K-最近邻算法

1.算法介绍

$KNearestNeighbors$算法又叫$K-NN$算法，这个算法是机器学习里面一个比较经典的算法， 总体来说$K-NN$算法是相对比较容易理解的算法

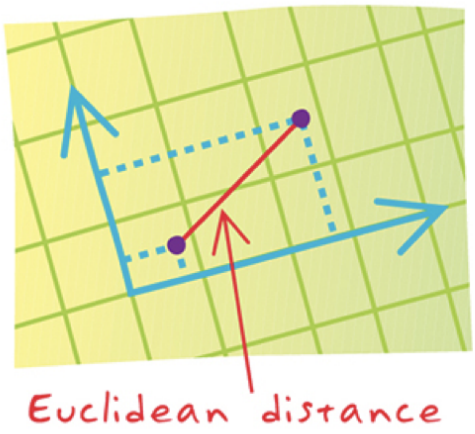
- 定义

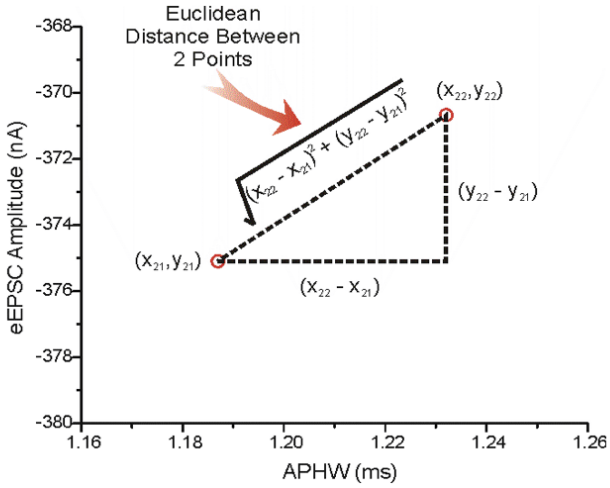
如果一个样本在特征空间中的k个最相似(即特征空间中最邻近)的样本中的大多数属于某一个类别，则该样本也属于这个类别。

> 来源：$K-NN$算法最早是由Cover和Hart提出的一种分类算法

2.算法公式

两个样本的距离可以通过如下公式计算，又叫欧式距离 ， 关于距离公式会在后面进行讨论





(1)分类

$K-NN$针对于离散型分类目标的一种非线性多分类的，基于加权距离的最大投票方案算法，公式如下：

$$f(z) = max_j \sum_{i=1}^{k} \varphi(d_y) I_y$$

- 预测函数$f(z)$是所有分类j的最大加权值
- 预测数据点到训练数据点i的加权距离用$\varphi(d_y)$表示
- I_y是指示函数,表示数据点i是否属于分类j时,指示函数为1,否则为0，同时k是距离预测数据点最近的训练数据个数。

(2)回归

$K-NN$针对于连续型回归目标，预测值是所有k个最近邻域数据点到预测数据点的加权平均，公式如下：

$f(z) = \frac{1}{k}\sum_{i=1}^{k}\varphi(d_i)$

在$K-NN$中可以发现,预测值严重依赖距离度量$\varphi(d)$方式的先择。常用的距离度量方式是$L1_{范数}$和$L2_{范数}$。公式如下:

(3)L1和L2范数距离

- $L1$范数距离(曼哈顿距离):

  $\varphi d_{L1}(x_i, x_j) = |x_i - x_j| = |x_{i1} - x_{j1}| + |x_{i2} - x_{j2}| + \ldots + |x_{in} - x_{jn}|$
- $L2$范数距离(欧几里得距离):

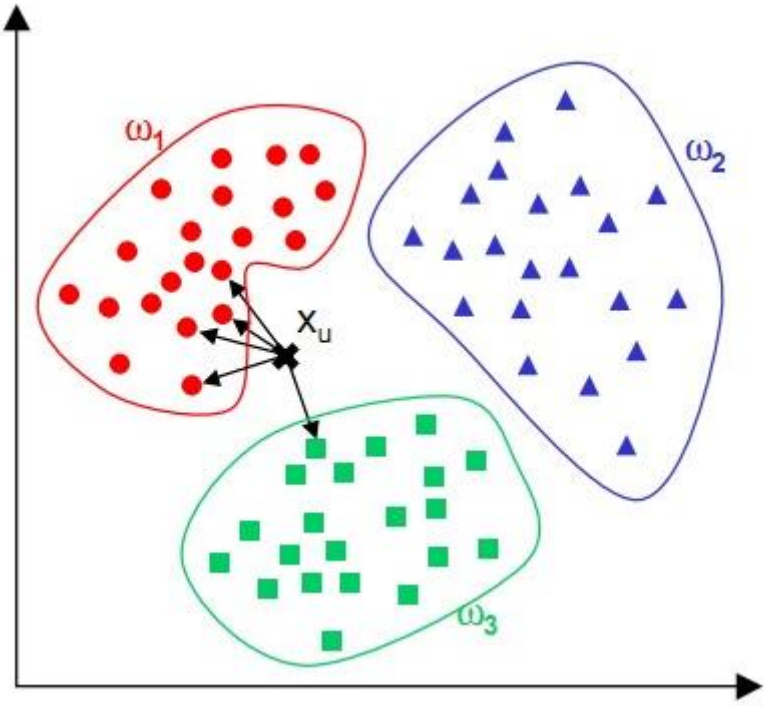
  $d_{L2}(x_i, x_j) = ||x_i - x_j|| = \sqrt{(x_{i1} - x_{j1})^2 + (x_{i2} - x_{j2})^2 + \ldots + (x_{in} - x_{jn})^2}$
- 闵可夫斯基(knn中使用的)
  - 当p=1的时候，它是曼哈顿距离
  - 当p=2的时候，它是欧式距离
  - 当p不选择的时候,它是切比雪夫

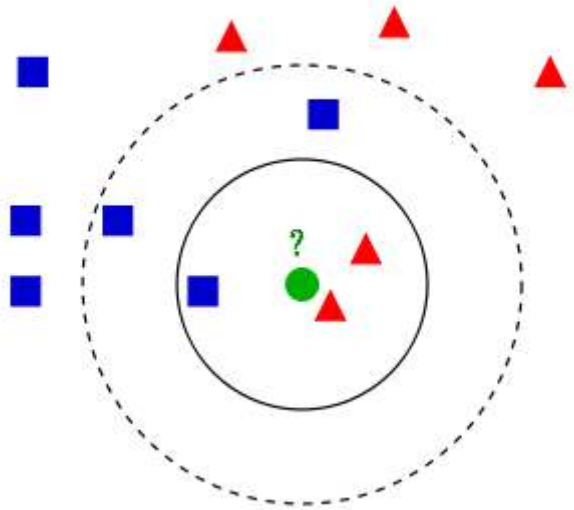
  $d_{12} = \sqrt[p]{\sum_{i=1}^{k} |x_{1k} - x_{2k}|^p}$

3.K值选择

K值选择问题，李航博士的一书`统计学习方法`**上所说：**

- 1) 选择较小的K值，就相当于用较小的领域中的训练实例进行预测，“学习”近似误差会减小，只有与输入实例较近或相似的训练实例才会对预测结果起作用，与此同时带来的问题是“学习”的估计误差会增大，换句话说，**K值的减小就意味着整体模型变得复杂(指的是数值发生一点点改变，分类结果就不相同了，模型的泛化性不佳)，容易发生过拟合；**
- 2) 选择较大的K值，就相当于用较大领域中的训练实例进行预测，其优点是可以减少学习的估计误差，但缺点是学习的近似误差会增大。这时候，**与输入实例较远（不相似的）训练实例也会对预测器作用，使预测发生错误，且K值的增大就意味着整体的模型变得简单。**
- 3) K=N（N为训练样本个数），则完全不足取，因为此时无论输入实例是什么，都只是简单的预测它属于在训练实例中最多的类，模型过于简单，忽略了训练实例中大量有用信息。





在实际应用中，K值一般取一个比较小的数值，例如采用交叉验证法（简单来说，就是把训练数据在分成两组:训练集和验证集）来选择最优的K值。对这个简单的分类器进行泛化，用核方法把这个线性模型扩展到非线性的情况，具体方法是把低维数据集映射到高维特征空间。

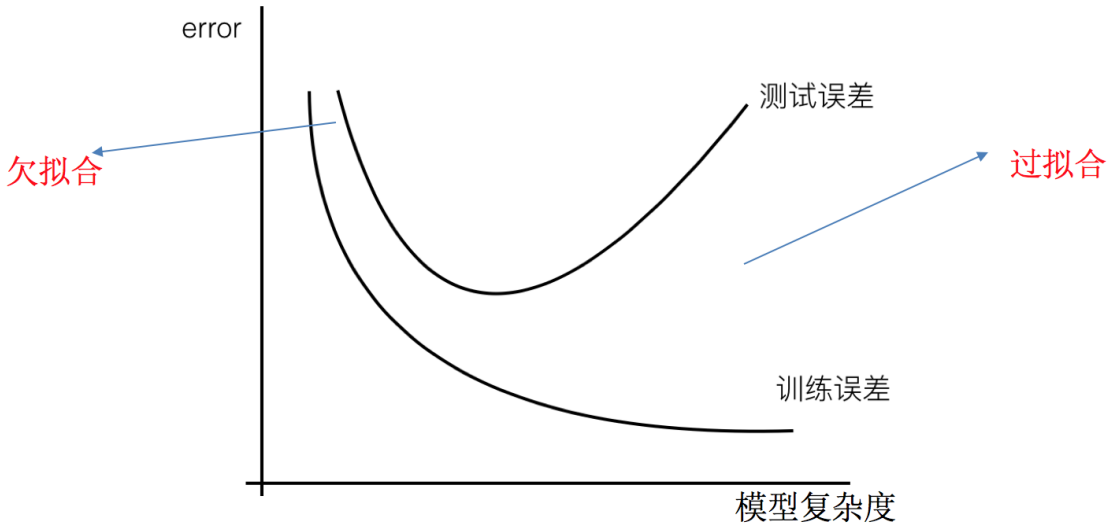
近似误差：对现有训练集的训练误差，关注训练集，如果近似误差过小可能会出现过拟合的现象，对现有的训练集能有很好的预测，但是对未知的测试样本将会出现较大偏差的预测。模型本身不是最接近最佳模型。

估计误差：可以理解为对测试集的测试误差，关注测试集，估计误差小说明对未知数据的预测能力好，模型本身最接近最佳模型。

4.欠拟合和过拟合

(1)定义

- 过拟合：一个假设**在训练数据上能够获得比其他假设更好的拟合， 但是在测试数据集上却不能很好地拟合数据**，此时认为这个假设出现了过拟合的现象。(模型过于复杂)
- 欠拟合：一个假设**在训练数据上不能获得更好的拟合，并且在测试数据集上也不能很好地拟合数据**，此时认为这个假设出现了欠拟合的现象。(模型过于简单)



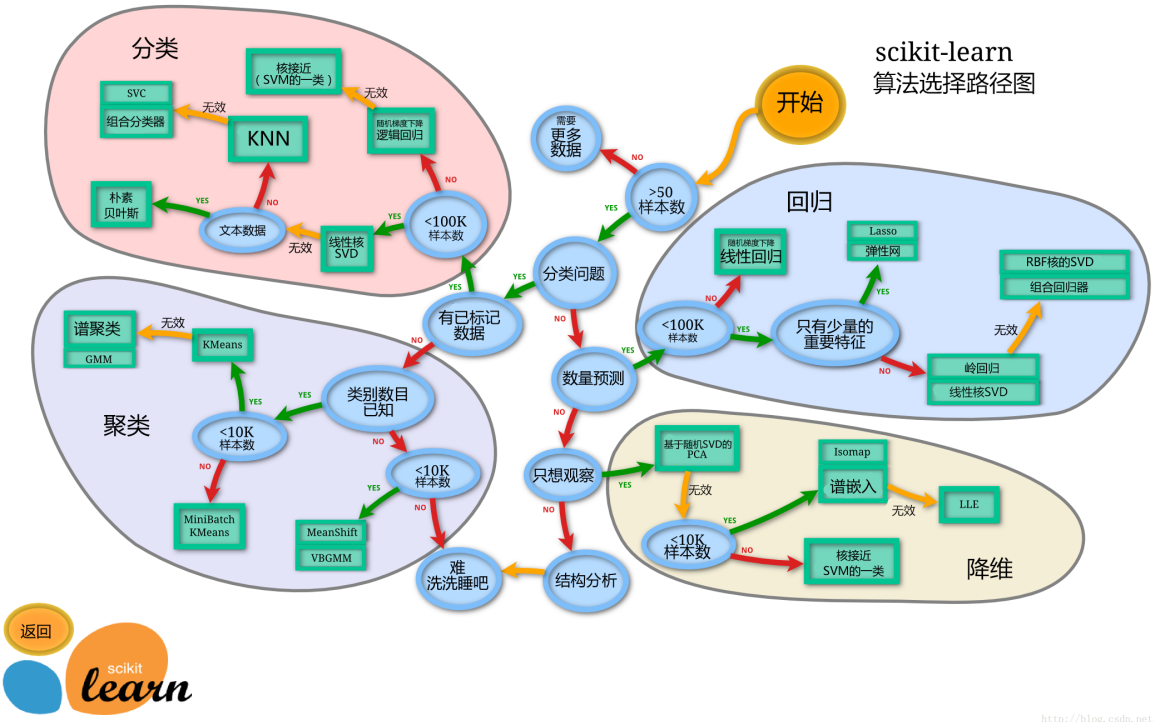
(2)原因以及解决办法

- 欠拟合原因以及解决办法
  - 原因：学习到数据的特征过少
  - 解决办法：
    - **1）添加其他特征项，**有时候我们模型出现欠拟合的时候是因为特征项不够导致的，可以添加其他特征项来很好地解决。例如，“组合”、“泛化”、“相关性”三类特征是特征添加的重要手段，无论在什么场景，都可以照葫芦画瓢，总会得到意想不到的效果。除上面的特征之外，“上下文特征”、“平台特征”等等，都可以作为特征添加的首选项。
    - **2）添加多项式特征**，这个在机器学习算法里面用的很普遍，例如将线性模型通过添加二次项或者三次项使模型泛化能力更强。
- 过拟合原因以及解决办法
  - 原因：原始特征过多，存在一些嘈杂特征， 模型过于复杂是因为模型尝试去兼顾各个测试数据点
  - 解决办法：
    - 1）重新清洗数据，导致过拟合的一个原因也有可能是数据不纯导致的，如果出现了过拟合就需要我们重新清洗数据。
    - 2）增大数据的训练量，还有一个原因就是我们用于训练的数据量太小导致的，训练数据占总数据的比例过小。
    - 3）减少特征维度，防止**维灾难**
    - 4) 对于Knn来说,近邻的数量非常的重要

5.sklearn

Machine Learning with Scikit-Learn

- Python语言的机器学习工具
- Scikit-learn包括许多知名的机器学习算法的实现
- Scikit-learn文档完善，容易上手，丰富的API



- 目前稳定版本0.19.1,最新版本为0.21.3

1.安装

```
pip install scikit-learn==0.19.1
```

2.使用sklearn的knn模块进行分类

```
#1.模型实例
#n_neighbors=5 默认找周围的5个离自己最近的数据
knn=KNeighborsClassifier(n_neighbors=5)
#2.将数据进行训练(把数据保存到内存中)
```

```
#X是特征数据
#y是目标数据(标签)
knn.fit(X_train,y_train)
#3.评估 1.0代表100%的准确
knn.score(X_train,y_train)
#准确率    被预测正确的数据数量   /   总数据数量
#4.对测试数据进行评估
knn.score(X_test,y_test)
#5.如果模型的分值高，那么模型可以上线,可以进行预测了
knn.predict(X_test)
```

6.使用python实现knn

```
class KNearestNeighbors(object):
    #初始化的部分
    def __init__(self,n_neighbors=5):
        self.k = n_neighbors

    #训练的函数
    def fit(self,X,y):
        self.X_train = np.array(X)
        self.y_train = np.array(y)

    #预测数据 返回预测的分类结果
    def predict(self,X_test):
        X_test = np.array(X_test)
        #计算欧式距离
        dist = self.distance(X_test)
        #获取到测试数据集的样本数量
        n_test_samples=len(X_test)
        #初始化分类记过的数组
        y_pred = np.zeros(n_test_samples)

        for i in range(n_test_samples):
            #获取分类的类别
            cls_y = self.y_train[np.argsort(dist[i])[:self.k]]
            #计算概率
            y_pred[i]= np.argmax(np.bincount(cls_y))

        return y_pred.astype(np.int)

    def distance(self,X_test):
        #获取到测试数据集的样本数量
        n_test_samples=len(X_test)
        #获取训练数据集的样本数量
        n_train_samples=len(self.X_train)
        #初始化的真实距离结果,
        dist = np.zeros((n_test_samples,n_train_samples))
        #计算真实距离
        for i in range(n_test_samples):
            #实现欧式距离计算
            dist[i] = np.sqrt(np.sum(np.square(self.X_train -
X_test[i]),axis=1))
        return dist

    def score(self,X,y):
        #获取预测的目标值
```

```
        y_pred = self.predict(X)
        #准确率 = 判断正确的值的数量 / 总数
        true = (y_pred == y).sum()
        return true/len(y)
```

7.KNN的优缺点

KNN的优缺点

- 优点：
  - 简单有效
  - 重新训练的代价低
  - 对于异常值不太敏感(基于邻域分类)
- 缺点：
  - 时间复杂度高、空间复杂度高
  - 惰性学习，效率低下
  - 输出的结果可解释性不强
  - 适用数据范围：数值型和标称型
  - 可解释性不强

8.模型提升KD-Tree

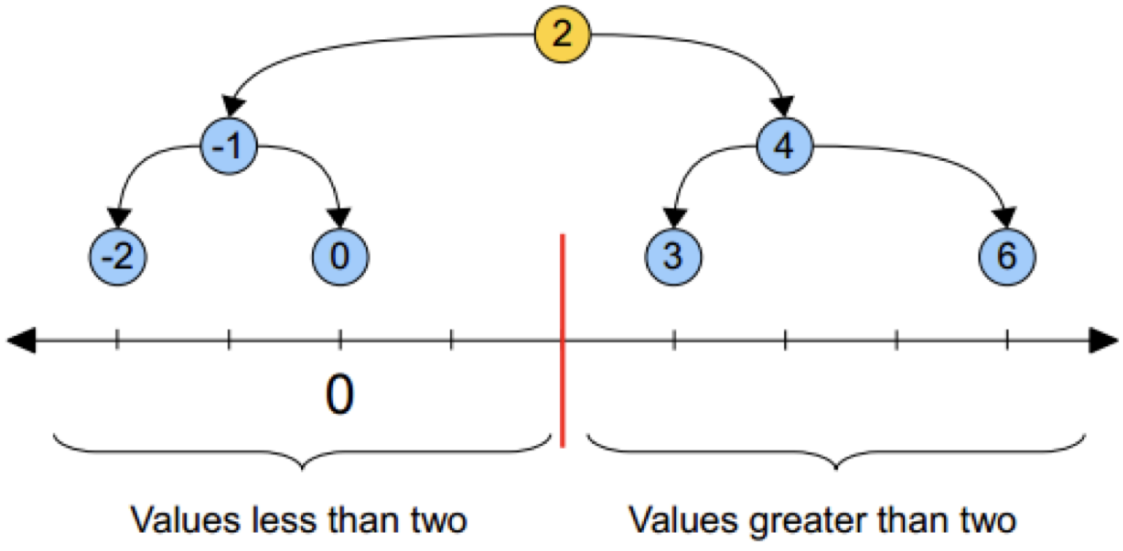
k最近邻法的实现是线性扫描（以穷举搜索的方式进行距离测算，加权分类），即要计算输入实例与每一个训练实例的距离。计算并存储好以后，再查找K近邻。当训练集很大时，计算效率非常的低下。

为了提高KNN最近邻搜索的效率，可以考虑使用特殊的结构存储训练数据，以减小计算距离的次数。

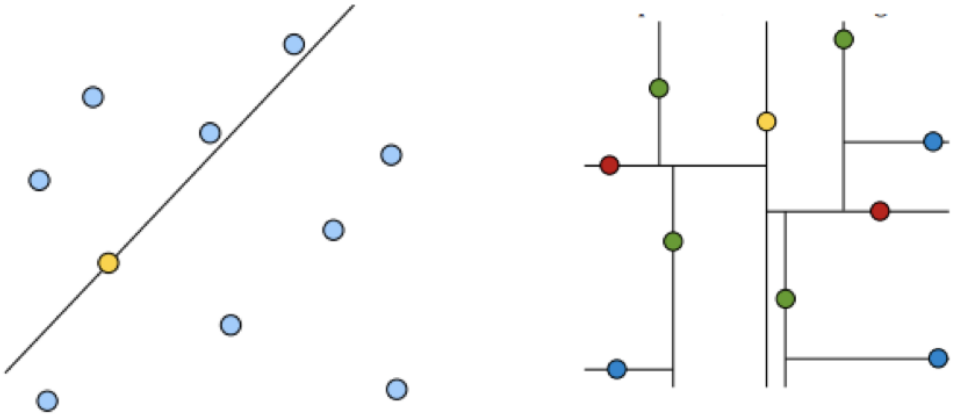
根据**KNN**每次需要预测一个点时，我们都需要计算训练数据集里每个点到这个点的距离，然后选出距离最近的k个点进行投票。**当数据集很大时，这个计算成本非常高，针对N个样本，D个特征的数据集，其算法复杂度为O（DN^2）。**

kd树：为了避免每次都重新计算一遍距离，算法会把距离信息保存在一个树状结构中，这样在计算之前从树结构查询距离信息，尽量避免重复计算。其基本原理是，**如果A和B距离很远，B和C距离很近，那么A和C的距离也很远。**有了这个信息，就可以在合适的时候跳过距离远的点。(A和B这两个类别是近邻类，B和C是近邻类,A和C不相近)

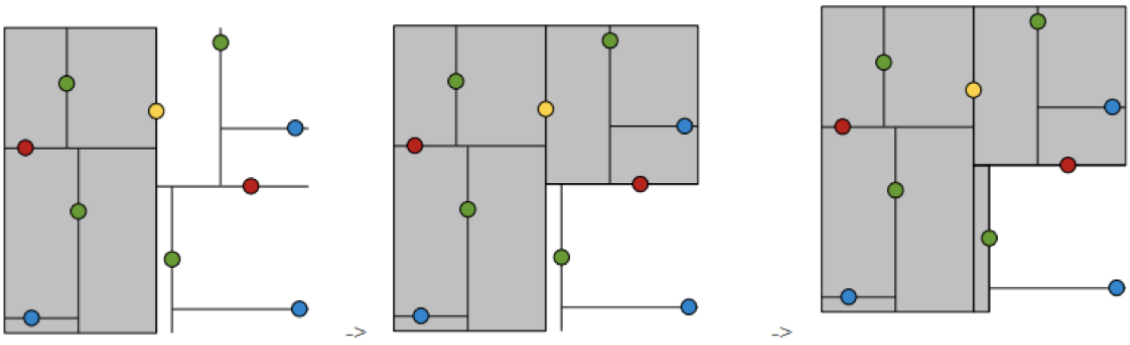
原理：



- 黄色的点作为根节，上面的点归左子树，下面的点归右子树，接下来再不断地划分，分割的那条线叫做分割超平面（splitting hyperplane）
- 在一维中是一个点，二维中是线，三维的是超平面。



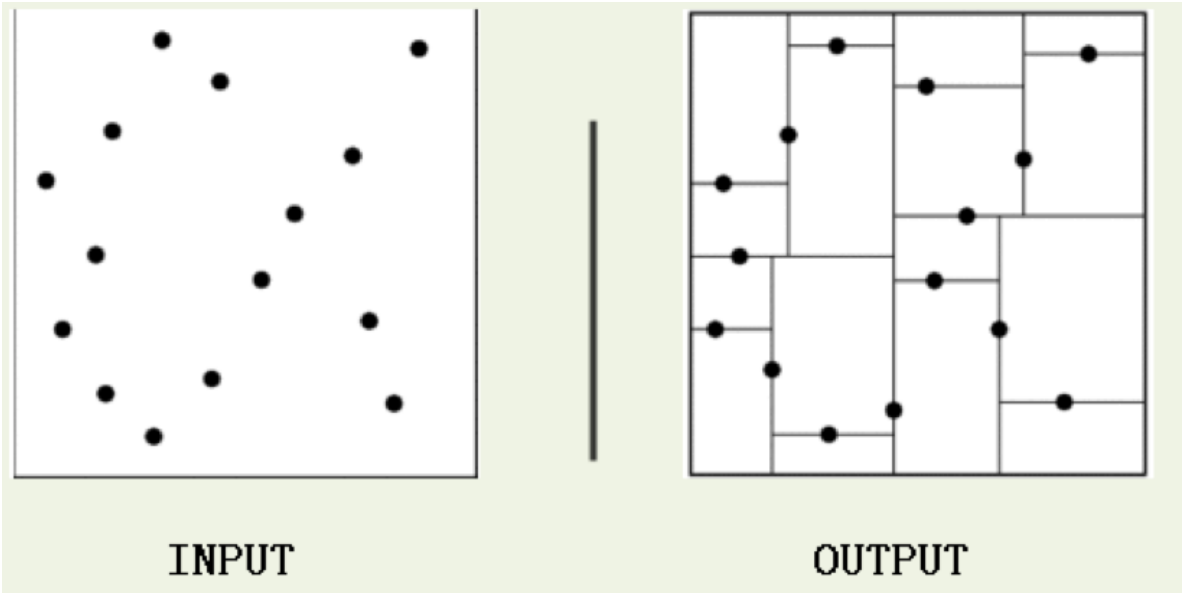
- 黄色点为根节点，一层子节点是红色，再下一层子节点是绿色，叶子节点为蓝色。



1.树的建立；

2.最近邻域搜索（Nearest-Neighbor Lookup）

kd树(K-dimension tree)是**一种对k维空间中的实例点进行存储以便对其进行快速检索的树形数据结构。**kd树是一种二叉树，表示对k维空间的一个划分，**构造kd树相当于不断地用垂直于坐标轴的超平面将K维空间切分，构成一系列的K维超矩形区域。**kd树的每个结点对应于一个k维超矩形区域。**利用kd树可以省去对大部分数据点的搜索，从而减少搜索的计算量。**



类比“二分查找”：给出一组数据：[9 1 4 7 2 5 0 3 8]，要查找8。如果挨个查找（线性扫描），那么将会把数据集都遍历一遍。而如果排一下序那数据集就变成了：[0 1 2 3 4 5 6 7 8 9]，按前一种方式我们进行了很多没有必要的查找，现在如果我们以5为分界点，那么数据集就被划分为了左右两个“簇” [0 1 2 3 4]和[6 7 8 9]。

因此，根本就没有必要进入第一个簇，可以直接进入第二个簇进行查找。把二分查找中的数据点换成k维数据点，这样的划分就变成了用超平面对k维空间的划分。空间划分就是对数据点进行分类，“挨得近”的数据点就在一个空间里面。

2 构造方法

（1）构造根结点，使根结点对应于K维空间中包含所有实例点的超矩形区域；

（2）通过递归的方法，不断地对k维空间进行切分，生成子结点。在超矩形区域上选择一个坐标轴和在此坐标轴上的一个切分点，确定一个超平面，这个超平面通过选定的切分点并垂直于选定的坐标轴，将当前超矩形区域切分为左右两个子区域（子结点）；这时，实例被分到两个子区域。

（3）上述过程直到子区域内没有实例时终止（终止时的结点为叶结点）。在此过程中，将实例保存在相应的结点上。

（4）通常，循环的选择坐标轴对空间切分，选择训练实例点在坐标轴上的中位数为切分点，这样得到的kd树是平衡的（平衡二叉树：它是一棵空树，或其左子树和右子树的深度之差的绝对值不超过1，且它的左子树和右子树都是平衡二叉树）。

KD树中每个节点是一个向量，和二叉树按照数的大小划分不同的是，KD树每层需要选定向量中的某一维，然后根据这一维按左小右大的方式划分数据。在构建KD树时，关键需要解决2个问题：

（1）选择向量的哪一维进行划分；

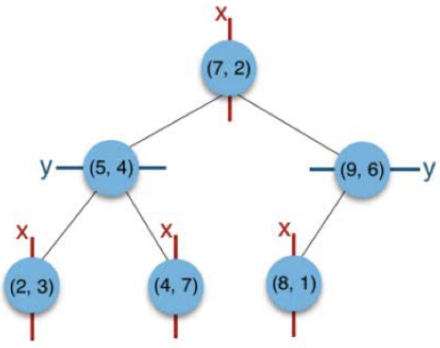
（2）如何划分数据；

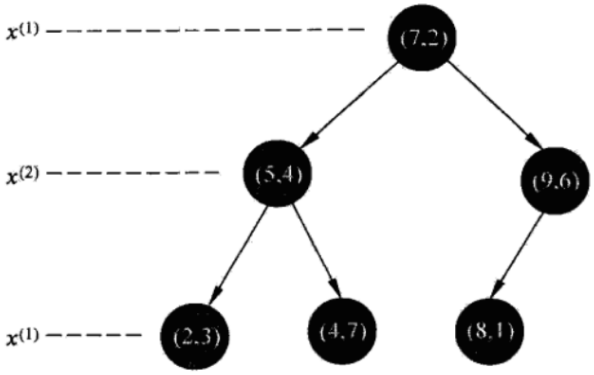
第一个问题简单的解决方法可以是随机选择某一维或按顺序选择，但是**更好的方法应该是在数据比较分散的那一维进行划分（分散的程度可以根据方差来衡量）**。好的划分方法可以使构建的树比较平衡，可以每次选择中位数来进行划分，这样问题2也得到了解决。

3 案例分析

3.1 树结构的建立

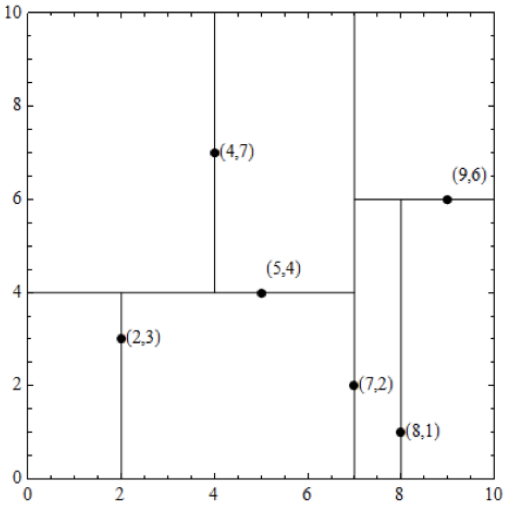
给定一个二维空间数据集：T={(2,3),(5,4),(9,6),(4,7),(8,1),(7,2)}，构造一个平衡kd树。

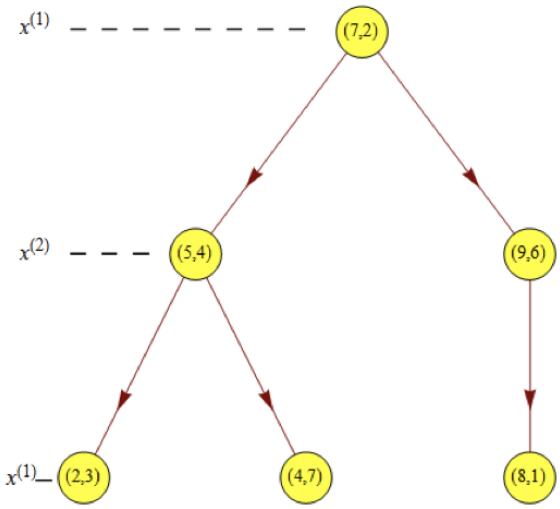




（1）思路引导：

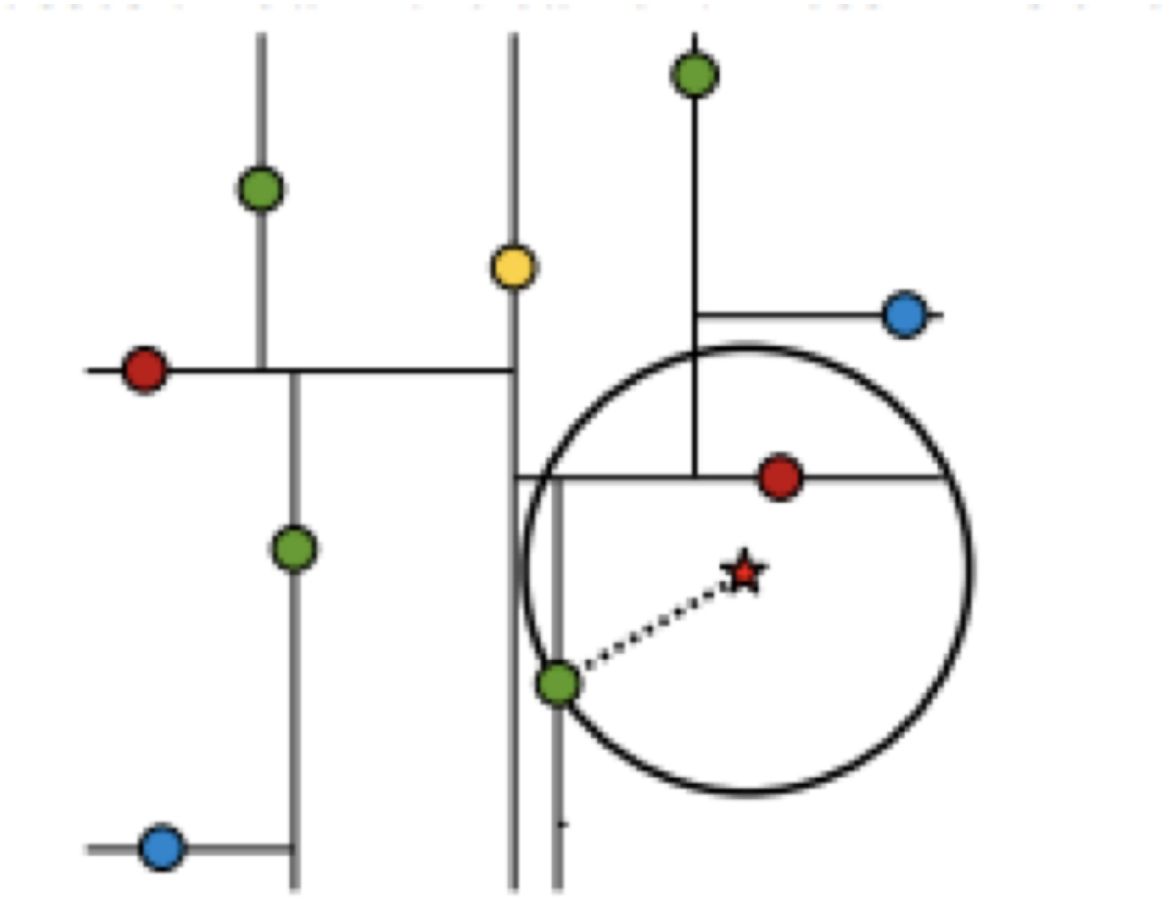
根结点对应包含数据集T的矩形，选择x(1)轴，6个数据点的x(1)坐标中位数是6，这里选最接近的(7,2)点，以平面x(1)=7将空间分为左、右两个子矩形（子结点）；接着左矩形以x(2)=4分为两个子矩形（左矩形中{(2,3),(5,4),(4,7)}点的x(2)坐标中位数正好为4），右矩形以x(2)=6分为两个子矩形，如此递归，最后得到如下图所示的特征空间划分和kd树。





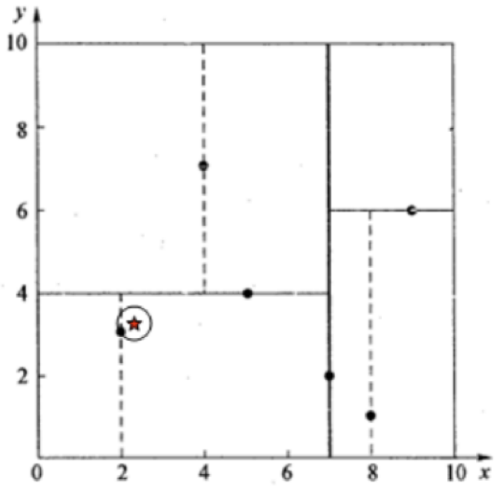
3.2 最近领域的搜索

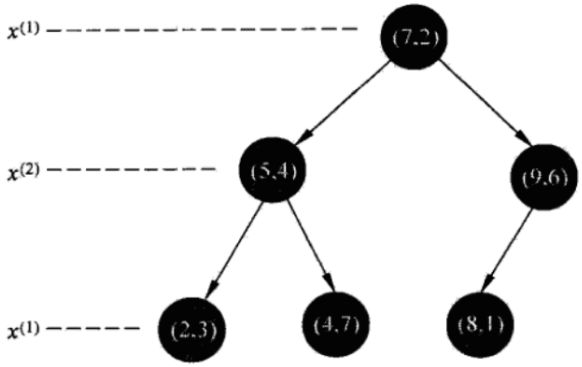
假设标记为星星的点是 test point， 绿色的点是找到的近似点，在回溯过程中，需要用到一个队列，存储需要回溯的点，在判断其他子节点空间中是否有可能有距离查询点更近的数据点时，做法是以查询点为圆心，以当前的最近距离为半径画圆，这个圆称为候选超球（candidate hypersphere），如果圆与回溯点的轴相交，则需要将轴另一边的节点都放到回溯队列里面来。



样本集{(2,3),(5,4), (9,6), (4,7), (8,1), (7,2)}

3.2.1 查找点(2.1,3.1)





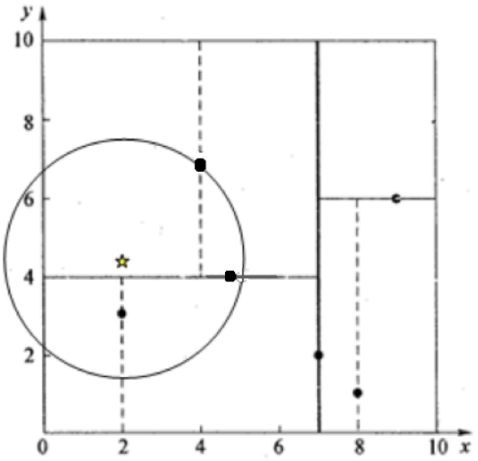
在(7,2)点测试到达(5,4)，在(5,4)点测试到达(2,3)，然后search_path中的结点为<(7,2),(5,4), (2,3)>，从search_path中取出(2,3)作为当前最佳结点nearest, dist为0.141；

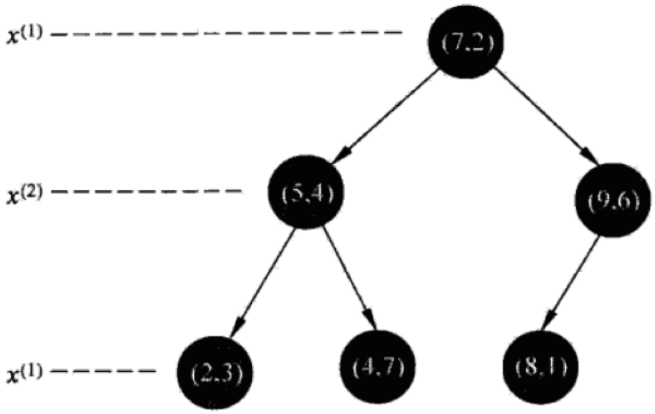
然后回溯至(5,4)，以(2.1,3.1)为圆心，以dist=0.141为半径画一个圆，并不和超平面y=4相交，如上图，所以不必跳到结点(5,4)的右子空间去搜索，因为右子空间中不可能有更近样本点了。

于是再回溯至(7,2)，同理，以(2.1,3.1)为圆心，以dist=0.141为半径画一个圆并不和超平面x=7相交，所以也不用跳到结点(7,2)的右子空间去搜索。

至此，search_path为空，结束整个搜索，返回nearest(2,3)作为(2.1,3.1)的最近邻点，最近距离为0.141。

3.2.2 查找点(2,4.5)





4 总结

首先**通过二叉树搜索**（比较待查询节点和分裂节点的分裂维的值，小于等于就进入左子树分支，大于就进入右子树分支直到叶子结点），**顺着"搜索路径"很快能找到最近邻的近似点**，也就是与待查询点处于同一个子空间的叶子结点；

然后再回溯搜索路径，并判断搜索路径上的结点的其他子结点空间中是否可能有距离查询点更近的数据点，如果有可能，则需要跳到其他子结点空间中去搜索（将其他子结点加入到搜索路径）。

重复这个过程直到搜索路径为空。

9.特征工程-特征预处理-幅度调整

什么是特征预处理?

```
provides several common utility functions and transformer classes to change raw
feature vectors into a representation that is more suitable for the downstream
estimators.
```

翻译过来：通过**一些转换函数**将特征数据**转换成更加适合算法模型**的特征数据过程

两种常用的幅度调整的方式：

- （1）归一化
- （2）标准化

```
#为什么我们要进行归一化/标准化？
特征的单位或者大小相差较大，或者某特征的方差相比其他的特征要大出几个数量级，容易影响（支配）目标结果，使得一些算法无法学习到其它的特征.
```

(1)归一化

通过计算把数据归纳统一到指定范围中去

$x' = \frac{item-min}{max-min}$

$x'' = x' * (mx - mi) + mi$

- item代表每列中索引从0开始到-1的元素
- max为列中的最大值，min为列中的最小值
- mx为选定范围的最大值，mi为选定范围的最小值

a)API使用

```
from sklearn.preprocessing import MinMaxScaler
```

```
#参数：feature_range=(0，1)为归一化调整的范围
sca = MinMaxScaler(feature_range=(1,3))
#fit的数据仅是被保存在内存中，并没有被执行转换
sca.fit(data)
#当数据被transform以后才会被执行归一化
sca_data = sca.transform(data)
```

```
#获取每列中的最大值，返回一个向量
sca.data_max
#讲归一化的数据反转回去
sca.inverse_transform(sca_data)
```

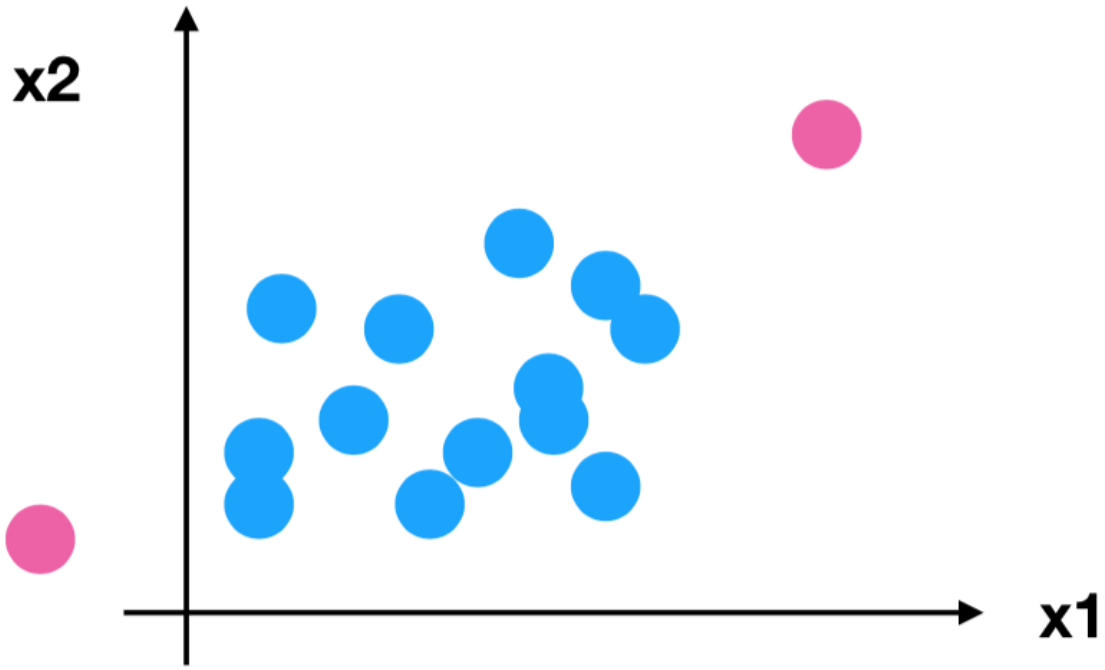
b) 原理解析(使用python模拟实现)

```
class MinMax(object):

    def __init__(self,feature_range=(0,1)):
        self.mi = feature_range[0]
        self.mx = feature_range[1]

    def fit(self,X):
        self.train = np.array(X)
```

```
        #获取列的数量
        col_num = self.train.shape[1]

        self.res = []
        for i in range(col_num):
            min_ = self.train[:,i].min()
            max_ = self.train[:,i].max()
            self.res.append(np.array([min_,max_]))


    def transform(self,X):
        data = np.array(X)
        self.res = np.array(self.res)

        #获取列的数量
        col_num = self.train.shape[1]
        for i in range(col_num):
            x = (data[:,i] - self.res[i,0]) / (self.res[i,1]-self.res[i,0])
            data[:,i] = x*(self.mx-self.mi)+self.mi
        return data
```

c) 归一化总结

注意最大值最小值是变化的，另外，最大值与最小值非常容易受异常点影响，**所以这种方法鲁棒性较差，只适合传统精确小数据场景。**(异常值往往就是极小值和极大值)

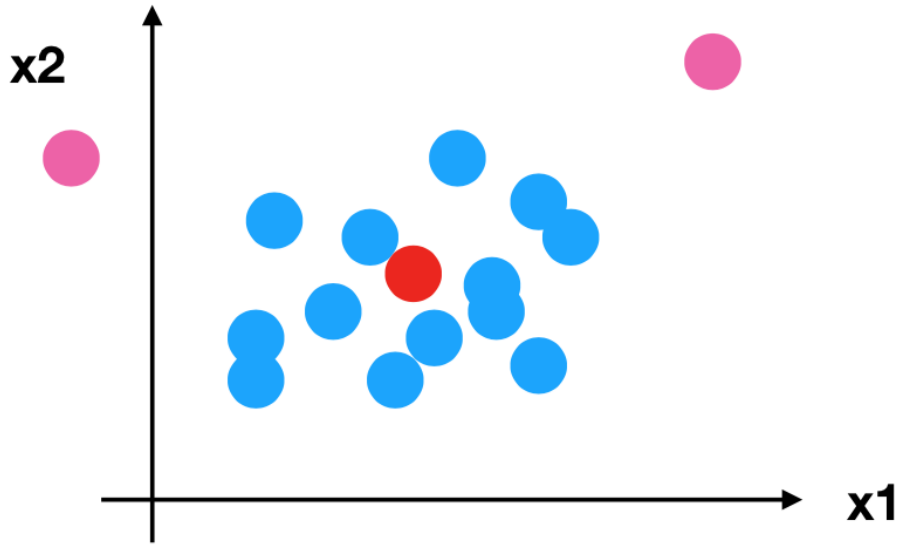


(2)标准化

通过对原始数据进行变换把数据变换到均值为0,标准差为1范围内

- 分子是去中心化$item-mean$代表每个元素减去本列中的均值
- 分母是列中的标准差$\sigma=\sqrt{\frac{\sum_{i=1}^{k}(item-mean)^2}{k}}$
- 整体公式为$X=\frac{item-mean}{\sigma}$
- 对于归一化来说：如果出现异常点，影响了最大值和最小值，那么结果显然会发生改变

- 对于标准化来说：如果出现异常点，由于具有一定数据量，少量的异常点对于平均值的影响并不大，从而方差改变较小。(墨汁并不会影响大海的颜色)



a) API使用

```
from sklearn.preprocessing import StandardScale
#处理之后每列来说所有数据都聚集在均值0附近标准差差为1
std_ = StandardScale()
#fit_transform =  fit() + transform()函数
res = std_.fit_transform(data)

#X:numpy array格式的数据[n_samples,n_features]
#返回值：转换后的形状相同的array
```

b)标准化总结

- 在已有样本足够多的情况下比较稳定，适合现代嘈杂大数据场景。
- 适合动态数据实时分析。

10.网格搜索和交叉验证

1.什么是网格搜索(Grid Search)

通常情况下，**有很多参数是需要手动指定的（如k-近邻算法中的K值），这种叫超参数。**但是手动过程繁杂，所以需要对模型预设几种超参数组合。**每组超参数都采用交叉验证来进行评估。最后选出最优参数组合建立模型。**

K值	K=3	K=5	K=7
模型	模型1	模型2	模型3

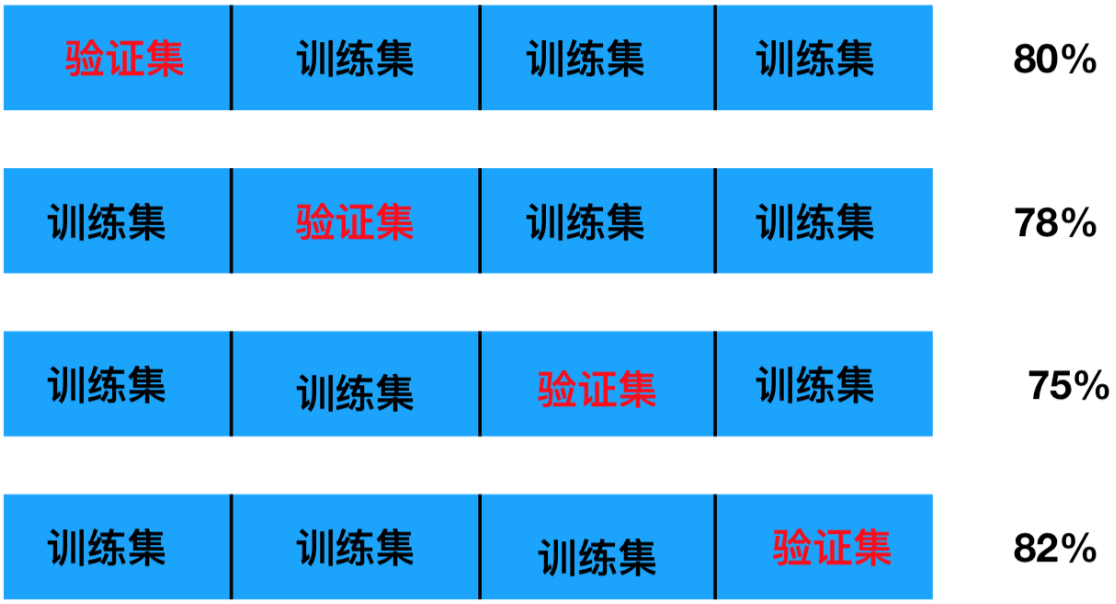
在代码的执行过程中是一个循环的过程，将每个不同的超参数进行笛卡儿积组合，生成大量的模型，记录模型每次的准确率值，并找到准确率比较高的模型。是一种暴力尝试的方法。

2.什么是交叉验证(cross validation)

交叉验证：将拿到的训练数据，分为训练和验证集。以下图为例：将数据分成4份，其中一份作为验证集。然后经过4次(组)的测试，每次都更换不同的验证集。即得到4组模型的结果，取平均值作为最终结果。又称4折交叉验证。

我们之前知道数据分为训练集和测试集，但是**为了让从训练得到模型结果更加准确。**做以下处理

- 训练集：训练集+验证集
- 测试集：测试集



为什么需要交叉验证?

交叉验证目的：**为了让被评估的模型更加准确可信**（泛化性更加好）

3.API调用

```
from sklearn.model_selection import GridSearchCV
gc = GridSearchCV(estimator, param_grid=None,cv=None)
gc.fit(X_train,y_train)

#estimator：估计器对象
#param_grid：估计器参数(dict){“n_neighbors”:[1,3,5]}
#cv：指定几折交叉验证
#n_jobs:指定开启进程数量
#gc.best_score_:在交叉验证中验证的最好结果
#gc,best_estimator_:返回泛化性和准确率比较高的模型
#gc,best_params_:返回最优参数
#gc.cv_results_:每次交叉验证后的验证集准确率结果和训练集准确率结果
```

11.模型的保存和加载

我们再建模完毕以后，并不需要每次都把数据重新执行，而是将数据模型给保留下来

```
from sklearn.externals import joblib
#保存：
joblib.dump(estimator, 'test.m')
#加载：
estimator = joblib.load('test.m')
```