Generalization

We've talked a lot about algorithms for finding ML models: linear regression, perceptron, gradient descent, minimum description length. Most of these algorithms work by trying to find a model that performs well on a set of training data (maybe in combination with other criteria).

But, we don't care how well our model performs on the training data. Instead, we care how well it performs when we deploy it. We hope that a model that performs well on training data will also do well in deployment — this property is called *generalization*.

Unfortunately, there are a *lot* of things that can go wrong and hurt generalization. Some examples:

- Concept shift: the thing we told the model to learn turns out to be not quite the right concept. E.g., the person who needs the ML model didn't communicate adequately with the person who labeled the training data.
- Distribution shift: even if the concept stays the same, the training data is not representative of the examples we see during deployment.
- Adversaries: someone wants to fool our learned model, and can influence either the training data or what the model can see during deployment.
- Missing information: we lose access to some information during deployment that we depended on during training. For example, we could measure some features of an example accurately during training, and noisily or sporadically during deployment.
- Goodhart's Law: "When a measure becomes a target, it ceases to be a good measure." If our learned model is used for anything important, people will try to present their examples in the best possible light. This is like giving a used car a new coat of paint: it's still rusty underneath, but the model doesn't see it. (This law is related to adversaries and to distribution shift, but it is enough different to be worth mentioning on its own.)
- Overfitting: even if none of the above problems happen, models rarely perform as well during deployment as they do during training.

The first five problems stem from how we use ML in the context of a larger *socio-technical system*: a combination of software, hardware, and humans that work together to try to accomplish some goal. As such, they are extremely important, but

beyond the scope of what we can cover here.

> *It's worth pointing out that system-level problems can do a whole lot more than just hurt generalization; for example, they can be at the root of unfairness, bias, and lack of transparency, as well as a host of other unintended consequences.*

By contrast, the last problem (overfitting) is a purely ML problem. In these notes, we'll look why overfitting happens, as well as how to measure it and how to compensate for it.

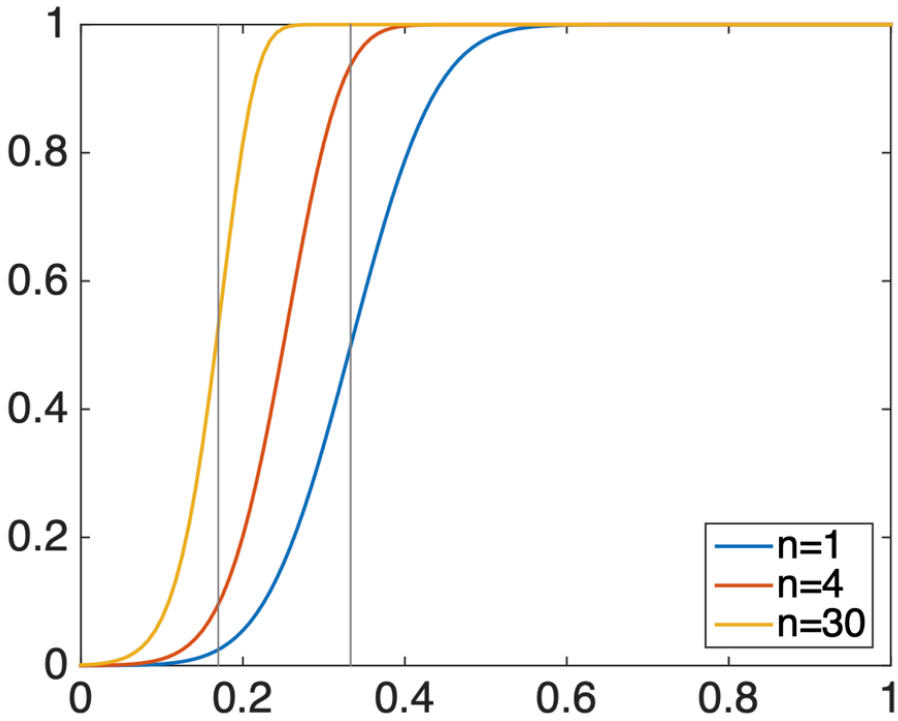
Lucky or good?

If a sports team wins a tournament, are they lucky or good? The answer is likely a little of both: they probably couldn't win if they were the worst team, but on any given day the second-best team might have a decent chance of beating the best team. Since there are often a fair number of good teams, and only one best team, it can even happen that we're fairly sure the best team will *not* win.

If a model performs best on our training data, is it lucky or good? Again, the answer is likely a little of both: since we selected our training data randomly, a model's measured performance on the training data can be higher or lower than its true performance. So, if we pick the model with the highest measured performance, we're likely to get one that was at least a bit lucky.

This phenomenon is called *selection bias*: if we use some criterion to select a model, then even if the criterion was an unbiased measurement to start off with, it will be biased for the model we pick. The amount of selection bias — the difference between the selected model's true performance and its measured performance — is called the *generalization gap*.

Here is selection bias in action:



In this plot, we look at the cumulative density function of the best of n samples from a fixed distribution. We can think of each sample as representing a performance measurement on our training data. If we measure $n = 1$ model, we'll get an unbiased but noisy estimate of its performance. If we measure more than one model (here we show $n = 4$ and $n = 30$), the distribution of the best model's performance will shift to the right — even though we set up this plot so that all models actually do equally well on our data.

> *In this plot, the performance is the model's error rate on a training set of 32 examples. All models have true error rate of $\frac{1}{3}$, but since our training set is not that big, the standard deviation of the measured error is about 0.0833.*

The amount of selection bias that we see depends on two things: how much can luck influence our performance measurement, and how many models are we picking from? In our example, with 4 models, we have almost a 95% chance of thinking our error rate is lower than it is. With 30 models, the median measured performance is around the 2.5% quantile of the unbiased distribution.

> *More precisely, instead of the total number of models, the amount of selection bias depends on how many halfway-reasonable models there are. We can get a lot of selection bias from a moderate number of pretty-good models, or a much larger number of OK models, or a really huge number of mediocre models.*

Overfitting

Overfitting is what happens when we get selection bias while trying to pick the best model. Overfitting actually covers two related problems:

- First, we might pick a bad model that just happens to get lucky. This tends to happen when there are a lot of models to choose from, and when performance measurements are fairly noisy.
- Second, we might pick a pretty good model, but have a too-rosy view of its performance.

We don't really want either of these problems, but the first problem is usually worse for us if it happens. Fortunately, overfitting has to be fairly bad for the first problem to kick in. So, typically, successful ML systems are able to mostly avoid the first problem, but still have to deal at least somewhat with the second problem.

For that reason, it's interesting to try to get an idea how strongly the second problem affects us — that is, how much we're overestimating our performance. There are two main strategies we can use for this purpose: *generalization bounds* and *holdout data*.

Generalization bounds

In some situations, we can analytically calculate a bound on our likely generalization gap. The array of available methods for this task could fill several courses on their own, but one of the simplest and most useful is the bound for choosing among finitely many classifiers. This bound is representative of the sort of results we can prove in other situations as well, so we will describe it in a bit of detail.

Suppose we have a supervised classification problem: we want to learn from data $(x_1, y_1), \ldots, (x_n, y_n)$ to predict $y_i \in \{-1, 1\}$. And, suppose we have a finite set $\{f_1, f_2, \ldots, f_m\}$ of classifiers.

We use our training data to measure the error rate of each classifier: define ϵ_{ij} to be 0 if $y_i = f_j(x_i)$, that is, if classifier j gets example i right. And define ϵ_{ij} to be 2 if $y_i \neq f_j(x_i)$, that is, if classifier j gets example i wrong. The observed error rate of classifier j is the average of ϵ_{ij} for $i = 1 \ldots n$, $\epsilon_j = \frac{1}{n} \sum_i \epsilon_{ij}$. Write $\bar{\epsilon}_j$ for the *true* error rate of f_j, that is, the expected value of ϵ_j. We'd like to pick the classifier f_j that minimizes $\bar{\epsilon}_j$, but the best we can do is to pick the one that minimizes ϵ_j.

Because of this difference, we expect selection bias. The more training data we have (the larger n is), the more accurately we can measure the error of each classifier f_j;

this mitigates selection bias. But the more classifiers we pick from (the larger m is), the worse our selection bias will get.

It turns out that we can prove the following bound. For all j and for any $\delta \in (0, 1)$, the following holds with probability at least δ:

$$\bar{\epsilon}_j \leq \epsilon_j + \sqrt{\frac{\ln m + \ln \frac{1}{\delta}}{2n}}$$

That is, the true error could be larger than the observed error. But with high probability the excess error is $O(\sqrt{\frac{1}{n} \ln m})$: it grows only logarithmically with the number of models we consider, and decreases as the inverse square root of the number of training examples. Importantly, this bound holds for *all* of our classifiers, including the one we picked by minimizing training error. So, if we have a lot of training examples and not hugely many models, our selection bias won't be too bad.

> *To prove the above bound, we can use the Chernoff inequality to bound the difference between each ϵ_j and its mean. Then we can use a union bound to get a result that holds for all j simultaneously.*
>
> *One of the most useful extensions of the above result is for the case of an infinite space of classifiers indexed by a vector of parameters or weights. We can typically **cover** such a space with a finite set, in the sense that each of our infinitely many original classifiers is close to at least one of the finitely many chosen classifiers. In this case, we can almost directly apply the above result, and the generalization gap will be bounded by the log of the size of our cover. The latter number can be related to the **dimension** of our set of classifiers, which often approximately coincides with the dimension of our parameter vector.*

Holdout data

The other way to estimate the generalization gap is with holdout or validation data. We'll show some slides about this class of approaches in lecture.

Within this class of approaches, there are at least three important sub-classes:

- validation
- cross-validation
- bootstrap