**目标：**

本章节你需要学习以下内容:

* 在本章中，我们将理解k-最近邻（kNN）算法的概念。
* 我们将使用我们在kNN上的知识来构建基本的OCR应用程序。
* 我们将尝试使用OpenCV附带的数字和字母数据。

## api：

### knn = cv.ml.KNearest\_create()

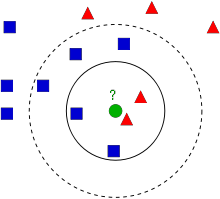
### knn.train(trainData, cv.ml.ROW\_SAMPLE, responses)

### ret, results, neighbours ,dist = knn.findNearest(newcomer, 3)

**了解k-最近邻**

**理论**

kNN是最简单的用于监督学习的分类算法之一。想法也很简单，就是找出测试数据在特征空间中的最近邻居。我们将用下图来介绍它。

[](https://camo.githubusercontent.com/12065308825bf00039e2787af0342aa6b5811c51acdc12acc8303f6f5a08ac1b/68747470733a2f2f646f63732e6f70656e63762e6f72672f342e302e302f6b6e6e5f7468656f72792e706e67) 上图中的对象可以分成两组，蓝色方块和红色三角。每一组也可以称为一个 类。我们可以把所有的这些对象看成是一个城镇中房子，而所有的房子分别属于蓝色和红色家族，而这个城镇就是所谓的特征空间。（你可以把一个特征空间看成是所有点的投影所在的空间。例如在一个 2D 的坐标空间中，每个数据都两个特征 x 坐标和 y 坐标，你可以在 2D 坐标空间中表示这些数据。如果每个数据都有 3 个特征呢，我们就需要一个 3D 空间。N 个特征就需要 N 维空间，这个 N 维空间就是特征空间。在上图中，我们可以认为是具有两个特征色2D 空间）。

现在城镇中来了一个新人，他的新房子用绿色圆盘表示。我们要根据他房子的位置把他归为蓝色家族或红色家族。我们把这过程成为 分类。我们应该怎么做呢？因为我们正在学习看 kNN，那我们就使用一下这个算法吧。

一个方法就是查看他最近的邻居属于那个家族，从图像中我们知道最近的是红色三角家族。所以他被分到红色家族。这种方法被称为简单 近邻，因为分类仅仅决定与它最近的邻居。

但是这里还有一个问题。红色三角可能是最近的，但如果他周围还有很多蓝色方块怎么办呢？此时蓝色方块对局部的影响应该大于红色三角。所以仅仅检测最近的一个邻居是不足的。所以我们检测 k 个最近邻居。谁在这 k 个邻居中占据多数，那新的成员就属于谁那一类。如果 k 等于 3，也就是在上面图像中检测 3 个最近的邻居。他有两个红的和一个蓝的邻居，所以他还是属于红色家族。但是如果 k 等于 7 呢？他有 5 个蓝色和 2 个红色邻居，现在他就会被分到蓝色家族了。k 的取值对结果影响非常大。更有趣的是，如果 k 等于 4呢？两个红两个蓝。这是一个死结。所以 k 的取值最好为奇数。这中根据 k 个最近邻居进行分类的方法被称为 kNN。

在 kNN 中我们考虑了 k 个最近邻居，但是我们给了这些邻居相等的权重，这样做公平吗？以 k 等于 4 为例，我们说她是一个死结。但是两个红色三角比两个蓝色方块距离新成员更近一些。所以他更应该被分为红色家族。那用数学应该如何表示呢？我们要根据每个房子与新房子的距离对每个房子赋予不同的权重。距离近的具有更高的权重，距离远的权重更低。然后我们根据两个家族的权重和来判断新房子的归属，谁的权重大就属于谁。这被称为 修改过的kNN。

那么你在这里看到的重要事情是什么？

* 我们需要整个城镇中每个房子的信息。因为我们要测量新来者到所有现存房子的距离，并在其中找到最近的。如果那里有很多房子，就要占用很大的内存和更多的计算时间。
* 训练和处理几乎不需要时间。

现在让我们在OpenCV中看到它。

**OpenCV中的K-最邻近**

我们这里来举一个简单的例子，和上面一样有两个类。下一节我们会有一个更好的例子。

这里我们将红色家族标记为 Class-0，蓝色家族标记为 Class-1。还要再创建 25 个训练数据，把它们非别标记为 Class-0 或者 Class-1。Numpy中随机数产生器可以帮助我们完成这个任务。

然后借助 Matplotlib 将这些点绘制出来。红色家族显示为红色三角蓝色家族显示为蓝色方块。

import cv2 as cv

import numpy as np

import matplotlib.pyplot as plt

# Feature set containing (x,y) values of 25 known/training data

trainData = np.random.randint(0,100,(25,2)).astype(np.float32)

# Labels each one either Red or Blue with numbers 0 and 1

responses = np.random.randint(0,2,(25,1)).astype(np.float32)

# Take Red families and plot them

red = trainData[responses.ravel()==0]

plt.scatter(red[:,0],red[:,1],80,'r','^')

# Take Blue families and plot them

blue = trainData[responses.ravel()==1]

plt.scatter(blue[:,0],blue[:,1],80,'b','s')

plt.show()

你可能会得到一个与上面类似的图形，但不会完全一样，因为你使用了随机数产生器，每次你运行代码都会得到不同的结果。

下面就是 kNN 算法分类器的初始化，我们要传入一个训练数据集，以及与训练数据对应的分类来训练 kNN 分类器（构建搜索树）。

最后要使用 OpenCV 中的 kNN 分类器，我们给它一个测试数据，让它来进行分类。在使用 kNN 之前，我们应该对测试数据有所了解。我们的数据应该是大小为数据数目乘以特征数目的浮点性数组。然后我们就可以通过计算找到测试数据最近的邻居了。我们可以设置返回的最近邻居的数目。返回值包括：

1. 由 kNN 算法计算得到的测试数据的类别标志（0 或 1）。如果你想使用最近邻算法，只需要将 k 设置为 1，k 就是最近邻的数目。
2. k 个最近邻居的类别标志。
3. 每个最近邻居到测试数据的距离。

让我们看看它是如何工作的。测试数据被标记为绿色。

newcomer = np.random.randint(0,100,(1,2)).astype(np.float32)

plt.scatter(newcomer[:,0],newcomer[:,1],80,'g','o')

knn = cv.ml.KNearest\_create()

knn.train(trainData, cv.ml.ROW\_SAMPLE, responses)

ret, results, neighbours ,dist = knn.findNearest(newcomer, 3)

print( "result: {}\n".format(results) )

print( "neighbours: {}\n".format(neighbours) )

print( "distance: {}\n".format(dist) )

plt.show()

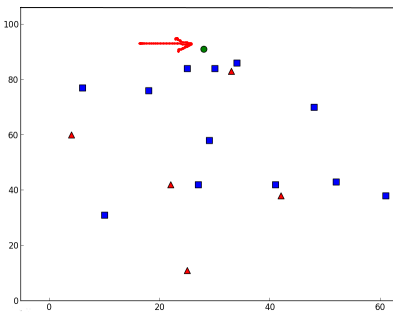
我得到的结果如下：

result: [[ 1.]]

neighbours: [[ 1. 1. 1.]]

distance: [[ 53. 58. 61.]]

这说明我们的测试数据有 3 个邻居，他们都是蓝色，所以它被分为蓝色家族。结果很明显，如下图所示：

[](https://camo.githubusercontent.com/e10ccd616b247f9af6917c1762c6851e02e3b2a519b8a722263161189bcaf098/68747470733a2f2f646f63732e6f70656e63762e6f72672f342e302e302f6b6e6e5f73696d706c652e706e67) 如果我们有大量的数据要进行测试，可以直接传入一个数组。对应的结果同样也是数组。

# 10 new comers

newcomers = np.random.randint(0,100,(10,2)).astype(np.float32)

ret, results,neighbours,dist = knn.findNearest(newcomer, 3)

# The results also will contain 10 labels.

## 完整实例1

|  |  |
| --- | --- |
|  | **import cv2 as cv import numpy as np import matplotlib.pyplot as plt  *# Feature set containing (x,y) values of 25 known/training data* trainData = np.random.randint(0,100,(25,2)).astype(np.float32)  *# Labels each one either Red or Blue with numbers 0 and 1* responses = np.random.randint(0,2,(25,1)).astype(np.float32)  *# Take Red families and plot them* red = trainData[responses.ravel()==0] plt.scatter(red[:,0],red[:,1],80,'r','^')  *# Take Blue families and plot them* blue = trainData[responses.ravel()==1] plt.scatter(blue[:,0],blue[:,1],80,'b','s') *# knn # 测试数据被标记为绿色 # # 回值包括 # 1. 由 kNN算法计算得到的测 数据的类别标志0或1 。 # 如果你想使用最近邻算法 只需 将 k 置为 1 k 就是最近邻的数目。 # 2. k 个最近邻居的类别标志。 # 3. 每个最近邻居到测 数据的 离。* newcomer = np.random.randint(0,100,(1,2)).astype(np.float32) plt.scatter(newcomer[:,0],newcomer[:,1],80,'g','o')  knn = cv.ml.KNearest\_create() knn.train(trainData, cv.ml.ROW\_SAMPLE, responses) ret, results, neighbours ,dist = knn.findNearest(newcomer, 3)  print( "result: {}\n".format(results) ) print( "neighbours: {}\n".format(neighbours) ) print( "distance: {}\n".format(dist) ) plt.show()** |

### 效果：注意：可能每一次的效果都不一样因为他是使用numpy的随机数生成的

|  |
| --- |
|  |

### 使用 kNN 对手写数字 OCR

我们的目的是创建一个可以对手写数字进行识别的程序。为了达到这个目的我们需要训练数据和测试数据。OpenCV附带一个images digits.png（在文件夹opencv/samples/data/中），其中有 5000 个手写数字（每个数字重复 500遍）。每个数字是一个 20x20 的小图。所以第一步就是将这个图像分割成 5000个不同的数字。我们在将拆分后的每一个数字的图像重排成一行含有 400 个像素点的新图像。这个就是我们的特征集，所有像素的灰度值。这是我们能创建的最简单的特征集。我们使用每个数字的前 250 个样本做训练数据，剩余的250 个做测试数据

## 实例代码：knn-num-ocr.py，需要把训练结果保存起来，可以减少不必要的训练

|  |  |
| --- | --- |
|  | **import numpy as np import cv2 as cv from matplotlib import pyplot as plt  img = cv.imread('digits.png') gray = cv.cvtColor(img,cv.COLOR\_BGR2GRAY)  *# Now we split the image to 5000 cells, each 20x20 size* cells = [np.hsplit(row,100) for row in np.vsplit(gray,50)]  *# Make it into a Numpy array. It size will be (50,100,20,20)* x = np.array(cells)  *# Now we prepare train\_data and test\_data.* train = x[:,:50].reshape(-1,400).astype(np.float32) *# Size = (2500,400)* test = x[:,50:100].reshape(-1,400).astype(np.float32) *# Size = (2500,400)  # Create labels for train and test data* k = np.arange(10) train\_labels = np.repeat(k,250)[:,np.newaxis] test\_labels = train\_labels.copy()  *# Initiate kNN, train the data, then test it with test data for k=1* knn = cv.ml.KNearest\_create() knn.train(train, cv.ml.ROW\_SAMPLE, train\_labels) ret,result,neighbours,dist = knn.findNearest(test,k=5)  *# Now we check the accuracy of classification # For that, compare the result with test\_labels and check which are wrong* matches = result==test\_labels correct = np.count\_nonzero(matches) accuracy = correct\*100.0/result.size print( accuracy )  *# 为了避免每次运行程序都要准备和训练分类器，我们最好把它保留，这样在下次运行是时，只需要从文件中读取这些数据开始进行分类就可以了。 # Numpy 函数 np.savetxt，np.load 等可以帮助我们搞定这些。  # save the data,保存训练结果，节省不必要的训练* np.savez('knn\_data.npz',train=train, train\_labels=train\_labels)** |

### 结果

|  |
| --- |
|  |

### 保存的数据

|  |
| --- |
|  |

### 英文字母的 OCR

接下来我们来做英文字母的 OCR。和上面做法一样，但是数据和特征集有一些不同。现在 OpenCV 给出的不是图片了，而是一个数据文件（/samples/cpp/letter-recognition.data）。如果打开它的话，你会发现它有 20000 行，第一样看上去就像是垃圾。实际上每一行的第一列是我们的一个字母标记。接下来的 16 个数字是它的不同特征。这些特征来源于UCI Machine LearningRepository。你可以在此页找到更多相关信息。

有 20000 个样本可以使用，我们取前 10000 个作为训练样本，剩下的10000 个作为测试样本。我们应在先把字母表转换成 asc 码，因为我们不正直接处理字母。

## 实例

|  |  |
| --- | --- |
|  | ***# -\*- encoding:utf-8 -\*- # @日期和时间：2024/7/14 15:10 # @Author: Kenny cai # @File name:knn-letter-recognition.py # @dev tool:PyCharm* import cv2 as cv import numpy as np import matplotlib.pyplot as plt  *# Load the data, converters convert the letter to a number* data= np.loadtxt('letter-recognition.data', dtype= 'float32', delimiter = ',', converters= {0: lambda ch: ord(ch)-ord('A')})  *# split the data to two, 10000 each for train and test* train, test = np.vsplit(data,2)  *# split trainData and testData to features and responses* responses, trainData = np.hsplit(train,[1]) labels, testData = np.hsplit(test,[1])  *# Initiate the kNN, classify, measure accuracy.* knn = cv.ml.KNearest\_create() knn.train(trainData, cv.ml.ROW\_SAMPLE, responses) ret, result, neighbours, dist = knn.findNearest(testData, k=5)  correct = np.count\_nonzero(result == labels) accuracy = correct\*100.0/10000 print( accuracy )** |

### 效果：

|  |
| --- |
|  |