北京郵電大學

机器学习

姓　　名 王明瑞

学　　号 2018211807

班　　级 2018211406

学　　院 现代邮政学院(自动化学院)

专　　业 自动化

内　　容 第三章作业

课　　程 机器学习

队　　友 赵浩然 2018211812

2021 年 3 月 31 日

目录

第三章 线性模型

1、数据集介绍

本次实验使用到三个数据集，分别是西瓜数据集 3.0α ，UCI 分类数据集中的糖尿病数据集“Diabetes.xls”和乳腺癌数据集“breast_cancer.csv”。

西瓜数据集 3.0α包含 17 条信息，每条信息对应西瓜的 2 种属性，给出了该西瓜是否为好瓜，“是”表示该西瓜是好瓜，“否”表该西瓜不是好瓜。西瓜数据集 3.0α的具体内容如下图所示。

表 1 西瓜数据集 3.0α

编号	密度	含糖率	好瓜
1	0.697	0.460	是
2	0.774	0.376	是
3	0.634	0.264	是
4	0.608	0.318	是
5	0.556	0.215	是
6	0.403	0.237	是
7	0.481	0.149	是
8	0.437	0.211	是
9	0.666	0.091	否
10	0.243	0.267	否
11	0.245	0.057	否
12	0.343	0.099	否
13	0.639	0.161	否
14	0.657	0.198	否
15	0.360	0.370	否
16	0.593	0.042	否
17	0.719	0.103	否

糖尿病“Diabetes”数据集共包含 768 条信息，每条信息对应一位可能患有糖尿病的患者的 8 个属性，并给出了该患者是否患有糖尿病的结果，“1”表示该患者确实患有糖尿病，“0”表该患者不患有糖尿病。“Diabetes”数据集的具体内容如下图所示。

表 2 糖尿病“Diabetes”数据集

1	类别	属性1	属性2	属性3	属性4	属性5	属性6	属性7	属性8
2	0	6	148	72	35	0	33.6	0.627	50
3	0	8	183	64	0	0	23.3	0.672	32
4	0	0	137	40	35	168	43.1	2.288	33
5	0	3	78	50	32	88	31	0.248	26
6	0	2	197	70	45	543	30.5	0.158	53
7	0	8	125	96	0	0	0	0.232	54
8	0	10	168	74	0	0	38	0.537	34
9	0	1	189	60	23	846	30.1	0.398	59
10	0	5	166	72	19	175	25.8	0.587	51
11	0	7	100	0	0	0	30	0.484	32
12	0	0	118	84	47	230	45.8	0.551	31
13	0	7	107	74	0	0	29.6	0.254	31
14	0	1	115	70	30	96	34.6	0.529	32
15	0	7	196	90	0	0	39.8	0.451	41
16	0	9	119	80	35	0	29	0.263	29
17	0	11	143	94	33	146	36.6	0.254	51
18	0	10	125	70	26	115	31.1	0.205	41
19	0	7	147	76	0	0	39.4	0.257	43

乳腺癌“breast_cancer”数据集共包含 568 条信息，每条信息对应一位可能患有乳腺癌的患者的 30 个属性，并给出了该患者是否患有乳腺癌的结果，“1”表示该患者确实患有乳腺癌，“0”表该患者不患有乳腺癌。乳腺癌“breast_cancer”数据集的具体内容如下所示：

表 3 乳腺癌“breast_cancer”数据集

属性1	属性2	属性3	属性4	属性5	属性6	属性7	属性8	属性9	属性10	属性11	属性12	属性13	属性14	属性15	属性16	属性17	属性18	属性19	属性
17.99	10.38	122.8	1001	0.1184	0.2776	0.3001	0.1471	0.2419	0.07871	1.095	0.9053	8.589	153.4	0.006399	0.04904	0.05373	0.01587	0.03003	0.00
20.57	17.77	132.9	1326	0.08474	0.07864	0.0869	0.07017	0.1812	0.05667	0.5435	0.7339	3.398	74.08	0.005225	0.01308	0.0186	0.0134	0.01389	0.00
19.69	21.25	130	1203	0.1096	0.1599	0.1974	0.1279	0.2069	0.05999	0.7456	0.7869	4.585	94.03	0.00615	0.04006	0.03832	0.02058	0.0225	0.00
11.42	20.38	77.58	386.1	0.1425	0.2839	0.2414	0.1052	0.2597	0.09744	0.4956	1.156	3.445	27.23	0.00911	0.07458	0.05661	0.01867	0.05963	0.00
20.29	14.34	135.1	1297	0.1003	0.1328	0.198	0.1043	0.1809	0.05883	0.7572	0.7813	5.438	94.44	0.01149	0.02461	0.05688	0.01885	0.01756	0.00
12.45	15.7	82.57	477.1	0.1278	0.17	0.1578	0.08089	0.2087	0.07613	0.3345	0.8902	2.217	27.19	0.00751	0.03345	0.03672	0.01137	0.02165	0.00
18.25	19.98	119.6	1040	0.09463	0.109	0.1127	0.074	0.1794	0.05742	0.4467	0.7732	3.18	53.91	0.004314	0.01382	0.02254	0.01039	0.01369	0.00
13.71	20.83	90.2	577.9	0.1189	0.1645	0.09366	0.05985	0.2196	0.07451	0.5835	1.377	3.856	50.96	0.008805	0.03029	0.02488	0.01448	0.01486	0.00
13	21.82	87.5	519.8	0.1273	0.1932	0.1859	0.09353	0.235	0.07389	0.3063	1.002	2.406	24.32	0.005731	0.03502	0.03553	0.01226	0.02143	0.00
12.46	24.04	83.97	475.9	0.1186	0.2396	0.2273	0.08543	0.203	0.08243	0.2976	1.599	2.039	23.94	0.007149	0.07217	0.07743	0.01432	0.01789	0.0
16.02	23.24	102.7	797.8	0.08206	0.06669	0.03299	0.03323	0.1528	0.05697	0.3795	1.187	2.466	40.51	0.004029	0.009269	0.01101	0.007591	0.0146	0.00
15.78	17.89	103.6	781	0.0971	0.1292	0.09954	0.06606	0.1842	0.06082	0.5058	0.9849	3.564	54.16	0.005771	0.04061	0.02791	0.01282	0.02008	0.00
19.17	24.8	132.4	1123	0.0974	0.2458	0.2065	0.1118	0.2397	0.078	0.9555	3.568	11.07	116.2	0.003139	0.08297	0.0889	0.0409	0.04484	0.0
15.85	23.95	103.7	782.7	0.08401	0.1002	0.09938	0.05364	0.1847	0.05338	0.4033	1.078	2.903	36.58	0.009769	0.03126	0.05051	0.01992	0.02981	0.00
13.73	22.61	93.6	578.3	0.1131	0.2293	0.2128	0.08025	0.2069	0.07682	0.2121	1.169	2.061	19.21	0.006429	0.05936	0.05501	0.01628	0.01961	0.00
14.54	27.54	96.73	658.8	0.1139	0.1595	0.1639	0.07364	0.2303	0.07077	0.37	1.033	2.879	32.55	0.005607	0.0424	0.04741	0.0109	0.01857	0.00
14.68	20.13	94.74	684.5	0.09867	0.072	0.07395	0.05259	0.1586	0.05922	0.4727	1.24	3.195	45.4	0.005718	0.01162	0.01998	0.01109	0.0141	0.00
16.13	20.68	108.1	798.8	0.117	0.2022	0.1722	0.1028	0.2164	0.07356	0.5692	1.073	3.854	54.18	0.007026	0.02501	0.03188	0.01297	0.01689	0.00
19.81	22.15	130	1260	0.09831	0.1027	0.1479	0.09498	0.1582	0.05395	0.7582	1.017	5.865	112.4	0.006494	0.01893	0.03391	0.01521	0.01356	0.00
13.54	14.36	87.46	566.3	0.09779	0.08129	0.06664	0.04781	0.1885	0.05766	0.2699	0.7886	2.058	23.56	0.008462	0.0146	0.02387	0.01315	0.0198	0
13.08	15.71	85.63	520	0.1075	0.127	0.04568	0.0311	0.1967	0.06811	0.1852	0.7477	1.383	14.67	0.004097	0.01898	0.01698	0.00649	0.01678	0.00
9.504	12.44	60.34	273.9	0.1024	0.06492	0.02956	0.02076	0.1815	0.06905	0.2773	0.9768	1.909	15.7	0.009606	0.01432	0.01985	0.01421	0.02027	0.00
15.34	14.26	102.5	704.4	0.1073	0.2135	0.2077	0.09756	0.2521	0.07032	0.4388	0.7096	3.384	44.91	0.006789	0.05328	0.06446	0.02252	0.03672	0.00
21.16	23.04	137.2	1404	0.09428	0.1022	0.1097	0.08632	0.1769	0.05278	0.6917	1.127	4.303	93.99	0.004728	0.01259	0.01715	0.01038	0.01083	0.00
16.65	21.38	110	904.6	0.1121	0.1457	0.1525	0.0917	0.1995	0.0633	0.8068	0.9017	5.455	102.6	0.006048	0.01882	0.02741	0.0113	0.01468	0.00
17.14	16.4	116	912.7	0.1186	0.2276	0.2229	0.1401	0.304	0.07413	1.046	0.976	7.276	111.4	0.008029	0.03799	0.03732	0.02397	0.02308	0.00
14.58	21.53	97.41	644.8	0.1054	0.1868	0.1425	0.08783	0.2252	0.06924	0.2545	0.9832	2.11	21.05	0.004452	0.03055	0.02681	0.01352	0.01454	0.00
18.61	20.25	122.1	1094	0.0944	0.1066	0.149	0.07731	0.1697	0.05699	0.8529	1.849	5.632	93.54	0.01075	0.02722	0.05081	0.01911	0.02293	0.00
15.3	25.27	102.4	732.4	0.1082	0.1697	0.1683	0.08751	0.1926	0.0654	0.439	1.012	3.498	43.5	0.005233	0.03057	0.03576	0.01083	0.01768	0.00
17.57	15.05	115	955.1	0.09847	0.1157	0.09875	0.07953	0.1739	0.06149	0.6003	0.8225	4.655	61.1	0.005627	0.03033	0.03407	0.01354	0.01925	0.00

2、Logistics 回归模型

2.1、Logistics 回归模型介绍

Logistic Regression 是经典的分类模型，常用于二分类。Logistics 回归可以认为是一个被 Sigmoid 函数（logistic 方程）所归一化后的线性回归模型。Logistic 回归的本质是：假设数据服从这个分布，然后使用极大似然估计做参数的估计。

2.2、Logistics 回归算法原理

Logistic 函数（或称为 Sigmoid 函数），函数形式为：

$$h(z) = \frac{1}{1+e^{-z}}$$

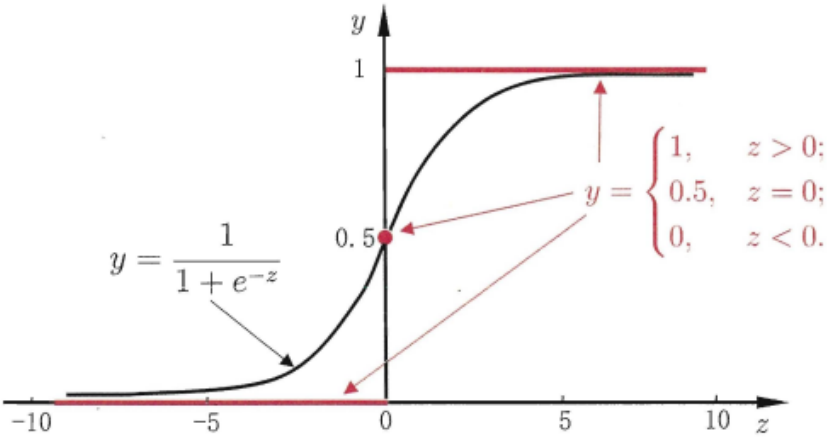


图 2.2.1 Logistic 函数

对于线性边界的情况，边界形式如下：

$$\mathrm{z} = W^T x = w_0 x_0 + w_1 x_1 + \cdots + w_n x_n = \sum_{i=0}^{n} w_i x_i$$

其中，训练数据为向量 $\mathrm{x} = [x_0, x_1, x_2, x_3, \ldots, x_n]$，最佳参数 $\mathrm{W} = [w_0, w_1, w_2, w_3, \ldots, w_n]^T$，构造预测函数为：

$$h_W(x) = \frac{1}{1+e^{-W^T x}}$$

基于最大似然估计推导得到代价函数 J：

$$\mathrm{J}(h_W(x), y) = \begin{cases} -\log\left(h_W(x)\right) & \text{if } y = 1 \\ -\log\left(1 - h_W(x)\right) & \text{if } y = 0 \end{cases}$$

构造整体代价函数 Cost 为：

$$\text{Cost} = -y^{*}\log(h) - (1-y^{*})\log(1-h)$$

使用梯度下降法求解 Cost 的最优解：

$$W_j := W_j - \alpha \frac{\delta}{\delta_{w_j}} Cost(W)$$

$$\frac{\delta}{\delta_W} Cost(W) = -\frac{1}{m}\sum_{i=1}^{m}\left(y_i \frac{1}{h_\theta(x_i)}\frac{\delta}{\delta_W} h_W(x_i) - (1-y_i)\frac{1}{1-h_w(x_i)}\frac{\delta}{\delta_w} h_w(x_i)\right)$$

$$= -\frac{1}{m}\sum_{i=1}^{m}\left(y_i \frac{1}{g(w^T x_i)} - (1-y_i)\frac{1}{1-g(w^T x_i)}\right)\frac{\delta}{\delta w_j} g(w^\top x_i)$$

$$= -\frac{1}{m}\sum_{i=1}^{m}\left(y_i \frac{1}{g(w^T x_i)} - (1-y_i)\frac{1}{1-g(w^\top x_i)}\right) g(w^\top x_i)\left(1-g(w^\top x_i)\right)\frac{\delta}{\delta_w} w^\top x_i$$

$$= -\frac{1}{m}\sum_{i=1}^{m}\left(y_i\left(1-g(w^\top x_i)\right) - (1-y_i)g(w^\top x_i)\right)x_i^j$$

$$= -\frac{1}{m}\sum_{i=1}^{m}\left(y_i - g(w^\top x_i)\right)x_i^j$$

$$= \frac{1}{m}\sum_{i=1}^{m}(h_w(x_i) - y_i)x_i^j$$

梯度下降法算法流程：

1. 初始化 W
2. 更新 W： W −= alpha $* \frac{dCost}{dW}$
3. 迭代达到一定的次数或一定阈值

2.3、Logistics 回归算法核心代码解释

1. 定义 Sigmoid 函数：

```
1. def sigmoid(x, W):
2.     return 1.0 / (1.0 + np.exp(-x * W))
```

2. 梯度下降法求解 logistics 回归权重 W

```
1. # Logistics 回归，返回W 权重
2. def logistic_regression(train_X, labels, alpha=0.01, max_iter=100
   1):
3.     X = np.mat(train_X)
4.     Y = np.mat(labels).T  #  转置为列向量
5.     m, n = np.shape(X)
```

```
    # 随机初始化 W
    W = np.mat(np.random.randn(n, 1))
    # 更新 W
    for i in range(max_iter):
        H = sigmoid(X, W)
        dW = X.T * (H - Y)  # dW:(3,1）根据梯度下降算法，需要先求得 dCost/dW，此处用 dW 代替
        W -= alpha * dW  # 梯度下降 W = W - alpha*dCost/dW

    return W
```

在梯度下降法求解 logistics 回归权重 W 的函数中，我们将训练集 train_X 和真实分类情况 labels 进行了矩阵化处理，得到 X，Y 矩阵。注意此时 X 矩阵为 X=[X, 1]，即为 X 多添加了一列全 1 列向量，方便$W^T x$计算，如下所示。

$$Z = w_1 x_1 + w_2 x_2 + b = [x_1 \quad x_2 \quad 1]\begin{bmatrix} w_1 \\ w_2 \\ b \end{bmatrix}$$

W 更新部分的代码参照梯度下降法推导的公式$W_j := W_j - \alpha \frac{\delta}{\delta_{w_j}} Cost(W)$

编写，$\frac{\delta}{\delta_W} Cost(W) = \frac{1}{m}\sum_{i=1}^{m} (h_w(x_i) - y_i)x_i^j$，推广到矩阵形式即为

$$\frac{\delta}{\delta_W} Cost(W) = X^T \times (H - Y)$$

最后函数返回 W 即为 logistics 回归模型的 W 参数。

3. 结果预测函数

```
# 结果输出函数
def predict(X, W):
    m = len(X)
    pred = np.zeros(m)
    for i in range(m):
        if sigmoid(X[i], W) > 0.5:      # 使用 sigmoid 判断，大于 0.5label 为 1，否则为 0
            pred[i] = 1

    return pred
```

将训练好的模型参数 W 和测试集 X 输入到函数，根据 sigmoid 函数进行划分，大于 0.5 为 1，否则为 0，可以输出预测结果。

4. 数据可视化处理

```
# 数据可视化
def show_diagram(train_X, labels, W):
    w1 = W[0, 0]
    w2 = W[1, 0]
    b = W[2, 0]
    plot_x1 = np.arange(0, 1, 0.01)
    plot_x2 = -w1 / w2 * plot_x1 - b / w2
    plt.plot(plot_x1, plot_x2, c='r', label='decision boundary')
    plt.title('watermelon_3a')
    plt.xlabel('density')
    plt.ylabel('ratio_sugar')
    plt.scatter(train_X[labels == 0, 0].A, train_X[labels == 0, 1].A, marker='^', color='r', s=80, label='bad')
    plt.scatter(train_X[labels == 1, 0].A, train_X[labels == 1, 1].A, marker='o', color='g', s=80, label='good')
    plt.legend(loc='upper right')
    plt.show()
```

根据数据集绘制散点图，并根据分类模型的结果绘制决策边界，$Z = w_1x_1 + w_2x_2 + b$，上式中 x1 对应于横坐标，x2 对用于纵坐标，决策边界 z=0，因此得到直线方程是：

$$0 = \mathrm{b} + w_1x_1 + w_2x_2$$

$$w_2x_2 = -b - w_1x_1$$

$$x_2 = -\frac{b}{w_2} - \frac{w_1}{w_2}x_1$$

2.4、Logistics 回归结果

输出可视化图形：

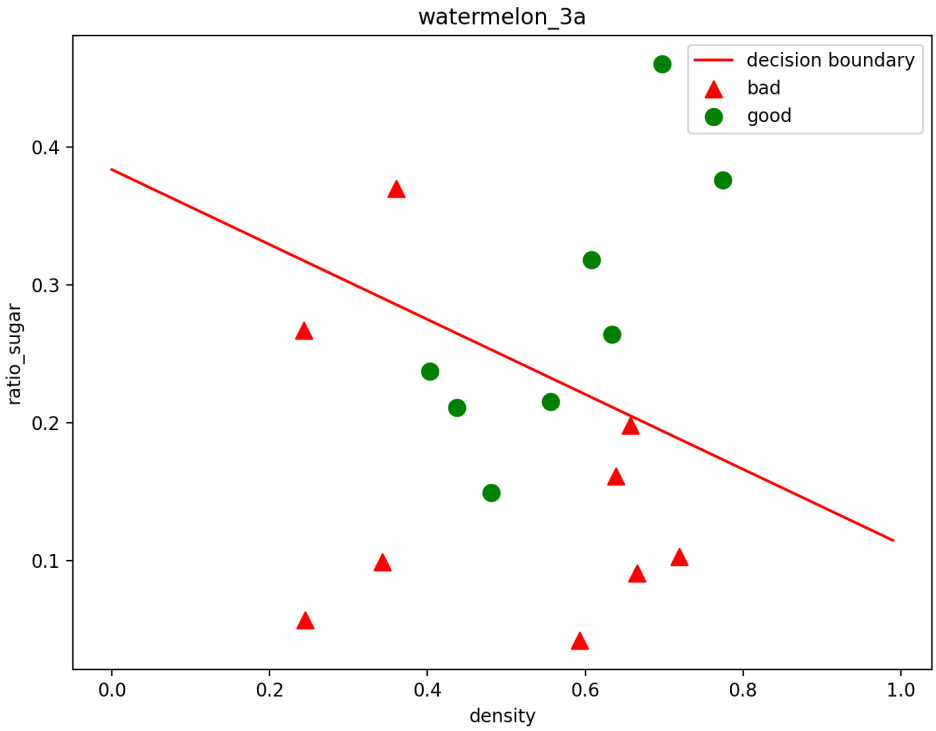


图 2.4.1 logistics 回归结果

可以看到，绿色代表 good，红色代表 bad，经过决策边界划分后，区域被分为两部分，代表二分类结果。

输出结果部分：

```
W: [[ 0.92944027  3.4218329  -1.31239253]]
[1. 1. 1. 1. 0. 0. 0. 0. 0. 0. 0. 0. 0. 1. 0. 0.]
[1. 1. 1. 1. 1. 1. 1. 1. 0. 0. 0. 0. 0. 0. 0. 0.]
              precision    recall  f1-score   support

         0.0       0.67      0.89      0.76         9
         1.0       0.80      0.50      0.62         8

    accuracy                           0.71        17
   macro avg       0.73      0.69      0.69        17
weighted avg       0.73      0.71      0.69        17
```

图 2.4.2 logistics 回归结果数据

logistics 回归最终的输出结果，前两行分别为预测结果和真实结果，下面为二分类的性能评估，准确率在 71%，梯度下降模型做的还不够完美，如果使用随机梯度下降的话，结果可能会更好一些。

3、对率回归模型的检验

选择两个 UCI 数据集，比较 10 折交叉验证法和留一法所估计的对率回归的错误率。
10 折交叉验证法函数：

```
def K_fold(train_X, labels, splits=10):
    order_id = []
    total_acc = 0
    sfolder = StratifiedKFold(n_splits=splits, shuffle=True)  # 十折交叉验证划分数据集
    for num, (train, test) in enumerate(sfolder.split(train_X, labels)):
        x_train = train_X[train, :]
        y_train = labels[train]
        x_test = train_X[test, :]
        y_test = labels[test]
        order_id.extend(test)

        # 开始进行 logistics 回归分类训练
        W = logistics.logistic_regression(x_train, y_train, alpha=0.01, max_iter=1001)  # 计算权重 W
        y_pred = logistics.predict(x_test, W)  # 根据训练好的模型进行预测并输出预测值

        acc = accuracy_score(y_test, y_pred)
        total_acc += acc
        print('第', num + 1, '折验证的错误率', 1 - acc)
    print("十折交叉验证的平均错误率为：", 1 - total_acc / splits)
```

这部分算法在上次作业中已经详细解释过，在本部分不再赘述

留一法函数：

```
def leave_one(train_X, labels):
    loo = LeaveOneOut()
    total_acc = 0
    loo.get_n_splits(train_X)
    num = 0
    for train_index, test_index in loo.split(train_X, labels):
        x_train, x_test = train_X[train_index], train_X[test_index]
        y_train, y_test = labels[train_index], labels[test_index]

        # 开始进行 logistics 回归分类训练
        W = logistics.logistic_regression(x_train, y_train, alpha=0.01, max_
```

```
    iter=1001)  # 计算权重W
12.         y_pred = logistics.predict(x_test, W)   # 根据训练好的模型进行预测并输
    出预测值
13.
14.         acc = accuracy_score(y_test, y_pred)
15.         total_acc += acc
16.         num += 1
17.         if num % 100 == 0:
18.             print("前", num, "组错误率为", 1 - (total_acc / num))
19.     print("留一法的平均错误率为：", 1 - (total_acc / num))
```

如果数据集 D 的大小为 N,那么用 N-1 条数据进行训练，用剩下的一条数据作为验证。用一条数据作为验证的坏处就是可能E_{val}和E_{out}相差很大，所以在留一交叉验证里，每次从 D 中取一组作为验证集，直到所有样本都作过验证集，共计算 N 次，最后对验证误差求平均，这种方法称之为留一法交叉验证。

3.1、乳腺癌“breast_cancer”数据集测试

带入 logistics 回归模型后得到的 10 折交叉验证法和留一法所估计的对率回归的错误率结果输出如下：

```
breast_cancer数据集logistics回归训练结果
(568, 31)
第 1 折验证的错误率 0.5087719298245614
第 2 折验证的错误率 0.052631578947368474
第 3 折验证的错误率 0.052631578947368474
第 4 折验证的错误率 0.14035087719298245
第 5 折验证的错误率 0.03508771929824561
第 6 折验证的错误率 0.10526315789473684
第 7 折验证的错误率 0.07017543859649122
第 8 折验证的错误率 0.1228070175438597
第 9 折验证的错误率 0.0892857142857143
第 10 折验证的错误率 0.1964285714285714
十折交叉验证的平均错误率为： 0.1373433583959901
-----------------------------------------
前 100 组错误率为 0.10999999999999999
前 200 组错误率为 0.08999999999999997
前 300 组错误率为 0.08999999999999997
前 400 组错误率为 0.08999999999999997
前 500 组错误率为 0.08999999999999997
留一法的平均错误率为： 0.0880281690140845
```

3.2、糖尿病“Diabetes”数据集测试

带入 logistics 回归模型后得到的 10 折交叉验证法和留一法所估计的对率回归的错误率结果输出如下：

```
diabetes数据集logistics回归训练结果
(768, 9)
第 1 折验证的错误率 0.35064935064935066
第 2 折验证的错误率 0.35064935064935066
第 3 折验证的错误率 0.3246753246753247
第 4 折验证的错误率 0.2987012987012987
第 5 折验证的错误率 0.35064935064935066
第 6 折验证的错误率 0.36363636363636365
第 7 折验证的错误率 0.35064935064935066
第 8 折验证的错误率 0.38961038961038963
第 9 折验证的错误率 0.3421052631578947
第 10 折验证的错误率 0.3421052631578947
十折交叉验证的平均错误率为： 0.34634313055365684
----------------------------------------
前 100 组错误率为 0.49
前 200 组错误率为 0.53
前 300 组错误率为 0.54
前 400 组错误率为 0.47
前 500 组错误率为 0.43600000000000005
前 600 组错误率为 0.41500000000000004
前 700 组错误率为 0.3957142857142857
留一法的平均错误率为： 0.38541666666666663
```

4、线性判别分析

线性判别分析是一种经典的线性学习方法，亦称“Fisher 判别分析”。LDA 的思想较为朴素：给定训练样例集，设法将阳历投影到一条直线上，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；在队新样本进行分类时，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。

给定数据集$D = \{(\boldsymbol{x}_i, y_i)\}_{i=1}^{m}, y_i \in \{0,1\}$，令$X_i$、$\mu_i$、$\Sigma_i$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$\boldsymbol{w}^{\mathrm{T}}\boldsymbol{\Sigma}_1\boldsymbol{w}$。由于直线式一维空间，因此$\boldsymbol{w}^{\mathrm{T}}\boldsymbol{\mu}_0$、$\boldsymbol{w}^{\mathrm{T}}\boldsymbol{\mu}_1$、$\boldsymbol{w}^{\mathrm{T}}\boldsymbol{\Sigma}_0\boldsymbol{w}$ 和 $\boldsymbol{w}^{\mathrm{T}}\boldsymbol{\Sigma}_1\boldsymbol{w}$均为试数。

故最大化目标

$$J = \frac{\|\boldsymbol{w}^{\mathrm{T}}\boldsymbol{\mu}_0 - \boldsymbol{w}^{\mathrm{T}}\boldsymbol{\mu}_1\|_2^2}{\boldsymbol{w}^{\mathrm{T}}\boldsymbol{\Sigma}_0\boldsymbol{w} + \boldsymbol{w}^{\mathrm{T}}\boldsymbol{\Sigma}_1\boldsymbol{w}} = \frac{\boldsymbol{w}^{\mathrm{T}}(\boldsymbol{\mu}_0 - \boldsymbol{\mu}_1)(\boldsymbol{\mu}_0 - \boldsymbol{\mu}_1)^{\mathrm{T}}\boldsymbol{w}}{\boldsymbol{w}^{\mathrm{T}}(\boldsymbol{\Sigma}_0 + \boldsymbol{\Sigma}_1)\boldsymbol{w}}$$

定义“类内散度矩阵”

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

以及“类间散度矩阵”

$$\mathbf{S}_b = (\boldsymbol{\mu}_0 - \boldsymbol{\mu}_1)(\boldsymbol{\mu}_0 - \boldsymbol{\mu}_1)^{\mathrm{T}}$$

结合“拉格朗日乘子法”，可得

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

在代码实现的过程中

```
mean1 = np.array([np.mean(x1[:, 0]), np.mean(x1[:, 1])])
mean2 = np.array([np.mean(x2[:, 0]), np.mean(x2[:, 1])])
Sw = np.zeros((2, 2))
for i in range(x1.shape[0]):
    Sw = calculate(Sw, x1[i, :] - mean1)
for i in range(x2.shape[0]):
    Sw = calculate(Sw, x2[i, :] - mean2)
w = np.linalg.inv(Sw) @ (mean1 - mean2).transpose()
```

变量 mean1 和 mean2 则表示两类样例的均值向量，Sw 则表示计算出的“类间散度矩阵，进而就计算出矩阵系数向量 w。由于我们仅关心样例点投影到直线后的投影点直接的距离关系，所以只计算出一条过原点的直线，直线沿垂直方向移动并不影响投影结果。

5、运行结果

5.1、第一题

输出可视化图形：

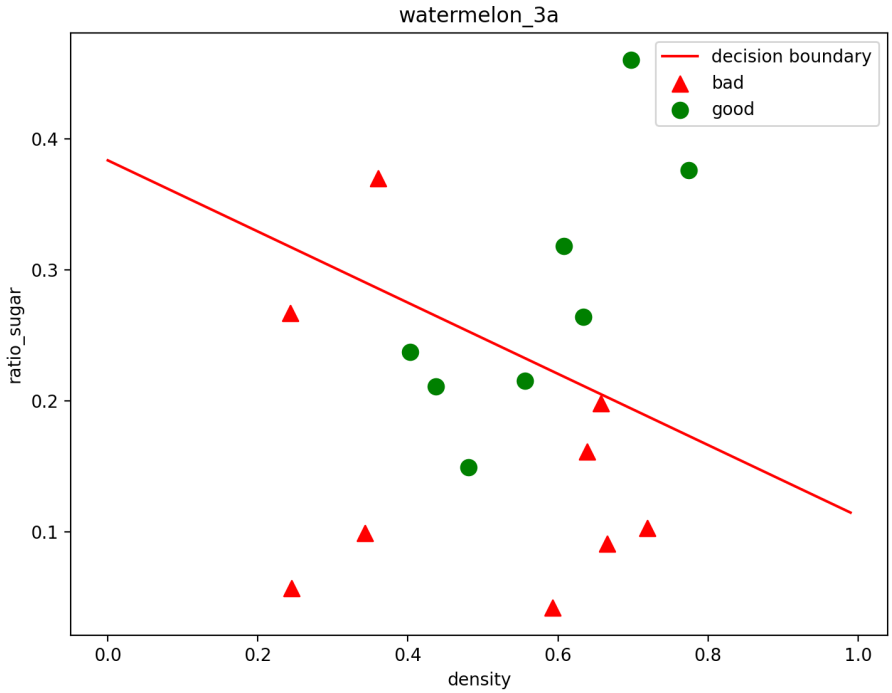


图 2.4.1 logistics 回归结果

可以看到，绿色代表 good，红色代表 bad，经过决策边界划分后，区域被分为两部分，代表二分类结果。

输出结果部分：

```
W: [[ 0.92944027  3.4218329  -1.31239253]]
[1. 1. 1. 1. 0. 0. 0. 0. 0. 0. 0. 0. 0. 1. 0. 0.]
[1. 1. 1. 1. 1. 1. 1. 1. 0. 0. 0. 0. 0. 0. 0. 0.]
              precision    recall  f1-score   support

         0.0       0.67      0.89      0.76         9
         1.0       0.80      0.50      0.62         8

    accuracy                           0.71        17
   macro avg       0.73      0.69      0.69        17
weighted avg       0.73      0.71      0.69        17
```

图 2.4.2 logistics 回归结果数据

5.2、第二题

将 UCI 乳腺癌数据集带入 logistics 回归模型后得到的 10 折交叉验证法和留一法所估计的对率回归的错误率结果输出如下

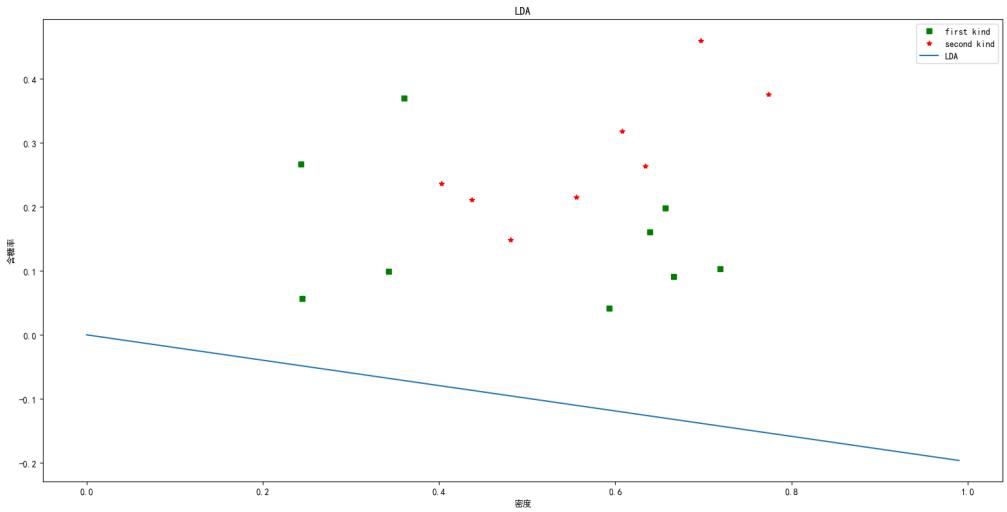
```
breast_cancer数据集logistics回归训练结果
(568, 31)
第 1 折验证的错误率 0.5087719298245614
第 2 折验证的错误率 0.052631578947368474
第 3 折验证的错误率 0.052631578947368474
第 4 折验证的错误率 0.14035087719298245
第 5 折验证的错误率 0.03508771929824561
第 6 折验证的错误率 0.10526315789473684
第 7 折验证的错误率 0.07017543859649122
第 8 折验证的错误率 0.1228070175438597
第 9 折验证的错误率 0.0892857142857143
第 10 折验证的错误率 0.1964285714285714
十折交叉验证的平均错误率为： 0.1373433583959901
----------------------------------------
前 100 组错误率为 0.10999999999999999
前 200 组错误率为 0.08999999999999997
前 300 组错误率为 0.08999999999999997
前 400 组错误率为 0.08999999999999997
前 500 组错误率为 0.08999999999999997
留一法的平均错误率为： 0.0880281690140845
```

将 UCI 糖尿病数据集带入 logistics 回归模型后得到的 10 折交叉验证法和留一法所估计的对率回归的错误率结果输出如下

```
diabetes数据集logistics回归训练结果
(768, 9)
第 1 折验证的错误率 0.35064935064935066
第 2 折验证的错误率 0.35064935064935066
第 3 折验证的错误率 0.3246753246753247
第 4 折验证的错误率 0.2987012987012987
第 5 折验证的错误率 0.35064935064935066
第 6 折验证的错误率 0.36363636363636365
第 7 折验证的错误率 0.35064935064935066
第 8 折验证的错误率 0.38961038961038963
第 9 折验证的错误率 0.3421052631578947
第 10 折验证的错误率 0.3421052631578947
十折交叉验证的平均错误率为： 0.34634313055365684
----------------------------------------
前 100 组错误率为 0.49
前 200 组错误率为 0.53
前 300 组错误率为 0.54
前 400 组错误率为 0.47
前 500 组错误率为 0.43600000000000005
前 600 组错误率为 0.41500000000000004
前 700 组错误率为 0.3957142857142857
留一法的平均错误率为： 0.38541666666666663
```

5.3、第三题

线性判别分析结果为



线性判别分析的直线为

```
LDA曲线为:  -0.14650981657728562 *x  -0.7387155670850031 *y = 0
```

即为

$$-0.1465 * x - 0.7387 * y = 0$$

由线性判别分析结果图可以看出，该数据集使用“线性判别分析”进行降维的结果并不理想。

6、代码

6.3、题 3.3

```
# -*- coding: utf-8 -*
# @File: logistics.py

import numpy as np
import matplotlib.pyplot as plt
from sklearn import metrics
```

```
# 读入文件
def load_data(filename):
    dataset = np.loadtxt(filename, encoding='utf-8', skiprows=1)
    temp = np.mat(dataset[:, 1:3])
    train_X = np.c_[temp, np.ones(len(temp))]  # 增加一列 1，用于计算β
    labels = dataset[:, -1]
    return train_X, labels


def sigmoid(x, W):
    return 1.0 / (1.0 + np.exp(-x * W))


# logistics 回归，返回 W 权重
def logistic_regression(train_X, labels, alpha=0.01, max_iter=1001):
    X = np.mat(train_X)
    Y = np.mat(labels).T
    m, n = np.shape(X)
    # 随机初始化 W
    W = np.mat(np.random.randn(n, 1))
    w_save = []
    # 更新 W
    for i in range(max_iter):
        H = sigmoid(X, W)
        dW = X.T * (H - Y)  # dW:(3,1）根据梯度下降算法，需要先求得
dCost/dW，此处用 dW 代替
        W -= alpha * dW  # 梯度下降 W = W - alpha*dCost/dW

    return W
```

```
# 数据可视化
def show_diagram(train_X, labels, W):
    w1 = W[0, 0]
    w2 = W[1, 0]
    b = W[2, 0]
    plot_x1 = np.arange(0, 1, 0.01)
    plot_x2 = -w1 / w2 * plot_x1 - b / w2
    plt.plot(plot_x1, plot_x2, c='r', label='decision boundary')
    plt.title('watermelon_3a')
    plt.xlabel('density')
    plt.ylabel('ratio_sugar')
    plt.scatter(train_X[labels == 0, 0].A, train_X[labels == 0, 1].A, marker='^',
color='r', s=80, label='bad')
    plt.scatter(train_X[labels == 1, 0].A, train_X[labels == 1, 1].A, marker='o',
color='g', s=80, label='good')
    plt.legend(loc='upper right')
    plt.show()


# 结果输出函数
def predict(X, W):
    m = len(X)
    pred = np.zeros(m)
    for i in range(m):
        if sigmoid(X[i], W) > 0.5:          # 使用 sigmoid 判断，大于 0.5label 为 1，
否则为 0
            pred[i] = 1
```

```
    return pred


def main():
    # 加载数据集
    train_X, labels = load_data('watermelon_3a.txt')
    # 对数据进行 logistics 回归分类
    W = logistic_regression(train_X, labels)
    print('W:', W.T)
    # 将数据集带入模型预测
    y_pred = predict(train_X, W)

    print(y_pred)
    print(labels)
    # 输出二分类的性能评估
    print(metrics.classification_report(labels, y_pred))
    # 可视化
    show_diagram(train_X, labels, W)


if __name__ == '__main__':
    main()
```

6.3、题 3.4

```
# -*- coding: utf-8 -*
# @File: 3.4.1.py

import logistics          # 导入已经习题 3.3 写好的 logistics 文件，直接调用
import numpy as np
import pandas as pd
```

```
import warnings


from sklearn.metrics import accuracy_score
from sklearn.model_selection import StratifiedKFold
from sklearn.model_selection import LeaveOneOut

warnings.filterwarnings('ignore')


# 读入文件
def load_data(filename):
    df = pd.read_csv(filename)
    dataset = df.values
    print(np.shape(dataset))
    temp = np.mat(dataset[:, 0:30])
    train_X = np.c_[temp, np.ones(len(temp))]  # 增加一列 1，用于计算β
    labels = dataset[:, -1]
    return train_X, labels


def leave_one(train_X, labels):
    loo = LeaveOneOut()
    total_acc = 0
    loo.get_n_splits(train_X)
    num = 0
    for train_index, test_index in loo.split(train_X, labels):
        x_train, x_test = train_X[train_index], train_X[test_index]
        y_train, y_test = labels[train_index], labels[test_index]
```

```
        # 开始进行 logistics 回归分类训练
        W = logistics.logistic_regression(x_train, y_train, alpha=0.01, max_iter=1001)  # 计算权重 W
        y_pred = logistics.predict(x_test, W)  # 根据训练好的模型进行预测并输出预测值

        acc = accuracy_score(y_test, y_pred)
        total_acc += acc
        num += 1
        if num % 100 == 0:
            print("前", num, "组错误率为", 1 - (total_acc / num))
    print("留一法的平均错误率为：", 1 - (total_acc / num))


def K_fold(train_X, labels, splits=10):
    order_id = []
    total_acc = 0
    sfolder = StratifiedKFold(n_splits=splits, shuffle=True)  # 十折交叉验证划分数据集
    for num, (train, test) in enumerate(sfolder.split(train_X, labels)):
        x_train = train_X[train, :]
        y_train = labels[train]
        x_test = train_X[test, :]
        y_test = labels[test]
        order_id.extend(test)

        # 开始进行 logistics 回归分类训练
        W = logistics.logistic_regression(x_train, y_train, alpha=0.01, max_iter=1001)  # 计算权重 W
```

```
            y_pred = logistics.predict(x_test, W)    # 根据训练好的模型进行预测并输出预测值

            acc = accuracy_score(y_test, y_pred)
            total_acc += acc
            print('第', num + 1, '折验证的错误率', 1 - acc)
        print("十折交叉验证的平均错误率为：", 1 - (total_acc / splits))


def main():
        x, y = load_data('breast_cancer.csv')       # 读取文件
        K_fold(x, y)
        print("----------------------------------------")
        leave_one(x, y)


if __name__ == '__main__':
        print("breast_cancer 数据集 logistics 回归训练结果")
        main()



# -*- coding: utf-8 -*
# @File: 3.4.2.py
import logistics
import numpy as np
import pandas as pd
import warnings

from sklearn.metrics import accuracy_score
```

```
from sklearn.model_selection import StratifiedKFold
from sklearn.model_selection import LeaveOneOut

warnings.filterwarnings('ignore')


# 读入文件
def load_data(filename):
    df = pd.read_excel(filename)
    dataset = df.values
    print(np.shape(dataset))
    temp = np.mat(dataset[:, 1:])
    train_X = np.c_[temp, np.ones(len(temp))]  # 增加一列 1，用于计算β
    labels = dataset[:, 0]
    return train_X, labels


def leave_one(train_X, labels):
    loo = LeaveOneOut()
    total_acc = 0
    loo.get_n_splits(train_X)
    num = 0
    for train_index, test_index in loo.split(train_X, labels):
        x_train, x_test = train_X[train_index], train_X[test_index]
        y_train, y_test = labels[train_index], labels[test_index]

        # 开始进行 logistics 回归分类训练
        W = logistics.logistic_regression(x_train, y_train, alpha=0.01,
max_iter=1000)  # 计算权重 W
```

```
        y_pred = logistics.predict(x_test, W)  # 根据训练好的模型进行预测并输出预测值

        acc = accuracy_score(y_test, y_pred)
        total_acc += acc
        num += 1
        if num % 100 == 0:
            print("前", num, "组错误率为", 1 - (total_acc / num))
    print("留一法的平均错误率为：", 1 - (total_acc / num))


def K_fold(train_X, labels, splits=10):
    order_id = []
    total_acc = 0
    sfolder = StratifiedKFold(n_splits=splits, shuffle=True)  # 十折交叉验证划分数据集
    for num, (train, test) in enumerate(sfolder.split(train_X, labels)):
        x_train = train_X[train, :]
        y_train = labels[train]
        x_test = train_X[test, :]
        y_test = labels[test]
        order_id.extend(test)

        # 开始进行 logistics 回归分类训练
        W = logistics.logistic_regression(x_train, y_train, alpha=0.01, max_iter=1001)  # 计算权重 W
        y_pred = logistics.predict(x_test, W)  # 根据训练好的模型进行预测并输出预测值

        acc = accuracy_score(y_test, y_pred)
```

```
            total_acc += acc
            print('第', num + 1, '折验证的错误率', 1 - acc)
        print("十折交叉验证的平均错误率为：", 1 - (total_acc / splits))


def main():
        x, y = load_data('Diabetes.xls')    # 读取文件
        K_fold(x, y)
        print("----------------------------------------")
        leave_one(x, y)


if __name__ == '__main__':
        print("diabetes 数据集 logistics 回归训练结果")
        main()
```

6.3、题 3.5

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


def calculate(sw, x):
        sw[0, 0] += x[0] * x[0]
        sw[0, 1] += x[0] * x[1]
        sw[1, 0] += x[1] * x[0]
        sw[1, 1] += x[1] * x[1]
        return sw
```

```
def LDA_w(df):
    df_labels = df.iloc[:, -1]
    labels = list(set(df_labels.values))
    index_1 = []
    index_2 = []
    for i in range(df.shape[0]):
        if df.iloc[i, -1] == labels[0]:
            index_1.append(i)
        else:
            index_2.append(i)
    df1 = df.iloc[index_1, :]
    df2 = df.iloc[index_2, :]
    x1 = df1.values[:, 1:3]
    x2 = df2.values[:, 1:3]
    mean1 = np.array([np.mean(x1[:, 0]), np.mean(x1[:, 1])])
    mean2 = np.array([np.mean(x2[:, 0]), np.mean(x2[:, 1])])
    Sw = np.zeros((2, 2))
    for i in range(x1.shape[0]):
        Sw = calculate(Sw, x1[i, :] - mean1)
    for i in range(x2.shape[0]):
        Sw = calculate(Sw, x2[i, :] - mean2)
    w = np.linalg.inv(Sw) @ (mean1 - mean2).transpose()
    return w


def LDA_plot(df, w):
    df_labels = df.iloc[:, -1]
    labels = list(set(df_labels.values))
    x1 = []
    y1 = []
```

```
    x2 = []
    y2 = []
    for i in range(df.shape[0]):
        if df.iloc[i, -1] == labels[0]:
            x1.append(df.iloc[i, 1])
            y1.append(df.iloc[i, 2])
        else:
            x2.append(df.iloc[i, 1])
            y2.append(df.iloc[i, 2])
    plt.plot(x1, y1, 'gs', label="first kind")
    plt.plot(x2, y2, 'r*', label="second kind")
    x = np.arange(0, 1, 0.01)
    y = np.array((-w[0] * x) / w[1])
    plt.plot(x, y, label="LDA")
    plt.rcParams['font.sans-serif'] = ['SimHei']
    plt.rcParams['axes.unicode_minus'] = False
    plt.xlabel(df.columns[1])
    plt.ylabel(df.columns[2])
    plt.title('LDA')
    plt.legend(loc='best')
    plt.show()


if __name__ == '__main__':
    io = 'D://西瓜数据集 3.0a.xlsx'
    dataframe = pd.read_excel(io)
    w = LDA_w(dataframe)
    LDA_plot(dataframe, w)
    print('LDA 曲线为：', w[0], '*x ', w[1], '*y = 0')
```