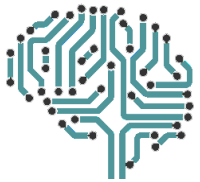
暨南大學
JINAN UNIVERSITY



机 器 学 习

第四章：K近邻算法

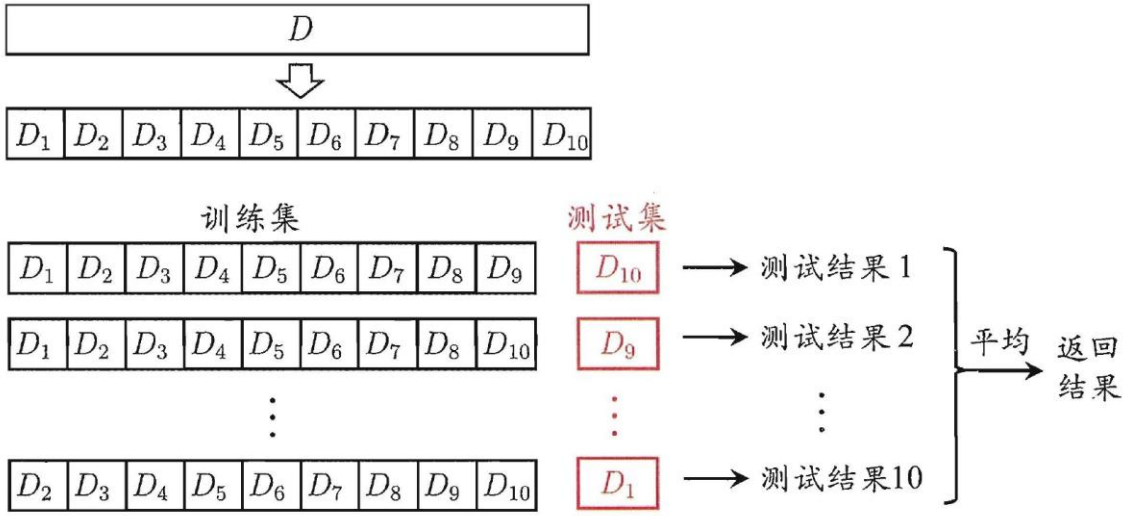
黄斐然

2022/3/14

数据分割

数据分割

- 原则：测试集应该与训练集“互斥”
- 常见方法：
  - 留出法 (hold-out)
  - 交叉验证法 (cross validation)



混淆矩阵

- 分类性能量度：

表 2.1 分类结果混淆矩阵

真实情况	预测结果	
	正例	反例
正例	TP (真正例)	FN (假反例)
反例	FP (假正例)	TN (真反例)

- 正确率：

$$Acc = \frac{TP + TN}{TP + FN + FP + TN}$$

- 精确率：

$$P = \frac{TP}{TP + FP}$$

- 召回率：

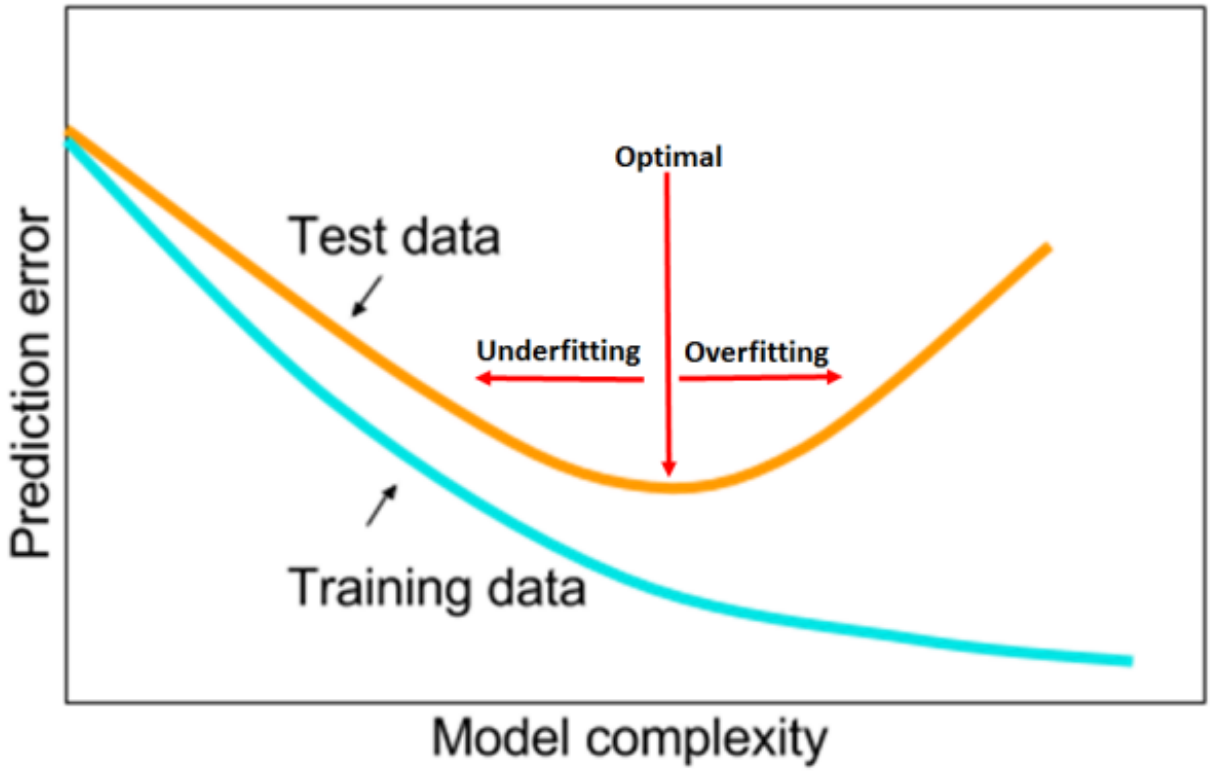
$$R = \frac{TP}{TP + FN}$$

- F1值：

$$F1 = \frac{2 \times P \times R}{P + R}$$

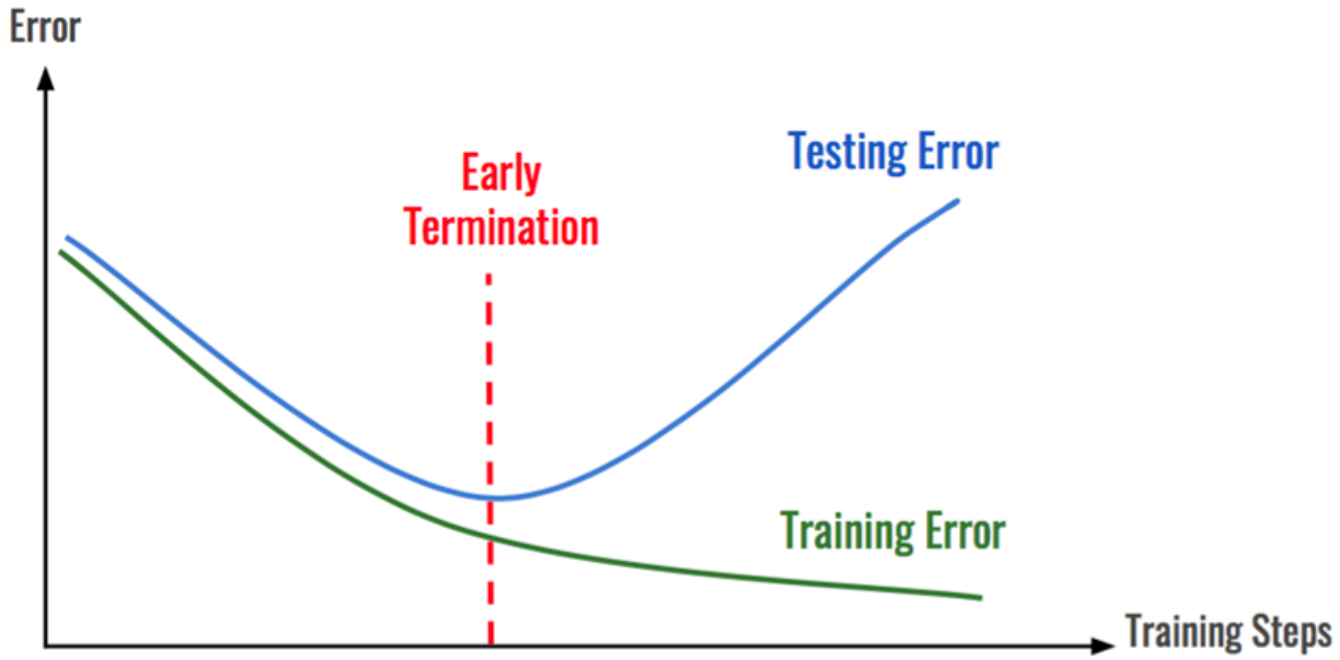
欠拟合与过拟合

- 模型复杂度与泛化能力的关系：



欠拟合与过拟合

- 训练迭代次数与泛化能力的关系：



目录

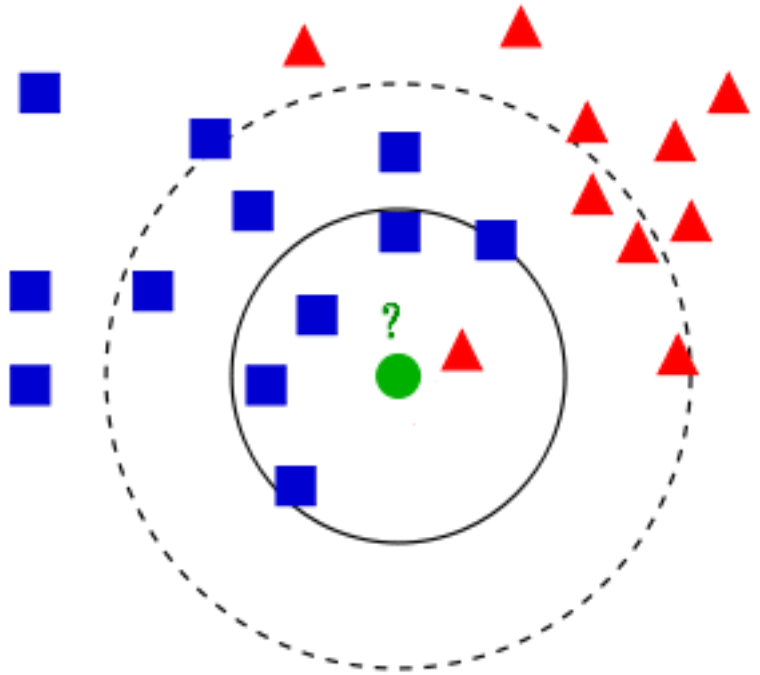
目录

1 K近邻算法简介

- 想一想：下面图片中只有三种豆，有三个豆是未知的种类，如何判定他们的种类?

1 K近邻算法简介

- 问题：有一个未知形状X(图中绿色的圆点)，如何判断X是什么形状?



1 K近邻算法简介

K近邻算法是什么?

- 最初的近邻法是由Cover和Hart于1968年提出的，随后得到理论上深入的分析与研究，是非参数分类法中最重要的方法之一。
- 这个算法首先贮藏所有的训练样本，然后通过分析（包括选举，计算加权和等方式）一个新样本周围K个最近邻，然后把新样本标记为在K近邻点中频率最高的类。

1 K近邻算法简介

K近邻算法特点

- 思想成熟
- 过程极其简单
- 应用数学知识少
- 效果好

目录

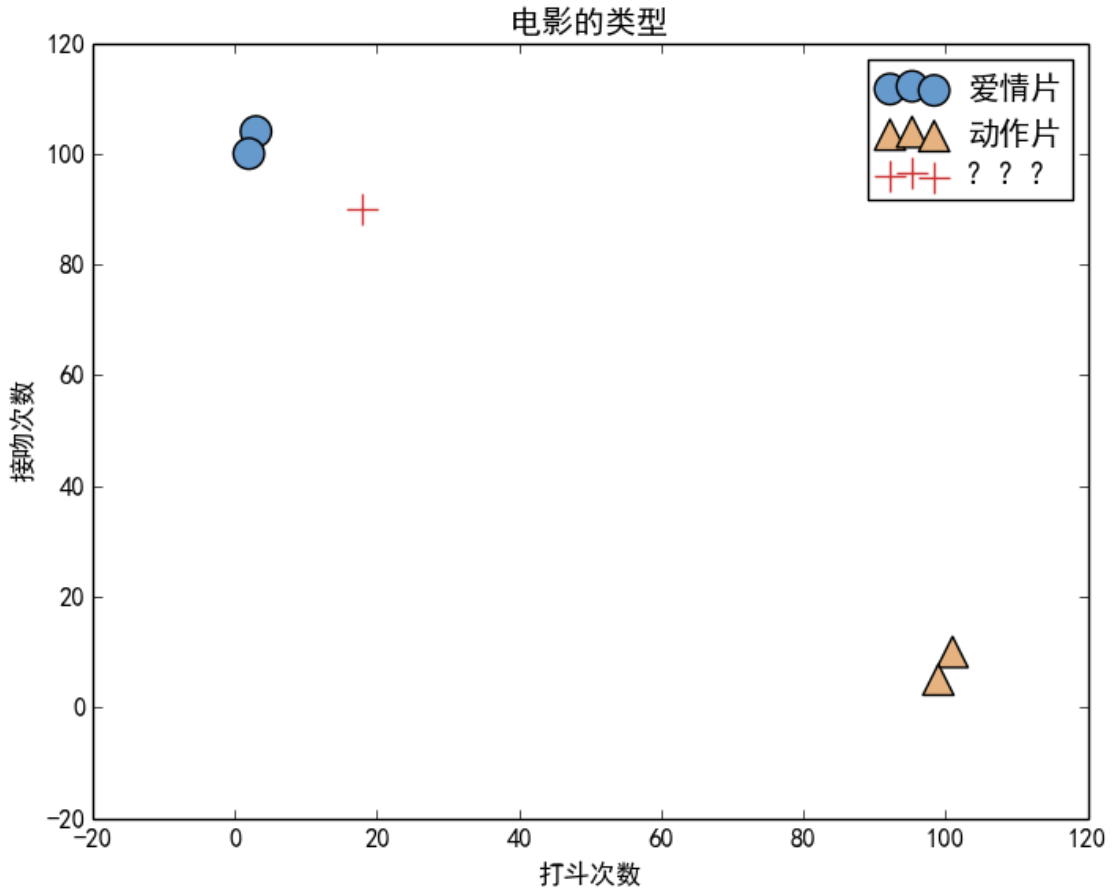
2 K近邻算法过程

- 猜猜看：乱世佳人属于什么类型的电影。

电影名称	打斗次数	接吻次数	电影类型
罗马假日	3	104	爱情片
傲慢与偏见	2	100	爱情片
速度与激情	101	10	动作片
谍影重重	99	5	动作片
乱世佳人	18	90	？？？

2 K近邻算法过程

- 猜猜看：乱世佳人属于什么类型的电影。



2 K近邻算法过程

算法步骤

1、存储训练样本（矩阵化）

- 训练样本的特征：
  - X_train = np.array([[3,104],[2,100],
    [101,10],[99,5]])
- 训练样本的分类：
  - Y_train = np.array([0,0,1,1])

2 K近邻算法过程

算法步骤

2、计算测试样本与训练样本的距离

- 测试样本的特征：
  - X_test = array([[18,90]])
- 计算测试数据和训练样本的距离：
  - dist =np.sum(np.square(X_test-X_train),axis=1)

2 K近邻算法过程

算法步骤

2、计算测试样本与训练样本的距离

- 测试样本和训练样本的距离：
  - Dist = np.sum(np.square(X_test-X_train),axis=1)

    =np.sum(([[-15,14],[-16,10],

    [83,-80],[81,-85]])**2, axis=1)

    =([421,356,13289,13786])

2 K近邻算法过程

算法步骤

3、根据距离长短进行排序

- sorted_index = np.argsort(dist)=([1,0,2,3])

4、计算与测试样本最近的K个训练数据

- 已知：Y_train = np.array([0,0,1,1])
- k =3
- pred_labels = Y_train[sorted_index[:k]]

  = Y_train[[1,0,2]]

  = [0,0,1]

2 K近邻算法过程

算法步骤

5、对不同的分类进行计数

- counts = np.bincount(pred_labels) #对label进行计数
- = [2,1]

6、找出计数最多的那个分类

pred_value = np.argmax(counts) # 找出计数最多的那个label值

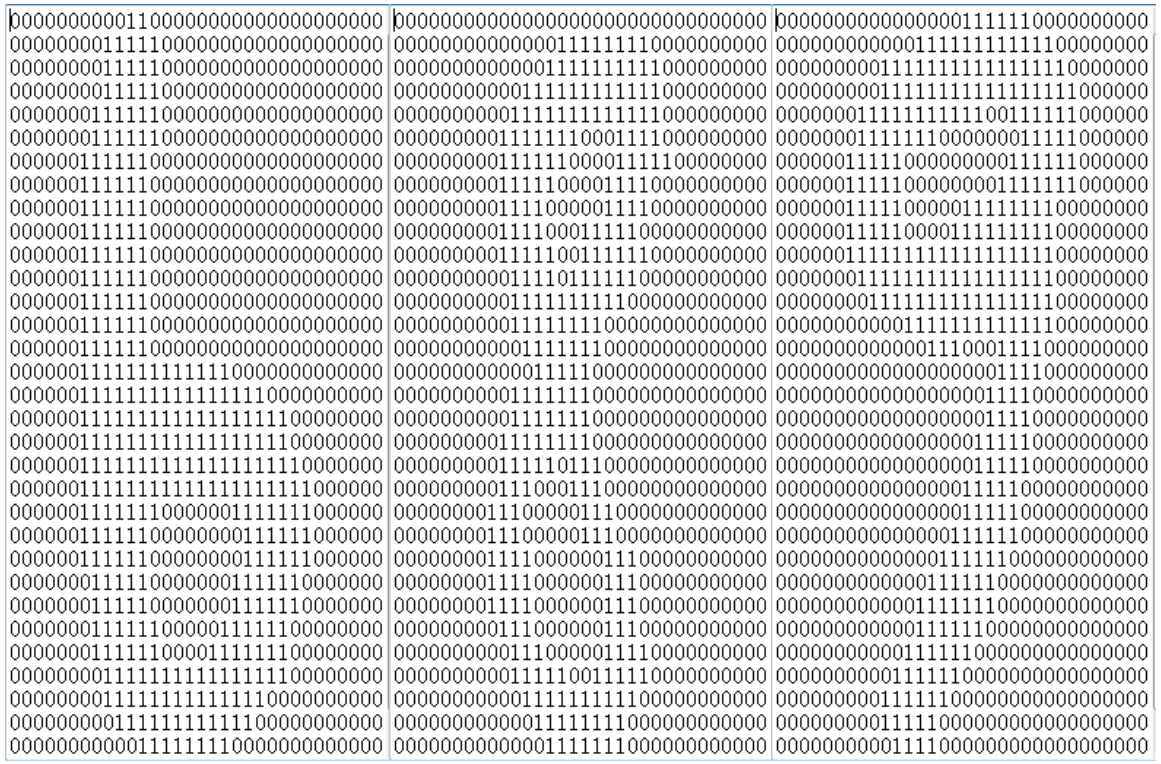
2 K近邻算法过程

- 不同K的结果：

```
euclidean distance: [  421   356 13289 13786]
sorted index: [1 0 2 3]
predicted labels: [0] predicted class: 0
predicted labels: [0 0] predicted class: 0
predicted labels: [0 0 1] predicted class: 0
predicted labels: [0 0 1 1] predicted class: 0
```

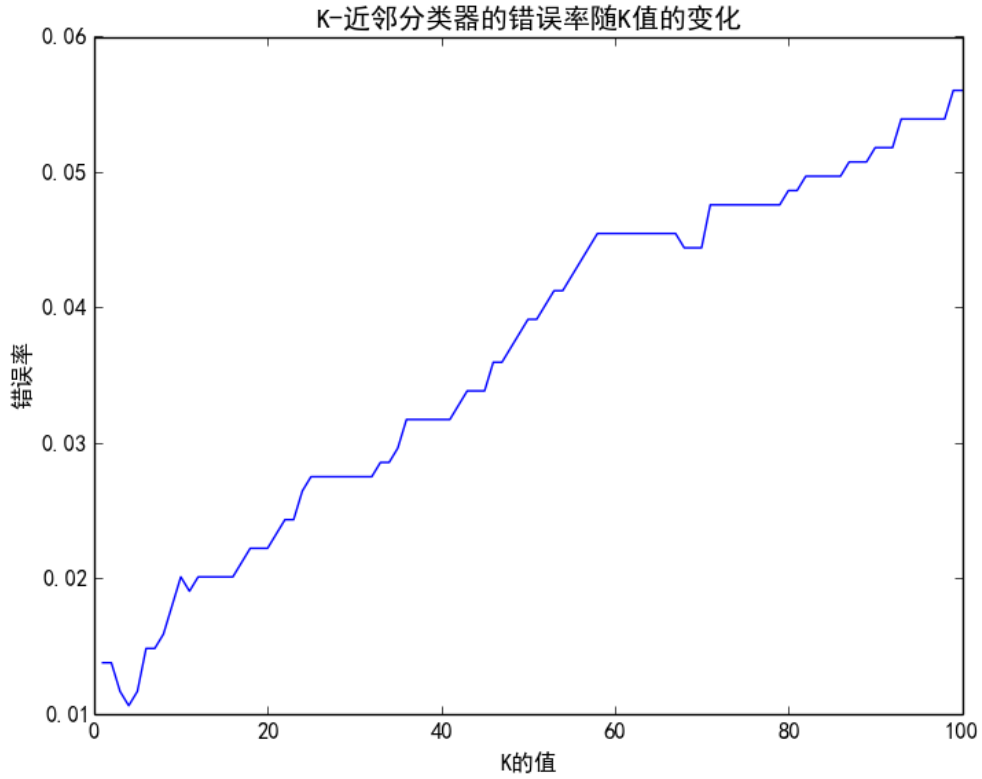
2 K近邻算法过程

- 例——数字识别：
  - 每个图片分辨率为32*32，有32*32=1024个像素，所以每个图片数据有1024维特征，每个特征可以取0\1
  - 每个图片数据的分类空间为0-9，总共十个分类



2 K近邻算法过程

- 例——数字识别：
  - K-近邻分类器的错误率随K值的变化



2 K近邻算法过程

- 距离的量度：
  - 曼哈顿距离(Manhattan Distance)

$$D(x,y)=\sum_{i=1}^{k}|x_i-y_i|$$

    - 曼哈顿距离通常称为出租车距离或城市街区距离，用来计算实值向量之间的距离。

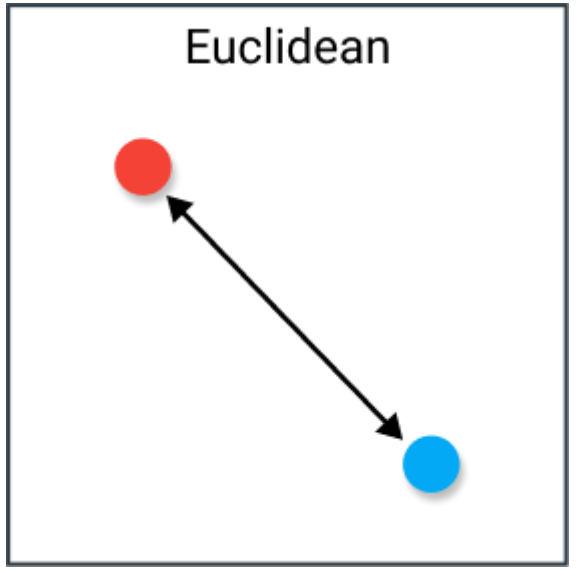
2 K近邻算法过程

距离的量度：

- 欧氏距离(Euclidean Distance)

$$D(x,y)=\sqrt{\sum_{i=1}^{n}(x_i-y_i)^2)}$$

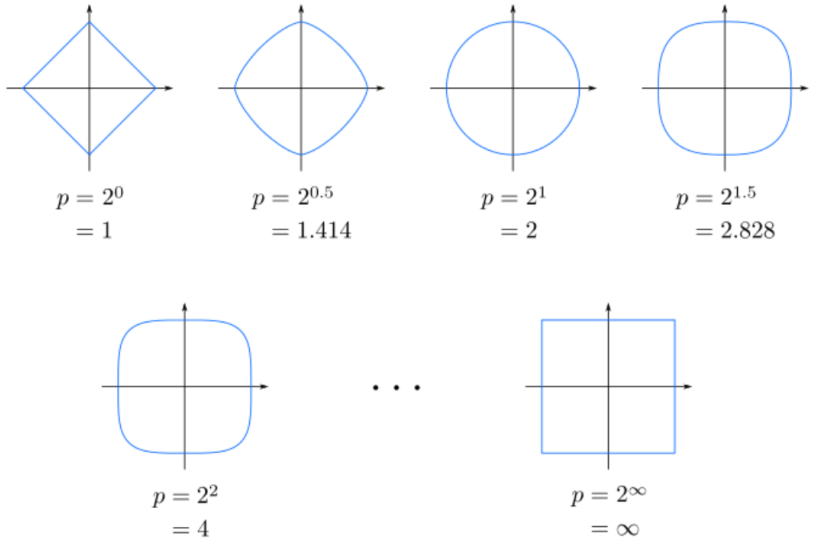
  - 欧氏距离是最常见的距离度量，衡量的是多维空间中各个点之间的绝对距离



2 K近邻算法过程

- 距离的量度：
  - Minkowski Distance（闵式距离）

$$D = \left(\sum_{i=1}^{d} |x_i - y_i|^p \right)^{\frac{1}{p}}$$



- 适用于数值型数据（numeric data）
- 欧氏距离是闵式距离在（p=2）的特例

目录

3 KD树

基础的K近邻算法：

- k近邻法最简单有效的方法是线性扫描（穷举搜索），即要计算输入实例与每一个训练实例的距离，再查找k近邻，当训练数据很大时，计算非常耗时，为提高KNN搜索效率，就引入了kd树的概念。

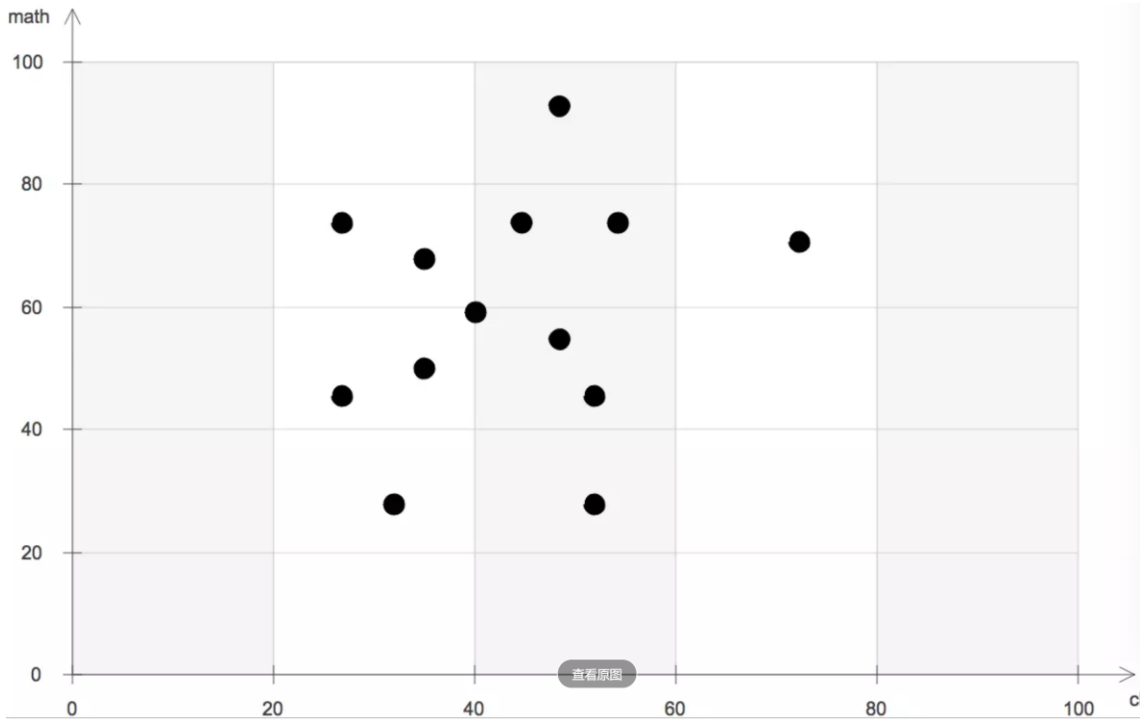
3 KD树

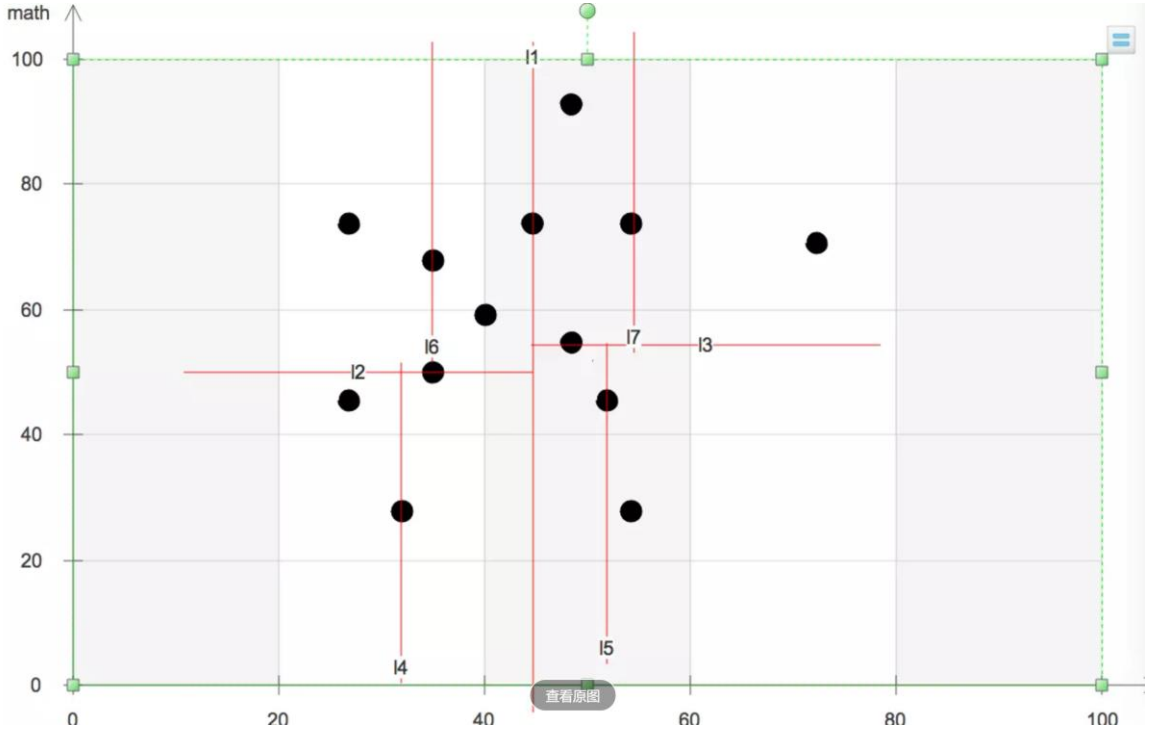
KD树：

- 概念：Kd树是一种对k维空间中的实例点进行储存以便对其进行快速检索的树形数据结构。
- 本质：二叉树，表示对k维空间的一个划分。构造过程：不断地用垂直于坐标轴的超平面将k维空间切分，形成k维超矩形区域。kd树的每一个结点对应于一个k维超矩形区域

3 KD树

- KD树：


math
100
80
60
40
20
0
0
20
40
60
80
100
c
查看原图


math
100
80
60
40
20
0
l1
l2
l3
l4
l5
l6
l7
0
20
40
60
80
100
查看原图

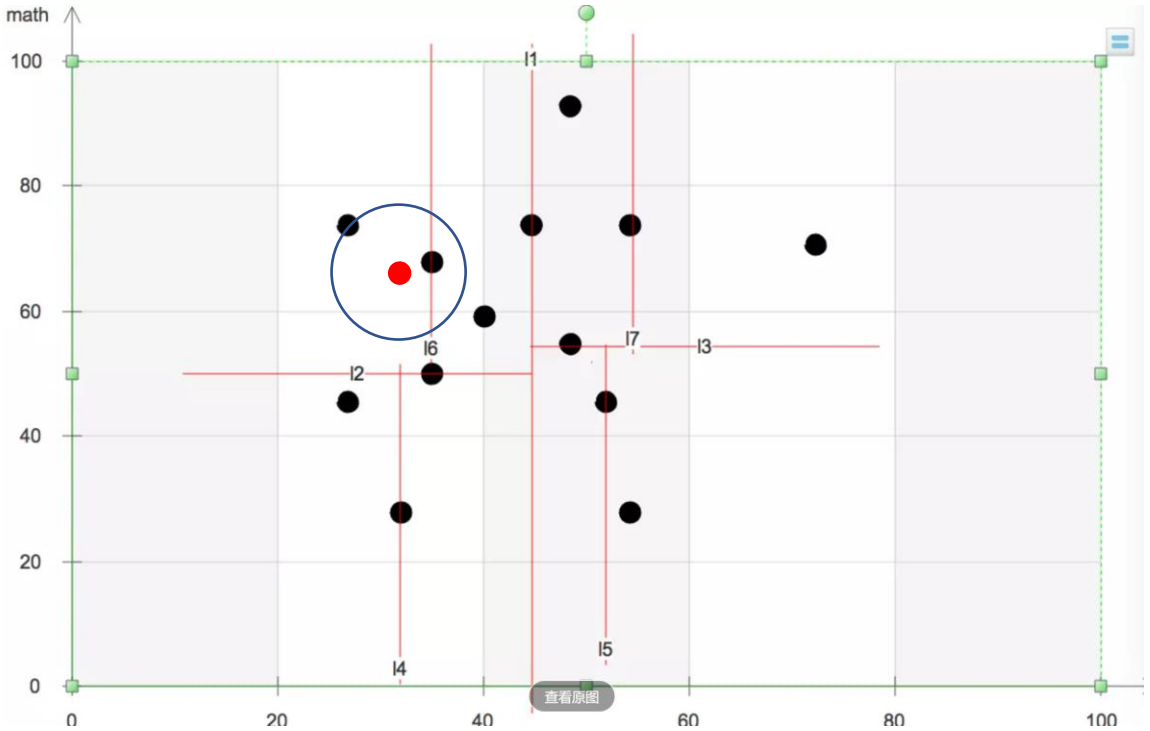
3 KD树

KD树搜索：

- 步骤：首先在KD树里面找到包含目标点的叶子节点。以目标点为圆心，以目标点到叶子节点样本实例的距离为半径，得到一个超球体，最近邻的点一定在这个超球体内部。然后返回叶子节点的父节点，检查另一个子节点包含的超矩形体是否和超球体相交，如果相交就到这个子节点寻找是否有更加近的近邻，有的话就更新最近邻。如果不相交那就简单了，直接返回父节点的父节点，在另一个子树继续搜索最近邻。当回溯到根节点时，算法结束，此时保存的最近邻节点就是最终的最近邻。

3 KD树

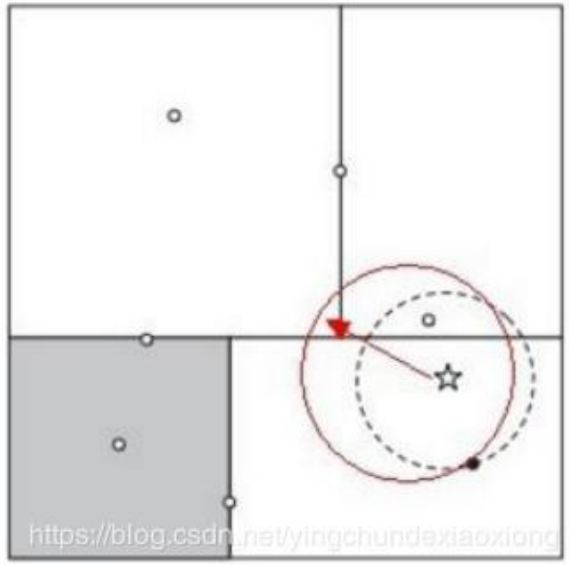
- KD树：

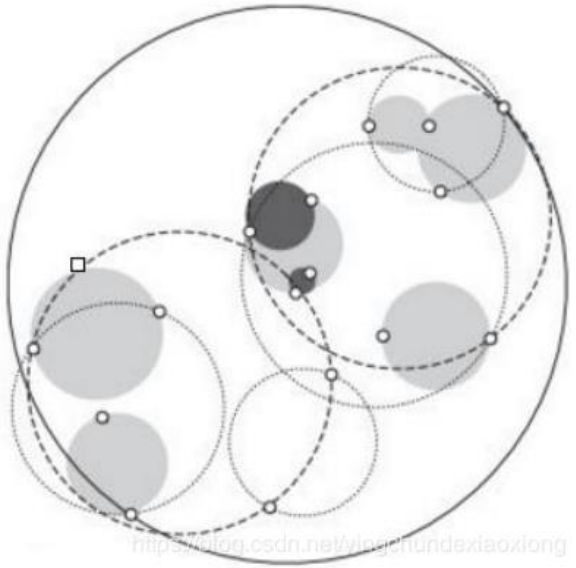


3 球树

球树：

- KD树算法虽然提高了KNN搜索的效率，但是在某些时候效率并不高，比如当处理不均匀分布的数据集时，不管是近似方形，还是矩形，甚至正方形，都不是最好的使用形状，因为他们都有角。一个例子如下图：





3 球树

球树过程：

- 1) 先构建一个超球体，这个超球体是可以包含所有样本的最小球体。
- 2) 从球中选择一个离球的中心最远的点，然后选择第二个点离第一个点最远，将球中所有的点分配到离这两个聚类中心最近的一个上，然后计算每个聚类的中心，以及聚类能够包含它所有数据点所需的最小半径。这样我们得到了两个子超球体，和KD树里面的左右子树对应。
- 3)对于这两个子超球体，递归执行步骤2) 最终得到了一个球树。　使用球树找出给定目标点的最近邻方法是首先自上而下贯穿整棵树找出包含目标点所在的叶子，并在这个球里找出与目标点最邻近的点，这将确定出目标点距离它的最近邻点的一个上限值，然后跟KD树查找一样，检查兄弟结点，如果目标点到兄弟结点中心的距离超过兄弟结点的半径与当前的上限值之和，那么兄弟结点里不可能存在一个更近的点；否则的话，必须进一步检查位于兄弟结点以下的子树。检查完兄弟节点后，我们向父节点回溯，继续搜索最小邻近值。当回溯到根节点时，此时的最小邻近值就是最终的搜索结果。

目录

4 算法总结

使用sklearn实现：

- from sklearn.neighbors import KNeighborsClassifier
- KNeighborsClassifier(n_neighbors=5, weights='uniform', algorithm='auto', leaf_size=30, p=2, metric='minkowski', metric_params=None, n_jobs=None, **kwargs)
- 重要参数：
  - n_neighbors：近邻数。默认5。
  - weights：近邻权重。默认全一致；也可使用‘distance’，与距离成反比。也可传函数自定义。
  - algorithm：默认自动选择。也可指定为‘ball_tree’（球树）、‘kd_tree’（KD树）、‘brute’（暴力搜索）
  - p：默认2。表示闵氏距离的p值。 $D(x,y)=\left(\sum_{u=1}^{n}|x_u-y_u|^p\right)^{\frac{1}{p}}$
  - n_jobs：默认一个。表示使用线程的个数，使用-1表示使用全部可用线程。

4 算法总结

■ 使用sklearn实现：

In [11]:

```
from sklearn.neighbors import KNeighborsClassifier
from sklearn.datasets import load_breast_cancer
from sklearn.model_selection import KFold
from sklearn import metrics
import numpy as np

dataset = load_breast_cancer()
X = dataset['data']
y = dataset['target']

avg_scores = []
for i in range(5):

    kf = KFold(n_splits=10,shuffle=True)
    score = []
    for train_inx, test_inx in kf.split(X):
        clf = KNeighborsClassifier(n_neighbors=5, weights='uniform', algorithm='auto', leaf_size=30
                                   ).fit(X[train_inx],y[train_inx])
        y_pre = clf.predict(X[test_inx])
        y_test = y[test_inx]
        score.append(metrics.accuracy_score(y_test,y_pre))
    avg_scores.append(np.mean(score))

print(np.mean(avg_scores))
```

```
0.9318483709273181
```

4 算法总结

- K近邻优点：
  - 思想简单，理论成熟，既可以用来做分类也可以用来做回归。
  - 可用于非线性分类。
  - 训练时间复杂度为O(n)。
  - 准确度高，对数据没有假设，对outlier不敏感。

4 算法总结

- K近邻缺点：
  - 计算量太大。
  - 对于样本分类不均衡的问题，会产生误判。
  - 需要大量的内存。
  - 输出的可解释性不强。

问题？

暨南大學
JINAN UNIVERSITY