Final Catch-up, Review

Outline

- Knowledge Representation using First-Order Logic
- ~~Inference in First-Order Logic~~
- Probability, Bayesian Networks
- Machine Learning

- Questions on any topic

- Review pre-mid-term material if time and class interest

Knowledge Representation using First-Order Logic

- Propositional Logic is **Useful** --- but has **Limited Expressive Power**
- First Order Predicate Calculus (FOPC), or First Order Logic (FOL).
  - FOPC has greatly expanded expressive power, though still limited.
- New Ontology
  - The world consists of OBJECTS (for propositional logic, the world was facts).
  - OBJECTS have PROPERTIES and engage in RELATIONS and FUNCTIONS.
- New Syntax
  - Constants, Predicates, Functions, Properties, Quantifiers.
- New Semantics
  - Meaning of new syntax.
- Knowledge engineering in FOL

Review: Syntax of FOL: Basic elements

- Constants KingJohn, 2, UCI,...
- Predicates Brother, >,...
- Functions Sqrt, LeftLegOf,...
- Variables x, y, a, b,...
- Connectives $\neg$, $\Rightarrow$, $\wedge$, $\vee$, $\Leftrightarrow$
- Equality =
- Quantifiers $\forall$, $\exists$

Syntax of FOL: Basic syntax elements are symbols

- **Constant** Symbols:
  - Stand for objects in the world.
    - E.g., KingJohn, 2, UCI, ...

- **Predicate** Symbols
  - Stand for relations (maps a tuple of objects to a **truth-value**)
    - E.g., Brother(Richard, John), greater_than(3,2), ...
  - P(x, y) is usually read as "x is P of y."
    - E.g., Mother(Ann, Sue) is usually "Ann is Mother of Sue."

- **Function** Symbols
  - Stand for functions (maps a tuple of objects to an **object**)
    - E.g., Sqrt(3), LeftLegOf(John), ...

- **Model** (world) = set of domain objects, relations, functions
- **Interpretation** maps symbols onto the model (world)
  - Very many interpretations are possible for each KB and world!
  - Job of the KB is to rule out models inconsistent with our knowledge.

Syntax of FOL: Terms

- **Term** = logical expression that **refers to an object**

- **There are two kinds of terms:**

  - **Constant Symbols** stand for (or name) objects:
    - E.g., KingJohn, 2, UCI, Wumpus, ...

  - **Function Symbols** map tuples of objects to an object:
    - E.g., LeftLeg(KingJohn), Mother(Mary), Sqrt(x)
    - This is nothing but a complicated kind of name
      - No “subroutine” call, no “return value”

Syntax of FOL: Atomic Sentences

- **Atomic Sentences** state facts (logical truth values).
  - An **atomic sentence** is a Predicate symbol, optionally followed by a parenthesized list of any argument terms
  - E.g., *Married(Father(Richard), Mother(John))*
  - An **atomic sentence** asserts that some relationship (some predicate) holds among the objects that are its arguments.

- An **Atomic Sentence is true** in a given model if the relation referred to by the predicate symbol holds among the objects (terms) referred to by the arguments.

Syntax of FOL: Connectives & Complex Sentences

- **Complex Sentences** are formed in the same way, and are formed using the same logical connectives, as we already know from propositional logic

- The **Logical Connectives**:
  - $\Leftrightarrow$ biconditional
  - $\Rightarrow$ implication
  - $\wedge$ and
  - $\vee$ or
  - $\neg$ negation

- **Semantics** for these logical connectives are the same as we already know from propositional logic.

Syntax of FOL: Variables

- **Variables** range over objects in the world.
- A **variable** is like a **term** because it represents an object.
- A **variable** may be used wherever a **term** may be used.
  - **Variables** may be arguments to functions and predicates.
- (A **term with NO variables** is called a **ground term**.)
- (A **variable not bound by a quantifier** is called **free**.)

Syntax of FOL: Logical Quantifiers

- There are two **Logical Quantifiers:**
  - **Universal:** $\forall$ x P(x) means "For all x, P(x)."
    - The "upside-down A" reminds you of "ALL."
  - **Existential:** $\exists$ x P(x) means "There exists x such that, P(x)."
    - The "upside-down E" reminds you of "EXISTS."

- Syntactic "sugar" --- we really only need one quantifier.
  - $\forall$ x P(x) $\equiv \neg\exists$ x $\neg$P(x)
  - $\exists$ x P(x) $\equiv \neg\forall$ x $\neg$P(x)
  - You can ALWAYS convert one quantifier to the other.

- **RULES:** $\forall \equiv \neg\exists\neg$ and $\exists \equiv \neg\forall\neg$

- **RULE:** To move negation "in" across a quantifier,
  change the quantifier to "the other quantifier"
  and negate the predicate on "the other side."
  - $\neg\forall$ x P(x) $\equiv \exists$ x $\neg$P(x)
  - $\neg\exists$ x P(x) $\equiv \forall$ x $\neg$P(x)

Semantics: Interpretation

- An **interpretation** of a sentence (wff) is an assignment that maps
  - Object constant symbols to objects in the world,
  - n-ary function symbols to n-ary functions in the world,
  - n-ary relation symbols to n-ary relations in the world

- Given an interpretation, an atomic sentence has the value "true" if it denotes a relation that holds for those individuals denoted in the terms. Otherwise it has the value "false."
  - Example: Kinship world:
    - Symbols = Ann, Bill, Sue, Married, Parent, Child, Sibling, …
  - World consists of individuals in relations:
    - Married(Ann,Bill) is false, Parent(Bill,Sue) is true, …

Combining Quantifiers --- Order (Scope)

The order of “unlike” quantifiers is important.

$\forall x \exists y$ Loves(x,y)

– For everyone (“all x”) there is someone (“exists y”) whom they love

$\exists y \forall x$ Loves(x,y)

- there is someone (“exists y”) whom everyone loves (“all x”)

Clearer with parentheses: $\exists y (\forall x$ Loves(x,y))

The order of “like” quantifiers does not matter.

$\forall x \forall y P(x, y) \equiv \forall y \forall x P(x, y)$

$\exists x \exists y P(x, y) \equiv \exists y \exists x P(x, y)$

De Morgan's Law for Quantifiers

De Morgan's Rule

$$P \wedge Q \equiv \neg(\neg P \vee \neg Q)$$

$$P \vee Q \equiv \neg(\neg P \wedge \neg Q)$$

$$\neg(P \wedge Q) \equiv \neg P \vee \neg Q$$

$$\neg(P \vee Q) \equiv \neg P \wedge \neg Q$$

Generalized De Morgan's Rule

$$\forall x\, P \equiv \neg\exists x\, (\neg P)$$

$$\exists x\, P \equiv \neg\forall x\, (\neg P)$$

$$\neg\forall x\, P \equiv \exists x\, (\neg P)$$

$$\neg\exists x\, P \equiv \forall x\, (\neg P)$$

Rule is simple: if you bring a negation inside a disjunction or a conjunction, always switch between them (or →and, and → or).

Outline

- Knowledge Representation using First-Order Logic
- ~~Inference in First-Order Logic~~
- Probability, Bayesian Networks
- Machine Learning

- Questions on any topic

- Review pre-mid-term material if time and class interest

Inference in First-Order Logic --- Summary

- FOL inference techniques
  - Unification
  - Generalized Modus Ponens
    - Forward-chaining
    - Backward-chaining
  - Resolution-based inference
    - Refutation-complete

Unification

- Recall: Subst(θ, p) = result of substituting θ into sentence p

- Unify algorithm: takes 2 sentences p and q and returns a unifier if one exists

  Unify(p,q) = θ where Subst(θ, p) = Subst(θ, q)

- Example:

  p = Knows(John,x)

  q = Knows(John, Jane)

  Unify(p,q) = {x/Jane}

Unification examples

- simple example: query = Knows(John,x), i.e., who does John know?

p	q	θ
Knows(John,x)	Knows(John,Jane)	{x/Jane}
Knows(John,x)	Knows(y,OJ)	{x/OJ,y/John}
Knows(John,x)	Knows(y,Mother(y))	{y/John,x/Mother(John)}
Knows(John,x)	Knows(x,OJ)	{fail}

- Last unification fails: only because x can't take values John and OJ at the same time
  - But we know that if John knows x, and everyone (x) knows OJ, we should be able to infer that John knows OJ
- Problem is due to use of same variable x in both sentences
- Simple solution: Standardizing apart eliminates overlap of variables, e.g., Knows(z,OJ)

Unification

- To unify *Knows(John,x)* and *Knows(y,z)*,

  θ = {y/John, x/z } or θ = {y/John, x/John, z/John}

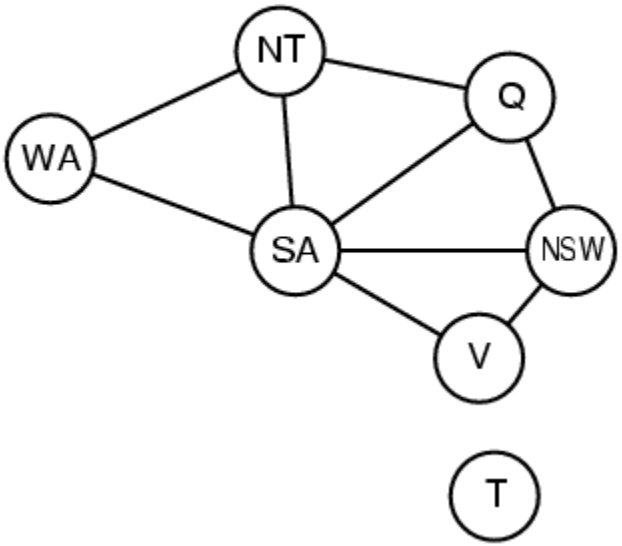
- The first unifier is more general than the second.

- There is a single most general unifier (MGU) that is unique up to renaming of variables.

  MGU = { y/John, x/z }

- General algorithm in Figure 9.1 in the text

Hard matching example



$Diff(wa,nt) \wedge Diff(wa,sa) \wedge Diff(nt,q) \wedge Diff(nt,sa) \wedge Diff(q,nsw) \wedge Diff(q,sa) \wedge Diff(nsw,v) \wedge Diff(nsw,sa) \wedge Diff(v,sa) \Rightarrow Colorable()$

Diff(Red,Blue) *Diff (Red,Green)*
Diff(Green,Red) *Diff(Green,Blue)*
Diff(Blue,Red) *Diff(Blue,Green)*

- To unify the grounded propositions with premises of the implication you need to solve a CSP!
- *Colorable*() is inferred iff the CSP has a solution
- CSPs include 3SAT as a special case, hence matching is NP-hard

Inference appoaches in FOL

- Forward-chaining
  - Uses GMP to add new atomic sentences
  - Useful for systems that make inferences as information streams in
  - Requires KB to be in form of first-order definite clauses

- Backward-chaining
  - Works backwards from a query to try to construct a proof
  - Can suffer from repeated states and incompleteness
  - Useful for query-driven inference
  - Requires KB to be in form of first-order definite clauses

- Resolution-based inference (FOL)
  - Refutation-complete for general KB
    - Can be used to confirm or refute a sentence p (but not to generate all entailed sentences)
  - Requires FOL KB to be reduced to CNF
  - Uses generalized version of propositional inference rule

- Note that all of these methods are generalizations of their propositional equivalents

Generalized Modus Ponens (GMP)

$$\frac{p_1', p_2', \ldots, p_n', (p_1 \wedge p_2 \wedge \ldots \wedge p_n \Rightarrow q)}{Subst(\theta, q)}$$

where we can unify p_i' and p_i for all *i*

Example:

p_1' is *King*(*John*) p_1 is *King*(*x*)

p_2' is *Greedy*(*y*) p_2 is *Greedy*(*x*)

θ is {x/John,y/John} q is *Evil*(*x*)

Subst(θ,q) is *Evil*(*John*)

- Implicit assumption that all variables universally quantified

Completeness and Soundness of GMP

- GMP is sound
  - Only derives sentences that are logically entailed
  - See proof in text on p. 326 (3rd ed.; p. 276, 2nd ed.)

- GMP is complete for a KB consisting of definite clauses
  - Complete: derives all sentences that are entailed
  - OR…answers every query whose answers are entailed by such a KB

  - Definite clause: disjunction of literals of which exactly 1 is positive,
    e.g., King(x) AND Greedy(x) -> Evil(x)
    NOT(King(x)) OR NOT(Greedy(x)) OR Evil(x)

Properties of forward chaining

- Sound and complete for first-order definite clauses
- Datalog = first-order definite clauses + no functions
- FC terminates for Datalog in finite number of iterations
- May not terminate in general if α is not entailed
- Incremental forward chaining: no need to match a rule on iteration *k* if a premise wasn't added on iteration *k-1*

  ⇒ match each rule whose premise contains a newly added positive literal

Properties of backward chaining

- Depth-first recursive proof search:
  - Space is linear in size of proof.

- Incomplete due to infinite loops
  - $\Rightarrow$ fix by checking current goal against every goal on stack

- Inefficient due to repeated subgoals (both success and failure)
  - $\Rightarrow$ fix using caching of previous results (memoization)

- Widely used for logic programming

- PROLOG:
  backward chaining with Horn clauses + bells & whistles.

Resolution in FOL

- Full first-order version:

$$\frac{\ell_1 \vee \cdots \vee \ell_k, \qquad m_1 \vee \cdots \vee m_n}{\text{Subst}(\theta, \ell_1 \vee \cdots \vee \ell_{i-1} \vee \ell_{i+1} \vee \cdots \vee \ell_k \vee m_1 \vee \cdots \vee m_{j-1} \vee m_{j+1} \vee \cdots \vee m_n)}$$

where $\text{Unify}(\ell_i, \neg m_j) = \theta$.

- The two clauses are assumed to be standardized apart so that they share no variables.

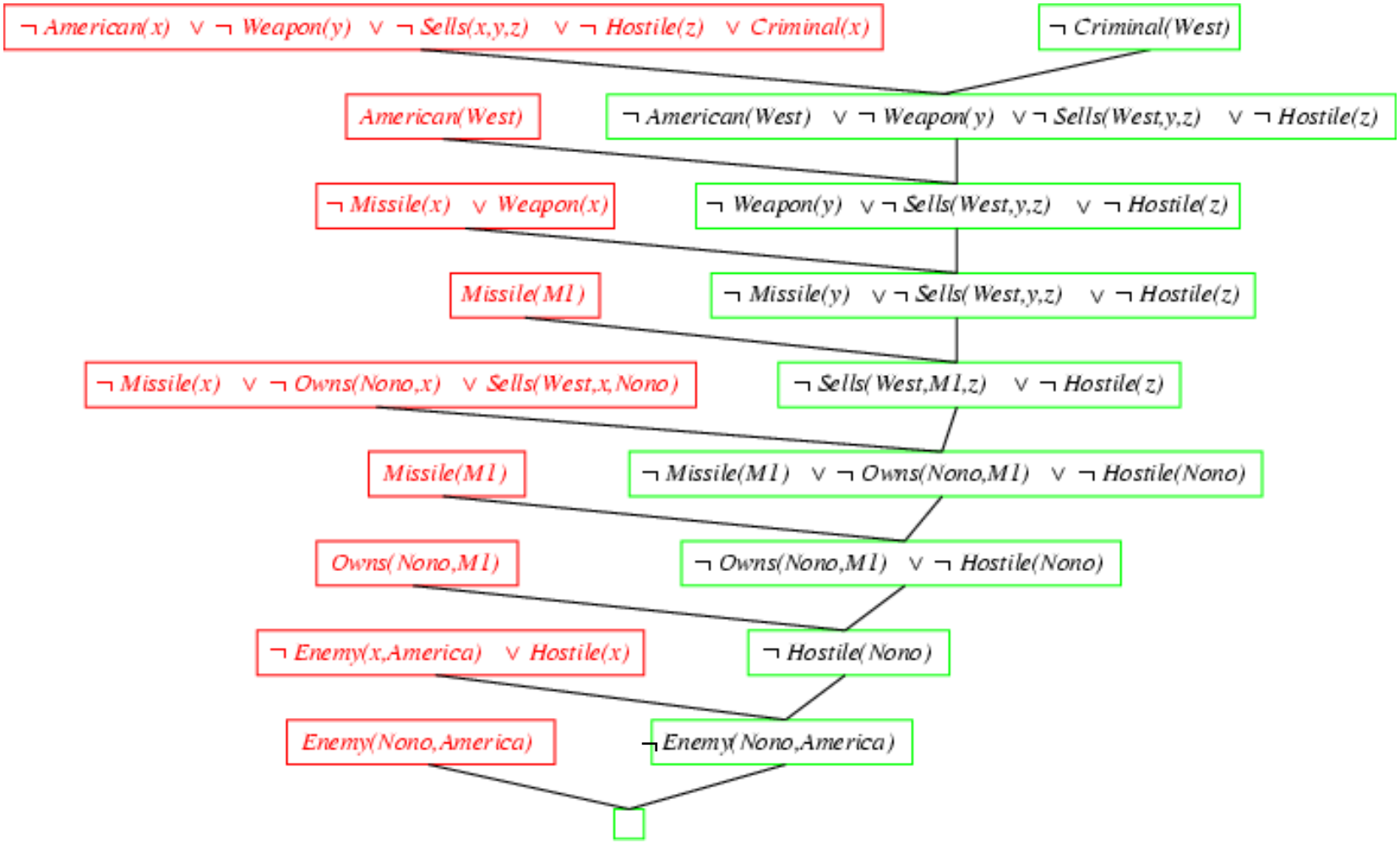
- For example,

$$\frac{\neg \mathit{Rich}(x) \vee \mathit{Unhappy}(x), \quad \mathit{Rich}(\mathit{Ken})}{\mathit{Unhappy}(\mathit{Ken})}$$

with $\theta = \{x/\text{Ken}\}$

- Apply resolution steps to CNF(KB $\wedge \neg \alpha$); complete for FOL

Resolution proof



Converting FOL sentences to CNF

Original sentence:

Everyone who loves all animals is loved by someone:

$\forall x\ [\forall y\ \mathit{Animal}(y) \Rightarrow \mathit{Loves}(x,y)] \Rightarrow [\exists y\ \mathit{Loves}(y,x)]$

1. Eliminate biconditionals and implications

$\forall x\ [\neg\forall y\ \neg\mathit{Animal}(y) \vee \mathit{Loves}(x,y)] \vee [\exists y\ \mathit{Loves}(y,x)]$

2. Move $\neg$ inwards:

Recall: $\neg\forall x\ p \equiv \exists x\ \neg p,\ \ \neg\exists x\ p \equiv \forall x\ \neg p$

$\forall x\ [\exists y\ \neg(\neg\mathit{Animal}(y) \vee \mathit{Loves}(x,y))] \vee [\exists y\ \mathit{Loves}(y,x)]$

$\forall x\ [\exists y\ \neg\neg\mathit{Animal}(y) \wedge \neg\mathit{Loves}(x,y)] \vee [\exists y\ \mathit{Loves}(y,x)]$

$\forall x\ [\exists y\ \mathit{Animal}(y) \wedge \neg\mathit{Loves}(x,y)] \vee [\exists y\ \mathit{Loves}(y,x)]$

Conversion to CNF contd.

3. Standardize variables:
   each quantifier should use a different one

$$\forall x\ [\exists y\ Animal(y) \wedge \neg Loves(x,y)] \vee [\exists z\ Loves(z,x)]$$

4. Skolemize: a more general form of existential instantiation.
   Each existential variable is replaced by a Skolem function of the enclosing universally quantified variables:

$$\forall x\ [Animal(F(x)) \wedge \neg Loves(x,F(x))] \vee Loves(G(x),x)$$

(reason: animal y could be a different animal for each x.)

Conversion to CNF contd.

5. Drop universal quantifiers:

$$[Animal(F(x)) \wedge \neg Loves(x,F(x))] \vee Loves(G(x),x)$$

(all remaining variables assumed to be universally quantified)

6. Distribute $\vee$ over $\wedge$:

$$[Animal(F(x)) \vee Loves(G(x),x)] \wedge [\neg Loves(x,F(x)) \vee Loves(G(x),x)]$$

Original sentence is now in CNF form – can apply same ideas to all sentences in KB to convert into CNF

Also need to include negated query

Then use resolution to attempt to derive the empty clause which show that the query is entailed by the KB

Outline

- Knowledge Representation using First-Order Logic
- ~~Inference in First-Order Logic~~
- Probability, Bayesian Networks
- Machine Learning

- Questions on any topic

- Review pre-mid-term material if time and class interest

Syntax

- Basic element: random variable
- Similar to propositional logic: possible worlds defined by assignment of values to random variables.

- Booleanrandom variables
  e.g., *Cavity (= do I have a cavity?)*
- Discreterandom variables
  e.g., *Weather is one of <sunny,rainy,cloudy,snow>*
- Domain values must be exhaustive and mutually exclusive

- Elementary proposition is an assignment of a value to a random variable:
  e.g., *Weather = sunny; Cavity = false(abbreviated as ¬cavity)*

- Complex propositions formed from elementary propositions and standard logical connectives :
  e.g., *Weather = sunny ∨ Cavity = false*

Probability

- P(a) is the probability of proposition “a”
  - E.g., P(it will rain in London tomorrow)
  - The proposition a is actually true or false in the real-world
  - P(a) = “prior” or marginal or unconditional probability
  - Assumes no other information is available

- Axioms:
  - $0 <= P(a) <= 1$
  - $P(NOT(a)) = 1 - P(a)$
  - $P(true) = 1$
  - $P(false) = 0$
  - $P(A \text{ OR } B) = P(A) + P(B) - P(A \text{ AND } B)$

- An agent that holds degrees of beliefs that contradict these axioms will act sub-optimally in some cases
  - e.g., de Finetti proved that there will be some combination of bets that forces such an unhappy agent to lose money every time.
  - No rational agent can have axioms that violate probability theory.

Conditional Probability

- P(a|b) is the conditional probability of proposition a, conditioned on knowing that b is true,
  - E.g., P(rain in London tomorrow | raining in London today)
  - P(a|b) is a “posterior” or conditional probability
  - The updated probability that a is true, now that we know b
  - P(a|b) = P(a AND b) / P(b)
  - Syntax: P(a | b) is the probability of a given that b is true
    - a and b can be any propositional sentences
    - e.g., p(John wins OR Mary wins | Bob wins AND Jack loses)

- P(a|b) obeys the same rules as probabilities,
  - E.g., P(a | b) + P(NOT(a) | b) = 1
  - All probabilities in effect are conditional probabilities
    - E.g., P(a) = P(a | our background knowledge)

Random Variables

- A is a random variable taking values a_1, a_2, … a_m
  - Events are A= a_1, A= a_2, ….
  - We will focus on discrete random variables

- Mutual exclusion

  $P(A = a_i \text{ AND } A = a_j) = 0$

- Exhaustive

  $\Sigma\ P(a_i) = 1$

  MEE (Mutually Exclusive and Exhaustive) assumption is often useful

  (but not always appropriate, e.g., disease-state for a patient)

  For finite m, can represent P(A) as a table of m probabilities

  For infinite m (e.g., number of tosses before “heads”) we can represent P(A) by a function (e.g., geometric)

Joint Distributions

- Consider 2 random variables: A, B
  - P(a, b) is shorthand for P(A = a AND B=b)
  - $\Sigma_a \Sigma_b$ P(a, b) = 1
  - Can represent P(A, B) as a table of m^2 numbers
- Generalize to more than 2 random variables
  - E.g., A, B, C, … Z
  - $\Sigma_a \Sigma_b \ldots \Sigma_z$ P(a, b, …, z) = 1
  - P(A, B, …. Z) is a table of m^K numbers, K = # variables
    - This is a potential problem in practice, e.g., m=2, K = 20

Linking Joint and Conditional Probabilities

- Basic fact:

  $P(a, b) = P(a \mid b)\, P(b)$

  – Why? Probability of a and b occurring is the same as probability of a occurring given b is true, times the probability of b occurring

- Bayes rule:

  $P(a, b) = P(a \mid b)\, P(b)$

  $= P(b \mid a)\, P(a)$ by definition

  $\Rightarrow P(b \mid a) = P(a \mid b)\, P(b) \,/\, P(a)$ [Bayes rule]

  Why is this useful?

  Often much more natural to express knowledge in a particular "direction", e.g., in the causal direction

  e.g., b = disease, a = symptoms

  More natural to encode knowledge as P(a|b) than as P(b|a)

Sequential Bayesian Reasoning

- h = hypothesis, e1, e2, .. en = evidence
- P(h) = prior
- P(h | e1) proportional to P(e1 | h) P(h)
  = likelihood of e1 x prior(h)
- P(h | e1, e2) proportional to P(e1, e2 | h) P(h)
  in turn can be written as P(e2| h, e1) P(e1|h) P(h)
  ~ likelihood of e2 x "prior"(h given e1)
- Bayes rule supports sequential reasoning
  - Start with prior P(h)
  - New belief (posterior) = P(h | e1)
  - This becomes the "new prior"
  - Can use this to update to P(h | e1, e2), and so on…..

Computing with Probabilities: Law of Total Probability

Law of Total Probability (aka "summing out" or marginalization)

$$P(a) = \Sigma_b P(a, b)$$
$$= \Sigma_b P(a \mid b) P(b)$$ where B is any random variable

Why is this useful?

Given a joint distribution (e.g., P(a,b,c,d)) we can obtain any "marginal" probability (e.g., P(b)) by summing out the other variables, e.g.,

$$P(b) = \Sigma_a \Sigma_c \Sigma_d P(a, b, c, d)$$

We can compute <u>any conditional probability</u> given a joint distribution, e.g.,

$$P(c \mid b) = \Sigma_a \Sigma_d P(a, c, d \mid b)$$
$$= \Sigma_a \Sigma_d P(a, c, d, b) / P(b)$$

where P(b) can be computed as above

Computing with Probabilities: The Chain Rule or Factoring

We can always write

$$P(a, b, c, \ldots z) = P(a \mid b, c, \ldots. z)\, P(b, c, \ldots z)$$

(by definition of joint probability)

Repeatedly applying this idea, we can write

$$P(a, b, c, \ldots z) = P(a \mid b, c, \ldots. z)\, P(b \mid c, .. z)\, P(c \mid .. z)..P(z)$$

This factorization holds for any ordering of the variables

This is the chain rule for probabilities

Independence

- 2 random variables A and B are independent iff

  $$P(a, b) = P(a)\,P(b) \quad \text{for all values } a, b$$

- More intuitive (equivalent) conditional formulation
  - A and B are independent iff

    $$P(a \mid b) = P(a) \quad \text{OR} \quad P(b \mid a)\,P(b), \quad \text{for all values } a, b$$

  - Intuitive interpretation:

    $P(a \mid b) = P(a)$ tells us that knowing b provides no change in our probability for a, i.e., b contains no information about a

- Can generalize to more than 2 random variables

- In practice true independence is very rare
  - "butterfly in China" effect
  - Weather and dental example in the text
  - Conditional independence is much more common and useful

- Note: independence is an assumption we impose on our model of the world - it does not follow from basic axioms

Conditional Independence

- 2 random variables A and B are conditionally independent given C iff

  $P(a, b \mid c) = P(a \mid c)\, P(b \mid c)$ for all values a, b, c

- More intuitive (equivalent) conditional formulation
  - A and B are conditionally independent given C iff

    $P(a \mid b, c) = P(a \mid c)$ OR $P(b \mid a, c)\, P(b \mid c)$, for all values a, b, c

  - Intuitive interpretation:

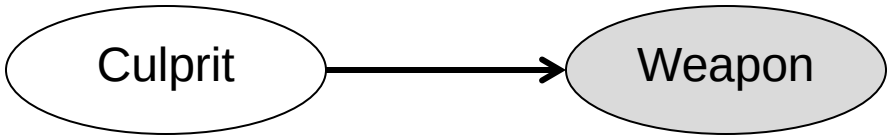
    $P(a \mid b, c) = P(a \mid c)$ tells us that learning about b, given that we already know c, provides no change in our probability for a,

    i.e., b contains no information about a beyond what c provides

- Can generalize to more than 2 random variables
  - E.g., K different symptom variables X1, X2, … XK, and C = disease
  - $P(X1, X2, \ldots. XK \mid C) = \prod P(Xi \mid C)$
  - Also known as the naïve Bayes assumption

Bayesian Networks

Your 1st Bayesian Network



- Each node represents a random variable
- Arrows indicate cause-effect relationship
- Shaded nodes represent observed variables
- Whodunit model in “words”:
  - Culprit chooses a weapon;
  - You observe the weapon and infer the culprit

Bayesian Networks

- Represent dependence/independence via a directed graph
  - Nodes = random variables
  - Edges = direct dependence
- Structure of the graph ⇔ Conditional independence relations
- Recall the chain rule of repeated conditioning:

$$P(X_1, X_2, X_3..., X_N) = P(X_1|X_2, X_3..., X_N)P(X_2|X_3, ..., X_N) \cdots P(X_N)$$

$$P(X_1, X_2, X_3..., X_N) = \prod_{i=1}^{n} P(X_i|parents(X_i))$$

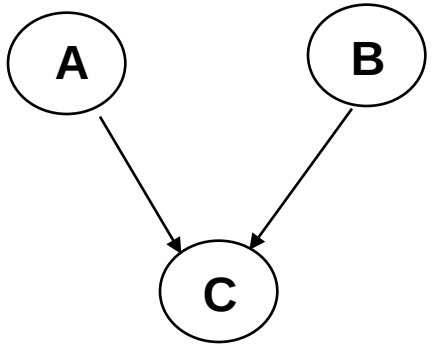
The full joint distribution

The graph-structured approximation

- Requires that graph is acyclic (no directed cycles)
- 2 components to a Bayesian network
  - The graph structure (conditional independence assumptions)
  - The numerical probabilities (for each variable given its parents)

Example of a simple Bayesian network

$$p(A,B,C) = p(C|A,B)p(A|B)p(B)$$
$$= p(C|A,B)p(A)p(B)$$



Probability model has simple factored form

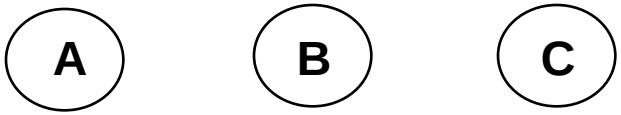
Directed edges => direct dependence

Absence of an edge => conditional independence

Also known as belief networks, graphical models, causal networks

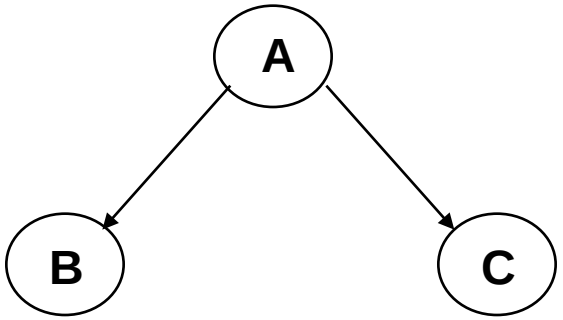
Other formulations, e.g., undirected graphical models

Examples of 3-way Bayesian Networks



Marginal Independence:
p(A,B,C) = p(A) p(B) p(C)

Examples of 3-way Bayesian Networks



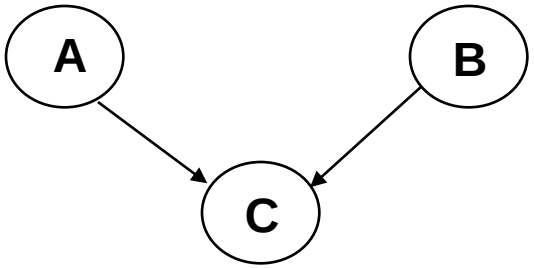
Conditionally independent effects:
p(A,B,C) = p(B|A)p(C|A)p(A)

B and C are conditionally independent Given A

e.g., A is a disease, and we model B and C as conditionally independent symptoms given A

e.g. A is culprit, B is murder weapon and C is fingerprints on door to the guest's room

Examples of 3-way Bayesian Networks

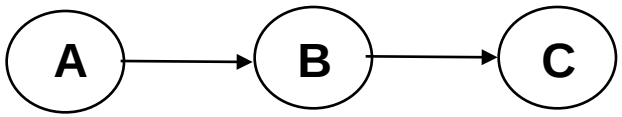


Independent Causes:
$p(A,B,C) = p(C|A,B)p(A)p(B)$

"Explaining away" effect:
Given C, observing A makes B less likely
e.g., earthquake/burglary/alarm example

A and B are (marginally) independent
but become dependent once C is known

Examples of 3-way Bayesian Networks



Markov chain dependence:
$p(A,B,C) = p(C|B)\, p(B|A)p(A)$

e.g. If Prof. Lathrop goes to party, then I might go to party. If I go to party, then my wife might go to party.

Bigger Example

- Consider the following 5 binary variables:
  - B = a burglary occurs at your house
  - E = an earthquake occurs at your house
  - A = the alarm goes off
  - J = John calls to report the alarm
  - M = Mary calls to report the alarm
- Sample Query: What is P(B|M, J) ?
- Using full joint distribution to answer this question requires
  - 2^5 - 1= 31 parameters
- Can we use prior domain knowledge to come up with a Bayesian network that requires fewer probabilities?

Constructing a Bayesian Network

- Order variables in terms of causality (may be a partial order)

  e.g., {E, B} -> {A} -> {J, M}

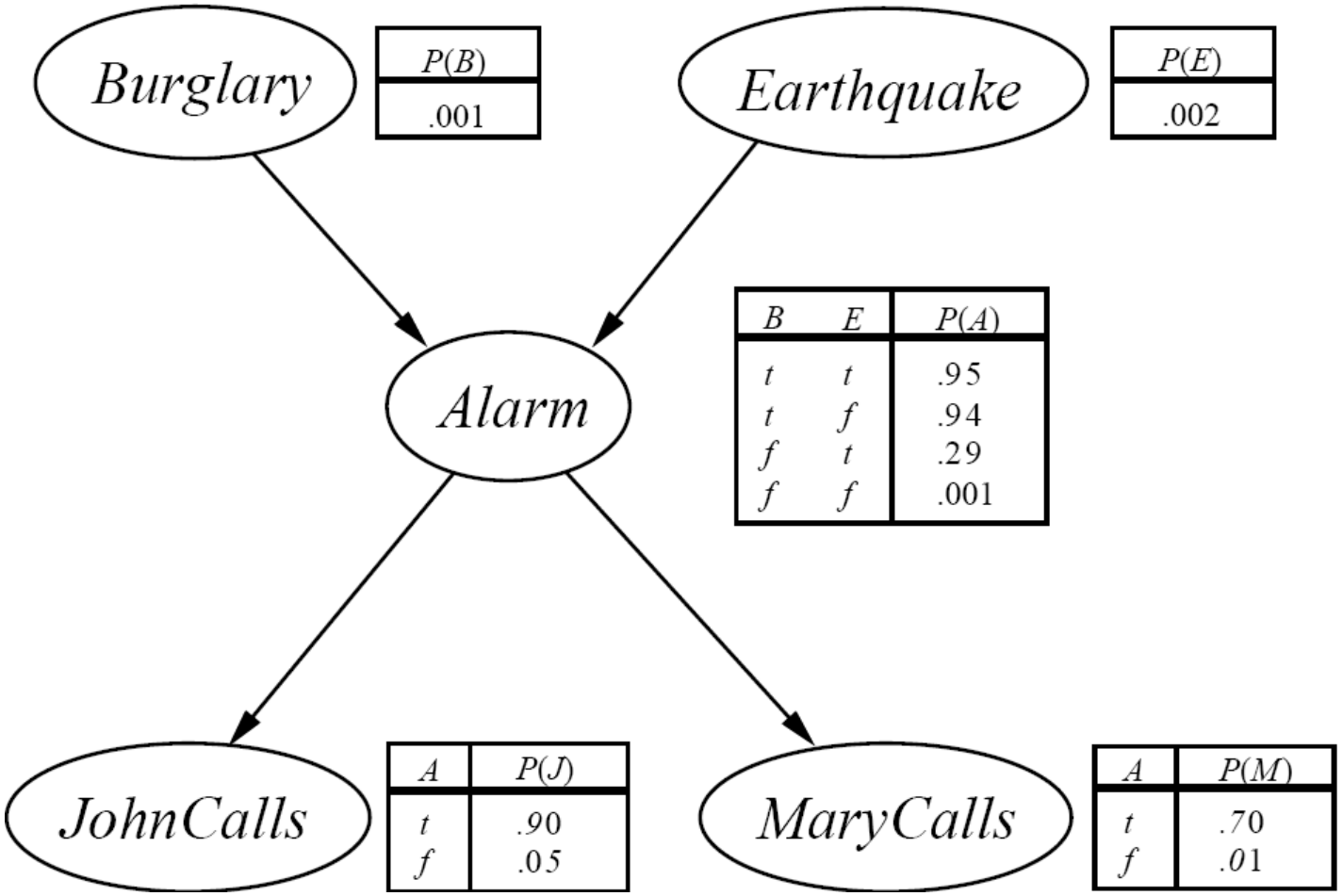
- $P(J, M, A, E, B) = P(J, M \mid A, E, B)\ P(A \mid E, B)\ P(E, B)$

  $\approx P(J, M \mid A)\ P(A \mid E, B)\ P(E)\ P(B)$

  $\approx P(J \mid A)\ P(M \mid A)\ P(A \mid E, B)\ P(E)\ P(B)$

- These conditional independence assumptions are reflected in the graph structure of the Bayesian network

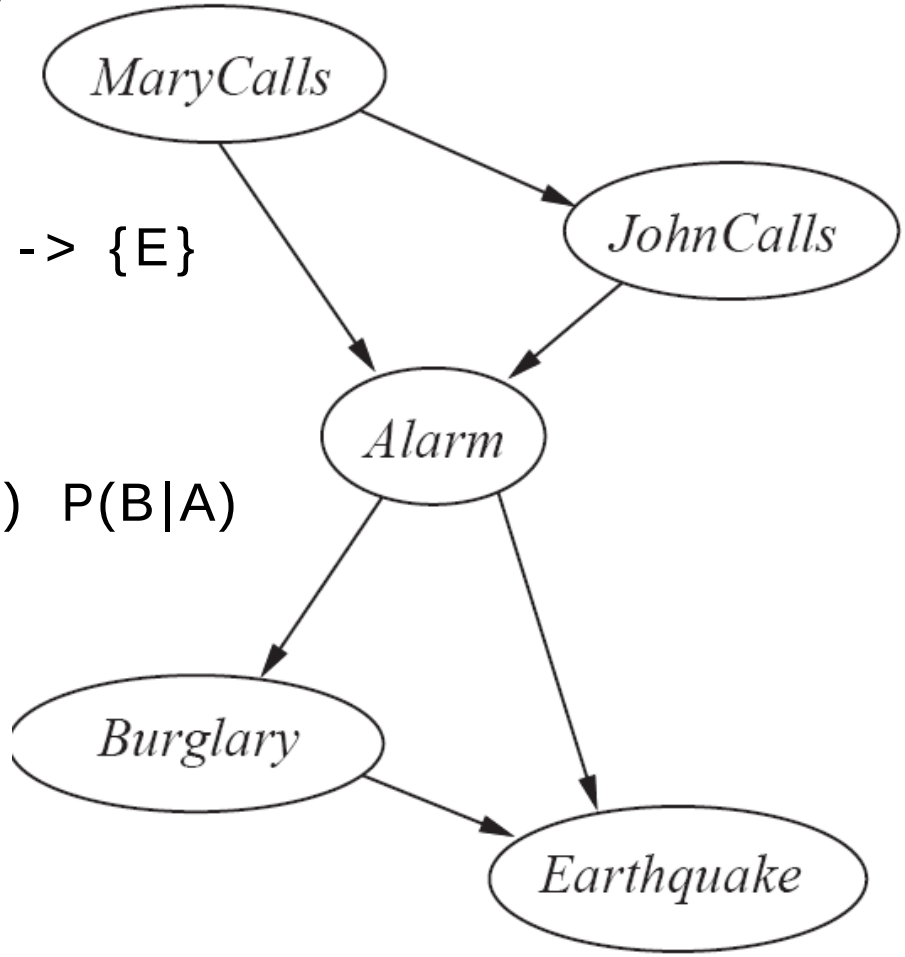
The Resulting Bayesian Network



The Bayesian Network from a different Variable Ordering

{M} -> {J} -> {A} -> {B} -> {E}

P(J, M, A, E, B) =
P(M) P(J|M) P(A|M,J) P(B|A)
P(E|A,B)



(a)

Inference by Variable Elimination

- Say that query is P(*B*|*j*,*m*)
  - P(*B*|*j*,*m*) = P(B,*j*,*m*) / P(*j*,*m*) = α P(B,*j*,*m*)
- Apply evidence to expression for joint distribution
  - P(j,m,A,E,B) = P(j|A)P(m|A)P(A|E,B)P(E)P(B)
- Marginalize out A and E

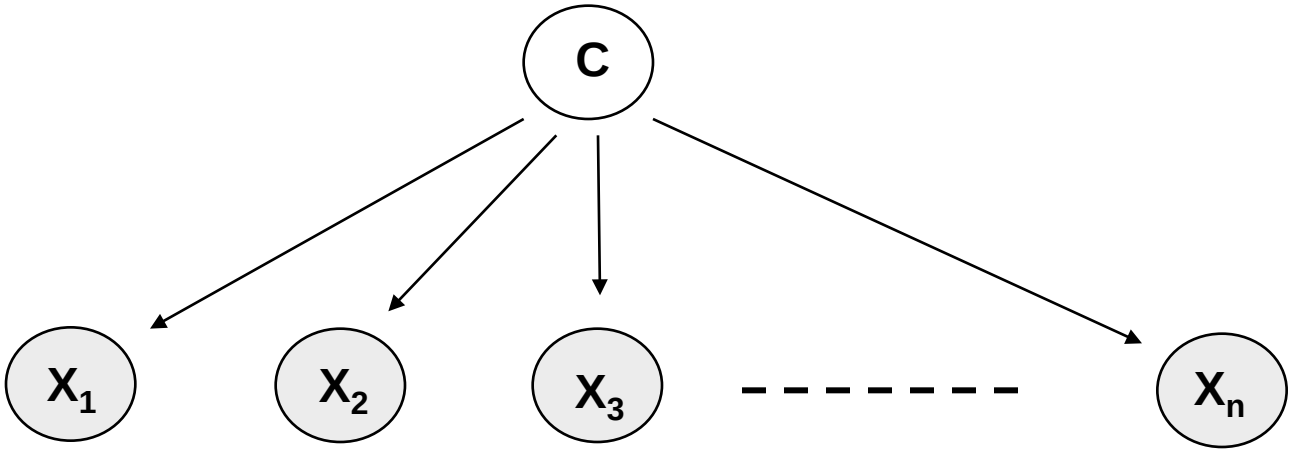
$$P(B|j,m) = \alpha \sum_a \sum_e p(j|a)p(m|a)p(a|e,B)P(e)P(B)$$

$$= \alpha P(B) \sum_e P(e) \sum_a p(j|a)p(m|a)p(a|e,B)$$

Distribution over variable B – i.e. over states {b,¬b}

Sum is over states of variable A – i.e. {a,¬a}

Naïve Bayes Model



$$P(C \mid X_1,\ldots X_n) = \alpha \prod P(X_i \mid C)\ P(C)$$

Features X are conditionally independent given the class variable C

Widely used in machine learning

e.g., spam email classification: X's = counts of words in emails

Probabilities P(C) and P(Xi | C) can easily be estimated from labeled data

Outline

- Knowledge Representation using First-Order Logic
- ~~Inference in First-Order Logic~~
- Probability, Bayesian Networks
- Machine Learning

- Questions on any topic

- Review pre-mid-term material if time and class interest

The importance of a good representation

- **Properties of a good representation:**

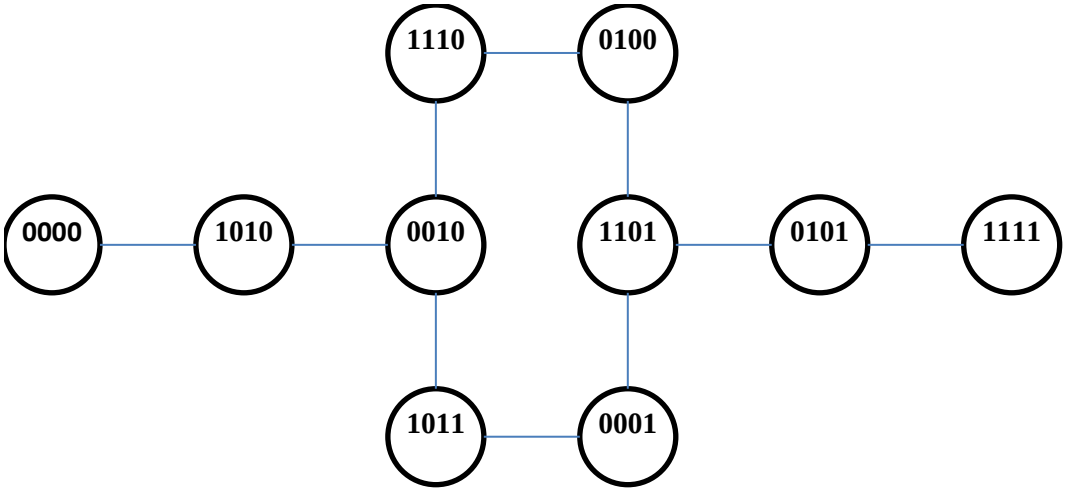
- Reveals important features
- Hides irrelevant detail
- Exposes useful constraints
- Makes frequent operations easy-to-do
- Supports local inferences from local features
  - Called the “soda straw” principle or “locality” principle
  - Inference from features “through a soda straw”
- Rapidly or efficiently computable
  - It’s nice to be fast

Reveals important features / Hides irrelevant detail

- **"You can't learn what you can't represent."** --- G. Sussman

- **In search:** *A man is traveling to market with a fox, a goose, and a bag of oats. He comes to a river. The only way across the river is a boat that can hold the man and exactly one of the fox, goose or bag of oats. The fox will eat the goose if left alone with it, and the goose will eat the oats if left alone with it.*

- **A good representation makes this problem easy:**



Exposes useful constraints

- **“You can’t learn what you can’t represent.”** --- G. Sussman

- **In logic:** *If the unicorn is mythical, then it is immortal, but if it is not mythical, then it is a mortal mammal. If the unicorn is either immortal or a mammal, then it is horned. The unicorn is magical if it is horned.*

- **A good representation makes this problem easy:**

$$(\neg Y \vee \neg R) \wedge (Y \vee R) \wedge (Y \vee M) \wedge (R \vee H) \wedge (\neg M \vee H) \wedge (\neg H \vee G)$$

Makes frequent operations easy-to-do

- **Roman numerals**
  - M=1000, D=500, C=100, L=50, X=10, V=5, I=1
  - 2011 = MXI; 1776 = MDCCLXXVI

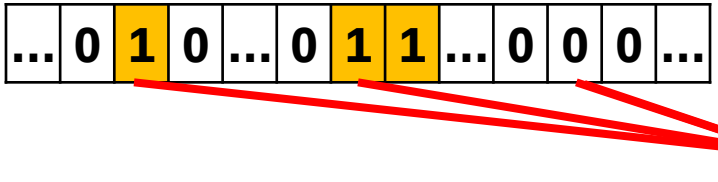
- Long division is **very tedious** (try MDCCLXXVI / XVI)
- Testing for N < 1000 is very easy (first letter is not “M”)

- **Arabic numerals**
  - 0, 1, 2, 3, 4, 5, 6, 7, 8, 9, “.”

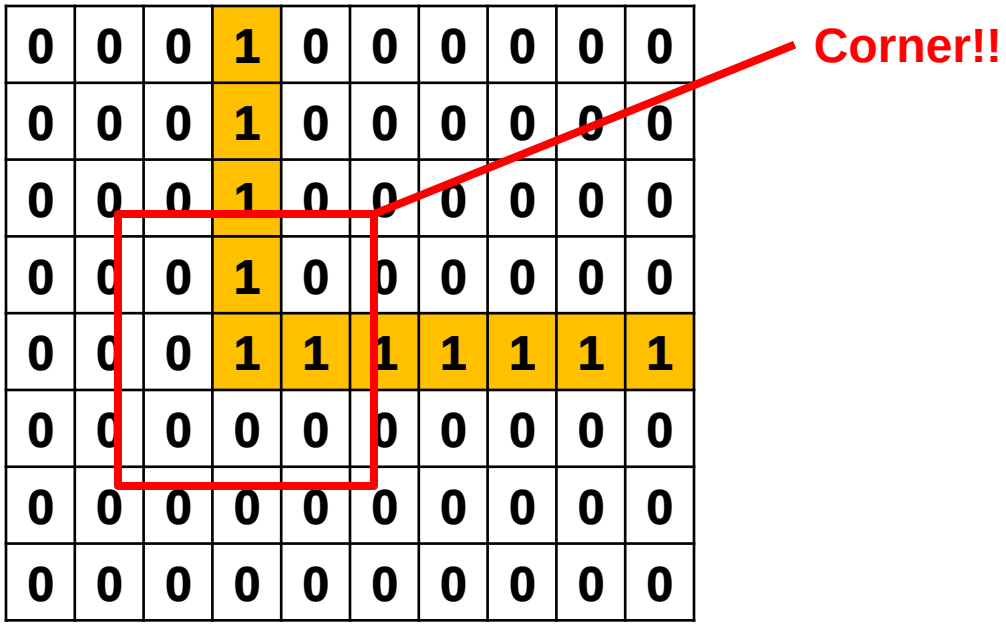
- Long division is **much easier** (try 1776 / 16)
- Testing for N < 1000 is slightly harder (have to scan the string)

Supports local inferences from local features

- Linear vector of pixels = highly non-local inference for vision



- Rectangular array of pixels = local inference for vision



Terminology

- Attributes
  - Also known as features, variables, independent variables, covariates
- Target Variable
  - Also known as goal predicate, dependent variable, …
- Classification
  - Also known as discrimination, supervised classification, …
- Error function
  - Objective function, loss function, …

Inductive learning

- Let x represent the input vector of attributes
- Let f(x) represent the value of the target variable for x
  - The implicit mapping from x to f(x) is unknown to us
  - We just have training data pairs, D = {x, f(x)} available
- We want to learn a mapping from x to f, i.e.,

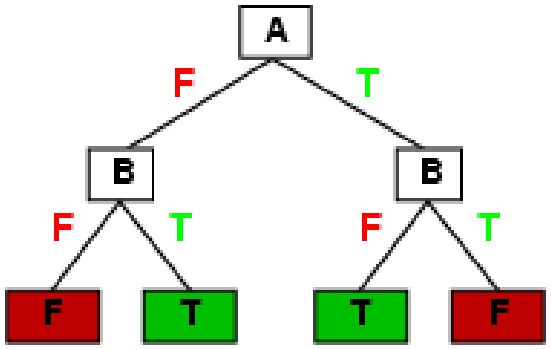
  h(x; θ) is "close" to f(x) for all training data points x

  θ are the parameters of our predictor h(..)
- Examples:
  - h(x; θ) = sign($w_1x_1 + w_2x_2 + w_3$)
  - h_k(x) = (x1 OR x2) AND (x3 OR NOT(x4))

Decision Tree Representations

- Decision trees are fully expressive
  - can represent any Boolean function
  - Every path in the tree could represent 1 row in the truth table
  - Yields an exponentially large tree
    - Truth table is of size 2^d, where d is the number of attributes

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



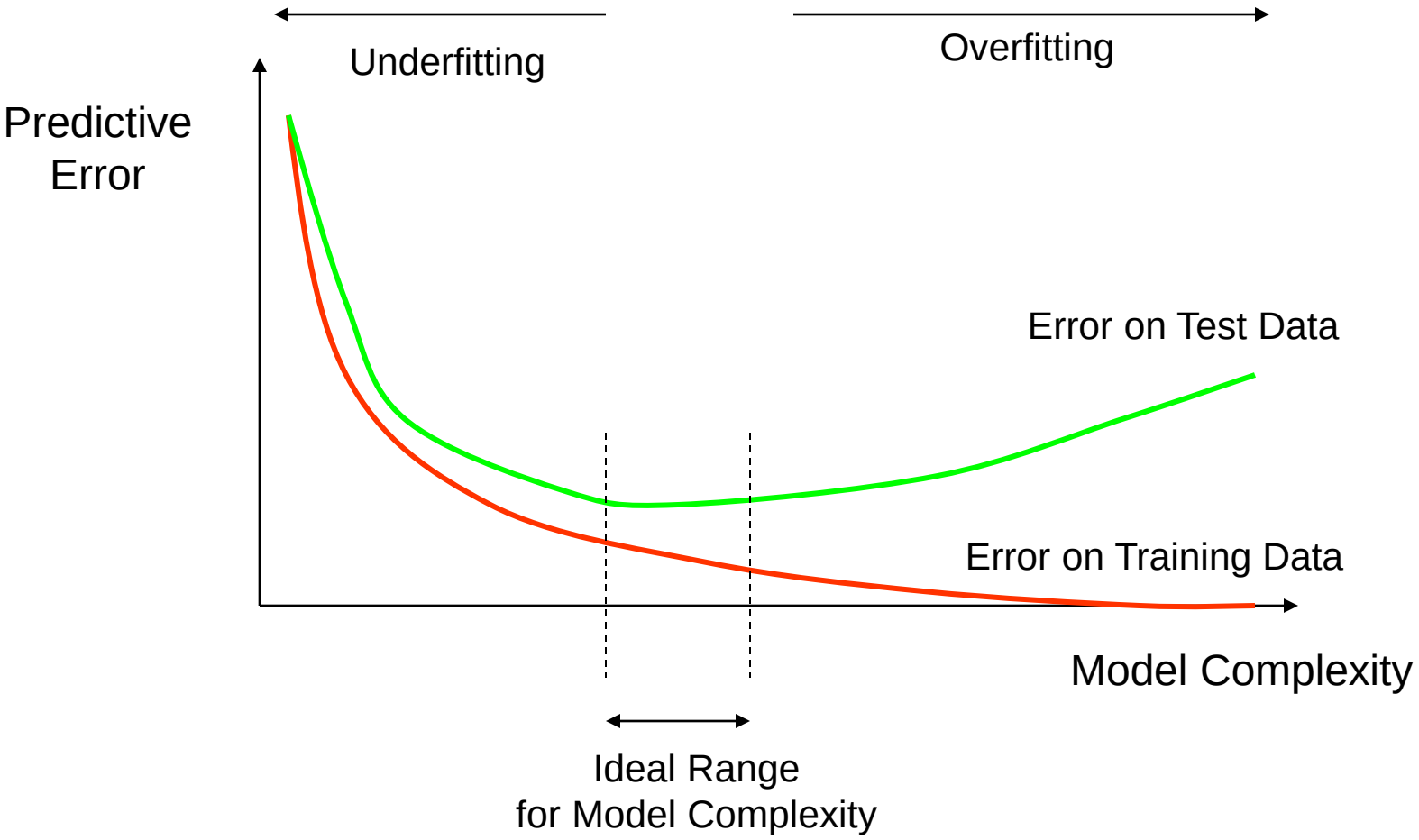
Pseudocode for Decision tree learning

```
function DTL(examples, attributes, default) returns a decision tree
    if examples is empty then return default
    else if all examples have the same classification then return the classification
    else if attributes is empty then return MODE(examples)
    else
        best ← CHOOSE-ATTRIBUTE(attributes, examples)
        tree ← a new decision tree with root test best
        for each value v_i of best do
            examples_i ← {elements of examples with best = v_i}
            subtree ← DTL(examples_i, attributes − best, MODE(examples))
            add a branch to tree with label v_i and subtree subtree
        return tree
```

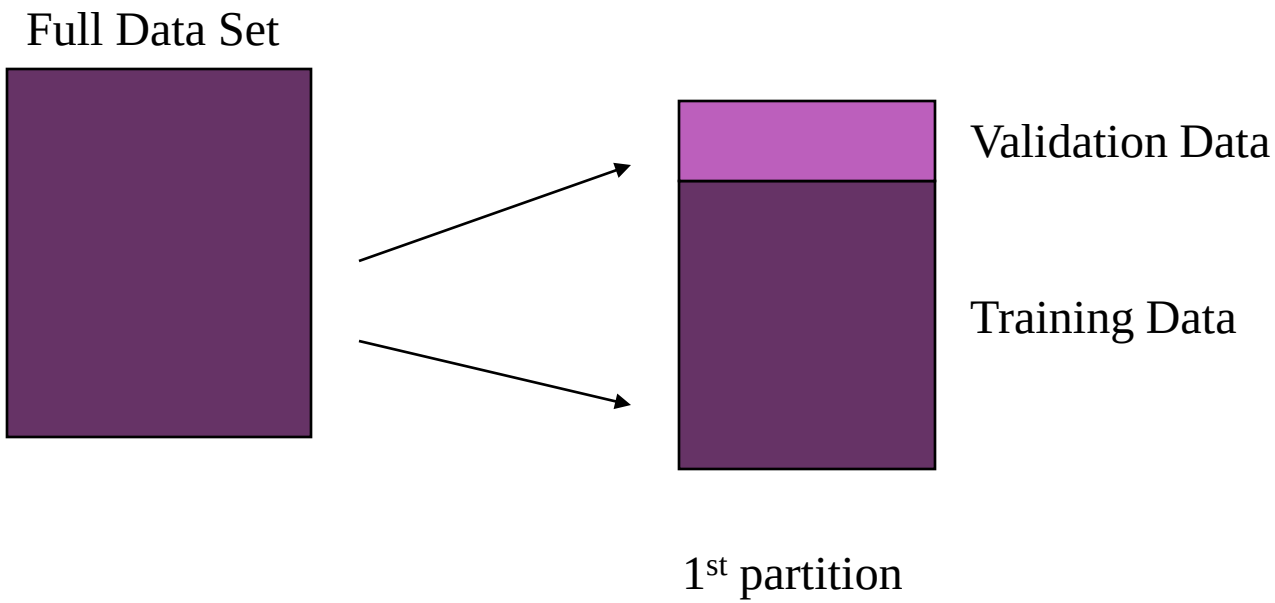
Information Gain

- H(p) = entropy of class distribution at a particular node
- H(p | A) = conditional entropy = average entropy of conditional class distribution, after we have partitioned the data according to the values in A
- Gain(A) = H(p) – H(p | A)
- Simple rule in decision tree learning
  - At each internal node, split on the node with the largest information gain (or equivalently, with smallest H(p|A))
- Note that by definition, conditional entropy can't be greater than the entropy

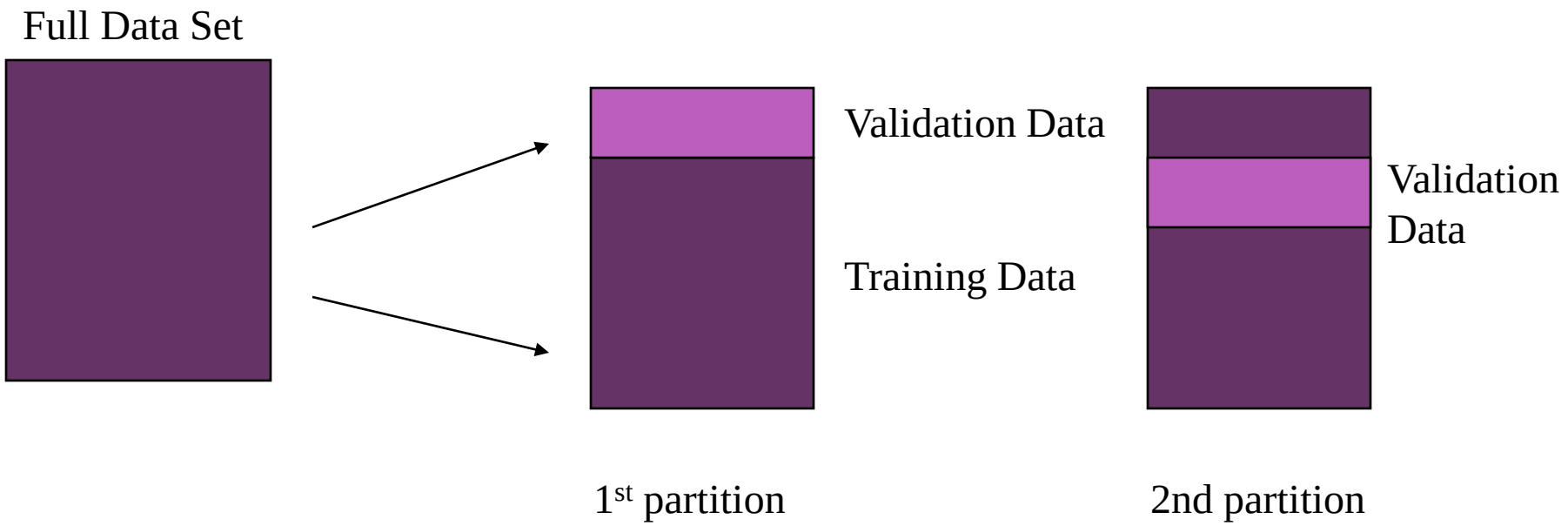
How Overfitting affects Prediction



Disjoint Validation Data Sets



Disjoint Validation Data Sets



Classification in Euclidean Space

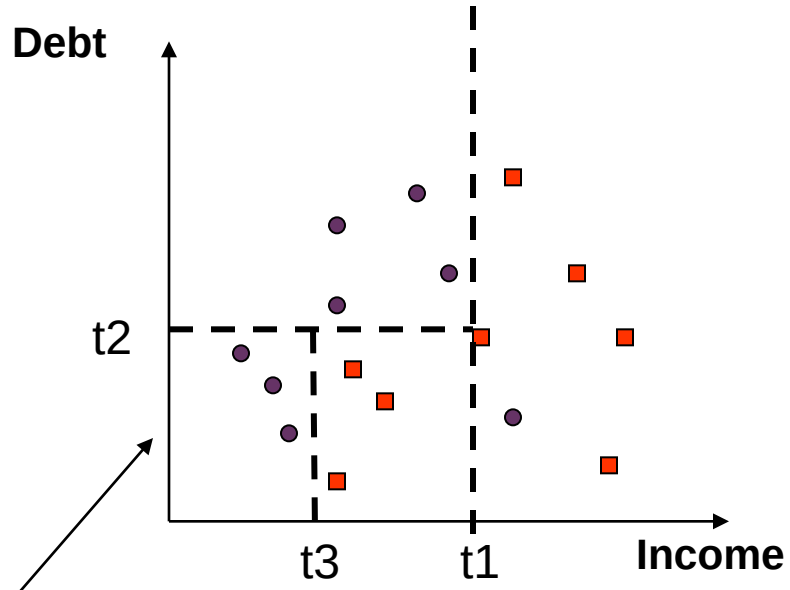
- A classifier is a partition of the space $\underline{x}$ into disjoint decision regions
  - Each region has a label attached
  - Regions with the same label need not be contiguous
  - For a new test point, find what decision region it is in, and predict the corresponding label

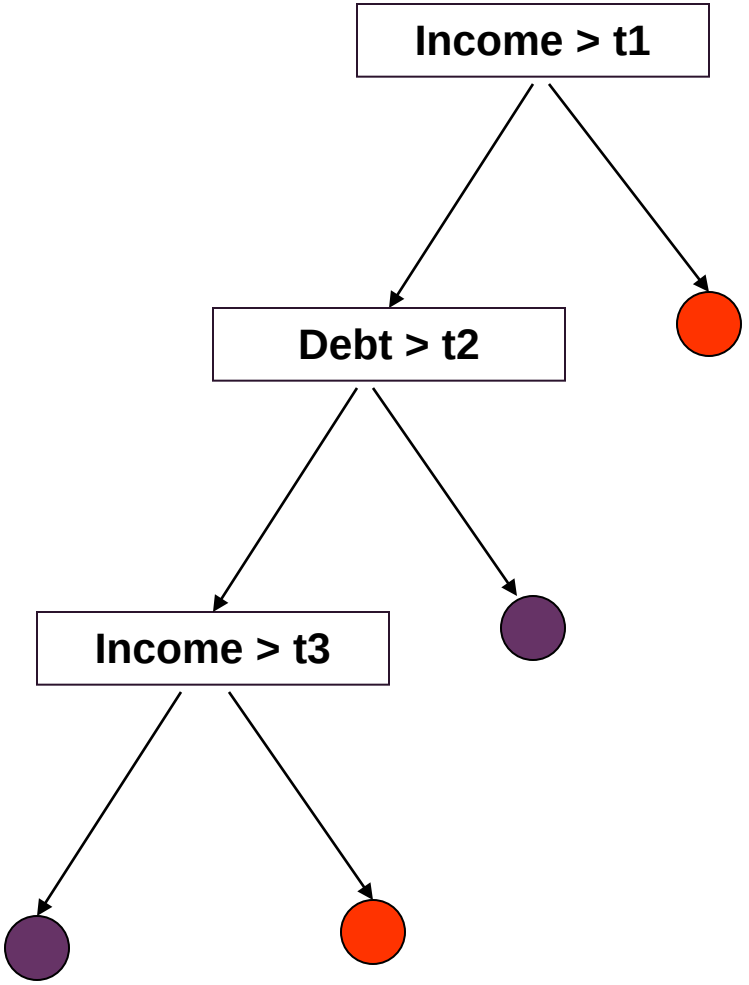
- Decision boundaries = boundaries between decision regions
  - The “dual representation” of decision regions

- We can characterize a classifier by the equations for its decision boundaries

- Learning a classifier ⇔ searching for the decision boundaries that optimize our objective function

Decision Tree Example





Note: tree boundaries are linear and axis-parallel

Another Example: Nearest Neighbor Classifier

- The nearest-neighbor classifier
  - Given a test point $\underline{x}$', compute the distance between $\underline{x}$' and each input data point
  - Find the closest neighbor in the training data
  - Assign $\underline{x}$' the class label of this neighbor
  - (sort of generalizes minimum distance classifier to exemplars)

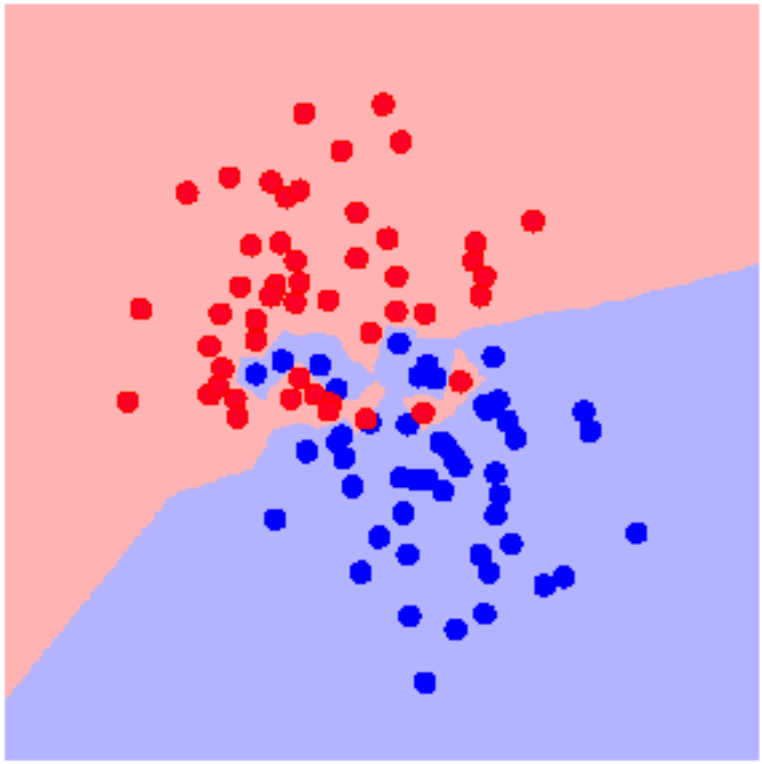
- If Euclidean distance is used as the distance measure (the most common choice), the nearest neighbor classifier results in piecewise linear decision boundaries

- Many extensions
  - e.g., kNN, vote based on k-nearest neighbors
  - k can be chosen by cross-validation

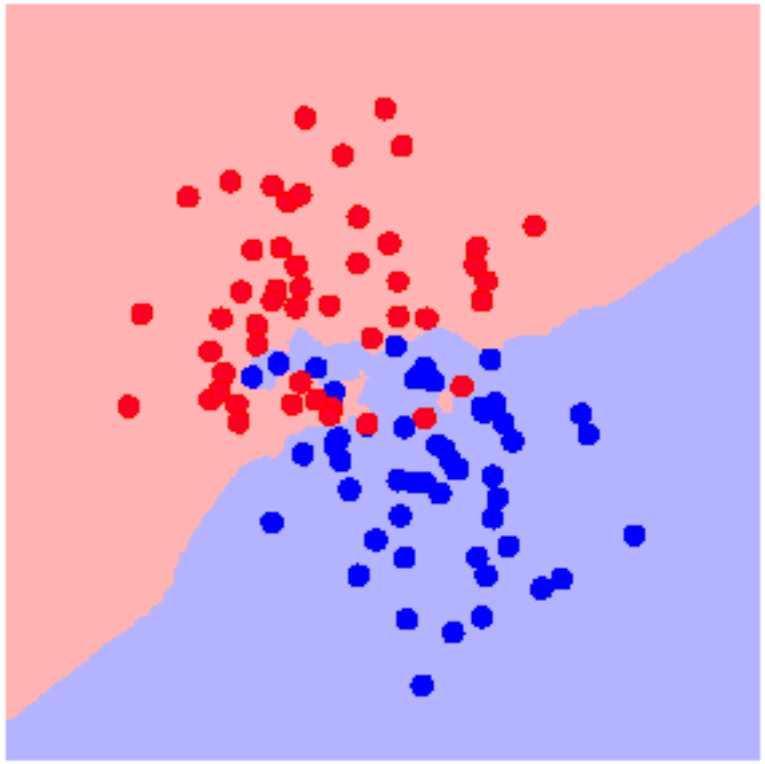
kNN Decision Boundary

- piecewise linear decision boundary
- Increasing k "simplifies" decision boundary
  - Majority voting means less emphasis on individual points

K = 1



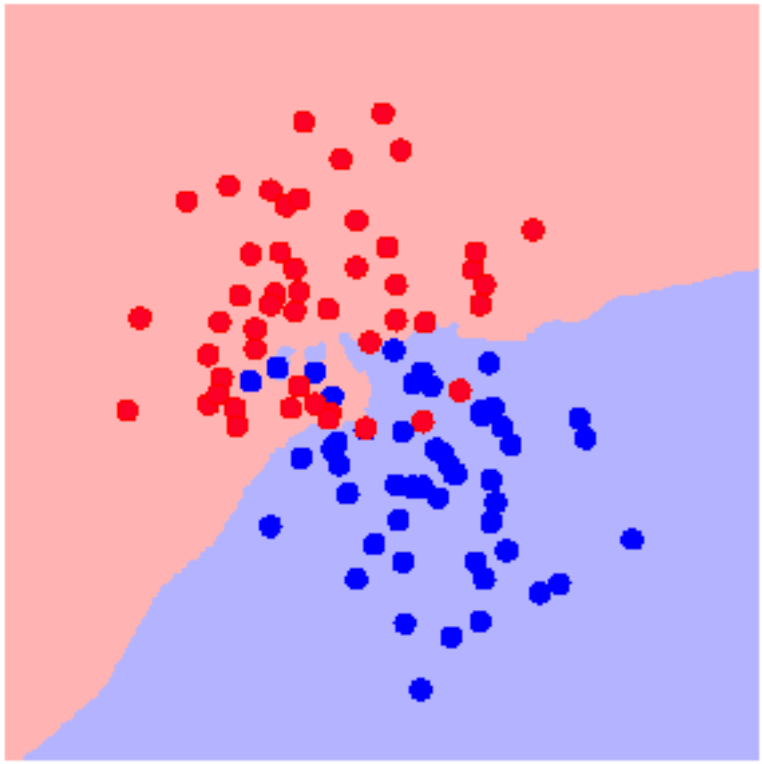
K = 3



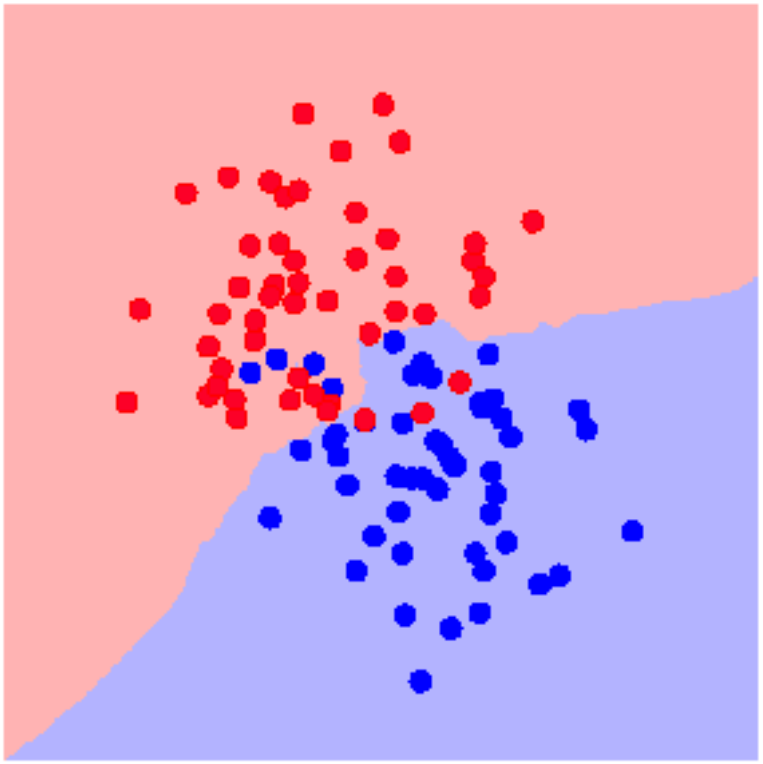
kNN Decision Boundary

- piecewise linear decision boundary
- Increasing k "simplifies" decision boundary
  - Majority voting means less emphasis on individual points

K = 5



K = 7

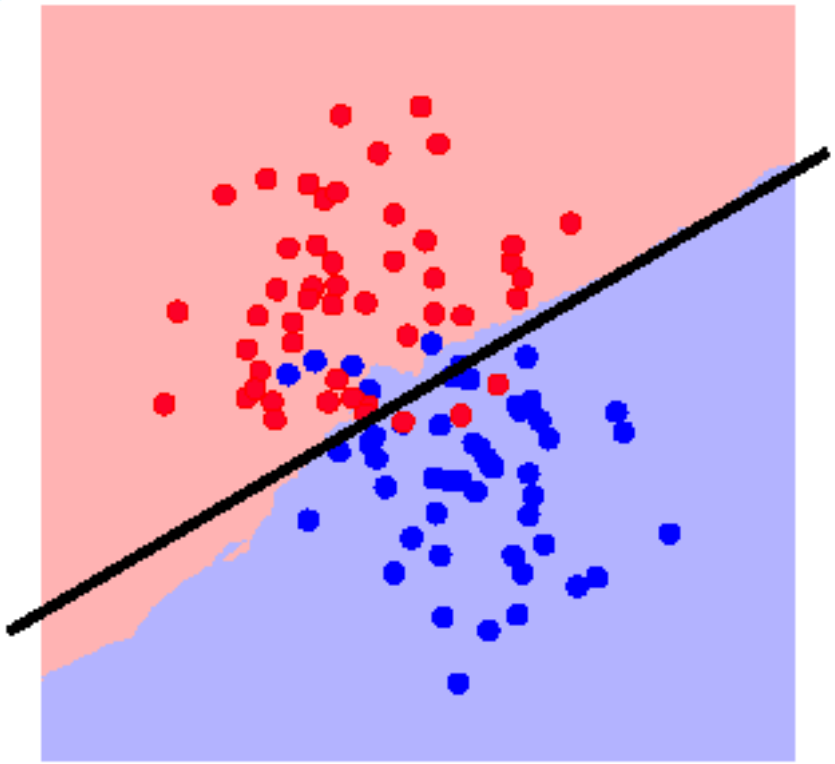


kNN Decision Boundary

- piecewise linear decision boundary
- Increasing k "simplifies" decision boundary
  - Majority voting means less emphasis on individual points

- True ("best") decision boundary
  - In this case is linear
  - Compared to kNN: not bad!

K = 25

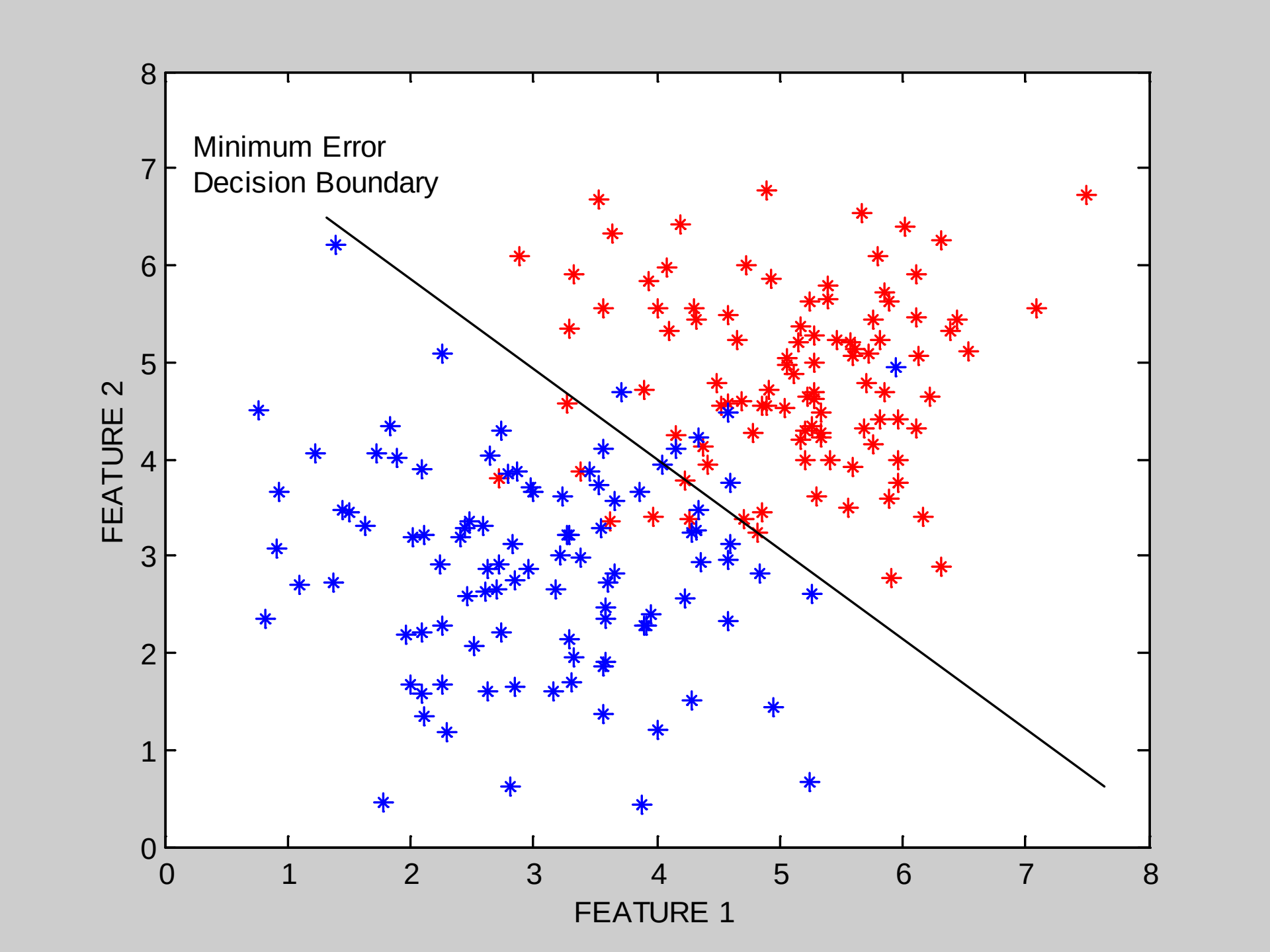


Linear Classifiers

- Linear classifier ⇔ single linear decision boundary (for 2-class case)
- We can always represent a linear decision boundary by a linear equation:

$$w_1 x_1 + w_2 x_2 + \ldots + w_d x_d = \sum w_j x_j = \underline{w}^t \underline{x} = 0$$

- In d dimensions, this defines a (d-1) dimensional hyperplane
  - d=3, we get a plane; d=2, we get a line
- For prediction we simply see if $\sum w_j x_j > 0$
- The w_i are the weights (parameters)
  - Learning consists of searching in the d-dimensional weight space for the set of weights (the linear boundary) that minimizes an error measure
  - A threshold can be introduced by a "dummy" feature that is always one; it weight corresponds to (the negative of) the threshold
- Note that a minimum distance classifier is a special (restricted) case of a linear classifier


Minimum Error
Decision Boundary
FEATURE 2
FEATURE 1
0
1
2
3
4
5
6
7
8

The Perceptron Classifier (pages 740-743 in text)

- The perceptron classifier is just another name for a linear classifier for 2-class data, i.e.,

  $$\text{output}(\underline{x}) = \text{sign}\left(\sum w_j x_j \right)$$

- Loosely motivated by a simple model of how neurons fire

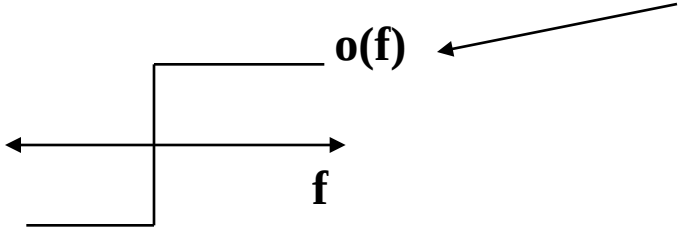
- For mathematical convenience, class labels are +1 for one class and -1 for the other

- Two major types of algorithms for training perceptrons
  - Objective function = classification accuracy ("error correcting")
  - Objective function = squared error (use gradient descent)

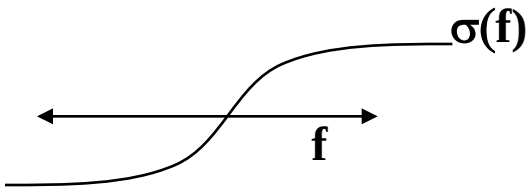
  - Gradient descent is generally faster and more efficient – but there is a problem! No gradient!

Two different types of perceptron output

x-axis below is f(x) = f = weighted sum of inputs
y-axis is the perceptron output



Thresholded output,
takes values +1 or -1



Sigmoid output, takes
real values between -1 and +1

The sigmoid is in effect an approximation
to the threshold function above, but
has a gradient that we can use for learning

Gradient Descent Update Equation

- From basic calculus, for perceptron with sigmoid, and squared error objective function, gradient for a single input $\underline{x}(i)$ is

$$\Delta (E[\underline{w}]) = - (y(i) - \sigma[f(i)]) \partial\sigma[f(i)] x_j(i)$$

- Gradient descent weight update rule:

$$w_j = w_j + \eta (y(i) - \sigma[f(i)]) \partial\sigma[f(i)] x_j(i)$$

  - can rewrite as:

$$w_j = w_j + \eta * error * c * x_j(i)$$

Pseudo-code for Perceptron Training

Initialize each w_j (e.g.,randomly)

While (termination condition not satisfied)
    for i = 1: N % loop over data points (an iteration)
        for j= 1 : d % loop over weights
            deltawj = η (y(i) – σ[f(i)]) $\partial\sigma$[f(i)] x_j(i)
            w_j = w_j + deltawj
        end
    calculate termination condition
    end

- Inputs: N features, N targets (class labels), learning rate η
- Outputs: a set of learned weights

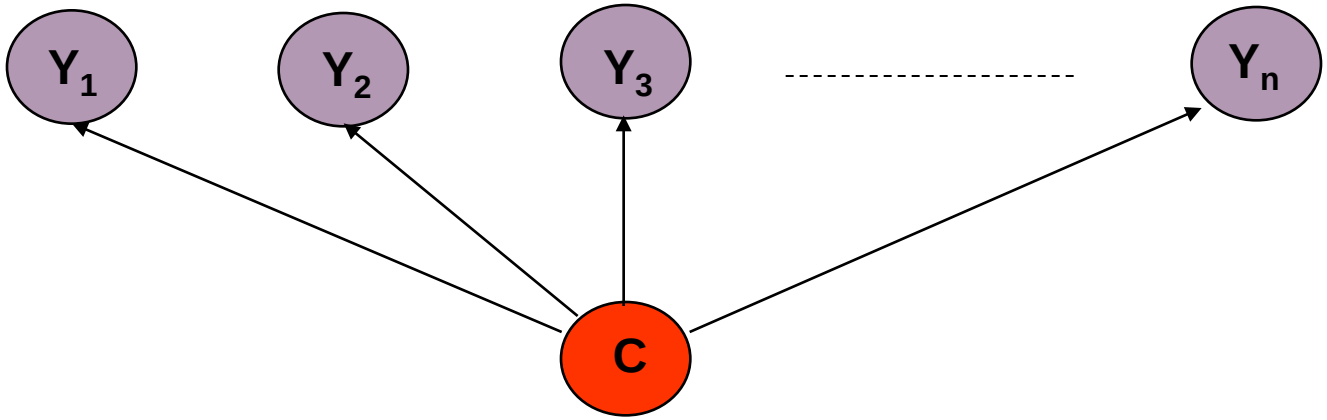
Multi-Layer Perceptrons (p744-747 in text)

- What if we took K perceptrons and trained them in parallel and then took a weighted sum of their sigmoidal outputs?
  - This is a multi-layer neural network with a single “hidden” layer (the outputs of the first set of perceptrons)
  - If we train them jointly in parallel, then intuitively different perceptrons could learn different parts of the solution
    - Mathematically, they define different local decision boundaries in the input space, giving us a more powerful model

- How would we train such a model?
  - Backpropagation algorithm = clever way to do gradient descent
  - Bad news: many local minima and many parameters
    - training is hard and slow
  - Neural networks generated much excitement in AI research in the late 1980’s and 1990’s
    - But now techniques like boosting and support vector machines are often preferred

Naïve Bayes Model

(p. 808 R&N 3rd ed., 718 2nd ed.)



$$P(C \mid Y_1,\ldots Y_n) = \alpha \prod P(Y_i \mid C)\ P(C)$$

Features Y are conditionally independent given the class variable C

Widely used in machine learning

e.g., spam email classification: Y's = counts of words in emails

Conditional probabilities P(Yi | C) can easily be estimated from labeled data

Problem: Need to avoid zeroes, e.g., from limited training data

Solutions: Pseudo-counts, beta[a,b] distribution, etc.

Naïve Bayes Model (2)

$$P(C \mid X_1,\ldots X_n) = \alpha \prod P(X_i \mid C)\ P(C)$$

Probabilities P(C) and P(Xi | C) can easily be estimated from labeled data

P(C = cj) ≈ #(Examples with class label cj) / #(Examples)

P(Xi = xik | C = cj)
≈ #(Examples with Xi value xik and class label cj)
/ #(Examples with class label cj)

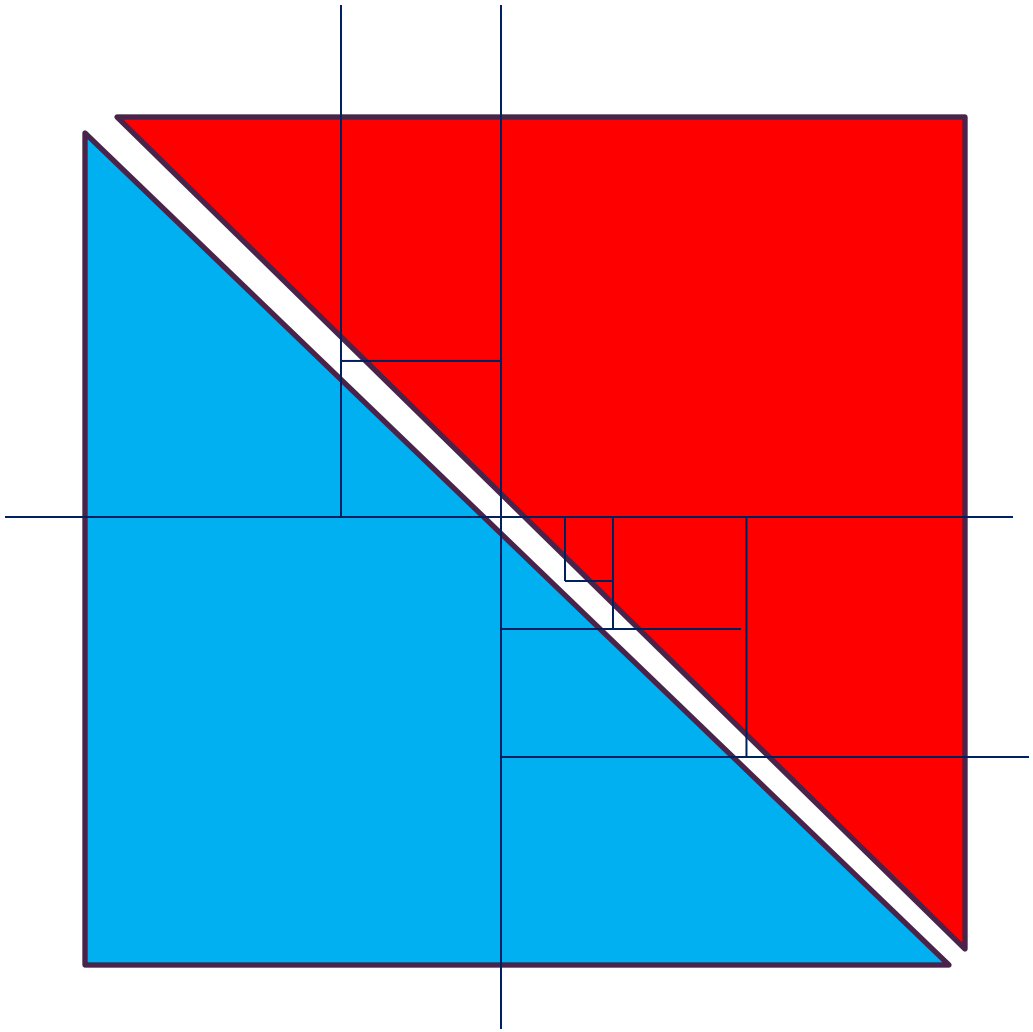
Usually easiest to work with logs

$$\log [\ P(C \mid X_1,\ldots X_n)\ ] = \log \alpha + \sum [\ \log P(X_i \mid C) + \log P(C)\ ]$$

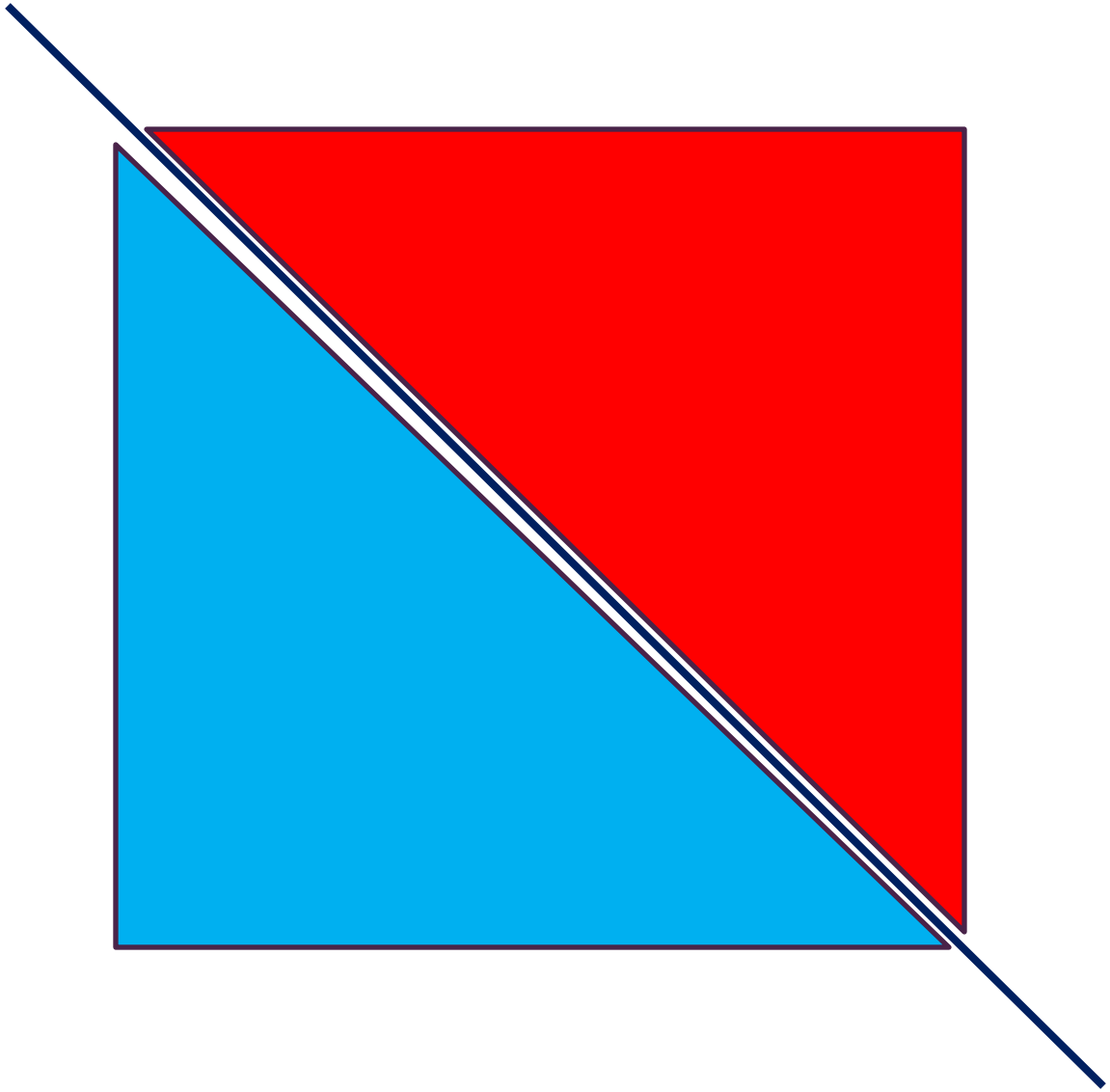
DANGER: Suppose ZERO examples with Xi value xik and class label cj ?
An unseen example with Xi value xik will NEVER predict class label cj !

Practical solutions: Pseudocounts, e.g., add 1 to every #() , etc.
Theoretical solutions: Bayesian inference, beta distribution, etc.

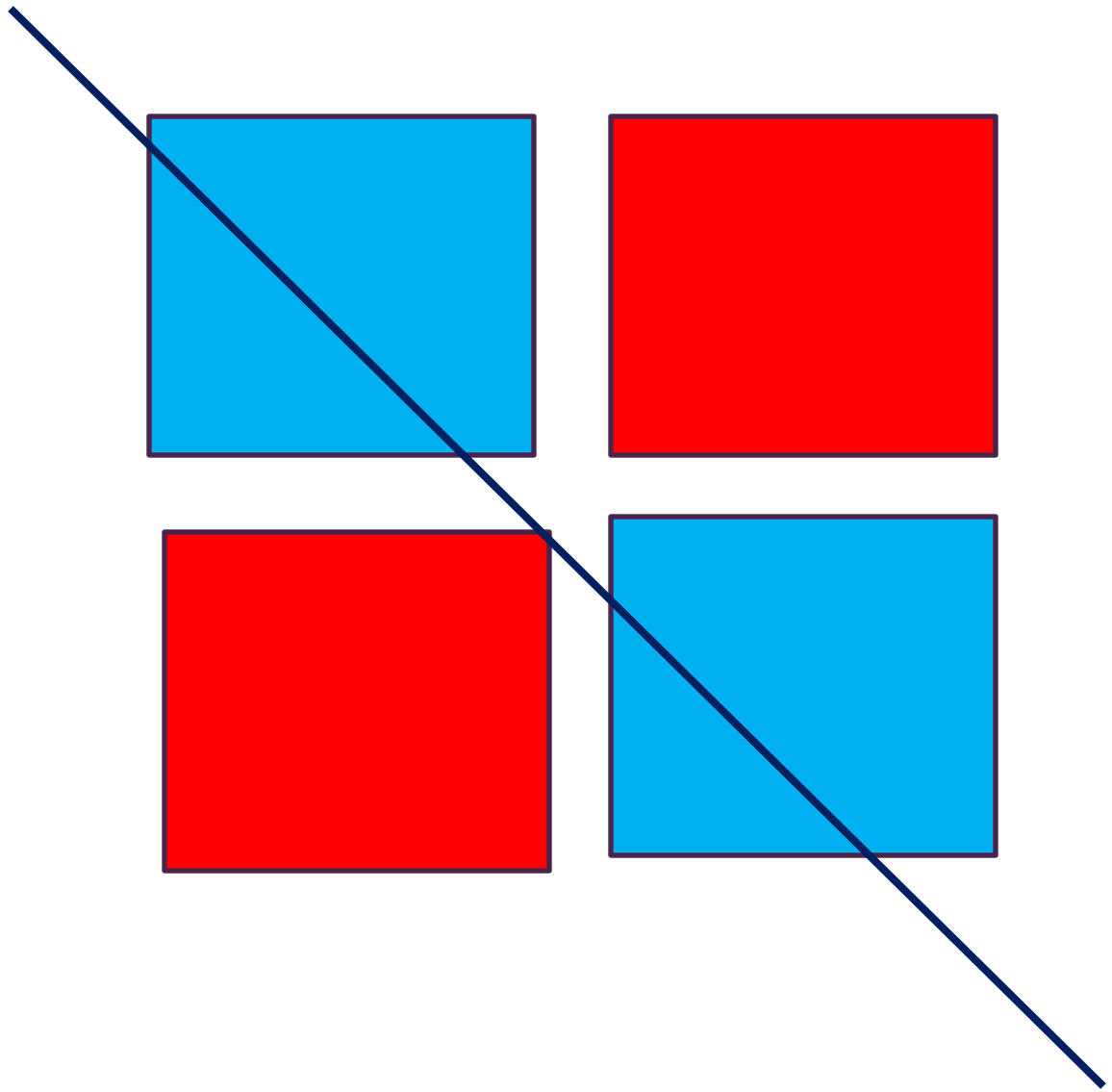
Classifier Bias — Decision Tree or Linear Perceptron?



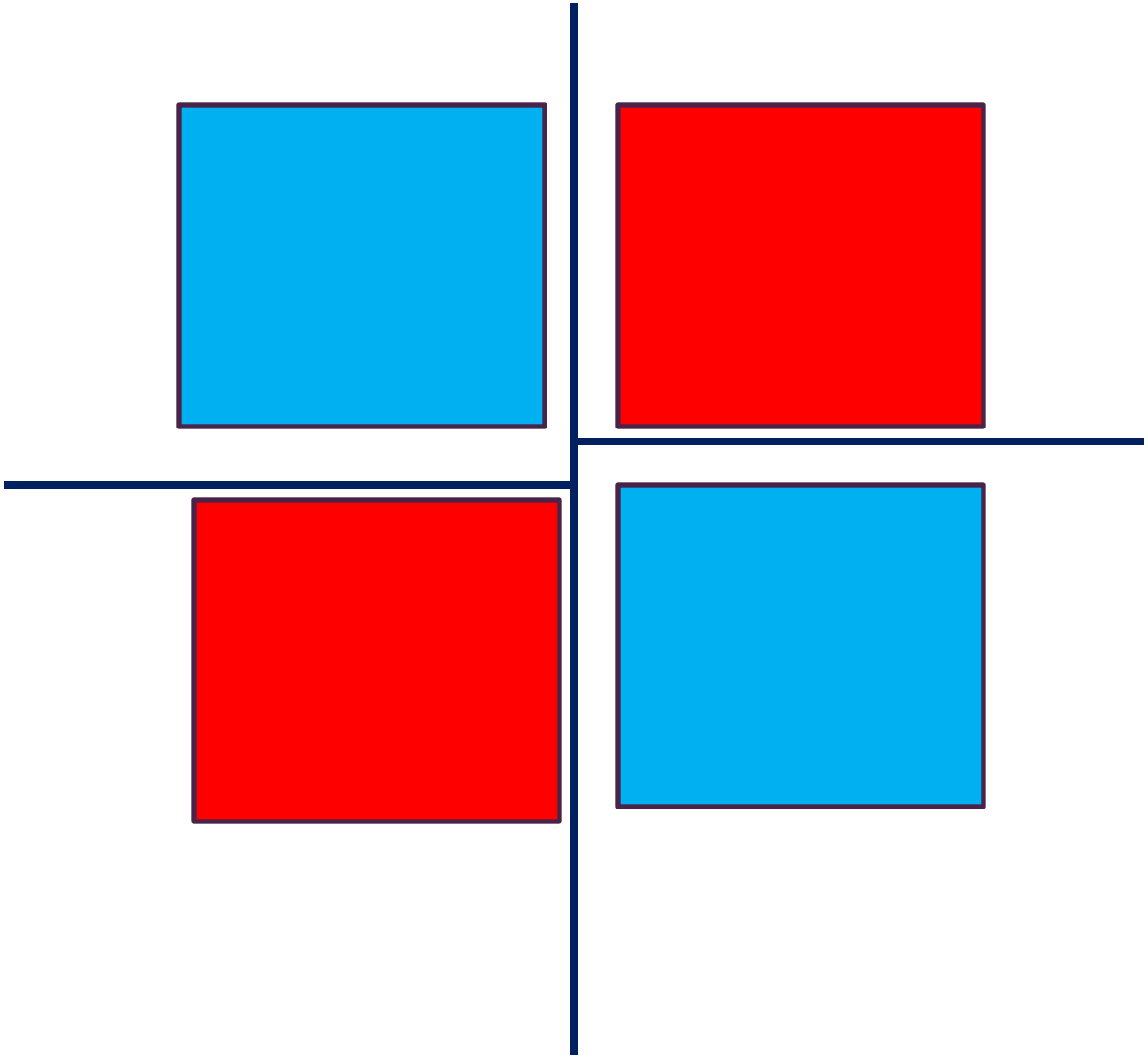
Classifier Bias — Decision Tree or Linear Perceptron?



Classifier Bias — Decision Tree or Linear Perceptron?



Classifier Bias — Decision Tree or Linear Perceptron?



Outline

- Knowledge Representation using First-Order Logic
- ~~Inference in First-Order Logic~~
- Probability, Bayesian Networks
- Machine Learning

- Questions on any topic

- Review pre-mid-term material if time and class interest