K-Nearest Neighbor(KNN) 最邻近分类算法及 Python 实现方式

K-Nearest Neighbor 最邻近分类算法:

简称 KNN，最简单的机器学习算法之一，核心思想俗称“随大流”。是一种分类算法，基于实例的学习（instance-based learning）和懒惰学习（lazy learning）。
懒惰学习：指的是在训练是仅仅是保存样本集的信息，直到测试样本到达是才进行分类决策。

核心想法:

在距离空间里，如果一个样本的最接近的 k 个邻居里，绝大多数属于某个类别，则该样本也属于这个类别。

范例:

假设，我们有这样一组电影数据:

电影名称	打斗次数	接吻次数	电影类型
California Man	3	104	Romance
He's Not Really into Dudes	2	100	Romance
Beautiful Woman	1	81	Romance
Kevin Longblade	101	10	Action
Robo Slayer 3000	99	5	Action
Amped II	98	2	Action
未知	18	90	Unknown

由数据可以看出，我们有上述 6 部电影的数据及分类，最后一部“未知”的是需要预测处于哪个分类中。
然后，我们将数据中的“打斗次数”属性标记为 X，“接吻次数”标记为 Y，这样上述数据都能化为坐标轴中的一点：

点	X坐标	Y坐标	点类型
A点	3	104	Romance
B点	2	100	Romance
C点	1	81	Romance
D点	101	10	Action
E点	99	5	Action
F点	98	2	Action
G点	18	90	Unknown

之后便是将所有点与“未知”的点 G 进行距离计算，因为这个例子是二维的，因此这里我们使用 E(x,y)=sqr((x2-x1)^2+(y2-y1)^2)，如果是多维的话，可以使用：

$$E(x,y)=\sqrt{\sum_{i=0}^{n}(x_i-y_i)^2}$$

最后可得到结果，这里我省略到 int：
a：20
b：18
c：19
d：115
e：117
f：118
因此可以看出，最近的三个点是 ABC 三点，而 ABC 三点都是 Romance 类型。

选择方式:

根据上述例子，如果 ABC 中三个电影分类有一个不是 Romance 怎么办。这里我们遵循少数服从多数的投票法则（majority-voting），让未知实例归类为最邻近样本中最多数的类别。

其他距离衡量方式:

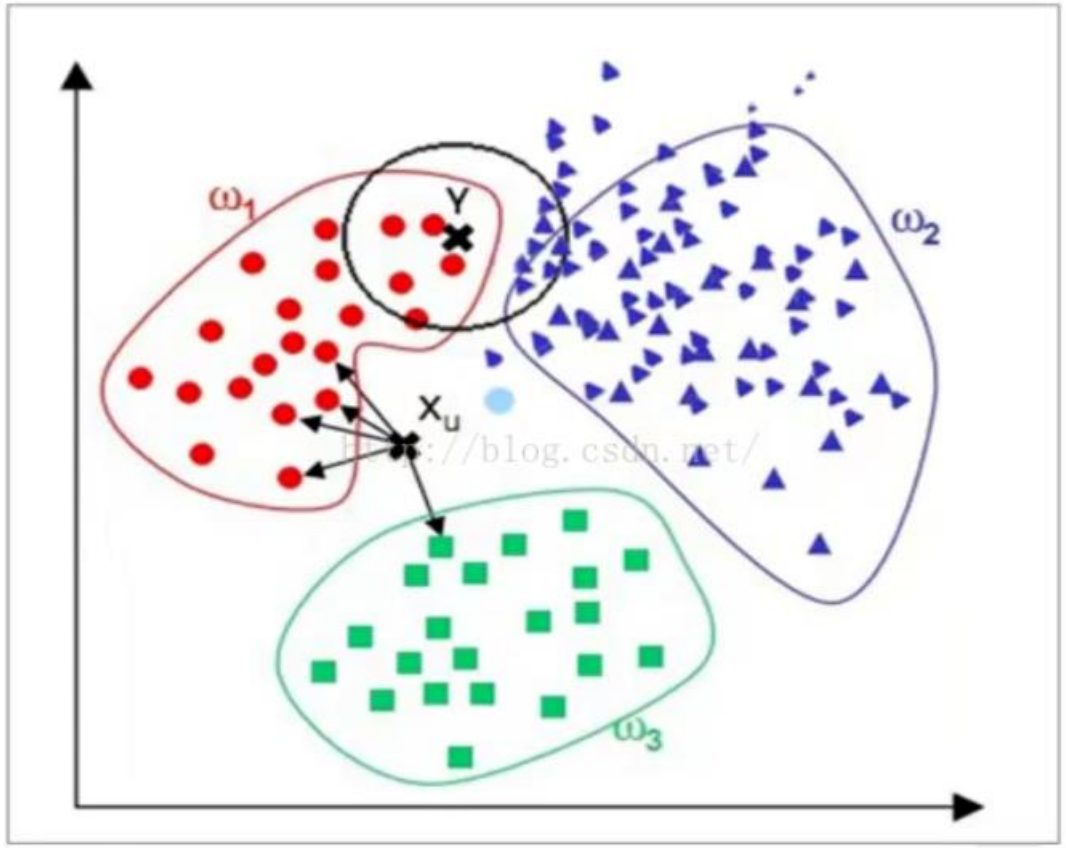
亦可使用余弦值（cos），相关度（correlation），曼哈顿距离等。

优点:

简单，易于实现，易于理解，通过对 K 的选择能一定程度上的具备丢噪音数据的健壮性（增大 K 值）

缺点:

需要大量的空间存储已知实例，算法复杂度高（需要比较所有已知实例）。当样本分布不平均时，比如其中一个样本实例过多，容易被归纳为实例多的样本，如下图 Y 点:



解决方法:

给距离增加权重，越近的距离权重越高，能一定程度的避免上述样本分布不平均的问题。

Python 实现方式:

```
import numpy as np
from sklearn import neighbors

knn = neighbors.KNeighborsClassifier() #取得 knn 分类器
data = np.array([[3,104],[2,100],[1,81],[101,10],[99,5],[98,2]]) #data 对应着打斗次数和接吻次数
labels = np.array([1,1,1,2,2,2]) #labels 则是对应 Romance 和 Action
knn.fit(data,labels) #导入数据进行训练
print(knn.predict([18,90]))
```

需要加载 numpy，sklearn 包，这两个都是机器学习或数据挖掘常用的包。

kNN 算法：K 最近邻(kNN，k-NearestNeighbor)分类算法

一、KNN 算法概述#

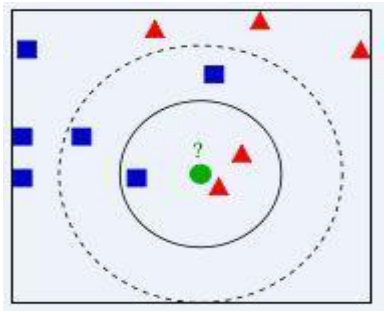
邻近算法，或者说 K 最近邻(kNN，k-NearestNeighbor)分类算法是数据挖掘分类技术中最简单的方法之一。所谓 K 最近邻，就是 k 个最近的邻居的意思，说的是每个样本都可以用它最接近的 k 个邻居来代表。Cover 和 Hart 在 1968 年提出了最初的邻近算法。KNN 是一种分类(classification)算法，它输入基于实例的学习（instance-based learning），属于懒惰学习（lazy learning）即 KNN 没有显式的学习过程，也就是说没有训练阶段，数据集事先已有了分类和特征值，待收到新样本后直接进行处理。与急切学习（eager learning）相对应。

KNN 是通过测量不同特征值之间的距离进行分类。

思路是：如果一个样本在特征空间中的 k 个最邻近的样本中的大多数属于某一个类别，则该样本也划分为这个类别。KNN 算法中，所选择的邻居都是已经正确分类的对象。该方法在定类决策上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。

提到 KNN，网上最常见的就是下面这个图，可以帮助大家理解。

我们要确定绿点属于哪个颜色（红色或者蓝色），要做的就是选出距离目标点距离最近的 k 个点，看这 k 个点的大多数颜色是什么颜色。当 k 取 3 的时候，我们可以看出距离最近的三个，分别是红色、红色、蓝色，因此得到目标点为红色。



算法的描述：#

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

二、关于 K 的取值#

K：临近数，即在预测目标点时取几个临近的点来预测。
K 值得选取非常重要，因为：
如果当 K 的取值过小时，一旦有噪声得成分存在们将会对预测产生比较大影响，例如取 K 值为 1 时，一旦最近的一个点是噪声，那么就会出现偏差，K 值的减小就意味着整体模型变得复杂，容易发生过拟合；
如果 K 的值取的过大时，就相当于用较大邻域中的训练实例进行预测，学习的近似误差会增大。这时与输入目标点较远实例也会对预测起作用，使预测发生错误。K 值的增大就意味着整体的模型变得简单；
如果 K==N 的时候，那么就是取全部的实例，即为取实例中某分类下最多的点，就对预测没有什么实际的意义了；
K 的取值尽量要取奇数，以保证在计算结果最后会产生一个较多的类别，如果取偶数可能会产生相等的情况，不利于预测。

K 的取法：#

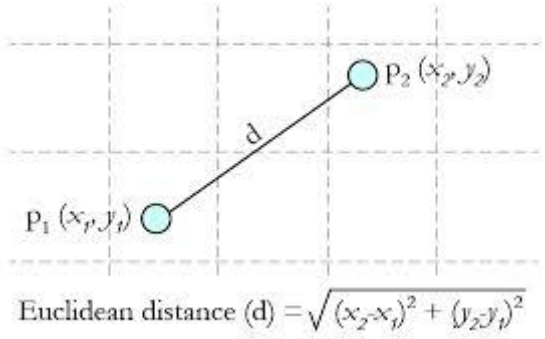
常用的方法是从 k=1 开始，使用检验集估计分类器的误差率。重复该过程，每次 K 增值 1，允许增加一个近邻。选取产生最小误差率的 K。
一般 k 的取值不超过 20，上限是 n 的开方，随着数据集的增大，K 的值也要增大。

三、关于距离的选取#

距离就是平面上两个点的直线距离
关于距离的度量方法，常用的有：欧几里得距离、余弦值（cos），相关度 （correlation），曼哈顿距离 （Manhattan distance）或其他。

Euclidean Distance 定义：#

两个点或元组 P1=（x1，y1）和 P2=（x2，y2）的欧几里得距离是



距离公式为：（多个维度的时候是多个维度各自求差）

$$E(x,y) = \sqrt{\sum_{i=0}^{n}(x_i - y_i)^2}$$

四、总结#

KNN 算法是最简单有效的分类算法，简单且容易实现。当训练数据集很大时，需要大量的存储空间，而且需要计算待测样本和训练数据集中所有样本的距离，所以非常耗时

KNN 对于随机分布的数据集分类效果较差，对于类内间距小，类间间距大的数据集分类效果好，而且对于边界不规则的数据效果好于线性分类器。

KNN 对于样本不均衡的数据效果不好，需要进行改进。改进的方法时对 k 个近邻数据赋予权重，比如距离测试样本越近，权重越大。

KNN 很耗时，时间复杂度为 O(n)，一般适用于样本数较少的数据集，当数据量大时，可以将数据以树的形式呈现，能提高速度，常用的有 kd-tree 和 ball-tree。

（弱小无助。。。根据许多大佬的总结整理的）

五、Python 实现#

根据算法的步骤，进行 kNN 的实现,完整代码如下

```
#!/usr/bin/env python
# -*- coding:utf-8 -*-
# Author: JYRoooy
import csv
import random
import math
import operator
```

```
8
9 # 加载数据集
10 def loadDataset(filename, split, trainingSet = [], testSet = []):
11     with open(filename, 'r') as csvfile:
12         lines = csv.reader(csvfile)
13         dataset = list(lines)
14         for x in range(len(dataset)-1):
15             for y in range(4):
16                 dataset[x][y] = float(dataset[x][y])
17             if random.random() < split:  #将数据集随机划分
18                 trainingSet.append(dataset[x])
19             else:
20                 testSet.append(dataset[x])
21
22 # 计算点之间的距离，多维度的
23 def euclideanDistance(instance1, instance2, length):
24     distance = 0
25     for x in range(length):
26         distance += pow((instance1[x]-instance2[x]), 2)
27     return math.sqrt(distance)
28
29 # 获取 k 个邻居
30 def getNeighbors(trainingSet, testInstance, k):
31     distances = []
32     length = len(testInstance)-1
33     for x in range(len(trainingSet)):
34         dist = euclideanDistance(testInstance, trainingSet[x], length)
35         distances.append((trainingSet[x], dist))   #获取到测试点到其他点的距离
36     distances.sort(key=operator.itemgetter(1))    #对所有的距离进行排序
37     neighbors = []
38     for x in range(k):   #获取到距离最近的 k 个点
39         neighbors.append(distances[x][0])
40         return neighbors
41
42 # 得到这 k 个邻居的分类中最多的那一类
```

```
43 def getResponse(neighbors):
44     classVotes = {}
45     for x in range(len(neighbors)):
46         response = neighbors[x][-1]
47         if response in classVotes:
48             classVotes[response] += 1
49         else:
50             classVotes[response] = 1
51     sortedVotes = sorted(classVotes.items(), key=operator.itemgetter(1), reverse=True)
52     return sortedVotes[0][0]   #获取到票数最多的类别
53 
54 #计算预测的准确率
55 def getAccuracy(testSet, predictions):
56     correct = 0
57     for x in range(len(testSet)):
58         if testSet[x][-1] == predictions[x]:
59             correct += 1
60     return (correct/float(len(testSet)))*100.0
61 
62 
63 def main():
64     #prepare data
65     trainingSet = []
66     testSet = []
67     split = 0.67
68     loadDataset(r'irisdata.txt', split, trainingSet, testSet)
69     print('Trainset: ' + repr(len(trainingSet)))
70     print('Testset: ' + repr(len(testSet)))
71     #generate predictions
72     predictions = []
73     k = 3
74     for x in range(len(testSet)):
75         # trainingsettrainingSet[x]
76         neighbors = getNeighbors(trainingSet, testSet[x], k)
77         result = getResponse(neighbors)
```

```
        predictions.append(result)
        print ('predicted=' + repr(result) + ', actual=' + repr(testSet[x][-1]))
    print('predictions: ' + repr(predictions))
    accuracy = getAccuracy(testSet, predictions)
    print('Accuracy: ' + repr(accuracy) + '%')

if __name__ == '__main__':
    main()
```

六、sklearn 库的应用#

我利用了 sklearn 库来进行了 kNN 的应用（这个库是真的很方便了，可以借助这个库好好学习一下，我是用 KNN 算法进行了根据成绩来预测，这里用一个花瓣萼片的实例，因为这篇主要是关于 KNN 的知识，所以不对 sklearn 的过多的分析，而且我用的还不深入😅）

sklearn 库内的算法与自己手搓的相比功能更强大、拓展性更优异、易用性也更强。还是很受欢迎的。（确实好用，简单）

```
from sklearn import neighbors    //包含有 kNN 算法的模块
from sklearn import datasets     //一些数据集的模块
```

调用 KNN 的分类器

```
knn = neighbors.KNeighborsClassifier()
```

预测花瓣代码

```
from sklearn import neighbors
from sklearn import datasets
knn = neighbors.KNeighborsClassifier()
iris = datasets.load_iris()
# f = open("iris.data.csv", 'wb')              #可以保存数据
# f.write(str(iris))
# f.close()
print iris
knn.fit(iris.data, iris.target)                 #用 KNN 的分类器进行建模，这里利用的默认的参数，大家可以自行查阅文档
predictedLabel = knn.predict([[0.1, 0.2, 0.3, 0.4]])
print ("predictedLabel is :" + predictedLabel)
```

上面的例子是只预测了一个，也可以进行数据集的拆分，将数据集划分为训练集和测试集

```
from sklearn.mode_selection import train_test_split    #引入数据集拆分的模块
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.3, random_state=42)
```

关于 train_test_split 函数参数的说明：

train_data：被划分的样本特征集

train_target：被划分的样本标签

test_size：float-获得多大比重的测试样本 （默认：0.25）
int - 获得多少个测试样本
random_state：是随机数的种子。

写在后面#

本人的在学习机器学习和深度学习算法中的源码 github 地址 https://github.com/JYRoy/MachineLearning
还是在校大学生，知识面不全，也参考了网上许多大佬的博客，一些个人的理解与应用可能有问题，欢迎大家指正，有学到新的相关知识也会对文章进行更新。

作者：JYRoy（感谢这位作者）

分类 ：kNN（k nearest neighbour）最近邻算法（Python）

kNN 算法概述

kNN 算法是比较好理解，也比较容易编写的分类算法。

简单地说，kNN 算法采用测量不同特征值之间的距离方法进行分类。

我们可以假设在一个 N 维空间中有很多个点，然后这些点被分为几个类。相同类的点，肯定是聚集在一起的，它们之间的距离相比于和其他类的点来说，非常近。如果现在有个新的点，我们不知道它的类别，但我们知道了它的坐标，那只要计算它和已存在的所有点的距离，然后以最近的 k 个点的多数类作为它的类别，则完成了它的分类。这个 k 就是 kNN 中的 k 值。

举个例子：我们知道地球是有经纬度的，中国人肯定绝大多数都集中在中国的土地上，美国人也一样多数都集中在自己的土地上。如果现在给我们某个人的坐标，让我们给它分类，判断他是哪国人。我们计算了他和世界上每个人的距离，然后取离他最近的 k 个人中最多国别的国别作为他的国别。这样我们就完成了他的国别分类。（当然也有可能一个外国人正好来中国游玩，我们错误的将他分类为中国人了，这个只是举例，不要在意这些细节啦 ^_^）

（图片略）

所以 kNN 算法无非就是计算一个未知点与所有已经点的距离，然后根据最近的 k 个点类别来判断它的类别。简单，粗暴，实用。

kNN 算法的重点

既然我们已经了解 kNN 算法了，那我们应该也大概了解到这个算法的重点是什么了

（1）怎么度量邻近度

我们首先想到的肯定是点和点之间距离。但除了距离，其实我们也可以考虑两个点之间的相似度，越相似，就代表两个点距离越近。同理，我们也可以考虑相异度，越相异，就代表两个点距离越远。其实距离的度量就是相异性度量的其中一种。

（2）k 值怎么取

k 值的选取关乎整个分类器的性能。如果 k 值取得过小，容易受噪点的影响而导致分类错误。而 k 值取得过大，又容易分类不清，混淆了其他类别的点。

（3）数据的预处理

拿到数据，我们不能直接就开始套用算法，而是需要先规范数据。例如我们想通过一个人的年龄和工资来进行分类，很明显工资的数值远大于年龄，如果我们不对它进行一个统一的规范，必然工资这个特征会左右我们的分类，而让年龄这个特征无效化，这不是我们想看到的。

邻近度的度量

临近度的度量，主要考虑相似性和相异性的度量。

一般的，我们把相似度定义为 s，常常在 0（不相似）和 1（完全相似）之间取值。而相异度 d 有时在 0（不相异）和 1（完全相异）之间取值，有时也在 0 和∞之间取值。
当相似度（相异度）落在区间[0,1]之间时，我们可以定义 d = 1 - s（或 s = 1 - d）。另一种简单的方法是定义相似度为负的相异度（或相反）。
通常，具有若干属性的对象之间的邻近度用单个属性的邻近度的组合来定义，下图是单个属性的对象之间的邻近度。

简单属性的相似度和相异度

属性类型	相异度	相似度
标称的	$d = \begin{cases} 0, & \text{如果} x = y \\ 1, & \text{如果} x \neq y \end{cases}$	$s = \begin{cases} 1, & \text{如果} x = y \\ 0, & \text{如果} x \neq y \end{cases}$
序数的	$d = \lvert x - y \rvert / (n-1)$ （值映射到整数0到n-1，其中n是值的个数）	$s = 1 - d$
区间或比率的	$d = \lvert x - y \rvert$	$s = 1 - d$，$s = \frac{1}{1+d}$，$s = e^{-d}$， $s = 1 - \frac{d - \min\_d}{\max\_d - \min\_d}$

下面我们讨论更复杂的涉及多个属性的对象之间的邻近性度量

1、距离

一维、二维、三维或高纬空间中两个点 **x** 和 **y** 之间的**欧几里得距离（Euclidean distance）**dd 由如下公式定义：

$$d(x,y)=\sum_{k=1}^{n}(x_k-y_k)^2\text{----------}\sqrt{d(x,y)=\sum k=1n(xk-yk)2}$$

其中，nn 是维数，而 xkxk 和 ykyk 分别是 **x** 和 **y** 的第 kk 个属性值（分量）。
欧几里得距离是最常用的距离公式。距离对特征都是区间或比率的对象非常有效。

2、二元数据的相似性度量

两个仅包含二元属性的对象之间的相似性度量也称为**相似系数（similarity coefficient）**，并且通常在 0 和 1 直接取值，值为 1 表明两个对象完全相似，而值为 0 表明对象一点也不相似。

设 **x** 和 **y** 是两个对象，都由 n 个二元属性组成。这样的两个对象（即两个二元向量）的比较可生成如下四个量（频率）：

f_{00}=x 取 0 并且 y 取 0 的属性个数 f00=x 取 0 并且 y 取 0 的属性个数

f_{01}=x 取 0 并且 y 取 1 的属性个数 f01=x 取 0 并且 y 取 1 的属性个数

f_{10}=x 取 1 并且 y 取 0 的属性个数 f10=x 取 1 并且 y 取 0 的属性个数

f_{11}=x 取 1 并且 y 取 1 的属性个数 f11=x 取 1 并且 y 取 1 的属性个数

简单匹配系数（Simple Matching Coefficient,SMC）一种常用的相似性系数是**简单匹配系数**，定义如下：

$$SMC=f_{11}+f_{00}f_{01}+f_{10}+f_{11}+f_{00}SMC=f11+f00f01+f10+f11+f00$$

该度量对出现和不出现都进行计数。因此，**SMC** 可以在一个仅包含是非题的测验中用来发现回答问题相似的学生。

Jaccard 系数（Jaccard Coefficient）假定 **x** 和 **y** 是两个数据对象，代表一个事务矩阵的两行（两个事务）。如果每个非对称的二元属性对应于商店的一种商品，则 **1** 表示该商品被购买，而 **0** 表示该商品未被购买。由于未被顾客购买的商品数远大于被其购买的商品数，因而像 **SMC** 这样的相似性度量将会判定所有的事务都是类似的。这样，常常使用 **Jaccard 系数**来处理仅包含非对称的二元属性的对象。Jaccard 系数通常用符号 **J** 表示，由如下等式定义：

$$J=匹配的个数不设计\ 0-0\ 匹配的属性个数=f_{11}f_{01}+f_{10}+f_{11}J=匹配的个数不设计\ 0-0\ 匹配的属性个数=f11f01+f10+f11$$

3、余弦相似度

文档的相似性度量不仅应当像 Jaccard 度量一样需要忽略 0-0 匹配，而且还必须能够处理非二元向量。下面定义的**余弦相似度（cosine similarity）**就是文档相似性最常用的度量之一。如果 **x** 和 **y** 是两个文档向量，则

$$cos(x,y)=x\cdot y||x||||y||cos⁡(x,y)=x\cdot y||x||||y||$$

其中，“·”表示向量点积，$x\cdot y=\sum_{k=1}^{n}x_k y_k x\cdot y=\Sigma k=1nxkyk$，$||x||=||x||$是向量 **x** 的长度，$||x||=\sqrt{\sum_{k=1}^{n}x_k^2}=\sqrt{x\cdot x}=||x||=\Sigma k=1nxk2=x\cdot x$

余弦相似度公式还可以写为：

$$cos(x,y)=x||x||\cdot y||y||=x'\cdot y'cos⁡(x,y)=x||x||\cdot y||y||=x'\ \cdot y'$$

x 和 **y** 被它们的长度除，将它们规范化成具有长度 1。这意味在计算相似度时，余弦相似度不考虑两个数据对象的量值。（当量值是重要的时，欧几里得距离可能是一种更好的选择）

余弦相似度为 1，则 **x** 和 **y** 之间夹角为 0 度，**x** 和 **y** 是相同的；如果余弦相似度为 0，则 **x** 和 **y** 之间的夹角为 90 度，并且它们不包含任何相同的词。

4、广义 Jaccard 系数

广义 Jaccard 系数可以用于文档数据，并在二元属性情况下归约为 Jaccard 系数。该系数用 EJ 表示：

$$EJ(x,y)=x\cdot y||x||2+||y||2-x\cdot yEJ(x,y)=x\cdot y||x||2+||y||2-x\cdot y$$

知道度量的方法后，我们还要考虑实际的邻近度计算问题

1、距离度量的标准化和相关性

距离度量的一个重要问题是当属性具有不同的值域时如何处理（这种情况通常称作“变量具有不同的尺度”）。前面，使用欧几里得距离，基于年龄和收入两个属性来度量人之间的距离。除非这两个属性是标准化的，否则两个人之间的距离将被收入所左右。

一个相关的问题是，除值域不同外，当某些属性之间还相关时，如何计算距离。当属性相关、具有不同的值域（不同的方差）、并且数据分布近似高斯（正态）分布时，欧几里得距离的拓广，**Mahalanobis 距离**是有用。

$$mahalanobis(x,y)=(x-y)\sum^{-1}(x-y)^T mahalanobis(x,y)=(x-y)\Sigma-1(x-y)T$$

其中$\sum^{-1}\Sigma-1$是数据协方差矩阵的逆。注意，协方差矩阵$\sum\Sigma$是这样的矩阵，它的第 ijij 个元素是第 ii 个和第 jj 个属性的协方差。

计算 Mahalanobis 距离的费用昂贵，但是对于其属性相关的对象来说是值得的。如果属性相对来说不相关，只是具有不同的值域，则只需要对变量进行标准化就足够了。

一般采用 $d'=(d-d_{min})/(d_{max}-d_{min})d'=(d-dmin)/(dmax-dmin)$来变化欧几米得距离的特征值域。

2、组合异种属性的相似度

前面的相似度定义所基于的方法都假定所有属性具有相同类型。当属性具有不同类型时，就需要更一般的方法。直截了当的方法是使用上文的表分别计算出每个属性之间的相似度，然后使用一种导致 0 和 1 之间相似度的方法组合这些相似度。总相似度一般定义为所有属性相似度的平均值。

不幸的是，如果某些属性是非对称属性，这种方法效果不好。处理该问题的最简单方法是：如果两个对象在非对称属性上的值都是 0，则在计算对象相似度时忽略它们。类似的方法也能很好地处理遗漏值。

概括地说，下面的算法可以有效地计算具有不同类型属性的两个对象 **x** 和 **y** 之间的相似度。修改该过程可以很轻松地处理相异度。

算法：

1：对于第 kk 个属性，计算相似度 $s_k(x,y)$sk(x,y)，在区间[0,1]中

2：对于第 kk 个属性，定义一个指示变量 $\delta_k\delta k$，如下：

$\delta_k=0\delta k=0$，如果第 kk 个属性是非对称属性，并且两个对象在该属性上的值都是 0，或者如果一个对象的第 kk 个属性具有遗漏值

$\delta_k=0\delta k=0$，否则

3：使用如下公式计算两个对象之间的总相似度：

$$similarity(x,y)=\sum_{k=1}^{n}\delta_k s_k(x,y)\sum_{k=1}^{n}\delta_k similarity(x,y)=\Sigma k=1n\delta ksk(x,y)\Sigma k=1n\delta k$$

3、使用权值

在前面的大部分讨论中，所有的属性在计算临近度时都会被同等对待。但是，当某些属性对临近度的定义比其他属性更重要时，我们并不希望这种同等对待的方式。为了处理这种情况，可以通过对每个属性的贡献加权来修改临近度公式。

如果权 $w_k$wk 的和为 1，则上面的公式变成：

$$similarity(x,y)=\sum_{k=1}^{n}w_k\delta_k s_k(x,y)\sum_{k=1}^{n}\delta_k similarity(x,y)=\Sigma k=1nwk\delta ksk(x,y)\Sigma k=1n\delta k$$

欧几里得距离的定义也可以修改为：

$$d(x,y)=(\sum_{k=1}^{n}w_k|x_k-y_k|2)1/2d(x,y)=(\Sigma k=1nwk|xk-yk|2)1/2$$

算法代码

算法伪码：

（1）计算已知类别数据集中的点与当前点之间的距离；

（2）按照距离递增次序排序；

（3）选取与当前点距离最小的 k 个点；

（4）确定前 k 个点所在类别的出现频率；

（5）返回前 k 个点出现频率最高的类别作为当前点的预测分类。

具体代码

以下代码都是博主根据自己的理解写的，因为才开始学习不久，如有代码的错误和冗余，请见谅，并同时欢迎指出，谢谢！
博主主要是根据 Pandas 和 Numpy 库来编写的，阅读的同学可能需要有一点这方面的基础。Pandas 和 Numpy 库都是处理数据分析的最佳库，要想学好数据分析，还是需要好好学习这两个库的。
算法采用的是欧几里得距离，采用 $d'=(d-d_{min})/(d_{max}-d_{min})$ $d'=(d-dmin)/(dmax-dmin)$来规范特征值

```
# -*- coding: utf-8 -*-
"""kNN 最近邻算法最重要的三点：
   (1)确定 k 值。k 值过小，对噪声非常敏感；k 值过大，容易误分类
   (2)采用适当的临近性度量。对于不同的类型的数据，应考虑不同的度量方法。除了距离外，也可以考虑相似性。
   (3)数据预处理。需要规范数据，使数据度量范围一致。
"""

import pandas as pd
import numpy as np

class kNN:
    def __init__(self,X,y=None,test='YES'):
        """参数 X 为训练样本集，支持 list，array 和 DataFrame；

           参数 y 为类标号，支持 list,array,Series
           默认参数 y 为空值，表示类标号字段没有单独列出来，而是存储在数据集 X 中的最后一个字段；
           参数 y 不为空值时，数据集 X 中不能含有字段 y

           参数 test 默认为'YES'，表是将原训练集拆分为测试集和新的训练集
        """
        if isinstance(X,pd.core.frame.DataFrame) != True:  #将数据集转换为 DataFrame 格式
            self.X = pd.DataFrame(X)
        else:
            self.X = X
        if y is None:                                      #将特征和类别分开
            self.y = self.X.iloc[:,-1]
            self.X = self.X.iloc[:,:-1]
            self.max_data = np.max(self.X,axis=0)          #获取每个特征的最大值，为下面规范数据用
            self.min_data = np.min(self.X,axis=0)          #获取每个特征的最小值，为下面规范数据用
            max_set = np.zeros_like(self.X); max_set[:] = self.max_data  #以每个特征的最大值，构建一个与训练集结构一样的数据集
            min_set = np.zeros_like(self.X); min_set[:] = self.min_data  #以每个特征的最小值，构建一个与训练集结构一样的数据集
```

```
            self.X = (self.X - min_set)/(max_set - min_set)  #规范训练集
        else:
            self.max_data = np.max(self.X,axis=0)
            self.min_data = np.min(self.X,axis=0)
            max_set = np.zeros_like(self.X); max_set[:] = self.max_data
            min_set = np.zeros_like(self.X); min_set[:] = self.min_data
            self.X = (self.X - min_set)/(max_set - min_set)
            if isinstance(y,pd.core.series.Series) != True:
                self.y = pd.Series(y)
            else:
                self.y = y
        if test == 'YES':        #如果 test 为'YES', 将原训练集拆分为测试集和新的训练集
            self.test = 'YES'    #设置 self.test, 后面 knn 函数判断测试数据需不需要再规范
            allCount = len(self.X)
            dataSet = [i for i in range(allCount)]
            testSet = []
            for i in range(int(allCount*(1/5))):
                randomnum = dataSet[int(np.random.uniform(0,len(dataSet)))]
                testSet.append(randomnum)
                dataSet.remove(randomnum)
            self.X,self.testSet_X = self.X.iloc[dataSet],self.X.iloc[testSet]
            self.y,self.testSet_y = self.y.iloc[dataSet],self.y.iloc[testSet]
        else:
            self.test = 'NO'


    def getDistances(self,point):  #计算训练集每个点与计算点的欧几米得距离
        points = np.zeros_like(self.X)   #获得与训练集 X 一样结构的 0 集
        points[:] = point
        minusSquare = (self.X - points)**2
        EuclideanDistances = np.sqrt(minusSquare.sum(axis=1))  #训练集每个点与特殊点的欧几米得距离
        return EuclideanDistances


    def getClass(self,point,k):    #根据距离最近的 k 个点判断计算点所属类别
```

```
        distances = self.getDistances(point)
        argsort = distances.argsort(axis=0)      #根据数值大小，进行索引排序
        classList = list(self.y.iloc[argsort[0:k]])
        classCount = {}
        for i in classList:
            if i not in classCount:
                classCount[i] = 1
            else:
                classCount[i] += 1
        maxCount = 0
        maxkey = 'x'
        for key in classCount.keys():
            if classCount[key] > maxCount:
                maxCount = classCount[key]
                maxkey = key
        return maxkey


    def knn(self,testData,k):      #kNN 计算，返回测试集的类别
        if self.test == 'NO':      #如果 self.test == 'NO'，需要规范测试数据（参照上面__init__）
            testData = pd.DataFrame(testData)
            max_set = np.zeros_like(testData); max_set[:] = self.max_data
            min_set = np.zeros_like(testData); min_set[:] = self.min_data
            testData = (testData - min_set)/(max_set - min_set)   #规范测试集
        if testData.shape == (len(testData),1):  #判断 testData 是否是一行记录
            label = self.getClass(testData.iloc[0],k)
            return label                         #一行记录直接返回类型
        else:
            labels = []
            for i in range(len(testData)):
                point = testData.iloc[i,:]
                label = self.getClass(point,k)
                labels.append(label)
            return labels                        #多行记录则返回类型的列表
```

```
    def errorRate(self,knn_class,real_class):    #计算 kNN 错误率,knn_class 为算法计算的类别，real_class 为真实的类别
        error = 0
        allCount = len(real_class)
        real_class = list(real_class)
        for i in range(allCount):
            if knn_class[i] != real_class[i]:
                error += 1
        return error/allCount
```

下面利用 sklearn 库里的 iris 数据（sklearn 是数据挖掘算法库），进行上述代码测试

```
from sklearn import datasets
sets = datasets.load_iris()  #载入 iris 数据集
X = sets.data  #特征值数据集
y = sets.target  #类别数据集
myknn = kNN(X,y)
knn_class = myknn.knn(myknn.testSet_X,4)
errorRate = myknn.errorRate(knn_class,myknn.testSet_y)
```

kNN 算法到此结束。如果发现有什么问题，欢迎大家指出。

机器学习实战笔记(Python 实现)-01-K 近邻算法(KNN)

目录

正文

本系列文章为《机器学习实战》学习笔记，内容整理自书本，网络以及自己的理解，如有错误欢迎指正。

源码在 Python3.5 上测试均通过，代码及数据 --> https://github.com/Wellat/MLaction

---

1 算法概述

1.1 算法特点

简单地说，k-近邻算法采用测量不同特征值之间的距离方法进行分类。

优点：精度高、对异常值不敏感、无数据输入假定

缺点：计算复杂度高、空间复杂度高

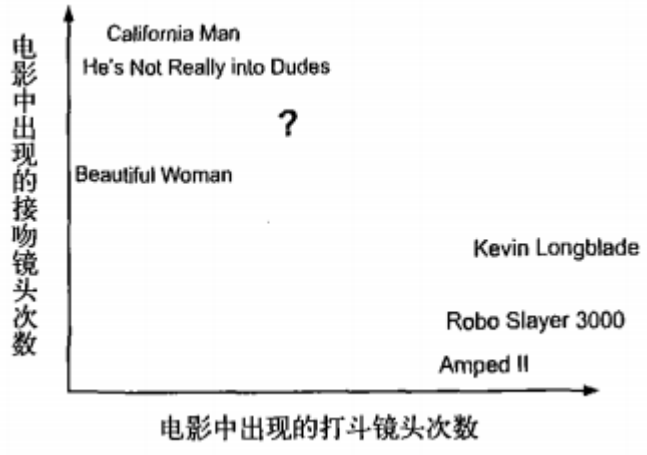
适用数据范围：数值型和标称型

1.2 工作原理

存在一个训练样本集，并且每个样本都存在标签（有监督学习）。输入没有标签的新样本数据后，将新数据的每个特征与样本集中数据对应的特征进行比较，然后算法提取出与样本集中特征最相似的数据（最近邻）的分类标签。一般来说，我们只选择样本数据集中前 k 个最相似的数据，这就是 k-近邻算法中 k 的出处，而且 k 通常不大于 20。最后选择 k 个最相似数据中出现次数最多的分类，作为新数据的分类。

1.3 实例解释

以电影分类为例子，使用 k-近邻算法分类爱情片和动作片。有人曾经统计过很多电影的打斗镜头和接吻镜头，下图显示了 6 部电影的打斗和接吻镜头数。 假如有一部未看过的电影，如何确定它是爱情片还是动作片呢？



①首先需要统计这个未知电影存在多少个打斗镜头和接吻镜头，下图中问号位置是该未知电影出现的镜头数

每部电影的打斗镜头数、接吻镜头数以及电影评估类型

电影名称	打斗镜头	接吻镜头	电影类型
California Man	3	104	爱情片
He's Not Really into Dudes	2	100	爱情片
Beautiful Woman	1	81	爱情片
Kevin Longblade	101	10	动作片
Robo Slayer 3000	99	5	动作片
Amped II	98	2	动作片
?	18	90	未知

②之后计算未知电影与样本集中其他电影的距离（相似度），具体算法先忽略，结果如下表所示：

已知电影与未知电影的距离

电影名称	与未知电影的距离
California Man	20.5
He's Not Really into Dudes	18.7
Beautiful Woman	19.2
Kevin Longblade	115.3
Robo Slayer 3000	117.4
Amped II	118.9

③将相似度列表排序，选出前 k 个最相似的样本。此处我们假设 k=3，将上表中的相似度进行排序后前 3 分别是：He’s Not Really into Dudes，Beautiful Woman，California Man。

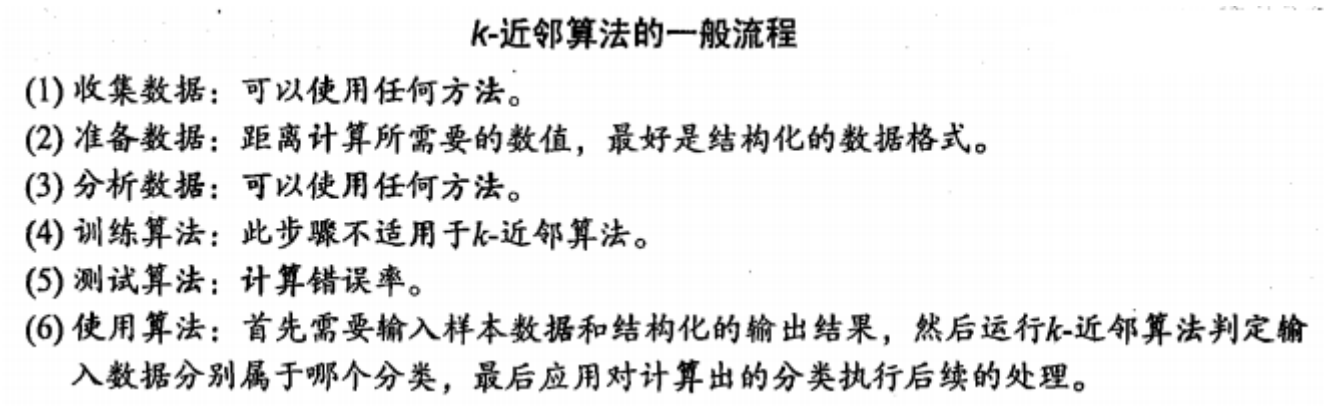
④统计最相似样本的分类。此处很容易知道这 3 个样本均为爱情片。

⑤将分类最多的类别作为未知电影的分类。那么我们就得出结论，未知电影属于爱情片。

2 代码实现

2.1 k-近邻简单分类的应用

2.1.1 算法一般流程



k-近邻算法的一般流程

(1) 收集数据：可以使用任何方法。
(2) 准备数据：距离计算所需要的数值，最好是结构化的数据格式。
(3) 分析数据：可以使用任何方法。
(4) 训练算法：此步骤不适用于k-近邻算法。
(5) 测试算法：计算错误率。
(6) 使用算法：首先需要输入样本数据和结构化的输出结果，然后运行k-近邻算法判定输入数据分别属于哪个分类，最后应用对计算出的分类执行后续的处理。

2.1.2 Python 实现代码及注释

```
#coding=UTF8
from numpy import *
import operator

def createDataSet():
    """
    函数作用：构建一组训练数据（训练样本），共 4 个样本
    同时给出了这 4 个样本的标签，及 labels
    """
    group = array([
        [1.0, 1.1],
        [1.0, 1.0],
        [0. , 0. ],
        [0. , 0.1]
    ])
    labels = ['A', 'A', 'B', 'B']
    return group, labels
```

```
18
19 def classify0(inX, dataset, labels, k):
20     """
21     inX 是输入的测试样本，是一个[x, y]样式的
22     dataset 是训练样本集
23     labels 是训练样本标签
24     k 是 top k 最相近的
25     """
26     # shape 返回矩阵的[行数，列数]，
27     # 那么 shape[0]获取数据集的行数，
28     # 行数就是样本的数量
29     dataSetSize = dataset.shape[0]
30
31     """
32     下面的求距离过程就是按照欧氏距离的公式计算的。
33     即 根号(x^2+y^2)
34     """
35     # tile 属于 numpy 模块下边的函数
36     # tile（A, reps）返回一个 shape=reps 的矩阵，矩阵的每个元素是 A
37     # 比如 A=[0,1,2] 那么，tile(A, 2)= [0, 1, 2, 0, 1, 2]
38     # tile(A,(2,2)) = [[0, 1, 2, 0, 1, 2],
39     #                  [0, 1, 2, 0, 1, 2]]
40     # tile(A,(2,1,2)) = [[[0, 1, 2, 0, 1, 2]],
41     #                    [[0, 1, 2, 0, 1, 2]]]
42     # 上边那个结果的分开理解就是：
43     # 最外层是 2 个元素，即最外边的[]中包含 2 个元素，类似于[C,D],而此处的 C=D，因为是复制出来的
44     # 然后 C 包含 1 个元素，即 C=[E],同理 D=[E]
45     # 最后 E 包含 2 个元素，即 E=[F,G],此处 F=G，因为是复制出来的
46     # F 就是 A 了，基础元素
47     # 综合起来就是(2,1,2)= [C, C] = [[E], [E]] = [[[F, F]], [[F, F]]] = [[[A, A]], [[A, A]]]
48     # 这个地方就是为了把输入的测试样本扩展为和 dataset 的 shape 一样，然后就可以直接做矩阵减法了。
49     # 比如，dataset 有 4 个样本，就是 4*2 的矩阵，输入测试样本肯定是一个了，就是 1*2，为了计算输入样本与训练样本的距离
50     # 那么，需要对这个数据进行作差。这是一次比较，因为训练样本有 n 个，那么就要进行 n 次比较；
51     # 为了方便计算，把输入样本复制 n 次，然后直接与训练样本作矩阵差运算，就可以一次性比较了 n 个样本。
52     # 比如 inX = [0,1],dataset 就用函数返回的结果，那么
```

```
    # tile(inX, (4,1))= [[ 0.0, 1.0],
    #                    [ 0.0, 1.0],
    #                    [ 0.0, 1.0],
    #                    [ 0.0, 1.0]]
    # 作差之后
    # diffMat = [[-1.0,-0.1],
    #            [-1.0, 0.0],
    #            [ 0.0, 1.0],
    #            [ 0.0, 0.9]]
    diffMat = tile(inX, (dataSetSize, 1)) - dataset

    # diffMat 就是输入样本与每个训练样本的差值，然后对其每个 x 和 y 的差值进行平方运算。
    # diffMat 是一个矩阵，矩阵**2 表示对矩阵中的每个元素进行**2 操作，即平方。
    # sqDiffMat = [[1.0, 0.01],
    #              [1.0, 0.0 ],
    #              [0.0, 1.0 ],
    #              [0.0, 0.81]]
    sqDiffMat = diffMat ** 2

    # axis=1 表示按照横轴，sum 表示累加，即按照行进行累加。
    # sqDistance = [[1.01],
    #               [1.0 ],
    #               [1.0 ],
    #               [0.81]]
    sqDistance = sqDiffMat.sum(axis=1)

    # 对平方和进行开根号
    distance = sqDistance ** 0.5

    # 按照升序进行快速排序，返回的是原数组的下标。
    # 比如，x = [30, 10, 20, 40]
    # 升序排序后应该是[10,20,30,40],他们的原下标是[1,2,0,3]
    # 那么，numpy.argsort(x) = [1, 2, 0, 3]
    sortedDistIndicies = distance.argsort()

```

```
    # 存放最终的分类结果及相应的结果投票数
    classCount = {}

    # 投票过程，就是统计前 k 个最近的样本所属类别包含的样本个数
    for i in range(k):
        # index = sortedDistIndicies[i]是第 i 个最相近的样本下标
        # voteIlabel = labels[index]是样本 index 对应的分类结果('A' or 'B')
        voteIlabel = labels[sortedDistIndicies[i]]
        # classCount.get(voteIlabel, 0)返回 voteIlabel 的值，如果不存在，则返回 0
        # 然后将票数增 1
        classCount[voteIlabel] = classCount.get(voteIlabel, 0) + 1

    # 把分类结果进行排序，然后返回得票数最多的分类结果
    sortedClassCount = sorted(classCount.items(), key=operator.itemgetter(1), reverse=True)
    return sortedClassCount[0][0]

if __name__== "__main__":
    # 导入数据
    dataset, labels = createDataSet()
    inX = [0.1, 0.1]
    # 简单分类
    className = classify0(inX, dataset, labels, 3)
    print('the class of test sample is %s' %className)
```

2.2 在约会网站上使用 k-近邻算法

2.2.1 算法一般流程

示例：在约会网站上使用k-近邻算法

(1) 收集数据：提供文本文件。
(2) 准备数据：使用Python解析文本文件。
(3) 分析数据：使用Matplotlib画二维扩散图。
(4) 训练算法：此步骤不适用于k-近邻算法。
(5) 测试算法：使用海伦提供的部分数据作为测试样本。
测试样本和非测试样本的区别在于：测试样本是已经完成分类的数据，如果预测分类与实际类别不同，则标记为一个错误。
(6) 使用算法：产生简单的命令行程序，然后海伦可以输入一些特征数据以判断对方是否为自己喜欢的类型。

2.2.2 Python 实现代码

datingTestSet.txt 文件中有 1000 行的约会数据，样本主要包括以下 3 种特征：

- 每年获得的飞行常客里程数
- 玩视频游戏所耗时间百分比
- 每周消费的冰淇淋公升数

将上述特征数据输人到分类器之前，必须将待处理数据的格式改变为分类器可以接受的格式 。在 kNN.py 中创建名为 file2matrix 的函数，以此来处理输人格式问题。该函数的输人为文件名字符串，输出为训练样本矩阵和类标签向量。autoNorm 为数值归一化函数，将任意取值范围的特征值转化为 0 到 1 区间内的值。最后，datingClassTest 函数是测试代码。

将下面的代码增加到 kNN.py 中。

```
def file2matrix(filename):
    """
    从文件中读入训练数据，并存储为矩阵
    """
    fr = open(filename)
    arrayOlines = fr.readlines()
    numberOfLines = len(arrayOlines)   #获取 n=样本的行数
    returnMat = zeros((numberOfLines,3))    #创建一个 2 维矩阵用于存放训练样本数据，一共有 n 行，每一行存放 3 个数据
    classLabelVector = []    #创建一个 1 维数组用于存放训练样本标签。
    index = 0
    for line in arrayOlines:
        # 把回车符号给去掉
```

```
13          line = line.strip()
14          # 把每一行数据用\t 分割
15          listFromLine = line.split('\t')
16          # 把分割好的数据放至数据集，其中 index 是该样本数据的下标，就是放到第几行
17          returnMat[index,:] = listFromLine[0:3]
18          # 把该样本对应的标签放至标签集，顺序与样本集对应。
19          classLabelVector.append(int(listFromLine[-1]))
20          index += 1
21      return returnMat,classLabelVector
22
23  def autoNorm(dataSet):
24      """
25      训练数据归一化
26      """
27      # 获取数据集中每一列的最小数值
28      # 以 createDataSet()中的数据为例，group.min(0)=[0,0]
29      minVals = dataSet.min(0)
30      # 获取数据集中每一列的最大数值
31      # group.max(0)=[1, 1.1]
32      maxVals = dataSet.max(0)
33      # 最大值与最小的差值
34      ranges = maxVals - minVals
35      # 创建一个与 dataSet 同 shape 的全 0 矩阵，用于存放归一化后的数据
36      normDataSet = zeros(shape(dataSet))
37      m = dataSet.shape[0]
38      # 把最小值扩充为与 dataSet 同 shape，然后作差，具体 tile 请翻看 第三节 代码中的 tile
39      normDataSet = dataSet - tile(minVals, (m,1))
40      # 把最大最小差值扩充为 dataSet 同 shape，然后作商，是指对应元素进行除法运算，而不是矩阵除法。
41      # 矩阵除法在 numpy 中要用 linalg.solve(A,B)
42      normDataSet = normDataSet/tile(ranges, (m,1))
43      return normDataSet, ranges, minVals
44
45  def datingClassTest():
46      # 将数据集中 10%的数据留作测试用，其余的 90%用于训练
47      hoRatio = 0.10
```

```
    datingDataMat,datingLabels = file2matrix('datingTestSet2.txt')       #load data setfrom file
    normMat, ranges, minVals = autoNorm(datingDataMat)
    m = normMat.shape[0]
    numTestVecs = int(m*hoRatio)
    errorCount = 0.0
    for i in range(numTestVecs):
        classifierResult = classify0(normMat[i,:],normMat[numTestVecs:m,:],datingLabels[numTestVecs:m],3)
        print("the classifier came back with: %d, the real answer is: %d, result is :%s" % (classifierResult,
datingLabels[i],classifierResult==datingLabels[i]))
        if (classifierResult != datingLabels[i]): errorCount += 1.0
    print("the total error rate is: %f" % (errorCount/float(numTestVecs)))
    print(errorCount)
```

2.3 手写识别系统实例

2.3.1 实例数据

为了简单起见，这里构造的系统只能识别数字 0 到 9。需要识别的数字已经使用图形处理软件，处理成具有相同的色彩和大小：宽高是 32 像素 x 32 像素的黑白图像。尽管采用文本格式存储图像不能有效地利用内存空间，但是为了方便理解，我们还是将图像转换为文本格式。

trainingDigits 是 2000 个训练样本，testDigits 是 900 个测试样本。

2.3.2 算法的流程

示例：使用k-近邻算法的手写识别系统

(1) 收集数据：提供文本文件。

(2) 准备数据：编写函数classify0()，将图像格式转换为分类器使用的list格式。

(3) 分析数据：在Python命令提示符中检查数据，确保它符合要求。

(4) 训练算法：此步骤不适用于k-近邻算法。

(5) 测试算法：编写函数使用提供的部分数据集作为测试样本，测试样本与非测试样本的区别在于测试样本是已经完成分类的数据，如果预测分类与实际类别不同，则标记为一个错误。

(6) 使用算法：本例没有完成此步骤，若你感兴趣可以构建完整的应用程序，从图像中提取数字，并完成数字识别，美国的邮件分拣系统就是一个实际运行的类似系统。

2.3.3 Python 实现代码

将下面的代码增加到 kNN.py 中，img2vector 为图片转换成向量的方法，handwritingClassTest 为测试方法：

```
from os import listdir
def img2vector(filename):
    """
    将图片数据转换为 01 矩阵。
    每张图片是 32*32 像素，也就是一共 1024 个字节。
    因此转换的时候，每行表示一个样本，每个样本含 1024 个字节。
    """
    # 每个样本数据是 1024=32*32 个字节
    returnVect = zeros((1,1024))
    fr = open(filename)
    # 循环读取 32 行，32 列。
    for i in range(32):
        lineStr = fr.readline()
        for j in range(32):
            returnVect[0,32*i+j] = int(lineStr[j])
    return returnVect

def handwritingClassTest():
    hwLabels = []
    # 加载训练数据
    trainingFileList = listdir('trainingDigits')
    m = len(trainingFileList)
    trainingMat = zeros((m,1024))
    for i in range(m):
        # 从文件名中解析出当前图像的标签，也就是数字是几
        # 文件名格式为 0_3.txt 表示图片数字是 0
        fileNameStr = trainingFileList[i]
        fileStr = fileNameStr.split('.')[0]     #take off .txt
        classNumStr = int(fileStr.split('_')[0])
        hwLabels.append(classNumStr)
        trainingMat[i,:] = img2vector('trainingDigits/%s' % fileNameStr)
    # 加载测试数据
    testFileList = listdir('testDigits')        #iterate through the test set
    errorCount = 0.0
```

```
    mTest = len(testFileList)
    for i in range(mTest):
        fileNameStr = testFileList[i]
        fileStr = fileNameStr.split('.')[0]     #take off .txt
        classNumStr = int(fileStr.split('_')[0])
        vectorUnderTest = img2vector('testDigits/%s' % fileNameStr)
        classifierResult = classify0(vectorUnderTest, trainingMat, hwLabels, 3)
        print("the classifier came back with: %d, the real answer is: %d, The predict result is: %s" % (classifierResult,
classNumStr, classifierResult==classNumStr))
        if (classifierResult != classNumStr): errorCount += 1.0
    print("\nthe total number of errors is: %d / %d" %(errorCount, mTest))
    print("\nthe total error rate is: %f" % (errorCount/float(mTest)))
```

k-近邻算法识别手写数字数据集，错误率为 1.2%。改变变量 k 的值、修改函数 handwritingClassTest 随机选取训练样本、改变训练样本的数目，都会对 k-近邻算法的错误率产生影响，感兴趣的话可以改变这些变量值，观察错误率的变化。

k-近邻算法是分类数据最简单最有效的算法。它必须保存全部数据集，如果训练数据集很大，必须使用大量的存储空间。此外，由于必须对数据集中的每个数据计算距离值，实际使用时可能非常耗时。其另一个缺陷是它无法给出任何数据的基础结构信息，因此我们也无法知晓平均实例样本和典型实例样本具有什么特征。

3 应用 scikit-learn 库实现 k 近邻算法

```
"""
scikit-learn 库对 knn 的支持
数据集是 iris 虹膜数据集
"""

from sklearn.datasets import load_iris
from sklearn import neighbors
import sklearn

#查看 iris 数据集
iris = load_iris()
print(iris)

'''
KNeighborsClassifier(n_neighbors=5, weights='uniform',
                     algorithm='auto', leaf_size=30,
                     p=2, metric='minkowski',
```

```
                      metric_params=None, n_jobs=1, **kwargs)
n_neighbors: 默认值为5，表示查询k个最近邻的数目
algorithm:   {‘auto’, ‘ball_tree’, ‘kd_tree’, ‘brute’},指定用于计算最近邻的算法，auto表示试图采用最适合的算法计算最近邻
leaf_size:   传递给‘ball_tree’或‘kd_tree’的叶子大小
metric:      用于树的距离度量。默认'minkowski与P = 2（即欧氏度量）
n_jobs:      并行工作的数量，如果设为-1，则作业的数量被设置为CPU内核的数量
查看官方api：http://scikit-
learn.org/dev/modules/generated/sklearn.neighbors.KNeighborsClassifier.html#sklearn.neighbors.KNeighborsClassifier
'''
knn = neighbors.KNeighborsClassifier()
#训练数据集
knn.fit(iris.data, iris.target)
#训练准确率
score = knn.score(iris.data, iris.target)

#预测
predict = knn.predict([[0.1,0.2,0.3,0.4]])
#预测，返回概率数组
predict2 = knn.predict_proba([[0.1,0.2,0.3,0.4]])

print(predict)
print(iris.target_names[predict])
```

代码解释参考原贴：http://blog.csdn.net/niuwei22007/article/details/49703719

分类: 机器学习