TDT4173 Machine Learning and Case-Based Reasoning

Lecture 1 – Introduction

Norwegian University of Science and Technology

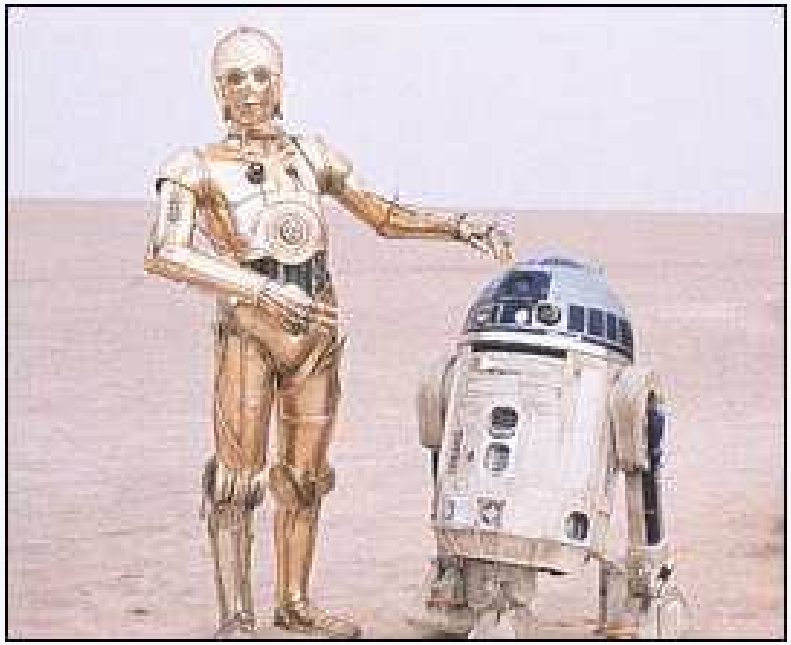
Helge Langseth and Anders Kofod-Petersen

NTNU

Outline

The grand vision

An autonomous self-moving machine that acts, reasons, and learnslike a human



We are still very far from achieving this. . .

Why Machine Learning

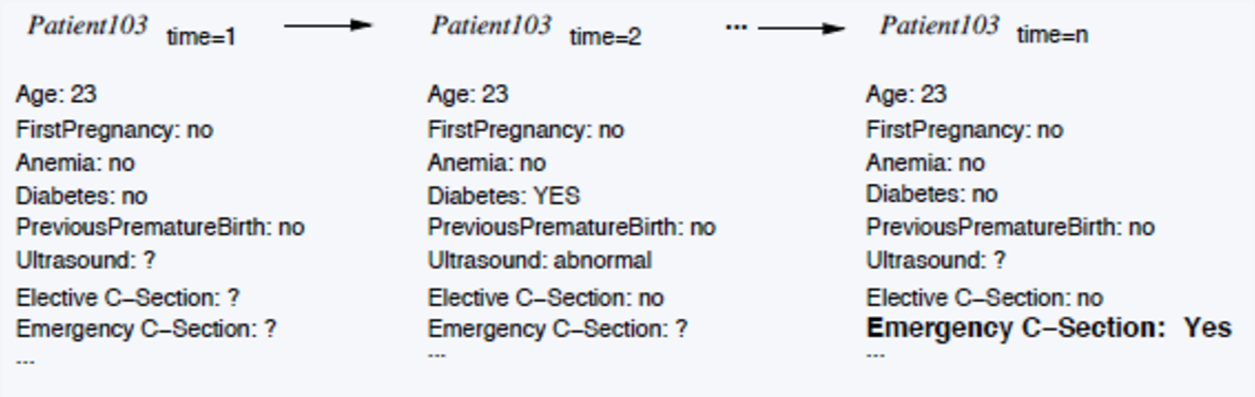
- Recent progress in algorithms and theory
- Growing flood of online data
- Computational power is available
- Budding industry

Three niches for machine learning:

- Data mining: using historical data to improve decisions
  - medical records $\rightarrow$ medical knowledge
- Software applications we can't program by hand
  - autonomous driving
  - speech recognition
- Self customizing programs
  - Recommendation systems

Typical Datamining Task

Data:



Given:

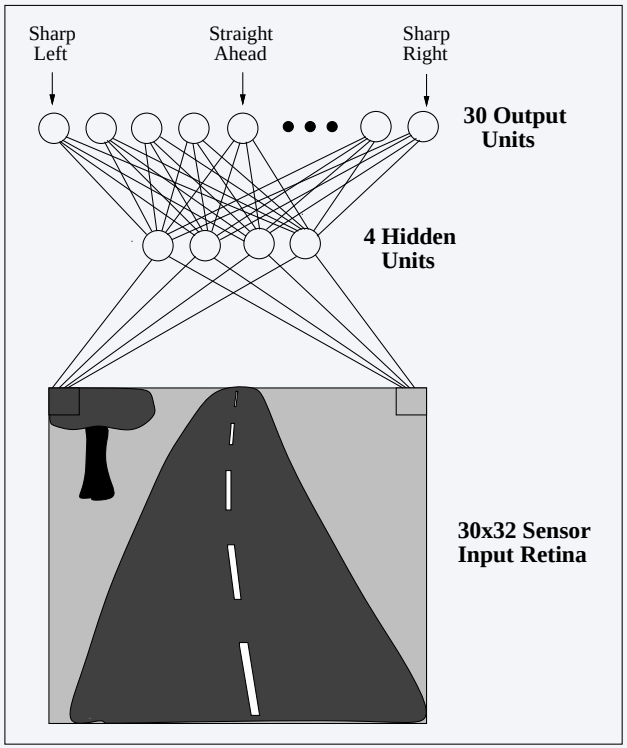
- 9714 patient records, each describing a pregnancy and birth
- Each patient record contains 215 features

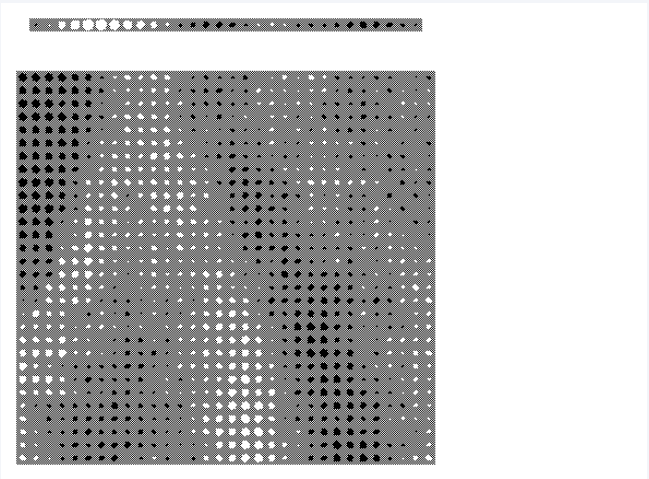
Learn to predict:

- Classes of future patients at high risk for Emergency Cesarean Section

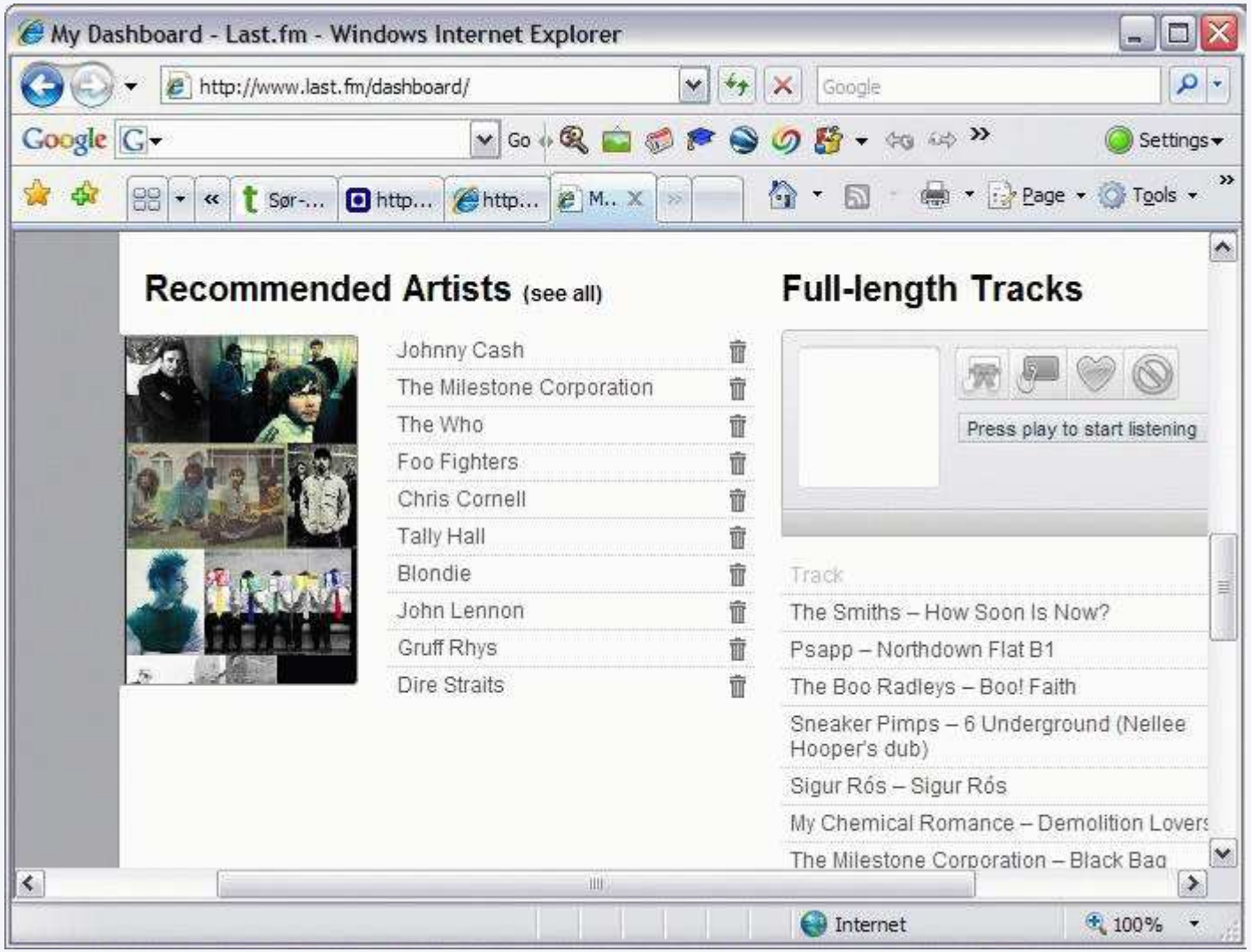
Problems Too Difficult to Program by Hand

ALVINN [Pomerleau] drives 70 mph on highways





Software that Customizes to User



http://www.last.fm

Where Is this Headed?

Today: tip of the iceberg:

- First-generation algorithms: neural nets, decision trees, regression ...
- Applied to well-formatted database
- Budding industry

Opportunity for tomorrow: enormous impact:

- Learn across full mixed-media data
- Learn by active experimentation
- Cumulative, lifelong learning
- Programming languages with learning embedded?
- . . . etc. (Only your imagination limits this list!)

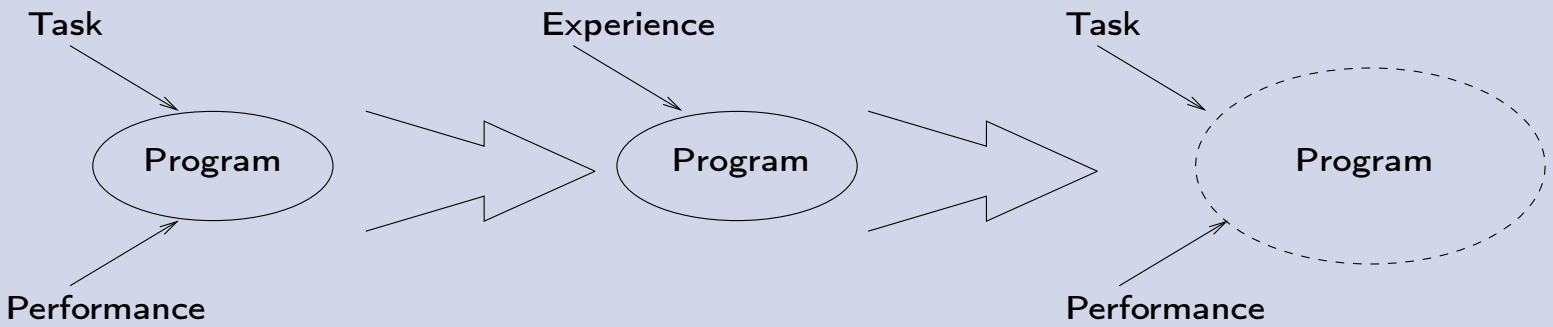
What is Machine Learning?

Learning = Improving with experience at some task

- Improve over task T,
- with respect to performance measure P,
- based on experience E.

Definition

Machine Learning Methods and techniques that makes computer systems able to update its knowledge and problem-solving ability.



TDT4173

Goals of the course:

The course will give a basic insight into principles and methods for how computer systems can learn from its own experience.

Syllabus:

- The text-book "Machine Learning" by Tom Mitchell.
- A number of papers top be decided and made available

How to get it...

Book available at Tapir. Papers will be made available for downloaded from our webpage

Exercises

- Designed to give hands-on experience with the different machine learning methods we talk about
- Will contain both coding tasks as well as requirements towards discussions
- Typically given with a one-and-a-half week deadline. Time of delivery Thursday at 20:00.
- 1 “big” and 4 “small” assignments.

NB! Counts towards final grade

All exercises count towards the final grade: If you fail one assignment, you will automatically take off 3.3% (6.7% for the big one) of the total available score.

Paper presentation

- A number of "classic texts"
- Papers to be presented by students
- Will make up the last part of the semester course

NB! Counts towards final grade

Each student must participate in presenting at least one paper. Otherwise, 3.3% of the total available score will be taken off.

Examination

- **December 6th**
- Written, 4 hours
- Counts for 80% of the final grade; the other 20% are determined by no. assignments handed in + participation in paper presentation.

Getting information

Sources for information:

- Check the web-page
  http://www.idi.ntnu.no/emner/tdt4173/
- It's Learning
- Ask (Contact info on web-page.)

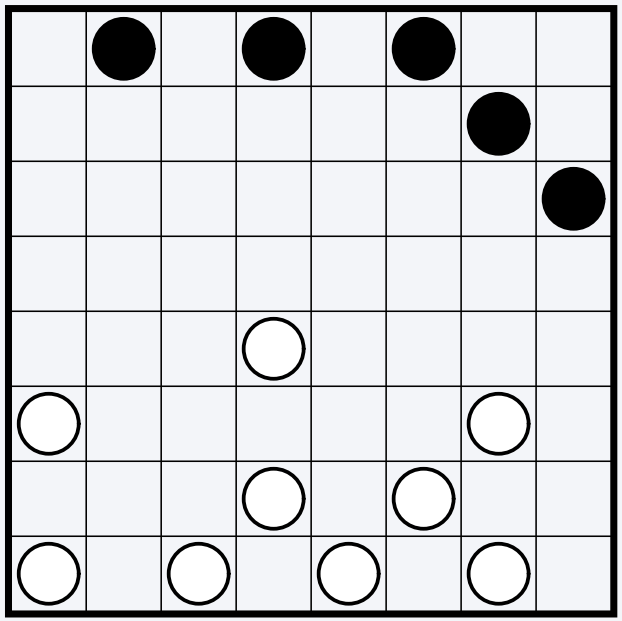
Reference group

If you want, we can have a reference group.
Not much work (if all goes well):

- Evaluation meeting(s)
- Evaluation report
- Students' spokesman if there is something I should take into account

It is not required, but if two students to volunteer to be in the ref.grp., it may smooth of any problems we meet as we move along.

Learning to Play Checkers



- T: Play checkers
- P: Percent of games won in world tournament
- E: opportunity to play against self

Design Choices

- What experience can we learn from?
- What exactly should be learned?
- How shall it be represented?
- Target function:
  - collection of rules?
  - neural network ?
  - polynomial function of board features?
  - ...
- What specific algorithm can we use to learn it?

Type of knowledge learned

We wish to learn a function that for any given board position $\mathcal{B}$ chooses the best move $\mathcal{M}$, `ChooseMove`$:\mathcal{B} \rightarrow \mathcal{M}$.

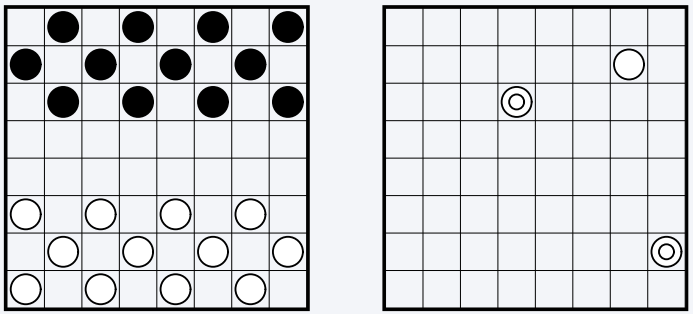
Type of knowledge learned

We wish to learn a function that for any given board position $\mathcal{B}$ chooses the best move $\mathcal{M}$, `ChooseMove`$:\mathcal{B} \rightarrow \mathcal{M}$.

Direct training: Examples of individual checkers board states and the correct move for each.

Indirect training: Examples of sequences of moves and final outcomes of the various games played.

Indirect training makes `ChooseMove` impractical to learn:



If we end up winning, is the first move then optimal?

Approximation – The start of the learning work

Instead of `ChooseMove`, we establish a *Value function* V:

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

that maps legal board states $\mathcal{B}$ into some real value.

Playing rule: For any board position, choose the move that maximizes the value of the resulting board position.

Approximation – The start of the learning work

Instead of `ChooseMove`, we establish a *Value function* V:

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

that maps legal board states $\mathcal{B}$ into some real value.

Playing rule: For any board position, choose the move that maximizes the value of the resulting board position.

1. if b is a final board state that is won, then $V(b) = 100$
2. if b is a final board state that is lost, then $V(b) = -100$
3. if b is a final board state that is drawn, then $V(b) = 0$
4. if b is a not a final state in the game, then $V(b) = ??$

Approximation – The start of the learning work

Instead of `ChooseMove`, we establish a *Value function* V:

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

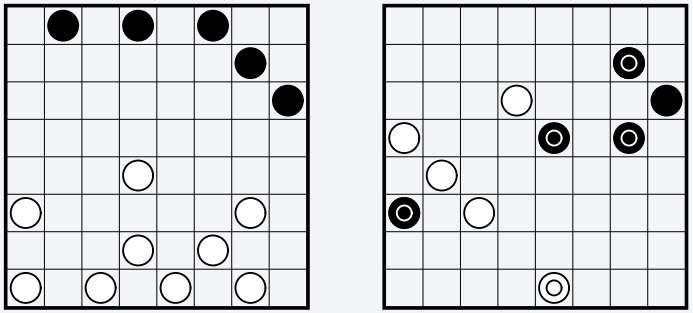
that maps legal board states $\mathcal{B}$ into some real value.

Playing rule: For any board position, choose the move that maximizes the value of the resulting board position.

1. if b is a final board state that is won, then $V(b) = 100$
2. if b is a final board state that is lost, then $V(b) = -100$
3. if b is a final board state that is drawn, then $V(b) = 0$
4. if b is a not a final state in the game, then $V(b) = V(b')$, b' is the best final board state that can be achieved starting from b and playing optimally until the end of the game.

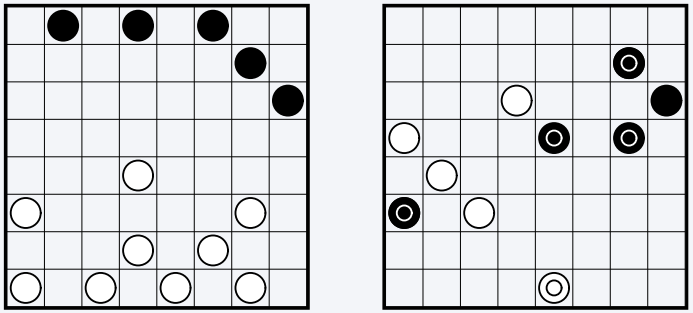
It is still not trivial...

What is of importance?



x_1: # black pieces
x_2: # white pieces
x_3: # black kings
x_4: # white kings
x_5: # white pieces threatened
x_6: # black pieces threatened

What is of importance?



x_1: # black pieces
x_2: # white pieces
x_3: # black kings
x_4: # white kings
x_5: # white pieces threatened
x_6: # black pieces threatened

Approximation:

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

where w_i is the weight assigned to x_i.

Learning task:

Determine the weights w_0, w_1, w_2, w_3, w_4, w_5, and w_6.

How to learn

In order to learn $\hat{V}$, we require a set of training examples,each describing a board state b and a training value V_{train} for b:

- Current weights: w_0, w_1, w_2, w_3, w_4, w_5, and w_6 yield $\hat{V}$.
- New game: $b_1, b_2, \ldots, b_{\text{end}}$

What value should $V_{\text{train}}(b_i)$ should we attach to position b_i?

How to learn

In order to learn $\hat{V}$, we require a set of training examples,each describing a board state b and a training value V_{train} for b:

- Current weights: w_0, w_1, w_2, w_3, w_4, w_5, and w_6 yield $\hat{V}$.
- New game: $b_1, b_2, \ldots, b_{\text{end}}$

What value should $V_{\text{train}}(b_i)$ should we attach to position b_i?

Idea:
When faced with a situation b_k, both players do the best they can – resulting in b_{end}.

How to learn

In order to learn $\hat{V}$, we require a set of training examples,each describing a board state b and a training value V_{train} for b:

- Current weights: w_0, w_1, w_2, w_3, w_4, w_5, and w_6 yield $\hat{V}$.
- New game: $b_1, b_2, \ldots, b_{\text{end}}$

What value should $V_{\text{train}}(b_i)$ should we attach to position b_i?

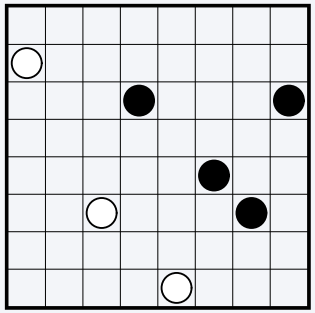
Idea:
When faced with a situation b_k, both players do the best they can – resulting in b_{end}.

In general:

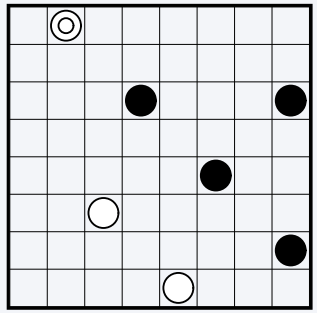
$$V_{\text{train}}(b_i) \leftarrow \hat{V}(b_{i+1})$$

This makes sense if $\hat{V}$ is more accurate for board states closer to the end of the game.

Ehhh. . . And what does this mean?



Current state: b_i



Next state: b_{i+1}

- The (system believes that) situation $b_i \rightsquigarrow b_{i+1}$
- Therefore $V_{\text{train}}(b_i) = V(b_{i+1})$
- $V(b_{i+1})$ is unknown, but assuming the system is very good, we have $\hat{V}(b_{i+1}) \approx V(b_{i+1})$. Thus, we decide that

$$V_{\text{train}}(b_i) \leftarrow \hat{V}(b_{i+1}).$$

And will it work?

Can we get reasonable training data?

- We know what $V(b_{\text{end}})$ is for any state b_{end}
- Using the previous setup, we should therefore be able to value situations that are *one*step away from being finished!
- . . . and using the same setup again, we should next be able to value situations that are *two*steps away from being finished
- . . . and so on

This should work, but we need to be able to ***use*** the training data. . .

How to learn the weights

- Current weights:w_0, w_1, w_2, w_3, w_4, w_5, and w_6 yield $\hat{V}$.
- New game:$< b_1, V_{\text{train}}(b_1) >, \ldots, < b_{\text{end}}, V_{\text{train}}(b_{\text{end}}) >$.

How to learn the weights

- Current weights: w_0, w_1, w_2, w_3, w_4, w_5, and w_6 yield $\hat{V}$.
- New game: $< b_1, V_{\text{train}}(b_1) >, \ldots, < b_{\text{end}}, V_{\text{train}}(b_{\text{end}}) >$.

Idea: Introduce *error function* E, and change weights such that the total error over all training examples is minimal.

$$E = \sum_{<b, V_{\text{train}}(b)> \in \text{ training examples}} \left(V_{\text{train}}(b) - \hat{V}(b) \right)^2$$

Note:
E is a function of the weights, $E = E(w_0, w_1, w_2, w_3, w_4, w_5, w_6)$, and we will change the weights to make E obtain its minimal value.

LMS Weight update rule

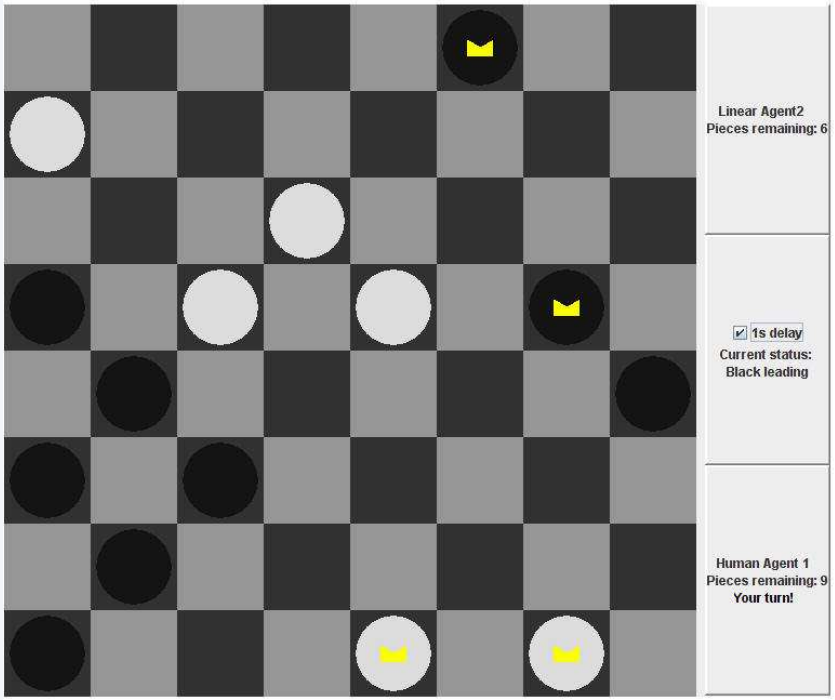
For each training set $< b, V_{\text{train}}(b) >$ do:

- Use the current weights to calculate $\hat{V}(b)$.
- For each weight w_i do:

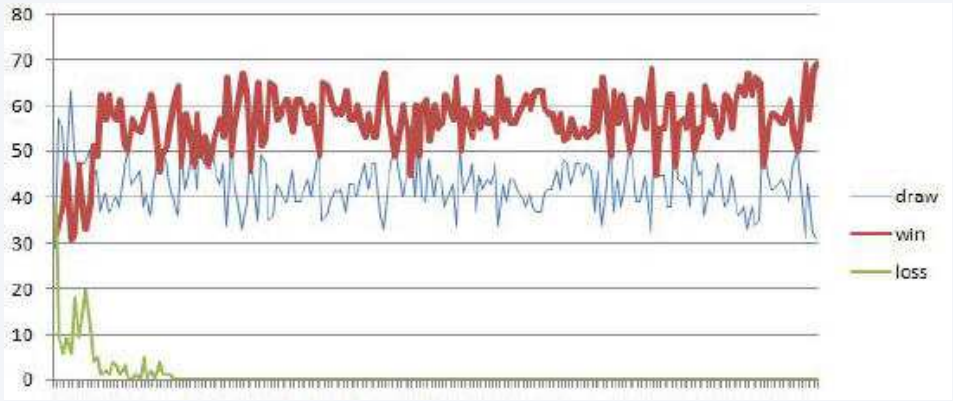
$$w_i \leftarrow w_i + \mu \cdot x_i \cdot \left(V_{\text{train}}(b) - \hat{V}(b) \right)$$

where μ is the *learning rate*.

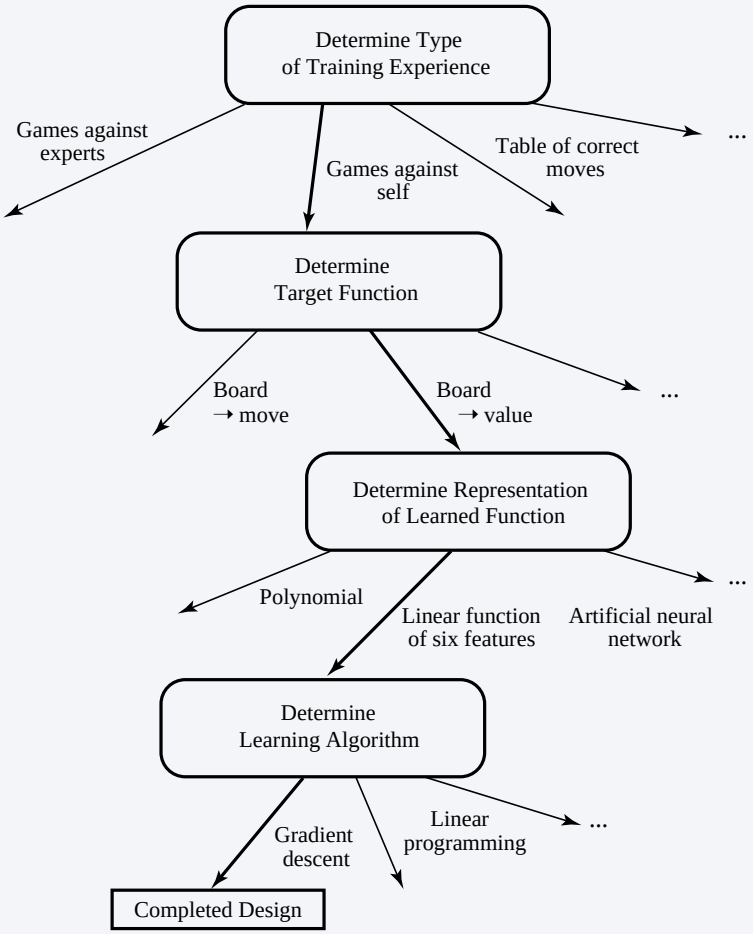
Implementation: A 5th year Student Project



Results



Design Choices



Training Examples for *EnjoySport*

Index	Sky	Temp	Humid	Wind	Water	Forecast	EnjoySport
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

What is the general concept?

Training Examples for *EnjoySport*

Index	Sky	Temp	Humid	Wind	Water	Forecast	EnjoySport
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

What is the general concept?

- "Sky = Sunny"?
- "Sky = Sunny **AND** Temp = Warm"?
- "Forecast = Same **OR** Water = Cool"?
- "When Index is written in binary digits it requires *one*1"?

Representing Hypotheses

Many possible representations

Here, h is conjunction of constraints on attributes

Each constraint can be

- a specific value (e.g., "$\texttt{Water} = \texttt{Warm}$")
- don't care (e.g., "$\texttt{Water} = ?$")
- no value allowed (e.g.,"$\texttt{Water} = \emptyset$")

For example,

Sky	AirTemp	Humid	Wind	Water	Forecast
$\langle$ Sunny	?	?	Strong	?	Same $\rangle$

Prototypical Concept Learning Task

Given:

- Instances X: Possible days, each described by the attributes `Sky, AirTemp, Humidity, Wind, Water, Forecast`
- Target function c: EnjoySport: $X \rightarrow \{0, 1\}$
- Hypotheses H: Conjunctions of literals, e.g., $\langle ?, \texttt{Cold}, \texttt{High}, ?, ?, ? \rangle$.
- Training examples D: Positive and negative examples of the target function $\langle x_1, c(x_1) \rangle, \ldots \langle x_m, c(x_m) \rangle$

Determine a hypothesis $h \in H$ such that $\forall x \in D : \ h(x) = c(x)$.

Prototypical Concept Learning Task

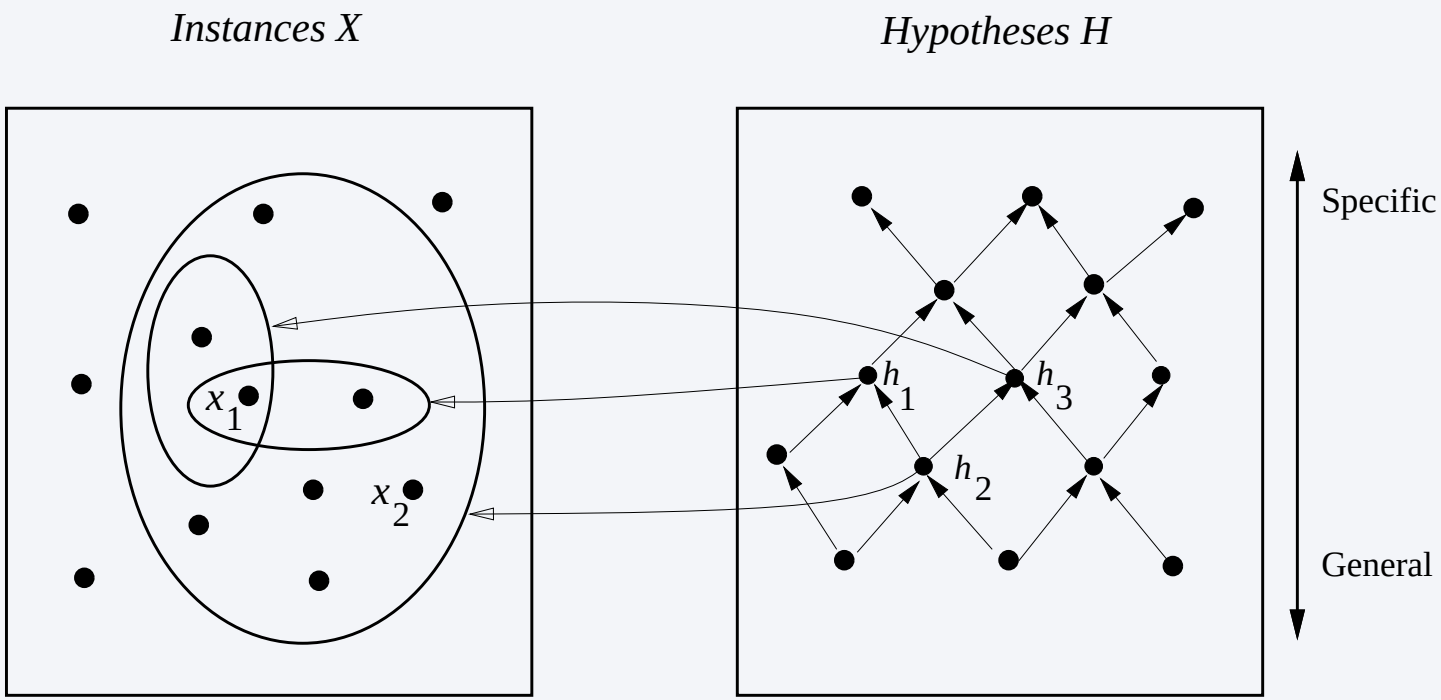
Given:

- Instances X: Possible days, each described by the attributes `Sky, AirTemp, Humidity, Wind, Water, Forecast`
- Target function c: EnjoySport: $X \rightarrow \{0, 1\}$
- Hypotheses H: Conjunctions of literals, e.g., $\langle ?, \texttt{Cold}, \texttt{High}, ?, ?, ? \rangle$.
- Training examples D: Positive and negative examples of the target function $\langle x_1, c(x_1) \rangle, \ldots \langle x_m, c(x_m) \rangle$

Determine a hypothesis $h \in H$ such that $\forall x \in D : \ h(x) = c(x)$.

The inductive learning hypothesis: Any hypothesis found to approximate the target function well over a sufficiently large set of training examples will also approximate the target function well over other unobserved examples.

Instance, Hypotheses, and More-General-Than



x_1 = <Sunny, Warm, High, Strong, Cool, Same>

x_2 = <Sunny, Warm, High, Light, Warm, Same>

h_1 = <Sunny, ?, ?, Strong, ?, ?>

h_2 = <Sunny, ?, ?, ?, ?, ?>

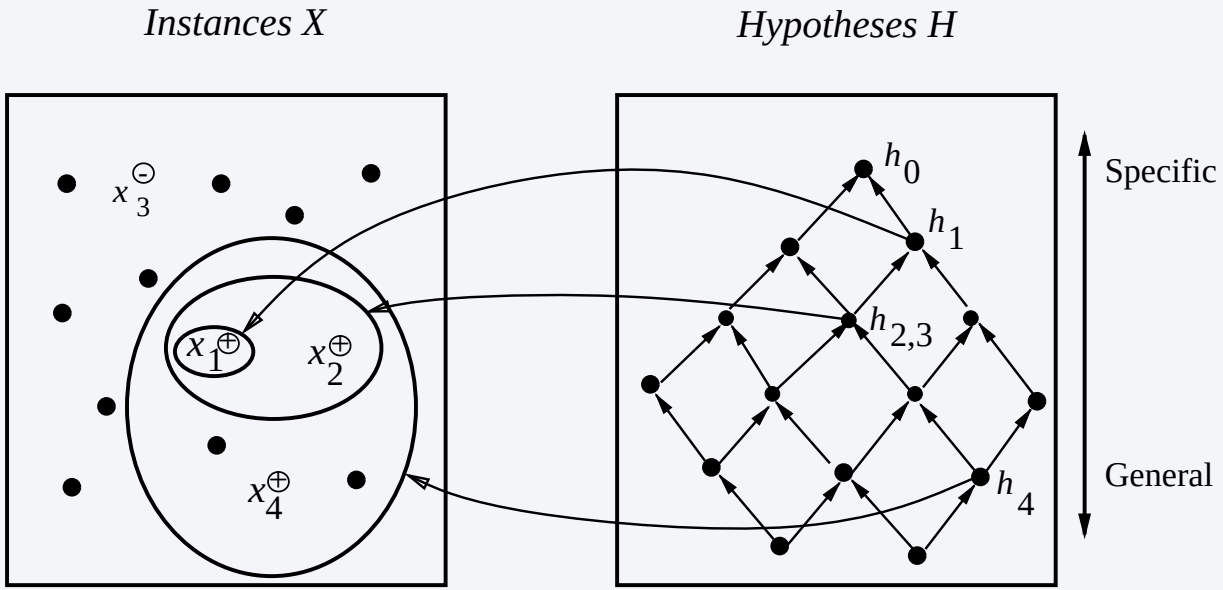
h_3 = <Sunny, ?, ?, ?, Cool, ?>

Find-S Algorithm

1. Initialize h to the most specific hypothesis in H
2. For each positive training instance x
   - For each attribute constraint a_i in h

     If a_i in h is satisfied by x Then
       do nothing
     Else
       replace a_i in h by the next more general
         constraint that is satisfied by x
3. Output hypothesis h

Hypothesis Space Search by Find-S



x_1 = <*Sunny Warm Normal Strong Warm Same*>, +
x_2 = <*Sunny Warm High Strong Warm Same*>, +
x_3 = <*Rainy Cold High Strong Warm Change*>, -
x_4 = <*Sunny Warm High Strong Cool Change*>, +

h_0 = <∅, ∅, ∅, ∅, ∅, ∅>
h_1 = <*Sunny Warm Normal Strong Warm Same*>
h_2 = <*Sunny Warm ? Strong Warm Same*>
h_3 = <*Sunny Warm ? Strong Warm Same*>
h_4 = <*Sunny Warm ? Strong ? ?* >

Complaints about Find-S

- Can't tell whether it has learned concept
- Can't tell when training data inconsistent
- Picks a maximally specific h (why?)
- Depending on H, there might be several!

Version Spaces

A hypothesis h is **consistent** *with a set of training examples D of target concept c if and only if $h(x) = c(x)$ for each training example $\langle x, c(x)\rangle$ in D.*

$$\mathit{Consistent}(h, D) \equiv (\forall \langle x, c(x)\rangle \in D)\ h(x) = c(x)$$

The **version space**, *$VS_{H,D}$, with respect to hypothesis space H and training examples D, is the subset of hypotheses from H consistent with all training examples in D.*

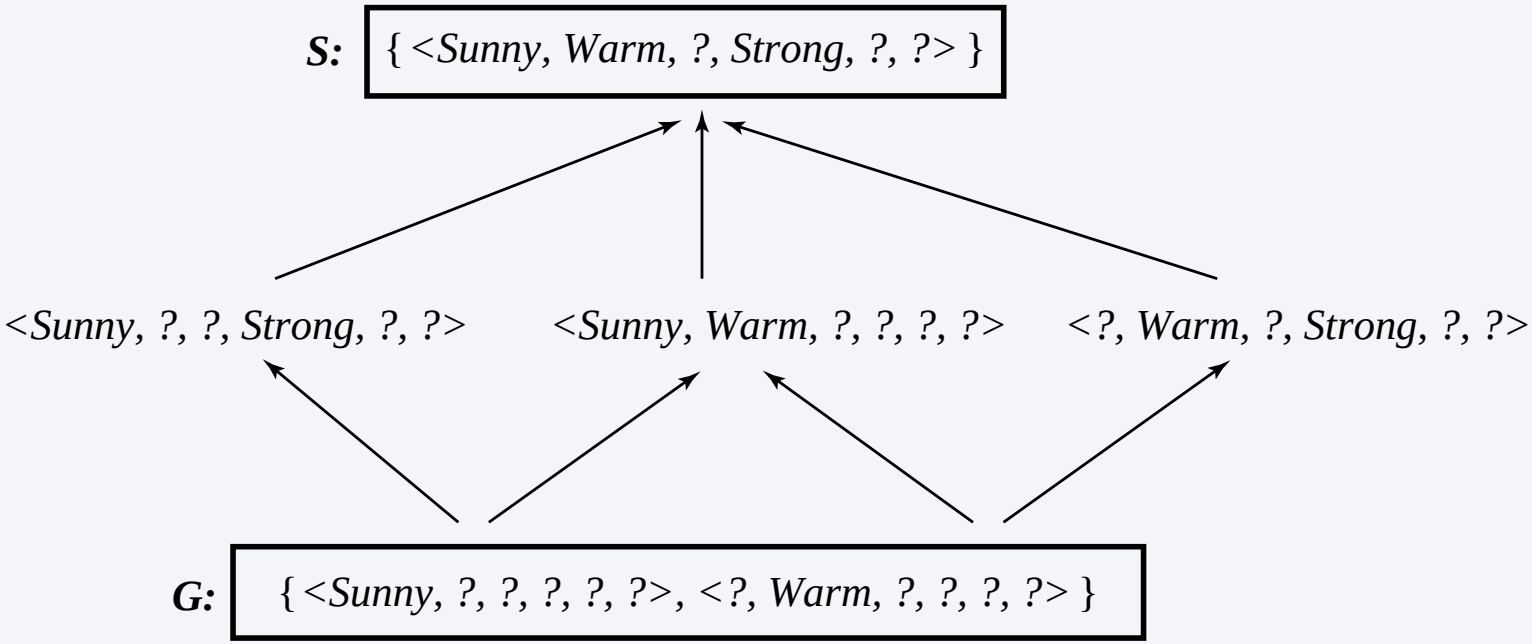
$$VS_{H,D} \equiv \{h \in H | \mathit{Consistent}(h, D)\}$$

The List-Then-Eliminate Algorithm

1. `VersionSpace` $\leftarrow$ a list containing every hypothesis in H
2. For each training example, $\langle x, c(x) \rangle$

   remove from `VersionSpace` any hypothesis h for which $h(x) \neq c(x)$
3. Output the list of hypotheses in `VersionSpace`

Example Version Space



Representing Version Spaces

- The **General boundary**, G, of version space $VS_{H,D}$ is the set of its maximally general members
- The **Specific boundary**, S, of version space $VS_{H,D}$ is the set of its maximally specific members
- Every member of the version space lies between these boundaries

$$VS_{H,D} = \{h \in H | (\exists s \in S)(\exists g \in G)(g \geq h \geq s)\}$$

where $x \geq y$ means x is more general or equal to y

Candidate Elimination Algorithm

$G \leftarrow$ maximally general hypotheses in H
$S \leftarrow$ maximally specific hypotheses in H
For each training example d, do

- If d is a positive example
  - Remove from G any hypothesis inconsistent with d
  - For each hypothesis s in S that is not consistent with d
    - Remove s from S
    - Add to S all minimal *generalizations* h of s such that
      - h is consistent with d, and
      - some member of G is more *general* than h
    - Remove from S any hypothesis that is more general than another hypothesis in S

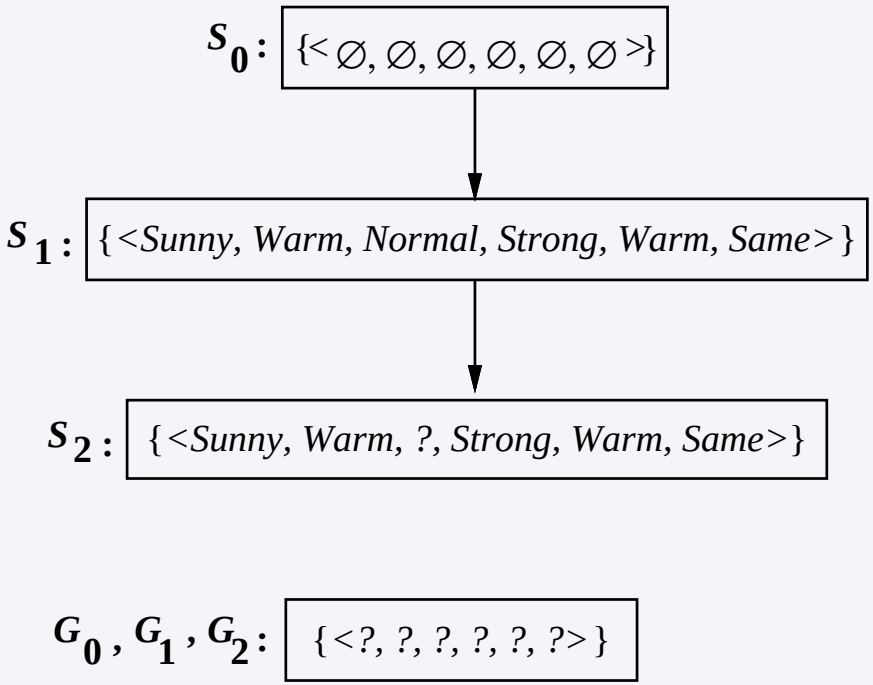
[CONT'd]

Candidate Elimination Algorithm

[. . . FROM PREVIOUS SLIDE]

- If d is a negative example
  - Remove from S any hypothesis inconsistent with d
  - For each hypothesis g in G that is not consistent with d
    - Remove g from G
    - Add to G all minimal *specializations* h of g such that
      - h is consistent with d, and - some member of S is more *specific* than h
    - Remove from G any hypothesis that is less general than another hypothesis in G

Example Trace

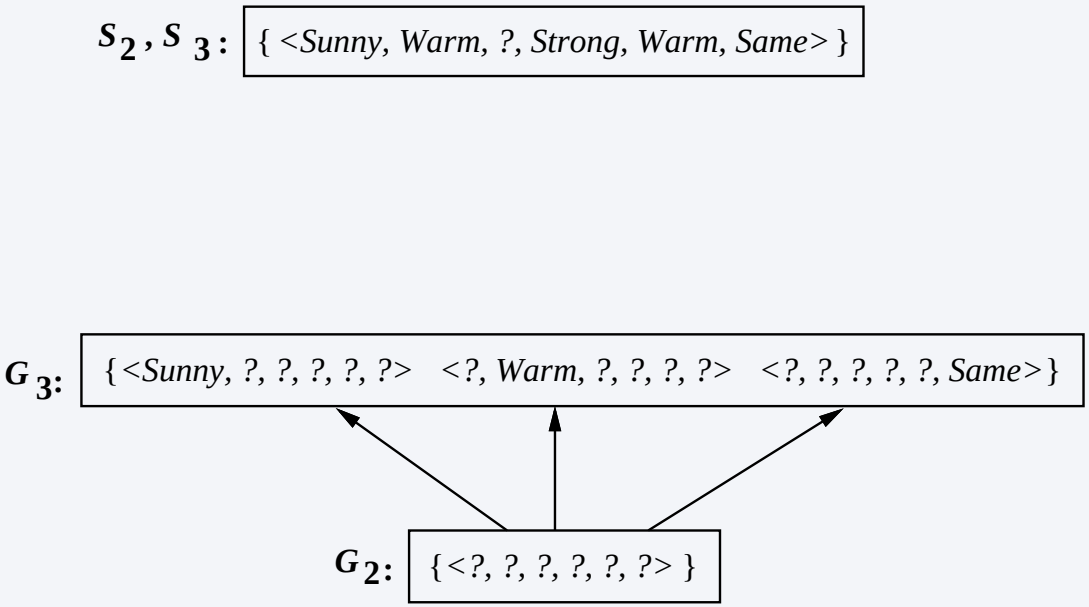


Training examples:

1 . <*Sunny, Warm, Normal, Strong, Warm, Same*>, *Enjoy Sport = Yes*

2 . <*Sunny, Warm, High, Strong, Warm, Same*>, *Enjoy Sport = Yes*

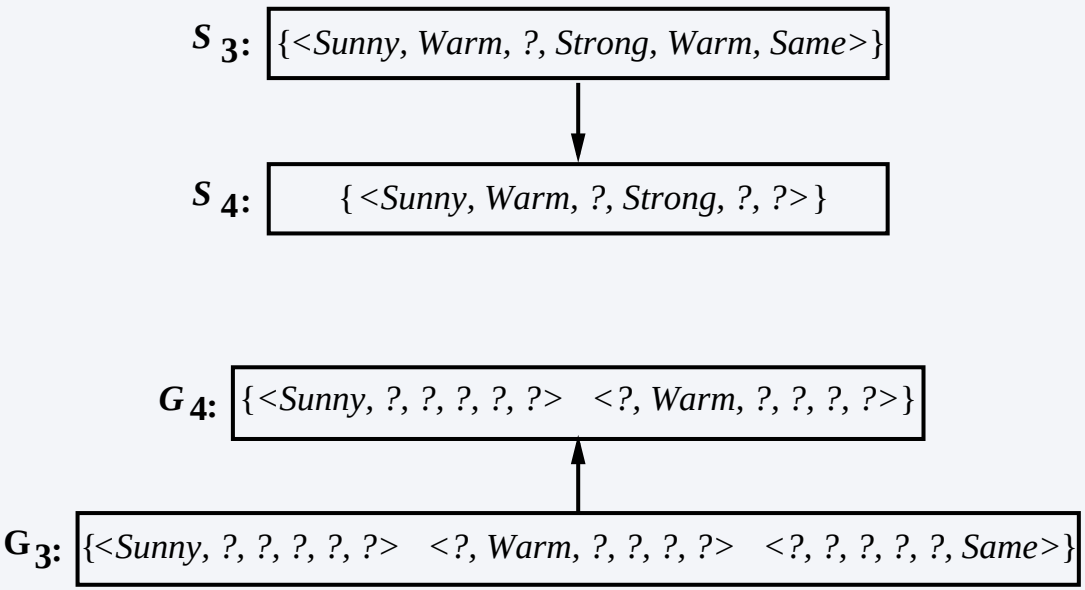
Example Trace



Training Example:

3. *<Rainy, Cold, High, Strong, Warm, Change>, EnjoySport=No*

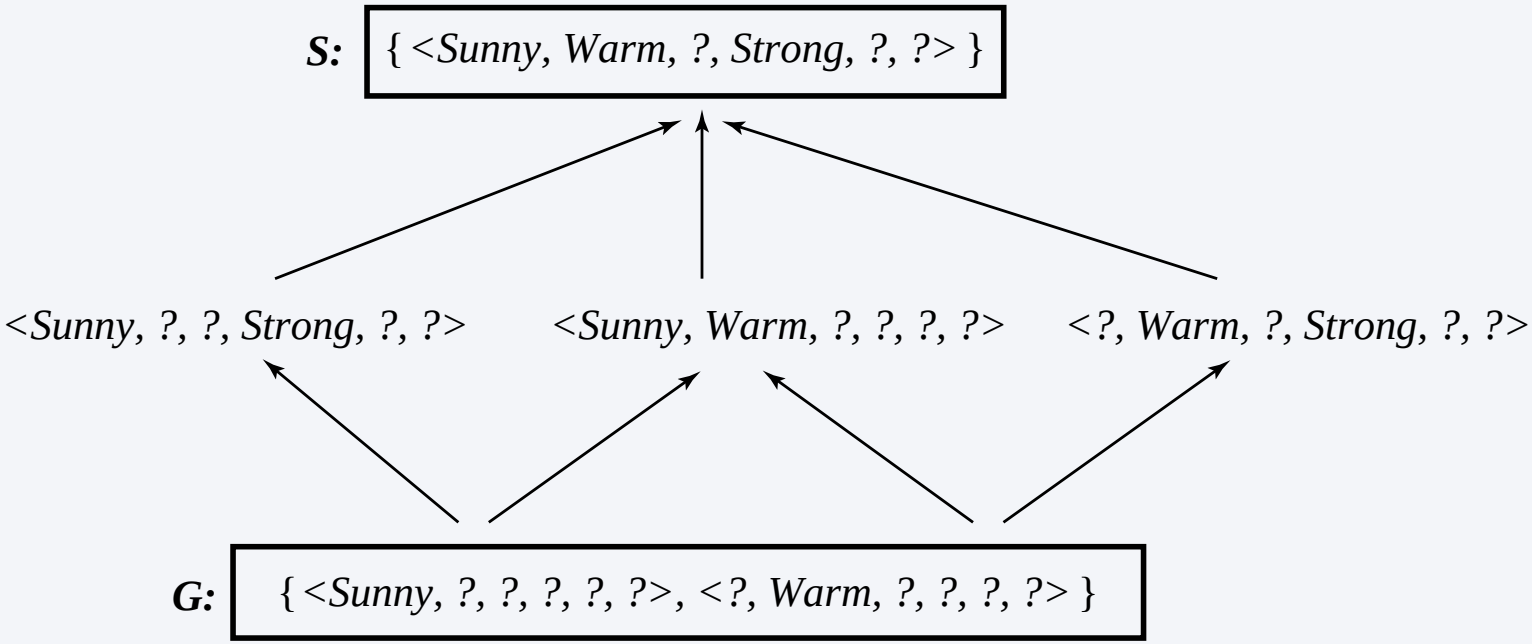
Example Trace



Training Example:

4. <*Sunny*, *Warm*, *High*, *Strong*, *Cool*, *Change*>, *EnjoySport* = *Yes*

How Should These Be Classified??



⟨Sunny Warm Normal Strong Cool Change⟩

⟨Rainy Cool Normal Light Warm Same⟩

⟨Sunny Warm Normal Light Warm Same⟩

What Justifies this Inductive Leap?

$+ \langle \texttt{Sunny Warm Normal Strong Cool Change} \rangle$

$+ \langle \texttt{Sunny Warm Normal Light Warm Same} \rangle$

$S: \langle \texttt{Sunny Warm Normal ? ? ?} \rangle$

Why believe we can classify the unseen

$\langle \texttt{Sunny Warm Normal Strong Warm Same} \rangle$

Inductive Bias

Consider

- concept learning algorithm L
- instances X, target concept c
- training examples $D_c = \{\langle x, c(x)\rangle\}$
- let $L(x_i, D_c)$ denote the classification assigned to the instance x_i by L after training on data D_c.

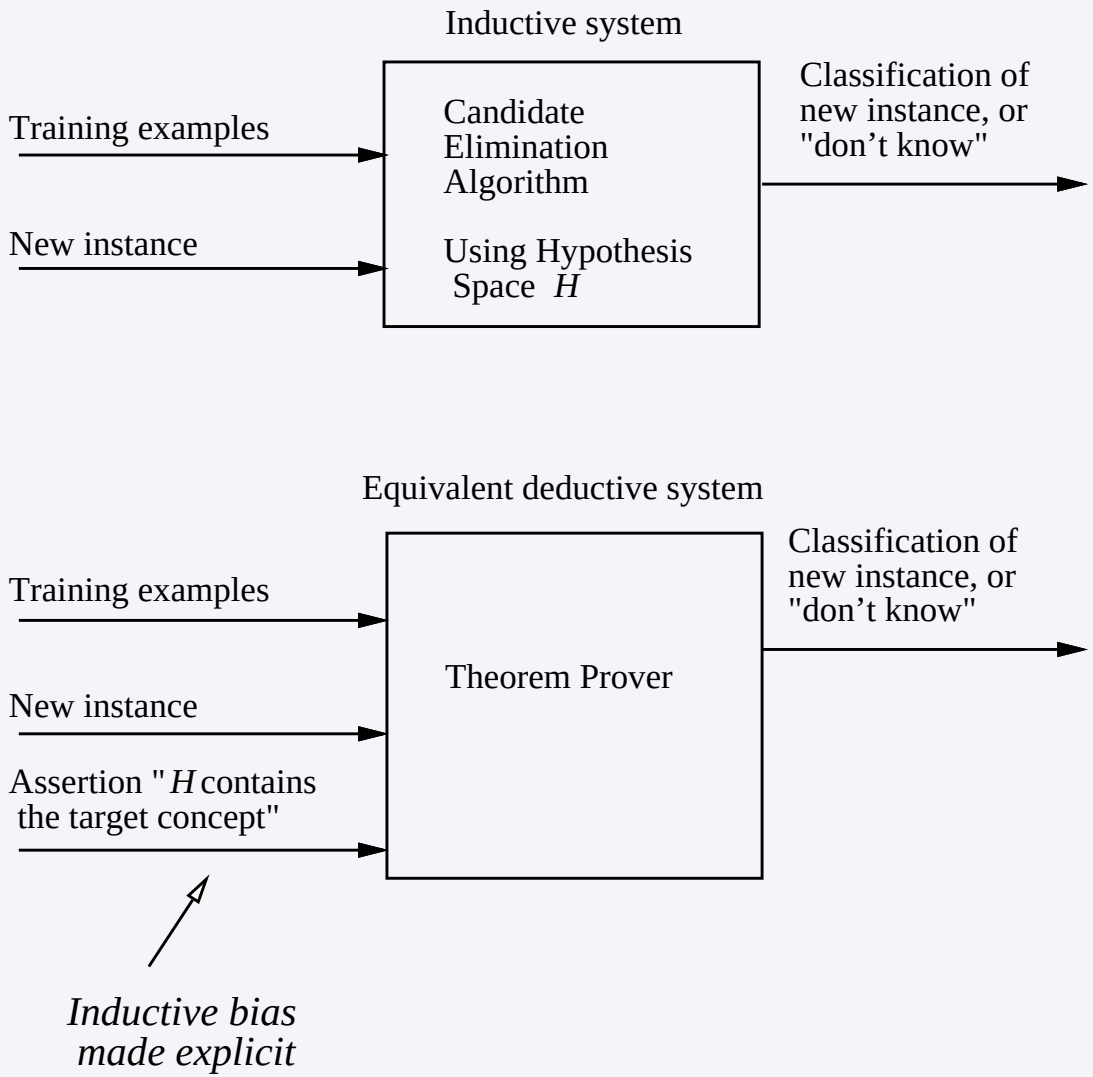
Definition:

The **inductive bias** *of L is any minimal set of assertions B such that for any target concept c and corresponding training examples D_c*

$$(\forall x_i \in X)[(B \wedge D_c \wedge x_i) \vdash L(x_i, D_c)]$$

where $A \vdash B$ means A logically entails B.

Inductive and Equivalent Deductive Systems



Three Learners with Different Biases

1. *Rote learner:* Store examples, Classify x iff it matches previously observed example.
2. *Version space candidate elimination algorithm*
3. *Find-S*

Summary Points

1. Concept learning as search through H
2. General-to-specific ordering over H
3. Version space candidate elimination algorithm
4. S and G boundaries characterize learner's uncertainty
5. Learner can generate useful queries
6. Inductive leaps possible only if learner is biased
7. Inductive learners can be modelled by equivalent deductive systems