Generalization = predicting power.
Built a decision tree: how well will it do on future data?
Lead us to <u>PAC model</u>.

What is the "true error" or generalization error of a classifier?

- Decision trees: Fix T.

probability distribution D on new examples.

(x, y)

challenge label

$$\boxed{\Pr_{(x,y)\sim D}\left[T(x) \neq y\right]} \text{ of } T$$

True error / Generalization error.

Consider:

$$\text{"}\left|\begin{array}{c|c} x^1 & y^1 \\ \vdots & \vdots \\ x^m & y^m \end{array}\right| \text{"} = S$$

S training set.
Learner is Given S.
Learner memorizes S, and hands it back.
On new example, it just outputs 0.

Now consider:

$$S = \begin{array}{|c|c|} \multicolumn{1}{c}{X} & \multicolumn{1}{c}{Y} \\ \hline x^1 & y^1 \\ \vdots & \vdots \\ x^m & y^m \\ \hline \end{array}$$

training set

Each
$x^i \in \{0,1\}^n$, e.g. 011010 if $n = 6$.
$y^i \in \{0,1\}$

Let's build a decision tree.

Assume x^i are distinct.

You can build a decision tree (size $\geq |S|$)
that is consistent with all the points in S.

Question : How well does the tree generalize?
What is the true error of this tree?

How can we estimate the true error of a classifier (in this case a decision tree)?
use a "hold-out" or a "validation set", call it "H".

S = training set.
H = hold-out set.

1) use S to build a decision tree.
2) Estimate tree's true error via its error on H.

Cross-validation : a way to reuse the data that you've held-out into a hold-out set.

Another approach :
trade-off training error with "model complexity."
Define another potential function ϕ
ϕ : trees $\rightarrow \mathbb{R}$. Given a training set S

$$\phi(T) = \underbrace{\text{training error on } S}_{} + \alpha \cdot \frac{\text{size}(T)}{|S|}$$

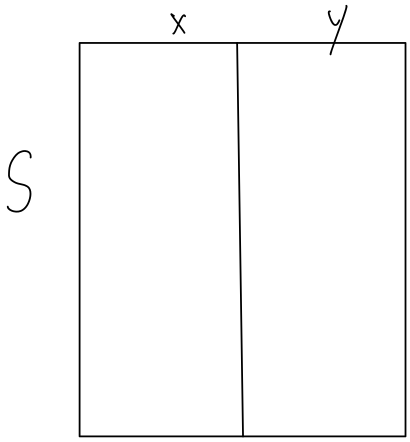
tree

hyperparameter.

size of training set.

Goal : minimize ϕ :
- min training error
  or
- min size of tree
  or
- increase $|S|$

Another approach: MDL / Minimum Description Length principle.



Some # of bits needed to encode S.

upper bound $m \cdot (n+1)$

label.

training examples

features

Build a tree T:

Let's say T is correct on 90% of S and incorrect on 10%.

We can encode S using # bits (T) and # bits to encode that 10% we got wrong.

This captures the notion of compression.