CSC 480/580 Principles of Machine Learning

01 Decision Trees

Jason Pacheco

THE UNIVERSITY OF ARIZONA

*some slides are from Daniel Hsu and Francesco Orabona with their permission

Example: course recommendation

- Build a software: given a student, recommend a set of courses that s/he would like

Examples:
- Is it a systems course?
- Is it a morning class?

course description
student info.

function
(=model)

rating $\in \{+, -\}$

Examples:
- which courses has taken before?
- likes morning class?

I'll explain
1. What kind of functions we'll be using
2. How to train one from data

Model: Decision Tree

Model: Decision Tree: Example

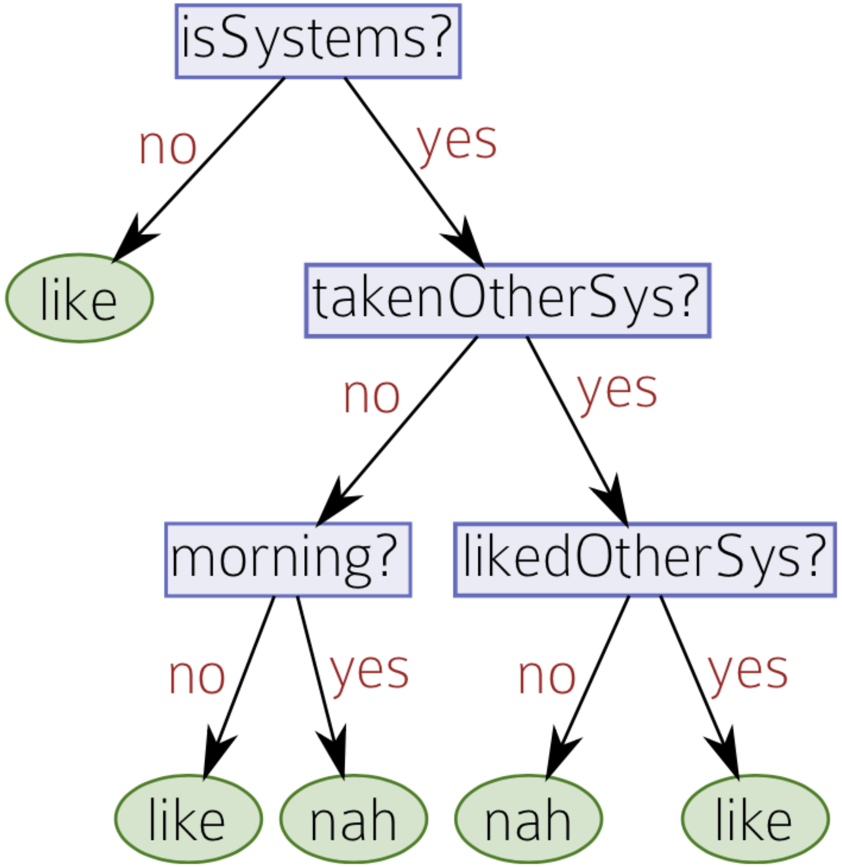


Figure 1.2: A decision tree for a course recommender system, from which the in-text "dialog" is drawn.

<u>**Input**</u>: the course & student info

Use questions to arrive at a conclusion.

Terminology:

- (Question, Answer) → (Feature, Feature Value)
- "Like" / "Nah" → Label
- {(A set of (Question & Answer)'s, Label)} → Train Data

Basic tree terminology

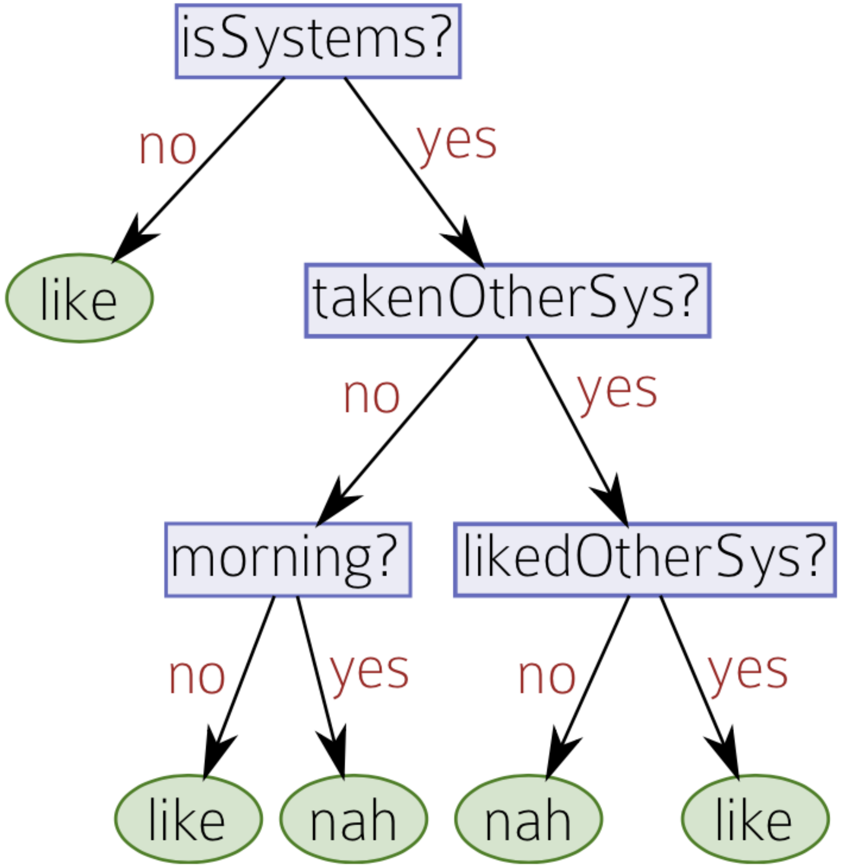


Figure 1.2: A decision tree for a course recommender system, from which the in-text "dialog" is drawn.

node
root node
leaf node
internal node

parent
children
ancestor
subtree
depth

Q: How many nodes are there?

Q: What's the depth of this tree?

- Key advantage of decision trees: *intepretability*
- Useful in consequential settings, e.g. medical treatment, loan approval, etc.

nodes organized in a tree-based structure, leading to a prediction (Fig. 1). The interpretability of decision trees allows physicians to understand why a prediction or stratification is being made, providing an account of the reasons behind the decision to subsequently accept or override the model's output. This interaction between humans and algorithms can provide

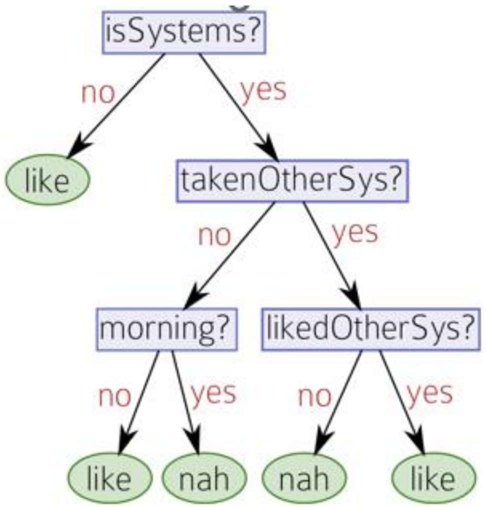
Prediction using a decision tree

- Test: predict using a decision tree:

test point: the data point to be classified
(vs train point: data point to be used for training)

Algorithm 2 DecisionTreeTest(*tree*, *test point*)

if *tree* is of the form Leaf(*guess*) **then**
    return *guess*
else if *tree* is of the form Node(f, *left*, *right*) **then**
    if $f = no$ in *test point* **then**
        return DecisionTreeTest(*left*, *test point*)
    else
        return DecisionTreeTest(*right*, *test point*)
    end if
end if



guess = prediction

left = no
right = yes

- Training: how to design a learning algorithm $\mathcal{A}$ that can build trees f from training data?

How to train

Train dataset

feature vector $\in \mathbb{R}^d$

label $\in \{+,-\}$

Define the labeled train data $S = \{(x_i, y_i)\}_{i=1}^n$

Features can be a function of the user being recommended; e.g., are you a morning person?

Features

Feature Values

Labels

(Labeled) Data Point

Rating	Easy?	AI?	Sys?	Thy?	Morning?
+2	y	y	n	y	n
+2	y	y	n	y	n
+2	n	y	n	n	n
+2	n	n	n	y	n
+2	n	y	y	n	y
+1	y	y	n	n	n
+1	y	y	n	y	n
+1	n	y	n	y	n
0	n	n	n	n	y
0	y	n	n	y	y
0	n	y	n	y	n
0	y	y	y	y	y
-1	y	y	y	n	y
-1	n	n	y	y	n
-1	n	n	y	n	y
-1	y	n	y	n	y
-2	n	n	y	y	n
-2	n	y	y	n	y
-2	y	n	y	n	n
-2	y	n	y	n	y

To make this a binary classification, we map

$\{+2,+1,0\} \Rightarrow$ "Liked" (+)

$\{-1,-2\} \Rightarrow$ "Nah" (-)

Background: Majority vote classifier

The most basic classifier you can think of.

How to train:

- Given: A (training) dataset with n data points $\{(\underbrace{x_i}_{\in \mathbb{R}^d}, \underbrace{y_i}_{\in \{1,\dots,C\}})\}_{i=1}^n$ with C classes.
- Compute the most common class c^* in the dataset.

$$c^* := \arg\max_{c \in \{1,\dots,C\}} \sum_{i=1}^n \mathbf{I}\{y_i = c\}$$

(break ties arbitrarily)

$$\mathbf{I}\{A\} := \begin{cases} 1 & \text{if } A \text{ is true} \\ 0 & \text{otherwise} \end{cases}$$

"indicator function"

- Output a classifier $f(x) = c^*$.

Stupid enough classifier! Always try to beat this classifier.

Often, state-of-the-art ML algorithms perform barely better than the majority vote classifier..

$\Rightarrow$ happens when there is no association between features and labels in the dataset

Background: Train set accuracy/error

- Suppose the ML algorithm has trained a function f using the dataset $D = \{(x_i, y_i)\}_{i=1}^n$
- Train set accuracy:

$$\widehat{\text{acc}}(f) := \frac{1}{n}\sum_{i=1}^{n} \mathbf{I}\{f(x_i) = y_i\}$$

- Train set error: $\widehat{\text{err}}(f) = \frac{1}{n}\sum_{i=1}^{n} \mathbf{I}\{f(x_i) \neq y_i\} = 1 - \widehat{\text{acc}}(f)$

- Q: We have 100 train set (images) consisting of 5 cats, 80 dogs, and 15 lions. What is the train set accuracy of the majority vote classifier? What is the error?

Training: The ideal criterion

- The training data $D = \{(x_i, y_i)\}_{i=1}^{n}$ $\quad x_i \in \{y, n\}^d \quad y_i \in \{+, -\}$

$$\hat{f} \coloneqq \arg \max_{f \in \mathrm{DecisionTrees}} \frac{1}{n} \sum_{i=1}^{n} \mathbf{I}\{f(x_i) = y_i\}$$

The main principle governing most of the ML algorithms.

It's called "**empirical risk minimization (ERM)**" (empirical risk = training set error)

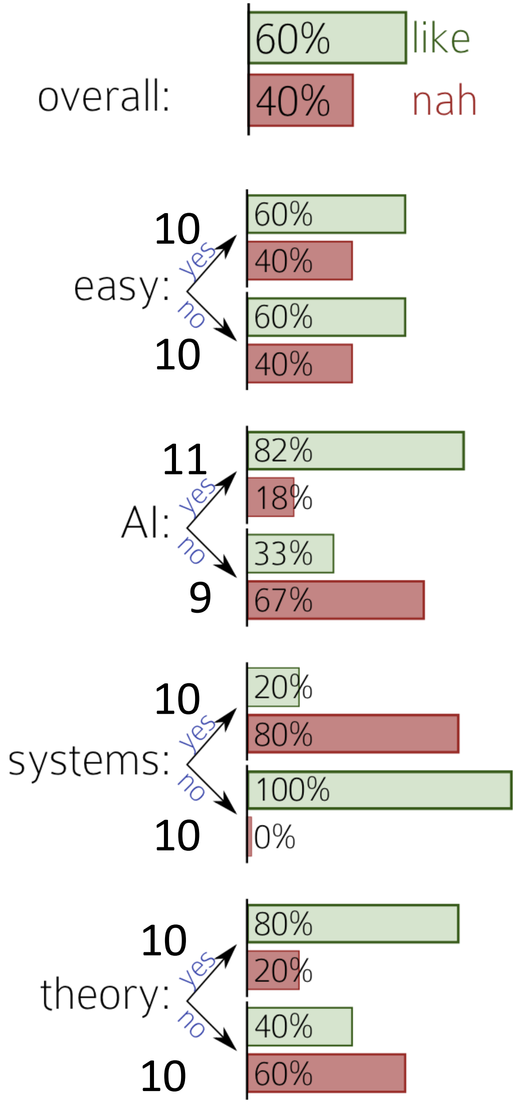
The issue:

- Naïve search: $O(d^d)$ time complexity
- It's NP-Hard -- don't expect to have an efficient algorithm.

Solution:

- Perform greedy approximation!

How to train a decision tree

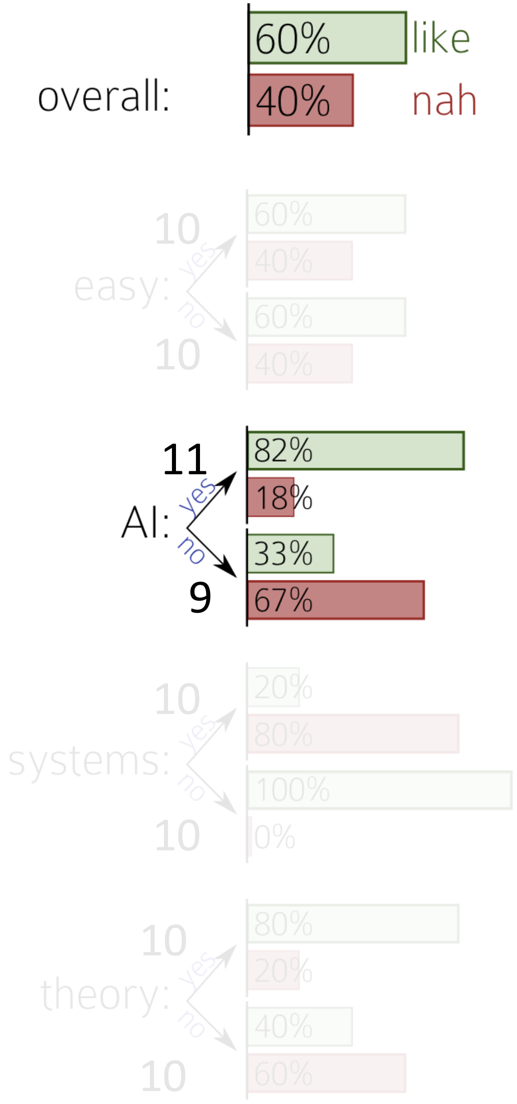


Rating	Easy?	AI?	Sys?	Thy?	Morning?
+2	y	y	n	y	n
+2	y	y	n	y	n
+2	n	y	n	n	n
+2	n	n	n	y	n
+2	n	y	y	n	y
+1	y	y	n	n	n
+1	y	y	n	y	n
+1	n	y	n	y	n
0	n	n	n	n	y
0	y	n	n	y	y
0	n	y	n	y	n
0	y	y	y	y	y
-1	y	y	y	n	y
-1	n	n	y	y	n
-1	n	n	y	n	y
-1	y	n	y	n	y
-2	n	n	y	y	n
-2	n	y	y	n	y
-2	y	n	y	n	n
-2	y	n	y	n	y

Baseline: ‘majority vote’ classifier

Q: What is the train set accuracy? 0.60

How to train a decision tree

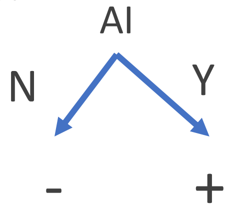


Rating	Easy?	AI?	Sys?	Thy?	Morning?
+2	y	y	n	y	n
+2	y	y	n	y	n
+2	n	y	n	n	n
+2	n	n	n	y	n
+2	n	y	y	n	y
+1	y	y	n	n	n
+1	y	y	n	y	n
+1	n	y	n	y	n
0	n	n	n	n	y
0	y	n	n	y	y
0	n	y	n	y	n
0	y	y	y	y	y
-1	y	y	y	n	y
-1	n	n	y	y	n
-1	n	n	y	n	y
-1	y	n	y	n	y
-2	n	n	y	y	n
-2	n	y	y	n	y
-2	y	n	y	n	n
-2	y	n	y	n	y

Baseline: ‘majority vote’ classifier

Q: What is the train set accuracy? 0.60

Suppose we place the node AI at the root. Let us set the prediction at each leaf node as the majority vote.

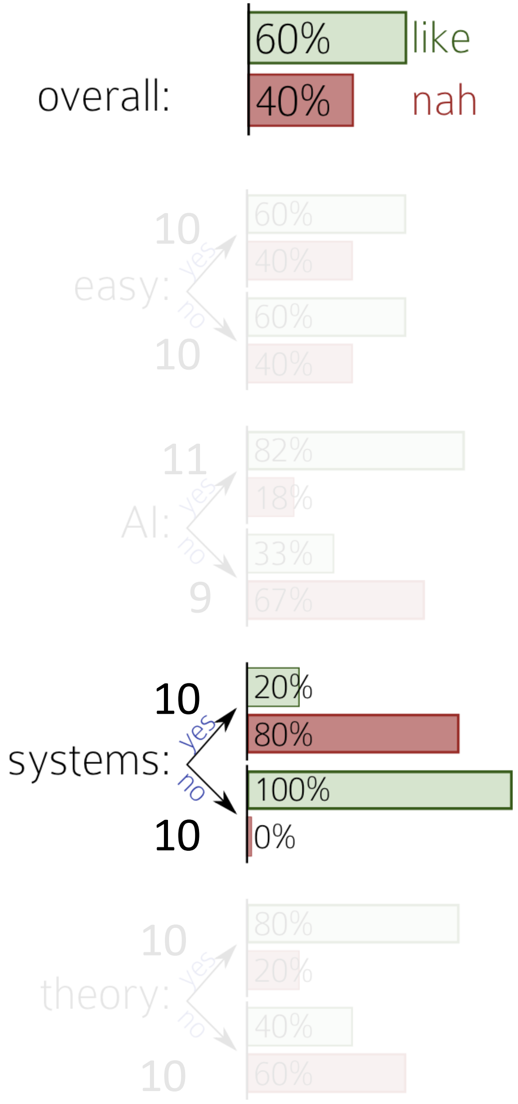


What is the train set accuracy now? Use weighted average.

$$\frac{9}{20}\cdot\frac{6}{9}+\frac{11}{20}\cdot\frac{9}{11}=\frac{15}{20}=0.75$$ improved!

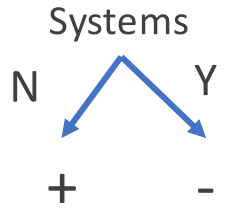
AI=N AI=Y

How to train a decision tree



Rating	Easy?	AI?	Sys?	Thy?	Morning?
+2	y	y	n	y	n
+2	y	y	n	y	n
+2	n	y	n	n	n
+2	n	n	n	y	n
+2	n	y	y	n	y
+1	y	y	n	n	n
+1	y	y	n	y	n
+1	n	y	n	y	n
0	n	n	n	n	y
0	y	n	n	y	y
0	n	y	n	y	n
0	y	y	y	y	y
-1	y	y	y	n	y
-1	n	n	y	y	n
-1	n	n	y	n	y
-1	y	n	y	n	y
-2	n	n	y	y	n
-2	n	y	y	n	y
-2	y	n	y	n	n
-2	y	n	y	n	y

Suppose placing the node Systems at the root.



What is the train set accuracy now?

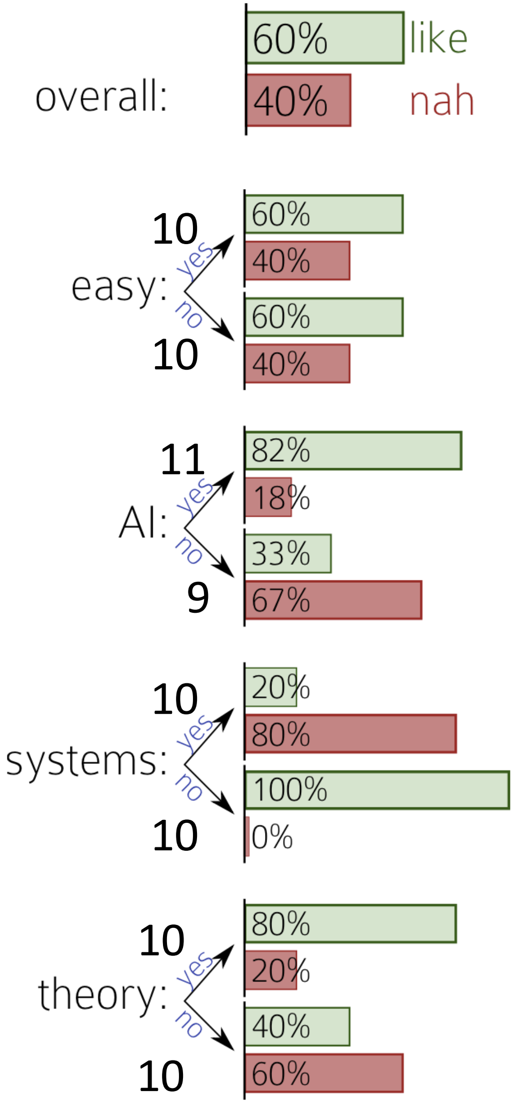
$$\frac{10}{20}\cdot\frac{10}{10}+\frac{10}{20}\cdot\frac{8}{10}=\frac{18}{20}=0.9$$

even better!

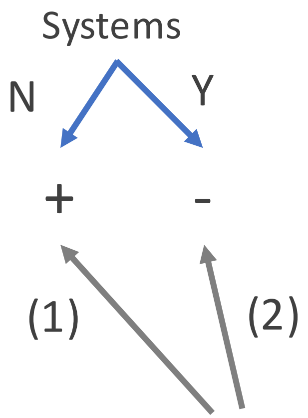
What would you do to build a depth-1 tree?

try out each feature and choose the one that leads to the largest accuracy!

How to train a decision tree



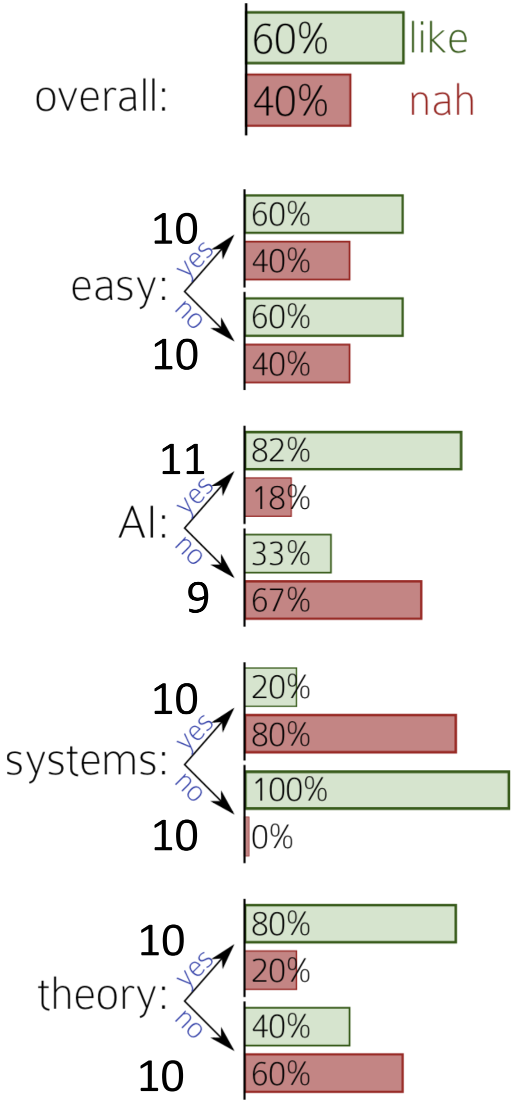
What about depth 2?

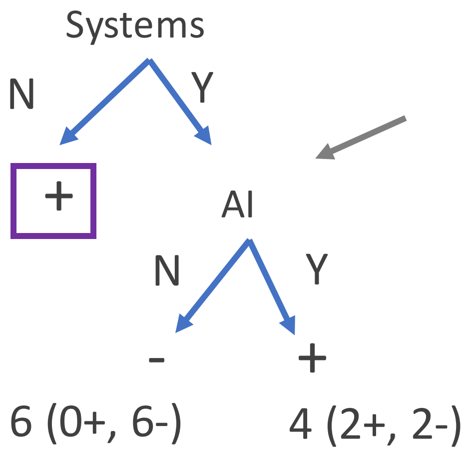


Which nodes to put at each leaf node?

Focus on (2). Try placing AI

How to train a decision tree





Q: How many training data points fall here? 10

Q: How many training data points arrive at these two leaves? How many for each label?

Q: what prediction should we use for each leaf?

Q: What is the train set accuracy, conditioning on Systems=Y?

'local' train set accuracy

$$\frac{6}{10}\cdot\frac{6}{6}+\frac{4}{10}\cdot\frac{2}{4}=\frac{8}{10}$$

Try all the other nodes and pick the one with the largest (local) acc.!

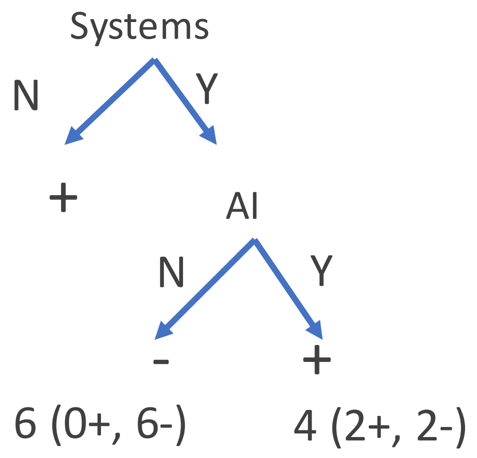
Then, repeat the same for Systems=N branch!

$\Rightarrow$ But this has 1.0 local train set acc. No need to expand anymore!

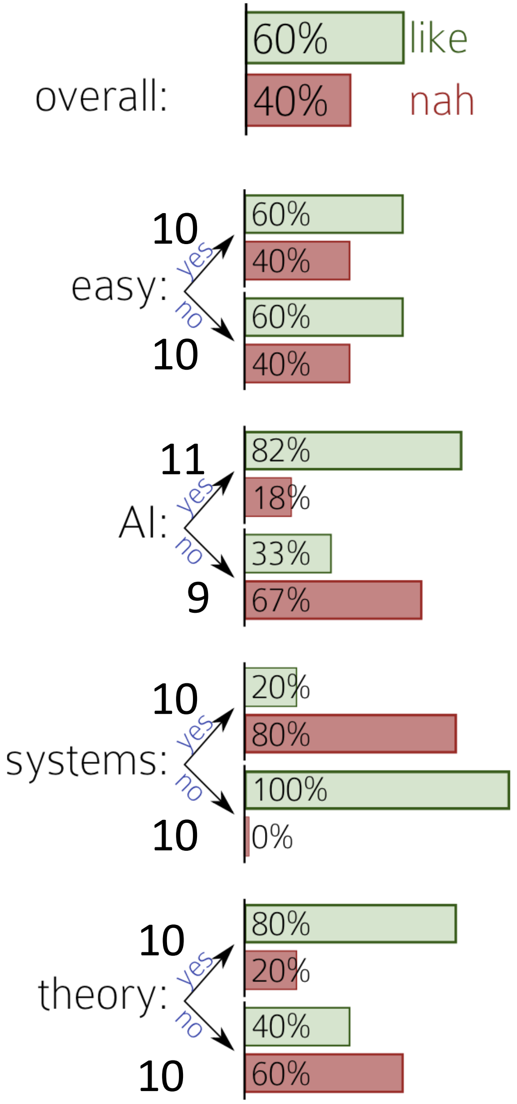
Move onto expanding nodes at depth 2!

How to train a decision tree

Rating	Easy?	AI?	Sys?	Thy?	Morning?
+2	y	y	n	y	n
+2	y	y	n	y	n
+2	n	y	n	n	n
+2	n	n	n	y	n
+2	n	y	y	n	y
+1	y	y	n	n	n
+1	y	y	n	y	n
+1	n	y	n	y	n
0	n	n	n	n	y
0	y	n	n	y	y
0	n	y	n	y	n
0	y	y	y	y	y
-1	y	y	y	n	y
-1	n	n	y	y	n
-1	n	n	y	n	y
-1	y	n	y	n	y
-2	n	n	y	y	n
-2	n	y	y	n	y
-2	y	n	y	n	n
-2	y	n	y	n	y



How to train a decision tree



Overall idea:

1. Set the root node as a leaf node.
2. Grab a leaf node for whose ‘local’ train accuracy is not 1.0.
3. Loop through features to find a feature f^* that maximizes the ’local’ train accuracy and replace the leaf node with a node with feature f^*; add its leaf nodes and set their predictions by majority vote.
   (note: skip the features used by an ancestor)
4. Repeat 2-3 until there is no more ‘expandable’ leaf node.

(i) local train acc. is not 1.0 <u>and</u>
(ii) ancestors did not use all the features yet

Algorithm 1 DecisionTreeTrain(*data*, *remaining features*)

```
guess ← most frequent answer in data              // default answer for this data
if the labels in data are unambiguous then
    return Leaf(guess)                            // base case: no need to split further
else if remaining features is empty then
    return Leaf(guess)                            // base case: cannot split further
else                                              // we need to query more features
    for all f ∈ remaining features do
        NO ← the subset of data on which f=no
        YES ← the subset of data on which f=yes
        score[f] ← # of majority vote answers in NO
                      + # of majority vote answers in YES
                          // the accuracy we would get if we only queried on f
    end for
    f ← the feature with maximal score(f)
    NO ← the subset of data on which f=no
    YES ← the subset of data on which f=yes
    left ← DecisionTreeTrain(NO, remaining features \ {f})
    right ← DecisionTreeTrain(YES, remaining features \ {f})
    return Node(f, left, right)
end if
```

guess=majority vote

unambiguous
= achieves 100% local acc. when
using the majority vote

has the same role as computing

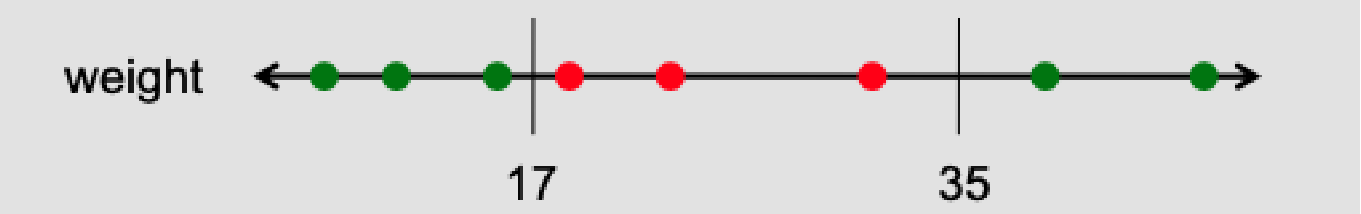
$$\frac{|NO|}{|YES| + |NO|}\,\widehat{acc}\,(NO) + \frac{|YES|}{|YES| + |NO|}\,\widehat{acc}\,(YES)$$

Type of features

- Binary
- Categorical: values in $\{1, \dots, C\}$ e.g., occupation, blood type
  - Option 1: Instead of 2 children, have C children.
  - Option 2: Derive C features of the form "feature=c?" for every $c \in C$.
    ↑ binary features!

  Q: How about features of the form "feature$\in D$" for every $D \subset C$? computational complexity ↑

- Real value e.g., weight, age, price
  - Sort the values.
  - Find the **breakpoints**: For every two adjacent points with opposite labels, compute the midpoint.
  - Derive features like "weight $\leq$ breakpoint"



Types of labels

- Binary
  - Accuracy is not sensitive to node purity…we will look at alternatives

- Multiclass: What changes do we need to make?
  - Almost none! Just extend the definition of accuracy to multiclass.

$$\widehat{\mathrm{acc}}(f) \coloneqq \frac{1}{n}\sum_{i=1}^{n} \mathbf{I}\{f(x_i) = y_i\}$$

- Real Value
  - This is a regression problem…we will get back to this

Variations: binary case

Notions of uncertainty: binary case ($\mathcal{Y} = \{0, 1\}$)

Suppose in a set of examples $S \subseteq \mathcal{X} \times \{0, 1\}$, a p fraction are labeled as 1

1. **Classification error**: (red)

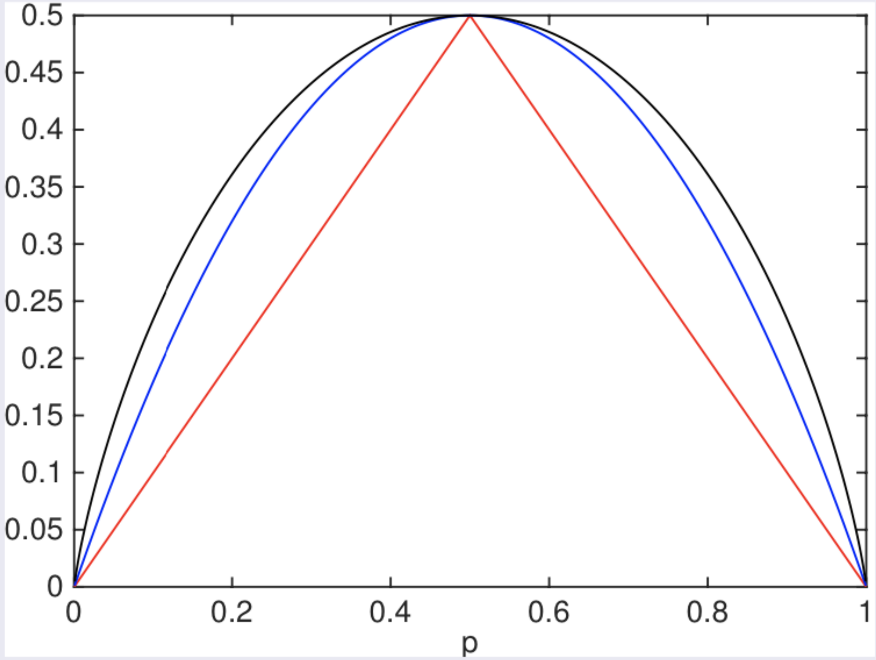
   $$u(S) := \min\{p,\ 1 - p\}$$

2. **Gini index**: (blue)

   $$u(S) := 2p(1 - p)$$

3. **Entropy**: (black)

   $$u(S) := p \log \frac{1}{p} + (1-p) \log \frac{1}{1-p}$$



Gini index and entropy (after some rescaling) are concave upper-bounds on classification error

"label uncertainty"

classification error here
= 1 – accuracy
(verify yourself)

Let q is the fraction of data points with feature=Y.

Modification:

Set score[f] as

$$q \cdot \big(-u(YES)\big) + (1 - q) \cdot (-u(NO))$$

If the number of classes is >2

Notions of uncertainty: general case

Suppose in $S \subseteq \mathcal{X} \times \mathcal{Y}$, a p_k fraction are labeled as k (for each $k \in \mathcal{Y}$).

1. **Classification error**:
$$u(S) := 1 - \max_{k \in \mathcal{Y}} p_k$$
2. **Gini index**:
$$u(S) := 1 - \sum_{k \in \mathcal{Y}} p_k^2$$
3. **Entropy**:
$$u(S) := \sum_{k \in \mathcal{Y}} p_k \log \frac{1}{p_k}$$

Each is *maximized* when $p_k = 1/|\mathcal{Y}|$ for all $k \in \mathcal{Y}$ (i.e., equal numbers of each label in S)

Each is *minimized* when $p_k = 1$ for a single label $k \in \mathcal{Y}$ (so S is **pure** in label)

Regression

Comparison: For classification

$$\widehat{err}(f) = \frac{1}{n}\sum_{i=1}^{n} \mathbf{I}\{f(x_i) \neq y_i\}$$

- Classification vs Regression
  - Both supervised learning
  - Regression has real-valued labels.
- Examples: Price prediction. Property value prediction.
- Standard measure of performance: mean squared error: $\frac{1}{n}\sum_{i=1}^{n}(f(x_i) - y_i)^2$

Q: why are we using squared error (f-y)^2 rather than absolute error |f-y|? my opinion: convenience & tradition

- Changes needed:
  - How to make predictions at the leaf node?

Average labels of the data at the leaf; denote by $\bar{y}_{YES}$ and $\bar{y}_{NO}$.

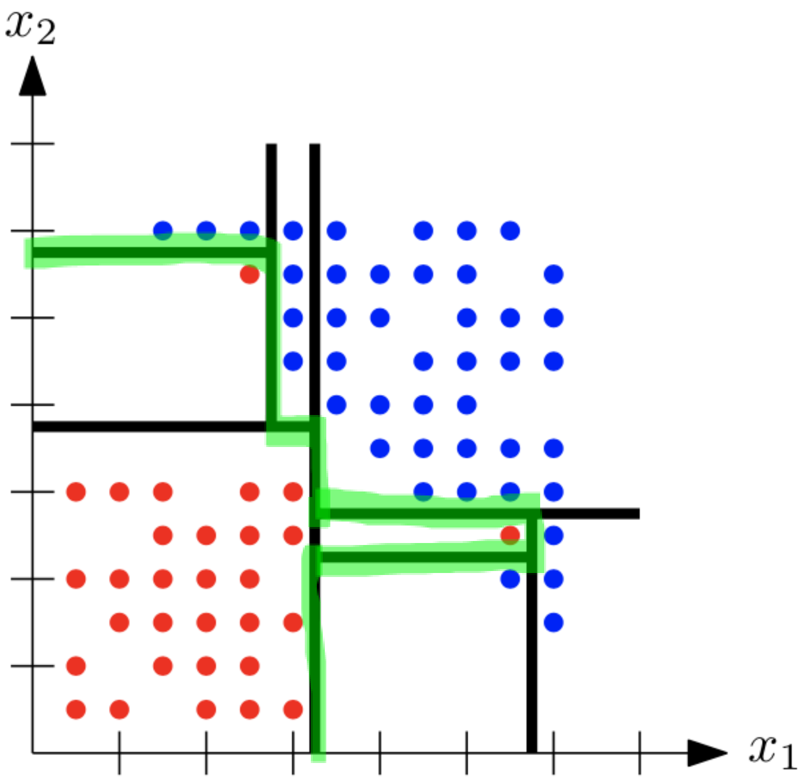
  - How to adjust score[f]?

Use negative squared error

$$\frac{|YES|}{|YES| + |NO|} \cdot \left(-\frac{1}{|YES|}\sum_{i \in YES}(\bar{y}_{YES} - y_i)^2\right) + \frac{|NO|}{|YES| + |NO|}\left(-\frac{1}{|NO|}\sum_{i \in NO}(\bar{y}_{NO} - y_i)^2\right)$$

(notations from the decision tree pseudocode)

“Spurious” patterns can be learned

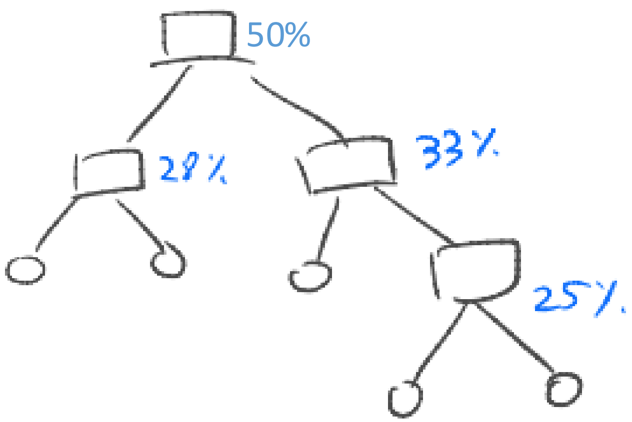


note axis-parallel decision boundaries

Unlearn spurious patterns by <u>**pruning**</u>

Split the given data into <u>**train set**</u> and <u>**validation set**</u>

- Build a decision tree based on the <u>**train set**</u>
- min_error ← compute the <u>**validation set**</u> error
- While true
  - For each <u>non-leaf node</u>, pretend that it is a leaf node and then compute the validation set error (but do not make it a leaf node yet)
  - current_error ← the smallest validation set error above.
  - If current_error ≥ min_error
    - Break
  - Else
    - **Prune** the one that reduces the validation set error the most
    - min_error ← current_error



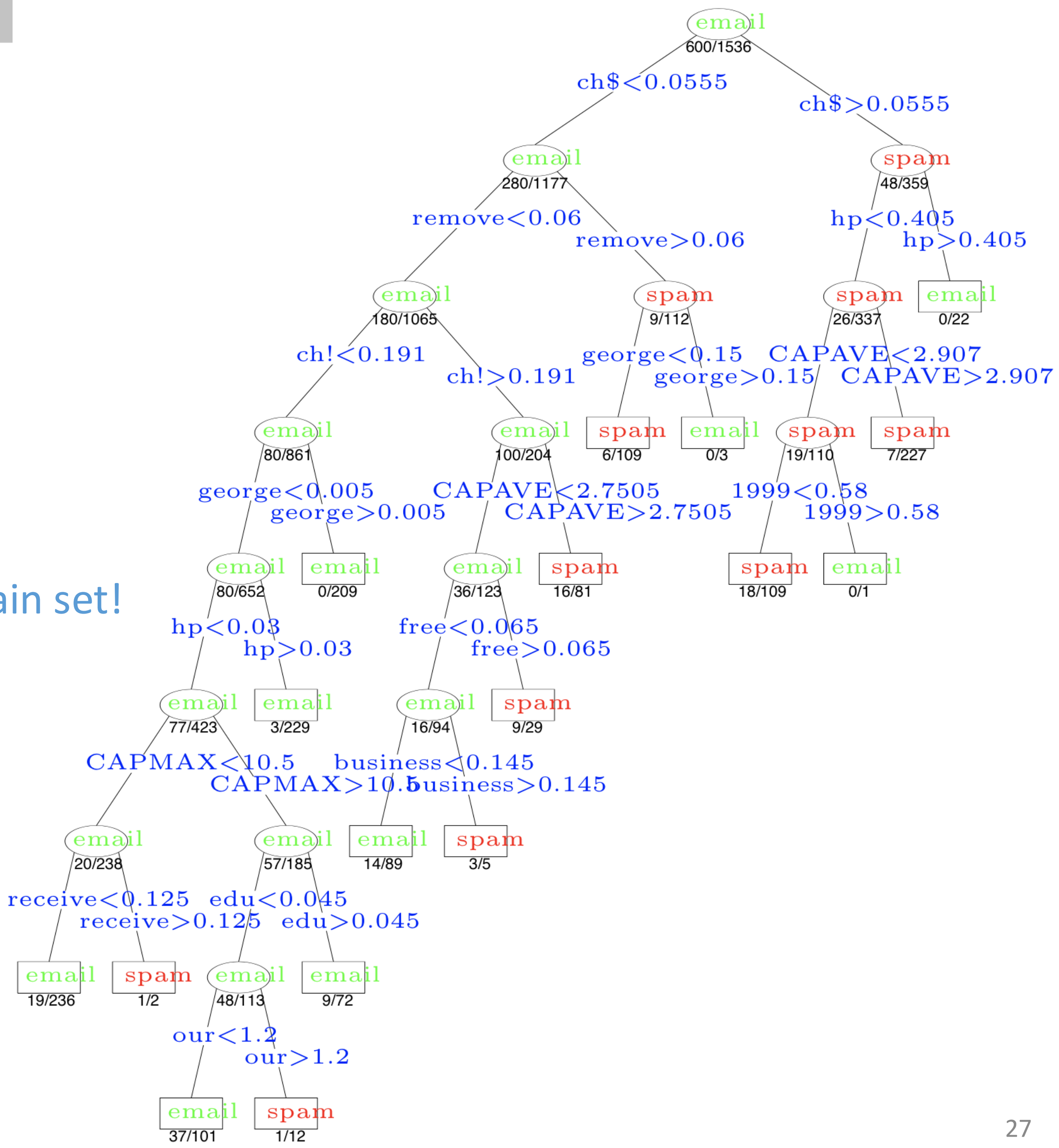
original validation set error: 35%

Example: spam filtering I

- Spam dataset
- 4601 email messages, about 39% are spam
- Classify message by spam and not-spam
- 57 features
  - 48 are of the form "percentage of email words that is (WORD)"
  - 6 are of the form "percentage of email characters is (CHAR)"
  - 3 other features (e.g., "longest sequence of all-caps")
- Final tree after pruning has 17 leaves, 9.3% test error rate

error rate computed on test set data
⇒ test set data should not have been part of the train set!

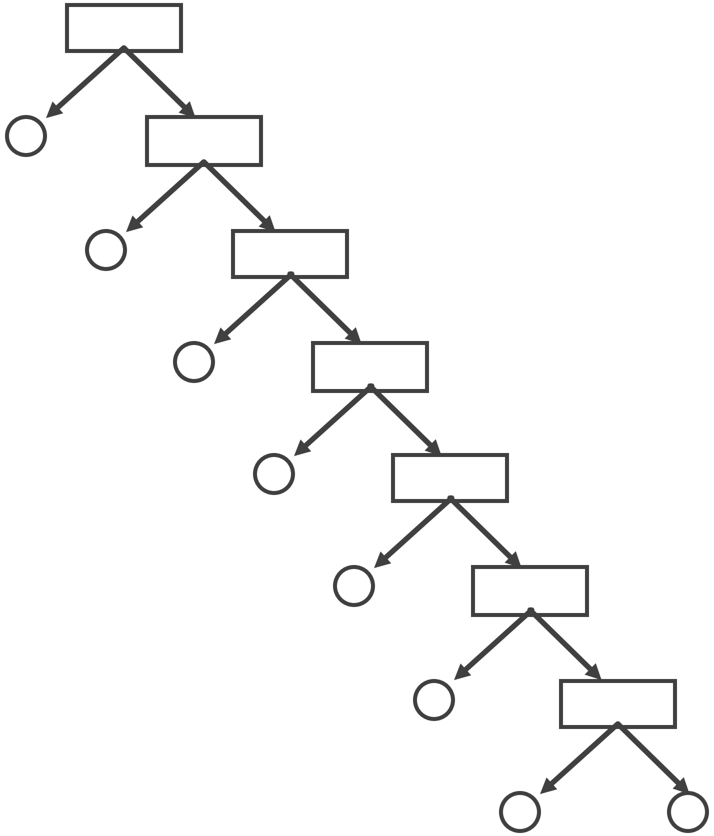
Q: what would be the majority vote accuracy?



Time complexity

- d: number of binary features, m: the number of data points

The worst-case configuration has $O(m)$ leaf nodes ⇒ O(m) internal node



⇒ Each internal node pays O(dm) for choosing which feature

⇒ Total: $O(dm^2)$