Machine Learning: Lecture 1

Overview of Machine Learning

(Based on Chapter 1 of Mitchell T.., *Machine Learning*, 1997)

Machine Learning: A Definition

Definition: A computer program is said to *learn* from experience E with respect to some class of tasks T and performance measure P, if its performance at tasks in T, as measured by P, improves with experience E.

Examples of Successful Applications of Machine Learning

- Learning to recognize spoken words (Lee, 1989; Waibel, 1989).
- Learning to drive an autonomous vehicle (Pomerleau, 1989).
- Learning to classify new astronomical structures (Fayyad et al., 1995).
- Learning to play world-class backgammon (Tesauro 1992, 1995).

Why is Machine Learning Important?

- Some tasks cannot be defined well, except by examples (e.g., recognizing people).
- Relationships and correlations can be hidden within large amounts of data. Machine Learning/Data Mining may be able to find these relationships.
- Human designers often produce machines that do not work as well as desired in the environments in which they are used.

Why is Machine Learning Important (Cont'd)?

- The amount of knowledge available about certain tasks might be too large for explicit encoding by humans (e.g., medical diagnostic).
- Environments change over time.
- New knowledge about tasks is constantly being discovered by humans. It may be difficult to continuously re-design systems "by hand".

Areas of Influence for Machine Learning

- **Statistics:** How best to use samples drawn from unknown probability distributions to help decide from which distribution some new sample is drawn?
- **Brain Models:** Non-linear elements with weighted inputs (Artificial Neural Networks) have been suggested as simple models of biological neurons.
- **Adaptive Control Theory:** How to deal with controlling a process having unknown parameters that must be estimated during operation?

Areas of Influence for Machine Learning (Cont'd)

- **Psychology:** How to model human performance on various learning tasks?
- **Artificial Intelligence:** How to write algorithms to acquire the knowledge humans are able to acquire, at least, as well as humans?
- **Evolutionary Models:** How to model certain aspects of biological evolution to improve the performance of computer programs?

Designing a Learning System: An Example

1. Problem Description
2. Choosing the Training Experience
3. Choosing the Target Function
4. Choosing a Representation for the Target Function
5. Choosing a Function Approximation Algorithm
6. Final Design

1. Problem Description: A Checker Learning Problem

- **Task T:** Playing Checkers
- **Performance Measure P:** Percent of games won against opponents
- **Training Experience E:** To be selected ==> Games Played against itself

2. Choosing the Training Experience

- **Direct versus Indirect Experience** [Indirect Experience gives rise to the **credit assignment** problem and is thus more difficult]
- **Teacher versus Learner Controlled Experience** [the teacher might provide training examples; the learner might suggest *interesting* examples and ask the teacher for their outcome; or the learner can be completely on its own with no access to correct outcomes]
- **How Representative is the Experience?** [Is the training experience representative of the task the system will actually have to solve? It is best if it is, but such a situation cannot systematically be achieved]

3. Choosing the Target Function

- Given a set of legal moves, we want to learn how to choose the best move [since the best move is not necessarily known, this is an *optimization* problem]
- *ChooseMove*: B --> M is called a **Target Function** [*ChooseMove*, however, is difficult to learn. An easier and related target function to learn is *V*: B --> R, which assigns a numerical score to each board. The better the board, the higher the score.]
- **Operational versus Non-Operational Description of a Target Function** [An operational description must be given]
- **Function Approximation** [The actual function can often not be learned and must be approximated]

4. Choosing a Representation for the Target Function

- **Expressiveness versus Training set size** [The more expressive the representation of the target function, the closer to the "truth" we can get. However, the more expressive the representation, the more training examples are necessary to choose among the large number of "representable" possibilities.]
- **Example of a representation:**
  - x1/x2 = # of black/red pieces on the board
  - x3/x4 = # of black/red king on the board
  - x5/x6 = # of black/red pieces threatened by red/black

wi's are adjustable or "learnable" coefficients

$$V(b) = w0+w1.x1+w2.x2+w3.x3+w4.x4+w5.x5+w6.x6$$

5. Choosing a Function Approximation Algorithm

- **Generating Training Examples of the form <b,Vtrain(b)>** [e.g. <x1=3, x2=0, x3=1, x4=0, x5=0, x6=0, +100 (=blacks won)]
  - Useful and Easy Approach: Vtrain(b) <- $\hat{V}$(Successor(b))
- **Training the System**
  - Defining a criterion for success [What is the error that needs to be minimized?]
  - Choose an algorithm capable of finding weights of a linear function that minimize that error [e.g. the Least Mean Square (LMS) training rule].

6. Final Design for Checkers Learning

- **The Performance Module**: Takes as input a new board and outputs a trace of the game it played against itself.
- **The Critic**: Takes as input the trace of a game and outputs a set of training examples of the target function
- **The Generalizer**: Takes as input training examples and outputs a *hypothesis* which estimates the target function. Good generalization to new cases is crucial.
- **The Experiment Generator:** Takes as input the current hypothesis (currently learned function) and outputs a new problem (an initial board state) for the performance system to explore

In this course, we are mostly concerned with the generalizer

Issues in Machine Learning (i.e., Generalization)

- What algorithms are available for learning a concept? How well do they perform?
- How much training data is sufficient to learn a concept with high confidence?
- When is it useful to use prior knowledge?
- Are some training examples more useful than others?
- What are best tasks for a system to learn?
- What is the best way for a system to represent its knowledge?