AI学习笔记--sklearn--KNN算法

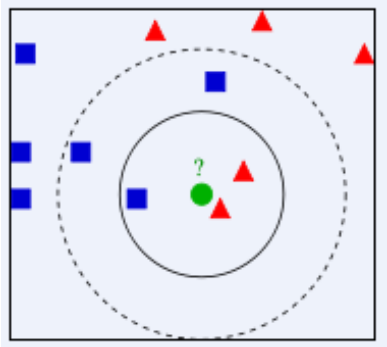
- ## 简介

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个最相邻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，并具有这个类别上样本的特性。该方法在确定分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。 kNN方法在类别决策时，只与极少量的相邻样本有关。由于kNN方法主要靠周围有限的邻近的样本，而不是靠判别类域的方法来确定所属类别的，因此对于类域的交叉或重叠较多的待分样本集来说，kNN方法较其他方法更为适合。

- ## 算法原理

图中，绿色圆要被决定赋予哪个类，是红色三角形还是蓝色四方形？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，由于蓝色四方形比例为3/5，因此绿色圆被赋予蓝色四方形类。



K最近邻(k-Nearest Neighbor，KNN)分类算法，是一个理论上比较成熟的方法，也是最简单的机器学习算法之一。该方法的思路是：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。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。 KNN方法虽然从原理上也依赖于极限定理，但在类别决策时，只与极少量的相邻样本有关。由于KNN方法主要靠周围有限的邻近的样本，而不是靠判别类域的方法来确定所属类别的，因此对于类域的交叉或重叠较多的待分样本集来说，KNN方法较其他方法更为适合。

KNN算法不仅可以用于分类，还可以用于回归。通过找出一个样本的k个最近邻居，将这些邻居的属性的平均值赋给该样本，就可以得到该样本的属性。更有用的方法是将不同距离的邻居对该样本产生的影响给予不同的权值(weight)，如权值与距离成反比。

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性指标，避免了对象之间的匹配问题，在这里距离一般使用欧氏距离或曼哈顿距离：

$$\text{欧式距离：}\ d(x,y)=\sqrt{\sum_{k=1}^{n}(x_k-y_k)^2}\ ,\qquad \text{曼哈顿距离：}\ d(x,y)=\sqrt{\sum_{k=1}^{n}|x_k-y_k|}$$

举例的选择有很多种，常用的距离函数：

1）Minkowski Distance 公式 —— λ 可以随意取值，可以是负数，也可以是正数，或是无穷大

$$d_{ij}=\sqrt[\lambda]{\sum_{k=1}^{n}\left|x_{ik}-x_{jk}\right|^{\lambda}}$$

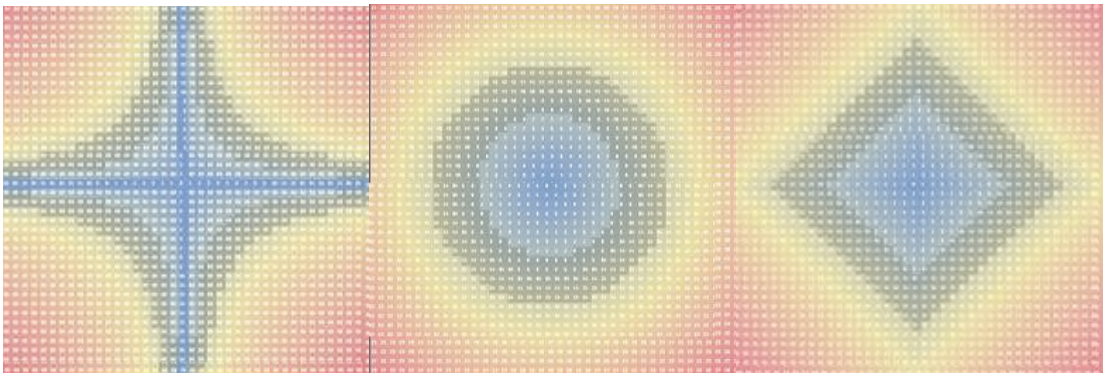
2）Euclidean Distance 公式 —— 也就是第一个公式 λ=2 的情况

$$d_{ij}=\sqrt{\sum_{k=1}^{n}\left(x_{ik}-x_{jk}\right)^2}$$

3）CityBlock Distance 公式 —— 也就是第一个公式 λ=1 的情况

$$d_{ij}=\sum_{k=1}^{n}\left|x_{ik}-x_{jk}\right|$$

这三个公式的求中心点有一些不一样的地方，我们看下图（对于第一个 λ 在 0-1之间）。



（1）Minkowski Distance （2）Euclidean Distance （3）CityBlock Distance

- 算法对比KMeans

Kmeans算法缺点

- 聚类中心的个数K 需要事先给定，但在实际中这个 K 值的选定是非常难以估计的，很多时候，事先并不知道给定的数据集应该分成多少个类别才最合适
- Kmeans需要人为地确定初始聚类中心，不同的初始聚类中心可能导致完全不同的聚类结果。（可以使用Kmeans++算法来解决）

k-means++代码：http://rosettacode.org/wiki/K-means%2B%2B_clustering

KNN	K-Means
1.KNN是分类算法 2.监督学习 3.喂给它的数据集是带label的数据，已经是完全正确的数据	1.K-Means是聚类算法 2.非监督学习 3.喂给它的数据集是无label的数据，是杂乱无章的，经过聚类后才变得有点顺序，先无序，后有序
没有明显的前期训练过程，属于memory-based learning	有明显的前期训练过程
K的含义：来了一个样本x，要给它分类，即求出它的y，就从数据集中，在x附近找离它最近的K个数据点，这K个数据点，类别c占的个数最多，就把x的label设为c	K的含义：K是人工固定好的数字，假设数据集合可以分为K个簇，由于是依靠人工定好，需要一点先验知识
相似点：都包含这样的过程，给定一个点，在数据集中找离它最近的点。即二者都用到了NN(Nears Neighbor)算法，一般用KD树来实现NN。	

- 算法示例

```
# !/usr/bin/env python
# -*- coding:utf-8 -*-

"""
    KNN algorithm
    use KNN algorithm to achieve digital recognition
"""
```

```python
import numpy as np
from os import listdir


# 读取32*32图像文本,转为向量返回
def img2vector(filename):
    retVect = np.zeros((1, 1024))
    fr = open(filename)
    for i in range(32):
        lineStr = fr.readline()
        for j in range(32):
            retVect[0, 32 * i + j] = int(lineStr[j])
    return retVect


# 预处理训练集和数据集，将图片文本向量化
def preprocessing():
    # 训练集
    trainFileList = listdir("trainingData/")
    m_train = len(trainFileList)
    trainData = np.zeros((m_train, 1024))
    trainLabel = []
    for i in range(m_train):
        trainData[i, :] = img2vector("trainingData/" + trainFileList[i])
        trainLabel.append(int(trainFileList[i].split('_')[0]))
    trainLabel = np.array(trainLabel)

    # 测试集
    testFileList = listdir("testData/")
    m_test = len(testFileList)
    testData = np.zeros((m_test, 1024))
    testLabel = []
    for i in range(m_test):
        testData[i, :] = img2vector("testData/" + testFileList[i])
        testLabel.append(int(testFileList[i].split('_')[0]))
    testLabel = np.array(testLabel)
    return trainData, trainLabel, testData, testLabel


# kNN算法
def kNN(testData, trainDataMat, trainLabelMat, k):
    m = trainDataMat.shape[0]

    # 计算距离,使用欧式距离
    difMat = trainDataMat - testData
    sqDifMat = difMat ** 2
    sqDistance = sqDifMat.sum(axis=1)
    distance = sqDistance ** 0.5

    # 对距离向量进行排序，获得其排序后的index
    sortedDistanceIndicies = distance.argsort()

    # 对前k个label进行计数排序
```

```
    classCount = {}
    for i in range(k):
        voteLabel = trainLabelMat[sortedDistanceIndicies[i]]
        classCount[voteLabel] = classCount.get(voteLabel, 0) + 1

    # 按item进行排序逆序排序，第一个label即为选举出的label
    result = sorted(classCount.items(), key=lambda item: item[1],
reverse=True)
    return result[0][0]


if __name__ == "__main__":
    # 获取训练集和测试集的向量
    trainDataMat, trainLabelMat, testDataMat, testLabelMat = preprocessing()

    m = testDataMat.shape[0]
    count = 0
    # 预测过程
    for i in range(m):
        predict = kNN(testDataMat[i, :], trainDataMat, trainLabelMat, 5)
        if predict != testLabelMat[i]:
            count += 1
        print("预测结果为: %d, 期望结果为: %d" % (predict, testLabelMat[i]))

    print("预测错误率为: %.2f %%" % (count / m * 100))
```

训练数据:

数据集合.zip

运行结果:

```
预测结果为: 8, 期望结果为: 8
预测结果为: 5, 期望结果为: 5
预测结果为: 5, 期望结果为: 5
预测结果为: 2, 期望结果为: 2
预测结果为: 2, 期望结果为: 2
预测结果为: 5, 期望结果为: 5
预测结果为: 5, 期望结果为: 5
预测结果为: 9, 期望结果为: 9
预测结果为: 3, 期望结果为: 3
预测结果为: 7, 期望结果为: 7
预测结果为: 7, 期望结果为: 7
预测结果为: 9, 期望结果为: 9
预测结果为: 0, 期望结果为: 0
预测结果为: 0, 期望结果为: 0
预测结果为: 4, 期望结果为: 4
预测结果为: 9, 期望结果为: 9
预测结果为: 7, 期望结果为: 7
预测结果为: 7, 期望结果为: 7
预测结果为: 1, 期望结果为: 1
```

```
预测结果为: 5, 期望结果为: 5
预测结果为: 4, 期望结果为: 4
预测结果为: 3, 期望结果为: 3
预测结果为: 3, 期望结果为: 3
预测错误率为: 0.00 %
```

Demo2:

```python
#!/usr/bin/python
# coding=utf-8
##############################################
# 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[
    # 假如:
    # Newinput: [1,0,2]
    # Dataset:
    # [1,0,1]
    # [2,1,3]
    # [1,0,2]
    # 计算过程即为:
    # 1、求差
    # [1,0,1]       [1,0,2]
    # [2,1,3]   --   [1,0,2]
    # [1,0,2]       [1,0,2]
    # =
    # [0,0,-1]
    # [1,1,1]
    # [0,0,-1]
    # 2、对差值平方
```

```
    # [0,0,1]
    # [1,1,1]
    # [0,0,1]
    # 3、将平方后的差值累加
    # [1]
    # [3]
    # [1]
    # 4、将上一步骤的值求开方，即得距离
    # [1]
    # [1.73]
    # [1]
    #
    # ]
    # 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
```

```
#!/usr/bin/python
# coding=utf-8
import KNNTest
from numpy import *
# 生成数据集和类别标签
dataSet, labels = KNNTest.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 = KNNTest.kNNClassify(testX, dataSet, labels, 3)
print "Your input is:", testX, "and classified to class: ", outputLabel

testX = array([0.1, 0.3])
outputLabel = KNNTest.kNNClassify(testX, dataSet, labels, 3)
print "Your input is:", testX, "and classified to class: ", outputLabel
```

输出结果：

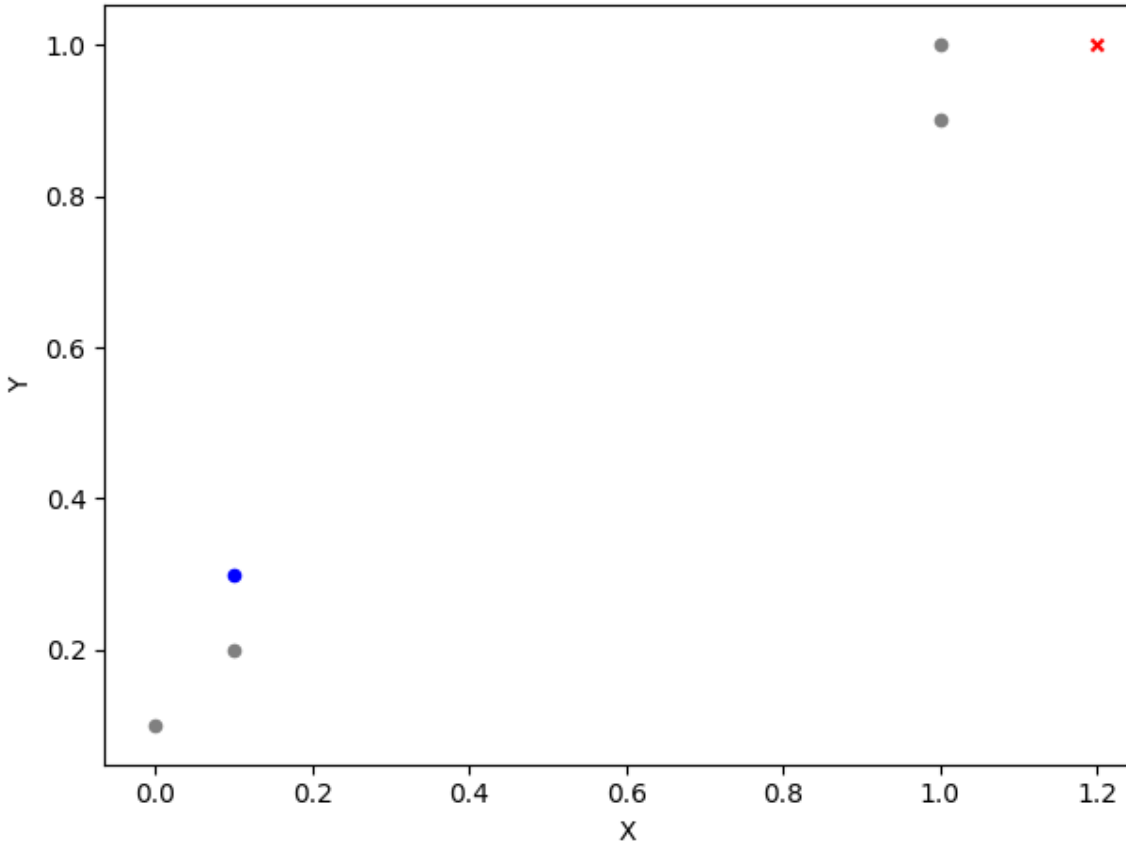
```
Your input is: [1.2 1. ] and classified to class:  A
Your input is: [0.1 0.3] and classified to class:  B
```

如果用图形方式可以更加直观的显示，则修改代码：

```
#!/usr/bin/python
# coding=utf-8
import KNNTest
from numpy import *
import matplotlib.pyplot as plt
from matplotlib.lines import Line2D
# 生成数据集和类别标签
group = array([[1.0, 0.9], [1.0, 1.0], [0.1, 0.2], [0.0, 0.1]])
dataSet, labels = KNNTest.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 = KNNTest.kNNClassify(testX, dataSet, labels, 3)
print "Your input is:", testX, "and classified to class: ", outputLabel
fig = plt.figure()
axes = fig.add_subplot(111)

labels_x = []
labels_y = []
for i in range(len(group)):
    labels_x.append(group[i][0])
    labels_y.append(group[i][1])
type3 = axes.scatter(labels_x,labels_y,marker='o', s=20 ,c='gray')
type1 = axes.scatter(testX[0], testX[1], marker='x', s=20, c='red')
testX = array([0.1, 0.3])
type2 = axes.scatter(testX[0], testX[1], marker='o', s=20, c='blue')
outputLabel = KNNTest.kNNClassify(testX, dataSet, labels, 3)
print "Your input is:", testX, "and classified to class: ", outputLabel
plt.xlabel('X')
plt.ylabel('Y')
plt.show()
```

图中，A分类为左下角两个采集，B分类为右上角两个采集点，依据分类，KNN算法可以归纳测试数据取得的归熟所属A还是B。结合打印和图可以看出相应的规律。


1.0
0.8
0.6
0.4
0.2
Y
0.0
0.2
0.4
0.6
0.8
1.0
1.2
X