Ecperiment2-2: KNN

Time: 2022/3/11
Location: Science_Building_119
Name: 易弘睿
Number: 20186103

Part I: Review

算法流程：
1.计算测试对象到训练集中每个对象的距离
2.按照距离的远近排序
3.选取与当前测试对象最近的k的训练对象,作为该测试对象的邻居
4.统计这k个邻居的类别频次
5.k个邻居里频次最高的类别，即为测试对象的类别

Part II: Introduction

对初始代码的修改主要如下：
1. 对代码按照不同部分功能进行命名；
2. 对代码进行每行注释；
3. 优化和修改KNN算法逻辑。

Part III: Annotation

1. 数据库的导入

In [1]:

```
import numpy as np                # 导入numpy库
import operator                   # 导入operator
import matplotlib.pyplot as plt # 导入matplotlib
```

2. KNN的定义

In [2]:

```
def knn(trainData, testData, labels, k):   #定义参数
    # 计算训练样本的行数
    rowSize = trainData.shape[0]
    # 计算训练样本和测试样本的差值
    diff = testData - trainData
    # 计算差值的平方和
    sqrDiff = diff ** 2
    sqrDiffSum = sqrDiff.sum(axis=1)
    # 计算距离
    distances = sqrDiffSum ** 0.5
    # 对所得的距离从低到高进行排序
    sortDistance = distances.argsort()
    # argsort函数将数组x中的元素从小到大排序，并且取出他们对应的索引，然后返回
    count = {}

    for i in range(k):
        vote = labels[sortDistance[i]]
        count[vote] = count.get(vote, 0) + 1
    # 对类别出现的频数从高到低进行排序
    sortCount = sorted(count.items(), key=operator.itemgetter(1), reverse=True)
    # 返回出现频数最高的类别
    return sortCount[0][0]

# 返回出的类别则为测试样本的种类
```

3. 数据集的制作

In [3]:

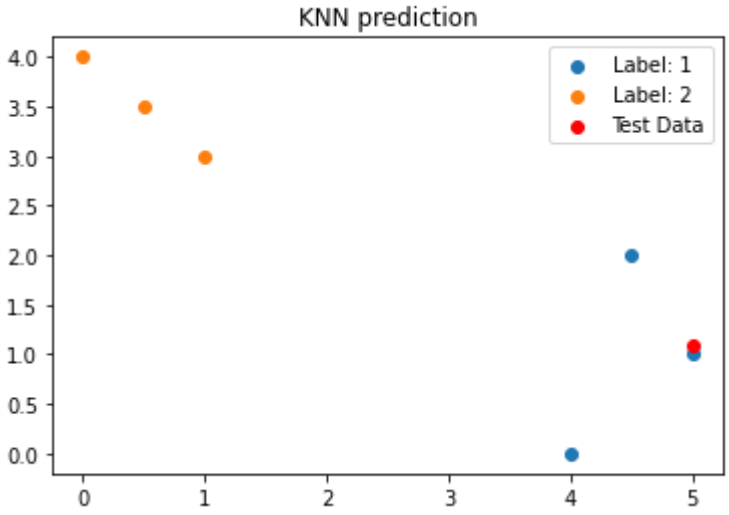
```
# 制作trainData
trainData = np.array([[5, 1], [4, 0],[4.5, 2] ,[1,3], [0, 4],[0.5,3.5]])
labels = ['1', '1','1','2', '2', '2']
```

4. 测试数据

In [4]:

```
testData = [5, 1.1]
fig = plt.figure()
ax = fig.gca()
# 绘制初始数据图
ax.scatter(trainData[:3,0],trainData[:3,1])
ax.scatter(trainData[3:,0],trainData[3:,1])
# 绘制待预测的点
ax.scatter([testData[0]], [testData[1]], color="r")
# 打印图例
plt.legend(labels=["Label: 1","Label: 2", "Test Data"])
# 打印标题
plt.title("KNN prediction")
# 测试数据
X = knn(trainData, testData, labels, 4)
print(X)
```

1



Part III: Conclusion

根据输出结果我们可以发现，knn算法可以很有效解决分类问题 算法复杂度低，速度快 结合实习经验，在工作中，knn算法是可以很快速的给出有效的分类 但是也存在问题 以上代码为例 在解决多类别问题时，当各个类别数据特征相差较大，knn效果很不多 但是在解决数据特征相差不大，而且特征维度较多的时候，每个特征权重就应该加入考虑 单纯的knn算法就已经很难解决了 我们可以给每种特征加入不同的权重数据，或者使用PCA降维等降维方法降低多特征数据的复杂程度 这样在面对多特征问题的时候可以更加符合实际的要求