AMATH 445

Scientific Machine Learning

Lecture Notes

Winter 2026

Contents

Lecture 1

1.1 What is Machine Learning

Definition 1.1 *(Machine Learning).* Machine learning is decision making based on data. The algorithm improves its performance on tasks based on experience (data).

Note 1.2 *(Traditional Algorithm Workflow).*

$$\left.\begin{matrix}\text{data/info}\\ \text{recipe}\end{matrix}\right\} \rightarrow \text{traditional algorithm} \rightarrow \text{output}$$

Note 1.3 *(Machine Learning Workflow).*

$$\left.\begin{matrix}\text{data/info}\\ \text{output}\end{matrix}\right\} \rightarrow \text{ML} \rightarrow \text{recipe}$$

What, When and Why

What does ML do really well:

- Prediction: Forecasting unknown outcome (temperature).
- Representation: Finding structure in data (clustering).
- Decision making: Choosing the optimal action.
- Generation: ChatGPT, new data, image generation.

When

- Complex patterns: When relationships in data are too complex for traditional programming and we are not able to find the patterns ourselves.
- Large datasets: When there is an abundance of data that can be leveraged for learning.

Why

- It can generalize well to unseen data, making accurate predictions or decisions based on learned patterns (not memorize only).

Important: ML and Data Quality

Machine Learning is only as good as the data it is trained on. Poor quality or biased data can lead to inaccurate or unfair outcomes.

1.2 Factors Driving Popularity of ML

There are several factors driving the popularity of ML:

1. Data availability.
2. Parallel computing (GPU).
3. Algorithm & infrastructure maturity.
    - CNN, RNN.
    - Backward differentiation.

1.3 Mechanics of ML

The mechanics of ML is as follows

1. Data representation → turn things into feature
2. Pattern discovery → structure emerges from randomness
3. Pick a model
4. Training – optimization
5. Generalize → good model works on new data
6. Evolution → test on new data (unseen)

1.4 Data

1.4.1 Data Splitting

A common convention for data splitting is 70% training, 15% validation, and 15% test.

Important

TEST DATA is NEVER used in training!

1.4.2 Label Availability

This table shows the availability of features and outputs during the modeling phases versus real-world deployment.

	$F_1 \dots F_n$	output
training	known	known
validation	known	known
test	known	known
new	known	unknown

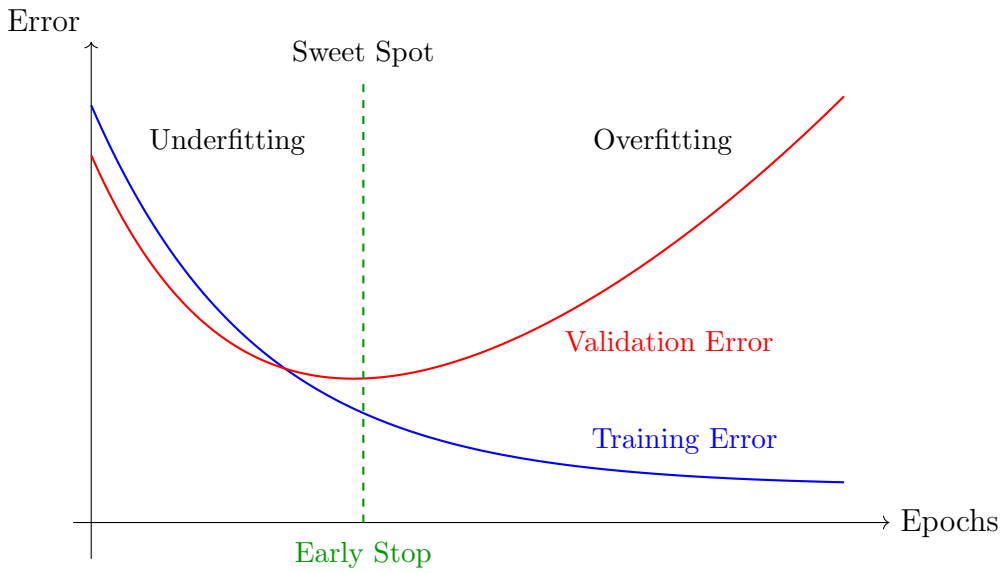
1.5 Types of Error

There are three types of error to track during model development:

- Training error: error on the training set. It measures how well the model fits the training data.
- Validation error: error on the validation set. It guides hyperparameter tuning and model selection.
- Test error: error on the test set. It provides an unbiased estimate of model performance on unseen data.

1.6 Training Dynamics

As training progresses, training and validation errors evolve differently. The figure below illustrates the typical behavior.



Note 1.4. Three key behaviors emerge from the training dynamics:

- **Underfitting**: Both training and validation errors are high. The model is too simple to capture the underlying patterns.
- **Overfitting**: Training error continues to decrease while validation error starts to increase. The model memorizes the training data but fails to generalize.
- **Early Stopping**: Stop training when the validation error begins to increase. This achieves the best generalization by stopping at the sweet spot.

1.6.1 Underfitting

A model underfits when it is too simple (high bias) to capture the underlying structure of the data. Both training and validation errors remain high. Remedies include:

- Increase model complexity.
- Add more or better features.
- Obtain more or better training data.

1.6.2 Overfitting

A model overfits when it learns the training data too closely — including its noise — and fails to generalize to new data. Training error is low but validation error is high. Remedies include:

- Get more data.
- Regularization.
- Dropout.
- Batch normalization.
- Data augmentation.
- Early stopping.

Lecture 2

2.1 Categories of Machine Learning

2.1.1 Supervised Learning

Definition 2.1 *(Supervised Learning).* In supervised learning, the model is trained on a labeled dataset to learn a map

$$f : \mathbb{R}^d \to \mathcal{Y}, \tag{2.1}$$

where $\boldsymbol{X}_i \in \mathbb{R}^d$ is the feature vector of the ith data point (with d features) and $y_i \in \mathcal{Y}$ is its label. The dataset consists of N labeled pairs

$$\{(\boldsymbol{X}_i, y_i)\}_{i=1}^N, \tag{2.2}$$

and the goal is to learn a mapping that generalizes to unseen data.

Supervised learning is used for:

- **Classification** (binary, multi-class or multi-label): Predict discrete class label.
- **Regression**: Predict continuous outputs (MSE, MAE are used commonly as loss function in regression).

2.1.2 Unsupervised Learning

Definition 2.2 *(Unsupervised Learning).* The model is trained on unlabeled dataset:

$$\{\boldsymbol{X}_i\}_{i=1}^N. \tag{2.3}$$

The goal is to learn the underlying structure or distribution $p(\boldsymbol{X})$ of the data, without any label information.

Common tasks include:

- **Clustering:** partition the data into groups of similar points,
- **Dimensionality Reduction:** learn a mapping $f : \mathbb{R}^d \to \mathbb{R}^{d'}$ with $d' \ll d$ that preserves important structure,
- **Anomaly Detection:** identify points $\boldsymbol{X}_i$ where $p(\boldsymbol{X}_i) \ll 1$.

2.1.3 Reinforcement Learning

The key elements of reinforcement learning are:

- **Agent:** The learner or decision-maker that interacts with the environment and learns to improve its behaviour over time.
- **Environment:** The external system the agent interacts with, which responds to the agent's actions and produces new states and rewards.
- **State $\boldsymbol{s}_t \in \mathcal{S}$:** The current situation of the agent in the environment, capturing all relevant information needed to make a decision.
- **Action $a_t \in \mathcal{A}$:** The choices available to the agent at each timestep, which can be discrete (e.g. legal moves in chess) or continuous (e.g. joint angles of a robot arm).
- **Reward r_t:** A feedback signal received after each action, indicating how successful that action was in progressing toward the goal.
- **Policy $\pi : \mathcal{S} \to \mathcal{A}$:** A strategy that defines the agent's actions based on the current state, which the agent refines over time to maximise cumulative reward.

Definition 2.3 *(Reinforcement Learning).* In reinforcement learning, an agent learns a policy

$$\pi : \mathcal{S} \to \mathcal{A}, \tag{2.4}$$

where $\mathcal{S}$ is the state space and $\mathcal{A}$ is the action space. At each timestep t, the agent:

1. observes state $\boldsymbol{s}_t \in \mathcal{S}$,
2. takes action $a_t \in \mathcal{A}$,
3. receives reward r_t, generating a trajectory of tuples

$$\{(\boldsymbol{s}_t,\, a_t,\, r_t,\, \boldsymbol{s}_{t+1})\}_{t=0}^{T}. \tag{2.5}$$

The goal is to learn a policy π that maximizes the cumulative discounted reward

$$R = \sum_{t=0}^{T} \gamma^t r_t, \tag{2.6}$$

where $\gamma \in [0, 1]$ is a discount factor controlling the importance of future rewards.

The state space $\mathcal{S}$ is the set of all possible situations the agent can observe. For example, in a chess game $\mathcal{S}$ is the set of all possible board configurations, while in a self-driving car $\mathcal{S}$ might encode the car's position, speed, and surrounding obstacles.

The action space $\mathcal{A}$ is the set of all possible actions the agent can take. The action space can be:

- **Discrete:** a finite set of distinct actions, $\mathcal{A} = \{a_1, a_2, \ldots, a_K\}$. For example, in a chess game $\mathcal{A}$ is the set of all legal moves.
- **Continuous:** an infinite range of actions, $\mathcal{A} \subseteq \mathbb{R}^k$. For example, in a robot arm $\mathcal{A}$ might be the set of all possible joint rotation angles.

Together, they define the full decision-making problem — the policy $\pi : \mathcal{S} \to \mathcal{A}$ maps each observed state to an action.

2.1.4 Semi-supervised and Self-supervised Learning

Definition 2.4 *(Semi-supervised Learning).* In semi-supervised learning, the model is trained on a small set of labeled data

$$\{(\boldsymbol{X}_i,\, y_i)\}_{i=1}^{N_l}, \tag{2.7}$$

combined with a large set of unlabeled data

$$\{\boldsymbol{X}_j\}_{j=1}^{N_u}, \quad N_u \gg N_l. \tag{2.8}$$

The model leverages the labeled data to learn initial patterns and then uses the unlabeled data to refine its understanding and improve generalization and performance.

Definition 2.5 *(Self-supervised Learning).* Self-supervised learning is a type of unsupervised learning in which the model generates its own pseudo-labels from the input data via a *pretext task*. Given an unlabeled dataset

$$\{\boldsymbol{X}_i\}_{i=1}^{N}, \tag{2.9}$$

the pretext task automatically constructs labeled training pairs $\{(\tilde{\boldsymbol{X}}_i,\, \hat{y}_i)\}_{i=1}^{N}$ by augmenting, masking, or transforming the original data, where $\hat{y}_i$ is the pseudo-label. The model then trains on these pairs in a supervised fashion, learning useful internal representations without requiring manual annotations.

Note 2.6 *(Pretext Task).* A pretext task is an artificial task designed to generate pseudo-labels from unlabeled data. The model does not ultimately care about solving the pretext task itself, the goal is to learn rich internal representations that can be

transferred to downstream tasks. Common approaches include:

- **Masking:** Hide part of the input and train the model to predict the missing piece (e.g., masked language modeling in BERT).
- **Transformation prediction:** Apply a known transformation (e.g., rotation by $0°, 90°, 180°, 270°$) and train the model to predict which transformation was applied.
- **Next-token prediction:** Train the model to predict the next element in a sequence given all previous elements (e.g., GPT).

Note 2.7 *(Self-supervised Learning vs. Unsupervised Learning).* Both self-supervised and unsupervised learning operate on unlabeled data $\{\boldsymbol{X}_i\}_{i=1}^N$, but they differ in mechanism:

- **Unsupervised learning** discovers structure directly from the data distribution $p(\boldsymbol{X})$ without any labels (e.g., clustering, dimensionality reduction).
- **Self-supervised learning** manufactures pseudo-labels via a pretext task and then trains in a supervised fashion on the resulting (input, pseudo-label) pairs.

In other words, unsupervised learning finds patterns without labels, while self-supervised learning creates its own labels and uses the supervised learning framework to learn representations.

Note 2.8 *(Integrating Mechanistic Models with ML).* Integrating Mechanistic Models with Machine Learning: Using physical laws or domain knowledge to inform and constrain machine learning models, enhancing their interpretability and reliability.

2.2 Supervised Learning and Classification

2.2.1 The Problem Statement of Classification

The input (feature vector) is $\boldsymbol{X} \in \mathbb{R}^d$, the output (class label) is a discrete variable $y \in \{1, 2, 3, \ldots, K\}$. Given a new $\boldsymbol{X}_{\text{new}}$, the model should predict which class it belongs to, y_{new}.

2.2.2 A Probabilistic View of Classification

We want to model $P(y = k \mid \boldsymbol{X})$, the probability of each class label given the input features, and classify by picking the most probable class. This requires tools from probability theory.

Definition 2.9 *(Conditional Probability).* The conditional probability of A given B is

$$P(A \mid B) = \frac{P(A \cap B)}{P(B)}. \tag{2.10}$$

Theorem 2.10 *(Bayes Theorem).* Since the joint probability can be written two ways,

$$P(A \cap B) = P(A \mid B)\, P(B) = P(B \mid A)\, P(A), \tag{2.11}$$

we can flip which variable is being conditioned on:

$$P(A \mid B) = \frac{P(B \mid A)\, P(A)}{P(B)}. \tag{2.12}$$

Applied to classification, where A is the class $y = k$ and B is the observed features $\boldsymbol{X}$, theorem 2.10 gives

$$P(y = k \mid \boldsymbol{X}) = \frac{P(\boldsymbol{X} \mid y = k)\, P(y = k)}{P(\boldsymbol{X})}. \tag{2.13}$$

Each term in eq. (2.13) has a name in the Bayesian vocabulary:

$$\underbrace{P(y = k \mid \boldsymbol{X})}_{\text{posterior}} = \frac{\overbrace{P(\boldsymbol{X} \mid y = k)}^{\text{class-conditional likelihood}} \cdot \overbrace{P(y = k)}^{\text{prior}}}{\underbrace{P(\boldsymbol{X})}_{\text{evidence}}}. \tag{2.14}$$

Note 2.11 *(On Bayesian Vocabulary in Classification).* The mathematics in this section, conditional probability and Bayes Theorem are universal and belongs to neither the Bayesian nor frequentist school of thought. Both camps agree on the formula $P(A \mid B) = P(A \cap B)/P(B)$. What is Bayesian is the *vocabulary* and *reasoning pattern* applied to it:

- The **math** (conditional probability, Bayes Theorem) is universal.
- The **vocabulary** (prior, posterior, likelihood) is Bayesian terminology borrowed for the classification setting.
- The **looseness** is only in calling $P(\boldsymbol{X} \mid y = k)$ a "likelihood", since $y = k$ is a class label, not a model parameter in the strict statistical sense. This is flagged explicitly in the Bayesian Vocabulary note below.

Note 2.12 *(Bayesian Vocabulary).* The terms posterior, likelihood, and prior are all conditional probabilities $P(A \mid B)$, but named by the role they play:

- **Posterior $P(y = k \mid \boldsymbol{X})$:** The class is unknown, the features $\boldsymbol{X}$ are observed. "Posterior" means *after*: our belief about the class *after* seeing the evidence. This is what we want for classification.
- **Class-conditional likelihood $P(\boldsymbol{X} \mid y = k)$:** Assumes a class k and asks how well $\boldsymbol{X}$ fits that class's distribution. It is called a "likelihood" because it plays the same role as a likelihood function: evaluating how plausible each hypothesis is given fixed data, even though $y = k$ is a class label rather than a model parameter in the strict statistical sense.
- **Prior $P(y = k)$:** The probability of class k *before* observing any features. Outside of the Bayesian context, this is just the marginal class probability — the name "prior" only has meaning in contrast with the posterior.
- **Evidence $P(\boldsymbol{X})$:** A normalizing constant that ensures the posterior sums to 1 over all classes. Since it is independent of k, it can be ignored when classifying.

Likelihood vs. Probability in STATISTICS

In statistics, **probability** and **likelihood** are the same mathematical expression viewed from opposite directions:

- **Probability** is *forward-looking*: fix the model parameters and ask "what outcomes might I see?" It is a distribution over outcomes and sums to 1.
- **Likelihood** is *backward-looking*: fix the observed data and ask "which parameters (or hypotheses) best explain what I saw?" It measures plausibility, not probability, and does *not* sum to 1.

Formally, both involve $P(\text{data} \mid \theta)$, but they differ in what is treated as the variable:

$$\underbrace{P(\text{data} \mid \theta)}_{\text{probability: fix } \theta,\ \text{vary data}} = \underbrace{L(\theta\,;\,\text{data})}_{\text{likelihood: fix data, vary } \theta} \tag{2.15}$$

Likelihood is Bayesian Classification

In the Bayesian classification context, $P(\boldsymbol{X} \mid y = k)$ is called a likelihood because $\boldsymbol{X}$ is fixed and we compare across classes k — the same reasoning pattern, even though k is a class label rather than a model parameter. In logistic regression, $L(\boldsymbol{\beta})$ is a likelihood in the strict sense — $\boldsymbol{\beta}$ is an actual model parameter.

Theorem 2.13 *(Bayes Decision Rule).* The Bayes classifier assigns an input $\boldsymbol{X}$ to the class with the largest posterior probability:

$$\hat{y}(\boldsymbol{X}) = \underset{k \in \{1,2,\ldots,K\}}{\arg\max}\ P(y = k \mid \boldsymbol{X}). \tag{2.16}$$

Since the evidence $P(\boldsymbol{X})$ is independent of k, combining theorem 2.13 and eq. (2.13) gives the following result:

Corollary 2.14 *(Bayes Classifier).* The Bayes classifier can also be expressed as

$$\hat{y}(\boldsymbol{X}) = \underset{k \in \{1,2,\ldots,K\}}{\arg\max}\ P(\boldsymbol{X} \mid y = k)\, P(y = k), \tag{2.17}$$

where $P(\boldsymbol{X} \mid y = k)$ is the class-conditional likelihood and $P(y = k)$ is the prior.

From Bayesian Classification to Practical Models

The Bayes Classifier (eq. (2.17)) tells us *what* to compute, but not *how*. To use it, we need to know $P(\boldsymbol{X} \mid y = k)$ and $P(y = k)$, which are unknown in practice. The rest of this section presents two fundamentally different strategies for making the Bayes Classifier practical:

- **Generative approach (LDA):** Explicitly model the class-conditional likelihood $P(\boldsymbol{X} \mid y = k)$ by assuming each class generates data from a Gaussian distribution, estimate $P(y = k)$ from the training data, and plug both into Bayes Theorem. This is called *generative* because it models how each class generates data.
- **Discriminative approach (Logistic Regression):** Skip the intermediate step and model the posterior $P(y = k \mid \boldsymbol{X})$ directly using a sigmoid function. This is called *discriminative* because it learns the decision boundary between classes directly, without modelling each class's distribution.

2.2.3 Linear Discriminant Analysis (LDA)

LDA is the generative approach: it explicitly models the class-conditional likelihood $P(\boldsymbol{X} \mid y = k)$ as a Gaussian, estimates the prior $P(y = k)$ from the training data, and plugs both into the Bayes Classifier (eq. (2.17)).

Why LDA Assumes a Gaussian

To use the Bayes Classifier, we need $P(\boldsymbol{X} \mid y = k)$ — the distribution of features within each class. But this distribution is unknown. We need to *assume* a specific form for it, and the Gaussian is the most natural choice:

- Many real-world measurements are approximately Gaussian (by the Central Limit Theorem).
- It is fully characterized by just two parameters: $\boldsymbol{\mu}_k$ (mean) and Σ (covariance), both of which can be estimated from the training data.
- The resulting math simplifies to a *linear* decision boundary, which is where the name "Linear Discriminant Analysis" comes from.

Before stating the LDA assumptions, we recall the multivariate Gaussian distribution that underlies them.

Note 2.15 *(Multivariate Gaussian Distribution).* In one dimension, a Gaussian (normal) distribution $\mathcal{N}(\mu, \sigma^2)$ has the density

$$p(x) = \frac{1}{\sqrt{2\pi}\,\sigma} \exp\left(-\frac{(x-\mu)^2}{2\sigma^2}\right), \tag{2.18}$$

where μ is the mean (center of the bell curve) and σ^2 is the variance (how spread out it is).

The **multivariate Gaussian** $\mathcal{N}(\boldsymbol{\mu}, \Sigma)$ generalizes this to d dimensions. For a feature vector $\boldsymbol{X} \in \mathbb{R}^d$:

$$p(\boldsymbol{X}) = \frac{1}{(2\pi)^{d/2}|\Sigma|^{1/2}} \exp\left(-\frac{1}{2}(\boldsymbol{X}-\boldsymbol{\mu})^\top \Sigma^{-1}(\boldsymbol{X}-\boldsymbol{\mu})\right), \tag{2.19}$$

where:

- $\boldsymbol{\mu} \in \mathbb{R}^d$ is the **mean vector**: the center of the distribution in d-dimensional space.
- $\Sigma \in \mathbb{R}^{d\times d}$ is the **covariance matrix**: a symmetric positive-definite matrix that encodes both the spread (variance) along each axis and the correlations between features. The diagonal entries Σ_{ii} are the variances of each feature, and the off-diagonal entries Σ_{ij} capture how features i and j co-vary.
- $|\Sigma|$ is the **determinant** of the covariance matrix — it scales the normalization so the density integrates to 1.
- Σ^{-1} is the **inverse covariance** (precision) matrix.
- $(\boldsymbol{X}-\boldsymbol{\mu})^\top \Sigma^{-1}(\boldsymbol{X}-\boldsymbol{\mu})$ is the **Mahalanobis distance** — a generalization of the

squared Euclidean distance that accounts for correlations between features. It measures how far $\boldsymbol{X}$ is from the mean $\boldsymbol{\mu}$, scaled by the covariance structure.

In 1D, the covariance matrix reduces to the scalar variance σ^2, and the Mahalanobis distance becomes $(x-\mu)^2/\sigma^2$.

With this background, LDA is built on the following assumptions:

- $P(\boldsymbol{X} \mid y=k) = \mathcal{N}(\boldsymbol{X} \mid \boldsymbol{\mu}_k, \Sigma)$, a multivariate Gaussian for each class.
- All classes share the same covariance matrix $\Sigma_k = \Sigma$.
- $\Pi_k = P(y=k)$, the prior probability of class k.
- $\boldsymbol{\mu}_k \in \mathbb{R}^d$ is the mean vector of class k.

Why Share the Covariance Matrix?

The shared covariance assumption $\Sigma_k = \Sigma$ is what makes the decision boundary *linear*. If each class had its own covariance Σ_k, the quadratic term $-\frac{1}{2}\boldsymbol{X}^\top \Sigma_k^{-1} \boldsymbol{X}$ in the log-likelihood would depend on k and could not be dropped, resulting in a *quadratic* decision boundary. That variant is called Quadratic Discriminant Analysis (QDA).

Deriving the LDA decision rule. Substituting the prior notation $\Pi_k = P(y=k)$ into eq. (2.17):

$$\hat{y}(\boldsymbol{X}) = \underset{k \in \{1,2,\ldots,K\}}{\arg\max}\; P(\boldsymbol{X} \mid y=k)\, \Pi_k. \tag{2.20}$$

Since log is monotonic, the arg max is unchanged if we take the log (this turns products into sums, which are easier to work with):

$$\hat{y}(\boldsymbol{X}) = \underset{k \in \{1,2,\ldots,K\}}{\arg\max}\; \left(\log P(\boldsymbol{X} \mid y=k) + \log \Pi_k\right). \tag{2.21}$$

Now we plug in the multivariate Gaussian assumption. Since $P(\boldsymbol{X} \mid y=k) = \mathcal{N}(\boldsymbol{X} \mid \boldsymbol{\mu}_k, \Sigma)$:

$$P(\boldsymbol{X} \mid y=k) = \frac{1}{(2\pi)^{d/2}|\Sigma|^{1/2}} \exp\left(-\frac{1}{2}(\boldsymbol{X}-\boldsymbol{\mu}_k)^\top \Sigma^{-1} (\boldsymbol{X}-\boldsymbol{\mu}_k)\right). \tag{2.22}$$

Taking the log and expanding the quadratic form:

$$\log P(\boldsymbol{X} \mid y=k) = \underbrace{-\frac{1}{2}\boldsymbol{X}^\top \Sigma^{-1} \boldsymbol{X}}_{\text{same for all } k} + \boldsymbol{X}^\top \Sigma^{-1} \boldsymbol{\mu}_k - \frac{1}{2}\boldsymbol{\mu}_k^\top \Sigma^{-1} \boldsymbol{\mu}_k \underbrace{-\frac{1}{2}\log|\Sigma| - \frac{d}{2}\log(2\pi)}_{\text{same for all } k}. \tag{2.23}$$

The terms marked "same for all k" do not affect which class wins the arg max, so we drop them. The first term $-\frac{1}{2}\boldsymbol{X}^\top \Sigma^{-1} \boldsymbol{X}$ drops out precisely *because* Σ is shared across all classes —

this is where the shared covariance assumption pays off. What remains is the **discriminant function**:

$$\delta_k(\boldsymbol{X}) = \boldsymbol{X}^\top \Sigma^{-1} \boldsymbol{\mu}_k - \frac{1}{2} \boldsymbol{\mu}_k^\top \Sigma^{-1} \boldsymbol{\mu}_k + \log \Pi_k. \tag{2.24}$$

Note that $\delta_k(\boldsymbol{X})$ is *linear* in $\boldsymbol{X}$: there are no $\boldsymbol{X}^\top(\cdots)\boldsymbol{X}$ terms left.

Definition 2.16 *(LDA).* The Linear Discriminant Analysis classifier is

$$\hat{y}(\boldsymbol{X}) = \underset{k \in \{1,2,\ldots,K\}}{\arg\max}\ \delta_k(\boldsymbol{X}), \tag{2.25}$$

where the discriminant function for class k is

$$\delta_k(\boldsymbol{X}) = \boldsymbol{X}^\top \Sigma^{-1} \boldsymbol{\mu}_k - \frac{1}{2} \boldsymbol{\mu}_k^\top \Sigma^{-1} \boldsymbol{\mu}_k + \log \Pi_k. \tag{2.26}$$

Special case: two classes ($K = 2$). For binary classification, we assign class 1 when $\delta_1(\boldsymbol{X}) - \delta_2(\boldsymbol{X}) \geq 0$. Substituting the definition of δ_k and simplifying, this reduces to

$$\boldsymbol{W}^\top \boldsymbol{X} + b \geq 0, \tag{2.27}$$

where

$$\boldsymbol{W} = \Sigma^{-1}(\boldsymbol{\mu}_1 - \boldsymbol{\mu}_2), \tag{2.28}$$

$$b = -\frac{1}{2}\left(\boldsymbol{\mu}_1^\top \Sigma^{-1} \boldsymbol{\mu}_1 - \boldsymbol{\mu}_2^\top \Sigma^{-1} \boldsymbol{\mu}_2\right) + \log \frac{\Pi_1}{\Pi_2}. \tag{2.29}$$

Note 2.17 *(Geometric Meaning of $\boldsymbol{W}^\top \boldsymbol{X} + b \geq 0$).* In the binary case, classification reduces to checking which side of a *hyperplane* $\boldsymbol{X}$ falls on:

- $\boldsymbol{W} = \Sigma^{-1}(\boldsymbol{\mu}_1 - \boldsymbol{\mu}_2)$ is the normal vector to the decision boundary: it points in the direction that best separates the two class means, adjusted for the covariance structure.
- b is the bias (offset), which shifts the boundary based on the priors Π_1, Π_2 and the class means.
- If $\boldsymbol{W}^\top \boldsymbol{X} + b \geq 0$, classify as class 1; otherwise, classify as class 2.

The decision boundary of LDA is a flat hyperplane in $\mathbb{R}^d$, not a curve.

2.2.4 Logistic Regression

Logistic Regression is the **discriminative** counterpart to LDA: instead of modelling the class-conditional density $P(\boldsymbol{X} \mid y = k)$ and applying Bayes Theorem (as LDA does), it models the posterior $P(y = k \mid \boldsymbol{X})$ *directly* using a parametric function. This bypasses the need to assume any distribution for the features within each class.

Problem setup. We consider binary classification: features $\boldsymbol{X} \in \mathbb{R}^d$ and label $y \in \{0, 1\}$. The goal is to model the probability that an observation belongs to class 1, given its features:

$$P(y = 1 \mid \boldsymbol{X}). \tag{2.30}$$

Why Not Just Use Linear Regression?

A natural first idea is to use a linear model $f(\boldsymbol{X}) = \boldsymbol{\beta}^\top \boldsymbol{X}$ and interpret the output as a probability. The problem is that $\boldsymbol{\beta}^\top \boldsymbol{X}$ can produce *any* real number: it might output -3.7 or 45, which are not valid probabilities. We need a function that takes any real number and squashes it into the interval (0, 1). That function is the **sigmoid**.

Note 2.18 *(What Is $\boldsymbol{\beta}$?).* The parameter vector $\boldsymbol{\beta} \in \mathbb{R}^d$ is what the model learns from data. Each component has a concrete role:

$$\boldsymbol{\beta} = \begin{pmatrix} \beta_0 \\ \beta_1 \\ \vdots \\ \beta_{d-1} \end{pmatrix}, \tag{2.31}$$

where β_j is the **weight** for feature x_j. A large positive β_j means that feature pushes the prediction toward class 1; a large negative β_j pushes toward class 0. The first component β_0 is the **intercept** (bias), which shifts the decision boundary independently of any feature. This is why we prepend a 1 to $\boldsymbol{X}$ in eq. (2.33).
In LDA, the equivalent role was played by $\boldsymbol{W} = \Sigma^{-1}(\boldsymbol{\mu}_1 - \boldsymbol{\mu}_2)$ and b, which were computed from sample statistics. In logistic regression, $\boldsymbol{\beta}$ is found by maximizing the likelihood (covered below).

From linear model to probabilities. Recall that in linear regression, we model the output as a linear combination of features:

$$\boldsymbol{\beta}^\top \boldsymbol{X} = \beta_0 + \beta_1 x_1 + \cdots + \beta_{d-1} x_d, \quad \boldsymbol{\beta} \in \mathbb{R}^d, \tag{2.32}$$

where the feature vector includes a leading 1 for the intercept:

$$\boldsymbol{X} = \begin{bmatrix} 1 \\ x_1 \\ \vdots \\ x_d \end{bmatrix}. \tag{2.33}$$

The output $\boldsymbol{\beta}^\top \boldsymbol{X}$ is called the **log-odds** or **logit** (we will see why shortly). To convert it into a probability, we pass it through the **sigmoid function**:

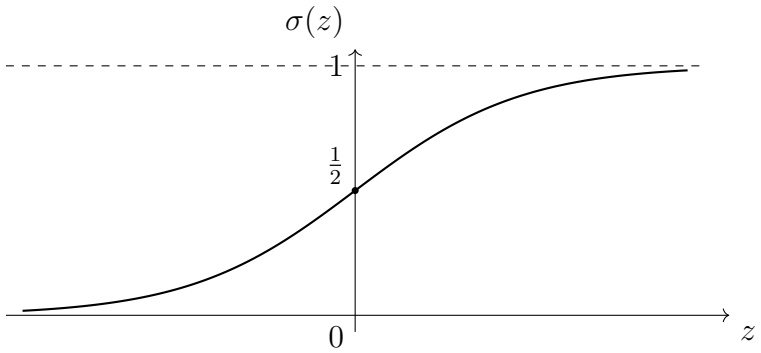
$$\boldsymbol{\beta}^\top \boldsymbol{X} \longrightarrow \sigma\big(\boldsymbol{\beta}^\top \boldsymbol{X}\big) = \frac{1}{1 + e^{-\boldsymbol{\beta}^\top \boldsymbol{X}}}. \tag{2.34}$$

Note 2.19 *(The Sigmoid Function $\sigma(z)$).* The sigmoid (also called the *logistic* function) is defined as

$$\sigma(z) = \frac{1}{1 + e^{-z}}. \tag{2.35}$$

Key properties:

- $\sigma(z) \in (0, 1)$ for all $z \in \mathbb{R}$, so the output is always a valid probability.
- $\sigma(0) = \frac{1}{2}$: when the linear model outputs exactly zero, the model is equally uncertain about both classes.
- $\sigma(z) \to 1$ as $z \to +\infty$ and $\sigma(z) \to 0$ as $z \to -\infty$: large positive scores give high confidence for class 1, large negative scores give high confidence for class 0.
- Symmetry: $\sigma(-z) = 1 - \sigma(z)$.
- The derivative has a clean form: $\sigma'(z) = \sigma(z)\big(1 - \sigma(z)\big)$, which is convenient for gradient computations.



The logistic regression model. Applying the sigmoid to the linear model gives the logistic regression model:

$$P(y = 1 \mid \boldsymbol{X}) = \sigma(\boldsymbol{\beta}^\top \boldsymbol{X}) = \frac{1}{1 + e^{-\boldsymbol{\beta}^\top \boldsymbol{X}}}, \tag{2.36}$$

$$P(y = 0 \mid \boldsymbol{X}) = 1 - \sigma(\boldsymbol{\beta}^\top \boldsymbol{X}). \tag{2.37}$$

Why "Logistic" Regression? The Log-Odds Connection

Take the ratio of the two class probabilities:

$$\log \frac{P(y = 1 \mid \boldsymbol{X})}{P(y = 0 \mid \boldsymbol{X})} = \log \frac{\sigma(\boldsymbol{\beta}^\top \boldsymbol{X})}{1 - \sigma(\boldsymbol{\beta}^\top \boldsymbol{X})} = \boldsymbol{\beta}^\top \boldsymbol{X}. \tag{2.38}$$

This quantity is called the **log-odds** (or **logit**). The logistic regression model is saying: the log-odds of belonging to class 1 is a *linear* function of the features. The sigmoid is simply the inverse of the logit function, which is why it maps the linear output back to a probability. This also explains the name "logistic" regression: it is a regression on the logit (log-odds).

Estimating $\boldsymbol{\beta}$: the likelihood function. Unlike LDA, where we estimated parameters ($\boldsymbol{\mu}_k$, Σ, Π_k) by computing sample statistics, logistic regression has no such shortcut. We need to find $\boldsymbol{\beta}$ using **maximum likelihood estimation** (MLE): choose the $\boldsymbol{\beta}$ that makes the observed data most probable.

Given the training set $\{(\boldsymbol{X}_i, y_i)\}_{i=1}^N$ with $y_i \in \{0, 1\}$, the probability of observing one data point is

$$P(y_i \mid \boldsymbol{X}_i, \boldsymbol{\beta}) = \left[\sigma(\boldsymbol{\beta}^\top \boldsymbol{X}_i)\right]^{y_i} \left[1 - \sigma(\boldsymbol{\beta}^\top \boldsymbol{X}_i)\right]^{1-y_i}. \tag{2.39}$$

Note 2.20 *(Why Does Eq.* (2.39) *Work?).* This is a compact way of writing "pick the right probability depending on the label":

- If $y_i = 1$: the expression becomes $\big[\sigma(\boldsymbol{\beta}^\top \boldsymbol{X}_i)\big]^1 \cdot \big[1 - \sigma(\boldsymbol{\beta}^\top \boldsymbol{X}_i)\big]^0 = \sigma(\boldsymbol{\beta}^\top \boldsymbol{X}_i) = P(y = 1 \mid \boldsymbol{X}_i)$. ✓
- If $y_i = 0$: the expression becomes $\big[\sigma(\boldsymbol{\beta}^\top \boldsymbol{X}_i)\big]^0 \cdot \big[1 - \sigma(\boldsymbol{\beta}^\top \boldsymbol{X}_i)\big]^1 = 1 - \sigma(\boldsymbol{\beta}^\top \boldsymbol{X}_i) = P(y = 0 \mid \boldsymbol{X}_i)$. ✓

The exponents y_i and $1 - y_i$ act as a "switch" that selects the correct class probability. This trick works because $y_i \in \{0, 1\}$.

Assuming all data points are independent, the likelihood of the entire dataset is the product over all N points:

$$L(\boldsymbol{\beta}) = \prod_{i=1}^{N} P(y_i \mid \boldsymbol{X}_i, \boldsymbol{\beta}) = \prod_{i=1}^{N} \big[\sigma(\boldsymbol{\beta}^\top \boldsymbol{X}_i)\big]^{y_i} \big[1 - \sigma(\boldsymbol{\beta}^\top \boldsymbol{X}_i)\big]^{1-y_i}. \tag{2.40}$$

Note 2.21 *(Why a Product?).* If the data points are independent (which we assume), then the probability of observing *all* of them is the product of the individual probabilities, just like flipping N independent coins. The likelihood $L(\boldsymbol{\beta})$ answers: "given this choice of $\boldsymbol{\beta}$, how probable is the dataset I actually observed?"

The log-likelihood. Products of many small numbers are numerically unstable and hard to differentiate. Taking the logarithm converts the product into a sum (since $\log(ab) = \log a + \log b$) without changing which $\boldsymbol{\beta}$ maximizes it (since log is monotonically increasing):

$$\ell(\boldsymbol{\beta}) = \log L(\boldsymbol{\beta}) = \sum_{i=1}^{N} \Big[y_i \log \sigma(\boldsymbol{\beta}^\top \boldsymbol{X}_i) + (1 - y_i) \log\big(1 - \sigma(\boldsymbol{\beta}^\top \boldsymbol{X}_i)\big) \Big]. \tag{2.41}$$

The goal is to find the $\boldsymbol{\beta}$ that maximizes $\ell(\boldsymbol{\beta})$.

Note 2.22 *(Why Is There No Closed-Form Solution?).* In LDA, we could compute $\boldsymbol{\mu}_k$ and Σ directly from sample means and covariances. Here, the sigmoid function inside the log makes eq. (2.41) a nonlinear function of $\boldsymbol{\beta}$. Setting $\nabla_{\boldsymbol{\beta}}\, \ell(\boldsymbol{\beta}) = \mathbf{0}$ does not yield a nice algebraic solution. Instead, we use **gradient descent** (covered in the next section) to iteratively search for the optimal $\boldsymbol{\beta}$.

Logistic Regression vs. LDA: Two Roads to the Same Goal

Both logistic regression and LDA aim to classify $\boldsymbol{X}$ into one of K classes, but they take opposite approaches:

- **LDA (generative):** models *how each class generates data*, $P(\boldsymbol{X} \mid y = k)$, using a Gaussian assumption. Then it applies Bayes Theorem to get $P(y = k \mid \boldsymbol{X})$. Parameters are estimated by computing sample statistics (means, covariance, class proportions). Has a closed-form solution.
- **Logistic regression (discriminative):** it models $P(y = k \mid \boldsymbol{X})$ *directly* via the sigmoid. Makes no assumption about what the features look like within each class. Parameters are estimated by maximizing the likelihood numerically (gradient descent). No closed-form solution.

Interestingly, both produce a *linear* decision boundary $\boldsymbol{W}^\top \boldsymbol{X} + b = 0$. The difference is in how $\boldsymbol{W}$ and b are obtained. LDA derives them from the Gaussian parameters; logistic regression learns them directly from the data.

Lecture 3

3.1 Review and Summary of Lecture 2

3.1.1 Bayes Decision Rule

Let $y \in \{0, 1\}$ denote the class label and $\boldsymbol{X}$ the feature vector. The Bayes classifier assigns the input to the class with the highest posterior probability:

$$\hat{y}(\boldsymbol{X}) = \arg\max_k \, P(y = k \mid \boldsymbol{X}). \tag{3.1}$$

Using Bayes Theorem, this can be rewritten as

$$\hat{y}(\boldsymbol{X}) = \arg\max_k \, P(\boldsymbol{X} \mid y = k)\, P(y = k), \tag{3.2}$$

where $P(\boldsymbol{X} \mid y = k)$ is the class-conditional likelihood and $P(y = k) = \Pi_k$ is the prior probability of class k.

3.1.2 Linear Discriminant Analysis (LDA)

LDA is the **generative** approach. It assumes that the class-conditional density

$$P(\boldsymbol{X} \mid y = k) = \mathcal{N}(\boldsymbol{X} \mid \boldsymbol{\mu}_k, \, \Sigma) \tag{3.3}$$

is multivariate Gaussian for each class k, with all classes sharing a common covariance matrix Σ. This leads to a linear discriminant function $\delta_k(\boldsymbol{X})$ and a linear decision boundary.

3.1.3 Logistic Regression

Logistic regression is the **discriminative** approach. It directly models the posterior probability:

$$P(y = 1 \mid \boldsymbol{X}) = \sigma(\boldsymbol{\beta}^\top \boldsymbol{X}) = \frac{1}{1 + e^{-\boldsymbol{\beta}^\top \boldsymbol{X}}}, \tag{3.4}$$

for binary classification with $y \in \{0, 1\}$ and labeled data $\{(\boldsymbol{X}_i, y_i)\}_{i=1}^N$.

3.1.4 Log-Likelihood and Loss Function

The likelihood of the training data under the logistic model is

$$L(\boldsymbol{\beta}) = \prod_{i=1}^{N} \big[\sigma(\boldsymbol{\beta}^\top \boldsymbol{X}_i)\big]^{y_i} \big[1 - \sigma(\boldsymbol{\beta}^\top \boldsymbol{X}_i)\big]^{1-y_i}. \tag{3.5}$$

Taking the logarithm gives the log-likelihood:

$$\ell(\boldsymbol{\beta}) = \log L(\boldsymbol{\beta}) = \sum_{i=1}^{N} \Big[y_i \log \sigma(\boldsymbol{\beta}^\top \boldsymbol{X}_i) + (1 - y_i) \log\big(1 - \sigma(\boldsymbol{\beta}^\top \boldsymbol{X}_i)\big) \Big]. \tag{3.6}$$

Definition 3.1 *(Binary Cross-Entropy Loss).* Maximizing the log-likelihood is equivalent to minimizing the negative log-likelihood. We define the **binary cross-entropy loss** (cost function) as

$$J(\boldsymbol{\beta}) = -\ell(\boldsymbol{\beta}) = -\sum_{i=1}^{N} \Big[y_i \log \sigma(\boldsymbol{\beta}^\top \boldsymbol{X}_i) + (1 - y_i) \log\big(1 - \sigma(\boldsymbol{\beta}^\top \boldsymbol{X}_i)\big) \Big]. \tag{3.7}$$

Our optimization problem is

$$\min_{\boldsymbol{\beta}} \; J(\boldsymbol{\beta}). \tag{3.8}$$

Note 3.2 *(Why Minimize the Negative Log-Likelihood?).* Since $J(\boldsymbol{\beta})$ is nonlinear in $\boldsymbol{\beta}$, there is no closed-form solution. We must use **iterative optimization methods**. For this course, we use gradient descent.

3.2 Gradient Descent

Definition 3.3 *(Gradient Descent).* Gradient descent is an iterative optimization algorithm for finding a local minimum of a differentiable function $f : \mathbb{R}^d \to \mathbb{R}$. Starting from an initial guess $\boldsymbol{\beta}^{(0)}$, we repeatedly update:

$$\boldsymbol{\beta}^{(k+1)} = \boldsymbol{\beta}^{(k)} - \eta \, \nabla f\big(\boldsymbol{\beta}^{(k)}\big), \tag{3.9}$$

where:

- $\boldsymbol{\beta}^{(k)}$ is the parameter vector at step k.
- $\eta > 0$ is the **learning rate** (step size), which can be fixed or adaptive.
- $\nabla f\big(\boldsymbol{\beta}^{(k)}\big)$ is the gradient of the loss function at the current step.

We stop when $\nabla f(\boldsymbol{\beta}^{(k)}) \approx \mathbf{0}$ (i.e., we have reached a minimum).

Note 3.4 *(Why the Negative Gradient?).* $\nabla f(\boldsymbol{\beta})$ points in the direction of steepest **increase**. Negating it gives the direction of steepest **decrease**, so each update in eq. (3.9) moves $\boldsymbol{\beta}$ downhill.

Note 3.5 *(Why Does This Guarantee Decrease?).* By the first-order Taylor expansion,

$$f\big(\boldsymbol{\beta}^{(k)} + \Delta\boldsymbol{\beta}\big) \approx f\big(\boldsymbol{\beta}^{(k)}\big) + \nabla f\big(\boldsymbol{\beta}^{(k)}\big)^\top \Delta\boldsymbol{\beta}. \tag{3.10}$$

Choosing $\Delta\boldsymbol{\beta} = -\eta\, \nabla f\big(\boldsymbol{\beta}^{(k)}\big)$ gives

$$\nabla f\big(\boldsymbol{\beta}^{(k)}\big)^\top \Delta\boldsymbol{\beta} = -\eta \left\|\nabla f\big(\boldsymbol{\beta}^{(k)}\big)\right\|^2 < 0, \tag{3.11}$$

so f decreases at each step (provided η is small enough).

3.2.1 Deriving the Gradient for Logistic Regression

Applying eq. (3.9) to logistic regression requires computing the gradient of $J(\boldsymbol{\beta})$ with respect to $\boldsymbol{\beta}$. We differentiate the binary cross-entropy loss term by term. We need

$$\nabla_{\boldsymbol{\beta}} J(\boldsymbol{\beta}) = -\sum_{i=1}^{N} \nabla_{\boldsymbol{\beta}} \Big[y_i \log \sigma(\boldsymbol{\beta}^\top \boldsymbol{X}_i) + (1 - y_i) \log\big(1 - \sigma(\boldsymbol{\beta}^\top \boldsymbol{X}_i)\big) \Big]. \tag{3.12}$$

Let $z_i = \boldsymbol{\beta}^\top \boldsymbol{X}_i$. We use the following properties of the sigmoid derivative:

$$\frac{d}{dz} \log \sigma(z) = 1 - \sigma(z), \tag{3.13}$$

$$\frac{d}{dz} \log\big(1 - \sigma(z)\big) = -\sigma(z), \tag{3.14}$$

$$\nabla_{\boldsymbol{\beta}}\, z_i = \boldsymbol{X}_i. \tag{3.15}$$

Applying the chain rule to each term:

$$\nabla_{\boldsymbol{\beta}} \big[y_i \log \sigma(z_i) \big] = y_i \big(1 - \sigma(z_i)\big) \boldsymbol{X}_i, \tag{3.16}$$

$$\nabla_{\boldsymbol{\beta}} \big[(1 - y_i) \log\big(1 - \sigma(z_i)\big) \big] = -(1 - y_i)\, \sigma(z_i)\, \boldsymbol{X}_i. \tag{3.17}$$

Summing these up and simplifying:

$$\nabla_{\boldsymbol{\beta}} J(\boldsymbol{\beta}) = \sum_{i=1}^{N} \big(\sigma(\boldsymbol{\beta}^\top \boldsymbol{X}_i) - y_i\big)\, \boldsymbol{X}_i. \tag{3.18}$$

Note 3.6 *(Intuition for the Gradient).* The gradient in eq. (3.18) has a clean interpretation: the term $\sigma(\boldsymbol{\beta}^\top \boldsymbol{X}_i) - y_i$ is the **prediction error** for data point i (predicted

probability minus actual label). Each data point contributes to the gradient in proportion to its error, weighted by its feature vector $\boldsymbol{X}_i$. When the prediction is perfect ($\sigma(\boldsymbol{\beta}^\top \boldsymbol{X}_i) = y_i$), that data point contributes zero to the gradient.

Corollary 3.7 *(Gradient Descent Update Rule: Logistic Regression).* The gradient descent update rule for logistic regression is:

$$\boldsymbol{\beta}^{(k+1)} = \boldsymbol{\beta}^{(k)} - \eta \sum_{i=1}^{N} \left(\sigma(\boldsymbol{\beta}^{(k)\top} \boldsymbol{X}_i) - y_i\right) \boldsymbol{X}_i. \tag{3.19}$$

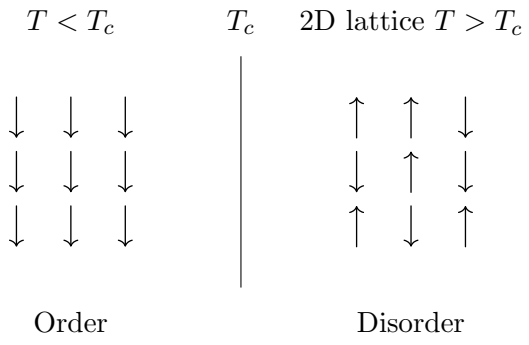
3.3 Example: Ising Model

To illustrate classification in a physics context, consider the Ising model phase transition. Given a spin configuration, can we classify whether the system is above or below the critical temperature?

The Hamiltonian of the system is

$$H = -J \sum_{\langle ij \rangle} \sigma_i \sigma_j, \tag{3.20}$$

where $\langle ij \rangle$ denotes nearest-neighbor pairs (2D lattice) and $\sigma_i \in \{-1, +1\}$. For $J = 1$ in the 2D square-lattice Ising model: $T_c \approx 2.269$.



In this setting, the feature vector $\boldsymbol{X}$ could consist of statistics computed from the spin configuration, such as the magnetization, energy, or spatial correlation functions. The label $y \in \{0, 1\}$ indicates whether the configuration was sampled from the ordered phase ($T < T_c$) or the disordered phase ($T > T_c$). Logistic regression can then be trained on labeled configurations to predict the phase from the features.

3.4 Example: Prediction of Immunotherapy Response

Consider the problem of predicting patient response to immunotherapy with pre-treatment clinical data. Can we predict, prior to treatment, the probability that a patient will respond to immunotherapy using measured clinical variables? We are given labeled clinical data consisting of six measured features per patient. Let

$$\boldsymbol{X} \in \mathbb{R}^6 \tag{3.21}$$

denote the feature vector for a single patient, and let

$$\{(\boldsymbol{X}_i, y_i)\}_{i=1}^n \tag{3.22}$$

be the full dataset, where n is the number of data points and

$$y_i \in \{0, 1\} \tag{3.23}$$

indicates whether patient i responds to immunotherapy. The goal is to model the probability of response given the clinical features.

We use a logistic regression model to estimate the probability of response:

$$\mathbb{P}(y = 1 \mid \boldsymbol{X}) = \sigma\left(\boldsymbol{\beta}^T \boldsymbol{X}\right) = \sigma\left(\beta_0 + \sum_{j=1}^{6} \beta_j X_j\right), \tag{3.24}$$

where β_0 is the bias (intercept) term. The quantity

$$\mathbb{P}(y = 1 \mid \boldsymbol{X}) \tag{3.25}$$

represents the predicted probability that a patient responds to immunotherapy given their pre-treatment clinical variables.

3.4.1 Log-Odds of Response Approach

An equivalent and often more interpretable form of logistic regression is obtained by considering the *odds* and *log-odds* of response.

Definition 3.8 *(Odds).* The **odds** of an event is the ratio of the probability it occurs

to the probability it does not:

$$\text{odds} = \frac{\mathbb{P}(\text{event})}{\mathbb{P}(\text{not event})} = \frac{\mathbb{P}(\text{event})}{1 - \mathbb{P}(\text{event})}. \tag{3.26}$$

For example, if $\mathbb{P}(y = 1 \mid \boldsymbol{X}) = 0.75$, then the odds of response are $\frac{0.75}{0.25} = 3$, meaning response is 3 times more likely than non-response.

The odds of response given the clinical features $\boldsymbol{X}$ are

$$\frac{\mathbb{P}(y = 1 \mid \boldsymbol{X})}{\mathbb{P}(y = 0 \mid \boldsymbol{X})} = \frac{\mathbb{P}(y = 1 \mid \boldsymbol{X})}{1 - \mathbb{P}(y = 1 \mid \boldsymbol{X})}. \tag{3.27}$$

Using the logistic regression model

$$\mathbb{P}(y = 1 \mid \boldsymbol{X}) = \sigma\left(\beta_0 + \sum_{j=1}^{6} \beta_j X_j\right), \tag{3.28}$$

write $z = \beta_0 + \sum_{j=1}^{6} \beta_j X_j$ for shorthand. Since $\sigma(z) = \frac{1}{1+e^{-z}}$:

$$1 - \sigma(z) = 1 - \frac{1}{1 + e^{-z}} = \frac{e^{-z}}{1 + e^{-z}}. \tag{3.29}$$

The odds become

$$\frac{\sigma(z)}{1 - \sigma(z)} = \frac{1/(1 + e^{-z})}{e^{-z}/(1 + e^{-z})} = \frac{1}{e^{-z}} = e^{z}. \tag{3.30}$$

Taking the logarithm of both sides gives the **log-odds**:

$$\log \frac{\mathbb{P}(y = 1 \mid \boldsymbol{X})}{\mathbb{P}(y = 0 \mid \boldsymbol{X})} = \log\, e^{z} = z \tag{3.31}$$

$$= \beta_0 + \sum_{j=1}^{6} \beta_j X_j. \tag{3.32}$$

Note 3.9 *(Why Rewrite the Model This Way?).* The sigmoid form and the log-odds form are the same model. The sigmoid form is what we optimize during training (it plugs into the likelihood). The log-odds form is for interpretation after training: each β_j is the additive change in log-odds per unit increase in feature X_j, regardless of the current probability. Equivalently, increasing X_j by one unit multiplies the odds by e^{β_j}.

Appendix

A Background: Linear Regression and Logistic Regression

This appendix provides background on linear regression and how logistic regression extends it. These ideas underpin the discriminative classification approach in §2.2.4.

A.1 Linear Regression

Linear regression is the simplest predictive model. Given features $\boldsymbol{X} \in \mathbb{R}^d$ and a **continuous** output $y \in \mathbb{R}$ (not a class label, but an actual number like temperature or price), we assume the relationship is linear:

$$y = \boldsymbol{\beta}^\top \boldsymbol{X} = \beta_0 + \beta_1 x_1 + \beta_2 x_2 + \cdots + \beta_{d-1} x_d. \tag{A.1}$$

Geometrically, this fits a straight line (in 1D), a plane (in 2D), or a hyperplane (in higher dimensions) through the data. The parameter vector $\boldsymbol{\beta}$ is chosen to minimize the sum of squared errors:

$$\min_{\boldsymbol{\beta}} \sum_{i=1}^{N} \big(y_i - \boldsymbol{\beta}^\top \boldsymbol{X}_i\big)^2. \tag{A.2}$$

Unlike logistic regression, eq. (A.2) has a **closed-form solution**: setting the gradient to zero and solving gives

$$\hat{\boldsymbol{\beta}} = (\boldsymbol{X}^\top \boldsymbol{X})^{-1} \boldsymbol{X}^\top \boldsymbol{y}, \tag{A.3}$$

where $\boldsymbol{X}$ is the $N \times d$ data matrix and $\boldsymbol{y}$ is the vector of outputs. This is known as the **normal equation**.

A.2 Why Linear Regression Fails for Classification

If we try to use linear regression for binary classification ($y \in \{0, 1\}$), the model predicts

$$\hat{y} = \boldsymbol{\beta}^\top \boldsymbol{X}, \tag{A.4}$$

which can output *any* real number. There is no guarantee the output falls in [0, 1], so it cannot be interpreted as a probability. For example, the model might predict $\hat{y} = -3.7$ or $\hat{y} = 45$ for a given input, neither of which is a valid probability.

A.3 From Linear Regression to Logistic Regression

Logistic regression solves this by keeping the linear structure but wrapping it in the sigmoid function:

$$\underbrace{\boldsymbol{\beta}^\top \boldsymbol{X}}_{\text{any real number}} \xrightarrow{\sigma} \underbrace{\sigma(\boldsymbol{\beta}^\top \boldsymbol{X})}_{\in (0,1)}. \tag{A.5}$$

The key insight is that this is not an arbitrary choice. Logistic regression assumes that the **log-odds** of class 1 versus class 0 is linear in the features:

$$\log \frac{P(y=1 \mid \boldsymbol{X})}{P(y=0 \mid \boldsymbol{X})} = \boldsymbol{\beta}^\top \boldsymbol{X}. \tag{A.6}$$

Solving eq. (A.6) for $P(y=1 \mid \boldsymbol{X})$, using the fact that $P(y=0 \mid \boldsymbol{X}) = 1 - P(y=1 \mid \boldsymbol{X})$, yields the sigmoid as the *only* function that satisfies this constraint:

$$P(y=1 \mid \boldsymbol{X}) = \frac{1}{1+e^{-\boldsymbol{\beta}^\top \boldsymbol{X}}} = \sigma(\boldsymbol{\beta}^\top \boldsymbol{X}). \tag{A.7}$$

So the sigmoid is not chosen because "it looks nice." It is the mathematical consequence of assuming linear log-odds.

A.4 Beyond Binary: Multinomial Logistic Regression

The logistic regression model in §2.2.4 handles binary classification ($y \in \{0,1\}$), but the idea extends naturally to $K > 2$ classes. Instead of a single parameter vector $\boldsymbol{\beta}$, each class k gets its own parameter vector $\boldsymbol{\beta}_k$. The sigmoid is replaced by the **softmax** function, which produces a probability for every class simultaneously:

$$P(y=k \mid \boldsymbol{X}) = \frac{e^{\boldsymbol{\beta}_k^\top \boldsymbol{X}}}{\sum_{j=1}^{K} e^{\boldsymbol{\beta}_j^\top \boldsymbol{X}}}. \tag{A.8}$$

The denominator ensures that the probabilities over all K classes sum to 1. This is called **multinomial logistic regression** (or **softmax regression**).

Note A.1 *(Sigmoid Is a Special Case of Softmax).* When $K = 2$, the softmax reduces to the sigmoid. Setting $\boldsymbol{\beta}_1 = \boldsymbol{\beta}$ and $\boldsymbol{\beta}_0 = \mathbf{0}$ in eq. (A.8) gives

$$P(y=1 \mid \boldsymbol{X}) = \frac{e^{\boldsymbol{\beta}^\top \boldsymbol{X}}}{e^{\mathbf{0}^\top \boldsymbol{X}} + e^{\boldsymbol{\beta}^\top \boldsymbol{X}}} = \frac{e^{\boldsymbol{\beta}^\top \boldsymbol{X}}}{1+e^{\boldsymbol{\beta}^\top \boldsymbol{X}}} = \frac{1}{1+e^{-\boldsymbol{\beta}^\top \boldsymbol{X}}} = \sigma(\boldsymbol{\beta}^\top \boldsymbol{X}). \tag{A.9}$$

So the binary model is not a separate method; it is the two-class simplification of the general framework.

If you have seen neural networks, this should look familiar: the softmax is exactly what sits at the output layer of a classification network. Just as a single neuron computes $\sigma(\boldsymbol{\beta}^\top \boldsymbol{X})$ (binary logistic regression), the final softmax layer performs multinomial logistic regression over K classes.

A.5 Connection to LDA

Interestingly, if the LDA assumptions hold (features are Gaussian with shared covariance), then $P(y = 1 \mid \boldsymbol{X})$ turns out to be *exactly* a sigmoid of a linear function. In other words, LDA implies the logistic regression model. Logistic regression simply adopts the same functional form without requiring the Gaussian assumption, which makes it more flexible but requires numerical optimization (gradient descent) instead of a closed-form solution.

Note A.2 *(Summary: Linear Regression vs. Logistic Regression).* The differences are

- **Linear regression:** predicts a continuous value $y \in \mathbb{R}$. Minimizes squared error. Has a closed-form solution (normal equation). Cannot be used directly for classification because outputs are unbounded.
- **Logistic regression:** predicts a probability $P(y = 1 \mid \boldsymbol{X}) \in (0, 1)$. Maximizes the likelihood. Requires gradient descent (no closed-form solution). The sigmoid maps the linear output to a valid probability.
- Both use a linear combination $\boldsymbol{\beta}^\top \boldsymbol{X}$ at their core. Logistic regression wraps it in a sigmoid; linear regression uses it directly.

A.6 Historical Context

Logistic regression was developed in the 1940s–1950s in biostatistics, where the core problem was: given measurements about a patient, what is the *probability* they have a disease? Doctors wanted a calibrated probability to make informed decisions.

The model remains widely used today for two reasons beyond classification:

- **Interpretability:** each weight β_j has a direct meaning. For instance, $\beta_{\text{smoking}} = 1.2$ means smoking multiplies the odds of the outcome by $e^{1.2} \approx 3.3$.
- **Building block for neural networks:** a single neuron in a neural network computes exactly $\sigma(\boldsymbol{\beta}^\top \boldsymbol{X})$. A neural network is, in essence, many logistic regressions composed together with nonlinearities between layers.