

# 1.机器学习中的熵

## 1.信息量

### 1.1自信息

首先考虑一个离散的随机变量 $x$ ,当我们观察到这个变量的一个具体值的时候,我们接收到多少信息呢?

我们暂时把信息看做在学习 $x$ 的值时候的”惊讶程度”(这样非常便于理解且有意义)。

当我们知道一件必然会发生的事情发生了,比如往下掉的苹果.我们并不惊讶,因为反正这件事情会发生在因此可以认为我们没有接收到信息。

但是要是一件平时觉得不可能发生的事情发生了,那么我们接收到的信息要大得多.

因此,我们对于信息内容的度量就将依赖于概率分布 $p(x)$ .

因此,我们想要寻找一个函数 $h(x)$ 来表示信息的多少且是关于概率分布的单调函数.我们定义:

$$I(x) = -\log_2 p(x)$$

我们把这个公式叫做**信息量**的公式,前面的负号确保了信息一定是正数或者是0.(**低概率事件带来高的信息量**).有时候有人也叫做**自信息** (self-information) 。

## 2.熵 (entropy)

**信息量:** 某个概率分布之下,某个概率值对应的信息量的公式。

**熵:** 整个概率分布对应的信息量的平均值.

$$H[x] = E_{x \sim p}[I(x)] = -E_{x \sim p}[\log_2 p(x)]$$

$$= -\sum_x p(x) \log_2 p(x)$$

$$= -\int p(x) \log_2 p(x) dx$$

信息熵的本质可以看做是某个分布的**自信息的期望**。

熵越大,随机变量的不确定性就越大

### 3. 相对熵 (KL散度)

相对熵又称Kullback-Leibler散度 (即**KL散度**)。

设 $p(x)$ 和 $q(x)$ 是取值的两个概率概率分布, 则 $p$ 对 $q$ 的相对熵为:

$$D(p \parallel q) = \sum_x p(x) \log \frac{p(x)}{q(x)} = E_{p(x)} \left( \log \frac{p(x)}{q(x)} \right)$$

在一定程度上面, 相对熵可以度量两个随机变量的相似程度。当两个随机分布相同的时候, 他们的相对熵为0, 当两个随机分布的差别增大的时候, 他们之间的相对熵也会增大。

### 4. 交叉熵

衡量两个变量之间的差异程度。

$$H(p, q) = - \sum P \log Q$$

**交叉熵与KL散度的关系:**

$$H(p, q) = - \sum P \log Q == - \sum P \log P + \sum P \log P - \sum P \log Q$$

$$= H(P) + \sum P \log P / Q = H(P) + D_{KL}(P \parallel Q)$$

**交叉熵就是信息熵与KL散度的和。**

而信息熵是确定的,与模型的参数 $\theta$ 无关,所以梯度下降求导时, **优化交叉熵和优化kl散度** (相对熵)是一样的;

## 5.互信息 (Mutual Information)

是一个随机变量中包含的关于另一个随机变量的信息量。衡量两个变量之间的相似程度。

定义为：联合分布 和 独立分布乘积 的相对熵。

$$I(X, Y) = D(P(X, Y) \parallel P(X)P(Y))$$

$$= \sum_{x,y} p(x, y) \log \frac{p(x, y)}{p(x)p(y)}$$

## 2. Base tree

### 1. 信息熵，信息增益

#### 1.1 信息熵

$$\text{Ent}(D) = - \sum_{k=1}^{|y|} p_k \log_2 p_k$$

其中P表示事件发生的概率。

#### 1.2 信息增益

用 a 属性对样本D进行划分所获得的"信息增益"(information gain)

$$\text{Gain}(D, a) = \text{Ent}(D) - \sum_{v=1}^V \frac{|D^v|}{|D|} \text{Ent}(D^v)$$

其中D表示按照某个特征分裂之后得到的样本子集内样本个数，V则表示分为样本子集。注意要进行加权。

## 2. ID3决策树

### 2.1 构建过程（原理）

下面以西瓜数据集为例, 该数据集包含17个样本, 用以学习一棵能预测没刨开的是不是好瓜的决策树. 显然 $y = 2$ , 下图中可以看到, 正例 占 8/17, 反例占 9/17,

$$\text{Ent}(D) = - \sum_{k=1}^2 p_k \log_2 p_k = - \left( \frac{8}{17} \log_2 \frac{8}{17} + \frac{9}{17} \log_2 \frac{9}{17} \right) = 0.998$$

编号	色泽	根蒂	敲声	纹理	脐部	触感	好瓜
1	青绿	蜷缩	浊响	清晰	凹陷	硬滑	是
2	乌黑	蜷缩	沉闷	清晰	凹陷	硬滑	是
3	乌黑	蜷缩	浊响	清晰	凹陷	硬滑	是
4	青绿	蜷缩	沉闷	清晰	凹陷	硬滑	是
5	浅白	蜷缩	浊响	清晰	凹陷	硬滑	是
6	青绿	稍蜷	浊响	清晰	稍凹	软黏	是

7	乌黑	稍蜷	浊响	稍糊	稍凹	软粘	是
8	乌黑	稍蜷	浊响	清晰	稍凹	硬滑	是
9	乌黑	稍蜷	沉闷	稍糊	稍凹	硬滑	否
10	青绿	硬挺	清脆	清晰	平坦	软粘	否
11	浅白	硬挺	清脆	模糊	平坦	硬滑	否
12	浅白	蜷缩	浊响	模糊	平坦	软粘	否
13	青绿	稍蜷	浊响	稍糊	凹陷	硬滑	否
14	浅白	稍蜷	沉闷	稍糊	凹陷	硬滑	否
15	乌黑	稍蜷	浊响	清晰	稍凹	软粘	否
16	浅白	蜷缩	浊响	模糊	平坦	硬滑	否
17	青绿	蜷缩	沉闷	稍糊	稍凹	硬滑	否

然后我们计算出当前属性集合{色泽,根蒂,敲声,纹理,脐部,触感} 中每个属性的信息增益.

以属性"色泽" 为例 ,它有三个可能的取值 :{青绿,乌黑,浅白} .

使用该属性对D进行划分, 则可得到 3个子集 ,分别记为 D1 (色泽=青绿) D2(色泽=乌黑) D3(色泽=浅白).

子集D1 包含的编号{1,4,6,10,13,17} , 正例(是)占  $p_1 = 3/6$  ,反例(否) 占  $p_2 = 3/6$  ;

子集D2 包含的编号 {2,3,7,8,9,15} , 正例占  $p_1 = 4/6$  , 反例占  $p_2 = 2/6$  ;

子集D3 包含的编号 {5,11,12,14,16} ,正例占 $p_1 = 1/5$  , 反例占 $p_2 = 4/5$  ;

可计算出"色泽"划分之后所获得的信息熵为:

$$\text{Ent}(D^1) = - \left( \frac{3}{6} \log_2 \frac{3}{6} + \frac{3}{6} \log_2 \frac{3}{6} \right) = 1.000 ,$$

$$\text{Ent}(D^2) = - \left( \frac{4}{6} \log_2 \frac{4}{6} + \frac{2}{6} \log_2 \frac{2}{6} \right) = 0.918 ,$$

$$\text{Ent}(D^3) = - \left( \frac{1}{5} \log_2 \frac{1}{5} + \frac{4}{5} \log_2 \frac{4}{5} \right) = 0.722 ,$$

于是 , 计算出属性 " 色泽"的信息增益为 :

$$\begin{aligned}
 \text{Gain}(D, \text{色泽}) &= \text{Ent}(D) - \sum_{v=1}^{|D^v|} \frac{|D^v|}{|D|} \text{Ent}(D^v) \\
 &= 0.998 - \left( \frac{6}{17} \times 1.000 + \frac{6}{17} \times 0.918 + \frac{5}{17} \times 0.722 \right) \\
 &= 0.109 .
 \end{aligned}$$

类似的，我们可计算出其他属性的信息增益：

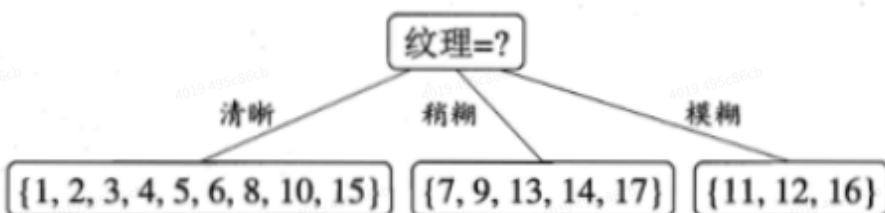
$$\text{Gain}(D, \text{根蒂}) = 0.143; \quad \text{Gain}(D, \text{敲声}) = 0.141;$$

$$\text{Gain}(D, \text{纹理}) = 0.381; \quad \text{Gain}(D, \text{脐部}) = 0.289;$$

$$\text{Gain}(D, \text{触感}) = 0.006.$$

[https://blog.csdn.net/qq\\_41661805](https://blog.csdn.net/qq_41661805)

显然这里  $\text{Gain}(D, \text{纹理}) = 0.381$  信息增益最大，于是他被选为划分属性。根据属性将样本分为多个子集，在子集中继续重新划分，直到叶节点）



以图中的一个分支节点("纹理= 清晰")为例，该节点包含的样例集合D1中有编号{1,2,3,4,5,8,10,15} 的9 个样例,可用的属性集合为 { 色泽,根蒂,敲声,脐部,触感}; 基于 D1计算出各属性的信息增益:

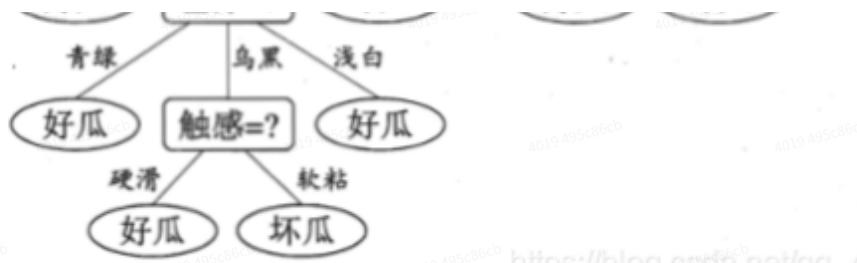
$$\text{Gain}(D^1, \text{色泽}) = 0.043; \quad \text{Gain}(D^1, \text{根蒂}) = 0.458;$$

$$\text{Gain}(D^1, \text{敲声}) = 0.331; \quad \text{Gain}(D^1, \text{脐部}) = 0.458;$$

$$\text{Gain}(D^1, \text{触感}) = 0.458.$$

"根蒂" , "脐部" , "触感" 3 个属性均取得最大的信息增益,可用选择其中一个作为划分属性,最终得到:





[https://blog.csdn.net/qq\\_41661](https://blog.csdn.net/qq_41661)

## 2.2 ID3的缺点

1. id3是多叉树，效率较低，并且只能处理离散特征；

2. 信息增益的衡量方式非常容易偏向取值数量特别多的特征.

对于id3来说，依据信息增益的原则，取值越多的特征，会切分的越细，即每个分支的数据越少，因为每一个分支的数据越少，每一个节点的“纯度”会越高，整体的信息增益越大。

3. 信息熵的计算比较涉及到求和和对数变换，比较费时.

## 3.C4.5

### 3.1 C4.5的特点

#### 3.1.1 信息增益率

针对信息增益对取值数目多的特征有偏好的问题，使用信息增益率替代信息增益

$$\text{Gain\_ratio}(D, a) = \frac{\text{Gain}(D, a)}{\text{IV}(a)}$$

分子部分就是信息增益没有变，分母部分是：

$$\text{IV}(a) = - \sum_{v=1}^V \frac{|D^v|}{|D|} \log_2 \frac{|D^v|}{|D|}$$

### 3.1.2 启发式方法

但是信息增益率存在的问题在于，**会对取值很少的特征有所偏好**，举个极端的例子，假设某个特征取值完全相同，则分母IV(a)的计算结果为0，则信息增益率为无穷大。

先从候选划分特征中找到**信息增益高于平均值的特征**

再从中**选择增益率最高的**

## 3.2 C4.5的缺点

1. C4.5和id3一样用的是**多叉树**，效率较低，**用二叉树效率更高**；
2. C4.5的信息增益率计算和信息熵一样都**比较计算复杂而麻烦**。

## 4. Cart

### 4.1 Cart的改进

1. Cart摒弃了麻烦的**多叉树**，而使用**二叉树**进行替代；
2. Cart使用了**gini指数**作为分裂标准。

### 4.2 基尼系数

基尼指数代表了模型的不纯度，基尼系数越小，不纯度越低，特征越好。这和信息增益（率）正好相反。

$$Gini(p) = \sum_{k=1}^K p_k(1-p_k) = 1 - \sum_{k=1}^K p_k^2$$

基尼指数的问题在于偏向于特征值较多的特征，对于特征值较多的特征比较容易算出高的gini值，这个缺点和信息增益是一样的。

实例：二分类

假设某个节点上的样本全是1，则其pk=1，gini=0；如果节点上的样本全是0，则pk=0，gini=0。如果节点上的样本一半是1，一半是0，则pk=0.5，gini=0.5\*0.5=0.25达到最大，此时混沌程度最大，划分了等于没划分；

## 5.剪枝策略

### 5.1预剪枝

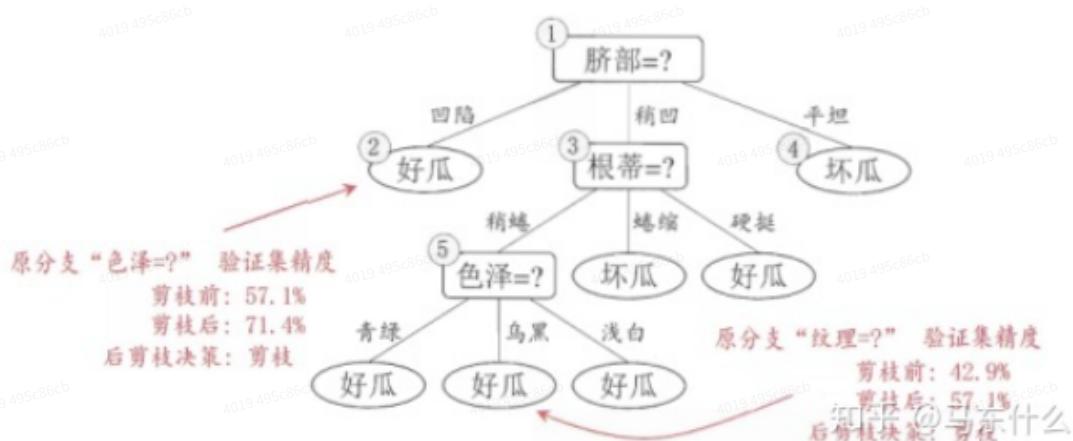
通过tree的最大深度或者叶节点的最小样本数量等超参数的调节就可以达到预剪枝的效果。但有可能带来会带来欠拟合的风险。

### 5.2后剪枝

引入验证集，对每一个叶子节点的父节点进行删除，然后观察验证集上的精度变化，例如下图：



去掉纹理这个叶子节点的父节点后：



验证集的精度梯度提高，因此进行剪枝。

## 6. 决策树面试版

### 6.1 原理

Date:

Page:

#### 3.1. 决策树

##### • 原理：

构建决策树每一轮迭代中，都会根据信息增益的原则选择出信息增益最大的特征进行分裂，将样本分成多个子集，然后对每个子集依次迭代，直至到达叶节点。（而叶节点就是我们的分类标准）

### 6.2 如何进行特征选择

##### • 如何进行特征选择：

根据信息增益准则的特征选择方法，对训练数据集 D，计算其每个特征的信息增益，并比较它们的大小，选择信息增益最大的特征。信息增益，就是信息熵的减少。

信息熵是一种对信息不确定性的度量，越大，不确定性越大，越无序。而在分类器的构建中，我们希望将无序的数据变有序，则需要减少信息熵。

### 6.3 损失函数

• 损失函数：包含剪枝的决策树的损失函数

$$L(T) = \sum_{t=1}^{|T|} N_t H_t(T) + \alpha |T|$$

决策树 T 的叶子节点个数为 |T|

它是树 T 的一个叶子节点，该叶节点有  $N_t$  个样本点。

$H_t(T)$  为叶节点的熵。

2. 正则化惩罚，此时  $J(\theta)$  为

$$J(\theta) = \sum_{t=1}^{|T|} M_t H_t(\theta) + \lambda |\theta|$$

$$\Leftrightarrow L(w, d) = \frac{1}{N} \sum_{i=1}^N (y_i - f(x_i))^2 + r(d)$$

## 6.4 回归树

• 回归树：

过程：(1) 将预测变量空间  $(x_1, x_2, x_3, \dots, x_p)$  的可能取值构成的集合分成  $J$  个互不重叠的区域  $\{R_1, R_2, R_3, \dots, R_J\}$ 。

(2) 对落入选区  $R_j$  的每个观测值作同样的预测，预测值等于  $R_j$  上训练集的新样本取值的算术平均数。

如：在第(1)步中，得到两个区域  $R_1$  与  $R_2$ ， $R_1$  中训练集的多个样本取值的算术平均数为 10， $R_2$  中训练集的多个样本取值的算术平均数为 20。

则，对给定的观测值  $x=x$ ，若  $x \in R_1$ ，给出的预测值为 10，若  $x \in R_2$ ，则预测值为 20。

 扫描全能王 创建

回归树：

在训练数据集所在的输入空间中，递归地将每个区域划分成两个子区域，并决定每个区域上的输出值，构建二叉决策树。

(1) 选择最优切分变量  $j$  与切分点  $s$ ，求解：(与  $\theta$  为对应的两个类别)

$$\min_{j, s} \left[ \min_{C_1} \sum_{x_i \in R_1(j, s)} (y_i - C_1)^2 + \min_{C_2} \sum_{x_i \in R_2(j, s)} (y_i - C_2)^2 \right]$$

遍历变量), 对固定树切分变量], 扫描切分点  $s$ , 选择使得与  $y$  的差的绝对值的对  $(y, s)$  [ $y$  是扫描一个输入区间,  $s$  是对空间的划分点]

(2) 用选定的  $(y, s)$  划分区域并决定相应的输出值:

$$R_1(y, s) = x \mid x^{(1)} \leq s, \text{ 在区间 } y \text{ 内的输入 } x, \text{ 若小于 } s, \text{ 则输出 } R$$

$$R_2(y, s) = x \mid x^{(1)} > s, \dots, \dots, \dots, \text{ 大于 } s, \dots, \dots, R$$

$$\hat{c}_m = \frac{1}{N} \sum_{x \in R_m(y, s)} y_i, x \in R_m, m=1, 2 \quad (\text{下个类别是})$$

(3) 继续对两个子区域调用步骤(1)与(2), 直到满足停止条件.

## 6.5 优缺点

· 优点:

1) 计算复杂度不高

2) 可以对 树形结构表示, 容易被理解

3) 决策树的预测准确性一般, 回归和分类都弱,  
但可以通过集成学习方法组合大量决策树, 显著提升预测效果.

# 3. 随机森林

## 3.1 随机森林原理

Date: \_\_\_\_\_  
Page: \_\_\_\_\_  
CART (Classification and Regression Trees)

### 26. 随机森林

通过对训练数据样本进行有放回的抽样。每次抽取部分样本，由此生成一棵CART树，剩下样本信息作为额外数据，用来作验证集计算额外误差测试模型。

然后把抽取的样本信息再放回到原始数据集中，重新抽取一组训练数据，再以此训练数据集生成一棵CART树。

这样依次生成的棵CART树，多棵树组成森林，并且他们生成都是通过随机采样的训练数据生成，因此叫做随机森林。

RF可以用于数据的回归，也可以用于数据的分类。  
回归时是由多棵树的预测结果求平均值；分类是由多棵树的预测结果进行投票。

## 3.2 随机森林优缺点

### 14. 随机森林优缺点

- 1) 优点：
- ① 综合表现性能较好，在分类效果上相对于其他算法有较大优势
  - ② RF能处理很高维度数据，并且不用做特征选择，在训练完之后随机森林能给出哪些特征比较重要。
  - ③ 抗干扰能力强：
    - 1) 对异常值不敏感
    - 2) 在不平衡数据集上，RF可以平衡误差。

④ RF并行运行，训练速度快

⑤ RF能做回归与分类

2) 缺点：

- ① 在某些噪音较大的分类或回归问题上会过拟合。

- ② 对于小样本高维度数据效果不佳。

# 4. Adaboost

## 4.1 Adaboost 实现原理

### 1) Adaptive Boosting (自适应增强)

它的自适应在于：前一个基本分类器分错的样本会得到加强，加权后整体样本再次被用来自训练下一个基分类器。

同时，在每一轮中加入一个新弱分类器，直到达到某个预定的足够小的错误率或达到预先指定的最大迭代次数。

### 具体解聚

① 初始化训练数据的权值分布。如果有  $N$  个样本，则每个训练样本最开始时都被赋予相同的权值： $1/N$ 。

② 训练弱分类器。如果某个样本点已经被准确地分类，那么在构造下一个训练集中，它的权值就会降低。相反，如果该样本点分类错误，那么它的权值就会提升，然后，权值更新过后的样



扫描全能王 创建

### 根据弱分类器 保养新模型 强分类器 与样本权值

Date:

Page:

本集被用于训练下一个分类器，整个训练过程如此迭代地进行下去。

③ 将多个弱分类器组合成强分类器。各个弱分类器训练结束后，加大分类误差小的弱分类器的权值，使其在最终的分类函数中起着较大的决定作用。  
而降低分类误差大的弱分类器的权值，使其在最终的分类函数中发挥较小作用。

## 4.2 Adaboost 做具体实现

### 3.3. AdaBoost

boosting家族需要解决的问题

- 1) 如何计算弱分类器?
- 2) 如何得到弱分类器权值?
- 3) 如何更新样本权值?
- 4) 使用何种损失函数?

#### 4.2.1 分类

AdaBoost 损失函数: **指数函数**

即该损失函数为:

$$\arg \min_{d, g} \sum_{i=1}^m \exp(-y_i f_k(x_i))$$

m 表样本数  
 $y_i$  样本真实标签  
 $f_k(x_i)$ : 弱分类器的预测结果

##### 1) 误差率 $\epsilon$

1) 误差率  $\epsilon$ :

$\epsilon_k$ : 第  $k$  个弱分类器  $g_k(x)$  在训练集上的加权误差率为:

$$\epsilon_k = P(G_k(x_i) \neq y_i) = \sum_{i=1}^m w_i I(G_k(x_i) \neq y_i)$$

↓  
 样本数

[回归无关]

[离散分类]

##### 2) 弱分类器权重

2) 弱分类器权重:

$$\alpha_k = \frac{1}{2} \log \frac{1 - \epsilon_k}{\epsilon_k}$$

$\alpha_1, \alpha_2, \dots$

##### 3) 更新样本权重

3) 更新样本权值:

$$w_{k+1, i} = \frac{w_{k,i}}{z_k} \exp(-\alpha_k y_i g_k(x_i))$$

##### 4) 集成策略

4) 集分策略：加权表决法，最终的强分类器

$$f(x) = \text{sign} \left( \sum_{k=1}^K \alpha_k G_k(x) \right)$$

## 4.2.2 回归

R2回归流程：

R2回归算法流程：

输入为样本集  $T = \{(x_1, y_1), (x_2, y_2), \dots, (x_m, y_m)\}$ ，  
弱学习器数量，弱学习器迭代次数  $K$ 。

输出：最终的强学习器  $f(x)$

1) 初始化样本集权重：

$$\mathbf{D}(1) = (w_{11}, w_{12}, \dots, w_{1m}) : w_{1j} = \frac{1}{m}, j=1, 2, \dots, m$$

2) 对于  $k=1, 2, \dots, K$ ：

a) 使用具有权重  $\mathbf{D}_k$  的样本集来训练数据，得到弱分类器  $G_k(x)$

★ b) 计算训练集的最大误差

$$E_k = \max |y_i - G_k(x_i)| \quad (i=1, 2, \dots, m) \quad (\text{最大误差率})$$

★ c) 调每个样本的相对误差：

$$\cdot \text{绝对误差: } e_{ki} = \frac{|y_i - G_k(x_i)|}{E_k} \quad \cdot \text{平方误差: } e_{ki} = \frac{(y_i - G_k(x_i))^2}{E_k^2}$$

★ d) 计算回归误差率：

$$e_k = \sum_{i=1}^m w_{ki} e_{ki}$$

• 指数误差:

$$e_{ki} = 1 - \exp \left( \frac{-|y_i - G_k(x_i)|}{E_k} \right)$$

e) 调弱分类器的系数：

$$\alpha_k = \frac{e_k}{1 - e_k}$$

f) 更新样本集的权重分布：

$$W_{k+1,i} = \frac{W_{ki}}{Z_k} \alpha_k^{1 - e_{ki}}$$

$$Z_k = \sum_{i=1}^m W_{ki} \alpha_k^{1 - e_{ki}}$$

$$f(x) = \sum_{k=1}^K \left( \ln \left( \frac{1}{\alpha_k} \right) \right) g_k(x)$$

其中  $g_k(x)$  是所有  $\underline{\alpha_k G_k(x)}$ ,  $k=1, 2, \dots, K$  的中位数。

但有点看不懂依据，还是采用加权平均法：

$$f(x) = \sum_{k=1}^K \left( \ln \frac{1}{\alpha_k} \right) G_k(x)$$

当然，也可以自定义租房策略。

## 背诵版

回答一个问题：

Adaboost 做回归原理：[模型输出不一致  $\Leftarrow$  误差增加方式不一致]

过程与做分类一致：

首先初始化所有样本权重相同；

(MSE)

接下来，训练版个基学习器，计算在训练集上相对误差，然后计算

每个样本的相对误差，最后得到样本加权误差。

并根据这个误差率来设置这个弱学习器的权重和对应训练

样本权。总体还是误差率高，弱学习器权重大，对应样本权重高。

部分由地方法训练若干基学习器组合为强学习器。

## 4.3 Adaboost 优缺点

## ⑥ AdaBoost 优缺点

AdaBoost 的学习器是：决策树与神经网络。

对于决策树：AdaBoost 分类用 CART 分类树，

AdaBoost 回归用了 CART 回归树

- 优点：
- 1) 作为分类器时，分类精度很高
  - 2) 可以集成同时分类与回归
  - 3) 不易发生过拟合



扫描全能王 仓

Date: / / Page: /

缺点：

对样本敏感，异常样本在迭代中可能会获得较高权值，最终影响强学习器的预测准确性。

# 6. Gradient Boosting Decision Tree (GBDT)

## 1.gbd介绍

### 1.1 gbd算法原理

模型的公式化：

$$f(x) = f_0(x) + \sum_{t=1}^T learningrate * f_t(x)$$

最初的设计并不复杂， $f_0(x)$  表示初始的基学习器，后面那一项表示后续拟合的所有其它基学习器。

目标函数：

$$Obj = \sum_{i=1}^n l(y_i, \hat{y}_i)$$

### 1.2. 实例

假设我们原始的label是：10.8 ,5.0, 4.3, 2.5, 8.9,

而我们预测出的label是：10.5, 5.3, 4.4, 2.6, 9.0,

我们计算一下二者的差值，看看当前的模型的预测误差是怎么样的：

$$0.3, -0.3, -0.1, -0.1, -0.1$$

既然原始的tree存在误差，那么我们不可以以原始的特征矩阵为特征，**误差作为新的标签**来训练一颗新tree，然后**直接对二者进行求和**，不就能神奇的降低误差了吗？我们上述的计算结果称之为**残差**，注意，残差的计算方向是——真实标签-预测结果，

而我们前面的计算，实际上 **残差就是推广来看就是平方损失函数的负梯度**：

$$L(y, F(x)) = \frac{1}{2} (y - F(x))^2$$

上式为平方损失函数，我们对其进行求导可得：

$$\frac{\partial J}{\partial F(x_i)} = \frac{\partial \sum_i L(y_i, F(x_i))}{\partial F(x_i)} = \frac{\partial L(y_i, F(x_i))}{\partial F(x_i)} = F(x_i) - y_i$$

其一阶梯度，取负号就可以得到**负梯度**为  $y_i - f(x_i)$ ，也就是我们上面例子里进行的计算。

这样一来，推而广之，**分类问题**也能很好的理解了，例如对于二分类问题，计算二元**交叉熵的负梯度**，将负梯度作为下一轮的tree进行拟合时使用的标签。

### 1.3 正则化手段（学习率）

gbdt引入了所谓shrinkage的正则化手段对模型进行约束，说白了就是引入了学习率。

$$f(x) = f_0(x) + \sum_{t=1}^T learningrate * f_t(x)$$

即上式中的learningrate，模型之间的组合不再是直接相加，而是乘上学习率

一方面（引入学习率的通用作用）：**引入合适的学习率可以更快更好的收敛到局部或全局最优**，学习率太小收敛太慢虽然也能保证达到最优，学习率太大容易发生震荡；

**另一方面：学习率的另一个重要作用是（tree独有），增大学习率之后，我们的base tree的数量会增加**，实际上就是引入了更多的tree来进行共同决策，比如学习率0.1的时候我们可能100棵tree就stop了，而学习率为0.001的时候就可能需要10000棵tree才stop，这样**整个gbdt的决策就是由于更多的base tree来共同支持，预测的稳定性更好，更不容易过拟合**。

举个极端的例子，假设gbdt只有一棵base tree，则整个gbdt的稳定性是很差的，完全由这一棵tree来决定，此时退化为普通的决策树，泛化性能大大下降。

## 2.gbdt的优点

### 1. 低偏差，拟合能力强。

同时加入了rf的采样的思想后（早期的gbdt没有引入这样的思想），gbdt在**方差和偏差**方面都有非常好的表现，是目前在各个领域的一种非常常用和流行的算法，尤其是结构化数据的领域，例如典型的**风控领域**，gbdt可以说是统治了半壁江山。

## 2. 在训练的过程中自动进行特征选择；

将上游的特征工程和下游的模型训练灵活的结合在一个训练过程中；(嵌入式)

## 3. 能较好的处理非线性问题；

## 4. 能够适应多种损失函数；

只要你能将实际的问题抽象成一个有监督问题，并且其损失函数是可导的

## 3.gbdt的缺点

### 3.1 难以处理高维稀疏的数据

切分增益很小没有太大意义

假设1000万个样本，高维稀疏的情况下，某个稀疏特征的0有9999900个，而1只有100个，此时切分出来左枝的样本有9999900个样本，右枝的样本只有100个，这个时候tree切分的增益是非常小的，不平衡的切分和不切分没什么区别。

### 3.2 对于异常点较为敏感

注意是异常点不是异常值，对于特征层面异常值，gbdt一点儿也不敏感，因为tree本身是基于特征的相对排序性来分裂的，特征是1 2 3 4 5还是1 2 3 4 100，分裂的时候没有区别。

因为异常点本身比较难有一个好的超平面去拟合，导致了对于这类样本的预测误差往往比较大，gbdt会在异常导致预测误差特别大地样本上不断地去用新的tree来拟合，导致模型太过拟合异常样本，最终的结果就是泛化性能。

### 3.3 集成模型本身的计算复杂度都是比较高的，训练耗时。

## 4. 面试问题

### 1. 为什么GBDT会使用回归树而不是分类树

Decision Tree: CART 回归树：

无论是处理回归问题还是分类问题，都是使用CART 回归树。

为什么不用CART 分类树？

因为GBDT每次迭代要拟合的是梯度值，是连续值所以要回归树。

### 2. Adaboost与GBDT差异

## 问题：Adaboost 与 GBDT 的差异：

1) Adaboost 通过修改样本权值与学习器权重，来提升模型准确率。

2) GBDT：通过训练其学习器拟合模型的残差项……

# 7. Extreme Gradient Boosting(XGBoost)

XGBoost是Extreme Gradient Boosting（极限梯度提升）的缩写，它是基于决策树的集成机器学习算法，大体上沿袭了之前说过的gbdt的框架，但是在此之上做了很多的改进

## 1. 相对于GBDT的改进

### 1.1 正则化概念的引入

传统的gbdt对模型进行优化的方法就是引入学习率来缓解模型过拟合的问题。

xgboost则引入了树的正则化的方法，在原始的代价函数上加入对模型的复杂度的定义作为代价函数的补充项从而生成最终的目标函数，

$$Obj = \sum_{i=1}^n l(y_i, \hat{y}_i) + \sum_{k=1}^K \Omega(f_k)$$

其中：

$$\Omega(f_t) = \gamma T + \frac{1}{2}\lambda \sum_{j=1}^T w_j^2$$

T表示叶子节点的数量；gamma则是这一个惩罚项的超参数，人工进行设置；  
w表示叶子节点的权重，也就是叶子节点的值。

第一项：希望叶子节点的数目能够少一点。

第二项：wi表示叶子节点的权重，和逻辑回归一样，我们希望权重整体是偏小的，因为过大的权重会给预测结果带来很大的不稳定性。

其实抛开前面的gamma\*T，后面这项的定义式和l2正则化的形式是完全一样的。

### 1.2 梯度信息的使用

原始的gbdt仅仅使用了一阶梯度的信息，而xgboost使用了一、二阶梯度。

Xgboost对损失函数实际上进行了二阶泰勒展开，这里要注意，我们所说的二阶泰勒展开是针对于gbdt原始的损失函数进行展开的，不包括tree的正则项的部分。

## ① 定义

· 目标:  $\text{obj}^{(t)} = \frac{1}{n} \sum_i L(y_i, y_i^{(t-1)} + f_t(x_i)) + \lambda f_t + \text{constant}$

· 用泰勒展开式近似

· 泰勒展开式:  $f(x+\Delta x) \approx f(x) + f'(x)\Delta x + \frac{1}{2}f''(x)\Delta x^2$

· 定义:  $g_i = \partial g_{(t-1)} / \partial (y_i, y_i^{(t-1)})$ ,  $h_i = \partial^2 g_{(t-1)} / \partial (y_i, y_i^{(t-1)})$

$$\text{obj}^{(t)} \approx \frac{1}{n} \left[ L(y_i, y_i^{(t-1)}) + g_i f_t(x_i) + \frac{1}{2} h_i f_t^2(x_i) \right] + \lambda f_t + \text{constant}$$

+ 学习率

泰勒公式的作用: 用多项式函数去逼近光滑函数。

这里xgboost所做的就是直接对原始的损失函数进行二阶泰勒展开, 舍弃其误差项, 将展开之后的新损失函数作为模型训练过程中使用的损失函数。

## 2. XGBoost与GBDT的不同

### · XGBoost VS GBDT

1) 正则项: XGBoost 显式地加入正则项来控制模型的复杂度, 有利于防止过拟合, 从而提高模型的泛化能力。

2) 学习率: GBDT 在模型训练时, 只使用了代价函数的一阶导数信息, XGBoost 对代价函数进行二阶泰勒展开, 可以同时使用一阶与二阶导数。

3) 采样: 传统的 GBDT 在每轮迭代时, 会使用全部的数据; XGBoost 则采用了与随机森林相似的策略, 支持对数据进行采样。

### 4) 支持线性分类器

传统GBDT以CART作为基分类器, xgboost还支持线性分类器, 这个时候xgboost相当于带L1和L2正则化项的逻辑斯蒂回归 (分类问题) 或者线性回归 (回归问题)

## 3.XGBoost的缺点

### 1. 难以处理高维稀疏的数据

因为tree本身的正则化对于高维稀疏的数据情况不像l1正则化能够带来有效的约束，并且tree本身的分裂在稀疏的情况下显著性很差；

### 2. 对于异常点较为敏感

因为gbdt会在因为异常导致预测误差特别大地样本上不断地去用新的tree来拟合，导致模型太过拟合异常样本，最终的结果就是泛化性能差；例如在回归问题中，假设标签是【1.6, 2.6, 3.5, 1.5, 1000】，这种情况下gbdt会不断的用新tree去拟合标签为1000的样本的负梯度。

### 3. 集成模型本身的计算复杂度都是比较高的，训练耗时

## 4.为什么xgb做二阶泰勒展开不做三阶泰勒或者n阶泰勒展开？

实际上这是精度和工程性能上的一个trade off。

我们知道，当对函数进行n阶展开的时候，n趋近于无穷大的时候，精度也是趋近于无穷大的，但是这也意味着计算量的增大，我们原来只要计算一阶和二阶梯度，如果引入三阶梯度，则又需要额外去计算三阶梯度，如果引入四阶梯度则。。。，显然，对于gbdt这样动辄成百上千棵tree的算法来说，这无疑大大增加了gbdt的模型训练的复杂度，并且对于gbm的框架来说，单个基学习器的少量误差并不是那么重要，精度不够，tree的数量来凑就好了~

# 8. bagging与boost对比

## 1). boosting 与 bagging区别

1) 样本选择上：Bagging：训练集在原始集中有放回的选取，从原始集中选出的各轮训练集之间是独立的。

Boosting：每一轮的训练集不变，只是训练集中每个样例在分类器中的权重发生变化。而权值是根据上一轮的分类结果进行调整。

2) 样例权值：Bagging：使用均匀抽样，每个样例的权值相等。

Boosting：根据错误率不断调整样本权值，错误率越大，权值越大。

3) 预测函数：Bagging：所有预测函数的权值相等。

Boosting：每个弱分类器都有权值。

对于分类误差小的分类器会有更大的权值。

4) 并行性：Bagging：多个预测函数可以并行。

Boosting：只能顺序生成，因为一个模型需要前一轮模型的结果。

B. 随机森林与GBDT区别：

1) 训练集的选取：随机森林采用 Bagging 思想，GBDT 采用 Boosting 思想；

这两种方法都是 Booststrap 思想的应用。

(从一个数据集中有放回地抽取 N 次，每次抽 M 个)

但 Bagging 是有放回地均匀抽样。

Boosting 根据错误率来取样。

2) 组成随机森林的树可以是分类树，也可以是回归树。

GBDT 只能由回归树组成。

3) 组成随机森林的树可以并行生成；而 GBDT 只能串行生成。

4) 对于最终的输出结果，随机森林采用投票法：

GBDT 采用加权累加。

5) RF 对异常值不敏感，GBDT 对异常值敏感。

6) 随机森林对训练集一视同仁；GBDT 是基于权值的弱模型的集成。

7) RF 是通过减少模型偏差提高性能；GBDT 是通过减少模型偏误提高性能。

# 6. LGB（一）原理

## 1. LightGBM简介

GBDT (Gradient Boosting Decision Tree) 是机器学习中一个长盛不衰的模型，其主要思想是利用弱分类器（决策树）迭代训练以得到最优模型，该模型具有训练效果好、不易过拟合等优点。

而LightGBM (Light Gradient Boosting Machine) 是一个实现GBDT算法的框架，支持高效率的并行训练，并且具有更快的训练速度、更低的内存消耗、更好的准确率、支持分布式可以快速处理海量数据等优点。

### 1.1 LightGBM提出的动机

常用的机器学习算法，例如神经网络等算法，都可以以mini-batch的方式训练，训练数据的大小不会受到内存限制。

而GBDT在每一次迭代的时候，都需要遍历整个训练数据多次。如果把整个训练数据装进内存则会限制训练数据的大小；如果不装进内存，反复地读写训练数据又会消耗非常大的时间。尤其面对工业级海量的数据，普通的GBDT算法是不能满足其需求的。

LightGBM提出的主要原因就是为了解决GBDT在海量数据遇到的问题，让GBDT可以更好更快地用于工业实践。

### 1.2 XGBoost的缺点

**XGBoost缺点：**每一次都需要遍历所有的特征，然后计算每一个特征的信息增益(涉及对每一个样本的计算)，然后选择最好的一个特征进行分裂，一次分裂计算需要计算 $N_{samples} * N_{features}$ 。

这样的预排序算法的优点是能精确地找到分割点。但是缺点也很明显：

#### 1.2.1 空间消耗大

这样的算法需要保存数据的特征值，还保存了特征排序的结果（例如，为了后续快速的计算分割点，保存了排序后的索引），这就需要消耗训练数据两倍的内存。

#### 1.2.2 时间上也有较大的开销

在遍历每一个分割点的时候，都需要进行分裂增益的计算，消耗的代价大。

## 1.3 LightGBM的优化

为了避免上述XGBoost的缺陷，并且能够在不损害准确率的条件下加快GBDT模型的训练速度，lightGBM在传统的GBDT算法上进行了如下优化：

- LightGBM采用直方图算法将遍历样本转变为遍历直方图，极大的降低了时间复杂度；
- LightGBM 在训练过程中采用**单边梯度算法**过滤掉梯度小的样本，减少了大量的计算；
- LightGBM 采用了**互斥特征捆绑算法**减少特征的维数。
- LightGBM 采用优化后的**特征并行、数据并行方法**加速计算，当数据量非常大的时候还可以采用投票并行的策略；

## 2.LightGBM的基本原理

### 2.1 基于Histogram(直方图)的决策树算法

#### 2.1.1基本思想

一个特征 对应 一个直方图。

将连续值，转为为多个离散区间存储。（将特征的值做一次映射）

对于一个特征，先把连续的浮点特征值离散化成 k个整数，构造一个宽度为k 的直方图。

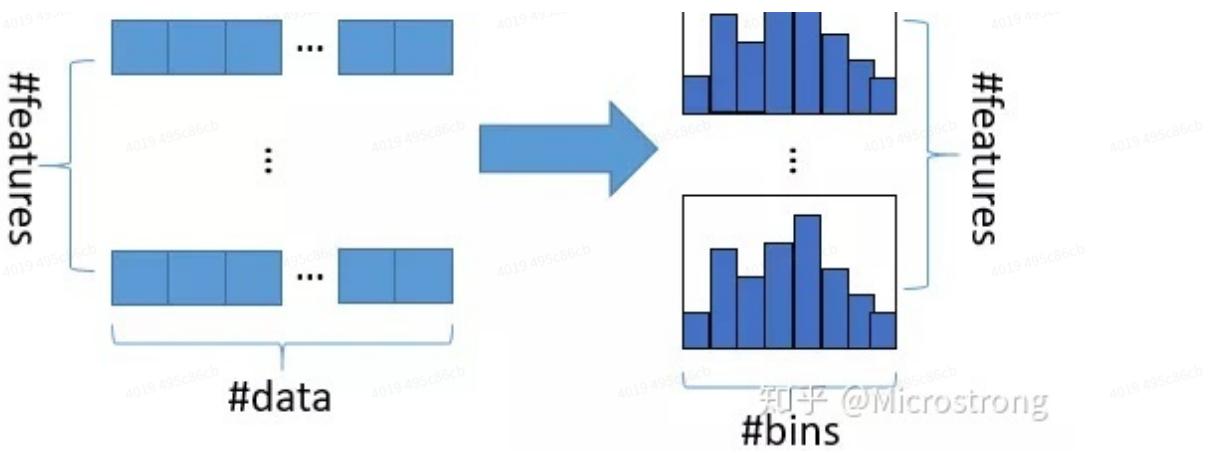
在遍历数据的时候，根据**离散化后的值作为索引**在直方图中累积统计量，当遍历一次数据后，直方图累积了原始样本的连续值在离散区间出现次数。

每一个直方图得到后，计算信息增益就直接对直方图的区间进行计算。这样不用遍历所有的样本，只需要遍历所有离散的区间值。而离散的区间值数目 远远小于 样本数量。这样就能大大减小时间复杂度。

直接将时间复杂度从 $O(\#data * \# feature)$  降低到  $O(k * \#feature)$ , 而 $\#data \gg k$ 。

然后根据直方图的离散值，遍历寻找最优的分割点。





## 2.1.2 优点

### 1. 内存占用更小

直方图算法不仅不需要额外存储预排序的结果，而且可以只保存特征离散化后的值

### 2. 计算代价更小

预排序算法XGBoost每遍历一个特征值就需要计算一次分裂的增益。

而直方图算法LightGBM只需要计算 k 次，直接将时间复杂度从  $O(\#data * \# feature)$  降低到  $O(k * \#feature)$ ，而  $\#data \gg k$ 。

## 2.3 单边梯度采样算法(减少样本数量)

Gradient-based One-Side Sampling 应该被翻译为单边梯度采样 (GOSS)。GOSS 算法从减少样本的角度出发，排除大部分小梯度的样本，仅用剩下的样本计算信息增益，它是一种在减少数据量和保证精度上平衡的算法。

### 2.3.1 动机

AdaBoost 中，样本权重是数据重要性的指标。然而在 GBDT 中没有原始样本权重，不能应用权重采样。幸运的是，我们观察到 GBDT 中每个数据都有不同的梯度值，对采样十分有用。

梯度小的样本，训练误差也比较小，说明数据已经被模型学习得很好了，直接想法就是丢掉这部分梯度小的数据。然而这样做会改变数据的分布，将会影响训练模型的精确度，为了避免此问题，提出了 GOSS 算法。

### 2.3.2 基本原理

GOSS 是一个样本的采样算法，目的是丢弃一些对计算信息增益没有帮助的样本留下有帮助的。根据计算信息增益的定义，梯度大的样本对信息增益有更大的影响。因此，GOSS 在进行数据采样的

时候只保留了梯度较大的数据，但是如果直接将所有梯度较小的数据都丢弃掉势必会影响数据的总体分布。

所以，GOSS首先将要进行分裂的特征的所有取值按照绝对值大小降序排序（XGBoost一样也进行了排序，但是LightGBM不用保存排序后的结果）。

选取绝对值最大的  $a$  个数据。然后在剩下的较小梯度数据中随机选择  $b$  个数据。接着讲这  $b$  个数据乘常数  $(1-a)/b$ ，这样算法就会更关注训练不足的样本，而不会过多的改变原始数据集的分布。最后用这个  $(a + b)$  个数据来计算信息增益。

一方面算法将更多的注意力放在训练不足的样本上，另一方面通过乘上权重来防止采样对原始数据分布造成太大的影响。

## 2.4 互斥特征捆绑算法（减小特征维度）

### 2.4.1 动机

高维度的数据往往是稀疏的，这种稀疏性启发我们设计一种无损的方法来减少特征的维度。通常被捆绑的特征都是互斥的（即特征不会同时为非零值，像one-hot），这样两个特征捆绑起来才不会丢失信息。

如果两个特征并不是完全互斥（部分情况下两个特征都是非零值），可以用一个指标对特征不互斥程度进行衡量，称之为冲突比率，当这个值较小时，我们可以选择把不完全互斥的两个特征捆绑，而不影响最后的精度。

### 2.4.2 原理

互斥特征捆绑算法（Exclusive Feature Bundling, EFB）指出如果将一些特征进行融合绑定，则可以降低特征数量。这样在构建直方图时的时间复杂度从  $O(\#data * \#feature)$  变为  $O(\#data * \#bundle)$ ，这里  $\#bundle$  指的是融合绑定后的特征个数， $\#bundle \ll \# feature$ 。

针对这种想法，我们会遇到两个问题：

- 怎么判定哪些特征应该绑在一起（build bundled）？
- 怎么把特征绑为一个（merge feature）？

#### 1. 解决哪些特征应该绑在一起

将相互独立的特征进行绑定是一个 NP-Hard 问题，LightGBM 的 EFB 算法将这个问题转化为图着色的问题来求解，将所有的特征视为图的各个顶点，将不是相互独立的特征用一条边连接起来，边的权重就是两个相连接的特征的总冲突值，这样需要绑定的特征就是在图着色问题中要涂上同一种颜色的那些点（特征）。

我们注意到通常有很多特征，尽管不是100%相互排斥，但也很少同时取非零值。如果我们的算法可以允许一小部分的冲突，我们可以得到更少的特征包，进一步提高计算效率。

具体步骤可以总结如下：

1. 构造一个加权无向图，顶点是特征，边有权重，其权重与两个特征间冲突相关；
2. 根据节点的度进行降序排序，度越大，与其它特征的冲突越大；
3. 遍历每个特征，将它分配给现有特征包，或者新建一个特征包，使得总体冲突最小。

## 2. 解决怎么把特征绑为一捆

特征合并算法，其关键在于原始特征能从合并的特征中分离出来。绑定几个特征在同一个bundle里需要保证绑定前的原始特征的值可以在bundle中识别，考虑到histogram-based算法将连续的值保存为离散的bins，我们可以使得不同特征的值分到bundle中的不同bin（箱子）中，这可以通过在特征值中加一个偏置常量来解决。

比如，我们在bundle中绑定了两个特征A和B，A特征的原始取值为区间[0,10)，B特征的原始取值为区间[0,20），我们可以在B特征的取值上加一个偏置常量10，将其取值范围变为[10,30），绑定后的特征取值范围为 [0, 30），这样就可以放心的融合特征A和B了。

# 6. LGB (二) 特点

## 3. LightGBM的工程优化

### 3.1 直接支持类别特征

实际上大多数机器学习工具都无法直接支持类别特征，一般需要把类别特征，通过 one-hot 编码，转化到多维的0/1特征，降低了空间和时间的效率。

但我们知道对于决策树来说并不推荐使用 one-hot 编码，尤其当类别特征中类别个数很多的情况下，会产生样本切分不平衡问题，导致切分增益非常小（即浪费了这个特征）

LightGBM优化了对类别特征的支持，可以直接输入类别特征，不需要额外的0/1展开。

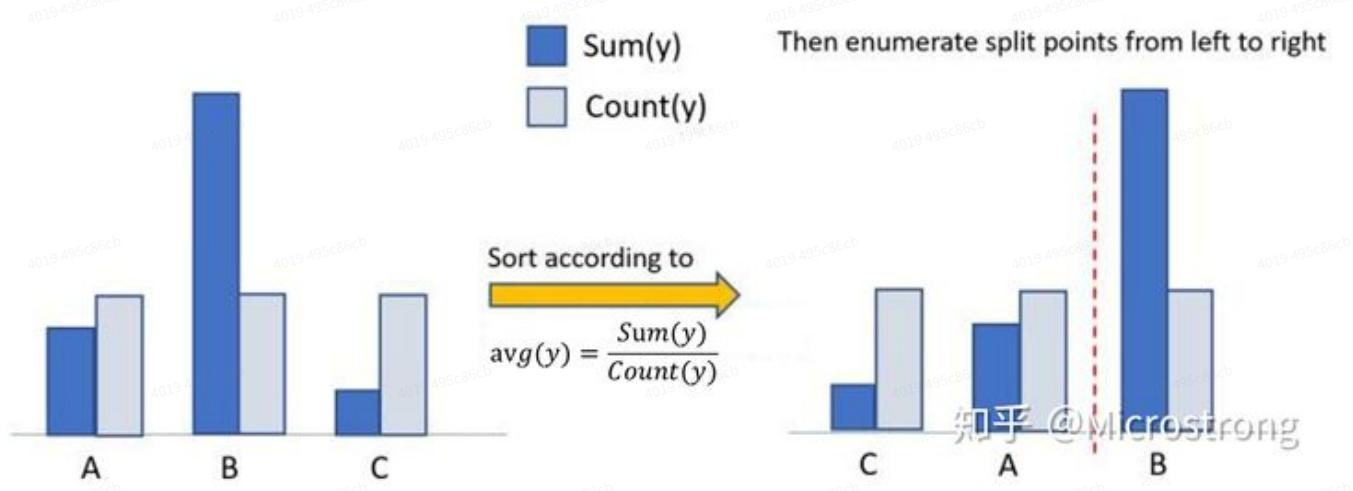
LightGBM采用 many-vs-many 的切分方式将类别特征分为两个子集，实现类别特征的最优切分。实现了 $O(k \log k)$ 的时间复杂度， $k$ 是某维度类别数量。

#### 3.1.1 算法实现 (many-vs-many 的切分方式)

编码方式类似于：目标编码（即把类别对应的标签的平均值当成对应的特征值）。

在枚举分割点之前，先把直方图按照每个类别对应的label均值进行排序；然后按照排序的结果依次枚举最优分割点。流程如下图：

然后根据这个替换的值进行最优分裂，将样本分成多个子集。



当然，这个方法很容易过拟合，所以LightGBM里面还增加了很多对于这个方法的约束和正则化。在Expo数据集上的实验结果表明，相比0/1展开的方法，使用LightGBM支持的类别特征可以使训练速度加速8倍，并且精度一致。更重要的是，LightGBM是第一个直接支持类别特征的GBDT工具。

## 3.2 支持高效并行

### 3.2.1 特征并行

特征并行的主要思想是不同机器在不同的特征集合上分别寻找最优的分割点，然后在机器间同步最优的分割点。XGBoost使用的就是这种特征并行方法。

这种特征并行方法有个很大的缺点：就是对数据进行垂直划分，每台机器所含数据不同，然后使用不同机器找到不同特征的最优分裂点，划分结果需要通过通信告知每台机器，增加了额外的复杂度。

LightGBM 则不进行数据垂直划分，而是在每台机器上保存全部训练数据，在得到最佳划分方案后可在本地执行划分而减少了不必要的通信。

### 3.2.2 数据并行

传统的数据并行策略主要为水平划分数据，然后本地构建直方图并整合成全局直方图，最后在全局直方图中找出最佳划分点。

这种数据划分有一个很大的缺点：通讯开销过大。如果使用点对点通信，一台机器的通讯开销大约为  $O(\#machine * \#feature * \#bin)$ ；如果使用集成的通信，则通讯开销为  $O(2 * \#feature * \#bin)$ ，

LightGBM 采用分散规约（Reduce scatter）的方式将直方图整合的任务分摊到不同机器上，从而降低通信代价，并通过直方图做差进一步降低不同机器间的通信。

### 3.2.3 投票并行

针对数据量特别大特征也特别多的情况下，可以采用投票并行。投票并行主要针对数据并行时数据合并的通信代价比较大的瓶颈进行优化，其通过投票的方式只合并部分特征的直方图从而达到降低通信量的目的。

大致步骤为两步：

1. 本地找出 Top K 特征，并基于投票筛选出可能是最优分割点的特征；
2. 合并时只合并每个机器选出来的特征

## 4. LightGBM的优缺点

### 4.1 优点

这部分主要总结下 LightGBM 相对于 XGBoost 的优点，从内存和速度两方面进行介绍。

#### (1) 速度更快

- LightGBM 采用了**直方图算法**将遍历样本转变为遍历直方图，极大的降低了时间复杂度；
- LightGBM 在训练过程中采用**单边梯度算法**过滤掉梯度小的样本，减少了大量的计算；
- LightGBM 采用了基于 **Leaf-wise 算法**的增长策略构建树，减少了很多不必要的计算量；
- LightGBM 采用优化后的**特征并行、数据并行方法**加速计算，当数据量非常大的时候还可以采用投票并行的策略；
- LightGBM 对**缓存**也进行了优化，增加了缓存命中率；

## (2) 内存更小

- XGBoost 使用预排序后需要记录特征值及其对应样本的统计值的索引，而 LightGBM 使用了直方图算法将特征值转变为 bin 值，且不需要记录特征到样本的索引，将空间复杂度从  $O(2 * \# \text{data})$  降低为  $O(\# \text{bin})$ ，极大的减少了内存消耗；
- LightGBM 采用了**直方图算法**将存储特征值转变为存储 bin 值，降低了内存消耗；
- LightGBM 在训练过程中采用**互斥特征捆绑算法**减少了特征数量，降低了内存消耗。

## 4.2 缺点

- 可能会长出比较深的决策树，产生过拟合。因此LightGBM在Leaf-wise之上增加了一个最大深度限制，在保证高效率的同时防止过拟合；
- Boosting族是迭代算法，每一次迭代都根据上一次迭代的预测结果对样本进行权重调整，所以随着迭代不断进行，误差会越来越小，模型的偏差（bias）会不断降低。由于LightGBM是基于偏差的算法，所以会对噪点较为敏感；
- 在寻找最优解时，依据的是最优切分变量，没有将**最优解是全部特征的综合这一理念**考虑进去；

# 1. LR, SVM

## 1. LR

### 4. 逻辑回归 (Logistic Regression, LR)

本质上是线性回归，只是在结果前加了一层逻辑函数。  
即：先把特征线性求和，然后使用函数  $g(z)$  作为假设函数来预测。

$$g(z) \text{ 为 sigmoid} \Rightarrow g(z) = \frac{1}{1+e^{-z}}$$

$$\Rightarrow g(\theta^T x) = \frac{1}{1+e^{-\theta^T x}}$$

= 分类的损失函数 (交叉熵) | 对数似然损失函数

$$L = \frac{1}{m} \sum_{i=1}^m L(g^{(i)}, y^{(i)}) = \frac{1}{m} \sum_{i=1}^m (-y^{(i)} \log(g^{(i)}) - (1-y^{(i)}) \log(1-g^{(i)}))$$

逻辑回归实现的分类：

使用 softmax 构造模型解决多分类。输出的值对应于每个类别的概率。

## 2. SVM 二分类

### 2.1 原理

#### 5. SVM (Support Vector Machine)

5.1 用于解决二分类或多分类问题。

SVM 的目标是寻找一个最优化超平面在空间中分割两类数据，  
该最优化超平面需要满足的条件是：离其最近的点到其距离最大化。  
而这些点称为支撑向量。

### 2.2 公式推导

5.2 划分平面可通过如下线性推来描述：

$$w^T x + b = 0$$

样本空间中任一点 $x$ 到超平面 $(w, b)$ 的距离 $\gamma$ 为：

$$\gamma = \frac{|w^T x + b|}{\|w\|}$$

假设超平面 $(w, b)$ 能将训练样本正确分类，有

$$\begin{cases} w^T x_i + b \geq +1, & y_i = +1 \\ w^T x_i + b \leq -1, & y_i = -1 \end{cases} \quad (1)$$

两个异类支持向量到超平面的距离之和为：(间隔 margin)

$$\gamma = \frac{2}{\|w\|} \quad (2)$$

欲找到具有“最大间隔”(maximum margin)的划分超平面，即找到满足(1)中的 $w, b$ ，使 $\gamma$ 最大，即

$$\max_{w, b} \frac{\gamma}{\|w\|}$$

$$\text{s.t. } y_i(w^T x_i + b) \geq 1,$$

最大化 $\frac{\gamma}{\|w\|}$ ，等价于最小化 $\|w\|^2$ ，优化目标为

$$\begin{cases} \min_{w, b} \frac{1}{2} \|w\|^2 \\ \text{s.t. } y_i(w^T x_i + b) \geq 1 \end{cases}$$

接下来为拉格朗日和法转化为“对偶问题”求解。

## 2.3 核函数

## 5.4 核函数

将不可分的数据映射到高维可分 (高斯核, 多项式核...)

什么时候线性核, 什么时候高斯核?

当数据特征提取地较好, 问题线性可分则可用线性核;

若特征较少, 样本集中, 问题线性不可分  $\Rightarrow$  高斯核

$$\text{核方法} \quad \min_{w,b} \frac{1}{2} \|w\|^2$$

$$\text{s.t. } y_i (w^T \phi(x_i) + b) \geq 1$$

## 2.4 SVM的硬间隔, 软间隔

$$\min_{w,b} \frac{1}{2} \|w\|^2$$

$$\text{s.t. } y^{(i)} (w^T x^{(i)} + b) \geq 1$$

硬间隔

$$\min_{w,b} \frac{1}{2} \|w\|^2 + C \sum_{i=1}^m \xi_i$$

$$\text{s.t. } y^{(i)} (w^T x^{(i)} + b) \geq 1 - \xi_i$$
  
$$\xi_i \geq 0$$

软间隔

软间隔是在硬间隔的基础上引入了松弛变量。[为了防止过拟合]

## 3. SVM回归

线性回归

分类的 hinge loss

$$\min_{w,b} \frac{1}{2} \|w\|^2 \quad : \text{让支持向量距离更大.}$$

$$\text{s.t. } y_i(w_i \cdot \phi(x_i) + b) \geq 1 \quad : \text{让样本都正确分类}$$

而对回归模型：支持向量之间距离大不重要，但均值要变。

回归模型的目标是让训练集中的每个点  $(x_i, y_i)$  尽量拟合到一个线性模型  $y_i = w \cdot \phi(x_i) + b$ 。

然后，定义一个量  $\epsilon > 0$ ，使  $|y_i - w_i \cdot \phi(x_i) - b| \leq \epsilon$  最小。

对于某个点  $(x_i, y_i)$ ，若  $|y_i - w \cdot \phi(x_i) - b| \leq \epsilon$ ，则完全无损失。

若  $|y_i - w \cdot \phi(x_i) - b| > \epsilon$ ，则对应的损失为  $|y_i - w \cdot \phi(x_i) - b| - \epsilon$

## 4. SVM优缺点

优点：

- 1) 面对高维数据，可以使用核函数进行有效处理

- 2) 使用核函数可以解决非线性问题。

1) SVM算法对大规模训练样本难以实施

2) 对缺失数据敏感，对参数和核函数的选择敏感

3) 解决多分类问题困难。

经典的SVM只提出了二分类算法，而在实际应用中，极要解决多分类问题，可以通过多个支持向量机组合来解决。

## 5. SVM与LR

1) LR是参数模型，SVM是非参数模型

2) LR采用的损失函数是 logistical loss，SVM是 hinge loss.

3) 在学习时，SVM考虑与分类最相关的少数支持向量点。

LR的模型相对简单，在进行大规模线性分类时比较方便。

## 6. 调参

① SVM

① 分类: `from sklearn.svm import SVC`

`clf = SVC(kernel='linear').fit(X,y)`

调1: 核函数有 "linear", "rbf", "poly"(多项式核)

调2: C 惩罚参数: 默认是 1。 (↑ C↑, 过拟合, C↓, 不拟合)

C 越大, 相当于惩罚松弛变量, 希望松弛变量接近 0, 即对误分类的惩罚增大, 越有利于训练集全对, 但泛化能力弱(过拟合)

(越小, 对误分类的惩罚减小, 允许容错, 将他们当成噪点, 泛化能力强。)

② 回归: `from sklearn.svm import SVR`

# 1. 贝叶斯理论

## 1.1 贝叶斯公式

“如果一个袋子中共有 10 个球，分别是黑球和白球，但是我们不知道它们之间的比例是怎样的，现在，**仅通过摸出的球的颜色，是否能判断出袋子里面黑白球的比例？**”

上述问题可能与我们高中时期所接受的概率有所冲突，因为你所接触的概率问题可能是这样的：“一个袋子里里面有 10 个球，其中 4 个黑球，6 个白球，如果你随机抓取一个球，那么是黑球的概率是多少？”毫无疑问，答案是 0.4。这个问题非常简单，因为我们事先知道了袋子里面黑球和白球的比例，所以很容易算出摸一个球的概率，**但是在某些复杂情况下，我们无法得知“比例”，此时就引出了贝叶斯提出的问题。**

在统计学中有两个较大的分支：一个是“频率”，另一个便是“贝叶斯”，它们都有各自庞大的知识体系，而“贝叶斯”主要利用了**“相关性”**一词。

下面以通俗易懂的方式描述一下“贝叶斯定理”：通常，

**事件 A 在事件 B 发生的条件下发生的概率：** $P(A|B)$

**事件 B 在事件 A 发生的条件下发生的概率：** $P(B|A)$

它们两者的概率并不相同， $P(A|B)$  **不等于**  $P(B|A)$

**但是它们两者之间存在一定的相关性，并具有以下公式**（称之为“贝叶斯公式”）：

$$P(A|B) = \frac{P(B|A)P(A)}{P(B)}$$

首先我们要了解上述公式中符号的意义：

- $P(A)$  这是概率中最基本的符号，表示 A 出现的概率
- $P(B|A)$  是**条件概率**的符号，表示事件 A 发生的条件下，事件 B 发生的概率，条件概率是“贝叶斯公式”的关键所在，它也被称为**“似然度”**。
- $P(A|B)$  是**条件概率**的符号，表示事件 B 发生的条件下，事件 A 发生的概率，这个计算结果也被称为**“后验概率”**。

有上述描述可知，贝叶斯公式可以预测事件发生的概率，**两个本来相互独立的事件，发生了某种“相关性”**，此时就可以通过“贝叶斯公式”实现预测。

## 1.2 先验概率

在贝叶斯看来，世界并非静止不动的，而是动态和相对的，他希望利用已知经验来进行判断，那么如何用经验进行判断呢？这里就必须要提到“先验”和“后验”这两个词语。

我们先讲解“先验”，其实“先验”就相当于“未卜先知”，在事情即将发生之前，做一个概率预判。比如从远处驶来了一辆车，是轿车的概率是45%，是货车的概率是35%，是大客车的概率是20%，在你没有看清之前基本靠猜，此时，我们把这个概率就叫做“先验概率”。

## 1.3 后验概率

在理解了“先验概率”的基础上，我们来研究一下什么是“后验概率”？

我们知道每一个事物都有自己的特征，比如前面所说的轿车、货车、客车，它们都有着各自不同的特征，距离过远的时候，我们无法用肉眼分辨，而当距离达到一定范围内就可以根据**各自的特征再次做出概率预判**，这就是后验概率。

比如轿车的速度相比于另外两者更快可以记做  $P(\text{轿车}|\text{速度快}) = 55\%$ ，而客车体型可能更大，可以记做  $P(\text{客车}|\text{体型大}) = 35\%$ 。

如果用**条件概率**来表述  $P(\text{体型大}|\text{客车})=35\%$ ，这种通过“车辆类别”推算出“类别特征”发生的概率的方法叫作“似然度”。这里的似然就是“可能性”的意思。

## 1.5 朴素+贝叶斯

了解完上述概念，你可能对贝叶斯定理有了一个基本的认识，**实际上贝叶斯定理就是求解后验概率的过程，而核心方法是通过似然度预测后验概率，通过不断提高似然度，自然也就达到了提高后验概率的目的。**

朴素贝叶斯是一种简单的贝叶斯算法，因为贝叶斯定理涉及到了概率学、统计学，其应用相对复杂，因此我们只能以简单的方式使用它，比如天真的认为，所有事物之间的特征都是相互独立的，彼此互不影响。

## 2.朴素贝叶斯算法

### 2.1 多特征分类问题

下面我们使统计学的相关知识解决上述分类问题，分类问题的样本数据大致如下所示：

C++

1 [特征 X<sub>1</sub> 的值, 特征 X<sub>2</sub> 的值, 特征 X<sub>3</sub> 的值, ..., 类别 A<sub>1</sub>]

2 [特征 X<sub>1</sub> 的值, 特征 X<sub>2</sub> 的值, 特征 X<sub>3</sub> 的值, ..., 类别 A<sub>2</sub>]

**解决思路：**这里我们先简单的采用 1 和 0 代表特征值的有无，比如当 X<sub>1</sub> 的特征值等于 1 时，则该样本属于 A<sub>1</sub> 的类别概率；特征值 X<sub>2</sub> 值为 1 时，该样本属于类别 A<sub>1</sub> 的类别的概率。

依次类推，然后最终算出该样本对于各个类别的概率值，哪个概率值最大就可能是哪个类。

上述思路就是贝叶斯定理的典型应用，如果使用条件概率表达，如下所示：

Apache

1 P(类别 A<sub>1</sub> | 特征 X<sub>1</sub>, 特征 X<sub>2</sub>, 特征 X<sub>3</sub>, ...)

上述式子表达的意思是：**在特征 X<sub>1</sub>、X<sub>2</sub>、X<sub>3</sub> 等共同发生的条件下，类别 A<sub>1</sub> 发生的概率**，也就是**后验概率**。

## 2.2 朴素贝叶斯算法

上一节我们已经了解了贝叶斯公式，下面使用贝叶斯公式将多特征分类问题表达出来，如下所示：

$$P(c | \mathbf{x}) = \frac{P(c)P(\mathbf{x} | c)}{P(\mathbf{x})}$$

来估计后验概率  $P(\mathbf{x} | c)$  的主要困难在于：类条件概率  $P(\mathbf{x} | c)$  是所有属性上的联合概率，难以从有限的训练样本直接估计而得。

为避开这个障碍，**朴素贝叶斯分类器(naive Bayes classifier)**采用了**"属性条件独立性假设"**(attribute conditional independence assumption):对已知类别，假设所有属性相互独立。

$$P(c|\mathbf{x}) = \frac{P(c)P(\mathbf{x}|c)}{P(\mathbf{x})} = \frac{P(c)}{P(\mathbf{x})} \prod_{i=1}^d P(x_i|c)$$

换言之，假设每个属性独立地对分类结果发生影响，其表达式为：**(目标方程)**

$$h_{nb}(\mathbf{x}) = \arg \max_{c \in \mathcal{C}} P(c) \prod_{i=1}^d P(x_i | c)$$

## 2.3 朴素贝叶斯分类器的训练

朴素贝叶斯的训练就是基于数据集 D，来估计“类先验概率  $P(c)$ ”，并为每个属性估计条件概率。

令  $D_c$  表示训练集 D 中第 c 类样本组成的集合，若有充足的独立同分布样本，则很容易估计出“类先验概率”

$$P(c) = \frac{|D_c|}{|D|}$$

- 对离散属性而言

令  $D_{c,x_i}$  表示  $D_c$  中在第 i 个属性上取值为  $x_i$  的样本组成的集

则条件概率  $P(x_i | c)$  可估计为

$$P(x_i | c) = \frac{|D_{c,x_i}|}{|D_c|}$$

- 对连续属性

可考虑概率密度函数

假定

$$p(x_i | c) \sim \mathcal{N}(\mu_{c,i}, \sigma_{c,i}^2)$$

其中  $\mu_{c,i}$  和  $\sigma_{c,i}^2$  分别是第 c 类样本在第 i 个属性上取值的“均值”和“方差”

则有

$$p(x_i | c) = \frac{1}{\sqrt{2\pi}\sigma_{c,i}} \exp\left(-\frac{(x_i - \mu_{c,i})^2}{2\sigma_{c,i}^2}\right)$$

## 2.4 手算朴素贝叶斯实例

训练集：

表 4.3 西瓜数据集 3.0

编号	色泽	根蒂	敲声	纹理	脐部	触感	密度	含糖率	好瓜
1	青绿	蜷缩	浊响	清晰	凹陷	硬滑	0.697	0.460	是
2	乌黑	蜷缩	沉闷	清晰	凹陷	硬滑	0.774	0.376	是
3	乌黑	蜷缩	浊响	清晰	凹陷	硬滑	0.634	0.264	是
4	青绿	蜷缩	沉闷	清晰	凹陷	硬滑	0.608	0.318	是
5	浅白	蜷缩	浊响	清晰	凹陷	硬滑	0.556	0.215	是
6	青绿	稍蜷	浊响	清晰	稍凹	软粘	0.403	0.237	是
7	乌黑	稍蜷	浊响	稍糊	稍凹	软粘	0.481	0.149	是
8	乌黑	稍蜷	浊响	清晰	稍凹	硬滑	0.437	0.211	是
9	乌黑	稍蜷	沉闷	稍糊	稍凹	硬滑	0.666	0.091	否
10	青绿	硬挺	清脆	清晰	平坦	软粘	0.243	0.267	否
11	浅白	硬挺	清脆	模糊	平坦	硬滑	0.245	0.057	否
12	浅白	蜷缩	浊响	模糊	平坦	软粘	0.343	0.099	否
13	青绿	稍蜷	浊响	稍糊	凹陷	硬滑	0.639	0.161	否
14	浅白	稍蜷	沉闷	稍糊	凹陷	硬滑	0.657	0.198	否
15	乌黑	稍蜷	浊响	清晰	稍凹	软粘	0.360	0.370	否
16	浅白	蜷缩	浊响	模糊	平坦	硬滑	0.593	0.042	否
17	青绿	蜷缩	沉闷	稍糊	稍凹	硬滑	0.719	0.103@程原	否

测试样例为：

编号	色泽	根蒂	敲声	纹理	脐部	触感	密度	含糖率	好瓜
测 1	青绿	蜷缩	浊响	清晰	凹陷	硬滑	0.697	0.460	?

## 1、计算“类先验概率”

共有17个样本，其中8个好瓜，9个坏瓜

$$P(\text{好瓜} = \text{是}) = \frac{8}{17} \approx 0.471$$

$$P(\text{好瓜} = \text{否}) = \frac{9}{17} \approx 0.529$$

## 2、为每个属性估计条件概率

### (1) 色泽 = 青绿

好瓜里“色泽=青绿”的有3个，好瓜共8个

坏瓜里“色泽=青绿”的有3个，坏瓜共9个

$$P_{\text{青绿}|\text{是}} = P(\text{色泽} = \text{青绿} | \text{好瓜} = \text{是}) = \frac{3}{8} = 0.375$$

$$P_{\text{青绿}|\text{否}} = P(\text{色泽} = \text{青绿} | \text{好瓜} = \text{否}) = \frac{3}{9} \approx 0.333$$

### (2) 纹理 = 蜷缩

好瓜里“纹理=蜷缩”的有5个，好瓜共8个

坏瓜里“纹理=蜷缩”的有3个，坏瓜共9个

$$P_{\text{蜷缩}|\text{是}} = P(\text{根蒂} = \text{蜷缩} | \text{好瓜} = \text{是}) = \frac{5}{8} = 0.375$$

$$P_{\text{蜷缩}|\text{否}} = P(\text{根蒂} = \text{蜷缩} | \text{好瓜} = \text{否}) = \frac{3}{9} \approx 0.333$$

### (3) 敲声 = 浊响

### (4) 纹理 = 清晰

### (5) 脐部 = 凹陷

### (6) 触感 = 硬滑

$$P_{\text{浊响}|\text{是}} = P(\text{敲声} = \text{浊响} | \text{好瓜} = \text{是}) = \frac{6}{8} = 0.750$$

$$P_{\text{浊响}|\text{否}} = P(\text{敲声} = \text{浊响} | \text{好瓜} = \text{否}) = \frac{4}{9} \approx 0.444$$

$$P_{\text{清晰|是}} = P(\text{纹理} = \text{清晰} | \text{好瓜} = \text{是}) = \frac{7}{8} = 0.875$$

$$P_{\text{清晰|否}} = P(\text{纹理} = \text{清晰} | \text{好瓜} = \text{否}) = \frac{2}{9} \approx 0.222$$

$$P_{\text{凹陷|是}} = P(\text{脐部} = \text{凹陷} | \text{好瓜} = \text{是}) = \frac{6}{8} = 0.750$$

$$P_{\text{凹陷|否}} = P(\text{脐部} = \text{凹陷} | \text{好瓜} = \text{否}) = \frac{2}{9} \approx 0.222$$

$$P_{\text{硬滑|是}} = P(\text{触感} = \text{硬滑} | \text{好瓜} = \text{是}) = \frac{6}{8} = 0.750$$

$$P_{\text{硬滑|否}} = P(\text{触感} = \text{硬滑} | \text{好瓜} = \text{否}) = \frac{6}{9} \approx 0.667$$

(7) 密度 = 0.697

$\mu$  和  $\sigma^2$  分别是“正样本”在密度属性上取值的“均值”和“方差”

$$\mu = 0.574, \sigma^2 = 0.129^2$$

$$p_{\text{密度: 0.697|是}} = p(\text{密度} = 0.697 | \text{好瓜} = \text{是})$$

$$= \frac{1}{\sqrt{2\pi} \cdot 0.129} \exp\left(-\frac{(0.697 - 0.574)^2}{2 \cdot 0.129^2}\right) \approx 1.959$$

$\mu$  和  $\sigma^2$  分别是“负样本”在密度属性上取值的“均值”和“方差”

$$p_{\text{密度: 0.697|否}} = p(\text{密度} = 0.697 | \text{好瓜} = \text{否})$$

$$= \frac{1}{\sqrt{2\pi} \cdot 0.195} \exp\left(-\frac{(0.697 - 0.496)^2}{2 \cdot 0.195^2}\right) \approx 1.203$$

(8) 含糖率 = 0.460

$$p_{\text{含糖: 0.460|是}} = p(\text{含糖率} = 0.460 | \text{好瓜} = \text{是})$$

$$= \frac{1}{\sqrt{2\pi} \cdot 0.101} \exp\left(-\frac{(0.460 - 0.279)^2}{2 \cdot 0.101^2}\right) \approx 0.788$$

$p_{\text{含糖: } 0.460|\text{否}} = p(\text{含糖率} = 0.460 | \text{好瓜} = \text{否})$

$$= \frac{1}{\sqrt{2\pi} \cdot 0.108} \exp\left(-\frac{(0.460 - 0.154)^2}{2 \cdot 0.108^2}\right) \approx 0.066$$

### 3、连乘，得出最终结果

公式：

$$h_{nb}(\mathbf{x}) = \arg \max_{c \in \mathcal{Y}} P(c) \prod_{i=1}^d P(x_i | c)$$

好瓜：

$$\begin{aligned} &P(\text{好瓜} = \text{是}) \times P_{\text{青绿|是}} \times P_{\text{蜷缩|是}} \times P_{\text{浊响|是}} \times P_{\text{清晰|是}} \times P_{\text{凹陷|是}} \\ &\times P_{\text{硬滑|是}} \times p_{\text{密度: } 0.697|\text{是}} \times p_{\text{含糖: } 0.460|\text{是}} \approx 0.038, \end{aligned}$$

坏瓜：

$$\begin{aligned} &P(\text{好瓜} = \text{否}) \times P_{\text{青绿|否}} \times P_{\text{蜷缩|否}} \times P_{\text{浊响|否}} \times P_{\text{清晰|否}} \times P_{\text{凹陷|否}} \\ &\times P_{\text{硬滑|否}} \times p_{\text{密度: } 0.697|\text{否}} \times p_{\text{含糖: } 0.460|\text{否}} \approx 6.80 \times 10^{-5}. \end{aligned}$$

结果：

$$0.038 > 6.80 \times 10^{-5}$$

可以看出“好瓜”概率更高，因此最终判别结果为“好瓜”

## 2.5 拉普拉斯修正

### 2.5.1 引入

上面的过程存在一个隐患！**若某个属性值在训练集的某个类“没有出现过”，则**

$$P(x_i | c) = 0$$

例如好瓜里没有“敲声=清脆”的例子

$$P_{\text{清脆}|\text{是}} = P(\text{敲声} = \text{清脆} | \text{好瓜} = \text{是}) = \frac{0}{8} = 0$$

而在连乘时，如果存在这一项，则整个结果永远为0。即“无论其他属性怎么样，永远不选择‘瓜=好’这个结果（因为结果=0）”，这显然是不合理的

### 2.5.2 拉普拉斯修正

为了避免这种情况，在估计概率值时通常需要进行“平滑”

拉普拉斯修正：

$$\hat{P}(c) = \frac{|D_c| + 1}{|D| + N},$$

$$\hat{P}(x_i | c) = \frac{|D_{c,x_i}| + 1}{|D_c| + N_i}.$$

$N$  表示训练集  $D$  中可能的类别数

$N_i$  表示第  $i$  个属性可能的取值数

理解：

其实就是加入了

$$\frac{1}{\text{类别数}}$$

例如：

1、共有17个样本，其中8个好瓜，9个坏瓜，类别数2

$$\hat{P}(\text{好瓜} = \text{是}) = \frac{8+1}{17+2} \approx 0.474, \quad \hat{P}(\text{好瓜} = \text{否}) = \frac{9+1}{17+2} \approx 0.526$$

2、色泽=青绿

好瓜里“色泽=青绿”的有3个，好瓜共8个

坏瓜里“色泽=青绿”的有3个，坏瓜共9个

色泽的可能取值数为3（青绿、乌黑、浅白）

$$\hat{P}_{\text{青绿}|\text{是}} = \hat{P}(\text{色泽} = \text{青绿} | \text{好瓜} = \text{是}) = \frac{3+1}{8+3} \approx 0.364$$

$$\hat{P}_{\text{青绿}|\text{否}} = \hat{P}(\text{色泽} = \text{青绿} | \text{好瓜} = \text{否}) = \frac{3+1}{9+3} \approx 0.333$$

## 效果

拉普拉斯修正避免了因“训练集样本不充分”而导致“概率估计值为零”的问题

并且当训练集变大时，修正过程引入的先验影响也会逐渐变得可以忽略

使得估计值趋向于实际概率值

## 3. sklearn应用朴素贝叶斯算法

在 sklearn 库中，基于贝叶斯定理的算法集中在 `sklearn.naive_bayes` 包中，根据对“似然度  $P(x_i|y)$ ”计算方法的不同，我们将朴素贝叶斯大致分为三种：多项式朴素贝叶斯（`MultinomialNB`）、伯努利分布朴素贝叶斯（`BernoulliNB`）、高斯分布朴素贝叶斯（`GaussianNB`）。

另外一点要牢记，**朴素贝叶斯算法的实现是基于假设而来，在朴素贝叶斯看来，特征之间是相互独立的，互不影响的。**

C++

1 高斯朴素贝叶斯适用于特征呈正态分布的，多项式贝叶斯适用于特征是多项式分布的，伯努利贝叶斯适用于二项分布。

### 3.1 算法使用流程

使用朴素贝叶斯算法，具体分为三步：

- 统计样本数，即统计先验概率  $P(y)$  和似然度  $P(x|y)$ 。
  - 根据待测样本所包含的特征，对不同类分别进行后验概率计算。
  - 比较  $y_1, y_2, \dots, y_n$  的后验概率，哪个的概率值最大就将其作为预测输出。

## 3.2 朴素贝叶斯算法应用

下面通过鸢尾花数据集对朴素贝叶斯分类算法进行简单讲解。如下所示：

# Python

# 3. 判别式模型和生成式模型

对于有监督学习可以将其分为两类模型：判别式模型和生成式模型。简单地说，判别式模型是针对条件分布建模，而生成式模型则针对联合分布进行建模。

## 1. 基本概念

假设我们有训练数据 $(X, Y)$ ， $X$ 是属性集合， $Y$ 是类别标记。这时来了一个新的样本 $x$ ，我们想要预测它的类别 $y$ 。

我们最终的目的是求得最大的条件概率 $P(y|x)$  【在特征是 $x$  的条件下标签是 $y$ 的概率】作为新样本的分类。

### 1.1 判别式模型这么做：

根据训练数据得到分类函数和分界面，比如说根据SVM模型得到一个分界面，然后直接计算条件概率 $P(y|x)$ ，我们将最大的 $P(y|x)$ 作为新样本的分类。

判别式模型是对条件概率建模，学习不同类别之间的最优边界，无法反映训练数据本身的特性，能力有限，其只能告诉我们分类的类别。

### 1.2 生成式模型这么做

一般会对每一个类建立一个模型，有多少个类别，就建立多少个模型。比如说类别标签有 {猫，狗，猪}，那首先根据猫的特征学习出一个猫的模型，再根据狗的特征学习出狗的模型，之后分别计算新样本 $x$  跟三个类别的联合概率 $P(x,y)$ ，然后根据贝叶斯公式：

$$P(y|x) = \frac{P(x,y)}{P(x)}$$

分别计算 $P(y|x)$ ，选择三类中最大的 $P(y|x)$ 作为样本的分类。

### 1.3 两个模型的小结

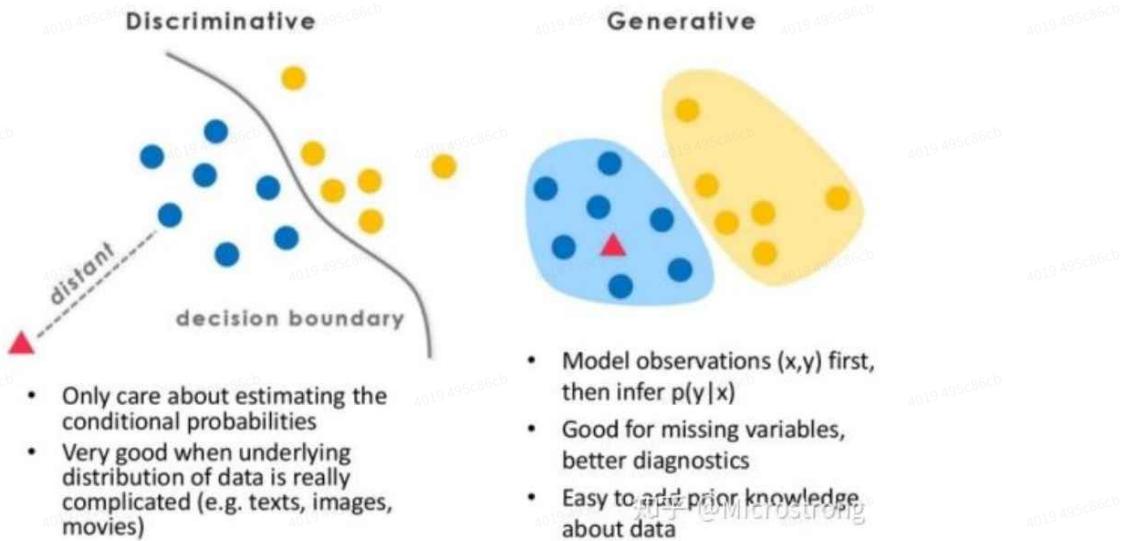
不管是生成式模型还是判别式模型，它们最终的判断依据都是条件概率 $P(y|x)$ 。

但是生成式模型先计算了联合概率 $P(x,y)$ ，再由贝叶斯公式计算得到条件概率。因此，生成式模型可以体现更多数据本身的分布信息，其普适性更广。

## 2. 两者区别

### 2.1 判别式模型和生成式模型的对比图

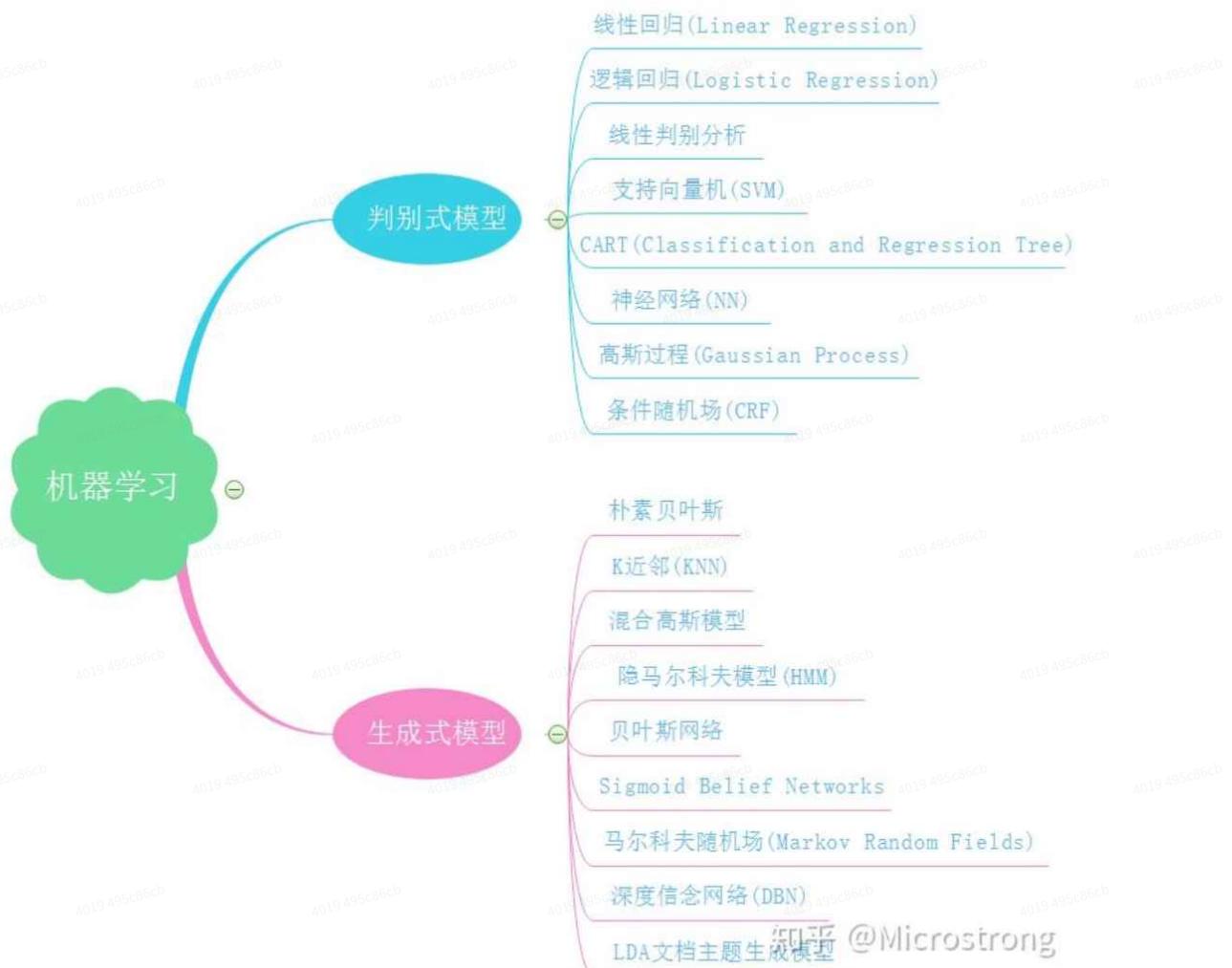
# Discriminative vs. Generative



上图左边为判别式模型而右边为生成式模型，可以很清晰地看到差别，**判别式模型是在寻找一个决策边界，通过该边界来将样本划分到对应类别。**

而生成式模型则不同，**它学习了每个类别的边界，它包含了更多信息，可以用来生成样本。**

## 2.2两者所包含的算法



# 2. K-means

## 1. 原理

### 1.1 算法过程

1. 初始化k个质心，作为初始的k个簇的中心点，k为人工设定的超参数；
2. 然后对于每一个样本分别计算其k个质心的距离，并将样本点归于最近的一类中。
3. 重新计算质心，即将每一类中的所有点取平均值。
4. 重复上述过程直到达到预定的迭代次数或质心不再发生明显变化

### 1.2 损失函数

$$SSE = \sum_{k=1}^K \sum_{p \in C_k} |p - m_k|^2$$

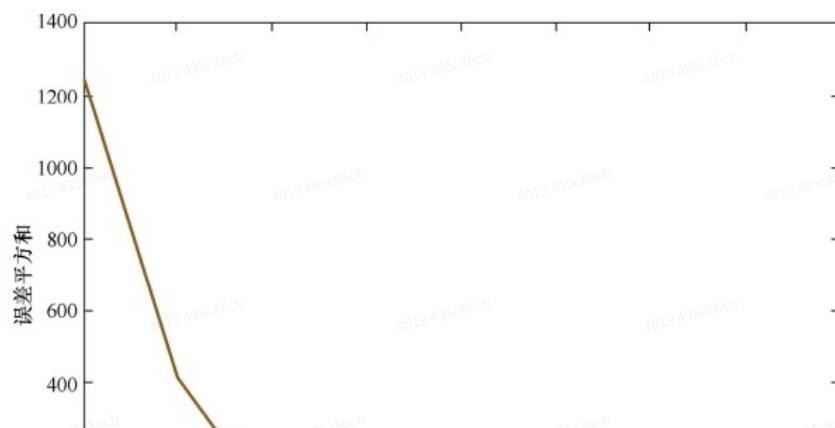
其中，K是聚类数量，p是样本， $m_k$ 是第k个聚类的中心点。SSE越小，说明样本聚合程度越高。

### 1.3 怎么确定聚类数量K（聚类如果不清楚有多少类，有什么方法？）

和评估分类或回归的方式一样，选择某个metric或某些metrics下最好的k，例如sse（其实就是kmeans的损失函数了），轮廓系数。

k的大小调参，手工方法，手肘法为代表。

手肘法其实没什么特别的，纵轴是聚类效果的评估指标，根据具体的问题而定，如果聚类是作为单独的任务存在则使用sse或轮廓系数这类无监督的metric作为纵坐标，然后找到metric最好并且k最小的结果对应的k为最终的选择；（我们说的学习曲线）



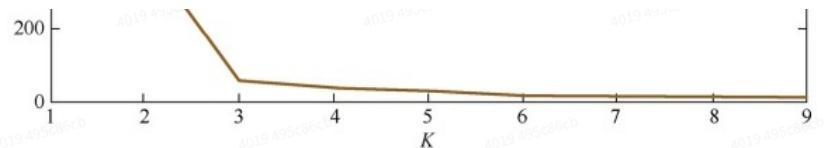


图5.3 K均值算法中K值的选取：手肘法

知乎 @马东什么

## 1.4k-means的缺点，怎么解决？

### 1. 对异常样本很敏感，簇心会因为异常样本被拉得很远

解决方法即做好预处理，将异常样本剔除或修正

### 2. k值需要事先指定，有时候难以确定

解决方法即针对k调参。

### 3. 只能拟合球形簇

对于流形簇等不规则的簇或是存在簇重叠问题的复杂情况等，效果较差。

解决方法，换算法。

### 4. 无法处理离散特征，缺失特征

# 3. DBSCAN

各个目标簇是由一群密集的数据点构成的，这些密集数据点构成的簇被稀疏区域分割，基于密度的聚类的最终目的是从稀疏区域中发现簇，并将稀疏区域中的孤立的样本点当作异常点；

基于密度的聚类，通过引入密度的概念，打破了基于划分的聚类的限制，可以拟合任意形状的簇。

## 1. 原理

DBSCAN的算法流程：

### Step1. (定义超参，半径，邻域下的最少样本)

定义超参数 $\text{epsilon}$ 表示半径（即 $\text{epsilon}$ -邻域），

$\text{minpoints}$ 表示 $\text{epsilon}$ -邻域下的最少样本数量

### Step2. (遍历样本)

从数据集中依次选择一个样本点进入后续的处理，直到所有样本遍历完毕；

### Step3. 确定核心点

对于选定的样本点 $s$ ，做3件事：

- (1) 标记为“已访问”；
- (2) 计算以样本 $s$ 为中心， $\text{epsilon}$ 为半径的内的样本数量 $\text{neighbor\_points}$ 。
- (3)

如果 $\text{neighbor\_points}$ 中的 $\text{points}$ 的数量小于 $\text{minpoints}$ ，则样本 $s$ 标记为噪声点，进入step2继续遍历；（邻域内样本点过少，就是离散点，标记位噪音）

如果 $\text{neighbor\_points}$ 中的 $\text{points}$ 的数量大于等于 $\text{minpoints}$ ，则样本 $s$ 标记为核心点。

为核心点 $s$ 生成一个聚类簇 $\text{cluster}_s$ ，此时 $\text{cluster}_s$ 是一个空簇。 $\text{neighbor\_points}$ 中的每一个 $\text{point}$ 都是样本 $s$ 的边界点，他们和核心点 $s$ 之间的关系是密度直达的；

（只要是在核心点半径内的点都是密度直达的，否则就是不可达）

（但是并非是只要在核心点的密度可达范围内就是该核心点对应簇的成员，原因是它有可能也是其他核心点的密度可达的点，这时候就是先到先得，标记就会发生作用，只能添加未标记的样本到自己的簇内，已经标记的样本就不行）

### Step4. 确定核心点对应的簇内样本

先将核心点s放入cluster\_s中，然后遍历neighbor\_points中的每一个neighbor:

如果neighbor被访问过，则这个neighbor只可能有两种情况：

1) 它是一个噪声点，不属于之前的任何一个cluster，则加入cluster\_s中，不再是噪声点了，而是核心点s的边界点（与核心点s密度直达）；

2) 它是其它cluster内的点，则别人家的东西不能动，跳过；

如果neighbor没被访问过，显然neighbor不是噪声点（都没访问过怎么可能被标注），也不可能别的cluster内的点（都没被访问过怎么可能是别的clsuter的点），则：

1) 先标记为“已访问”，免得被别的cluster抢走了；

2) 判断neighbor是否为核心点，

如果不是，则neighbor直接并入cluster\_s中，

如果是，就与该neighbor的簇内样本进行合并，得到一个更大的簇内样本。

即neighbor\_points=neighbor\_points+neighbor的neighbor\_points（注意这里是递归计算的），然后继续step4进行遍历；

## 2. 怎么解决kmeans缺点的

### 1. K-means对异常样本很敏感

DBSCAN定义了密度的计算方式，不涉及到任何的平均这种鲁棒性较差的计算方式，对异常样本不敏感，还能检测异常样本呢；

### 2. K值需要事先指定；

DBSCAN不需要指定簇的数量；算法迭代过程中自然而然产生最优的k个聚类簇；

### 3. 只能拟合球形簇，

基于密度的聚类可以拟合任意形状的簇，这也归功于密度的计算方式，基于密度的聚类本身不对聚类簇的形状有任何的假设；

### 4. 无法处理离散特征，缺失特征：缺失特征要插补，离散特征可以换离散特征的距离度量方法，

基于密度的聚类算法可以灵活搭配各种不同的distance的度量方式；



# 2.0.PCA(Principal Component Analysis)

## 1.PCA的思想

PCA的主要思想是将n维特征映射到k维上，这k维是全新的**正交特征**也被称为**主成分**，**是在原有n维特征的基础上重新构造出来的k维特征。**

### 1.1PCA做法

通过计算数据矩阵的协方差矩阵，然后得到**协方差矩阵的特征值特征向量**，选择**特征值最大**(即方差最大的)k个特征所对应的特征向量组成的矩阵。

这样就可以将数据矩阵转换到新的空间当中，实现数据特征的降维。

### 1.2做法解释

**PCA的目标：**

- 1) 使得保留下来的维度间的相关性尽可能小。
- 2) 使得保留下来的维度含有尽可能多的原始信息（方差大）

所以，要知道各维度间的相关性以及各维度上的方差。那这个时候就想到了协方差矩阵。

**协方差矩阵的含义：**

主对角线上的元素是各个维度上的方差。

其他元素是两两维度间的协方差（即相关性）。

**两者结合：**

**PCA目标的第一条反应到协方差矩阵中就是，使协方差矩阵中的非对角元素基本为0。**

现在的目标是： $Y = PX$ ，

找到一个P，使Y的协方差矩阵( $YY^T$ )变成一个对角矩阵。

$$\begin{aligned} D &= \frac{1}{m} YY^T \\ &= \frac{1}{m} (PX)(PX)^T \\ &= \frac{1}{m} PXX^TP^T \end{aligned}$$

$$= P\left(\frac{1}{m}XX^T\right)P^T$$

$$= PCP^T$$

我们要找的P不是别的，而是能让原始协方差矩阵 ( $XX^T$ ) 对角化的P。所以后面就是对X的协方差矩阵分解即可得到最优的P，完成降维。

## 2. 协方差矩阵

样本均值：

$$\bar{x} = \frac{1}{n} \sum_{i=1}^N x_i$$

样本方差：

$$S^2 = \frac{1}{n-1} \sum_{i=1}^n (x_i - \bar{x})^2$$

样本X和样本Y的协方差：

$$\begin{aligned} Cov(X, Y) &= E[(X - E(X))(Y - E(Y))] \\ &= \frac{1}{n-1} \sum_{i=1}^n (x_i - \bar{x})(y_i - \bar{y}) \end{aligned}$$

由上面的公式，我们可以得到以下结论：

(1) 方差的计算公式是针对一维特征，即针对同一特征不同样本的取值来进行计算得到；

而协方差则必须要求至少满足二维特征；方差是协方差的特殊情况。 $Cov(X, X)$ 就是X的方差。

(2) 方差和协方差的除数是n-1,这是为了得到方差和协方差的无偏估计。

协方差为正时，说明X和Y是正相关关系；协方差为负时，说明X和Y是负相关关系；协方差为0时，说明X和Y是相互独立。

当样本是n维数据时，它们的协方差实际上是协方差矩阵(对称方阵)。例如，对于3维数据(x,y,z)，计算它的协方差就是：

$$Cov(X, Y, Z) = \begin{bmatrix} Cov(x, x) & Cov(x, y) & Cov(x, z) \\ Cov(y, x) & Cov(y, y) & Cov(y, z) \\ Cov(z, x) & Cov(z, y) & Cov(z, z) \end{bmatrix}$$

### 3.PCA算法两种实现方法

#### 3.1基于特征值分解协方差矩阵实现PCA算法

##### 3.1.1过程（记忆这个）

输入：数据集  $X = \{x_1, x_2, x_3, \dots, x_n\}$ ，需要降到k维。

1) 去平均值(即去中心化)，即每一位特征减去各自的平均值。、

$$\frac{1}{n} XX^T$$

2) 计算协方差矩阵  $\frac{1}{n} XX^T$ ，注：这里除或不除样本数量n或n-1，其实对求出的特征向量没有影响。

$$\frac{1}{n} XX^T$$

3) 用特征值分解方法求协方差矩阵  $\frac{1}{n} XX^T$  的特征值与特征向量。

4) 对特征值从大到小排序，选择其中最大的k个。然后将其对应的k个特征向量分别作为行向量组成特征向量矩阵P。

5) 将数据转换到k个特征向量构建的新空间中，即 $Y = PX$ 。

##### 3.1.2实例

以X为例，我们用PCA方法将这两行数据降到一行。

$$X = \begin{pmatrix} -1 & -1 & 0 & 2 & 0 \\ -2 & 0 & 0 & 1 & 1 \end{pmatrix}$$

解：

1) 因为X矩阵的每行已经是零均值，所以不需要去平均值。

2) 求协方差矩阵：

$$C = \frac{1}{5} \begin{pmatrix} -1 & -1 & 0 & 2 & 0 \\ -2 & 0 & 0 & 1 & 1 \end{pmatrix} \begin{pmatrix} -1 & -2 \\ -1 & 0 \\ 0 & 0 \\ 2 & 1 \\ 0 & 1 \end{pmatrix} = \begin{pmatrix} \frac{6}{5} & \frac{4}{5} \\ \frac{4}{5} & \frac{6}{5} \end{pmatrix}$$

[https://blog.csdn.net/program\\_developer](https://blog.csdn.net/program_developer)

3) 求协方差矩阵的特征值与特征向量。

求解后的特征值为：

$$\lambda_1 = 2, \lambda_2 = \frac{2}{5}$$

对应的特征向量为：

$$c_1 \begin{pmatrix} 1 \\ 1 \end{pmatrix}, c_2 \begin{pmatrix} -1 \\ 1 \end{pmatrix}$$

其中对应的特征向量分别是一个通解，C1和C2可以取任意实数。那么标准化后的特征向量为：

$$\begin{pmatrix} \frac{1}{\sqrt{2}} \\ \frac{1}{\sqrt{2}} \end{pmatrix}, \begin{pmatrix} -\frac{1}{\sqrt{2}} \\ \frac{1}{\sqrt{2}} \end{pmatrix}$$

4) 矩阵P为：

$$P = \begin{pmatrix} \frac{1}{\sqrt{2}} & \frac{1}{\sqrt{2}} \\ -\frac{1}{\sqrt{2}} & \frac{1}{\sqrt{2}} \end{pmatrix}$$

5) 最后我们用P的第一行乘以数据矩阵X，就得到了降维后的表示：

$$Y = \left( \frac{1}{\sqrt{2}} \quad \frac{1}{\sqrt{2}} \right) \begin{pmatrix} -1 & -1 & 0 & 2 & 0 \\ -2 & 0 & 0 & 1 & 1 \end{pmatrix} = \left( -\frac{3}{\sqrt{2}}, -\frac{1}{\sqrt{2}}, 0, \frac{3}{\sqrt{2}}, -\frac{1}{\sqrt{2}} \right)$$

结果如图1所示：

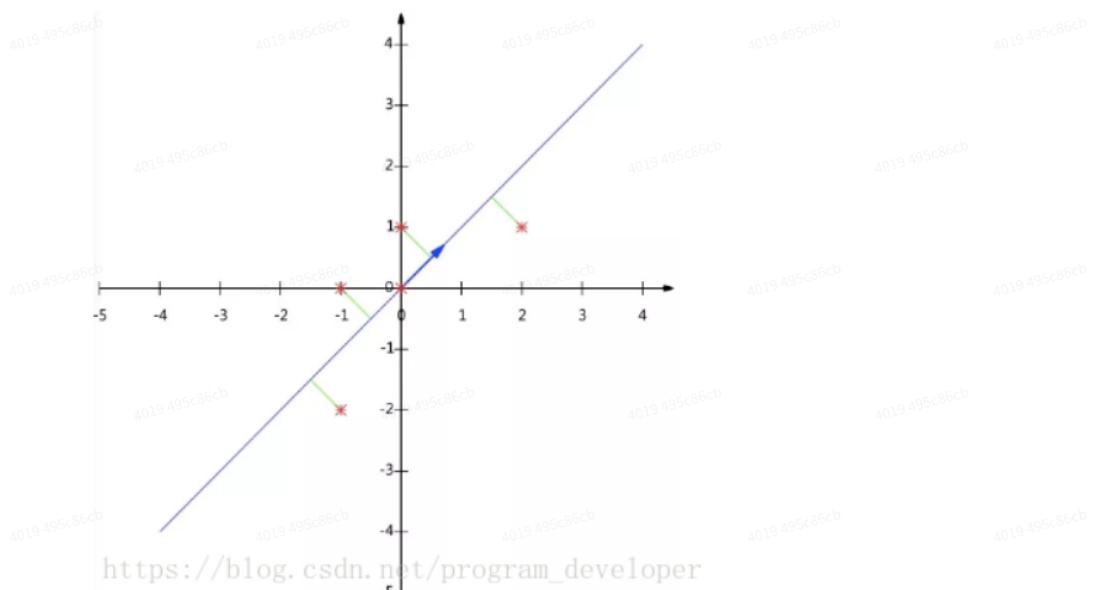


图1：数据矩阵X降维投影结果

## 4. 选择降维后的维度K(主成分的个数)

如何选择主成分个数K呢？先来定义两个概念：

- average squared projection error :  $\frac{1}{m} \sum_{i=1}^m \|x^{(i)} - x_{approx}^{(i)}\|^2$ , 其中 $x_{approx}$ 为映射值。
- total variation in the data :  $\frac{1}{m} \sum_{i=1}^m \|x^{(i)}\|^2$

选择不同的K值，然后用下面的式子不断计算，选取能够满足下列式子条件的最小K值即可。

$$\frac{\frac{1}{m} \sum_{i=1}^m \|x^{(i)} - x_{approx}^{(i)}\|^2}{\frac{1}{m} \sum_{i=1}^m \|x^{(i)}\|^2} \leq t$$

其中t值可以由自己定，比如t值取0.01，则代表了该PCA算法保留了99%的主要信息。当你觉得误差需要更小，你可以把t值设置的更小。



## 2.1 KPCA

KPCA，中文名称“核主成分分析”，是对PCA算法的非线性扩展，言外之意，PCA是线性的，其对于非线性数据往往显得无能为力。**KPCA能够挖掘到数据集中蕴含的非线性信息。**

其实很好理解，就是原始的数据线性不可分，那么就升维变成线性可分。这个过程就需要使用核函数。升维后线性可分那么就采用PCA。

从  $XX^T$  操作变成对  $\phi(x)\phi(x^T)$

### 1. 理论部分

1. 为了更好处理非线性数据，引入**非线性映射函数  $\phi(x)$** ，将原空间中的数据映射到高维空间。
2. 引入了一个定理：**空间中的任一向量（哪怕是基向量），都可以由该空间中的所有样本线性表示。**

假设中心化后的样本集合X（d\*N，N个样本，维数d维，样本”按列排列“），现将X映射到高维空间，得到  $\phi(x)$ ，假设在这个高维空间中，本来在原空间中线性不可分的样本现在线性可分了，然后呢？想啥呢！果断上PCA啊！

于是乎！假设D ( $D \gg d$ ) 维向量 **Wi** 为高维空间中的特征向量，为对应的特征值  $\lambda_i$ ，高维空间中的PCA如下：

$$\Phi(X)\Phi(X)^T w_i = \lambda_i w_i \quad (1)$$

这个时候，在利用刚才的定理，将**特征向量Wi 利用样本集合  $\phi(x)$  线性表示**，如下：

$$w_i = \sum_{k=1}^N \alpha_i \Phi(x_i) = \Phi(X)\alpha \quad (2)$$

然后，在把  $w_i$  ( $i = 1, \dots, d$ ) 代入上上公式，得到如下的形式：

$$\Phi(X)\Phi(X)^T \Phi(X)\alpha = \lambda_i \Phi(X)\alpha \quad (3)$$

进一步，等式两边同时左乘  $\Phi(X)^T$ ，得到如下公式：

$$\Phi(X)^T \Phi(X)\Phi(X)^T \Phi(X)\alpha = \lambda_i \Phi(X)^T \Phi(X)\alpha \quad (4)$$

这样做的目的是，构造两个 $\Phi(X)$ 、 $\Phi(X)^T$ 出来，进一步用核矩阵K（为对称矩阵）替代其中：

$$\Phi(X)^T \Phi(X) \Phi(X)^T \Phi(X) \alpha = \lambda_i \Phi(X)^T \Phi(X) \alpha \quad (5)$$

于是，公式进一步变为如下形式：

$$K^2 \alpha = \lambda_i K \alpha \quad (6)$$

两边同时去除K，得到了PCA相似度极高的求解公式：

$$K \alpha = \lambda_i \alpha \quad (7)$$

求解公式的含义就是求K最大的几个特征值所对应的特征向量，由于K为对称矩阵，所得的解向量彼此之间肯定是正交的。

但是，请注意，这里的 $\alpha$ 只是K的特征向量，但是其不是高维空间中的特征向量，回看公式(2)，高维空间中的特征向量w应该是由 $\alpha$ 进一步求出。

## 2. 核函数

(1) 线性核函数(可视为特例)

$$K(x, x_i) = x \cdot x_i ;$$

(2)  $p$  阶多项式核函数

$$K(x, x_i) = [(x \cdot x_i) + 1]^p ;$$

(3) 高斯径向基函数(RBF)核函数

$$K(x, x_i) = \exp\left(-\frac{\|x - x_i\|}{\sigma^2}\right) ;$$

(4) 多层感知器(MLP)核函数

$$K(x, x_i) = \tanh[\nu(x \cdot x_i) + c] ;$$



## 2.2 线性判别分析LDA

线性判别分析的基本思想是将高维的模式样本投影到最佳鉴别矢量空间，以达到抽取分类信息和压缩特征空间维数的效果，投影后保证模式样本在新的子空间有 **最大的类间距离** 和 **最小的类内距** 离，即模式在该空间中有最佳的可分离性。

### 1. LDA假设以及符号说明：

假设对于一个  $R^n$  空间有  $m$  个样本分别为  $x_1, x_2, \dots, x_m$  即 每个  $x$  是一个  $n$  行的矩阵，其中  $n_i$  表示属于  $i$  类的样本个数，假设有一个有  $c$  个类，则  $n_1 + n_2 + \dots + n_i + \dots + n_c = m$ 。

$S_b$  ..... 类间离散度矩阵

$S_w$  ..... 类内离散度矩阵

$n_i$  ..... 属于  $i$  类的样本个数

$x_i$  ..... 第  $i$  个样本

$u$  ..... 所有样本的均值

$u_i$  ..... 类  $i$  的样本均值

### 2. 公式推导，算法形式化描述

根据符号说明可得类  $i$  的样本均值为：

$$u_i = \frac{1}{n_i} \sum_{x \in \text{class } i} x \quad (1)$$

同理我们也可以得到总体样本均值：

$$u = \frac{1}{m} \sum_{i=1}^m x_i \quad (2)$$

根据类间离散度矩阵和类内离散度矩阵定义，可以得到如下式子：

$$S_b = \sum_{i=1}^c n_i (u_i - u)(u_i - u)^T \quad (3)$$

$$S_w = \sum_{i=1}^c \sum_{x_k \in \text{class } i} (u_i - x_k)(u_i - x_k)^T \quad (4)$$

LDA作为一个分类的算法，我们当然希望它所分的类之间耦合度低，类内的聚合度高，即类内离散度矩阵中的数值要小，而类间离散度矩阵中的数值要大，这样的分类的效果才好。



# 3. 特征工程

## 1. Filter: 过滤法

① Filter: 过滤法

根据每个属性的一些指标(如差异), 来确定这个属性的重要性, 然后对所有属性按照重要程度排序, 从高到低的选择属性。

## 2. Embedding: 嵌入式

② Embedding: 嵌入式

嵌入

把特征选择的过程作为学习过程的一部分, 在学习的过程中进行特征选择。即先使用某些机器学习的算法和模型进行训练, 得到各个特征的权值系数, 根据权值系数从大到小选择特征。这些权值系数往往代表了特征对于模型的贡献或重要性。(决策树)

## 3. Wrapper: 包裹式

③ Wrapper

也是特征选择与算法训练同时进行的方法, 可以调用 coef\_ 或 feature\_importance\_ 属性来完成特征选择。

但不同的是, 我们往往使用一个目标函数作为黑盒来帮助我们选取特征。包裹法在初始特征集上训练评估器, 并且通过 coef\_ 属性或 feature\_importances\_ 属性获得每个特征的重要性。然后, 从当前的一组特征中移除最不重要的特征。在移除的集合上递归地重复该过程, 直到最终到达所需要的特征数。

区别于过滤法和嵌入法的二次训练解决所有问题, 包裹法要使用特征子集进行多次训练, 因此包裹法的成本较高。

## 最典型的方法：递归特征消除法 (RFE).

它是一种贪心的优化算法，首先找到性能最佳的特征子集。它反复创建模型，并在每次迭代时保留最佳特征或剔除最差特征，下一次迭代时，它会使用上次及模中没有被选中的特征来构建下一个模型，直到所有特征都被剔为止。  
然后，根据自己保留或剔除特征的顺序来对特征进行排名。



扫描全能王 创

Date: \_\_\_ / \_\_\_ / \_\_\_ Page: \_\_\_

最终选出一个最佳子集。

包裹法是所有特征选择方法中最有利于提升模型表现的，它可以使用很多的特征，达到很优秀的效果。但其计算量大，不太适合太大型的数据。

# 4. 树模型对特征重要性进行评估

## 1. 随机森林 (RF) 简介

随机森林的算法可以用如下几个步骤概括：

- 1) 用有抽样放回的方法 (bootstrap) 从样本集中选取n个样本作为一个训练集
- 2) 用抽样得到的样本集生成一棵决策树。在生成的每一个结点：

### 2.1 随机不重复地选择d个特征

2.2 利用这d个特征分别对样本集进行划分，找到最佳的划分特征（可用基尼系数、增益率或者信息增益判别）

- 3) 重复步骤1到步骤2共k次，k即为随机森林中决策树的个数。
- 4) 用训练得到的随机森林对测试样本进行预测，并用票选法决定预测的结果。

## 2. 特征重要性评估

计算每个特征在随机森林中的每棵树上做了多大的贡献，然后取个平均值，最后比一比特征之间的贡献大小。

好了，那么这个贡献是怎么一个说法呢？通常可以用基尼指数（Gini index）或者袋外数据（OOB）错误率作为评价指标来衡量。

### 2.1 基于基尼系数

我们将变量重要性评分 (variable importance measures) 用 VIM 来表示，将 Gini 指数用 GI 来表示，假设有 J 个特征  $X_1, X_2, X_3, \dots, X_J$ ，I 棵决策树，C 个类别，现在要计算出每个特征  $X_j$  的 Gini 指数评分  $VIM_j^{(Gini)}$ ，亦即第 j 个特征在 RF 所有决策树中节点分裂不纯度的平均改变量。

第 i 棵树节点 q 的 Gini 指数的计算公式为

$$GI_q^{(i)} = \sum_{c=1}^{|C|} \sum_{c' \neq c} p_{qc}^{(i)} p_{qc'}^{(i)} = 1 - \sum_{c=1}^{|C|} (p_{qc}^{(i)})^2 \quad (3-1)$$

$p_{qc}$  的含义：节点 q 中类别 c 所占的比例。

特征 $X_j$ 在第*i*棵树节点 $q$ 的重要性，即节点 $q$ 分枝前后的Gini指数变化量为

$$VIM_{jq}^{(Gini)(i)} = GI_q^{(i)} - GI_l^{(i)} - GI_r^{(i)} \quad (3-2)$$

其中， $GI_l^{(i)}$ 和 $GI_r^{(i)}$ 分别表示分枝后两个新节点的Gini指数。

如果，特征 $X_j$ 在决策树*i*中出现的节点为集合 $Q$ ，那么 $X_j$ 在第*i*颗树的重要性为

$$VIM_j^{(Gini)(i)} = \sum_{q \in Q} VIM_{jq}^{(Gini)(i)} \quad (3-3)$$

假设RF中共有*I*颗树，那么

$$VIM_j^{(Gini)} = \sum_{i=1}^I VIM_j^{(Gini)(i)} \quad (3-4)$$

最后，把所有求得的重要性评分做一个归一化处理即可。

$$VIM_j^{(Gini)} = \frac{VIM_j^{(Gini)}}{\sum_{j'=1}^J VIM_{j'}^{(Gini)}} \quad (3-5)$$

# 1. 数据降维与SVD

## 3. SVD

### 3.1 矩阵论预备知识

#### 3.1.1 特征值、特征向量

如果一个向量 $v$ 是矩阵 $A$ 的特征向量，将一定可以表示成下面的形式：

$$Av = \lambda v$$

其中， $\lambda$ 是特征向量 $v$ 对应的特征值，一个矩阵的一组特征向量是一组正交向量。

**思考：**为什么一个向量和一个数相乘的效果与一个矩阵和一个向量相乘的效果是一样的呢？

**答案：**矩阵 $A$ 与向量 $v$ 相乘，本质上是对向量 $v$ 进行了一次**线性变换**（旋转或拉伸），而该变换的效果为常数 $\lambda$ 乘以向量 $v$ 。

当我们求特征值与特征向量的时候，就是为了求矩阵 $A$ 能使哪些向量（特征向量）只发生伸缩变换（**线性**），而变换的程度可以用特征值 $\lambda$ 表示。

#### 3.1.2 特征值分解

对于矩阵 $A$ ，有一组特征向量 $v$ ，将这组向量进行正交化单位化，就能得到一组正交单位向量。**特征值分解**，就是将矩阵 $A$ 分解为如下式：

$$A = Q\Sigma Q^{-1}$$

其中， $Q$ 是矩阵 $A$ 的特征向量组成的矩阵，不同的特征值对应的特征向量线性无关。同时对于实对称矩阵而言，不同的特征向量必定正交。

$\Sigma$ 矩阵是一个对角阵，对角线上的元素就是特征值。

**实例：**

这里我们用一个简单的方阵来说明特征值分解的步骤。我们的方阵 $A$ 定义为：

$$A = \begin{pmatrix} -1 & 1 & 0 \\ -4 & 3 & 0 \\ 1 & 0 & 2 \end{pmatrix}$$

首先，由方阵A的特征方程，求出特征值。

$$|A - \lambda E| = \begin{vmatrix} -1 - \lambda & 1 & 0 \\ -4 & 3 - \lambda & 0 \\ 1 & 0 & 2 - \lambda \end{vmatrix} = (2 - \lambda) \begin{vmatrix} -1 - \lambda & 1 \\ -4 & 3 - \lambda \end{vmatrix} = (2 - \lambda)(\lambda - 1)^2 = 0$$

特征值为  $\lambda = 2, 1$  (重数是2)。

然后，把每个特征值  $\lambda$  带入线性方程组  $(A - \lambda E)x = 0$ ，求出特征向量。

当  $\lambda = 2$  时，解线性方程组  $(A - 2E)x = 0$ 。

$$(A - 2E) = \begin{pmatrix} -3 & 1 & 0 \\ -4 & 1 & 0 \\ 1 & 0 & 0 \end{pmatrix} \rightarrow \begin{pmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 0 \end{pmatrix}$$

$$p_1 = \begin{pmatrix} 0 \\ 0 \\ 1 \end{pmatrix}$$

解得  $x_1 = 0, x_2 = 0$ 。特征向量为：

当  $\lambda = 1$  时，解线性方程组  $(A - E)x = 0$

$$(A - E) = \begin{pmatrix} -2 & 1 & 0 \\ -4 & 2 & 0 \\ 1 & 0 & 1 \end{pmatrix} \rightarrow \begin{pmatrix} 1 & 0 & 1 \\ 0 & 1 & 2 \\ 0 & 0 & 0 \end{pmatrix}$$

$$n_2 = \begin{pmatrix} -1 \\ -2 \end{pmatrix}$$

$$x_1 + x_3 = 0, \quad x_2 + 2x_3 = 0。特征向量为：$$

$$P^2 = \begin{pmatrix} -4 \\ 1 \end{pmatrix}.$$

最后，方阵A的特征值分解为：

$$A = Q\Sigma Q^{-1} = \begin{pmatrix} 0 & -1 & -1 \\ 0 & -2 & -2 \\ 1 & 1 & 1 \end{pmatrix} \begin{pmatrix} 2 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 1 \end{pmatrix} \begin{pmatrix} 0 & -1 & -1 \\ 0 & -2 & -2 \\ 1 & 1 & 1 \end{pmatrix}^{-1}$$

我们来分析一下特征值分解的式子，分解得到的Σ矩阵是一个对角矩阵，里面的特征值是由大到小排列的，**这些特征值所对应的特征向量就是描述这个矩阵变换方向**（从主要的变化到次要的变化排列）。

矩阵是高维的情况下，那么Σ矩阵就是高维空间下的一个线性变换，这个线性变换可能没法通过图片来表示，但是可以想象，这个变换也同样有很多的变化方向，**我们通过特征值分解得到的前N个特征向量，就对应了这个矩阵最主要N个变化方向**。我们利用这前N个变化方向，就可以近似这个矩阵变换。也就是之前说的：**提取这个矩阵最重要的特征**。

## 总结：

特征值分解可以得到特征值与特征向量。

**特征值表示的是这个特征到底有多么重要，而特征向量表示这个特征是什么**，可以将每一个特征向量理解为一个线性的子空间，我们可以利用这些线性的子空间干很多事情。

不过，特征值分解也有很多的局限，比如说变换的矩阵必须是方阵。当矩阵不是方阵的时候，这个时候就需要使用SVD对非方阵矩阵进行分解。

## 3.2 SVD分解

### 3.2.1思想

奇异值分解是一个能适用于**任意矩阵**的一种分解的方法，对于任意矩阵A总是存在一个奇异值分解：

$$A = U\Sigma V^T$$

假设A是一个 $m \times n$ 的矩阵。

那么得到的U是一个 $m \times m$ 的方阵，U里面的正交向量被称为**左奇异向量**。

$\Sigma$ 是一个 $m \times n$ 的矩阵， $\Sigma$ 除了对角线其它元素都为0，**对角线上的元素称为奇异值**。

$V^T$ 是V的转置矩阵，是一个 $n \times n$ 的矩阵，它里面的正交向量被称为**右奇异值向量**。

由于U，V都是**正交阵**，可以得到：

$$U^T U = I, V^T V = I$$

而且一般来讲，我们会将 $\Sigma$ 上的值按从大到小的顺序排列。上面矩阵的维度变化可以参照图4所示

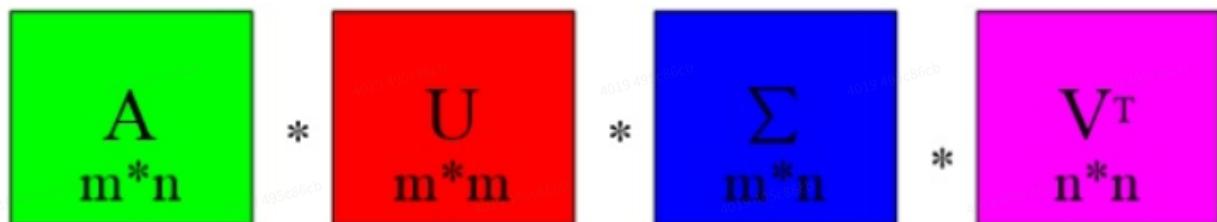


图4：奇异值分解中各个矩阵维度变化

### 3.2.2计算奇异值，奇异值向量

#### 1. 奇异值向量

把奇异值和特征值联系起来。先构造出一个方阵A出来。

首先，我们用矩阵A的转置乘以A，得到一个方阵，用这样的方阵进行特征分解，得到的特征值和特征向量满足下面的等式：

$$(A^T A)v_i = \lambda_i v_i$$

这里的 $v_i$ 就是我们要求的右奇异向量。 (Why? ? ? )

我们说 ATA 的特征向量组成的矩阵就是我们 SVD 中的 V 矩阵(why?)

证明：

$$A = U\Sigma V^T \Rightarrow A^T = V\Sigma^T U^T \Rightarrow \boxed{A^T A} = V\Sigma^T U^T U\Sigma V^T = \boxed{V\Sigma^2 V^T}$$

所以 ATA 的特征向量就是我们要求的右奇异值向量。

同理，我们将 A 和 A 的转置做矩阵的乘法，得到一个方阵，用这样的方阵进行特征分解，得到的特征和特征向量满足下面的等式：

$$(AA^T)u_i = \lambda_i u_i$$

这里的 $u_i$ 就是左奇异向量。

## 2. 奇异值

奇异值求法有两种：

方法一：

$$A = U\Sigma V^T \Rightarrow AV = U\Sigma V^T V \Rightarrow \boxed{AV = U\Sigma} \Rightarrow Av_i = \sigma_i u_i \Rightarrow \sigma_i = \frac{Av_i}{u_i}$$

方法二：

通过下面可以看出：

$$A = U\Sigma V^T \Rightarrow A^T = V\Sigma^T U^T \Rightarrow \boxed{A^T A} = V\Sigma^T U^T U\Sigma V^T = \boxed{V\Sigma^2 V^T}$$

ATA 的特征值矩阵等于奇异值矩阵的平方， (不能是 AAT，维度不匹配) 也就是说特征值和奇异值满足如下关系：

$$\sigma_i = \sqrt{\lambda_i}$$

### 3.2.3 SVD 的意义

**思考：**我们已经知道如何用奇异值分解任何矩阵了，那么问题又来了，一个 $m \times n$ 的矩阵A，你把它分解成 $m \times m$ 的矩阵U、 $m \times n$ 的矩阵Σ和 $n \times n$ 的矩阵 $V^T$ 。这三个矩阵中任何一个的维度似乎一点也不比A的维度小，而且还要做两次矩阵的乘法，这不是没事找事干嘛！把简单的事情搞复杂了么！并且我们知道矩阵乘法的时间复杂度为 $O(n^3)$ 。○那奇异值分解到底要怎么做呢？

在奇异值分解矩阵中Σ里面的奇异值按从大到小的顺序排列，奇异值从大到小的顺序减小的特别快。**在很多情况下，前10%甚至1%的奇异值的和就占了全部的奇异值之和的99%以上。也就是说，剩下的90%甚至99%的奇异值几乎没有作用。**因此，我们可以用前面r个大的奇异值来近似描述矩阵，于是奇异值分解公式可以写成如下：

$$A_{m \times n} \approx U_{m \times r} \Sigma_{r \times r} V_{r \times n}^T$$

其中r是一个远远小于m和n的数，右边的三个矩阵相乘的结果将会使一个接近A的矩阵。如果r越接近于n，则相乘的结果越接近于A。如果r的取值远远小于n，从计算机内存的角度来说，右边三个矩阵的存储内存要远远小于矩阵A的。**所以在奇异值分解中r的取值很重要，就是在计算精度和时间空间之间做选择。**

## 3.3 SVD分解的应用

### 3.3.1 降维

通过奇异值分解的公式，我们可以很容易看出来，原来矩阵A的特征有n维。经过SVD分解后，**可以用前r个非零奇异值对应的奇异向量表示矩阵A的主要特征**，这样就把矩阵A进行了降维。

### 3.3.2 压缩

通过奇异值分解的公式，我们可以看出来，矩阵A经过SVD分解后，**要表示原来的大矩阵A，我们只需要存储U、Σ、V三个较小的矩阵即可**。而这三个较小规模的矩阵占用内存上也是远远小于原有矩阵A的，这样SVD分解就起到了压缩的作用。

# 1. 最大似然估计

## 1. 实例 (引出背景)

实例：现有黄球，红球若干个在一个布袋中（无序，不放回）  
要求其分布特点。

X	X=黄	X=红
P	θ	1-θ

(估计)

从袋中抽 100 个球（随机地），以样本试整体分布：

取出：40 个黄球，60 个红球

最大似然估计的原则：存在即合理

为什么是 40 个黄球，60 个红球；而不是其他结果  
只能说明是因为抽取 40 个黄球，60 个红球的组合概率最大

或者说，在数次条件下，抽 40 个黄球，60 个红球概率是最大的。

上述问题的解为：

$$\text{则 } L(\theta) = \theta^{40} (1-\theta)^{60} \Rightarrow \ln L(\theta) = 40 \ln \theta + 60 \ln(1-\theta)$$

$$\text{对 } L(\theta) \text{ 求导, } \Rightarrow \frac{dL(\theta)}{d\theta} = 40 \cdot \frac{1}{\theta} - \frac{60}{1-\theta} = 0$$

$$\text{极值点 } \Rightarrow \theta = \frac{2}{5}$$

所以当  $\theta = \frac{2}{5}$  时，抽取 100 个球，其中 40 个黄球，60 个红球概率最大；

所以我们就认为在原始样本中， $\theta = \frac{2}{5}$

因为只有在  $\theta = \frac{2}{5}$  时，我们才能一次抽样中，抽 40 个黄球，60 个红球。

实例 2：

计算  $\mu, \sigma$

Date: \_\_\_\_\_ Page: \_\_\_\_\_

实例2：现需要调查一个岁数男性的身高分布，已知其服从分布  $\Omega = (\mu, \sigma)$

方法：先随机采样，100个男生，记录他们的身高。

男性的身高分别为  $X_1, X_2, \dots, X_{100}$

现在认为：在  $\mu$  与  $\sigma$  的条件下，抽取出是  $(X_1, X_2, \dots, X_{100})$  的概率最大。

而抽取出  $(X_1, X_2, \dots, X_{100})$  的概率是：

$$\prod_{i=1}^{100} P(X_i; \Omega).$$

如何找到  $\mu$  与  $\sigma$  使  $L(\theta) = L(X_1, X_2, \dots, X_{100}; \theta) = \prod_{i=1}^n P(X_i; \theta)$

最大。求得极值就知  $\mu, \sigma$ 。

则利用最大似然理论计算出的样本分布特征有： $\mu, \sigma$ 。

## 2. 似然函数

反映的是在不同参数  $\theta$  取值下，取得当前样本集的可靠性。

因此，参数  $\theta$  相对于样本集  $X$  的似然函数，记为  $L(\theta)$

$$L(\theta) = L(X_1, \dots, X_n; \theta) = \prod_{i=1}^n P(X_i; \theta)$$

现在，让  $L(\theta)$  最大，求一个

$$\hat{\theta} = \arg \max \{L(\theta)\}$$

 扫描全能王 仓

Date: \_\_\_\_\_ Page: \_\_\_\_\_

试：求参数，令参数为 0，得到似然方程。

解似然方程，得到参数即为所求。

## 3. 总结

总结：一种统计方法，去估计样本的分布特征值  $\mu, \sigma$  等。

原因是总体样本过多，我们无法直接统计，所以抽取部分样本出来辅助计算。

现在考虑一个问题：

为什么在原始分布情况下，抽取出来的样本是这些，而不是其他



在原始分布情况下，这些样本被同时抽取的概率最大



写出组合概率公式（似然函数），最大组合概率

↑  
最大似然函数



对似然函数求极值，得最大似然点，就是原始分布中的频率量值。

## 2. EM算法

实例2 进阶：

现在知道该校学生总体身高分布为  $\mathcal{N}(\mu, \sigma^2)$ ,

但：你现在只知道有一个人是该校男生，问其身高期望为多少？

【男女身高有别，不能直接利用总体  $\mathcal{N}(\mu, \sigma^2)$ ，否则性别信息被忽略】

现在问题成为：

我们抽取的每个样本都不知道从哪个分布抽取的。

现在要估计男女各自分布特征。

即知道总体分体，求男女各自分布。

## 1. EM算法

假设我们想估计  $A$  与  $B$  两个参数，在开始状态下二者都是未知的，如果知道了  $A$  的信息就可以得到  $B$  的信息。反过来知  $B \Rightarrow A$

可以考虑首先赋予  $A$  某种初值，以此得到  $B$  的估计值，然后从  $B$  的当前值出发，重新估计  $A$  的取值，这个过程一直持续到收敛为止。

对于身高而言，已知每个人的身份，但不知性别。

要求男女身高各自服从正态分布，则需要首先知道其性别，然后根据性别分开这些人，对男没分别使用极大似然估计法。

最大期望算法：

E步：先随便猜一下男生（身高）的正态分布参数，如男生均值1.7m，方差0.1m。

然后测出每个人更属于这个分布，是男 → 性别：男  
否 → 性别：女

M步：将上面每个人分为男没两部分后，

我们对其分别利用最大似然估计法，计算两分布的参数。

在更新完两分布参数后，每个样本属于这两个分布的概率再次改变，则继续E,M如此往复，直到参数不再发生变化为止。

## 2. 算法推导

假设我们有一个样本集  $x^{(1)}, x^{(2)}, \dots, x^{(n)}$ ，包含  $n$  个独立样本。

但每个样本对应的类别  $z^{(i)}$  是未知的（相当于聚类），即称为隐含变量。

由于有隐含变量限制，所以无法直接使用最大似然求解。

但是，对于参数估计，我们本质上还想获得一个使似然函数最大的参数  $\theta$ ，  
现在只不过似然函数式中多了个未知量  $z$

$$L(\theta) = \prod_{i=1}^n P(x^{(i)}, \theta) = \prod_{i=1}^n \sum_{z^{(i)}} P(x^{(i)}, z^{(i)}, \theta)$$

目标：找到合适的  $\theta$  与  $z$  使  $L(\theta)$  最大。

首先给出一个公式，计算  
未知数，再利用未知数  
参数，依次往复。

notebook