# 《机器学习基础》实验报告

| 年级、专业、班级 | 计算机科学与技术 2019 级 | 姓名 | 李燕琴 |
| --- | --- | --- | --- |
| 实验题目 | 对数几率回归算法实践 | | |
| 实验时间 | 2021/10/24 | 实验地点 | D1422 |
| 实验成绩 | | 实验性质 | ☐验证性 ☐设计性 ☐综合性 |

**教师评价：**

☐算法/实验过程正确； ☐源程序/实验内容提交 ☐程序结构/实验步骤合理；

☐实验结果正确； ☐语法、语义正确； ☐报告规范；

其他：

评价教师签名：

## 一、实验目的

掌握线性模型、对率回归算法原理。

## 二、实验项目内容

1. 理解对率回归算法原理。
2. 编程实现对数几率回归算法。
3. 将算法应用于西瓜数据集、鸢尾花数据集分类问题。

## 三、实验过程或算法（源程序）

**（一）对数回归算法原理编程实现：**

1、预测模型：

通过对数几率函数（如下）可以计算得到对应特征x，属于正类的概率。

$$p_1 = \frac{1}{1 + e^{-(w^T x + b)}}$$

2、损失函数:

Sigmoid 函数可以转换为：

$$\ln \frac{p}{1-p} = w^T x + b$$

其中$\ln \frac{p}{1-p}$称为对数几率。w, b 的最佳值可以通过极大似然估计法求解，即:

$$\mathrm{loss}(w,b) = \prod_{i=1}^{m} (y_i p_1 + (1-y_i) p_0)$$

其中$p_1$为样本属于正类的概率，$p_0$为样本属于负类的概率，且$p_1 + p_0 = 1$。

为了避免计算出来的loss(w,b)过小，对其进行取对数得到：

$$\mathrm{loss}(w,b) = \sum_{i=1}^{m} \ln(y_i * p_1 + (1 - y_i) * y_0)$$

令$\beta = [w; b]$，则在训练过程中求解的最小值即是：

$$\beta^* = \mathrm{argmin}_\beta\ \mathrm{loss}(\beta)$$

3、求解最优值

根据牛顿法求解最优值，即β的迭代公式如下：

$$\frac{\partial loss(\beta)}{\partial \beta} = -\sum_{i=1}^{m} x_i (y_i - p_1)$$

$$\frac{\partial^2 loss(\beta)}{\partial \beta \partial \beta^T} = \sum_{i=1}^{m} x_i x_i^T p_1 (1 - p_1)$$

$$\beta' = \beta - \left(\frac{\partial^2 loss(\beta)}{\partial \beta\, \partial \beta^T}\right)^{-1} * \frac{\partial loss(\beta)}{\partial \beta}$$

虽然牛顿法迭代速度快，但是由于二阶导数难以求得数值解，故牛顿法也存在一定的缺陷。故，本实验也结合了梯度下降方法，来弥补牛顿法的缺陷。公式如下：

$$\beta' = \beta - lr * \frac{\partial loss(\beta)}{\partial \beta}$$

至此，整个推导完毕。

**（二）参考资料：**

1. 《机器学习》，周志华
2. https://zhuanlan.zhihu.com/p/36670444

**（三）多分类原理**

Iris 数据中共有三类，属于多分类问题。本模型 MyMultiLogisticRegression 基于周志华老师的《机器学习》中提到的 MvM 方法实现了 3v3 的多分类模型；预测阶段，基于集成学习思想，通过 3 个 LogisticRegression 投票得到最终预测结果。

**（四）代码解释**

1、Baseline 代码基于 sklearn 格式进行框架搭建

```python
class MyLogisticRegression:
    '''
    自定义对数几率回归，实现牛顿法和梯度下降法计算
    '''

    def __init__(self, method="drop", lr=0.1, max_iter=10

    def fit(self, X, Y):···

    def one_diff(self):···

    def double_diff(self):···

    def cal_grad_Newton(self):···

    def cal_grad_drop(self):···

    def predict_prob(self, x=None):···

    def predict(self, x=None):···

    def loss(self, y=None, pred_prob=None):···

    def score(self, x=None, y=None):···

    def draw_process(self, img_name=None):···
```

2、一阶、二阶公式求解导数

```python
def one_diff(self):
    ''' 求解一阶导数 '''
    p1 = self.predict_prob()  # 计算属于正类的概率
    one_diff = -np.sum(np.multiply(self.new_X, self.Y - p1), axis=0)
    # one_diff = -np.mean(self.new_X*(self.Y - p1),axis=0)
    one_diff = one_diff[:, np.newaxis]
    return one_diff

def double_diff(self):
    ''' 求解二阶导数 '''
    p1 = self.predict_prob()  # 计算属于正类的概率
    samples_num, features_num = self.new_X.shape

    double_diff = np.zeros([features_num, features_num])
    for i, a in enumerate(self.new_X):
        a = a[:, np.newaxis]  # (3,1)
        double_diff += np.dot(a, a.T) * p1[i] * (1 - p1[i])
    return double_diff
```

3、梯度下降更新

```python
def cal_grad_Newton(self):
    '''
    牛顿法计算梯度：\n
    beta* = beta - np.linalg.inv(double_diff).dot(one_diff)\n
    '''
    return np.dot(np.linalg.inv(self.double_diff()), self.one_diff())

def cal_grad_drop(self):
    '''
    梯度下降法计算梯度：\n
    beta* = beta - lr * one_diff \n
    NOTE:计算iris数据时，double_diff会非常小，inv(double_diff)会非常大，以至于beta
    '''
    return self.lr * self.one_diff()
```

4、多分类模型训练

```python
def fit(self, train_X, train_Y):
    '''
    基于多个MyLogisticRegression实现多分类\n
    构建M个不同的分类器，并进行fit\n
    :param train_X:
    :param train_Y:
    :return:
    '''
    self.X = train_X
    self.Y = train_Y

    self.class_list = list(set(train_Y.flatten()))  # set->list，好办事
    self.split_list = get_split_lists(len(self.class_list), seed=0)  # 每个分类器的数据
    self.model_list = []
    for split in self.split_list:
        temp_Y = train_Y.copy()
        for idx, y in enumerate(self.class_list):
            temp_Y[temp_Y == y] = split[idx]
        model = MyLogisticRegression(lr=self.lr, max_iter=self.max_iter, seed=self.se
        model.fit(train_X, temp_Y)
        self.model_list.append(model)
```

## （五）源代码

==========================MyLogisticRegression.py==========================

```python
import numpy as np
import matplotlib.pyplot as plt
from utils import my_save_fig
def _sigmoid(z):
    return 1.0 / (1.0 + np.exp(-z))
class MyLogisticRegression:
    '''
    自定义对数几率回归，实现牛顿法和梯度下降法计算
    '''
    def __init__(self, method="drop", lr=0.1, max_iter=10000, seed=None, episilon=1e-6)
-> None:
        '''
        模型初始化
        :param method: ["drop","Newton"] 求解梯度方法。注意：数据较多较杂时，使用 Newton
法，其二阶导数难以求出，建议选择 drop 法
        :param lr:学习率
        :param max_iter:最大迭代次数
        :param seed:随机数种子
        :param episilon:计算精度
        '''
```

```
        methods = ["drop", "Newton"]
        if method not in methods:
            method = "drop"
        if seed is not None:
            np.random.seed(seed)
        self.lr = lr
        self.max_iter = max_iter  # 优化次数限制，防止无限循环
        self.method = method
        self.episilon = episilon  # 计算精度
    def fit(self, X, Y):
        '''
        自定义对数几率回归，牛顿法进行训练\n
        参考资料：\n
        - 《机器学习》-周志华\n
        - 框架参考 https://zhuanlan.zhihu.com/p/36670444 \n
        :param X:
        :param Y:
        :return:
        '''
        self.X = X
        self.Y = Y
        self.train_score_list = []  # 准确率得分
        self.train_loss_list = []  # 损失函数
        self.new_X = np.hstack([X, np.ones([X.shape[0], 1])])
        self.beta = np.random.random([self.new_X.shape[1], 1])
        for i in range(0, self.max_iter):
            delta = self.cal_grad_drop()
            # delta = self.cal_grad_Newton()
            if np.abs(np.max(delta)) < self.episilon:
                break
            self.beta = self.beta - delta
            self.train_loss_list.append(self.loss())
            self.train_score_list.append(self.score())
        # print("迭代次数：", i)
        self.coef_ = self.beta[:-1]
        self.intercept_ = self.beta[-1]
    def one_diff(self):
        ''' 求解一阶导数 '''
        p1 = self.predict_prob()  # 计算属于正类的概率
        one_diff = -np.sum(np.multiply(self.new_X, self.Y - p1), axis=0)
        # one_diff = -np.mean(self.new_X*(self.Y - p1),axis=0)
        one_diff = one_diff[:, np.newaxis]
        return one_diff
    def double_diff(self):
        ''' 求解二阶导数 '''
        p1 = self.predict_prob()  # 计算属于正类的概率
        samples_num, features_num = self.new_X.shape
        double_diff = np.zeros([features_num, features_num])
        for i, a in enumerate(self.new_X):
            a = a[:, np.newaxis]  # (3,1)
            double_diff += np.dot(a, a.T) * p1[i] * (1 - p1[i])
        return double_diff
    def cal_grad_Newton(self):
        '''
        牛顿法计算梯度：\n
        beta* = beta - np.linalg.inv(double_diff).dot(one_diff)\n
        '''
        return np.dot(np.linalg.inv(self.double_diff()), self.one_diff())
    def cal_grad_drop(self):
```

```
        '''
        梯度下降法计算梯度：\n
        beta* = beta - lr * one_diff \n
        NOTE:计算 iris 数据时，double_diff 会非常小，inv(double_diff)会非常大，以至于 beta 非
常大，计算就会出现误差，暂且不明白原因，所以提供了梯度下降法
        '''
        return self.lr * self.one_diff()
    def predict_prob(self, x=None):
        ''' 计算预测概率 '''
        if x is None:
            x = self.X
        new_X = np.hstack([x, np.ones([x.shape[0], 1])])
        pred_prob = _sigmoid(np.dot(new_X, self.beta))
        return pred_prob
    def predict(self, x=None):
        '''计算预测值'''
        if x is None:
            x = self.X
        if len(x.shape) == 1:
            x = x[np.newaxis, :]
        pred_prob = self.predict_prob(x)
        pred = np.array([0 if i < 0.5 else 1 for i in pred_prob])
        pred = pred[:, np.newaxis]
        return pred
    def loss(self, y=None, pred_prob=None):
        '''计算损失函数'''
        if y is None or pred_prob is None:
            y = self.Y
            pred_prob = self.predict_prob(self.X)
        return -np.log(y * (pred_prob) + (1 - y) * (1 - pred_prob)).sum()
    def score(self, x=None, y=None):
        '''计算准确率'''
        if x is None or y is None:
            x = self.X
            y = self.Y
            pred = self.predict(x)
        return (y == pred).sum() / len(y)
    def draw_process(self, img_name=None):
        '''绘制训练过程'''
        fig, axs = plt.subplots(2, 1, sharex='row')
        fig.suptitle(img_name)
        plt.subplots_adjust(left=None, bottom=None, right=None, top=None, wspace=None,
hspace=0.5)
        axs[0].plot(self.train_loss_list, '-')
        axs[0].set_title("loss")
        axs[1].plot(self.train_score_list, '-')
        axs[1].set_title("acc")
        if img_name is not None:
            my_save_fig(fig, img_name)
        plt.show()


======================MyMultiLogisticRegression.py======================
import numpy as np
from collections import Counter
from MyLogisticRegression import MyLogisticRegression
from utils import get_split_lists
class MyMultiLogisticRegression:
    '''
    以 MyLogisticRegression 为 baseline 的多分类模型
```

```
    '''
    def __init__(self, lr=0.1, max_iter=10000, seed=None, episilon=1e-6) -> None:
        if seed is not None:
            np.random.seed(seed)
        self.seed = seed
        self.lr = lr  # 学习率
        self.max_iter = max_iter  # 次数限制，防止无限循环
        self.episilon = episilon  # 计算精度
    def fit(self, train_X, train_Y):
        '''
        基于多个 MyLogisticRegression 实现多分类\n
        构建 M 个不同的分类器，并进行 fit\n
        :param train_X:
        :param train_Y:
        :return:
        '''
        self.X = train_X
        self.Y = train_Y
        # TODO class_list，split_list，model_list 也许可以设置为私有变量
        self.class_list = list(set(train_Y.flatten()))  # set->list，好办事
        self.split_list = get_split_lists(len(self.class_list), seed=0)  # 每个分类器的数
据，只需要重新构造 0,1
        self.model_list = []
        for split in self.split_list:
            temp_Y = train_Y.copy()
            for idx, y in enumerate(self.class_list):
                temp_Y[temp_Y == y] = split[idx]
            model = MyLogisticRegression(lr=self.lr, max_iter=self.max_iter,
seed=self.seed, episilon=self.episilon)
            model.fit(train_X, temp_Y)
            self.model_list.append(model)
    def predict(self, x=None):
        '''计算预测值'''
        if x is None:
            x = self.X.copy()
        pred = []
        if len(x.shape) == 1:
            return self.get_one_pred(x)
        else:
            for row in x:
                pred.append(self.get_one_pred(row))
        pred = np.array(pred)
        pred = pred[:, np.newaxis]
        return pred
    def get_one_pred(self, x):
        '''投票法获取单个预测值'''
        vote_cnt = Counter()
        for i, model in enumerate(self.model_list):
            temp_pred = model.predict(x)
            maybe_class_list = [self.class_list[idx] for idx, flag in
enumerate(self.split_list[i]) if
                                flag == temp_pred]  # 有点长，慢慢看，其实很简单
            vote_cnt.update(maybe_class_list)
        one_pred = vote_cnt.most_common(n=1)[0][0]  # 获取最常见的类别，若有相同的，则按照
字典序排序
        return one_pred
    def score(self, x=None, y=None):
        '''计算准确率'''
        if x is None or y is None:
```

```
            x = self.X
            y = self.Y
        pred = self.predict(x)
        return (y == pred).sum() / len(y)
    def get_params(self):
        '''获取模型参数'''
        print("=" * 20 + "模型参数" + "=" * 20)
        print("model\tcoef_\t\t\tintercept_")
        for i, model in enumerate(self.model_list):
            print(f'{i}\t{model.coef_.flatten().tolist()}\t{model.intercept_}')
    def draw_process(self, img_name=None):
        '''绘制训练过程'''
        for i, model in enumerate(self.model_list):
            model  # type:MyLogisticRegression
            model.draw_process(img_name="LR" + str(i) + " Trian Process")
```

==========================test_watermelon.py=========================

```
import os
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
from sklearn.utils import shuffle
from sklearn.linear_model import LogisticRegression
from sklearn.metrics import classification_report
from MyLogisticRegression import MyLogisticRegression
from utils import my_save_fig
# LogisticRegression 边界函数
my_split_boundary_func = lambda x: (-my_model.intercept_ - my_model.coef_[0] * x) /
my_model.coef_[1]
sk_split_boundary_func = lambda x: (-sk_model.intercept_ - sk_model.coef_[0][0] * x) /
sk_model.coef_[0][1]
def plot_data(X,Y,img_name=None,my_acc=None,sk_acc=None):
    # Watermelon_data
    fig, ax = plt.subplots()
    scatter = ax.scatter(X[:, 0], X[:, 1], c=Y, marker='*')
    handles, labels = scatter.legend_elements()
    legend1 = ax.legend(handles, labels, loc="upper left", title="Classes")
    ax.add_artist(legend1)
    # boundary line
    boundx = np.linspace(0, 1, 50, endpoint=True)
    plt.plot(boundx, my_split_boundary_func(boundx), c='red')
    plt.plot(boundx, sk_split_boundary_func(boundx), c='blue')
    plt.title('Watermelon_data')
    if my_acc is not None and sk_acc is not None:
        plt.text(0.75, 0.7, "my_acc=%.4f\nsk_acc=%.3f" % (my_acc, sk_acc),
transform=ax.transAxes)
    plt.legend(['my_model', 'sk_model'], )
    my_save_fig(fig, img_name)
    plt.show()
if __name__=="__main__":
    # 1、数据读取
    data = pd.read_excel('./Watermelon_data.xls')
    data = shuffle(data)
    data['好瓜'].replace('是',1,inplace=True)
    data['好瓜'].replace('否',0,inplace=True)
    # 2、数据准备
    X = np.array(data[['密度','含糖率']].values)
    Y = np.array(data['好瓜'].values)
    Y = Y[:,np.newaxis]
```

```python
    # 3、训练集和测试集，这里数量比较少，全部作为训练集
    # train_percent = 1
    # train_num = int(np.floor(X.shape[0] * train_percent))
    # trainX,testX = X[0:train_num],X[train_num,-1]
    # trainY,testY = Y[0:train_num],X[train_num,-1]
    trainX = X
    trainY = Y
    # 4、建立模型
    print("="*20+"my model"+"="*20)
    my_model = MyLogisticRegression()
    my_model.fit(trainX,trainY)
    my_acc = my_model.score()
    my_loss = my_model.loss()
    my_predY = my_model.predict(trainX)
    print(f'1、模型参数：\n\tw={my_model.coef_.T}\n\tb={my_model.intercept_}')
    print("2、评级指标：")
    print(f'acc  = {my_acc}')
    print(f'loss = {my_loss}')
    print("3、分类结果：")
    print(classification_report(trainY, my_predY, target_names=['否', '是']))
    # 5、可视化训练过程
    img_name = 'Watermelon_data MyLogisticRegression Trian'
    my_model.draw_process(img_name)
    # ============== sklearn 对比 ==============
    print("=" * 20 + "sk model" + "=" * 20)
    sk_model = LogisticRegression()
    sk_model.fit(trainX,trainY)
    sk_acc = sk_model.score(trainX,trainY)
    sk_predY = sk_model.predict(trainX)
    print(f'1、模型参数：\n\tw={sk_model.coef_}\n\tb={sk_model.intercept_}')
    print("2、评级指标：")
    print(f'acc  = {sk_acc}')
    print("3、分类结果：")
    print(classification_report(trainY, sk_predY, target_names=['否', '是']))
    # 可视化训练结果
    img_name = 'Watermelon_data'
    plot_data(X, Y, img_name, my_acc, sk_acc)
```

==========================test_iris.py=========================

```python
# 导入包
from sklearn import datasets
import numpy as np
from sklearn.model_selection import train_test_split
from MyMultiLogisticRegression import MyMultiLogisticRegression
iris = datasets.load_iris()
# 键：data target target_names
X = iris.data
Y = iris.target
Y = Y[:,np.newaxis]
train_X,test_X,train_Y,test_Y = train_test_split(X,Y,test_size=0.3)
mmLR = MyMultiLogisticRegression()
mmLR.fit(train_X,train_Y)
mmLR.get_params()
mmLR.draw_process()
print("训练准确率:%f"%(mmLR.score()))
print("测试准确率:%f"%(mmLR.score(test_X,test_Y)))
```

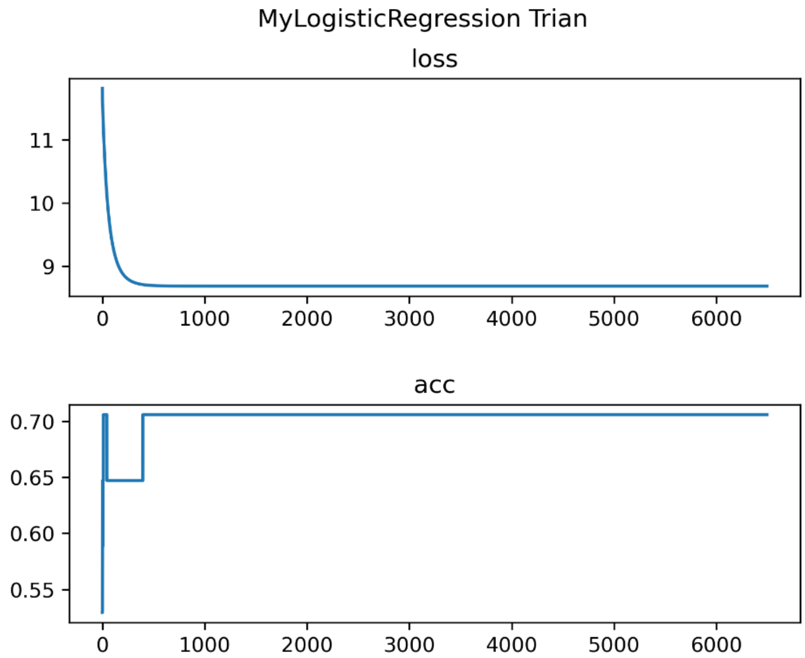
==========================utils.py=========================

```python
import os
import numpy as np
from logging import error
image_cnt = 1
def my_save_fig(fig, img_name=None):
    global image_cnt
    if img_name is None:
        img_name = 'image' + str(image_cnt)
        image_cnt += 1
    save_path = os.path.join(os.getcwd(), img_name)
    fig.savefig(save_path, dpi=300, bbox_inches="tight", pad_inches=0.1)
def get_split_lists(N, M=None, seed=None):
    '''
    不重复生成 M 个长度为 N 的 0，1 序列
    :param N: 序列长度
    :param M: 序列个数
    :param seed: 随机数种子
    :return:
    '''
    if seed is not None:
        np.random.seed(seed)
    if M is None:
        M = N  # 默认 OvR 多分类方法
    if M > N*(N-1)/2:
        error("输入的 M 超过 N*(N-1)/2 的限制")
        return False
    split_list = []  # 输出结果为 0,1 代表二分类形式
    t = M
    while t:
        temp = np.random.randint(0, 2, N)
        if 0 <= sum(temp) < N:
            # 避免和已有 split 刚好相反
            flag = True
            for split in split_list:
                oxr_res = sum(split ^ temp)
                if oxr_res == N or oxr_res == 0:
                    flag = False
                    break
            if flag:
                split_list.append(temp)
                t -= 1
    return split_list
```

# 四、实验结果及分析

## 1、my_model 训练过程

如下图，设置牛顿法精度 1e-15，再迭代 6000 次左右后，结果趋于平稳。




图表 1 my_model 训练过程图

**2、训练结果分析：**

实验中,将自己实现的 LogisticRegression 与 sklearn 中的 LogisticRegression 进行对比，如图 2、图 3，可以看到，在 17 列西瓜分类数据中，my_model 的结果会更好。

```
====================my model====================
1、模型参数：
    w=[[ 3.15832966 12.52119579]]
    b=[-4.42886451]
2、评级指标：
acc  = 0.7058823529411765
loss = 8.683660584232863
3、分类结果：
              precision    recall  f1-score   support


           否       0.70      0.78      0.74         9
           是       0.71      0.62      0.67         8


    accuracy                           0.71        17
   macro avg       0.71      0.70      0.70        17
weighted avg       0.71      0.71      0.70        17
```

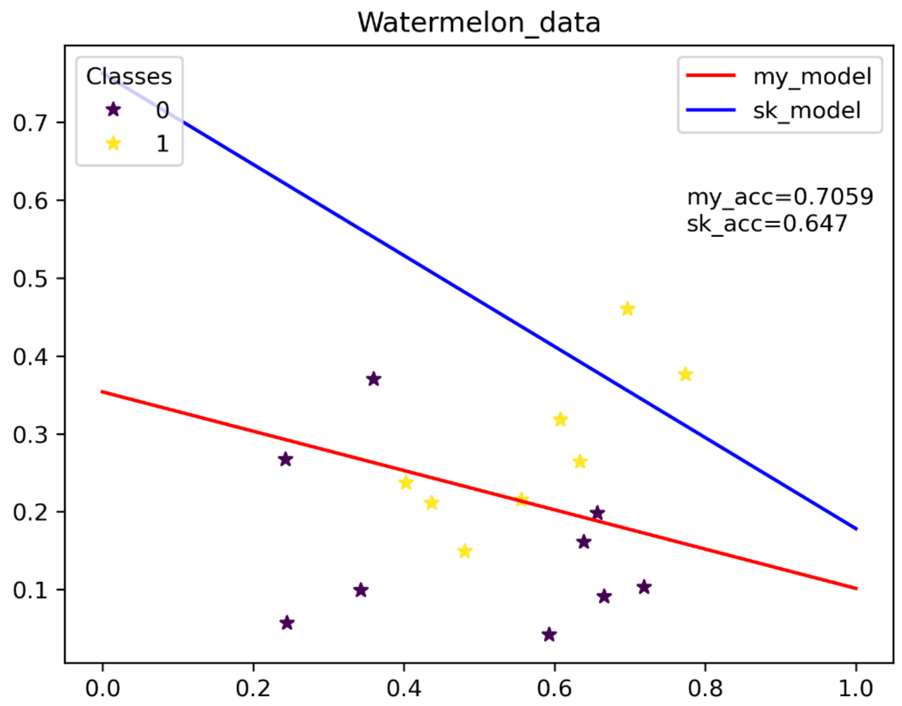
图表 2 my_model 训练结果

```
====================sk model====================
1、模型参数:
    w=[[0.2890631  0.49458092]]
    b=[-0.37718839]
2、评级指标:
acc  = 0.6470588235294118
3、分类结果:
              precision    recall  f1-score   support

           否       0.60      1.00      0.75         9
           是       1.00      0.25      0.40         8

    accuracy                           0.65        17
   macro avg       0.80      0.62      0.57        17
weighted avg       0.79      0.65      0.59        17
```

图表 3 sklearn model 训练结果

**3、分类图**

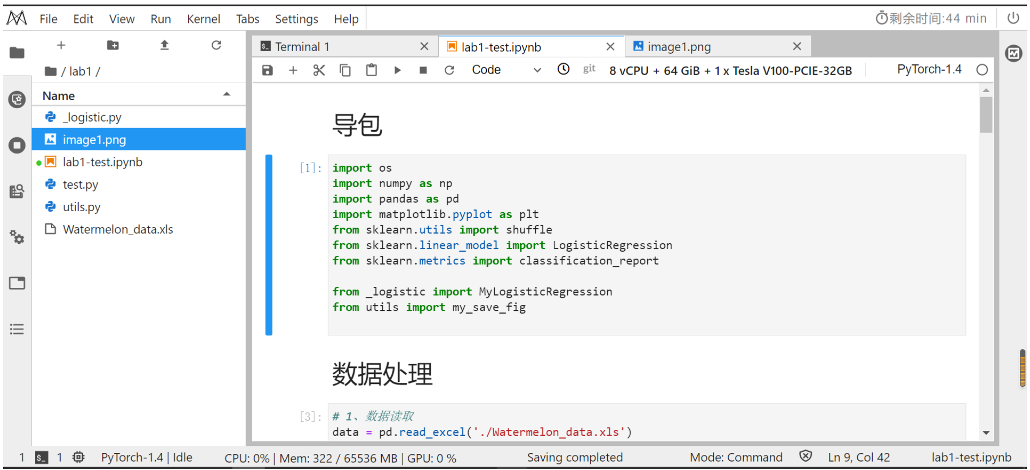
sklearn 模型和 my_model 的分类效果图，如图 4。



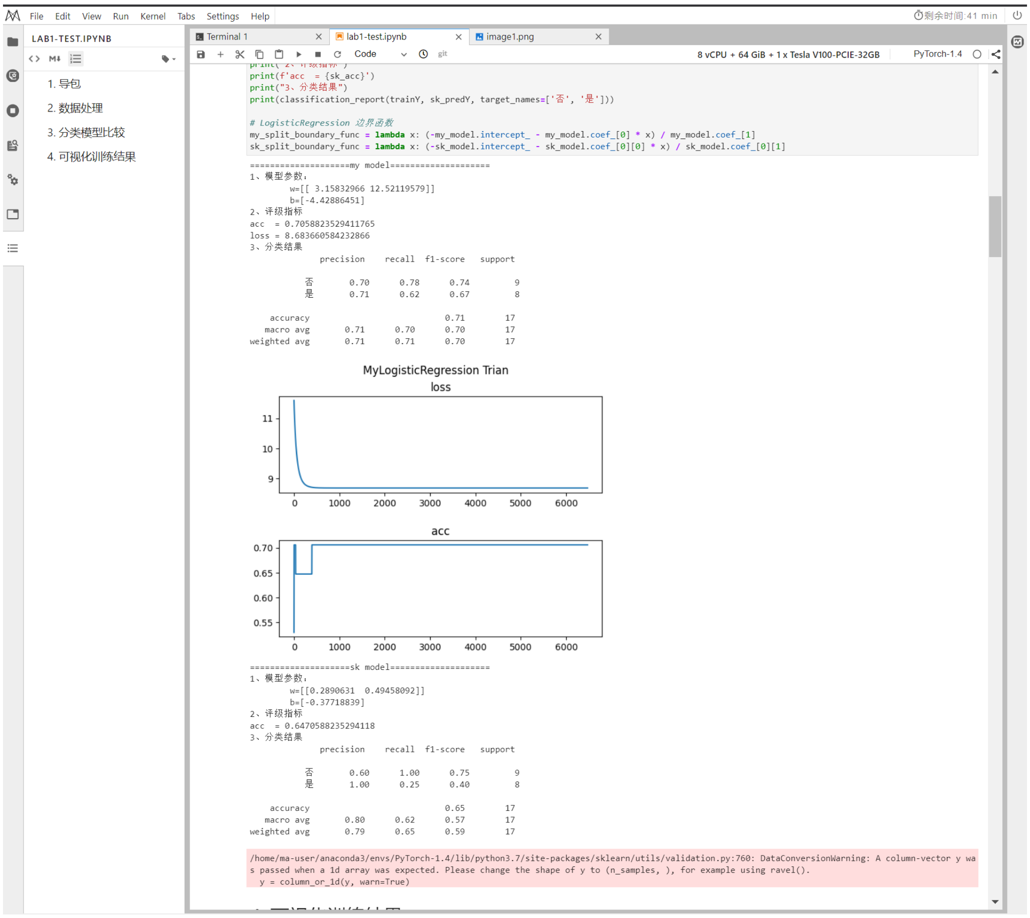

图表 4 西瓜数据集分类示意图

4、华为云实现

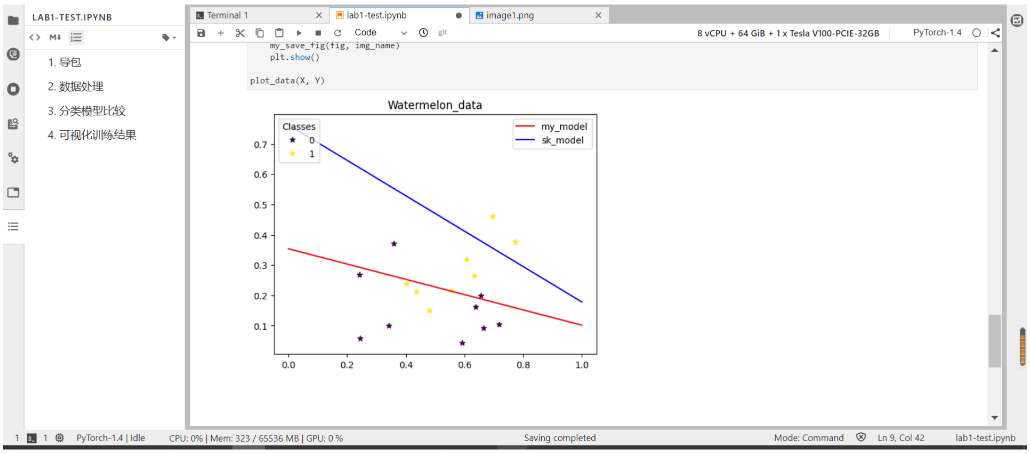

图表 5 obs 桶数据




图表 6 主要上传数据




图表 7 模型比较结果



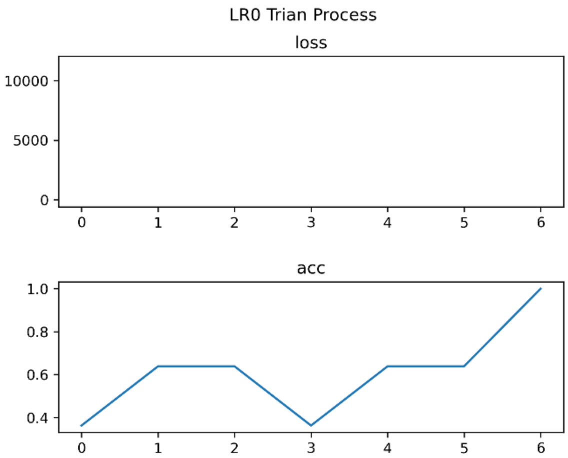

图表 8 模型分类结果示意图

5、iris 数据集分类

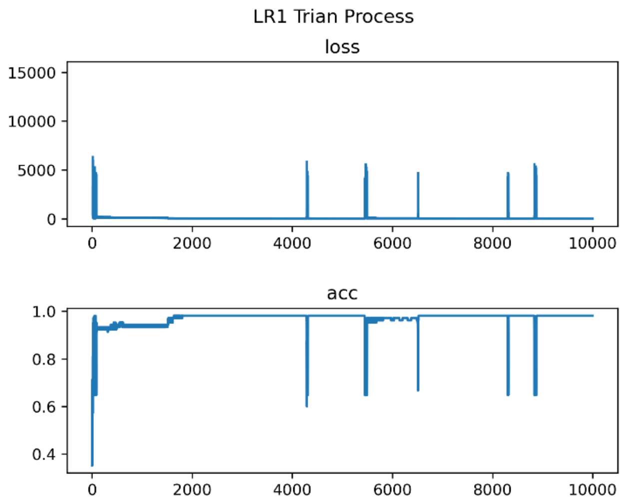
模型在理想情况下，可以得到 98%的准确率。

```
===================模型参数===================
model   coef_               intercept_
0   [-10.042555660529306, -26.69994254195937, 39.58633629830628, 18.953867250587486]    [-4.81928701]
1   [1.5779051543195977, 26.329189685449027, -24.139689288072017, -57.61420694207595]   [130.71495256]
2   [-8.139905625277594, -195.79856951940127, 7.094816151371333, -70.08254603358836]    [542.53863016]
训练准确率:0.980952
测试准确率:1.000000
```

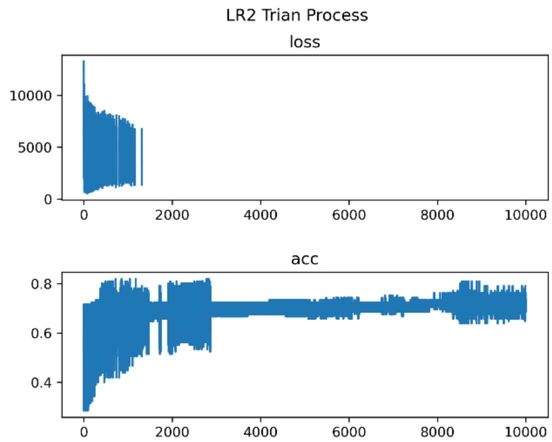
具体训练过程记录如下：




图表 9 LR0 Trian Process




图表 10 LR1 Trian Process




图表 11 LR2 Trian Process