



人工智能导论

监督学习
(supervised learning)

郭兰哲

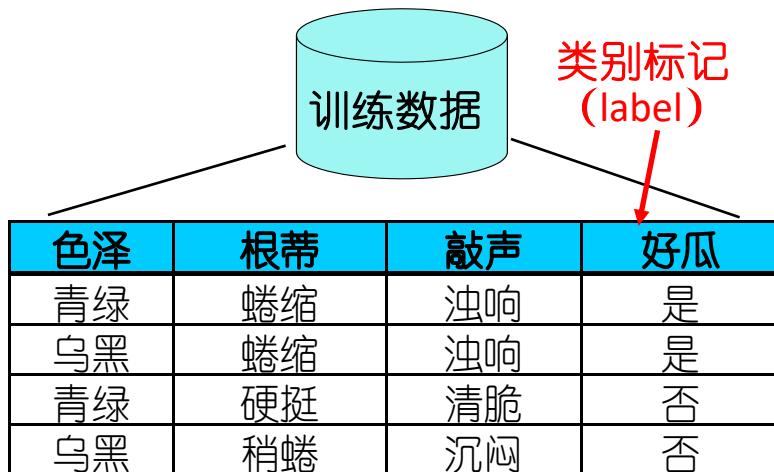
南京大学 智能科学与技术学院

Homepage: www.lamda.nju.edu.cn/guolz

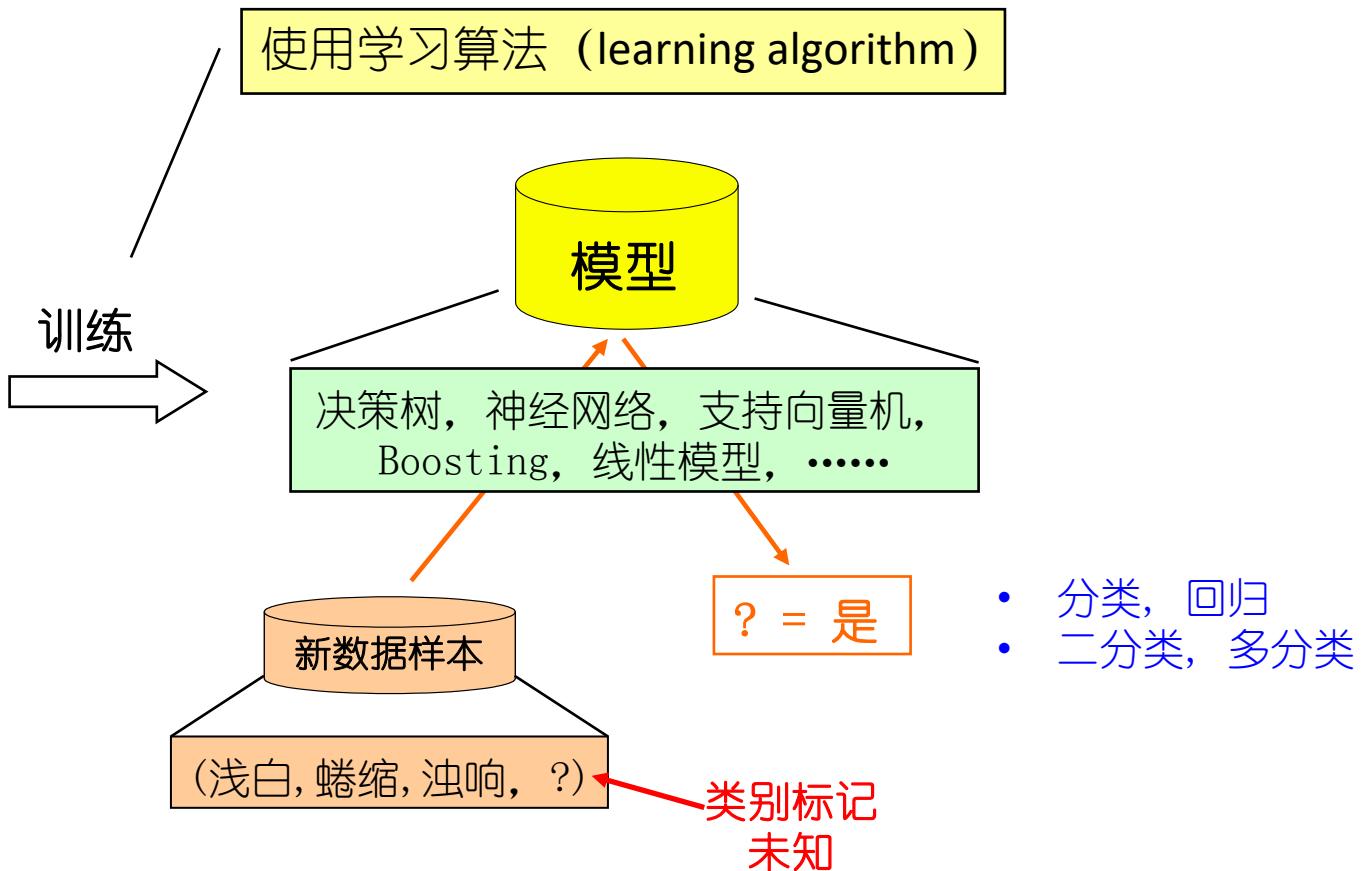
Email: guolz@nju.edu.cn

典型的机器学习过程

- 监督学习 (supervised learning)
- 无监督学习 (unsupervised learning)
- 强化学习 (reinforcement learning)



- 数据集：训练集、测试集
- 示例(instance), 样例(example), 样本(sample)
- 属性(attribute), 特征(feature)
- 属性值
- 属性空间, 样本空间, 输入空间
- 特征向量(feature vector)
- 标记空间, 输出空间



如何评估模型性能

- 机器学习的目标
 - 构建泛化能力强的模型，即能在未见的测试样本上具有优良性能的模型
- 然而，我们手上没有 `unseen instance`.....



注意：

- 保持数据分布一致性
- 测试集不能太大、不能太小（通常可设置为10%~30%）

使用Sklearn实现

```
sklearn.model_selection.train_test_split(*arrays, test_size=None,  
train_size=None, random_state=None, shuffle=True, stratify=None)
```

```
>>> import numpy as np  
>>> from sklearn.model_selection import train_test_split  
>>> X, y = np.arange(10).reshape((5, 2)), range(5)  
>>> X  
array([[0, 1],  
       [2, 3],  
       [4, 5],  
       [6, 7],  
       [8, 9]])  
>>> list(y)  
[0, 1, 2, 3, 4]
```

```
>>> X_train, X_test, y_train, y_test = train_test_split(  
...      X, y, test_size=0.33, random_state=42)  
...  
>>> X_train  
array([[4, 5],  
       [0, 1],  
       [6, 7]])  
>>> y_train  
[2, 0, 3]  
>>> X_test  
array([[2, 3],  
       [8, 9]])  
>>> y_test  
[1, 4]
```

性能度量

- 性能度量(performance measure)是衡量模型能力的评价标准，反映了任务需求,使用不同的性能度量往往会导致不同的评判结果
- 回归(regression)任务常用均方误差(mean squared error)：

$$Loss(f) = \frac{1}{n} \sum_{i=1}^n (f(x_i) - y_i)^2$$

- 分类(classification)任务常用错误率(error)或正确率(accuracy)：

$$Loss(f) = \frac{1}{n} \sum_{i=1}^n \mathbb{I}(f(x_i) \neq y_i)$$

$$Acc(f) = \frac{1}{n} \sum_{i=1}^n \mathbb{I}(f(x_i) = y_i)$$

使用Sklearn实现

Scoring	Function	Comment
Classification		
'accuracy'	metrics.accuracy_score	
'balanced_accuracy'	metrics.balanced_accuracy_score	
'top_k_accuracy'	metrics.top_k_accuracy_score	
'average_precision'	metrics.average_precision_score	
'neg_brier_score'	metrics.brier_score_loss	
'f1'	metrics.f1_score	for binary targets
'f1_micro'	metrics.f1_score	micro-averaged
'f1_macro'	metrics.f1_score	macro-averaged
'f1_weighted'	metrics.f1_score	weighted average
'f1_samples'	metrics.f1_score	by multilabel sample
'neg_log_loss'	metrics.log_loss	requires predict_proba support
'precision' etc.	metrics.precision_score	suffixes apply as with 'f1'
'recall' etc.	metrics.recall_score	suffixes apply as with 'f1'
'jaccard' etc.	metrics.jaccard_score	suffixes apply as with 'f1'
'roc_auc'	metrics.roc_auc_score	
'roc_auc_ovr'	metrics.roc_auc_score	
'roc_auc_ovo'	metrics.roc_auc_score	
'roc_auc_ovr_weighted'	metrics.roc_auc_score	
'roc_auc_ovo_weighted'	metrics.roc_auc_score	
Clustering		
'adjusted_mutual_info_score'	metrics.adjusted_mutual_info_score	
'adjusted_rand_score'	metrics.adjusted_rand_score	
'completeness_score'	metrics.completeness_score	
'fowlkes_mallows_score'	metrics.fowlkes_mallows_score	
'homogeneity_score'	metrics.homogeneity_score	
'mutual_info_score'	metrics.mutual_info_score	
'normalized_mutual_info_score'	metrics.normalized_mutual_info_score	
'rand_score'	metrics.rand_score	
'v_measure_score'	metrics.v_measure_score	
Regression		
'explained_variance'	metrics.explained_variance_score	
'max_error'	metrics.max_error	
'neg_mean_absolute_error'	metrics.mean_absolute_error	
'neg_mean_squared_error'	metrics.mean_squared_error	
'neg_root_mean_squared_error'	metrics.mean_squared_error	
'neg_mean_squared_log_error'	metrics.mean_squared_log_error	
'neg_median_absolute_error'	metrics.median_absolute_error	
'r2'	metrics.r2_score	
'neg_mean_poisson_deviance'	metrics.mean_poisson_deviance	
'neg_mean_gamma_deviance'	metrics.mean_gamma_deviance	
'neg_mean_absolute_percentage_error'	metrics.mean_absolute_percentage_error	
'd2_absolute_error_score'	metrics.d2_absolute_error_score	
'd2_pinball_score'	metrics.d2_pinball_score	
'd2_tweedie_score'	metrics.d2_tweedie_score	

- 正确率(accuracy)：

```
>>> import numpy as np
>>> from sklearn.metrics import accuracy_score
>>> y_pred = [0, 2, 1, 3]
>>> y_true = [0, 1, 2, 3]
>>> accuracy_score(y_true, y_pred)
0.5
```

- 均方误差(mean squared error)：

```
>>> from sklearn.metrics import mean_squared_error
>>> y_true = [3, -0.5, 2, 7]
>>> y_pred = [2.5, 0.0, 2, 8]
>>> mean_squared_error(y_true, y_pred)
0.375
>>> y_true = [[0.5, 1], [-1, 1], [7, -6]]
>>> y_pred = [[0, 2], [-1, 2], [8, -5]]
>>> mean_squared_error(y_true, y_pred)
0.7083...
```

大纲

- 近邻算法
- 决策树
- 贝叶斯分类器
- 线性回归
- 对数几率回归
- 支持向量机

大纲

□ 近邻算法

□ 决策树

□ 贝叶斯分类器

□ 线性回归

□ 对数几率回归

□ 支持向量机

考虑一个二分类问题

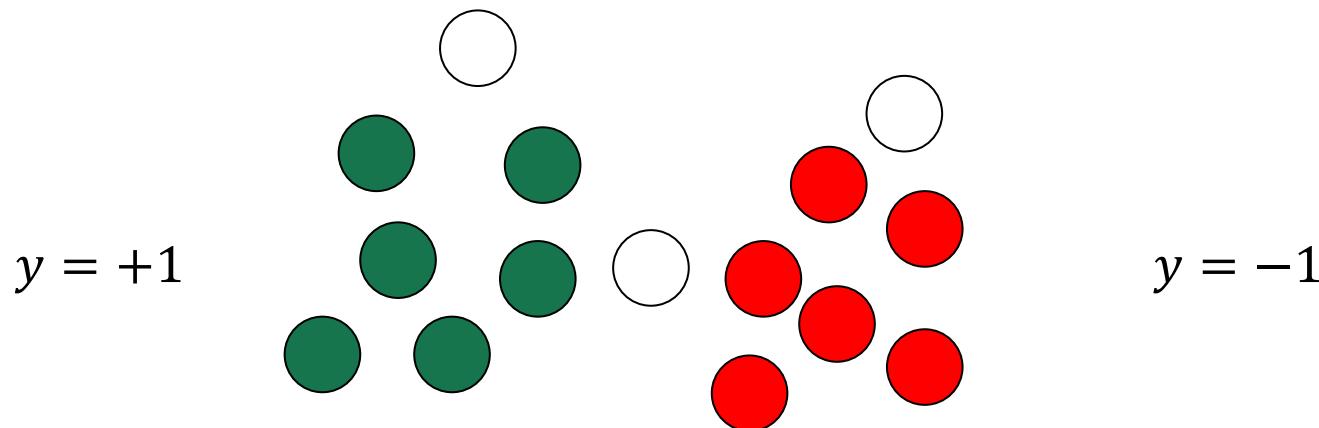
- 给定训练数据集 $D = \{(x_1, y_1), (x_2, y_2), \dots, (x_n, y_n)\}$
- 学习模型 $f: \mathcal{X} \rightarrow \mathcal{Y}, \mathcal{Y} = \{-1, +1\}$

基本思路：
近朱者赤，近墨者黑

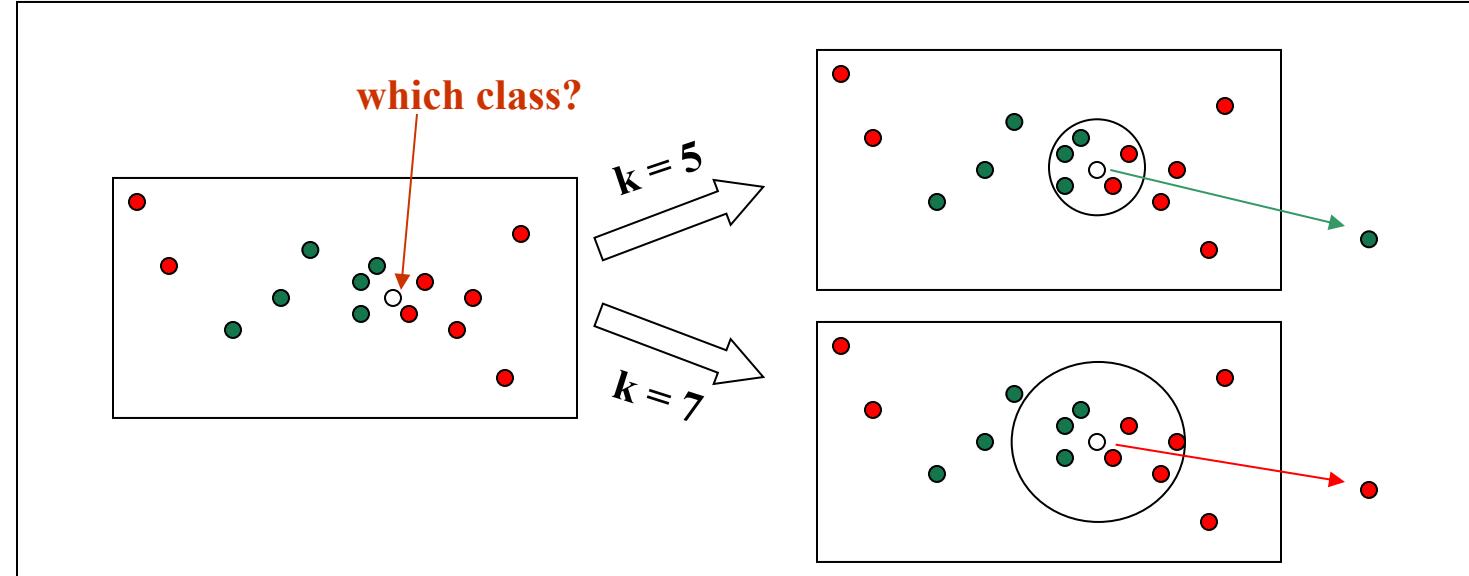
绿色代表正类训练样本，红色代表负类训练样本

懒惰学习(lazy learning)的代表

白色测试样本应该被预测为哪个类别？



k 近邻学习器

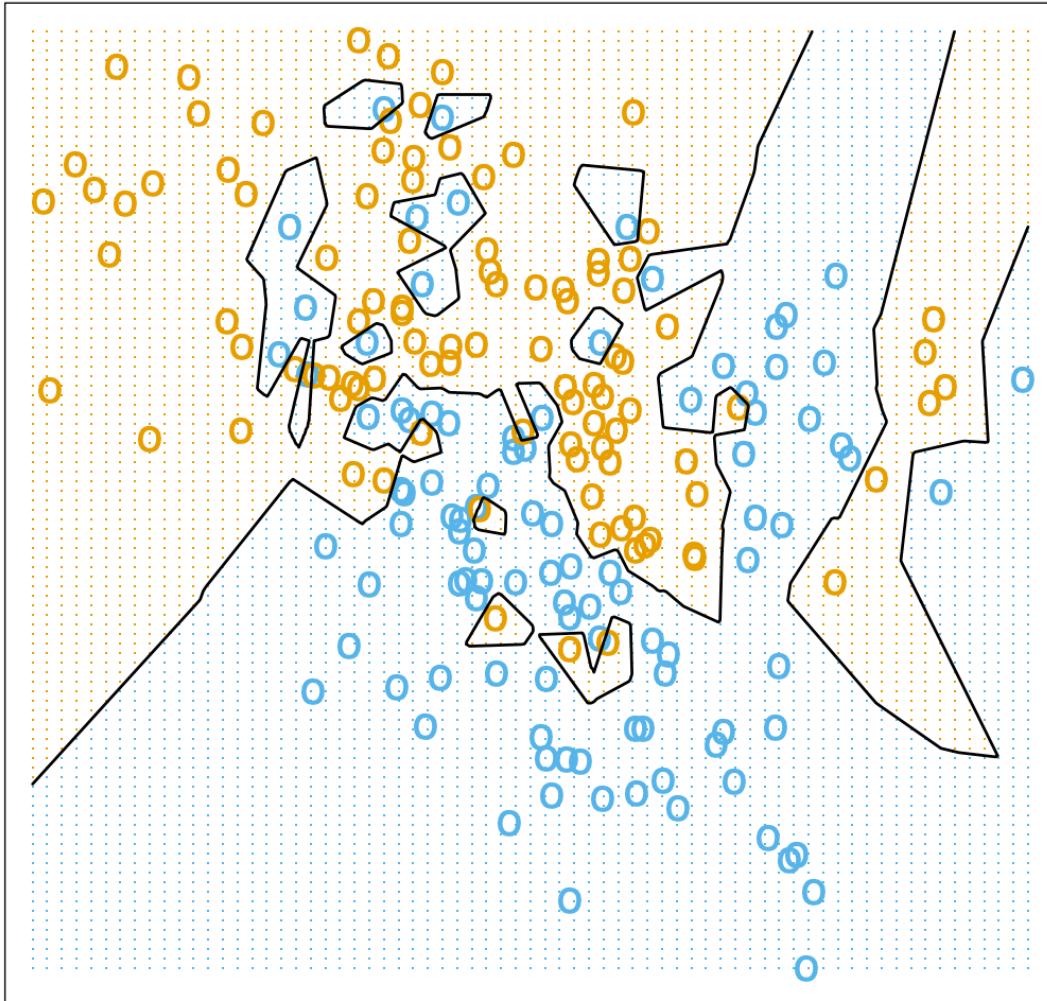


找到最近的 k 个样本，投票或者平均

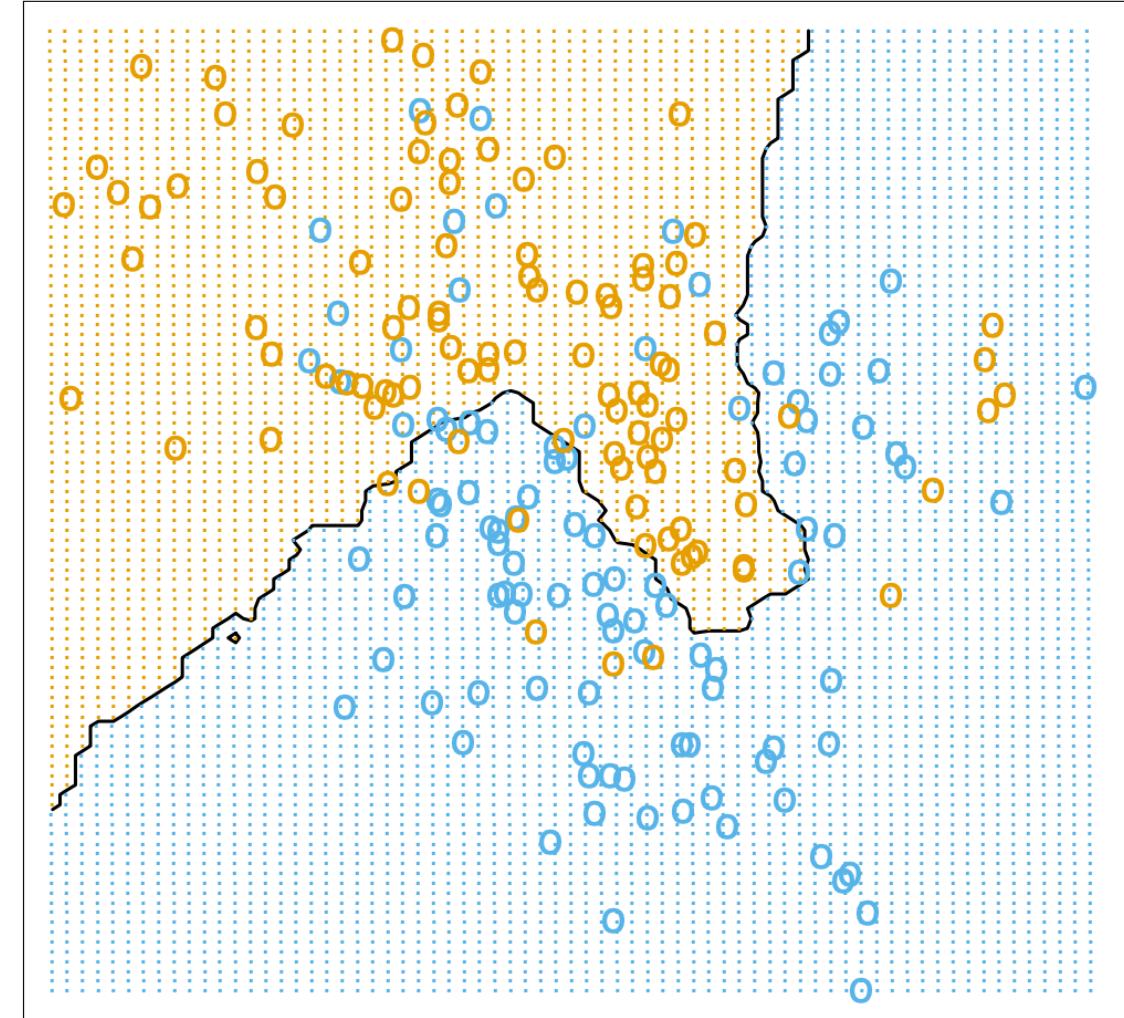
关键问题： k 值选取； 距离计算

Example Results

1-Nearest Neighbor Classifier



15-Nearest Neighbor Classifier



k 值的选择

k 值的选择会对 k 近邻学习器的结果产生重要影响

- 如果选择较小的 k ，相当于用较小的邻域中的训练样本进行预测，训练误差会减小，泛化误差会增大，即 k 的减小意味着整体模型变得更复杂，容易发生过拟合
- 如果选择较大的 k ，相当于用较大的邻域中的训练样本进行预测，会让模型变得简单，一定程度上可以减小泛化误差
- 如果 $k=n$?
 - 无论输入样本是什么，都将预测它属于在训练样本中最多的类，模型过于简单
 - 在应用中， k 值一般取一个较小的数值，然后通过对性能进行评估来选取最优的 k 值

距离度量

距离度量 (distance metric) 需满足的基本性质：

非负性: $\text{dist}(x_i, x_j) \geq 0$;

同一性: $\text{dist}(x_i, x_j) = 0$ 当且仅当 $x_i = x_j$;

对称性: $\text{dist}(x_i, x_j) = \text{dist}(x_j, x_i)$;

直递性: $\text{dist}(x_i, x_j) \leq \text{dist}(x_i, x_k) + \text{dist}(x_k, x_j)$.

常用距离形式：闵可夫斯基距离 (Minkowski distance)

假设输入样本为 d 维向量: $x = (x_1, x_2, \dots, x_d)$

$$\text{dist}_{mk}(x, x') = \left(\sum_{i=1}^d |x_i - x'_i|^p \right)^{\frac{1}{p}}$$

$p = 2$: 欧氏距离(Euclidean distance)

$p = 1$: 曼哈顿距离(Manhattan distance)

使用Sklearn实现

```
class sklearn.neighbors.KNeighborsClassifier(n_neighbors=5, *, weight  
s='uniform', algorithm='auto', leaf_size=30, p=2, metric='minkowski',  
metric_params=None, n_jobs=None)
```

```
>>> X = [[0], [1], [2], [3]]  
>>> y = [0, 0, 1, 1]  
>>> from sklearn.neighbors import KNeighborsClassifier  
>>> neigh = KNeighborsClassifier(n_neighbors=3)  
>>> neigh.fit(X, y)  
>>> print(neigh.predict([[1.1]]))  
[0]
```

k 近邻学习器

□ 思考：

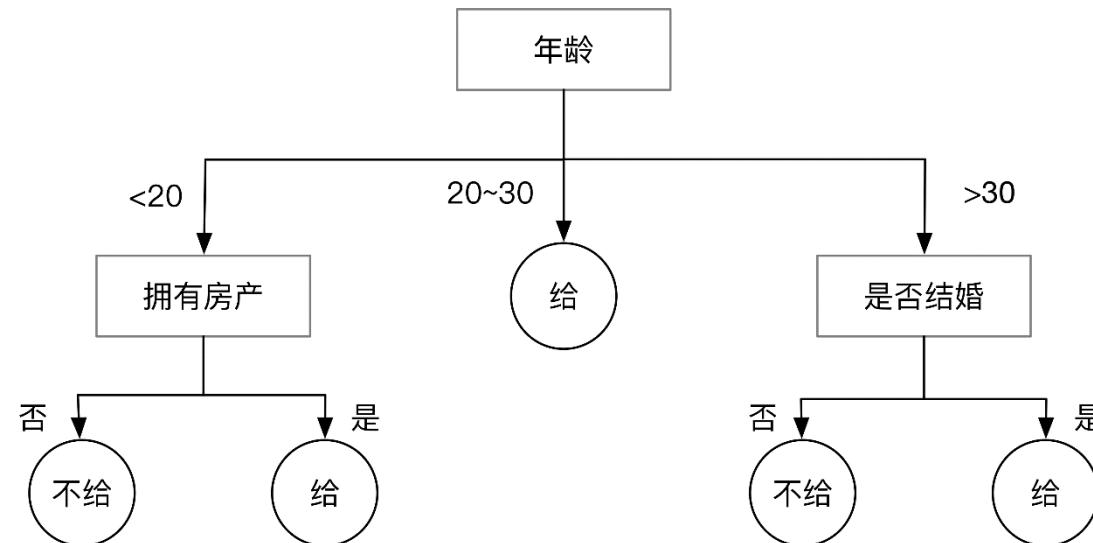
- 如何计算离散属性之间的距离？
- 测试时的复杂度是多少？能否优化？
- k 近邻学习器的性能和样本特征维度之间的关系？

大纲

- 近邻算法
- **决策树**
- 贝叶斯分类器
- 线性回归
- 对数几率回归
- 支持向量机

决策树

- 决策树是一种通过树形结构来进行分类的方法
- 每个非叶子节点表示对样本在某个属性上的一个判断，根据判断的不同结果产生不同的分支，每个叶子节点代表一种分类结果



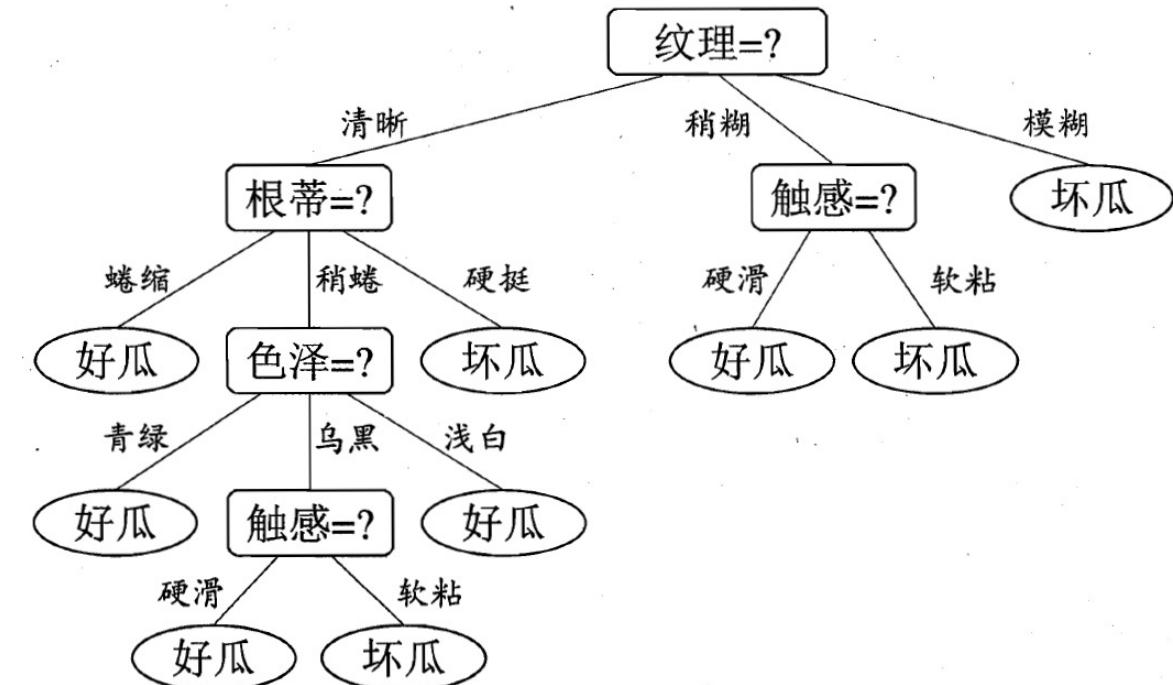
银行判断是否放贷的决策树

决策树

西瓜数据集

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	软粘	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

基于西瓜数据集学习的决策树



关键问题：如何选择划分属性

决策树

- 随着划分的不断进行，我们希望决策树分支结点所包含的样本集尽可能属于同一类别，即节点的“纯度”越高约好
- 信息熵 (entropy) 是度量样本集合“纯度”最常用的一种指标，假定当前样本集合 D 中第 k 类样本所占的比例为 p_k ，则 D 的信息熵定义为

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

计算信息熵时约定：若
 $p = 0$ ，则 $p \log_2 p = 0$.

$\text{Ent}(D)$ 的最小值为 0，
最大值为 $\log_2 |Y|$.

$\text{Ent}(D)$ 的值越小，则 D 的纯度越高

决策树

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	软粘	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

根节点（未划分之前）的信息熵为？

数据集包含17个样本，类别为2，正类数量为8，负类数量为9，所以，

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

如果采用“色泽”作为划分属性，可以得到几个分支节点？每个分支节点的熵是多少？

D_1 (色泽 = 青绿), D_2 (色泽 = 浅白), D_3 (色泽 = 乌黑)

决策树

D_1 (色泽 = 青绿)

正类3/6, 负类3/6:

$$Ent(D_1) = -\left(\frac{3}{6}\log_2 \frac{3}{6} + \frac{3}{6}\log_2 \frac{3}{6}\right) = 1.0$$

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	软粘	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

决策树

D_2 (色泽 = 浅白)

正类1/5, 负类4/5

$$Ent(D_2) = -\left(\frac{1}{5} \log_2 \frac{1}{5} + \frac{4}{5} \log_2 \frac{4}{5}\right) = 0.722$$

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	软粘	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

决策树

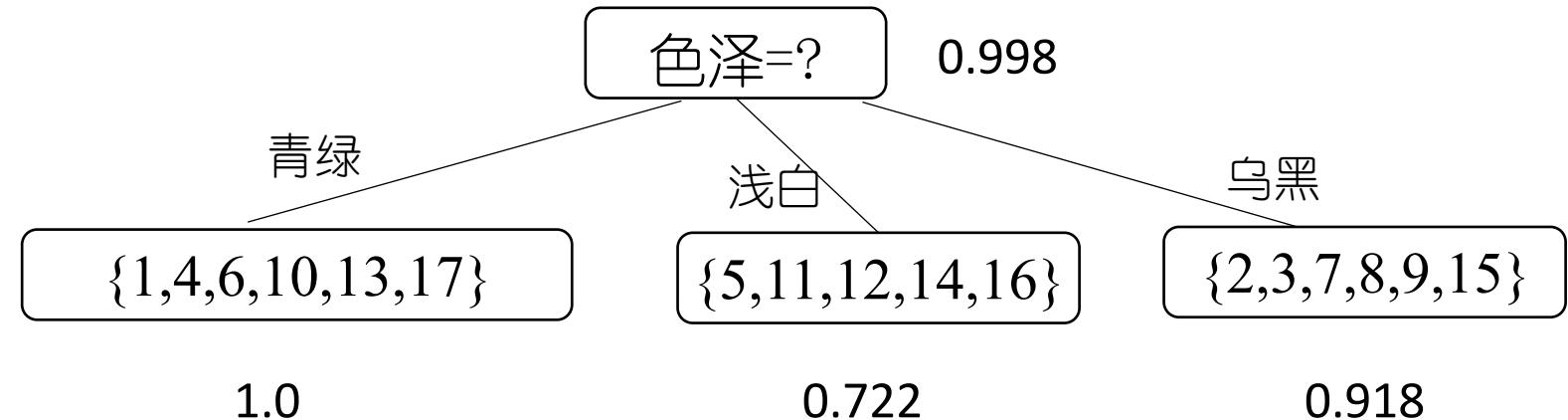
D_3 (色泽 = 乌黑)

正例4/6, 反例2/6

$$Ent(D_3) = -\left(\frac{4}{6} \log_2 \frac{4}{6} + \frac{2}{6} \log_2 \frac{2}{6}\right) = 0.918$$

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	软粘	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

决策树



如何计算使用“色泽”作为划分节点带来的好处？

简单的想法：比较划分前和划分后的信息熵

信息增益

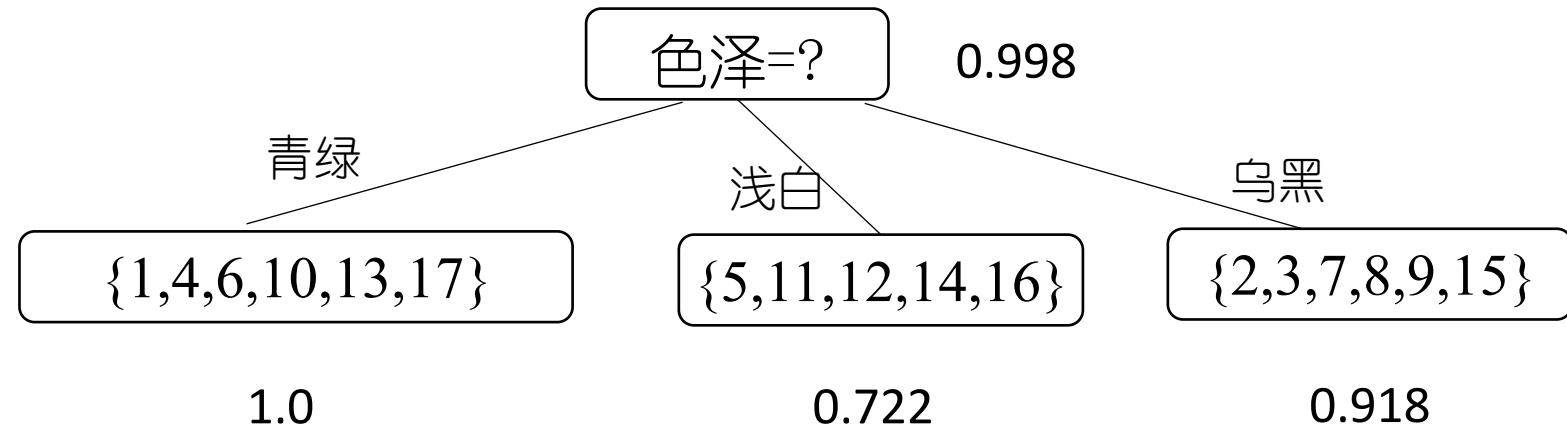
以属性 a 对 数据集 D 进行划分所获得的信息增益为：

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

↓ ↓
划分前的信息熵 第 v 个分支的权重，
 样本越多越重要

↓
划分后的信息熵

信息增益



$$\begin{aligned}\text{Gain}(D, \text{色泽}) &= \text{Ent}(D) - \sum_{v=1}^3 \frac{|D^v|}{|D|} \text{Ent}(D^v) \\ &= 0.998 - \left(\frac{6}{17} \times 1.000 + \frac{6}{17} \times 0.918 + \frac{5}{17} \times 0.722 \right) = 0.109\end{aligned}$$

决策树

类似的，其他属性的信息增益为

$$\text{Gain}(D, \text{根蒂}) = 0.143$$

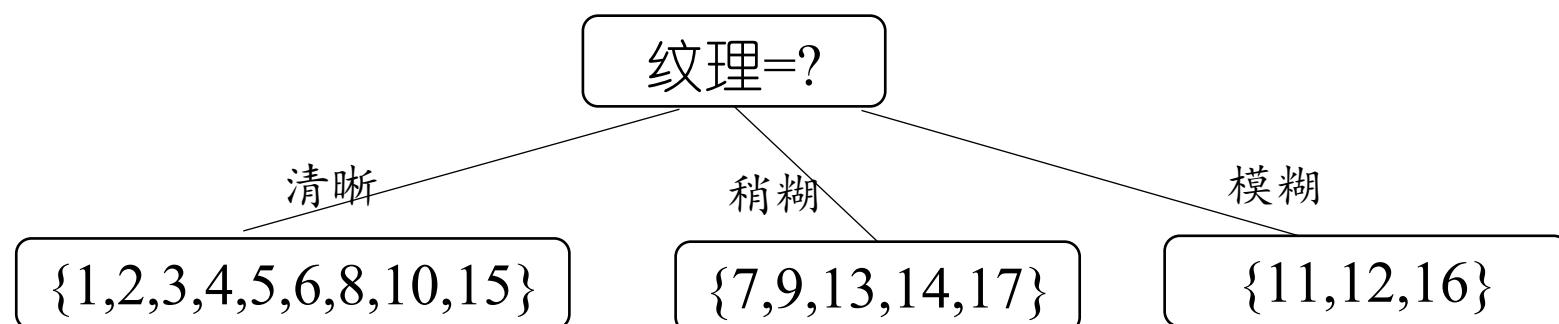
$$\text{Gain}(D, \text{敲声}) = 0.141$$

$$\text{Gain}(D, \text{纹理}) = 0.381$$

$$\text{Gain}(D, \text{脐部}) = 0.289$$

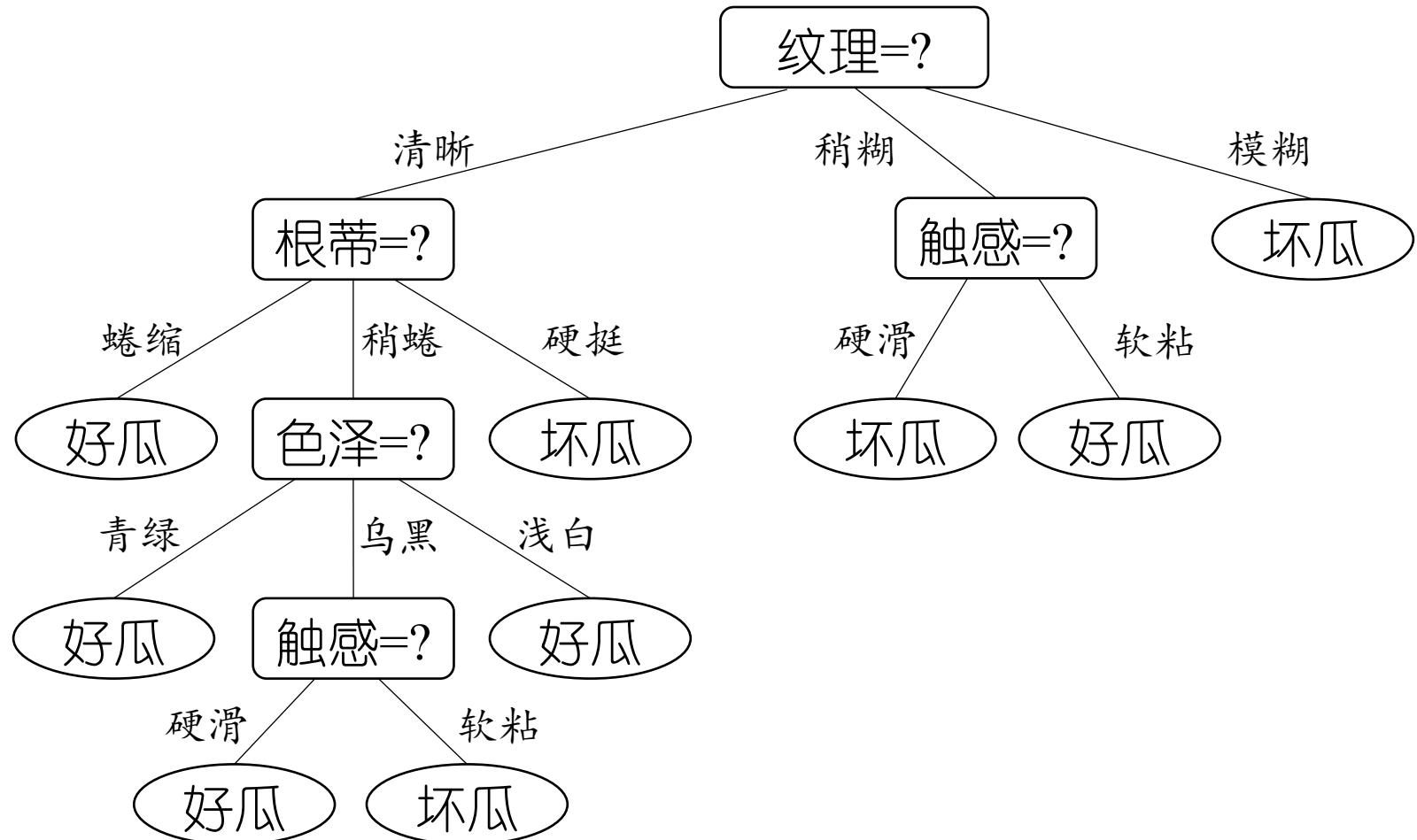
$$\text{Gain}(D, \text{触感}) = 0.006$$

属性“纹理”的信息增益最大，被选为划分属性



决策树

对每个分支结点做进一步划分，最终得到决策树



使用Sklearn实现

```
class sklearn.tree.DecisionTreeClassifier(*, criterion='gini',
    splitter='best', max_depth=None, min_samples_split=2, min_samples_leaf=1,
    min_weight_fraction_leaf=0.0, max_features=None,
    random_state=None, max_leaf_nodes=None, min_impurity_decrease=0.0,
    class_weight=None, ccp_alpha=0.0)
```

```
>>> from sklearn import tree
>>> X = [[0, 0], [1, 1]]
>>> Y = [0, 1]
>>> clf = tree.DecisionTreeClassifier()
>>> clf = clf.fit(X, Y)
>>> clf.predict([[2., 2.]])
array([1])
```

使用Sklearn实现

```
In [2]: import pandas as pd  
from sklearn import tree  
import graphviz  
from sklearn.preprocessing import LabelEncoder
```

```
D = [  
    {'色泽': '青绿', '根蒂': '蜷缩', '敲声': '浊响', '纹理': '清晰', '脐部': '凹陷', '触感': '硬滑', '好瓜': '是'},  
    {'色泽': '乌黑', '根蒂': '蜷缩', '敲声': '沉闷', '纹理': '清晰', '脐部': '凹陷', '触感': '硬滑', '好瓜': '是'},  
    {'色泽': '乌黑', '根蒂': '蜷缩', '敲声': '浊响', '纹理': '清晰', '脐部': '凹陷', '触感': '硬滑', '好瓜': '是'},  
    {'色泽': '青绿', '根蒂': '蜷缩', '敲声': '沉闷', '纹理': '清晰', '脐部': '凹陷', '触感': '硬滑', '好瓜': '是'},  
    {'色泽': '浅白', '根蒂': '蜷缩', '敲声': '浊响', '纹理': '清晰', '脐部': '凹陷', '触感': '硬滑', '好瓜': '是'},  
    {'色泽': '青绿', '根蒂': '稍蜷', '敲声': '浊响', '纹理': '清晰', '脐部': '稍凹', '触感': '软粘', '好瓜': '是'},  
    {'色泽': '乌黑', '根蒂': '稍蜷', '敲声': '浊响', '纹理': '稍糊', '脐部': '稍凹', '触感': '软粘', '好瓜': '是'},  
    {'色泽': '乌黑', '根蒂': '稍蜷', '敲声': '浊响', '纹理': '清晰', '脐部': '稍凹', '触感': '硬滑', '好瓜': '是'},  
    {'色泽': '乌黑', '根蒂': '稍蜷', '敲声': '沉闷', '纹理': '稍糊', '脐部': '稍凹', '触感': '硬滑', '好瓜': '否'},  
    {'色泽': '青绿', '根蒂': '硬挺', '敲声': '清脆', '纹理': '清晰', '脐部': '平坦', '触感': '软粘', '好瓜': '否'},  
    {'色泽': '浅白', '根蒂': '硬挺', '敲声': '清脆', '纹理': '模糊', '脐部': '平坦', '触感': '硬滑', '好瓜': '否'},  
    {'色泽': '浅白', '根蒂': '蜷缩', '敲声': '浊响', '纹理': '模糊', '脐部': '平坦', '触感': '软粘', '好瓜': '否'},  
    {'色泽': '青绿', '根蒂': '稍蜷', '敲声': '浊响', '纹理': '稍糊', '脐部': '凹陷', '触感': '硬滑', '好瓜': '否'},  
    {'色泽': '浅白', '根蒂': '稍蜷', '敲声': '沉闷', '纹理': '稍糊', '脐部': '凹陷', '触感': '硬滑', '好瓜': '否'},  
    {'色泽': '乌黑', '根蒂': '稍蜷', '敲声': '浊响', '纹理': '清晰', '脐部': '稍凹', '触感': '软粘', '好瓜': '否'},  
    {'色泽': '浅白', '根蒂': '蜷缩', '敲声': '浊响', '纹理': '模糊', '脐部': '平坦', '触感': '硬滑', '好瓜': '否'},  
    {'色泽': '青绿', '根蒂': '蜷缩', '敲声': '沉闷', '纹理': '稍糊', '脐部': '稍凹', '触感': '硬滑', '好瓜': '否'}  
]
```

```
In [3]: data = pd.DataFrame(D)  
data
```

Out[3]:

	色泽	根蒂	敲声	纹理	脐部	触感	好瓜
0	青绿	蜷缩	浊响	清晰	凹陷	硬滑	是
1	乌黑	蜷缩	沉闷	清晰	凹陷	硬滑	是
2	乌黑	蜷缩	浊响	清晰	凹陷	硬滑	是
3	青绿	蜷缩	沉闷	清晰	凹陷	硬滑	是
4	浅白	蜷缩	浊响	清晰	凹陷	硬滑	是
5	青绿	稍蜷	浊响	清晰	稍凹	软粘	是
6	乌黑	稍蜷	浊响	稍糊	稍凹	软粘	是
7	乌黑	稍蜷	浊响	清晰	稍凹	硬滑	是
8	乌黑	稍蜷	沉闷	稍糊	稍凹	硬滑	否
9	青绿	硬挺	清脆	清晰	平坦	软粘	否
10	浅白	硬挺	清脆	模糊	平坦	硬滑	否
11	浅白	蜷缩	浊响	模糊	平坦	软粘	否
12	青绿	稍蜷	浊响	稍糊	凹陷	硬滑	否
13	浅白	稍蜷	沉闷	稍糊	凹陷	硬滑	否
14	乌黑	稍蜷	浊响	清晰	稍凹	软粘	否
15	浅白	蜷缩	浊响	模糊	平坦	硬滑	否
16	青绿	蜷缩	沉闷	稍糊	稍凹	硬滑	否

使用Sklearn实现

```
In [4]: Encoder = LabelEncoder()
for i in range(data.shape[1]):
    data.iloc[:,i] = Encoder.fit_transform(data.iloc[:,i])
```

```
In [5]: data
```

```
Out[5]:
```

	色泽	根蒂	敲声	纹理	脐部	触感	好瓜
0	2	2	1	1	0	0	1
1	0	2	0	1	0	0	1
2	0	2	1	1	0	0	1
3	2	2	0	1	0	0	1
4	1	2	1	1	0	0	1
5	2	1	1	1	2	1	1
6	0	1	1	2	2	1	1
7	0	1	1	1	2	0	1
8	0	1	0	2	2	0	0
9	2	0	2	1	1	1	0
10	1	0	2	0	1	0	0
11	1	2	1	0	1	1	0
12	2	1	1	2	0	0	0
13	1	1	0	2	0	0	0
14	0	1	1	1	2	1	0
15	1	2	1	0	1	0	0
16	2	2	0	2	2	0	0

```
In [6]: clf = tree.DecisionTreeClassifier()
clf.fit(data.iloc[:, :-1], data.iloc[:, -1])
```

```
Out[6]:
```

```
▼ DecisionTreeClassifier
DecisionTreeClassifier()
```

```
In [7]: clf.predict([[1,1,0,1,1,0]])
```

```
Out[7]: array([1])
```

决策树

□ 思考：

- 决策树的生成是一个递归过程，什么情形会导致递归返回？
- 测试时的复杂度是多少？能否优化？
- 如何处理连续值？

大纲

- 近邻算法
- 决策树
- 贝叶斯分类器
- 线性回归
- 对数几率回归
- 支持向量机

期望风险最小化

- 给定 N 个类别，令 λ_{ij} 代表将第 j 类样本误分类为第 i 类所产生的损失，则基于后验概率 $P(c|x)$ 将样本 x 分到第 i 类的风险为：

$$R(c_i|x) = \sum_{j=1}^N \lambda_{ij} P(c_j|x)$$

- 机器学习的目标：寻找模型 $f: \mathcal{X} \rightarrow \mathcal{Y}$ 以最小化总体风险

$$R(f) = \mathbb{E}_x[R(f(x)|x)]$$

- 为最小化总体风险，只需在每个样本上选择使条件风险 $R(c|x)$ 最小的类别标记

$$f(x) = \arg \min_{c \in \mathcal{Y}} R(c|x)$$

后验概率最大化

- 若目标是最小化分类错误率，则 λ_{ij} 可写为

$$\lambda_{ij} = \begin{cases} 0, & \text{if } i = j \\ 1, & \text{otherwise} \end{cases}$$

- 此时，风险 $R(c|x)$ 为：

$$R(c|x) = 1 - P(c|x)$$

- 于是可以得到，最优分类器为：

$$f(x) = \arg \max_{c \in \mathcal{Y}} P(c|x)$$

判别式 vs. 生成式

$P(c|x)$ 在现实中通常难以直接获得

从概率框架来理解，机器学习所要实现的是基于有限的训练样本集，
尽可能准确地估计出后验概率 $P(c|x)$

两种基本策略：

判别式 (discriminative) 模型

思路：直接对 $P(c|x)$ 建模

代表：
决策树
神经网络

...

生成式 (generative) 模型

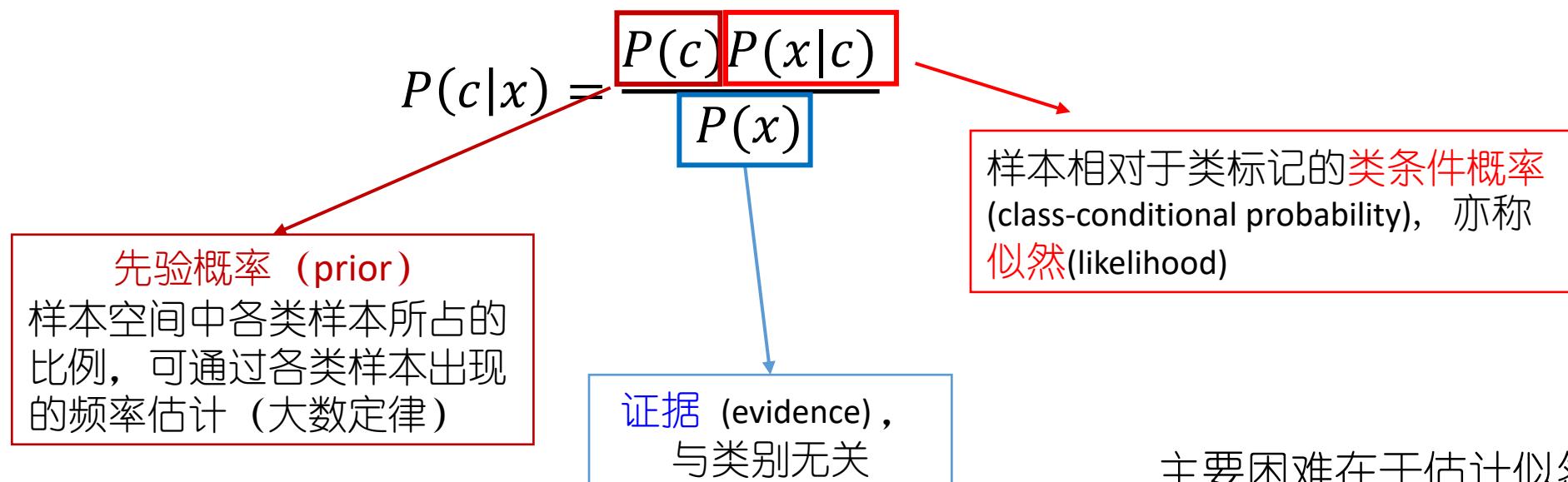
思路：先对联合概率分布 $P(c, x)$ 建模，再获得 $P(c|x)$

代表：贝叶斯分类器

贝叶斯定理

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

根据贝叶斯定理，有



主要困难在于估计似然
 $P(x|c)$

例子

- 例：给定下列训练数据，表中 $X^{(1)}$, $X^{(2)}$ 为特征，取值集合分别为 $A_1 = \{1, 2, 3\}$, $A_2 = \{S, M, L\}$, $Y \in \mathcal{Y} = \{1, -1\}$ 为标记

	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
$X^{(1)}$	1	1	1	1	1	2	2	2	2	2	3	3	3	3	3
$X^{(2)}$	S	M	M	S	S	S	M	M	L	L	M	M	L	L	L
Y	-1	-1	1	1	-1	-1	-1	1	1	1	1	1	1	1	-1

如何估计 $P(c), P(x|c)$?

$P(x|c)$ 是所有属性上的联合概率，基于有限计算样本在计算上会遭遇组合爆炸问题，在数据上会遭遇样本稀疏问题

朴素贝叶斯分类器

- 朴素贝叶斯分类器采用了“属性独立假设”：
- 对已知类别，假设所有属性相互独立，即，每个属性独立地对分类结果发生影响

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

- 朴素贝叶斯分类器的表达式

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

例子

□ 例：给定下列训练数据，表中 $X^{(1)}$, $X^{(2)}$ 为特征，取值集合分别为 $A_1 = \{1, 2, 3\}$, $A_2 = \{S, M, L\}$, $Y \in \mathcal{Y} = \{1, -1\}$ 为标记

	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
$X^{(1)}$	1	1	1	1	1	2	2	2	2	2	3	3	3	3	3
$X^{(2)}$	S	M	M	S	S	S	M	M	L	L	M	M	L	L	L
Y	-1	-1	1	1	-1	-1	-1	1	1	1	1	1	1	1	-1

$$P(Y = 1) = \frac{9}{15} \quad P(Y = -1) = \frac{6}{15}$$

$$P(X^{(1)} = 1|Y = 1) = \frac{2}{9} \quad P(X^{(1)} = 2|Y = 1) = \frac{3}{9} \quad P(X^{(1)} = 3|Y = 1) = \frac{4}{9}$$

$$P(X^{(2)} = S|Y = 1) = \frac{1}{9} \quad P(X^{(2)} = M|Y = 1) = \frac{4}{9} \quad P(X^{(2)} = L|Y = 1) = \frac{4}{9}$$

$$P(X^{(1)} = 1|Y = -1) = \frac{3}{6} \quad P(X^{(1)} = 2|Y = -1) = \frac{2}{6} \quad P(X^{(1)} = 3|Y = -1) = \frac{1}{6}$$

$$P(X^{(2)} = S|Y = -1) = \frac{3}{6} \quad P(X^{(2)} = M|Y = -1) = \frac{2}{6} \quad P(X^{(2)} = L|Y = -1) = \frac{1}{6}$$

例子

- 例：给定下列训练数据，表中 $X^{(1)}$, $X^{(2)}$ 为特征，取值集合分别为 $A_1 = \{1, 2, 3\}$, $A_2 = \{S, M, L\}$, $Y \in \mathcal{Y} = \{1, -1\}$ 为标记

	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
$X^{(1)}$	1	1	1	1	1	2	2	2	2	2	3	3	3	3	3
$X^{(2)}$	S	M	M	S	S	S	M	M	L	L	M	M	L	L	L
Y	-1	-1	1	1	-1	-1	-1	1	1	1	1	1	1	1	-1

对于给定的测试样本 $x = (2, S)^\top$ 进行分类： $f_{nb}(x) = \arg \max_{c \in \mathcal{Y}} P(c) \prod_{i=1}^d P(x_i | c)$

$$P(Y = 1)P(X^{(1)} = 2 | Y = 1)P(X^{(2)} = S | Y = 1) = \frac{9}{15} * \frac{3}{9} * \frac{1}{3} = \frac{1}{45}$$

$$P(Y = -1)P(X^{(1)} = 2 | Y = -1)P(X^{(2)} = S | Y = -1) = \frac{6}{15} * \frac{2}{6} * \frac{3}{6} = \frac{1}{15}$$

预测结果为-1

使用Sklearn实现

```
In [1]: from sklearn.naive_bayes import CategoricalNB
```

```
In [2]: X=[[1,1],[1,2],[1,2],[1,1],[1,1],[2,1],[2,2],[2,2],[2,3],[2,3],[3,3],[3,2],[3,2],[3,3],[3,3]]  
Y = [-1, -1, 1, 1, -1, -1, 1, 1, 1, 1, 1, 1, 1, 1, 1, -1]
```

```
In [3]: f_nb = CategoricalNB()  
f_nb.fit(X,Y)
```

```
Out[3]:
```

```
  ▾ CategoricalNB  
CategoricalNB()
```

```
In [4]: x = [[2,1]]
```

```
In [5]: f_nb.predict(x)
```

```
Out[5]: array([-1])
```

贝叶斯分类器

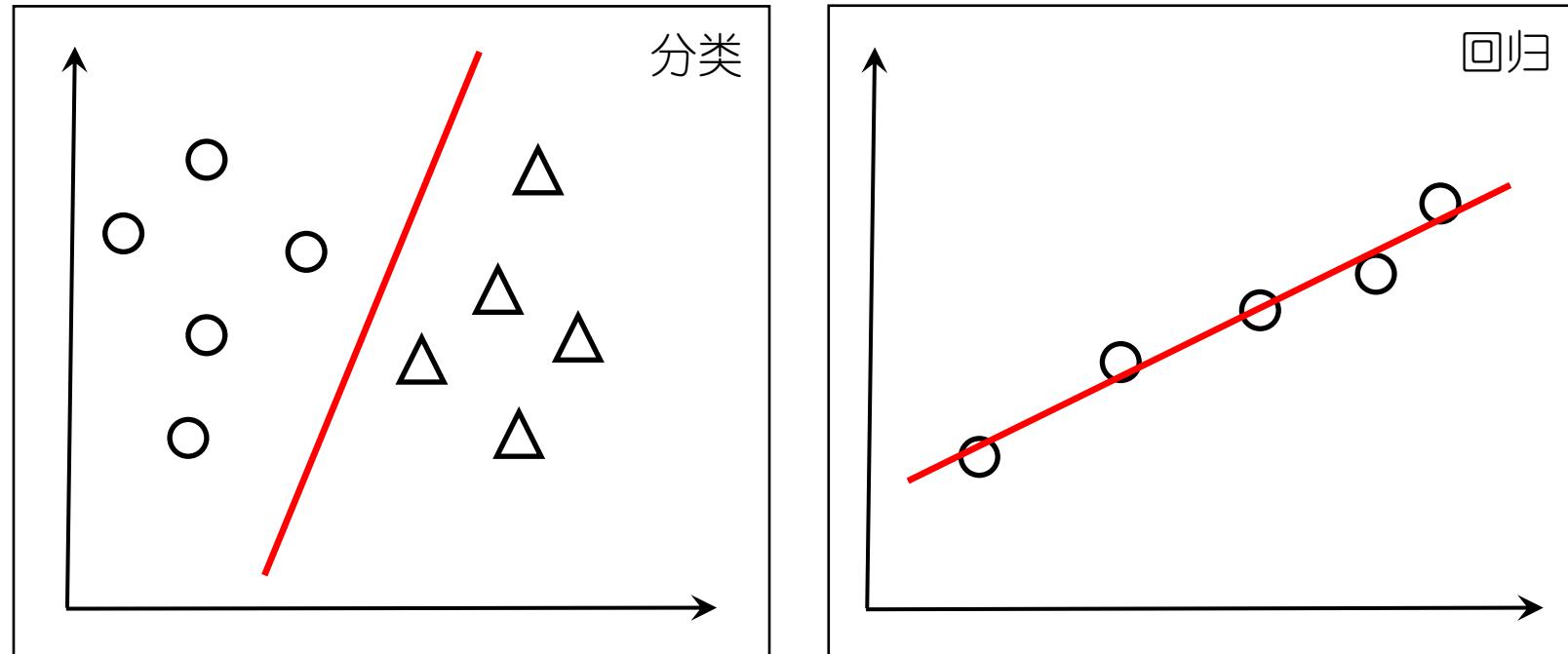
□ 思考：

- 如果某个属性值在训练集中没有与某个类同时出现过，朴素贝叶斯分类器会出现什么问题？

大纲

- 近邻算法
- 决策树
- 贝叶斯分类器
- 线性回归**
- 对数几率回归
- 支持向量机

线性模型



线性回归(linear regression)

- 在现实生活中，往往需要分析若干变量之间的关系，如碳排放量与气候变暖之间的关系、某一商品广告投入量与该商品销售量之间的关系等
- 这种分析不同变量之间存在关系的研究叫[回归分析](#)，刻画不同变量之间关系的模型被称为[回归模型](#)，如果这个模型是线性的，则称为[线性回归模型](#)

$$y = 33.73(\text{英寸}) + 0.516x$$

y : 子女平均身高 x : 父母平均身高

- 父母平均身高每增加一个单位，其成年子女平均身高只增加 0.516 个单位，它反映了这种“衰退 (regression)” 效应（“回归”到正常人平均身高）
- 虽然 x 和 y 之间并不总是具有“衰退”（回归）关系，但是“线性回归”这一名称就保留了下来了

一元线性回归(linear regression)

□ 例子：下表给出了芒提兹尼欧（Montesinho）地区发生森林火灾的部分历史数据，表中列举了每次发生森林火灾时的气温温度取值 x 和受到火灾影响的森林面积 y

气温温度 x	5.1	8.2	11.5	13.9	15.1	16.2	19.6	23.3
火灾影响面积 y	2.14	4.62	8.24	11.24	13.99	16.33	19.23	28.74

如何对气温温度与火灾所影响的森林面积之间关系进行建模？

$$f(x_i) = wx_i + b \text{ 使得 } f(x_i) \approx y_i$$

- 需要学习的参数： w, b
- 机器学习的任务：基于训练数据确定 w, b 的取值

一元线性回归(linear regression)

模型: $f(x_i) = wx_i + b$, 目标: $f(x_i) \cong y_i$

- 模型在每个样本 x_i 的预测值 $f(x_i)$ 与真实标注(实际值) y_i 之差记为

$$(f(x_i) - y_i)^2$$

- 训练集中 n 个样本所产生误差总和为:

$$L(f) = \sum_{i=1}^n (y_i - f(x_i))^2$$

均方误差(mean squared error)

- 目标: $(w^*, b^*) = \arg \min_{(w,b)} \sum_{i=1}^n (y_i - f(x_i))^2$ 最小二乘法(least square method)

一元线性回归(linear regression)

$$\begin{aligned}(w^*, b^*) &= \arg \min_{(w,b)} \sum_{i=1}^n (y_i - f(x_i))^2 \\&= \arg \min_{(w,b)} \sum_{i=1}^n (y_i - wx_i - b)^2\end{aligned}$$

将 $L(w, b)$ 分别对 w 和 b 求导：

$$\frac{\partial L(w, b)}{\partial w} = \sum_{i=1}^n 2(y_i - wx_i - b)(-x_i) = 2 \left(w \sum_{i=1}^n x_i^2 - \sum_{i=1}^n (y_i - b)x_i \right)$$

$$\frac{\partial L(w, b)}{\partial b} = \sum_{i=1}^n 2(y_i - wx_i - b)(-1) = 2 \left(nb - \sum_{i=1}^n (y_i - wx_i) \right)$$

一元线性回归(linear regression)

令导数为0， 得到闭式(closed-form)解：

$$b = \frac{1}{n} \sum_{i=1}^n (y_i - wx_i) = \bar{y} - w\bar{x} \quad (\bar{x}, \bar{y} \text{表示均值})$$

$$w = \frac{\sum_{i=1}^n x_i y_i - n\bar{x}\bar{y}}{\sum_{i=1}^n x_i^2 - n\bar{x}^2}$$

$$\frac{\partial L(w, b)}{\partial w} = 2 \left(w \sum_{i=1}^n x_i^2 - \sum_{i=1}^n (y_i - b)x_i \right)$$

$$\frac{\partial L(w, b)}{\partial b} = 2 \left(nb - \sum_{i=1}^n (y_i - wx_i) \right)$$

一元线性回归(linear regression)

- 例子：下表给出了芒提兹尼欧 (Montesinho) 地区发生森林火灾的部分历史数据，表中列举了每次发生森林火灾时的气温温度取值 x 和受到火灾影响的森林面积 y

气温温度 x	5.1	8.2	11.5	13.9	15.1	16.2	19.6	23.3
火灾影响面积 y	2.14	4.62	8.24	11.24	13.99	16.33	19.23	28.74

$$b = \frac{1}{n} \sum_{i=1}^n (y_i - w x_i) = \bar{y} - w \bar{x} \quad w = \frac{\sum_{i=1}^n x_i y_i - n \bar{x} \bar{y}}{\sum_{i=1}^n x_i^2 - n \bar{x}^2}$$

基于训练样本可得： $w = \frac{x_1 y_1 + x_2 y_2 + \dots + x_8 y_8 - 8 \bar{x} \bar{y}}{x_1^2 + x_2^2 + \dots + x_8^2 - 8 \bar{x}^2} = 1.428, b = \bar{y} - w \bar{x} = -7.09$

即预测芒提兹尼欧地区火灾所影响森林面积与气温温度之间的一元线性回归模型为
“火灾所影响的森林面积 = 1.428 × 气温温度 - 7.09”，即 $y = 1.428x - 7.09$

多元线性回归(linear regression)

- 例子：接下来扩展到数据特征的维度是多维的情况，在上述数据中增加一个影响火灾影响面积的潜在因素—风力

气温 x	5.1	8.2	11.5	13.9	15.1	16.2	19.6	23.3
风力 z	4.5	5.8	4	6.3	4	7.2	6.3	8.5
火灾影响面积 y	2.14	4.62	8.24	11.24	13.99	16.33	19.23	28.74

假设总共有 n 个训练样本 $\{(x_i, y_i)\}_{i=1}^n$ ，其中 $x_i = [x_{i,1}, x_{i,2}, \dots, x_{i,d}] \in \mathbb{R}^d$ ， d 为数据特征的维度

线性模型(linear model)试图学得一个通过属性的线性组合来进行预测的函数

$$f(x_i) = w_1 x_{i1} + w_2 x_{i2} + \cdots + w_d x_{id} + b$$

向量形式： $f(x) = w^\top x + b$

多元线性回归-矩阵形式

- 矩阵形式：给定数据集 $D = \{X, Y\}, X \in \mathbb{R}^{n \times d}, Y \in \mathbb{R}^{n \times 1}$

$$X = \begin{pmatrix} x_{11} & x_{12} & \cdots & x_{1d} \\ x_{21} & x_{22} & \cdots & x_{2d} \\ \vdots & \vdots & \ddots & \vdots \\ x_{n1} & x_{n2} & \cdots & x_{nd} \end{pmatrix} \quad Y = \begin{bmatrix} y_1 \\ y_2 \\ \vdots \\ y_d \end{bmatrix}$$

- $f(X) = XW + b$

其中 $W = (w_1; w_2; \dots; w_d) \in \mathbb{R}^{d \times 1}$

$f(X) = XW + b$ 使得 $f(X) \cong Y$

多元线性回归-齐次表达

- 把 W 和 b 吸收入矩阵 \hat{W}

$$\hat{W} = \begin{bmatrix} w_1 \\ w_2 \\ \vdots \\ w_d \\ b \end{bmatrix}$$

- 在特征矩阵后添加一列，元素全部置为1

$$X = \begin{pmatrix} x_{11} & x_{12} & \cdots & x_{1d} & 1 \\ x_{21} & x_{22} & \cdots & x_{2d} & 1 \\ \vdots & \vdots & \ddots & \vdots & \vdots \\ x_{n1} & x_{n2} & \cdots & x_{nd} & 1 \end{pmatrix}$$

- 可得：

$$f(X) = X\hat{W}$$

多元线性回归(linear regression)

- 同样采用最小二乘法求解，有

$$\hat{W} = \arg \min_{\hat{W}} (Y - X\hat{W})^\top (Y - X\hat{W})$$

- 对 \hat{W} 求导可得：

$$\frac{\partial L(\hat{W})}{\partial \hat{W}} = -2X^\top (X\hat{W} - Y)$$

- 令导数为0可得：

$$\hat{W} = (X^\top X)^{-1} X^\top Y$$

均方误差函数是一个二次的凸函数，所以函数只存在一个极小值点，也同样是最大值点

多元线性回归(linear regression)

□ 例子：接下来扩展到数据特征的维度是多维的情况，在上述数据中增加一个影响火灾影响面积的潜在因素—风力

气温 x	5.1	8.2	11.5	13.9	15.1	16.2	19.6	23.3
风力 z	4.5	5.8	4	6.3	4	7.2	6.3	8.5
火灾影响面积 y	2.14	4.62	8.24	11.24	13.99	16.33	19.23	28.74

对于上面的例子，转化为矩阵的表示形式为：

$$X = \begin{bmatrix} 5.1 & 8.2 & 11.5 & 13.9 & 15.1 & 16.2 & 19.6 & 23.3 \\ 4.5 & 5.8 & 4 & 6.3 & 4 & 7.2 & 6.3 & 8.5 \\ 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 \end{bmatrix}^T$$
$$Y = [2.14 \quad 4.62 \quad 8.24 \quad 11.24 \quad 13.99 \quad 16.33 \quad 19.23 \quad 28.74]^T$$

计算可得

$$W = [1.312 \quad 0.626 \quad -9.103]$$

$$Y = 1.312x + 0.626z - 9.103$$

使用sklearn实现

```
In [10]: import numpy as np
from sklearn.linear_model import LinearRegression
x = np.array([[5.1],[8.2],[11.5],[13.9],[15.1],[16.2],[19.6],[23.3]])
y = np.array([2.14,4.62,8.24,11.24,13.99,16.33,19.23,28.74])
reg = LinearRegression().fit(x,y)
```

```
In [11]: print(reg.coef_)
print(reg.intercept_)
```

```
[1.42835273]
-7.09137783221065
```

```
In [14]: reg.predict([[18.1]])
```

```
Out[14]: array([18.76180649])
```

```
In [15]: x = np.array([[5.1,4.5],[8.2,5.8],[11.5,4],[13.9,6.3],[15.1,4],[16.2,7.2],[19.6,6.3],[23.3,8.5]])
y = np.array([2.14,4.62,8.24,11.24,13.99,16.33,19.23,28.74])
```

```
In [16]: reg = LinearRegression().fit(x,y)
```

```
In [17]: print(reg.coef_)
print(reg.intercept_)
```

```
[1.31227219 0.62649508]
-9.102525137839692
```

线性回归

□ 思考：

- 闭式解为： $\hat{W} = (X^T X)^{-1} X^T Y$ ，回想一下矩阵求逆，什么时候没有唯一解？应该怎么办？

大纲

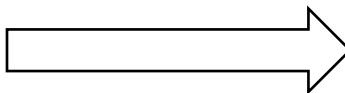
- 近邻算法
- 决策树
- 贝叶斯分类器
- 线性回归
- 对数几率回归
- 支持向量机

考虑二分类任务

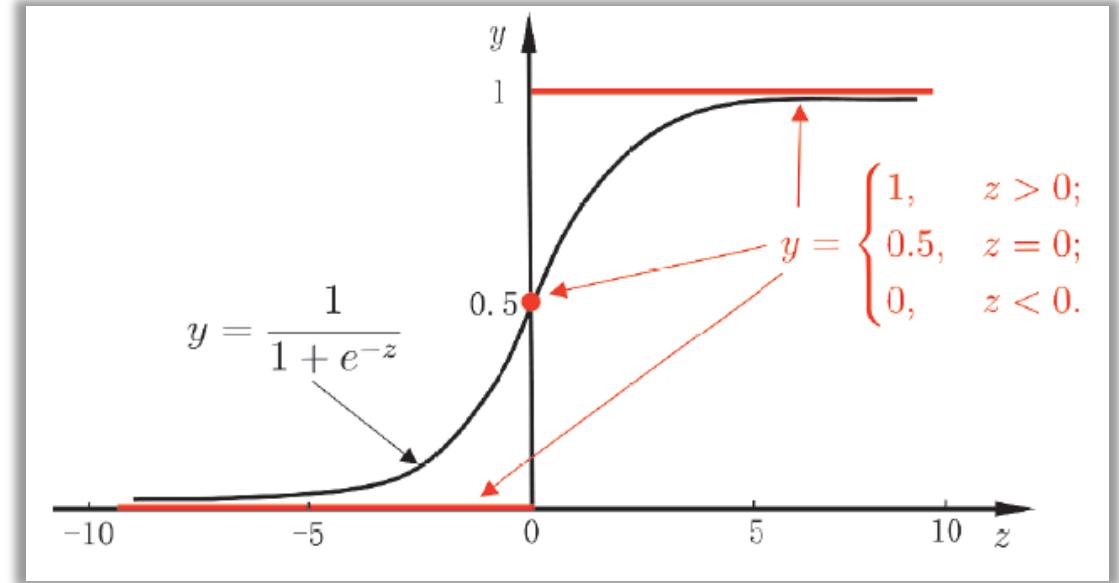
- 线性回归模型产生的实值输出 $z = w^T x + b$
- 期望输出 $y \in \{0,1\}$
- 能不能找一个函数把 z 和 y 联系起来？

“单位阶跃函数”
(unit-step function)

$$y = \begin{cases} 0, z < 0 \\ 0.5, z = 0 \\ 1, z > 0 \end{cases}$$



性质不好，
需找 “替代函数”
(surrogate function)



单调可微、任意阶可导
常用

$$y = \frac{1}{1 + e^{-z}}$$

Sigmoid 函数

对数几率回归(logistic regression)

- 对数几率回归(logistic regression)就是在回归模型中引入 sigmoid 函数的一种模型
- Logistic 回归模型可如下表示：

$$y = \frac{1}{1 + e^{-z}} = \frac{1}{1 + e^{-(\mathbf{w}^T \mathbf{x} + b)}}$$

其中 $\frac{1}{1+e^{-z}}$ 是 sigmoid 函数、 $x \in \mathbb{R}^d$ 是输入数据、 $w \in \mathbb{R}^d$ 和 $b \in \mathbb{R}$ 是待求解的模型参数

对数几率回归(logistic regression)

$$y = \frac{1}{1 + e^{-(w^T x + b)}} \rightarrow \ln \frac{y}{1 - y} = w^T x + b$$

“对数几率”
(log odds, 亦称 logit)

几率(odds), 反映了 x 作为正例的相对可能性

- 如果输入数据 x 属于正例的概率大于其属于负例的概率, 即 $p(y = 1|x) > 0.5$, 则输入数据 x 可被判断属于正例, 这一结果等价于 $\frac{p(y = 1|x)}{p(y = 0|x)} > 1$, 即 $\ln \left(\frac{p(y = 1|x)}{p(y = 0|x)} \right) > \ln 1 = 0$, 也就是 $w^T x + b > 0$ 成立

“对数几率回归” (logistic regression)
简称 “对率回归”, 跟逻辑没有关系



对数几率回归(logistic regression)

- 给定数据集 $D = \{(x_1, y_1), (x_2, y_2), \dots, (x_n, y_n)\}$, $x_i \in \mathbb{R}^d$, $y_i \in \{0,1\}$
- 若将 y 看作类后验概率估计, 则有

$$\ln \frac{p(y=1|x)}{p(y=0|x)} = w^\top x + b$$

- 显然有,

$$p(y=1|x) = \frac{e^{w^\top x + b}}{1 + e^{w^\top x + b}}$$

$$p(y=0|x) = \frac{1}{1 + e^{w^\top x + b}}$$

对数几率回归(logistic regression)

- 我们希望每个样本属于其真实标记的概率越大越好

$$\max \sum_{i=1}^n \ln p(y_i|x_i; w, b)$$

似然项

极大似然估计
(maximum likelihood method)

- 根据

$$p(y = 1|x) = \frac{e^{w^\top x + b}}{1 + e^{w^\top x + b}} = f(x)$$

$$p(y = 0|x) = \frac{1}{1 + e^{w^\top x + b}} = 1 - f(x)$$

- 等价于最大化

$$\sum_{i=1}^n y_i \ln(f(x_i)) + (1 - y_i) \ln(1 - f(x_i))$$

- 即最小化

$$L(f) = \sum_{i=1}^n -y_i \ln(f(x_i)) - (1 - y_i) \ln(1 - f(x_i))$$

交叉熵损失

高阶可导连续凸函数，可用经典的数值优化方法
如梯度下降法/牛顿法 [Boyd and Vandenberghe, 2004]

使用sklearn实现

```
class sklearn.linear_model.LogisticRegression(penalty='l2', *, dual=False,  
tol=0.0001, C=1.0, fit_intercept=True, intercept_scaling=1, class_weight=None,  
random_state=None, solver='lbfgs', max_iter=100, multi_class='auto',  
verbose=0, warm_start=False, n_jobs=None, l1_ratio=None)
```

In [19]: data

Out[19]:

	色泽	根蒂	敲声	纹理	脐部	触感	好瓜
0	2	2	1	1	0	0	1
1	0	2	0	1	0	0	1
2	0	2	1	1	0	0	1
3	2	2	0	1	0	0	1
4	1	2	1	1	0	0	1
5	2	1	1	1	2	1	1
6	0	1	1	2	2	1	1
7	0	1	1	1	2	0	1
8	0	1	0	2	2	0	0
9	2	0	2	1	1	1	0
10	1	0	2	0	1	0	0
11	1	2	1	0	1	1	0
12	2	1	1	2	0	0	0
13	1	1	0	2	0	0	0
14	0	1	1	1	2	1	0
15	1	2	1	0	1	0	0
16	2	2	0	2	2	0	0

In [6]:

```
from sklearn.linear_model import LogisticRegression  
from sklearn.tree import DecisionTreeClassifier  
LR = LogisticRegression()  
clf = DecisionTreeClassifier()  
  
LR.fit(data.iloc[:, :-1], data.iloc[:, -1])  
clf.fit(data.iloc[:, :-1], data.iloc[:, -1])
```

In [7]:

```
print(clf.predict([[1, 1, 0, 1, 1, 0]]))  
print(LR.predict([[1, 1, 0, 1, 1, 0]]))
```

[1]
[0]

In [8]:

```
from sklearn.metrics import accuracy_score  
print("Training Accuracy of Logistic Regression: ", accuracy_score(LR.predict(data.iloc[:, :-1]), data.iloc[:, -1]))  
print("Training Accuracy of Decision Tree: ", accuracy_score(clf.predict(data.iloc[:, :-1]), data.iloc[:, -1]))
```

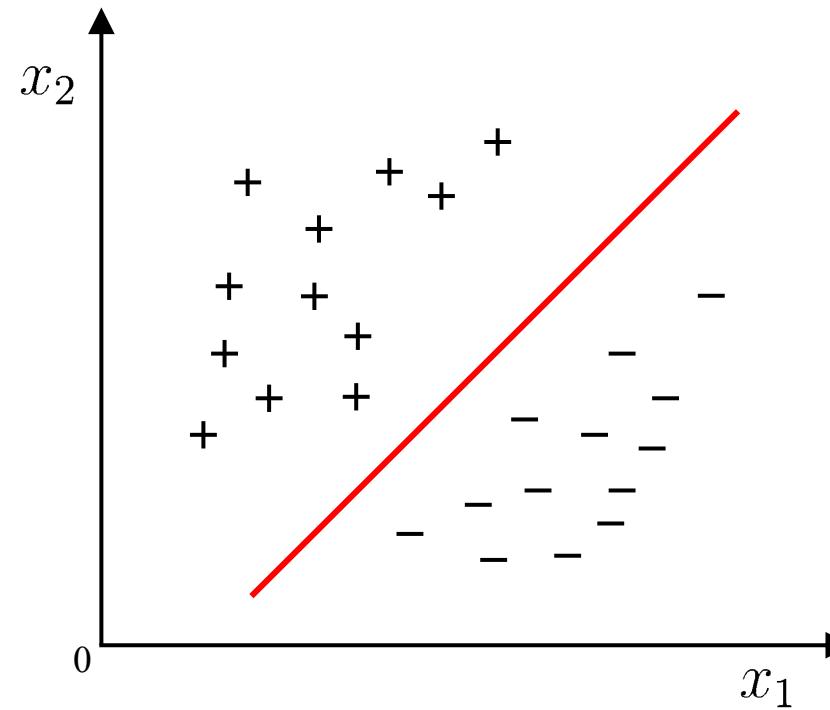
Training Accuracy of Logistic Regression: 0.7058823529411765
Training Accuracy of Decision Tree: 1.0

大纲

- 近邻算法
- 决策树
- 贝叶斯分类器
- 线性回归
- 对数几率回归
- 支持向量机

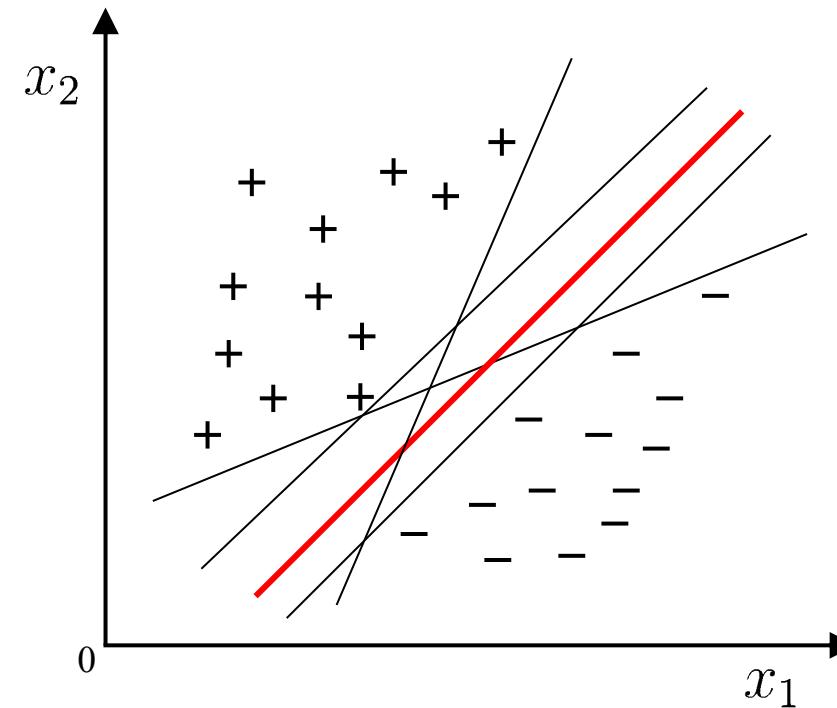
线性分类器

在样本空间中寻找一个超平面，将不同类别的样本分开



线性分类器

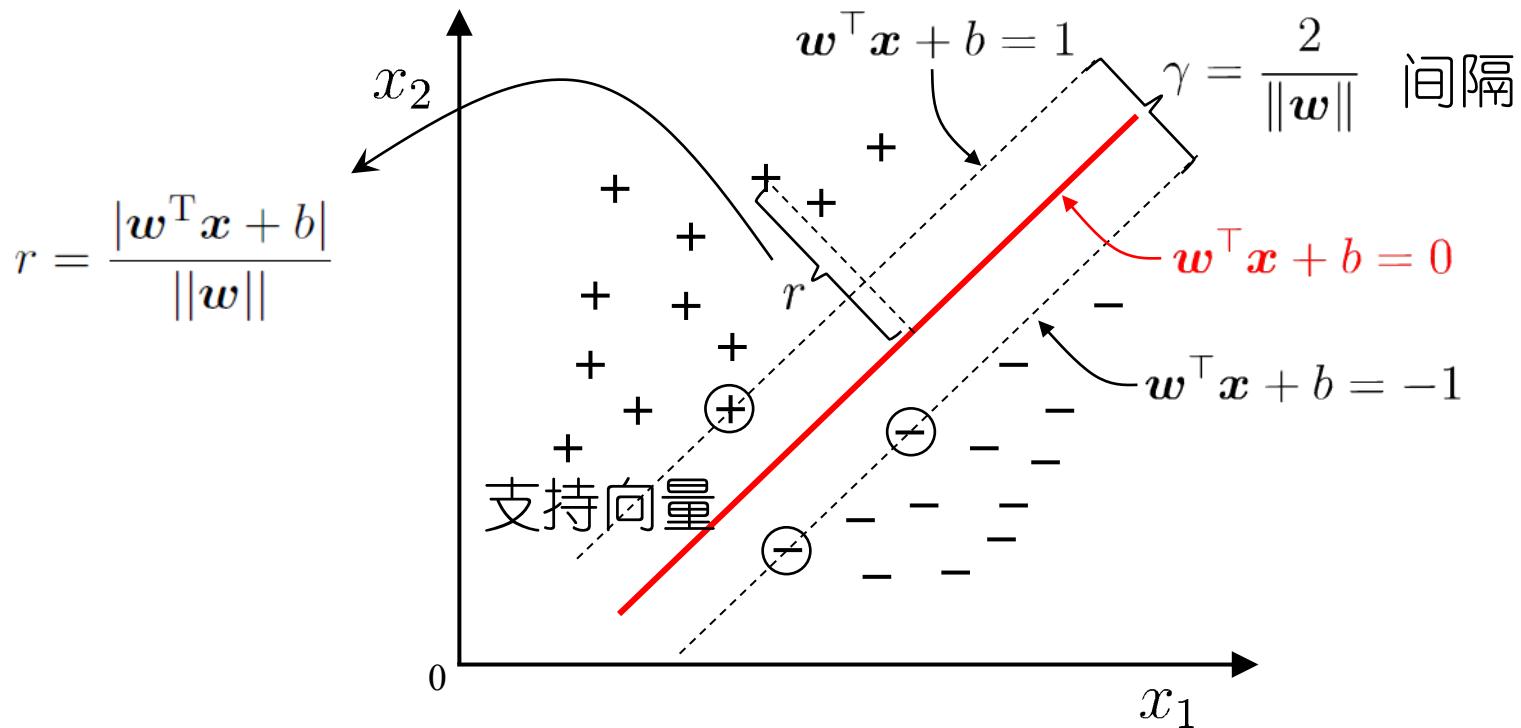
将训练样本分开的超平面可能有很多，哪一个更好呢？



“正中间”的：鲁棒性最好，泛化能力最强

间隔(margin)与支持向量(support vector)

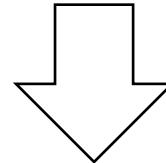
$$\mathbf{w}^\top \mathbf{x} + b = 0$$



支持向量机-基本型

最大间隔：寻找参数 \mathbf{w} 和 b ，使得 γ 最大

$$\begin{aligned} & \arg \max_{\mathbf{w}, b} \frac{2}{\|\mathbf{w}\|} \\ \text{s.t. } & y_i(\mathbf{w}^\top \mathbf{x}_i + b) \geq 1, \quad i = 1, 2, \dots, m. \end{aligned}$$



$$\begin{aligned} & \arg \min_{\mathbf{w}, b} \frac{1}{2} \|\mathbf{w}\|^2 \\ \text{s.t. } & y_i(\mathbf{w}^\top \mathbf{x}_i + b) \geq 1, \quad i = 1, 2, \dots, m. \end{aligned}$$

凸二次规划问题，能用优化计算包求解，但可以有更高效的办法

支持向量机-对偶型

拉格朗日乘子法

□ 第一步：引入拉格朗日乘子 $\alpha_i \geq 0$ 得到拉格朗日函数

$$L(\mathbf{w}, b, \boldsymbol{\alpha}) = \frac{1}{2} \|\mathbf{w}\|^2 + \sum_{i=1}^m \alpha_i (1 - y_i (\mathbf{w}^\top \mathbf{x}_i + b))$$

□ 第二步：令 $L(\mathbf{w}, b, \boldsymbol{\alpha})$ 对 \mathbf{w} 和 b 的偏导为零可得

$$\mathbf{w} = \sum_{i=1}^m \alpha_i y_i \mathbf{x}_i , \quad 0 = \sum_{i=1}^m \alpha_i y_i$$

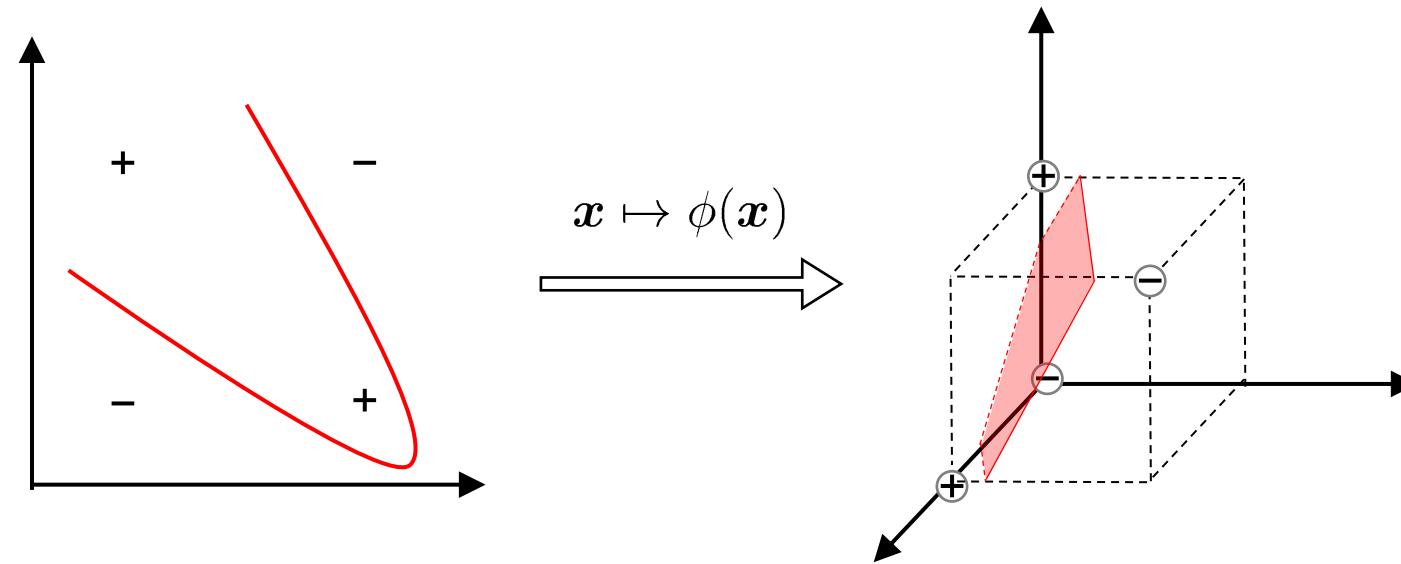
□ 第三步：回代可得

$$\begin{aligned} \max_{\boldsymbol{\alpha}} \quad & \sum_{i=1}^m \alpha_i - \frac{1}{2} \sum_{i=1}^m \sum_{j=1}^m \alpha_i \alpha_j y_i y_j \mathbf{x}_i^\top \mathbf{x}_j \\ \text{s.t.} \quad & \sum_{i=1}^m \alpha_i y_i = 0 , \quad \alpha_i \geq 0 , \quad i = 1, 2, \dots, m \end{aligned}$$

特征空间映射

若不存在一个能正确划分两类样本的超平面，怎么办？

将样本从原始空间映射到一个更高维的特征空间，使样本在这个特征空间内线性可分



如果原始空间是有限维(属性数有限)，那么一定存在一个高维特征空间使样本可分

特征空间映射

设样本 \mathbf{x} 映射后的向量为 $\phi(\mathbf{x})$, 划分超平面为 $f(\mathbf{x}) = \mathbf{w}^\top \phi(\mathbf{x}) + b$

原始问题

$$\begin{aligned} & \min_{\mathbf{w}, b} \quad \frac{1}{2} \|\mathbf{w}\|^2 \\ \text{s.t.} \quad & y_i (\mathbf{w}^\top \phi(\mathbf{x}_i) + b) \geq 1, \quad i = 1, 2, \dots, m. \end{aligned}$$

对偶问题

$$\begin{aligned} & \max_{\alpha} \quad \sum_{i=1}^m \alpha_i - \frac{1}{2} \sum_{i=1}^m \sum_{j=1}^m \alpha_i \alpha_j y_i y_j \boxed{\phi(\mathbf{x}_i)^\top \phi(\mathbf{x}_j)} \\ \text{s.t.} \quad & \sum_{i=1}^m \alpha_i y_i = 0, \quad \alpha_i \geq 0, \quad i = 1, 2, \dots, m \end{aligned}$$



特征空间映射

基本思路：设计核函数

$$\kappa(\mathbf{x}_i, \mathbf{x}_j) = \phi(\mathbf{x}_i)^T \phi(\mathbf{x}_j)$$

绕过显式考虑特征映射、以及计算高维内积的困难

Mercer 定理：若一个对称函数所对应的核矩阵半正定，则它就能作为核函数来使用

任何一个核函数，都隐式地定义了一个RKHS (Reproducing Kernel Hilbert Space, 再生核希尔伯特空间)

“核函数选择”成为决定支持向量机性能的关键！

核函数

常用核函数

名称	表达式	参数
线性核	$\kappa(\mathbf{x}_i, \mathbf{x}_j) = \mathbf{x}_i^T \mathbf{x}_j$	
多项式核	$\kappa(\mathbf{x}_i, \mathbf{x}_j) = (\mathbf{x}_i^T \mathbf{x}_j)^d$	$d \geq 1$ 为多项式的次数
高斯核	$\kappa(\mathbf{x}_i, \mathbf{x}_j) = \exp\left(-\frac{\ \mathbf{x}_i - \mathbf{x}_j\ ^2}{2\sigma^2}\right)$	$\sigma > 0$ 为高斯核的带宽(width)
拉普拉斯核	$\kappa(\mathbf{x}_i, \mathbf{x}_j) = \exp\left(-\frac{\ \mathbf{x}_i - \mathbf{x}_j\ }{\sigma}\right)$	$\sigma > 0$
Sigmoid 核	$\kappa(\mathbf{x}_i, \mathbf{x}_j) = \tanh(\beta \mathbf{x}_i^T \mathbf{x}_j + \theta)$	\tanh 为双曲正切函数, $\beta > 0, \theta < 0$

使用sklearn实现

```
class sklearn.svm.SVC(*, C=1.0, kernel='rbf', degree=3, gamma='scale',
coef0=0.0, shrinking=True, probability=False, tol=0.001, cache_size=200,
, class_weight=None, verbose=False, max_iter=-1, decision_function_shape='ovr', break_ties=False, random_state=None
```

```
>>> import numpy as np
>>> from sklearn.pipeline import make_pipeline
>>> from sklearn.preprocessing import StandardScaler
>>> X = np.array([[-1, -1], [-2, -1], [1, 1], [2, 1]])
>>> y = np.array([1, 1, 2, 2])
>>> from sklearn.svm import SVC
>>> clf = make_pipeline(StandardScaler(), SVC(gamma='auto'))
>>> clf.fit(X, y)
Pipeline(steps=[('standardscaler', StandardScaler()),
                 ('svc', SVC(gamma='auto'))])
```

```
>>> print(clf.predict([-0.8, -1]))
[1]
```

小结

- 常用评价指标：分类正确率，回归均方误差
- 近邻学习器：懒惰学习的代表，无需训练
- 决策树：不断选择属性划分节点建树
- 贝叶斯分类器：生成式方法的代表，估计联合分布再计算后验分布
- 线性回归：最小二乘法的闭式解
- 对数几率回归：引入sigmoid函数解决分类问题
- 支持向量机：大间隔、核函数