4.1 取数据

iris 数据集共有 150 个样本，每 50 个样本属于一个分类，共有 $C_3^2 = 3$ 种取分类的方法，检验时需要取 100 个只属于两个分类的样本。

```
1 # 取前50个、后50个数据
```

```
iris = datasets.load_iris()
x = np.r_[iris.data[0:50, :], iris.data[100:150, :]]
y = np.r_[iris.target[0:50], iris.target[100:150]]
```

2.4.2 定义模型

这里用对数几率回归检验错误率，因此定义模型 LogisticRegression。

```
logisticRegression = LogisticRegression()
```

2.4.3 10 折交叉验证

调用 cross_val_predict 函数，得到预测值；调用 accuracy_score 函数，得到正确率。

```
# 10折交叉验证
y_predict = cross_val_predict(logisticRegression, x, y, cv=10)
print("10折交叉验证错误率：", format(1 - metrics.accuracy_score(y, y_predict), ".20%"))
```

2.4.4 留一法

创建留一法实例 leaveOneOut，调用其 split 函数，将数据集划分为测试集和训练集。共有 100 个样本，因此有 100 种划分方法，遍历每一个划分方案，统计预测正确的次数，可得到最终的正确率。

```
# 留一法
leaveOneOut = LeaveOneOut()
# 命中数
hits = 0
# 分割为训练集和测试集
for train, test in leaveOneOut.split(x):
    logisticRegression.fit(x[train], y[train])
    p = logisticRegression.predict(x[test])
    if p == y[test]:
        hits += 1
print("留一法错误率", format(1 - hits / 100, ".20%"))
```

2.5 实验结果分析

下载iris 数据集，可以看到共有 setosa、versicolor、virginica 三种鸢尾花，取不同的两种鸢尾花用于进行对数几率回归，得到结果如表2。

留一法使用的训练集与初始可以看到 10 折交叉验证法和留一法所估计出的对率回归的错误率在 iris 数据集上没有显示出明显的差别。因此，我尝试比较了多个 k 折交叉验证的错误率，其中 $k \in \{2, 3, 5, 10, 15, 30, 50\}$，得到的结果如图7。当 k 逐渐增大，对率回归在 iris 数据集上的错误率逐渐减小，并最终达到留一法的错误率，并不再改变。从这一结果来看，对于机器学习算法得到的模型，其针对特定数据集的预测正确率似乎存在阈值，达到阈值后难以继续提高。依据7，我们可初步得到结论：留一

法使用的训练集与与初始数据集相比只少了一个样本，这样就使得绝大多数情况下，留一法中被实际评估的模型与期望评估的用数据集 D 训练出的模型很相似；因此，留一法的评估结果往往比较准确。

评估方法	错误率		
	(setosa, versicolor)	(setosa, virginica)	(versicolor, virginica)
10 折交叉验证	0.00000000000000000000%	0.00000000000000000000%	4.00000000000000035527%
留一法	0.00000000000000000000%	0.00000000000000000000%	4.00000000000000035527%

表 2: 分类结果混淆矩阵

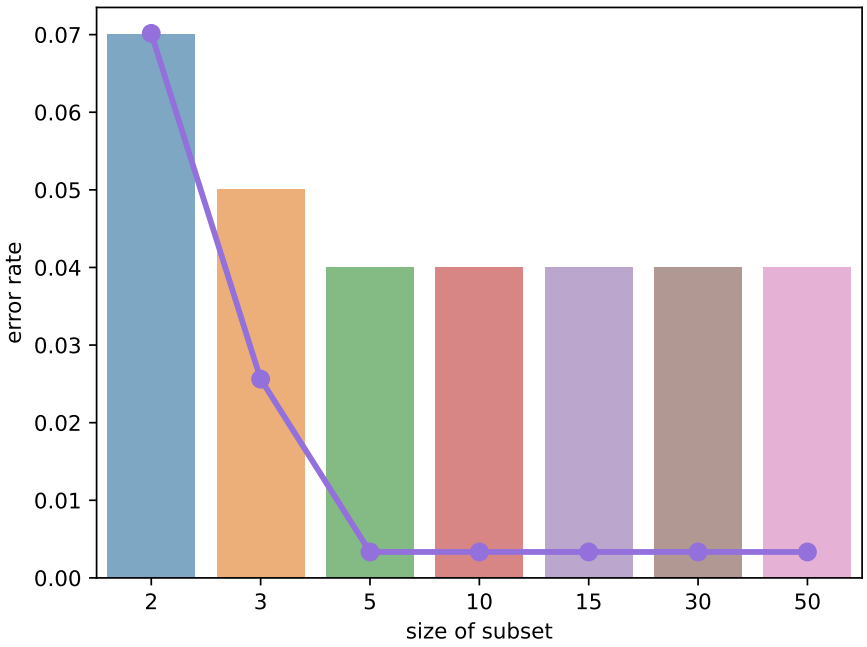

图 7: k 折交叉验证错误率变化

2.6 学习收获

在本次实验中，我们使用对率回归对 iris 数据集分类，对率回归只能完成二分类任务，面对多类别的 iris 数据集显得有些捉襟见肘。

通过查阅资料可知，将二分类的 Logistic 回归扩展，就是多分类的 Softmax 回归。Softmax 函数能将 n 维实向量“压缩”到另一个 n 为实向量 $\sigma(z)$ 中，使得每个元素的范围都在 $[0,1]$ 之间，并且所有元素的和为 1。

$$\boldsymbol{\sigma}(\boldsymbol{z})_i = \frac{e^{z_i}}{\sum_{k=1}^{n} e^{z_k}} \tag{13}$$

假设有 n 种分类，可以将分类的条件概率写成 Softmax 函数的形式

$$P(y_i = c) = \frac{e^{\boldsymbol{\beta}_c^{\mathrm{T}} \hat{\boldsymbol{x}}_i}}{\sum_{k=1}^{n} e^{\boldsymbol{\beta}_k^{\mathrm{T}} \hat{\boldsymbol{x}}_i}} \tag{14}$$

2.7 参考资料

- 《机器学习》2.2.2；周志华；清华大学出版社
- 多分类对数几率回归算法（CSDN sai_simon）
- Multinomial logistic regression（Wikipedia）

3 习题 3.5

3.1 编程题目理解

题目要求编程实现线性判别分析，并给出西瓜数据集 3.0α 上的结果。

在西瓜数据集 3.0α 中，西瓜包含“密度”和“含糖率”两个参数，因此可将西瓜看做横坐标为“密度”纵坐标为“含糖率”的坐标系中的点。使用 LDA，就是要将这些代表西瓜的点投影到平面中的一条直线上，使得好瓜样例的投影点尽可能接近、坏瓜样例的投影点尽可能远离。

3.2 LDA 原理阐述

LDA 的思想非常朴素: 给定训练样例集，设法将样例投影到一条直线上，使得同类样例的投影点尽可能接近、异类样例的投影点尽可能远离；在对新样本进行分类时，将其投影到同样的这条直线上，再根据投影点的位置来确定新样本的类别。

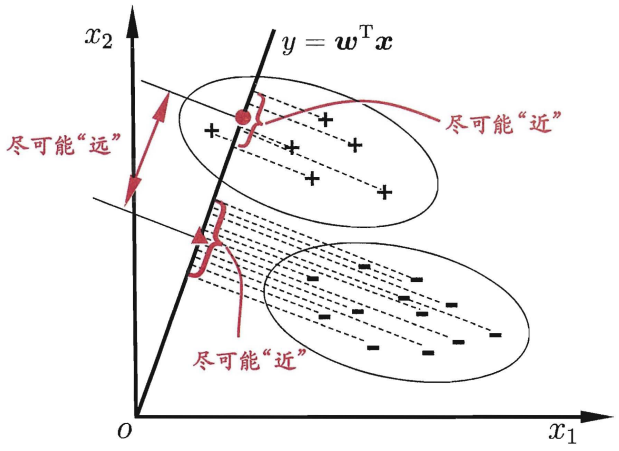

图 8: LDA 二维示意图

对于二分类问题，给定数据集 $D = \{(\boldsymbol{x}_i, y_i)\}_{i=1}^{m}$，$y_i \in \{0, 1\}$，令 X_i、$\boldsymbol{\mu}_i$、$\boldsymbol{\Sigma}_i$ 分别表示第 $i \in \{0, 1\}$ 类示例的集合、均值向量、协方差矩阵。若将数据投影到过原点的直线 $\boldsymbol{w}$ 上，则两类样本的中心在

直线上的投影分别为 $\boldsymbol{w}^{\mathrm{T}}\boldsymbol{\mu}_0$ 和 $\boldsymbol{w}^{\mathrm{T}}\boldsymbol{\mu}_1$；若将所有样本点都投影到直线上，则两类样本的协方差分别为 $\boldsymbol{w}^{\mathrm{T}}\boldsymbol{\Sigma}_0\boldsymbol{w}$ 和 $\boldsymbol{w}^{\mathrm{T}}\boldsymbol{\Sigma}_1\boldsymbol{w}$。由于直线是一维空间，因此 $\boldsymbol{w}^{\mathrm{T}}\boldsymbol{\mu}_0$、$\boldsymbol{w}^{\mathrm{T}}\boldsymbol{\mu}_1$、$\boldsymbol{w}^{\mathrm{T}}\boldsymbol{\Sigma}_0\boldsymbol{w}$、$\boldsymbol{w}^{\mathrm{T}}\boldsymbol{\Sigma}_1\boldsymbol{w}$ 均为实数。

欲使同类样例的投影点尽可能接近，可以让同类样例投影点的协方差尽可能小，即 $\boldsymbol{w}^{\mathrm{T}}\boldsymbol{\Sigma}_0\boldsymbol{w}+\boldsymbol{w}^{\mathrm{T}}\boldsymbol{\Sigma}_1\boldsymbol{w}$ 尽可能小；而欲使异类样例的投影点尽可能远离，可以让类中心之间的距离尽可能大，即 $||\boldsymbol{w}^{\mathrm{T}}\boldsymbol{\mu}_0-\boldsymbol{w}^{\mathrm{T}}\boldsymbol{\mu}_1||_2^2$ 尽可能大。同时考虑二者，则可得到欲最大化的目标

$$\begin{aligned} J&=\frac{||\boldsymbol{w}^{\mathrm{T}}\boldsymbol{\mu}_0-\boldsymbol{w}^{\mathrm{T}}\boldsymbol{\mu}_1||_2^2}{\boldsymbol{w}^{\mathrm{T}}\boldsymbol{\Sigma}_0\boldsymbol{w}+\boldsymbol{w}^{\mathrm{T}}\boldsymbol{\Sigma}_1\boldsymbol{w}}\\ &=\frac{\boldsymbol{w}^{\mathrm{T}}(\boldsymbol{\mu}_0-\boldsymbol{\mu}_1)[\boldsymbol{w}^{\mathrm{T}}(\boldsymbol{\mu}_0-\boldsymbol{\mu}_1)]^{\mathrm{T}}}{\boldsymbol{w}^{\mathrm{T}}\boldsymbol{\Sigma}_0\boldsymbol{w}+\boldsymbol{w}^{\mathrm{T}}\boldsymbol{\Sigma}_1\boldsymbol{w}}\\ &=\frac{\boldsymbol{w}^{\mathrm{T}}(\boldsymbol{\mu}_0-\boldsymbol{\mu}_1)(\boldsymbol{\mu}_0-\boldsymbol{\mu}_1)^{\mathrm{T}}\boldsymbol{w}}{\boldsymbol{w}^{\mathrm{T}}\boldsymbol{\Sigma}_0\boldsymbol{w}+\boldsymbol{w}^{\mathrm{T}}\boldsymbol{\Sigma}_1\boldsymbol{w}} \end{aligned} \tag{15}$$

定义“类内散度矩阵”

$$\begin{aligned} \mathbf{S}_w&=\boldsymbol{\Sigma}_0+\boldsymbol{\Sigma}_1\\ &=\sum_{\boldsymbol{x}\in X_0}(\boldsymbol{x}-\boldsymbol{\mu}_0)(\boldsymbol{x}-\boldsymbol{\mu}_0)^{\mathrm{T}}+\sum_{\boldsymbol{x}\in X_1}(\boldsymbol{x}-\boldsymbol{\mu}_1)(\boldsymbol{x}-\boldsymbol{\mu}_1)^{\mathrm{T}} \end{aligned} \tag{16}$$

定义“类间散度矩阵”

$$\mathbf{S}_b=(\boldsymbol{\mu}_0-\boldsymbol{\mu_1})(\boldsymbol{\mu}_0-\boldsymbol{\mu_1})^{\mathrm{T}} \tag{17}$$

则式(15)可重写为

$$J=\frac{\boldsymbol{w}^{\mathrm{T}}\mathbf{S}_b\boldsymbol{w}}{\boldsymbol{w}^{\mathrm{T}}\mathbf{S}_w\boldsymbol{w}} \tag{18}$$

注意到式(18)的分子和分母都是关于 $\boldsymbol{w}$ 的二次项，因此式(18)的解与 $\boldsymbol{w}$ 的长度无关，只与其方向有关，这一点从图8中可以直观看出。不失一般性，令 $\boldsymbol{w}^{\mathrm{T}}\mathbf{S}_w\boldsymbol{w}=1$，则式(18)等价于

$$\begin{aligned} \min_{\boldsymbol{w}}\quad & -\boldsymbol{w}^{\mathrm{T}}\mathbf{S}_b\boldsymbol{w}\\ \text{s.t.}\quad & \boldsymbol{w}^{\mathrm{T}}\mathbf{S}_w\boldsymbol{w}=1 \end{aligned} \tag{19}$$

根据式(19)，可构造 Lagrange 函数

$$L(\boldsymbol{w},\lambda)=-\boldsymbol{w}^{\mathrm{T}}\mathbf{S}_b\boldsymbol{w}+\lambda(\boldsymbol{w}^{\mathrm{T}}\mathbf{S}_w\boldsymbol{w}-1) \tag{20}$$

已知

$$\begin{aligned} \frac{\partial L(\boldsymbol{w},\lambda)}{\partial\boldsymbol{w}}&=\frac{\partial}{\partial\boldsymbol{w}}[-\boldsymbol{w}^{\mathrm{T}}\mathbf{S}_b\boldsymbol{w}+\lambda(\boldsymbol{w}^{\mathrm{T}}\mathbf{S}_w\boldsymbol{w}-1)]\\ &=-\frac{\partial(\boldsymbol{w}^{\mathrm{T}}\mathbf{S}_b\boldsymbol{w})}{\partial\boldsymbol{w}}+\frac{\partial(\lambda\boldsymbol{w}^{\mathrm{T}}\mathbf{S}_w\boldsymbol{w})}{\partial\boldsymbol{w}}-\frac{\partial\lambda}{\partial\boldsymbol{w}}\\ &=-(\mathbf{S}_b+\mathbf{S}_b^{\mathrm{T}})\boldsymbol{w}+\lambda(\mathbf{S}_w+\mathbf{S}_w^{\mathrm{T}})\boldsymbol{w}\\ &=-2\mathbf{S}_b\boldsymbol{w}+2\lambda\mathbf{S}_w\boldsymbol{w} \end{aligned} \tag{21}$$

令 $\frac{\partial L(\boldsymbol{w},\lambda)}{\partial \boldsymbol{w}} = 0$，则有 $\mathbf{S}_b\boldsymbol{w} = \lambda\mathbf{S}_w\boldsymbol{w}$，即 $(\boldsymbol{\mu}_0 - \boldsymbol{\mu}_1)(\boldsymbol{\mu}_0 - \boldsymbol{\mu}_1)^{\mathrm{T}}\boldsymbol{w} = \lambda\mathbf{S}_w\boldsymbol{w}$。由于 $(\boldsymbol{\mu}_0 - \boldsymbol{\mu}_1)^{\mathrm{T}}\boldsymbol{w}$ 为实数且矩阵乘法满足结合律，因此 $\mathbf{S}_b\boldsymbol{w}$ 为 $(\boldsymbol{\mu}_0 - \boldsymbol{\mu}_1)$ 方向的向量；我们并不关心 λ 的数值，并且 $\boldsymbol{w}$ 的长度也不影响结果，所以不妨令 $\mathbf{S}_b\boldsymbol{w} = \lambda(\boldsymbol{\mu}_0 - \boldsymbol{\mu}_1)$，得到最终结果

$$\boldsymbol{w} = \mathbf{S}_w^{-1}(\boldsymbol{\mu}_0 - \boldsymbol{\mu}_1) \tag{22}$$

3.3 LDA 算法设计思路

西瓜包含“密度”和“含糖率”两个参数，因此可将西瓜看做横坐标为“密度”纵坐标为“含糖率”的坐标系中的向量 $\boldsymbol{x}_i = \begin{pmatrix} x_{i1} \\ x_{i2} \end{pmatrix}$。正例和反例分别组成矩阵 $X_1 = \begin{pmatrix} \boldsymbol{x}_1^{\mathrm{T}} \\ \boldsymbol{x}_2^{\mathrm{T}} \\ \vdots \\ \boldsymbol{x}_8^{\mathrm{T}} \end{pmatrix}$ 和 $X_0 = \begin{pmatrix} \boldsymbol{x}_9^{\mathrm{T}} \\ \boldsymbol{x}_{10}^{\mathrm{T}} \\ \vdots \\ \boldsymbol{x}_{17}^{\mathrm{T}} \end{pmatrix}$，我们可以使用 NumPy 的 ndarray 类型表示这一矩阵结构。接着就可以调用“np.mean(X, axis=0).reshape((2, 1))”得到均值向量 $\boldsymbol{\mu}_i$，注意这里的均值向量为二维列向量。然后需要计算协方差矩阵，已知 $\Sigma_i = \sum\limits_{\boldsymbol{x}\in X_i}(\boldsymbol{x} - \boldsymbol{\mu}_i)(\boldsymbol{x} - \boldsymbol{\mu}_i)^{\mathrm{T}}$，因此需要将代码中的反例和正例矩阵转置后与均值向量相减，在代码中表示为“Sigma = (X.T - mu) @ (X.T - mu).T”。类内散度矩阵就是两个协方差矩阵的简单相加，可写成“Sw = Sigma0 + Sigma1”。最后调用 NumPy 求解矩阵逆的函数，可计算得到直线向量 $\boldsymbol{w}$，“w = np.linalg.inv(Sw) @ (mu0 - mu1)”。计算流程如图9。

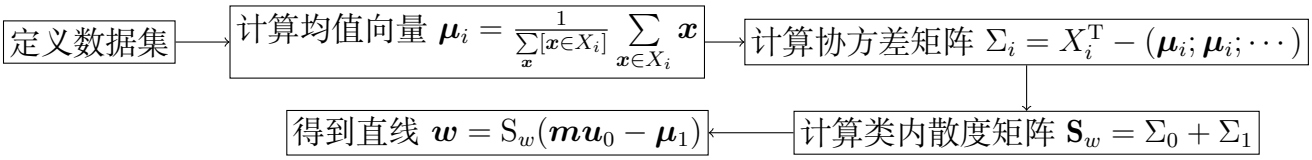

图 9: LDA 计算流程

在得到直线 $\boldsymbol{w}$ 后，我们可以借用 matplotlib 库画出投影的图形。首先，调用 scatter 函数可画出原始点的位置，调用 plot 函数可画出直线 $\boldsymbol{w}$。我们主要考虑如何计算得到投影后的点。已知直线向量为 $\boldsymbol{w}$，原始点表示的向量为 $\boldsymbol{x}$，那么两向量点积 $\boldsymbol{w}\cdot\boldsymbol{x} = ||\boldsymbol{w}||_2||x||_2\cos<\boldsymbol{w},\boldsymbol{x}>$，于是有投影 $d = ||\boldsymbol{x}||_2\cos<\boldsymbol{w},\boldsymbol{x}> = \frac{\boldsymbol{w}\cdot\boldsymbol{x}}{||\boldsymbol{w}||_2}$。设 $\boldsymbol{w} = \begin{pmatrix} w_1 \\ w_2 \end{pmatrix}$，则直线同 x 轴夹角 $\theta = \arctan\frac{w_2}{w_1}$，投影点坐标为 $(d\cos\theta, d\sin\theta)$。在代码中，可调用 NumPy 的数学函数得到投影点坐标。

3.4 核心代码详解

3.4.1 数据集格式

好瓜和坏瓜都用 NumPy 的 ndarry 格式存储，构成矩阵 $X = \begin{pmatrix} \boldsymbol{x}^{\mathrm{T}} \\ \vdots \end{pmatrix}$，数组 y 表示瓜的好坏。

```
1 # 正例
2 X1 = np.array([
```

```
    [0.697, 0.460],
    [0.774, 0.376],
    [0.634, 0.264],
    [0.608, 0.318],
    [0.556, 0.215],
    [0.403, 0.237],
    [0.481, 0.149],
    [0.437, 0.211]
])

# 反例
X0 = np.array([
    [0.666, 0.091],
    [0.243, 0.267],
    [0.245, 0.057],
    [0.343, 0.099],
    [0.639, 0.161],
    [0.657, 0.198],
    [0.360, 0.370],
    [0.593, 0.042],
    [0.719, 0.103]
])
y = np.array([1, 1, 1, 1, 1, 1, 1, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0])
```

3.4.2 LDA（实现）

均值向量 $\boldsymbol{\mu}_i = \sum_{\boldsymbol{x} \in X_i} \boldsymbol{x}$，因此需要对矩阵 X_i 中的各个行向量求平均，并转为符合3.2公式推导的列向量。接着可按式(16)计算协方差矩阵得到类内散度矩阵 $\mathbf{S}_w$。最后调用 NumPy 库求逆矩阵函数 numpy.linalg.inv 得到直线方向向量。计算得到的直线方向向量需要做一定修正，使其方向朝向 y 正半轴，方便接下来计算投影点坐标。

```
# 均值向量（列向量）
mu0 = np.mean(X0, axis=0).reshape((2, 1))
mu1 = np.mean(X1, axis=0).reshape((2, 1))

# 协方差矩阵
Sigma0 = (X0.T - mu0) @ (X0.T - mu0).T
Sigma1 = (X1.T - mu1) @ (X1.T - mu1).T

# 类内散度矩阵
Sw = Sigma0 + Sigma1

# 直线方向向量
w = np.linalg.inv(Sw) @ (mu0 - mu1)

# 输出直线斜率
```

```
print(w[1] / w[0])

# 使直线朝向为y正半轴
if w[1] < 0:
    w = -w
```

3.4.3 绘制图像

根据3.3处的推导，计算投影点的函数如下。函数首先计算得到投影点与原点之间的距离，再根据向量与 x 正半轴的夹角，得到投影点的坐标。

```
# 获取投影坐标
def get_projection(w, X):
    # 到原点距离
    dis = w.T @ X.T / np.linalg.norm(w)

    # 夹角
    theta = np.arctan(w[1] / w[0])

    # 返回投影点坐标
    return dis * np.cos(theta), dis * np.sin(theta)
```

首先绘制原始点，根据原始点得到投影点，最后将原始点和投影点连线，得到垂直于投影直线的线段。

```
plt.figure()

# 原始点
plt.scatter(X0[:, 0], X0[:, 1], color='green', label='bad', alpha=.8, marker='.')
plt.scatter(X1[:, 0], X1[:, 1], color='red', label='good', alpha=.8, marker='.')

# 坏瓜投影点
pro_X, pro_Y = get_projection(w, X0)
plt.scatter(pro_X, pro_Y, color='green', label='bad(projection)', alpha=.8, marker='x')
# 垂直线
for i in  range(9):
    plt.plot([pro_X.T[i], X0[i][0]], [pro_Y.T[i][0], X0[i][1]], color='green', linestyle='--', linewidth
        =0.5)

# 坏瓜投影点
pro_X, pro_Y = get_projection(w, X1)
plt.scatter(pro_X, pro_Y, color='red', label='good(projection)', alpha=.8, marker='x')
# 垂直线
for i in  range(8):
    plt.plot([pro_X.T[i], X1[i][0]], [pro_Y.T[i][0], X1[i][1]], color='red', linestyle='--', linewidth
        =0.5)

```

```
# 投影直线
plt.plot([0, w[0]], [0, w[1]], label=r'$y=w^T$x')

plt.xlabel('density', fontsize=10)
plt.ylabel('sugar content', fontsize=10)
plt.legend()
plt.show()
```

3.4.4 调用 scikit-learn

scikit-learn 库提供了类 LinearDiscriminantAnalysis，我们可以使用该类实施 LDA 降维。对于西瓜数据集 3.0α，其属性 coef_ 即为直线向量 $\boldsymbol{w}$。

```
w = LinearDiscriminantAnalysis(n_components=1).fit(X, y).coef_
# 输出直线斜率
print(w[0][1] / w[0][0])
```

3.5 实验结果分析

运行代码，解得 $\boldsymbol{w} = \begin{pmatrix} 0.14650982 \\ 0.73871557 \end{pmatrix}$，即直线斜率为 5.04208922，在西瓜数据集 3.0α 上得到的结果如图10。

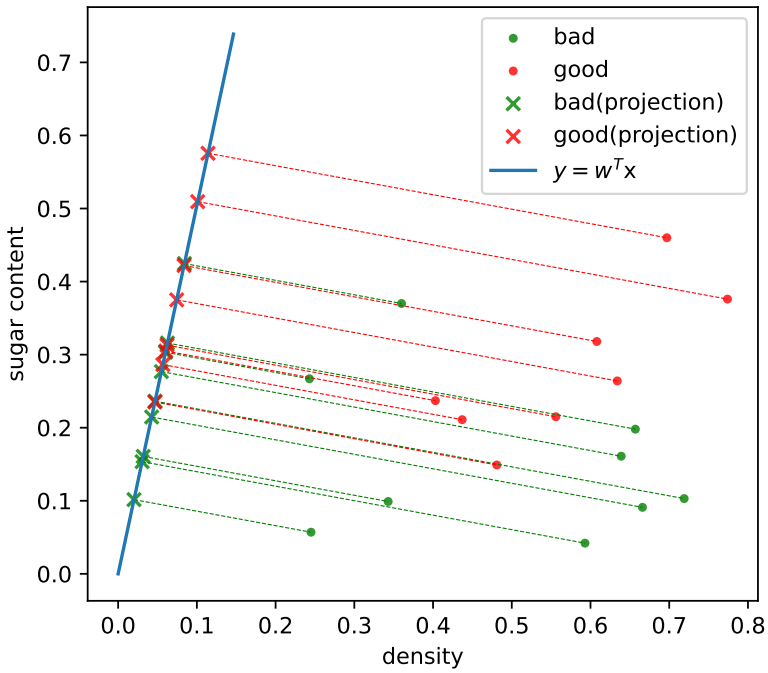

图 10: LDA 在西瓜数据集 3.0α 上的结果。

可以看到，在西瓜数据集 3.0α 上，LDA 的分类效果并不显著，有一些坏瓜（绿点）和好瓜（红点）在投影后距离相差较小，这可能是训练集规模较小造成的。此外，LDA 降维的意义应当大于分类。

3.6 学习收获

在章节3.2，我们直接取均值向量投影为 $\boldsymbol{w}^{\mathrm{T}}\boldsymbol{\mu}_i$，这曾是我的一大疑点。已知直线方向向量 $\boldsymbol{w}$，那么向量 $\boldsymbol{x}$ 在直线上的投影为 $\frac{\boldsymbol{w}\cdot\boldsymbol{x}}{||\boldsymbol{w}||_2}$，即 $\frac{1}{||\boldsymbol{w}||_2}\boldsymbol{w}^{\mathrm{T}}\boldsymbol{x}$；仅在 $\boldsymbol{w}$ 为单位向量时，投影为 $\boldsymbol{w}^{\mathrm{T}}\boldsymbol{x}$。因此接下来的所有推导时理应都会包含 $\frac{1}{||\boldsymbol{w}||_2^k}$ 这一系数，其中 $k\in\mathbb{N}$。因为投影与 $||\boldsymbol{w}||_2$ 并无关系，因此我们不妨直接将这项系数丢弃，并灵活利用这一性质在推导中简化计算。

在本次实验中，LDA 将二维的西瓜数据投射到了一维的直线上；对此加以扩展，将 $\boldsymbol{W}$ 视为一个投影矩阵，则多分类 LDA 就可以将样本投影到 d' 空间，其中 d' 一般远小于原有的属性 d。于是们可通过这个投影来减小样本点的尾数，因此 LDA 是一种使用了类别信息的监督降维技术。

与自己实现的 LDA 相比，scikit-learn 提供的函数能实施高维向量向低维向量的投影，同时支持奇异值分解、最小二乘解、特征值分解三种求解方式，且可以设置收缩参数、先验概率、奇异值阈值等参数。

3.7 参考资料

- sklearn.discriminant_analysis.LinearDiscriminantAnalysis（scikit-learn 中文社区）
- Iris 数据集 LDA 和 PCA 二维投影的比较（scikit-learn 中文社区）
- 《机器学习》3.4；周志华；清华大学出版社
- 《机器学习公式详解》3.37；谢文睿、秦州；人民邮电出版社