第三章　k近邻法

3.1　k近邻算法

注：本章只介绍分类问题的k近邻算法。

输入：训练数据集

$$T=\{(x_1,y_1),(x_2,y_2),...,(x_N,y_N)\}$$

其中，$x_i\in\mathcal{X}\subset R^N$为实例的特征变量，$y_i\in\mathcal{Y}=\{c_1,c_2,...,c_K\}$为实例的类别，$i=1,2,3,...,N$；实例特征变量 x。

输出：实例 x 的类别 y（可以取多类）

算法过程：

- 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，涵盖着k个点的x的邻域记作$N_k(x)$;
- 在$N_k(x)$中根据分类决策规则（如多数表决）决定x的类别y。

$$y=argmax_{c_j}\sum_{x_i\in N_k(x)}I(y_i=c_j),i=1,2,...,N;j=1,2,...,K \tag{3.1}$$

式中，I为指示函数，当$y_i=c_j$时 I 为1，否则 I 为0。 当k = 1时，k邻近算法变为**最邻近算法**，即只考虑距离输入向量x最邻近的点。

3.2　k近邻模型

- 实质：对于特征空间的划分
- 三要素
  - 距离度量
    - 特征空间：一般是n维实数向量空间$\mathcal{R}^n$
    - 距离度量：欧氏距离（或$\mathcal{L}_p$距离或Minkowski（曼哈顿）距离）
      - L_p距离：
        - 设特征空间$\mathcal{X}$是n为实数向量空间$\mathcal{R}^n$，x_i,x_j的L_p距离定义为：

$$L_p(x_i,x_j)=(\sum_{l=1}^{n}|x_i^{(l)}-x_j^{(l)}|^p)^{\frac{1}{p}}$$

    当$p=1$时，此距离成为曼哈顿距离（Minkowski）；$p=2$时，此距离成为欧氏距离；$p=\infty$时，此距离为各个坐标距离的最大值，即

$$L_\infty(x_i,x_j)=max_l|x_i^{(l)}-x_j^{(l)}|$$

  - k值的选择
    - k值直接决定预测的邻域大小。
    - k越小，分类的近似误差越小，但是"学习"的估计误差会变大，模型更复杂，容易发生过拟合。
    - k越大，可以减小学习的估计误差，模型相对简单。

        - 在应用中，k值一般取一个比较小的数值，之后采用交叉验证发选取最优的k值。

    - 分类决策规则
        - 多数表决：由输入实例的k个临近的训练实例中的多数类决定输入实例的类。
        - 解释：如果分类的损失函数为0-1损失函数，分类函数为：

$$f:R^n \rightarrow \{c_1,c_2,...,c_k\}$$

那么误分类的概率为：

$$P(Y \neq f(X)) = 1-P(Y=f(X))$$

对于给定的实例$x \in \mathcal{X}$，其最近邻的k个训练实例点构成集合$N_k(x)$。如果涵盖$N_k(x)$的区域的类别是c_j，那么误分类率为：

$$\frac{1}{k}\sum_{x_i \in N_k(x)}I(y_i \neq c_j) = 1-\frac{1}{k}\sum_{x_i \in N_k(x)}I(y_i = c_j)$$

要使误分类率最小即经验风险最小，就要使$\sum_{x_i \in N_k(x)}I(y_i = c_j)$最大，所以多数表决规则等价于经验风险最小化。

3.3　k近邻法的实现：kd树

- 主要问题：如何对训练数据进行快速的k近邻搜索
- 简单实现方法：线性扫描
- kd树方法：
    - kd树表示一个对于k维空间的切分
    - kd树是二叉树
- 构造算法：
    - 构造根结点，根结点对应与k维空间中包含所有实例点的超矩形区域；
    - 不停的递归，对k维空间进行切分，切分方法为：
        - 在超矩形区域（结点）上选择一个坐标轴和再次坐标轴上的一个**切分点**，确定一个超平面，这个超平面**垂直于选定的坐标轴**，将这个超矩形区域分割为两个部分。这两个部分分别对应两个子结点。
        - 通常切分点选择为坐标轴上的*中位数*，这样得到的kd树就是平衡的，**但搜索效率未必最优**
- 搜索算法：
    - 以最近邻为例：首先找到包含目标点的叶结点，然后从该节点出发，一次回退到父节点；不断查找与目标点最邻近的结点，当确定不可能存在更近的结点时终止。
    - 具体过程：

1. 在树中找到包含目标点x的叶结点：从根结点出发，递归的访问当前结点对应子树，直到当前的子节点为叶结点为止。
2. 以此叶结点为“当前最近点”
3. 递归回退，对于每一个结点规则如下：
    1. 如果该节点的实例点比当前最近距离点距离目标更近，则该实例点“为当前最近点”

2. 对于一个包含最近点的区域，要检查该节点的父节点和兄弟节点是否存在更近的点。如果存在的区域包含距目标点更近的点，移动到另一个自己点上，之后继续递归搜索。不相交时，回退上一级。

4. 回到根结点时，搜索结束

- 平均复杂度为$log(N)$，N为训练实例数。
- kd树适用于训练实例数远大于空间维数时的k近邻搜索，当空间维数接近训练实例数时，效率接近于线性扫描。

附：

普通k近邻法代码实现

```
# numpy实现k近邻法（线性扫描）
'''
数据集：
(1.0,1.1)   A
(1.0,1.0)   A
(0,0)       B
(0,0.1)     B
实验数据：(0.7,0.7)
k = 1
'''
from numpy import *
import operator

def createDataSet():
    group = array([[1.0,1.1],[1.0,1.0],[0,0],[0,0.1]])
    labels = ['A','A','B','B']
    return group,labels

def classify(inX,dataSet,labels,k):
    dataSetSize = dataSet.shape[0]
    diffMat = tile(inX,(dataSetSize,1))-dataSet
    sqDiffMat = diffMat**2
    sqDistances = sqDiffMat.sum(axis = 1)
    distances = sqDistances**0.5
    sortedDistIndicies = distances.argsort()
    classCount = {}
    for i in range(k):
        voteIlabel = labels[sortedDistIndicies[i]]
        classCount[voteIlabel] = classCount.get(voteIlabel,0)+1
    sortedClassCount = sorted(classCount.items(),key = operator.itemge
    return sortedClassCount[0][0]

group,labels = createDataSet()
print(group)
print(labels)
inA = [0.7,0.7]
labelOfA = classify(inA,group,labels,1)
print(labelOfA)
```

kd树伪代码

```
input : 数据集dataSet
output: 二叉树kdTree（根结点treeRoot）
-------
1 struct node* treeRoot
2 //树的结点的个数并非数据集个数
3 //node的结构为：左子节点指针left，右子节点指针right,item数组data
4 //item的结构为：int数组x
3 treeNode->data = dataSet
4 void cutTheSet(node* root,int divisionNum, int mid)
5 {
6   if *root == null:
7     return ;
8   for item in root->data:
9     if(item.x[divisionNum] < mid) :
10      root->left->data.add(item);
11    else:
12      root->right->data.add(item);
13  newDivisionNum = (divisionNum+1)%(root->totalDimension);
14  cutTheSet(root->left,newDivisionNum,mom(root->left->data.getDivisi
15  cutTheSet(root->right,newDivisionNum,mom(root->right->data.getDivi
16  return root;
17  }
```