**实战四：手把手教你实现数字识别**

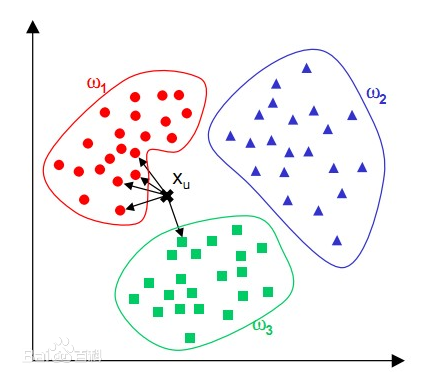
**一、KNN实现数字识别**

**1.原理**

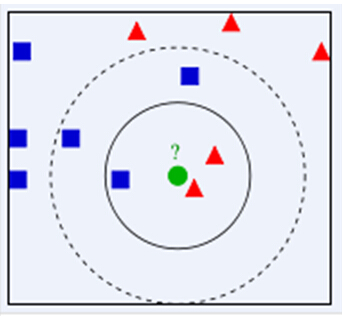
最简单最初级的分类器是将全部的训练数据所对应的类别都记录下来，当测试对象的属性和某个训练对象的属性完全分配时，便可以对其进行分类。但是怎么可能所有测试对象都会找到与之完全匹配的训练对象呢，其次就是存在一个测试对象同时与多个训练对象匹配，导致一个训练对象被分到了多个类的问题，基于这些问题呢，就产生了KNN。

邻近算法，或者说K最近邻（KNN，k-NearestNeighbor）分类算法是数据挖掘分类技术中最简单的方法之一。所谓K最近邻，就是K个最近的邻居的意思，说的是每个样本都可以用它最接近的k个邻居来代表。

KNN是通过测量不同特征值之间的距离进行分类。它的思路是：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，其中K通常是不大于20的整数.KNN算法中，所选择的邻居都是已经正确分类的对象。该方法在定类决策中只依据最邻近的一个或者几个样本的类别来决定待分样本所属的类别。

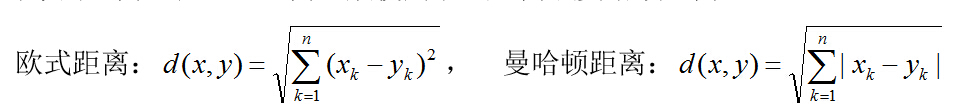


下面通过一个简单的例子来说明一下：如下图，绿色圆要被决定赋予哪个类，是红色三角形还是蓝色正方形？如果K=3，由于红色三角形所占比例为2/3，绿色圆将被赋予红色三角形那个类，如果K=5，由于蓝色正方形比例为3/5,因此绿色圆被赋予蓝色四方形类。



由此也说明了KNN算法的结果很大程度取决于K的选择。

在KNN中，通过计算对象间距离来作为各个对象之间的非相似指标，避免了对象之间的匹配问题，在这里距离一般使用欧式距离或曼哈顿距离：



同时，KNN通过依据k个对象中占优的类别进行决策，而不是单一的对象类别决策。这两点就是KNN算法的优势。

接下来对KNN算法的思想总结一下：就是在训练集中数据和标签已知的情况下，输入测试数据，将测试数据的特征与训练集中对应的特征进行相互比较，找到训练集中与之最为相似的前K个数据，则该测试数据对应的类别就是K个数据中出现次数最多的那个分类，其算法的描述为：

（1）计算测试数据与各个训练数据之间的距离；

（2）按照距离的递增关系进行排序；

（3）选取距离最小的K个点；

（4）确定前K个点所在类别的出现频率；

（5）返回前K个点中出现频率最高的类别作为测试数据的预测分类。

**2.KNN实现数字识别**

**（1）概要**

加载mnist数据集，mnist数据集分为四组数据，训练图片，训练标签，测试图片，测试标签。

而这里我们实现的功能：从训练图片中随机抽取一定数量的训练图片，并从测试图片中也随机抽取一定数量的测试图片，然后这些测试与训练图片进行KNN计算，并从这些一定数量的训练图片中找出K张与当前测试图片KNN距离最近的照片，然后解析训练图片中的内容，从而预测出数字，并检测结果是否正确。

**（2）实现整体步骤**

a.加载mnist数据集

b.选取一定数量的测试图片与训练图片

c.测试图片与训练图片进行KNN计算，测试图片分别于训练图片做差，并找出K张训练图片与当前测试图片最近的

d.解析当前K张训练图片，得到其训练图片标签

e.将图片标签转化成具体的数字，得到预测数据

f.获取测试图片的标签是数据并与上述预测数据对比，得出成功率。

**（3）代码实现**

# KNN 最近临域

# CNN 卷积神经网络

# 旧瓶装新酒：数字识别的不同

# 样本地址：http://yann.lecun.com/exdb/mnist/

# 代码下载

# from tensorflow.examples.tutorials.mnist import input\_data #第一次下载数据时用

# data = input\_data.read\_data\_sets('MNIST\_data/')

# Knn test 样本 K个 max

# 数据都是以随机数加载，四组数据，训练图片，训练标签，测试图片，测试标签。

# 1 load data

# 2 knn test train distance

# 3 knn k个最近图片 5张测试 500张训练图片做差 -->500张找出4张与当前最近的照片

# 4 解析图片中的内容parse centent==>label

# 5 label转化成具体的数字

# 6 检测结果是否正确

import tensorflow as tf

import numpy as np

import random

from tensorflow.examples.tutorials.mnist import input\_data

# load data 数据装载

mnist = input\_data.read\_data\_sets('MNIST\_data/',one\_hot=True)

# 属性设置

trainNum = 55000

testNum = 10000

trainSize = 500

testSize = 5

k = 4

# 随机选取一定数量的测试图片与训练图片 在0-trainNum之间随机选取trainSize个数字，不可重复

trainIndex = np.random.choice(trainNum,trainSize,replace=False)

testIndex = np.random.choice(testNum,testSize,replace=False)

# 获取训练图片

trainData = mnist.train.images[trainIndex]

# 获取训练标签

trainLabel = mnist.train.labels[trainIndex]

# 获取测试图片

testData = mnist.test.images[testIndex]

# 获取测试标签

testLabel = mnist.test.labels[testIndex]

# 打印数据，获取数据的维度信息

print('trainData.shape',trainData.shape) #(500,784) 28\*28=784图片所有像素点

print('trainLabel.shape',trainLabel.shape) #(500,10) 500行表示500个数，10列用来表示第几个数

print('testData.shape',testData.shape) #(5,784)

print('testLabel.shape',testLabel.shape) #(5,10)

print('testLabel',testLabel)

# TensorFlow input输入的定义

trainDataInput = tf.placeholder(shape=[None,784],dtype=tf.float32)

trainLabelInput = tf.placeholder(shape=[None,10],dtype=tf.float32)

testDataInput = tf.placeholder(shape=[None,784],dtype=tf.float32)

testLabelInput = tf.placeholder(shape=[None,10],dtype=tf.float32)

# KNN 距离 distance

# f1测试数据输入维度扩展 由5\*784==>5\*1\*784

f1 = tf.expand\_dims(testDataInput,1)

# f2测试图片与训练图片两者之差

f2 = tf.subtract(trainDataInput,f1)

# f3完成数据累加 784个像素点的差值

f3 = tf.reduce\_sum(tf.abs(f2),reduction\_indices=2)

# f4对f3取反

f4 = tf.negative(f3)

# f5,f6 选取f4中最大的4个数，即选取f3中最小的4个数（值的内容及值的下标）

f5,f6 = tf.nn.top\_k(f4,k=4)

# f6 存储最近的四张图片的下标

f7 = tf.gather(trainLabelInput,f6)

# f8 将最近的四张图片 累加 维度为1

f8 = tf.reduce\_sum(f7,reduction\_indices=1)

# f9 根据训练数据 推测的值

f9 = tf.argmax(f8,dimension=1)

with tf.Session() as sess:

p1 = sess.run(f1,feed\_dict={testDataInput:testData[0:5]})

print('p1=',p1.shape) #(5,1,784)

p2 = sess.run(f2,feed\_dict={trainDataInput:trainData,testDataInput:testData[0:5]})

print('p2=',p2.shape) #(5,500,784)

p3 =sess.run(f3,feed\_dict={trainDataInput:trainData,testDataInput:testData[0:5]})

print('p3=',p3.shape) #(5,500)

print('p3[0,0]',p3[0,0])

p4 = sess.run(f4,feed\_dict={trainDataInput:trainData,testDataInput:testData[0:5]})

print('p4[0,0]',p4[0,0])

p5,p6 = sess.run((f5,f6),feed\_dict={trainDataInput:trainData,testDataInput:testData[0:5]})

print('p5=',p5.shape)

print('p6=',p6.shape)

#p5=(5,4) 每一张测试图片（5张）分别对应当前四张最近的图片

#p6=(5,4) 每一张测试图片（5张）分别对应当前四张最近的图片下标

print('p5[0,0]=',p5[0])

print('p6[0,0]=',p6[0])

p7 = sess.run(f7,feed\_dict={trainDataInput:trainData,testDataInput:testData[0:5],trainLabelInput:trainLabel})

print('p7=',p7.shape) #(5,4,10)

print('p7[]=',p7)

p8 = sess.run(f8,feed\_dict={trainDataInput:trainData,testDataInput:testData[0:5],trainLabelInput:trainLabel})

print('p8=',p8.shape) #(5,10)

print('p8[]=',p8)

p9 = sess.run(f9,feed\_dict={trainDataInput:trainData,testDataInput:testData[0:5],trainLabelInput:trainLabel})

print('p9=',p9.shape)

print('p9[]=',p9)

# p10 根据测试数据 得到的值

p10 = np.argmax(testLabel[0:5],axis=1)

print('p10[]=',p10)

# 比较p9和p10,计算准确率

j = 0

for i in range(0,5):

if p10[i] == p9[i]:

j=j+1

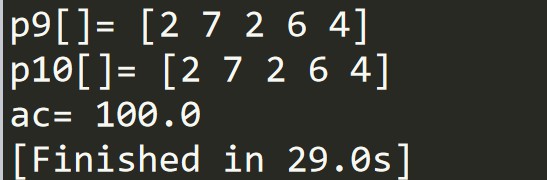
print('ac=',j\*100/5)

**（4）结果展示**

p9是预测结果

p10是真实结果

ac是准确率



**二、CNN实现数字识别**

<https://blog.csdn.net/sinat_35821976/article/details/81503953>

CNN这里掌握的并不是很好，具体有关CNN的基础知识可参考上述博客地址，

这里一并给出CNN实现代码。

# CNN 卷积神经网络

import tensorflow as tf

import numpy as np

from tensorflow.examples.tutorials.mnist import input\_data

# load data

mnist = input\_data.read\_data\_sets('MNIST\_data/',one\_hot=True)

# 设置input

imageInput = tf.placeholder(tf.float32,[None,784]) #输入图片28\*28=784维

labelInput = tf.placeholder(tf.float32,[None,10]) #标签 参考CNN内容

# 数据维度的调整 data reshape 由[None,784]-->M\*28\*28\*1 即2维转换成4维

imageInputReshape = tf.reshape(imageInput,[-1,28,28,1]) #完成维度的调整

# 卷积运算 w0是权重矩阵 本质是卷积内核 5\*5 output 32 一维 1 stddev 方差

w0 = tf.Variable(tf.truncated\_normal([5,5,1,32],stddev=0.1))

b0 = tf.Variable(tf.constant(0.1,shape=[32]))

#layer1:激励函数+卷积运算

# imageInputReshape: M\*28\*28\*1 w0:5,5,1,32 strides:步长 padding:卷积核可以停留到图像的边缘

layer1 = tf.nn.relu(tf.nn.conv2d(imageInputReshape,w0,strides=[1,1,1,1],padding='SAME')+b0)

# M\*28\*28\*32 ===>M\*7\*7\*32

# 池化层 下采样 数据量减少了很多

layer1\_pool = tf.nn.max\_pool(layer1,ksize=[1,4,4,1],strides=[1,4,4,1],padding='SAME')

#layer2(output输出层):softmax（激励函数+乘加运算）

w1 = tf.Variable(tf.truncated\_normal([7\*7\*32,1024],stddev=0.1))

b1 = tf.Variable(tf.constant(0.1,shape=[1024]))

h\_reshape = tf.reshape(layer1\_pool,[-1,7\*7\*32]) #M\*7\*7\*32->N\*N1

# [N\*7\*7\*32] [7\*7\*32,1024]=N\*1024

h1 = tf.nn.relu(tf.matmul(h\_reshape,w1)+b1)

# softmax

w2 = tf.Variable(tf.truncated\_normal([1024,10],stddev=0.1))

b2 = tf.Variable(tf.constant(0.1,shape=[10]))

pred = tf.nn.softmax(tf.matmul(h1,w2)+b2) # N\*1024 1024\*10=N\*10

# N\*10(概率)

loss0 = labelInput\*tf.log(pred)

loss1 = 0

for m in range(0,100):

for n in range(0,10):

loss1 = loss1 + loss0[m,n]

loss = loss1/100

# train 训练

train = tf.train.GradientDescentOptimizer(0.01).minimize(loss)

# run 运行

with tf.Session() as sess:

sess.run(tf.global\_variables\_initializer())

for i in range(100):

images,labels = mnist.train.next\_batch(500)

sess.run(train,feed\_dict={imageInput:images,labelInput:labels})

pred\_test = sess.run(pred,feed\_dict={imageInput:mnist.test.images,labelInput:labels})

acc = tf.equal(tf.arg\_max(pred\_test,1),tf.arg\_max(mnist.test.labels,1))

acc\_float = tf.reduce\_mean(tf.cast(acc,tf.float32))

acc\_result = sess.run(acc\_float,feed\_dict={imageInput:mnist.test.images,labelInput:mnist.test.labels})

print(acc\_result)