

重庆科技学院学生实验报告

课程名称	机器学习	实验项目名称	Logistic 回归		
开课学院及实验室	智能技术与工程学院		实验日期	9 月 30 日	
学生姓名	刘志贵	学号	2023440221	专业班级	智科 2023-02
指导教师	易军		实验成绩		

一、实验目的和要求

1. 实验目的：实验的核心是利用逻辑回归算法，根据马疝病数据集，构建一个可以预测患病马匹存活情况的二分类模型。实验要求完成数据预处理，算法实现，模型评估与应用。
2. 实验要求：首先需要完成数据集空值填充，然后实现批量梯度算法并可视化它的二维数据集上的决策边界，还要实现带退火学习率的随机梯度上升算法用于核心模型训练。

二、实验内容

1. 数据集分别有：训练集 horseColicTraining.txt 和测试集 horseColicTest.txt，是否存活为标签。以及演示用的 testSet.txt 数据集，用于可视化决策边界。
2. 算法与实现：Logistic 回归的批量梯度上升与随机梯度上升（退火学习率）两种优化，动态调整退火学习率： $\alpha = 4 / (1.0 + j + i) + 0.0001$ 。

三、实验步骤与结果

1. 准备训练集 horseColicTraining.txt 和测试集 horseColicTest.txt，演示数据集 testSet.txt 放在同一个目录下，然后安装本次实验需要的依赖：numpy, matplotlib, scikit-learn 等。本次实验代码放在 logRegres.py 文件中，目录如下：

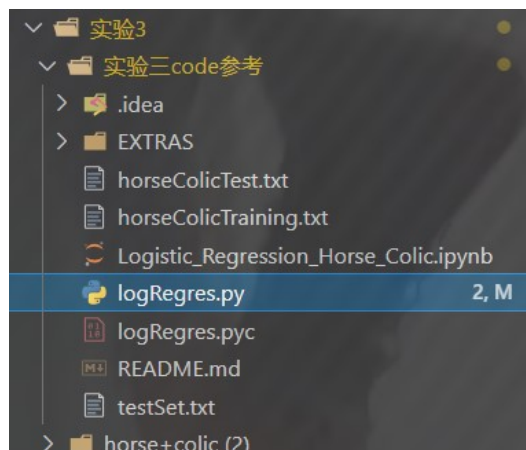
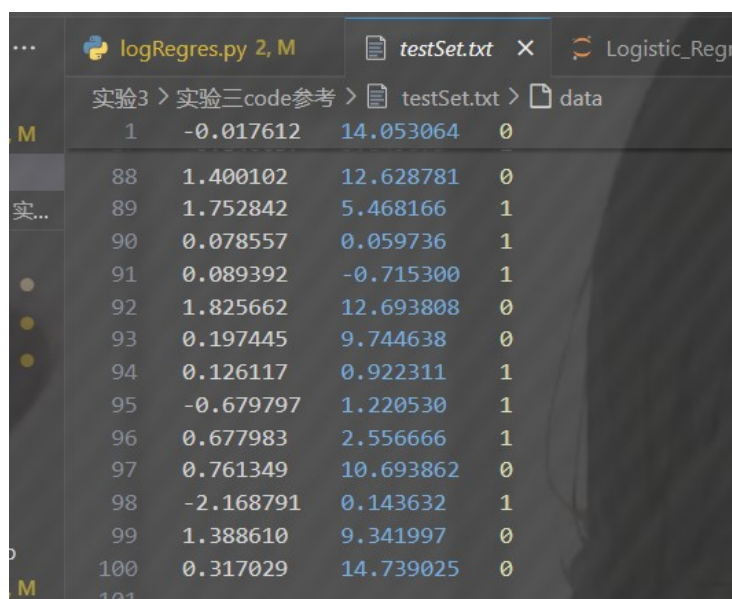


图 1 实验目录

2. 算法原理可视化:

函数: `run_visualization()` 函数。

步骤: 先加载 `testSet.txt` 数据, 数据集如下所示:



Index	Feature 1	Feature 2	Class Label
1	-0.017612	14.053064	0
88	1.400102	12.628781	0
89	1.752842	5.468166	1
90	0.078557	0.059736	1
91	0.089392	-0.715300	1
92	1.825662	12.693808	0
93	0.197445	9.744638	0
94	0.126117	0.922311	1
95	-0.679797	1.220530	1
96	0.677983	2.556666	1
97	0.761349	10.693862	0
98	-2.168791	0.143632	1
99	1.388610	9.341997	0
100	0.317029	14.739025	0

图 2 testSet 数据集图

然后调用 `train_bga` 训练模型, 并使用 `matplotlib` 绘制决策边界, 如下图为结果图, 展示算法线性分类能力:

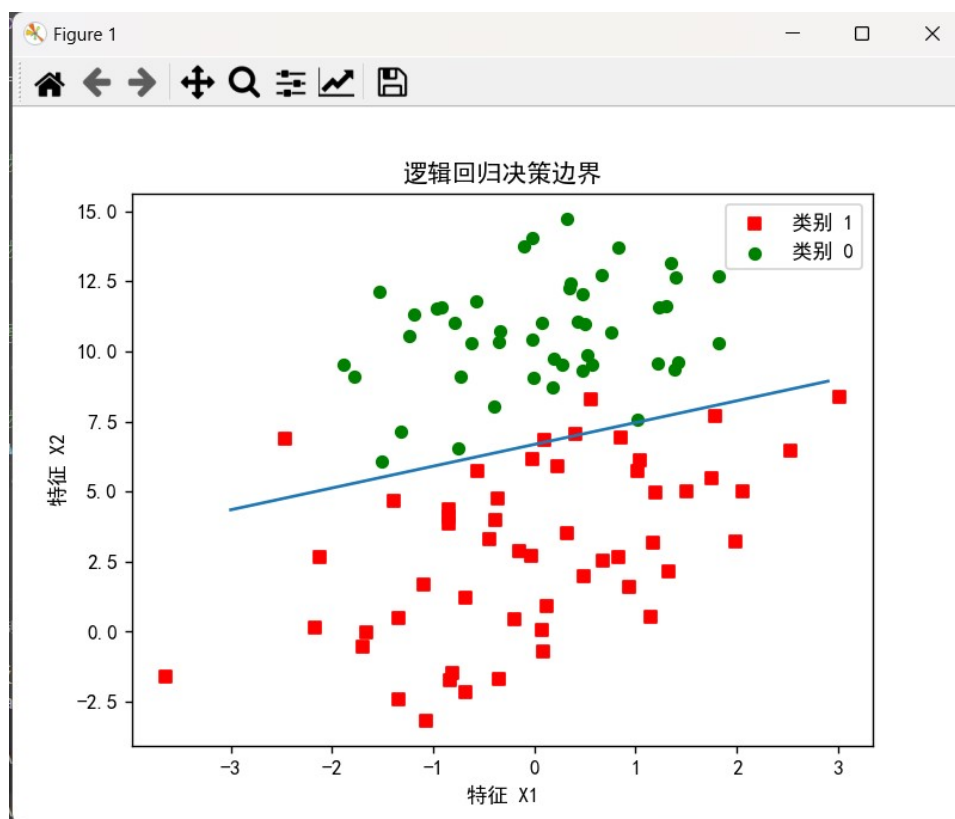


图 3 可视化决策边界

3. 准备数据，训练与测试算法：

函数：run_multi_tests()，内部调用 test_model()。

步骤：test_model 函数加载训练和测试数据，代码如下：

```
def test_model(self, num_iter=1000):
    # 训练并测试模型
    training_set, training_labels = self.load_data('horseColicTraining.txt')
    test_set, test_labels = self.load_data('horseColicTest.txt')

    processed_training_set = self.preprocess_mean_fill(training_set, is_training=True)
    processed_test_set = self.preprocess_mean_fill(test_set, is_training=False)

    weights = self.train_sgd(processed_training_set, training_labels, num_iter)

    error_count = sum(1 for i, vec in enumerate(processed_test_set) if int(self.classify(vec, weights)) != int(test_labels[i]))
```

图 4 加载并测试模型

然后调用 preprocess_mean_fill 对数据进行均值填充，缺值之前已经采用 0 填充数据集，这里将 0 替换成均值，代码如下：

```
def preprocess_mean_fill(self, dataset, is_training=True):
    # 预处理：使用训练集的均值填充缺失值(0)
    feature_matrix = np.array(dataset)
    if is_training:
        num_features = feature_matrix.shape[1]
        means = np.zeros(num_features)
        for i in range(num_features):
            non_zero_vals = feature_matrix[:, i][feature_matrix[:, i] != 0]
            if len(non_zero_vals) > 0:
                means[i] = np.mean(non_zero_vals)
        self.training_means = means

    for i in range(feature_matrix.shape[1]):
        zero_indices = np.where(feature_matrix[:, i] == 0)[0]
        feature_matrix[zero_indices, i] = self.training_means[i]
    return feature_matrix
```

图 5 均值填充函数代码图

预处理完成后开始训练模型，先默认迭代 1000 次，代码如下：

```
def train_sgd(self, data_matrix, class_labels, num_iter=150):
    # 随机梯度上升 (SGD)
    data_matrix = np.array(data_matrix)
    m, n = data_matrix.shape
    weights = np.ones(n)
    for j in range(num_iter):
        data_index = list(range(m))
        for i in range(m):
            alpha = 4 / (1.0 + j + i) + 0.0001
            rand_idx = random.choice(data_index)
            h = self.sigmoid(np.sum(data_matrix[rand_idx] * weights))
            error = class_labels[rand_idx] - h
            weights = weights + alpha * error * data_matrix[rand_idx]
            data_index.remove(rand_idx)
    return weights
```

图 6 随机梯度上升算法训练模型代码

需要注意的是，虽然函数默认 `num_iter=150`，但在调用这个函数的时候，我传入的是更大的值，为 1000 次，因为经过多次测试发现 1000 次迭代的效果明显更优，如下图所示：

```
predictor.run_visualization()  
predictor.run_multi_tests(num_iter=1000)
```

图 7 主函数调用训练函数输入迭代次数图

训练完成后就开始在测试集上评估并返回错误率，我进行了 10 测试取平均错误率的操作，这样可以尽量避免偶然，如下图为终端结果：

```
马疝病预测 (均值填充法)...\n第 1 次测试错误率: 0.2239\n第 2 次测试错误率: 0.2239\n第 3 次测试错误率: 0.2687\n第 4 次测试错误率: 0.3134\n第 5 次测试错误率: 0.2537\n第 6 次测试错误率: 0.2537\n第 7 次测试错误率: 0.2388\n第 8 次测试错误率: 0.2388\n第 9 次测试错误率: 0.2687\n第 10 次测试错误率: 0.1791\n\n10次测试平均错误率: 0.2463
```

图 7 测试数据错误率

其中使用的 `run_multi_tests` 函数测试 10 次并且计算平均错误率，代码如下：

```
def run_multi_tests(self, num_tests=10, num_iter=1000):  
    # 多次测试，打印每次的错误率，并计算平均值  
    print("马疝病预测 (均值填充法)...\n")  
    error_rates = []  
    for i in range(num_tests):  
        error_rate, _ = self.test_model(num_iter)  
        print(f"第 {i+1} 次测试错误率: {error_rate:.4f}")  
        error_rates.append(error_rate)  
  
    avg_error = np.mean(error_rates)  
    print(f"\n{num_tests}次测试平均错误率: {avg_error:.4f}\n")
```

图 8 测试错误率计算代码图

4. 与线性回归对比：

函数：`compare_with_linear_reg()`

步骤：使用相同均值填充数据集训练一个线性回归模型，然后计算模型在测试集上的分类错误率，代码如下：


```
def compare_with_linear_reg(self):
    # 与线性回归对比
    print("模型对比 (vs. 线性回归)...")
    training_set, training_labels = self.load_data('horseColicTraining.txt')
    test_set, test_labels = self.load_data('horseColicTest.txt')

    X_train = self.preprocess_mean_fill(training_set, is_training=True)
    X_test = self.preprocess_mean_fill(test_set, is_training=False)
    y_train = np.array(training_labels)

    X_b = np.c_[np.ones((X_train.shape[0], 1)), X_train]
    try:
        weights = np.linalg.inv(X_b.T @ X_b) @ X_b.T @ y_train
    except np.linalg.LinAlgError:
        print("线性回归失败：矩阵为奇异矩阵。")
        return

    X_test_b = np.c_[np.ones((X_test.shape[0], 1)), X_test]
    predictions = X_test_b @ weights

    error_count = sum(1 for i, p in enumerate(predictions) if (1 if p > 0.5 else 0) != test_labels[i])
    error_rate = error_count / len(test_labels)
    print(f"线性回归错误率: {error_rate:.4f}\n")
```

图 9 线性回归训练模型对比代码图

终端运行结果如下：

```
模型对比 (vs. 线性回归)...
线性回归错误率: 0.2537
```

图 10 线性回归模型错误率图

5. 命令行预测：

函数：run_cli_predictor()

步骤：主程序完成评估后，重新训练了一个最终模型获得权重，然后调用这个函数启动交互式预测，输入 0 值也会被均值填充逻辑处理。

我采用 test 数据集里面第 7 和第 8 行的数据进行测试，数据如下：

```
7 2 1 38.40 80 60 3 2 2 1 3 2 1 2 2 0 1 1 54.00 6.90 0 0 1
8 2 1 37.80 48 12 2 1 2 1 3 0 1 2 0 0 2 0 48.00 7.30 1 0 1
```

图 11 命令行测试数据

终端显示如下：

```
预测结果：存活
> 2 1 38.40 80 60 3 2 2 1 3 2 1 2 2 0 1 1 54.00 6.90 0 0
预测结果：存活
> 2 1 37.80 48 12 2 1 2 1 3 0 1 2 0 0 2 0 48.00 7.30 1 0 1
预测结果：存活
```

图 12 命令行终端运行结果

结果：终端显示的结果和 test 数据集的结果一样，说明交互式的预测效果较好。

四、实验结果

1. 逻辑回归模型性能:

采用均值填充预处理后, 模型在测试集上的平均错误率有 24.63%。

2. 线性回归模型性能:

用相同的数据集线性回归模型的错误率有 25.37%。

3. 可视化:

开始就利用数据集实现了算法原理的决策边界图, 最后还实现了一个命令行的交互预测工具。

五、模型评价

1. 虽然数据集经过了 0 填充预处理, 但还是采用了均值填充替换 0, 这样更加贴合实际, 最后模型有约 75.37% 的准确率。这个结果要比 0 填充好一点, 0 填充法准确率在 67% 左右, 说明合适的预处理方式对提升模型性能有一点效果。

2. 在模型对比上, 逻辑回归的错误率在 24.63%, 线性回归在 25.37%, 为了避免误差, 我进行了多次测试, 结果都显示逻辑回归比线性回归的错误率更低一点, 说明这个数据用逻辑回归模型的效果更好。

附件

```
import numpy as np
import matplotlib.pyplot as plt
import random

class HorseColicPredictor:
    # 一个封装了逻辑回归实验全过程的类。
    # 遵循 PDF 文档要求, 包括均值填充预处理, 并简化了终端输出。
    def __init__(self):
        # 初始化, 用于存储训练集的特征均值
        self.training_means = None

    @staticmethod
    def sigmoid(inX):
        # Sigmoid 函数
        inX = np.clip(inX, -500, 500)
        return 1.0 / (1 + np.exp(-inX))
```

```

def train_bga(self, data_matrix, class_labels, alpha=0.001, max_cycles=500):
    # 批量梯度上升 (BGA), 用于可视化
    data_matrix = np.array(data_matrix)
    label_mat = np.array(class_labels).reshape(-1, 1)
    weights = np.ones((data_matrix.shape[1], 1))
    for _ in range(max_cycles):
        h = self.sigmoid(data_matrix @ weights)
        error = label_mat - h
        weights = weights + alpha * data_matrix.T @ error
    return weights

def train_sgd(self, data_matrix, class_labels, num_iter=150):
    # 随机梯度上升 (SGD), 用于主模型训练
    data_matrix = np.array(data_matrix)
    m, n = data_matrix.shape
    weights = np.ones(n)
    for j in range(num_iter):
        data_index = list(range(m))
        for i in range(m):
            # 动态学习率
            alpha = 4 / (1.0 + j + i) + 0.0001
            rand_idx = random.choice(data_index)
            h = self.sigmoid(np.sum(data_matrix[rand_idx] * weights))
            error = class_labels[rand_idx] - h
            weights = weights + alpha * error * data_matrix[rand_idx]
            data_index.remove(rand_idx)
    return weights

def classify(self, inX, weights):
    # 分类函数
    prob = self.sigmoid(np.sum(inX * weights))
    return 1.0 if prob > 0.5 else 0.0

def load_data(self, filename, is_simple_data=False):
    # 通用数据加载函数
    feature_set, label_set = [], []
    with open(filename) as f:
        for line in f.readlines():

```

```

        parts = line.strip().split()
        if is_simple_data:
            feature_set.append([1.0, float(parts[0]), float(parts[1])])
            label_set.append(int(parts[2]))
        else:
            feature_set.append([float(val) for val in parts[:-1]])
            label_set.append(float(parts[-1]))
    return feature_set, label_set

def preprocess_mean_fill(self, dataset, is_training=True):
    # 预处理：使用训练集的均值填充缺失值(0)
    feature_matrix = np.array(dataset)
    if is_training:
        num_features = feature_matrix.shape[1]
        means = np.zeros(num_features)
        for i in range(num_features):
            non_zero_vals = feature_matrix[:, i][feature_matrix[:, i] != 0]
            if len(non_zero_vals) > 0:
                means[i] = np.mean(non_zero_vals)
        self.training_means = means

    for i in range(feature_matrix.shape[1]):
        zero_indices = np.where(feature_matrix[:, i] == 0)[0]
        feature_matrix[zero_indices, i] = self.training_means[i]
    return feature_matrix

def run_visualization(self):
    # 分析数据 - 可视化
    print("算法原理可视化...")
    data_arr, label_mat = self.load_data('testSet.txt', is_simple_data=True)
    weights = self.train_bga(data_arr, label_mat)

    fig = plt.figure()
    ax = fig.add_subplot(111)
    ax.scatter(np.array(data_arr)[np.array(label_mat)==1,
np.array(data_arr)[np.array(label_mat)==1, 2], s=30, c='red', marker='s', label='类别 1')
ax.scatter(np.array(data_arr)[np.array(label_mat)==0,
np.array(data_arr)[np.array(label_mat)==0, 2], s=30, c='green', label='类别 0')

```



```

x = np.arange(-3.0, 3.0, 0.1)
y = (-weights[0] - weights[1] * x) / weights[2]
ax.plot(x, y.flatten())
plt.rcParams['font.sans-serif'] = ['SimHei']
plt.rcParams['axes.unicode_minus'] = False
plt.title('逻辑回归决策边界'); plt.xlabel('特征 X1'); plt.ylabel('特征 X2')
plt.legend(); plt.show()
print("可视化完成。 \n")

def test_model(self, num_iter=1000):
    # 训练并测试模型
    training_set, training_labels = self.load_data('horseColicTraining.txt')
    test_set, test_labels = self.load_data('horseColicTest.txt')

    processed_training_set = self.preprocess_mean_fill(training_set, is_training=True)
    processed_test_set = self.preprocess_mean_fill(test_set, is_training=False)

    weights = self.train_sgd(processed_training_set, training_labels, num_iter)

    error_count = sum(1 for i, vec in enumerate(processed_test_set) if
int(self.classify(vec, weights)) != int(test_labels[i]))

    return float(error_count) / len(test_labels), weights

def run_multi_tests(self, num_tests=10, num_iter=1000):
    # 多次测试，打印每次的错误率，并计算平均值
    print("马疝病预测 (均值填充法)...")
    error_rates = []
    for i in range(num_tests):
        error_rate, _ = self.test_model(num_iter)
        print(f"第 {i+1} 次测试错误率: {error_rate:.4f}")
        error_rates.append(error_rate)

    avg_error = np.mean(error_rates)
    print(f"\n{num_tests} 次测试平均错误率: {avg_error:.4f}\n")

def compare_with_linear_reg(self):

```

```

# 与线性回归对比
print("模型对比 (vs. 线性回归)...")
training_set, training_labels = self.load_data('horseColicTraining.txt')
test_set, test_labels = self.load_data('horseColicTest.txt')

X_train = self.preprocess_mean_fill(training_set, is_training=True)
X_test = self.preprocess_mean_fill(test_set, is_training=False)
y_train = np.array(training_labels)

X_b = np.c_[np.ones((X_train.shape[0], 1)), X_train]
try:
    weights = np.linalg.inv(X_b.T @ X_b) @ X_b.T @ y_train
except np.linalg.LinAlgError:
    print("线性回归失败: 矩阵为奇异矩阵。")
    return

X_test_b = np.c_[np.ones((X_test.shape[0], 1)), X_test]
predictions = X_test_b @ weights

error_count = sum(1 for i, p in enumerate(predictions) if (1 if p > 0.5 else 0) !=
test_labels[i])
error_rate = error_count / len(test_labels)
print(f"线性回归错误率: {error_rate:.4f}\n")

def run_cli_predictor(self, weights):
    # 使用算法 - 命令行预测工具
    print("命令行预测...")
    print("输入 21 项指标:")

    while True:
        line = input("> ")
        if line.lower() == 'quit':
            print("已退出。"); break

        try:
            features = np.array([float(p) for p in line.strip().split()])
            if len(features) != 21:

```

```

        print(f'错误: 需要输入 21 个指标, 您输入了 {len(features)} 个。')
        continue

    processed_features = self.preprocess_mean_fill([features],
is_training=False)
    prediction = self.classify(processed_features[0], weights)
    print(f'预测结果: {'死亡' if prediction == 1.0 else '存活'}")

except ValueError: print("错误: 请确保所有输入都是有效的数字。")

if __name__ == '__main__':
    predictor = HorseColicPredictor()
    predictor.run_visualization()
    predictor.run_multi_tests(num_iter=1000)
    predictor.compare_with_linear_reg()
    _, final_weights = predictor.test_model(num_iter=1000)
    predictor.run_cli_predictor(final_weights)

```