|  |  |
| --- | --- |
|  |  |
| To: | ADAN7430 |
| From: | Prof. Parravano |
| Date: | September 10, 2025 |
| Re: | Week 3 Memo: Modeling Choices, Sampling Methods, and the Role of Randomness in Machine Learning |

As we continue building the foundations of machine learning, it’s important to pause and reflect on the choices we make—not just in *how* we fit models, but in *what kinds of models* we choose, *how* we evaluate them, and *how* we think about uncertainty and noise.

This memo introduces three crucial ideas:

1. **Parametric vs. non-parametric models** — and how this choice shapes what a model can (and cannot) learn from data.
2. **Sampling methods** — including the bootstrap and cross-validation, which help us estimate uncertainty and evaluate performance.
3. **Randomness and noise** — the unavoidable backdrop to any modeling task, and the reason why no model is ever perfect.

Together, these concepts help us move from simply *fitting* models to *reasoning* about them with clarity, skepticism, and creativity.

# Parametric vs. Non-Parametric Models

Every time we build a machine learning model, we are making decisions—not just about the data and the algorithm, but about the shape of the relationship we believe exists between inputs and outputs. These decisions are often implicit, but they carry real consequences. They determine how flexible our model can be, how much data it needs, how we interpret its results, and how it will behave when confronted with new information. One of the most fundamental of these decisions is whether to use a **parametric** or **non-parametric** model.

This distinction isn’t just a technical one—it reflects a philosophical stance on the nature of the problem we’re trying to solve. Are we assuming a specific form for the relationship between variables? Or are we letting the data speak for itself, even if it means giving up some structure or interpretability? Let’s explore both sides of this modeling spectrum.

## Parametric Models: Structure First, Data Second

**Parametric models** begin with a strong assumption: we believe the relationship between inputs and outputs follows a specific, pre-defined form. In this world, the learning task becomes one of estimating the parameters of that form based on the data we observe.

Think of it like deciding in advance that the relationship we’re modeling is a straight line, a parabola, or a logistic curve. We’re not asking the model to discover the shape of the relationship from scratch—we’re telling it what shape to assume, and then asking it to find the best-fitting version of that shape.

The most familiar example is **linear regression**, where we assume the output is a linear combination of the input features ​, plus some error term. No matter how much data we collect, we’re fitting exactly parameters: one for each feature, plus an intercept.[[1]](#footnote-1)

Because the structure is fixed in advance, parametric models[[2]](#footnote-2) tend to be:

* **Data-efficient**: They can perform well even with relatively small datasets, because they’re not trying to learn very many parameters.
* **Interpretable**: Each parameter has a clear meaning—typically representing the marginal effect of a specific input variable.
* **Fast and stable**: Estimating parameters can often be done using simple matrix operations or closed-form solutions.

But that structure also comes at a cost. If the true relationship between inputs and outputs doesn’t match the form we’ve assumed, a parametric model may miss important patterns. In other words, the simplicity of parametric models can lead to bias—the model systematically overlooks complex behavior in the data because it was never capable of representing that behavior in the first place. In short, the model may **underfit** the data, failing to recognize patterns that fall outside the structure it was designed to express.

Parametric models aren’t limited to straight lines. Logistic regression is another example: it models binary outcomes as a function of a linear combination of inputs passed through the logistic function. Naive Bayes classifiers, Poisson regression, and many time series models (like ARIMA) are also parametric. In all these cases, the modeler chooses a mathematical structure first, and the algorithm focuses on estimating the parameters that best align that structure with the observed data.

In practice, parametric models are often the **first step** in modeling. They offer a principled way to summarize relationships, generate interpretable coefficients, and make fast predictions. And they provide a useful baseline—if a parametric model performs poorly, it may be a sign that we need to reach for more flexible tools.

### Neural Networks: A Parametric Model That Doesn’t Behave Like One

Strictly speaking, neural networks are parametric models. They consist of a finite set of weights and biases, and learning involves estimating these parameters through gradient-based optimization. But in practice, neural networks behave very differently from traditional parametric models.

For one, they tend to have a massive number of parameters—far more than the number of input features, and sometimes even more than the number of training examples. This gives them an extraordinary capacity to model complex, nonlinear relationships, but it also makes them:

* **Data-hungry**: They often require large datasets to generalize well.
* **Computationally intensive**: Training can take hours, days, or even weeks.
* **Opaque**: The parameters are not directly interpretable in the way they are in, say, linear regression

Neural networks are also adaptive. Adding more layers or neurons increases the number of parameters, allowing the model to learn increasingly complex functions. This makes them powerful—but also prone to overfitting, and highly sensitive to tuning.

In other words, neural networks break the mold of what we typically think of as “parametric.” They are parametric in name, but not in spirit. Their behavior is much closer to that of non-parametric models, because their effective complexity grows with the data and architecture.

Neural networks blur the line between parametric and non-parametric modeling—  
offering the expressive power of the latter with the formal structure of the former.

For this reason, we often treat them as a special case: powerful, flexible, and increasingly dominant—but with unique challenges that deserve separate treatment.

## Non-Parametric Models: Flexibility First, Structure Later

Where parametric models begin with a fixed structure and a small number of parameters, **non-parametric models** flip that logic. They make **few or no assumptions** about the form of the relationship between inputs and outputs. Instead, they let the data determine the shape of the model.

In a non-parametric framework, the model’s **complexity can grow as the amount of data increases**. This makes these models especially powerful for capturing rich, nonlinear, and context-specific patterns that parametric models would miss or oversimplify. You can think of non-parametric modeling as asking:

“Rather than assuming the world follows a particular equation, what can we learn just by observing it closely and responding to what we see?”

This flexibility makes non-parametric models well-suited for:

* Tasks where the true relationship is unknown or complicated,
* Applications where interpretability is less important than accuracy,
* Situations with large, rich datasets that support flexible pattern discovery.

Here are some key examples:

* **k-Nearest Neighbors (k-NN):** This is one of the most intuitive non-parametric methods. To make a prediction for a new observation, the model finds the most similar points in the training data (its “neighbors”) and averages their outputs. There are no parameters to estimate in advance—just a memory of the data and a rule for comparing points.
* **Decision Trees:** Trees partition the feature space into regions by recursively splitting on the most informative features. They don’t assume any particular functional form. Instead, they build up structure dynamically as they observe the data. The more complex the tree, the more subtle the patterns it can capture.
* **Kernel Methods (e.g., Kernel Density Estimation, SVMs):** Kernel methods rely on similarity between points, using kernel functions to build flexible estimates of distributions or decision boundaries. Like k-NN, they don’t presuppose a global structure—only local relationships between points.
* **Gaussian Processes:** These are non-parametric models used for regression and classification. They model distributions over functions, where complexity grows with the data. GPs are highly expressive and provide built-in measures of uncertainty.

The main strength of non-parametric models is their **adaptability**. They can model intricate patterns without requiring a predefined formula. But that same flexibility comes with trade-offs:

* **Interpretability**: Non-parametric models are often harder to explain. They typically don't produce simple coefficients or summary statistics.
* **Computation**: Many non-parametric methods require storing or searching through the entire training dataset to make predictions, which can be slow and memory-intensive.
* **Overfitting**: With enough flexibility, these models can fit *anything*—including noise. Without proper regularization or validation, they risk losing generalizability.

The key takeaway is this:

* **Parametric models** are compact, interpretable, and efficient—but potentially biased.
* **Non-parametric models** are flexible, adaptive, and expressive—but potentially overfit.

Good data scientists know how to use both—and when to reach for one over the other.

# Sampling Methods: Bootstrap and Cross-Validation

In machine learning, we often work with a single dataset—but we want to answer questions that reach far beyond it:

* How well will this model perform on new, unseen data?
* How confident should we be in the patterns we've found?
* Could the results have turned out differently with a slightly different sample?

These questions are subtle, but critical. They touch on the **robustness** of our models, the **reliability** of our conclusions, and the **limits of what we can know** from finite data.

In an ideal world, we would gather fresh data every time we wanted to check our model’s performance or assess uncertainty. But in practice, data is finite—and often expensive or time-consuming to collect. So instead, we turn to a powerful idea: **sampling from what we already have.**

Sampling methods like the **bootstrap** and **cross-validation** help us simulate what might have happened had we collected different data. They give us insight into both the **stability** of our estimates and the **generalization** of our models—without requiring new data collection.

## The Bootstrap: Estimating Uncertainty from a Single Sample

The bootstrap is a resampling-based method for estimating the variability of a statistic—such as a model coefficient, prediction, or error rate.

Here’s how it works:

1. From your original dataset of size , draw a new sample of size with replacement. Some observations will appear more than once; others not at all.
2. Fit your model to this resampled dataset.
3. Repeat the process many times (e.g., 1,000 times), each time recording the result.
4. Use the distribution of these results to estimate standard errors, confidence intervals, or other uncertainty measures.

The key idea is this: even though we have only one dataset, we can simulate variability in our estimates by pretending we had access to many similar datasets, each drawn from the same underlying population.

### Why this matters

In many modeling situations, theoretical formulas for variance (especially under non-standard conditions) are hard to derive—or unreliable. The bootstrap gives us a flexible, data-driven way to understand how sensitive our results are to the particular data we happened to observe.

You’ll encounter the bootstrap repeatedly in this course—not just as a technique, but as a way of thinking: What if the data had been just a little different? Would the model still tell the same story?

### Why it works

At first glance, the bootstrap feels like statistical sleight of hand. We take a single dataset—possibly the only one we’ll ever get—and then generate hundreds or thousands of synthetic datasets by sampling from it *with replacement*. We act as if these pseudo-datasets are meaningful stand-ins for new data drawn from the population. And somehow, the results we get—standard errors, confidence intervals, bias estimates—are often remarkably good.

It’s fair to pause here and ask: *How can this possibly work? Aren’t we just recycling the same data over and over again?*

The key insight is this: If your original sample is representative of the population, then the empirical distribution of your sample is a good proxy for the true distribution.

That’s what the bootstrap exploits. By sampling with replacement from the original data, we’re treating it as an approximation of the population distribution. We're using it to simulate the process of drawing new samples *as if* we had access to the full population. It’s not perfect—but when the sample is large and reasonably representative, it’s often surprisingly accurate

The bootstrap was introduced by Bradley Efron in 1979. At the time, statistical inference was dominated by theoretical approximations and rigid assumptions—most of which relied on known distributions (like the normal or t-distribution) and often broke down in real-world situations.

Efron’s insight was radical: instead of relying on analytical formulas or textbook distributions, why not let the data itself define the uncertainty?

By treating the data as an empirical distribution and simulating the process of resampling, the bootstrap offered a new way to:

* Estimate standard errors without needing derivatives or asymptotic theory,
* Construct confidence intervals without assuming normality,
* Analyze statistics for which no formulas existed.

Today, the bootstrap is one of the most widely used tools in both statistics and machine learning—particularly for complex models, small samples, and nonstandard situations where theory fails or is hard to apply.

The bootstrap is not a free lunch. It assumes your original sample is a good stand-in for the population, and it works best when the sample size is reasonably large. If the original data is biased or unrepresentative, the bootstrap can reinforce that bias. And for some statistics (like extreme quantiles), it may not perform as well.

Still, its flexibility, generality, and conceptual simplicity make it one of the most powerful and democratic tools in applied statistics.

When we bootstrap, we’re not pretending we know more than we do—We’re just squeezing more insight from the data we already have.

## Cross-Validation: Evaluating Model Performance on Unseen Data

While the bootstrap helps us understand how variable our estimates are, cross-validation helps us understand how well our model will perform on data it hasn’t seen yet. Cross-validation answers a practical question: If I train my model on this dataset, how well can I expect it to perform on new, unseen observations?

The most common version is k-fold cross-validation:

1. Split your dataset into equally-sized folds.
2. For each fold:
   1. Train the model on the other folds.
   2. Evaluate it on the held-out fold.
3. After repeating this process times, compute the average performance across folds.

This gives us a robust estimate of out-of-sample error, which is crucial for:

* Comparing different models fairly,
* Tuning hyperparameters (like tree depth, regularization strength, or number of neighbors),
* Detecting overfitting.

Cross-validation can also be modified:

* **Leave-one-out cross-validation (LOOCV)** trains on n-1 data points and tests on 1, repeating times.
* **Stratified cross-validation** ensures balanced representation of classes in each fold.
* **Repeated cross-validation runs the procedure multiple times with different fold partitions.**

In short, training error alone is a poor indicator of model quality. Cross-validation helps us estimate how the model will generalize, which is ultimately what we care about in any predictive task.

## Model Selection vs Model Evaluation

Once we've trained a model and thought carefully about uncertainty and generalization, a natural next step is to compare models and decide which one to use. But there are two distinct goals we must keep in mind:

1. **Model selection**: Choosing the best model (or best set of parameters) from a set of candidates.
2. **Model evaluation**: Estimating how well the final, chosen model will perform on truly unseen data.

Both steps are essential—but they’re often confused or accidentally blended. Worse, if we don’t handle them properly, we can end up overestimating how well our model performs in the real world.

Let’s unpack the difference—and how sampling methods help us get both steps right.

### Model Selection: Choosing the Best from a Set

Imagine you're comparing several types of models: a linear model, a decision tree, a k-nearest neighbors classifier. Or maybe you're tuning the hyperparameters of a single model—like the number of neighbors in k-NN or the depth of a decision tree.

Your goal is to pick the best one based on how it performs on your data.

This is where cross-validation shines. You split your data into training and validation folds, train each model on one subset, and evaluate it on the other. You repeat this process across folds and compare average performance.

This allows you to select:

* The best-performing model class,
* The best hyperparameter configuration,
* Or the best combination of features.

Key point: The validation data used here is not true test data. It’s used to guide choices—it informs the model-building process. So we can’t use it again to claim how well the model generalizes.

### Model Evaluation: Estimating Generalization Performance

Once you’ve selected your final model, you want to report how well it performs. This is model evaluation—and it should be done using data that has *not* been used for training *or* selection.

This could be:

* A separate holdout test set,
* A nested cross-validation scheme, where an outer loop reserves folds for final evaluation,
* Or, in some cases, cross-validation with honest reporting, where you acknowledge the role of selection in your performance estimate.

If you use the same validation data to both choose your model and evaluate it, your performance estimate will be biased upward. It will look better than it actually is on new data—because the model has already seen and adapted to the validation set.

### Grid Search: Meaning and Importance

In practice, choosing the best model often means choosing the best combination of hyperparameters—the settings that define how a model behaves before it even sees the data. These might include:

* The number of neighbors in a k-NN classifier,
* The maximum depth of a decision tree,
* The regularization strength in ridge or lasso regression,
* The learning rate or architecture of a neural network.

These are not parameters the model learns from the data—they’re design choices we set manually. So how do we know which values will work best?

Enter grid search: Grid search is a brute-force, but powerful, approach to hyperparameter tuning. Here’s how it works:

1. You define a grid of possible values for each hyperparameter.  
   For example, you might try regularization strengths of [0.01, 0.1, 1, 10] and tree depths of [3, 5, 7, 9].
2. For each combination of values in the grid, you:
   * Train the model using cross-validation,
   * Record the average performance score (e.g., accuracy, mean squared error).
3. You select the combination that performs best across folds.

The result is a model that has been tuned empirically—not based on intuition or guesswork, but on how it performs under simulated out-of-sample conditions.

Grid search lets the data speak, not just about *which model* is best—but about *how* that model should be configured.

Without tuning, even a strong model can perform poorly. Grid search ensures you're comparing well-optimized versions of each model—so your model selection process is fair.

That said, grid search can be computationally expensive, especially when:

* You’re tuning multiple hyperparameters,
* You have many candidate values for each,
* Or your model takes a long time to train.

In those cases, alternatives like random search or Bayesian optimization may be more efficient. But grid search remains a foundational concept—especially for understanding the link between hyperparameter tuning, cross-validation, and model selection.

Think of grid search as a systematic experiment: It asks, “What if we try all reasonable configurations—and let performance decide?”

## A Sampling-Aware Workflow

Putting it all together, here’s how sampling methods help us keep things clean:

1. Split off a test set (or prepare an outer cross-validation loop) and put it aside.
2. Use cross-validation on the remaining data for model selection—comparing models, tuning parameters, etc.
3. Once the best model is chosen, go back and use the held-out test set to estimate its true generalization error.

This helps you avoid “data leakage” between the stages and ensures that your performance estimates are realistic.

Model selection is about *choosing wisely*.  
Model evaluation is about *estimating honestly*.

Both rely on sampling. Both require discipline. And both are essential if we want our models to be useful beyond the dataset we happened to start with.

# Randomness and Noise: The Limits of What We Can Predict

As we wrap up this week’s topics—model complexity, selection, sampling, and evaluation—it’s worth stepping back and acknowledging a deeper reality: **some parts of the world are just unpredictable.**

In machine learning, we talk often about signal: patterns in the data that can be learned, modeled, and used to make accurate predictions. But every dataset also contains something else: **noise.**

### What is Randomness?

Randomness in machine learning refers to unexplained variability—variation in outcomes that can’t be attributed to the input features available to the model. This randomness can come from many sources:

* **Measurement error**: Sensors drift, surveys contain typos, and timestamps get rounded.
* **Omitted variables**: A dataset can never capture every causal factor. The world is always more complex than our model.
* **Natural variability**: Biological systems, human behavior, and economic conditions often have inherent randomness.
* **Stochastic processes**: Some systems are fundamentally probabilistic in nature (e.g., quantum physics, coin flips).

This randomness appears in the model as residual error—differences between the model’s predictions and the actual observed outcomes. In linear regression, it’s represented as; in any supervised learning model, it’s the part of y that’s not captured by .

### Noise vs Signal

A key goal of any learning algorithm is to separate signal from noise—to learn meaningful patterns while ignoring what’s random or idiosyncratic.

This is easier said than done.

* **Underfitting** occurs when we miss the signal—our model is too simple to capture real structure.
* **Overfitting** occurs when we mistake noise for signal—our model is so flexible that it captures patterns that aren’t real.

That’s why cross-validation and regularization are so important: they help us avoid chasing noise and stay focused on what generalizes.

But even the best models will never have zero error—not because they’re bad, but because some of the error is irreducible. There will always be noise we can’t explain.

### Randomness in Learning Itself

Randomness doesn’t just show up in the data—it shows up in the learning process, too.

* Stochastic gradient descent adds randomness to optimization.
* Random forest algorithms use randomness in feature selection and bootstrapping.
* Train/test splits, bootstrap samples, and cross-validation folds all rely on randomness.

Far from being a nuisance, this randomness is often a strength. It helps us generalize better, explore model space, and quantify uncertainty.

In modern machine learning, randomness is not the opposite of rigor.It’s part of how we reason honestly about data and models.

### Embracing Uncertainty

In the end, randomness reminds us of something deeply important: **machine learning is not about certainty—it’s about approximation**. It’s about building models that are useful, not perfect. It’s about being transparent about what we know, and humble about what we don’t.

Every dataset is a window. Every model is a lens. And every prediction carries with it a shadow of uncertainty that we must learn to see, measure, and respect.

1. You can, of course, choose to not include an intercept , in which case you would be fitting parameters. [↑](#footnote-ref-1)
2. Many classical models fall under this taxonomy:

   Logistic regression: Predicts probabilities of binary outcomes by passing a linear combination of inputs through the logistic function.

   Naive Bayes: Assumes strong independence between features and estimates conditional probabilities from data.

   Poisson regression: Models count data with the assumption that the outcome follows a Poisson distribution conditioned on inputs.

   ARIMA models: Widely used in time series forecasting, these assume a fixed form for trends, seasonality, and noise.

   Linear discriminant analysis (LDA): Assumes normally distributed features within classes, allowing for probabilistic classification. [↑](#footnote-ref-2)