机器学习算法day02\_KNN分类算法及应用

# 课程大纲

|  |  |
| --- | --- |
| KNN分类算法原理 | KNN概述 |
| KNN算法图示 |
| KNN算法要点 |
| KNN算法不足之处 |
| KNN分类算法Python实战 | KNN简单数据分类实践 |
| KNN实现手写数字识别 |
| KNN算法补充 | KNN算法中k值的选取 |
| 类别判定 |
| 如何选择合适的衡量距离 |
| 训练样本/性能问题 |

课程目标：

1. 理解KNN算法的核心思想
2. 理解KNN算法的实现
3. 掌握KNN算法的应用步骤：数据处理、建模、运算和结果判定

# 1. kNN分类算法原理

## 1.1 概述

K最近邻（k-Nearest Neighbor，KNN）**分类算法**是最简单的机器学习算法。

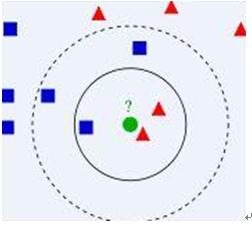
KNN算法的指导思想是“近朱者赤，近墨者黑”，由你的邻居来推断出你的类别。

**本质上，KNN算法就是用距离来衡量样本之间的相似度**

## 1.2 算法图示

* 从训练集中找到和新数据最接近的k条记录，然后根据多数类来决定新数据类别。
* 算法涉及3个主要因素：

1. 训练数据集
2. 距离或相似度的计算衡量
3. k的大小



* 算法描述

1. 已知两类“先验”数据，分别是蓝方块和红三角，他们分布在一个二维空间中
2. 有一个未知类别的数据（绿点），需要判断它是属于“蓝方块”还是“红三角”类
3. 考察离绿点***最近***的3个（或k个）数据点的类别，占多数的类别即为绿点判定类别

## 1.3 算法要点

### 1.3.1、计算步骤

计算步骤如下：

1）算距离：给定测试对象，计算它与训练集中的每个对象的距离

2）找邻居：圈定距离最近的k个训练对象，作为测试对象的近邻

3）做分类：根据这k个近邻归属的主要类别，来对测试对象分类

### 1.3.2、相似度的衡量

* 距离越近应该意味着这两个点属于一个分类的可能性越大。

但，距离不能代表一切，有些数据的相似度衡量并不适合用距离

* 相似度衡量方法：包括欧式距离、夹角余弦等。

*（简单应用中，一般使用欧氏距离，但对于文本分类来说，使用余弦(cosine)来计算相似度就比欧式(Euclidean)距离更合适）*

### 1.3.3、类别的判定

* 简单投票法：少数服从多数，近邻中哪个类别的点最多就分为该类。
* 加权投票法：根据距离的远近，对近邻的投票进行加权，距离越近则权重越大（权重为距离平方的倒数）

## 1.4 算法不足之处

1. 样本不平衡容易导致结果错误

* 如一个类的样本容量很大，而其他类样本容量很小时，有可能导致当输入一个新样本时，该样本的K个邻居中大容量类的样本占多数。
* 改善方法：对此可以采用权值的方法（和该样本距离小的邻居权值大）来改进。

1. 计算量较大

* 因为对每一个待分类的文本都要计算它到全体已知样本的距离，才能求得它的K个最近邻点。
* 改善方法：事先对已知样本点进行剪辑，事先去除对分类作用不大的样本。

该方法比较适用于样本容量比较大的类域的分类，而那些样本容量较小的类域采用这种算法比较容易产生误分。

# 2. KNN分类算法Python实战

## 2.1 kNN简单数据分类实践

### 2.1.1 需求

<比如：计算地理位置的相似度>

……

有以下先验数据，使用knn算法对未知类别数据分类

|  |  |  |
| --- | --- | --- |
| 属性1 | 属性2 | 类别 |
| 1.0 | 0.9 | A |
| 1.0 | 1.0 | A |
| 0.1 | 0.2 | B |
| 0.0 | 0.1 | B |

未知类别数据

|  |  |  |
| --- | --- | --- |
| 属性1 | 属性2 | 类别 |
| 1.2 | 1.0 | ? |
| 0.1 | 0.3 | ? |

### 2.1.2 Python实现

首先，我们新建一个kNN.py脚本文件，文件里面包含两个函数，一个用来生成小数据集，一个实现kNN分类算法。代码如下：

|  |
| --- |
| #########################################  # kNN: k Nearest Neighbors  # 输入: newInput: (1xN)的待分类向量  # dataSet: (NxM)的训练数据集  # labels: 训练数据集的类别标签向量  # k: 近邻数    # 输出: 可能性最大的分类标签  #########################################  from numpy import \*  import operator  #创建一个数据集，包含2个类别共4个样本  def createDataSet():  # 生成一个矩阵，每行表示一个样本  group = array([[1.0, 0.9], [1.0, 1.0], [0.1, 0.2], [0.0, 0.1]])  # 4个样本分别所属的类别  labels = ['A', 'A', 'B', 'B']  return group, labels  # KNN分类算法函数定义  def kNNClassify(newInput, dataSet, labels, k):  numSamples = dataSet.shape[0] # shape[0]表示行数  ## step 1: 计算距离  # tile(A, reps): 构造一个矩阵，通过A重复reps次得到  # the following copy numSamples rows for dataSet  diff = tile(newInput, (numSamples, 1)) - dataSet # 按元素求差值  squaredDiff = diff \*\* 2 #将差值平方  squaredDist = sum(squaredDiff, axis = 1) # 按行累加  distance = squaredDist \*\* 0.5 #将差值平方和求开方，即得距离  ## step 2: 对距离排序  # argsort() 返回排序后的索引值  sortedDistIndices = argsort(distance)  classCount = {} # define a dictionary (can be append element)  for i in xrange(k):  ## step 3: 选择k个最近邻  voteLabel = labels[sortedDistIndices[i]]  ## step 4: 计算k个最近邻中各类别出现的次数  # when the key voteLabel is not in dictionary classCount, get()  # will return 0  classCount[voteLabel] = classCount.get(voteLabel, 0) + 1  ## step 5: 返回出现次数最多的类别标签  maxCount = 0  for key, value in classCount.items():  if value > maxCount:  maxCount = value  maxIndex = key  return maxIndex |

然后调用算法进行测试：

|  |
| --- |
| import kNN  from numpy import \*  #生成数据集和类别标签  dataSet, labels = kNN.createDataSet()  #定义一个未知类别的数据  testX = array([1.2, 1.0])  k = 3  #调用分类函数对未知数据分类  outputLabel = kNN.kNNClassify(testX, dataSet, labels, 3)  print "Your input is:", testX, "and classified to class: ", outputLabel  testX = array([0.1, 0.3])  outputLabel = kNN.kNNClassify(testX, dataSet, labels, 3)  print "Your input is:", testX, "and classified to class: ", outputLabel |

这时候会输出

|  |
| --- |
| Your input is: [ 1.2 1.0] and classified to class: A  Your input is: [ 0.1 0.3] and classified to class: B |

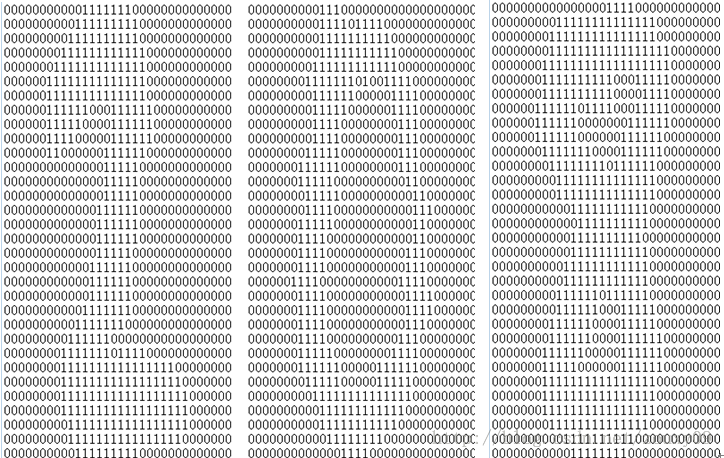
## 2.2 kNN实现手写数字识别

### 2.2.1 需求

利用一个手写数字“先验数据”集，使用knn算法来实现对手写数字的自动识别；

先验数据（训练数据）集：

* 数据维度比较大，样本数比较多。
* 数据集包括数字0-9的手写体。
* 每个数字大约有200个样本。
* 每个样本保持在一个txt文件中。
* 手写体图像本身的大小是32x32的二值图，转换到txt文件保存后，内容也是32x32个数字，0或者1，如下：



数据集压缩包解压后有两个目录：

* 目录trainingDigits存放的是大约2000个训练数据
* 目录testDigits存放大约900个测试数据。

### 2.2.2 模型分析

本案例看起来跟前一个案例几乎风马牛不相及，但是一样可以用KNN算法来实现。没错，这就是机器学习的魅力，不过，也是机器学习的难点**：模型抽象能力！**

思考：

1. 手写体因为每个人，甚至每次写的字都不会完全精确一致，所以，识别手写体的关键是“相似度”
2. 既然是要求样本之间的相似度，那么，首先需要将样本进行抽象，将每个样本变成一系列特征数据（即特征向量）
3. 手写体在直观上就是一个个的图片，而图片是由上述图示中的像素点来描述的，样本的相似度其实就是像素的位置和颜色之间的组合的相似度
4. 因此，将图片的像素按照固定顺序读取到一个个的向量中，即可很好地表示手写体样本
5. 抽象出了样本向量，及相似度计算模型，即可应用KNN来实现

### 2.2.3 python实现

新建一个kNN.py脚本文件，文件里面包含四个函数：

1. 一个用来生成将每个样本的txt文件转换为对应的一个向量，
2. 一个用来加载整个数据集，
3. 一个实现kNN分类算法。
4. 最后就是实现加载、测试的函数。

|  |
| --- |
| #########################################  # kNN: k Nearest Neighbors  # 参数: inX: vector to compare to existing dataset (1xN)  # dataSet: size m data set of known vectors (NxM)  # labels: data set labels (1xM vector)  # k: number of neighbors to use for comparison    # 输出: 多数类  #########################################  from numpy import \*  import operator  import os  **# KNN分类核心方法**  def kNNClassify(newInput, dataSet, labels, k):  numSamples = dataSet.shape[0] # shape[0]代表行数  ## step 1: 计算欧式距离  # tile(A, reps): 将A重复reps次来构造一个矩阵  # the following copy numSamples rows for dataSet  diff = tile(newInput, (numSamples, 1)) - dataSet # Subtract element-wise  squaredDiff = diff \*\* 2 # squared for the subtract  squaredDist = sum(squaredDiff, axis = 1) # sum is performed by row  distance = squaredDist \*\* 0.5  ## step 2: 对距离排序  # argsort()返回排序后的索引  sortedDistIndices = argsort(distance)  classCount = {} # 定义一个空的字典  for i in xrange(k):  ## step 3: 选择k个最小距离  voteLabel = labels[sortedDistIndices[i]]  ## step 4: 计算类别的出现次数  # when the key voteLabel is not in dictionary classCount, get()  # will return 0  classCount[voteLabel] = classCount.get(voteLabel, 0) + 1  ## step 5: 返回出现次数最多的类别作为分类结果  maxCount = 0  for key, value in classCount.items():  if value > maxCount:  maxCount = value  maxIndex = key  return maxIndex  **# 将图片转换为向量**  def img2vector(filename):  rows = 32  cols = 32  imgVector = zeros((1, rows \* cols))  fileIn = open(filename)  for row in xrange(rows):  lineStr = fileIn.readline()  for col in xrange(cols):  imgVector[0, row \* 32 + col] = int(lineStr[col])  return imgVector  **# 加载数据集**  def loadDataSet():  ## step 1: 读取训练数据集  print "---Getting training set..."  dataSetDir = 'E:/Python/ml/knn/'  trainingFileList = os.listdir(dataSetDir + 'trainingDigits') # 加载测试数据  numSamples = len(trainingFileList)  train\_x = zeros((numSamples, 1024))  train\_y = []  for i in xrange(numSamples):  filename = trainingFileList[i]  # get train\_x  train\_x[i, :] = img2vector(dataSetDir + 'trainingDigits/%s' % filename)  # get label from file name such as "1\_18.txt"  label = int(filename.split('\_')[0]) # return 1  train\_y.append(label)  ## step 2:读取测试数据集  print "---Getting testing set..."  testingFileList = os.listdir(dataSetDir + 'testDigits') # load the testing set  numSamples = len(testingFileList)  test\_x = zeros((numSamples, 1024))  test\_y = []  for i in xrange(numSamples):  filename = testingFileList[i]  # get train\_x  test\_x[i, :] = img2vector(dataSetDir + 'testDigits/%s' % filename)  # get label from file name such as "1\_18.txt"  label = int(filename.split('\_')[0]) # return 1  test\_y.append(label)  return train\_x, train\_y, test\_x, test\_y  **# 手写识别主流程**  def testHandWritingClass():  ## step 1: 加载数据  print "step 1: load data..."  train\_x, train\_y, test\_x, test\_y = loadDataSet()  ## step 2: 模型训练.  print "step 2: training..."  pass  ## step 3: 测试  print "step 3: testing..."  numTestSamples = test\_x.shape[0]  matchCount = 0  for i in xrange(numTestSamples):  predict = kNNClassify(test\_x[i], train\_x, train\_y, 3)  if predict == test\_y[i]:  matchCount += 1  accuracy = float(matchCount) / numTestSamples  ## step 4: 输出结果  print "step 4: show the result..."  print 'The classify accuracy is: %.2f%%' % (accuracy \* 100) |

测试非常简单，只需要在命令行中输入：

|  |
| --- |
| import kNN  kNN.testHandWritingClass() |

输出结果如下：

|  |
| --- |
| step 1: load data...  ---Getting training set...  ---Getting testing set...  step 2: training...  step 3: testing...  step 4: show the result...  The classify accuracy is: 98.84% |

# KNN算法补充

## 3.1、k值设定为多大？

k太小，分类结果易受噪声点影响；k太大，近邻中又可能包含太多的其它类别的点。

（对距离加权，可以降低k值设定的影响）

k值通常是采用交叉检验来确定（以k=1为基准）

经验规则：k一般低于训练样本数的平方根

## 3.2、类别如何判定最合适？

投票法没有考虑近邻的距离的远近，距离更近的近邻也许更应该决定最终的分类，所以加权投票法更恰当一些。而具体如何加权，需要根据具体的业务和数据特性来探索

## 3.3、如何选择合适的距离衡量？

高维度对距离衡量的影响：众所周知当变量数越多，欧式距离的区分能力就越差。

变量值域对距离的影响：值域越大的变量常常会在距离计算中占据主导作用，因此应先对变量进行标准化。

## 3.4、训练样本是否要一视同仁？

在训练集中，有些样本可能是更值得依赖的。

也可以说是样本数据质量的问题

可以给不同的样本施加不同的权重，加强依赖样本的权重，降低不可信赖样本的影响。

## 3.5、性能问题？

kNN是一种懒惰算法，平时不好好学习，考试（对测试样本分类）时才临阵磨枪（临时去找k个近邻）。

懒惰的后果：构造模型很简单，但在对测试样本分类地的系统开销大，因为要扫描全部训练样本并计算距离。

已经有一些方法提高计算的效率，例如压缩训练样本量等。