Table of Contents

1 什么是KNN?——>既可以分类也可以回归

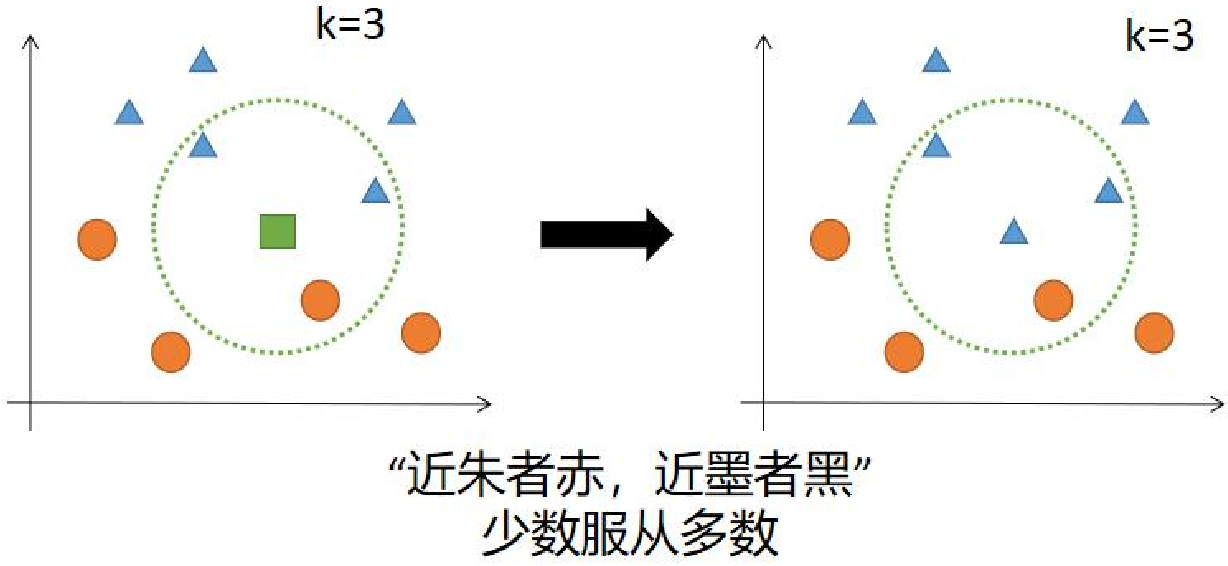
- ## 1.1 简单定义： K Nearest Neighbors，K 个最近的邻居

- ## 1.2 关键：

  (1) 近朱者赤，近墨者黑：根据样本近邻属性对其进行预测

  (2) 少数服从多数

- ## 1.3 k的含义？离某点距离最近的k个点。



2 问题：

- 已知 $Data=\{(x_i,y_i),i=1,2,..,N\}$，$x_i \in R^n$， $y_i \in \{c_1,c_2,..c_k\}$为分成的类别数，对于新的实例 $x \in R^n$，如何确定 y？

3 过程：

- (1)计算距离 $d_i=d(x_i,x)$
- (2)与近邻的k个点的距离排序
- (3)多数表决：多数类来决定结果

4 最近距离，那距离的定义?

4.1 闵可夫斯基（Minkowski）距离 （L_p距离）

- 设特征空间 χ 是 n 维实数向量空间 R^n, x_i,$x_j \in \chi$, $x_i=(x_i^{(1)},x_i^{(2)},...,x_i^{(n)})^T$ ， $x_j=(x_j^{(1)},x_j^{(2)},...,x_j^{(n)})^T$, x_i,x_j的 L_p 距离定义为

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

4.2 曼哈顿（Manhattan）距离（L_1距离）

- 当 $p=1$ 时的 L_1 距离定义为

$$L_1\left(x_i,x_j\right)=\sum_{l=1}^{n} \mid x_i^{(l)}-x_j^{(l)}|$$

4.3 欧式距离（Eucliden）距离（L_2距离）

- 当 $p=2$ 时的 L_2 距离定义为

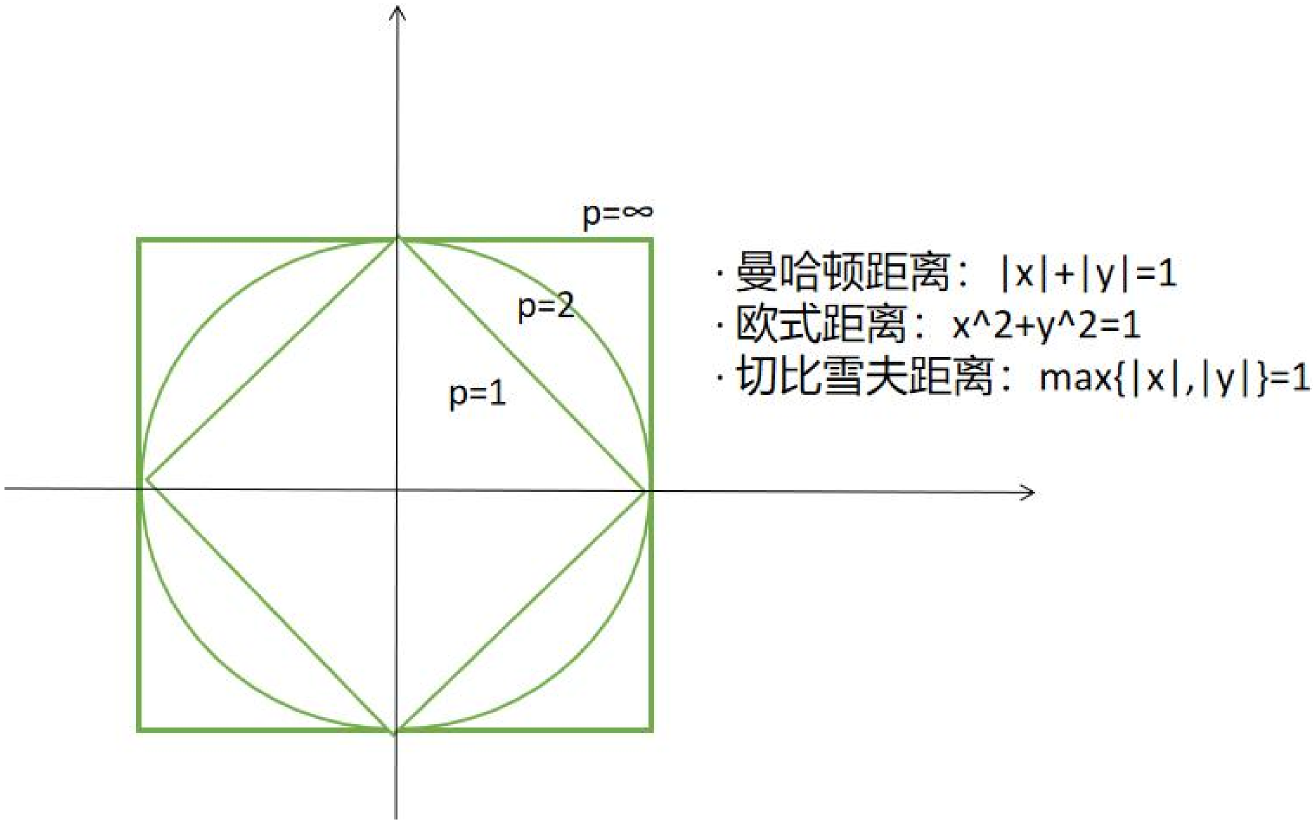
$$L_2\left(x_i, x_j\right)=\left(\sum_{l=1}^{n} \mid x_i^{(l)}-x_j^{(l)}|^2\right)^{\frac{1}{2}}$$

4.4 切比雪夫（Chebyshev）距离（L_∞距离）

- 当 $p=\infty$ 时的 L_∞ 距离定义为 ——> **各坐标距离的最大值**

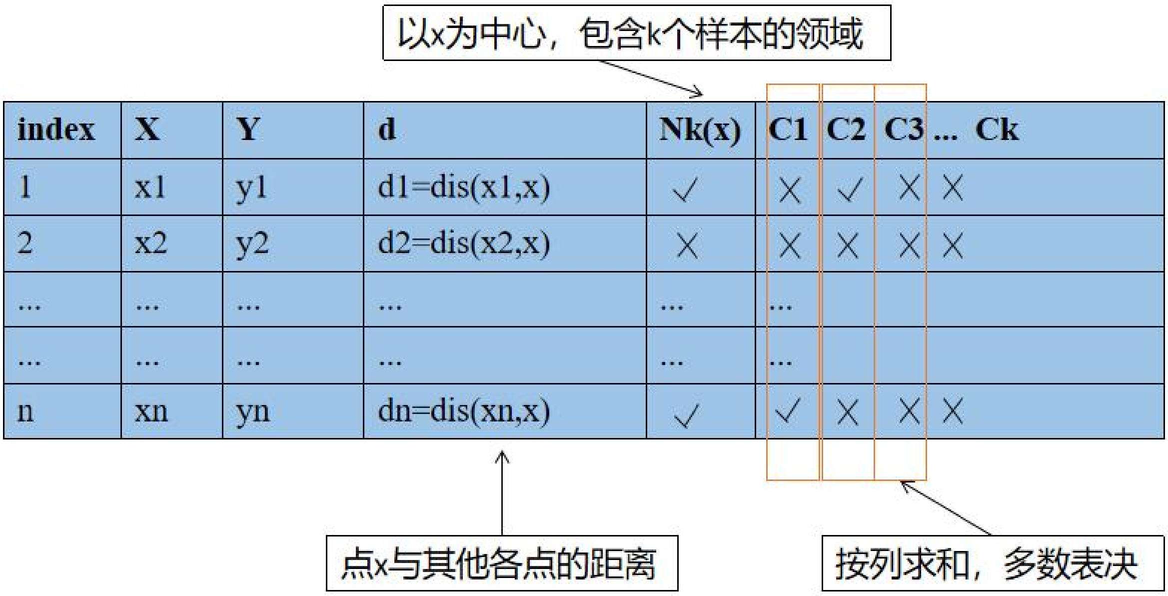
$$L_\infty\left(x_i, x_j\right)=\max_l \mid x_i^{(l)}-x_j^{(l)}\mid$$

4.5 在二维空间中 p 取不同值时，与原点的 L_P 距离为1的点的图形：



5 模型 model

- ### 5.1 抽象图解



- **5.2 具体模型：**

  给定一个训练集：$T=\{(x_1,y_1),(x_2,y_2),\ldots,(x_n,y_n)\}$，其中$x_i$为实例的特征向量，$y_i\in y=\{c_1,c_2,\ldots,c_k\}$，输入实例特征向量$x$，输出该实例特征向量的类别$y$

  $$y=arg\max_{c_j}\sum_{x_i\in N_k(x)}I(y_i=c_j),i=1,2...,N,j=1,2,\ldots,K$$

  其中 I 为指示函数，即当 $y_i=c_j$ 时 I 为 1，否则 I 为 0

6 策略 strategy

$$y=arg\max_{c_j}\sum_{x_i\in N_k(x)}I(y_i=c_j),i=1,2...,N,j=1,2,\ldots,K$$

- KNN中的决策规则通常就是“投票选举”——**少数服从多数** 的方式
- 如果损失函数是0-1损失函数，那么分类函数就是：

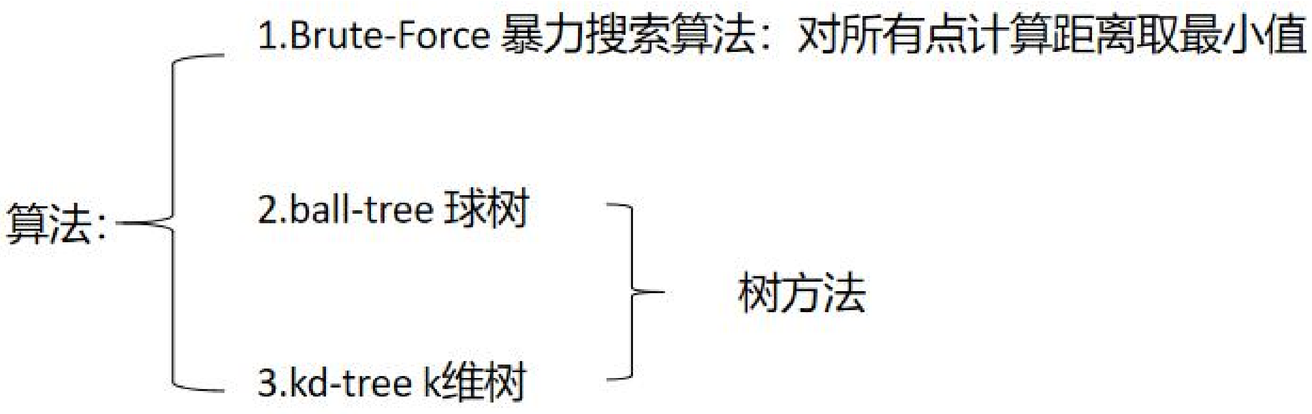
$$f:R^n\rightarrow c_1,c_2,\ldots,c_k$$

- 对于相邻k个训练实例点构成集合N，误分类率是：

$$\frac{1}{k}\sum_{x_i\in N_k(x)}I(y_i\neq c_j)=1-\frac{1}{k}\sum_{x_i\in N_k(x)}I(y_i=c_j)$$

- 要使误分类率最小，那么就是要求正确的概率最大，所以少数服从多数的规则正好可以满足**经验风险**最小化。

7 算法 algorithm



==

输入: 给定一个训练集：$T=\{(x_1,y_1),(x_2,y_2),\ldots,(x_n,y_n)\}$，其中$x_i$为实例的特征向量，$y_i\in y=\{c_1,c_2,\ldots,c_k\}$为实例的类别，$i=1,2,\ldots,N$

输出:实例 x 所属的类别 y

(1) 给定距离度量，在训练集T 中找出与 x 最近邻的 k 个点，涵盖这 k 个点的 x的邻域记作 $N_k(x)$

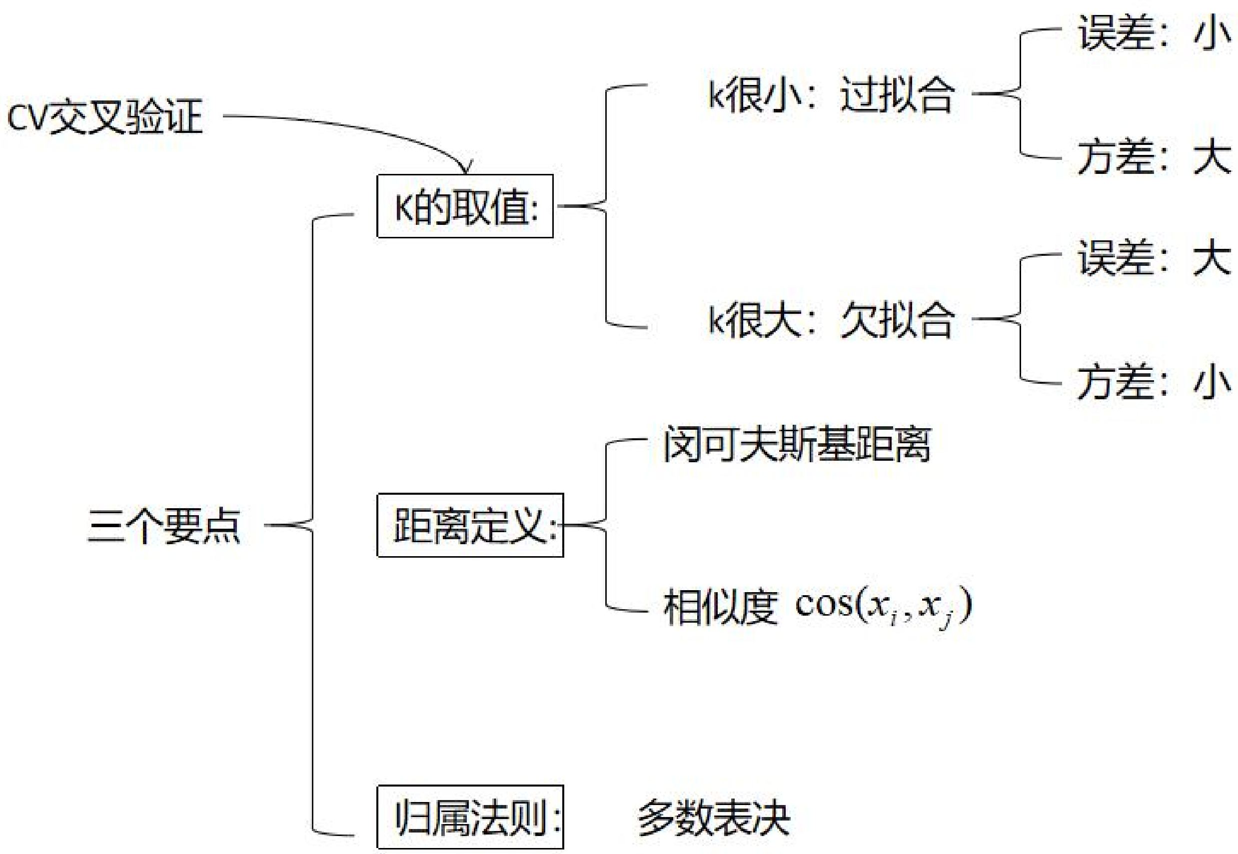
(2) 在 $N_k(x)$ 中根据分类决策规则（eg.多数表决）决定 x 的类别 y

$$y=arg\max_{c_j}\sum_{x_i\in N_k(x)}I(y_i=c_j), i=1,2...,N, j=1,2,\ldots,K$$

其中 I 为指示函数，即当 $y_i=c_j$ 时 I 为 1，否则 I 为 0

==

8 K-NN的三个要点



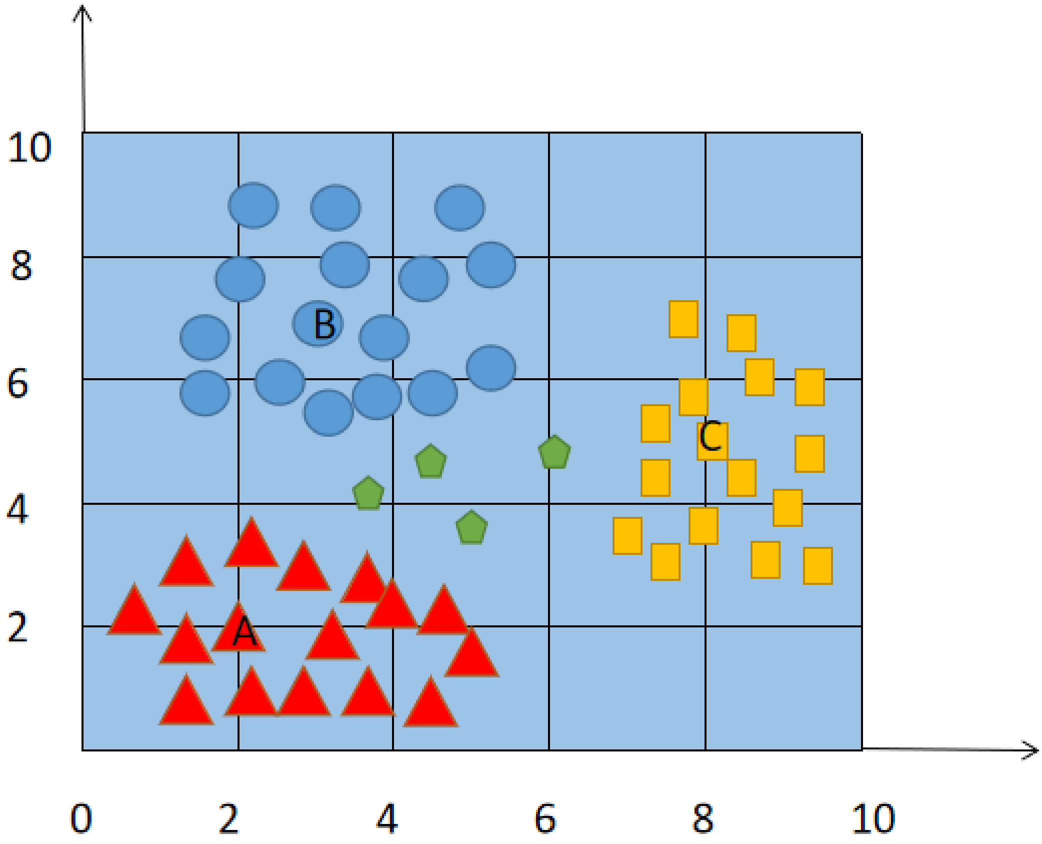
9 K-NN的回归

- 已知 $Data = \{(x_i, y_i), i = 1, 2, .., N\}$，$x_i \in R^n$， $y_i \in \{c_1, c_2, .. c_k\}$

$$y = \frac{1}{k} \sum_{x_i \in N_k(x)} I(y_i = c_j), i = 1, 2..., N, j = 1, 2, \ldots, K$$

10 K-NN编程小练习：

- 在平面上产生三个点A(2,2),B(3,7),C(8,5)在上述三个点的附近用正态随机数各生成15个点作为A类、B类、C类，再在 $x_1 \in [0, 10]$,$x_2 \in [0, 10]$上均匀生成若干点，用KNN进行分类


10
8
6
4
2
0
2
4
6
8
10
B
C
A

In [1]:

```
import numpy as np
Y = np.repeat(np.array(['A', 'B', 'C']), 15)
X = np.zeros(shape=(len(Y), 2))
cc = np.array([[2, 2], [3, 7], [8, 5]])
for i in range(len(Y)):
    if Y[i] == 'A':
        X[i, 0] = np.random.normal(cc[0][0], 1.5)
        X[i, 1] = np.random.normal(cc[0][1], 1.5)
    elif Y[i] == 'B':
        X[i, 0] = np.random.normal(cc[1][0], 1.5)
        X[i, 1] = np.random.normal(cc[1][1], 1.5)
    else:
        X[i, 0] = np.random.normal(cc[2][0], 1.5)
        X[i, 1] = np.random.normal(cc[2][1], 1.5)

testset = np.random.uniform(low=0, high=10, size=(3, 2))
abc = np.array(['A', 'B', 'C'])


def dis(x, y):
    return np.sqrt(np.sum((x-y)**2))


def k_nn(feature, labels, x, k):
    distance = np.zeros(shape=np.shape(feature)[0])
    distance = np.array([dis(feature[i, :], x) for i in range(len(distance))])
    thred = np.sort(distance)[0:k-1][-1]
    Index = np.where(distance <= thred)
    temp = np.array([np.sum(labels[Index] == abc[i]) for i in range(len(abc))])
    for i in range(len(abc)):
        if temp[i] == max(temp):
            result = abc[i]

    return result

k_nn(X, Y, testset, 3)
```

Out[1]:

```
'B'
```

11 与上例类似，但是只是新增一个点判断是4类中哪一类

In [2]:

```python
import numpy as np
import operator

def kNN_classify0(inX, dataSet, labels, k):
    dataSetSize = dataSet.shape[0]

    # 计算欧式距离
    diffMat = np.tile(inX, (dataSetSize, 1)) - dataSet
    sqDiffMat = diffMat ** 2
    sqDistances = sqDiffMat.sum(axis=1)
    distances = sqDistances ** 0.5
    sortedDistIndicies = distances.argsort()
    classCount = {}

    # 选择距离最小的k个点
    for i in range(k):
        voteIlabel = labels[sortedDistIndicies[i]]
        classCount[voteIlabel] = classCount.get(voteIlabel, 0) + 1

    # 排序
    sortedClassCount = sorted(classCount.items(), key=operator.itemgetter(1), reverse=True)
    return sortedClassCount[0][0]

def testDataSet():
    data = np.array(
        [[0.1, 0.1], [0.31, 0.15], [0.66, 0.93], [0.54, 0.89], [1.0, 0.3], [0.86, 0.21], [0.4
    labels = ['M', 'M', 'N', 'N', 'P', 'P', 'Q', 'Q']
    return data, labels


data, labels = testDataSet()
predict = kNN_classify0([0.1, 0.5], data, labels, 4)
print("kNN predicts the [0.1,0.5] belongs to ", predict)
```

```
kNN predicts the [0.1,0.5] belongs to  M
```

12 鸢尾花数据集按照3-7拆分后用KNN进行分类，并查看其效果

In [3]:

```python
import numpy as np
from sklearn.datasets import load_iris
from sklearn.model_selection import train_test_split
from collections import Counter
#from sklearn.neighbors import KNeighborsClassifier,KDTree

class KNN(object):
    def __init__(self, n_neighbors=3, p=2):
        """
        parameter: n_neighbors 临近点个数
        parameter: p 距离度量
        """
        self.n = n_neighbors
        self.p = p


    def fit(self, X_train, y_train):
        self.X_train = X_train
        self.y_train = y_train

    def predict(self, X):
        # 取出n个点
        knn_list = []
        for i in range(self.n):
            dist = np.linalg.norm(X - self.X_train[i], ord=self.p)
            knn_list.append((dist, self.y_train[i]))

        for i in range(self.n, len(self.X_train)):
            max_index = knn_list.index(max(knn_list, key=lambda x: x[0]))
            dist = np.linalg.norm(X - self.X_train[i], ord=self.p)
            if knn_list[max_index][0] > dist:
                knn_list[max_index] = (dist, self.y_train[i])

        # 统计
        knn = [k[-1] for k in knn_list]
        return Counter(knn).most_common()[0][0]

    # 统计准确度
    def score(self, X_test, y_test):
        right_count = 0
        for X, y in zip(X_test, y_test):
            label = self.predict(X)
            if label == y:
                right_count += 1
        return right_count / len(X_test)

######

iris = load_iris()
X = iris.data[:100, [0, 2]]
y = iris.target[:100]
y = np.where(y == 1, 1, -1)

X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.3)
knn = KNN()
knn.fit(X_train, y_train)
score = knn.score(X_test, y_test)
print("socre = %s"%score)
```

```
socre = 1.0
```

13 新增5个点的两分类KNN

In [4]:

```python
# 两分类的分离器
import numpy as np

def eu_dis(a, b):
    dis = np.sqrt(np.dot(a-b, a-b))
    return dis


class Knn_algorithm(object):
    def __init__(self, X_train, X_test, Y_train, k):  # Y_test,k):
        self.X_train = X_train
        self.X_test = X_test
        self.Y_train = Y_train
        # self.Y_test=Y_test
        self.k = k

    def classifiy(self):
        Y_res = np.zeros(shape=np.shape(self.X_test)[0])
        ndist = np.zeros(shape=np.shape(self.X_train)[0])
        for ii in range(np.shape(self.X_test)[0]):
            for jj in range(np.shape(self.Y_train)[0]):
                ndist[jj] = eu_dis(self.X_test[ii], self.X_train[jj])
            temp = np.sort(ndist)[self.k-1]
            index = np.where(ndist <= temp)
            # count=np.sum(self.Y_train[index]==1)
            if np.sum(self.Y_train[index] == 1) > self.k/2:
                Y_res[ii] = 1
            else:
                Y_res[ii] = -1
        return Y_res


X_1 = np.array([[3, 3], [4, 3], [1, 1], [4.5, 2.5], [0.5, 1],
                [0.5, 1.5], [2, 3], [2.5, 4], [2, 0.5], [3, 0.1],
                [0.7, 0.8], [1, 1.2], [2, 0.9], [3, 4], [4, 1]])
Y_1 = np.array([1, 1, -1, 1, -1, -1, 1, 1, -1, -1, -1, -1, -1, 1, 1])
X_2 = np.array([[1.3, 4], [2.2, 2], [0.8, 1.2], [3.4, 4.3], [1.7, 3.6]])


KNN = Knn_algorithm(X_1, X_2, Y_1, 5)
print(KNN.classifiy())
```

```
[ 1. -1. -1.  1.  1.]
```

14 玻璃分类（glass.data文件）

In [5]:

```
import os
import numpy as np
import pandas as pd
import operator

#os.chdir("D:\\陈雪东文件\\2021秋季学期授课\\数据挖掘与机器学习\\code_myself\\ML_myself_code\

glass=pd.read_csv('glass.data',sep=',',header=None)
glass.columns=['Id','RI','Na','Mg','Al','Si','K','Ca','Be','Ke','Type']

data_x=glass.loc[:,glass.columns[1:10]]

def testDataSet():
    data = np.array(data_x)
    labels = [str(x) for x in glass['Type'].tolist()]
    return data, labels

def kNN_classify0(inX, dataSet, labels, k):
    dataSetSize = dataSet.shape[0]

    # 计算欧式距离
    diffMat = np.tile(inX, (dataSetSize, 1)) - dataSet
    sqDiffMat = diffMat ** 2
    sqDistances = sqDiffMat.sum(axis=1)
    distances = sqDistances ** 0.5
    sortedDistIndicies = distances.argsort()
    classCount = {}

    # 选择距离最小的k个点
    for i in range(k):
        voteIlabel = labels[sortedDistIndicies[i]]
        classCount[voteIlabel] = classCount.get(voteIlabel, 0) + 1

    # 排序
    sortedClassCount = sorted(classCount.items(), key=operator.itemgetter(1), reverse=True)
    return sortedClassCount[0][0]


data, labels = testDataSet()
predict = kNN_classify0(np.array(glass.loc[23,glass.columns[1:10]]), data, labels, 5)
print(predict)
print(labels[23])
```

```
1
1
```

In [6]:

```
import os
import numpy as np
import pandas as pd
import operator

#os.chdir("D:\\陈雪东文件\\2021秋季学期授课\\数据挖掘与机器学习\\code_myself\\ML_myself_code\

glass=pd.read_csv('glass.data',sep=',',header=None)
glass.columns=['Id','RI','Na','Mg','Al','Si','K','Ca','Be','Ke','Type']

data_x=glass.loc[:,glass.columns[1:10]]

def testDataSet():
    data = np.array(data_x)
    labels = [str(x) for x in glass['Type'].tolist()]
    return data, labels

#######################################
#######################################

def eu_dis(a,b):
    dis=np.sqrt(np.dot(a-b,a-b))
    return dis

class Knn_algorithm(object):
    def __init__(self,X_train,X_test,Y_train,k):#Y_test,k):
        self.X_train=X_train
        self.X_test=X_test
        self.Y_train=Y_train
        #self.Y_test=Y_test
        self.k=k

    def classifiy(self):
        Y_res=np.zeros(shape=np.shape(self.X_test)[0])
        ndist=np.zeros(shape=np.shape(self.X_train)[0])
        for ii in range(np.shape(self.X_test)[0]):
            for jj in range(np.shape(self.Y_train)[0]):
                ndist[jj]=eu_dis(self.X_test[ii],self.X_train[jj])
            temp=np.sort(ndist)[self.k-1]
            index=np.where(ndist<=temp)
            #count=np.sum(self.Y_train[index]==1)
            if np.sum(self.Y_train[index]==1)>self.k/2:
                Y_res[ii]=1
            else:
                Y_res[ii]=-1
        return Y_res

########################################
########################################

KNN = Knn_algorithm(X_1,X_2,Y_1,5)
print(KNN.classifiy())
```

```
[ 1. -1. -1.  1.  1.]
```

In [7]:

```
import os
import numpy as np
import pandas as pd
#import operator

#os.chdir("D:\\陈雪东文件\\2021秋季学期授课\\数据挖掘与机器学习\\code_myself\\ML_myself_code\

glass=pd.read_csv('glass.data',sep=',',header=None)
glass.columns=['Id','RI','Na','Mg','Al','Si','K','Ca','Be','Ke','Type']

data_x=glass.loc[:,glass.columns[1:10]]

def DataSet():
    data = np.array(data_x)
    labels = np.array(glass['Type'])
    return data, labels

#######################################
#######################################

def eu_dis(a,b):
    dis=np.sqrt(np.dot(a-b,a-b))
    return dis

class Knn_algorithm(object):
    def __init__(self,X_train,X_test,Y_train,k):
        self.X_train=X_train
        self.X_test=X_test
        self.Y_train=Y_train
        self.k=k

    def classifiy(self):
        Y_res=np.zeros(shape=np.shape(self.X_test)[0])
        ndist=np.zeros(shape=np.shape(self.X_train)[0])
        uniY_train=np.unique(self.Y_train)
        count=np.zeros(shape=np.shape(uniY_train)[0])
        for ii in range(np.shape(self.X_test)[0]):
            for jj in range(np.shape(self.Y_train)[0]):
                ndist[jj]=eu_dis(self.X_test[ii],self.X_train[jj])
            temp=np.sort(ndist)[self.k-1]
            index=np.where(ndist<=temp)
            count= [np.sum(np.array(self.Y_train[index])==uniY_train[i]) for i in range(len(
            #count=np.sum(self.Y_train[index]==1)
            bb=np.where(count==max(count))
            Y_res[ii]=uniY_train[bb[0][0]]
            #if np.sum(self.Y_train[index]==1)>self.k/2:
            #    Y_res[ii]=1
            #else:
            #    Y_res[ii]=-1
        return Y_res

#########################################
#########################################

X_1,Y_1=DataSet()
X_2=np.array(data_x.iloc[103:205,:])

KNN = Knn_algorithm(X_1,X_1,Y_1,3)
print(KNN.classifiy())
```

```
[1. 1. 2. 1. 1. 2. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 3. 1. 1. 1. 1. 1. 1.
 1. 1. 3. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1.
 1. 1. 1. 1. 1. 1. 1. 2. 1. 1. 1. 1. 3. 1. 1. 1. 1. 1. 1. 1. 1. 1. 2. 2.
 2. 2. 2. 2. 2. 2. 1. 2. 2. 2. 1. 2. 2. 2. 2. 2. 2. 2. 2. 1. 2. 2. 1. 2.
 2. 1. 2. 2. 2. 2. 2. 2. 2. 2. 2. 2. 6. 2. 2. 2. 2. 2. 2. 2. 2. 2. 2. 2.
 2. 2. 2. 2. 2. 1. 1. 2. 2. 2. 2. 5. 2. 2. 1. 2. 1. 2. 2. 2. 2. 2. 2. 2.
 1. 1. 1. 3. 3. 1. 1. 1. 3. 3. 1. 1. 1. 1. 3. 3. 3. 3. 3. 7. 5. 5. 5. 5.
 5. 5. 5. 5. 5. 5. 5. 5. 6. 6. 6. 6. 7. 6. 2. 2. 6. 7. 7. 1. 2. 7. 6. 7.
 7. 7. 7. 7. 7. 7. 7. 7. 7. 1. 7. 7. 7. 7. 7. 7. 7. 7. 7. 7. 7. 7.]
```

15 32*32手写数字识别（文件夹testDigits和文件夹trainingDigits）

In [8]:

```
import os
import numpy as np
import operator
from numpy import *

def kNN_classify0(inX, dataSet, labels, k):
    dataSetSize = dataSet.shape[0]

    # 计算欧式距离
    diffMat = np.tile(inX, (dataSetSize, 1)) - dataSet
    sqDiffMat = diffMat ** 2
    sqDistances = sqDiffMat.sum(axis=1)
    distances = sqDistances ** 0.5
    sortedDistIndicies = distances.argsort()
    classCount = {}

    # 选择距离最小的k个点
    for i in range(k):
        voteIlabel = labels[sortedDistIndicies[i]]
        classCount[voteIlabel] = classCount.get(voteIlabel, 0) + 1

    # 排序
    sortedClassCount = sorted(
        classCount.items(), key=operator.itemgetter(1), reverse=True)
    return sortedClassCount[0][0]


def img2vector(filename):
    """
    格式化输入的数据图像，转换为向量
    :param filename: 文本文件名称
    :return: 1*1024的数字向量
    """
    dataVect = np.zeros((1, 1024))  # 预先分配一部分数据
    fr = open(filename)
    for i in range(32):
        lineText = fr.readline()
        for j in range(32):
            dataVect[0, 32 * i + j] = int(lineText[j])  # 利用偏移量使整个手写数字数据分布在
    return dataVect

def handwritingClassficationTest():
    hwLabels = []
    trainingFileList = os.listdir('trainingDigits')  # 加载训练数据
    m = len(trainingFileList)
    trainingMat = np.zeros((m, 1024))
    for i in range(m):
        fileNameStr = trainingFileList[i]
        fileStr = fileNameStr.split('.')[0]  # 切分数据
        classNumStr = int(fileStr.split('_')[0])
        hwLabels.append(classNumStr)
        trainingMat[i, :] = img2vector('trainingDigits/%s' % fileNameStr)
    testFileList = os.listdir('testDigits')  # 遍历测试数据集
    errorCount = 0.0
    mTest = len(testFileList)
    for i in range(mTest):
        fileNameStr = testFileList[i]
        fileStr = fileNameStr.split('.')[0]  # 切分数据
        classNumStr = int(fileStr.split('_')[0])
```

```
        vectorUnderTest = img2vector('testDigits/%s' % fileNameStr)
        classifierResult = kNN_classify0(
            vectorUnderTest, trainingMat, hwLabels, 3)
        print("the classifier came back with: %d, the real answer is: %d" %
              (classifierResult, classNumStr))
        if (classifierResult != classNumStr):
            errorCount += 1.0
    # 打印测试数据
    print("\nthe total number of errors is: %d" % errorCount)
    print("\nthe total accuracy is: %f" % (1 - errorCount / float(mTest)))

handwritingClassficationTest()
```

```
the classifier came back with: 0, the real answer is: 0
the classifier came back with: 0, the real answer is: 0
the classifier came back with: 0, the real answer is: 0
the classifier came back with: 0, the real answer is: 0
the classifier came back with: 0, the real answer is: 0
the classifier came back with: 0, the real answer is: 0
the classifier came back with: 0, the real answer is: 0
the classifier came back with: 0, the real answer is: 0
the classifier came back with: 0, the real answer is: 0
the classifier came back with: 0, the real answer is: 0
the classifier came back with: 0, the real answer is: 0
the classifier came back with: 0, the real answer is: 0
the classifier came back with: 0, the real answer is: 0
the classifier came back with: 0, the real answer is: 0
the classifier came back with: 0, the real answer is: 0
the classifier came back with: 0, the real answer is: 0
the classifier came back with: 0, the real answer is: 0
the classifier came back with: 0, the real answer is: 0
the classifier came back with: 0, the real answer is: 0
```

In [ ]:

```

```