Machine Learning Module 2

Learning Sets of Rules: Sequential Covering & Analytical Learning

Educational Slide Deck

Slide 1: The $50 Million Question

Why Learning Rules Matters

Think About This: - Bank processes 10 million transactions daily - Fraud costs $50M annually - False alarms annoy 100,000 legitimate customers - Regulators demand **explainable** decisions

The Challenge: You need a system that doesn't just predict fraud—it must **EXPLAIN WHY** a transaction is flagged.

Neural Networks: "This is fraud" (but can't explain why) **Rule-Based Systems:** "This is fraud BECAUSE amount > $10,000 AND location = overseas AND time = 3am"

Business Impact: - Customers deserve explanations for declined transactions - Regulators require transparent, auditable decisions - Doctors need to understand why AI recommends a treatment - Loan officers must justify credit decisions

Key Insight: Unlike neural networks (black boxes), rule-based systems provide human-readable explanations that stakeholders can understand, audit, and trust.

Slide 2: Module 2 Overview

What You'll Master

Part 1: Sequential Covering Algorithms - Learn rules one at a time (not all at once like decision trees) - Cover positive examples iteratively - Build disjunctive rule sets: Rule1 OR Rule2 OR Rule3

Part 2: Learning First-Order Rules (FOIL) - Rules with variables: Parent(x, y) instead of just attributes - Expressive representations for relational data - Can learn recursive definitions (Ancestor, family trees)

Part 3: Analytical Learning - Learning with prior knowledge (domain theories) - Explanation-Based Learning (EBL) - When you know the "rules of the game" already

Part 4: Inductive-Analytical Approaches - Combining data-driven + knowledge-driven learning - Search control learning (making AI faster) - Real applications: SOAR, PRODIGY systems

Learning Objectives: By the end of this module, you will be able to: 1. Implement sequential covering algorithms from scratch 2. Design and apply FOIL to learn first-order rules 3. Use prior knowledge to guide learning (EBL) 4. Choose the right rule learning approach for your problem 5. Explain when rules are better than decision trees or neural networks

---

Slide 3: How Humans Learn Rules

Learning Email Filtering Step by Step

Your Task: Identify spam emails (Learn like a human!)

Email 1: "FREE VIAGRA! Click now!!!" → **Spam**

Your Brain's Response: - Trial 1: "Hmm, all caps and 'FREE' seems suspicious…" - **Rule 1:** IF contains("FREE") THEN spam - Memory: Remember this pattern

Email 2: "Meeting at 3pm tomorrow" → **Not Spam**

Your Brain's Response: - Trial 2: "Wait, this is fine. My rule still works!" - Rule 1 still valid

Email 3: "You won $1,000,000!" → **Spam**

Your Brain's Response: - Trial 3: "Also spam, but different pattern… money amounts!" - **Refined Rule:** IF contains("FREE") OR (contains("won") AND contains("$")) THEN spam - Learning: Combining multiple patterns

Email 4: "Invoice for $500 attached" → **Not Spam**

Your Brain's Response: - Trial 4: "Oops! This has $ but it's legitimate…" - **Final Rule:** IF (contains("FREE") AND all_caps) OR (contains("won") AND contains("$") AND no_attachment) THEN spam - Memory: Refined understanding

Neural Network Parallel: - Your trials = Iterations/Epochs - Your errors = Loss function - Your adjustments = Gradient descent updates - Your memory = Learned weights - Your pattern recognition = Bias term

Key Insight: Humans naturally learn rules through trial, error, and refinement—exactly what ML algorithms do!

Slide 4: Sequential Covering: The Main Idea

Divide and Conquer Approach

Two Ways to Learn Rules:

Decision Trees (ID3) - Simultaneous Covering: - Learn entire tree at once - Each split affects all branches - All rules share decisions at top nodes

Rule Learning (CN2, FOIL) - Sequential Covering: - Learn rules ONE AT A TIME - Remove covered examples after each rule - Each rule is independent

The Sequential Covering Process (Like Peeling an Onion):

Step 1: Learn Rule 1 that covers many positive examples - Example: IF age > 65 THEN high_risk - Covers 30% of positive examples

Step 2: Remove all positive examples covered by Rule 1 - These examples are "explained" — we're done with them! - Focus on remaining 70%

Step 3: Learn Rule 2 on remaining examples - Example: IF smoker = yes AND BMI > 30 THEN high_risk - Covers another 25% of positives

Step 4: Repeat until all (or most) positive examples are covered - Rule 3, Rule 4, etc. - Stop when coverage threshold reached

Final Rule Set (Disjunction): IF age > 65 THEN high_risk OR IF smoker = yes AND BMI > 30 THEN high_risk OR IF family_history = yes AND cholesterol > 240 THEN high_risk

Why Sequential? - Makes n × k independent choices (more than decision trees) - Good when data is plentiful - Each rule can be very different from others - Easy to add/remove/modify individual rules

---

Slide 5: Sequential Covering Algorithm

The Outer Loop

Pseudocode:

```
SEQUENTIAL_COVERING(Target_attribute, Attributes, Examples, Threshold)

    Learned_rules ← {}

    WHILE examples remain:
        Rule ← LEARN_ONE_RULE(Target_attribute, Attributes, Examples)

        IF PERFORMANCE(Rule, Examples) > Threshold:
            Learned_rules ← Learned_rules + Rule
            Examples ← Examples - {correctly classified by Rule}
```

```
        ELSE:
            BREAK  // Stop if rule quality drops

    Sort Learned_rules by accuracy (best first)

    RETURN Learned_rules
```

Key Characteristics:

Advantages: - Each rule is independent (easy to understand) - Easy to debug individual rules - Handles disjunctive concepts naturally - Rules can be sorted by accuracy/priority - Can add domain knowledge easily

Considerations: - Greedy search (no backtracking) - Order of learning matters - May not find smallest rule set - Quality depends on LEARN_ONE_RULE

Real-World Application - Medical Diagnosis: Each rule represents a different diagnostic pathway: - Rule 1: IF fever > 102°F AND cough THEN influenza - Rule 2: IF rash AND fever THEN measles
- Rule 3: IF chest_pain AND shortness_of_breath THEN cardiac_issue

Business Value: Doctors can review and validate each rule independently!

Slide 6: LEARN_ONE_RULE: The Inner Loop

Finding One Good Rule

Goal: Find ONE rule with high accuracy (but can have low coverage)

Two Main Search Strategies:

1. General-to-Specific Search (CN2, FOIL)

```
Start with most general rule:
    IF TRUE THEN class
    (covers everything!)

Add constraints iteratively:
    IF age > 30 THEN class
    IF age > 30 AND income > 50K THEN class
    IF age > 30 AND income > 50K AND credit_score > 700 THEN class

Stop when:
    - Rule is accurate enough (few negative examples covered)
    - Or no more refinements improve accuracy
```

2. Specific-to-General Search (AQ Algorithm)

```
Start with one positive example:
    IF age=35 AND income=60K AND city=NYC THEN class
    (very specific, covers only this example)

Remove constraints iteratively:
    IF age=35 AND income>50K AND city=NYC THEN class
    IF age>30 AND income>50K THEN class

Stop before:
    - Covering negative examples
```

Search Enhancement: Beam Search - Keep top-k candidate rules at each step (beam width = k) - Evaluate using metrics: information gain, accuracy, m-estimate - Expand most promising candidates - Prevents getting stuck in local optima

Analogy: - **General-to-specific:** Zooming in from world map to street address - **Specific-to-general:** Zooming out from street address to world map

Key Insight: General-to-specific is most common because it's easier to add constraints than remove them!

Slide 7: Moving to First-Order Rules

Why We Need Variables

The Limitation of Propositional Rules:

Propositional (Attribute-Value):

```
IF age > 65 AND smoker = yes THEN high_risk
```

- Limited to talking about attributes of single object
- Cannot express relationships between objects
- Need separate rules for each person

First-Order Rules (With Variables):

Example 1: Family Relationships

```
IF Parent(x, y) AND Parent(y, z) THEN Grandparent(x, z)
```

- Works for ANY triple of people!
- One rule replaces infinite propositional rules

Example 2: Social Networks

Propositional Approach (Impossible!):

```
IF Alice.friend = Bob AND Bob.friend = Carol THEN recommend(Alice, Carol)
IF Bob.friend = Dave AND Dave.friend = Emma THEN recommend(Bob, Emma)
```

```
... (need rule for every possible triple!)
```

First-Order Approach (Elegant!):

```
IF Friend(x, y) AND Friend(y, z) AND NOT Friend(x, z)
THEN Recommend(x, z)
```

- ONE rule for everyone!
- Captures the "friend of friend" pattern

Why First-Order Rules Are Powerful:

1. **Expressiveness:** Can model complex relational structures
2. **Recursion:** Rules can reference themselves
   - `Ancestor(x,z) ← Parent(x,z)`
   - `Ancestor(x,z) ← Parent(x,y) AND Ancestor(y,z)`
3. **Generalization:** One rule → infinite instances
4. **Programming:** First-order rules = PROLOG programs!

Real-World Applications: - Database queries (SQL-like reasoning) - Knowledge graphs (Google, Facebook) - Chemical structure analysis - Program synthesis

---

Slide 8: FOIL Algorithm Overview

First-Order Inductive Learner

FOIL = Sequential Covering + Variables + Specialized Search

Algorithm Structure:

```
FOIL(Target_predicate, Predicates, Examples)

    Positives ← examples where Target_predicate is True
    Negatives ← examples where Target_predicate is False
    Learned_rules ← {}

    WHILE Positives not empty:

        // Learn one rule (general-to-specific)
        New_rule ← Target_predicate(vars) ←
        New_rule_negatives ← Negatives

        WHILE New_rule_negatives not empty:
            // Add literals to specialize
            Candidate_literals ← generate candidates
            Best_literal ← argmax FoilGain(literal)
            Add Best_literal to New_rule
            New_rule_negatives ← negatives still covered
```

```
        Learned_rules ← Learned_rules + New_rule
        Positives ← Positives - {covered by New_rule}

    RETURN Learned_rules
```

What Makes FOIL Special?

Feature	FOIL Approach	Why It Matters
Variables	Can introduce new variables	Express relationships between objects
Gain Metric	FoilGain considers variable bindings	Handles multiple instances per example
Recursion	Can reference target predicate in body	Learn recursive definitions
Negation	Allows negated literals	More expressive than Horn clauses

Key Differences from Propositional Sequential Covering:

1. **Variable Introduction:** Each literal can add new variables
2. **Binding Evaluation:** Must consider all possible variable bindings
3. **FoilGain Metric:** Different from information gain (accounts for bindings)
4. **Search Space:** Much larger due to variables

Example Target Concepts: - `GrandDaughter(x, y)` — x is granddaughter of y - `Ancestor(x, y)` — x is ancestor of y (recursive!) - `Uncle(x, y)` — x is uncle of y

---

Slide 9: How FOIL Generates Candidate Specializations

Building First-Order Rules

Current Rule Being Learned:

`P(x , x , ..., x ) ← L   L   ...   L`

Three Types of Literals FOIL Can Add:

Type 1: New Predicate with Variables

`Q(v , v , ..., v )`

- Q is any predicate from available predicates
- At least ONE variable must already exist in rule
- Can introduce NEW variables (extends search)

- Example: `Father(y, z)` where y exists, z is new

Type 2: Equality Test

`Equal(x , x )`

- Both variables must already exist
- Tests if two objects are the same
- Example: `Equal(x, z)` — checks if x and z refer to same person

Type 3: Negations

`¬Q(...) or ¬Equal(...)`

- Negation of Types 1 or 2
- Example: `¬Friend(x, z)` — x and z are NOT friends

Candidate Generation Example:

Learning: `GrandDaughter(x, y) ← ...` Available predicates: `Father(x,y)`, `Female(x)`

Iteration 1: Current rule = `GrandDaughter(x, y) ←`

Candidates generated: - `Female(x)`, `Female(y)` [Type 1, no new vars] - `Father(x, y)`, `Father(y, x)` [Type 1, no new vars] - `Father(x, z)`, `Father(z, x)`, `Father(y, z)`, `Father(z, y)` [Type 1, NEW var z] - `Equal(x, y)` [Type 2] - `¬Female(x)`, `¬Father(x, y)`, … [Type 3]

Iteration 2: If we selected `Father(y, z)`, now rule = `GrandDaughter(x, y) ← Father(y, z)`

New candidates include: - All previous candidates - `Female(z)` [using the new variable z] - `Father(z, x)`, `Father(z, w)` [new variable w] - `Equal(z, x)`, `Equal(z, y)` [equality with z]

Key Constraint: At least one variable in new literal must already exist (prevents disconnected rules)

Slide 10: FoilGain: Choosing the Best Literal

How FOIL Evaluates Candidates

The FoilGain Formula:

```
FoilGain(L, Rule) = t × (log (p /(p +n )) - log (p /(p +n )))

Where:
    t  = positive bindings still covered after adding L
    p  = positive bindings before adding L
    n  = negative bindings before adding L
    p  = positive bindings after adding L
```

```
n = negative bindings after adding L
```

Why "Bindings" Not "Examples"?

In first-order rules, one example can create multiple bindings!

Example: Training example: `Father(Tom, Bob), Father(Bob, Alice)`

For rule `GrandDaughter(x, y) ← Father(y, z)`: - Binding 1: x=Alice, y=Bob, z=Tom - Binding 2: x=Alice, y=Tom, z=Bob - Each binding is evaluated separately!

What FoilGain Measures: - Prefers literals that **keep many positive bindings** - While **eliminating many negative bindings** - The t factor gives bonus for covering more positives

Intuition:

```
High FoilGain = Many positives stay AND Many negatives eliminated
Low FoilGain  = Few positives stay OR Few negatives eliminated
```

Complete Example: Learning GrandDaughter(x, y)

Training Data:

```
Positives: GrandDaughter(Tom, Alice)
Facts: Father(Tom, Bob), Father(Bob, Alice), Female(Alice)
```

Initial Rule: `GrandDaughter(x, y) ←` - p = all positive bindings - n = all negative bindings (everything else)

Candidate: Add `Father(y, z)` - p = positives where y is someone's father (still many) - n = negatives eliminated (fewer than n) - Calculate FoilGain — if highest, select this literal!

Slide 11: FOIL Example - Learning GrandDaughter(x, y)

Complete Step-by-Step Trace

Available Predicates: `Father(x, y)`, `Female(x)`

Training Data:

```
Facts:
    Father(Tom, Bob)
    Father(Bob, Alice)
    Father(Tom, Carol)
    Female(Alice)
    Female(Carol)

Positive Examples:
    GrandDaughter(Tom, Alice)
```

```
Negative Examples:
    GrandDaughter(Tom, Bob)
    GrandDaughter(Tom, Carol)
    GrandDaughter(Bob, Tom)
    ... etc
```

Learning Process:

Iteration 1: Start Most General

`GrandDaughter(x, y) ←`

- Covers EVERYTHING (all positives AND all negatives!)
- Need to specialize

Iteration 2: Add First Literal

Candidates evaluated: - `Female(x)` → some gain - `Female(y)` → HIGH gain (y must be female!) - `Father(x, y)` → some gain - `Father(y, z)` → **HIGHEST FoilGain**

Selected: `Father(y, z)`

`GrandDaughter(x, y) ← Father(y, z)`

- Introduced new variable z
- Still covers some negatives (Tom-Bob, Tom-Carol covered because they satisfy Father relationship)

Iteration 3: Continue Specializing

Current rule covers negatives, so continue…

Candidates now include literals with variables x, y, z: - `Female(y)` → high gain
- `Father(z, x)` → **HIGHEST FoilGain** - `Equal(z, x)` → some gain

Selected: `Father(z, x)`

`GrandDaughter(x, y) ← Father(y, z)   Father(z, x)`

- Creates the grandparent chain: y's father is z, z's father is x
- Still may cover some negatives

Iteration 4: Final Refinement

Candidates: - `Female(y)` → **HIGHEST FoilGain** - Other literals don't help much

Selected: `Female(y)`

`GrandDaughter(x, y) ← Father(y, z)   Father(z, x)   Female(y)`

Final Rule Interpretation: "x is granddaughter of y IF: - y's father is z, AND - z's father is x, AND
- y is female"

Covers all positives, zero negatives → Done!

Key Insight: FOIL built a chain of relationships connecting x and y through intermediate variable z!

Slide 12: Quick Check - Test Your Understanding

Verify Your Grasp of Key Concepts

Question 1: Conceptual Understanding

What is the main difference between sequential covering (CN2, FOIL) and simultaneous covering (ID3)?

a) Sequential learns rules one at a time and removes covered examples

b) Sequential is always more accurate

c) Sequential can't handle noisy data

d) Sequential only works with propositional logic

Answer: (a) Sequential covering learns one rule at a time, removes covered positive examples, then learns the next rule. Decision trees (simultaneous) learn all rules together as part of one tree structure.

Question 2: FOIL Mechanics

When FOIL adds a new literal to a rule, what constraint must be satisfied?

a) All variables must be new

b) At least one variable must already exist in the rule

c) The literal must be negated

d) The literal must use only the target predicate

Answer: (b) At least one variable in the new literal must already appear in the rule. This ensures the rule remains connected. New variables can be introduced, but not in isolation.

Question 3: Application Challenge

Given predicates `Parent(x,y)` and `Male(x)`, you want to learn `Uncle(x, y)` (x is uncle of y).

Starting from: `Uncle(x, y) ←`

What would be the FIRST literal FOIL likely adds?

a) `Male(x)`

b) `Parent(x, y)`

c) `Parent(z, y)` where z is new

d) `Equal(x, y)`

Answer: (c) Need to connect x to y through someone else (z). An uncle is your parent's sibling, so first step is finding y's parent. Then we'll connect x to that parent as a sibling.

Full solution would be something like:

`Uncle(x, y) ← Parent(z, y)  Parent(w, x)  Parent(w, z)  Male(x)  ¬Equal(x, z)`

(x and z share parent w, making them siblings, and x is male)

Slide 13: Analytical Learning - A Different Paradigm

Learning with Prior Knowledge

Two Fundamentally Different Approaches:

Inductive Learning (What We've Done So Far):

```
Given:
    • Training examples (data)
    • Hypothesis space

Find:
    • Hypothesis that fits the data

Problem:
    • Many hypotheses might fit!
    • Need lots of data
    • Pure pattern matching
```

Analytical Learning (Using Knowledge!):

```
Given:
    • Training examples (data)
    • Hypothesis space
    • Domain theory (prior knowledge!)

Find:
```

- `Hypothesis consistent with BOTH data AND theory`

`Advantage:`

- `Less ambiguity`
- `Learn from fewer examples`
- `Justified decisions`

Real-World Analogy: Learning Chess

Pure Inductive Learning: - Watch millions of chess games - Try to figure out patterns from scratch - "Hmm, this L-shaped piece moves in weird ways…" - "People often move pawns early…" - Takes forever, unclear why moves are good

Analytical Learning (With Domain Theory): - Domain theory: Legal moves, piece values, checkmate rules - Now just learn: "Which legal moves are GOOD in different positions?" - Much faster! Can explain based on principles - "This move controls the center" (theory-justified)

Key Insight: Prior knowledge reduces hypothesis space and guides generalization, allowing accurate learning from fewer examples!

Slide 14: Perfect Domain Theories

When We Know the Rules

Definition: Perfect Domain Theory

A domain theory is **perfect** if it is:

1. **Correct:** Every assertion in the theory is TRUE about the world
2. **Complete:** Every positive example can be proven using the theory

Examples of Perfect Domain Theories:

Domain	Perfect Theory	What We Learn
Chess	Legal move rules	Good strategy, position evaluation
Physics	Newton's laws F=ma	How to solve problems quickly
Logic Circuits	Gate behaviors (AND, OR, NOT)	Circuit optimization patterns
Planning	Action preconditions & effects	Which actions to try first
Arithmetic	Addition, multiplication rules	Mental math shortcuts

The Paradox: If We Know Everything, Why Learn?

The Answer: Difference between what we "know in principle" vs. what we can "compute efficiently"

Example: Newton's Laws - We know F = ma perfectly - But solving complex physics problems from first principles takes forever! - **Learning:** Transform deep knowledge into operational shortcuts

Example: Chess - We know all legal moves perfectly - But evaluating all possible games is impossible! - **Learning:** Recognize patterns like "control the center" without deep search

What EBL Does:

```
Deep, Principled Knowledge (slow to apply)
           ↓
        Learning
           ↓
Shallow, Operational Rules (fast to apply)
```

Like memorizing "9 × 7 = 63" instead of adding 9 seven times every time!

Key Insight: Perfect theories exist when we know the rules but need to learn efficient application!

Slide 15: Explanation-Based Learning (EBL)

The Three-Step Process

The Core Idea: Explain each example using domain theory, then generalize the explanation.

The EBL Process:

Step 1: EXPLAIN - Prove why the training example satisfies the target concept - Use domain theory to build logical derivation - Create a "proof tree" showing reasoning

Step 2: ANALYZE
- Determine general conditions under which explanation holds - Find the "weakest preimage" — most general conditions - Abstract away specific details

Step 3: REFINE - Add new rule to hypothesis capturing these conditions - Create operational rule that skips future explanations - Cache the pattern for reuse

Two Perspectives on EBL:

1. Theory-Guided Generalization of Examples - Uses domain theory to distinguish relevant from irrelevant features - Rational generalization (not just statistical) - Avoids sample complexity issues of pure induction

2. Example-Guided Reformulation of Theory - Reformulates domain theory into operational form - Creates special-case rules for common scenarios - One-step inference instead of deep reasoning

What EBL Really Does:

```
Before EBL:
    Problem → Deep reasoning from first principles → Answer
    (SLOW but correct)

After EBL:
    Problem → Match cached pattern → Answer
    (FAST and still correct!)
```

Example: Expert Physics Student - Before: Derives every problem from F=ma - After: Recognizes "pulley problem" and applies template - Same correctness, 10x faster!

Key Insight: EBL doesn't discover new knowledge—it reformulates existing knowledge into more usable form!

Slide 16: PROLOG-EBG Algorithm

Explanation-Based Learning with Horn Clauses

PROLOG-EBG: Representative EBL algorithm using first-order logic

Algorithm:

```
PROLOG_EBG(Target_concept, Training_examples, Domain_theory)

    Learned_rules ← {}
    Positives ← positive examples from Training_examples

    FOR EACH positive example NOT yet covered:

        // STEP 1: EXPLAIN
        Explanation ← prove(example, Domain_theory)

        // STEP 2: ANALYZE
        General_rule ← extract_weakest_preimage(Explanation)

        // STEP 3: REFINE
        Learned_rules ← Learned_rules + General_rule
        Mark examples covered by General_rule as done

    RETURN Learned_rules
```

Key Concepts:

Explanation (Proof): - Logical derivation showing how example satisfies target - In PROLOG: A proof tree - Uses only rules from domain theory

Weakest Preimage: - Most general conditions under which explanation holds - Replaces specific values with variables where possible - Result: Rule that covers example and all similar cases

Properties of PROLOG-EBG:

1. **Deductive:** Learned rules follow logically from domain theory
2. **Sequential Covering:** Learns one rule at a time (like FOIL)
3. **Feature Construction:** Creates useful intermediate features automatically
4. **Correctness Guarantee:** If domain theory is correct, hypothesis is correct
5. **Sample Efficient:** Can learn from very few examples

Difference from Inductive Methods: - FOIL: Searches data for patterns → might be wrong - PROLOG-EBG: Derives rules from theory → guaranteed correct (if theory correct)

Slide 17: EBL Example - SafeToStack(x, y)

Complete Walkthrough

Problem: Learn when it's safe to stack object x on object y

Training Example (Positive):

```
SafeToStack(Obj1, Obj2)

Obj1: Material=Plastic, Density=0.5, Volume=10
Obj2: Material=Wood, Density=0.6, Volume=20
```

Domain Theory (Horn Clauses):

```
Rule 1: SafeToStack(x, y) ← Lighter(x, y)
Rule 2: SafeToStack(x, y) ← ¬Fragile(y)
Rule 3: Lighter(x, y) ← Weight(x, wx)   Weight(y, wy)   LessThan(wx, wy)
Rule 4: Weight(x, w) ← Volume(x, v)   Density(x, d)   Equal(w, v×d)
Rule 5: Fragile(x) ← Material(x, Glass)
```

STEP 1: EXPLAIN (Build Proof Tree)

```
SafeToStack(Obj1, Obj2)
    ↓ [Apply Rule 1]
Lighter(Obj1, Obj2)
    ↓ [Apply Rule 3]
```

```
Weight(Obj1, w1)  Weight(Obj2, w2)  LessThan(w1, w2)
    ↓ [Apply Rule 4 twice]
Volume(Obj1, 10)  Density(Obj1, 0.5)  Equal(w1, 10×0.5)
  Volume(Obj2, 20)  Density(Obj2, 0.6)  Equal(w2, 20×0.6)
  LessThan(5, 12)
    ↓ [Evaluate]
TRUE
```

STEP 2: ANALYZE (Extract General Conditions)

Look at proof tree, replace specific values with variables: - Obj1 → x (any object) - Obj2 → y (any object) - 10, 0.5, 5 → vx, dx, (vx×dx) - 20, 0.6, 12 → vy, dy, (vy×dy)

Weakest Preimage: "The explanation holds for ANY x and y where x's weight < y's weight"

STEP 3: REFINE (Create Operational Rule)

```
Learned Rule:
SafeToStack(x, y) ←
    Volume(x, vx)
    Density(x, dx)
    Volume(y, vy)
    Density(y, dy)
    LessThan(vx×dx, vy×dy)
```

What We Achieved: - Original domain theory: 3-step reasoning (Lighter → Weight → calculation) - Learned rule: 1-step direct check! - Works for ANY objects with lighter-than relationship - Bypasses intermediate concepts (Lighter, Weight)

Key Insight: EBL flattened multi-step reasoning into single operational rule!

Slide 18: Inductive vs Analytical Learning

Comparing the Two Paradigms

Aspect	Inductive Learning	Analytical Learning
Input	Training data + Hypothesis space	Training data + Hypothesis space + **Domain theory**
Learning Style	Pattern matching from data	Explanation + generalization
Sample Complexity	Needs many examples	Can learn from few examples

Aspect	Inductive Learning	Analytical Learning
Guarantee	Statistical (probably correct)	Deductive (logically correct if theory correct)
Hypothesis Space	Searches entire space	Constrained by theory
Explainability	Often black box	Fully explainable
New Knowledge	Can discover truly novel patterns	Reformulates existing knowledge
Robustness	Handles noisy data well	Sensitive to theory errors
When Theory Wrong	Still learns from data	May learn incorrect rules

When to Use Each?

Use Inductive Learning When: - Lots of training data available - No good domain theory exists - Discovering novel patterns - Data is noisy or messy - Want to find surprising insights

Examples: Image recognition, spam detection, recommendation systems

Use Analytical Learning When: - Limited training data - Strong prior knowledge exists - Need explainable decisions - Rules of domain are well-known - Correctness is critical

Examples: Chess, physics problems, logic puzzles, planning

The Best Approach: Combine Both!

```
Inductive-Analytical Hybrid:
1. Start with domain theory (analytical)
2. Use data to refine/correct theory (inductive)
3. Use theory to guide search (analytical)
4. Handle exceptions with data (inductive)
```

Key Insight: Pure induction is too data-hungry. Pure analysis is too rigid. Combination is best!

Slide 19: Real-World Applications

Where These Techniques Excel

1. Search Control Learning (SOAR & PRODIGY)

Problem: Planning and search problems have huge state spaces - Game playing: 10^120 possible chess positions - Route planning: Millions of possible paths - Theorem proving: Infinite proof attempts

Domain Theory: Legal operators, goal conditions (perfect!)

What EBL Learns: Which operators to try first

Results: - SOAR: 10-100x speedup on puzzle solving through "chunking" - PRODIGY: Learns to prioritize promising search branches - Caches successful solution patterns

Example - 8-Puzzle: - Domain theory: Legal moves (up, down, left, right) - EBL learns: "When blank in corner, move toward center first" - Speedup: 10x fewer states explored

---

2. Robot Planning & Control

Traditional Approach	With EBL
Plan from scratch each time	Learn from successful plans
Expensive search	Recognize similar situations
Slow reaction times	Fast, reactive behavior

Application: Autonomous navigation - Domain theory: Physics (friction, momentum), map structure - Learn: "In narrow corridor, slow down before turn" - Result: Smooth, efficient movement

---

3. Game Playing

Chess Example: - Domain theory: Legal moves (perfect!) - Training: Analyze master games - Learn: "In king-side castled position, advance h-pawn for attack" - Result: Fast move generation without deep search

Go Example: - Domain theory: Rules, basic patterns - Learn: Opening sequences, joseki (corner patterns) - Result: Human-level play with less computation

---

4. Chemical Structure Analysis (Real Research!)

Applications: - FOIL learned rules for mass spectrometer fragmentation - Predicted which chemical bonds break - Learned mutagenic activity patterns (related to cancer risk)

Success Story: - Srinivasan et al. (1994) used FOIL on chemical structures - Learned rules predicting mutagenicity - Combined structural knowledge with experimental data - Accuracy: 90%+ on new compounds

---

5. Medical Diagnosis

Domain Theory: Medical knowledge (symptoms → diseases)

EBL Application: - Explain diagnosis for training cases - Extract patterns: "IF symptoms X, Y, Z THEN likely disease D" - Create fast diagnostic rules

Benefit: - Explainable (doctors can verify) - Faster than reasoning from first principles - Based on established medical knowledge

Key Insight: These aren't toy problems—real systems use these techniques to solve complex real-world challenges!

Slide 20: Key Takeaways & Hands-On Assignment

Summary & Next Steps

Key Takeaways:

1. Sequential Covering: - Learn rules one at a time, remove covered examples - Alternative to decision trees (simultaneous covering) - Each rule is independent and interpretable

2. FOIL Algorithm: - Extends sequential covering to first-order logic - Handles variables and relations elegantly - Can learn recursive rules (Ancestor, family trees) - Uses FoilGain to evaluate candidates

3. Analytical Learning: - Uses prior knowledge (domain theory) - Explains examples, then generalizes explanations - Sample-efficient (learns from few examples) - Produces justified, explainable rules

4. EBL Process: - EXPLAIN: Prove example using domain theory - ANALYZE: Extract general conditions - REFINE: Create operational rule

5. Key Tradeoffs: - Inductive: Data-hungry but discovers novel patterns - Analytical: Sample-efficient but needs correct theory - **Best: Combine both approaches!**

Hands-On Assignment (Due: 2 Weeks)

Objective: Implement and test rule learning algorithms

Part 1: Implement Mini-FOIL (50 points) 1. Implement sequential covering outer loop 2. Implement learn-one-rule with general-to-specific search 3. Generate candidate literals (at least Type 1) 4. Use information gain or FoilGain for selection

Part 2: Testing (30 points) Test on family relationship dataset (provided): - Learn rules for: `Sibling(x, y)`, `Uncle(x, y)`, `Cousin(x, y)` - Report accuracy on training and test sets - Show learned rules in readable format

Part 3: Analysis (20 points) - Compare learned rules to human-written rules - Discuss what worked and what didn't - Explain any surprising rules learned

Optional Challenge (+15 bonus points): Implement PROLOG-EBG for SafeToStack domain: - Provide explanation tree visualization - Show weakest preimage extraction - Compare to FOIL's learned rules

Deliverables: - Python/Java code (well-commented) - README with setup instructions - Report (3-4 pages) with results and analysis - Test results CSV file

---

Next Steps:

Next Module Preview: Combining Inductive and Analytical Learning - Imperfect domain theories - Theory refinement - Knowledge-based neural networks - Hybrid learning systems

Recommended Reading: - Quinlan (1990): "Learning Logical Definitions from Relations" - Mitchell et al. (1986): "Explanation-Based Generalization: A Unifying View" - Textbook: Chapters 10-11

Office Hours: Monday 2-4pm, Wednesday 10am-12pm

Resources: Datasets, starter code, and tutorials at course website

---

End of Module 2 Slide Deck