实验一：KNN分类算法

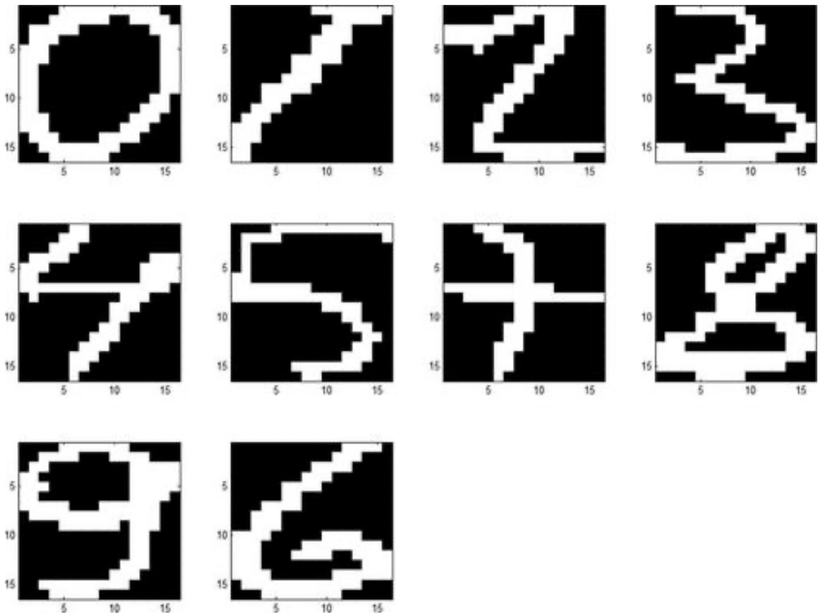
K-Nearest Neighbor Classification

数据读取与预处理

- 数据读取：前 256 列为特征值，后 10 列为数据的类别(以one-hot编码表示)。
- one-hot编码：使用N位状态寄存器对N个状态进行编码，每个状态都有独立的寄存器位，并且在任意时候，只有一位有效。

  e.g. [1 0 0 0 0 0 0 0 0 0] —> 0

  [0 0 0 0 0 0 0 0 0 1] —> 9



- 归一化：$x' = \frac{x - X_{min}}{X_{max} - X_{min}}$

kNN算法实现

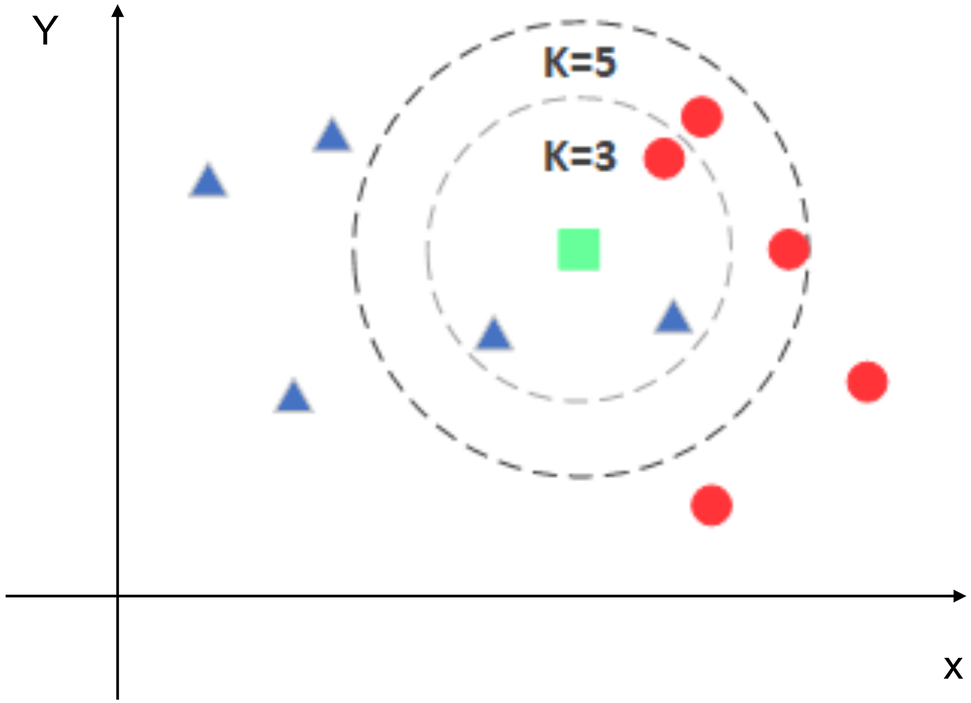
- 计算测试集样本到训练集样本的距离，按照距离递增次序排序

$$dist = \left(\sum_{k=1}^{n} |p_k - q_k|^r \right)^{\frac{1}{r}}$$

  - r = 2，即欧式距离
  - r = 1，即曼哈顿距离
  - r = ∞，即切比雪夫距离，各个坐标距离的最大值

kNN算法实现

- 选择k个最近的训练集样本，对他们的标签进行多数投票，以前k个点出现次数最高的类别作为该测试集样本的预测标签



K = 3 时， 预测结果为三角形

K = 5 时， 预测结果为圆形

K值的选择

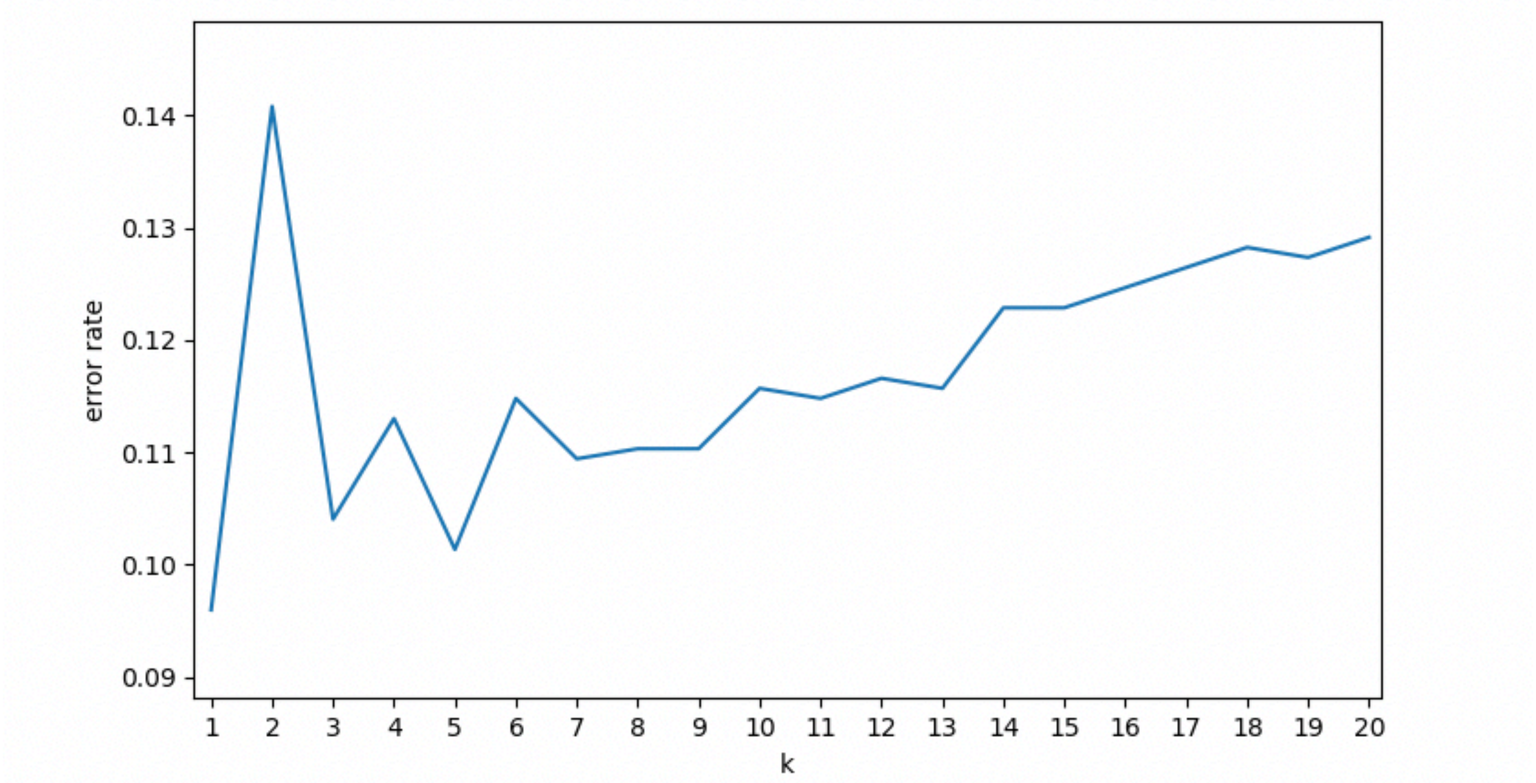
- K 较小，学到的近似误差会减小，只有与实例较近的训练样本才会对决策结果起作用。缺点是预测结果会对近邻的样本点十分敏感，如果近邻的实例点恰好是噪音预测就会出错。（过拟合）

- k较大，可以减少学习的估计误差，但是近似误差会增大。模型变简单。k=N，简单预测实例在训练集中最多的类。（欠拟合）

✦先选一个较小的值，然后用交叉验证法来选取合适的最终值

交叉验证

- K折交叉检验：将训练集划分为 5 份，并依次选择其中的一份作为验证集，其他作为训练集。



与机器学习包测试结果对比

- 在Python 3环境下，调用 sklearn.neighbors.KNeighborsClassifier( )

```
def run_lp(k,train_data,test_data):
    # 创建分类器
    neigh = KNeighborsClassifier(n_neighbors=k)
    # 载入训练数据
    neigh.fit(train_data[...,:-1], train_data[...,-1])
    # 进行测试
    predict_y=neigh.predict(test_data[...,:-1])
    # 统计正确个数
    correct_cnt=0
    for i in range(test_data.shape[0]):
        if test_data[i][256]==predict_y[i]:
            correct_cnt+=1
    return correct_cnt
```

算法优缺点

- 优点
  - 简单有效，容易理解
  - 对异常值不敏感
- 缺点
  - 对内存要求较高，因为该算法存储了所有训练数据
  - 计算量大，预测阶段可能很慢
  - 样本不平衡，如一类居多时，新样本点的K个邻近中该类占据多数，影响预测效果

实验要求

- 基本要求：编程实现kNN算法；给出在不同k值（1，3，5）情况下，kNN算法对手写数字的识别精度
- 中级要求：与 Python 机器学习包中的kNN分类结果进行对比
- 提高要求：画一个以k值为x轴，交叉验证集分类精度为y轴的曲线
- 截止日期：3月16日
- 以学号+姓名(1)的命名形式打包实验代码+实验报告，发送到邮箱 2120190428@mail.nankai.edu.cn

拓展训练

- 数据
  - 旋转，缩放
  - 加噪音
- 算法
  - 优化数据结构，减小计算量，比如KDTree
  - 采用加权投票方法，增加邻近样本的权值，应对样本不平衡问题