Machine Learning Module 1

Introduction & Concept Learning

Educational Slide Deck

Slide 1: The $2 Million Question

Why Can't Computers Learn Like We Do?

Think About This:

- Netflix recommends shows you'll love—how does it know?
- Gmail blocks spam emails automatically—who taught it?
- Self-driving cars recognize pedestrians—but never took a driving lesson

The Business Impact:

- Companies lose $62 billion annually due to poor customer understanding
- Machine Learning reduces fraud detection costs by 40%
- Personalization engines increase revenue by 15-20%

The Challenge: Can we teach computers to improve from experience without explicitly programming every scenario?

Key Insight: The ability to learn from data is transforming industries—from healthcare to finance to entertainment.

Slide 2: Learning Like Humans Do

Solving Problems Through Experience

The Human Way - Problem: Solve 2x + 3 = 9

Trial 1: "Let me try x = 2"

- Result: 2(2) + 3 = 7
- Error: Off by 2
- Learning: "x needs to be bigger"

Trial 2: "Now try x = 3"

- Result: 2(3) + 3 = 9 ✓

- Success!
- Memory: "When I see similar patterns, I know what to try"

Neural Network Parallel:

- Trial = Iteration/Epoch
- Error = Loss Function
- Adjustment = Gradient Descent
- Memory = Learned Weights
- Past Experience = Bias Term

Connection: ML algorithms use these same principles!

Slide 3: The ML Learning Cycle

How Machines Learn - Visual Overview

[DIAGRAM: Circular Flow]

```
┌─────────────────┐
│  RAW DATA       │ (emails, images, sensors)
└────────┬────────┘
         ↓
┌──────────────────────────┐
│ LEARNING ALGORITHM │ (Find-S, Neural Nets)
└────────┬─────────────────┘
         ↓
┌─────────────────────┐
│ HYPOTHESIS      │ (Learned Rules/Patterns)
└────────┬────────────┘
         ↓
┌──────────────────────┐
│ PREDICTIONS     │ → Test on New Data
└────────┬─────────────┘
         ↓
┌──────────────────────┐
│ FEEDBACK/ERROR  │ → Improve
└──────────────────────┘
```

Key Components:

- Data feeds the algorithm
- Algorithm searches for patterns

- Creates a hypothesis (model)
- Tests and improves iteratively

Slide 4: Learning Objectives

What You'll Be Able To Do

By the end of this module, you will:

1. **Define** well-posed learning problems with task, performance, and experience
2. **Design** the four key components of any learning system
3. **Explain** concept learning as a search through hypothesis space
4. **Apply** the Find-S algorithm to find maximally specific hypotheses
5. **Implement** the Candidate-Elimination algorithm using version spaces
6. **Analyze** the role of inductive bias in learning

Career Impact: These fundamentals form the foundation for understanding modern ML frameworks like TensorFlow, PyTorch, and scikit-learn.

Slide 5: What is a Well-Posed Learning Problem?

The Three Essential Elements

Definition: A computer program learns from experience E with respect to some task T and performance measure P, if its performance at T (measured by P) improves with experience E.

The Three Components:

1. Task (T) - What are we trying to do?

- Classify emails as spam/not spam
- Predict house prices
- Play chess

2. Performance Measure (P) - How do we measure success?

- Accuracy percentage
- Mean squared error
- Win rate

3. Experience (E) - What data do we learn from?

- Labeled examples (supervised)
- Unlabeled data (unsupervised)
- Feedback from actions (reinforcement)

Think About: Banking fraud detection

- T: Classify transactions as fraudulent or legitimate
- P: Percentage correctly classified
- E: Database of past transactions with fraud labels

Slide 6: Real-World Example - Email Spam Filter

Applying the Well-Posed Framework

Task (T): Classify emails as spam or not spam

Performance Measure (P):

- Accuracy: % of emails correctly classified
- False Positive Rate: % of legitimate emails marked as spam (Critical!)
- False Negative Rate: % of spam emails that get through

Experience (E):

- Thousands of labeled emails (spam/not spam)
- User feedback (marking emails as spam)
- Features: sender, subject keywords, links, attachments

Business Impact:

- Gmail blocks 99.9% of spam and phishing attempts
- Processes 100+ billion spam attempts daily
- False positive rate < 0.05% (critical for user trust)

Why This Matters: A well-defined problem is half solved!

Slide 7: Designing a Learning System - The Checkers Game

Four Key Design Choices

The Classic Example: Teaching a computer to play checkers

Step-by-Step Design:

Choice 1: Determine the Training Experience

- Direct or Indirect feedback?
- Teacher control vs. Self-play?
- Representative distribution of examples?

Choice 2: Choose the Target Function

- What exactly should the system learn?
- ChooseMove: Board → Move (too complex!)
- V: Board → $\mathbb{R}$ (better - evaluation score)

Choice 3: Choose Representation

- How to represent the learned function?
- Linear combination? Decision tree? Neural network?

Choice 4: Choose Learning Algorithm

- How to fit the data to the representation?
- Minimize error? Gradient descent?

Key Insight: These four choices apply to ANY machine learning problem!

Slide 8: Checkers Example - Target Function Design

What Should the Computer Learn?

Option 1: ChooseMove Function

- Input: Current board state
- Output: Best legal move
- Problem: Too complex, difficult to learn directly!

Option 2: Board Evaluation Function V ✓

- Input: Current board state
- Output: Score (how good is this position?)
- V(b) = +100 if win, -100 if loss, 0 if draw
- For non-terminal: Estimate how likely to win from this state

Representation Choice - Linear Function:

$$V(b) = w_0 + w_1 \cdot x_1 + w_2 \cdot x_2 + w_3 \cdot x_3 + w_4 \cdot x_4 + w_5 \cdot x_5 + w_6 \cdot x_6$$

Board Features:

- x_1 = # black pieces on board
- x_2 = # red pieces on board
- x_3 = # black kings
- x_4 = # red kings
- x_5 = # black pieces threatened
- x_6 = # red pieces threatened

The Learning Task: Find weights $w_0 \ldots w_6$ that best predict game outcomes!

Slide 9: Quick Check - Understanding the Basics

Test Your Knowledge

Question 1 (Conceptual): Which component determines "how we measure success" in a learning problem?

- a) Task
- b) Performance Measure ✓
- c) Experience
- d) Hypothesis

Question 2 (Predictive): If we increase training data from 1,000 to 10,000 examples, what happens?

- a) Task changes
- b) Performance typically improves ✓
- c) Algorithm changes
- d) Nothing changes

Question 3 (Practical): For a house price prediction system, which is the best performance measure?

- a) Number of houses in dataset
- b) Mean Absolute Error ($ difference) ✓
- c) Number of features
- d) Training time

Discuss: Why is defining the right performance measure critical for business success?

Slide 10: Concept Learning - The Core Problem

Learning Boolean-Valued Functions

What is Concept Learning? Inferring a boolean-valued function from training examples of its input and output.

Everyday Example: Learning "days Aldo enjoys water sports"

Attributes describe each day:

- Sky: Sunny, Cloudy, Rainy
- AirTemp: Warm, Cold
- Humidity: Normal, High
- Wind: Strong, Weak
- Water: Warm, Cool
- Forecast: Same, Change

Target Concept: EnjoySport: Yes or No

Training Data:

Sky	AirTemp	Humidity	Wind	Water	Forecast	EnjoySport
Sunny	Warm	Normal	Strong	Warm	Same	**Yes**
Sunny	Warm	High	Strong	Warm	Same	**Yes**
Rainy	Cold	High	Strong	Warm	Change	**No**
Sunny	Warm	High	Strong	Cool	Change	**Yes**

The Challenge: Can you predict if Aldo will enjoy water sports on a new day?

Slide 11: Hypothesis Representation

Describing Concepts with Constraints

How do we represent hypotheses?

Each hypothesis = conjunction of constraints on attributes

Three Types of Constraints:

1. **"?" (Any value)** - This attribute can be anything
2. **Specific value** - Must match exactly (e.g., "Warm")
3. **"∅" (No value)** - Impossible to satisfy (all negative)

Example Hypotheses:

h_1 = ⟨Sunny, ?, ?, Strong, ?, ?⟩

- Means: Sky must be Sunny, Wind must be Strong, others don't matter

h_2 = ⟨Sunny, ?, ?, ?, ?, ?⟩

- Means: Only Sky = Sunny matters

h_3 = ⟨?, Warm, Normal, Strong, Warm, Same⟩

- Means: All six attributes must match exactly

Most General Hypothesis: ⟨?, ?, ?, ?, ?, ?⟩ (all instances positive)

Most Specific Hypothesis: ⟨∅, ∅, ∅, ∅, ∅, ∅⟩ (all instances negative)

Hypothesis Space Size: $3 \times 2 \times 2 \times 2 \times 2 \times 2 = 96$ syntactically distinct

- Plus 1 empty hypothesis = 97 total
- But only 973 semantically distinct concepts possible!

Slide 12: The General-to-Specific Ordering

Structure of the Hypothesis Space

Key Idea: Hypotheses can be ordered by specificity

Definition: Hypothesis h_j is **more general than or equal to** h_k (written $h_j \geq_m h_k$) if:

- Every instance classified positive by h_k is also classified positive by h_j

Visual Representation:

```
        Most General
      ⟨?, ?, ?, ?, ?, ?⟩
            ↑
    More general hypotheses
    ⟨Sunny, ?, ?, ?, ?, ?⟩
         ↑      ↑
⟨Sunny, Warm, ?, ?, ?, ?⟩  ⟨?, Warm, ?, ?, ?, ?⟩
            ↑
    More specific hypotheses
⟨Sunny, Warm, Normal, Strong, Warm, Same⟩
            ↓
        Most Specific
      ⟨∅, ∅, ∅, ∅, ∅, ∅⟩
```

Why This Matters:

- Provides structure to search through hypothesis space
- Enables efficient learning algorithms
- Basis for Find-S and Candidate-Elimination

Slide 13: Find-S Algorithm - Finding Maximally Specific Hypothesis

Step-by-Step Learning Process

Algorithm Goal: Find the most specific hypothesis that fits all positive examples

The Find-S Algorithm:

```
1. Initialize h to the most specific hypothesis ⟨∅, ∅, ∅, ∅, ∅, ∅⟩

2. For each positive training example x:
  - For each attribute constraint a_i in h:
    If constraint a_i is satisfied by x:
      Do nothing
    Else:
      Replace a_i in h by the next more general constraint
      that is satisfied by x

3. Output hypothesis h
```

Key Properties:

- Only considers positive examples

- Moves from specific to general
- Guaranteed to find maximally specific hypothesis
- Ignores negative examples (limitation!)

Python Code Example:

```python
def find_s(examples):
    # Initialize to most specific
    h = ['∅', '∅', '∅', '∅', '∅', '∅']

    for x, label in examples:
        if label == 'Yes':  # Only positive examples
            for i in range(len(h)):
                if h[i] == '∅':
                    h[i] = x[i]  # First positive example
                elif h[i] != x[i]:
                    h[i] = '?'  # Generalize if different

    return h
```

Slide 14: Find-S Example Trace

Learning from the EnjoySport Data

Training Examples:

1. ⟨Sunny, Warm, Normal, Strong, Warm, Same⟩ → **Yes**
2. ⟨Sunny, Warm, High, Strong, Warm, Same⟩ → **Yes**
3. ⟨Rainy, Cold, High, Strong, Warm, Change⟩ → **No**
4. ⟨Sunny, Warm, High, Strong, Cool, Change⟩ → **Yes**

Step-by-Step Execution:

Initial: $h_0 = \langle \emptyset, \emptyset, \emptyset, \emptyset, \emptyset, \emptyset \rangle$

After Example 1 (+): h_1 = ⟨Sunny, Warm, Normal, Strong, Warm, Same⟩

- First positive example, copy all attributes

After Example 2 (+): h_2 = ⟨Sunny, Warm, ?, Strong, Warm, Same⟩

- Humidity differs (Normal vs High) → generalize to "?"

After Example 3 (-): h_3 = ⟨Sunny, Warm, ?, Strong, Warm, Same⟩

- Negative example, no change (Find-S ignores negatives)

After Example 4 (+): h_4 = ⟨Sunny, Warm, ?, Strong, ?, ?⟩

- Water differs (Warm vs Cool) → "?"
- Forecast differs (Same vs Change) → "?"

Final Hypothesis: ⟨Sunny, Warm, ?, Strong, ?, ?⟩

Interpretation: Aldo enjoys water sports on Sunny, Warm days with Strong wind!

Slide 15: Limitations of Find-S & Version Spaces

Why We Need Something Better

Find-S Problems:

1. **Ignores Negative Examples**
   - Can't detect when hypothesis is too general
   - Might classify negative examples as positive
2. **No Way to Know if Converged**
   - Did we find the correct concept?
   - Are there other equally good hypotheses?
3. **Can't Handle Inconsistent Data**
   - What if training examples have noise/errors?
   - No mechanism to detect contradictions

Solution: Version Spaces 🎯

Key Idea: Instead of a single hypothesis, maintain the set of ALL hypotheses consistent with training data!

Version Space Definition: The subset of all hypotheses from H that are consistent with the observed training examples D.

Mathematical Notation: $VS_{h,D} = \{h \in H \mid Consistent(h, D)\}$

Representation: Rather than list all consistent hypotheses (could be thousands!), represent by:

- **S (Specific boundary):** Most specific consistent hypotheses
- **G (General boundary):** Most general consistent hypotheses

---

Slide 16: Candidate-Elimination Algorithm

Maintaining Version Spaces Efficiently

The Algorithm:

```
Initialize:
  G ← {⟨?, ?, ?, ?, ?, ?⟩}  (most general hypothesis)
  S ← {⟨∅, ∅, ∅, ∅, ∅, ∅⟩}  (most specific hypothesis)

For each training example d = ⟨x, c(x)⟩:

  If d is a POSITIVE example:
    • Remove from G any hypothesis inconsistent with d
    • For each s in S not consistent with d:
        - Remove s from S
        - Add all minimal generalizations h of s:
            where h is consistent with d
            and some member of G is more general than h
    • Remove from S any hypothesis more general than another in S

  If d is a NEGATIVE example:
    • Remove from S any hypothesis inconsistent with d
    • For each g in G not consistent with d:
        - Remove g from G
        - Add all minimal specializations h of g:
            where h is consistent with d
            and some member of S is more specific than h
    • Remove from G any hypothesis less general than another in G
```

Key Insight: Version space is fully characterized by S and G boundaries!

Slide 17: Candidate-Elimination Example

Learning EnjoySport Step-by-Step

Training Data:

1. ⟨Sunny, Warm, Normal, Strong, Warm, Same⟩ → **Yes**
2. ⟨Sunny, Warm, High, Strong, Warm, Same⟩ → **Yes**
3. ⟨Rainy, Cold, High, Strong, Warm, Change⟩ → **No**
4. ⟨Sunny, Warm, High, Strong, Cool, Change⟩ → **Yes**

Initial State:

- S_0 = {⟨∅, ∅, ∅, ∅, ∅, ∅⟩}
- G_0 = {⟨?, ?, ?, ?, ?, ?⟩}

After Example 1 (+):

- S_1 = {⟨Sunny, Warm, Normal, Strong, Warm, Same⟩}
- G_1 = {⟨?, ?, ?, ?, ?, ?⟩} (no change)

After Example 2 (+):

- S_2 = {⟨Sunny, Warm, ?, Strong, Warm, Same⟩} (generalized Humidity)
- G_2 = {⟨?, ?, ?, ?, ?, ?⟩}

After Example 3 (-): ⟨Rainy, Cold, High, Strong, Warm, Change⟩

- S_3 = {⟨Sunny, Warm, ?, Strong, Warm, Same⟩} (still consistent)
- G_3 = {⟨Sunny, ?, ?, ?, ?, ?⟩, ⟨?, Warm, ?, ?, ?, ?⟩, ⟨?, ?, ?, ?, ?, Same⟩} (G specialized to exclude negative example)

After Example 4 (+):

- S_4 = {⟨Sunny, Warm, ?, Strong, ?, ?⟩}
- G_4 = {⟨Sunny, ?, ?, ?, ?, ?⟩, ⟨?, Warm, ?, ?, ?, ?⟩}

Final Version Space: All hypotheses between S_4 and G_4

Slide 18: Interview Question - Version Spaces

Testing Your Understanding

Junior Level: *"What is the difference between Find-S and Candidate-Elimination algorithms?"*

Expected Answer:

- Find-S maintains only one hypothesis (most specific)
- Candidate-Elimination maintains all consistent hypotheses via S and G boundaries
- Find-S ignores negative examples; C-E uses both positive and negative
- C-E can detect inconsistencies; Find-S cannot

Mid Level: *"Given this version space, classify the new instance ⟨Sunny, Warm, Normal, Light, Warm, Same⟩"*

With S = {⟨Sunny, Warm, ?, Strong, ?, ?⟩} and G = {⟨Sunny, ?, ?, ?, ?, ?⟩, ⟨?, Warm, ?, ?, ?, ?⟩}

Expected Answer:

- Check against S: Wind is Light (not Strong) → **Not satisfied by S**
- Check against G: Both satisfied
- Result: **Cannot classify with confidence** (some hypotheses say yes, some say no)
- Need more training data to narrow version space

Senior Level: *"What happens to the version space size as we add more training examples? Can it increase?"*

Expected Answer:

- Version space can only shrink or stay same (monotonic)
- Each example eliminates hypotheses
- Cannot increase because we never add back eliminated hypotheses
- Converges when S = G (single hypothesis remains)

Slide 19: Inductive Bias - The Necessity of Assumptions

Why Learning Requires Bias

The Fundamental Problem:

Question: Can an unbiased learner generalize beyond training data?

Answer: NO! An unbiased learner cannot make inductive leaps.

Example - Unbiased Learner (Power Set Hypothesis Space):

Given 3 positive examples:

- ⟨Sunny, Warm, Normal, Strong, Warm, Same⟩ → Yes
- ⟨Sunny, Warm, High, Strong, Warm, Same⟩ → Yes
- ⟨Rainy, Cold, High, Strong, Cool, Change⟩ → No

With power set H: Every unobserved instance will be classified positive by exactly HALF the consistent hypotheses!

- Cannot make predictions with confidence
- Voting provides no information

The Futility of Bias-Free Learning: Without assumptions, every classification is equally likely!

Inductive Bias Definition: The set of assumptions a learner uses to predict outputs for inputs it has not encountered.

Candidate-Elimination Inductive Bias: *"The target concept can be represented as a conjunction of attribute constraints."*

Key Insight:

- Stronger bias → more assumptions → better generalization (if assumptions correct)
- Weaker bias → fewer assumptions → less generalization ability
- No bias → no generalization!

Trade-off: Bias allows learning but risks being wrong if assumptions don't hold.

Slide 20: Key Takeaways & Next Steps

Summary & Hands-On Assignment

Key Takeaways:

1. **Well-posed learning requires:** Task (T), Performance (P), Experience (E)
2. **Four design choices:** Training experience, target function, representation, learning algorithm
3. **Concept learning** is search through hypothesis space with general-to-specific ordering
4. **Find-S:** Finds maximally specific hypothesis (ignores negatives)
5. **Version Spaces:** All hypotheses consistent with data, represented by S and G boundaries
6. **Candidate-Elimination:** Efficiently maintains version space using both positive and negative examples
7. **Inductive bias is necessary:** Bias-free learning cannot generalize beyond training data

Hands-On Assignment (Due: Next Week):

Objective: Implement and test the Candidate-Elimination algorithm

Required Tasks:

1. Implement the Candidate-Elimination algorithm in Python
2. Apply it to the EnjoySport dataset (provided)
3. Visualize how S and G boundaries evolve with each example
4. Test classification on 5 new instances

Optional Challenge: Modify the algorithm to handle noisy data (contradictory examples)

Deliverables:

- Python code (well-commented)
- Report showing S/G evolution

- Analysis of 3 classification decisions

Next Module Preview:

- Decision Tree Learning
- Overfitting and generalization
- Practical ML with scikit-learn

Resources: Textbook Chapter 1-2, online notebooks at course website

---

End of Module 1 Slide Deck