目 录

1. 题目 1
1.1. 实验目的 1
1.2. 实验要求 1
1.3. 实验内容 1
1.4. 实验数据 1
2. 算法阐述或实验步骤说明 2
2.1. Softmax Regression 算法阐述 2
2.1.1. 假设函数 2
2.1.2. 代价函数 2
2.1.3. 求解 2
2.1.4. 正则化 3
2.2. 实验步骤说明 3
3. 实验结果与截图 7
4. 总结 9
5. 参考文献 9

1. 题目

1.1. 实验目的

通过用 Softmax Regression 算法对鸢尾花数据进行分类，帮助学生掌握 Softmax Regression 算法的原理和使用场景、掌握导入数据、预处理数据、模型训练和可视化方法，使研究生们尽快熟悉人工智能、机器学习与大数据领域的基础算法。

1.2. 实验要求

(1) 理解算法本身，用程序设计语言(不限语言，但建议选择 Python、C、C++或 Java)实现算法。

(2) 除了数学运算(例如求函数值、矩阵求逆等)可以调用现有的库函数，其他一律不准调库。

1.3. 实验内容

Iris 也称鸢尾花卉数据集[1]，是常用的分类实验数据集，由 R. A. Fisher 于 1936 年收集整理的。其中包含 3 种植物种类，分别是山鸢尾(setosa)、变色鸢尾(versicolor)和维吉尼亚鸢尾(virginica)，每类 50 个样本，共 150 个样本。该数据集包含 4 个特征变量，1 个类别变量。iris 每个样本都包含了 4 个特征：花萼长度，花萼宽度，花瓣长度，花瓣宽度，以及 1 个类别变量(label)。

本实验需要通过自行实现 Softmax Regression 算法，通过利用数据集中的特征，建立鸢尾花分类器，预测鸢尾花的分类。实现并掌握导入数据、预处理数据、模型训练和可视化方法。

1.4. 实验数据

本实验所用数据集是鸢尾花卉数据集，来源为 sklearn.datasets 中的 iris 数据集。iris 数据集有 150 个样本数据，分为 3 个类别，分别用整数表示：0(Setosa)，1(Versicolour)，2(Virginica)。每个样本包含 4 个特征变量：花萼长度(sepal_length)、花萼宽度(sepal_width)、花瓣长度(petal_length)、花瓣宽度(petal_width)。

在实验中，使用数据集中的所有特征变量，实现山鸢尾(Setosa)、变色鸢尾(Versicolor)和维吉尼亚鸢尾(Virginica)的多分类。

2. 算法阐述或实验步骤说明

2.1. Softmax Regression 算法阐述

2.1.1. 假设函数

Softmax 回归[2]是逻辑回归在多分类问题上的推广。假设有 m 个训练样本，对应训练数据集 $\{(x^{(1)},y^{(1)}),(x^{(2)},y^{(2)}),...,(x^{(m)},y^{(m)})\}$。对于 Softmax 回归，每一个输入特征 $x^{(i)}\in\mathbf{R}^{n+1}$，对应类标记为 $y^{(i)}\in\{0,1,...,k\}$，假设函数可以表示为公式(2-1)。

$$h_\theta(x^{(i)})=\begin{bmatrix}p(y^{(i)}=1\mid x^{(i)},\theta)\\p(y^{(i)}=2\mid x^{(i)},\theta)\\\vdots\\p(y^{(i)}=k\mid x^{(i)},\theta)\end{bmatrix}=\frac{1}{\sum_{j=1}^{k}e^{\theta_j^T x^{(i)}}}\begin{bmatrix}e^{\theta_1^T x^{(i)}}\\e^{\theta_2^T x^{(i)}}\\\vdots\\e^{\theta_k^T x^{(i)}}\end{bmatrix}. \tag{2-1}$$

其中，$\theta_1,\theta_2,...,\theta_k\in\mathbf{R}^{n+1}$ 为模型的参数，对于每一个样本估计其所属的类别的概率可以表示为公式(2-2).

$$p(y^{(i)}=j\mid x^{(i)},\theta)=\frac{e^{\theta_j^T x^{(i)}}}{\sum_{l=1}^{k}e^{\theta_l^T x^{(i)}}}. \tag{2-2}$$

2.1.2. 代价函数

与逻辑回归算法引入交叉熵损失函数相似，在 Softmax 回归的代价函数中引入指示函数 $I(y^{(i)}=j)$，如公式(2-3)所示。

$$I(y^{(i)}=j)=\begin{cases}1 & \text{if } y^{(i)}=j \text{ is True}\\0 & \text{if } y^{(i)}=j \text{ is False}\end{cases}. \tag{2-3}$$

那么，Softmax 回归的代价函数可以表示为公式(2-4)。

$$J(\theta)=-\frac{1}{m}[\sum_{i=1}^{m}\sum_{j=1}^{k}I(y^{(i)}=j)\log\frac{e^{\theta_j^T x^{(i)}}}{\sum_{l=1}^{k}e^{\theta_l^T x^{(i)}}}]. \tag{2-4}$$

2.1.3. 求解

对上述代价函数的优化，即为找到一组 $\theta_1,\theta_2,...,\theta_k\in\mathbf{R}^{n+1}$ 的最优解使公式(2-4)的值最小。我们常见的方法即为使用梯度下降算法求解，即 $\frac{\partial}{\partial\theta_j}J(\theta)$，如公式

(2-5)。

$$\frac{\partial}{\partial\theta_j}J(\theta)=-\frac{1}{m}\sum_{i=1}^{m}[x^{(i)}I(y^{(i)}=j)-p(y^{(i)}=j\mid x^{(i)},\theta)]. \tag{2-5}$$

2.1.4. 正则化

由于 Softmax 回归模型被过度参数化[3]，模型中存在冗余的参数，因此我们考虑加入正则项，通过权重衰减的办法解决参数冗余的情况。此时代价函数如公式(2-6)所示。

$$J(\theta)=-\frac{1}{m}[\sum_{i=1}^{m}\sum_{j=1}^{k}I(y^{(i)}=j)\log\frac{e^{\theta_j^T x^{(i)}}}{\sum_{l=1}^{k}e^{\theta_l^T x^{(i)}}}]+\frac{\lambda}{2}\sum_{i=1}^{k}\sum_{j=0}^{n}\theta_{ij}^2. \tag{2-6}$$

对该函数进行梯度下降算法时的求导结果为公式(2-7)所示。

$$\frac{\partial}{\partial\theta_j}J(\theta)=-\frac{1}{m}\sum_{i=1}^{m}[x^{(i)}I(y^{(i)}=j)-p(y^{(i)}=j\mid x^{(i)},\theta)]+\lambda\theta_j. \tag{2-7}$$

2.2. 实验步骤说明

(1) 搭建实验环境

① 操作系统：Windows10 64 位，CPU 为 Intel(R) Core(TM) i7-8550U；

② 开发语言及软件：Python，PyCharm 2021.3.2 (Community Edition)；

③ 编译器及软件包版本：Python 3.6，scikit-learn 0.19，seaborn 0.8；

(2) 引入库函数

```
from definition import *
from sklearn.datasets import load_iris
from sklearn.model_selection import train_test_split
from sklearn import metrics
import numpy as np
import seaborn as sns
import matplotlib.pyplot as plt
```

(3) 数据载入

```
# test
# load data
iris = load_iris()
x = iris.data
y = iris.target  # y in {0, 1, 2}
feature_names = iris.feature_names
```

(4) 可视化描述数据信息

```
# 输出数据集散点分布图
```

```
plt.scatter(x[y == 0, 1], x[y == 0, 2], c='r', label="Setosa")
plt.scatter(x[y == 1, 1], x[y == 1, 2], c='b', label="Versicolor")
plt.scatter(x[y == 2, 1], x[y == 2, 2], c='g', label="Virginica")
plt.xlabel(feature_names[1])
plt.ylabel(feature_names[2])
plt.legend()
plt.show()
```

(5) 数据处理

将载入数据样本按照 $\frac{\text{训练集数据量}}{\text{测试集数据量}}=1:1$ 的比率进行随机选取。

```
# 将数据集按一定比例分为训练集和测试集
x_train, x_test, y_train, y_test = train_test_split(x, y, test_size=0.5,
random_state=2048)
y_label = y_train.reshape((len(y_train), 1))
```

(6) 模型训练

① 初始化 Softmax 回归模型

```
def __init__(self, iteration, learning_rate, weights=None):
    self.iteration = iteration
    self.learning_rate = learning_rate
    self.weights = weights
```

② 定义代价函数

```
def softmaxLoss(self, error, y):
    num_train = np.shape(error)[0]
    cost = 0
    for i in range(num_train):
        if error[i, y[i, 0]] / np.sum(error[i, :]) > 0:
            cost += (-1) * np.log(error[i, y[i, 0]] / np.sum(error[i, :]))
    return cost / num_train
```

③ 定义模型训练过程

```
# 使用梯度下降算法进行训练
def gradientDescent(self, x, y, num_type):
    num_train, num_feature = np.shape(x)
    self.weights = np.mat(np.ones((num_feature, num_type)))
    for i in range(self.iteration):
        error = np.exp(x * self.weights)
        if i % 1000 == 0:
            print(
                "Iteration: " + str(i) + "\tCost: " +
                str(self.softmaxLoss(error, y)))
        row_sum = -error.sum(axis=1)
```

```
        row_sum = row_sum.repeat(num_type, axis=1)
        error = error / row_sum
        for j in range(num_train):
            error[j, y[j, 0]] += 1
        self.weights = self.weights +
                        self.learning_rate * x.T * error / num_train
    return self.weights
```

④ 定义输出训练数据集预测结果

```
def softmaxTraining(self, x, weights):
    train_result = x * weights
    train_result = train_result.argmax(axis=1)
    return train_result.flatten()
```

⑤ 定义输出测试数据集预测结果

```
def softmaxPrediction(self, x, weights):
    predict_result = x * weights
    predict_result = predict_result.argmax(axis=1)
    return predict_result.flatten()
```

(7) 进行数据测试

```
# 使用s o f t m a x  r e g r e s s i o n算法进行训练
SoftRe = SoftmaxRegression(iteration=10000, learning_rate=0.1)
weights = SoftRe.gradientDescent(x_train, y_label, num_type=3)
y_prediction_train = SoftRe.softmaxTraining(x_train, weights)
y_prediction_test = SoftRe.softmaxPrediction(x_test, weights)
```

(8) 输出结果并进行可视化描述

```
print("=================== 训练结果 ===================")
print()
print("Weights List: \n" + str(weights))
print("Train Label: \n" + str(y_train))
print("Train Prediction: \n" + str(y_prediction_train))
# 查看混淆矩阵，绘制热力图
confusion_matrix_test =
metrics.confusion_matrix(y_prediction_train.tolist()[0], y_train)
plt.figure(figsize=(8, 6))
sns.heatmap(confusion_matrix_test, annot=True, cmap="Pastel2",
annot_kws={'size':14})
# 支持中文
plt.rcParams['font.sans-serif'] = ['SimHei']  # 用来正常显示中文标签
plt.rcParams['axes.unicode_minus'] = False  # 用来正常显示负号
plt.xlabel("预测品种", fontsize=14)
plt.ylabel("实际品种", fontsize=14)
plt.xticks(fontsize=14)
plt.yticks(fontsize=14)
```

```
plt.title("使用热力图查看训练结果的混淆矩阵", fontsize = 14)
plt.show()
print()
print()
print("==================== 测试结果 ====================")
print()
print("Test Label: \n" + str(y_test))
print("Test Prediction: \n" + str(y_prediction_test))
# 查看混淆矩阵，绘制热力图
confusion_matrix_test =
metrics.confusion_matrix(y_prediction_test.tolist()[0], y_test)
plt.figure(figsize=(8, 6))
sns.heatmap(confusion_matrix_test, annot=True, cmap="Pastel2",
annot_kws={'size':14})
# 支持中文
plt.rcParams['font.sans-serif'] = ['SimHei']  # 用来正常显示中文标签
plt.rcParams['axes.unicode_minus'] = False  # 用来正常显示负号
plt.xlabel("预测品种", fontsize=14)
plt.ylabel("实际品种", fontsize=14)
plt.xticks(fontsize=14)
plt.yticks(fontsize=14)
plt.title("使用热力图查看测试结果的混淆矩阵", fontsize = 14)
plt.show()
```

3. 实验结果与截图

(1) 载入数据后，对数据进行可视化描述信息，如图 3-1 所示。

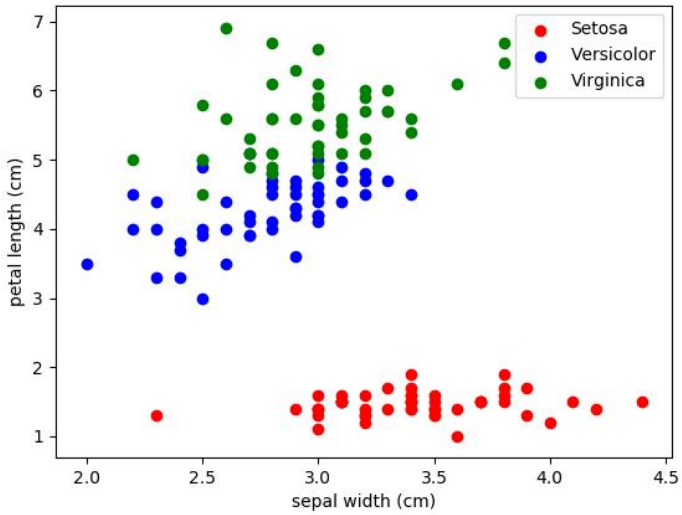

图 3-1 载入数据集中数据信息分布散点图

(2) 使用 Softmax 回归对训练数据集进行训练，如图 3-2 所示。

```
Run:  run_this
E:\Anaconda\envs\MachineLearning\python.exe F:/Workspaces/PythonProject/Machine
Iteration: 0    Cost: 1.09861228867
Iteration: 1000 Cost: 0.124859796393
Iteration: 2000 Cost: 0.100437378846
Iteration: 3000 Cost: 0.0919434941347
Iteration: 4000 Cost: 0.087761882357
Iteration: 5000 Cost: 0.0853432220599
Iteration: 6000 Cost: 0.0838056487232
Iteration: 7000 Cost: 0.0827649473146
Iteration: 8000 Cost: 0.0820280888984
Iteration: 9000 Cost: 0.0814881848489
```

图 3-2 训练过程中迭代次数与对应 Loss 值

(3) 输出训练结果，即参数θ的值以及模型在训练集上的预测结果，如图 3-3 所示。

```
=================== 训练结果 ===================

Weights List:
[[ 2.73649036  3.00574983 -2.74224018]
 [ 4.4988528   0.29569004 -1.79454284]
 [-3.88599977  0.18162888  6.7043709 ]
 [-1.27305241 -0.85442697  5.12747938]]
Train Label:
[1 0 1 1 1 2 0 0 2 0 0 2 1 0 0 2 2 0 0 2 0 0 1 2 1 2 0 1 1 1 2 1 1 0 1 0 2
 1 2 2 2 0 1 2 2 0 1 1 2 2 0 0 1 0 1 2 2 1 2 2 1 0 0 0 0 1 0 2 2 1 0 1 0 1
 1]
Train Prediction:
[[1 0 1 1 1 2 0 0 2 0 0 2 1 0 0 2 2 0 0 2 0 0 1 2 1 2 0 1 1 1 2 1 1 0 1 0 2
  1 2 2 2 0 1 2 2 0 1 1 2 2 0 0 1 0 1 2 2 1 2 2 2 0 0 0 0 1 0 2 1 1 0 1 0 1
  1]]
```

图 3-3 训练集预测结果输出

(4) 借助混淆矩阵绘制模型在训练集上的预测结果的热力图，如图 3-4 所示。

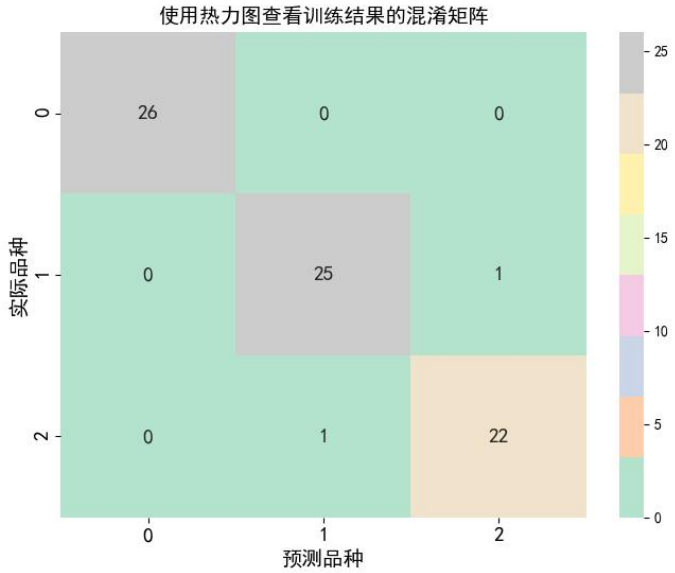

图 3-4 训练集预测结果热力图

(5) 使用测试及对训练出的模型进行测试，输出测试结果，即模型在测试集上的预测结果，如图 3-5 所示。

```
=================== 测试结果 ===================

Test Label:
[1 1 1 1 2 2 2 0 2 0 1 2 0 2 0 2 1 2 0 1 1 0 2 2 0 1 0 1 1 0 0 1 2 1 0 0 1
 2 0 0 1 2 0 0 2 1 0 0 1 0 0 1 2 2 1 2 2 2 2 2 2 1 0 2 0 2 2 1 1 1 1 0 0 2 2
 2]
Test Prediction:
[[1 1 1 1 2 2 2 0 2 0 1 2 0 2 0 2 2 2 2 0 1 1 0 2 2 2 0 1 0 1 1 0 0 1 2 1 0 0 1
  2 0 0 2 2 0 0 2 1 0 0 1 0 0 1 2 2 1 2 2 2 2 2 2 1 0 2 0 2 2 1 1 1 1 0 0 2 2
  2]]

Process finished with exit code 0
```

图 3-5 测试集预测结果输出

(6) 借助混淆矩阵绘制模型在测试集上的预测结果的热力图，如图 3-6 所示。

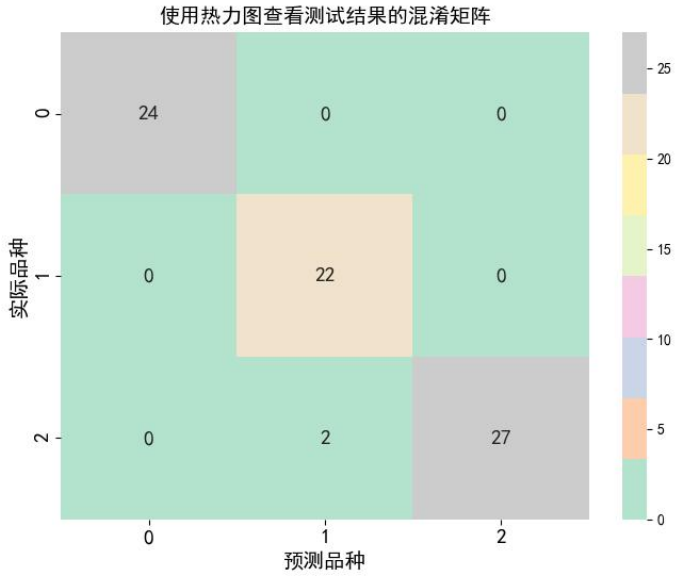

图 3-6 测试集预测结果热力图

4. 总结

通过实验我们可以发现，逻辑回归算法和 Softmax 回归算法极为相似，在 Softmax 回归算法中，当$k=2$时就简化成了逻辑回归算法。虽然逻辑回归算法可以通过“one vs. all”的思想解决多分类，但相比于 Softmax 回归算法来说，Softmax 回归可以解决k个互斥的类别之间的分类，而逻辑回归在解决这样的问题上效率明显不如 Softmax 回归算法。

5. 参考文献

[1] R. A. Fisher. Iris Data Set[DB/OL]. (1936)[2022-07-08]. http://archive.ics.uci.edu/ml/datasets/Iris.

[2] Wikipedia. Softmax function[EB/OL]. (2022-08-05)[2022-08-06]. https://en.wikipedia.org/wiki/Softmax_function

[3] Tz28. softmax 回归[EB/OL]. (2017-05-15)[2022-08-06]. https://blog.csdn.net/u012328159/article/details/72155874?spm=1001.2014.3001.5502