

1: Decision Trees

CS1420: Machine Learning

Michael L. Littman
Spring 2018

<http://www.cs.huji.ac.il/~shais/UnderstandingMachineLearning/> . Chapters 2 & 18.

Communication Lights

We want to find what kind of lights work for communicating with people at a distance.

Attributes:

- Multicolored: yes/no
- Textured: yes/no
- Transparent: yes/no
- Pointed: yes/no
- Effective: yes/no



Given attribute values on many examples, which attribute is the most useful for predicting whether the lights will be effective?

Example Data

multicolored	textured	transparent	pointed	effective
yes	no	no	yes	<i>yes</i>
yes	no	no	no	<i>no</i>
yes	no	yes	yes	<i>no</i>
no	no	yes	yes	<i>no</i>
yes	no	yes	yes	<i>no</i>
no	yes	no	yes	<i>no</i>
yes	no	no	yes	<i>yes</i>
no	no	yes	yes	<i>no</i>

Best attribute?

Example Data

multicolored	textured	transparent	pointed	effective
yes	no	no	yes	<i>yes</i>
yes	no	no	no	<i>no</i>
yes	no	yes	yes	<i>no</i>
no	no	yes	yes	<i>no</i>
yes	no	yes	yes	<i>no</i>
no	yes	no	yes	<i>no</i>
yes	no	no	yes	<i>yes</i>
no	no	yes	yes	<i>no</i>

Best attribute? not transparent: 2/8 errors.

Problem Statement Terminology

Domain set: \mathcal{X} . Here, each $x \in \mathcal{X}$ is a d -dimensional Boolean vector. So, $\mathcal{X} = \{0,1\}^d$. The dimensions are sometimes called *features* or *attributes*.

Label set: \mathcal{Y} . Here, \mathcal{Y} is simply $\{0,1\}$.

Training data: $S = \{(x_1, y_1) \dots (x_m, y_m)\}$ is a finite set of pairs in $\mathcal{X} \times \mathcal{Y}$. It is pairs of instances and labels, which the learner can use to construct its rule. Sometimes S is called the *training set* and its elements *training examples*.

Learner's output: The learner needs to find $h: \mathcal{X} \rightarrow \mathcal{Y}$, a *predictor*, *hypothesis*, or a *classifier*. Given data S , algorithm A returns classifier $A(S)$. Our hypotheses here have the form $h(x) = x^i$ or $h(x) = 1 - x^i$ for some $0 \leq i < d$.

Problem Objective Terminology

Domain set: \mathcal{X} . Here, each $x \in \mathcal{X}$ is a d -dimensional Boolean vector. So, $\mathcal{X} = \{0,1\}^d$. The dimensions are sometimes called *features* or *attributes*.

Data-generation model: Assumption on how training data is generated. For this example, let \mathcal{D} be a distribution over \mathcal{X} , not known to learner. Labels produced by the *true labeling function* (*true function*, *target function*) $f: \mathcal{X} \rightarrow \mathcal{Y}$. So, $x_i \sim \mathcal{D}$, $y_i = f(x_i)$.

Measure of success: Error of h : $L_{\mathcal{D}}(h) = \Pr_{x \sim \mathcal{D}}(f(x) \neq h(x))$. Also called *generalization error*, the *risk*, *true error*, and is one kind of *loss*.

Training error: Error of h on the training data: $L_S(h) = \sum_{(x,y) \in S} (y \neq h(x))$. Also called *empirical error* and *empirical risk*.

Approach to Learning

We want to identify h that minimizes true error, but computing the true error requires knowing the true function.

Instead, *empirical risk minimization (ERM)*: find the hypothesis that does the best on the training set: $\operatorname{argmin}_{h \in H} L_S(h)$.

Here, the minimization is done with respect to some set of hypotheses H . In our example, that's the set of rules that look at just one attribute.

Is it Good?

How efficient is ERM? Recall, our data consists of d dimensional vectors and we have m examples.

Question

We have d dimensions and m examples in our training set.

How expensive is it to compute the empirical loss for all of the possible rules in terms of d and m ?

Decision stumps.

Is it Good?

How efficient is ERM? Recall, our data consists of d dimensional vectors and we have m examples.

For each possible rule (2^d , for decision stumps), we run through all the data computing the empirical risk ($O(m)$). We choose the best of these rules. Total time $O(dm)$.

Can we do better?

Generalization Error

Yes, the algorithm is “good” in that sense. But, does it produce good predictions??

Assumption: Realizability. $f \in H$. Or, more generally, $\exists h^* \in H$ s.t. $L_{\mathcal{D}}(h^*) = 0$.

What's the difference?

It means that there is a perfect hypothesis in the class, even if we don't know what it is.

What follows from this assumption?

Generalization Error

Yes, the algorithm is “good” in that sense. But, does it produce good predictions??

Assumption: Realizability. $f \in H$. Or, more generally, $\exists h^* \in H$ s.t. $L_{\mathcal{D}}(h^*) = 0$.

What’s the difference?

It means that there is a perfect hypothesis in the class, even if we don’t know what it is.

What follows from this assumption? $L_S(h^*) = 0$.

But, other “chaff” functions might also look good on the training data.

Probably Approximately Correct

Given a hypothesis h , it has some true error $L_{\mathcal{D}}(h)$. We might not be able to find the best h , but let's try to get one that is within ϵ of the best. Such a hypothesis is *approximately* correct.

Given that data is sampled randomly, we might get really unlucky and just get the same example over and over again, learning nothing useful. So, let's just guarantee our procedure works most of the time $1-\delta$. That is, it should *probably* be approximately correct.

If we can show that the amount of data needed to meet these conditions doesn't depend on \mathcal{D} and is small relative to $1/\epsilon$ and $1/\delta$, we call the procedure *PAC*.

Getting Rid of Chaff

Assumption: iid. Data is sampled independently and identically distributed. That is, each $x \sim \mathcal{D}$ separately.

Our procedure returns a hypothesis such that $L_S(h) = 0$.

Our procedure fails if it returns a *bad* h , one with $L_{\mathcal{D}}(h) > \epsilon$.

What's the probability that a randomly chosen instance will prove a bad hypothesis is actually bad?

Getting Rid of Chaff

Assumption: iid. Data is sampled independently and identically distributed. That is, each $x \sim \mathcal{D}$ separately.

Our procedure returns a hypothesis such that $L_S(h) = 0$.

Our procedure fails if it returns a *bad* h , one with $L_{\mathcal{D}}(h) > \varepsilon$.

What's the probability that a randomly chosen instance will prove a bad hypothesis is actually bad? At least ε .

What's the probability that it won't? At most $1 - \varepsilon$.

What's the probability that it won't m times in a row?

Getting Rid of Chaff

Assumption: iid. Data is sampled independently and identically distributed. That is, each $x \sim \mathcal{D}$ separately.

Our procedure returns a hypothesis such that $L_S(h) = 0$.

Our procedure fails if it returns a *bad* h , one with $L_{\mathcal{D}}(h) > \epsilon$.

What's the probability that a randomly chosen instance will prove a bad hypothesis is actually bad? At least ϵ .

What's the probability that it won't? At most $1 - \epsilon$.

What's the probability that it won't m times in a row? At most $(1 - \epsilon)^m$. (Because iid.)

Running Out of Luck

A single bad hypothesis will evade capture m times in a row with probability at most $(1 - \epsilon)^m$.

A single bad hypothesis will get caught if *any* data in the data set proves that it is bad. That happens with probability at least $1 - (1 - \epsilon)^m$.

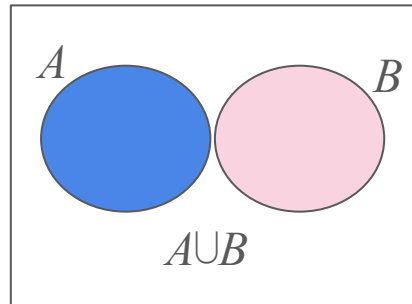
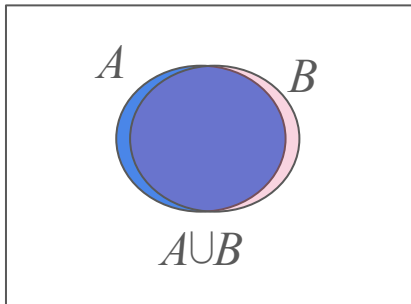
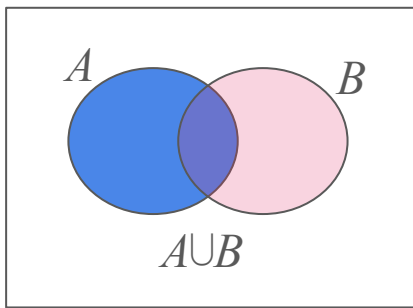
But, we want to enough data to weed out *all* bad hypotheses. And, there's no guarantee that hypotheses are iid or anything like that.

We need a trick to relate the probability of catching any one bad hypothesis to the probability of catching all of them.

Union Bound

For any two events A and B and any distribution \mathcal{D} ,

how does $\mathcal{D}(A \cup B)$ relate to $\mathcal{D}(A)$ and $\mathcal{D}(B)$?

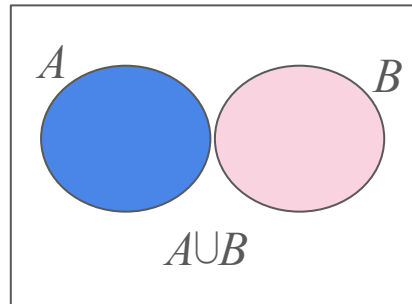
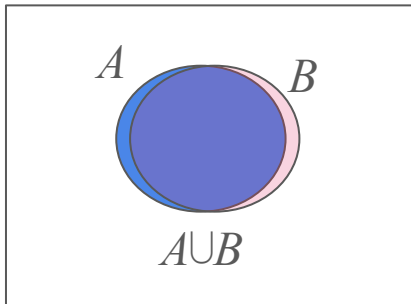
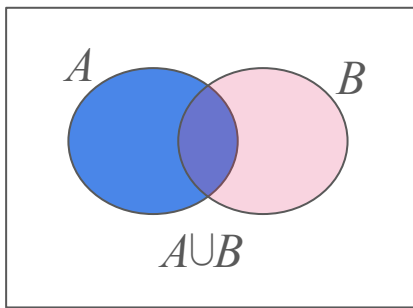


- A : Failed to catch bad hypothesis h_1 .
- B : Failed to catch bad hypothesis h_2 .
- $A \cup B$: At least one of these two bad hypotheses slipped through our fingers.

Union Bound

For any two events A and B and any distribution \mathcal{D} ,

$$\mathcal{D}(A \cup B) \leq \mathcal{D}(A) + \mathcal{D}(B).$$



- A : Failed to catch bad hypothesis h_1 .
- B : Failed to catch bad hypothesis h_2 .
- $A \cup B$: At least one of these two bad hypotheses slipped through our fingers.

Putting Pieces Together

A single bad hypothesis will evade capture m times in a row with probability at most $(1 - \epsilon)^m$.

If we have b bad hypotheses, the probability that any one of them slips through m times in a row is at most $b (1 - \epsilon)^m$ (union bound).

How many bad hypotheses can there be?

Putting Pieces Together

A single bad hypothesis will evade capture m times in a row with probability at most $(1 - \epsilon)^m$.

If we have b bad hypotheses, the probability that any one of them slips through m times in a row is at most $b (1 - \epsilon)^m$ (union bound).

How many bad hypotheses can there be? $|H|$

So, $\Pr(L_{\mathcal{D}}(h) > \epsilon) \leq |H| (1 - \epsilon)^m$. Since, $1 - \epsilon \leq e^{-\epsilon}$, $\Pr(L_{\mathcal{D}}(h) > \epsilon) \leq |H| e^{-\epsilon m}$

Finally, we want \square to bound this probability, $|H| e^{-\epsilon m} \leq \square$.

Solved for m , $m \geq \log(|H|/\square)/\epsilon$.

Algebraic Aside

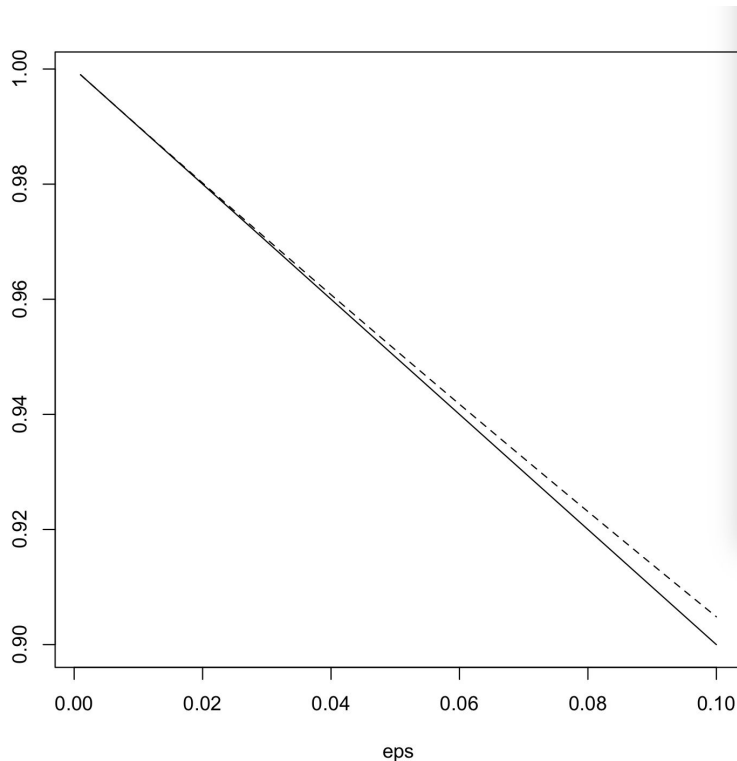
Note: $1 - \epsilon \leq e^{-\epsilon}$

Equal at $\epsilon=0$.

Derivatives: $-1 \leq -e^{-\epsilon}$

and $e^{-\epsilon} \leq 1$.

(Solid line is $1 - \epsilon$.)



Using PAC Result for the Lights Problem

We are guaranteed that the ERM hypothesis will be ϵ accurate with probability at least $1-\delta$ as long as we get at least m samples where: $m \geq \log(|H|/\delta)/\epsilon$.

$|H| = ?$

Using PAC Result for the Lights Problem

We are guaranteed that the ERM hypothesis will be ϵ accurate with probability at least $1 - \delta$ as long as we get at least m samples where: $m \geq \log(|H|/\delta)/\epsilon$.

$$|H| = 2^d$$

where d is the number of attributes. So, $m \geq \log(2^d/\delta)/\epsilon$.

Arbitrary Boolean Formulas

multicolored	textured	transparent	pointed	effective
yes	no	no	yes	<i>yes</i>
yes	no	no	no	<i>no</i>
yes	no	yes	yes	<i>no</i>
no	no	yes	yes	<i>no</i>
yes	no	yes	yes	<i>no</i>
no	yes	no	yes	<i>no</i>
yes	no	no	yes	<i>yes</i>
no	no	yes	yes	<i>no</i>

Best formula?

Arbitrary Boolean Formulas

multicolored	textured	transparent	pointed	effective
yes	no	no	yes	<i>yes</i>
yes	no	no	no	<i>no</i>
yes	no	yes	yes	<i>no</i>
no	no	yes	yes	<i>no</i>
yes	no	yes	yes	<i>no</i>
no	yes	no	yes	<i>no</i>
yes	no	no	yes	<i>yes</i>
no	no	yes	yes	<i>no</i>

multicolored and not textured and not transparent and pointed

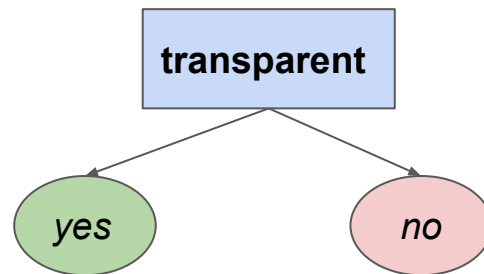
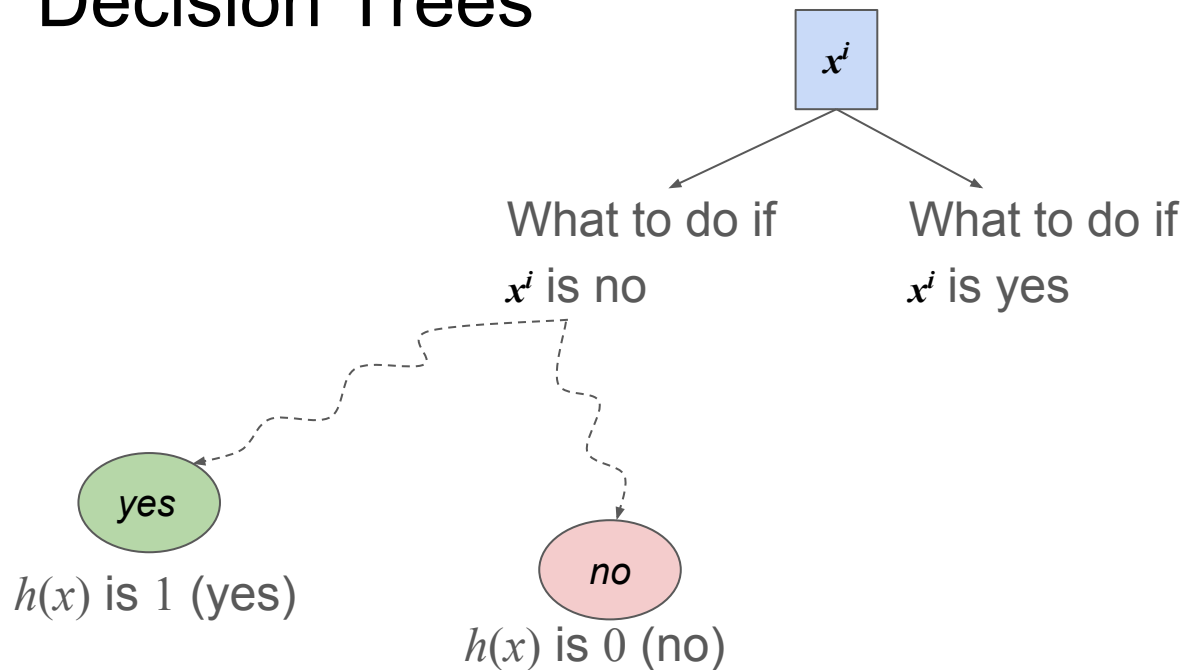
Overfitting

Consider the rule that reports the label corresponding to anything in the training set and guesses randomly otherwise.

True rule: $(x_1 \text{ and not } x_7) \text{ or } (x_2 \text{ and } x_{10})$.

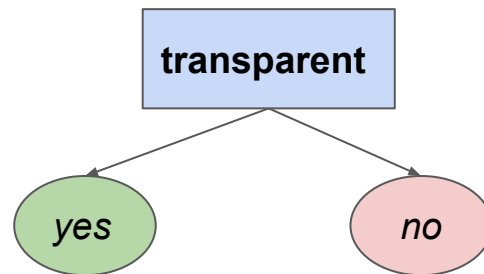
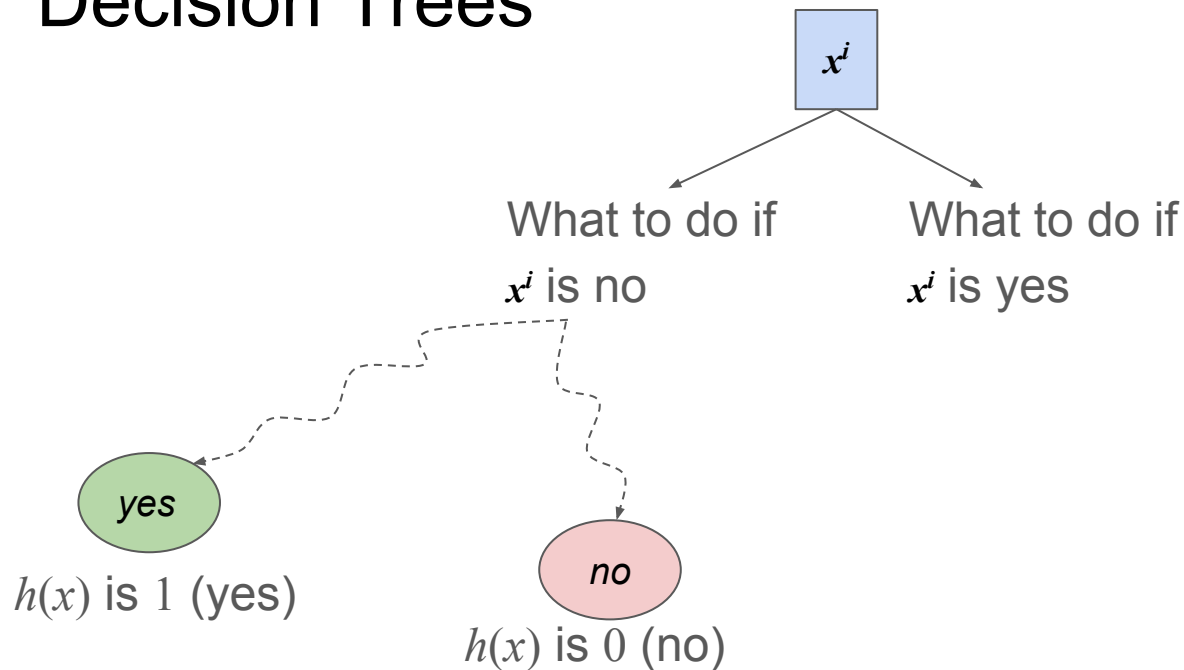
Empirical risk = 0. True risk = essentially chance (if m is small relative to 2^d).

Decision Trees



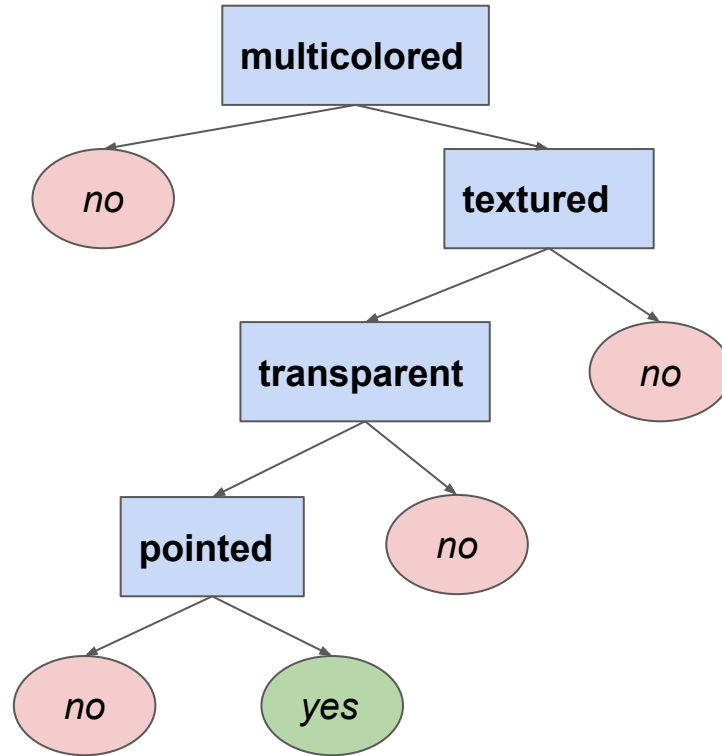
What rule is this?

Decision Trees



What rule is this? not transparent

Deeper Decision Tree



multicolored and not textured and not transparent and pointed

Representational Expressiveness

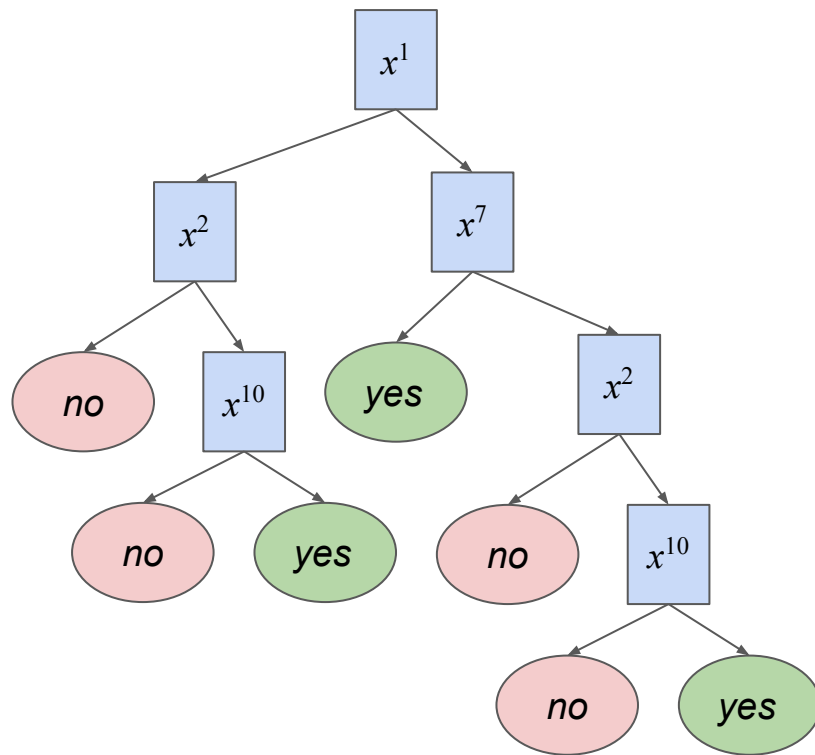
How many internal nodes to represent
(x_1 and not x_7) or (x_2 and x_{10}) ?

Representational Expressiveness

How many internal nodes to represent
(x_1 and not x_7) or (x_2 and x_{10}) ?

Six. Need to split on some nodes in
multiple places.

Generalizes single attribute splits and
conjunctions. Can represent any
function, but might get quite big to do it.



All Tree Search

Let H be the set of all possible decision trees:

- How compute ERM?
- What happens to generalization error?

All Tree Search

Let H be the set of all possible decision trees:

- How compute ERM? List all trees and pick best. Too many.
- What happens to generalization error? Poor. Overfits.


How about limiting to n internal nodes?

Can list them all if n is very small. Get generalize well if n is small.

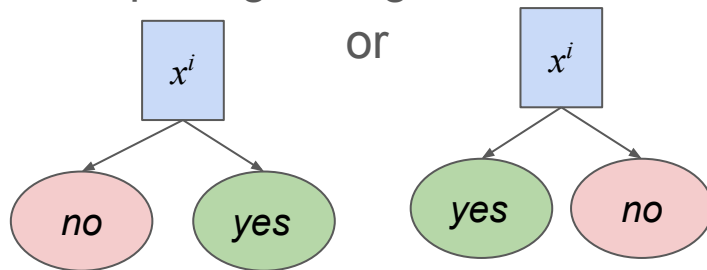
Greedy Search

We want a small tree that has low empirical risk.

Idea: Build the tree gradually, reducing empirical risk as fast as possible.

Start with the best possible 1-node tree. 

On each iteration, we examine the effect of splitting a single leaf into one of two trees for each possible attribute x^i .



Splitting rule 1: Greedily choose the split that reduces empirical risk the most.

Scoring a Tree

With respect to the training set S , a decision tree partitions each (x,y) in S into the leaves. Let l_S be the subset of S that reaches leaf l .

The training error is the sum of the training error in the leaves, weighted by the number of training examples in each leaf: $\sum_l \min_{v \in \{0,1\}} |\{(x,y) \in l_S \text{ s.t. } y \neq v\}|$.

Splitting a leaf doesn't change the score in other leaves. If we replace l with a split of x^i , we get two new leaves: $l_0 = \{(x,y) \in l_S \text{ s.t. } x^i = 0\}$ and $l_1 = \{(x,y) \in l_S \text{ s.t. } x^i = 1\}$.

Change:

$$\begin{aligned} \min_{v \in \{0,1\}} |\{(x,y) \in l_S \text{ s.t. } y \neq v\}| &- \min_{v \in \{0,1\}} |\{(x,y) \in l_S \text{ s.t. } y \neq v \text{ and } x^i = 0\}| \\ &- \min_{v \in \{0,1\}} |\{(x,y) \in l_S \text{ s.t. } y \neq v \text{ and } x^i = 1\}| \end{aligned}$$

Splits Visualized



Desiderata: ?

Splits Visualized



Training error: 2/8

Desiderata: homogeneity, balance

Alternative Splitting Rules

$$\text{Gain}(S, i) := C(P_S[y=1]) - (P_S[x_i=1] C(P_S[y=1|x_i=1]) + P_S[x_i=0] C(P_S[y=1|x_i=0]))$$

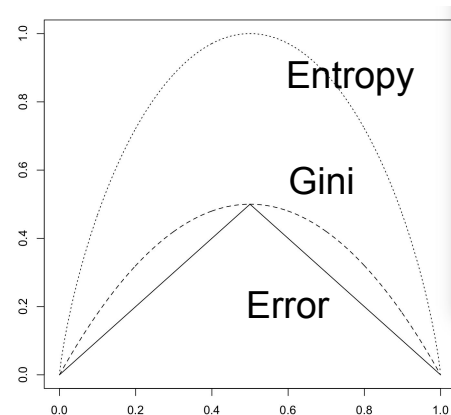
Train error: Empirical risk. $C(a) = \min(a, 1-a)$.

Information gain: The amount of information provided by the split.

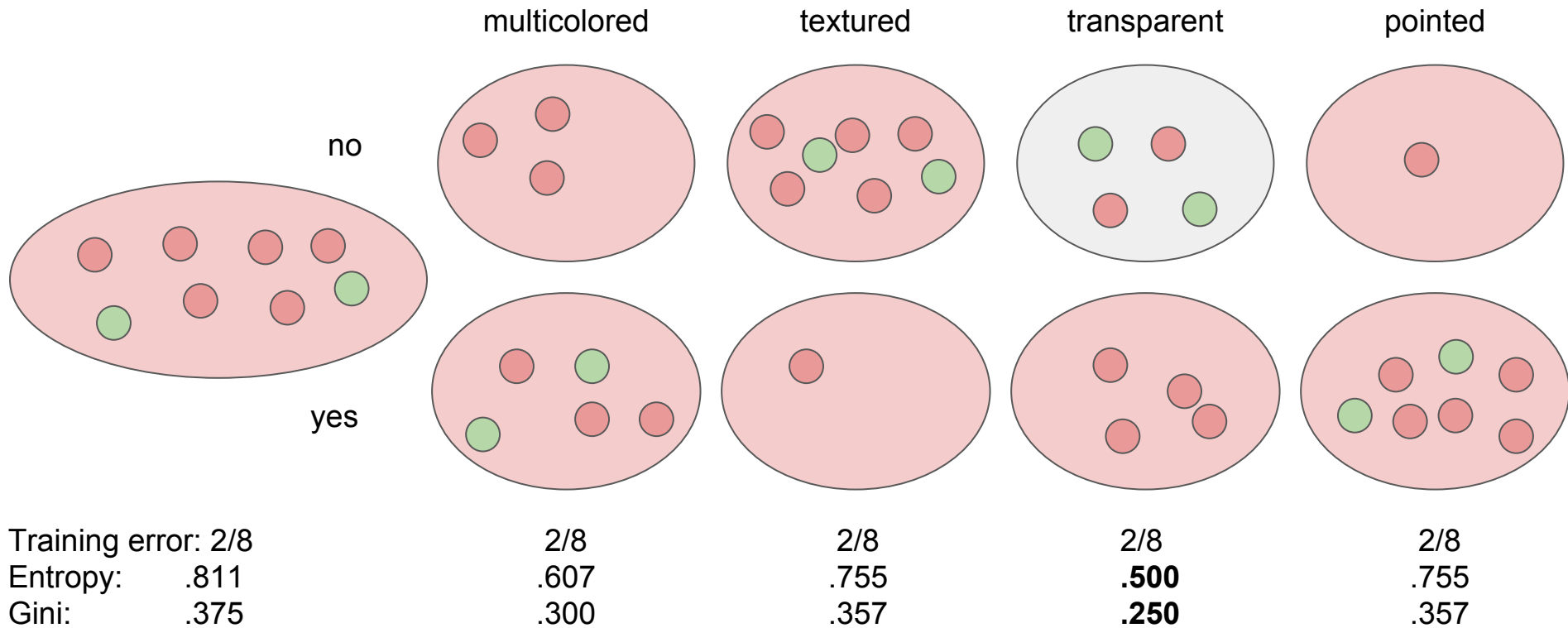
$$C(a) = -a \log_2(a) - (1-a) \log_2(1-a).$$

Gini index: Approximation of information without logs.

$$C(a) = 2 a (1-a).$$



Splits Visualized



Aside on Entropy

Big idea from Shannon: Given a distribution over outcomes, entropy is

$$-\sum_i p_i \log_2(p_i)$$

$-\log_2(p_i)$: minimum number of bits needed to convey an event with probability p_i .

Then, take the expectation over the set of events to get the average number of bits.

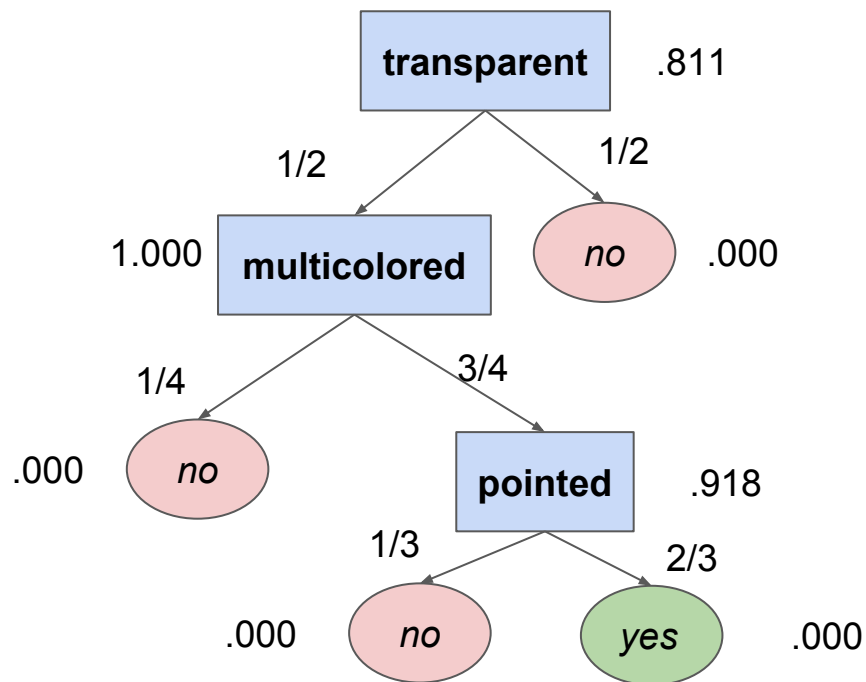
Surprisingly powerful concept because it provides a consistent way of dealing with conditional probabilities.

Optimized Tree Top Down

Keep splitting leaves using the rule until leaf is completely homogeneous or all attributes have been split on that branch.

Here (using entropy), we can get perfect classification.

Should you split when the splitting rule shows no improvement?



Stopping Early Can Get Stuck

Initial entropy: 1.

Splitting on **a**: 1.

Splitting on **b**: 1.

BUT, splitting on both **a** and **b** gets us to zero training error.

a	b	label
yes	no	yes
yes	yes	no
no	no	no
no	yes	yes

Not Stopping Overfits

The tree will grow until training error is zero or all splits have been made.

We're optimizing over the space of complete depth trees.

LOTS of data needed to get an accurate estimate.

Pruning: Best of Both Worlds

Build the whole tree, top down.

Chop off pieces, bottom up, that don't seem to helping much.

How do we measure “helping”? Hold out some data for *validation*!

Prune nodes (bottom up) whenever doing so improves classification performance on a held out validation set. It's the most common way to fight overfitting.

Alternatives to Greedy Tree Construction

Representation: n -node decision trees

Loss function: empirical risk

Optimizer: Greedy top-down construction

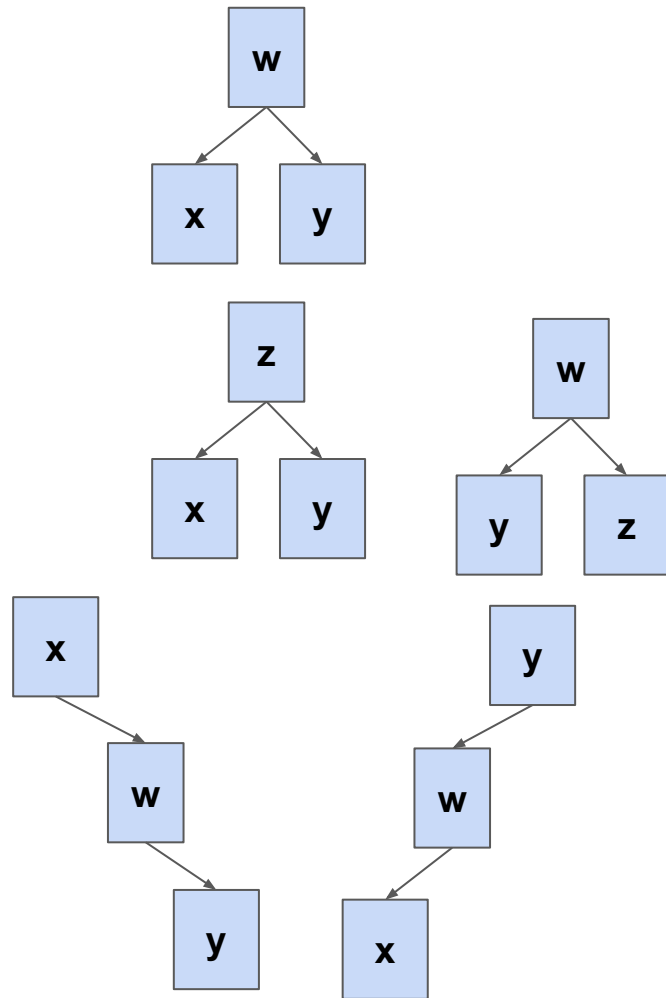
What else can we do?

Local Search

Build a random n -node decision tree.

Consider local improvements:

- Switch splitting attribute at a node.
- Change class of a leaf.
- Swap left and right subtrees of a node.
- “Rotate” at a node.
- All n -node trees are reachable in a small number of local moves.
- Make move if it improves accuracy.
- Restart if stuck.



Some Open Issues

How trade off tree size and training error in a provably justified way?

How justify the cross-validation approach? Does it really fit our general ML schema?