实例编写：

KNN实现“手写识别”

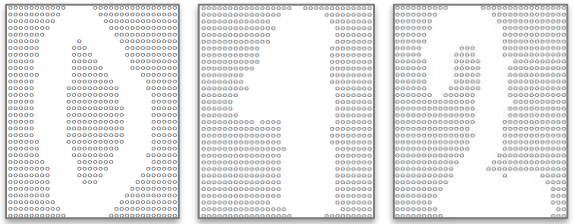
ML21

礼欣

www.python123.org

任务介绍

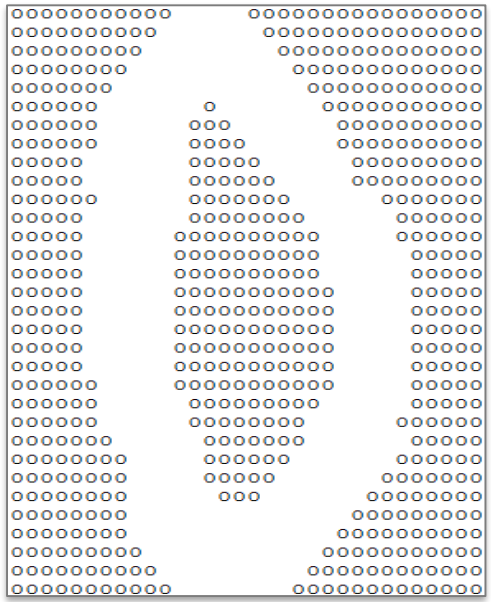
手写数字识别是一个多分类问题，共有10个分类，每个手写数字图像的类别标签是0~9中的其中一个数。例如下面这三张图片的标签分别是0，1，2。



- 本实例利用sklearn来训练一个K最近邻（k-Nearest Neighbor，KNN）分类器，用于识别数据集DBRHD的手写数字。
- 比较KNN的识别效果与多层感知机的识别效果。

KNN的输入

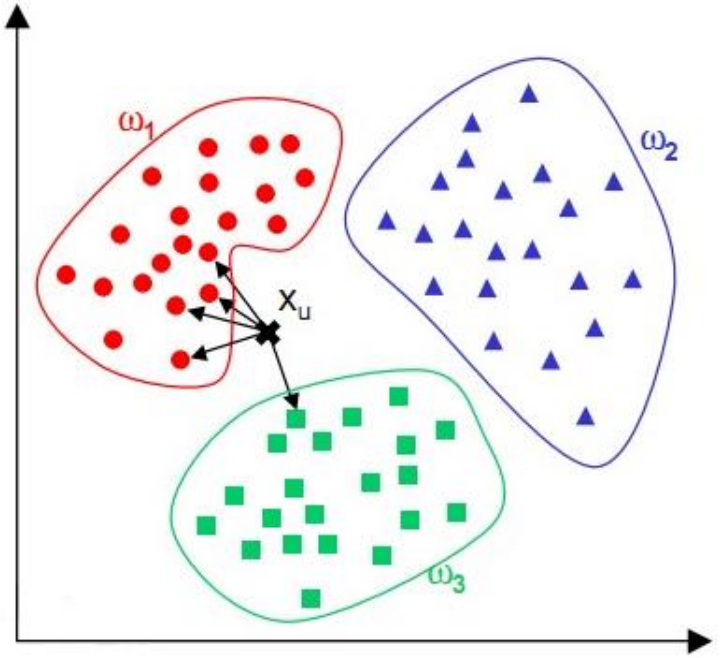
- DBRHD数据集的每个图片是一个由0或1组成的32*32的文本矩阵。
- KNN的输入为图片矩阵展开的一个1024维的向量。



KNN手写识别实体构建

本实例的构建步骤如下：

- 步骤1：建立工程并导入sklearn包
- 步骤2：加载训练数据
- 步骤3：构建KNN分类器
- 步骤4：测试集评价



步骤1：建立工程并导入sklearn包

1） 创建sklearnKNN.py文件

2） 在sklearnKNN.py文件中导入sklearn相关包

```
import numpy as np      #导入numpy工具包
from os import listdir #使用listdir模块，用于访问本地文件
from sklearn import neighbors
```

步骤2：加载训练数据

1）在sklearnKNN.py文件中，定义img2vector函数，将加载的32*32的图片矩阵展开成一列向量

```
def img2vector(fileName):
    retMat = np.zeros([1024],int) #定义返回的矩阵，大小为1*1024
    fr = open(fileName)           #打开包含32*32大小的数字文件
    lines = fr.readlines()        #读取文件的所有行
    for i in range(32):           #遍历文件所有行
        for j in range(32):       #并将01数字存放在retMat中
            retMat[i*32+j] = lines[i][j]
    return retMat
```

步骤2：加载训练数据

2）在sklearnKNN.py文件中定义加载训练数据的函数readDataSet。

```
def readDataSet(path):
    fileList = listdir(path)     #获取文件夹下的所有文件
    numFiles = len(fileList)     #统计需要读取的文件的数目
    dataSet = np.zeros([numFiles,1024],int)     #用于存放所有的数字文件
    hwLabels = np.zeros([numFiles])#用于存放对应的标签(与神经网络的不同)
    for i in range(numFiles):       #遍历所有的文件
        filePath = fileList[i]      #获取文件名称/路径
        digit = int(filePath.split('_')[0])    #通过文件名获取标签
        hwLabels[i] = digit         #直接存放数字，并非one-hot向量
        dataSet[i] = img2vector(path +'/'+filePath)     #读取文件内容
    return dataSet,hwLabels
```

步骤2：加载训练数据

3）在sklearnKNN.py文件中，调用readDataSet和img2vector函数加载数据，将训练的图片存放在train_dataSet中，对应的标签则存在train_hwLabels中

```
train_dataSet, train_hwLabels = readDataSet('trainingDigits')
```

步骤3：构建KNN分类器

1）在sklearnKNN.py文件中，构建KNN分类器：设置查找算法以及邻居点数量(k)值。

- KNN是一种懒惰学习法，没有学习过程，只在预测时去查找最近邻的点，数据集的输入就是构建KNN分类器的过程。
- 构建KNN时我们同时调用了fit函数。

```
knn = neighbors.KNeighborsClassifier(algorithm='kd_tree', n_neighbors=3)
knn.fit(train_dataSet, train_hwLabels)
```

步骤4：测试集评价

1）加载测试集：

```
dataSet,hwLabels = readDataSet('testDigits')
```

2）使用构建好的KNN分类器对测试集进行预测，并计算预测的错误率

```
res = knn.predict(dataSet)  #对测试集进行预测
error_num = np.sum(res != hwLabels) #统计分类错误的数目
num = len(dataSet)          #测试集的数目
print("Total num:",num," Wrong num:", \
      error_num,"  WrongRate:",error_num / float(num))
```

实验效果

邻居数量K影响分析：设置K为1、3、5、7的KNN分类器，对比他们的实验效果

邻居数量K	1	3	5	7
错误数量	12	10	19	24
正确率	0.9873	0.9894	0.9799	0.9746

K=3时正确率最高，当K>3时正确率开始下降，这是由于当样本为稀疏数据集时（本实例只有946个样本），其第k个邻居点可能与测试点距离较远，因此投出了错误的一票进而影响了最终预测结果。

对比实验

KNN分类器vs.多层感知机:

我们取在上节对不同的隐藏层神经元个数、最大迭代次数、学习率进行的各个对比实验中准确率最高（H）与最差（L）的MLP分类器来进行对比，其各个MLP的参数设置如下

MLP代号	隐藏层神经元个数	最大迭代次数	优化方法	初始学习率/学习率
MLP-YH	200	2000	adam	0.0001
MLP-YL	50	2000	adam	0.0001
MLP-DH	100	2000	adam	0.0001
MLP-DL	100	500	adam	0.0001
MLP-XH	100	2000	sgd	0.1
MLP-XL	100	2000	sgd	0.0001

对比实验

将效果最好的KNN分类器（K=3）和效果最差的KNN分类器（K=7）与各个MLP分类器作对比如下：

分类器		MLP隐藏层神经元个数（MLP-Y）	MLP迭代次数（MLP-D）	MLP学习率（MLP-X）	KNN邻居数量
最好	错误数量	37	40	35	10
	正确率	0.9608	0.9577	0.9630	0.9894
最差	错误数量	47	50	222	24
	正确率	0.9503	0.9471	0.7653	0.9746

结论：

- KNN的准确率远高于MLP分类器，这是由于MLP在小数据集上容易过拟合的原因。
- MLP对于参数的调整比较敏感，若参数设置不合理，容易得到较差的分类效果，因此参数的设置对于MLP至关重要。