explanation

olxh

22 .5 .3

1 K近邻

1.1 K近邻思路

K近邻法(k-nearest neighbor,k-NN)存在一个样本集合，当我们对新的数据进行比较的时候，我们选取前K个进行比较。通常K是不大于20的整数，最后，选出K个最相似数据中出现次数最多的分类，作为我们新数据的分类。

对于每一个样本，我们有一个n维度的向量v，向量里面的值代表了这个样本中n个不同的特征值，当我们进行比较的时候，我们直接通过比较两个向量之间的差距即可。

此外，对于每一个样本，它都有一个自己所属的分类。

我们可以运用类似两点之间的距离公式计算距离。我们定义两个样本之间的“差距”为：

对于两个样本A和B而言，它们之间的“差”可以用下面这个公式来表示

$$|AB| = \sqrt{\sum_{i=1}^{n}(A_i - B_i)^2}$$

对于一个只有两个特征的样本，这个公式我们应该很熟悉，就类似两点之间的距离公式。

接着，对于一个新加入的样本，我们可以分别求出它与其它各个样本的差距，由小到大选出前K个进行计算。通过找出这K个样本里面出现最多的分类来确定当下这个点的所属分类。

2 代码的实现

2.1 准备数据集

下面是一段准备数据集的代码：

```
def createDataSet():
group = np.array([[1,101],[5,89],[108,5],[115,8]])
labels = ['1','1','2','2']
return group,labels
```

2.2 主函数处理

接着是处理的主函数

```
def classify0(inX,dataSet,labels,k):
#shape[0] means the rows
dataSetSize = dataSet.shape[0]
#repeat the array
diffMat = np.tile(inX, (dataSetSize, 1)) - dataSet

#count the differences between two vectors
sqDiffMat = diffMat ** 2
sqDistances = sqDiffMat.sum(axis = 1)
distances = sqDistances ** 0.5
sortedDistIndices = distances.argsort()



#count the apperances of each label
classCount = {}
for i in range (k) :
    voteIlabel = labels[sortedDistIndices[i]]
    classCount[voteIlabel] = classCount.get(voteIlabel,0)

#itemgetter was used for sorting
#reverse means sorting from big to small ones
```

```
    sortedClassCount = sorted(classCount.items(),key=operator.itemget

    return sortedClassCount[0][0]
```

3 不足与改进

因为训练集数量过少，现在我们得到的结果并不是很令人满意，我们可以通过提高数据的训练强度来达到我们的目的。