CS 5100: Foundations of Artificial Intelligence

Machine Learning (Naïve Bayes & Decision Trees)

Prof. Amy Sliva
November 17, 2011

Outline

- Introduction to learning
- Naïve Bayes classification (review)
- Naïve Bayes application
- Decision trees

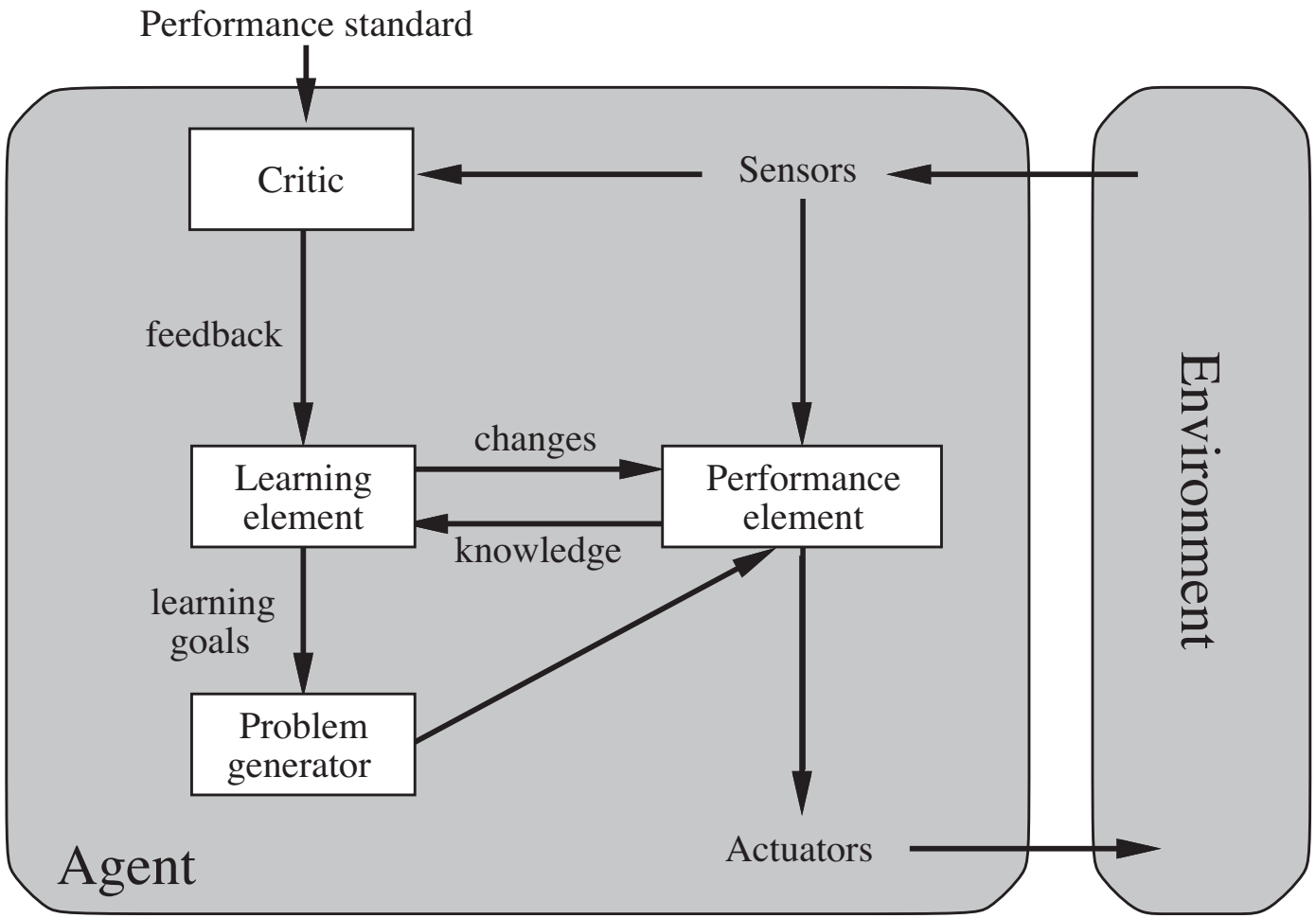
Learning in AI

- Learning is essential for unknown environments
  i.e., when the agent designer lacks **omniscience**

- Learning is useful as a **system construction** method
  i.e., expose the agent to reality rather than trying to write it down

- Learning modifies the agent's decision mechanisms to **improve performance**

Learning agents



Agents can be improved by learning

- Design of learning element is determined by
  1. Which **functional component** is to be improved
  2. **Prior knowledge** of the agent
  3. **Representation** used for data and the component
  4. **Feedback available** for learning

Learning paradigms

- Paradigms based on feedback

- Unsupervised learning (clustering algorithms)—**no feedback**
- Supervised learning—provide **correct answers** for each instance
- Reinforcement learning (genetic algorithms)—provide **occasional rewards** (or punishments)

- We will consider supervised learning, the most widely used and most successful paradigm
  - **Also the most expensive—why??**

Supervised learning methods

- Naïve Bayes
- Decision tree learning (ID3 and C4.5)
- Neural networks
- Support Vector Machines

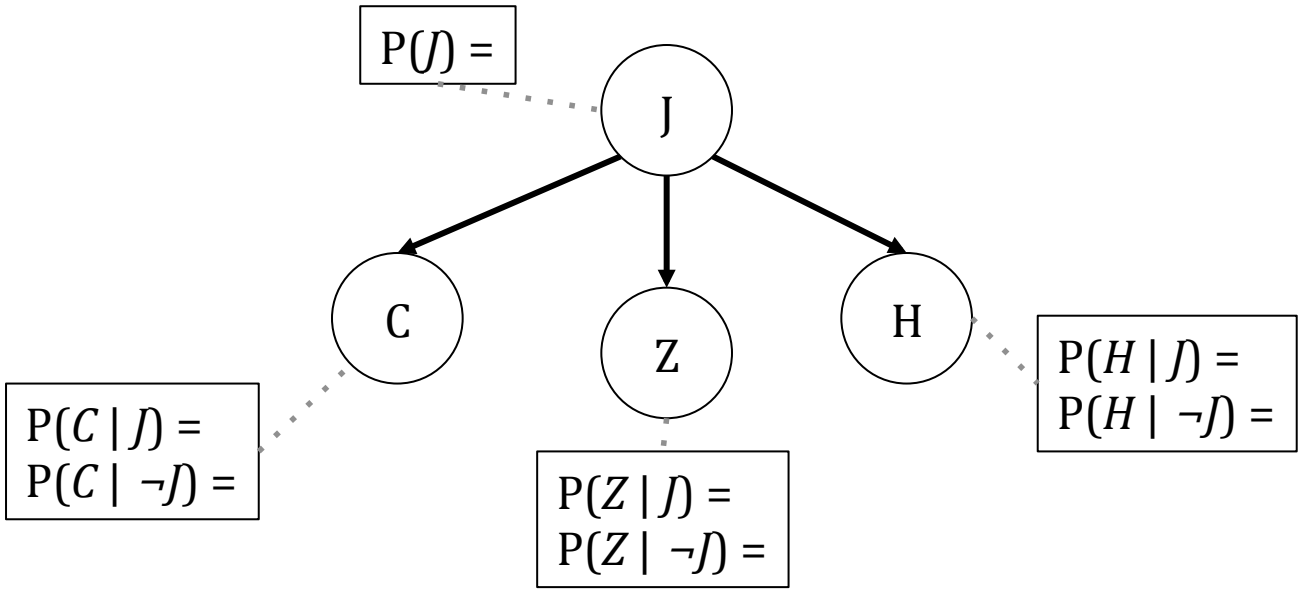
Naïve Bayes classifiers

- Our first example of machine learning!
- A supervised learning method
- Make (naïve) **independence assumption**
  - Can explore a simple subset of Bayesian nets s.t. it is easy to estimate the CPTs from sample data

- **Maximum likelihood estimation**
  - Given a set of correctly classified representative examples
  - **Q:** What estimates of conditional probabilities maximize the likelihood of the data that was observed?
  - **A:** The estimates that reflect the sample proportions

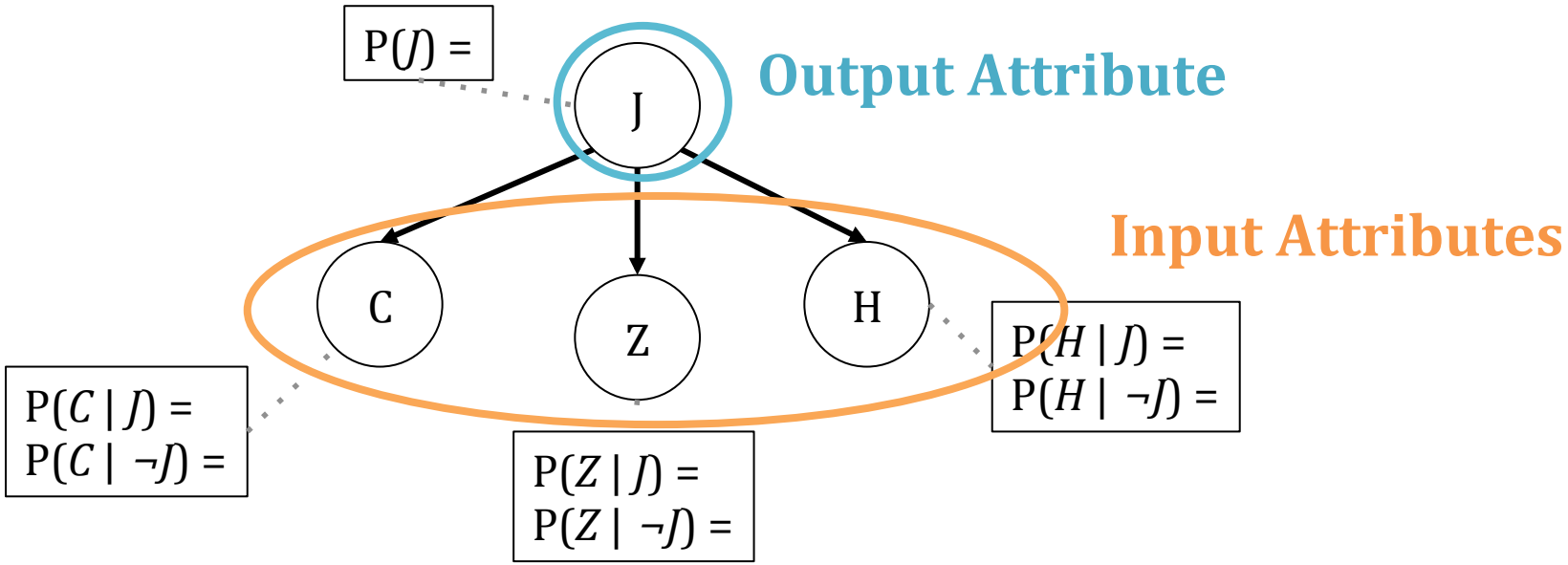
Start with a simple Bayes net

J: Person is a junior
C: Brought coat to classroom
Z: Lives in zip code 12345
H: Saw *Harry Potter* more than once



Naïve Bayes classifier

J: Person is a junior
C: Brought coat to classroom
Z: Lives in zip code 12345
H: Saw *Harry Potter* more than once

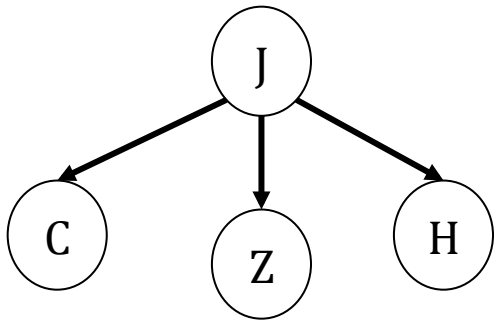


Inference with a Naïve Bayes classifier

- $P(J \mid C \wedge \neg Z \wedge H) = \dfrac{P(J \wedge C \wedge \neg Z \wedge H)}{P(C \wedge \neg Z \wedge H)}$

$$= \frac{P(J \wedge C \wedge \neg Z \wedge H)}{P(J \wedge C \wedge \neg Z \wedge H) + P(\neg J \wedge C \wedge \neg Z \wedge H)}$$

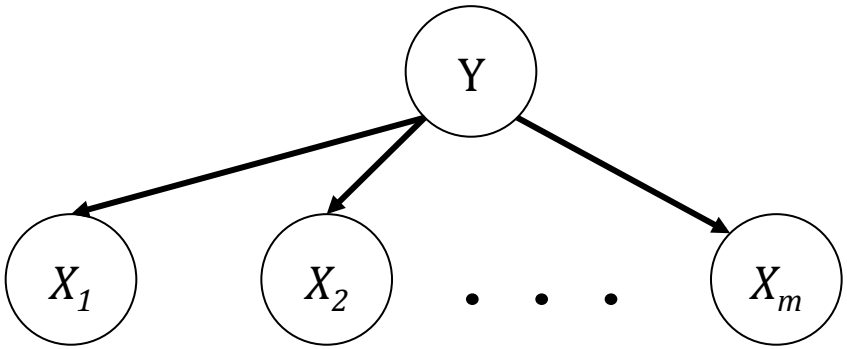
$$= \frac{P(C \mid J)\, P(\neg Z \mid J)\, P(H \mid J)\, P(J)}{\left(\begin{array}{l} P(C \mid J)\, P(\neg Z \mid J)\, P(H \mid J)\, P(J) + \\ P(C \mid \neg J)\, P(\neg Z \mid \neg J)\, P(H \mid \neg J)\, P(\neg J) \end{array}\right)}$$



Naïve Bayes—the general case

- Estimate $P(Y = v)$ as a fraction of records with $Y = v$
- Estimate $P(X_i = u_i \mid Y = v)$ as fraction of $Y = v$ records that also have $X_i = u_i$
- To **predict** the Y value given observations of all the X_i values, compute:

$$Y^{predict} = \underset{v}{argmax}\, P(Y = v \mid X_1 = u_1, \ldots, X_m = u_m)$$



- Assume features $X_1,\ldots,X_m$ are independent of each other given the class Y

Computing Naïve Bayes classification

$$Y^{predict} = \underset{v}{argmax}\, \mathrm{P}(Y = v \mid X_1 = u_1, \ldots, X_m = u_m)$$

- With **Bayes rule** we have

$$Y^{predict} = \underset{v}{argmax}\, \frac{\mathrm{P}(X_1 = u_1, \ldots, X_m = u_m \mid Y = v)\, \mathrm{P}(Y = v)}{\mathrm{P}(X_1 = u_1, \ldots, X_m = u_m)}$$

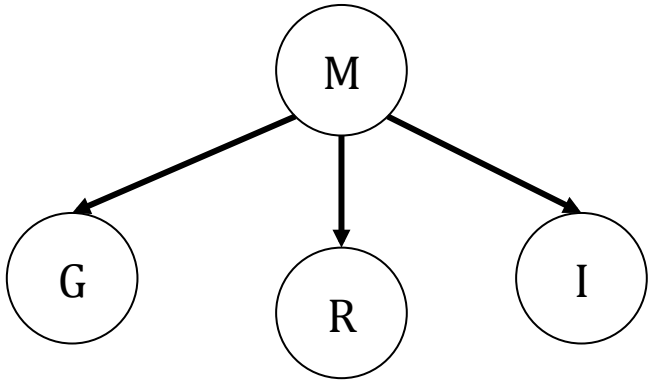
- We really only need the numerator to make a classification

$$= \underset{v}{argmax}\, \mathrm{P}(X_1 = u_1, \ldots, X_m = u_m \mid Y = v)\, \mathrm{P}(Y = v)$$

$$= \underset{v}{argmax}\, \mathrm{P}(X_1 = u_1 \mid Y = v) \ldots \mathrm{P}(X_m = u_m \mid Y = v)\, \mathrm{P}(Y = v)$$

$$= \underset{v}{argmax}\, \mathrm{P}(Y = v) \prod_i \mathrm{P}(X_i = u_i \mid Y = v)$$

Multi-valued naïve Bayes classification

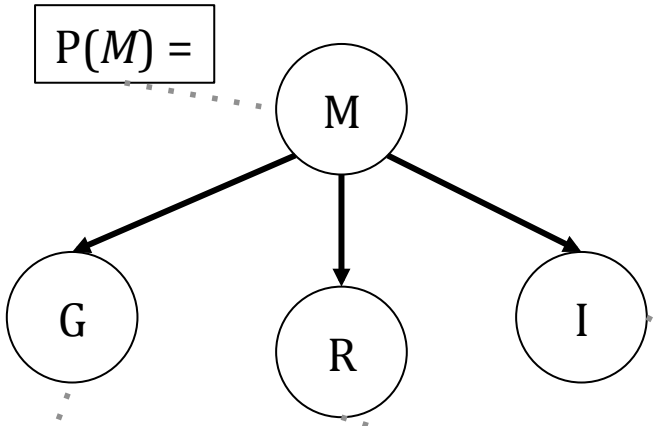


M: Major = Science, Arts, or Social Science

G: Gender = M or F
R: Race/Ethnicity = W, B, H, or A
I: Is international student

- **What do the conditional probability tables look like?**

Multi-valued naïve Bayes classification



M: Major = Science, Arts, or Social Science

G: Gender = M or F
R: Race/Ethnicity = W, B, H, or A
I: Is international student

P(*G* = *M* | *M* = *Science*) =
P(*G* = *M* | *M* = *Arts*) =
P(*G* = *M* | *M* = *SS*) =
P(*G* = *F* | *M* = *Science*) =
P(*G* = *F* | *M* = *Arts*) =
P(*G* = *F* | *M* = *SS*) =

P(*R* = *W* | *M* = *Science*) =
P(*R* = *W* | *M* = *Arts*) =
P(*R* = *W* | *M* = *SS*) =
. . .
P(*R* = *A* | *M* = *Science*) =
P(*R* = *A* | *M* = *Arts*) =
P(*R* = *A* | *M* = *SS*) =

P(*I* | *M* = *Science*) =
P(*I* | *M* = *Arts*) =
P(*I* | *M* = *SS*) =

- **What do the conditional probability tables look like?**

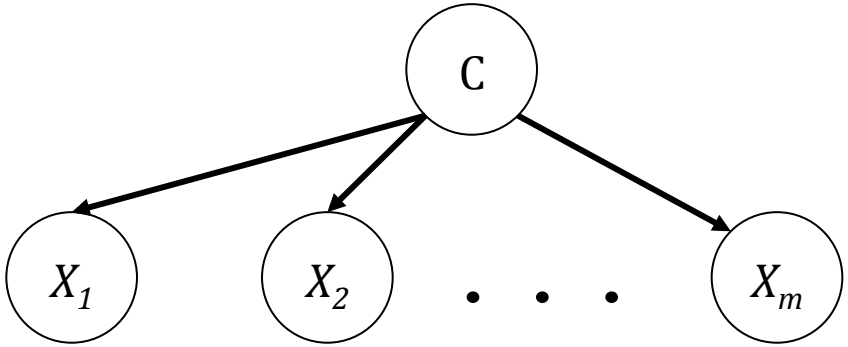
Naïve Bayes Algorithm

- **Learn:** Input = Data Set,
  Output = For each class c_j: Estimate $\hat{P}(c_j)$
  - For each attribute value x_i of each attribute X_i
    - Estimate $\hat{P}(x_i \mid c_j)$

- **Classify:** new instance <x1,…,xn> as

$$c^{pred} = \underset{c}{argmax}\ \hat{P}(c) \prod_i \hat{P}(x_i \mid c)$$

Learning the model

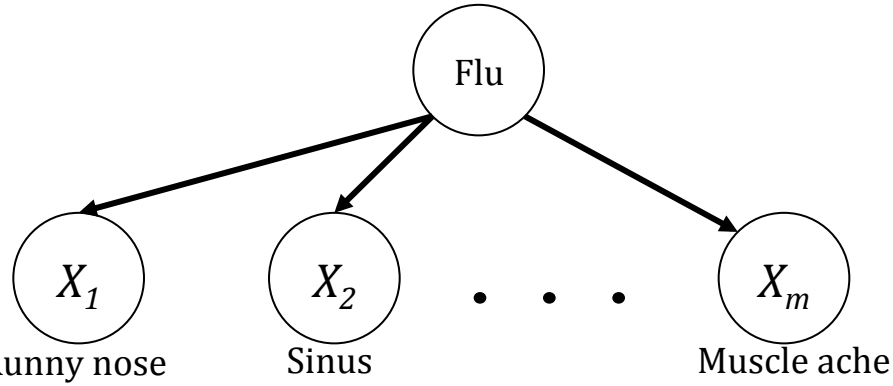


- Common practice—**maximum likelihood**
  - Find model s.t. likelihood of data is maximized
  - Simply use the frequencies in the data

$$\hat{P}(c_j) = \frac{N\,(C = c_j)}{N}$$

$$\hat{P}(x_i \mid c_j) = \frac{N\,(X_i = x_i, C = c_j)}{N(C = c_j)}$$

Problem with max likelihood



$$P(X_1,\ldots,X_m \mid C) = P(X_1 \mid C)\, P(X_2 \mid C) \ldots P(X_m \mid C)$$

- What if we have seen no training cases where patient had no flu and muscle aches?

$$\hat{P}(X_m = t \mid C = nf) = \frac{N(X_m = t, C = nf)}{N(C = nf)} = 0$$

- **Zero probabilities** cannot be conditioned away, no matter the other evidence!

Smoothing to avoid overfitting

- Assume at least one case of everything

$$\hat{\mathrm{P}}(x_i \mid c_j) = \frac{\mathrm{N}\,(X_i = x_i, C = c_j) + 1}{\mathrm{N}(C = c_j) + k}$$

- Somewhat more subtle version

$$\hat{\mathrm{P}}(x_{i,k} \mid c_j) = \frac{\mathrm{N}\,(X_i = x_{i,k}, C = c_j) + mp_{i,k}}{\mathrm{N}(C = c_j) + m}$$

Smoothing to avoid overfitting

- Assume at least one case of everything

$$\hat{P}(x_i \mid c_j) = \frac{N(X_i = x_i, C = c_j) + 1}{N(C = c_j) + k}$$

# of values of X_i

- Somewhat more subtle version

$$\hat{P}(x_{i,k} \mid c_j) = \frac{N(X_i = x_{i,k}, C = c_j) + mp_{i,k}}{N(C = c_j) + m}$$

Overall fraction in data where $X_i = x_{i,k}$

Extent of "smoothing"

Why does conditional independence work?

- Conditional independence assumption is typically **false**
  - Sinus condition not independent of runny nose, even given flu

- Nevertheless, it works surprisingly well!
  - **Reason 1: small number of parameters**
    If we try to fit too many parameters with sparse data, we'll get really strange models
  - **Reason 2: don't need probabilities to be correct**, only the classification (i.e., *argmax*)

$$c^{pred} = \underset{c}{argmax}\ \hat{\mathrm{P}}(c) \prod_i \hat{\mathrm{P}}(x_i|\ c) = \underset{c}{argmax}\ \mathrm{P}(c) \prod_i \mathrm{P}(x_i|\ c) = c$$

Naïve Bayes for text classification

- **Input:** document consisting of words
- **Output:** classification into a set of classes

- Examples
  - Learn which news articles are "interesting"
  - Learn to classify webpages by topic
  - **Learn the sentiment of blogs, news articles, and reviews**

- Naïve Bayes is surprisingly good at this task

First possible model

- Multi-variate binomial
  - One feature X_w for each word in dictionary
  - X_w = *true* in document *d* if *w* appears in *d*

  - Naïve Bayes assumption:
    - Given the document's topic, appearance of one word in a document tells us nothing about chances that another word appears
      (i.e., **word occurrence is conditionally independent given the topic**)

Second possible model

- Multinomial
  - One feature X_i for each word in document
    - Feature values are all words in dictionary
  - Value of X_i is the word in position i

  - Naïve Bayes assumption
    - Given the document's topic, word in one position in document tells us nothing about value of words in other positions
      (i.e., **words/positions are conditionally independent given the topic**)
  - Second assumption
    - Word appearance does not depend on position

      $P(X_i = w \mid c) = P(X_j = w \mid c)$ for all positions i, j, word w, and class c

Parameter estimation

- Binomial model:

$$\hat{P}(X_w = t \mid c_j) = \text{fraction of documents of topic } c_j \text{ in which word } w \text{ appears}$$

- Multinomial model:

$$\hat{P}(X_i = w \mid c_j) = \text{fraction of times in which word } w \text{ appears across all documents of topic } c_j$$

  - Creating a mega-document for topic j by concatenting all documents on this topic
  - Use frequency of w in mega-document

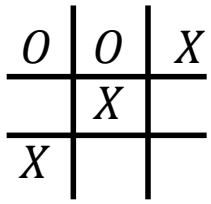
Sentiment analysis demo!

- Sentiment Analysis with Python NLTK Text Classification
  - NLTK = Natural Language Toolkit
  - http://text-processing.com/demo/sentiment/

- Trained on movie reviews
  - How does it do on reviews from Rotten Tomatoes?
  - What about less "well-formed" reviews (i.e., Yelp)

- Can use these tools to customize content on web pages (e.g., Amazon recommendations), determine market opinion (e.g., advertising), etc.

Inductive learning (a.k.a. Science)

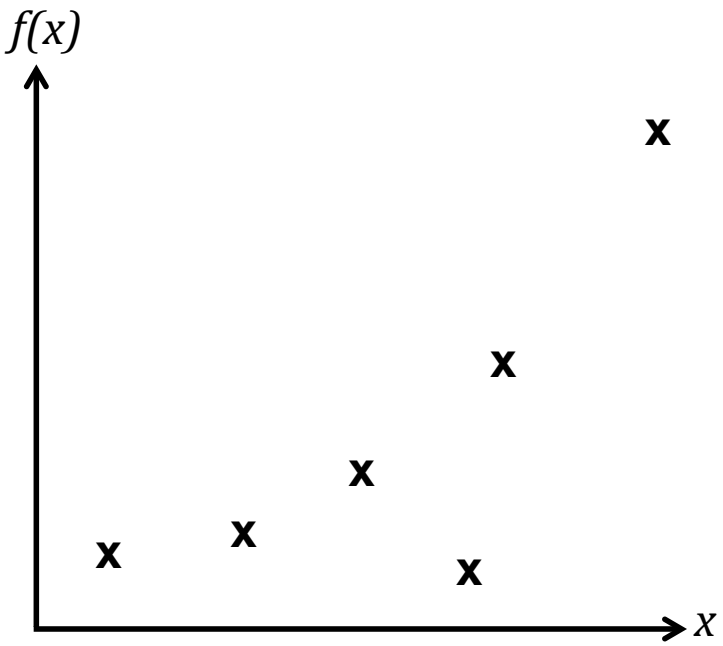
- Simplest form—learn a function from examples (**tabula rasa**)
- f is the **target function**
- An **example** is a pair $x, f(x)$, e.g.,  , +1
- Problem: find hypothesis ***h*** s.t. $h \approx f$ given a **training set** of examples

- This is a highly simplified model of real learning:
  - Ignores prior knowledge
  - Assumes a deterministic, observable "environment"
  - Assumes examples are given
  - Assumes that the agent wants to learn f

Inductive learning method

- Construct/adjust *h* to agree with *f* on training set (*h* is **consistent** if it agrees with *f* on all examples)

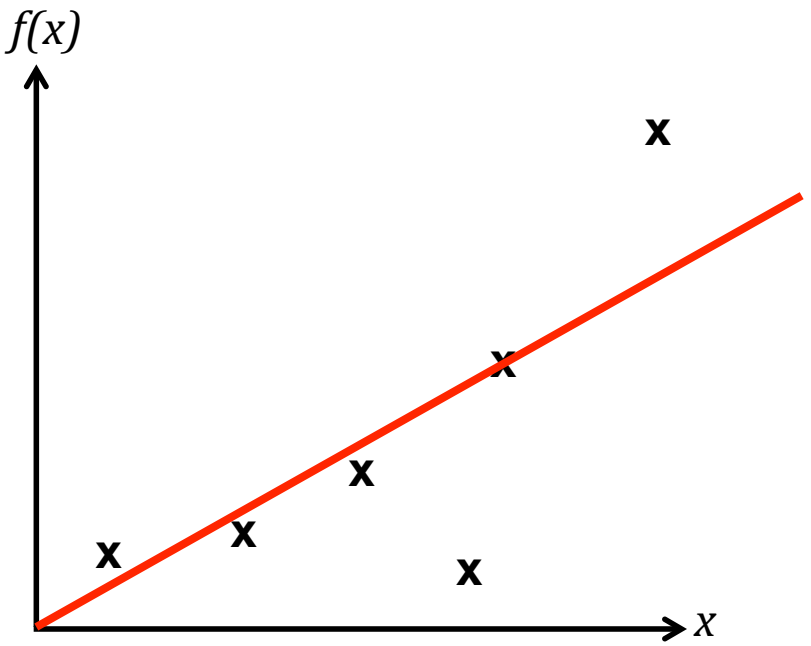
  e.g., curve fitting:



Inductive learning method

- Construct/adjust *h* to agree with *f* on training set (*h* is **consistent** if it agrees with *f* on all examples)

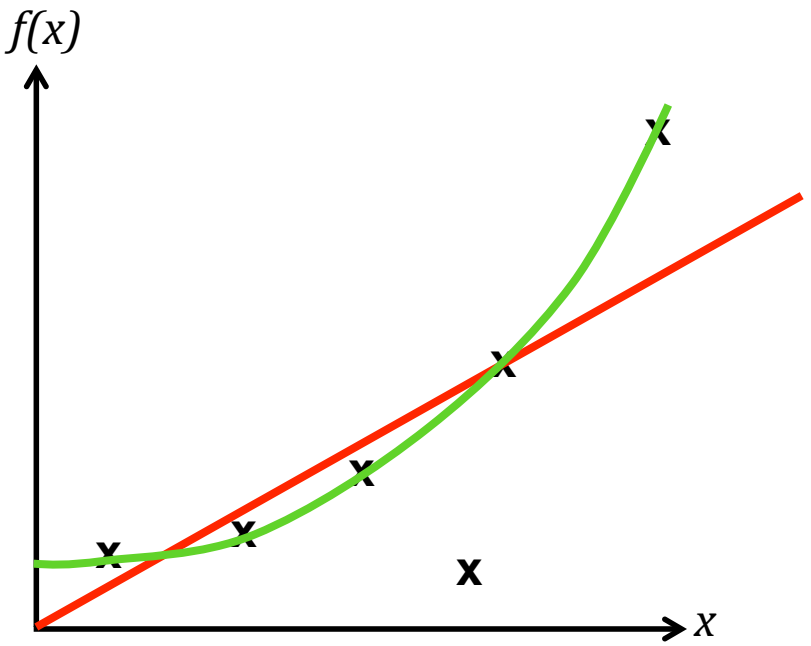
e.g., curve fitting:



Inductive learning method

- Construct/adjust *h* to agree with *f* on training set (*h* is **consistent** if it agrees with *f* on all examples)

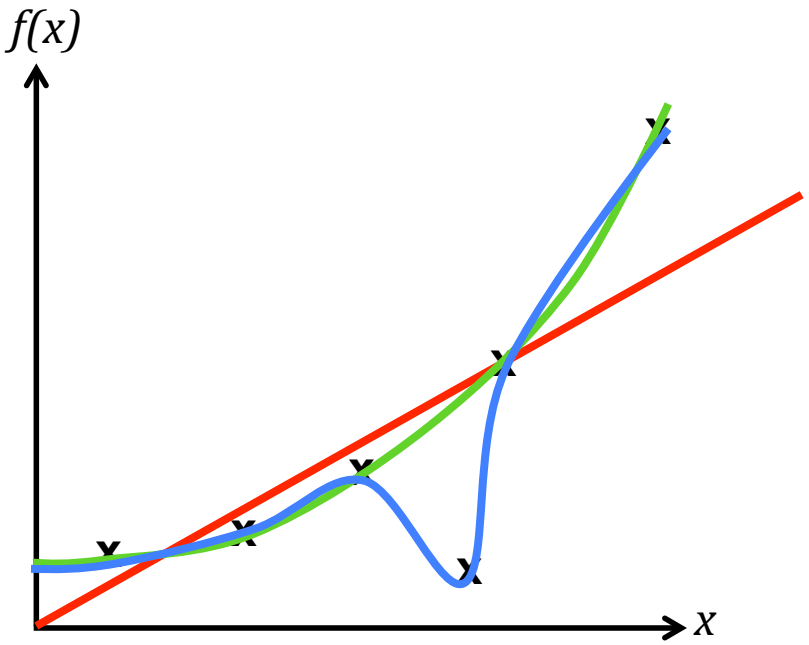
  e.g., curve fitting:



Inductive learning method

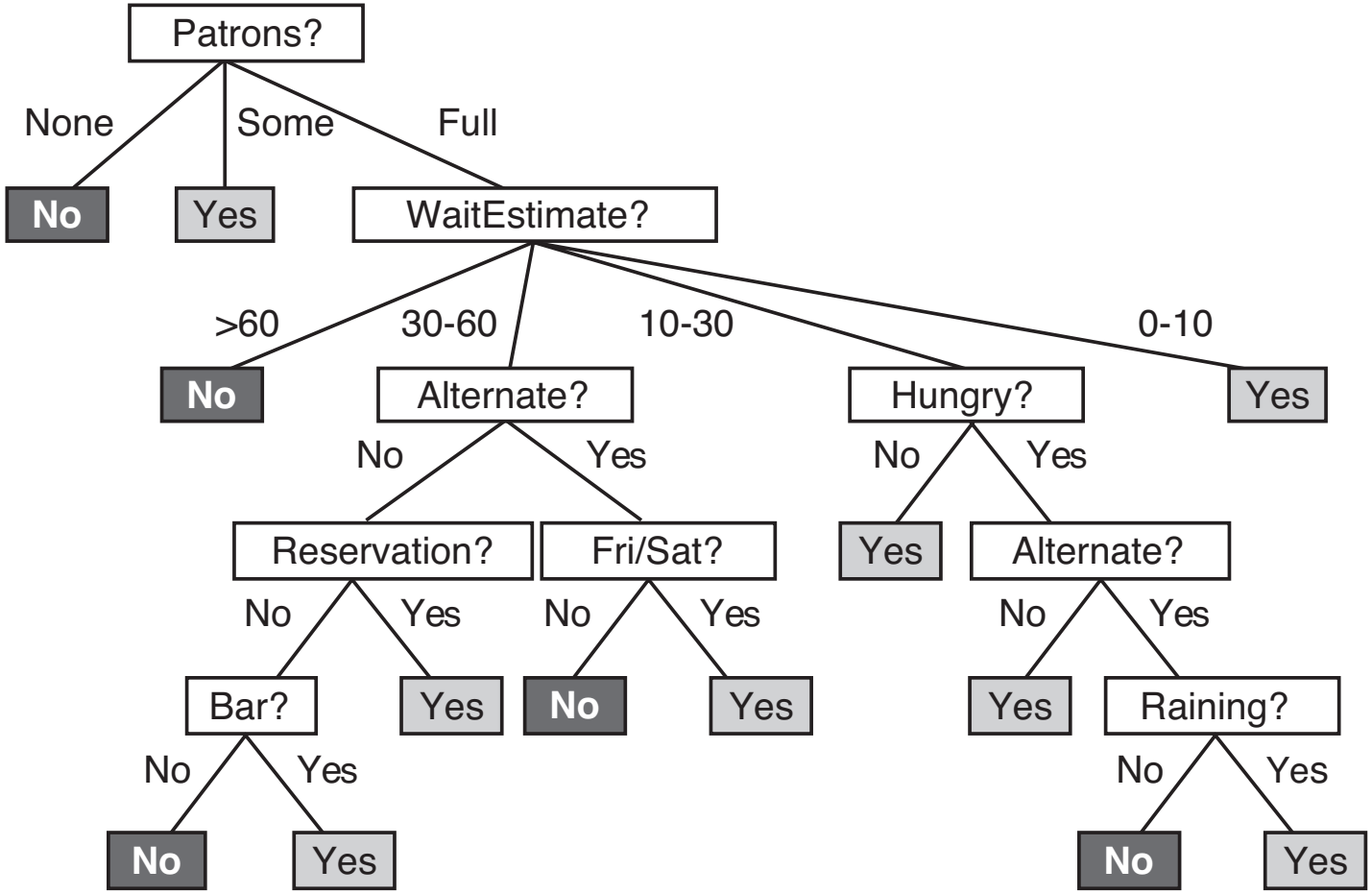
- Construct/adjust *h* to agree with *f* on training set (*h* is **consistent** if it agrees with *f* on all examples)

e.g., curve fitting:



Decision tree representation

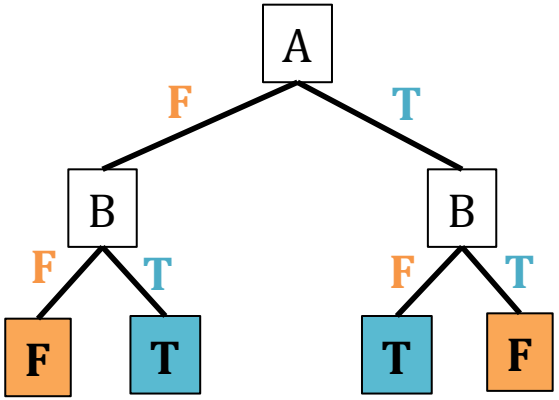
- One possible representation for the hypotheses



Expressiveness of decision trees

- Can express **any function** of the input attributes
  e.g., for Boolean functions, truth table row = path to leaf

A	B	A xor B
F	F	F
F	T	T
T	F	T
T	T	F



- Trivially, there is a consistent decision tree for any training set with one path to leaf for each example (unless *f* nondeterministic in *x*)
  - Probably won't generalize to new examples (overfitting)

- Prefer to find more **compact** decision trees

Hypothesis spaces

- How many **distinct decision trees** with *n* Boolean attributes?
  = number of Boolean functions
  = number of distinct truth tables with 2^n rows = $2^{2\text{^n}}$

  e.g., with 6 Boolean attributes, there are
  18,446,744,073,709,551,616 trees…YIKES!

- How many **purely conjunctive hypotheses**
  (e.g., $Hungry \wedge \neg Rain$)?
  = 3^n distinct conjunctive hypotheses—each attribute can be in (positive), in (negative), or out

- More expressive hypothesis space
  - Increases chance that target function can be expressed ☺
    Increase number of hypotheses consistent w/ training set—may get worse predictions ☹

Decision tree learning

- **Goal:** find a small tree consistent with the training examples
- **Intuition:** (recursively) choose "most significant" attribute as root of (sub)tree

```
function DECISION-TREE-LEARNING(examples, attributes, parent_examples) returns
tree

  if examples is empty then return PLURALITY-VALUE(parent_examples)
  else if all examples have the same classification then return the classification
  else if attributes is empty then return PLURALITY-VALUE(examples)
  else
      A ← argmax_{a ∈ attributes} IMPORTANCE(a, examples)
      tree ← a new decision tree with root test A
      for each value v_k of A do
          exs ← {e : e ∈ examples and e.A = v_k}
          subtree ← DECISION-TREE-LEARNING(exs, attributes − A, examples)
          add a branch to tree with label (A = v_k) and subtree subtree
      return tree
```

- **How do we choose the most significant attribute?**

Detour into information theory…

- Information is about **categories** and **classification**
- We measure **quantity** of information by resources need to represent/store/transmit
- Messages are sequences of 0s and 1s (dots/dashes)—called "bits" (for binary digits)

- Suppose you need to send message containing the identity of a spy
  - It is known to be Mr. Brown or Mr. Smith…
- Can send the message with 1 bit, therefore the event "the spy is Smith" has 1 bit of information

Calculating the quantity of information

- **Uniform distribution** of a set of possible outcomes $\{X_1, \ldots, X_n\}$ means they are equally probable
  i.e., each has a probability of $1/n$

- Suppose there are 8 people who can be the spy (much more exciting!)
- Now, the message requires 3 bits
  - If there are 64 possible spies, message requires 6 bits, etc. (assuming uniform distribution)

- **Information quantity** of a message where the (uniform) probability of each value is p:

  $I = -\log p$ bits

Information quantity intuition and examples

- The more "**surprising**" a message is, the more information it contains
  - If there are 64 equally probable spies, we are more surprised by the identity than if there are only 2 equally probable spies

- There are 26 letters in the alphabet. Assuming a uniform distribution, how much information is in each letter:
  $I = -log\ (1/26) = \log 26 = 4.7$ bits

- Assuming the digits from 0 to 9 are equally probable, will the information in each digit be more or less than in each letter?

Sequences of messages

- Things get interesting when we look beyond single messages to a long **sequence**

- Consider a 4-sided die with symbols *A*, *B*, *C*, *D*
  - Let 00 = *A*, 01 = *B*, 10 = *C*, 11 = *D*
  - Each message is 2 bits
  - If you throw the die 800 times, you get a message 1600 bits long

- That's the best you can do if *A*, *B*, *C*, *D* are equally probable

What about non-uniform distributions?

- Possible outcomes $x_1,\ldots,x_n$ with probabilities $p_1,\ldots,p_n$ that sum to 1
  - E.g., an unfair coin where $n = 2$, $x_1 = H$ (0.75), $x_2 = T$ (0.25)

- In a long **sequence** of events $E = e_1,\ldots,e_k$ assume outcome x_i will occur $\boldsymbol{k * p_i}$ times
  - E.g., $E = HHTHTHHHTTHHHHHTHHHHTTHTHTHHHH$ …….
    If $k = 10000$, assume H occurs 7500 times, T occurs 2500

Example of nonuniform information

- Consider a 4-sided die with symbols A, B, C, D
  - Assume $P(A) = 7/8$ and $P(B) = P(C) = P(D) = 1/24$
  - We can take advantage of that with a **more efficient code**:
    $0 = A$, $10 = B$, $110 = C$, $111 = D$

  - If you throw the die 800 times, expected length of the message is
    $(1 * 700) + (2 * 33.3) + (3 * 33.3) + (3 * 33.3) = 966.67$ bits

Entropy

- Entropy $H(E)$ is the **average information** (in bits) of events $e_1,\ldots,e_k$ in a long repeated sequence E

$$1/k * \sum_{j=1}^{k} I(e_j) =$$

$$1/k * \sum_{i=1}^{n} I(x_i)\ (k * p_i) \ = k/k * \sum_{i=1}^{n} I(x_i)\ (p_i)$$

$$\sum_{i=1}^{n} -log(p_i) * p_i \ = \mathbf{-\sum_{i=1}^{n} p_i * log(p_i)\ bits}$$

- For uniform distribution this is same as $-log\ P(x_i)$ since all $P(x_i)$ are equal

Entropy (cont.)

- Entropy sometimes called "**disorder**"
  - Represents lack of predictability as to the outcome for any element of a sequence (or set)

- If a set has just one outcome

  entropy = 1 * -*log*(1) = 0

- If there are 2 outcomes, then 50/50 probability gives the **maximum entropy**—complete unpredictability
  - Generalizes to any uniform distribution for *n* outcomes

  -(0.5 * *log*(0.5) + 0.5 * *log*(0.5)) = 1 bit

  NOTE: *log*(1/2) = -*log*(2) = -1

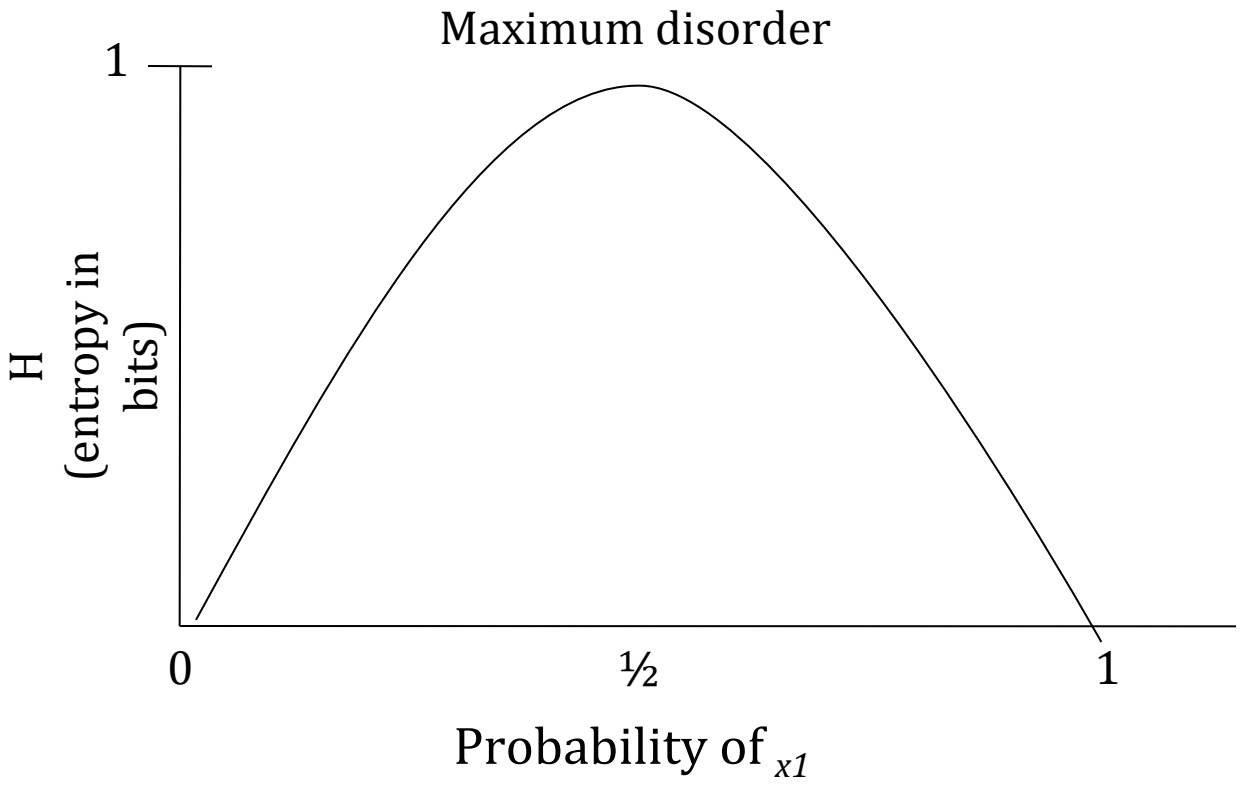
Calculating entropy

- Consider a biased coin: P(*heads*) = 0.75 and P(*tails*) = 0.25
- What is the entropy of a coin toss outcome?

$H = 0.25 * -log(0.25) + 0.75 * -log(0.75) = 0.811$ bits

- A fair coin toss has more "information"
- **The more unbalanced the probabilities, the more predictable the outcome, the less you learn from each message**

Maximum entropy



- Entropy for a set containing 2 possible outcomes (x_1, x_2)
- What if there are 3 possible outcomes?
  - For uniform case: $H = -log(1/3)$ = about 1.58

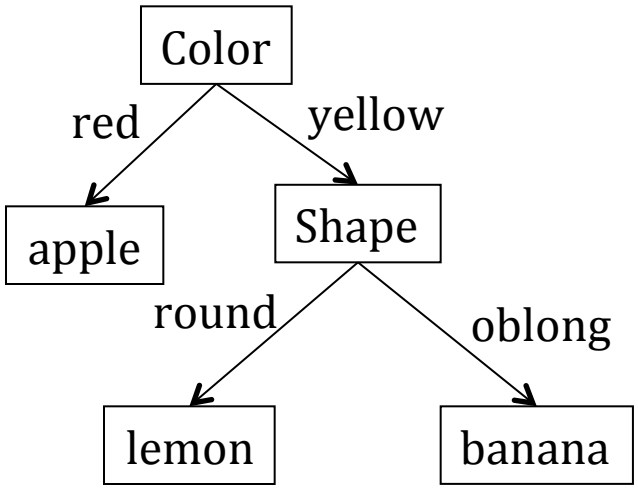
Back to decision trees!

- Given a table with one **result attribute** and several **predictor attributes**, a **classification** (decision) **tree** is a tree s.t.
  1. Each leaf is labeled with a value of the result attribute
  2. Each non-leaf is labeled with the name of a predictor attribute
  3. Each link is labeled with one value of the parent's predictor

- **ID3 algorithm**
  - Takes table as input
  - "Learns" a classification tree that efficiently maps predictor value sets to their results from the table

Trivial example of a classification tree

Record#	Color	Shape	Fruit
1	red	round	apple
2	yellow	round	lemon
3	yellow	oblong	banana



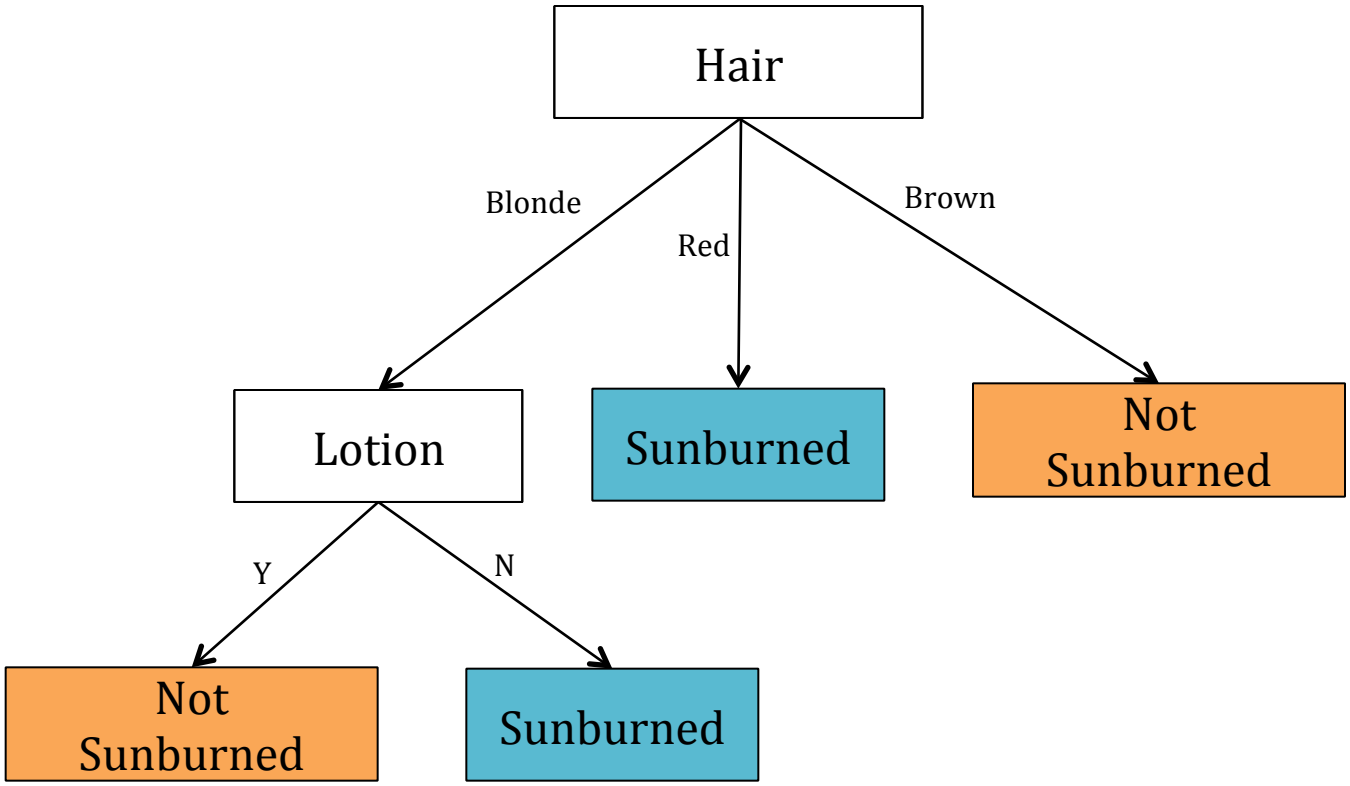
- The goal is to create an "**efficient**" classification tree that always gives the same answer as the table

A well-known toy example: sunburn data

Name	Hair	Height	Weight	Lotion	Sunburned
Sarah	Blonde	Average	Light	No	Yes
Dana	Blonde	Tall	Average	Yes	No
Alex	Brown	Short	Average	Yes	No
Annie	Blonde	Short	Average	No	Yes
Emily	Red	Average	Heavy	No	Yes
Pete	Brown	Tall	Heavy	No	No
John	Brown	Average	Heavy	No	No
Katie	Blonde	Short	Light	Yes	No

- Predictor attributes: Hair, Height, Weight, Lotion

Sunburn decision tree



Outline of the ID3 algorithm

- Create the root and make its COLLECTION the entire table
- Select any **non-singular** leaf node *N* to SPLIT
  - Choose the best attribute *A* for splitting *N* (**use info theory**)
  - For each value of *A* (a_1, a_2,…) create a child of *N*, N_{ai}
  - Label the links from *N* to its children "$A = a_i$"
  - SPLIT the collection of *N* among its children according to their values of *A*
- When no more non-singular nodes exist, the tree is finished

- **Singular node** is one whose COLLECTION includes just one value for the result attribute
  i.e., entropy = 0

Decision tree learning

- **Goal:** find a small tree consistent with the training examples
- **Intuition:** (recursively) choose "most significant" attribute as root of (sub)tree

```
function DECISION-TREE-LEARNING(examples, attributes, parent_examples) returns
tree

  if examples is empty then return PLURALITY-VALUE(parent_examples)
  else if all examples have the same classification then return the classification
  else if attributes is empty then return PLURALITY-VALUE(examples)
  else
      A ← argmax_{a ∈ attributes} IMPORTANCE(a, examples)
      tree ← a new decision tree with root test A
      for each value v_k of A do
          exs ← {e : e ∈ examples and e.A = v_k}
          subtree ← DECISION-TREE-LEARNING(exs, attributes − A, examples)
          add a branch to tree with label (A = v_k) and subtree subtree
      return tree
```

- **How do we choose the most significant attribute?**

Choosing the best attribute to SPLIT

- Choose the attribute that is **most informative** and reduces entropy (disorder) the most
  - Attribute with highest **information gain**

- Assume there are k attributes we can choose
  - For each one, compute how much less entropy exists in the resulting children than we had in the parents
    $H(N)$ = weighted sum of H(children of N)

  - Each child's entropy is weighted by the "probability" of that child (estimated by the proportion of the parent's collection that would be transferred to the child in the split)

Constructing a decision tree with ID3

- Collection for split node 1 (root node) is full set of data
  $C(S_1)$ = {S, D, X, A, E, P, J, K}
  P(*Sunburn*) = {3/8, 5/8}
  $H(N_1)$ = -[3/8 *log*(3/8) + 5/8 *log*(5/8)]
  = 0.53 + 0.424 = **0.954**

- Find information gain (IG) for all four predictors:
  hair, height, weight, lotion

- Start with **lotion**: values {yes, no}
  Child 1 (yes) = {D, X, K}
  P(*Sunburn*) = {0/3, 3/3}
  H(Child 1) = 0

  Child 2 (no) = {S, A , E, P, J}
  P(*Sunburn*) = {3/5, 2/5}
  H(C2) = -[3/5 *log*(3/5) + 2/5 *log*(2/5)]
  = 0.971

  H({C1, C2}) = 3/8 *0 + 5/8 * 0.971 = 0.607
  IG(lotion) = 0.954 - 0.607 = 0.347

Constructing a decision tree with ID3

- Next try **hair color**: values {blonde, brown, red}
  Child 1 (blonde) = {S, D, A, K}
  P(*Sunburn*) = {2/4, 2/4}
  H(C1) = 1

  Child 2 (brown) = {X, P, J}
  P(*Sunburn*) = {0/3, 3/3}
  H(C2) = 0

  Child 3 (blonde) = {E}
  P(*Sunburn*) = {1/1, 0/1}
  H(C3) = 0

  H({C1, C2, C3}) = 4/8 *1 + 3/8 * 0 + 1/8 * 0 = 0.5
  IG(hair color) = 0.954 - 0.5 = 0.454

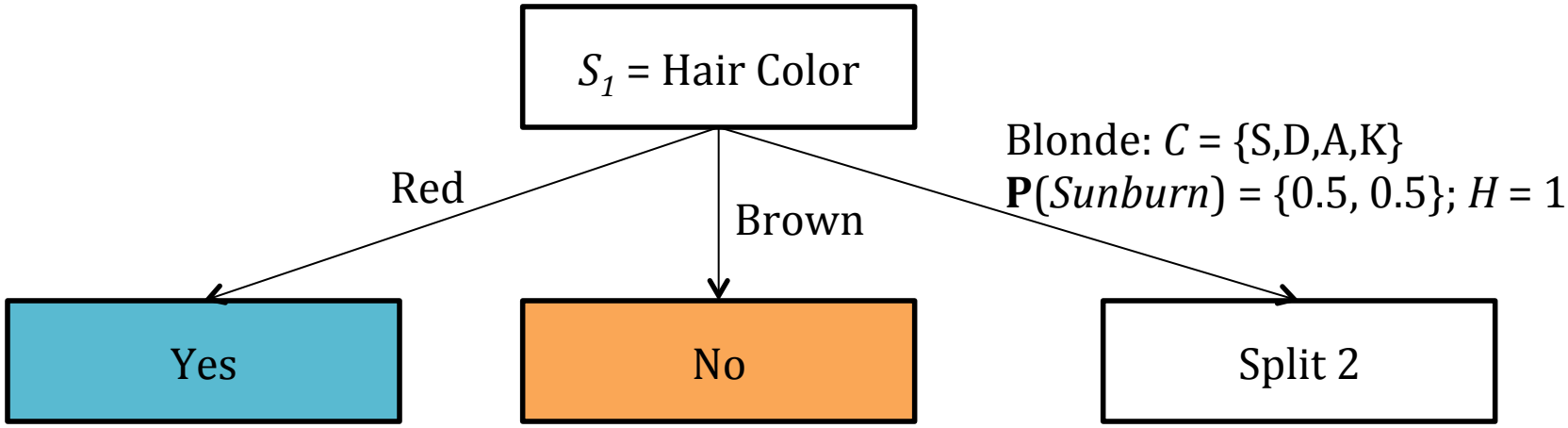
- Next try **height**…
  IG(height) = 0.954 - 0.69 = 0.26

- Next try **weight**…
  IG(weight) = 0.954 - 0.94 = 0.014

Constructing a decision tree with ID3

- Hair color wins!

- Draw the first split and assign the collections



Constructing a decision tree with ID3

- Collection for split node 2

  $C(S_2)$ = {S, D, A, K}

  P(*Sunburn*) = {0.5, 0.5}

  $H(S_1)$ = -[0.5 *log*(0.5) + 0.5 *log*(0.5)] = **1**

- Start with **lotion**: values {yes, no}

  Child 1 (yes) = {D, K}

  P(*Sunburn*) = {0/2, 2/2}

  H(C1) = 0

  Child 2 (no) = {S, A}

  P(*Sunburn*) = {2/2, 0/2}

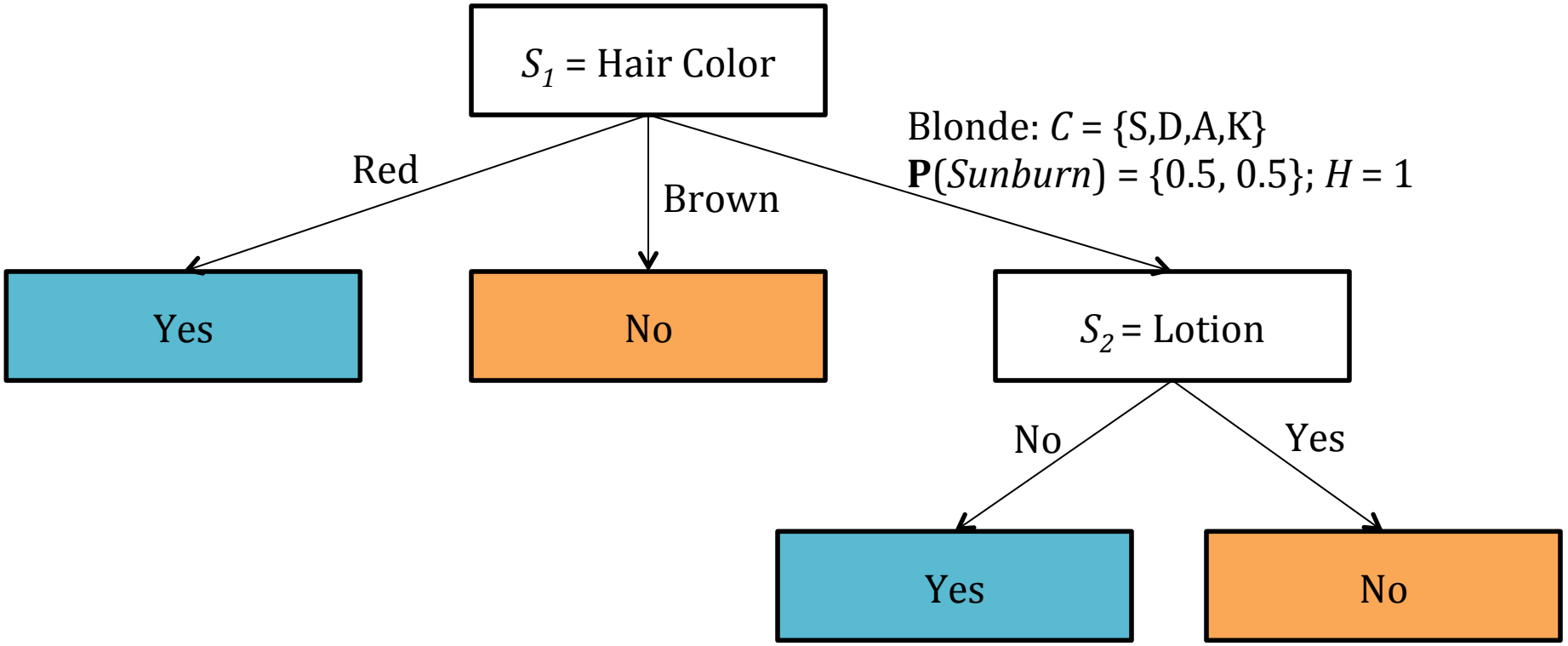
  H(C2) = 0

  H({C1, C2}) = 0

  IG(lotion) = 1- 0 = 1

- **No need to go any further!**

Constructing a decision tree with ID3



- Compact decision tree—only need two attributes to classify entire table!