

Generalization in Machine Learning

Sam, Joe, and Arthur

2020 Summer

1 The Cake Problem

1.1 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. When 4 people encircle the cake, it ends up cut into 8 pieces. We wonder: how many pieces arise when a different number of people sit around the cake?

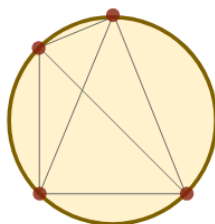


Figure 1: With $n = 4$ people, we make $\binom{n}{2} = 6$ cuts, which gives 8 pieces in total (4 outside and 4 inside).

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?

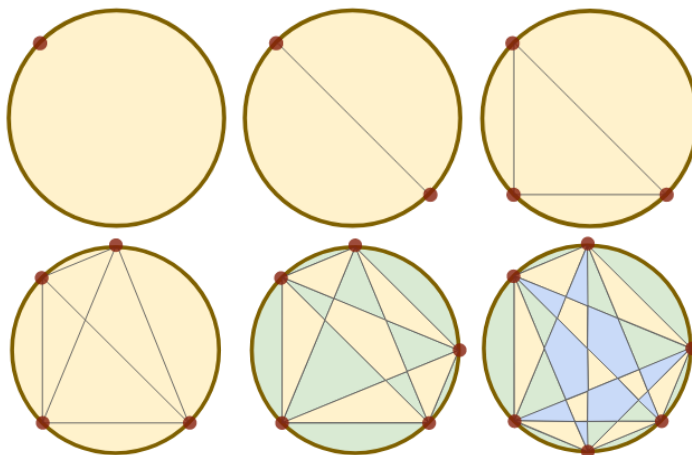


Figure 2: Here, we color some of the cake pieces just to make them easier to see and to count. The $n = 1$ case doesn't have any cuts. The $n = 5$ case has 5 outside green pieces, 5 inside green pieces, 5 yellow triangles — so far this is three fives, which makes fifteen — and what's left is 1 yellow 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, namely $8 + 8 = 16$. 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. So the pattern breaks.

1.2 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 polyhedron formula**. This formula says that if we connect a bunch of dots by edges to form some enclosed 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?

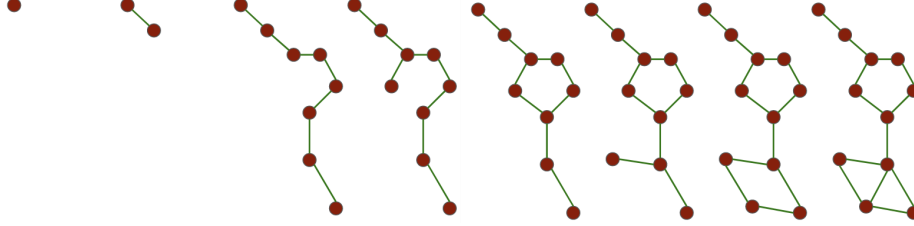


Figure 3: Steps to build an example constellation (right) starting from a single dot (left). By the way, this method only helps us build constellations that don't have crossing edges, and each two of whose dots are connected by a path of edges. Euler's formula only applies to constellations that follow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

Well, we can build up our constellation starting from a single dot. The single dot follows the formula (since there are no regions and no edges, and $0 - 0 + 1 = 1$). And each step of building preserves the formula: **either** we connect a new dot to an old dot (so both $-Edges$ and $+Dots$ change by one, meaning that the total stays the same) **or** we connect two old dots to create a new region (so both $Regions$ and $-Edges$ change by one, meaning that the total stays the same). 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.

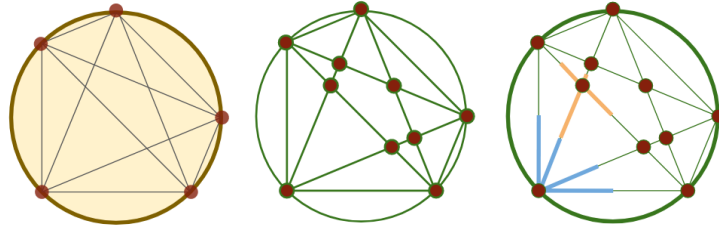


Figure 4: We can think of a cake (left) as a constellation (middle) by adding inner dots. We can count the half-edges of the constellation (right) by binning them into three groups: the outer arcs (green), the straight half-edges between outer dots (blue), and the straight half-edges emanating from inner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

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

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

Since $\binom{n-1}{0} + \dots + \binom{n-1}{k} = 2^{n-1}$ as long as $k \leq n-1$, this looks like powers of two as long as $n \leq 5$. So we have solved the mystery. With n people, the number $p(n)$ of pieces equals the number of subsets of $n-1$ people with at most 4 members. Ain't that cool?

2 Finite Hypothesis Classes

2.1 What is Learning?

Today, we'll analyze machines that learn to classify images into two possible buckets such as Cow and Dog. 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 $\{\text{img1}, \text{img2}, \dots\} \in \mathcal{X}$, and \mathcal{D} might say that $(\text{img1}, \text{Cow})$ is more likely to occur in nature than $(\text{img2}, \text{Cow})$, which is more likely to occur than $(\text{img2}, \text{Dog})$, which in turn is more likely than $(\text{img3}, \text{Dog})$.¹

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 f s. In particular, an **algorithm** is a function $\mathcal{L} : (\mathcal{X} \times \mathcal{Y})^N \rightarrow \mathcal{Y}^{\mathcal{X}}$.

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:

$$\begin{aligned} \mathcal{H} = \{ & \text{always say Cow,} \\ & \text{say Cow if } x \text{ is brown on average and otherwise say Dog,} \\ & \text{say Cow if } x \in \{\text{img1}, \text{img2}, \text{img3}\} \text{ and otherwise say Dog} \} \end{aligned}$$

In practice, \mathcal{H} will actually be something like a set of neural networks, which is a much bigger set.

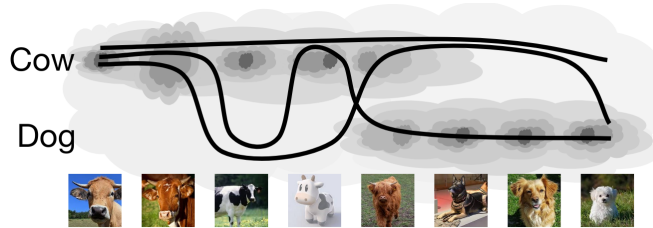


Figure 5: A cartoon of \mathcal{D} (clouds) and \mathcal{H} (curves) with the \mathcal{X} axis horizontal and the \mathcal{Y} axis vertical.

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** of a candidate $f \in \mathcal{H}$ as

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

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

We hope \mathcal{E}_{out} is low, but we don't know \mathcal{E}_{out} : from \mathcal{S} , we can directly calculate only $\mathcal{E}_{\text{in},\mathcal{S}}$. Intuit 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 $\mathcal{E}_{\text{in},\mathcal{S}}$.

The ERM \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 millions of datapoints, then $\mathcal{E}_{\text{in},\mathcal{S}}(\mathcal{L}) = \mathcal{E}_{\text{out}}(\mathcal{L})$, then $\mathcal{E}_{\text{in},\mathcal{S}}(f)$ will very probably be very close to $\mathcal{E}_{\text{out}}(f)$. This is for the same reason that we expect close to 16.6% sixes among a million rolls of a fair die. 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 is, f will misclassify every fresh image of a Cow as a Dog, so it will have a huge out-error!

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.

¹Let's say that each individual image appears with has negligible probability and that Cows occurs roughly as much as Dogs.

2.2 Is Learning Possible?

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

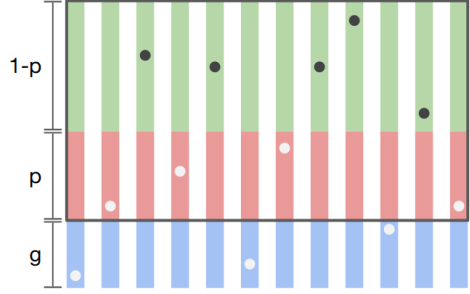


Figure 6: We randomly select points on N vertical sticks. Each stick has three parts: **green** with length $1-p$, **red** with length p , and **blue** with length g . We call non-blue points **boxed** and non-green points **white**.

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 $M \geq M_0$:

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

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

$$\begin{aligned}
& \mathbb{P}[\text{at least } M_0 \text{ are red} \mid \text{all are boxed}] \\
&\leq (1+g/p)^{-M_0} / (1+g)^{-N} && \text{probabilities are at most 1} \\
&\leq \exp(-M_0 g/p) \exp(Ng) && \text{exp is convex} \\
&= \exp(-(p+g)Ng/p + Ng) = \exp(-Ng^2/p) && \text{substitute } M_0 = (p+g)N \\
&\leq \exp(-Ng^2) && \text{probabilities are at most 1}
\end{aligned}$$

This is the **Chernoff bound** for coin flips. We can say the same thing in a different way by solving for g ; we find that with probability at most δ will the generalization gap exceed $\sqrt{\log(1/\delta)/N}$. Call this $\mathcal{G}(\delta)$.

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

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

What we have shown is that, if our algorithm \mathcal{L} selects from among a finite number of candidates, then learning is possible: the more data points we have, the smaller will be the generalization gap bound $\mathcal{G}(\delta/H)$.

Here's an example. Suppose we train a neural that has 10^3 parameters, each a floating point number. Since there are at most 2^{32} possible floating point numbers, H is at most $2^{32 \cdot 10^3}$. If we train on 10^6 images, then with probability 99.99% = $1 - 10^{-4}$, the generalization gap is less than

$$\sqrt{\log(2^{32 \cdot 10^3} / 10^{-4}) / 10^6} \approx 9.8\% \approx 10\%$$

So if after training, the neural net's training accuracy is 80%, we can expect its testing accuracy to be at least 70%. It is amazing that we can mathematically prove this about its test accuracy.

3 Infinite Hypothesis Classes

3.1 Symmetrization

3.2 VC Dimension

4

5

6

Helping Handout: Binomial Coefficients $\binom{n}{k}$

How many ways can we select k people out of n people? For example, there is just 1 way to choose a pair ($k = 2$) from among $n = 2$ people. There are 3 pairs among $n = 3$ people. And there are 6 pairs among $n = 4$ people. We write the answer to this question as $\binom{n}{k}$; so $\binom{4}{2} = 6$.

While $\binom{n}{k}$ counts size- k subsets, so that picking Alice and Bob is the same as picking Bob and Alice, we might want to count ordered subsets. How about if we want to order $k = 4$ people out of $n = 10$ people? Well, there are 10 ways to select the first person, 9 ways to select the second person, 8 ways to select the third person, and 7 ways to select the fourth person. So the answer is $10 \cdot 9 \cdot 8 \cdot 7$. But each size- k unordered subset corresponds to $4 \cdot 3 \cdot 2 \cdot 1$ ordered subsets. So we conclude that

$$\binom{10}{4} = \frac{10 \cdot 9 \cdot 8 \cdot 7}{4 \cdot 3 \cdot 2 \cdot 1}$$

Notice that $\binom{n}{0} = 1$, since there is always exactly one way to choose none of the n people. Also, when n is positive, $\binom{n}{k} = \binom{n-1}{k-1} + \binom{n-1}{k}$, since to choose k folks from among a group of $n - 1$ students and 1 teacher, we can **either** choose k students and leave out the teacher or choose $k - 1$ students and also the teacher. Another neat fact is that $\binom{n}{0} + \binom{n}{1} + \binom{n}{2} + \cdots + \binom{n}{k} = 2^n$ as long as $n \leq k$. This is because that sum of binomial coefficients counts the number of ways to choose some subset of size at most k from among n people.

Helping Handout: Asymptotics

Helping Handout: Probability and Expectation

How can we model the notion of **chance** using math? For example, we might have a set $\mathcal{X} = [0 \text{ cm}, 50 \text{ cm}]$ of possible daily rainfall levels. Intuitively, there is zero chance that any one particular rainfall level like $6.28318 \cdots \text{ cm}$ will occur today. But we'd say there is some chance that the rainfall will fall within, say, $[6.2 \text{ cm}, 6.3 \text{ cm}]$.

Thus, in order to treat \mathcal{X} probabilistically, we want to assign to each interval $[a, b]$ inside \mathcal{X} a nonnegative number $\mathbb{P}([a, b])$ that tells us the chance that rainfall will fall within $[a, b]$.

More generally, we In order for this bunch of numbers to make sense, we demand $\mathbb{P}(\mathcal{X}) = 1$ and $\mathbb{P}([a, c]) + \mathbb{P}([b, b]) = \mathbb{P}([a, b]) + \mathbb{P}([b, c])$.

Helping Handout: Vectors and Covectors