Sapienza University of Rome

Master in Artificial Intelligence and Robotics
Master in Engineering in Computer Science

Machine Learning

A.Y. 2020/2021

Prof. L. Iocchi, F. Patrizi

1. Introduction

L. Iocchi, F. Patrizi

Overview

- What is a Machine Learning problem
- Example: learning to play checkers
- Machine Learning problem formulations
- Learning as search in hyphotesis space
- Concept learning
- Machine Learning issues

References
T. Mitchell. Machine Learning. Chapters 1, 2

Machine Learning

Machine learning is programming computers to improve a performance criterion using example data or past experience.

Machine learning (or data mining) is the task of producing knowledge from data.

Machine Learning useful when

- Human expertise does not exist
- Humans are unable to explain their expertise
- Solution needs to be adapted to particular cases

Machine Learning

Machine Learning exploits

- Recent progress in algorithms and theory
- Growing flood of on-line data (Big Data)
- Increasing computational power (GPU)

What is a Learning Problem?

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

improving performance over time thanks to available data for ex.

Learning = Improving with experience at some task

- Improve over task T, we need to specify which is the task to solve
- with respect to performance measure P,
- based on experience E.

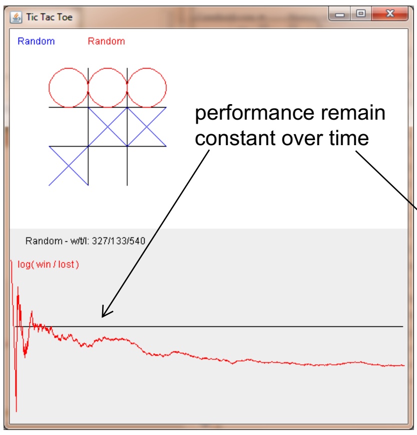
improving performance P with experience E at some task T

Improving performance over time

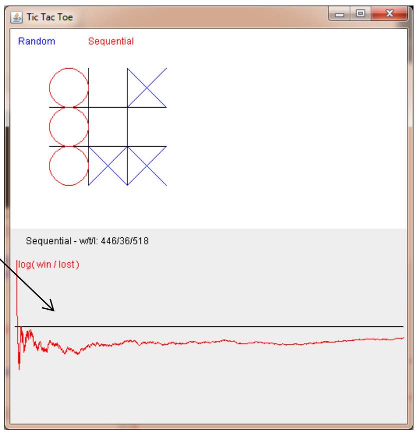
Example: Tic Tac Toe

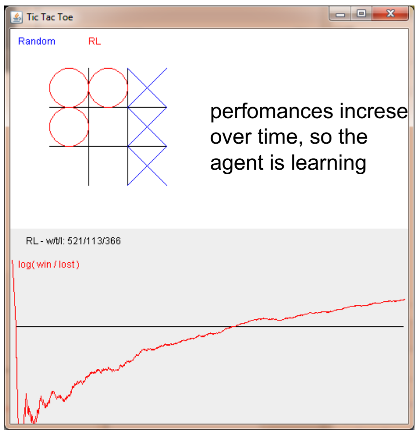
sarebbe tris

blue player chose randomly the position, while the red player uses firt a random strategy, then it uses a sequential strategy, and the finally uses a ML approach, in particular RL



performance remain constant over time





perfomances increse over time, so the agent is learning

the plot is the log(win/lost),

Example

Learn to play checkers

- T: Play checkers   the task is well defined, a game with specific rule
- P: % of games won in world tournament
- E: opportunity to play against self

Learning to Play Checkers

- What experience?
- What exactly should be learned?
- How shall it be represented?
- What specific algorithm to learn it?

Type of Training Experience

The first design choice we face is to choose the type of training experience from which our system will learn. The type of training experience available can have a significant impact on success or failure of the learner.

One key attribute is whether the training experience provides direct or indirect feedback regarding the choices made by the performance system. A second important attribute of the training experience is the degree to which the learner controls the sequence of training examples

- Human expert suggests optimal move for each configuration of the board
- Human expert evaluates each configuration of the board
- Computer plays against a human and automatically detects win/draw/loss configurations
- Computer plays against itself

A third important attribute of the training experience is how well it represents the distributionof examples over which the final system performance P must be measured. In general, learning is most reliable when the training examples follow a distribution similar to that of future test examples

A problem: is training experience representative of performance goal?

In our checkers Learning scenario, the performance metric P is the percent of games the system wins in the world tournament. If its training experience E consists only of games played against itself, there is an obvious danger that this training experience might not be fully representative of the distribution of situations over which it will later be tested

most current theory of machine learning rests on the crucial assumption that the distribution of training examples is identical to the distribution of test examples

Choose the Target Function

The next design choice is to determine exactly what type of knowledge will be learned and how this will be used by the performance program

Let us begin with a checkers-playing program that can generate the legal moves from any board state. The program needs only to learn how to choose the best move from among these legal moves. This learning task is representative of a large class of tasks for which the legal moves that define some large search space are known a priori, but for which the best search strategy is not known

given the configuration of the board, the function returns the move

- $ChooseMove : Board \to Move$
- $V : Board \to \Re$
- ...

to indicate that this function accepts as input any board from the set of legal board states Board and produces as output some move from the set of legal moves Move

Given this setting where we must learn to choose among the legal moves, the most obvious choice for the type of information to be learned is a program, or function

An alternative target function is an evaluation function that assigns a numerical score to any given board state. V maps any legal board state from the set B to some real value

Possible Definition for Target Function V

possible definition of value function V, in the previous slide

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

Because this definition is not efficiently computable by our checkers playing program, we say that it is a nonoperational definition

This gives correct values, but is not operational!

The goal of learning in this case is to discover an operational description of V ; that is, a description that can be used by the checkers-playing program to evaluate states and select moves within realistic time bounds

Choose Representation for Target Function

Now that we have specified the ideal target function V, we must choose a representation that the learning program will use to describe the funtion V that it will learn

- collection of rules
- neural network
- polynomial function of board features
- ...

A Representation for Learned Function

for any given board state, the function V will be calculated as a linear combination of the following board features

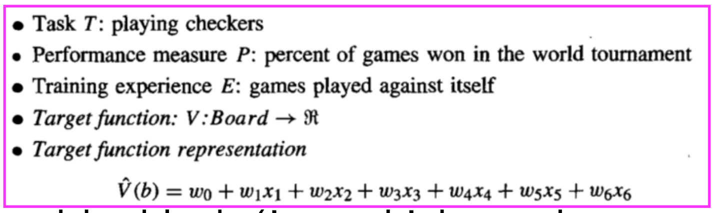
function is a linear combination of features

$$V(b) = w_0 + w_1 \cdot bp(b) + w_2 \cdot rp(b) + w_3 \cdot bk(b) + w_4 \cdot rk(b) + w_5 \cdot bt(b) + w_6 \cdot rt(b)$$

coeff of linear combination

features →

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



- Task T: playing checkers
- Performance measure P: percent of games won in the world tournament
- Training experience E: games played against itself
- *Target function:* $V : Board \rightarrow \Re$
- *Target function representation*

$$\hat{V}(b) = w_0 + w_1x_1 + w_2x_2 + w_3x_3 + w_4x_4 + w_5x_5 + w_6x_6$$

Note: Unknown w_i Learning $V \equiv$ estimating w_i

we need to estimate the weights of this linear combination, this is the meaning of learning V

Obtaining Training Examples

How we can estimate these weights? We need data

Recall that according to our formulation of the learning problem, the only training information available to our learner is whether the game was eventually won or lost. On the other hand, we require training examples that assign specific scores to specific board states (intermediate board states are more difficult to assign a value with respect to the final state board)

Notation:

- $V(b)$: the true target function (always unknown)
- $\hat{V}(b)$: the learned function (approximation of $V(b)$ computed by the learning algorithm)
- $V_{train}(b)$: the training value obtained at $b \in$ training data

→we assign the training value of Vtrain for any intermediate board state b

Dataset $D = \{(b_i, V_{train}(b_i))_{i=1}^n\}$

Estimating training values:

- $V_{train}(b_i) \leftarrow$ human expert
- $V_{train}(b_i) \leftarrow$ set of games
- ...

Learning algorithm

All that remains is to specify the Learning algorithmfor choosing the weights wi to best fit the set of training examples

LMS (least mean squares): for each observed training example it adjusts the weights a small amount in the direction that reduces the error on this training example

LMS Weight update rule:

For each observed training example it adjusts the weights a small amount in the direction that reduces the error on this training example

Initialize w_i

starts from a random initialization of the weights and converge to a reasonable estimation of them

Do repeatedly:

- Select a training example b at random
  1. Compute $error(b)$:

     if error(b)=0, no weights are changed
     if error(b)>0, (V^(b) is too low) each weight is increased in proportion to the value of its corresponding feature

     $$error(b) = V_{train}(b) - \hat{V}(b)$$

  2. For each board feature f_i, update weight w_i:

     $$w_i \leftarrow w_i + c \cdot f_i \cdot error(b)$$

if f_i=0 its weight is not altered regardless of the error so that the only weights updated are those whose features actually occur on the training example board

c is some small constant, say 0.1, to moderate the rate of learning

[illegible] ?

Testing and Evaluation

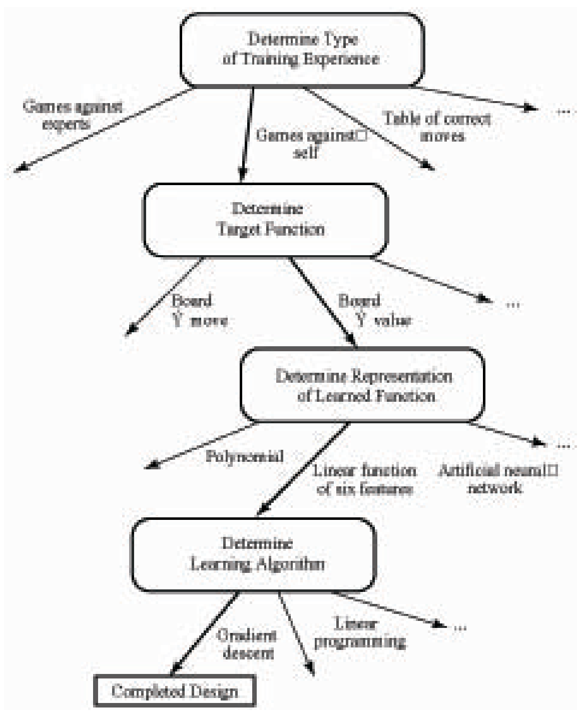
Learned function (after training)

$$\hat{V}(b) = \hat{w}_0 + \hat{w}_1 \cdot bp(b) + \hat{w}_2 \cdot rp(b) + \hat{w}_3 \cdot bk(b) + \hat{w}_4 \cdot rk(b) + \hat{w}_5 \cdot bt(b) + \hat{w}_6 \cdot rt(b)$$

Use this function to play against human players or other computer programs.

How many games won against other players?

Design Choices


Determine Type of Training Experience
Games against experts
Games against self
Table of correct moves
...
Determine Target Function
Board Ÿ move
Board Ÿ value
...
Determine Representation of Learned Function
Polynomial
Linear function of six features
Artificial neural network
...
Determine Learning Algorithm
Gradient descent
Linear programming
...
Completed Design

Machine Learning Problems

One useful perspective on machine learning is that it involves searching a very large space of possible hypotheses to determine one that best fits the observed data and any prior knowledge held by the learner

- Supervised Learning
  - Classification
  - Regression
- Unsupervised Learning
- Reinforcement Learning

Machine Learning Problems

dataset: set of samples that contain info about f. Set of sampled info about f

General machine learning problem:

Learning a function $f : X \to Y$, given a training set D containing information about f

Learning a function f means computing an approximated function $\hat{f}$ that returns values as close as possible to f, specially for samples x not present in the training set D. what is important is that f^ is a good approximation for samples which are not in the dataset

$$\hat{f}(x) \approx f(x) \text{ for } x \in X \setminus X_D$$

Note: $X_D = \{x | x \text{ in } D\} \subset X$ with $|X_D| \lll |X|$

Machine Learning Problems

Different types of ML problems depending on
- type of input dataset different ML approach based on the type of dataset

Learning a function $f : X \to Y$, given ...

- $D = \{(x_i, y_i)_{i=1}^n\}$ **(Supervised Learning)** the dataset is a set of pairs that consist in a sampling of the function (input output pairs)
- $D = \{(x_i)_{i=1}^n\}$ **(Unsupervised Learning)** the dataset s formed only by sample of input

Learning a behavior function $\pi : S \to A$, given ...

- $D = \{(< s_0, a_1, r_1, s_1, \ldots, a_n, r_n, s_n >_i)_{i=1}^n\}$ **(Reinforcement Learning)**

Supervised Learning

Different types of ML problems depending on
- type of function to be learned types of input domain

$$X \equiv \begin{cases} A_1 \times \ldots \times A_m,\ A_i \text{ finite sets } \textbf{(Discrete)} \\ \Re^n \ \textbf{(Continuous)} \end{cases}$$

$$Y \equiv \begin{cases} \Re^k \ \textbf{(Regression)} \\ \{C_1, \ldots, C_k\}, \ \textbf{(Classification)} \end{cases}$$

Special case:
$X \equiv A_1 \times \ldots A_m$ (A_i finite) and $Y \equiv \{0, 1\}$ **(Concept Learning)**

Classification (aka Pattern Recognition)

Return the class to which a specific instance belong.

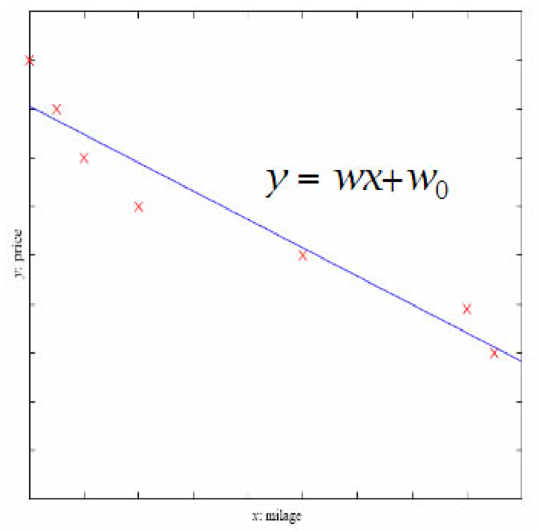
Examples:

- Face recognition: pose, lighting, occlusion (glasses, beard), make-up, hair style.
- Character recognition: different handwriting styles.
- Speech recognition: temporal dependency, multi-modality (dictionary, visual and acoustic)
- Medical diagnosis

Regression

Approximate real-valued functions

Example



Unsupervised Learning

- Learning what normally happens
- No output available
- Clustering: grouping similar instances
- Example applications
  - Customer segmentation in CRM
  - Image compression: color quantization
  - Bioinformatics: learning motifs

Reinforcement Learning

- Learning a policy: a sequence of outputs
- No supervised output available, only sparse and time-delayed rewards
- Example applications
  - Game playing
  - Robotic tasks
  - Multiple agents
  - ... (any dynamic system with unknown or partially known evolution model)

Open questions in Machine Learning

- What algorithms can approximate functions well (and when)?
- How does number of training examples influence performance?
- How does complexity of hypothesis representation impact performance?
- How does noisy data influence performance?
- What are the theoretical limits of learnability?
- How can prior knowledge of learner help?
- How can systems alter their own representations?
- How can systems learn continuously?
- What clues can we get from biological learning systems?
- ...

Concept Learning: Notation

Concept learning can be viewed as the task of searching through a large space of hypotheses implicitly defined by the hypothesis representation. The goal of this search is to find the hypothesis that best fits the training examples

c: target function $c : X \rightarrow \{0, \overset{Y}{1}\}$

X: instance space

$x \in X$: one instance

$D = \{(x_i, c(x_i))_{i=1}^n\}$: training set

$c(x)$: value of the target function over x (*true value* known only for instances in D)

H: hypothesis space

$h \in H$: one hypothesis (an approximation of c)

$h(x)$: estimation of h over x (*predicted or estimated value*)

Learning task

Notice that although the learning task is to determine a hypothesis h identical to the target concept c over the entire set of instances X, the only information available about c is its value over the training examples

input-output pairs

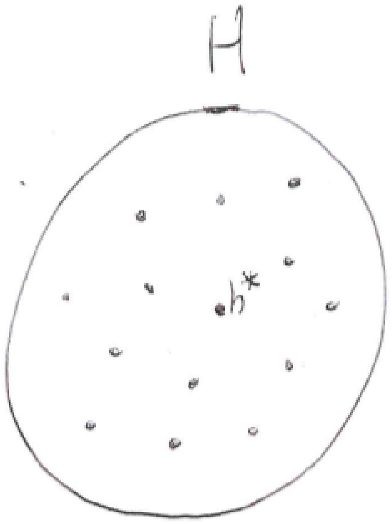
Given a training set $D = \{(x_i, c(x_i))\}$, find the *best* approximation $h^* \in H$ of the target function $c : X \to \{0, 1\}$.

Steps:

1. Define the hypothesis space H (i.e., a representation of the hypotheses)
2. Define a performance metric to determine the *best* approximation
3. Define an appropriate algorithm that selects among all the possible hp, the best one

Learning as a search problem

Given a representation of the hypothesis space H, search for the *best* hypothesis $h^* \in H$, according to a given performance measure.



Evaluating instances on hypotheses

$h(x)$ can be computed for every $x \in X$

$h(x_i) = c(x_i)$ can be verified only for instances x_i appearing in the data set D ($x_i \in D$), for which we know $c(x_i)$

> *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 $(x, c(x))$ in D.*

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

The *real goal* of a machine learning system is to find the *best* hypothesis h that predicts correct values of $h(x')$ for instances $x' \notin D$ with respect to the unknown values $c(x')$.

Performance measure

Given a training set $D = \{(x_i, c(x_i))\}$ and a hypothesis $h \in H$, a performance measure is based on evaluating $c(x_i) = h(x_i)$ for all $x_i \in D$.

Inductive learning hypothesis: Any hypothesis that approximates the target function well over a sufficiently large set of training examples will also approximate the target function well over other unobserved examples.

Training Examples

Day	Outlook	Temperature	Humidity	Wind	PlayTennis
D1	Sunny	Hot	High	Weak	No
D2	Sunny	Hot	High	Strong	No
D3	Overcast	Hot	High	Weak	Yes
D4	Rain	Mild	High	Weak	Yes
D5	Rain	Cool	Normal	Weak	Yes
D6	Rain	Cool	Normal	Strong	No
D7	Overcast	Cool	Normal	Strong	Yes
D8	Sunny	Mild	High	Weak	No
D9	Sunny	Cool	Normal	Weak	Yes
D10	Rain	Mild	Normal	Weak	Yes
D11	Sunny	Mild	Normal	Strong	Yes
D12	Overcast	Mild	High	Strong	Yes
D13	Overcast	Hot	Normal	Weak	Yes
D14	Rain	Mild	High	Strong	No

Prototypical Concept Learning Task

Given:

we have a set of samples, we want to learn a target function (playtennis)

- Instances X (e.g., possible days, each described by the attributes *Outlook, Temperature, Humidity, Wind*)
- Target function $c : X \to \{0, 1\}$ (e.g., $c = PlayTennis : X \to \{No, Yes\}$)
- Hypotheses H

  combination of attributes and Yes or No
- Training examples $D = \{(x_1, c(x_1)), \ldots, (x_n, c(x_n))\}$, positive and negative examples of the target function

Determine:

- A consistent hypothesis (i.e., $h \in H$ such that $h(x) = c(x)$, $\forall x \in D$).

Representation of hypothesis space

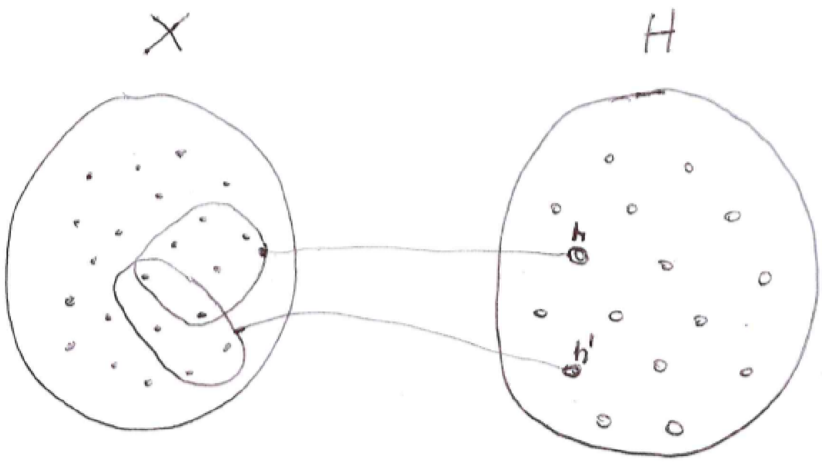
... let's skip this part for the moment ...

Representation of hypothesis space

In concept learning (i.e., binary classification), every hypothesis is associated to a set of instances (i.e., all the instances that are classified as positive by such hypothesis). in our case x1, x2, ... which are the instances is a set of attributes (Outlook, humidity ...)

relation between hypothesis and the set

$$h \leftrightarrow S = \{x \in X | h(x) = 1\} \subseteq X \qquad H \leftrightarrow 2^X$$

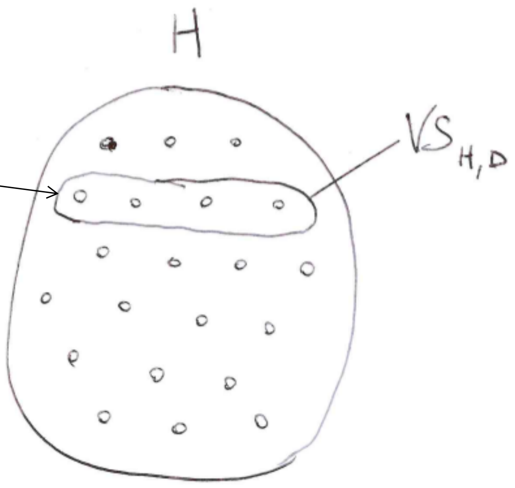


Version Spaces

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.*

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

only hypothesis that are consistent wih the dataset



LIST-THEN-ELIMINATE Algorithm

One obvious way to represent the version space is simply to list all of its members. This leads to a simple learning algorithm, which we might call the LIST-THEN-ELIMINATE algorithm

The LIST-THEN-ELIMINATE algorithm first initializes the version space to contain all hypotheses in H, then eliminates any hypothesis found inconsistent with any training example

1. $VersionSpace \leftarrow$ a list containing every hypothesis in H
2. For each training example, $(x, c(x))$
   remove from $VersionSpace$ any hypothesis h for which $h(x) \neq c(x)$
   until ideally just one hypothesis remains that is consistent with all the observed examples
3. Output the list of hypotheses in $VersionSpace$

Note: enumerating all the hypotheses!

the LIST-THEN-ELIMINATE algorithm can be applied whenever the hypothesis space H is finite. It has many advantages, including the fact that it is guaranteed to output all hypotheses consistent with the training data. Unfortunately, it requires exhaustively enumerating all hypotheses in H

Example 1

Instance space X: integer points in a 2D plane

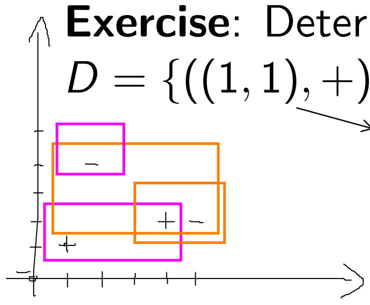
Hypotheses H: rectangles in the 2D plane with edges parallel to the axes

h is consistent only if it contains all points with the same sign

Data set D: positive and negative examples of points belonging to a rectangle

Exercise: Determine the version space of data set
$D = \{((1,1),+),((4,2),+),((2,4),-),((0,0),-),((5,2),-)\}$

(x, c(x))



🟪 consistent hypothesis

🟧 inconsistent hypothesis

Example 2

Now let's consider a different hypothesis space.

Hypotheses H': sets of rectangles in the 2D plane with edges parallel to the axes

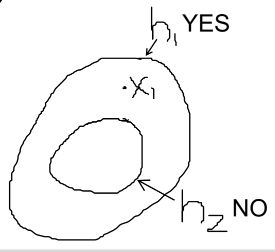
Note: H' can represent any possible subset of X

$$\forall S \in 2^X, \exists h \in H', such\ that\ S = \{x \in X | h(x) = +\}$$

(while this property is not true for the hypothesis space H)

if H' is consistent we will have that the hypotesis sets differ only for 1 element x1 and, h1 and h2 return two different values

H' $\quad h_1, h_2 \in H'$



Output of a learning system

VS: set of hypoteses h consistent with the training examples D

1. $VS_{H',D}$ (H' can represent all the subsets of X)
2. $VS_{H,D}$ (H can not represent all the subsets of X)
3. $h^* \in VS_{H,D}$ (h^* is one hypothesis chosen within the VS)

Classify new instances

Given $x' \notin D$, predict class $+$ or $-$.

1. $VS_{H',D} \Rightarrow \forall x' \notin D$, for some $h' \in VS_{H',D}, h'(x') = +$, for some other $h' \in VS_{H',D}, h'(x') = -$
2. $VS_{H,D} \Rightarrow \exists x' \notin D$, for some $h \in VS_{H,D}, h(x') = +$, for some other $h \in VS_{H,D}, h(x') = -$
3. $h^* \in VS_{H,D} \Rightarrow \forall x' \notin D$, $h^*(x')$ is either $+$ or $-$.

Machine Learning representation issue

1. $VS_{H',D} \Rightarrow$ cannot predict instances not in D (for all $x' \notin D$, it will answer *unknown*)
2. $VS_{H,D} \Rightarrow$ limited prediction instances not in D (for some $x' \notin D$, it will answer *unknown*)
3. $h^* \in VS_{H,D} \Rightarrow$ maximum prediction power on values not in D (for all $x' \notin D$, it will always return a predicted value)

If hypothesis space is too powerful (any subset of the instances can be represented in it), and the search is complete (all the solutions are computed), then the system is not able to classify new instances (no generalization power).

Machine Learning noisy data issue

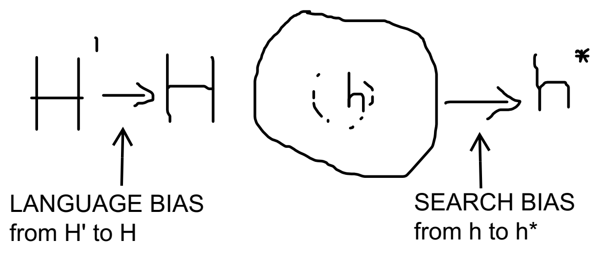
Data set may contain noisy data (a typical case in real applications)
$D = \{(x_i, y_i)\}$ with $y_i \neq c(x_i)$ for some i

There may be no consistent hypotheses, i.e., $VS_{H,D} = \emptyset$

In case of noisy data set, statistical methods must be used to implement robust algorithms.

Summary

- Machine Learning can be seen as learning a function from samples.
- Learning as search requires definition of a hypothesis space and an algorithm to search solutions in this space
- Performance are based on consistency
- Two main issues: 1) representation vs. generalization power, 2) noisy data
- **Statistical methods are needed!!!**



in some cases from H'->h* we have only directly the search bias