

---

# Machine Learning Introduction

Based on **Raymond J. Mooney's slides**  
University of Texas at Austin

# What is Learning?

---

- Herbert Simon: “Learning is any process by which a system improves performance from experience.”
- What is the task?
  - Classification
  - Problem solving / planning / control

# Classification

---

- Assign object/event to one of a given finite set of categories.
  - Medical diagnosis
  - Credit card applications or transactions
  - Fraud detection in e-commerce
  - Worm detection in network packets
  - Spam filtering in email
  - Recommended articles in a newspaper
  - Recommended books, movies, music, or jokes
  - Financial investments
  - DNA sequences
  - Spoken words
  - Handwritten letters
  - Astronomical images

# Problem Solving / Planning / Control

---

- Performing actions in an environment in order to achieve a goal.
  - Solving calculus problems
  - Playing checkers, chess, or backgammon
  - Balancing a pole
  - Driving a car or a jeep
  - Flying a plane, helicopter, or rocket
  - Controlling an elevator
  - Controlling a character in a video game
  - Controlling a mobile robot

# Measuring Performance

---

- Classification Accuracy
- Solution correctness
- Solution quality (length, efficiency)
- Speed of performance

# Why Study Machine Learning?

## Engineering Better Computing Systems

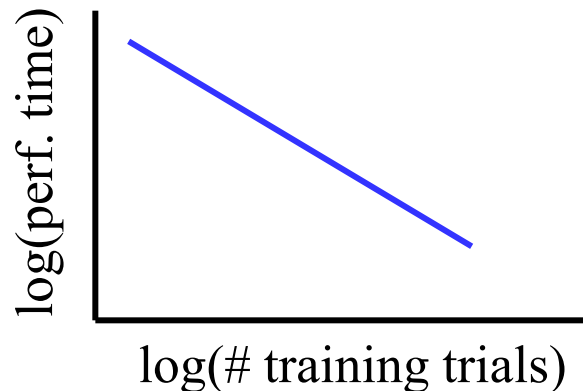
---

- Develop systems that are too difficult/expensive to construct manually because they require specific detailed skills or knowledge tuned to a specific task (*knowledge engineering bottleneck*).
- Develop systems that can automatically adapt and customize themselves to individual users.
  - Personalized news or mail filter
  - Personalized tutoring
- Discover new knowledge from large databases (*data mining*).
  - Market basket analysis (e.g. diapers and beer)
  - Medical text mining (e.g. migraines to calcium channel blockers to magnesium)

# Why Study Machine Learning? Cognitive Science

---

- Computational studies of learning may help us understand learning in humans and other biological organisms.
  - Hebbian neural learning
    - “Neurons that fire together, wire together.”
  - Human’s relative difficulty of learning disjunctive concepts vs. conjunctive ones.
  - Power law of practice



# Why Study Machine Learning?

## The Time is Ripe

---

- Many basic effective and efficient algorithms available.
- Large amounts of on-line data available.
- Large amounts of computational resources available.



# Related Disciplines

---

- Artificial Intelligence
- Data Mining
- Probability and Statistics
- Information theory
- Numerical optimization
- Computational complexity theory
- Control theory (adaptive)
- Psychology (developmental, cognitive)
- Neurobiology
- Linguistics
- Philosophy

# Defining the Learning Task

---

Improve on task, T, with respect to performance metric, P, based on experience, E.

T: Playing checkers

P: Percentage of games won against an arbitrary opponent

E: Playing practice games against itself

T: Recognizing hand-written words

P: Percentage of words correctly classified

E: Database of human-labeled images of handwritten words

T: Driving on four-lane highways using vision sensors

P: Average distance traveled before a human-judged error

E: A sequence of images and steering commands recorded while observing a human driver.

T: Categorize email messages as spam or legitimate.

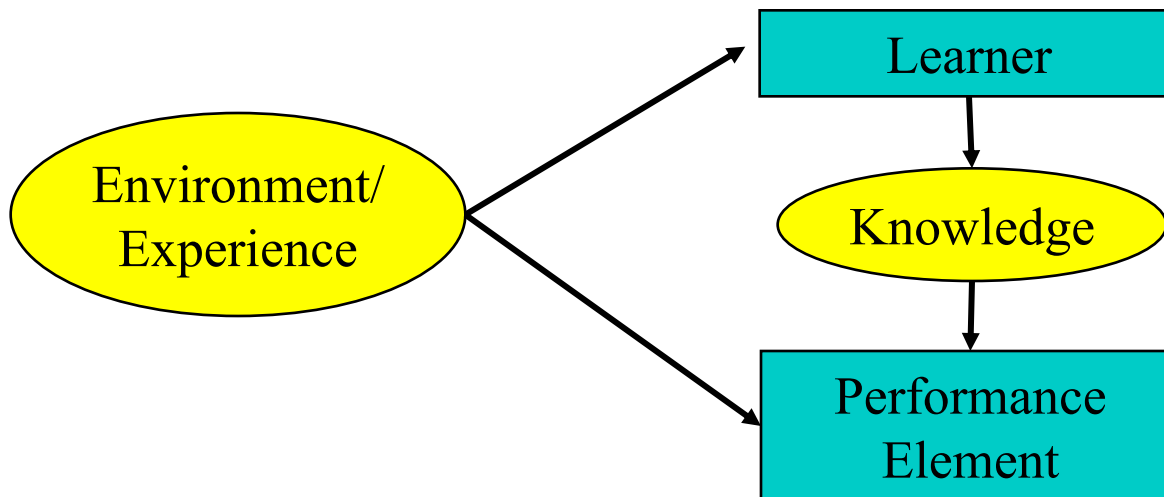
P: Percentage of email messages correctly classified.

E: Database of emails, some with human-given labels

# Designing a Learning System

---

- Choose the training experience
- Choose exactly what is to be learned, i.e. the *target function*.
- Choose how to represent the target function.
- Choose a learning algorithm to infer the target function from the experience.



# Sample Learning Problem

---

- Learn to play checkers from self-play
- We will develop an approach analogous to that used in the first machine learning system developed by Arthur Samuels at IBM in 1959.

# Training Experience

---

- **Direct experience:** Given sample input and output pairs for a useful target function.
  - Checker boards labeled with the correct move, e.g. extracted from record of expert play
- **Indirect experience:** Given feedback which is *not* direct I/O pairs for a useful target function.
  - Potentially arbitrary sequences of game moves and their final game results.
- **Credit/Blame Assignment Problem:** How to assign credit blame to individual moves given only indirect feedback?

# Source of Training Data

---

- Provided random examples outside of the learner's control.
  - Negative examples available or only positive?
- Good training examples selected by a “benevolent teacher.”
  - “Near miss” examples
- Learner can query an oracle about class of an unlabeled example in the environment.
- Learner can construct an arbitrary example and query an oracle for its label.
- Learner can design and run experiments directly in the environment without any human guidance.

# Training vs. Test Distribution

---

- Generally assume that the training and test examples are independently drawn from the same overall distribution of data.
  - IID: Independently and identically distributed
- If examples are not independent, requires *collective classification*.
- If test distribution is different, requires *transfer learning*.

# Choosing a Target Function

---

- What function is to be learned and how will it be used by the performance system?
- For checkers, assume we are given a function for generating the legal moves for a given board position and want to decide the best move.
  - Could learn a function:  
ChooseMove(board, legal-moves)  $\rightarrow$  best-move
  - Or could learn an *evaluation function*,  $V(\text{board}) \rightarrow \mathbb{R}$ , that gives each board position a score for how favorable it is.  $V$  can be used to pick a move by applying each legal move, scoring the resulting board position, and choosing the move that results in the highest scoring board position.



# Ideal Definition of $V(b)$

---

- If  $b$  is a final winning board, then  $V(b) = 100$
- If  $b$  is a final losing board, then  $V(b) = -100$
- If  $b$  is a final draw board, then  $V(b) = 0$
- Otherwise, then  $V(b) = V(b^*)$ , where  $b^*$  is the highest scoring final board position that is achieved starting from  $b$  and playing optimally until the end of the game (assuming the opponent plays optimally as well).
  - Can be computed using complete mini-max search of the finite game tree.

# Approximating $V(b)$

---

- Computing  $V(b)$  is intractable since it involves searching the complete exponential game tree.
- Therefore, this definition is said to be *non-operational*.
- An *operational* definition can be computed in reasonable (polynomial) time.
- Need to learn an operational *approximation* to the ideal evaluation function.

# Representing the Target Function

---

- Target function can be represented in many ways: lookup table, symbolic rules, numerical function, neural network.
- There is a trade-off between the expressiveness of a representation and the ease of learning.
- The more expressive a representation, the better it will be at approximating an arbitrary function; however, the more examples will be needed to learn an accurate function.

# Linear Function for Representing $V(b)$

---

- In checkers, use a linear approximation of the evaluation function.

$$V(b) = w_1 \cdot bp(b) + w_2 \cdot rp(b) + w_3 \cdot bk(b) + w_4 \cdot rk(b) + w_5 \cdot bt(b) + w_6 \cdot rt(b)$$

- $bp(b)$ : number of black pieces on board  $b$
- $rp(b)$ : number of red pieces on board  $b$
- $bk(b)$ : number of black kings on board  $b$
- $rk(b)$ : number of red kings on board  $b$
- $bt(b)$ : number of black pieces threatened (i.e. which can be immediately taken by red on its next turn)
- $rt(b)$ : number of red pieces threatened

# Obtaining Training Values

---

- Direct supervision may be available for the target function.
  - $\langle \langle bp=3, rp=0, bk=1, rk=0, bt=0, rt=0 \rangle, 100 \rangle$   
(win for black)
- With indirect feedback, training values can be estimated using *temporal difference learning* (used in *reinforcement learning* where supervision is *delayed reward*).

# Temporal Difference Learning

---

- Estimate training values for intermediate (non-terminal) board positions by the estimated value of their successor in an actual game trace.

$$V_{train}(b) = V(\text{successor}(b))$$

where  $\text{successor}(b)$  is the next board position where it is the program's move in actual play.

- Values towards the end of the game are initially more accurate and continued training slowly “backs up” accurate values to earlier board positions.

# Learning Algorithm

---

- Uses training values for the target function to induce a hypothesized definition that fits these examples and hopefully generalizes to unseen examples.
- In statistics, learning to approximate a continuous function is called *regression*.
- Attempts to minimize some measure of error (*loss function*) such as *mean squared error*:

$$E = \frac{\sum (b - \tilde{b})^2}{|B|}$$

# Least Mean Squares (LMS) Algorithm

---

- A gradient descent algorithm that incrementally updates the weights of a linear function in an attempt to minimize the mean squared error

Until weights converge :

For each training example  $b$  do :

- 1) Compute the absolute error :

$$err(b) = |act(b) - b|$$

- 2) For each board feature,  $f_i$ , update its weight,  $w_i$  :

$$w_i = w_i + c \cdot f_i \cdot err(b)$$

for some small constant (learning rate)  $c$



# LMS Discussion

---

- Intuitively, LMS executes the following rules:
  - If the output for an example is correct, make no change.
  - If the output is too high, lower the weights proportional to the values of their corresponding features, so the overall output decreases
  - If the output is too low, increase the weights proportional to the values of their corresponding features, so the overall output increases.
- Under the proper weak assumptions, LMS can be proven to eventually converge to a set of weights that minimizes the mean squared error.

# Lessons Learned about Learning

---

- Learning can be viewed as using direct or indirect experience to approximate a chosen target function.
- Function approximation can be viewed as a search through a space of hypotheses (representations of functions) for one that best fits a set of training data.
- Different learning methods assume different hypothesis spaces (representation languages) and/or employ different search techniques.

# Various Function Representations

---

- Numerical functions
  - Linear regression
  - Neural networks
  - Support vector machines
- Symbolic functions
  - Decision trees
  - Rules in propositional logic
  - Rules in first-order predicate logic
- Instance-based functions
  - Nearest-neighbor
  - Case-based
- Probabilistic Graphical Models
  - Naïve Bayes
  - Bayesian networks
  - Hidden-Markov Models (HMMs)
  - Probabilistic Context Free Grammars (PCFGs)
  - Markov networks

# Various Search Algorithms

---

- Gradient descent
  - Perceptron
  - Backpropagation
- Dynamic Programming
  - HMM Learning
  - PCFG Learning
- Divide and Conquer
  - Decision tree induction
  - Rule learning
- Evolutionary Computation
  - Genetic Algorithms (GAs)
  - Genetic Programming (GP)
  - Neuro-evolution = evolutionary algorithms for training neural nets

# Evaluation of Learning Systems

---

- Experimental
  - Conduct controlled cross-validation experiments to compare various methods on a variety of benchmark datasets.
  - Gather data on their performance, e.g. test accuracy, training-time, testing-time.
  - Analyze differences for statistical significance.
- Theoretical
  - Analyze algorithms mathematically and prove theorems about their:
    - Computational complexity
    - Ability to fit training data
    - Sample complexity (number of training examples needed to learn an accurate function)

# History of Machine Learning

---

- 1950s
  - Samuel's checker player
  - Selfridge's Pandemonium = collection of specialised demons
- 1960s:
  - Neural networks: Perceptron
  - Pattern recognition
  - Learning in the limit theory
  - Minsky and Papert prove limitations of Perceptron
- 1970s:
  - Symbolic concept induction
  - Winston's arch learner
  - Expert systems and the knowledge acquisition bottleneck
  - Quinlan's ID3
  - Michalski's AQ and soybean diagnosis
  - Scientific discovery with BACON
  - Mathematical discovery with AM

# History of Machine Learning (cont.)

---

- 1980s:
  - Advanced decision tree and rule learning
  - Explanation-based Learning (EBL)
  - Learning and planning and problem solving
  - Utility problem
  - Analogy
  - Cognitive architectures
  - Resurgence of neural networks (connectionism, backpropagation)
  - Valiant's PAC Learning Theory
  - Focus on experimental methodology
- 1990s
  - Data mining
  - Adaptive software agents and web applications
  - Text learning
  - Reinforcement learning (RL)
  - Inductive Logic Programming (ILP)
  - Ensembles: Bagging, Boosting, and Stacking
  - Bayes Net learning

# History of Machine Learning (cont.)

---

- 2000s
  - Support vector machines
  - Kernel methods
  - Graphical models
  - Statistical relational learning
  - Transfer learning
  - Sequence labeling
  - Collective classification and structured outputs
  - Computer Systems Applications
    - Compilers
    - Debugging
    - Graphics
    - Security (intrusion, virus, and worm detection)
  - Email management
  - Personalized assistants that learn
  - Learning in robotics and vision