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 can be used for example to estimate how much data will be needed in a decision-support project.

In practice, this is true only for simple "crisp" model classes such as decision trees, rule-sets, linear models etc.

Computational Learning Theory

Currently prevailing *deep neural models* have huge amounts of floating-point parameters.

Here, COLT-based conclusions tend to disagree with real-life experience. More advanced theoretical techniques are required.

- Zhang et al.: *Understanding deep learning requires rethinking generalization*, ICLR 2017
- Belkin et al.: *Reconciling modern machine learning practice and the bias-variance trade-off*, PNAS 2019

Such techniques are beyond our 3-lecture tour, but you need the COLT basics to even start reading about them.

Concept Learning

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. For example

- In reinforcement learning, identifying the transition model $P(s'|s,a)$
- can be posed as supervised regression $s', s, a \rightarrow [0;1]$
- which can be discretized into a multi-class classification task
- which corresponds to a set of concept-learning tasks

This is not a recipe! It just shows how a complex task reduces to an elementary task we study here.

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.

A concept $C \in \mathcal{C}$ can also be referred to by a function $c : X \to \{ 0, 1 \}$

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

Hypothesis Class

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

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

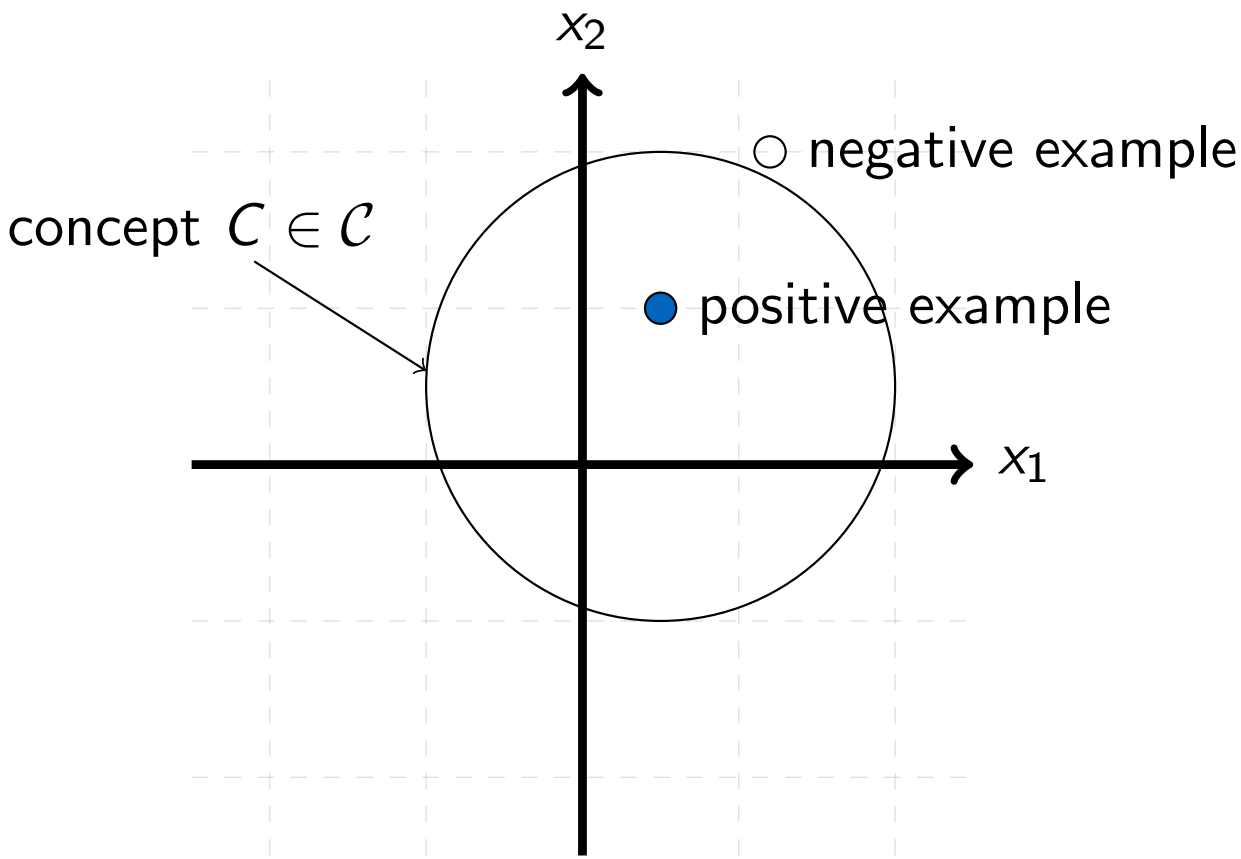
$$h = \text{man} \wedge \overline{\text{married}}$$

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

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

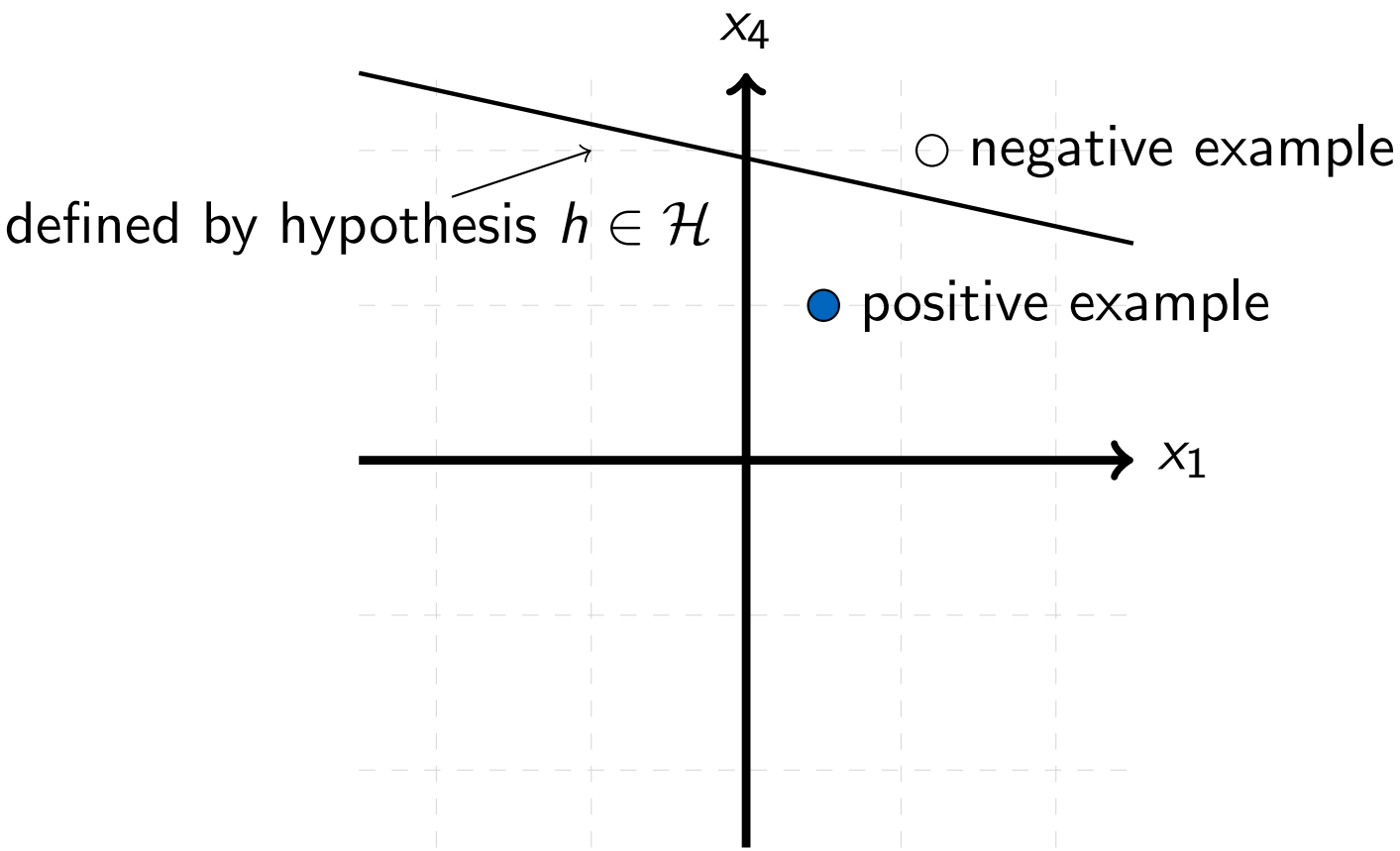
A Continuous-Domain Example

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



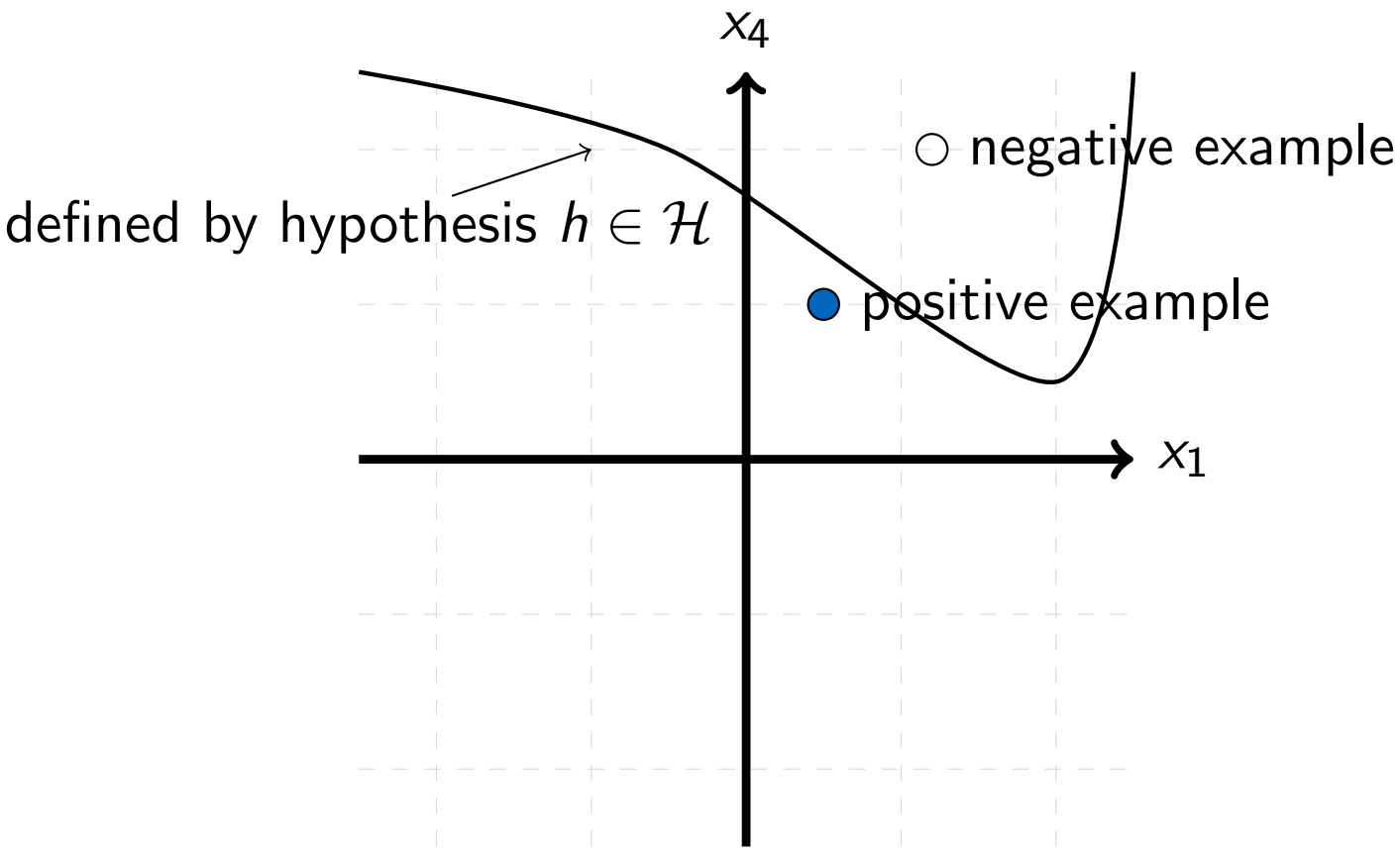
A Continuous-Domain Example (cont'd)

- 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".

Explainable AI

> *"High-risk AI systems shall be designed and developed in such a way as to ensure that their operation is sufficiently transparent to enable deployers to interpret a system's output and use it appropriately." EU AI Act, Article 13(1)*

Industry shift toward *Explainable AI (XAI)*. Gartner (March 2026): "by 2029, 70% of government agencies will require Explainable AI and human-in-the-loop mechanisms for all automated decisions."

Deep neural networks are black-box models. In high-risk applications, this lack of transparency is unacceptable.

Symbolic models such as decision trees or lists, unless oversized, are intrinsically interpretable. We can formally verify their inference logic.

Summary of Symbols

Note the symbol overloading:

Hypotheses	**Concepts**	**Meaning**
h	(none)	finite symbolic description
$h(x)$	$c(x)$	mapping $X \to \{0, 1\}$
$H = \{x \in X \mid h(x) = 1\}$	$C = \{x \in X \mid c(x) = 1\}$	set of instances (we have not used H yet)
$\mathcal{H} = \{H_1, H_2, \ldots\} \subseteq 2^X$	$\mathcal{C} = \{C_1, C_2, \ldots\} \subseteq 2^X$	hypothesis / concept class
$\mathcal{H} = \{h_1, h_2, \ldots\}$		hypothesis class as a set of descriptions

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*.

In practice, *hypotheses* are often also called *(data) 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.). This aligns with the "adversarial learning" scenario.
- 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 $h = h_1h_2\ldots h_n$ 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 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).

Efficient learning of k-term DNF using k-term DNF as the hypothesis class *impossible* (unless P=NP). 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.

The WINNOW Algorithm

An algorithm to learn linearly separable concepts on $\{ 0, 1 \}^n$, which pioneered the *Multiplicative Weights Update* paradigm widely used in machine learning today (Arora, 2012).

Monotone (no negation) conjunctions and monotone disjunctions are linearly separable. Non-monotone convertible to monotone by doubling the number of variables.

WINNOW hypothesis space is R^n, $h = [h_1, h_2, \ldots, h_n]$. h_i are "weights". Initially $h = [1, 1, \ldots 1]$.

Decision rule: decide "yes" if $\sum_{i=1}^{n} h_i \cdot x_i > n/2$, else "no".

Similar to the well-known perceptron algo. Main difference is the learning rule.

The WINNOW Algorithm: Learning rule

Unlike the perceptron, WINNOW adapts weights *multiplicatively*.

When an example x is misclassified, h changes to h':

- If x is positive (i.e., "false negative"), *double* all h_i where $x_i = 1$:

$$h'_i = 2h_i$$

- If x is negative (i.e., "false positive"), *nullify* all h_i where $x_i = 1$:

$$h'_i = 0$$

Other weights ($\forall i, x_i = 0$) remain same.

Let us develop a mistake bound for WINNOW learning *monotone k-disjunctions*, i.e., monotone disjunctions of up to $k \in N$ variables.

The WINNOW Algorithm: Analysis

No weight in h ever becomes negative.

- Only doublings and nullifications from the initial $h = [1, 1, \ldots, 1]$

No weight in h ever exceeds n.

- Assume for contradiction that some $h_j \leq n$ gets doubled to $h'_j > n$ after an example x. That means $h_j > n/2$.
- $x_j = 1$ as otherwise h_j would not get doubled.
- Doubling occurs only after a false negative so $\sum_{i=1}^{n} h_i \cdot x_i \leq n/2$. But that contradicts $h_j > n/2$ considering none of h_i is negative.

The WINNOW Algorithm: Analysis

The total increase in weights after a *false negative* x is at most $n/2$:

$$\sum_{i=1}^{n} h'_i - \sum_{i=1}^{n} h_i = \sum_{i=1}^{n} (h'_i - h_i)x_i = \sum_{i=1}^{n} (2h_i - h_i)x_i = \sum_{i=1}^{n} h_i x_i \leq \frac{n}{2}$$

- first equality due to $h'_i = h_i$ when $x_i = 0$
- second equality due to the doubling rule
- last inequality due to the decision rule and the fact that x was classified negative

The total decrease in weights after a *false positive* x is larger than $n/2$ (shown analogically).

The WINNOW Algorithm: Analysis

For the initial hypothesis, $\sum_{i=1}^{n} h_i = \sum_{i=1}^{n} 1 = n$.

After $\mathcal{N}$ false negatives and $\mathcal{P}$ false positives (using the results from the previous page):

$$0 \leq \sum_{i=1}^{n} h_i \leq n + \mathcal{N}\frac{n}{2} - \mathcal{P}\frac{n}{2}$$

thus

$$\mathcal{P}\frac{n}{2} \leq n + \mathcal{N}\frac{n}{2}$$

i.e. $(n > 0)$,

$$\mathcal{P} \leq 2 + \mathcal{N}$$

The WINNOW Algorithm: Analysis

On each false negative, at least one of the k weights corresponding to the k variables in the concept disjunction gets doubled. (At least one of these variables must have been true for the disjunction to be true.)

So after $\mathcal{N}$ false negatives, one of them (h_j) was doubled at least $\mathcal{N}/k$ times so

$$h_j \geq 2^{\frac{\mathcal{N}}{k}}$$

i.e.,

$$\lg h_j \geq \frac{\mathcal{N}}{k}$$

We have shown that no h_i exceeds n. So $\lg h_j \leq \lg n$ and

$$\lg n \geq \frac{\mathcal{N}}{k}$$

The WINNOW Algorithm: Analysis

So we have a bound for the false negatives

$$\mathcal{N} \leq k \lg n$$

and since we have shown that $\mathcal{P} \leq 2 + \mathcal{N}$, we have a total *mistake bound*

$$\mathcal{P} + \mathcal{N} \leq 2 + 2k \lg n$$

The $\lg n$ factor makes WINNOW much faster than the generalization algorithm or the perceptron when k is a small ($k \ll n$) constant.

$k \ll n$ means a "sparse" target concept disjunction—many irrelevant attributes.

The Halving Algorithm (Version Space)

Maintains a *finite set of hypotheses* $\mathcal{H}$ ("version space") and on each example x, deletes from it all hypotheses that misclassify it.

$$\mathcal{H}' = \{\, h \in \mathcal{H} : h(x) = c(x) \,\}$$

Decides by *majority vote* among the current $\mathcal{H}$, i.e., "yes" iff

$$|\{\, h \in \mathcal{H} : h(x) = 1 \,\}| > |\{\, h \in \mathcal{H} : h(x) = 0 \,\}|$$

On each mistake, at least half of the hypotheses were wrong so at least *half* of them get deleted. This gives the *mistake bound*

$$\lg |\mathcal{H}|$$

where $\mathcal{H}$ is the initial version space, i.e., the learner's hypothesis class.

The Halving Algorithm (Version Space)

Any finite class $\mathcal{C}$ of computable concepts is learnable if $\lg |\mathcal{C}| \leq \text{poly}(n)$.

Proof: Use the halving algorithm with any $\mathcal{H} \supseteq \mathcal{C}$ such that $\lg |\mathcal{H}| \leq \text{poly}(n)$. [1]

That does not mean $\mathcal{C}$ is learnable *efficiently*!

If $|\mathcal{C}|$ is exponentially large, then the halving algo is necessarily non-efficient.

The Halving algorithm can be thought of a *class* of learning algorithms; one for each choice of $\mathcal{H}$.

[1] We overload the symbol $\mathcal{H}$ to mean both a class of hypotheses and the collection of subsets of X it induces.

Sizes of Some Concept Classes

- Conjunctions or disjunctions: $|\mathcal{C}| = 2^{2n}$ resp. 3^n if contradictions/tautologies excluded.
  - Both halving and generalization algos have linear mistake bound, but the latter is efficient
- k-disjunctions: $|\mathcal{C}| = \sum_{i=1}^{k} \binom{2n}{i}$ resp. $\sum_{i=1}^{k} \binom{n}{i} 2^i \leq \text{poly}(n)$
  - Both halving and WINNOW: logarithmic mistake bound, efficient
  - k-conjunctions: same, except WINNOW won't apply
- k-DNF, k-CNF: $|\mathcal{C}| = 2^{|k\text{-disjunctions}|} \leq 2^{poly(n)}$
  - Halving: poly mistake bound, non-efficient
  - Reduction to monotone conjuctions (disjuctions): poly m.b., efficient

VC Dimension

We say that concept class $\mathcal{C}$ *shatters* a set of instances $X' \subseteq X$ if for every subset $X'' \subseteq X'$ there is a concept $C \in \mathcal{C}$ such that $C \cap X' = X''$.

A *finite* X' is shattered by $\mathcal{C}$ if it can be split by concepts from $\mathcal{C}$ in all $2^{|X'|}$ possible ways.

The *VC-dimension* of $\mathcal{C}$ denoted $\mathrm{VC}(\mathcal{C})$ is the size of the largest subset of X shattered by $\mathcal{C}$ (if such a largest subset exists):

$$\mathrm{VC}(\mathcal{C}) = \max\{\ |X'|\ :\ \mathcal{C} \text{ shatters } X',\ X' \subseteq X\ \}$$

$\mathrm{VC}(\mathcal{H})$ for a *hypothesis* class $\mathcal{H}$ defined analogically.

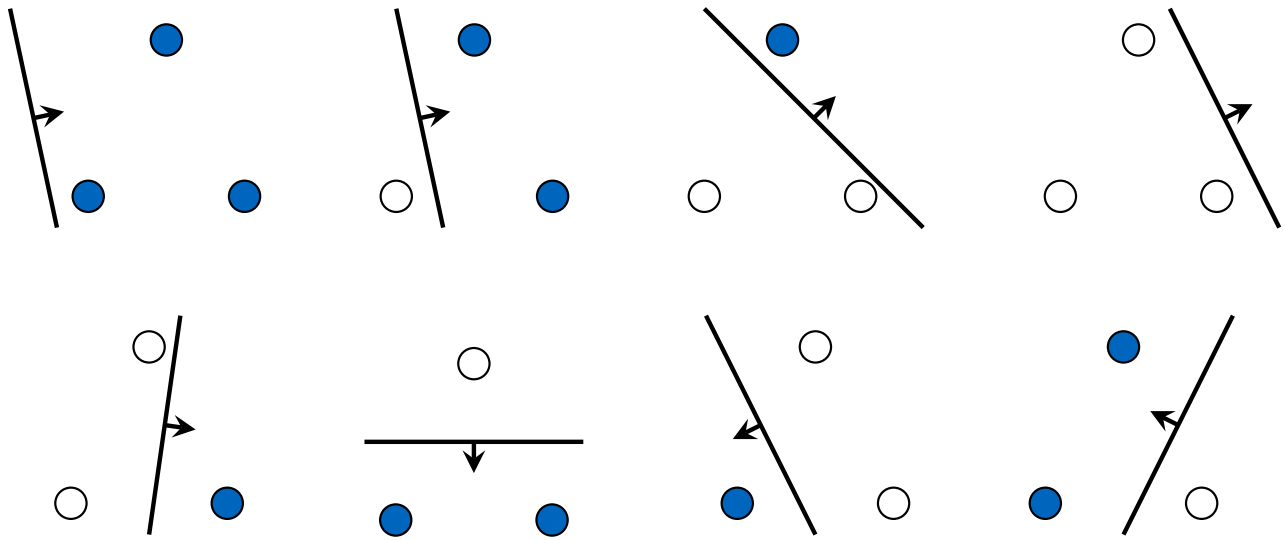
Determining VC-Dimension: Example

- If *some* $X' \subseteq X$, $|X'| = d_1$ shattered by $\mathcal{C}$ then $\text{VC}(\mathcal{C}) \geq d_1$.
- If *none* $X' \subseteq X$, $|X'| = d_2$ shattered by $\mathcal{C}$ then $\text{VC}(\mathcal{C}) < d_2$.

Usually, you try to prove both with $d_1 = d_2 - 1$ to get $\text{VC}(\mathcal{C}) = d_1$.

Example: $\mathcal{C}$ = half-planes in R^2 (i.e., linear separation)

- *Some* 3 points can be shattered so $\text{VC}(\mathcal{C}) \geq 3$.



Determining VC-Dimension: Example

- *No* 4 points can be shattered. Obvious if 3 in line. Otherwise two cases possible:

One point in the middle No point in the middle

In both cases, the colored subset cannot be separated by a line. So $\text{VC}(\mathcal{C}) < 4$

We have $\text{VC}(\mathcal{C}) \geq 3$ and $\text{VC}(\mathcal{C}) < 4$, thus $\text{VC}(\mathcal{C}) = 3$.

Generally, VC(half-planes in R^n) $= n + 1$

Poly VC-Dimension Necessary for Learnability

Recall: $\lg(\mathcal{C}) \leq \text{poly}(n)$ is a *sufficient* condition for learnability. In contrast:

Concept class $\mathcal{C}$ on X is learnable *only if* $\text{VC}(\mathcal{C}) \leq \text{poly}(n)$.

Proof: There exists a set of $\text{VC}(\mathcal{C})$ instances from X shattered by $\mathcal{C}$ so there exists a sequence $x_1, x_2, \ldots x_{\text{VC}(\mathcal{C})}$ of instances such that for any sequence of the learner's decisions there is a concept $c \in \mathcal{C}$ making all these decisions wrong.

So $\lg |\mathcal{C}| \leq \text{poly}(n)$ implies $\text{VC}(\mathcal{C}) \leq \text{poly}(n)$ but not the other way around.

$\text{VC}(\mathcal{C})$ may be finite (even $\text{poly}(n)$) even if $|\mathcal{C}| = \infty$!

PAC Learning Model

PAC = Probably Approximately Correct

Main differences from the mistake bound model:

- A "batch" or "offline" style of learning rather than "online":
  - A *training* multiset of examples is provided to the learner.
  - The learner outputs a hypothesis.
- Assumes an arbitrary probability distribution on X from which examples are drawn mutually independently ("i.i.d. assumption") with replacement.
- No bound on the total number of mistakes, instead the output hypothesis should have a bounded *error* rate (mistake probability).
- Probability of failure (good hypothesis not found) also bounded.
- Size of the training multiset only polynomial in n and the inverse of the two bounds.

PAC Learning Model: Definition

Given a probability distribution P on X, a concept C and a hypothesis H, define the *error* of H: $err(H) = P(C \triangle H) = P(c(x) \neq h(x))$

Define also: $\text{err}(h) = \text{err}(H)$

We say that an algorithm *PAC-learns concept class* $\mathcal{C}$ if for any $C \in \mathcal{C}$, an arbitrary distribution P on X, and arbitrary numbers $0 < \epsilon, \delta < 1$, the algorithm, which receives a $\text{poly}(1/\epsilon, 1/\delta, n)$ number of i.i.d. examples from $P(X)$, outputs with probability at least $1 - \delta$ a hypothesis h such that $\text{err}(h) \leq \epsilon$. If such an algorithm exists, we call $\mathcal{C}$ *PAC-Learnable*.

If an algorithm PAC-learns $\mathcal{C}$ and runs in $\text{poly}(1/\epsilon, 1/\delta, n)$ time, we say it PAC-learns $\mathcal{C}$ *efficiently* and we call $\mathcal{C}$ *efficiently PAC-learnable*.

PAC Learning Conjunctions

Use the generalization algo (previous lecture) for PAC learning: provide m examples to it, run it as if online, keep the last h.

Let $P_{\text{ic}}(z)$ be the prob. that literal z ($z \in \left\{ h_1, \overline{h_1}, h_2, \ldots \overline{h_n} \right\}$) is inconsistent with a random example x drawn from $P(X)$.

For example, x with $x_3 = 0$ is inconsistent with h_3 and consistent with $\overline{h_3}$.

$\text{err}(h) = P(\text{at least one literal in } h \text{ inconsistent}) \leq \sum_z P_{\text{ic}}(z)$

Call z *bad* if $P_{\text{ic}}(z) \geq \frac{\epsilon}{2n}$. So if h has no bad literals then

$$\text{err}(h) \leq \sum_z \frac{\epsilon}{2n} = 2n\frac{\epsilon}{2n} = \epsilon$$

PAC Learning Conjunctions

Prob. that a bad literal z "survived" (was consistent with) one random example is

$$1 - P_{\text{ic}}(z) \leq 1 - \frac{\epsilon}{2n}$$

Prob. that z survived *m such i.i.d. examples* is thus at most

$$\left(1 - \frac{\epsilon}{2n}\right)^m$$

In the worst case, all $2n$ literals are bad. The probability that one of them survived m i.i.d. examples is at most

$$2n\left(1 - \frac{\epsilon}{2n}\right)^m < 2ne^{-\frac{m\epsilon}{2n}}$$

because of the general inequality $1 - x < e^{-x}$ for $x > 0$. (The RHS is easier to work with.)

PAC Learning Conjunctions

To satisfy PAC-learning conditions, we need

$$2ne^{-\frac{m\epsilon}{2n}} < \delta$$

after arrangements:

$$m \geq \frac{2n}{\epsilon}\left(\ln 2n + \ln \frac{1}{\delta}\right)$$

Thus $m \leq \text{poly}(1/\epsilon, 1/\delta, n)$ examples suffice to make $\text{err}(h) \leq \epsilon$ with probability at least $1 - \delta$.

So the generalization algorithm *PAC-learns* conjunctions (also *efficiently* - same argument as in the mistake-bound framework).

Mistake-Bound Learnability Implies PAC-Learnability

Any mistake-bound learner L can be transformed into a PAC-learner. Let $M \leq \text{poly}(n)$ be the mistake bound of L.

Call L *lazy* if it changes its hypo h *only* following a mistake. If L is not lazy, make it lazy (prevent changing h after correct decisions).

Run L on the training multiset but halt if any hypo h survives (=is consistent with) $\frac{1}{\epsilon} \ln \frac{M}{\delta}$ *consecutive* examples. Output h.

Observe that this *will* terminate within $m = M \cdot \frac{1}{\epsilon} \ln \frac{M}{\delta}$ examples. Otherwise more than M mistakes would be made.

To see why, think of the worst case: M chains of correct decisions, each just shy of the $\frac{1}{\epsilon} \ln \frac{M}{\delta}$ length threshold.

Mistake-Bound Learnability Implies PAC-Learnability

Prob. that a *bad* h ($\text{err}(h) > \epsilon$) survives $\frac{1}{\epsilon} \ln \frac{M}{\delta}$ consecutive i.i.d. examples is at most $(1-\epsilon)^{\frac{1}{\epsilon} \ln \frac{M}{\delta}}$. So the prob. that one of the bad hypotheses the algo tries (at most M of them) survives and is output is at most

$$M(1-\epsilon)^{\frac{1}{\epsilon} \ln \frac{M}{\delta}} < Me^{-\frac{\epsilon}{\epsilon} \ln \frac{M}{\delta}} = M\frac{\delta}{M} = \delta$$

(Here we used $x > 0 \to 1 - x < e^{-x}$ again.)

Since $M \leq \text{poly}(n)$ (condition of MB learning), also

$$m = \frac{M}{\epsilon} \ln \frac{M}{\delta} \leq \text{poly}(1/\epsilon, 1/\delta, n)$$

So PAC-learning conditions satisfied: we have $m \leq \text{poly}(1/\epsilon, 1/\delta, n)$, and $\text{err}(h) \leq \epsilon$ with prob. at least $1 - \delta$.

PAC-Learning Implies Consistency

Although $\text{err}(h) > 0$ is allowed, the output h of a PAC-learner is necessarily consistent with all the training examples (zero *training error*).

Assume for contradiction that given training multiset X_T, a PAC algorithm outputs h *inconsistent* with some $x^* \in X_T$.

Distribution P is arbitrary so it can be such that $P(x) > 0$ for all $x \in X_T$. Numbers $\epsilon > 0, \delta > 0$ also arbitrary so they may be such that

- $\delta < \prod_{x \in X_T} P(x)$ so we receive X_T with probability $> \delta$, resulting in a h making an error on x^*.
- $\epsilon < P(x^*)$ so h exceeds the error threshold

So the algorithm *does not* PAC-learn.

Consistency + Polynomial In $|\mathcal{H}|$ Imply PAC-Learning

An algorithm using hypothesis class $\mathcal{H}$ is *$\mathcal{C}$-consistent* if, given an arbitrary training multiset X_T from an arbitrary concept $C \in \mathcal{C}$, it returns a $h \in \mathcal{H}$ consistent with X_T.

$\mathcal{H} \supseteq \mathcal{C}$ is a necessary condition for $\mathcal{C}$-consistency.

Proposition: a $\mathcal{C}$-consistent algorithm using $\mathcal{H}$ PAC-learns $\mathcal{C}$ if $\ln |\mathcal{H}| \leq \text{poly}(n)$.

Indeed, prob. that a given *bad* h ($\text{err}(h) > \epsilon$) survives (i.e., is consistent with) a random example is at most $(1 - \epsilon)$ (... turn page)

Consistency + Polynomial In $|\mathcal{H}|$ Imply PAC-Learning

Prob. that h survives m i.i.d. examples is at most $(1-\epsilon)^m$.

Prob. that one of the bad hypotheses $h \in \mathcal{H}$ survives is at most $|\mathcal{H}|(1-\epsilon)^m < |\mathcal{H}|e^{-\epsilon m}$.

To make this smaller than δ, it suffices to set the number of examples to

$$m = \frac{1}{\epsilon} \ln \frac{|\mathcal{H}|}{\delta}$$

which is $\leq \text{poly}(1/\epsilon, 1/\delta, n)$ iff $\ln |\mathcal{H}| \leq \text{poly}(n)$.

Compare this to the similar result in the mistake-bound model (Halving algorithm).

Consistency + Polynomial VC($\mathcal{H}$) Imply PAC-Learning

Using VC($\mathcal{H}$), a bound can be established even for $|\mathcal{H}| = \infty$:

With probability at least $1 - \delta$, no bad hypothesis $h \in \mathcal{H}$ survives m i.i.d. examples where

$$m \geq \frac{8}{\epsilon}\left(\text{VC}(\mathcal{H}) \ln \frac{16}{\epsilon} + \ln \frac{2}{\delta}\right)$$

(We omit the proof.)

Thus a $\mathcal{C}$-consistent algorithm using $\mathcal{H}$ PAC-learns $\mathcal{C}$ if VC($\mathcal{H}$) $\leq$ poly(n).

For example, let $\mathcal{C}$ = half-planes in R^n. $|\mathcal{C}| = \infty$, $\mathcal{H} \supseteq \mathcal{C}$ (consistency condition), thus $|\mathcal{H}| = \infty$.

But for $\mathcal{H}$ = half-planes, VC($\mathcal{H}$) $= n + 1 \leq$ poly(n).

PAC-Learning k-term DNF

We know that $\mathcal{C} = k$-term DNF is learnable efficiently, but improperly using $\mathcal{H} = k$-CNF in the MB model and thus also in PAC.

What about proper learning ($\mathcal{H} = \mathcal{C}$)?

$\ln |\mathcal{C}| = \ln |k\text{-term DNF}| \leq \ln |k\text{-CNF}| \leq \text{poly}(n)$ so $\mathcal{C}$ is properly PAC-learnable.

BUT: this cannot be done *efficiently*. We show this for $k = 3$.

Finding a $h \in \mathcal{H} = k$-term DNF consistent with the training examples is as hard as the *graph 3-coloring problem*:

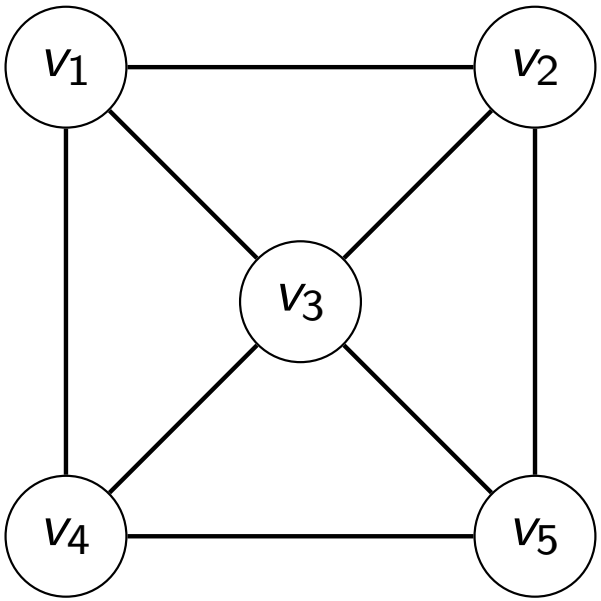
- Give each vertex one of 3 colors, adjacent vertices - different colors

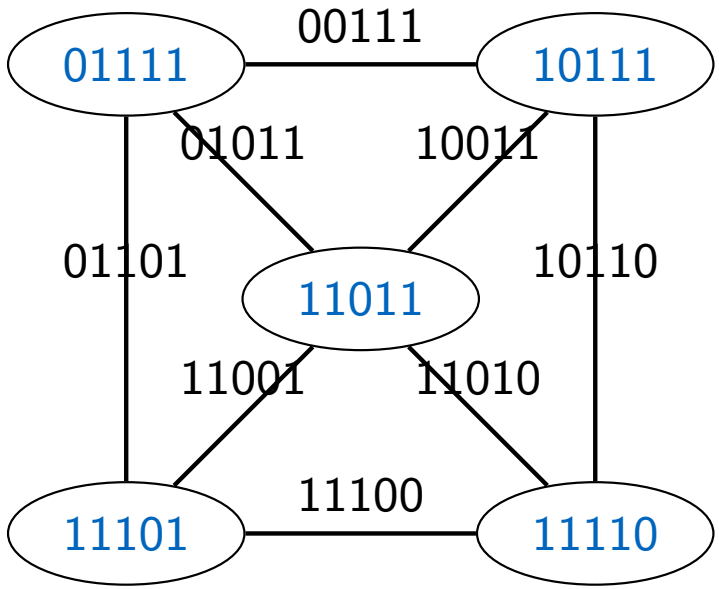
3-Coloring as Finding a Consistent 3-term DNF

Efficient reduction:

$$\text{vertex } v_i \quad \sim \quad \text{pos. example } x,\ x[k] = \begin{cases} 0 \text{ if } k = i \\ 1 \text{ otherwise} \end{cases}$$

$$\text{edge } e_{ij} \quad \sim \quad \text{neg. example } x,\ x[k] = \begin{cases} 0 \text{ if } k = i \text{ or } k = j \\ 1 \text{ otherwise} \end{cases}$$





3-Coloring as Finding a Consistent 3-term DNF

Graph 3-colorable iff a 3-term DNF on logical variables $h_1, h_2, \ldots$ exists consistent with the examples:

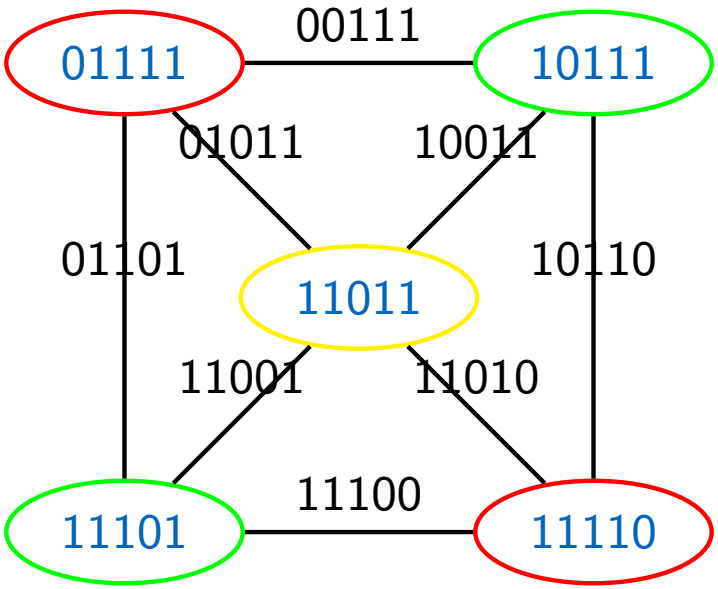
- Given a 3-colored graph, a consistent 3-term DNF can be constructed:

$$\left(\bigwedge_{v_k \text{ NOT red}} h_k \right) \bigvee \left(\bigwedge_{v_k \text{ NOT green}} h_k \right) \bigvee \left(\bigwedge_{v_k \text{ NOT yellow}} h_k \right)$$

- Given a consistent 3-term DNF, the graph can be validly colored
  - Give each vertex the color corresponding to any term consistent with the vertex variable

3-Coloring as Finding a Consistent 3-term DNF

Example:



$\leftrightarrow$

$$h_2 \wedge h_3 \wedge h_4$$
$$\vee$$
$$h_1 \wedge h_3 \wedge h_5$$
$$\vee$$
$$h_1 \wedge h_2 \wedge h_4 \wedge h_5$$

3-colorability NP-hard $\rightarrow$ finding a consistent 3-term DNF NP-hard.

Generally, $\mathcal{C} = k$-term DNF cannot be PAC-learned efficiently AND properly ($\mathcal{H} = \mathcal{C}$).

k-Decision Trees

(Binary) decision tree: a binary tree-graph

- non-leaf vertices: binary variables
- leafs: class indicators

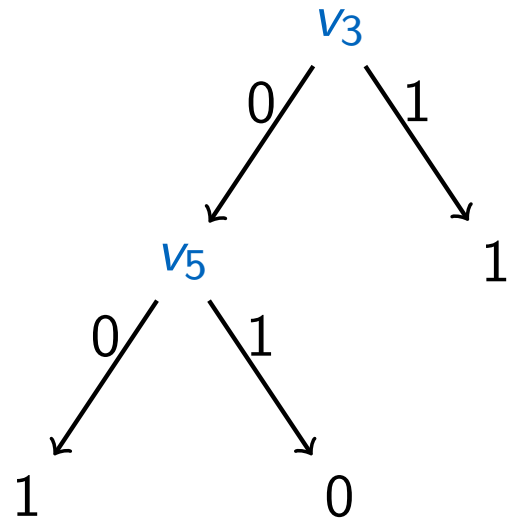
Classification: go from root to leaf, path according to truth-values of variables.

k-DT = dec. trees of max depth k

Like k-term DNF,

- finding a consistent k-DT is NP-hard (proof omitted). *k-DT thus cannot be PAC-learned efficiently + properly.*
- Can they be learned efficiently **or** properly?

Example:



2-Decision Tree

PAC-Learning k-Decision Trees Efficiently

Every k-DT has an equivalent k-DNF:

- For every path going from root to a 1 leaf, add to the DNF a k-conjunction of all variables on the path ($v_3 \vee \overline{v_3}\ \overline{v_5}$ for the example)

Thus

$$k\text{-DT} \subseteq k\text{-DNF}$$

and $\mathcal{C} = k$-DT can be *efficiently (but not properly) PAC-learned* using $\mathcal{H} = k$-DNF.

Note that also

$$k\text{-DT} \subseteq k\text{-CNF}$$

- Create a clause for each path to a 0 leaf ($v_3 \vee \overline{v_5}$ for the example)

PAC-Learning k-Decision Trees Properly

We will show that $\lg |k\text{-DT}| \leq \text{poly}(n)$. Denote $c_k = |k\text{-DT}|$.

- $c_1 = 2$ (two options for the single vertex = leaf) so

$$\lg c_1 = 1 \tag{1}$$

- $c_{k+1} = nc_k^2$ (n options for vertex, c_k options for each of the 2 subtrees)

$$\lg c_{k+1} = \lg n + 2 \lg c_k \tag{2}$$

(1) and (2) are a recursive formula for a geometric series in variable $\lg c_k = \lg |k\text{-DT}|$. Solution exponential in k but polynomial in n.

So $\mathcal{C} = k$-DT can be *properly (but not efficiently) PAC-learned* by a $\mathcal{C}$-consistent algorithm.

k-Decision Lists

k-Decision List: a list of k-conjunctions (each with a class indicator) + default class indicator.

Example:

$$\begin{array}{rcl} v_1\overline{v_3} & \rightarrow & 0 \\ v_2 & \rightarrow & 1 \\ \text{default} & & 0 \end{array}$$

An example is classified to the class indicated at the first from top conjunction satisfied by the example, or the default if none satisfied.

We will show an efficient consistent learning algorithm for k-DL.

Finding a Consistent k-Decision List

Require: training multiset $T = \{ (x_1, y_1), (x_2, y_2) \ldots (x_m, y_m) \}$ ▷ (the $y_i \in \{ 0, 1 \}$ are class labels)

1: $L :=$ [] (empty list)
2: **while** $T \neq \emptyset$ **do**
3: $\gamma =$ any k-conjunction true for some pos. and no neg. example in T, *or* some neg. and no pos. example in T (*respectively*)
4: Remove examples covered by γ: $T := T \setminus \{(x, y) \in T : x \models \gamma\}$
5: **if** $T = \emptyset$ **then**
6: append default 1 *or* default 0 (*respectively*) to L.
7: **else**
8: append $\gamma \to 1$ *or* $\gamma \to 0$ (*respectively*) to L
9: **end if**
10: **end while**

Known as the *Covering algorithm*.

Finding a Consistent k-Decision List

In Step 3, the algorithm always succeeds in finding the required k-conjunction γ, if examples come from a k-DL concept.

Indeed, such a γ exists:

- Let c be the DL encoding the target concept;
- Let $\gamma^* \rightarrow \mathit{class}$ be the top-most rule in c which 'fires' ($x \models \gamma^*$) for least one $x \in T$;
- $\gamma^* \rightarrow \mathit{class}$ must be consistent with T; if inconsistent with any $x' \in T$, a rule higher in c would have to fire for x' but that contradicts the 'top-most' assumption above;
- so $\gamma = \gamma^*$ is one possible choice.

In the worst case, the algo needs to search all of the $\leq$ poly(n) number of k-conjunctions.

PAC-Learning k-DL Properly Efficiently

The k-DL-consistent algorithm PAC-learns k-DL if $\ln |k\text{-DL}| \leq \text{poly}(n)$.

$$|k\text{-DL}| = 3^{|k\text{-conjunctions}|}!$$

- base 3: each k-conjunction either absent, present with class 0 or present with class 1
- factorial: different order of k-conjunctions - different k-DL's

Since $|k\text{-conjunctions}| \leq \text{poly}(n)$, we indeed have

$$\ln |k\text{-DL}| \leq \text{poly}(n)$$

k-DL Subsumes k-DNF and k-CNF

For any k-DNF, an equivalent k-DL can be made:

- for each k-conjunction c in the k-DNF, add c to the DL with class 1
- add to the DL the default rule with class 0

So

$$k\text{-DNF} \subseteq k\text{-DL}$$

k-DL is closed under negation (just flip the class indicators) and each k-CNF is the negation of some k-DNF. Therefore

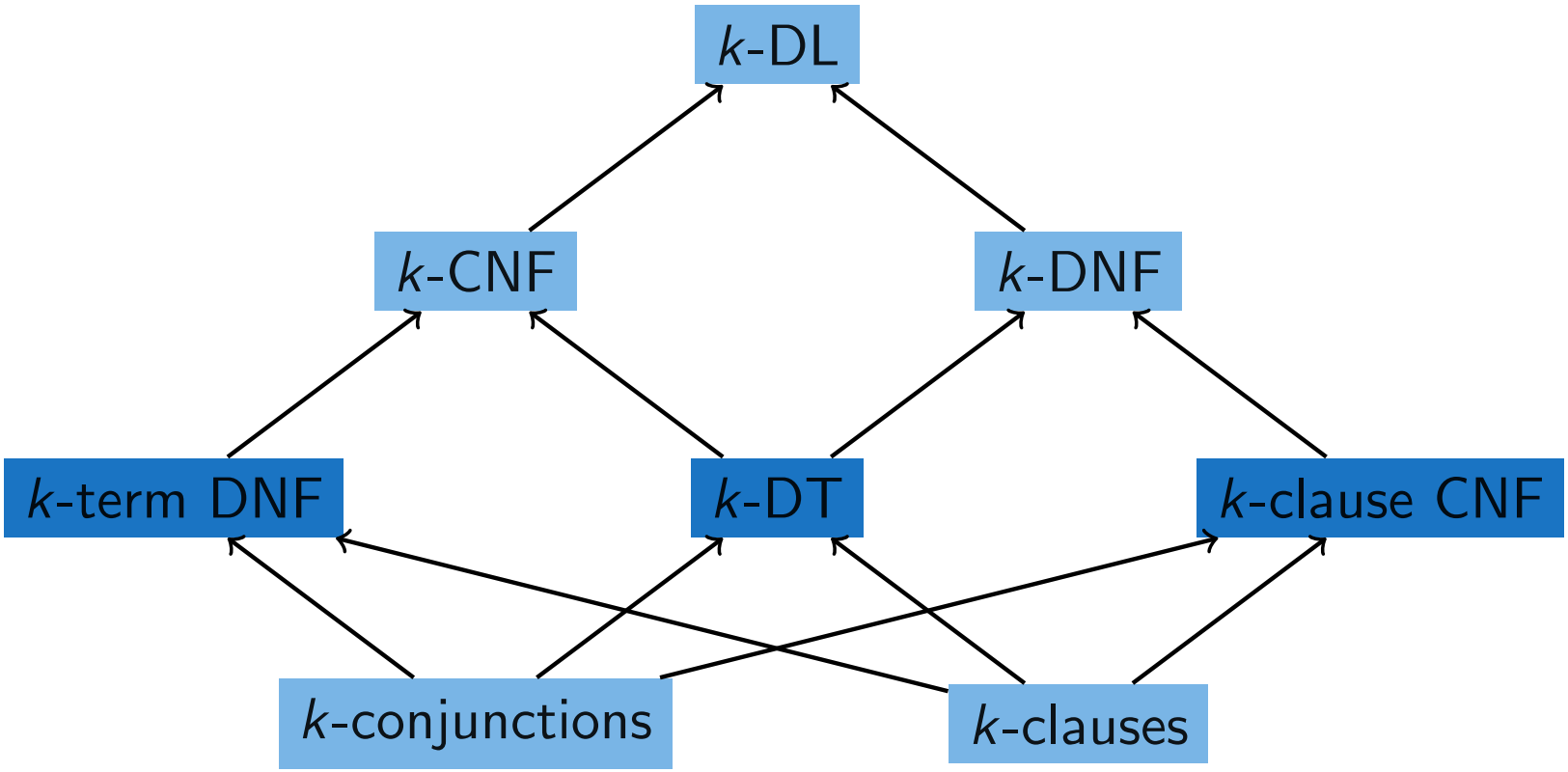
$$k\text{-CNF} \subseteq k\text{-DL}$$

(The inclusions are actually strict because k-DNF $\neq$ k-CNF.)

Subset Hierarchy of Some Concept Classes

efficiently properly PAC-learnable

efficiently **or** properly PAC-learnable



Inconsistent Learning

Returning a hypothesis consistent with the training multiset may not be possible for reasons such as

- $\mathcal{H} \not\supseteq \mathcal{C}$;
- $\mathcal{C}$ is not known ('agnostic learning') so $\mathcal{H} \not\supseteq \mathcal{C}$ cannot be excluded;
- There is 'noise' in data so the training multiset may include the same instance as both a positive and a negative example.

Define the *training error* $\widehat{\mathrm{err}}(h)$ as the proportion of training examples inconsistent with h. $\widehat{\mathrm{err}}(h)$ is also called the *empirical risk*.

We are interested in the relationship btw. $err(h)$ and $\widehat{\mathrm{err}}(h)$.

Hoeffding Inequality

Hoeffding: Let $\{ z_1, z_2, \ldots, z_m \}$ be a set of i.i.d. samples from $P(z)$ on $\{ 0, 1 \}$. The probability that $\left| P(1) - \frac{1}{m} \sum_{i=1}^{m} z_i \right| > \epsilon$ is at most $2e^{-2\epsilon^2 m}$.

Let $z_i = 1$ iff i.i.d. example x_i is misclassified by h. So

$$P(1) = \text{err}(h)$$

$$\frac{1}{m} \sum_{i=1}^{m} z_i = \widehat{\text{err}}(h)$$

Thus for a given h, $|\text{err}(h) - \widehat{\text{err}}(h)| > \epsilon$ with prob. at most $2e^{-2\epsilon^2 m}$.

Error Bound for Inconsistent Learning

For a finite $\mathcal{H}$, the prob. that $|\text{err}(h) - \widehat{\text{err}}(h)| > \epsilon$ for *some* $h \in \mathcal{H}$ is at most

$$|\mathcal{H}|2e^{-2\epsilon^2 m}$$

We want to make this no greater than δ. Solving $\delta = |\mathcal{H}|2e^{-2\epsilon^2 m}$ gives

$$\epsilon = \sqrt{\frac{1}{m} \ln \frac{2|\mathcal{H}|}{\delta}}$$

So with prob. at least $1 - \delta$, the difference btw. $\text{err}(h)$ and $\widehat{\text{err}}(h)$ is at most as above for all $h \in \mathcal{H}$.

Dilemma: A large $\mathcal{H}$ allows to achieve a small $\widehat{\text{err}}(h)$ but means a loose bound on $\text{err}(h)$.

Sample Complexity for Inconsistent Learning

Solving $\delta = |\mathcal{H}|2e^{-2\epsilon^2 m}$ instead for m gives

$$m = \frac{1}{2\epsilon^2} \ln \frac{2|\mathcal{H}|}{\delta}$$

which is thus a number of examples sufficient to make

$$|\mathrm{err}(h) - \widehat{\mathrm{err}}(h)| \leq \epsilon$$

with probability at least $1 - \delta$ for all $h \in \mathcal{H}$.

$m \leq \mathrm{poly}(1/\epsilon, 1/\delta, n)$ iff $\ln |\mathcal{H}| \leq \mathrm{poly}(n)$

Error Bound for ERM

Assume the learner returns

$$h_{\text{erm}} = \arg\min_{h \in \mathcal{H}} \widehat{\text{err}}(h)$$

This is known as the *empirical risk minimization* (ERM) learning principle.

Let $h^* = \arg\min_{h \in \mathcal{H}} \text{err}(h)$, i.e. h^* is the best hypothesis.

Let further $m = \frac{1}{2\epsilon^2} \ln \frac{2|\mathcal{H}|}{\delta}$. Then with prob. at least $1 - \delta$:

$$\begin{aligned}
\text{err}(h_{\text{erm}}) &\leq \widehat{\text{err}}(h_{\text{erm}}) + \epsilon && \text{which we just proved for any } h \in \mathcal{H} \\
&\leq \widehat{\text{err}}(h^*) + \epsilon && \text{because } h_{\text{erm}} \text{ minimizes } \widehat{\text{err}} \\
&\leq \text{err}(h^*) + 2\epsilon && \text{because } \widehat{\text{err}}(h^*) \leq \text{err}(h^*) + \epsilon
\end{aligned}$$

So the the ERM hypothesis is 'worse' than the best possible hypothesis by at most 2ϵ with prob. at most $1 - \delta$.

Bias-Variance Trade-Off

Plugging in the expression for ϵ and the error of h^*, we get that with prob. at least $1 - \delta$:

$$\text{err}(h_{\text{erm}}) \leq \min_{h \in \mathcal{H}} \text{err}(h) + 2\sqrt{\frac{1}{2m} \ln \frac{2|\mathcal{H}|}{\delta}}$$

Which shows a variant of the *bias-variance trade-off* known from statistical learning theory.

Large $\mathcal{H}$: small bias (first summand, we minimize over a large hypothesis class, can achieve a low minimum), large variance ("much to choose from", large $|\mathcal{H}|$ makes the second summand large). Too large $\mathcal{H}$ = *overfitting*.

Small $\mathcal{H}$: Opposite effects, *underfitting*.

The more training data (m), the larger $\mathcal{H}$ can be 'afforded' without overfitting.