KNN 算法在 MNIST 数据集上的 分类效果实验

1. MNIST数据集介绍

MNIST数据集是NIST数据集的扩展版，是一个为测试机器学习算法设计的简易手写数字数据集。MNIST数据集含有70000张28x28像素大小的黑白图片（其中60000张为训练集，10000张为测试集）。将数据集这样分离，可以模型使用该数据集可以充分训练成合格的二分类模型。

2. 算法：K-nearest neighbor (KNN)

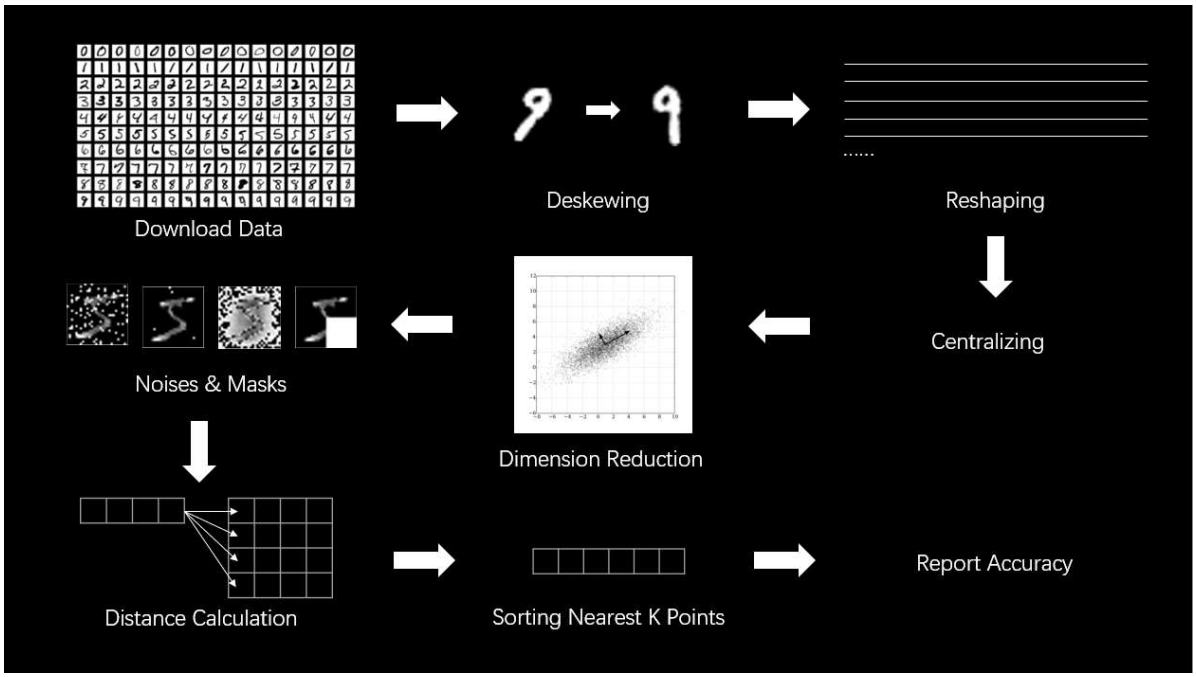
它是有监督学习分类算法的一种。所谓 K 近邻，就是 K 个最近的邻居。比如对一个样本数据进行分类，我们可以用与它最邻近的 K 个样本来表示它，这与俗语"近朱者赤，近墨者黑"是一个道理。

KNN是通过测量不同特征值之间的距离进行分类。它的思路是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，则该样本也属于这个类别，其中K通常是不大于20的整数。KNN算法中，所选择的邻居都是已经正确分类的对象。该方法在定类决策上只依据最邻近的一个或者几个样本的类别来决定待分样本所属的类别。

最简单最初级的分类器是将全部的训练数据所对应的类别都记录下来，当测试对象的属性和某个训练对象的属性完全匹配时，便可以对其进行分类。但由于不是所有对象都可以找到与之匹配的类，且存在一个测试对象同时与多个训练对象匹配的情况，KNN算法便应运而生。KNN是通过测量不同特征值之间的距离进行分类。它的思路是：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，则该样本也属于这个类别，其中K通常是不大于20的整数。KNN算法中，所选择的邻居都是已经正确分类的对象。该方法在定类决策上只依据最邻近的一个或者几个样本的类别来决定待分样本所属的类别。

同时，KNN通过依据k个对象中占优的类别进行决策，而不是单一的对象类别决策。这两点就是KNN算法的优势。

我的算法过程：



（1）使用pytorch库加载数据。

（2）图像抗扭曲处理。

（3）一维化处理，将每张图像展开为一维数组，方便运算.

（4）中心化处理，将图像的每一个像素减去所有像素的平均值，可以减小运算的数量级。

（5）数据降维，主要分别使用主成分分析（PCA）与线性判别分析（LDA）。

（6）模型晃动，主要测试添加噪声、矩形蒙版，减少训练数据等。

（7）距离计算，对于10000个测试点中的每一个测试点，分别计算其到60000个训练点的距离。

（8）排序，对60000个距离排序，选择最近的k个点。

（9）统计k个点的标签个数，选择数量最多的标签作为未知点的标签。

（10）统计正确率，输出报告。

3. 最佳结果

3.1. 最佳结果：

Accuracy = 0.976400, duration = 237.759806 seconds

控制参数为：K = 3，L = L2, 使用PCA降维，并只对标签为“0”的测试集图片进行偏斜校正。下面会详细讲述为什么使用这样的参数。

3.2. 次好结果：

Accuracy: 0.976200, duration = 213.119608 seconds

控制条件为：K = 3, L = L2, 使用PCA降维，不进行偏斜校正。

可能由于计算性能有波动，有时使用这组参数，准确率也可以达到97.64%。但是当达到上述最佳结果时，使用本组参数仅能达到97.62%，仍说明使用上组参数时准确率应略好于本组参数。

4. 参数选择

KNN的结果很大程度上依赖K的选择和距离的选择。

控制条件	测试数据	正确数据	正确率	运行时间
L=L2, k=1	10000	9693	0.9693	2161.0477 seconds
L=L2, k=3	10000	9714	0.9714	2068.4964 seconds
L=L2, k=4	10000	9712	0.9712	
L=L2, k=5	10000	9691	0.9691	

3.3. 距离选择

控制条件	测试数据	正确数据	正确率	运行时间
L=L1, k=3	10000	1710	0.1710	1692.8180 seconds
L=L2, k=3	10000	9714	0.9714	2068.4964 seconds
L=L3, k=3, with PCA	10000	9694	0.9694	947.308111 seconds

对比得出曼哈顿距离在本数据集上基本不起作用，欧拉距离效果更好；使用欧拉距离时，选k=3的效果最好，此时正确率为97.14%。

5. 主成分分析Principle Component Analysis (PCA)

主成分分析（PCA）是一种对高维数据进行降维处理的算法，其特点是最大程度减少数据分布信息的损失，在降维过程中维持数据点之间的距离差值。PCA由于不处理测试集的标签信息，一般更适用于无监督学习。测试PCA对MNIST数据集的降维效果：

控制条件	测试数据	正确数据	正确率	运行时间
component = 50	10000	9760	0.976000	165.128043 seconds
components = 55	10000	9761	0.976100	175.654464 seconds
component = 60	10000	9762	0.976200	254.208601 seconds
component = 70	10000	9751	0.975100	241.581530 seconds
component = 100	10000	9737	0.973700	330.123000 seconds
component = 300	10000	9711	0.971100	849.023178 seconds

使用PCA降维后准确率明显提升。

6. 线性差值分析Linear Discriminant Analysis (LDA)

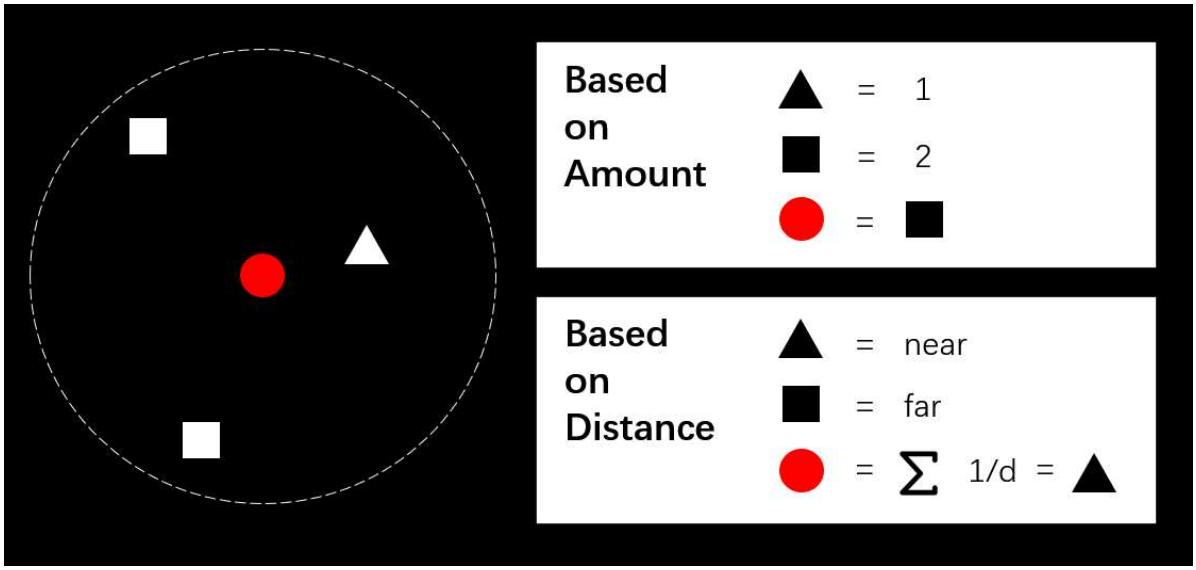
线性差值分析是另一种高维数据的降维算法，其特点是按照数据的类别标签，使同类数据点尽量靠近，不同类数据点尽量远离，一般适用于有监督学习。测试LDA在MNIST数据集上的准确率：

控制条件	测试数据	正确数据	正确率	运行时间
component = 2	10000	5040	0.5040	83.626468 seconds
component = 8	10000	9048	0.9048	109.745287 seconds
component = 9	10000	9115	0.9115	108.179354 seconds

虽然LDA更适用于监督学习，但由于MNIST任务中，LDA的特征数量不能大于class数量-1即9为可设定维度的最大值。测试发现准确率随维度数增加而增加，调整至最大值后效果仍不佳，可以推断LDA不太适用于该任务。

7. 添加超参数distance

测试在K个点排序时不只是考虑K个点中类别的个数，也K个点的距离。一般将距离的倒数作为权重，加入运算将原先只考虑最近标签的做法，变成用最近k个点的距离倒数之和排序。



举例说明，如图是距一个点（圆）最近的k = 3个点，设矩形点A距离为4，矩形点B距离为3，三角形点距离为1。如果根据经典的投票原则，矩形点个数为2，三角形点个数为1，则未知点应当归为矩形类。但我们考虑其实三角形点距离未知点更近，如果加权计算，计算权重后的预测值为：

```
三角形：1

矩形：1/3 + 1/4 = 7/12
```

预测未知点为矩形类，与经典方法的预测值不同。

代码实现

源代码（个数排序法）：

```
for j in range(num_test):
    distances = np.sqrt(np.sum(((self.Xtr - np.tile(X_test[j],
(self.Xtr.shape[0], 1)))) ** 2, axis=1))
    nearest_k = np.argsort(distances)
    topK = nearest_k[:k]
    class_count = {}
    for i in topK:
        class_count[self.Ytr[i]] = class_count.get(self.Ytr[i], 0) + 1
    sorted_class_count = sorted(class_count.items(), key=lambda elem: elem[1],
reverse=True)
    label_list.append(sorted_class_count[0][0])
```

修改后（距离排序法）：

```
for j in range(num_test):
    distances = np.sqrt(np.sum(((self.Xtr - np.tile(X_test[j],
(self.Xtr.shape[0], 1)))) ** 2, axis=1))
    nearest_k = np.argsort(distances)
    topK = nearest_k[:k]
    distance_count = {}
    for i in topK:
        distance_count[self.Ytr[i]] = distance_count.get(self.Ytr[i], 0) +
(1/distances[self.Ytr[i]])
    sorted_distance_count = sorted(distance_count.items(), key=lambda elem:
elem[1], reverse=True)
    label_list.append(sorted_distance_count[0][0])
```

实验结果

控制条件	测试数据	正确数据	正确率	运行时间
L=欧拉距离, k=3	10000	9680	0.9680	2063.929378 seconds

准确率有所下降，说明考虑距离的排序方法不适用于MNIST数据集。

8. Deskewing 偏斜校正

偏斜校正指通过寻找图像中内容的最小外接矩阵，并将该外接矩阵内的内容校正实现校正图像效果。其方法分为两个步骤：

1. 找到图像内容的重心
2. 计算像素重心的协方差矩阵，从而得出图像的偏斜角度。

$$\begin{bmatrix} 1 & 0 \\ \alpha & 1 \end{bmatrix}$$

其中

$$\alpha = \frac{Cov(X,Y)}{Var(X)}$$

本实验调用opencv库中的偏斜校正函数，测试将训练集图像校正后是否能提高训练准确度。随机画出数张校正前和校正后图像，上侧为校正前图像，下图为校正后图像。可以看出经过矫正，数字方向更加垂直：

测试结果为：

控制条件	测试数据	正确数据	正确率	运行时间
PCA components = 100, k = 3	10000	8741	0.8741	316.894558 seconds
PCA components = 60, k = 3	10000	8753	0.8753	211.439254 seconds

偏斜校正并没有使准确率提升很高，通过输出测试集中的10张图片检查，猜测是因为测试集中的图像本身就带有与训练集相似的倾斜角度（尤其是数字1，均有比较统一的向左倾斜），导致一部分数字的识别准确率反而因偏斜校正而下降。

为验证以上这点，分别打印使用偏斜校正后分类错误的数字类别。（校正前参数PCA component = 55， k = 3，校正后参数PCA component = 60， k = 3）

校正前类别	校正前错误数	校正后类别	校正后错误数
0	20	0	14
1	26	1	43
2	14	2	308
3	28	3	78
4	21	4	98
5	25	5	86
6	19	6	34
7	32	7	84
8	21	8	216
9	33	9	66

对比得知，校正后只有数字“0”的错误数减少，于是考虑下次测试时只将训练集中类别为“0”的图片校正。结果为：

控制条件	测试数据	正确数据	正确率	运行时间
K = 3，L = L2, with PCA component = 60	10000	9764	0.9764	237.759806 seconds

为本次实验最好结果。

9. 减少测试点数量

减少测试点数量，测试KNN算法对测试集规模的要求。

控制条件	测试数据	正确数据	正确率	运行时间
train_set = 60000, without PCA	10000	9714	0.9714	2068.4964 seconds
train_set = 30000, without PCA	10000	9262	0.926200	2064.643301 seconds
train_set = 10000, without PCA	10000	9483	0.948300	366.647280 seconds
train_set = 5000, without PCA	10000	9362	0.936200	183.190923 seconds
train_set = 1000, without PCA	10000	8715	0.871500	39.543307 seconds

可以看到即使测试集数量降至1000，准确率仍然较高，猜测因为计算时是每个点的距离分别计算，1000已经远大于1，因此还能得到较好的结果。

有PCA时效果更好，猜测PCA提取特征时使用的是全体训练集的特征：

控制条件	测试数据	正确数据	正确率	运行时间
train_set = 60000, with PCA	10000	9764	0.976400	254.208601 seconds
train_set = 10000, with PCA	10000	9511	0.951100	47.993513 seconds
train_set = 5000, with PCA	10000	9397	0.939700	18.796648 seconds
train_set = 3000, with PCA	10000	9269	0.926900	8.762770 seconds
train_set = 1000, with PCA	10000	8805	0.880500	2.968542 seconds

10. 加噪声测试正确率下降

10.1. 随机噪声

随机噪声的算法是生成随机数，在随机数的位置上将像素值设置为255，即白色点。效果图为设置生成100个随机数时的图像。

控制条件	测试数据	正确数据	正确率	运行时间
L=欧拉距离, k=3, noise_sum = 10	10000	9690	0.969000	2063.8370 seconds
L=欧拉距离, k=3, noise_sum = 100	10000	9262	0.926200	2064.643301 seconds
L=欧拉距离, k=3, noise_sum = 300	10000	6950	0.6950	2063.180387 seconds
L=欧拉距离, k=3, noise_sum = 364 (half of the pixels)	10000	5691	0.5691	2064.940598 seconds

10.2. 椒盐噪声

椒盐噪声的算法是对于每个像素点，产生一个0至1之间的随机数，并有一个0至1之间的阈值，如果随机数大于阈值，则根据设定好的概率将像素值设置为0或255。图为阈值0.99，椒盐噪声比为1：1时生成的图像。可以看到大部分椒噪声与黑色背景重合，而盐噪声比较明显。

控制条件	测试数据	正确数据	正确率	运行时间
L=欧拉距离, k=3, threshold = 0.99	10000	9690	0.969000	2065.1634 seconds
L=欧拉距离, k=3, threshold = 0.5	10000	4882	0.488200	2066.6973 seconds

10.3. 高斯噪声

高斯噪声的算法是先创建一个像素值符合高斯分布的蒙版，然后将蒙版与原图像叠加。下图为添加高斯噪声的图像示意图。

控制条件	测试数据	正确数据	正确率	运行时间
L=欧拉距离, k=3	10000	1372	0.1372	2083.26 seconds

10.4. 矩形蒙版

矩形蒙版的做法是有规律地将一个矩形内的像素点全部变为255或0。

控制条件	测试数据	正确数据	正确率	运行时间
size = 5, top = 0, left = 0, with PCA	10000	9714	0.971400	2064.577296 seconds
size = 14, top = 14, left = 14, with PCA	10000	9635	0.963500	2064.024163 seconds
size = 14, top = 0, left = 0, with PCA	10000	9430	0.943000	2064.868109 seconds

控制条件	测试数据	正确数据	正确率	运行时间
size = 14, top = 14, left = 14, with PCA	10000	9679	0.967900	210.297433 seconds
size = 14, top = 0, left = 0, with PCA	10000	9633	0.963300	212.390084 seconds

可能由于图像背景为黑色，加黑色矩形蒙版时准确率略高于白色蒙版。推测经过PCA降维，矩形蒙版的特征被忽略，导致即使蒙版达到图像四分之一大小时，准确率仍比较高。矩形蒙版加在右下时准确率稍高于加在左上时。

11. 讨论

11.1. KNN模型特点

上述实验中可以发现KNN是一种训练准确度不高，训练时间较短，对训练集规模要求不高，不能并行运算的简单模型。KNN模型对噪声的鲁棒性都不高，对有规律的矩形蒙版鲁棒性较高（有PCA时）。

11.2. 分类错误原因讨论

打印分类错误测试点的预测类别与原类别得知，“9“与”4“，”1“与”7“和”0“与”6“是最易混淆的类别。

11.3. 实验反思

本次实验中的L3距离与图像偏斜校正尝试都未达到相关文献中所得到的准确率效果，推测自己在图像处理方式上与文献所使用方法可能略有不同。