decision trees

basics

- test & prediction
- input & output:

- We will denote the input space by $\boldsymbol{X}$; points in $\boldsymbol{X}$ are vectors $\boldsymbol{x} = (x_1, x_2, \ldots, x_n)$
- Any $\boldsymbol{x}$ is mapped to exactly one value y for Y
- Hence, the tree represents a function from $\boldsymbol{X}$ to Y

- representation power

  can every imaginable boolean function be represented?

  yes
- continuous input attributes

  cannot make a different child node for each possible value!

  solution: comparative tests
- classication tree & regression tree
- why? (advantages)

  efficient

  good predictive accuracy

  interpretable

the basic learning algorithm

task 1: smallest

Find the smallest tree T such that $\forall(\boldsymbol{x}, f(\boldsymbol{x})) \in D : T(\boldsymbol{x}) = f(\boldsymbol{x})$
(= smallest tree *consistent* with the data)

- smallest tree = simplest explanation, provides insight in the data
- not practical, D is only a sample from some larger distribution (lack of generalization)

task 2: minimal risk (loss)

- loss function
- expected loss
  - The **risk** R of T, relative to f, is $\mathbf{E}_{\boldsymbol{x}\sim\mathscr{D}}[(\ell(T(\boldsymbol{x}), f(\boldsymbol{x}))]$ (the expected value of $\ell(T(\boldsymbol{x}), f(\boldsymbol{x}))$, with $\boldsymbol{x}$ drawn from $\mathscr{D}$)

using only the data in D, we want both aspects to be achieved (is small & generalizes well)

learning algorithm

- hardness of learning decision trees: NP-hard
- the basic principle: TDIDT, recursive partitioning
  - Start with the full data set D
  - Find a test such that examples in D with the same outcome for the test tend to have the same value of Y
  - Split D into subsets, one for each outcome of that test
  - Repeat this procedure on each subset that is not yet sufficiently "pure" (meaning, not all elements have the same Y)
  - Keep repeating until no further splits possible
- two important questions:
  - how to choose the "best" test
  - when to stop splitting nodes

choosing a test: classification

- information entropy

- Given a set of values $c_1, c_2, \ldots, c_k$ with respective probabilities $p_1, p_2, \ldots, p_k$, an encoding exists that uses, on average, e bits for representing a randomly drawn value, where

$$e = -\sum_{i=1}^{k} p_i \log_2(p_i)$$

e reflects the minimal number of bits that you will need (on average) to encode one value

- class entropy

- The **class entropy** of a set S of objects $(\boldsymbol{x}, y)$, where y can be any of k classes c_i, is defined as

$$CE(S) = -\sum_{i=1}^{k} p_i \log_2(p_i) \qquad \text{with } p_i = \frac{|\{(\boldsymbol{x}, y) \in S \,|\, y = c_i\}|}{|S|}$$

(proportion of elements in S with class c_i)

high entropy = (high uncertainty)

"many possibilities, all equally likely"

low entropy =

"few possibilities" /

"many possibilities but most are highly unlikely"

- information gain

expected reduction of entropy by obtaining the answer to a question

- In the case of classification trees: expected reduction of class entropy:

$$IG(S, t) = CE(S) - \mathbf{E}[CE(S_i)] = CE(S) - \sum_{i=1}^{o} \frac{|S_i|}{|S|} CE(S_i)$$

with t a test, o the number of possible outcomes of t, and S_i the subset of S for which the i'th outcome was obtained

this measurement is for: given **a dataset** and **a test**!

choosing a test: regression

<u>goal</u>: examples in one subset have similar Y values

use variance reduction instead of information gain:

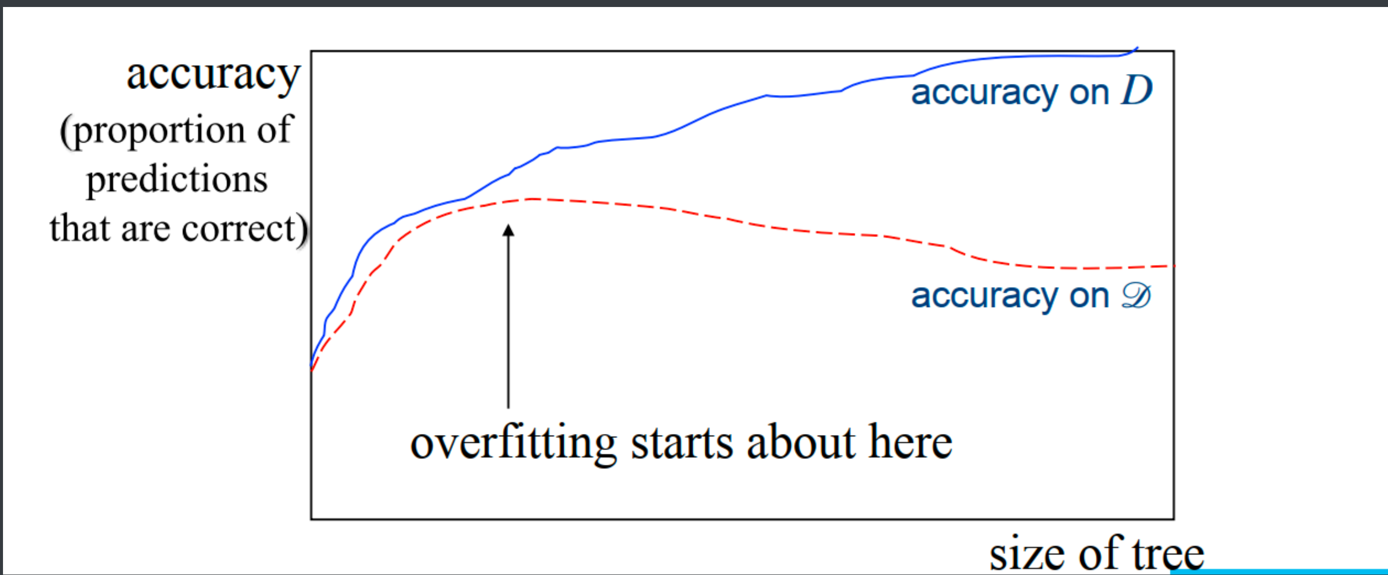
$$Var(S) = \frac{\sum_{(x,y)\in S}(y-\bar{y})^2}{|S|-1} \qquad \text{with } \bar{y} = \frac{\sum_{(x,y)\in S} y}{|S|}$$

$$VR(S,t) = Var(S) - \sum_{i=1}^{o} \frac{|S_i|}{|S|} Var(S_i)$$

stopping criteria

until all instances in a subset have the same Y value

-> useful for classificaiton, but overfitting for regression!



to avoid overfitting,

1. cautious splitting

   use a **validation set** to guess if the model is going to be overfitted

   but what if the guess is wrong? this is not always reliable

2. post-pruning

   grow the tree to full size, then cut aways branches that don't contribute to getting better predictions

   顺序：叶节点到高层节点

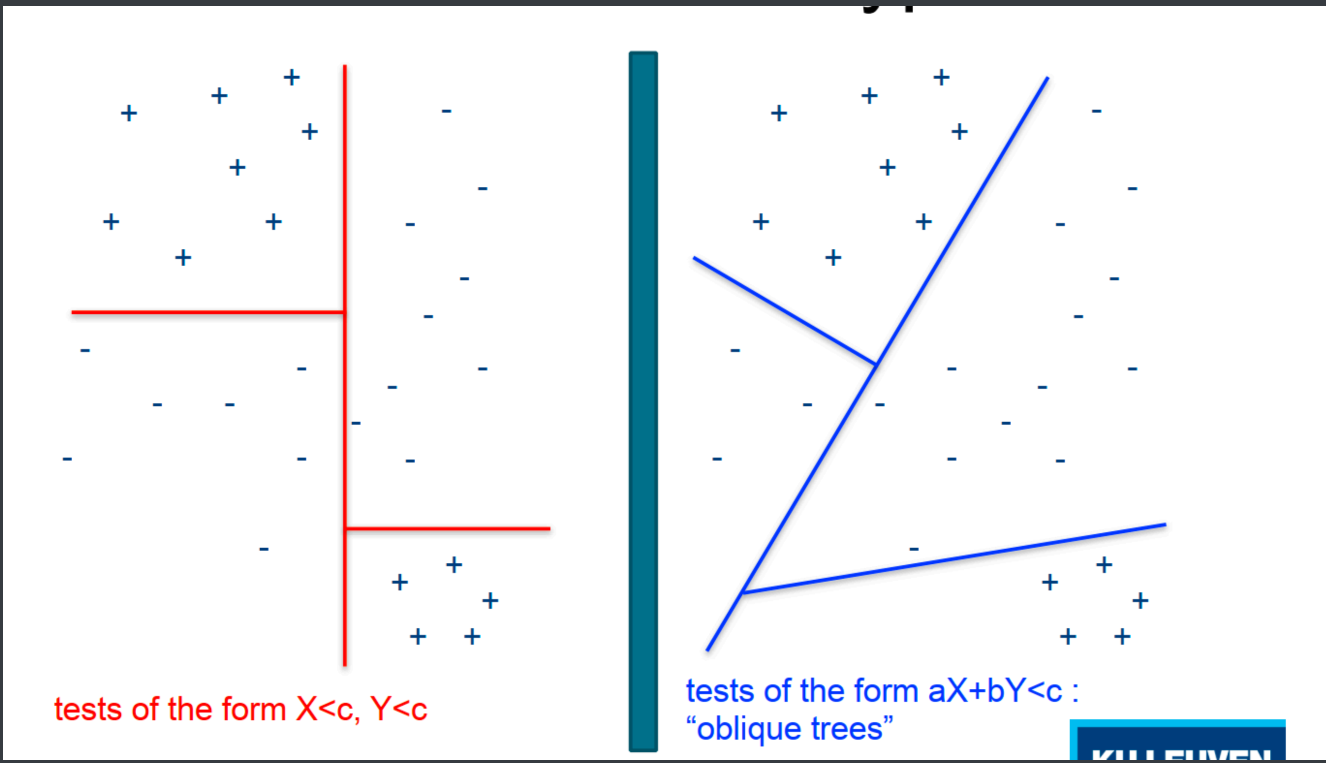
a generic TDIDT algorithm

```
function TDIDT(E: set of examples) returns tree;
    T' = grow_tree(E );
    T = prune_tree(T');
    return T;

function grow_tree(E: set of examples) returns tree;
    T = generate_tests(E);
    t = best_test(T, E);                              (call t's outcomes v1…vk)
    P = {E1, E2, …, Ek} with Ei={x∈E | t(x)=vi}       (P = partition induced on E by t)
    if stop_criterion(E, P)
    then return leaf(info(E))
    else
        for all Ei in P: Ti := grow_tree(Ei);
        return node(t, {(v1,T1), (v2, T2), … (vk, Tk)} );
```

- generate_tests

  variants: one subtree per set of values, **oblique trees**

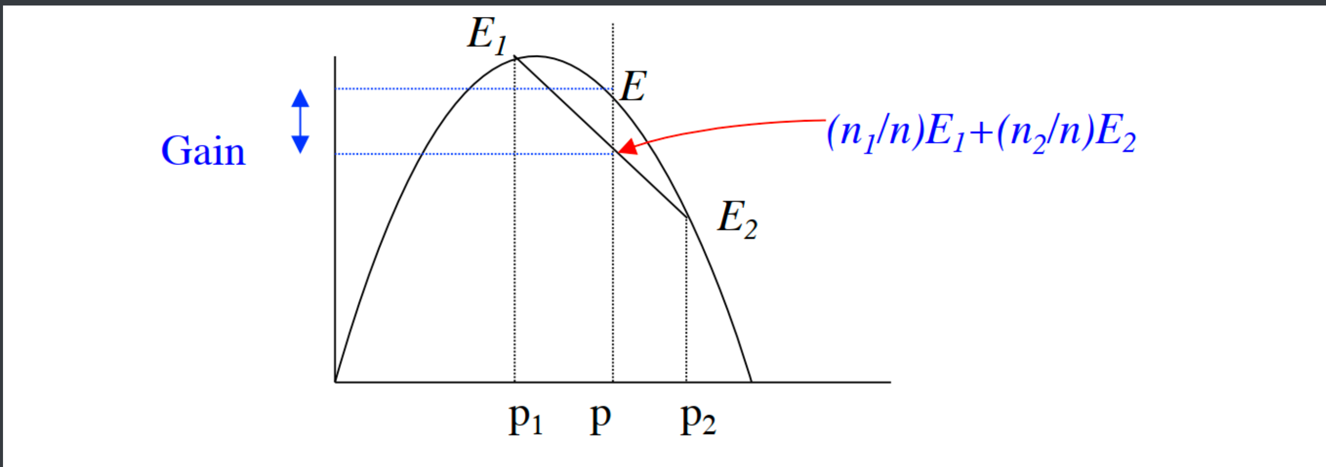


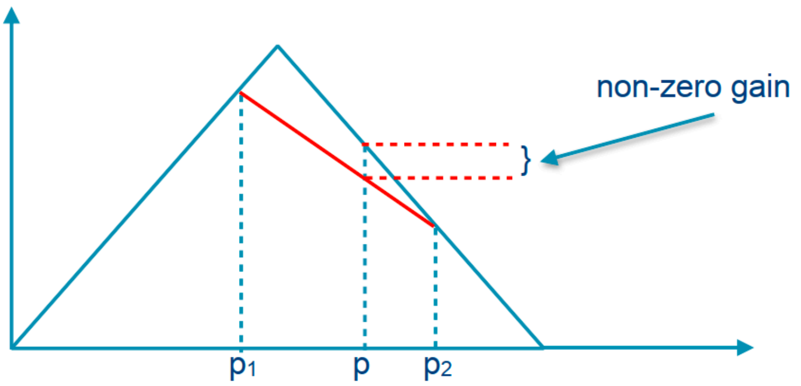
in practice, non-oblique trees are more common

- best_test

  impurity 不纯度 should be the reduction goal of building a tree

  SEMINAL conclusion: good impurity measures are strictly concave





Accuracy will only improve if at least one child node has a majority class different from that of its parent

但这通常too restrictive：

一些good tests虽然不是严格的非零增益，却还是能降低数据的不纯度

<u>for classification trees</u>

information gain:

Gini impurity reduction:

$$Gini(S) = 1 - \sum_{i=1}^{k} p_i^2$$

gain ratio

= IG / class entropy:

- For a test t that splits S into n subsets S_i :

$$SI(S,t) = -\sum_{i=1}^{n} \frac{|S_i|}{|S|} \log_2 \frac{|S_i|}{|S|}$$ (Note: this is the “t-entropy” in S; cf. class entropy)

- The Gain Ratio is: $GR(S,t) = IG(S,t)/SI(S,t)$

<u>for regression trees</u>

how to decide the c in "x<c"?

typically, just try all values, complexity -- in next section

- stop_criterion

post-pruning is always preferred!

still, early drop out

- info (in leaf nodes)

  for classification trees

  most frequent class in this leaf / class distribution / all training examples relevant for that leaf

  for regression trees

  mean / median ...

- prune_tree

  order matters

complexity

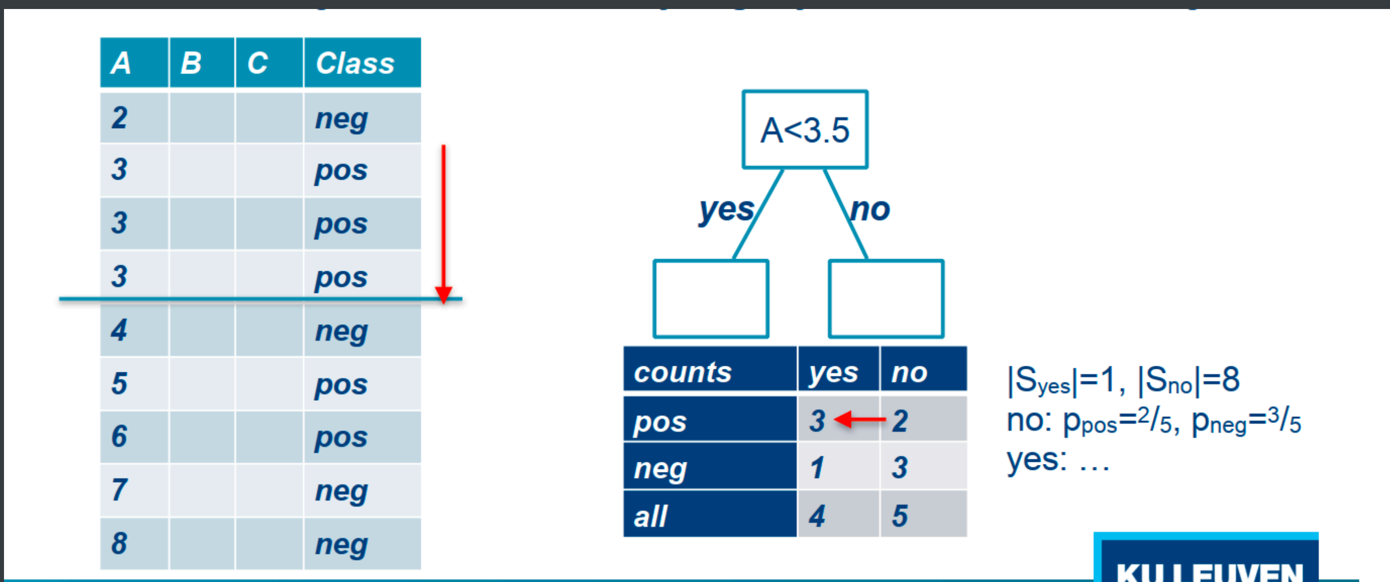
tree construction 的主要计算量：

split每个节点的时候，为每个可能的test，partition数据集并计算在此partition上的性质

对于取值离散的classification任务，这个计算是简单的；

对于取值连续的回归任务，尝试all values：

但实际上复杂度并不高——首先根据分类标准A将样本排序，移动threshold即可



- 预测是非常快的（理想情况下，如果树的高度不高）；而对于训练：
- 分裂一个节点的复杂度：

节点上的样本数n（不是总样本数N），属性数量m

则复杂度(mn) （对每个属性计算，每次计算遍历n个样本）

- 分裂多个节点的复杂度：

  属性数量不变，m * n = m * (n1 + n2) = m * n1 + m * n2

  所以无论怎么分裂，树的每一层总工作量都是几乎不变的

- 总复杂度

  对于平衡的树，总复杂度就是O(m * N * logN)

  对于不太平衡的树，高度甚至可能是线性的，总复杂度O(m * N^2)

  好的分裂heuristic，比如IG，倾向于构建出更平衡的树

handling missing values

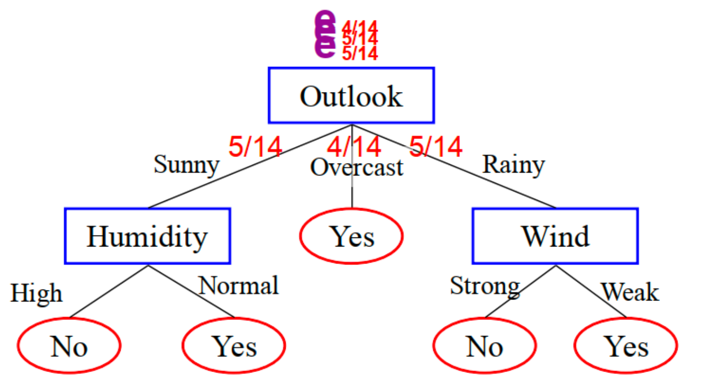
遇到缺省值

计算test quality的时候，直接跳过该样本即可

partition数据集的时候，两种方案：

1. 插值
2. distribution：一个样本在即将split的属性上有缺省值，将其加权重分配到不同分支上：

	Outlook	Hum.	Wind	Play
1	Sunny	High	Weak	No
2	sunny	high	strong	No
3	overcast	High	weak	yes
4	rain	High	Weak	yes
5	rain	normal	weak	yes
6	rain	normal	strong	no
7	overcast	normal	strong	yes
8	sunny	high	weak	no
9	Sunny	normal	weak	yes
10	rain	normal	weak	yes
11	Sunny	normal	strong	yes
12	overcast	high	Strong	yes
13	overcast	normal	weak	yes
14	rain	high	strong	No



e: 10/14 no, 4/14 yes => guess no

How to classify e : [15, ?, high, strong] ?

如果是预测阶段，直接加权投票得到预测结果；

如果是训练阶段，以0.3的权重分配到分支1算“0.3个”样本来计算基尼系数等；

model trees

each leaf contains a linear model

方差reduction假设每个叶节点是一个常数，不适用于包含一个线性模型的叶节点；

RETIS将其换成：对每个subset计算线性回归，然后计算平均预测残差的减少；但当属性数量非常多的时候效率很低；

mauve对此优化：计算 单变量线性回归 后的平均残差；

multi-target trees

to predict multiple labels, 3 main approaches:

1. binary relevance: learn a binary tree (yes / no) for each label
2. label powersets: build one tree that predicts the combined label
3. vector encoding: variance of vectors in `best_test`, mean vector in `info`

hierarchical multilabel classification (HMC problem)

a label can only occur in a set if all its ancestors also occur