Notes for Understanding

Machine Learning

A personal learning textbook

Ioanna Stamou

Contents

Introduction

Imagine you have a robot friend.

You give the robot a task:

"Look at these examples, learn the pattern, and make predictions."

This is machine learning.

Machine learning = finding patterns in data.

Example:

You give a model 1000 houses with their *price*, *size*, and *number of rooms*. The model learns the relationship between these features and the price. Then, when you give it a *new* house, it predicts the price.

That's all.

> **Key Idea**
>
> Machine learning is not magic. It is a systematic way to:
>
> 1. collect data,
> 2. find patterns,
> 3. use those patterns to make predictions or decisions.

The rest of these notes are about:

- the **statistics** you need to talk precisely about patterns,
- the **Python tools** you will use to work with data,
- the **machine learning algorithms** that actually learn from examples.

1

Statistics You Need for Machine Learning

In this chapter we will develop the basic statistical ideas that appear everywhere in machine learning: random variables, distributions, expectation, variance, covariance, correlation, and a few key theorems that explain why learning from data can work.

1.1 Types of Variables

Numerical Variables

Numerical variables take quantitative values.

- **Continuous:** $x \in \mathbb{R}$ Examples: height, temperature, house price.
- **Discrete:** $x \in \mathbb{Z}$ Examples: number of rooms, number of customers.

Categorical Variables

Categorical variables represent non-numerical classes.

- **Nominal (unordered)** Examples: color, country, type of food.
- **Ordinal (ordered)** Examples: rating levels, education level.

Important: Machine learning models expect numerical inputs. Categorical variables must be encoded (one-hot, label encoding, etc.).

1.2 Descriptive Statistics

1.2.1 Mean

The mean (average) of n observations is:

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

1.2.2 Median

The median is the middle value of the sorted data. It is more robust to outliers than the mean.

1.2.3 Variance

Variance measures how far the values are spread from the mean:

$$\mathrm{Var}(X) = \frac{1}{n}\sum_{i=1}^{n} (x_i - \mu)^2.$$

1.2.4 Standard Deviation

Standard deviation is the square root of the variance:

$$\sigma = \sqrt{\mathrm{Var}(X)}.$$

It measures the typical deviation from the mean.

1.2.5 Covariance

Variance tells us how a single variable varies. Covariance tells us how *two* variables vary *together*.

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

Interpretation:

- Positive covariance: when X is above its mean, Y tends to be above its mean.
- Negative covariance: when X is above its mean, Y tends to be below its mean.
- Zero covariance: no linear relationship.

1.3 Correlation

Correlation measures the strength of linear dependence between two variables.

- Range: $[-1, 1]$
- Indicates how two variables move together

$$\rho_{XY} = \frac{\mathrm{Cov}(X,Y)}{\sigma_X \sigma_Y}.$$

- $+1$: perfect positive relationship
- 0: no linear relationship
- -1: perfect negative relationship

Machine learning uses correlation to select and weight relevant features.

1.4 Probability Theory Basics

Machine learning models predict probabilities. Even regression models assume probabilistic noise.

Key ideas:

- **Joint probabilities**: describe combined events.
- **Conditional probabilities**: describe relationships.
- **Marginal probabilities**: describe overall tendencies.

This logic becomes the backbone of all ML algorithms.

1.4.1 Joint Probability

For events A and B:

$$P(A,B) = P(A \cap B).$$

1.4.2 Conditional Probability

$$P(A \mid B) = \frac{P(A,B)}{P(B)}.$$

1.4.3 Marginal Probability

Discrete:

$$P(A) = \sum_{b} P(A, b).$$

Continuous:

$$P(A) = \int P(A, b)\, db.$$

Probability quantifies uncertainty. Conditional probability expresses relationships between events.

1.5 Bayes' Theorem: Updating Beliefs from Data

1.5.1 Formula

Bayes' theorem describes how we update our belief about an event A after observing some evidence B:

$$P(A \mid B) = \frac{P(B \mid A)\, P(A)}{P(B)}.$$

What does this mean?

Bayes' theorem tells us:

Posterior = Likelihood $\times$ Prior / Evidence

More explicitly:

Key Idea

$$P(A \mid B) = \underbrace{P(B \mid A)}_{\text{How well } A \text{ explains the data}} \quad \underbrace{P(A)}_{\text{What we believed before}} \Big/ \underbrace{P(B)}_{\text{How expected the data is overall}}.$$

- $P(A)$ [**prior**] : What we believed before seeing any data.
- $P(B \mid A)$ [**likelihood**] : How compatible the observed data B is with our hypothesis A.
- $P(B)$ [**evidence**] : The probability of seeing the data under all possible explanations.
- $P(A \mid B)$ [**posterior**] : Our updated belief about A after incorporating evidence B.

Intuition

Bayes' theorem answers the question:

"Given what I just observed, how should I update what I believe?"

It combines two ideas:

1. **What we believed before** (the prior)
2. **How surprising the data is under each hypothesis** (the likelihood)

The evidence $P(B)$ ensures everything stays a valid probability distribution (sums to 1).

Why this matters in Machine Learning

Bayes' theorem is the foundation of: Naive Bayes classifier, Bayesian Linear Regression, Bayesian Neural Networks, All probabilistic and generative models.

It formalizes **learning from data**. We start with a belief (prior), then see data, and update our belief (posterior).

1.6 Likelihood

The concept of **likelihood** answers a very important question in statistics and machine learning:

Key Idea

"If this parameter θ were true, how well would it explain the data I observed?"

This is different from asking about the probability of future events. Likelihood is specifically about evaluating how plausible a parameter is, given the data we already have.

Definition

For a dataset $x = (x_1, x_2, \ldots, x_n)$ and a model with parameter θ, the likelihood is defined as:

$$L(\theta \mid x) = P(x \mid \theta)$$

This expression should be read as:

"Given θ, what is the probability of observing the dataset x?"

Interpretation

- $x = (x_1, x_2, \ldots, x_n)$: the observed data. These values are **fixed**. We already saw them.
- θ: the parameter of the model. This is **unknown** and is what we want to estimate.
- $P(x \mid \theta)$: the model's prediction about how likely the data is, if θ were the true parameter.

Important Distinction: Probability vs. Likelihood

- **Probability:** θ is fixed, data is random. (Used to predict future observations.)
- **Likelihood:** data is fixed, θ is variable. (Used to evaluate or estimate parameters.)

This is a subtle but essential idea in statistical learning.

The Likelihood Function

Note

The likelihood function is simply the probability of the data, viewed as a function of the parameter:

$$L(x; \theta) = P(x \mid \theta)$$

It tells us how much support the data gives to each possible value of θ.

"Better-fitting parameters give higher likelihood."

Crucially: likelihood is *not* a probability distribution over θ. It does **not** integrate to 1. It is just a scoring function that says which values of θ explain the data best.

The i.i.d. Assumption

In most machine learning models, the data points are assumed to be:

- **independent:** knowing one datapoint tells nothing about another,
- **identically distributed:** all datapoints come from the same distribution.

Under this assumption:

$$L(x \mid \theta) = \prod_{i=1}^{n} P(x_i \mid \theta)$$

Why a product?

Because the joint probability of independent events equals the product of their individual probabilities.

This is a key simplification that makes likelihood computation possible for large datasets.

Summary

- Likelihood measures how well a model with parameter θ explains the observed data.
- It is the central tool for parameter estimation.
- Almost all ML algorithms rely on likelihood:
  - Linear Regression
  - Logistic Regression
  - Neural Networks
  - Decision Trees and Random Forests (implicitly)
  - XGBoost
- Training a model often means:

$$\text{maximize likelihood} \quad \text{or} \quad \text{minimize negative log-likelihood.}$$

In short, likelihood connects your model to your data. It is the mathematical backbone of statistical learning.

1.7 Probability Distributions: Shape of Data

A probability distribution describes **how data behaves**. It tells us which values are likely, which are rare, and what kind of uncertainty or randomness we should expect in the data.

Machine learning models often assume certain distributions because they determine the model's behavior and how it handles noise.

1.7.1 Discrete Distributions

Discrete distributions describe random variables that take values from a finite or countable set (e.g., 0, 1, 2, 3, ...).

Bernoulli Distribution A Bernoulli variable represents a single binary outcome:

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

Interpretation:

- Models "success or failure" events.
- Examples: coin flip, email is spam/not spam, churning/not churning customer.
- Parameter p is the probability of success.

Binomial Distribution The binomial distribution models the number of successes in n independent Bernoulli trials:

$$P(X = k) = \binom{n}{k} p^k (1-p)^{n-k}.$$

Interpretation:

- Repeating the same experiment n times.
- Examples: number of heads in n coin flips, number of defective items in a batch.
- Gives the distribution of the *count of successes*.

Poisson Distribution The Poisson distribution models the number of events occurring in a fixed time or space interval:

$$P(X = k) = \frac{\lambda^k e^{-\lambda}}{k!}.$$

Interpretation:

- Used for rare events happening unpredictably.
- Examples: number of incoming calls per minute, number of website hits per second, number of accidents per day.

- Parameter λ is both the mean and the variance.

1.7.2 Continuous Distributions

Continuous distributions describe variables that take real-number values (e.g., height, temperature, time).

Gaussian (Normal Distribution)

$$f(x) = \frac{1}{\sqrt{2\pi\sigma^2}} \exp\left(-\frac{(x-\mu)^2}{2\sigma^2}\right)$$

Interpretation:

- The most important distribution in statistics.
- Data clusters around a mean μ with spread σ.
- Many ML models assume Gaussian noise.
- Examples: measurement error, natural variations, height.

Exponential Distribution

$$f(x) = \lambda e^{-\lambda x}$$

Interpretation:

- Models the time between independent random events.
- Examples: time until next earthquake, time until customer arrival.
- Memoryless property: past values do not change future probability.

Multivariate Gaussian

$$f(x) = \frac{1}{(2\pi)^{d/2}|\Sigma|^{1/2}} \exp\left(-\frac{1}{2}(x-\mu)^T\Sigma^{-1}(x-\mu)\right)$$

Interpretation:

- A Gaussian distribution in multiple dimensions.
- μ is a mean vector, Σ is a covariance matrix.

- Models correlated variables.
- Used heavily in:
  - PCA (Principal Component Analysis)
  - Gaussian Mixture Models (GMMs)
  - Bayesian inference

Why Distributions Matter in Machine Learning

Different distributions encode different assumptions about uncertainty:

- Bernoulli / Binomial → binary or count data
- Gaussian → continuous data with natural noise
- Poisson → rare event counts
- Exponential → waiting times
- Multivariate Gaussian → correlated continuous data

Choosing an appropriate distribution helps us build models that accurately reflect the data-generating process.

1.8 Statistical Inference: Learning Parameters From Data

Statistical inference is about using data to learn the best values for the parameters of a model. In machine learning, almost every algorithm can be understood as an inference method.

We focus on two fundamental ideas:

- **Maximum Likelihood Estimation (MLE)**
- **Maximum A Posteriori Estimation (MAP)**

These appear throughout regression, classification, probabilistic models, and even deep learning.

1.8.1 Maximum Likelihood Estimation (MLE)

MLE answers a simple question:

Key Idea

"Which parameter value makes the observed data the most likely?"

If $x = (x_1, \ldots, x_n)$ is the observed data and θ is an unknown parameter, the MLE estimate is [1]:

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

where

$$L(\theta \mid x) = P(x \mid \theta)$$

is the likelihood of the data under the parameter θ.

Why do we take the log?

Because products of probabilities become sums:

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

This is numerically stable and makes optimization easier.

Intuition

- The data x is considered fixed.
- The parameter θ varies.
- We choose the θ that best *explains* the data.

This idea lies behind most ML algorithms: Linear regression (least squares), Logistic regression, Naive Bayes, Neural networks (via cross-entropy).

All of them optimize a likelihood or its negative log.

1.8.2 Maximum A Posteriori Estimation (MAP)

MAP estimation is the Bayesian version of learning parameters.

Instead of asking:

[1] max → what is the highest value. argmax → where is the highest value

"Which parameter makes the data most likely?"

we ask:

Key Idea

"Which parameter is most plausible, given the data and my prior beliefs?"

The MAP estimate is:

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

Using Bayes' theorem:

$$P(\theta \mid x) = \frac{P(x \mid \theta) P(\theta)}{P(x)}$$

Since $P(x)$ does not depend on θ, we ignore it in maximization:

$$\hat{\theta}_{\text{MAP}} = \arg\max_{\theta} \left[\log P(x \mid \theta) + \log P(\theta)\right]$$

Interpretation

- **MLE**: only cares about the data.
- **MAP**: cares about data *and* prior knowledge.

Example

Examples of priors:
Priors express what we believe about the parameter θ before seeing data. In machine learning, they often appear naturally as regularization.
1. Prior: "Parameters should be small"
This corresponds to a Gaussian prior:

$$P(\theta) \propto \exp\left(-\theta^2\right),$$

which leads to the penalty θ^2 after taking $-\log$. This is exactly **L2 regularization**.
2. Prior: "Many parameters should be zero"
This corresponds to a Laplace prior:

$$P(\theta) \propto \exp(-|\theta|),$$

which leads to the penalty $|\theta|$. This is **L1 regularization**, which encourages sparsity.

Connection between MLE and MAP

$$\text{Posterior} \propto \text{Likelihood} \times \text{Prior}$$

- If the prior is flat (uninformative), MAP = MLE.
- If the prior is strong, MAP pulls the estimate toward the prior belief.

MAP can be seen as:

$$\text{MAP} = \text{MLE} + \text{regularization}.$$

This is why many ML algorithms with regularization (ridge, lasso, weight decay) are actually MAP estimators in disguise.

1.9 Linear Regression: Statistical Interpretation

Linear regression is one of the simplest and most fundamental models in machine learning. It describes a relationship between input features and a continuous output.

Model

We assume the data is generated according to:

$$y = X\beta + \varepsilon$$

- X is the matrix of input features.
- β is a vector of unknown parameters.
- ε is random noise (unpredictable variation).

The key probabilistic assumption:

$$\varepsilon \sim \mathcal{N}(0, \sigma^2 I)$$

This means:

- noise is Gaussian,
- noise has zero mean,
- noise is independent with constant variance.

With this assumption, each y_i is normally distributed around $X_i\beta$.

Least Squares Estimation

The most common way to estimate β is to minimize the sum of squared errors:

$$\hat{\beta} = \arg\min_{\beta} \|y - X\beta\|^2$$

This objective measures how well the model's predictions match the data.

The minimizer has a closed-form solution:

$$\hat{\beta} = (X^T X)^{-1} X^T y$$

This solution exists when $X^T X$ is invertible.

Probabilistic Interpretation

Ordinary Least Squares (OLS) is not only a geometric method—it is also a probabilistic one.

If the noise is Gaussian, then the likelihood of observing y given β is:

$$P(y \mid X, \beta) = \frac{1}{(2\pi\sigma^2)^{n/2}} \exp\left(-\frac{1}{2\sigma^2}\|y - X\beta\|^2\right)$$

Maximizing this likelihood is equivalent to minimizing the squared error.

Thus:

$$\text{OLS} = \text{MLE under Gaussian noise.}$$

Interpretation:

- Linear regression assumes data is generated as a linear trend + Gaussian noise.
- Fitting the model means choosing β that makes the data most likely.

- This gives a deep statistical foundation to the method.

Linear regression is therefore both:

- a geometric projection problem,
- a probabilistic parameter estimation problem.

Key Idea

What is linear regression?
Linear regression is a simple model that tries to describe a relationship between input variables (features) and an output variable by fitting a straight line (or a hyperplane in higher dimensions).

- You give the model input data X and outputs y.
- The model tries to find parameters β so that the prediction $X\beta$ is close to y.
- The assumption is that the underlying true relationship is approximately *linear*.

Geometric view:

- The model projects y onto the space spanned by the columns of X.
- The result is the "best-fitting line" in the least-squares sense.

Statistical view:

- The data is assumed to follow
$$y = X\beta + \varepsilon,$$
where ε is random noise with Gaussian distribution.
- Fitting the model means choosing β that makes the observed data most probable.

Linear regression is a fundamental tool because it is easy to interpret, computationally efficient, and the foundation of more advanced models.

1.10 Logistic Regression: Probabilistic Classification

Linear regression predicts a continuous outcome. Logistic regression adapts the idea to predict *probabilities* for binary outcomes (0 or 1).

Model

We model the probability of the outcome being 1 as:

$$P(y = 1 \mid x) = \sigma(w^T x)$$

using the sigmoid function:

$$\sigma(z) = \frac{1}{1 + e^{-z}}.$$

Properties of the sigmoid:

- outputs values between 0 and 1,
- smoothly increases, S-shaped,
- interpretable as a probability.

This means logistic regression predicts:

"How likely is it that $y = 1$ given the input x?"

Log-Likelihood

Assume each data point (x_i, y_i) is drawn independently. The likelihood of the entire dataset under parameters w is:

$$\ell(w) = \sum_{i=1}^{n} \left[y_i \log \sigma(w^T x_i) + (1 - y_i) \log\big(1 - \sigma(w^T x_i)\big) \right]$$

This is the log-likelihood of the Bernoulli model.

Training = maximize the log-likelihood.

This chooses the w that makes the observed labels most probable.

Connection to Cross-Entropy Loss

Instead of maximizing $\ell(w)$, we minimize its negative:

$$\mathscr{L}(w) = -\ell(w)$$

This is the **cross-entropy loss**, the most widely used loss in classification and neural networks.

Interpretation:

- If the model assigns a high probability to the true label, the loss is small.
- If the model assigns a low probability to the true label, the loss is large.

Thus training logistic regression is about making correct labels more probable.

Summary

Logistic regression is:

- a probabilistic model for binary classification,
- trained via maximum likelihood,
- equivalent to minimizing cross-entropy,
- foundational for neural networks, softmax classifiers, and deep learning.

It takes the linear model $w^T x$ and turns it into a probability through the sigmoid function.

1.11 Bias–Variance Tradeoff: Why Models Fail

When we train a model, we want its predictions $\hat{f}(x)$ to be close to the true output y. A key question in machine learning is:

Why do models make errors, even after training?

The answer is given by the bias–variance decomposition:

$$\mathbb{E}[(y-\hat{f}(x))^2] = \underbrace{\text{Bias}[\hat{f}(x)]^2}_{\text{model too simple}} + \underbrace{\text{Var}[\hat{f}(x)]}_{\text{model too complex}} + \sigma^2_{\text{noise}}.$$

This formula says that prediction error comes from *three* different sources.

1. Bias: Error from Wrong Assumptions

Bias measures how far the model's average prediction is from the true relationship.

- High bias means the model is too simple.
- It cannot capture the true pattern.
- It makes the same mistakes no matter how much data we have.

Examples:

- Using a straight line to fit a quadratic curve.
- A linear model for data with strong nonlinear structure.

This is called **underfitting**.

2. Variance: Error from Sensitivity to Data

Variance measures how much predictions change if we train on a different dataset.

- High variance means the model is too flexible.
- It memorizes random noise in the training data.
- Small changes in the data lead to large changes in the model.

Examples:

- A decision tree that grows too deep.
- A neural network trained with too few data points.

This is called **overfitting**.

3. Irreducible Noise

$$\sigma^2_{\text{noise}}$$

This is randomness in the data that no model can ever explain.

Examples:

- Measurement error.
- Natural variability in human behavior.
- Random fluctuations in physical or economic systems.

Even a perfect model cannot reduce this part of the error.

Why This Matters

The bias–variance tradeoff explains:

- **Underfitting**: high bias, low variance.
- **Overfitting**: low bias, high variance.
- **Regularization**: reduces variance by simplifying the model.
- **Cross-validation**: detects overfitting by testing on unseen data.
- **Ensembles**: average predictions to reduce variance.

> **The goal of learning is to find a model with a good balance:** low bias *and* low variance.

Example-Recap

Understanding Probability, Likelihood, MLE, MAP

We have a coin, and we toss it $n = 10$ times. We observe:

$$x = \text{H H T H H T H H H T} \qquad \Rightarrow \qquad k = 7 \text{ heads, } 3 \text{ tails.}$$

Let $\theta = P(\text{Heads})$ be the (unknown) probability that the coin lands Heads.

1. Probability: parameter fixed, data random

Here we assume that the coin bias *is already known*. Suppose:

$$\theta = 0.6.$$

Then we ask:

"If the coin has $\theta = 0.6$, what is the probability of getting exactly 7 heads in 10 tosses?"

This is a binomial probability:

$$P(X = 7 \mid \theta = 0.6) = \binom{10}{7} (0.6)^7 (0.4)^3.$$

Let us break down each part:

- $\binom{10}{7}$ counts how many sequences of 7 heads and 3 tails exist.
- $(0.6)^7$ is the probability of getting Heads 7 times.
- $(0.4)^3$ is the probability of getting Tails 3 times.

A numerical evaluation:

$$\binom{10}{7} = 120, \quad (0.6)^7 \approx 0.02799, \quad (0.4)^3 = 0.064.$$

Thus:

$$P(X = 7 \mid \theta = 0.6) = 120 \times 0.02799 \times 0.064 \approx 0.215.$$

So if $\theta = 0.6$, this data occurs with probability about 21.5%.
Here:

$$\theta \text{ fixed}, \quad X \text{ random}.$$

This is the usual "forward" direction in probability.

2. Likelihood: data fixed, parameter varies

Now we switch perspectives completely. We already saw the data:

$$k = 7.$$

We treat the data as fixed. Now we ask:

"For this fixed data (7 Heads), how plausible is each possible value of θ?"

This is the likelihood:

$$L(\theta \mid x) = P(x \mid \theta) = \binom{10}{7} \theta^7 (1-\theta)^3.$$

Key idea:

- The combinatorial factor $\binom{10}{7}$ does **not** depend on θ.
- So the likelihood is shaped by:
$$\theta^7 (1-\theta)^3.$$

Interpretation:

- Data is fixed ($k = 7$).
- θ varies between 0 and 1.
- The likelihood is a *score function*, not a probability distribution over θ.

Probability vs Likelihood:

$$\text{Probability: fix } \theta, \text{ vary data.}$$

$$\text{Likelihood: fix data, vary } \theta.$$

3. Bayesian View: Prior, Likelihood, Posterior

The Bayesian view adds one more ingredient: a **prior** about θ.
Instead of treating θ as an unknown but fixed number, we treat it as a random variable with its own distribution $P(\theta)$ (our belief *before* seeing the data).
Bayes' theorem connects:

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

In this formula:

- $P(\theta)$ is the **prior**: what we believe about θ before seeing data.
- $P(x \mid \theta)$ is the **likelihood**: how well each θ explains the data.
- $P(\theta \mid x)$ is the **posterior**: updated belief about θ after seeing data.
- $P(x)$ is the **evidence**: a normalizing constant to make probabilities sum to 1.

We often write this in proportional form:

$$P(\theta \mid x) \propto P(x \mid \theta)\, P(\theta).$$

So in words:

Posterior = Likelihood $\times$ Prior (up to a constant).

The likelihood from step 2 is what *transforms* the prior into the posterior.

4. Maximum Likelihood Estimation (MLE)

MLE asks a simple question:

"Which value of θ makes the observed data (7 heads) most likely?"

We take the likelihood:

$$L(\theta \mid x) = \theta^7 (1-\theta)^3,$$

and choose the θ that makes this expression as large as possible:

$$\hat{\theta}_{\text{MLE}} = \arg\max_{\theta} \theta^7 (1-\theta)^3.$$

For the binomial model, the answer is always:

$$\hat{\theta}_{\text{MLE}} = \frac{k}{n}.$$

Here:

$$\hat{\theta}_{\text{MLE}} = \frac{7}{10} = 0.7.$$

Meaning: MLE simply uses the data. It says:

"You saw heads 70% of the time. So my best estimate is $\theta = 0.7$."

5. Bayesian Inference and MAP

The Bayesian approach adds one more ingredient: a **prior belief** about θ.

Suppose our prior is:

$$\theta \sim \text{Beta}(5,5),$$

which expresses the idea that the coin is probably close to fair ($\theta \approx 0.5$).[a]
After seeing the data (7 heads out of 10), the posterior becomes:

$$\theta \mid x \sim \text{Beta}(12,8).$$

MAP estimate

MAP chooses the value of θ that is most probable under the posterior:

$$\hat{\theta}_{\text{MAP}} = \frac{\alpha' - 1}{\alpha' + \beta' - 2} = \frac{12-1}{12+8-2} = \frac{11}{18} \approx 0.611.$$

Meaning:

- The data pushes the estimate toward 0.7.
- The prior pulls the estimate toward 0.5.

So MAP becomes:

$$\text{something between 0.5 and 0.7,}$$

which here is ≈ 0.611.
MAP = "**update your belief by mixing prior + data**".
If the prior were weak (e.g. Beta(1,1)), MAP would be almost the same as MLE.

6. Summary

- **Probability** $P(x \mid \theta)$: given a model, what data is expected?
- **Likelihood** $L(\theta \mid x)$: given data, how plausible is each parameter?
- **MLE**: maximize likelihood (data only).
- **Bayesian Update**: Posterior $\propto$ Likelihood $\times$ Prior.
- **MAP**: maximize posterior = MLE + prior information.

[a] **About the Beta distribution and the meaning of α and β.**
The *Beta distribution* is a probability distribution defined on the interval $[0,1]$. It is widely used as a prior for unknown probabilities such as the bias θ of a coin, because it can represent many shapes: peaked, flat, symmetric, or skewed. It is written

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

where the two parameters α and β control the shape.
A simple and intuitive interpretation is:

- α behaves like a "prior count of heads" $+1$,
- β behaves like a "prior count of tails" $+1$.

Thus, a Beta(α,β) prior acts as if, *before observing any real data*, we had already seen:

$$\alpha-1 \text{ imaginary heads}, \qquad \beta-1 \text{ imaginary tails}.$$

For example, the prior

$$\text{Beta}(5,5)$$

acts as if we had already seen 4 heads and 4 tails. This expresses a belief that the coin is probably close to fair ($\theta \approx 0.5$), but we are not completely certain.

The Beta distribution is important because it is a *conjugate prior* for the binomial model: if the prior is Beta(α,β) and we observe k heads and $n-k$ tails, then the posterior is also a Beta distribution:

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

This makes Bayesian updating very simple: we just add the new counts to the old ones. In our example:

$$\text{Beta}(5,5) \xrightarrow{k=7} \text{Beta}(12,8).$$

The updated parameters $\alpha' = 12$ and $\beta' = 8$ summarize how our prior belief and the new data combine to form the posterior.

2

Python for Machine Learning

2.1 Supervised Data

Supervised learning refers to a setting where we observe a collection of examples, each consisting of:

- an **input** (also called *features*), and
- an **output** (also called *label* or *target*).

The goal is to learn a rule that maps inputs to outputs so that we can make accurate predictions for new, unseen data.

Key Idea

"We show the model many examples of input → correct output,
and we ask it to learn the pattern so it can predict the output for new inputs."

2.1.1 The Form of Supervised Data

We are given a dataset consisting of n labeled examples:

$$D = \{(x_1, y_1), (x_2, y_2), \ldots, (x_n, y_n)\}.$$

Each pair (x_i, y_i) contains:

$$x_i \in \mathbb{R}^d \quad \text{(a feature vector with } d \text{ components)},$$

$$y_i \quad \text{(a target value we want to predict)}.$$

The target variable may be:

- **Real-valued** ($y_i \in \mathbb{R}$) → **regression**

- **Categorical** ($y_i \in \{0,1\}$ or $y_i \in \{1,\ldots,K\}$) $\rightarrow$ **classification**

In supervised learning, the dataset includes both inputs and correct outputs. This is what distinguishes it from **unsupervised** learning, where we only have the inputs x_i and no labels.

📈 Example

Example: House Price Dataset

- $x_i = (\texttt{size}, \texttt{number\_of\_rooms}, \texttt{age\_of\_house}) \in \mathbb{R}^3$
- $y_i = \texttt{price}$ of the house

Here:

- Each (x_i, y_i) is one house in the dataset.
- Supervised learning asks: *"Given the features of a new house, can we predict its price?"*
- Unsupervised learning would only have x_i and would ask: *"Do houses naturally form groups or patterns?"*

2.1.2 The Objective of Supervised Learning

We assume there exists an unknown function f^* that relates inputs to outputs:

$$y = f^*(x) + \varepsilon,$$

where:

- f^* is the **true** (but unknown) relationship,
- ε is random **noise** or unpredictable variation.

A machine learning model builds an approximation $f_\theta(x)$, parameterized by θ:

$$\hat{y} = f_\theta(x).$$

The goal of supervised learning is to find parameters θ such that $f_\theta(x)$ approximates $f^*(x)$ as closely as possible.

Key Idea

Think of f^* vs. f_θ:

- f^*: the ideal rule that nature is using (we never see it exactly).
- f_θ: our model's best attempt to imitate this rule using data.

Training = adjusting θ so that $f_\theta(x)$ behaves like $f^*(x)$ on the data we have.

2.1.3 Regression vs. Classification

Supervised learning problems fall into two main categories.

Regression The target is continuous:

$$y \in \mathbb{R}.$$

Examples: predicting house prices, estimating temperature, forecasting demand or sales.

Classification The target is discrete:

$$y \in \{0,1\} \quad \text{(binary)}, \qquad y \in \{1,\dots,K\} \quad \text{(multi-class)}.$$

Examples: spam vs. not spam, image classification (cat / dog / car / ...), medical diagnosis (disease present / not present).

Note

Same input, different task:
If we use patient data (age, blood pressure, etc.) to predict *blood sugar level* $\Rightarrow$ regression.
If we use the same data to predict *diabetic vs. not diabetic* $\Rightarrow$ classification.
The data can be the same, but the **type of target** changes the learning problem.

2.1.4 i.i.d. Assumption

Supervised learning typically assumes that the samples

$$(x_1,y_1),\dots,(x_n,y_n)$$

are **i.i.d.** (independent and identically distributed):

- **Independent:** one data point does not influence another (e.g. one customer's purchase does not change another customer's features).
- **Identically distributed:** all samples come from the same underlying distribution (same population, same data-generating process).

This assumption underlies key ML methods such as: empirical risk minimization, maximum likelihood estimation, and cross-validation.

Key Idea

Why i.i.d. is important:
If the training and test data come from the same distribution, then good performance on the training set (with proper validation) tells us something about performance on future data.
If this is not true (distribution shift), models can fail even if they seem to work well during training.

2.1.5 Features and Target Variables

A feature vector can be written as:

$$x_i = (x_{i1}, x_{i2}, \ldots, x_{id})^\top.$$

Types of features include:

- **Numerical:** real numbers (e.g. age, income, temperature)
- **Categorical:** categories (e.g. country, color) encoded using one-hot or label encoding
- **Ordinal:** ordered categories (e.g. low / medium / high)
- **Boolean:** true/false (0 or 1)
- **Derived:** new features created from raw data (e.g. BMI from height and weight, room_per_person, etc.)

Example

Example: Feature Vector for a House

$$x_i = (\text{size}, \text{rooms}, \text{age}, \text{distance\_to\_center})^\top = (85, 3, 20, 4.5)^\top$$

If we also know the house price y_i, the pair (x_i, y_i) becomes one supervised example in our

dataset.

A machine learning model uses these features to learn patterns that generalize well to new, unseen data.

2.1.6 The Learning Goal

Given supervised data, we seek a function $f_\theta(x)$ that:

- fits the training data well,
- is not overly complex (to avoid overfitting),
- performs well on new, unseen data.

Key Idea

Core tension in supervised learning:

- If the model is too simple $\rightarrow$ it cannot capture the pattern (underfitting).
- If the model is too complex $\rightarrow$ it memorizes noise (overfitting).

The art of machine learning is to find a model and training procedure that *captures the signal* in the data while ignoring as much noise as possible.

2.2 Loss Function and Risk Minimization

In supervised learning, a model makes predictions $\hat{y} = f_\theta(x)$. To measure how good these predictions are, we use a **loss function**. The loss function tells us how far the prediction is from the true value.

A **loss function** is a mathematical rule that measures the error between a model prediction $\hat{y} = f_\theta(x)$ and the true target y:

$$L(y, \hat{y}) \in \mathbb{R}_{\geq 0}.$$

A small loss means the prediction is good; a large loss means the prediction is bad.

Different machine learning tasks use different loss functions.

Key Idea

Key Idea: Loss Function
A loss function assigns a number (the "error") to each prediction.

$$\text{Small loss} \Rightarrow \text{good prediction}, \qquad \text{Large loss} \Rightarrow \text{bad prediction}.$$

Training a model means **choosing parameters θ that minimize the total loss**.

2.2.1 Loss for a Single Example

For one training example (x_i, y_i), the loss measures how different the prediction $\hat{y} = f_\theta(x_i)$ is from the true target:

$$L(y_i, f_\theta(x_i)).$$

A small loss means a good prediction; a large loss means a bad one.

- **Squared loss (regression):**
$$L(y, \hat{y}) = (y - \hat{y})^2.$$
  - Penalizes large errors heavily.
  - Smooth and easy to optimize.
  - Leads to the mean squared error (MSE).
- **Absolute loss:**
$$L(y, \hat{y}) = |y - \hat{y}|.$$
  - More robust to outliers.
- **Cross-entropy loss (classification):**
$$L(y, \hat{p}) = -\left[\, y \log(\hat{p}) + (1 - y) \log(1 - \hat{p}) \,\right],$$
  where $\hat{p} = f_\theta(x)$ is the predicted probability of class 1.
  - Standard loss for binary classification.
  - Equivalent to negative log-likelihood of the Bernoulli model.

Different loss functions lead to different behaviours and different models.

2.2.2 Empirical Risk: Total Loss on the Dataset

Since we cannot compute the true risk we approximate it using training dataset. The total loss over all training samples is called the **empirical risk**:

$$R_{\text{emp}}(\theta) = \frac{1}{n}\sum_{i=1}^{n} \ell\big(y_i, f_\theta(x_i)\big).$$

This is simply the average error the model makes on the training data.

Note

Empirical risk is the quantity we can actually compute, because we only have access to the training dataset.

2.2.3 Risk Minimization

The goal of training is to find parameters θ that minimize this risk:

$$\hat{\theta} = \arg\min_{\theta} R_{\text{emp}}(\theta).$$

This principle is called **Empirical Risk Minimization (ERM)**.

Key Idea

ERM = "Choose the model that fits the training data best."

All learning algorithms — linear regression, logistic regression, neural networks, SVMs, decision trees — can be seen as risk minimizers under an appropriate loss function.

Note

• In theory, we care about the model's error on the **true** data distribution:

$$R(\theta) = \mathbb{E}_{(x,y)\sim P_{\text{data}}}\big[\ell(y, f_\theta(x))\big].$$

But we cannot compute $R(\theta)$ because the true distribution is unknown. So we approximate it with the empirical risk computed from the dataset.

• Training a model means:

$$\text{minimize empirical risk} \quad \text{to approximate} \quad \text{minimizing true risk.}$$

Generalization techniques (regularization, cross-validation) prevent overfitting.

2.2.4 Example: Linear Regression Loss

Example

For linear regression, predictions are:

$$\hat{y}_i = w^T x_i.$$

The loss is squared error:

$$\ell_i = (y_i - w^T x_i)^2.$$

The empirical risk is:

$$R_{\text{emp}}(w) = \frac{1}{n} \sum_{i=1}^{n} (y_i - w^T x_i)^2.$$

Minimizing this leads to the ordinary least-squares solution.

2.2.5 Example: Logistic Regression Loss

Example

For binary classification:

$$\hat{p}_i = \sigma(w^T x_i).$$

The loss is cross-entropy:

$$\ell_i = -\big[y_i \log \hat{p}_i + (1 - y_i) \log(1 - \hat{p}_i)\big].$$

Minimizing empirical risk under this loss is equivalent to **maximizing the log-likelihood**.

2.2.6 Regularization: Controlling Model Complexity

Minimizing empirical risk alone often leads to **overfitting**: the model fits the training data too closely and performs poorly on new data.

To prevent this, we add a **regularization term** that penalizes overly complex models:

$$\hat{\theta} = \arg\min_{\theta} \Big[R_{\text{emp}}(\theta) + \lambda\, \Omega(\theta)\Big].$$

- $\lambda > 0$ controls the strength of regularization (large λ = stronger penalty).
- $\Omega(\theta)$ is a measure of model complexity.

Common choices:

- **L2 regularization (Ridge):**
$$\Omega(\theta) = \|\theta\|_2^2$$
  - discourages large parameter values,
  - smooths the model,
  - corresponds to a Gaussian prior in MAP.
- **L1 regularization (Lasso):**
$$\Omega(\theta) = \|\theta\|_1$$
  - encourages sparsity (many parameters become exactly zero),
  - performs implicit feature selection,
  - corresponds to a Laplace prior in MAP.

Regularization balances two goals:

- **fit the data** (low empirical risk),
- **remain simple enough** to generalize (low complexity).

2.2.7 Summarize

To sum up, until now:

- A **loss function** measures prediction error for a single example.
- **Risk** extends this idea over the whole data distribution.
- Because the true distribution is unknown, we minimize the **empirical risk** on the training set.
- To avoid overfitting, we add a **regularization term** that penalizes overly complex models.

Altogether, most machine learning algorithms solve an optimization problem of the form:

$$\theta^* = \arg\min_{\theta} \left[\frac{1}{n} \sum_{i=1}^{n} L(y_i, \hat{y}_i) + \lambda\, \Omega(\theta) \right].$$

This framework is the foundation of nearly all modern machine learning methods, from linear and logistic regression to support vector machines and deep learning.

2.3 Train / Validation / Test

When we train a model, we minimize the empirical risk on the training set:

$$R_{\text{emp}}(\theta) = \frac{1}{n}\sum_{i=1}^{n} L\big(y_i, f_\theta(x_i)\big).$$

However, a model can achieve very low loss on the training data and still perform poorly on new data. This is **overfitting**: the model memorizes the training data instead of learning general patterns.

Key Idea

Why split the data?
To detect and prevent overfitting, we must evaluate the model on data it has *never* seen during training.

The Three Splits

1. Training Set

- Used to learn the model parameters θ.
- Examples:
  - Linear regression learns weights β.
  - Random Forest learns tree splits.
  - Neural networks learn millions of parameters via gradient descent.
- The model updates its parameters **only** using this data.

Note

Training = fitting the parameters. The model tries to minimize the loss on this set.

2. Validation Set The validation set is **not** used to update the model's parameters. Instead, it evaluates how design choices affect performance.

Used for:

- hyperparameter tuning (learning rate, regularization λ, tree depth, ...)
- selecting the best model architecture

- early stopping in neural networks

Parameters vs. Hyperparameters

- **Parameters** learned directly from training data (weights, coefficients, thresholds)
- **Hyperparameters** chosen using the validation set (learning rate, regularization strength, number of layers, max depth)

Example

Example: Try several models with different λ values. Choose the one with the lowest *validation loss*.

3. Test Set The test set provides the **final, unbiased** evaluation.

- Used only once, after all training and hyperparameter tuning is complete.
- Simulates real-world, future data.
- Must never be used to make design decisions.

Note

Never leak information from the test set. If the test set influences training decisions, the performance estimate becomes invalid.

Data Leakage

A crucial rule:

The test set must remain completely unseen.

Examples of leakage:

- normalizing using the full dataset instead of the training set,
- choosing hyperparameters using the test set,
- extracting features using all data (e.g., PCA on the whole dataset).

These mistakes artificially inflate performance and break generalization.

k-Fold Cross-Validation

When the dataset is small, we cannot afford to hold out a large validation set. Instead, we use **k-fold cross-validation**.

Procedure:

1. Split the dataset into k equal folds.
2. For each fold:
    - train on $k-1$ folds,
    - validate on the remaining fold.
3. Average the validation results.

Typical choices: $k = 5$ or $k = 10$.

Cross-validation is essential for:

- robust hyperparameter tuning,
- model selection,
- reducing variance in evaluation,
- avoiding overfitting on a single validation set.

Summary

- **Training set:** learn parameters.
- **Validation set:** tune hyperparameters and choose the model.
- **Test set:** final, unbiased evaluation of generalization.
- **Cross-validation:** a stable validation method when data is limited.

Splitting the data correctly is essential to ensure that a model truly generalizes and is not simply memorizing the training examples.

2.4 Python Ecosystem

Code Example

Linear Regression on the California Housing Dataset

```
from sklearn.model_selection import train_test_split
from sklearn.linear_model import LinearRegression
from sklearn.metrics import mean_squared_error
from sklearn.datasets import fetch_california_housing

# 1. Load the California housing dataset
data = fetch_california_housing()
X = data.data         # features (2D array)
y = data.target       # target (1D array)

# 2. Split the data into train and test sets
X_train, X_test, y_train, y_test = train_test_split(
    X, y, test_size=0.2, random_state=42
)

# 3. Create the model
model = LinearRegression()

# 4. Train the model
model.fit(X_train, y_train)

# 5. Predict on the test set
y_pred = model.predict(X_test)

# 6. Evaluate the model using MSE (mean squared error)
mse = mean_squared_error(y_test, y_pred)
print("MSE:", mse)
```

3

Machine Learning Algorithms

Here we will study the main learning algorithms: linear regression, logistic regression, decision trees, ensembles, and neural networks, always connecting them back to the statistics and Python tools from the previous chapters.

4

Neural Networks & Deep Learning

5

Practical ML