模式识别与统计学习实验四：Softmax Regression

张力群 2020080301001

一、 实验原理

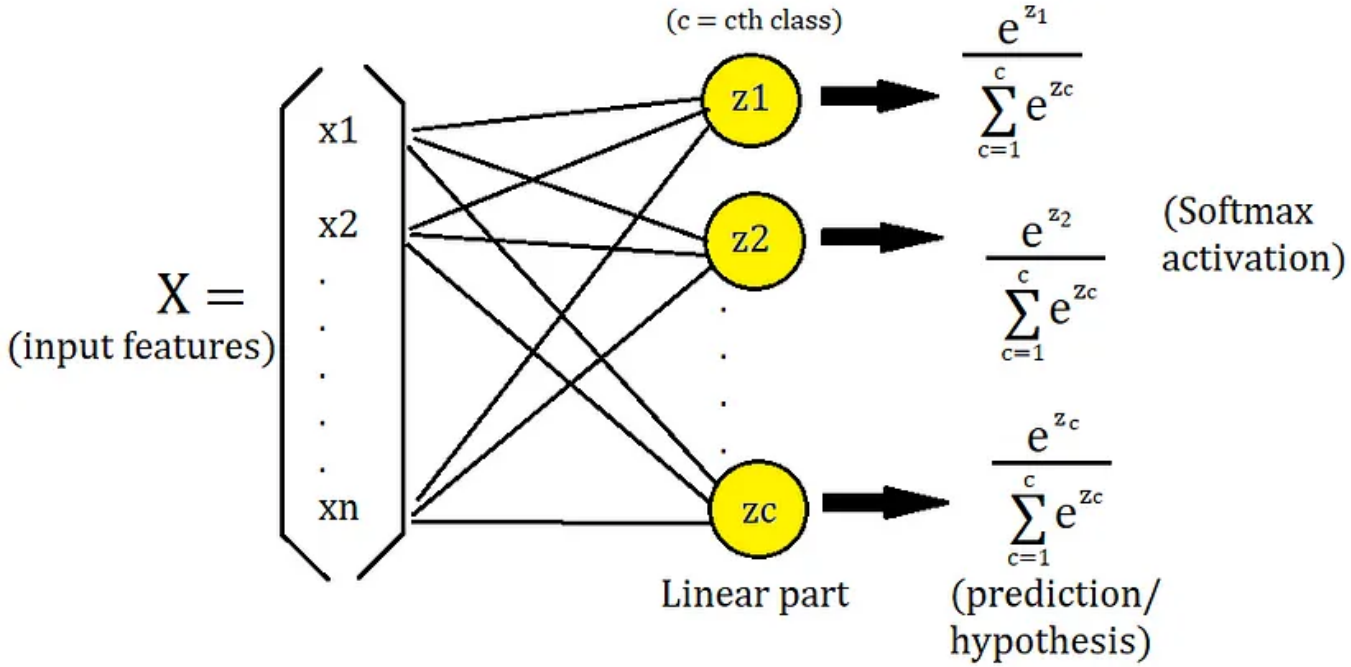
1.1 问题描述

Softmax回归是一种常用的分类算法，适用于多分类问题。在本次实验中，我们将使用逻辑回归算法来识别MNIST 数据集中的数字 0 到 9。

MNIST 数据集是一个手写数字识别数据集，包含了 60,000 张训练图像和 10,000 张测试图像。每张图像都是 28 像素 x 28 像素的灰度图像，像素值的范围是 0 到 255。我们将使用该数据集来训练和测试逻辑回归模型。

1.2 Softmax回归原理

softmax回归是一种常见的分类算法，用于将样本分到多个类别中。给定一个输入特征x和一个权重矩阵w，softmax回归模型输出每个类别的概率。它通过最大化似然函数来学习权重矩阵w，从而使得模型对已有数据能够尽可能准确地预测每个样本所对应的类别。



具体而言，我们定义一个函数为

$$P(y=j|x;w)=\frac{e^{w_j^T x}}{\sum_{k=1}^{K} e^{w_k^T x}}$$

该函数将输入特征x映射为一个向量，其中第j个元素表示样本x属于第j个类别的概率。其中，w是一个权重矩阵，每一列对应一个类别的参数向量。k是类别总数，x是输入特征向量。我们的目標是最大化一个似然函数，该函数用于衡量模型预测当前样本为正确类别的概率。这里的似然函数可以写成下面的形式：

$$L(w) = \prod_{i=1}^{N} P(y = y_i|x_i; w)$$

其中，N是样本总数，y_i和x_i分别是第i个样本的真实标签和特征向量。

我们使用负对数似然函数作为损失函数，它的形式如下：

$$J(w) = -\frac{1}{N}\sum_{i=1}^{N}\log(P(y = y_i|x_i; w))$$

求解数据集的最优权重矩阵w需要借助于梯度下降法，其中梯度的计算公式为：

$$\nabla_{w_j} J(w_j) = -\frac{1}{N}\sum_{i=1}^{N} X_{ij}(1y_i = j - P(y_i = j|x_i; w_j))$$

其中，$1y_i = j$为指示函数，当$y_i = j$时，该函数的值为1，否则为0。X_{ij}表示数据集中第i个样本的第j个特征。

为了使得模型能够更好地泛化到新的数据，我们需要对数据进行一些预处理，如特征缩放和添加偏置项。此外，还可以采用正则化策略来避免过拟合问题。

二、 python 代码实现

2.1 softmax_regression.py

```
import math
import pandas as pd
import numpy as np
import random
import time
import matplotlib.pyplot as plt

from sklearn.model_selection import train_test_split
from sklearn.metrics import accuracy_score


class Softmax(object):

    def __init__(self):
        self.learning_step = 0.000001           # 学习速率
        self.max_iteration = 100000             # 最大迭代次数
        self.weight_lambda = 0.01               # 衰退权重


    def cal_e(self,x,l):

        theta_l = self.w[l]                      # 取出第l行，即第l个分类器的权重
        product = np.dot(theta_l,x)             # 计算theta_l和x的点积
```

```
        return math.exp(product)                # 返回e的指数

    def cal_probability(self,x,j):              # 计算第j个分类器对x分类的概率

        molecule = self.cal_e(x,j)             # 分子，e的指数
        denominator = sum([self.cal_e(x,i) for i in range(self.k)]) # 分母

        return molecule/denominator         # 返回概率


    def cal_partial_derivative(self,x,y,j):

        first = int(y==j)                               # 计算示性函数
        second = self.cal_probability(x,j)              # 计算后面那个概率

        return -x*(first-second) + self.weight_lambda*self.w[j]

    # 预测标签，x是测试集的一个实例
    def predict_(self, x):
        result = np.dot(self.w,x)
        row, column = result.shape

        # 找最大值所在的列
        _positon = np.argmax(result)
        m, n = divmod(_positon, column)

        return m

    def train(self, features, labels):
        self.k = len(set(labels))

        self.w = np.zeros((self.k,len(features[0])+1))
        time = 0

        while time < self.max_iteration:
            print('loop %d' % time)
            time += 1
            index = random.randint(0, len(labels) - 1)

            x = features[index]
            y = labels[index]

            x = list(x)
            x.append(1.0)
            x = np.array(x)

            derivatives = [self.cal_partial_derivative(x,y,j) for j in
range(self.k)]

            for j in range(self.k):
                self.w[j] -= self.learning_step * derivatives[j]

    def predict(self,features):
        labels = []
```

```
        for feature in features:
            x = list(feature)
            x.append(1)

            x = np.matrix(x)
            x = np.transpose(x)

            labels.append(self.predict_(x))
        return labels


if __name__ == '__main__':

    print('Start read data')

    time_1 = time.time()

    # raw_data = pd.read_csv('/home/carton/workspace/python/Statistical-
learning/database/homework4/train.csv', header=0)
    raw_data = pd.read_csv('../../database/homework4/train.csv', header=0)
    data = raw_data.values

    imgs = data[0::, 1::]
    labels = data[::, 0]

    # 选取 2/3 数据作为训练集， 1/3 数据作为测试集
    train_features, test_features, train_labels, test_labels =
train_test_split(
        imgs, labels, test_size=0.33, random_state=23323)
    # print train_features.shape
    # print train_features.shape

    time_2 = time.time()
    print('read data cost '+ str(time_2 - time_1)+' second')

    print('Start training')
    p = Softmax()
    p.train(train_features, train_labels)

    time_3 = time.time()
    print('training cost '+ str(time_3 - time_2)+' second')

    print('Start predicting')
    test_predict = p.predict(test_features)
    time_4 = time.time()
    print('predicting cost ' + str(time_4 - time_3) +' second')

    score = accuracy_score(test_labels, test_predict)
    print("The accruacy socre is " + str(score))

    # 画图展示10个数字的预测结果
    fig, axes = plt.subplots(nrows=2, ncols=5, figsize=(10, 4))
    for i, ax in enumerate(axes.flat):
        # ax.imshow(X_test[i, 1:].reshape(28, 28), cmap='binary')
```

```
        # ax.set(title=f"Prediction: {np.argmax(y_pred, axis=1)[i]}")
        # ax.axis('off')
        ax.imshow(test_features[i].reshape(28, 28))
        ax.set(title=f"Prediction: {test_predict[i]}")
        ax.axis('off')

    plt.show()
```

1. 加载数据集，进行训练集和测试集的
2. 定义one-hot编码函数, 将标签转换为one-hot编码
3. 定义softmax函数，计算每个类别的概率
4. 定义损失函数，计算损失
5. 训练模型，使用梯度下降法更新参数
6. 预测模型，使用训练好的模型对测试集进行预测
7. 计算准确率，评估模型的性能

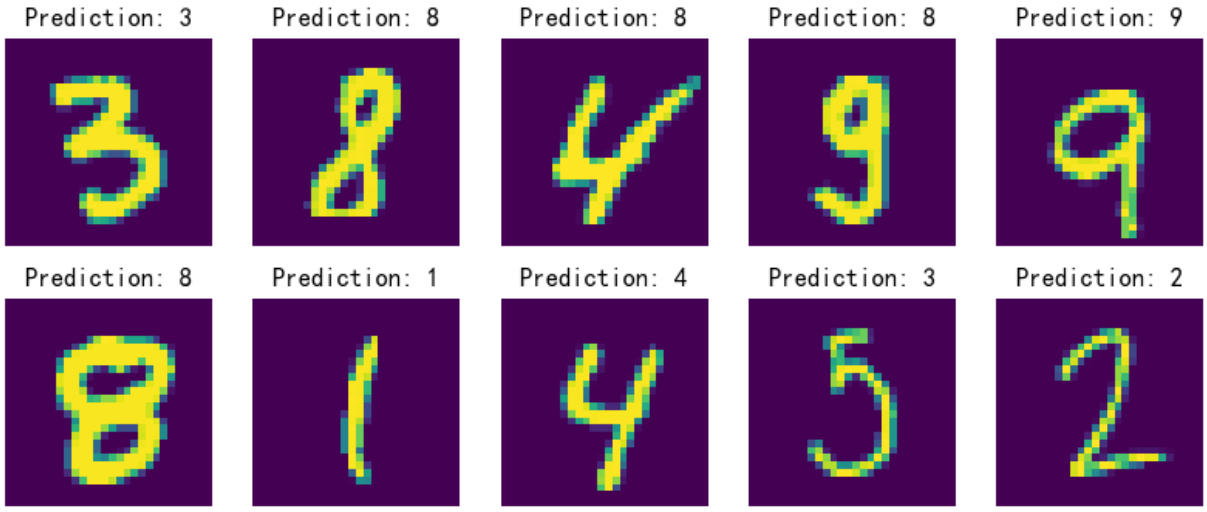
三、实验结果

1.分类准确度

```
loop 99995
loop 99996
loop 99997
loop 99998
loop 99999
training cost 35.81374549865723 second
Start predicting
predicting cost 0.9894149303436279 second
The accruacy socre is 0.8865079365079365
```

可以看到，softmax回归算法在 MNIST 数据集上的分类准确率为 0.89，说明该算法在处理简单的分类问题时具有较好的性能。

2.部分预测结果



四、总结体会

本次实验使用python实现了softmax回归算法，并使用该算法对 MNIST 数据集中的手写数字进行了分类。实验结果表明，softmax回归算法在处理简单的分类问题时具有较好的性能。

softmax回归算法是一种常见的分类算法，适用于多分类问题。它通过最大化似然函数来学习权重矩阵w，从而使得模型对已有数据能够尽可能准确地预测每个样本所对应的类别。在实际应用中，我们可以使用梯度下降法来求解最优权重矩阵w。此外，为了使得模型能够更好地泛化到新的数据，我们还可以对数据进行一些预处理，如特征缩放和添加偏置项。此外，还可以采用正则化策略来避免过拟合问题。

总之，softmax回归算法是一种简单而有效的分类算法，值得我们在实际应用中加以应用。