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

To decide $x \in C$ for any given $x \in X$, the learner must be able to *compute* C, i.e., the function

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

- *Countable* number of computable concepts (any algorithm has a finite description so their number is countable)
- But *uncountable* number of concepts if X infinite, e.g. $X = N$
- $\rightarrow$ Non-computable concepts exist.

COLT studies the behavior of learners with respect to selected subsets $\mathcal{C} \subset 2^X$ called *concept classes*.

Hypothesis Class

A finite description of a learner's decision model is called a *hypothesis*. Learners use constrained languages (rules, polynomials, graphs, ...) to encode their hypotheses.

For example, the hypothesis

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

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

Hypothesis languages are typically not Turing-complete so not all computable concepts can be expressed by hypotheses.

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

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*.

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

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

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 $n' = \sum_{i=1}^{k} \binom{n}{i} 2^i \leq \mathrm{poly}(n)$ different clauses.

Introduce a new variable for each of the n' clauses and use an efficient learner to learn a 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 $n' \leq \mathrm{poly}(n)$.

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.,

$$(abc) \vee (de) \mathrel{\models\!\!\!\dashv} (a \vee d)(a \vee e)(b \vee d)(b \vee e)(c \vee d)(c \vee e)$$

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 $\{ 0, 1 \}^n$.

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 $h'_j > n$ (i.e., $h_j > n/2$) after an example x.
- $x_j = 1$ 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.