Computational Learning Theory

COLT tries to explain why and when machine learning works.

It studies two aspects of machine learning to provide insights for the design of learning algorithms.

- *Statistical*: how much data is needed to learn good models?
- *Algorithmic*: how computationally hard is it to learn such models?

COLT usually assumes a simple learning scenario called *concept learning*, which is (roughly) noise-free binary classification learning.

More complex scenarios often have concept learning at their heart.

Concept Learning Elements

- *Instance space*: a set X. Elements $x \in X$ are *instances*.
- *Concept*: a subset $C \subseteq X$.

The algorithm should learn to decide whether $x \in C$ for any given $x \in X$.

Example: $X =$ animals described as tuples of binary variables

	aquatic	airborne	backbone
$x =$	0	1	0

$C =$ all mammals.

- *Learning examples*: the learner must get some instances $x \in X$ with the information whether $x \in C$ or not.

Concept Class

In general, there is an un-countable number of concepts on X if X is infinite, e.g. $X = N$.

In practice, we often have prior knowledge about the concepts we want to learn: they belong to a restricted set of concepts

$$\mathcal{C} \subset 2^X$$

which is called a *concept class*.

COLT studies the behavior of learners with respect to selected concept classes.

Hypothesis Class

A *finite* description of a learner's decision model is called a *hypothesis*. It is an *algorithm* implementing the function $c : X \to \{ 0, 1 \}$

$$c(x) = \begin{cases} 1 \text{ if } x \in C \\ 0 \text{ if } x \notin C \end{cases}$$

Note: in general, there are fewer hypotheses than concepts due to the finiteness of the former.

Learners use constrained languages (rules, polynomials, graphs, ...) to encode their hypotheses. So the algorithmic semantics is implicit. For example, the hypothesis

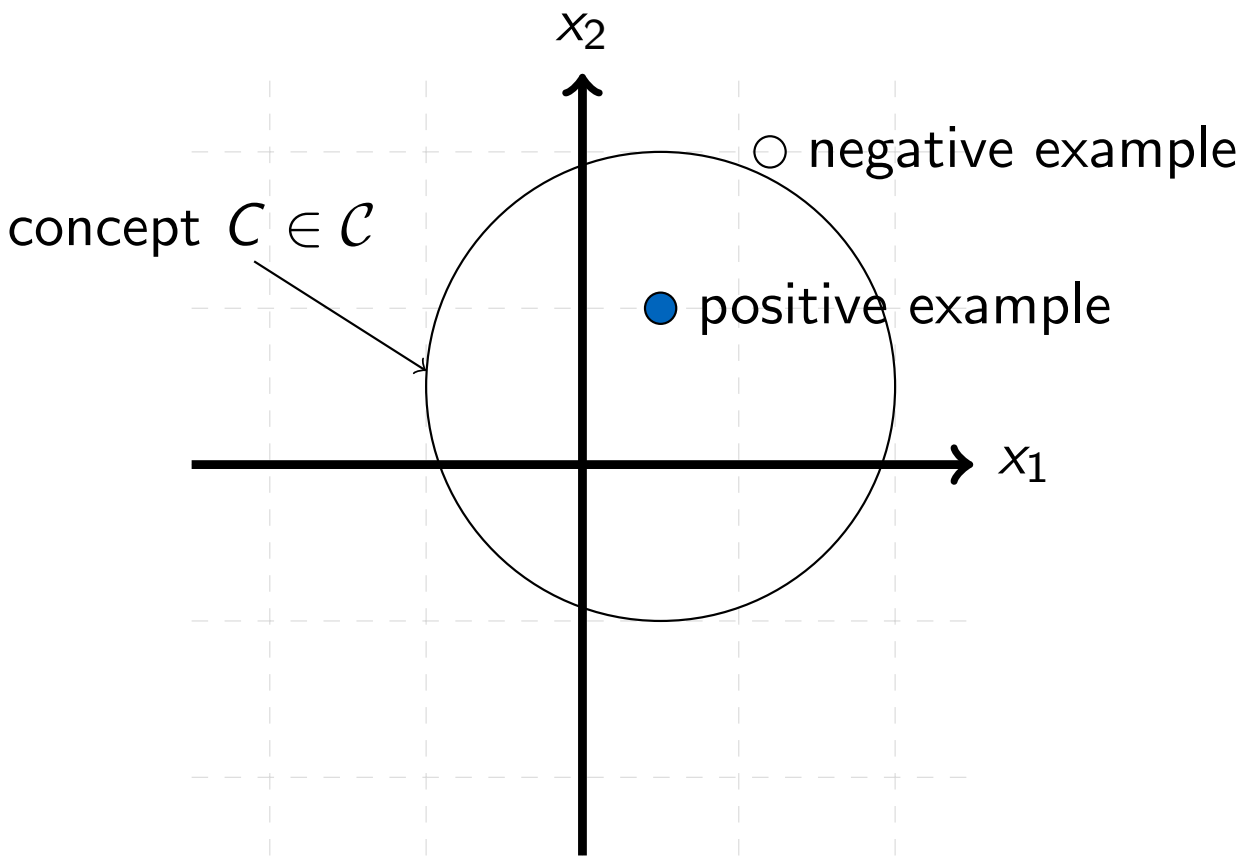
$$\text{man} \wedge \overline{\text{married}}$$

which is a *logical conjunction* defines the 'bachelor' concept.

The set of all hypotheses a learner can express is called its *hypothesis class*.

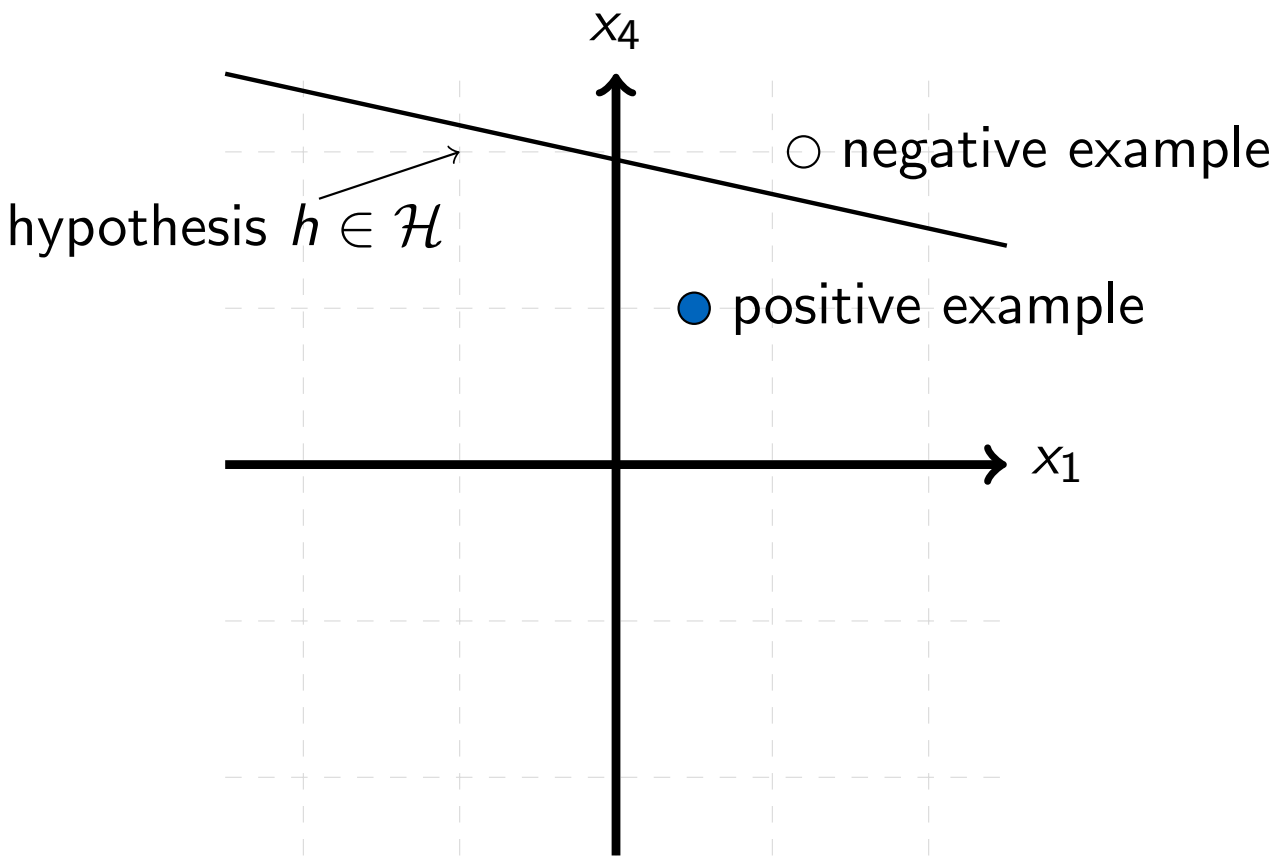
A Continuous-Domain Example

- Instance space $X = R^2$
- Possible concept class $\mathcal{C}$: disks $(x_1 - a)^2 + (x_2 - b)^2 < r$



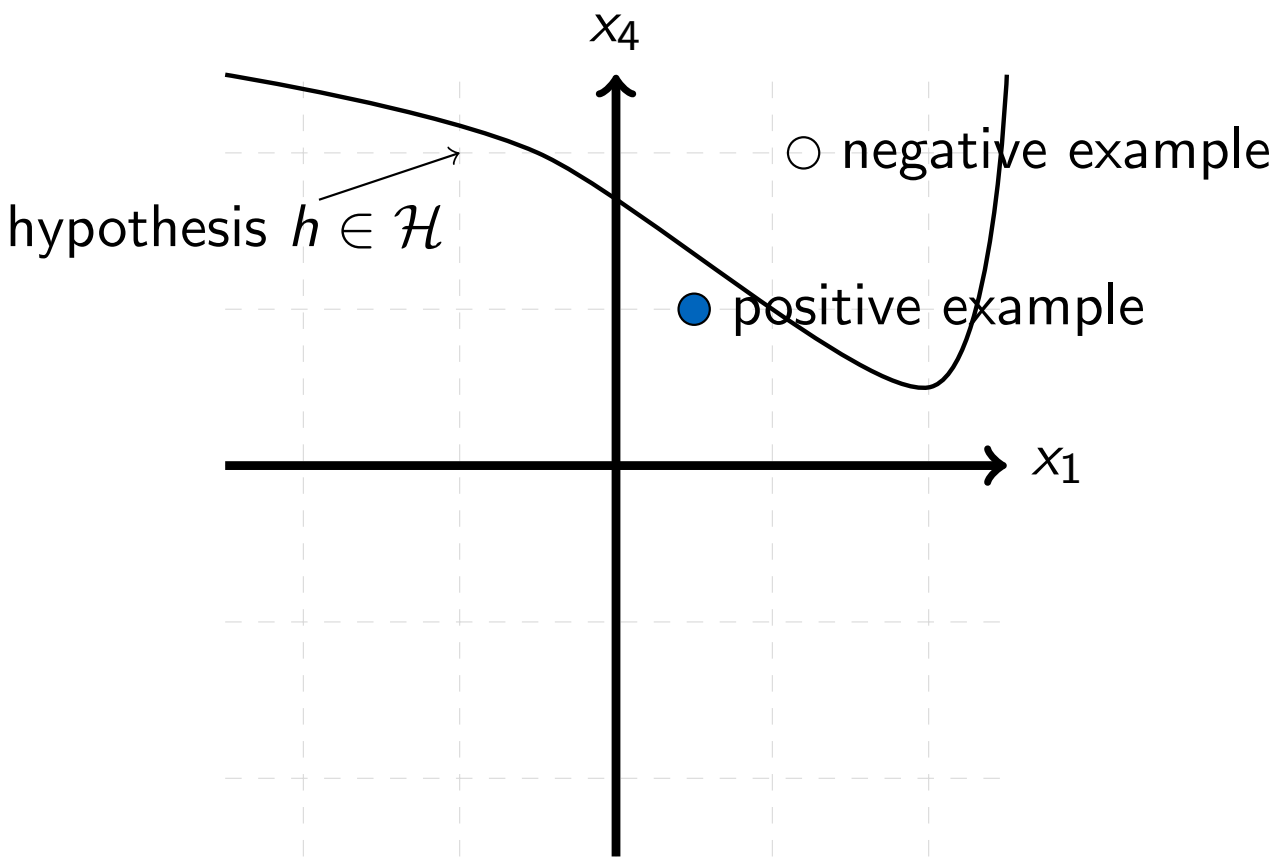
A Continuous-Domain Example (cont'd)

- Possible hypothesis class $\mathcal{H}$: half-planes $x_2 - ax_1 > b$
- Hypothesis description: (a, b) (with finite precision number repr.)



A Continuous-Domain Example (cont'd)

- Possible hypothesis class $\mathcal{H}$: neural networks
- Hypothesis description: graph + weights



Continuous vs. Discrete

Instances and hypotheses in continuous domains are largely the topic of a parallel course (Statistical Machine Learning).

Here we focus mainly on discrete domains that allow convenient symbolic representations. Typically:

- Instance attributes are Boolean values;
- Hypotheses are logical formulas (or structures that can be converted to them).

Symbolic representations have the advantage of *understandability* to a human. Important e.g. in medical applications.

Currently studied in the field of "Explainable AI".

Learning Models

A *learning model* is an abstract description of real-life machine-learning scenarios. It defines

- The learner-environment interaction protocol
- How learning examples are conveyed to the learner
- What properties the examples must posses
- What it means to learn successfully

We will discuss two learning models:

- *Mistake Bound Learning*
- *Probably Approximately Correct Learning*.

Sometimes, *hypotheses* are also called *models* but here we mean a *model of learning*.

Mistake Bound Model

A very simple model assuming an *online* interaction: a concept C is chosen from a fixed concept class and the following is then repeated indefinitely:

1. The learner receives an example $x \in X$
2. It predicts whether x is positive ($x \in C$) or negative ($x \notin C$)
3. It is told the correct answer (so it can adapt after a wrong prediction)

To define the model, we assume there is a measure n of *instance complexity*. When X consists of fixed-arity tuples, we set $n =$ their arity.

Denote poly(n) to mean "at most polynomial in n".

In math expressions, $f(n) \leq \text{poly}(n)$ means that $f(n)$ grows at most polynomially.

Mistake Bound Model

We say that an algorithm *learns concept class $\mathcal{C}$* if for any $C \in \mathcal{C}$, the number of mistakes it makes is poly(n); if such an algorithm exists, $\mathcal{C}$ is called *learnable* in the mistake bound model.

We will omit "in the mistake bound model" in this section.

Note that the learner

- cannot assume anything about the choice of examples (no i.i.d. or order assumption etc.);
- which learns $\mathcal{C}$ stops making mistakes after a finite number of decisions.

If an algorithm learns $\mathcal{C}$ and the maximum time it uses to process a single example is also poly(n), we say it learns $\mathcal{C}$ *efficiently* and we call $\mathcal{C}$ *efficiently* learnable.

Learning Conjunctions

Assume $X = \{ 0, 1 \}^n$ ($n \in N$) and $\mathcal{C}$ consists of all concepts expressible via conjunctions on n variables. Consider the following *generalization* algorithm.

1. Initial hypothesis $h = h_1 \overline{h_1} h_2 \overline{h_2} \ldots h_n \overline{h_n}$
2. Receive example x, decide "yes" iff h true for x ($x \models h$)
3. If decision was "no" and was wrong, remove all h's literals false for x
4. If decision was "yes" and was wrong, output "Concept cannot be described by a conjunction."
5. Go to 2

To adapt this algo for $\mathcal{C} =$ *monotone conjunctions* (conj. with no negations), use $h = h_1 h_2 \ldots h_n$ in Step 1.

Learning Conjunctions

Let $C \in \mathcal{C}$ be the concept used to generate the examples and c the conjunction that encodes it. Observe and explain why:

- Initial h tautologically false, n literals get deleted from it on first mistake on a positive (in-concept) example, resulting in $|h| = n$.
- If a literal is in c, it is never deleted from h, so $c \subseteq h$ (literal-wise).
- At least one literal is deleted on each mistake.
- So the max number of mistakes is $n + 1 \leq \text{poly}(n)$.

Thus the algorithm learns conjunctions (in the MB model) and does so efficiently (time per example is linear in n).

So *conjunctions are efficiently learnable*.

Learning Disjunctions

Efficient learnability of conjunctions implies the same for *disjunctions*.

If disjunction c defines concept C then $\overline{c}$ is a *conjunction* defining the *complementary* concept $X \setminus C$.

Use any efficient conjunction learner to learn $X \setminus C$, so the correct answers provided to the learner are according to $\overline{c}$.

Then negate the hypothesis returned by the algorithm, obtaining a disjunction for C.

Learning k-CNF and k-DNF

k-CNF (DNF) is the class of CNF (DNF) formulas whose clauses (terms) have at most k literals. For example, 3-CNF includes

$$(a \vee b)(b \vee \overline{c} \vee d)$$

k-CNF is efficiently learnable.

With n variables, there are $n' = \sum_{i=1}^{k} \binom{n}{i} 2^i \leq \text{poly}(n)$ different clauses.

Introduce a new variable for each of the n' clauses and use an efficient learner to learn a monotone conjunction on these variables. Then plug the original clauses for the variables in the resulting conjunction, obtaining a k-CNF formula. This is efficient due to $n' \leq \text{poly}(n)$.

Analogically, also *k-DNF is efficiently learnable.*

Learning k-term DNF and k-clause CNF

k-term DNF (k-clause CNF): at most k terms (clauses).

No algorithm known for efficient learning of k-term DNF using k-term DNF as the hypothesis class. Same for k-clause CNF.

But *k-term DNF $\subseteq$ k-CNF* since any k-term DNF can be written as an equivalent k-CNF by "multiplying-out." E.g.,

$$(abc) \vee (de) \models\!\!\models (a \vee d)(a \vee e)(b \vee d)(b \vee e)(c \vee d)(c \vee e)$$

So k-term DNF is efficiently learnable by an algorithm using k-CNF as its hypothesis class. This is called *improper* learning.

Analogically: k-clause CNF learnable using k-DNF.