一、任务介绍

(一) KNN算法概述

KNN算法，也就是K-Nearest Neighbor算法，是机器学习领域中一个非常经典且基础的算法。它最初由Cover和Hart在1968年提出，至今已经有半个多世纪的历史了。尽管它的出现时间较早，但凭借其简单直观的原理，依然在众多机器学习算法中占据着重要的地位。

KNN算法的核心思想非常容易理解：如果一个样本在特征空间中的K个最相似（即特征空间中最邻近）的样本中的大多数属于某一个类别，那么这个样本也属于这个类别。换句话说，它主要是通过查看周围邻居的类别来判断当前样本的类别。这种基于“近朱者赤，近墨者黑”的思想进行分类的方式，使得KNN算法在处理一些简单的分类问题时表现得非常出色。

(二) KNN算法的流程

KNN算法的实现过程可以概括为以下四个步骤：

- 数据准备**：首先，我们需要对数据进行预处理。这一步骤是至关重要的，因为数据的质量直接影响到算法的性能。在我们的项目中，我们使用的是Mnist数据集，这是一个标准的手写数字数据集，包含了大量已经标注好的手写数字图像。我们对这些图像进行了归一化处理，以便更好地进行后续的计算。
- 距离计算**：接下来，我们需要计算测试样本点到其他每个样本点的距离。在我们的项目中，我们选择的是欧氏距离作为距离度量。欧氏距离是最常用的距离度量方式之一，它能够很好地衡量两个点在多维空间中的实际距离。通过计算测试样本与训练样本之间的欧氏距离，我们可以得到一个距离列表。
- 选择最近邻**：有了距离列表之后，我们需要对每个距离进行排序，然后选择出距离最小的K个点。这K个点就是测试样本的最近邻。这个步骤是KNN算法的关键所在，因为最近邻的类别将直接影响到测试样本的分类结果。
- 投票决策**：最后，我们需要对这K个最近邻所属的类别进行比较。根据少数服从多数的原则，将测试样本点归入在K个最近邻中占比最高的那一类。也就是说，如果在这K个最近邻中，有5个属于类别A，3个属于类别B，那么测试样本就会被归为类别A。

(三) KNN算法的优缺点

KNN算法的优点是显而易见的。首先，它的思路非常简单，易于理解和实现。即使是初学者也能够快速地掌握其基本原理，并且实现一个简单的KNN分类器。其次，它无需估计参数，这使得它在处理一些简单的分类问题时非常高效。因为不需要进行复杂的参数估计，所以算法的运行速度相对较快。

然而，KNN算法也存在一些缺点。当样本不平衡时，例如一个类的样本容量很大，而其他类样本容量很小时，有可能导致当输入一个新样本时，该样本的K个邻居中大容量类的样本占多数。这就会影响到分类的准确性。此外，KNN算法的另一个不足之处是计算量较大。因为对每一个待分类的文本都要计算它到全体已知样本的距离，才能求得它的K个最近邻点，所以在处理大规模数据集时，算法的运行时间会变得非常长。

(四) Mnist数据集介绍

Mnist数据集是机器学习领域的经典手写数字识别基准库。它包含了0到9共10类数字，数据集由6万张28×28像素训练图像和1万张测试图像组成。每张图像都是灰度图，像素值在0到255之间，经过归一化处理后，数据更加适用于机器学习算法的训练和测试。Mnist数据集的一个显著特点是它的数据经过了尺寸标准化和居中处理，字符线条清晰且背景纯净。这使得它成为验证分类算法性能的首选测试平台，广泛应用于教学和算法原型开发。

二、任务实现

(一) 导入库和模块

在实现KNN算法的过程中，我们使用了一些常用的Python库和模块。其中，numpy用于数值计算，它提供了强大的数组操作功能，能够帮助我们高效地处理数据。matplotlib是一个绘图库，虽然在我们的代码中没有直接使用它的绘图功能，但它在数据可视化方面有着不可替代的作用。torch是PyTorch深度学习框架，它为我们提供了强大的深度学习工具，使得我们能够更加方便地处理数据和实现算法。DataLoader用于批量加载数据，它能够将数据集封装为可迭代的对象，方便我们进行批量处理。datasets.MNIST是PyTorch内置的MNIST数据集接口，通过它我们可以轻松地下载并加载MNIST数据集。Utils是我们自定义的工具模块，它包含了KNN实现和预处理函数，这些函数在我们的项目中发挥了重要的作用。

(二) 加载MNIST数据集

我们首先下载并加载了MNIST的训练集和测试集。然后，我们将数据集封装为可迭代的DataLoader对象，用于批量加载。为了保证数据的随机性，我们还打乱了数据顺序。这一步骤是数据预处理的重要环节，它能够确保我们在训练和测试过程中使用到的数据是随机的，从而提高算法的泛化能力。

(三) 数据预处理

在数据预处理阶段，我们对x_train和x_test进行了处理。x_train和x_test是形状为（样本数，28，28）的NumPy数组。我们对这些数据进行了中心化处理，即通过减去均值图像对数据进行归一化。这种处理方式能够消除数据中的偏差，使得数据更加符合算法的要求，从而提高算法的性能。

(四) KNN分类与评估

在KNN分类与评估阶段，我们首先初始化了分类器。然后，我们存储了训练数据和标签，这是通过fit方法实现的。接下来，我们对每个测试样本，计算与所有训练样本的欧氏距离，找到最近的k个邻居，投票预测类别。这一步骤是KNN算法的核心，它直接决定了分类的准确性。

我们还计算了训练集的平均图像。这个功能是将输入的三维图像数据展平为二维数组，并计算每个像素位置（列）的平均值。这个平均图像在数据预处理阶段起到了重要的作用，它能够帮助我们更好地对数据进行归一化处理。

我们还对输入数据（测试集）进行了中心化处理，即减去训练集的平均图像。这个步骤能够消除数据中的偏差，使得数据更加符合算法的要求。

最后，我们实现了预测算法。这个算法的输入包括近邻数k、距离类型dis以及测试数据X_test，输出则是预测标签数组。通过这个算法，我们能够对测试数据进行分类，并且评估算法的性能。

三、运行展示

在运行展示阶段，我们对KNN算法在Mnist手写体识别中的性能进行了评估。当测试集为10000张图时，我们分别测试了K=1、K=3和K=5的情况。结果显示，当K=1时，识别率为0.9693；当K=3时，识别率为0.9714；当K=5时，识别率为0.9691。从这些数据可以看出，KNN算法在Mnist手写体识别中表现出了非常出色的性能，尤其是在K=3时，识别率达到了最高的0.9714。

四、总结

通过本次项目，我们了解了KNN算法的原理和实现过程，并且成功地将其应用于Mnist手写体识别中。我们不仅实现了算法的基本功能，还对算法的性能进行了评估。在项目过程中，我们遇到了许多困难和挑战，但通过小组成员的共同努力，我们一一克服了这些困难。这个项目不仅提高了我们的编程能力和算法设计能力，还培养了我们的团队协作能力和问题解决能力。

在未来的研究中，我们希望能够进一步优化KNN算法的性能，提高算法的运行效率。我们还可以尝试将KNN算法与其他算法相结合，以实现更好的分类效果。此外，我们还可以探索KNN算法在其他领域的应用，例如图像识别、语音识别等，为人工智能的发展做出更大的贡献。

谢谢大家！