北京工業大學

BEIJINGUNIVERSITYOFTECHNOLOGY

KNN 实现手写数字识别

姓名：陈立标

学号：20090121

班级：200702

完成时间：2022 年 5 月 23 日

1. KNN 算法

1.1 算法介绍

k 近邻法（k-nearest neighbor,k-NN）是一种基本分类与回归方法。k 近邻法的输入为实例的特征对象，对应于特征空间的点，输出为实例的类别。分类的策略为对于新的输入，根据其 k 个最近的已知样本类型，通过多数表决的方式进行预测，最终结果为 k 个近邻中出现次数最多的类别

对于如贝叶斯的大部分机器学习模型，均存在随着训练次数而不断调整变化的参数，而 k 近邻法不具有显式的学习过程，仅利用训练数据集对特征向量空间进行划分，通过该确定空间对测试集样本进行预测，因此具有简单、直观、解释性强的特点。

1.2 数学模型

输入：训练数据集

$$T=\{(x_1,y_1),(x_2,y_2),...(x_n,y_n)\}$$

其中 $x_i \in \chi \subseteq R^n$ 为实例的特征向量，$yi \in \mathrm{Y}=\{c_1,c_2,...c_k\}$ 为实例的类别，i=1，2，3...N;实例特征向量 x;

输出：实例 x 所属的类 y。

（1）根据给定的距离度量，在训练集 T 中找出与 x 最邻近的 k 个点，涵盖这 k 个点的邻域记作 $N_k(x)$;

（2）在 $N_k(x)$中根据多数表决决定 x 的类别 y：

$$y=\arg\max_{c_j}\sum_{x_i\in N_k(x)} I(y_i=c_j), i=1,2,...,N; j=1,2...,K$$

其中，I 为指示函数，即当 $y_i=c_j$ 时 I 为 1，否则 I 为 0。

2. 数据集介绍

本次使用的数据集为机器学习中十分常见的 MNIST 手写数据集，示例见图 1。为方便代码调用，本次实验使用 pytorch 自下载的 MNIST 数据集。数据集中共有 60000 张图片，每张均为 28*28 像素、一通道。图片内容以 28*28 的二维灰度矩阵所决定，对于每一个像素点，其灰度取值均介于 0-255 之间，其中 0 为纯黑，255 为纯白。因此，对于图片的处理均可转化为对二维灰度矩阵的处理。

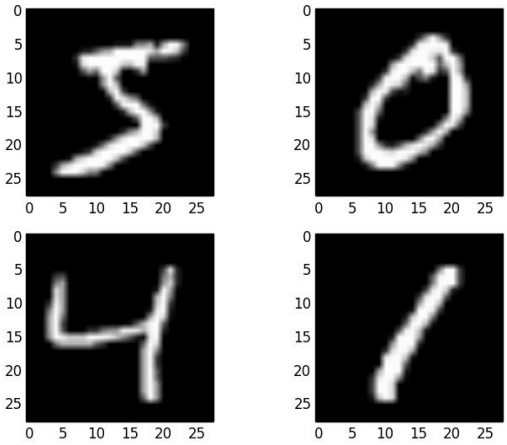

图 1.MNIST 数据集示例

3. KNN 实现手写数字识别

3.1 大体思路

对于给定的数据集，可以将其视作在特征空间中的若干点，空间维度即为特征数量 784。当输入一个待预测样本时，首先逐个计算其与所有数据集样本的欧氏距离，在其中选择 k 个距离最小的样本，以对应类别出现次数最多的类别作为预测值。

3.2 具体实现

数据集处理

数据集内样本本身为 28*28 的二维矩阵，为方便后续欧氏距离计算，先将样本展开为长度为 784 的一维行向量，再将数据集样本总数作为行数，生成处理后的数据集。

为保证预测结果少受偶然因素影响，设定测试集为 MNIST 数据集后 1000 个样本。数据集数量由 1000 递增至 20000 共 20 组。

距离计算

对于输入的待检测样本，以上述处理方式对其进行同样展开处理，再求出其与所有训练集样本之间的距离。具体求解方法为，对于两个行向量中下标相同的每对元素，均进行求差再平方，随后求解所有 784 对元素的平方和再开方即得到两样本之间的欧氏距离。

k 近邻选取与预测值确定

将距离计算中得到的距离列表进行从小到大排序，则前 k 个样本即为 k

个最近邻，随后以多数表决法选出 k 个样本中出现次数最多的类别作为待检测样本的预测类别，本次 k 值取 3 和 5。

4. 实验结果及分析

4.1 实验结果

训练集依次选取 1000，2000,...20000 共 20 组，测试集选取固定后 1000 个样本，k 值分别选取 3 和 5。算法使用 Python 进行实现、matplotlib 进行绘图，得到结果如图 2。

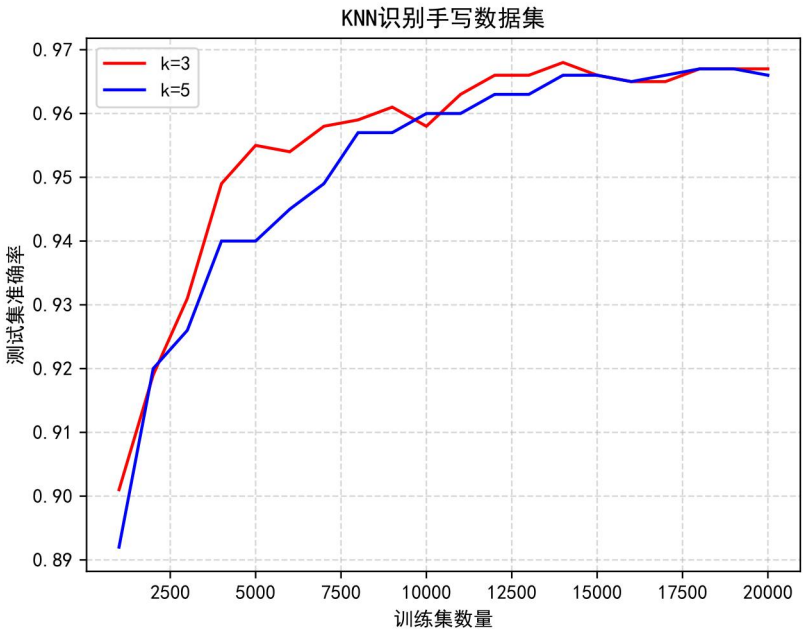

图 2.KNN 识别手写数据集

4.2 结果分析

由图 2 可以看出，随着训练集数量的上升，特征空间被划分的越来越细致，测试集识别准确率总体呈明显上升趋势，最高可达 96.7%；对于相同的训练集数量，整体上来看 k=3 时的准确率略微高于 k=5 时的准确率。

5. 模型总结与感想

后续将数据集数量提高到 50000，在 k=3 时得到准确率达到 97.1%，这样的准确率在 MNIST 数据集预测中已经称得上出色。由于 KNN 模型并没有通过对训练集的学习来不断调整参数这一过程，因此即使在数据集数量较少时，KNN 仍然能够得到较好的预测结果。

本人曾在使用贝叶斯完成 MNIST 数据集识别时提到，贝叶斯模型由于会产生维度灾难而不得不忽略各像素点之间的联系，而这种必要的忽略直接导致了图片丧失了位置信息，随后提到使用全连接神经网络将二维矩阵展开为一维向量进行不断降维求解留存了一定的位置信息，使得准确率有所上升。

可以注意到的是，KNN 算法在计算样本之间距离之前同样将二维矩阵展开为一维行向量，并且准确率同样比贝叶斯更高，但这与全连接神经网络本质上并不相同。

以本人浅薄的认知来看，全连接神经网络以像素点作为最小单位，通过搜寻图片中各像素间的位置信息，在每次降维时构成更大的特征，最终降维至 10 类即可分为数字 0-9，神经网络中包含了巨量的参数，需要通过大量的训练才能够得到较好的识别结果。

而 KNN 算法似乎更加“简单粗暴”，并不去考虑样本内部位置信息，而是以样本作为最小单位考虑其“更像”身边的哪一类样本，以此得到预测结果，因此将二维矩阵展开对于 KNN 来说并没有损失任何判断精度，反而方便后续对距离进行计算。

对于 k 值的选择，如果选择较小的 k 值，就相当于用较小的邻域中的训练集样本进行预测，只有与待检测样本较近的才会对预测结果起作用。这样的优点是可以减少近似误差，但是如果邻近的点是噪声点便有可能会出错，类似于其他模型中的过拟合现象。如果选择较大的 k 值，可以减少估计误差，但与待检测样本较远的训练集样本也会对预测起作用，会增大近似误差，模型属于欠拟合。

不妨用两个极端情况来通俗解释 k 值选择带来的影响。若 k 值选取为 1，则距离最近的测试集样本直接决定了待检测样本的类别，如果该样本为噪声点，会直接导致预测的出错，显然模型并不够合理；若 k 值选取为训练集总数量，那么无论是什么测试集输入，最终的预测结果都是训练集中占比最大的类别，显然模型更不合理。因此，合理选择 k 值对准确性影响较大，但如何选取似乎只能依靠过往经验或遍历尝试。

6. 代码实现

6.1 运行环境

Python	3.6

Matplotlib	3.3.4
Torch	1.10.0+cu113
Torchvision	0.11.1+cu113
Numpy	1.19.5

6.2 源代码

```
import numpy as np
import matplotlib.pyplot as plt
import torch
from torch.utils.data import DataLoader
from torchvision import datasets, transforms


class Knn:
    def __init__(self):
        pass

    def fit(self, X_train, y_train):
        self.Xtr = X_train
        self.ytr = y_train

    def predict(self, k, X_test):
        num_test = X_test.shape[0]
        label_list = []
        for i in range(num_test):
            distances = np.sqrt(np.sum(((self.Xtr - np.tile(X_test[i],
                                                            (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])
        return np.array(label_list)


plt.rcParams["font.sans-serif"] = ["SimHei"]  # 设置字体
```

```
33.  plt.rcParams["axes.unicode_minus"] = False  # 该语句解决图像中的“-”负号的乱码问题
34.
35.  batch_size = 100
36.  path = './'
37.  train_datasets = datasets.MNIST(root=path, train=True, download=True)
38.  test_datasets = datasets.MNIST(root=path, train=False, download=True)
39.
40.  # 加载数据
41.  train_loader = DataLoader(train_datasets, batch_size=batch_size, shuffle=True)
42.  test_loader = DataLoader(test_datasets, batch_size=batch_size, shuffle=True)
43.
44.  # 对训练数据处理
45.  x_train = train_loader.dataset.data.numpy()
46.  x_train = np.reshape(x_train, (x_train.shape[0], -1))
47.  x_train = x_train.astype(np.float)
48.  y_train = train_loader.dataset.targets.numpy()
49.
50.  # 取后 1000 个测试数据
51.  test_num = 1000
52.  x_test = test_loader.dataset.data[-1 * test_num - 1:-1].numpy()
53.  # mean_image = getXmean(x_test)
54.  x_test = np.reshape(x_test, (x_test.shape[0], -1))
55.  x_test = x_test.astype(np.float)
56.  y_test = test_loader.dataset.targets[-1 * test_num - 1:-1].numpy()
57.
58.  acc_k3 = []
59.  acc_k5 = []
60.
61.  # 利用 KNN 计算识别率
62.  for train_num in range(1000, 21000, 1000):
63.      print("When train_num is {}".format(train_num))
64.      for k in [3, 5]:  # 不同 K 值计算识别率
65.          x_train_real = x_train[:train_num]
66.          y_train_real = y_train[:train_num]
67.
68.          classifier = Knn()
69.          classifier.fit(x_train_real, y_train_real)
70.          y_pred = classifier.predict(k, x_test)
71.          num_correct = np.sum(y_pred == y_test)
72.          accuracy = float(num_correct) / test_num
73.          if k == 3:
```

```
74.                 acc_k3.append(accuracy)
75.             else:
76.                 acc_k5.append(accuracy)
77.
78.             print('Got %d / %d correct when k= %d => accuracy: %f' % (num_c
    orrect, test_num, k, accuracy))
79.
80.     x = [i for i in range(1000, 21000, 1000)]
81.     plt.figure()
82.     plt.xlabel("训练集数量")
83.     plt.ylabel("测试集准确率")
84.     plt.plot(x, acc_k3, color="r", label="k=3")
85.     plt.plot(x, acc_k5, color="b", label="k=5")
86.     plt.legend()
87.     plt.grid(linestyle="--", alpha=0.5)
88.     plt.title("KNN 识别手写数据集")
89.     plt.savefig(fname="figure.svg", format="svg")
90.     plt.show()
```