山东大学 软件 学院

模式识别 课程实验报告

|  |  |  |  |
| --- | --- | --- | --- |
| 学号：201500301229 | 姓名：李洪德 | | 班级：15级二班 |
| 实验题目：kNN算法 | | | |
| 实验学时：4 | | 实验日期：2017.12.25 | |
| 实验目的：  编写并掌握k近邻算法的思想和实现方法 | | | |
| 硬件环境：  Dell laptop  Intel i7 5500U | | | |
| 软件环境：  操作系统：Windows 8.1  使用语言：Python 3.6  开发软件包：Numpy  编译器：pycharm | | | |
| 实验步骤与内容：  实验数据分析：  **数据集介绍：**  digits 目录下有两个文件夹，分别是:  trainingDigits：训练数据，1934个文件，每个数字大约200个文件。  testDigits：测试数据，946个文件，每个数字大约100个文件。  每个文件中存储一个手写的数字，文件的命名类似0\_7.txt，第一个数字0表示文件中的手写数字是0，后面的7是个序号。  我们使用目录trainingDigits中的数据训练分类器，使用目录testDigits中的数据测试分类器的效果。两组数据没有重叠，你可以检查一下这些文件夹的文件是否符合要求。根据这些数据我们开始实现KNN算法。   1. 实现最基本的KNN算法，使用trainingDigits文件夹下的数据，对testDigits中的数据进行预测。（K赋值为1，使用欧氏距离，多数投票决定分类结果） 2. 改变K的值，并观察对正确率的影响。 3. 更改距离度量方式，更改投票方式（距离加权），分析错误率。   算法分析：  投票的方法，这里写了两种，一种是平均法，以k近邻中类别多的作为它的标签，第二种是加权平均法，最近的权重最大，k个点总权重为1 用dnum = k\*(k-1)/2 (k+i)/dnum作为第i个点的权重。  *#平均* **def** average(k, sortDis, labels):  classCount={}  **for** i **in** range(k):  voteIlabel=labels[sortDis[i]]  classCount[voteIlabel] = classCount.get(voteIlabel,0)+1  sortedClassCount = sorted(classCount.items(),key=operator.itemgetter(1),reverse=**True**)  **return** sortedClassCount *#加权平均* **def** wAverage(k, sortDis, labels):  classCount={}  dnum = k\*(k+1)/2  **for** i **in** range(k):  voteIlabel=labels[sortDis[i]]  classCount[voteIlabel] = classCount.get(voteIlabel,0)+(k-i)/dnum  sortedClassCount = sorted(classCount.items(),key=operator.itemgetter(1),reverse=**True**)  **return** sortedClassCount  根据距离判断一个给定样本的类别，这里用了两种距离度量的方法，一种是欧氏距离，一种是曼哈顿距离。其中投票的方法可以更改。  *#欧氏距离* **def** classify2(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  sortDis=distances.argsort()  sortedClassCount = average(k, sortDis, labels)#平均法  **return** sortedClassCount[0][0] *#曼哈顿距离* **def** classify1(inX, dataSet, labels, k):  dataSetSize = dataSet.shape[0]diffMat = fabs(tile(inX,(dataSetSize,1))-dataSet)  distances = diffMat.sum(axis=1)  sortDis=distances.argsort()  sortedClassCount = wAverage(k, sortDis, labels)#加权平均法  **return** sortedClassCount[0][0]  编写将图片转化为矩阵的函数：  **def** img2vector(filename):  returnVect = zeros((1,1024))*#图片是32×32的* **with** open(filename) **as** fr:  **for** i **in** range(32):  lineStr = fr.readline()  **for** j **in** range(32):  returnVect[0, 32\*i+j] = int(lineStr[j])  **return** returnVect  编写识别手写数字的函数：  **def** hwClassTest():  *#输入近邻数* k = int(input(**'Input k:'**))  *#根据训练集，生成训练矩阵，存在矩阵trainMat中  # 用于后面的计算* hwLabels = []  trainingFileList = listdir(**'digits/trainingDigits'**)  m = len(trainingFileList)  trainMat = zeros((m,1024))  **for** i **in** range(m):  fileName = trainingFileList[i]  fileStr = fileName.split(**'.'**)[0]  classNum = int(fileStr.split(**'\_'**)[0])  hwLabels.append(classNum)  trainMat[i,:] = img2vector(**'digits/trainingDigits/%s'** % fileName)  testFileList = listdir(**'digits/testDigits'**)   *#计算给定的图片的k近邻* err = 0.0  n = len(testFileList)  **for** i **in** range(n):  filename = testFileList[i]  fileStr = filename.split(**'.'**)[0]  classNum = int(fileStr.split(**'\_'**)[0])  vectorUnderTest = img2vector(**'digits/testDigits/%s'** % filename)  classifierResult = classify1(vectorUnderTest,trainMat,hwLabels,k)   print(**'the classifier came back with: %d,the real number is: %d'** % (classifierResult,classNum))  **if** (classifierResult != classNum):  err += 1.0  print(**'\nthe total number of error is: %d'** % err)  print(**'\nthe total error rate is: %f'** % (err/float(n)))  实验结果：  每个图片一个输出，输出比较多，只是稍微截几张有代表性的          算法运行：  在python shell中输入kNN.hwClassTest()  记得把数据集放到相同的目录下  实验结果分析：  当用平均法的时候采用欧氏距离的结果与用曼哈顿距离的结果是一样的，因为数据全是0或1 。当采用  欧氏距离：  k=1：  the total number of error is: 13  the total error rate is: 0.013742  k=2：  the total number of error is: 13  the total error rate is: 0.013742  k=3：  the total number of error is: 10  the total error rate is: 0.010571  k=4：  the total number of error is: 11  the total error rate is: 0.011628  可以看到k=3时效果最好。  当采用加权平均法时，可以降低错误率，但可能受限于测试集样本的数量，效果没有太明显。  参考书籍及网站：  《机器学习》 周志华  《machine learning in action》 Peter Harrington  实验二  K-D树：  特征匹配算子大致可以分为两类。一类是线性扫描法，即将数据集中的点与查询点逐一进行距离比较，也就是穷举，缺点很明显，就是没有利用数据集本身蕴含的任何结构信息，搜索效率较低，第二类是建立数据索引，然后再进行快速匹配。因为实际数据一般都会呈现出簇状的聚类形态，通过设计有效的索引结构可以大大加快检索的速度。索引树属于第二类，其基本思想就是对搜索空间进行层次划分。根据划分的空间是否有混叠可以分为Clipping和Overlapping两种。前者划分空间没有重叠，其代表就是k-d树；后者划分空间相互有交叠，其代表为R树。  参考网站：<https://www.cnblogs.com/eyeszjwang/articles/2429382.html>  这里介绍的很详细。  maxHeap最大堆算法：  学数据结构的时候写过这个算法，放到下面  template<class T>  void MaxHeap<T>::Initialize(T a[],int size,int ArraySize){  //初始化为非最大堆  delete []heap;  //heap=a;  CurrentSize=size;  MaxSize=ArraySize;  for(int i=0;i<CurrentSize;i++){  heap[i+1]=a[i];  }  //调整称为最大堆  for(int i=CurrentSize/2;i>=1;i--){  T y=heap[i];  int c=2\*i;  while(c<=CurrentSize){  if(c+1<=CurrentSize&&heap[c]<heap[c+1]) c++;  if(y>=heap[c]) break;  heap[c/2]=heap[c];  c=c\*2;  }  heap[c/2]=y;  }  }  可以有线性时间的初始化速度，查找最大值时是常数时间。  相关介绍网站：<http://blog.csdn.net/genios/article/details/8157031>  <http://blog.csdn.net/xiaoxiaoxuewen/article/details/7570621/> | | | |
| 结论分析与体会： | | | |