# K近邻算法

1. 模型概述
2. 概念描述

K近邻算法是一种基本分类和回归方法。当对数据的分布只有很少或者没有任何先验知识时，K 近邻算法是一个不错的选择。K近邻算法有三要素：k 值的选择、距离度量和分类决策规则

K近邻算法，即是给定一个训练数据集，对新的输入实例，在训练数据集中找到与该实例最邻近的K个实例，这K个实例的多数属于某个类，就把该输入实例分类到这个类中。（这就类似于现实生活中少数服从多数的思想）

1. 公式描述

|  |  |  |  |
| --- | --- | --- | --- |
| **数据集** | = | = | y= |
| **数据归一化** | = | | |
| **欧氏距离** |  | | |

1. 重要代码详解

数据归一化处理：ranges=;np.tile()表示扩容矩阵，这里表示将扩容为和data一样的形状，对应元素相减，减少运算。

minValue=data.min()

maxValue=data.max()

ranges=maxValue-minValue

data=(data-np.tile(minValue,(np.matrix(data).shape[0],1)))/np.tile(ranges,(np.matrix(data).shape[0],1))

欧氏距离：给出两个点，然后求出欧氏距离。

def distance(p1, p2):

    return np.sqrt(((p1[0]-p2[0])\*\*2)+((p1[1]-p2[1])\*\*2))

KNN主算法：给出K表示最近K个点。首先求出所有与其中一点的距离，然后按照值的大小对其索引进行升序排列，选出其中K个索引，然后比对这K个点中0和1标签哪个多，哪个多则该点就属于哪个标签。

def KNN(K,x\_train,y\_train,x\_test,y\_test):

    predict=[]

    for item\_test in x\_test:

        dist=[]

        for item\_train in x\_train:

            dist.append(distance(item\_test,item\_train))

        #对其索引进行升序排列

        nn\_index=np.argsort(dist)

        #求出K个最近点的索引值

        nn\_y=[]

        for i in nn\_index[:K]:

            nn\_y.append(y\_train[i])

        if list(nn\_y).count(0)>list(nn\_y).count(1):

            predict.append(0)

        else:

            predict.append(1)

    return predict

准确率函数：将根据KNN求得的标签与真实标签比对，计算准确率。

def prediction(predict,y\_test):

    correct=0

    for i in range(len(predict)):

        if predict[i] == y\_test[i]:

            correct+=1

    return correct/len(y\_test)

1. 结果分析e

我这里每隔四个数据点选取一个测试点，K选取为5，最后正确率为0.86。

我又选取了iris数据集来计算KNN,选取K为3，同样是每隔四个数据点选取一个测试点，正确率为0.76。

可见我自己写的KNN正确率有限，可待后期优化。

四.源码

from sklearn.datasets import load\_iris

import numpy as np

import pandas as pd

iris=load\_iris()

data = pd.DataFrame(iris.data, columns=iris.feature\_names)

minValue=data.min()

maxValue=data.max()

ranges=maxValue-minValue

data=(data-np.tile(minValue,(np.matrix(data).shape[0],1)))/np.tile(ranges,(np.matrix(data).shape[0],1))

data['target'] = iris.target

train=[]

test=[]

num=1

for item in np.array(data):

    if num==4:

        test.append(item)

        num=1

    else:

        train.append(item)

        num+=1

cols = pd.DataFrame(train).shape[1]

x\_train=np.array(pd.DataFrame(train).iloc[:,0:cols-1])

y\_train=np.array(pd.DataFrame(train).iloc[:,cols-1:cols])

cols = pd.DataFrame(test).shape[1]

x\_test=np.array(pd.DataFrame(test).iloc[:,0:cols-1])

y\_test=np.array(pd.DataFrame(test).iloc[:,cols-1:cols])

def distance(p1, p2):

    return np.sqrt(((p1[0]-p2[0])\*\*2)+((p1[1]-p2[1])\*\*2))

def KNN(K,x\_train,y\_train,x\_test,y\_test):

    predict=[]

    for item\_test in x\_test:

        dist=[]

        for item\_train in x\_train:

            dist.append(distance(item\_test,item\_train))

        #对其索引进行升序排列

        nn\_index=np.argsort(dist)

        #求出K个最近点的索引值

        nn\_y=[]

        for i in nn\_index[:K]:

            nn\_y.append(y\_train[i])

        if list(nn\_y).count(0)>list(nn\_y).count(1) and list(nn\_y).count(0)>list(nn\_y).count(2):

            predict.append(0)

        elif list(nn\_y).count(1)>list(nn\_y).count(0) and list(nn\_y).count(1)>list(nn\_y).count(2):

            predict.append(1)

        else:

            predict.append(2)

    return predict

def prediction(predict,y\_test):

    correct=0

    for i in range(len(predict)):

        if predict[i] == y\_test[i]:

            correct+=1

    return correct/len(y\_test)

predict=KNN(3,x\_train,y\_train,x\_test,y\_test)

correct\_rate=prediction(predict,y\_test)

print("鸢尾花分类准确率为：%.2f"%correct\_rate)