CPSC 532D — STATISTICAL LEARNING THEORY

Danica J. Sutherland

University of British Columbia, Vancouver

Fall 2025

———

For syllabus-type material, see the course website.

This file contains all the notes for the course, and will be updated frequently as we add more sections. Each section is also available as its own standalone file from the course website; these will generally only be updated after class to correct mistakes (and I'll tell you either in class or on Piazza if any of these are important).

These notes are heavily indebted to a variety of sources, including [SSBD14; MRT18; Bach24; Wai19; Tel21; Ma22] as well as papers cited throughout.

Contents

For more, visit https://cs.ubc.ca/~dsuth/532D/25w1/.

1 Setup; ERM

This course is about: When should we expect machine learning algorithms to work? What kind of problems are possible for machine learning models to represent? What is possible to learn from data?

There are many complementary ways to study these questions. This course takes a primarily theoretical, mathematical approach, but tries to be guided by experimental results.

To phrase these questions more precisely, let's start by thinking about one of the simplest and best-understood machine learning models, linear regression. We'll use this as an example to set up our more general problem and think about how we can address those questions.

Is linear regression "really machine learning"? Obviously Legendre and Gauss didn't use that term in the early 1800s, but one reasonable definition of machine learning is "something you can publish a NeurIPS paper about," and as someone with multiple NeurIPS papers about linear regression, that makes the answer yes.

1.1 LINEAR REGRESSION

In the typical linear regression setting, we have m training inputs $x_i \in \mathbb{R}^d$ and corresponding outputs $y_i \in \mathbb{R}$. (For one of many possible examples, x_i might be a collection of summary features for a large geographic area, and y_i the number of hectares that burnt in forest fires in a given year.) We'll denote this as

$$S = \big((x_1, y_1), \ldots, (x_m, y_m)\big) \subset (\mathbb{R}^d \times \mathbb{R})^m. \tag{1.1}$$

While I used the term "training set" (because it's extremely well-established terminology), we actually want to potentially allow repeated data points. Occasionally, we might also care about the order (e.g. in online learning), so mathematically, we're going to use treat S as an m-tuple, not a set.

The usual assumption – which is definitely *not* always true, but is overwhelmingly the usual assumption in analyzing these kinds of things – is that these (x, y) pairs are independent samples from a distribution $\mathcal{D}$. We also write this $S \sim \mathcal{D}^m$, meaning that each of the m entries of S is an independent sample from $\mathcal{D}$.

$\mathcal{D}^m$ is called a product distribution, *and is a distribution over $(\mathcal{R}^d \times \mathcal{R})^m$ of m-length iid sequences.*

Our goal is to use S to find a weight vector w such that $w \cdot x \approx y$, for *fresh* (test) samples $(x, y) \sim \mathcal{D}$. Another way to phrase this is that we're looking for a *linear predictor*, i.e. a function $h_w : \mathbb{R}^d \to \mathbb{R}$ of the form $h_w(x) = w \cdot x$, such that $h_w(x) \approx y$.

You might also want an offset, $w \cdot x + b$, but we'll usually ignore that, since you can just add a constant 1 feature to x.

Statisticians, econometricians, etc. are often most concerned with getting the "right" w vector. For instance, we might assume $y = w^* \cdot x +$ Gaussian noise, or written another way $(y \mid x) \sim \mathcal{N}\big(w^* \cdot x, \sigma^2\big)$, and ask how well we recover w^*, e.g. by showing that $\|w - w^*\|$ goes to zero as $m \to \infty$.

Machine learners are generally more concerned with getting predictions right. We'll typically measure this with the *squared loss*,

$$\mathbb{E}_{(x,y)\sim\mathcal{D}}\Big[\tfrac{1}{2}(w \cdot x - y)^2\Big] = \mathbb{E}_{(x,y)\sim\mathcal{D}}\Big[\tfrac{1}{2}(h_w(x) - y)^2\Big]. \tag{1.2}$$

Often, recovering the "right" parameter vector, i.e. finding small $\|w - w^*\|$, implies that the predictive error is small (though not always, with nasty enough $\mathcal{D}$).

There are many situations, though, where you can have large $\|w - w^*\|$ but small predictive error. For instance, imagine that the first dimension of x is a person's height in metres, and the second dimension is their height in inches, both measured to arbitrary precision. Because one inch is exactly 2.54 centimetres, the w vectors $(1, 0, \dots)$ and $(0, 1/.0254, \dots)$ give *identical* predictions. So do any $(w_1, w_2, \dots)$ such that $w_1 + .0254 w_2 = 1$. There are infinitely many such vectors, and they can be arbitrarily far from w^*. Statisticians call (versions of) this problem *multicollinearity* and talk about how it causes issues with *identifiability*. For the most part, machine learners ignore this kind of problem; the different ws give identical predictions on $\mathcal{D}$. This is in some ways good, because these problems can be far worse with more complicated kinds of models, such as deep learning; it's bad in other ways, since although these examples have identical predictions on $\mathcal{D}$, they can have very different predictions on *other* distributions!

Anyway, we'd ideally like to solve the following optimization problem, which minimizes a quantity known variously as the *risk*, the *population loss*, or various other names:

$$\min_w \mathbb{E}_{(x,y) \sim \mathcal{D}} \left[\tfrac{1}{2}(w \cdot x - y)^2 \right] = \min_h \mathbb{E}_{(x,y) \sim \mathcal{D}} \left[\tfrac{1}{2}(h(x) - y)^2 \right]. \tag{1.3}$$

We don't have direct access to $\mathcal{D}$, though. Instead, we the most common choice is to estimate that expectation with an empirical average on the training set (simple Monte Carlo), and instead solve

$$\min_w \frac{1}{2m} \sum_{i=1}^m (w \cdot x_i - y_i)^2 = \min_h \frac{1}{2m} \sum_{i=1}^m (h(x_i) - y_i)^2. \tag{1.4}$$

The thing we minimize here is called the *training loss*, the *empirical risk*, or various other names.

As you've probably seen before, we can solve this minimization in closed form. Let $\mathbf{X} \in \mathbb{R}^{m \times d}$ be the matrix whose ith row is $x_i^\top$, and $\mathbf{y} \in \mathbb{R}^m$ the vector whose ith entry is y_i. Then the objective is $\frac{1}{2m} \left\| \mathbf{X}w - y \right\|^2$, which is a convex problem whose objective has gradient $\frac{1}{m} \mathbf{X}^\top (\mathbf{X}w - \mathbf{y})$. That's zero iff $\mathbf{X}^\top \mathbf{X}w = \mathbf{X}^\top \mathbf{y}$; if $\mathbf{X}^\top \mathbf{X}$ (which is $d \times d$ of rank at most $\min(n, d)$) is invertible, there's a unique solution $w = (\mathbf{X}^\top \mathbf{X})^{-1} \mathbf{X}^\top \mathbf{y} = \mathbf{X}^\dagger \mathbf{y}$, where $\mathbf{X}^\dagger$ is the [pseudoinverse](). Otherwise, there are infinitely many minimizers, which can be expressed as $\mathbf{X}^\dagger \mathbf{y} + z$ where z is any vector in the null space of $\mathbf{X}$. The training set predictions for any of these solutions is the same: $\mathbf{X}w = \mathbf{X}\mathbf{X}^\dagger \mathbf{y} + \mathbf{X}z = \mathbf{X}\mathbf{X}^\dagger \mathbf{y}$.

The metres-and-inches example above is one such case: since one column is 0.0254 times the other, $\mathbf{X}$ is not full-rank.

Having found this solution, we have lots of questions to ask: how well did solving (1.4) do as a proxy for the problem (1.3)? That is, given an estimate $\hat{w}$ or equivalently $\hat{h}$, what is the loss (1.2) of that predictor? Is it likely to be small, in different situations? Is it as small as any algorithm could hope to achieve? If we try to estimate (1.2) for that solution with a test set, how tight should we expect our estimate to be? If $\mathbf{X}^\top \mathbf{X}$ isn't invertible (always the case if $d > n$), which of the infinitely many minimizers should we pick?

For linear regression in particular, many of these questions can be tackled with basic linear algebra. You may have seen some of them in statistics courses; Chapter 3 of the textbook of Bach [Bach24] does some of these analyses framed in the language of learning theory. This is quite interesting, but we're not going to pursue that

approach in any more detail in this course, instead generalizing to other problems.

1.2 GENERAL PROBLEM SETUP

Our default, general learning problem is as follows:

- Instead of a data distribution $\mathcal{D}$ over $\mathbb{R}^d \times \mathbb{R}$, $\mathcal{D}$ is over some domain $\mathcal{Z}$. For *supervised learning*, $\mathcal{Z}$ is often a product space $\mathcal{Z} = \mathcal{X} \times \mathcal{Y}$ of (x, y) pairs, where x is an input object (e.g. an image) and y is a label (e.g. whether the image contains a dog). Occasionally, though, we want to learn over some other kind of space that doesn't have clear input-outputs.

- We have m independent, identically distributed samples $z_1, \ldots, z_m \sim \mathcal{D}$, collected in a training "set" $S = (z_1, \ldots, z_m) \sim \mathcal{D}^m$.

- We have a *hypothesis class* $\mathcal{H}$. In supervised learning, this is usually a set of predictors $h : \mathcal{X} \to \hat{\mathcal{Y}}$, a space of prediction functions.

  - In linear regression, $\mathcal{H}$ was the set of d-dimensional linear predictors, $\{x \mapsto w \cdot x : w \in \mathbb{R}^d\}$.

  - We could use bounded-norm linear predictors, $\{x \mapsto w \cdot x : \|w\| \le B\}$.

  - We could use decision trees of a certain depth, decision forests of a certain size, neural networks of a certain architecture, ...

  - Often, $\hat{\mathcal{Y}} = \mathcal{Y}$, but it might not; for example, it's common to have a problem with binary labels so that $\mathcal{Y} = \{0, 1\}$, but to make probabilistic predictions in $\hat{\mathcal{Y}} = [0, 1]$, or general confidence predictions in $\mathbb{R}$.

$x \mapsto 2x + 3$ means "the function which, given the argument x, returns $2x + 3$"; $\mathcal{H}$ is a set of functions. This is like `lambda x: 2*x+3` in Python.

- We have a *loss function* $\ell : \mathcal{H} \times \mathcal{Z} \to \mathbb{R}$. In supervised learning, this often takes the form $\ell(h, (x, y)) = l(h(x), y)$ for some $l : \hat{\mathcal{Y}} \times \mathcal{Y} \to \mathbb{R}$. Some common examples:

  - Squared loss: $l(\hat{y}, y) = \frac{1}{2}(\hat{y} - y)^2$. (Sometimes the $\frac{1}{2}$ isn't included.)

  - Zero-one loss: $l(\hat{y}, y) = \mathbb{1}(\hat{y} \ne y)$, usually used for $\mathcal{Y} = \hat{\mathcal{Y}}$ a discrete set of labels. This corresponds to one minus the *accuracy* of a predictor.

The function $\mathbb{1}$ returns one if its boolean argument is true, and zero if not.

  - Logistic loss: $l(\hat{y}, y) = \log(1 + \exp(-\hat{y}y))$ for $\hat{\mathcal{Y}} = \mathbb{R}$, $\mathcal{Y} = \{-1, 1\}$. This loss $\to 0$ if $\hat{y} \to y\infty$, i.e. if $y = 1$ and $\hat{y} \to \infty$, or $y = -1$ and $\hat{y} \to -\infty$: you're very confidently right. It's $\log 2$ if $\hat{y} = 0$, a totally ambiguous prediction. The loss goes $\to \infty$ if $\hat{y} \to (-y)\infty$: you're very confidently wrong.

  - Softmax cross-entropy loss, a multi-class generalization of logistic: here $\mathcal{Y} = [k] = \{1, 2, \ldots, k\}$, $\hat{\mathcal{Y}} = \mathbb{R}^k$ is the space of logits, and the loss is
  $$l(\hat{y}, y) = -\log \frac{\exp(\hat{y}_y)}{\sum\limits_{j=1}^{k} \exp(\hat{y}_j)} = -\hat{y}_y + \log \sum\limits_{j=1}^{k} \exp(\hat{y}_j).$$

- $L_{\mathcal{D}}(h) = \mathbb{E}_{z \sim \mathcal{D}} \, \ell(h, z) = \mathbb{E}_{(x,y) \sim \mathcal{D}} \, l(h(x), y)$ is called the *risk*, the *population loss*, the *true loss*, etc; this was (1.2) in logistic regression.

- $L_S(h) = \frac{1}{m} \sum\limits_{i=1}^{m} \ell(h, z_i) = \frac{1}{m} \sum\limits_{i=1}^{m} l(h(x_i), y_i)$ is the *empirical risk*, the *sample loss*, the *training loss* (if S is the training set), etc.

- A learning algorithm $\mathcal{A}$ is a function that takes in a sample S and returns a hypothesis in $\mathcal{H}$. Ideally, one with low risk.

Here's a recap of our notation, and a quick reference for how to translate notations across some relevant textbooks.

	These notes	[SSBD14]	[MRT18]	[Bach24]	[Zhang23]
Data distribution	$\mathcal{D}$	$\mathcal{D}$	$\mathcal{D}$	$\mathcal{D}$	$\mathcal{D}$
Number of samples	m	m	m	n	n
Sample set	S	S	S	$\mathcal{D}_n$	S_n
Hypothesis/parameter	$h \in \mathcal{H}$	$h \in \mathcal{H}$	$h \in \mathcal{H}$	$\theta \in \Theta$	$w \in \Omega$
Prediction on x	$h(x)$	$h(x)$	$h(x)$	$f_\theta(x)$	$f(w, x)$
Loss of hypothesis	$\ell(h, z)$	$\ell(h, z)$	–	–	$\phi(w, z)$
Loss of prediction	$l_y(\hat{y})$	–	$L(\hat{y}, y)$	$\ell(y, \hat{y})$	$L(\hat{y}, y)$
Empirical risk	$L_S(h)$	$L_S(h)$	$\hat{R}_S(h)$	$\hat{\mathcal{R}}(\theta)$	$\phi(w, \mathcal{D})$
Population risk	$L_{\mathcal{D}}(h)$	$L_{\mathcal{D}}(h)$	$R(h)$	$\mathcal{R}(\theta)$	$\phi(w, S_n)$

1.3 EMPIRICAL RISK MINIMIZATION

The most common general learning algorithm we'll think about, the general case of (1.4), is *empirical risk minimization*:

$$\text{ERM}(S) \in \underset{h \in \mathcal{H}}{\arg\min} \, L_S(h).$$

If $\mathcal{H}$ is infinite, there might be not be a minimizer. We usually won't worry about this explicitly, but basically everything we talk about could be generalized to approximate minimizers.

If there are ties, by default we think of the algorithm returning any arbitrary choice.

The returned hypothesis, ERM(S), which we will also often denote $\hat{h}_S$, is called an empirical risk minimizer ("an ERM").

For example, ERM with the squared loss and $\mathcal{H} = \{x \mapsto w \cdot x\}$ does indeed recover ordinary least squares:

$$\text{ERM}(S) \in \underset{h \in \{x \mapsto w \cdot x \,:\, w \in \mathbb{R}^d\}}{\arg\min} \, L_S(h)$$

$$= \underset{h \in \{x \mapsto w \cdot x \,:\, w \in \mathbb{R}^d\}}{\arg\min} \, \frac{1}{m} \sum_{i=1}^{m} \ell(h, z_i)$$

$$= \underset{h \in \{x \mapsto w \cdot x \,:\, w \in \mathbb{R}^d\}}{\arg\min} \, \frac{1}{m} \sum_{i=1}^{m} l_{y_i}(h(x_i))$$

$$= \left\{ x \mapsto x \cdot \hat{w} : \hat{w} \in \underset{w \in \mathbb{R}^d}{\arg\min} \, \frac{1}{m} \sum_{i=1}^{m} l(w \cdot x_i, y_i) \right\}$$

$$= \left\{ x \mapsto x \cdot \hat{w} : \hat{w} \in \underset{w \in \mathbb{R}^d}{\arg\min} \, \frac{1}{m} \sum_{i=1}^{m} \frac{1}{2}(w \cdot x_i - y_i)^2 \right\},$$

In our notation here, ERM(S) is returning a function (which makes these last couple of lines slightly tedious); we could equally well have let $\mathcal{H}$ be a set of parameter vectors and define a loss on parameters, $\ell(w, (x, y)) = \frac{1}{2}(x \cdot w - y)^2$.

and now $\hat{w}$ in the last line is probably what your intro stats class wrote down in the first place as the definition of linear regression.

We know that $L_S(\text{ERM}(S))$ is small by definition, but when can we expect $L_{\mathcal{D}}(\text{ERM}(S))$ to be small? The first big chunk of this course is about this question in particular. There are several ways to analyze this; the classic way is by making sure we choose an appropriate hypothesis class $\mathcal{H}$. If $\mathcal{H}$ is too simple, you'll never be able to learn the pattern you're looking for, but if it's too complicated, you'll *overfit* and pick one that seems good by chance, i.e. has good $L_S(\text{ERM}(S))$ but bad $L_{\mathcal{D}}(\text{ERM}(S))$.

Figure 1.1 illustrates this trade-off for polynomial regression. This is similar to

what you saw in your intro machine learning class; one of the things we'll do in this course is formalize this general intuition and prove theorems about it.
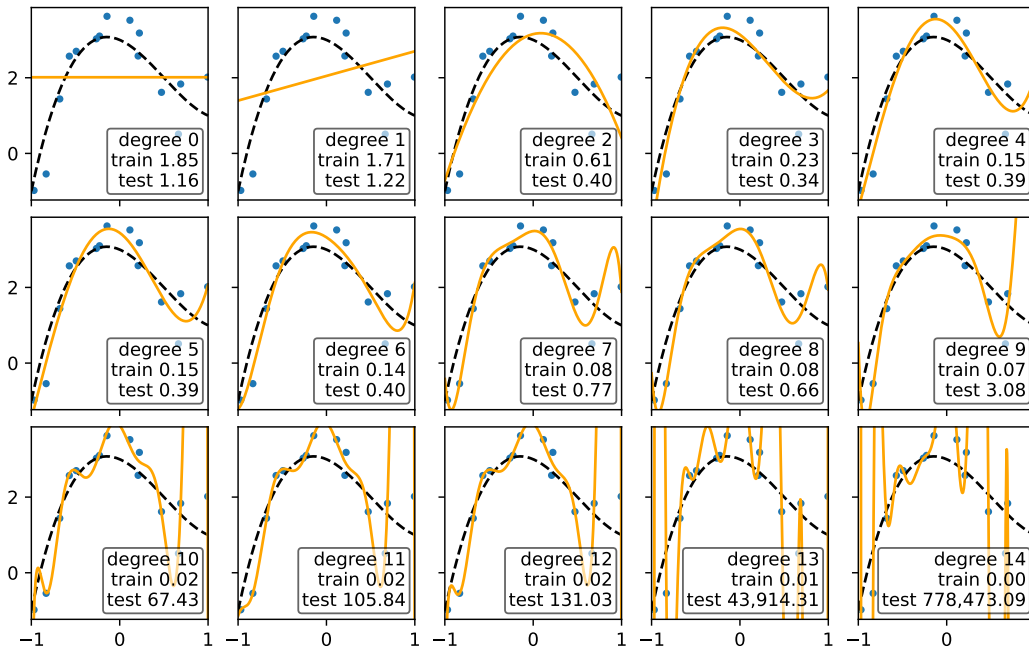
(a) Polynomial regression, $h(x) = w_0 + w_1 x + w_2 x^2 + \cdots + w_k x^k$, for increasing k, to data points shown in blue. ERM fits are in orange; dashed black lines show $\mathbb{E}[y \mid x]$, a cubic function. Text gives mean squared error for training and testing sets.
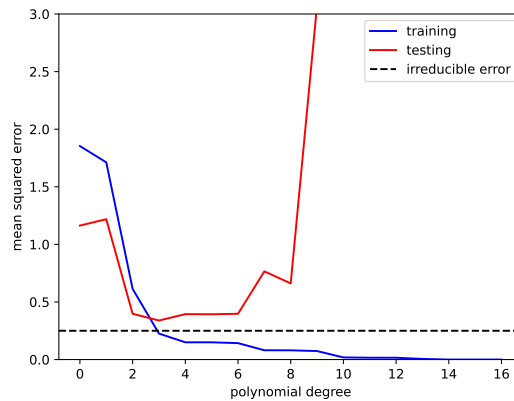
(b) Training and test errors from Figure 1.1a.

Figure 1.1: Underfitting to overfitting as $\mathcal{H}$ gets bigger.

1.4 ERROR DECOMPOSITIONS

Here's some standard terminology to know to formalize that intuition. For any estimator $\hat{h}_S \in \mathcal{H}$ (not necessarily just the ERM), we can write

$$\underbrace{L_{\mathcal{D}}(\hat{h}_S) - L_{bayes}}_{\text{excess error}} = \underbrace{L_{\mathcal{D}}(\hat{h}_S) - \inf_{h \in \mathcal{H}} L_{\mathcal{D}}(h)}_{\text{estimation error}} + \underbrace{\inf_{h \in \mathcal{H}} L_{\mathcal{D}}(h) - L_{bayes}}_{\text{approximation error}}.$$

The *excess error* is how much worse you are than the irreducible error L_{bayes}, also called the Bayes error or the error of the Bayes predictor (see A1 Q2 for more). No predictor, no matter its form, could do better than this: there's just inherent noise in the problem. The general $\ell(h, z)$ form unfortunately doesn't make this

7

easy to define (the domain is $\mathcal{H}$), but for $l_y(\hat{y})$ we can say something like $\mathrm{L}_{bayes} = \inf_{h:\mathcal{X}\to\hat{y}\text{ measurable }} l_y(h(x))$.

The *estimation error*, also called the *statistical error*, is the error that comes about from using your algorithm $\hat{h}_S$ rather than picking the best possible predictor in $\mathcal{H}$. As $m \to \infty$, this should (ideally) go to zero.

CPSC 340 used to use "approximation error" for the generalization gap, $\mathrm{L}_{\mathcal{D}}(h) - \mathrm{L}_S(h)$. This was a nonstandard use; it's been changed now in 340, and you should wipe it from your memory. :)

The *approximation error* doesn't (directly) depend on the number of samples you see: it's a function only of how well your hypothesis class $\mathcal{H}$ can do, regardless of estimation.

For example, in the polynomial regression case of Figure 1.1, using a $\mathcal{H}$ of linear functions results in some approximation error, but not much estimation error (because linear functions are easy to fit). Using a $\mathcal{H}$ of degree-fifteen polynomials has zero approximation error (it contains the Bayes predictor) but really high estimation error (too many parameters to fit).

Intuitively, as $\mathcal{H}$ gets "bigger," approximation error decreases but estimation error increases. Usually, approximation error is pretty problem-specific, but we'll see at least a few examples of formal analyses of it later in the course. First, we'll think about estimation error bounds.

1.4.1 *ERM estimation error*

Our usual basic way to prove when ERM generalizes well is to take the following decomposition, where we compare the loss of $\hat{h}_S$ to the loss of some arbitrary comparator hypothesis $h^* \in \mathcal{H}$. Note that we'll usually think of h^* as being roughly the best predictor in $\mathcal{H}$, but we don't require that, since it might not exist if $\mathcal{H}$ is infinite; instead we'll start by just comparing to any arbitrary predictor.

$$\mathrm{L}_{\mathcal{D}}(\hat{h}_S) - \mathrm{L}_{\mathcal{D}}(h^*) = \mathrm{L}_{\mathcal{D}}(\hat{h}_S) \underbrace{- \mathrm{L}_S(\hat{h}_S) + \mathrm{L}_S(\hat{h}_S)}_{0} \underbrace{- \mathrm{L}_S(h^*) + \mathrm{L}_S(h^*)}_{0} - \mathrm{L}_{\mathcal{D}}(h^*)$$

$$= \left(\mathrm{L}_{\mathcal{D}}(\hat{h}_S) - \mathrm{L}_S(\hat{h}_S)\right) + \underbrace{\mathrm{L}_S(\hat{h}_S) - \mathrm{L}_S(h^*)}_{\leq 0:\ \hat{h}_S \text{ minimizes } \mathrm{L}_S} + \left(\mathrm{L}_S(h^*) - \mathrm{L}_{\mathcal{D}}(h^*)\right)$$

$$\leq \underbrace{\mathrm{L}_{\mathcal{D}}(\hat{h}_S) - \mathrm{L}_S(\hat{h}_S)}_{\text{A: } \hat{h}_S\text{'s overfitting}} + \underbrace{\mathrm{L}_S(h^*) - \mathrm{L}_{\mathcal{D}}(h^*)}_{\text{B: } h^*\text{'s "underfitting"}} . \tag{1.5}$$

So, if we can bound A and B, then we can say that $\hat{h}_S$ isn't too much worse than h^*.

Now, as long as our bound on B doesn't depend on the particular choice of h^*, then this implies that

If you aren't familiar with inf, *it's like* min *but makes sense even if there isn't a minimizer (it's the largest lower bound). For example,* $\inf_{x\in\mathbb{R}:x>0} x = 0$ *even though 0 isn't in that set.*

$$\mathrm{L}_{\mathcal{D}}(\hat{h}_S) - \inf_{h\in\mathcal{H}} \mathrm{L}_{\mathcal{D}}(h) \leq \mathrm{A} + \mathrm{B}.$$

The next few weeks will be denoted to bounding A + B, how much worse $\hat{h}_S$ is than the best possible thing in $\mathcal{H}$.

Bibliography

[Bach24]   Francis Bach. *Learning Theory from First Principles*. Draft version. August 2024.

[Ma22]   Tengyu Ma. *Lecture Notes for Machine Learning Theory (CS229M/STATS214)*. June 2022.

[MRT18]   Mehryar Mohri, Afshin Rostamizadeh, and Ameet Talkwalkar. *Foundations of Machine Learning*. 2nd edition. MIT Press, 2018.

[SSBD14]   Shai Shalev-Shwartz and Shai Ben-David. *Understanding Machine Learning: From Theory to Algorithms*. Cambridge University Press, 2014.

[Tel21]   Matus Telgarsky. *Deep learning theory lecture notes*. October 2021.

[Wai19]   Martin Wainwright. *High-dimensional statistics: a non-asymptotic viewpoint*. Cambridge University Press, 2019.

[Zhang23] Tong Zhang. *Mathematical Analysis of Machine Learning Algorithms*. Pre-publication version. 2023.