Dr. Bayeslove: How I Learned to Stop Worrying and Love the model

Viswadruth Akkaraju

February 2026

1 Introduction

God does play dice, especially in our day-to-day life, and no model can ever predict anything without a pilgrimage to the cathedral of *Probability*.

What my incoherent dramatics were meant to emphasize is that machine learning cannot truly be understood without a chat with the humble priest of probability theory: the random variable. And that is exactly what the first lecture in class focused on alongside a high level Idea of ML.

2 What is a Model and Will I See It on the Cover of Vogue

So, in the words of a wise man: we must start at the start. And what is the start?

(I promise I am sober as I write this.)

When we say **machine learning model**, what we usually mean is a mathematical object (or a mysterious black box, depending on how much coffee you have had) that takes an **input** and produces an **output**. Formally, we think of a model as a function

$$f : \mathcal{X} \to \mathcal{Y},$$

which means it takes some input $x \in \mathcal{X}$ and produces an output $y \in \mathcal{Y}$.

The input x could be literally anything: height, weight, temperature, the number of hours you spent doomscrolling TikTok, or even your credit card number and CVV (please do not give that to the model, but I feel free to email them to me). The output y could be something equally diverse: whether an email is spam, whether a tumor is malignant, whether you will pass this class, or the probability that the object in a picture is a cat.

So in short: a model takes in data, and outputs a prediction. Or, paraphrasing a famous Indian politician : a machine learning model is the machine that takes in a potato and outputs gold. This is hyperbole, of course, but to be fair,

ML predictions often *are* gold, especially to companies trying to monetize your existence.

We can visualize the process as follows:

Figure 1: A machine learning model but minimalist

Of course, in reality, the computer does not have a tiny cat inside it making decisions (unfortunately). Instead, the model must **learn a relationship** between input and output from data. That is, given training examples $\{(x_i, y_i)\}_{i=1}^{N}$, we want to find a function f such that $f(x)$ gives a reasonable prediction of y.

2.1 A Simple Example: Linear Models

The simplest possible model is a line:

$$y = mx + c,$$

where m and c are parameters. Learning the model means choosing m and c so that the line fits the data reasonably well. This is the essence of **linear regression** a topic in a later lecture.

Now you may ask: if we already have an equation, why do people spend $5000 on a GPU? The answer is simple: **real life is not linear.**

If life were linear, my bank account would be a straight line too, and unfortunately they are more like squiggly lines with the occasional catastrophic jumps.

Well then why don't we just you know use unlimited power instead (or in this case use a very high order polynomial for every problem)?.

Figure 2: You and Me RN waiting for our fields medal for solving AGI

2.2 Model Complexity: Why nuking a cockroach infestation is not the solution

Often, a straight line is not flexible enough to represent the relationship between x and y. So we consider richer families of functions. A common example is the polynomial model:

$$y = a_n x^n + a_{n-1} x^{n-1} + \cdots + a_1 x + a_0,$$

where the coefficients $a_0, a_1, \ldots, a_n$ are parameters we must learn from data.

The degree n is often called the **order** of the polynomial. A degree-1 polynomial gives us a line. A degree-7 polynomial gives us something much more flexible. In general, higher order means more expressive power.

However, power comes with responsibility.

If we choose a model that is **too simple**, it will fail to capture important structure in the data. This is called **underfitting**.

If we choose a model that is **too complex**, it might fit not only the true pattern but also the random noise. This is called **overfitting**.We call this process of selecting the polynomial degree as model selection.

In other words, a model will always learn *something*, but it can learn the wrong thing. This is like teaching a kid that cocaine is healthy and sugar is poison: you are technically teaching, but the NHS will not thank you.

So we must choose model complexity carefully. Throwing a nuclear bomb at a roach infestation may solve the roach problem, but you will also lose the house, the neighborhood, and possibly the concept of civilization.

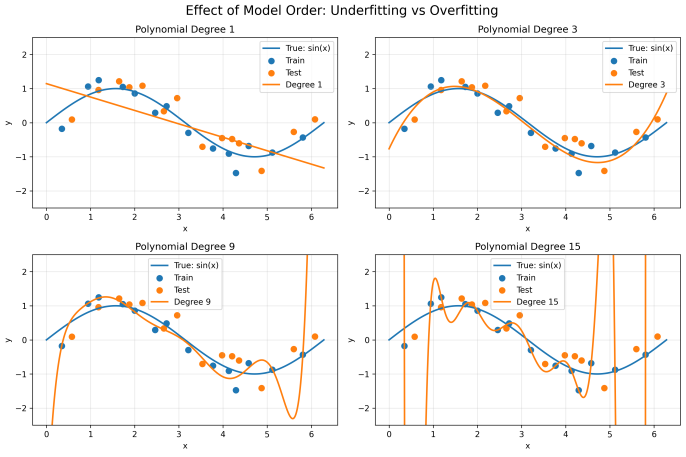

Figure 3: Choosing the wrong model order: too small underfits, too large overfits, and degree 15 starts behaving like it actually is on cocaine.

2.3 Manifolds, Spaces, and the Weird Shapes of Reality

Machine learning often deals with data that is stored using a lot of numbers. For example, an image might be represented by tens of thousands of pixel values.

At first glance, this makes reality look terrifyingly high-dimensional. So before we talk about manifolds, we should clarify what we even mean by **dimension**.

In simple terms, the **dimension** of a space is the number of coordinates needed to describe a point in it.

For example:

- A point on a line needs one number (x), so a line is **1-dimensional**.
- A point on a flat page needs two numbers ((x, y)), so the plane is **2-dimensional**.
- A point in the real world needs three numbers ((x, y, z)), so physical space is **3-dimensional**.

So when we say data is **high-dimensional**, we simply mean that each data point requires a large number of coordinates to represent. For instance, an image with 256×256 pixels requires 65,536 numbers to store, so mathematically it lives in a space like $\mathbb{R}^{65536}$.

On the other hand, something is **low-dimensional** if it can be described using only a few numbers.

Now comes the important idea:

Even if data is written using many coordinates, the valid data can be represented as part of a simpler (but higher dimensional structure) similar to borrowing when subtracting we can borrow dimensions to make the maths less painful

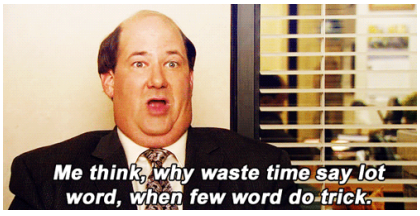

Figure 4: How the Manifold was discovered

To see what this means, consider a simple example.

A point on a flat page can be represented by two coordinates (x, y), so it seems like we need two numbers to describe it. However, if the point is forced to lie on a circle, then it must satisfy the constraint

$$x^2 + y^2 = 1.$$

This means that not every pair (x, y) is allowed.

In fact, every point on the circle can be generated using just a single parameter t:

$$x = \cos(t), \qquad y = \sin(t).$$

So although the circle lives inside $\mathbb{R}^2$, it can be described using only one variable. In other words, the circle behaves like a one-dimensional object hiding inside a two-dimensional space.

This is the intuition behind a **manifold**: it is a structured surface (or curve) that lives inside a larger space, and the data is constrained to stay on that surface.

Now for the machine learning connection.

An image might be represented as a vector in a massive space like $\mathbb{R}^{65536}$ (one coordinate per pixel). But realistic images do not fill this entire space. Most points in $\mathbb{R}^{65536}$ correspond to random noise, not actual photographs.

Real images are constrained by structure: lighting, shapes, edges, geometry, and the physical rules of the universe (sadly, no floating eyeballs). So the set of realistic images occupies only a tiny structured region inside the larger space.

The moral of the story is:

Real-world data is high-dimensional, but the same data usually lies on a smaller structured manifold.

Machine learning works well when our models learn this structure instead of memorizing noise.

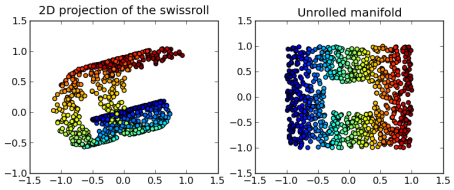

Figure 5: A Swiss roll manifold unrolled (tasty)

2.4 Why Taylor Series Shows Up (and Why It's on Tour Through the Eras of Math)

One reason polynomials appear everywhere is because they are the "Legos" of mathematics. If you have enough polynomial pieces, you can build something that looks like almost any smooth function.

The Taylor series is one of the most famous ways of doing exactly that.

Suppose we have some complicated function $f(x)$, and we want to understand it near a point a. Instead of trying to deal with the full function, we approximate it using a polynomial:

$$f(x) \approx f(a) + f'(a)(x-a) + \frac{f''(a)}{2!}(x-a)^2 + \cdots$$

At first glance this looks like a random pile of derivatives. But the intuition is surprisingly simple:

- $f(a)$ tells us the **height** of the function at a.
- $f'(a)$ tells us the **slope** (how the function is tilted).
- $f''(a)$ tells us the **curvature** (how the slope itself is changing).
- higher derivatives tell us how the function bends, twists, and behaves in increasingly detailed ways.

So Taylor series is basically the process of asking a function:

"Give me your value, your slope, your curvature, your higher-order derivatives yearning to be free, and I shall raise from them an approximate polynomial"

Each additional term in the Taylor series gives us a higher-degree polynomial, meaning a more flexible approximation. A first-order Taylor approximation is just a straight line. A second-order approximation gives us a parabola. Higher orders can create increasingly strange and expressive shapes.

This is why Taylor series matters for machine learning: it explains why polynomials (and more generally, flexible function families) can approximate complicated relationships.

Even neural networks are essentially doing a similar job: they create highly nonlinear functions by combining simpler nonlinear pieces. The difference is that neural networks do not explicitly write down a Taylor series; they *learn* a flexible function through parameters.

So when we increase the **model order** (or model complexity), we are giving our model more expressive power. This can be good, because the real world is complicated. But it can also be dangerous, because too much expressivity allows the model to fit noise.

At the end of the day, machine learning is often just this:

choose a family of functions, learn the parameters, and pray the model generalizes.

Or in other words: build a powerful approximation machine, and hope it learns the true pattern instead of hallucinating 2+2=5.

3 Terminology and Latin: Duolingo for ML

Machine learning is not difficult because the math is impossible. Machine learning is difficult because every sentence contains at least three Latin words and a probability distribution.

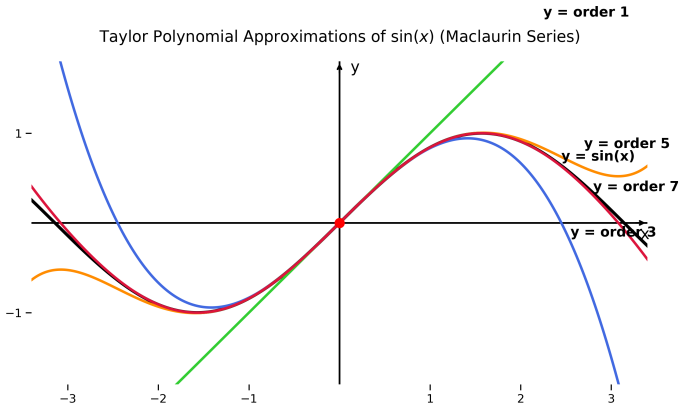

Figure 6: Taylor (Maclaurin) polynomial approximations of $\sin(x)$ about $x = 0$. Higher-order polynomials match the true function more closely near the expansion point.

So before we do anything meaningful, we must learn the sacred vocabulary of uncertainty. This sadly is also the end of pictures for a while, its only a bunch of math (THE HORROR)

3.1 Uncertainty: Did I Leave the Gas On?

At the heart of machine learning is the fact that the world is not deterministic (at least not the interesting parts of it). After all, how fun would books, poetry, or paintings be if everything were always the same? Imagine every movie being like the smashing hit *Morbius*. Terrifying.

Think of your favourite reality TV show: we love it because of the drama, and drama is simply uncertainty in human form.

Similarly, most real-world problems that machine learning tries to solve prediction, classification, decision making, forecasting are uncertain by nature. Even if you have the best model, the best GPU, and the blessings of every professor on earth, you still cannot perfectly predict reality.

This is because uncertainty comes from two main sources:

- randomness in the world itself,
- and ignorance in our model or knowledge.

3.2 Likelihood: Probability's Evil Twin

The word **likelihood** is confusing because it looks exactly like probability, is written exactly like probability, and behaves exactly like probability. But it is not probability in the usual sense. It is probability viewed from the opposite direction.

Suppose we have a probabilistic model

$$p(x \mid \theta),$$

where:

- x is the observed data,
- θ is an unknown model parameter.

If θ is fixed, then $p(x \mid \theta)$ is a probability distribution over x. This is the usual probability question:

"If the world truly had parameter θ, what is the probability that I would see data x?"

However, machine learning often works in reverse. We observe the data first, and then we want to guess what the parameter θ should be.

3.2.1 The Dating App Analogy

Think of x as your dating app profile: your photos, bio, height, music taste, and the fact that you still listen to Vanilla Ice. That is the data.

Now think of θ as the type of person the dating app thinks you are: funny, boring, mysterious, emotionally available (unlikely), etc. That is the hidden parameter.

Now the dating app is basically asking:

"If this person had personality type θ, how likely is it that they would produce this profile x?"

That is exactly what likelihood measures.

So we define the **likelihood function** as:

$$L(\theta) = p(x \mid \theta),$$

but now we treat x as fixed (because we already observed the profile) and we treat θ as the unknown.

So likelihood is not the probability of θ. Instead, it is a score that tells us how well a particular θ explains the observed data.

In machine learning, training often means choosing the parameter θ that maximizes this score:

$$\hat{\theta}_{\text{MLE}} = \arg\max_{\theta} L(\theta),$$

which is called **Maximum Likelihood Estimation (MLE)**.

Probability predicts data from parameters. Likelihood judges parameters using data.

So likelihood is probability, but with commitment issues: same formula, opposite mindset.

3.3 Prior and Posterior: The Before and After of Belief

Now comes the Bayesian magic (For now Bayseian is just a word, we will give meaning to it later just trust me bro).

In Bayesian probability, parameters are not fixed unknown numbers. They are treated as random variables (another trust me bro moment, for now think of it as a variable with some value).

So we assign a probability distribution to the parameter θ itself.

- The **prior** $p(\theta)$ represents what we believe about θ before seeing any data.
- The **posterior** $p(\theta \mid x)$ represents what we believe after observing data.

Bayes' theorem connects them:

$$p(\theta \mid x) = \frac{p(x \mid \theta)p(\theta)}{p(x)}.$$

This equation is basically the holy scripture of Bayesian machine learning.

The posterior is our updated belief about the model parameters, after reality has slapped us with data.

3.4 Epistemic vs Aleatoric Uncertainty: Ignorance vs Chaos

Uncertainty in machine learning comes in two main flavors, and both have fancy names:

3.4.1 Aleatoric Uncertainty (Data Uncertainty)

Aleatoric uncertainty is uncertainty that comes from randomness in the data itself. Even if you knew the perfect model, you could not eliminate it.

Example:

- sensor noise,
- randomness in weather,
- measurement errors,
- humans being inconsistent.

This uncertainty is irreducible. In other words, the universe is rolling dice and refuses to stop.

3.4.2 Epistemic Uncertainty (Model Uncertainty)

Epistemic uncertainty comes from our ignorance. It exists because we do not have enough data or because our model is imperfect.

Example:

- training data is limited,

- model has never seen this input region before,
- distribution shift (reality updated its software without telling you).

This uncertainty is reducible: more data, better models, and better assumptions can reduce epistemic uncertainty.

3.5 Covariance: When Random Variables Gossip About Each Other

Variance tells us how much *one* random variable spreads out. (And if you don't know what variance is probably figure that out first) But in machine learning we almost never have just one variable. We have many: height, weight, GPA, number of coffees consumed, and how close we are to a mental breakdown.

So we need a way to measure whether two random variables "move together".

3.5.1 The Basic Question

Covariance answers a very simple question:

When X goes up, does Y also tend to go up?

If yes, they are positively related. If no, they might be negatively related. If they have no relationship, then they are basically strangers.

3.5.2 The Definition

The covariance between two random variables X and Y is defined as:

$$\mathrm{Cov}(X, Y) = \mathbb{E}[(X - \mathbb{E}[X])(Y - \mathbb{E}[Y])].$$

This looks scary, but the meaning is simple.

- If X is larger than its average, then $(X - \mathbb{E}[X])$ is positive.
- If Y is also larger than its average, then $(Y - \mathbb{E}[Y])$ is positive.
- Positive $\times$ positive = positive.

So if X and Y are usually above average at the same time (or below average at the same time), the covariance becomes positive.

On the other hand, if X is above average while Y is usually below average, the product becomes negative, and the covariance becomes negative.

3.5.3 How to Interpret Covariance

- If $\text{Cov}(X, Y) > 0$, X and Y tend to move in the same direction.
- If $\text{Cov}(X, Y) < 0$, X and Y tend to move in opposite directions.
- If $\text{Cov}(X, Y) = 0$, they are uncorrelated (they are not texting each other).

A real-life example:

- Temperature and ice-cream sales usually have positive covariance.
- Temperature and hot-chocolate sales usually have negative covariance.

3.5.4 Why Covariance Matters in ML

Covariance is important in machine learning because it tells us which features are related. It also shows up everywhere:

- Gaussian distributions,
- PCA (Principal Component Analysis),
- Kalman filters,
- Gaussian Processes,
- and basically every model that wants to look mathematically sophisticated.

If variance is how much a random variable is chaotic by itself, covariance is how much two random variables are chaotic *together*.

3.6 Frequentist vs Bayesian: Two Religions, One Dataset

Now we arrive at one of the most dramatic arguments in probability theory.

Both frequentists and Bayesians use probability. Both use math. Both use integrals. Both suffer.

But they interpret probability in very different ways.

3.6.1 Frequentist View: Probability as Long-Run Frequency

In the frequentist world:

- probability is defined by repeating an experiment many times,
- parameters are fixed constants (even if we do not know them).

A frequentist thinks like this:

"If I repeated this experiment 1 million times, what fraction of the time would this happen?"

Figure 7: Me after listening to the millionth frequentist vs bayseian argument

So if we say a coin has probability 0.5 of landing heads, the frequentist interpretation is:

"If I flip the coin many times, about half the flips will be heads."

This is clean and objective. But it does not directly answer the kind of question humans naturally ask.

3.6.2 Bayesian View: Probability as Belief

In the Bayesian world:

- probability measures uncertainty or belief,
- parameters are treated as random variables.

A Bayesian thinks like this:

"Given the evidence I have seen, how confident am I that something is true?"

For example, a Bayesian might say:

"Given my data, I believe there is a 90% chance the coin is biased."

The key difference is that Bayesians allow probability statements about unknown parameters.

3.6.3 Why This Matters and why was it in the class lecture

Frequentists treat uncertainty as something that disappears if you repeat experiments infinitely.

Bayesians treat uncertainty as something that exists because we do not know everything. Which is very relatable, because none of us know what we are doing.

Machine learning loves Bayesian thinking because ML is not just about making predictions. It is about making predictions *under uncertainty*.

In real life we rarely have perfect data. We have noisy measurements, missing information, and small datasets. So we want models that do not just output an answer, but also output confidence.

Bayesian methods give us a framework for doing exactly that.

3.6.4 Bayes in One Sentence

The Bayesian philosophy is:

Start with a belief, observe data, and update your belief.

This update is done using Bayes' theorem:

$$p(\theta \mid x) = \frac{p(x \mid \theta)p(\theta)}{p(x)}.$$

This equation says:

- Start with a prior belief $p(\theta)$.
- See evidence through the likelihood $p(x \mid \theta)$.
- Combine them to get an updated belief called the posterior $p(\theta \mid x)$.

3.6.5 Why Bayes is Useful in ML

Bayesian thinking is useful because it allows:

- incorporating prior knowledge,
- quantifying uncertainty,
- updating beliefs as new data arrives,
- controlling model complexity (priors can prevent overfitting).

Many machine learning methods secretly have Bayesian interpretations. For example:

- **MLE** chooses the parameter that best explains the data:

$$\hat{\theta}_{\text{MLE}} = \arg\max_{\theta} p(x \mid \theta).$$

- **MAP** chooses the parameter that best explains the data *and* respects the prior:

$$\hat{\theta}_{\text{MAP}} = \arg\max_{\theta} p(\theta \mid x).$$

In simple words:

MLE trusts the data completely. MAP trusts the data, but keeps one eye on common sense.

Bayesian thinking is especially important in high-stakes systems—medicine, finance, self-driving cars—where being wrong is bad, but being confidently wrong is catastrophic.

So yes, machine learning loves Bayes, because Bayes does not just predict the future. It tells us how uncertain we are about that prediction.

And in a universe that plays dice, uncertainty is the only honest output.

4 The Devil is in the Details (and Apparently Also in My Grades)

If you made it this far, *uurah*. You have survived the philosophy, the metaphors, and my questionable humor.

Now we reach the part where probability theory stops being poetic and starts being extremely specific. This section focuses on **random variables** and the discrete probability distributions that show up so often in machine learning that they might as well be recurring characters in a Netflix series.

Most of these distributions were introduced briefly in Lecture 1, but they were explained properly in Lecture 2. So think of this section as the official "character introduction" before they become the main cast.

4.1 Random Variables: they are variables that are random OMG

Figure 8: Me as I write the heading

A **random variable** is not random in the sense of "unpredictable vibes". It is simply a function that maps outcomes of a random experiment into numbers.

Formally, a random variable is a mapping:

$$X : \Omega \to \mathbb{R},$$

where Ω is the sample space.

In plain English: a random variable is how we convert randomness into something we can do math with.

For example, consider the random experiment of tossing a coin. The outcome of the experiment is not a number, it is a word: {Heads, Tails}. That is not very useful if we want to compute averages, variances, or probabilities.

So we define a random variable X such that:

$$X = \begin{cases} 1 & \text{if the outcome is Heads} \\ 0 & \text{if the outcome is Tails} \end{cases}$$

Now the randomness of the coin toss is encoded into a number. Instead of saying "the coin landed heads", we say "$X = 1$".

Once we do this, we can start asking meaningful mathematical questions:

- What is the probability that $X = 1$?
- What is the expected value $\mathbb{E}[X]$?
- How much does X vary (variance)?

So a random variable is not the randomness itself—it is the *numerical representation* of randomness. It is the greates example of how lazy mathematicians are. (Not really they pretty smart don't beat me up pls)

4.1.1 Discrete vs Continuous Random Variables

Random variables usually come in two flavors:

- **Discrete:** takes countable values (0,1,2,...).
- **Continuous:** takes values in an interval (real numbers).

In this section we focus on **discrete** random variables, since they dominate early ML probability models.

4.2 Bernoulli Random Variable: The Binary Icon

The **Bernoulli** distribution is the simplest discrete distribution. It models a single yes/no event.

$$X \sim \text{Bernoulli}(p)$$

PMF:

$$P(X = x) = p^x(1-p)^{1-x}, \qquad x \in \{0, 1\}.$$

Interpretation:

- $X = 1$ means success (heads, spam, disease, pass the class, etc.)
- $X = 0$ means failure (tails, not spam, no disease, fail the class, etc.)

Expectation and variance:

$$\mathbb{E}[X] = p, \qquad \text{Var}(X) = p(1-p).$$

4.2.1 Why Bernoulli Matters in ML

Bernoulli is the backbone of binary classification. If you use logistic regression, you are essentially modeling:

$$P(Y = 1 \mid X = x) = \sigma(w^T x),$$

and then assuming the output label Y is Bernoulli.

So Bernoulli is the official probability distribution of:

"yes or no, pass or fail, cat or not cat."

4.3 Binomial Random Variable: Bernoulli's Older Sibling

If Bernoulli is one trial, then **Binomial** is many Bernoulli trials.

$$X \sim \text{Binomial}(n, p)$$

Meaning: X counts the number of successes in n independent Bernoulli trials.

PMF:

$$P(X = k) = \binom{n}{k} p^k (1-p)^{n-k}, \qquad k = 0, 1, 2, \ldots, n.$$

Expectation and variance:

$$\mathbb{E}[X] = np, \qquad \text{Var}(X) = np(1-p).$$

4.3.1 ML Intuition

Binomial shows up in situations like:

- "How many users click on an ad out of n users?"
- "How many packets are successfully received out of n transmissions?"
- "How many correct predictions out of n test samples?"

So Binomial is basically the distribution of counting wins in a fixed number of attempts.

4.4 Geometric Random Variable: How Long Until Success?

The **Geometric** distribution models the number of trials needed until the first success.

$$X \sim \text{Geometric}(p)$$

PMF (one common convention):

$$P(X = k) = (1 - p)^{k-1} p, \qquad k = 1, 2, 3, \ldots$$

Interpretation:

"How many tries until something finally works?"

Expectation and variance:

$$\mathbb{E}[X] = \frac{1}{p}, \qquad \text{Var}(X) = \frac{1 - p}{p^2}.$$

4.4.1 Why Geometric Matters

Geometric distributions appear naturally in:

- communication systems (retransmissions until packet success),
- randomized algorithms,
- waiting-time models,
- rejection sampling.

Also, it models the average number of times you refresh your inbox until your advisor replies.

4.5 Negative Binomial: Geometric, But With More Suffering

The **Negative Binomial** distribution generalizes the geometric distribution.
Instead of asking:

"How long until the first success?"

it asks:

"How long until the r-th success?"

$$X \sim \text{NegBin}(r, p)$$

PMF:

$$P(X = k) = \binom{k-1}{r-1} p^r (1-p)^{k-r}, \qquad k = r, r+1, r+2, \dots$$

Expectation and variance:

$$\mathbb{E}[X] = \frac{r}{p}, \qquad \text{Var}(X) = \frac{r(1-p)}{p^2}.$$

4.5.1 ML Intuition

Negative binomial is useful when:

- we count how long until we get r "positive outcomes",
- we model over-dispersed count data,
- we do probabilistic modeling of event arrivals.

In other words, it models repeated attempts until you achieve enough wins.

4.6 Hypergeometric: Sampling Without Replacement

Binomial assumes sampling **with replacement** (independent trials). But real life sometimes does not let you reset the universe after each sample.

Hypergeometric models sampling **without replacement**.

Suppose:

- population size is N,
- number of successes in population is K,
- we sample n items without replacement.

Then:

$$X \sim \text{Hypergeometric}(N, K, n)$$

PMF:

$$P(X = k) = \frac{\binom{K}{k}\binom{N-K}{n-k}}{\binom{N}{n}}.$$

4.6.1 ML Intuition

Hypergeometric appears when:

- selecting a subset of data without replacement,
- analyzing biased sampling,
- drawing training samples from a finite dataset.

A simple example: if a dataset has 100 spam emails and 900 normal emails, and you sample 50 emails without replacement, hypergeometric models how many spams you might pick.

4.7 Categorical Random Variable: The Multiclass Celebrity

Bernoulli is binary classification. But ML is not always binary. Sometimes the output is one of many labels: cat, dog, horse, or "whatever the hell this is".

The **Categorical** distribution models a random variable that can take one of K categories.

$$X \sim \text{Categorical}(p_1, \ldots, p_K)$$

PMF:

$$P(X = i) = p_i, \qquad i \in \{1, 2, \ldots, K\}$$

where:

$$\sum_{i=1}^{K} p_i = 1.$$

4.7.1 ML Intuition: Softmax and Classification

In neural networks, the softmax layer outputs:

$$p_i = \frac{e^{z_i}}{\sum_{j=1}^{K} e^{z_j}}$$

These p_i are interpreted as probabilities for a categorical distribution.

So when we say:

"the model predicts class probabilities"

we are literally talking about a categorical distribution.

4.8 Multinomial: Binomial's Multiclass Upgrade

Binomial counts successes in n Bernoulli trials.

Multinomial counts outcomes in n categorical trials.

$$(X_1, \ldots, X_K) \sim \text{Multinomial}(n, p_1, \ldots, p_K)$$

PMF:

$$P(X_1 = x_1, \ldots, X_K = x_K) = \frac{n!}{x_1! x_2! \cdots x_K!} \prod_{i=1}^{K} p_i^{x_i},$$

where:

$$\sum_{i=1}^{K} x_i = n.$$

4.8.1 ML Intuition

Multinomial appears in:

- word counts in documents (bag-of-words models),
- Naive Bayes text classification,
- topic modeling,
- language modeling.

Basically, if you count how many times each class appears, multinomial is your friend.

4.9 Priors and Posteriors: Bayesian DLC Pack

Now comes the Bayesian upgrade.

In Bayesian machine learning we treat unknown parameters as random variables. This means we can assign them **prior distributions**, update them with data, and obtain **posterior distributions**.

The main reason this is useful is that it gives uncertainty estimates, not just point estimates.

4.9.1 Beta Prior for Bernoulli/Binomial

For Bernoulli or Binomial models, the parameter p is a probability. So p must lie between 0 and 1.

The most common prior for p is the **Beta distribution**:

$$p \sim \text{Beta}(\alpha, \beta).$$

The reason Beta is popular is because it is a **conjugate prior** for Bernoulli/Binomial likelihoods. Meaning: if the prior is Beta and the likelihood is Binomial, the posterior is also Beta.

If we observe k successes and $n - k$ failures, then:

$$p(p \mid x) = \text{Beta}(\alpha + k, \beta + n - k).$$

In other words, the posterior is the prior plus the data counts.

This is one of the most satisfying results in probability theory because it feels like math is actually being cooperative.

4.9.2 Dirichlet Prior for Categorical/Multinomial

For categorical distributions, we have a probability vector:

$$(p_1, \ldots, p_K)$$

The conjugate prior is the **Dirichlet distribution**:

$$(p_1, \ldots, p_K) \sim \text{Dirichlet}(\alpha_1, \ldots, \alpha_K).$$

If we observe counts $x_1, \ldots, x_K$, the posterior becomes:

$$(p_1, \ldots, p_K \mid x) \sim \text{Dirichlet}(\alpha_1 + x_1, \ldots, \alpha_K + x_K).$$

Again: prior counts + observed counts = posterior counts.

Bayesian probability is basically "updating your beliefs by counting things". Which is exactly how humans behave, except humans also panic.

4.10 Why These Random Variables Matter in ML

These distributions are not just homework problems. They show up everywhere:

- Bernoulli and Binomial: binary classification, A/B testing, click-through prediction.
- Categorical and Multinomial: multiclass classification, NLP, language modeling.
- Geometric and Negative Binomial: waiting-time models, retransmissions, event arrival counts.
- Hypergeometric: sampling without replacement, dataset bias, finite population effects.

Most importantly, these distributions form the building blocks of probabilistic modeling. Once you understand them, Bayesian inference and ML theory become much less terrifying.

So yes, the devil is in the details.

And unfortunately, so are my grades.

4.11 Cheat Sheet: Which Distribution to Use When

At this point, the names Bernoulli, Binomial, Geometric, etc. may all start blending together. So here is a survival guide.

If you cannot remember the PMF, at least remember when to use the distribution.

Distribution	**What it models**	**Typical ML/Real-world use**
Bernoulli(p)	One yes/no outcome	Binary classification, logistic regression
Binomial(n, p)	# of successes in n trials	A/B testing, click prediction, accuracy cou
Geometric(p)	Trials until first success	Retransmissions, waiting-time problems
Negative Binomial(r, p)	Trials until r successes	Count modeling, event-arrival problems
Hypergeometric(N, K, n)	Sampling without replacement	Dataset sampling, finite population analysi
Categorical($p_1, \ldots, p_K$)	One outcome among K classes	Multiclass classification, softmax outputs
Multinomial($n, p_1, \ldots, p_K$)	Counts of K outcomes in n trials	NLP word counts, Naive Bayes, topic mode

Table 1: Cheat sheet for common discrete random variables and when to use them.

The fastest way to choose the correct distribution is to ask:

- Am I counting *successes* (Binomial), or counting *trials until success* (Geometric/NegBin)?
- Do I have only two outcomes (Bernoulli), or multiple categories (Categorical/Multinomial)?
- Am I sampling with replacement (Binomial) or without replacement (Hypergeometric)?

If you answer these questions correctly, probability theory becomes much less painful.