第5章

扫一扫

AI学习助手

KNN分类算法

本章概要

分类是数据分析中非常重要的方法，是对已有数据进行学习，得到一个分类函数或构造出一个分类模型（即通常所说的分类器（Classifier））。

分类函数或模型能够将数据样本对应某个给定的类别，完成数据的类别预测。分类器是机器学习算法中对数据样本进行分类的方法的统称，包含决策树、SVM、逻辑回归、朴素贝叶斯、神经网络等算法。

本章主要介绍K近邻分类算法的原理、算法的核心要素，并以K近邻算法的实现为例，对算法的数据获取、数据预处理、模型实现以及性能评价做了整体介绍。

学习目标

当完成本章的学习后，要求：

（1）了解KNN分类算法的基本概念。

（2）熟悉KNN算法的核心要素。

（3）熟悉距离的度量方法。

（4）掌握使用KNN算法解决实际分类问题。

扫一扫

视频讲解

5.1 KNN分类

分类是使用已知类别的数据样本，训练出分类器，使其能够对未知样本进行分类。分类算法是最为常用的机器学习算法之一，属于监督学习算法。

KNN分类（K-Nearest-Neighbors Classification）算法，又叫K近邻算法。它是概念极其简单，而效果又很优秀的分类算法，于1967年由Cover T和Hart P提出。KNN分

类算法的核心思想是，如果一个样本在特征空间中的 k 个最相似（即特征空间中最邻近）的样本中的大多数属于某一个类别，则该样本也属于这个类别。

如图 5.1 所示，假设已经获取一些动物的特征，且已知这些动物的类别。现在需要识别一个新动物，判断它是哪类动物。

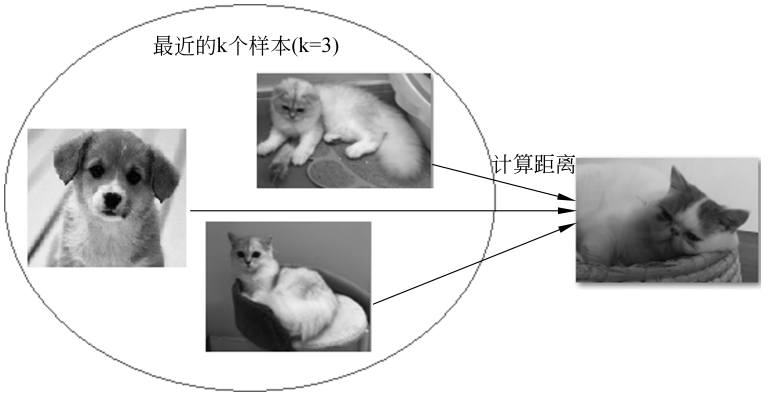


图 5.1　KNN 分类示意图

首先找到与这个物体最接近的 k 个动物。假设 k＝3，则可以找到两只猫和一只狗。由于找到的结果中大多数是猫，则把这个新动物划分为猫类。

KNN 没有专门的学习过程，是基于数据实例的一种学习方法。从上面的描述中不难看出，KNN 方法有以下三个核心要素。

1. k 值

k 值也就是选择几个和新动物相邻的已知动物。如果 k 取值太小，好处是近似误差会减小，只有特征与这个新动物很相似的才对预测新动物的类别起作用。但同时预测结果对近邻的样本点非常敏感，仅由非常近的训练样本决定预测结果。因此会使模型变得复杂，容易过拟合。如果 k 值太大，学习的近似误差会增大，导致分类模糊，即欠拟合。

下面举例看 k 值对预测结果的影响。对图 5.2 中的动物进行分类，当 k＝3 时，分类结果为“猫：狗＝2：1”，所以属于猫；当 k＝6 时，表决结果为“猫：狗：熊猫＝2：3：1”，所以判断目标动物为狗。

那么 k 值到底怎么选取呢？这就涉及距离的度量问题。

2. 距离的度量

距离决定了哪些是邻居哪些不是。度量距离有很多种方法，不同的距离所确定的近邻点不同。平面上比较常用的是欧氏距离。此外，还有曼哈顿距离、余弦距离、球面距离等。例如，图 5.3 中的四个点为训练样本点，对于新的点 new(3,3)进行预测。其中，cat1、cat2、dog1、dog2、dog3 为训练数据，new 为测试数据。

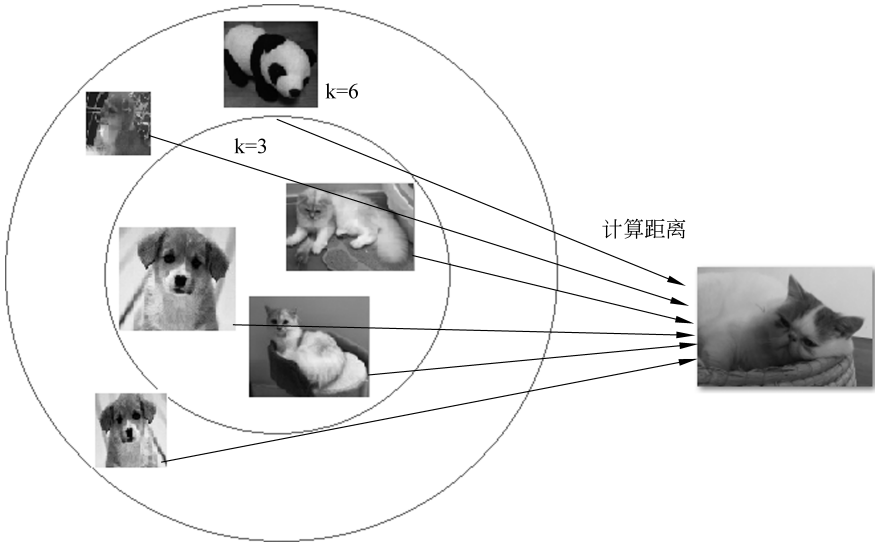


图 5.2　不同 k 值对结果的影响示意图

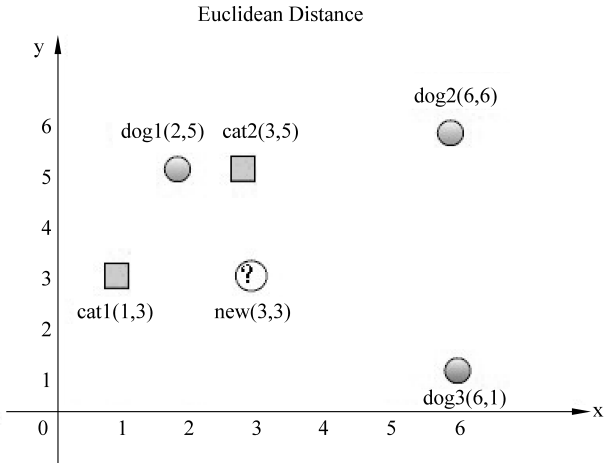


图 5.3　KNN 中距离的度量

通过坐标计算 new(3,3)到各个点的欧氏距离,根据二维平面上的欧氏距离公式

$$d=\sqrt{(x_2-x_1)^2+(y_2-y_1)^2} \tag{5.1}$$

可以得到距离如图 5.4 所示。

3. 分类决策规则

分类结果的确定往往采用多数表决原则,即由输入实例的 k 个最邻近的训练实例中的多数类决定输入实例的类别。

KNN 算法是一个简单高效的分类算法,可用于多个类别的分类,还可以用于回归。

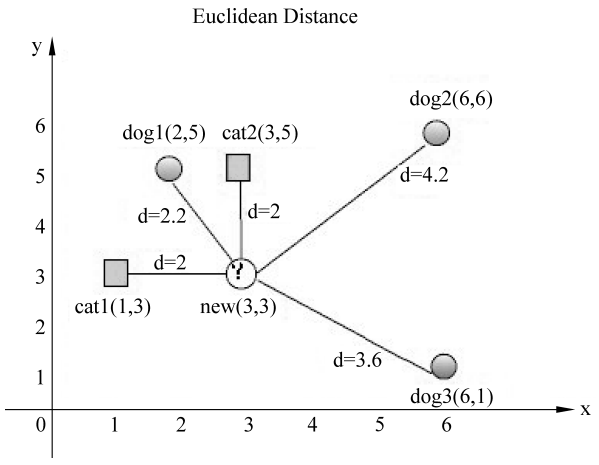


图 5.4　欧氏距离计算结果

扫一扫

视频讲解

5.2　初识 KNN——鸢尾花分类

本节使用 KNN 算法对 SKlearn 的鸢尾花数据集进行分类。

鸢尾花数据集：鸢尾花(iris)是单子叶百合目花卉，鸢尾花数据集最初由科学家 Anderson 测量收集而来。1936 年因用于公开发表的 Fisher 线性判别分析的示例，在机器学习领域广为人知。

数据集中的鸢尾花数据主要收集自加拿大加斯帕半岛，是一份经典数据集。鸢尾花数据集共收集了三类鸢尾花，即 Setosa 山鸢尾花、Versicolour 杂色鸢尾花和 Virginica 弗吉尼亚鸢尾花，每类鸢尾花有 50 条记录，共 150 条数据。数据集包括 4 个属性特征，分别是花萼长度、花萼宽度、花瓣长度和花瓣宽度。

在对鸢尾花数据集进行操作之前，先对数据进行详细观察。SKlearn 中的 iris 数据集有 5 个 key，分别如下。

(1) target_names：分类名称，包括 setosa、versicolor 和 virginica 类。

(2) data：特征数据值。

(3) target：分类(150 个)。

(4) DESCR：数据集的简介。

(5) feature_names：特征名称。

1. 查看数据

【例 5.1】 对鸢尾花 iris 数据集进行调用，查看数据的各方面特征。

```
from sklearn.datasets import load_iris
iris_dataset = load_iris()
#下面是查看数据的各项属性
```

```
print("数据集的 Keys:\n",iris_dataset.keys())        #查看数据集的 keys
print("特征名:\n",iris_dataset['feature_names'])     #查看数据集的特征名称
print("数据类型:\n",type(iris_dataset['data']))      #查看数据类型
print("数据维度:\n",iris_dataset['data'].shape)      #查看数据的结构
print("前五条数据:\n{}".format(iris_dataset['data'][:5]))  #查看前 5 条数据
#查看分类信息
print("标记名:\n",iris_dataset['target_names'])
print("标记类型:\n",type(iris_dataset['target']))
print("标记维度:\n",iris_dataset['target'].shape)
print("标记值:\n",iris_dataset['target'])            #查看数据集的简介
print('数据集简介:\n',iris_dataset['DESCR'][:20] + "\n.......")
                                                     #数据集简介前 20 个字符
```

运行结果如下。

```
数据集的Keys:
 dict_keys(['data', 'target', 'target_names', 'DESCR', 'feature_names'])
特征名:
 ['sepal length (cm)', 'sepal width (cm)', 'petal length (cm)', 'petal width (cm)']
数据类型:
 <class 'numpy.ndarray'>
数据维度:
 (150, 4)
前五条数据:
[[5.1 3.5 1.4 0.2]
 [4.9 3.  1.4 0.2]
 [4.7 3.2 1.3 0.2]
 [4.6 3.1 1.5 0.2]
 [5.  3.6 1.4 0.2]]
标记名:
 ['setosa' 'versicolor' 'virginica']
标记类型:
 <class 'numpy.ndarray'>
标记维度:
 (150,)
标记值:
 [0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
 0 0 0 0 0 0 0 0 0 0 0 0 0 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1
 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 2 2 2 2 2 2 2 2 2 2
 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2
 2 2]
数据集简介:
 Iris Plants Database
.......
```

2. 数据集拆分

对鸢尾花数据集进行训练集和测试集的拆分操作,可以使用 train_test_split()函数。train_test_split()函数属于 sklearn.model_selection 类中的交叉验证功能,能随机地将样本数据集合拆分成训练集和测试集。其格式为:

```
X_train,X_test, y_train, y_test =
cross_validation.train_test_split(train_data,train_target,test_size=0.4,
random_state=0)
```

【例 5.2】　对 iris 数据集进行拆分，并查看拆分结果。

```
from sklearn.datasets import load_iris
from sklearn.model_selection import train_test_split
iris_dataset = load_iris()
X_train, X_test, y_train, y_test = train_test_split( iris_dataset['data'], iris
_dataset['target'], random_state=2)
print("X_train",X_train)
print("y_train",y_train)
print("X_test",X_test)
print("y_test",y_test)
print("X_train shape: {}".format(X_train.shape))
print("X_test shape: {}".format(X_test.shape))
```

运行结果中，X_train 和 X_test 的维度分别如下。

```
X_train shape: (112, 4)
X_test shape: (38, 4)
```

3. 使用散点矩阵查看数据特征关系

在数据分析中，同时观察一组变量的多个散点图是很有意义的，这也被称为散点图矩阵。创建这样的图表工作量巨大，可以使用 scatter_matrix()函数。scatter_matrix()函数是 Pandas 提供的一个能从 DataFrame 创建散点图矩阵的函数。

函数格式：

```
scatter_matrix(frame, alpha=0.5, c,figsize=None, ax=None, diagonal='hist',
marker='.', density_kwds=None,hist_kwds=None, range_padding=0.05, **kwds)
```

主要参数如下。

frame：Pandas DataFrame 对象。

alpha：图像透明度，一般取(0,1)的小数。

figsize：以英寸为单位的图像大小，一般以元组(width，height)形式设置。

diagonal：必须且只能在{'hist'，'kde'}中选择一个，'hist'表示直方图(Histogram Plot)，'kde'表示核密度估计(Kernel Density Estimation)。该参数是 scatter_matrix()函数的关键参数。

marker：Matplotlib 可用的标记类型，如'.''，''o'等。

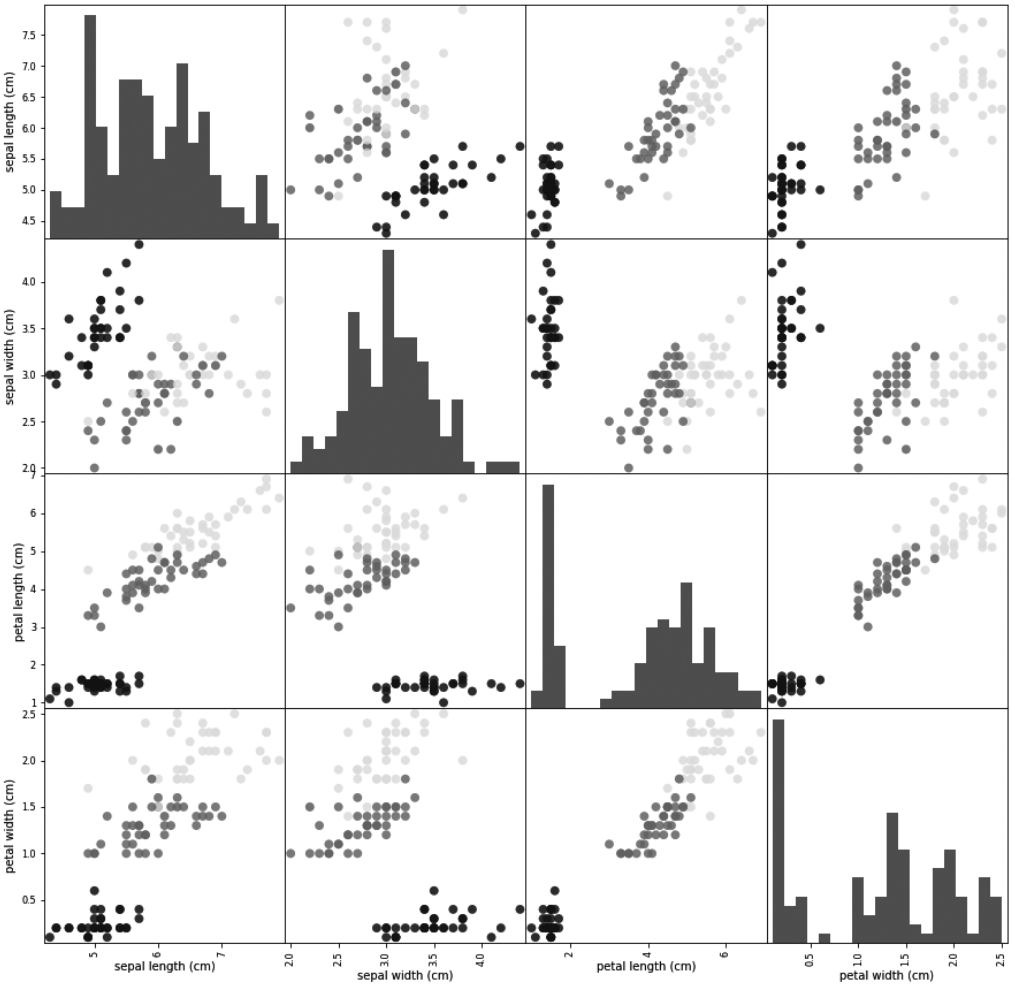
【例 5.3】　对例 5.2 的数据结果，使用 scatter_matrix()显示训练集与测试集。

可以在例 5.2 的基础上添加如下语句。

```
import pandas as pd
iris_dataframe = pd.DataFrame(X_train, columns=iris_dataset.feature_names)
#创建一个 scatter matrix,颜色值来自 y_train
pd.plotting.scatter_matrix(iris_dataframe, c=y_train, figsize=(15, 15),
marker='o', hist_kwds={'bins': 20}, s=60, alpha=.8)
```

运行结果如图 5.5 所示。可以看到,散点矩阵图呈对称结构,除对角上的密度函数图之外,其他子图分别显示了不同特征列之间的关联关系。例如,petal length 与 petal width 之间近似呈线性关系,说明这对特征关联性很强。相反,有些特征列之间的散布状态比较杂乱,基本无规律可循,说明特征间的关联性不强。

因此,在训练模型时,要优先选择关联明显的特征对进行学习。



扫一扫 看彩图

图 5.5 iris 数据集的特征散点矩阵图

4. 建立 KNN 模型

初步对数据集了解后,选取合适的模型并对模型进行初始化。然后对数据集进行分类学习,得到训练好的模型。

在 Python 中,实现 KNN 方法使用的是 KNeighborsClassifier 类,KNeighborsClassifier 类属于 Scikit-learn 的 neighbors 包。

KNeighborsClassifier 使用很简单,核心操作包括以下三步。

(1) 创建 KNeighborsClassifier 对象,并进行初始化。

基本格式:

```
sklearn.neighbors.KNeighborsClassifier(n_neighbors=5, weights='uniform',
algorithm='auto', leaf_size=30,p=2, metric='minkowski', metric_params=None,
n_jobs=None, **kwargs)
```

主要参数如下。

n_neighbors: int 型,可选,默认值是 5,代表 KNN 中的近邻数量 k 值。

weights: 计算距离时使用的权重,默认值是"uniform",表示平等权重。也可以取值"distance",则表示按照距离的远近设置不同权重。还可以自主设计加权方式,并以函数形式调用。

metric: 距离的计算,默认值是"minkowski"。当 p=2, metric='minkowski'时,使用的是欧氏距离。p=1,metric='minkowski'时为曼哈顿距离。

(2) 调用 fit()方法,对数据集进行训练。

函数格式:

```
fit(X, y)
```

说明: 以 X 为训练集,以 y 为测试集对模型进行训练。

(3) 调用 predict()函数,对测试集进行预测。

函数格式:

```
predict(X)
```

说明: 根据给定的数据预测其所属的类别标签。

【例 5.4】 使用 KNN 对鸢尾花 iris 数据集进行分类的完整代码实现。

扫一扫

视频讲解

```
from sklearn import datasets
from sklearn.neighbors import KNeighborsClassifier
from sklearn.model_selection import train_test_split
#导入鸢尾花数据并查看数据特征
iris = datasets.load_iris()
print('数据集结构:',iris.data.shape)
#获取属性
iris_X = iris.data
#获取类别
iris_y = iris.target
#划分成测试集和训练集
iris_train_X,iris_test_X,iris_train_y,iris_test_y=train_test_split(iris_X,
iris_y,test_size=0.2, random_state=0)
#分类器初始化
knn = KNeighborsClassifier()
#对训练集进行训练
knn.fit(iris_train_X, iris_train_y)
```

```
#对测试集数据的鸢尾花类型进行预测
predict_result = knn.predict(iris_test_X)
print('测试集大小:',iris_test_X.shape)
print('真实结果:',iris_test_y)
print('预测结果:',predict_result)
#显示预测精确率
print('预测精确率:',knn.score(iris_test_X, iris_test_y))
```

运行结果如下。

```
数据集结构： (150, 4)
测试集大小： (30, 4)
真实结果： [2 1 0 2 0 2 0 1 1 1 2 1 1 1 1 0 1 1 0 0 2 1 0 0 2 0 0 1 1 0]
预测结果： [2 1 0 2 0 2 0 1 1 1 2 1 1 1 2 0 1 1 0 0 2 1 0 0 2 0 0 1 1 0]
预测精确率： 0.9666666666666667
```

从结果可以看出,拆分的测试集中有 30 个样本,其中有一个判断错误,总体精确率约为 96.7%,精度较高。主要原因在于数据集中的数据比较好,数据辨识度较高。

也可以将 KNN 用于图像等分类场合,通过对目标图像进行归类,能够解决类似图像识别等问题。

5.3 KNN 手写数字识别

图像识别是模式识别研究中一个重要的领域,通过对图像进行分析和理解,识别出不同模式的目标对象。图像识别包括文字识别、图像识别与物体识别。文字识别是常见的图像识别问题,目的是分析并识别图片中包含的文字。

文字识别中难度较高的是手写文字识别,因为手写体与印刷体相比,个人风格迥异、图片大小不一。手写数字识别的目标相对简单,是从图像中识别出数字 0～9,经常用于自动邮件分拣等生产领域。在机器学习中,有时将识别问题转换为分类问题。

本实验使用的数据集修改自“手写数字光学识别数据集”[①],共保留了 1600 张图片。通过拆分,其中 1068 张作为训练集,其余的 532 张为测试集。图片为长宽都是 32px 的二值图,为方便处理,将图片预存为文本文件(过程省略,参考 3.6.3 节)。

扫一扫

视频讲解

【例 5.5】 使用 KNN 方法实现手写数字识别。

本例的素材文件夹为 HWdigits,子目录 trainSet 下存放训练数据,子目录 testSet 存放测试数据。数据为文本文件形式,每个文件表示一个手写数字。

在对文件系统进行操作时,可以使用模块 os 提供的 listdir()方法。listdir()方法返回指定文件夹下的文件/文件夹列表,格式为 os.listdir(path),字符型参数 path 指明目标路径。operator 模块中的 itemgetter()函数用于获取对象的某个维度的数据,参数为序号。

① 来源:http://archive.ics.uci.edu/ml/datasets,Alpaydin 与 Kaynak 提供,1998-07-01 发布。

```
#coding=utf-8
import numpy as np
from os import listdir

def loadDataSet():                                  #加载数据集
    #获取训练数据集
    print("1.Loading trainSet...")
    trainFileList = listdir('HWdigits/trainSet')
    trainNum = len(trainFileList)

    trainX = np.zeros((trainNum, 32 * 32))
    trainY = []
    for i in range(trainNum):
        trainFile = trainFileList[i]
        #将训练数据集向量化
        trainX[i, :] = img2vector('HWdigits/trainSet/%s' % trainFile,32,32)
        label = int(trainFile.split('_')[0]) #读取文件名的第一位作为标记
        trainY.append(label)
    #获取测试数据集
    print("2.Loading testSet...")
    testFileList = listdir('HWdigits/testSet')
    testNum = len(testFileList)
    testX = np.zeros((testNum, 32 * 32))
    testY = []
    for i in range(testNum):
        testFile = testFileList[i]
        #将测试数据集向量化
        testX[i, :] = img2vector('HWdigits/testSet/%s' % testFile,32,32)
        label = int(testFile.split('_')[0])  #读取文件名的第一位作为标记
        testY.append(label)
    return trainX, trainY, testX, testY
def img2vector(filename,h,w):                       #将 32 * 32 的文本转换为向量
    imgVector = np.zeros((1, h * w))
    fileIn = open(filename)
    for row in range(h):
        lineStr = fileIn.readline()
        for col in range(w):
            imgVector[0, row * 32 + col] = int(lineStr[col])
    return imgVector
def myKNN(testDigit, trainX, trainY, k):
    numSamples = trainX.shape[0]       #shape[0]代表行,每行一个图片,得到样本个数
    #1.计算欧氏距离
    diff=[]
    for n in range(numSamples):
        diff.append(testDigit-trainX[n])     #每个个体差
    diff=np.array(diff)                      #转变为 ndarray
    #对差求平方和,然后取和的平方根
    squaredDiff = diff ** 2
    squaredDist = np.sum(squaredDiff, axis = 1)
```