目 录

1. 题目

1.1. 实验目的

通过使用 KNN 算法实现对手写体数字图片中的数字进行识别，帮助学生掌握 KNN 算法的原理和使用场景、掌握导入数据、预处理数据、模型训练和模型评估，使研究生们尽快熟悉人工智能、机器学习与大数据领域的基础算法。

1.2. 实验要求

(1) 理解算法本身，用程序设计语言(不限语言，但建议选择 Python、C、C++或 Java)实现算法。

(2) 除了数学运算(例如求函数值、矩阵求逆等)可以调用现有的库函数，其他一律不准调库。

1.3. 实验内容

sklearn 自带的 load_digits 数据集[1]是常用的机器学习数据集，是 E. Alpaydin 和 C. Kaynak[2]于 1998 年收集整理的数字数据集中的一部分数据。其中包含 10 个种类，分别为数字 0-9，每个类别大约 180 个左右的样本，共 1797 个样本。该数据集包含 16 个特征变量，每个变量对应一个像素值，组成的每幅图像都是 8*8 像素的图片。

本实验需要通过自行实现 KNN 算法，通过利用数据集中的特征，实现对手写数字图片中数字的识别，并掌握导入数据、预处理数据、模型训练和模型评估。

1.4. 实验数据

本实验所用的数据集是手写数字图片数据集，来源为 sklearn.datasets 中的 load_digits 数据集。

load_digits 数据集有 1797 个样本数据，分为 10 个类别，分别为数字 0-9.每个样本包含 16 个特征变量，每个特征向量对应一个像素值，组成的每幅图像都是 8*8 像素的图片。

在实验中，使用数据集中的所有特征变量，实现图片中手写数字的识别。

2. 算法阐述或实验步骤说明

2.1. 算法阐述

2.1.1. KNN 算法简介

K 最邻近算法[3](K-Nearest Neighbour，KNN)是一种聚类算法。同时，它也是最常用的分类算法。当预测一个新的样本时，根据距离它最近的 k 个点的类别来判断新样本的类别。

2.1.2. KNN 算法三要素

(1) k 值

对于 k 值的选择没有固定的经验，可以通过交叉验证选择一个合适的 k 值。若选择 k 值较小，训练误差会减小，但与此同时泛化误差会增大；若选择 k 值较大，可以减少繁华误差，但训练误差会增大。因此，在选择 k 值时应根据实际情况选择一个较合适的值。

(2) 距离的度量方式

距离的度量方式有三种：欧氏距离、曼哈顿距离、闵可夫斯基距离。

① 欧氏距离(最常用)

$$d(dot\_1, dot\_2) = \sqrt{\sum_{i=1}^{n}(dot\_1_i - dot\_2_i)^2}. \tag{2-1}$$

② 曼哈顿距离

$$d(dot\_1, dot\_2) = \sum_{i=1}^{n}\left|dot\_1_i - dot\_2_2\right|. \tag{2-2}$$

③ 闵可夫斯基距离

$$d(dot\_1, dot\_2) = \sqrt[p]{\sum_{i=1}^{n}\left|dot\_1_i - dot\_2_i\right|^p}. \tag{2-3}$$

(3) 分类决策的规则

分类决策的规则一般假设一个样本在特征空间中的 k 个最相似的样本中的大多数属于同一类别，那么该样本也属于该类别。

2.1.3. KNN 算法步骤

KNN 算法的步骤如下:

① 计算新样本点到所有样本点的距离；

② 设置参数k并选择离新样本最近的k个点；

③ 统计这k个点的类别及每个类别的个数；

④ 选择出现频率最多的类别作为新样本的类别，输出结果；

2.2. 实验步骤说明

(1) 搭建实验环境

① 操作系统：Windows10 64 位，CPU 为 Intel(R) Core(TM) i7-8550U；

② 开发语言及软件：Python，PyCharm 2021.3.2 (Community Edition)；

③ 编译器及软件包版本：Python 3.6，scikit-learn 0.19；

(2) 引入库函数

```
import numpy as np
import matplotlib as plt
import sklearn.datasets as datasets
from sklearn.model_selection import train_test_split
from definition import *
```

(3) 数据载入及预处理

将载入数据样本按照$\frac{\text{训练集数据量}}{\text{测试集数据量}}=4:1$的比率进行随机选取。

```
# test
# load data
x, y = datasets.load_digits(return_X_y=True)
# 将数据集按一定比例分为训练集和测试集
x_train, x_test, y_train, y_test = train_test_split(
    x, y, test_size=0.2, random_state=2048)
```

(4) 可视化描述数据信息

读取数据集前两百条，输出为 8*8 像素图片进行查看。

```
# 可视化查看数据集前 200 的数据
for i in range(200):
    x_i = x[i]
    x_i = np.array(x_i)
    x_i = np.reshape(x_i, (8, 8))
    plt.imshow(x_i)
    plt.axis('off')
    plt.savefig('dataset/x_' + str(i) + '.png')
```

(5) 模型训练

① 初始化相关参数

```
# 初始化相关参数
def __init__(self, x_test, x_train, k):
    # 测试数据与已知样本的距离
    self.neighbor_distance = np.zeros((len(x_test), len(x_train)))
    # 样本预测值
    self.prediction = np.zeros((len(x_test,)))
    # 邻近样本点个数
    self.k = k
```

② 定义欧式距离

```
# 计算两点间的欧式距离
def distance(self, dot_1, dot_2):
    return np.sqrt(np.sum(np.power((dot_1 - dot_2), 2)))
```

③ 定义分类决策规则

```
# 预测测试样本属于 k 类中的哪一类
def KNNPrediction(self, distance, y_train):
    # 获取距离最小的前 k 个索引
    k_index = distance.argsort()[:self.k]
    # 获取距离最小的前 k 个物体的类别
    k_prediction = [y_train[index] for index in k_index]
    # 取出类别出现次数最多的类别作为预测结果
    prediction = np.argmax(np.bincount(k_prediction))
    return prediction
```

④ 定义 KNN 过程

```
def KNNTraining(self, x_test, x_train, y_train):
    for i in range(len(x_test)):
        for j in range(len(x_train)):
            self.neighbor_distance[i][j] = self.distance(
                x_test[i], x_train[j])
            # 获取所有样本数据的预测值
            self.prediction[i] = self.KNNPrediction(
                self.neighbor_distance[i], y_train)
    return self.prediction
```

(6) 进行数据测试并输出测试数据集前 20 个样本的测试结果

```
# 使用 KNN 进行预测
knn = KNN(x_test, x_train, k=7)
# 输出预测结果
y_prediction = knn.KNNTraining(x_test, x_train, y_train)
print("=================== 测试结果 ===================")
print()
```

```
print("         INDEX         |         PREDICTION")
print("-----------------------------------------------")
for index in range(20):
    print("          " + str(index) + "                     " +
str(y_prediction[index]))
print("-----------------------------------------------")
print("Accuracy on Test Set: " + str(np.mean(y_prediction == y_test)))
print()
print("===============================================")
```

3. 实验结果与截图

(1) 载入数据后，将数据集前两百条数据输出为 8*8 像素图片，如图 3-1 所示。

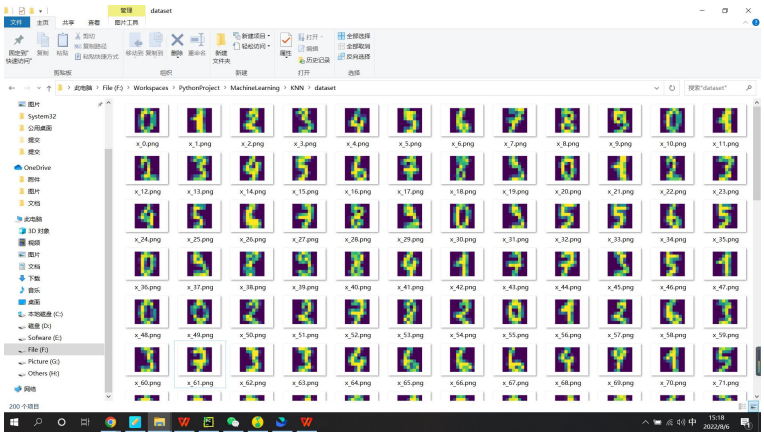
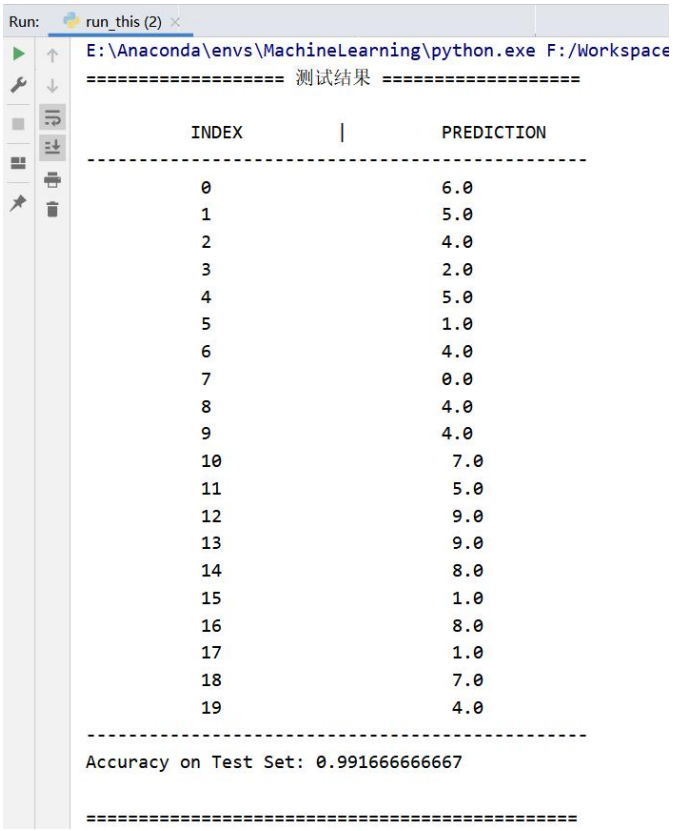


图 3-1 数据集输出为图片数据集

(2) 使用 KNN 算法对测试数据集进行预测，输出预测结果，即每个样本对应索引和预测结果列表，如图 3-2 所示。



```
Run:  run_this (2)
E:\Anaconda\envs\MachineLearning\python.exe F:/Workspace
================== 测试结果 ==================

        INDEX          |          PREDICTION
-------------------------------------------------
            0                        6.0
            1                        5.0
            2                        4.0
            3                        2.0
            4                        5.0
            5                        1.0
            6                        4.0
            7                        0.0
            8                        4.0
            9                        4.0
           10                        7.0
           11                        5.0
           12                        9.0
           13                        9.0
           14                        8.0
           15                        1.0
           16                        8.0
           17                        1.0
           18                        7.0
           19                        4.0
-------------------------------------------------
Accuracy on Test Set: 0.991666666667

=================================================
```

图 3-2KNN 算法预测结果输出

(3) 计算 KNN 算法在测试机上的准确率并输出结果，由图 3-2 可知，KNN 算法在测试数据集上的准确率为 99.17%。

4. 总结

从本次实验，我们可以发现 KNN 算法在对样本进行预测时，预测结果受异常值影响较小。其算法的时间复杂度较低，为 $O(n)$。但若样本容量较小，采用 KNN 算法极易产生错误的分类预测结果，同时在分类前需要计算预测点到所有点之间的距离，若特征数较多，计算量就会明显增加，因此预测速度也会相应变慢。

5. 参考文献

[1] Scikit-learn. sklearn.datasets.load_digits[DB/OL]. [2022-07-21]. https://scikit-learn.org/stable/modules/generated/sklearn.datasets.load_digits.html.

[2] E. Alpaydin, C. Kaynak. Optical Recognition of Handwritten Digits Data Set[DB/OL]. (1998-07-01)[2022-07-21]. https://archive.ics.uci.edu/ml/datasets/Optical+Recognition+of+Handwritten+Digits.

[3] A. Padmanabha, C. Williams. K-nearest Neighbors[DB/OL]. [2022-07-21]. https://brilliant.org/wiki/k-nearest-neighbors/.