模式识别实验报告

专业：人工智能

学号：

年级：22 级

姓名：

签名：

时间：

说明

本实验选择的课题为：数据降维与分类实验、KNN 分类实验和神经网络图像分类实验。

其中，前两个实验我在自己手动实现了相关算法后，mindspore 没有提供官方的可以直接使用的 PCA、LDA、KNN 算法来供我进行对比。于是我在前两个课题中仅使用了 mindspore.numpy 来进行相关运算，对比主体是调用了 sklearn 的成熟算法。

而神经网络的课题我则是自己纯手工实现了一个网络、基于 mindspore 实现了一个网络来进行对比的。

具体任务和代码的对应请查看每个课题的代码文件夹中的 README 文档。

目录

实验一 数据降维与分类实验 5
1. 问题描述 5
1.1 数据集介绍：红白葡萄酒 5
1.2 加分项数据集介绍：Iris 5
1.3 任务介绍与代码文件对应关系 5
2. 实现步骤与流程 6
2.1 实验原理与数学推导 6
（1）线性回归模型 6
（2）PCA 降维 7
（3）LDA 处理 7
（4）对比 LDA 与 PCA 8
2.2 具体实现 9
（1）热力图绘制与共线性检测 9
（2）超参数选取实验：保留主成分个数选取 9
（3）数据预处理 10
（4）纯手动实现 PCA、LDA 算法和对数几率回归模型 11
3. 实验结果与分析 13
3.1 任务一：自己手动实现 PCA、LDA、对数几率回归 13
3.2 任务二：调用 sklearn 的 PCA、LDA、对数几率回归并对比手写算法 15
3.3 任务三（加分项）：在 MNIST 数据集上实验 16
4. MindSpore 学习使用心得体会 16
实验二 KNN 分类任务实验 18
1. 问题描述 18
1.1 数据集介绍：鸢尾花数据集 18
1.2 加分项数据集介绍：breast_cancer 19
1.3 任务介绍与代码文件对应关系 19
2. 实现步骤与流程 20
2.1 实验原理与数学推导 20
（1）欧式距离 20
（2）马氏距离 20

（3）KNN 及其实现 20
3. 实验结果与分析 22
3.1 任务一：自己手动实现基于欧式距离的 KNN 22
3.2 任务二：自己手动实现基于马氏距离的 KNN 23
3.3 模型选择实验结果——k 值和距离度量选取结果 23
3.4 任务三：基于官方库实现基于欧式距离的 KNN 24
3.5 任务四（加分项）：在 breast_cancer 数据集上实验 25
实验三 神经网络图像分类实验 27
1. 问题描述 27
1.1 数据集介绍：cifar10 27
1.2 加分项数据集介绍：MNIST 数据集 27
1.3 任务说明与对应文件： 28
2. 实现步骤与流程 28
2.1 实验原理与数学推导 28
（1）手动实现全连接神经网络 28
（2）手动实现反向传播算法 28
（3）CNN 网络原理 29
（4）超参数选取实验：学习率 learning-rate 31
（5）超参数选取实验：训练轮数 epoch 31
（6）超参数选取实验：批处理大小 batch 32
3. 实验结果与分析 33
3.1 任务一：自己手动实现全连接神经 33
3.2 任务二：基于 mindspore 实现全连接神经 NN 和卷积神经网络 CNN 34
3.3 任务三（加分项）：在 MNIST 数据集上进行实验 35
4 MindSpore 学习使用心得体会 36
心得体会 37

实验一 数据降维与分类实验

1. 问题描述

本实验要求利用 LDA 和 PCA 两种降维技术对葡萄酒数据进行降维，并对降维前后的数据进行分类。

1.1 数据集介绍：红白葡萄酒

所给数据集包含两个.csv 文件，分别为红葡萄酒数据（包括 1599 个样本）和白葡萄酒数据（包含 4898 个样本），每个样本含有 11 个特征（文件的前 11 列）：固定酸度、挥发酸度、柠檬酸、残糖、氯化物、游离二氧化硫、总二氧化硫、密度、pH 值、硫酸盐、酒精和专家对此葡萄酒的打分（文件的最后一列）。

以上是实验要求中的原文，事实上文件最后一列`quality`也是一个特征，也就是说每个样本具有 12 个属性，这 12 个均为连续值，无需进行离散值属性相关的编码操作；1 个标签，即红/白。

1.2 加分项数据集介绍：IRIS

Iris 数据集包含 150 条记录，每条记录代表一朵鸢尾花的四个特征及其对应的类别。这四个特征是：
Sepal Length（花萼长度）：以厘米为单位测量。
Sepal Width（花萼宽度）：以厘米为单位测量。
Petal Length（花瓣长度）：以厘米为单位测量。
Petal Width（花瓣宽度）：以厘米为单位测量。
类别：
Iris 数据集包含三类鸢尾花，每类 50 条记录。

1.3 任务介绍与代码文件对应关系

- `PCA-LDA-task1-manual`：对应任务一，包含纯手动实现对数几率回归模型、

LDA 算法、PCA 算法，超参数主成分个数选择的实验以及在数据集上的训练、测试结果。

- `PCA-LDA-task2-mindspore`：对应任务二，由于 mindspore 没有较为好用的机器学习算法，这里选择 sklearn 实现回归模型、LDA 算法、PCA 算法并对比手动实现在数据集上的性能。

- （加分项）`PCA-LDA-Bonus-MNIST`：对应任务三加分项，使用 scikit-learn 中加载手写数字上对比手动实现算法和官方算法的性能。

2. 实现步骤与流程

2.1 实验原理与数学推导

（1）线性回归模型

Sigmoid 函数为：

$$y = \frac{1}{1 + e^{-z}}$$

将$z = w^T x + b$代入其中：

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

令:

$$p(y = 1|x) = y$$

$$p(y = 0|x) = 1 - y$$

可以得到：

$$w^T x + b = \ ln\frac{p(y = 1|x)}{p(y = 0|x)}$$

进一步解得：

$$\begin{cases} p(y = 1|x) = \dfrac{e^{w^T x + b}}{1 + e^{w^T x + b}} \\ p(y = 0|x) = \dfrac{1}{1 + e^{w^T x + b}} \end{cases}$$

增广特征矩阵：$\hat{x} = (x; 1)$,稀疏矩阵：$\hat{w} = (w; b)$。进行极大似然估计：

$$L(B)=\sum_{i=1}^{n} y_i lnp(y=1|\hat{x}_i)+(1-y_i)lnp(y=0|\hat{x}_i)$$

$$=\sum_{i=1}^{n} y_i \hat{w}^T \hat{x}_i-\ln\left(1+e^{\hat{w}^T \hat{x}_i}\right)$$

对其求一阶导数：

$$\frac{\partial L}{\partial \hat{w}}=-\sum_{i=1}^{n} \hat{x}_i(y_i-\hat{y_i})$$

（2）PCA 降维

m 维随机变量$y=(y_1,y_2,\dots,y_m)^T$的分量依次是x的第一主成分到第 m 主成分的充分必要条件是：

1. $y=A^T x, A$*是正交矩阵*
2. $cov(y)=diag(\lambda_1,\lambda_2,\dots,\lambda_m),\lambda_1\geq\lambda_2\geq\cdots\geq\lambda_m$, *n 是样本总体协方差的第 k 个特征值*

获得主成分随机变量y后，再考察方差贡献率。第 k 主成分y_k的方差贡献率定义为其的方差和所有方差之和的比，记作η_k：

$$\eta_k=\frac{\lambda_k}{\sum_{i=i}^{m}\lambda_i}$$

k 个主成分的累计方差贡献率定义为 k 个方差之和与所有方差之比：

$$\sum_{i=1}^{k}\eta_k=\frac{\sum_{i=1}^{k}\lambda_i}{\sum_{i=i}^{m}\lambda_i}$$

通常取 k 使得累积方差贡献率达到规定的百分比以上，如 70%~80%以上。累积方差贡献率反映了主成分保留信息的比例，但它不能反应某个原有变量x_i保留信息的比例，这时通常利用 k 个主成分$y_1,y_2,\dots,y_k$对原有变量x_i的贡献率。

（3）LDA 处理

LDA 算法的思想是将数据投影到低维空间之后，而 PCA 是为了最大化方差。在给定数据集$\{(x_i,y_i)\}i=1\sim m$中：

第i类示例的集合X_i

第i类示例的均值向量μ_i

第i类示例的协方差矩阵Σ_i

两类样本的中心在直线上的投影：$w^T\mu_0$和$w^T\mu_1$

两类样本的协方差：$w^T\Sigma_0 w$和$w^T\Sigma_1 w$

同样样例的投影点尽可能接近：$w^T\Sigma_0 w + w^T\Sigma_1 w$尽可能小

异样样例的投影点尽可能原理：$||w^T\mu_0 - w^T\mu_1||$

于是，最大化：

$$J = \frac{||w^T\mu_0 - w^T\mu_1||}{w^T\Sigma_0 w + w^T\Sigma_1 w} = \frac{w^T(\mu_0 - \mu_1)(\mu_0 - \mu_1)^T w}{w^T(\Sigma_0 - \Sigma_1)w}$$

类内散度矩阵（within-class scatter matrix）

$$S_w = \Sigma_0 + \Sigma_1 = \sum_{x\epsilon X_n}(x - \mu_0)(x - \mu_0)^T + \sum_{x\epsilon X_1}(x - \mu_1)(x - \mu_1)^T$$

类间三度矩阵（between-class scatter matrix）

$$S_b = (\mu_0 - \mu_1)(\mu_0 - \mu_1)^T$$

LDA 的目标：最大化广义瑞利商（generalized Rayleigh quotient）

$$J = \frac{w^T S_b w}{w^T S_w w}$$

（4）对比 LDA 与 PCA

LDA 用于降维，和 PCA 有很多相同，也有很多不同的地方，因此值得好好的比较一下两者的降维异同点。

相同点

1）两者均可以对数据进行降维。

2）两者在降维时均使用了矩阵特征分解的思想。

3）两者都假设数据符合高斯分布。

不同点

1）LDA 是有监督的降维方法，而 PCA 是无监督的降维方法

2）LDA 降维最多降到类别数 k-1 的维数，而 PCA 没有这个限制。

3）LDA 除了可以用于降维，还可以用于分类。

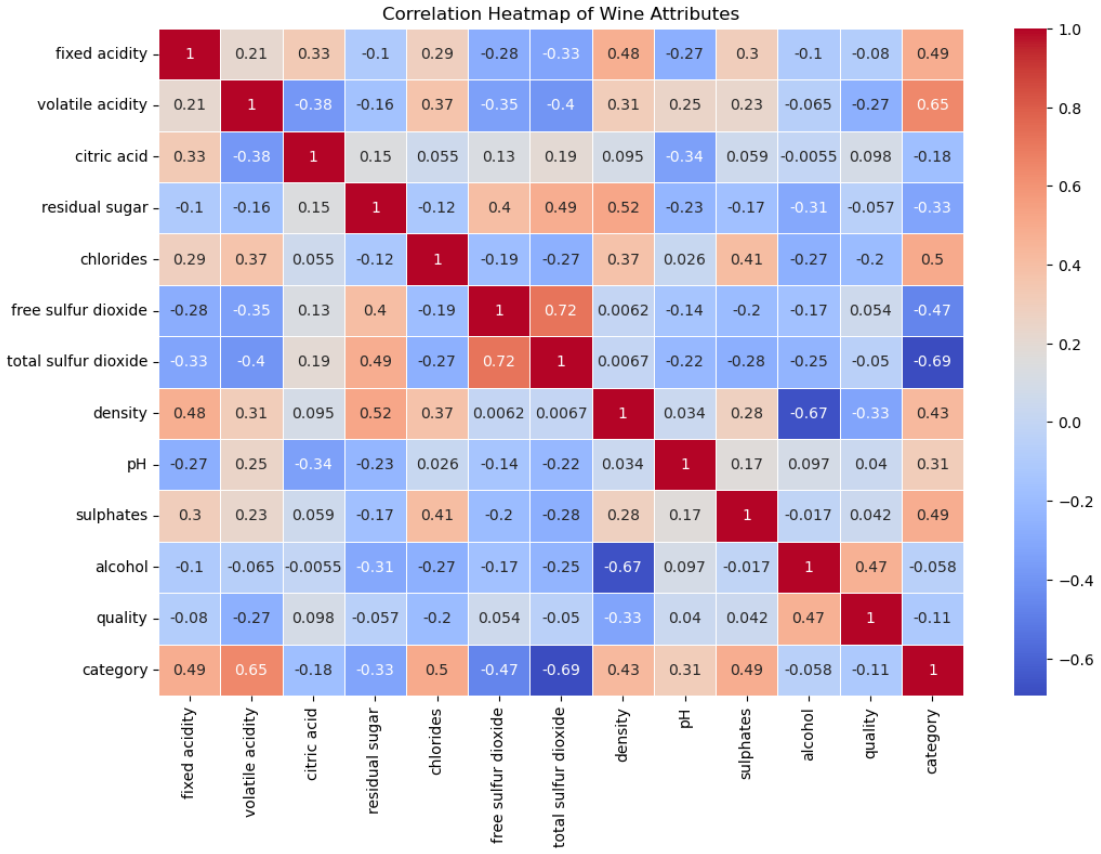
4）LDA 选择分类性能最好的投影方向，而 PCA 选择样本点投影具有最大方差的方向。这点可以从下图形象的看出，在某些数据分布下 LDA 比 PCA 降维较优。

2.2 具体实现

（1）热力图绘制与共线性检测

共线性(Collinearity)指的是在拟合线性回归模型的过程中两个或两个以上的 prediction 互相之间存在较强的相关关系。当自变量之间存在共线性时，模型的参数会变得极其不稳定，模型的预测能力会下降。很难确切区分每个自变量对因变量影响，因此增强了对于模型结果的预测成本。

而 PCA 主成分分析就是很好的解决共线性的问题。在本实验中运用 seaborn 库的工具绘制属性热力图，横纵属性之间共线性越强，对应交叉模块的颜色越深。得到如图的结果：



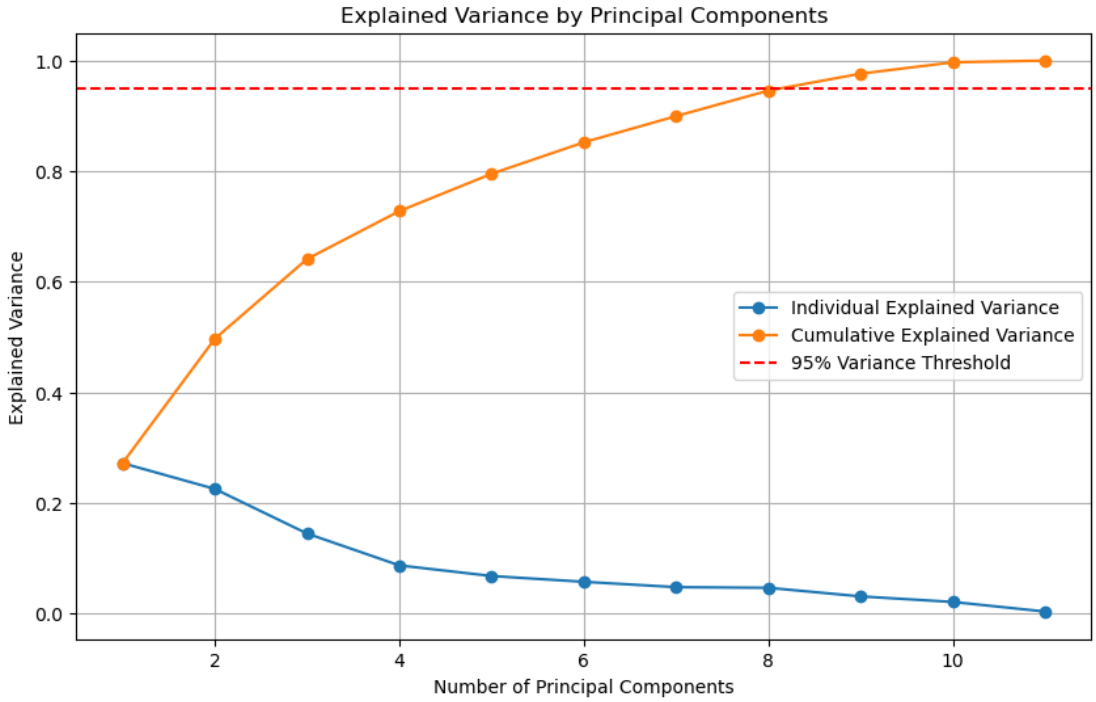
可以发现属性`total sulfur dioxide`和`free sulfur dioxide`等属性之间有较强的共线性。因此进行 PCA 处理是有一定必要的。

（2）超参数选取实验：保留主成分个数选取

在这次实验中，我没有选择直接根据经验确定主成分个数，而是通过使用训练

数据集中的连续值属性来拟合 PCA 模型。PCA 会提取与特征数量相同的主成分，目的是为了分析和决定应该保留多少主成分。

通过绘制图表，展示了每个主成分的解释方差比率以及前 11 个主成分的累积解释方差比率。这有助于了解每个主成分对原始数据变异性的贡献，以及一共需要多少主成分才能达到满意的解释程度（95%的累积解释方差）。



基于累积解释方差比率的分析，**确定保留的主成分数量为 9**。这意味着选择了前 9 个主成分来代表原始数据集中的大部分信息。

```
...   Number of principal components to retain 95% variance: 9
```

（3）数据预处理

【去重】

```
';').drop_duplicates().assign(category=0)
.drop_duplicates().assign(category=1)
```

重复的数据行可能会导致数据分析结果的偏差，在机器学习中，重复数据可能会对模型训练产生不利影。而在本实验所用的线性模型中，模型假定了样本之间是彼此独立且互异的，如果样本重复，则计算参数中的自由度就会降低。

通过`.drop_duplicates()`进行去重操作，共删除了 1177 个样本数据

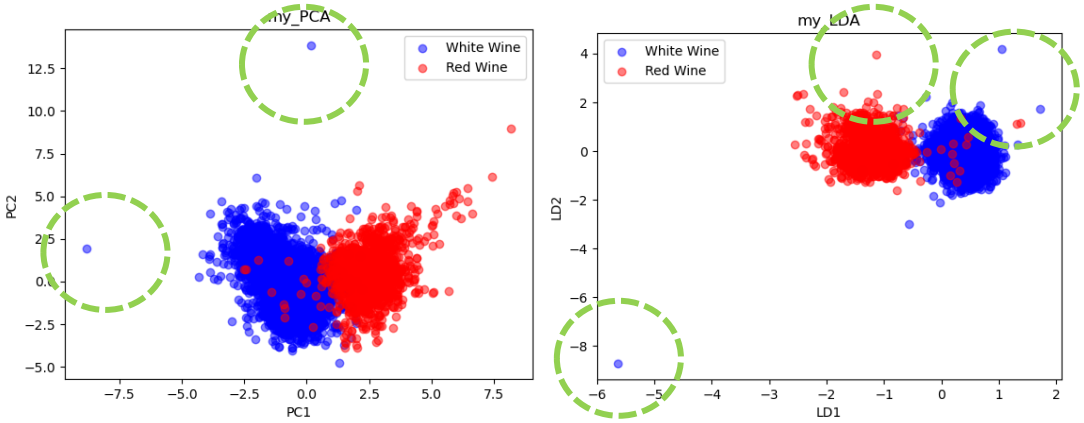
【标准化】

数据标准化是最常见的数据预处理技术之一，通过将数据集中在一个区间上，可以加速模型收敛、提高模型性能。

```
# 标准化数据
X_mean = np.mean(X, axis=0)
X_std = np.std(X, axis=0)
X = (X - X_mean) / X_std
```

【离群值处理】

在此步处理前我绘制的 PCA、LDA 降维后散点图为：



可以发现有明显的离群值现象。这样的数据很可能是错误数据，会对模型的拟合造成干扰，应当用适当的操作进行筛查并删除：

```
mask = np.all(np.abs(X) <= 3, axis=1)
X = X[mask]
y = y[mask]
```

如图处理离群值，去除超过 3 个标准差的数据点。

（4）纯手动实现 PCA、LDA 算法和对数几率回归模型

根据 2.1 实验原理部分的数学推导，我基于 numpy 和 mindspore 的一些基础库纯手动实现了 PCA、LDA 算法和 logisticRegression 模型：

```python
class PCA:
    def __init__(self, n_components):
        """
        初始化PCA模型。

        参数:
        n_components (int): 降维后的维数
        """
        self.n_components = n_components
```

```python
class LDA:
    def __init__(self, n_components):
        """
        初始化LDA模型。

        参数:
        n_components (int): 降维后的维数
        """
        self.n_components = n_components
```

```python
class LogisticRegression:
    def __init__(self, lr=0.01, num_iter=1000):
        self.lr = lr
        self.num_iter = num_iter
        self.history = []
```

根据超参数实验选取 n_components=9，对原始数据、预处理后的数据、PCA 降维后的数据、LDA 降维后的数据（已通过 train_test_spilt 划分为训练集和测试集）进行分类，输出准确率和可视化结果。

（5）调用 sklearn 库中的 PCA、LDA 和 LogisticRegression

重复手动部分的训练与测试。并对比其与（4）中自己手动实现的算法的性能。

（6）换用 MNIST 数据集

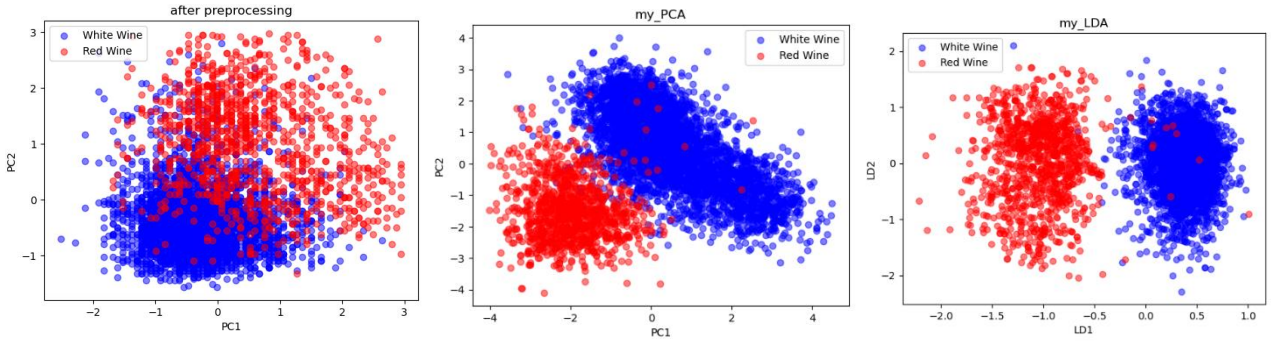
```python
digits = load_digits()
X = scaler.fit_transform(digits.data)
y = digits.target
```

`from sklearn.datasets import load_digits`通过 sklearn 提供的方法加载手写数字数据集 MNIST，并分别利用自己实现的 PCA、LDA 和回归模型，sklearn 库中的 PCA、LDA 和回归模型训练和测试，对比结果。

3. 实验结果与分析

3.1 任务一：自己手动实现 PCA、LDA、对数几率回归

分别绘制了简单预处理（包括离散值处理）后、PCA 降维后、LDA 降维后的散点图：



预处理前、预处理后、PCA 降维后、LDA 降维后的准确率：

```
原始数据（预处理前）的准确率: 0.9257518796992481
原始数据（预处理后）的准确率: 0.9887410440122825
```

```
PCA后的准确率: 0.9907881269191402
```

```
LDA后的准确率: 0.9825997952917093
```

【详细分析】

在预处理后但未进行降维的数据中，红白葡萄酒样本混杂在一起。而预处理使得准确率由 0.9258 升至 0.9887，这个显著提升表明预处理步骤在提高模型性能方面非常有效，通过确保特征在相似的尺度上，并**减少噪声数据对模型的影响**，提升了准确率。

PCA 将高维数据降维到 PC1 和 PC2 上，可以从图中看出红白葡萄酒由**较好的分离**；而其准确率是 0.9908，比仅仅预处理后的准确率还要略高，表明 PCA 在降低数据维度的同时保持了数据的主要信息，有助于提高模型的泛化能力。

而经过了 LDA 处理后的数据，通过找到最大化类别间差异并最小化类别内差异的投影方式，明显可以看到红白葡萄酒在 LDA 和 LD1 和 LD2 轴上有比 PCA 效果更好的分离。其准确率 0.9826 略低于 PCA 结果，但仍然较高，表明 LDA 在区分两类酒上也有较好的效果。

对比 PCA 和 LDA，**PCA 虽然降维了数据，但并没有丢失重要的信息，反而将模型的预测准确性提升了**；LDA 则在分类任务中展现了其**强大的特征分离能力**。

【为什么 LDA 比简单预处理准确率略低？】

回顾 LDA 的原理，我发现其假设数据服从正态分布，且各类别共享相同的协方差矩阵。如果数据不完全满足这些假设，LDA 的性能可能会受到影响。虽然 LDA 在数据分离上表现出色，但其假设条件限制了其在某些数据集上的表现。

与此同时，尽管降维方法有助于提高分类性能，但有时可能会丢失一些重要的特征信息，导致准确率下降。显然在这点上，本任务中的 PCA 表现更好。

3.2 任务二：调用 sklearn 的 PCA、LDA、对数几率回归并对比手写算法

手动实现的结果

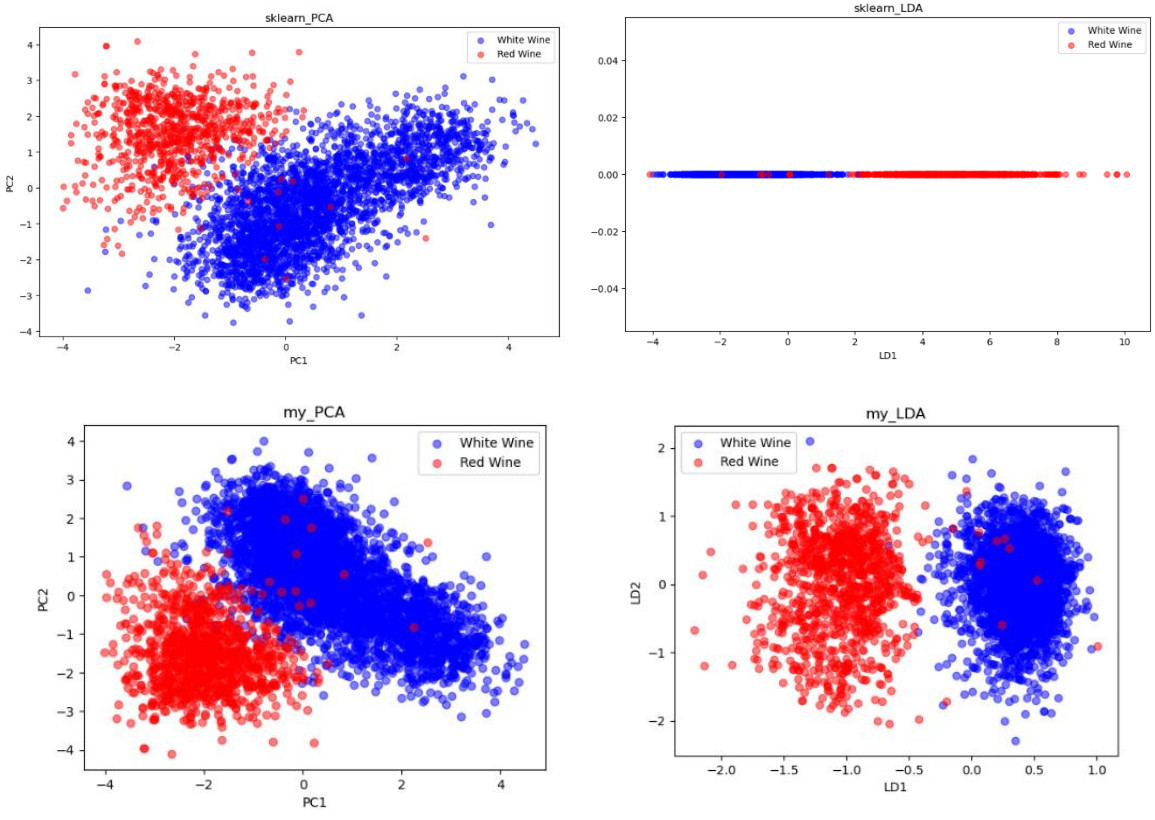
1. **预处理前准确率**: 0.9257518796992481
2. **预处理后准确率**: 0.9887410440122825
3. **PCA后的准确率**: 0.9907881269191402
4. **LDA后的准确率**: 0.9825997952917093

`sklearn` 实现的结果

1. **预处理前准确率**: 0.981203007518797
2. **预处理后准确率**: 0.9938900203665988
3. **PCA后的准确率**: 0.9928425357873211
4. **LDA后的准确率**: 0.9969325153374233

仅仅比较预处理前的准确率，仍可以发现 sklearn 库中的回归模型效果更好，因此 sklearn 在数据处理方面更加优化和稳定。
由此也可以发现，良好的数据预处理，即使不进行特征方面的特殊处理，也可以极大地提高模型的泛化和预测能力。

而 LDA 处理后的结果上，sklearn 实现的明显高于手动实现的，表明 sklearn 的 LDA 算法在保持数据特征和分离能力方面更强大。



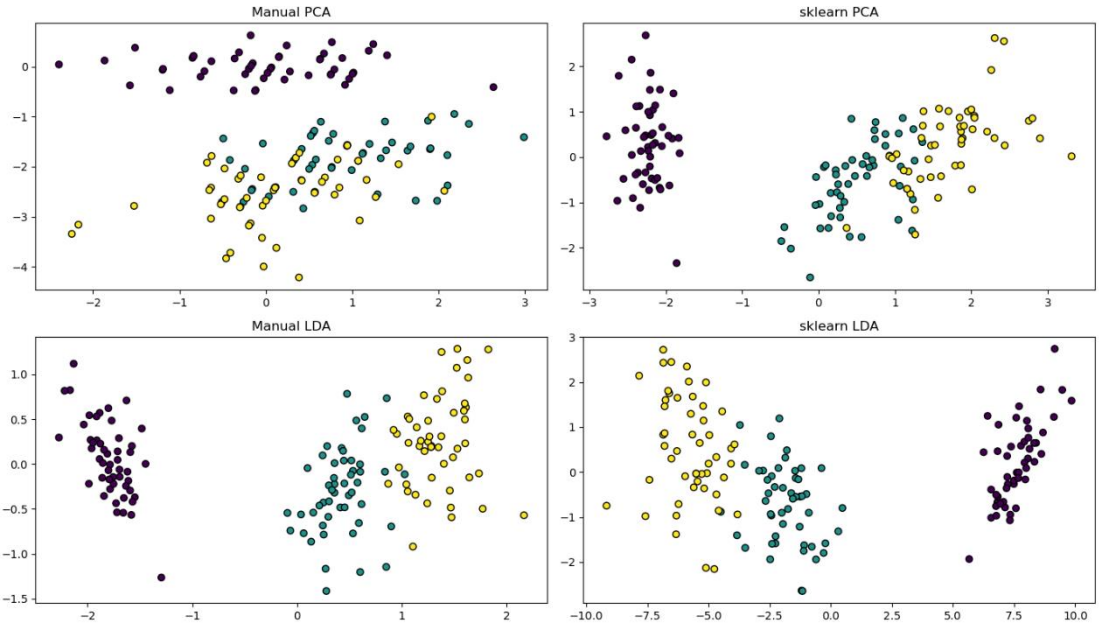
通过对比可以看出，sklearn 实现的 PCA 和 LDA 方法在准确率和图像表现上都优于手动实现。这可能是因为 sklearn 库中算法经过高度优化，能更好地处理数据，并且在特征选择和模型训练方面更加稳定和高效。

这可能是 sklearn 在 PCA 中使用了奇异值分解带来了更好的数据值稳定性。

3.3 任务三（加分项）：在 MNIST 数据集上实验

【降维效果对比】

分别绘制使用自己实现的算法和 sklearn 中算法在 iris 数据集上降维的情况。通过对比，可以发现，sklearn 的 PCA 算法明显可以把数据分的更开，而 LDA 算法二者差别较小。



【分类效果对比】

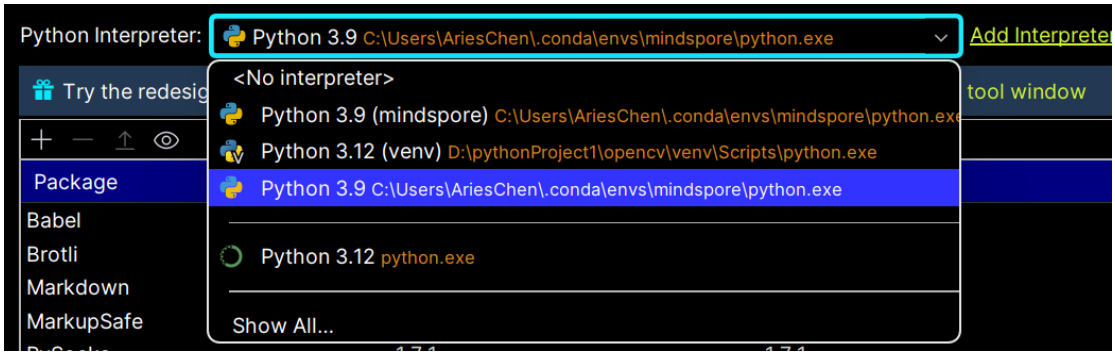
```
手动 PCA 的分类准确率: 0.8667
sklearn PCA 的分类准确率: 0.9111
手动 LDA 的分类准确率: 1.0000
sklearn LDA 的分类准确率: 1.0000
```

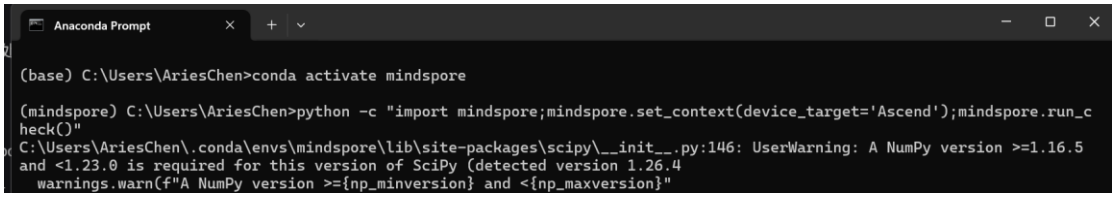
分别使用自己实现和 sklearn 实现的 PCA、LDA 算法对数据集降维然后使用线性回归模型分类。在多分类任务中，sklearn 算法显著优于手写算法。

4. MindSpore 学习使用心得体会

初次安装和配置 MindSpore 时，我感到它的环境配置相对复杂，特别是在使用 GPU 或 Ascend 硬件时，需要仔细阅读官方文档并满足各种依赖。

版本问题也相对棘手，我原本的 python 是 3.12，而 mindspore 只支持 7-9，于是我重新在我的电脑里配置了 3.9 的 python 与 mindspore 进行兼容。为了防止其它库的版本与其产生冲突，我专门开辟了一个虚拟环境来进行基于 mindspore 的实验。





但当一切配置完成后，运行起来非常顺畅，官方文档和教程也十分详细，为初学者提供了很多帮助。MindSpore 在模型构建和训练方面的灵活性让我非常满意。它提供了类似 Keras 和 PyTorch 的高级 API，使得模型的定义和训练变得直观且高效。

希望 mindspore 尽快兼容更新版本的 python，并解决它和我环境中的 numpy、pandas 冲突的问题。

实验二 KNN 分类任务实验

1. 问题描述

本实验要求利用 KNN 对 Iris 鸢尾花书记局中的测试集进行分类。

1.1 数据集介绍：鸢尾花数据集

Iris 数据集包含 150 条记录，每条记录代表一朵鸢尾花的四个特征及其对应的类别。这四个特征是：

Sepal Length（花萼长度）：以厘米为单位测量。
Sepal Width（花萼宽度）：以厘米为单位测量。
Petal Length（花瓣长度）：以厘米为单位测量。
Petal Width（花瓣宽度）：以厘米为单位测量。

类别
Iris 数据集包含三类鸢尾花，每类 50 条记录。这三类分别是：

Iris-setosa
Iris-versicolor
Iris-virginica

数据集已被划分为训练集、验证集和测试集，分别存储于 data 文件夹中的 train.csv, val.csv, test.csv。train.csv 和 val.csv 文件包含 data, label 字段，分别存储着特征$X \in R^{N\times d}$和标记$Y \in R^{N\times 1}$。其中，N是样例数量，d= 4 为特征维度，每个样例的标记$y \in \{0,1,2\}$。test.csv 文件则仅包含 data 字段。

这三类鸢尾花在特征空间中的分布有所不同，使得它们在某些维度上有较好的可分性。

1.2 加分项数据集介绍：breast_cancer

Breast_cancer 数据集是机器学习领域中常用的经典数据集之一，它通常用于分类任务的训练和测试。这个数据集是关于乳腺癌诊断的数据，其中包含了患者的一些生理特征信息，以及他们是否患有良性或恶性乳腺肿瘤的标签。

特征数量：数据集包含了 30 个特征。
样本数量：数据集包含了 569 个样本。
特征类型：特征主要是关于细胞核的一些特征，如细胞核的大小、形状、质地等。
类别：目标变量是乳腺肿瘤的性质，分为良性（benign）和恶性（malignant）两类。

本实验中利用的数据集是我在官网上下载的.csv 形式，存于代码同文件夹下。

1.3 任务介绍与代码文件对应关系

- `KNN-task1-task2-manual`：对应任务一和任务二，包含纯手动实现基于欧式距离和马氏距离的 knn 算法，k 值选择的实验（留出法、5 折交叉验证法）以及在数据集上的训练、测试结果。
- `KNN-task3-mindspore`：对应任务三，使用 mindspore.numpy 进行矩阵运算，选择 sklearn 实现基于欧式距离的 knn 算法，k 值选择实验、并在数据集上进行训练、测试。
- （加分项）`KNN-Bonus-breastcancer`：对应任务三加分项，在乳腺癌肿瘤数据集上对比自己实现的算法和官方代码。

2. 实现步骤与流程

2.1 实验原理与数学推导

（1）欧式距离

$$d_E(\boldsymbol{x},\boldsymbol{y}) = \|\boldsymbol{x}-\boldsymbol{y}\|_2 = \sqrt{\sum_i (x_i - y_i)^2}$$

（2）马氏距离

马氏距离是一种用于衡量样本间相似度的距离度量方法，它考虑了特征之间的相关性。在统计学和机器学习中，马氏距离通常用于分类、聚类以及异常检测等任务中。

假设有一个样本集合$X = \{x_1, x_2, \dots, x_n\}$, 每个样本都是一个$d$维的向量。而协方差矩阵$S$表示了样本集合的协方差结构。马氏距离$D_M(x, y)$ 用于衡量两个样本x和y之间的相似度，公式如下：

$$D_M(x, y) = \sqrt{(x-y)^T S^{-1}(x-y)}$$

其中，$(x-y)$是样本之间的差异向量，S^{-1}是协方差矩阵的逆矩阵。

（3）KNN 及其实现

【距离度量】

样本点间的距离，本实验选用欧氏距离和马氏距离。

【构建 kd 树】

输入：k 维空间数据集$T = \{x_1, x_2, \dots, x_n\}$，其中$x_i = \{x_i^{(1)}, x_i^{(2)}, \dots, x_i^{(k)}\}$

输出：kd 树

初始化：

构造根节点，根节点对应于包含T的 k 维空间的超矩形区域。选择$x^{(1)}$为坐标轴，以T中所有实例的$x^{(1)}$的中位数为切分点，将根节点对应的超矩形区域切分为两个子区域。切分由通过切分点并与坐标$x^{(1)}$垂直的超平面实现。

由根节点生成深度为 1 的左右子节点：左节点对应坐标$x^{(1)}$小于切分点的子区域，右节点对应坐标$x^{(1)}$于切分点的子区域。

循环：

对深度为 j 的节点，选择$x^{(l)}$为切分的坐标轴，$l = j(mod\ k) + 1$以该节点的区域中的所有实例的$x^{(l)}$坐标的中位数作为切分点，将该节点对应的超矩形区域切分为两个子区域。左右子树所对应区域同上

终止：

两个子区域没有实例时停止划分。

【交叉验证选择 k 值】

在许多实际应用中数据是不充足的。为了选择好的模型，可以采用交叉验证方法。交叉验证的基本想法是重复地使用数据，把给定的数据进行切分，将切分的数据组合为训练集与测试集，在此基础上反复进行训练测试以及模型的选择。

由 KNN 的原理可知，k 值的选择直接影响了模型的复杂度。较小的 k 值会使模型更复杂，因为它会更多地受到噪声的影响，可能导致过拟合。而较大的 k 值则会使模型更简单，因为它会更多地考虑到整体数据的趋势，可能导致欠拟合。

特别说明：

本次实验数据集其实已经提供好了验证集，可以直接进行不同 k 值的选择测试，类似于留出法（下文统称为留出法）。但由于我想在本次实验中更加熟悉 k 折交叉验证的方法，所以我又手动实现了一个 5 折交叉验证的算法，在训练集上进行 k 值的选择比较实验。结果都呈现在 notebook 中了。

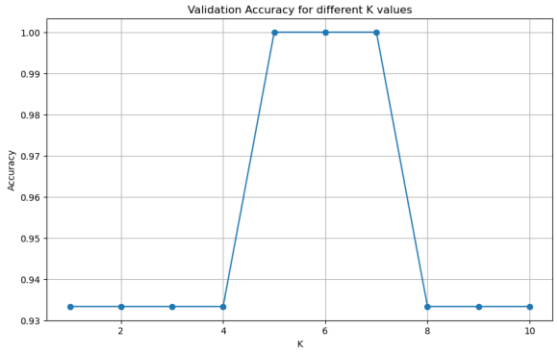
```python
# 使用留出法进行评估
def validate(X_train, y_train, X_val, y_val, k_values):
    val_accuracies = []

    for k in k_values:
        y_val_pred = knn_euclidean(X_train, y_train, X_val, k)
        acc = accuracy(y_val, y_val_pred)
        val_accuracies.append(acc)
        print(f'Validation accuracy for k={k}: {acc:.4f}')

    return k_values[np.argmax(val_accuracies)], val_accuracies
```

```python
# 手动实现5折交叉验证
def cross_validate(X, y, k_values, num_folds=5):
    fold_size = len(X) // num_folds
    indices = np.arange(len(X))
    np.random.shuffle(indices)
    average_accuracies = []

    for k in k_values:
        fold_accuracies = []

        for fold in range(num_folds):
            val_indices = indices[fold * fold_size:(fold +
```

3. 实验结果与分析

3.1 任务一：自己手动实现基于欧式距离的 KNN

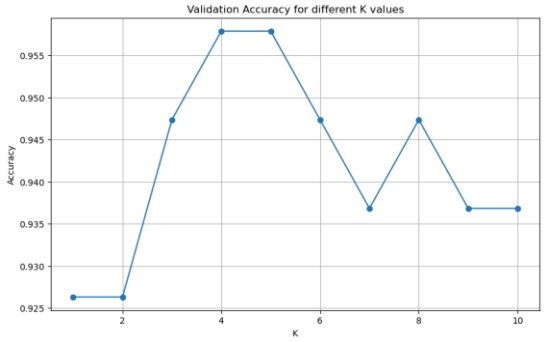
【运用已给的独立验证集】




```
Validation accuracy for k=1: 0.9333
Validation accuracy for k=2: 0.9333
Validation accuracy for k=3: 0.9333
Validation accuracy for k=4: 0.9333
Validation accuracy for k=5: 1.0000
Validation accuracy for k=6: 1.0000
Validation accuracy for k=7: 1.0000
Validation accuracy for k=8: 0.9333
Validation accuracy for k=9: 0.9333
Validation accuracy for k=10: 0.9333
Best K value: 5
```

在给定的验证集上进行测试，选取的最优 k 值为 5，达到验证集 100%准确率。

【在训练集上运用 5 折交叉验证】




```
Average accuracy for k=1: 0.9263
Average accuracy for k=2: 0.9263
Average accuracy for k=3: 0.9474
Average accuracy for k=4: 0.9579
Average accuracy for k=5: 0.9579
Average accuracy for k=6: 0.9474
Average accuracy for k=7: 0.9368
Average accuracy for k=8: 0.9474
Average accuracy for k=9: 0.9368
Average accuracy for k=10: 0.9368
Best K value: 4
```

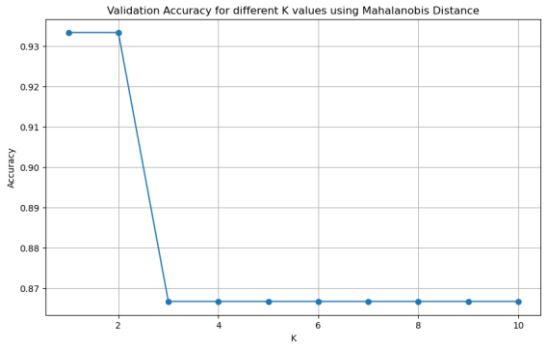
在所给的训练集上进行 5 折交叉验证，选取的最优 k 值为 4，达到验证集 95.6%的准确率。

独立验证集上 k=5 和 k=6 的 100%准确率可能表示模型对特定的验证集过拟合。交叉验证提供了更全面的评估，显示出 k=4 的性能在多个数据子集上都表现出色。尽管独立验证集显示 k=5 最佳，但综合考虑交叉验证的稳定性，最终我还是选择 k=4 作为模型的最佳 k 值。

因此选择输出 k=4 时的测试集标签到`task1_test_prediction.csv`文件中。

3.2 任务二：自己手动实现基于马氏距离的 KNN

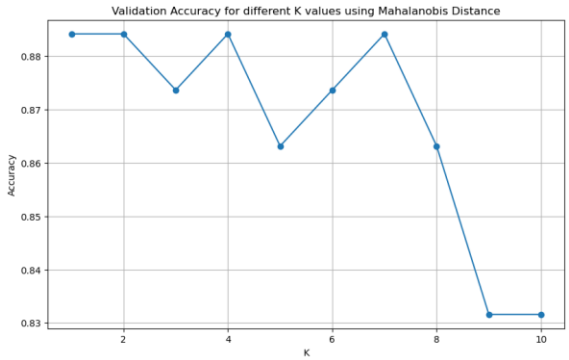
【运用已给的独立验证集】




```
Validation accuracy for k=1: 0.9333
Validation accuracy for k=2: 0.9333
Validation accuracy for k=3: 0.8667
Validation accuracy for k=4: 0.8667
Validation accuracy for k=5: 0.8667
Validation accuracy for k=6: 0.8667
Validation accuracy for k=7: 0.8667
Validation accuracy for k=8: 0.8667
Validation accuracy for k=9: 0.8667
Validation accuracy for k=10: 0.8667
Best K value: 1
```

在给定的验证集上进行测试，选取的最优 k 值为 1，达到验证集 93.3%准确率。k=1 时，模型表现最好，可能是因为在独立验证集中，单个最近邻居能够很好地表示数据分类。较大的 k 值表现较差，说明使用更多邻居反而使得分类结果变差，可能是由于不同类别样本的相互干扰。

【在训练集上运用 5 折交叉验证】




```
Average accuracy for k=1: 0.8842
Average accuracy for k=2: 0.8842
Average accuracy for k=3: 0.8737
Average accuracy for k=4: 0.8842
Average accuracy for k=5: 0.8632
Average accuracy for k=6: 0.8737
Average accuracy for k=7: 0.8842
Average accuracy for k=8: 0.8632
Average accuracy for k=9: 0.8316
Average accuracy for k=10: 0.8316
Best K value: 1
```

在所给的训练集上进行 5 折交叉验证，选取的最优 k 值仍然为 1，达到验证集 88.42%的准确率。

因此选择输出 k=1 时的测试集标签到`task2_test_prediction.csv`文件中。

3.3 模型选择实验结果——k 值和距离度量选取结果

从以上结果中可以看到，不同距离度量和验证方法下，k-NN 算法的性能表现有所不同。

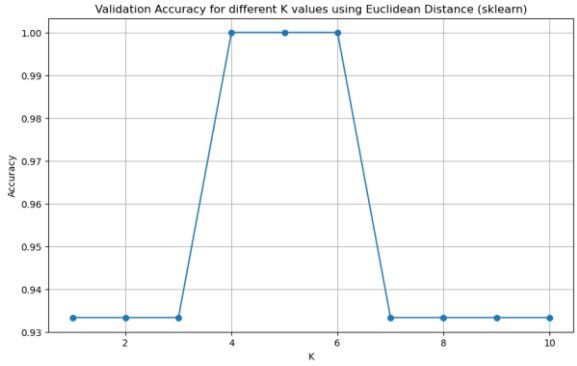
独立验证集能够显示模型在独立数据集上的直接表现，反映模型的实际泛化能力。交叉验证提供了对模型稳定性的综合评估，通过不同数据子集的验证，能更全面

地评估模型性能。

根据独立验证集的结果，使用欧式距离时 k=5 或 k=6 是最佳选择；根据交叉验证的结果，使用欧式距离时 k=4 是最佳选择。综合考虑，欧式距离在不同验证方法下都表现出色，且交叉验证的结果更具稳定性，因此建议选择**欧式距离下的 k=4 或 k=5 作为最佳 k 值。下面在调用官方库时，我便选择了这个欧式距离作为距离度量。**

3.4 任务三：基于官方库实现基于欧式距离的 KNN

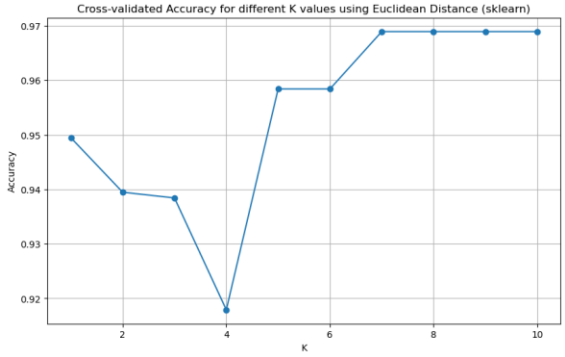
【运用已给的独立验证集】




```
Validation accuracy for k=1: 0.9333
Validation accuracy for k=2: 0.9333
Validation accuracy for k=3: 0.9333
Validation accuracy for k=4: 1.0000
Validation accuracy for k=5: 1.0000
Validation accuracy for k=6: 1.0000
Validation accuracy for k=7: 0.9333
Validation accuracy for k=8: 0.9333
Validation accuracy for k=9: 0.9333
Validation accuracy for k=10: 0.9333
Best K value: 4
```

使用 sklearn 的 KNN，在给定的验证集上进行测试，选取的最优 k 值为 4，达到验证集 100%准确率。与任务一中手动实现的结果相近。

【在训练集上运用 5 折交叉验证】




```
Cross-validated accuracy for k=1: 0.9495
Cross-validated accuracy for k=2: 0.9395
Cross-validated accuracy for k=3: 0.9384
Cross-validated accuracy for k=4: 0.9179
Cross-validated accuracy for k=5: 0.9584
Cross-validated accuracy for k=6: 0.9584
Cross-validated accuracy for k=7: 0.9689
Cross-validated accuracy for k=8: 0.9689
Cross-validated accuracy for k=9: 0.9689
Cross-validated accuracy for k=10: 0.9689
Best K value: 7
```

在所给的训练集上进行 5 折交叉验证，选取的最优 k 值为 7，达到验证集 96.89%的准确率。与任务一中手动实现的结果也相近。

最终选择 k 值为 4 作为超参数对测试集进行预测，将结果输入`task3_test_prediction.csv`文件中。

3.5 任务四（加分项）：在 breast_cancer 数据集上实验

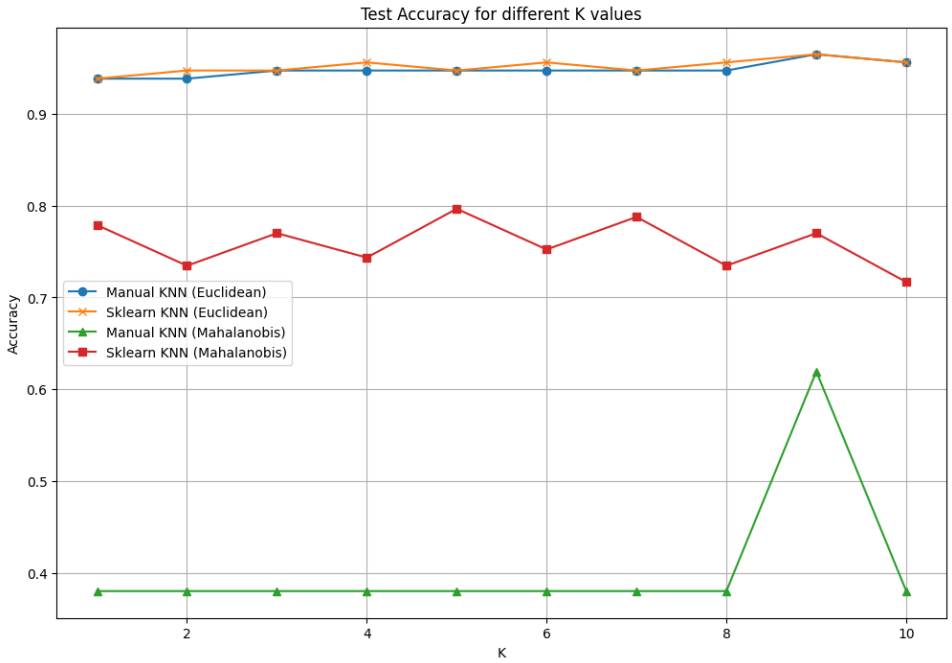
分别在基于手动欧式距离、手动马氏距离的手动 KNN 算法和 sklearn 实现的欧式距离、马氏距离 KNN 算法上进行 breast_cancer 数据集预测。

【对比算法】

手动实现 KNN (Euclidean) 和 sklearn 实现 KNN (Euclidean) 的表现几乎相同，在不同的 k 值下准确率都保持在 0.92 到 0.93 之间。

这种一致性表明，手动实现和 sklearn 实现的算法逻辑是一致的，欧式距离在这个数据集上表现稳定。

手动实现的马氏距离准确率与 sklearn 实现的有较大差距，表明在计算马氏距离时可能存在错误。只有在 k=8 时准确率异常高，可能是由于数据集的特定分布导致的偶然现象，但这进一步表明手动实现存在潜在问题。



【对比距离度量方式】

马氏距离考虑了数据特征之间的相关性，因此在某些情况下能够比欧式距离提供更准确的结果。然而，**如果数据特征之间的相关性较低或协方差矩阵接近奇异**，那么马氏距离的效果可能不如欧式距离。

在这个数据集中，特征之间的相关性可能并不足以显著提升分类效果，甚至可能由于协方差矩阵的计算误差导致准确率降低。

4. MindSpore 学习使用心得体会

在整个 KNN 实验过程中，MindSpore 的 NumPy 模块始终表现稳定，计算速度快且结果准确。特别是在处理大规模数据时，我能明显感受到其性能优势。这种流畅的体验让我的实验过程更加高效。

使用 MindSpore 的 NumPy 模块让我感受到了一种熟悉而又高效的计算体验。我个人认为对于希望提升数值计算性能的从业者，MindSpore 无疑是一个值得尝试的选择。

实验三 神经网络图像分类实验

1. 问题描述

本实验要求利用神经网络算法对 cifar10 数据集进行分类。

1.1 数据集介绍：cifar10

CIFAR-10 是计算机视觉领域中的一个重要的数据集。 原始数据集分为训练集和测试集，其中训练集包含 50000 张、测试集包含 30000 张图像。
这些图片共涵盖 10 个类别：飞机、汽车、鸟类、猫、鹿、狗、青蛙、马、船和卡车，高度和宽度均为 32 像素并有三个颜色通道（RGB）。

我的数据集是从官网下载的，包括 5 个 batch 的训练集和 1 个测试集。

1.2 加分项数据集介绍：MNIST 数据集

MNIST 是著名的手写体数字识别数据集。该数据集由训练数据集和测试数 据集两部分组成，其中训练数据集包含了 60000 张样本图片及其对应标签，每 张图片由 28×28 的像素点构成；测试数据集包含了 10000 张样本图片及其对应标签，每张图片由 28×28 的像素点构成。

该数据集我通过 mindspore 提供的 dataset 下载获取。

```
from download import download

url = "https://mindspore-website.obs.cn-north-4.myhuaweicloud.com/" \
      "notebook/datasets/MNIST_Data.zip"
path = download(url, "./", kind="zip", replace=True)
```

```
Downloading data from https://mindspore-website.obs.cn-north-4.myhuaweicloud.com/no

file_sizes: 100%|████████████████████████| 10.8M/10.8M [00:01<00:00, 6.91MB/s]
Extracting zip file...
Successfully downloaded / unzipped to ./
```

1.3 任务说明与对应文件：

- `NN-task1-manual-cifar10`：对应任务一，纯手动实现神经网络，包含了手动搭建全过程，以及在 cifar10 数据集上的训练、绘图、测试结果。

- `NN-task2-mindspore-cifar10`：对应任务二，对比 MindSpore，`trainANDtest_mindspore_cifar10`包含了基于 MindSpore 的神经网络（NN 和 CNN）的实现,以及在 cifar10 数据集上的训练、绘图、测试结果。

- （加分项）`NN-Bonus-MNIST`：对应任务三加分项，在其他数据集上对比自己实现的算法和官方代码，利用 Mindspore 提供的接口下载了 MNIST 数据集，并利用手动网络和官方网络进行了训练、测试和对比。

2. 实现步骤与流程

2.1 实验原理与数学推导

（1）手动实现全连接神经网络

单隐层神经网络包含一个输入层、一个隐藏层和一个输出层，其中每个神经元与相邻层的神经元具有全连接的结构特点，而与同层的神经元之间没有连接。在本题的情境中，对应于输入图像的 32*32*3 个像素，输入层的节点个数应当为 32*32*3；而隐藏层的节点个数应当是一个可调整的超参数，会在后续的实验中进行测试与调整，选取最优的个数；对应于图像被划分为的 10 个类别，输出层的节点个数应当为 10。

```
model_manner = MannerNeuralNetwork(32 * 32 * 3, 256, 10)

model_manner.print_model_structure()
```

```
Model Structure:
Input Layer: 3072 neurons
Hidden Layer: 256 neurons
Output Layer: 10 neurons
```

对于另外两类参数：权重和阈值，这是训练过程中模型要学习的参数。将会进行验证实验选取最合适的。

（2）手动实现反向传播算法

正向传播，得到各层输出$\boldsymbol{h}^{(0)},\boldsymbol{h}^{(0)},\boldsymbol{h}^{(0)}$：

$$\boldsymbol{z}^{(0)} = \boldsymbol{x}, \boldsymbol{h}^{(0)} = \boldsymbol{z}^{(0)}$$
$$\boldsymbol{z}^{(1)} = \boldsymbol{W}^{(1)}\boldsymbol{h}^{(0)} + \boldsymbol{b}^{(1)}, \boldsymbol{h}^{(1)} = ReLU(\boldsymbol{z}^{(1)})$$
$$\boldsymbol{z}^{(2)} = \boldsymbol{W}^{(2)}\boldsymbol{h}^{(1)} + \boldsymbol{b}^{(2)}, \boldsymbol{h}^{(2)} = \boldsymbol{softmax}(\boldsymbol{z}^{(2)})$$

反向传播，得到输出层误差$\boldsymbol{e}^{(2)}$：

$$\boldsymbol{e}^{(2)} = \frac{\partial L}{\partial \boldsymbol{z}^{(2)}} = \boldsymbol{h}^{(2)} - \boldsymbol{y}$$

其中，L是损失函数。

计算各层梯度和隐藏层误差$\boldsymbol{e}^{(1)}$

$$\nabla_{\boldsymbol{W}^{(2)}} L = {\boldsymbol{h}^{(2)}}^{T} \cdot \boldsymbol{e}^{(2)}$$

$$\nabla_{\boldsymbol{b}^{(2)}} L = \boldsymbol{e}^{(2)}$$

$$\boldsymbol{e}^{(1)} = \frac{\partial ReLU(\boldsymbol{z}^{(1)})}{\partial \boldsymbol{z}^{(1)}} \odot ({\boldsymbol{e}^{(2)}} \cdot {\boldsymbol{W}^{(2)}}^{T})$$

$$\nabla_{\boldsymbol{W}^{(1)}} L = {\boldsymbol{h}^{(1)}}^{T} \cdot \boldsymbol{e}^{(1)}$$

$$\nabla_{\boldsymbol{b}^{(1)}} L = \boldsymbol{e}^{(1)}$$

对$ReLU$求导得

$$\frac{\partial ReLU(x)}{\partial x} = \begin{cases} 0, x \leq 0 \\ 1, x > 0 \end{cases}$$

更新各层参数：

$$\boldsymbol{W}^{(t)} \leftarrow \boldsymbol{W}^{(t)} - \eta \nabla_{\boldsymbol{W}^{(t)}} L$$
$$\boldsymbol{b}^{(t)} \leftarrow \boldsymbol{b}^{(t)} - \eta \nabla_{\boldsymbol{b}^{(t)}} L$$

（3）CNN 网络原理

卷积神经网络（Convolutional Neural Network, CNN）是一种深度学习模型，广泛应用于图像处理和计算机视觉领域。CNN 通过卷积层、池化层和全连接层的组合，能够自动提取图像的特征并进行分类。

【卷积层】

卷积层是 CNN 的核心部分，通过卷积操作从输入图像中提取特征。卷积操作使用滤波器（或卷积核）在输入图像上滑动，计算局部区域的加权和，生成特征图。

公式表示为：

$$(f * g)(t) = \int_{-\infty}^{+\infty} f(t-\tau)d\tau$$

在离散情况下，二维卷积可表示为：

$$(I * K)(i,j) = \sum_m \sum_n I(i-m, j-n)K(m,n)$$

其中，I 是输入图像，K 是卷积核，i,j 是输出特征图的位置索引。

【激活函数】

为了引入非线性，卷积层的输出通常通过激活函数处理。常用的激活函数包括 ReLU（Rectified Linear Unit）

$$ReLU(x) = \begin{cases} 0, x \leq 0 \\ x, x > 0 \end{cases}$$

【池化层】

池化层（Pooling Layer）通过下采样操作减少特征图的尺寸，保留主要特征并减少计算量。常用的池化操作包括最大池化（Max Pooling）和平均池化（Average Pooling）。

最大池化的公式为：

$$y_{i,j} = \max(x_{m,n})$$

其中，$x_{m,n}$ 是池化窗口内的值。

【全连接层】

全连接层（Fully Connected Layer）连接到最后的卷积层或池化层，用于将特征图展平并输出分类结果。全连接层的计算公式类似于传统的神经网络：

$$y = f(Wx + b)$$

W 是权重矩阵，x 是输入向量，b 是偏置向量，f 是激活函数。

【损失函数与优化】

CNN 的训练过程通过最小化损失函数来调整模型参数。常用的损失函数包括交叉

熵损失（Cross-Entropy Loss）：

$$L = -\sum_I y_i \, log(\hat{y}_i)$$

其中，y_i 是实际标签，$\hat{y}_i$ 是预测概率。优化方法常用梯度下降算法及其变种，如随机梯度下降（SGD）和 Adam 优化算法。

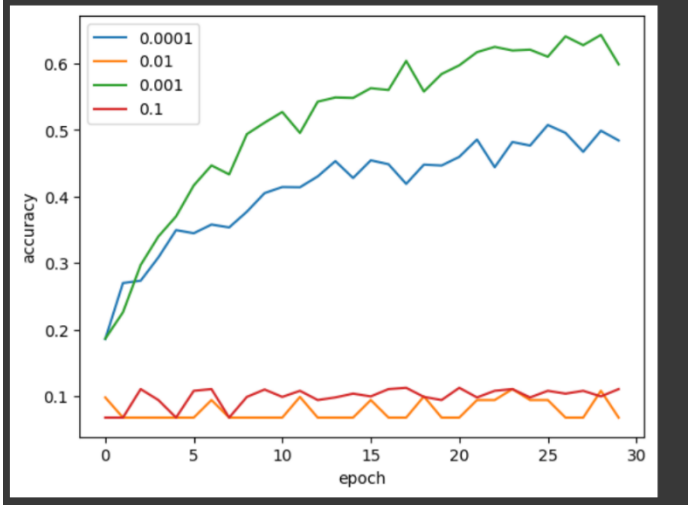
本实验选取的是 Adam 优化算法。

由于时间有限，完全基于 numpy 的 CNN 难以实现，我尝试调用 mindspore 实现了一个 CNN，以更好地契合 cifar10 数据集。

（4）超参数选取实验：学习率 learning-rate

在本次实验中我在 nn+Adam+30 个 epoch 的条件下进行了学习率分别为 0.1、0.01、0.001、0.0001 的四轮训练，绘制了四条 epoch-accuracy 曲线，并将其对应模型应用于测试集进行性能比较，在比较中可以看出学习率为 0.001 为最佳选择。

```
plot_accuracies(history001)
plot_accuracies(history01)
plot_accuracies(history0001)
plot_accuracies(history1)

plt.legend(["0.0001","0.01","0.001","0.1"])
plt.show()
```



过高的学习率（如 0.1 和 0.01）会导致训练过程中的权重更新过大，使得模型无法稳定收敛，表现为准确率低且波动大。

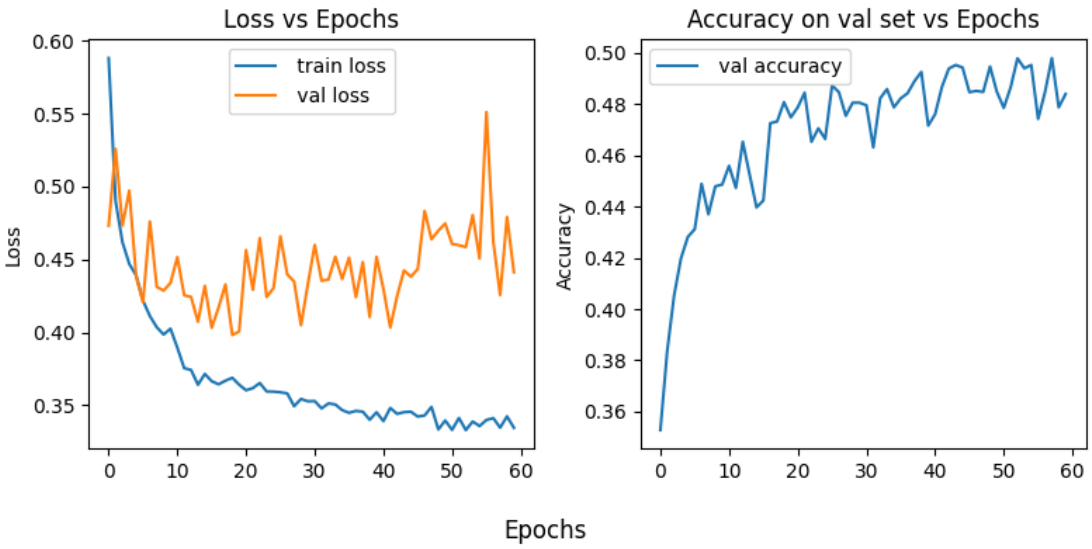
因此选择 lr=0.001 作为本实验的学习率。

（5）超参数选取实验：训练轮数 epoch

Epoch 是指将整个训练数据集在神经网络中前向传播和反向传播的一次完整迭代。 Epoch 的数量直接影响模型是否能够收敛到最优解。较少的 Epoch 可能导致模型尚未完全学 习训练数据中的模式和特征，从而无法达到最佳性能。较多

的 Epoch 可以让模型有更多机会 从训练数据中学习，但过多的 Epoch 可能导致过拟合，即模型过度适应训练数据而在未见过 的数据上表现不佳。

为了更好地看出 trade-off 在曲线上的反映，我调整了更大的训练轮数，绘制了损失-学习周期图，**可以看到验证集的 loss 从 epoch 为 20 左右开始回升，过拟合现象出现。**

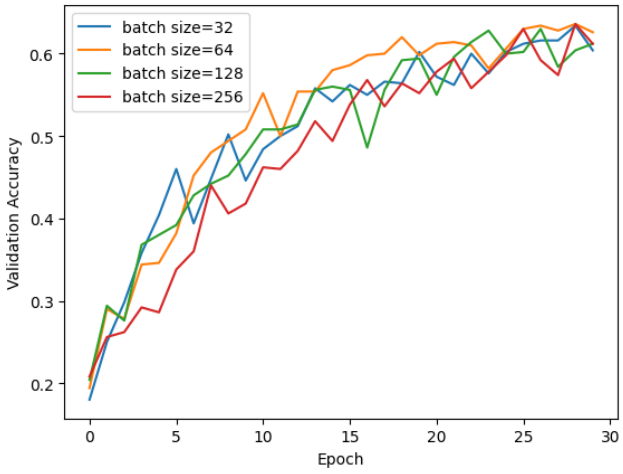


因此本实验我选择的 epoch=20 进行训练。

（6）超参数选取实验：批处理大小 batch

批量大小是指在神经网络的训练过程中，每一次迭代更新权重时所使用的样本数量，其对训练速度、内存需求、泛化能力、噪声影响都会产生影响。

批量大小应当在内存需求允许的范围内尽可能大，本次实验我测试了当 batch_size 分别为 32、64、128、256 时，模型训练的速度和准确率上升情况：



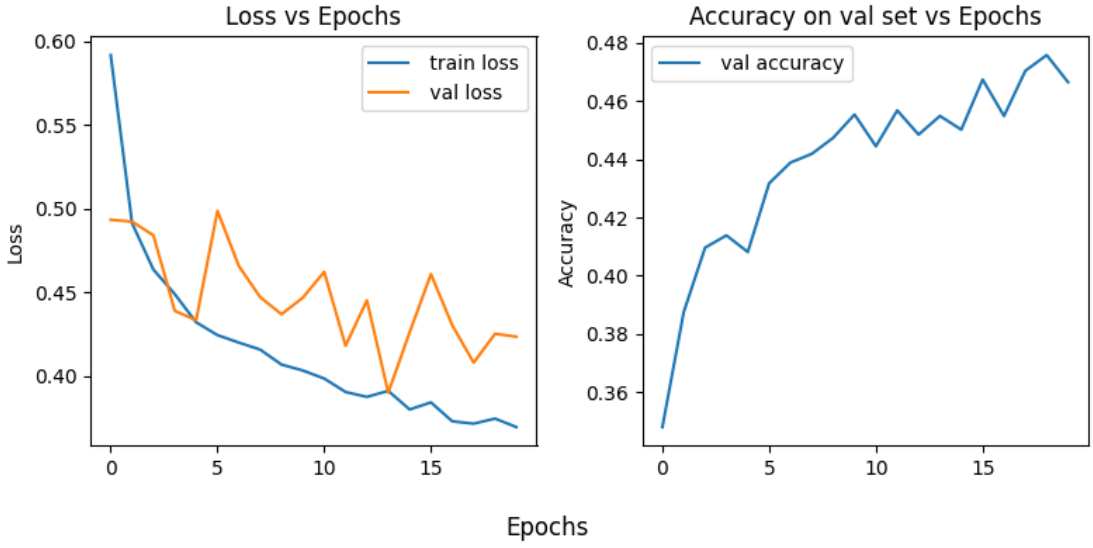
从图中可以看出，batch size 为 64 在整个训练过程中表现最佳，验证准确性上升较快且最终达到了较高的值。较大的 batch size（128 和 256）：尽管最终能达到较好的准确性，但训练初期的上升速度较慢。

最终我选择 batch size 64 作为训练的默认值，因为它在稳定性和学习效率之间取得了较好的平衡。

3. 实验结果与分析

3.1 任务一：自己手动实现全连接神经

我将下载的 batch1-4 作为训练集，batch5 作为验证集进行实验。
根据超参数选取实验的结果，选择 lr=0.001, epoch=20, batch=64 进行训练，绘制损失和预测准确率关于轮数的变化曲线：




```
Epoch[1]: train_loss: 0.5916, val_loss: 0.4933, val_acc: 0.3480
Epoch[2]: train_loss: 0.4913, val_loss: 0.4922, val_acc: 0.3875
Epoch[3]: train_loss: 0.4636, val_loss: 0.4841, val_acc: 0.4097
Epoch[4]: train_loss: 0.4490, val_loss: 0.4390, val_acc: 0.4139
Epoch[5]: train_loss: 0.4322, val_loss: 0.4334, val_acc: 0.4082
Epoch[6]: train_loss: 0.4246, val_loss: 0.4986, val_acc: 0.4318
Epoch[7]: train_loss: 0.4201, val_loss: 0.4658, val_acc: 0.4389
Epoch[8]: train_loss: 0.4159, val_loss: 0.4472, val_acc: 0.4419
Epoch[9]: train_loss: 0.4070, val_loss: 0.4370, val_acc: 0.4474
Epoch[10]: train_loss: 0.4034, val_loss: 0.4469, val_acc: 0.4554
Epoch[11]: train_loss: 0.3986, val_loss: 0.4623, val_acc: 0.4445
Epoch[12]: train_loss: 0.3906, val_loss: 0.4182, val_acc: 0.4568
Epoch[13]: train_loss: 0.3877, val_loss: 0.4452, val_acc: 0.4485
Epoch[14]: train_loss: 0.3913, val_loss: 0.3902, val_acc: 0.4549
Epoch[15]: train_loss: 0.3802, val_loss: 0.4262, val_acc: 0.4502
Epoch[16]: train_loss: 0.3844, val_loss: 0.4609, val_acc: 0.4674
Epoch[17]: train_loss: 0.3731, val_loss: 0.4303, val_acc: 0.4549
Epoch[18]: train_loss: 0.3717, val_loss: 0.4081, val_acc: 0.4705
Epoch[19]: train_loss: 0.3748, val_loss: 0.4253, val_acc: 0.4759
Epoch[20]: train_loss: 0.3697, val_loss: 0.4236, val_acc: 0.4665
```

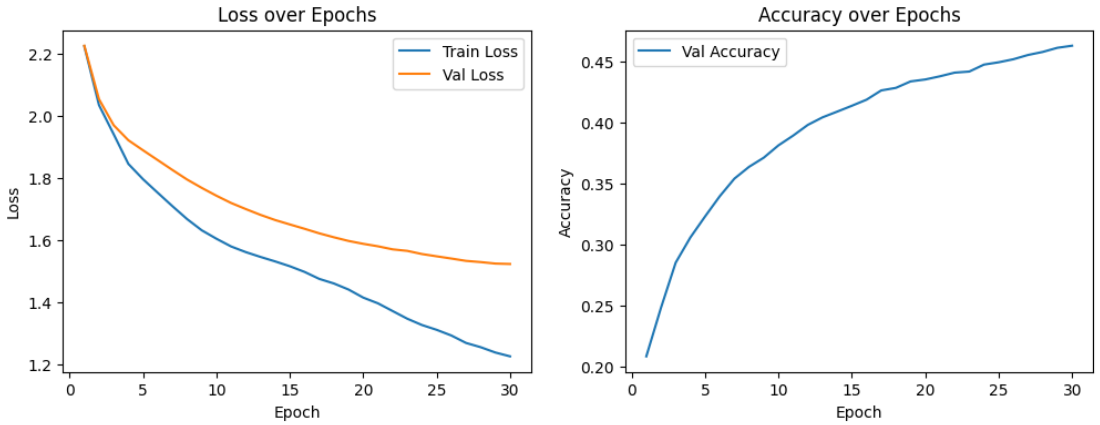
```
[10]
...   Test Accuracy: 0.4674
```

最终，手动实现的全连接网络在验证集上达到了 46.6%的准确率，在测试集上达到了 46.7%的准确率。

3.2 任务二：基于 mindspore 实现全连接神经 NN 和卷积神经网络 CNN

Mindspore 有较为成熟的神经网络实现库，包括 nn\Tensor 等可以直接使用的成熟算法。

与手动实现不同的是，mindspore 实现的网络在训练到 30 轮时仍然有较为明显的 accuracy 的上升，于是我将 epoch 提升至了 30 轮。在 mindspore 实现的全连接神经网络下进行训练，绘制 loss 曲线和验证集 accuracy 曲线：




```
Epoch[30]: train_loss: 1.2272, val_loss: 1.5240, val_acc: 0.4632
Test accuracy: 0.4684
```

最终在验证集上实现了 46.3%的正确率，在测试集上则是实现了 46.8%的准确率，证明拟合效果较好。与手动实现的全连接网络准确率接近。

【与手动实现的对比】

虽然最终准确率接近，但可以从曲线上明显看出，**手动实现的网络 Loss 下降得很不稳定**，有时还会有回升的趋势。对此做出分析：

优化器的实现差异：

手动实现的 Adam 优化器可能在某些细节上与 MindSpore 实现存在差异，例如参数更新方式、计算精度等，导致训练过程不够平稳。

超参数设置：

MindSpore 的默认超参数可能经过更好的调优，适用于更多场景，训练过程更加稳定。而手动实现的网络对于 0.001 的学习率而言还是略大了一些。

数值稳定性：

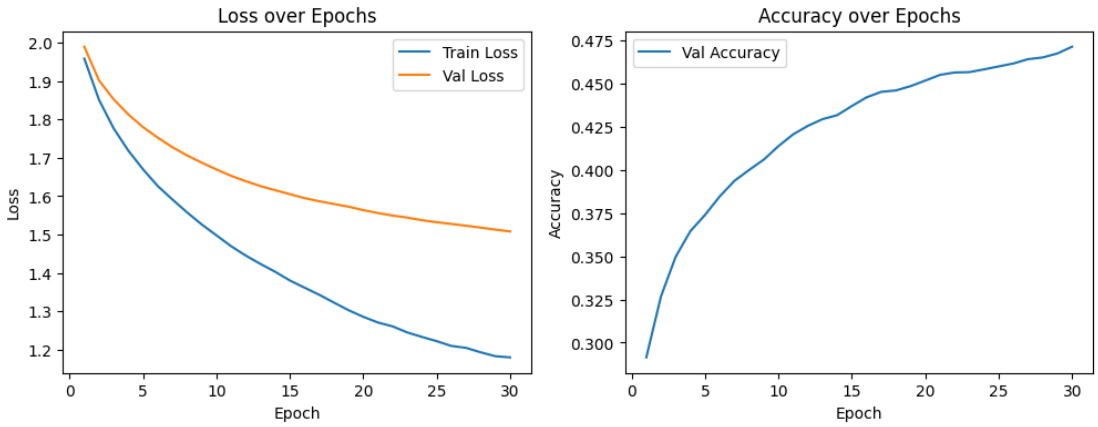
MindSpore 在处理梯度计算、参数更新等数值运算时，可能采用了更高精度的数值方法，避免了数值不稳定导致的训练波动。

模型实现细节：

手动实现的模型在前向传播、反向传播等计算过程中，可能存在一些小的实现误差，积累后影响模型性能。MindSpore 经过严格测试和优化，其模型实现更加可靠。

【CNN 尝试实现】

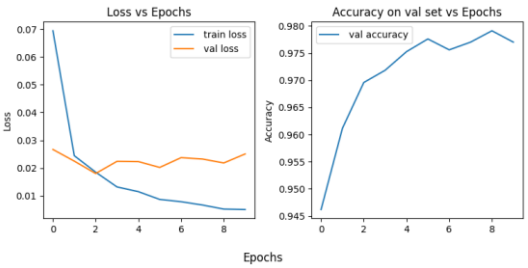
对于 cifar10 数据集而言，其作为一个 3 通道的多分类图象识别任务，而且类别与类别之间的差异较大，简单的全连接网络并不是很适用。于是我尝试基于 mindspore 实现了一个卷积神经网络进行训练：



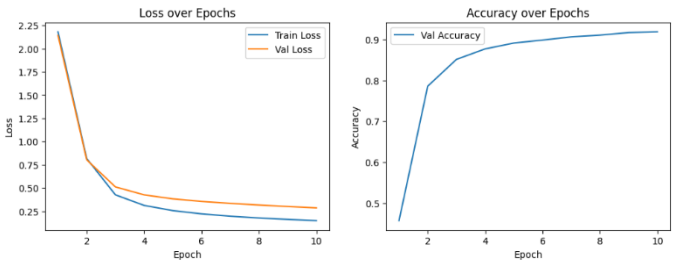
最终获得了 50%的准确率。并没有非常显著的提升，后期可以考虑增大数据集的样本数量再次实验。

3.3 任务三（加分项）：在 MNIST 数据集上进行实验

分别用手动搭建的全连接网络（左）和 mindspore 实现（右）的网络在 mnist 数据集上进行训练，可以看出手动的网络仍然没有 Mindspore 下的稳定：




```
Test accuracy: 0.9195
```

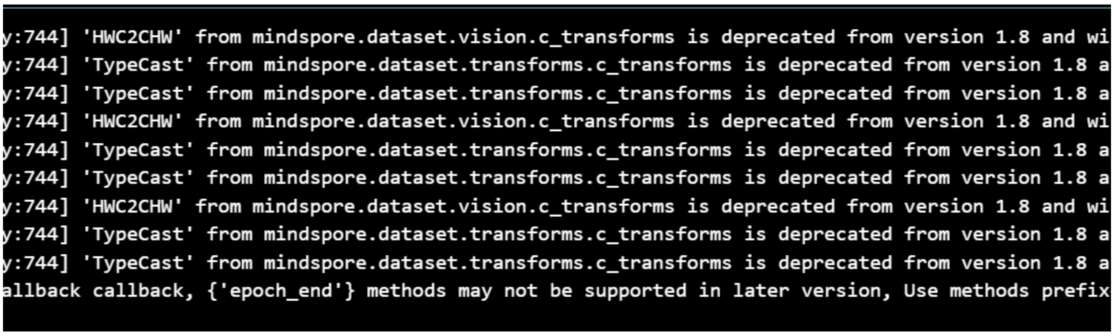



```
Test accuracy: 0.9772
```

最终自己的网络达到了 91.95%的准确率，mindspore 为 97.72%。

4 MindSpore 学习使用心得体会

与我熟悉的 Torch 库相比，MindSpore 的操作方式确实有所不同，这使得我需要花费更多的时间去适应和理解。最初，我经常遇到各种奇怪的错误，这让我感到有些崩溃。



版本问题也非常棘手。MindSpore 的不同版本之间存在一些冲突，这导致了一些问题的出现，还会报一些我从未谋面的错，让我不知道从何解决。为了解决这些问题，我不得不仔细查阅文档和社区讨论，以确保我所使用的版本与我正在进行的实验兼容。好在最后磕磕巴巴跑出了合理的结果。

尽管遇到了一些困难， MindSpore 的优点也是显而易见，它自动并行计算功能对于加速训练过程非常有帮助。此外，MindSpore 的图模式训练使得实现更高效的模型变得更加容易。

从第一个课题，到神经网络的课题，我逐渐适应了 MindSpore，并且发现了它的一些优点。相信多用就会更加熟练。

心得体会

此段我决定抛开一个报告撰写者的严谨口吻，走心地谈谈本次实验的真实心得。

配置 mindspore 实际上比实验内容本身和课内知识的联系更让我受益匪浅。我曾听过一个学长的调侃:“一个实验你把环境配置好，基本已经实现一半了。”此言不假。尤其是对于 python 而言，一个没有很科学地入门的小白很可能会造成自己电脑的 python 一团糟，各种库的版本互相打架。好在我已经略有经验，在一开始就选择在虚拟环境里配 mindspore，但惊人地发现它只支持 Python3.7-3.9，而我的 python 是 3.12 的。于是我有非常小心翼翼的重新装了一个 3.9（因为任何命名问题都会导致我的两个版本的 Python 在我的电脑里打架）（虽然调 pycharm 的解释器我又调了半天）。事实上，为了高效地完成所有的实验，我在 pycharm（更好用的操作界面）、jupyter（更好的交互性，保存输出结果）、colab（更高的计算性能，主要为了训练神经网络用的）上都跑过。每一个平台装配 mindspore、装配数据集、文件路径等前期工作就会耗时良久

——而比起题目背后的数学原理，这实操过程会遇到的一切难题，也许才是实验对于一个计算机类的学生最大的意义所在。

实验从发布到 ddl 总共一个月，从第一节实验课我就开始着手配置 mindspore，自此未间断过每日对实验进度的推进，一直到 6 月 5 日撰写报告时复盘代码文件，我都仍然在改进。我想说，想完成如上这些工作量的标准任务，这是最基本的耗时。我的时间非常紧，和同级的其它同学相比，我作为一个转系生还需要额外修 6 门专业课，但我还是会在赶 ddl 的时候反复问自己，代码是否写得够高效？够简洁？够科学？谋篇布局是不是合理？可读性强不强？我又也会反复挣扎，唉，其实可以跑出来已经很不错了——
好在最后我选择在有限的时间，尽我最大的努力去完成了一个‘tradeoff’。

完结撒花。