Learning via Uniform Convergence

Uniform Convergence Is Sufficient for Learnability

The idea behind the learning condition discussed in this chapter is very simple. Recall that, given a hypothesis class, $\mathcal{H}$, the ERM learning paradigm works as follows: Upon receiving a training sample, S, the learner evaluates the risk (or error) of each h in $\mathcal{H}$ on the given sample and outputs a member of $\mathcal{H}$ that minimizes this empirical risk. The hope is that an h that minimizes the empirical risk with respect to S is a risk minimizer (or has risk close to the minimum) with respect to the true data probability distribution as well. For that, it suffices to ensure that the empirical risks of all members of $\mathcal{H}$ are good approximations of their true risk. Put another way, we need that uniformly over all hypotheses in the hypothesis class, the empirical risk will be close to the true risk, as formalized in the following.

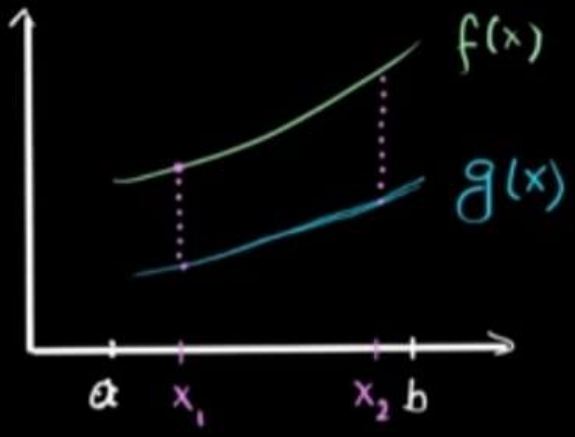
DEFINITION 4.1 (ϵ-representative sample) A training set S is called ϵ-representative (w.r.t. domain Z, hypothesis class $\mathcal{H}$, loss function ℓ, and distribution $\mathcal{D}$) if

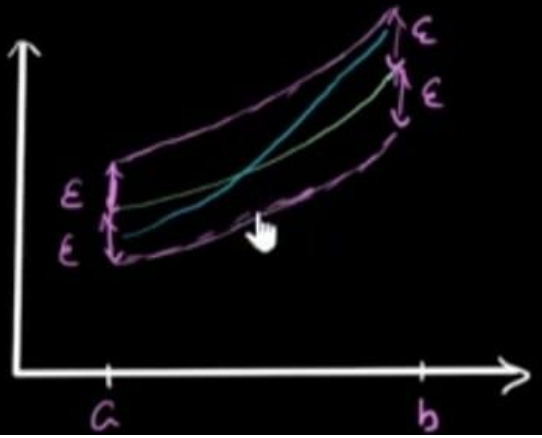
$$\forall h \in \mathcal{H}, \quad |L_S(h) - L_{\mathcal{D}}(h)| \leq \epsilon.$$

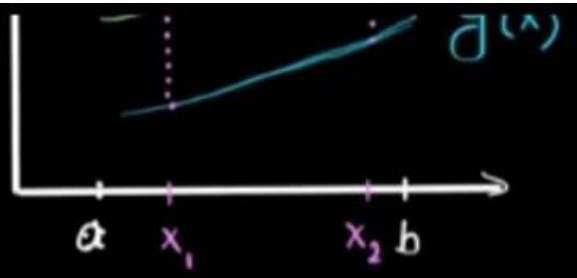
Uniform Convergence of a Sequence of Functions

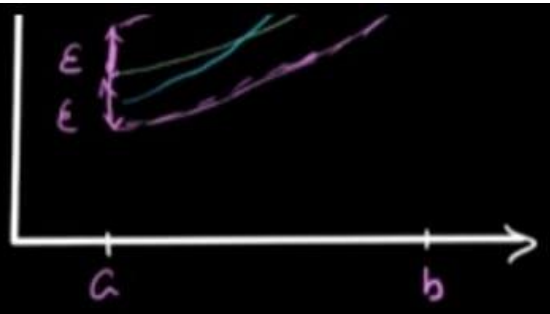
$a \in \mathbb{R}$ $\quad |a - 4| < 1$

$|f(x) - g(x)| < \varepsilon$





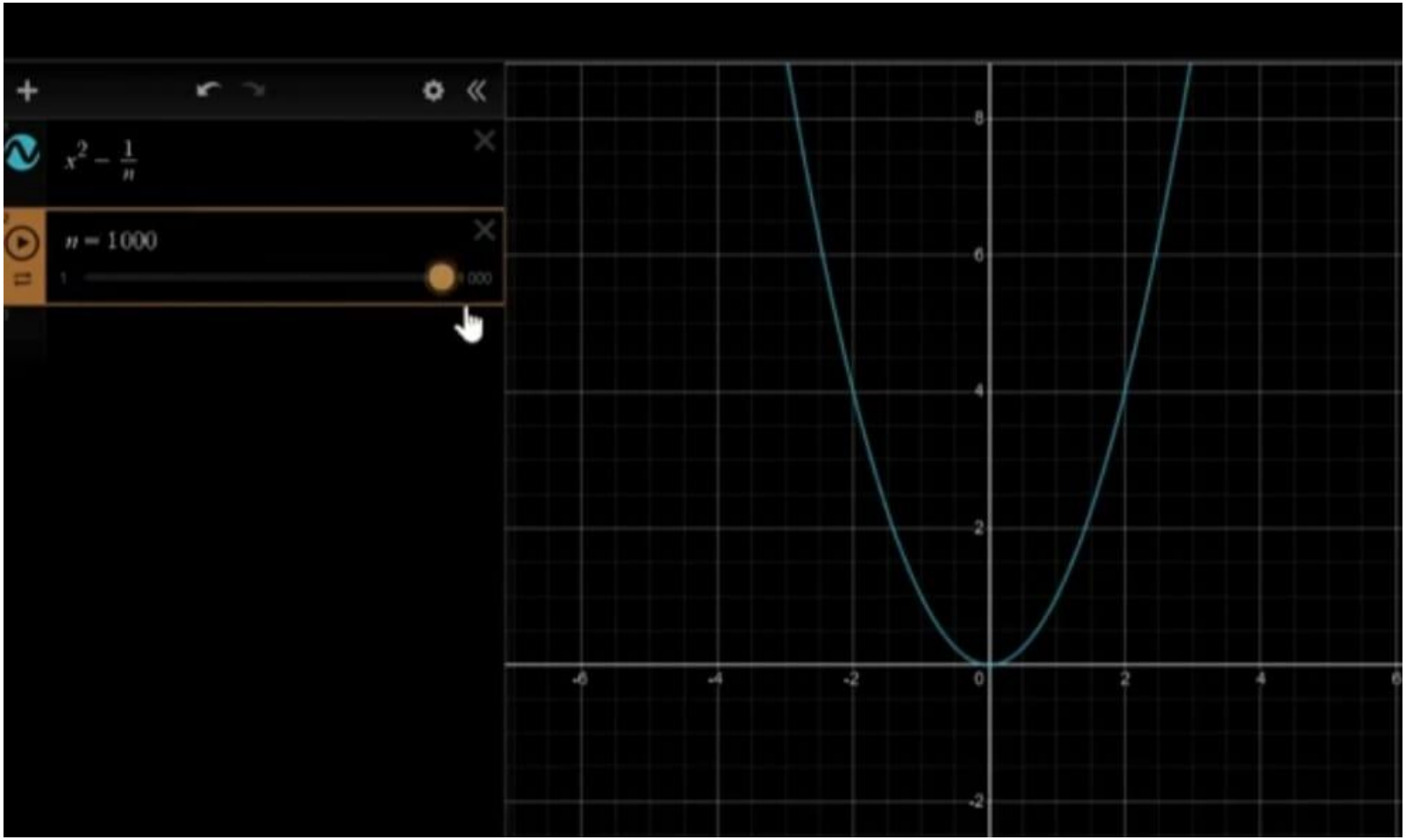




$(f_n)_{n\in\mathbb{N}}$ defined by $f_n(x) = x^2 - \frac{1}{n}$

$\left(x^2-1,\ x^2-\frac{1}{2},\ x^2-\frac{1}{3},\ \dots \right)$

$\lim_{n\to\infty} x^2 - \frac{1}{n} = x^2$


$x^2 - \frac{1}{n}$
$n = 1000$
1
000
8
6
4
2
-2
-6
-4
-2
0
2
4

The next simple lemma states that whenever the sample is $(\epsilon/2)$-representative, the ERM learning rule is guaranteed to return a good hypothesis.

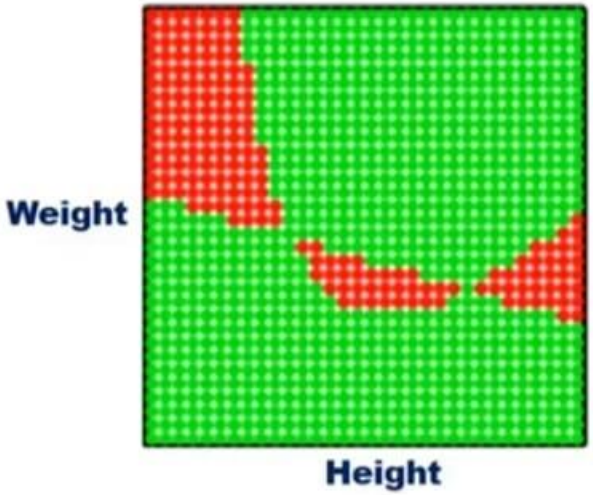
LEMMA 4.2 *Assume that a training set S is $\frac{\epsilon}{2}$-representative (w.r.t. domain Z, hypothesis class $\mathcal{H}$, loss function ℓ, and distribution $\mathcal{D}$). Then, any output of $\mathrm{ERM}_{\mathcal{H}}(S)$, namely, any $h_S \in \mathrm{argmin}_{h \in \mathcal{H}} L_S(h)$, satisfies*

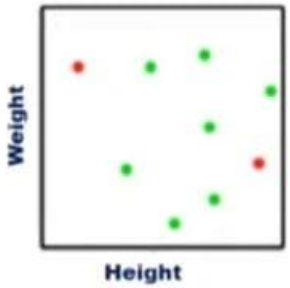
$$L_{\mathcal{D}}(h_S) \leq \min_{h \in \mathcal{H}} L_{\mathcal{D}}(h) + \epsilon.$$

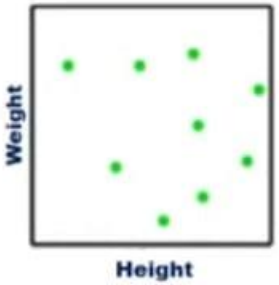
DEFINITION 4.3 (Uniform Convergence) We say that a hypothesis class $\mathcal{H}$ has the *uniform convergence property* (w.r.t. a domain Z and a loss function ℓ) if there exists a function $m_{\mathcal{H}}^{\text{UC}} : (0,1)^2 \to \mathbb{N}$ such that for every $\epsilon, \delta \in (0,1)$ and for every probability distribution $\mathcal{D}$ over Z, if S is a sample of $m \geq m_{\mathcal{H}}^{\text{UC}}(\epsilon, \delta)$ examples drawn i.i.d. according to $\mathcal{D}$, then, with probability of at least $1 - \delta$, S is ϵ-representative.

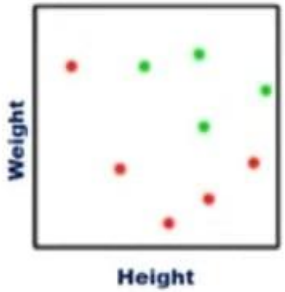
Similar to the definition of sample complexity for PAC learning, the function $m_{\mathcal{H}}^{\text{UC}}$ measures the (minimal) sample complexity of obtaining the uniform convergence property, namely, how many examples we need to ensure that with probability of at least $1 - \delta$ the sample would be ϵ-representative.

The term *uniform* here refers to having a fixed sample size that works for all members of $\mathcal{H}$ and over all possible probability distributions over the domain.

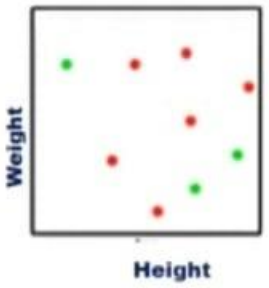

Weight
Height

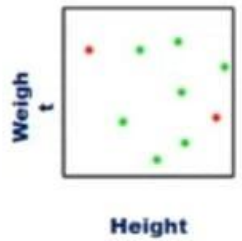

Weight
Height

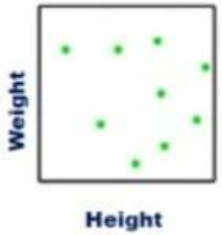

Weight
Height

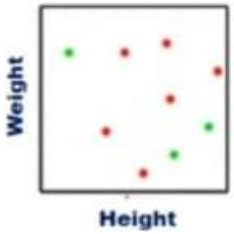

Weight
Height

. . .


Weight
Height







Error(H1)	Error(h2)
0.04	0.04
0.03	0.035
0.09	0.039
0.06	0.06
0.025	0.025
0.049	0.059
0.04	0.04
0.03	0.03
0.05	0.55
0.043	0.043

$\varepsilon = 0.05$ $\delta = 0.20$

P (H1) = 8/10=0.80

P (H1) = 8/10=0.80 >=1- 0.20

Hence H1 is probably approximately correct

P(H2)= 7/10=0.70

P(H2)= 7/10=0.70 <1-0.20

Hence H2 is not probably approximately correct

COROLLARY 4.4 *If a class $\mathcal{H}$ has the uniform convergence property with a function $m_{\mathcal{H}}^{UC}$ then the class is agnostically PAC learnable with the sample complexity $m_{\mathcal{H}}(\epsilon, \delta) \leq m_{\mathcal{H}}^{UC}(\epsilon/2, \delta)$. Furthermore, in that case, the $\mathrm{ERM}_{\mathcal{H}}$ paradigm is a successful agnostic PAC learner for $\mathcal{H}$.*

Nonuniform Learnability

"Nonuniform learnability" allows the sample size to be nonuniform with respect to the different hypotheses with which the learner is competing. We say that a hypothesis h is (ϵ, δ)-competitive with another hypothesis h' if, with probability higher than $(1 - \delta)$,

$$L_{\mathcal{D}}(h) \leq L_{\mathcal{D}}(h') + \epsilon.$$

In PAC learnability, this notion of "competitiveness" is not very useful, as we are looking for a hypothesis with an absolute low risk (in the realizable case) or with a low risk compared to the minimal risk achieved by hypotheses in our class (in the agnostic case). Therefore, the sample size depends only on the accuracy and confidence parameters. In nonuniform learnability, however, we allow the sample size to be of the form $m_{\mathcal{H}}(\epsilon, \delta, h)$; namely, it depends also on the h with which we are competing.

DEFINITION 7.1 A hypothesis class $\mathcal{H}$ is *nonuniformly learnable* if there exist a learning algorithm, A, and a function $m_{\mathcal{H}}^{\text{NUL}} : (0,1)^2 \times \mathcal{H} \to \mathbb{N}$ such that, for every $\epsilon, \delta \in (0,1)$ and for every $h \in \mathcal{H}$, if $m \geq m_{\mathcal{H}}^{\text{NUL}}(\epsilon, \delta, h)$ then for every distribution $\mathcal{D}$, with probability of at least $1 - \delta$ over the choice of $S \sim \mathcal{D}^m$, it holds that

$$L_{\mathcal{D}}(A(S)) \leq L_{\mathcal{D}}(h) + \epsilon.$$

At this point it might be useful to recall the definition of agnostic PAC learnability (Definition 3.3):

A hypothesis class $\mathcal{H}$ is agnostically PAC learnable if there exist a learning algorithm, A, and a function $m_{\mathcal{H}} : (0,1)^2 \to \mathbb{N}$ such that, for every $\epsilon, \delta \in (0,1)$ and for every distribution $\mathcal{D}$, if $m \geq m_{\mathcal{H}}(\epsilon, \delta)$, then with probability of at least $1 - \delta$ over the choice of $S \sim \mathcal{D}^m$ it holds that

$$L_{\mathcal{D}}(A(S)) \leq \min_{h' \in \mathcal{H}} L_{\mathcal{D}}(h') + \epsilon.$$

Note that this implies that for every $h \in \mathcal{H}$

$$L_{\mathcal{D}}(A(S)) \leq L_{\mathcal{D}}(h) + \epsilon.$$

In both types of learnability, we require that the output hypothesis will be (ϵ, δ)-competitive with every other hypothesis in the class. But the difference between these two notions of learnability is the question of whether the sample size m may depend on the hypothesis h to which the error of $A(S)$ is compared. Note that that nonuniform learnability is a relaxation of agnostic PAC learnability. That is, if a class is agnostic PAC learnable then it is also nonuniformly learnable.

Characterizing Nonuniform Learnability

Our goal now is to characterize nonuniform learnability. In the previous chapter we have found a crisp characterization of PAC learnable classes, by showing that a class of binary classifiers is agnostic PAC learnable if and only if its VC-dimension is finite. In the following theorem we find a different characterization for nonuniform learnable classes for the task of binary classification.

THEOREM 7.2 *A hypothesis class $\mathcal{H}$ of binary classifiers is nonuniformly learnable if and only if it is a countable union of agnostic PAC learnable hypothesis classes.*

THEOREM 7.3 *Let $\mathcal{H}$ be a hypothesis class that can be written as a countable union of hypothesis classes, $\mathcal{H} = \bigcup_{n \in \mathbb{N}} \mathcal{H}_n$, where each $\mathcal{H}_n$ enjoys the uniform convergence property. Then, $\mathcal{H}$ is nonuniformly learnable.*

Structural Risk Minimization

So far, we have encoded our prior knowledge by specifying a hypothesis class $\mathcal{H}$, which we believe includes a good predictor for the learning task at hand. Yet another way to express our prior knowledge is by specifying preferences over hypotheses within $\mathcal{H}$. In the Structural Risk Minimization (SRM) paradigm, we do so by first assuming that $\mathcal{H}$ can be written as $\mathcal{H} = \bigcup_{n \in \mathbb{N}} \mathcal{H}_n$ and then specifying a weight function, $w : \mathbb{N} \to [0, 1]$, which assigns a weight to each hypothesis class, $\mathcal{H}_n$, such that a higher weight reflects a stronger preference for the hypothesis class. In this section we discuss how to learn with such prior knowledge. In the next section we describe a couple of important weighting schemes, including Minimum Description Length.

Concretely, let $\mathcal{H}$ be a hypothesis class that can be written as $\mathcal{H} = \bigcup_{n \in \mathbb{N}} \mathcal{H}_n$. For example, $\mathcal{H}$ may be the class of all polynomial classifiers where each $\mathcal{H}_n$ is the class of polynomial classifiers of degree n (see Example 7.1). Assume that for each n, the class $\mathcal{H}_n$ enjoys the uniform convergence property (see Definition 4.3 in Chapter 4) with a sample complexity function $m^{\text{UC}}_{\mathcal{H}_n}(\epsilon, \delta)$. Let us also define the function $\epsilon_n : \mathbb{N} \times (0,1) \to (0,1)$ by

$$\epsilon_n(m, \delta) = \min\{\epsilon \in (0,1) : m^{\text{UC}}_{\mathcal{H}_n}(\epsilon, \delta) \leq m\}. \tag{7.1}$$

In words, we have a fixed sample size m, and we are interested in the lowest possible upper bound on the gap between empirical and true risks achievable by using a sample of m examples.

From the definitions of uniform convergence and ϵ_n, it follows that for every m and δ, with probability of at least $1-\delta$ over the choice of $S \sim \mathcal{D}^m$ we have that

$$\forall h \in \mathcal{H}_n, \quad |L_{\mathcal{D}}(h) - L_S(h)| \leq \epsilon_n(m, \delta). \tag{7.2}$$

Let $w : \mathbb{N} \to [0,1]$ be a function such that $\sum_{n=1}^{\infty} w(n) \leq 1$. We refer to w as a *weight function* over the hypothesis classes $\mathcal{H}_1, \mathcal{H}_2, \ldots$. Such a weight function can reflect the importance that the learner attributes to each hypothesis class, or some measure of the complexity of different hypothesis classes. If $\mathcal{H}$ is a finite union of N hypothesis classes, one can simply assign the same weight of $1/N$ to all hypothesis classes. This equal weighting corresponds to no a priori preference to any hypothesis class. Of course, if one believes (as prior knowledge) that a certain hypothesis class is more likely to contain the correct target function, then it should be assigned a larger weight, reflecting this prior knowledge. When $\mathcal{H}$ is a (countable) infinite union of hypothesis classes, a uniform weighting is not possible but many other weighting schemes may work. For example, one can choose $w(n) = \frac{6}{\pi^2 n^2}$ or $w(n) = 2^{-n}$. Later in this chapter we will provide another convenient way to define weighting functions using description languages.

The SRM rule follows a "bound minimization" approach. This means that the goal of the paradigm is to find a hypothesis that minimizes a certain upper bound on the true risk. The bound that the SRM rule wishes to minimize is given in the following theorem.

THEOREM 7.4 *Let $w : \mathbb{N} \to [0,1]$ be a function such that $\sum_{n=1}^{\infty} w(n) \leq 1$. Let $\mathcal{H}$ be a hypothesis class that can be written as $\mathcal{H} = \bigcup_{n \in \mathbb{N}} \mathcal{H}_n$, where for each n, $\mathcal{H}_n$ satisfies the uniform convergence property with a sample complexity function $m^{UC}_{\mathcal{H}_n}$. Let ϵ_n be as defined in Equation (7.1). Then, for every $\delta \in (0,1)$ and distribution $\mathcal{D}$, with probability of at least $1-\delta$ over the choice of $S \sim \mathcal{D}^m$, the following bound holds (simultaneously) for every $n \in \mathbb{N}$ and $h \in \mathcal{H}_n$.*

$$|L_{\mathcal{D}}(h) - L_S(h)| \leq \epsilon_n(m, w(n) \cdot \delta).$$

Therefore, for every $\delta \in (0,1)$ *and distribution* $\mathcal{D}$*, with probability of at least* $1-\delta$ *it holds that*

$$\forall h \in \mathcal{H}, \quad L_{\mathcal{D}}(h) \leq L_S(h) + \min_{n:h\in\mathcal{H}_n} \epsilon_n(m, w(n)\cdot\delta). \tag{7.3}$$

Applying the union bound over $n = 1, 2, \ldots$, we obtain that with probability of at least $1 - \sum_n \delta_n = 1 - \delta \sum_n w(n) \geq 1 - \delta$, the preceding holds for all n, which concludes our proof. □

Denote

$$n(h) = \min\{n : h \in \mathcal{H}_n\}, \tag{7.4}$$

The SRM paradigm searches for h that minimizes this bound, as formalized in the following pseudocode:

`Structural Risk Minimization (SRM)`

prior knowledge:
$\mathcal{H} = \bigcup_n \mathcal{H}_n$ where $\mathcal{H}_n$ has uniform convergence with $m^{\text{UC}}_{\mathcal{H}_n}$
$w : \mathbb{N} \to [0,1]$ where $\sum_n w(n) \leq 1$
define: ϵ_n as in Equation (7.1) ; $n(h)$ as in Equation (7.4)
input: training set $S \sim \mathcal{D}^m$, confidence δ
output: $h \in \operatorname{argmin}_{h \in \mathcal{H}} \left[L_S(h) + \epsilon_{n(h)}(m, w(n(h)) \cdot \delta) \right]$

THEOREM 7.5 *Let $\mathcal{H}$ be a hypothesis class such that $\mathcal{H} = \bigcup_{n \in \mathbb{N}} \mathcal{H}_n$, where each $\mathcal{H}_n$ has the uniform convergence property with sample complexity $m^{UC}_{\mathcal{H}_n}$. Let $w : \mathbb{N} \to [0,1]$ be such that $w(n) = \frac{6}{n^2 \pi^2}$. Then, $\mathcal{H}$ is nonuniformly learnable using the SRM rule with rate*

$$m^{NUL}_{\mathcal{H}}(\epsilon, \delta, h) \leq m^{UC}_{\mathcal{H}_{n(h)}}\left(\epsilon/2, \frac{6\delta}{(\pi n(h))^2}\right).$$

Indeed, there is no inherent generalizability difference between hypotheses. The crucial aspect here is the dependency order between the initial choice of language (or, preference over hypotheses) and the training set. As we know from the basic Hoeffding's bound (Equation (4.2)), if we commit to any hypothesis *before* seeing the data, then we are guaranteed a rather small estimation error term $L_{\mathcal{D}}(h) \leq L_S(h) + \sqrt{\frac{\ln(2/\delta)}{2m}}$. Choosing a description language (or, equivalently, some weighting of hypotheses) is a weak form of committing to a hypothesis. Rather than committing to a single hypothesis, we spread out our commitment among many. As long as it is done independently of the training sample, our generalization bound holds. Just as the choice of a single hypothesis to be evaluated by a sample can be arbitrary, so is the choice of description language.

Other Notions of Learnability – Consistency

The notion of learnability can be further relaxed by allowing the needed sample sizes to depend not only on ϵ, δ, and h but also on the underlying data-generating probability distribution $\mathcal{D}$ (that is used to generate the training sample and to determine the risk). This type of performance guarantee is captured by the notion of *consistency*[1] of a learning rule.

DEFINITION 7.8 (Consistency) Let Z be a domain set, let $\mathcal{P}$ be a set of probability distributions over Z, and let $\mathcal{H}$ be a hypothesis class. A learning rule A is *consistent* with respect to $\mathcal{H}$ and $\mathcal{P}$ if there exists a function $m_{\mathcal{H}}^{\text{CON}} : (0,1)^2 \times \mathcal{H} \times \mathcal{P} \to \mathbb{N}$ such that, for every $\epsilon, \delta \in (0,1)$, every $h \in \mathcal{H}$, and every $\mathcal{D} \in \mathcal{P}$, if $m \geq m_{\mathcal{H}}^{\text{NUL}}(\epsilon, \delta, h, \mathcal{D})$ then with probability of at least $1 - \delta$ over the choice of $S \sim \mathcal{D}^m$ it holds that

$$L_{\mathcal{D}}(A(S)) \leq L_{\mathcal{D}}(h) + \epsilon.$$

Example 7.4 Consider the classification prediction algorithm `Memorize` defined as follows. The algorithm memorizes the training examples, and, given a test point x, it predicts the majority label among all labeled instances of x that exist in the training sample (and some fixed default label if no instance of x appears in the training set). It is possible to show (see Exercise 6) that the `Memorize` algorithm is universally consistent for every countable domain $\mathcal{X}$ and a finite label set $\mathcal{Y}$ (w.r.t. the zero-one loss).

Intuitively, it is not obvious that the `Memorize` algorithm should be viewed as a *learner*, since it lacks the aspect of generalization, namely, of using observed data to predict the labels of unseen examples. The fact that `Memorize` is a consistent algorithm for the class of all functions over any countable domain set therefore raises doubt about the usefulness of consistency guarantees. Furthermore, the sharp-eyed reader may notice that the "bad learner" we introduced in Chapter 2, which led to overfitting, is in fact the `Memorize` algorithm.

Discussing the Different Notions of Learnability

What Is the Risk of the Learned Hypothesis?

The first possible goal of deriving performance guarantees on a learning algorithm is bounding the risk of the output predictor. Here, both PAC learning and nonuniform learning give us an upper bound on the true risk of the learned hypothesis based on its empirical risk. Consistency guarantees do not provide such a bound. However, it is always possible to estimate the risk of the output predictor using a validation set (as will be described in Chapter 11).

How Many Examples Are Required to Be as Good as the Best Hypothesis in $\mathcal{H}$?

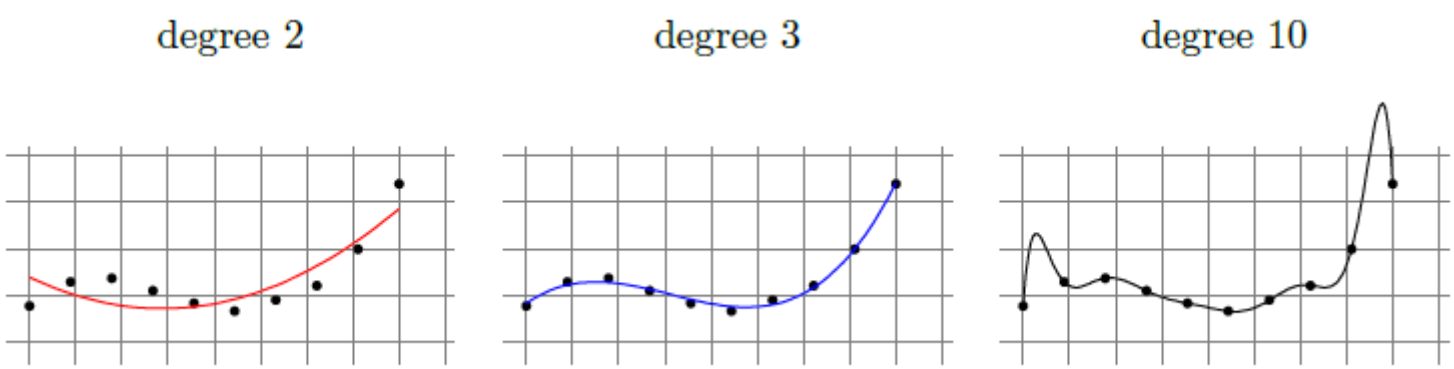
When approaching a learning problem, a natural question is how many examples we need to collect in order to learn it. Here, PAC learning gives a crisp answer. However, for both nonuniform learning and consistency, we do not know in advance how many examples are required to learn $\mathcal{H}$. In nonuniform learning this number depends on the best hypothesis in $\mathcal{H}$, and in consistency it also depends on the underlying distribution. In this sense, PAC learning is the only useful definition of learnability. On the flip side, one should keep in mind that even if the estimation error of the predictor we learn is small, its risk may still be large if $\mathcal{H}$ has a large approximation error. So, for the question "How many examples are required to be as good as the Bayes optimal predictor?" even PAC guarantees do not provide us with a crisp answer. This reflects the fact that the usefulness of PAC learning relies on the quality of our prior knowledge.

PAC guarantees also help us to understand what we should do next if our learning algorithm returns a hypothesis with a large risk, since we can bound the part of the error that stems from estimation error and therefore know how much of the error is attributed to approximation error. If the approximation error is large, we know that we should use a different hypothesis class. Similarly, if a nonuniform algorithm fails, we can consider a different weighting function over (subsets of) hypotheses. However, when a consistent algorithm fails, we have no idea whether this is because of the estimation error or the approximation error. Furthermore, even if we are sure we have a problem with the estimation error term, we do not know how many more examples are needed to make the estimation error small.

How to Learn? How to Express Prior Knowledge?

Maybe the most useful aspect of the theory of learning is in providing an answer to the question of "how to learn." The definition of PAC learning yields the limitation of learning (via the No-Free-Lunch theorem) and the necessity of prior knowledge. It gives us a crisp way to encode prior knowledge by choosing a hypothesis class, and once this choice is made, we have a generic learning rule – ERM. The definition of nonuniform learnability also yields a crisp way to encode prior knowledge by specifying weights over (subsets of) hypotheses of $\mathcal{H}$. Once this choice is made, we again have a generic learning rule – SRM. The SRM rule is also advantageous in model selection tasks, where prior knowledge is partial.

Consider the problem of fitting a one dimensional polynomial to data; namely, our goal is to learn a function, $h : \mathbb{R} \to \mathbb{R}$, and as prior knowledge we consider the hypothesis class of polynomials. However, we might be uncertain regarding which degree d would give the best results for our data set: A small degree might not fit the data well (i.e., it will have a large approximation error), whereas a high degree might lead to overfitting (i.e., it will have a large estimation error). In the following we depict the result of fitting a polynomial of degrees 2, 3, and 10 to the same training set.



It is easy to see that the empirical risk decreases as we enlarge the degree. Therefore, if we choose $\mathcal{H}$ to be the class of all polynomials up to degree 10 then the ERM rule with respect to this class would output a 10 degree polynomial and would overfit. On the other hand, if we choose too small a hypothesis class, say, polynomials up to degree 2, then the ERM would suffer from underfitting (i.e., a large approximation error). In contrast, we can use the SRM rule on the set of all polynomials, while ordering subsets of $\mathcal{H}$ according to their degree, and this will yield a 3rd degree polynomial since the combination of its empirical risk and the bound on its estimation error is the smallest. In other words, the SRM rule enables us to select the right model on the basis of the data itself. The price we pay for this flexibility (besides a slight increase of the estimation error relative to PAC learning w.r.t. the optimal degree) is that we do not know in advance how many examples are needed to compete with the best hypothesis in $\mathcal{H}$.

Unlike the notions of PAC learnability and nonuniform learnability, the definition of consistency does not yield a natural learning paradigm or a way to encode prior knowledge. In fact, in many cases there is no need for prior knowledge at all. For example, we saw that even the `Memorize` algorithm, which intuitively should not be called a learning algorithm, is a consistent algorithm for any class defined over a countable domain and a finite label set. This hints that consistency is a very weak requirement.

Which Learning Algorithm Should We Prefer?

One may argue that even though consistency is a weak requirement, it is desirable that a learning algorithm will be consistent with respect to the set of all functions from $\mathcal{X}$ to $\mathcal{Y}$, which gives us a guarantee that for enough training examples, we will always be as good as the Bayes optimal predictor. Therefore, if we have two algorithms, where one is consistent and the other one is not consistent, we should prefer the consistent algorithm. However, this argument is problematic for two reasons. First, maybe it is the case that for most "natural" distributions we will observe in practice that the sample complexity of the consistent algorithm will be so large so that in every practical situation we will not obtain enough examples to enjoy this guarantee. Second, it is not very hard to make any PAC or nonuniform learner consistent with respect to the class of all functions from $\mathcal{X}$ to $\mathcal{Y}$. Concretely, consider a countable domain, $\mathcal{X}$, a finite label set $\mathcal{Y}$, and a hypothesis class, $\mathcal{H}$, of functions from $\mathcal{X}$ to $\mathcal{Y}$. We can make any nonuniform learner for $\mathcal{H}$ be consistent with respect to the class of *all* classifiers from $\mathcal{X}$ to $\mathcal{Y}$ using the following simple trick: Upon receiving a training set, we will first run the nonuniform learner over the training set, and then we will obtain a bound on the true risk of the learned predictor. If this bound is small enough we are done. Otherwise, we revert to the `Memorize` algorithm. This simple modification makes the algorithm consistent with respect to all functions from $\mathcal{X}$ to $\mathcal{Y}$. Since it is easy to make any algorithm consistent, it may not be wise to prefer one algorithm over the other just because of consistency considerations.

The Runtime of Learning

So far in the book we have studied the statistical perspective of learning, namely, how many samples are needed for learning. In other words, we focused on the amount of information learning requires. However, when considering automated learning, computational resources also play a major role in determining the complexity of a task: that is, how much *computation* is involved in carrying out a learning task. Once a sufficient training sample is available to the learner, there is some computation to be done to extract a hypothesis or figure out the label of a given test instance. These computational resources are crucial in any practical application of machine learning. We refer to these two types of resources as the *sample complexity* and the *computational complexity*. In this chapter, we turn our attention to the computational complexity of learning.

The actual runtime (in seconds) of an algorithm depends on the specific machine the algorithm is being implemented on (e.g., what the clock rate of the machine's CPU is). To avoid dependence on the specific machine, it is common to analyze the runtime of algorithms in an asymptotic sense. For example, we say that the computational complexity of the merge-sort algorithm, which sorts a list of n items, is $O(n \log(n))$. This implies that we can implement the algorithm on any machine that satisfies the requirements of some accepted abstract model of computation, and the actual runtime in seconds will satisfy the following: there exist constants c and n_0, which can depend on the actual machine, such that, for any value of $n > n_0$, the runtime in seconds of sorting any n items will be at most $c\, n \log(n)$. It is common to use the term *feasible* or *efficiently computable* for tasks that can be performed by an algorithm whose running time is $O(p(n))$ for some polynomial function p. One should note that this type of analysis depends on defining what is the input size n of any instance to which the algorithm is expected to be applied. For "purely algorithmic" tasks, as discussed in the common computational complexity literature, this input size is clearly defined; the algorithm gets an input instance, say, a list to be sorted, or an arithmetic operation to be calculated, which has a well defined size (say, the number of bits in its representation). For machine learning tasks, the notion of an input size is not so clear. An algorithm aims to detect some pattern in a data set and can only access random samples of that data.

Computational Complexity of Learning

Recall that a learning algorithm has access to a domain of examples, Z, a hypothesis class, $\mathcal{H}$, a loss function, ℓ, and a training set of examples from Z that are sampled i.i.d. according to an unknown distribution $\mathcal{D}$. Given parameters ϵ, δ, the algorithm should output a hypothesis h such that with probability of at least $1 - \delta$,

$$L_{\mathcal{D}}(h) \leq \min_{h' \in \mathcal{H}} L_{\mathcal{D}}(h') + \epsilon.$$

As mentioned before, the actual runtime of an algorithm in seconds depends on the specific machine. To allow machine independent analysis, we use the standard approach in computational complexity theory. First, we rely on a notion of an abstract machine, such as a Turing machine (or a Turing machine over the reals (Blum, Shub & Smale 1989)). Second, we analyze the runtime in an asymptotic sense, while ignoring constant factors, thus the specific machine is not important as long as it implements the abstract machine. Usually, the asymptote is with respect to the size of the input to the algorithm. For example, for the merge-sort algorithm mentioned before, we analyze the runtime as a function of the number of items that need to be sorted.

In the context of learning algorithms, there is no clear notion of "input size." One might define the input size to be the size of the training set the algorithm receives, but that would be rather pointless. If we give the algorithm a very large number of examples, much larger than the sample complexity of the learning problem, the algorithm can simply ignore the extra examples. Therefore, a larger training set does not make the learning problem more difficult, and, consequently, the runtime available for a learning algorithm should not increase as we increase the size of the training set. Just the same, we can still analyze the runtime as a function of natural parameters of the problem such as the target accuracy, the confidence of achieving that accuracy, the dimensionality of the domain set, or some measures of the complexity of the hypothesis class with which the algorithm's output is compared.

To illustrate this, consider a learning algorithm for the task of learning axis aligned rectangles. A specific problem of learning axis aligned rectangles is derived by specifying ϵ, δ, and the dimension of the instance space. We can define a sequence of problems of the type "rectangles learning" by fixing ϵ, δ and varying the dimension to be $d = 2, 3, 4, \ldots$. We can also define another sequence of "rectangles learning" problems by fixing d, δ and varying the target accuracy to be $\epsilon = \frac{1}{2}, \frac{1}{3}, \ldots$. One can of course choose other sequences of such problems. Once a sequence of the problems is fixed, one can analyze the asymptotic runtime as a function of variables of that sequence.

Before we introduce the formal definition, there is one more subtlety we need to tackle. On the basis of the preceding, a learning algorithm can "cheat," by transferring the computational burden to the output hypothesis. For example, the algorithm can simply define the output hypothesis to be the function that stores the training set in its memory, and whenever it gets a test example x it calculates the ERM hypothesis on the training set and applies it on x. Note that in this case, our algorithm has a fixed output (namely, the function that we have just described) and can run in constant time. However, learning is still hard – the hardness is now in implementing the output classifier to obtain a label prediction. To prevent this "cheating," we shall require that the output of a learning algorithm must be applied to predict the label of a new example in time that does not exceed the runtime of training (that is, computing the output classifier from the input training sample). In the next subsection the advanced reader may find a formal definition of the computational complexity of learning.

Formal Definition

The definition that follows relies on a notion of an underlying abstract machine, which is usually either a Turing machine or a Turing machine over the reals. We will measure the computational complexity of an algorithm using the number of "operations" it needs to perform, where we assume that for any machine that implements the underlying abstract machine there exists a constant c such that any such "operation" can be performed on the machine using c seconds.

DEFINITION 8.1 (The Computational Complexity of a Learning Algorithm) We define the complexity of learning in two steps. First we consider the computational complexity of a fixed learning problem (determined by a triplet $(Z, \mathcal{H}, \ell)$ – a domain set, a benchmark hypothesis class, and a loss function). Then, in the second step we consider the rate of change of that complexity along a sequence of such tasks.

1. Given a function $f : (0,1)^2 \to \mathbb{N}$, a learning task $(Z, \mathcal{H}, \ell)$, and a learning algorithm, $\mathcal{A}$, we say that $\mathcal{A}$ solves the learning task in time $O(f)$ if there exists some constant number c, such that for every probability distribution $\mathcal{D}$ over Z, and input $\epsilon,\ \delta \in (0,1)$, when $\mathcal{A}$ has access to samples generated i.i.d. by $\mathcal{D}$,
   - $\mathcal{A}$ terminates after performing at most $cf(\epsilon,\ \delta)$ operations
   - The output of $\mathcal{A}$, denoted $h_{\mathcal{A}}$, can be applied to predict the label of a new example while performing at most $cf(\epsilon,\ \delta)$ operations
   - The output of $\mathcal{A}$ is probably approximately correct; namely, with probability of at least $1-\delta$ (over the random samples $\mathcal{A}$ receives), $L_{\mathcal{D}}(h_{\mathcal{A}}) \leq \min_{h' \in \mathcal{H}} L_{\mathcal{D}}(h') + \epsilon$

2. Consider a sequence of learning problems, $(Z_n, \mathcal{H}_n, \ell_n)_{n=1}^{\infty}$, where problem n is defined by a domain Z_n, a hypothesis class $\mathcal{H}_n$, and a loss function ℓ_n.

   Let $\mathcal{A}$ be a learning algorithm designed for solving learning problems of this form. Given a function $g : \mathbb{N} \times (0,1)^2 \to \mathbb{N}$, we say that the runtime of $\mathcal{A}$ with respect to the preceding sequence is $O(g)$, if for all n, $\mathcal{A}$ solves the problem $(Z_n, \mathcal{H}_n, \ell_n)$ in time $O(f_n)$, where $f_n : (0,1)^2 \to \mathbb{N}$ is defined by $f_n(\epsilon, \delta) = g(n, \epsilon, \delta)$.

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

From this definition we see that the question whether a general learning problem can be solved efficiently depends on how it can be broken into a sequence of specific learning problems. For example, consider the problem of learning a finite hypothesis class. As we showed in previous chapters, the ERM rule over $\mathcal{H}$ is guaranteed to (ϵ, δ)-learn $\mathcal{H}$ if the number of training examples is order of $m_{\mathcal{H}}(\epsilon, \delta) = \log(|\mathcal{H}|/\delta)/\epsilon^2$. Assuming that the evaluation of a hypothesis on an example takes a constant time, it is possible to implement the ERM rule in time $O(|\mathcal{H}|\, m_{\mathcal{H}}(\epsilon, \delta))$ by performing an exhaustive search over $\mathcal{H}$ with a training set of size $m_{\mathcal{H}}(\epsilon, \delta)$. For any fixed finite $\mathcal{H}$, the exhaustive search algorithm runs in polynomial time. Furthermore, if we define a sequence of problems in which $|\mathcal{H}_n| = n$, then the exhaustive search is still considered to be efficient. However, if we define a sequence of problems for which $|\mathcal{H}_n| = 2^n$, then the sample complexity is still polynomial in n but the computational complexity of the exhaustive search algorithm grows exponentially with n (thus, rendered inefficient).

Surrogate Loss Functions

As mentioned, and as we will see in the next chapters, convex problems can be learned effficiently. However, in many cases, the natural loss function is not convex and, in particular, implementing the ERM rule is hard.

As an example, consider the problem of learning the hypothesis class of half-spaces with respect to the $0-1$ loss. That is,

$$\ell^{0-1}(\mathbf{w},(\mathbf{x},y)) = \mathbb{1}_{[y \neq \text{sign}(\langle \mathbf{w},\mathbf{x}\rangle)]} = \mathbb{1}_{[y\langle \mathbf{w},\mathbf{x}\rangle \leq 0]}.$$

This loss function is not convex with respect to $\mathbf{w}$ and indeed, when trying to minimize the empirical risk with respect to this loss function we might encounter local minima (see Exercise 1). Furthermore, as discussed in Chapter 8, solving the ERM problem with respect to the $0-1$ loss in the unrealizable case is known to be NP-hard.

To circumvent the hardness result, one popular approach is to upper bound the nonconvex loss function by a convex surrogate loss function. As its name indicates, the requirements from a convex surrogate loss are as follows:

1. It should be convex.
2. It should upper bound the original loss.

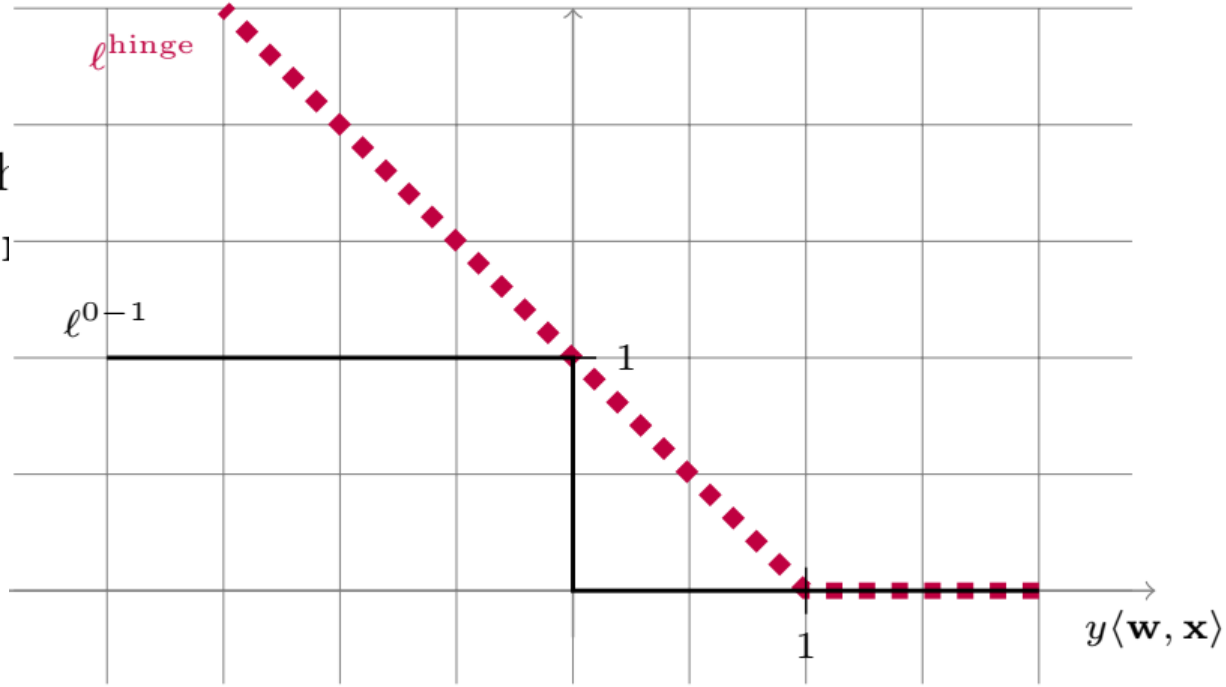
For example, in the context of learning halfspaces, we can define the so-called *hinge* loss as a convex surrogate for the $0-1$ loss, as follows:

$$\ell^{\text{hinge}}(\mathbf{w}, (\mathbf{x}, y)) \stackrel{\text{def}}{=} \max\{0, 1 - y\langle \mathbf{w}, \mathbf{x}\rangle\}.$$

Clearly, for all $\mathbf{w}$ and all $(\mathbf{x}, y)$, $\ell^{0-1}(\mathbf{w}, (\mathbf{x}, y)) \leq \ell^{\text{hinge}}(\mathbf{w}, (\mathbf{x}, y))$.

oss satisfies th

oss. An illustr



ce, the hinge
the zero-one
following.

- *Approximation error*: This is the term $\min_{\mathbf{w}\in\mathcal{H}} L_{\mathcal{D}}^{0-1}(\mathbf{w})$, which measures how well the hypothesis class performs on the distribution.

- *Estimation error*: This is the error that results from the fact that we only receive a training set and do not observe the distribution $\mathcal{D}$.

- *Optimization error*: This is the term $\left(\min_{\mathbf{w}\in\mathcal{H}} L_{\mathcal{D}}^{\text{hinge}}(\mathbf{w}) - \min_{\mathbf{w}\in\mathcal{H}} L_{\mathcal{D}}^{0-1}(\mathbf{w})\right)$ that measures the difference between the approximation error with respect to the surrogate loss and the approximation error with respect to the original loss. The optimization error is a result of our inability to minimize the training loss with respect to the original loss. The size of this error depends on the specific distribution of the data and on the specific surrogate loss we are using.

Rademacher Complexities

measures the rate of uniform convergence.

The Rademacher Complexity

DEFINITION 26.1 (ϵ-Representative Sample) A training set S is called ϵ-representative (w.r.t. domain Z, hypothesis class $\mathcal{H}$, loss function ℓ, and distribution $\mathcal{D}$) if

$$\sup_{h \in \mathcal{H}} |L_{\mathcal{D}}(h) - L_S(h)| \leq \epsilon.$$

We have shown that if S is an $\epsilon/2$ representative sample then the ERM rule is ϵ-consistent, namely, $L_{\mathcal{D}}(\mathrm{ERM}_{\mathcal{H}}(S)) \leq \min_{h \in \mathcal{H}} L_{\mathcal{D}}(h) + \epsilon$.

To simplify our notation, let us denote

$$\mathcal{F} \stackrel{\text{def}}{=} \ell \circ \mathcal{H} \stackrel{\text{def}}{=} \{z \mapsto \ell(h, z) : h \in \mathcal{H}\},$$

and given $f \in \mathcal{F}$, we define

$$L_{\mathcal{D}}(f) = \mathop{\mathbb{E}}_{z \sim \mathcal{D}}[f(z)] \quad , \quad L_S(f) = \frac{1}{m} \sum_{i=1}^{m} f(z_i).$$

We define the *representativeness* of S with respect to $\mathcal{F}$ as the largest gap between the true error of a function f and its empirical error, namely,

$$\mathrm{Rep}_{\mathcal{D}}(\mathcal{F}, S) \stackrel{\mathrm{def}}{=} \sup_{f \in \mathcal{F}} \big(L_{\mathcal{D}}(f) - L_S(f) \big). \tag{26.1}$$

Now, suppose we would like to estimate the representativeness of S using the sample S only. One simple idea is to split S into two disjoint sets, $S = S_1 \cup S_2$; refer to S_1 as a validation set and to S_2 as a training set. We can then estimate the representativeness of S by

$$\sup_{f \in \mathcal{F}} \big(L_{S_1}(f) - L_{S_2}(f) \big). \tag{26.2}$$

This can be written more compactly by defining $\boldsymbol{\sigma} = (\sigma_1, \ldots, \sigma_m) \in \{\pm 1\}^m$ to be a vector such that $S_1 = \{z_i : \sigma_i = 1\}$ and $S_2 = \{z_i : \sigma_i = -1\}$. Then, if we further assume that $|S_1| = |S_2|$ then Equation (26.2) can be rewritten as

$$\frac{2}{m} \sup_{f \in \mathcal{F}} \sum_{i=1}^{m} \sigma_i f(z_i). \tag{26.3}$$

The Rademacher complexity measure captures this idea by considering the expectation of the above with respect to a random choice of $\boldsymbol{\sigma}$. Formally, let $\mathcal{F} \circ S$ be the set of all possible evaluations a function $f \in \mathcal{F}$ can achieve on a sample S, namely,

$$\mathcal{F} \circ S = \{(f(z_1), \ldots, f(z_m)) : f \in \mathcal{F}\}.$$

Let the variables in $\boldsymbol{\sigma}$ be distributed i.i.d. according to $\mathbb{P}[\sigma_i = 1] = \mathbb{P}[\sigma_i = -1] = \frac{1}{2}$. Then, the Rademacher complexity of $\mathcal{F}$ with respect to S is defined as follows:

$$R(\mathcal{F} \circ S) \stackrel{\text{def}}{=} \frac{1}{m} \mathop{\mathbb{E}}_{\boldsymbol{\sigma} \sim \{\pm 1\}^m} \left[\sup_{f \in \mathcal{F}} \sum_{i=1}^{m} \sigma_i f(z_i) \right]. \tag{26.4}$$

More generally, given a set of vectors, $A \subset \mathbb{R}^m$, we define

$$R(A) \stackrel{\text{def}}{=} \frac{1}{m} \mathop{\mathbb{E}}_{\boldsymbol{\sigma}} \left[\sup_{\mathbf{a} \in A} \sum_{i=1}^{m} \sigma_i a_i \right]. \tag{26.5}$$

Thanks