

## 633-600 Machine Learning

- Instructor: Yoonsuck Choe
  - Contact info: HRBB 322B, 845-5466, [choe@tamu.edu](mailto:choe@tamu.edu)
- TA: see web page
- Course web page: <http://courses.cs.tamu.edu/choe/17spring/633>

1

## Course Info

- Grading, academic policy, students with disabilities, lecture notes, computer accounts, programming languages.
- See course web page.

3

## Textbook

- Ethem Alpaydin (2014) “Introduction to Machine Learning”, 3rd edition. MIT Press.
- Book webpage: <http://www.cmpe.boun.edu.tr/~ethem/i2ml3e/>
- Optional (but strongly recommended): Tom M. Mitchell (1997) “Machine Learning”, McGraw-Hill.
- Book webpage: <http://www.cs.cmu.edu/~tom/mlbook.html>
- Text and figures, etc. will be quoted from the textbook without repeated acknowledgment. Instructor’s perspective will be indicated by “YC” where appropriate.

2

## Relation to Other Courses

Some overlaps:

- Neural Networks: perceptrons, backpropagation, radial basis function networks etc.
- Pattern analysis: Bayesian learning, instance-based learning
- Artificial intelligence: decision trees (in some courses), neural networks (in some courses).
- Statistics: hypothesis testing
- (Relatively) unique to this course: computational learning theory, genetic algorithms, reinforcement learning, decision trees (in depth treatment), local learning (some aspects), dimensionality reduction, deep learning (also in neural networks)

4

## ML Overview (I)

- How can machines (computers) learn?  
How can machines **improve automatically with experience**?
- How can machines learn **from data**?
- Benefits:
  - Improved performance
  - Automated optimization
  - New uses of computers
  - Reduced programming (YC)
  - Insights into human learning and learning disabilities

5

## ML Overview (II)

- Current status: Yet unsolved problem.
  - Theoretical insights emerging.
  - Practical applications.
  - Huge data volume demands ML, and provides opportunity to ML (datamining).
- State of the art:
  - speech recognition
  - medical predictions
  - fraud detection
  - drive autonomous vehicles (highway and desert)
  - board games (Backgammon, Chess, and Go!)
  - theoretical bounds on error, number of inputs needed, etc.

6

## ML Overview (III)

Multidisciplinary roots:

- AI
- probability and statistics
- computational complexity theory
- control theory
- information theory
- philosophy
- psychology
- neurobiology

7

## Well-Posed Learning Problem

A program is said to **learn** from

- experience  $E$  with respect to
- task  $T$  and
- performance measure  $P$ ,
- $P$  in  $T$  increase with  $E$ .

Examples: Playing checkers, Handwriting recognition, Robot driving, etc.

Goal of ML: “define precisely a class of problems that encompasses interesting forms of learning [but not all: YC], to explore algorithms that solve such problems, and to understand the fundamental structure of learning problems and processes” (Mitchell, 1997)

8

## Designing a Learning System (I)

Training experience:

- direct vs. indirect (problem of credit assignment)
- degree of control over training examples (teacher-dependent or learner-generated)
- closeness of training example distribution to true distribution over which  $P$  is measured: in many cases, ML algorithms assume that both distributions are similar, which may not be the case in practice.

9

## Design: Target Function (I)

Type of knowledge to be learned: for example, we want to learn **best move** in a board game.

- Can represent as a function ( $B$ : board states,  $M$ : moves):

$$\textit{ChooseMove} : B \rightarrow M,$$

but it is hard to learn directly.

11

## Designing a Learning System (II)

Remaining design choices:

- Exact type of knowledge to be learned.
- A representation for this target knowledge.
- A learning mechanism.
- functional/operational principle giving rise to the learning mechanism (YC)

10

## Design: Target Function (II)

- Another function ( $B$ : board states,  $\mathcal{R}$ : real numbers):

$$V : B \rightarrow \mathcal{R},$$

which gives the **evaluation** of each board state.

- $V(b = \textit{win}) = 100$
- $V(b = \textit{lose}) = -100$
- $V(b = \textit{draw}) = 0$
- $V(b = \textit{otherwise}) = V(b')$ , where  $b'$  is the best final board state that can be reached from  $b$ .
- However, this is not **efficiently computable**, i.e., it is a **nonoperational** definition.
- Goal of ML is to find an **operational** description of  $V$ , however, in practice, an **approximation** is all we can get.

12

## Design: Representation for Target Function

Given an ideal target function  $V$ , we want to learn an approximate function  $\hat{V}$ :

- Trade-off between rich and parsimonious representation.
- Example:  $\hat{V}$  as a linear combination of number of pieces, number of particular relational situations in the board (e.g., threatened), etc. (represented as  $x_i$ ) in board configuration  $b$ :

$$\hat{V}(b) = w_0 + \sum_{i=1}^n w_i x_i,$$

where  $w_i$  are the weight values **to be learned**.

- Advantage of the above representation: **reduction of scope (or dimensionality)** from the original problem.

13

## Design: Adjusting Weights (I)

Last component in defining a learning algorithm: adjustment of weights.

- Want to learn weights  $w_i$  that **best fit** the set of training samples  $\{ \langle b, V_{train}(b) \rangle \}$ .
- How to define best fit? Once we have  $\hat{V}$  we can calculate all  $\hat{V}(b)$  for all  $b$  in the training set, and calculate the error.

$$E \equiv \sum_{\langle b, V_{train}(b) \rangle \in \text{training set}} \left( V_{train}(b) - \hat{V}(b) \right)^2$$

- How to reduce  $E$ ?

15

## Design: Function Approximation Algorithm

Given **board state and true  $V$** , we want to learn the **weights  $w_i$**  that specify  $\hat{V}$ .

- Start with a set of a large number of input-target pairs  $\langle b, V_{train}(b) \rangle$ .
- Problem: cannot come up with a full set of  $\langle b, V_{train}(b) \rangle$  pairs.
- Solution: If  $V_{train}(b)$  is unknown, set it to the **estimated  $\hat{V}$**  of its successor board state:

$$V_{train}(b) = \hat{V}_{train}(Successor(b)).$$

14

## Design: Adjusting Weights (II)

**Least Mean Squares** (LMS) learning rule: For each training example  $\langle b, V_{train}(b) \rangle$ ,

- Use the current weights to calculate  $\hat{V}(b)$ .
- For each weight  $w_i$ , update it as

$$w_i \leftarrow w_i + \eta (V_{train}(b) - \hat{V}(b)) x_i,$$

where  $\eta$  is a small **learning rate** constant.

- The error  $V_{train}(b) - \hat{V}(b)$  and the input  $x_i$  both contribute to the weight update.

16

## Final Design

Putting together the system (checker player):

- Performance system: input = problem, output = solution trace = game history (using what is learned so far)
- Critic: input = solution trace, output = training examples (estimated  $V_{train}(b)$ )
- Generalizer: input = training examples, output = estimated hypothesis  $\hat{V}$  (i.e., learned weights  $w_i$ )
- Experiment generator: input = hypothesis  $\hat{V}$ , output = new problem (new initial condition, to explore particular regions)

17

## Alternatives (II)

- Memorize (instance-based learning)
- Spawn a population and make them compete with each other (genetic algorithms)
- Analyze and reason about things

19

## Alternatives (I)

- Training experience: against experts, against self, table of correct moves, ...
- Target function: board  $\rightarrow$  move, board  $\rightarrow$  value, ...
- Representation of target function: polynomial, linear function of small number of features, artificial neural network
- Learning algorithm: gradient descent, linear programming, ...

18

## Perspectives on ML: Hypothesis Space Search

- Useful to think of ML as **searching** a very large space of **possible hypotheses** to **best fit** the data and the learner's prior knowledge.
- For example, the hypothesis space for  $\hat{V}$  would be all possible  $\hat{V}$ s with different weight assignment.
- Useful concepts regarding hypothesis space search:
  - Size of hypothesis space
  - Number of training examples available/needed.
  - Confidence in generalizing to new unseen data.

20

## Issues in ML

- What algorithms exist for generalizable learners given specific training set? Requirements for convergence? Which algorithms are best for a particular domain?
- How much training data needed? Bounds on confidence, based on data size? How long to train?
- Use of prior knowledge?
- How to choose best training experience? Impact of the choice?
- How to reduce ML problem to function approximation?
- How can learner **alter** the representation itself?

21

## Classification of learning algorithms (YC)

What to do with given data? What kinds of data are given?

- Supervised learning: input-target pairs given.
- Unsupervised learning: only input distribution is given.
- Reinforcement learning: sparse reward signal is given for action based on sensory input; environment-altering actions.

22

## Broader questions (YC)

- Can machines themselves formulate their own learning tasks?
  - Can they come up with their own representations?
  - Can they come up with their own learning strategy?
  - Can they come up with their own motivation?
  - Can they come up with their own questions/problems?
- What if the machines are faced with multiple, possibly conflicting tasks? Can there be a meta learning algorithm?
- What if performance is hard to measure (i.e., hard to quantify, or even worse, subjective)?
- Lesson: think outside the box; question the questions themselves.

23