Final Catch-up, Review
Outline

- Propositional Logic
- Knowledge Representation using First-Order Logic
- Inference in First-Order Logic
- Probability
- Machine Learning

- Questions on any topic

- Review pre-mid-term material if time and class interest
Inference in Formal Symbol Systems: Ontology, Representation, Inference

• **Formal Symbol Systems**
  – **Symbols** correspond to **things/ideas** in the world
  – **Pattern matching** corresponds to inference

• **Ontology**: What exists in the world?
  – What must be represented?

• **Representation**: Syntax vs. Semantics
  – What’s Said vs. What’s Meant

• **Inference**: Schema vs. Mechanism
  – Proof Steps vs. Search Strategy
Ontology:
What kind of things exist in the world?
What do we need to describe and reason about?

- **Reasoning**
  - **Representation**
    - A Formal Symbol System
  - **Inference**
    - Formal Pattern Matching

- **Syntax**
  - What is said

- **Semantics**
  - What it means

- **Schema**
  - Rules of Inference

- **Execution**
  - Search Strategy
Propositional Logic --- Review

• Definitions:
  – Syntax, Semantics, Sentences, Propositions, Entails, Follows, Derives, Inference, Sound, Complete, Model, Satisfiable, Valid (or Tautology)

• Syntactic Transformations:
  – E.g., \((A \Rightarrow B) \iff (\neg A \lor B)\)

• Semantic Transformations:
  – E.g., \((KB \models \alpha) \equiv (\models (KB \Rightarrow \alpha))\)

• Truth Tables:
  – Negation, Conjunction, Disjunction, Implication, Equivalence (Biconditional)

• Inference:
  – By Model Enumeration (truth tables)
  – By Forward chaining, Backward chaining, Resolution
Recap propositional logic: Syntax

• Propositional logic is the simplest logic – illustrates basic ideas

• The proposition symbols $P_1$, $P_2$ etc are sentences

  - If $S$ is a sentence, $\neg S$ is a sentence (negation)
  - If $S_1$ and $S_2$ are sentences, $S_1 \land S_2$ is a sentence (conjunction)
  - If $S_1$ and $S_2$ are sentences, $S_1 \lor S_2$ is a sentence (disjunction)
  - If $S_1$ and $S_2$ are sentences, $S_1 \Rightarrow S_2$ is a sentence (implication)
  - If $S_1$ and $S_2$ are sentences, $S_1 \Leftrightarrow S_2$ is a sentence (biconditional)
Recap propositional logic: Semantics

Each model/world specifies true or false for each proposition symbol

E.g. \( P_{1,2} \quad P_{2,2} \quad P_{3,1} \)
\[
\begin{array}{ccc}
\text{false} & \text{true} & \text{false}
\end{array}
\]

With these symbols, 8 possible models, can be enumerated automatically.

Rules for evaluating truth with respect to a model \( m \):

\[
\begin{align*}
\neg S & \quad \text{is true iff} \quad S \quad \text{is false} \\
S_1 \land S_2 & \quad \text{is true iff} \quad S_1 \quad \text{is true and} \quad S_2 \quad \text{is true} \\
S_1 \lor S_2 & \quad \text{is true iff} \quad S_1 \quad \text{is true or} \quad S_2 \quad \text{is true} \\
S_1 \rightarrow S_2 & \quad \text{is true iff} \quad S_1 \quad \text{is false or} \quad S_2 \quad \text{is true} \\
\quad \text{(i.e.,} \quad \text{is false iff} \quad S_1 \quad \text{is true and} \quad S_2 \quad \text{is false}) \\
S_1 \leftrightarrow S_2 & \quad \text{is true iff} \quad S_1 \quad \Leftrightarrow S_2 \quad \text{is true and} \quad S_2 \quad \Leftrightarrow S_1 \quad \text{is true}
\end{align*}
\]

Simple recursive process evaluates an arbitrary sentence, e.g.,

\[
\neg P_{1,2} \land (P_{2,2} \lor P_{3,1}) = true \land (true \lor false) = true \land true = true
\]
Recap propositional logic:  
**Truth tables for connectives**

<table>
<thead>
<tr>
<th></th>
<th></th>
<th>¬P</th>
<th>P ∧ Q</th>
<th>P ∨ Q</th>
<th>P ⇒ Q</th>
<th>P ⇔ Q</th>
</tr>
</thead>
<tbody>
<tr>
<td>false</td>
<td>false</td>
<td>true</td>
<td>false</td>
<td>false</td>
<td>true</td>
<td>true</td>
</tr>
<tr>
<td>false</td>
<td>true</td>
<td>true</td>
<td>false</td>
<td>true</td>
<td>true</td>
<td>false</td>
</tr>
<tr>
<td>true</td>
<td>false</td>
<td>false</td>
<td>false</td>
<td>true</td>
<td>false</td>
<td>false</td>
</tr>
<tr>
<td>true</td>
<td>true</td>
<td>true</td>
<td>true</td>
<td>true</td>
<td>true</td>
<td>true</td>
</tr>
</tbody>
</table>

**OR:** P or Q is true or both are true.  
**XOR:** P or Q is true but not both.  
**Implication is always true when the premises are False!**
Recap propositional logic:  
**Logical equivalence and rewrite rules**

- To manipulate logical sentences we need some rewrite rules.
- Two sentences are logically equivalent iff they are true in same models: $\alpha \equiv \beta$ iff $\alpha \models \beta$ and $\beta \models \alpha$

$$
\begin{align*}
(\alpha \land \beta) & \equiv (\beta \land \alpha) \quad \text{commutativity of } \land \\
(\alpha \lor \beta) & \equiv (\beta \lor \alpha) \quad \text{commutativity of } \lor \\
((\alpha \land \beta) \land \gamma) & \equiv (\alpha \land (\beta \land \gamma)) \quad \text{associativity of } \land \\
((\alpha \lor \beta) \lor \gamma) & \equiv (\alpha \lor (\beta \lor \gamma)) \quad \text{associativity of } \lor \\
\neg(\neg \alpha) & \equiv \alpha \quad \text{double-negation elimination} \\
(\alpha \Rightarrow \beta) & \equiv (\neg \beta \Rightarrow \neg \alpha) \quad \text{contraposition} \\
(\alpha \Rightarrow \beta) & \equiv (\neg \alpha \lor \beta) \quad \text{implication elimination} \\
(\alpha \leftrightarrow \beta) & \equiv ((\alpha \Rightarrow \beta) \land (\beta \Rightarrow \alpha)) \quad \text{biconditional elimination} \\
\neg(\alpha \land \beta) & \equiv (\neg \alpha \lor \neg \beta) \quad \text{de Morgan} \\
\neg(\alpha \lor \beta) & \equiv (\neg \alpha \land \neg \beta) \quad \text{de Morgan} \\
(\alpha \land (\beta \lor \gamma)) & \equiv ((\alpha \land \beta) \lor (\alpha \land \gamma)) \quad \text{distributivity of } \land \text{ over } \lor \\
(\alpha \lor (\beta \land \gamma)) & \equiv ((\alpha \lor \beta) \land (\alpha \lor \gamma)) \quad \text{distributivity of } \lor \text{ over } \land
\end{align*}
$$
Recap propositional logic: **Entailment**

- **Entailment** means that one thing follows from another:
  \[ \text{KB} \models \alpha \]

- Knowledge base \( KB \) entails sentence \( \alpha \) if and only if \( \alpha \) is true in all worlds where \( KB \) is true
  
  - E.g., the KB containing “the Giants won and the Reds won” entails “The Giants won”.
  - E.g., \( x+y = 4 \) entails \( 4 = x+y \)
  - E.g., “Mary is Sue’s sister and Amy is Sue’s daughter” entails “Mary is Amy’s aunt.”
Review: Models (and in FOL, Interpretations)

• **Models** are formal worlds in which truth can be evaluated

• We say *m* is a model of a sentence *α* if *α* is true in *m*

• **M(α)** is the set of all models of *α*

• Then **KB ⊨ α** iff **M(KB) ⊆ M(α)**
  – E.g. **KB, = “Mary is Sue’s sister and Amy is Sue’s daughter.”**
  – **α = “Mary is Amy’s aunt.”**

• Think of KB and α as constraints, and of models m as possible states.
• **M(KB)** are the solutions to KB and **M(α)** the solutions to α.
• Then, **KB ⊨ α**, i.e., **⊨ (KB ⇒ α)**, when all solutions to KB are also solutions to α.
Review: Wumpus models

- $KB = \text{all possible wumpus-worlds consistent with the observations and the "physics" of the Wumpus world.}$
\( \alpha_1 = "[1,2] \text{ is safe}" \), \( KB \models \alpha_1 \), proved by model checking.

Every model that makes \( KB \) true also makes \( \alpha_1 \) true.
If KB is true in the real world, then any sentence $\alpha$ *entailed* by KB and any sentence $\alpha$ *derived* from KB by a sound inference procedure is also true in the real world.
Schematic Example: Follows, Entails, and Derives

**Inference**

“Mary is Sue’s sister and Amy is Sue’s daughter.”

“An aunt is a sister of a parent.”

---

**Derives**

Is it provable?

“Mary is Amy’s aunt.”

---

**Entails**

Is it true?

“Mary is Amy’s aunt.”

---

**Follows**

Is it the case?

“Mary is Amy’s aunt.”

---

**World**

Mary → Sister → Sue

Daughter → Amy

Mary → Aunt → Amy
Recap propositional logic: **Validity and satisfiability**

A sentence is **valid** if it is true in **all** models, e.g., True, A ∨ ¬A, A ⇒ A, (A ∧ (A ⇒ B)) ⇒ B

Validity is connected to inference via the **Deduction Theorem**:

KB ⊨ α if and only if (KB ⇒ α) is valid

A sentence is **satisfiable** if it is true in **some** model e.g., A ∨ B, C

A sentence is **unsatisfiable** if it is false in **all** models e.g., A ∧ ¬A

Satisfiability is connected to inference via the following:

KB ⊨ A if and only if (KB ∧ ¬A) is unsatisfiable (there is no model for which KB is true and A is false)
Inference Procedures

• $KB \models_i A$ means that sentence $A$ can be derived from $KB$ by procedure $i$

• **Soundness:** $i$ is sound if whenever $KB \models_i \alpha$, it is also true that $KB \models \alpha$
  – *(no wrong inferences, but maybe not all inferences)*

• **Completeness:** $i$ is complete if whenever $KB \models \alpha$, it is also true that $KB \models_i \alpha$
  – *(all inferences can be made, but maybe some wrong extra ones as well)*

• Entailment can be used for inference (Model checking)
  – enumerate all possible models and check whether $\alpha$ is true.
  – For $n$ symbols, time complexity is $O(2^n)$...

• Inference can be done directly on the sentences
  – Forward chaining, backward chaining, resolution (see FOPC, later)
Propositional Logic --- Summary

• Logical agents apply inference to a knowledge base to derive new information and make decisions.

• Basic concepts of logic:
  – syntax: formal structure of sentences
  – semantics: truth of sentences wrt models
  – entailment: necessary truth of one sentence given another
  – inference: deriving sentences from other sentences
  – soundness: derivations produce only entailed sentences
  – completeness: derivations can produce all entailed sentences
  – valid: sentence is true in every model (a tautology)

• Logical equivalences allow syntactic manipulations.

• Propositional logic lacks expressive power:
  – Can only state specific facts about the world.
  – Cannot express general rules about the world
    (use First Order Predicate Logic instead)
Outline

- Propositional Logic
- Knowledge Representation using First-Order Logic
- Inference in First-Order Logic
- Probability
- 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  ¬, ⇒, ∧, ∨, ⇔
- 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**:
  - $\equiv$ 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.”
  - $\neg \forall x \ P(x) \equiv \exists x \neg P(x)$
  - $\neg \exists x \ P(x) \equiv \forall x \neg P(x)$
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.
\[ \forall x \exists y \text{ Loves}(x, y) \]
- 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: \[ \exists y ( \forall x \text{ Loves}(x, y) ) \]

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

De Morgan’s Rule

\[ P \land Q \equiv \neg(\neg P \lor \neg Q) \]
\[ P \lor Q \equiv \neg(\neg P \land \neg Q) \]
\[ \neg(P \land Q) \equiv \neg P \lor \neg Q \]
\[ \neg(P \lor Q) \equiv \neg P \land \neg Q \]

Generalized De Morgan’s Rule

\[ \forall x \ P \equiv \exists x \ (\neg P) \]
\[ \exists x \ P \equiv \forall x \ (\neg P) \]
\[ \neg \forall x \ P \equiv \exists x \ (\neg P) \]
\[ \neg \exists x \ P \equiv \forall x \ (\neg P) \]

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

- Propositional Logic
- Knowledge Representation using First-Order Logic
- Inference in First-Order Logic
- Probability
- 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: $\text{Subst}(\theta, p) = \text{result of substituting } \theta \text{ into sentence } p$

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

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

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

  $\theta = \{y/\text{John}, x/z\}$ 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
Hard matching example

$$\text{Diff}(wa,nt) \land \text{Diff}(wa,sa) \land \text{Diff}(nt,q) \land \text{Diff}(nt,sa) \land \text{Diff}(q,nsw) \land \text{Diff}(q,sa) \land \text{Diff}(nsw,v) \land \text{Diff}(nsw,sa) \land \text{Diff}(v,sa) \Rightarrow \text{Colorable}()$$

$$\text{Diff}(\text{Red}, \text{Blue}) \land \text{Diff} (\text{Red}, \text{Green}) \land \text{Diff} (\text{Green}, \text{Red}) \land \text{Diff} (\text{Green}, \text{Blue}) \land \text{Diff} (\text{Blue}, \text{Red}) \land \text{Diff} (\text{Blue}, \text{Green})$$

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

\[ p_1', p_2', \ldots, p_n', (p_1 \land p_2 \land \ldots \land p_n \Rightarrow q) \]

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 \(Greedy(x)\)
\(\theta\) is \(\{x/John, y/John\}\) \(q\) is \(Evil(x)\)
Subst(\(\theta, q\)) 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

• 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 \( k-1 \)
  \[ \Rightarrow \] 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
  – ⇒ fix by checking current goal against every goal on stack

• Inefficient due to repeated subgoals (both success and failure)
  – ⇒ 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:

\[ \ell_1 \lor \cdots \lor \ell_k, \quad m_1 \lor \cdots \lor m_n \]

\[ \text{Subst}(\theta, \ell_1 \lor \cdots \lor \ell_{i-1} \lor \ell_{i+1} \lor \cdots \lor \ell_k \lor m_1 \lor \cdots \lor m_{j-1} \lor m_{j+1} \lor \cdots \lor m_n) \]

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,

\[ \neg \text{Rich}(x) \lor \text{Unhappy}(x), \quad \text{Rich}(\text{Ken}) \]

\[ \text{Unhappy}(\text{Ken}) \]

with \( \theta = \{x/\text{Ken}\} \)

- Apply resolution steps to \( \text{CNF}(\text{KB} \land \neg \alpha) \); complete for FOL
Resolution proof

\neg American(x) \lor \neg Weapon(y) \lor \neg Sells(x,y,z) \lor \neg Hostile(z) \lor Criminal(x)

\neg Criminal(West)

American(West)

\neg American(West) \lor \neg Weapon(y) \lor \neg Sells(West,y,z) \lor \neg Hostile(z)

\neg Weapon(y) \lor \neg Sells(West,y,z) \lor \neg Hostile(z)

Missile(M1)

\neg Missile(y) \lor \neg Sells(West,y,z) \lor \neg Hostile(z)

\neg Missile(x) \lor \neg Owns(Nono,x) \lor Sells(West,x,Nono)

\neg Sells(West,M1,z) \lor \neg Hostile(z)

Missile(M1)

\neg Missile(M1) \lor \neg Owns(Nono,M1) \lor \neg Hostile(Nono)

Owns(Nono,M1)

\neg Owns(Nono,M1) \lor \neg Hostile(Nono)

\neg Enemy(x,America) \lor Hostile(x)

\neg Hostile(Nono)

Enemy(Nono,America)

\neg Enemy(Nono,America)
Converting FOL sentences to CNF

Original sentence:
Everyone who loves all animals is loved by someone:
\[ \forall x \left[ \forall y \ Animal(y) \Rightarrow Loves(x,y) \right] \Rightarrow \left[ \exists y \ Loves(y,x) \right] \]

1. Eliminate biconditionals and implications
\[
\forall x \left[ \neg \forall y \ \neg Animal(y) \lor Loves(x,y) \right] \lor \left[ \exists y \ Loves(y,x) \right]
\]

2. Move \( \neg \) inwards:
Recall:
\( \neg \forall x \ p \equiv \exists x \ \neg p \), \( \neg \exists x \ p \equiv \forall x \ \neg p \)
\[
\forall x \left[ \exists y \ \neg (\neg Animal(y) \lor Loves(x,y)) \right] \lor \left[ \exists y \ Loves(y,x) \right]
\]
\[
\forall x \left[ \exists y \ \neg \neg Animal(y) \land \neg Loves(x,y) \right] \lor \left[ \exists y \ Loves(y,x) \right]
\]
\[
\forall x \left[ \exists y \ Animal(y) \land \neg Loves(x,y) \right] \lor \left[ \exists y \ Loves(y,x) \right]
\]
\[
\forall x \left[ \exists y \ Animal(y) \land \neg Loves(x,y) \right] \lor \left[ \exists y \ Loves(y,x) \right]
\]
Conversion to CNF contd.

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

   \[ \forall x \left[ \exists y \ Animal(y) \land \neg Loves(x,y) \right] \lor \left[ \exists z \ Loves(z,x) \right] \]

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

   \[ \forall x \left[ \text{Animal}(F(x)) \land \neg Loves(x,F(x)) \right] \lor Loves(G(x),x) \]

   (reason: animal y could be a different animal for each x.)
Conversion to CNF contd.

5. Drop universal quantifiers:

\[\text{Animal}(F(x)) \land \neg \text{Loves}(x,F(x))] \lor \text{Loves}(G(x),x)\]

(all remaining variables assumed to be universally quantified)

6. Distribute \(\lor\) over \(\land\):

\[[\text{Animal}(F(x)) \lor \text{Loves}(G(x),x)] \land [\neg \text{Loves}(x,F(x)) \lor \text{Loves}(G(x),x)]\]

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

- Propositional Logic
- Knowledge Representation using First-Order Logic
- Inference in First-Order Logic
- Probability
- 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.

• Boolean random variables
e.g., Cavity (= do I have a cavity?)

• Discrete random 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 OR B) = P(A) + P(B) - P(A 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

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

• \( P(a \mid b) \) obeys the same rules as probabilities,
  - E.g., \( P(a \mid b) + P(\neg a \mid b) = 1 \)
  - All probabilities in effect are conditional probabilities
    • E.g., \( P(a) = P(a \mid \text{our background knowledge}) \)
Random Variables

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

- Mutual exclusion
  \[ P(A = a_i \text{ AND } A = a_j) = 0 \]

- Exhaustive
  \[ \sum P(a_i) = 1 \]

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 $P(A = a \text{ AND } B=b)$
  – $\sum_a \sum_b P(a, b) = 1$
  – 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
  – $\sum_a \sum_b... \sum_z P(a, b, ..., z) = 1$
  – $P(A, B, ..., Z)$ is a table of $m^K$ numbers, $K = \# \text{ variables}$
    • 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:
  \[ P(a, b) = P(a \mid b) \ P(b) \]
  \[ = P(b \mid a) \ P(a) \quad \text{by definition} \]

  => \[ P(b \mid a) = \frac{P(a \mid b) \ P(b)}{P(a)} \quad \text{[Bayes rule]} \]

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)

\[ P(a) = \sum_b P(a, b) = \sum_b P(a \mid b) P(b) \quad \text{where } B \text{ is any random variable} \]

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.,

\[ P(b) = \sum_a \sum_c \sum_d P(a, b, c, d) \]

We can compute any conditional probability given a joint distribution, e.g.,

\[ P(c \mid b) = \sum_a \sum_d P(a, c, d \mid b) = \sum_a \sum_d P(a, c, d, b) / P(b) \quad \text{where } P(b) \text{ can be computed as above} \]
Computing with Probabilities:  
The Chain Rule or Factoring

We can always write

\[ P(a, b, c, \ldots z) = P(a \mid b, c, \ldots z) \cdot P(b, c, \ldots z) \]

(by definition of joint probability)

Repeatedly applying this idea, we can write

\[ P(a, b, c, \ldots z) = P(a \mid b, c, \ldots z) \cdot P(b \mid c, \ldots z) \cdot P(c \mid \ldots z) \ldots P(z) \]

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
  \[ P(a, b) = P(a) \cdot P(b) \]  for all values \( a, b \)

• More intuitive (equivalent) conditional formulation
  – A and B are independent iff
    \[ P(a | b) = P(a) \]  OR  \[ P(b | a) \cdot P(b), \]  for all values \( a, b \)

  – Intuitive interpretation:
    \[ P(a | b) = P(a) \] 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

- Two random variables A and B are conditionally independent given C iff
  \[ P(a, b | c) = P(a | c) P(b | c) \]
  for all values a, b, c

- More intuitive (equivalent) conditional formulation
  - A and B are conditionally independent given C iff
    \[ P(a | b, c) = P(a | c) \quad \text{OR} \quad P(b | a, c) P(b | c), \]
    for all values a, b, c
  - Intuitive interpretation:
    \[ P(a | b, c) = P(a | c) \]
    tells us that learning about b, given that we already know c, provides no change in our probability for a,
    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
  - \[ P(X1, X2, ..., XK | C) = \prod P(Xi | C) \]
  - Also known as the naïve Bayes assumption
Outline

- Propositional Logic
- Knowledge Representation using First-Order Logic
- Inference in First-Order Logic
- Probability
- 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
"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:

\[(\neg Y \lor \neg R) \land (Y \lor R) \land (Y \lor M) \land (R \lor H) \land (\neg M \lor H) \land (\neg H \lor G)\]
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 = MDCCCLXXVI

• Long division is **very tedious** (try MDCCCLXXVI / 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
  
  ...010...011...000...

- Rectangular array of pixels = local inference for vision

```
0 0 0 1 0 0 0 0 0 0
0 0 0 1 0 0 0 0 0 0
0 0 0 1 0 0 0 0 0 0
0 0 0 1 0 0 0 0 0 0
0 0 0 1 0 0 0 0 0 0
0 0 0 1 0 0 0 0 0 0
0 0 0 1 1 1 1 1 1 1
0 0 0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0 0 0
```
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:
  - $h(x; \theta) = \text{sign}(w_1x_1 + w_2x_2 + w_3)$
  - $h_k(x) = (x_1 \text{ OR } x_2) \text{ AND } (x_3 \text{ OR NOT}(x_4))$
**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

<table>
<thead>
<tr>
<th></th>
<th></th>
<th>A xor B</th>
</tr>
</thead>
<tbody>
<tr>
<td>F</td>
<td>F</td>
<td>F</td>
</tr>
<tr>
<td>F</td>
<td>T</td>
<td>T</td>
</tr>
<tr>
<td>T</td>
<td>F</td>
<td>T</td>
</tr>
<tr>
<td>T</td>
<td>T</td>
<td>F</td>
</tr>
</tbody>
</table>

![Decision Tree Diagram](image)
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
        $examples_i ← \{\text{elements of examples with } best = v_i\}$
        $subtree ← DTL(examples_i, attributes - best, Mode(examples))$
        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

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

- \( \text{Gain}(A) = H(p) - H(p \mid A) \)

- 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
How Overfitting affects Prediction

Predictive Error

Underfitting

Overfitting

Error on Test Data

Error on Training Data

Ideal Range for Model Complexity

Model Complexity
Disjoint Validation Data Sets
Disjoint Validation Data Sets

Full Data Set

1st partition

2nd partition

Validation Data

Training Data

Validation Data
Classification in Euclidean Space

- A classifier is a partition of the space $x$ 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
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 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)

  - 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 $f(x) = f = \text{weighted sum of inputs}$
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
Gradient Descent Update Equation

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

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

- Gradient descent weight update rule:

$$
w_j = w_j + \eta \left( y(i) - \sigma[f(i)] \right) \frac{\partial \sigma[f(i)]}{\partial x_j(i)}
$$

- can rewrite as:

$$
w_j = w_j + \eta \times \text{error} \times c \times x_j(i)
$$
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
            $\text{deltaw}_j = \eta \left( y(i) - \sigma[f(i)] \right) \frac{\partial \sigma[f(i)]}{\partial x_j(i)} x_j(i)$
            $w_j = w_j + \text{deltaw}_j$
        end
    calculate termination condition
end

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

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

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 \( P(Y_i \mid C) \) 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.
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

- Propositional Logic
- Knowledge Representation using First-Order Logic
- Inference in First-Order Logic
- Probability
- Machine Learning

- Questions on any topic

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