CS-171 Final Review

• First-Order Logic, Knowledge Representation
  • (8.1-8.5, 9.1-9.2)

• Probability & Bayesian Networks
  • (13, 14.1-14.5)

• Machine Learning
  • (18.1-18.12, 20.1-20.2)

• Questions on any topic

• Pre-mid-term material if time and class interest

• Please review your quizzes, mid-term, & old tests
  • At least one question from a prior quiz or old CS-171 test will appear on the Final Exam (and all other tests)
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  ¬, ⇒, ∧, ∨, ⇔
- Equality  =
- Quantifiers  ∀, ∃
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
  - $\land$ and
  - $\lor$ 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.”
  - \( \neg \forall x \ P(x) \equiv \exists x \neg P(x) \)
  - \( \neg \exists x \ P(x) \equiv \forall x \neg P(x) \)
Universal Quantification ∀

- ∀ means “for all”
- Allows us to make statements about all objects that have certain properties
- Can now state general rules:
  - ∀ x  King(x) => Person(x)  “All kings are persons.”
  - ∀ x  Person(x) => HasHead(x)  “Every person has a head.”
  - ∀ i  Integer(i) => Integer(plus(i,1))  “If i is an integer then i+1 is an integer.”

Note that
- ∀ x  King(x) ∧ Person(x)  is not correct!
- This would imply that all objects x are Kings and are People
- ∀ x  King(x) => Person(x) is the correct way to say this

**Note that => is the natural connective to use with ∀.**
Existential Quantification $\exists$

- $\exists x$ means “there exists an $x$ such that….” (at least one object $x$)

- Allows us to make statements about some object without naming it

- Examples:

  $\exists x$ King($x$)  “Some object is a king.”

  $\exists x$ Lives_in(John, Castle($x$))  “John lives in somebody’s castle.”

  $\exists i$ Integer($i$) $\land$ GreaterThan($i,0$)  “Some integer is greater than zero.”

**Note that $\land$ is the natural connective to use with $\exists$**

(And remember that $=>$ is the natural connective to use with $\forall$ )
Combining Quantifiers --- Order (Scope)

The order of “unlike” quantifiers is important.
\[ \forall x \exists y \text{ Loves}(x, y) \]
- For everyone (“all x”) there is someone (“exists y”) whom they love

\[ \exists y \forall x \text{ Loves}(x, y) \]
- there is someone (“exists y”) whom everyone loves (“all x”)

Clearer with parentheses: \[ \exists y \big( \forall x \text{ Loves}(x, y) \big) \]

The order of “like” quantifiers does not matter.
\[ \forall x \forall y \text{ P}(x, y) \equiv \forall y \forall x \text{ P}(x, y) \]
\[ \exists x \exists y \text{ P}(x, y) \equiv \exists y \exists x \text{ P}(x, y) \]
De Morgan’s Law for Quantifiers

<table>
<thead>
<tr>
<th>De Morgan’s Rule</th>
<th>Generalized De Morgan’s Rule</th>
</tr>
</thead>
<tbody>
<tr>
<td>$P \land Q \equiv \neg(\neg P \lor \neg Q)$</td>
<td>$\forall x \ P \equiv \exists x (\neg P)$</td>
</tr>
<tr>
<td>$P \lor Q \equiv \neg(\neg P \land \neg Q)$</td>
<td>$\exists x \ P \equiv \neg \forall x (\neg P)$</td>
</tr>
<tr>
<td>$\neg(P \land Q) \equiv \neg P \lor \neg Q$</td>
<td>$\neg \forall x \ P \equiv \exists x (\neg P)$</td>
</tr>
<tr>
<td>$\neg(P \lor Q) \equiv \neg P \land \neg Q$</td>
<td>$\neg \exists x \ P \equiv \forall x (\neg P)$</td>
</tr>
</tbody>
</table>

Rule is simple: if you bring a negation inside a disjunction or a conjunction, always switch between them (or $\rightarrow$ and, and $\rightarrow$ or).
Brothers are siblings

∀ x, y  Brother(x, y) ⇒ Sibling(x, y).

“Sibling” is symmetric

∀ x, y  Sibling(x, y) ⇔ Sibling(y, x).

One’s mother is one’s female parent

∀ x, y  Mother(x, y) ⇔ (Female(x) ∧ Parent(x, y)).

A first cousin is a child of a parent’s sibling

∀ x, y  FirstCousin(x, y) ⇔ ∃ p, ps  Parent(p, x) ∧ Sibling(ps, p) ∧ Parent(ps, y)
More fun with sentences

• “All persons are mortal.”
  [Use: Person(x), Mortal(x) ]

• \( \forall x \text{ Person}(x) \Rightarrow \text{Mortal}(x) \)
• \( \forall x \neg \text{Person}(x) \lor \text{Mortal}(x) \)

• **Common Mistakes:**
• \( \forall x \text{ Person}(x) \land \text{Mortal}(x) \)

• **Note that \( \Rightarrow \) is the natural connective to use with \( \forall \).**
More fun with sentences

• “Fifi has a sister who is a cat.”
  [Use: Sister(Fifi, x), Cat(x) ]

• \( \exists x \) Sister(Fifi, x) \( \land \) Cat(x)

• **Common Mistakes:**
  • \( \exists x \) Sister(Fifi, x) \( \Rightarrow \) Cat(x)

• **Note that \( \land \) is the natural connective to use with \( \exists \)**
More fun with sentences

• “For every food, there is a person who eats that food.”
• [Use: Food(x), Person(y), Eats(y, x) ]

• All are correct:
  • \( \forall x \exists y \text{ Food}(x) \Rightarrow [ \text{ Person}(y) \land \text{ Eats}(y, x) ] \)
  • \( \forall x \text{ Food}(x) \Rightarrow \exists y [ \text{ Person}(y) \land \text{ Eats}(y, x) ] \)
  • \( \forall x \exists y \neg \text{ Food}(x) \lor [ \text{ Person}(y) \land \text{ Eats}(y, x) ] \)
  • \( \forall x \exists y [ \neg \text{ Food}(x) \lor \text{ Person}(y) ] \land [ \neg \text{ Food}(x) \lor \text{ Eats}(y, x) ] \)
  • \( \forall x \exists y [ \text{ Food}(x) \Rightarrow \text{ Person}(y) ] \land [ \text{ Food}(x) \Rightarrow \text{ Eats}(y, x) ] \)

• Common Mistakes:
  • \( \forall x \exists y [ \text{ Food}(x) \land \text{ Person}(y) ] \Rightarrow \text{ Eats}(y, x) \)
  • \( \forall x \exists y \text{ Food}(x) \land \text{ Person}(y) \land \text{ Eats}(y, x) \)
More fun with sentences

• “Every person eats every food.”
  [Use: Person (x), Food (y), Eats(x, y) ]

• \( \forall x \forall y \ [ \ \text{Person}(x) \land \text{Food}(y) \ ] \Rightarrow \text{Eats}(x, y) \)

• \( \forall x \forall y \ \neg \text{Person}(x) \lor \neg \text{Food}(y) \lor \text{Eats}(x, y) \)

• \( \forall x \forall y \ \text{Person}(x) \Rightarrow [ \ \text{Food}(y) \Rightarrow \text{Eats}(x, y) ] \)

• \( \forall x \forall y \ \text{Person}(x) \Rightarrow [ \ \neg \text{Food}(y) \lor \text{Eats}(x, y) ] \)

• \( \forall x \forall y \ \neg \text{Person}(x) \lor [ \ \text{Food}(y) \Rightarrow \text{Eats}(x, y) ] \)

• **Common Mistakes:**

  • \( \forall x \forall y \ \text{Person}(x) \Rightarrow [ \text{Food}(y) \land \text{Eats}(x, y) ] \)

  • \( \forall x \forall y \ \text{Person}(x) \land \text{Food}(y) \land \text{Eats}(x, y) \)
More fun with sentences

• “All greedy kings are evil.”
  [Use: King(x), Greedy(x), Evil(x) ]

• \( \forall x \ [ \text{Greedy}(x) \land \text{King}(x) ] \Rightarrow \text{Evil}(x) \)
• \( \forall x \ \neg \text{Greedy}(x) \lor \neg \text{King}(x) \lor \text{Evil}(x) \)
• \( \forall x \ \text{Greedy}(x) \Rightarrow [ \text{King}(x) \Rightarrow \text{Evil}(x) ] \)

• Common Mistakes:
• \( \forall x \ \text{Greedy}(x) \land \text{King}(x) \land \text{Evil}(x) \)
More fun with sentences

• “Everyone has a favorite food.”
  • [Use: Person(x), Food(y), Favorite(y, x)]

• \( \forall x \\exists y \text{ Person}(x) \Rightarrow [ \text{ Food}(y) \land \text{ Favorite}(y, x) ] \)
• \( \forall x \text{ Person}(x) \Rightarrow \exists y [ \text{ Food}(y) \land \text{ Favorite}(y, x) ] \)
• \( \forall x \\exists y \neg \text{ Person}(x) \lor [ \text{ Food}(y) \land \text{ Favorite}(y, x) ] \)
• \( \forall x \\exists y [ \neg \text{ Person}(x) \lor \text{ Food}(y) ] \land [ \neg \text{ Person}(x) \lor \text{ Favorite}(y, x) ] \)

• Common Mistakes:
  • \( \forall x \exists y [ \text{ Person}(x) \Rightarrow \text{ Food}(y) ] \land [ \text{ Person}(x) \Rightarrow \text{ Favorite}(y, x) ] \)

  • \( \forall x \exists y [ \text{ Person}(x) \land \text{ Food}(y) ] \Rightarrow \text{ Favorite}(y, x) \)
• \( \forall x \exists y \text{ Person}(x) \land \text{ Food}(y) \land \text{ Favorite}(y, x) \)
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, ...

- **Your job, as a Knowledge Engineer, is to construct KB so it is true *exactly* for your world and intended interpretation.**
An interpretation and possible world satisfies a wff (sentence) if the wff has the value “true” under that interpretation in that possible world.

A domain and an interpretation that satisfies a wff is a model of that wff.

Any wff that has the value “true” in all possible worlds and under all interpretations is valid.

Any wff that does not have a model under any interpretation is inconsistent or unsatisfiable.

Any wff that is true in at least one possible world under at least one interpretation is satisfiable.

If a wff w has a value true under all the models of a set of sentences KB then KB logically entails w.
Unification

• Recall: $\text{Subst}(\theta, p) = \text{result of substituting } \theta \text{ into sentence } p$

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

$\text{Unify}(p,q) = \theta$ where $\text{Subst}(\theta, p) = \text{Subst}(\theta, q)$

• Example:
  $p = \text{Knows}(\text{John}, x)$
  $q = \text{Knows}(\text{John}, \text{Jane})$

$\text{Unify}(p,q) = \{x/\text{Jane}\}$
Unification examples

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

<table>
<thead>
<tr>
<th>p</th>
<th>q</th>
<th>θ</th>
</tr>
</thead>
<tbody>
<tr>
<td>Knows(John,x)</td>
<td>Knows(John,Jane)</td>
<td>{x/Jane}</td>
</tr>
<tr>
<td>Knows(John,x)</td>
<td>Knows(y,OJ)</td>
<td>{x/OJ,y/John}</td>
</tr>
<tr>
<td>Knows(John,x)</td>
<td>Knows(y,Mother(y))</td>
<td>{y/John,x/Mother(John)}</td>
</tr>
<tr>
<td>Knows(John,x)</td>
<td>Knows(x,OJ)</td>
<td>{fail}</td>
</tr>
</tbody>
</table>

- 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 $\text{Knows(John, } x \text{)}$ and $\text{Knows(y, } z \text{)},$
  
  $\theta = \{y/\text{John, } x/z \}$ or $\theta = \{y/\text{John, } x/\text{John, } z/\text{John} \}$

- 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.
  
  $\text{MGU} = \{ y/\text{John, } x/z \}$

- General algorithm in Figure 9.1 in the text
Unification Algorithm

function UNIFY(x, y, θ) returns a substitution to make x and y identical
inputs: x, a variable, constant, list, or compound expression
        y, a variable, constant, list, or compound expression
        θ, the substitution built up so far (optional, defaults to empty)

if θ = failure then return failure
else if x = y then return θ
else if VARIABLE?(x) then return UNIFY-VAR(x, y, θ)
else if VARIABLE?(y) then return UNIFY-VAR(y, x, θ)
else if COMPOUND?(x) and COMPOUND?(y) then
    return UNIFY(x.ARGS, y.ARGS, UNIFY(x.OP, y.OP, θ))
else if LIST?(x) and LIST?(y) then
    return UNIFY(x.REST, y.REST, UNIFY(x.FIRST, y.FIRST, θ))
else return failure

function UNIFY-VAR(var, x, θ) returns a substitution

if \{var/val\} ∈ θ then return UNIFY(val, x, θ)
else if \{x/val\} ∈ θ then return UNIFY(var, val, θ)
else if OCCUR-CHECK?(var, x) then return failure
else return add \{var/x\} to θ

Figure 9.1 The unification algorithm. The algorithm works by comparing the structures of the inputs, element by element. The substitution θ that is the argument to UNIFY is built up along the way and is used to make sure that later comparisons are consistent with bindings that were established earlier. In a compound expression such as F(A, B), the Op field picks out the function symbol F and the ARGS field picks out the argument list (A, B).
Knowledge engineering in FOL

1. Identify the task
2. Assemble the relevant knowledge
3. Decide on a vocabulary of predicates, functions, and constants
4. Encode general knowledge about the domain
5. Encode a description of the specific problem instance
6. Pose queries to the inference procedure and get answers
7. Debug the knowledge base
The electronic circuits domain

1. Identify the task
   – Does the circuit actually add properly?

2. Assemble the relevant knowledge
   – Composed of wires and gates; Types of gates (AND, OR, XOR, NOT)
   – Irrelevant: size, shape, color, cost of gates

3. Decide on a vocabulary
   – Alternatives:
     - Type($X_1$) = XOR (function)
     Type($X_1$, XOR) (binary predicate)
     XOR($X_1$)
     (unary predicate)
The electronic circuits domain

4. Encode general knowledge of the domain
   - $\forall t_1, t_2 \ \text{Connected}(t_1, t_2) \Rightarrow \text{Signal}(t_1) = \text{Signal}(t_2)$
   - $\forall t \ \text{Signal}(t) = 1 \lor \text{Signal}(t) = 0$
   - $1 \neq 0$
   - $\forall t_1, t_2 \ \text{Connected}(t_1, t_2) \Rightarrow \text{Connected}(t_2, t_1)$
   - $\forall g \ \text{Type}(g) = \text{OR} \Rightarrow \text{Signal}(\text{Out}(1,g)) = 1 \iff \exists n \ \text{Signal}(\text{In}(n,g)) = 1$
   - $\forall g \ \text{Type}(g) = \text{AND} \Rightarrow \text{Signal}(\text{Out}(1,g)) = 0 \iff \exists n \ \text{Signal}(\text{In}(n,g)) = 0$
   - $\forall g \ \text{Type}(g) = \text{XOR} \Rightarrow \text{Signal}(\text{Out}(1,g)) = 1 \iff \text{Signal}(\text{In}(1,g)) \neq \text{Signal}(\text{In}(2,g))$
   - $\forall g \ \text{Type}(g) = \text{NOT} \Rightarrow \text{Signal}(\text{Out}(1,g)) \neq \text{Signal}(\text{In}(1,g))$
The electronic circuits domain

5. Encode the specific problem instance

- Type($X_1$) = XOR
- Type($A_1$) = AND
- Type($O_1$) = OR
- Type($X_2$) = XOR
- Type($A_2$) = AND

Connected(Out(1,$X_1$),In(1,$X_2$))
Connected(Out(1,$X_1$),In(2,$A_2$))
Connected(Out(1,$A_2$),In(1,$O_1$))
Connected(Out(1,$A_1$),In(2,$O_1$))
Connected(Out(1,$X_2$),Out(1,$C_1$))
Connected(Out(1,$O_1$),Out(2,$C_1$))

Connected(In(1,$C_1$),In(1,$X_1$))
Connected(In(1,$C_1$),In(1,$A_1$))
Connected(In(2,$C_1$),In(2,$X_1$))
Connected(In(2,$C_1$),In(2,$A_1$))
Connected(In(3,$C_1$),In(2,$X_2$))
Connected(In(3,$C_1$),In(1,$A_2$))
6. Pose queries to the inference procedure
   What are the possible sets of values of all the terminals for the adder circuit?

   \[ \exists i_1, i_2, i_3, o_1, o_2 \quad \text{Signal}(\text{In}(1, C_1)) = i_1 \land \text{Signal}(\text{In}(2, C_1)) = i_2 \land \text{Signal}(\text{In}(3, C_1)) = i_3 \land \text{Signal}(\text{Out}(1, C_1)) = o_1 \land \text{Signal}(\text{Out}(2, C_1)) = o_2 \]

7. Debug the knowledge base
   May have omitted assertions like \(1 \neq 0\)
CS-171 Final Review

• First-Order Logic, Knowledge Representation
  • (8.1-8.5, 9.1-9.2)
• Probability & Bayesian Networks
  • (13, 14.1-14.5)
• Machine Learning
  • (18.1-18.12, 20.1-20.2)
• Questions on any topic
• Pre-mid-term material if time and class interest
• Please review your quizzes, mid-term, & old tests
  • At least one question from a prior quiz or old CS-171 test will appear on the Final Exam (and all other tests)
CS-171 Final Review

- First-Order Logic, Knowledge Representation
  - (8.1-8.5, 9.1-9.2)
- Probability & Bayesian Networks
  - (13, 14.1-14.5)
- Machine Learning
  - (18.1-18.12, 20.1-20.2)
- Questions on any topic
- Pre-mid-term material if time and class interest
- Please review your quizzes, mid-term, & old tests
  - At least one question from a prior quiz or old CS-171 test will appear on the Final Exam (and all other tests)
You will be expected to know

• Basic probability notation/definitions:
  – Probability model, unconditional/prior and conditional/posterior probabilities, factored representation (= variable/value pairs), random variable, (joint) probability distribution, probability density function (pdf), marginal probability, (conditional) independence, normalization, etc.

• Basic probability formulae:
  – Probability axioms, product rule, Bayes’ rule.

• How to use Bayes’ rule:
  – Naïve Bayes model (naïve Bayes classifier)
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

• **Probability Axioms:**
  – $0 \leq P(a) \leq 1$
  – $P(\text{NOT}(a)) = 1 - P(a)$ $\Rightarrow \Sigma_A P(A) = 1$
  – $P(\text{true}) = 1$
  – $P(\text{false}) = 0$
  – $P(A \text{ OR } B) = P(A) + P(B) - P(A \text{ AND } B)$

• Any agent that holds degrees of beliefs that contradict these axioms will act irrationally in some cases

• **Rational agents cannot violate probability theory.**
  – Acting otherwise results in irrational behavior.
Concepts of Probability

• **Unconditional Probability**
  – $P(a)$, the probability of “a” being true, or $P(a=True)$
  – Does not depend on anything else to be true (unconditional)
  – Represents the probability prior to further information that may adjust it (prior)

• **Conditional Probability**
  – $P(a|b)$, the probability of “a” being true, given that “b” is true
  – Relies on “b” = true (conditional)
  – Represents the prior probability adjusted based upon new information “b” (posterior)
  – Can be generalized to more than 2 random variables:
    ▪ e.g. $P(a|b, c, d)$

• **Joint Probability**
  – $P(a, b) = P(a \land b)$, the probability of “a” and “b” both being true
  – Can be generalized to more than 2 random variables:
    ▪ e.g. $P(a, b, c, d)$
Random Variables

• **Random Variable**:  
  – Basic element of probability assertions  
  – Similar to CSP variable, but values reflect probabilities not constraints.  
    ▪ Variable: A  
    ▪ Domain: \{a_1, a_2, a_3\} <-- events / outcomes

• **Types of Random Variables**:  
  – **Boolean** random variables = \{true, false\}  
    ▪ e.g., Cavity (= do I have a cavity?)
  
  – **Discrete** random variables = One value from a set of values  
    ▪ e.g., Weather is one of <sunny, rainy, cloudy, snow>
  
  – **Continuous** random variables = A value from within constraints  
    ▪ e.g., Current temperature is bounded by (10°, 200°)

• **Domain values must be exhaustive and mutually exclusive**:  
  – One of the values must always be the case (Exhaustive)  
  – Two of the values cannot both be the case (Mutually Exclusive)
Basic Probability Relationships

• \( P(\overline{A}) + P(A) = 1 \)
  – Implies that \( P(\overline{A}) = 1 - P(A) \)

• \( P(A, B) = P(A \land B) = P(A) + P(B) - P(A \lor B) \)
  – Implies that \( P(A \lor B) = P(A) + P(B) - P(A \land B) \)

• \( P(A \mid B) = \frac{P(A, B)}{P(B)} \)
  – Conditional probability; “Probability of \( A \) given \( B \)”

• \( P(A, B) = P(A \mid B) P(B) \)
  – Product Rule (Factoring); applies to any number of variables
  – \( P(a, b, c, \ldots z) = P(a \mid b, c, \ldots z) P(b \mid c, \ldots z) P(c|\ldots z) \ldots P(z) \)

• \( P(A) = \sum_{B, C} P(A, B, C) \)
  – Sum Rule (Marginal Probabilities); for any number of variables
  – \( P(A, D) = \sum_{B} \sum_{C} P(A, B, C, D) \)

• \( P(B \mid A) = \frac{P(A \mid B) P(B)}{P(A)} \)
  – Bayes’ Rule; for any number of variables
Summary of Probability Rules

• **Product Rule:**
  - \( P(a, b) = P(a | b) P(b) = P(b | a) P(a) \)
  - Probability of “a” and “b” occurring is the same as probability of “a” occurring given “b” is true, times the probability of “b” occurring.
    - e.g., \( P(\text{rain, cloudy}) = P(\text{rain | cloudy}) \cdot P(\text{cloudy}) \)

• **Sum Rule:** (AKA Law of Total Probability)
  - \( P(a) = \sum_b P(a, b) = \sum_b P(a | b) P(b), \) where B is any random variable
  - Probability of “a” occurring is the same as the sum of all joint probabilities including the event, provided the joint probabilities represent all possible events.
  - Can be used to “marginalize” out other variables from probabilities, resulting in prior probabilities also being called marginal probabilities.
    - e.g., \( P(\text{rain}) = \sum_{\text{Windspeed}} P(\text{rain, Windspeed}) \)
    - where Windspeed = \{0-10mph, 10-20mph, 20-30mph, etc.\}

• **Bayes’ Rule:**
  - \( P(b | a) = \frac{P(a | b) P(b)}{P(a)} \)
  - Acquired from rearranging the product rule.
  - Allows conversion between conditionals, from \( P(a | b) \) to \( P(b | a) \).
    - e.g., \( b = \text{disease}, a = \text{symptoms} \)
    - More natural to encode knowledge as \( P(a | b) \) than as \( P(b | a) \).
Full Joint Distribution

• We can fully specify a probability space by constructing a **full joint distribution**:
  – A full joint distribution contains a probability for every possible combination of variable values.
  – E.g., \( P( J=f \land M=t \land A=t \land B=t \land E=f ) \)

• From a full joint distribution, the product rule, sum rule, and Bayes’ rule can create any desired joint and conditional probabilities.
Independence

• **Formal Definition:**
  – 2 random variables A and B are independent iff:
    \[ P(a, b) = P(a) \cdot P(b), \quad \text{for all values } a, b \]

• **Informal Definition:**
  – 2 random variables A and B are independent iff:
    \[ P(a \mid b) = P(a) \quad \text{OR} \quad P(b \mid a) = P(b), \quad \text{for all values } a, b \]
  – \( P(a \mid b) = P(a) \) tells us that knowing \( b \) provides no change in our probability for \( a \), and thus \( b \) contains no information about \( a \).

• Also known as **marginal independence**, as all other variables have been marginalized out.

• In practice true independence is very rare:
  – “butterfly in China” effect
  – Conditional independence is much more common and useful
Conditional Independence

• **Formal Definition:**
  – 2 random variables A and B are conditionally independent given C iff:
    \[ P(a, b | c) = P(a | c) P(b | c), \quad \text{for all values } a, b, c \]

• **Informal Definition:**
  – 2 random variables A and B are conditionally independent given C iff:
    \[ P(a | b, c) = P(a | c) \quad \text{OR} \quad P(b | a, c) = P(b | c), \quad \text{for all values } a, b, c \]
  – \( P(a | b, c) = P(a | c) \) tells us that learning about b, given that we already know c, provides no change in our probability for a, and thus b contains no information about a beyond what c provides.

• **Naïve Bayes Model:**
  – Often a single variable can directly influence a number of other variables, all of which are conditionally independent, given the single variable.
  – E.g., k different symptom variables \( X_1, X_2, \ldots X_k \), and C = disease, reducing to:
    \[ P(X_1, X_2, \ldots X_k | C) = \prod P(X_i | C) \]
Examples of Conditional Independence

- **H=Heat, S=Smoke, F=Fire**
  - \( P(H, S \mid F) = P(H \mid F) P(S \mid F) \)
  - \( P(S \mid F, S) = P(S \mid F) \)
  - If we know there is/is not a fire, observing heat tells us no more information about smoke

- **F=Fever, R=RedSpots, M=Measles**
  - \( P(F, R \mid M) = P(F \mid M) P(R \mid M) \)
  - \( P(R \mid M, F) = P(R \mid M) \)
  - If we know we do/don’t have measles, observing fever tells us no more information about red spots

- **C=SharpClaws, F=SharpFangs, S=Species**
  - \( P(C, F \mid S) = P(C \mid S) P(F \mid S) \)
  - \( P(F \mid S, C) = P(F \mid S) \)
  - If we know the species, observing sharp claws tells us no more information about sharp fangs
Review Bayesian Networks (Chapter 14.1-5)

• **You will be expected to know:**

• **Basic concepts and vocabulary of Bayesian networks.**
  – Nodes represent random variables.
  – Directed arcs represent (informally) direct influences.
  – Conditional probability tables, $P(X_i \mid \text{Parents}(X_i))$.

• **Given a Bayesian network:**
  – Write down the full joint distribution it represents.
  – Inference by Variable Elimination

• **Given a full joint distribution in factored form:**
  – Draw the Bayesian network that represents it.

• **Given a variable ordering and background assertions of conditional independence among the variables:**
  – Write down the factored form of the full joint distribution, as simplified by the conditional independence assertions.
Bayesian Networks

- Represent dependence/independence via a directed graph
  - Nodes = random variables
  - Edges = direct dependence
- Structure of the graph $\Leftrightarrow$ Conditional independence

- 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 (of each variable given its parents)
Bayesian Network

• A Bayesian network specifies a joint distribution in a structured form:

\[ p(A,B,C) = p(C|A,B)p(A|B)p(B) = p(C|A,B)p(A)p(B) \]

• Dependence/independence represented via a directed graph:
  - Node = random variable
  - Directed Edge = conditional dependence
  - Absence of Edge = conditional independence

• Allows concise view of joint distribution relationships:
  - Graph nodes and edges show conditional relationships between variables.
  - Tables provide probability data.
Examples of 3-way Bayesian Networks

Independent Causes:
A Earthquake
B Burglary
C Alarm

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

You heard alarm, and observe Earthquake
…. It explains away burglary

Nodes: Random Variables
A, B, C

Edges: P(Xi | Parents) → Directed edge from parent nodes to Xi
A → C
B → C
Examples of 3-way Bayesian Networks

Marginal Independence:
\[ p(A, B, C) = p(A) \ p(B) \ p(C) \]

Nodes: Random Variables
A, B, C

Edges: P(Xi | Parents) → Directed edge from parent nodes to Xi
No Edge!
Extended example of 3-way Bayesian Networks

Common Cause
A: Fire
B: Heat
C: Smoke

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

B and C are conditionally independent given A

“Where there’s Smoke, there’s Fire.”

If we see Smoke, we can infer Fire.

If we see Smoke, observing Heat tells us very little additional information.
Examples of 3-way Bayesian Networks

Markov Dependence
A Rain on Mon
B Ran on Tue
C Rain on Wed

Markov dependence:
\[ p(A,B,C) = p(C|B) \, p(B|A) \, p(A) \]

A affects B and B affects C
Given B, A and C are independent

e.g.
If it rains today, it will rain tomorrow with 90%

On Wed morning...
If you know it rained yesterday, it doesn’t matter whether it rained on Mon

Nodes: Random Variables
A, B, C

Edges: \( P(X_i \mid \text{Parents}) \) → Directed edge from parent nodes to \( X_i \)
A → B
B → C
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: Step 1

- Order the variables in terms of influence (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.
Constructing this Bayesian Network: Step 2

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

- There are 3 conditional probability tables (CPDs) to be determined: \( P(J \mid A) \), \( P(M \mid A) \), \( P(A \mid E, B) \)
  - Requiring \( 2 + 2 + 4 = 8 \) probabilities

- And 2 marginal probabilities \( P(E) \), \( P(B) \) -> 2 more probabilities

- Where do these probabilities come from?
  - Expert knowledge
  - From data (relative frequency estimates)
  - Or a combination of both - see discussion in Section 20.1 and 20.2 (optional)
The Resulting Bayesian Network

- **Burglary**: \( P(B) = 0.001 \)
- **Earthquake**: \( P(E) = 0.002 \)
- **Alarm**
  - **JohnCalls**:
    | \( A \) | \( P(J) \) |
    |---|---|
    | \( t \) | 0.90 |
    | \( f \) | 0.05 |
  - **MaryCalls**:
    | \( A \) | \( P(M) \) |
    |---|---|
    | \( t \) | 0.70 |
    | \( f \) | 0.01 |

<table>
<thead>
<tr>
<th>( B )</th>
<th>( E )</th>
<th>( P(A) )</th>
</tr>
</thead>
<tbody>
<tr>
<td>( t )</td>
<td>( t )</td>
<td>0.95</td>
</tr>
<tr>
<td>( t )</td>
<td>( f )</td>
<td>0.94</td>
</tr>
<tr>
<td>( f )</td>
<td>( t )</td>
<td>0.29</td>
</tr>
<tr>
<td>( f )</td>
<td>( f )</td>
<td>0.001</td>
</tr>
</tbody>
</table>
The Bayesian Network from a different Variable Ordering

(b)
Computing Probabilities from a Bayesian Network

Shown below is the Bayesian network for the Burglar Alarm problem, i.e.,
\[ P(J, M, A, B, E) = P(J \mid A) \cdot P(M \mid A) \cdot P(A \mid B, E) \cdot P(B) \cdot P(E). \]

Suppose we wish to compute \( P( J=f \land M=t \land A=t \land B=t \land E=f ) \):

\[
P( J=f \land M=t \land A=t \land B=t \land E=f ) \\
= P( J=f \mid A=t ) \cdot P( M=t \mid A=t ) \cdot P( A=t \mid B=t \land E=f ) \cdot P( B=t ) \cdot P( E=f ) \\
= .10 \cdot .70 \cdot .94 \cdot .001 \cdot .998
\]

Note: \( P( E=f ) = [ 1 - P( E=t ) ] = [ 1 - .002 ] = .998 \)

\( P( J=f \mid A=t ) = [ 1 - P( J=t \mid A=t ) ] = .10 \)
Inference in Bayesian Networks

Simple Example

Query Variables A, B

Hidden Variable C

Evidence Variable D

Suppose that two different diseases influence some imaginary internal body temperature regulator, which in turn influences whether fever is present.

Note: Not an anatomically correct model of how diseases cause fever!
Inference in Bayesian Networks

- \( X = \{ X_1, X_2, ..., X_k \} = \text{query variables} \) of interest
- \( E = \{ E_1, ..., E_l \} = \text{evidence variables} \) that are observed
- \( Y = \{ Y_1, ..., Y_m \} = \text{hidden variables} \) (nonevidence, nonquery)

- What is the posterior distribution of \( X \), given \( E \)?
  - \( P( X | e ) = \alpha \sum_y P( X, y, e ) \)
    Normalizing constant \( \alpha = \sum_x \sum_y P( X, y, e ) \)

- What is the most likely assignment of values to \( X \), given \( E \)?
  - \( \arg\max_x P( x | e ) = \arg\max_x \sum_y P( x, y, e ) \)
What is the posterior conditional distribution of our query variables, given that fever was observed?

\[
P(A,B|d) = \alpha \sum_c P(A,B,c,d) = \alpha \sum_c P(A)P(B)P(c|A,B)P(d|c) = \alpha P(A)P(B)\sum_c P(c|A,B)P(d|c) \\
= \alpha .05 \times .02 \times \{.95 \times .95 + .995 \times .002\} \approx \alpha .014
\]

\[
P(\neg a,b|d) = \alpha P(\neg a)P(b)\sum_c P(c|\neg a,b)P(d|c) = \alpha P(\neg a)P(b)\{ P(c|\neg a,b)P(d|c)+P(\neg c|\neg a,b)P(d|\neg c) \} \\
= \alpha .95 \times .02 \times \{.90 \times .95 + .01 \times .002\} \approx \alpha .0162 \approx .248
\]

\[
P(a,\neg b|d) = \alpha P(a)P(\neg b)\sum_c P(c|a,\neg b)P(d|c) = \alpha P(a)P(\neg b)\{ P(c|a,\neg b)P(d|c)+P(\neg c|a,\neg b)P(d|\neg c) \} \\
= \alpha .05 \times .98 \times \{.90 \times .95 + .995 \times .002\} \approx \alpha .0419 \approx .642
\]

\[
P(\neg a,\neg b|d) = \alpha P(\neg a)P(\neg b)\sum_c P(c|\neg a,\neg b)P(d|c) = \alpha P(\neg a)P(\neg b)\{ P(c|\neg a,\neg b)P(d|c)+P(\neg c|\neg a,\neg b)P(d|\neg c) \} \\
= \alpha .95 \times .98 \times \{.005 \times .95 + .995 \times .002\} \approx \alpha .00627 \approx .096
\]

\[
\alpha \approx 1 / (.000903+.0162+.0419+.00627) \approx 1 / .06527 \approx 15.32 \quad \text{[Note: } \alpha = \text{ normalization constant, p. 493]} \]
CS-171 Final Review

- First-Order Logic, Knowledge Representation
  - (8.1-8.5, 9.1-9.2)
- Probability & Bayesian Networks
  - (13, 14.1-14.5)
- Machine Learning
  - (18.1-18.12, 20.1-20.2)
- Questions on any topic
- Pre-mid-term material if time and class interest
- Please review your quizzes, mid-term, & old tests
  - At least one question from a prior quiz or old CS-171 test will appear on the Final Exam (and all other tests)
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:**

```
  1110  0100
  0000  1010

  0010  1101

  1111  0101

  0100

  0001
```
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.,
  \( h(x; \theta) \) is “close” to \( f(x) \) for all training data points \( x \)

\( \theta \) are the parameters of our predictor \( h(\ldots) \)

• Examples:
  - \( h(x; \theta) = \text{sign}(w_1x_1 + w_2x_2 + w_3) \)
  - \( h_k(x) = (x_1 \text{ OR } x_2) \text{ AND } (x_3 \text{ OR NOT}(x_4)) \)
Empirical Error Functions

• Empirical error function:
  \[ E(h) = \sum_{x} \text{distance}[h(x; \theta), f] \]

  e.g., distance = squared error if h and f are real-valued (regression)
  distance = delta-function if h and f are categorical (classification)

  Sum is over all training pairs in the training data D

In learning, we get to choose

  1. what class of functions h(..) that we want to learn
     - potentially a huge space! (“hypothesis space”)

  2. what error function/distance to use
     - should be chosen to reflect real “loss” in problem
     - but often chosen for mathematical/algorithmic convenience
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
Pseudocode for Decision tree learning

```plaintext
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
```
Entropy with only 2 outcomes

Consider 2 class problem: \( p = \) probability of class 1, \( 1 - p = \) probability of class 2

In binary case, \( H(p) = - p \log p - (1-p) \log (1-p) \)
Information Gain

- $H(p) =$ entropy of class distribution at a particular node

- $H(p \mid A) =$ conditional entropy = average entropy of conditional class distribution, after we have partitioned the data according to the values in $A$

- Gain($A) = H(p) - H(p \mid A)$

- Simple rule in decision tree learning
  - At each internal node, split on the node with the largest information gain (or equivalently, with smallest $H(p \mid A)$)

- Note that by definition, conditional entropy can’t be greater than the entropy
Overfitting and Underfitting
A Complex Model

\[ Y = \text{high-order polynomial in} \ X \]
A Much Simpler Model

\[ Y = aX + b + \text{noise} \]
How Overfitting affects Prediction

Predictive Error

Underfitting

Overfitting

Error on Test Data

Error on Training Data

Ideal Range for Model Complexity

Model Complexity
Training and Validation Data

Idea: train each model on the “training data” and then test each model’s accuracy on the validation data.
The k-fold Cross-Validation Method

• Why just choose one particular 90/10 “split” of the data?
  – In principle we could do this multiple times

• “k-fold Cross-Validation” (e.g., k=10)
  – randomly partition our full data set into k disjoint subsets (each roughly of size n/k, n = total number of training data points)
  • for i = 1:10 (here k = 10)
    – train on 90% of data,
    – Acc(i) = accuracy on other 10%
  • end

  • Cross-Validation-Accuracy = 1/k ∑ i Acc(i)
  – choose the method with the highest cross-validation accuracy
  – common values for k are 5 and 10
  – Can also do “leave-one-out” where k = n
Disjoint Validation Data Sets

Full Data Set

Validation Data (aka Test Data)

Validation Data

1\textsuperscript{st} partition

2\textsuperscript{nd} partition

Training Data

3\textsuperscript{rd} partition

4\textsuperscript{th} partition

5\textsuperscript{th} partition
Classification in Euclidean Space

• A classifier is a partition of the space $\mathbf{x}$ 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 $\Leftrightarrow$ 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 $x'$, compute the distance between $x'$ and each input data point
  - Find the closest neighbor in the training data
  - Assign $x'$ 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
Overall Boundary = Piecewise Linear

Decision Region for Class 1

Decision Region for Class 2

Feature 1

Feature 2
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

$$K = 25$$

- True (“best”) decision boundary
  - In this case is linear
  - Compared to kNN: not bad!
Linear Classifiers

- Linear classifier ⇔ single linear decision boundary (for 2-class case)

- We can always represent a linear decision boundary by a linear equation:
  \[ w_1 x_1 + w_2 x_2 + \ldots + w_d x_d = \sum w_j x_j = w^t x = 0 \]

- 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
The Perceptron Classifier  (pages 729-731 in text)

• The perceptron classifier is just another name for a linear classifier for 2-class data, i.e.,
  \[ \text{output}(x) = \text{sign} ( \sum w_j x_j ) \]

• 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)
The Perceptron Classifier (pages 729-731 in text)
Two different types of perceptron output

x-axis below is $f(x) = f = \text{weighted sum of inputs}$
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
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
      \[
      \text{deltaw}_j = \eta \left( y(i) - \sigma[f(i)] \right) \partial \sigma[f(i)] x_j(i) \\
      w_j = w_j + \text{deltaw}_j
      \]
    end
  calculate termination condition
end

- Inputs: \( N \) features, \( N \) targets (class labels), learning rate \( \eta \)
- Outputs: a set of learned weights
Support Vector Machines (SVM): “Modern perceptrons” (section 18.9, R&N)

- A modern linear separator classifier
  - Essentially, a perceptron with a few extra wrinkles

- Constructs a “maximum margin separator”
  - A linear decision boundary with the largest possible distance from the decision boundary to the example points it separates
  - “Margin” = Distance from decision boundary to closest example
  - The “maximum margin” helps SVMs to generalize well

- Can embed the data in a non-linear higher dimension space
  - Constructs a linear separating hyperplane in that space
    - This can be a non-linear boundary in the original space
      - Algorithmic advantages and simplicity of linear classifiers
      - Representational advantages of non-linear decision boundaries

- Currently most popular “off-the-shelf” supervised classifier.
Constructs a “maximum margin separator”

Figure 18.30  FILES: . Support vector machine classification: (a) Two classes of points (black and white circles) and three candidate linear separators. (b) The maximum margin separator (heavy line), is at the midpoint of the margin (area between dashed lines). The support vectors (points with large circles) are the examples closest to the separator.
Can embed the data in a non-linear higher dimension space

Figure 18.31 FILES: (a) A two-dimensional training set with positive examples as black circles and negative examples as white circles. The true decision boundary, $x_1^2 + x_2^2 \leq 1$, is also shown. (b) The same data after mapping into a three-dimensional input space $(x_1^2, x_2^2, \sqrt{2}x_1x_2)$. The circular decision boundary in (a) becomes a linear decision boundary in three dimensions. Figure 18.29(b) gives a closeup of the separator in (b).
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
Multi-Layer Perceptrons (Artificial Neural Networks)
(sections 18.7.3-18.7.4 in textbook)
**Naïve Bayes Model**  
(Section 20.2.2 R&N 3rd ed.)

**Basic Idea:** We want to estimate $P(C \mid X_1,...,X_n)$, but it’s hard to think about computing the probability of a class from input attributes of an example.

**Solution:** Use Bayes’ Rule to turn $P(C \mid X_1,...,X_n)$ into an equivalent expression that involves only $P(C)$ and $P(X_i \mid C)$.

We can estimate $P(C)$ easily from the frequency with which each class appears within our training data, and $P(X_i \mid C)$ from the frequency with which each $X_i$ appears in each class $C$ within our training data.
Naïve Bayes Model

Bayes Rule: \[ P(C \mid X_1, \ldots, X_n) \text{ is proportional to } P(C) \prod_i P(X_i \mid C) \]

[Note: denominator \( P(X_1, \ldots, X_n) \) is constant for all classes, may be ignored.]

Features \( X_i \) are conditionally independent given the class variable \( C \)

- choose the class value \( c_i \) with the highest \( P(c_i \mid x_1, \ldots, x_n) \)
- simple to implement, often works very well
- e.g., spam email classification: \( X \)'s = counts of words in emails

Conditional probabilities \( P(X_i \mid C) \) 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 \( P(X_i \mid C) \) can easily be estimated from labeled data

\[ P(C = c_j) \approx \frac{\#(\text{Examples with class label } c_j)}{\#(\text{Examples})} \]

\[ P(X_i = x_{ik} \mid C = c_j) \]
\[ \approx \frac{\#(\text{Examples with } X_i \text{ value } x_{ik} \text{ and class label } c_j)}{\#(\text{Examples with class label } c_j)} \]

Usually easiest to work with logs
\[
\log \left[ P(C \mid X_1, \ldots X_n) \right]
= \log \alpha + \sum \left[ \log P(X_i \mid C) + \log P(C) \right]
\]

DANGER: Suppose ZERO examples with \( X_i \) value \( x_{ik} \) and class label \( c_j \)? An unseen example with \( X_i \) value \( x_{ik} \) will NEVER predict class label \( c_j \)!

Practical solutions: Pseudocounts, e.g., add 1 to every \#()\, etc.
Theoretical solutions: Bayesian inference, beta distribution, etc.
CS-171 Final Review

• First-Order Logic, Knowledge Representation
  • (8.1-8.5, 9.1-9.2)

• Probability & Bayesian Networks
  • (13, 14.1-14.5)

• Machine Learning
  • (18.1-18.12, 20.1-20.2)

• Questions on any topic
• Pre-mid-term material if time and class interest
• Please review your quizzes, mid-term, & old tests
  • At least one question from a prior quiz or old CS-171 test will appear on the Final Exam (and all other tests)