中国科学技术大学
University of Science and Technology of China

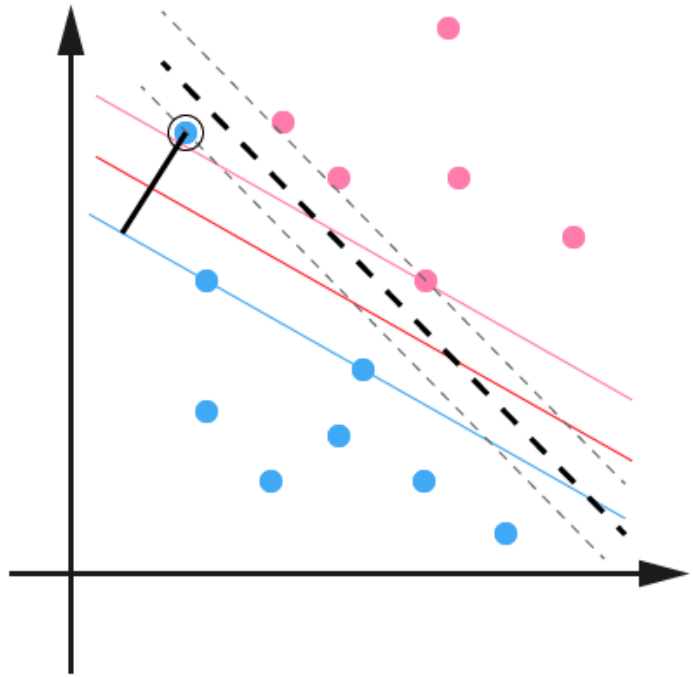
人工智能/机器学习/数据挖掘

非线性处理

April 14, 2022

Outline

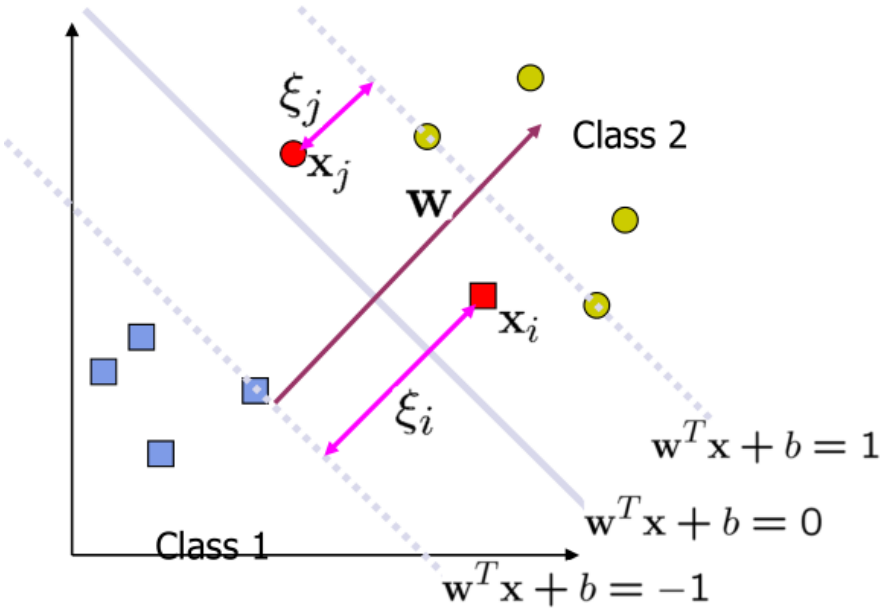
问题描述



如图所示，线性不可分或不同类支持向量靠得太近

- 原因可能是测量误差造成的噪声，比如黑圈内的蓝点，作为 sv，其几何间隔由黑色平行虚线间隔确定；而忽略黑圈内蓝点，得到红色实线确定的几何间隔更大
- 若黑圈内的蓝点受噪声影响偏离较大，那么我们得到的黑虚线判决界面过分地信任“错误/噪声数据”，对今后的判决会造成泛化能力变差；
- 因此，我们尝试将某些数据看成是有噪声的，因此，在他们被选择为支持向量的时候，进行“松弛”

松弛变量法



- **约束条件**：$\forall(x_i, y_i) \in D_{train}$
  $1-(W\cdot x+b)y \leq 0 \Longrightarrow$
  $1-[(W\cdot x_i+b)y_i+\xi_i] \leq 0$
- **目标函数**：
  $\Longrightarrow \min_{W,\xi}(\frac{1}{2}W\cdot W+C\sum_i \xi_i)$
- **训练误差**：原来线性可分时，误差为 $0 \Longrightarrow \sum_i \xi_i$

松弛变量法

- 如图所示，假设真实/我们想要的分类面如灰实线所示，x_i 错分了，“越界”了，x_j “调皮”或因为噪声，跑到“禁区”里了
- 添加 ξ_i, ξ_j 把 x_i, x_j 给“拉回”各自的“支持向量”边界。
- 我们称 ξ_i, ξ_j 为松弛变量
- 上述操作，体现在公式中或问题中，如右上所示。

松弛变量法

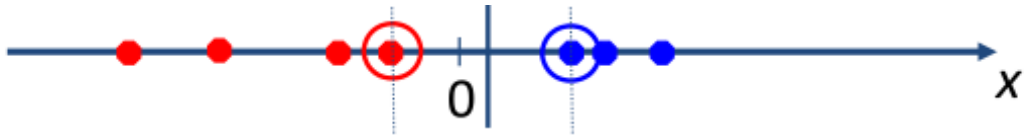
SVM 中训练误差的产生

- 被平移了的数据点，虽然成了 sv，但是它们带来了训练数据的误差/损失，非支持向量带来 0 误差/损失
- 两类支持向量，它们的拉格朗日乘法因子 $\alpha_i > 0$
- 一类 $\xi_i = 0$, 它们是真正的 sv，损失/误差为 0
- 一类 $\xi_i > 0$, 它们是平移后的 sv，损失/误差为一类 ξ_i
- 当获得最优解时，$\xi_i = 1 - (W \cdot x_i + b)y_i$，这就是样本 (x_i, y_i) 的 $hinge$ 损失

正则化：引入新的优化目标

- $\min_{W,\xi}(\frac{1}{2} W \cdot W + C\sum_i \xi_i)$，也即 $\min_{W,\xi}(\frac{1}{2C} W \cdot W + \sum_i loss_{hinge})$
- 其中原线性 SVM 的优化目标 $\min_W \frac{1}{2} W \cdot W$ 被称为正则化项，训练集上的 $hinge$ 损失 $\sum_i \xi_i$ 称为训练误差最小化项
- 新优化目标既考虑了训练误差最小，又考虑控制模型复杂性（正则化项，也可以说是结构风险最小化）
- 参数 C，用于平衡两个优化目标之间的相对重要性。

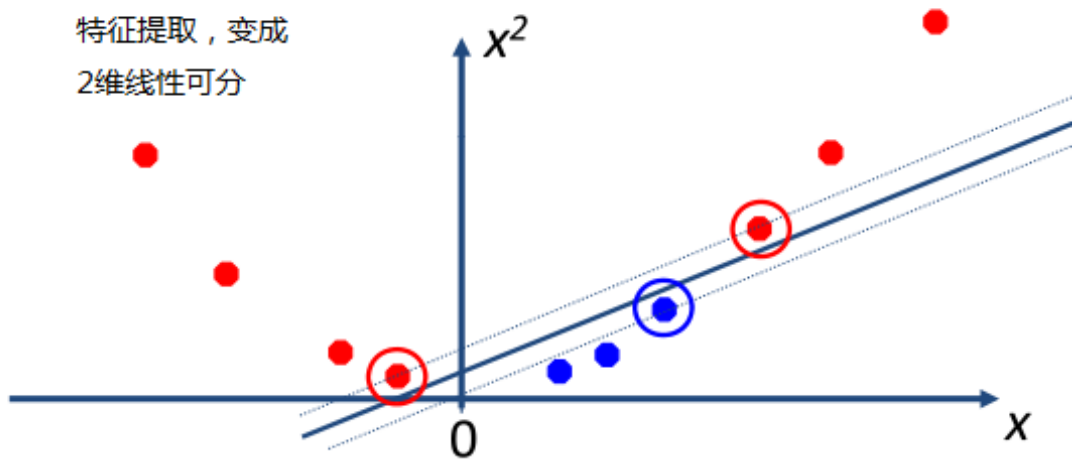
非线性可分



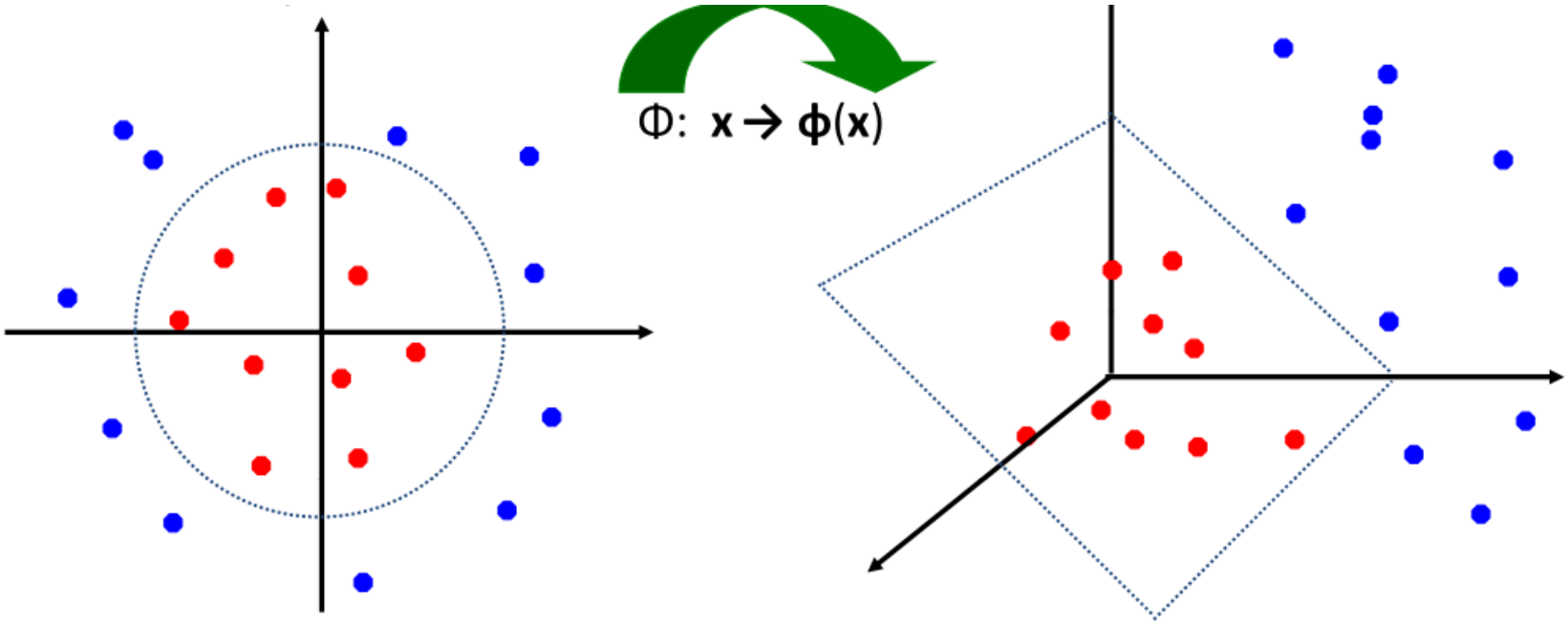
线性可分：1维的例子

线性不可分：1维的例子

特征提取，变成
2维线性可分



非线性可分



解释说明

- 如上图所示，从 2 维 $\Longrightarrow$3 维空间，红蓝点用平面线性可分
- 从低维空间 $\Longrightarrow$ 高维空间，不同类的数据可在高维空间中用线性超平面分开
- 问题：如何映射到高维空间/特征空间？如何确保在高维空间中线性可分？ 聚焦在“特征提取函数”！

特征提取函数

例子

- $\mathbf{x} = [x_1, x_2]^t$
- $\phi([x_1, x_2]^t) = [1, \sqrt{2}x_1, \sqrt{2}x_2, x_1^2, x_2^2, \sqrt{2}x_1x_2]^t$
- 2 维输入变成了 6 维特征，实现了升维，这是一个特征提取函数

特征提取函数的得与失

- 得：线性不可分 可能变成线性可分；
- 失：时间和空间代价增加，把数据集变换为新的高维空间的数据集。

能避免所失吗？

线性 SVM 处理非线性情形

SVM

- 假设所有数据形成的空间结构中，存在这样的超平面，可以完美地将数据分开，那么
- 依据 $W = \sum_i \alpha_i y_i x_i, b = y_i - W \cdot x_i, \forall (x_i, y_i)$ is a sv, SVM 得到线性 $h(x) = \sum_i \alpha_i y_i \phi(x_i) \cdot \phi(x) + b$, where (x_i, y_i) is a sv
- 上式表明，获得分类器 h 后，对任何新数据进行预测时，只需要将它与支持向量进行内积，然后求加权和即可。
- 如果数据分布不好，不存在这样的超平面，怎么办？
- 特征提取函数可以把在低维空间中线性不可分的数据集映射到高维空间，可能实现线性可分。

如果能直接获得 $K(x_i, x) = \phi(x_i) \cdot \phi(x)$ 的值，不需要显式地先分别计算出 $\phi(x_i), \phi(x)$，然后求内积，就可以避免一些特征提取函数带来的时空代价！直接先定义 $K(x_i, x)$，不用执行把数据集从低维空间变换到高维特征空间的操作。那是否线性可分？(先放下别管！)

核函数/kernel function

核函数/Kernel function

- 称 $K(x_i, x_j)$ 为核函数，它等于 x_i, x_j 两个变量升维（特征提取 $<$ 怎么提取的？不关心！ $>$）后的特征向量的内积；
- 在线性 SVM 优化过程中，出现特征向量的地方，都是求两个特征向量的内积。

例子

- $\mathbf{x} = [x_1, x_2]^t, \phi([x_1, x_2]^t) = [1, \sqrt{2}x_1, \sqrt{2}x_2, x_1^2, x_2^2, \sqrt{2}x_1x_2]^t$
  $\phi(\mathbf{x}) \cdot \phi(\mathbf{a}) = 1 + 2a_1x_1 + 2a_2x_2 + a_1^2x_1^2 + a_2^2x_2^2 + 2a_1a_2x_1x_2$
  设计一个核函数：$K(\mathbf{x}, \mathbf{a}) = \phi(\mathbf{x}) \cdot \phi(\mathbf{a}) = (1 + \mathbf{x} \cdot \mathbf{a})^2$
- 常用的，可供我们选择的核函数有：
  - $K(x, a) = (1 + x \cdot a)^2$ 多项式核函数,上例的核函数
  - $K(x, a) = x \cdot a$ 内积核函数
  - $K(x, a) = e^{-\frac{\|x-a\|^2}{2\sigma^2}}$ 高斯核函数

核函数的意义

意义

- 核函数是我们“假设存在一个特征提取函数，把数据集映射到了高维特征空间”，核函数就是高维特征空间的内积计算公式；
- 核函数对应的特征提取函数，把数据集映射到了一个线性可分的高维空间了吗？能检验吗？一般无法检验或确保高维特征空间中的数据集线性可分，此时缺省就采用松弛变量的方法，把“捣乱”的数据看成是噪声。
- 理解核函数另一种观点：描述了两个数据之间的“相似性”，是一种 x_i, x_j 之间的距离/相似性的计算方法；
- 错误的理解：核函数把数据 x 映射到了高维特征空间！

核函数的更多知识

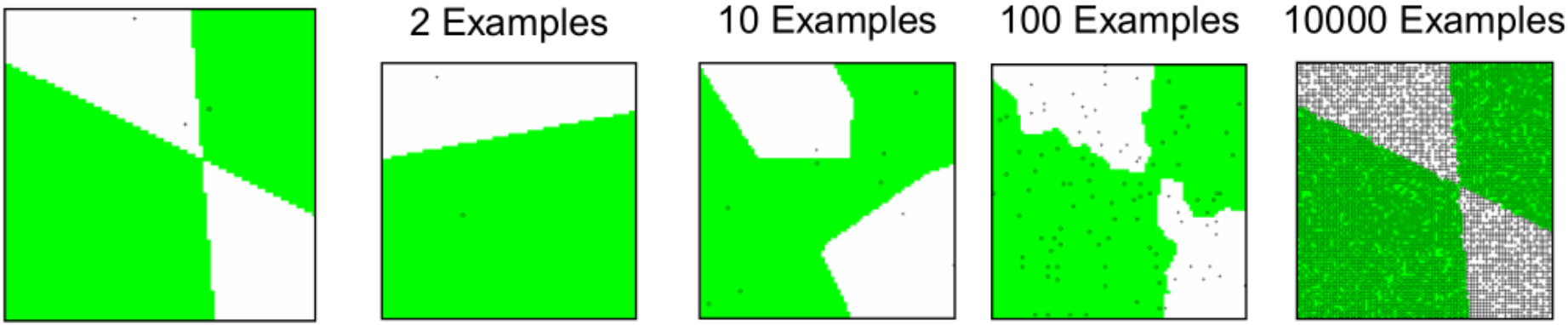
再生核希尔伯特空间

- 有兴趣参考两份文献：
- 《AN INTRODUCTION TO THE THEORY OF REPRODUCING KERNEL HILBERT SPACES》，VERN I. PAULSEN
- 打开网址：
  https://en.wikipedia.org/wiki/Reproducing_kernel_Hilbert_space

kernel embedding of distributions

- 请感兴趣的同学自行查阅相关资料。
- 如：打开网址
  https://en.wikipedia.org/wiki/Kernel_embedding_of_distributions

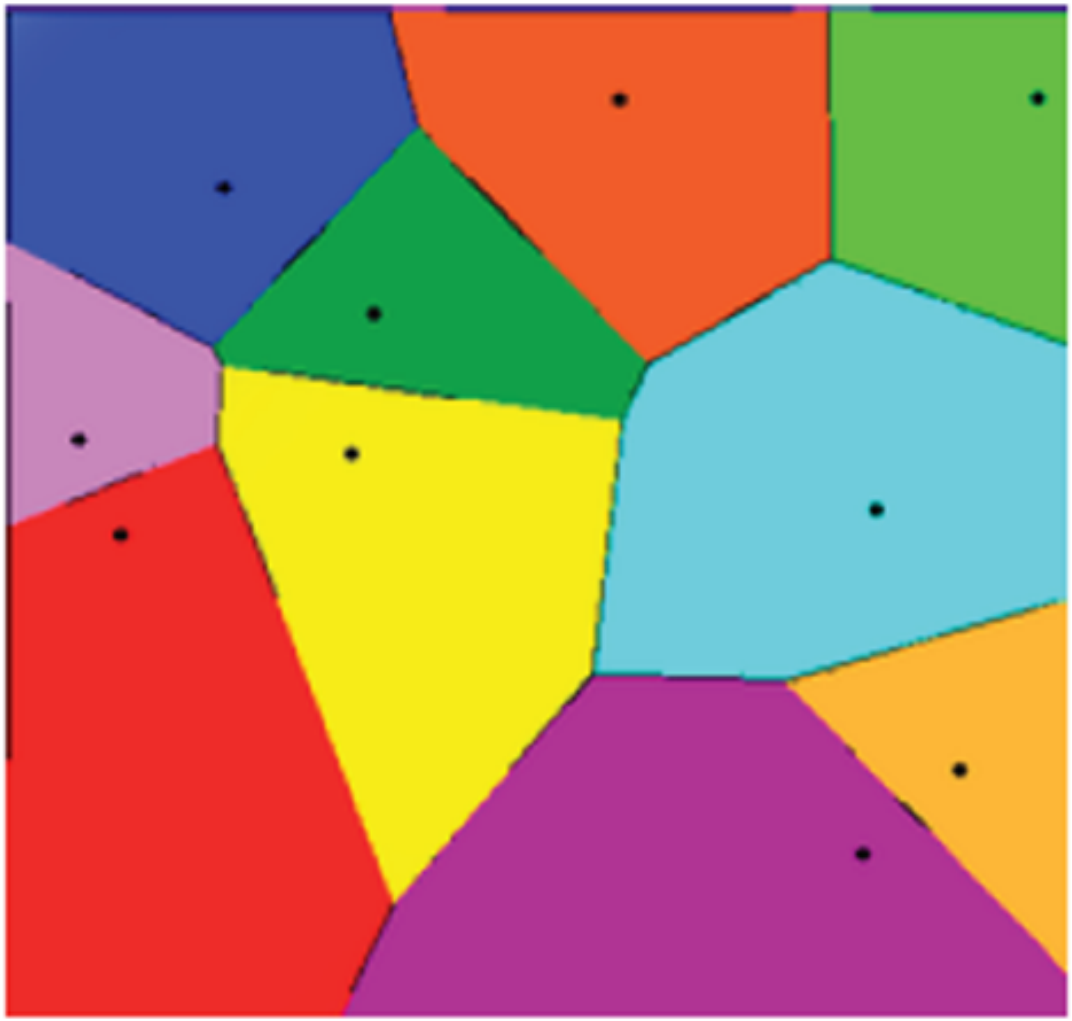
K 近邻方法：样本增加



例子：训练样本数目越多，分类精度越高

- 左图给出了平面上数据真正分布：不同颜色代表不同类（绿和白）
- 右图采用 1-近邻方法来判决：判决任意未知样本的色彩为离它最近的已知样本的色彩（KNN 就是由最近 K 个已知样本投票决定）
- 只有两个已知样本时，二者的中垂线就是分类界面；随着已知样本的数目增加，分类边界越来越复杂
- 当有 1000 个已知样本时，分类边界由很多短小的“线段”连接成一个复杂的分类边界，同时准确性越来越高，逼近真正分类边界。

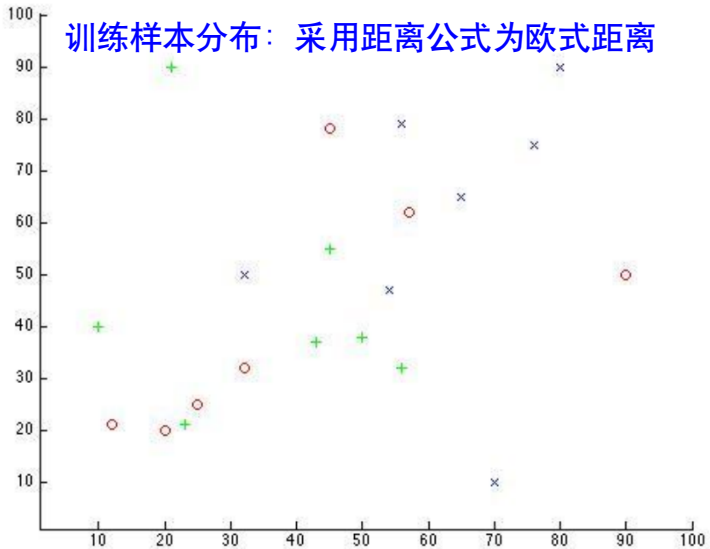
K 近邻方法：边界更复杂

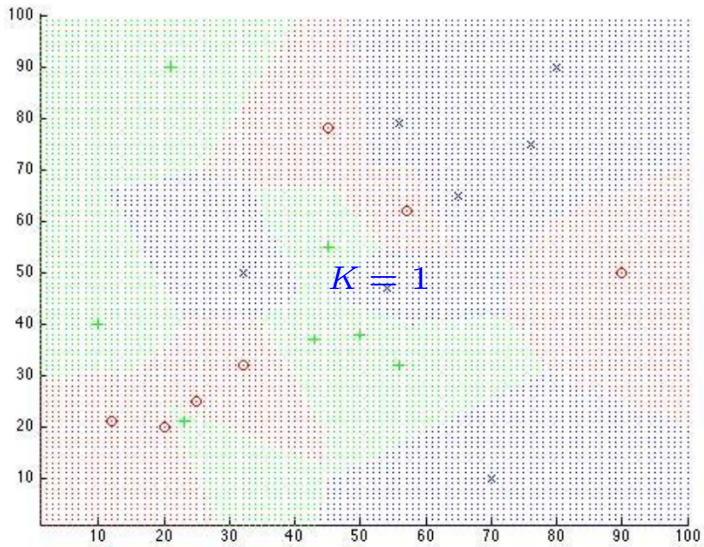


解释说明

- 1-nn 的例子
- 样本越多，决策边界越复杂，很多直线组合在一起，用非线性边界划分了整个平面

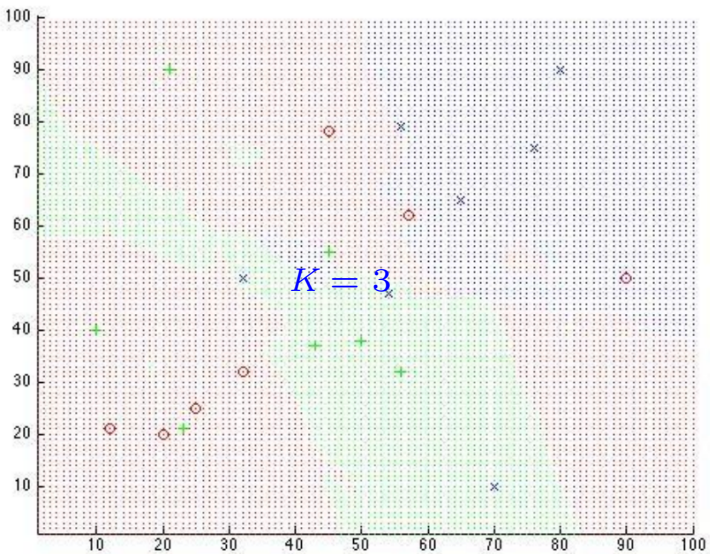
K 近邻方法：K 的影响

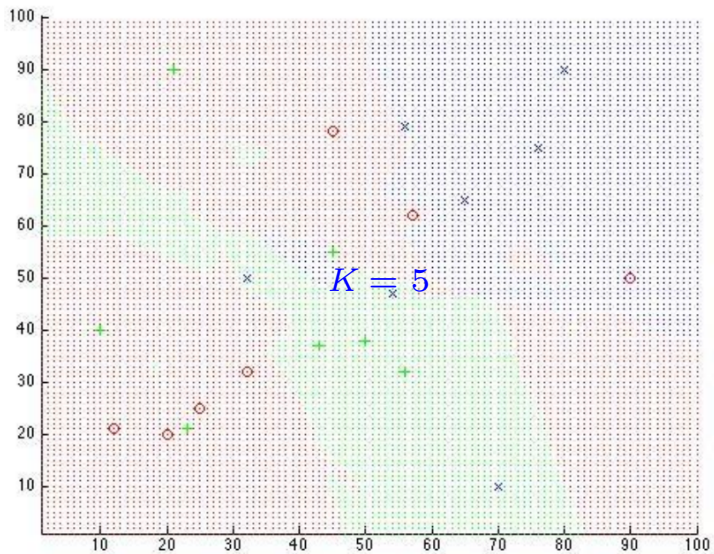




不同颜色代表不同的类

K 越大，边界越光滑





KNN: 算法描述

训练过程

- 将所有已知样本存储起来。

预测/分类过程: 在给定距离定义下,

- 计算新数据/未知类别数据的 K 个最近邻居
- 以 K 个邻居中占多数的类标号作为新数据的类标号

算法评述

- 优点:
  - 训练简单，基本是 0 代价
  - 可解释性好
- 缺点
  - 预测代价大，找最近的 K 个邻居，算法时间复杂度不低
  - 样本数目太少时，易过拟合，泛化能力不足

参数方法与非参数方法

非参数方法

- 参数的个数随样本数目增加而增加
- 参数可以理解为决策边界的细节；样本越多，参数越多，分类器复杂性越高。
- 想象一下，完整映射表几乎快填完整了，此时再来一个新数据，判定错误的可能性，直觉上就会觉得变小了。
- KNN 是非参数方法

参数方法

- 相对应的参数方法，参数个数是固定的
- 如线性分类器，权向量 $\mathbf{W}$ 的维数不会因样本增加而增加。
- 通常，参数方法中，样本越多，参数设置得会更合理。

KNN 算法扩展 1

机械学习算法：KNN 预测新数据代价太大

- 训练过程：存储所有的训练样本
- 预测过程：随机产生类标号给新样本/或直接返回“不知道答案”

评述

- 训练和 KNN 一样，觉得 KNN 预测过程太复杂，改进了一下，几乎不费代价就能做到预测
- 按照“最小化训练集损失”的思想，该算法训练误差为 0，也就是完美拟合了训练数据
- 想想线性回归/分类问题，都在追求“最小化训练集损失”，这里为什么这么容易其它就实现了？(我们在其它方面失去了东西！)
- 失去的东西/付出的代价，即决策边界的复杂性：相对于线性 $h(x)$ 的简单，KNN 描述复杂性增加了！
- 这也就是所谓的，因为模型太复杂，过拟合了训练数据，泛化能力可能不够。
- 解决办法：正则化，约束模型的复杂度。该算法中模型极度复杂。
- 同时也说明了，仅仅追求“最小化训练集损失”是不够的，不应该是我们最终的目标，而应该是完整映射表 (所有数据上的) 损失最小，实践中用“最小化测试集损失”（泛化能力）来代替。

KNN 算法扩展 2

算法描述

- 训练过程：找到训练集中出现次数最多的 y 值
- 预测过程：对任何新样本，其类标号为该 y 值

算法评述

- 该算法可以理解为 KNN 的 K 变成和训练数据集大小一样，该算法也简化了 KNN 的预测过程
- 这个算法它没有复杂的分类边界，没有过拟合训练数据，泛化能力非常好
- 但是预测能力很差，原因是太简单了，找到的规律近乎于“常识”，能最大化压缩数据，整个数据集就用一个 y 值就存储了！
- 所谓泛化能力，是指“训练集上的损失”和“测试集上的损失”是一致的
- 该算法中模型极度简单。

一些概念的整理

过拟合与泛化能力

- 过拟合：定义在训练数据集上，训练误差非常小的情况。可能造成的原因：模型太复杂，太“迁就特殊的训练数据”，而训练数据本身可能因为“噪声”就存在不精确的问题。
- 泛化能力：无法在所有数据上计算误差，折衷利用测试集上的误差，该误差和训练误差接近即认为是泛化能力好。

模型/假设的复杂性和正则化

- 完整映射表具有最大描述复杂性，KNN 的复杂性随样本增加而增加；
- 正则化可以认为是人为“强行”在模型中添加“预定义”的知识，以此来控制描述复杂性。比如 KNN 算法扩展 2

KNN 与 SVM 的比较

能否用 SVM 来改进 KNN

- SVM 得到的支持向量是不是可以用在 KNN 中，当成是构建 KNN 分类器的数据？
- 复杂的 KNN 决策边界用 sv 来简化？
- SVM 的中的分类界面表达式中有核函数，如内积/点积运算，核函数可以理解为两个数据的相似性，也就是预测的过程就是综合新数据和 sv 之间的相似性，最后给出一个判决。
- 因此，SVM 只是用核函数定义了一种新的距离计算公式，压缩了 KNN 的大量样本，保留了一些关键样本（sv）用于构建分类器/距离计算公式，而 KNN 设计的关键之一就是定义合适的距离计算公式。

例子：语义描述

张三去吃饭，没座了，等吗？考虑以下影响因素：

- Alternate: is there an alternative restaurant nearby?
- Bar: is there a comfortable bar area to wait in?
- Fri/Sat: is today Friday or Saturday?
- Hungry: are we hungry?
- Price: price range ($, $$, $$$)
- Raining: is it raining outside?
- Patrons: number of people in the restaurant (None, Some, Full)
- Reservation: have we made a reservation?
- Type: kind of restaurant (French, Italian, Thai, Burger)
- WaitEstimate: estimated waiting time (0-10, 10-30, 30-60, >60)

所有情形共计有：$2^6 \times 3^2 \times 4^2 = 9216$

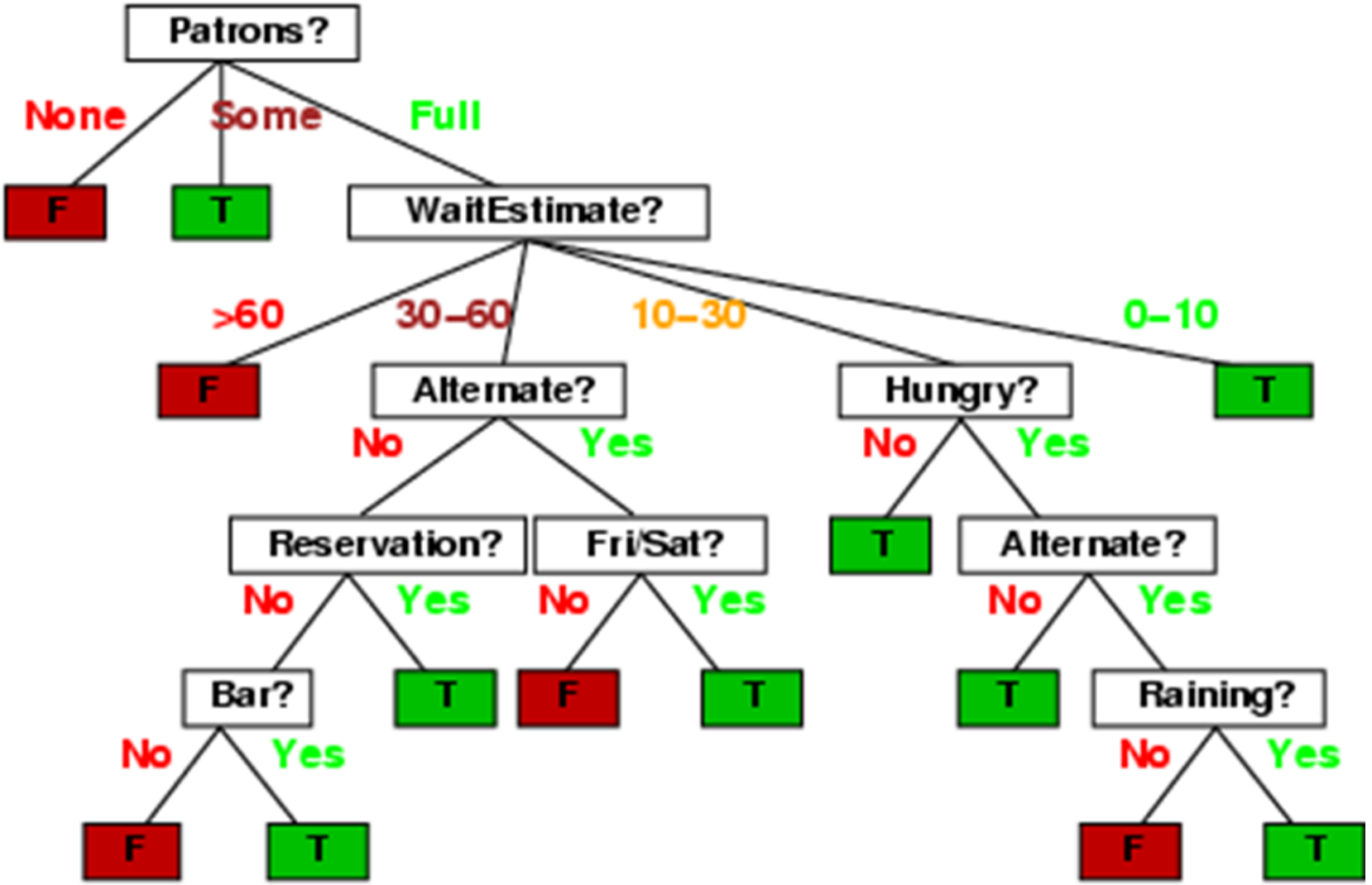
例子：训练数据集

Example	Attributes										Target
	Alt	*Bar*	*Fri*	*Hun*	*Pat*	*Price*	*Rain*	*Res*	*Type*	*Est*	*Wait*
X_1	T	F	F	T	Some	$$$	F	T	French	0–10	T
X_2	T	F	F	T	Full	$	F	F	Thai	30–60	F
X_3	F	T	F	F	Some	$	F	F	Burger	0–10	T
X_4	T	F	T	T	Full	$	F	F	Thai	10–30	T
X_5	T	F	T	F	Full	$$$	F	T	French	>60	F
X_6	F	T	F	T	Some	$$	T	T	Italian	0–10	T
X_7	F	T	F	F	None	$	T	F	Burger	0–10	F
X_8	F	F	F	T	Some	$$	T	T	Thai	0–10	T
X_9	F	T	T	F	Full	$	T	F	Burger	>60	F
X_{10}	T	T	T	T	Full	$$$	F	T	Italian	10–30	F
X_{11}	F	F	F	F	None	$	F	F	Thai	0–10	F
X_{12}	T	T	T	T	Full	$	F	F	Burger	30–60	T

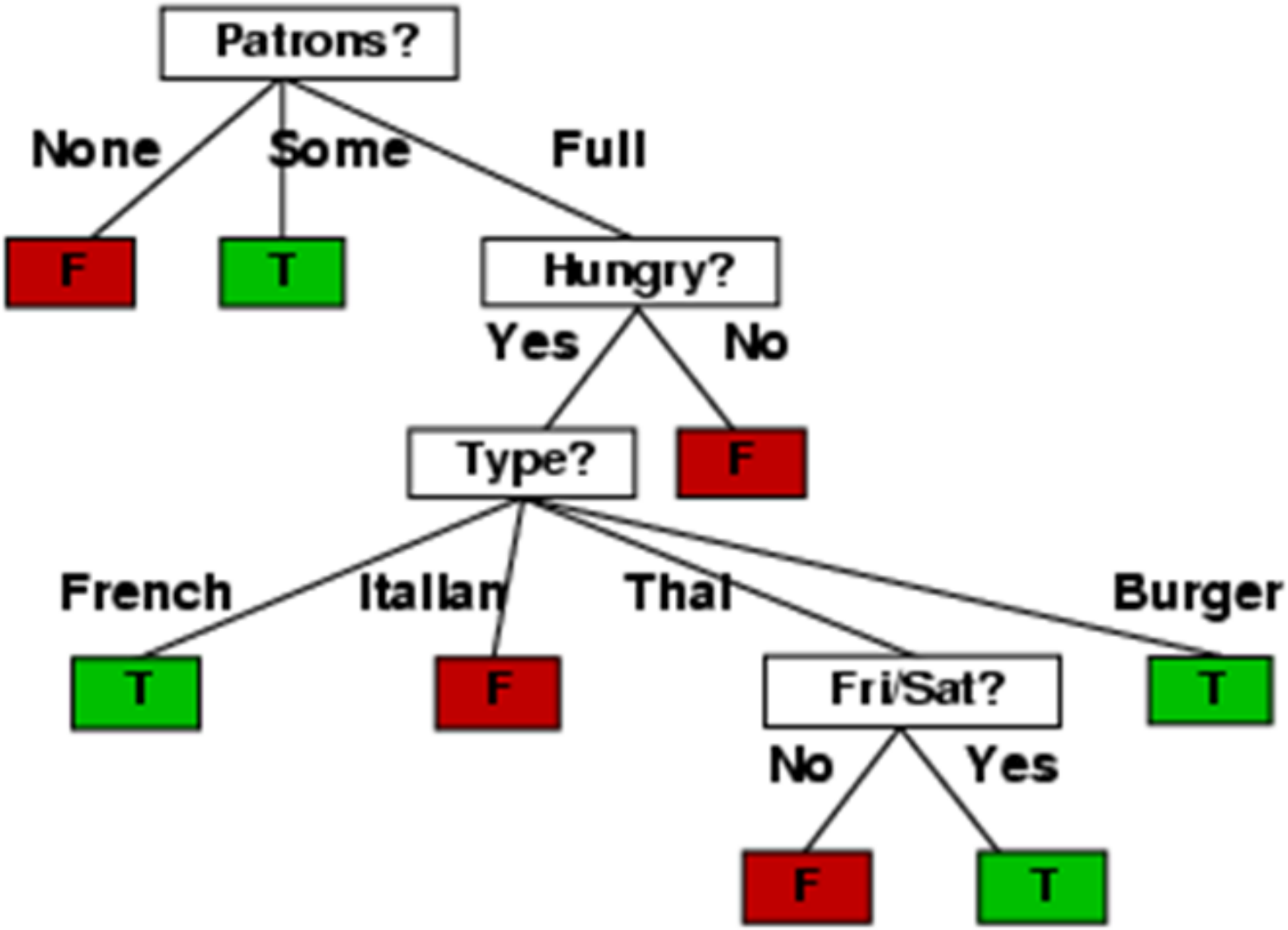
张三曾经去吃饭遇到的各种情形及其应对结果

- 12 次经历，即 12 个样本数据
- 今天周三，天下着雨，餐馆里坐满了人，张三会等着吃法国餐吗？

例子：判决树/决策树


Patrons?
None
Some
Full
F
T
WaitEstimate?
>60
30–60
10–30
0–10
F
Alternate?
Hungry?
T
No
Yes
No
Yes
Reservation?
Fri/Sat?
T
Alternate?
No
Yes
No
Yes
No
Yes
Bar?
T
F
T
T
Raining?
No
Yes
No
Yes
F
T
F
T

例子：判决树/决策树


Patrons?
None
Some
Full
F
T
Hungry?
Yes
No
Type?
F
French
Italian
Thai
Burger
T
F
Fri/Sat?
T
No
Yes
F
T

决策树描述

决策树静态结构

- 树形结构，每个节点是一个简单的线性决策器 $h_i(x) = A_i$，依据属性 A_i 的取值不同划分为不同的类
- 决策树中任意一个非叶节点有两个特点：拥有一个训练数据集的一个子集 D_i，对应数据的一个列/属性（称为分割属性）
- 决策树中任何一层中所有节点的训练数据子集互不相交，且并集为完整的训练数据集
- 叶节点不包括任何数据，只有一个标记父节点中数据所属的类别号。

决策树动态部分

- 训练过程：即构建决策树的过程，确定每个非叶结点的分割属性；并分割节点拥有的数据子集给孩子节点；
- 分类/预测过程：沿从树根开始到叶节点的一条路径，依次使用路过节点的线性分类器。
- 一种特殊“树形”组合多个线性决策器的方法。

决策树的构造

决策树的构造

- 在初始时刻，构建树根节点，树根拥有所有训练数据 D_{train}
- 对任何一个节点 N_i，选择数据的某一个属性/特征 A，以 A 的不同取值，把节点 N_i 拥有的训练数据集 D_i 分成若干个不相交的子集，每个子集变成节点 N_i 的一个子节点（每个子节点拥有训练数据的一个子集）；
- 当某个节点内的所有训练数据都属于同一类（y 值相同）时，该节点的划分过程结束，它生成唯一一个子节点（叶节点），叶节点只有一个属性：它的父节点拥有的数据集所属的类标号。

选择数据属性 A 做分割属性的时候，要一一测试所有的候选属性，依据测试结果选择一个最好的， 何谓最好？

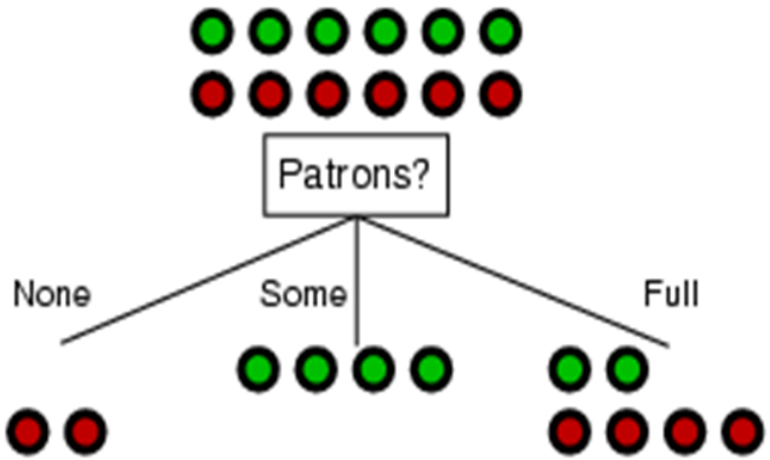
关键技术一

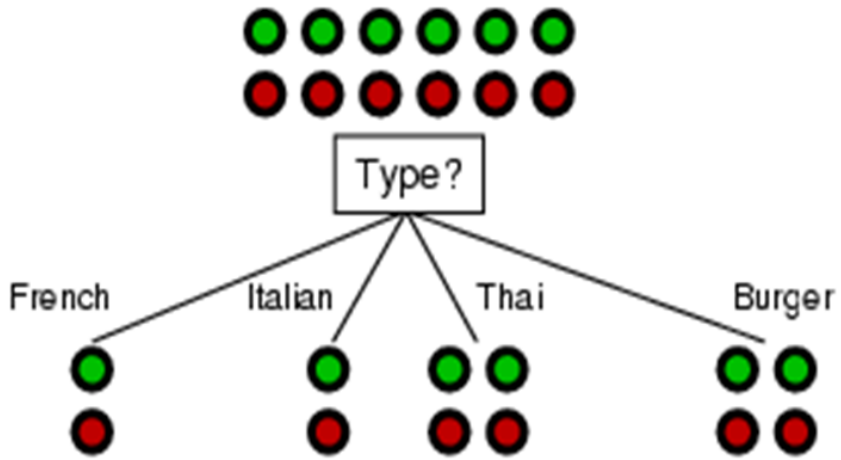
分割属性的选择

- 模仿人的决策过程
- 买车/买房/买手机时，你考虑哪些因素？每个因素就对应商品的一个特征；
- 人做决策的时候就一般会把自己最看重的“品质”第一个考虑（根节点？），然后在满足第一个“品质”要求的商品中，选择第二看重的“品质”，依次类推；
- 给你一个数据集，是人们购买商品的记录，包括产品功能、质量、价格在内的信息，虽然每个人对商品的“品质”看法具有个性，但是你能不能从中找到共性，让计算机自动找到大部分顾客最看重的“品质”？
- 对生产厂商、顾客都非常重要。

问题：如何从数据集中找到最被看重的属性！

例子：选择哪一个分割属性





例子

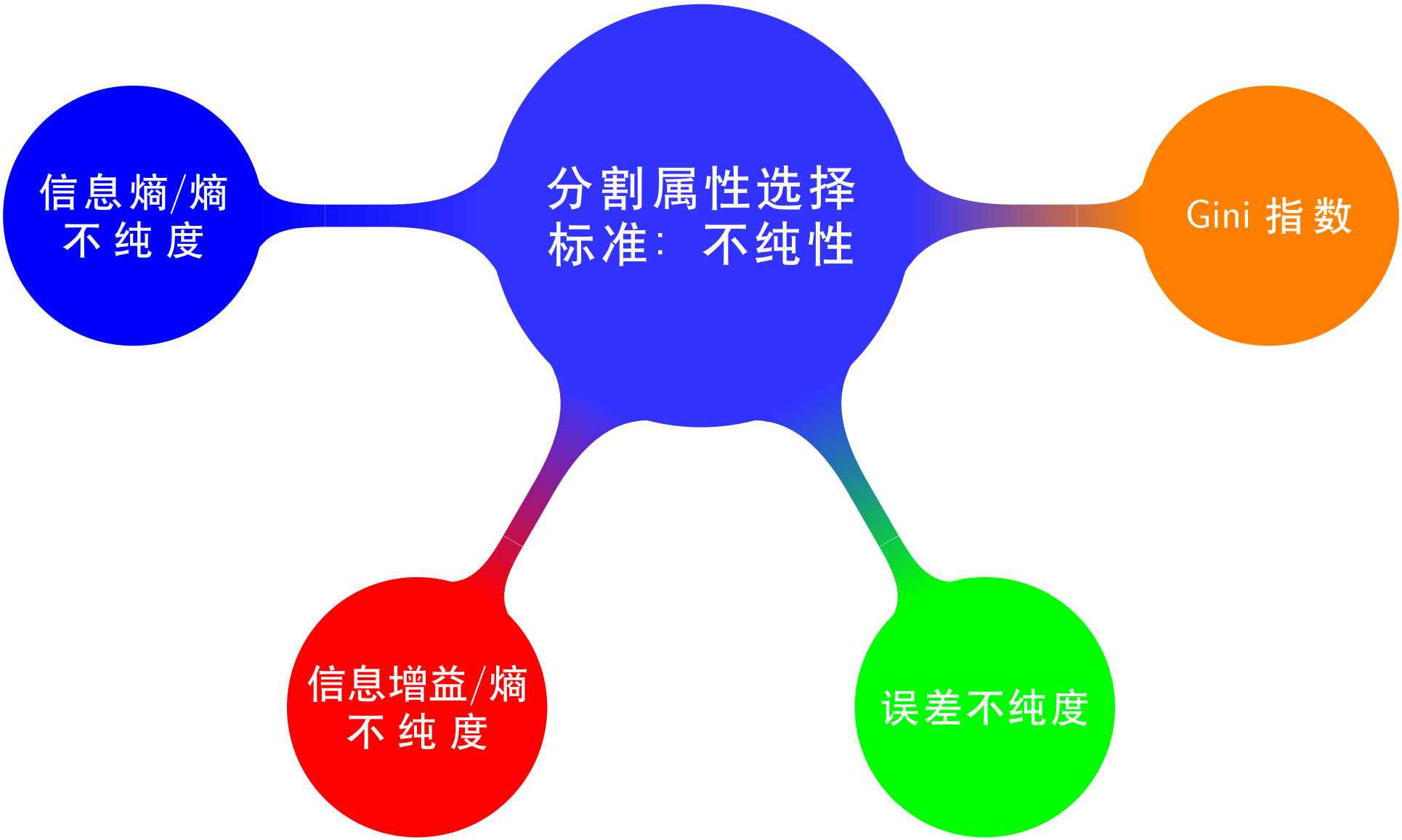
- Patrons 比 Type 更好
- why ?

描述属性的分割能力

数据集合的不纯性

- 假设采用了某个属性划分了数据集，我们通过划分结果的好坏来定义属性分割数据集能力
- 分割属性选择的标准：
- 纯：意指分割完后的子集异类的数据越少越好，将来可以很快地把“不纯”的数据划分掉，降低子集未来划分出的节点数目
- 不纯：比“纯度”的定义和计算更直接和简单一些。集合内的数据全部属于一类，则不纯度为 0，均匀来自 2 类或多类，则不纯度最大

不纯性的度量方法



熵不纯度

信息熵

- 给定随机变量的概率分布向量 $(p_1, p_2, \ldots, p_m)$，信息熵即概率向量的对数期望值：$H = -\sum_i p_i \log p_i$，量化该随机变量的不确定性。
- 假设：集合中有无穷多 m 种不同的球，每种球的占比服从上述概率分布；若每个球的抽取概率相等；随机抽取一个球出来，当上述概率分布相等时，抽出哪一种球是最“混乱”无规律的，不确定性最大的；当集合最“纯”时，几乎全是一种球，那么随机抽取出一个球，很大可能都是确定的。

熵不纯度

- 给定决策树节点 N，定义其熵不纯度为 $i(N) = H(N) = -\sum_i p_i \log p_i$
- 其中 p_i 是节点 N 拥有数据集 D 中，属于第 i 类的数据在 D 中的占比

熵不纯度

应用方法：给定节点 N 确定采用哪个属性来分割

- 计算 $i(N) = H(N)$
- 对所有的属性 A_i 依次执行下述计算：
  - 依据 A_i 的不同取值，获得多个子节点及其拥有的子数据集 D_i
  - 对所有的子节点 N_i 计算 $i(N_i) = H(N_i)$;
  - 计算信息增益：$IG(N|A_i) = H(N) - \sum_i \frac{|N_i|}{|N|} H(N_i) \triangleq H(N) - H(N|A_i)$，即不确定性降低的量

ID3 算法

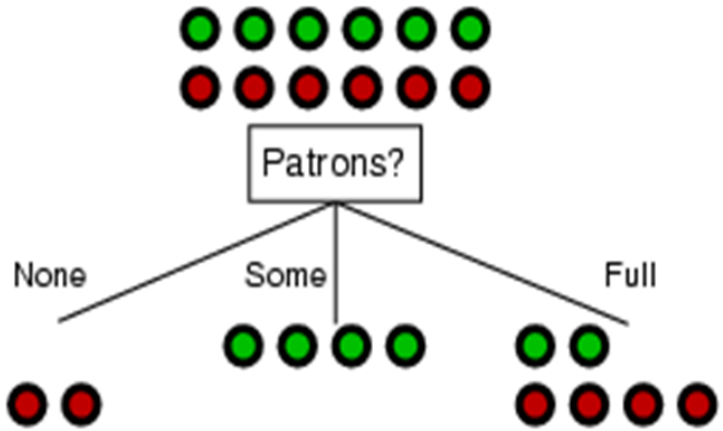
熵不纯度

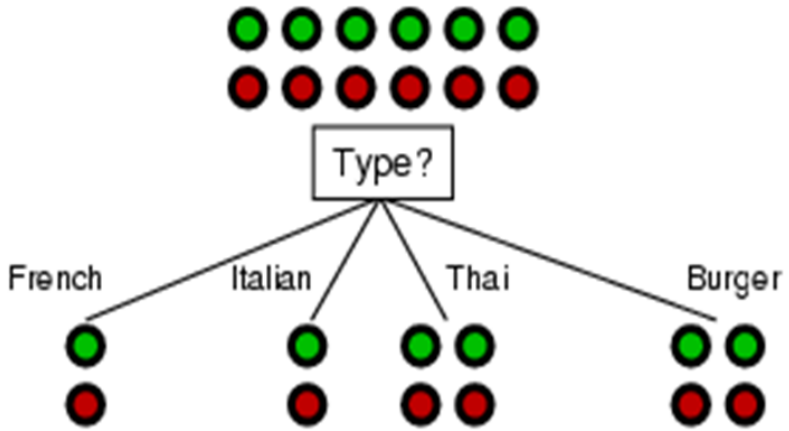
age	p_i	n_i	信息熵
<=30	2	3	0.971
31…40	4	0	0
>40	3	2	0.971

信息熵和信息增益计算的例子

- 数据集中，年龄取三个不同的值，把数据集划分为三个子集，每个子集分别包含不同的正例个数 p_i 和负例个数 n_i，如表，可以算出第四列的信息熵
- 分割后的信息熵为：5/14*0.971+4/14*0+5/14*0.971=0.694
- 故信息增益为：
  -9/14*log(9/14)-5/14*log(5/14)-0.694=0.940-0.694=0.246

熵不纯度





Patrons 比 Type 更好

- 请给出用熵不纯度判断的计算过程。

熵不纯度

信息增益率: 避免信息增益倾向选择值多的属性

- 信息增益率: $\frac{IG(N|A_i)}{\sum_j H(N_j)} = \frac{H(N)-H(N|A_i)}{\sum_j H(N_j)}$
- 用信息增益除以分割后的信息熵，分母中的 N_j 是用属性 A_i 分割数据集 N 后得到子数据集

信息增益率 + 对连续属性的处理

$\Longrightarrow$C4.5 算法

误差不纯度和 Gini 指数

误差不纯度

- $i(N) \triangleq H(N) = 1 - \max_i p_i$
- 其中 p_i 是集合中，第 i 类数据的占比

Gini 指数

- $i(N) \triangleq H(N) = \sum_{i \neq j} p_i p_j = 1 - \sum_i p_i^2$，其中 p_i 是集合中，第 i 类数据的占比

Gini 指数的“增益”

- 类似于信息增益
- 本质上还是找到一个属性，它分割数据集后，剩下的 Gini 指数最小
- 在相同的初始水平上，下降量和最后下降到的水平，二者是一致的

Gini 指数

Gini 指数的应用方法

- Gini 系数的计算过程为不停地二分节点 N 为两个子节点，故得到的是一棵“二叉树”
- 因为二分节点 N，所以对于任意一个多值属性需要一个“值域二分”过程，即把值域分成两个子集，以此来划分数据集；
- 值域二分过程非常耗时，时间复杂度是值域大小的指数函数
- 在各种值域二分后的结果上都要计算最优的基尼指数

CART 树算法

Gini 指数

age	income	student	credit_rating	buys_computer
<=30	high	no	fair	no
<=30	high	no	excellent	no
31…40	high	no	fair	yes
>40	medium	no	fair	yes
>40	low	yes	fair	yes
>40	low	yes	excellent	no
31…40	low	yes	excellent	yes
<=30	medium	no	fair	no
<=30	low	yes	fair	yes
>40	medium	yes	fair	yes
<=30	medium	yes	excellent	yes
31…40	medium	no	excellent	yes
31…40	high	yes	fair	yes
>40	medium	no	excellent	no

D

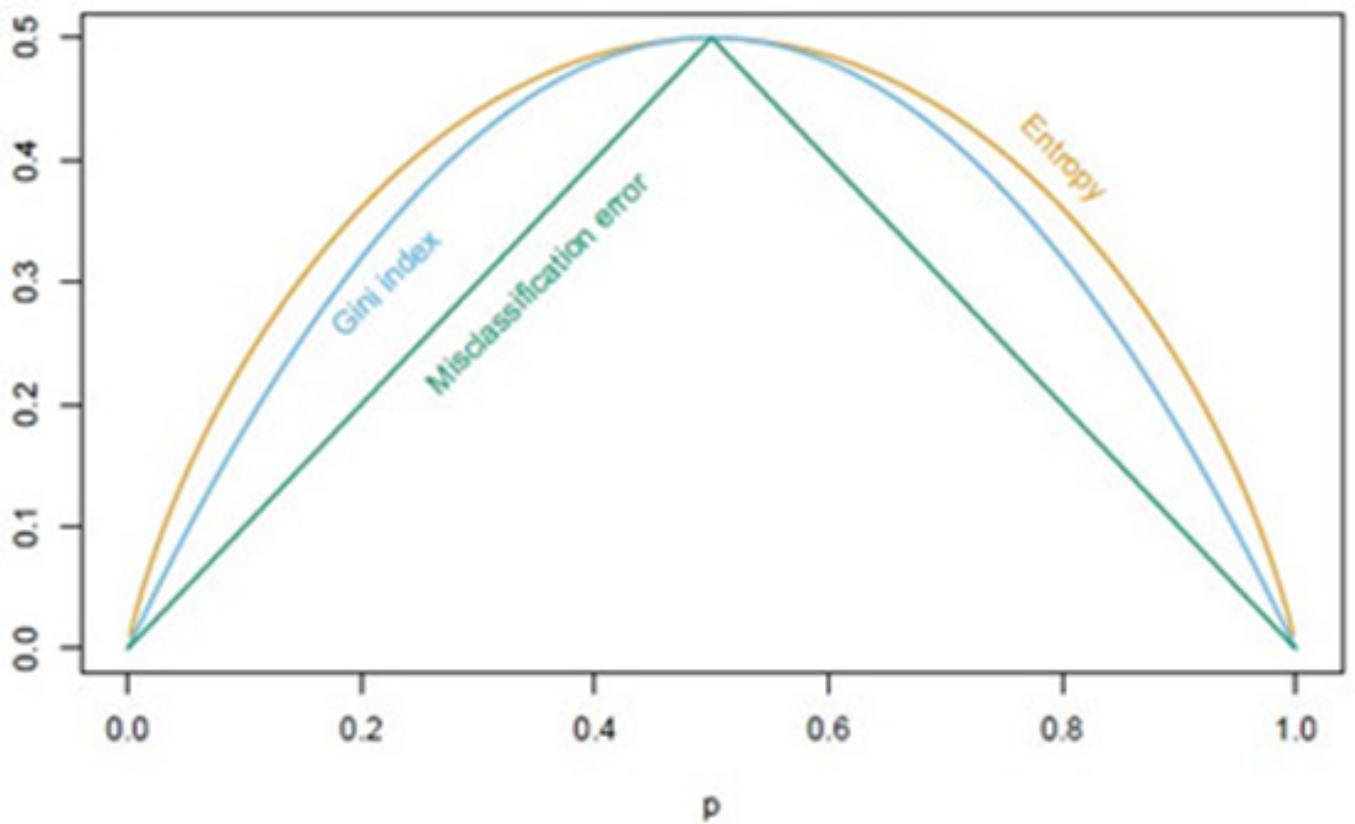
age	p_i	n_i	$H(p_i, n_i)$
<=30	2	3	0.971
31…40	4	0	0
>40	3	2	0.971

- Class P: buys_computer = "yes"
- Class N: buys_computer = "no"
- 考虑属性income的Gini系数

应用的例子

- $gini(D) = 1 - (\frac{5}{14})^2 - (\frac{9}{14})^2 = 0.459$
- income 是多值的 $\{l, m, h\}$，有三种值域划分方法
- $gini_{l,m}(D) = \frac{10}{14} gini(D_1) + \frac{4}{14} gini(D_2) = 0.443$
- 类似有 $gini_{l,h}(D) = 0.458, gini_{m,h}(D) = 0.450$
- 所以最优值域二分为:$\{l, m, h\} = \{l, m\} \cup \{h\}$，故 $gini_{income}(D) = 0.443$
- 类似可以计算 $gini_{age}(D) = 0.357, \ldots$

如何选择不纯度度量？



考量的因素

- 信息增益：偏向选择值多的属性
- 信息增益率：解决了 ID3/信息增益偏向值多属性的弊端，但是会造成不平衡分割，偏向于分割为大小相差大的各个子集
- Gini 系数：偏向于值多的属性，类多（y 值多）的时候存在困难；偏向于子集大小大致相等，纯度大致相等的分割
- 不纯度度量的选择往往不是决策树设计时最重要的因素，大量的实践表明，不同的不纯度，结果基本相当。

更多不纯度度量

其它候选项

- $CHAID$：基于卡方对属性进行独立性检验
- G-统计：类似于卡方
- 最小描述长度：用一种方法描述一棵树，树的增长用到了最小的长度，就最好，核心要描述的就是“树”以及“树的异常（错分的数据）”
- 属性的线性组合获得新的属性

关键技术二

提高决策树的性能：避免过拟合

- 分支太多，有些靠近叶的分支可能因为噪声，异常点等造成过拟合
- 泛化能力太差

解决办法

- 先剪枝：提前停止树的构建，不再分割某个节点，停下来直接构建叶节点，叶节点标号为父节点的占优类或者类分布/比例。如设置信息增益的阈值，某个节点无论如何分割，都无法获得超过阈值的信息增益，那么分割就停止。阈值如何设置？难点。
- 后剪枝：生成决策树之后，再去掉某些分支。比如：训练时保留一部分训练数据（剪枝集）不用，用来把一个完全训练好的决策树剪枝。

增强决策树的应用能力

连续型数据/属性的处理

- 离散化处理技术：离散区间

缺失值的处理

- 中心趋势度量
- 每个取值赋予一个取值概率

决策树的可扩展性增强

面对大数据集：无法一次在内存中放下所有的训练数据

- 在决策树的每个节点考察每个候选分割属性的时候都要载入一次训练数据集；涉及内外存数据交换，性能低；
- 方法一：采用 RainForest 算法来改进，载入一次数据，完成对所有候选分割属性的不纯度计算；（详细算法请自行查阅资料）
- 方法二：用随机采样获得可以全部放入内存的训练数据子集，在获得的子集上构造决策树；可重复采样，构造多棵决策树；然后用集成学习的方法综合多棵决策树的结果获得最终判决。

决策树评述

决策树：综合多个线性判决的方法

- 简单，可解释性好
- 是完备的，因为所有函数在计算机内都可以转化为 n 维真值表，任何一个 n 维真值表都可以如对应一个完全二叉树；也就是任何函数的任何 x 都可以用一颗二叉树来判决其输出；
- 描述复杂性一般用节点的数目来度量。