Advanced Machine Learning

Bogdan Alexe,

bogdan.alexe@fmi.unibuc.ro

University of Bucharest, 2nd semester, 2020-2021

Recap: The fundamental theorem of statistical learning

Theorem (The Fundamental Theorem of Statistical Learning).
Let $\mathcal{H}$ be a hypothesis class of functions from a domain $\mathcal{X}$ to $\{0,1\}$ and let the loss function be the $0-1$ loss. Then, the following are equivalent:

1. $\mathcal{H}$ has the uniform convergence property.
2. Any ERM rule is a successful agnostic PAC learner for $\mathcal{H}$.
3. $\mathcal{H}$ is agnostic PAC learnable.
4. $\mathcal{H}$ is PAC learnable.
5. Any ERM rule is a successful PAC learner for $\mathcal{H}$.
6. $\mathcal{H}$ has a finite VC-dimension.

A finite VC- dimension guarantees learnability*. *Hence, the VC-dimension characterizes PAC learnability.

The Growth function

Definition

Let $\mathcal{H}$ be a hypothesis class. Then the growth function of $\mathcal{H}$, denoted by τ_H, where $\tau_{\mathcal{H}}$: **N**→**N**, is defined as:

$$\tau_H(m) = \max_{C \subseteq X : |C| = m} |H_C|$$

In other words, $\tau_H(m)$ is the maximum number of different functions from a set C of size m to {0,1} that can be obtained by restricting $\mathcal{H}$ to C.

Observation: if VCdim($\mathcal{H}$) = d then for any m ≤ d we have $\tau_{\mathcal{H}}(m) = 2^m$. In such cases, $\mathcal{H}$ induces all possible functions from C to {0,1}.

What happens when m becomes larger than the VC-dimension?

Answer given by the Sauer's lemma: the growth function $\tau_{\mathcal{H}}$ increases polynomially rather than exponentially with m.

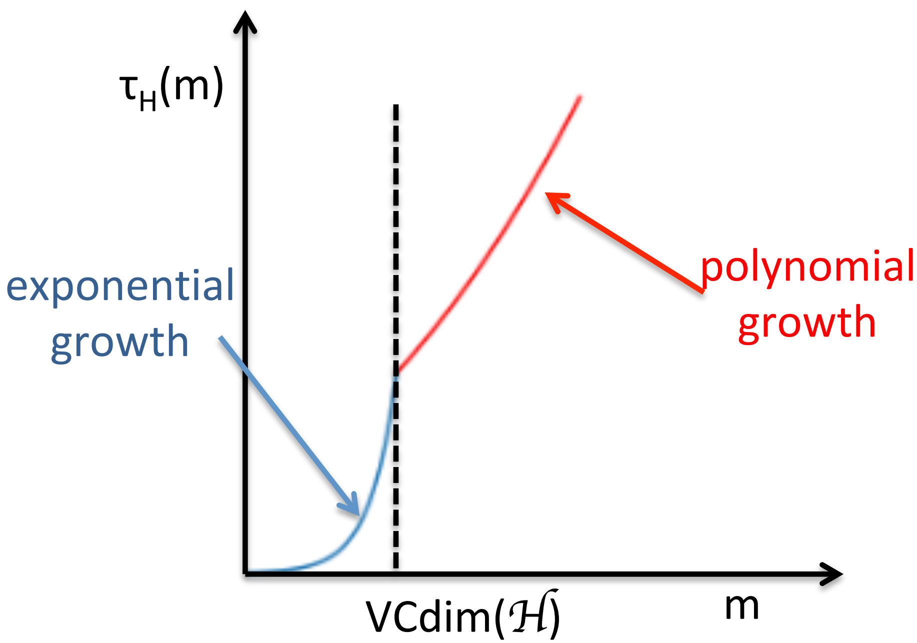
The Sauer's lemma

Lemma (Sauer – Shelah – Perles)

Let $\mathcal{H}$ be a hypothesis class with VCdim($\mathcal{H}$) $\leq$ d < ∞ . Then, for all m , we have that:

$$\tau_H(m) \leq \sum_{i=0}^{d} C_m^i$$

In particular, if m > d +1 then $\tau_{\mathcal{H}}(m) \leq (em/d)^d = O(m^d)$



Proof for 6 → 1

We want to prove that finite VC-dimension → *uniform convergence property*

Two steps:

1. (Sauer's lemma) If VCdim($\mathcal{H}$) $\leq d < \infty$, then even though $\mathcal{H}$ might be infinite, when restricting it to a finite set $C \subseteq \mathcal{X}$, its "effective" size, $|\mathcal{H}_C|$, is only $O(|C|^d)$. That is, the size of $\mathcal{H}_C$ grows polynomially rather than exponentially with $|C|$.

2. we have shown in lecture 4 that finite hypothesis classes enjoy the uniform convergence property. We generalize this result and show that uniform convergence holds whenever the hypothesis class has a "small effective size." By "small effective size" we mean classes for which $|\mathcal{H}_C|$ grows polynomially with $|C|$.

The fundamental theorem of statistical learning – quantitative version

Theorem

Let $\mathcal{H}$ be a hypothesis class of functions from a domain $\mathcal{X}$ to $\{0,1\}$ and let the loss function be the 0−1 loss. Assume that VCdim($\mathcal{H}$) = d <∞. Then, there are absolute constants C_1, C_2 such that:

1. $\mathcal{H}$ has the uniform convergence property with sample complexity:

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

2. $\mathcal{H}$ is agnostic PAC learnable with sample complexity:

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

3. $\mathcal{H}$ is PAC learnable with sample complexity:

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

The VC dimension determines (along with ε, δ) the samples complexities of learning a class. It gives us a lower and an upper bound.

Computational complexity of learning

Computational resources of learning

For learning we need 2 type of resources:

1. *Information = training data*
   - so far we analyzed how much training data (sample size) we need in order to learn
   - *sample complexity*

2. *Computation = runtime*
   - for how much time an algorithm (that implements learning) will run, once we have sufficiently many training examples
   - *computational complexity*
   - crucial when we need fast ML applications (driver surveillance, stock exchange trading, etc)
   - runtime = number of elementary instructions executed - arithmetic operations over real numbers - in an asymptotic sense (with respect to input size) of the algorithm, e.g. *$O(n)$ – where n is the size of the input size*

Input size parameter of learning

What should play the role of the input size parameter in learning?

- size of the training set that the algorithm receives?
  - for a very large number of examples, much larger than the sample complexity of learning, the algorithm can ignore the extra samples
  - a larger training set does not make the problem more difficult

- size of the hypothesis class?
  - might be infinite: $|\mathcal{H}_{\text{thresholds}}| = \infty$

- accuracy ε, confidence δ and another parameter n related to the size/complexity of $\mathcal{X}$, $\mathcal{H}$
  - how much computation we need in order to get accuracy ε with confidence δ
  - want to have *efficient learning* (give a formal definition later): polynomial in 1/ε, 1/δ and n (some parameter related to the size/complexity of domain/hypothesis class: more complex hypothesis needs more computation time)

Input size parameter of learning

What should play the role of the input size parameter in learning?

- accuracy ε, confidence δ and another parameter n related to the size/ complexity of $\mathcal{X}$, $\mathcal{H}$
  - how much computation we need in order to get accuracy ε with confidence δ
  - want to have *efficient learning* (give a formal definition later): polynomial in $1/\varepsilon$, $1/\delta$ and n (some parameter related to the size/complexity of domain/hypothesis class: more complex hypothesis needs more computation time)

- parameter n can be the embedding dimension
  - if we decide to use n features to describe objects, how will that increase runtime?

- we study the runtime in an asymptotic sense by defining a sequence of pairs $(\mathcal{X}_n, \mathcal{H}_n)_{n=1,2,\ldots}$ and studying asymptotic complexity of learning $\mathcal{X}_n$, $\mathcal{H}_n$ as n grows to ∞

Prevent "cheating"

The output of the learning algorithm L is a hypothesis h from $\mathcal{H}$.

- a learning algorithm L can "cheat" by transferring the computational burden to the output hypothesis
  - define the output hypothesis to be the function that stores the training set in memory and computes the ERM hypothesis on the training set and applies it to a test example x
- the runtime of a learning algorithm A defined as the maximum of:
  - the time it takes A to output some h
  - the time it takes h to output a label on any given x from $\mathcal{X}$

Example 1: Conjunctions of Boolean literals

- $\mathcal{H}_{conj}{}^{d}$ = class of conjunctions of at most d Boolean literals $x_1, \ldots, x_d$
  - a Boolean literal is either x_i or its negation $\overline{x_i}$ (or 1 = missing literal)
  - can interpret x_i as feature i
  - example: $h = x_1 \wedge \overline{x_2} \wedge x_4$ where $\overline{x_2}$ denotes the negation of the Boolean literal x_2
  - $\mathcal{X} = \{0,1\}^d$
- consider the realizable case
  - there is a conjunction h^* in $\mathcal{H}_{conj}{}^{d}$ that labels the examples

- $|\mathcal{H}_{conj}{}^{d}| = 3^d + 1 < \infty$ so it has finite VC dimension (less than $\log_2(3^d+1)$), so it's PAC learnable. In seminar class 3 we shown that VCdim($\mathcal{H}_{conj}{}^{d}$) = d so the sample complexity $m_{\mathcal{H}}(\varepsilon,\delta)$ is bounded by:

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

- So, $m_{\mathcal{H}}(\varepsilon,\delta)$ is polynomial in $1/\varepsilon$, $1/\delta$, d (measures the complexity of the hypothesis class $\mathcal{H}_{conj}{}^{d}$)

Example 1: Conjunctions of Boolean literals

- $\mathcal{H}_{\text{conj}}{}^{d}$ = class of conjunctions of at most d Boolean literals $x_1, \ldots, x_d$
- a simple algorithm for finding an ERM hypothesis is based on positive examples and consists of the following:
  - for each positive example $(b_1, \ldots b_d)$,
    - if $b_i = 1$ then $\overline{x_i}$ is ruled out as a possible literal in the concept class
    - if $b_i = 0$ then x_i is ruled out.
  - the conjunction of all the literals not ruled out is thus a hypothesis consistent with the target

0	1	1	0	1	1	+
0	1	1	1	1	1	+
0	0	1	1	0	1	-
0	1	1	1	1	1	+
1	0	0	1	1	0	-
0	1	0	0	1	1	+

$x_1 \quad \overline{x}_1 \quad x_2 \quad \overline{x}_2 \quad x_3 \quad \overline{x}_3 \quad x_4 \quad \overline{x}_4 \quad x_5 \quad \overline{x}_5 \quad x_6 \quad \overline{x}_6$

$$\rightarrow \overline{x}_1 \wedge x_2 \wedge x_5 \wedge x_6$$

Example 1: Conjunctions of Boolean literals

- $\mathcal{H}_{conj}{}^{d}$ = class of conjunctions of at most d Boolean literals $x_1, \ldots, x_d$
- a simple algorithm for finding an ERM hypothesis is based on positive examples and consists of the following:
  - for each positive example $(b_1, \ldots b_n)$,
    - if $b_i = 1$ then $\overline{x_i}$ is ruled out as a possible literal in the concept class
    - if $b_i = 0$ then x_i is ruled out.
  - the conjunction of all the literals not ruled out is thus a hypothesis consistent with the target

- runtime of the algorithm is $O(m_{\mathcal{H}}(\varepsilon,\delta)*\mathrm{d})$, so is polynomial in $1/\varepsilon$, $1/\delta$, d

- in the agnostic (unrealizable) case: unless P = NP, there is no algorithm whose running time is polynomial in $m_{\mathcal{H}}(\varepsilon,\delta)$ and d that is guaranteed to find an ERM hypothesis for the class of Boolean conjunctions.

Example 2: Learning axis aligned rectangles in $\mathbf{R}^d$

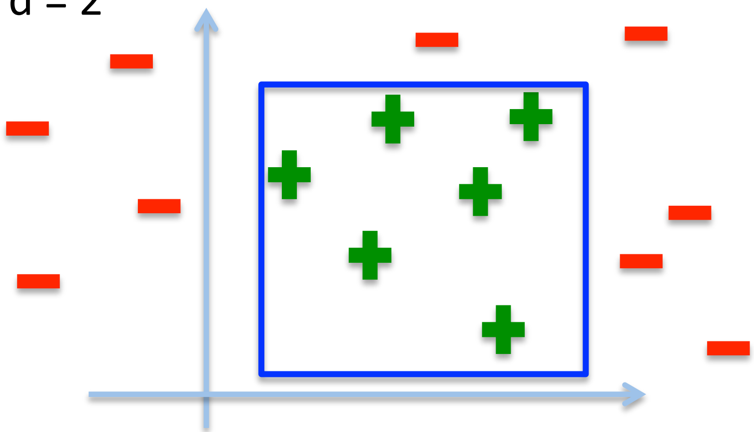
- $\mathcal{H}_{rec}^{d}$ = the class of axis aligned rectangles in $\mathbf{R}^d$

$$H_{rec}^{d} = \{h_{a_1,b_1,a_2,b_2,\ldots,a_d,b_d} : R^d \to \{0,1\} \mid a_1 \le b_1, a_2 \le b_2, \ldots, a_d \le b_d, a_i \in R, b_i \in R\}$$

$$h_{a_1,b_1,a_2,b_2,\ldots,a_d,b_d}(x_1,x_2\ldots,x_d) = \begin{cases} 1, if\, a_1 \le x_1 \le b_1, a_2 \le x_2 \le b_2, \ldots, a_d \le x_d \le b_d \\ 0, otherwise \end{cases}$$

- consider the realizable case:
  - there exists a rectangle h* in $\mathcal{H}_{rec}^{d}$ with real risk = 0

d = 2



We have shown in the seminar class that:

- the algorithm that returns the rectangle enclosing all positive examples is ERM
- $\mathcal{H}_{rec}^{d}$ is PAC learnable with sample size

$$m_{H_{rec}^{d}}(\varepsilon,\delta) \le \left\lceil \frac{2d\log(\frac{2d}{\delta})}{\varepsilon} \right\rceil$$

- the runtime is $O(m_H d)$ as for each dimension, the algorithm has to find the minimal and the maximal values among the positive instances in the training sequence. So it is polynomial in $1/\varepsilon$, $1/\delta$, d.

Example 2: Learning axis aligned rectangles in $\mathbf{R}^d$

- $\mathcal{H}_{rec}{}^{d}$ = the class of axis aligned rectangles in $\mathbf{R}^d$

$$H_{rec}^{d} = \{h_{a_1,b_1,a_2,b_2,\dots,a_d,b_d} : R^d \to \{0,1\} \mid a_1 \le b_1, a_2 \le b_2, \dots, a_d \le b_d, a_i \in R, b_i \in R\}$$

$$h_{a_1,b_1,a_2,b_2,\dots,a_d,b_d}(x_1, x_2 \dots, x_d) = \begin{cases} 1, if\, a_1 \le x_1 \le b_1, a_2 \le x_2 \le b_2, \dots, a_d \le x_d \le b_d \\ 0, otherwise \end{cases}$$

- consider the agnostic case:
  - distribution $\mathcal{D}$ over $\mathcal{Z} = \mathbf{R}^d \times \{0,1\}$ (a sample could get both labels)
  - if there exist a labeling function f this might not be in $\mathcal{H}_{rec}{}^{d}$

- VCdim($\mathcal{H}_{rec}{}^{d}$) = 2d (see seminar class), so we have that:

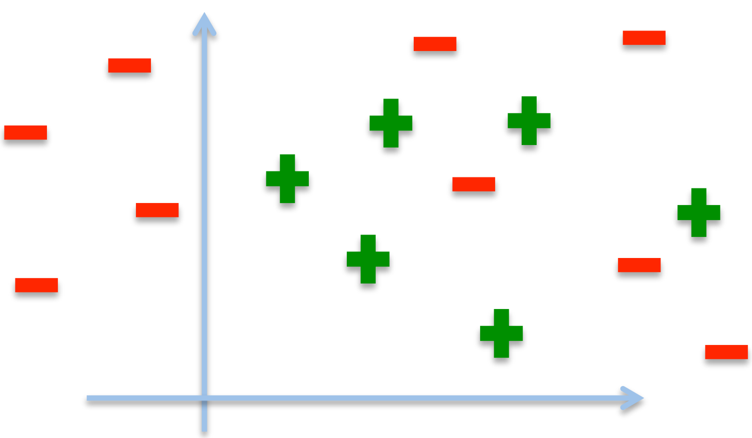
$$C_1 \frac{2d + \log(\frac{1}{\delta})}{\varepsilon^2} \le m_{H_{rec}^{d}}(\varepsilon, \delta) \le C_2 \frac{2d + \log(\frac{1}{\delta})}{\varepsilon^2}$$

- $m_{H_{rec}^{d}}(\varepsilon, \delta)$ is polynomial in 1/ε, 1/δ , d (measures the complexity of the $\mathcal{H}_{rec}{}^{d}$)

Example 2: Learning axis aligned rectangles in $\mathbf{R}^d$

- $\mathcal{H}_{rec}{}^{d}$ = the class of axis aligned rectangles in $\mathbf{R}^d$
- consider that we have a sample S of size: $m_{H_{rec}^d}(\varepsilon,\delta) \approx C \dfrac{2d + \log(\frac{1}{\delta})}{\varepsilon^2}$
- what is the runtime of the ERM algorithm?
  - how long it will take to find the best rectangle in R^d?
  - go over all axis aligned rectangles in R^d and choose the best one (based on minimizing the error on the training data)

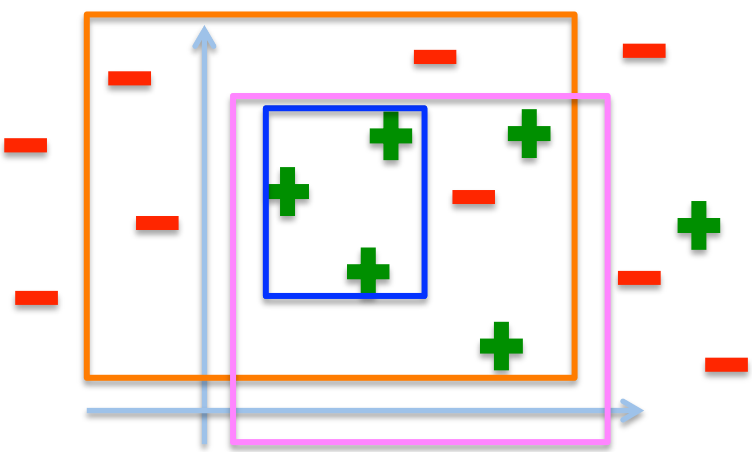
d = 2



Example 2: Learning axis aligned rectangles in $\mathbf{R}^d$

- $\mathcal{H}_{rec}{}^{d}$ = the class of axis aligned rectangles in $\mathbf{R}^d$
- consider that we have a sample S of size: $m_{H_{rec}^d}(\varepsilon,\delta) \approx C\dfrac{2d+\log(\frac{1}{\delta})}{\varepsilon^2}$
- what is the runtime of the ERM algorithm?
  - how long it will take to find the best rectangle in R^d?
  - go over all axis aligned rectangles in R^d and choose the best one (based on minimizing the error on the training data)

d = 2



error: (1 + 4)/ (6 + 9) = 5/15

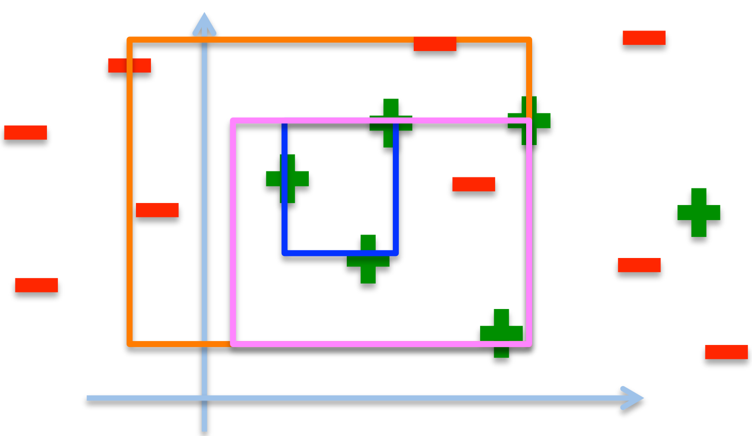
error: (3 + 0)/ (6 + 9) = 3/15

error: (1 + 1)/ (6 + 9) = 2/15

Example 2: Learning axis aligned rectangles in $\mathbf{R}^d$

- $\mathcal{H}_{rec}{}^d$ = the class of axis aligned rectangles in $\mathbf{R}^d$
- consider that we have a sample S of size: $m_{H^d_{rec}}(\varepsilon,\delta) \approx C\dfrac{2d + \log(\frac{1}{\delta})}{\varepsilon^2}$
- what is the runtime of the ERM algorithm?
  - how long it will take to find the best rectangle in R^d?
  - go over all axis aligned rectangles in R^d and choose the best one (based on minimizing the error on the training data)

d = 2



error: (1 + 4)/ (6 + 9) = 5/15

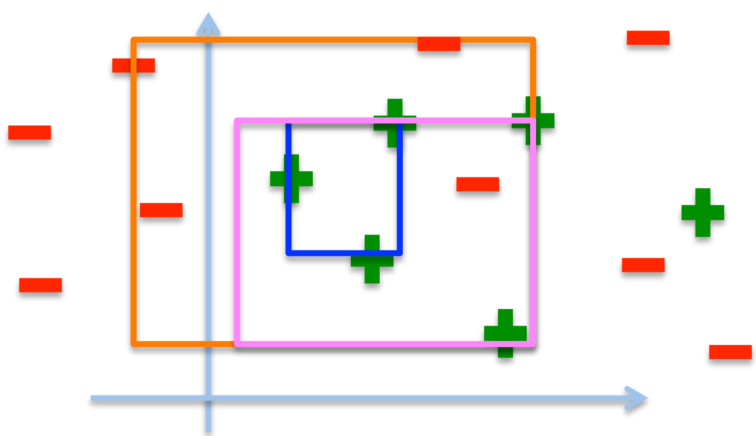
error: (3 + 0)/ (6 + 9) = 3/15

error: (1 + 1)/ (6 + 9) = 2/15

- the number of all possible rectangles can be reduces to all possible rectangles that have points of S on every boundary edge (very efficient algorithm)

Example 2: Learning axis aligned rectangles in $\mathbf{R}^d$

- $\mathcal{H}_{rec}^{d}$ = the class of axis aligned rectangles in $\mathbf{R}^d$
- consider that we have a sample S of size: $m_{H_{rec}^d}(\varepsilon,\delta) \approx C \dfrac{2d + \log(\frac{1}{\delta})}{\varepsilon^2}$
- what is the runtime of the ERM algorithm?
  - how long it will take to find the best rectangle in R^d?
  - Step 1: generate all the rectangles based on the sample points in R^d
  - Step 2: for each such rectangle compute the training error
  - Step 3: choose the rectangle with the smallest training error



error: (1 + 4)/ (6 + 9) = 5/15

error: (3 + 0)/ (6 + 9) = 3/15

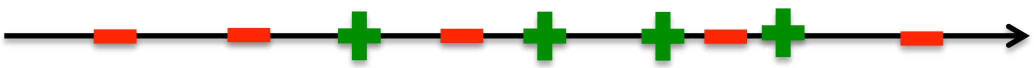
error: (1 + 1)/ (6 + 9) = 2/15

- how many possible rectangles can we construct based on the points in the sample S?

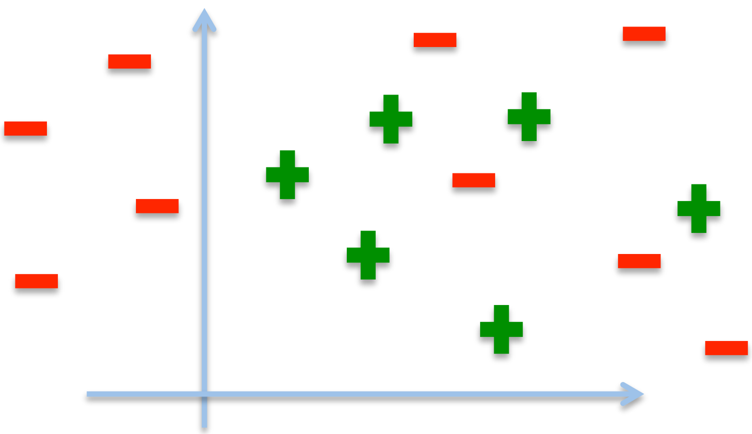
Example 2: Learning axis aligned rectangles in $\mathbf{R}^d$

- $\mathcal{H}_{rec}{}^{d}$ = the class of axis aligned rectangles in $\mathbf{R}^d$
- how many possible rectangles can we construct based on the points in the sample S?

d = 1

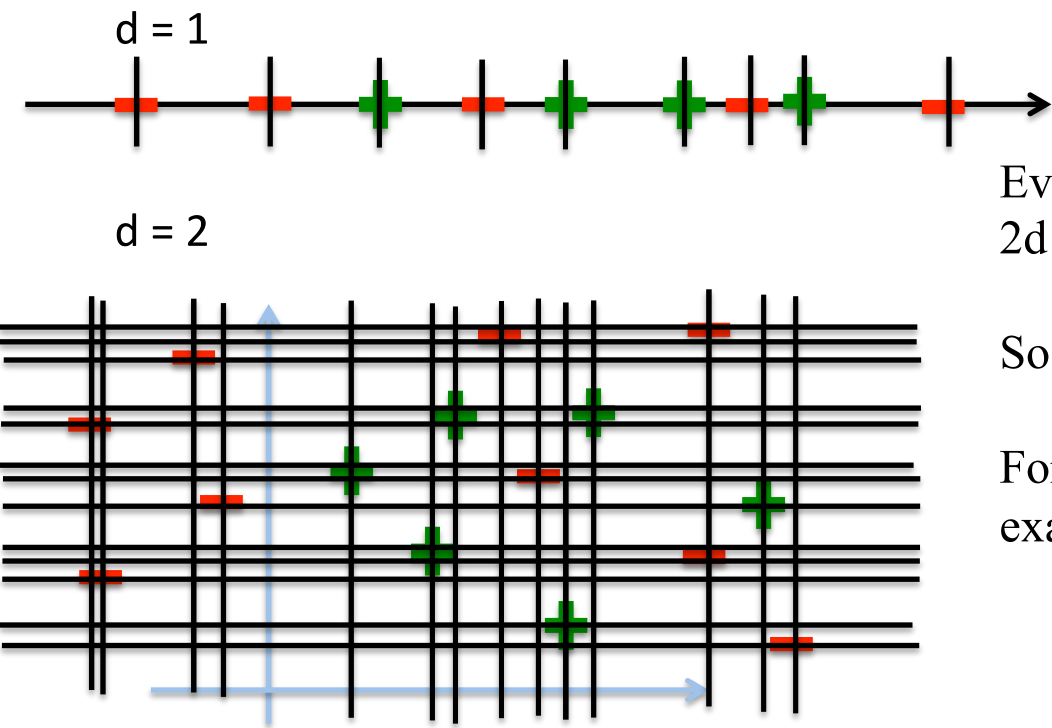


d = 2



Example 2: Learning axis aligned rectangles in $\mathbf{R}^d$

- $\mathcal{H}_{rec}{}^{d}$ = the class of axis aligned rectangles in $\mathbf{R}^d$
- how many possible rectangles can we construct based on the points in the sample S?



Every such rectangle is determined by at most 2d points from S

So there are at most $|S|^{2d}$ such rectangles.

For each rectangle we need to iterate over all examples to compute the training error.

So, the runtime is: $O\left(\left[C\frac{2d+\log(\frac{1}{\delta})}{\varepsilon^2}\right]^{2d+1}\right)$

Example 2: Learning axis aligned rectangles in $\mathbf{R}^d$

- $\mathcal{H}_{rec}^{d}$ = the class of axis aligned rectangles in $\mathbf{R}^d$

- the runtime of the ERM_H is: $O\left(\left[C\dfrac{2d+\log(\frac{1}{\delta})}{\varepsilon^2}\right]^{2d+1}\right)$

- for every fixed dimension *d,* ERM_H can be implemented in time which is polynomial in $1/\varepsilon$, $1/\delta$, *d* (measures the complexity of the $\mathcal{H}_{rec}^{d}$) therefore we have efficient learning (see the formal definition later)

- however, as a function of d the runtime of the algorithm implementing the ERM_H presented is exponential in *d.* It can be proved that there is no better algorithm (unless P = NP) than the one proposed.

Formal definition of efficient learning

Definition 1

Given a function $f : (0,1)^2 \rightarrow \mathrm{N}$, a learning task $(\mathcal{Z}, \mathcal{H}, \ell)$, and a learning algorithm A, we say that A solves the learning task in time *O(f)* if there exists some constant number c, such that for every probability distribution $\mathcal{D}$ over $\mathcal{Z}$, and input $\varepsilon, \delta \in (0,1)$, when A has access to samples generated i.i.d by $\mathcal{D}$, we have that:

- A terminates after performing at most $c * f(\varepsilon, \delta)$ operations;
- the output of A, denoted h_A, can be applied to predict the label of a new example while performing at most $c * f(\varepsilon, \delta)$ operations;
- the output of A is probably approximately correct; namely, with probability of at least $1 - \delta$ (over the random samples A receives):

$$L_{\mathcal{D}}(h_A) \leq \min_h L_{\mathcal{D}}(h) + \varepsilon$$

Formal definition of efficient learning

Definition (for graded hypothesis spaces)

Consider a sequence of learning problems, $(\mathcal{Z}_n, \mathcal{H}_n, \ell_n)_{n=1,2,\ldots}$ where problem n is defined by a domain $\mathcal{Z}_n$, a hypothesis class $\mathcal{H}_n$, and a loss function ℓ_n. Let A be a learning algorithm designed for solving learning problems of this form. Given a function g: $\mathrm{N} \times (0,1)^2 \rightarrow \mathrm{N}$, we say that the runtime of A with respect to the preceding sequence is $O(g)$, if for all n, A solves the problem $(\mathcal{Z}_n, \mathcal{H}_n, \ell_n)$ in time $O(f_n)$, where $f_n : (0,1)^2 \rightarrow \mathrm{N}$ is defined by $f_n(\varepsilon, \delta) = g(n, \varepsilon, \delta)$.

We say that A is an *efficient* PAC algorithm with respect to a sequence $(\mathcal{Z}_n, \mathcal{H}_n, \ell_n)$ if its runtime is $O(p(n, 1/\varepsilon, 1/\delta))$ for some polynomial p.

Formal definition of efficient PAC learning (Valiant 1984)

In 1984, Leslie Valiant defined efficient PAC learning: PAC learnability + require the number of examples and the runtime of the algorithm A (training + testing) to be polynomial in $1/\varepsilon$, $1/\delta$, n.

Example:

- $\mathcal{U}_n = \{h: B^n \rightarrow \{0,1\}\}$ - the concept class formed by all subsets of B^n
- $|\mathcal{U}_n| = 2^{2^n}$ – finite, so is PAC learnable with $m_{\mathcal{H}}(\varepsilon, \delta)$ in the order of m:

$$m \geq \left\lceil \frac{1}{\varepsilon}\left(2^n \log(2) + \log(\frac{1}{\delta})\right)\right\rceil$$

- sample complexity exponential in n, number of variables
- it is not efficient PAC-learnable in any practical sense (need polynomial sample complexity)