Welcome

MSRA Researcher新手，现阶段从事多模态大模型预训练方向。对NLP感兴趣或对博客内容有疑惑及意见建议的，欢迎评论或添加我微信。此外如果有需要内推的同学，也欢迎来骚扰我。联系方式详见concat页面。

前往GitHub (https://github.com/Dod-o)

统计学习方法 | K近邻原理剖析及实现

11月 19, 2018 (http://www.pkudodo.com/2018/11/)

统计学习方法|K近邻原理剖析及实现

作者 Dodo (http://www.pkudodo.com/author/root/) 在 (http://www.pkudodo.com/2018/11/19/1-2/)
机器学习 (http://www.pkudodo.com/category/%e6%9c%ba%e5%99%a8%e5%ad%a6%e4%b9%a0/) 标签
K近邻 (http://www.pkudodo.com/tag/k%e8%bf%91%e9%82%bb/),
实现 (http://www.pkudodo.com/tag/%e5%ae%9e%e7%8e%b0/),
统计方法学习
(http://www.pkudodo.com/tag/%e7%bb%9f%e8%ae%a1%e6%96%b9%e6%b3%95%e5%ad%a6%e4%b9%a0/)

阅读数：16,568

前言

《统计学习方法》一书在前几天正式看完，由于这本书在一定程度上对于初学者是有一些难度的，趁着热乎劲把自己走过的弯路都写出来，后人也能走得更顺畅一点。

以下是我的github地址，其中有《统计学习方法》书中所有涉及到的算法的实现，也是在写博客的同时编写的。在编写宗旨上是希望所有人看完书以后，按照程序的思路再配合书中的公式，能知道怎么讲所学的都应用上。（有点自恋地讲）程序中的注释可能比大部分的博客内容都多。希望大家能够多多捧场，如果可以的话，为我的github打颗星，也能让更多的人看到。

github：

GitHub|手写实现李航《统计学习方法》书中全部算法 (https://github.com/Dod-o/Statistical-Learning-Method_Code)

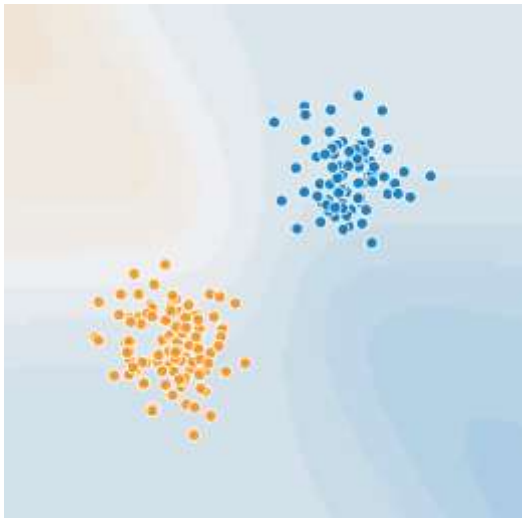
相关博文：

1. 统计学习方法|感知机原理剖析及实现 (http://www.pkudodo.com/2018/11/18/1-4/)
2. 统计学习方法|K近邻原理剖析及实现 (http://www.pkudodo.com/2018/11/19/1-2/)
3. 统计学习方法|朴素贝叶斯原理剖析及实现 (http://www.pkudodo.com/2018/11/21/1-3/)
4. 统计学习方法|决策树原理剖析及实现 (http://www.pkudodo.com/2018/11/30/1-5/)
5. 统计学习方法|逻辑斯蒂原理剖析及实现 (http://www.pkudodo.com/2018/12/03/1-6/)

正文

K近邻的直观理解

我们先看一张图：



(http://www.pkudodo.com/wp-content/uploads/2018/11/K近邻_两堆点.png)

在感知机一节中我们看过这张图，当时使用划分超平面的方式来划分数据。那么除了划线，还有别的方式来划分数据吗？

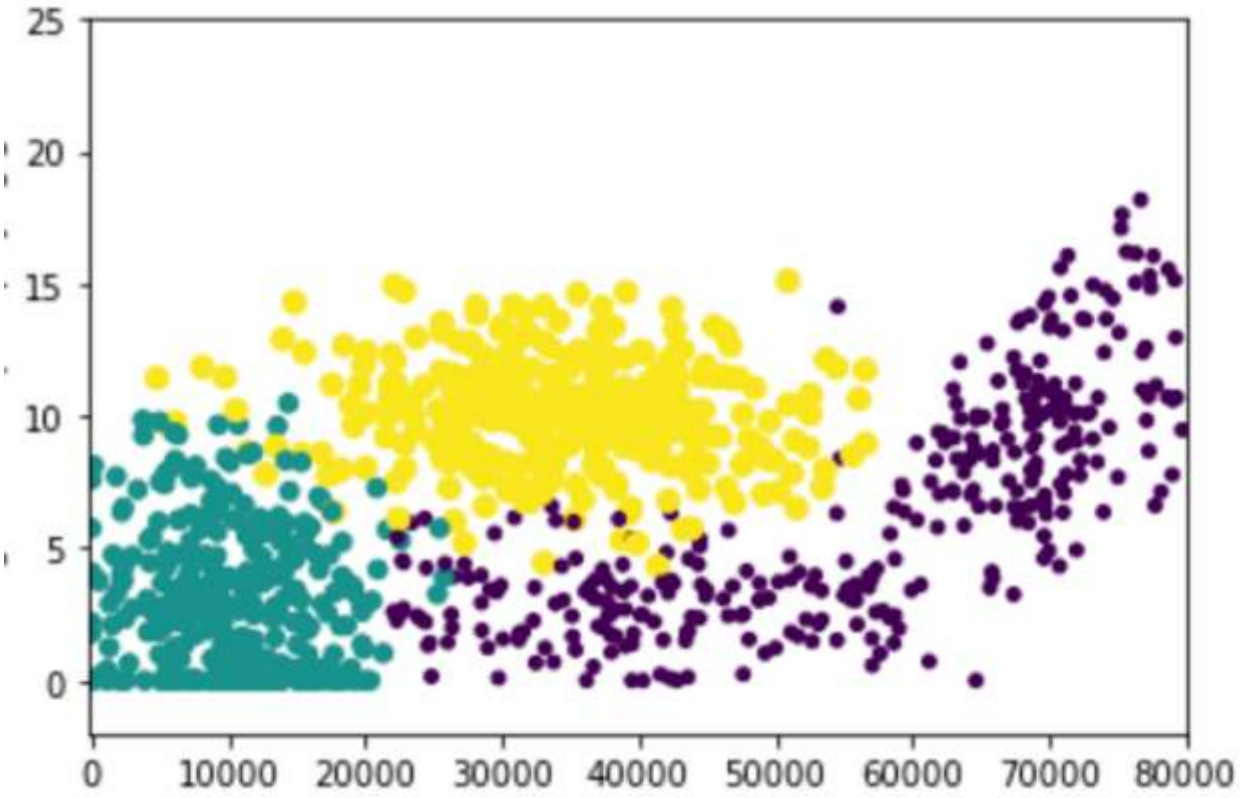
观察一下，黄点和蓝点代表了两种标签，比如每个蓝点都是一个合格的产品，黄点是劣质的产品。事实上在图中可以看到，相同标记的样本点通常是成团的形式聚在一起，因为合格的产品在属性上一定是相同或相似的（合格的产品在属性上不太可能会跑到不合格的一类中去）。

那么我们预测过程中，查看被预测的样本x是属于哪一堆来判断它是黄豆还是蓝豆是不是可行呢？

当然可以啦，K近邻就是一种基于该原理的算法。从名字里就可以看到，K近邻样本的预测上，是看被预测样本x离哪一团最近，那它就是属于哪一类的。

接下来我们正儿八经地来看一下K近邻，看下面这张图。

(http://www.pkudodo.com/wp-content/uploads/2018/11/K近邻_样本图.png)



图里有两种标记，就叫它黄豆、绿豆、紫豆好了（这里也能看到，K近邻不像感知机只能划分两种类，K近邻是一种多类划分的模型）。当我们要预测一个样本x时，将x的特征转换为在图中的坐标点，分别计算和绿豆、黄豆、紫豆的举例，比如说距离分别为1， 1.5， 0.8。选择其中距离最小的紫豆作为样本x的预测类别。

那啥是样本x和一团豆的距离?

1.找到样本x最近的点，该点的类就是样本的预测类：这是一种方法，但是如果有噪音呢（同一块区域又有黄点又有绿点）？比如说x实际上是黄豆，但是它的位置在黄豆和绿豆的边界上（例如上图黄点和绿点的交叉处），很可能它最近的点是一个绿点，所以....不太好

2.与每一团的中心点进行距离计算：分别计算绿色、黄色、紫色的中心点，判断距离最小的类即为预测输出类。这样会不会有问题吗？我们看一下上图中绿色和紫色交叉的地方，很明显在这个交叉位置离绿色很近，与紫色中心点较远，但实际上紫色的。所以.....不太好

3.找到样本点周围的K个点，其中占数目最多的类即预测输出的类：克服了前两种方法的弊端，实际上这就是K近邻所使用的算法

感知机的数学角度（配合《统计学习方法》食用更佳）

算法剖析

(http://www.pkudodo.com/wp-content/uploads/2018/11/K邻近_算法.png)

算法 3.1（k 近邻法）

输入：训练数据集

$$T=\{(x_1,y_1),(x_2,y_2),\cdots,(x_N,y_N)\}$$

其中，$x_i\in\mathcal{X}\subseteq\mathbf{R}^n$ 为实例的特征向量，$y_i\in\mathcal{Y}=\{c_1,c_2,\cdots,c_K\}$ 为实例的类别，$i=1,2,\cdots,N$；实例特征向量 x；

输出：实例 x 所属的类 y.

（1）根据给定的距离度量，在训练集 T 中找出与 x 最邻近的 k 个点，涵盖这 k 个点的 x 的邻域记作 $N_k(x)$；

（2）在 $N_k(x)$ 中根据分类决策规则（如多数表决）决定 x 的类别 y：

$$y=\arg\max_{c_j}\sum_{x_i\in N_k(x)}I(y_i=c_j),\quad i=1,2,\cdots,N;\quad j=1,2,\cdots,K \tag{3.1}$$

式 (3.1) 中，I 为指示函数，即当 $y_i=c_j$ 时 I 为 1，否则 I 为 0. ■

k 近邻法的特殊情况是 $k=1$ 的情形，称为最近邻算法. 对于输入的实例点（特征向量）x，最近邻法将训练数据集中与 x 最邻近点的类作为 x 的类.

K近邻并没有显式的学习过程，也就是不需要对训练集进行学习。预测过程中直接遍历预测点与所有点的距离，并找到最近的K个点即可。找到K个最近点后，使用多数表决（即投票）的方式确定预测点的类别。式3.1I(yi=ci)中的I为指示函数，当括号内条件为真时I(true)=1，I(false)=0。argmax表示令后式数值最大时的参数，例如argmax(-X^2 + 1)的结果为0，因为x=0时-X^2 + 1结果为1，为最大值。

式3.1表示对于每一个类Cj，进行I(yi=cj)进行求和，就是计算这K个点中有多少个标记为Cj的点，例如K=25，一共有四个类分别为C1、C2、C3、C4，25个点中他们的个数分别有10、5、1、9个，那么最多数目的类别C1就是样本点的预测类别。

距离度量

设特征空间 $\mathcal{X}$ 是 n 维实数向量空间 $\mathbf{R}^n$，$x_i, x_j \in \mathcal{X}$，$x_i = (x_i^{(1)}, x_i^{(2)}, \cdots, x_i^{(n)})^{\mathrm{T}}$，$x_j = (x_j^{(1)}, x_j^{(2)}, \cdots, x_j^{(n)})^{\mathrm{T}}$，$x_i, x_j$ 的 L_p 距离定义为

$$L_p(x_i, x_j) = \left(\sum_{l=1}^{n} | x_i^{(l)} - x_j^{(l)} |^p \right)^{\frac{1}{p}} \tag{3.2}$$

这里 $p \geqslant 1$．当 $p = 2$ 时，称为欧氏距离(Euclidean distance)，即

$$L_2(x_i, x_j) = \left(\sum_{l=1}^{n} | x_i^{(l)} - x_j^{(l)} |^2 \right)^{\frac{1}{2}} \tag{3.3}$$

当 $p = 1$ 时，称为曼哈顿距离（Manhattan distance），即

$$L_1(x_i, x_j) = \sum_{l=1}^{n} | x_i^{(l)} - x_j^{(l)} | \tag{3.4}$$

当 $p = \infty$ 时，它是各个坐标距离的最大值，即

$$L_\infty(x_i, x_j) = \max_l | x_i^{(l)} - x_j^{(l)} | \tag{3.5}$$

(http://www.pkudodo.com/wp-content/uploads/2018/11/K近邻_各种距离计算公式.png)

在多维空间中有很多种方式可以计算点与点之间的举例，通常采用欧氏距离作为K近邻的度量单位（大部分模型中欧氏距离都是一种不错的选择）。其实就是样本A、B中每一个特征都互相相减，再平方、再求和。与二维中两点之间距离计算方式相同，只是扩展到了多维。

曼哈顿与P=无穷可以不用深究，在本文中使用曼哈顿准确度极差（仅针对Mnist数据集使用K近邻的情况），这两种方式目前仅作了解即可。

K近邻算法缺点：

1、在预测样本类别时，待预测样本需要与训练集中所有样本计算距离，当训练集数量过高时（例如Mnsit训练集有60000个样本），每预测一个样本都要计算60000个距离，计算代价过高，尤其当测试集数目也较大时（Mnist测试集有10000个）。

1、K近邻在高维情况下时（高维在机器学习中并不少见），待预测样本需要与依次与所有样本求距离。向量维度过高时使得欧式距离的计算变得不太迅速了。本文在60000训练集的情况下，将10000个测试集缩减为200个，整个过程仍然需要308秒（曼哈顿距离为246秒，但准确度大幅下降）。

使用欧氏距离还是曼哈顿距离，性能上的差别相对来说不是很大，说明欧式距离并不是制约计算速度的主要方式。最主要的是训练集的大小，每次预测都需要与60000个样本进行比对，同时选出距离最近的K项。

为了解决这一问题，前人提出了KD树算法。

KD树

KD树将整个特征空间划分成多个区域，直观上来看，首先将整个空间分成A、B区域，待测样本判断在A区的时候，那B区过远，内部的点就不需要再判断了，大幅度减少需要比较的样本数量。

但遗憾的是作者暂时没有对KD树进行实现，仅仅在理论层面上讲解并不是我所希望的。所以各位同学请耐心等待，等我将其实现后，写出来的内容可能会更有深度一些。

贴代码：（建议去本文最上方的github链接下载，有书中所有算法的实现以及详细注释）

```
#coding=utf-8
#Author:Dodo
#Date:2018-11-16
#Email:lvtengchao@pku.edu.cn

'''
数据集：Mnist
训练集数量：60000
测试集数量：10000（实际使用：200）
------------------------------
运行结果：（邻近k数量：25）
向量距离使用算法—欧式距离
    正确率：97%
    运行时长：308s
向量距离使用算法—曼哈顿距离
    正确率：14%
```

```
    运行时长：246s
'''

import numpy as np
import time

def loadData(fileName):
    '''
    加载文件
    :param fileName:要加载的文件路径
    :return: 数据集和标签集
    '''
    print('start read file')
    #存放数据及标记
    dataArr = []; labelArr = []
    #读取文件
    fr = open(fileName)
    #遍历文件中的每一行
    for line in fr.readlines():
        #获取当前行，并按“，”切割成字段放入列表中
        #strip：去掉每行字符串首尾指定的字符（默认空格或换行符）
        #split：按照指定的字符将字符串切割成每个字段，返回列表形式
        curLine = line.strip().split(',')
        #将每行中除标记外的数据放入数据集中（curLine[0]为标记信息）
        #在放入的同时将原先字符串形式的数据转换为整型
        dataArr.append([int(num) for num in curLine[1:]])
        #将标记信息放入标记集中
        #放入的同时将标记转换为整型
        labelArr.append(int(curLine[0]))
    #返回数据集和标记
    return dataArr, labelArr

def calcDist(x1, x2):
    '''
    计算两个样本点向量之间的距离
    使用的是欧氏距离，即 样本点每个元素相减的平方  再求和  再开方
    欧式举例公式这里不方便写，可以百度或谷歌欧式距离（也称欧几里得距离）
    :param x1:向量1
    :param x2:向量2
    :return:向量之间的欧式距离
    '''
    return np.sqrt(np.sum(np.square(x1 - x2)))

    #马哈顿距离计算公式
    # return np.sum(x1 - x2)
```

```
65.
66. def getClosest(trainDataMat, trainLabelMat, x, topK):
67.     '''
68.     预测样本x的标记。
69.     获取方式通过找到与样本x最近的topK个点，并查看它们的标签。
70.     查找里面占某类标签最多的那类标签
71.     （书中3.1 3.2节）
72.     :param trainDataMat:训练集数据集
73.     :param trainLabelMat:训练集标签集
74.     :param x:要预测的样本x
75.     :param topK:选择参考最邻近样本的数目（样本数目的选择关系到正确率，详看3.2.3 K值的选择）
76.     :return:预测的标记
77.     '''
78.     #建立一个存放向量x与每个训练集中样本距离的列表
79.     #列表的长度为训练集的长度，distList[i]表示x与训练集中第
80.     ## i个样本的距离
81.     distList = [0] * len(trainLabelMat)
82.     #遍历训练集中所有的样本点，计算与x的距离
83.     for i in range(len(trainDataMat)):
84.         #获取训练集中当前样本的向量
85.         x1 = trainDataMat[i]
86.         #计算向量x与训练集样本x的距离
87.         curDist = calcDist(x1, x)
88.         #将距离放入对应的列表位置中
89.         distList[i] = curDist
90.
91.     #对距离列表进行排序
92.     #argsort：函数将数组的值从小到大排序后，并按照其相对应的索引值输出
93.     #例如：
94.     #   >>> x = np.array([3, 1, 2])
95.     #   >>> np.argsort(x)
96.     #   array([1, 2, 0])
97.     #返回的是列表中从小到大的元素索引值，对于我们这种需要查找最小距离的情况来说很合适
98.     #array返回的是整个索引值列表，我们通过[:topK]取列表中前topL个放入list中。
99.     #-----------------优化点-------------------
100.     #由于我们只取topK小的元素索引值，所以其实不需要对整个列表进行排序，而argsort是对整个
101.     #列表进行排序的，存在时间上的浪费。字典有现成的方法可以只排序top大或top小，可以自行查阅
102.     #对代码进行稍稍修改即可
103.     #这里没有对其进行优化主要原因是KNN的时间耗费大头在计算向量与向量之间的距离上，由于向量高维
104.     #所以计算时间需要很长，所以如果要提升时间，在这里优化的意义不大。（当然不是说就可以不优化了，
105.     #主要是我太懒了）
```

```
    topKList = np.argsort(np.array(distList))[:topK]        #升序排序
    #建立一个长度时的列表，用于选择数量最多的标记
    #3.2.4提到了分类决策使用的是投票表决，topK个标记每人有一票，在数组中每个标记代表的位置中投入
    #自己对应的地方，随后进行唱票选择最高票的标记
    labelList = [0] * 10
    #对topK个索引进行遍历
    for index in topKList:
        #trainLabelMat[index]：在训练集标签中寻找topK元素索引对应的标记
        #int(trainLabelMat[index])：将标记转换为int（实际上已经是int了，但是不int的话，报错）
        #labelList[int(trainLabelMat[index])]：找到标记在labelList中对应的位置
        #最后加1，表示投了一票
        labelList[int(trainLabelMat[index])] += 1
    #max(labelList)：找到选票箱中票数最多的票数值
    #labelList.index(max(labelList))：再根据最大值在列表中找到该值对应的索引，等同于预测的标记
    return labelList.index(max(labelList))


def test(trainDataArr, trainLabelArr, testDataArr, testLabelArr, topK):
    '''
    测试正确率
    :param trainDataArr:训练集数据集
    :param trainLabelArr: 训练集标记
    :param testDataArr: 测试集数据集
    :param testLabelArr: 测试集标记
    :param topK: 选择多少个邻近点参考
    :return: 正确率
    '''
    print('start test')
    #将所有列表转换为矩阵形式，方便运算
    trainDataMat = np.mat(trainDataArr); trainLabelMat = np.mat(trainLabelArr).T
    testDataMat = np.mat(testDataArr); testLabelMat = np.mat(testLabelArr).T

    #错误值技术
    errorCnt = 0
    #遍历测试集，对每个测试集样本进行测试
    #由于计算向量与向量之间的时间耗费太大，测试集有6000个样本，所以这里人为改成了
    #测试200个样本点，如果要全跑，将行注释取消，再下一行for注释即可，同时下面的print
    #和return也要相应的更换注释行
```

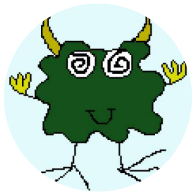
```
    # for i in range(len(testDataMat)):
    for i in range(200):
        # print('test %d:%d'%(i, len(trainDataArr)))
        print('test %d:%d' % (i, 200))
        #读取测试集当前测试样本的向量
        x = testDataMat[i]
        #获取预测的标记
        y = getClosest(trainDataMat, trainLabelMat, x, topK)
        #如果预测标记与实际标记不符，错误值计数加1
        if y != testLabelMat[i]: errorCnt += 1

    #返回正确率
    # return 1 - (errorCnt / len(testDataMat))
    return 1 - (errorCnt / 200)


if __name__ == "__main__":
    start = time.time()

    #获取训练集
    trainDataArr, trainLabelArr =
loadData('../Mnist/mnist_train.csv')
    #获取测试集
    testDataArr, testLabelArr =
loadData('../Mnist/mnist_test.csv')
    #计算测试集正确率
    accur = test(trainDataArr, trainLabelArr, testDataArr,
testLabelArr, 25)
    #打印正确率
    print('accur is:%d'%(accur * 100), '%')

    end = time.time()
    #显示花费时间
print('time span:', end - start)
```



Dodo

26条评论

匿名 **发布于9:32 上午 - 1月 28, 2023**

()

To the pkudodo.com administrator, You always provide great resources and references.

回复

匿名 **发布于9:22 上午 - 1月 10, 2023**

()

I do imagine the many principles you may have released on your own write-up. They're seriously convincing and may undoubtedly do the job. Still, the posts are far too quick for newcomers. May possibly you remember to extend them a bit from up coming time? Thanks for that put up

回复

匿名 **发布于8:34 上午 - 1月 10, 2023**

()

A lot of thanks for every of your respective exertions on this Online page. Kim really likes entering into investigation and it's really easy to realize why. Many of us know most of the dynamic method you make a must have measures on this World wide web weblog and and enhance participation from men and women on that idea so our Lady is unquestionably Discovering loads of matters. Have a great time With all the remaining percentage of The brand new yr. You're conducting a reasonably great work.

回复

匿名 **发布于8:02 上午 - 1月 10, 2023**

()

Your mode of describing anything In this particular paragraph is absolutely pleasant, Each one can effortlessly pay attention to it, Many thanks a whole lot. asla.munhea.se/map10.php mat och h?¤lsa

回复

()

匿名 **发布于9:08 上午 - 11月 30, 2022**

It’s hard to occur by proficient men and women Within this certain matter, however, you seem like you know what you’re referring to! Many thanks orof.infoforwomen.be/map22.php lediga jobb bor??s

回复

()

匿名 **发布于9:03 上午 - 9月 7, 2022**

bunadisisj nsjjsjsisjsmizjzjjzjzz zumzsert

回复

()

匿名 **发布于3:40 下午 - 12月 6, 2021**

大佬太强了，瑞思拜

回复

()

匿名 **发布于3:00 下午 - 6月 28, 2021**

作者大大写的很好，不知道会不会补上kd树的代码

回复

()

匿名 **发布于3:31 下午 - 5月 16, 2021**

可以可以，最近打算花一个半月捡起来统计学习，几乎有一年没有好好看算法了

回复

匿名 **发布于2:10 下午 - 5月 16, 2021**

()

感谢大佬!

回复

()

匿名 **发布于1:10 下午 - 10月 30, 2020**

补充一下:
1. 已经有人提到了, 可以用 pandas 更快录入数据
2. 已经有人提到了, 曼哈顿距离加上 np.abs() 取绝对值
3. 加上绝对值后, 测试集的准确率是: 95.5%

回复

()

匿名 **发布于5:34 下午 - 1月 21, 2021**

我用pandas导入数据总体运行时间多了150s

回复

()

匿名 **发布于10:17 下午 - 9月 28, 2020**

大佬良心解释，太详细了，几乎不借助第三方库手撕代码，佩服!

回复

()

匿名 **发布于11:32 下午 - 8月 20, 2020**

建议loadData函数这样写，更python,借助pandas:

```
def loadData(filename):

print(“start to read”)

df = pd.read_csv(filename, header=None)
dataArr = df.iloc[:,1:]
labelArr= df.iloc[:,0]
```

return dataArr, labelArr

回复

匿名 **发布于7:57 下午 - 6月 16, 2020**

()

真的太厉害了!

回复

匿名 **发布于3:59 下午 - 5月 20, 2020**

()

KD树有代码啦!
github.com/wenffe/python-KD-

回复

匿名 **发布于10:09 上午 - 5月 18, 2020**

()

太棒了！！！ 初学者之光！！！

回复

匿名 **发布于5:52 下午 - 1月 29, 2020**

()

赞哦,谢谢你给初学者领路

回复

匿名 **发布于10:04 下午 - 11月 26, 2019**

()

代码简直是全网最强注释，基本上认真看都能看懂，非常详细了！！！！！ 楼主辛苦了，向楼主学习！一天学习一个算法，充实!

回复

匿名 **发布于2:28 下午 - 8月 18, 2019**

受教了

()

回复

匿名 **发布于8:12 下午 - 7月 9, 2019**

是大佬

()

回复

匿名 **发布于10:17 上午 - 7月 1, 2019**

博主，曼哈顿距离精度不高是因为算法公式有点问题,np.sum(np.abs(x1 – x2))

()

回复

匿名 **发布于3:15 下午 - 5月 21, 2020**

是的，没有漏了绝对值

()

回复

匿名 **发布于4:23 下午 - 2月 26, 2019**

厉害厉害，跟随大佬脚步

()

回复

匿名 **发布于4:04 下午 - 1月 3, 2019**

赞一个哈，非常用心

()

回复

匿名 **发布于10:33 上午 - 11月 20, 2018**

???

()

回复

发表评论

内容(链接请去除http协议头，否则会被误判为垃圾评论)

设置显示昵称(选填)

发送信息

近期文章

- 论文|记忆网络之Memory Networks (http://www.pkudodo.com/2019/06/14/1-13/)
- 梳理|对话系统中的DST (http://www.pkudodo.com/2019/06/09/1-12/)
- 论文阅读|How Does Batch Normalizetion Help Optimization (http://www.pkudodo.com/2019/05/23/1-11/)
- 学习规划|机器学习和NLP入门规划 (http://www.pkudodo.com/2019/03/20/1-10/)
- 面试体会|微软、头条、滴滴、爱奇艺NLP面试感想 (http://www.pkudodo.com/2019/03/10/1-9/)

文章归档

- 2019年6月 (http://www.pkudodo.com/2019/06/)
- 2019年5月 (http://www.pkudodo.com/2019/05/)
- 2019年3月 (http://www.pkudodo.com/2019/03/)
- 2018年12月 (http://www.pkudodo.com/2018/12/)
- 2018年11月 (http://www.pkudodo.com/2018/11/)
- 2018年10月 (http://www.pkudodo.com/2018/10/)
- 2016年9月 (http://www.pkudodo.com/2016/09/)

标签

DST (http://www.pkudodo.com/tag/dst/)

K近邻 (http://www.pkudodo.com/tag/k%e8%bf%91%e9%82%bb/)

LSI (http://www.pkudodo.com/tag/lsi/)

memory network (http://www.pkudodo.com/tag/memory-network/)

MQTT (http://www.pkudodo.com/tag/mqtt/) NLP (http://www.pkudodo.com/tag/nlp/)

s3c2440 (http://www.pkudodo.com/tag/s3c2440/) SLU (http://www.pkudodo.com/tag/slu/)

SVM (http://www.pkudodo.com/tag/svm/) VSM (http://www.pkudodo.com/tag/vsm/)

体会 (http://www.pkudodo.com/tag/%e4%bd%93%e4%bc%9a/)

决策树 (http://www.pkudodo.com/tag/%e5%86%b3%e7%ad%96%e6%a0%91/)

分类器 (http://www.pkudodo.com/tag/%e5%88%86%e7%b1%bb%e5%99%a8/)

实现 (http://www.pkudodo.com/tag/%e5%ae%9e%e7%8e%b0/)

对话系统 (http://www.pkudodo.com/tag/%e5%af%b9%e8%af%9d%e7%b3%bb%e7%bb%9f/)

感想 (http://www.pkudodo.com/tag/%e6%84%9f%e6%83%b3/)

感知机 (http://www.pkudodo.com/tag/%e6%84%9f%e7%9f%a5%e6%9c%ba/)

支持向量机 (http://www.pkudodo.com/tag/%e6%94%af%e6%8c%81%e5%90%91%e9%87%8f%e6%9c%ba/)

文章相似度 (http://www.pkudodo.com/tag/%e6%96%87%e7%ab%a0%e7%9b%b8%e4%bc%bc%e5%ba%a6/)

最大熵 (http://www.pkudodo.com/tag/%e6%9c%80%e5%a4%a7%e7%86%b5/)

朴素贝叶斯 (http://www.pkudodo.com/tag/%e6%9c%b4%e7%b4%a0%e8%b4%9d%e5%8f%b6%e6%96%af/)

机器学习 (http://www.pkudodo.com/tag/%e6%9c%ba%e5%99%a8%e5%ad%a6%e4%b9%a0/)

爬虫 (http://www.pkudodo.com/tag/%e7%88%ac%e8%99%ab/)

统计学习方法 (http://www.pkudodo.com/tag/%e7%bb%9f%e8%ae%a1%e5%ad%a6%e4%b9%a0%e6%96%b9%e

统计方法学习 (http://www.pkudodo.com/tag/%e7%bb%9f%e8%ae%a1%e6%96%b9%e6%b3%95%e5%ad%a6%e

综述 (http://www.pkudodo.com/tag/%e7%bb%bc%e8%bf%b0/)

网易云歌单 (http://www.pkudodo.com/tag/%e7%bd%91%e6%98%93%e4%ba%91%e6%ad%8c%e5%8d%95/)

规划 (http://www.pkudodo.com/tag/%e8%a7%84%e5%88%92/)

详解 (http://www.pkudodo.com/tag/%e8%af%a6%e8%a7%a3/)

逻辑回归 (http://www.pkudodo.com/tag/%e9%80%bb%e8%be%91%e5%9b%9e%e5%bd%92/)

逻辑斯蒂回归 (http://www.pkudodo.com/tag/%e9%80%bb%e8%be%91%e6%96%af%e8%92%82%e5%9b%9e%e

面试 (http://www.pkudodo.com/tag/%e9%9d%a2%e8%af%95/)