模式识别实验报告

专业：人工智能

学号：58121124

年级：21 级

姓名：张博彦

签名：

时间：

实验一 数据降维与分类实验

1. 问题描述

对红白葡萄酒数据分别进行 PCA 和 LDA 降维处理，在降维后的数据集上完成基于 logistic 回归分类器的训练和测试。

2. 实现步骤与流程

（1） 实验思路：使用 python 分别实现 PCA 降维函数和 LDA 降维函数，之后读取数据集中的数据，并分别调用 PCA，LDA 函数，得到降维后的数据集，再使用逻辑回归函数通过自己选取的训练集进行训练，得到两个数据降维后的模型，再使用该模型在测试集上进行预测并与其标签进行比对得到模型的正确率。最后使用未降维的训练集直接进行逻辑回归，并将得到的模型在测试集上预测，与其标签对比得到未降维模型的正确率。从而对比降维前后正确率的变化。

（2） 实验的重点和难点：

（a）PCA 中需自己通过实际的特征值选取能够保留 0.99 以上信息的特征值个数。

（b）LDA 为有监督的学习，故需先统计训练集总中两个类别的样本个数，再分别计算类内散度矩阵和类间散度矩阵。

（c）逻辑回归时需自己手动实现 sigmoid 函数，由于在实际计算时可能出现 e^(-x)的值越界的情况，故需进行一些数值调整。

（d）由于降维后的两个模型以及未降维的模型各不相同，故模型预测时的阈值需分别调整，得到最佳正确率。

以下为具体实现代码的解析：

（3）全局变量定义：

（a）红酒样本数：M = 1599

（b）白酒样本数：N = 4898

（c）数据集的 80%作为训练集：num_train = int((M + N) * 0.8)

（d）数据集的 20%作为测试集：num_test = M + N - num_train

（e）样本特征数量：num_feature = 12

（f）假设红酒的 label 为 0，白酒的 label 为 1,故标签数量：num_label = 2

（g）学习率：learning_rate = 0.01

（h）训练次数：epochs = 2000

（4）PCA 降维函数实现：

（a）传入参数说明：ds 为训练集的样本特征数据

（b）根据 ds 取得平均值 m，同时使用 num_sample 变量记录 ds 中的样本个数。

（c）计算每 PCA 技术中的矩阵 S。

（d)使用 np.linalg.eig 函数计算得到 S 矩阵的特征值 eigenvalue 和特征向量 feature_vector。

（e）将 eigenvalue 中的所有元素累加得到特征值之和 sigma。

（f）定义变量 count 用以记录保留信息超过 0.99 时所需要选取的特征值个数。

（g）由于通过 np.linalg.eig 函数得到的特征值以从大到小的顺序排序，故使用变量 eigenvalue_sigma 记录从下标为 0 开始累加特征值的数值。

（h）当 eigenvalue_sigma / sigma > 0.99 时，表明保存的数据信息超过了 0.99，此时终止 for 循环并打印 count 的值显示选取的特征值个数。

（i）根据 count 的值截取 feature_vector 中的特征向量得到投影矩阵 W。

（j）由于 ds 中含有 num_sample 个样本，而 m 平均值需对每个样本进行中心化，故将 m 变量变形为 num_sample * num_feature 的矩阵.

（k）最后，将原数据集 ds 减去变形后的矩阵 m 后，再乘以 W 得到降维后的样本数据集 ds 并作为返回值返回。

（5）LDA 函数实现：

（a）传入参数说明：ds 为训练集的样本特征数据， label 为训练集对应样本的标签。

（b）根据 ds，label 得到类间平均值 m，总样本数 n，类内平均值 mi，每个类的样本数 ni，其中 mi 为 num_label * num_feature 的矩阵，每一个行向量为一个类的均值，ni 为 1 * num_label 的数组，每一个元素为一个类的样本数。

（c）计算 LDA 中的类内散度矩阵 SW 和类间散度矩阵 SB。

（d）使用 np.linalg.eig 函数得到 SWˆ(-1) * SB 的特征值 eigenvalue 和特征向量 feature_vector。

（e）选取（num_label - 1）特征向量，并对 ds 进行降维，得到降维后的数据集 ds 并作为返回值返回。

（6）由于逻辑回归中需要用到 sigmoid 函数，故自己定义 sigmoid 函数，为防止出现 eˆ(-x)越界而计算错误的情况，故当可能越界时采取分子分母同时

乘以 e^x 从而通过计算 e^x 代替计算 e^(-x)，使得实验结果更加准确。

（7）定义逻辑回归函数 fun_log，并根据 epochs 对模型进行训练，得到逻辑回归模型中的 β 值。

（8）定义预测函数 decision，根据所给阈值，将大于阈值的测试样本预测为白酒，小于阈值的测试样本预测为红酒

（9）由于数据集存储在.csv 文件中，故使用 pd.rad_csv 函数读取数据，之后使用 shuffle 函数打乱数据，使得之后的模型能够正常拟合。

（10）分别进行 PCA，LDA 降维，得到的训练集进行逻辑回归模型拟合，之后使用模型在测试集上进行预测并计算正确率

（11）使用未降维的训练集进行模型拟合并在测试集上预测，计算正确率。

（12）将三个模型的预测结果写入.csv 文件。

3. 实验结果与分析

以下为在 pycharm 上运行得到的结果

```
对数据集进行PCA降维：
根据所求得的特征值，选取2个特征值即可使保留的信息超过0.99
对数据集进行LDA降维：
由于一共有2个label，故降维至1维并选取1个特征值及特征向量
PCA降维后正确率为： 0.9246153846153846
LDA降维后正确率为： 0.9276923076923077
使用原数据集进行模型拟合并预测后，正确率为： 0.9407692307692308
```

以下为在 MindSpore 平台上运行得到的结果

```
PCA降维后正确率为： 0.9230769230769231
LDA降维后正确率为： 0.8953846153846153
使用原数据集进行模型拟合并预测后，正确率为： 0.9015384615384615
```

（1）对于 PCA 降维后得到的模型，经过对阈值在 0.1-0.6 范围内的反复调整，发现当阈值取到 0.2 时，有最好的预测结果

（2）对于 LDA 降维后得到的模型，经过对阈值在 0.4-0.9 范围内的反复调整，发现当阈值取到 0.75 时，有最好的预测结果

（3）对于未经降维的模型，经过对阈值在 0.4-0.6 围内的反复调整，发现当阈值取到 0.5 时，有最好的预测结果

（4）对比降维前后，发现降维前后模型的正确率降低了 0.02，可能是由于在降维时，保留的信息仍旧不够充分，导致损失了部分重要信息，使得模型的拟合并未达到提高正确率的效果，同时发现 PCA 和 LDA 的正确率相差无及，可见在该实验中两种降维技术表象并无太大的差别。同时由于在该实验中，模型在原数据集上就已经取得了较好的预测效果，所以使用降维技术并不能够显著提高模型的预测能力。

4. MindSpore 学习使用心得体会

由于 MindSpore 平台无相关开发成熟的库函数，故只能使用自己的代码在 MindSpore 云平台上运行，由于创建的 notebook 为 2 核 8GB，故在运行速度方面相较于自己的电脑，会较慢。希望之后的 MindSpore 平台能够开发出有关 pca 和 lda 的相关降维库函数同时在使用说明方面能够更加详细一些，省去学生独自研究如何使用该平台的时间，能够专注于代码编写，而不是环境配置以及代码运行平台的运行性能选择。

5. 代码附录

```
import numpy as np
import pandas as pd


M = 1599  # 红酒的样本数
N = 4898  # 白酒的样本数
num_train = int((M + N) * 0.8)  # 选取数据集的 80%作为训练集
num_test = M + N - num_train  # 选取数据集的 20%作为测试集
num_feature = 12  # 样本的特征数量
num_label = 2  # 假设红酒的 label 为 0，白酒的 label 为 1,一共有两个标签
learning_rate = 0.01  # 学习率
epochs = 2000  # 训练次数


# PCA 降维技术
def PCA(ds):  # ds 为所要降维的数据集
    # 求平均值
    m = np.zeros(num_feature)
    num_sample = ds.shape[0]
    for i in ds:
        m += i
    m /= num_sample
    # 求 S 矩阵
    S = np.zeros((num_feature, num_feature))
    for i in ds:
        S += np.outer(np.array(i - m), np.array(i - m))
```

```
        eigenvalue, feature_vector = np.linalg.eig(S)
        # 确定应将数据降至几维，即需要选取多少个特征值
        sigma = 0
        for i in eigenvalue:
            sigma = sigma + abs(i)
        count = 0  # 用于记录需要选取多少个特征值，才能使保留的信息超过0.99
        eigenvalue_sigma = 0
        for i in range(len(eigenvalue)):
            eigenvalue_sigma = eigenvalue_sigma + eigenvalue[i]
            count += 1
            if eigenvalue_sigma / sigma > 0.99:
                break
        print("根据所求得的特征值，选取%s个特征值即可使保留的信息超过0.99" % count)

        W = feature_vector[:count, ]  # 得到投影矩阵
        m = np.outer(np.ones(num_sample).reshape(-1, 1), m.reshape(-1, num_feature))  # 用于中心化每一个样本数据
        ds = (ds - m) @ W.T  # 对ds进行降维
        return ds
    # LDA降维技术
    def LDA(ds, label):  # ds为数据集的feature， label为数据集的label
        m, n = 0, (M + N)  # m为类间平均值， n为总样本数
        mi, ni = np.zeros((num_label, num_feature)), np.zeros(num_label)  # mi为类内平均值， ni为每个类中的样本数
        for i in range(n):  # 求和
            m = m + ds[i]
            mi[int(label[i])] += ds[i]
            ni[int(label[i])] += 1
        # 取平均值
        m /= n
        for i in range(num_label):
```

```
            mi[i] /= ni[i]
        # 计算类内，类间散度矩阵
        SW, SB = np.zeros((num_feature, num_feature)), np.zeros((num_feature, num_feature))
        for i in range(n):
            SW += np.outer((ds[i] - mi[int(label[i])]), (ds[i] - mi[int(label[i])]))
        for i in range(num_label):
            SB += ni[i] * np.outer((mi[i] - m), (mi[i] - m))

        eigenvalue, feature_vector = np.linalg.eig(np.dot(np.linalg.pinv(SW), SB))
        print(f"由于一共有{num_label}个 label，故降维至{num_label - 1}维并选取{num_label - 1}个特征值及特征向量")
        W = feature_vector[(num_label - 1):num_label, ]
        ds = ds @ W.T
        return ds
    # sigmoid 函数
    def sigmoid(x):  # 由于会出现 np.exp(-x)值越界情况，故进行手动调整
        x_ravel = x.ravel()  # 将 numpy 数组展平
        length = len(x_ravel)
        y = []
        for i in range(length):
            if x_ravel[i] >= 0:
                y.append(1.0 / (1 + np.exp(-x_ravel[i])))
            else:
                y.append(np.exp(x_ravel[i]) / (np.exp(x_ravel[i]) + 1))
        return np.array(y).reshape(x.shape)
    # 逻辑回归模型构建函数
    def func_log(feature, label, lr, epochs):  # X_train 为 feature, Y_train 为 label
        num_train = feature.shape[0]
        beta = np.zeros(feature.shape[1]).reshape(-1, 1)  # 初始化模
```

型参数

```
        for i in range(epochs):
            z = np.dot(feature, beta)
            h = sigmoid(z)
            error = h - label
            beta = beta - lr * np.dot(feature.T, error) / num_train

        return beta
    # 预测函数
    def decision(beta, feature, value): # X_val 为 feature, value 为阈
值
        Y_pred = []
        log = feature @ beta
        for i in log:
            if sigmoid(i) > value:
                Y_pred.append(1)
            else:
                Y_pred.append(0)
        return np.array(Y_pred)

    # 读取文件数据
    red = pd.read_csv("winequality-red.csv")
    white = pd.read_csv("winequality-white.csv")
    # 将每个样本个体的数据提取成数组形式
    ds = []
    label = []
    for n in range(M):  # 提取红葡萄酒数据
        data = red.iloc[n]
        arr = []
        temp = ""
        for i in data[0]:
            if i != ";":
                temp = temp + i
            else:
```

```
                temp = float(temp)
                arr.append(temp)
                temp = ""
        arr.append(float(temp))
        ds.append(arr)
        label.append(0)
    for n in range(N):  # 提取白葡萄酒数据
        data = white.iloc[n]
        arr = []
        temp = ""
        for i in data[0]:
            if i != ";":
                temp = temp + i
            else:
                temp = float(temp)
                arr.append(temp)
                temp = ""
        arr.append(float(temp))
        ds.append(arr)
        label.append(1)
    ds = np.array(ds)
    label = np.array(label)
    data = np.c_[ds, label]
    np.random.seed(2)
    np.random.shuffle(data)  # 打乱样本顺序便于进行模型拟合
    feature, label = data[:, :num_feature], data[:, num_feature:]  # 将数据集的feature与label分割为两个矩阵

    print("对数据集进行PCA降维：")
    feature_pca = PCA(feature)
    print("对数据集进行LDA降维：")
    feature_lda = LDA(feature, label)

    # 将数据集中80%作为训练集，20%作为测试集
```

```
    train_feature_pca, test_feature_pca = feature_pca[:num_train, ], feature_pca[num_train:, ]
    train_feature_lda, test_feature_lda = feature_lda[:num_train, ], feature_lda[num_train:, ]
    train_label, test_label = label[:num_train, ], label[num_train:, ] #由于降维不影响 label，故 PCA,LDA 以及原数据集都使用相同分割后的 label 数组

    # 进行逻辑回归
    beta_pca = func_log(train_feature_pca, train_label, learning_rate, epochs)
    beta_lda = func_log(train_feature_lda, train_label, learning_rate, epochs)
    # 进行预测
    value_pca = 0.2  # pca 设置阈值为 0.2
    value_lda = 0.75  # lda 设置阈值为 0.75
    pred_pca = decision(beta_pca, test_feature_pca, value_pca)
    pred_lda = decision(beta_lda, test_feature_lda, value_lda)
    # 分别计算 PCA 和 LDA 过后预测的准确率
    total = num_test
    correct_pca = 0
    correct_lda = 0
    for i in range(total):
        if pred_pca[i] == test_label[i]:
            correct_pca += 1
        if pred_lda[i] == test_label[i]:
            correct_lda += 1
    print("PCA 降维后正确率为:", correct_pca / total)
    print("LDA 降维后正确率为:", correct_lda / total)

    total = num_test
    train_feature, test_feature = feature[:num_train, ], feature[num_train:, ]
    beta_origin = func_log(train_feature, train_label, learning_rate,
```

```
epochs)
    value_origin = 0.5  # 设置阈值为0.5
    pred = decision(beta_origin, test_feature, value_origin)
    correct = 0
    for i in range(total):
        if pred[i] == test_label[i]:
            correct += 1
    print("使用原数据集进行模型拟合并预测后，正确率为:", correct / total)
    # 写入PCA模型的预测结果
    dataframe = pd.DataFrame({"class": pred_pca})
    dataframe.to_csv("pred_pca_result.csv", index=False, sep=',')
    #写入LDA模型的预测结果
    dataframe = pd.DataFrame({"class": pred_lda})
    dataframe.to_csv("pred_lda_result.csv", index=False, sep=',')
    #写入原数据集模型的预测结果
    dataframe = pd.DataFrame({"class": pred})
    dataframe.to_csv("pred_origin_data_result.csv",      index=False, sep=',')
```

实验二 KNN 分类任务实验

1. 问题描述

分别利用欧式距离，马氏距离作为 KNN 算法的度量函数，利用 KNN 算法对 Iris 鸢尾花数据集中的测试集进行分类。

2. 实现步骤与流程

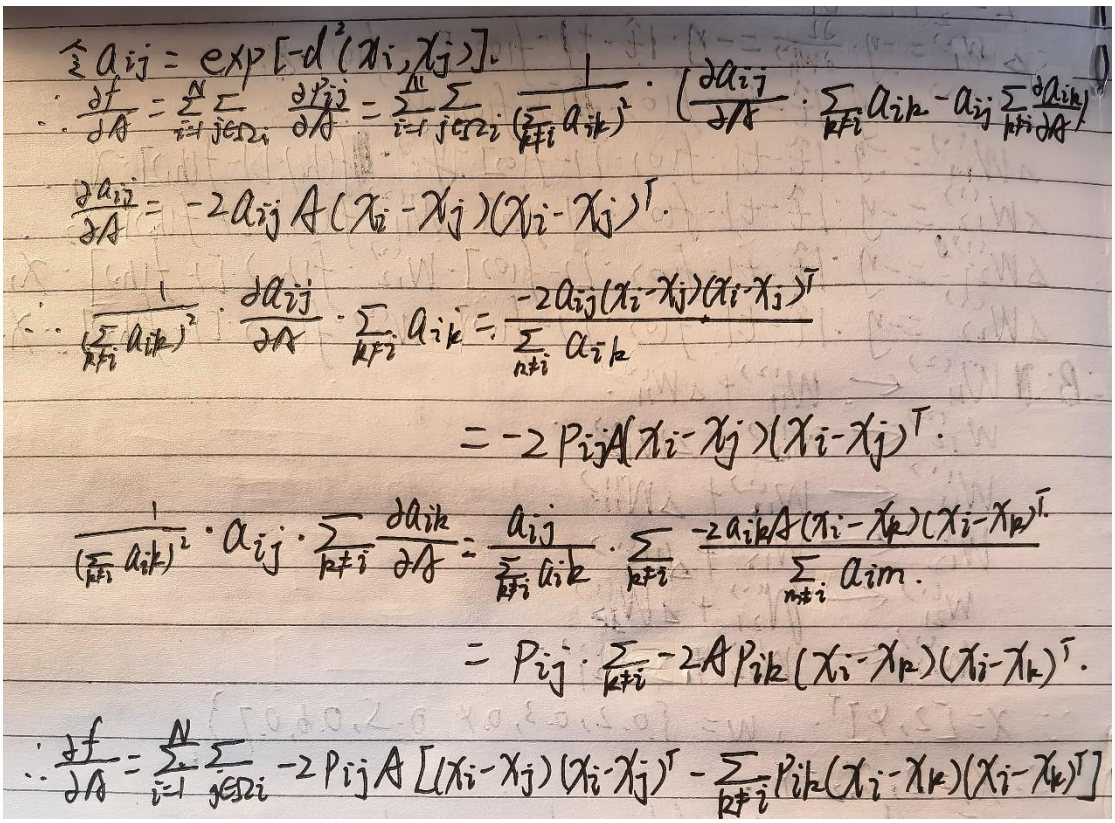
（1）实验思路：使用 python 实现使用欧氏距离作为度量标准，在验证集上查看不同 K 值的影响，使用在验证集上取得最好预测结果的 K 值在测试集上预测。使用训练集训练马氏距离中的参数 A，拟合之后使用马氏距离作为度量标准，在验证集上查看不同 K 值的影响，使用在验证集上取得最好预测结果的 K 值在测试集上预测。

（2）实验的重点和难点：

（a）需要自己根据选用不同 K 值得到的预测结果，选择最好的 K 值在测试集上进行预测。

（b）由于马氏距离需要进行梯度下降，故对 f 函数关于 A 的求导需要准确计算

（3）马氏距离中 f 函数关于 A 的导数推导过程如下：



以下为具体实现代码的解析

（4）全局变量定义：

（a）计算马氏距离时的训练轮数：epochs = 10

（b）学习率：lr = 0.18

（c）样本的特征数：num_feature = 4

（4）欧氏距离作为度量的 KNN 函数实现：

（a）传入参数说明：X 为所要进行预测的样本特征集， X_train 为训练集的样本特征，Y_train 为训练集的样本标签，K 为近邻数

（b）对于每一个需要进行预测的样本，计算该样本与每个训练样本的欧氏距离，之后将欧式距离进行排序得到 K 个距离最近的样本，然后将 K 个样本中三种标签的样本进行排序，得到最多的标签作为该样本的预测标签。

（5）马氏距离中的参数 A 训练函数：

（a）传入参数说明：X_train 为训练集的样本特征， Y_train 为训练集的样本标签

（b）根据（3）中的求导公式，对 A 进行训练将训练好的 A 作为返回值返回。

（6）马氏距离作为度量的 KNN 函数实现

（a）传入参数说明：X 为所要进行预测的样本特征集， X_train 为训练集的样本特征，Y_train 为训练集的样本标签，A 为马氏距离的参数，K 为近邻数

（b）对于每一个需要进行预测的样本，计算该样本与每个训练样本的马氏距离，之后将马氏距离进行排序得到 K 个距离最近的样本，然后将 K 个样本中三种标签的样本进行排序，得到最多的标签作为该样本的预测标签。

（7）由于数据集存储在.csv 文件中，故使用 pd.rad_csv 函数读取数据.

（8）对于 K 的值，取 1 到 10，分别使用两种距离度量方式的 KNN 在验证集上进行预测，并可视化不同 K 值的预测结果，最后得到最佳的 K 值

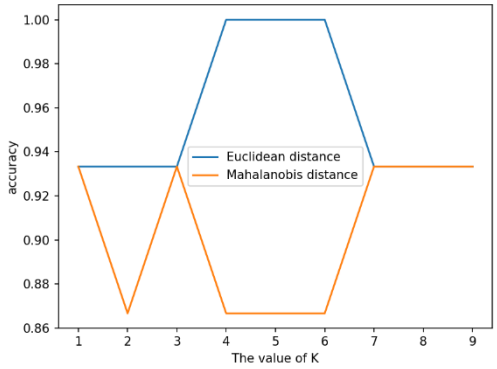
（9）使用最佳的 K 值，在测试集上进行预测

（10）将预测结果写入文件

3. 实验结果与分析

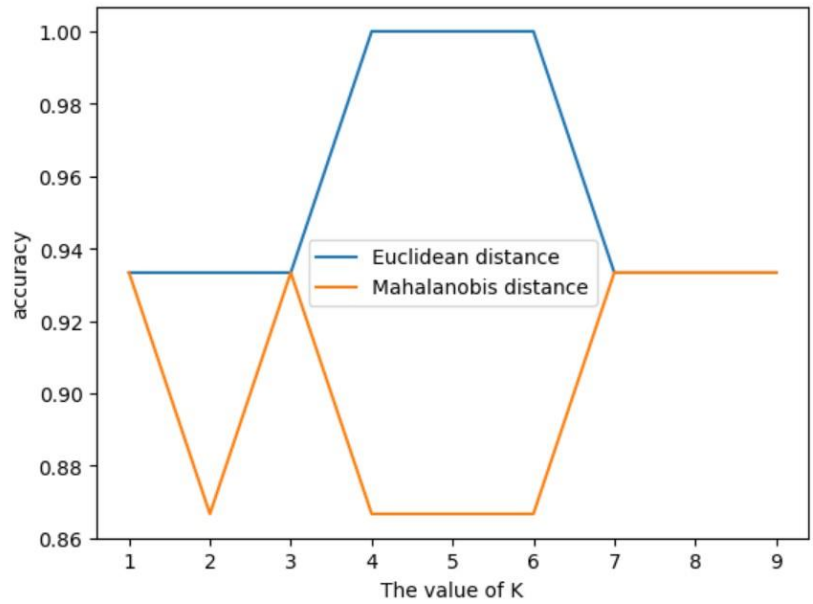
在 pycharm 上运行的结果：

```
Epoch0: 目标函数f的值为87.65835909154744
Epoch1: 目标函数f的值为83.78141753279012
Epoch2: 目标函数f的值为83.53544515977259
Epoch3: 目标函数f的值为83.30825269260866
Epoch4: 目标函数f的值为83.17253736842453
Epoch5: 目标函数f的值为83.19887275149324
Epoch6: 目标函数f的值为83.59678106160376
Epoch7: 目标函数f的值为84.13782475576836
Epoch8: 目标函数f的值为84.57002543964032
Epoch9: 目标函数f的值为84.68467269377874
```



在 mindspore 上运行的结果：

```
Epoch0: 目标函数f的值为87.65835909154745
Epoch1: 目标函数f的值为83.7814175327901
Epoch2: 目标函数f的值为83.5354451597726
Epoch3: 目标函数f的值为83.30825269260868
Epoch4: 目标函数f的值为83.17253736842453
Epoch5: 目标函数f的值为83.19887275149323
Epoch6: 目标函数f的值为83.59678106160376
Epoch7: 目标函数f的值为84.13782475576839
Epoch8: 目标函数f的值为84.57002543964035
Epoch9: 目标函数f的值为84.68467269377874
```



由于 MindSpore 没有专门的 KNN 库函数，故只能将自己的源代码放在 MindSpore 平台上运行。由于验证集、测试集样本数量较少，在使用欧氏距离进行 KNN 分类时能够在多个 K 值处取得极好的预测结果。而在使用马氏距离进行 KNN 分类时，正确率一直保持在 0.93 左右，无法达到 1，无论如何改变学习率和训练轮数，正确率都没有提升，可能是由于自身代码中对于马氏距离参数 A 的处理不够细节，部分求导实现代码可能需要优化，使得函数 f 能够到达最优值处。

4. MindSpore 学习使用心得体会

由于 MindSpore 平台无相关开发成熟的库函数，故只能使用自己的代码在 MindSpore 云平台上运行，由于创建的 notebook 为 2 核 8GB，故在运行速度方面相较于自己的电脑，会较慢。希望之后的 MindSpore 平台能够开发出有关 KNN 分类的相关库函数同时在使用说明方面能够更加详细一些，省去学生独自研究如何使用该平台的时间，能够专注于代码编写，而不是环境配置以及代码运行平台的运行性能选择。

5. 代码附录

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

epochs = 10  # 计算马氏距离时的训练轮数
lr = 0.18  # 学习率
num_feature = 4  # 特征数
X_train = pd.read_csv("train.csv").drop("label", axis=1).to_numpy()
Y_train = pd.read_csv("train.csv")["label"].to_numpy()
X_val = pd.read_csv("val.csv").drop("label", axis=1).to_numpy()
Y_val = pd.read_csv("val.csv")["label"].to_numpy()
X_test = pd.read_csv("test_data.csv").to_numpy()

# 欧氏距离实现 KNN 分类
def my_KNN_1(X, X_train, Y_train, K):  # X 为所要进行预测的数据集，K 为近邻数
    # num = X_train.shape[0]  # 得到训练集中所含的样本数
    preds = []
    for sample in X:
```

```
        distance = []
        for i in X_train:
            distance.append(np.linalg.norm(sample - i))  # 计算欧氏
距离
        label_order =np.argsort(distance)
        num_label = np.zeros(3)  # 记录K个近邻点中三个label的数量
        for i in range(K):
            num_label[Y_train[label_order[i]]] += 1
        preds.append(np.argsort(-num_label)[0])  # 得到3类中数量最
多的label并作为预测结果
    return preds
# 马氏距离实现KNN分类
def train_A(X_train, Y_train):
    num = X_train.shape[0]  # 计算样本个数
    np.random.seed(10)
    A = np.random.randn(num_feature, num_feature)  # 初始化正交基A

    for epoch in range(epochs):
        derivative = np.zeros((num_feature, num_feature))  # 初始化
导数
        f = 0  # 目标函数
        for i in range(num):
            sigma_k_deno = 0  # p_ij, p_ik的分母
            for k in range(num):
                if k == i:
                    pass
                else:
                    sigma_k_deno += np.exp(-
(np.linalg.norm((X_train[i] - X_train[k]) @ A)**2))
            for j in range(num):
                if Y_train[i] != Y_train[j] or i == j:
                    pass
                else:
                    sigma_k_mol = np.zeros((num_feature,
```

```
num_feature))  # 关于 k 求和的分子之和（分母相同）
                    for k in range(num):
                        if k == i or k == j:
                            pass
                        else:
                            sigma_k_mol += np.exp(-
(np.linalg.norm((X_train[i] - X_train[k]) @ A)**2)) * (
                                        np.outer(X_train[i] -
X_train[k], X_train[i] - X_train[k])
                                        - np.outer(X_train[i] -
X_train[j], X_train[i] - X_train[j]))
                    p_ij = np.exp(-(np.linalg.norm((X_train[i] -
X_train[j]) @ A)**2)) / sigma_k_deno
                    derivative += p_ij * sigma_k_mol / sigma_k_deno
                    f += p_ij
        print(f"Epoch{epoch}: 目标函数 f 的值为{f}")
        A -= 2 * lr * A @ derivative
    return A
def my_KNN_2(X, X_train, Y_train, A, K):
    preds = []
    for sample in X:
        distance = []
        for i in X_train:
            distance.append(np.sqrt(abs((sample - i) @ A @ (sample -
i))))  # 计算欧氏距离
        label_order =np.argsort(distance)
        num_label = np.zeros(3)  # 记录 K 个近邻点中三个 label 的数量
        for i in range(K):
            num_label[Y_train[label_order[i]]] += 1
        preds.append(np.argsort(-num_label)[0])  # 得到 3 类中数量最
多的 label 并作为预测结果
    return preds

accuray_1 = []
```

```
accuray_2 = []
A = train_A(X_train, Y_train)
for K in range(1, 10):
    pred_1 = my_KNN_1(X_val, X_train, Y_train, K)
    pred_2 = my_KNN_2(X_val, X_train, Y_train, A, K)
    acc_1 = 0
    acc_2 = 0
    for i in range(Y_val.shape[0]):
        if pred_1[i] == Y_val[i]:
            acc_1 += 1
        if pred_2[i] == Y_val[i]:
            acc_2 += 1
    accuray_1.append(acc_1 / Y_val.shape[0])
    accuray_2.append(acc_2 / Y_val.shape[0])

X = np.arange(1, 10)
plt.xlabel('The value of K')
plt.ylabel('accuracy')
plt.plot(X, accuray_1, label="Euclidean distance")
plt.plot(X, accuray_2, label="Mahalanobis distance")
plt.legend()
plt.show()

# 在测试集上进行运行
K_1 = 4  # 根据可视化结果得到使用欧式距离是选取 K = 4 预测结果最好
K_2 = 3  # 根据可视化结果得到使用马氏距离是选取 K = 5 预测结果最好
pred_test_1 = my_KNN_1(X_test, X_train, Y_train, K_1)
pred_test_2 = my_KNN_2(X_test, X_train, Y_train, A, K_2)
# 将预测结果写入文件
dataframe = pd.DataFrame({"label": pred_test_1})
dataframe.to_csv("task1_test_prediction.csv", index=False, sep=',')
dataframe = pd.DataFrame({"label": pred_test_2})
dataframe.to_csv("task2_test_prediction.csv", index=False, sep=',')
```

实验三 神经网络

1. 问题描述

基于神经网络模型及 BP 算法，根据训练集中的数据对所设计的神经网络模型进行训练，随后对 MNIST 数据集中给定的测试集进行分类。

2. 实现步骤与流程

（1）实验思路：使用 python 首先对原数据集 MNIST 进行预处理，之后根据设计一层隐藏层的神经网络，然后在训练集上使用 BP 算法对参数进行训练，得到拟合之后的模型，最后使用该模型在测试集上进行预测。

（2）实验的重点和难点：

（a）由于无法使用 Datasets 库以及 Dataloader 函数，故需要手动实现对于原数据集的数据处理。

（b）BP 算法为该模型的核心，需要精准实现损失关于各个参数的导数，从而使参数达到预期拟合效果。

（c）训练时既要防止欠拟合问题，同时也要避免过拟合问题。

以下为具体实现代码的解析：

（3）全局变量定义：

（a）输入层节点数：num_input = 784

（b）隐藏层节点数：num_hidden = 256

（c）输出层节点数：num_output = 10

（d）学习率：lr = 0.01

（e）训练轮数：epochs = 10

（4）读取原数据集函数实现：

（a）传入参数说明：imgf 为图像数据集， labelf 为标签数据集，outf 为将原文件数据读取后存储的文件， n 为数据集的样本数

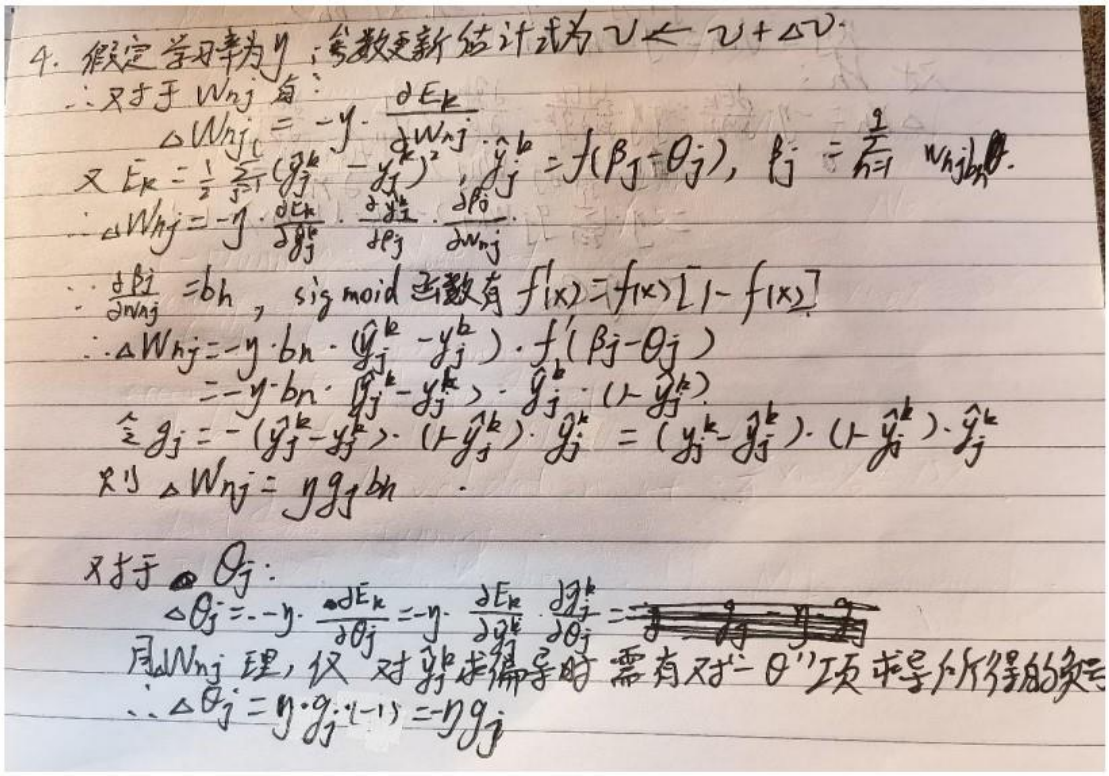
（b）对于每一个样本，使用 ord、join 函数原文件的二进制数据转化为纯文本.csv 文件

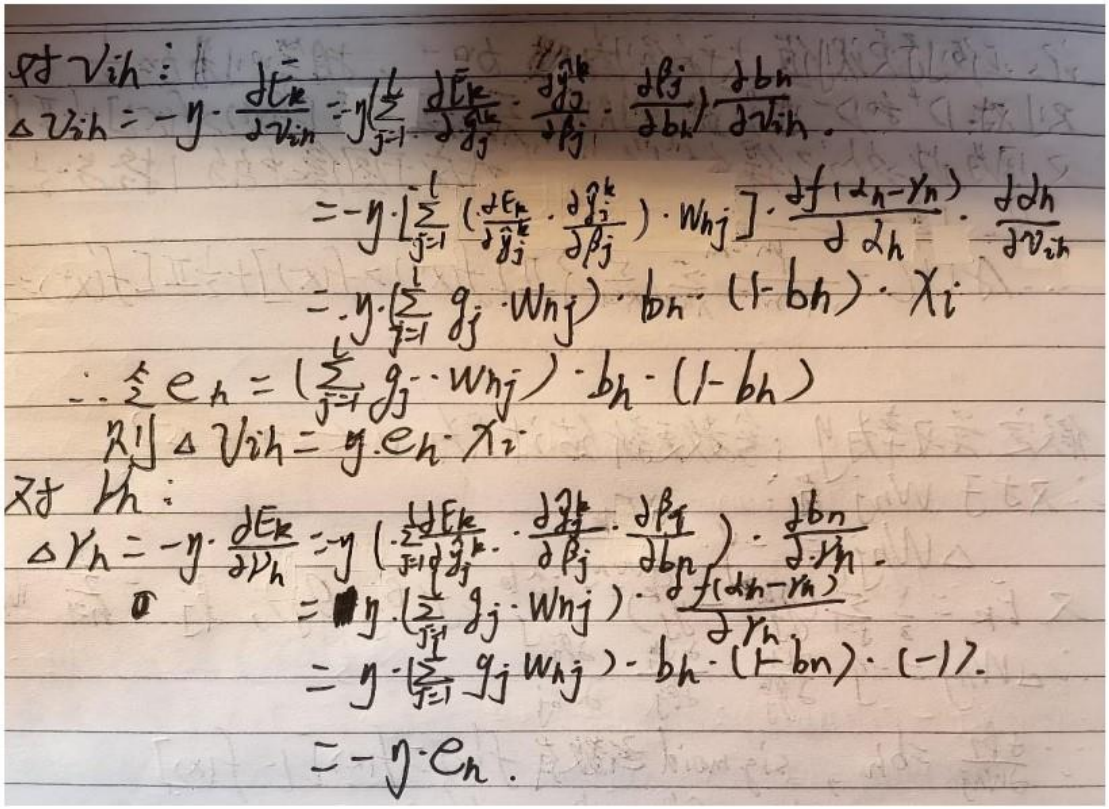
（5）全连接神经网络类实现：

（a）传入参数说明：num_input 为输入层节点数， num_hidden 为隐藏层节点数， num_output 为输出层节点数， lr 为学习率

（b）前向算法函数：使用 sigmoid 函数作为激活函数，根据全连接神经网络定义，依次计算隐藏层输入值和输出值、输出层输入值和输出层。（此处定义的神经网络仅使用 W1,W2 两个参数，W1 为（w1，b1）形式，W2 为（w2， b2）形式）

（c）后向算法实现：依据 BP 算法推导公式，计算 ΔW1， ΔW2





（8）由于无法使用已实现的 sigmoid 函数，故手动定义 sigmoid 函数

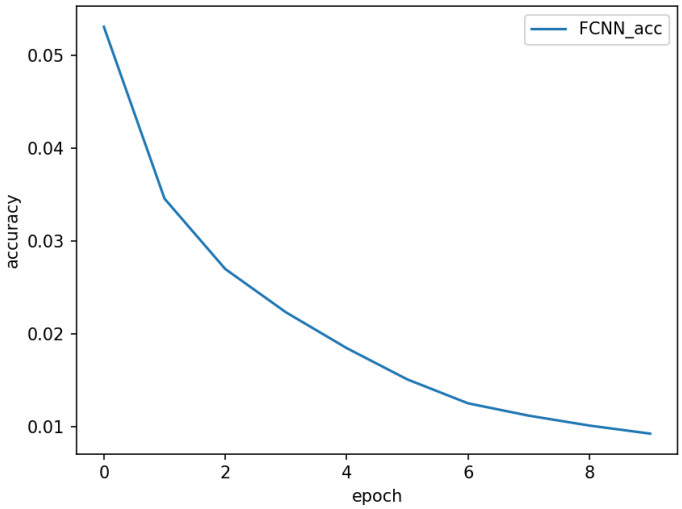
（9）首先对原数据集进行处理形成.csv 文件，然后通过读取.csv 文件得到训练样本和测试样本，之后调用 FCNN 函数在训练集上进行训练并记录每次训练后的损失并可视化该损失。

（10 将训练好的模型对测试集进行预测并计算准确率，并将预测结果进行保存

3. 实验结果与分析

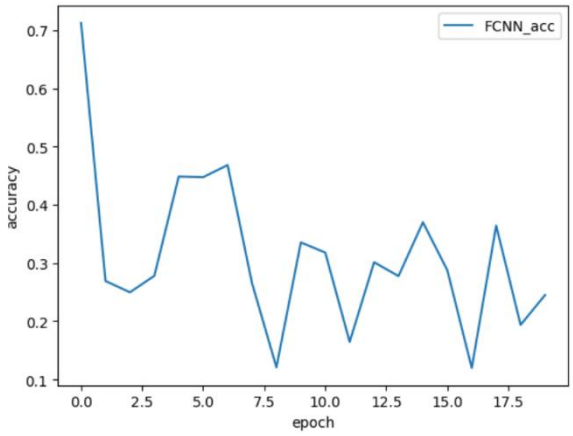
pycharm 上运行的结果：

```
Epoch0:loss is 0.05308154429782264
Epoch1:loss is 0.03454409997616712
Epoch2:loss is 0.02696329173447665
Epoch3:loss is 0.022293777883753297
Epoch4:loss is 0.01843579740222571
Epoch5:loss is 0.015055322965170296
Epoch6:loss is 0.012480004782258922
Epoch7:loss is 0.01114899097089548
Epoch8:loss is 0.010081358085036995
Epoch9:loss is 0.00921078638945102
accuray is  0.8688
```



在 mindspore 上运行的结果：

```
Epoch0损失为 0.7122677564620972
Epoch1损失为 0.26913323998451233
Epoch2损失为 0.24978162348270416
Epoch3损失为 0.2780814468860626
Epoch4损失为 0.4486592411994934
Epoch5损失为 0.44752445816993713
Epoch6损失为 0.4682551622390747
Epoch7损失为 0.26547688245773315
Epoch8损失为 0.12109623104333878
Epoch9损失为 0.335458368062973
Epoch10损失为 0.31799307465553284
Epoch11损失为 0.16462475061416626
Epoch12损失为 0.3014112412929535
Epoch13损失为 0.27765458822250366
Epoch14损失为 0.37008875608444214
Epoch15损失为 0.2875370681285858
Epoch16损失为 0.11989988386631012
Epoch17损失为 0.36430099606513977
Epoch18损失为 0.19384244084358215
Epoch19损失为 0.24500423669815063
```



测试集上的正确率为： 0.8785

自己手动实现的 FCNN 网络准确率较为理想，但发现在训练过程中，由于是逐个样本梯度下降，所以运行时间为较长，同时发现即使增加训练轮数，损失仍旧无法达到最低处，可能需要优化数据预处理方式和激活函数才能提高模型的预测准确率。在 MindSpore 平台上使用相关库实现发现，由于库函数为批处理，故训练时间会大幅下降且能够保持较好的模型性能，同时选用库函数 ReLU 函数作为激活函数，发现，模型准确率有所提高。

4. MindSpore 学习使用心得体会

MindSpore 提供了神经网络的一系列库函数，但由于先前没有相关了解基础且和 torch 库有较大的差别，故耗费了较多的时间在学习 MindSpore 库函数的使用方法上，同时在损失函数中没有交叉熵损失函数，因此仅能使用均方误差作为损失函数。

5. 代码附录

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

num_input = 784  # 输入层节点数
num_hidden = 256  # 隐藏层节点数
num_output = 10  # 输出层节点数
lr = 0.01  # 学习率
epochs = 10  # 训练轮数

def convert(imgf, labelf, outf, n):  # 将原文件读取成.csv 文件
    f = open(imgf, "rb")
    o = open(outf, "w")
    l = open(labelf, "rb")

    f.read(16)
    l.read(8)
    images = []

    for i in range(n):
        image = [ord(l.read(1))]
        for j in range(28 * 28):
            image.append(ord(f.read(1)))
        images.append(image)

    for image in images:
        o.write(",".join(str(pix) for pix in image)+"\n")
```

```
    f.close()
    o.close()
    l.close()

class FCNN:
    def __init__(self, num_input, num_hidden, num_output, lr):
        self.input = num_input
        self.hidden = num_hidden
        self.output = num_output
        self.W1 = np.random.normal(0.0, pow(self.hidden, -0.5),
(self.hidden, self.input))  # 隐藏层权重
        self.W2 = np.random.normal(0.0, pow(self.hidden, -0.5),
(self.output, self.hidden))  # 输出层权重
        self.lr = lr
        self.hidden_inputs = 0  # 此处仅用于定义为类变量
        self.hidden_outputs = 0  # 此处仅用于定义为类变量
        self.final_inputs = 0  # 此处仅用于定义为类变量
        self.final_outputs = 0  # 此处仅用于定义为类变量

    def forward(self, inputs_list):
        # 进行前向传播
        inputs = np.array(inputs_list, ndmin=2).T
        self.hidden_inputs = np.dot(self.W1, inputs)
        self.hidden_outputs = my_sigmoid(self.hidden_inputs)
        self.final_inputs = np.dot(self.W2, self.hidden_outputs)
        self.final_outputs = my_sigmoid(self.final_inputs)

    def backward(self, inputs_list, targets_list):
        # 进行反向传播
        inputs = np.array(inputs_list, ndmin=2).T
        targets = np.array(targets_list, ndmin=2).T
        output_errors = targets - self.final_outputs
        hidden_errors = np.dot(self.W2.T, output_errors)
```

```
        self.W2 += self.lr * np.dot((output_errors *
self.final_outputs * (1.0 - self.final_outputs)),
np.transpose(self.hidden_outputs))
        self.W1 += self.lr * np.dot((hidden_errors *
self.hidden_outputs * (1.0 - self.hidden_outputs)),
np.transpose(inputs))
        return np.linalg.norm(output_errors)

def my_sigmoid(x):
    return 1 / (1 + np.exp(-x))
# 读取文件
# 先将原文件转化为.csv 文件
convert("train-images-idx3-ubyte", "train-labels-idx1-ubyte",
"mnist_train.csv", 60000)
convert("t10k-images-idx3-ubyte", "t10k-labels-idx1-ubyte",
"mnist_test.csv", 10000)
# 读取.csv 文件
training_data_file = open("mnist_train.csv", 'r')
training_data_list = training_data_file.readlines()
training_data_file.close()
test_data_file = open("mnist_test.csv", 'r')
test_data_list = test_data_file.readlines()
test_data_file.close()

model = FCNN(num_input, num_hidden, num_output, lr)
Loss = []
for epoch in range(epochs):
    for record in training_data_list:
        all_values = record.split(',')  # 读取特征
        inputs = np.asfarray(all_values[1:]) / 255.0# 对灰度图进行归
一化
        # 独热编码，方便后续求导进行梯度下降
        targets = np.zeros(num_output)
        targets[int(all_values[0])] = 1
```

```
            # 前向预测
            model.forward(inputs)
            # 调用后向算法，并记录损失值
            loss = model.backward(inputs, targets)
            Loss.append(loss)
        print(f"Epoch{epoch}:loss is {loss}")

# 将训练过程中的损失可视化
x = np.arange(epochs)
plt.xlabel('epoch')
plt.ylabel('accuracy')
plt.plot(x, Loss, label='FCNN_acc')  # 训练过程中的损失
plt.legend()
plt.show()

# 在测试集上进行预测
correct = 0
preds = []
for record in test_data_list:
    all_values = record.split(',')
    correct_label = int(all_values[0])
    inputs = np.asfarray(all_values[1:]) / 255.0
    model.forward(inputs)
    pred = model.final_outputs
    preds.append(pred)
    label = np.argmax(pred)  # 将 pred 中值最大的元素下标为预测值
    if label == correct_label:
        correct += 1

print("accuray is ", correct / 10000)
# 将预测结果写入文件
dataframe = pd.DataFrame({"label": preds})
dataframe.to_csv("pred_test.csv", index=False, sep=',')
```

心得体会

在进行第一个实验的过程中，需要将所学的理论知识实践到代码上，其实会出现许许多多的问题，比如在 pca 和 lda 降维技术中，通过网上的资料查找发现许多方法都会选择使用奇异值代替特征值，而在本次实验中完全可以根据所学方法使用特征值进行降维也能取得较好的拟合结果。同时在实验时发现，在使用降维技术之后，模型的准确率不增反降，与原本的降维目标相违背，通过分析实际情况发现，原始数据集在模型拟合上就已经取得了 0.94 左右的准确率，猜测由于降维技术并无法保留原数据集的所有信息，所以不仅没有提高正确率，反而由于部分信息丢失导致模型性能下降。这些都是需要根据具体的实验数据集和应用场景得到的结果，和理论相比跟家的真实，也能让我更加了解到实际的代码实现算法开发需要多注重实际处理的情况，不仅仅是做好算法复现即可。

实现第二三的实验时，都需要使用梯度下降优化模型参数，实现过程中，也接触到了许多课程内没有学习的知识，比如 KNN 中的马氏距离度量标准以及全连接神经网络中的 ReLU 激活函数。只有在手动实现的过程中，才能具体了解到学习率，训练轮数对于模型性能的影响，以及激活函数对于不同神经网络拟合能力的影响，这些都是课上所没有提及的，但在实验过程中，为了不断优化模型，自己去主动了解的重要知识。

可以说这三个实验首先便时让我对于降维技术，KNN 技术，神经网络的理论知识有了更加深入的了解，其次也拓宽了我的知识面了解了许多的数据处理，模型性能优化方法，最后也让我知道了实际情况千变万化，算法实现时需要切切实实的各种情况进行考虑，不能仅仅拘泥于书本知识。