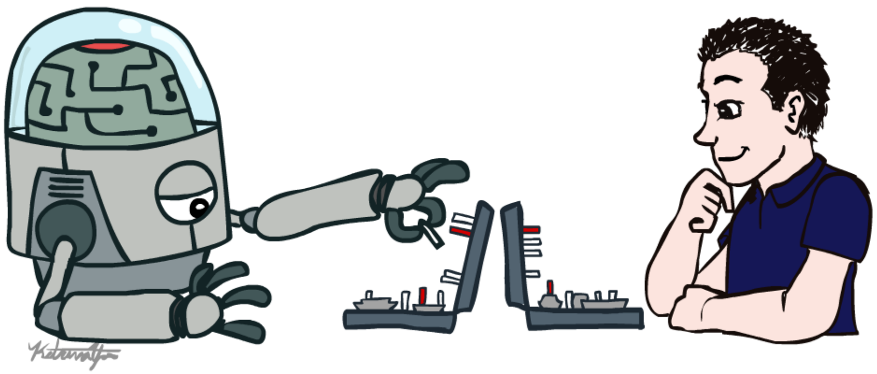
CSE 3521:
Introduction to Artificial Intelligence



[Many slides are adapted from the UC Berkeley. CS188 Intro to AI at UC Berkeley and previous CSE 3521 course at OSU.]

The Ohio State University

Exam Format

- 55 minutes
  - Available for 24 hrs
  - i.e, you can take it anytime during the day

- Carman Quiz
  - Mixed format: MCQ, T/F, fill-in-the-blanks, ...

- If you need any special accommodation email me **ASAP**
  - tabassum.13@osu.edu

PEAS

PEAS

- **P**erformance – measuring the agent’s success
- **E**nvironment – what populates the problem’s world?
- **A**ctuators – what can the agent act with?
- **S**ensors – how can the agent perceive the world?

Peas: Examples

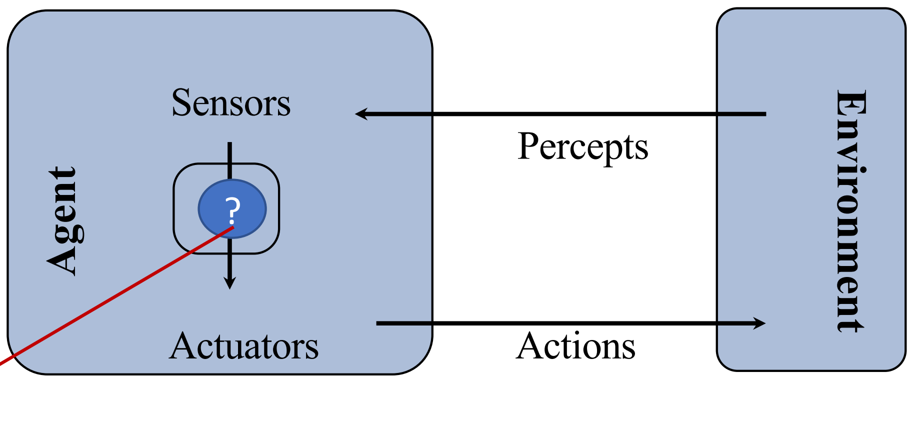
Agent Type	**Perf. Measure**	**Environment**	**Actuators**	**Sensors**
Medical diagnosis system	Healthy patient, minimize costs/lawsuits	Patient, hospital, staff	Display questions, tests, diagnoses, treatments, referrals	Keyboard entry of symptoms, findings, patient's answers
Satellite image analysis system	Correct image classification	Downlink from orbiting satellite	Display classification of scene	Color pixel arrays (cameras)
Part-picking robot	Percentage of parts in correct bins	Conveyor belt with parts, bins	Jointed arm and hand	Camera, joint angle sensors
Refinery controller	Maximize purity, yield, safety	Refinery, operators	Valves, pumps, heaters, displays	Temperature, pressure, chemical sensors
Interactive English tutor	Maximize student's score on test	Set of students, testing agency	Display exercises, suggestions, corrections	Keyboard entry

Rational Agent

What makes an AI agent

- **Agent** – an entity that perceives its environment through sensors, and acts on it with effectors (actuators).

- Percepts are constrained by Sensors + Environment
- Actions are constrained by Actuators + Environment

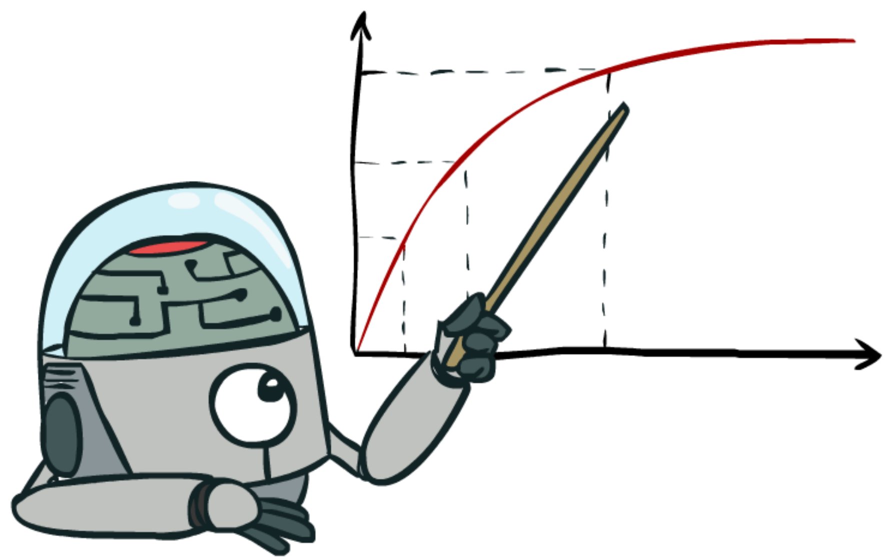


Agent Function (policy) – how does it choose the action?

What is a rational AI agent?

- A **rational agent** always acts to **maximize its expected performance measure**, given current **percept/state**

- Rationality $\neq$ omniscience
  - There is “uncertainty” in the environment.
  - That is why we emphasize “expected”.



Kinds of Environments

- **Six** common properties to distinguish environments (not exhaustive)
  - Fully observable vs Partially observable
  - Single agent vs Multiagent
  - Deterministic vs Stochastic
  - Episodic vs Sequential
  - Static vs Dynamic
  - Discrete vs Continuous

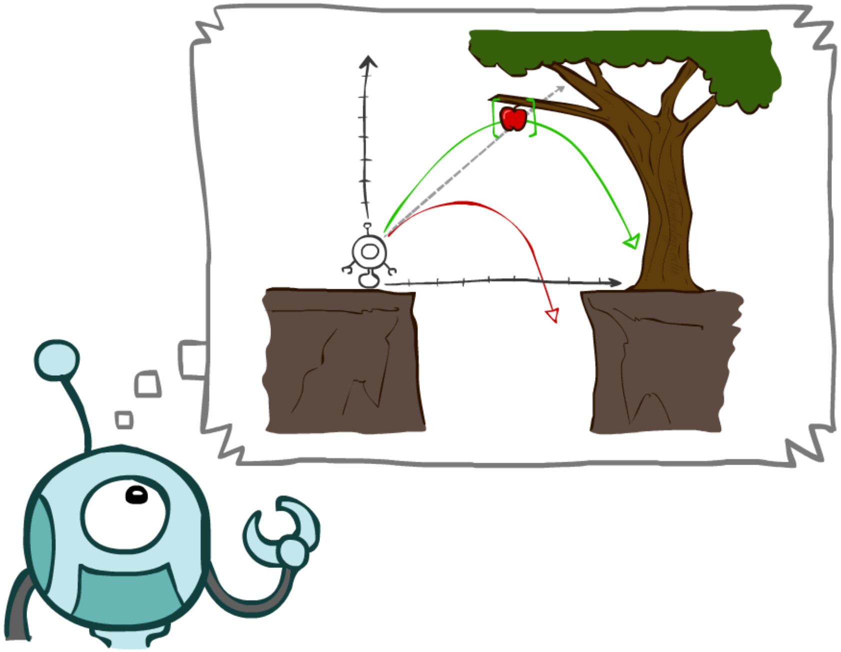
Examples

	Crossword puzzle	Taxi Driving
Observability	Fully	Partially
Deterministic vs Stochastic	Deterministic	Stochastic
Episodic vs Sequential	Sequential	Sequential
Static vs Dynamic	Static	Dynamic
Discrete vs Continuous	Discrete	Continuous
Single vs Multi Agent	Single	Multi

Search

Search

- Types
  - Uninformed Search Methods
    - Depth-First Search
    - Breadth-First Search
    - Uniform-Cost Search
  - Informed Search Methods
    - Greedy Search
    - A* Search



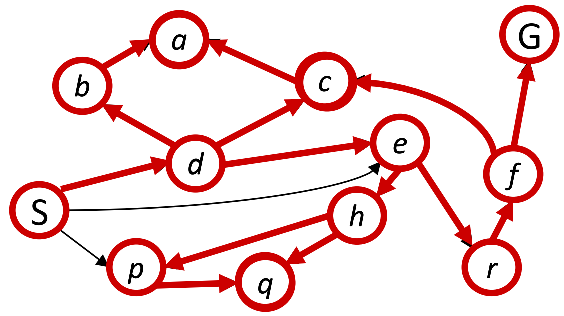
Search Problems

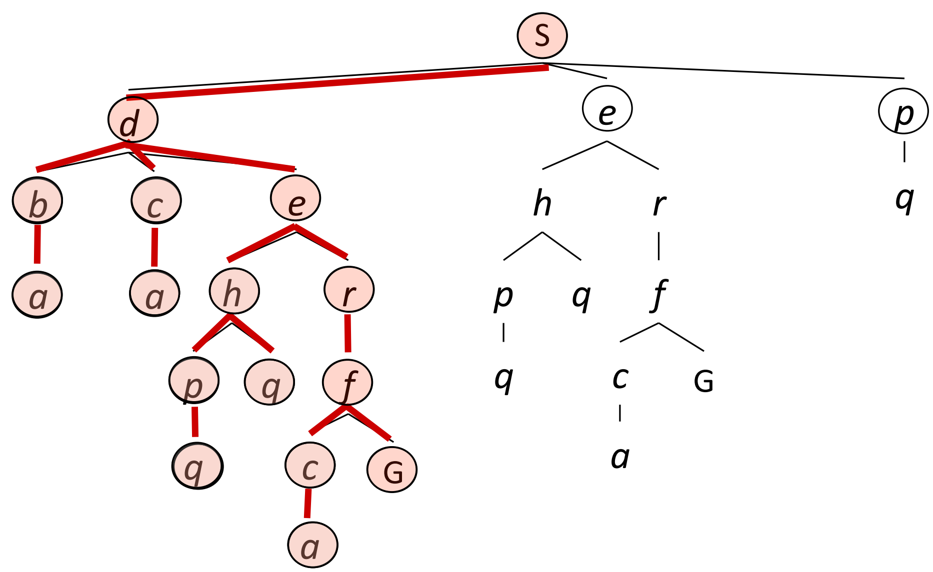
- A search problem consists of:
  - A state space
  - A successor function
    (with actions, costs)
  - A start state and a goal test
- A solution is a sequence of actions (a plan) which transforms the start state to a goal state

Depth-First Search

Strategy: expand a deepest node first

Implementation: Fringe is a LIFO stack

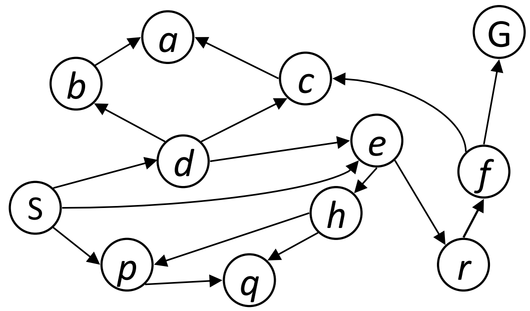


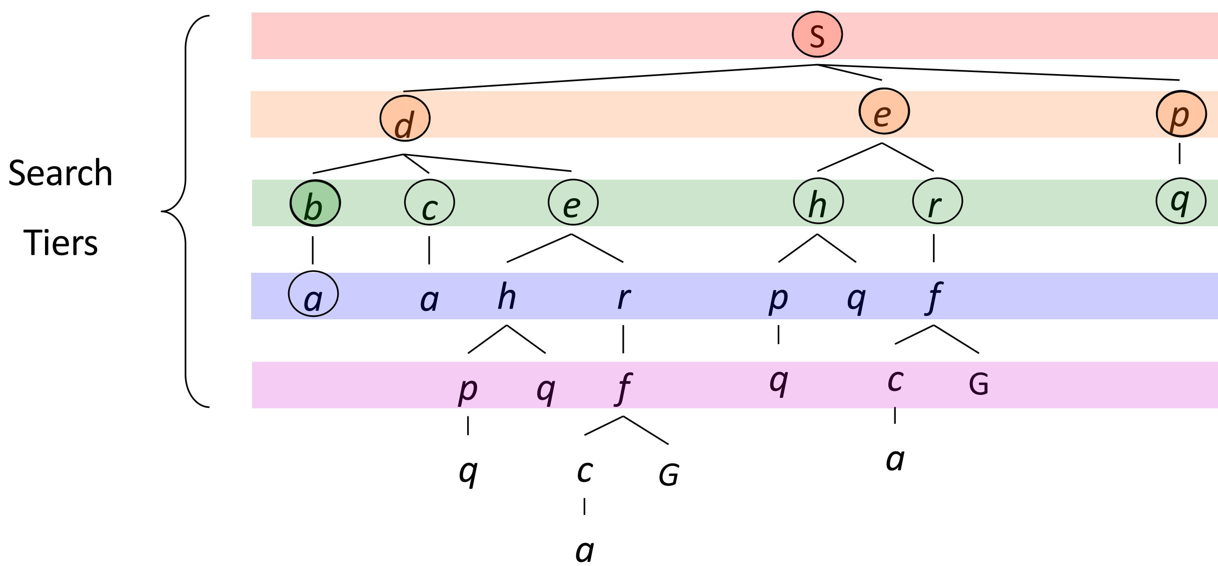


Breadth-First Search

Strategy: expand a shallowest node first

Implementation: Fringe is a FIFO queue



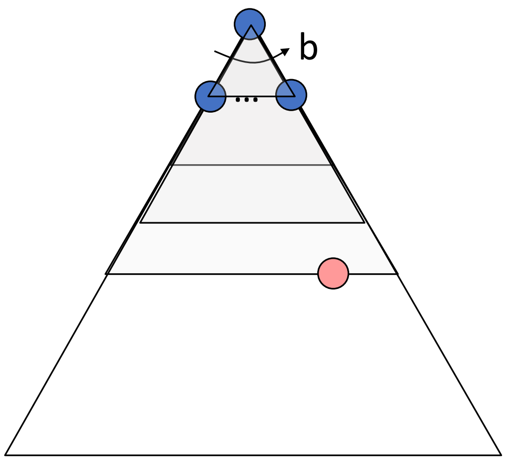


Iterative Deepening Search

- Idea: get DFS's space advantage with BFS's time / shallow-solution advantages
  - Run a DFS with depth limit 1. If no solution…
  - Run a DFS with depth limit 2. If no solution…
  - Run a DFS with depth limit 3. …..

- Isn't that wastefully redundant?
  - Generally most work happens in the lowest level searched, so not so bad!

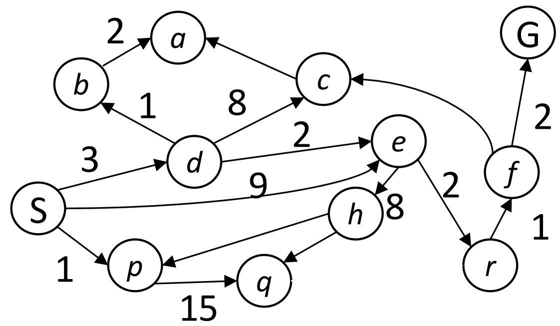
- A Preferred method with <u>large search space</u> and <u>depth of solution not known</u>

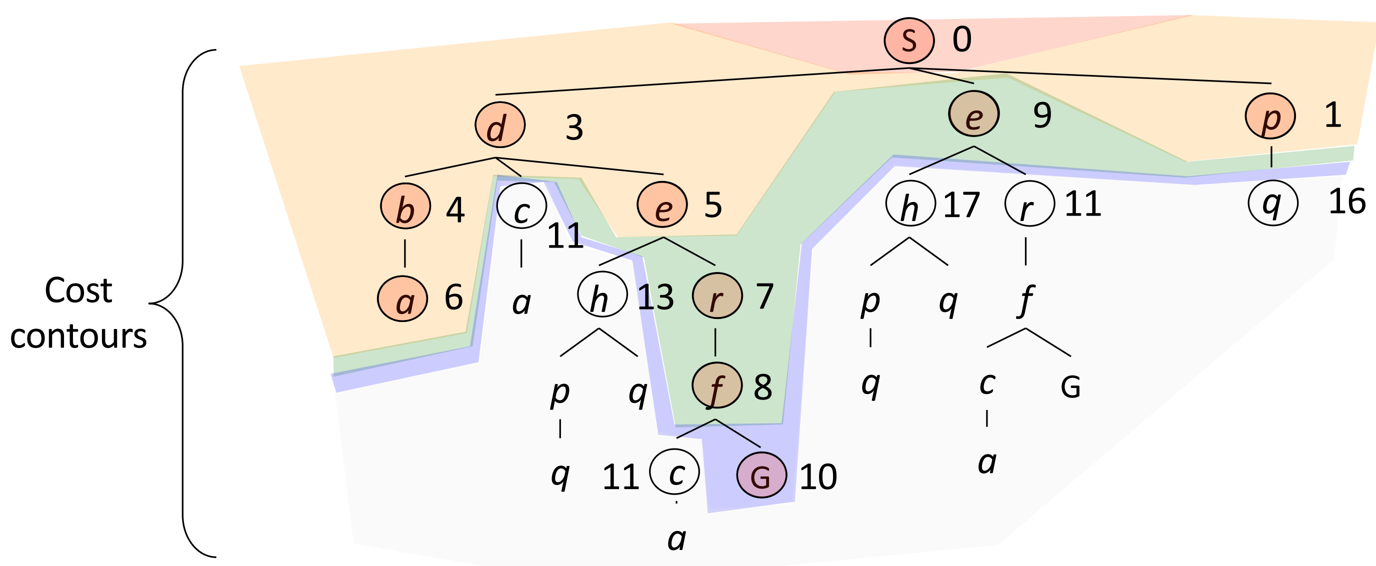


Uniform Cost Search (USC)

Strategy: expand a cheapest node first:

Fringe is a priority queue (priority: cumulative cost)



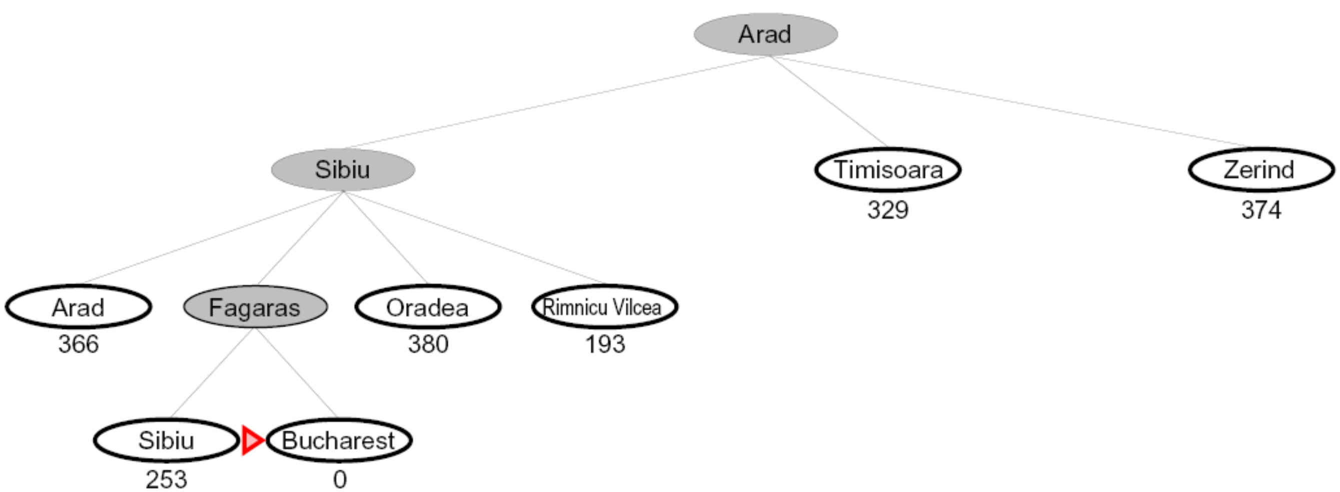


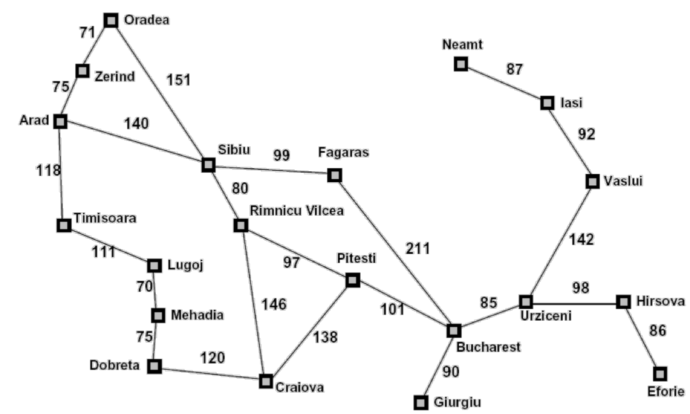
Search Heuristics

- A heuristic is
  - A function that *estimates* how close a state is to a goal
  - Designed for a particular search problem
  - Examples: Manhattan distance, Euclidean distance for pathing
    - not the exact “path” distance

Greedy Search

- Expand the node that seems closest…

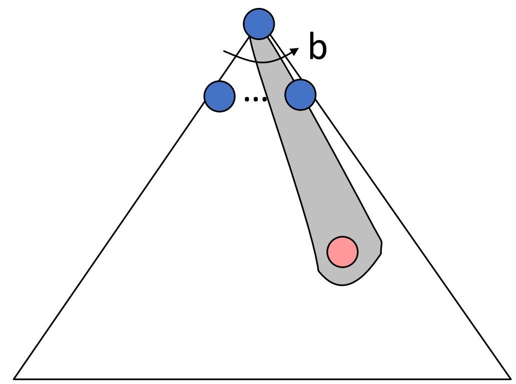




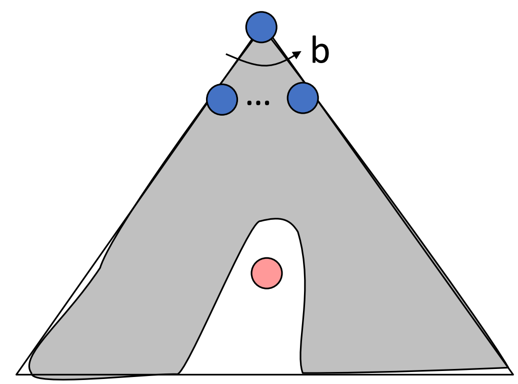
- What can go wrong?
  - Does not guarantee the optimal solution

Greedy Search

- Strategy: expand a node that you think is closest to a goal state
  - Heuristic: estimate of distance to nearest goal for each state



- A common case:
  - Best-first takes you straight to the (wrong) goal



- Worst-case: like a badly-guided DFS

A* Search

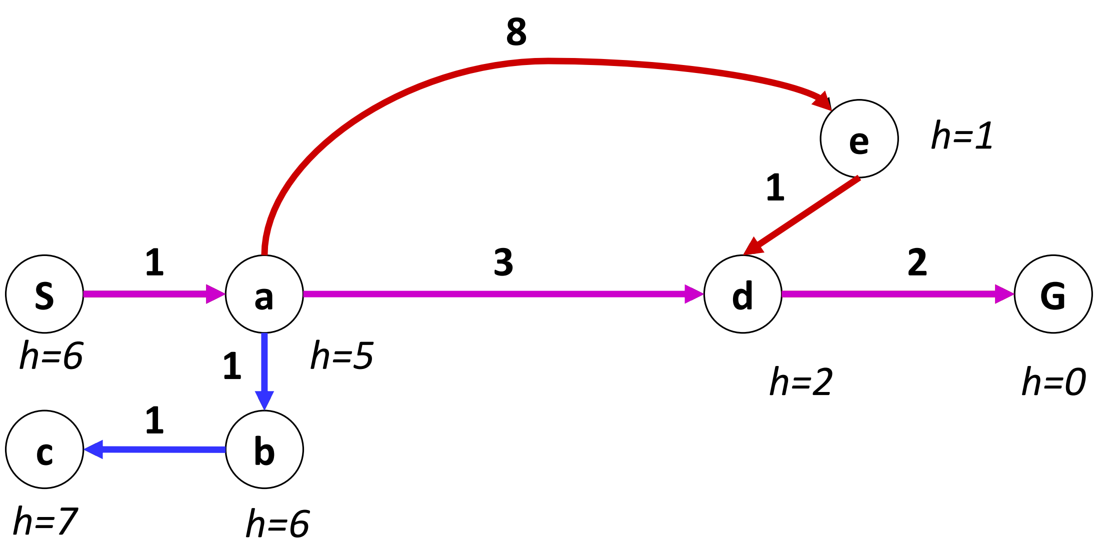
UCS

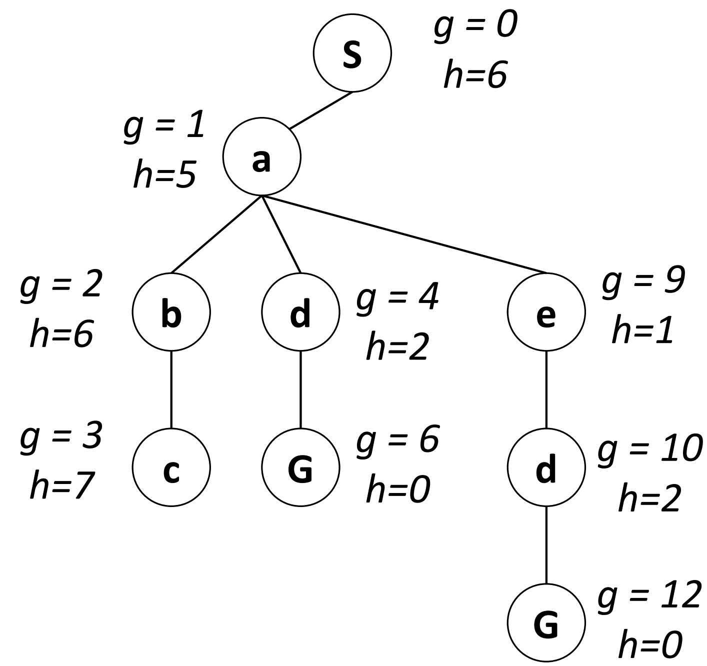
Greedy

A*

A* is a Combination of UCS and Greedy

- Uniform-cost orders by path cost, or *backward cost* g(n)
- Greedy orders by goal proximity, or *forward cost* h(n)





- A* Search orders by the sum: f(n) = g(n) + h(n)

Propositional Logic

Logic

- For <u>logical</u> agents, knowledge is <u>definite</u>
  - Each proposition is either "True" or "False"

- Logic has advantage of being simple representation for knowledge-based agents
  - But limited in its ability to handle uncertainty

- We will examine propositional logic and first-order logic

Logical Agent

- Need agent to represent beliefs
  - "There is a pit in (2, 2) or (3, 1)"
  - "There is no Wumpus in (2, 2)"
- Need to make inferences
  - If available information is correct, draw a conclusion that is guaranteed to be correct
- Need <u>representation</u> and <u>reasoning</u>
  - Support the operation of knowledge-based agent

Knowledge Representation

- For expressing knowledge in computer-tractable form
- Knowledge representation language defined by
  - **Syntax**
    - Defines the possible well-formed configurations of sentences in the language
  - **Semantics**
    - Defines the “meaning” of sentences (need interpreter)
    - Defines the truth of a sentence in a world (or model)

The Language of Arithmetic

- Syntax: "$x + 2 \geq y$" is a sentence

  "$x2 + y >$" is not a sentence

- Semantics: $x + 2 \geq y$ is true iff the number $x + 2$ is no less than the number y

  $x + 2 \geq y$ is True in a world where x=7, y=1

  $x + 2 \geq y$ is False in a world where x=0, y=6

Inference

- Sentence is valid iff it is true under all possible interpretations in all possible worlds
  - Also called tautologies
  - “There is a stench at (1,1) or there is not a stench at (1,1)”
  - “There is an open area in front of me” is not valid in all worlds

- Sentence is satisfiable iff there is some interpretation in some world for which it is true
  - “There is a wumpus at (1,2)” could be true in some situation
  - “There is a wall in front of me and there is no wall in front of me” is unsatisfiable

Propositional Logic: Syntax

- Syntax of propositional logic defines <u>allowable</u> sentences
- Atomic sentences consists of a single proposition symbol
  - Each symbol stands for proposition that can be True or False
- Symbols of propositional logic
  - Propositional symbols: *P*, *Q*, … (e.g., "Today is Tuesday")
  - Logical constants: *True*, *False*
- Making complex sentences
  - Logical connectives of symbols: $\wedge$, $\vee$, $\Leftrightarrow$, $\Rightarrow$, $\neg$
  - Also have parentheses to enclose each sentence: (…)
- Sentences will be used for inference/problem-solving

Propositional Logic: Syntax

- *True*, *False*, S_1, S_2, … are sentences
- If S is a sentence, $\neg S$ is a sentence
  - Not (negation)
- $S_1 \wedge S_2$ is a sentence, also $(S_1 \wedge S_2)$
  - And (conjunction)
- $S_1 \vee S_2$ is a sentence
  - Or (disjunction)
- $S_1 \Rightarrow S_2$ is a sentence (e.g., "Today is Tuesday" implies "Tomorrow is Wednesday")
  - Implies (conditional)
- $S_1 \Leftrightarrow S_2$ is a sentence
  - Equivalence (biconditional)

Propositional Logic: Semantics

- Semantics defines the rules for determining the truth of a sentence
  - With respect to a particular model)
    - $\neg S$ is true iff S is false
    - $S_1 \wedge S_2$ is true iff S_1 is true <u>and</u> S_2 is true
    - $S_1 \vee S_2$ is true iff S_1 is true <u>or</u> S_2 is true
    - $S_1 \Rightarrow S_2$ is true iff S_1 is false <u>or</u> S_2 is true
      (is false iff S_1 is true <u>and</u> S_2 is false)
      (if S_1 is true, then claiming that S_2 is true, otherwise make no claim)
    - $S_1 \Leftrightarrow S_2$ is true iff $S_1 \Rightarrow S_2$ is true <u>and</u> $S_2 \Rightarrow S_1$ is true (S_1 same as S_2)

Propositional Inference: Enumeration Method

- Truth tables can test for valid sentences
  - True under all possible interpretations in all possible worlds
- For a given sentence, make a truth table
  - Columns as the combinations of propositions in the sentence
  - Rows with all possible truth values for proposition symbols
- If sentence true in every row, then valid

Example

- Test $(P \wedge H) \Rightarrow (P \vee \neg H)$

P	H	$P \wedge H$	$\neg H$	$(P \vee \neg H)$	$(P \wedge H) \Rightarrow (P \vee \neg H)$
False	False	False	True	True	True
False	True	False	False	False	True
True	False	False	True	True	True
True	True	True	False	True	True

Inference Rules for Prop. Logic

- Modus Ponens
  - From implication and premise of implication, can infer conclusion

$$\frac{\alpha \Rightarrow \beta, \quad \alpha}{\beta}$$

Inference Rules for Prop. Logic

- <u>And-Elimination</u>
  - From conjunction, can infer any of the conjuncts

$$\frac{\alpha_1 \land \alpha_2 \land \cdots \land \alpha_n}{\alpha_i}$$

Inference Rules for Prop. Logic

- And-Introduction
  - From list of sentences, can infer their conjunction

$$\frac{\alpha_1, \alpha_2, \ldots, \alpha_n}{\alpha_1 \land \alpha_2 \land \cdots \land \alpha_n}$$

Inference Rules for Prop. Logic

- Or-Introduction
  - From sentence, can infer its disjunction with anything else

$$\frac{\alpha_i}{\alpha_1 \lor \alpha_2 \lor \cdots \lor \alpha_n}$$

Inference Rules for Prop. Logic

- Double-Negation Elimination
  - From doubly negated sentence, can infer a positive sentence

$$\frac{\neg\neg\alpha}{\alpha}$$

Inference Rules for Prop. Logic

- <u>Unit Resolution</u>
  - From disjunction, if one of the disjuncts is false, can infer the other is true

$$\frac{\alpha \vee \beta, \quad \neg \beta}{\alpha}$$

Inference Rules for Prop. Logic

- Resolution
  - Most difficult because β cannot be both true and false
  - One of the other disjuncts must be true in one of the premises
    - (implication is transitive)

$$\frac{\alpha \vee \beta, \quad \neg\beta \vee \gamma}{\alpha \vee \gamma}$$

$\neg\beta$	**OR**	β
$\dfrac{\alpha \vee \beta, \neg\beta}{\alpha}$	**OR**	$\dfrac{\neg\beta \vee \gamma, \beta}{\gamma}$
α	**OR**	γ

α	β	γ	$\alpha \vee \beta$	$\neg\beta \vee \gamma$	$\alpha \vee \gamma$
T	T	T	T	T	T
T	T	F	T	F	T
T	F	T	T	T	T
T	F	F	T	T	T
F	T	T	T	T	T
F	T	F	T	F	T
F	F	T	F	T	F
F	F	F	F	T	F

First Order Logic

First-Order Logic

- Also called first-order predicate calculus
  - FOL, FOPC
- Makes stronger commitments
  - World consists of objects
    - Things with identities
    - e.g., people, houses, colors, …
  - Objects have properties/relations that distinguish them from other objects
    - e.g., Properties: red, round, square, …
    - e.g., Relations: brother of, bigger than, inside, …
  - Have functional relations
    - Return the object with a certain relation to given "input" object
    - The "inverse" of a (binary) relation
    - e.g., father of, best friend

Syntax of FOL: Basic Elements

- Constant symbols for specific objects
  KingJohn, 2, *OSU*, …
- Variables
  x, *y*, *a*, *b*, …
- Predicate properties (unary) / relations (pairwise or more)
  Smart(), Brother(), Married(), >, …
- Functions (return objects)
  Sqrt() , LeftTo(), FatherOf(), …
- Connectives
  $\wedge \; \vee \; \neg \; \Rightarrow \; \Leftrightarrow$
- Quantifiers
  $\forall \; \exists$
- Equality
  =

Atomic Sentences

- Collection of terms and relation(s) together to state facts
- Atomic sentence
  - $predicate(term_1, \ldots, term_n)$
  - Or $term_1 = term_n$
- Examples

  Brother(Richard, John)

  Married(FatherOf(Richard), MotherOf(John))

Complex Sentences

- Made from atomic sentences using <u>logical connectives</u>

  $\neg S$, $S_1 \wedge S_2$, $S_1 \vee S_2$, $S_1 \Rightarrow S_2$, $S_1 \Leftrightarrow S_2$

  - Examples:
    - *Older(John, 30)* $\Rightarrow \neg$ *Younger(John, 30)*
    - $> (1,2) \vee \leq (1,2)$

Quantifiers

- Currently have logic that allows objects
- Now want to express properties of entire <u>collections of objects</u>
  - Rather than enumerate the objects by name
- Two standard quantifiers
  - Universal $\forall$
  - Existential $\exists$

Universal Qualification

- "For all …" (typically use implication $\Rightarrow$)
  - Allows for "rules" to be constructed
- $\forall$ *<variables> <sentence>*
  - Everyone at OSU is smart
    - $\forall x\ At(x, OSU) \Rightarrow Smart(x)$
- $\forall x\ P$ is equivalent to <u>conjunction</u> of all <u>instantiations</u> of P
    - $(At(John, OSU) \Rightarrow Smart(John))$
      $\wedge\ (At(Bob, OSU) \Rightarrow Smart(Bob))$
      $\wedge\ (At(Mary, OSU) \Rightarrow Smart(Mary)) \wedge\ \ldots$

Existential Quantification

- "There exists …" (typically use conjunction $\wedge$)
  - Makes a statement about <u>some</u> object (not all)
- $\exists$ *<variables> <sentences>*
  - Someone at OSU is smart
    - $\exists x\ At(x, OSU) \wedge Smart(x)$
- $\exists\, x\, P$ is equivalent to <u>disjunction</u> of all <u>instantiations</u> of P
    - $(At(John, OSU) \wedge Smart(John))$
      $\vee\ (At(Bob, OSU) \wedge Smart(Bob))$
      $(At(Mary, OSU) \wedge Smart(Mary))\ \vee\ \ldots$
- Uniqueness quantifier
  - $\exists!\, x$ says a <u>unique</u> object exists (i.e. there is exactly one)

Properties of Quantifiers

- Quantifier duality: Each can be expressed using the other
  - $\forall x\ \ Person(x) \Rightarrow Likes(x, IceCream)$ "Everybody likes ice cream"
  - $\neg\exists x\ \ Person(x) \wedge \neg Likes(x, IceCream)$ "Not exist anyone who does not like ice cream"
  - $\exists x\ Person(x) \wedge Likes(x, Broccoli)$ "Someone likes broccoli"
  - $\neg\forall x\ Person(x) \Rightarrow \neg Likes(x, Broccoli)$ "Not the case that everyone does not like broccoli"

Properties of Quantifiers

- Important relations
  - $\exists x \; P(x) \; = \; \neg\forall x \; \neg P(x)$
  - $\forall x \; P(x) \; = \; \neg\exists x \, \neg P(x)$
  - $P(x) \Rightarrow Q(x)$ is same as $\neg P(x) \vee Q(x)$
  - $\neg \, (P(x) \wedge Q(x))$ is same as $\neg P(x) \vee \neg Q(x)$

Universal Quantifiers

- $\forall x \, \forall y$ is same as $\forall y \, \forall x$ ($\forall x,y$)
- $\exists \, x \, \exists \, y$ is same as $\exists \, y \, \exists \, x$ ($\exists \, x,y$)
- $\exists \, x \, \forall y$ is <u>not same</u> as $\forall y \, \exists \, x$
  - $\exists \, y \; Person(y) \wedge (\forall x \; Person(x) \Rightarrow Loves(x,y))$
    - "There is someone who is loved by everyone"
  - $\forall x \; Person(x) \Rightarrow \exists y \; Person(y) \wedge Loves(x,y)$
    - "Everybody loves somebody"
(not guaranteed to be the same person)

How to do inference in FOPC

- Reduction of first-order inference to propositional inference
- First-order inference algorithms
  - Generalized Modus Ponens
  - Forward chaining ***
  - Backward chaining ***
  - Resolution-based theorem proving ***

Reduction to Propositional Inference

- Universal Quantifiers ($\forall$)
  - Recall: Sentence must be true *for all* objects in the world (all values of variable)
  - So substituting any object must be valid (Universal Instantiation, UI)
- Example
  - $\forall x \;\; Person(x) \Rightarrow Likes(x, IceCream)$
    - Substituting: (1), $\{x/Jack\}$
  - $Person(Jack) \Rightarrow Likes(Jack, IceCream)$

Reduction to Propositional Inference (con't)

- Existential Quantifiers ($\exists$)
  - Recall: Sentence must be true *for some* object in the world (or objects)
  - Assume we know this object and give it an arbitrary (unique!) name (Existential Instantiation, EI)
  - Known as Skolem constant (SK1, SK2, …)
- Example
  - $\exists x \;\; Person(x) \wedge Likes(x, IceCream)$
    - Substituting: (1), $\{x/SK1\}$
  - $Person(SK1) \wedge Likes(SK1, IceCream)$
- We don't know who "SK1" is (and usually can't), but we know they must exist

Reduction to Propositional Inference (con't)

- Multiple Quantifiers
  - No problem if same type ($\forall x,y$ or $\exists x,y$)
  - Also no problem if: $\exists x\ \forall y$
    - There must be some *x* for which the sentence is true with every possible *y*
    - Skolem constant still works (for *x*)
- Problem with $\forall x\ \exists y$
  - For every possible *x*, there must be some *y* that satisfies the sentence
  - Could be different *y* value to satisfy for each *x*!

Reduction to Propositional Inference (con't)

- Problem with $\forall x \; \exists y$ (con't)
  - The value we substitute for *y* must depend on *x*
  - Use a Skolem function instead
- Example
  - $\forall x \; \exists y \; Person(x) \Rightarrow Loves(x,y)$
    - Substitute: (1), $\{y/SK1(x)\}$
  - $\forall x \; Person(x) \Rightarrow Loves(x, SK1(x))$
    - Then: (2), $\{x/Jack\}$
  - $Person(Jack) \Rightarrow Loves(Jack, SK1(Jack))$
- *SK1*(*x*) is *effectively* a function which returns a person that *x* loves. But, again, we can't generally know the specific value it returns.

Reduction to Propositional Inference (con't)

- Internal Quantifiers
  - Previous rules only work if quantifiers are external (left-most)
  - Consider: $\forall x\ (\exists y\ Loves(x,y)) \Rightarrow Person(x)$
  - "For all *x*, if there is some *y* that *x* loves, then *x* must be a person"
  - A Skolem function limits the values *y* could take (to one) and we can't know what it is.
- Need to move the quantifier outward
  - $\forall x\ (\exists y\ Loves(x,y)) \Rightarrow Person(x)$
  - $\forall x\ \neg(\exists y\ Loves(x,y)) \vee Person(x)$ (convert to $\neg,\vee,\wedge$)
  - $\forall x\ \forall y\ \neg Loves(x,y) \vee Person(x)$ (move $\neg$ inward)
  - $\forall x\ \forall y\ Loves(x,y) \Rightarrow Person(x)$
- Now we can see that we can actually substitute *anything* for *y*
- May need to rename variables before moving quantifier left

Reduction to Propositional Inference (con't)

- Once have non-quantified sentences (from quantified sentences using UI, EI), possible to reduce first-order inference to propositional inference
- Suppose KB contains:

$$\forall x \;\; King(x) \wedge Greedy(x) \Rightarrow Evil(x)$$

$$King(John)$$

$$Greedy(John)$$

$$Brother(Richard, John)$$

- Using UI with $\{x/John\}$ and $\{x/Richard\}$, we get

$$King(John) \wedge Greedy(John) \Rightarrow Evil(John)$$

$$King(Richard) \wedge Greedy(Richard) \Rightarrow Evil(Richard)$$

Reduction to Propositional Inference (con't)

- Now the KB is essentially propositional:

$$King(John) \land Greedy(John) \Rightarrow Evil(John)$$

$$King(Richard) \land Greedy(Richard) \Rightarrow Evil(Richard)$$

$$King(John)$$

$$Greedy(John)$$

$$Brother(Richard, John)$$

- Then can use propositional inference algorithms to obtain conclusions
  - Modus Ponens yields *Evil*(*John*)

$$\frac{\alpha, \quad \alpha \rightarrow \beta}{\beta} \qquad \frac{King(John) \land Greedy(John), \ King(John) \land Greedy(John) \Rightarrow Evil(John)}{Evil(John)}$$

Forward and Backward Chaining

- Have language representing knowledge (FOL) and inference rules (Generalized Modus Ponens)
  - Now study how a reasoning program is constructed
- Generalized Modus Ponens can be used in two ways:
  - Start with sentences in KB and generate new conclusions (forward chaining)
    - **“Used when a new fact is added to database and want to generate its consequences”**
      or
  - Start with something want to prove, find implication sentences that allow to conclude it, then attempt to establish their premises in turn (backward chaining)
    - **“Used when there is a goal to be proved”**

Forward Chaining

- Forward chaining normally triggered by addition of <u>new</u> fact to KB (using TELL)
- When new fact *p* added to KB:
  - For each rule such that *p* unifies with a premise
    - If the other premises are <u>known</u>, then add the conclusion to the KB and continue chaining
  - Premise: Left-hand side of implication
    - Or, each term of conjunction on left hand side
  - Conclusion: Right-hand side of implication
- Forward chaining uses unification
  - Make two sentences (fact + premise) match by substituting variables (if possible)
- Forward chaining is <u>data-driven</u>
  - Inferring properties and categories from percepts

Forward Chaining Example

Knowledge Base

$A \Rightarrow B$
$A \Rightarrow D$
$D \Rightarrow C$
$A \Rightarrow E$
$D \Rightarrow F$
$E \Rightarrow G$

Add A:

A, $A \Rightarrow B$ gives B [done]
A, $A \Rightarrow D$ gives D
    D, $D \Rightarrow C$ gives C [done]
    D, $D \Rightarrow F$ gives F [done]
A, $A \Rightarrow E$ gives E
    E, $E \Rightarrow G$ gives G [done]
[done]

Order of generation B, D, C, F, E, G

Backward Chaining

- Backward chaining designed to find all answers to a question posed to KB (using ASK)
- When a query *q* is asked:
  - If a matching fact *q*′ is known, return the unifier
  - For each rule whose consequent *q*′ matches *q*
    - Attempt to prove each premise of the rule by backward chaining
- Added complications
  - Keeping track of unifiers, avoiding infinite loops
- Two versions
  - Find any solution
  - Find all solutions
- Backward chaining is basis of logic programming
  - Prolog

Backward Chaining Example

Given facts/rules 1-5 in KB:

1. $Pig(y) \wedge Slug(z) \Rightarrow Faster(y, z)$
2. $Slimy(z) \wedge Creeps(z) \Rightarrow Slug(z)$
3. $Pig(Pat)$
4. $Slimy(Steve)$
5. $Creeps(Steve)$

Prove: $Faster(Pat, Steve)$

$\boxed{Faster(Pat, Steve)}$

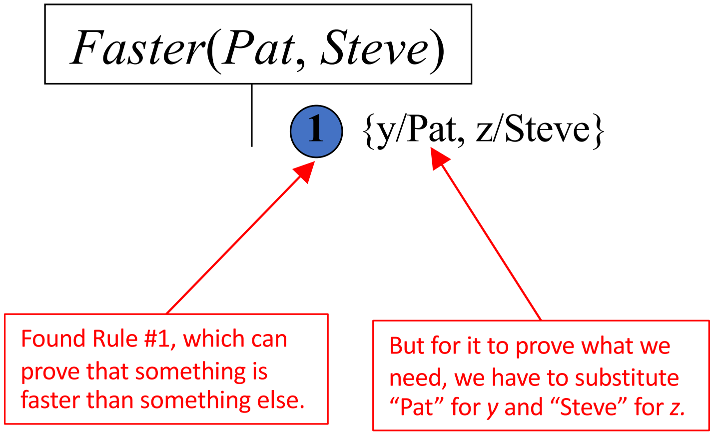
Start with what we want to prove.

Backward Chaining Example

Given facts/rules 1-5 in KB:

1. $Pig(y) \wedge Slug(z) \Rightarrow Faster(y, z)$
2. $Slimy(z) \wedge Creeps(z) \Rightarrow Slug(z)$
3. $Pig(Pat)$
4. $Slimy(Steve)$
5. $Creeps(Steve)$

Prove: *Faster(Pat, Steve)*

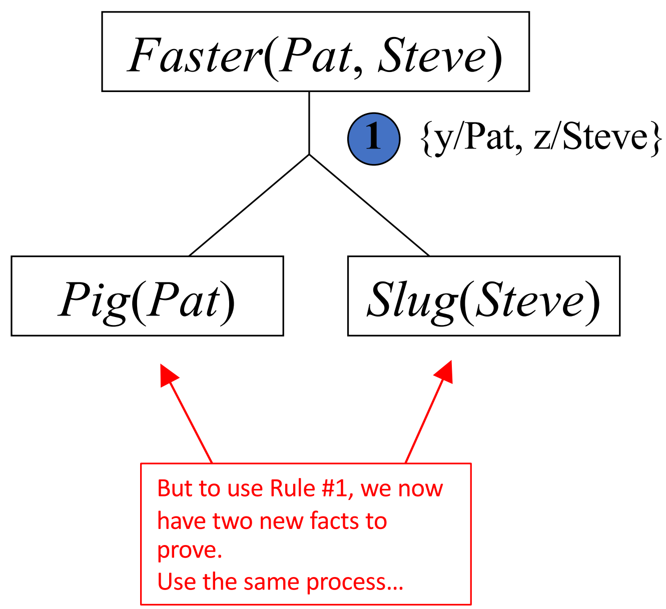


Backward Chaining Example

Given facts/rules 1-5 in KB:

1. $Pig(y) \wedge Slug(z) \Rightarrow Faster(y, z)$
2. $Slimy(z) \wedge Creeps(z) \Rightarrow Slug(z)$
3. $Pig(Pat)$
4. $Slimy(Steve)$
5. $Creeps(Steve)$

Prove: $Faster(Pat, Steve)$

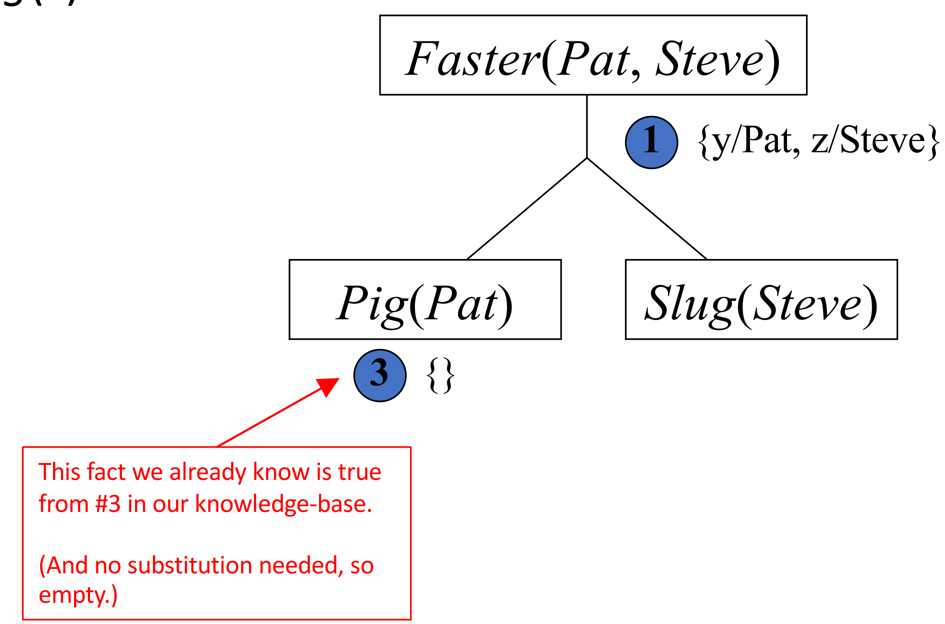


Backward Chaining Example

Given facts/rules 1-5 in KB:

1. $Pig(y) \wedge Slug(z) \Rightarrow Faster(y, z)$
2. $Slimy(z) \wedge Creeps(z) \Rightarrow Slug(z)$
3. $Pig(Pat)$
4. $Slimy(Steve)$
5. $Creeps(Steve)$

Prove: *Faster(Pat, Steve)*

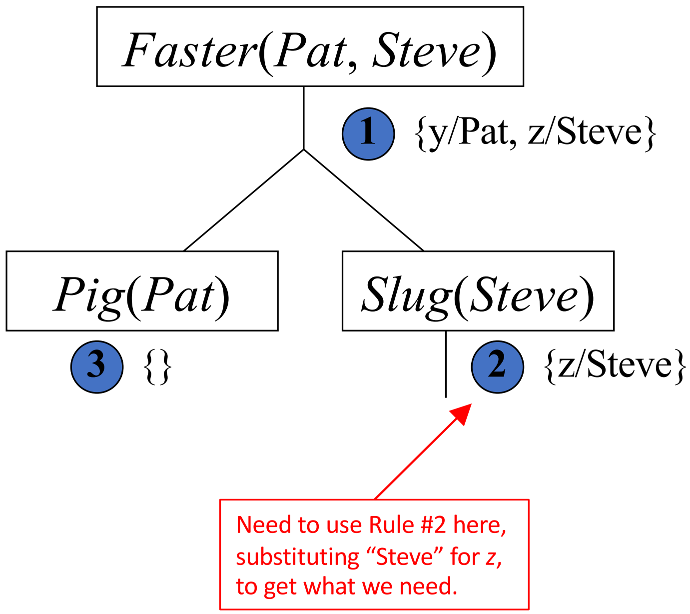


Backward Chaining Example

Given facts/rules 1-5 in KB:

1. $Pig(y) \wedge Slug(z) \Rightarrow Faster(y, z)$
2. $Slimy(z) \wedge Creeps(z) \Rightarrow Slug(z)$
3. $Pig(Pat)$
4. $Slimy(Steve)$
5. $Creeps(Steve)$

Prove: $Faster(Pat, Steve)$

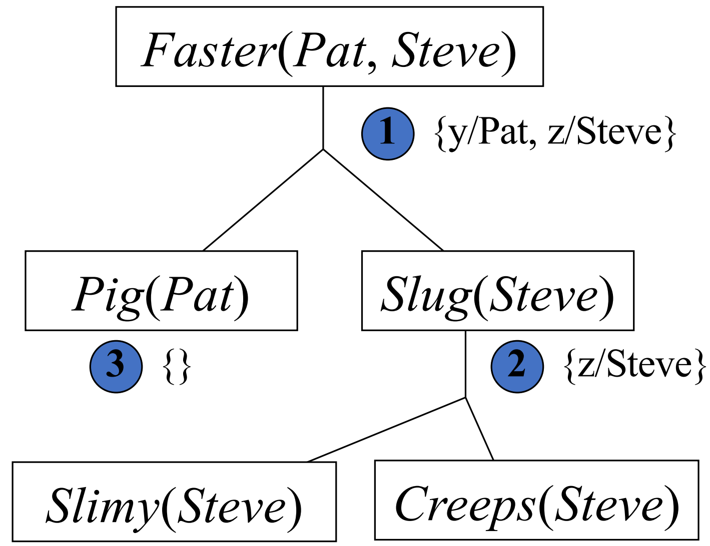


Backward Chaining Example

Given facts/rules 1-5 in KB:

1. $Pig(y) \wedge Slug(z) \Rightarrow Faster(y, z)$
2. $Slimy(z) \wedge Creeps(z) \Rightarrow Slug(z)$
3. $Pig(Pat)$
4. $Slimy(Steve)$
5. $Creeps(Steve)$

Prove: *Faster(Pat, Steve)*



And Rule #2 requires these two facts…

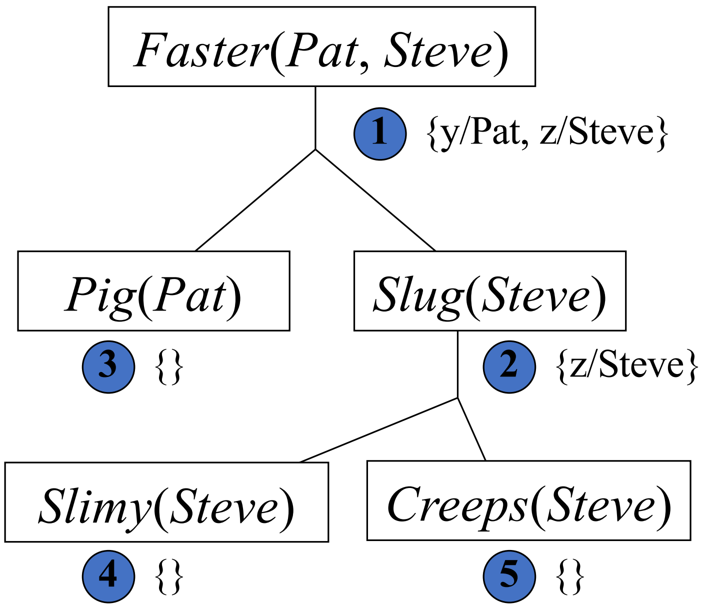
Backward Chaining Example

Given facts/rules 1-5 in KB:

1. $Pig(y) \wedge Slug(z) \Rightarrow Faster(y, z)$
2. $Slimy(z) \wedge Creeps(z) \Rightarrow Slug(z)$
3. $Pig(Pat)$
4. $Slimy(Steve)$
5. $Creeps(Steve)$

Prove: $Faster(Pat, Steve)$

Which we know are true directly from our knowledge-base.



Resolution

- Uses proof by contradiction
  - Referred to by other names
    - Refutation
    - Reductio ad absurdum
- Inference procedure using resolution
  - To prove *P*:
    - Assume *P* is FALSE
    - Add $\neg P$ to KB
    - Prove a contradiction
  - Given that the KB is known to be True, we can believe that the negated goal is in fact False, meaning that the original goal must be True

Resolution Example

KB:

kb-1: $A(x,bar) \vee B(x) \vee C(x)$

kb-2: $D(y,foo) \vee \neg B(y)$

kb-3: $E(z) \vee \neg A(z,bar)$

kb-4: $\neg D(Minsky,foo)$

kb-5: $\neg A(Minsky,bar)$

Goal: prove C(Minsky)

Resolution Example

KB:

- kb-1: $A(x, \text{bar}) \vee B(x) \vee C(x)$
- kb-2: $D(y, \text{foo}) \vee \neg B(y)$
- kb-3: $E(z) \vee \neg A(z, \text{bar})$
- kb-4: $\neg D(\text{Minsky}, \text{foo})$
- kb-5: $\neg A(\text{Minsky}, \text{bar})$

Goal: prove C(Minsky)

0: $\neg C(\text{Minsky})$

Start off using our negated goal (proof by contradiction)

Resolution Example

KB:

- kb-1: $A(x,\text{bar}) \vee B(x) \vee C(x)$
- kb-2: $D(y,\text{foo}) \vee \neg B(y)$
- kb-3: $E(z) \vee \neg A(z,\text{bar})$
- kb-4: $\neg D(\text{Minsky},\text{foo})$
- kb-5: $\neg A(\text{Minsky},\text{bar})$

Goal: prove C(Minsky)

0: $\neg C(\text{Minsky})$

1: $A(\text{Minsky},\text{bar}) \vee B(\text{Minsky}) \vee C(\text{Minsky})$ *[kb-1]*
{x/Minsky}

Look for a rule that has C(Minsky) *to oppose* ¬C(Minsky) *from #0.*
This rule (kb-1) needed a substitution for it to work, giving us the new sentence #1.

Resolution Example

KB:

- kb-1: $A(x,\text{bar}) \vee B(x) \vee C(x)$
- kb-2: $D(y,\text{foo}) \vee \neg B(y)$
- kb-3: $E(z) \vee \neg A(z,\text{bar})$
- kb-4: $\neg D(\text{Minsky},\text{foo})$
- kb-5: $\neg A(\text{Minsky},\text{bar})$

Goal: prove C(Minsky)

0: $\neg C(\text{Minsky})$

1: $A(\text{Minsky},\text{bar}) \vee B(\text{Minsky}) \vee C(\text{Minsky})$ *[kb-1]*
{x/Minsky}

2: $\neg C(\text{Minsky})$, $A(\text{Minsky},\text{bar}) \vee B(\text{Minsky}) \vee C(\text{Minsky})$
2.a: $A(\text{Minsky},\text{bar}) \vee B(\text{Minsky})$ *[resolution: 0,1]*

Now that we have #0 and #1 with opposing terms, use resolution to eliminate them.

Resolution Example

KB:

- kb-1: $A(x,\text{bar}) \vee B(x) \vee C(x)$
- kb-2: $D(y,\text{foo}) \vee \neg B(y)$
- kb-3: $E(z) \vee \neg A(z,\text{bar})$
- kb-4: $\neg D(\text{Minsky},\text{foo})$
- kb-5: $\neg A(\text{Minsky},\text{bar})$

Goal: prove C(Minsky)

0: $\neg C(\text{Minsky})$

1: $A(\text{Minsky},\text{bar}) \vee B(\text{Minsky}) \vee C(\text{Minsky})$ *[kb-1]*
{x/Minsky}

2: $\neg C(\text{Minsky})$, $A(\text{Minsky},\text{bar}) \vee B(\text{Minsky}) \vee C(\text{Minsky})$
2.a: $A(\text{Minsky},\text{bar}) \vee B(\text{Minsky})$ *[resolution: 0,1]*

3: $D(\text{Minsky},\text{foo}) \vee \neg B(\text{Minsky})$ *[kb-2]*
{y/Minsky}

4: $A(\text{Minsky},\text{bar}) \vee B(\text{Minsky})$, $D(\text{Minsky},\text{foo}) \vee \neg B(\text{Minsky})$
4.a: $A(\text{Minsky},\text{bar}) \vee D(\text{Minsky},\text{foo})$ *[resol: 2a,3]*

And repeat to find and eliminate other opposing terms.

Resolution Example

KB:

kb-1: $A(x,\text{bar}) \vee B(x) \vee C(x)$

kb-2: $D(y,\text{foo}) \vee \neg B(y)$

kb-3: $E(z) \vee \neg A(z,\text{bar})$

kb-4: $\neg D(\text{Minsky},\text{foo})$

kb-5: $\neg A(\text{Minsky},\text{bar})$

Goal: prove C(Minsky)

0: $\neg C(\text{Minsky})$

1: $A(\text{Minsky},\text{bar}) \vee B(\text{Minsky}) \vee C(\text{Minsky})$ *[kb-1]*
{x/Minsky}

2: $\neg C(\text{Minsky})$, $A(\text{Minsky},\text{bar}) \vee B(\text{Minsky}) \vee C(\text{Minsky})$
2.a: $A(\text{Minsky},\text{bar}) \vee B(\text{Minsky})$ *[resolution: 0,1]*

3: $D(\text{Minsky},\text{foo}) \vee \neg B(\text{Minsky})$ *[kb-2]*
{y/Minsky}

4: $A(\text{Minsky},\text{bar}) \vee B(\text{Minsky})$, $D(\text{Minsky},\text{foo}) \vee \neg B(\text{Minsky})$
4.a: $A(\text{Minsky},\text{bar}) \vee D(\text{Minsky},\text{foo})$ *[resol: 2a,3]*

5: $\neg A(\text{Minsky},\text{bar})$, $A(\text{Minsky},\text{bar}) \vee D(\text{Minsky},\text{foo})$
5.a: $D(\text{Minsky},\text{foo})$ *[resol: 4a,kb-5]*

And again…

Resolution Example

KB:

kb-1: $A(x,\text{bar}) \vee B(x) \vee C(x)$

kb-2: $D(y,\text{foo}) \vee \neg B(y)$

kb-3: $E(z) \vee \neg A(z,\text{bar})$

kb-4: $\neg D(\text{Minsky},\text{foo})$

kb-5: $\neg A(\text{Minsky},\text{bar})$

Goal: prove C(Minsky)

0: $\neg C(\text{Minsky})$

1: $A(\text{Minsky},\text{bar}) \vee B(\text{Minsky}) \vee C(\text{Minsky})$ *[kb-1]*
{x/Minsky}

2: $\neg C(\text{Minsky})$, $A(\text{Minsky},\text{bar}) \vee B(\text{Minsky}) \vee C(\text{Minsky})$
2.a: $A(\text{Minsky},\text{bar}) \vee B(\text{Minsky})$ *[resolution: 0,1]*

3: $D(\text{Minsky},\text{foo}) \vee \neg B(\text{Minsky})$ *[kb-2]*
{y/Minsky}

4: $A(\text{Minsky},\text{bar}) \vee B(\text{Minsky})$, $D(\text{Minsky},\text{foo}) \vee \neg B(\text{Minsky})$
4.a: $A(\text{Minsky},\text{bar}) \vee D(\text{Minsky},\text{foo})$ *[resol: 2a,3]*

5: $\neg A(\text{Minsky},\text{bar})$, $A(\text{Minsky},\text{bar}) \vee D(\text{Minsky},\text{foo})$
5.a: $D(\text{Minsky},\text{foo})$ *[resol: 4a,kb-5]*

6: $D(\text{Minsky},\text{foo}) \wedge \neg D(\text{Minsky},\text{foo})$

FALSE, CONTRADICTION!!!
must be C(Minsky)

Decision Tree

ID3 Algorithm

```
ID3 (S, A, V)
    Let:
        S = Learning Set
        A = Attibute Set
        V = Attribute Values

    Begin
        Load learning sets  and create decision tree root node(rootNode),
        Add learning set S into root not as its subset

        For rootNode, compute Entropy(rootNode.subset)

        If Entropy(rootNode.subset) == 0 (subset is homogeneous)
            return a leaf node

        If Entropy(rootNode.subset)!= 0 (subset is not homogeneous)
            compute Information Gain for each attribute left (not been used for spliting)
            Find attibute A with Maximum(Gain(S,A))
            Create child nodes for this root node and add to rootNode in the decision tree

        For each child of the rootNode
            Apply ID3(S,A,V)
            Continue until a node with Entropy of 0 or a leaf node is reached
    End
```

Decision Tree Equations

$$Entropy(S) = \sum_{i=1}^{c} -p_i log_2 p_i$$

S= Collection of Examples
P_i = proportion of S that belongs to class i

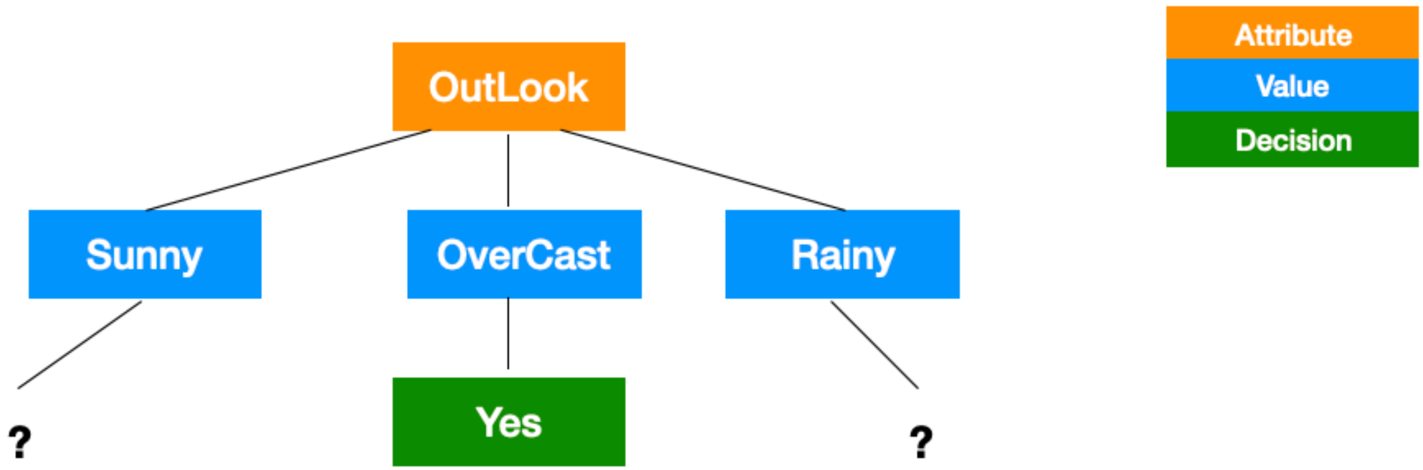
$$Gain(S, A) = Entropy(S) - \sum_{v \in values(A)} \frac{|S_v|}{|S|} Entropy(S_v)$$

A= Attribute
values(A) = set of all possible values for A
|S| = number of example in S
$|S_v|$ =number of example in S where the value of A is v

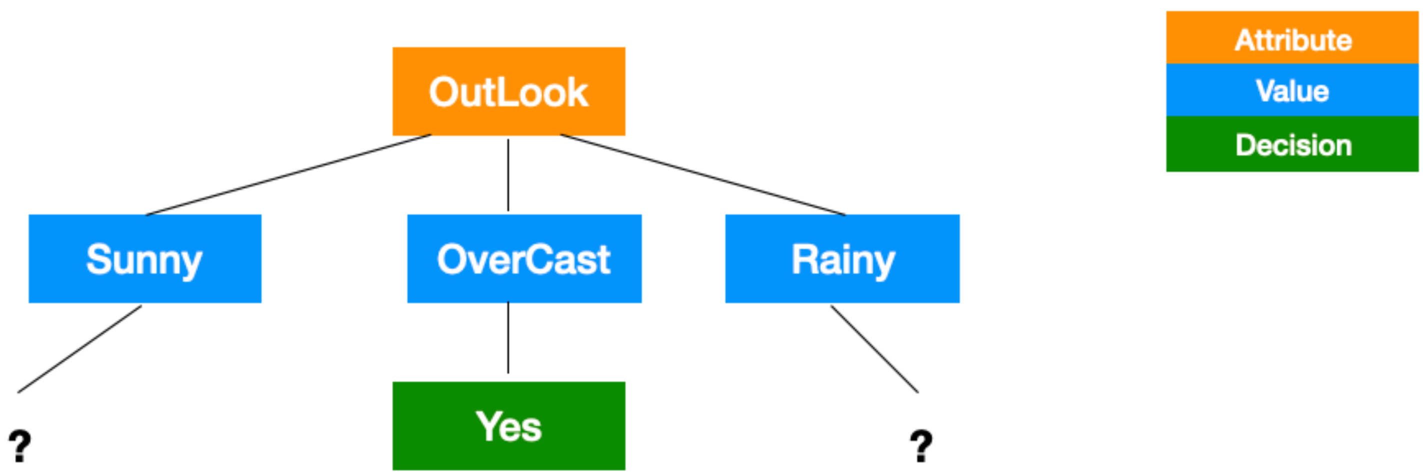
Simple Training Dataset

Day	Outlook	Temperature	Humidity	Wind	PlayGolf?
D1	Sunny	Hot	High	Weak	No
D2	Sunny	Hot	High	Strong	No
D3	Overcast	Hot	High	Weak	Yes
D4	Rain	Mild	High	Weak	Yes
D5	Rain	Cool	Normal	Weak	Yes
D6	Rain	Cool	Normal	Strong	No
D7	Overcast	Cool	Normal	Strong	Yes
D8	Sunny	Mild	High	Weak	No
D9	Sunny	Cool	Normal	Weak	Yes
D10	Rain	Mild	Normal	Weak	Yes
D11	Sunny	Mild	Normal	Strong	Yes
D12	Overcast	Mild	High	Strong	Yes
D13	Overcast	Hot	Normal	Weak	Yes
D14	Rain	Mild	High	Strong	No

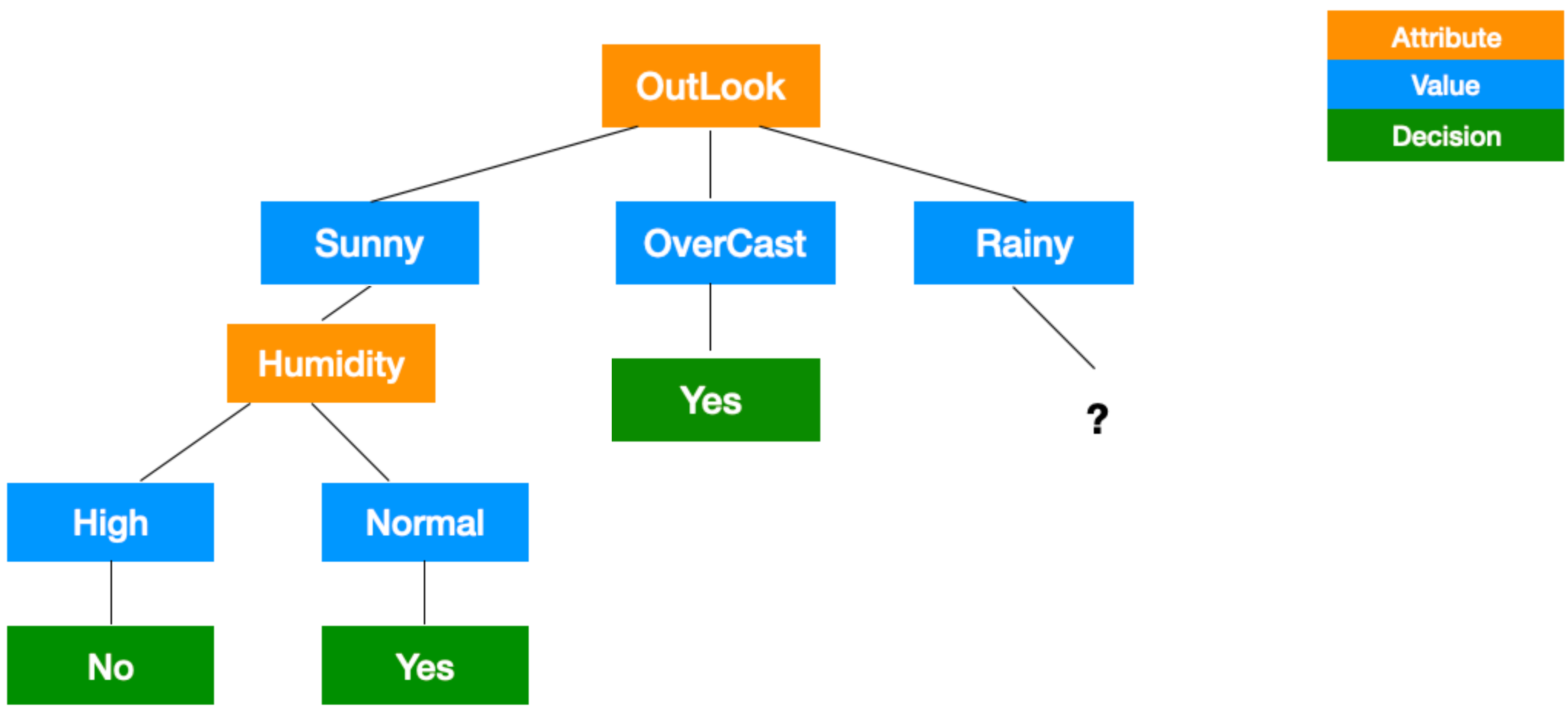
Simple Training Dataset



Simple Training Dataset



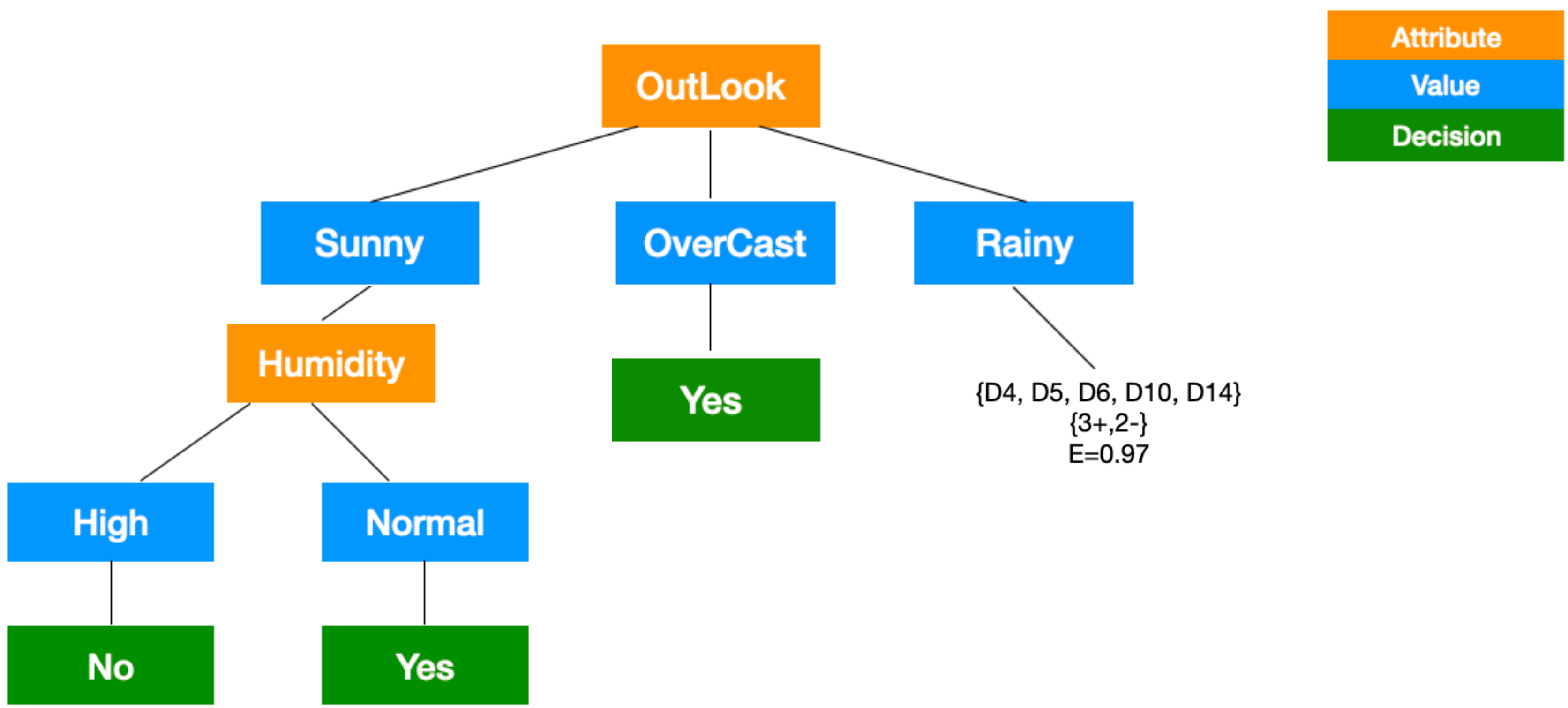
Simple Training Dataset



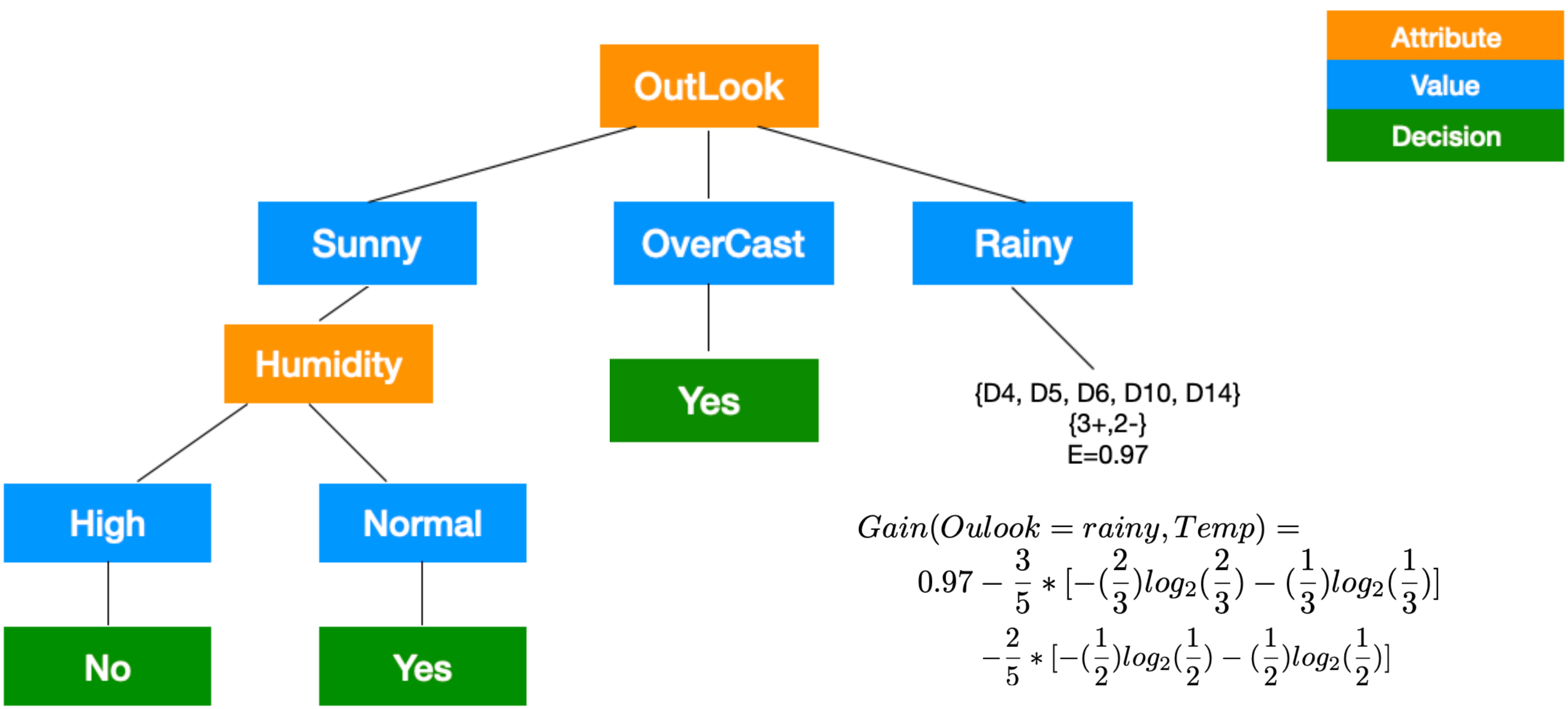
Simple Training Dataset

Day	Outlook	Temperature	Humidity	Wind	PlayGolf?
D1	Sunny	Hot	High	Weak	No
D2	Sunny	Hot	High	Strong	No
D3	Overcast	Hot	High	Weak	Yes
D4	Rain	Mild	High	Weak	Yes
D5	Rain	Cool	Normal	Weak	Yes
D6	Rain	Cool	Normal	Strong	No
D7	Overcast	Cool	Normal	Strong	Yes
D8	Sunny	Mild	High	Weak	No
D9	Sunny	Cool	Normal	Weak	Yes
D10	Rain	Mild	Normal	Weak	Yes
D11	Sunny	Mild	Normal	Strong	Yes
D12	Overcast	Mild	High	Strong	Yes
D13	Overcast	Hot	Normal	Weak	Yes
D14	Rain	Mild	High	Strong	No

Simple Training Dataset

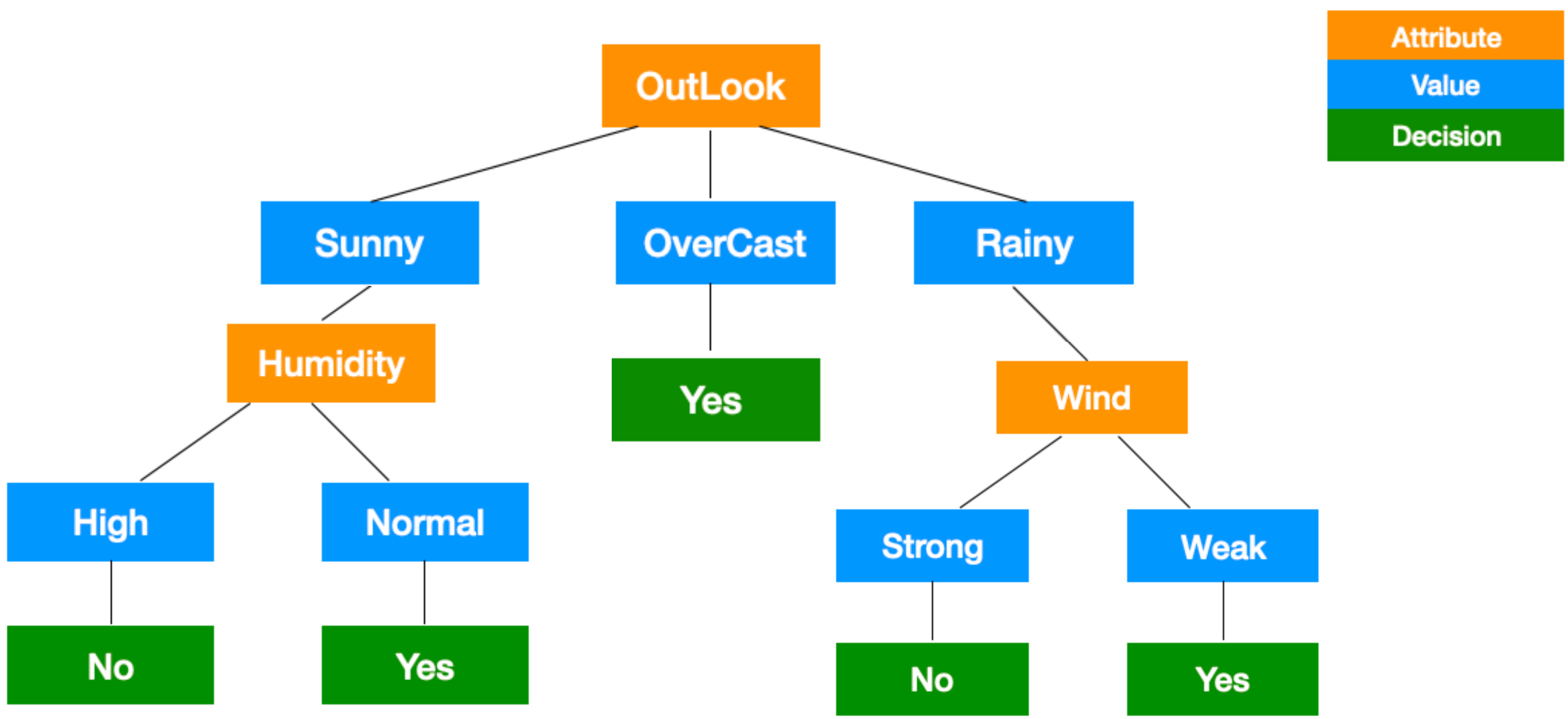


Simple Training Dataset



$$
\begin{aligned}
&Gain(Oulook = rainy, Temp) = \\
&\quad 0.97 - \frac{3}{5} * [-(\frac{2}{3})log_2(\frac{2}{3}) - (\frac{1}{3})log_2(\frac{1}{3})] \\
&\quad -\frac{2}{5} * [-(\frac{1}{2})log_2(\frac{1}{2}) - (\frac{1}{2})log_2(\frac{1}{2})] \\
&\quad = 0.024
\end{aligned}
$$

Simple Training Dataset



Continuous Valued Attribute

Create a discrete attribute to test continuous

- $Temperature = 82.5$
- $(Temperature > 72.3) = t, f$

Temperature:	40	48	60	72	80	90
Play Golf:	No	No	Yes	Yes	Yes	No

Unknown Attribute

Attributes | Classes

Outlook	Temperature	Humidity	Windy	Play Golf
Rainy	Hot	High	FALSE	No
Rainy	Hot		TRUE	No
Overcast	Hot	High	FALSE	Yes
Sunny	Mild	High	FALSE	Yes
Sunny	Cool	Normal	FALSE	Yes
Sunny	Cool	Normal	TRUE	No
Overcast	Cool	Normal	TRUE	Yes
Rainy	Mild	High	FALSE	No
Rainy	Cool	Normal		Yes
Sunny	Mild	Normal	FALSE	Yes
Rainy	Mild	Normal	TRUE	Yes
Overcast			TRUE	Yes
Overcast	Hot	Normal	FALSE	Yes
Sunny	Mild	High	TRUE	No

Unknown Attribute

Outlook	Temperature	Humidity	Windy	Play Golf
Rainy	Hot	High	FALSE	No
Rainy	Hot		TRUE	No
Overcast	Hot	High	FALSE	Yes
Sunny	Mild	High	FALSE	Yes
Sunny	Cool	Normal	FALSE	Yes
Sunny	Cool	Normal	TRUE	No
Overcast	Cool	Normal	TRUE	Yes
Rainy	Mild	High	FALSE	No
Rainy	Cool	Normal		Yes
Sunny	Mild	Normal	FALSE	Yes
Rainy	Mild	Normal	TRUE	Yes
Overcast			TRUE	Yes
Overcast	Hot	Normal	FALSE	Yes
Sunny	Mild	High	TRUE	No

Unknown Attribute

Outlook	Temperature	Humidity	Windy	Play Golf
Rainy	Hot	High	FALSE	No
Rainy	Hot		TRUE	No
Overcast	Hot	High	FALSE	Yes
Sunny	Mild	High	FALSE	Yes
Sunny	Cool	Normal	FALSE	Yes
Sunny	Cool	Normal	TRUE	No
Overcast	Cool	Normal	TRUE	Yes
Rainy	Mild	High	FALSE	No
Rainy	Cool	Normal		Yes
Sunny	Mild	Normal	FALSE	Yes
Rainy	Mild	Normal	TRUE	Yes
Overcast			TRUE	Yes
Overcast	Hot	Normal	FALSE	Yes
Sunny	Mild	High	TRUE	No

Attributes: Outlook, Temperature, Humidity, Windy. Classes: Play Golf.

#High=5
#Normal = 7

Unknown Attribute

Attributes Classes

Outlook	Temperature	Humidity	Windy	Play Golf
Rainy	Hot	High	FALSE	No
Rainy	Hot	Normal	TRUE	No
Overcast	Hot	High	FALSE	Yes
Sunny	Mild	High	FALSE	Yes
Sunny	Cool	Normal	FALSE	Yes
Sunny	Cool	Normal	TRUE	No
Overcast	Cool	Normal	TRUE	Yes
Rainy	Mild	High	FALSE	No
Rainy	Cool	Normal		Yes
Sunny	Mild	Normal	FALSE	Yes
Rainy	Mild	Normal	TRUE	Yes
Overcast			TRUE	Yes
Overcast	Hot	Normal	FALSE	Yes
Sunny	Mild	High	TRUE	No

#High=5
#Normal = 7

Unknown Attribute

Attributes / Classes

Outlook	Temperature	Humidity	Windy	Play Golf
Rainy	Hot	High	FALSE	No
Rainy	Hot		TRUE	No
Overcast	Hot	High	FALSE	Yes
Sunny	Mild	High	FALSE	Yes
Sunny	Cool	Normal	FALSE	Yes
Sunny	Cool	Normal	TRUE	No
Overcast	Cool	Normal	TRUE	Yes
Rainy	Mild	High	FALSE	No
Rainy	Cool	Normal		Yes
Sunny	Mild	Normal	FALSE	Yes
Rainy	Mild	Normal	TRUE	Yes
Overcast			TRUE	Yes
Overcast	Hot	Normal	FALSE	Yes
Sunny	Mild	High	TRUE	No

For PlayGlolf=No
#High=3
#Normal = 1

Unknown Attribute

Outlook	Temperature	Humidity	Windy	Play Golf
Rainy	Hot	High	FALSE	No
Rainy	Hot	**High**	TRUE	No
Overcast	Hot	High	FALSE	Yes
Sunny	Mild	High	FALSE	Yes
Sunny	Cool	Normal	FALSE	Yes
Sunny	Cool	Normal	TRUE	No
Overcast	Cool	Normal	TRUE	Yes
Rainy	Mild	High	FALSE	No
Rainy	Cool	Normal		Yes
Sunny	Mild	Normal	FALSE	Yes
Rainy	Mild	Normal	TRUE	Yes
Overcast			TRUE	Yes
Overcast	Hot	Normal	FALSE	Yes
Sunny	Mild	High	TRUE	No

For PlayGlolf=No
#High=3
#Normal = 1

Unknown Attribute Values

What if some examples are missing values of A?

Use training example anyway, sort through tree

- If node n tests A, assign most common value of A among other examples sorted to node n
- Assign most common value of A among other examples with same target value

Supervised Learning

- Split the data in 2 parts
  - Train
    - To train the model
  - Test
    - To evaluate the performance
      - How many of the test data prediction is correct

Classification vs. regression

- Classification
  - Supervised learning
  - Form: Data point → Desired category (integer number index)
  - Ex: 1: Cat, 2: Dog, 3: Horse, ……, 1000: Car

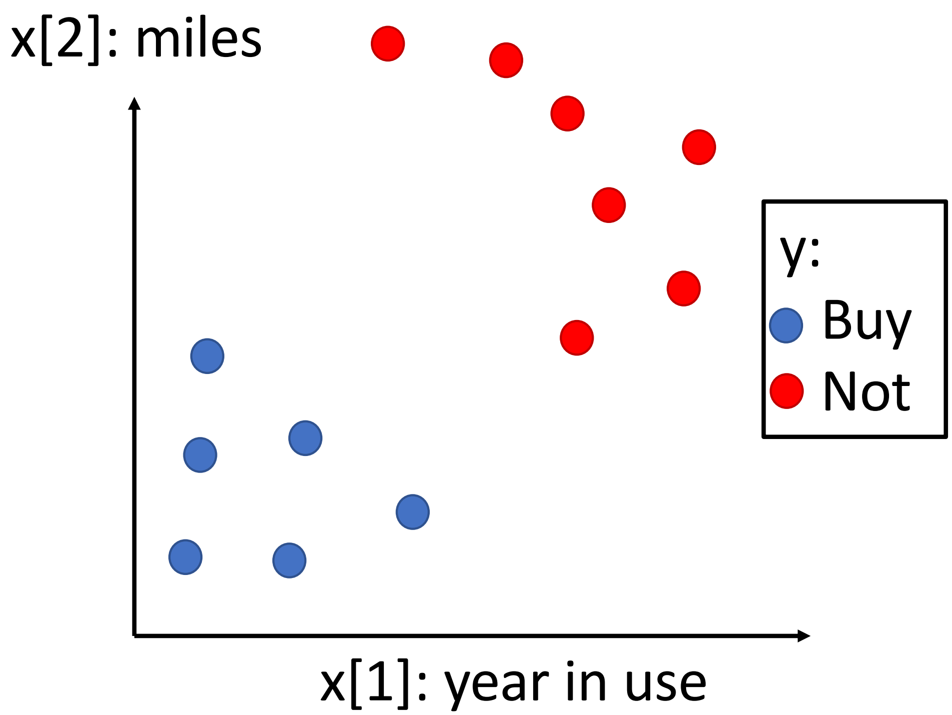
- Regression (curve Fitting)
  - Supervised learning
  - Form: Data point → Desired real number (e.g., price, chance, etc.)

Machine learning: capture patterns from training data that can be generalized to future data

Classification vs. regression: training data

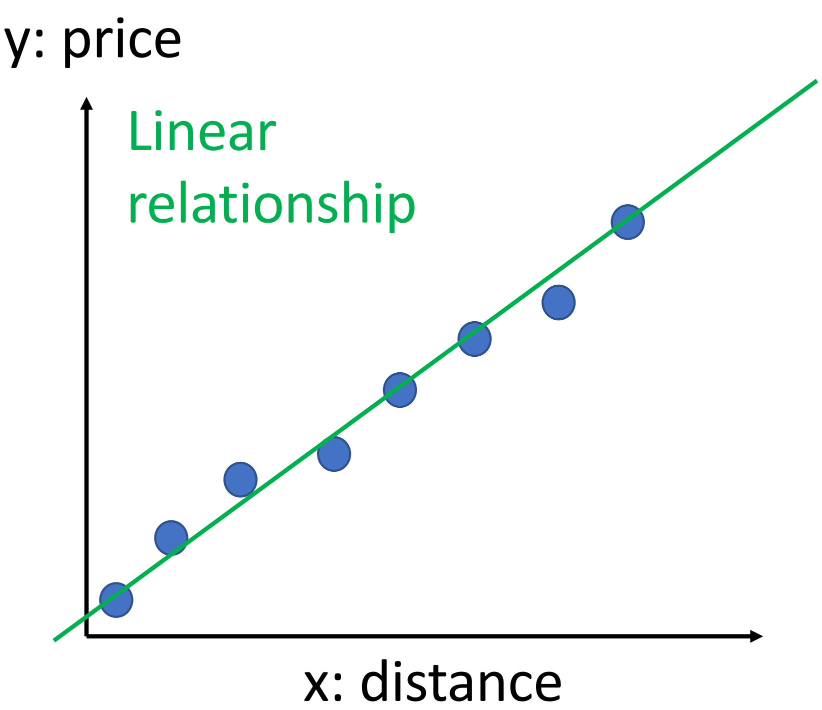
Regression (bus ticket):
From x (distance), predict y (price)

Classification (car buying company):
From ***x*** (year, miles), predict y (buy or not)

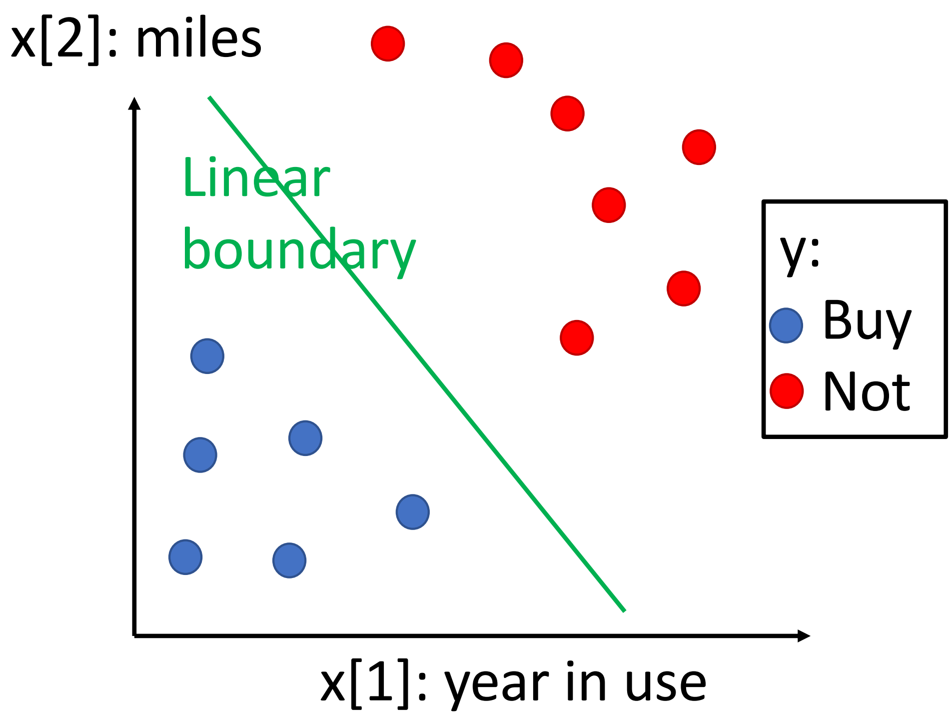


Classification vs. regression: find patterns

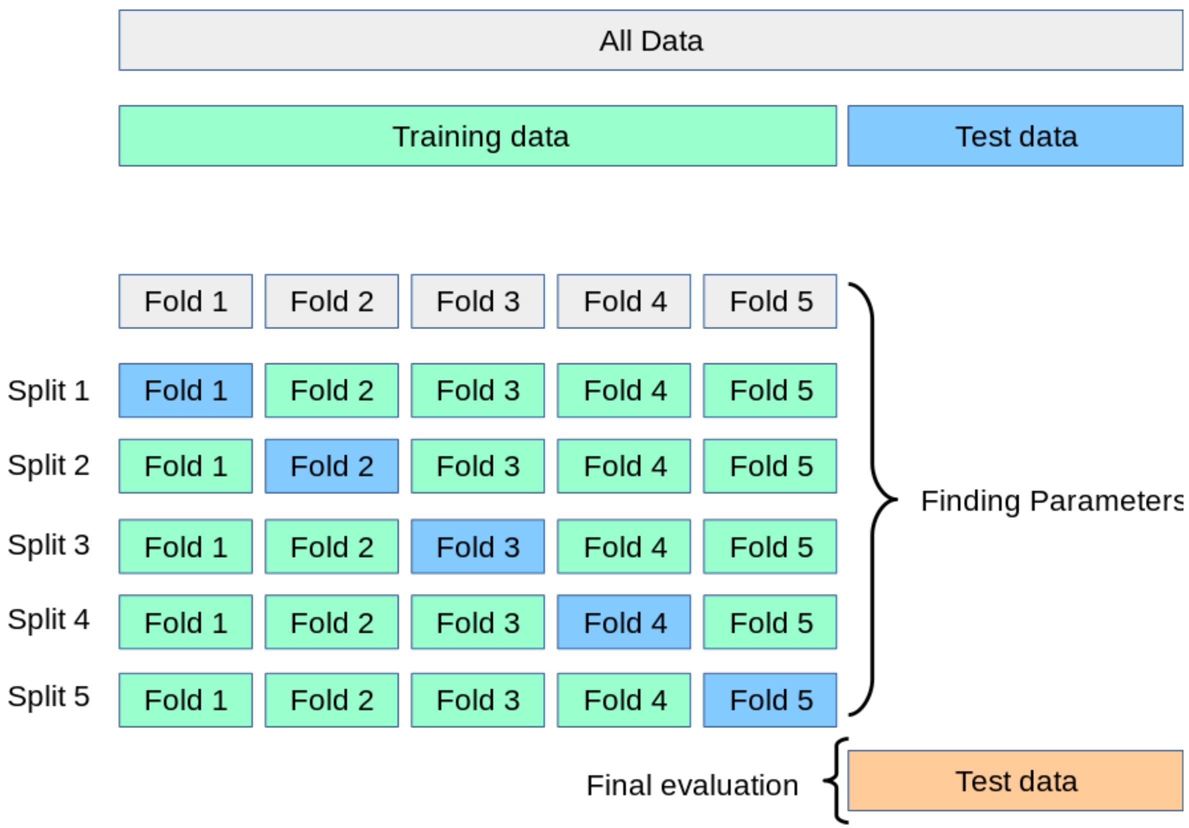
Regression (bus ticket):
From x (distance), predict y (price)



Classification (car buying company):
From ***x*** (year, miles), predict y (buy or not)



Cross validation



Bag of Words

Bag of Words (BOW)

Given:

- A dictionary, vocabulary, or codebook: f(token) → index ∈ $\{1, \cdots, D\}$ or N/A
- An all-0 vector: $\boldsymbol{x} = \begin{bmatrix} x[1] \\ \vdots \\ x[D] \end{bmatrix} = \begin{bmatrix} 0 \\ \vdots \\ 0 \end{bmatrix}$

Bag of Words (BOW)

Given:

- A dictionary, vocabulary, or codebook: f(token) → index $\in \{1, \cdots, D\}$ or N/A
- An all-0 vector: $\boldsymbol{x} = \begin{bmatrix} x[1] \\ \vdots \\ x[D] \end{bmatrix} = \begin{bmatrix} 0 \\ \vdots \\ 0 \end{bmatrix}$

Input: A sequence of word tokens w[1], w[2], ….. w[M]

for m = 1 : M

  if f(w[m]) != N/A

    $x[\mathrm{f(w[m])}]$ += 1

  end

end

Return: $\boldsymbol{x}$

Bag of Words (BOW)

Given:

- A <u>dictionary, vocabulary, or codebook</u>: f(token) → index ∈ $\{1, \cdots, D\}$ or N/A
- An all-0 vector: $\boldsymbol{x} = \begin{bmatrix} x[1] \\ \vdots \\ x[D] \end{bmatrix} = \begin{bmatrix} 0 \\ \vdots \\ 0 \end{bmatrix}$

Input: A sequence of word tokens w[1], w[2], ….. w[M]

for m = 1 : M

  if f(w[m]) != N/A

    x[f(w[m])] += 1

  end

end

Return: $\boldsymbol{x}$

- **Out-of-vocabulary**
- **"Stop" words:** too frequent in all sentences/documents and less informative in differentiating them (e.g., "is", "are", "and", ……)

Word token counts

Naïve Bayes

Bayes Rules

Bayes Rule tells us how to flip the conditional
Reason about effects to causes
Useful if you assume a generative model for your data

Likelihood

Prior

$$P(H|D) = \frac{P(D|H)P(H)}{\sum_h P(D|H)P(H)}$$

Posterior

Normalizer

Bayes Rules

Bayes Rule tells us how to flip the conditional
Reason about effects to causes
Useful if you assume a generative model for your data

Likelihood

Prior

$$P(H|D) \propto P(D|H)P(H)$$

Posterior

Proportional To
(Doesn't sum to 1)

Classification Definition

- *Input*:
  - a document d
  - a fixed set of classes $C = \{c_1, c_2, \ldots, c_J\}$
- *Output*: a predicted class $c \in C$

Bayes Rule Applied to Documents

For a document d and a class c

$$P(c \mid d) = \frac{P(d \mid c)P(c)}{P(d)}$$

Bayes Rule Applied to Documents

For a document d and a class c

Posterior $\longrightarrow$ $$P(c \mid d) = \frac{P(d \mid c)P(c)}{P(d)}$$

$$c_{MAP} = \operatorname*{argmax}_{c \in C} P(c \mid d)$$

Naïve Bayes Classifier

$$c_{MAP} = \underset{c \in C}{\operatorname{argmax}}\, P(c \mid d)$$

MAP is "maximum a posteriori" = most likely class

$$= \underset{c \in C}{\operatorname{argmax}} \frac{P(d \mid c) P(c)}{P(d)}$$

Bayes Rule

$$= \underset{c \in C}{\operatorname{argmax}}\, P(d \mid c) P(c)$$

Dropping the denominator

$$= \underset{c \in C}{\operatorname{argmax}}\, P(x_1, x_2, \ldots, x_n \mid c) P(c)$$

Naïve Bayes Classifier

$$c_{MAP} = \underset{c \in C}{\operatorname{argmax}} P(x_1, x_2, \ldots, x_n \mid c) P(c)$$

How often does this class occur?

We can just count the relative frequencies in a corpus

Could only be estimated if a very, very large number of training examples was available.

Naïve Bayes Classifier: Independence Assumption

- **Bag of Words assumption**: Assume position doesn't matter
- **Conditional Independence**: Assume the feature probabilities $P(x_i|c_j)$ are independent given the class c_j.

$$c_{MAP} = \underset{c \in C}{\operatorname{argmax}}\, P(x_1, x_2, \ldots, x_n \mid c) P(c)$$

$$c_{NB} = \underset{c \in C}{\operatorname{argmax}}\, P(c_j) \prod_{x \in X} P(x \mid c)$$

Learning: Naïve Bayes Classifier

- First attempt: maximum likelihood estimates
  - simply use the frequencies in the data

$$\hat{P}(c_j) = \frac{doccount(C = c_j)}{N_{doc}}$$

$$\hat{P}(w_i \mid c_j) = \frac{count(w_i, c_j)}{\sum_{w \in V} count(w, c_j)}$$

Learning: Naïve Bayes Parameter Estimation

$$\hat{P}(w_i \mid c_j) = \frac{count(w_i, c_j)}{\sum_{w \in V} count(w, c_j)}$$

fraction of times word w_i appears among all words in documents of topic c_j

fraction of word in the full vocabulary that appeared in topic

Learning: Naïve Bayes Parameter Estimation

$$\hat{P}(w_i \mid c_j) = \frac{count(w_i, c_j)}{\sum_{w \in V} count(w, c_j)}$$

fraction of times word w_i appears among all words in documents of topic c_j

fraction of word in the full vocabulary that appeared in topic

- What if we have seen no training documents with the word ***fantastic*** in the topic **positive**?
- Zero probabilities cannot be conditioned away

Smoothing: Naïve Bayes Parameter Estimation

$$\hat{P}(w_i|c) = \frac{count(w_i, c)}{\sum_{w \in V} count(w_i, c)}$$

$$\hat{P}(w_i|c) = \frac{count(w_i, c) + \alpha}{\sum_{w \in V} count(w_i, c) + \alpha|V|}$$

Laplace Smoothing: Naïve Bayes Parameter Estimation

$$\hat{P}(w_i|c) = \frac{count(w_i, c)}{\sum_{w \in V} count(w_i, c)}$$

$$\hat{P}(w_i|c) = \frac{count(w_i, c) + \alpha}{\sum_{w \in V} count(w_i, c) + \alpha|V|}$$

$$\hat{P}(w_i|c) = \frac{count(w_i, c) + 1}{\sum_{w \in V} count(w_i, c) + |V|}$$

Naïve Bayes Parameter Learning: Step

- Calculate $P(c_j)$ terms
  - For each c_j in C do

    $docs_j \leftarrow$ all docs with class = c_j

    $$P(c_j) \leftarrow \frac{|docs_j|}{|\text{total \# documents}|}$$

- From training corpus, extract Vocabulary

- Calculate $P(w_k \mid c_j)$ terms
  - $\text{Text}_j \leftarrow$ single doc containing all sentences from class = c_j
  - For each word w_k in *Vocabulary*

    $n_k \leftarrow$ # of occurrences of w_k in $Text_j$

    $n \leftarrow$ # of words in class $Text_j$

    $$P(w_k \mid c_j) \leftarrow \frac{n_k + \alpha}{n + \alpha\,|Vocabulary|}$$

Naïve Bayes Evaluation: Step

- # of correct prediction = 0
- For each t_i in Test data:
  - max_prob = 0
  - For each class c_j
    - Find p_j = probability of t_j to be in c_j
    - If pj > max_prob:
      - max_prob = p_j, max_prob_class= c_j
  - If max_prob_class == actual label of tj:
    - # of correct prediction += 1
- Accuracy = $\frac{\text{\# of correct prediction}}{\text{\# of test data}}$