Designing a Learning System (agent viewpoint)

Relevant Readings: Sections 1.2-1.5, 2.1-2.2, 2.3 (but not 2.3.1) in Mitchell

CS495 - Machine Learning, Fall 2009

Building a checkers program [Mitchell, Chapter 1.2]

- Suppose we want to solve the following learning problem

Building a checkers program [Mitchell, Chapter 1.2]

- Suppose we want to solve the following learning problem
  - Task T: playing checkers

Building a checkers program [Mitchell, Chapter 1.2]

- Suppose we want to solve the following learning problem
  - Task T: playing checkers
  - Performance measure P: percent of games won in the world tournament

Building a checkers program [Mitchell, Chapter 1.2]

- Suppose we want to solve the following learning problem
  - Task T: playing checkers
  - Performance measure P: percent of games won in the world tournament
  - Training experience E: games played against itself

Building a checkers program [Mitchell, Chapter 1.2]

- Suppose we want to solve the following learning problem
  - Task T: playing checkers
  - Performance measure P: percent of games won in the world tournament
  - Training experience E: games played against itself
- Since we don't have complete knowledge of what the optimal moves are in checkers, providing suitable *direct* training examples would be difficult.

Building a checkers program [Mitchell, Chapter 1.2]

- Suppose we want to solve the following learning problem
  - Task T: playing checkers
  - Performance measure P: percent of games won in the world tournament
  - Training experience E: games played against itself
- Since we don't have complete knowledge of what the optimal moves are in checkers, providing suitable *direct* training examples would be difficult.
- Instead we will need to work with *indirect* knowledge of how the game turned out.

Building a checkers program [Mitchell, Chapter 1.2]

- Suppose we want to solve the following learning problem
  - Task T: playing checkers
  - Performance measure P: percent of games won in the world tournament
  - Training experience E: games played against itself
- Since we don't have complete knowledge of what the optimal moves are in checkers, providing suitable *direct* training examples would be difficult.
- Instead we will need to work with *indirect* knowledge of how the game turned out.
- We will should also hope that E is fairly representative of P

Building a checkers program [Mitchell, Chapter 1.2]

- Suppose we want to solve the following learning problem
  - Task T: playing checkers
  - Performance measure P: percent of games won in the world tournament
  - Training experience E: games played against itself
- Since we don't have complete knowledge of what the optimal moves are in checkers, providing suitable *direct* training examples would be difficult.
- Instead we will need to work with *indirect* knowledge of how the game turned out.
- We will should also hope that E is fairly representative of P
- Enumerating all legal moves from a given position is a straightforward (CS-122?) programming task, so we'll assume that's already done.

Building a checkers program [Mitchell, Chapter 1.2]

- Suppose we want to solve the following learning problem
  - Task T: playing checkers
  - Performance measure P: percent of games won in the world tournament
  - Training experience E: games played against itself
- Since we don't have complete knowledge of what the optimal moves are in checkers, providing suitable *direct* training examples would be difficult.
- Instead we will need to work with *indirect* knowledge of how the game turned out.
- We will should also hope that E is fairly representative of P
- Enumerating all legal moves from a given position is a straightforward (CS-122?) programming task, so we'll assume that's already done.
- So should we learn which move is the best - or how "good" a given board position is?

Building a checkers program [Mitchell, Chapter 1.2]

- Suppose we want to solve the following learning problem
  - Task T: playing checkers
  - Performance measure P: percent of games won in the world tournament
  - Training experience E: games played against itself
- Since we don't have complete knowledge of what the optimal moves are in checkers, providing suitable *direct* training examples would be difficult.
- Instead we will need to work with *indirect* knowledge of how the game turned out.
- We will should also hope that E is fairly representative of P
- Enumerating all legal moves from a given position is a straightforward (CS-122?) programming task, so we'll assume that's already done.
- So should we learn which move is the best - or how "good" a given board position is?
  - It's a design decision. We'll do the later of these two

Building a checkers program [Mitchell, Chapter 1.2]

- Suppose we want to solve the following learning problem
  - Task T: playing checkers
  - Performance measure P: percent of games won in the world tournament
  - Training experience E: games played against itself
- Since we don't have complete knowledge of what the optimal moves are in checkers, providing suitable *direct* training examples would be difficult.
- Instead we will need to work with *indirect* knowledge of how the game turned out.
- We will should also hope that E is fairly representative of P
- Enumerating all legal moves from a given position is a straightforward (CS-122?) programming task, so we'll assume that's already done.
- So should we learn which move is the best - or how "good" a given board position is?
  - It's a design decision. We'll do the later of these two

Review of set and function theory

- A *set* is a collection of *elements* (integers, program objects, mathematical objects, anything really)

Review of set and function theory

- A *set* is a collection of *elements* (integers, program objects, mathematical objects, anything really)
- Notation for functions

Review of set and function theory

- A *set* is a collection of *elements* (integers, program objects, mathematical objects, anything really)
- Notation for functions
  - $f : D \rightarrow R$

Review of set and function theory

- A *set* is a collection of *elements* (integers, program objects, mathematical objects, anything really)
- Notation for functions
  - $f : D \to R$
  - D and R are sets

Review of set and function theory

- A *set* is a collection of *elements* (integers, program objects, mathematical objects, anything really)
- Notation for functions
  - $f : D \to R$
  - D and R are sets
  - This means f is a function that takes an element of D as input and outputs an element of R as output

Review of set and function theory

- A *set* is a collection of *elements* (integers, program objects, mathematical objects, anything really)
- Notation for functions
  - $f : D \rightarrow R$
  - D and R are sets
  - This means f is a function that takes an element of D as input and outputs an element of R as output
  - Examples

Review of set and function theory

- A *set* is a collection of *elements* (integers, program objects, mathematical objects, anything really)
- Notation for functions
  - $f : D \rightarrow R$
  - D and R are sets
  - This means f is a function that takes an element of D as input and outputs an element of R as output
  - Examples
- So in this case, we would like to learn the *target function* $V : B \rightarrow \mathbb{R}$, where:

Review of set and function theory

- A *set* is a collection of *elements* (integers, program objects, mathematical objects, anything really)
- Notation for functions
  - $f : D \rightarrow R$
  - D and R are sets
  - This means f is a function that takes an element of D as input and outputs an element of R as output
  - Examples
- So in this case, we would like to learn the *target function* $V : B \rightarrow \mathbb{R}$, where:
  - V is the name we are giving to this function

Review of set and function theory

- A *set* is a collection of *elements* (integers, program objects, mathematical objects, anything really)
- Notation for functions
  - $f : D \rightarrow R$
  - D and R are sets
  - This means f is a function that takes an element of D as input and outputs an element of R as output
  - Examples
- So in this case, we would like to learn the *target function* $V : B \rightarrow \mathbb{R}$, where:
  - V is the name we are giving to this function
  - B is the set of all possible checker board states

Review of set and function theory

- A *set* is a collection of *elements* (integers, program objects, mathematical objects, anything really)
- Notation for functions
  - $f : D \to R$
  - D and R are sets
  - This means f is a function that takes an element of D as input and outputs an element of R as output
  - Examples
- So in this case, we would like to learn the *target function* $V : B \to \mathbb{R}$, where:
  - V is the name we are giving to this function
  - B is the set of all possible checker board states
  - $\mathbb{R}$ is the real numbers

Review of set and function theory

- A *set* is a collection of *elements* (integers, program objects, mathematical objects, anything really)
- Notation for functions
  - $f : D \rightarrow R$
  - D and R are sets
  - This means f is a function that takes an element of D as input and outputs an element of R as output
  - Examples
- So in this case, we would like to learn the *target function* $V : B \rightarrow \mathbb{R}$, where:
  - V is the name we are giving to this function
  - B is the set of all possible checker board states
  - $\mathbb{R}$ is the real numbers
  - $V(b) = 100$ if black can guarantee a win starting from board b

Review of set and function theory

- A *set* is a collection of *elements* (integers, program objects, mathematical objects, anything really)
- Notation for functions
  - $f : D \to R$
  - D and R are sets
  - This means f is a function that takes an element of D as input and outputs an element of R as output
  - Examples
- So in this case, we would like to learn the *target function* $V : B \to \mathbb{R}$, where:
  - V is the name we are giving to this function
  - B is the set of all possible checker board states
  - $\mathbb{R}$ is the real numbers
  - $V(b) = 100$ if black can guarantee a win starting from board b
  - $V(b) = -100$ if red can guarantee a win starting from board b

Review of set and function theory

- A *set* is a collection of *elements* (integers, program objects, mathematical objects, anything really)
- Notation for functions
  - $f : D \rightarrow R$
  - D and R are sets
  - This means f is a function that takes an element of D as input and outputs an element of R as output
  - Examples
- So in this case, we would like to learn the *target function* $V : B \rightarrow \mathbb{R}$, where:
  - V is the name we are giving to this function
  - B is the set of all possible checker board states
  - $\mathbb{R}$ is the real numbers
  - $V(b) = 100$ if black can guarantee a win starting from board b
  - $V(b) = -100$ if red can guarantee a win starting from board b
  - $V(b) = 0$ if the game is drawn

Review of set and function theory

- A *set* is a collection of *elements* (integers, program objects, mathematical objects, anything really)
- Notation for functions
  - $f : D \to R$
  - D and R are sets
  - This means f is a function that takes an element of D as input and outputs an element of R as output
  - Examples
- So in this case, we would like to learn the *target function* $V : B \to \mathbb{R}$, where:
  - V is the name we are giving to this function
  - B is the set of all possible checker board states
  - $\mathbb{R}$ is the real numbers
  - $V(b) = 100$ if black can guarantee a win starting from board b
  - $V(b) = -100$ if red can guarantee a win starting from board b
  - $V(b) = 0$ if the game is drawn
  - It would be difficult to calculate this function completely, so we will approximate it instead

Review of set and function theory

- A *set* is a collection of *elements* (integers, program objects, mathematical objects, anything really)
- Notation for functions
  - $f : D \to R$
  - D and R are sets
  - This means f is a function that takes an element of D as input and outputs an element of R as output
  - Examples
- So in this case, we would like to learn the *target function* $V : B \to \mathbb{R}$, where:
  - V is the name we are giving to this function
  - B is the set of all possible checker board states
  - $\mathbb{R}$ is the real numbers
  - $V(b) = 100$ if black can guarantee a win starting from board b
  - $V(b) = -100$ if red can guarantee a win starting from board b
  - $V(b) = 0$ if the game is drawn
  - It would be difficult to calculate this function completely, so we will approximate it instead

The target function

- $\hat{V}(b) = w_1x_1 + w_2x_2 + w_3x_3 + w_4x_4 + w_5x_5 + w_6x_6$

The target function

- $\hat{V}(b) = w_1x_1 + w_2x_2 + w_3x_3 + w_4x_4 + w_5x_5 + w_6x_6$
  - Each x_i is for some attribute of the board state (i.e. how many red pieces)

The target function

- $\hat{V}(b) = w_1 x_1 + w_2 x_2 + w_3 x_3 + w_4 x_4 + w_5 x_5 + w_6 x_6$
  - Each x_i is for some attribute of the board state (i.e. how many red pieces)
  - Each w_i is how much to weight that particular attribute

The target function

- $\hat{V}(b) = w_1 x_1 + w_2 x_2 + w_3 x_3 + w_4 x_4 + w_5 x_5 + w_6 x_6$
  - Each x_i is for some attribute of the board state (i.e. how many red pieces)
  - Each w_i is how much to weight that particular attribute
  - This is slightly different than the one in the book (no w_0)

The target function

- $\hat{V}(b) = w_1x_1 + w_2x_2 + w_3x_3 + w_4x_4 + w_5x_5 + w_6x_6$
  - Each x_i is for some attribute of the board state (i.e. how many red pieces)
  - Each w_i is how much to weight that particular attribute
  - This is slightly different than the one in the book (no w_0)
  - What are the best settings for the w_i weights?

The target function

- $\hat{V}(b) = w_1x_1 + w_2x_2 + w_3x_3 + w_4x_4 + w_5x_5 + w_6x_6$
  - Each x_i is for some attribute of the board state (i.e. how many red pieces)
  - Each w_i is how much to weight that particular attribute
  - This is slightly different than the one in the book (no w_0)
  - What are the best settings for the w_i weights?
- A particular setting of the w_i's is called a *hypothesis*

The target function

- $\hat{V}(b) = w_1x_1 + w_2x_2 + w_3x_3 + w_4x_4 + w_5x_5 + w_6x_6$
  - Each x_i is for some attribute of the board state (i.e. how many red pieces)
  - Each w_i is how much to weight that particular attribute
  - This is slightly different than the one in the book (no w_0)
  - What are the best settings for the w_i weights?
- A particular setting of the w_i's is called a *hypothesis*
- The set of all possible settings for all weight assignments w_i is called the *hypothesis space*

The target function

- $\hat{V}(b) = w_1x_1 + w_2x_2 + w_3x_3 + w_4x_4 + w_5x_5 + w_6x_6$
  - Each x_i is for some attribute of the board state (i.e. how many red pieces)
  - Each w_i is how much to weight that particular attribute
  - This is slightly different than the one in the book (no w_0)
  - What are the best settings for the w_i weights?
- A particular setting of the w_i's is called a *hypothesis*
- The set of all possible settings for all weight assignments w_i is called the *hypothesis space*
- We want to find a hypothesis so that $\hat{V}$ best approximates V

The target function

- $\hat{V}(b) = w_1x_1 + w_2x_2 + w_3x_3 + w_4x_4 + w_5x_5 + w_6x_6$
  - Each x_i is for some attribute of the board state (i.e. how many red pieces)
  - Each w_i is how much to weight that particular attribute
  - This is slightly different than the one in the book (no w_0)
  - What are the best settings for the w_i weights?
- A particular setting of the w_i's is called a *hypothesis*
- The set of all possible settings for all weight assignments w_i is called the *hypothesis space*
- We want to find a hypothesis so that $\hat{V}$ best approximates V

Training experience

- We would like to get training example pairs $(b, V(b))$ of what the function "should be" for a given position

Training experience

- We would like to get training example pairs $(b, V(b))$ of what the function "should be" for a given position
  - That's hard!

Training experience

- We would like to get training example pairs $(b, V(b))$ of what the function "should be" for a given position
  - That's hard!
- It turns out that training examples of the form $(b, \hat{V}(nextMove(b))$ do a pretty good job

Training experience

- We would like to get training example pairs $(b, V(b))$ of what the function "should be" for a given position
  - That's hard!
- It turns out that training examples of the form $(b, \hat{V}(nextMove(b))$ do a pretty good job
  - We are really using the function to train itself (think about this)!

Training experience

- We would like to get training example pairs $(b, V(b))$ of what the function "should be" for a given position
  - That's hard!
- It turns out that training examples of the form $(b, \hat{V}(nextMove(b))$ do a pretty good job
  - We are really using the function to train itself (think about this)!

Updating the weights

- To update the weights, we look at the discrepancy between the training example and the function

Updating the weights

- To update the weights, we look at the discrepancy between the training example and the function
- We adjust the weights w_i in the “right direction” to make up for the discrepancy...

Updating the weights

- To update the weights, we look at the discrepancy between the training example and the function
- We adjust the weights w_i in the "right direction" to make up for the discrepancy...
  - ...but only a little bit

Updating the weights

- To update the weights, we look at the discrepancy between the training example and the function
- We adjust the weights w_i in the "right direction" to make up for the discrepancy...
  - ...but only a little bit
- This is essentially a gradient descent search

Updating the weights

- To update the weights, we look at the discrepancy between the training example and the function
- We adjust the weights w_i in the "right direction" to make up for the discrepancy...
  - ...but only a little bit
- This is essentially a gradient descent search

Four main components of the agent

- Performance system

Four main components of the agent

- Performance system
  - Uses the learned target function to solve the problem at hand.

Four main components of the agent

- Performance system
  - Uses the learned target function to solve the problem at hand.
- Critic

Four main components of the agent

- Performance system
  - Uses the learned target function to solve the problem at hand.
- Critic
  - Produces training examples after an experiment

Four main components of the agent

- Performance system
  - Uses the learned target function to solve the problem at hand.
- Critic
  - Produces training examples after an experiment
- Generalizer

Four main components of the agent

- Performance system
  - Uses the learned target function to solve the problem at hand.
- Critic
  - Produces training examples after an experiment
- Generalizer
  - Updates the learned target function using the critic's training examples

Four main components of the agent

- Performance system
  - Uses the learned target function to solve the problem at hand.
- Critic
  - Produces training examples after an experiment
- Generalizer
  - Updates the learned target function using the critic's training examples
- Experiment generator

Four main components of the agent

- Performance system
  - Uses the learned target function to solve the problem at hand.
- Critic
  - Produces training examples after an experiment
- Generalizer
  - Updates the learned target function using the critic's training examples
- Experiment generator
  - Creates an experiment to start the whole process over again

Four main components of the agent

- Performance system
  - Uses the learned target function to solve the problem at hand.
- Critic
  - Produces training examples after an experiment
- Generalizer
  - Updates the learned target function using the critic's training examples
- Experiment generator
  - Creates an experiment to start the whole process over again