Învățare Automată (Machine Learning)

Bogdan Alexe,

bogdan.alexe@fmi.unibuc.ro

Master Informatică, anul I, 2018-2019, cursul 10

RAAI 2019

UNIVERSITATEA DIN BUCURESTI [RO] | https://conferences.unibuc.ro/raai2019/

CONFERENCE

ABOUT INVITED SPEAKERS SUBMISSION ORGANIZERS PROGRAM SPONSORS VENUE

JUNE 28-30, 2019, BUCHAREST

3rd Conference on Recent Advances in Artificial Intelligence

Invited Speakers

PRESLAV NAKOV
QATAR COMPUTING RESEARCH INSTITUTE

RAHUL SUKTHANKAR
GOOGLE RESEARCH AND CARNEGIE MELLON UNIVERSITY

TINNE TUYTELAARS
UNIVERSITY OF LEUVEN

JOAO CARREIRA
DEEPMIND

MATEUSZ MALINOWSKI
DEEPMIND

RADU TIMOFTE
ETH ZURICH

RAAI 2019 + EEML

This year's EEML Summer School is co-located with RAAI (Recent Advances in Artificial Intelligence) international conference, Bucharest, 28-30 June. We encourage EEML participants to attend RAAI as well. It has a great lineup of keynote speakers.

In addition, for those who missed the EEML application deadline or were unlucky in the selection, you have one more chance! ***We will provide up to 5 additional spots at the summer school to the best RAAI submissions***.

Check-out RAAI website for submission instructions (2 page extended abstract, template is provided, travel grants are available). Don't forget to mention in your submission if you wish to be considered for EEML 2019.

Deadline: May 20th.

Assignment 1 - submissions

- de fixat data de examen!!!

Assignment 1 - comments

- partial (not fully) solution provided on the internet

A. PAC Learning

- Let $X = \mathbb{R}^2$ with orthonormal basis (e_1, e_2) and consider the set of concepts defined by the area inside a right triangle ABC with two sides parallel to the axes, with $\overrightarrow{AB}/AB = e_1$ and $\overrightarrow{AC}/AC = e_2$, and $AB/AC = \alpha$ for some positive real $\alpha \in \mathbb{R}_+$. Show, using similar methods to those used in the lecture slides for the axis-aligned rectangles, that this class can be (ϵ, δ)-PAC-learned from training data of size $m \geq (3/\epsilon)\log(3/\delta)$.

  As in the case of axis-aligned rectangles, consider three regions r_1, r_2, r_3, along the sides of the target concept as indicated in Figure 1. Note that the triangle formed by the points $A"$, $B"$, $C"$ is similar to ABC (same angles) since $A"B"$ must be parallel to AB, and similarly for the other sides.

  Assume that $\Pr[ABC] > \epsilon$, otherwise the statement would be trivial. Consider a triangle $A'B'C'$ similar to ABC and consistent with the training sample and such that it meets all three regions r_1, r_2, r_3.

  Since it meets r_1, the line $A'B'$ must be below $A"B"$. Since it meets r_2 and r_3, A' must be in r_2 and B' in r_3 (see Figure 1). Now, since the angle $\widehat{A'B'C'}$ is equal to $\widehat{A"B"C"}$, C' must be necessarily above $C"$. This implies that triangle $A'B'C'$ contains $A"B"C"$ and thus $error(A'B'C') \leq \epsilon$.

$$error(A'B'C') > \epsilon \implies \exists i \in \{1,2,3\} : A'B'C' \cap r_i = \emptyset.$$

  Thus, by the union bound,

$$\Pr[error(A'B'C') > \epsilon] \leq \sum^{3} \Pr[A'B'C' \cap r_i = \emptyset] \leq 3(1-\epsilon/3)^m \leq 3e^{-3m\epsilon}.$$

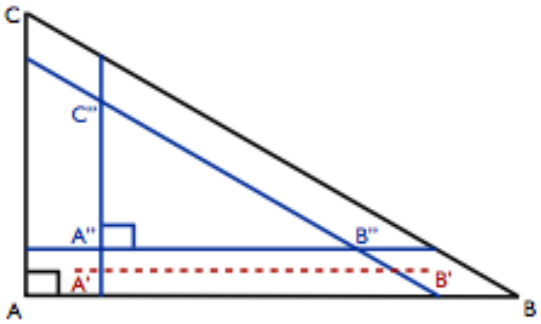


Figure 1: Rectangle triangles.

Recap – Randomised algorithms

- a randomised algorithm A is allowed to use random numbers as part of its input

- a randomised algorithm A 'solves' a binary decision problem (YES/NO) if it behaves in the following way:
  - the algorithm always halts and produces an output
  - if the output of A is YES then the answer is 100% right
  - if the output of A is NO then the answer is right with probability $\geq 50\%$

- randomised algorithm for the problem "number n is composite (is not prime)"
  - if the output is YES then we are sure that number n is composite (not prime)
  - if the output is NO then with probability $\geq 50\%$ number n is not composite (is prime)

Theorem

Let $\mathcal{H}$ be a hypothesis class that is efficient PAC learnable. Then, there exists a randomised algorithm which solves the problem of finding a hypothesis in $\mathcal{H}$ consistent with a given training sample, and which has runtime polynomial in m (the length of the training sample).

Recap – intractability of learning

There exist classes that are PAC learnable for which the sample complexity is polynomial but the runtime is not polynomial.

Consider $\mathcal{H}_{3\text{DNF}}^{d}$ = class of 3-term disjunctive normal form formulae consisting of hypothesis of the form h: $\{0,1\}^d \rightarrow \{0,1\}$,

$$h(\mathbf{x}) = A_1(\mathbf{x}) \vee A_2(\mathbf{x}) \vee A_3(\mathbf{x}),$$

where $A_1(\mathbf{x})$ is a Boolean conjunction (in $\mathcal{H}_{\text{conj}}^{d}$) with at most d Boolean literals $x_1, \ldots, x_d$.

$|\mathcal{H}_{3\text{DNF}}^{d}| = 3^{3d} < \infty$, so is PAC learnable with sample complexity $3d \log(3/\delta)/\varepsilon$ (polynomial in $1/\varepsilon$, $1/\delta$, d).

However, from the computational perspective, this learning problem is hard. There is no polynomial time algorithm (unless RP = NP) that *properly* learns from training data. So ERM_H is not efficient. If $\mathcal{H}_{3\text{DNF}}^{d}$ is efficient PAC learnable then the problem of graph 3-coloring problem (which is shown to be NP-complete) is in RP.

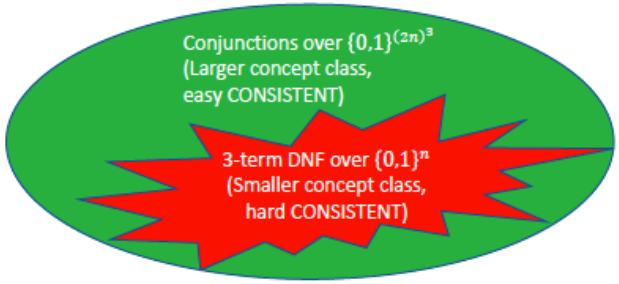
Recap - Improper learning of 3-term DNFs

Use the distribution rule to obtain: $(a \wedge b) \vee (c \wedge d) = (a \vee b) \wedge (a \vee d) \wedge (b \vee c) \wedge (b \vee d)$
A 3-term DNF formula can be written as a 3-CNF formulae:

$$A_1 \vee A_2 \vee A_3 = \bigwedge_{u \in A_1, v \in A_2, w \in A_3} (u \vee v \vee w)$$

So we have that a 3-term DNF can be viewed as a conjunction of $(2n)^3$ variables:

$$H^n_{3DNF} \subseteq H^{(2n)^3}_{conj} \qquad \left|H^{(2n)^3}_{conj}\right| = 3^{(2n)^3} + 1$$



We can efficiently PAC learn the new class of conjunctions, $H^{(2n)^3}_{conj}$ with sample complexity $(n^3+\log(1/\delta))/\varepsilon$. The overall runtime of this approach is polynomial in $1/\varepsilon$, $1/\delta$, n. Pay polynomially in sample complexity, gain exponentially in computational complexity.

	3-term DNF	Conjunctions over $\{0,1\}^{(2n)^3}$
Sample complexity	$O\left(\frac{n+\log\frac{1}{\delta}}{\varepsilon}\right)$	$O\left(\frac{n^3+\log\frac{1}{\delta}}{\varepsilon}\right)$ ↑
CONSISTENT Computational-Complexity	NP-Hard	$O\left(n^3 \times \frac{n^3+\log\frac{1}{\delta}}{\varepsilon}\right)$ ↓

Today's lecture: Overview

- Hardness of learning
- Boosting

Hardness of learning

Some classes are hard to PAC learn if we place certain restrictions on the hypothesis class used by the learning algorithm

- the problem of properly learning 3-term DNF formulae is computationally hard (the learning algorithm is restricted to output a hypothesis of the class $\mathcal{H}_{3\text{DNF}}{}^{\text{d}}$)
- this problems can be solved efficiently by letting the learning algorithm to output a hypothesis from the class $H_{conj}^{(2n)^3}$
- representation dependent hardness

Interesting and fundamental question regarding the PAC learning model:

- are there any classes that are computationally hard to learn, independent of the representation used?
- we are interested in the existence of classes with polynomial VC dimension (such that we need polynomial number of training examples to PAC learn them – thus is no information-theoretic barrier to fast learning), yet there is no algorithm with runtime polynomial.
- how to prove that a class is computational hard, independent of its representation?

Learning vs. cryptography

In some sense, cryptography is the opposite of learning:

- in learning we try to uncover some rule underlying the examples we see
- in cryptography, the goal is to make sure that nobody will be able to discover some secret, in spite of having access to some partial information about it.

Results about the cryptographic security of some system translate into results about the un-learnability of some corresponding task.

The common approach for proving that cryptographic protocols are secure is to start with some cryptographic assumptions. It is our belief that they really hold.

Deduce hardness of learnability from cryptographic assumptions.

Cryptographic assumptions

- there exists a ***one way function*** $f : \{0,1\}^n \rightarrow \{0,1\}^n$ (more formally, f it is a sequence of functions, one for each dimension *n*) that is ***easy to compute but is hard to invert***. More formally, f can be computed in time *poly(n)* but for any randomized polynomial time algorithm A, and for every polynomial p(), A cannot learn from pairs $(f(\mathbf{x}), \mathbf{x}) = (f(\mathbf{x}), f^{-1}(f(\mathbf{x})))$

$$\mathbb{P}[f(A(f(\mathbf{x}))) = f(\mathbf{x})] < \frac{1}{p(n)},$$

(the probability is taken over a random choice of **x** in $\{0, 1\}^n$ and the randomness of A)

- a one way function, f, is ***called trapdoor one way function*** if, for some polynomial function *p*, for every *n* there exists a bit-string s_n (called a secret key) of length $\leq p(n)$, such that there is a polynomial time algorithm that, for every *n* and every $\mathbf{x} \in \{0, 1\}^n$, on input $(f(\mathbf{x}), s_n)$ outputs x . In other words, ***although f is hard to invert, once one has access to its secret key, inverting f becomes feasible***. Such functions are parameterized by their secret key.

A hard to learn class

Let $F_n = \{f : \{0,1\}^n \rightarrow \{0,1\}^n$ is a trapdoor function$\}$that can be calculated by a polynomial time algorithm.

Consider the task of learning the class of the corresponding inverses, $\mathcal{H}_F^n = \{ f^{-1} : f \in F_n \}$.

Since each function in this class can be inverted by some secret key s_n of size polynomial in n, the class $\mathcal{H}_F^n$ can be parameterized by these keys and its size is at most $2^{p(n)}$. So, the $Vcdim(\mathcal{H}_F^n) \leq p(n)$, so its sample complexity is therefore polynomial in n .

We show that there can be no efficient learner for this class.
If there were such a learner, L, then by sampling uniformly at random a polynomial number of strings in $\{0, 1\}^n$, and computing f over them, we could generate a labeled training sample of pairs (f(**x**), **x**), which should suffice for our learner to figure out an (ε, δ)-approximation of f^{-1} (w.r.t. the uniform distribution over the range of f), which would violate the one way property of f.

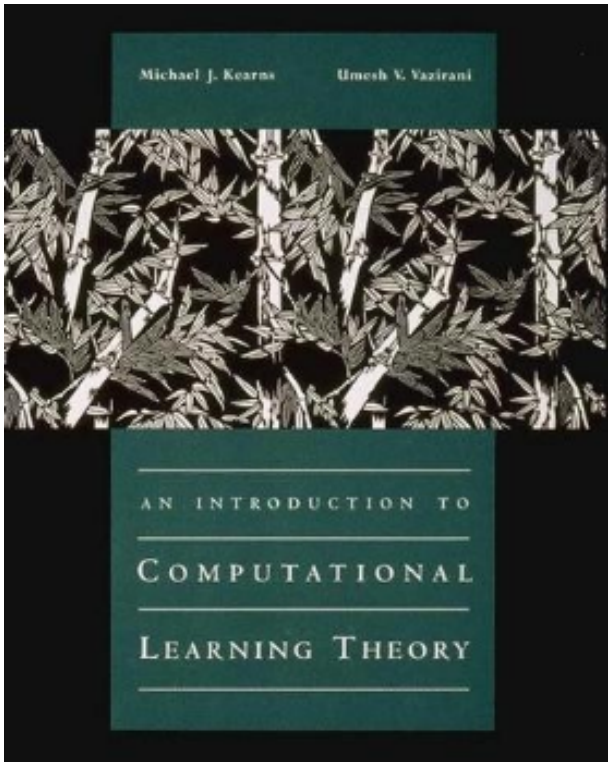
Other examples

Discrete Cube Root Problem. Let p and q be two primes. And 3 does not divide $(p-1)(q-1)$. Let $N = pq$. Then Let $x \in Z^*$ and $x \leq N$. Let $f_N(x) = x^3 \ mod \ N$. The problem is that given N and $y = f_N(x)$, find x.

For any polynomial $p(n)$, there is no algorithm such that,

1. That runs in time $p(n)$ and
2. On input $N = pq$ from two randomly chosen n-bit primes p and q, such that 3 does not divide $(p-1)(q-1)$ and input $y \neq 0$ s.t. neither p nor q can divide y, chose uniformly at random and returns x, such that $x^3 \ mod \ N = y$. w.p$\geq \frac{1}{p(n)}$ (over p, q, y and algorithm choices)

Kearns & Vazirani, MIT Press:
An Introduction to Computational Learning Theory



Boosting

Overview of Boosting

Boosting = general method of converting simple (weak) classifiers (better than chance) into highly accurate prediction rule

Main idea of Boosting:

- assume given "weak" learning algorithm that uses a simple "rule of thumb" to output a hypothesis that comes from an easy-to-learn hypothesis class and performs just slightly better than a random guess

- a boosting algorithm amplifies the accuracy of weak learners by aggregating such weak classifiers to approximates gradually good predictors for larger, and complex, classes.

Boosting is a great example for the practical impact of learning theory. While boosting originated as a purely theoretical problem, it has led to popular and widely used algorithms. It has been successfully used for learning to detect faces in images.

PAC Learnability = Strong learnability

Remember the definition of a class $\mathcal{H}$ being PAC Learnable:

A hypothesis class $\mathcal{H}$ is called ***PAC learnable*** if there exists a function $m_{\mathcal{H}}$: $(0,1)^2 \rightarrow N$ and a learning algorithm A with the following property:

- for every $\varepsilon > 0$ (*accuracy* → *"approximately correct"*)
- for every $\delta > 0$ (*confidence* → *"probably"*)
- for every labeling $f \in \mathcal{H}$ (*realizability case*)
- for every distribution $\mathcal{D}$ over $\mathcal{X}$

when we run the learning algorithm A on a training set, consisting of $m \geq m_{\mathcal{H}}(\varepsilon, \delta)$ examples sampled i.i.d. from $\mathcal{D}$ and labeled by f the algorithm A returns a hypothesis $h \in \mathcal{H}$ such that, with probability at least $1-\delta$ (over the choice of examples), $L_{D,f}(h) \leq \varepsilon$.

Given sufficiently many examples we can learn a classifier from $\mathcal{H}$ with arbitrary small generalization error ε and with arbitrary high confidence $1-\delta$.
Strong learnability = ability to learn a classifier with arbitrary small generalization error

Weak learnability

Definition (γ-Weak-Learnability)
A learning algorithm, A, is a γ-weak-learner for a class $\mathcal{H}$ if there exists a function $m_{\mathcal{H}}:(0,1)\rightarrow N$ such that:

- for every $\delta > 0$ (*confidence*)
- for every labeling $f \in \mathcal{H}, f : \mathcal{X} \rightarrow \{-1,+1\}$ (*realizability case*)
- for every distribution $\mathcal{D}$ over $\mathcal{X}$

when we run the learning algorithm A on a training set, consisting of $m \geq m_{\mathcal{H}}(\delta)$ examples sampled i.i.d. from $\mathcal{D}$ and labeled by *f,* the algorithm A returns a hypothesis *h* (*h* might not be from $\mathcal{H}$ - improper learning) such that, with probability at least $1-\delta$ (over the choice of examples), $L_{\mathcal{D},f}(h) \leq 1/2 - \gamma$.

A hypothesis class $\mathcal{H}$ is γ-weak-learnable if there exists a γ-weak-learner for that class.

Weak vs Strong learnability

Strong learnability implies the ability to find an arbitrarily good classifier (with error rate at most ε for an arbitrarily small $\varepsilon > 0$).

In weak learnability, however, we only need to output a hypothesis whose error rate is at most $1/2-\gamma$, namely, whose error rate is slightly better than what a random labeling would give us.

The hope is that it may be easier to come up with ***efficient*** weak learners than with efficient (strong) PAC learners. If we have access to an ***efficient*** weak learner, can we use it to build an ***efficient*** strong learner?

We will see that strong learnability $\Leftrightarrow$ weak learnability

$\Rightarrow$ easy to show, based on the definition

$\Leftarrow$ use boosting (improper learning algorithm) that combines weak learners to obtain a strong learner.

If the learning problem is hard, boosting cannot help as we can't find efficient weak learners.

Weak vs Strong learnability

The fundamental theorem of learning (lecture 7) states that if a hypothesis class $\mathcal{H}$ has a VC dimension d, then the sample complexity of PAC learning $\mathcal{H}$ satisfies

$$C_1 \frac{d + \log(1/\delta)}{\epsilon} \leq m_{\mathcal{H}}(\epsilon, \delta) \leq C_2 \frac{d \log(1/\epsilon) + \log(1/\delta)}{\epsilon}$$

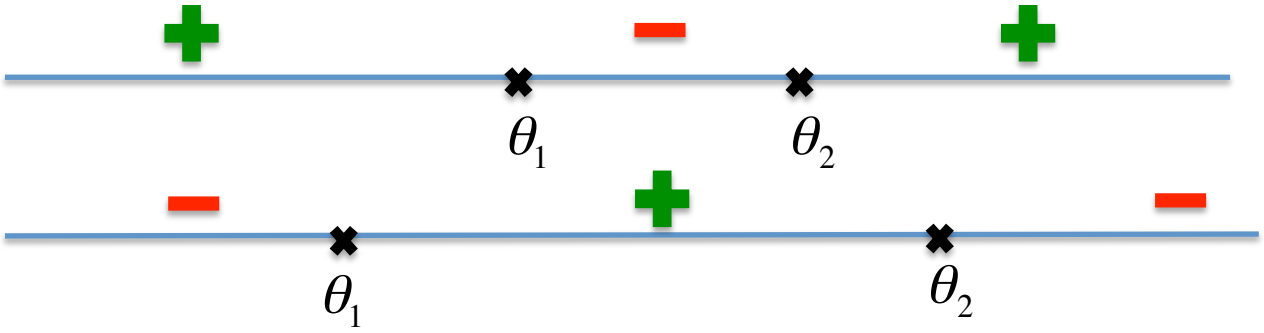
Applying this with $\varepsilon = 1/2 - \gamma$ we immediately obtain that if $d = \infty$ then $\mathcal{H}$ is not γ -weak-learnable. This implies that ***from the statistical perspective*** (i.e., if we ignore computational complexity), ***weak learnability*** is also characterized by the VC dimension of $\mathcal{H}$ and therefore ***is just as hard as PAC (strong) learning.***

However, when we do consider computational complexity, the potential advantage of weak learning is that maybe there is an algorithm that satisfies the requirements of weak learning and can be implemented efficiently.

Weak learnability - example

Let $\mathcal{X} = R$, $\mathcal{H}$ is the class of 3-piece classifiers (signed intervals):

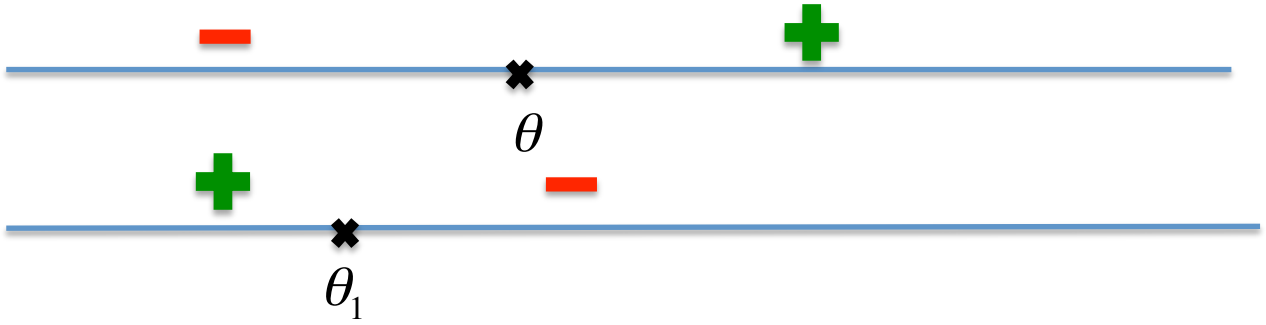
$$H = \{h_{\theta_1,\theta_2,b} \mid \theta_1,\theta_2 \in R, \theta_1 < \theta_2, b \in \{-1,+1\}\} \quad h_{\theta_1,\theta_2,b}(x) = \begin{cases} +b, \text{ if } x < \theta_1 \text{ or } x > \theta_2 \\ -b, \text{ if } \theta_1 \le x \le \theta_2 \end{cases}$$



b = 1

b = -1

Consider $\mathcal{B}$ the class of Decision Stumps = class of 1-node decision trees
$\mathcal{B} = \{h_{\theta,b}: \mathbf{R} \to \{-1,1\}, h_{\theta,b}(x) = \text{sign}(x - \theta) \times b, \theta \in \mathbf{R}, b \in \{-1,+1\}\}$.



b = 1

b = -1

$\text{ERM}_{\mathcal{B}}$ is a γ-weak learner for $\mathcal{H}$, for $\gamma < 1/6$.

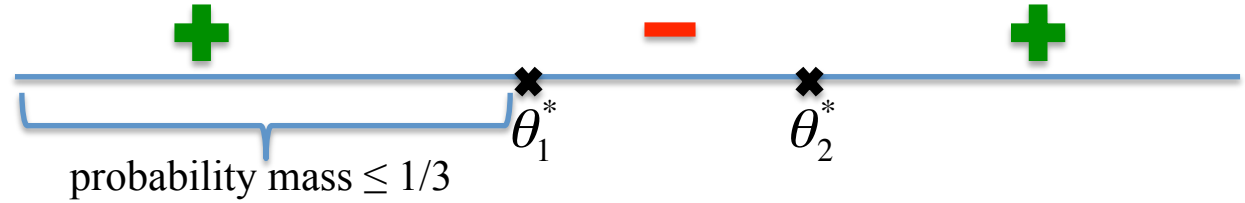
Weak learnability - example

$\mathrm{ERM}_{\mathcal{B}}$ is a γ-weak learner for $\mathcal{H}$, for $\gamma < 1/6$.

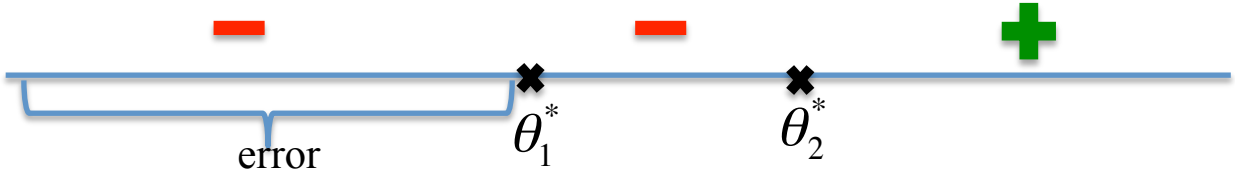
Proof: Consider a $h^* = h_{\theta_1^*, \theta_2^*, b^*} \in \mathcal{H}$ that labels a training set S.

Consider a distribution $\mathcal{D}$ over $\mathcal{X} = \mathbf{R}$. Then, we are sure that at least one of the regions $(-\infty, \theta_1^*)$, $[\theta_1^*, \theta_2^*]$, $(\theta_2^*, +\infty)$ has a probability mass wrt $\mathcal{D} \leq 1/3$.

Consider, without loss of generality that $\mathcal{D}((-\infty, \theta_1^*)) = \mathrm{P}_{x \sim \mathcal{D}}(x \in (-\infty, \theta_1^*)) \leq 1/3$
Then the hypothesis $h_{\theta,b} \in \mathrm{B}$, where $\theta = \theta_2^*$, $b = b^*$ errors on $(-\infty, \theta_1^*)$.



$h_{\theta_1^*, \theta_2^*, b^*} \in \mathcal{H}$



$h_{\theta_2^*, b^*} \in \mathcal{B}$

Weak learnability - example

$\mathcal{B} = \{h_{\theta,b}: \mathbf{R} \rightarrow \{-1,1\}, h_{\theta,b}(x) = \text{sign}(x - \theta) \times b, \theta \in \mathbf{R}, b \in \{-1,+1\}\}$.

It is easy to show that VCdim($\mathcal{B}$) = VCdim(class of signed thresholds) = 2.

The fundamental theorem of learning states that if the hypothesis class $\mathcal{B}$ has VCdim($\mathcal{B}$) = d, then the sample complexity of agnostic PAC learning $\mathcal{B}$ satisfies:

$$C_1 \frac{d + \log(1/\delta)}{\epsilon^2} \leq m_{\mathcal{H}}(\epsilon, \delta) \leq C_2 \frac{d + \log(1/\delta)}{\epsilon^2}$$

So, if the sample size is greater than the sample complexity, then with probability of at least $1 - \delta$, the $\text{ERM}_{\mathcal{B}}$ rule learns in the agnostic case a hypothesis such that:

$$L_{\mathcal{D}}(\text{ERM}_B(S)) \leq \min_{h \in \mathcal{H}} L_{\mathcal{D}}(h) + \epsilon = 1/3 + \varepsilon$$

Take ε such that $1/3 + \varepsilon < 1/2$, $\varepsilon < 1/6$. Than $\text{ERM}_{\mathcal{B}}$ is a γ-weak learner for $\mathcal{H}$, where $\gamma = \varepsilon$.

Efficient implementation of ERM for Decision Stumps

In practice, we use the following base hypothesis class of decision stumps over $\mathbf{R}^d$ for weak learners:

$\mathcal{H}_{DS}^d = \{h_{i,\theta,b}: \mathbf{R}^d \rightarrow \{-1,1\}, h_{i,\theta,b}(\mathbf{x}) = \text{sign}(\theta - x_i) \times b, 1 \leq i \leq d, \theta \in \mathbf{R}, b \in \{-1,+1\}\}$

-pick a coordinate i (from 1 to d), project the input $\mathbf{x} = (x_1, x_2, \ldots, x_d)$ on the i-th coordinate and obtain x_i, if $x_i \leq$ threshold θ label the example with b, else with –b

How to implement efficient ERM rule for the class $\mathcal{H}_{DS}^d$?

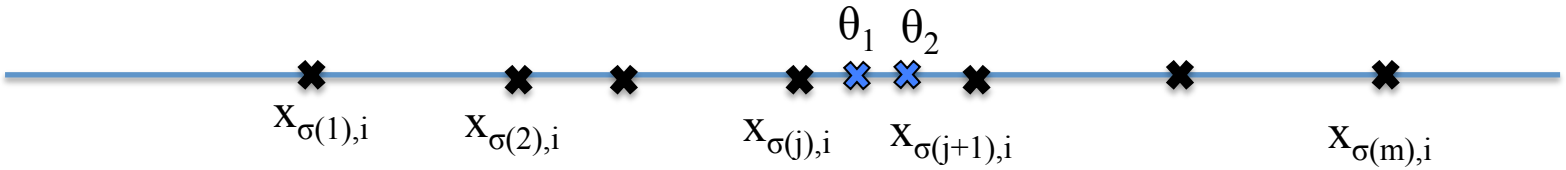
Let $S = ((\mathbf{x_1}, y_1), \ldots, (\mathbf{x_m}, y_m))$ be a training set of size *m*. We want to find the best h_{i^*,θ^*,b^*} which minimizes the training error on S:

$$h_{i^*,\theta^*,b^*} = \operatorname*{argmin}_{h_{i,\theta,b} \in H_{DS}^d} L_S(h_{i,\theta,b}) = \operatorname*{argmin}_{\substack{1 \leq i \leq d \\ \theta \in R \\ b \in \{-1,+1\}}} L_S(h_{i,\theta,b})$$

Efficient implementation of ERM for Decision Stumps

We have $1 \leq i \leq d$, $\theta \in \mathbf{R}$, $b \in \{-1,+1\}$. We fix $i \in \{1, 2, \ldots, d\}$ and $b \in \{-1,+1\}$. Then we are interested in minimizing the error on $S_i = ((x_{1,i}, y_1), \ldots, (x_{m,i}, y_m))$. By sorting $x_{1,i}, x_{2,i}, \ldots, x_{m,i}$ we obtain $x_{\sigma(1),i} \leq x_{\sigma(2),i} \leq \ldots \leq x_{\sigma(m),I}$

We have that $\theta \in \mathbf{R}$. Pick θ_1 and θ_2 in $[x_{\sigma(j),i}, x_{\sigma(j+1),i})$



We see that $h_{i,\theta 1,b}$ and $h_{i,\theta 2,b}$ have the same error. For all $\theta \in [x_{\sigma(j),i}\ x_{\sigma(j+1),i})$ we obtain the same error, all the hypothesis $h_{i,\theta,b}$ are very similar.

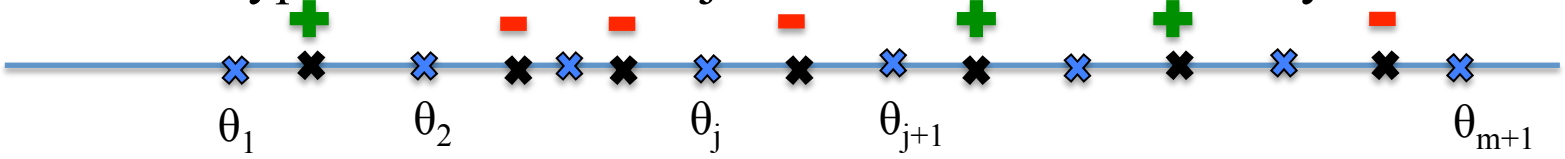
In fact, we can restrict the search over all $\theta \in \mathbf{R}$ just to m + 1 values of θ, chosen in the intervals $(-\infty, x_{\sigma(1),i})$, $[x_{\sigma(1),i}, x_{\sigma(2),i})$, …, $[x_{\sigma(m),i}, +\infty)$ by taking a representative threshold in each interval. For example we can take the set of representative thresholds as containing extreme points + middle of the segments: $\Theta_i = \{x_{\sigma(1),i}-1, 1/2 \times (x_{\sigma(1),i} + x_{\sigma(2),i}), \ldots, 1/2 \times (x_{\sigma(m-1),i} + x_{\sigma(m),i}), x_{\sigma(m),i}+1\}$

Efficient implementation of ERM for Decision Stumps

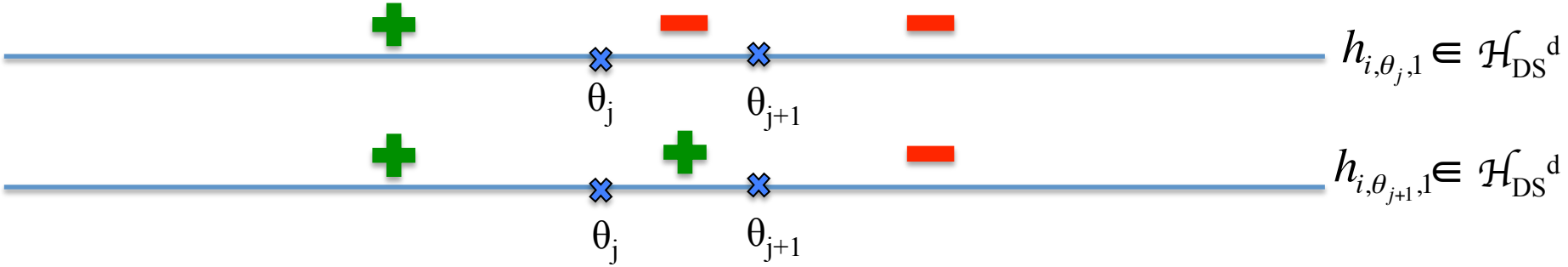
$\mathcal{H}_{DS}^{d} = \{h_{i,\theta,b}: \mathbf{R}^d \rightarrow \{-1,1\}, h_{i,\theta,b}(\mathbf{x}) = \text{sign}(\theta - x_i) \times b, 1 \le i \le d, \theta \in \mathbf{R}, b \in \{-1,+1\}\}$

We have $1 \le i \le d$, $\theta \in \Theta_i$, $|\Theta_i| = m+1$, $b \in \{-1,+1\}$. So we have $d \times (m+1) \times 2$ possible hypothesis. Each hypothesis takes O(m) runtime. So the runtime is polynomial.

You can decrease the entire runtime using dynamic programming as computing the loss of two hypothesis for two adjacent threshold can be recycled.



Suppose b = 1. Then $L_S(h_{i,\theta_{j+1},1}) = L_S(h_{i,\theta_j,1}) - \frac{1}{m} y_j$



Efficient implementation of ERM for Decision Stumps

$\mathcal{H}_{DS}^{d} = \{h_{i,\theta,b}: \mathbf{R}^d \rightarrow \{-1,1\}, h_{i,\theta,b}(\mathbf{x}) = \text{sign}(\theta - x_i) \times b, 1 \leq i \leq d, \theta \in \mathbf{R}, b \in \{-1,+1\}\}$

input: training set $S = ((\mathbf{x_1}, y_1), \ldots, (\mathbf{x_m}, y_m))$ of size m
goal: find h_{i^*,θ^*,b^*} which minimizes the training error on S
initialize: $L^* = +\infty$
for $b = -1,+1$
  for $i = 1, \ldots, d$
    sort S on the i-th coordinate, obtain $x_{\sigma(1),i} \leq x_{\sigma(2),i} \leq \ldots \leq x_{\sigma(m),i}$
    take $\Theta_i = \{\theta_1, \theta_2, \ldots, \theta_{m+1}\} = \{x_{\sigma(1),i}-1, \ldots 1/2 \times (x_{\sigma(j),i} + x_{\sigma(j+1),i}), \ldots, x_{\sigma(m),i}+1\}$
    compute current loss $L_{current} = L_S(h_{i,\theta_1,b})$
    if $L_{current} < L^*$
      $L^* = L_{current}$, $i^* = i$, $\theta^* = \theta_1$, $b^* = b$
    for $j = 1, \ldots, m$
      compute $L_{current} = L_S(h_{i,\theta_{j+1},b}) = L_S(h_{i,\theta_j,b}) - \frac{1}{m} b y_j$
      if $L_{current} < L^*$
        $L^* = L_{current}$, $i^* = i$, $\theta^* = \theta_1$, $b^* = b$
output h_{i^*,θ^*,b^*}
Runtime: $O(2 \times d \times (m \times \log_2 m + m)) = O(d \times m \times \log_2 m)$

A formal description of boosting

- given training set S = (($\mathbf{x_1}$, y_1), …, ($\mathbf{x_m}$, y_m)) of size *m*
- $y_i \in \{-1, +1\}$ correct label of instance $x_i \in X$

- for t = 1,...,T (number of rounds):
  - construct distribution $\mathbf{D}^{(t)}$ on {1,..., m}

  - find weak classifier ("rule of thumb") $h_t : \mathcal{X} \rightarrow \{-1,+1\}$ with error ε_t on $\mathbf{D}^{(t)}$:

$$\varepsilon_t = \Pr_{i \sim D^{(t)}}[h_t(x_i) \neq y_i]$$

- output final/combined classifier h_{final}

Each round involves building the distribution $\mathbf{D}^{(t)}$ as well as a single call to the weak learner. Therefore, if the weak learner can be implemented efficiently (as happens in the case of ERM with respect to decision stumps – improper learning) then the total training process will be efficient.
Different variants of boosting comes from constructing distribution $\mathbf{D}^{(t)}$ + obtaining the final classifier h_{final}

First boosting algorithms

- [Schapire ‘89]:
  - first provable boosting algorithm
- [Freund ‘90]:
  - “optimal” algorithm that “boosts by majority”
- [Drucker, Schapire & Simard ‘92]:
  - first experiments with boosting
  - limited by practical drawbacks
- [Freund & Schapire ‘95]:
  - introduced “AdaBoost” algorithm
  - strong practical advantages over previous boosting algorithms

AdaBoost

- construct distribution $\mathbf{D}^{(t)}$ on {1,..., m}:
  - $\mathbf{D}^{(1)}(i) = 1/m$
  - given $\mathbf{D}^{(t)}$ and h_t: $D^{(t+1)}(i) = \frac{D^{(t)}(i)}{Z_{t+1}} \times e^{-w_t h_t(x_i) y_i}$

  where Z_{t+1} normalization factor (s. t. $\mathbf{D}^{(t+1)}$ is a distribution), $w_t = \frac{1}{2}\ln(\frac{1}{\varepsilon_t} - 1) > 0$

  If example x_i is correctly classified then $h_t(x_i) = y_i$ so at the next iteration t+1 its importance (probability distribution) will be decreased to $D^{(t+1)}(i) = \frac{D^{(t)}(i)}{Z_{t+1}} \times e^{-w_t}$

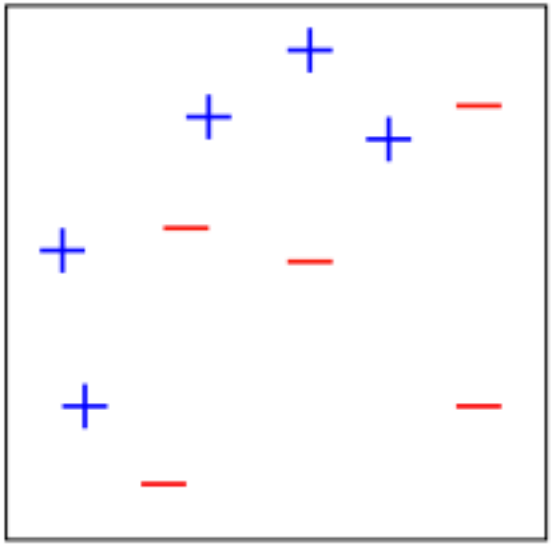
  If example x_i is misclassified then $h_t(x_i) \neq y_i$ so at the next iteration t+1 its importance (probability distribution) will be increased to $D^{(t+1)}(i) = \frac{D^{(t)}(i)}{Z_{t+1}} \times e^{w_t}$

- output final/combined classifier h_{final}:

$$h_{final}(x) = sign(\sum_{t=1}^{T} w_t h_t(x))$$

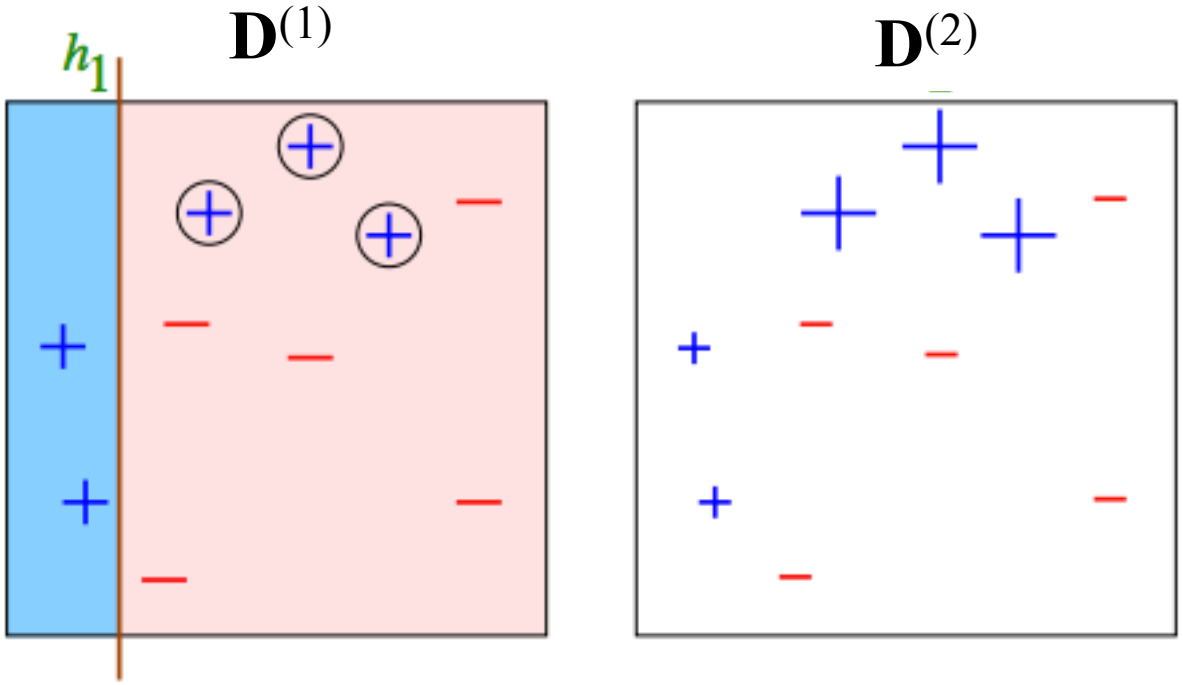
Toy example

$\mathbf{D}^{(1)}$



Weak classifiers = vertical or horizontal half- planes = hypothesis from $\mathcal{H}_{DS}{}^{2}$

Toy example – Round 1

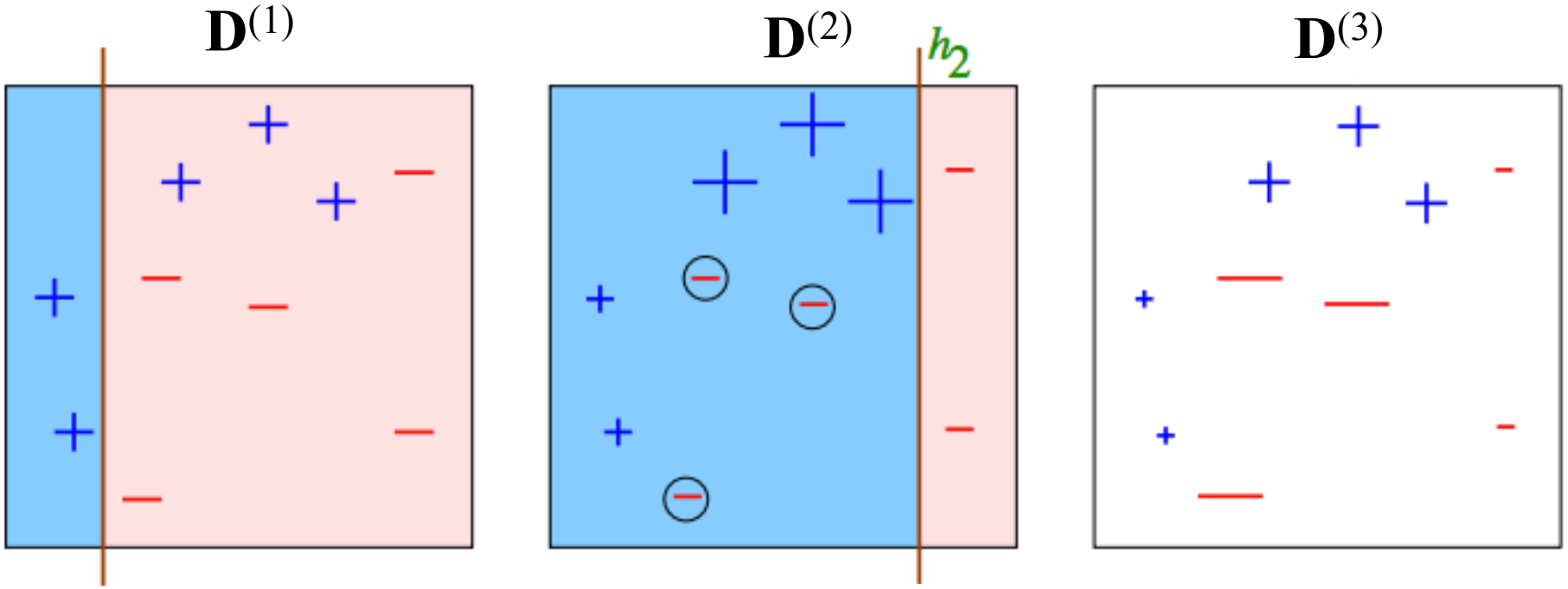


$\varepsilon_1 = 0.30$

$w_1 = 0.42$

Weak classifiers = vertical or horizontal half- planes = hypothesis from $\mathcal{H}_{DS}{}^2$

Toy example – Round 2

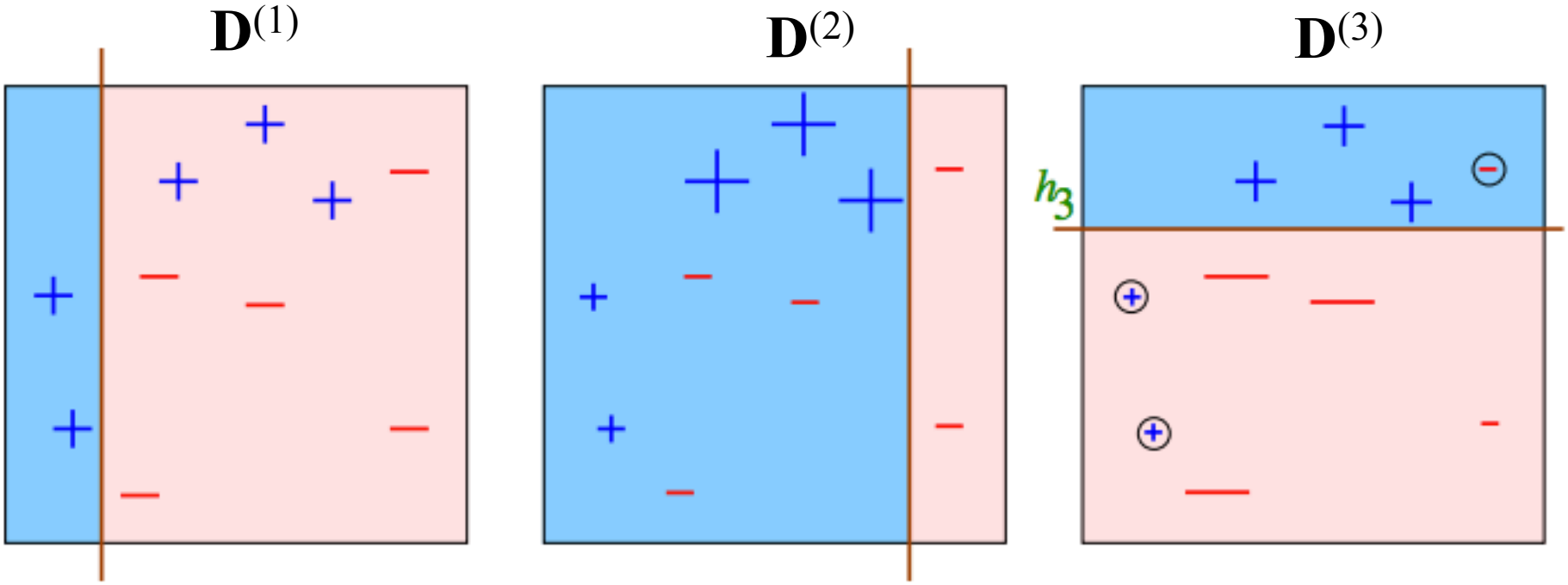


$\varepsilon_2 = 0.21$

$w_2 = 0.65$

Weak classifiers = vertical or horizontal half- planes = hypothesis from $\mathcal{H}_{DS}{}^2$

Toy example – Round 3

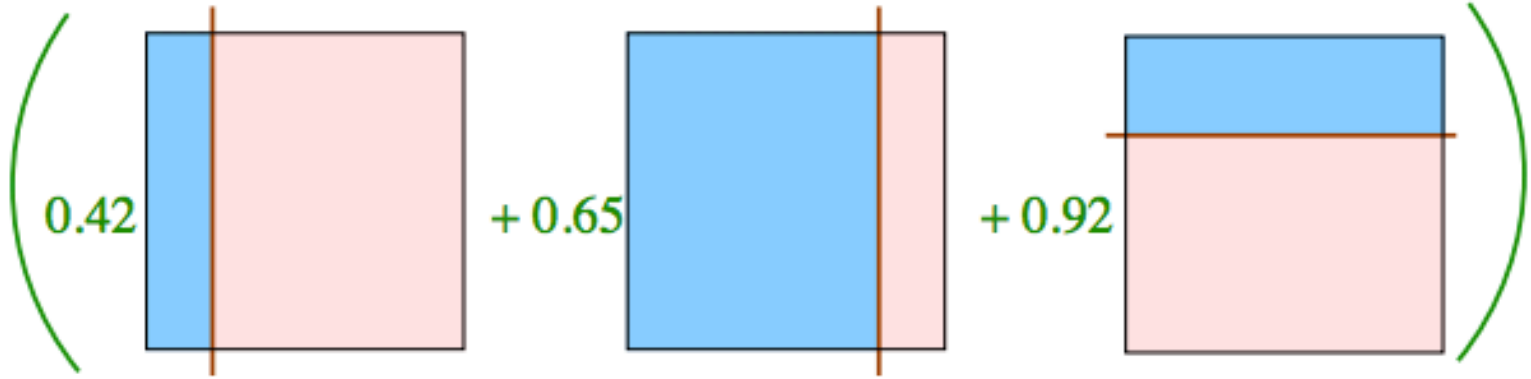


$\varepsilon_3 = 0.14$
$w_3 = 0.92$

Weak classifiers = vertical or horizontal half- planes = hypothesis from $\mathcal{H}_{DS}^2$

Toy example – final classifier

$H_{\text{final}} = \text{sign}\left(0.42 \; + 0.65 \; + 0.92 \right)$



=