# CS168: The Modern Algorithmic Toolbox
# Lecture #5: Generalization
# (Or, How Much Data Is Enough?)

Tim Roughgarden & Gregory Valiant*

April 15, 2019

## 1 Data Sets as Random Samples

When analyzing a data set, you can be in one of two different modes. In the first mode, you really care about understanding the data set at hand. For example, in sociology, often the main point of a research project is to understand deeply the peculiarities of a single data set (e.g., the friendship structure among the 34 members of a karate club [3]).

In the second mode, you care about the data set at hand only inasmuch as it allows you to learn something more general—to extrapolate accurately to additional data. For example, think about the results of a medical trial ("treatment X helped 37% of the subjects while a placebo helped 25%"). The implicit assumption (or hope) behind any such trial is that the results for the trial subjects are representative of what would happen for the entire population. In this and the next lecture, we'll be solidly in this second mode.

> **Today's viewpoint:** data points are random samples from some underlying population (i.e., distribution).

In the medical trial example, the distribution would be the uniform distribution over possible trial subjects (from which the actual subjects are assumed to be random samples).

For another example that you might have seen before, consider the problem of spam detection—given an email, label it as spam or not. You might have done the following project in a different class: given "labeled training data," meaning emails classified (by hand) as either spam or not, use an algorithm (a "learning" or "training" algorithm) to compute a "prediction function," whose job is to offer an opinion as to whether or not an arbitrary (perhaps never-before-seen) email is spam. Again, the reason you spend time with the data set of labeled emails is not because you want to understand them per se, but rather to learn how to extrapolate correctly to emails not in the data set.

In this course, you'll see plenty of lectures in each mode. We'll also return to the distributional viewpoint in future lectures on techniques for sampling and estimation. Last week (e.g., with $k$-d trees or dimensionality reduction), and the upcoming weeks on linear algebraic techniques, are more in the first mode.

## 2 Binary Classification

There are many different types of learning problems. For concreteness, we'll illustrate our points using *binary classification problems.*[1] Here are the ingredients of the model:

1. Data points correspond to points in $d$-dimensional Euclidean space $\mathbb{R}^d$. (For example, the vector of word frequencies in an email, for $d$ different words that you're keeping track of.)

2. There is a "ground truth" function $f : \mathbb{R}^d \to \{0, 1\}$ specifying the correct classification of every possible data point. (E.g., specifying for each possible email whether or not it is actually spam .)[2] *The function $f$ is initially unknown, and is what we want to learn.*

3. There is an unknown distribution $D$ on $\mathbb{R}^d$ (the origin of our training data, see below). For example, you can think of $D$ as the uniform distribution over a very large finite subset of $\mathbb{R}^d$ (e.g., all emails with at most $10,000$ characters that could conceivably be received).

Here's the problem solved by a learning algorithm:[3]

> **Input:** $n$ data points $\mathbf{x}_1, \ldots, \mathbf{x}_n \in \mathbb{R}^d$, with each $\mathbf{x}_i$ drawn independently and identically ("i.i.d.") from the distribution $D$, and the corresponding ground truth labels $f(\mathbf{x}_1), \ldots, f(\mathbf{x}_n) \in \{0, 1\}$. The $\mathbf{x}_i$'s (and their labels) constitute the *training data.* For example, the $\mathbf{x}_i$'s could be emails, with each email correctly labeled by a human as spam or not.[4]
>
> **Output:** The responsibility of the learning algorithm is to output a *prediction function $g : \mathbb{R}^d \to \{0, 1\}$*, its "best guess" as to what the ground truth $f$ is.
>
> **Success criterion:** the prediction function $g$ is identical to the ground truth function $f$. (Or later in lecture, $g$ is at least "close" to $f$.)

[1] The lessons learned in this lecture carry over to many other learning problems, such as linear and logistic regression.

[2] Again, one can consider variations, with $\{0, 1\}$ replaced by $[0, 1]$ (allowing "soft" classifications) or $\mathbb{R}$—the main points and results of this lecture remain the same.

[3] This type of learning problem is called "batch" or "offline" learning, because all of the training data is available up front in a batch.

[4] This type of problem, where the learning algorithm is given labeled examples, is called a *supervised* learning problem. Also important are *unsupervised* learning problems, where an algorithm is given unlabeled data and is responsible for identifying "interesting patterns." We'll talk more about unsupervised learning next week, when we discuss principal components analysis (PCA).

Figure 1 shows an example with $d = 2$, with $n$ data points labeled as "+" or "-." The learning algorithm needs to label the entire plane with "+"s and "-"s, ideally with 100% accuracy (with respect to the ground truth $f$). Figure 1 also shows a simple example of what a ground truth function $f$ might look like—a line (or hyperplane in higher dimensions), with all "+"s on one side and all "-"s on the other. (We'll return to such "linear classifiers" later in the lecture, see Section 6.)

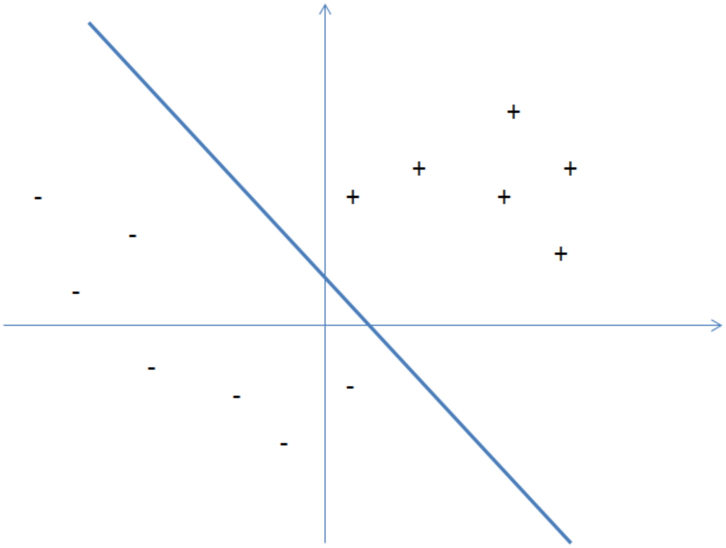

Figure 1: Labeled data, and a candidate prediction function.

Again, we emphasize that $f$ is initially unknown, with the learning algorithm acting as a detective who is trying to reconstruct $f$. The learning algorithm does receive some clues about $f$, specifically its evaluation at $n$ different data points (the $\mathbf{x}_i$'s). The prediction function $g$ is the algorithm's extrapolation of $f$ from these $n$ data points to all of the data points (included the never-before-seen ones). "Learning" means extrapolating correctly, in that $g$ should coincide with $f$.

Are there any learning algorithms that actually achieve this goal? It depends, with two parameters of the application being crucial.

*The amount of data* (i.e., $n$). The more data you have, the more you know about the ground truth function $f$, and the better your prospects for figuring it out.

*The number of possible functions that might be the ground truth.* The fewer functions that you're trying to distinguish between, the easier the learning problem.

# 3 Training Error and Generalization

How do we assess the "goodness" of a prediction function $g$? What we really care about is the *generalization error*, defined as the probability that $g$ disagrees with the ground truth $f$ on the label of a random sample (from the underlying distribution $D$):

$$\text{generalization error}(g) = \mathbf{Pr}_{\mathbf{x}\sim D}[g(\mathbf{x}) \neq f(\mathbf{x})]\,.$$

Our learning goal can be rephrased as identifying a function $g$ with 0% generalization error (or later in lecture, with very small generalization error).

We're not in a position to evaluate the generalization error of a prediction function, as we know neither the ground truth $f$ nor the distribution $D$. What we do know is $n$ sample points and their ground truth labels, and it's natural to use $g$'s performance on these as a proxy for $g$'s generalization error. This proxy is called the *training error* of $g$ (with respect to a sample $\mathbf{x}_1, \ldots, \mathbf{x}_n$ with ground truth labels $f(\mathbf{x}_1), \ldots, f(\mathbf{x}_n)$):

$$\text{training error}(g) = \frac{1}{n} \cdot [\text{number of } \mathbf{x}_i\text{'s with } g(\mathbf{x}_i) \neq f(\mathbf{x}_i)].$$

For any given prediction function $g$, its expected training error (over the random sample $\mathbf{x}_1, \ldots, \mathbf{x}_n$) is exactly its generalization error.[5] But is the training error of $g$ on a random sample very likely to be close to its generalization error, or might there be a big discrepancy? As a special case of this, is a function $g$ with 0% training error likely to also have 0% generalization error?

Such questions are usually phrased as: does $g$ *generalize*? The answer depends on the amount of training data $n$ (with more being better), and in some cases also on the learning algorithm used to come up with the prediction function $g$ (see Mini-Project #3). When a learning algorithm outputs a prediction function $g$ with generalization error much higher than its training error, then it is said to have *overfit* the training data. In this case, the prediction function learned is largely an artifact of the particular sample, and does not capture the more fundamental patterns in the data distribution.[6]

[5] We have $\mathbf{E}_{\mathbf{x}_1,\ldots,\mathbf{x}_n\sim D}[\text{training error}(g) \text{ on } \mathbf{x}_1, \ldots, \mathbf{x}_n] = \frac{1}{n}\sum_{i=1}^{n} \mathbf{Pr}_{\mathbf{x}_i\sim D}[g(\mathbf{x}_i) \neq f(\mathbf{x}_i)] =$ generalization error$(g)$, where in the first equation we're using the linearity of expectation.

[6] In practice, one checks for generalization/overfitting by dividing the available data set into two parts, the "training set" and the "test set." The learning algorithm is given the training set only, and the *test error* of the computed prediction function $\hat{f}$ is then defined as the fraction of test set data points that $\hat{f}$ labels incorrectly. If there is a significant difference between the training and test error of the output of the learning algorithm (generally with the latter much bigger than the former), then you've got a problem, and should consider gathering more data and/or using some of the techniques discussed in the next lecture to learn a better prediction function.

# 4 Analysis: The Well-Separated Finite Case

## 4.1 Assumptions

To develop our intuition, we first consider the learning problem with two additional assumptions:

(A1) *(Finite)* The ground truth function $f$ belongs to a known set $\{f_1, \ldots, f_h\}$ of $h$ different functions. That is, there are only $h$ different possibilities for what $f$ might be, and we know the options up front.

(A2) *(Well-separated)* No function (other than $f$) is extremely close to $f$, meaning that every candidate $f_j \neq f$ has generalization error at least $\epsilon$. (Here $\epsilon$ is a parameter in our assumption, which would be e.g. 1% or 10%.)

Later this lecture we will relax both of these assumptions.

## 4.2 Our First Learning Algorithm

Recall that our goal is to design a learning algorithm whose output $g$ is the same as the a priori unknown ground truth $f$ (with high probability over the sampled data points), i.e., a function $g$ with 0% generalization error. A simple but important point is that, if we ever identify a data point $\mathbf{x}$ such that $g(\mathbf{x}) \neq f(\mathbf{x})$, then we can rule out $g$ from future consideration (it can't possibly be the same as $f$). In particular, a necessary condition for $g$ to have 0% generalization error is that it has 0% training error (otherwise there is a training sample that proves that $g \neq f$). This leads us to the first version of our basic learning algorithm:

**The Basic Learning Algorithm**

Output an arbitrary function $g \in \{f_1, \ldots, f_h\}$ that has 0% training error.

How well does this algorithm work? Does its output $g$ generalize, in the sense of also having 0% generalization error? The answer depends on $n$. For example, if $n = 1$, there are presumably lot of choices of $f_j$ that are correct on this one data point, and all but one of them have non-zero generalization error. So our goal is to identify a sufficient condition on the data set size $n$ such that the prediction function output by the learning algorithm generalizes (i.e., is always correct even on not-yet-seen data points).

## 4.3 Getting Tricked by a Single Function

For the analysis, first consider a fixed function $f_j$ different from the true $f$. We're concerned about getting "tricked" by $f_j$, meaning that it winds up having 0% training error (despite having generalization error at least $\epsilon$, by (A2)). The probability of this bad event is

$$
\begin{aligned}
\mathbf{Pr}_{\mathbf{x}_1,\ldots,\mathbf{x}_n\sim D}[f_j(\mathbf{x}_i)=f(\mathbf{x}_i) \text{ for all } i=1,2,\ldots,n] &= \prod_{i=1}^{n}\mathbf{Pr}_{\mathbf{x}_i\sim D}[f_j(\mathbf{x}_i)=f(\mathbf{x}_i)] \\
&\le (1-\epsilon)^n \\
&\le e^{-\epsilon n},
\end{aligned}
$$

where the equation follows from our assumption that the $\mathbf{x}_i$'s are independent samples from $D$, the first inequality follows from assumption (A2), and the last inequality follows because $1+x\le e^x$ for all $x\in\mathbb{R}$ (see Figure 2, we're taking $x=-\epsilon$). As we hoped, this probability is decreasing (exponentially, in fact) with the number of samples $n$.

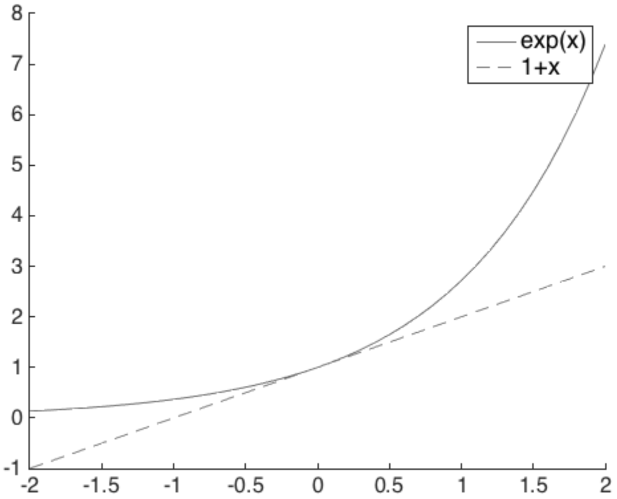

Figure 2: The inequality $1+x\le e^x$ holds for all real-valued $x$.

## 4.4 The Union Bound

The computation above was just for a fixed incorrect prediction function $f_j\ne f$, and we have $h-1$ such functions to worry about. To deal with this, recall the union bound, which should be familiar from CS109. The union bound states that for events $A_1,\ldots,A_h$,

$$\mathbf{Pr}[\text{at least one of the } A_i\text{'s occurs}] \le \sum_{i=1}^{h}\mathbf{Pr}[A_i]\,.$$

Importantly, the events are completely arbitrary, and do not need to be independent. The proof is a one-liner. In terms of Figure 3, the union bound just says that the area (i.e., probability mass) in the union is bounded above by the sum of the areas of the circles.

The bound is tight if the events are disjoint; otherwise the right-hand side is larger, due to double-counting the overlaps. In most applications, including the present one, the events $A_1, \ldots, A_h$ represent "bad events" that we're hoping don't happen; the union bound says that as long as each event occurs with low probability and there aren't too many events, then with high probability none of them occur.

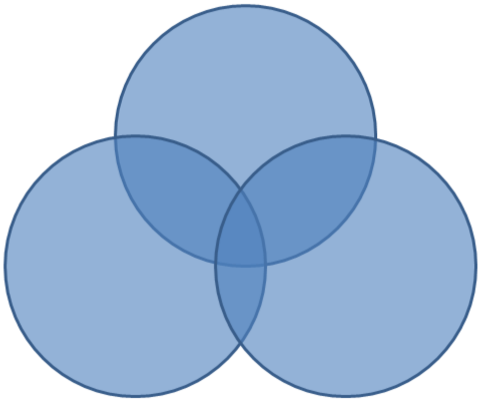

Figure 3: Union bound: area of the union is bounded by the sum of areas of the circles.

## 4.5 Completing the Analysis

Let $A_j$ denote the event that an incorrect function $f_j \neq f$ has 0% training error on $n$ samples, despite having generalization error at least $\epsilon$. From the derivation in Section 4.3, we know that $\mathbf{Pr}[A_j] \leq e^{-\epsilon n}$ for every $j$. Since there are only $h$ different values of $j$ to consider (by assumption (A1)), the union bound of the previous section implies that

$$\mathbf{Pr}_{\mathbf{x}_1,\ldots,\mathbf{x}_n \sim D}[\text{some } f_j \neq f \text{ has } 0\% \text{ training error}] \leq \sum_{j=1}^{h} e^{-\epsilon n} = h \cdot e^{-\epsilon n}.$$

Also, because our learning algorithm can only fail to output the ground truth $f$ when there is a function $f_j \neq f$ with 0% training error (otherwise the only remaining candidate is the correct answer $f$), we have

$$\mathbf{Pr}_{\mathbf{x}_1,\ldots,\mathbf{x}_n \sim D}[\text{output of learning algorithm is } f] \geq 1 - he^{-\epsilon n}.$$

That is, $he^{-\epsilon n}$ is an upper bound on the failure probability of our learning algorithm. This upper bound increases linearly with the number of possible functions (remember the learning problem is harder as you're trying to differentiate between more functions) but decreases exponentially with the size of training data set.

So suppose you want a failure probability of at most $\delta$ (say, $\delta = 1\%$).[7] How much data do you need? Setting $he^{-\epsilon n} = \delta$ and solving for $n$, we get the following sufficient condition for generalization.

**Theorem 4.1** *Suppose assumptions (A1) and (A2) hold, and assume that*

$$n \geq \frac{1}{\epsilon}\left(\ln h + \ln \frac{1}{\delta}\right). \tag{1}$$

*Then with probability at least $1 - \delta$ over the samples $\mathbf{x}_1, \ldots, \mathbf{x}_n \sim D$, the output of the learning algorithm is the ground truth function $f$.*

The *sample complexity* of a learning task is the minimum number of i.i.d. samples necessary to accomplish it. Thus Theorem 4.1 states that the right-hand side of (1) is an upper bound on the sample complexity of learning an unknown ground truth function with probability at least $1 - \delta$ (under assumptions (A1) and (A2)). Let's inspect this upper bound more closely. The not-so-good news is that the dependence on $\frac{1}{\epsilon}$ is linear. So to reduce $\epsilon$ from 10% to 1%, according to this bound you need 10 times as much data. The much better news is that the dependence on $h$ and $\frac{1}{\delta}$ is only logarithmic, so there's generally no problem with taking $\delta$ to be quite small (1% or even less), and large (even exponential-size) values of $h$ can be accommodated. For example, if we take $\epsilon = 5\%$, $\delta = 1\%$, and $h = 1000$, then our bound on the sample complexity is something like 230. If we reduce $\epsilon$ to 1% then the bound shoots up past 1000, but if we reduce $\delta$ to 0.1% then the bound remains under 300.

# 5 PAC Guarantees: The Finite Case

Now we extend our sample complexity bounds to hold under weaker assumptions. For starters, let's drop assumption (A2) entirely while keeping assumption (A1) (i.e., a finite number of possible $f$'s). The obstacle that then arises is functions $f_j$ with generalization error that is strictly positive but very close to 0. In this case, the learning algorithm is unlikely to ever see a data point to differentiate such an incorrect function $f_j$ from the ground truth $f$, and might get tricked into outputting $f_j$ rather than $f$. But if $f_j$ is almost the same as $f$ anyway, shouldn't that be good enough for our purposes?

More formally, keep the learning algorithm exactly the same as in the previous section. Theorem 4.1 then translates, without any change to the proof, to the following guarantee.

**Theorem 5.1** *Suppose assumption (A1) holds, and assume that*

$$n \geq \frac{1}{\epsilon}\left(\ln h + \ln \frac{1}{\delta}\right).$$

*Then with probability at least $1 - \delta$ over the samples $\mathbf{x}_1, \ldots, \mathbf{x}_n \sim D$, the output of the learning algorithm is a function $f_j$ with generalization error less than $\epsilon$.*

[7] Note that there's no hope of having failure probability 0. There's always some chance, however remote, that you get a totally uninformative sample (e.g., $n$ copies of the same point).

Several comments. First, note the differences between the statements in Theorems 4.1 and 5.1: the latter has one less assumption, but compromises by allowing the learning algorithm to output a function with small (but non-zero) generalization error. Note that the semantics of the parameter $\epsilon$ have changed: in Theorem 4.1, $\epsilon$ depended on the $f_j$'s (and $D$) and was outside anyone's control, while in Theorem 5.1, $\epsilon$ is a user-specified parameter (in the same sense as $\delta$), controlling the trade-off between the sample complexity and the error of the learned prediction function. The type of guarantee in Theorem 5.1 is often called a *PAC* guarantee, which stands for "probably approximately correct" [2]. (The "probably" refers to the failure probability $\delta$, and the "approximately" to the allowed generalization error $\epsilon$.) Finally, note that the proof of Theorem 4.1 also immediately implies Theorem 5.1: the analysis in Section 4 continues to apply to the functions $f_j$ with generalization error at least $\epsilon$, so with probability at least $1 - \delta$, the only hypotheses eligible for having 0% training error are those with generalization error less than $\epsilon$. Our learning algorithm then outputs an arbitrary such hypothesis.

# 6 PAC Guarantees: Linear Classifiers

## 6.1 The Need for Assumptions

We now consider relaxing the assumption that $f$ is one of a finite number of possibilities (assumption (A1)). It is important to note that this assumption cannot be dropped entirely. To see this, suppose the ground truth $f$ could be any function whatsoever from $\mathbb{R}^d$ to $\{0, 1\}$. A learning algorithm gets no information about $f$ other than its value on $n$ sample points. If $f$ is unrestricted, then the labels of unseen points could be *literally anything*, and the learning algorithm cannot extrapolate at all from the training data to any other data points. In effect, the learning algorithm outputs a function that has memorized the training data (0% training error) and doesn't generalize at all (large generalization error, say 50%). This is an extreme case of overfitting.

## 6.2 Linear Classifiers

So to move forward, we insist that the ground truth function $f$ is "structured," meaning it takes on a particular form. In this lecture, we'll focus on *linear classifiers*.

**Definition 6.1 (Linear Classifier)** A *linear classifier in* $\mathbb{R}^d$ is specified by a $d$-vector $\mathbf{a} = (a_1, \ldots, a_d) \in \mathbb{R}^d$ of real coefficients, and is defined as the function $f_{\mathbf{a}} : \mathbb{R}^d \rightarrow \{0, 1\}$ with

$$f_{\mathbf{a}}((x_1, \ldots, x_d)) = \begin{cases} 1 & \text{if } \sum_{i=1}^d a_i x_i \geq 0 \\ 0 & \text{if } \sum_{i=1}^d a_i x_i < 0. \end{cases} \tag{2}$$

Note that a linear classifier in $d$ dimensions has $d$ degrees of freedom (the $a_i$'s). Geometrically, a linear classifier corresponds to a hyperplane through the origin (with normal vector $\mathbf{a}$), with all points on the same side of the hyperplane as the normal vector labeled "1" and all points on the other side labeled "0". Linear classifiers may feel like a toy example, but they're not—for example, "support vector machines (SVMs)" are basically the same as linear classifiers.

## 6.3 From the Curse of Dimensionality to Generalization

There are an infinite number of linear classifiers (even when $d = 1$), so Theorem 5.1 does not directly apply to the problem of learning a good linear classifier from labeled training data. However, recall the discussion last week (Lecture #4) about the "curse of dimensionality" and kissing numbers of spheres. We saw that the number of "distinct directions" in $\mathbb{R}^d$ grows exponentially (but not more) with $d$.[8] Last lecture, we viewed this as a negative result, which explained why many problems we care about (like computing nearest neighbors) become difficult in high dimensions. Today, we're actually going to use this exponential dependence for a *positive* result. Recall that our sample complexity bound in (1) for the finite case depended only *logarithmically* on the number of relevant functions $f_j$. So if all linear classifiers can be well approximated by a finite set of linear classifiers with size "only" exponential in $d$, then we might hope that the sample complexity of learning a good classifier is only linear in $d$. And this is in fact the case.[9]

**Theorem 6.2** *Suppose $f$ is a linear classifier of the form in* (2)*, and assume that*

$$n \geq \frac{c}{\epsilon}\left(d + \ln\frac{1}{\delta}\right),$$

*where $c$ is a sufficiently large constant. Then with probability at least $1 - \delta$ over the samples $\mathbf{x}_1, \ldots, \mathbf{x}_n \sim D$, the output of the learning algorithm is a linear classifier with generalization error less than $\epsilon$.*

The algorithm in Theorem 6.2 is the same as before—just output an arbitrary linear classifier that has 0% training error. Current proofs of Theorem 6.2 require the constant $c$ to be larger than one would like. In practice, a reasonable guideline is to think of $c$ as 1.

## 6.4 A Rule of Thumb

Theorem 6.2 leads to a surprisingly robust rule of thumb, which will serve you well in your future endeavors.

[8] For example, if you take $\alpha^d$ random points $R$ on the unit sphere in $\mathbb{R}^d$, where $\alpha$ is a sufficiently large constant, then with high probability, *every* unit vector will be in roughly the same direction as some point of $R$ (meaning the angle between them is less than 1°). This fact can be proved using the formulas for the volume and surface area of a sphere in $d$ dimensions—every random point "covers" a small but non-trivial set of directions around it, and eventually the whole unit sphere is covered.

[9] The proof sketch in the previous footnote leads to a slightly weaker bound that depends on $d \log \frac{1}{\epsilon}$ rather than $d$. A more clever argument gets rid of the $\log \frac{1}{\epsilon}$ factor, see e.g. [1].

> **Upshot:** to guarantee generalization, make sure that your training data set size $n$ is at least linear in the number $d$ of free parameters in the function that you're trying to learn.

Mini-Project #3 will drive home this guideline—you'll observe empirically that learned prediction functions tend to generalize when $n \gg d$ but not when $n \ll d$.

Theorem 6.2 makes the rule of thumb above precise for the problem of learning a linear classifier. For most other types of prediction functions that you might want to learn (e.g., in linear or logistic regression, or even with neural nets) there is a sensible notion of "number of parameters," and the guideline above is usually pretty accurate.

### 6.5 FAQ

Three questions you might have at this point are:

1. How do you actually implement the basic learning algorithm?
2. What if, contrary to our standing assumption, *no* function under consideration has 0% training error?
3. What should you do if $n \ll d$?

The next two sections answer the first two questions; next lecture is devoted to answering the third.

## 7 Computational Considerations

How do we actually implement the basic learning algorithm? That is, given a bunch of correctly labeled data points, how do we compute a candidate function $\hat{f}$ with 0% training error? If there are only a finite number of candidates (as in Sections 4–5), then if nothing else we can resort to exhaustive search. (For some types of functions, there will also be faster algorithms.) But we can't very well try each of the infinitely many linear classifiers, can we?

More formally, the relevant algorithmic problem for learning a good linear classifier is: given a number of "+"s and "-"s in $\mathbb{R}^d$, compute a hyperplane such that all of the "+"s are on one side and all of the "-"s are on the other (see Figure 4).[10]

One way to solve the problem is via linear programming, which we'll discuss in Lecture #18. With this approach you can even maximize the "margin," meaning the smallest Euclidean distance between a training point and the hyperplane. (This requires convex programming, which is still tractable and again will be discussed in Lecture #18.) This is one of the key ideas behind support vector machines (SVMs). Alternatively, one can apply iterative methods (like stochastic gradient descent, the perceptron algorithm, etc.).

[10] We're glossing over the distinction of whether or not the hyperplane has to go through the origin. It really doesn't matter—hyperplanes in $d$ dimensions can be simulated by hyperplanes through the origin in $d + 1$ dimensions (by adding a dummy coordinate with value "1" for every data point). More on this next lecture.

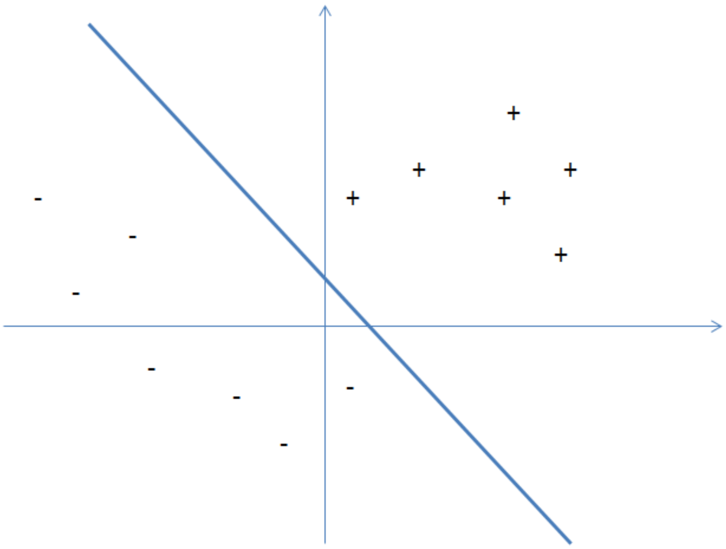

Figure 4: We want to find a hyperplane that separates the positive points (plus signs) from the negative points (minus signs).

# 8 Non-Zero Training Error and the ERM Algorithm

What if no function $\hat{f}$ under consideration has 0% training error? For example, what if you restrict yourself to learning a linear classifier, but the ground truth is not actually a linear function?

## 8.1 Generalization Guarantees for the ERM Algorithm

One solution is to extend the learning algorithm in the obvious way, to output the best of the available functions.[11]

**Empirical Risk Minimization (ERM)**

Output the function $\hat{f}$ (from the set of functions under consideration) that has the smallest training error, breaking ties arbitrarily.

This algorithm is often called the *ERM algorithm*, for "empirical risk minimization." (Sounds fancy, but it's really just the one-line algorithm above.)

[11] A second solution (compatible also with the first) is to enrich your class of functions so that there *is* a function with 0% training error (or at least smaller than before). One systematic way of doing this, discussed in detail next lecture, is to use linear classifiers in a higher-dimensional space (which can correspond to non-linear classifiers in the original space). There are various ways of adding new dimensions, such as appending to each data point $(x_1, \ldots, x_d)$ new coordinates containing quadratic terms in the original coordinates $(x_1^2, \ldots, x_d^2, x_1x_2, x_1x_3, \ldots)$.

The ERM algorithm has a PAC guarantee analogous to that of the basic learning algorithm, although with somewhat larger sample complexity. We state the guarantee for linear classifiers; the same result holds for arbitrary finite sets of functions (as before with the $d$ replaced by $\ln h$, where $h$ is the number of candidate functions). The key behind the ERM's generalization guarantee is the following theorem, which states that, with high probability after sufficiently many samples, the training error of *every* linear classifier is very close to its actual generalization error.

**Theorem 8.1 (Uniform Convergence)** *Assume that*

$$n \geq \frac{c}{\epsilon^2}\left(d + \ln\frac{1}{\delta}\right), \tag{3}$$

*where $c$ is a sufficiently large constant. Then with probability at least $1-\delta$ over the samples $\mathbf{x}_1, \ldots, \mathbf{x}_n \sim D$, for every linear classifier $\hat{f}$,*

$$\textit{generalization error of } \hat{f} \in \textit{training error of } \hat{f} \pm \epsilon. \tag{4}$$

Note that Theorem 8.1 does not assume that the ground truth $f$ is a linear function—it could be anything. Of course, if $f$ looks nothing like any linear classifier, then all linear classifiers will have large generalization error and the guarantee in Theorem 8.1 is not very relevant.[12]

A special case of Theorem 8.1 is: if $\hat{f}$ has 0% training error, then it has generalization error at most $\epsilon$. So Theorem 8.1 generalizes Theorem 6.2, with the caveat that its sample complexity is larger by a $\frac{1}{\epsilon}$ factor.[13]

Theorem 8.1 is a sufficient condition for generalization, in the following sense.

**Corollary 8.2 (Guarantee for the ERM Algorithm)** *In the setting of Theorem 8.1, let $\tau \in [0,1]$ denote the minimum generalization error of any linear classifier. Assume that*

$$n \geq \frac{c}{\epsilon^2}\left(d + \ln\frac{1}{\delta}\right),$$

[12] This highlights two different types of errors in learning. The first type of error is *approximation error*, where discrepancies between the ground truth and the set of allowed prediction functions cause all available functions to have large generalization error. No amount of algorithmic cleverness or data can reduce approximation error; the only solution is to enrich the set of allowable prediction functions (see also the previous footnote and next lecture). The second type of error, which we're more focused on here, is *estimation error*. The concern is that we choose a prediction function with training error much less than its generalization error, due to an overly small or unlucky sample.

These two types of errors connect to the "bias-variance" trade-off that you might hear about in a statistics course; enlarging the set of possible prediction functions tends to decrease the approximation error (reducing "bias") but increase estimation error (enlarging the "variance").

[13] The guarantee in (4) is sometimes called a "uniform convergence" result. "Convergence" refers to the fact that the training error of every classifier $\hat{f}$ approaches the generalization error (with high probability), while "uniform" refers to the fact that a single finite set of samples (with size as in (3)) suffices to give the guarantee in (4) simultaneously for all infinitely many linear classifiers.

*where $c$ is a sufficiently large constant. Then with probability at least $1-\delta$ over the samples $\mathbf{x}_1, \ldots, \mathbf{x}_n \sim D$, the output $\hat{f}$ of the ERM algorithm satisfies*

$$\textit{generalization error of } \hat{f} \le \tau + 2\epsilon.$$

Thus with high probability, the ERM algorithm learns a linear classifier that is almost as accurate as the best linear classifier.

*Proof of Corollary 8.2:* Let $f^*$ denote the most accurate linear classifier, with generalization error $\tau$. Let $\hat{f}$ denote the linear classifier returned by the ERM algorithm. Theorem 8.1 implies that, with high probability,

$$\text{training error of } f^* \le \tau + \epsilon$$

and also

$$\text{training error of } \hat{f} \ge \text{generalization error of } \hat{f} - \epsilon.$$

Since the ERM algorithm picked $\hat{f}$ over $f^*$, we must have

$$\text{training error of } \hat{f} \le \text{training error of } f^*.$$

Chaining together the three inequalities proves the corollary. ■

## 8.2 Sample Complexity Considerations

There are two things that get worse when passing from the case where some function has 0% training error (called the *realizable case*) and the general case. First, the sample complexity bound in (3) is worse than that in (1) by a factor of $\frac{1}{\epsilon}$. This gap is necessary, at least in theory. One way to develop intuition about the difference between the two scenarios is to think about two different experiments that involve repeatedly flipping a coin with unknown bias $p$:

1. Suppose I promise you that the bias is either a 0% chance of heads, or a 1% chance. If any of your coin flips come up heads, then you know you're in the second case. What if you only observe tails, say $k$ in a row? What are the chances that the bias actually is 1%? If you observe 300 heads in a row, then you can be pretty confident that the bias is actually 0 (there's only a roughly 5% chance of getting 300 consecutive tails when the bias is 1%). This thought experiment corresponds to the original realizable setting, and a $\frac{1}{\epsilon}$ dependence in the sample complexity (where $\epsilon$ corresponds to 1%).

2. Now suppose you want to estimate the unknown bias $p$, up ±1%. Suppose you toss the coin 300 times, and see exactly 150 heads and 150 tails. Can you confidently conclude that $p \in 50\% \pm 1\%$? No — if $p = 52\%$, say, there's a perfectly reasonable chance of seeing an equal split of heads and tails across 300 trials.[14] About all you can say is that $p$ is very likely to be between 40% and 60%. You would need more like 30,000 coin flips to be very confident about $p$ up to ±1%. This thought experiment corresponds to our general learning problem, and its sample complexity dependence on $\frac{1}{\epsilon^2}$.

[14] This can be verified by computing some binomial coefficients.

Another way to think about the difference is the following. In the first experiment, there is "one-sided error;" you might mistake a 1%-bias coin for a 0%-bias coin (from a long run of tails), but never vice versa (once you see a heads you know with certainty that you're correct). The second problem, of estimating the bias, has "two-sided error"—no matter what bias you guess, there's a chance that your guess is too high, and also that it's too low. Needing to hedge between two different types of error causes the sample complexity to jump by $\frac{1}{\epsilon}$.

Despite all this, the dependence on $d$ in (3) remains linear, and our key take-away from before remains valid:

- **Upshot:** to guarantee generalization, make sure your training data set size $n$ is at least linear in the number $d$ of free parameters in the function that you're trying to learn.

Empirically, one usually sees pretty good generalization when $n$ is equal to or a small multiple of $d$; see Mini-Project #3.

### 8.3 Computational Considerations

The second complication in the general case, relative to the realizable case, concerns the complexity of implementing the ERM algorithm. In Section 7, we noted that there are various computationally efficient methods for checking if there is a linear classifier with 0% training error. The more general problem that the ERM algorithm needs to solve, of identifying the linear classifier with the minimum (possibly non-zero) training error, is unfortunately an $NP$-hard problem. Because of this computational intractability (in both theory and practice), one has to resort to heuristics. Many of these heuristics are based on the methods already mentioned for the realizable case (linear programming, stochastic gradient descent, etc.).

## 9 Recap of Key Points

Here are some of the high-order bits to take away from this lecture:

1. Data can be fruitfully thought of as samples from some underlying population or distribution.

2. *Learning* means successfully extrapolating from labeled training data to as-yet-unseen data points (i.e., learning a prediction function with low generalization error).

3. *Generalization* means that the observed training error of a prediction function is an accurate proxy for the quantity you really care about, its generalization error. With generalization guarantees, learning reduces to the problem of computing a prediction function with small training error.

4. A sufficient condition for generalization is for the data set size to exceed the number of free parameters in the prediction function being learned. This rule of thumb is surprisingly robust across learning problems (e.g., linear classifiers, linear and logistic regression, and even in many cases for neural nets).

5. In practice, you might be stuck with $n$ much smaller than $d$. Next lecture discusses additional techniques that are useful for this case, including regularization.

# References

[1] Martin Anthony and Peter L. Bartlett. *Neural Network Learning: Theoretical Foundations*. Cambridge University Press, NY, NY, USA, 1999.

[2] Leslie G Valiant. A theory of the learnable. *Communications of the ACM*, 27(11):1134–1142, 1984.

[3] W. Zachary. An information flow model for conflict and fission in small groups. *Journal of Anthropological Research*, 33(4):452–473, 1977.