Advanced Machine Learning

Bogdan Alexe,

bogdan.alexe@fmi.unibuc.ro

University of Bucharest, 2nd semester, 2019-2020

Assignment 1 - submissions

Nr. crt.	Numele, inițiala și prenumele candidatului	Grupa	Nr. crt.	Numele, inițiala și prenumele candidatului	Grupa
1	ADAM G. IULIANA-ALEXANDRA	407	39	MANDREȘI P. GEORGE-ADRIAN	407
2	AGARICI A.V. EDUARD	407	40	MANOLACHE C. ANDREI-MARIAN	407
3	AVRAM S. ANDREI-ALEXANDRU	407	41	MANOLESCU GH. COSMIN- MIHAI	411
4	BANU C. CĂTĂLIN	407	42	MARE N. TUDOR-ALEXANDRU	407
5	BARBU I. ALEX	407	43	MARIN N.T. TIBERIU	407
6	BEJAN G. MATEI	407	44	MATEI G. MIHAI	411
7	BELCINEANU V. ALEXANDRU-IOAN	407	45	MELINTE F.M. ANA-MARIA	407
8	BENEDIC B. MATEI	411	46	MEREUȚĂ A. ȘTEFAN	411
9	BERBEC V. DELIA-CRISTINA	411	47	MICLUȚĂ-CÂMPEANU M.A. MARIUS	407
10	BEȘLEAGĂ T. NELI-MONICA	411	48	MOCANU N. ALEXANDRA-MIRELA	411
11	BLESSING EMMANUEL	407	49	MULȚESCU A.C. ANDREI-NICOLAE	407
12	BLIDĂRESCU F. BOGDAN-FLORIN	407	50	MUNTEANU V. ANDREI	411
13	BOMHER R. SEBASTIAN	407	51	NĂSTASE G.D. ȘTEFAN	407
14	CĂLIN G. MĂDĂLINA-ANDREEA	407	52	NECULAI R.A.S. VLAD	407
15	CĂLINESCU G. VALENTIN-GELU	407	53	NEDELCU C. VLAD	411
16	CĂLINOIU R. MARIA-IRENE	407	54	NISTOR I.MIHAELA	407
17	CERNAT C. CĂTĂLIN -ȘTEFAN	411	55	NUȚ I.C. LUCIAN-VIRGIL	411
18	CHITIC M.F. IOANA-ANDREEA	407	56	OLARU F.M. BOGDAN-IOAN	407
19	CÎRSTOIU I. ANDREEA-IOANA	411	57	OROS N.R. ȘTEFAN	407
20	CIUCARDEL I.G. ANDREI	407	58	PĂCIOIANU C. MIHAI-DAVID	407
21	CROITORU F. FLORINEL-ALIN	407	59	PANAIT V. CORINA-MARIA	411
22	DASCĂLU C. ȘTEFAN	407	60	PETRE I. CRISTOFOR-IONUȚ	411
23	DIACONU F. ALEXANDRA	407	61	PINCU M. MIHAI-CĂTĂLIN	407
24	DOSPRA N. CRISTIAN-VASILE	407	62	POESINA M. EDUARD-GABRIEL	407
25	DRANCA V. CONSTANTIN	407	63	PROCOP G.VLADIMIR-ALEXANDRU	411
26	DUMITRIU G. ANDREI	407	64	RĂDOIU I. VALERIU-TONI	407
27	DUȚĂ I. GEORGIAN-EMILIAN	407	65	RADU M. MARIUS-GABRIEL	407
28	ENICĂ M. DIANA -MARIA	407	66	ROGOZ O. ANA-CRISTINA	407
29	FICUTA RAZVAN	506	67	ȘANDRU S.C. ADRIAN	407
30	GAVRILĂ L. ALEXANDRU	407	68	STANCU P. ROBERT-GABRIEL	407
31	GHINEA A. ALEXANDRU-ȘTEFAN	407	69	ȘTIRBU A ELENA-AURELIA	411
32	IONESCU EDUARD GEORGE	510	70	STOICA A. ALEXANDRU-IOAN	407
33	IORDACHE C.L. FLORIN-NICOLAE	407	71	TALPĂU N.STELUȚA	407
34	IORDACHE S. ȘTEFAN	411	72	TEIANU D. MARINA-CAMELIA	411
35	IVAN O. ANDREI-DAVID	407	73	TUDOR M.P. ELENA-GABRIELA	407
36	JITCĂ D. DAVID	407	74	UȚĂ T.I. ȘTEFANA-CRISTINA	407
37	LAZĂR P. ALEXANDRU-ȘTEFAN	411	75	VASILIU A. CONSTANTIN	411
38	LUPAȘCU I. MARIAN	407	76	ZUGRAVU B.G. ANDREI	407

Final exam date

- We have to fix the date for the final exam.
- My proposal is to fix the date for the final exam = to be de date for the deadline of assignment 2 (Sunday, 21st of June).
- **Consider to take into account for the final grade only the grades for the 2 assignments (no final exam) – so each of the 2 assignments will worth 5 points (instead of 3.5 points)**

Recap – intractability of learning

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

Consider $\mathcal{H}_{3DNF}{}^{d}$ = 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 each term $A_i(\mathbf{x})$ is a Boolean conjunction (in $\mathcal{H}_{conj}{}^{d}$) with at most d Boolean literals $x_1, \ldots, x_d$.

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

Let $\mathcal{H}$ be a hypothesis class that is efficient PAC learnable. Then, there exists a randomized 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

The class $\mathcal{H} = \mathcal{H}_{3DNF}^{n}$ is PAC learnable but from the computational perspective, the 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 $\mathcal{H}_{3DNF}^{d}$ is efficient PAC learnable then the problem of graph 3-coloring problem (which is shown to be NP-complete) is in RP.

So, the class $\mathcal{H}_{3DNF}^{n}$ is not efficient PAC learnable under the assumption that NP-complete problems cannot be solved with high probability by a probabilistic polynomial–time algorithm (technically, under the assumption RP $\neq$ NP).

- "Consistent learning is hard" $\not\Rightarrow$ "learning is hard"

- "Consistent learning is hard" $\Rightarrow$ "*proper* learning is hard"
- A proper learning algorithm is a learning algorithm that must output $h \in \mathcal{H}$

Today’s lecture: Overview

- Improper learning
- Boosting

Improper learning

Improper learning of 3-term DNFs

Use the distribution rule to obtain:
$(a \wedge b) \vee (c \wedge d) = (a \vee b) \wedge (a \vee d) \wedge (b \vee c) \wedge (b \vee d)$

Apply for the 3-term DNF formula to obtain a 3-CNF formulae:

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

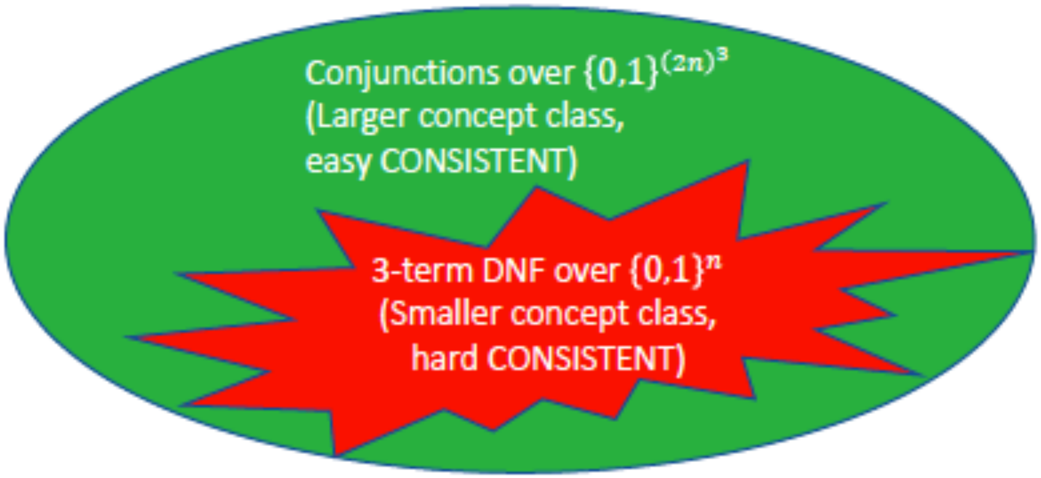
each u,v,w can take 2n values in $\{x_1, \overline{x_1}, x_2, \overline{x_2}, \ldots, x_n, \overline{x_n}\}$

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.

Improper learning



	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	$O\left(n^3 \times \frac{n^3 + log\frac{1}{\delta}}{\varepsilon}\right)$ ↓

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 $\mathcal{H}_{3DNF}{}^{d}$)
- 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.

Use the concepts of one way function (*easy to compute, hard to invert*) + trapdoor one way function *(although is hard to invert, once one has access to its secret key, inverting becomes feasible) to build the class* $\mathcal{H}_F^n = \{ f^{-1} : f$ is a trapdoor function$\}$ with polynomial VC dimension but hard to learn.

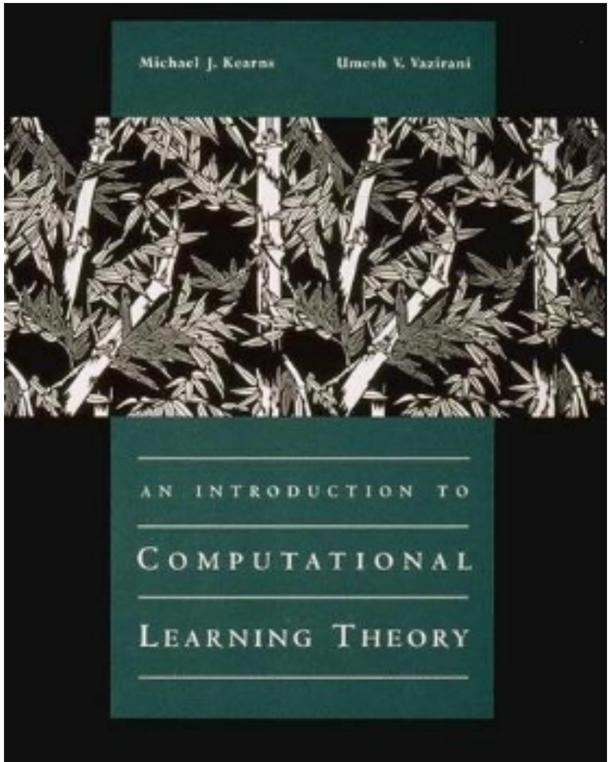
Other examples

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.

Next week: Viola-Jones face detector

ACCEPTED CONFERENCE ON COMPUTER VISION AND PATTERN RECOGNITION 2001

Rapid Object Detection using a Boosted Cascade of Simple Features

Paul Viola
viola@merl.com
Mitsubishi Electric Research Labs
201 Broadway, 8th FL
Cambridge, MA 02139

Michael Jones
mjones@crl.dec.com
Compaq CRL
One Cambridge Center
Cambridge, MA 02142

Abstract

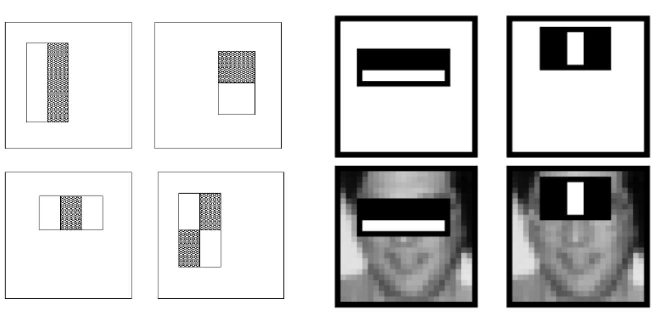
This paper describes a machine learning approach for visual object detection which is capable of processing images

tected at 15 frames per second on a conventional 700 MHz Intel Pentium III. In other face detection systems, auxiliary information, such as image differences in video sequences, or pixel color in color images, have been used to achieve

Viola-Jones face detector

Main idea:

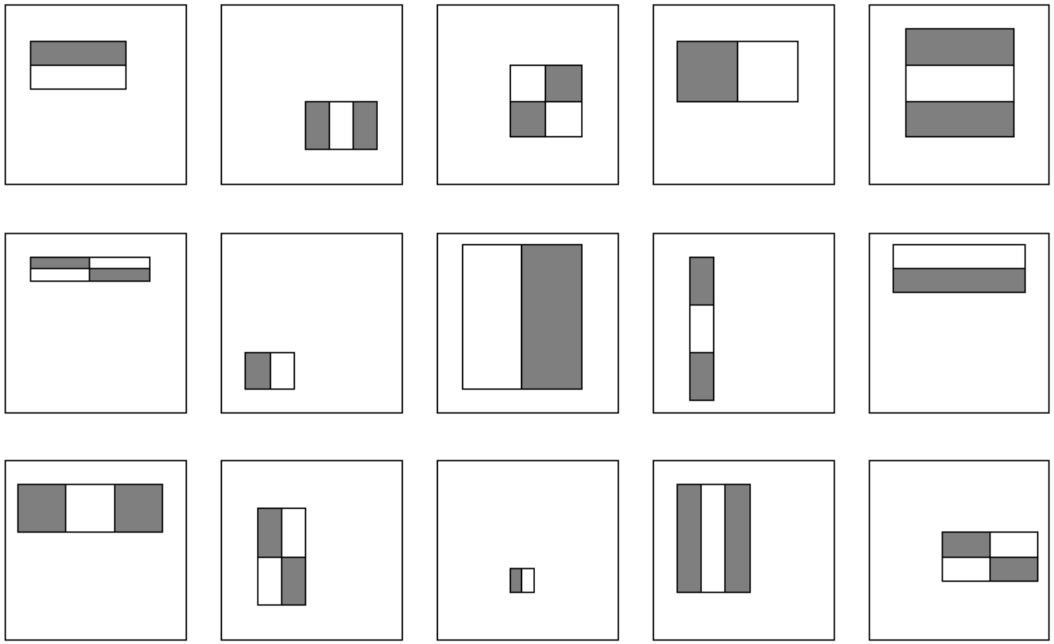
- represent local texture with efficiently computable "rectangular" features within window of interest
- select discriminative features to be weak classifiers
- use boosted combination of them as final classifier
- form a cascade of such classifiers, rejecting clear negatives quickly



"Rectangular" filters

Feature output is difference between adjacent regions

Viola-Jones face detector: features



Considering all possible filter parameters: position, scale, and type:

180,000+ possible features associated with each 24 x 24 window

Which subset of these features should we use to determine if a window has a face?

Use AdaBoost both to select the informative features and to form the classifier

Viola-Jones face detector: features

Figure 3: The first and second features selected by AdaBoost. The two features are shown in the top row and then overlayed on a typical training face in the bottom row. The first feature measures the difference in intensity between the region of the eyes and a region across the upper cheeks. The feature capitalizes on the observation that the eye region is often darker than the cheeks. The second feature compares the intensities in the eye regions to the intensity across the bridge of the nose.

For example an excellent first stage classifier can be constructed from a two-feature strong classifier by reducing the threshold to minimize false negatives. Measured against a validation training set, the threshold can be adjusted to detect 100% of the faces with a false positive rate of 40%. See Figure 3 for a description of the two features used in this classifier.

Computation of the two feature classifier amounts to about 60 microprocessor instructions. It seems hard to imagine that any simpler filter could achieve higher rejection rates. By comparison, scanning a simple image template, or a single layer perceptron, would require at least 20 times as many operations per sub-window.

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), $L_{\mathrm{D},f}(h) \leq \varepsilon$.

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

$\Longrightarrow$ 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

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

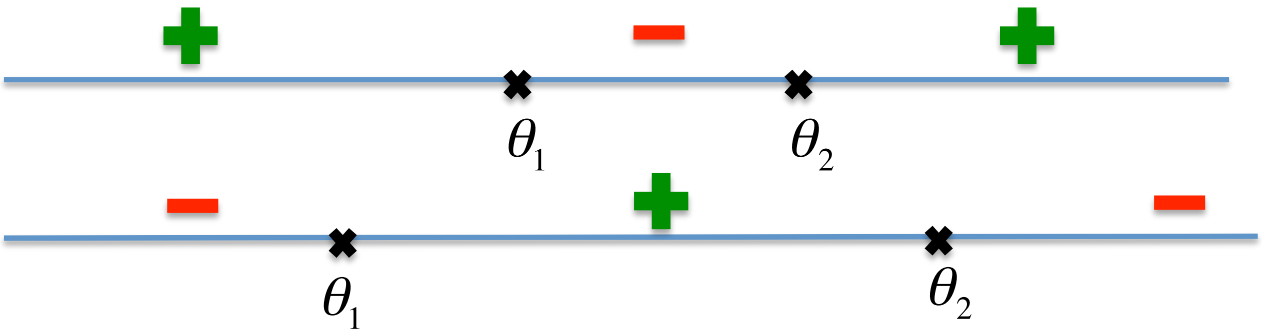
Applying this with $\varepsilon = 1/2 - \gamma$ we immediately obtain that if $d = \infty$ then $\mathcal{H}$ is not γ -weak-learnable. This implies that ***from the statistical perspective*** (i.e., if we ignore computational complexity), ***weak learnability*** is also characterized by the VC dimension 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 $\mathcal{X} = \mathrm{R}$, $\mathcal{H}$ is the class of 3-piece classifiers (signed intervals):

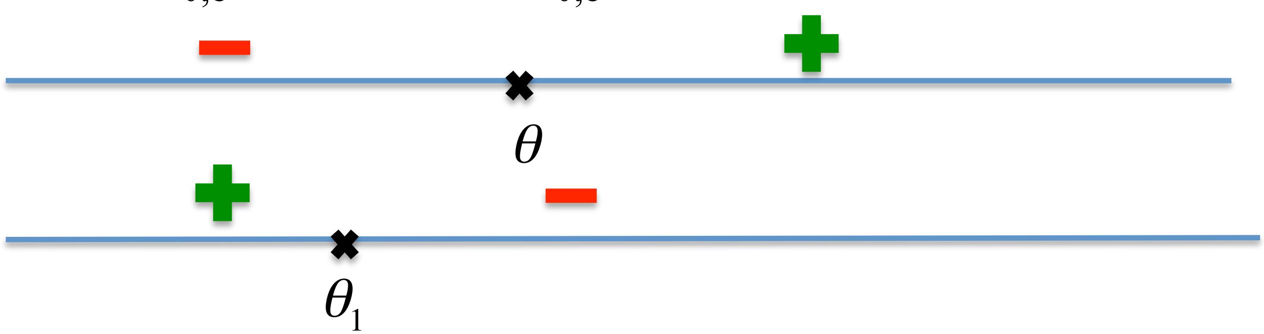
$$H = \{h_{\theta_1,\theta_2,b} \,|\, \theta_1,\theta_2 \in R, \theta_1 < \theta_2, b \in \{-1,+1\}\} \quad h_{\theta_1,\theta_2,b}(x) = \begin{cases} +\mathrm{b}, \text{ if } \mathrm{x} < \theta_1 \text{ or } \mathrm{x} > \theta_2 \\ -\mathrm{b}, \text{ if } \theta_1 \le \mathrm{x} \le \theta_2 \end{cases}$$



b = 1

b = -1

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



b = 1

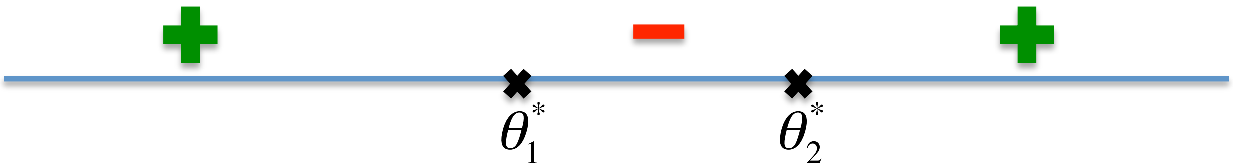
b = -1

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

Weak learnability - example

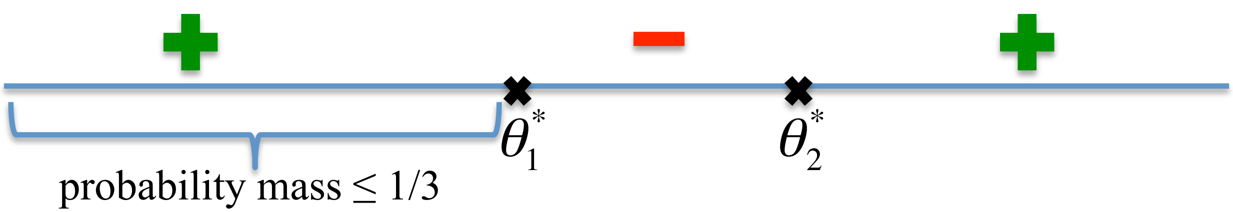
$ERM_{\mathcal{B}}$ is a γ-weak learner for $\mathcal{H}$, for $\gamma < 1/6$.

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

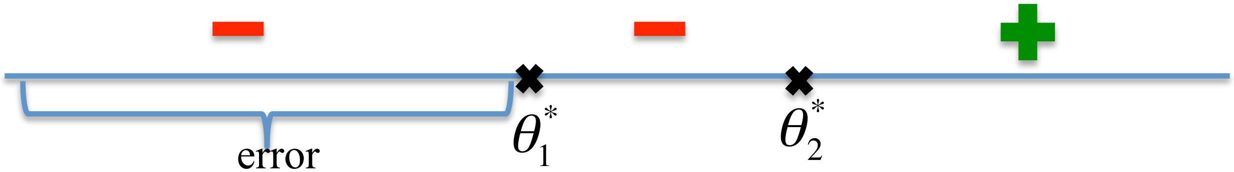


Consider a distribution $\mathcal{D}$ over $\mathcal{X} = \mathbf{R}$. Then, we are sure that at least one of the regions $(-\infty, \theta_1^*)$, $[\theta_1^*, \theta_2^*]$, $(\theta_2^*, +\infty)$ has a probability mass wrt $\mathcal{D} \leq 1/3$.

Consider, without loss of generality that $\mathcal{D}((-\infty, \theta_1^*)) = P_{x \sim \mathcal{D}}(x \in (-\infty, \theta_1^*)) \leq 1/3$
Then the hypothesis $h_{\theta, b} \in$ 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

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

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:

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

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

$$L_{\mathcal{D}}(\mathrm{ERM}_{\mathrm{B}}(S)) \le \min_{h\in\mathcal{H}} L_{\mathcal{D}}(h) + \epsilon = 1/3 + \varepsilon$$

Take ε such that $1/3 + \varepsilon < 1/2$, $\varepsilon < 1/6$. Than $\mathrm{ERM}_{\mathcal{B}}$ 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 $\mathcal{H}_{DS}{}^d$?

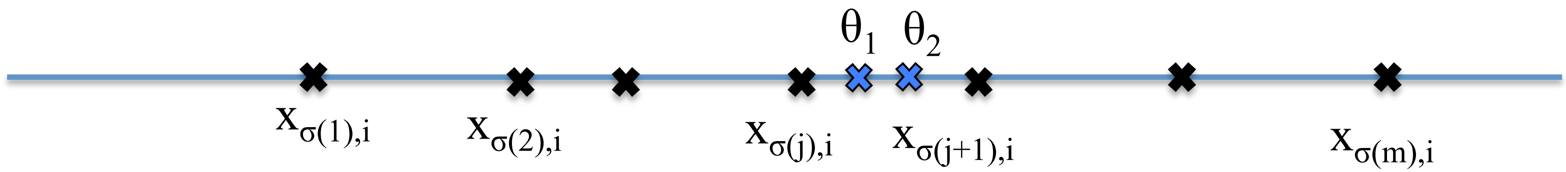
Let $S = ((\mathbf{x_1}, y_1), \ldots, (\mathbf{x_m}, y_m))$ be a training set of size *m*. We want to find the best h_{i^*,θ^*,b^*} which minimizes the training error on S:

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

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.

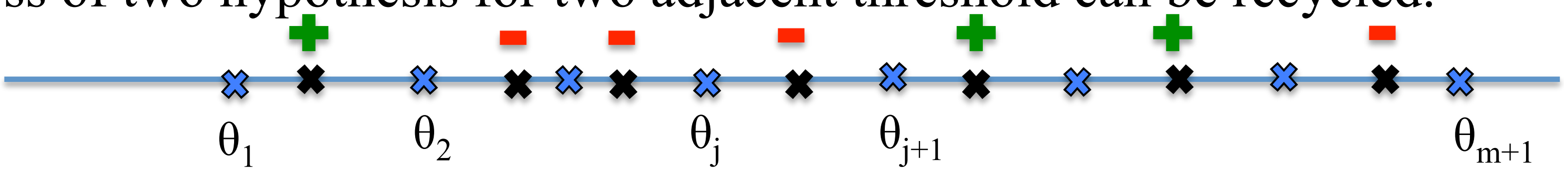
In fact, we can restrict the search over all $\theta \in \mathbf{R}$ just to $m + 1$ values of θ, chosen in the intervals $(-\infty, x_{\sigma(1),i})$, $[x_{\sigma(1),i}, x_{\sigma(2),i})$, …, $[x_{\sigma(m),i}, +\infty)$ by taking a representative threshold in each interval. For example we can take 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

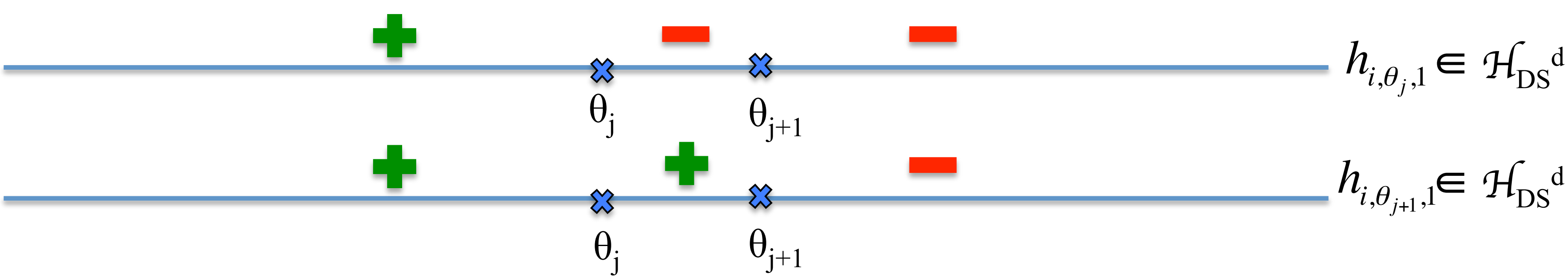
$\mathcal{H}_{DS}{}^{d} = \{h_{i,\theta,b}: \mathbf{R}^d \rightarrow \{-1,1\},\ h_{i,\theta,b}(\mathbf{x}) = \text{sign}(\theta - x_i) \times b,\ 1 \leq i \leq d,\ \theta \in \mathbf{R},\ b \in \{-1,+1\}\}$

We have $1 \leq i \leq d$, $\theta \in \Theta_i$, $|\Theta_i| = m+1$, $b \in \{-1,+1\}$. 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

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

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 $x_{\sigma(1),i} \le x_{\sigma(2),i} \le \ldots \le x_{\sigma(m),i}$
    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: $D^{(t+1)}(i) = \dfrac{D^{(t)}(i) \times e^{-w_t h_t(x_i) y_i}}{Z_{t+1}}$

where Z_{t+1} normalization factor ($\mathbf{D}^{(t+1)}$ is a distribution): $Z_{t+1} = \sum_{i=1}^{n} D^{(t)}(i) \times e^{-w_t h_t(x_i) y_i}$

w_t is a weight: $w_t = \dfrac{1}{2} \ln(\dfrac{1}{\varepsilon_t} - 1) > 0$ as the error $\varepsilon_t < 0.5$

ε_t is the error of h_t on $\mathbf{D}^{(t)}$: $\varepsilon_t = \Pr_{i \sim D^{(t)}}[h_t(x_i) \neq y_i] = \sum_{i=1}^{m} D^{(t)}(i) \times 1_{[h_t(x_i) \neq y_i]}$

If example $\mathbf{x}_i$ is correctly classified then $h_t(\mathbf{x}_i) = y_i$ so at the next iteration t+1 its importance (probability distribution) will be decreased to:

$$D^{(t+1)}(i) = \frac{D^{(t)}(i) \times e^{-w_t}}{Z_{t+1}} = \frac{D^{(t)}(i) \times e^{-\frac{1}{2}\ln(\frac{1}{\varepsilon_t} - 1)}}{Z_{t+1}} = \frac{D^{(t)}(i) \times (\frac{1}{\varepsilon_t} - 1)^{-\frac{1}{2}}}{Z_{t+1}} = \frac{D^{(t)}(i) \times \sqrt{\frac{\varepsilon_t}{1 - \varepsilon_t}}}{Z_{t+1}}$$

AdaBoost

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

where Z_{t+1} normalization factor ($\mathbf{D}^{(t+1)}$ is a distribution): $Z_{t+1} = \sum_{i=1}^{n} D^{(t)}(i) \times e^{-w_t h_t(x_i) y_i}$

w_t is a weight: $w_t = \frac{1}{2}\ln(\frac{1}{\varepsilon_t} - 1) > 0$ as the error $\varepsilon_t < 0.5$

ε_t is the error of h_t on $\mathbf{D}^{(t)}$: $\varepsilon_t = \Pr_{i \sim D^{(t)}}[h_t(x_i) \neq y_i] = \sum_{i=1}^{m} D^{(t)}(i) \times 1_{[h_t(x_i) \neq y_i]}$

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:

$$D^{(t+1)}(i) = \frac{D^{(t)}(i) \times e^{w_t}}{Z_{t+1}} = \frac{D^{(t)}(i) \times e^{\frac{1}{2}\ln(\frac{1}{\varepsilon_t} - 1)}}{Z_{t+1}} = \frac{D^{(t)}(i) \times (\frac{1}{\varepsilon_t} - 1)^{\frac{1}{2}}}{Z_{t+1}} = \frac{D^{(t)}(i) \times \sqrt{\frac{1-\varepsilon_t}{\varepsilon_t}}}{Z_{t+1}}$$

AdaBoost

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

where Z_{t+1} normalization factor ($\mathbf{D}^{(t+1)}$ is a distribution): $Z_{t+1} = \sum_{i=1}^{n} D^{(t)}(i) \times e^{-w_t h_t(x_i) y_i}$

w_t is a weight: $w_t = \frac{1}{2} \ln(\frac{1}{\varepsilon_t} - 1) > 0$ as the error $\varepsilon_t < 0.5$

ε_t is the error of h_t on $\mathbf{D}^{(t)}$: $\varepsilon_t = \Pr_{i \sim D^{(t)}}[h_t(x_i) \neq y_i] = \sum_{i=1}^{m} D^{(t)}(i) \times 1_{[h_t(x_i) \neq y_i]}$

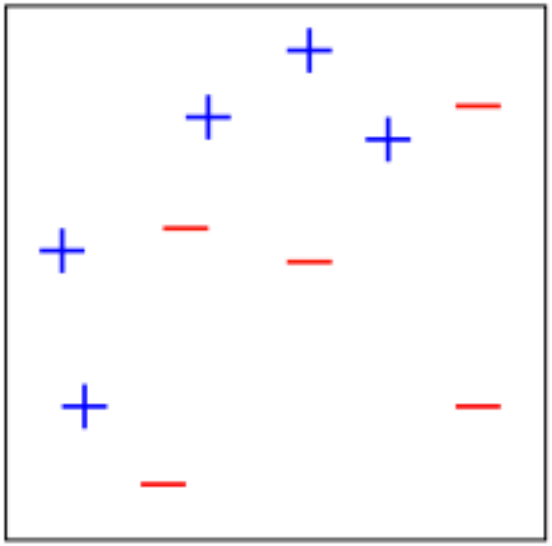
If example $\mathbf{x}_i$ is correctly classified then $h_t(\mathbf{x}_i) = y_i$ so at the next iteration t+1 its importance (probability distribution) will be decreased to $D^{(t+1)}(i) = \dfrac{D^{(t)}(i) \times e^{-w_t}}{Z_{t+1}}$

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 $D^{(t+1)}(i) = \dfrac{D^{(t)}(i) \times e^{w_t}}{Z_{t+1}}$

- output final/combined classifier h_{final}: $h_{final}(x) = sign(\sum_{t=1}^{T} w_t h_t(x))$

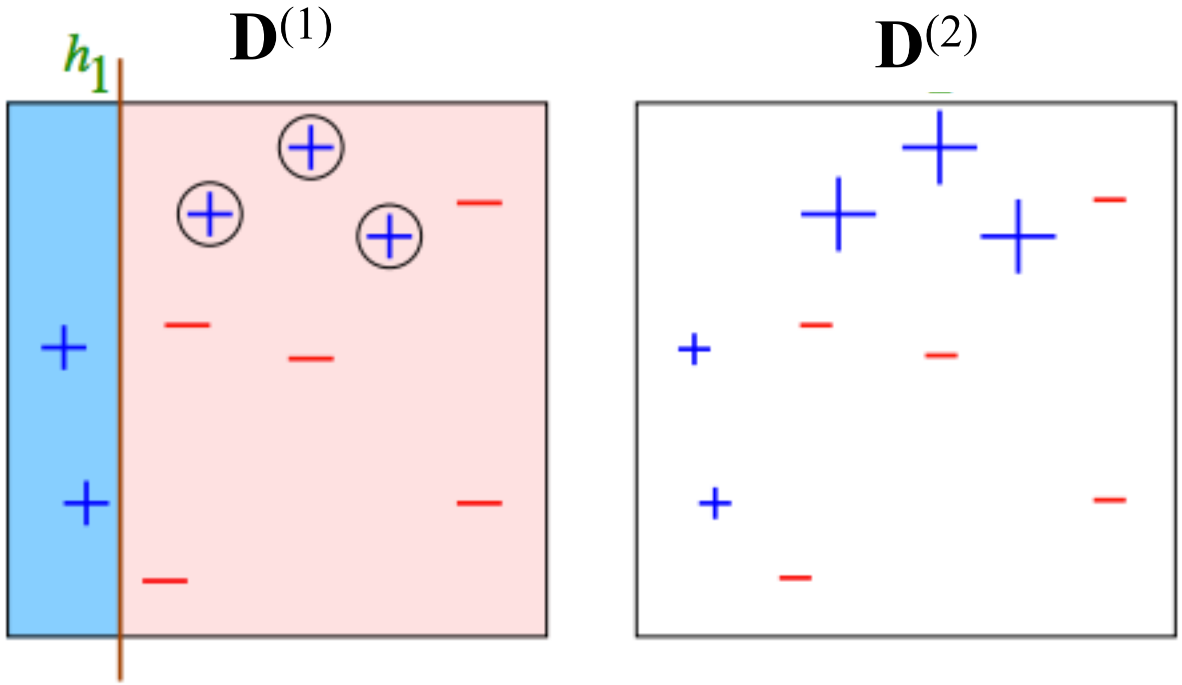
Toy example

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



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

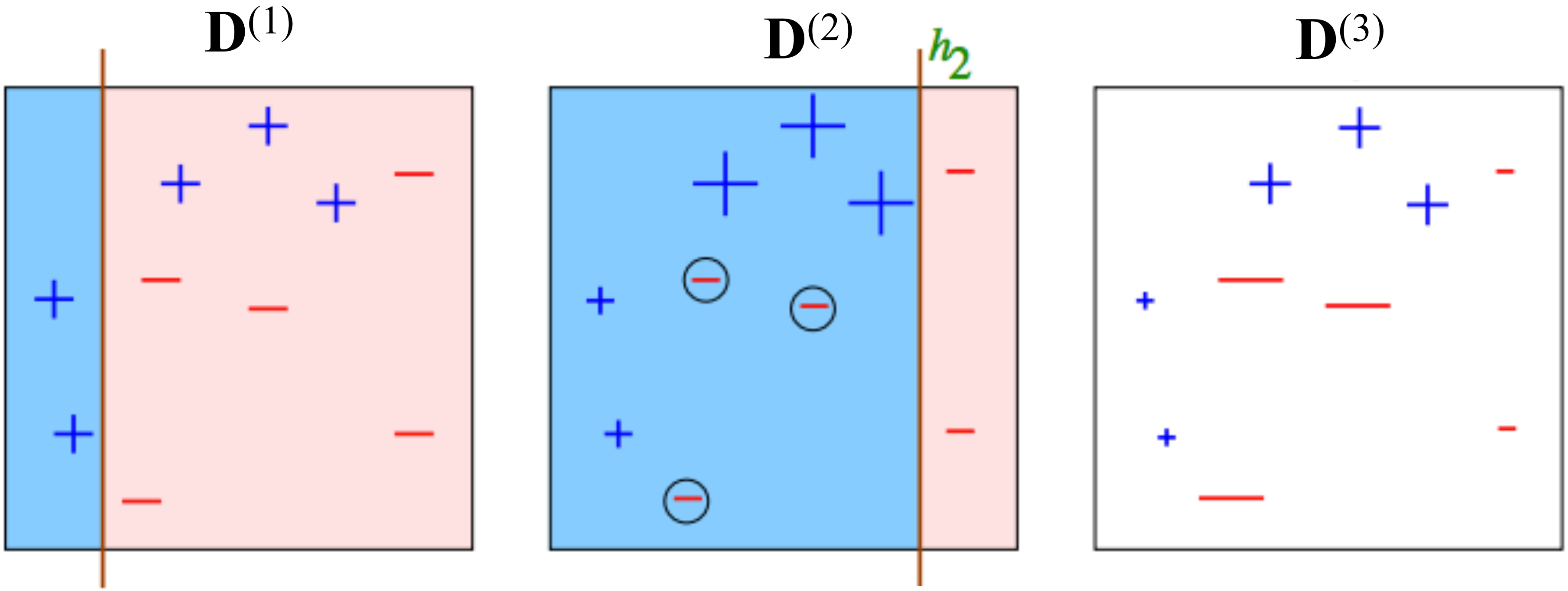
Toy example – Round 1



$\varepsilon_1 = 0.30$
$w_1 = 0.42$

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

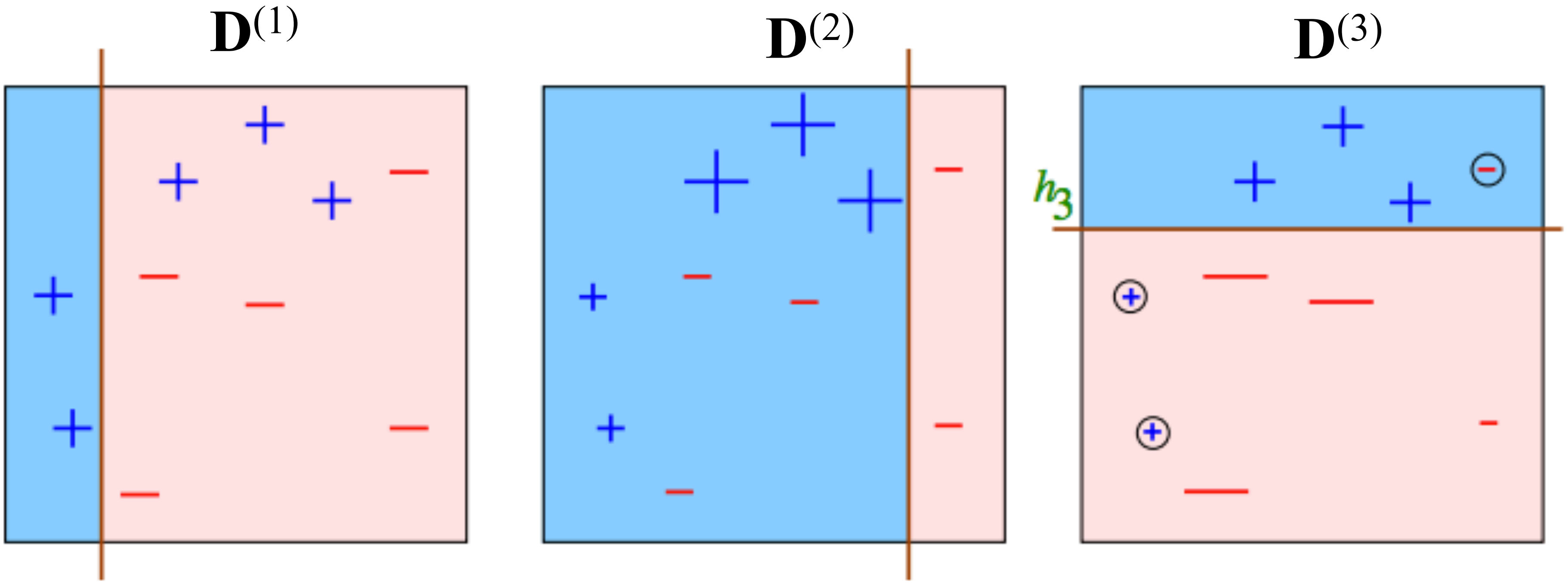
Toy example – Round 2



$\varepsilon_2 = 0.21$
$w_2 = 0.65$

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