Generalization in Machine Learning

Sam, Arthur, and Joe 2020 Summer

The Cake Problem

A tempting pattern

We'll start out with a fun puzzle that's not directly related to machine learning but whose math will later help us. Imagine n people are seated around a large disk-shaped cake. Each pair of people will use a two-person handsaw to cut the cake. For example, the picture below shows a group of 4 people; there are 6 pairs of people, so the group makes 6 cuts total. With 4 people, the cake ends up in 8 pieces. We wonder: how many pieces arise when more people sit around the cake?

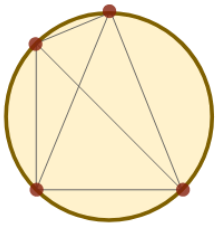

Figure 1: *With $n = 4$ people, we make $\binom{n}{2} = 6$ cuts, which gives 8 pieces in total (4 outside and 4 inside).*

Here are some drawings for possibilities ranging from $n = 1$ people to $n = 6$ people. Try counting the pieces and guessing a pattern! Can you find a formula for how many pieces arise when there are n people?

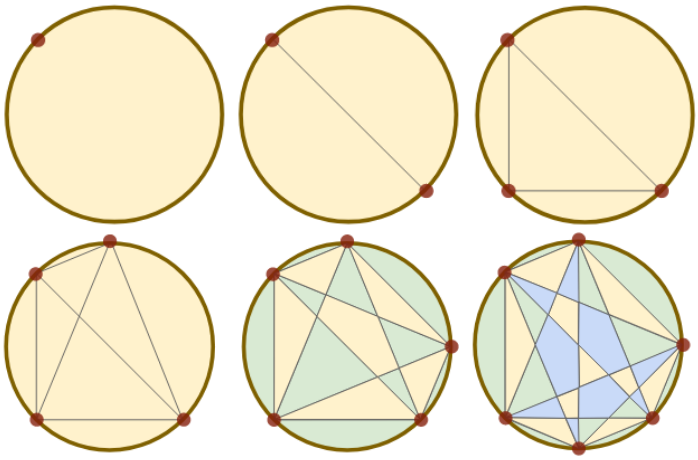

Figure 2: *Here, we color some of the cake pieces just to make them easier to see and to count. The $n = 1$ case doesn't have any cuts. The $n = 5$ case has 5 outside green pieces, 5 inside green pieces, 5 yellow triangles — so far this is three fives, which makes fifteen — and what's left is 1 yellow center piece. The $n = 6$ case has 6 outside green pieces, 6 inside green pieces, 6 outer yellow triangles, 6 blue triangles — so far this is four sixes, which makes twenty four — and what's left are the 7 central blue and yellow shapes.*

Well, it seems that with $n = 1, 2, 3, 4$ people, there are $p(n) = 1, 2, 4, 8$ pieces. (Here, p stands for "pieces", and we'll use this way of writing just to save time). It seems that $p(n)$ doubles for each next n, meaning that p looks like powers of two. In symbols, our guess is: $p(n) \stackrel{?}{=} 2^{n-1}$.

Let's check this guess. Does it continue to $n = 5$? We expect the next power of two: $8 + 8 = 16$. And yes: $p(5)$ really is 16! How about $n = 6$? We expect $16 + 16 = 32$. But — uh oh! — it seems that there are only 31 pieces. The pattern breaks.

An explanation

Let's now figure out $p(n)$ for any n; along the way, we'll see why the powers-of-two pattern seemed to work until $n = 6$. There are two steps: we'll relate cake to constellations and then relate constellations to counting.

We'll use **Euler's formula**, which says that if we connect a bunch of dots by edges to enclose regions, then the numbers of dots, edges, and regions are related:

$$\text{Regions} - \text{Edges} + \text{Dots} = 1$$

For example, the constellation that we build up in stages below has 3 regions (one pentagon and two triangles), 13 edges, and 11 dots. And $3 - 13 + 11 = 1$, just like the formula says. Why is the formula true? Well, we can build up our constellation from

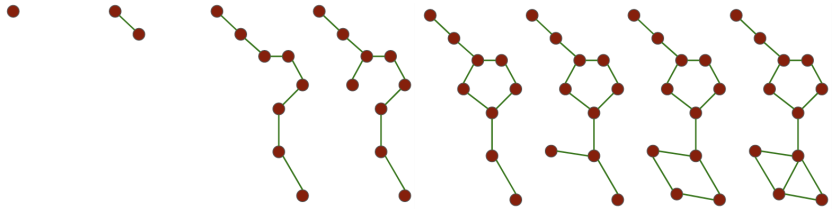

Figure 3: *Steps to build an example constellation (right) starting from a single dot (left). By the way, this method only helps us build constellations that don't have crossing edges, and each two of whose dots are connected by a path of edges. Euler's formula only applies to constellations that follow these rules.*

a single dot. The dot follows the formula (there are no regions and no edges, and $0 - 0 + 1 = 1$). And each step of building keeps the formula true: **either** we connect a new dot to an old dot (so both $-$Edges and $+$Dots change by one, preserving the total) **or** we connect two old dots to create a new region (so both Regions and $-$Edges change by one, preserving the total). This logic proves Euler's formula.

To wrap up, let's think of a cake as a constellation as shown below. In addition to n outer dots, there are $\binom{n}{4}$ inner dots, because from each inner dot emanate 4 rays pointing toward 4 outer dots. By similar logic, the number of edges is $n + 2\binom{n}{2}/2 + 4\binom{n}{4}/2$, since there are n outer arcs (green), $2\binom{n}{2}$ straight-half edges (blue) between outer dots, and 4 half-edges (orange) emanating from each of $\binom{n}{4}$ inner dots.

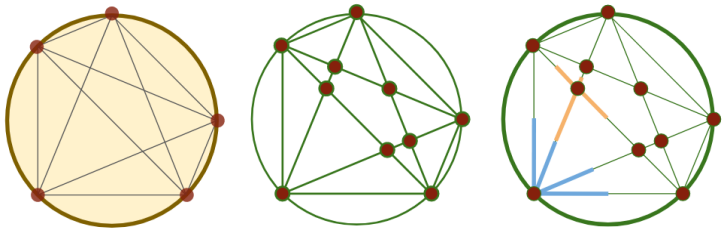

Figure 4: *We can think of a cake (left) as a constellation (middle) by adding inner dots. We can count the half-edges of the constellation (right) by binning them into three groups: the outer arcs (green), the straight half-edges between outer dots (blue), and the straight half-edges emanating from inner dots (orange).*

Putting the pieces together, we see $\text{Regions} = 1 + \text{Edges} - \text{Dots} = 1 + \binom{n}{2} + \binom{n}{4}$. We can simplify this by noting that $1 = \binom{n-1}{0}$ and $\binom{n}{k} = \binom{n-1}{k-1} + \binom{n-1}{k}$, so

$$\text{Regions} = \binom{n-1}{0} + \binom{n-1}{1} + \binom{n-1}{2} + \binom{n-1}{3} + \binom{n-1}{4}$$

In other words, the number $p(n)$ of pieces for n people equals the number of subsets of $n-1$ people with at most 4 members. This explains why we saw powers of two for small n: for $n \leq 5$, counting small subsets is the same as counting all subsets!

What is Learning?

Learning to classify

Today, we'll analyze machines that learn to classify images into two possible buckets such as Cow and Dog.[1] To benefit from math, we need a precise set-up. What do we mean by "learning to classify"?

Say $\mathcal{X}$ contains all possible images and $\mathcal{Y} = \{\text{Cow}, \text{Dog}\}$ is the set of buckets. We posit a probability distribution $\mathcal{D}$ over $\mathcal{X} \times \mathcal{Y}$ that says which pairs $(x, y) \in \mathcal{X} \times \mathcal{Y}$ are more likely and which are less likely. For instance: we might have  ,  , $\cdots \in \mathcal{X}$, and $\mathcal{D}$ might say that ( , Cow) is more likely to occur in nature than ( , Cow), which is more likely to occur than ( , Dog), which in turn is more likely than ( , Dog).

A **classifier** is then a function from images to buckets, that is, some $f \in \mathcal{Y}^{\mathcal{X}}$. By **learning**, we mean acquiring information about $\mathcal{D}$ from "training samples" $\mathcal{S} \in (\mathcal{X} \times \mathcal{Y})^N$ drawn independently from $\mathcal{D}$. So "learning to classify" means mapping $\mathcal{S}$s to fs. In particular, an **algorithm** is a function $\mathcal{L} : (\mathcal{X} \times \mathcal{Y})^N \to \mathcal{Y}^{\mathcal{X}}$.[2]

Instead of considering all possible classifiers, we usually limit ourselves to some subset $\mathcal{H} \subseteq \mathcal{Y}^{\mathcal{X}}$ of especially nice, "candidate" classifiers. As an example, $\mathcal{H}$ might have just three elements:

$$\begin{aligned}\mathcal{H} = \{&\text{always say Cow,}\\ &\text{say Cow if } x \text{ is brown on average and otherwise say Dog,}\\ &\text{say Cow if } x \in \{\text{  ,  ,  }\} \text{ and otherwise say Dog}\}\end{aligned}$$

In practice, $\mathcal{H}$ might actually be a much bigger set of neural networks.

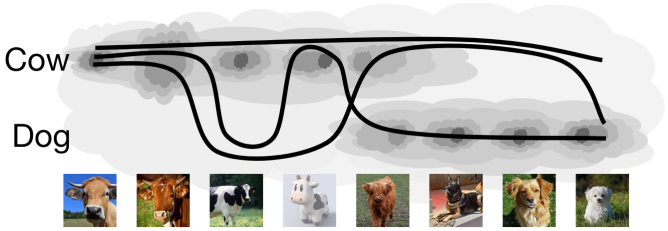

Figure 5: *A cartoon of $\mathcal{D}$ (clouds) and $\mathcal{H}$ (curves) with the $\mathcal{X}$ axis horizontal and the $\mathcal{Y}$ axis vertical.*

[1] Our discussion extends to more complicated situations, but we'll focus on the simple case.

[2] By the way, we don't have to know $\mathcal{D}$ in order to apply our theory; we just named it in order to reason about it.

When is an algorithm good?

An algorithm $\mathcal{L}$ is good when $\mathcal{L}(\mathcal{S})$ accurately classifies images freshly drawn from $\mathcal{D}$. Notice that this is different from merely asking that $\mathcal{L}(\mathcal{S})$ accurately classifies the elements of $\mathcal{S}$! To notate this difference, we define the **out-error** and **in-error** (also called the test and train errors) of a candidate $f \in \mathcal{H}$ as

$$\mathcal{E}_{\text{out}}(f) = \mathbb{P}_{(x,y)\sim\mathcal{D}}\left[f(x) \neq y\right] \qquad \mathcal{E}_{\text{in},\mathcal{S}}(f) = \mathbb{P}_{(x,y)\sim\mathcal{S}}\left[f(x) \neq y\right]$$

Likewise, the out- and in-errors of a algorithm are $\mathcal{E}_{\text{...}}(\mathcal{L}) = \mathcal{E}_{\text{...}}(\mathcal{L}(\mathcal{S}))$ where $\mathcal{S} \sim \mathcal{D}$.

We hope $\mathcal{E}_{\text{out}}$ is low, but we don't know $\mathcal{E}_{\text{out}}$: from $\mathcal{S}$, we can directly calculate only $\mathcal{E}_{\text{in},\mathcal{S}}$. Intuiting that $\mathcal{E}_{\text{gap},\mathcal{S}} = \mathcal{E}_{\text{out}} - \mathcal{E}_{\text{in},\mathcal{S}}$ tends to be small, we might design $\mathcal{L}$ to minimize $\mathcal{E}_{\text{in},\mathcal{S}}$. That is, $\mathcal{L}$ computes $\mathcal{E}_{\text{in},\mathcal{S}}(f)$ for each $f \in \mathcal{H}$, then (breaking ties arbitrarily) to settle on an f with the least $\mathcal{E}_{\text{in},\mathcal{S}}$.[3]

[3] This $\mathcal{L}$ is called **ERM**; its variants dominate modern machine learning.

This $\mathcal{L}$ will work when $\mathcal{E}_{\text{gap},\mathcal{S}}$ is small. But is $\mathcal{E}_{\text{gap},\mathcal{S}}$ actually small? In particular, if $\mathcal{L}(\mathcal{S})$ accurately classifies its training samples, will it also accurately classify test samples it has never before seen?

Sometimes, the answer is **yes**. If there is 1 candidate $f \in \mathcal{H}$ and a thousand datapoints, then $\mathcal{E}_{\text{...}}(\mathcal{L}) = \mathcal{E}_{\text{...}}(f)$, and $\mathcal{E}_{\text{in},\mathcal{S}}(f)$ will very probably be very close to $\mathcal{E}_{\text{out}}(f)$. A technical reason for this is that $\mathcal{L}(\mathcal{S})$ and $\mathcal{S}$ are independent, and we may apply the law of large numbers. It's as if we roll a fair die a thousand times: we'd expect about a sixth of the rolls to land on ⚄.

Other times, the answer is **no**. If every possible classifier is a candidate (so $\mathcal{H}$ is very infinite) and only a few datapoints, then for any training sequence $\mathcal{S}$, many candidates will perfectly classify the training samples. Say that $\mathcal{L}$ breaks ties by settling on

$$f(x) = \text{say Cow if } (x, \text{Cow}) \in \mathcal{S}\text{; otherwise, say Dog}$$

The problem (assuming each image appears with negligible chance and that Cows occur with non-negligible chance) is f will misclassify every fresh image of a Cow as a Dog, so it will have a huge out-error! It's as if we do a trillion experiments where in each experiment we roll a fair die a thousand times: in any one experiment, we'd expect about a sixth of the rolls to land on ⚄, but if we focus on the experiment that has the most ⚄s (which is like focusing on the candidates that do the best on the training data), we are likely to focus on an outlier that has disproportionately many ⚄s!

These two examples illustrate that $\mathcal{L}$'s training performance generalizes to test time when $\mathcal{L}$ uses lots of datapoints to select from only a small set of candidates. How do these forces balance, for example if there are a lot of candidates and also a lot of datapoints? Let's find out.

Is Learning Possible?

A single candidate

Let's analyze the case $|\mathcal{H}| = 1$ more closely. We write p as shorthand for $\mathcal{E}_{\text{out}}(f)$; then $\mathcal{E}_{\text{in},\mathcal{S}}$ counts the fraction of heads that appear in N independent flips of a coin that has chance p of landing heads. Intuitively $\mathcal{E}_{\text{in},\mathcal{S}}$ will usually be close to p when N is big. Let's make "usually", "close to", and "big" precise.

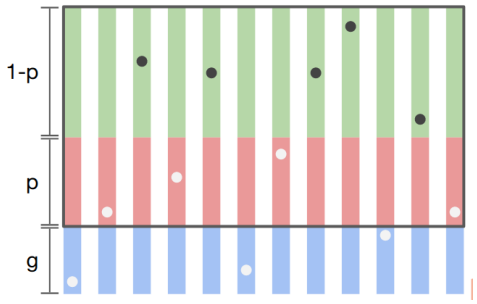

Figure 6: *We randomly select points on N vertical sticks. Each stick has three parts: **green** with length $1-p$, **red** with length p, and **blue** with length g. We call non-blue points **boxed** and non-green points **hollow**.*

We'll switch viewpoints: flipping a coin is like choosing a boxed point on a stick where green means tails and red means heads. We'll show that probably at most $M_0 = (p+g)N$ heads appear. That is, we want to show — given that all points are boxed — that probably at most M_0 points are red. For any $M \geq M_0$:

$$
\begin{aligned}
&\mathbb{P}[M \text{ are red} \mid \text{all are boxed}] \\
&= \mathbb{P}[M \text{ are red and all are boxed}] \;/\; \mathbb{P}[\text{all are boxed}] \\
&= \mathbb{P}[M \text{ are hollow}] \cdot \mathbb{P}[\text{all hollows are red} \mid M \text{ are hollow}] \;/\; \mathbb{P}[\text{all are boxed}] \\
&= \mathbb{P}[M \text{ are hollow}] \cdot (1+g/p)^{-M} \;/\; (1+g)^{-N} \\
&\leq \mathbb{P}[M \text{ are hollow}] \cdot (1+g/p)^{-M_0} \;/\; (1+g)^{-N}
\end{aligned}
$$

Since the above holds for all $M \geq M_0$, we see that:

$$
\begin{aligned}
&\mathbb{P}[\text{at least } M_0 \text{ are red} \mid \text{all are boxed}] \\
&\leq \mathbb{P}[\text{at least } M_0 \text{ are hollow}] \cdot (1+g/p)^{-M_0}/(1+g)^{-N} \\
&\leq (1+g/p)^{-M_0}/(1+g)^{-N} && \text{probabilities are at most 1} \\
&\leq \exp(-M_0 g/p)\exp(Ng) && 1+x \leq \exp(x)
\end{aligned}
$$

We can simplify using algebra to conclude:

$$\begin{aligned} \cdots &= \exp(-(p+g)Ng/p + Ng) && \text{substitute } M_0 = (p+g)N \\ &= \exp(-Ng^2/p) && \text{simplify} \\ &\leq \exp(-Ng^2) && \text{probabilities are at most 1} \end{aligned}$$

This is the **Chernoff bound** for coin flips.[4]

[4] There are lots of versions of this bound. Google up "multiplicative Chernoff bound" or "Bernstein inequality" for some fun examples!

What have we learned? We have learned that if we flip a fair coin N times, the odds of deviating by more than 1% from the expected value of 50% heads decays very quickly with N! Tying back to learning, if candidate f accurately classifies 80% of a large training set ($\mathcal{E}_{\text{in},\mathcal{S}} = 0.2$), then f will probably accurately classify about 80% of its test set ($\mathcal{E}_{\text{out}} \approx 0.2$). For example, if there are $N = 300$ training points, then the odds of f suffering less than 70% test accuracy are less than

$$\exp(-Ng^2) = \exp(-300 \cdot 10\%^2) \approx 5.0\%$$

Notice the exponential dependence on N! Intuitively, exponential dependencies are strong dependencies. For example, if $N = 500$ instead of 300, then we get

$$\exp(-Ng^2) = \exp(-500 \cdot 10\%^2) \approx 0.67\%$$

The chance of an inacceptable test accuracy decays quickly with N. Data is useful!

Like N, the gap bound g can also have a huge effect on the odds of success. What if we think that a generalization gap of $g = 10\%$ is too much? For $g = 5\%$, we find:

$$\exp(-Ng^2) = \exp(-500 \cdot 5\%^2) \approx 29\%$$

The chance of an inacceptable test accuracy increases very quickly as our notion of acceptability becomes more stringent. We can say the same thing a different way by solving for g in terms of a probability δ of failure. We find that with probability at most δ will the generalization gap exceed $\sqrt{\log(1/\delta)/N}$. Call this $\mathcal{G}(\delta)$.

Multiple candidates

What if $\mathcal{H}$ contains multiple (say, H many) candidates? Well, each candidate $f \in \mathcal{H}$ has $\mathcal{E}_{\text{gap},\mathcal{S}}(f) \geq \mathcal{G}(\delta/H)$ with probability at most δ/H. Therefore, the chance that some $f \in \mathcal{H}$ has $\mathcal{E}_{\text{gap},\mathcal{S}}(\mathcal{L}) \geq \mathcal{G}(\delta/H)$ is at most H times δ/H. In sum, with probability at least $1 - \delta$ the gap will be small:

$$\mathcal{E}_{\text{gap},\mathcal{S}}(\mathcal{L}) \leq \max_{f \in \mathcal{H}} \mathcal{E}_{\text{gap},\mathcal{S}}(f) < \mathcal{G}(\delta/H) = \sqrt{\log(H/\delta)/N}$$

In short, if $\mathcal{L}$ selects from among finitely many candidates, then learning is possible: with enough training data, we can ensure the generalization gap is small.

For example, suppose we train a neural net that has 10^4 parameters, each represented by 32 bits. Since there are 2^{32} values for each parameter, $H \leq 2^{32 \cdot 10^4}$. If we train on 10^7 images, then with probability $99\% = 1 - 10^{-2}$, the gap is less than

$$\sqrt{\log(2^{32 \cdot 10^4}/10^{-2})/10^7} \approx 15\%$$

So if the neural net's train accuracy is 97%, its test accuracy will likely exceed 82%.[5]

[5] We proved this without knowing $\mathcal{D}$; we used simply that $\mathcal{D}$ exists at all! Ain't that cool?

Structure in $\mathcal{H}$

Symmetrization

Last time, we bounded the generalization gap by noting that computers (approximately) represent real numbers using 32 bits each:

$$\text{gap} \leq \sqrt{\frac{32 \cdot \text{number of parameters} \cdot \log(2) + \log(1/\delta)}{N}}$$

But, shouldn't 32 bits or 64 bits or infinitely many bits yield similar behavior? Intuitively, the $\mathcal{H}$s we care about depend smoothly instead of jaggedly on the parameters, meaning that tiny changes in the parameters yield practically the same candidate. Let's strive for a bound that depends on $\mathcal{H}$'s "jaggedness" instead of its literal size.

Even though $\mathcal{H}$ may be infinite, the candidates in $\mathcal{H}$ may classify any finite (train set $\mathcal{S}$ and) test set $\tilde{\mathcal{S}}$ in only finitely many ways.[6] So let us fix $f \in \mathcal{H}$ and replace $\mathcal{E}_{\text{out}}(f)$ by $\mathcal{E}_{\text{in},\tilde{\mathcal{S}}}(f)$. We may do this because $\mathcal{E}_{\text{out}}, \mathcal{E}_{\text{in},\tilde{\mathcal{S}}}$ are probably close: by Chernoff, $\mathcal{E}_{\text{out}}(f) \leq \mathcal{E}_{\text{in},\tilde{\mathcal{S}}}(f) + g/2$ with chance at least $1/2$ for reasonable values of g.[7]

$$\begin{aligned}
&\mathbb{P}[\mathcal{E}_{\text{in},\mathcal{S}} + g \leq \mathcal{E}_{\text{out}}] \\
=\ &\mathbb{P}[\mathcal{E}_{\text{in},\mathcal{S}} + g \leq \mathcal{E}_{\text{out}} \mid \mathcal{E}_{\text{out}} \leq \mathcal{E}_{\text{in},\tilde{\mathcal{S}}} + g/2] && \mathcal{S}, \tilde{\mathcal{S}} \text{ are independent} \\
\leq\ &\mathbb{P}[\mathcal{E}_{\text{in},\mathcal{S}} + g/2 \leq \mathcal{E}_{\text{in},\tilde{\mathcal{S}}} \mid \mathcal{E}_{\text{out}} \leq \mathcal{E}_{\text{in},\tilde{\mathcal{S}}} + g/2] && \text{chain the inequalities} \\
\leq\ &\mathbb{P}[\mathcal{E}_{\text{in},\mathcal{S}} + g/2 \leq \mathcal{E}_{\text{in},\tilde{\mathcal{S}}}] \cdot 2 && \mathcal{E}_{\text{out}}, \mathcal{E}_{\text{in},\tilde{\mathcal{S}}} \text{ are probably close}
\end{aligned}$$

Now let f range over $\mathcal{H}_{\mathcal{S}\cup\tilde{\mathcal{S}}}$, the set of candidates **restricted** to $\mathcal{S} \cup \tilde{\mathcal{S}}$. Then:

$$\begin{aligned}
\mathbb{P}[g \leq \mathcal{E}_{\text{gap}}(\mathcal{L}(\mathcal{S}))] \leq\ &\mathbb{P}[\mathcal{E}_{\text{in},\mathcal{S}}(\mathcal{L}(\mathcal{S})) + g/2 \leq \mathcal{E}_{\text{in},\tilde{\mathcal{S}}}(\mathcal{L}(\mathcal{S}))] \cdot 2 \\
\leq\ &\sum_{f \in \mathcal{H}|_{\mathcal{S}\cup\tilde{\mathcal{S}}}} \mathbb{P}[\mathcal{E}_{\text{in},\mathcal{S}}(f) + g/2 \leq \mathcal{E}_{\text{in},\tilde{\mathcal{S}}}(f)] \cdot 2 \\
=\ &\sum_{f \in \mathcal{H}|_{\mathcal{S}\cup\tilde{\mathcal{S}}}} \mathbb{P}[\mathcal{E}_{\text{in},\mathcal{S}}(f) + g/4 \leq \mathcal{E}_{\text{in},\mathcal{S}\cup\tilde{\mathcal{S}}}(f)] \cdot 2
\end{aligned}$$

In the final line above, we noted $\mathcal{E}_{\text{in},\mathcal{S}\cup\tilde{\mathcal{S}}}$ is the average of $\mathcal{E}_{\text{in},\mathcal{S}}$ and $\mathcal{E}_{\text{in},\tilde{\mathcal{S}}}$. Now, imagine $\mathcal{S}$ as sampled *without replacement* from $\mathcal{S} \cup \tilde{\mathcal{S}}$; this should estimate the mean $\mathcal{E}_{\text{in},\mathcal{S}\cup\tilde{\mathcal{S}}}$ even better than sampling with replacement, so Chernoff applies, and each summand is at most $\exp(-Ng^2/16)$. Likewise, there are at most $H(2N)$ many terms if $H(2N)$ is the biggest possible size of $\mathcal{H}_{\mathcal{S}\cup\tilde{\mathcal{S}}}$ given that $|\mathcal{S} \cup \tilde{\mathcal{S}}| = 2N$. So:

$$\cdots \leq H(2N) \cdot \exp(-Ng^2/16) \cdot 2$$

To bound the chance the gap isn't small, we just have to bound $H(2N)$.

[6] Let $|\tilde{\mathcal{S}}| = |\mathcal{S}| = N$. The train and test sets are symmetrical, hence the section's name.

[7] Specifically, when g is no smaller than $2/\sqrt{N}$. A gap of $\approx 1/\sqrt{N}$ is the best we may hope for, since this is the one-candidate gap for lenient δ.

The jaggedness of $\mathcal{H}$

We see that $H(n) \leq 2^n$. What's amazing is that this bound is never somewhat-tight: depending on $\mathcal{H}$, it either is an equality or extremely loose!

Indeed, consider $\mathcal{H}$ restricted to a set S of size n. Let us order S so that we may write each candidate as a string of $+$s and $-$s. We now translate these strings from the alphabet $\{+,-\}$ to the alphabet $\{\blacksquare,\square\}$ in order to clarify their structure. The idea is that $\blacksquare$ represents "surprisingly $+$". More precisely, we translate left to right. Whenever two (partially translated) strings differ in exactly one coordinate $s \in S$, we write $\blacksquare$ for the $+$ version. Otherwise, we write $\square$.

Figure 7: Translating from restrictions of candidates (left) to strings of choice points (right). Each row corresponds to one of 7 candidates and each column corresponds to one of of 4 datapoints. We color pairs of strings that differ in exactly one coordinate.

Observe that each step of this process keeps distinct strings distinct. So there will be as many untranslated strings as translated strings. Moreover, whenever some k indices of a translated string are $\blacksquare$s, then at those k points in S, the of candidates attain all 2^k configurations. This is because $\blacksquare$s mark choice points where the candidates attain both $+$ and $-$. Therefore, **either** $H(n) = 2^n$ for all n, **or** $H(k) \neq 2^k$ for some k. In the latter case, no translated string may have k or more $\blacksquare$s. So $\mathcal{H}_S$ contains at most $\binom{n}{0} + \binom{n}{1} + \cdots + \binom{n}{k-1}$ many strings. We conclude that

$$H(n) \leq \binom{n}{0} + \binom{n}{1} + \cdots + \binom{n}{k-1} \leq (n+2)^{k-1}$$

As with Cake, what might have grown like 2^n ends up growing only polynomially!

Plugging this all in, we find that $g \leq \mathcal{E}_{\text{gap}}(\mathcal{L}(\mathcal{S}))$ with chance at most $(2N+2)^k \cdot \exp(-Ng^2/16) \cdot 2$. We can solve for the gap in terms of our tolerance δ. If we call the (smallest) k for which $H(k+1) \neq 2^{k+1}$ the **jaggedness**[8] of $\mathcal{H}$, then what we get is:

$$\text{gap} \leq \sqrt{\frac{16\log(2N+2)\cdot\text{jaggedness} + 16\log(2/\delta)}{N}}$$

Here's an example. Consider classifying rainfall levels as "desert" or "non-desert" according to how they compare to some positive real number r. Each r gives a different candidate, so there are infinitely many candidates. But for any $k = 2$ rainfall levels $a < b$, no candidate classifies a but not b as "desert". So $\mathcal{H}$ has jaggedness $2 - 1 = 1$. In cases like these, where the jaggedness counts the parameters, we have essentially replaced the 32 (bits per parameter) by $\approx \log(N)$.[9] Intuitively, each parameter has $\approx \log(N)$ bits relevant to distinguishing the datapoints; further bits are irrelevant to overfitting.

Generalization thus stems from $\mathcal{H}$'s shape, not its literal size. Unless $\mathcal{H}$ is infinitely jagged, the gap will shrink as N grows. In this case, learning from data is possible.

[8] To recap, the jaggedness of $\mathcal{H}$ is the smallest k such that $\mathcal{H}$ fails to fit all 2^{k+1} many $\pm$ patterns on each dataset S of size $k+1$. If no such k exists, we say that $\mathcal{H}$ is infinitely jagged. The jaggedness is also known as the *Vapnik-Chervonenkis dimension*.

[9] Due to the constants involved, this is in practice not a substantial improvement.

Linear classification

Linear candidates

Today we will develop a practical candidate class $\mathcal{H}$ and learning rule $\mathcal{L}$. The picture is that we cut the input space $\mathcal{X} = \mathbb{R}^d$ into two halves by a line or plane through the origin (Figure 8). A convenient way to write $\mathcal{H}$ is in terms of linear functions θ:

$$\mathcal{H} = \mathcal{H}_{\text{lin}} = \{x \mapsto \text{sign}(\theta(x)) \text{ where } \theta : \mathbb{R}^d \to \mathbb{R} \text{ is linear}\}$$

Here, sign sends negative numbers to Dog and positive numbers to Cow. Let's not worry about the boundary case of 0 except to note that images on which θ is 0 together form a **decision boundary** between the Cows and the Dogs.

The jaggedness of $\mathcal{H}$ is at most d, because any $d+1$ points $x_0, x_1, \cdots x_d \in \mathbb{R}^d$ must participate in some linear relation:

$$x_s = \sum_{i \in I} c_i x_i - \sum_{j \in J} c_j x_j$$

for some $\{s\}, I, J$ mutually disjoint and each c positive. Thus, if we assign to each x_i a positive label (Cow) and to each x_j a negative label (Dog), then x_s must have a positive label (Cow). Therefore, not all 2^d possibilities occur. As long as $\sqrt{\log(N) \cdot d / N}$ is small, learning with $\mathcal{H}_{\text{lin}}$ will generalize.

By the way, what if lines through the origin don't make sense for our classification problem? For example, what if we want to model decision boundaries that don't pierce the origin? Well, one trick is to pre-process the data so that linear classification of the transformed data corresponds to richer classification of the old data. If we map

$$\text{old features} = (a, b) \mapsto \text{new features} = (1, a, b)$$

then boundaries may avoid the origin (Figure 9). This is called "adding a **bias** term". Instead of classifying points in $\mathcal{X} = \mathbb{R}^2$, we now classify points in $\mathcal{X} = \mathbb{R}^3$, meaning that our jaggedness is 3 instead of 2 and generalization may be a bit worse.

The same trick helps us enrich the class of candidates beyond straight lines. For example, we might featurize using polynomials:

$$\text{old features} = (a, b) \mapsto \text{new features} = (1, a, b, a^2, ab, b^2, a^3, a^2 b, ab^2, b^3)$$

to get decision boundaries as in Figure 11. Instead of classifying points in $\mathcal{X} = \mathbb{R}^2$, we now classify points in $\mathcal{X} = \mathbb{R}^{10}$, meaning that our jaggedness is 10 instead of 2 and generalization may be substantially worse.

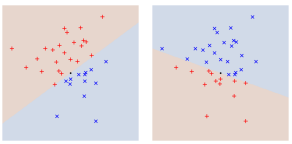

Figure 8: *Two fs in $\mathcal{H}_{lin}$. A line through the origin divides the plane ($\mathcal{X} = \mathbb{R}^2$) into red (Cow) and blue (Dog) parts. We also show how each f classifies some random points.*

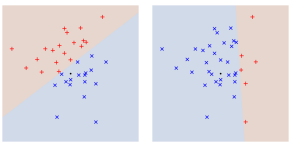

Figure 9: *Decision boundaries with bias may avoid the origin.*

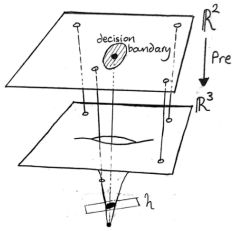

Figure 10: *Linear planes (bottom) in the transformed $\mathcal{X}$ induce non-linear boundaries (top) in the old $\mathcal{X}$.*

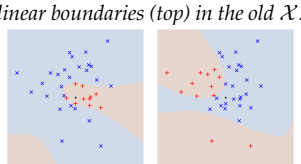

Figure 11: *Polynomial features allow decision boundaries to exhibit peninsulas, channels, and islands.*

Optimization

Now, how do we select $f \in \mathcal{H}_{\text{lin}}$ based on the data $\mathcal{S}$? Since the jaggedness d is finite, we expect that given enough data, the test and train error will be very close. So it makes sense to (approximately) minimize the training error:

$$N \cdot \mathcal{E}_{\text{in},\mathcal{S}}(\theta) = \sum_{(x,\text{Cow})\in\mathcal{S}} \begin{cases} 1 & \text{if } \theta(x) < 0 \\ 0 & \text{otherwise} \end{cases} + \sum_{(x,\text{Dog})\in\mathcal{S}} \begin{cases} 1 & \text{if } \theta(x) > 0 \\ 0 & \text{otherwise} \end{cases}$$

Now, a general strategy to find a good f is to start with some f and to keep replacing it with better f's. The problem is that $\mathcal{E}_{\text{in},\mathcal{S}}(f)$ takes discrete values $0/N$, $1/N$, $\cdots$, N/N, so it is hard to tell in which direction we should nudge f to improve it.

Imagine that the true label might be Cow, meaning that we want $\theta(x) > 0$, but that the current guess for θ has $\theta(x) = -0.20$. In some sense, $\theta(x) = -0.15$ is an improvement, even though it is still a misclassification. Likewise, $\theta(x) = +0.05$ is precariously close to being negative, even though it is currently a correct classification. Therefore, let's take small steps to improve a notion of **badness** according to which -0.20 is worse than -0.15 is worse than 0.05 is worse than 1.00. At some point, we consider the classification safe instead of precarious, so let's say that 1.00 is no worse than 1.05. Turning this into a formula, we get:[10]

$$\text{badness}(\theta) = \sum_{(x,\text{Cow})\in\mathcal{S}} \begin{cases} +1-\theta(x) & \text{if } \theta(x) < +1 \\ 0 & \text{otherwise} \end{cases} + \sum_{(x,\text{Dog})\in\mathcal{S}} \begin{cases} -1+\theta(x) & \text{if } \theta(x) > -1 \\ 0 & \text{otherwise} \end{cases}$$

Notice that each term of badness is at least the corresponding term for $N \cdot \mathcal{E}_{\text{in},\mathcal{S}}$. Minimizing badness thus makes sense as an approximate way to minimize $\mathcal{E}_{\text{in},\mathcal{S}}$.

The upshot is that we can make incremental improvements with small steps. What's the difference between badness(θ) and badness$(\theta+\epsilon)$, where ϵ is a "small" linear function? Each of the "$+1-\theta(x)$" terms of increases by $\epsilon(x)$ when we change θ into $\theta+\epsilon$. So the net decrease in badness is:[11]

$$\begin{aligned} \text{improvement} &\approx \sum_{(x,\text{Cow})\in\mathcal{S}} \begin{cases} \epsilon(x) & \text{if } \theta(x) < +1 \\ 0 & \text{otherwise} \end{cases} + \sum_{(x,\text{Dog})\in\mathcal{S}} \begin{cases} -\epsilon(x) & \text{if } \theta(x) > -1 \\ 0 & \text{otherwise} \end{cases} \\ &= \epsilon\left(\sum_{\substack{(x,\text{Cow})\in\mathcal{S} \\ \text{and } \theta(x)<+1}} x - \sum_{\substack{(x,\text{Dog})\in\mathcal{S} \\ \text{and } \theta(x)>-1}} x \right) = \epsilon(x_{\text{squeaky}}) \end{aligned}$$

Above, we used linearity of ϵ and then renamed its input x_{squeaky} to save ink.[12] How shall we choose a small linear function ϵ that makes an improvement? There are lots of choices, and each leads to a different algorithm. If we write out input data in coordinates, like $x = (x_a, x_b, x_c, \cdots)$, then one way is to define:

$$\epsilon(y) = 0.03 \cdot x_a y_a + 0.02 \cdot x_b y_b + 0.01 \cdot x_a y_b + 0.01 \cdot x_c y_c + \cdots$$

We chose small coefficients to ensure that ϵ is "small", and as long as the coefficients of the cross terms like $0.01 \cdot x_a y_b$ are close to 0, this formula will guarantee that the improvement $\epsilon(x_{\text{squeaky}})$ is positive.

[10] There are lots of ways to turn the qualitative pattern we discussed into a formula. What we present is called **hinge loss**. Our favorite is called **logistic loss**, because it has a beautiful probabilistic interpretation, but we won't have time to talk about it.

[11] Well, actually this isn't exactly right, since whether we use the "ifs" of the "otherwises" might change between θ and $\theta+\epsilon$. We imagine that ϵ is so small that this is negligible. That's what we mean by "small", and that's why we write $\approx$ instead of $=$.

[12] After all, each term in x_{squeaky} is (or is precariously close to being) misclassified. And you know what they say: *the squeaky wheel gets the grease*.

Regularization

Rademacher complexity

Margin bounds